目录

Introduction 1.1

Regressiom 1.2

- Regression_P1 1.2.1
- Regression_P2 1.2.2

General Guidance 1.3

Local Minimum And Saddle Point 1.4

Batch and Momentum 1.5

Adaptive Learning Rate 1.6

Batch Normalization 1.7

Classification 1.8

CNN 1.9

Self-Attention 1.10

- Self Attention P1 1.10.1
- Self Attention P2 1.10.2

Transformer 1.11

- Transformer P1 1.11.1
- Transformer P2 1.11.2

GAN 1.12

- GAN P1 1.12.1
- GAN P2 1.12.2
- GAN P3 1.12.3
- GAN P4 1.12.4

BERT 1.13

- BERT P1 1.13.1
- BERT P2 1.13.2
- BERT P3 1.13.3

Auto-Encoder 1.14

- Auto Encoder P1 1.14.1
- Auto Encoder P2 1.14.2

Adversarial-Attack 1.15

- Adversarial Attack P1 1.15.1
- Adversarial Attack P2 1.15.2

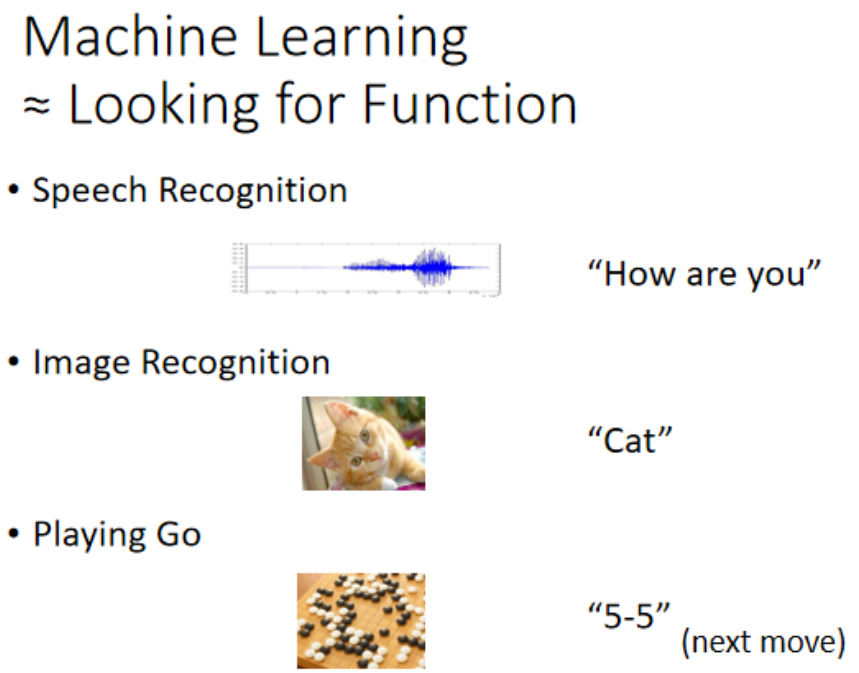


Regression

Machine Learning

第一堂课 是要简单跟大家介绍一下machine learning还有deep learning的基本概念，等一下会讲一个跟宝可梦完全没有关係的故事。想必大家在报章杂志上其实往往都已经听过机器学习这一个词汇，那你可能也知道说机器学习就是跟今天很热门的AI好像有那么一点关联。

那所谓的机器学习到底是什么呢？顾名思义，机器他具备有学习的能力，那些科普文章往往把机器学习这个东西吹得玄之又玄好像机器会学习以后，我们就有了人工智慧，有了人工智慧以后机器接下来就要统治人类了。那机器学习到底是什么呢？事实上，机器学习概括来说可以用一句话来描述机器学习这件事，**==机器学习就是让机器具备找一个函式的能力==**。



那机器具备找函式的能力以后,他确实可以做很多事 举例来说

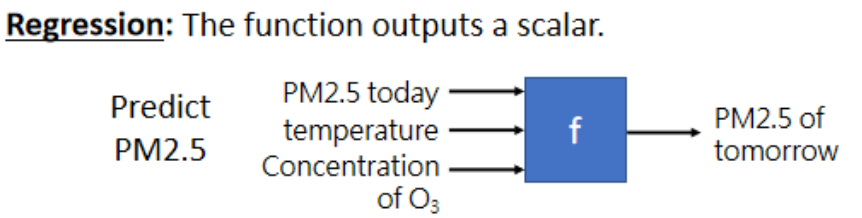
1. 假设你今天想要叫机器做语音辨识,机器听一段声音,產生这段声音对应的文字,那你需要的就是一个函式,这个函式的输入是声音讯号,输出是这段声音讯号的内容。那你可以想像说 这个可以把声音讯号,当作输入文字当作输出的函式,显然非常非常的复杂，他绝对不是,你可以用人手写出来的方程式,这个函式他非常非常的复杂,人类绝对没有能力把它写出来,所以我们期待**凭藉著机器的力量,把这个函式自动找出来，这件事情,就是机器学习**。

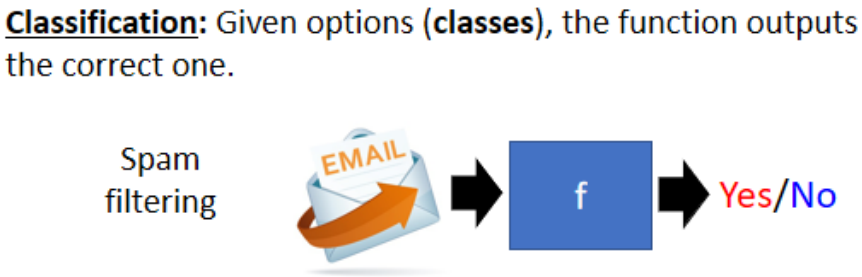
2. 还有好多好多的任务,我们都需要找一个很复杂的函式,举例来说 假设我们现在要做,影像辨识 那这个影像辨识,我们需要什麼样的函式呢？这个函式的输入是一张图片,他的输出是什麼呢？他是这个,图片裡面 有什麼样的内容。

3. 或者是大家都知道的AlphaGo,其实也可以看作是一个函式,要让机器下围棋 我们需要的,就是一个函式 这个函式的输入,是棋盘上黑子跟白子的位置,输出是什麼 输出是机器下一步,应该落子的位置 假设你可以,找到一个函式 这个函式的输入,就是棋盘上黑子跟白子的位置,输出就是下一步应该落子的位置,那我们就可以让机器做自动下围棋,这件事 就可以做一个AlphaGo。

Different types of Functions

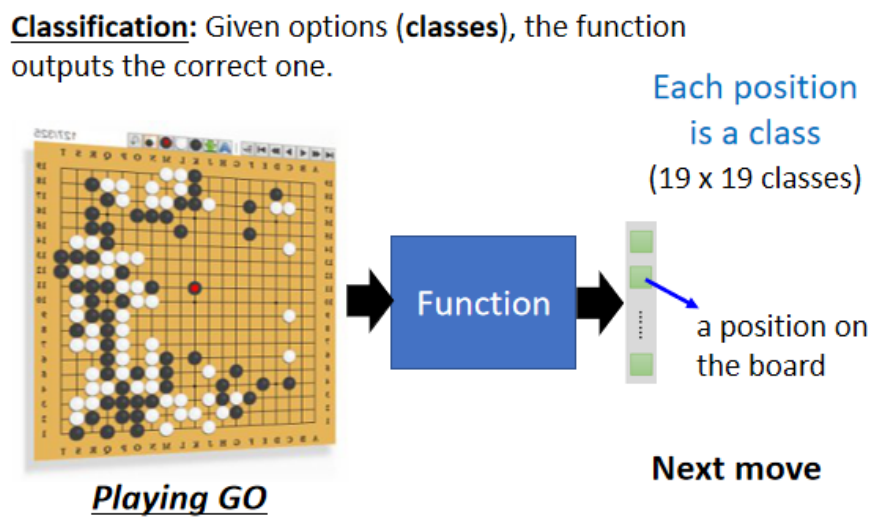
随著我们要找的函式不同,机器学习有不同的类别,那这边介绍几个专有名词给大家,认识一下 第一个专有名词,叫作==**Regression**==,Regression的意思是说,假设我们今天要找的函式,他的输出是一个数值,他的输出是一个 scalar,那这样子的机器学习的任务,我们称之為**Regression**。



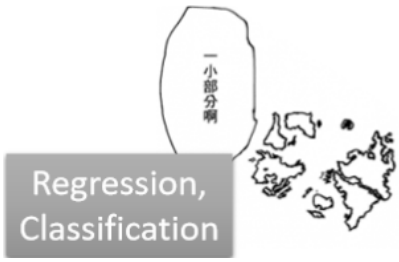


那这边举一个Regression的例子,假设我们今天要机器做的事情,是预测未来某一个时间的,PM2.5的数值 你要叫机器,做的事情是找一个函式,这个我们用f来表示,这个函式的输出,是明天中午的PM2.5的数值,他的输入可能是种种跟预测PM2.5,有关的指数 包括今天的PM2.5的数值,今天的平均温度,今天平均的臭氧浓度等等,这一个函式可以拿这些数值当作输入,输出明天中午的PM2.5的数值,那这一个找这个函式的任务,叫作Regression。

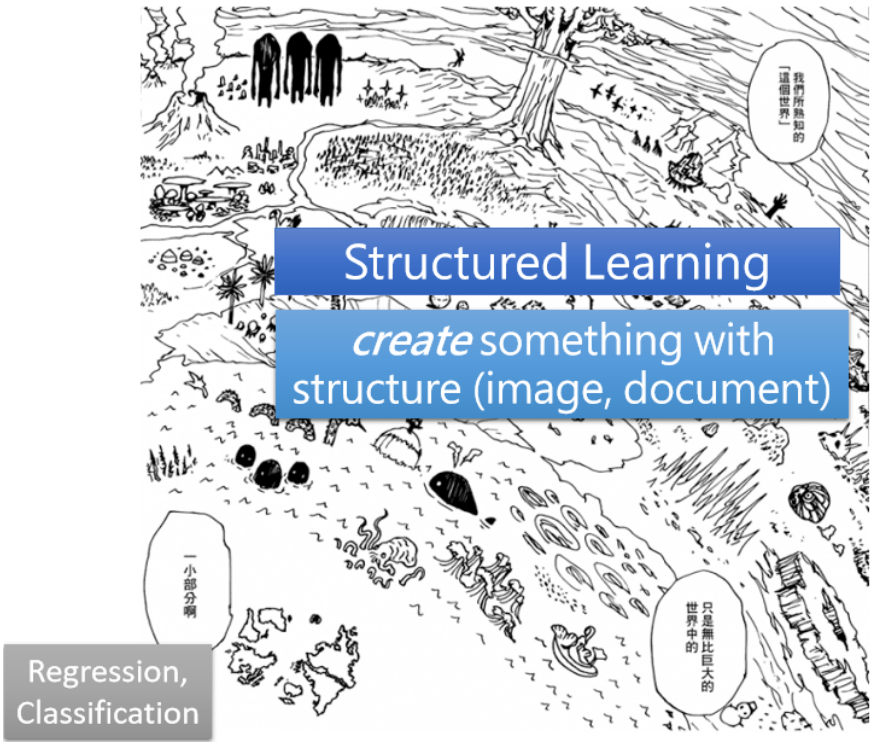
除了Regression以外,另外一个大家耳熟能详的任务,叫作==**Classification**==,那Classification这个任务,要**机器做的是选择题** 我们人类,先準备好一些选项 那这些选项,又叫作类别 又叫作classes,我们现在要找的函式它的输出,就是从我们设定好的选项裡面,选择一个当作输出 那这个问题,这个任务就叫作Classification。举例来说,现在每个人都有gmail account,那gmail account裡面有一个函式,这个函式可以帮我们侦测一封邮件,是不是垃圾邮件,这个函式的输入是一封电子邮件,那他的输出是什麼呢,你要先準备好你要机器选的选项,在侦测垃圾邮件这个问题裡面,可能的选项就是两个,是垃圾邮件 或不是垃圾邮件,Yes或者是No,那机器要从Yes跟No裡面,选一个选项出来,这个问题叫作Classification,那Classification不一定只有两个选项,也可以有多个选项 。



举例来说,alpha go本身也是一个Classification,的问题 那只是这个Classification,他的选项是比较多的,那如果要叫机器下围棋,你想做一个 alpha go的话,我们要给机器多少的选项呢,你就想想看 棋盘上有多少个位置,那我们知道棋盘上有19乘19个位置,那叫机器下围棋这个问题 其实,就是一个有19乘19个选项的选择题,你要叫机器做的就是找一个函式,这个函式的输入是棋盘上,黑子跟白子的位置,输出就是从19乘19个选项裡面,选出一个正确的选项,从19乘19个可以落子的位置裡面,选出下一步应该要落子的位置,那这个问题也是一个分类的问题。



其实很多教科书 在讲机器学习的,种种不同类型的任务的时候,往往就讲到这边 告诉你说,机器学习 两大类任务,一个叫作Regression,一个叫作Classification,然后就结束了,但是假设你对机器学习的认知,只停留在机器学习就是两大类任务,Regression跟Classification,那就好像你以為说,这个世界只有五大洲一样。

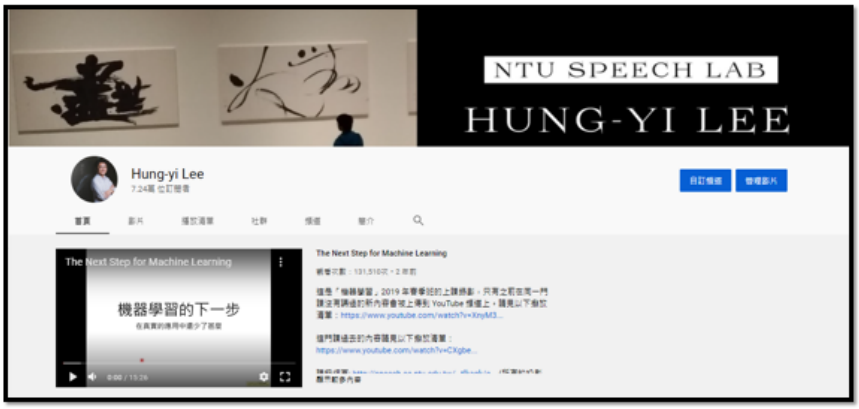


你知道这个世界不是只有五大洲,这个世界外面是有一个,黑暗大陆的 这鬼灭之刃连载之前,我们就已经出发前往黑暗大陆了,鬼灭之刃连载以后,我们居然都还没有到,可见这个黑暗大陆距离那麼远,那在机器学习这个领域裡面,所谓的黑暗大陆是什麼呢,在於Regression跟Classification以外,大家往往害怕碰触的问题,叫作Structured Learning,也就是机器今天不只是要做选择题,不只是输出一个数字 还要產生,一个有结构的物件 举例来说,**机器画一张图 写一篇文章,这种叫机器產生有结构的东西的问题,就叫作Structured Learning**,那如果要讲得比较拟人化,比较潮一点 Structured Learning,你可以用拟人化的讲法说,我就是要叫机器学会创造这件事情,好那到目前为止,我们就是讲了三个机器学习的任务,Regression Classification跟Structured Learning,

Case Study

我们说,机器学习就是要找一个函式,那机器怎麼找一个函式呢,那这边要用个例子跟大家说明说。在讲这个例子之前 先跟大家说一下,说这一门课有一个youtube的频道 https://www.youtube.com/c/HungyiLeeNTU,然后这个我会把上课的录影,放到这个youtube的频道上面,那这个频道 感谢过去修过这门课的,同学不嫌弃 其实也蛮多人订阅,所以我算是一个三流的youtuber,是没有什麼太多流量,但是这边也是有7万多订阅。

YouTube Channel



那為什麼突然提到,这个youtube的频道呢,因為我们等一下要举的例子,跟youtube是有关係的,youtuber在意的,就是这个频道的流量,假设有一个youtuber,是靠著youtube维生的,他会在意频道有没有流量,这样他才会知道他可以获利多少,假设你自己有youtube频道的话,你会知道说在youtube后台,你可以看到很多相关的资讯,你可以看到很多相关的资讯,比如说每一天按讚的人数有多少,每一天订阅的人数有多少,每一天观看的次数有多少,我们能不能够根据,一个频道过往所有的资讯去预测,它明天有可能的观看的次数是多少呢,我们能不能够**找一个函式,这个函式的输入是youtube上面,youtube后台是我的资讯,输出就是某一天,隔天这个频道会有的总观看的次数**.

The function we want to find ...

$y = f($ no. of views on 2/26 $)$



机器学习找这个函式的过程,分成三个步骤,那我们就用Youtube频道,点阅人数预测这件事情,来跟大家说明这三个步骤,是怎麼运作的

1. Function with Unknown Parameters



$y = f($ $)$

Model $y = b + wx_1$ based on domain knowledge

feature

y: no. of views on 2/26, x_1: no. of views on 2/25

w and b are unknown parameters (learned from data)

weight bias

第一个步骤是我们要==**写出一个,带有未知参数的函式**==,简单来说就是 我们先猜测一下,我们打算找的这个函式F,它的数学式到底长什麼样子。举例来说,我们这边先做一个最初步的猜测,我们写成这个样子

$$ y=b+w*x_1

$$ 这边的每一个数值是什麼呢,这个y啊 就假设是今天吧,不过今天还没有过完,所以我还不知道,今天总共的点阅次数是多少,所以这件事情是我们未知的,

- **y是我们準备要预测的东西**,我们準备要预测的是今天,2月26号这个频道总共观看的人
- **x_1是这个频道,前一天总共观看的人数**,y跟x_1都是数值,
- **b跟w是未知的参数,它是準备要透过资料去找出来的**,我们还不知道w跟b应该是多少,我们只是隐约的猜测

这个猜测往往就来自於,你对这个问题本质上的了解,也就是Domain knowledge,所以才会听到有人说,这个做机器学习啊,就需要一些Domain knowledge,.

所以我们这个能够预测未来点阅次数的函式F,它就一定是前一天的点阅次数,乘上w 再加上b呢,我们先不知道 这是一个猜测,也许我们觉得说,这个今天的点阅次数,总是会跟昨天的点阅次数有点关联,所以我们把昨天的点阅次数,乘上一个数值,但是总是不会一模一样,所以再加上一个b做修正,当作是对於2月26号,点阅次数的预测,这是一个猜测,它不一定是对的,我们等一下回头会再来修正这个猜测。

Model $y = b + wx_1$ based on domain knowledge

feature

y: no. of views on 2/26, x_1: no. of views on 2/25

w and b are unknown parameters (learned from data)

weight bias

那现在总之,我们就随便猜说,$y=b+wx_1$,*而**b跟w是未知的**,这个带有未知的==**参数**==,这个**Parameter中文通常翻成参数**,==这个带有Unknown的Parameter的Function 我们就叫做Model==,所以我们常常听到有人说,模型 Model,*Model这个东西在机器学习裡面,就是一个带有,未知的Parameter的Function*,

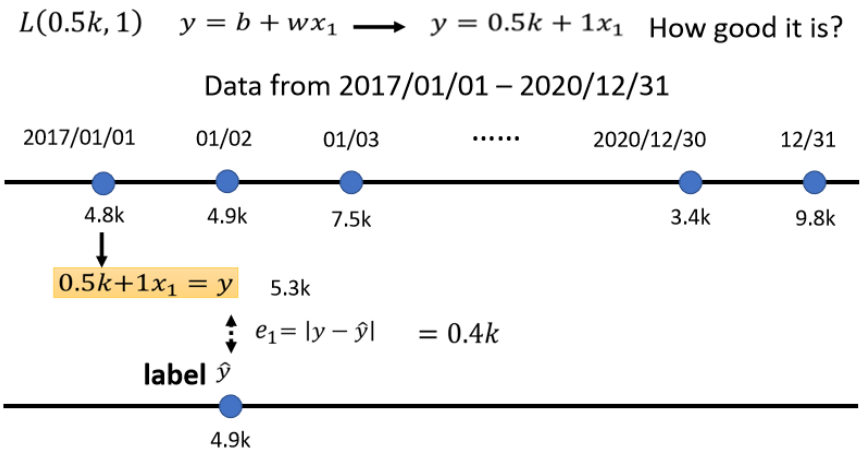
这个x_1是这个Function裡面我们已知的,已经知道的东西,它是来自於Youtube后台的资讯,我们已经知道2月25号,点阅的总人数是多少,这个东西叫做==Feature==,而w跟b是我们不知道的,它是Unknown的Parameter,那这边我们也给w跟b,给他一个名字,这个跟Feature做相乘的未知的参数,这个w 我们叫它==weight==,这个没有跟Feature相乘的,是直接加下去的,这个我们叫它==Bias==,那这个只是一些名词的定义而已,等一下我们讲课的时候,我们在称呼,模型裡面的每一个东西的时候,会更為方便,好那这个是第一个步骤.

2. Define Loss from Training Data

- Loss is a function of parameters $L(b, w)$
- Loss: how good a set of values is.

第二个步骤,是我们要==定义一个东西叫做Loss==,**Loss它也是一个Function,那这个Function它的输入,是我们Model裡面的参数**,我们的Model叫做,$y=b+wx_1$,*而b跟w是未知的,是我们準备要找出来的,所谓的这个Loss啊,它是一个Function,这个Function的输入,就是b跟w,所以L它是一个Function,它的输入是Parameter,是model裡面的Parameter,那这个Loss 这个Function,*这个Function输出的值代表说,现在如果我们把这一组未知的参数,设定某一个数值的时候,这笔数值好还是不好。*

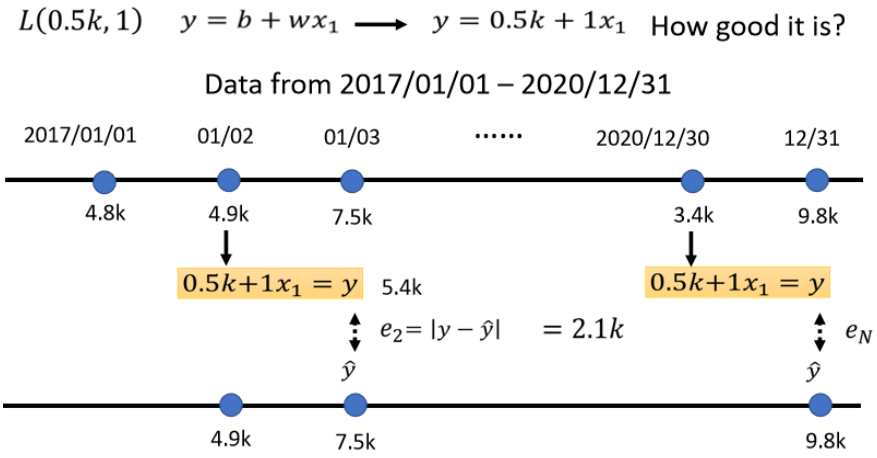
那这样讲可能你觉得有点抽象,所以我们就举一个具体的例子,假设现在我们给未知的参数的设定是b,这bias等於0.5k,这个w呢直接等於1,那这个Loss怎麼计算呢,如果我们b设0.5k,这个w设1,那我们拿来预测,未来的点阅次数的函式 就变成,y等於0.5k加1倍的x_1,那这样子的一个函式,这个0.5k跟1,他们所代表的这个函式,它有多少呢,这个东西就是Loss.



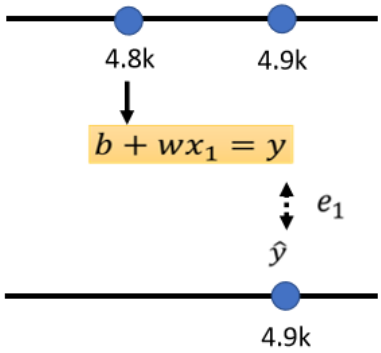
我们要怎麼计算这个Loss呢,这个我们就要==**从训练资料来进行计算**==,在这个问题裡面,我们的训练资料是,这一个频道过去的点阅次数,举例来说,从2017年到2020年的点阅次数,每天的这个频道的点阅次数都知道,这边是假的数字啦,随便乱编的,好

所以那我们知道,2017年1月1号,到2020年12月31号的,点阅数字是多少,接下来我们就可以计算Loss。

我们把2017年1月1号的点阅次数,代入这一个函式裡面,我们已经说我们想要知道,b设定为0.5k,w设定为1的时候,这个函式有多棒,当b设定为0.5k,w设定为1的时候,我们拿来预测的这个函数,是y等於0.5k加一倍的x_i,那我们就把这个x_i代4.8k,看它的预测出来的结果是多少,所以 根据这一个函式,根据b设0.5k,w设1的这个函式,如果1月1号,是4.8k的点阅次数的话,那隔天应该是4.8k乘1加0.5k,就是5.3k的点阅次数,那隔天实际上的点阅次数,1月2号的点阅次数我们知道吗,从后台的资讯裡面 我们是知道的,所以我们可以**比对一下,现在这个函式预估的结果,跟真正的结果,它的差距有多大**,这个函式预估的结果是5.3k,真正的结果是多少呢,真正的结果是4.9k,那这个真实的值叫做==Label==,它是高估了,高估了这个频道可能的点阅人数,那就可以计算一下这个差距,计算一下估测的值,跟真实的值的差距,这边**估测的值用y来表示,真实的值用ŷ**来表示,你可以计算y跟ŷ之间的差距,得到一个e_i,代表估测的值跟真实的值之间的差距,那**计算差距其实不只一种方式**,我这边把y跟ŷ相减,直接取绝对值,算出来的值是0.4k。



那我们不是只能用1月1号,来预测1月2号的值,我们可以用1月2号的值,来预测1月3号的值,如果我们现在的函式是,y等於0.5k加一倍的x_i,那1月2号,根据1月2号的点阅次数,预测的1月3号的点阅次数的,值是多少呢 是5.4k,以x_i代4.9k进去,乘1在加0.5k等於5.4k,接下来计算这个5.4k,跟真正的答案,跟Label之间的差距,Label是7.5k,看来是一个低估,低估了这个频道,在1月3号的时候的点阅次数,才可以算出e_2,这个e_2是,y减y跟ŷ之间的差距,算出来是2.1k,那同一个方法,你就可以算过这三年来,每一天的预测的误差,假设我们今天的Function,是y等於0.5k加一倍的x_i,这三年来每一天的误差,通通都可以算出来,每一天的误差都可以给我们一个小e,



Loss: $L = \frac{1}{N}\sum_n e_n$

$e = |y - \hat{y}|$ L is mean absolute error (MAE)

$e = (y - \hat{y})^2$ L is mean square error (MSE)

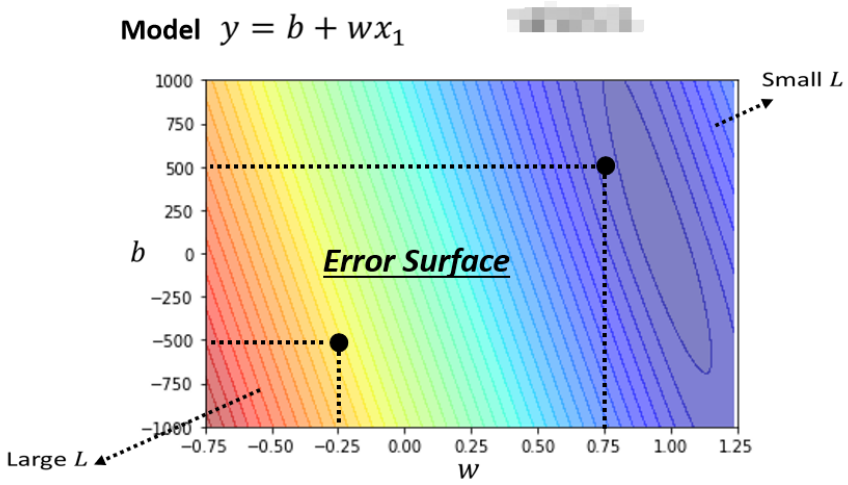
接下来我们就**把每一天的误差,通通加起来,加起来然后取得平均**,这个大N代表我们的训验资料的个数,,就是三年来的训练资料,就365乘以3,每年365天 所以365乘以3,那我们算出一个L,我们算出一个大L,这大L是每一笔训练资料的误差,这个e相加以后的结果,这个大L就是我们的Loss.

==大L越大,代表我们现在这一组参数越不好,这个大L越小,代表现在这一组参数越好==

估测的值跟实际的值之间的差距,其实有不同的计算方法,在我们刚才的例子裡面,我们是算y跟ŷ之间,绝对值的差距,这一种计算差距的方法,得到的这个大L,得到的Loss叫 mean absolute error,缩写是**MAE**,如果你今天的e是用,相减y平方算出来的,这个叫mean square error,又叫**MSE**,那MSE跟MAE,他们其实有非常微妙的差别.我们就是选择MAE,作为我们计算这个误差的方式,把所有的误差加起来,就得到Loss,那你要选择MSE也是可以的,在作业裡面我们会用MSE,那有一些任务,如果y和ŷ它都是机率,都是**机率分佈**的话,在这个时候,你可能会选择**Cross-entropy**,这个我们都之后再说,我们这边就是选择了MAE,那这个是机器学习的第二步.

Error Surface

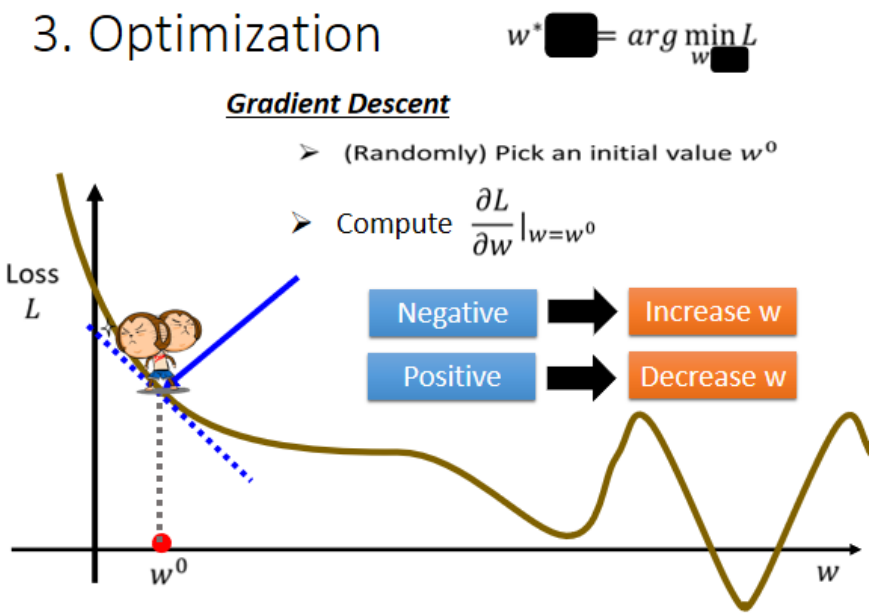
我刚才举的那些数字,不是真正的例子,以下的数字,是真实的例子,是这个频道真实的后台的数据,所计算出来的结果,那我们可以调整不同的w,我们可以调整不同的b,求取各种w 求取各种b,组合起来以后,我们可以為不同的w跟b的组合,都去计算它的Loss,然后就可以画出以下这一个**等高线图.**



在这个等高线图上面,**越偏红色系,代表计算出来的Loss越大,就代表这一组w跟b越差,如果越偏蓝色系,就代表Loss越小,就代表这一组w跟b越好**,拿这一组w跟b,放到我们的Function裡面,放到我们的Model裡面,那我们的预测会越精準,所以你就知道说,假设w在负0.25,这个b在负500,就代表说呢这个W在负0.25,b在负500 就代表说,这个频道每天看的人越来越少,而且Loss这麼大,跟真实的状况不太合,如果w代0.75 b代500,那这个正确率,这个估测会比较精準,那估测最精準的地方看起来,应该是在这裡啦,如果你今天w代一个很接近1的值,b带一个小小的值,比如说100多,那这个时候估测是最精準的,那这跟大家的预期可能是比较接近的,就是你拿前一天的点阅的总次数,去预测隔天的点阅的总次数,那可能前一天跟隔天的点阅的总次数,其实是差不多的,所以w设1,然后b设一个小一点的数值,也许你的估测就会蛮精準的,那像这样子的一个等高线图,就是你试著**试了不同的参数,然后计算它的Loss,画出来的这个等高线图**,叫做==Error Surface==,那这个是机器学习的第二步,

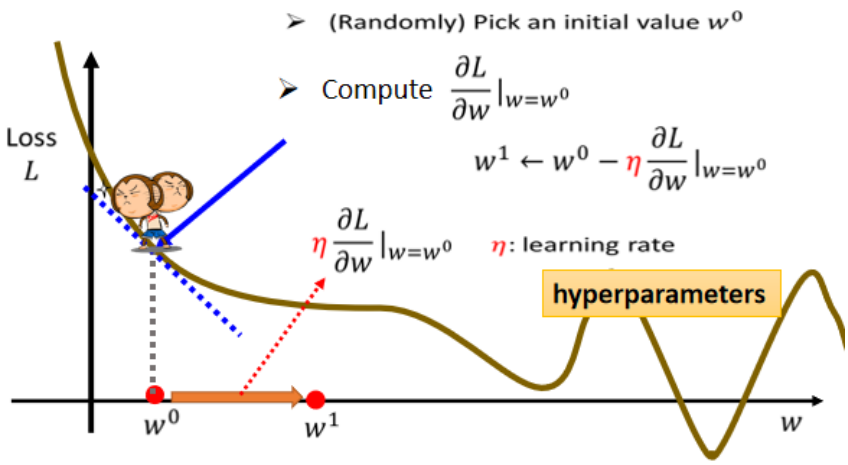
3. Optimization

接下来我们进入机器学习的第三步,那第三步要做的事情,其实是==解一个最佳化的问题==,如果你不知道最佳化的问题,是什麼的话也没有关係,我们今天要做的事情就是,**找一个w跟b,把未知的参数,找一个数值出来,看代那一个数值进去,可以让我们的大L,让我们的Loss的值最小,那个就是我们要找的w跟b**,那这个可以让loss最小的w跟b,我们就叫做w *跟* b,$代表说他们是最好的一组w跟b,可以让loss的值最小.



在这一门课裡面 我们唯一会用到的,Optimization的方法,叫做==Gradient Descent==。**為了要简化起见,我们先假设我们未知的参数只有一个,就是w**,我们先假设没有b那个未知的参数,只有w这个未知的参数.

那当我们**w代不同的数值的时候,我们就会得到不同的Loss,这一条曲线就是error surface**,只是刚才在前一个例子裡面,我们看到的error surface,是二维的是2D的,那**这边只有一个参数,所以我们看到的这个error surface,是1D**的.



==那怎麼样找一个w,去让这个loss的值最小呢?==

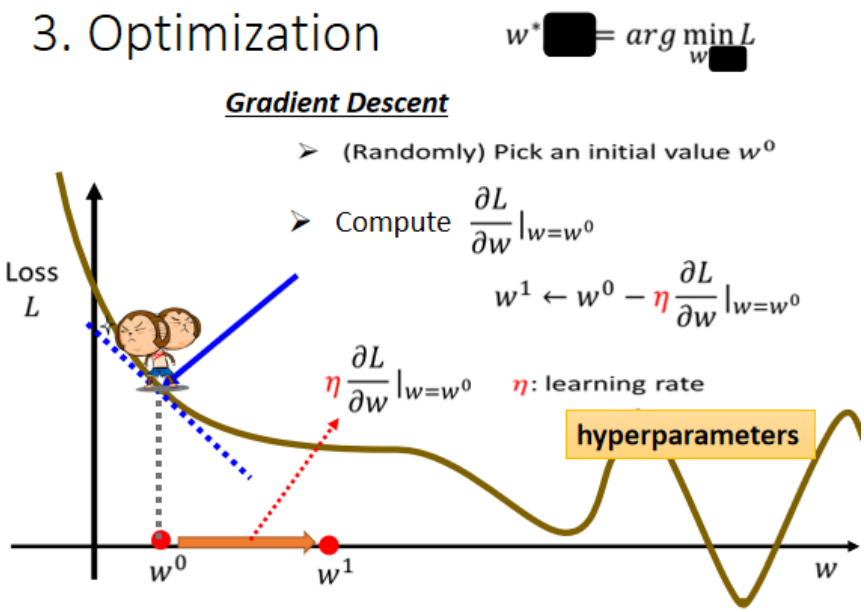
- 那首先你要**随机选取一个初始的点**,那这个初始的点,我们叫做**w_0**,那这个初始的点往往真的就是随机的,那在往后的课程裡面,我们其实会看到也许有一些方法,可以给我们一个比较好的w零的值.我们先当作都是随机的,那假设我们随机决定的结果,是在w0这个地方。
- 那接下来你就要计算,**在w等於w0的时候,w这个参数对loss的微分是多少**$∂L/∂w |(w=w^0)$,如果你不知道微分是什麼的话,那没有关係反正我们做的事情就是,计算在这一个点,在**w_0这个位置的error surface的切线斜率**,也就是这一条**蓝色的虚线,它的斜率**,那如果这一条虚线的斜率是负的,那代表什麼意思呢,代表说左边比较高 右边比较低,在这个位置附近,左边比较高 右边比较低。
- 那如果**左边比较高右边比较低**的话,我们就把**w的值变大**,那我们就可以让**loss变小**,如果算出来的斜率是正的,就代表说左边比较低右边比较高,是这个样子的,左边比较低右边比较高,如果左边比较低 右边比较高的话,那就代表我们把w变小了,w往左边移 我们可以让Loss的值变小,那这个时候你就应该把w的值变小,那假设你连斜率是什麼都不知道的话,也没有关係,你就想像说有一个人站在这个地方,然后他左右环视一下,那这一个算微分这件事啊,就是左右环视,它会知道左边比较高还是右边比较高,看哪边比较低,它就往比较低的地方跨出一步,

  那这一步要跨多大呢,这一步的步伐的大小取**决於两件事情**,

  - 第一件事情是**这个地方的斜率有多大**,这个地方的斜率大,这个步伐就跨大一点,斜率小步伐就跨小一点,
  - 另外 除了斜率以外,就是除了微分这一项,还有另外一个东西会影响步伐大小,这个东西我们这边用$η$来表示,这个$η$叫做==learning rate==,叫做学习速率,这个learning rate 它是怎麼来的呢,==它是你自己设定的,你自己决定这个$η$的大小==,如果$η$设大一点,那你每次参数update就会量大,你的学习可能就比较快,如果η设小一点,那你参数的update就很慢,每次只会改变一点点参数的数值,那这种你在做机器学习,需==要自己设定的东西,叫做hyperparameters==

有同学问说,為什麼loss可以是负的呢?

Loss这个函数是自己定义的,所以在刚才我们的定义裡面,我们说loss就是估测的值,跟正确的值,它的绝对值,那如果根据刚才loss的定义,那它不可能是负的,但是loss的这一个function,是你自己决定的,你可以说,我今天要决定一个loss function,就是绝对值再减100,那你可能就有负的,所以我这边这一个curve,我这边可能刚才忘了跟大家说明说,这个curve并不是一个真实的loss,它是我随便乱举的一个例子,因為在今天,我想要举一个比较general 的case,它并不是一个真实任务的,Error surface,所以这个loss的这个curve,这个error surface,它可以是任何形状,这边没有预设立场说,它一定要是什麼形状,但是确实在真实,在刚才这一个如果loss的定义,就跟我刚才定的一样是绝对值,那它就不可能是负值,但这个loss,**这个function是你自己决定的,所以它有可能是负的**



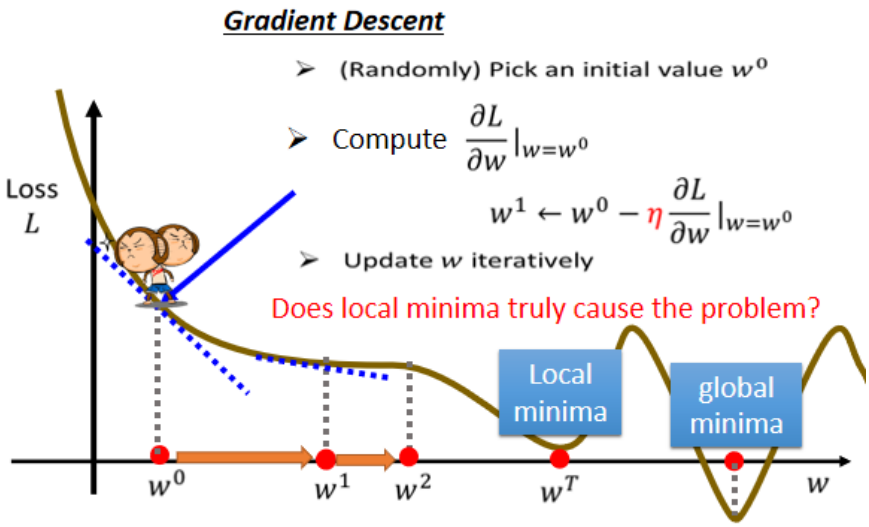
那我们说我们要把w^0往右移一步,那这个新的位置就叫做w^1,这一步的步伐是η乘上微分的结果,那如果你要用数学式来表示它的话,就是把w^0减掉η乘上微分的结果,得到w^1

$$ w^1←w^0-η \frac{∂L}{∂w}|(w=w^0)

$$,那接下来你就是反覆进行刚才的操作,你就计算一下w^1微分的结果,然后再决定现在要把w^1移动多少,然后再移动到w^2,然后你再继续反覆做同样的操作,不断的把w移动位置,最后你会停下来.

==什麼时候会停下来呢?往往有两种状况==,

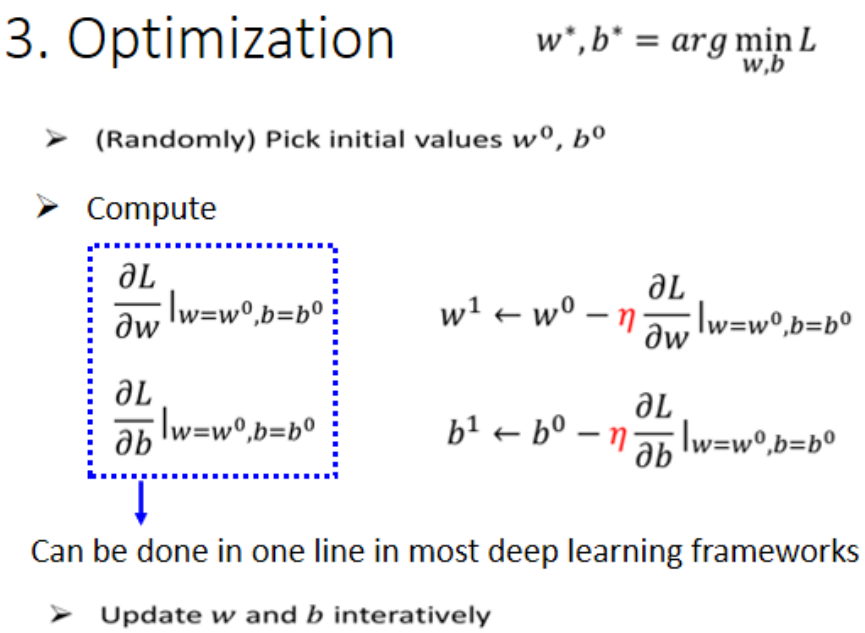
- 第一种状况是你失去耐心了,你一开始会设定说,我今天在调整我的参数的时候,我在计算我的微分的时候,我**最多计算几次**,你可能会设说,我的上限就是设定100万次,就我参数更新100万次以后,我就不再更新了,那至於==要更新几次,这个也是一个hyperparameter,这个是你自己决定的==,做一个deadline是明天,那你可能更新的次数就设少一点,对它下周更新的次数就设多一点,
- 那还有另外一种理想上的,停下来的可能是,今天当我们不断调整参数,调整到一个地方,它的微分的值就是这一项,算出来正好是0的时候,如果这一项正好算出来是0,0乘上learning rate η还是0,所以你的参数就不会再移动位置,那假设我们是这个理想的状况,我们把w^0更新到w^1,再更新到w^2,最后更新到w^t有点卡,w^t卡住了,也就是算出来这个微分的值是0了,那参数的位置就不会再更新,



你可能会马上发现说,Gradient Descent 这个方法,有一个巨大的问题,我们没有找到真正最好的解,**我们没有找到那个,可以让Loss最小的那个w**,在这个例子裡面,把w设定在右侧红点附近这个地方,你可以让loss最小,但是如果 Gradient Descent,是从W^0^这个地方,当作随机初始的位置的话,也很有可能走到W^T^这裡,你的训练就停住了,你就没有办法再移动w的位置。

那右侧红点这一个位置,这个真的可以让loss最小的地方,叫做==global 的minima==,而W^T^这个地方叫做==local 的minima==,它的左右两边,都比这个地方的loss还要高一点,但是它不是整个error surface上面的最低点。

所以常常可能会听到有人讲到,**Gradient Descent,不是个好方法,这个方法会有local minima的问题**,没有办法真的找到global minima,但教科书常常这样讲,农场文常常这样讲,但这个其实**只是幻觉而已**,事实上,假设你有做过深度学习相关的事情,假设你有自己训练network,自己做过Gradient Descent 经验的话,其实**local minima是一个假问题**,我们在做Gradient Descent 的时候,真正面对的难题不是local minima,到底是什麼,这个 我们之后会再讲到,在这边你就先接受,先相信多数人的讲法说,Gradient Descent,有local minima的问题,在这个图上在这个例子裡面,显然有local minima的问题,但之后会再告诉你说,Gradient Descent真正的痛点,到底是什麼。



刚才举的例子,是只有一个参数的例子而已,那实际上**我们刚才的模型有两个参数,有w跟b**,那有两个参数的情况下,怎麼用Gradient Descent呢,其实跟刚才一个参数没有什麼不同,若一个参数你没有问题的话,你可以很快的推广到两个参数。

- 我们现在有两个参数,都给它随机的初始的值,就是w^0跟b^0
- 你要计算w跟loss的微分,你要计算b对loss的微分,计算是在w等於w^0的位置,b等於b_0的位置,在w等於w_0的位置,b等於b^0的位置,你要计算w对L的微分,计算b对L的微分, $$ \frac{∂L}{∂b} |(w=w^0,b=b^0) $$

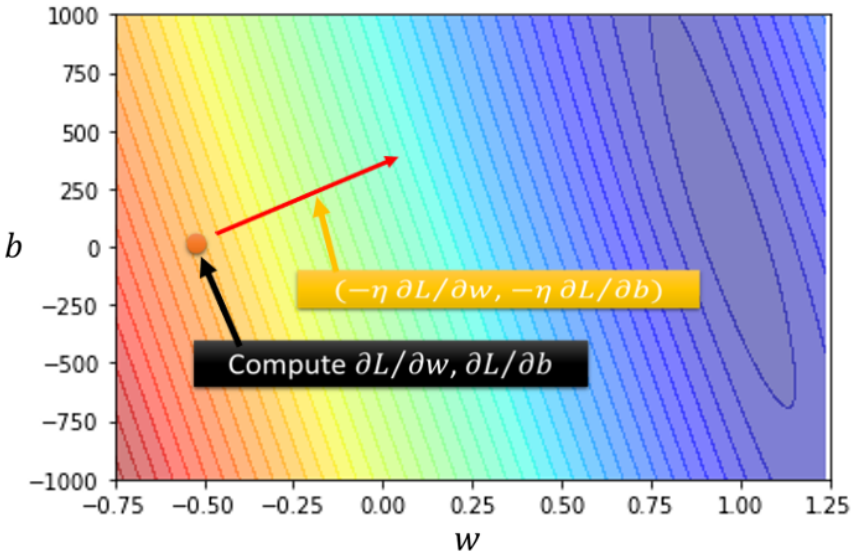
  $$ \frac{∂L}{∂w} |(w=w^0,b=b^0) $$

  计算完以后,就根据我们刚才,一个参数的时候的做法,去更新w跟b,把w^0减掉learning rate,乘上微分的结果得到w^1,把b^0减掉learning rate,乘上微分的结果得到b^1 $$ w^1←w^0-η \frac{∂L}{∂w} |(w=w^0,b=b^0) $$

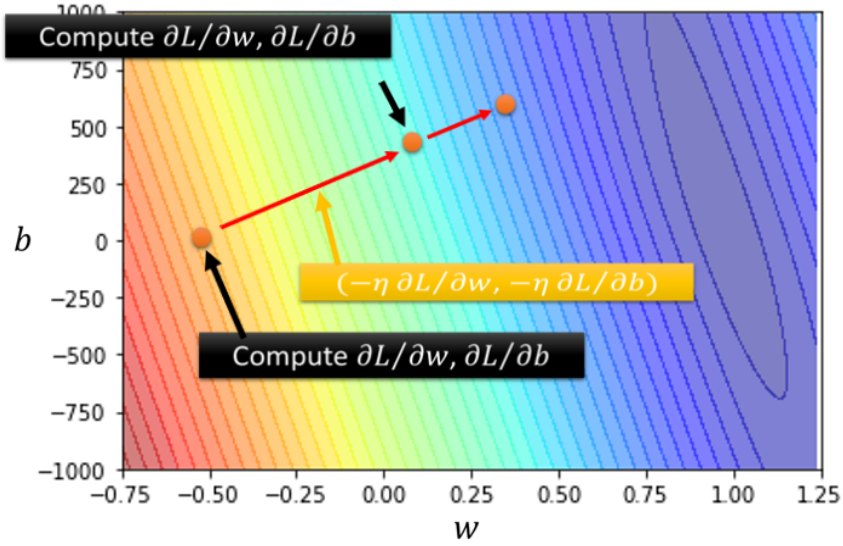
$$ b^1←b^0-η \frac{∂L}{∂b} |(w=w^0,b=b^0)

$$

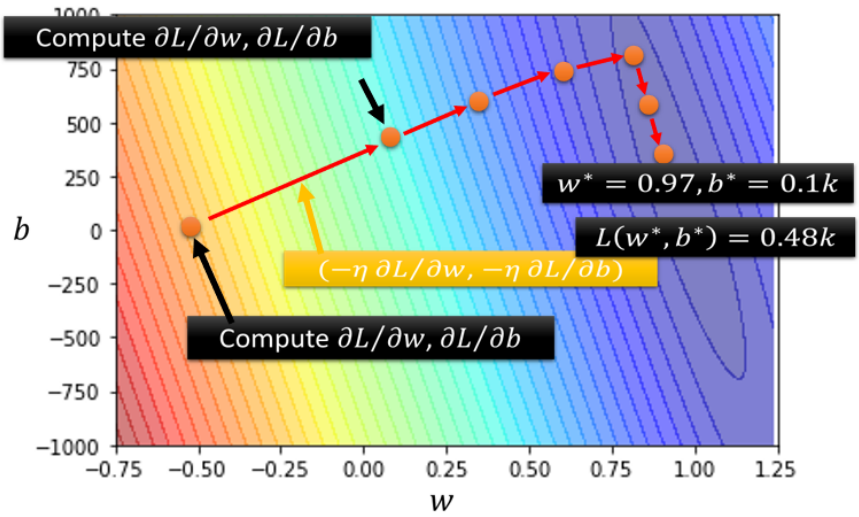
如果你不会算微分的话,不用紧张,在deep learning 的framework裡面,或在我们作业一,会用的pytorch裡面,算微分都是程式自动帮你算的,你就co一行 就写一行程式,自动就把微分的值就算出来了,你就算完全不知道自己在干嘛,也还是可以把微分的值算出来,所以这边,如果你根本就不知道微分是什麼,不用担心,这一步骤就是一行程式,这个等一下之后在做AE的时候,大家可以自己体验看看,那就是反覆同样的步骤,就不断的更新w跟b,然后期待最后,你可以找到一个最好的w,$w^$ *跟最好的b $b*^$.



如果在这一个问题上,它操作起来是什麼样子,假设,你随便选一个初始的值在这个地方,那你就先计算一下w对L的微分,跟计算一下b对L的微分,然后接下来你就要更新w跟b,更新的方向就是w对L的微分,乘以η再乘以一个负号,b对L的微分,**算出这个微分的值,你就可以决定更新的方向**,你就可以决定w要怎麼更新,那**把w跟b更新的方向结合起来,就是一个向量,就是这个红色的箭头**,我们就从这个位置移到这个位置,

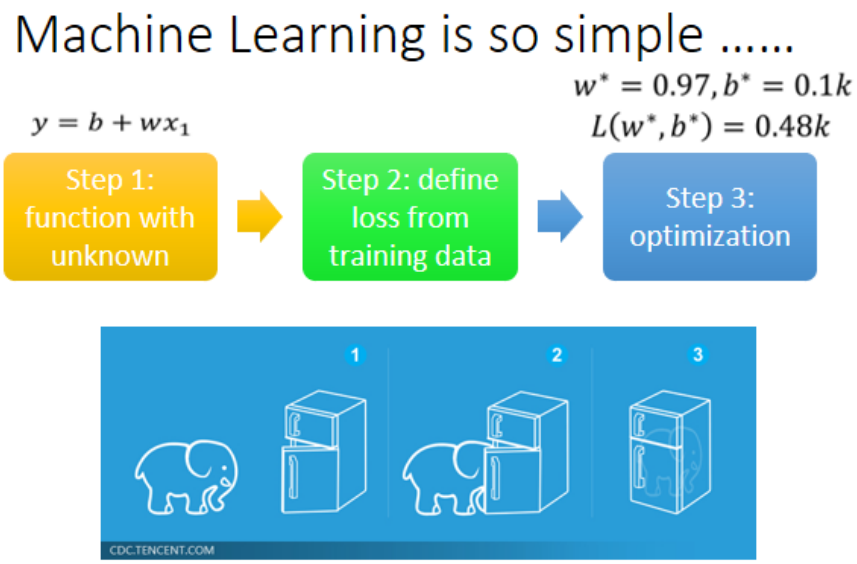


然后再计算一次微分,然后你再决定要走什麼样的方向,把这个微分的值乘上learning rate,再乘上负号,你就知道红色的箭头要指向那裡,你就知道怎麼移动w跟b的位置,一直移动一直移动一直移动,期待最后可以找出一组不错的w跟b,

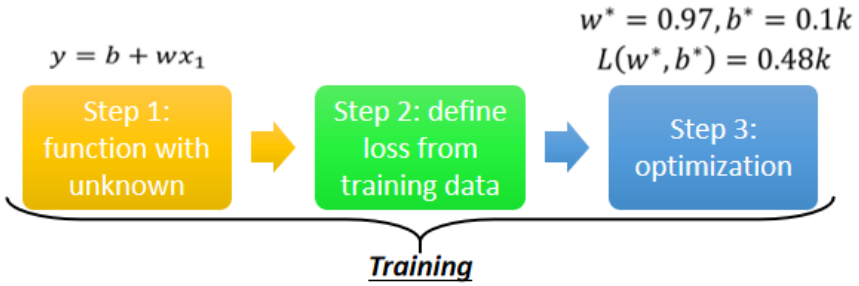


那实际上真的用Gradient Descent,进行一番计算以后,这个是真正的数据,我们算出来的最好的w是0.97,最好的b是0.1k,跟我们的猜测蛮接近的,因为x_1的值可能跟y很接近,所以这个w就设一个接近1的值,b就设一个比较偏小的值,那loss多大呢,loss算一下是0.48k,也就是在2017到2020年的资料上,如果使用这一个函式,b代0.1k,w代0.97,那平均的误差是0.48k,也就是它的预测的观看人数误差,大概是500人次左右.

Linear Model



那w跟b的值刚才已经找出来的,那这组w跟b可以让loss小到0.48k,但是这样,是一个让人满意或值得称道的结果吗,也许不是,因为==这三个步骤合起来==,叫做==训练==,**我们现在是在,我们已经知道答案的资料上,去计算loss**,我们其实已经知道2017到2020年,每天的观看次数 所以,其实我们现在其实只是在自high而已,就是假装我们不知道隔天的观看次数,然后拿这一个函式来进行预测,发现误差是0.48k。

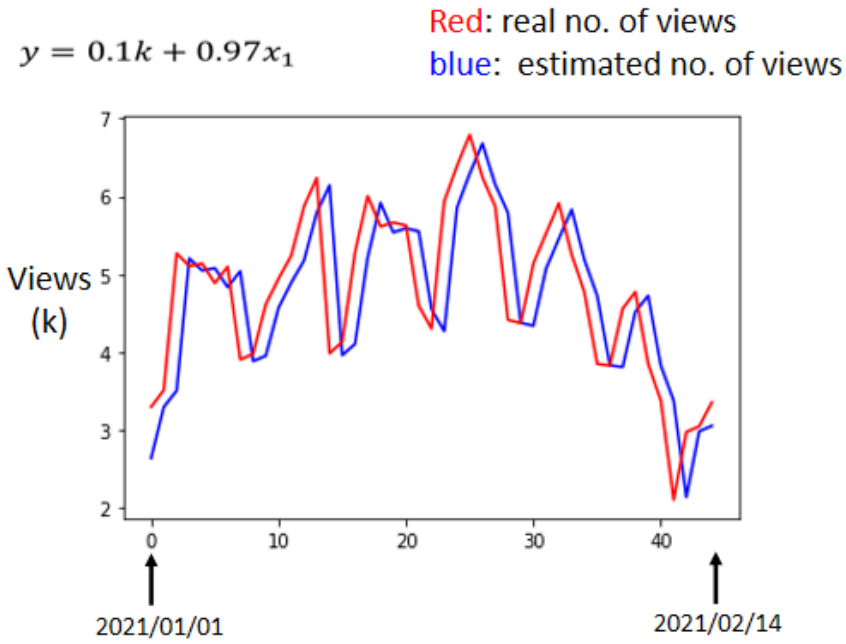


$y = 0.1k + 0.97x_1$ achieves the smallest loss $L = 0.48k$ on data of 2017 – 2020 (**training data**)

How about data of 2021 (**unseen during training**)?

$L' = 0.58k$

但是我们真正要在意的,我们不知道的,未来的观看的次数是多少,所以我们接下来要做的事情就是拿这个函式,来**真的预测一下未来的观看次数**,那这边,我们只有2017年到2020年的值,我们在2020年的最后一天,跨年夜的时候,找出了这个函式,接下来从2021年开始每一天,我们都拿这个函式,去预测隔天的观看人次,我们就拿2020年的12月31号的,观看人次,去预测2021年元旦的观看人次,用2021年元旦的观看人次,预测一下2021年元旦隔天,1月2号的观看人次,用1月2号的观看人次去预测,1月3号的观看人次,每天都做这件事,一直做到2月14号,就做到情人节,然后得到平均的值,平均的误差值是多少呢,这个是真实的数据的结果,在2021年没有看过的资料上,这个误差值是,我们这边用L prime来表示,它是0.58,所以在有看过的资料上,在训练资料上,误差值是比较小的,在没有看过的资料上,在2021年的资料上,看起来误差值是比较大的,那我们每一天的平均误差,有580人左右,600人左右。



能不能够做得更好呢,在做得更好之前,我们先来分析一下结果

- **横轴是代表的是时间**,所以0这个点 最左边的点,代表的是2021年1月1号,最右边点,代表的是2021年2月14号
- **纵轴就是观看的人次**,这边是用千人当作单位,
- **红色的线是真实的观看人次**

- **蓝色的线**是机器用这一个函式,**预测出来的观看人次**

你有发现很明显的,这蓝色的线没什麼神奇的地方,它几乎就是,红色的线往右平移一天而已,它其实也没做什麼特别厉害的预测,就把红色的线往右平移一天,因為这很合理,因為我们觉得,x_1也就是前一天的观看人次,跟隔天观看人次的,要怎麼拿前一天的观看人次,去预测隔天的观看人次呢,前一天观看人次乘以0.97,加上0.1k 加上100,就是隔天的观看人次,所以你会发现说,机器几乎就是拿前一天的观看人次来预测,隔天的观看人次,但是如果你仔细观察这个图,你就会发现,**这个真实的资料有一个很神奇的现象,它是有週期性的**,它每隔七天就会有两天特别低,**两天观看的人特别少**,那两天是什麼日子呢,那我发现那两天都固定,**是礼拜五跟礼拜六**,礼拜五跟礼拜六我可以了解,就礼拜五週末 大家出去玩,谁还要学机器学习,礼拜六谁还要学机器学习,那不知道為什麼,礼拜天大家去学机器学习,这个我还没有参透為什麼是这个样子,也许跟youtube背后,神奇的演算法有关係,比如说youtube都会推频道的影片,也许youtube在推频道的影片的时候,它都选择礼拜五礼拜六不推,只推礼拜天到礼拜四,可是為什麼推礼拜天到礼拜四呢,这个我也不了解,但是反正看出来的结果,我们看真实的数据,就是这个样子,每隔七天一个循环,每个礼拜五礼拜六,看的人就是特别少。所以既然我们已经知道每隔七天,就是一个循环,那这一个式子 这一个model,显然很烂,因為它只能够看前一天。

每隔七天它一个循环,如果我们**一个模型,它是参考前七天的资料**,把七天前的资料,直接复製到拿来当作预测的结果,也许预测的会更準也说不定,所以我们就要修改一下我们的模型,通常一个模型的修改,往往来自於你对这个问题的理解,也就是**Domain Knowledge**。

$y = b + wx_1$	2017 - 2020 $L = 0.48k$	2021 $L' = 0.58k$
$y = b + \sum_{j=1}^{7} w_j x_j$	2017 - 2020 $L = 0.38k$	2021 $L' = 0.49k$

所以一开始,我们对问题完全不理解的时候,我们就胡乱写一个

$$ y=b+wx_1

$$ 并没有做得特别好,接下来我们观察了真实的数据以后,得到一个结论是,每隔七天有一个循环,所以我们应该要把,前七天的观看人次都列入考虑,所以我们写了一个新的模型,

$$y = b + \sum_{j=1}^{7} w_j x_j$$

x_j代表什麼,这个下标j代表是几天前,然后这个j等於1到7,也就是从一天前两天前,一直考虑到七天前,那七天前的资料,通通乘上不同的weight,乘上不同的w_j,加起来,再加上bias,得到预测的结果,

如果这个是我们的model,那我们得到的结果是怎麼样呢,我们在训练资料上的loss是0.38k,那因為这边只考虑一天,这边考虑七天,所以在训练资料上,你会得到比较低的loss,这边考虑了比较多的资讯,在训练资料上你应该要得到更好的,更低的loss,这边算出来是0.38k,但它在没有看过的资料上面,做不做得好呢,在没有看到的资料上,有比较好,是0.49k,所以刚才只考虑一天是0.58k的误差,考虑七天是0.49k的误差。

那这边每一个w跟b,我们都会用Gradient Descent,算出它的最佳值,它的最佳值长什麼样子呢,这边show出来,给你看 它的最佳值长这样,当然机器的逻辑我是有点没有办法了解,我本来以為它会选七天前的数据,七天前的观看人数,直接复製过来,我看来它没有这样选就是了。

b	w_1^*	w_2^*	w_3^*	w_4^*	w_5^*	w_6^*	w_7^*
0.05k	0.79	-0.31	0.12	-0.01	-0.10	0.30	0.18

$$y = b + \sum_{j=1}^{28} w_j x_j$$

2017 - 2020 $L = 0.33k$

2021 $L' = 0.46k$

$$y = b + \sum_{j=1}^{56} w_j x_j$$

2017 - 2020 $L = 0.32k$

2021 $L' = 0.46k$

Linear models

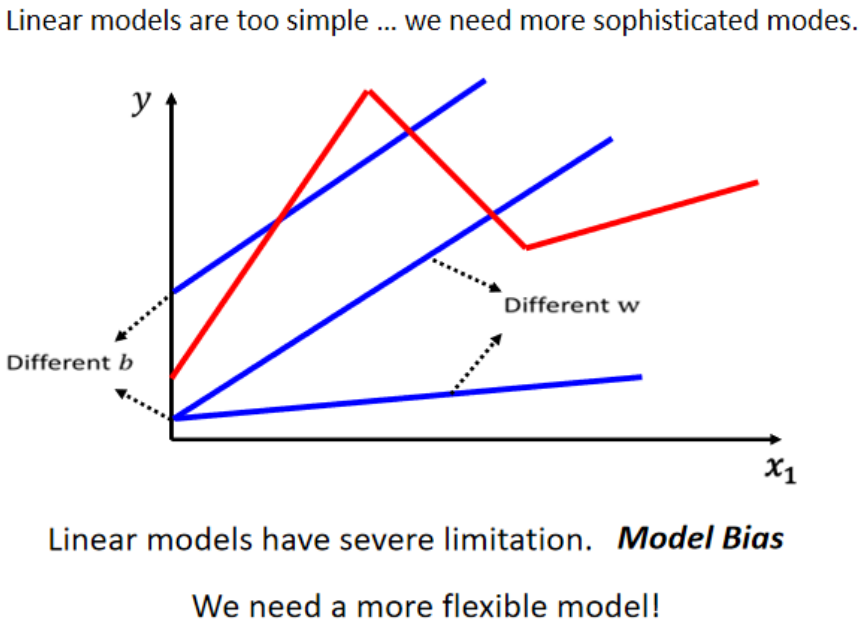
它的逻辑是前一天,跟你要预测的隔天的数值的关係很大,所以w_1*是0.79,那不知道為什麼 它还考虑前三天,前三天是0.12,然后前六天是0.3,前七天是0.18,不过它知道说,如果是前两天前四天前五天,它的值会跟未来我们要预测的,隔天的值是成反比的,所以w_2 w_4跟w_5它们最佳的值,让Loss可以在训练资料上,是0.38k的值 是负的,但是w_1 w_3 w_6跟w_7是正的,我们考虑前七天的值,那你可能会问说,能不能够考虑更多天呢,可以,那这个轻易的改考虑更多天,本来是考虑前七天

然后考虑28天会怎麼样呢,28天就一个月,考虑前一个月每一天的观看人次,去预测隔天的观看人次,预测出来结果怎样呢,训练资料上是0.33k,那在2021年的资料上,在没有看过的资料上是0.46k,看起来又更好一点 好 28天,好那接下来考虑56天会怎麼样呢,在训练资料上是稍微再好一点,是0.32k,在没看过的资料上还是0.46k,看起来,考虑更多天没有办法再更进步了,看来考虑天数这件事,也许已经到了一个极限,好那这边这些模型,它们都是把输入的这个x,这个x 还记得它叫什麼吗,它叫做feature,把feature乘上一个weight,再加上一个bias就得到预测的结果,这样的模型有一个共同的名字,叫做==Linear model==,那我们接下来会看,怎麼把Linear model做得更好。

Regression P2

Piecewise Linear Curves

Linear 的 Model,也许太过简单了,我们可以想像说 x1 跟 y,也许它中间有比较复杂的关係,**对 Linear 的 Model 来说,x1 跟 y 的关係就是一条直线**,随著 x1 越来越高,y 就应该越来越大,你可以设定不同的 w,改变这条线的斜率,你可以设定不同的 b,改变这一条蓝色的直线,跟 y 轴的交叉点,但是无论你怎麼改 w 跟 b,它永远都是一条直线,永远都是 x1 越大,y 就越大,前一天观看的人数越多,隔天的观看人数就越多

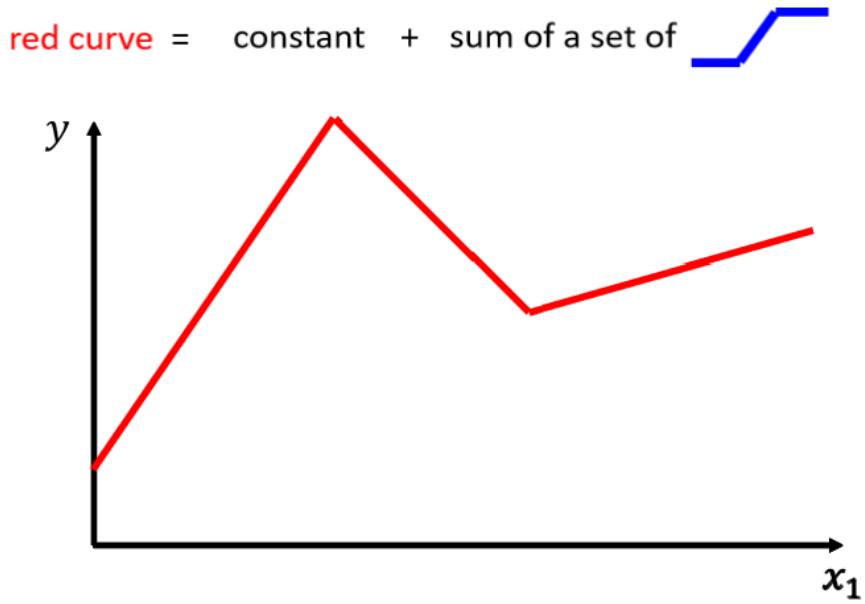


但也许现实并不是这个样子

- 也许在 x1 小於某一个数值的时候,前一天的观看人数跟隔天的观看人数是成正比,
- 也许当 x1 大於一个数值的时候,这个物极必反,过了一个假设 x1 太大,前天观看的人数太高,那隔天观看人数就会变少,也说不定
- 也许 x1 跟 y 中间,有一个比较复杂的,像这个红色线一样的关係

但你不管怎麼摆弄w 跟 b,你永远製造不出红色那一条线,你永远无法用 Linear 的 Model,製造红色这一条线,显然 Linear 的 Model 有很大的限制,**这一种来自於 Model 的限制,叫做 ==Model 的 Bias==**,那其实我们刚才在课堂一开始的时候也叫做,也说 b 叫做 Bias,那这个地方有一点,在用词上有一点 Ambiguous,所以特别强调说,这个东西叫做 **Model 的 Bias,跟 b 的这个 Bias 不太一样**,它指的意思是说,没有办法模拟真实的状况

所以我们需要写一个更复杂的,更有弹性的,有未知参数的 Function,



我们可以观察一下红色的这一条曲线,它可以看作是**一个常数,再加上一群蓝色的这样子的 Function**,.

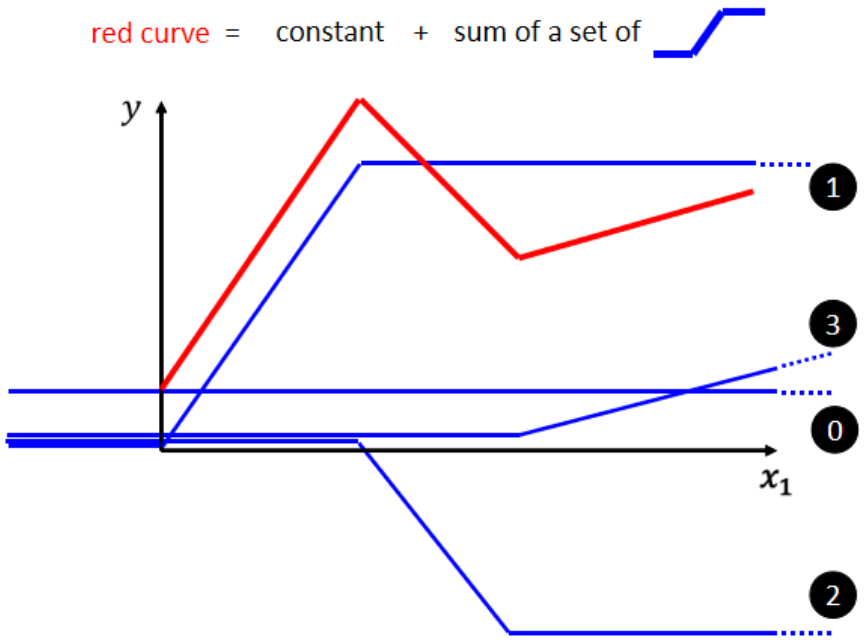


这个蓝色的 Function,它的特性是

- 当输入的值,当 x 轴的值小於某一个这个 Flash Hold 的时候,它是某一个定值,
- 大於另外一个 Flash Hold 的时候,又是另外一个定值,
- 中间有一个斜坡

所以它是先水平的,然后再斜坡,然后再水平的,那它其实有名字,它的名字我们等一下再讲,这边我们因為它是蓝色的 Function,我们就先叫它蓝方吧 这样子,好 那所以呢 这个红色的线啊,它可以看作是一个常数项加一大堆的蓝方,好 那这个**常数项**,它的值应该要有多大呢,你就看这一条**红色的线啊,它跟 x 轴的交点在哪裡**,好 那这个常数项呢,就设跟 x 轴的交点一样大

==那怎麼加上这个蓝色的 Function 以后,变成红色的这一条线?==

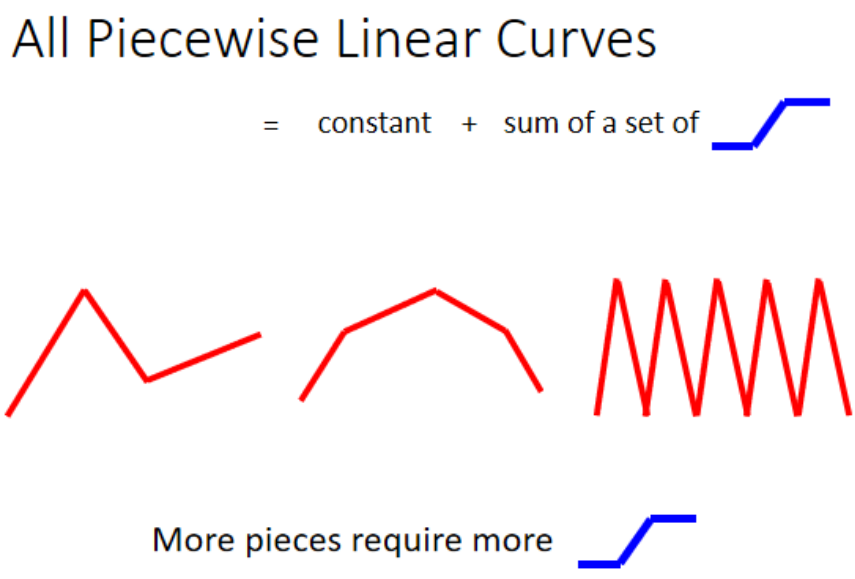


蓝线“1”Function 斜坡的**起点**,设在**红色 Function** 的**起始**的地方,然后第二个,**斜坡的终点**设在第一个转角处,你刻意让这边这个蓝色 Function 的斜坡,跟这个红色 Function 的斜坡,它们的**斜率是一样**的,这个时候如果你把 0 加上 1,你就可以得到红色曲线

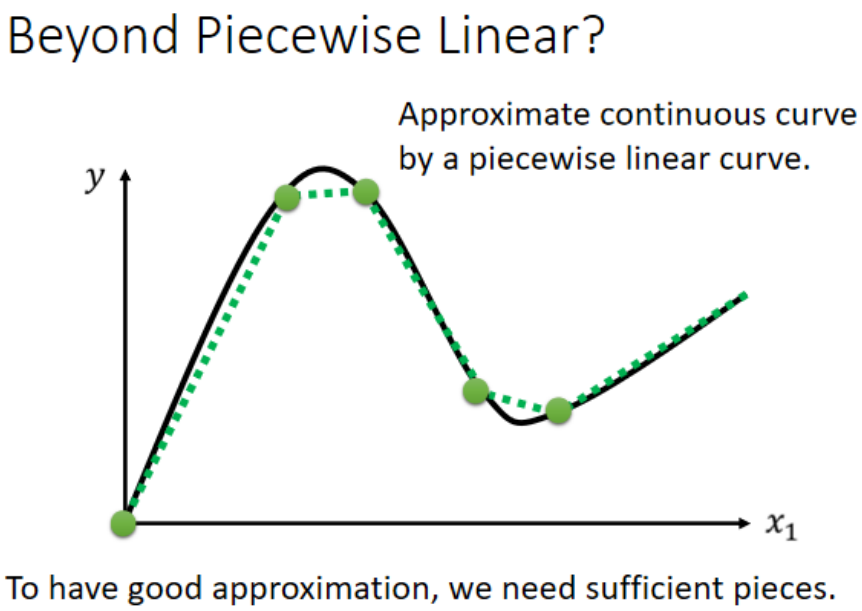
然后接下来,再加第二个蓝色的 Function,你就看红色这个线,**第二个转折点**出现在哪裡, 所以第二个**蓝色 Function**,它的斜坡就在红色 Function 的**第一个转折点,到第二个转折点之间**,你刻意让这边的**斜率跟这边的斜率一样**,这个时候你把 0加 1+2,你就可以得到两个转折点这边的线段,就可以得到红色的这一条线这边的部分

然后接下来第三个部分,第二个转折点之后的部分,你就加第三个蓝色的 Function,第三个蓝色的 Function,它这个坡度的起始点,故意设的跟这个**转折点一样**,这边的斜率,故意设的跟这边的**斜率一样**,好 接下来你把 0加 1+2+3 全部加起来,你就得到红色的这个线。

所以**红色这个线,可以看作是一个常数,再加上一堆蓝色的 Function**



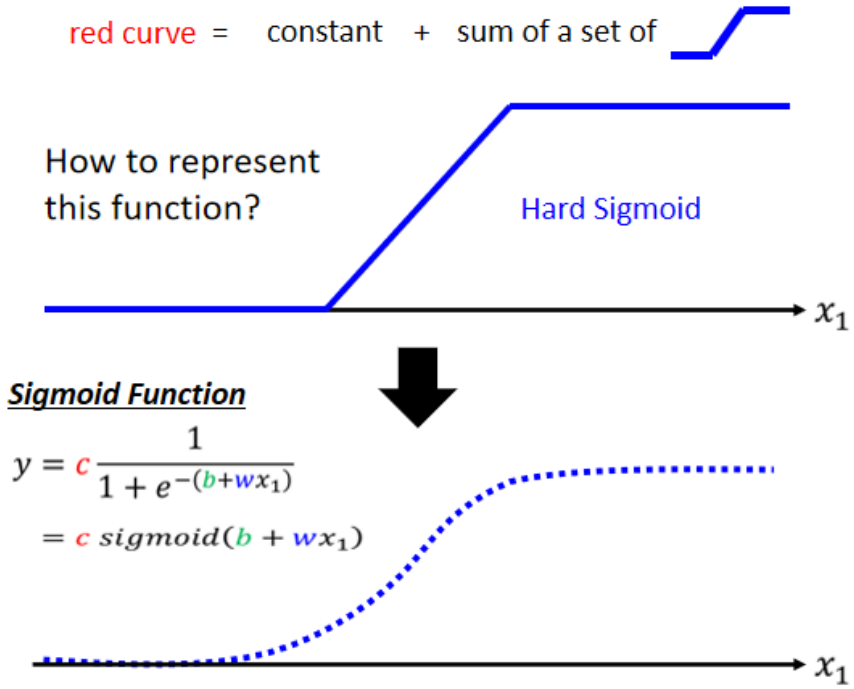
你现在这个 Curves 啊,它是有很多线段所组成的,它是有很多锯齿状的线段所组成的,这个叫做 **Piecewise Linear 的 Curves**,那你会发现说这些 Piecewise Linear 的 Curves,你有办法用常数项,加一大堆的蓝色 Function 组合出来,只是他们用的蓝色 Function 不见得一样,你要有很多不一样的蓝色 Function,加上一个常数以后,你就可以组出这些 Piecewise Linear 的 Curves。那如果你今天 Piecewise Linear 的 Curves 越复杂,也就是这个转折的点越多啊,那你需要的这个蓝色的 Function 就越多



讲到这边有人可能会说,那也许我们今天要考虑的 x 跟 y 的关係**不是 Piecewise Linear 的 Curves** ,也许它是这样子的曲线,那就算是这样的曲线,也无所谓,我们可以在这样的曲线上面,先取一些点,再把这些点点起来,变成一个 Piecewise Linear 的 Curves,而这个 Piecewise Linear 的 Curves 跟原来的曲线,它会非常接近,如果你今天点取的够多,或你点取的位置适当的话,这个 Piecewise Linear 的 Curves,就可以逼近这一个,连续的这一个曲线,就可以逼近这一个不是 Piecewise Linear,它是有角度的 有弧度的这一条曲线。

所以我们今天知道一件事情,==你可以用 Piecewise Linear 的 Curves,去逼近任何的连续的曲线,而每一个 Piecewise Linear 的 Curves,又都可以用一大堆蓝色的 Function 组合起来,也就是说,我只要有足够的蓝色 Function 把它加起来,我也许就可以变成任何连续的曲线==

所以今天,假设我们的 **x 跟 y 的关係,它也许非常地复杂**,那也没关係,我们就想办法写一个带有未知数的 Function,这个带有未知数的 Function 它表示的,就是一堆蓝色的 Function,加上一个 Constant,那我们接下来要问的问题就是,这一个蓝色 Function,它的式子应该要怎麼把它写出来呢?



也许你要直接写出它 没有那麼容易,但是你可以用一条曲线来理解它,用一个 Sigmoid 的 Function,来逼近这一个蓝色的 Function,那 Sigmoid Function,它的式子长的是这个样子的,

$$ y=c \frac{1}{1+e^{-(b+wx_1)}}

$$ 它的横轴输入是 x1,输出是 y,输入的 x1,我们先乘上一个 w,再加上一个 b,再取一个负号,再取 Exponential,再加 1,这一串被放在分放在,放在分母的地方

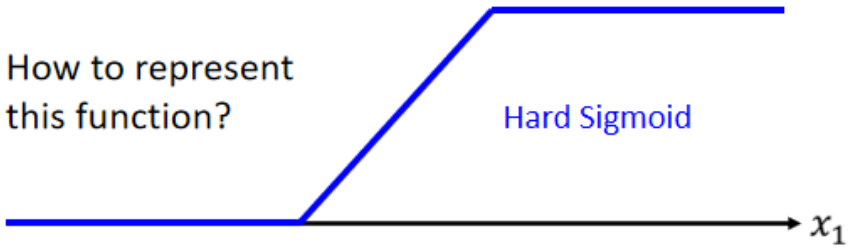
把 1 除以 1 加上 Exponential -b+wx1,前面 你可以乘上一个 Constant 叫做 c,好 那

- 如果你今天输入的这个 x1 的值,趋近於无穷大的时候,那 $e^{-(b+wx_1)}$这一项就会消失,那当 x1 非常大的时候,这一条这边就会收敛在这个高度是 c 的地方
- 那如果今天 x1 负的非常大的时候,分母的地方就会非常大,那 y 的值就会趋近於 0.

所以你可以用这样子的一个 Function逼近这一个蓝色的 Function,那这个东西它的名字叫做 Sigmoid,Sigmoid,如果你要 硬要翻成中文的话,可以翻成 S 型的,所以 Sigmoid Function 就是 S 型的 Function,因為它长得是 有点像是 S 型的哦,所以叫它 Sigmoid Function,那这边我们之后都懒得把 Exponential 写出来,我们就直接写成这个样子

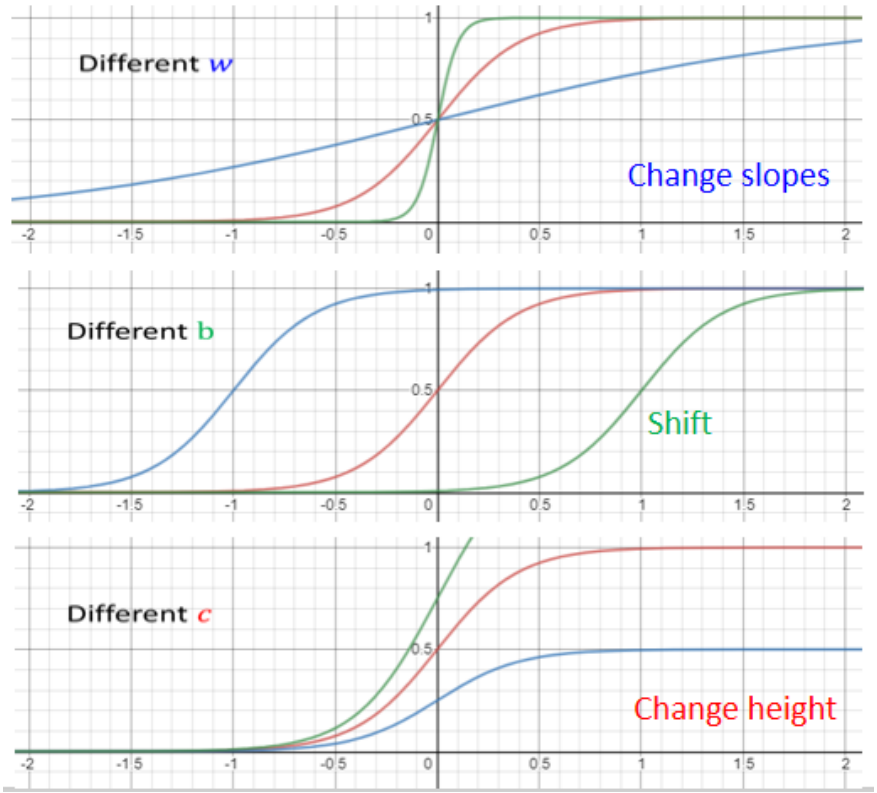
$$ y=c*sigmoid(b+wx_1)

$$ 就是 y 等於 c 倍的 Sigmoid,然后这个括号裡面放$b+wx_1$,然后这个 $b+wx_1$,实际上做的事情,就是把它放在 Exponential 的指数下,前面加一个负号,然后 1+Exponential 的$-(b+wx_1)$ 放在分母的地方,然后前面乘上 c,就等於 y



所以我们可以用这个 Sigmoid Function,去逼近一个蓝色的 Function,那其实这个蓝色的 Function,比较常见的名字就叫做,Hard 的 Sigmoid 啦,只是我本来是想说一开始,我们是先介绍蓝色的 Function,才介绍 Sigmoid,所以一开始说它叫做 Hard Sigmoid,有一点奇怪,所以我们先告诉你说,有一个 Sigmoid Function,它可以逼近这个蓝色的 Function,那这个蓝色的 Function,其实通常就叫做 Hard 的 Sigmoid

那我们今天我们需要各式各样不同的,蓝色的 Function,还记得吗,我们要**组出各种不同的曲线,那我们就需要各式各样合适的蓝色的 Function**,而这个合适的蓝色的 Function 怎麼製造出来呢



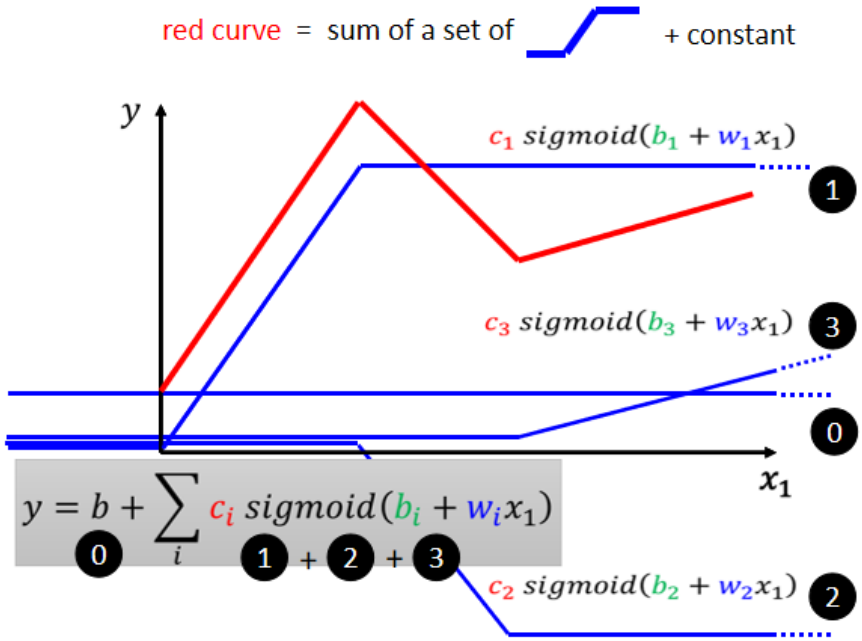
$$ y=c \frac{1}{1+e^{-(b+wx_1)}}

$$ 我们就需要调整这裡的 b 跟 w 跟 c,你就可以製造各种不同形状的 Sigmoid Function,用各种不同形状的 Sigmoid Function,去逼近这个蓝色的 Function.

- 如果你今天改 w 你就会改变**斜率**你就会改变斜坡的坡度
- 如果你动了 $b $ 你就可以把这一个 Sigmoid Function 左右移动
- 如果你改 c 你就可以改变它的高度

所以你只要有**不同的 w 不同的 b 不同的 c,你就可以製造出不同的 ==Sigmoid Function==**,把**不同的 Sigmoid Function 叠起来以后,你就可以去逼近各种不同的,Piecewise Linear 的 Function**,然后 Piecewise Linear 的 Function,可以拿来近

似各种不同的 Continuous 的 Function



所以假设我们要把红色的这条线,它的函数写出来的话,那可能长什麼样子呢?

我们知道说**红色这条线** 就是 0加 1+2+3,而这个 123 啊,它们都是蓝色的 Function,所以它们的函式 就是有一个固定的样子,它们都写做 $(b+wx_1)$,去做 Sigmoid 再乘上 c1,只是 1 跟 2 跟 3,它们的 w 不一样,它们的 b 不一样,它们的 c 不一样,如果是第一个蓝色 Function,它就是 w1 b1 c1,第二个蓝色 Function,我们就说它的,它用的是 w2 b2 c2,第三个蓝色 Function,我们就说它用的是 w3 b3 c3,好 那我们接下来呢,就是把 0 跟 123 全部加起来以后,我们得到的函式,就长这一个样子

$$ y = b + \sum_i {c_isigmoid(b_i+w_ix_1)}

$$ 我们把 1+2+3 加起来,然后 Summation 裡面呢,就是 ci 乘上 Sigmoid,bi+wi 乘上 x1,所以这边**每一个式子,都代表了一个不同蓝色的 Function,Summation 的意思,就是把不同的蓝色的 Function 给它加起来**,就是这边 Summation 的意思.

别忘了加一个 Constant,这边用 b 来表示这个 Constant

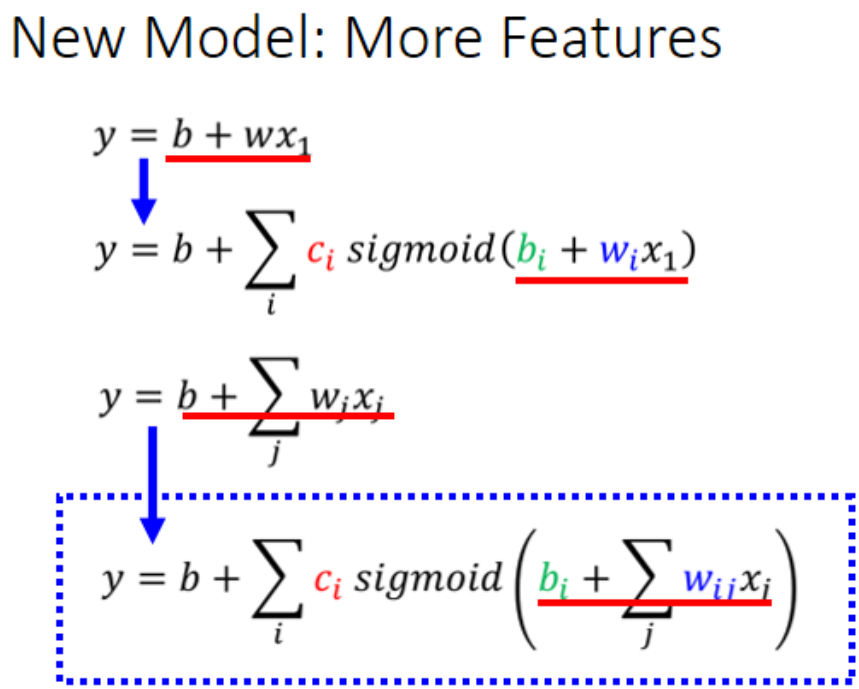
所以我们今天就写出了一个这样子的 Function,如果我们假设裡面的 b 跟 w 跟 c,它是未知的,它是我们未知的参数,那我们就可以设定不同的 b 跟 w 跟 c,设定不同的 b 跟 w 跟 c,我们就可以製造不同的蓝色的 Function,製造不同的蓝色的 Function 叠起来以后,就可以製造出不同的红色的 Curves,製造出不同的红色的 Curves,就可以製造出不同的 Piecewise Linear 的 Curves,就可以去逼近,各式各样不同的 Continuous 的 Function

$$y = b + wx_1$$

$$y = b + \sum_i c_i\ sigmoid(b_i + w_ix_1)$$

所以我们其实有办法写出一个,这个非常有弹性的,有未知参数的 Function,它长这个样子就是 Summation 一堆 Sigmoid,但它们有不同的 c 不同的 b 不同的 w,好 那所以本来我们是 Linear 的 Model,y 等於 b+w 乘上 x1,它有非常大的限制,这个限制叫做 Model 的 Bias,那我们要**如何减少 Model 的 Bias** 呢

我们可以写一个更有弹性的,有未知参数的 Function,它叫做 $y=b + \sum_i {c_isigmoid(b_i+w_ix_1)}$ 本来这边是 $b+wx_1$,这边变成$b_i+w_ix_1$,然后我们有很多不同的 bi,有很多不同的 wi,它们都通过 Sigmoid 都乘上 ci,把它统统加起来再加 b 等於 y,我们只要带入不同的 c 不同的 b 不同的 w,我们就可以变出各式各样,就可以组合出各式各样不同的 Function



那我们刚才其实已经进化到,不是只用一个 Feature,即X_1,我们可以**用多个 Feature**

我们这边用 **==j 来代表 Feature 的编号==**

举例来说刚才如果要考虑前 28 天的话,j 就是 1 到 28,考虑前 56 天的话,j 就是 1 到 56,那如果把这个 Function,再扩展成我们刚才讲的上面这个,比较有弹性的 Function 的话那也很简单,我们就把 Sigmoid 裡面的东西换掉,本来这边是

$$ y=b + \sum_j {w_ix_j }

$$ 那这边呢,就把这一项放到这个括号裡面,改成

$$ y = b + \sum_i {c_isigmoid(b_i+\sum_j w_i{_j} x_j)}

$$ 把本来放在这边的东西放到 Sigmoid 裡面,然后呢这个每一个 Sigmoid 的 Function 裡面呢,都有不同的 bi 不同的 $w_i{_j}$,然后取 Sigmoid 以后乘上 ci 就全部加起来,再加上 b 就得到 y,我们只要这边 ci bi 跟 $w_i{_j}$ 在放不同的值,就可以变成不同的 Function.

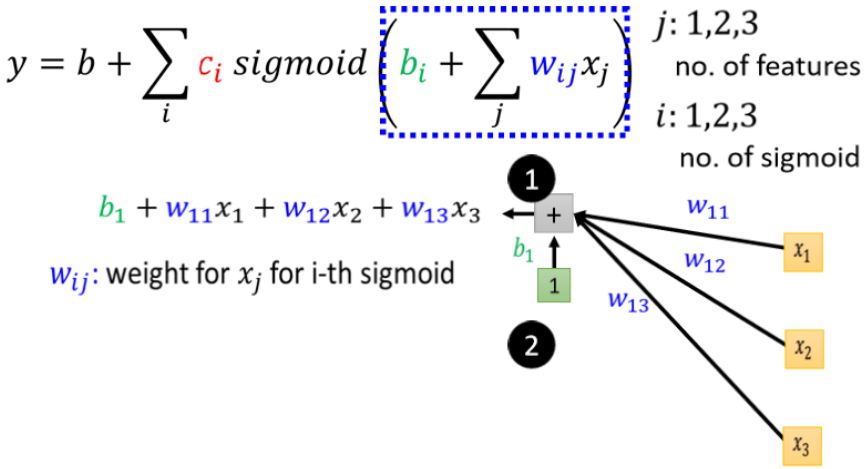
那如果讲到这边你还是觉得有点抽象的话,如果你看这个式子觉得有点头痛的话,那我们用比较直观的方式,把这个式子实际上做的事把它画出来,

我们先考虑一下 j 就是 1 2 3 的状况,就是我们**只考虑三个 Feature**

举例来说 我们只考虑前一天 前两天,跟前三天的 Case,

- 所以 j 等於 1 2 3,那所以输入就是 **x1 代表前一天的观看人数,x2 两天前观看人数,x3 三天前的观看人数**
- 每一个 **i 就代表了一个蓝色的 Function**,只是我们现在每一个蓝色的 Function,都用一个 Sigmoid Function 来比近似它

那这边呢,这个 1 2 3 就代表我们有三个 Sigmoid Function,那我们先来看一下,这个**括号裡面**做的事情是什麼



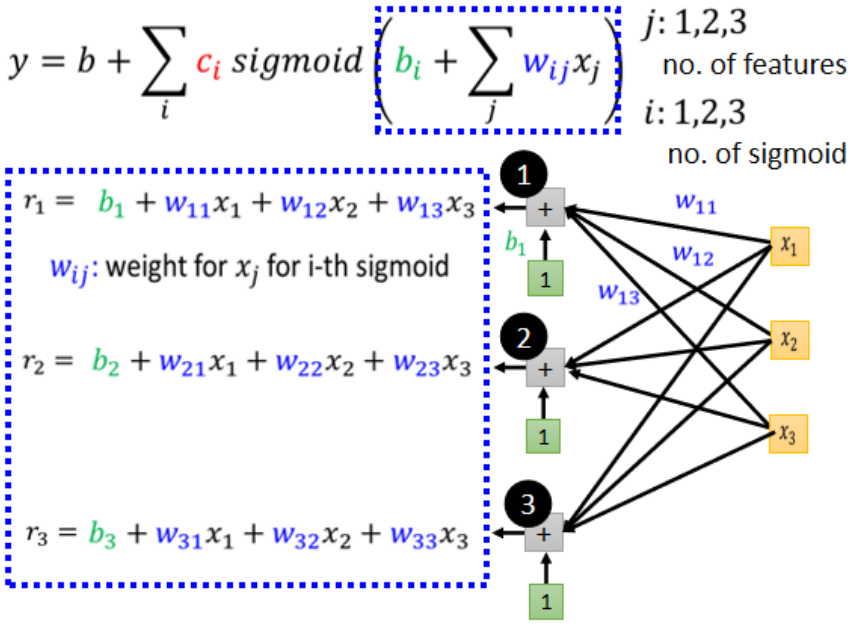
每一个 Sigmoid 都有一个括号,第一个 Sigmoid i 等於 1 的 Case ,就是把

- x1 乘上一个 Weight 叫 w_{11}
- x2 乘上另外一个 Weight 叫 w_{12}
- x3 再乘上一个 Weight 叫做 w_{13}
- 全部把它**加起来**,不要忘了再加一个 b

$$ b*1*+*w*{11} x*1*+*w*{12} x*2*+*w*{13} x_3

$$

这个得到的式子就是这个样子,所以这边我们**用 w_{ij} ,来代表在第 i 个 Sigmoid 裡面,乘给第 j 个 Feature 的 Weight**,第一个 Feature 它就是 w11,第二个 Features 就是乘 w12,第三个 Feature 都是乘 w13,所以三个 Features1 2 3,这个 w 的第二个下标就是 123,w 的第一个下标代表是,现在在考虑的是第一个 Sigmoid Function,那我们有三个 Sigmoid Function,



第二个 Sigmoid Function,它在括号裡面做的事情就是把 x1 乘上 w21,把 x2 x2 乘上 w22,把 x3 x3 乘上 w23,统统加起来再加 b2

第三个 Sigmoid 呢,第三个 Sigmoid 在括号裡面做的事情,就是把 x1 x2 x3,分别乘上 w31 w32 跟 w33 再加上 b3

我们现在為了**簡化**起见,我们把括弧裡面的数字,用一个比较简单的符号来表示,所以这一串东西我们当作 r1,这一串东西我们当作 r2,这一串东西我们叫它 r3.

$$r_1 = b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

$$r_2 = b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

$$r_3 = b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3$$

这个 x1 x2 跟 x3 和 r1 r2 r3,中间的关係是什麼呢,你可以用矩阵跟向量相乘的方法,写一个比较简单的 简洁的写法.我们刚才已经知道说 r1 r2 r3,也就是括弧裡面算完的结果啊,三个 Sigmoid 括弧裡面算完的结果,r1 r2 r3 跟输入的三个 Feature x1 x2 x3,它们中间的关係就是这样,把 x1 x2 x3 乘上不同的 Weight,加上不同的 Bias,也就是不同的 b 会得到不同的 r.

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

如果你熟悉线性代数的话,简化成矩阵跟向量的相乘,把 x1 x2 x3 拼在一起变成一个向量,把这边所有的 w 统统放在一起变成一个矩阵,把 b1 b2 b3 拼起来变成一个向量,把 r1 r2 r3 拼起来变成一个向量,那这是三个式子,你就可以简写成,有一个向量叫做 x,这个 x 乘 1个矩阵叫做 w,这个 w 裡面有 9 个数值就是这边的 9 个 w,就是这边的 9 个 Weight,x 先乘上 w 以后再加上 b 就得到 r 这个向量,那**这边做的事情跟上边做的事情是一模一样**的,没有半毛钱的不同,只是表示的方式不一样而已,

$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right) \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$

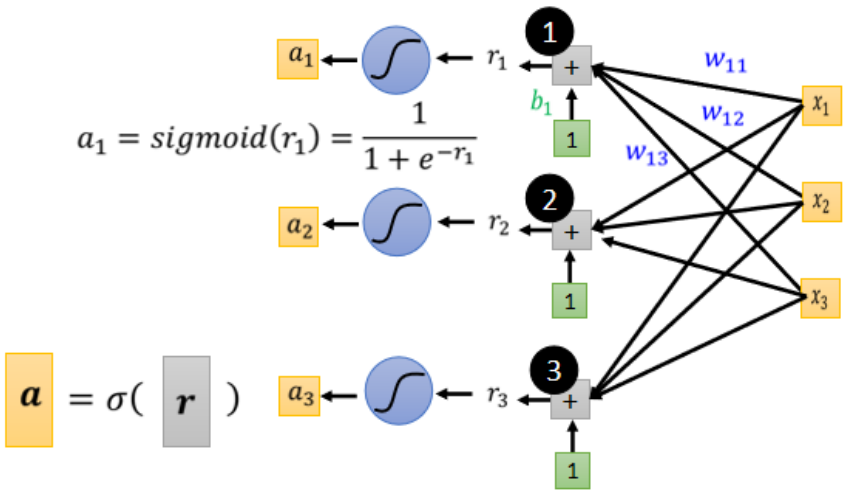
$$\begin{aligned} r_1 &= b_1 + w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ r_2 &= b_2 + w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ r_3 &= b_3 + w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{aligned}$$

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

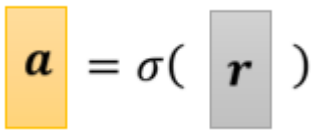
$$\boldsymbol{r} = \boldsymbol{b} + W \boldsymbol{x}$$

那把它改成线性代数比较常用的表示方式,x 乘上矩阵 w 再加上向量 b,会得到一个向量叫做 r

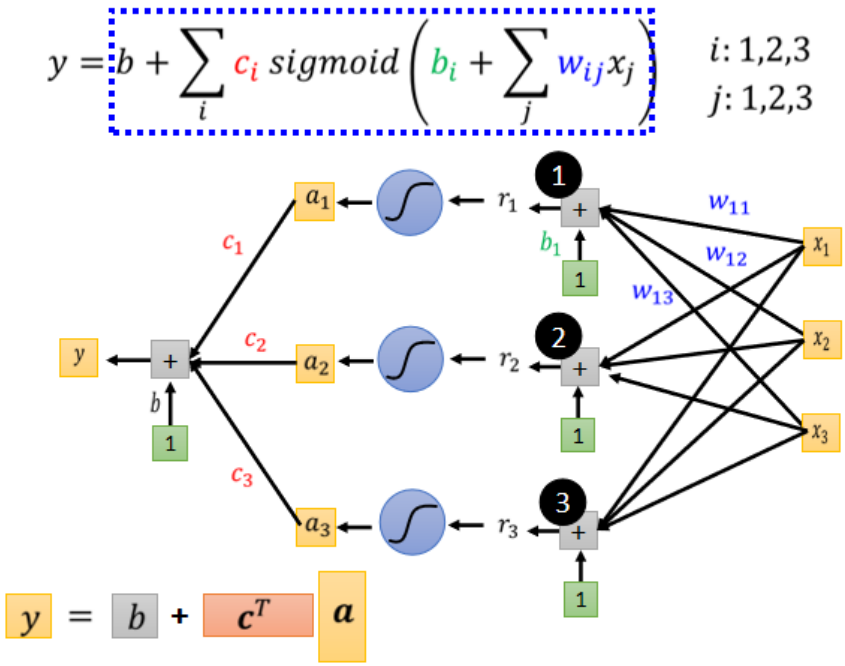
$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right) \qquad \begin{matrix} i: 1,2,3 \\ j: 1,2,3 \end{matrix}$$



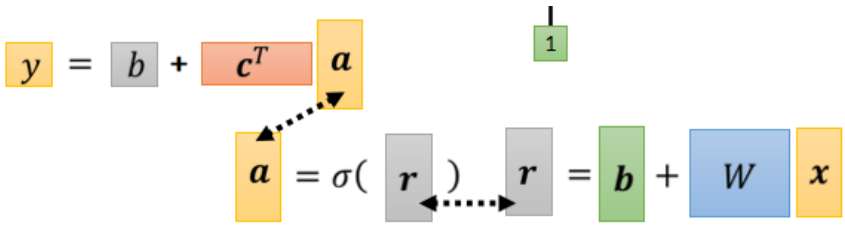
在这个括号裡面做的事情就是这麼一回事,把 x 乘上 w 加上 b 等於 r,r 呢就是这边的 r1 r2 r3,这是 r1 r2 r3,好 那接下来这个 r1 r2 r3 哪,就要分别通过 Sigmoid Function,好 分别通过 Sigmoid Function,因為我们实际上做的值就是,做的事情就是把 r1 取一个负号,再乘 再做 Exponential 再加 1,然后把它放到分母的地方,1 除以 1+Exponential 负 r1 等於 a1,然后同样的方法由 r2 去得到 a2,把 r3 透过 Sigmoid Function 得到 a3,所以这边这个蓝色的虚线框框裡面做的事情,就是从 x1 x2 x3 得到了 a1 a2 a3,



接下来呢,我们这边呢有一个简洁的表示方法,是我们用 r 通过一个,叫做这个 Sigmoid 的 Function,我们用这个东西,我们这边呢用这个符号呢,来代表通过这个 Sigmoid 的 Function,然后呢 所以我们得到了 a 这个向量,就把 r1 r2 r3 分别通过 Sigmoid Function,但我们直接用这个符号来表示它,然后得到 a1 a2 a3



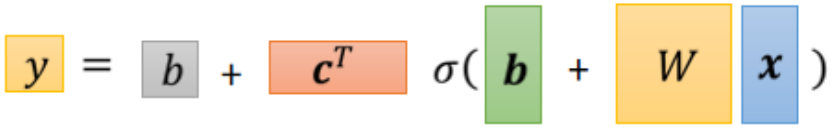
接下来我们这个 Sigmoid 的输出,还要乘上 ci 然后还要再加上 b,如果你要用向量来表示的话,a1 a2 a3 拼起来叫这个向量 a,c1 c2 c3 拼起来叫一个向量 c,那我们可以把这个 c 呢,作 **Transpose**,好 那 a 呢 乘上 c 的 Transpose 再加上 b,好再加上 b 我们就得到了 y



它整体而言做的事情就是 x 输入是 x,我们的 Feature 是 x 这个向量,x 乘上矩阵 w 加上向量 b 得到向量 r,再把向量 r 透过 Sigmoid Function得 到向量 a,再把向量 a 跟乘上 c 的 Transpose 加上 b 就得到 y。

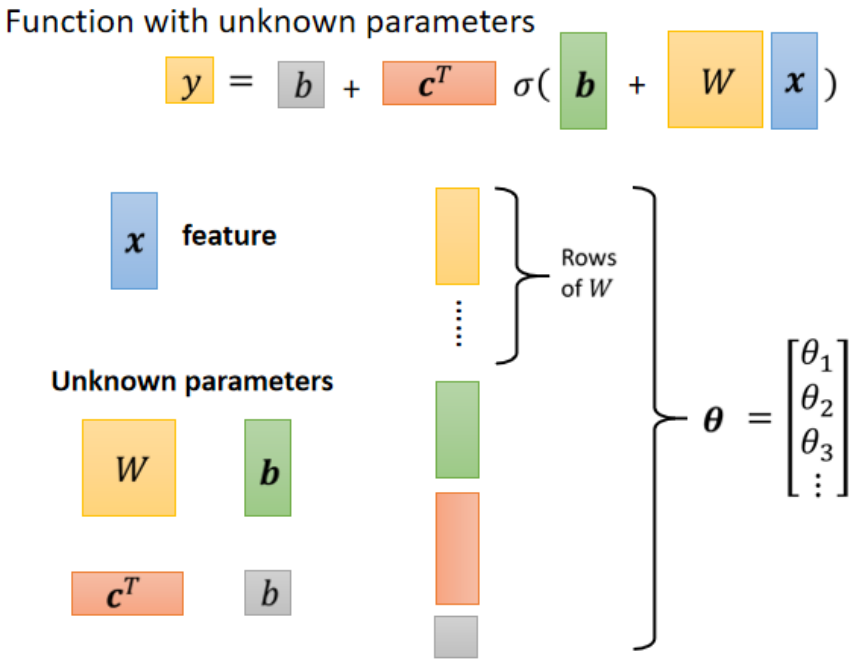
所以这是上面这件事情,如果你想要用**线性代数**的方法来表示它,用向量矩阵相乘方法来表示它,欸 就长得一副这个样子,那这边的这个 r 就是这边的 r,这边的 a 就这边的 a,所以我们可以把这一串东西,放到这个括号裡面,再把这个 a 呢 放到这裡来,所

以把相同的东西併起来以后,整体而言就是长这个样子,上面这一串东西,我们觉得比较这个,比较有弹性的这个 Function,如果你要线性代数来表示它的话,就是下面这个式子啦



x 乘上 w 再加上 b 通过 Sigmoid Function,乘上 c 的 Transpose 加 b 就得到 y

接下来，怎麼把这些未知的参数找出来之前,我们先再稍微**重新定义一下我们的符号**,



这边的这个 ==x 是 Feature==,这边的 W b c 跟 b,这边有两个 b 啊,但是**这两个 b 是不一样**的,**绿色**这一个是一个**向量**,**灰色**这个是一个**数值**,显示它们是不一样的东西

我们把这个黄色的这个 w,把这个 b 把这个 c 把这个 b 统统拿出来,集合在这边,它们就是我们的 Unknow 的 Parameters,就是我们的未知的参数

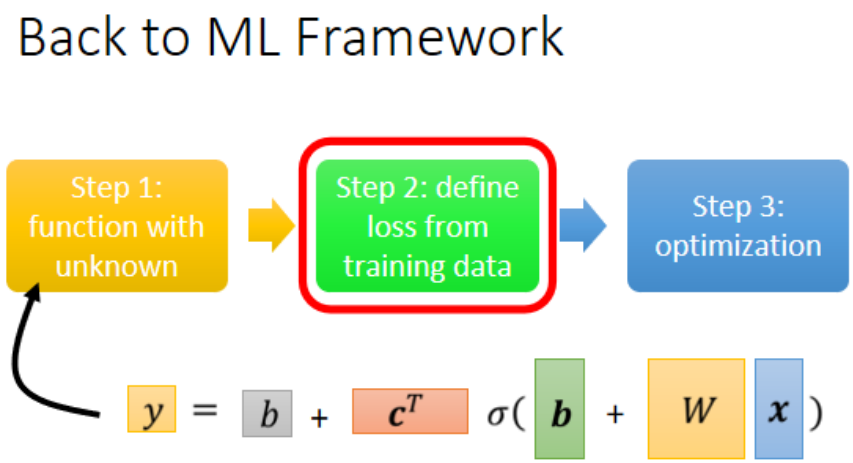
那我们把**这些东西通通拉直**,**拼成一个很长的向量**,我们把 w 的每一个 Row,或者是每一个 Column 拿出来,今天不管你是拿过 Row 或拿 Column 都可以,你就把 w 的每一个 Column 或每一个 Row 拿出来,拼成一个长的向量,把 b 拼上来 把 c 拼上来把 b 拼上来,这个长的向量,我们直接用一个符号叫做 $θ$ 来表示它

$θ$ 是一个很长的向量,裡面的第一个数值我们叫 $θ1$,第二个叫 $θ2$ 这个叫 $θ3$,那 $θ $裡面,这个向量裡面有一些数值是来自於这个**矩阵**,有些数值是来自於 b,有些数值来自於 c,有些数值来自於这边这个 b,那我们就不分了,反正**$ θ $它统称我们所有的未知的参数,我们就一律统称 $θ$**

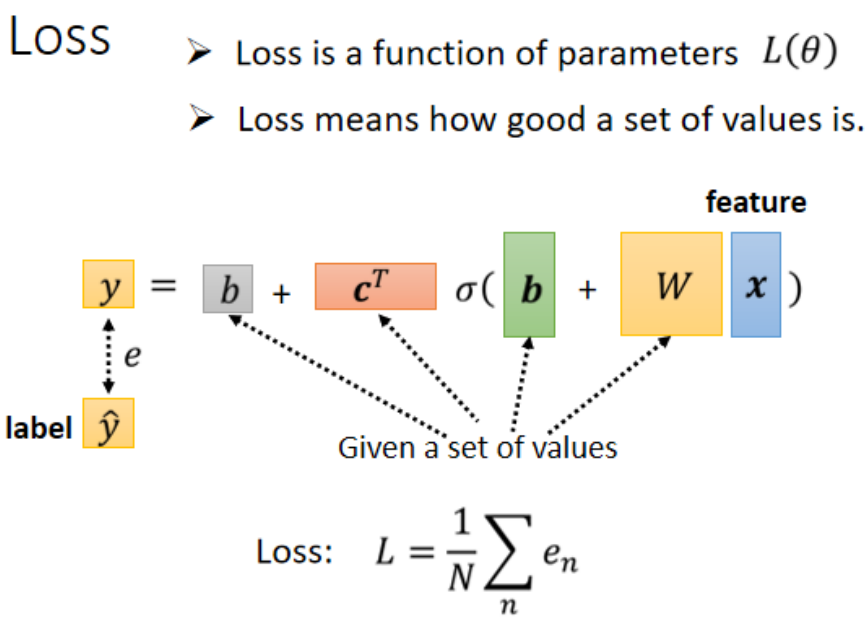
Q&A

1. 我试著回答看看,我猜他的问题是说,我们其实要做 Optimization 这件事,找一个可以让 Loss 最小的参数,有一个最**暴力的方法**就是,爆收所有可能的未知参数的值对不对,像我们刚才在只有 w 跟 b 两个参数的前提之下,我根本就可以爆收所有可能的 w 跟 b 的值嘛,所以在**参数很少**的情况下,你不 甚至你有可能不用 Gradient Descent,不需要什麼 Optimization 的技巧,但是我们今天参数很快就会变得非常多,像在这个例子裡面参数有一大把,有 w b 有 c 跟 b 串起来,变成一个很长的向量叫 θ,那这个时候你就不能够用爆收的方法了,你需要 Gradient Descent 这样的方法,来找出可以让 Loss 最低的参数
2. 这位同学的问题是说,刚才的例子裡面有三个 Sigmoid,那為什麼是三个呢,能不能够四个 五个 六个呢,可以 Sigmoid 的数目是你自己决定的,而且 Sigmoid 的数目越多,你可以產生出来的,Piecewise Linear 的 Function 就越复杂,就是假设你只有三个 Sigmoid,意味著你只能產生三个线段,但是假设你有越多 Sigmoid,你就可以產生有越多段线的,Piecewise Linear 的 Function,你就可以逼近越复杂的 Function,但是至於要几个 Sigmoid,这个又是另外一个 Hyper Parameter,这个你要自己决定,我们在刚才例子裡面举三个,那只是一个例子,也许我以后不应该举三个,因為这样会让你误以為说,Input Feature 是三个,Sigmoid也是三个,不是就是说,Sigmoid 几个可以自己决定
3. Hard 的 Sigmoid,首先它的 Function 你写出来可能会比较复杂,你一下子写不出它的 Function,但如果你可以写得出它的 Function 的话,你其实也可以用 Hard Sigmoid,你想要用也可以,所以不是一定只能够用,刚才那个 Sigmoid 去逼近那个 Hard Sigmoid,完全有别的做法,等一下我们就会讲别的做法

Back to ML_Step 2 :define loss from training data



那接下来进入第二步了,我们要定 Loss,有了新的这个 Model 以后,我们 Loss 没有什麼不同,定义的方法是一样的,只是我们的符号改了一下,之前是 L (w 跟 b),因為 w 跟 b 是未知的,那我们现在接下来的未知的参数很多了,你再把它一个一个列出来,太累了,所以我们直接**用 θ 来统设所有的参数**,所以我们现在的 Loss Function 就变成 $L(θ)$

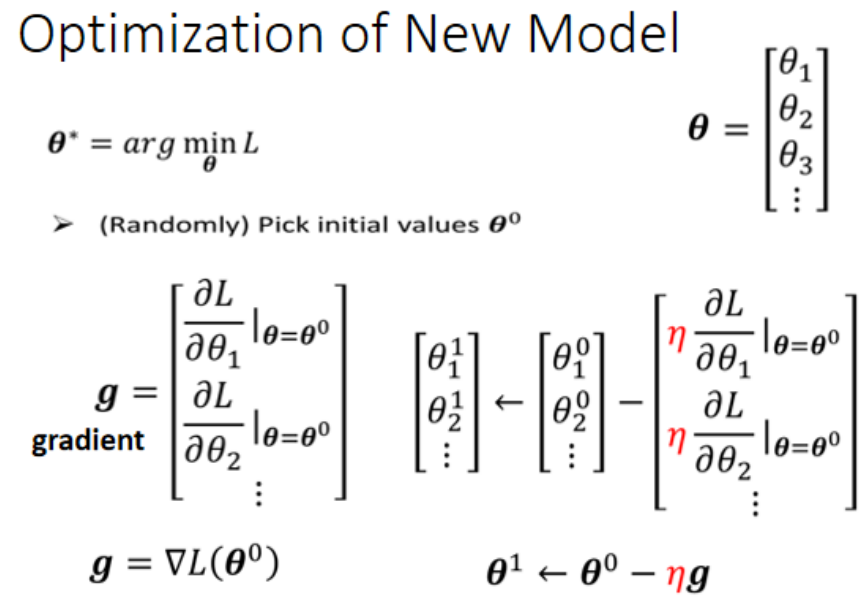


这个 Loss Function 要问的就是,这个 θ 如果它是某一组数值的话,会有多不好或有多好,那计算的方法,**跟刚才只有两个参数的时候,其实是一模一样的**

- 先给定某一组 W b c^T 跟 b 的值,你先给定某一组 $θ $的值,假设你知道 w 的值是多少,把 w 的值写进去 b 的值写进去,c 的值写进去 ,b 的值写进去
- 然后把一种 Feature x 带进去,然后看看你估测出来的 y 是多少
- 再计算一下跟真实的 Label 之间的差距,你得到一个 e
- 把所有的误差通通加起来,你就得到你的 Loss

Back to ML_Step 3: Optimization

接下来下一步就是 Optimization,Optimization跟前面讲的没有什麼不同 还是一样的,所以就算我们换了一个新的模型,这个 Optimization 的步骤.



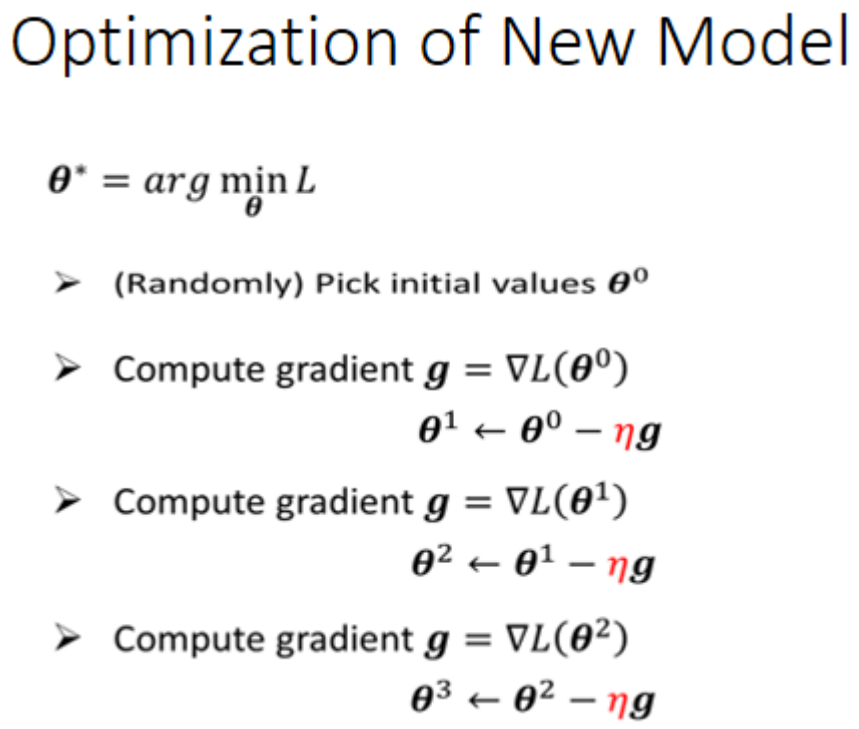
我们现在的 **θ 它是一个很长的向量**,我们把它表示成 θ1 θ2 θ3 等等等,我们现在就是要==**找一组 θ,这个 θ 可以让我们的 Loss 越小越好**==,可以让 Loss 最小的那一组 θ,我们叫做 θ 的 Start $θ^*$

- 我们一开始要**随机选一个初始的数值,这边叫做 $θ_0$** 你可以随机选,那之后也可能会讲,也会讲到更好的找初始值的方法,我们现在先随机选就好
- 接下来呢你要**计算微分**,你要对每一个未知的参数,这边用 θ1 θ2 θ3 来表示,你要**为每一个未知的参数,都去计算它对 L 的微分**,那把每一个参数都拿去计算对 L 的微分以后,集合起来它就是一个向量,那个向量我们用 g 来表示它,这边假设有 1000 个参数,这个向量的长度就是 1000,这个向量裡面就有 1000 个数字,这个东西有一个名字,,这个向量有一个名字叫做 ==Gradient==,那很多时候你会看到,Gradient 的表示方法是这个样子的,你把 L 前面放了一个**倒三角形**,这个就代表了 Gradient,这是一个 Gradient 的简写的方法,那其实我要表示的就是这个向量,L 前面放一个倒三角形的意思就是,把所有的参数 θ1 θ2 θ3,通通拿去对 L 作微分.那后面放 θ0 的意思是说,我们这个算微分的位置,是在 θ 等於 θ0 的地方,在 θ 等於 θ0 的地方,我们算出这个 Gradient
- 算出这个 g 以后,接下来呢我们 **Update 参数**,更新的方法,跟刚才只有两个参数的状况是一模一样的,只是从更新两个参数,可能换成更新成 1000 个参数,但更新的方法是一样的,本来有一个参数叫 θ1,**上标 0** 代表它是一个**起始的值**,它是一个随机选的起始的值,把这个 **$θ_1^0$ 减掉 η 乘上微分的值,得到 $θ_1^1$,代表 θ1 更新过一次的结果**,$θ_2^0$ 减掉微分乘以,减掉 η 乘上微分的值,得到 $θ_2^1$,以此类推,你就可以把那 1000 个参数统统都更新了.

简写把这边**所有的 θ 合起来当做一个向量,我们用 $θ^0 $来表示**,把 η 提出来,那剩下**每一个参数对 L 微分的部分**,叫做 **Gradient 叫做 g**,所以 θ0 减掉 η 乘上 g,就得到 θ1

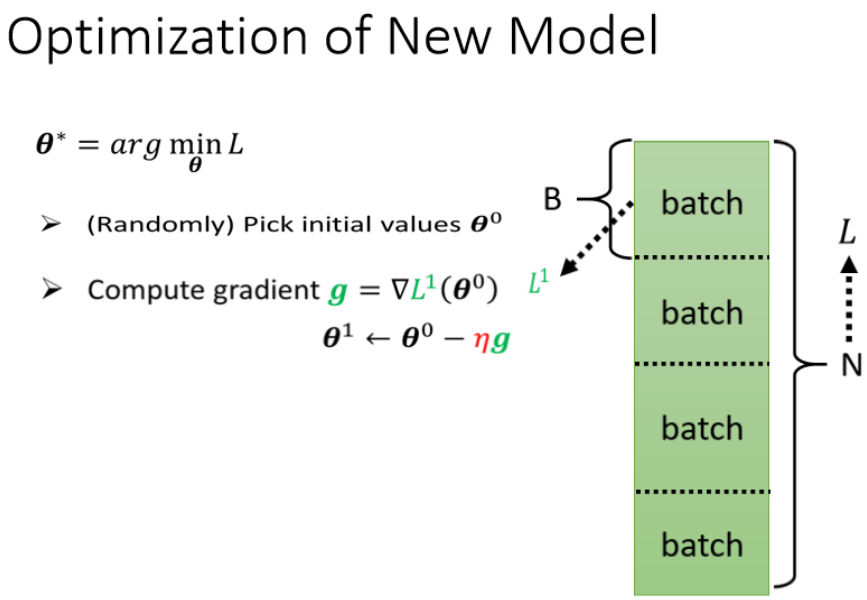
$$ θ^1 ← θ^0-ηg

$$



θ0 减掉 θ0 这个向量,减掉 η 乘上 g,g 也是一个向量会得到 θ1,那假设你这边参数有 1000 个,那 θ0 就是 1000 个数值,1000 微的向量,g 是1000 微的向量,θ1 也是 1000 微的向量

那整个操作就是这样了,就是由 θ0 算 Gradient,根据 Gradient 去把 θ0 更新成 θ1,然后呢再算一次 Gradient,然后呢根据 Gradient 把 θ1 再更新成 θ2,再算一次 Gradient 把 θ2 更新成 θ3,以此类推直到你不想做,或者是你算出来的这个 Gradient,是 0 向量 是 Zero Vector,导致你没有办法再更新参数为止,不过在实作上你几乎不太可能,作出 Gradient 是 0 向量的结果,通常你会停下来就是你不想做了.

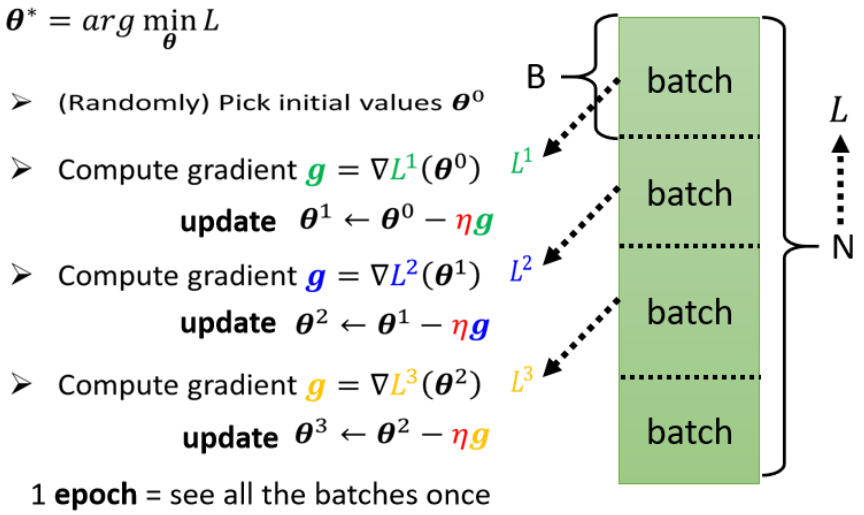


但是实作上,那这边是一个实作的 Detail 的 Issue,实际上我们在做 Gradient的时候,我们会这麼做

我们这边有大 N 笔资料,我们会把这大 N 笔资料分成一个一个的 ==Batch==,就是一包一包的东西 一组一组的,怎麼分,随机分就好

所以每个 Batch 裡面有大 B 笔资料,所以本来全部有大 N 笔资料,现在大 B 笔资料一组,**一组叫做 Batch**

那**本来我们是把所有的 Data 拿出来算一个 Loss**,那现在我们不这麼做,我们**只拿一个 Batch 裡面的 Data出来算一个 Loss**,我们这边把它叫 L1,那跟这个 L 呢以示区别,因為你把全部的资料拿出来算 Loss,跟只拿一个 Batch 拿出来,的资料拿出来算 Loss,它不会一样嘛,所以这边用 L1 来表示它

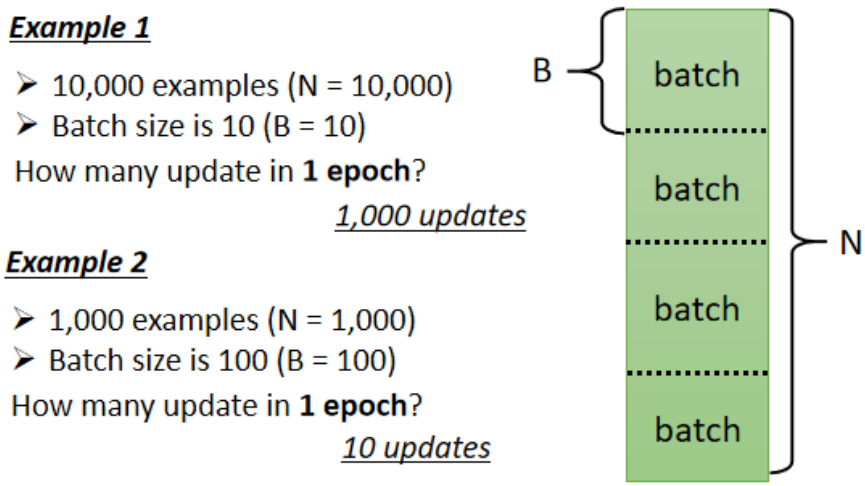


但是你可以想像说假设这个 B 够大,也许 L 跟 L1 会很接近 也说不定,所以实作上的时候,每次我们会先选一个 Batch,用这个 Batch 来算 L,**根据这个 L1 来算 Gradient,用这个 Gradient 来更新参数**,接下来再选下一个 Batch 算出 L2,根据 L2 算出 Gradient,然后再更新参数,再取下一个 Batch 算出 L3,根据 L3 算出 Gradient,再用 L3 算出来的 Gradient 来更新参数

所以我们并不是拿大 L 来算 Gradient,实际上我们是拿一个 Batch 算出来的 L1 L2 L3,来计算 Gradient,那把所有的 Batch 都看过一次,叫做一个 ==Epoch==,**每一次更新参数叫做==一次 Update==**, Update 跟 Epoch 是不一样的东西

每次更新一次参数叫做一次 Update,把所有的 Batch 都看过一遍,叫做一个 Epoch

那至於為什麼要分一个一个 Batch,那这个我们下週再讲,但是為了让大家更清楚认识,Update 跟 Epoch 的差别,这边就举一个例子



假设我们有 10000 笔 Data,也就是**大 N 等於 10000**,假设我们的 **Batch 的大小是设 10**,也就大 B 等於10

接下来问,我们在一个 Epoch 中,总共 Update 了几次参数?

那你就算一下这个大 N 个 Example,10000 笔 Example,总共形成了 10000 除以 10,也就是 **1000 个 Batch**,所以在一个 Epoch 裡面,你其实已经**更新了参数 1000 次**,所以一个 Epoch 并不是更新参数一次,在这个例子裡面一个 Epoch,已经更新了参数 1000 次了

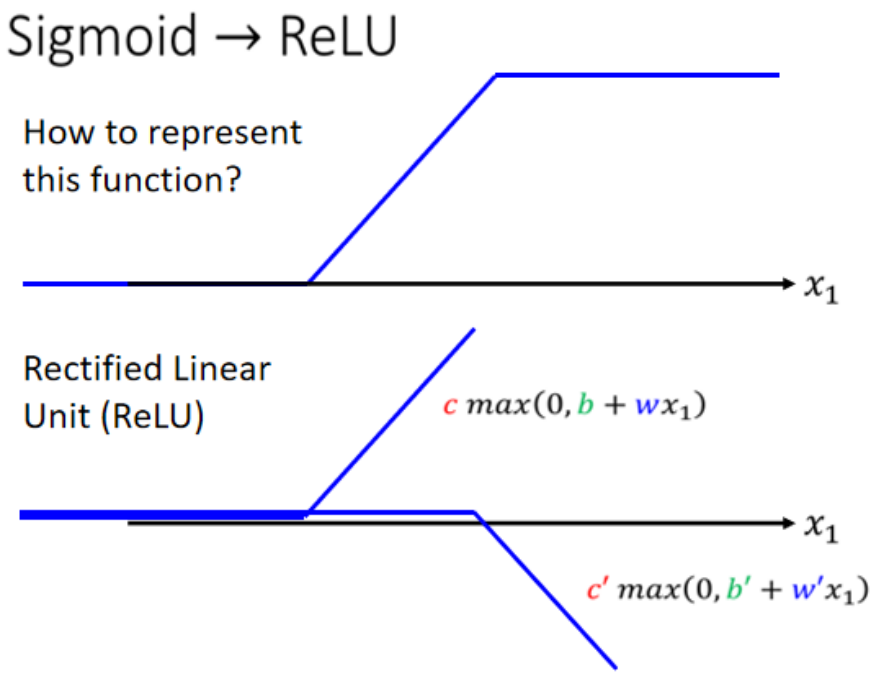
那第二个例子,就是假设有 1000 个资料,==Batch Size== 设 100,那其实 Batch Size 的大小也是你自己决定的,所以这边我们又多了一个 HyperParameter,所谓 HyperParameter 就是你自己决定的东西,人所设的东西不是.机器自己找出来的,叫做 HyperParameter,我们今天已经听到了,几个 Sigmoid 也是一个 HyperParameters,Batch Size 也是一个 HyperParameter,好 1000 个 Example,Batch Size 设 100,那1个 Epoch 总共更新几次参数呢,是 10 次

所以做了一个 Epoch 的训练,你其实不知道它更新了几次参数,有可能 1000 次,也有可能 10 次,**取决於它的 Batch Size 有多大**

模型变型

那我们其实还可以对模型做更多的变形,刚才有同学问到说,这个 Hard Sigmoid 不好吗,為什麼我们一定要把它换成 Soft 的 Sigmoid

你确实可以不一定要换成 Soft 的 Sigmoid,有其他的做法,举例来说这个 Hard 的 Sigmoid,我刚才说它的函式有点难写出来,其实也没有那麼难写出来,它可以看作是**两个 Rectified Linear Unit 的加总**,所谓 Rectified Linear Unit 它就是长这个样

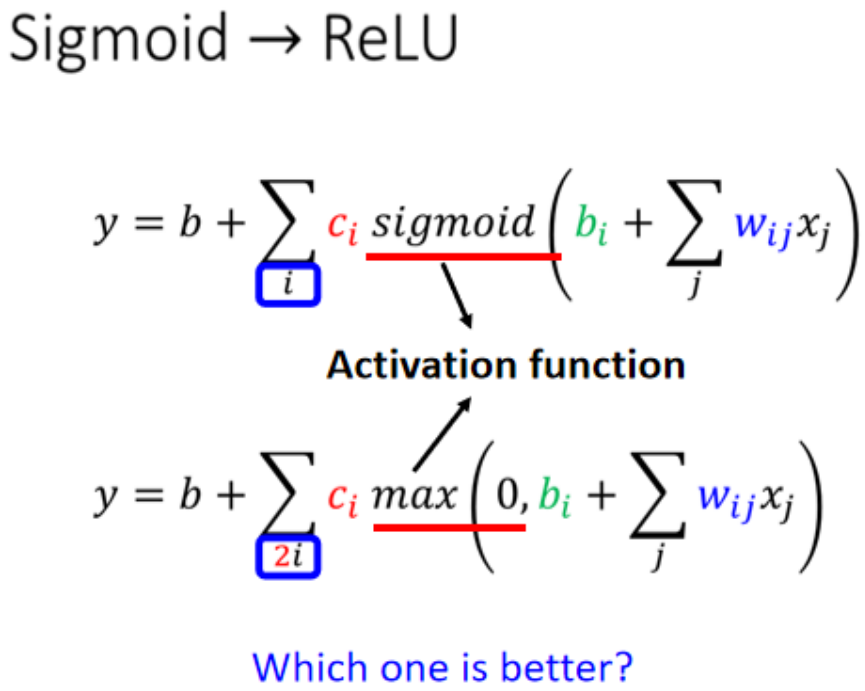


它有一个水平的线,走到某个地方有一个转折的点,然后变成一个斜坡,那这种 Function 它的式子,写成

$$ c* max(0, b + wx_1)

$$ 这个$max(0, b + wx_1)$的意思就是,看 0 跟 $b + wx_1$ 谁比较大,**比较大的那一个就会被当做输出**,所以如果 b + wx1 小於 0,那输出就是0,如果 $b + wx_1$ 大於 0,输出就是 $b + wx_1$

那总之这一条线,可以写成$c *max(0, b + wx_1)$,每条不同的 w 不同的 b 不同的 c,你就可以挪动它的位置,你就可以改变这条线的斜率,那这种线呢在机器学习裡面,我们叫做 *Rectified Linear Unit*,它的缩写叫做 ==ReLU==,名字念起来蛮有趣的,它真的就唸ReLU



那你把两个 ReLU 叠起来,就可以变成 Hard 的 Sigmoid,你想要用 ReLU 的话,就把 Sigmoid 的地方,换成$ max(0, b_i + w_i{_j}x_j)$.

那本来这边只有 i 个 Sigmoid,你要 **2 个 ReLU,才能够合成一个 Hard Sigmoid** ,所以这边有 i 个 Sigmoid,那如果 ReLU 要做到一样的事情,那你可能需要 2 倍的 ReLU,因为 2 个 ReLU 合起来,才是一个 Hard Sigmoid,所以要 2 倍的 ReLU,所以我们把 Sigmoid 换成 ReLU,这边就是把一个式子换了,因为要表示一个 Hard 的 Sigmoid,表示那个蓝色的 Function 不是只有一种做法,你完全可以用其他的做法,好那这个 Sigmoid 或是 ReLU,他们在机器学习裡面,我们就叫它 ==Activation Function==,他们是有名字的,他们统称為 Activation Function.

当然还有其他常见的,还有其他的 Activation Function,但 Sigmoid 跟 ReLU,应该是今天最常见的 Activation Function,那哪一种比较好呢,这个我们下次再讲,哪一种比较好呢,我接下来的实验都选择用了 ReLU,显然 ReLU 比较好,至於它為什麼比较好,那就是下週的事情了.

接下来就真的做了这个实验,这个都是真实的数据.

Experimental Results

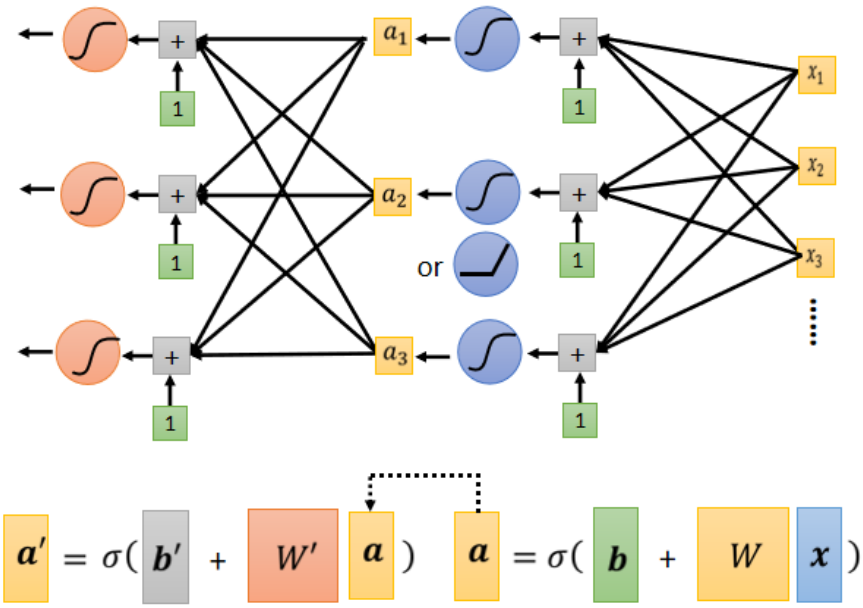
$$y = b + \sum_{2i} c_i \, max\left(0, b_i + \sum_j w_{ij} x_j\right)$$

	linear	10 ReLU	100 ReLU	1000 ReLU
2017 – 2020	0.32k	0.32k	0.28k	0.27k
2021	0.46k	0.45k	0.43k	0.43k

- 如果是 Linear 的 Model,我们现在考虑 56 天,训练资料上面的 Loss 是 0.32k,没看过的资料 2021 年资料是 0.46k
- 如果用 10 个 ReLU,好像没有进步太多,这边跟用 Linear 是差不多的,所以看起来 10 个 ReLU 不太够
- 100 个 ReLU 就有显著的差别了,100 个 ReLU 在训练资料上的 Loss,就可以从 0.32k 降到 0.28k,有 100 个 ReLU,我们就可以製造比较复杂的曲线,本来 Linear 就是一直线,但是 100 个 ReLU 我们就可以產生 100 个,有 100 个折线的Function,在测试资料上也好了一些.
- 接下来换 1000 个 ReLU,1000 个 ReLU,在训练资料上 Loss 更低了一些,但是在没看过的资料上,看起来也没有太大的进步

多做几次

接下来还可以做什麼呢,我们还可以继续改我们的模型,



举例来说,刚才我们说从 x 到 a 做的事情,是把 x 乘上 w 加 b,再通过 Sigmoid Function,不过我们现在已经知道说,不一定要通过 Sigmoid Function,通过 ReLU 也可以,然后得到 a。

我们可以把这个同样的事情,再反覆地多做几次,刚才我们把 w x 乘上 w 加 b,通过 Sigmoid Function 得到 a,我们可以把 a 再乘上另外一个 w',再加上另外一个 b',再通过 Sigmoid Function,或 RuLU Function,得到 a'

所以我们可以把 x,做这一连串的运算產生 a,接下来把 a 做这一连串的运算產生 a',那我们可以反覆地多做几次,**要做几次,这个又是另外一个 Hyper Parameter**,这是另外一个你要自己决定的事情,你要做两次吗 三次吗 四次吗 一百次吗,这个你自己决定,不过这边的 w 跟这边的 w',它们不是同一个参数喔,这个 b 跟这边的 b',它们不是同一个参数喔,是增加了更多的未知的参数

那就是接下来就真的做了实验了,我们就是每次都加 100 个 ReLU,那我们就是 Imput Features,就是 56 天前的资料

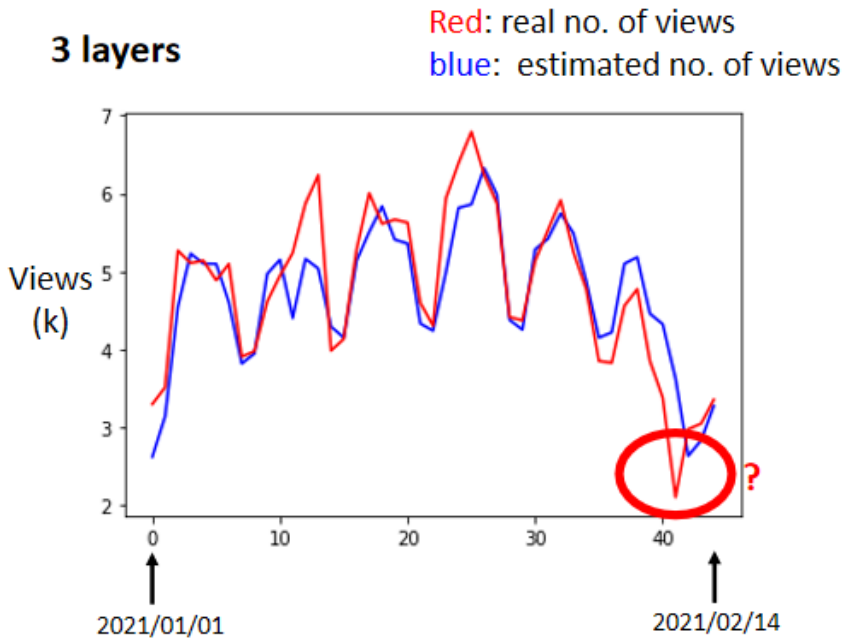
Experimental Results

- Loss for multiple hidden layers
  - 100 ReLU for each layer
  - input features are the no. of views in the past 56 days

	1 layer	2 layer	3 layer
2017 – 2020	0.28k	0.18k	0.14k
2021	0.43k	0.39k	0.38k

- 如果是只做一次 只做一次,就那个乘上 w 再加 b,再通过 ReLU 或 Sigmoid,这件事只做一次的话,这是我们刚才看到的结果
- 两次，这个 Loss 降低很多,0.28k 降到 0.18k,没看过的资料上也好了一些
- 三层，又有进步,从 0.18k 降到 0.14k,所以从一层到 从就是乘一次 w,到通过一次 ReLU,到通过三次 ReLU,我们可以从 0.28k 到 0.14k,在训练资料上,在没看过的资料上,从 0.43k 降到了 0.38k,看起来也是有一点进步的,

那这个是那个真实的实验结果啦,就我们来看一下,今天有做通过三次 ReLU 的时候,做出来的结果怎麼样



横轴就是时间,纵轴是观看的人次 是千人,**红色的线代表的是真实的数据,蓝色的线是预测出来的数据**

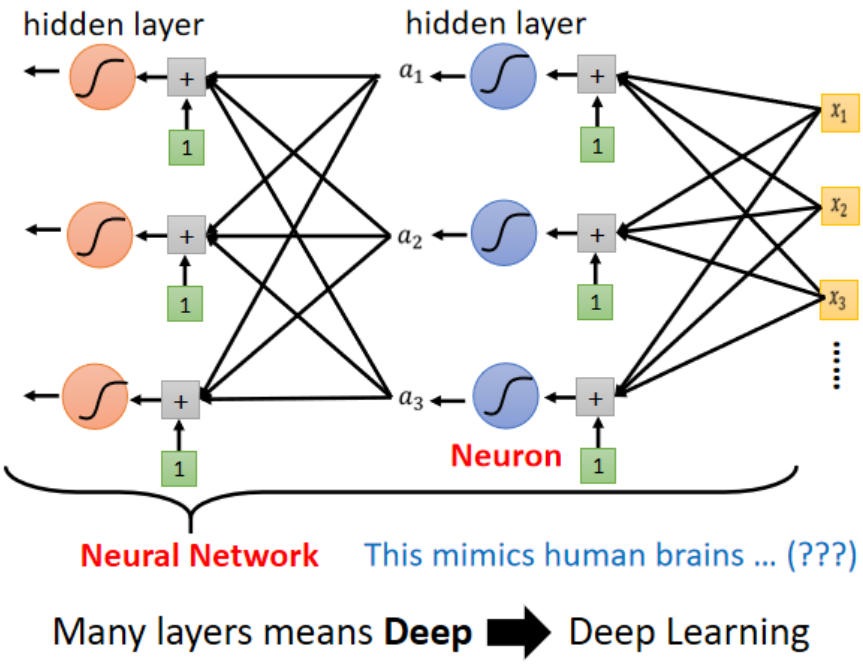
那你会发现说,在这种低点的地方啊,你看红色的数据是每隔一段时间,就会有两天的低点,在低点的地方,机器的预测还算是蛮準确的,

那这边有一个神奇的事情,这个机器高估了真实的观看人次,尤其是在这一天,这一天有一个很明显的低谷,但是机器没有预测到这一天有明显的低谷,它是晚一天才预测出低谷

这天最低点就是除夕啊,谁除夕还学机器学习 对不对,好 所以当然对机器来说,你不能怪它,它根本不知道除夕是什麼,它只知道看前 56 天的值,来预测下一天会发生什麼事,所以它不知道那一天是除夕,所以你不能怪它预测地不準,这一天就是除夕

好名字

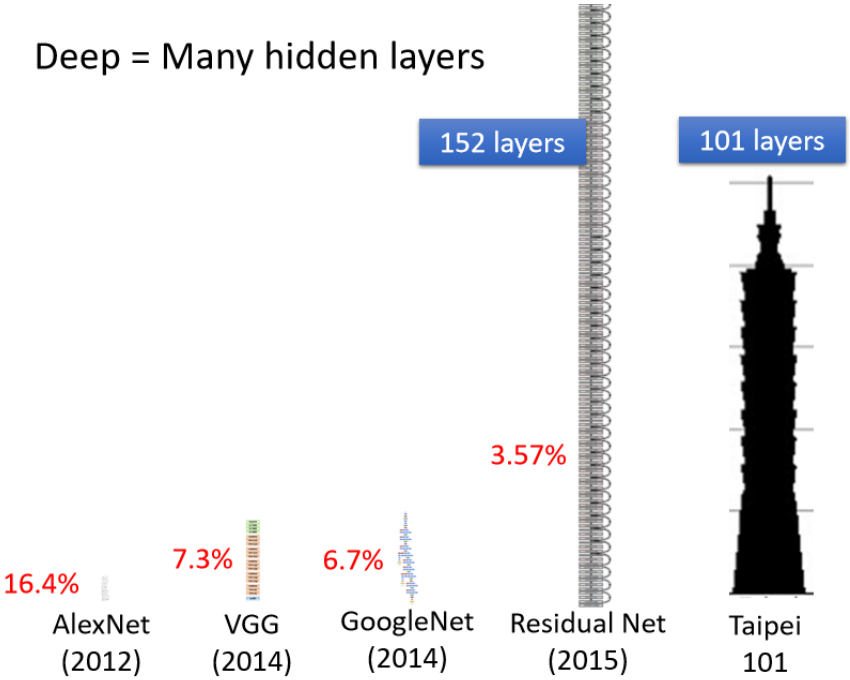
到目前為止,我们讲了很多各式各样的模型,那我们现在还缺了一个东西,缺一个好名字,你知道这个外表啊是很重要的,一个死臭酸宅穿上西装以后就潮了起来,或者是隻鞋半缕的,说他是汉左将军宜城亭侯中山靖王之后,也就潮了起来 ,所以我们的模型也需要一个好名字,所以它叫做什麼名字呢,这些 Sigmoid 或 ReLU 啊,它们叫做 Neuron,我们这边有很多的 Neuron,很多的 Neuron 就叫做 Neural Network,Neuron 就是神经元,人脑中就是有很多神经元,很多神经元串起来就是一个神经网路,跟你的脑是一样的,接下来你就可以到处骗麻瓜说,看到没有 这个模型就是在模拟人们脑 知道吗,这个就是在模拟人脑,这个就是人工智慧,然后麻瓜就会吓得把钱掏出来



但是啊 这个把戏在 80 90 年代的时候,已经玩过了这样,Neural Network 不是什麼新的技术,80 90 年代就已经用过了,当时已经把这个技术的名字搞到臭掉了,Neural Network 因為之前吹捧得太过浮夸,所以后来大家对 Neural Network 这个名字,都非常地感冒,它就像是个脏话一样,写在 Paper 上面都註定会被,就会註定害你的 Paper 被拒绝, 所以后来為了要重振 Neural Network 的雄风,所以怎麼办呢,需要新的名字,怎麼样新的名字呢,这边有很多的 Neural,每一排 Neural 我们就叫它一个 Layer,它们叫 Hidden Layer,有很多的 Hidden Layer 就叫做 Deep,这整套技术就叫做 ==Deep Learning==,好 我们就把 Deep Learning 讲完了,就是这麼回事。

Deep

就是这样来的,好 所以人们就开始,把类神经网路越疊越多 越疊越深,12 年的时候有一个 AlexNet,它有 8 层 它的错误率是 16.4%,两年之后 VGG 19层,错误率在影像辨识上进步到 7.3 %,这个都是在影像辨识上一个,这个基準的资料库上面的结果,后来 GoogleNet 有错误率降到 6.7%,有 22 层,但这些都不算是什麼



Residual Net 有 152 层啊,它比 101 还要高啊,但是这个 Residual Net 啊,其实要训练这麼深的 Network 是有诀窍的,这个我们之后再讲。

如果你仔细思考一下,我们一路的讲法的话,你有没有发现一个奇妙的违和的地方,我们一开始说,我们想要用 ReLU 或者是 Sigmoid,去逼近一个复杂的 Function,实际上只要够多的 ReLU 够多的 Sigmoid,就可以逼近任何的 连续的 Function ,我们只要有够多的 Sigmoid,就可以知道够复杂的线段,就可以逼近任何的 Continuous 的 Function,所以我们只要一排 ReLU 一排 Sigmoid,够多就足够了,**那深的意义到底何在呢**

把 ReLU Sigmoid Function 反覆用,到底有什麼好处呢,為什麼不把它们直接排一排呢,直接排一排也可以表示任何 Function 啊,所以把它反覆用没什麼道理啊,所以有人就说把 Deep Learning,把 ReLU Sigmoid 反覆用,不过是个噱头,你之所以喜欢 Deep Learning,只是因為 Deep 这个它名字好听啦,ReLU Sigmoid 排成一排,你只可以製造一个肥胖的 Network,Fat Neural Network,跟 Deep Neural Network 听起来,量级就不太一样,Deep 听起来就比较厉害啦,Fat Neural Network 还以為是死肥宅 Network,就不厉害这样子,那到底 Deep 的理由,為什麼我们不把 Network 变胖,只把 Network 变深呢,这个是我们日后要再讲的话题

那有人就说,那**怎麼不变得更深呢**,刚才只做到 3 层,应该要做得更深嘛,现在 Network 都是叠几百层的啊,没几百层都不好意思说,你在叫做 Deep Learning 所以要做更深,

Why don't we go deeper?

- Loss for multiple hidden layers
  - 100 ReLU for each layer
  - input features are the no. of views in the past 56 days

	1 layer	2 layer	3 layer	4 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k
2021	0.43k	0.39k	0.38k	0.44k

Better on training data, worse on unseen data

➡ Overfitting

所以确实做得更深 做 4 层,4 层在训练资料上,它的 Loss 是 0.1k,在没有看过 2021 年的资料上,是如何呢 是 0.44k,惨掉了。在训练资料上,3 层比 4 层差,4 层比 3 层好,但是在没看过的资料上,4 层比较差,3 层比较好,在有看过的资料上,在训练资料上,跟没看过的资料上,它的结果是不一致的,这种训练资料跟测试,这种训练资料跟没看过的资料,它的结果是不一致的状况,这个状况叫做 ==Overfitting==,

机器学习会发生 **Overfitting** 的问题,指的就是**在训练资料上有变好,但是在没看过的资料上没有变好**这件事情,

但是做到目前为止,我们都还没有真的发挥这个模型的力量,你知道我们要发挥这个模型的力量,和 2021 的资料到 2 月 14 号之前的资料,我们也都已经手上有了

Let's predict no. of views today!

- If we want to select a model for predicting no. of views today, which one will you use?

	1 layer	2 layer	3 layer	4 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k
2021	0.43k	0.39k	0.38k	0.44k

We will talk about model selection next time. ☺

所以我们要真正做的事情是什麼,我们要做的事情就是预测未知的资料,但是如果我们要预测未知的资料,我们应该选 3 层的 Network,还是 4 层的 Network 呢,举例来说今天是 2 月 26 号,今天的观看人数我们还不知道,如果我们要用一个 Neural Network,用我们已经训练出来的 Neural Network,去预测今天的观看人数,

至於怎麼选模型,这个是下週会讲的问题,大家 但是大家都非常有 Sense,知道我们要选 3 层的,多数人都决定要选 3 层的,你可能会说 我怎麼不选 4 层呢,4 层在训练资料上的结果比较好啊,**可是我们并不在意训练资料的结果啊,我们在意的是没有看过的资料**,而 2 月 26 号是没有看过的资料,我们应该选一个在训练的时候,没有看过的资料上表现会好的模型,所以我们应该选 3 层的 Network

那你可能以为这门课就到这边结束了,其实不是 我们真的来预测一下,2 月 26 号应该要有的观看次数是多少,但是因為其实 YouTube 的统计,它没有那麼及时,所以它现在只统计到 2 月 24 号,没关係 我们先计算一下 2 月 25 号的,观看人数是多少,这个 3 层的 Network 告诉我说,2 月 25 号这个频道的总观看人次,应该是 5250 人,那我们先假设 2 月 25 号是对的,但实际上我还不知道 2 月 25 号对不对,因為 YouTube 后台统计的数据还没有出来啊,但我们先假设这一天都是对的,然后再给我们的模型去预测 2 月 26 号的数字,得到的结果是 3.96k 有 3960 次,那它為什麼这边特别低,因為模型知道说,这个礼拜五观看的人数,就是比较少啊,所以它预测特别低,听起来也是合理的



好 那今天其实就讲了深度学习,那今天讲的不是一般的介绍方式,如果你想要听一般的介绍方式,过去的课程影片也是有的,我就把连结附在这边,然后深度学习的训练,会用到一个东西叫 Backpropagation,其实它就是比较有效率,算 Gradients 的方法,跟我们今天讲的东西没有什麼不同,但如果你真的很想知道,Backpropagation 是什麼的话,影片连结也附在这边.

General Guidance : overfit

Framework of ML

我们已经看了作业一了,其实之后好几个作业,它看起来的样子,基本上都是大同小异

Training data: $\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \ldots, (x^N, \hat{y}^N)\}$

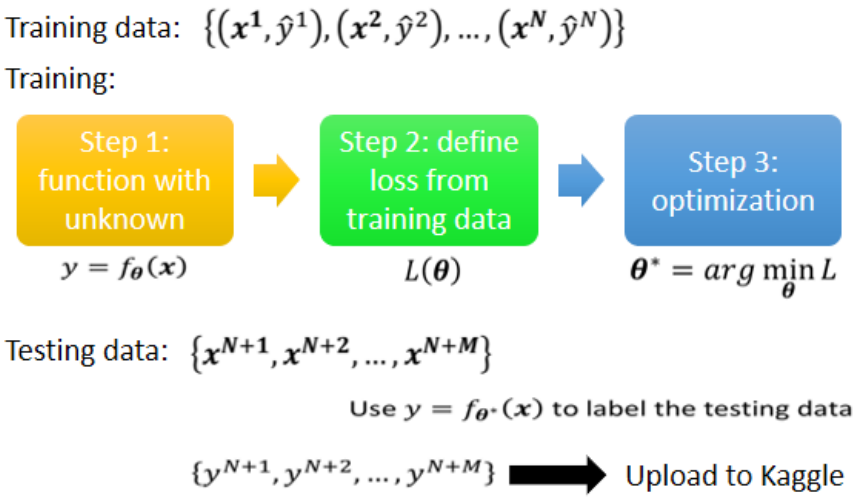
Testing data: $\{x^{N+1}, x^{N+2}, \ldots, x^{N+M}\}$

就是你会有一堆训练的资料,这些**训练集**裡面,会包含了x跟y的hat,x^1 和跟它对应的$\hat{y}^1$,x^2 跟它对应的$\hat{y}^2$,一直到x^n 还有它对应的$\hat{y}^n$

测试集,**测试集就是你只有x没有y**,其实在之后每一个作业,看起来都是非常类似的格式



- 作业二,其实是做**语音辨识**,我们的x就是非常小的一段声音讯号,其实这个不是真正完整的语音辨识系统,它是语音辨识系统的一个阉割版,ŷ是要去预测判断,这一小段声音讯号,它对应到哪一个phoneme。你不知道phoneme是什麼没有关係,你就把它想成是音标就可以了
- 作业三叫做**图像识别**,这个时候我们的x是一张图片,ŷ是机器要判断说这张图片裡面,有什麼样的东西
- 作业四是**语者辨识**,语者辨识要做的事情是,这个x也是一段声音讯号,ŷ现在不是phoneme,ŷ是现在是哪一个人在说话,这样的系统,现在其实非常的有用,如果你打电话去银行的客服,现在都有自动的语者辨认系统,它会听说现在打电话进来的人,是不是客户本人,就少了客服人员问你身份验证的时间
- 作业五是做**机器翻译**,x就是某一个语言,比如说,这是我唯一会的一句日文,痛みを知れ,它的ŷ就是另外一句话。



训练集就要拿来训练我们的Model,训练Model的过程上週已经讲过了,训练的过程就是**三个步骤**

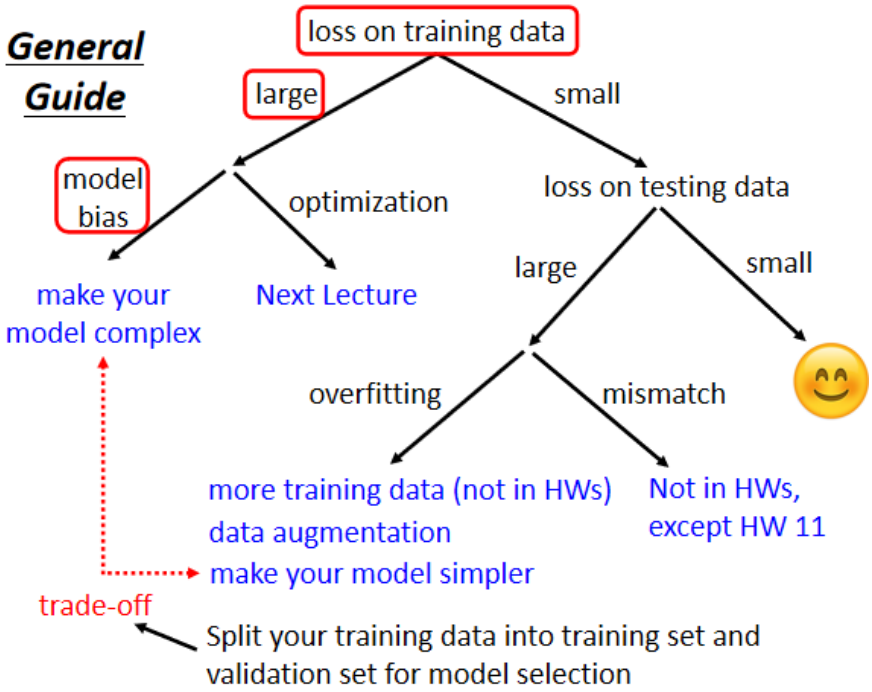
- 第一步,你要先写出一个有未知数的function,这个未知数,以后我们都用$θ$来代表,一个Model裡面所有的未知函数,所以$f_θ(x)$的意思就是说,我现在有一个function叫f(x),它裡面有一些未知的参数,这些未知的参数表示成$θ$,它的input叫做x,这个input叫做feature
- 第二步， 你要定一个东西叫做loss,loss是一个function,这个loss的输入就是一组参数,去判断说这一组参数是好还是不好
- 第三步， 你要解一个,Optimization的problem,你要去找一个θ,这个θ 可以让loss的值越小越好,可以让loss的值 最小的那个$θ$,我们叫做$θ^*$

有了$θ^$*以后,那你就把它拿来用在测试集上,也就是你把$θ*^$带入这些未知的参数,本来$f_θ(x)$裡面有一些未知的参数,现在这个$θ$ 用$θ^*$来取代,它的输入就是你现在的测试集,输出的结果 你就把它存起来,然后上传到Kaggle就结束了。

接下来你就会遇到一个问题,直接执行助教的sample code,往往只能够给你simple baseline的结果而已,如果你想要做得更好,那应该要怎麼办,

General Guide

以下就是如何让你做得更好的攻略,它适用於前期所有的作业,这个攻略是怎麼走的呢?



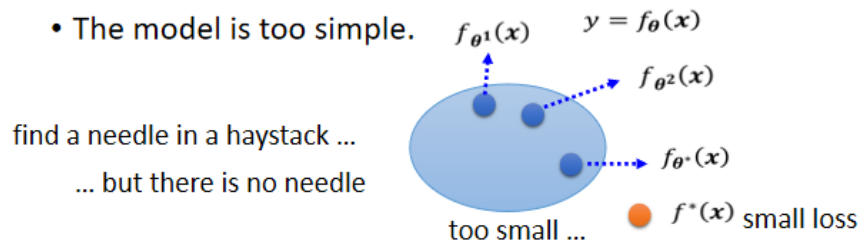
从最上面开始走起,第一个是你今天如果你觉得,你在Kaggle上的结果不满意的话,第一件事情你要做的事情是,**检查你的training data的loss**

有人说"我在意的不是应该是,testing data的loss吗,因為Kaggle上面的结果,呈现的是testing data的结果"

但是你要先检查你的training data,看看你的model在training data上面,有没有学起来,再去看testing的结果,如果你发现,你的**training data的loss很大**,显然它**在训练集上面也没有训练好**,接下来你要分析一下,在训练集上面没有学好,是什麼样的原因,这边有两个可能,第一个可能是model的bias

Model bias

model的bias这件事情,我们在上週已经跟大家讲过了,所谓model bias的意思是说,假设你的model太过简单。



举例来说,我们现在写了一个有未知parameter的function,这个未知的parameter,我们可以代各种不同的数字,你代θ¹ 得到一个function $f{*θ^1}(x)$,*我们把那个function用这个,一个点来表示,θ² 得到另一个function* $f{θ^1}(x)$,你把所有的function集合起来,得到一个function的set.

但是这个function的set太小了,这个function的set裡面,没有包含任何一个function,可以让我们的loss变低,即可以让loss变低的function,不在你的model可以描述的范围内。

在这个情况下,就算你找出了一个θ*,它是这些蓝色的function裡面,最好的那一个,也无济於事了,那个loss还是不够低。

这个状况就是你想要在**大海裡面捞针**,这个针指的是一个loss低的function,结果**针根本就不在海裡**,白忙一场,你怎麼捞都捞不出针,因為针根本就不在你的,这个function set裡面,不在你的这个大海裡面,所以怎麼办?

- Solution: redesign your model to make it more flexible

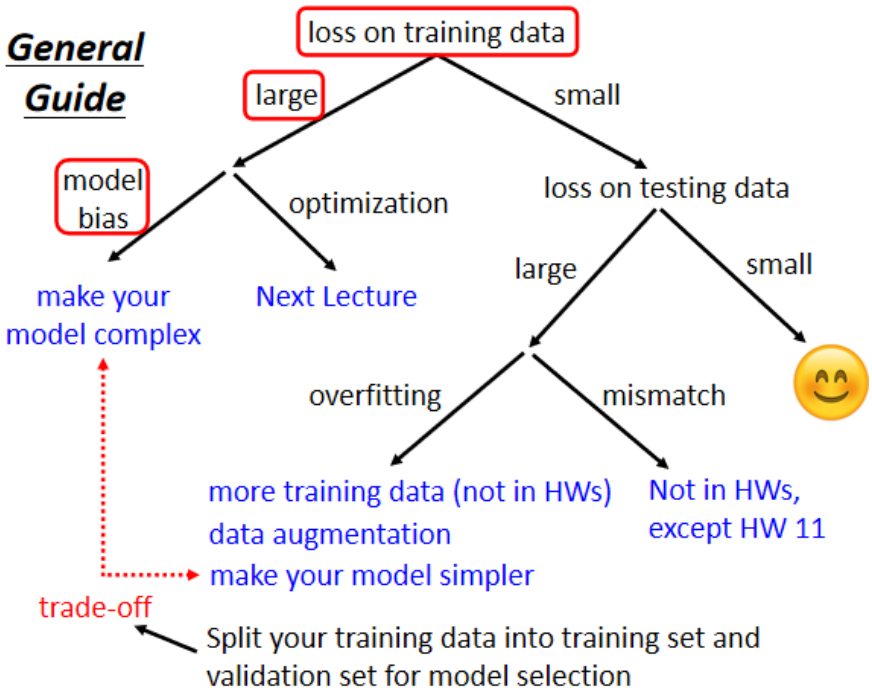
$$y = b + wx_1 \xrightarrow{\text{More features}} y = b + \sum_{j=1}^{56} w_j x_j$$

Deep Learning (more neurons, layers)

$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j\right)$$

这个时候**重新设计一个model,给你的model更大的弹性**,举例来说,你可以增加你输入的features,我们上週说,本来我们输入的features,只有前一天的资讯,假设我们要预测接下来的这个,观看人数的话,我们用前一天的资讯,不够多,那用56天前的资讯,那model的弹性就比较大了

你也可以用Deep Learning,增加更多的弹性,所以如果你觉得,你的model的弹性不够大,那你可以增加更多features,可以设一个更大的model,可以用deep learning,来增加model的弹性,这是第一个可以的解法。

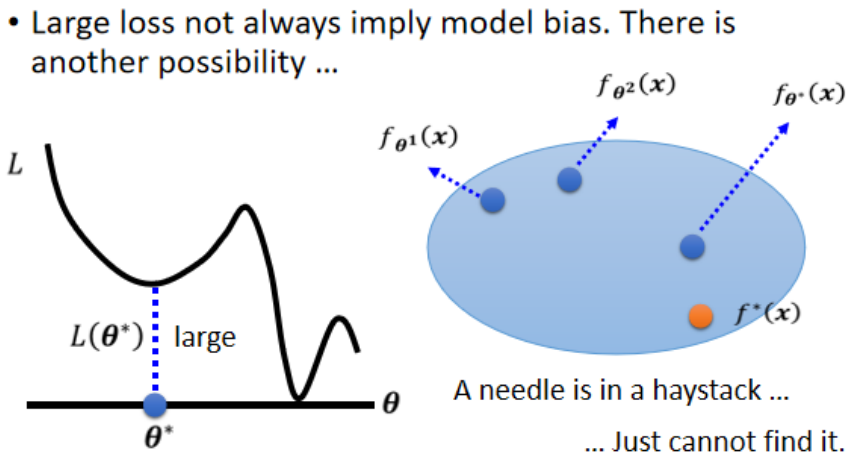


但是并不是training的时候,loss大就代表一定是model bias,你可能会遇到另外一个问题,这个问题是什麼,这个问题是**optimization做得不好**,什麼意思呢?

Optimization Issue

我们知道说,我们今天用的optimization,在这门课裡面,我们其实都只会用到gradient descent,这种optimization的方法,这种optimization的方法很多的问题。

举例来说 我们上週也讲过说,你可能会卡在==local minima==的地方,你没有办法找到一个,真的可以让loss很低的参数,如果用图具象化的方式来表示,就像这个样子



蓝色部分是你的model可以表示的函式所形成的集合,你可以把θ 代入不同的数值,形成不同的function,把所有的function通通集合在一起,得到这个蓝色的set,这个蓝色的set裡面,确实包含了一些function,这些function它的loss是低的。

但问题是**gradient descent这一个演算法,没办法帮我们找出,这个loss低的function**,gradient descent是解一个optimization的problem,找到θ* 然后就结束了

但是这个$θ^*$ 它给我的loss不够低,这一个model裡面,存在著某一个function,它的loss是够低的,gradient descent,没有给我们这一个function

这就好像是说 我们想**大海捞针,针确实在海裡,但是我们却没有办法把针捞起来**,这边问题就来了

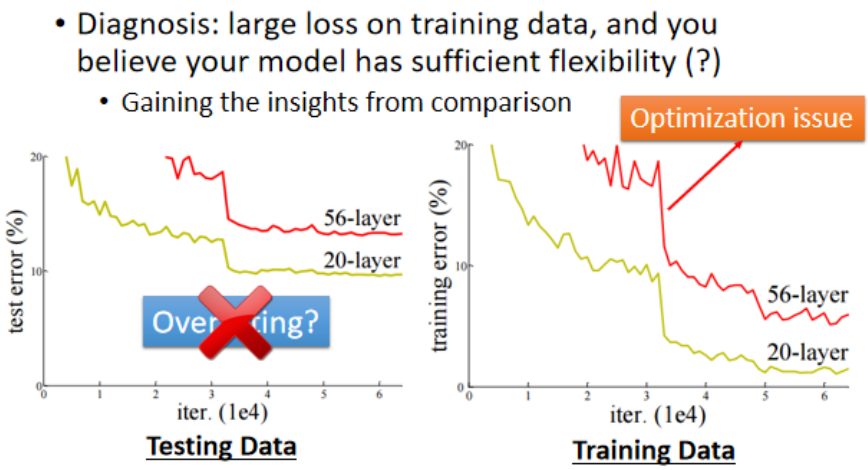
==training data的loss不够低的时候,到底是model bias,还是optimization的问题呢==

- 找不到一个loss低的function,到底是因為我们的model的弹性不够,我们的海裡面没有针
- 还是说,我们的model的弹性已经够了,只是optimization gradient descent不给力,它没办法把针捞出来

到底是哪一个呢,到底我们的model已经够大了,还是它不够大,怎麼判断这件事呢

Gaining the insights from comparison

一个建议判断的方法,就是你可以**透过比较不同的模型,来得知说,你的model现在到底够不够大**,怎麼说呢



我们这边举一个例子,这一个实验是从residual network,那篇paper裡面节录出来的,我们把paper链接：http://arxiv.org/abs/1512.03385

这篇paper一开头就跟你讲一个故事,它说 我想测2个networks

- **一个network有20层**
- **一个network有56层**

我们把它们测试在测试集上,这个**横轴指的是training的过程**,就是你参数update的过程,随著参数的update,当然你的loss会越来越低,但是结果**20层的loss比较低,56层的loss还比较高**

这个residual network是比较早期的paper,2015年的paper,如果你现在大学生的话,那个时候你都还是高中生而已,所以那个时候大家对Deep Learning,了解还没有那麼的透彻,大家对deep learning,有各种奇怪的误解,很多人看到这张图都会说,这个代表overfitting,告诉你deep learning不work,56层太深了 不work,根本就不需要那麼深

那个时候大家也不是每个人都觉得deep learning是好的,那时候还有很多,对deep learning的质疑,所以看到这个实验有人就会说,最深没有比较好,所以这个叫做overfitting,但是这个是overfitting吗

这个不是overfitting,等一下会告诉你overfitting是什麼,**并不是所有的结果不好,都叫做overfitting**

你要**检查一下训练集上面的解释**,你检查训练集的结果发现说,现在20层的network,跟56层的network比起来,在训练集上,20层的network loss其实是比较低的,56层的network loss是比较高的

这代表56层的network,它的optimization没有做好,它的optimization不给力

你可能问说,你怎麼知道是56层的optimization不给力,搞不好是model bias,搞不好是56层的network,它的model弹性还不够大,它要156层才好,56层也许弹性还不够大,但是你比较56层跟20层,20层的loss都已经可以做到这样了,56层的弹性一定比20层更大对不对

如果今天**56层的network要做到20层的network可以做到的事情**,对它来说是轻而易举的

它**只要前20层的参数,跟这个20层的network一样**,剩下36层就什麼事都不做,identity copy前一层的输出就好了,56层的network一定可以做到20层的network可以做到的事情,所以20层的network已经都可以走到这麼底的loss了,56层的

network,它比20层的network弹性还要更大,所以没有道理

所以56层的network,如果你optimization成功的话,它应该要比20层的network,可以得到更低的loss,但结果在训练集上面没有,这个不是overfitting,这个也不是model bias,因为56层network弹性是够的,这个问题是你的**optimization不给力,optimization做得不够好**

Start from shallower networks (or other models), which are easier to train.

所以刚才那个例子就告诉我们,你怎麼知道你的optimization有没有做好,这边给大家的建议是

看到一个你从来没有做过的问题,也许你可以先跑一些比较小的,比较浅的network,或甚至用一些,不是deep learning的方法,比如说 linear model,比如说support vector machine,support vector machine不知道是什麼也没有关係,它们可能是比较容易做Optimize的,它们比较不会有optimization失败的问题

也就是这些model它会竭尽全力的,在它们的能力范围之内,找出一组最好的参数,它们比较不会有失败的问题,所以你可以先train一些,比较浅的model,或者是一些比较简单的model,先知道 先有个概念说,这些简单的model,到底可以得到什麼样的loss

If deeper networks do not obtain smaller loss on training data, then there is optimization issue.

接下来还缺一个深的model,如果你发现你深的model,跟浅的model比起来,深的model明明弹性比较大,但loss却没有办法比浅的model压得更低,那就代表说你的optimization有问题,你的gradient descent不给力,那你要有一些其它的方法,来把optimization这件事情做得更好

- Diagnosis: large loss on training data, and you believe your model has sufficient flexibility (?)
  - Gaining the insights from comparison
  - Start from shallower networks (or other models), which are easier to train.
  - If deeper networks do not obtain smaller loss on **training data**, then there is optimization issue.

	1 layer	2 layer	3 layer	4 layer	5 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k	0.34k

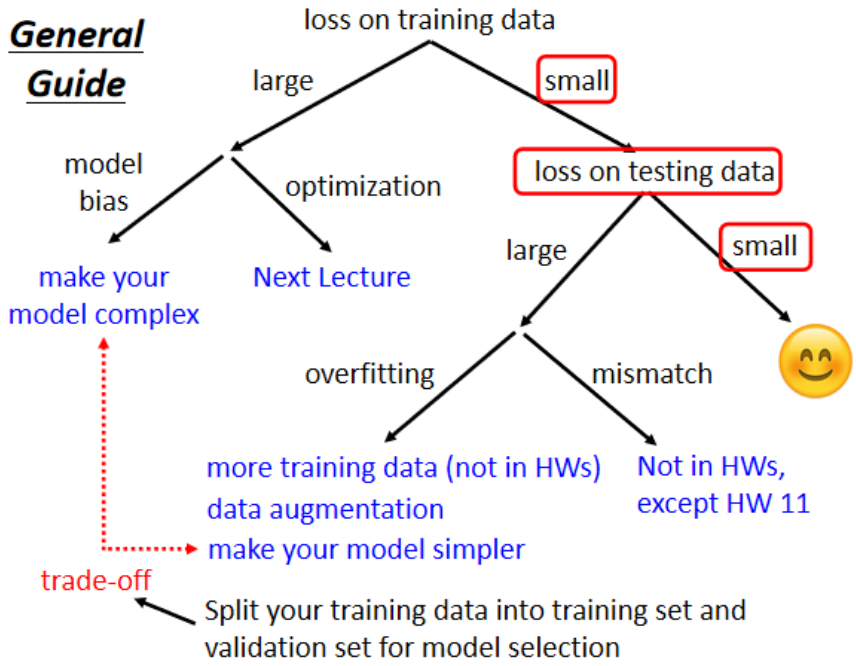
- Solution: More powerful optimization technology (next lecture)

举例来说,我们上次看到的这个,观看人数预测的例子,我们说在训练集上面,2017年到2020年的资料是训练集,一层的network,它的loss是0.28k,2层就降到0.18k,3层就降到0.14k,4层就降到0.10k。

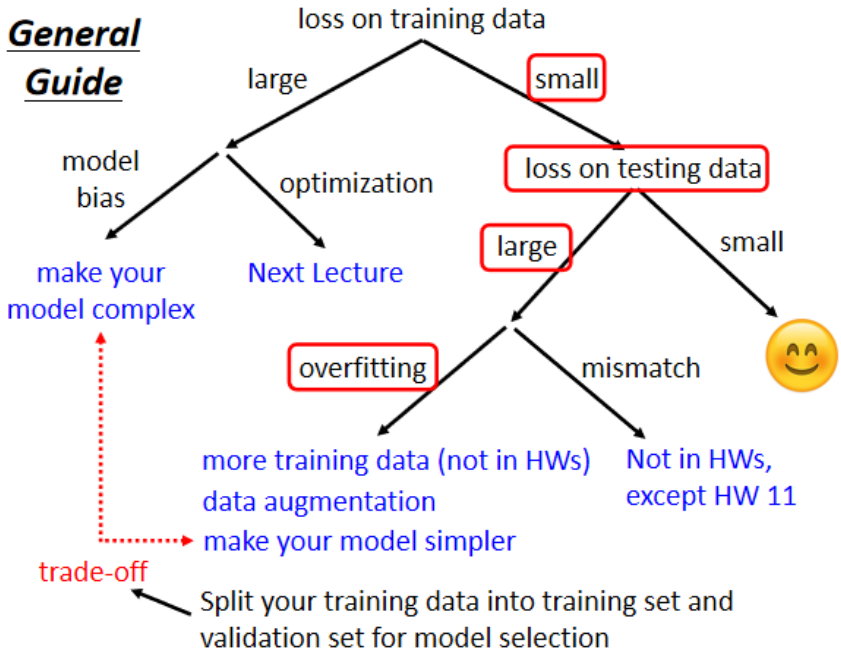
但是 我测5层的时候结果变成0.34k,这是什麼问题

我们现在loss很大，这显然不是model bias的问题,因为4层都可以做到0.10k了,5层应该可以做得更低,这个是optimization的problem,这个是optimization的时候做得不好,才造成这样子的问题。

那如果optimization做得不好的话,怎麼办呢,这个我们下一节课,就会告诉大家要怎麼办,你现在就知道怎麼判断,现在如果你的training的loss大,到底是model bias还是optimization,如果model bias 那就把model变大,如果是optimization失败了,那就看等一下的课程怎麼解这个问题。



假设你现在经过一番的努力,你已经可以让你的,training data的loss变小了,那接下来你就可以来看,testing data loss,如果testing data loss也小,有比这个strong baseline还要小就结束了,没什麼好做的就结束了。



那但是如果你觉得还不够小呢,==如果training data上面的loss小,testing data上的loss大,那你可能就是真的遇到overfitting的问题==

你要注意 是training的loss小,testing的loss大 才叫做overfitting,很多同学每次一看到结果不好,在testing上的结果不好,就说这个是overfitting,不一定是overfitting。

你拿一个结果来问我说,老师这个结果要怎麼做得更好的时候,我第一个问题就会问你说,你在training data上的loss,到底做得怎麼样,我发现十个同学有八个同学都说,要看training data的loss吗,我没有把training data loss记下来,你要把training data loss记下来,先确定说你的optimization没有问题,你的model够大了,然后接下来,才看看是不是testing的问题,**如果是training的loss小,testing的loss大,这个有可能是overfitting**

Overfitting

为什麼会有overfitting这样的状况呢,为什麼有可能training的loss小,testing的loss大呢,这边就举一个极端的例子来告诉你说,为什麼会发生这样子的状况

- Small loss on training data, large loss on testing data. Why?

An extreme example

Training data: $\{(\boldsymbol{x^1}, \hat{y}^1), (\boldsymbol{x^2}, \hat{y}^2), \ldots, (\boldsymbol{x^N}, \hat{y}^N)\}$

$$f(\boldsymbol{x}) = \begin{cases} \hat{y}^i & \exists \boldsymbol{x}^i = \boldsymbol{x} \\ random & otherwise \end{cases}$$

Learns nothing ... !

This function obtains **zero training loss**, but **large testing loss**.

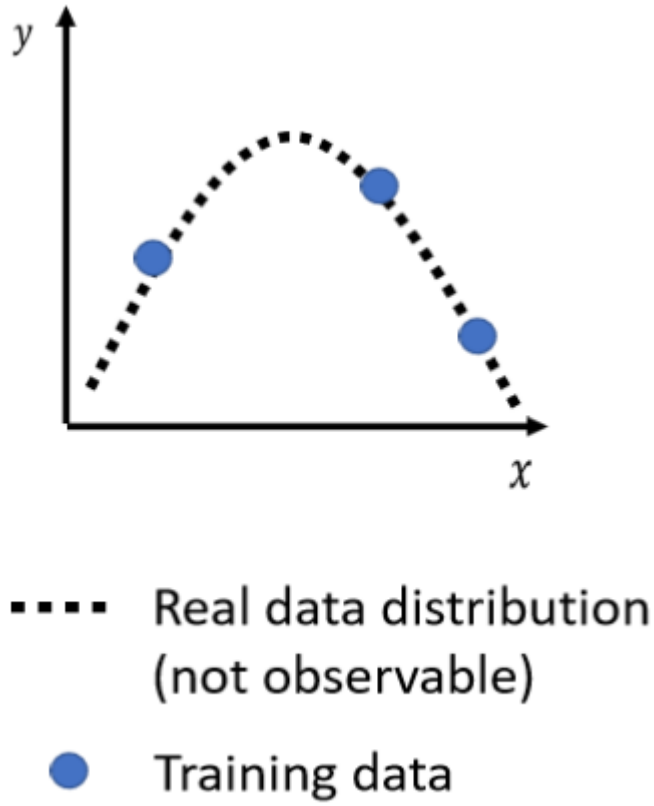
这是我们的训练集,假设根据这些训练集,某一个很废的,machine learning的方法,它找出了一个一无是处的function

这个一无是处的function说,**如果今天x当做输入的时候,我们就去比对这个x,有没有出现在训练集裡面,如果x有出现在训练集裡面,就把它对应的ŷ当做输出,如果x没有出现在训练集裡面,就输出一个随机的值**

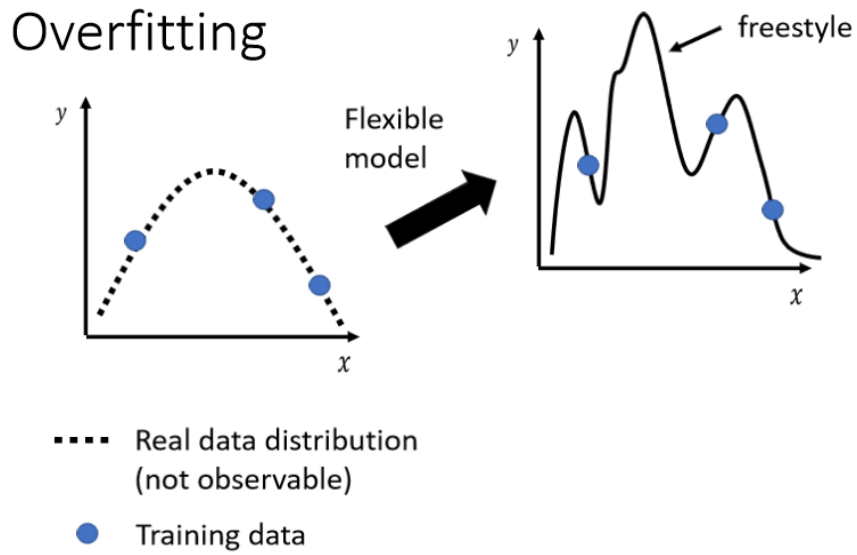
那你可以想像这个function啥事也没有干,它是一个一无是处的function,但虽然它是一个一无是处的function,它**在training的data上，它的loss可是0呢**

你把training的data,通通丢进这个function裡面,它的输出跟你的训练集的level,是一模一样的,所以在training data上面,这个一无是处的function,它的loss可是0呢,可是在testing data上面,它的loss会变得很大,因为**它其实什麼都没有预测**,这是一个比较极端的例子,在一般的状况下,也有可能发生类似的事情。

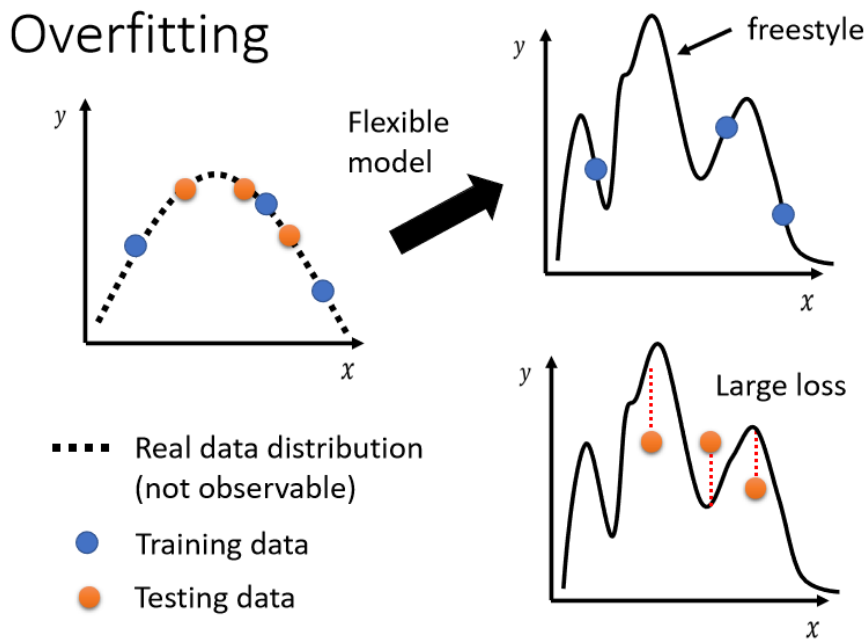
举例来说,假设我们输入的feature叫做x,我们输出的level叫做y,那x跟y都是一维的



x跟y之间的关係,是这个二次的曲线,这个曲线我们刻意用虚线来表示,因為我们通常没有办法,直接观察到这条曲线,我们真正可以观察到的是什麼,我们真正可以观察到的,是我们的训练集,训练集 你可以想像成,就是从这条曲线上面,随机sample出来的几个点



今天的模型 它的能力非常的强,它的flexibility很大,它的弹性很大的话,你只给它这三个点,它会知道说,在这三个点上面我们要让loss低,所以今天你的model,它的这个曲线会通过这三个点,但是其它没有训练集做為限制的地方,它就会有freestyle,因為它的flexibility很大,它弹性很大,所以你的model,可以变成各式各样的function,你没有给它资料做為训练,它就会有freestyle， 可以產生各式各样奇怪的结果。

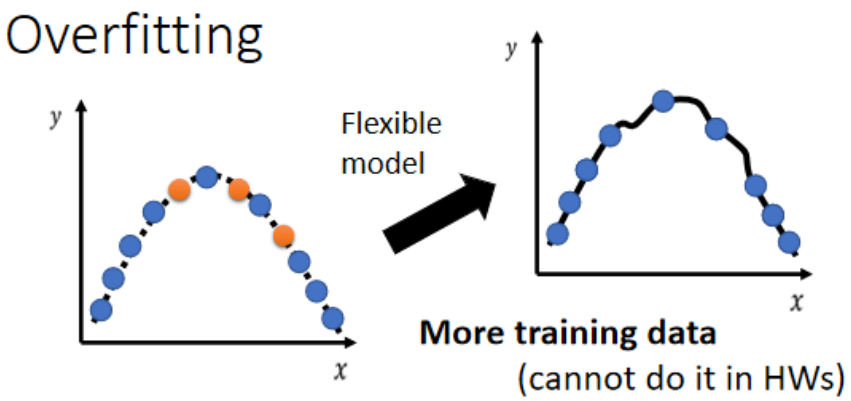


这个时候,如果你再丢进你的testing data,你的testing data 和training的data,当然不会一模一样,它们可能是从同一个,distribution sample出来的,testing data是橙色的这些点,训练data是蓝色的这些点

用蓝色的这些点,找出一个function以后,你测试在橘色的这些点上,不一定会好,**如果你的model它的自由度很大的话,它可以產生非常奇怪的曲线,导致训练集上的结果好,但是测试集上的loss很大**,那至於更详细的背后的数学原理,為什麼这个比较有弹性的model,它就比较会,overfitting背后的数学原理,我们留待下下週

那怎麼解决刚才那个,overfitting的问题呢,有两个可能的方向

1. 第一个方向是,也许这个方向往往是最有效的方向,是**增加你的训练集**



今天假设你自己,想要做一个application,你发现有overfitting的问题,其实我觉得,最简单解决overfitting的方法,就是增加你的训练集

所以今天如果训练集,蓝色的点变多了,那虽然你的model它的弹性可能很大,但是因為你这边的点非常非常的多,它就可以限制住,它看起来的形状还是会很像,產生这些资料背后的二次曲线,但是你在作业裡面,你是不能够使用这一招的,因為我们并不希望大家浪费时间,来收集资料

那你可以做什麼呢,你可以做**data augmentation**，这个方法并不算是使用了额外的资料。



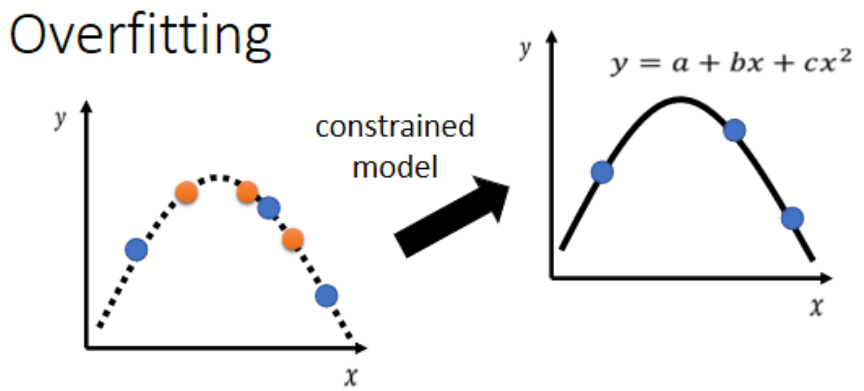
Data augmentation就是,你用一些你对於这个问题的理解,自己创造出新的资料。

举例来说在做影像辨识的时候,非常常做的一个招式是,假设你的训练集裡面有某一张图片,把它左右翻转,或者是把它其中一块截出来放大等等,你做左右翻转 你的资料就变成两倍,那这个就是data augmentation

但是你要注意一下data augmentation,不能够随便乱做,这个augment 要augment得有道理,举例来说在影像辨识裡面,你就**很少看到有人把影像上下颠倒**当作augmentation,因為这些图片都是合理的图片,你把一张照片左右翻转,并不会影响到裡面是什麼样的东西,但你把它颠倒 那就很奇怪了,这可能不是一个训练集裡面,可能不是真实世界会出现的影像

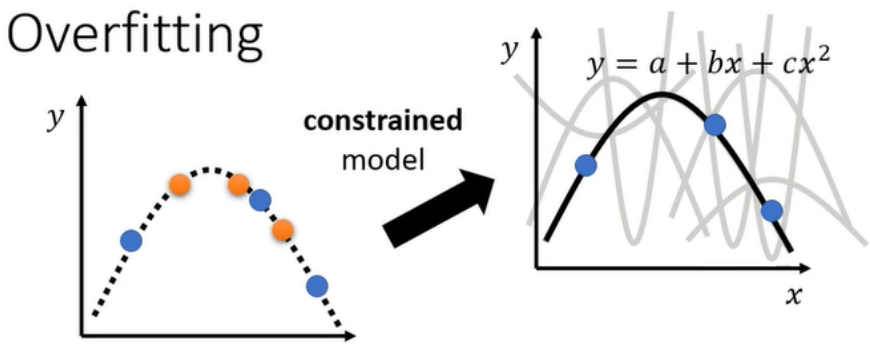
那如果你给机器看这种,奇怪的影像的话,它可能就会学到奇怪的东西,所以**data augmentation,要根据你对资料的特性,对你现在要处理的问题的理解,来选择合适的,data augmentation的方式**,好 那这边是增加资料的部分

1. 另外一个解法就是**不要让你的模型,有那麼大的弹性**,给它一些限制,



举例来说 假设我们直接限制说,现在我们的model,我们somehow猜测出 知道说,x跟y背后的关係,其实就是一条二次曲线,只是我们不明确的知道这二次曲线,裡面的每一个参数长什麼样

那你说你怎麼会猜测出这样子的结果,你怎麼会知道说,**要用多constrain的model才会好呢**,那这就取决於你对这个问题的理解,因為这种model是你自己设计的,到底model要多constrain多flexible,结果才会好,那这个要问你自己,那要看这个设计出不同的模型,你就会得出不同的结果

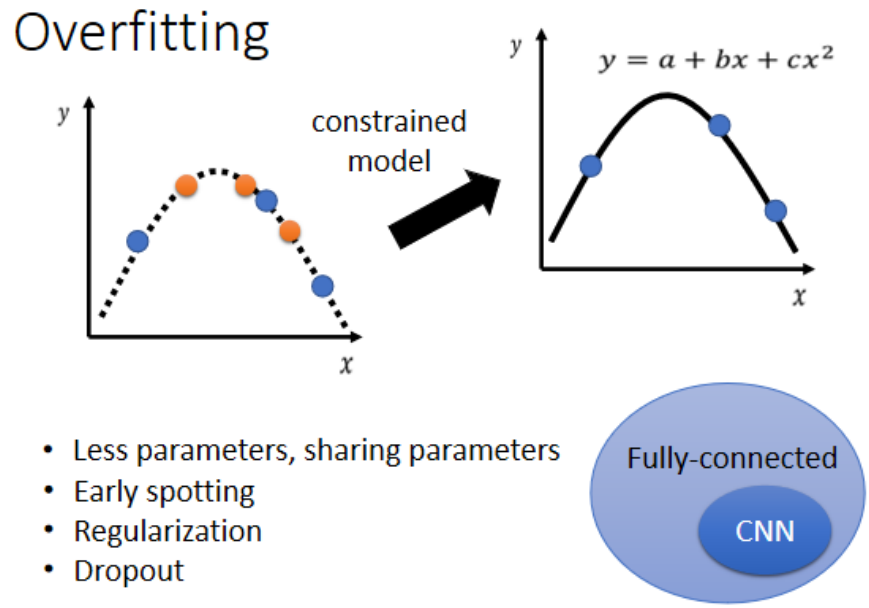


那现在假设我们已经知道说,模型就是二次曲线,那你就会给你,你就会在选择function的时候,有很大的限制,因為二次曲线要嘛就是这样子,来来去去就是那几个形状而已,所以当我们的训练集有限的时候,因為我们来来去去,只能够选那几个function

所以你可能,虽然说只给了三个点,但是因為我们能选择的function有限,你可能就会正好选到,跟真正的distribution,比较接近的function,然后在测试集上得到比较好的结果。

所以这是第二个方法,解决overfitting的问题,你要给你的model一些限制,最好你的model正好,跟背后產生资料的过程,你的process是一样的,那你可能就会,你就有机会得到好的结果

有哪些方法可以给model製造限制呢,举例来说,



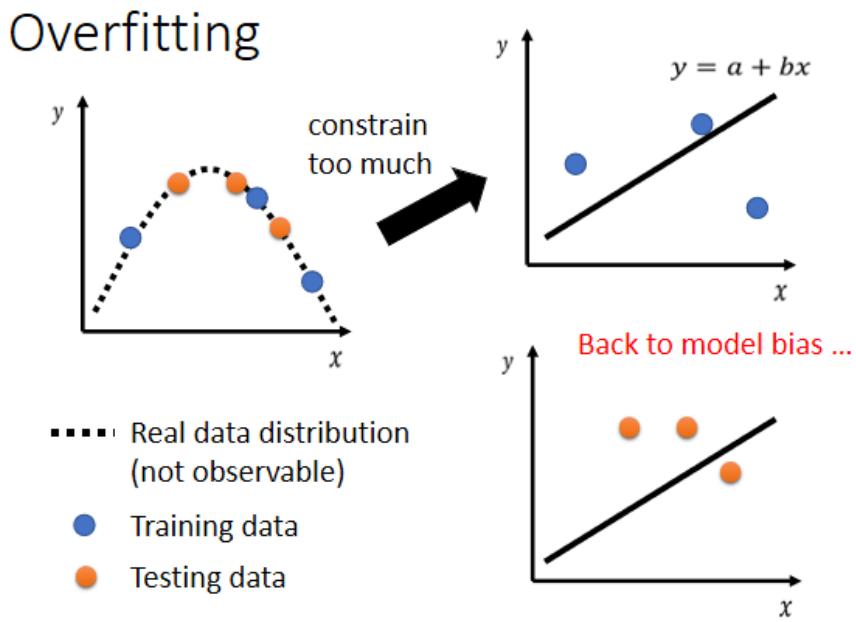
- 给它**比较少的参数**,如果是deep learning的话,就给它比较少的神经元的数目,本来每层一千个神经元,改成一百个神经元之类的,或者是你可以让model共用参数,你可以让一些参数有一样的数值,那这个部分如果你没有很清楚的话,也没有关係,我们之后在讲CNN的时候,会讲到这个部分,所以这边先前情 先预告一下,就是我们之前讲的network的架构,叫做==fully-connected network==,那fully-connected network,其实是一个比较有弹性的架构,而==CNN是一个比较有限制的架构==,就说你可能会说,CNN不是比较厉害吗,大家都说做影像就是要CNN,比较厉害的model,难道它比较没有弹性吗,没错,它是一种比较没有弹性的

model,它厉害的地方就是,它是针对影像的特性,来限制模型的弹性,所以你今天fully-connected的network,可以找出来的function所形成的集合,其实是比较大的,CNN这个model所找出来的function,它形成的集合其实是比较小的,其实包含在fully-connected的,network裡面的,但是就是因為CNN给了,比较大的限制,所以CNN在影像上,反而会做得比较好,那这个之后都还会再提到,

- 另外一个就是用**比较少的features**,那刚才助教已经示范过,本来给三天的资料,改成用给两天的资料,其实结果就好了一些,那这个是一个招数
- 还有一个招数叫做**Early stopping**,Early stopping,Regularization跟Dropout,都是之后课程还会讲到的东西,那这三件事情在作业一的程式裡面,这个Early stopping其实是有的,助教有写在它的code裡面,所以不知道这是什麼也没有关係,反正你直接执行sample code,裡面就有了,Regularization,助教留下了一个空格给大家填,那你不知道什麼是regularization,没有关係,反正你可以过得了middle的baseline,那如果你想做得更好,也许你可以先自己survey一下,regularization是什麼,看有没有办法自己写
- **Dropout**,这是另外一个在Deep Learning裡面,常用来限制模型的方法,那这个之后还会再提到

但是我们也不要给太多的限制,為什麼不能给模型太多的限制呢

假设我们现在给模型更大的限制说,我们假设我们的模型,一定是Linear的Model,一定是写成y=a+bx,那你的model它能够產生的function,就一定是一条直线

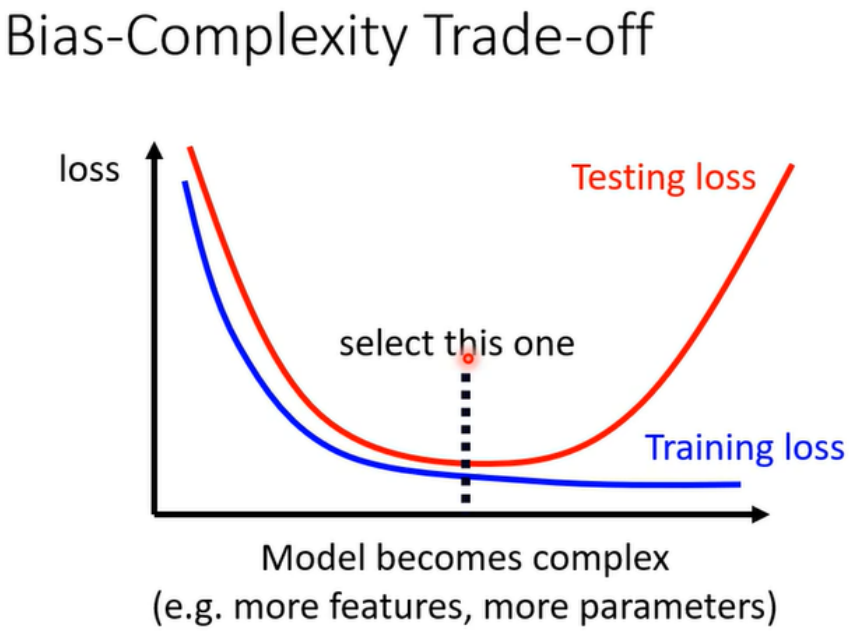


今天给三个点,没有任何一条直线,可以同时通过这三个点,但是你只能找到一条直线,这条直线跟这些点比起来,它们的距离是比较近的,但是你没有办法找到任何一条直线,同时通过这三个点,这个时候你的模型的限制就太大了,你在测试集上就不会得到好的结果

但是 这个不是overfitting,因為你又回到了**model bias**的问题,所以你现在这样在这个情况下,这个投影片的case上面你结果不好,并不是因為overfitting了,而是因為你给你模型太大的限制,大到你有了model bias的问题。

所以你就会发现说,这边**產生了一个矛盾的状况**,今天你让你的模型的复杂的程度,或这样让你的模型的弹性越来越大,但是什麼叫做复杂的程度,什麼叫做弹性,在今天这堂课裡面,我们其实都没有给明确的定义,只给你一个概念上的叙述,那在下下週的课

程裡面,你会真的认识到,什麼叫做一个模型很复杂,什麼叫做一个模型有弹性,怎麼真的衡量一个模型的弹性,复杂的程度有多大,那今天我们先用直观的来了解

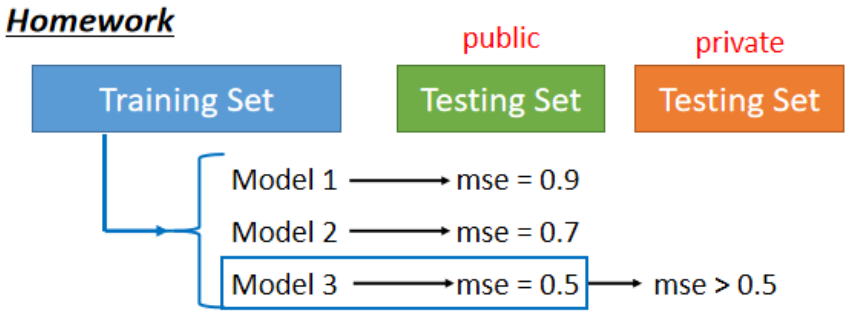


所谓比较复杂就是,**它可以包含的function比较多,它的参数比较多,这个就是一个比较复杂的model**

那一个比较复杂的model,如果你看它的training的loss,你会发现说 **随著model越来越复杂,Training的loss可以越来越低**,但是testing的时候呢,**当model越来越复杂的时候,刚开始,你的testing的loss会跟著下降,但是当复杂的程度,超过某一个程度以后,Testing的loss就会突然暴增了**

那这就是因为说,当你的model越来越复杂的时候,复杂到某一个程度,overfitting的状况就会出现,所以你在training的loss上面,可以得到比较好的结果,那在Testing的loss上面,你会得到比较大的loss,那我们当然期待说,我们可以选一个中庸的模型,不是太复杂的 也不是太简单的,刚刚好可以在训练集上,给我们最好的结果,给我们最低的loss,给我们最低的testing loss,怎麼选出这样的model呢

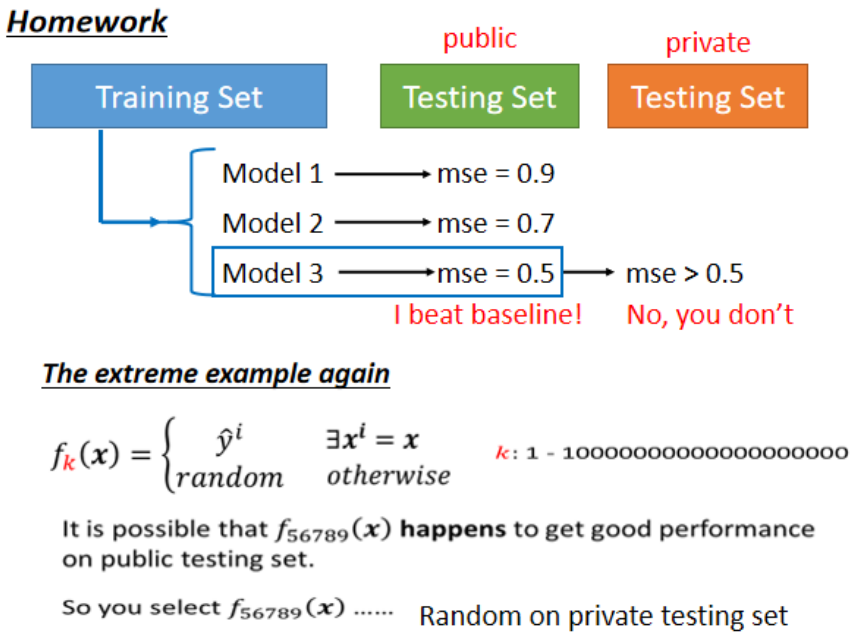
一个很直觉的 你很有可能,没有人告诉你要怎麼做的话,你可能很直觉就会这麼做的,做法就是说,这个kaggle不是立刻上传,就可以知道答案了吗



所以假设我们有三个模型,它们的复杂的程度不太一样,我不知道要选哪一个模型才会刚刚好,在测试集上得到最好的结果,因为你选太复杂的就overfitting,选太简单的有model bias的问题,那怎麼选一个不偏不倚的,不知道 那怎麼办

把这三个模型的结果都跑出来,然后上传到kaggle上面,你及时的知道了你的分数,看看哪个分数最低,那个模型显然就是最好的模型

但是并不建议你这麼做,為什麼不建议你这麼做呢



我们再举一个极端的例子,我们再把刚才那个极端的例子拿出来,假设现在有一群model,这一群model不知道為什麼都非常废,它们每一个model產生出来的,都是一无是处的function,我们有一到一兆个model,这一到一兆个model不知道為什麼,learn出来的function,都是一无是处的function

它们会做的事情就是,**训练集裡面有的资料就把它记下来,训练集没看过的,就直接output随机的结果**

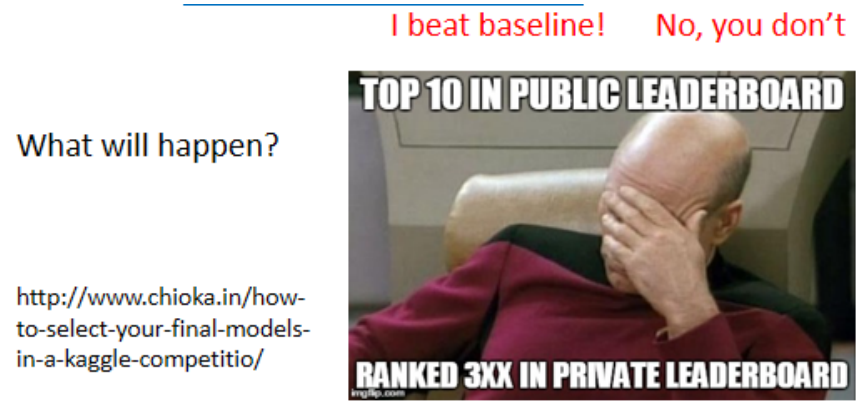
那你现在有一兆个模型,那你再把这一兆个模型的结果,通通上传到kaggle上面,你就得到一兆个分数,然后看这一兆的分数裡面,哪一个结果最好,你就觉得那个模型是最好的

那虽然说每一个模型,它们在这个Testing data上面,这个testing data它都没有看过啊,所以它输出的结果都是随机的,但虽然在testing data上面,输出的结果都是随机的,但是**你不断的随机,你总是会找到一个好的结果**,所以也许编号五六七八九的那个模型,它找出来的function,正好在testing data上面,就给你一个好的结果,那你就会很高兴觉得说,这个model编号五六七八九,是个好model,这个好model得到一个好function

虽然它其实是随机的 但你不知道,但这个好function,在这个testing data上面,给我们好的结果,所以你就觉得说 这个结果不错,就这样 我就**选这一个model,这个function,当作我们最后上传的结果,当作我最后要用在,private testing set上的结果**

但是**如果你这样做,往往就会得到非常糟的结果**,因為这个model毕竟是随机的,它**恰好在public的testing set data上面得到一个好结果,但是它在private的testing set上,可能仍然是随机的**

我们这个testing set,分成public的set跟private的set,你在看分数的时候 你只看得到public的分数， private的分数要deadline以后才知道,但假设你在挑模型的时候,你完全看你在public set上面的,也就leaderboard上的分数,来选择你的模型的话,你可能就会这个样子：你在public的leaderboard上面排前十,但是deadline一结束,你就心态就崩了这样,你就掉到三百名之外,而且我们这修课的人这麼多,你搞不好会掉到一千名之外,也说不定。



而且这件事情并不是传说,并没有夸饰,每年都会有这样子的状况发生,那因為今年我们会看public,就是说我们在算分数的时候,你在public上面的结果好,还是会给你一点分数,我们不是只看private的分数而已,是public跟private的分数都看,那过去有些学期,是只看private的分数的时候,发生这种状况,你心态就会整个崩掉这样子,你就会非常非常的郁闷

那為什麼我们要把testing的set,分成public跟private呢,為什麼我们不能,就通通都分public就好呢,為什麼要為难大家呢,為什麼要让大家疑神疑鬼,不知道自己private上的结果是什麼

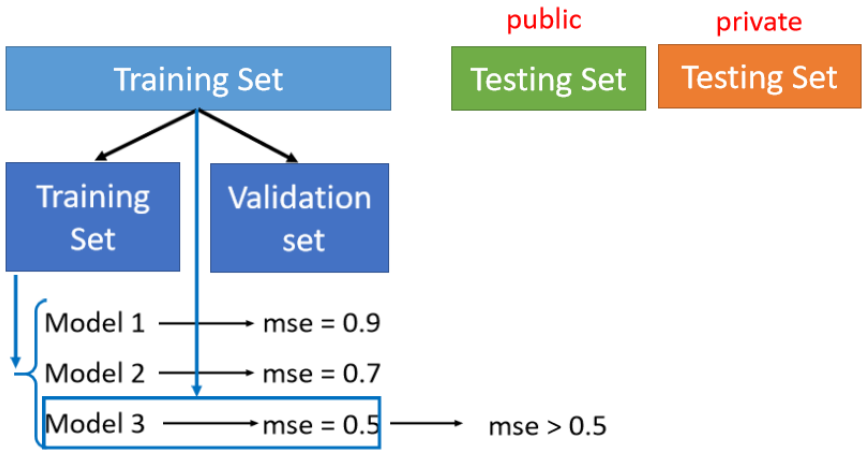
你自己想想看,假设所有的data都是public,那我刚才说,就算是一个一无是处的Model,得到了一无是处的function,它也有可能在public的data上面,得到好的结果,如果我们今天只有public的testing set,没有private的testing set,那你就回去写一个程式,不断random產生输出就好,然后不断把random的输出,上传到kaggle,然后看你什麼时候,可以random出一个好的结果,那这个作业就结束了

这个显然没有意义,显然不是我们要的,而且因為如果今天 你想想看,然后这边有另外一个有趣的事情就是,你知道因為今天如果,public的testing data是公开的,你可以知道public的,testing data的结果,那你就算是一个很废的模型,產生了很废的function,也可能得到非常好的结果

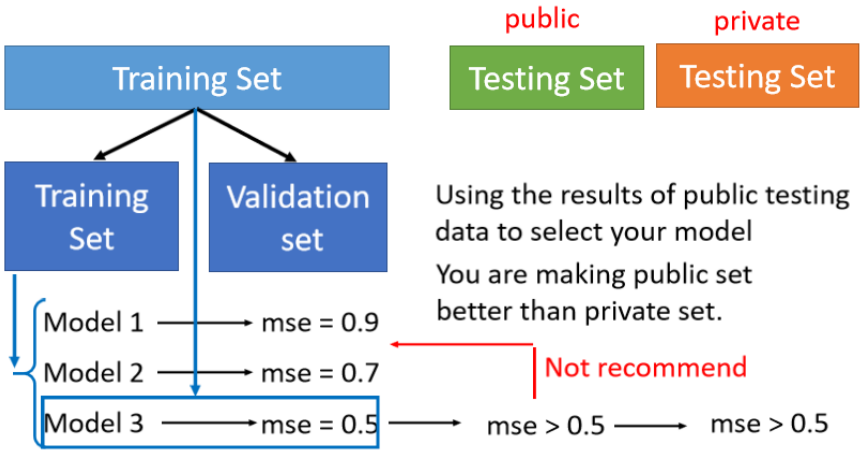
所以讲了这麼多,只是想要告诉大家说,我们為什麼要切public的testing set,我為什麼要切private的testing set,然后你其实不要花,不要用你public的testing set,去调你的模型,因為你可能会在,private的testing set上面,得到很差的结果,那不过因為今年,你在public set上面的,好的结果也有算分数,所以怎麼办呢,為了避免你 就你可能会说,好 那我放弃private set的结果,就只拿public set的结果,然后不断地產生随机的结果,去上传到Kaggle来,然后看看说能不能够,正好随机出一个好的结果,為了避免你浪费时间做这件事情,所以有每日上传的限制,让你不会说,我拿很废的模型只產生随机的结果,不断的测试public的testing的score

Cross Validation

那到底要怎麼做才选择model,才是比较合理的呢,那界定的方法是这个样子的,那助教程式裡面也都帮大家做好了,你要**把Training的资料分成两半,一部分叫作Training Set,一部分是Validation Set**



刚才助教程式裡面已经看到说,有90%的资料放在Training Set裡面,有10%的资料,会被拿来做Validation Set,你在Training Set上训练出来的模型,**你在Validation Set上面,去衡量它们的分数,你根据Validation Set上面的分数,去挑选结果**,再把这个结果上传到Kaggle上面,去看看你得到的public的分数,那因為你在挑分数的时候,是用Validation Set来挑你的model,所以你的public的Testing Set的分数,就可以反应你的,private Testing Set的分数,就比较不会得到说,在public上面结果很好,但是在private上面结果很差,这样子的状况



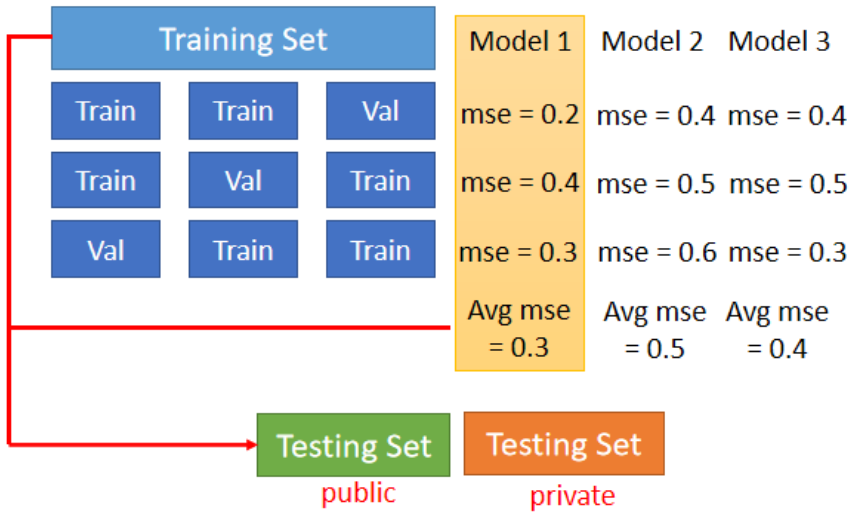
当我知道说,其实你看到public的结果以后,你就会去想要调它,你看到你现在弄了一堆模型,然后用Validation Set检查一下,找了一个模型放到public set上以后,发现结果不好,你其实不太可能不根据这一个结果,去调整你的模型,但是假设这一个route做太多次,你根据你的,public Testing Set上的结果,去调整你的model太多次,你就又有可能fit在,你的public Testing Set上面,然后在private Testing Set上面,得到差的结果,不过还好反正我们有限制上传的次数,所以这个route,你也没有办法走太多次,可以避免你太过fit在,public的Testing Set上面的结果。

那我知道说今天因為,public的Testing Set上面的结果,是大家都可以看到的,然后很多人都会,然后名字你又可以随便乱取,所以假设有一个人洗到第一名的话,他就会非常的得意,他就把自己的名字改成一些什麼,我第一次试就第一名了,或是 我其实只是个旁听,那其实他不是旁听的,那他改成说,我其实只是个旁听的,随便做就第一名了,那这个时候你就会觉得很紧张,尤其他如果是你认识的,隔壁小毛得到第一名,到处耀武扬威的时候,你就会开始有点紧张,你就会说,等一下 你不要得意,我等一下就去把你刷下来这样,那这个时候你要不要理他呢,你不要理他,根据过去的经验,就在public leaderboard上排前几名的,往往private很容易就惨掉这样子,所以在public的Testing上面,得到太好的结果,也不用高兴得太早,其实 最好的做法,就是用Validation loss,最小的直接挑就好了,就是你不要去管,你的public Testing Set的结果 这样,那我知道说在实作上,你不太可能这麼做,因為public set的结果你有看到,所以它对你的模型的选择,可能还是会有些影响的,但是你要越少去看那个,public Testing Set的结果越好

线上直播的同学,我复述一下刚才那个同学的问题,他的问题是说,所以我们不能去看,public Testing Set的结果吗,理想上是,理想上你就用Validation Set挑就好,然后上传以后 怎样就是怎样,有过那个strong basseline以后,就不要再去动它了,那这样子就可以避免,你overfit在Testing Set上面,好 那但是这边会有一个问题,就是怎麼分Training Set,跟Validation Set呢,那如果在助教程式裡面,就是随机分的,但是你可能会说,搞不好我这个分 分得不好啊,搞不好我分到很奇怪的Validation Set,会导致我的结果很差,

N-fold Cross Validation

如果你有这个担心的话,那你可以用N-fold Cross Validation,



N-fold Cross Validation就是你先**把你的训练集切成N等份**,在这个例子裡面我们切成三等份,切完以后,你拿其中**一份当作Validation Set**,**另外两份当Training Set**,然后这件事情你要**重复三次**

也就是说,你先第一份第二份当Train,第三份当Validation,然后第一份第三份当Train,第二份当Validation,第一份当Validation,第二份第三份当Train

然后接下来 你有三个模型,你不知道哪一个是好的,你就把这三个模型,在这三个setting下,在这三个Training跟Validation的,data set上面,通通跑过一次,**然后把这三个模型,在这三种状况的结果都平均起来**,把每一个模型在这三种状况的结果,都平均起来,再看看谁的结果最好

那假设现在model 1的结果最好,你用这三个fold得出来的结果是,这个model 1最好,然后你再把model 1,用在全部的Training Set上,然后训练出来的模型,再用在Testing Set上面,好 那这个是N-fold Cross Validation,好 那这个就是这门课前期的攻略,它可以带你打赢前期所有的副本

那接下来也许你要问的一个问题是,上週结束的时候,不是讲到预测2/26,也就是上週五的观看人数吗,到底结果做得怎麼样

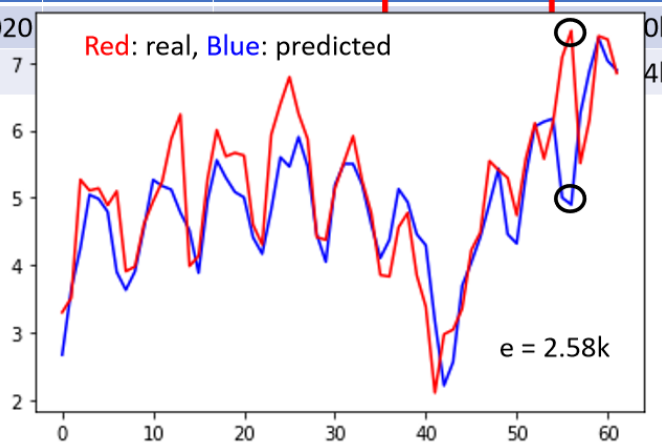
Let's predict no. of views of 2/26!

	1 layer	2 layer	3 layer	4 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k
2021	0.43k	0.39k	0.38k	0.44k

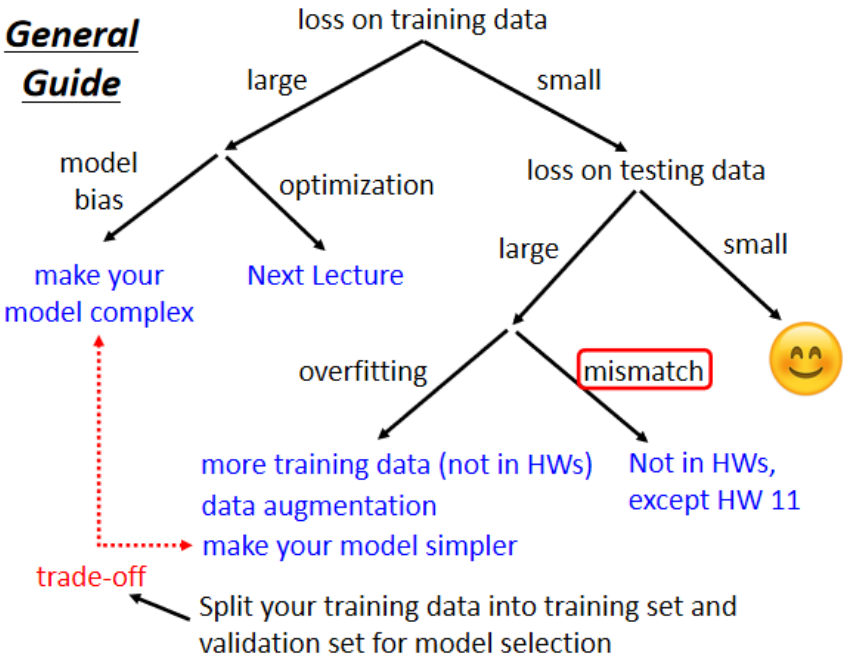
那这个就是我们要做的结果,上週比较多人选了三层的network,所以我们就把三层的network,拿来测试一下,以下是测试的结果,我们就没有再调参数了,大家决定用三层的就是下好离手了,就直接用上去了

Let's predict no. of views of 2/26!

	1 layer	2 layer	3 layer	4 layer
2017 – 2020				0k
2021				4k



得到的结果是这个样子了,这个图上 这个横轴就是从,2021年的1月1号开始 一直往下,然后红色的线是真实的数字,蓝色的线是预测的结果,2/26在这边 这个是今年2021年,观看人数最高的一天了,那机器的预测怎样呢,哇 非常的惨 差距非常的大,差距有2.58k这麼多,感谢大家 为了让这个模型不準,上週五花了很多力气,去点了这个video,所以这一天是,今年观看人数最多的一天,那你可能开始想说,那别的模型怎麼样呢,其实我也跑了一层二层跟四层的看看,所有的模型 都会惨掉,两层跟三层的错误率都是2点多k,其实四层跟一层比较好,都是1.8k左右,但是这四个模型不约而同的,觉得2/26应该是个低点,但实际上2/26是一个公值,那模型其实会觉得它是一个低点,也不能怪它,因為根据过去的资料,礼拜五就是没有人要学机器学习,礼拜五晚上大家都出去玩了对不对,礼拜五的观看人数是最少了,但是2/26出现了反常的状况,好 那这个就不能怪模型了,那我觉得出现这种状况,应该算是另外一种错误的形式,这种错误的形式,我们这边叫作mismatch。



那也有人会说,mismatch也算是一种Overfitting,这样也可以,这都只是名词定义的问题,那我这边想要表达的事情是,**mismatch它的原因跟overfitting,其实不一样**,一般的overfitting,你可以用搜集更多的资料来克服,但是**mismatch意思是说,你今天的训练集跟测试集,它们的分佈是不一样的**

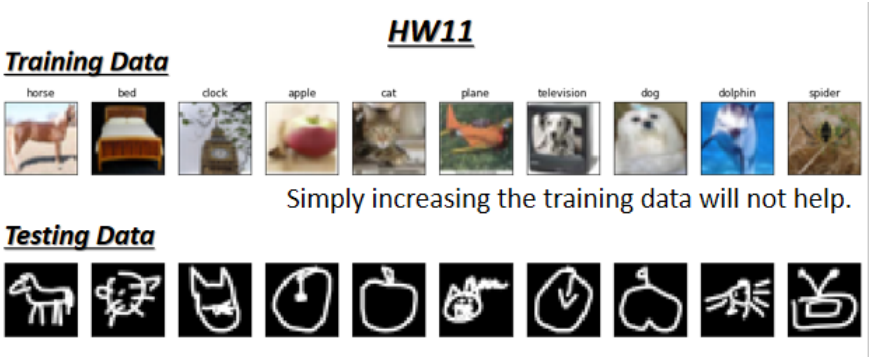
Mismatch

- Your training and testing data have different distributions. Be aware of how data is generated.

在训练集跟测试集,分佈是不一样的时候,你训练集再增加,其实也没有帮助了,那其实在多数的作业裡面,我们不会遇到这种mismatch的问题,我们都有把题目设计好了,所以资料跟测试集它的分佈差不多

举例来说 以刚才作业一的,Covid19為例的话,假设我们今天资料在,分训练集跟测试集的时候,我们说**2020年的资料是训练集,2021年的资料是测试集**,那mismatch的问题可能就很严重了,这个我们其实有试过了 试了一下,如果今天用2020年当训练集,2021年当测试集,你就怎麼做都是惨了 就做不起来,训练什麼模型都会惨掉

因為2020年的资料跟2021年的资料,它们的背后的分佈其实都是不一样,所以你拿2020年的资料来训练,在2021年的作业一的资料上,你根本就预测不準,所以后来助教是用了别的方式,来分割训练集跟测试集,好 所以我们多数的作业,都不会有这种mismatch的问题,那除了作业十一。



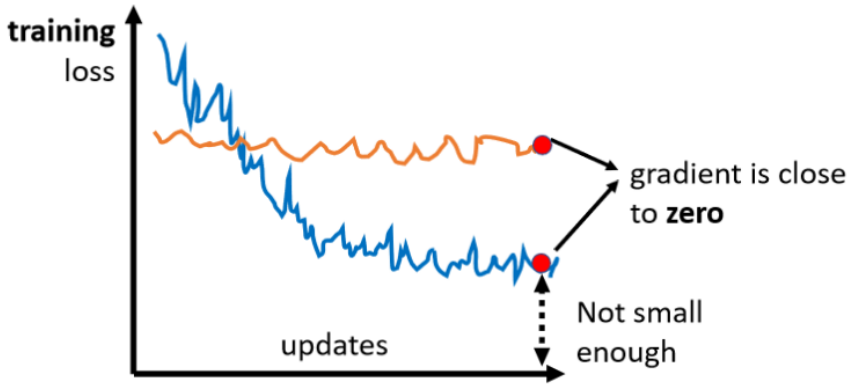
因為作业十一就是,针对mismatch的问题来设计的,作业十一也是一个影像分类的问题,这是它的训练集,看起来蛮正常的,但它测试集就是长这样子了,所以你知道这个时候,这个时候增加资料哪有什麼用呢,增加资料,你也没有办法让你的模型做得更好,所以这种问题要怎麼解决,那犹待作业十一的时候再讲,好 那你可能会问说 我怎麼知道,现在到底是不是mismatch呢,那我觉得知不知道是mismatch,那就要看你对这个资料本身的理解了,你可能要对你的训练集跟测试集,的產生方式有一些理解,你才能判断说,它是不是遇到了mismatch的状况,好 那这个就是我们作业的攻略,

When gradient is small

Critical Point

Training Fails because

现在我们要讲的是Optimization的部分,所以我们要讲的东西基本上跟Overfitting没有什麼太大的关联,我们只讨论Optimization的时候,怎麼把gradient descent做得更好,那為什麼Optimization会失败呢?



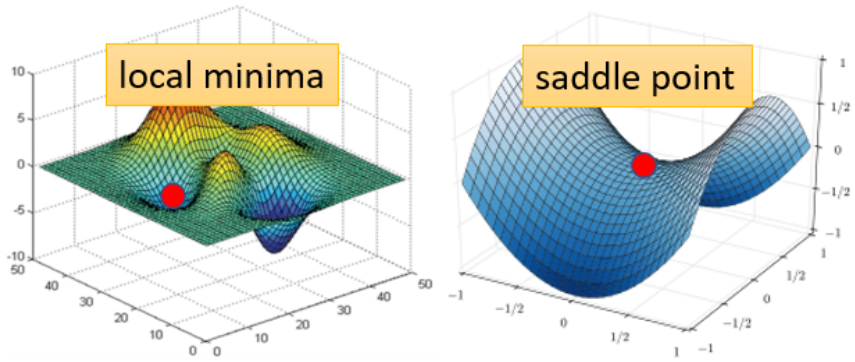
你常常在做Optimization的时候,你会发现,**随著你的参数不断的update,你的training的loss不会再下降**,但是你对这个loss仍然不满意,就像我刚才说的,你可以把deep的network,跟linear的model,或比较shallow network 比较,发现说它没有做得更好,所以你觉得deepnetwork,没有发挥它完整的力量,所以Optimization显然是有问题的

但有时候你会甚至发现,一开始你的model就train不起来,一开始你不管怎麼update你的参数,你的loss通通都掉不下去,那这个时候到底发生了什麼事情呢?

过去常见的一个猜想,是因為我们现在走到了一个地方,**这个地方参数对loss的微分為零**,当你的参数对loss微分為零的时候,gradient descent就没有办法再update参数了,这个时候training就停下来了,loss当然就不会再下降了。

讲到gradient為零的时候,大家通常脑海中最先浮现的,可能就是==local minima==,所以常有人说做deep learning,用gradient descent会卡在local minima,然后所以gradient descent不work,所以deep learning不work。

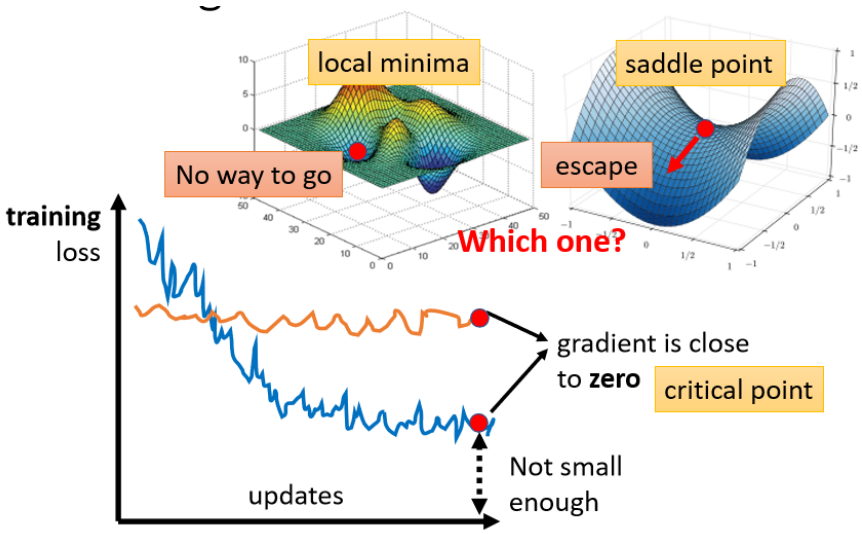
但是如果有一天你要写,跟deep learning相关paper的时候,你千万不要讲卡在local minima这种事情,别人会觉得你非常没有水準,為什麼



因為**不是只有local minima的gradient是零**,还有其他可能会让gradient是零,比如说 ==saddle point==,所谓的saddle point,其实就是gradient是零,但是不是local minima,也不是local maxima的地方,像在右边这个例子裡面 红色的这个点,它在左右这个方向是比较高的,前后这个方向是比较低的,它就像是一个马鞍的形状,所以叫做saddle point,那中文就翻成鞍点

像saddle point这种地方,它也是gradient為零,但它不是local minima,那像这种gradient為零的点,统称為==critical point==,所以**你可以说你的loss,没有办法再下降,也许是因為卡在了critical point,但你不能说是卡在local minima,因為saddle point也是微分為零的点**

但是今天如果你发现你的gradient,真的很靠近零,卡在了某个critical point,我们有没有办法知道,到底是local minima,还是saddle point? 其实是有办法的



為什麼我们想要知道到底是卡在local minima,还是卡在saddle point呢

- 因為如果是**卡在local minima,那可能就没有路可以走了**,因為四周都比较高,你现在所在的位置已经是最低的点,loss最低的点了,往四周走 loss都会比较高,你会不知道怎麼走到其他的地方去
- 但saddle point就比较没有这个问题,如果你今天是**卡在saddle point的话,saddle point旁边还是有路可以走的,**还是有路可以让你的loss更低的,你只要逃离saddle point,你就有可能让你的loss更低

所以鉴别今天我们走到,critical point的时候,到底是local minima,还是saddle point,是一个值得去探讨的问题,那怎麼知道今天一个critical point,到底是属於local minima,还是saddle point呢?

Warning of Math

这边需要用到一点数学,以下这段其实没有很难的数学,就只是微积分跟线性代数,但如果你没有听懂的话,以下这段skip掉是没有关系的

那怎麼知道说一个点,到底是local minima,还是saddle point呢?

你要知道我们loss function的形状,可是我们怎麼知道,loss function的形状呢,network本身很复杂,用复杂network算出来的loss function,显然也很复杂,我们怎麼知道loss function,长什麼样子,虽然我们没有办法完整知道,整个loss function的样子

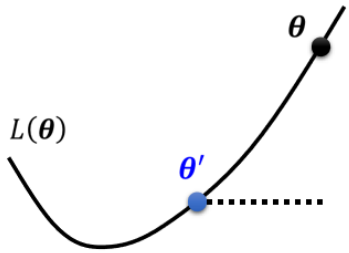
Tayler Series Approximation

但是如果给定某一组参数,比如说蓝色的这个$θ'$,在$θ'$附近的loss function,是有办法被写出来的,它写出来就像是这个样子

Tayler Series Approximation

$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T H(\boldsymbol{\theta} - \boldsymbol{\theta}')$$



所以这个$L(θ)$完整的样子写不出来,但是它在$θ'$附近,你可以用这个式子来表示它,这个式子是,Tayler Series Appoximation泰勒级数展开,这个假设你在微积分的时候,已经学过了,所以我就不会细讲这一串是怎麼来的,但我们就只讲一下它的概念,这一串裡面包含什麼东西呢?

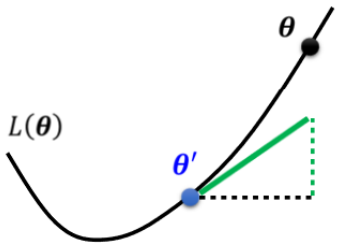
- 第一项是$L(θ')$,就告诉我们说,当$θ$跟$θ'$很近的时候,$L(θ)$应该跟$L(θ')$还蛮靠近的
- 第二项是$(θ-θ')^Tg$

$L(\theta)$ around $\theta = \theta'$ can be approximated below

$$L(\theta) \approx L(\theta') + (\theta - \theta')^T g + \frac{1}{2}(\theta - \theta')^T H (\theta - \theta')$$

Gradient g is a ***vector***

$$g = \nabla L(\theta') \quad g_i = \frac{\partial L(\theta')}{\partial \theta_i}$$



g是一个向量,这个g就是我们的gradient,我们用绿色的这个g来代表gradient,这个**gradient会来弥补,$θ'$跟$θ$之间的差距**,我们虽然刚才说$θ'$跟$θ$,它们应该很接近,但是中间还是有一些差距的,那这个差距,第一项我们用这个gradient,来表示他们之间的差距,有时候gradient会写成$∇L(θ')$,这个地方的g是一个向量,**它的第i个component,就是θ的第i个component对L的微分**,光是看g还是没有办法,完整的描述L(θ),你还要看第三项

- 第三项跟Hessian有关,这边有一个$H $

$L(\theta)$ around $\theta = \theta'$ can be approximated below

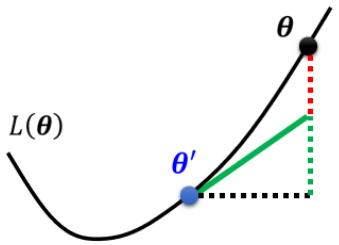
$$L(\theta) \approx L(\theta') + (\theta - \theta')^T g + \frac{1}{2}(\theta - \theta')^T H (\theta - \theta')$$

Gradient g is a ***vector***

$$g = \nabla L(\theta') \quad g_i = \frac{\partial L(\theta')}{\partial \theta_i}$$

Hessian H is a ***matrix***

$$H_{ij} = \frac{\partial^2}{\partial \theta_i \partial \theta_j} L(\theta')$$

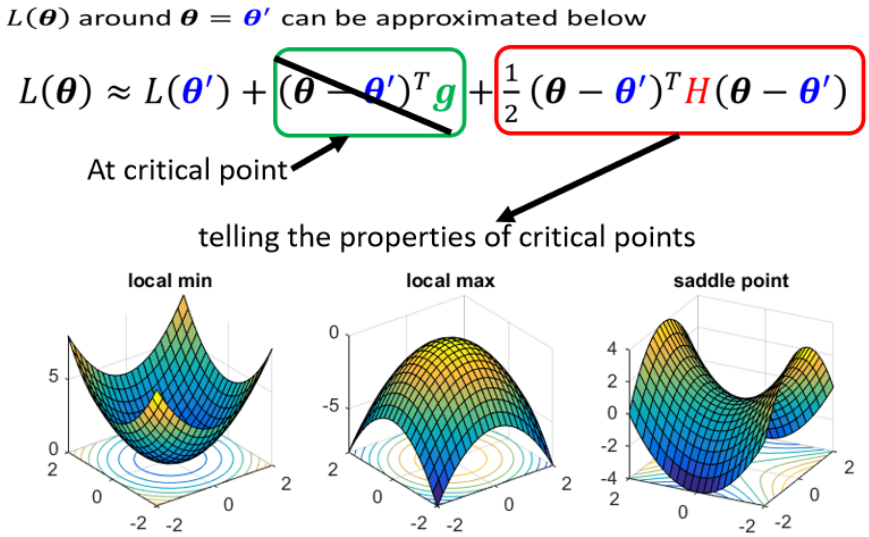


这个H叫做Hessian,它是一个矩阵,这个第三项是,再$(θ-θ')^TH(θ-θ')$,所以第三项会再补足,再加上gradient以后,与真正的L(θ)之间的差距.**H裡面放的是L的二次微分,它第i个row,第j个column的值,就是把θ的第i个component,对L作微分,再把θ的第j个component,对L作微分,再把θ的第i个component,对L作微分,做两次微分以后的结果** 就是这个$H_i{_j}$

如果这边你觉得有点听不太懂的话,也没有关係,反正你就记得这个$L(θ)$,这个loss function,这个error surface在$θ'$附近,可以写成这个样子,这个式子跟两个东西有关係,**跟gradient有关係,跟hessian有关係,gradient就是一次微分,hessian就是裡面有二次微分的项目**

Hession

那如果我们今天走到了一个critical point,意味著gradient為零,也就是绿色的这一项完全都不见了



g**是一个zero vector,绿色的这一项完全都不见了**,只剩下红色的这一项,所以当在critical point的时候,这个loss function,它可以被近似为$L(θ')$,加上红色的这一项

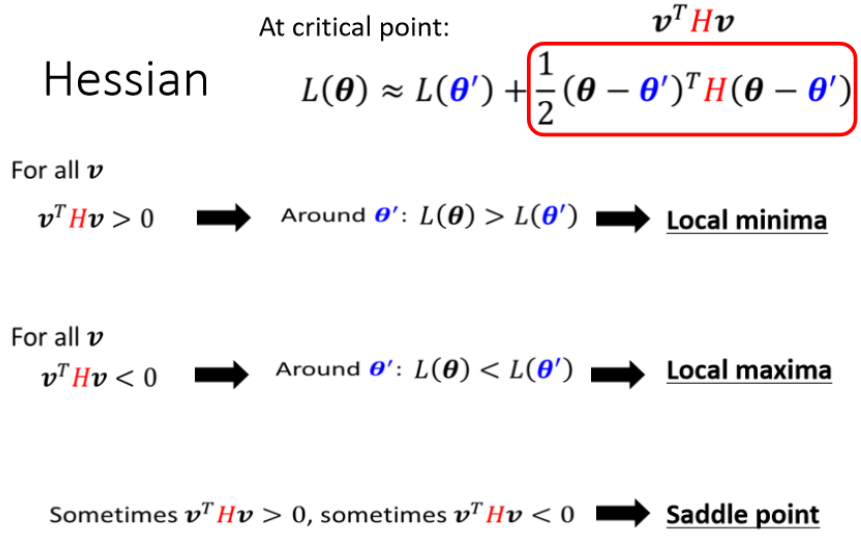
我们可以**根据红色的这一项来判断**,在$θ'$附近的error surface,到底长什麼样子

知道error surface长什麼样子,我就可以判断

$θ'$它是一个==local minima==,是一个==local maxima==,还是一个==saddle point==

我们可以靠这一项来了解,这个error surface的地貌,大概长什麼样子,知道它地貌长什麼样子,我们就可以知道说,现在是在什麼样的状态,这个是Hessian

那我们就来看一下怎麼根据Hessian,怎麼根据红色的这一项,来判断θ'附近的地貌

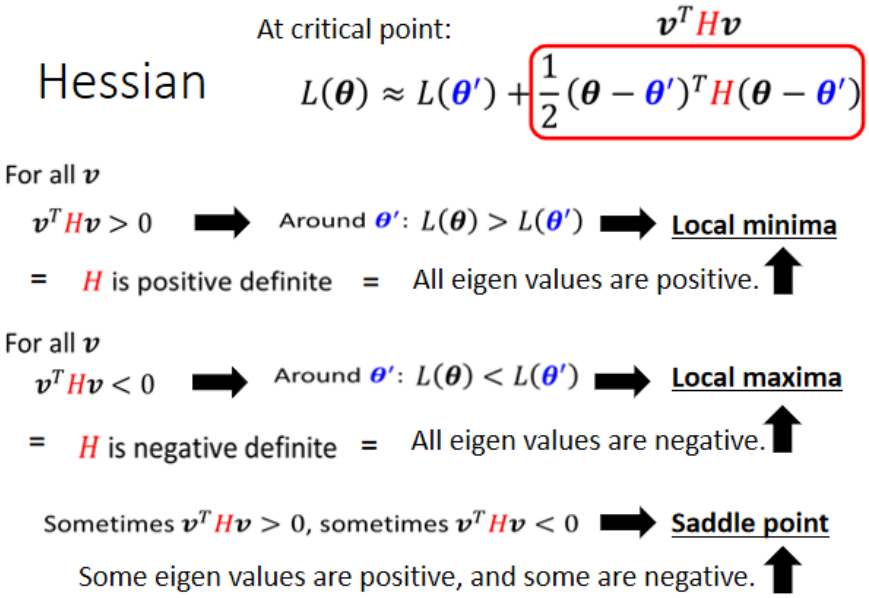


我们现在為了等一下符号方便起见,我们**把$(θ-θ')$用v这个向量来表示**

- 如果今天对任何可能的v,**v^THv都大於零**,也就是说 现在θ不管代任何值,v可以是任何的v,也就是θ可以是任何值,不管θ代任何值,**红色框框裡面通通都大於零**,那意味著说 $L(θ)>L(θ')$。$L(θ)$不管代多少 只要在$θ'$附近,$L(θ)$都大於$L(θ')$,**代表$L(θ')$是附近的一个最低点,所以它是local minima**

- 如果今天反过来说,对所有的v而言,**v^THv都小於零,也就是红色框框裡面永远都小於零**,也就是说$θ$不管代什麼值,红色框框裡面都小於零,意味著说$L(θ)<L(θ')$,**代表$L(θ')$是附近最高的一个点,所以它是local maxima**
- 第三个可能是假设,**v^THv,有时候大於零 有时候小於零**,你代不同的v进去 代不同的θ进去,红色这个框框裡面有时候大於零,有时候小於零,意味著说在θ'附近,有时候L(θ)>L(θ') 有时候L(θ)<L(θ'),在L(θ')附近,有些地方高 有些地方低,这意味著什麼,**这意味著这是一个saddle point**

但是你这边是说我们要代所有的v,去看v^THv是大於零,还是小於零.我们怎麼有可能把所有的v,都拿来试试看呢,所以有一个更简便的方法,去确认说这一个条件或这一个条件,会不会发生.



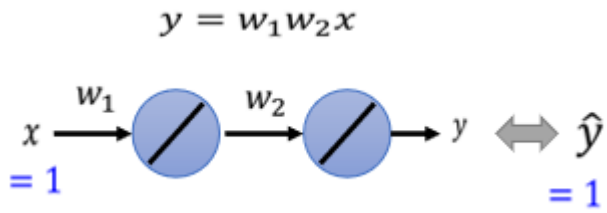
这个就直接告诉你结论,线性代数理论上是有教过这件事情的,如果今天对所有的v而言,v^THv都大於零,那这种矩阵叫做**positive definite 正定矩阵**,positive definite的矩阵,**它所有的eigen value特征值都是正的**

所以如果你今天算出一个hessian,你不需要把它跟所有的v都乘看看,你只要去直接看这个H的eigen value,如果你发现

- **所有eigen value都是正的**,那就代表说这个条件成立,就v^THv,会大於零,也就代表说是一个local minima。所以你从hessian metric可以看出,它是不是local minima,你只要算出hessian metric算完以后,看它的eigen value发现都是正的,它就是local minima。
- 那反过来说也是一样,如果今天在这个状况,对所有的v而言,v^THv小於零,那H是negative definite,那就代表所有**eigen value都是负的**,就保证他是local maxima
- **那如果eigen value有正有负**,那就代表是saddle point,

那假设在这裡你没有听得很懂的话,你就可以记得结论,**你只要算出一个东西,这个东西的名字叫做hessian,它是一个矩阵,这个矩阵如果它所有的eigen value,都是正的,那就代表我们现在在local minima,如果它有正有负,就代表在saddle point。**

那如果刚才讲的,你觉得你没有听得很懂的话,我们这边举一个例子

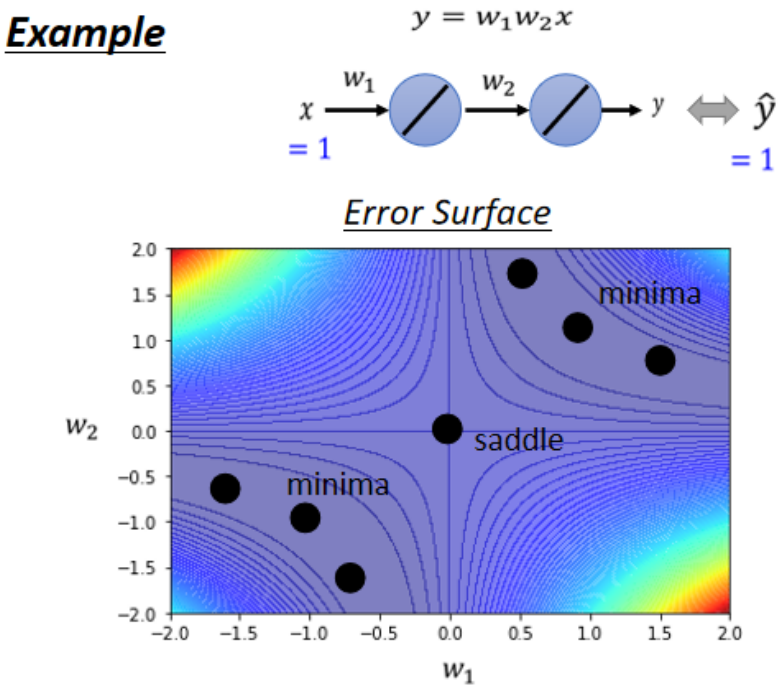


我们现在有一个史上最废的network,输入一个x,它只有一个neuron， 乘上w_1,而且这个neuron,还没有activation function,所以x乘上w_1以后 之后就输出,然后再乘上w_2 然后就再输出,就得到最终的数据就是y.总之这个function非常的简单

$$ y= $w_1 \times w_2 \times x$

$$ 我们有一个史上最废的training set,这个data set说,我们只有一笔data,这笔data是x,是1的时候,它的level是1 所以输入1进去,你希望最终的输出跟1越接近越好

而这个史上最废的training,它的error surface,也是有办法直接画出来的,因为反正只有两个参数 w_1 w_2,连bias都没有,假设没有bias,只有w_1跟w_2两个参数,这个network只有两个参数 w_1跟w_2,那我们可以穷举所有w_1跟w_2的数值,算出所有w_1 w_2数值所代来的loss,然后就画出error surface 长这个样



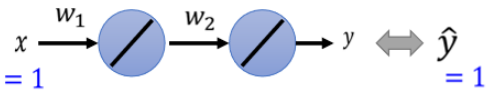
四个角落loss是高的,好 那这个图上你可以看出来说,有一些critical point,这个黑点点的地方(0,0),**原点的地方是critical point**,然后事实上,**右上三个黑点也是一排critical point,左下三个点也是一排critical point**

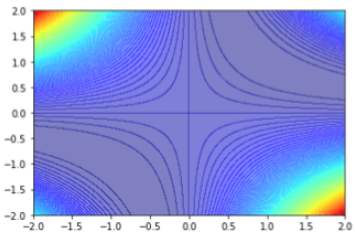
如果你更进一步要分析,他们是saddle point,还是local minima的话,那圆心这个地方,**原点这个地方 它是saddle point**,為什麼它是saddle point呢

你往左上这个方向走 loss会变大,往右下这个方向走 loss会变大,往左下这个方向走loss会变小,往右下这个方向走 loss会变小,它是一个saddle point

而这两群critical point,它们都是local minima,所以这个山沟裡面,有一排local minima,这一排山沟里面有一排local minima,然后在原点的地方,有一个saddle point,这个是我们把error surface,暴力所有的参数,得到的loss function以后,得到的loss的值以后,画出error surface,可以得到这样的结论

现在假设如果不暴力所有可能的loss,如果要直接算说一个点,是local minima,还是saddle point的话 怎麼算呢





$$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$$

$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2) = 0$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1) = 0$$

Critical point: $w_1 = 0, w_2 = 0$

我们可以把loss的function写出来,这个loss的function 这个L是

$$ L=(\hat{y}-w_1 w_2 x)^2

$$ 正确答案 ŷ减掉model的输出,也就是w_1 w_2x,这边取square error,这边**只有一笔data,所以就不会summation over所有的training data**,因為反正只有一笔data,x代1 ŷ代1,我刚才说过只有一笔训练资料最废的,所以只有一笔训练资料,所以loss function就是$L=(\hat{y}-w_1 w_2 x)^2$,那你可以把这一个loss function,它的gradient求出来,w_1对L的微分,w_2对L的微分写出来是这个样子

$$ \frac{∂L}{∂w_1 }=2(1-w_1 w_2)(-w_2)

$$

$$ \frac{∂L}{∂w_2 }=2(1-w_1 w_2)(-w_1)

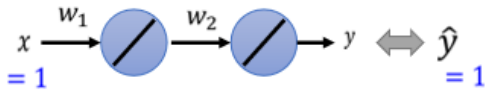
$$

这个东西

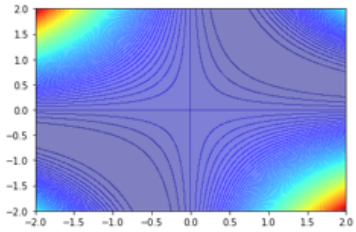
$$ \begin{bmatrix} \frac{∂L}{∂w_1 }\\ \frac{∂L}{∂w_2 } \end{bmatrix}

$$ 就是所谓的g,所谓的gradient,什麼时候gradient会零呢,什麼时候会到一个critical point呢?

举例来说 如果w_1=0 w_2=0,就在圆心这个地方,如果w_1代0 w_2代0,w_1对L的微分 w_2对L的微分,算出来就都是零 就都是零,这个时候我们就知道说,原点就是一个critical point,但**它是local maxima,它是local maxima,local minima,还是saddle point呢,那你就要看hessian才能够知道了**



$$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$$



$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2) = 0$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1) = 0$$

Critical point: $w_1 = 0, w_2 = 0$

$$H = \begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix} \quad \lambda_1 = 2, \lambda_2 = -2$$

Saddle point

$$\frac{\partial^2 L}{\partial {w_1}^2} = 2(-w_2)(-w_2) = 0 \qquad \frac{\partial^2 L}{\partial w_1 \partial w_2} = -2 + 4 w_1 w_2 = -2$$

$$\frac{\partial^2 L}{\partial w_2 \partial w_1} = -2 + 4 w_1 w_2 = -2 \qquad \frac{\partial^2 L}{\partial {w_2}^2} = 2(-w_1)(-w_1) = 0$$

当然 我们刚才已经暴力所有可能的w_1 w_2了,所以你已经知道说,它显然是一个saddle point,但是现在假设还没有暴力所有可能的loss,所以我们要看看能不能够用H,用Hessian看出它是什麼样的critical point,那怎麼算出这个H呢

H它是一个矩阵,这个矩阵裡面元素就是L的二次微分,所以这个矩阵裡面第一个row,第一个coloumn的位置,就是w_1对L微分两次,第一个row 第二个coloumn的位置,就是先用w_2对L作微分,再用w_1对L作微分,然后这边就是w_1对L作微分,w_2对L作微分,然后w_2对L微分两次,这四个值组合起来,就是我们的hessian,那这个hessian的值是多少呢

这个hessian的式子,我都已经把它写出来了,你只要把w_1=0 w_2=0代进去,代进去 你就得到在原点的地方,hessian是这样的一个矩阵

$$ \begin{bmatrix} {0}&-2\\ {-2}&0 \end{bmatrix}

$$ 这个hessian告诉我们,它是local minima,还是saddle point呢,那你就要看这个矩阵的eigen value,算一下发现,这个矩阵有两个eigen value,2跟-2 **eigen value有正有负,代表saddle point**

所以我们现在就是用一个例子,跟你操作一下 告诉你说,你怎麼从hessian看出一个点,它一个critical point 它是saddle point,还是local minima,

Don't afraid of saddle point

At critical point: $L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T H (\boldsymbol{\theta} - \boldsymbol{\theta}')$ ($\boldsymbol{v}^T H \boldsymbol{v}$)

Sometimes $\boldsymbol{v}^T H \boldsymbol{v} > 0$, sometimes $\boldsymbol{v}^T H \boldsymbol{v} < 0$ ➡ Saddle point

H may tell us parameter update direction!

如果今天你卡的地方是saddle point,也许你就不用那麼害怕了,因為如果你今天你发现,你停下来的时候,是因為saddle point 停下来了,那其实就有机会可以放心了

因為H它不只可以帮助我们判断,现在是不是在一个saddle point,它还指出了我们参数,可以update的方向,就之前我们参数update的时候,都是看gradient 看g,但是我们走到某个地方以后,发现g变成0了 不能再看g了,g不见了 gradient没有了,但如果是一个saddle point的话,还可以再看H,怎麼再看H呢,H怎麼告诉我们,怎麼update参数呢

At critical point: $L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}')$ ($\boldsymbol{v}^T H \boldsymbol{v}$)

Sometimes $\boldsymbol{v}^T H \boldsymbol{v} > 0$, sometimes $\boldsymbol{v}^T H \boldsymbol{v} < 0$ ➡ Saddle point

H may tell us parameter update direction!

$\boldsymbol{u}$ is an eigen vector of H

λ is the eigen value of $\boldsymbol{u}$

$\lambda < 0$

$$\boldsymbol{u}^T H \boldsymbol{u} = \boldsymbol{u}^T(\lambda \boldsymbol{u}) = \lambda \|\boldsymbol{u}\|^2$$

$\boldsymbol{u}^T H \boldsymbol{u} < 0$ $\quad \lambda \|\boldsymbol{u}\|^2 < 0$

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}') \Rightarrow L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}')$$

($\boldsymbol{u} = \boldsymbol{\theta}-\boldsymbol{\theta}'$)

$\boldsymbol{\theta}-\boldsymbol{\theta}' = \boldsymbol{u}$ $\quad \boldsymbol{\theta} = \boldsymbol{\theta}' + \boldsymbol{u}$ $\quad$ Decrease L

我们这边假设μ是H的eigenvector特征向量,然后$λ$是u的eigen value特征值。

如果我们把这边的v换成μ的话,我们把μ乘在H的左边,跟H的右边,也就是$\mu^TH\mu$, $H\mu$会得到$λ\mu$， 因為μ是一个eigen vector。H乘上eigen vector特征向量会得到特征向量λ eigen value乘上eigen vector即$λ\mu$

$$\boldsymbol{u}^T H \boldsymbol{u} = \boldsymbol{u}^T(\lambda \boldsymbol{u}) = \lambda \|\boldsymbol{u}\|^2$$

所以我们在这边得到uᵀ乘上λu,然后再整理一下,把uᵀ跟u乘起来,得到‖u‖²,所以得到λ‖u‖²

($\boldsymbol{v}^T H \boldsymbol{v}$)

$$\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}')$$

假设我们这边v,代的是一个eigen vector,我们这边θ减θ',放的是一个eigen vector的话,会发现说我们这个红色的项裡面,其实就是λ‖u‖²

$$\text{At critical point:}\quad L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \underbrace{\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}')}_{\boldsymbol{v}^T H \boldsymbol{v}}$$

Sometimes $\boldsymbol{v}^T H \boldsymbol{v} > 0$, sometimes $\boldsymbol{v}^T H \boldsymbol{v} < 0$ ➡ Saddle point

H may tell us parameter update direction!

$\boldsymbol{u}$ is an eigen vector of H

λ is the eigen value of $\boldsymbol{u}$

$\lambda < 0$

$$\Rightarrow \quad \underbrace{\boldsymbol{u}^T H \boldsymbol{u}}_{<0} = \boldsymbol{u}^T(\lambda \boldsymbol{u}) = \underbrace{\lambda \|\boldsymbol{u}\|^2}_{<0}$$

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}')^T H(\boldsymbol{\theta}-\boldsymbol{\theta}') \quad \Rightarrow \quad L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}')$$

$$\boldsymbol{\theta}-\boldsymbol{\theta}' = \boldsymbol{u} \qquad \boldsymbol{\theta} = \boldsymbol{\theta}' + \boldsymbol{u} \qquad \text{Decrease } L$$

那今天如果**λ小於零**,eigen value小於零的话,那λ‖u‖²就会小於零,因为‖u‖²一定是正的,所以eigen value是负的,那这一整项就会是**负的**,也就是u的transpose乘上H乘上u,它是负的,也就是**红色这个框裡是负的**

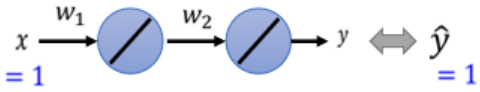
所以这意思是说假设$θ-θ'=\mu$,那这一项$(θ-θ')^TH(θ-θ')$就是负的,也就是$L(θ)<L(θ')$

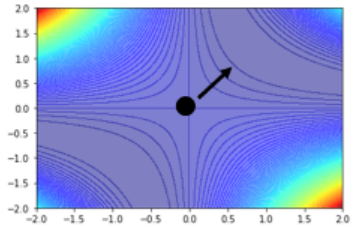
也就是说假设$θ-θ'=\mu$,也就是,**你在θ'的位置加上u,沿著u的方向做update得到θ,你就可以让loss变小**

因為根据这个式子,你只要θ减θ'等於u,loss就会变小,所以你今天只要让θ等於θ'加u,你就可以让loss变小,你只要沿著u,也就是eigen vector的方向,去更新你的参数 去改变你的参数,你就可以让loss变小了

所以虽然在critical point没有gradient,如果我们今天是在一个**saddle point**,你也不一定要惊慌,你只要**找出负的eigen value,再找出它对应的eigen vector,用这个eigen vector去加θ',就可以找到一个新的点,这个点的loss比原来还要低**

举具体的例子





$$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$$

$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2)$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1)$$

Critical point: $w_1 = 0, w_2 = 0$

$$H = \begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix} \quad \lambda_1 = 2, \lambda_2 = -2$$

Saddle point

$\lambda_2 = -2$ Has eigenvector $\boldsymbol{u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Update the parameter along the direction of $\boldsymbol{v}_2$

You can escape the saddle point and decrease the loss.

(this method is seldom used in practice)

刚才我们已经发现,原点是一个critical point,它的Hessian长这个样,那我现在发现说,这个Hessian有一个负的eigen value,这个eigen value等於-2,那它对应的eigen vector,它有很多个,其实是无穷多个对应的eigen vector,我们就取一个出来,我们取$\begin{bmatrix}{1} \\ {1}\end{bmatrix}$是它对应的一个eigen vector,那我们其实只要顺著这个u的方向,顺著$\begin{bmatrix}{1} \\ {1}\end{bmatrix}$这个vector的方向,去更新我们的参数,就可以找到一个,比saddle point的loss还要更低的点

如果以今天这个例子来看的话,你的saddle point在(0,0)这个地方,你在这个地方会没有gradient,Hessian的eigen vector告诉我们,只要往$\begin{bmatrix}{1} \\ {1}\end{bmatrix}$的方向更新,你就可以让loss变得更小,也就是说你可以逃离你的saddle point,然后让你的loss变小,所以从这个角度来看,似乎saddle point并没有那麼可怕

如果你今天在training的时候,你的gradient你的训练停下来,你的gradient变成零,你的训练停下来,是因為saddle point的话,那似乎还有解

但是当然实际上,在实际的implementation裡面,你几乎不会真的把Hessian算出来,这个要是二次微分,要计算这个矩阵的computation,需要的运算量非常非常的大,更遑论你还要把它的eigen value,跟 eigen vector找出来,所以在实作上,你几乎没有看到,有人用这一个方法来逃离saddle point

等一下我们会讲其他,也有机会逃离saddle point的方法,他们的运算量都比要算这个H,还要小很多,那今天之所以我们把,这个saddle point跟 eigen vector,跟Hessian的eigen vector拿出来讲,是想要告诉你说,如果是卡在saddle point,也许没有那麼可怕,最糟的状况下你还有这一招,可以告诉你要往哪一个方向走.

Saddle Point v.s. Local Minima

讲到这边你就会有一个问题了,这个问题是,那到底**saddle point跟local minima,谁比较常见呢**,我们说,saddle point其实并没有很可怕,那如果我们今天,常遇到的是saddle point,比较少遇到local minima,那就太好了,那到底saddle point跟local minima,哪一个比较常见呢?这边我们要讲一个不相干的故事,先讲一个故事

• A.D. 1543



这个故事发生在1543年,1543年发生了什麼事呢,那一年君士坦丁堡沦陷,这个是君士坦丁堡沦陷图,君士坦丁堡本来是东罗马帝国的领土,然后被鄂图曼土耳其帝国佔领了,然后东罗马帝国就灭亡了,在鄂图曼土耳其人进攻,君士坦丁堡的时候,那时候东罗马帝国的国王,是君士坦丁十一世,他不知道要怎麼对抗土耳其人,有人就献上了一策,找来了一个魔法师叫做狄奥伦娜

來源《三體III·死神永生》

Saddle Point v.s. Local Minima

• The Magician Diorena (魔法師狄奧倫娜)



Source of image: https://read01.com/mz2DBPE.html#.YECz22gzbIU

这是真实的故事,出自三体的故事,这个狄奥伦娜这样说,狄奥伦娜是谁呢,他有一个能力跟张飞一样,张飞不是可以万军从中取上将首级,如探囊取物吗,狄奥伦娜也是一样,他可以直接取得那个苏丹的头,他可以从万军中取得苏丹的头,大家想说狄奥伦娜怎麼这麼厉害,他真的有这麼强大的魔法吗,所以大家就要狄奥伦娜,先展示一下他的力量,这时候狄奥伦娜就拿出了一个圣杯,大家看到这个圣杯就大吃一惊,為什麼大家看到这个圣杯,要大吃一惊呢,因為这个圣杯,本来是放在圣索菲亚大教堂的地下室,而且它是被放在一个石棺裡面,这个石棺是密封的,没有人可以打开它.

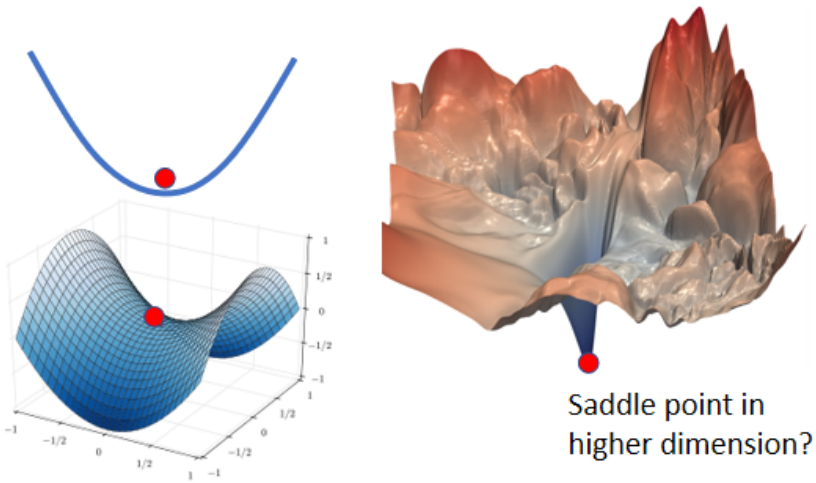
• The Magician Diorena（魔法師狄奧倫娜）



Source of image: https://read01.com/mz2DBPE.html#.YECz22gzbIU

但是狄奥伦娜他从裡面取得了圣杯,而且还放了一串葡萄进去,君士坦丁十一世為了要验证,狄奥伦娜是不是真的有这个能力,就带了一堆人真的去撬开了这个石棺,发现圣杯真的被拿走了,裡面真的有一串新鲜的葡萄,就知道狄奥伦娜真的有,这个万军从中取上将首级的能力,那為什麼迪奥伦娜可以做到这些事呢,那是因為这个石棺你觉得它是封闭的,那是因為你是从三维的空间来看,从三维的空间来看,这个石棺是封闭的,没有任何路可以进去,但是狄奥伦娜可以进入四维的空间,从高维的空间中,这个石棺是有路可以进去的,它并不是封闭的,至於狄奥伦娜有没有成功刺杀苏丹呢,你可以想像一定是没有嘛,所以君坦丁堡才沦陷,那至於為什麼没有,大家请见於三体这样 就不雷大家,

总之这个**从三维的空间来看,是没有路可以走的东西,在高维的空间中是有路可以走的,error surface会不会也一样呢**

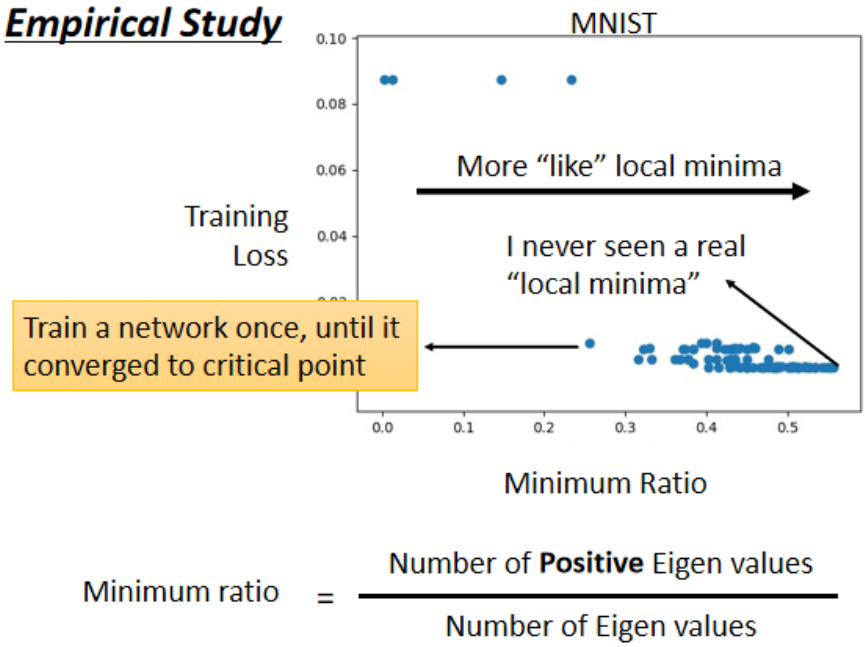


When you have lots of parameters, perhaps local minima is rare?

所以你在一维的空间中,一维的一个参数的error surface,你会觉得好像到处都是local minima,但是会不会在二维空间来看,它就只是一个saddle point呢,常常会有人画类似这样的图,告诉你说Deep Learning的训练,是非常的复杂的,如果我们移动某两个参数,error surface的变化非常的复杂,是这个样子的,那显然它有非常多的local minima,我的这边现在有一个local minima,但是会不会这个local minima,只是在二维的空间中,看起来是一个local minima,在更高维的空间中,它看起来就是saddle point,在二维的空间中,我们没有路可以走,那会不会在更高的维度上,因為更高的维度,我们

没办法visualize它,我们没办法真的拿出来看,会不会在更高维的空间中,其实有路可以走的,那如果维度越高,是不是可以走的路就越多了呢,所以 今天我们在训练,一个network的时候,我们的参数往往动輒百万千万以上,所以我们的error surface,其实是在一个非常高的维度中,对不对,我们参数有多少,就代表我们的error surface的,维度有多少,参数是一千万 就代表error surface,它的维度是一千万,竟然维度这麼高,会不会其实,根本就有非常多的路可以走呢,那既然有非常多的路可以走,会不会其实local minima,根本就很少呢,

而经验上,如果你自己做一些实验的话,也支持这个假说



这边是训练某一个network的结果,每一个点代表,训练那个network训练完之后,把它的Hessian拿出来进行计算,所以这边的每一个点,都代表一个network,就我们训练某一个network,然后把它训练训练,训练到gradient很小,卡在critical point,把那组参数出来分析,看看它比较像是saddle point,还是比较像是local minima

- 纵轴代表training的时候的loss,就是我们今天卡住了,那个loss没办法再下降了,那个loss是多少,那很多时候,你的loss在还很高的时候,训练就不动了 就卡在critical point,那很多时候loss可以降得很低,才卡在critical point,这是纵轴的部分
- 横轴的部分是minimum ratio,minimum ratio是**eigen value的数目分之正的eigen value的数目**,又**如果所有的eigen value都是正的,代表我们今天的critical point,是local minima,如果有正有负代表saddle point**,那在实作上你会发现说,你几乎找不到完全所有eigen value都是正的critical point,你看这边这个例子裡面,这个minimum ratio代表eigen value的数目分之正的eigen value的数目,最大也不过0.5到0.6间而已,代表说只有一半的eigen value是正的,还有一半的eigen value是负的,

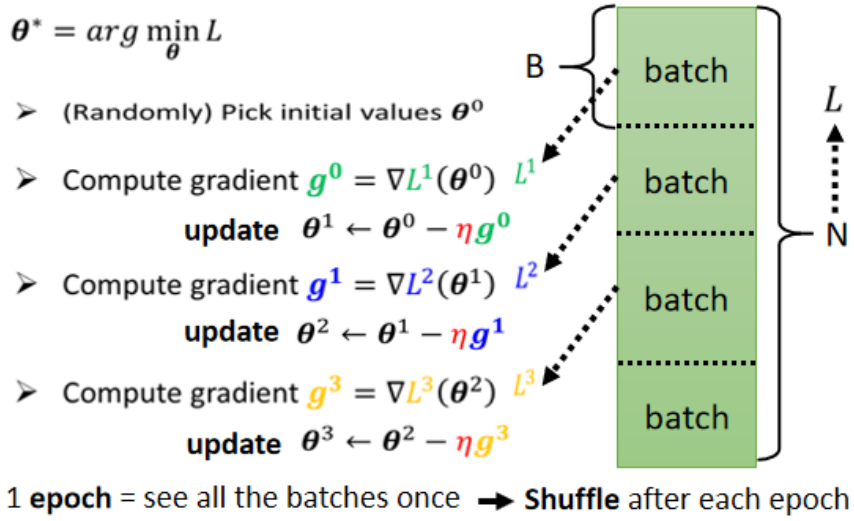
所以今天虽然在这个图上,越往右代表我们的critical point越像local minima,**但是它们都没有真的,变成local minima**,就算是在最极端的状况,我们仍然有一半的case,我们的eigen value是负的,这一半的case eigen value是正的,代表说在所有的维度裡面有一半的路,这一半的路 如果要让loss上升,还有一半的路可以让loss下降。

所以从经验上看起来,其实local minima并没有那麼常见,多数的时候,你觉得你train到一个地方,你gradient真的很小,然后所以你的参数不再update了,往往是因為你卡在了一个saddle point。

Batch and Momentum

Review： Optimization with Batch

上次我们有讲说,我们**实际上在算微分的时候,并不是真的对所有 Data 算出来的 L 作微分**,你是把所有的 Data 分成一个一个的 Batch,有的人是叫Mini Batch ,那我这边叫做 Batch,其实指的是一样的东西,助教投影片裡面,是写 Mini Batch



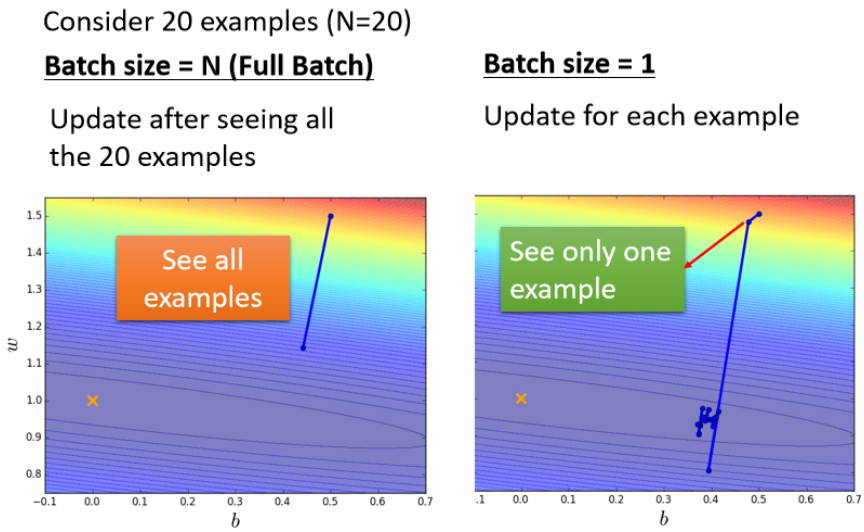
每一个 Batch 的大小呢,就是大 B 一笔的资料,我们每次**在 Update 参数的时候,我们是拿大 B 一笔资料出来,算个 Loss,算个 Gradient,Update 参数**,拿另外B一笔资料,再算个 Loss,再算个 Gradient,再 Update 参数,以此类推,所以我们不会拿所有的资料一起去算出 Loss,我们只会拿一个 Batch 的资料,拿出来算 Loss

所有的 Batch 看过一遍,叫做一个 Epoch,那事实上啊,你今天在做这些 Batch 的时候,你会做一件事情叫做 Shuffle

Shuffle 有很多不同的做法,但一个常见的做法就是,**在每一个 Epoch 开始之前,会分一次 Batch,然后呢,每一个 Epoch 的 Batch 都不一样**,就是第一个 Epoch,我们分这样子的 Batch,第二个 Epoch,会重新再分一次 Batch,所以哪些资料在同一个 Batch 裡面,每一个 Epoch 都不一样的这件事情,叫做 Shuffle。

Small Batch v.s. Large Batch

我们先解释為什麼要用 Batch,再说 Batch 对 Training 带来了什麼样的帮助。



我们来比较左右两边这两个 Case,那假设现在我们有20笔训练资料

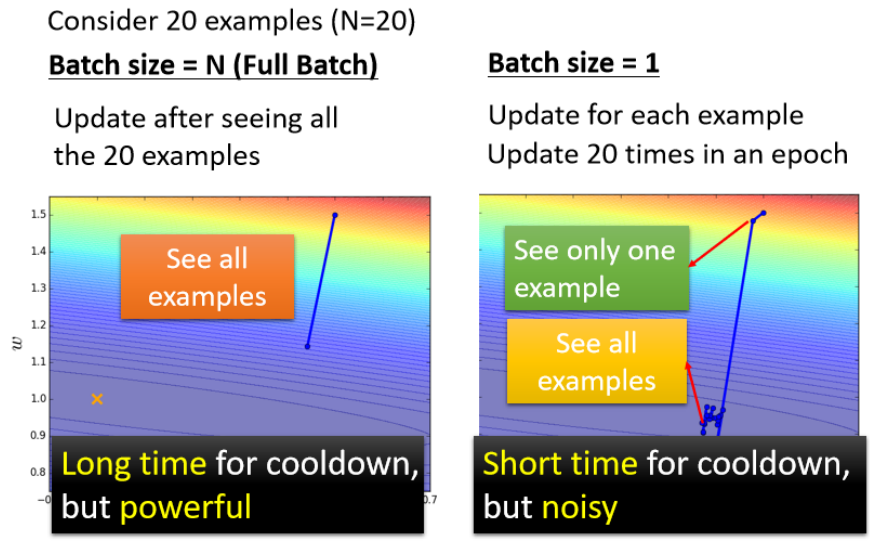
- 左边的 Case 就是没有用 Batch,Batch Size,直接设的跟我训练资料一样多,这种状况叫做 Full Batch,就是没有用 Batch 的意思
- 那右边的 Case 就是,Batch Size 等於1

这是两个最极端的状况

我们先来看左边的 Case,在左边 Case 裡面,因為没有用 Batch,我们的 Model 必须把20笔训练资料都看完,才能够计算 Loss,才能够计算 Gradient,所以我们必须要把**所有20笔 Example s 都看完以后,我们的参数才能够 Update 一次**。就假设开始的地方在上边边,把所有资料都看完以后,Update 参数就从这裡移动到下边。

如果 Batch Size 等於1的话,代表我们只需要拿一笔资料出来算 Loss,我们就可以 Update 我们的参数,所以每次我们 Update 参数的时候,看一笔资料就好,所以我们开始的点在这边,看一笔资料 就 Update 一次参数,再看一笔资料 就 Update 一次参数,如果今天总共有20笔资料的话 那**在每一个 Epoch 裡面,我们的参数会 Update 20 次**,那不过,因為我们现在是只看一笔资料,就 Update 一次参数,所以用一笔资料算出来的 Loss,显然是比较 Noisy 的,所以我们今天 Update 的方向,你会发现它是曲曲折折的

所以如果我们比较左边跟右边，哪一个比较好呢,他们有什麼差别呢?



你会发现左边没有用 Batch 的方式,它蓄力的时间比较长,还有它技能冷却的时间比较长,你要把所有的资料都看过一遍,才能够 Update 一次参数

而右边的这个方法,Batch Size 等於1的时候,蓄力的时间比较短,每次看到一笔参数,每次看到一笔资料,你就会更新一次你的参数

所以今天假设有20笔资料,看完所有资料看过一遍,你已经更新了20次的参数,但是左边这样子的方法有一个优点,就是它这一步走的是稳的,那右边这个方法它的缺点,就是它每一步走的是不稳的

看起来左边的方法跟右边的方法,他们各自都有擅长跟不擅长的东西,左边是蓄力时间长,但是威力比较大,右边技能冷却时间短,但是它是比较不準的,看起来各自有各自的优缺点,但是你会觉得说,左边的方法技能冷却时间长,右边的方法技能冷却时间短,那只是你没有考虑并行运算的问题。

实际上考虑并行运算的话,左边这个并不一定时间比较长

Larger batch size does not require longer time to compute gradient

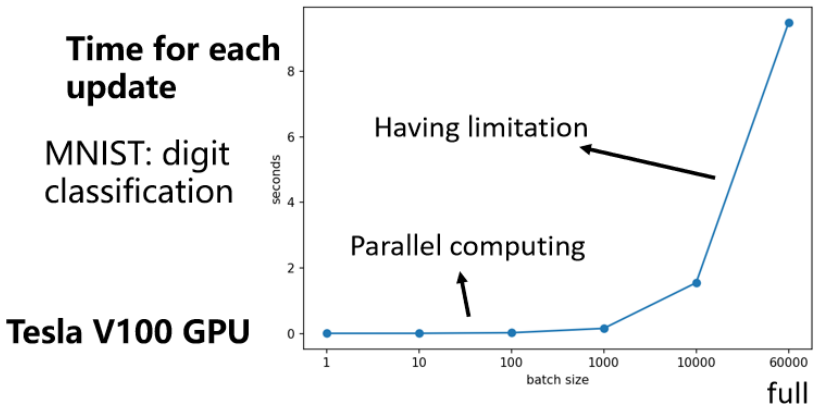
这边是真正的实验结果了,事实上,比较大的 Batch Size,你要算 Loss,再进而算 Gradient,所需要的时间,不一定比小的 Batch Size 要花的时间长

那以下是做在一个叫做 ==MNIST== 上面,MNIST (Mixed National Institute of Standards and Technology database)是美国国家标准与技术研究院收集整理的大型手写数字数据库,机器要做的事情,就是给它一张图片,然后判断这张图片,是0到9的哪一个数字,它要做数字的分类,那 MNIST 呢 是机器学习的helloworld,就是假设你今天,从来没有做过机器学习的任务,一般大家第一个会尝试的机器学习的任务,往往就是做 MNIST 做手写数字辨识,

oldest slides: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20(v4).pdf
old slides: http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/Keras.pdf

Small Batch v.s. Large Batch

- Larger batch size does not require longer time to compute gradient (unless batch size is too large)



这边我们就是做了一个实验,我们想要知道说,给机器一个 Batch,它要计算出 Gradient,进而 Update 参数,到底需要花多少的时间

这边列出了 Batch Size 等於1 等於10,等於100 等於1000 所需要耗费的时间

你会发现说 Batch Size 从1到1000,需要耗费的时间几乎是一样的,你可能直觉上认为有1000笔资料,那需要计算 Loss,然后计算 Gradient,花的时间不会是一笔资料的1000倍吗,但是实际上并不是这样的

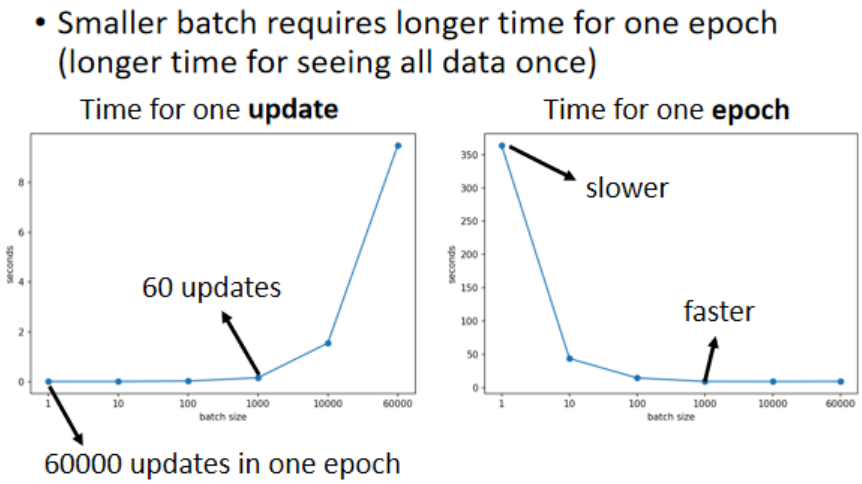
因為在实际上做运算的时候,我们有 GPU,可以做并行运算,是因為你可以做平行运算的关係,这1000笔资料是平行处理的,所以1000笔资料所花的时间,并不是一笔资料的1000倍,当然 GPU 平行运算的能力还是有它的极限,当你的 Batch Size 真的非常非常巨大的时候,GPU 在跑完一个 Batch,计算出 Gradient 所花费的时间,还是会随著 Batch Size 的增加,而逐渐增长

所以今天如果 Batch Size 是从1到1000,所需要的时间几乎是一样的,但是当你的 Batch Size 增加到 10000,乃至增加到60000的时候,你就会发现 GPU 要算完一个 Batch,把这个 Batch 裡面的资料都拿出来算 Loss,再进而算 Gradient,所要耗费的时间,确实有随著 Batch Size 的增加而逐渐增长,但你会发现这边用的是 V100,所以它挺厉害的,给它60000笔资料,一个 Batch 裡面,塞了60000笔资料,它在10秒鐘之内,也是把 Gradient 就算出来

而那这个 Batch Size 的大小跟时间的关係,其实每年都会做这个实验,我特别把旧的投影片放在这边了,如果你有兴趣的话m,,可以看到这个时代的演进这样,17年的时候用的是那个980啊,2015年的时候用的是那个760啊,然后980要跑什麼60000个 Batch,那要跑好几分鐘才跑得完啊,现在只要10秒鐘就可以跑得完了,你可以看到这个时代的演进,

Smaller batch requires longer time for one epoch

所以 GPU 虽然有平行运算的能力,但它平行运算能力终究是有个极限,所以你 Batch Size 真的很大的时候,时间还是会增加的



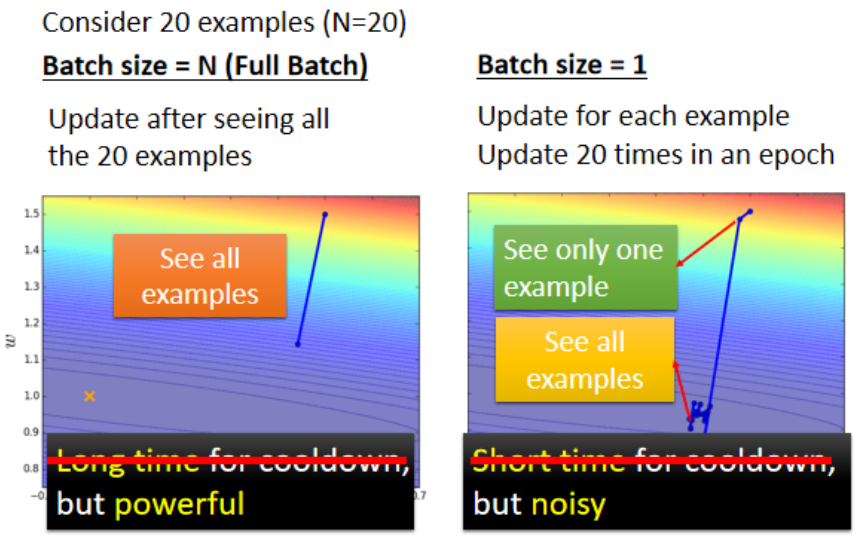
但是因為有平行运算的能力,因此实际上,当你的 **Batch Size 小的时候,你要跑完一个 Epoch,花的时间是比大的 Batch Size 还要多**的,怎麼说呢

如果今天假设我们的训练资料只有60000笔,那 Batch Size 设1,那你要60000个 Update 才能跑完一个 Epoch,如果今天是 Batch Size 等於1000,你要60个 Update 才能跑完一个 Epoch,假设今天一个 Batch Size 等於1000,要算 Gradient 的时间根本差不多,那60000次 Update,跟60次 Update 比起来,它的时间的差距量就非常可观了

所以左边这个图是 Update 一次参数,拿一个 Batch 出来计算一个 Gradient,Update 一次参数所需要的时间,右边这个图是,跑完一个完整的 Epoch,需要花的时间,你会发现左边的图跟右边的图,它的趋势正好是相反的,假设你 Batch Size 这个1,跑完一个 Epoch,你要 Update 60000次参数,它的时间是非常可观的,但是假设你的 Batch Size 是1000,你只要跑60次,Update 60次参数就会跑完一个 Epoch,所以你跑完一个 Epoch,看完所有资料的时间,如果你的 Batch Size 设1000,其实是比较短的,Batch Size 设1000的时候,把所有的资料看过一遍,其实是比 Batch Size 设1 还要更快

所以如果我们看右边这个图的话,看完一个 Batch,把所有的资料看过一次这件事情,大的 Batch Size 反而是较有效率的,是不是跟你直觉想的不太一样

在没有考虑平行运算的时候,你觉得大的 Batch 比较慢,但实际上,在有考虑平行运算的时候,一个 Epoch 大的 Batch 花的时间反而是比较少的



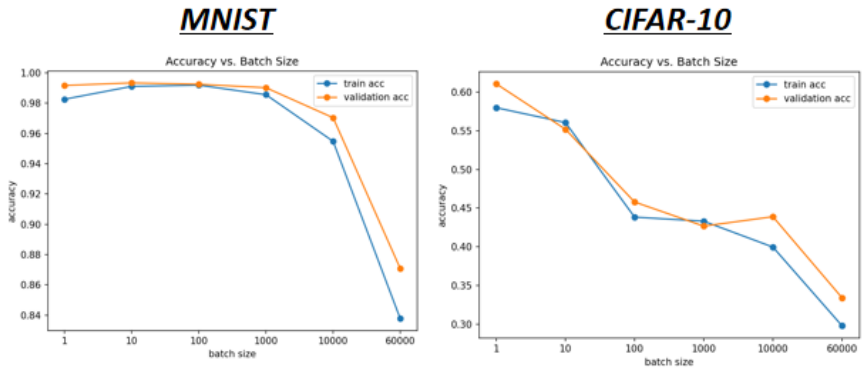
我们如果要比较这个 Batch Size 大小的差异的话,看起来直接用技能时间冷却的长短,并不是一个精确的描述,看起来在技能时间上面,大的 Batch 并没有比较吃亏,甚至还佔到优势了.

所以事实上,20笔资料 Update 一次的时间,跟右边看一笔资料 Update 一次的时间,如果你用 GPU 的话,其实可能根本就是所以一样的,所以大的 Batch,它的技能时间,它技能冷却的时间,并没有比较长,那所以这时候你可能就会说,欸 那个大的 Batch 的劣势消失了,那难道它真的就,那这样看起来大的 Batch 应该比较好?

你不是说大的 Batch,这个 Update 比较稳定,小的 Batch,它的 Gradient 的方向比较 Noisy 吗,那这样看起来,大的 Batch 好像应该比较好哦,小的 Batch 应该比较差,因為现在大的 Batch 的劣势已经,因為平行运算的时间被拿掉了,它好像只剩下优势而已.

那神奇的地方是 **Noisy 的 Gradient,反而可以帮助 Training**,这个也是跟直觉正好相反的

如果你今天拿不同的 Batch 来训练你的模型,你可能会得到这样子的结果,左边是坐在 MNIST 上,右边是坐在 CIFAR-10 上,不管是 MNIST 还是 CIFAR-10,都是影像辨识的问题



- Smaller batch size has better performance
- What's wrong with large batch size? Optimization Issue

- 横轴代表的是 Batch Size,从左到右越来越大
- 纵轴代表的是正确率,越上面正确率越高,当然正确率越高越好

而如果你今天看 Validation Acc 上的结果，会发现说,Batch Size 越大,Validation Acc 上的结果越差,但这个不是 Overfitting,因為如果你看你的 Training 的话,会发现说 Batch Size 越大,Training 的结果也是越差的,而我们现在用的是同一个模型哦,照理说,它们可以表示的 Function 就是一模一样的

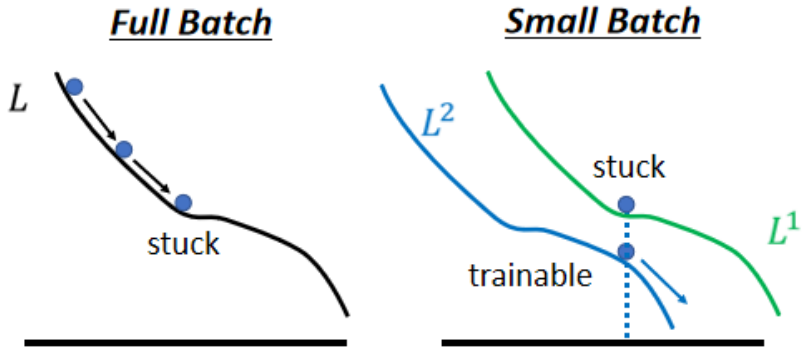
但是神奇的事情是,大的 Batch Size,往往在 Training 的时候,会给你带来比较差的结果

所以这个是什麼样的问题,同样的 Model,所以这个不是 Model Bias 的问题,**这个是 Optimization 的问题,代表当你用大的 Batch Size 的时候,你的 Optimization 可能会有问题**,小的 Batch Size,Optimization 的结果反而是比较好的,好 為什麼会这样子呢

"Noisy" update is better for training

為什麼小的 Batch Size,在 Training Set 上会得到比较好的结果,為什麼 Noisy 的 Update,Noisy 的 Gradient 会在 Training 的时候,给我们比较好的结果呢？一个可能的解释是这样子的

- Smaller batch size has better performance
- "Noisy" update is better for training



假设你是 Full Batch,那你今天在 Update 你的参数的时候,你就是沿著一个 Loss Function 来 Update 参数,今天 Update 参数的时候走到一个 Local Minima,走到一个 Saddle Point,显然就停下来了,Gradient 是零,如果你不特别去看Hession的话,那你用 Gradient Descent 的方法,你就没有办法再更新你的参数了

但是假如是 Small Batch 的话,因為我们每次是挑一个 Batch 出来,算它的 Loss,所以等於是,等於你每一次 Update 你的参数的时候,你用的 Loss Function 都是越有差异的,你选到第一个 Batch 的时候,你是用 L1 来算你的 Gradient,你选到第二个 Batch 的时候,你是用 L2 来算你的 Gradient,假设你用 L1 算 Gradient 的时候,发现 Gradient 是零,卡住了,但 L2 它的 Function 跟 L1 又不一样,L2 就不一定会卡住,所以 L1 卡住了 没关係,换下一个 Batch 来,L2 再算 Gradient。

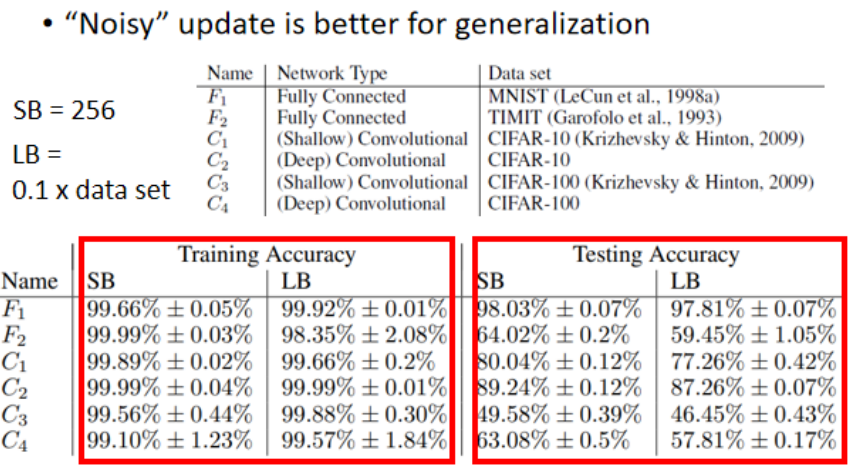
你还是有办法 Training 你的 Model,还是有办法让你的 Loss 变小,所以今天这种 Noisy 的 Update 的方式,结果反而对 Training,其实是有帮助的。

“Noisy” update is better for generalization

那这边还有另外一个更神奇的事情，其实**小的 Batch 也对 Testing 有帮助**。

假设我们今天在 Training 的时候,都不管是大的 Batch 还小的 Batch,都 Training 到一样好,刚才的 Case是Training 的时候就已经 Training 不好了

假设你有一些方法,你努力的调大的 Batch 的 Learning Rate,然后想办法把大的 Batch,跟小的 Batch Training 得一样好,结果你会发现**小的 Batch,居然在 Testing 的时候会是比较好的**,那以下这个实验结果是引用自,On Large-Batch Training For Deep Learning,Generalization Gap And Sharp Minimahttps://arxiv.org/abs/1609.04836,这篇 Paper 的实验结果



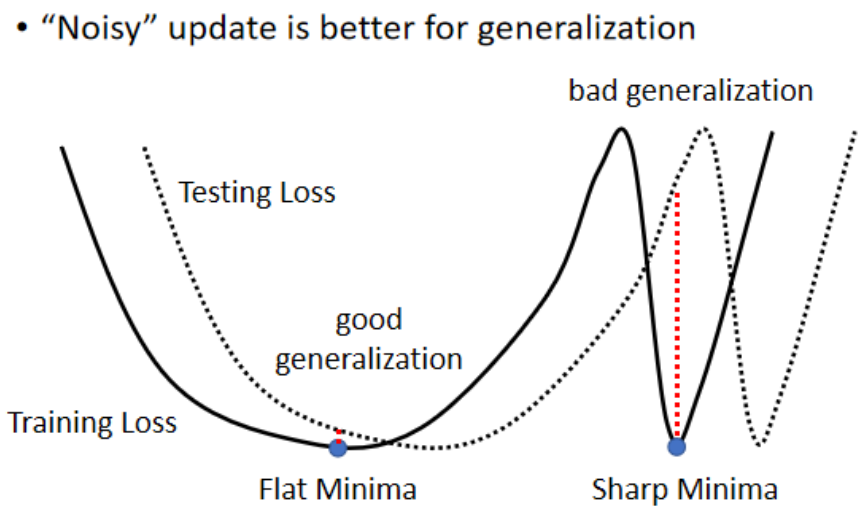
Name	Network Type	Data set
F_1	Fully Connected	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	CIFAR-10
C_3	(Shallow) Convolutional	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	CIFAR-100

	Training Accuracy		Testing Accuracy	
Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99\% \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

那这篇 Paper 裡面,作者 Train 了六个 Network 裡面有 CNN 的,有 Fully Connected Network 的,做在不同的 Cover 上,来代表这个实验是很泛用的,在很多不同的 Case 都观察到一样的结果,那它有小的 Batch,一个 Batch 裡面有256笔 Example,大的 Batch 就是那个 Data Set 乘 0.1,Data Set 乘 0.1,Data Set 有60000笔,那你就是一个 Batch 裡面有6000笔资料

然后他想办法,在大的 Batch 跟小的 Batch,都 Train 到差不多的 Training 的 Accuracy,所以刚才我们看到的结果是,Batch Size 大的时候,Training Accuracy 就已经差掉了,这边不是想办法 Train 到大的 Batch 的时候,Training Accuracy 跟小的 Batch,其实是差不多的

但是就算是在 Training 的时候结果差不多,Testing 的时候你还是看到了,小的 Batch 居然比大的 Batch 差,Training 的时候都很好,**Testing 的时候小的 Batch 差,代表 Over Fitting**,这个才是 Over Fitting 对不对,好 那為什麼会有这样子的现象呢? 在这篇文章裡面也给出了一个解释,



假设这个是我们的 Training Loss,那在这个 Training Loss 上面呢,可能有很多个 Local Minima,有不只一个 Local Minima,那这些 Local Minima 它们的 Loss 都很低,它们 Loss 可能都趋近於 0,但是这个 **Local Minima,还是有好 Minima 跟坏 Minima 之分**

如果一个 Local Minima 它在一个峡谷裡面,它是坏的 Minima,然后它在一个平原上,它是好的 Minima,為什麼会有这样的差异呢

- 因為假设现**在 Training 跟 Testing 中间,有一个 Mismatch**,Training 的 Loss 跟 Testing 的 Loss,它们那个 Function 不一样,有可能是本来你 Training 跟 Testing 的 Distribution就不一样。
- 那也有可能是因為 Training 跟 Testing,你都是从 Sample 的 Data 算出来的,也许 Training 跟 Testing,Sample 到的 Data 不一样,那所以它们算出来的 Loss,当然是有一点差距。

那我们就假设说这个 Training 跟 Testing,它的差距就是把 Training 的 Loss,这个 Function 往右平移一点,这时候你会发现,对左边这个在一个盆地裡面的 Minima 来说,它的在 Training 跟 Testing 上面的结果,不会差太多,只差了一点点,但是对右边这个在峡谷裡面的 Minima 来说,一差就可以天差地远

它在这个 Training Set 上,算出来的 Loss 很低,但是因為 Training 跟 Testing 之间的不一样,所以 Testing 的时候,这个 Error Surface 一变,它算出来的 Loss 就变得很大,而很多人相信这个**大的 Batch Size,会让我们倾向於走到峡谷裡面,而小的 Batch Size,倾向於让我们走到盆地裡面**

那他直觉上的想法是这样,就是小的 Batch,它有很多的 Loss,它每次 Update 的方向都不太一样,所以如果今天这个峡谷非常地窄,它可能一个不小心就跳出去了,因為每次 Update 的方向都不太一样,它的 Update 的方向也就随机性,所以一个很小的峡谷,没有办法困住小的 Batch

如果峡谷很小,它可能动一下就跳出去,之后停下来如果有一个非常宽的盆地,它才会停下来,那对於大的 Batch Size,反正它就是顺著规定 Update,然后它就很有可能,走到一个比较小的峡谷裡面

但这只是一个解释,那也不是每个人都相信这个解释,那这个其实还是一个**尚待研究的问题**

那这边就是比较了一下,大的 Batch 跟小的 Batch

	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster WIN
Gradient	Noisy	Stable
Optimization	Better WIN	Worse
Generalization	Better WIN	Worse

Batch size is a hyperparameter you have to decide.

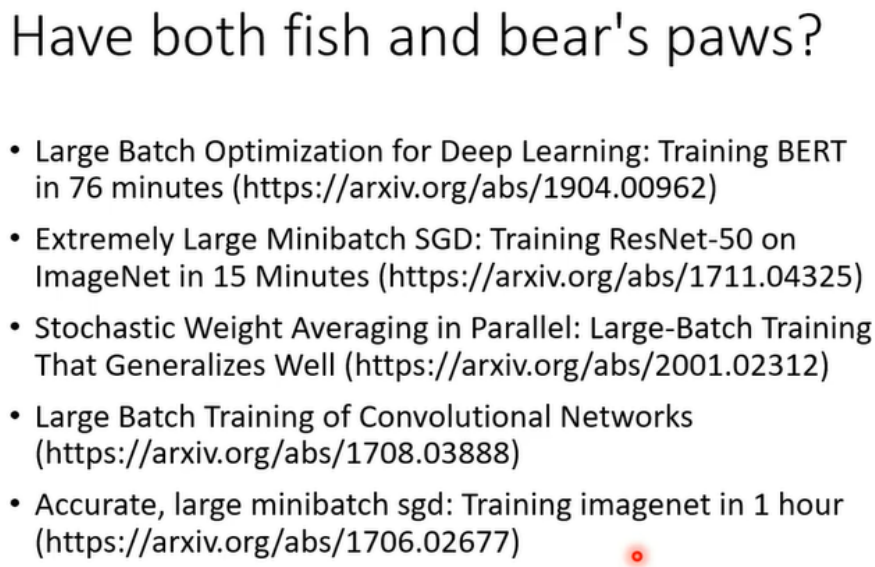
左边这个是第一个 Column 是小的 Batch,第二个 Column 是大的 Batch

在有平行运算的情况下,小的 Batch 跟大的 Batch,其实运算的时间并没有太大的差距,除非你的大的 Batch 那个大是真的非常大,才会显示出差距来。但是一个 Epoch 需要的时间,小的 Batch 比较长,大的 Batch 反而是比较快的,所以从一个 Epoch 需要的时间来看,大的 Batch 其实是佔到优势的。

而小的 Batch,你会 Update 的方向比较 Noisy,大的 Batch Update 的方向比较稳定,但是 Noisy 的 Update 的方向,反而在 Optimization 的时候会佔到优势,而且在 Testing 的时候也会佔到优势,所以大的 Batch 跟小的 Batch,它们各自有它们擅长的地方

所以 Batch Size,变成另外一个 你需要去调整的 Hyperparameter。

那我们能不能够鱼与熊掌兼得呢,我们能不能够截取大的 Batch 的优点,跟小的 Batch 的优点,我们用大的 Batch Size 来做训练,用平行运算的能力来增加训练的效率,但是训练出来的结果同时又得到好的结果呢,又得到好的训练结果呢。

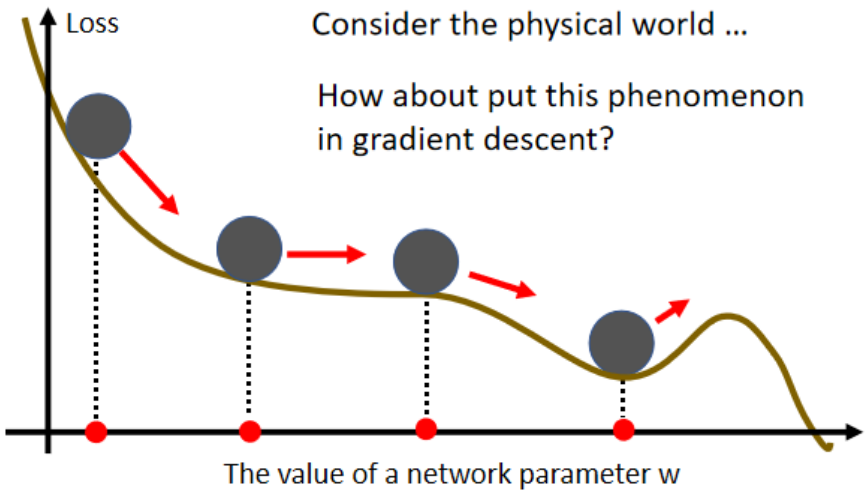


這是有可能的,有很多文章都在探讨这个问题,那今天我们就不细讲,我们把这些 Reference 列在这边给大家参考,那你发现这些 Paper,往往它想要做的事情都是什麼,哇 76分鐘 Train BERT,15分鐘 Train ResNet,一分鐘 Train Imagenet 等等,这為什麼他们可以做到那麼快,就是因為他们 Batch Size 是真的开很大,比如说在第一篇 Paper 裡面,Batch Size 裡面有三万笔 Example 这样,Batch Size 开很大,Batch Size 开大 真的就可以算很快,你可以在很短的时间内看到大量的资料,那他们需要有一些特别的方法来解决,Batch Size 可能会带来的劣势。

Momentum

Momentum,这也是另外一个,有可能可以对抗 Saddle Point,或 Local Minima 的技术,Momentum 的运作是这个样子的

Small Gradient



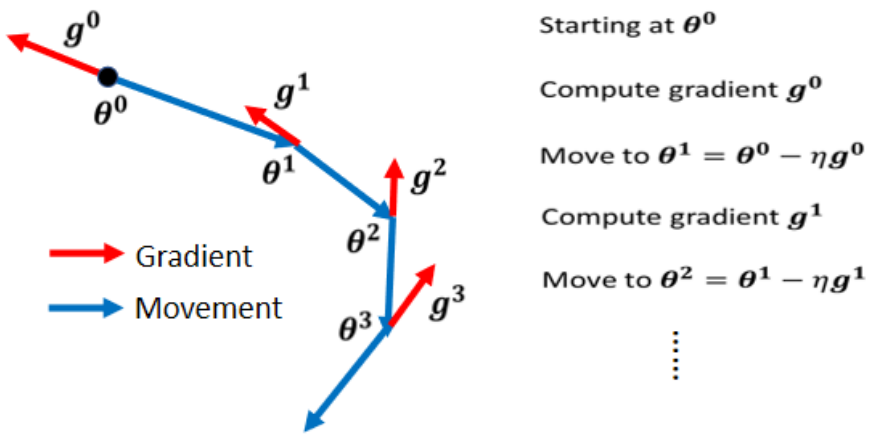
它的概念,你可以想像成在物理的世界裡面,假设 Error Surface 就是真正的斜坡,而我们的参数是一个球,你把球从斜坡上滚下来,如果今天是 Gradient Descent,它走到 Local Minima 就停住了,走到 Saddle Point 就停住了

但是在物理的世界裡,一个球如果从高处滚下来,从高处滚下来就算滚到 Saddle Point,如果有**惯性**,它从左边滚下来,因為惯性的关係它还是会继续往右走,甚至它走到一个 Local Minima,如果今天它的动量够大的话,它还是会继续往右走,甚至翻过这个小坡然后继续往右走

那所以今天在物理的世界裡面,一个球从高处滚下来的时候,它并不会被 Saddle Point,或 Local Minima卡住,不一定会被 Saddle Point,或 Local Minima 卡住,我们有没有办法运用这样子的概念,到 Gradient Descent 裡面呢,那这个就是我们等一下要讲的,Momentum 这个技术

Vanilla Gradient Descent

那我们先很快的复习一下,原来的 Gradient Descent 长得是什麼样子,这个是 Vanilla 的 Gradient Descent,Vanilla 的意思就是一般的的意思,它直译是香草的,但就其实是一般的,一般的 Gradient Descent 长什麼样子呢



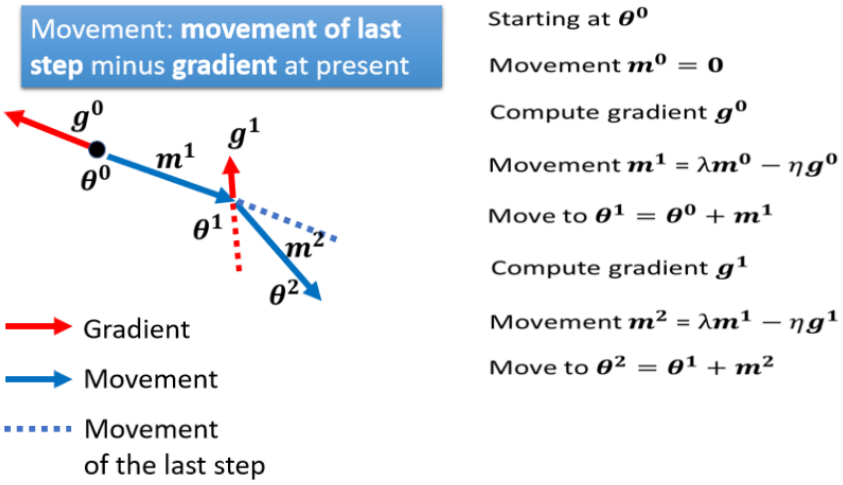
一般的 Gradient Descent 是说,我们有一个初始的参数叫做 $θ^0$,我们计算一下 Gradient,然后计算完这个 Gradient 以后呢,我们往 Gradient 的反方向去 Update 参数

$$ θ^1 = θ^0 - {\eta}g^0

$$ 我们到了新的参数以后,再计算一次 Gradient,再往 Gradient 的反方向,再 Update 一次参数,到了新的位置以后再计算一次 Gradient,再往 Gradient 的反方向去 Update 参数,这个 Process 就一直这样子下去

Gradient Descent + Momentum

加上 Momentum 以后,每一次我们在移动我们的参数的时候,我们不是只往 Gradient Descent,我们不是只往 Gradient 的反方向来移动参数,我们是 **Gradient 的反方向,加上前一步移动的方向,两者加起来的结果,去调整去到我们的参数,**



那具体说起来是这个样子,一样找一个初始的参数,然后我们假设前一步的参数的 Update 量呢,就设为 0

$$ m^0 = 0

$$ 接下来在 $θ^0$ 的地方,计算 Gradient 的方向g^0

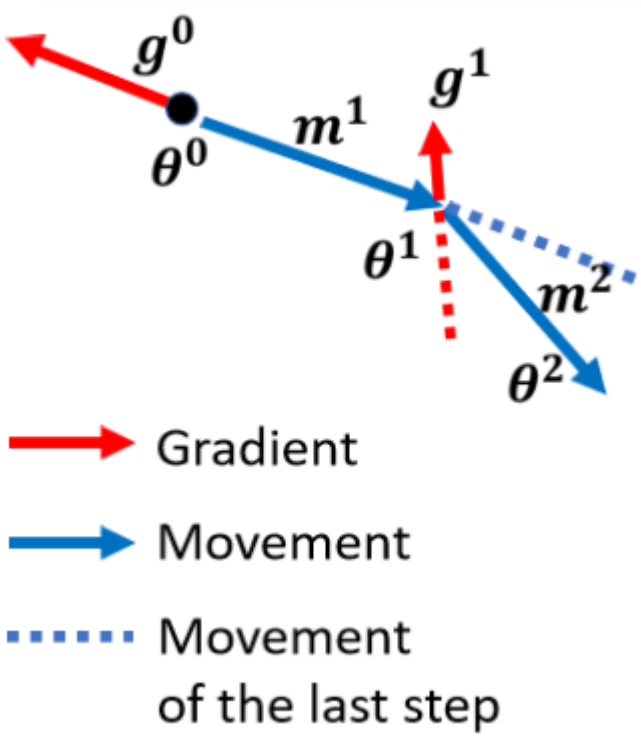
然后接下来你要决定下一步要怎麼走,它是 Gradient 的方向加上前一步的方向,不过因為前一步正好是 0,现在是刚初始的时候所以前一步是 0,所以 Update 的方向,跟原来的 Gradient Descent 是一样的,这没有什麼有趣的地方

$$ m^1 = {\lambda}m^0-{\eta}g^0\\ θ^1 = θ^0 + m^1

$$ 但从第二步开始,有加上 Momentum 以后就不太一样了,从第二步开始,我们计算 g^1,然后接下来我们 Update 的方向,不是 g^1的反方向,而是根据上一次 Update 方向,也就是 m1 减掉 g1,当做我们新的 Update 的方向,这边写成 m2

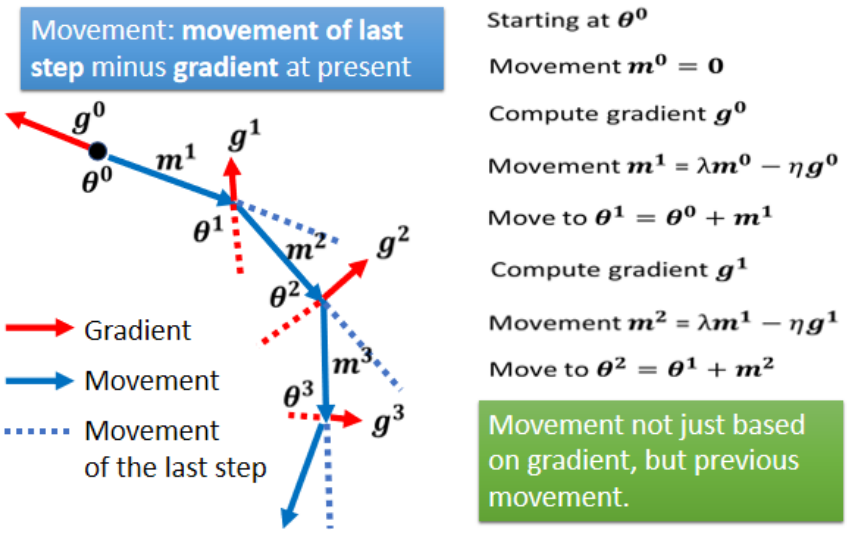
$$ m^2 = {\lambda}m^1-{\eta}g^1

$$ 那我们就看下面这个图

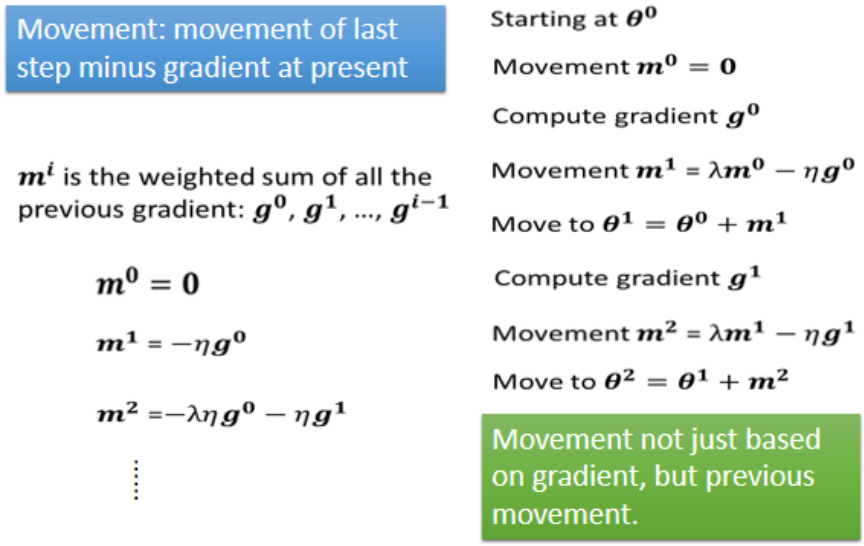


g1 告诉我们,**Gradient 告诉我们要往红色反方向这边走**,但是我们不是只听 Gradient 的话,加上 Momentum 以后,我们不是只根据 Gradient 的反方向,来调整我们的参数,我们**也会看前一次 Update 的方向**

- 如果前一次说要往**m^1蓝色及蓝色虚线**这个方向走
- Gradient 说要往**红色反方向这个方向**走
- **把两者相加起来**,走两者的折中,也就是往**蓝色m^2这一个方向走**,所以我们就移动了 m2,走到 θ2 这个地方



接下来就反覆进行同样的过程,在这个位置我们计算出 Gradient,但我们不是只根据 Gradient 反方向走,我们看前一步怎麼走,前一步走这个方向,走这个蓝色虚线的方向,我们把蓝色的虚线加红色的虚线,前一步指示的方向跟 Gradient 指示的方向,当做我们下一步要移动的方向



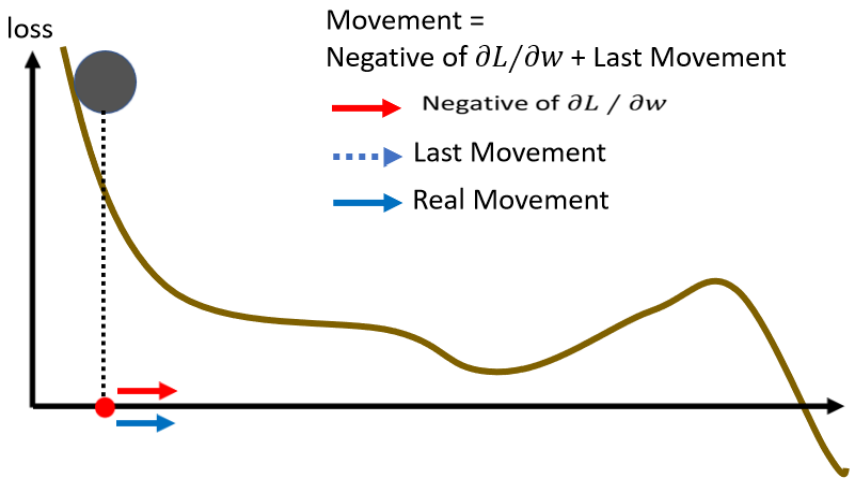
每一步的移动,我们都用 m 来表示,那这个 m 其实可以写成之前所有算出来的,Gradient 的 Weighted Sum.从右边的这个式子,其实就可以轻易的看出来

$$ m^0 = 0\\ m^1 = -{\eta}g^0\\ m^2 = -{\lambda}{\eta}g^0-{\eta}g^1\\ ...

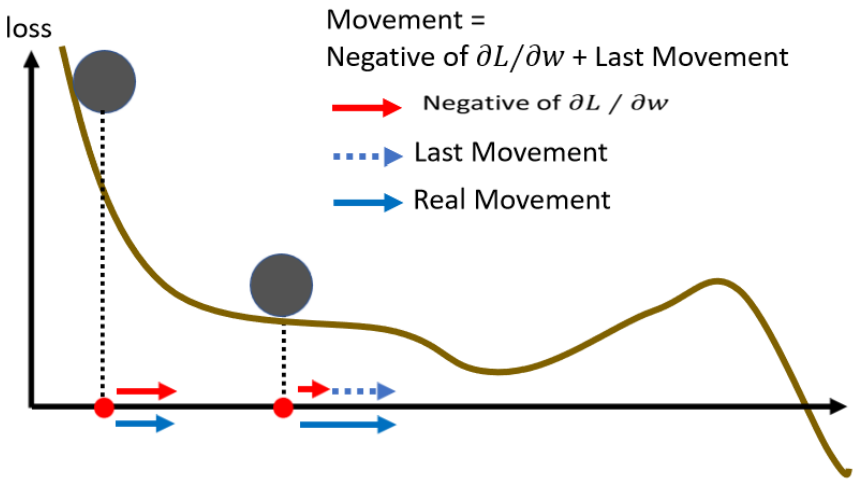
$$ m0 我们把它设为 0,m1 是 m0 减掉 g0,m0 为 0,所以 m1 就是 g0 乘上负的 η,m2 是 λ 乘上 m1,λ 就是另外一个参数,就好像 η 是 Learning Rate 我们要调,λ 是另外一个参数,这个也是需要调的,m2 等於 λ 乘上 m1,减掉 η 乘上 g1,然后 m1 在哪裡呢,m1 在这边,你把 m1 代进来,就知道说 m2,等於负的 λ 乘上 η 乘以 g0,减掉 η 乘上 g1,它是 g0 跟 g1 的 Weighted Sum

以此类推,所以你会发现说,现在这个加上 Momentum 以后,一个解读是 Momentum 是,Gradient 的负反方向加上前一次移动的方向,那但另外一个解读方式是,所谓的 Momentum,当加上 Momentum 的时候,我们 Update 的方向,不是只考虑现在的 Gradient,而是考虑过去所有 Gradient 的总合.

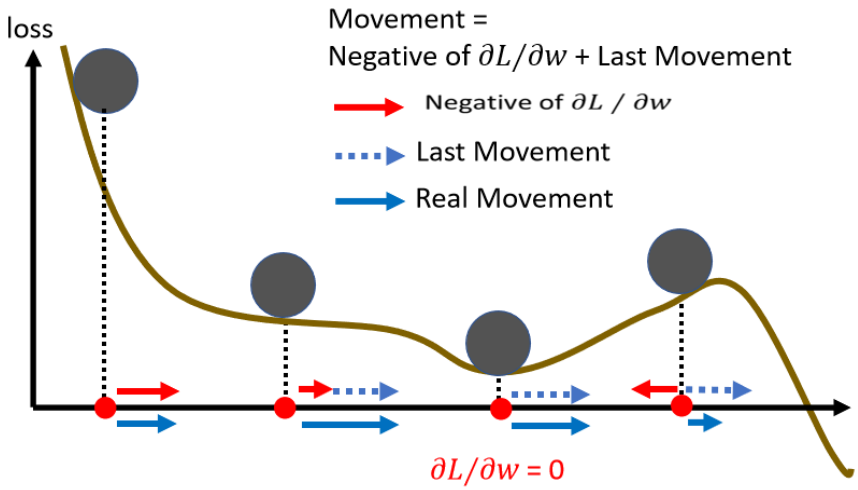
有一个更简单的例子,希望帮助你了解 Momentum



那我们从这个地方开始 Update 参数,根据 Gradient 的方向告诉我们,应该往右 Update 参数,那现在没有前一次 Update 的方向,所以我们就完全按照 Gradient 给我们的指示,往右移动参数,好 那我们的参数,就往右移动了一点到这个地方



Gradient 变得很小,告诉我们往右移动,但是只有往右移动一点点,但前一步是往右移动的,我们把前一步的方向用虚线来表示,放在这个地方,我们把之前 Gradient 告诉我们要走的方向,跟前一步移动的方向加起来,得到往右走的方向,那再往右走 走到一个 Local Minima,照理说走到 Local Minima,一般 Gradient Descent 就无法向前走了,因为已经没有这个 Gradient 的方向,那走到 Saddle Point 也一样,没有 Gradient 的方向已经无法向前走了



但没有关係,如果有 Momentum 的话,你还是有办法继续走下去,因為 Momentum 不是只看 Gradient,Gradient 就算是 0,你还有前一步的方向,前一步的方向告诉我们向右走,我们就继续向右走,甚至你走到这种地方,Gradient 告诉你应该要往左走了,但是假设你前一步的影响力,比 Gradient 要大的话,你还是有可能继续往右走,甚至翻过一个小丘,搞不好就可以走到更好 Local Minima,这个就是 Momentum 有可能带来的好处

那这个就是今天想要跟大家说的内容,

Concluding Remarks

- Critical points have zero gradients.
- Critical points can be either saddle points or local minima.
  - Can be determined by the Hessian matrix.

    - Local minima may be rare.
    - It is possible to escape saddle points along the direction of eigenvectors of the Hessian matrix
- Smaller batch size and momentum help escape critical points.

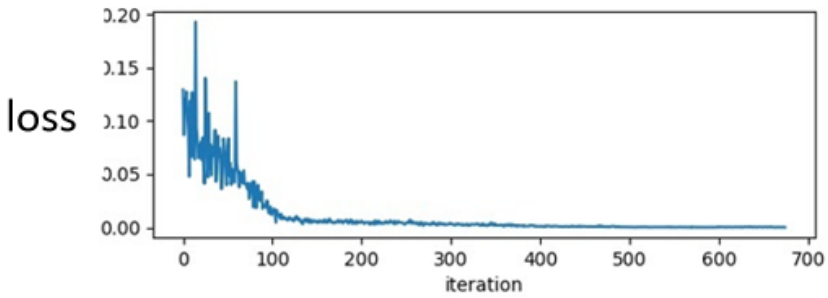
Tips for training: Adaptive Learning Rate

critical point其实不一定是,你在训练一个Network的时候,会遇到的最大的障碍,今天要告诉大家的是一个叫做Adaptive Learning Rate的技术,我们要给每一个参数不同的learning rate

Training stuck ≠ Small Gradient

People believe training stuck because the parameters are around a critical point …

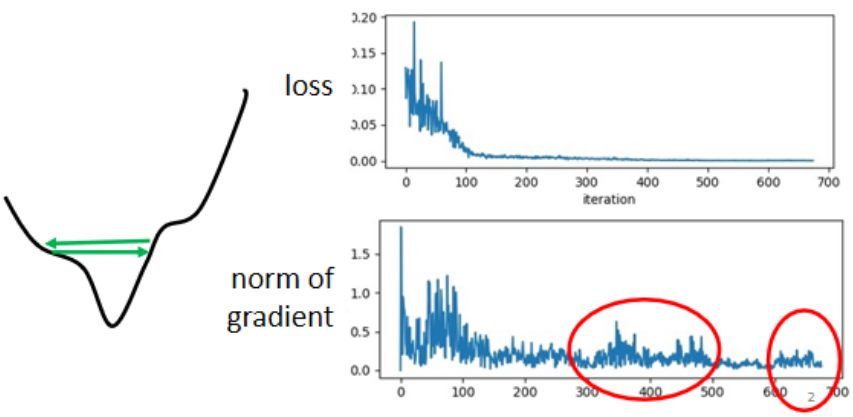
為什麼我说这个critical point不一定是我们训练过程中,最大的阻碍呢?



往往同学们,在训练一个network的时候,你会把它的loss记录下来,所以你会看到,你的loss原来很大,随著你参数不断的update,横轴代表参数update的次数,随著你参数不断的update,这个loss会越来越小,最后就卡住了,你的loss不再下降

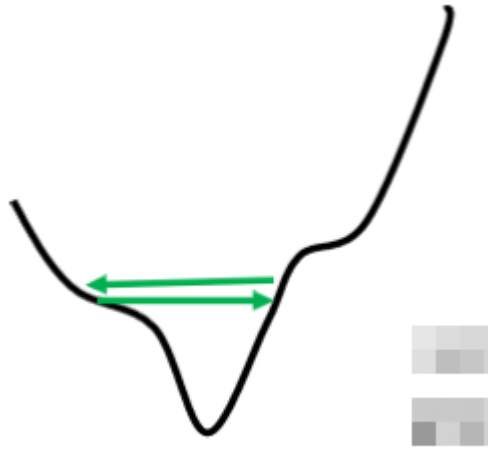
那多数这个时候,大家就会猜说,那是不是走到了critical point,因為gradient等於零的关係,所以我们没有办法再更新参数,但是真的是这样吗

当我们说 走到critical point的时候,意味著gradient非常的小,但是你有确认过,当**你的loss不再下降的时候,gradient真的很小吗?** 其实多数的同学可能,都没有确认过这件事,而事实上在这个例子裡面,在今天我show的这个例子裡面,当我们的loss不再下降的时候,gradient并没有真的变得很小



gradient是一个向量，下面是gradient的norm,即gradient这个向量的长度,随著参数更新的时候的变化,你会发现说**虽然loss不再下降,但是这个gradient的norm,gradient的大小并没有真的变得很小**

这样子的结果其实也不难猜想,也许你遇到的是这样子的状况



这个是我们的error surface,然后你现在的gradient,在error surface山谷的两个谷壁间,**不断的来回的震荡**

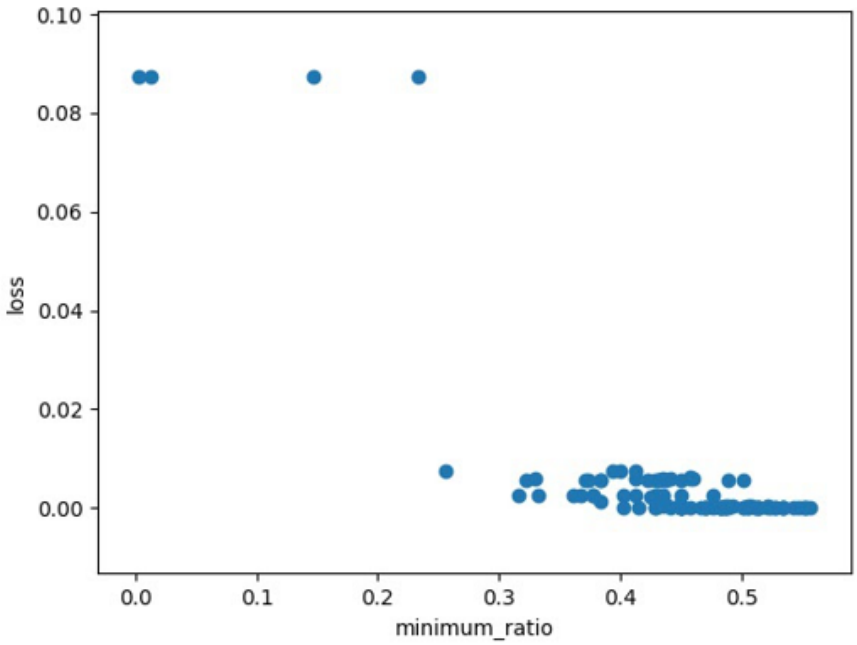
这个时候你的loss不会再下降,所以你会觉得它真的卡到了critical point,卡到了saddle point,卡到了local minima吗？不是的,**它的gradient仍然很大,只是loss不见得再减小了**

所以你要注意,当你今天训练一个network,train到后来发现,loss不再下降的时候,你不要随便说,我卡在local minima,我卡在saddle point,**有时候根本两个都不是,你只是单纯的loss没有办法再下降**

就是為什麼你在在作业2-2,会有一个作业叫大家,算一下gradient的norm,然后算一下说,你现在是卡在saddle point,还是critical point,因為多数的时候,当你说你训练卡住了,很少有人会去分析卡住的原因,為了强化你的印象,我们有一个作业,让你来分析一下,卡住的原因是什麼,

Wait a minute

有的同学就会有一个问题,如果我们在训练的时候,其实很少卡到saddle point,或者是local minima,那这一个图是怎麼做出来的呢?



我们上次有画过这个图是说我们现在训练一个Network,训练到现在参数**在critical point附近,然后我们再来根据eigen value的正负号,来判断说这个critical point,比较像是saddle point,还是local minima**

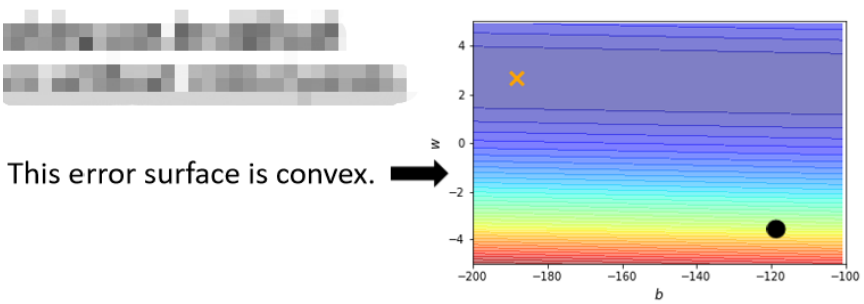
那如果实际上在训练的时候,要走到saddle point,或者是local minima,是一件困难的事情,那这个图到底是怎麼画出来的

那这边告诉大家一个秘密,这个图你要训练出这样子的结果,你要训练到你的参数很接近critical point,用一般的gradient descend,其实是做不到的,用一般的gradient descend train,你往往会得到的结果是,你在这个gradient还很大的时候,你的loss就已经掉了下去,这个是需要特别方法train的

所以做完这个实验以后,我更感觉你要走到一个critical point,其实是困难的一件事,多数时候training,在还没有走到critical point的时候,就已经停止了,那这并不代表说,critical point不是一个问题,我只是想要告诉你说,我们真正目前,**当你用gradient descend,来做optimization的时候,你真正应该要怪罪的对象,往往不是critical point,而是其他的原因,**

Training can be difficult even without critical points

如果今天critical point不是问题的话,為什麼我们的training会卡住呢,我这边举一个非常简单的例子,我这边有一个,非常简单的error surface

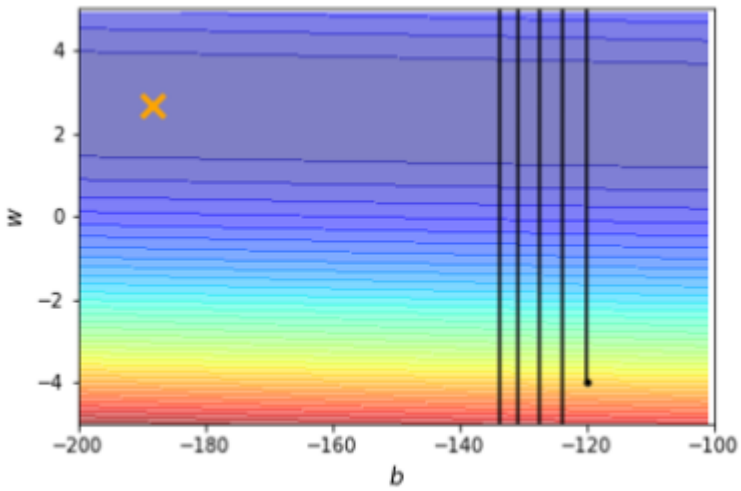


我们只有两个参数,这两个参数值不一样的时候,Loss的值不一样,我们就画出了一个error surface,这个**error surface的最低点**在==黄色X==这个地方,事实上,这个error surface是convex的形状(可以理解为凸的或者凹的，convex optimization常翻译为“凸优化”)

如果你不知道convex是什麼,没有关係,总之它是一个,它的这个等高线是椭圆形的,只是它在横轴的地方,它的gradient非常的小,它的坡度的变化非常的小,非常的平滑,所以这个椭圆的长轴非常的长,短轴相对之下比较短,在纵轴的地方gradient的变化很大,error surface的坡度非常的陡峭

那现在我们要从**黑点**这个地方,这个地方当作**初始的点**,然后来做gradient descend

你可能觉得说,这个convex的error surface,做gradient descend,有什麼难的吗？不就是一路滑下来,然后可能再走过去吗,应该是非常容易。你实际上自己试一下,你会发现说,就连这种convex的error surface,形状这麼简单的error surface,你用gradient descend,都不见得能把它做好,举例来说这个是我实际上,自己试了一下的结果

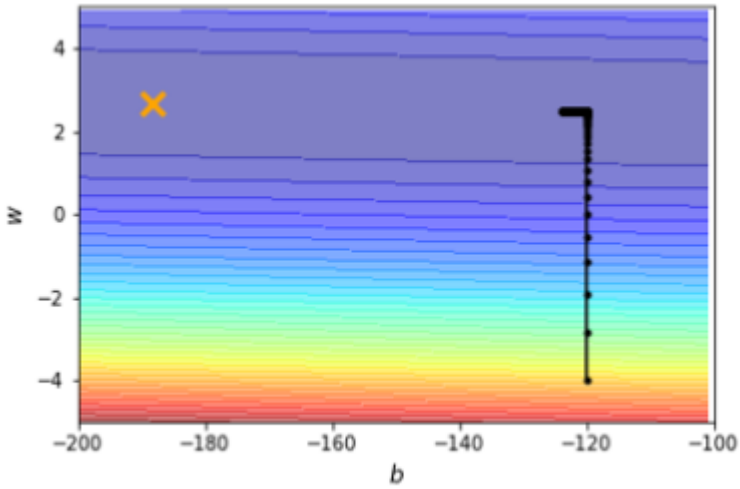


$\eta = 10^{-2}$

我learning rate设10^{-2}的时候,我的这个参数在峡谷的两端,我的参数在山壁的两端不断的震盪,我的loss掉不下去,但是gradient其实仍然是很大的

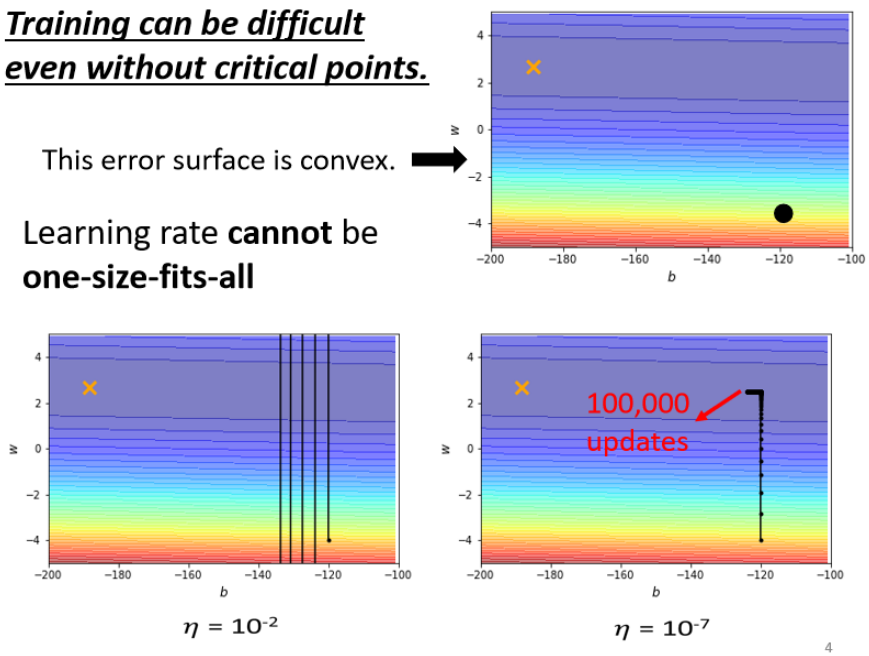
那你可能说,就是因為你**learning rate设太大了**阿,learning rate决定了我们update参数的时候步伐有多大,learning rate显然步伐太大,你没有办法慢慢地滑到山谷裡面只要把learning rate设小一点,不就可以解决这个问题了吗?

事实不然,因為我试著去,调整了这个learning rate,就会发现你光是要train这种convex的optimization的问题,你就觉得很痛苦,我就调这个learning rate,从10^{-2},一直调到10^{-7},调到10^{-7}以后,终於不再震盪了



$\eta = 10^{-7}$

终於从这个地方滑滑滑,滑到山谷底终於左转,但是你发现说,这个训练永远走不到终点,因為我的**learning rate已经太小了**,竖直往上这一段这个很斜的地方,因為这个坡度很陡,gradient的值很大,所以还能够前进一点,左拐以后这个地方坡度已经非常的平滑了,这麼小的learning rate,根本没有办法再让我们的训练前进



事实上在左拐这个地方,看到这边一大堆黑点,这边**有十万个点**,这个是张辽八百冲十万的那个十万,但是我都没有办法靠近,这个local minima的地方,所以显然**就算是一个convex的error surface,你用gradient descend也很难train**

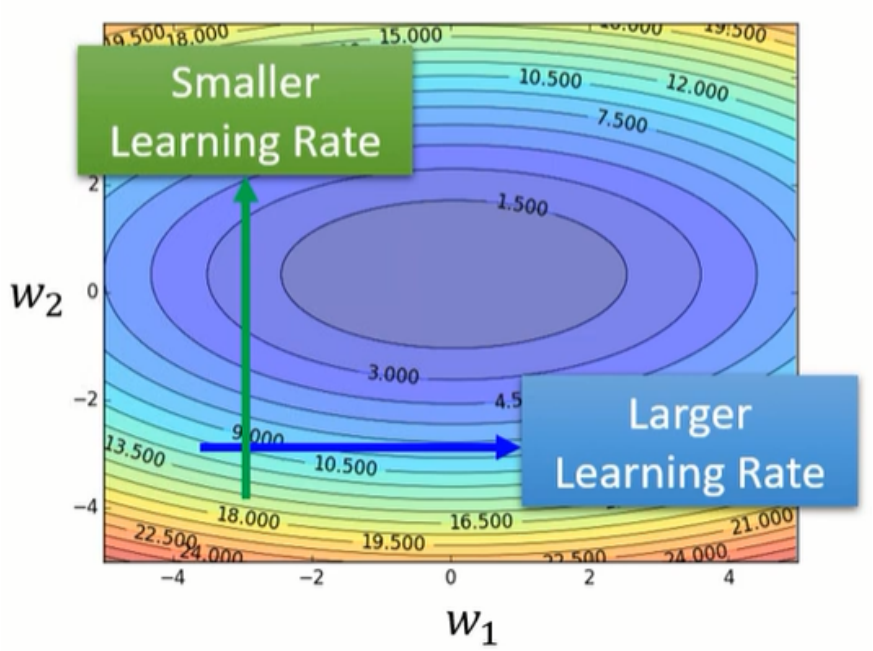
这个convex的optimization的问题,确实有别的方法可以解,但是你想想看,如果今天是更复杂的error surface,你真的要train一个deep network的时候,gradient descend是你,唯一可以仰赖的工具,但是gradient descend这个工具,连这麼简单的error surface都做不好,==一室之不治 何以天下国家為==,这麼简单的问题都做不好,那如果难的问题,它又怎麼有可能做好呢

所以我们需要更好的gradient descend的版本,在**==之前我们的gradient descend裡面,所有的参数都是设同样的learning rate,这显然是不够的,learning rate它应该要为,每一个参数客製化==**,所以接下来我们就是要讲,客製化的learning rate,怎麼做到这件事情

Different parameters needs different learning rate

那我们要怎麼客製化learning rate呢,我们不同的参数到底,需要什麼样的learning rate呢

从刚才的例子裡面,其实我们可以看到一个大原则,**如果在某一个方向上,我们的gradient的值很小,非常的平坦,那我们会希望learning rate调大一点,如果在某一个方向上非常的陡峭,坡度很大,那我们其实期待,learning rate可以设得小一点**



那这个learning rate要如何自动的,根据这个gradient的大小做调整呢

我们要改一下,gradient descend原来的式子,我们只放某一个参数update的式子,我们之前在讲gradient descend,我们往往是讲,所有参数update的式子,那这边为了等一下简化这个问题,我们只看一个参数,但是你完全可以把这个方法,推广到所有参数的状况

$$ {θ{_i}{^{t+1}}} ← {θ{_i}{^{t}}}-{\eta}{g{_i}{^{t}}}

$$ 我们只看一个参数,这个参数叫做${θ{_i}{^{t}}}$,这个${θ{_i}{^{t}}}$在第t个iteration的值,减掉在第t个iteration这个参数i算出来的gradient ${g{_i}{^{t}}}$

$$ {g{*i*}{^{t}}}=*frac{\partial{L}}{\partial{θ_i}}*|{θ=θ^t}

$$ 这个${g{_i}{^{t}}}$代表在第t个iteration,也就是θ等於$θ^t$的时候,参数$θ_i$对loss的微分,我们把这个$θ_i^t$减掉learning rate,乘上g_i^t会更新learning rate到$θ_i^{t+1}$,**这是我们原来的gradient descend,我们的learning rate是固定的**

现在我们要有一个**随著参数客製化的learning rate**,我们把原来learning rate $η$这一项呢,改写成$\frac{η}{σᵢᵗ}$

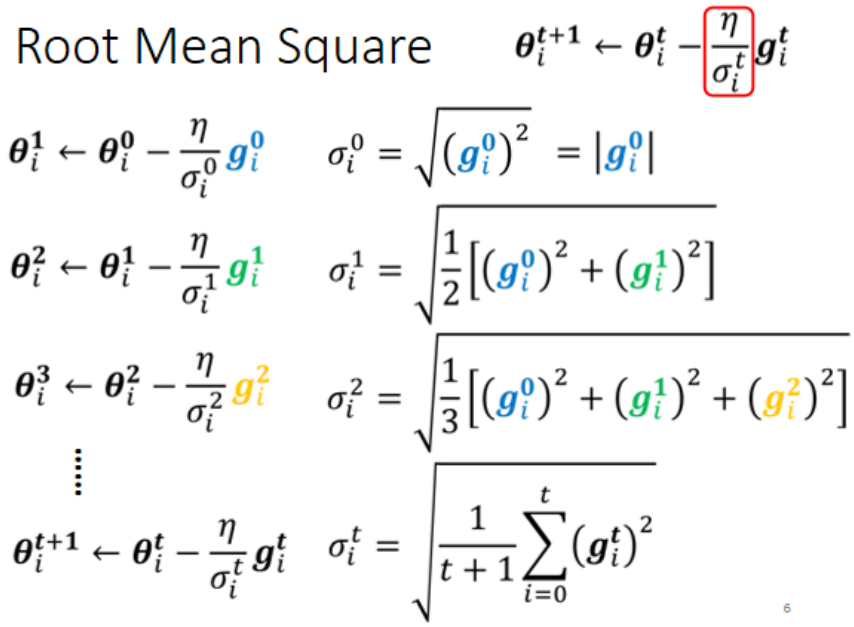
$$ {θ{_i}{^{t+1}}} ← {θ{_i}{^{t}}}-{\frac{η}{σᵢᵗ}}{g{_i}{^{t}}}

$$ 这个$σᵢᵗ$你发现它有一个上标t,有一个下标i,这代表说这个σ这个参数,首先它是depend on i的,**不同的参数我们要给它不同的σ,同时它也是iteration dependent的,不同的iteration我们也会有不同的σ**

所以当我们把我们的learning rate,从η改成$\frac{η}{σᵢᵗ}$的时候,我们就有一个,parameter dependent的learning rate,接下来我们是要看说,这个parameter dependent的learning rate有什麼常见的计算方式

Root mean square

那这个σ有什麼样的方式,可以把它计算出来呢,一个常见的类型是算,gradient的Root Mean Square



现在参数要update的式子,我们从$θᵢ^0$初始化参数减掉$gᵢ^0$,乘上learning rate η除以$σᵢ^0$,就得到$θᵢ^1$,

$$ {θ{_i}{^{1}}} ← {θ{_i}{^{0}}}-{\frac{η}{σᵢ^0}}{g{_i}{^{0}}}

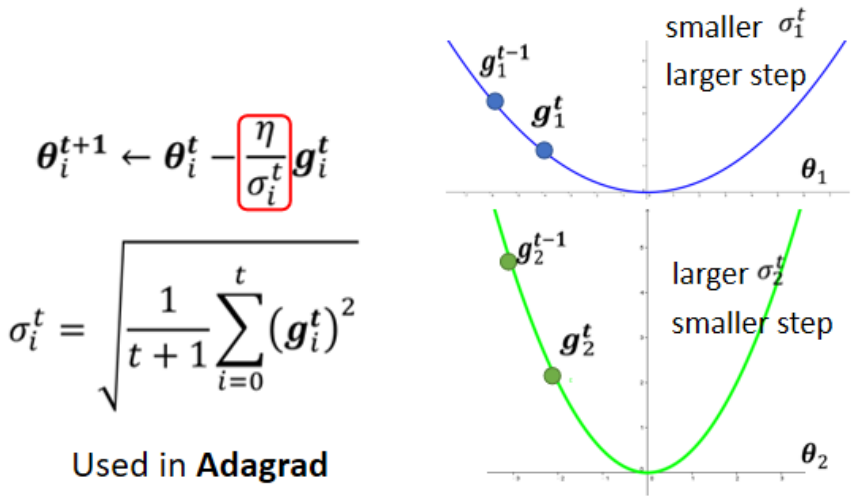
$$

- 这个**$σᵢ^0$在第一次update参数**的时候,这个$σᵢ^0$是$(gᵢ^0)²$开根号 $$ {σᵢ^0}=\sqrt{({g{_i}{^{0}}})^2}=|{g{_i}{^{0}}}| $$ 这个$gᵢ^0$就是我们的gradient,就是gradient的平方开根号,其实就是$gᵢ^0$的绝对值,所以你把$gᵢ^0$的绝对值代到${θ{_i}{^{1}}} ← {θ{_i}{^{0}}}-{\frac{η}{σᵢ^0}}{g{_i}{^{0}}}$,这个式子中$gᵢ^0$跟这个根号底下的$gᵢ^0$,它们的大小是一样的,所以式子中这一项只会有一个,要嘛是正一 要嘛是负一,就代表说我们第一次在update参数,从$θᵢ^0$update到$θᵢ^1$的时候,要嘛是加上η要嘛是减掉η,跟这个gradient的大小没有关係,是看你η设多少,这个是第一步的状况

- 重点是接下来怎麼处理,那θ_i^1它要一样,减掉gradient g_i^1乘上η除以σ_i^1, $$ {θ{_i}{^{1}}}-{\frac{η}{σᵢ^1}}{g{_i}{^{1}}} $$ 现在在第二次update参数的时候,是要除以σ_i^1 ,这个σ_i^1就是我们过去,**所有计算出来的gradient,它的平方的平均再开根号** $$ {σᵢ^1}=\sqrt{\frac{1}{2}[{(g{_i}{^{0}}})^2+{(g{_i}{^{1}}})^2]} $$ 我们到目前為止,在第一次update参数的时候,我们算出了g_i^0,在第二次update参数的时候,我们算出了g_i^1,所以这个σ_i^1就是$(g_i^0)^2$,加上$(g_i^1)^2$除以½再开根号,这个就是Root Mean Square,我们算出这个σ_i^1以后,我们的learning rate就是η除以σ_i^1,然后把θ_i^1减掉,η除以σ_i^1乘以g_i^1 得到θ_i^2 $$ {θ{_i}{^{2}}} ← {θ{_i}{^{1}}}-{\frac{η}{σᵢ^1}}{g{_i}{^{1}}} $$
- 同样的操作就反覆继续下去,在θ_i^2的地方,你要减掉η除以σ_i^2乘以g_i^2, $$ {θ{_i}{^{2}}}-{\frac{η}{σᵢ^2}}{g{_i}{^{2}}} $$ 那这个σ是什麼呢,这个σ_i^2就是过去,所有算出来的gradient,它的平方和的平均再开根号 $$ {σᵢ^2}=\sqrt{\frac{1}{3}[{(g{_i}{^{0}}})^2+{(g{_i}{^{1}}})^2+{(g{_i}{^{2}}})^2]} $$ 所以你把g_i^0取平方,g_i^1取平方g_i^2取平方,的平均再开根号,得到σ_i^2放在这个地方,然后update参数 $$ {θ{_i}{^{3}}} ← {θ{_i}{^{2}}}-{\frac{η}{σᵢ^2}}{g{_i}{^{2}}} $$
- 所以这个process这个过程,就反覆继续下去,到第t次update参数的时候,其实这个是第t + 1次,第t + 1次update参数的时候,你的这个σ_i^t它就是过去所有的gradient,g_i^t从第一步到目前為止,所有算出来的g_i^t的平方和,再平均 再开根号得到σ_i^t, $$ {σᵢ^t}=\sqrt{\frac{1}{t+1}\sum_{i=0}^{t}{(g{_i}{^{t}}})^2} $$ 然后在把它除learning rate,然后用这一项当作是,新的learning rate来update你的参数, $$ {θ{_i}{^{t+1}}} ← {θ{_i}{^{t}}}-{\frac{η}{σᵢ^t}}{g{_i}{^{t}}} $$

Adagrad

那这一招被用在一个叫做==Adagrad==的方法裡面,**為什麼这一招可以做到我们刚才讲的,坡度比较大的时候,learning rate就减小,坡度比较小的时候,learning rate就放大呢?**



你可以想像说,现在我们有两个参数:**一个叫θ_i^1 一个叫θ_i^2 θ_i^1坡度小 θ_i^2坡度大**

- θ_i^1因為它坡度小,所以你在θ_i^1这个参数上面,算出来的gradient值都比较小
- 因為gradient算出来的值比较小,然后这个σ是gradient的平方和取平均再开根号

$$ {σᵢ^t}=\sqrt{\frac{1}{t+1}\sum_{i=0}^{t}{(g{_i}{^{t}}})^2}

$$

- 所以算出来的σ就小,σ小 learning rate就大 $$ {\frac{η}{σᵢ^t}} $$

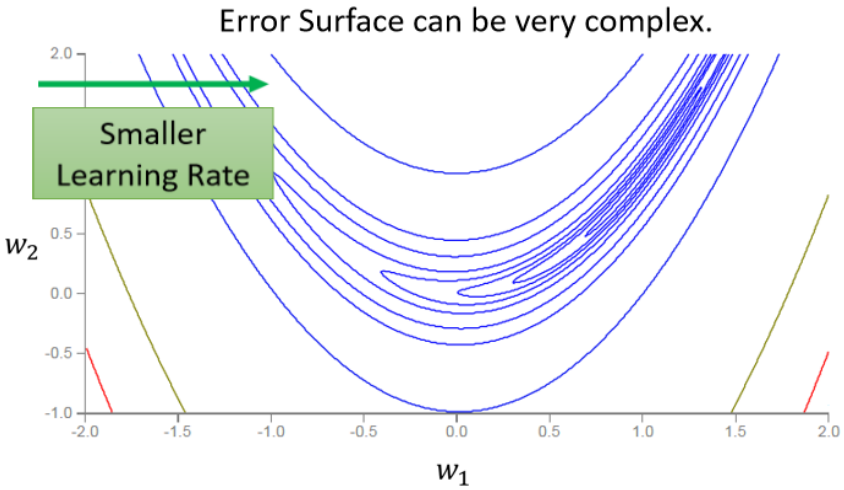
反过来说$θᵢ^2$,$θᵢ^2$是一个比较陡峭的参数,在$θᵢ^2$这个方向上loss的变化比较大,所以算出来的gradient都比较大,,你的σ就比较大,你在update的时候 你的step,你的参数update的量就比较小

所以有了σ这一项以后,你就可以随著gradient的不同,每一个参数的gradient的不同,来自动的调整learning rate的大小,那这个并不是,你今天会用的最终极的版本,

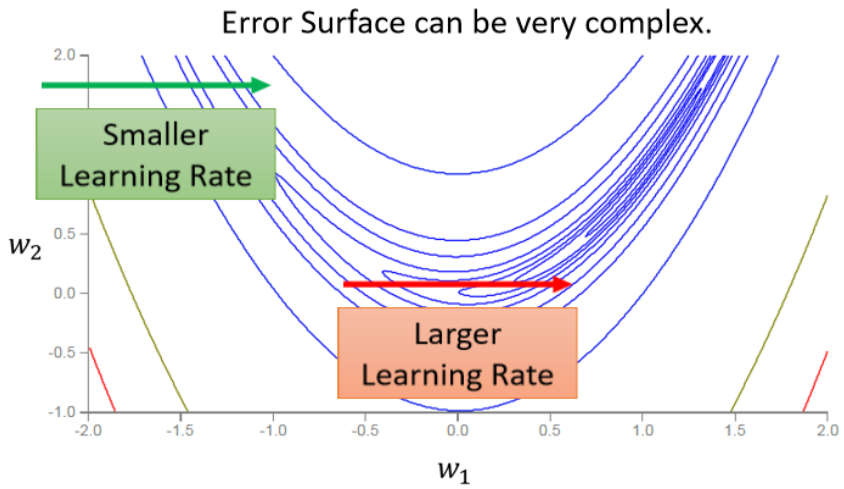
RMSProp

刚才那个版本,就算是同一个参数,它需要的learning rate,也会随著时间而改变,我们刚才的假设,好像是同一个参数,它的gradient的大小,就会固定是差不多的值,但事实上并不一定是这个样子的

举例来说我们来看,这个新月形的error surface



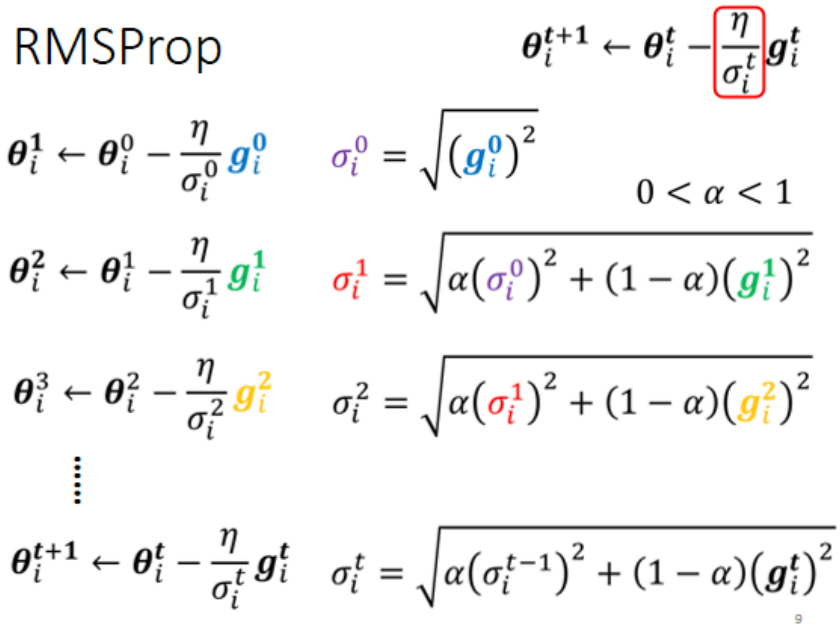
如果我们考虑横轴的话,考虑左右横的水平线的方向的话,你会发现说,在绿色箭头这个地方坡度**比较陡峭,所以我们需要比较小的learning rate**,



但是走到了中间这一段，到了红色箭头的时候呢,坡度又变得平滑了起来,**平滑了起来就需要比较大的learning rate**,所以就算是**同一个参数同一个方向,我们也期待说,learning rate是可以动态的调整的**,于是就有了一个新的招数,这个招数叫做

==RMS Prop==

RMS Prop这个方法有点传奇,它传奇的地方在於它找不到论文,非常多年前应该是将近十年前,Hinton在Coursera上,开过deep learning的课程,那个时候他在他的课程裡面,讲了RMS Prop这个方法,然后这个方法没有论文,所以你要cite的话,你要cite那个影片的连结,这是个传奇的方法叫做RMS Prop



RMS Prop这个方法,**它的第一步跟刚才讲的Root Mean Square,也就是那个Apagrad的方法,是一模一样的**

$$ {σᵢ^0}=\sqrt{({g_i^0})^2}

$$ 我们看第二步,一样要算出$σ_i^1$,只是我们现在算出$σ_i^1$的方法跟刚才,算Root Mean Square的时候不一样,刚才在算Root Mean Square的时候,每一个gradient都有同等的重要性,但**在RMS Prop裡面,它决定你可以自己调整,现在的这个gradient,你觉得它有多重要**

$$ {σᵢ^1}=\sqrt[]{\alpha(σ_i^0)^2+(1-\alpha)(g_i^1)^2}

$$ 所以在RMS Prop裡面,我们这个$σ_i^1$它是前一步算出来的$σ_i^0$,裡面就是有g_i^0,所以这个**$σ_i^0$就代表了g_i^0的大小**,所以它是$(σ_i^0)^2$,乘上α加上(1-α),乘上现在我们刚算出来的,新鲜热腾腾的gradient就是g_i^1

那这个**α就像learning rate一样,这个你要自己调它,它是一个hyperparameter**

- 如果我今天**α设很小趋近於0**,就代表我觉得**g_i^1相较於之前所算出来的gradient而言,比较重要**
- 我**α设很大趋近於1**,那就代表我觉得**现在算出来的g_i^1比较不重要,之前算出来的gradient比较重要**

所以同理在第三次update参数的时候,我们要算$σ_i^2$,我们就把$σ_i^1$拿出来取平方再乘上α,那$σ_i^1$裡面有g_i^1跟$σ_i^0$,$σ_i^0$裡面又有g_i^0,所以你知道$σ_i^1$裡面它有g_i^1有g_i^0, 然后这个g_i^1跟g_i^0呢他们会被乘上α,然后再加上1-α乘上这个$(g_i^2)^2$

$$ {σᵢ^2}=\sqrt[]{\alpha(σ_i^1)^2+(1-\alpha)(g_i^2)^2}

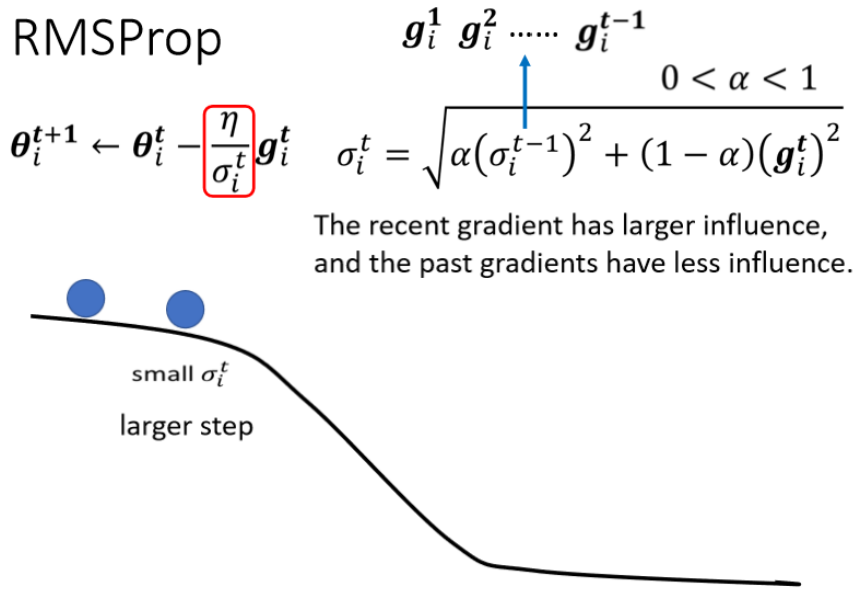
$$ 所以这个α就会决定说g_i^2,它在整个σ_i^2裡面佔有多大的影响力

那同样的过程就反覆继续下去,σ_i^t等於根号α乘上$(\sigma_i^{t-1})^2$,加上(1-α) $(g_i^t)^2$,

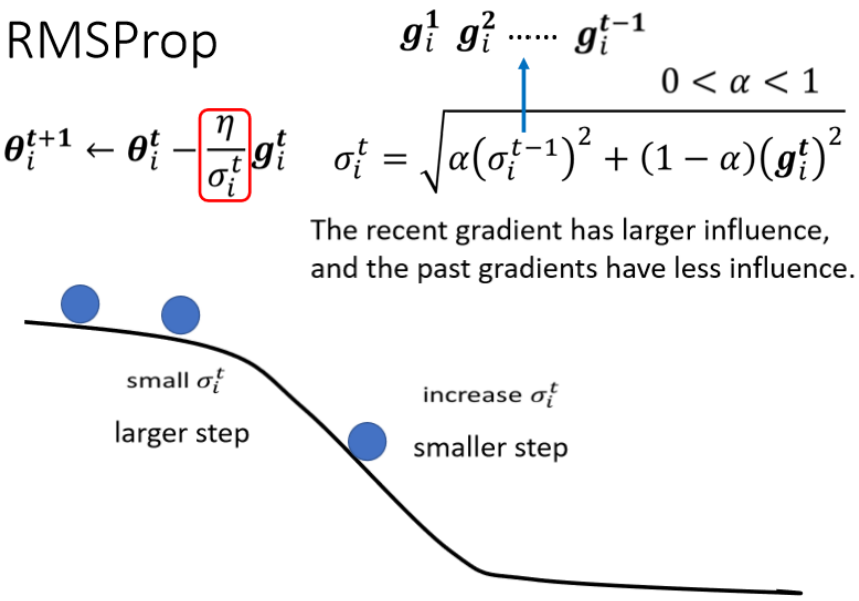
$$ {σᵢ^t}=\sqrt[]{\alpha(σ_i^{t-1})^2+(1-\alpha)(g_i^t)^2}

$$ 你用α来决定现在刚算出来的g_i^t,它有多重要,好那这个就是RMSProp

那RMSProp我们刚刚讲过说,透过α这一项你可以决定说,g_i^t相较於之前存在,σ_i^{t-1}裡面的g_i^t到g_i^{t-1}而言,它的重要性有多大,如果你用RMS Prop的话,你就可以动态调整σ这一项,我们现在假设从这个地方开始

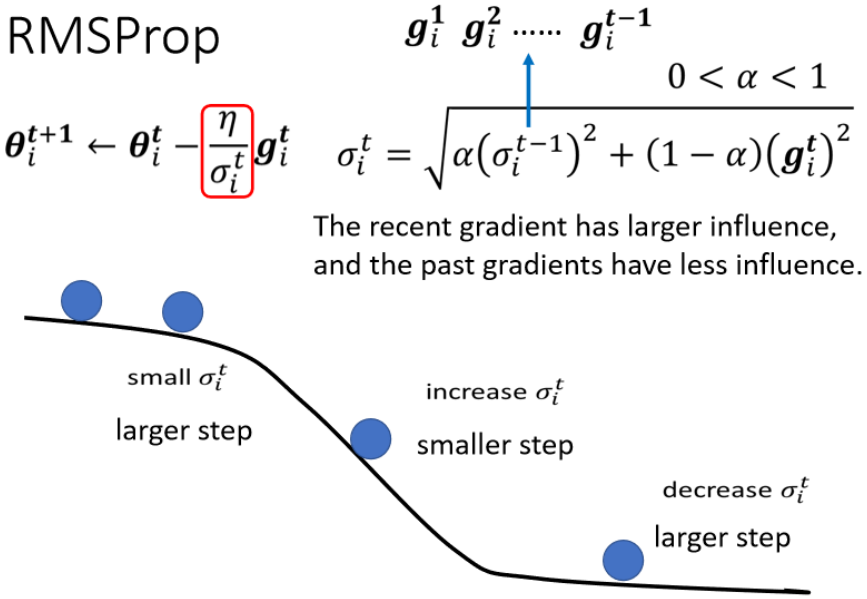


这个黑线是我们的error surface,从这个地方开始你要update参数,好你这个球就从这边走到这边,那因為一路上都很平坦,很平坦就代表说g算出来很小,代表现在update参数的时候,我们会走比较大的步伐



接下来继续滚,滚到这边以后我们gradient变大了,如果不是RMS Prop,原来的Adagrad的话它反应比较慢,但如果你用RMS Prop,然后呢你把α设小一点,你就是让新的,刚看到的gradient影响比较大的话,那你就可以很快的让σ的值变大,也可以很快的让你的步伐变小

你就可以踩一个煞车,本来很平滑走到这个地方,突然变得很陡,那RMS Prop可以很快的踩一个煞车,把learning rate变小,如果你没有踩刹车的话,你走到这裡这个地方,learning rate太大了,那gradient又很大,两个很大的东西乘起来,你可能就很快就飞出去了,飞到很远的地方



如果继续走,又走到平滑的地方了,因為这个σᵢᵗ 你可以调整α,让它比较看重於,最近算出来的gradient,所以你gradient一变小,σ可能就反应很快,它的这个值就变小了,然后呢你走的步伐就变大了,这个就是RMS Prop,

Adam

那今天你最常用的,optimization的策略,有人又叫做optimizer,今天最常用的optimization的策略,就是Adam

Adam: RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

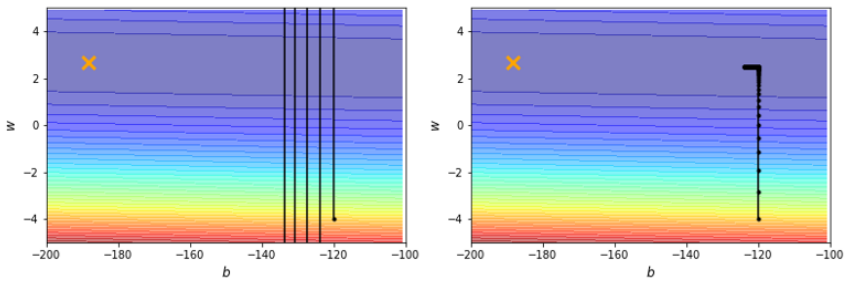
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
$v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Adam就是RMS Prop加上Momentum,那Adam的演算法跟原始的论文
https://arxiv.org/pdf/1412.6980.pdf

今天pytorch裡面,都帮你写得好好的了,所以这个你今天,不用担心这种optimization的问题,optimizer这个deep learning的套件,往往都帮你做好了,然后这个optimizer裡面,也有一些参数需要调,也有一些hyperparameter,需要人工决定,但是你往往用预设的,那一种参数就够好了,你自己调有时候会调到比较差的,往往你直接copy,这个pytorch裡面,Adam这个optimizer,然后预设的参数不要随便调,就可以得到不错的结果了,关於Adam的细节,就留给大家自己研究

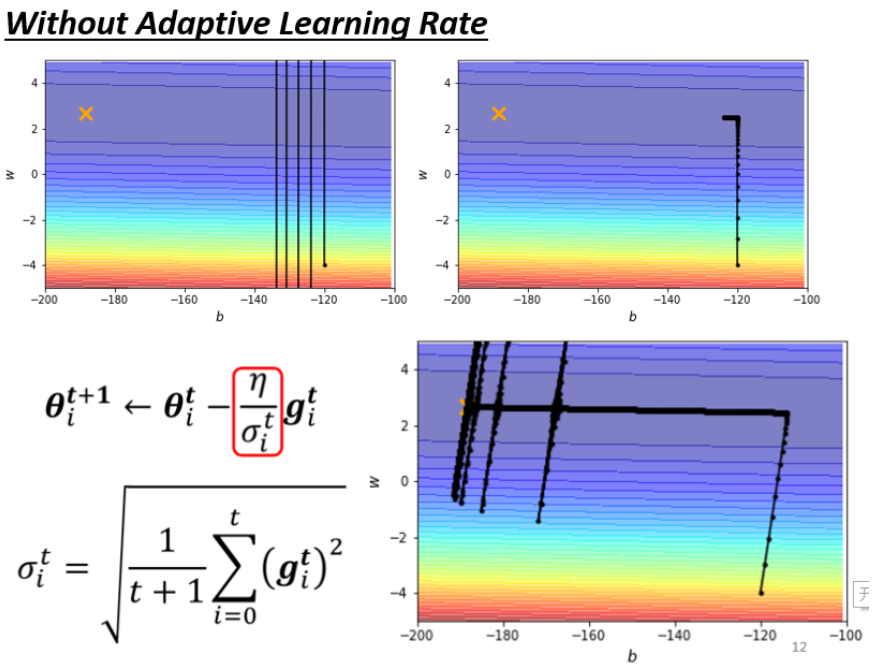
Learning Rate Scheduling

Without Adaptive Learning Rate



我们刚才讲说这个简单的error surface,我们都train不起来,现在我们来看一下,加上Adaptive Learning Rate以后,train不train得起来,

那这边是採用,最原始的Adagrad那个做法啦,就是把过去看过的,这个learning rate通通都,过去看过的gradient,通通都取平方再平均再开根号当作这个σ ,做起来是这个样子的



这个走下来没有问题,然后接下来在左转的时候,这边也是update了十万次,之前update了十万次,只卡在左转这个地方

那现在有Adagrad以后,你可以再继续走下去,走到非常接近终点的位置,因为当你走到这个地方的时候,你因為这个左右的方向的,这个gradient很小,所以learning rate会自动调整,左右这个方向的,learning rate会自动变大,所以你这个步伐就可以变大,就可以不断的前进

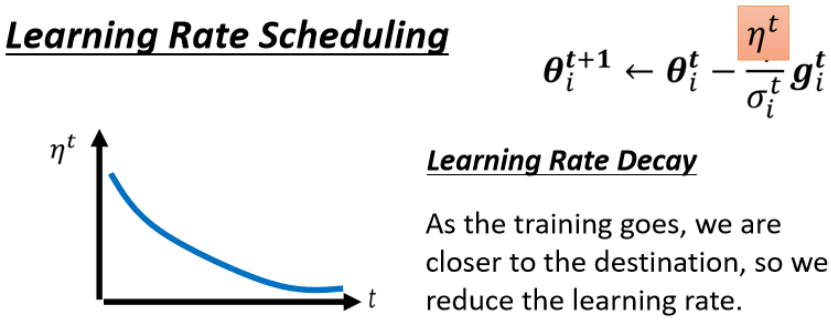
接下来的问题就是,為什麼快走到终点的时候突然爆炸了呢

你想想看 我们在做这个σ的时候,我们是把过去所有看到的gradient,都拿来作平均

- 所以这个纵轴的方向,在这个初始的这个地方,感觉gradient很大
- 可是这边走了很长一段路以后,这个纵轴的方向,gradient算出来都很小,所以纵轴这个方向,这个y轴的方向就累积了很小的σ
- 因為我们在这个y轴的方向,看到很多很小的gradient,所以我们就累积了很小的σ,累积到一个地步以后,这个step就变很大,然后就爆走就喷出去了
- 喷出去以后没关係,有办法修正回来,因為喷出去以后,就走到了这个gradient比较大的地方,走到gradient比较大的地方以后,这个σ又慢慢的变大,σ慢慢变大以后,这个参数update的距离,Update的步伐大小就慢慢的变小

你就发现说走著走著,突然往左右喷了一下,但是这个喷了一下不会永远就是震盪,不会做简谐运动停不下来,这个力道慢慢变小,有摩擦力 让它慢慢地慢慢地,又回到中间这个峡谷来,然后但是又累计一段时间以后 又会喷,然后又慢慢地回来 怎麼办呢,**有一个方法也许可以解决这个问题,这个叫做learning rate的scheduling**

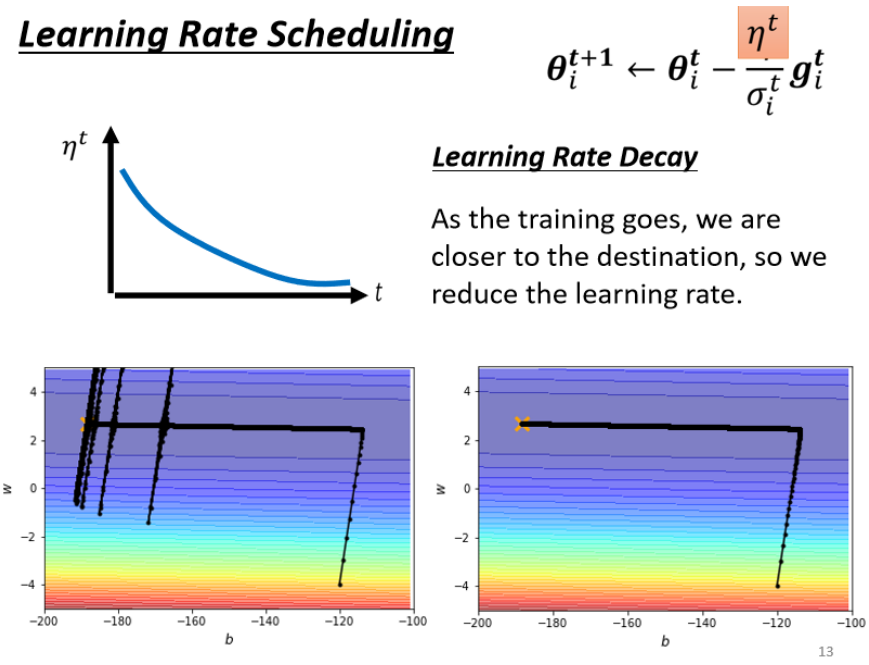
什麼是learning rate的scheduling呢



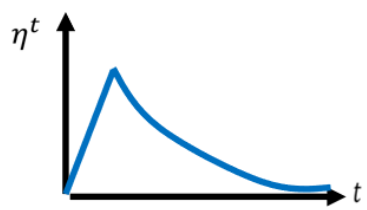
我们刚才这边还有一项η,这个η是一个固定的值,learning rate scheduling的意思就是说,我们**不要把η当一个常数,我们把它跟时间有关**

最常见的策略叫做==Learning Rate Decay==,也就是说，**随著时间的不断地进行,随著参数不断的update,我们这个η让它越来越小**

那这个也就合理了,因為一开始我们距离终点很远,随著参数不断update,我们距离终点越来越近,所以我们把learning rate减小,让我们参数的更新踩了一个煞车,让我们参数的更新能够慢慢地慢下来,所以刚才那个状况,如果加上Learning Rate Decay有办法解决



刚才那个状况,如果加上Learning Rate Decay的话,我们就可以很平顺的走到终点,因為在这个地方,这个η已经变得非常的小了,虽然说它本来想要左右乱喷,但是因为乘上这个非常小的η,就停下来了 就可以慢慢地走到终点,那除了Learning Rate Decay以外,还有另外一个经典，常用的Learning Rate Scheduling的方式,叫做==Warm Up==



Warm Up

Increase and then decrease?

Warm Up这个方法,听起来有点匪夷所思,这Warm Up的方法是**让learning rate,要先变大后变小**,你会问说 变大要变到多大呢,变大速度要多快呢， 小速度要多快呢,**这个也是hyperparameter**,你要自己用手调的,但是大方向的大策略就是,learning rate要先变大后变小,那这个方法听起来很神奇,就是一个黑科技这样,这个黑科技出现在,很多远古时代的论文裡面

这个warm up,最近因為在训练BERT的时候,往往需要用到Warm Up,所以又被大家常常拿出来讲,但它并不是有BERT以后,才有Warm Up的,Warm Up这东西远古时代就有了,举例来说,Residual Network裡面是有Warm Up的

We further explore $n = 18$ that leads to a 110-layer ResNet. In this case, we find that the initial learning rate of 0.1 is slightly too large to start converging[5]. So we use 0.01 to warm up the training until the training error is below 80% (about 400 iterations), and then go back to 0.1 and continue training. The rest of the learning schedule is as done previously. This 110-layer network converges well (Fig. 6, middle). It has *fewer* parameters than other deep and thin

[5]With an initial learning rate of 0.1, it starts converging (<90% error) after several epochs, but still reaches similar accuracy.

Residual Network

https://arxiv.org/abs/1512.03385

这边是放了Residual network,放在arXiv上面的文章连结啦,今天这种有关machine learning 的,文章往往在投conference之前,投国际会议之前,就先放到一个叫做arXiv的网站上,把它公开来让全世界的人都可以看

你其实看这个arXiv的网址,你就可以知道,这篇文章是什麼时候放到网路上的,怎麼看呢 arXiv的前四个数字,这15代表年份,代表说residual network这篇文章,是2015年放到arXiv上面的,后两个数字代表月份,所以它是15年的12月,15年的年底放在arXiv上面的

所以五六年前的文章,在deep learning变化,这麼快速的领域裡面,五六年前就是上古时代,那在上古时代,这个Residual Network裡面,就已经记载了Warm Up的这件事情,它说我们**用learning rate 0.01,取Warm Up,先用learning rate 0.01,再把learning rate改成0.1**

用过去我们通常最常见的train,Learning Rate Scheduling的方法,就是让learning rate越来越小,但是Residual Network,这边特别註明它反其道而行,一开始要设0.01接下来设0.1,还特别加一个註解说,一开始就用0.1反而就train不好,不知道為什麼 也没解释,反正就是train不好,需要Warm Up这个黑科技。

而在这个黑科技,在知名的Transformer裡面(这门课也会讲到),也用一个式子提了它

5.3 Optimizer

We used the Adam optimizer [17] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (3)$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

Transformer https://arxiv.org/abs/1706.03762

它这边有一个式子说,它的learning rate遵守这一个,神奇的function来设定,它的learning rate,这个神奇的function,乍看之下会觉得 哇 在写什麼,不知道在写些什麼

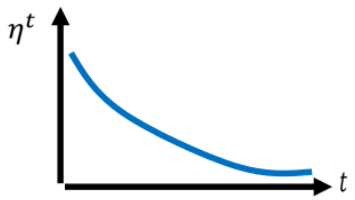
这个东西你实际上,把这个function画出来,你实际上把equation画出来的话,就会发现它就是Warm Up,learning rate会先增加,然后接下来再递减

所以你发现说Warm Up这个技术,在很多知名的network裡面都有,被当作一个黑科技,就论文裡面不解释说,為什麼要用这个,但就偷偷在一个小地方,你没有注意到的小地方告诉你说,这个network要用这种黑科技,才能够把它训练起来

那為什麼需要warm Up呢,这个仍然是今天,一个可以研究的问题啦

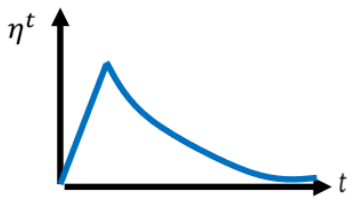
Learning Rate Scheduling

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} g_i^t$$



Learning Rate Decay

After the training goes, we are close to the destination, so we reduce the learning rate.



Warm Up

Increase and then decrease?

At the beginning, the estimate of σ_i^t has large variance.

Please refer to **RAdam** https://arxiv.org/abs/1908.03265

这边有一个可能的解释是说,你想想看当我们在用Adam RMS Prop,或Adagrad的时候,我们会需要计算σ,它是一个统计的结果,**σ告诉我们,某一个方向它到底有多陡,或者是多平滑**,那这个统计的结果,**要看得够多笔数据以后,这个统计才精準,所以一开始我们的统计是不精準的**

一开始我们的σ是不精準的,所以我们一开始不要让我们的参数,走离初始的地方太远,先让它在初始的地方呢,做一些像是探索这样,所以**一开始learning rate比较小,是让它探索 收集一些有关error surface的情报**,先收集有关σ的统计数据,**等σ统计得比较精準以后,在让learning rate呢慢慢地爬升**

所以这是一个解释,為什麼我们需要warm up的可能性,那如果你想要学更多,有关warm up的东西的话,你其实可以看一篇paper,它是Adam的进阶版叫做RAdam,裡面对warm up这件事情,有更多的理解

那有关optimization的部分,其实我们就讲到这边啦,

Summary of Optimization

所以我们从最原始的gradient descent,进化到这一个版本

(Vanilla) Gradient Descent

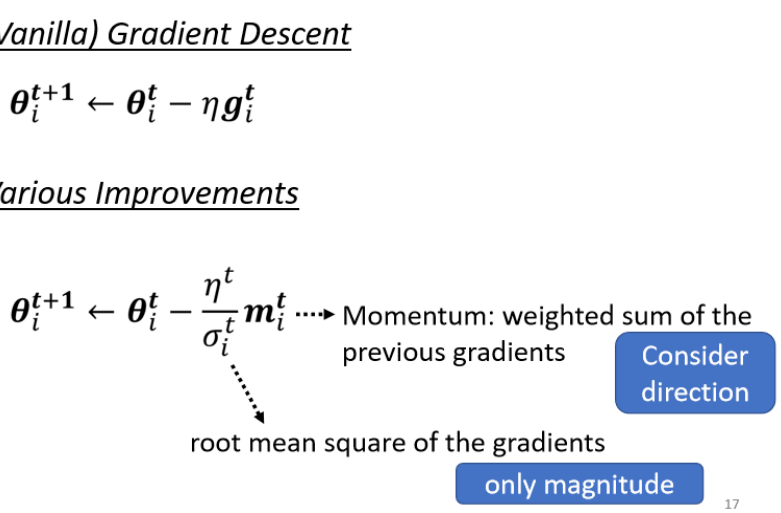
$$\theta_i^{t+1} \leftarrow \theta_i^t - \eta g_i^t$$

Various Improvements

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\eta^t}{\sigma_i^t} m_i^t$$ ⋯▸ Momentum: weighted sum of the previous gradients

这个版本裡面

- 我们有Momentum,也就是说我们现在,不是完全顺著gradient的方向,现在不是完全顺著这一个时间点,算出来的gradient的方向,来update参数,而是把过去,所有算出来gradient的方向,做一个加总当作update的方向,这个是momentum
- 接下来应该要update多大的步伐呢,我们要除掉,gradient的Root Mean Square



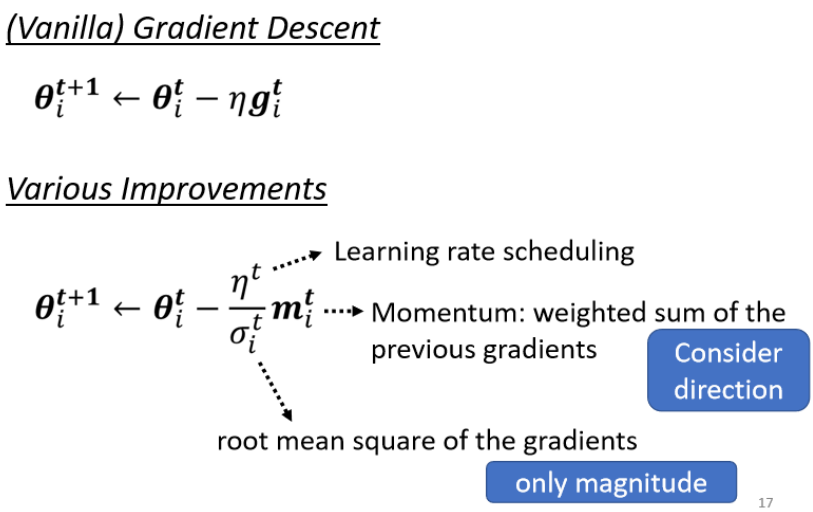
那讲到这边可能有同学会觉得很困惑,这一个momentum是考虑,过去所有的gradient,这个σ也是考虑过去所有的gradient,一个放在分子一个放在分母,都考虑过去所有的gradient,不就是正好**抵销了吗**,

但是其实这个Momentum跟这个σ,它们在使用过去所有gradient的方式是不一样的,**Momentum是直接把所有的gradient通通都加起来**,所以它有考虑**方向**,它有考虑gradient的**正负号**,它有考虑gradient**是往左走还是往右走**

但是这个**Root Mean Square,它就不考虑gradient的方向**了,它**只考虑gradient的大小**,记不记得我们在算σ的时候,我们都要取平方项,我们都要把gradient取一个平方项,我们是把平方的结果加起来,所以我们只考虑gradient的大小,不考虑

它的方向,所以Momentum跟这个σ,算出来的结果并不会互相抵销掉

- 那最后我们还会加上,一个learning rate的scheduling,

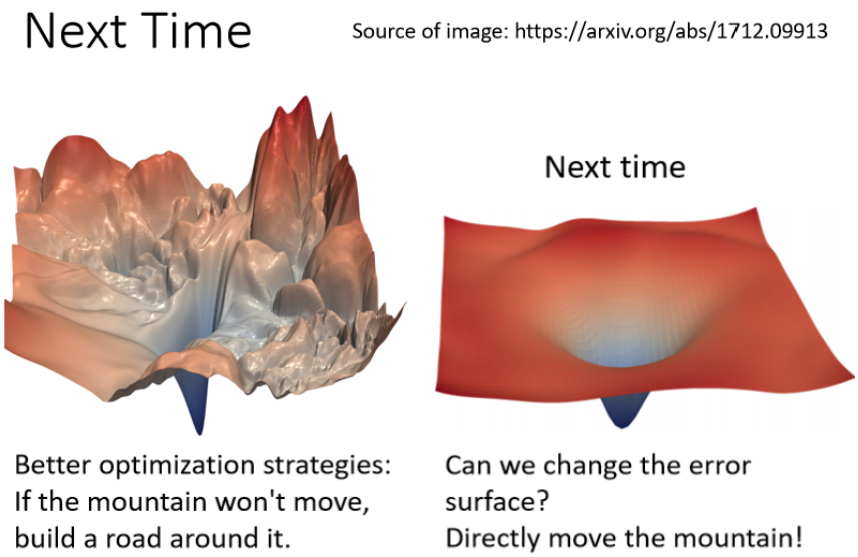


那这个是今天optimization的,完整的版本了,这种Optimizer,除了Adam以外,Adam可能是今天最常用的,但除了Adam以外,还有各式各样的变形,但其实各式各样的变形都不脱,就是要嘛不同的方法算M,要嘛不同的方法算σ,要嘛不同的,Learning Rate Scheduling的方式

那如果你想要知道更多,跟optimization有关的事情的话,那有之前助教的录影,给大家参考到这裡,影片蛮长的大概两个小时,所以你可以想见说,有关Optimizer的东西,其实是还有蛮多东西可以讲的,所以时间的关係我们就不讲下去



到目前為止呢 我们讲的是什麼,我们讲的是,当我们的error surface非常的崎嶇,就像这个例子一样非常的崎嶇的时候



我们需要一些比较好的方法,来做optimization,前面有一座山挡著,我们希望可以绕过那座山,山不转路转的意思这样,你知道这个gradient,这奇怪的error surface,会让人觉得很痛苦

那就要用神罗天征,把这个炸平这样子,所以接下来我们会讲的技巧,就是有没有可能,直接把这个error surface移平,我们改Network裡面的什麼东西,改Network的架构activation function,或者是其它的东西,直接移平error surface,让它变得比较好train,也就是山挡在前面,就把山直接剷平的意思

Quick Introduction of Batch Normalization

本篇是一个很快地介绍,Batch Normalization 这个技术

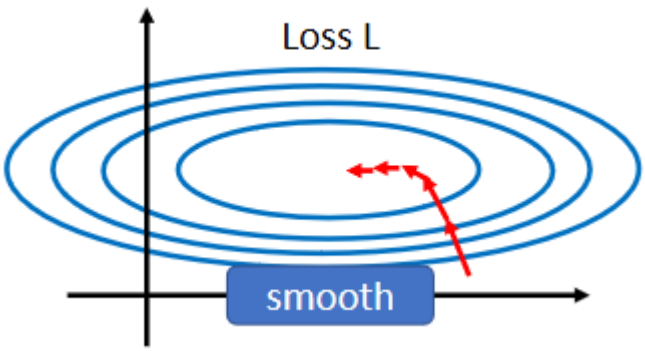
Changing Landscape

之前才讲过说,我们能不能够直接改error surface 的 landscape,我们觉得说 error surface 如果很崎嶇的时候,它比较难 train,那我们能不能够直接把山剷平,让它变得比较好 train 呢?

==Batch Normalization== 就是其中一个,**把山剷平的想法**

我们一开始就跟大家讲说,不要小看 optimization 这个问题,有时候就算你的 error surface 是 convex的,它就是一个碗的形状,都不见得很好 train

假设你的两个参数啊,它们对 **Loss 的斜率差别非常大**,在 w_1 这个方向上面,你的斜率变化很小,在 w_2 这个方向上面斜率变化很大

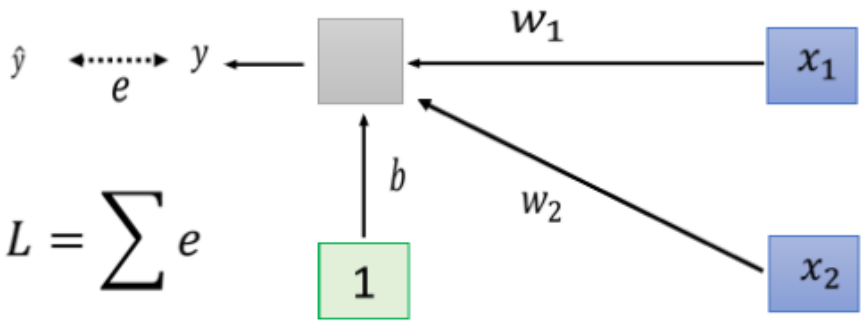


如果是**固定的 learning rate**,你可能很难得到好的结果,所以我们才说你需要 adaptive 的 learning rate、 Adam 等等比较进阶的 optimization 的方法,才能够得到好的结果

现在我们要从另外一个方向想,**直接把难做的 error surface 把它改掉**,看能不能够改得好做一点

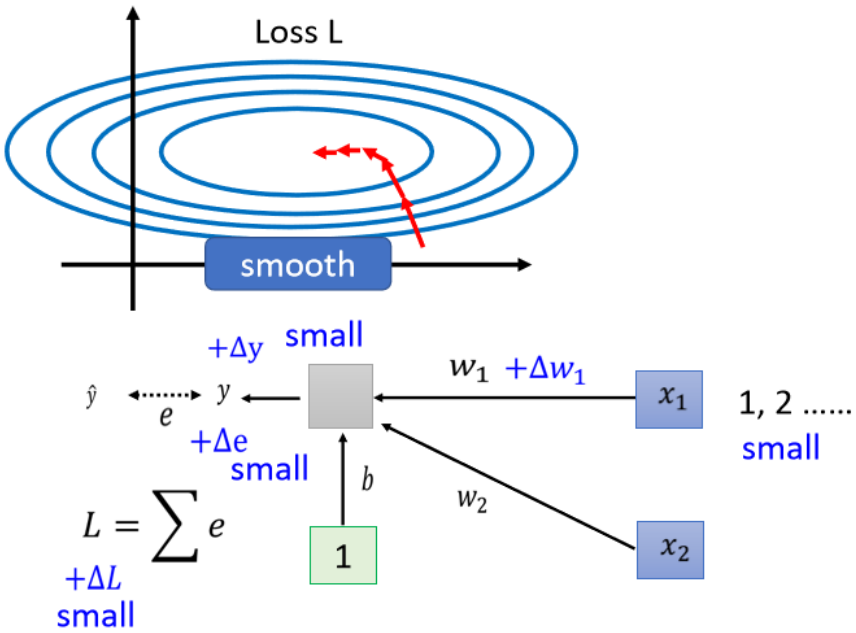
在做这件事之前,也许我们第一个要问的问题就是,有这一种状况,$w_1 $ 跟 w_2 它们的**斜率差很多**的这种状况,到底是从什麼地方来的

假设我现在有一个非常非常非常简单的 model,它的输入是 x_1 跟 x_2,它对应的参数就是 $ w_1 $ 跟 w_2,它是一个 linear 的 model,没有 activation function



$ w_1 $ 乘 x_1,w_2 乘 x_2 加上 b 以后就得到 y,然后会计算 y 跟 $\hat{y}$ 之间的差距当做 e,把所有 training data e 加起来就是你的 Loss，然后去 minimize 你的 Loss

那什麼样的状况我们会產生像上面这样子,**比较不好 train 的 error surface** 呢?

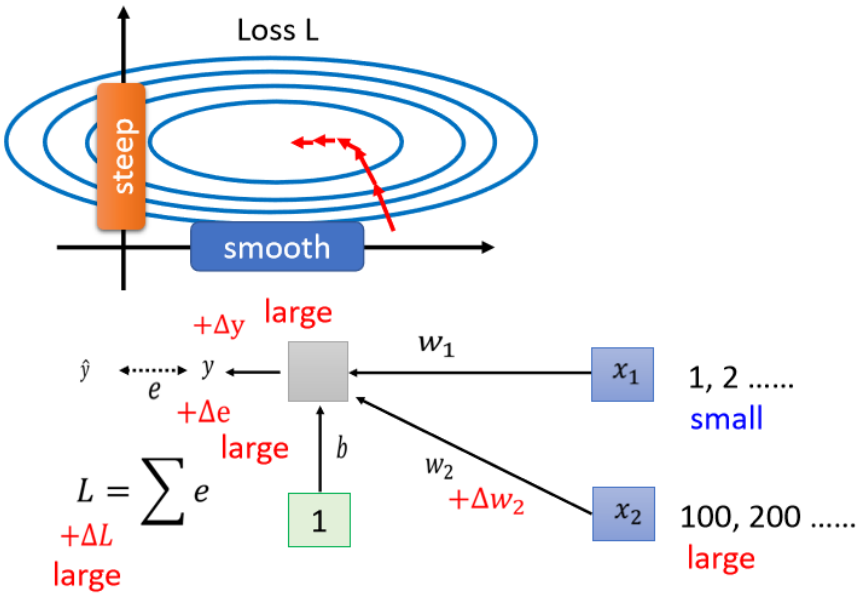


当我们对 **$ w_1 $ 有一个小小的改变**,比如说加上 delta $ w_1 $ 的时候,那这个 L 也会有一个改变,那这个 $ w_1 $ 呢,是透过 $ w_1 $ 改变的时候,你就改变了 y,y 改变的时候你就改变了 e,然后接下来就**改变了 L**

那什麼时候 $ w_1 $ 的改变会对 L 的影响很小呢,也就是它在 error surface 上的斜率会很小呢

一个可能性是当你的 **input 很小的时候**,假设 x_1 的值在不同的 training example 裡面,它的值都很小,那因为 x_1 是直接乘上 $ w_1 $，如果 x_1 的值都很小,$ w_1 $ 有一个变化的时候,它得到的,它**对 y 的影响也是小的**,对 e 的影响也是小的,它对 L 的影响就会是小的

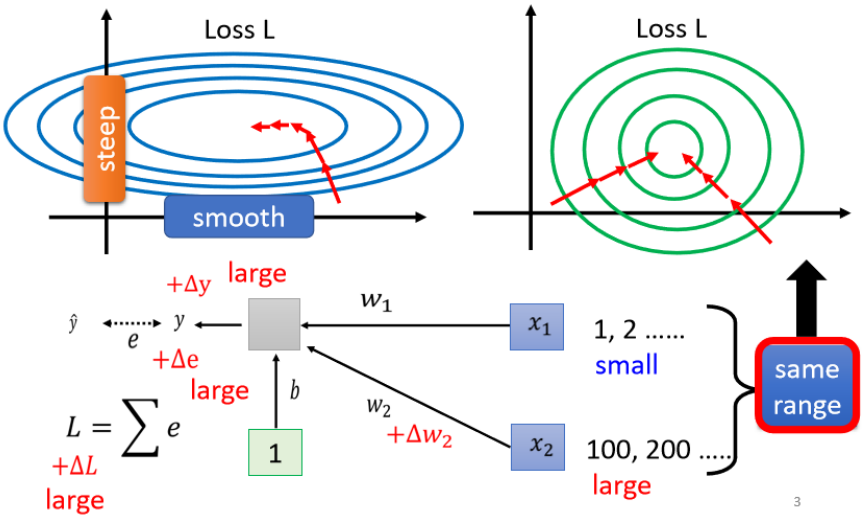
反之呢,如果今天是 x_2 的话



那假设 **x_2 的值都很大**,当你的 w_2 有一个小小的变化的时候,虽然 w_2 这个变化可能很小,但是因为它乘上了 x_2,x_2 的值很大,那 y 的变化就很大,那 e 的变化就很大,那 L 的变化就会很大,就会导致我们在 w 这个方向上,做变化的时候,我们把 w 改变一点点,那我们的 error surface 就会有很大的变化

所以你发现说,既然在这个 linear 的 model 裡面,当我们 input 的 feature,**每一个 dimension 的值,它的 scale 差距很大**的时候,我们就可能產生像这样子的 error surface,就可能產生**不同方向,斜率非常不同,坡度非常不同的 error surface**

所以怎麼办呢,我们有没有可能给feature 裡面**不同的 dimension,让它有同样的数值的范围**

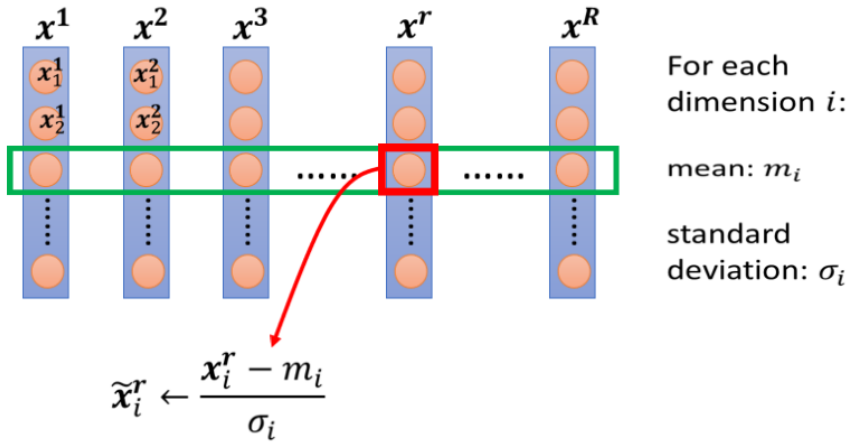


如果我们可以给不同的 dimension,同样的数值范围的话,那我们可能就可以製造比较好的 error surface,让 training 变得比较容易一点

其实有很多不同的方法,这些不同的方法,往往就合起来统称为==Feature Normalization==

Feature Normalization

以下所讲的方法只是Feature Normalization 的一种可能性,它**并不是 Feature Normalization 的全部**,假设 x^1 到 x^R,是我们所有的训练资料的 feature vector



我们把所有训练资料的 feature vector ,统统都集合起来,那每一个 vector ,x_1 裡面就 $x^1_1 $代表 x_1 的第一个 element,$x^2_1 $,就代表 x_2 的第一个 element,以此类推

那我们把**不同笔资料即不同 feature vector,同一个 dimension** 裡面的数值,把它取出来,然后去计算某一个 dimension 的 mean，它的 mean 呢 就是m_i，我们计算第 i 个 dimension 的,standard deviation,我们用σ_i来表示它

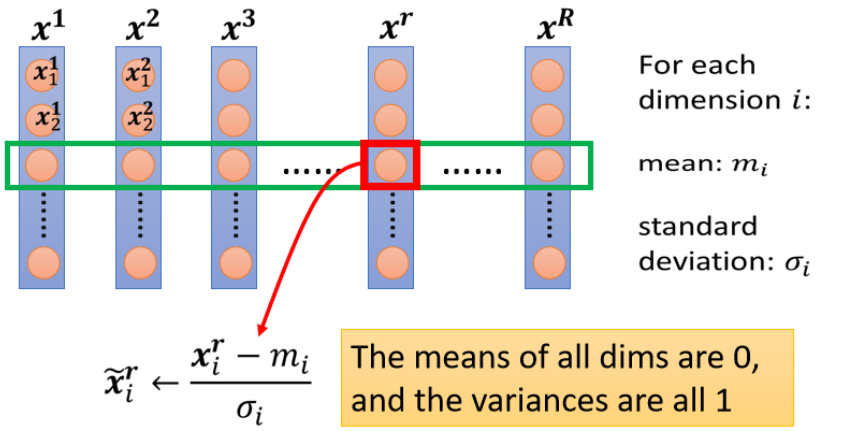
那接下来我们就可以做一种 normalization,那这种 normalization 其实叫做==**标準化**==,其实叫 ==standardization==,不过我们这边呢,就等一下都统称 normalization 就好了

$$ \tilde{x}^r_i ← \frac{x^r_i-m_i}{\sigma_i}

$$ 我们就是把这边的某一个数值x,减掉这一个 dimension 算出来的 mean,再除掉这个 dimension,算出来的 standard deviation,得到新的数值叫做 $\tilde{x}$

然后得到新的数值以后,**再把新的数值把它塞回去**,以下都用这个 tilde来代表有被 normalize 后的数值

那做完 normalize 以后有什麼好处呢?



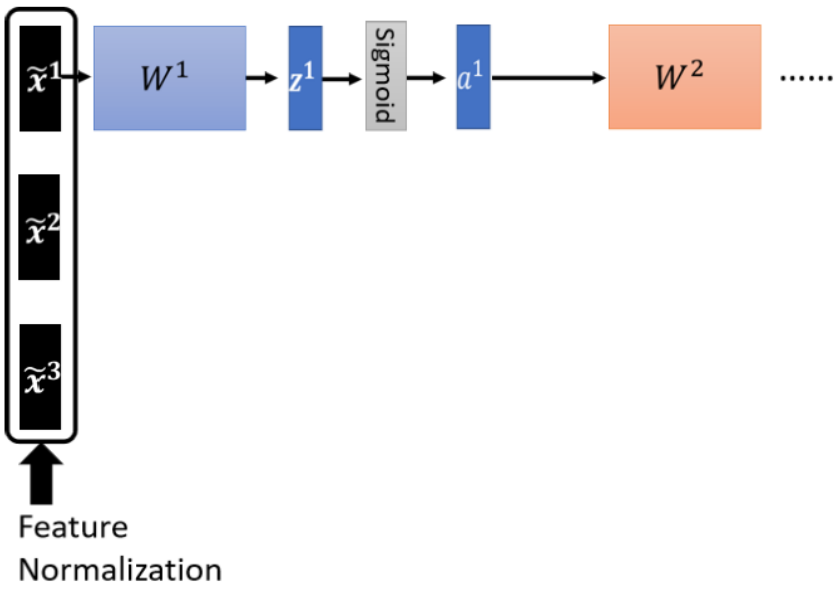
In general, feature normalization makes gradient descent converge faster.

- 做完 normalize 以后啊,这个 dimension 上面的数值就会平均是 0,然后它的 variance就会是 1,所以**这一排数值的分布就都会在 0 上下**
- 对每一个 dimension都做一样的 normalization,就会发现所有 feature 不同 dimension 的数值都在 0 上下,那你可能就可以**製造一个,比较好的 error surface**

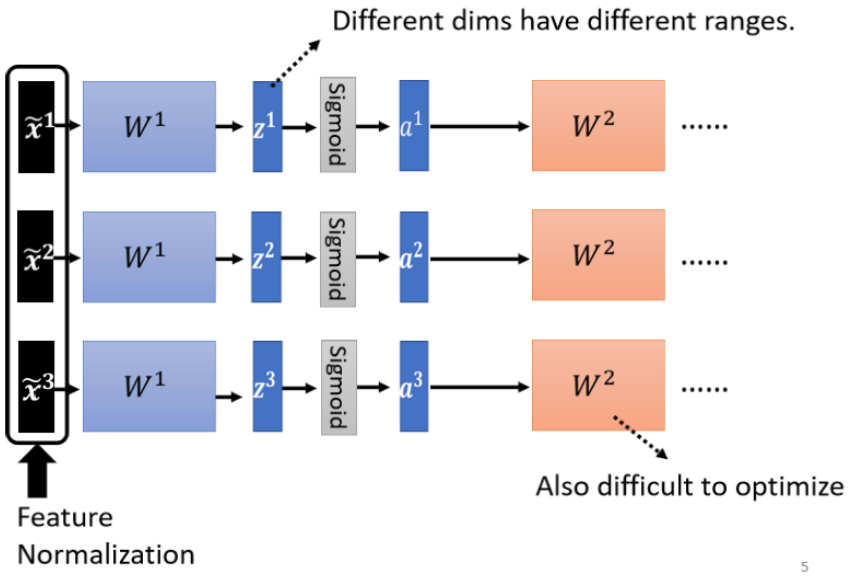
所以像这样子 Feature Normalization 的方式,往往对你的 training 有帮助,它可以让你在做 gradient descent 的时候,这个 gradient descent,**它的 Loss 收敛更快一点,可以让你的 gradient descent,它的训练更顺利一点**,这个是 Feature Normalization

Considering Deep Learning

$\tilde{x}$ 代表 normalize 的 feature,把它丢到 deep network 裡面,去做接下来的计算和训练,所以把 x_1 tilde 通过第一个 layer 得到 z^1,那你有可能通过 activation function,不管是选 Sigmoid 或者 ReLU 都可以,然后再得到 a^1,然后再通过下一层等等,那就看你有几层 network 你就做多少的运算



所以每一个 x 都做类似的事情,但是如果我们进一步来想的话,对 w_2 来说



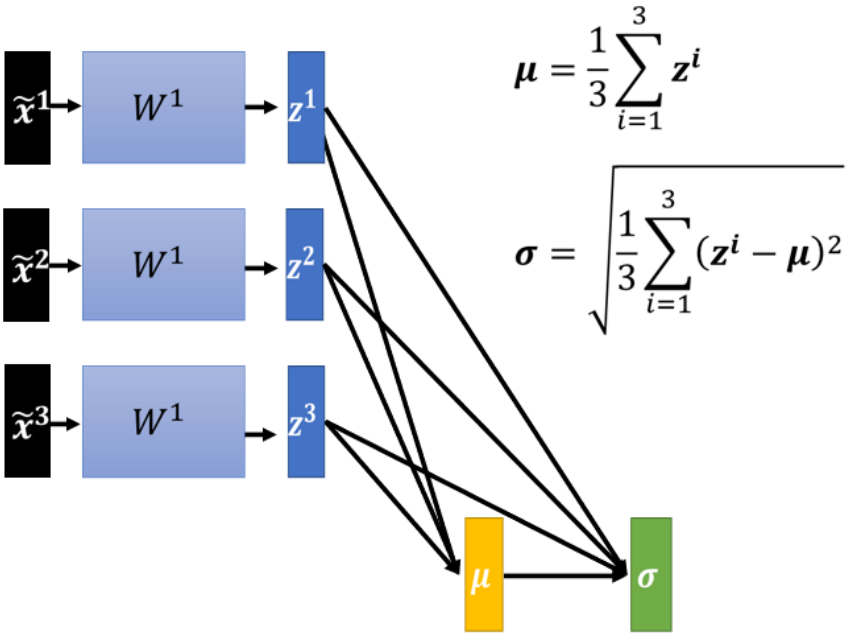
这边的 a^1 a^3 这边的 z^1 z^3,其实也是另外一种 input,如果这边 $\tilde{x}$,虽然它已经做 normalize 了,但是通过 $ w_1 $ 以后它就**没有做 normalize**,如果 $\tilde{x}$ 通过 $ w_1 $ 得到是 z^1,而 z^1 不同的 dimension 间,它的数值的分布仍然有很大的差异的话,那我们要 train w_2 第二层的参数,会不会也有困难呢

对 w_2 来说,这边的 a 或这边的 z 其实也是一种 feature,我们应该要对这些 feature 也做 normalization

那如果你选择的是 Sigmoid,那可能比较推荐对 z 做 Feature Normalization,因為 Sigmoid 是一个 s 的形状,那它在 0 附近斜率比较大,所以如果你对 z 做 Feature Normalization,把所有的值都挪到 0 附近,那你到时候算 gradient 的时候,算出来的值会比较大

那不过因為你不见得是用 sigmoid ,所以你也不一定要把 Feature Normalization放在 z 这个地方,如果是选别的,也许你选a也会有好的结果,也说不定，**Ingeneral 而言,这个 normalization,要放在 activation function 之前,或之后都是可以的,在实作上,可能没有太大的差别**,好 那我们这边呢,就是对 z 呢,做一下 Feature Normalization

那怎麼对 z 做 Feature Normalization 呢

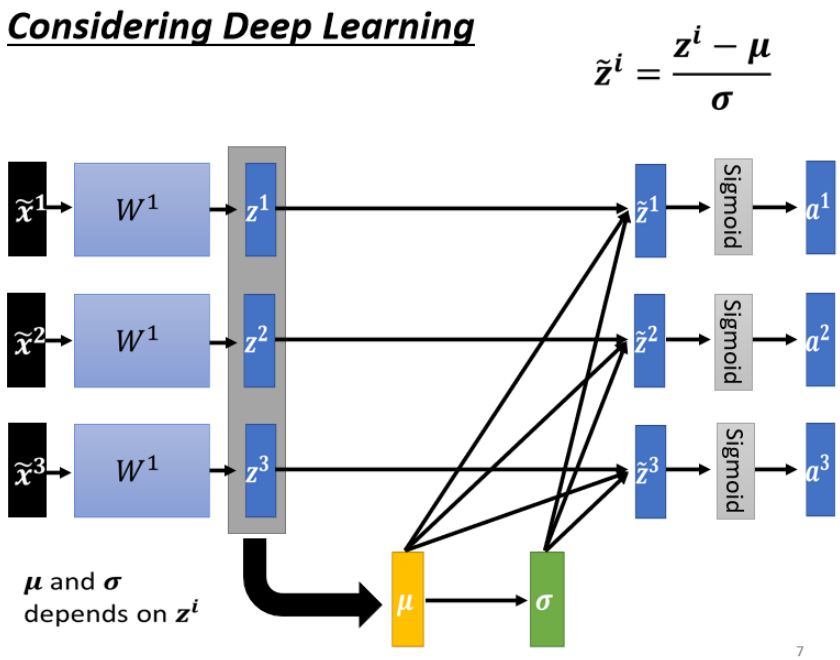


那你就把 z,想成是另外一种 feature ,我们这边有 z^1 z^2 z^3,我们就把 z^1 z^2 z^3 拿出来

- 算一下它的 mean，这边的 $μ$ 是一个 vector,我们就把 z^1 z^2 z^3,这三个 vector 呢,把它平均起来,得到 $μ$ 这个 vector
- 那我们也算一个 standard deviation,这个 standard deviation 呢,这边这个成 σ,它也代表了一个 vector,那这个 vector 怎麼算出来呢,你就把 z^i减掉 $μ$,然后取平方,这边的平方,这个 notation 有点 abuse 啊,这边的平方就是指,对每一个 element 都去做平方,然后再开根号,这边开根号指的是对每一个 element,向量裡面的每一个 element,都去做开根号,得到 σ,反正你知道我的意思就好

把这三个 vector,裡面的每一个 dimension,都去把它的 $μ$ 算出来,把它的 σ 算出来,好 我这边呢,就不把那些箭头呢 画出来了,从 z^1 z^2 z^3,算出 $μ$,算出 σ

接下来就把这边的每一个 z ,都去减掉 $μ$ 除以 σ,你把 z^i减掉 $μ$,除以 σ,就得到 z^i的 tilde

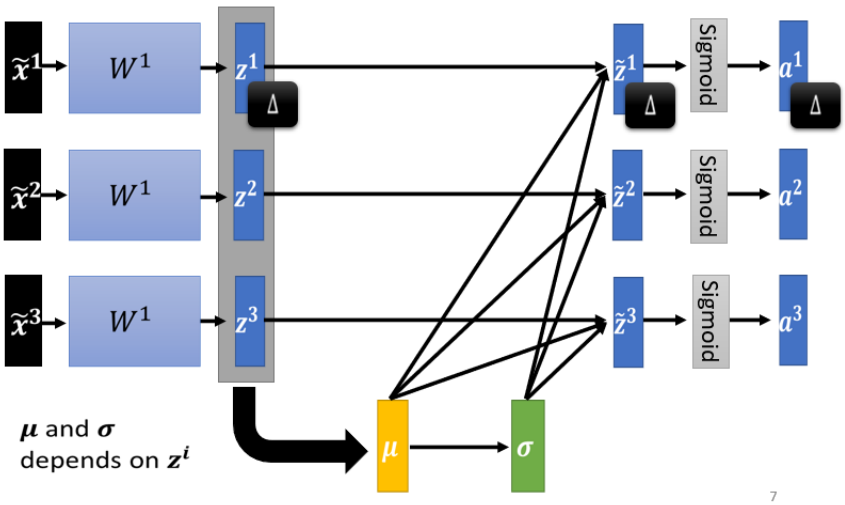


那这边的 $μ$ 跟 σ,它都是**向量**,所以这边这个除的意思是**element wise 的相除**,就是 z^i减 $μ$,它是一个向量,所以分子的地方是一个向量,分母的地方也是一个向量,把这个两个向量,它们对应的 element 的值相除,是我这边这个除号的意思,这边得到 Z 的 tilde

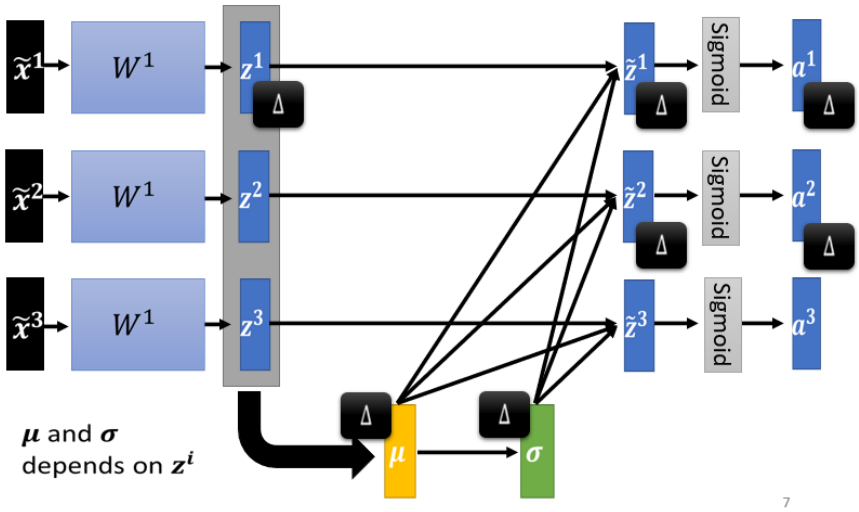
所以我们就是把 z^1 减 $μ$ 除以 σ,得到 z^1 tilde,同理 z^2 减 $μ$ 除以 σ,得到 z^2 tilde,z^3 减 $μ$ 除以 σ,得到 z^3 tilde,那就把这个 z^1 z^2 z^3,做 Feature Normalization,变成 z^1 tilde,z^2 tilde 跟 z^3 的 tilde

接下来就看你爱做什麼 就做什麼啦,通过 activation function,得到其他 vector,然后再通过,再去通过其他 layer 等等,这样就可以了,这样你就等於对 z^1 z^2 z^3,做了 Feature Normalization,变成 $\tilde{z}^1$ $\tilde{z}^2$ $\tilde{z}^3$

在这边有一件有趣的事情,这边的 $μ$ 跟 σ,它们其实都是根据 z^1 z^2 z^3 算出来的

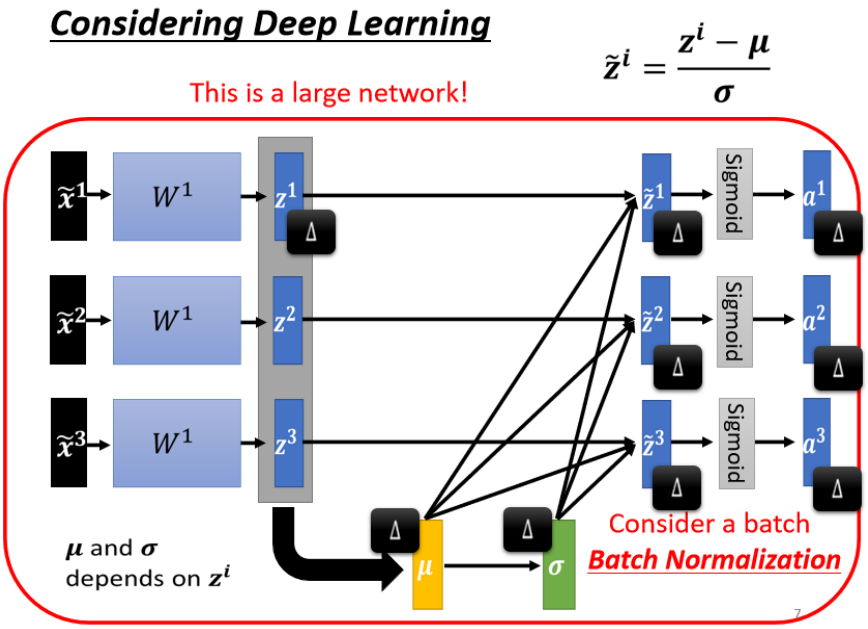


所以这边 z^1 啊,它本来,如果我们没有做 Feature Normalization 的时候,你改变了 z^1 的值,你会改变这边 a 的值,但是现在啊,当你改变 z^1 的值的时候,$μ$ 跟 σ 也会跟著改变,$μ$ 跟 σ 改变以后,z^2 的值 a^2 的值,z^3 的值 a^3 的值,也会跟著改变



所以**之前**,我们每一个 $\tilde{x}_1$ $\tilde{x}_2$ $\tilde{x}_3$,它是**独立分开处理的**,但是我们在做 **Feature Normalization 以后**,这三个 example,它们变得**彼此关联**了

我们这边 z^1 只要有改变,接下来 z^2 a^2 z^3 a^3,也都会跟著改变,所以这边啊,其实你要把,当你有做 Feature Normalization 的时候,你要把这一整个 process,就是有收集一堆 feature,把这堆 feature 算出 $μ$ 跟 σ 这件事情,当做是 network 的一部分



也就是说,你现在有一个比较大的 network

- 你之前的 network,都只吃一个 input,得到一个 output
- 现在你有一个比较大的 network,这个大的 network,它是吃一堆 input,用这堆 input 在这个 network 裡面,要算出 $μ$ 跟 σ,然后接下来產生一堆 output

那这个地方比较抽象,只可会意 不可言传这样子

那这边就会有一个问题了,因為你的训练资料裡面的 data 非常多,现在一个 data set,benchmark corpus 都上百万笔资料， GPU 的 memory,根本没有办法,把它整个 data set 的 data 都 load 进去。

在实作的时候,你不会让这一个 network 考虑整个 training data 裡面的所有 example,你**只会考虑一个 batch 裡面的 example**,举例来说,你 batch 设 64,那你这个巨大的 network,就是把 64 笔 data 读进去,算这 64 笔 data 的 $μ$,算这 64 笔 data 的 σ,对这 64 笔 data 都去做 normalization

因為我们在实作的时候,我们只对一个 batch 裡面的 data,做 normalization,所以这招叫做 ==Batch Normalization==

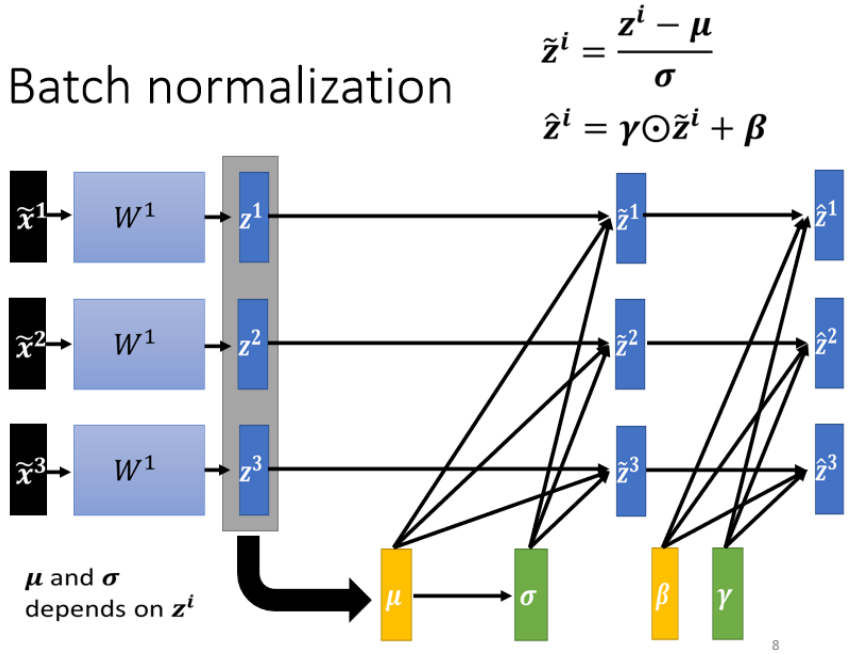
那这个 Batch Normalization,显然有一个问题 就是,**你一定要有一个够大的 batch,你才算得出 $μ$ 跟 σ**,假设你今天,你 batch size 设 1,那你就没有什麼 $μ$ 或 σ 可以算

所以这个 Batch Normalization,是适用於 batch size 比较大的时候,因為 batch size 如果比较大,**也许这个 batch size 裡面的 data,就足以表示,整个 corpus 的分布**,那这个时候你就可以,把这个本来要对整个 corpus,做 Feature Normalization 这件事情,改成只在一个 batch,做 Feature Normalization,作為 approximation,

在做 Batch Normalization 的时候,往往还会有这样的设计你算出这个 $\tilde{z}$ 以后

- 接下来你会把这个 $\tilde{z}$,再乘上另外一个向量叫做 $γ$,这个 $γ$ 也是一个向量,所以你就是把 $\tilde{z}$ 跟 $γ$ 做 element wise 的相乘,把 z 这个向量裡面的 element,跟 $γ$ 这个向量裡面的 element,两两做相乘
- 再加上 $β$ 这个向量,得到 $\hat{z}$

而 $β$ 跟 $γ$,你要把它想成是 network 的参数,它是另外再被learn出来的,



那為什麼要加上 $β$ 跟 $γ$ 呢

有人可能会觉得说,如果我们做 normalization 以后,那这边的 $\tilde{z}$,它的平均就一定是 0,那也许,今天如果平均是 0 的话,就是给那 network 一些限制,那**也许这个限制会带来什麼负面的影响**,所以我们把 $β$ 跟 $γ$ 加回去

然后让 network 呢,现在它的 hidden layer 的 output平均不是 0 的话,他就自己去 learn这个 $β$ 跟 $γ$,来调整一下输出的分布,来调整这个 $\hat{z}$ 的分布

但讲到这边又会有人问说,刚才不是说做 Batch Normalization 就是,為了要让每一个不同的 dimension,它的 range 都是一样吗,现在如果加去乘上 $γ$,再加上 $β$,把 $γ$ 跟 $β$ 加进去,这样不会不同 dimension 的分布,它的 range 又都不一样了吗

有可能,但是你实际上在训练的时候,这个 $γ$ 跟 $β$ 的初始值啊

- 你会把这个 $γ$ 的初始值 就都设為 1,所以 $γ$ 是一个裡面的值,一开始其实是一个裡面的值,全部都是 1 的向量
- 那 $β$ 是一个裡面的值,全部都是 0 的向量,所以 $γ$ 是一个 one vector,都是 1 的向量,$β$ 是一个 zero vector,裡面的值都是 0 的向量

所以让你的 network 在一开始训练的时候,每一个 dimension 的分布,是比较接近的,也许训练到后来,你已经训练够长的一段时间,已经找到一个比较好的 error surface,走到一个比较好的地方以后,那再把 $γ$ 跟 $β$ 慢慢地加进去,好 所以加 Batch Normalization,往往对你的训练是有帮助的

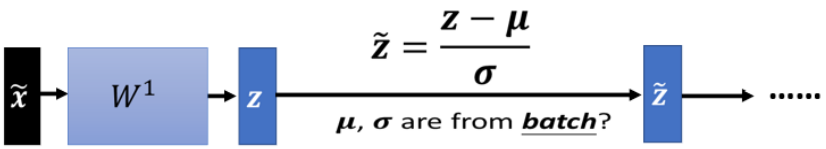
Testing

以上说的都是 training 的部分,testing ,有时候又叫 ==inference== ,所以有人在文件上看到有人说,做个 inference,inference 指的就是 testing

这个 Batch Normalization 在 inference,或是 testing 的时候,会有什麼样的问题呢

在 testing 的时候,如果 当然如果今天你是在做作业,我们一次会把所有的 testing 的资料给你,所以你确实也可以在 testing 的资料上面,製造一个一个 batch

但是假设你真的有系统上线,你是一个真正的线上的 application,你可以说,我今天一定要等 30,比如说你的 batch size 设 64,我一定要等 64 笔资料都进来,我才一次做运算吗,这显然是不行的

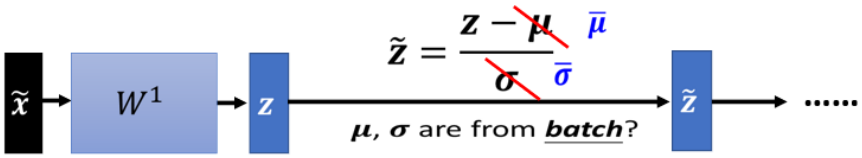


但是在做 Batch Normalization 的时候,一个 $\tilde{x}$,一个 normalization 过的 feature 进来,然后你有一个 z,你的 z 呢,要减掉 $μ$ 跟除 σ,那这个 $μ$ 跟 σ,是**用一个 batch 的资料算出来的**

但如果今天在 testing 的时候,根本就没有 batch,那我们要怎麼算这个 $μ$,跟怎麼算这个 σ 呢

所以真正的,这个实作上的解法是这个样子的,如果你看那个 PyTorch 的话呢,Batch Normalization 在 testing 的时候,你并不需要做什麼特别的处理,PyTorch 帮你处理好了

在 training 的时候,如果你有在做 Batch Normalization 的话,在 training 的时候,你每一个 batch 计算出来的 $μ$ 跟 σ,他都会拿出来算 ==moving average==



We do not always have ***batch*** at testing stage.

Computing the moving average of μ and σ of the batches during training.

$$\mu^1 \quad \mu^2 \quad \mu^3 \quad ..... \quad \mu^t$$

$$\bar{\mu} \leftarrow p\bar{\mu} + (1-p)\mu^t$$

你每一次取一个 batch 出来的时候,你就会算一个 $μ^1$,取第二个 batch 出来的时候,你就算个 $μ^2$,一直到取第 t 个 batch 出来的时候,你就算一个 $μ^t$

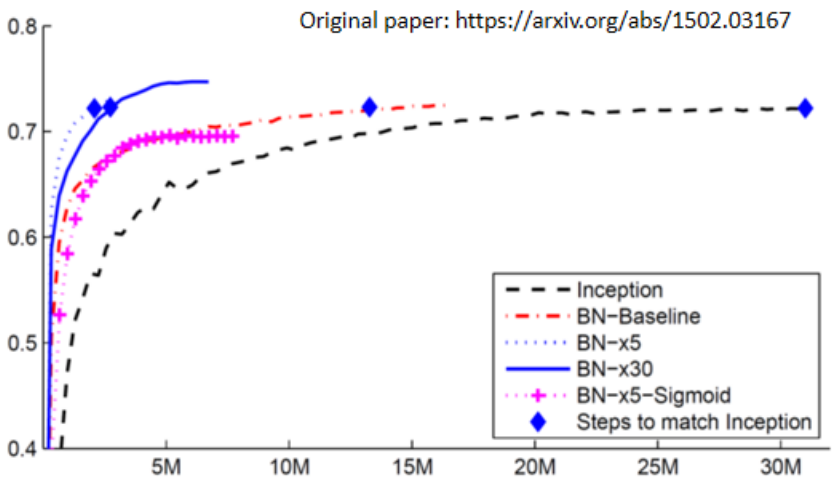
接下来你会算一个 moving average,你会把你现在算出来的 $μ$ 的一个平均值,叫做 $μ$ bar,乘上某一个 factor,那这也是一个常数,这个这也是一个 constant,这也是一个那个 hyper parameter,也是需要调的那种啦

在 PyTorch 裡面,我没记错 他就设 0.1,我记得他 P 就设 0.1,好,然后加上 1 减 P,乘上 $μ^t$,然后来更新你的 $μ$ 的平均值,然后最后在 testing 的时候,你就不用算 batch 裡面的 $μ$ 跟 σ 了

因為 testing 的时候,在真正 application 上,也没有 batch 这个东西,你就直接拿 $\barμ$ 跟 $\bar\sigma$,也就是 $μ$ 跟 σ 在训练的时候,得到的 moving average,$\barμ$ 跟 $\bar\sigma$,来取代这边的 $μ$ 跟 σ,这个就是 Batch Normalization,在 testing 的时候的运作方式

Comparison

好 那这个是从 Batch Normalization,原始的文件上面截出来的一个实验结果,那在原始的文件上还讲了很多其他的东西,举例来说,我们今天还没有讲的是,Batch Normalization 用在 CNN 上,要怎麼用呢,那你自己去读一下原始的文献,裡面会告诉你说,Batch Normalization 如果用在 CNN 上,应该要长什麼样子



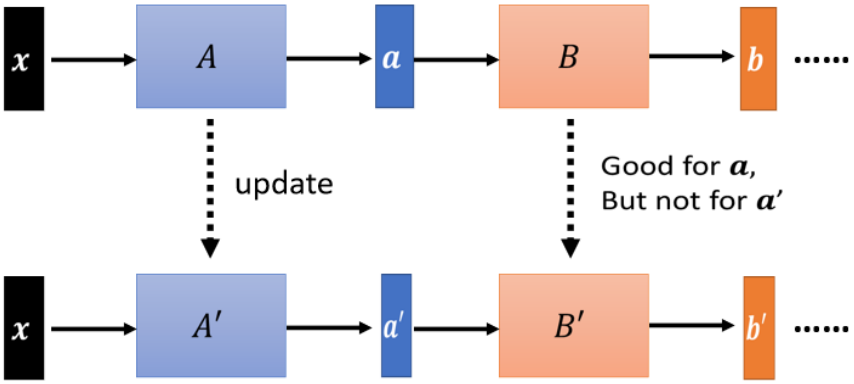
这个是原始文献上面截出来的一个数据

- 横轴呢,代表的是训练的过程，纵轴代表的是 validation set 上面的 accuracy
- 那这个**黑色**的虚线是**没有做 Batch Normalization** 的结果,它用的是 inception 的 network,就是某一种 network 架构啦,也是以 CNN 为基础的 network 架构
- 然后如果有做 Batch Normalization,你会得到**红色**的这一条虚线,那你会发现说,红色这一条虚线,它**训练的速度,显然比黑色的虚线还要快很多**,虽然最后收敛的结果啊,就你只要给它足够的训练的时间,可能都跑到差不多的 accuracy,但是**红色这一条虚线,可以在比较短的时间内,就跑到一样的 accuracy**,那这边这个蓝色的菱形,代表说这几个点的那个 accuracy 是一样的
- **粉红色**的线是 sigmoid function,就 sigmoid function 一般的认知,我们虽然还没有讨论这件事啦,但一般都会选择 ReLu,而不是用 sigmoid function,因为 sigmoid function,它的 training 是比较困难的,但是这边想要强调的点是说,**就算是 sigmoid 比较难搞的,加 Batch Normalization,还是 train 的起来**,那这边没有 sigmoid,没有做 Batch Normalization 的结果,因為在这个实验上,作者有说,sigmoid 不加 Batch Normalization,根本连 train 都 train 不起来
- **蓝色**的实线跟这个蓝色的虚线呢,是把 learning rate 设比较大一点,乘 5,就是 learning rate 变原来的 5 倍,然后乘 30,就是 learning rate 变原来的 30 倍,那因為如果你**做 Batch Normalization 的话,那你的 error surface 呢,会比较平滑比较容易训练**,所以你可以把你的比较不崎嶇,所以你就可以把你的 learning rate 呢,设大一点,那这边有个不好解释的奇怪的地方,就是不知道為什麼,learning rate 设 30 倍的时候,是比 5 倍差啦,那作者也没有解释啦,你也知道做 deep learning 就是,有时候会產生这种怪怪的,不知道怎麼解释的现象就是了,不过作者就是照实,把他做出来的实验结果,呈现在这个图上面

Internal Covariate Shift?

好 接下来的问题就是,Batch Normalization,它為什麼会有帮助呢,在原始的 Batch Normalization,那篇 paper 裡面,他提出来一个概念,叫做 ==internal covariate shift==,==covariate shift==(训练集和预测集样本分布不一致的问题就叫做“*covariate shift*”现象) 这个词汇是原来就有的,internal covariate shift,我认為是,Batch Normalization 的作者自己发明的

他认為说今天在 train network 的时候,会有以下这个问题,这个问题是这样



Batch normalization make a and a' have similar statistics.

network 有很多层

- x 通过第一层以后 得到 a
- a 通过第二层以后 得到 b
- 计算出 gradient 以后,把 A update 成 A′,把 B 这一层的参数 update 成 B′

但是作者认为说,我们在计算 B,update 到 B′ 的 gradient 的时候,这个时候前一层的参数是 A 啊,或者是前一层的 output 是小 a 啊

那当前一层从 A 变成 A′ 的时候,它的 output 就从小 a 变成小 a′ 啊

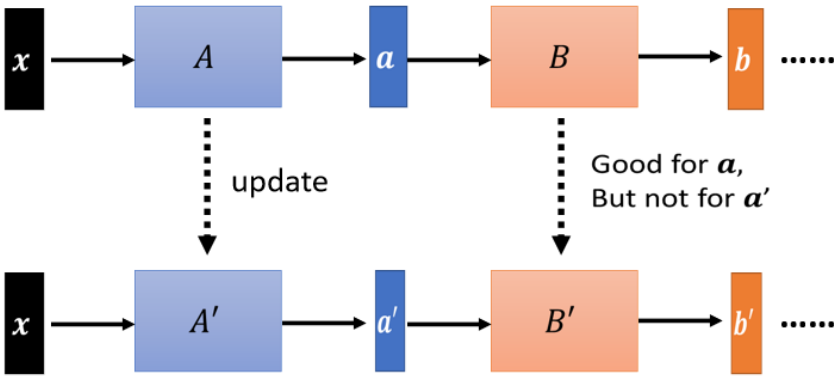
但是我们计算这个 gradient 的时候,我们是根据这个 a 算出来的啊,所以这个 update 的方向,也许它**适合用在 a 上,但不适合用在 a′ 上面**

那如果说 Batch Normalization 的话,我们会让,因為我们每次都有做 normalization,我们就会让 a 跟 a′ 呢,它的分布比较接近,也许这样就会对训练呢,有帮助

Internal Covariate Shift?

How Does Batch Normalization Help Optimization?

https://arxiv.org/abs/1805.11604



Batch normalization make a and a' have similar statistics.

Experimental results do not support the above idea.

但是有一篇 paper 叫做,How Does Batch Normalization,Help Optimization,然后他就**打脸了internal covariate shift 的这一个观点**

在这篇 paper 裡面,他从各式各样的面向来告诉你说,**internal covariate shift,首先它不一定是 training network 的时候的一个问题,然后 Batch Normalization,它会比较好,可能不见得是因為,它解决了 internal covariate shift**

那在这篇 paper 裡面呢,他做了很多很多的实验,比如说他比较了训练的时候,这个 a 的分布的变化 发现,**不管有没有做 Batch Normalization,它的变化都不大**

然后他又说,就算是变化很大,对 training 也没有太大的伤害,然后他又说,不管你是根据 a 算出来的 gradient,还是根据 a′ 算出来的 gradient,方向居然都差不多

所以他告诉你说,internal covariate shift,可能不是 training network 的时候,最主要的问题,它可能也不是,Batch Normalization 会好的一个的关键,那有关更多的实验,你就自己参见这篇文章,

為什麼 Batch Normalization 会比较好呢,那在这篇 How Does Batch Normalization,Help Optimization 这篇论文裡面,他从实验上,也从理论上,至少**支持了 Batch Normalization,可以改变 error surface,让 error surface 比较不崎嶇这个观点**

How Does Batch Normalization Help Optimization?

https://arxiv.org/abs/1805.11604

Experimental results (and theoretically analysis) support batch normalization change the landscape of error surface.

and 12 of Appendix B.) This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous. Therefore, it might be valuable to perform a principled exploration of the design space of normalization schemes as it can lead to better performance.

serendipitous (偶然的)

penicillin



所以这个观点是有理论的支持,也有实验的佐证的,那在这篇文章裡面呢,作者还讲了一个非常有趣的话,他说他觉得啊,这个 Batch Normalization 的 positive impact

因為他说,如果我们要让 network,这个 error surface 变得比较不崎嶇,**其实不见得要做 Batch Normalization,感觉有很多其他的方法,都可以让 error surface 变得不崎嶇**,那他就试了一些其他的方法,发现说,跟 Batch Normalization performance 也差不多,甚至还稍微好一点,所以他就讲了下面这句感嘆

他觉得说,这个,positive impact of batchnorm on training,可能是 somewhat,**serendipitous**,什麼是 serendipitous 呢,这个字眼可能可以翻译成偶然的,但偶然并没有完全表达这个词汇的意思,这个词汇的意思是说,你发现了一个什麼意料之外的东西

举例来说,盘尼西林就是,意料之外的发现,大家知道盘尼西林的由来就是,有一个人叫做弗莱明,然后他本来想要那个,培养一些葡萄球菌,然后但是因為他实验没有做好,他的那个葡萄球菌被感染了,有一些霉菌掉到他的培养皿裡面,然后发现那些培养皿,那些霉菌呢,会杀死葡萄球菌,所以他就发明了,发现了盘尼西林,所以这是一种偶然的发现

那这篇文章的作者也觉得,Batch Normalization 也像是盘尼西林一样,是一种偶然的发现,但无论如何,它是一个有用的方法

To learn more ……

那其实 Batch Normalization,不是唯一的 normalization,normalization 的方法有一把啦,那这边就是列了几个比较知名的,

Batch Renormalization https://arxiv.org/abs/1702.03275 Layer Normalization https://arxiv.org/abs/1607.06450 Instance Normalization https://arxiv.org/abs/1607.08022 Group Normalization https://arxiv.org/abs/1803.08494 Weight Normalization https://arxiv.org/abs/1602.07868 Spectrum Normalization https://arxiv.org/abs/1705.10941

Classification

To learn more

接下来讲有关分类怎麼做这件事情,这边讲的是一个短的版本,因為时间有限的关係,如果你想要看长的版本的话,可以看一下过去上课的录影

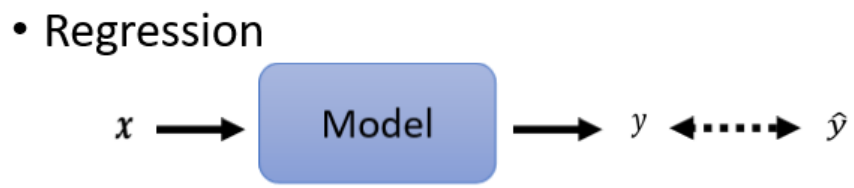
https://youtu.be/fZAZUYEeIMg
(in Mandarin)

https://youtu.be/hSXFuypLukA
(in Mandarin)

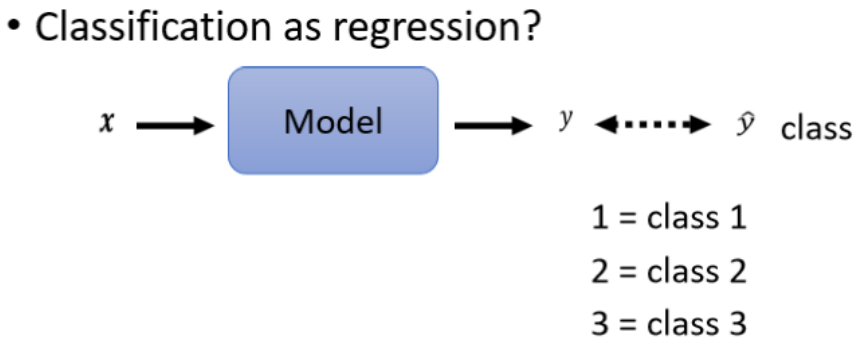
过去可能是花两个小时,到三个小时的时间才讲完,分类这件事情,我们这边用一个最快的方法,直接跟你讲分类是怎麼做的

Classification as Regression?

分类是怎麼做的呢 我们已经讲了,Regression就是输入一个向量,然后输出一个数值,我们希望输出的数值跟某一个label,也就是我们要学习的目标,越接近越好，这门课里面，**如果是正确的答案就有加Hat,Model的输出没有加Hat**



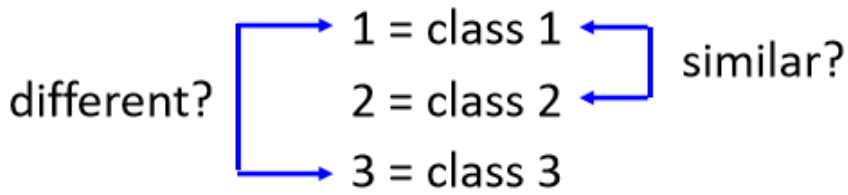
有一个可能,假设你会用Regression的话,我们其实可以**把Classification,当作是Regression来看**



这个方法不一定是个好方法,这是一个比较奇妙的方法,输入一个东西以后,我们的输出仍然是一个scaler,它叫做y 然后这一个y,我们要让它跟正确答案,那个Class越接近越好,但是**y是一个数字**,我们怎麼让它跟Class越接近越好呢,我们**必须把Class也变成数字**

举例来说 Class1就是编号1,Class2就是编号2,Class3就是编号3,接下来呢 我们要做的事情,就是希望y可以跟Class的编号,越接近越好

但是这会是一个好方法吗,如果你仔细想想的话,这个方法也许在某些状况下,是会有瑕疵的

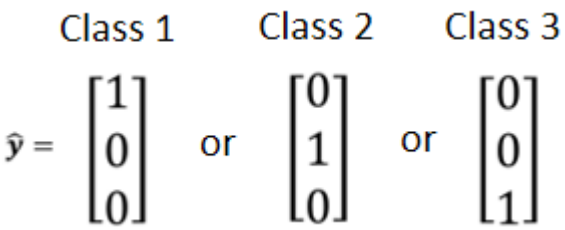


因為如果你假设说Class one就是编号1,Class two就是编号2,Class3就是编号3,意味著说你觉得**Class1跟Class2是比较像**,然后**Class1跟Class3 它是比较不像**,像这样子的表示Class的方式,有时候可行 有时候不可行

- 假设你的Class one two three**真的有某种关係**举例来说,你想要根据一个人的身高跟体重,然后预测他是几年级的小学生,一年级 二年级 还是三年级,那可能一年级真的跟二年级比较接近,一年级真的跟三年级比较没有关係
- 但是假设你的三个Class本身,**并没有什麼特定的关係**的话,你说Class one是1,Class two是2 Class two是3,那就很奇怪了,因為你这样是预设说,一二有比较近的关係,一三有比较远的关係,所以怎麼办呢

Class as one-hot vector

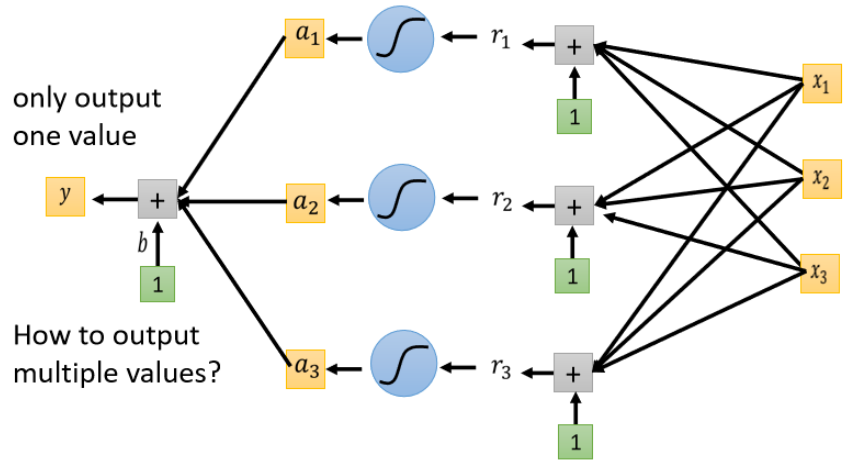
当你在做分类的问题的时候,比较常见的做法是把你的Class,用 One-hot vector来表示



如果有三个Class,我们的 label 这个ŷ,就是一个三维的向量,然后呢 如果是Class1就是$\begin{bmatrix}{1} \\ {0}\\ {0}\end{bmatrix}$,如果是Class2就是$\begin{bmatrix} {0}\\ {1}\\ {0}\end{bmatrix}$,如果是Class3就是$\begin{bmatrix} {0}\\ {0}\\ {1}\end{bmatrix}$,所以每一个Class,你都用一个One-hot vector来表示

而且你用One-hot vector来表示的话,就没有说Class1跟Class2比较接近,Class1跟Class3比较远这样子的问题,如果你把这个One-hot vector,**用算距离的话,Class之间两两它们的距离都是一样**

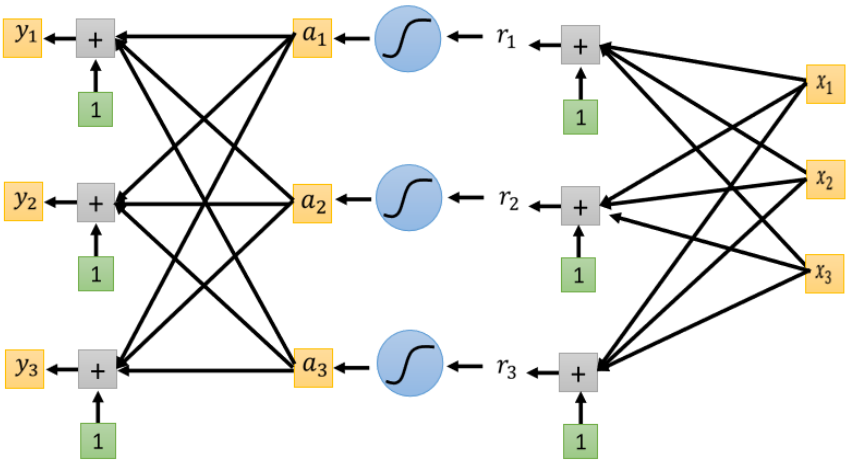
如果我们今天的目标y hat是一个向量 比如说,ŷ是有三个element的向量，那我们的network,也应该要Output的维度也是三个数字才行



到目前為止我们讲的network,其实都只Output一个数值,因為我们**过去做的都是Regression的问题,所以只Output一个数字**

其实**从一个数值改到三个数值,它是没有什麼不同的**

你可以Output一个数值,你就可以Output三个数值,所以把本来Output一个数值的方法,重复三次

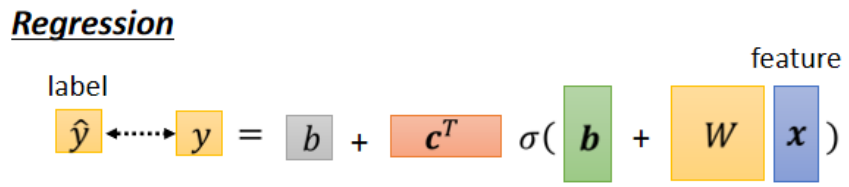


- 把a_1 a_2 a_3,乘上三个不同的Weight 加上bias,得到y_1
- 再把a_1 a_2 a_3乘上另外三个Weight,再加上另外一个bias得到y_2
- 再把a_1 a_2 a_3再乘上另外一组Weight,再加上另外一个bias得到y_3

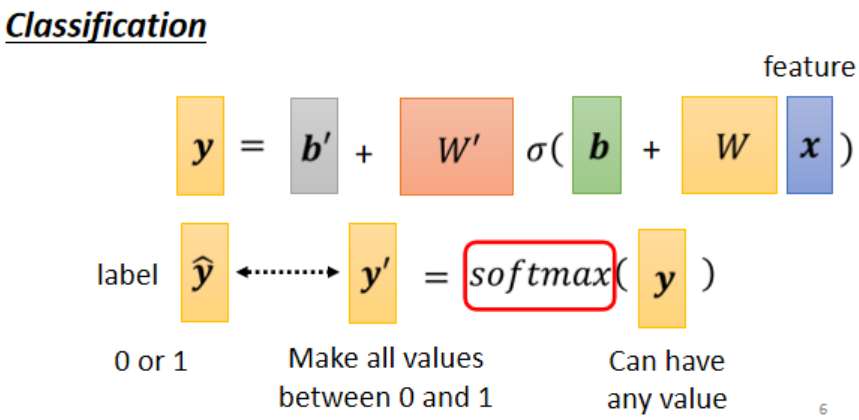
你就可以產生三组数字,所以你就可以Input一个feature的Vector,然后產生y_1 y_2 y_3,然后希望y_1 y_2 y_3,跟我们的目标越接近越好,

Classification with softmax

好 那所以我们现在,知道了Regression是怎麼做的,Input x Output y 要跟 label ŷ,越接近越好



如果是Classification,input x可能乘上一个W,再加上b 再通过activation function,再乘上W'再加上b' 得到y,我们现在的**y它不是一个数值,它是一个向量**



但是在做Classification的时候,我们往往会**把y再通过一个叫做Soft-max的function得到y'**,然后我们才去计算,y'跟y hat之间的距离

為什麼要加上Soft-max呢,一个比较简单的解释（如果是在过去的课程裡面,我们会先从generative的Model开始讲起,然后一路讲到Logistic Regression）

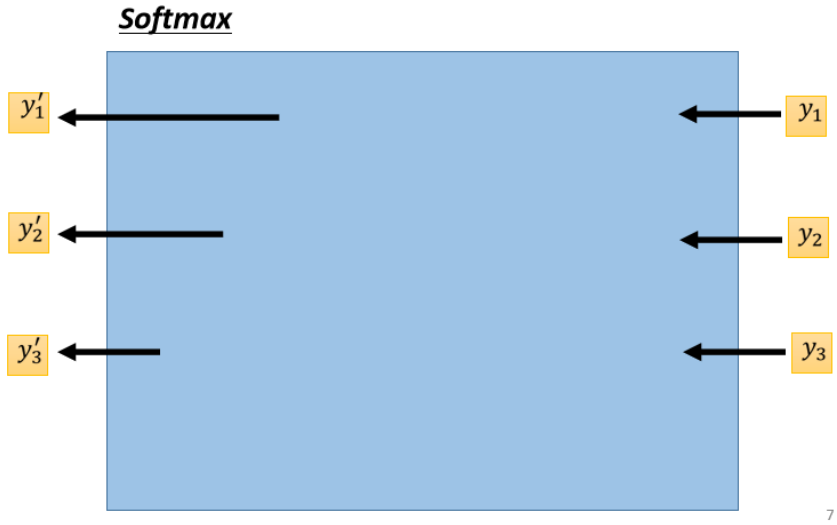
这边有一个骗小孩的解释就是,这个ŷ 它裡面的值,都是0跟1,它是One-hot vector,所以裡面的值只有0跟1，但是y裡面有任何值

既然我们的目标只有0跟1,但是==y有**任何值**,我们就先把它Normalize到**0到1之间**,这样才好跟 label 的计算相似度==,这是一个比较简单的讲法

如果你真的想要知道,為什麼要用Soft-max的话,你可以参考过去的上课录影,如果你不想知道的话,你就记得这个Soft-max要做的事情,就是把本来y裡面可以放任何值,改成挪到0到1之间

Softmax

这个是Soft-max的block,输入y_1 y_2 y_3,它会產生y_1' y_2' y_3'

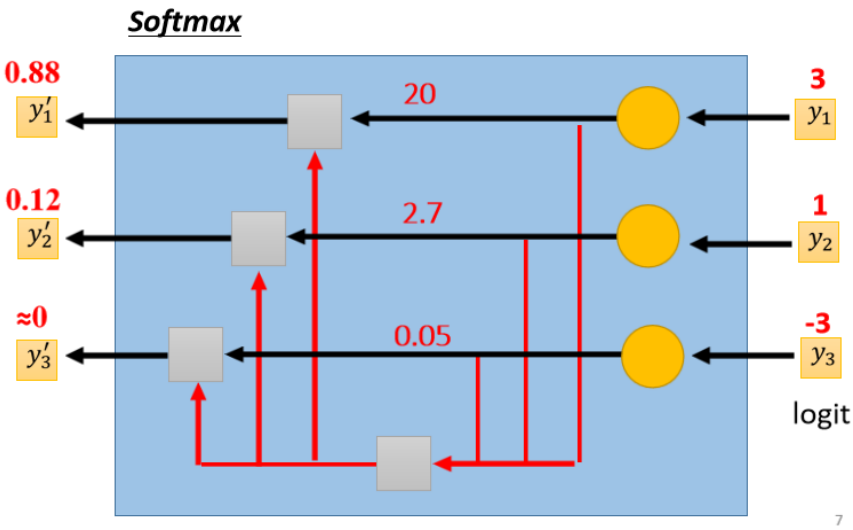


它裡面运作的模式是这个样子的

$$ y_i'=\frac{exp(y_i)}{\sum_j exp(y_i)}

$$ 我们会先把所有的y取一个exponential,就算是负数,取exponential以后也变成正的,然后你再对它做Normalize,除掉所有y的exponential值的和,然后你就得到y'

或者是用图示化的方法是这个样子



y_1取exp y_2取exp y_3取exp,把它全部加起来,得到一个Summation,接下来再把exp y_1'除掉Summation,exp y_2'除掉Summation,exp y_3'除掉Summation,就得到y_1' y_2' y_3'

有了这个式子以后,你就会发现

- y_1' y_2' y_3',它们都是介於**0到1之间**
- y_1' y_2' y_3',它们的**和是1**

如果举一个例子的话,本来 y_1等於3y_2等於1,y_3等於负3,取完exponential的时候呢,就变成exp3 就是20,exp1就是2.7,exp-3就是0.05,做完Normalization以后,这边就变成0.88 0.12 跟0

所以这个Soft-max它要做的事情,除了Normalized,让 y_1' y_2' y_3',**变成0到1之间**,还有**和為1**以外,它还有一个附带的效果是,==它会让大的值跟小的值的差距更大==

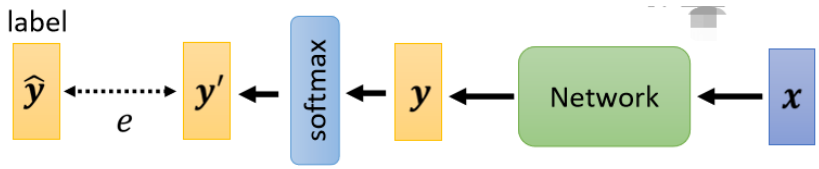
本来-3 然后通过exponential,再做Normalized以后,会变成趋近於0的值,然后这个Soft-max的输入,往往就叫它==logit==

这边考虑了3个class的状况,那**如果两个class会是怎麼样**

如果是两个class你当然可以直接套soft-max这个function没有问题,但是也许你更常听到的是,当有两个class的时候,我们就不套soft-max,我们直接取sigmoid

那==当两个class用sigmoid,跟soft-max两个class,你如果推一下的话,会发现说这两件事情是等价的==

Loss of Classification



我们把x,丢到一个Network裡面產生y以后,我们会通过soft-max得到y',再去计算y'跟ŷ之间的距离,这个写作e

计算y'跟ŷ之间的距离不只一种做法,举例来说,如果我喜欢的话,我要让这个距离是Mean Square Error

$$ e=\sum_i(\hat{y_i}-y_i')^2

$$ 就是把ŷ裡面每一个element拿出来,然后计算它们的平方和,当作我们的error,这样也是计算两个向量之间的距离,你也可以说,你也可以做到说当minimize,Mean Square Error的时候,我们可以让ŷ等於y'

但是有另外一个更常用的做法,叫做Cross-entropy

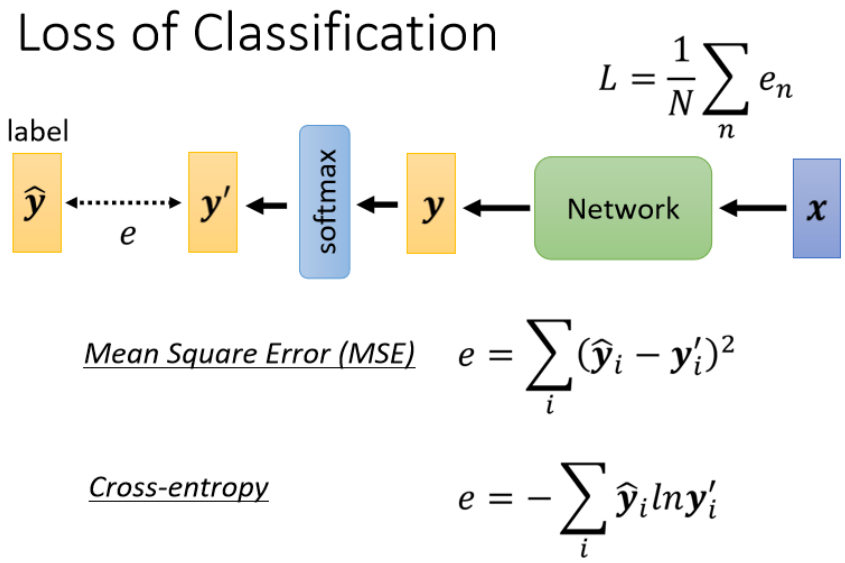
$$ e=-\sum_i\hat{y_i}\ln{y_i'}

$$ 这个Cross-entropy它的式子乍看之下,会让你觉得有点匪夷所思,怎麼是这个样子呢

- Cross-entropy是summation over所有的i
- 然后把ŷ的第i位拿出来,乘上y'的第i位取Natural log
- 然后再全部加起来

这个是Cross-entropy,那当ŷ跟y'一模一样的时候,你也可以Minimize Cross-entropy的值,此时,MSE会是最小的,Cross-entropy也会是最小的

但是為什麼会有Cross-entropy,这麼奇怪的式子出现呢?



那如果要讲得长一点的话,这整个故事我们可以把它讲成,**Make Minimize Cross-entropy其实就是maximize likelihood**,你很可能在很多地方,都听过likelihood这个词汇,详见过去上课影片

所以如果有一天有人问你说,如果我们今天在做分类问题的时候,maximize likelihood,跟Minimize Cross-entropy,有什麼关係的时候,不要回答说它们其实很像,但是其实又有很微妙的不同这样,不是这样,它们两个就是一模一样的东西,只是同一件事不同的讲法而已

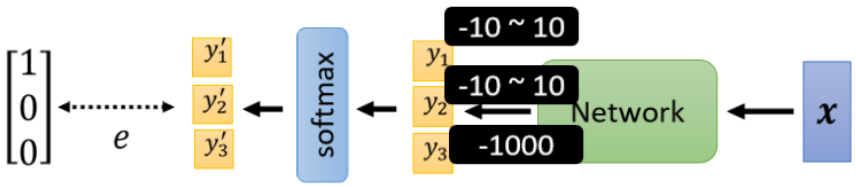
所以假设你可以接受说,我们在训练一个classifier的时候,应该要maximize likelihood就可以接受,应该要Minimizing Cross-entropy

在pytorch裡面,Cross-entropy跟Soft-max,他们是被绑在一起的,他们是一个Set,你只要Copy Cross-entropy,裡面就自动内建了Soft-max

那接下来**从optimization的角度,来说明相较於Mean Square Error,Cross-entropy是被更常用在分类上,**

那这个部分,你完全可以在数学上面做证明,但是我这边,是直接用举例的方式来跟你说明,如果你真的非常想看数学证明的话,我把连结放在这边
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Deep%20More%20(v2).ecm.mp4/index.html你可以一下过去上课的录影

如果你不想知道的话,那我们就是举一个例子来告诉你说,為什麼是Cross-entropy比较好



那现在我们要做一个**3个Class的分类**

Network先输出y_1 y_2 y_3,在通过soft-max以后,產生y_1' y_2'跟y_3'

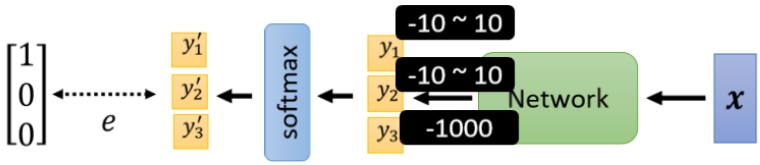
那接下来假设我们的正确答案就是100,我们要去计算100这个向量,跟y_1' y_2'跟y_3'他们之间的距离,那这个距离我们用e来表示,e可以是Mean square error,也可以是Cross-entropy,

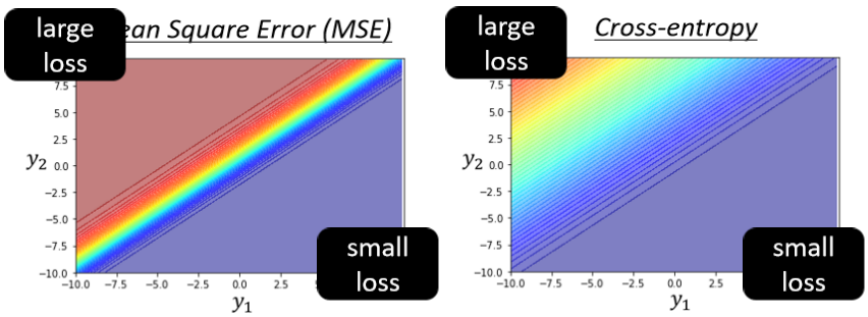
我们现在假设y_1的变化是从-10到10,y_2的变化也是从-10到10,y_3我们就固定设成-1000

因为y_3设很小,所以过soft-max以后y_3'就非常趋近於0,它跟正确答案非常接近,且它对我们的结果影响很少

总之我们**y_3设一个定值**,我们**只看y_1跟y_2有变化的时候,对我们的e对我们的Loss对我们loss有什麼样的影响**

那我们看一下 如果我们这个e,设定為Mean Square Error,跟Cross-entropy的时候,算出来的Error surface会有什麼样,不一样的地方.底下这两个图,就分别在我们e是Mean square error,跟Cross-entropy的时候,y_1 y_2的变化对loss的影响,对Error surface的影响,

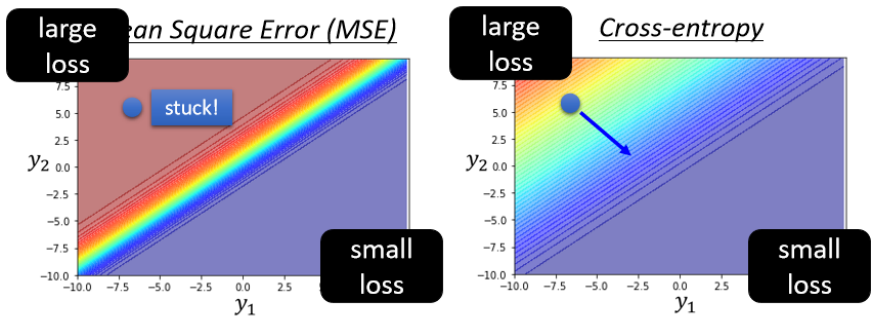




我们这边是用**红色代表Loss大,蓝色代表Loss小**

- 那如果今天**y_1很大 y_2很小**,就代表**y_1'会很接近1,y_2'会很接近0**,所以不管是对Mean Square Error,或是Cross-entropy而言,**y_1大 y_2小的时候 Loss都是小的**
- 如果**y_1小 y_2大**的话,这边**y_1'就是0 y_2'就是1**,所以这个时候Loss会比较大

所以这两个图都是**左上角Loss大,右下角Loss小**,所以我们就期待说,我们最后**在Training的时候,我们的参数可以走到右下角**的地方



Changing the loss function can change the difficulty of optimization.

那假设**我们开始的地方,都是左上角**

- 如果我们选择Cross-Entropy,左上角这个地方,它是有斜率的,所以你有办法透过gradient,一路往右下的地方走,
- 如果你选Mean square error的话,你就卡住了,Mean square error在这种Loss很大的地方,它是非常平坦的,它的gradient是非常小趋近於0的,如果你初始的时候在这个地方,离你的目标非常远,那它gradient又很小,你就会没有办法用gradient descent,顺利的走到右下角的地方去,

所以你如果你今天自己在做classification,你选Mean square error的时候,你有非常大的可能性会train不起来,当然这个是在你没有好的optimizer的情况下,今天如果你用Adam,这个地方gradient很小,那gradient很小之后,它learning rate之后会自动帮你调大,也许你还是有机会走到右下角,不过这会让你的training,比较困难一点,让你training的起步呢,比较慢一点

所以这边有一个很好的例子,是告诉我们说,就算是Loss function的定义,都可能影响Training是不是容易这件事情,刚才说要用神罗天征,直接把error surface炸平,这边就是一个好的例子告诉我们说,你可以改Loss function,居然可以改变optimization的难度,

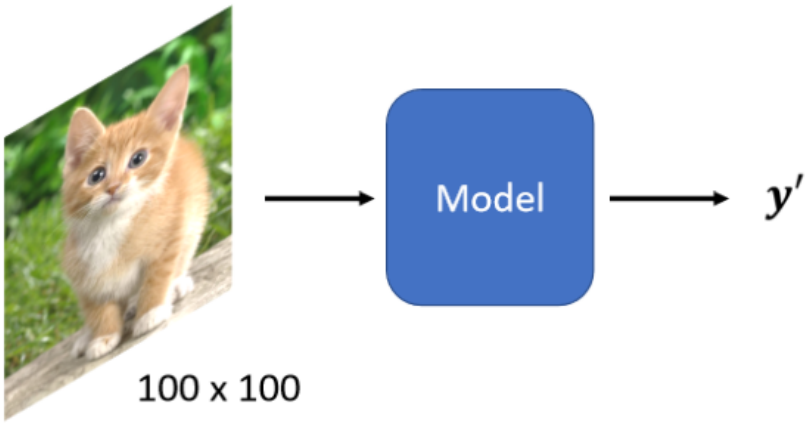
CNN

我们开始探讨 Network 的架构设计,第一个Network 架构的变形是 Convolutional 的 Neural Network,它的缩写是 CNN,它是专门被用在影像上的,我希望透过 CNN 这个例子,来让大家知道Network 的架构,它的设计有什麼样的想法,那為什麼设计 Network 的架构,可以让我们的 Network 结果做得更好。

Image Classification

接下来要讲的例子是跟影像有关的,我们要做影像的分类,也就是给机器一张图片,它要去决定说这张图片裡面有什麼样的东西,那怎麼做呢

我们已经跟大家讲过怎麼做分类这件事情,在以下的讨论裡面,我们都假设我们的模型**输入的图片大小是固定的**,举例来说 它固定输入的图片大小,都是 100 × 100 的解析度,就算是今天 Deep Learning 已经这麼的 Popular,我们往往都还是需要假设说,一个模型输入的影像大小都是一样的

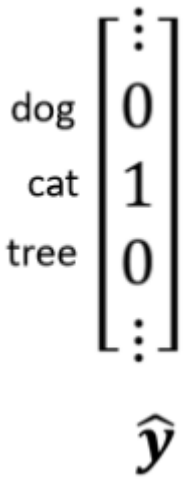


(All the images to be classified have the same size.)

图片可能有大有小,而且不是所有图片都是正方形的啊,有长方形的怎麼办

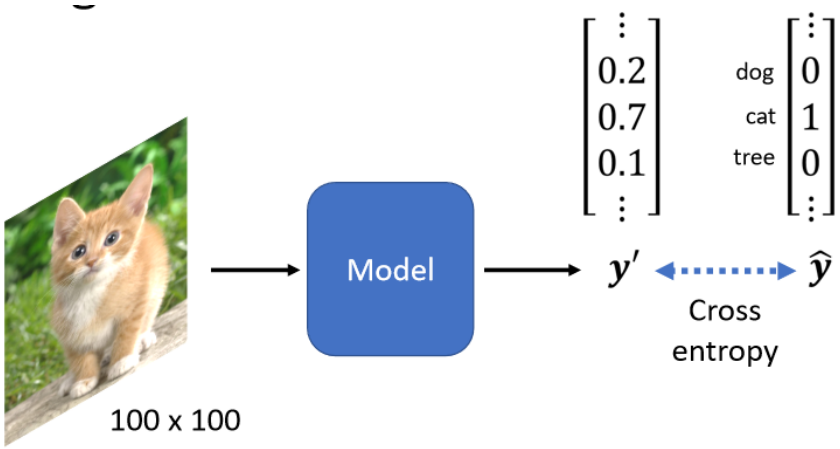
今天常见的处理方式,丢进影像辨识系统处理方式就是,**把所有图片都先 Rescale 成大小一样**,再丢到影像的辨识系统裡面,

我们的模型的输出应该是什麼呢,我们模型的目标是分类,所以我们会把每一个类别,表示成一个 One-Hot 的 Vector,我的目标就叫做 $\hat{y}$



在这个 One-Hot 的 Vector 裡面,假设我们现在**类别是一个猫的话,那猫所对应的 Dimension,它的数值就是 1,其他的东西所对的 Dimension 的数值就是 0**

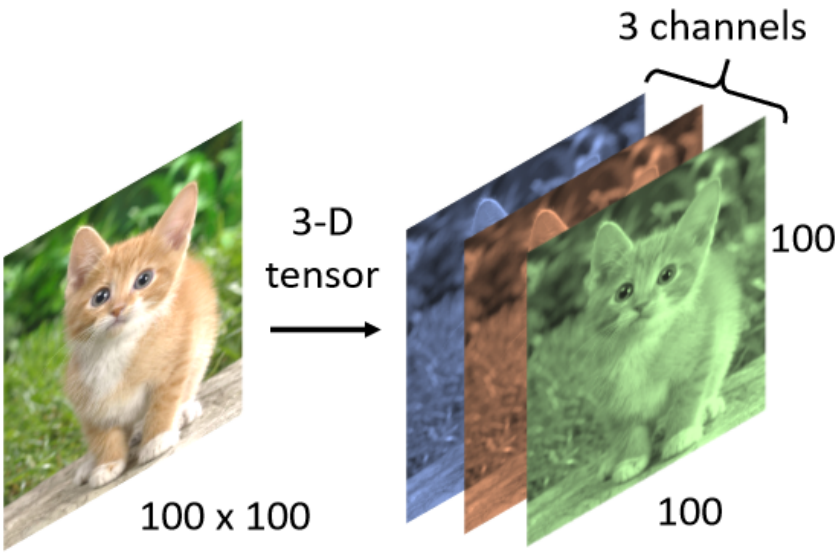
那**这个 Dimension 的长度就决定了,你现在的模型可以辨识出多少不同种类的东西**,如果你向量的长度是 2000,就代表说你这个模型,可以辨识出 2000 种不同的东西,那今天比较强的影像辨识系统,往往可以辨识出 1000 种以上的东西,甚至到上万种不同的 Object,那如果你今天希望你的影像辨识系统,它可以辨识上万种 Object,那你的 Label 就会是一个上万维,维度是上万的 One-Hot Vector



(All the images to be classified have the same size.)

我的模型的输出通过 Softmax 以后,输出是 y',然后我们希望 y' 和$\hat{y}$的 Cross Entropy 越小越好,接下来的问题是怎麼**把一张影像当做一个模型的输入**

其实对於一个 Machine 来说,**一张图片其实是一个三维的 Tensor**



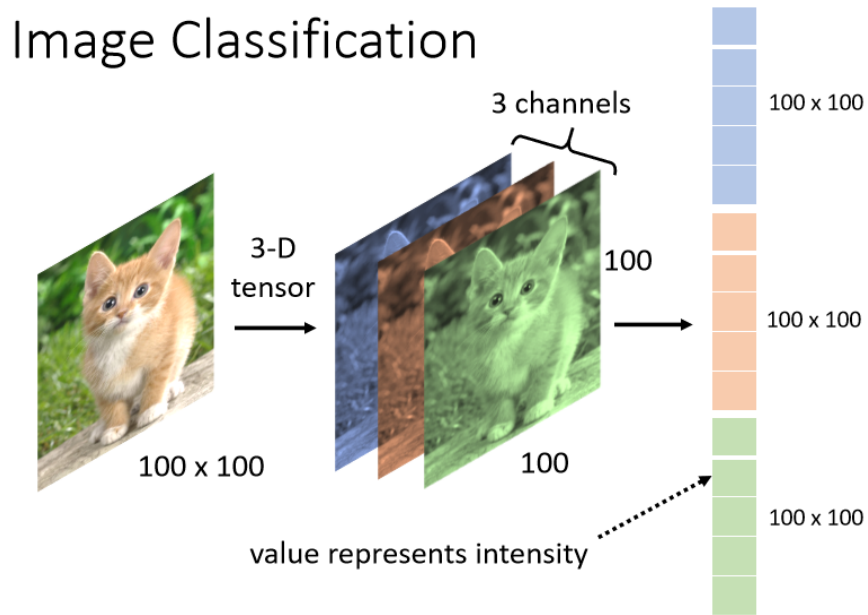
如果不知道 Tensor 是什麼的话,你就想成它是**维度大於 2 的矩阵就是 Tensor**,矩阵是二维,那二维以上的 超过二维的矩阵,你就叫它 Tensor

一张图片它是一个三维的 Tensor,其中一维代表图片的**宽**,另外一维代表图片的**高**,还有一维代表图片的 **Channel 的数目**

一张彩色的图片,今天它每一个 Pixel,都是由 R G B 三个顏色所组成的,所以这三个 Channel 就代表了 R G B 三个顏色,那长跟宽就代表了今天这张图片的解析度,代表这张图片裡面有的 Pixel,有的像素的数目

那接下来我们就要**把这一个三维的 Tensor拉直,就可以丢到一个 Network 裡面去了**

到目前為止我们所讲的 Network,它的输入其实都是一个向量,所以我们只要能够把一张图片变成一个向量,我们就可以把它当做是 Network 的输入,但是怎麼把这个三维的 Tensor 变成一个向量呢,那最直觉的方法就是直接拉直它

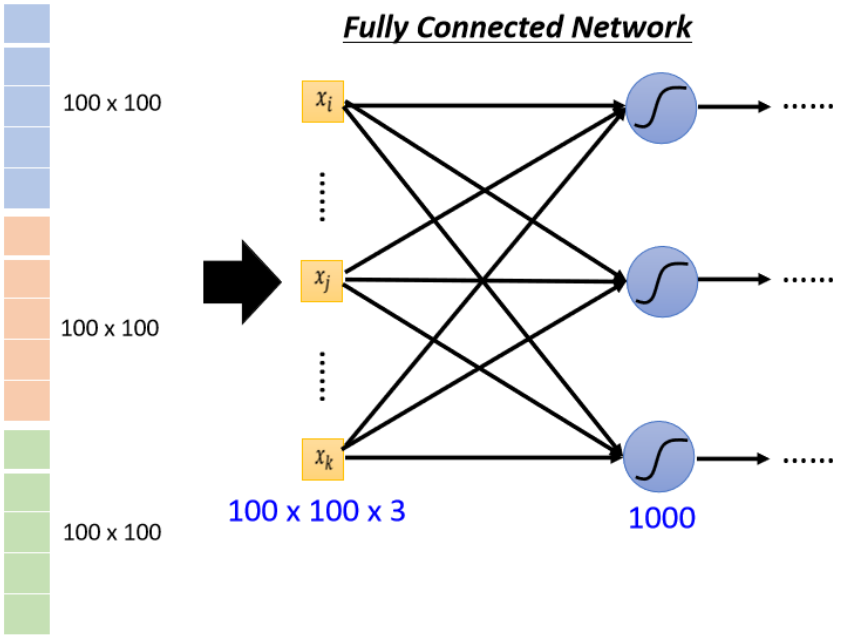


一个三维的 Tensor 裡面有几个数字呢

在这个例子裡面有 100 × 100×3 个数字,把这些数字通通拿出来排成一排,就是一个巨大的向量,这个向量可以作为 Network 的输入

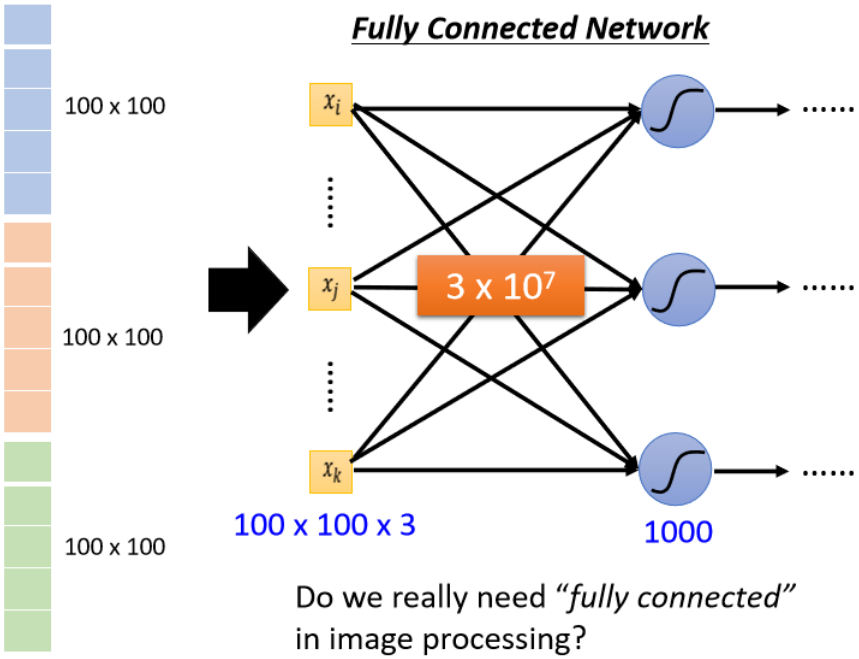
而这个向量裡面,**每一维它裡面存的数值,其实就是某一个 Pixel某一个顏色的强度**,每一个 Pixel 有 R G B 三个顏色所组成

这个向量啊,我们可以把它当做是一个 Network 的输入,那我们到目前为止,只讲过了 Fully Connected Network,好 如果我们把向量当做 Network 的输入,我们 Input 这边 Feature Vector,它的长度就是 100 × 100×3



非常长的一个 Vector,那假设我们现在的,第一层的 Neuron 的数目有 1000 个,那你能计算一下这边第一层,总共有多少个 Weight 吗

我们每一个 Neuron,它跟输入的向量的每一个数值,都会有一个 Weight,所以如果输入的向量长度是 100 × 100×3,有 1000 个 Neuron,那我们现在第一层的 Weight,就有 1000×100 × 100×3,也就是 3×10 的 7 次方



是一个非常巨大的数目,那如果参数越多会有什麼样的问题呢

虽然随著参数的增加,我们可以增加模型的弹性,我们可以增加它的能力,但是我们也**增加了 Overfitting 的风险**,有关什麼叫模型的弹性,到底 Overfitting 怎麼產生的,下週吴培元老师https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/W14_PAC-introduction.pdf会从数学上,给大家非常清楚的证明,那我们这边就讲概念上,如果模型的弹性越大,就越容易 Overfitting

那我们怎麼减少在做影像辨识的时候,怎麼避免使用这麼多的参数呢

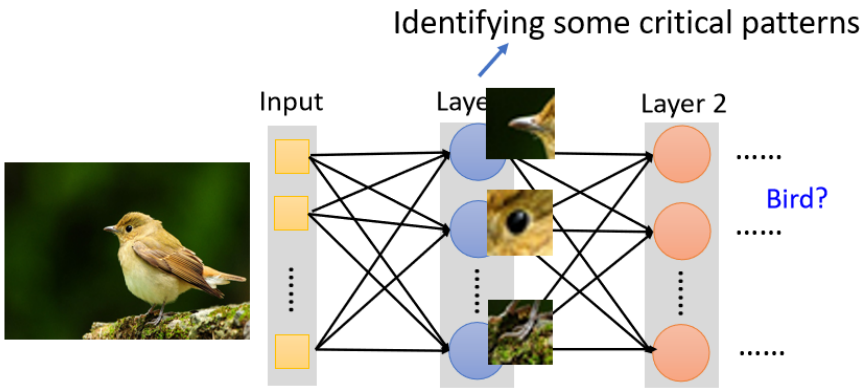
那考虑到影像辨识这个问题本身的特性,其实**我们并不一定需要 Fully Connected** 这件事,我们其实不需要每一个 Neuron,跟 Input的每一个 Dimension 都有一个 Weight

怎麼说呢,接下来就是对影像辨识这个问题,对影像本身的特性的一些观察

Observation 1

第一个观察是,对影像辨识这个问题而言,假设我们想要知道这张图片裡面有一隻动物,这个动物是一个鸟 要怎麼做呢

也许对一个影像辨识的系统而言,对一个影像辨识的 Neuron,对一个影像辨识的类神经网路裡面的神经而言,它要做的就是侦测说现在这张图片裡面,有没有出现一些特别重要的 Pattern



Perhaps human also identify birds in a similar way … ☺

举例来说 如果现在

- 有某一个 Neuron ,它看到鸟嘴这个 Pattern
- 有某个 Neuron 又说,它看到眼睛这个 Pattern
- 又有某个 Neuron 说,它看到鸟爪这个 Pattern

也许看到这些 Pattern 综合起来就代表说,我们看到了一隻鸟,类神经网路就可以告诉你说,因為看到了这些 Pattern,所以它看到了一隻鸟

那也许你会觉得说,看 Pattern 然后决定它是什麼,这件事情好像没有很聪明,但你仔细想想,**人是不是也是用同样的方法,来看一张图片中有没有一隻鸟呢,**

举例来说这一个例子,不知道你有没有看到,这裡面有什麼样的动物?



https://www.dcard.tw/f/funny/p/233833012

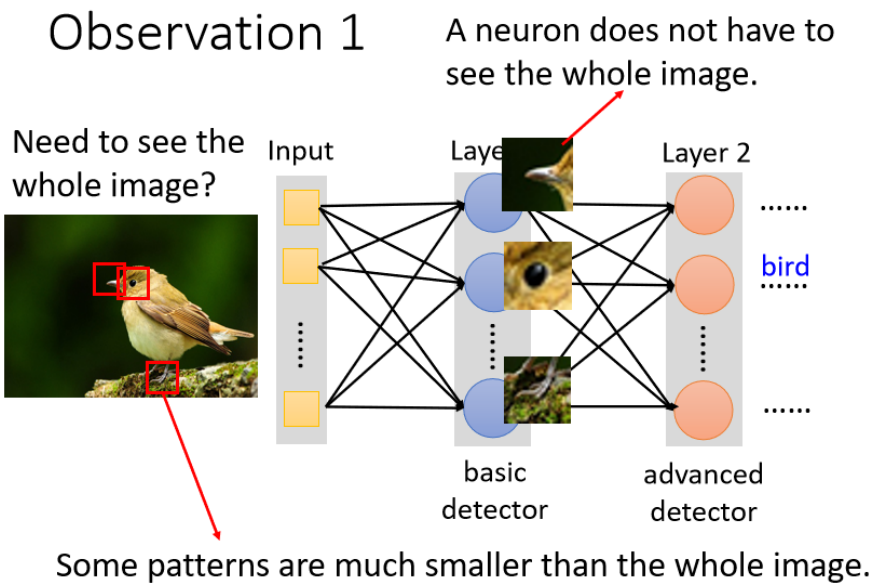
你看这边**有一个鸟嘴**,这边有一个眼睛,看起来牠是一个乌鸦

但是牠其实是一隻猫

如果你看到牠是一隻鸟的话,那你就应该放下酒杯了,因為这是一隻猫

所以其实就算是人,我们在判断一个物件的时候,往往也是抓最重要的特徵,然后看到这些特徵以后,你很直觉的就会觉得说,你看到了某种物件,对机器来说,也许这也是一个有效的,判断影像中有什麼物件的方法

但是假设我们现在用 Neuron 做的事情,其实就是判断说现在有没有某种 Pattern 出现,那也许**我们并不需要每一个 Neuron都去看一张完整的图片**



因為这一些重要的 Pattern,比如说鸟嘴 比如说眼睛 比如说鸟爪,并不需要看整张完整的图片,才能够得到这些资讯,

所以这些 Neuron 也许根本就不需要,把整张图片当作输入,它们只需要把图片的一小部分当作输入,就足以让它们侦测某些特别关键的 Pattern有没有出现了

这是第一个观察,根据这个观察,我们就可以做第一个简化 怎麼简化呢

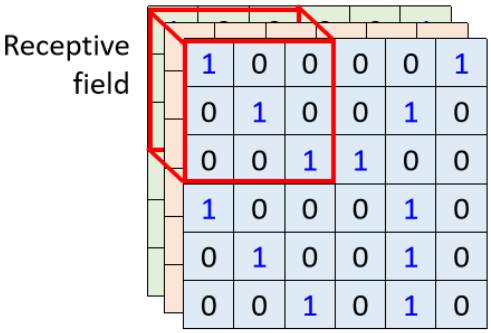
Simplification 1

在 CNN 裡面有一个这样的做法,我们会设定一个区域叫做 ==Receptive Field==,**每一个 Neuron 都只关心自己的 Receptive Field 裡面发生的事情就好了**

举例来说 你会先定义说这个**蓝色的 Neuron**,它的守备范围就是这一个 Receptive Field,那这个 Receptive Field 裡面有 3×3×3 个数值,那对蓝色的 Neuron 来说,它只需要关心这一个小范围就好了,不需要在意整张图片裡面有什麼东西

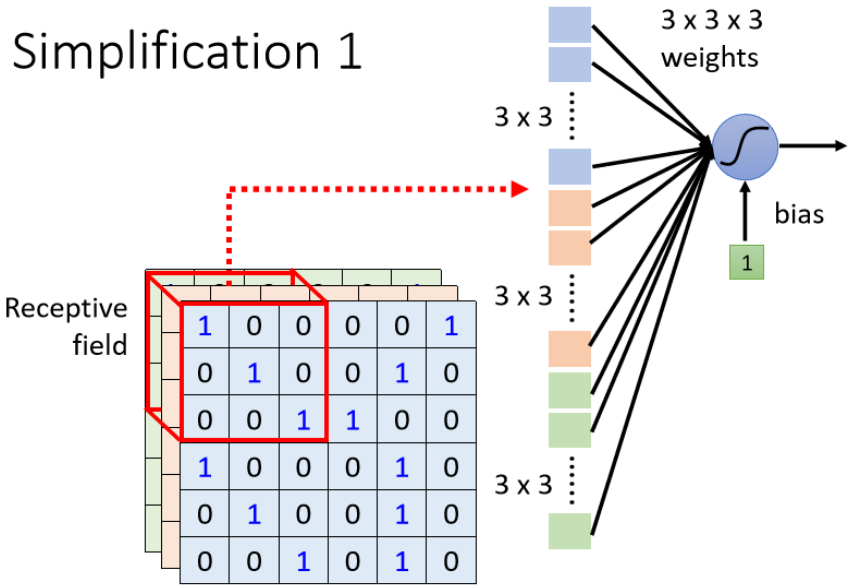
那这个 Neuron,怎麼考虑这个 Receptive Field 裡,有没有发生什麼样的事情呢



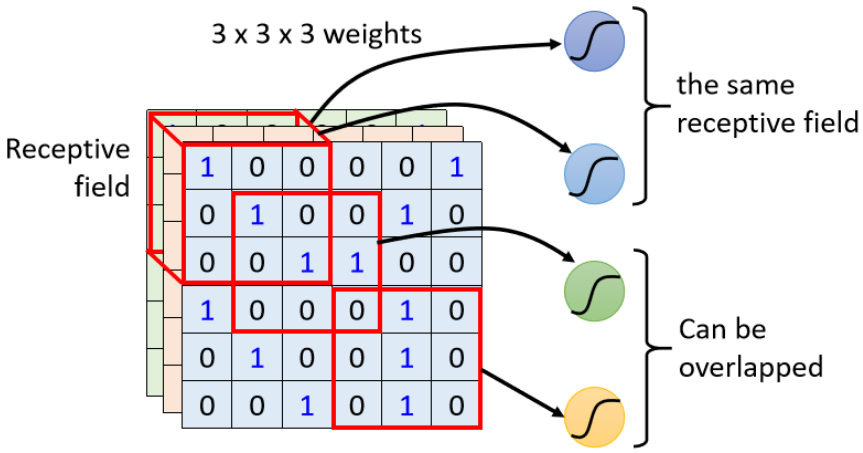


它要做的事情就是

- 把这 3×3×3 的数值**拉直**,变成一个长度是 3×3×3 也就是 27 维的向量,再把这 27 维的向量作为这个 Neuron 的输入
- 这个 Neuron 会给 27 维的向量的,**每一个 Dimension 一个 Weight,**所以这个 Neuron 有 3×3×3 27个 Weight,
- 再**加上 Bias** 得到的输出,这个输出再送给下一层的 Neuron 当作输入,

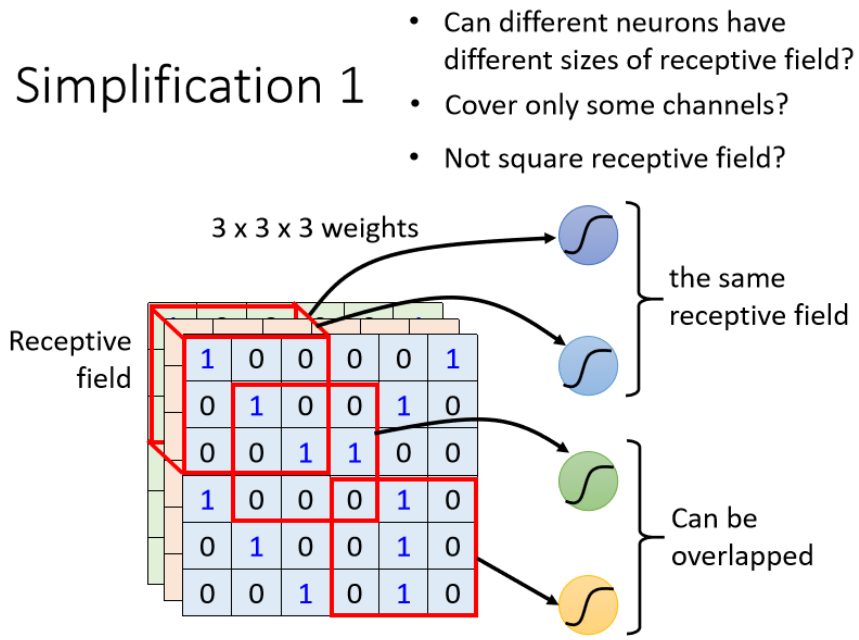


所以每一个 Neuron,它只考虑自己的 Receptive Field,那这个 Receptive Field 要怎麼决定出来呢,那**这个就要问你自己了**



- 你可以说这边有个蓝色的 Neuron,它就看左上角这个范围,这是它的 Receptive Field
- 另外又有另外一个黄色的 Neuron,它是看右下角这个 3×3×3 的范围
- 那 Receptive Field 彼此之间也**可以是重叠的**,比如说我现在画一个 Receptive Field,那这个地方它是绿色的 Neuron 的守备范围,它跟蓝色的跟黄色的都有一些重叠的空间
- 那你甚至可以**两个不同的 Neuron,它们守备看到的范围是一样的**,也许一个范围使用一个 Neuron 来守备,你没有办法侦测所有的 Pattern,所以同个范围可以有多个不同的 Neuron,所以同个 Receptive Field,它们可以有多个

那接下来你就会浮想联翩有各式各样的想法,举例来说,



- 那我可不可以 **Receptive Field 有大有小**呢?因為毕竟 Pattern 有的比较小 有的比较大,有的 Pattern 也许在 3×3 的范围内,就可以被侦测出来,有的 Pattern 也许要 11×11 的范围,才能被侦测出来,可以,这个算是常见的招式了
- 我可不可以 Receptive Field,**只考虑某些 Channel** 呢,我们这边看起来我们的 Receptive Field,是 R G B 三个 Channel 都考虑,但也许有些 Pattern,只在红色的 Channel 会出现,也许有些 Pattern,只在蓝色的 Channel 会出现啊,我可不可

以有的 Neuron 只考虑一个 Channel 呢？可以,其实之后在讲到 Network Compression的时候,会讲到这种 Network 的架构,在一般 CNN 裡面你不常这样子的考虑,但是有这样子的做法

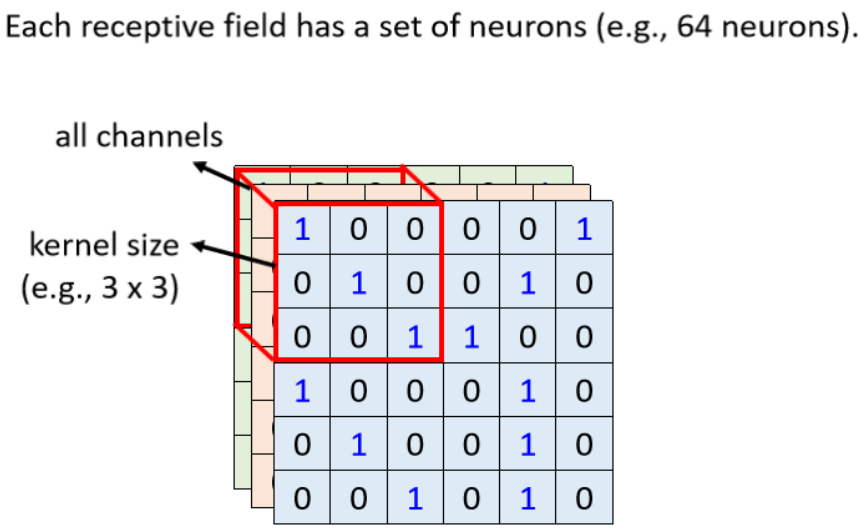
- 那有人会问说这边的 Receptive Field,通通都是正方形的,你刚才举的例子裡面,3×3 11×11 也都是正方形的,可不可以是长方形？可以！**可以是长方形的**,这完全都是你自己设计的,Receptive Field 是你自己定义的,你完全可以根据你对这个问题的理解,决定你觉得 Receptive Field 应该要长什麼样子
- 你可能会说 Receptive Field **一定要相连吗**？我们可不可以说有一个 Neuron,它的 Receptive Field 就是影像的左上角,跟右上角。**理论上可以**,但是你就要想看為什麼你要这麼做嘛,会不会有什麼 Pattern 是会,也要看一个图片的左上角,跟右下角才能够找到的,也许没有 如果没有的话,这种 Receptive Field 就没什麼用,我们之所以 Receptive Field 都是,一个相连的领地,就是我们觉得要侦测一个 Pattern,那这个 Pattern,它就出现在整个图片裡面的某一个位置,而不是出现,而不是分成好几部分,出现在图片裡面的不同的位置,所以 Receptive Field 呢 它都是相,通常见到的都是相连的领地

如果你说你要设计很奇怪的 Receptive Field,去解决很特别的问题,那完全是可以的,这都是你自己决定的

Simplification 1 – Typical Setting

虽然 Receptive Field 你可以任意设计,但也有**最经典的 Receptive Field 的安排方式**

1. **看所有的 Channel**



一般在做影像辨识的时候我们可能,你可能不会觉得有些 Pattern 只出现某一个 Channel 裡面,所以会看全部的 Channel,所以既然会看全部的 Channel

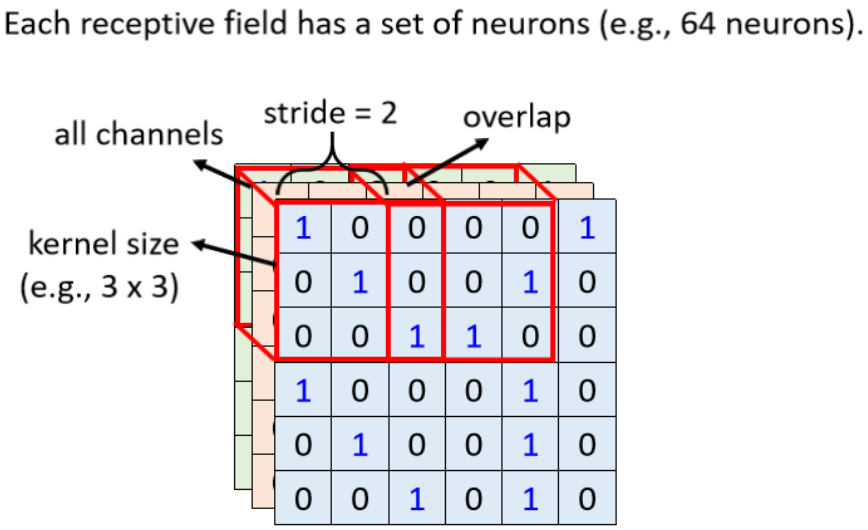
我们在描述一个 Receptive Field 的时候,**只要讲它的高跟宽就好**了,就不用讲它的深度,反正**深度一定是考虑全部的 Channel**,而这个高跟宽合起来叫做 ==Kernel Size==

举例来说在这个例子裡面,我们的 Kernel Size 就是 3×3,那一般我们 Kernel Size 其实不会设太大,你在影像辨识裡面,往往做个 3×3 的 Kernel Size 就足够了,如果你说你设个 7×7 9×9,那这算是蛮大的 Kernel Size,一般往往都做 3×3

可能会有人疑问那如果 Kernel Size 都是 3×3,意味著说我们觉得在做影像辨识的时候,重要的 Pattern 都只在 3×3 这麼小的范围内,就可以被侦测出来了,听起来好像怪怪的,有些 Pattern 也许很大啊,也许 3×3 的范围没办法侦测出来啊

等一下我们会再回答这个问题,那我现在先告诉你说,常见的 Receptive Field 设定方式,就是 Kernel Size 3×3,然后**一般同一个 Receptive Field,不会只有一个 Neuron 去关照它,往往会有一组 一排 Neuron 去守备它**,比如说 64 个 或者是 128 个 Neuron 去守备一个 Receptive Field 的范围,

1. 到目前為止我们讲的都是一个 Receptive Field,那各个不同 Receptive Field 之间的关係,是怎麼样呢,你会把你在最左上角的这个 Receptive Field,往右移一点,然后製造一个另外一个 Receptive Field,这个移动的量叫做 ==Stride==

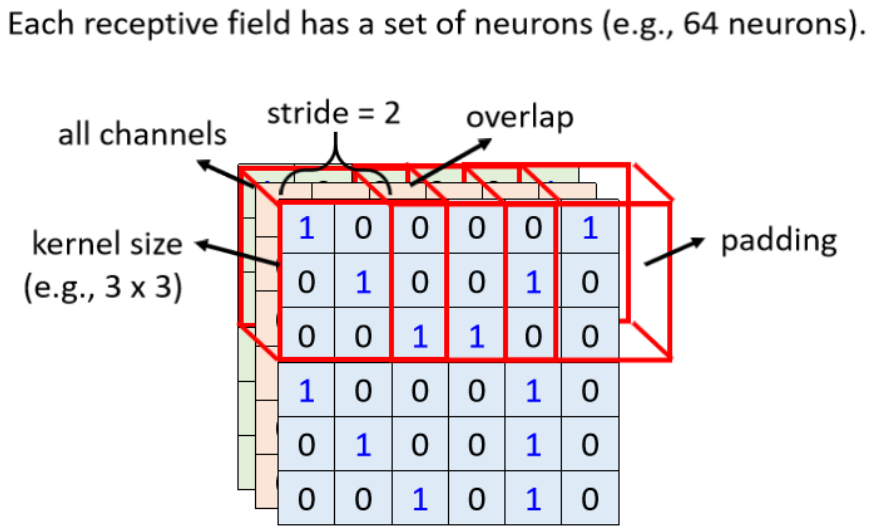


像在这个例子裡面 Stride 就等於 2,那 **Stride 是一个你自己决定的 Hyperparameter**,但这个 Stride 你往往不会设太大,往往设 1 或 2 就可以了

因為你希望这些 Receptive Field,跟 Receptive Field 之间是有重叠的,因為假设 Receptive Field 完全没有重叠,那有一个 Pattern 就正好出现,在两个 Receptive Field 的交界上面,那就会变成没有任何 Neuron 去侦测它,那你也可能就会 Miss 掉这个 Pattern,所以我们希望 Receptive Field 彼此之间,有高度的重叠

那假设我们设 Stride = 2,那第一个 Receptive Field 就在这边,那第二个就会在这边

1. 再往右移两格就放在这边,那这边就遇到一个问题了,它**超出了影像的范围**怎麼办呢

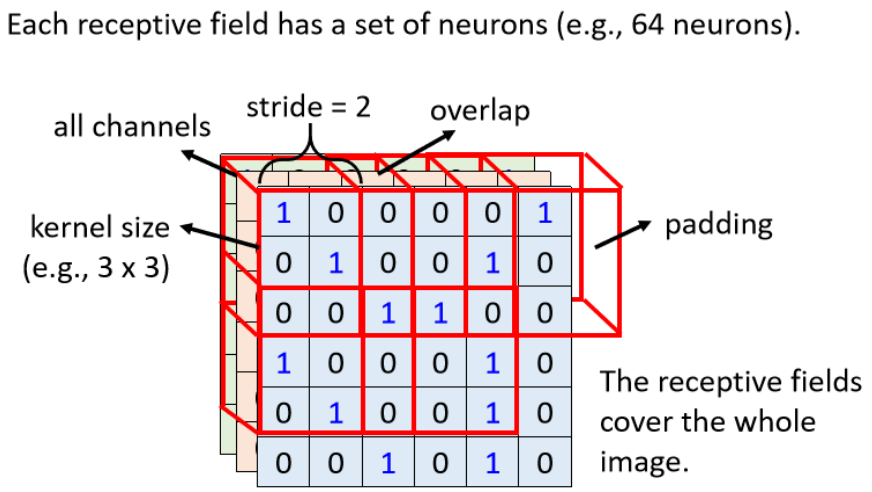


那有人可能会说,那就不要在这边摆 Receptive Field,但你这样就漏掉了影像的边边的地方啊,如果有个 Pattern 就在边边的地方,你就没有 Neuron 去关照那些 Pattern 啦,没有 Neuron 去侦测出现在边边的 Pattern 了,所以一般边边的地方也会考虑的,但超出范围了怎麼办呢

超出范围你就做 **==Padding==**,**Padding 就是补 0**,好 所以如果你今天 Receptive Field 有一部分,超出影像的范围之外了,那就当做那个裡面的值都是 0

其实也有**别的补值的方法,Padding 就是补值的意思**,比如说有人会说,我这边不要补 0 好不好,我补整张图片裡面所有 Value 的平均,或者你说,我把边边的这些数字拿出来补，有各种不同的 Padding 的方法

1. 那你**除了这个横著移动,你也会这个直著移动**,你有会有这个垂直方向上的移动,



在这边呢 我们一样垂直方向 Stride 也是设 2,所以你有一个 Receptive Field 在这边,垂直方向移动两格,就有一个 Receptive Field 在这个地方,你就按照这个方式,扫过整张图片,所以整张图片裡面,每一吋土地都是有被某一个,Receptive Field 覆盖的,也就是图片裡面每一个位置,都有一群 Neuron 在侦测那个地方,有没有出现某些 Pattern

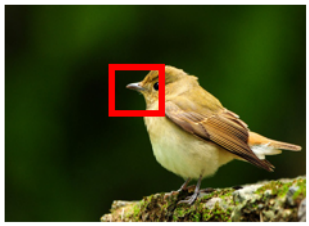
好 那这个是第一个简化,Fully Connected Network 的方式,好

Observation 2

第二个观察就是,**同样的 Pattern,它可能会出现在图片的不同区域裡面**,

- The same patterns appear in different regions.



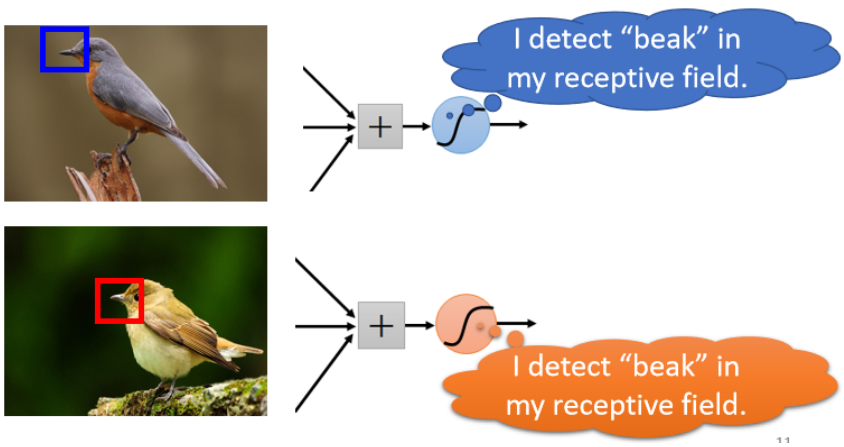


比如说鸟嘴这个 Pattern,它可能出现在图片的左上角,也可能出现在图片的中间,虽然它们的形状都是一样的 都是鸟嘴,但是它们可能出现在,图片裡面的不同的位置

按照我们刚才的讨论,你同样的 Pattern,出现在图片的不同的位置,似乎也不是太大的问题,因為出现在左上角的鸟嘴,它一定落在某一个 Receptive Field 裡面,因為 Receptive Field 是移动完之后会覆盖满整个图片的,所以图片裡面没有任何地方不是在某个 Neuron 的守备范围内

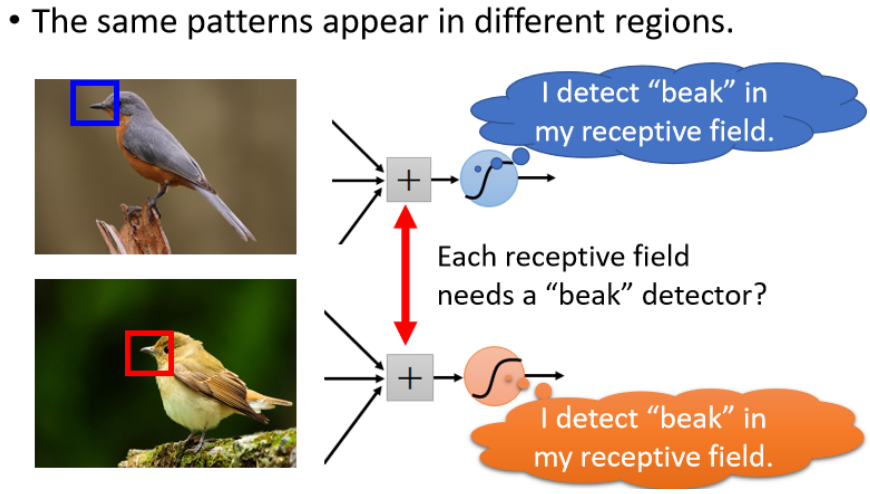
那假设在那个 Receptive Field 裡面,有一个 Neuron 它的工作,就是侦测鸟嘴的话,那鸟嘴就会被侦测出来

- The same patterns appear in different regions.



所以就算鸟嘴出现在中间,也没有关係,假设有 这边一定是在某一个,Receptive Field 的范围裡面,那个 Receptive Field,一定有一组 Neuron 在照顾,那假设其中有一个 Neuron,它可以侦测鸟嘴的话,那鸟嘴出现在图片的中间,也会被侦测出来

但这边的问题是,这些侦测鸟嘴的 Neuron,它们做的事情其实是一样的,只是它们守备的范围是不一样,**我们真的需要每一个守备范围,都去放一个侦测鸟嘴的 Neuron 吗?**

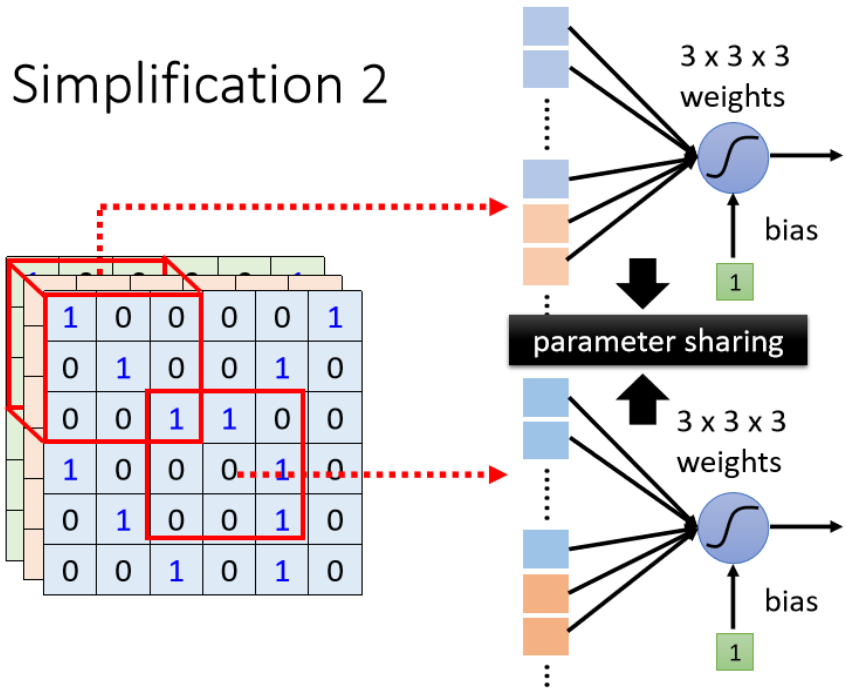


如果不同的守备范围,都要有一个侦测鸟嘴的 Neuron,那你的参数量不会太多了吗?

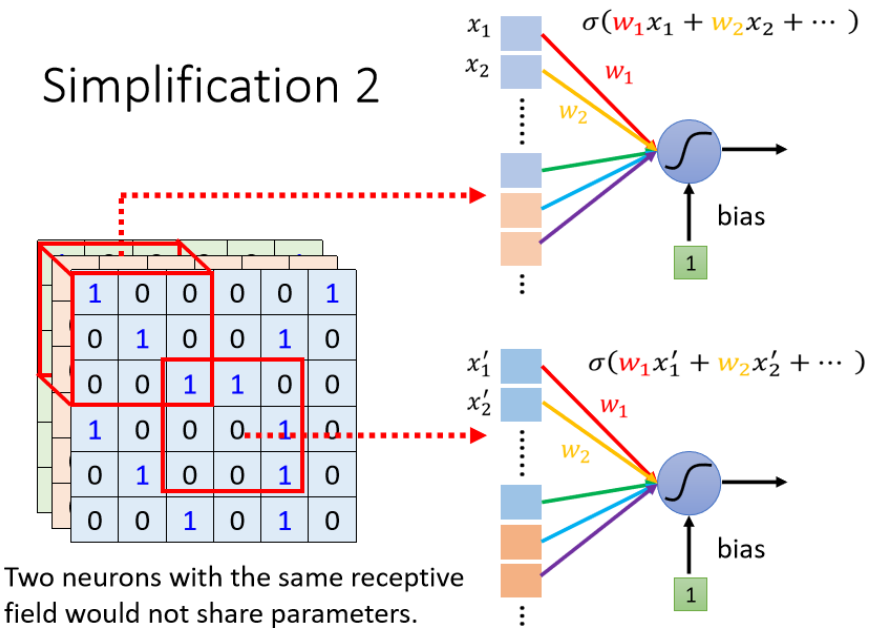
而这个概念就好像为什麼教务处希望,可以推大型的课程一样,假设说每一个科系,其实都需要程式相关的课程,或每一个科系都需要机器学习相关的课程,那到底需不需要在每一个系所都开机器学习的课程,还是说开一个比较大班的课程,让所有系所的人都可以修课。

Simplification 2

如果放在影像处理上的话,我们能不能够让,不同 Receptive Field 的 Neuron共享参数,也就是做 ==Parameter Sharing权值共享==,那共享参数是什麼意思呢



所谓共享参数就是,这**两个 Neuron 它们的 weights完全是一样的**,我这边特别用颜色来告诉你说,它们的 weights 完全是一样的



- 上面这个 Neuron 的第一个 weight,叫做 w1,下面这个 Neuron 的第一个 weight 也是 w1,它们是同一个 weight,我用**红色**来表示
- 上面这个 Neuron 的第二个 weight 是 w2,下面这个 Neuron 的第二个 weight 也是w2,它们都用**黄色**来表示,以此类推

上面这个 Neuron 跟下面这个 Neuron,**它们守备的 Receptive Field 是不一样的,但是它们的参数是一模一样的**

那有人可能就会问说,欸 它的参数是一模一样,那它会不会输出永远都是一样?

不会,因為它们的**输入是不一样**的,这两个 Neuron 的参数一模一样,但是它们**照顾的范围是不一样**的

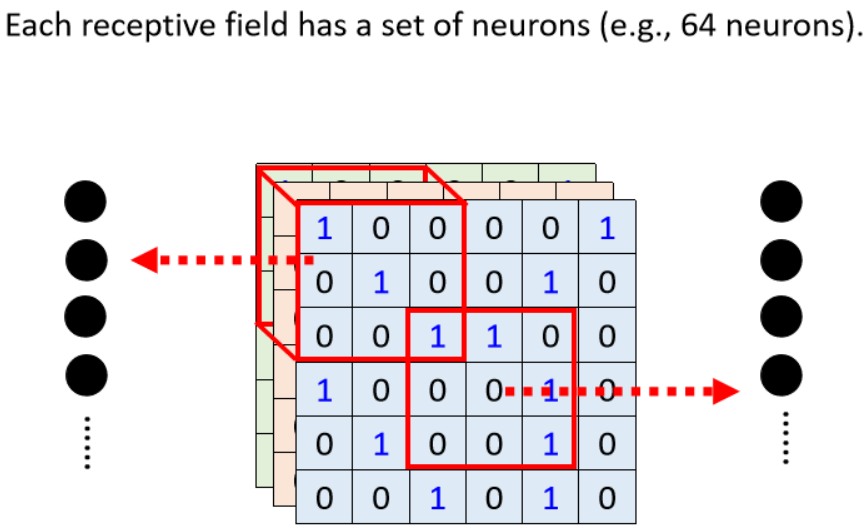
- 上面这个 Neuron,我们说它的输入是 x_1, x_2,下面这个 Neuron它的输入是 x_1' , x_2',
- 上面这个 Neuron 的输出就是,x1 × w1 + x2 × w2,全部加加加 再加 Bias,然后透过 Activation Function 得到输出 $$ \sigma(w_1x_1+w_2x_2+……) $$
- 下面这个 Neuron 虽然也有 w1 w2,但 w1 跟 w2 是乘以 x1' x2',所以它的**输出不会跟上面这个 Neuron 一样** $$ \sigma(w_1x_1'+w_2x_2'+……) $$

所以两个 Neuron 守备的范围不一样,就算它们的参数一样,它们的输出也不会是一样的,所以这是第二个简化

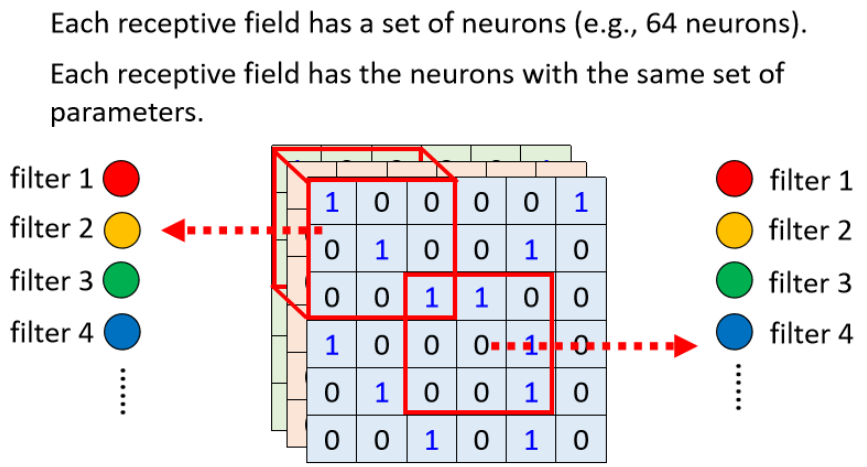
Simplification 2 – Typical Setting

我们让一些 Neuron 可以共享参数,那**至於要怎麼共享,你完全可以自己决定,而这个是你可以自己决定的事情**,但是接下来还是要告诉大家,**常见的在影像辨识上面的共享的方法**,是怎麼设定的

那我们刚才已经讲说,每一个 Receptive Field,它都有一组 Neuron 在负责守备



比如说 64 个 Neuron,所以这个 Receptive Field有 64 个 Neuron,这个 Receptive Field 也有 64 个 Neuron,那它们彼此之间会怎麼共享参数呢



我们这边用一样的颜色,就代表说这两个 Neuron,共享一样的参数

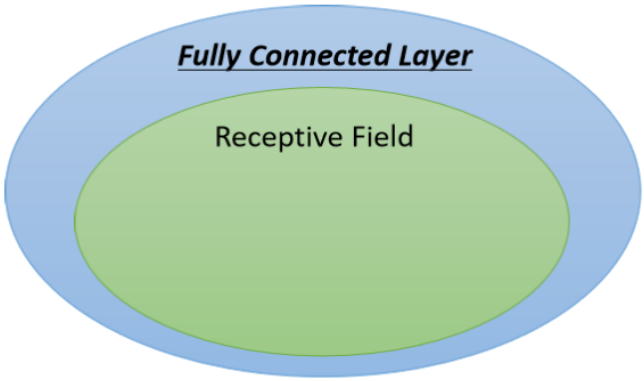
所以其实**每一个 Receptive Field都只有一组参数而已**,就是

- 左上边这个 Receptive Field 的第一个红色 Neuron,会跟右下边这个 Receptive Field 的第一个 红色Neuron 共用参数
- 它的第二个橙色 Neuron,跟它的第二个 橙色Neuron 共用参数
- 它的第三个绿色Neuron,跟它的第三个绿色 Neuron 共用参数
- 所以每一个 Receptive Field,都只有一组参数而已

那这些参数有一个名字，叫做 ==Filter==,所以这两个红色 Neuron,它们共用同一组参数,这组参数就叫 Filter1,橙色这两个 Neuron 它们共同一组参数,这组参数就叫 Filter2 叫 Filter3 叫 Filter4,以此类推

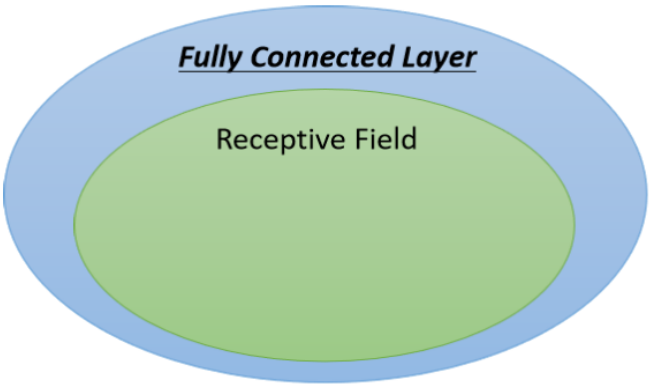
Benefit of Convolutional Layer

目前已经讲了两个简化的方法,那我们来整理一下我们学到了什麼,这是 **Fully Connected 的 Network**



- Some patterns are much smaller than the whole image.

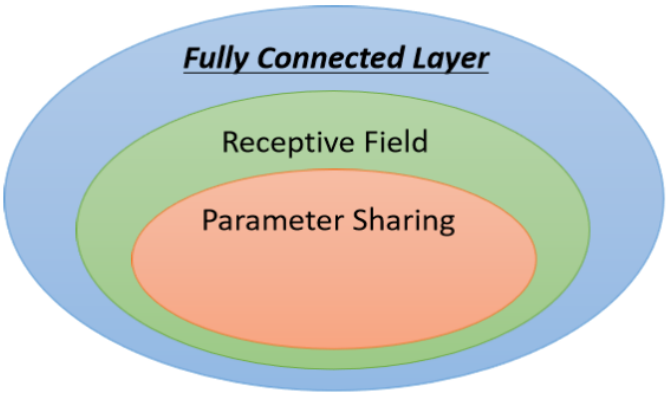
它是**弹性最大**的,但有时候不需要看整张图片,也许**只要看图片的一小部分**就可以侦测出重要的 Pattern,所以我们有了 **Receptive Field** 的概念



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

当我们强制一个 Neuron只能看一张图片裡面的一个范围的时候,它的**弹性是变小**的,如果是 Fully Connected 的 Network,它可以决定看整张图片，还是只看一个范围,就如果它只想看一个范围,就把很多 Weight 设成 0,就只看一个范围，所以加入 Receptive Field 以后,你的 Network 的弹性是变小的

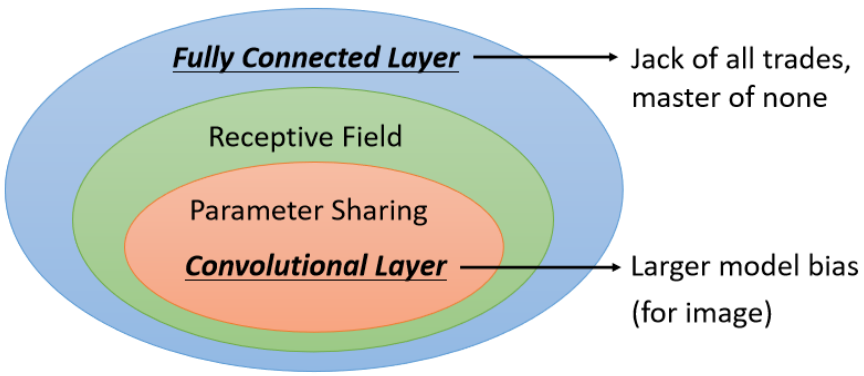
接下来我们还有权值共享,权值共享又更进一步限制了 Network 的弹性



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

本来在 Learning 的时候,它可以决定这两个 Network 的参数要是什麼,每一个 Neuron 可以各自有不同的参数,它们可以正好学出一模一样的参数,也可以有不一样的参数

但是加入参数共享以后,就意味著说 某一些 Neuron参数要一模一样,所以这又更增加了对 Neuron 的限制,而 **Receptive Field 加上 Parameter Sharing,就是 ==Convolutional Layer==**



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

有用到 Convolutional Layer 的 Network,就叫 Convolutional Neural Network,就是 CNN,所以从这个图上啊,你可以很明显地看出,其实 **CNN 的 Bias 比较大,它的 Model 的 Bias 比较大**

可能有小伙伴会说 Model Bias 比较大 不是一件坏事吗?

Model Bias 大，不一定是坏事

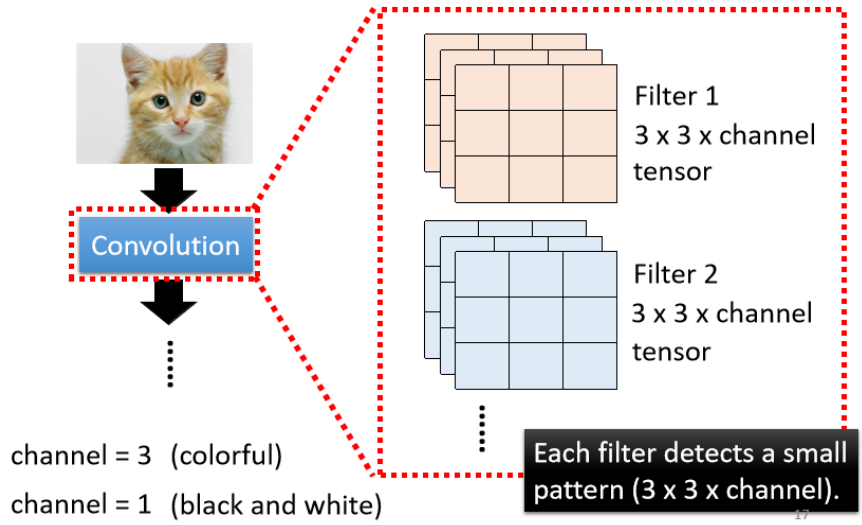
- 因為当 **Model Bias 小,Model 的 Flexibility 很高**的时候,它比较**容易 Overfitting**,Fully Connected Layer可以做各式各样的事情,它可以有各式各样的变化,但是它可能没有办法在,任何特定的任务上做好
- 而 Convolutional Layer,它是专门為影像设计的,刚才讲的 Receptive Field 参数共享,这些观察 都是為影像设计的,所以它在影像上仍然可以做得好,虽然它的 Model Bias 很大,但这个在影像上不是问题,但是如果它用在影像之外的任务,你就要仔细想想,那些任务有没有我们刚才讲的,影像用的特性

Convolutional Layer

我们以上讲的,影像用的特性,其实是 CNN 的某一种介绍方式,那现在我们要讲另外一种介绍方式,第二种介绍方式跟刚才讲的介绍方式是一模一样的,只是同一个故事,用不同的版本来说明

第二个版本是,你比较常见的说明 CNN 的方式,假设第一个版本,你听得有点迷迷糊糊的话,那我们来听第二个版本

第二个版本是这样的,**Convolutional 的 Layer 就是裡面有很多的 Filter**

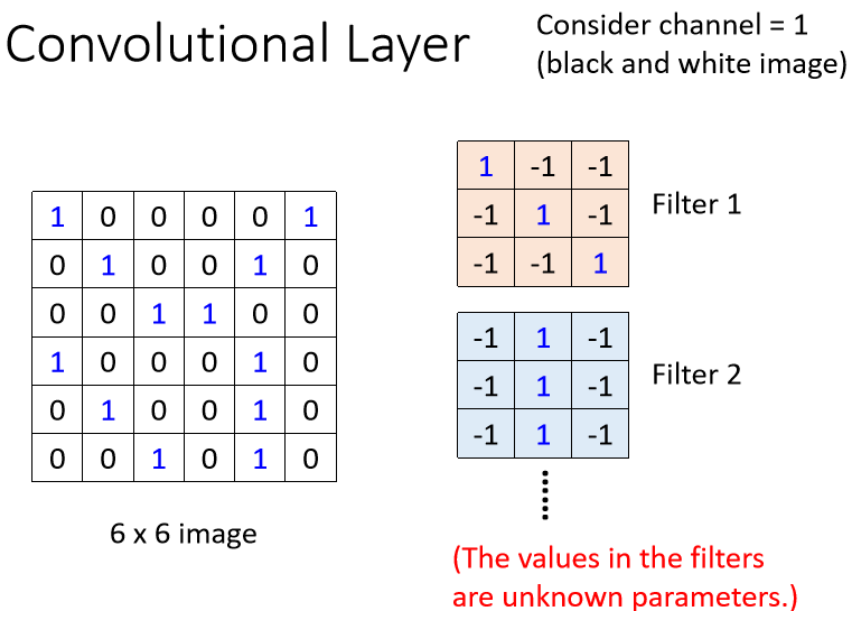


这些 Filter 啊 它们的大小是,3 × 3 × Channel 的 Size,如果今天是彩色图片的话,那就是 RGB 三个 Channel,如果是黑白的图片的话,它的 Channel 就等於 1

一个 Convolutional 的 Layer 裡面就是有一排的 Filter,每一个 Filter ,它都是一个 3 × 3 × Channel,这麼大的 Tensor

每一个 Filter 的作用就是要去图片裡面某一个 Pattern,当然这些 Pattern,要在 3 × 3 × Channel,那麼小的范围内啦,它才能够被这些 Filter 抓出来

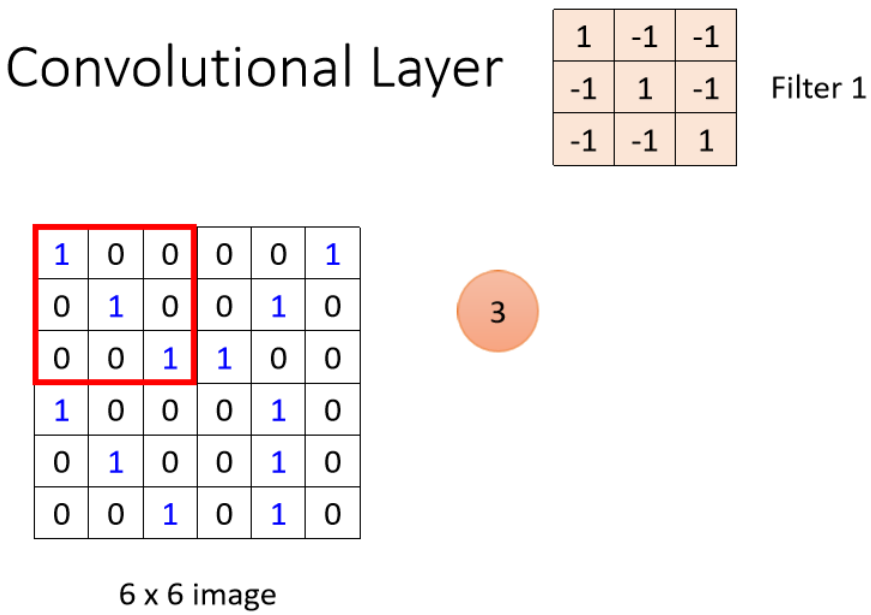
那这些 Filter,怎麼去图片裡面抓 Pattern 的呢,我们现在举一个实际的例子



例子裡面,我们假设 Channel 是 1,也就是说我们图片是**黑白的图片**

那我们假设这些 Filter 的参数是已知的,Filter 就是一个一个的 Tensor,这个 Tensor 裡面的数值,我们都已经知道了,那实际上这些 Tensor 裡面的数值,其实就是 Model 裡面的 Parameter,这些 Filter 裡面的数值,**其实是未知的,它是要透过gradient decent去找出来的**

那我们现在已经假设这些 Filter 裡面的数值已经找出来了,我们来看看说这些 Filter,是怎麼跟一张图片进行运作,怎麼去图片上面把 Pattern 侦测出来的



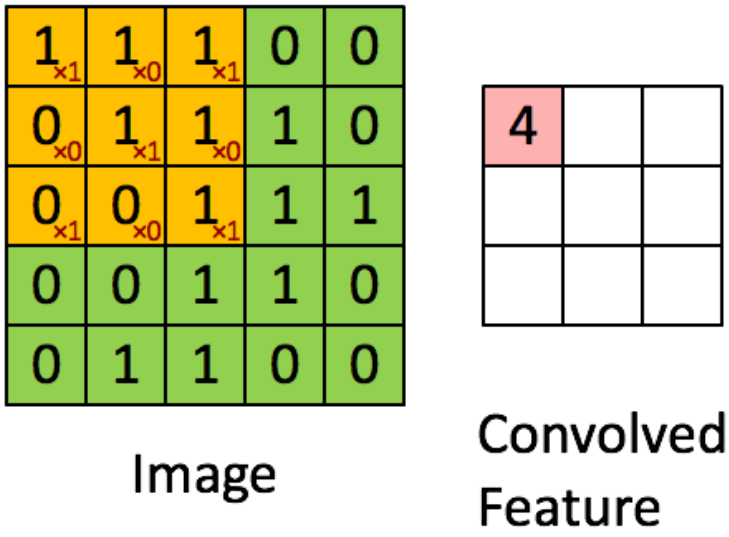
这是我们的图片,它是一个 6 × 6 的大小的图片,那这些 Filter的做法就是,先把 Filter 放在图片的左上角,然后**把 Filter 裡面所有的值,跟左上角这个范围内的 9 个值做相乘**,也就是把这个 Filter 裡面的 9 个值,跟这个范围裡面的 9 个值呢,做 Inner Product,做完是 3

==注意！在此处卷集运算中，老师讲的**内积，不要理解为**线性代数中**矩阵的乘法**，而是**filter**跟**图片对应位置**的数值**直接相乘**，所有的都乘完以后**再相加**==

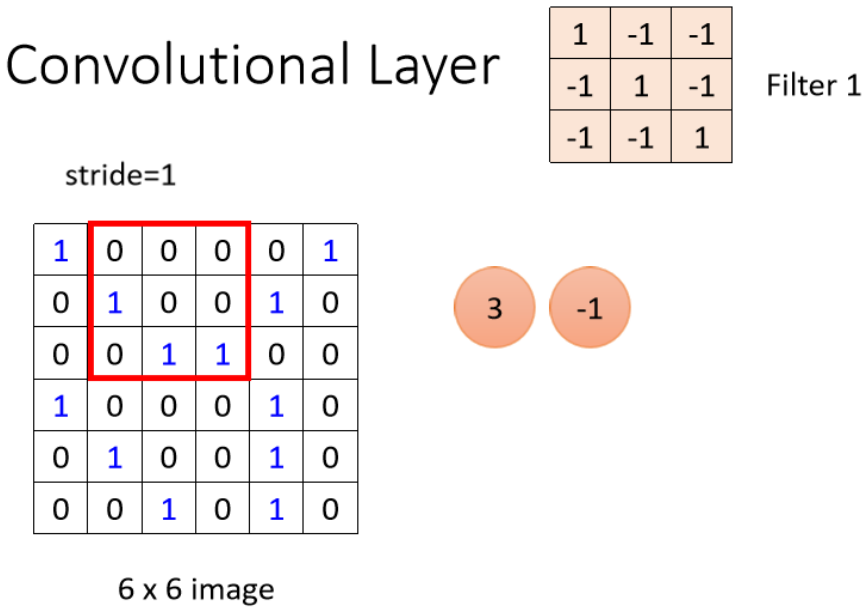
老师用的filter为$\begin{bmatrix} 1&-1&-1\ -1&1&-1\ -1&-1&1\ \end{bmatrix}$ 与图片的第一个3×3矩阵$\begin{bmatrix} 1&0&0\ 0&1&0\ 0&0&1\ \end{bmatrix}$ 做卷积运算的过程为

$(1\times1)+(-1\times0)+(-1\times0)+(-1\times0)+(1\times1)+(-1\times0)+(-1\times0)+(-1\times0)+(1\times1)=3$

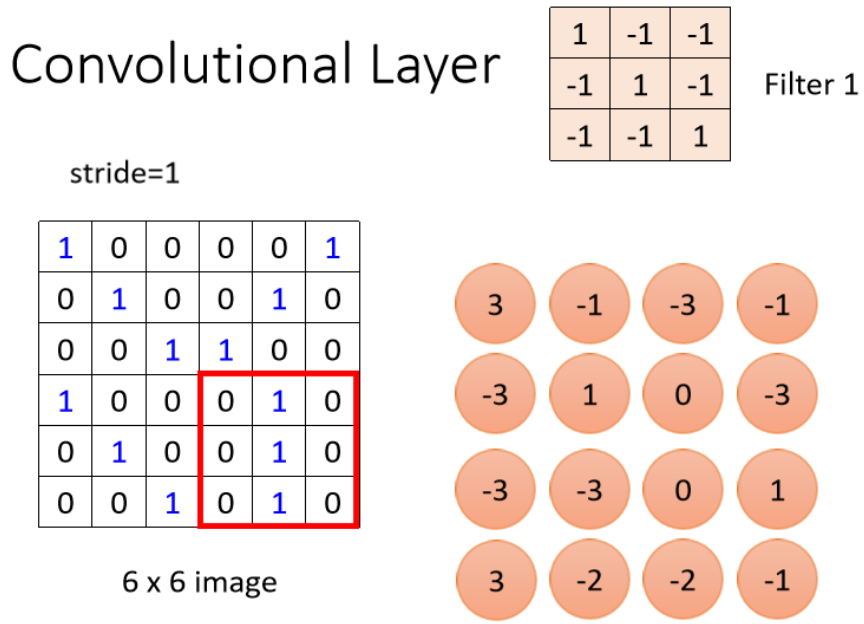
在下图中 filter为$\begin{bmatrix} 1&0&1\ 0&1&0\ 1&0&1\ \end{bmatrix}$ 大家可以自行验证



接下来这个 Filter 呢,本来放在左上角,接下来就往右移一点,那这个移动的距离叫做 ==Stride==

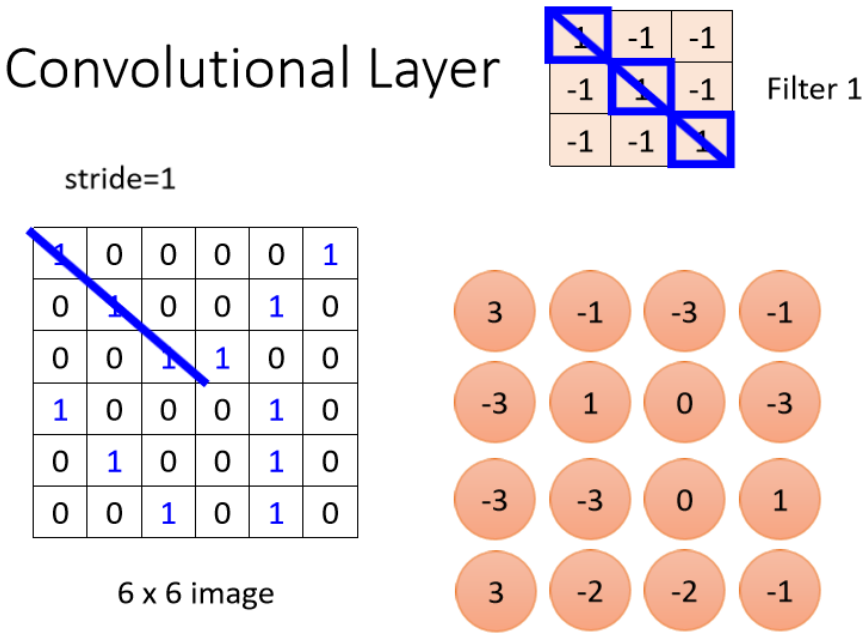


那在刚才讲前一个版本的故事的时候,我们的 Stride 是设 2,那在这个版本的故事裡面,我们 Stride 就设为 1,那往右移一点,然后再把这个 Filter,跟这个范围裡面的数值,算 Inner Product 算出来是 -1

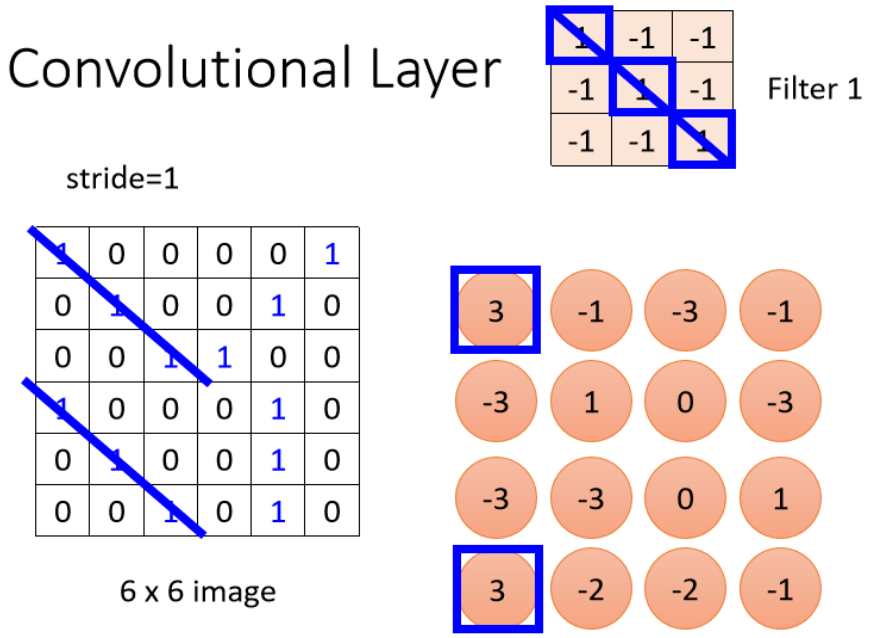


然后就以此类推,再往右移一点 再算一下,然后这边全部扫完以后,就往下移一点 再算一下,以此类推,一直到把这个 Filter 放在右下角,算出一个数值,答案就是 -1 就这样

这个 Filter 怎麼说它在侦测 Pattern 呢

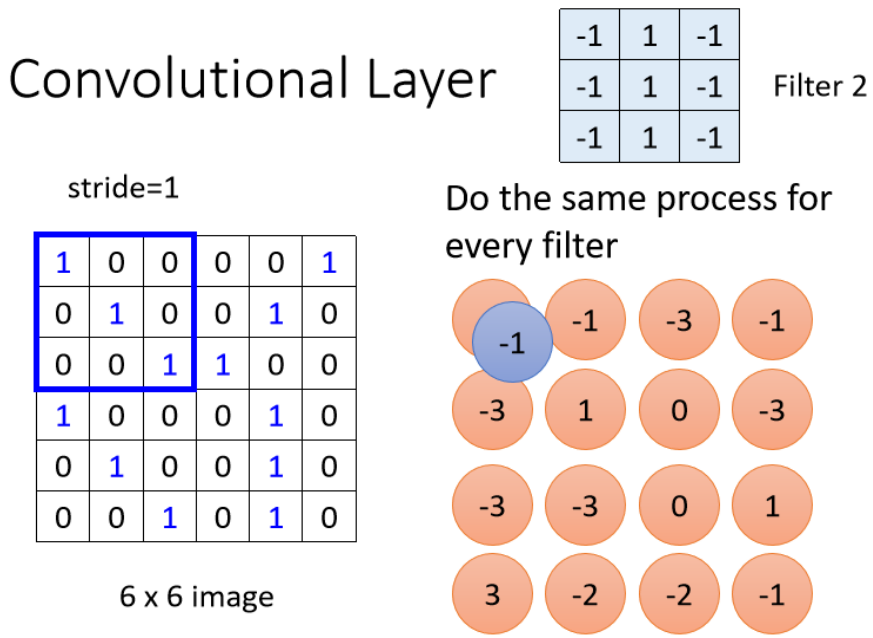


你看这个 Filter 裡面,它**对角线的地方都是1**,所以它看到 Image 裡面也出现连三个 1 的时候,它的数值会最大

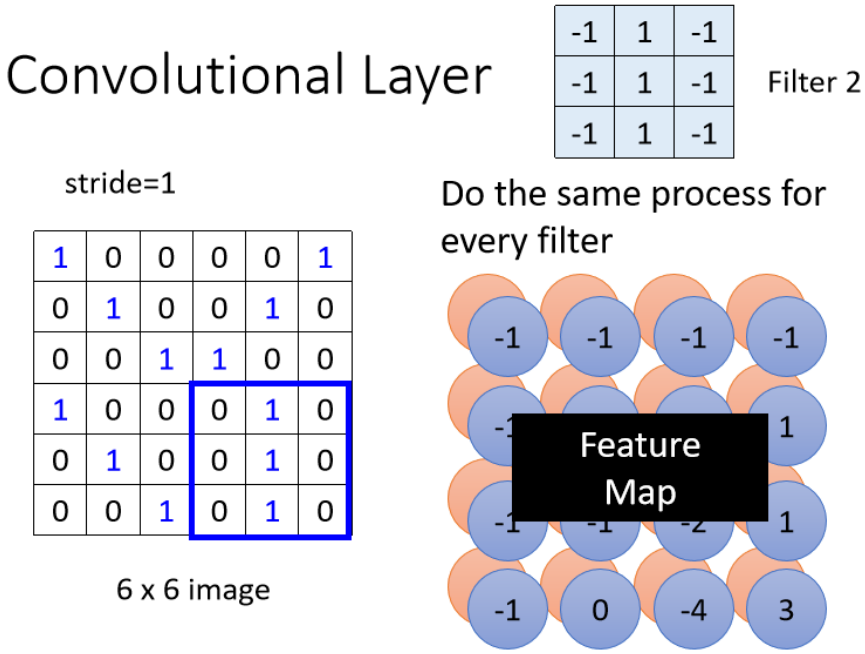


所以你会发现左上角的地方的值最大,左下角的地方的值最大,就告诉我们说,这个图片裡面左上角有出现 3,左上角有出现这个 Pattern,左下角有出现这个,三个 1 连在一起的 Pattern,这个是第一个 Filter

好 那接下来呢,我们把每一个 Filter,都做重复的 Process,比如说这边有第二个 Filter

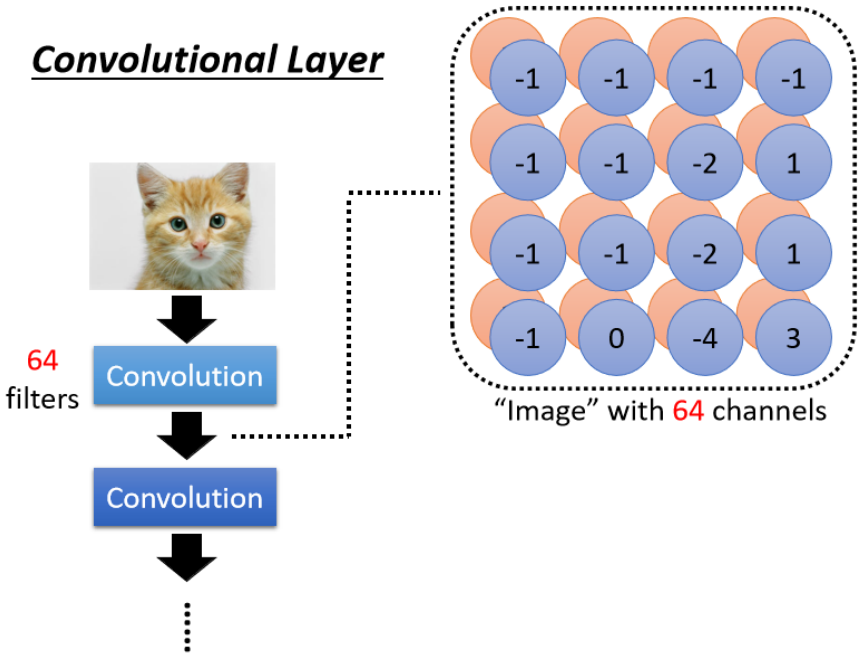


我们就把第二个 Filter,先从左上角开始扫起,得到一个数值,往右移一点 再得到一个数值,再往右移一点 再得到一个数值,反覆同样的 Process,反覆同样的操作,直到把整张图片都扫完,我们又得到另外一群数值



所以每一个 Filter,都会给我们一群数字,红色的 Filter 给我们一群数字,蓝色的 Filter 给我们一群数字,如果我们有 64 个 Filter,我们就得到 64 群的数字了,那这一群数字啊,它又有一个名字,它叫做 ==Feature Map==

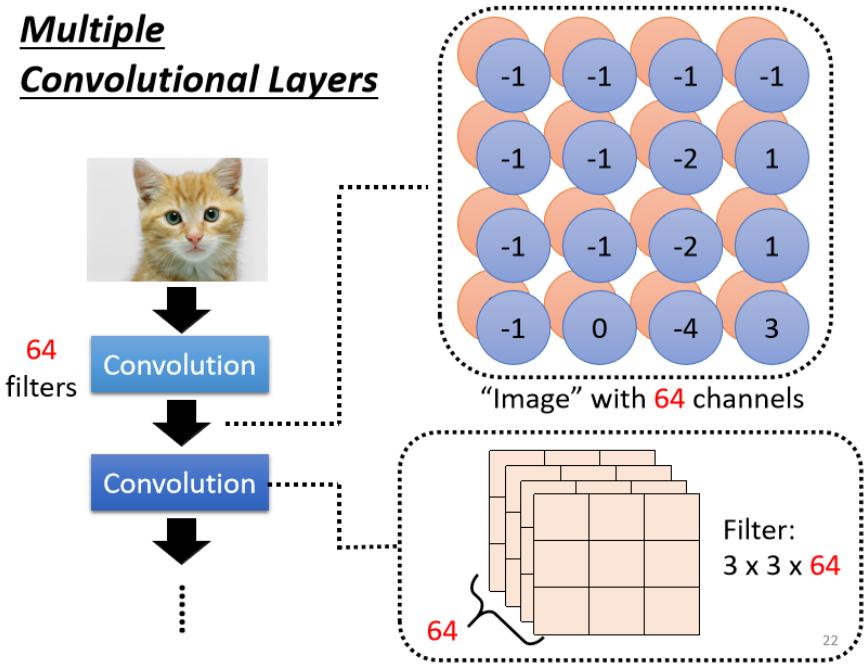
所以当我们把一张图片,通过一个 Convolutional Layer,裡面有一堆 Filter 的时候,我们產生出来了一个 Feature Map



那假设这个 Convolutional Layer裡面,它有 64 个 Filter,那我们產生出来64个 Feature Map,每一组在这个例子裡面是 4 × 4，这个 **Feature Map你可以看成是,另外一张新的图片**

只是这个图片的 Channel 它有 64 个,而且这并不是 RGB 这个图片的 Channel,在这里**每一个 Channel 就对应到一个 Filter**,本来一张图片它三个 Channel,通过一个 Convolution,它变成一张新的图片,有 64 个 Channel

这个 **Convolutional Layer 是可以疊很多层的**,刚才是疊了第一层,那如果疊第二层会发生什麼事呢



第二层的 Convolution 裡面,也有一堆的 Filter,那每一个 Filter 呢,它的大小我们这边也设 3 × 3,那**它的高度必须设為 64**

Filter 的这个高度就是它要处理的影像的 Channel,所以跟刚才第一层的 Convolution,假设输入的影像是黑白的 Channel是 1,那我们的 Filter 的高度就是 1,输入的影像如果是彩色的 Channel 是 3,那 Filter 的高度就是 3,那在第二层裡面,我们也会得到一张影像,对第二个 Convolutional Layer 来说,它的输入也是一张图片,那这个图片的 Channel 是多少,这个图片的 Channel 是 64

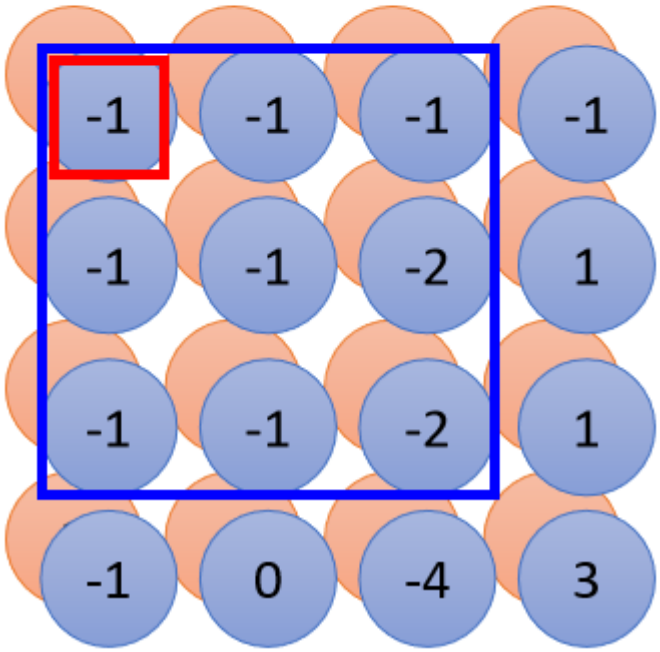
这个 64 是前一个 Convolutional Layer 的,Filter 数目,前一个 Convolutional Layer,它 Filter 数目 64,那输出以后就是 64 个 Channel

如果我们的 Filter 的大小一直设 3 × 3,会不会让我们的 Network,没有办法看比较大范围的 Pattern 呢

其实不会的,因為你想想看,如果我们在第二层 Convolutional Layer,我们的 Filter 的大小一样设 3 × 3 的话,会发生什麼事情呢

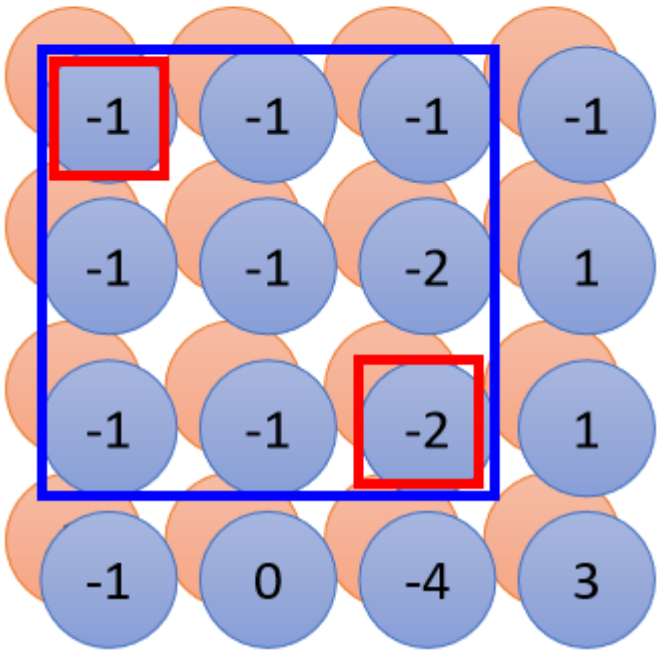
如果我们一样设 3 × 3 的话,当我们看最左上角这个数值的时候,最左上角这个数值在影像上,其实是对应到这个范围,

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

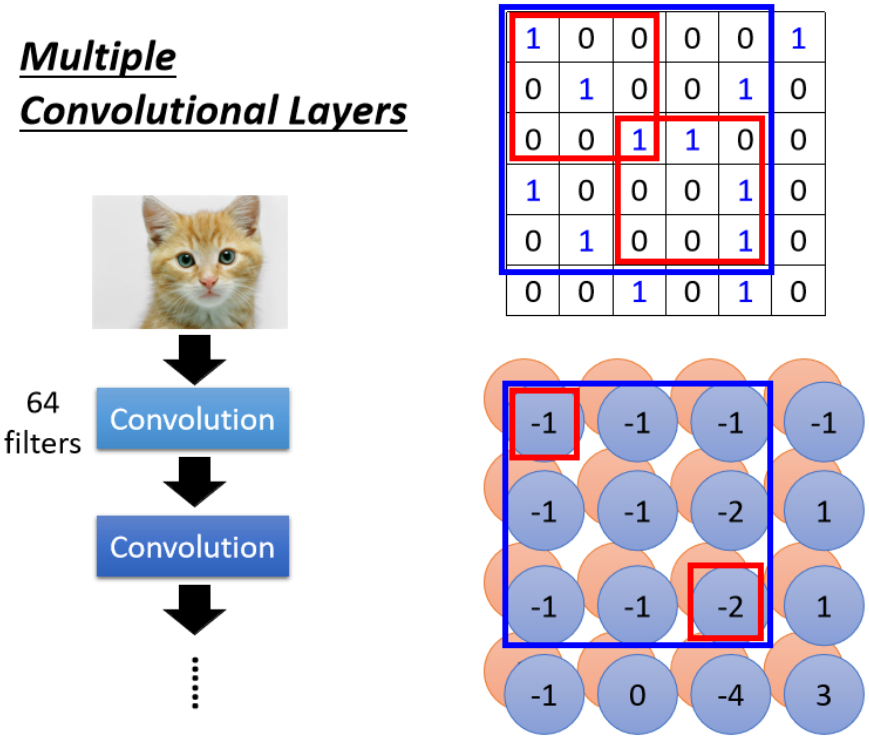


右下角的数值在影像上,其实是对应到这个范围,

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

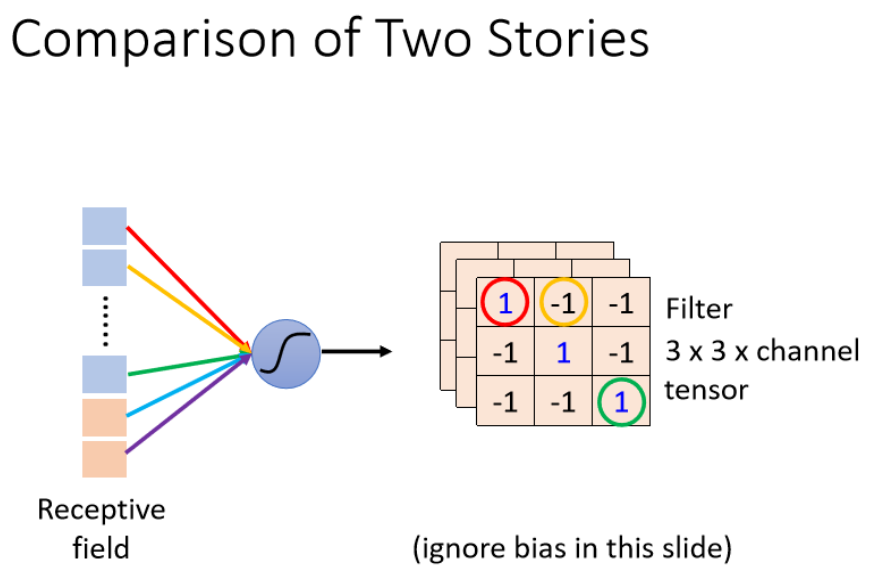


所以当我们看这 3 × 3 的范围的时候,和第一个 Convolutional Layer 的输出的,这个 Feature Map 的,3 × 3 的范围的时候,我们在原来的影像上,其实是考虑了一个 5 × 5 的范围



所以虽然我们的 Filter 只有 3 × 3,但它在影像上考虑的范围,是比较大的 是 5 × 5,所以今天你的 **Network 叠得越深,同样是 3 × 3 的大小的 Filter,它看的范围就会越来越大**,所以 Network 够深,你不用怕你侦测不到比较大的 Pattern,它还是可以侦测到比较大的 Pattern

刚才我们讲了两个版本的故事了,那这两个版本的故事,是一模一样的

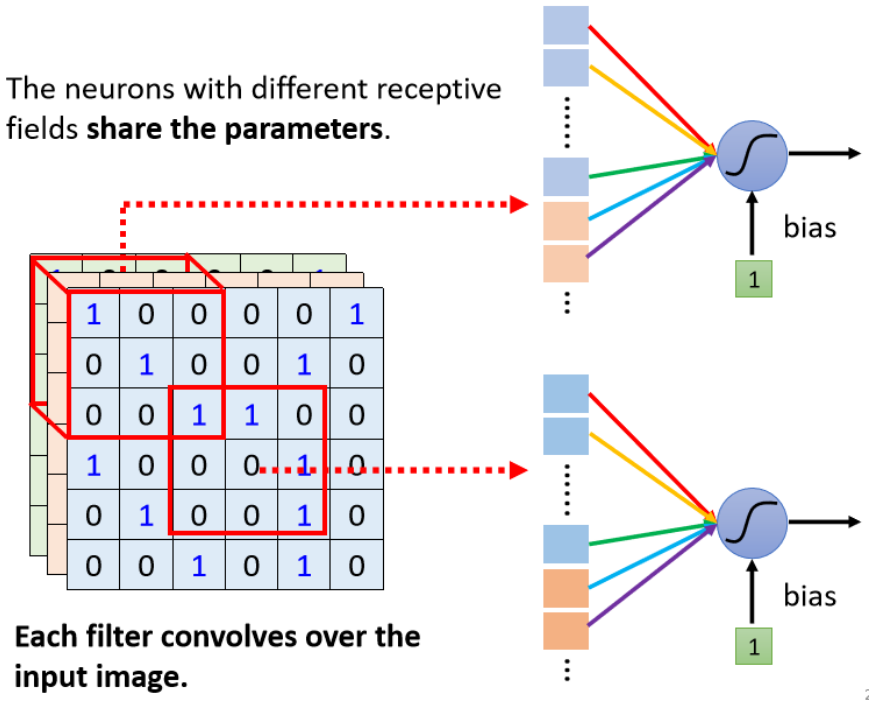


我们在第一个版本的故事裡面,说到了有一些 Neuron,这些 Neuron 会共用参数,这些**共用的参数,就是第二个版本的故事裡面的 Filter**

我们这个 Filter 裡面有 3 × 3 × 3个数字,我这边特别还**用颜色,把这些数字圈起来,告诉你说 这个 Weight 就是这个数字**

以此类推,那这边我把 Bias 去掉了, Neuron 这个是有 Bias 的,这个 Filter 有没有 Bias 呢,其实是有的,只是在刚才的故事裡面没有提到,在一般的这个实作上,你的 CNN 的这些 Filter,其实都还是有那个 Bias 的数值的

在刚才第一个版本的故事裡面,我们说不同的 Neuron,它们可以 Share Weight,然后去守备不同的范围,而 **Share Weight 这件事,其实就是我们把 Filter 扫过一张图片**



那把 **Filter 扫过一张图片这件事,其实就是 Convolution**,这就是為什麼 Convolutional Layer,要叫 Convolutional Layer 的关係

那所谓的把 Filter 扫过图片这件事情,其实就是不同的 Receptive Field,Neuron 可以共用参数,而这组共用的参数,就叫做一个 Filter

今天特别从两个不同的面向,讲 CNN 这个东西,希望可以帮助你对 CNN 有更深地了解

Convolutional Layer

Neuron Version Story	***Filter Version Story***
Each neuron only considers a receptive field.	There are a set of filters detecting small patterns.
The neurons with different receptive fields share the parameters.	Each filter convolves over the input image.

They are the same story.

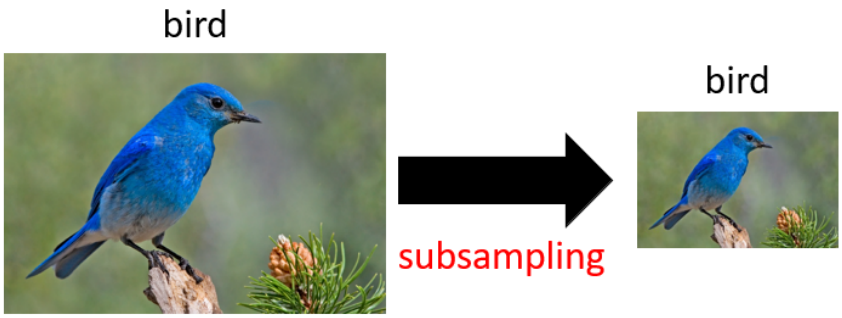
為什麼用 CNN 是基於两个观察

- 第一个观察是 我们不需要看整张图片,那对 Neuron 的故事版本,对於第一个故事而言就是,Neuron 只看图片的一小部分,对 Filter 的故事而言就是,我们有一组 Filter,每个 Filter 只看一个小范围,它只侦测小的 Pattern
- 然后我们说 同样的 Pattern,可能出现在图片的不同的地方,所以 Neuron 间可以共用参数,对 Filter 的故事而言就是,一个 Filter 要扫过整张图片,这个就是 Convolutional Layer

Observation 3

Convolutional Layer,在做影像辨识的时候呢,其实还有第三个常用的东西,这个东西呢 叫做 Pooling,那 Pooling 是怎麼来的呢

• Subsampling the pixels will not change the object



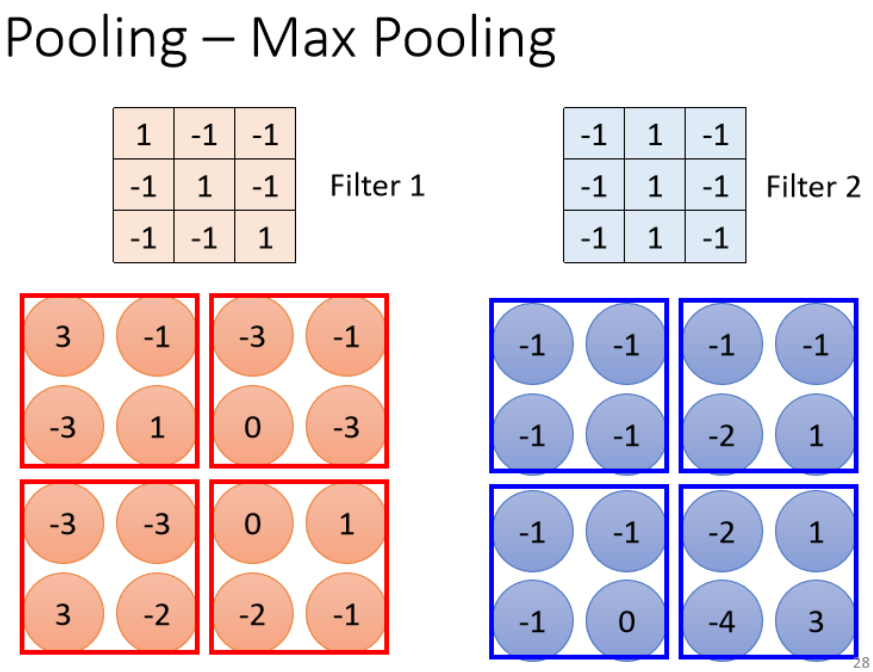
Pooling 来自於另外一个观察

我们把一张比较大的图片做 Subsampling,举例来说你把偶数的 Column 都拿掉,奇数的 Row 都拿掉,图片变成為原来的1/4,但是不会影响裡面是什麼东西,把一张大的图片缩小,这是一隻鸟,这张小的图片看起来还是一隻鸟

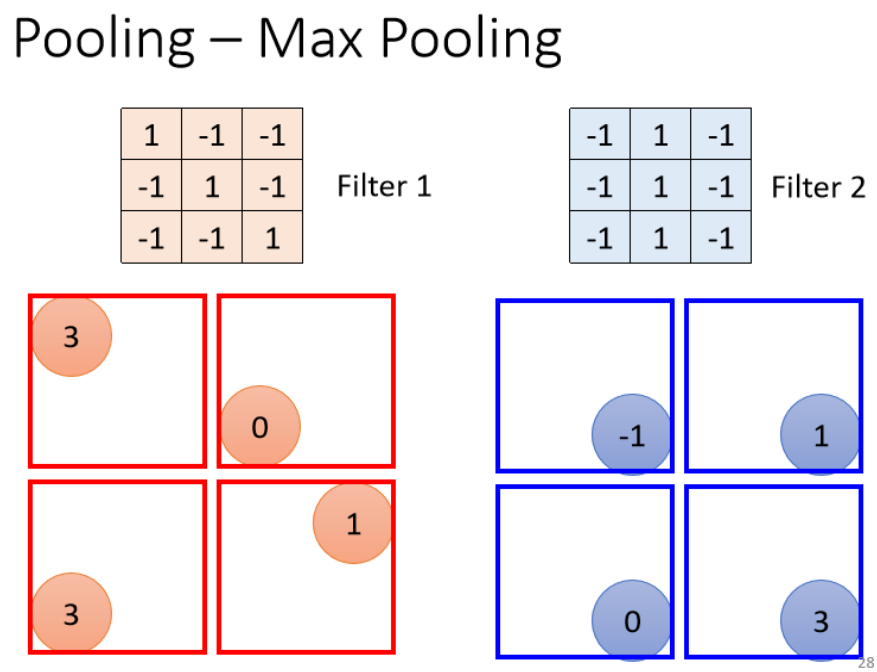
那所以呢 有了Pooling 这样的设计,那 Pooling 是怎麼运作的呢

Pooling 这个东西啊,它本身没有参数,所以它不是一个 Layer,它裡面没有 Weight,它没有要 Learn 的东西,所以有人会告诉你说 Pooling 比较像是一个 Activation Function,比较像是 Sigmoid， ReLU 那些,因為它裡面是没有要 Learn 的东西的,它就是一个 Operator,它的行為都是固定好的,没有要根据 Data 学任何东西

那 Pooling 其实也有很多不同的版本,我们这边讲的是 Max Pooling



刚才说每一个 Filter 都產生一把数字,要做 Pooling 的时候,我们就把这些数字几个几个一组,比如说在这个例子裡面就是 2×2 个一组,每一组裡面选一个代表,在 Max Pooling 裡面,我们选的代表就是**最大的那一个**

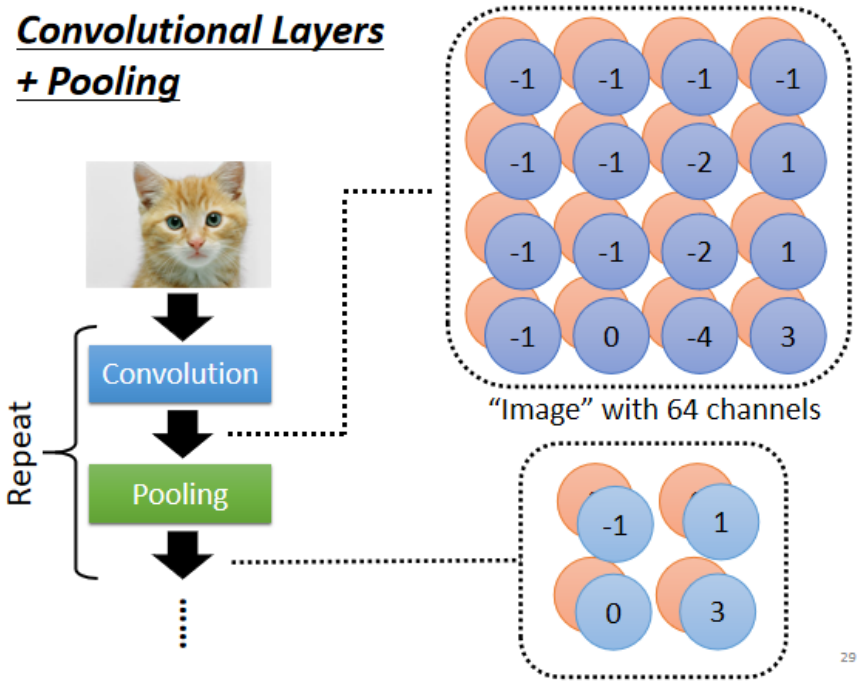


你不一定要选最大的那一个,这个是你自己可以决定的,Max Pooling 这一个方法是选最大的那一个,但是也有 average Pooling， 就是选平均嘛,还有选几何平均的,所以有各式各样的 Pooling 的方法

也一定要 2×2 个一组吗,这个也是你自己决定的,你要 3×3 4×4 也可以

Convolutional Layers + Pooling

所以我们做完 Convolution 以后,往往后面还会搭配 Pooling,那 **Pooling 做的事情就是把图片变小**,做完 Convolution 以后我们会得到一张图片,这一张图片裡面有很多的 Channel,那做完 Pooling 以后,我们就是把这张图片的 Channel 不变,本来 64 个 Channel 还是 64 个 Channel,但是我们会把图片变得比较狭长一点



在刚才的例子裡面,本来 4×4 的图片,如果我们把这个 Output 的数值啊,2×2 个一组的话,那 4×4 的图片就会变成 2×2 的图片,这个就是 Pooling 所做的事情

那一般在实作上啊,往往就是 Convolution 跟 Pooling 交替使用,就是你可能做几次 Convolution,做一次 Pooling,比如两次 Convolution 一次 Pooling,两次 Convolution 一次 Pooling

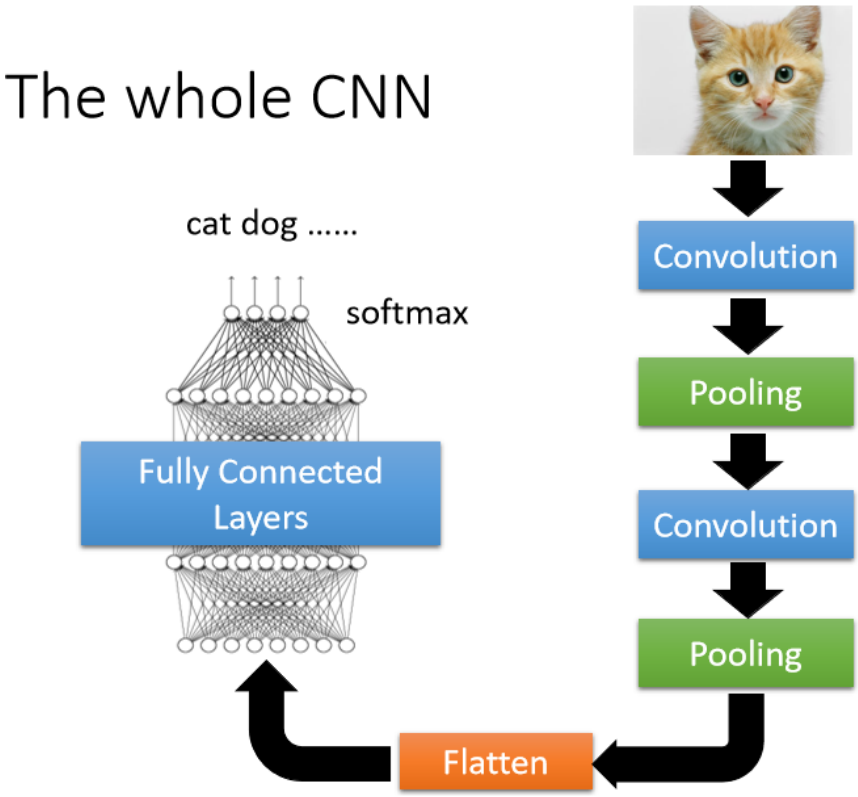
不过你可以想见说 **Pooling,对於你的 Performance,还是可能会带来一点伤害的**,因為假设你今天要侦测的是非常微细的东西,那你随便做 Subsampling,Performance 可能会稍微差一点

所以近年来你会发现,很多影像电视的 Network 的设计,往往也开始把 Pooling 丢掉,他会做这种,Full Convolution 的 Neural Network,就整个 Network 裡面统统都是 Convolution,完全都不用 Pooling

那是因為近年来运算能力越来越强,Pooling 最主要的理由是為了减少运算量,做 Subsampling,把影像变少 减少运算量,那如果你今天你的运算资源,足够支撑你不做 Pooling 的话,很多 Network 的架构的设计,往往今天就不做 Pooling,全 Convolution,Convolution 从头到尾,然后看看做不做得起来,看看能不能做得更好

The whole CNN

那一般以后,你的架构就是 Convolution 加 Pooling,那我刚才讲过说 Pooling 是可有可无啦,今天很多人可能会选择不用 Pooling,好 那如果你做完几次 Convolution 以后,接下来呢,最终怎麼得到最后的结果呢

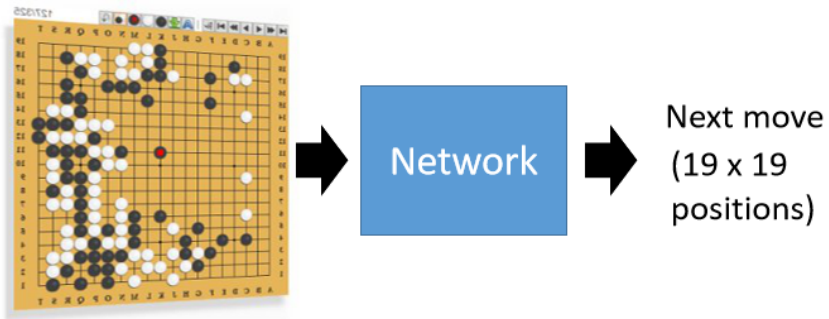


你会把 Pooling 的 Output 做一件事情,叫做 Flatten,Flatten 的意思就是,把这个影像裡面啊,本来排成矩阵的样子的东西拉直,把所有的数值拉直变成一个向量,再把这个向量,丢进 Fully Connected 的 Layer 裡面

最终你可能还要过个 Softmax,然后最终得到影像辨识的结果,这就是一个经典的影像辨识的Network,它可能有的样子就是长这样,裡面有 Convolution,有 Pooling 有 Flatten,最后再通过几个,Fully Connected 的 Layer 或 Softmax,最终得到影像辨识的结果

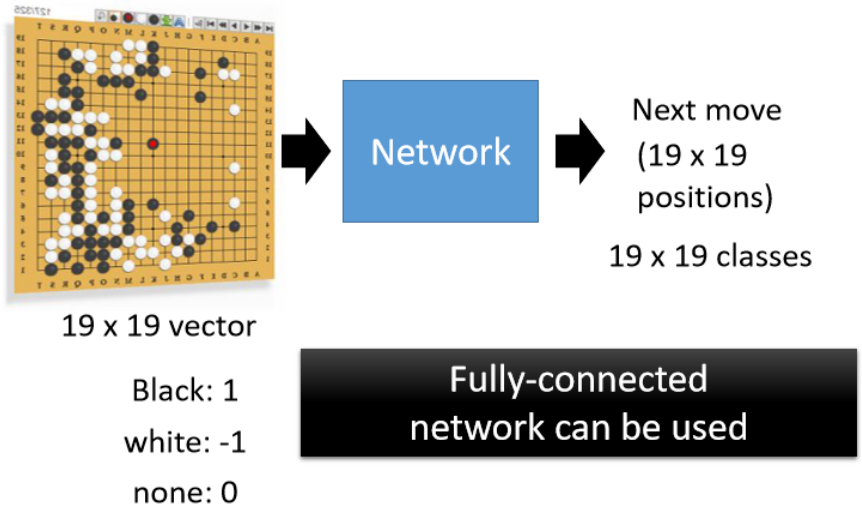
Application： Alpha Go

那除了影像辨识以外啊,你可能听过 CNN 另外一个最常见的,最耳熟能详的应用,就是用来下围棋,那今天呢 如果讲个机器学习的课,没有提到 AlphaGo,大家就觉得你什麼都没有讲对不对,所以我们来提一下 AlphaGo,好 怎麼用这个 CNN 来下围棋呢



我们说下围棋其实就是一个分类的问题,你的 Network 的输入,是棋盘上黑子跟白子的位置,你的输出就是下一步应该要落子的位置

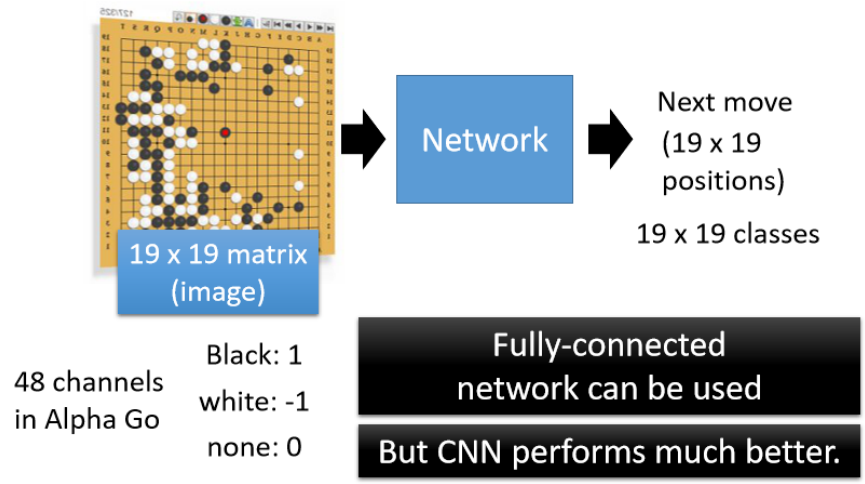
可是我们今天已经知道说,Network 的输入就是一个向量啊,那**怎麼把棋盘表示成一个向量**呢,完全没有问题,棋盘上就是有 19 × 19 个位置嘛,那我们就把一个棋盘,表示成一个 19 × 19 维的向量,在这个向量裡面,如果某一个位置有一个黑子,那那个位置我们就填 1,如果白子我们就填 -1,如果没有子我们就填 0



我们就可以告诉 Network 说,现在棋盘上的盘势长什麼样,我们可以用一个 19 × 19 维的向量,来描述一个棋盘,当然这不一定是要这麼做啦,不一定要黑子是 1 白子是 -1,然后没有子就是 0,这只是一个可能的表示方式而已,你们可以想出其他更神奇的表示方式,总之我们有办法把棋盘上的盘势,用一个向量来描述

把这个向量输到一个 Network 裡面,然后呢,你就可以把下围棋当作一个分类的问题,叫这个 Network 去预测,下一步应该落子的位置落在哪裡最好

所以下围棋,就是一个有 19 × 19 个类别的分类的问题,Network 会 Output 说,在这 19 × 19 个类别裡面,哪一个类别是最好的,应该要选择下一步落子的位置应该在哪裡,那这个问题完全可以用一个,Fully Connected 的 Network 来解决,但是用 **CNN 的效果更好**



為什麼用 CNN 的效果更好呢,首先你完全可以把一个棋盘,看作是一张图片,一个棋盘,可以看作是一个解析度 19 × 19 的图片,一般图片很大,一般图片可能都 100 × 100 的解析度,都是很小的图片了啊,但是棋盘是一个更小的图片,这个图片它的解析度只有 19 × 19,这个图片裡面每一个像素 每一个 Pixel,就代表棋盘上一个可以落子的位置

那 Channel 呢,一般图片的 Channel 就是 RGB 嘛,RGB 代表一个颜色,那棋盘上每一个 Pixel 的 Channel,应该是什麼呢

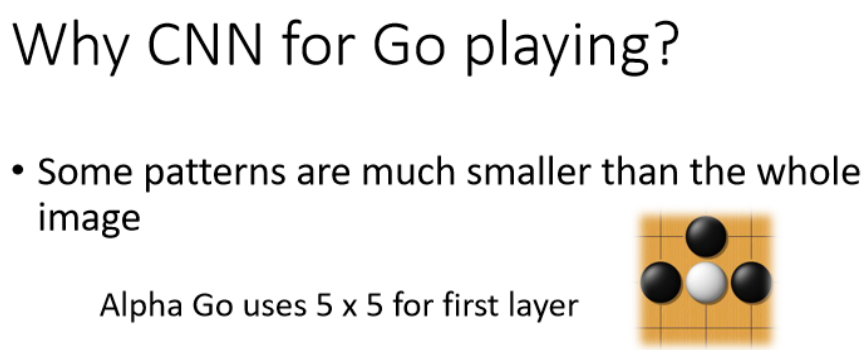
在 AlphaGo 的原始论文裡面它告诉你说,每一个棋盘的位置,**每一个棋盘上的 Pixel,它是用 48 个 Channel 来描述**,也就是说棋盘上的每一个位置,它都用 48 个数字,来描述那个位置发生了什麼事

那至於為什麼是这 48 个,那这个显然是围棋高手设计出来的,那 48 个位置包括,比如说 啊 这个位置是不是要被叫吃了,这个位置旁边有没有顏色不一样的等等,就是这样子描述每一个位置,所以你会用 48 个数字,来描述棋盘上的一个位置,这一个棋盘它就是 19 × 19 的解析度的图片,它的 Channel 就是 48,

但是為什麼 CNN 可以用在下围棋上呢,我们刚才就有强调说 **CNN ,其实并不是你随便用都会好的,它是為影像设计的**,所以如果一个问题,它跟影像没有什麼共通的特性的话,你其实不该用 CNN,所以今天既然在下围棋可以用 CNN,那意味著什麼,那意味著围棋跟影像有共同的特性,什麼样共同的特性呢

Why CNN for Go playing?

我们刚才讲说在影像上的**第一个观察**是,很多重要的 Pattern,你只需要看小范围就知道,下围棋是不是也是一样呢



举例来说这一个 Pattern,你就算**不用看整个棋盘的盘势,你都知道说这边发生了什麼事**,这个就是白子被黑子围住了嘛,那接下来黑子如果放在这边,就可以把白子提走,那白子要放在这边才不会,才可以接这个白子 才不会被提走,那其实在 AlphaGo 裡面啊,它的第一层的 Layer,它的 Filter 的大小就是 5 × 5,所以显然在设计这个 Network 的人觉得说,棋盘上很多重要的 Pattern,也许看 5 × 5 的范围就可以知道了

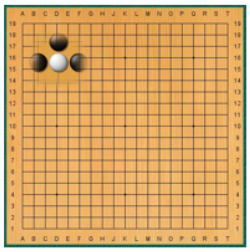
再来是我们说影像上的**第二个观察**是,同样的 Pattern 可能会出现在不同的位置,在下围棋裡面是不是也是一样呢,

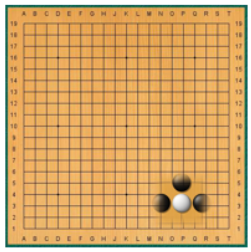
- Some patterns are much smaller than the whole image

Alpha Go uses 5 x 5 for first layer



- The same patterns appear in different regions.





这个叫吃的 Pattern,它可以出现在棋盘上的任何位置,它可以出现在左上角,也可以出现在右下角,所以从这个观点来看,影像跟下围棋有很多共同之处

但是让人想不透的地方是,在做影像的时候我们说我们都会做 Pooling,也就是一张**影像做 Subsampling 以后,并不会影响我们对影像中物件的判读**

但是棋盘是这个样子吗,你可以把棋盘上的奇数行跟偶数列拿掉,还是同一个棋局吗,听起来好像不是对不对,下围棋这麼精细的任务,你随便拿掉一个 Column 拿掉一个 Row,整个棋整个局势就不一样啦,怎麼可能拿掉一个 Row 拿掉一个 Column,还会没有问题呢

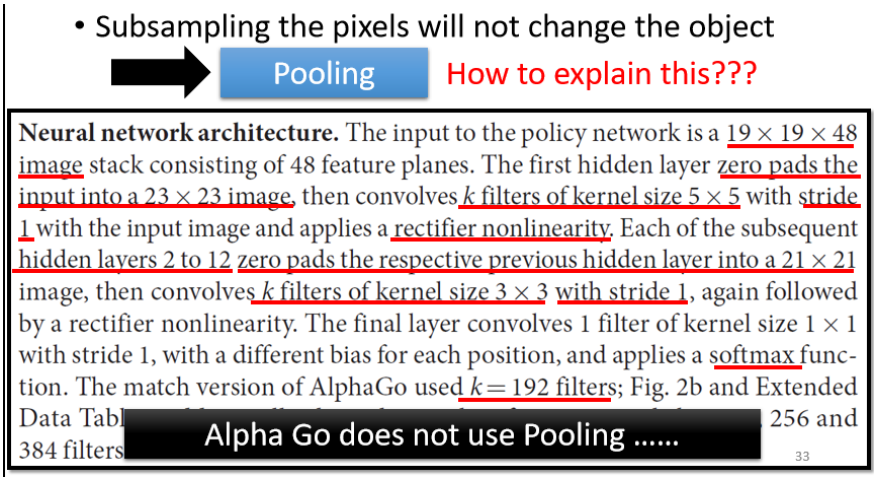
- Subsampling the pixels will not change the object



所以有人就会说,啊 CNN 裡面就是要有 Pooling,然后影像辨识都是用 Pooling,所以 AlphaGo 也一定有用 Pooling,所以代表 AlphaGo 很烂啊,针对 Pooling 这个弱点去攻击它,一定就可以把它打爆,真的是这样吗

可是 AlphaGo 又这麼强,显然它没有这麼显而易见的弱点,所以这个问题就让我有点困扰,好 但后来我就去仔细读了一下,AlphaGo 那篇 Paper

其实 AlphaGo 在 Nature 上的 Paper,其实没有多长,大概就,我记得就五 六页而已,所以其实你一下子就可以看完了,而且在这个文章的正文裡面,甚至没有提它用的网路架构是什麼样子,它没有讲 Network 架构的细节,这个细节在哪裡呢,这个细节在附件裡面,所以我就仔细读了一下附件,看看说 AlphaGo 的网路结构长什麼样子



我们就来看一下,这个附件裡面是怎麼描述,AlphaGo 的类神经网路结构的,它先说呢

- 我们把一个棋盘,看作 19 × 19 × 48 那麼大小的 Image
- 接下来它说它有做 Zero Padding,Padding 这件事我们有讲嘛,就是你的 Filter 如果超出影像的范围就补 0,Zero Padding 就是超出范围就补 0 的意思
- 它说它的 Filter 的大小啊,Kernel Size 就是 Filter 的大小是 5 × 5
- 然后有 k 个 Filter,k 是多少,k 是 192,这当然是试出来的啦,它也试了 128 跟 256 发现 192 最好了,好 这是第一层
- 然后 Stride=1,Stride 是什麼 我们刚才也解释过了
- 然后这边有用了 Rectifier Nonlinearity,这是什麼,这个就是 ReLU 啦,这个就是 ReLU
- 然后在第二层呢,到第 12 层都有做 Zero Padding,然后呢 这个 Kernel Size 都是 3 × 3,一样是 k 个 Filter,也就是每一层都是 192 个 Filter,Stride 呢 一样设 1,就这样叠了很多层以后呢,因為是一个分类的问题
- 最后加上了一个 Softmax

读完以后你有发现什麼玄机吗,发现它==没有用 Pooling==

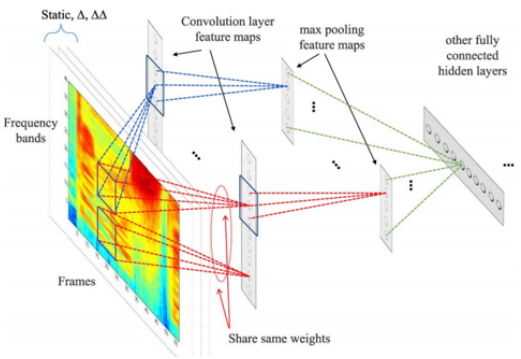
所以这给我们一个很好的例子就是,类神经网路的设计这个==应用之道,存乎一心==

你不要看影像上面都有用 Pooling,就觉得 Pooling 一定是好的,在下围棋的时候就是不适合用 Pooling,所以你要想清楚说,你今天用一个 Network 架构的时候,我这个 Network 的架构到底代表什麼意思,它适不适合用在我现在这个任务上,

More Applications

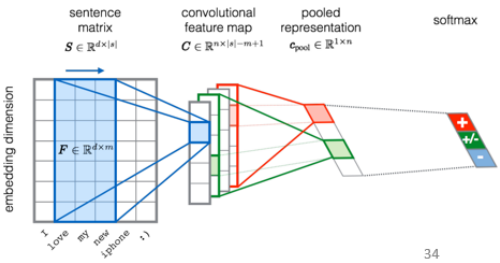
而那 CNN 呢,除了下围棋还有影像以外,欸 近年来也用在语音上,也用在文字处理上,这边我们就不再细讲

More Applications



Speech

https://dl.acm.org/doi/10.1109/TASLP.2014.2339736



Natural Language Processing

https://www.aclweb.org/anthology/S15-2079/

34

但是呢 你如果你真的想把 CNN 用在语音上,用在这个文字处理上,你要仔细看一下文献上的方法

这个在影像 在语音上,在文字上,那个 Receptive Field 的设计啊,这个参数共享的设计啊,跟影像上不是一样的

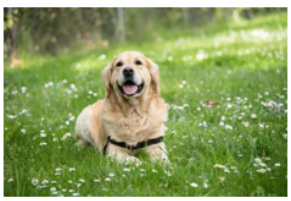
所以你要想清楚,那些 Receptive Field 用在语音,用在文字上的设计跟影像上不是一样,是考虑了语音跟文字的特性以后所设计的

所以你不要以為在影像上的 CNN,直接套到语音上它也 Work,可能是不 Work 的,你要想清楚说影像,语音有什麼样的特性,那你要怎麼设计合适的 Receptive Field,

To learn more

有人会说 CNN,其实 CNN,它没有办法处理影像放大缩小,或者是旋转的问题,怎麼说呢,假设今天你给 CNN 看的狗都是这个大小,它可以辨识说这是一隻狗,当你把这个图片放大的时候,它可以辨识说牠还是一隻狗吗,可能是不行的

- CNN is not invariant to scaling and rotation (we need data augmentation ☺).





Spatial Transformer Layer

https://youtu.be/SoCywZ1hZak
(in Mandarin)

你可能会想说 欸怎麼会不能够辨识呢,这两个形状不是一模一样啊,怎麼放大就不能辨识呢,CNN 这麼笨吗

它就是这麼笨,对它来说这两张图片,虽然这个形状是一模一样的,但是如果你把它拉长成向量的话,它裡面的数值就是不一样的啊,所以对 CNN 来说,虽然你人眼一看觉得它形状很像,但对 CNN 的 Network 来说它是非常不一样

所以事实上,CNN 并不能够处理影像放大缩小,或者是旋转的问题,当它今天在某种大小的影像上,假设你裡面的物件都是比较小的,它在上面学会做影像辨识,你把物件放大它就会整个惨掉

所以 CNN 并没有你想像的那麼强,那就是為什麼在做影像辨识的时候,往往都要做 ==Data Augmentation==,所谓 Data Augmentation 的意思就是说,你把你的训练资料,每张图片都裡面截一小块出来放大,让 CNN 有看过不同大小的 Pattern,然后把图片旋转,让它有看过说,某一个物件旋转以后长什麼样子,CNN 才会做到好的结果

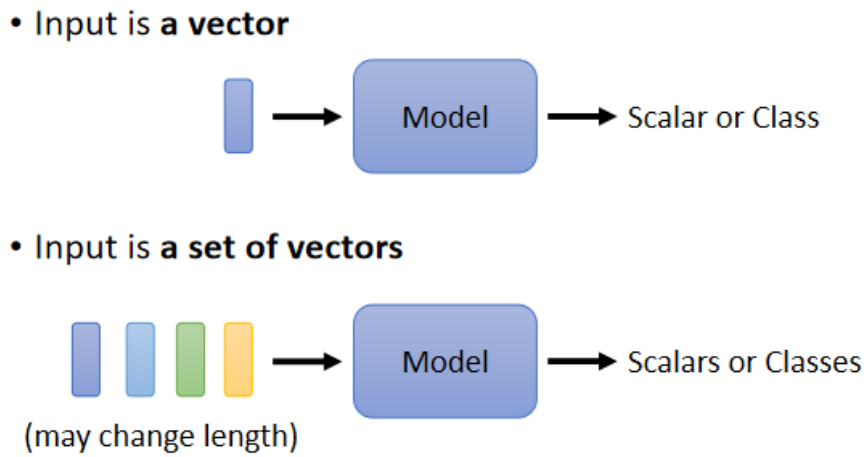
那你说 欸 CNN 这个不能够处理 Scaling,跟 Rotation 的问题啊,那有没有什麼 Network 架构,是可以处理这个问题呢,其实有的,有一个架构叫 **Special Transformer Layer**,我们不会讲它,就把它的这个录影的连结放在这边,给大家参考

Self-attention

CNN以后,我们要讲另外一个常见的Network架构,这个架构叫做Self-Attention,而这个Self-Attention

Sophisticated Input

到目前為止,我们的Network的**Input**都是**一个向量**,不管是在预测这个,YouTube观看人数的问题上啊,还是影像处理上啊,我们的输入都可以看作是一个向量,然后我们的输出,可能是一个**数值**,这个是**Regression**,可能是一个**类别**,这是**Classification**



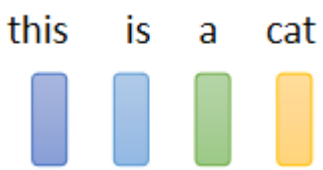
但假设我们遇到更復杂的问题呢,假设我们说**输入是多个向量**,而且这个**输入的向量的数目是会改变的**呢

我们刚才在讲影像辨识的时候,我还特别强调我们假设输入的**影像大小都是一样**的,那现在假设每次我们Model输入的Sequence的数目,Sequence的长度都不一样呢,那这个时候应该要怎麼处理?

Vector Set as Input

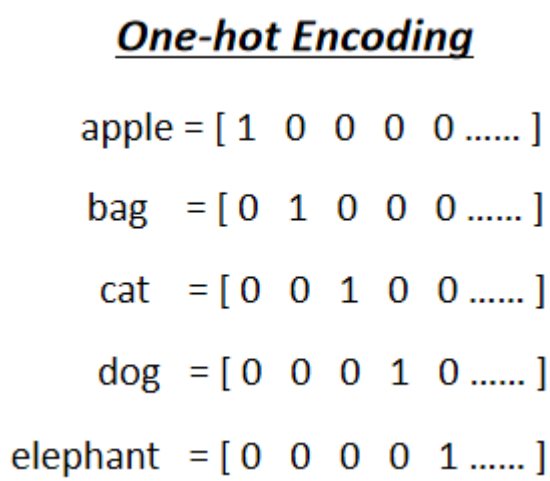
文字处理

假设我们今天要Network的输入是一个句子,每一个句子的长度都不一样,每个句子裡面词汇的数目都不一样



如果我们把一个**句子裡面的每一个词汇,都描述成一个向量**,那我们的Model的**输入**,就会**是一个Vector Set**,而且这个Vector Set的大小,每次都不一样,句子的长度不一样,那你的Vector Set的大小就不一样

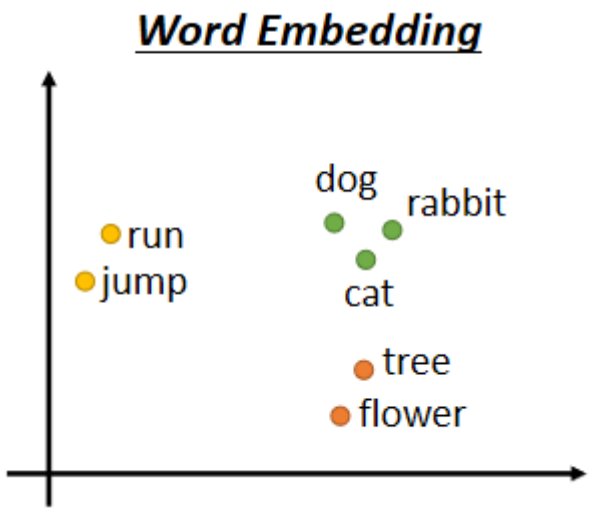
那**怎麼把一个词汇表示成一个向量**,最简单的做法是==One-Hot的Encoding==



你就开一个很长很长的向量,这个向量的长度跟世界上存在的词汇的数目是一样多的,每一个维度对应到一个词汇,Apple就是100,Bag就是010,Cat就是001,以此类推

但是这样子的表示方法有一个非常严重的问题,它假设所有的**词汇彼此之间都是没有关係**的,从这个向量裡面你看不到：Cat跟Dog都是动物所以他们比较接近,Cat跟Apple一个动物一个植物,所以他们比较不相像。这个向量裡面,没有任何语义的资讯

有另外一个方法叫做==Word Embedding==



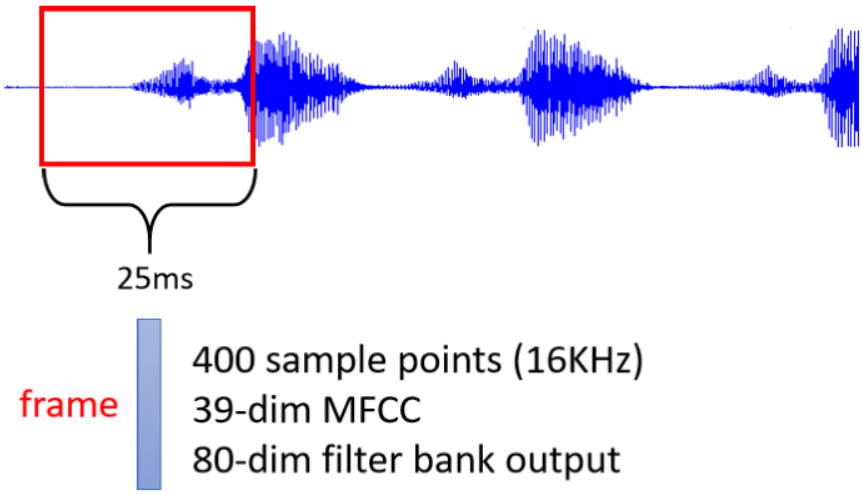
Word Embedding就是,我们会给每一个词汇一个向量,而这个**向量是有语义的资讯的**

如果你把Word Embedding画出来的话,你会发现,所有的动物可能聚集成一团,所有的植物可能聚集成一团,所有的动词可能聚集成一团等等

Word Embedding，如果你有兴趣的话,可以看一下以下的录影 https://youtu.be/X7PH3NuYW0Q,总之你现在在网路上,可以载到一种东西叫做 Word Embedding,这个Word Embedding,会给每一个词汇一个向量,而**一个句子就是一排长度不一的向量**

声音信号

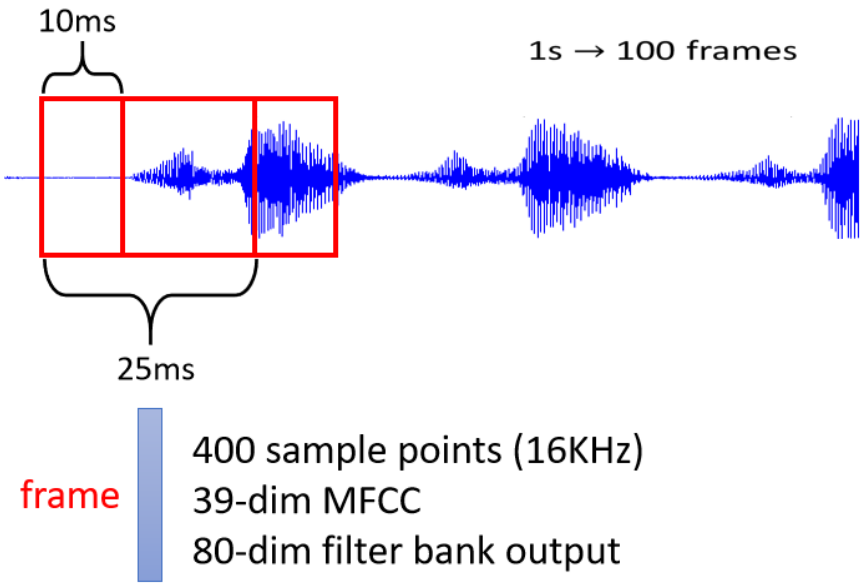
一段声音讯号其实是一排向量,怎麼说呢,我们会把一段声音讯号取一个范围,这个范围叫做一个==Window==



把这个Window裡面的资讯描述成一个向量,这个向量就叫做一个==Frame==,在语音上,我们会把**一个向量叫做一个Frame**,通常这个Window的长度就是25个Millisecond

把这麼一个小段的声音讯号变成一个Frame,变成一个向量就有百百种做法,那这边就不细讲

一小段25个Millisecond裡面的语音讯号,為了要描述一整段的声音讯号,你会把这个**Window往右移一点**,通常移动的大小是10个Millisecond

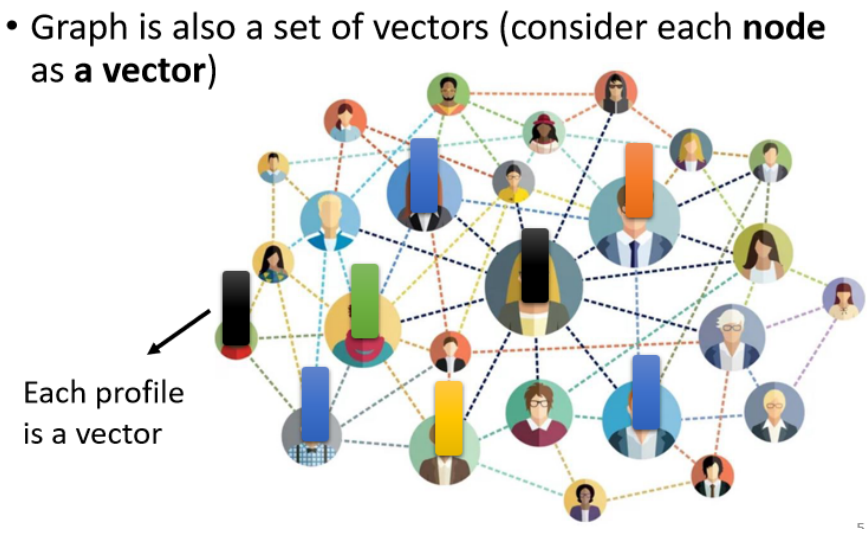


一段声音讯号,你就是用一串向量来表示,而因為每一个Window啊,他们往右移都是移动10个Millisecond,所以一秒鐘的声音讯号有100个向量,所以一分鐘的声音讯号,就有这个100乘以60,就有6000个向量

所以语音其实很复杂的,一小段的声音讯号,它裡面包含的资讯量其实是非常可观的

图

一个Graph 一个图,也是一堆向量,我们知道说Social Network就是一个Graph



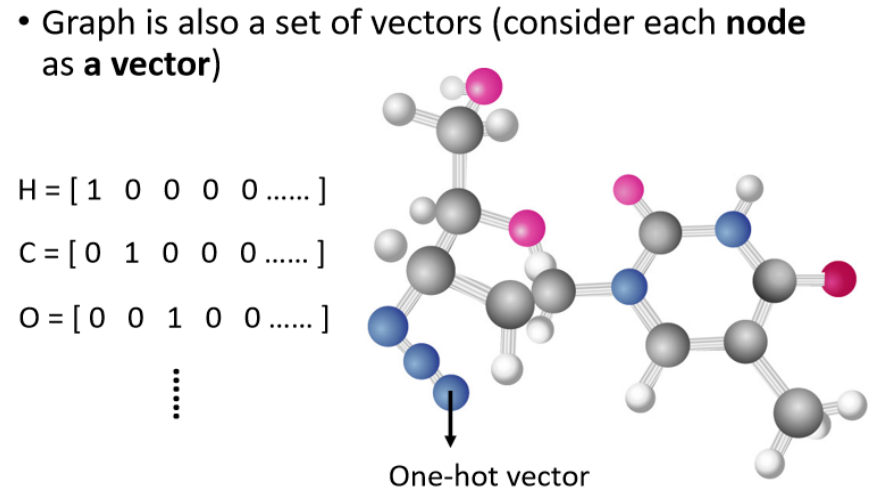
在Social Network上面**每一个节点就是一个人**,然后**节点跟节点之间的edge就是他们两个的关系连接**,比如说是不是朋友等等

而**每一个节点可以看作是一个向量**,你可以拿每一个人的,比如说他的Profile裡面的资讯啊,他的性别啊 他的年龄啊,他的工作啊 他讲过的话啊等等,把这些资讯用一个向量来表示

所以一个Social Network 一个Graph,你也可以看做是一堆的向量所组成的

分子信息

一个分子,它也可以看作是一个Graph



现在Drug Discovery的应用非常地受到重视,尤其是在Covid-19这一段时间,很多人都期待,也许用机器学习,可以在Drug Discovery上面做到什麼突破,那这个时候,你就需要把一个分子,当做是你的模型的输入

一个分子可以看作是一个Graph,分子上面的每一个球,也就是**每一个原子，可以表述成一个向量**

一个**原子可以用One-Hot Vector**来表示,氢就是1000,碳就是0100,然后这个氧就是0010,所以一个分子就是一个Graph,它就是一堆向量。

What is the output?

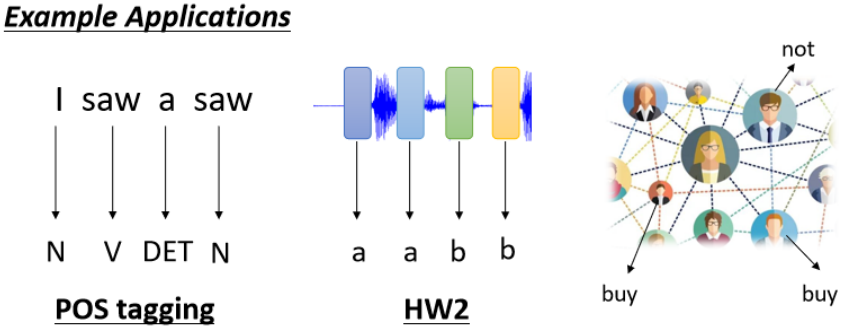
我们刚才已经看说输入是一堆向量,它可以是文字,可以是语音,可以是Graph,那这个时候,我们有可能有什麼样的输出呢,有三种可能性

1. 每一个向量都有一个对应的Label

当你的模型,看到输入是四个向量的时候,它就要输出四个Label,而每一个Label,它可能是一个数值,那就是Regression的问题,如果每个Label是一个Class,那就是一个Classification的问题

- Each vector has a label.





- 举例来说 在文字处理上,假设你今天要做的是==POS Tagging==,POS Tagging就是词性标註,你要让机器自动决定每一个词汇 它是什麼样的词性,它是名词 还是动词 还是形容词等等

  这个任务啊,其实并没有很容易,举例来说,你现在看到一个句子,I saw a saw

  这并不是打错,并不是“我看一个看”,而是“我看到一个锯子”,这个第二个saw当名词用的时候,它是锯子， 那所以机器要知道,第一个saw是个动词,第二个saw虽然它也是个saw,但它是名词,但是每一个输入的词汇,都要有一个对应的输出的词性

  这个任务就是,输入跟输出的长度是一样的Case,这个就是属於第一个类型的输出

- 那如果是语音的话,你可以想想看我们作业二就是这样子的任务

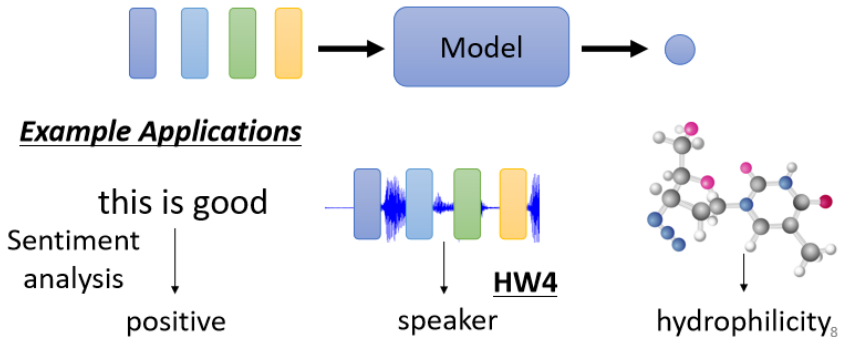
  虽然我们作业二,没有给大家一个完整的Sequence,我们是把每一个每一个每一个Vector分开给大家了,但是串起来就是一段声音讯号裡面,有一串Vector,每一个Vector你都要决定,它是哪一个Phonetic， 这是一个语音辨识的简化版

- 或者是如果是Social Network的话,就是给一个Graph

  你的Model要决定每一个节点,它有什麼样的特性,比如说他会不会买某一个商品,这样我们才知道说,要不要推荐某一个商品给他,

  所以以上就是举输入跟输出 数目一样的例子

2. 一整个Sequence,只需要输出一个Label



- 举例来说,如果是文字的话,我们就说Sentiment Analysis

  Sentiment Analysis就是给机器看一段话,它要**决定说这段话是正面的还是负面的**

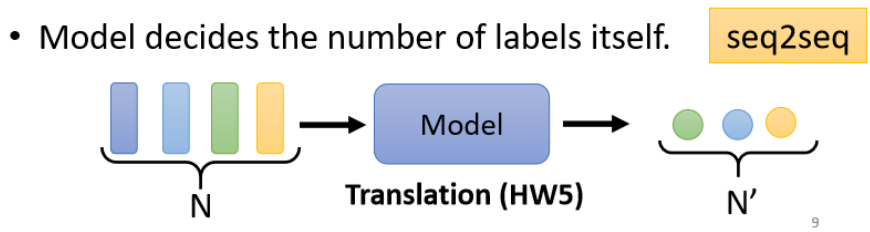
  那你可以想像说这种应用很有用,假设你的公司开发了一个產品,这个產品上线了,你想要知道网友的评价怎麼样,但是你又不可能一则一则网友的留言都去分析,那也许你就可以用这种,Sentiment Analysis的技术,让机器自动去判读说,当一则贴文裡面有提到某个產品的时候,它是正面的 还是负面的,那你就可以知道你的產品,在网友心中的评价怎麼样,这个是Sentiment Analysis给一整个句子,只需要一个Label,那Positive或Negative,那这个就是第二类的输出

- 那如果是语音的例子的话呢,在作业四裡面我们会做语者辨认,机器要听一段声音,然后决定他是谁讲的

- 或者是如果是Graph的话呢,今天你可能想要给一个分子,然后要预测说这个分子,比如说它有没有毒性,或者是它的亲水性如何,那这就是给一个Graph 输出一个Label

3. 机器要自己决定,应该要输出多少个Label

我们不知道应该输出多少个Label,机器要自己决定,应该要输出多少个Label,可能你输入是N个向量,输出可能是N'个Label,為什麼是N',机器自己决定



这种任务又叫做==sequence to sequence==的任务,那我们在作业五会有sequence to sequence的作业,所以这个之后我们还会再讲

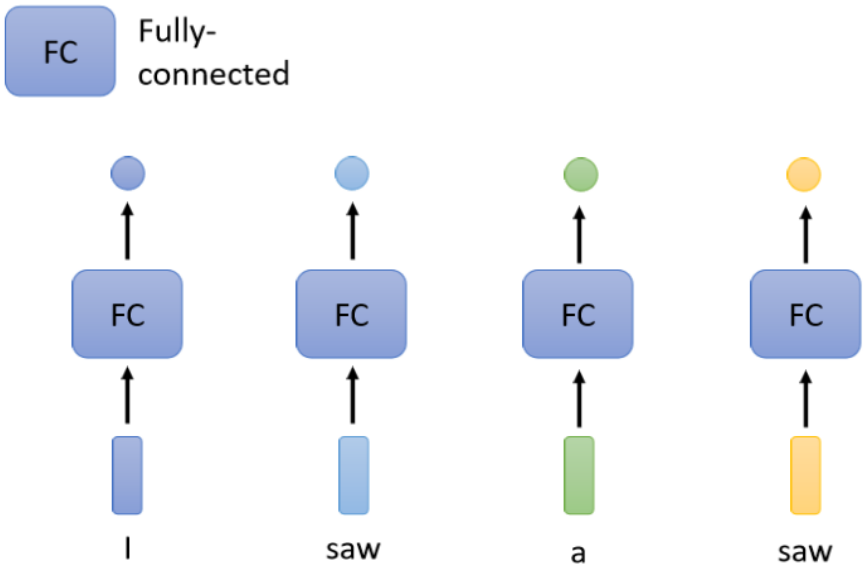
- 翻译就是sequence to sequence的任务,因為输入输出是不同的语言,它们的词汇的数目本来就不会一样多
- 或者是语音辨识也是,真正的语音辨识也是一个sequence to sequence的任务,输入一句话,然后输出一段文字,这也是一个sequence to sequence的任务

第二种类型有作业四,感兴趣可以去看看作业四的程式，那因為上课时间有限,所以上课,我们今天就先只讲第一个类型,也就是输入跟输出数目一样多的状况

Sequence Labeling

那这种输入跟输出数目一样多的状况又叫做==Sequence Labeling==,你要给Sequence裡面的每一个向量,都给它一个Label,那要怎麼解Sequence Labeling的问题呢

那直觉的想法就是我们就拿个**Fully-Connected的Network**



然后虽然这个输入是一个Sequence,但我们就各个击破,不要管它是不是一个Sequence,把每一个向量,分别输入到Fully-Connected的Network裡面

然后Fully-Connected的Network就会给我们输出,那现在看看,你要做的是Regression还是Classification,產生正确的对应的输出,就结束了,

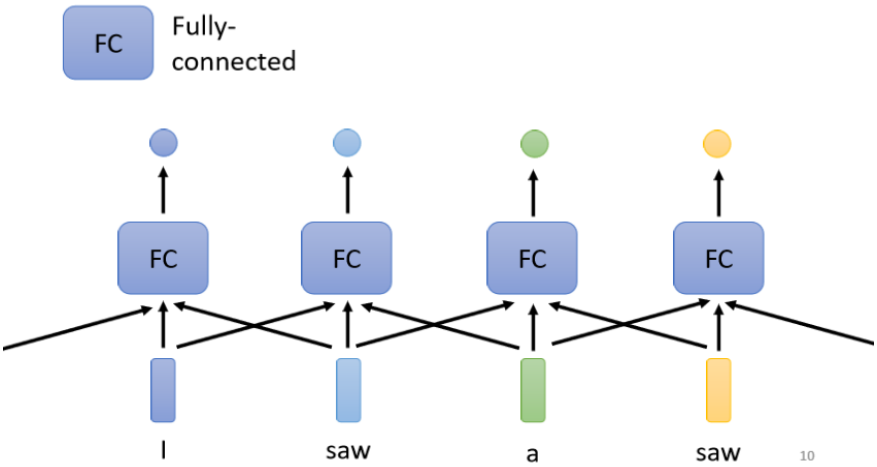
那这麼做显然有**非常大的瑕疵**,假设今天是,词性标记的问题,你给机器一个句子,I saw a saw,对Fully-Connected Network来说,**后面这一个saw跟前面这个saw完全一模一样**,它们是同一个词汇啊

既然Fully-Connected的Network**输入同一个词汇,它没有理由输出不同的东西**

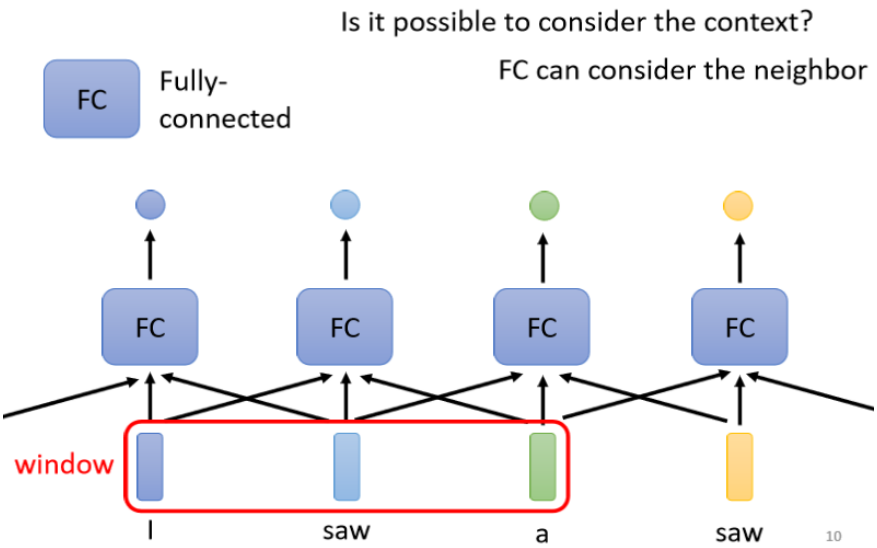
但实际上,你期待第一个saw要输出动词,第二个saw要输出名词,但对Network来说它不可能做到,因為这两个saw 明明是一模一样的,你叫它一个要输出动词,一个要输出名词,它会非常地困惑,完全不知道要怎麼处理

所以怎麼办,有没有可能**让Fully-Connected的Network,考虑更多的,比如说上下文的Context的资讯**呢

这是有可能的,你就**把前后几个向量都串起来,一起丢到Fully-Connected的Network就结束了**



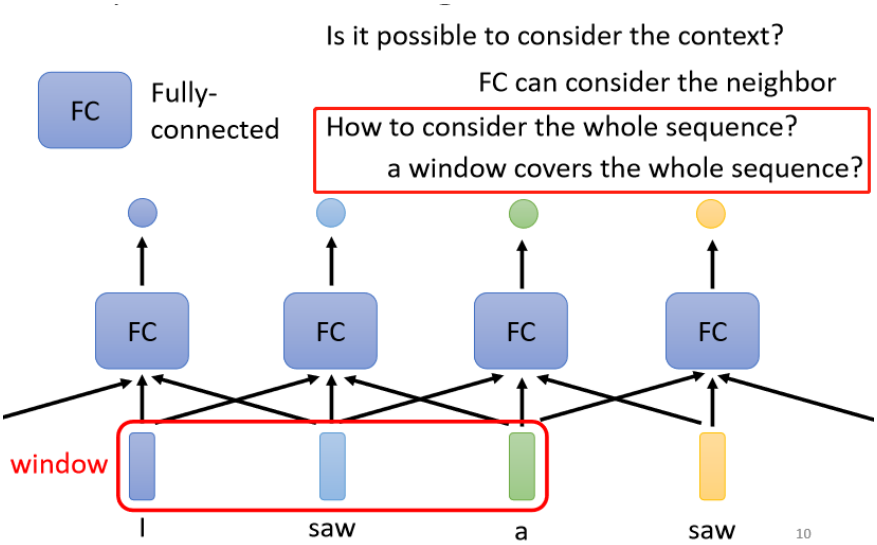
在作业二裡面,我们不是只看一个Frame,去判断这个Frame属於哪一个Phonetic,也就属於哪一个音标,而是看这个Frame的前面五个加后面五个,也就总共看十一个Frame,来决定它是哪一个音标



所以我们可以给Fully-Connected的Network,一整个Window的资讯,让它可以考虑一些上下文的,跟我现在要考虑的这个向量,相邻的其他向量的资讯

但是**这样子的方法还是有极限**,作业二就算是给你Sequence的资讯,你考虑整个Sequence,你可能也很难再做的更好啦,作业二考虑前后五个Frame,其实就可以得到很不错的结果了,所以你要过Strong Baseline,重点并不在於考虑整个Sequence,你就不需要往那个方向想了,用助教现有给你的Data,你就可以轻易的过Strong Baseline,

但是真正的问题,但是如果今天我们有某一个任务,不是考虑一个Window就可以解决的,而是要**考虑一整个Sequence才能够解决**的话,那要怎麼办呢



那有人可能会想说这个很容易，我就**把Window开大一点啊,大到可以把整个Sequence盖住就结束了**

但是，今天**Sequence的长度是有长有短的**,我们刚才有说,我们输入给我们的Model的Sequence的长度,每次可能都不一样

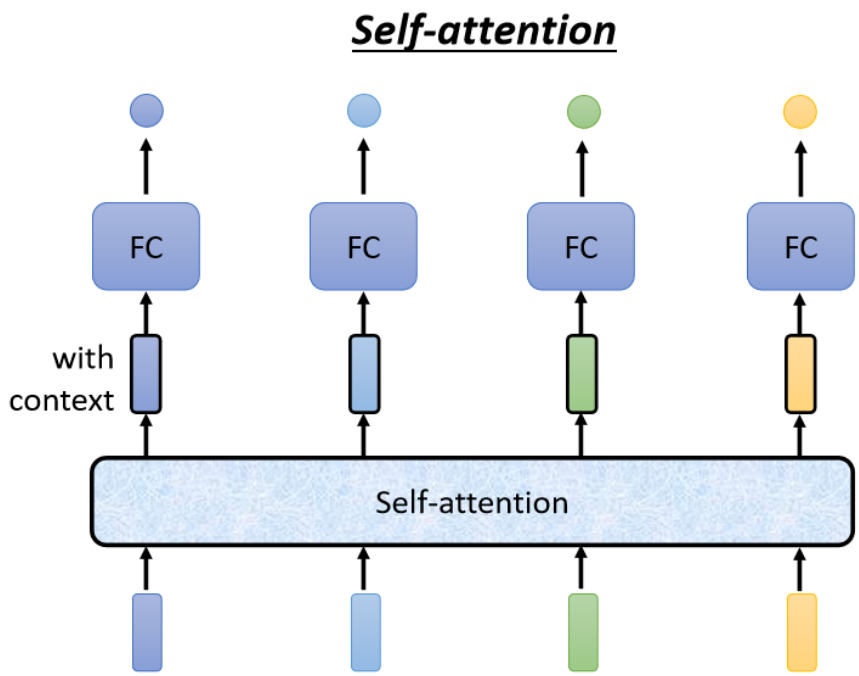
如果你今天说我真的要开一个Window,把整个Sequence盖住,那你可能要**统计一下你的训练资料**,然后看看你的训练资料裡面,最长的Sequence有多长,然后开一个Window比最长的Sequence还要长,你才有可能把整个Sequence盖住

但是你开一个这麼大的Window,意味著说你的Fully-Connected的Network,它需要非常多的参数,那可能不只**运算量很大,可能还容易Overfitting**

所以有没有更好的方法,来考虑整个Input Sequence的资讯呢,这就要用到我们接下来要跟大家介绍的,==Self-Attention==这个技术

Self-Attention

Self-Attention的运作方式就是,**Self-Attention会吃一整个Sequence的资讯**

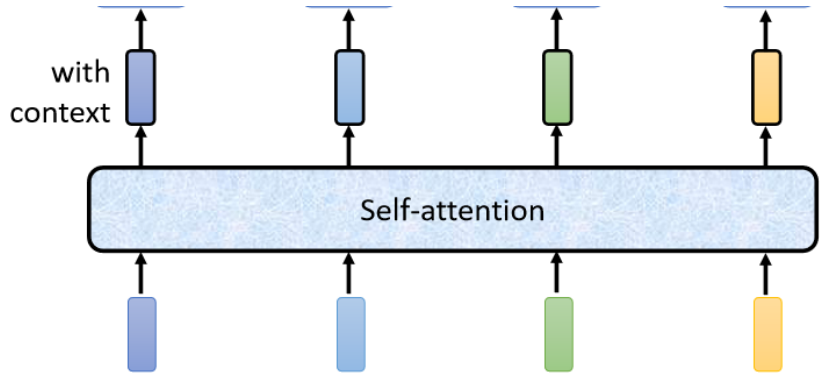


然后你Input几个Vector,它就输出几个Vector,比如说你这边Input一个深蓝色的Vector,这边就给你一个另外一个Vector

这边给个浅蓝色,它就给你另外一个Vector,这边输入4个Vector,它就Output 4个Vector

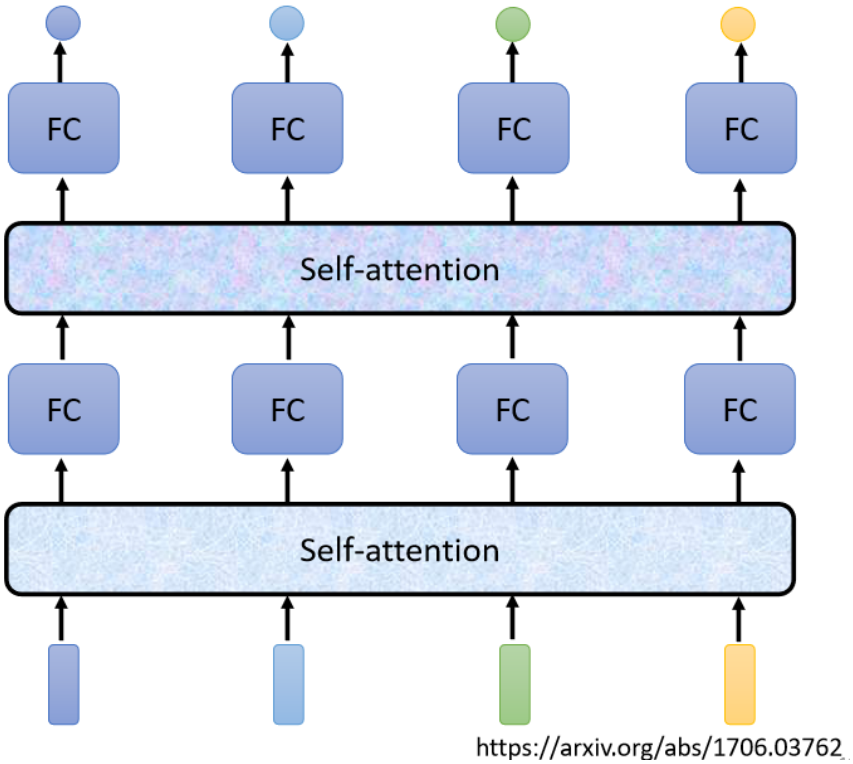
那这4个Vector有什麼特别的地方呢,**这4个Vector,他们都是考虑一整个Sequence以后才得到的**,那等一下我会讲说Self-Attention,怎麼考虑一整个Sequence的资讯

所以这边每一个向量,我们特别给它一个黑色的框框代表说它不是一个普通的向量



如此一来你这个Fully-Connected的Network,它就不是只考虑一个非常小的范围,或一个小的Window,而是考虑整个Sequence的资讯,再来决定现在应该要输出什麼样的结果，这个就是Self-Attention。

Self-Attention不是只能用一次,你可以叠加很多次



可以Self-Attention的输出,通过Fully-Connected Network以后,再做一次Self-Attention,Fully-Connected的Network,再过一次Self-Attention,再重新考虑一次整个Input Sequence的资讯,再丢到另外一个Fully-Connected的Network,最后再得到最终的结果

所以**可以把Fully-Connected的Network,跟Self-Attention交替使用**

- Self-Attention处理整个Sequence的资讯
- Fully-Connected的Network,专注於处理某一个位置的资讯
- 再用Self-Attention,再把整个Sequence资讯再处理一次
- 然后交替使用Self-Attention跟Fully-Connected

有关Self-Attention,最知名的相关的文章,就是《Attention is all you need》.那在这篇Paper裡面呢,Google提出了==Transformer==这样的Network架构,那Transformer就是变形金刚,所以提到这个Network的时候呢,我们就会有变形金刚这个形象



Transformer我们今天还不会讲到,但我们之后会讲到,Transformer裡面一个最重要的Module就是Self-Attention,它就是变形金刚的火种源

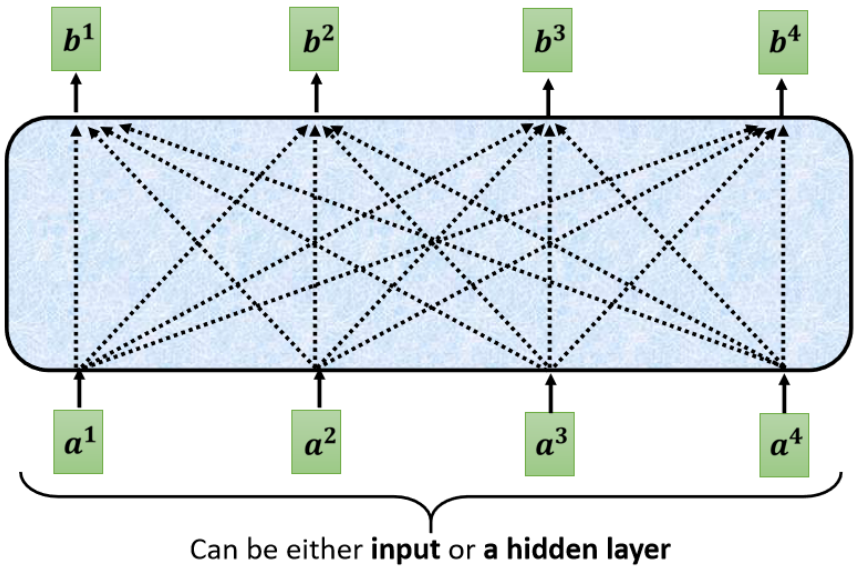
那这篇Paper最厉害的地方,就是它有一个**霸气的名字Attention is all you need.**

那其实像Self-Attention这样的架构,最早我并不会说它是出现在《Attention is all you need》这样的Paper,因為其实很多更早的Paper,就有提出过类似的架构,只是不见得叫做Self-Attention,比如说叫做Self-Matching,或者是叫别的名字,不过呢是Attention is all you need.这篇Paper,把Self-Attention这个Module,把它发扬光大

那Self-Attention是怎麼运作的呢

Self-Attention过程

Self-Attention的Input,它就是一串的Vector,那**这个Vector可能是你整个Network的Input,它也可能是某个Hidden Layer的Output**,所以我们这边不是用x来表示它,

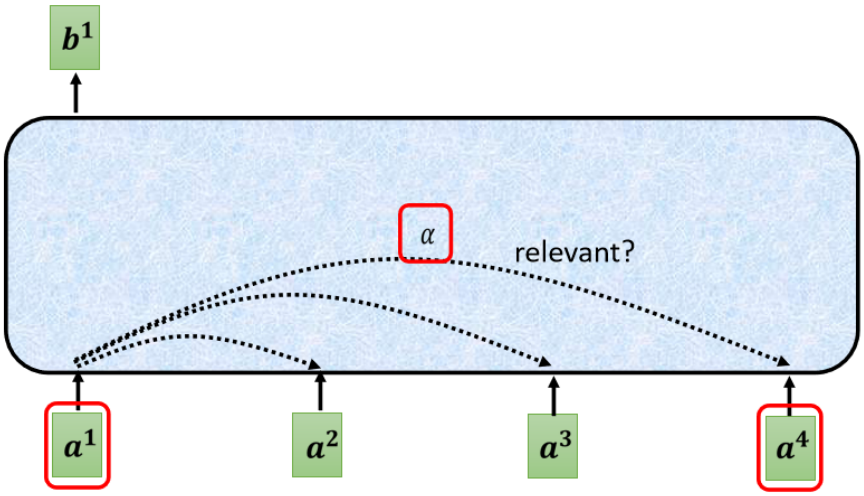


我们用a来表示它，代表它有可能是前面已经做过一些处理,它是某个Hidden Layer的Output,那Input一排a这个向量以后,Self-Attention要Output另外一排b这个向量

那这**每一个b都是考虑了所有的a以后才生成出来的**,所以这边刻意画了非常非常多的箭头,告诉你$b^1 $考虑了$a^1$到$a^4$產生的,$b^2$考虑$a^1$到$a^4$產生的,$b^3 b^4$也是一样,考虑整个input的sequence,才產生出来的

那接下来呢就是要跟大家说明,**怎麼產生b^1这个向量**,那你知道怎麼產生b^1这个向量以后,你就知道怎麼產生剩下$b^1 b^2 b^3 b^4$剩下的向量

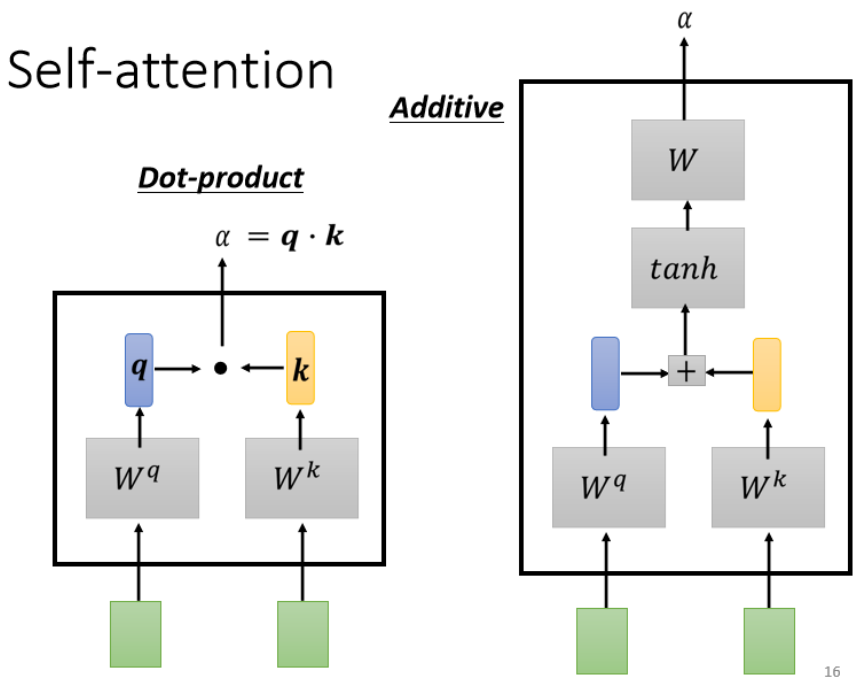
这里有一个**特别的机制,这个机制是根据a^1这个向量,找出整个很长的sequence裡面,到底哪些部分是重要的,哪些部分跟判断a^1是哪一个label是有关係的,哪些部分是我们要决定a^1的class,决定a^1的regression数值的时候,所需要用到的资讯**



Find the relevant vectors in a sequence

每一个向量跟a^1的关联的程度,用一个数值叫α来表示

这个self-attention的module,**怎麼自动决定两个向量之间的关联性**呢,你给它两个向量a^1跟a^4,它怎麼决定a^1跟a^4有多相关,然后给它一个数值α呢,那这边呢你就需要一个**计算attention的模组**



这个计算attention的模组,就是拿**两个向量作為输入**,然后它就直接输出α那个数值,

计算这个α的数值有各种不同的做法

- 比较常见的做法呢,叫做用==dot product==,**输入的这两个向量分别乘上两个不同的矩阵**,左边这个向量乘上W^q这个矩阵得到矩阵q,右边这个向量乘上W^k这个矩阵得到矩阵k

  再把q跟k做dot product,就是把他们做element-wise 的相乘,再全部加起来以后就得到一个 scalar,这个scalar就是α,这是一种计算α的方式

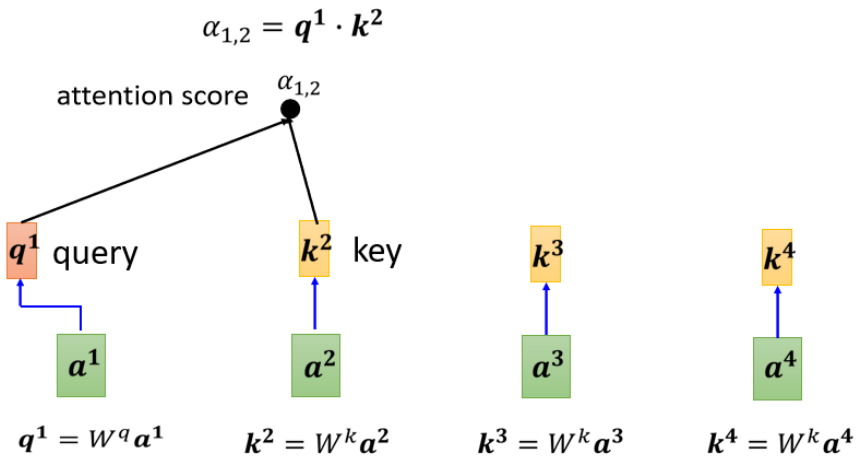
- 有另外一个叫做==Additive==的计算方式,它的计算方法就是,把同样这两个向量通过W^q W^k,得到q跟k,那我们不是把它做Dot-Product,是把它这个串起来,然后丢到这个过一个Activation Function

  然后再通过一个Transform,然后得到α

总之有非常多不同的方法,可以计算Attention,可以计算这个α的数值,可以计算这个关联的程度

但是在接下来的讨论裡面,我们都**只用左边这个方法**,这也是今日最常用的方法,也**是用在Transformer裡面的方法**

那你就要把这边的a^1去跟这边的$a^2 a^3 a^4$,分别都去计算他们之间的关联性,也就是计算他们之间的α

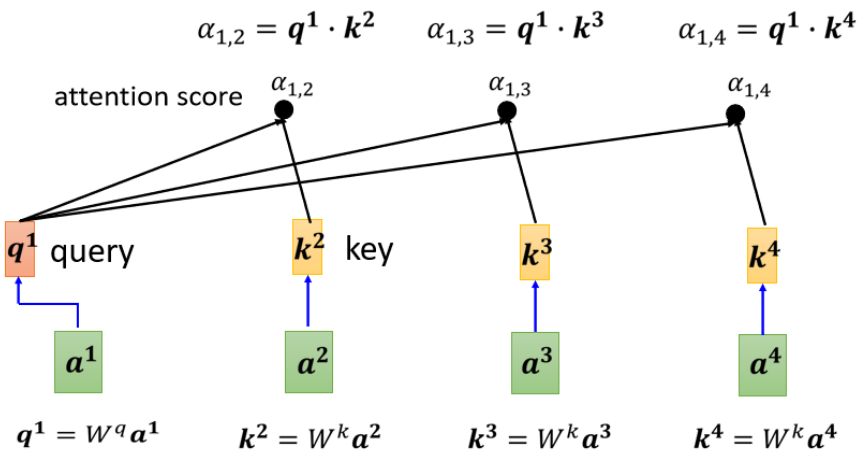


你把a^1乘上$W^q $得到$q^1$,那这个q有一个名字,我们叫做==Query==,它就像是你搜寻引擎的时候,去搜寻相关文章的问题,就像搜寻相关文章的关键字,所以这边叫做Query

然后接下来呢,$a^2 a^3 a^4$你都要去把它乘上W^k,得到k这个Vector,k这个Vector叫做==Key==,那你把这个**Query q1,跟这个Key k2,算Inner-Product就得到α**

我们这边用$α_{1,2}$来代表说,Query是1提供的,Key是2提供的时候,这个1跟2他们之间的关联性,这个α这个关联性叫做==Attention的Score==,叫做Attention的分数,

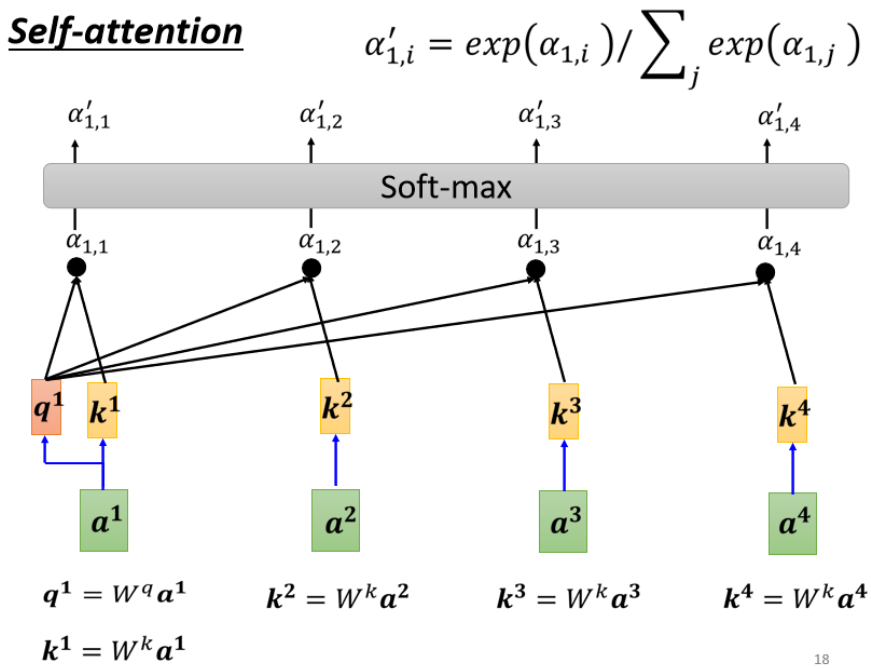
接下来也要跟$a^3 a^4$来计算



把$a3$乘上W^k,得到另外一个Key也就是k^3,a^4乘上W^k得到k^4,然后你再把k^3这个Key,跟q^1这个Query做Inner-Product,得到1跟3之间的关联性,得到1跟3的Attention,你把k^4跟q^1做Dot-Product,得到$α{1,4}$,得到1跟4之间的关联性

其实一般在实作时候,**q^1也会跟自己算关联性**,自己跟自己计算关联性这件事情有多重要,你可以自己在做作业的时候试试看,看这件事情的影响大不大了

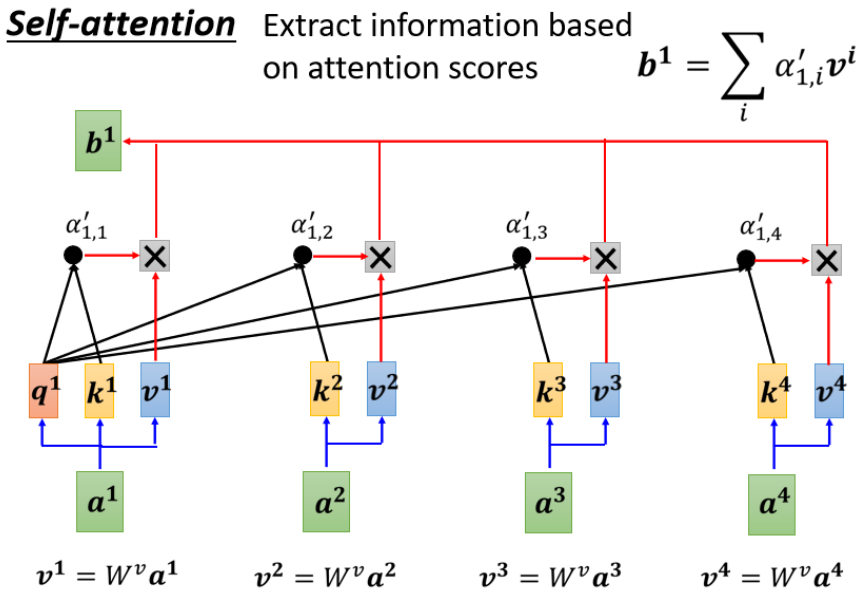
计算出,a1跟每一个向量的关联性以后,接下来这边会**接入一个Soft-Max**



这个Soft-Max跟分类的时候的那个Soft-Max是一模一样的,所以Soft-Max的输出就是一排α,所以本来有一排α,通过Soft-Max就得到$α'$

这边你**不一定要用Soft-Max,用别的替代也没问题**,比如说有人尝试过说做个ReLU,这边通通做个ReLU,那结果发现还比Soft-Max好一点,所以这边你不一定要用Soft-Max,这边你要用什麼Activation Function都行,你高兴就好,你可以试试看,那Soft-Max是最常见的,那你可以自己试试看,看能不能试出比Soft-Max更好的结果

接下来得到这个$α'$以后,我们就要根据这个$α'$去抽取出这个Sequence裡面重要的资讯,根据这个α我们已经知道说,哪些向量跟a^1是最有关係的,怎麼抽取重要的资讯呢,



- 首先把$a^1到$a^4$这边每一个向量,乘上$W^v $得到新的向量,这边分别就是用$v^1 v^2 v^3 v^4$来表示
- 接下来把这边的$v^1到$v^4$,每一个向量都去乘上Attention的分数,都去乘上$α'$
- 然后再把它加起来,得到b^1

$$ b^1=\sum*i**alpha*'{1,i}v^i

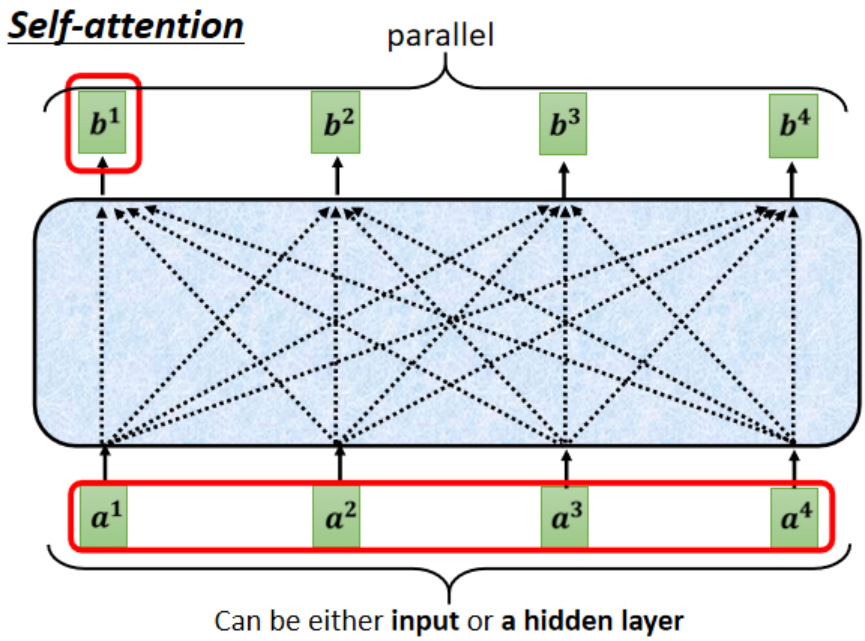
$$

如果某一个向量它得到的分数越高,比如说如果$a^1跟$a^2$的关联性很强,这个$α'$得到的值很大,那我们今天在做Weighted Sum以后,得到的b^1的值,就可能会比较接近v^2

所以**谁的那个Attention的分数最大,谁的那个v就会Dominant你抽出来的结果**

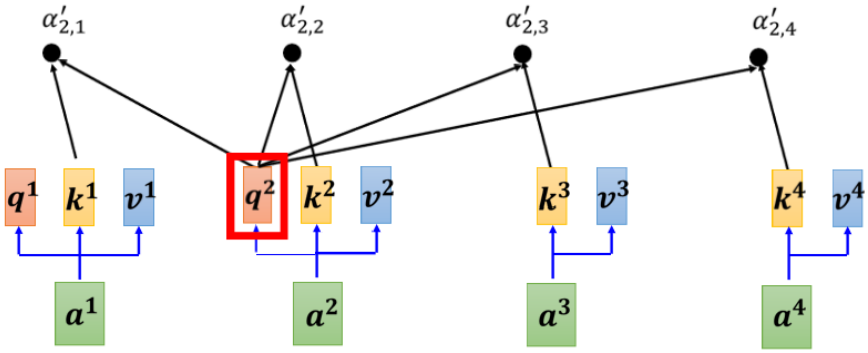
所以这边呢我们就讲了怎麼从一整个Sequence 得到b^1

Self-attention_P2



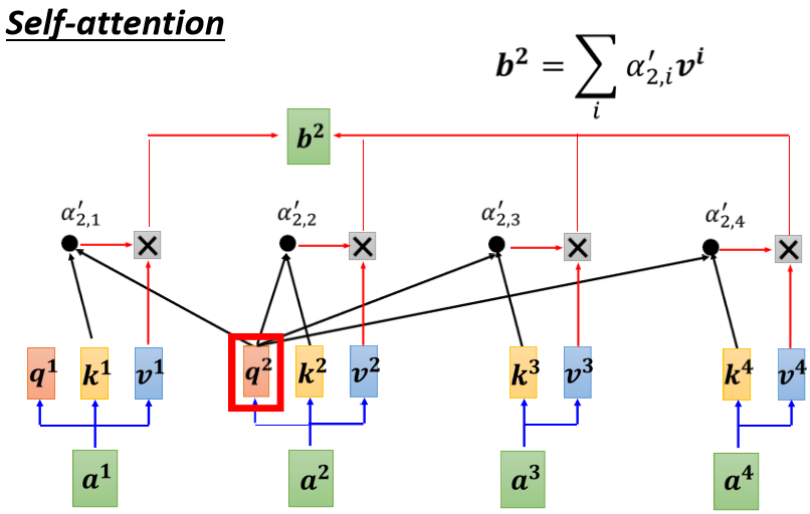
从这一排 vector 得到 b^1,跟从这一排 vector 得到 b^2,它的操作是一模一样的.要强调一点是,这边的 b^1 到 b^4,它们并**不需要依序產生**,它们是一次同时被计算出来的

怎麼计算这个 b^2? 我们现在的主角,就变成 a^2



- 把 a^2 乘上一个 matrix,变成 q^2
- 然后接下来根据 q^2,去对a^1到 a^4 这四个位置,都去计算 attention 的 score
  - 把 q^2 跟 k^1 做个这个 dot product
  - 把 q^2 跟 k^2 也做个 dot product
  - 把 q^2 跟 k^3 也做 dot product
  - 把 q^2 跟 k^4 也做 dot product,得到四个分数
- 得到这四个分数以后,可能还会做一个 normalization,比如说 softmax,然后得到最后的 attention 的 score,$α'*{2,1}* \space α'{2,2} \space α'*{2,3}* \space α'{2,4}$那我们这边用 $α'$表示经过 normalization 以后的attention score

- 接下来拿这四个数值,分别乘上 $v^1 \space v^2 \space v^3 \space v^4$



  - 把 $α'_{2,1}$乘上 v^1
  - 把 $α'_{2,2}$ 乘上 v^2
  - 把 $α'_{2,3}$ 乘上 v^3
  - 把 $α'_{2,4}$ 乘上 v^4,然后全部加起来就是 $ b^2$

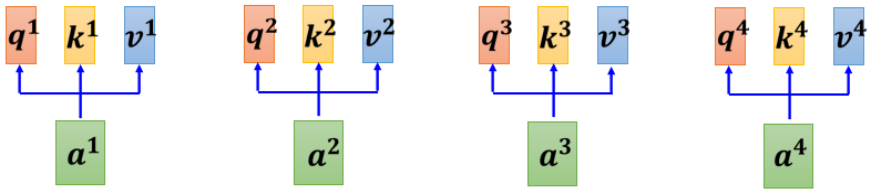
    $$ b^2=\sum_i α'_{2,i}v^i $$

同理就可以,由 a^3 乘一个 transform 得到 q^3,然后就计算 b^3,从 a^4 乘一个 transform 得到 q^4,就计算 b^4,以上说的是 Self-attention 它运作的过程

矩阵的角度

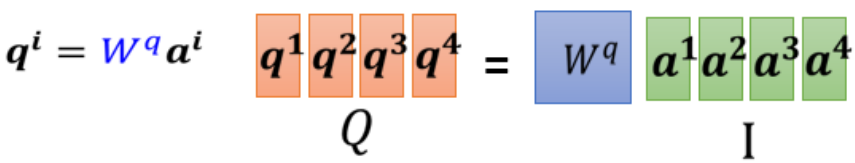
接下来我们从矩阵乘法的角度,再重新讲一次我们刚才讲的,Self-attention 是怎麼运作的

我们现在已经知道每一个 a 都產生 q k v



如果要用矩阵运算表示这个操作的话,是什麼样子呢

我们每一个 a,都乘上一个矩阵,我们这边用 W^q 来表示它,得到 q^i,每一个 a 都要乘上 W^q,得到q^i,**这些不同的 a 你可以把它合起来,当作一个矩阵来看待**



一样$a^2\space a^3\space a^4 $也都乘上 W^q 得到$q^2 q^3 $跟 q^4,那你可以**把 a1 到 a4 拼起来**,看作是一个矩阵,这个矩阵我们用 I 来表示，这个矩阵的四个 column 就是 a^1 到 a^4

I 乘上 W^q 就得到另外一个矩阵,我们用 Q 来表示它,这个 Q 就是把 q^1 到 q^4 这四个 vector 拼起来,就是 Q 的四个 column

所以我们从 a^1 到 a^4,得到 q^1 到 q^4这个操作,其实就是**把 I 这个矩阵,乘上另外一个矩阵 W^q，得到矩阵Q**。I 这个矩阵它裡面的 column就是我们 Self-attention 的 input是 a^1 到 a^4；**W^q其实是 network 的参数,它是等一下会被learn出来的**； Q 的四个 column,就是 q^1 到 q^4

接下来產生 k 跟 v 的操作跟 q 是一模一样的

$$q^i = W^q a^i \quad \underbrace{q^1 q^2 q^3 q^4}_{Q} = W^q \underbrace{a^1 a^2 a^3 a^4}_{I}$$

$$k^i = W^k a^i \quad \underbrace{k^1 k^2 k^3 k^4}_{K} = W^k \underbrace{a^1 a^2 a^3 a^4}_{I}$$

$$v^i = W^v a^i \quad \underbrace{v^1 v^2 v^3 v^4}_{V} = W^v \underbrace{a^1 a^2 a^3 a^4}_{I}$$

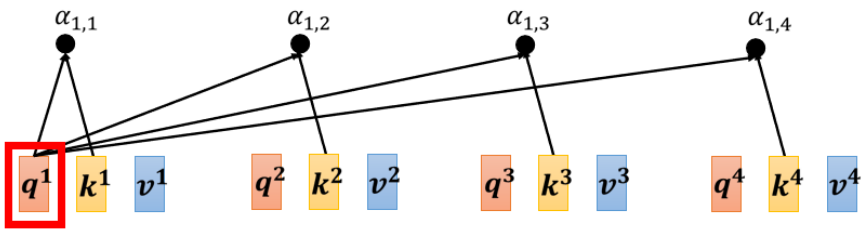
所以每一个 a 得到 q k v ,其实就是把输入的这个,vector sequence 乘上三个不同的矩阵,你就得到了 q,得到了 k,跟得到了 v

下一步是,每一个 q 都会去跟每一个 k,去计算这个 inner product,去**得到这个 attention 的分数**

那得到 attention 分数这一件事情,如果从矩阵操作的角度来看,它在做什麼样的事情呢

$$\alpha_{1,1} = k^1 q^1 \quad \alpha_{1,2} = k^2 q^1$$

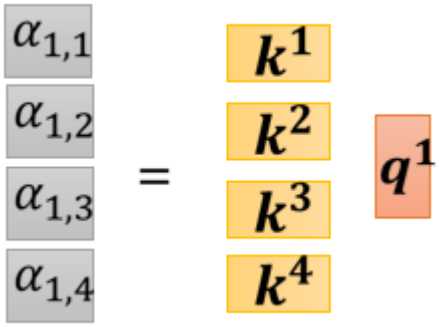
$$\alpha_{1,3} = k^3 q^1 \quad \alpha_{1,4} = k^4 q^1$$



你就是把 q^1 跟 k^1 做 inner product,得到 $α*{1,1}$,*所以 $α*{1,1}$就是 q^1跟k^1 的 inner product,那这边我就把这个,k^1它背后的这个向量,把它画成比较宽一点代表说它是 transpose

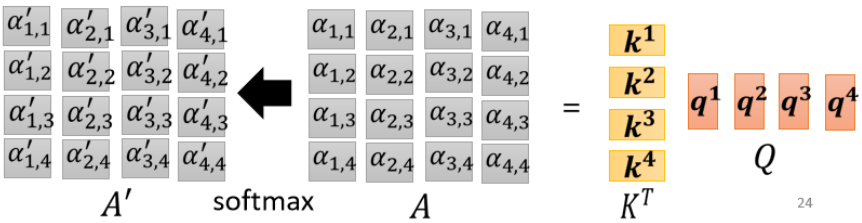
同理 $α{1,2}$ 就是 q^1 跟 k^2,做 inner product, $α{1,3}$ 就是 q^1 跟 k^3 做 inner product,这个 $α_{1,4}$ 就是 q^1 跟 k^4 做 inner product

那这个四个步骤的操作,你其实可以把它拼起来,看作是**矩阵跟向量相乘**



这四个动作,你可以看作是我们**把 k^1 到 k^4 拼起来,当作是一个矩阵的四个 row**

那我们刚才讲过说,我们不只是 q^1,要对k^1 到 k^4 计算 attention,q^2,q^3,q^4也要对 k^1 到 k^4 计算 attention,操作其实都是一模一样的



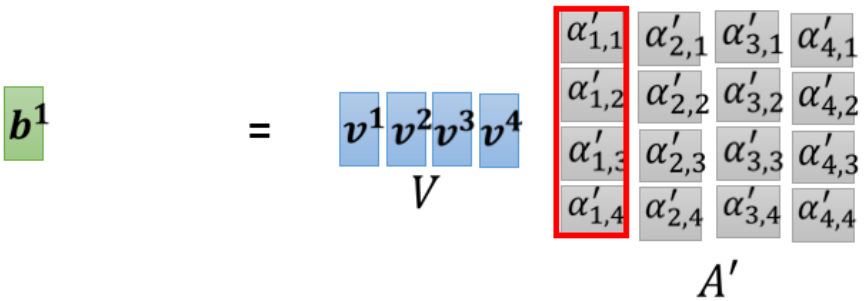
所以这些 **attention 的分数可以看作是两个矩阵的相乘**,一个矩阵它的 row,就是 k^1 到 k^4,另外一个矩阵它的 column

我们会在 attention 的分数,**做一下 normalization**,比如说你会做 softmax,你会对这边的每一个 column,每一个 column 做 softmax,让每一个 column 裡面的值相加是 1

之前有讲过说 其实这边做 **softmax不是唯一的选项**,你完全可以选择其他的操作,比如说 ReLU 之类的,那其实得到的结果也不会比较差,通过了 softmax 以后,它得到的值有点不一样了,所以我们用 A',来表示通过 softmax 以后的结果

我们已经计算出 $A' $

那我们把这个v^1 到 v^4乘上这边的 α 以后,就可以得到 b



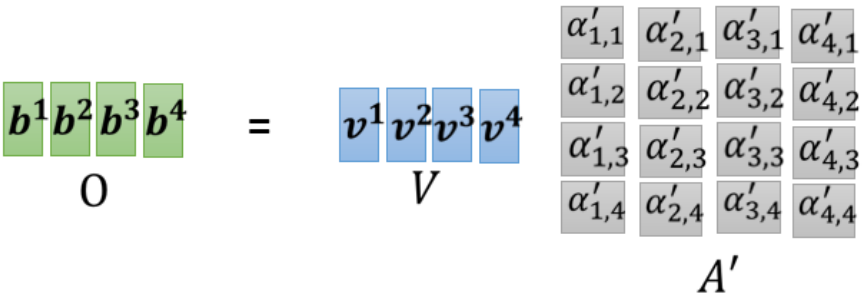
你就把v^1 到 v^4 拼起来,你**把 v^1 到 v^4当成是V 这个矩阵的四个 column**,把它拼起来,然后接下来你把 v 乘上,A' 的第一个 column 以后,你得到的结果就是 b^1

如果你熟悉线性代数的话,你知道说把这个 A' 乘上 V,就是把 A'的第一个 column,乘上 V 这一个矩阵,你会得到你 output 矩阵的第一个 column

而把 A 的第一个 column乘上 V 这个矩阵做的事情,其实就是把 V 这个矩阵裡面的每一个 column,**根据第 A' 这个矩阵裡面的每一个 column 裡面每一个 element,做 weighted sum**,那就得到 b^1

那就是这边的操作,把 v^1 到 v^4 乘上 weight,全部加起来得到 b^1,

如果你是用矩阵操作的角度来看它,就是把$ A'$ 的第一个 column 乘上 V,就得到 b^1,然后接下来就是以此类推

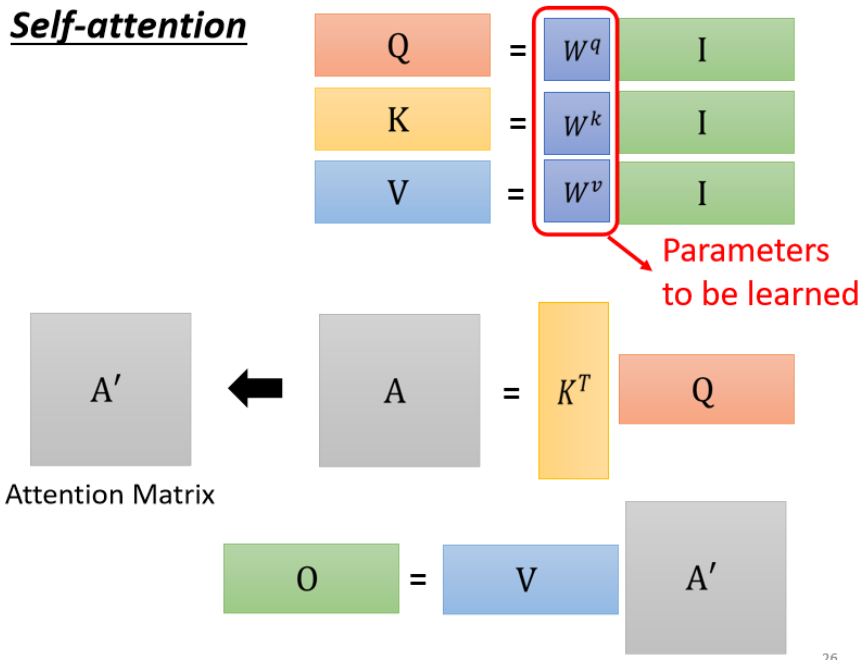


就是以此类推,把 A' 的第二个 column 乘上 V,就得到 b^2,A' 的第三个 column 乘上 V 就得到 b^3,A' 的最后一个 column 乘上 V,就得到 b^4

所以我们等於就是把 A' 这个矩阵,乘上 V 这个矩阵,得到 O 这个矩阵,O 这个矩阵裡面的每一个 column,就是 Self-attention 的输出,也就是 b^1 到 b^4,

所以其实整个 Self-attention,我们在讲操作的时候,我们在最开始的时候 跟你讲的时候我们讲说,我们先產生了 q k v,然后再根据这个 q 去找出相关的位置,然后再对 v 做 weighted sum,其实这一串操作,**就是一连串矩阵的乘法而已**

我们再复习一下我们刚才看到的矩阵乘法



- I 是 Self-attention 的 input,Self-attention 的 input 是一排的vector,这排 vector 拼起来当作矩阵的 column,就是 I
- 这个 input 分别乘上三个矩阵,W^q W^k 跟$ W^v$,得到 Q K V
- 这三个矩阵,接下来 Q 乘上 K 的 transpose,得到 A 这个矩阵,A 的矩阵你可能会做一些处理,得到 A',那有时候我们会把这个 A',叫做 ==Attention Matrix==，**生成Q矩阵就是为了得到Attention的score**
- 然后接下来你把 A' 再乘上 V,就得到 O,O 就是 Self-attention 这个 layer 的输出,**生成V是为了计算最后的b，也就是矩阵O**

所以 Self-attention 输入是 I,输出是 O,那你会发现说虽然是叫 attention,但是**其实 Self-attention layer 裡面,唯一需要学的参数,就只有 W^q W^k 跟$ W^v$ 而已,只有W^q W^k 跟$ W^v$是未知的**,是需要透过我们的训练资料把它找出来的

但是其他的操作都没有未知的参数,都是我们人為设定好的,都不需要透过 training data 找出来,那这整个就是 Self-attention 的操作,从 I 到 O 就是做了 Self-attention

Multi-head Self-attention

Self-attention 有一个进阶的版本,叫做 ==Multi-head Self-attention==, Multi-head Self-attention,其实今天的使用是非常地广泛的

在作业 4 裡面,助教原来的 code 4 有,Multi-head Self-attention,它的 head 的数目是设成 2,那刚才助教有给你提示说,把 head 的数目改少一点 改成 1,其实就可以过 medium baseline

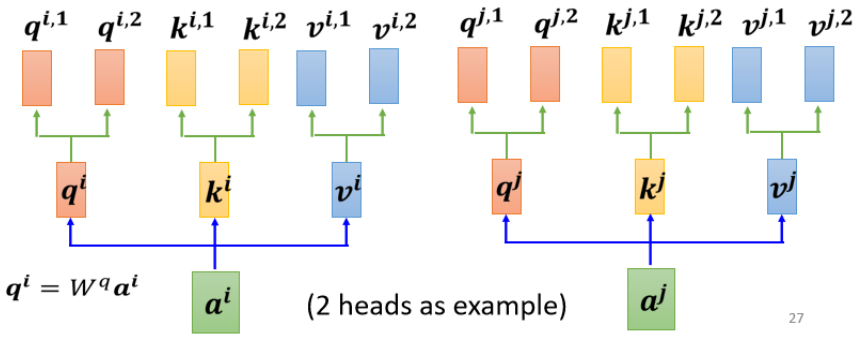
但并不代表所有的任务,都适合用比较少的 head,有一些任务,比如说翻译,比如说语音辨识,其实用比较多的 head,你反而可以得到比较好的结果

至於**需要用多少的 head,这个又是另外一个 hyperparameter**,也是你需要调的

那為什麼我们会需要比较多的 head 呢,你可以想成说相关这件事情

我们在做这个 Self-attention 的时候,我们就是用 q 去找相关的 k,但是**==相关==这件事情有很多种不同的形式**,有很多种不同的定义,所以也许我们不能只有一个 q,我们应该要有多个 q,**不同的 q 负责不同种类的相关性**

所以假设你要做 Multi-head Self-attention 的话,你会怎麼操作呢?



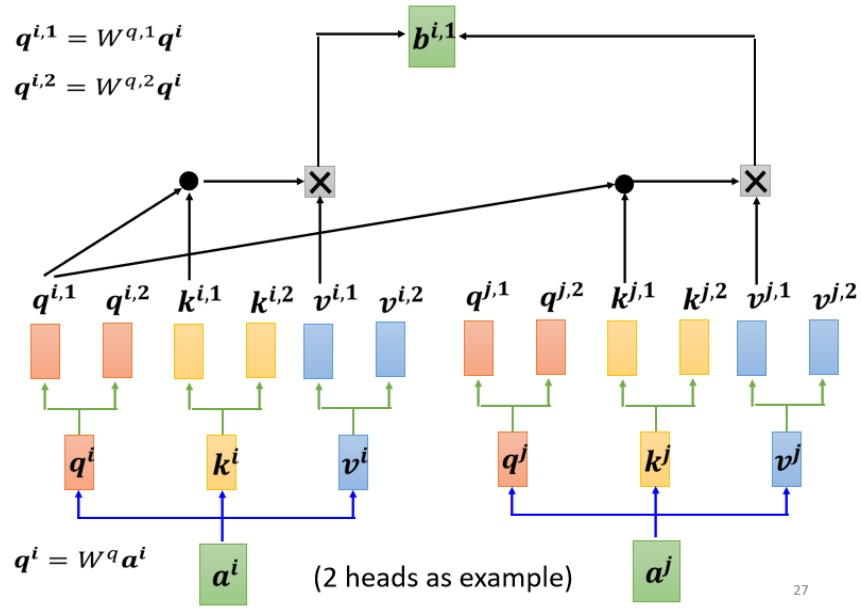
- 先把 a 乘上一个矩阵得到 q
- 再把 q 乘上另外两个矩阵,分别得到 q^1 跟 q^2,那这边还有 这边是用两个上标,i 代表的是位置,然后这个 1 跟 2 代表是,这个位置的第几个 q,所以这边有 $q^{i,1}$ 跟 $q^{i,2}$,代表说我们有两个 head

我们认為这个问题,裡面有两种不同的相关性,是我们需要產生两种不同的 head,来找两种不同的相关性

既然 q 有两个,那 k 也就要有两个,那 v 也就要有两个,从 q 得到 $q^1 q^2$,从 k 得到 $k^1 k^2$,从 v 得到 $v^1 v^2$,那其实就是把 q 把 k 把 v,分别乘上两个矩阵,得到这个不同的 head,就这样子而已,

对另外一个位置,也做一样的事情

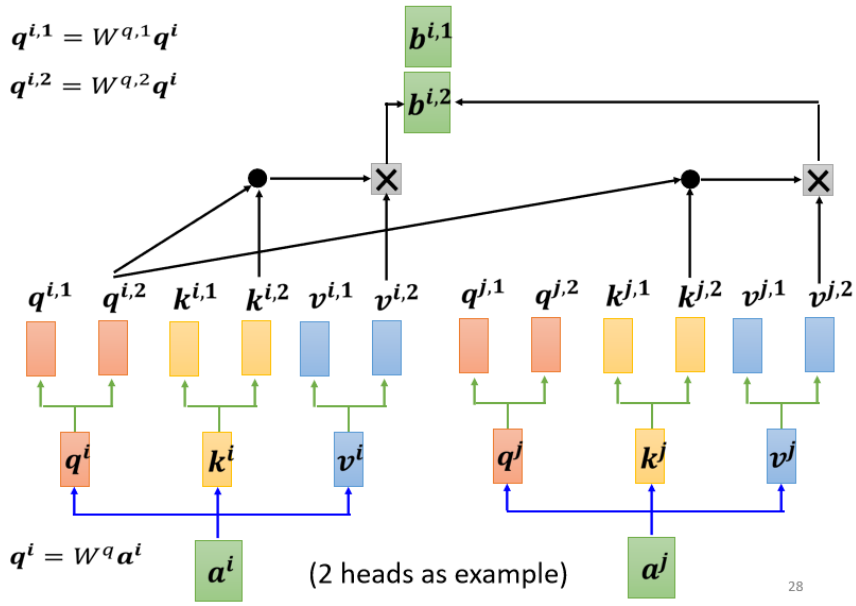
只是现在q^1,它在算这个 attention 的分数的时候,它就不要管那个 k^2 了



- 所以 $q_{i,1}$ 就跟 $k^{i,1}$ 算 attention
- $q_{i,1}$ 就跟 $k^{j,1}$ 算 attention,也就是算这个 dot product,然后得到这个 attention 的分数

- 然后今天在做 weighted sum 的时候,也不要管 v^2 了,看 $V^{i,1}$ 跟 $v^{j,1}$ 就好,所以你把 attention 的分数乘 $v^{i,1}$,把 attention 的分数乘 $v^{j,1}$
- 然后接下来就得到 $b^{i,1}$

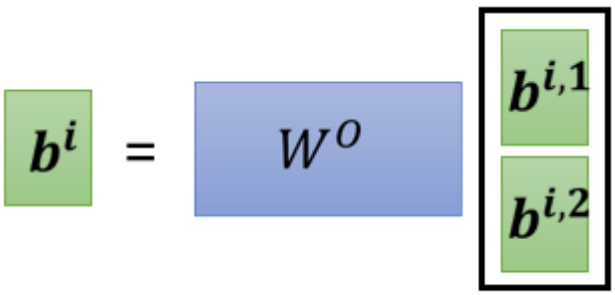
这边只用了其中一个 head,那你会用另外一个 head,也做一模一样的事情



所以 q^2 只对 k^2 做 attention,它们在做 weighted sum 的时候,只对 v^2 做 weighted sum,然后接下来你就得到 $b^{i,2}$

如果你有多个 head,有 8 个 head 有 16 个 head,那也是一样的操作,那这边是用两个 head 来当作例子,来给你看看有两个 head 的时候,是怎麼操作的,现在得到 $b^{i,1}$ 跟 $b^{i,2}$

然后接下来你可能会把 $b^{i,1}$ 跟 $b^{i,2}$,把它接起来,然后再通过一个 transform



也就是再乘上一个矩阵,然后得到 bi,然后再送到下一层去,那这个就是 Multi-head attention,一个这个 Self-attention 的变形

Positional Encoding

No position information in self-attention

那讲到目前为止,你会发现说 Self-attention 的这个 layer,它少了一个也许很重要的资讯,这个资讯是**位置的资讯**

对一个 Self-attention layer 而言,每一个 input,它是出现在 sequence 的最前面,还是最后面,它是完全没有这个资讯的

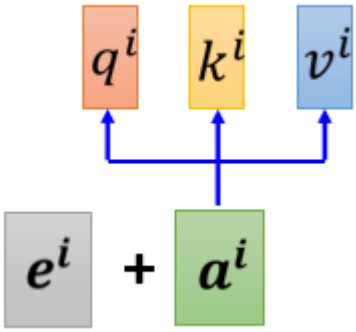
对 Self-attention 而言,**位置 1 跟位置 2 跟位置 3 跟位置 4,完全没有任何差别,这四个位置的操作其实是一模一样**,对它来说 q1 到跟 q4 的距离,并没有特别远,1 跟 4 的距离并没有特别远,2 跟 3 的距离也没有特别近

对它来说就是天涯若比邻,所有的位置之间的距离都是一样的,没有任何一个位置距离比较远,也没有任何位置距离比较近,也没有谁在整个 sequence 的最前面,也没有谁在整个 sequence 的最后面

但是这样子设计可能会有一些问题,因为有时候位置的资讯也许很重要,举例来说,我们在做这个 POS tagging,就是词性标记的时候,也许你知道说**动词比较不容易出现在句首**,所以如果我们知道说,某一个词汇它是放在句首的,那它是动词的可能性可能就比较低,这样子的位置的资讯往往也是有用的

Each positon has a unique positional vector e^i

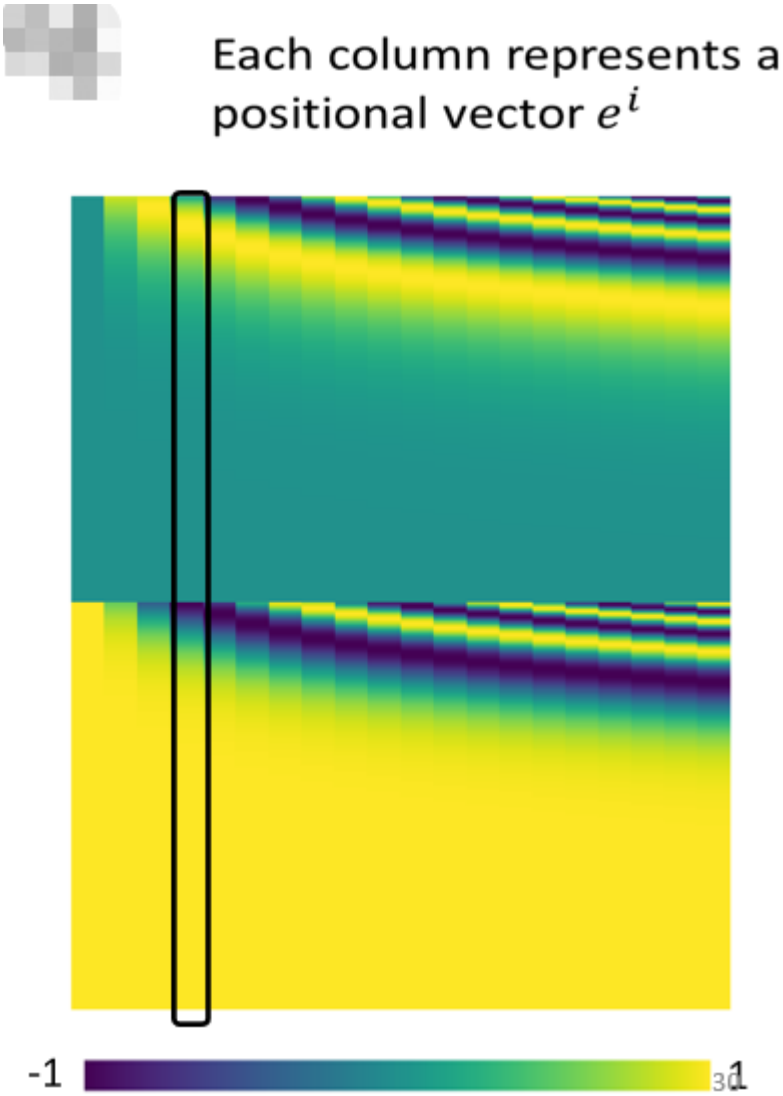
可是在我们到目前为止,讲的 Self-attention 的操作裡面,根本就没有位置的资讯,所以怎麼办呢,所以你做 Self-attention 的时候,如果你觉得位置的资讯是一个重要的事情,那你可以把位置的资讯把它塞进去,怎麼把位置的资讯塞进去呢,这边就要用到一个叫做,==positional encoding== 的技术



你为每一个位置设定一个 vector,叫做 positional vector,这边**用 e^i 来表示,上标 i 代表是位置,每一个不同的位置**,就有不同的 vector,就是 e^1 是一个 vector,e^2 是一个vector,e^{128} 是一个vector,不同的位置都有一个它专属的 e,然后把这个 e 加到 a^i 上面,就结束了

就是告诉你的 Self-attention,位置的资讯,如果它看到说 a^i 好像有被加上 $ e^i$,它就知道说现在出现的位置,应该是在 i 这个位置

最早的这个 transformer,就 Attention Is All You Need 那篇 paper 裡面,它用的 $ e^i$长的是这个样子



这边这个图上面,每一个 column 就代表一个 e,第一个位置就是 e^1,第二个位置就是 e^2,第三个位置就是 e^3,以此类推

所以它就是把这边这个向量,放在第一个位置,把这个向量加到第二个位置的 a上,把这个向量加到第三个位置的 a 上,以此类推,每一个位置都有一个专属的 e,希望透过给每一个位置不同的 e,你的 model 在处理这个 input 的时候,它可以知道现在的 input,它的位置的资讯是什麼样子

Hand-crafted or Learned from data

这样子的 positional vector,它是 handcrafted 的,也就是它是人设的,那人设的这个 vector 有很多问题,就假设我现在在定这个 vector 的时候,只定到 128,那我现在 sequence 的长度,如果是 129 怎麼办呢

不过在最早的那个,Attention Is All You Need paper裡面,没有这个问题,**它 vector 是透过某一个规则所產生的**,透过一个很神奇的sin cos 的 function 所產生的

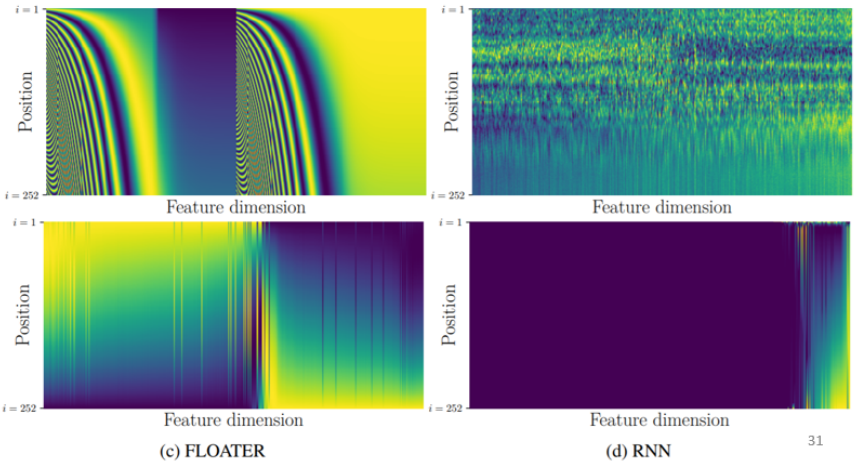
其实你不一定要这麼產生, **positional encoding仍然是一个尚待研究的问题**,你可以创造自己新的方法,或甚至 positional encoding,是可以根据资料学出来的

那有关 positional encoding,你可以再参考一下文献,这个是一个尚待研究的问题,比如说我这边引用了一篇,这个是去年放在 arxiv 上的论文,所以可以想见这其实都是很新的论文

https://arxiv.org/abs/2003.09229

Table 1. Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal (Vaswani et al., 2017)	✓	✗	✓
Embedding (Devlin et al., 2018)	✗	✓	✗
Relative (Shaw et al., 2018)	✗	✓	✓
This paper	✓	✓	✓



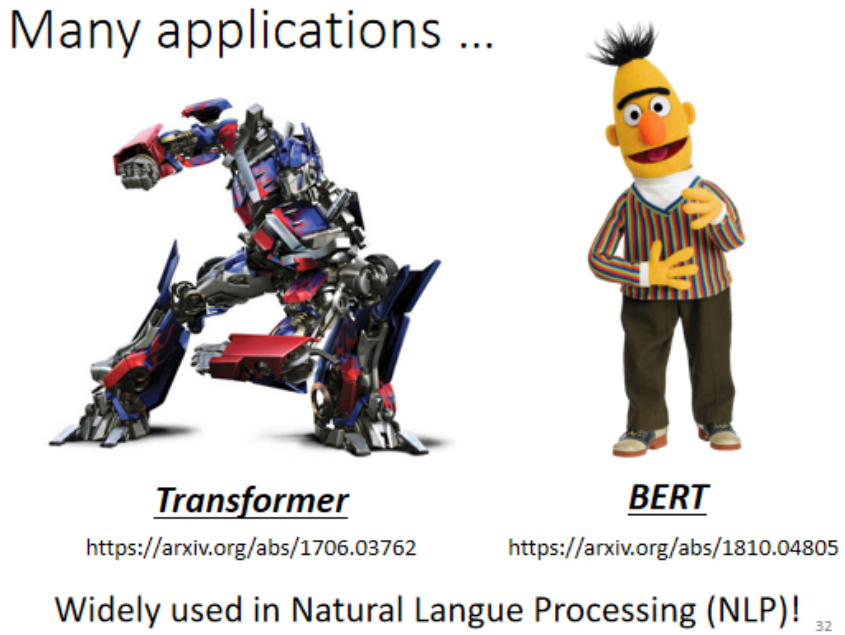
裡面就是比较了跟提出了,新的 positional encoding

- 比如说这个是最早的 positional encoding,它是用一个神奇的 sin function 所產生的
- 那如果你的 positional encoding,你把 positional encoding 裡面的数值,当作 network 参数的一部分,直接 learn 出来,看起来是这个样子的,这个图是那个横著看的,它是横著看的,它是每一个 row,代表一个 position,好 所以这个是这个最原始的,用 sin function 產生的,这个是 learn 出来的
- 它裡面又有神奇的做法,比如说这个,这个是用 RNN 生出来的,positional encording 是用 RNN 出来的,这篇 paper 提出来的叫做 FLOATER,是用个神奇的 network 生出来的,

总之你有各式各样不同的方法,来產生 positional encoding,那目前我们还不知道哪一种方法最好,这是一个尚待研究中的问题,所以你不用纠结说,為什麼 Sinusoidal 最好,**你永远可以提出新的做法**

Applications …

Self-attention 当然是用得很广,我们已经提过很多次 transformer 这个东西



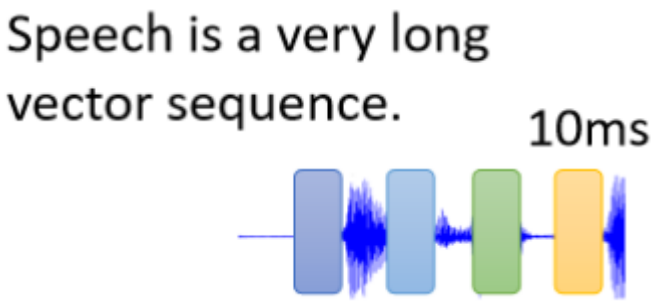
那我们大家也都知道说,在 NLP 的领域有一个东西叫做 BERT,BERT 裡面也用到 Self-attention,所以 Self-attention 在 NLP 上面的应用,是大家都耳熟能详的

但 **Self-attention,不是只能用在 NLP 相关的应用上,它还可以用在很多其他的问题上**,

Self-attention for Speech

比如说在做语音的时候,你也可以用 Self-attention,不过在做语音的时候,你可能会对 Self-attention,做一些小小的改动

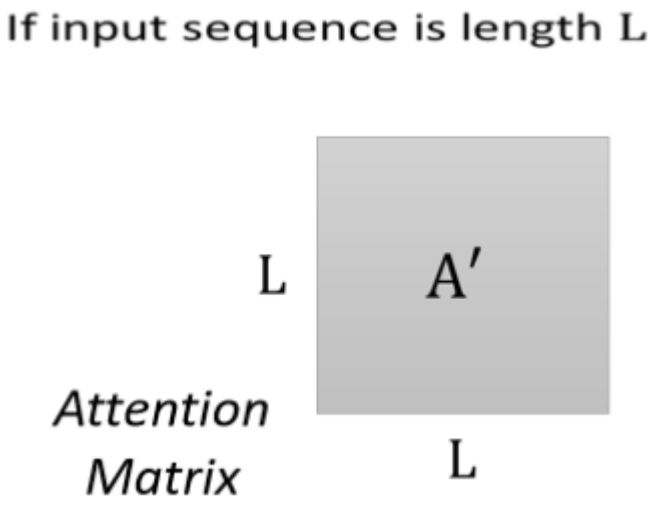
因為一般语音的,如果你要把一段声音讯号,表示成一排向量的话,这排**向量可能会非常地长**,



而每一个向量,其实只代表了 10 millisecond 的长度而已,所以如果今天是 1 秒鐘的声音讯号,它就有 100 个向量了,5 秒鐘的声音讯号,就 500 个向量了,你随便讲一句话,都是上千个向量了

所以一段声音讯号,你要描述它的时候,那个像这个 vector 的 sequence 它的长度是非常可观的,那可观的 sequence,可观的长度,会造成什麼问题呢

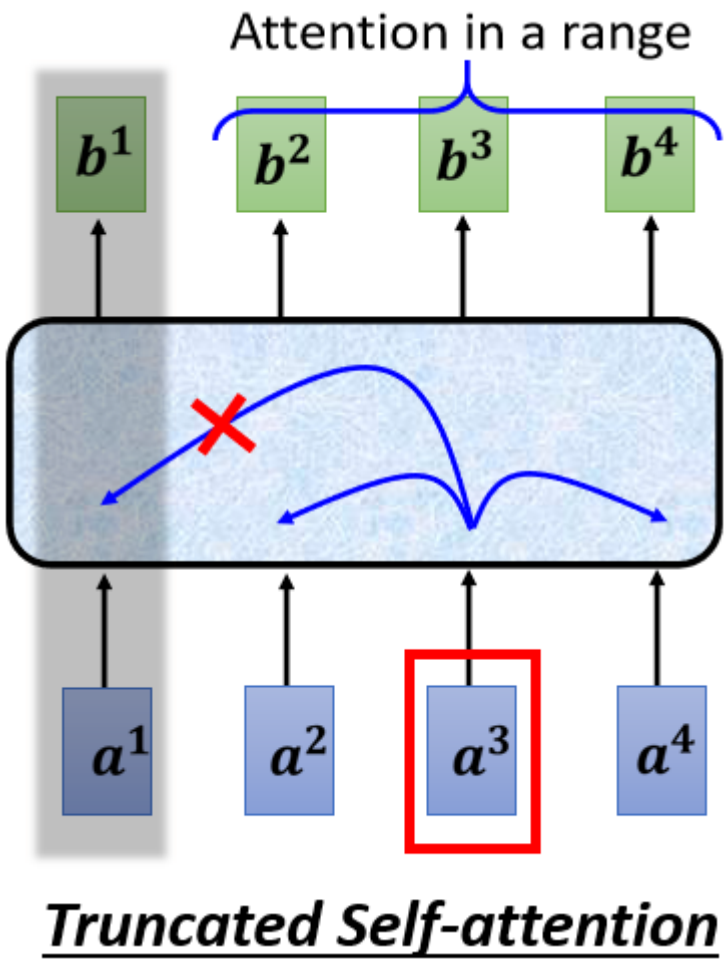
你想想看,我们今天在**计算这个 attention matrix 的时候,它的 计算complexity 是长度的平方**



计算这个 attention matrix A′你需要做 L 乘以 L 次的 inner product,那如果这个 L 的值很大的话,它的计算量就很可观,你也需要很大的这个 memory,才能够把这个矩阵存下来

所以今天如果在做语音辨识的时候,一句话所產生的这个 attention matrix,可能会太大,大到你根本就不容易处理,不容易训练,所以怎麼办呢

在做语音的时候,有一招叫做 ==Truncated Self-attention==



Truncated Self-attention 做的事情就是,我们今天在做 Self-attention 的时候,**不要看一整句话,就我们就只看一个小的范围就好**

那至於**这个范围应该要多大,那个是人设定的**

那為什麼我们知道说,今天在做语音辨识的时候,也许只需要看一个小的范围就好,那就是**取决於你对这个问题的理解**,也许我们要辨识这个位置有什麼样的**phoneme**,这个位置有什麼样的内容,我们并不需要看整句话,只要看这句话,跟它前后一定范围之内的资讯,其实就可以判断

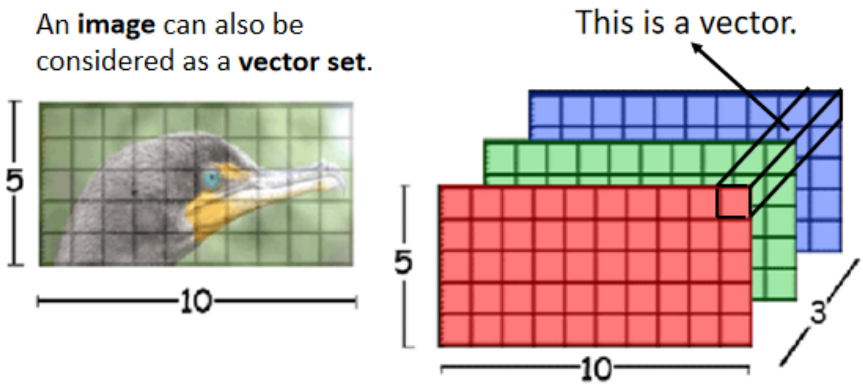
所以如果在做 Self-attention 的时候,也许没有必要看过一整个句子,也许没有必要让 Self-attention 考虑一整个句子,也许只需要考虑一个小范围就好,这样就可以加快运算的速度，这个是 Truncated Self-attention,

Self-attention for Image

那其实 Self-attention ,还可以被用在影像上,Self-attention

那到目前為止,我们在讲 Self-attention 的时候,我们都说 **Self-attention 适用的范围是：输入是一个 vector set 的时候**

一张图片啊,我们把它看作是一个很长的向量,那**其实一张图片,我们也可以换一个观点,把它看作是一个 vector 的 set**

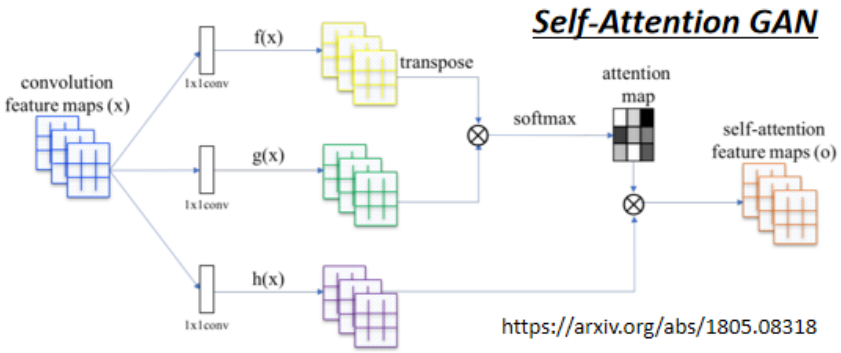


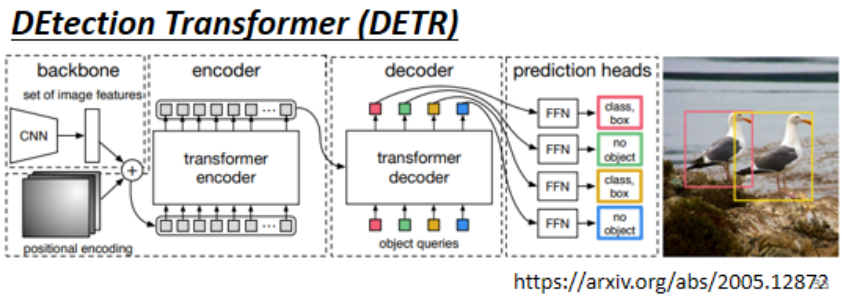
Source of image: https://www.researchgate.net/figure/Color-image-representation-and-RGB-matrix_fig15_282798184

这个是一个解析度 5 乘以 10 的图片,那这一张图片呢,可以看作是一个 tensor,这个 tensor 的大小是 5 乘以 10 乘以 3,3 代表 RGB 这 3 个 channel

你可以把每一个位置的 pixel,看作是一个三维的向量,所以**每一个 pixel,其实就是一个三维的向量**,那**整张图片,其实就是 5 乘以 10 个向量的set**

所以我们其实可以换一个角度,影像这个东西,其实也是一个 vector set,它既然也是一个 vector set 的话,你完全可以用 Self-attention 来处理一张图片,那有没有人用 Self-attention 来处理一张图片呢,是有的



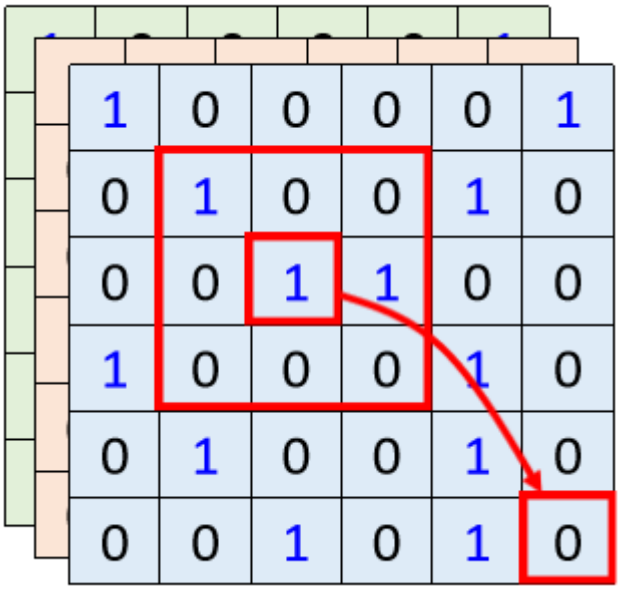


那这边就举了两个例子,来给大家参考,那现在把 Self-attention 用在影像处理上,也不算是一个非常石破天惊的事情,

Self-attention v.s. CNN

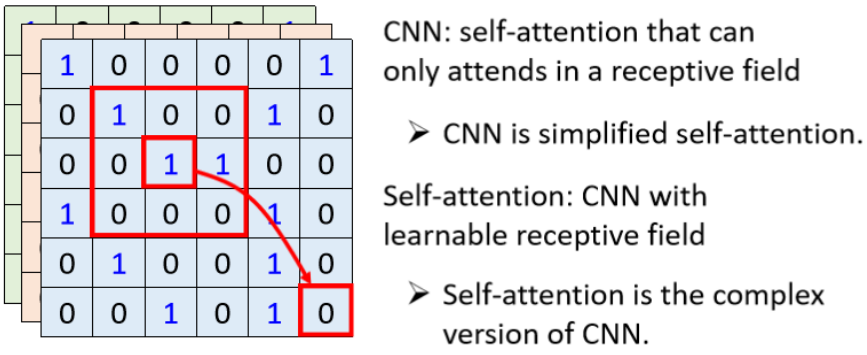
我们可以来比较一下,Self-attention 跟 CNN 之间,有什麼样的差异或者是关联性

如果我们今天,是用 Self-attention 来处理一张图片,代表说,假设这个是你要考虑的 pixel,那它產生 query,其他 pixel 產生 key,



你今天在做 inner product 的时候,你考虑的不是一个小的receptive field的信息,而是整张影像的资讯

但是今天在做 CNN 的时候,,会画出一个 receptive field,每一个 filter,每一个 neural,只考虑 receptive field 范围裡面的资讯



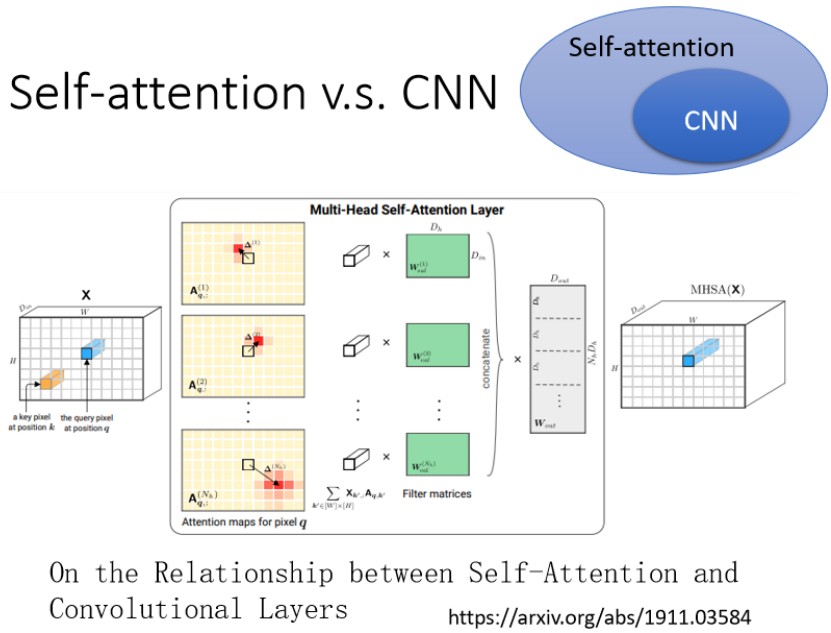
- 所以如果我们比较 CNN 跟 Self-attention 的话,**CNN 可以看作是一种简化版的 Self-attention**，因為在做CNN的时候,我们只考虑 receptive field 裡面的资讯,而在做 Self-attention 的时候,我们是考虑整张图片的资讯,所以 CNN,是简化版的 Self-attention
- 或者是你可以反过来说,**Self-attention 是一个复杂化的 CNN**

在 CNN 裡面,我们要划定 receptive field,每一个 neural,只考虑 receptive field 裡面的资讯,而 **receptive field 的范围跟大小,是人决定的,**

而对 Self-attention 而言,我们用 attention,去找出相关的 pixel,就好像是 **receptive field 是自动被学出的**,network 自己决定说,receptive field 的形状长什麼样子,network 自己决定说,以这个 pixel 為中心,哪些 pixel 是我们真正需要考虑的,那些 pixel 是相关的

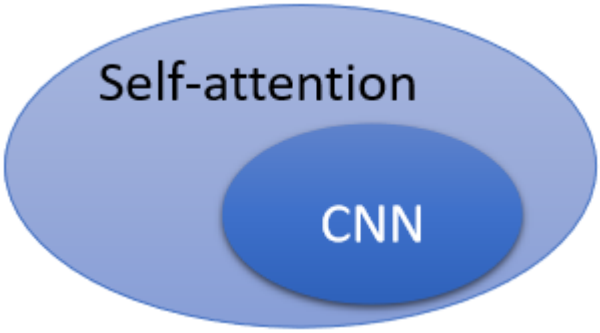
所以 receptive field 的范围,不再是人工划定,而是让机器自己学出来

其实你可以读一篇 paper,叫做 On the Relationship,between Self-attention and Convolutional Layers



在这篇 paper 裡面,会用数学的方式严谨的告诉你说,其实这个 **CNN就是 Self-attention 的特例,Self-attention 只要设定合适的参数,它可以做到跟 CNN 一模一样的事情**

所以 self attention,是更 flexible 的 CNN,而 CNN 是有受限制的 Self-attention,Self-attention 只要透过某些设计,某些限制,它就会变成 CNN

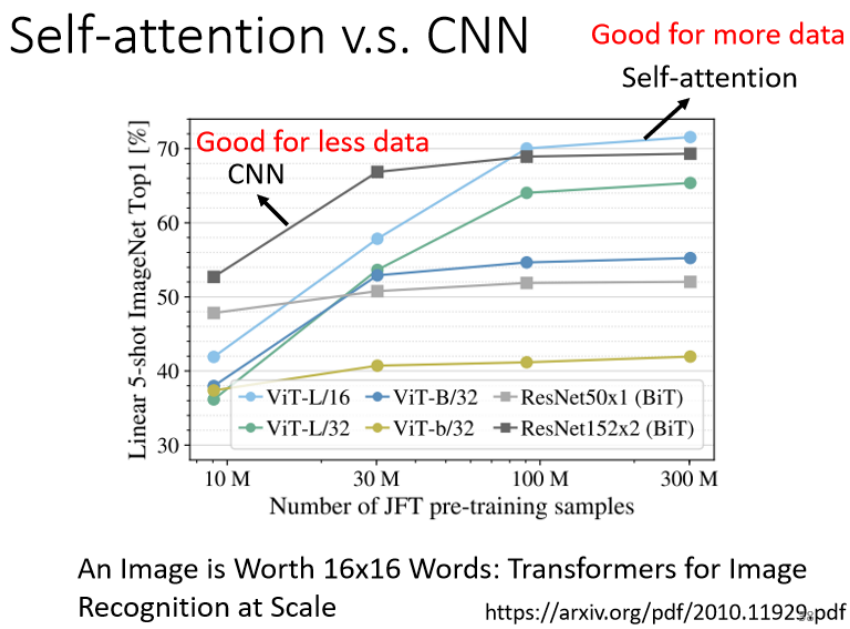


那这也不是很旧的 paper,你发现它放到网路上的时间呢,是 19 年的 11 月,所以你知道这些,我们今天上课裡面讲的东西,其实都是很新的资讯

既然Self-attention 比较 flexible,之前有讲说**比较 flexible 的 model,比较需要更多的 data,如果你 data 不够,就有可能 overfitting**

而小的 model,而比较有限制的 model,它适合在 data 小的,少的时候,它可能比较不会 overfitting,那如果你这个限制设的好,也会有不错的结果

如果你今天用不同的 data 量,来训练 CNN 跟 Self-attention,你确实可以看到我刚才讲的现象



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale https://arxiv.org/pdf/2010.11929.pdf

那这个实验结果,来自於 An image is worth 16 乘以 16 的 words,这个是 Google 的 paper,它就是把这个 Self-attention,apply 在影像上面

那其实**把一张影像呢,拆成 16 乘以 16 个 patch,它把每一个 patch想像成是一个 word**,因為一般我们这个 Self-attention,比较常用在 NLP 上面,所以他就说,想像每一个 patch 其实就是一个 word,所以他就取了一个很 fancy 的 title,叫做一张图呢,值 16 乘以 16 个文字

横轴是训练的影像的量,那你发现说,对 Google 来说 用的,所谓的资料量比较少,也是你没有办法用的资料量啦这边有 10 个 million 就是,1000 万张图,是资料量比较小的 setting,然后资料量比较大的 setting 呢,有 3 亿张图片,在这个实验裡面呢,比较了 Self-attention 是浅蓝色的这一条线,跟 CNN 是深灰色的这条线

就会发现说,**随著资料量越来越多,那 Self-attention 的结果就越来越好,最终在资料量最多的时候,Self-attention 可以超过 CNN,但在资料量少的时候,CNN 它是可以比 Self-attention,得到更好的结果的**

那為什麼会这样,你就可以从 CNN 跟 Self-attention,它们的弹性来加以解释

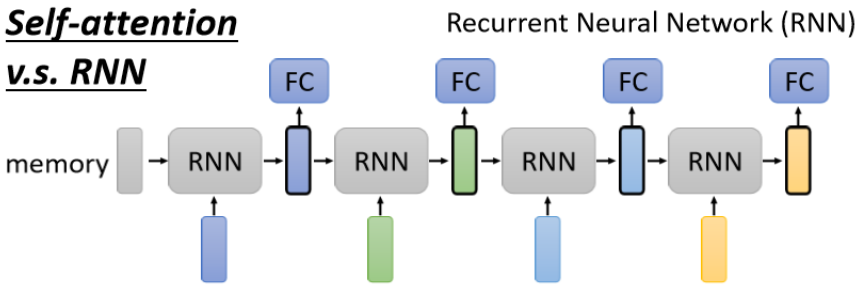
- Self-attention 它弹性比较大,所以需要比较多的训练资料,训练资料少的时候,就会 overfitting
- 而 CNN 它弹性比较小,在训练资料少的时候,结果比较好,但训练资料多的时候,它没有办法从更大量的训练资料得到好处

所以这个就是 Self-attention 跟 CNN 的比较， 那 Self-attention 跟 CNN,谁比较好呢,**我应该选哪一个呢,事实上你也可以都用**,在我们作业四裡面,如果你要做 strong baseline 的话,就特别给你一个提示,就是用 conformer,裡面就是有用到 Self-attention,也有用到 CNN

Self-attention v.s. RNN

我们来比较一下,Self-attention 跟 RNN,RNN就是 recurrent neural network,这门课裡面现在就不会讲到 recurrent neural network,因為 recurrent neural network 的角色,很大一部分都可以用 Self-attention 来取代了,

但是 RNN 是什麼呢,假设你想知道的话,那这边很快地三言两语把它带过去,RNN 跟 Self-attention 一样,都是要处理 input 是一个 sequence 的状况



在 RNN 裡面呢

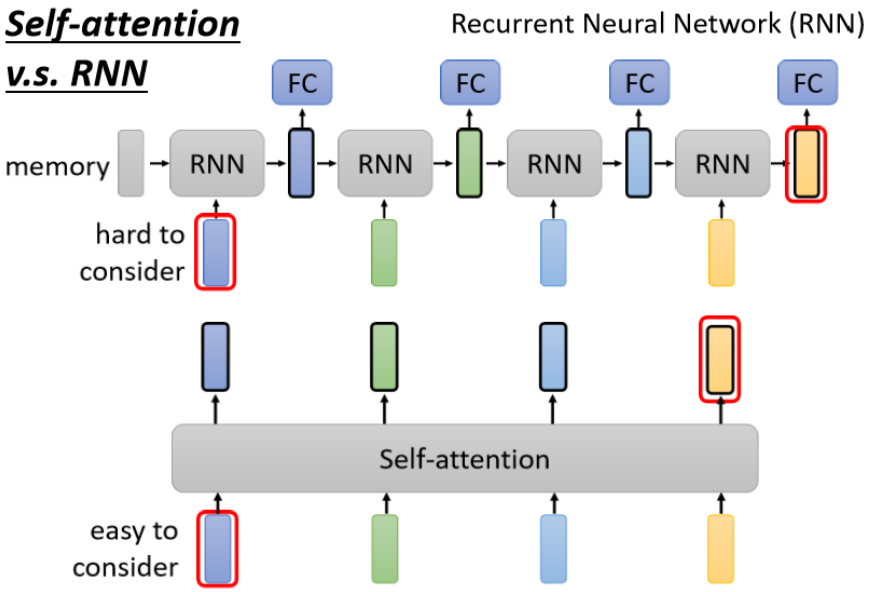
- 左边是你的 input sequence,你有一个 memory 的 vector
- 然后你有一个 RNN 的 block,这个 RNN 的 block 呢,它吃 memory 的 vector,吃第一个 input 的 vector
- 然后 output 一个东西,然后根据这个 output 的东西,我们通常叫做这个 hidden,这个 hidden 的 layer 的 output
- 然后通过这个 fully connected network,然后再去做你想要的 prediction

接下来当sequence 裡面,第二个 vector 作為 input 的时候,也会把前一个时间点吐出来的东西,当做下一个时间点的输入,再丢进 RNN 裡面,然后再產生新的 vector,再拿去给 fully connected network

然后第三个 vector 进来的时候,你把第三个 vector 跟前一个时间点的输出,一起丢进 RNN,再產生新的输出,然后在第四个时间点

第四个 vector 输入的时候,把第四个 vector 跟前一个时间点,產生出来的输出,再一起做处理,得到新的输出,再通过 fully connected network 的 layer,这个就是 RNN

Recurrent Neural Network跟 Self-attention 做的事情其实也非常像,它们的 **input 都是一个 vector sequence**



Self-attention output 是另外一个 vector sequence,这裡面的每一个 vector,都**考虑了整个 input sequence 以后**,再给 fully connected network 去做处理

那 RNN 呢,它也会 output 另外一群 vector,这另外一排 vector 也会给,fully connected network 做进一步的处理,那 Self-attention 跟 RNN 有什麼不同呢

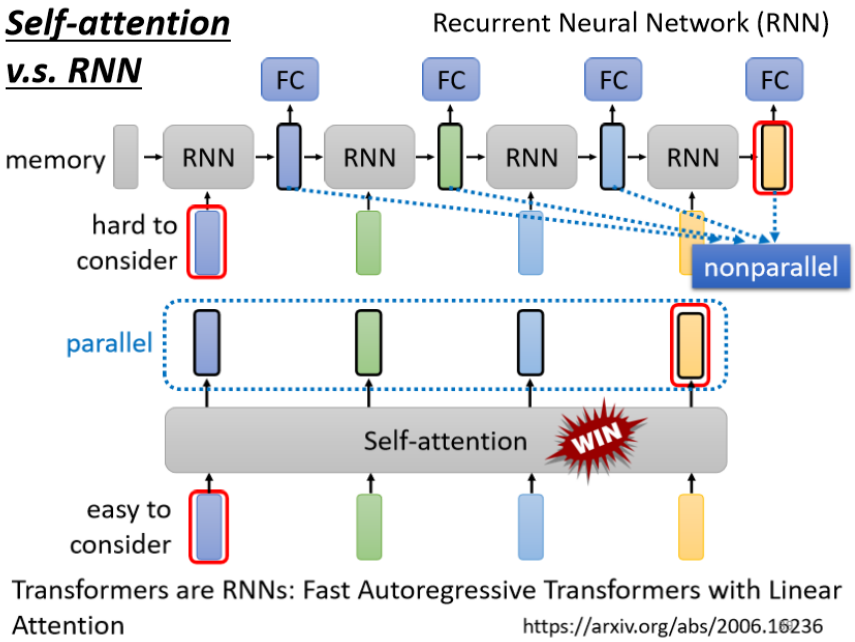
当然一个非常显而易见的不同,你可能会说,这边的每一个 vector,它都考虑了整个 input 的 sequence,而 RNN 每一个 vector,只考虑了左边已经输入的 vector,它没有考虑右边的 vector,那这是一个很好的观察

但是 **RNN 其实也可以是双向的**,所以如果你 RNN 用双向的 RNN 的话,其实这边的每一个 hidden 的 output,每一个 memory 的 output,其实也可以看作是考虑了整个 input 的 sequence

但是假设我们把 RNN 的 output,跟 Self-attention 的 output 拿来做对比的话,就算你用 bidirectional 的 RNN,还是有一些差别的

- 对 RNN 来说,假设最右边这个黄色的 vector,要考虑最左边的这个输入,那它必须要把最左边的输入存在 memory 裡面,然后接下来都不能够忘掉,一路带到最右边,才能够在最后一个时间点被考虑
- 但对 Self-attention 来说没有这个问题,它只要这边输出一个 query,这边输出一个 key,只要它们 match 得起来,天涯若比邻,你可以从非常远的 vector,在整个 sequence 上非常远的 vector,轻易地抽取资讯,所以这是 RNN 跟 Self-attention,一个不一样的地方

还有另外一个更主要的不同是,RNN 今天在处理的时候, input 一排 sequence,output 一排 sequence 的时候,**RNN 是没有办法平行化的**



RNN 它今天 input 一排是 vector,output 另外一排 vector 的时候,它没有办法一次处理,没有办法平行处理所有的 output

但 Self-attention 有一个优势,是它可以平行处理所有的输出,你今天 input 一排 vector,再 output 这四个 vector 的时候,**这四个 vector 是平行產生的,并不需要等谁先运算完才把其他运算出来**,output 的这个 vector,裡面的 output 这个 vector sequence 裡面,每一个 vector 都是同时產生出来的

所以在运算速度上,Self-attention 会比 RNN 更有效率

那你今天发现说,**很多的应用都往往把 RNN 的架构,逐渐改成 Self-attention 的架构了**,如果你想要更进一步了解,RNN 跟 Self-attention 的关係的话,你可以看下面这篇文章,Transformers are RNNs,裡面会告诉你说,Self-attention 你加上了什麼东西以后,其实它就变成了 RNN,发现说这也不是很旧的 paper,这个是去年的六月放到 arXiv 上

所以今天讲的都是一些很新的研究成果,那 RNN 的部分呢,我们这门课就不会提到,假设你对 RNN 有兴趣的话,以下是这一门课之前的上课录影,那 RNN 的部分,因為这一次不会讲到,所以特别有做了英文的版本,RNN 呢 是中文英文版本,都同时有放在 YouTube 上面

To learn more about RNN .....

https://youtu.be/xCGidAeyS4M

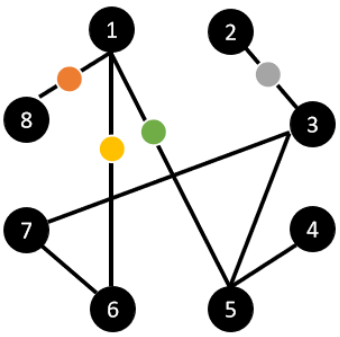
(in Mandarin)

https://youtu.be/Jjy6ER0bHv8

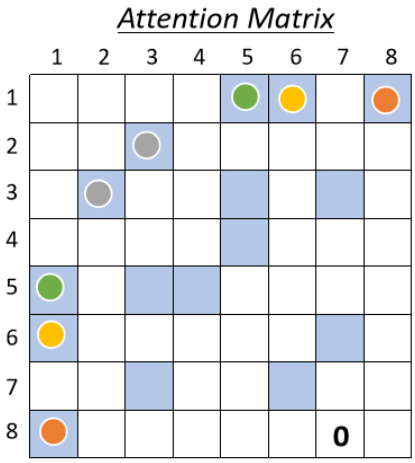
(in English)

Self-attention for Graph

Graph 也可以看作是一堆 vector,那如果是一堆 vector,就可以用 Self-attention 来处理,所以 Self-attention 也可以用在 Graph 上面,但是当我们把 Self-attention,用在 Graph 上面的时候,有什麼样特别的地方呢,、



Consider **edge**: only attention to connected nodes



This is one type of **Graph Neural Network (GNN).**

在 Graph 上面,每一个 node 可以表示成一个向量,但**不只有 node 的资讯,还有 edge 的资讯**,我们知道哪些 node 之间是有相连的,也就是哪些 node 是有关联的

我们知道哪些向量间是有关联,那之前我们在做 Self-attention 的时候,所谓的关联性是 network 自己找出来的,但是现在既然有了 Graph 的资讯,**有了 edge 的资讯,那关联性也许就不需要透过机器自动找出来,这个图上面的 edge 已经暗示了我们,node 跟 node 之间的关联性**

所以今天当你把 Self-attention,用在 Graph 上面的时候,你有一个选择是你在做这个,Attention Matrix 计算的时候,你可以**只计算有 edge 相连的 node 就好**

举例来说在这个图上,node 1 跟 node 8 有相连,那我们只需要计算 node 1 跟 node 8,这两个向量之间的 attention 的分数,那 1 跟 6 相连,所以只有 1 跟 6 之间,需要计算 attention 的分数,1 跟 5 有相连,所以只有 1 跟 5 需要计算 attention 的分数,2 跟 3 有相连,所以只有 2 跟 3 需要计算 attention 的分数,以此类推

那如果两个 node 之间没有相连,那其实很有可能就暗示我们,这两个 node 之间没有关係,**既然没有关係,我们就不需要再去计算它的 attention score,直接把它设为 0 就好了**

因為这个 **Graph 往往是人為根据某些 domain knowledge 建出来的**,那 domain knowledge 告诉我们说,这两个向量彼此之间没有关联,我们就没有必要再用机器去学习这件事情

其实当我们把 Self-attention,按照我们这边讲的这种限制,用在 Graph 上面的时候,其实就是一种 Graph Neural Network,也就是一种 GNN

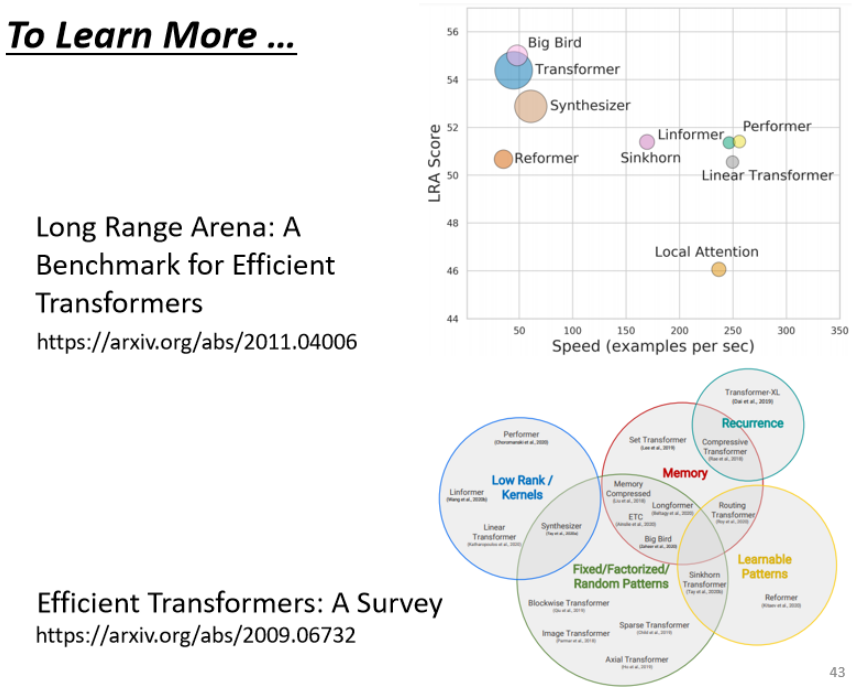
那我知道 GNN,现在也是一个很 fancy 的题目,那我不会说 Self-attention 就要囊括了,所有 GNN 的各种变形了,但把 Self-attention 用在 Graph 上面,是某一种类型的 Graph Neural Network,那这边呢,一样我们也没有办法细讲了,GNN 这边坑也是很深啊,这边水是很深,那就放一下助教之前上课的连结



大概花了快三个小时,在讲 Graph Neural Network,而且其实还没有讲完,就告诉你说这个 Graph Neural Network,也是有非常深的技术,这边水也是很深,那这不是我们今天这一堂课可以讲的内容,好

More

其实Self-attention 有非常非常多的变形,你可以看一篇 paper 叫做,Long Range Arena,裡面比较了各种不同的 Self-attention 的变形



因為 Self-attention 它最大的问题就是,**它的运算量非常地大**,所以怎麼样减少 Self-attention 的运算量,是一个未来的重点,可以看到这边有,各种各式各样 Self-attention 的变形

Self-attention 最早是,用在 Transformer 上面,所以很多人讲 Transformer 的时候,其实它指的就是这个 Self-attention,有人说广义的 Transformer,指的就是 Self-attention,那所以后来各式各样的,Self-attention 的变形都这样做,都叫做是什麼 former,比如说 Linformer Performer Reformer 等等,所以 Self-attention 的变形,现在都叫做 xxformer

那可以看到，往右代表它运算的速度,所以有很多各式各样新的 xxformer,它们的速度会比原来的 Transformer 快,但是快的速度带来的就是 performance 变差

这个纵轴代表是 performance,所以它们往往比原来的 Transformer,performance 差一点,但是速度会比较快

那到底什麼样的 Self-attention,才能够真的又快又好,这仍然是一个尚待研究的问题,如果你对 Self-attention,想要进一步研究的话,你还可以看一下,Efficient Transformers: A Survey 这篇 paper,裡面会跟你介绍,各式各样 Self-attention 的变形。

Transformer_P1_Encoder



变形金刚的英文就是Transformer,那Transformer也跟我们之后会,提到的BERT有非常强烈的关係,所以这边有一个BERT探出头来,代表说Transformer跟BERT,是很有关係的

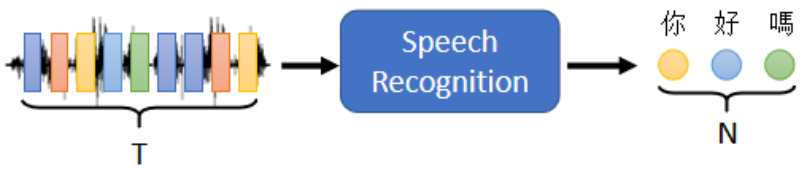
Sequence-to-sequence (Seq2seq)

Transformer就是一个,==Sequence-to-sequence==的model,他的缩写,我们会写做==Seq2seq==,那Sequence-to-sequence的model,又是什麼呢

我们之前在讲input a sequence的,case的时候,我们说input是一个sequence,那output有几种可能

- 一种是input跟output的长度一样,这个是在作业二的时候做的
- 有一个case是output指,output一个东西,这个是在作业四的时候做的
- 那接来作业五的case是,我们不知道应该要output多长,由机器自己决定output的长度,即Seq2seq

1. 举例来说,Seq2seq一个很好的应用就是 **语音辨识**

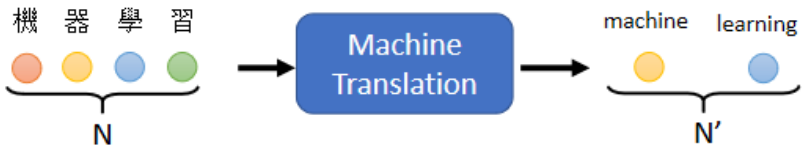


在做语音辨识的时候,输入是声音讯号,声音讯号其实就是一串的vector,输出是语音辨识的结果,也就是输出的这段 声音讯号,所对应的文字

我们这边用圈圈来代表文字,每一个圈圈就代表,比如说中文裡面的一个方块子,今天**输入跟输出的长度,当然是有一些关係,但是却没有绝对的关係**，输入的声音讯号,他的长度是大T,我们并没有办法知道说,根据大T输出的这个长度N一定是多少。

输出的长度由机器自己决定,由机器自己去听这段声音讯号的内容,自己决定他应该要输出几个文字,他输出的语音辨识结果,输出的句子裡面应该包含几个字,由机器自己来决定,这个是语音辨识

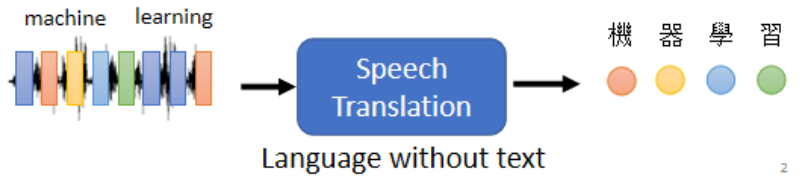
2. 还有很多其他的例子,比如说作业五我们会做机器翻译



让机器读一个语言的句子,输出另外一个语言的句子,那在做机器翻译的时候,**输入的文字的长度是N,输出的句子的长度是N',那N跟N'之间的关係,也要由机器自己来决定**

输入机器学习这个句子,输出是machine learning,输入是有四个字,输出有两个英文的词汇,但是并不是所有中文跟英文的关係,都是输出就是输入的二分之一,到底输入一段句子,输出英文的句子要多长,由机器自己决定

3. 甚至可以做更复杂的问题,比如说做语音翻译



语音翻译就是,你对机器说一句话,比如说machine learning,他输出的不是英文,他直接**把他听到的英文的声音讯号翻译成中文文字**

你对他说machine learning,他输出的是机器学习

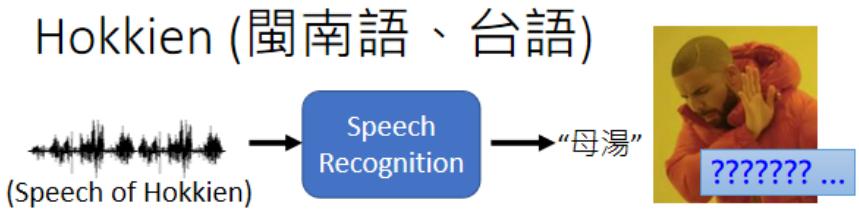
為什麼我们要做,Speech Translation这样的任务,為什麼我们不直接先做一个语音辨识,再做一个机器翻译,把语音辨识系统跟机器翻译系统,接起来 就直接是语音翻译?

因為**世界上有很多语言,他根本连文字都没有**,世界上有超过七千种语言,那其实在这七千种语言,有超过半数其实是没有文字的,对这些没有文字的语言而言,你要做语音辨识,可能根本就没有办法,因為他没有文字,所以你根本就没有办法做语音辨识,但我们有没有可能对这些语言,做语音翻译,直接把它翻译成,我们有办法阅读的文字

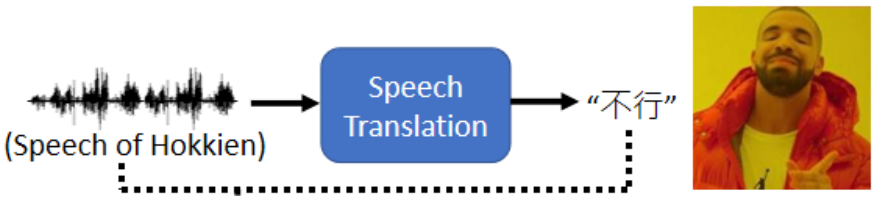
Hokkien（闽南语、台语）

一个很好的例子也许就是,台语的语音辨识,但我不会说台语没有文字,很多人觉得台语是有文字的,但台语的文字并没有那麼普及,现在听说小学都有教台语的文字了,但台语的文字,并不是一般人能够看得懂的

如果你做语音辨识,你给机器一段台语,然后它可能输出是母汤,你根本就不知道,这段话在说什麼。

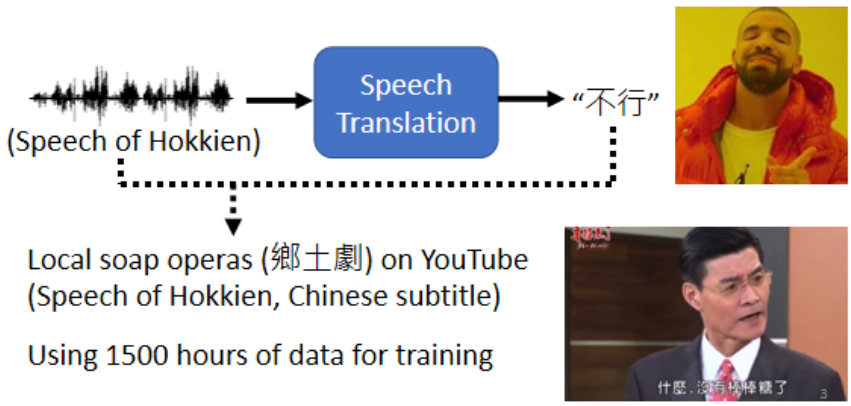


所以我们期待说机器也许可以做语音的翻译,对它讲一句台语,它直接输出的是同样意思的,中文的句子,那这样一般人就可以看懂



我们可以训练一个类神经网路,这个类神经网路听某一种语言,的声音讯号,输出是另外一种语言的文字。

今天你要训练一个neural network,你就需要有input跟output的配合,你需要有台语的声音讯号,跟中文文字的对应关係,那这样的资料是比较容易收集的。比如说YouTube上面,有很多的乡土剧

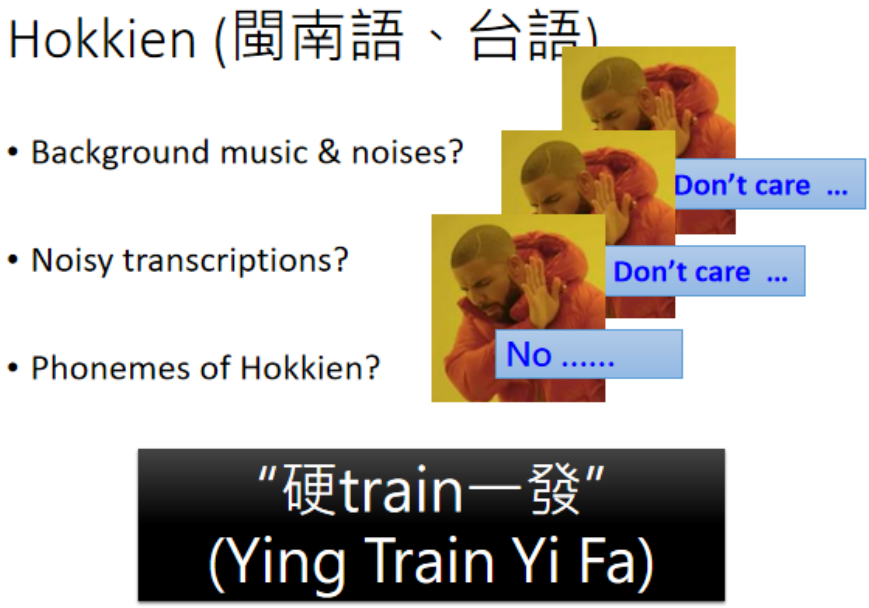


乡土剧就是,台语语音 中文字幕,所以你只要它的台语语音载下来,中文字幕载下来,你就有台语声音讯号,跟中文之间的对应关係,你就可以硬train一个模型,然后叫机器直接做台语的语音辨识,输入台语 输出中文

那你可能会觉得这个想法很狂,而且好像听起来有很多很多的问题,那我们实验室就载了,一千五百个小时的乡土剧的资料,然后 就真的拿来训练一个,语音辨识系统

你可能会觉得说,这听起来有很多的问题

- 乡土剧有很多杂讯,有很多的音乐,不要管它这样子
- 乡土剧的字幕,不一定跟声音有对起来,就不要管它这样子
- 台语还有一些,比如说台罗拼音,台语也是有类似音标这种东西,也许我们可以先辨识成音标,当作一个中介,然后在从音标转成中文,也没有这样做



直接训练一个模型,输入是声音讯号,输出直接就是中文的文字,这种没有想太多 直接资料倒进去,就训练一个模型的行为,就叫作==硬train一发==

那你可能会想说,这样子硬train一发到底能不能够,做一个台语语音辨识系统呢,其实还真的是有可能的,以下是一些真正的结果

机器在听的一千五百个小时的,乡土剧以后,你可以对它输入一句台语,然后他就输出一句中文的文字,以下是真正的例子

感謝 李仲翊 同學提供實驗結果



机器听到的声音是这样子的

- 你的身体撑不住(台语),那机器输出是什麼呢,它的输出是 你的身体撑不住,这个声音讯号是你的身体撑不住(台语),但机器并不是输出无勘,而是它就输出撑不住
- 或者是机器听到的,是这样的声音讯号,没事你為什麼要请假(台语),没事你為什麼要请假,机器听到没事(台语),它并不是输出 没代没誌,它是输出 没事,这样听到四个音节没代没誌(台语),但它知道说台语的没代没誌(台语),翻成中文 也许应该输出 没事,所以机器的输出是,没事你為什麼要请假

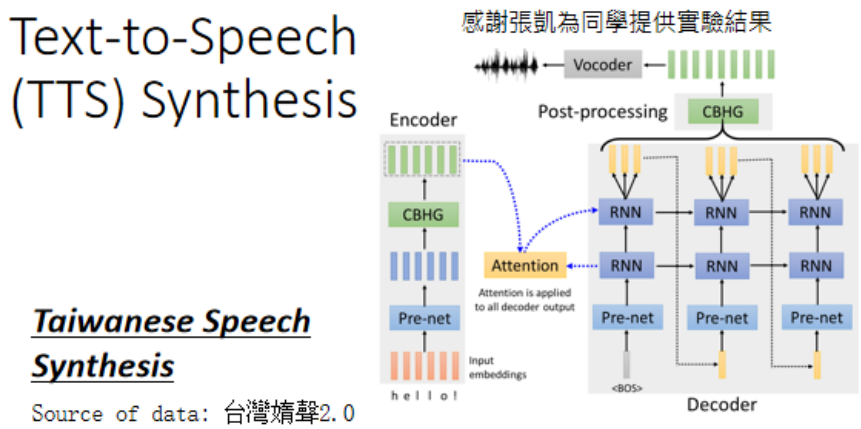
- 但机器其实也是蛮容易犯错的,底下特别找机个犯错的例子,给你听一下,你听听这一段声音讯号,不会腻吗(台语),他说不会腻吗(台语),我自己听到的时候我觉得,我跟机器的答案是一样的,就是说**要生了吗**,但其实这句话,正确的答案就是,不会腻吗(台语),不会腻吗
- 当然机器在倒装,你知道有时候你从台语,转成中文句子需要倒装,在倒装的部分感觉就没有太学起来,举例来说它听到这样的句子,我有跟厂长拜託(台语),他说我有跟厂长拜託(台语),那机器的输出是,我有帮厂长拜託,但是你知道说这句话,其实是倒装,我有跟厂长拜託(台语),是我拜託厂长,但机器对於它来说,如果台语跟中文的关係需要倒装的话,看起来学习起来还是有一点困难

这个例子想要告诉你说,直接台语声音讯号转繁体中文,不是没有可能,是有可能可以做得到的,那其实台湾有很多人都在做,台语的语音辨识,如果你想要知道更多有关,台语语音辨识的事情的话,可以看一下下面这个网站

Text-to-Speech (TTS) Synthesis

台语语音辨识反过来,就是台语的**语音合成**,我们如果是一个模型,输入台语声音 输出中文的文字,那就是语音辨识,反过来 输入文字 输出声音讯号,就是语音合成

这边就是demo一下台语的语音合成,这个资料用的是,台湾 婧声(台语)的资料,来找GOOGLE台湾婧声(台语),就可以找到这个资料集,裡面就是台语的声音讯号,听起来像是这个样子



比如说你跟它说,欢迎来到台湾台大语音处理实验室

不过这边是需要跟大家说明一下,现在还没有真的做End to End的模型,**这边模型还是分成两阶**,他会先把中文的文字,转成台语的台罗拼音,就像是台语的KK音标,在把台语的KK音标转成声音讯号,不过从台语的KK音标,转成声音讯号这一段,就是一个像是Transformer的network,其实是一个叫做echotron的model,它本质上就是一个Seq2Seq model,大概长的是这个样子

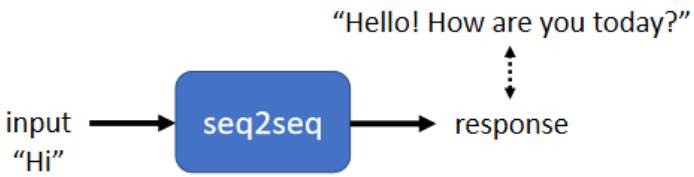
所以你输入文字,欢迎来到台大语音处理实验室,机器的输出是这个样子的,欢迎来到台大(台语),语音处理实验室(台语),或是你对他说这一句中文,然后他输出的台语是这个样子,最近肺炎真严重(台语),要记得戴口罩 勤洗手(台语),有病就要看医生(台语)

所以你真的是可以,合出台语的声音讯号的,就用我们在这一门课裡面学到的,Transformer或者是Seq2Seq的model

Seq2seq for Chatbot

刚才讲的是跟语音比较有关的,那在**文字**上,也会很广泛的使用了Seq2Seq model

举例来说你可以用Seq2Seq model,来训练一个聊天机器人



Training data:

[PERSON 1:] Hi
[PERSON 2:] Hello ! How are you today ?
[PERSON 1:] I am good thank you , how are you.
[PERSON 2:] Great, thanks ! My children and I were just about to watch Game of Thrones.
[PERSON 1:] Nice ! How old are your children?
[PERSON 2:] I have four that range in age from 10 to 21. You?
[PERSON 1:] I do not have children at the moment.
[PERSON 2:] That just means you get to keep all the popcorn for yourself.
[PERSON 1:] And Cheetos at the moment!
[PERSON 2:] Good choice. Do you watch Game of Thrones?
[PERSON 1:] No, I do not have much time for TV.
[PERSON 2:] I usually spend my time painting: but, I love the show.

聊天机器人就是你对它说一句话,它要给你一个回应,**输入输出都是文字**,文字就是一个vector Sequence,所以你完全可以用Seq2Seq 的model,来做一个聊天机器人

你就要**收集大量人的对话**,像这种对话你可以收集,电视剧 电影的台词 等等,你可以收集到,一堆人跟人之间的对话

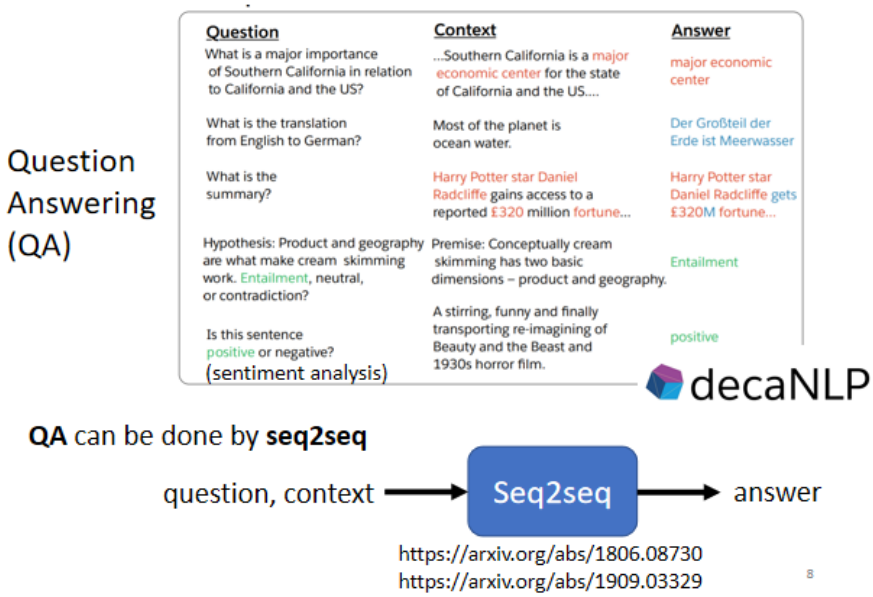
假设在对话裡面有出现,某一个人说Hi,和另外一个人说,Hello How are you today,那你就可以教机器说,看到输入是Hi,那你的输出就要跟,Hello how are you today,越接近越好

那就可以训练一个Seq2Seq model,那跟它说一句话,它就会给你一个回应

Question Answering (QA)

那事实上Seq2Seq model,在NLP的领域,在natural language processing的领域的使用,是比你想像的更为广泛,其实很多**natural language processing的任务,都可以想成是==question answering,QA==的任务**

Question Answering,就是给机器读一段文字,然后你问机器一个问题,希望他可以给你一个正确的答案

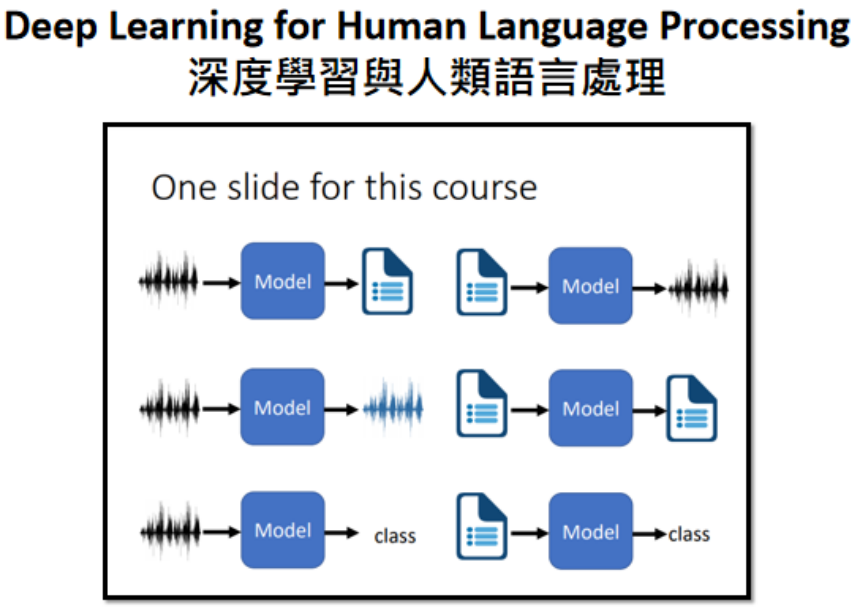


- 假设你今天想做的是翻译,那机器读的文章就是一个英文句子,**问题**就是这个句子的德文翻译是什麼,然后输出的**答案**就是德文
- 或者是你想要叫机器自动作摘要,摘要就是给机器读一篇长的文章,叫他把长的文章的重点节录出来,那你就是给机器一段文字,**问题**是这段文字的摘要是什麼,然后期待他**答案**可以输出一个摘要
- 或者是你想要叫机器做Sentiment analysis,Sentiment analysis就是机器要自动判断一个句子,是正面的还是负面的；假设你有做了一个產品,然后上线以后,你想要知道网友的评价,但是你又不可能一直找人家ptt上面,把每一篇文章都读过,所以就做一个Sentiment analysis model,看到有一篇文章裡面,有提到你的產品,然后就把这篇文章丢到,你的model裡面,去判断这篇文章,是正面还是负面。你就给机器要判断正面还负面的文章,**问题**就是这个句子,是正面还是负面的,然后希望机器可以告诉你**答案**

所以各式各样的NLP的问题,往往都可以看作是QA的问题,而**QA的问题,就可以用Seq2Seq model来解**

具体来说就是有一个Seq2Seq model输入,就是有问题跟文章把它接在一起,输出就是问题的答案,就结束了,你的问题加文章合起来,是一段很长的文字,答案是一段文字

Seq2Seq model只要是**输入一段文字**,**输出一段文字**,只要是**输入一个Sequence**,**输出一个Sequence**就可以解,所以你可以把QA的问题,硬是用Seq2Seq model解,叫它读一篇文章读一个问题,然后就直接输出答案,所以各式各样NLP的任务,其实都有机会使用Seq2Seq model



必须要强调一下,对多数NLP的任务,或对多数的语音相关的任务而言,往往為这些任务**客製化模型,你会得到更好的结果**

但是各个任务客製化的模型,就不是我们这一门课的重点了,如果你对人类语言处理,包括语音 包括自然语言处理,这些相关的任务有兴趣的话呢,可以参考一下以下课程网页的连结,就是去年上的深度学习,与人类语言处理,这门课的内容裡面就会教你,各式各样的任务最好的模型,应该是什麼

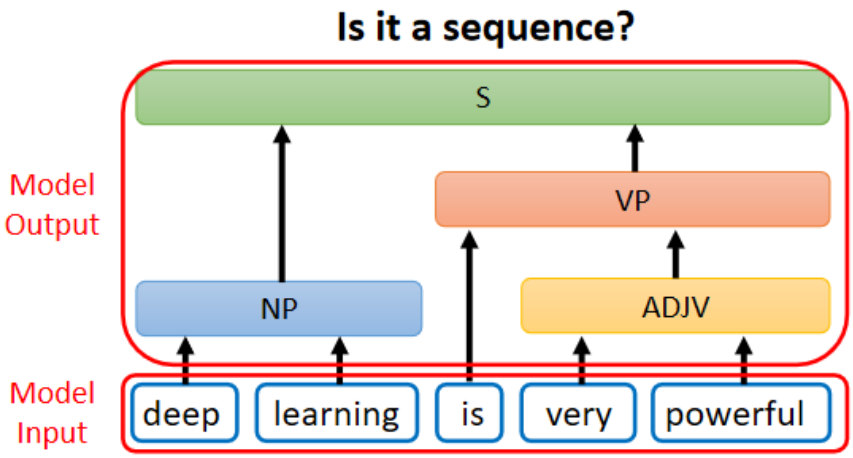
举例来说在做语音辨识,我们刚才讲的是一个Seq2Seq model,输入一段声音讯号,直接输出文字,今天啊 Google的 pixel4,Google官方告诉你说,Google pixel4也是用,N to N的Neural network,pixel4裡面就是,有一个Neural network,输入声音讯号,输出就直接是文字

但他其实用的不是Seq2Seq model,他用的是一个叫做,RNN transducer的 model,像这些模型他就是為了,语音的某些特性所设计,这样其实可以表现得更好,至於每一个任务,有什麼样客製化的模型,这个就是另外一门课的主题,就不是我们今天想要探讨的重点

Seq2seq for Syntactic Parsing

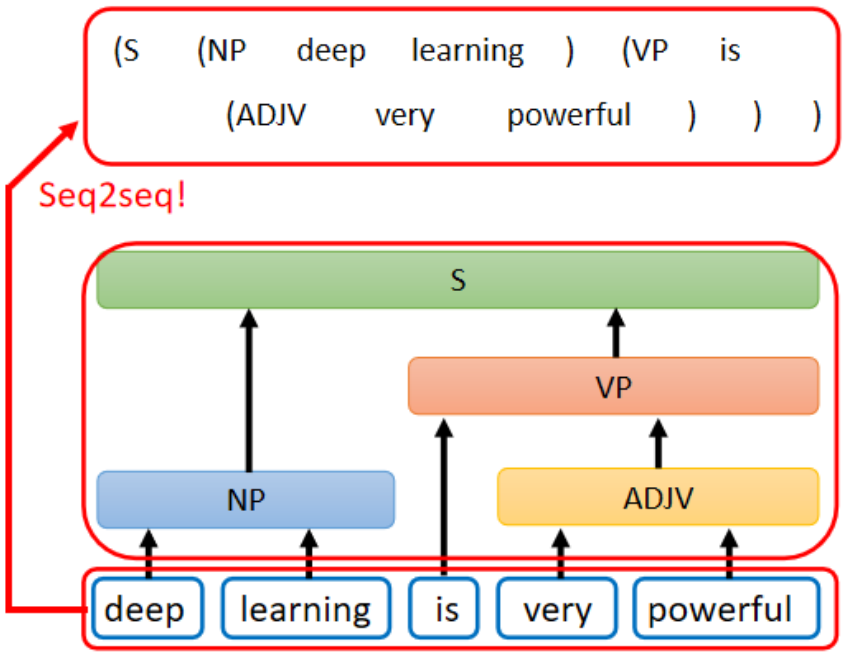
在语音还有自然语言处理上的应用,其实有很多应用,你**不觉得他是一个Seq2Seq model的问题,但你都可以硬用Seq2Seq model的问题硬解他**

举例来说**文法剖析**,给机器一段文字,比如Deep learning is very powerful



机器要做的事情是產生,一个**文法的剖析树** 告诉我们,deep加learning合起来,是一个名词片语,very加powerful合起来,是一个形容词片语,形容词片语加is以后会变成,一个动词片语,动词片语加名词片语合起来,是一个句子

那今天文法剖析要做的事情,就是產生这样子的一个Syntactic tree,所以在文法剖析的任务裡面,假设你想要deep learning解的话,输入是一段文字,他是一个Sequence,但输出看起来不像是一个Sequence,输出是一个树状的结构,但**事实上一个树状的结构,可以硬是把他看作是一个Sequence**



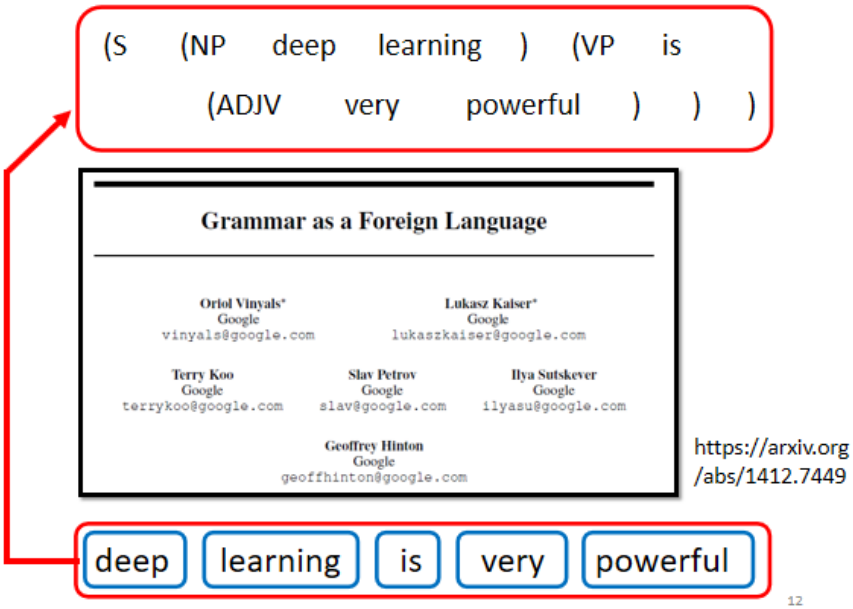
这个树状结构可以对应到一个,这样子的Sequence,从这个Sequence裡面,你也可以看出

- 这个树状的结构有一个S，有一个左括号,有一个右括号
- S裡面有一个noun phrase,有一个左括号跟右括号
- NP裡面有一个左括号跟右括号,NP裡面有is
- 然后有这个形容词片语,他有一个左括号右括号

这一个**Sequence就代表了这一个tree 的structure**,你先把tree 的structure,转成一个Sequence以后,你就可以用Seq2Seq model硬解他

train一个Seq2Seq model,读这个句子,然后直接输入这一串文字,再把这串文字转成一个树状的结构,你就可以硬是用Seq2Seq model,来做文法剖析这件事,这个概念听起来非常的狂,但这是真的可以做得到的,

你可以读一篇文章叫做,grammar as a Foreign Language



这篇文章其实不是太新的文章,你会发现她放在arxiv上面的时间,是14年的年底,所以其实也是一个,上古神兽等级的文章,这篇文章问世的时候,那个时候Seq2Seq model还不流行,那时候Seq2Seq model,主要只有被用在翻译上,所以这篇文章的title才会取说,grammar as a Foreign Language

他把文法剖析这件事情,当作是一个翻译的问题,把文法当作是另外一种语言,直接套用当时人们认为,只能用在翻译上的模型硬做,结果他得到state of the art的结果

我(李宏毅老师)其实在国际会议的时候,有遇过这个第一作者Oriol Vlnyals,那个时候Seq2Seq model,还是个非常潮的东西,那个时候在我的认知裡面,我觉得这个模型,应该是挺难train的,我问他说,train Seq2Seq model有没有什麼tips,没想到你做个文法剖析,用Seq2Seq model,居然可以硬做到state of the art,这应该有什麼很厉害的tips吧

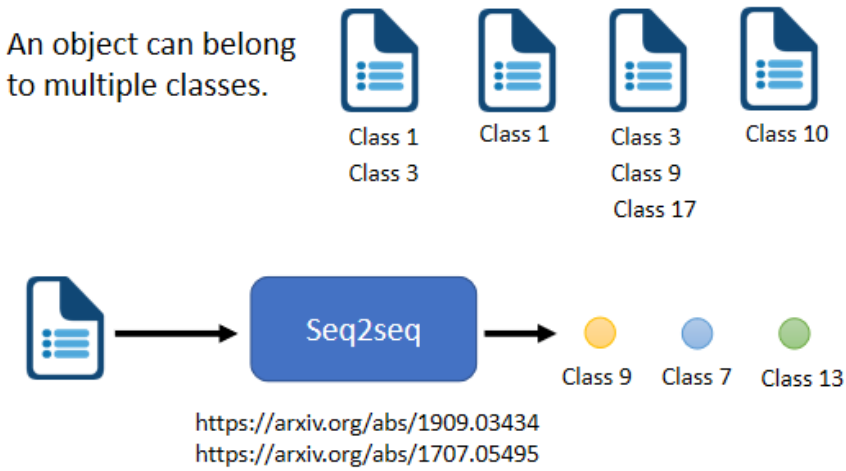
他说什麼没有什麼tips,他说我连Adam都没有用,我直接gradient descent,就train起来了,我第一次train就成功了,只是我要冲到state of the art,还是稍微调了一下参数而已,我也不知道是真的还假的啦,不过今天Seq2Seq model,真的是已经被很广泛地,应用在各式各样的应用上了

multi-label classification

还有一些任务可以用seq2seq's model,举例来说 ==multi-label的classification==

==multi-class==的classification,跟==multi-label==的classification,听起来名字很像,但他们其实是不一样的事情,multi-class的classification意思是说,我们有不只一个class机器要做的事情,是从数个class裡面,选择某一个class出来

但是multi-label的classification,意思是说**同一个东西,它可以属於多个class**,举例来说 你在做文章分类的时候



可能这篇文章 属於class 1跟3,这篇文章属於class 3 9 17等等,你可能会说,这种multi-label classification的问题,能不能**直接把它当作一个multi-class classification的问题来解**

举例来说,我把这些文章丢到一个classifier裡面

- 本来classifier只会输出一个答案,输出分数最高的那个答案
- 我现在就输出分数最高的前三名,看看能不能解,multi-label的classification的问题

但**这种方法可能是行不通的**,因為每一篇文章对应的class的数目,根本不一样 有些东西 有些文章,对应的class的数目,是两个 有的是一个 有的是三个

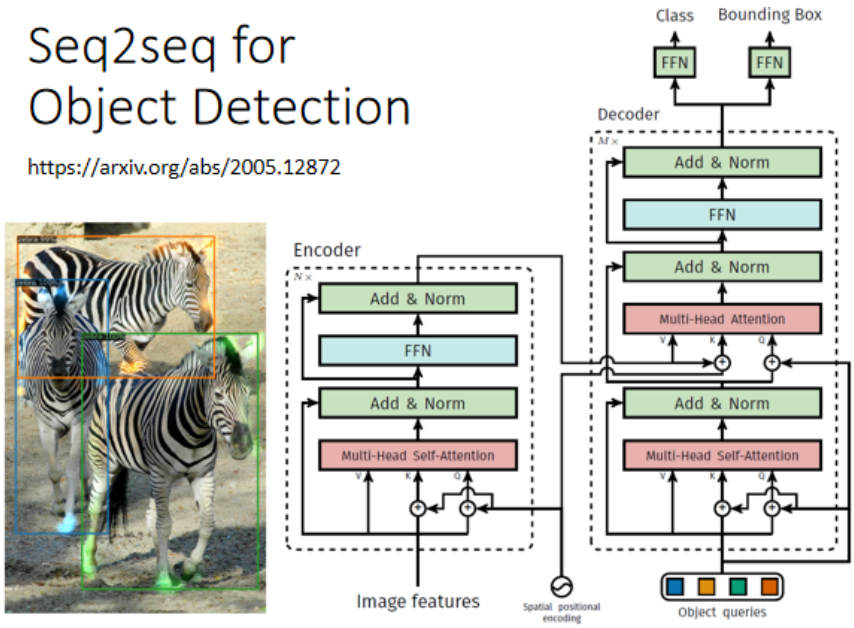
所以 如果你说 我直接取一个threshold,我直接取分数最高的前三名,class file output分数最高的前三名,来当作我的输出 显然,不一定能够得到好的结果 那怎麼办呢

这边可以用seq2seq硬做,**输入一篇文章 输出就是class** 就结束了,机器自己决定 它要输出几个class

我们说seq2seq model,就是由机器自己决定输出几个东西,输出的output sequence的长度是多少,既然 你没有办法决定class的数目,那就让机器帮你决定,每篇文章 要属於多少个class

Seq2seq for Object Detection

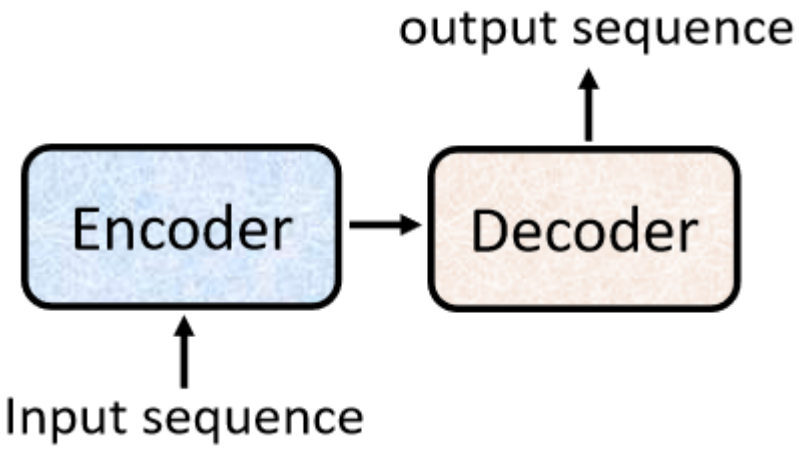
或者是object detection,这个看起来跟seq2seq model,应该八竿子打不著的问题,它也可以用seq2seq's model硬解



object detection就是给机器一张图片,然后它把图片裡面的物件框出来,把它框出说这个是斑马 这个也是斑马,但这种问题 可以用seq2seq's硬做,至於怎麼做 我们这边就不细讲,我在这边放一个文献,放一个连结给大家参考,讲这麼多就是要告诉你说,seq2seq's model 它是一个,很powerful的model,它是一个很有用的model

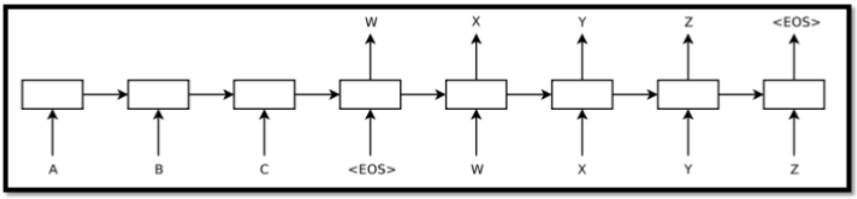
Encoder-Decoder

我们现在就是要来学,怎麼做seq2seq这件事,一般的seq2seq's model,它裡面会分成两块 一块是Encoder,另外一块是Decoder



你input一个sequence有Encoder,负责处理这个sequence,再把处理好的结果丢给Decoder,由Decoder决定,它要输出什麼样的sequence,等一下 我们都还会再细讲,Encoder跟 Decoder内部的架构

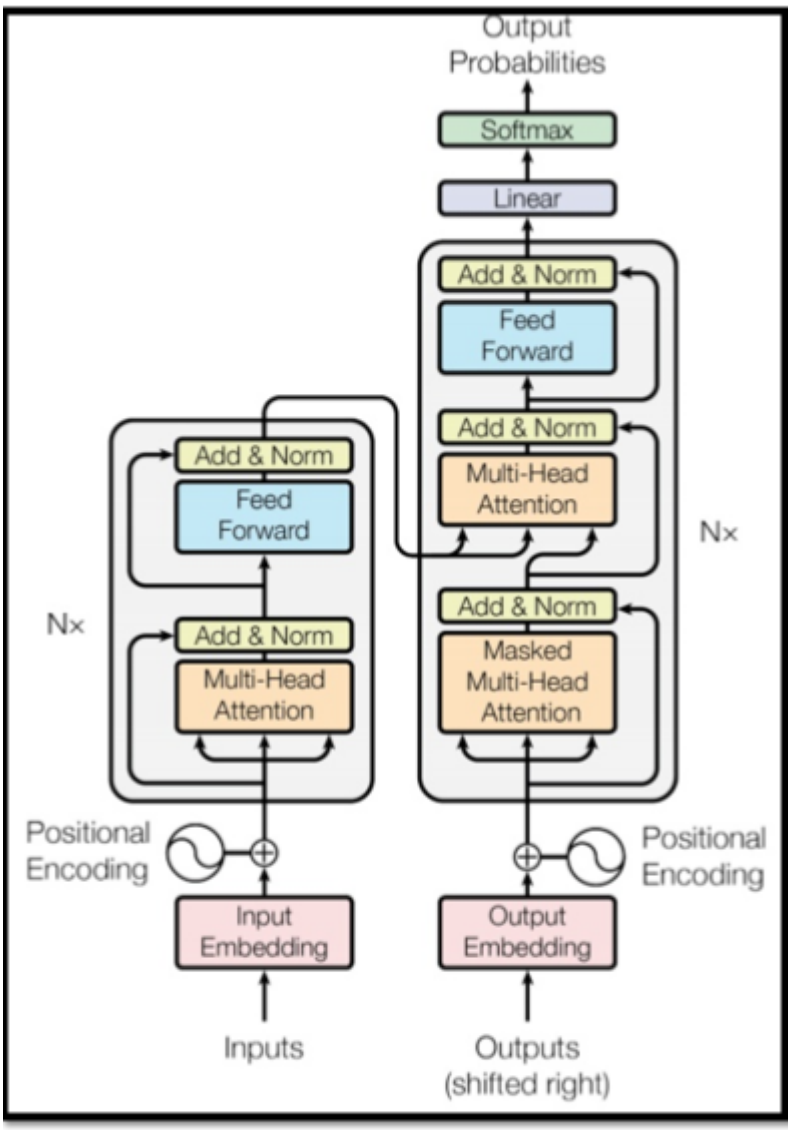
seq2seq model的起源,其实非常的早 在14年的9月,就有一篇seq2seq's model,用在翻译的文章 被放到Arxiv上



Sequence to Sequence Learning with Neural Networks

https://arxiv.org/abs/1409.3215

可以想像当时的seq2seq's model,看起来还是比较阳春的,今天讲到seq2seq's model的时候,大家第一个会浮现在脑中的,可能都是我们今天的主角,也就是transformer



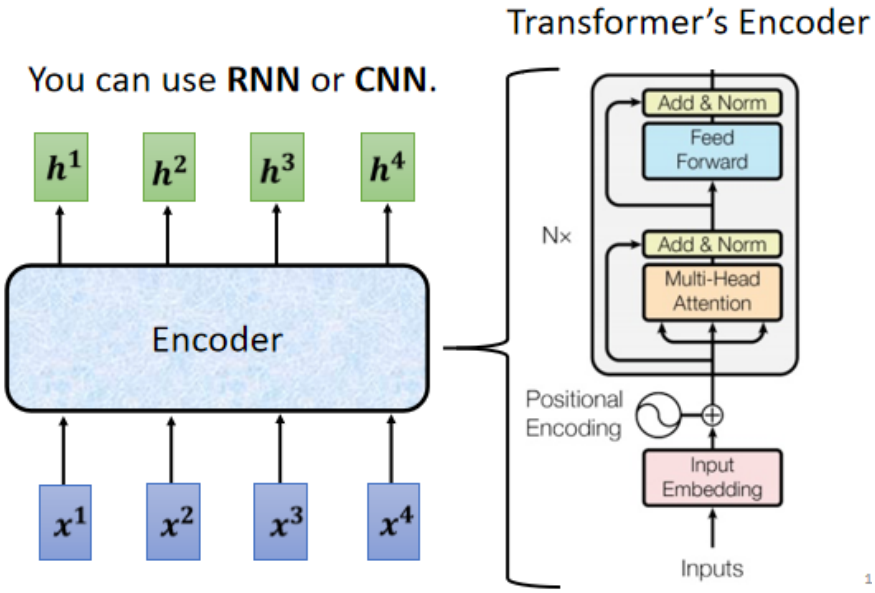
Transformer

https://arxiv.org/abs/1706.03762

它有一个Encoder架构,有一个Decoder架构,它裡面有很多花花绿绿的block,等一下就会讲一下,这裡面每一个花花绿绿的block,分别在做的事情是什麼

Encoder

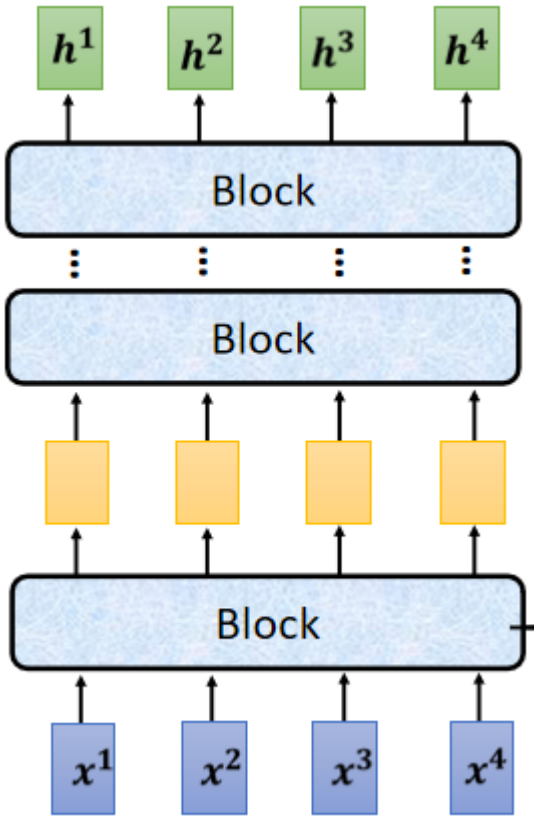
seq2seq model ==Encoder==要做的事情,就是**给一排向量，输出另外一排向量**



给一排向量、输出一排向量这件事情,很多模型都可以做到,可能第一个想到的是,我们刚刚讲完的self-attention,其实不只self-attention,RNN CNN 其实也都能够做到,input一排向量,output另外一个同样长度的向量

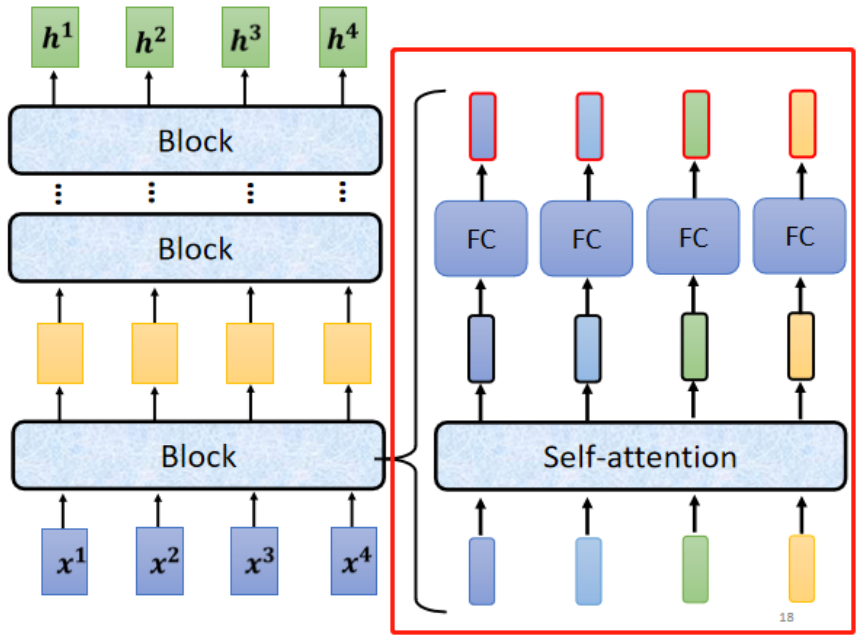
在transformer裡面,transformer的Encoder,用的就是self-attention,这边看起来有点复杂,我们用另外一张图,来仔细地解释一下,这个Encoder的架构,等一下再来跟原始的transformer的,论文裡面的图进行比对,

现在的Encoder裡面,会**分成很多很多的block**



每一个block都是输入一排向量,输出一排向量,你输入一排向量 第一个block,第一个block输出另外一排向量,再输给另外一个block,到最后一个block,会输出最终的vector sequence,**每一个block 其实,并不是neural network的一层**

每一个block裡面做的事情,是好几个layer在做的事情,在transformer的Encoder裡面,每一个block做的事情,大概是这样子的

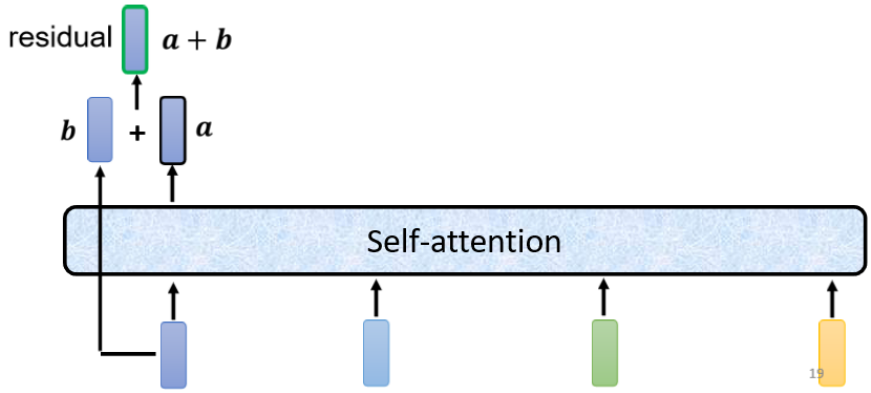


- 先做一个self-attention,input一排vector以后,做self-attention,考虑整个sequence的资讯，Output另外一排vector.

- 接下来这一排vector,会再丢到fully connected的feed forward network裡面,再output另外一排vector,这一排vector就是block的输出

事实上在原来的**transformer裡面,它做的事情是更复杂的**

在之前self-attention的时候,我们说 输入一排vector,就输出一排vector,这边的每一个vector,它是考虑了 所有的input以后,所得到的结果



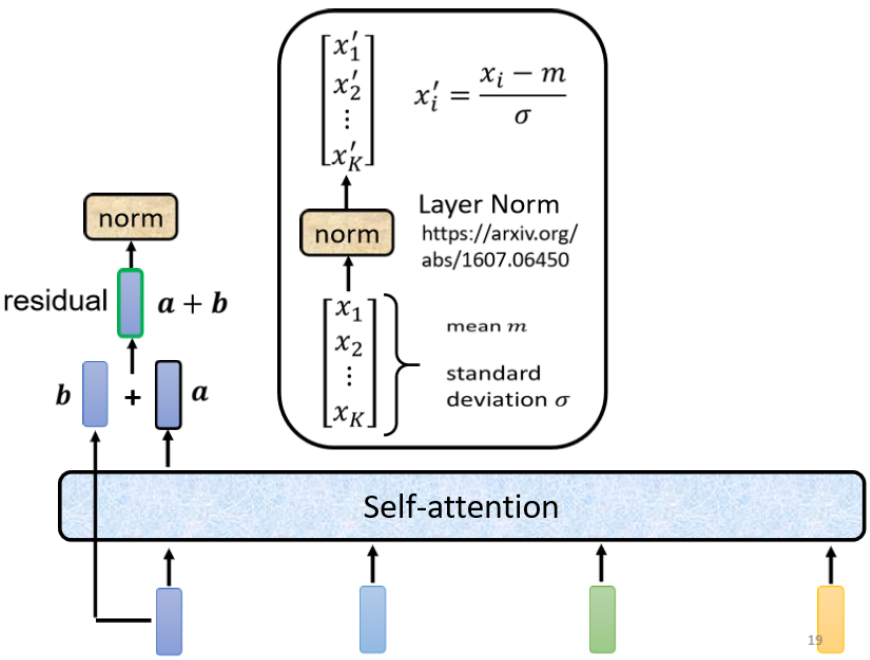
在transformer裡面,它加入了一个设计,我们**不只是输出这个vector**,我们还要**把这个vector加上它的input**,它要把input拉过来 直接加给输出,得到新的output

也就是说,这边假设这个vector叫做a,这个vector叫做b 你要把$a+b$当作是新的输出

这样子的network架构,叫做==residual connection==,那其实这种residual connection,在deep learning的领域用的是非常的广泛,之后如果我们有时间的话,再来详细介绍,為什麼要用residual connection

那你现在就先知道说,有一种network设计的架构,叫做**residual connection,它会把input直接跟output加起来,得到新的vector**

得到residual的结果以后,再把它做一件事情叫做normalization,这边用的不是batch normalization,这边用的叫做==layer normalization==



layer normalization做的事情,比bacth normalization更简单一点

输入一个向量 输出另外一个向量,不需要考虑batch,它会**把输入的这个向量,计算它的mean跟standard deviation**

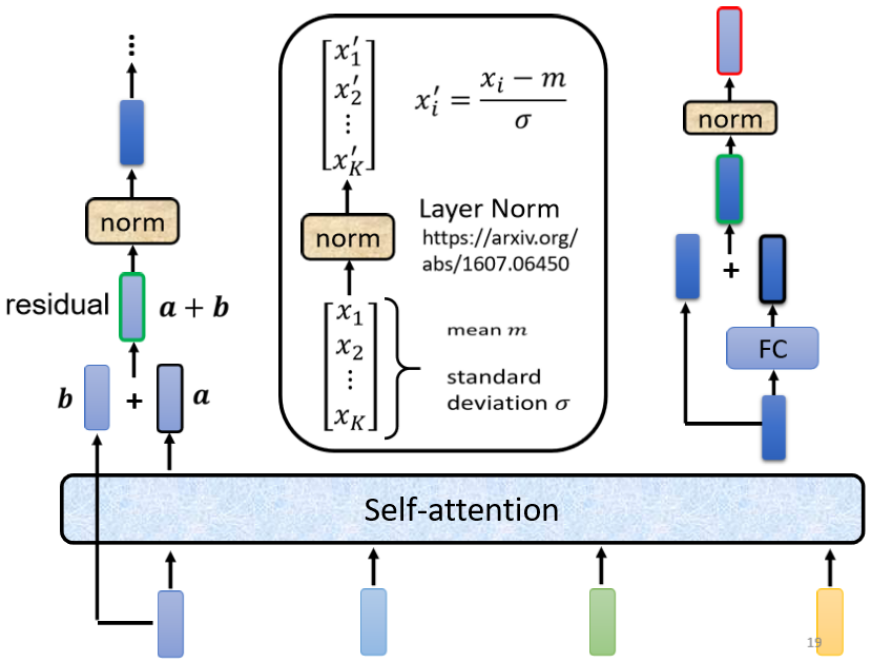
但是要注意一下,**==batch normalization==是对不同example,不同feature的同一个dimension,去计算mean跟standard deviation**

但**==layer normalization==,它是对同一个feature,同一个example裡面,不同的dimension,去计算mean跟standard deviation**

计算出mean,跟standard deviation以后,就可以做一个normalize,我们把input 这个vector裡面每一个,dimension减掉mean,再除以standard deviation以后得到x',就是layer normalization的输出

$$ x'_i=\frac{x_i-m}{\sigma}

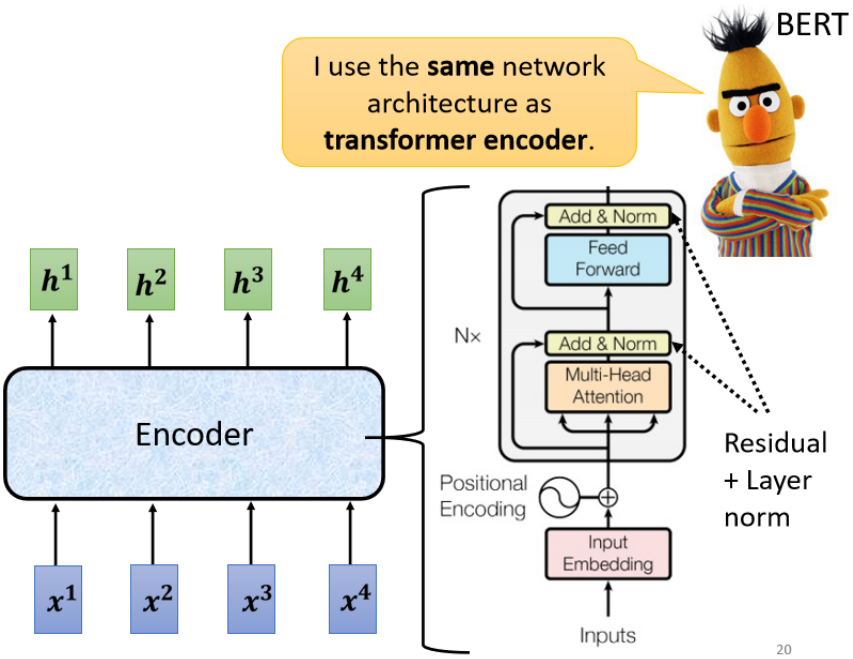
$$ 得到layer normalization的输出以后,它的这个输出 才是FC network的输入



而**FC network这边,也有residual的架构**,所以 我们会把FC network的input,跟它的output加起来 做一下residual,得到新的输出

这个FC network做完residual以后,还不是结束 你要把residual的结果,**再做一次layer normalization**,得到的输出,才是residual network裡面,一个block的输出,所以这个是挺复杂的

所以我们这边讲的 这一个图,其实就是我们刚才讲的那件事情



- 首先 你有self-attention,其实在input的地方,还有加上positional encoding,我们之前已经有讲过,如果你只光用self-attention,你没有未知的资讯,所以你需要加上positional的information,然后在这个图上,有特别画出positional的information
- Multi-Head Attention,这个就是self-attention的block,这边有特别强调说,它是Multi-Head的self-attention
- Add&norm,就是residual加layer normalization,我们刚才有说self-attention,有加上residual的connection,加下来还要过layer normalization,这边这个图上的Add&norm,就是residual加layer norm的意思
- 接下来,要过feed forward network
- fc的feed forward network以后再做一次Add&norm,再做一次residual加layer norm,才是一个block的输出,
- 然后这个block会重复n次,这个复杂的block,其实在之后会讲到的,一个非常重要的模型BERT裡面,会再用到 BERT,它其实就是transformer的encoder

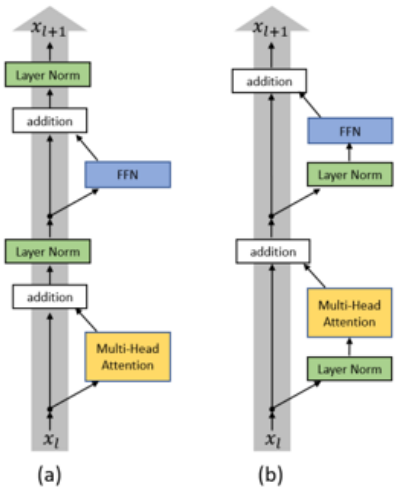
To Learn more

讲到这边 你心裡一定充满了问号,就是為什麼 transformer的encoder,要这样设计 不这样设计行不行?

行 不一定要这样设计,这个encoder的network架构,现在设计的方式,本文是按照原始的论文讲给你听的,但**原始论文的设计 不代表它是最好的,最optimal的设计**

- On Layer Normalization in the Transformer Architecture
- https://arxiv.org/abs/2002.04745

- PowerNorm: Rethinking Batch Normalization in Transformers
- https://arxiv.org/abs/2003.07845



- 有一篇文章叫,on layer normalization in the transformer architecture，它问的问题就是 为什麼,layer normalization是放在那个地方呢,為什麼我们是先做,residual再做layer normalization,能不能够把layer normalization,放到每一个block的input,也就是说 你做residual以后,再做layer normalization,再加进去 你可以看到说左边这个图,是原始的transformer,右边这个图是稍微把block,更换一下顺序以后的transformer,更换一下顺序以后 结果是会比较好的,这就代表说,原始的transformer 的架构,并不是一个最optimal的设计,你永远可以思考看看,有没有更好的设计方式
- 再来还有一个问题就是,為什麼是layer norm 為什麼是别的,不是别的,為什麼不做batch normalization,也许这篇paper可以回答你的问题,这篇paper是Power Norm：,Rethinking Batch Normalization In Transformers,它首先告诉你说 為什麼,batch normalization不如,layer normalization,在Transformers裡面為什麼,batch normalization不如,layer normalization,接下来在说,它提出来一个power normalization,一听就是很power的意思,都可以比layer normalization,还要performance差不多或甚至好一点

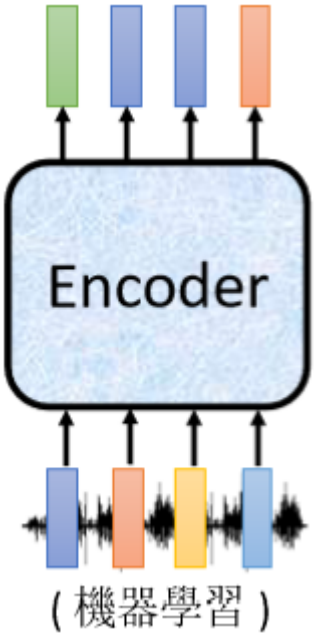
Transformer P2_Decoder

Decoder – Autoregressive (AT)

Decoder其实有两种,接下来会花比较多时间介绍,比较常见的 ==Autoregressive Decoder==,这个 Autoregressive 的 Decoder,是怎麼运作的

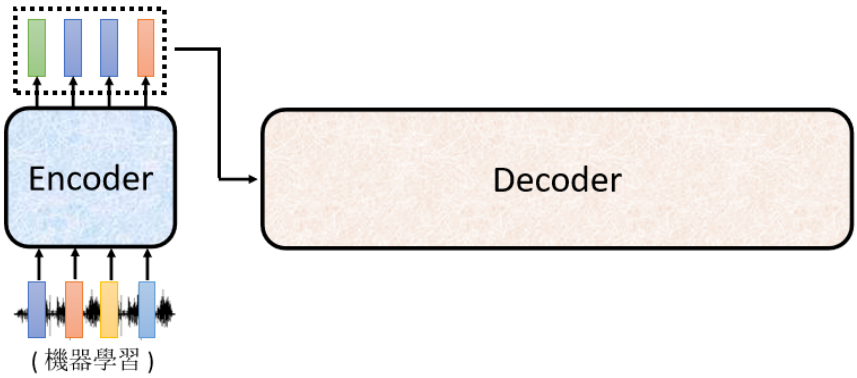
用**语音辨识**,来当作例子来说明,或用在作业裡面的**机器翻译**,其实是一模一样的,你只是把输入输出,改成不同的东西而已

语音辨识就是**输入一段声音,输出一串文字**,你会把一段声音输入给 Encoder,比如说你对机器说,机器学习,机器收到一段声音讯号,声音讯号 进入 Encoder以后,输出会是什麼,输出会变成一排 Vector



Encoder 做的事情,就是**输入一个 Vector Sequence,输出另外一个 Vector Sequence**

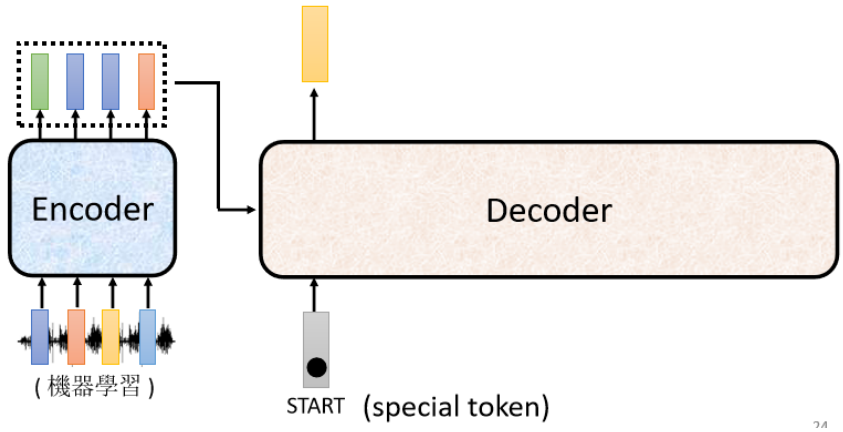
接下来,就轮到 Decoder 运作了,**Decoder 要做的事情就是產生输出**,也就是**產生语音辨识的结果**, Decoder 怎麼產生这个语音辨识的结果



Decoder 做的事情,就是**把 Encoder 的输出先读进去**,至於怎麼读进去,这个我们等一下再讲 我们先,你先假设 Somehow 就是有某种方法,把 Encoder 的输出读到 Decoder 裡面,这步我们等一下再处理

Decoder 怎麼產生一段文字

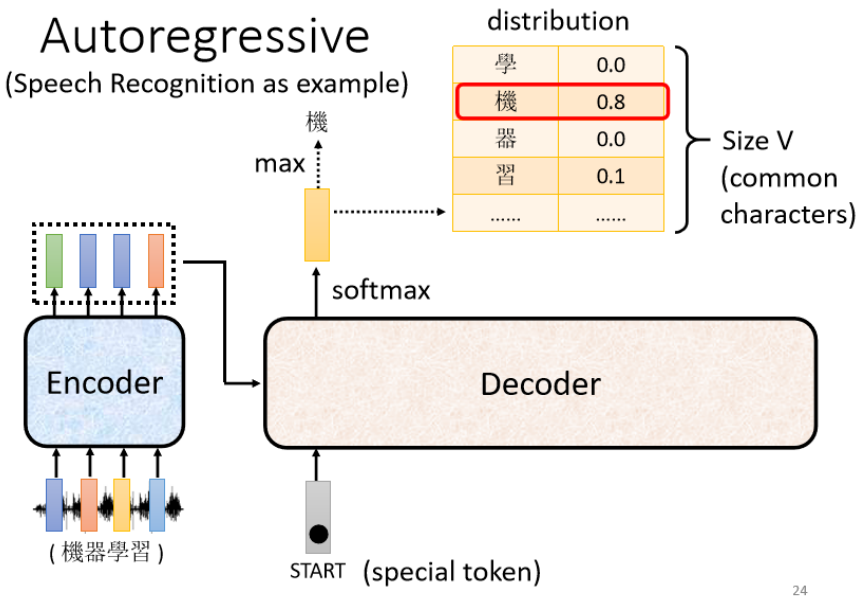
首先,你要先给它一个特殊的符号,这个特殊的符号,代表开始,在助教的投影片裡面,是写 Begin Of Sentence,缩写是 BOS



就是 Begin 的意思,这个是一个 Special 的 Token,你就是在你的个 Lexicon 裡面,你就在你可能,本来 Decoder 可能產生的文字裡面,多加一个特殊的字,这个字就代表了 BEGIN,代表了开始这个事情

在这个机器学习裡面,假设你要处理 NLP 的问题,**每一个 Token,你都可以把它用一个 One-Hot 的 Vector 来表示**,One-Hot Vector 就其中一维是 1,其他都是 0,所以 **BEGIN 也是用 One-Hot Vector 来表示**,其中一维是 1,其他是 0

接下来Decoder 会**吐出一个向量,这个 Vector 的长度很长,跟你的 Vocabulary 的 Size 是一样的**



==Vocabulary Size==则是什麼意思

你就先想好说,你的 Decoder **输出的单位**是什麼,假设我们今天做的是中文的语音辨识,我们 Decoder 输出的是中文,你这边的 Vocabulary 的 Size ,可能就是中文的方块字的数目

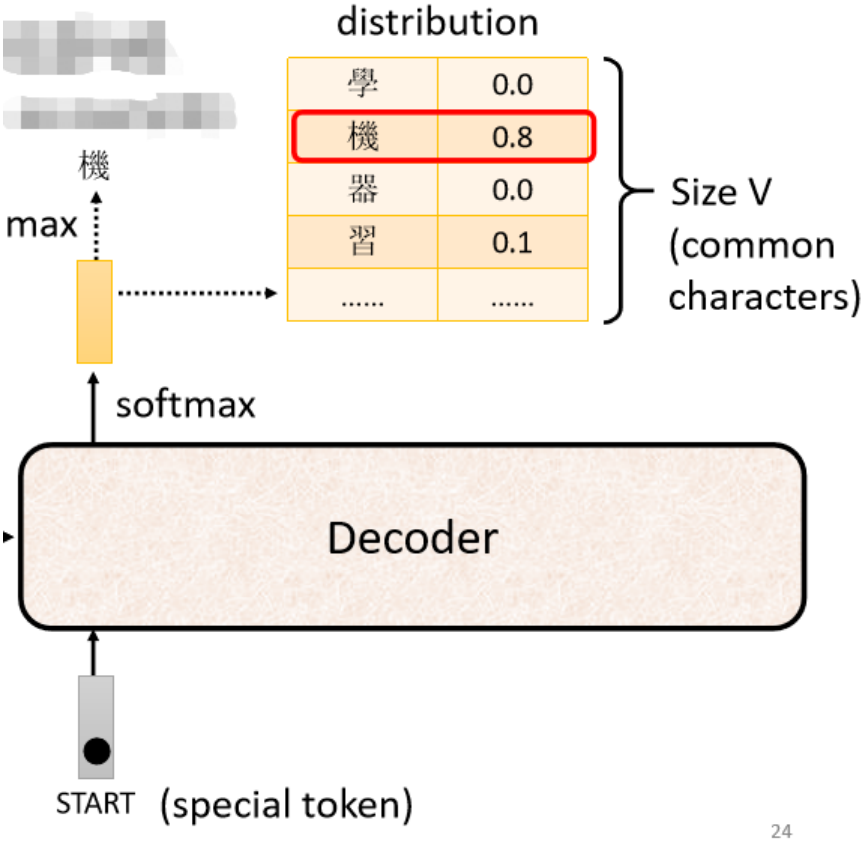
不同的字典,给你的数字可能是不一样的,常用的中文的方块字,大概两 三千个,一般人,可能认得的四 五千个,在更多都是罕见字 冷僻的字,所以你就看看说,你要让你的 Decoder,输出哪些可能的中文的方块字,你就把它列在这边

举例来说,你觉得这个 Decoder,能够输出常见的 3000 个方块字就好了,就把它列在这个地方,不同的语言,它输出的单位 不见不会不一样,这个取决於你对个语言的理解

比如说英文,你可以选择输出**字母的 A 到 Z**,输出英文的字母,但你可能会觉得字母这个单位太小了,有人可能会选择输出**英文的词汇**,英文的词汇是用空白作為间隔的,但如果都用词汇当作输出,又太多了

所以你会发现,刚才在助教的投影片裡面,助教说他是用 Subword 当作英文的单位,就有一些方法,可以把英文的字首字根切出来,拿字首字根当作单位,如果中文的话,我觉得就比较单纯,通常今天你可能就用中文的这个方块字,来当作单位

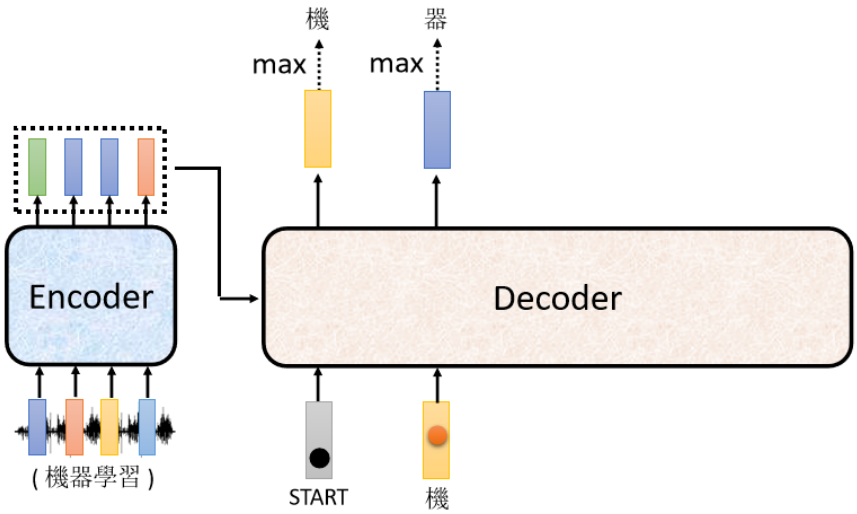
每一个中文的字,都会对应到一个数值,因為在**產生这个向量之前,你通常会先跑一个 Softmax**,就跟做分类一样,所以这一个向量裡面的分数,它是一个 Distribution,也就是,它这个向量裡面的值,它全部加起来,总和 会是 1



分数最高的一个中文字,它就是最终的输出

在这个例子裡面,机的分数最高,所以机,就当做是这个 Decoder 第一个输出

然后接下来,你**把“机”当做是 Decoder 新的 Input**,原来 Decoder 的 Input,只有 BEGIN 这个特别的符号,现在它除了 BEGIN 以外,它还有“机”作為它的 Input



所以 Decoder **现在它有两个输入**

- 一个是 **BEGIN** 这个符号
- 一个是**“机”**

根据这两个输入,它输出一个蓝色的向量,根据这个蓝色的向量裡面,给每一个中文的字的分数,我们会决定第二个输出，哪一个字的分数最高,它就是输出,假设"器"的分数最高,**"器"就是输出**

然后现在 Decoder

- 看到了 BEGIN
- 看到了"机"
- 看到了"器"

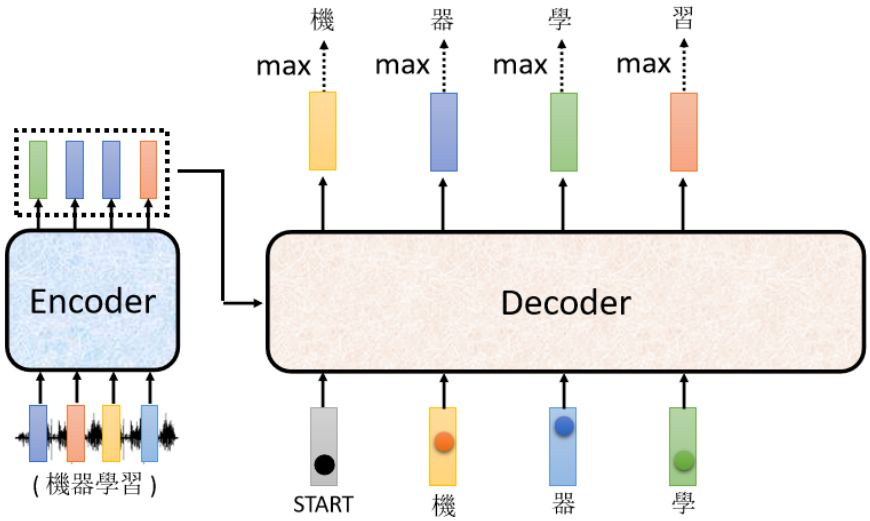
它接下来,还要再决定接下来要输出什麼,它可能,就输出"学",这一个过程就反覆的持续下去

所以现在 Decode

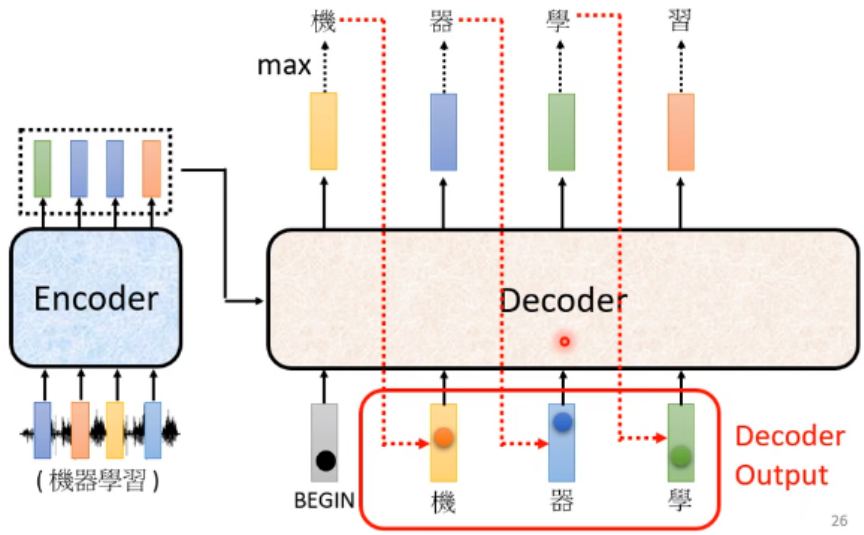
- 看到了 BEGIN
- 看到了"机"
- 看到了"器"
- 还有"学"

Encoder 这边其实也有输入,等一下再讲 Encoder 的输入,Decoder 是怎麼处理的,

所以 Decoder 看到 Encoder 这边的输入,看到"机" 看到"器" 看到"学",决定接下来输出一个向量,这个向量裡面,"习"这个中文字的分数最高的,所以它就输出"习"



然后这个 Process ,就反覆持续下去,这边有一个关键的地方,我们特别用红色的虚线把它标出来



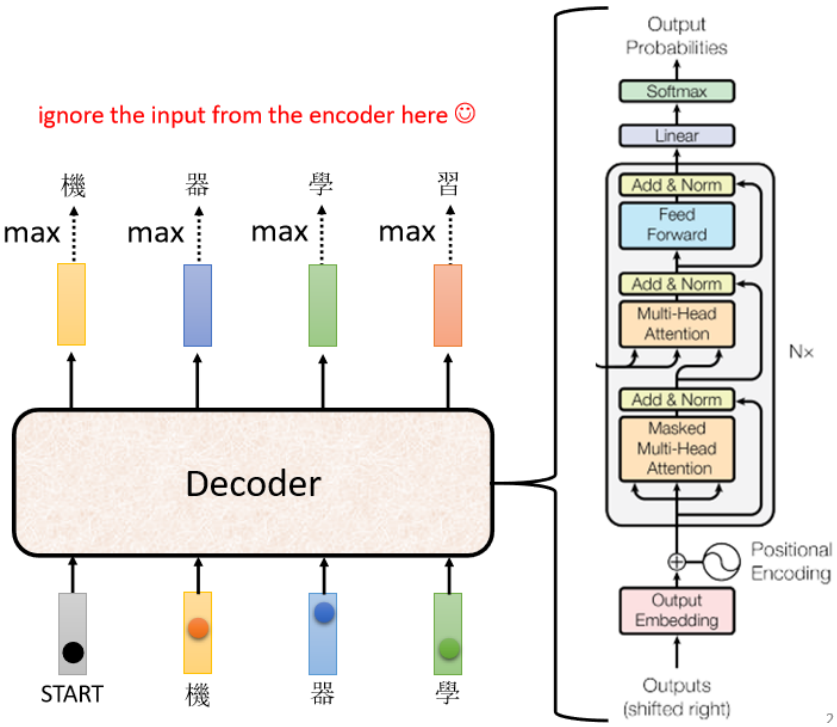
也就是说 Decoder 看到的输入,其实是它在前一个时间点,自己的输出,**Decoder 会把自己的输出,当做接下来的输入**

如果Decoder 看到**错误的输入**,让 Decoder 看到自己產生出来的错误的输入,再被 Decoder 自己吃进去,会不会造成 ==Error Propagation== 的问题

Error Propagation 的问题就是,**一步错 步步错**这样,就是在这个地方,如果不小心把机器的“器”,不小心写成天气的"气",会不会接下来就整个句子都坏掉了,都没有办法再產生正确的词汇了?

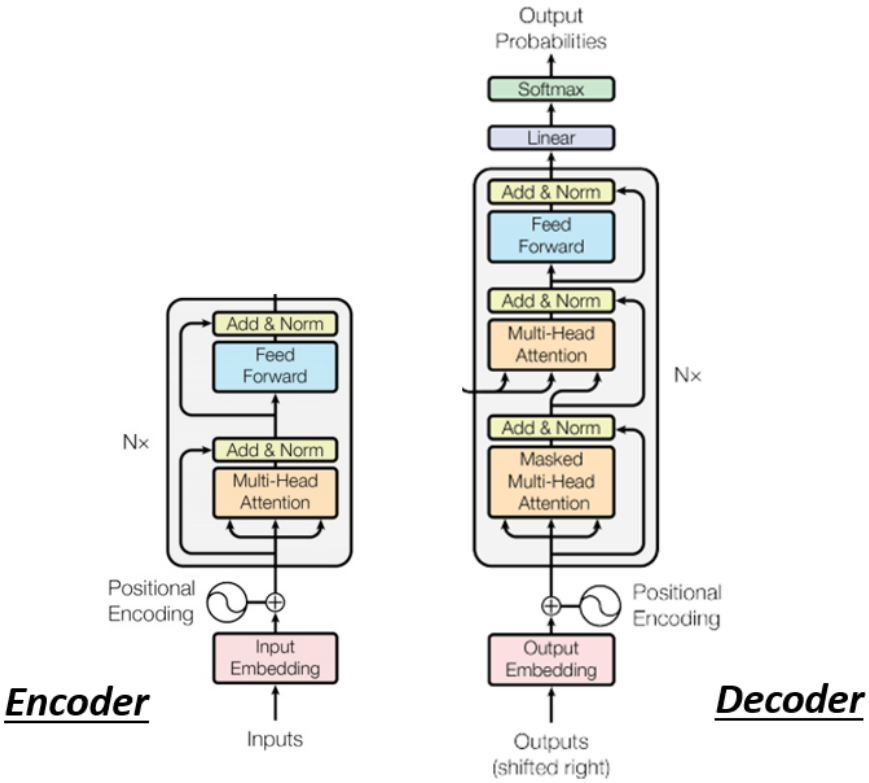
有可能,这个等一下,我们最后会稍微讲一下,这个问题要怎麼处理,我们现在,先无视这个问题,继续走下去

我们来看一下这个 **Decoder内部的结构**长什麼样子

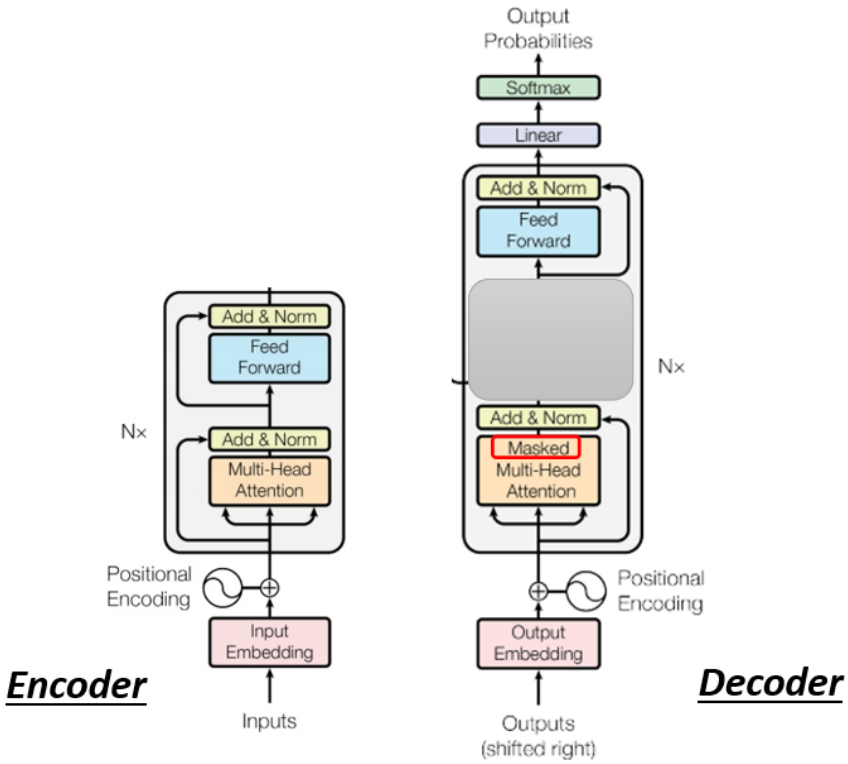


那我们这边,**把 Encoder 的部分先暂时省略掉**,那在 Transformer 裡面,Decoder 的结构,长得是这个样子的,看起来有点复杂,比 Encoder 还稍微复杂一点,

那我们现在先把 Encoder 跟 Decoder 放在一起



稍微比较一下它们之间的差异,那你会发现说,如果我们把 Decoder 中间这一块,**中间这一块把它盖起来,其实 Encoder 跟 Decoder,并没有那麼大的差别**

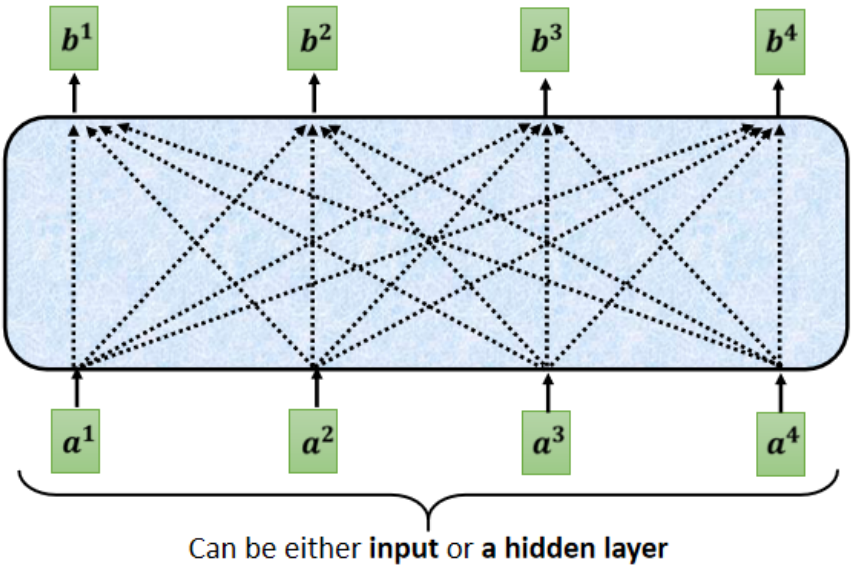


你看 Encoder 这边,Multi-Head Attention,然后 Add & Norm,Feed Forward,Add & Norm,重复 N 次,Decoder 其实也是一样

当我们把中间这一块遮起来以后,我们等一下再讲,遮起来这一块裡面做了什麼事,但当我们把中间这块遮起来以后,欸 那 Decoder 也是,有一个 Multi-Head Attention,Add & Norm,然后 Feed Forward,然后 Add & Norm,所以 Encoder 跟 Decoder,其实并没有非常大的差别,除了中间这一块不一样的地方,

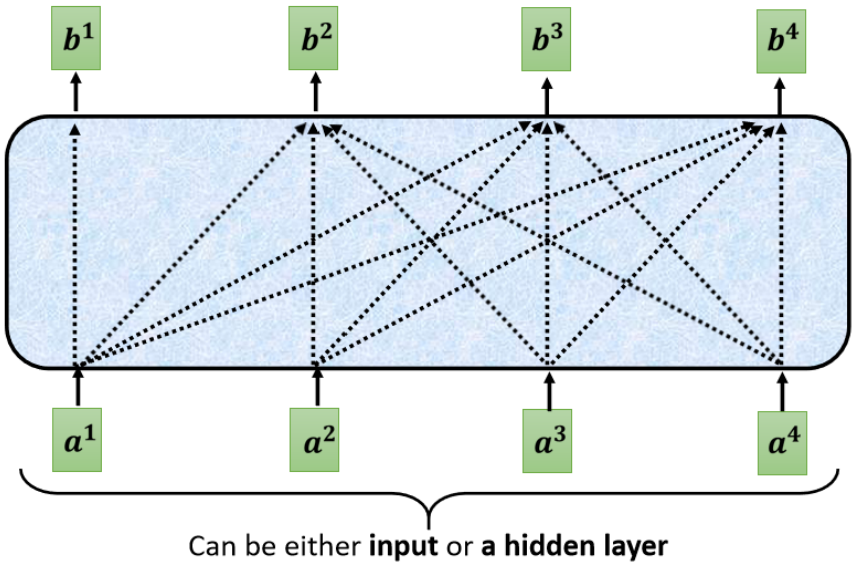
那只是最后,我们可能会再做一个 Softmax,使得它的输出变成一个机率,那这边有一个**稍微不一样的地方**是,在 Decoder 这边,Multi-Head Attention 这一个 Block 上面,还**加了一个 ==Masked==**,

这个 Masked 的意思是这样子的,这是我们原来的 Self-Attention

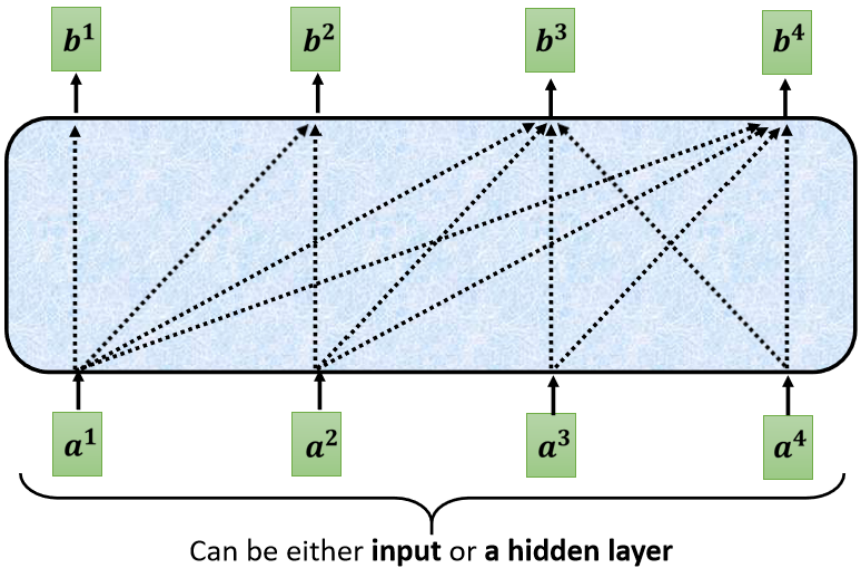


Input 一排 Vector,Output 另外一排 Vector,这一排 Vector **每一个输出**,都要看过完整的 Input 以后,才做决定,所以输出 b^1 的时候,其实是根据 a^1 到 a^4 所有的资讯,去输出 b^1

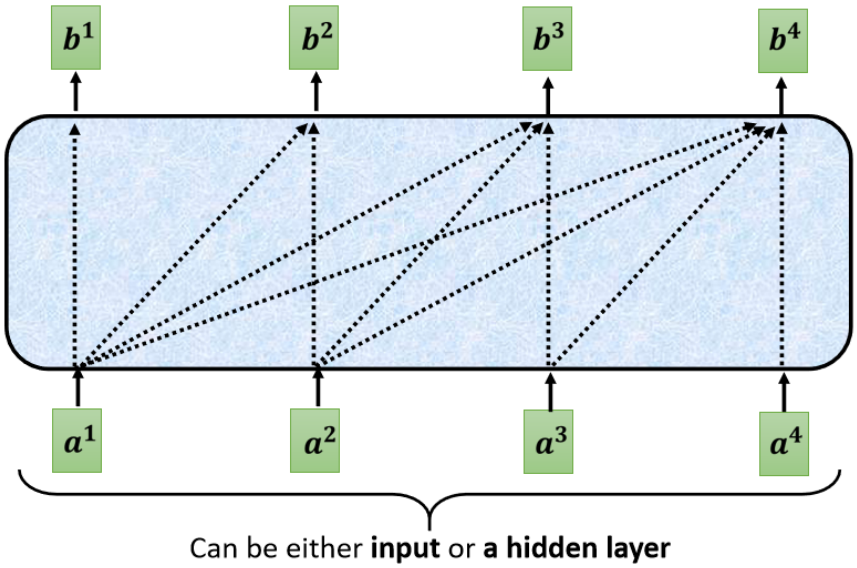
当我们把 Self-Attention,转成 Masked Attention 的时候,它的不同点是,现在我们不能再看右边的部分,也就是產生 b^1 的时候,我们只能考虑 a^1 的资讯,你不能够再考虑 a^2 a^3 a^4



產生 b^2 的时候,你只能考虑 a^1 a^2 的资讯,不能再考虑 a^3 a^4 的资讯

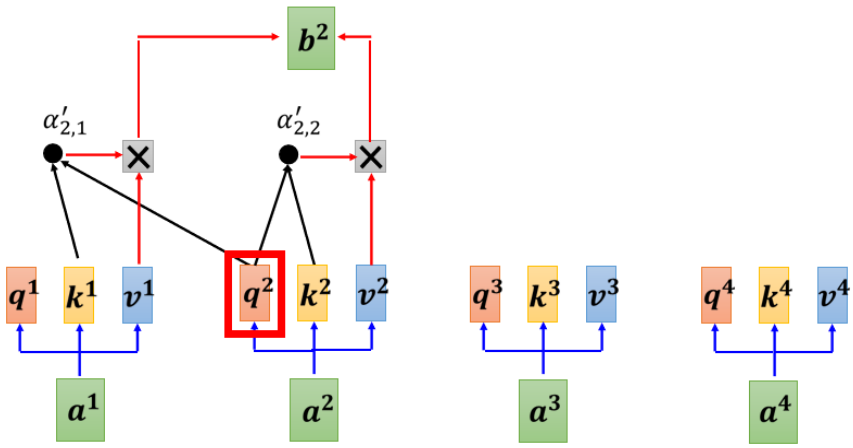


產生 b^3 的时候,你就不能考虑 a^4 的资讯,



產生 b^4 的时候,你可以用整个 Input Sequence 的资讯,这个就是 Masked 的 Self-Attention,

讲得更具体一点,你做的事情是,当我们要產生 b^2 的时候,我们只拿第二个位置的 Query b^2,去跟第一个位置的 Key,和第二个位置的 Key,去计算 Attention,第三个位置跟第四个位置,就不管它,不去计算 Attention

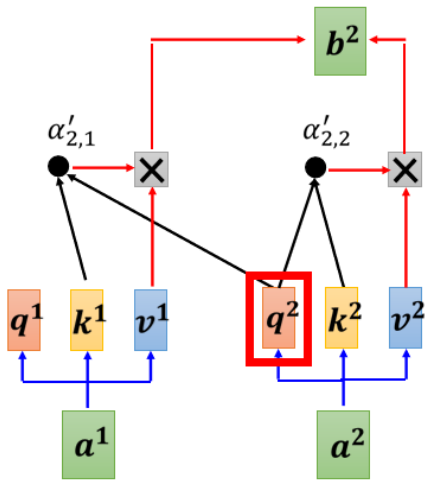


我们这样子不去管这个 a^2 右边的地方,只考虑 a^1 跟 a^2,只考虑 q^1 q^2,只考虑 k^1 k^2,q^2 只跟 k^1 跟 k^2 去计算 Attention,然后最后只计算 b^1 跟 b^2 的 Weighted Sum

然后当我们输出这个 b^2 的时候,b^2 就只考虑了 a^1 跟 a^2,就没有考虑到 a^3 跟 a^4

那為什麼会这样,為什麼需要加 Masked

Self-attention → Masked Self-attention



Why masked? Consider how does decoder work

这件事情其实非常地直觉:我们一开始 Decoder 的运作方式,它是**一个一个输出**,所以是先有 a^1 再有 a^2,再有 a^3 再有 a^4

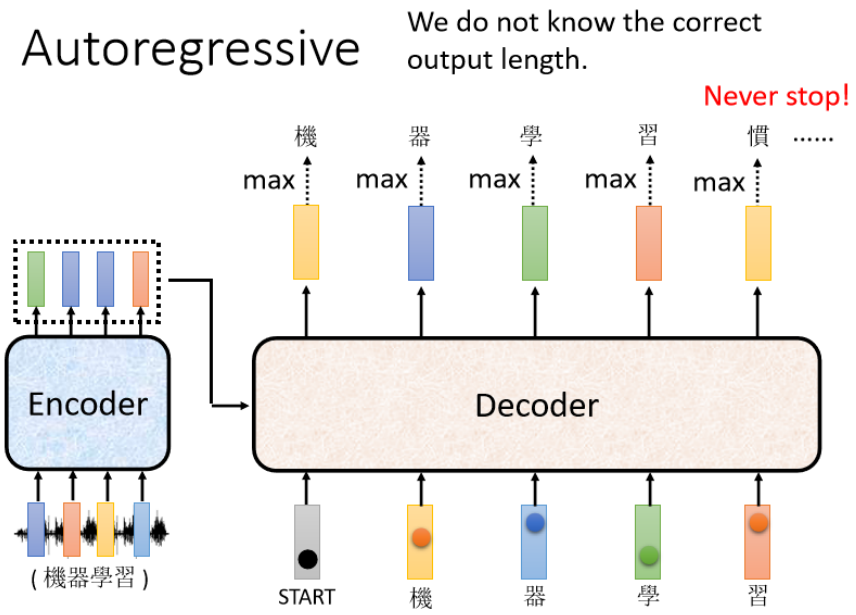
这跟原来的 Self-Attention 不一样,原来的 Self-Attention,a^1 跟 a^4 是一次整个输进去你的 Model 裡面的,在我们讲 Encoder 的时候,Encoder 是一次把 a^1 跟 a^4,都整个都读进去

但是对 Decoder 而言,先有 a^1 才有 a^2,才有 a^3 才有 a^4,所以实际上,当你有 a^2,你要计算 b^2 的时候,你是没有 a^3 跟 a^4 的,所以你根本就没有办法把 a^3 a^4 考虑进来

所以这就是為什麼,在那个 Decoder 的那个图上面,Transformer 原始的 Paper 特别跟你强调说,**那不是一个一般的 Attention,这是一个 Masked 的 Self-Attention**,意思只是想要告诉你说,Decoder 它的 Tokent,**它输出的东西是一个一个產生的,所以它只能考虑它左边的东西,它没有办法考虑它右边的东西**

讲了 Decoder 的运作方式,但是这边,还有一个非常关键的问题,**Decoder 必须自己决定,输出的 Sequence 的长度**

可是到底输出的 Sequence 的长度应该是多少,我们不知道



你没有办法轻易的从输入的 Sequence 的长度,就知道输出的 Sequence 的长度是多少,并不是说,输入是 4 个向量,输出一定就是 4 个向量

这边在这个例子裡面,输入跟输出的长度是一样的,但是你知道实际上在你真正的应用裡面,并不是这样,输入跟输出长度的关係,是非常复杂的,我们其实是期待机器可以自己学到,今天给它一个 Input Sequence 的时候,Output 的 Sequence 应该要多长

但在我们目前的这整个 Decoder的这个运作的机制裡面,**机器不知道它什麼时候应该停下来**,它產生完习以后,它还可以继续重复一模一样的 Process,就把习,当做输入,然后也许 Decoder ,就会接一个惯,然后接下来,就一直持续下去,**永远都不会停下来**

这就让我想到推文接龙

推文接龍 (Tweet Solitaire)

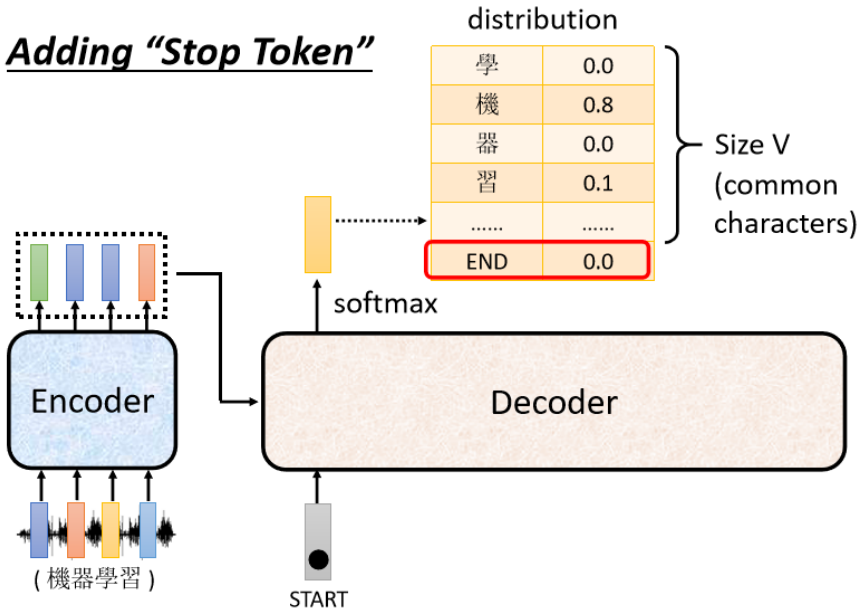


我不知道大家知不到这是什麼,这是一个这个古老的民俗传统,流传在 PTT 上面,这个民俗传统是怎麼运作的,就有一个人,先推一个中文字,然后推一个超,然后接下来,就会有另外一个乡民,去推另外一个字,然后可以接上去的,所以就可以產生一排的词汇

啦,一排字啦,就是超人正大中天外飞仙草,不知道在说些什麼,这个是 Process ,可以持续好几个月,都不停下来,我也不知道為什麼,那怎麼让这个 Process 停下来,那要怎麼让它停下来

要有人冒险去推一个断,推个断,它就停下来了

所以我们要让 Decoder 做的事情,也是一样,要让它可以输出一个断,所以你要**特别準备一个特别的符号**,这个符号,就叫做断,我们这边,用 END 来表示这个特殊的符号



所以除了所有中文的方块字,还有 BEGIN 以外,你还要**準备一个特殊的符号,叫做"断"**,那其实在助教的程式裡面,它是把 BEGIN 跟 END,就是开始跟这个断,用同一个符号来表示

反正这个,这个 BEGIN 只会在输入的时候出现,断只会在输出的时候出现,所以在助教的程式裡面,如果你仔细研究一下的话,会发现说 END 跟 BEGIN,用的其实是同一个符号,但你用不同的符号,也是完全可以的,也完全没有问题

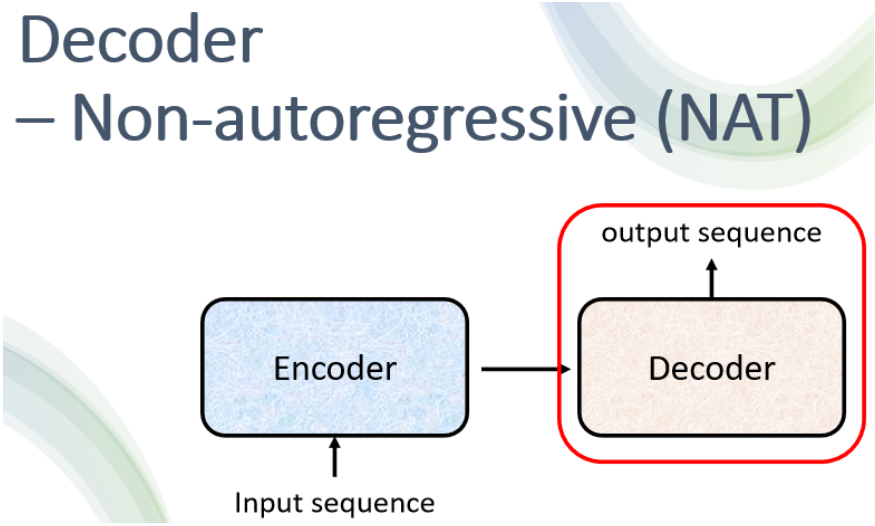
所以我们现在,当把"习"当作输入以后,就 Decoder 看到 Encoder 输出的这个 Embedding,看到了 "BEGIN",然后"机" "器" "学" "习"以后,看到这些资讯以后 它要知道说,这个语音辨识的结果已经结束了,不需要再產生更多的词汇了

它產生出来的向量END,就是断的那个符号,它的机率必须要是最大的,然后你就输出断这个符号,那整个运作的过程,整个 Decoder 產生 Sequence 的过程,就结束了

这个就是 ==Autoregressive Decoder==,它运作的方式

Decoder – Non-autoregressive (NAT)

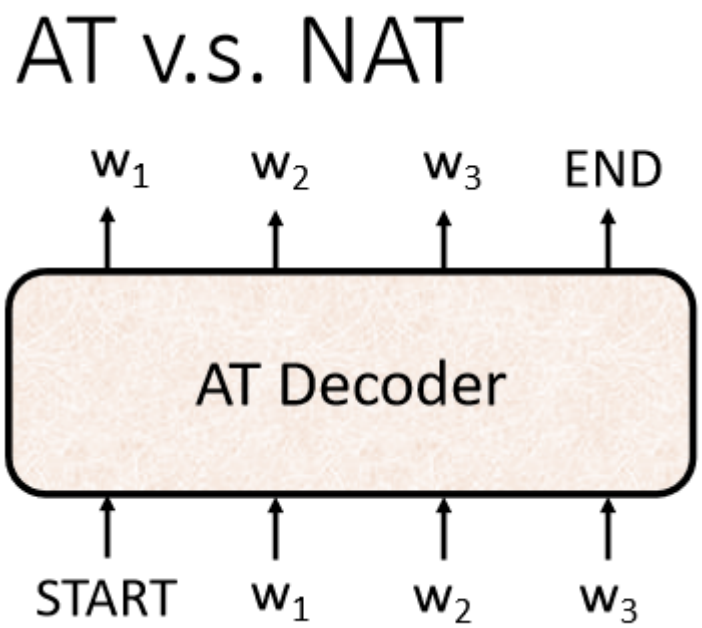
用两页投影片,非常简短地讲一下,Non-Autoregressive 的 Model



Non-Autoregressive ,通常缩写成 NAT,所以有时候 Autoregressive 的 Model,也缩写成 AT,Non-Autoregressive 的 Model 是怎麼运作的

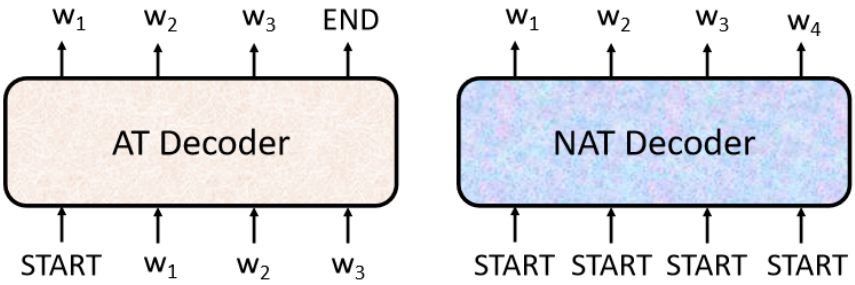
AT v.s. NAT

这个 ==Autoregressive== 的 Model 是



先输入 BEGIN,然后出现 w1,然后**再**把 w1 当做输入,**再**输出 w2,**直到输出 END 為止**

那 ==NAT== 是这样,它**不是依次產生**

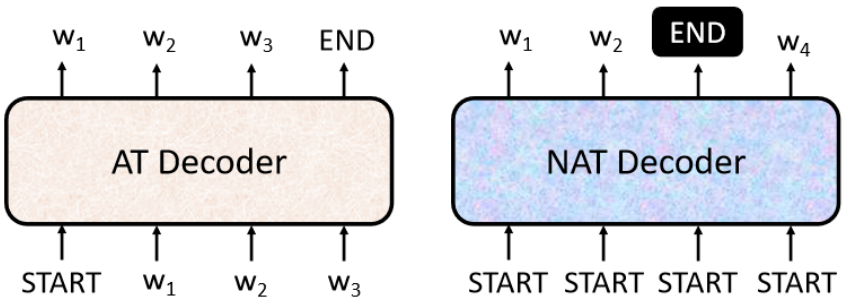


就假设我们现在產生是中文的句子,它不是依次產生一个字,它是**一次把整个句子都產生出来**

NAT 的 Decoder**可能吃的是一整排的 BEGIN 的 Token**,你就把一堆一排 BEGIN 的 Token 都丢给它,让它一次產生一排 Token 就结束了

举例来说,如果你丢给它 4 个 BEGIN 的 Token,它就產生 4 个中文的字,变成一个句子,就结束了，所以它只要一个步骤,就可以完成句子的生成

这边你可能会问一个问题：刚才不是说不知道输出的长度应该是多少吗,那我们这边**怎麼知道 BEGIN 要放多少个**,当做 NAT Decoder 的收入?



- How to decide the output length for NAT decoder?
  - Another predictor for output length
  - Output a very long sequence, ignore tokens after END

没错 这件事没有办法很自然的知道,没有办法很直接的知道,所以有几个,所以有几个做法

- 一个做法是,你**另外learn一个 Classifier**,这个 Classifier ,它吃 Encoder 的 Input,然后输出是一个数字,这个数字代表 Decoder 应该要输出的长度,这是一种可能的做法
- 另一种可能做法就是,你就不管三七二十一,**给它一堆 BEGIN 的 Token**,你就假设说,你现在输出的句子的长度,绝对不会超过 300 个字,你就假设一个句子长度的上限,然后 BEGIN ,你就给它 300 个 BEGIN,然后就会输出 300 个字嘛,然后,你再看看**什麼地方输出 END**,输出 END 右边的,就当做它没有输出,就结束了,这是另外一种处理 NAT 的这个 Decoder,它应该输出的长度的方法

那 NAT 的 Decoder,它有什麼样的**好处**,

- Advantage: parallel, more stable generation (e.g., TTS)

- 它第一个好处是,**并行化**,这个 AT 的 Decoder,它在输出它的句子的时候,是一个一个一个字產生的,所以你有你的,假设要输出长度一百个字的句子,那你就需要做一百次的 Decode

但是 NAT 的 Decoder 不是这样,不管句子的长度如何,都是**一个步骤就產生出完整的句子**,所以在速度上,NAT 的 Decoder 它会跑得比,AT 的 Decoder 要快,那你可以想像说,这个 NAT Decoder 的想法显然是在,由这个 Transformer 以后,有这种 Self-Attention 的 Decoder 以后才有的

因為以前如果你是用那个 ==LSTM==,用 ==RNN== 的话,那你就算给它一排 BEGIN,它也没有办法同时產生全部的输出,它的输出还是一个一个產生的,所以在没有这个 Self-Attention 之前,只有 RNN,只有 LSTM 的时候,根本就不会有人想要做什麼 NAT 的 Decoder,不过自从有了 Self-Attention 以后,那 NAT 的 Decoder,现在就算是一个热门的研究的主题了

- 那 NAT 的 Decoder 还有另外一个好处就是,你比较能够**控制它输出的长度**,举语音合成為例,其实在语音合成裡面,NAT 的 Decoder 算是非常常用的,它并不是一个什麼稀罕 罕见的招数

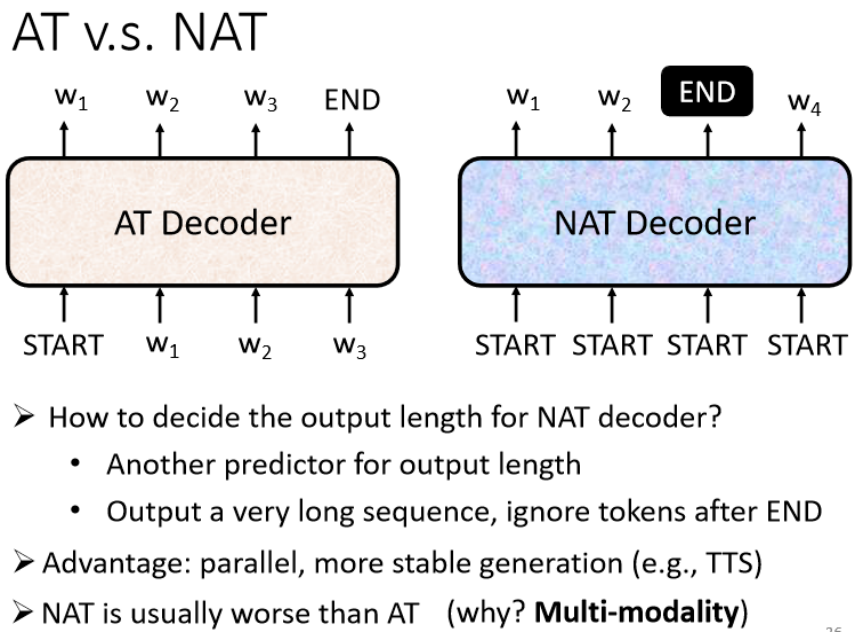
  比如说有,所以语音合成今天你都可以用,Sequence To Sequence 的模型来做,那最知名的,是一个叫做 ==Tacotron== 的模型,那它是 AT 的 Decoder

  那有另外一个模型叫 ==FastSpeech==,那它是 NAT 的 Decoder,那 NAT 的 Decoder 有一个好处,就是你可以控制你输出的长度,那我们刚才说怎麼决定,NAT 的 Decoder 输出多长

  你可能有一个 Classifier,决定 NAT 的 Decoder 应该输出的长度,那如果在做语音合成的时候,假设你现在突然想要让你的系统讲快一点,加速,那你就把那个 Classifier 的 Output 除以二,它讲话速度就变两倍快,然后如果你想要这个讲话放慢速度,那你就把那个 Classifier 输出的那个长度,它 Predict 出来的长度乘两倍,那你的这个 Decoder ,说话的速度就变两倍慢

  所以你可以如果有这种 NAT 的 Decoder,那你有 Explicit 去 Model,Output 长度应该是多少的话,你就比较有机会去控制,你的 Decoder 输出的长度应该是多少,你就可以做种种的变化

NAT 的 Decoder,最近它之所以是一个热门研究主题,就是它虽然表面上看起来有种种的厉害之处,尤其是平行化是它最大的优势,但是 **NAT 的 Decoder ,它的 Performance,往往都不如 AT 的 Decoder**



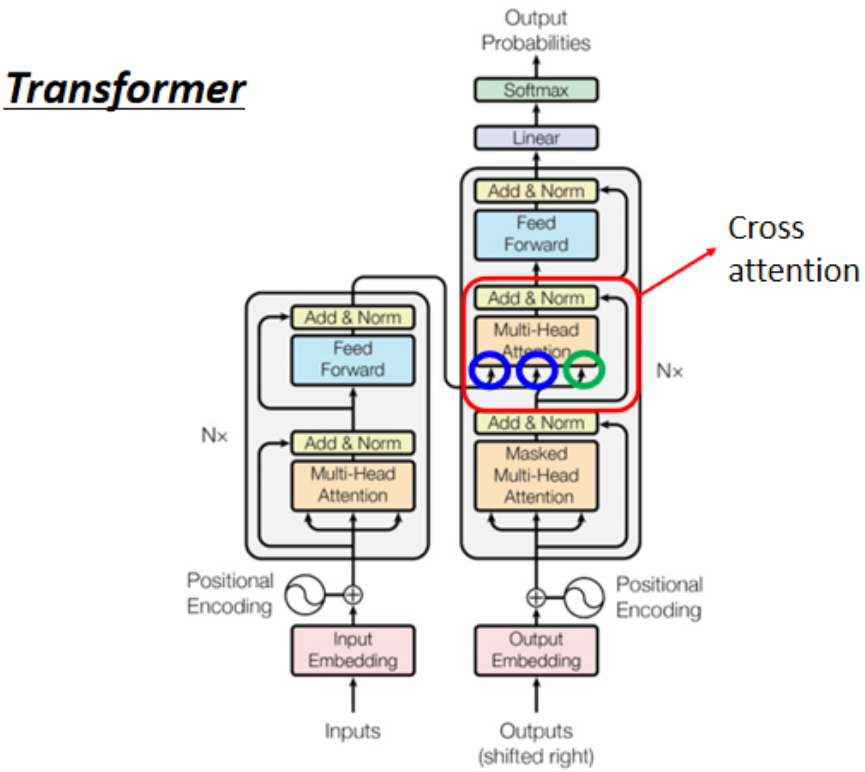
所以发现有很多很多的研究试图让,NAT 的 Decoder 的 Performance 越来越好,试图去逼近 AT 的 Decoder,不过今天你要让 NAT 的 Decoder,跟 AT 的 Decoder Performance 一样好,你**必须要用非常多的 Trick** 才能够办到,就 AT 的 Decoder 随便 Train 一下,NAT 的 Decoder 你要花很多力气,才有可能跟 AT 的 Performance 差不多

為什麼 NAT 的 Decoder Performance 不好,有一个问题我们今天就不细讲了,叫做 ==Multi-Modality== 的问题,那如果你想要这个深入了解 NAT,那就把之前上课,助教这个上课补充的内容,连结https://youtu.be/jvyKmU4OM3c放在这边给大家参考



Encoder-Decoder

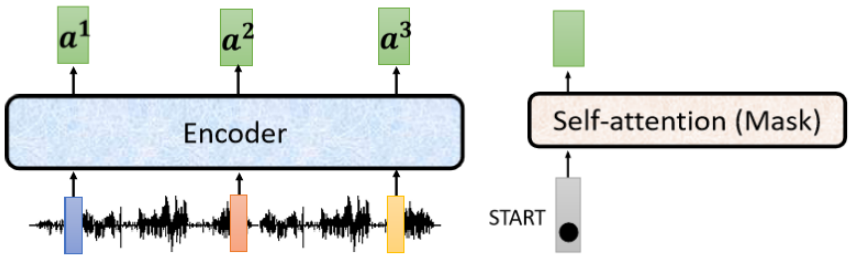
接下来就要讲**Encoder 跟 Decoder它们中间是怎麼传递资讯**的了,也就是我们要讲,刚才我们刻意把它遮起来的那一块



这块叫做 ==Cross Attention==,它是连接 Encoder 跟 Decoder 之间的桥樑,那这一块裡面啊,会发现有**两个输入来自於 Encoder**,Encoder 提供两个箭头,然后 **Decoder 提供了一个箭头**,所以从左边这两个箭头,Decoder 可以读到 Encoder 的输出

那这个模组实际上是怎麼运作的呢,那我们就实际把它运作的过程跟大家展示一下

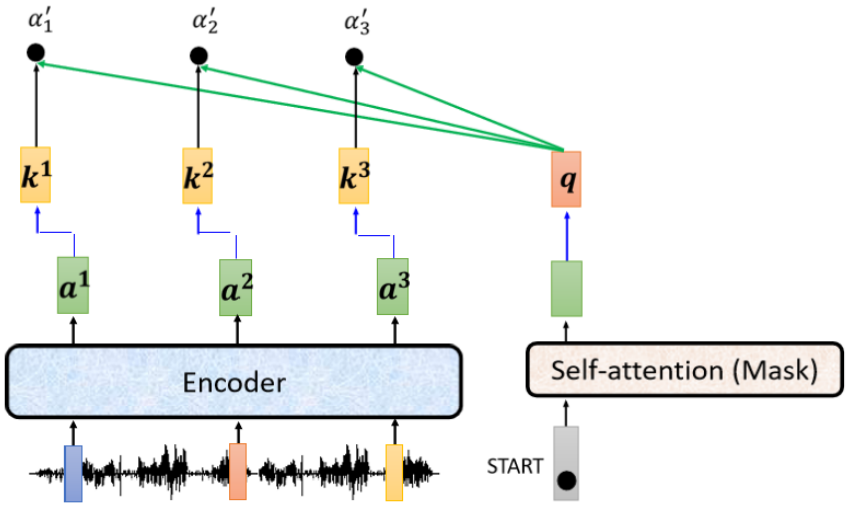
这个是你的 Encoder



输入一排向量,输出一排向量,我们叫它 $a^1 a^2 a^3$

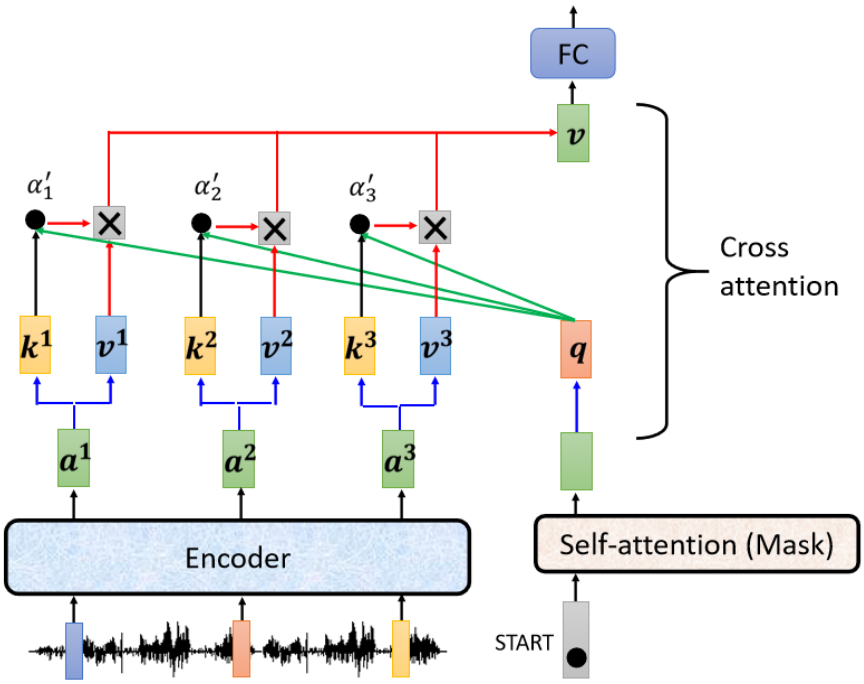
接下来 轮到你的 Decoder,你的 Decoder 呢,会先吃 BEGIN 当做,BEGIN 这个 Special 的 Token,那 BEGIN 这个 Special 的 Token 读进来以后,你可能会经过 Self-Attention,这个 Self-Attention 是有做 Mask 的,然后得到一个向量,就是 Self-Attention 就算是有做 Mask,还是一样输入多少长度的向量,输出就是多少向量

所以输入一个向量 输出一个向量,然后接下来把这个向量呢,乘上一个矩阵做一个 Transform,得到一个 Query 叫做 q



然后这边的 $a^1 a^2 a^3$ 呢,也都產生 Key,Key1 Key2 Key3,那把这个 q 跟 $k^1 k^2 k^3$,去计算 Attention 的分数,得到 $α_1 α_2 α_3$,当然你可能一样会做 Softmax,把它稍微做一下 Normalization,所以我这边加一个 ',代表它可能是做过 Normalization

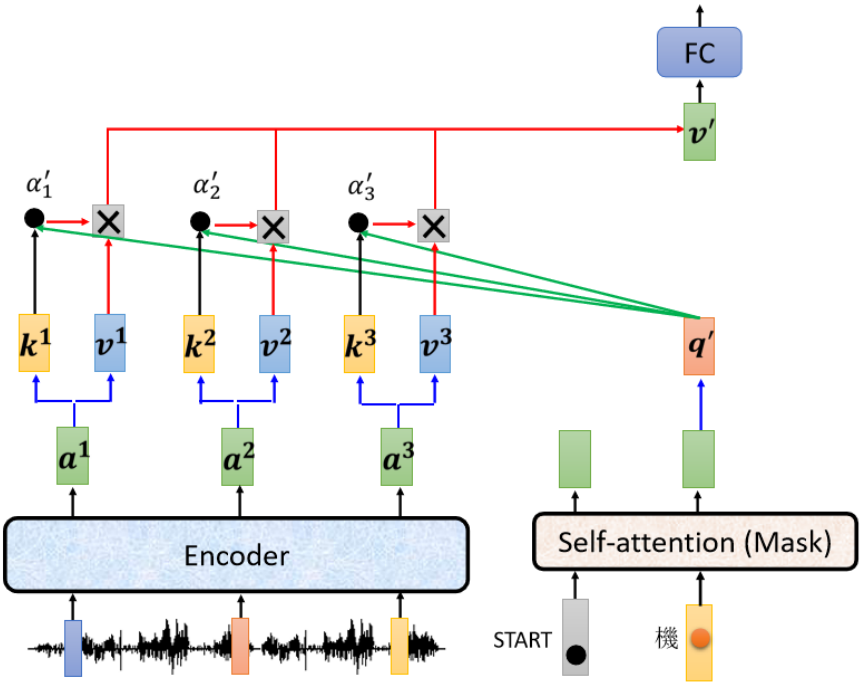
接下来再把 $α_1 α_2 α_3$,就乘上 $v^1 v^2 v^3$,再把它 Weighted Sum 加起来会得到 v



那这一个 V,就是接下来会丢到 Fully-Connected 的,Network 做接下来的处理,那这个步骤就是 q 来自於 Decoder,k 跟 v 来自於 Encoder,这个步骤就叫做 Cross Attention

所以 **Decoder 就是凭藉著產生一个 q,去 Encoder 这边抽取资讯出来,当做接下来的 Decoder 的,Fully-Connected 的 Network 的 Input**

当然这个,就现在假设產生第二个,第一个这个中文的字產生一个“机”,接下来的运作也是一模一样的



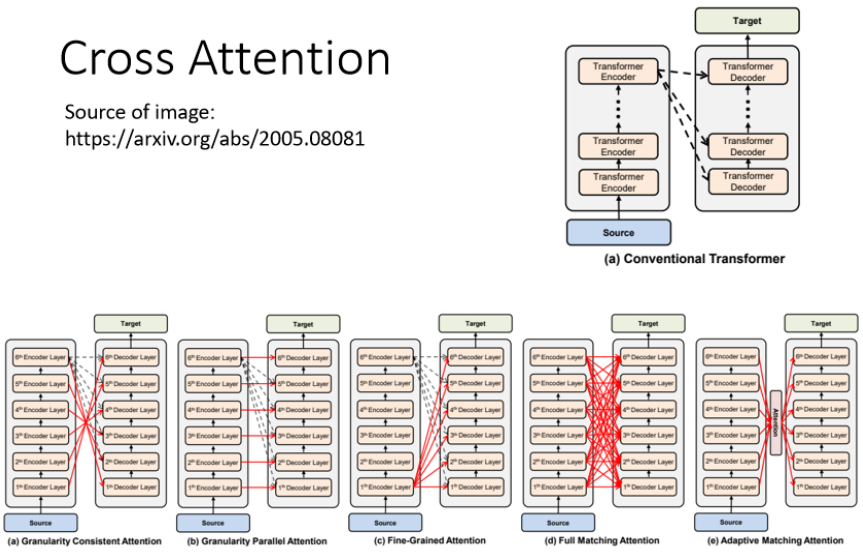
输入 BEGIN 输入机,產生一个向量,这个向量一样乘上一个 Linear 的 Transform,得到 q',得到一个 Query,这个 Query 一样跟 $k^1 k^2 k^3$,去计算 Attention 的分数,一样跟 $v^1 v^2 v^3$ 做 Weighted Sum 做加权,然后加起来得到 v',交给接下来 Fully-Connected Network 做处理

所以这就是Cross Attention 的运作的过程

也许有人会有**疑问**：那这个 Encoder 有很多层啊,Decoder 也有很多层啊,从刚才的讲解裡面好像听起来,这个 Decoder 不管哪一层,都是**拿 Encoder 的最后一层的输出**这样对吗?

对,**在原始 Paper 裡面的实做是这样子**,那一定要这样吗

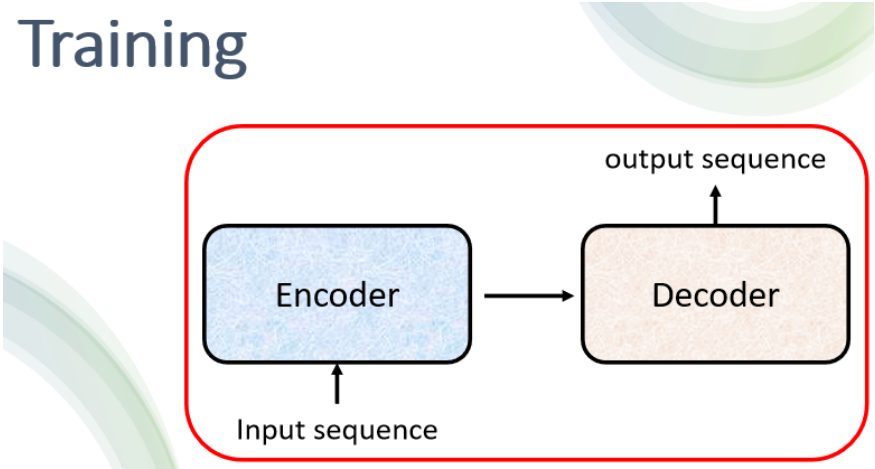
不一定要这样,你永远可以自己兜一些新的想法,所以我这边就是引用一篇论文告诉你说,也有人尝试不同的 Cross Attension 的方式



Encoder 这边有很多层,Decoder 这边有很多层,為什麼 Decoder 这边每一层都一定要看,Encoder 的最后一层输出呢,能不能够有各式各样不同的连接方式,这完全可以当做一个研究的问题来 Study

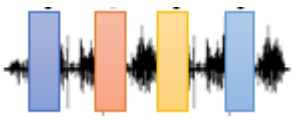
Training

已经清楚说 Input 一个 Sequence,是怎麼得到最终的输出,那接下来就进入训练的部分



刚才讲的都还只是,假设你模型训练好以后它是怎麼运作的,它是怎麼做 Testing 的,它是怎麼做 Inference 的,Inference 就是 Testing， 那是怎麼做训练的呢?

接下来就要讲怎麼做训练,那如果是做语音辨识,那你要有**训练资料**,你要收集一大堆的声音讯号,每一句声音讯号都要有工读生来听打一下,打出说它的这个对应的词汇是什麼

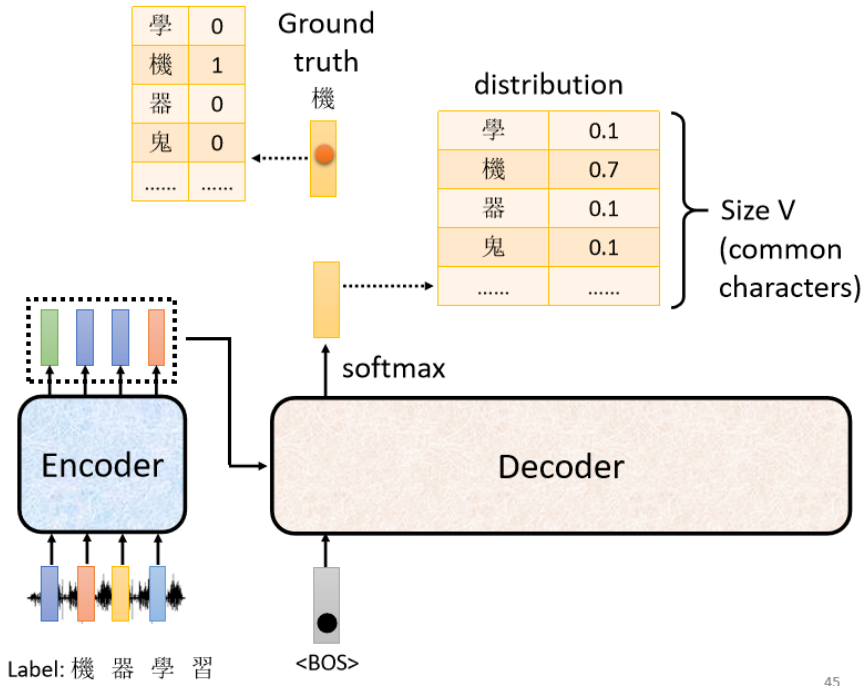


Label: 機 器 學 習

工读生听这段是机器学习,他就把机器学习四个字打出来,所以就知道说你的这个 Transformer,应该要学到 听到这段声音讯号,它的输出就是机器学习这四个中文字

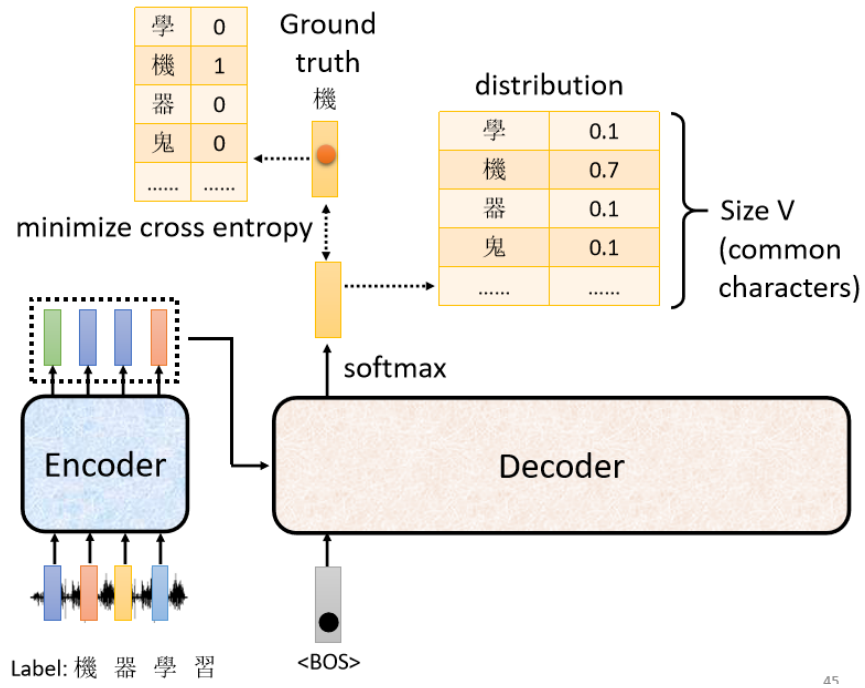
那怎麼让机器学到这件事呢

我们已经知道说输入这段声音讯号,第一个应该要输出的中文字是“机”,所以今天当我们把 BEGIN,丢给这个 Encoder 的时候,它第一个输出应该要跟“机”越接近越好



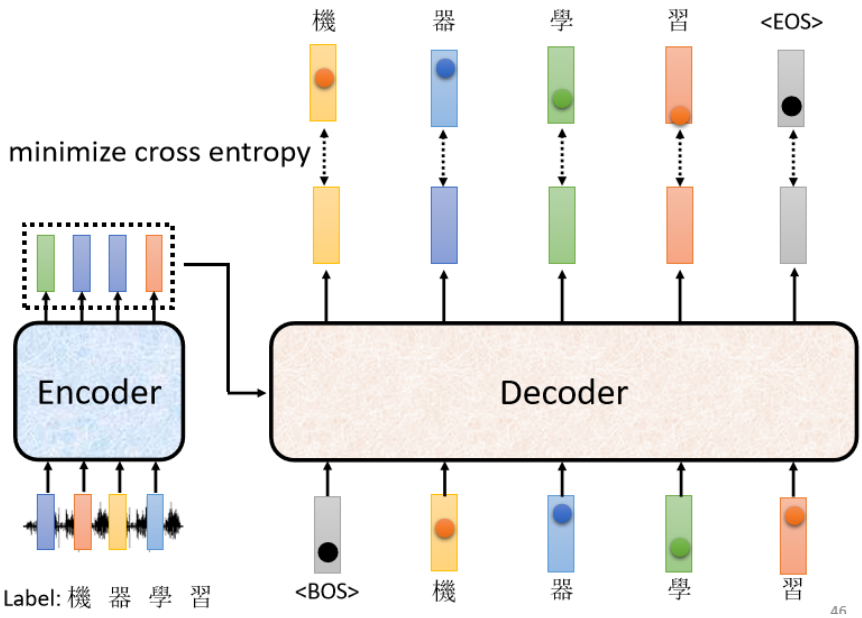
“机”这个字会被表示成一个 One-Hot 的 Vector,在这个 Vector 裡面,只有机对应的那个维度是 1,其他都是 0,这是正确答案,那我们的 Decoder,它的输出是一个 Distribution,是一个机率的分布,我们会希望这一个机率的分布,跟这个 One-Hot 的 Vector 越接近越好

所以你会去计算这个 Ground Truth,跟这个 Distribution 它们之间的 Cross Entropy,然后我们希望这个 ==Cross Entropy== 的值,越小越好



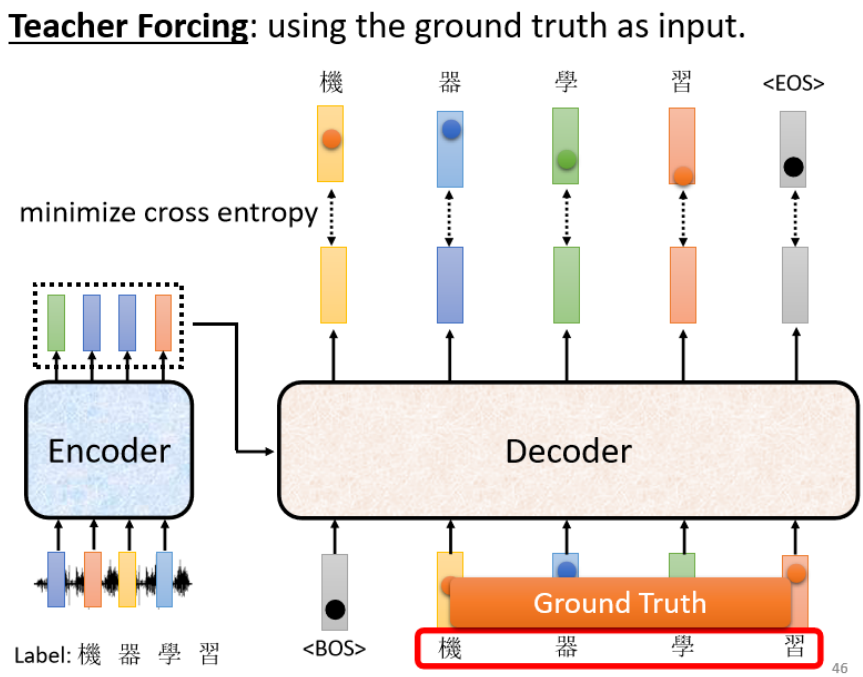
它就**跟分类很像**,刚才助教在讲解作业的时候也有提到这件事情,你可以想成每一次我们在產生,每一次 Decoder 在產生一个中文字的时候,其实就是做了一次分类的问题,中文字假设有四千个,那就是**做有四千个类别的分类的问题**

所以实际上训练的时候这个样子,我们已经知道输出应该是“机器学习”这四个字,就告诉你的 Decoder ,现在你第一次的输出 第二次的输出,第三次的输出 第四次输出,应该分别就是“机” “器” “学”跟“习”,这四个中文字的 One-Hot Vector,我们**希望我们的输出,跟这四个字的 One-Hot Vector 越接近越好**



在训练的时候,每一个输出都会有一个 Cross Entropy,每一个输出跟 One-Hot Vector,跟它对应的正确答案都有一个 Cross Entropy,我们要希望所有的 Cross Entropy 的总和最小,越小越好

所以这边做了四次分类的问题,我们希望这些分类的问题,它总合起来的 Cross Entropy 越小越好,**还有 END 這个符号**



那这个就是 Decoder 的训练：**把 Ground Truth ,正确答案给它,希望 Decoder 的输出跟正确答案越接近越好**

那这边有一件值得我们注意的事情,在**训练**的时候我们会给 Decoder 看**正确答案**,也就是我们会告诉它说

- 在已经有 "BEGIN",在有"机"的情况下你就要输出"器"
- 有 "BEGIN" 有"机" 有"器"的情况下输出"学"
- 有 "BEGIN" 有"机" 有"器" 有"学"的情况下输出"习"
- 有 "BEGIN" 有"机" 有"器" 有"学" 有"习"的情况下,你就要输出"断"

在 Decoder 训练的时候,我们会在输入的时候给它**正确的答案**,那这件事情叫做 ==Teacher Forcing==

那这个时候你马上就会有一个问题了

- 训练的时候,Decoder 有偷看到正确答案了
- 但是测试的时候,显然没有正确答案可以给 Decoder 看

刚才也有强调说在真正使用这个模型,在 Inference 的时候,Decoder 看到的是自己的输入,这**中间显然有一个 ==Mismatch==**,那等一下我们会有一页投影片的说明,有什麼样可能的解决方式

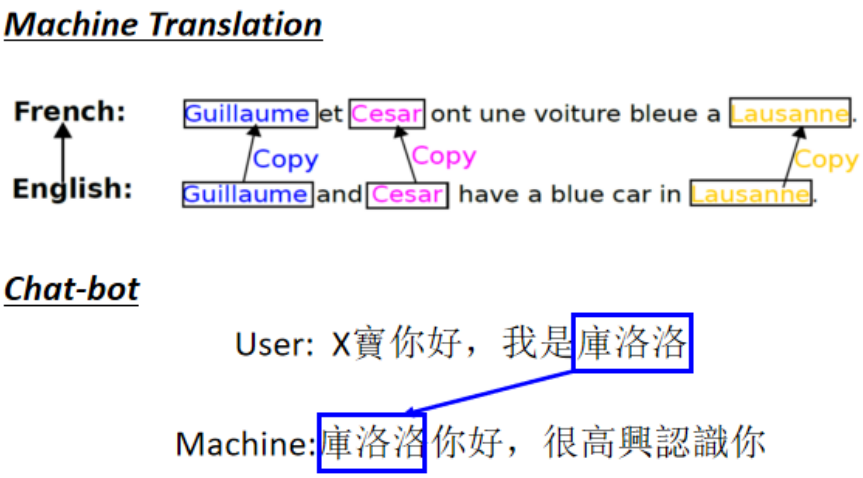
Tips

那接下来,不侷限於 Transformer ,讲一些训练这种 Sequence To Sequence Model 的Tips

Copy Mechanism

在我们刚才的讨论裡面,我们都要求 Decoder 自己產生输出,但是对很多任务而言,也许 **Decoder 没有必要自己创造输出**出来,它需要做的事情,也许是**从输入的东西裡面复製**一些东西出来

像这种复製的行為在哪些任务会用得上呢,一个例子是做聊天机器人



- 人对机器说:你好 我是库洛洛,
- 机器应该回答说:库洛洛你好 很高兴认识你

对机器来说,它其实**没有必要创造**库洛洛这个词汇,这对机器来说一定会是一个非常怪异的词汇,所以它可能很难,在训练资料裡面可能一次也没有出现过,所以它不太可能正确地產生这段词汇出来

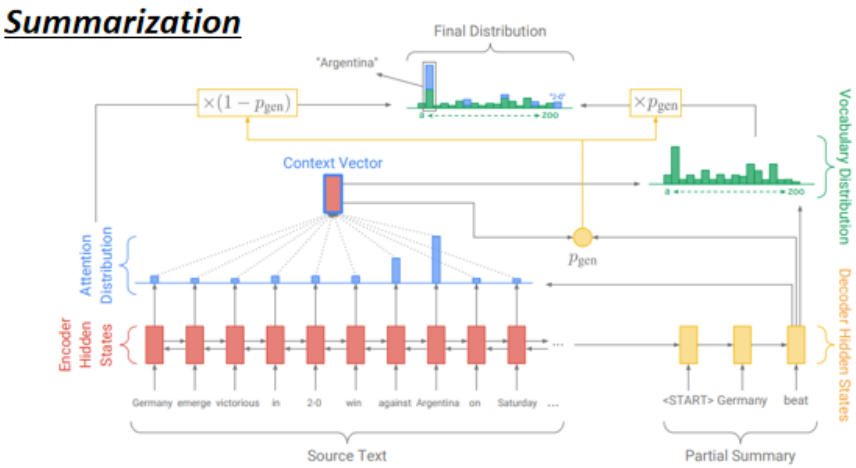
但是假设今天机器它在学的时候,它学到的是看到输入的时候说我是某某某,就直接把某某某,不管这边是什麼复製出来说某某某你好

那这样子机器的**训练显然会比较容易**,它显然比较有可能得到正确的结果,所以复製对於对话来说,可能是一个需要的技术 需要的能力

Summarization

或者是在做摘要的时候,你可能更需要 Copy 这样子的技能

https://arxiv.org/abs/1704.04368



摘要就是,你要训练一个模型,然后这个**模型去读一篇文章,然后產生这篇文章的摘要**

那这个任务完全是有办法做的,你就是收集大量的文章,那每一篇文章都有人写的摘要,然后你就训练一个,Sequence-To-Sequence 的 Model,就结束了

你要做这样的任务,**只有一点点的资料是做不起来的**,有的同学收集个几万篇文章,然后训练一个这样的,Sequence-To-Sequence Model,发现结果有点差

你要训练这种,你要叫机器说合理的句子,通常这个==百万篇文章==是需要的,所以如果你有百万篇文章,那些文章都有人标的摘要,那有时候你会把,直接把文章标题当作摘要,那这样就不需要花太多人力来标,你是可以训练一个,直接可以帮你读一篇文章,做个摘要的模型

对**摘要**这个任务而言,其实**从文章裡面直接复製一些资讯出来**,可能是一个很关键的能力,那 Sequence-To-Sequence Model,有没有办法做到这件事呢,那简单来说就是有,那我们就不会细讲

最早有从输入复製东西的能力的模型,叫做 Pointer Network

Pointer Network



https://youtu.be/VdOyqNQ9aww

Incorporating Copying Mechanism in Sequence-to-Sequence Learning

https://arxiv.org/abs/1603.06393

那这个过去上课是有讲过的,我把录影放在这边给大家参考,好 那后来还有一个变形,叫做 Copy Network,那你可以看一下这一篇,Copy Mechanism,就是 Sequence-To-Sequence,有没有问题,你看 Sequence-To-Sequence Model,是怎麼做到从输入复製东西到输出来的

Guided Attention

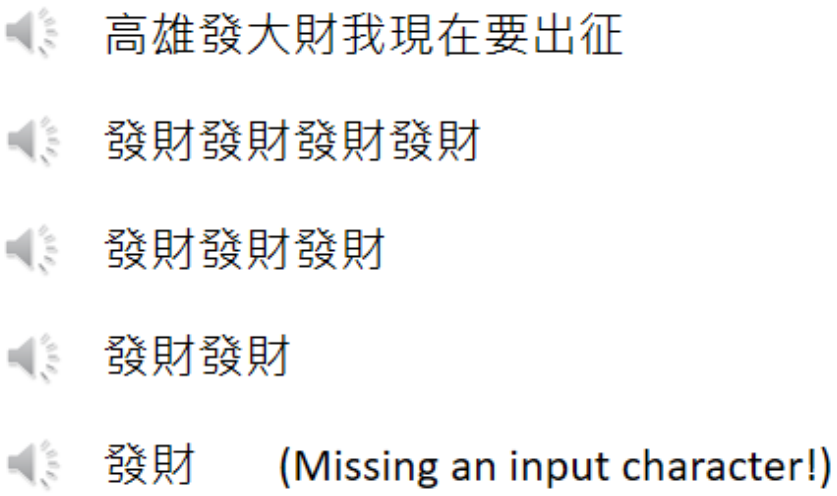
机器就是一个黑盒子,有时候它裡面学到什麼东西,你实在是搞不清楚,那有时候它会犯**非常低级的错误**

这边举的例子是**语音合成**

你完全可以就是训练一个,Sequence-To-Sequence 的 Model,Transformer 就是一个例子

- 收集很多的声音,文字跟声音讯号的对应关係
- 然后接下来告诉你的,Sequence-To-Sequence Model ,看到这段中文的句子,你就输出这段声音
- 然后就没有然后,就==硬 Train 一发==就结束了,然后机器就可以学会做语音合成了

像这样的方法做出来结果,其实还不错,



举例来说我叫机器连说 4 次发财,看看它会怎麼讲,机器输出的结果是:发财 发财 发财 发财

就发现很神奇,我输入的发财是明明是同样的词汇,只是重复 4 次,机器居然自己有一些抑扬顿挫,它怎麼学到这件事,不知道,它自己训练出来就是这个样子

那你让它讲 3 次发财也没问题,那它讲 2 次发财也没问题,让它讲 1 次发财,**它不念“发”**

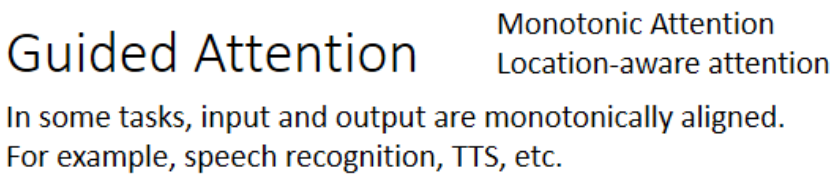
不知道為什麼这样子,就是你这个 Sequence-To-Sequence Model,有时候 Train 出来就是,会產生莫名其妙的结果,**也许在训练资料裡面,这种非常短的句子很少**,所以机器不知道要怎麼处理这种非常短的句子,你叫它念发财,它把发省略掉只念财,你居然叫它念 4 次的发财,重复 4 次没问题,叫它只念一次,居然会有问题,就是这麼的奇怪

当然其实这个例子并没有那麼常出现,就这个用 Sequence-To-Sequence,Learn 出来 TTS,也没有你想像的那麼差,这个要找这种差的例子也是挺花时间的,要花很多时间才找得到这种差的例子,但这样子的例子是存在的

所以怎麼办呢

我们刚才发现说机器居然**漏字**了,输入有一些东西它居然没有看到,我们能不能**够强迫它,一定要把输入的每一个东西通通看过**呢

这个是有可能的,这招就叫做 ==Guided Attention==

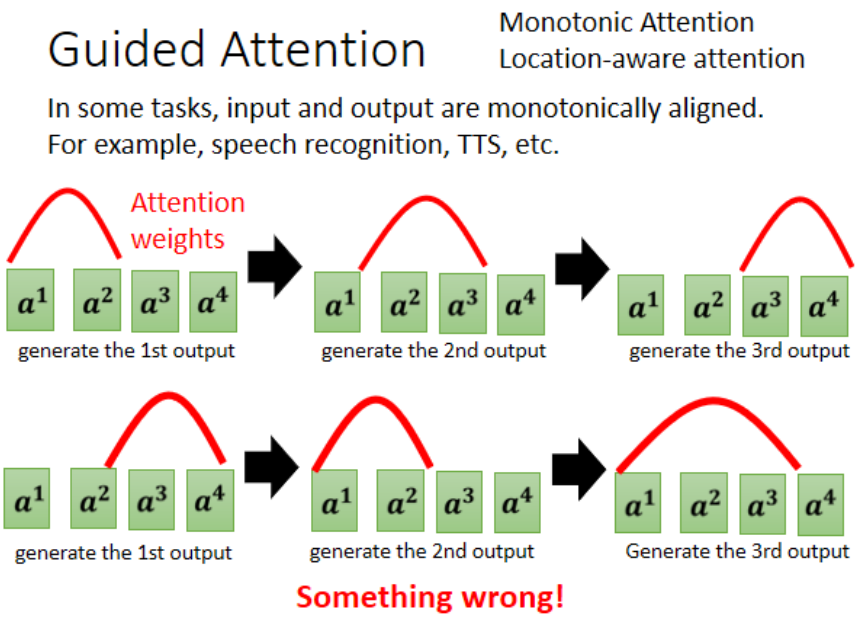


像语音辨识这种任务,你其实很难接受说,你讲一句话,今天辨识出来,居然有一段机器没听到,或语音合成你输入一段文字,语音合出来居然有一段没有念到,这个人很难接受

那如果是其它应用,比如说 Chat Bot,或者是 Summary,可能就没有那麼严格,因為对一个 Chat Bot 来说,输入后一句话,它就回一句话,它到底有没有把整句话看完,其实你 Somehow 也不在乎,你其实也搞不清楚

但是**对语音辨识 语音合成,Guiding Attention,可能就是一个比较重要的技术**

Guiding Attention 要做的事情就是,**要求机器它在做 Attention 的时候,是有固定的方式的**,举例来说,对语音合成或者是语音辨识来说,我们想像中的 **Attention,应该就是由左向右**



在这个例子裡面,我们用红色的这个曲线,来代表 Attention 的分数,这个越高就代表 Attention 的值越大

我们以语音合成為例,那你的输入就是一串文字,那你在合成声音的时候,显然是**由左念到右**,所以机器应该是,先看最左边输入的词汇產生声音,再看中间的词汇產生声音,再看右边的词汇產生声音

如果你今天在做语音合成的时候,你发现机器的 Attention,是**颠三倒四的**,它先看最后面,接下来再看前面,那再胡乱看整个句子,那显然有些是做错了,显然有些是,Something is wrong,有些是做错了,

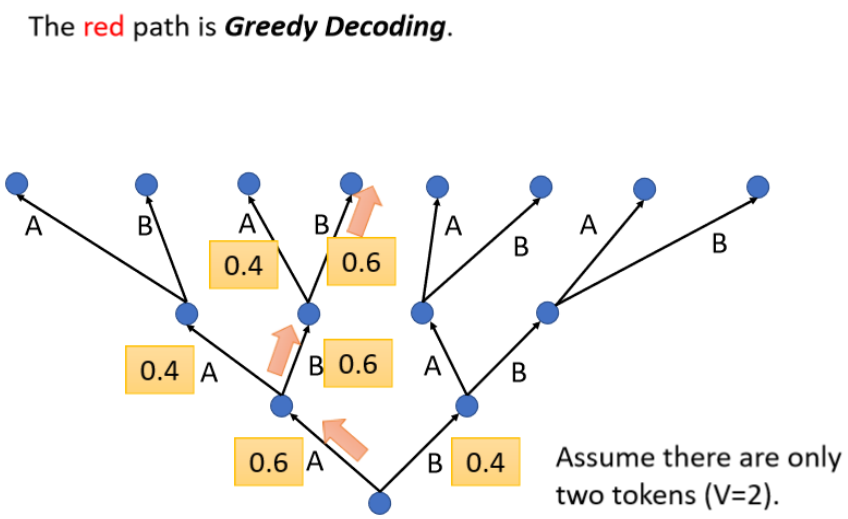
所以 Guiding Attention 要做的事情就是,强迫 Attention 有一个固定的样貌,那如果你对这个问题,本身就已经有理解知道说,语音合成 TTS 这样的问题,你的 Attention 的分数,Attention 的位置都应该由左向右,那不如就直接把这个限制,放进你的 Training 裡面,要求机器学到 Attention,就应该要由左向右

那这件事怎麼做呢,有一些关键词汇我就放在这边,让大家自己 Google 了,比如说某某 Mnotonic Attention,或 Location-Aware 的 Attention,那这个部分也是大坑,也不细讲,那就留给大家自己研究

Beam Search

Beam Search ,我们这边举一个例子,在这个例子裡面我们假设说,我们现在的这个 **Decoder就只能產生两个字**,一个叫做 A 一个叫做 B

那对 Decoder 而言,它做的事情就是,**每一次在第一个 Time Step,它在 A B 裡面决定一个,然后决定了 A 以后,再把 A 当做输入,然后再决定 A B 要选哪一个**

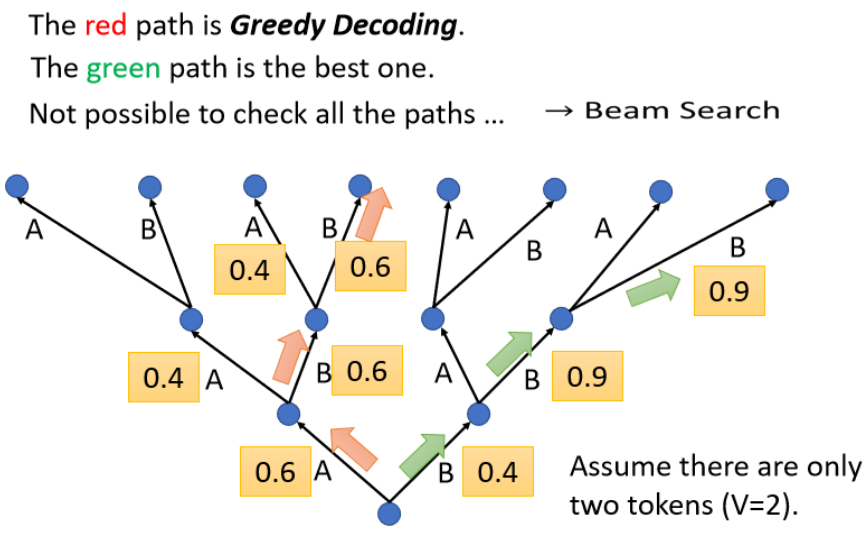


那举例来说,它可能选 B 当作输入,再决定 A B 要选哪一个,那在我们刚才讲的 Process 裡面,每一次 Decoder 都是选,分数最高的那一个

我们**每次都是选Max 的那一个**,所以假设 A 的分数 0.6,B 的分数 0.4,Decoder 的第一次就会输出 A,然后接下来假设 B 的分数 0.6,A 的分数 0.4,Decoder 就会输出 B,好,然后再假设把 B 当做 Input,就现在输入已经有 A 有 B 了,然后接下来,A 的分数 0.4,B 的分数 0.6,那 Decoder 就会选择输出 B,所以输出就是 A 跟 B 跟 B

那像这样子每次找分数最高的那个 Token,每次找分数最高的那个字,来当做输出这件事情叫做,==Greedy Decoding==

但是 Greedy Decoding,一定是更好的方法吗,有没有可能我们在第一步的时候,先稍微捨弃一点东西



比如说第一步虽然 B 是 0.4,但我们就先选 0.4 这个 B,然后接下来我们选了 B 以后,也许接下来的 B 的可能性就大增,就变成 0.9,然后接下来第三个步骤,B 的可能性也是 0.9

如果你比较红色的这一条路,跟绿色这条路的话,你会发现说绿色这一条路,虽然一开始第一个步骤,你选了一个比较差的输出,但是**接下来的结果是好的**

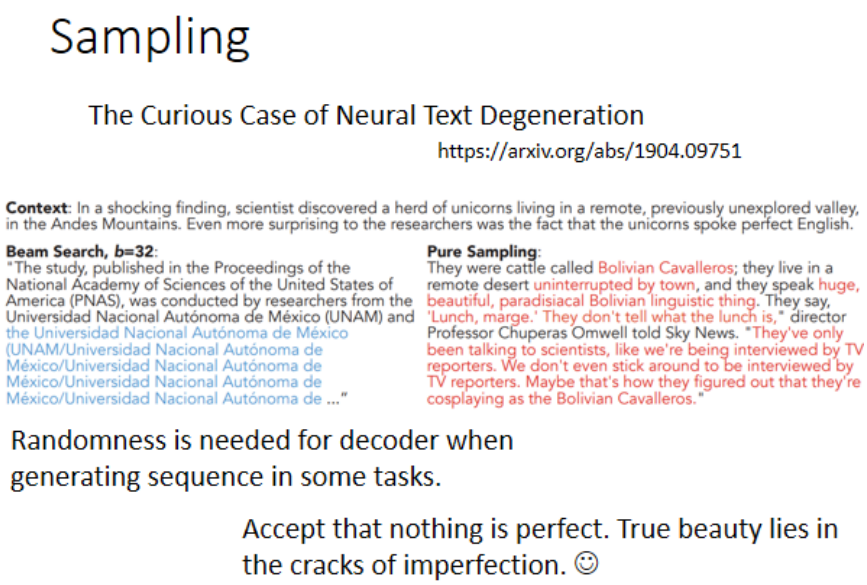
这个就跟那个天龙八部的真龙棋局一样,对不对,先堵死自己一块,结果接下来反而赢了

那所以我,如果我们要怎麼找到,这个最好的绿色这一条路呢,也许一个可能是,**爆搜所有可能的路径**,但问题是我们实际上,**并没有办法爆搜所有可能的路径**,因為实际上每一个转捩点可以的选择太多了,如果是在对中文而言,我们中文有 4000 个字,所以这个树每一个地方分叉,都是 4000 个可能的路径,你走两三步以后,你就无法穷举

所以怎麼办呢,有一个演算法叫做 ==Beam Search==,它用比较有效的方法,找一个 Approximate,找一个估测的 Solution,找一个不是很精準的,不是完全精準的 Solution,这个技术叫做 Beam Search,那这个也留给大家自己 Google,好

那这个 Beam Search 这个技术,到底有没有用呢,有趣的事就是,**它有时候有用,有时候没有用,**你会看到有些文献告诉你说,Beam Search 是一个很烂的东西

举例来说这篇 Paper 叫做,The Curious Case Of Neural Text Degeneration,那这个任务要做的事情是,Sentence Completion,也就是机器先读一段句子,接下来它要把这个句子的后半段,把它完成,你给它一则新闻,或者是一个故事的前半部,哇 它自己发挥它的想像创造力,把这个文章,把故事的后半部把它写完



那你会发现说,Beam Search 在这篇文章裡面,一开头就告诉你说,Beam Search 自己有问题：如果你用 Beam Search 的话,会发现说机器不断讲重复的话,它不断开始陷入鬼打墙 无穷迴圈,不断说重复的话

如果你今天不是用 Beam Search,有加一些随机性,虽然结果不一定完全好,但是看起来至少是比较正常的句子,所以有趣的事情是,有时候对 Decorder 来说,没有找出分数最高的路,反而结果是比较好的

这个时候你又觉得乱乱的 对不对,就是刚才前一页投影片才说,要找出分数最高的路,现在又要讲说**找出分数最高的路不见得比较好**,到底是怎麼回事呢

那其实这个就是要,**看你的任务的本身的特性**

- 就假设一个任务,它的**答案非常地明确**

  举例来说,什麼叫答案非常明确呢,比如说语音辨识,说一句话辨识的结果就只有一个可能,就那一串文字就是你唯一可能的正确答案,并没有什麼模糊的地带

  对这种任务而言,通常 Beam Search 就会比较有帮助,那什麼样的任务

- **你需要机器发挥一点创造力的时候,这时候 Beam Search 就比较没有帮助**,

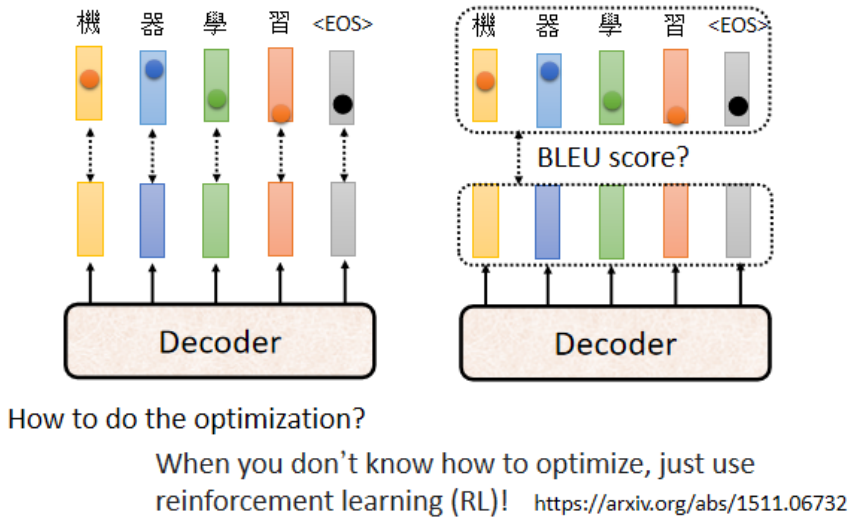
> 举例来说在这边的 Sentence Completion,给你一个句子,给你故事的前半部,后半部有无穷多可能的发展方式,那这种需要有一些创造力的,有不是只有一个答案的任务,往往会比较需要在 Decoder 裡面,加入随机性,还有另外一个 Decoder,也非常需要随机性的任务,叫做语音合成,TTS 就是语音合成的缩写

这也许就呼应了一个英文的谚语,就是要**接受没有事情是完美的,那真正的美也许就在不完美之中**,对於 TTS 或 Sentence Completion 来说,Decoder 找出最好的结果,不见得是人类觉得最好的结果,反而是奇怪的结果,那你加入一些随机性,结果反而会是比较好的

Optimizing Evaluation Metrics?

在作业裡面,我们评估的标準用的是,BLEU Score,BLEU Score 是你的 Decoder,先產生一个完整的句子以后,再去跟正确的答案一整句做比较,我们是拿两个句子之间做比较,才算出 BLEU Score

但我们在训练的时候显然不是这样,**训练**的时候,**每一个词汇是分开考虑的**,训练的时候,我们 Minimize 的是 Cross Entropy,Minimize Cross Entropy,真的可以 Maximize BLEU Score 吗



不一定,因為这两个根本就是,它们可能有一点点的关联,但它们又没有那麼直接相关,它们根本就是两个不同的数值,所以我们 Minimize Cross Entropy,不见得可以让 BLEU Score 比较大

所以你发现说在助教的程式裡面,助教在做 Validation 的时候,并不是拿 Cross Entropy 来挑最好的 Model,而是挑 BLEU Score 最高的那一个 Model,所以我们训练的时候,是看 Cross Entropy,但是我们实际上你作业真正评估的时候,看的是 BLEU Score,所以你 Validation Set,其实应该考虑用 BLEU Score

那接下来有人就会想说,那我们能不能**在 Training 的时候,就考虑 BLEU Score 呢**,我们能不能够训练的时候就说,我的 Loss 就是,BLEU Score 乘一个负号,那我们要 Minimize 那个 Loss,假设你的 Loss 是,BLEU Score乘一个负号,它也等於就是 Maximize BLEU Score

但是**这件事实际上没有那麼容易**,你当然可以把 BLEU Score,当做你训练的时候,你要最大化的一个目标,但是 BLEU Score 本身很复杂,它是不能微分的,

这边之所以採用 Cross Entropy,而且是每一个中文的字分开来算,就是因為这样我们才有办法处理,如果你是要计算,两个句子之间的 BLEU Score,这一个 Loss,根本就没有办法做微分,那怎麼办呢

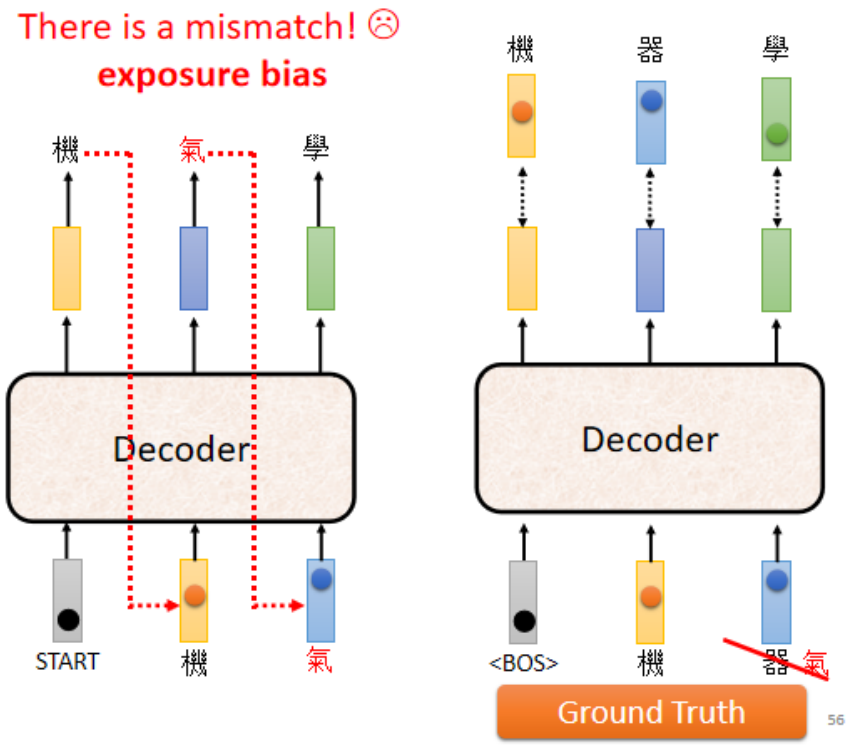
这边就教大家一个口诀,遇到你在 Optimization 无法解决的问题,==用 RL 硬 Train 一发==就对了这样,遇到你无法 Optimize 的 Loss Function,把它当做是 RL 的 Reward,把你的 Decoder 当做是 Agent,它当作是 RL,Reinforcement Learning 的问题硬做

其实也是有可能可以做的,**有人真的这样试过**,我把 Reference 列在这边给大家参考,当然这是一个比较难的做法,那并没有特别推荐你在作业裡面用这一招

Scheduled Sampling

那我们要讲到,我们刚才反覆提到的问题了,就是**训练跟测试居然是不一致**的

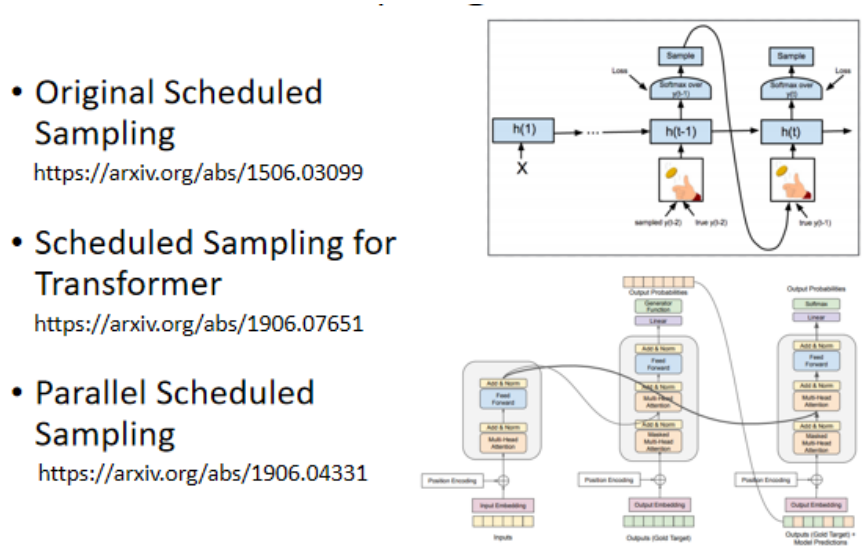
测试的时候,Decoder 看到的是自己的输出,所以测试的时候,Decoder 会看到一些错误的东西,但是在训练的时候,Decoder 看到的是完全正确的,那这个不一致的现象叫做,==Exposure Bias==



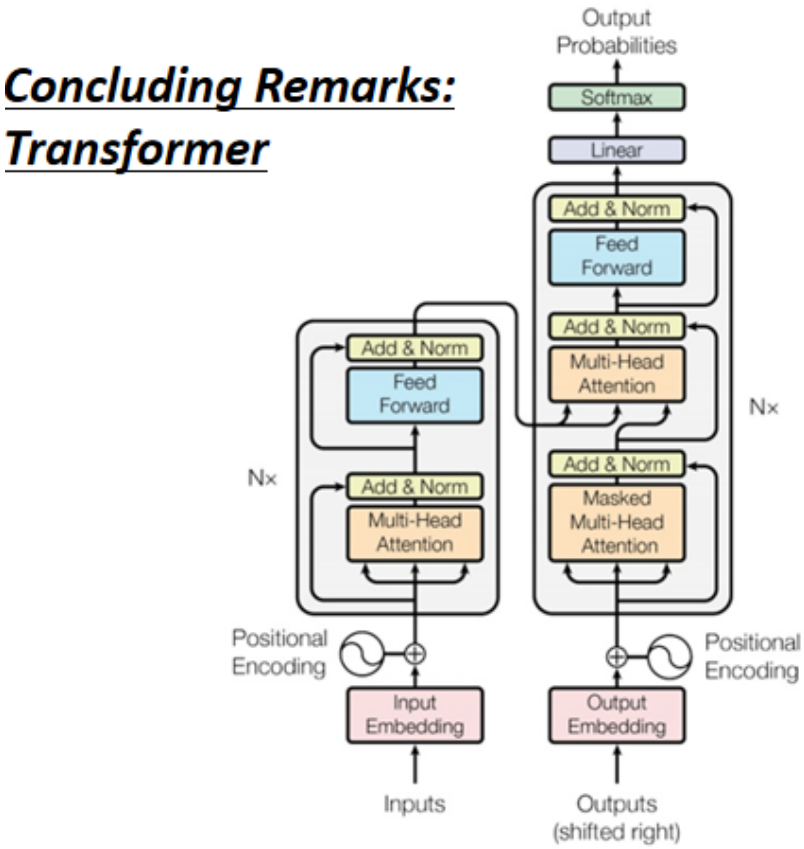
假设 Decoder 在训练的时候,永远只看过正确的东西,那在测试的时候,你只要有一个错,那就会**一步错 步步错**,因為对 Decoder 来说,它从来没有看过错的东西,它看到错的东西会非常的惊奇,然后接下来它產生的结果可能都会错掉

所以要怎麼解决这个问题呢

有一个可以的思考的方向是,**给 Decoder 的输入加一些错误的东西**,就这麼直觉,你不要给 Decoder 都是正确的答案,偶尔给它一些错的东西,它反而会学得更好,这一招叫做,==Scheduled Sampling==,它不是那个 Schedule Learning Rate,刚才助教有讲 Schedule Learning Rate,那是另外一件事,不相干的事情,这个是 Scheduled Sampling



Scheduled Sampling 其实很早就有了,这个是 15 年的 Paper,很早就有 Scheduled Sampling,在还没有 Transformer,只有 LSTM 的时候,就已经有 Scheduled Sampling,但是 Scheduled Sampling 这一招,它其实会伤害到,Transformer 的平行化的能力,那细节可以再自己去了解一下,所以对 Transformer 来说,它的 Scheduled Sampling,另有招数跟传统的招数,跟原来最早提在,这个 LSTM上被提出来的招数,也不太一样,那我把一些 Reference 的,列在这边给大家参考



好 那以上我们就讲完了,Transformer 和种种的训练技巧,这个我们已经讲完了 Encoder,讲完了 Decoder,也讲完了它们中间的关係,也讲了怎麼训练,也讲了种种的 Tip

GAN_P1

Generation

Network as Generator

接下来要进入一个,新的主题 我们要讲==生成==这件事情

到目前为止大家学到的network,都是**一个function,你给他一个X就可以输出一个Y**



我们已经学到各式各样的,network架构,可以处理不同的X 不同的Y

我们学到**输入**的X

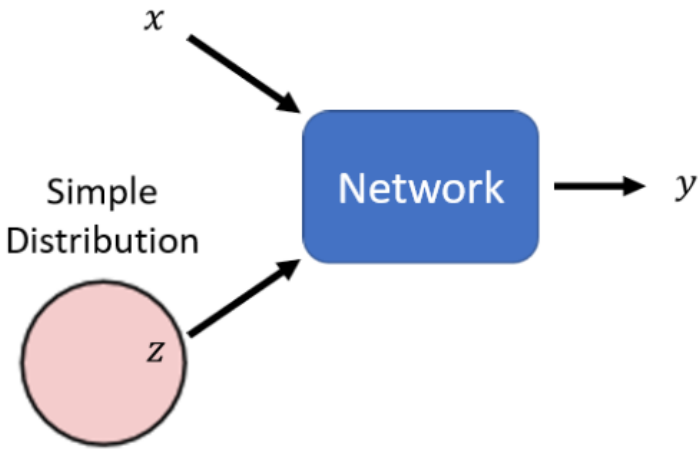
- 如果是一张图片的时候怎麼办
- 如果是一个sequence的时候怎麼办

我们也学到**输出**的Y

- 可以是一个数值
- 可以是一个类别
- 也可以是一个sequence

接下来我们要进入一个新的主题,这个新的主题是要把network,当做一个==generator==来用,我们要把network拿来做生成使用

那把network拿来,当作generator使用,他特别的地方是现在network的输入,会加上一个**random的variable,会加上一个Z**



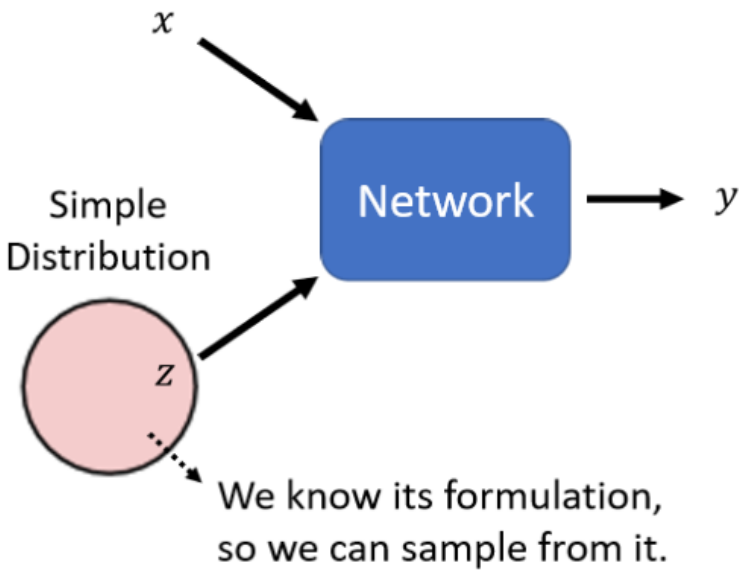
这个Z,是从某一个,distribution sample出来的,所以现在network它**不是只看一个固定的X**得到输出,它是**同时看X跟Z**得到输出

network怎麼同时看X跟Z，有很多不同的做法,就看你怎样设计你的network架构

- 你可以说X是个向量,Z是个向量 两个向量直接接起来,变成一个比较长的向量,就当作network的input
- 或者是你的X跟Z正好长度一模一样,把它们相加以后,当做network的input
- 等等

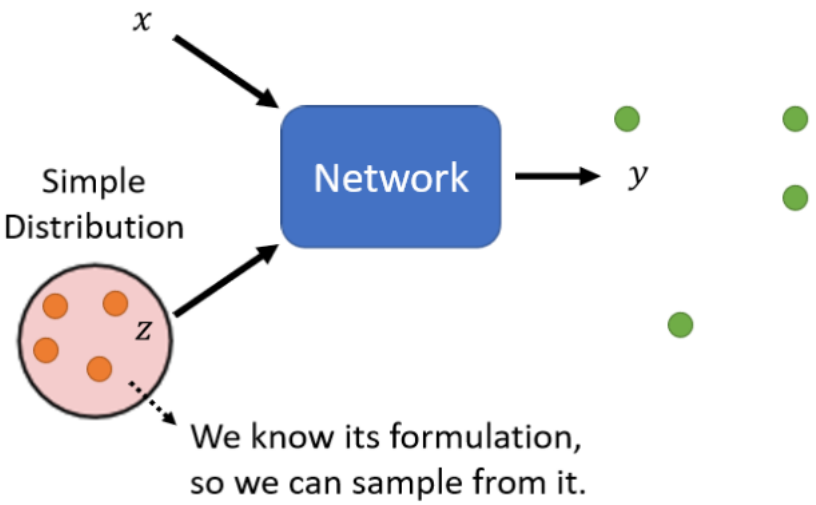
Z特别的地方是 **它是不固定的**,每一次我们用这个network的时候,都会随机生成一个Z,所以Z每次都不一样,它是从一个distribution裡面,sample出来的

这个distribution,这边有一个限制是,它必须够简单，够简单的意思是,我们知道它的式子长什麼样子,我们可以从这个distribution,去做sample

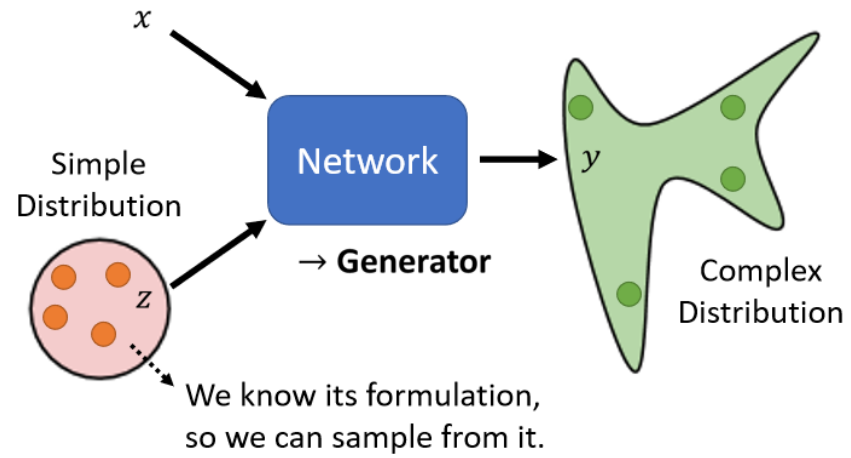


举例来说这个distribution,可以是一个function distribution,你知道function distribution的式子,你知道怎麼从,gaussian distribution做sample

它也可以是uniform distribution,那uniform distribution,的式子你一定知道,你也知道怎麼从,uniform distribution做sample,所以这一个distribution,的形状你自己决定,但你只要记得说它是简单的,你能够sample它的 就结束了



所以每次今天,有一个X进来的时候,你都从这个distribution,裡面做一个sample,然后得到一个output,随著你sample到的Z不同,Y的输出也就不一样,所以这个时候我们的**network输出,不再是单一一个固定的东西,而变成了一个复杂的distribution**,同样的X作為输入,我们这边每次sample到,不一样的东西,通过一个复杂的network转换以后,它就会变成一个复杂的分布,你的network的输出,就变成了一个distribution



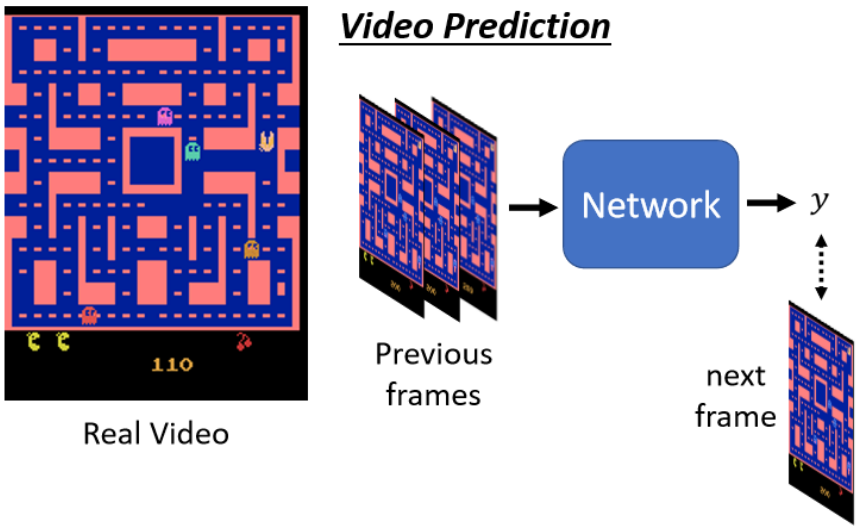
那这种**可以输出,一个distribution的network,我们就叫它==generator==**

Why distribution

在讲怎麼训练出generator之前,我们第一个想要回答的问题是,為什麼我们需要**generator输出是一个分布**?输入X输出Y,这样固定的输入跟输出关係不好吗?

所以以下就举一个例子来跟你说明,**為什麼输出有时候需要是一个分布**

这边举的例子,是video prediction,就是给机器一段的影片,然后它要预测接下来会发生什麼事情



那这个例子,是我从上面这个,github的连结
https://github.com/dyelax/Adversarial_Video_Generation找到的,那在这个连结裡面
它要做的事情,是去预测小精灵这个游戏,接下来的游戏画面会长什麼样子

video prediction,那你就**给你的network过去的游戏画面,然后它的输出就是新的游戏画面,下一秒的下一个时间点的游戏画面**

有人可能会问说怎麼输出一张图片?

这个一张图片就是一个很长的向量,所以你只要让你的network,可以输出一个很长的向量,然后这个向量整理成图片以后,跟你的目标越接近越好

其实在这个github裡面,它不是直接把整个画面当做输入,它是从画面裡面只切一小块当做输入,就它把这整个画面切成很多块,然后分开来处理,不过我们这边為了简化说明,就当作network是一次,输入一整个这样的画面

如果你用我们学过的network training的方法,Supervise learning train下去,你得到的结果可能会是这个样子,这个是机器预测出来的结果(第三部分 Non-Adversarial)

所以你看有点模模糊糊的,而且中间还会**变换角色**,神奇的是那个小精灵,走著走著 居然就**分裂**了,它走到转角的时候,看它走到转角的时候就分裂成两隻了,走到这边又再分裂成两隻了,有时候走著走著还会**消失**

因為今天对这个network而言,在**训练资料裡面同样的输入**,有时候同样的转角

- 有时候小精灵是往左转
- 有时候小精灵是往右转,

这样这**两种可能性,同时存在你的训练资料裡面**

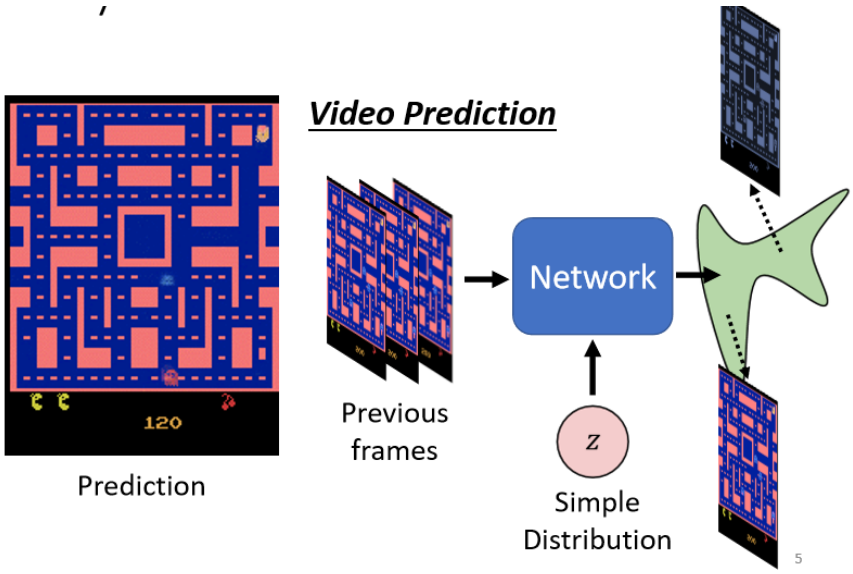
你在训练的时候,今天你的network,得到的训练的指示是,给定这一段输入,那今天得到这一笔训练资料,那它就要学到给定这段输入,输出应该要往右转,给定这一些训练资料,有时候你会看到的是向左转,那机器就会学到给定这一段输入,它要向左转

所以你的network,学到的就是**两面讨好**,

因為它需要得到一个输出,这个输出同时距离向左转最近,同时也距离向右转最近,那怎麼样同时距离向左转最近,向右转最近,也许就是同时向左也向右转

所以你的network,它就会得到一个错误的结果,向左转是对的 向右转也是对的,但是同时向左向右转 反而是不对的,

那有什麼样的可能性,可以处理这个问题,一个可能性就是,让**机器的输出是有机率的**,让它不再是输出单一的输出,让它输出一个机率的分佈



当我们给这个network,一个distribution的时候,当我们给这个network input,加上一个Z的时候,它的输出就变成了一个distribution,它的输出就不再是固定的

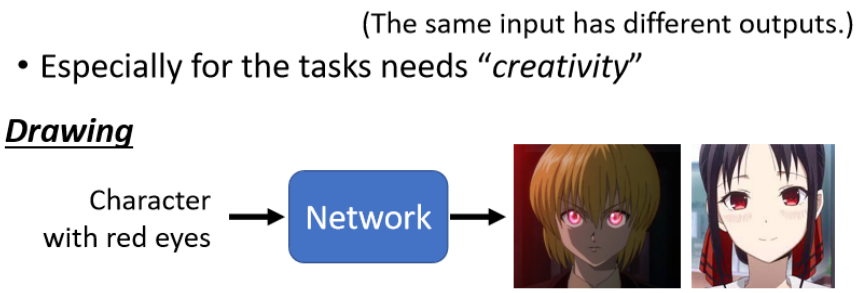
我们希望训练一个network,它可以知道说它的输出的这个分佈,包含了向左转的可能,也包含向右转的可能

举例来说假设你选择你的**Z,是一个比如说,binary的random variable,它就只有0跟1那可能各佔50%,也许你的network就可以学到说,Z sample到1的时候就向左转,Z sample到0的时候就向右转**,就可以解决,这个世界有很多不可预测的东西,的状况

那什麼时候我们会特别需要,处理这种问题,什麼时候,我们会特别需要这种,generator的model,当我们的任务需要一点**创造力**的时候

任务需要一点创造力这件事情,是比较拟人化的讲法,更具体一点的讲法就是,我们想要找一个function,但是**同样的输入**有**多种可能的输出**,而这些**不同的输出**都是**对的**,

举例来说,画图这件事情,可能就需要一些创造力

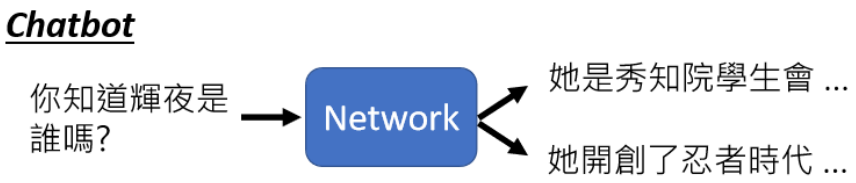


举例来说假设叫一个人,画一个红眼睛的角色,那每个人可能画出来,或者心裡想的动画人物可能都不一样,有哪些角色是红眼睛的

举例来说库拉皮卡是红眼睛的,它是窟卢塔族的,窟卢塔族的愤怒以后,就会有火红眼,那辉夜也是红眼睛的,那因為这个库拉皮卡从黑暗大陆,回来以后 就用他在黑暗大陆,得到的资源成立了四宫集团,辉夜其实是库拉皮卡的子孙,那她那一代火红眼变成是显性的,不用生气火红眼也会显现出来,所以库拉皮卡跟辉夜,他们都有火红眼,所以同样要画一个红眼睛的角色,每个人心裡想像的红眼睛的角色,都是不一样的

那这个时候,我们就需要让机器能够,让我们的model,能够output一个distribution

那还有什麼样的例子,会需要用到创造力,举例来说 对话这件事情



举例来说假设你跟另外一个人说,**你知道辉夜是谁吗**,其实有很多不同的答案对不对

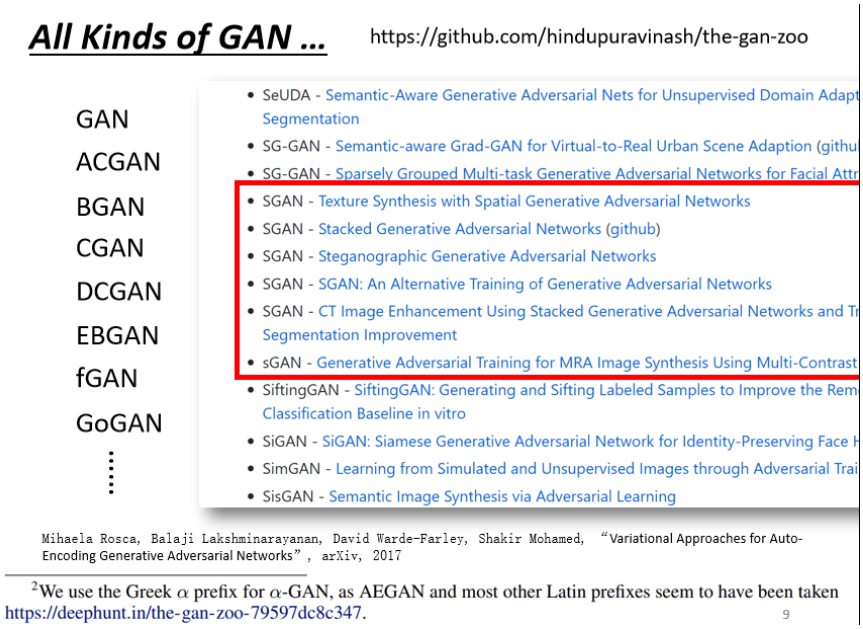
辉夜她是秀知院的学生会部会长,最近跟会长做了一些不可描述的事情,我没有说是什麼事情,所以也不算是爆雷,但是我们知道说辉夜,其实还有另外一个成就,其实虽然说辉夜姬,这个动画还没有完结,但是他的后传其实已经演完了,后来白银英年早逝所以辉夜,就把自己的头髮染白,她為了想要离开伤心的地球,就坐著太空船到另外一个星球,另外一个星球也有一些原始的生命,那些生命跟人类长得也是挺像的,他们还过著农耕的生活,然后辉夜看到其中一个小国的国王,长得跟白银有点像,所以她就跟那个小国国王在一起,她们生下了一个小孩就是六道仙人,於是就开创了忍者时代,真是可喜可贺 可喜可贺。所以我们今天学到什麼事,我们今天就是学到说,这个辉夜大小姐这个动画,它真是一个了不起的动画,它其实是在两部大长篇中间的小品,它既是猎人的后传,也是火影忍者的前传,这三个故事是可以串在一起的,就是这麼回事

所以我们对机器说一句话,问它说辉夜是谁,其实每个人也可能都有不同的答案,这个时候我们就需要,generative的model

Generative Adversarial Network (GAN)

generative的model,其中一个非常知名的,就是==generative adversarial network==,它的缩写 是==GAN==,那我们这一堂课主要就是介绍,generative adversarial network,发音就是 gàn

它其实有很多各式各样的变形,你可以在网路上找到,一个GAN的动物园,找到一个GAN的zoo



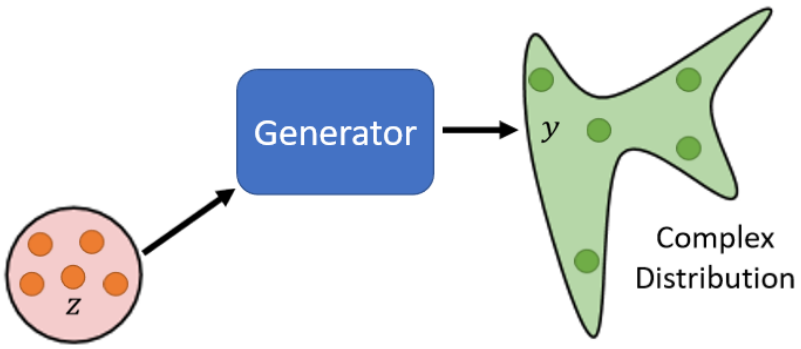
那个GAN的动物园裡面,收集了超过五百种以上的GAN,每次有人发明了,一个新的GAN的时候,他就会在前面加一个英文的字母,但是英文的字母是有限的,很快的英文的字母就被用尽了

举例来说在GAN的动物园裡面,至少就有六种的SGAN,它们都是不同的东西,但它们通通被叫做SGAN,甚至还发生了的状况,有一篇paper他提出来的叫做,"Variational auto-encoding GAN",照理说应该所写成,AEGAN或者是AGAN,但是作者加了一个註解说,哎呀AEGAN被别人用了,所有的英文字母,看起来都被别人用了,我只好叫做α-GAN

Anime Face Generation

我们现在要举的例子,就是要让机器生成动画人物的,二次元人物的脸,我们举的例子是Unconditional的generation,unconditional generation,就是我们这边先把X拿掉

- Unconditional generation



那之后我们在讲到conditional,generation的时候,我们会再把X加回来,这边先把X拿掉,所以我们的generator它输入就是Z,它的输出就是Y

那输入的这个Z是什麼

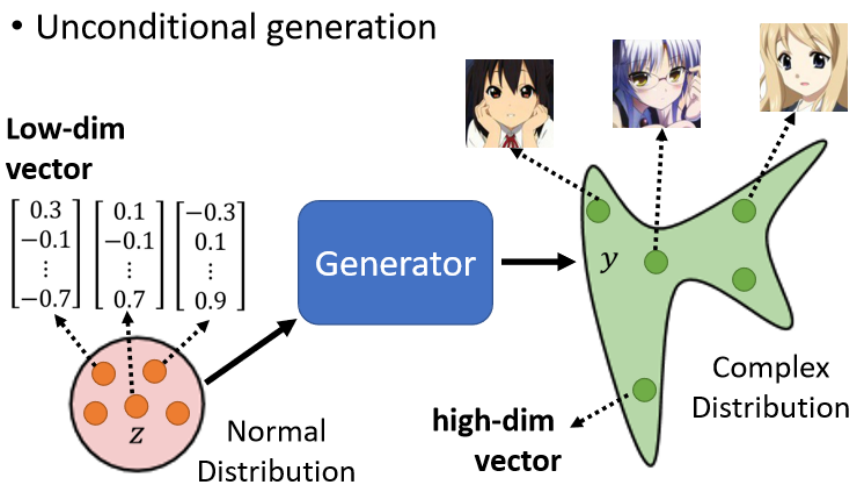
我们都假设**Z是从一个normal distribution里sample出来的向量**,那这个向量通常会是一个,**low-dimensional**的向量,它的维度其实是你自订的,你自己决定的,那通常你就订个50100,的大小,它是你自己决定的

好你从这边Z,你从这个normal distribution,裡面sample一个向量以后,丟到generator裡面,Generator就给你一个对应的输出,那我们希望对应的输出,就是一个二次元人物的脸

那到底generator要输出,怎麼样的东西,才会变成一个二次元人物的人脸,其实这个问题没有你想像的那麼困难

一张图片就是一个**非常高维的向量**,所以generator实际上做的事情,就是產生一个非常高维的向量,举例来说 假设这是一个64X64,然后彩色的图片,那你的generator输出就是64X64X3,那麼长的向量 把那个向量整理一下,就变成一张二次元人物,这个就是generator要做的事情

当你输入的向量不同的时候,你的输出就会跟著改变,所以你从这个,normal distribution裡面,Sample Z出来 Sample到不同的Z,那你输出来的Y都不一样,那我们希望说不管你这边sample到什麼Z,输出来的都是动画人物的脸



那讲到这边可能有同学会问说,这边為什麼是,normal distribution,不能是别的吗?

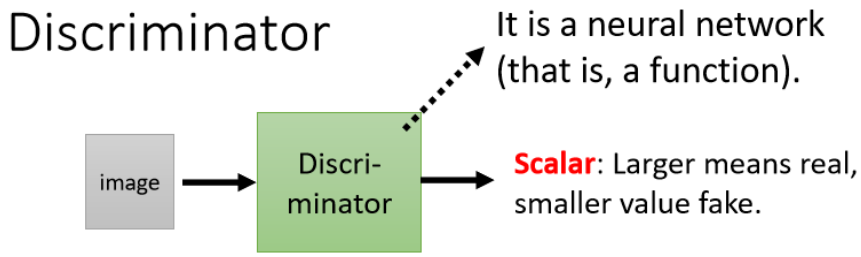
可以是别的,这边选别的你其实也会问同样的问题,就是了,那我(李宏毅本人)的经验是**不同的distribution之间的差异,可能并没有真的非常大**,不过你还是可以找到一些文献,试著去探讨不同的distribution之间,有没有差异

但是这边其实你只要选一个,够简单的distribution就行,因為你的**generator会想办法,把这个简单的distribution,对应到一个复杂的distribution**,所以你可以把选择,distribution这件事情,交给你的generator来处理,那这边我们在等一下的讨论裡面,都假设是一个,normal distribution

Discriminator

在GAN裡面,一个特别的地方就是,除了==generator==以外,我们要多训练一个东西,叫做==discriminator==

discriminator它的作用是,它会拿一张图片作為输入,它的输出是一个数值,这个discriminator本身,也是一个neural network,它**就是一个function**



它输入一张图片,它的输出就是一个数字,它输出就是一个scalar,这个**scalar越大**就代表说,现在输入的这张图片,**越像是真实的二次元人物的图像**

举例来说

这个是二次元人物的头像,那就输出1 假设1是最大的值,那这个也是画得很好的就输出1,这个不知道在画什麼就输出0.1,这个不知道在画什麼就输出0.1

至於discriminator的,neural network的架构啊,这也完全是你自己设计的,所以**generator**,它是个**neural network**,**Discriminator**,也是个**neural network**,他们的**架构长什麼样子,你完全可以自己设计**,你可以用CNN 你可以用,transformer 都可以,**只要你能够產生出你要的输入输出**,就可以了

那在这个例子裡面,像discriminator,因為输入是一张图片,你很显然会选择CNN对不对,CNN在处理影像上有非常大的优势,既然输入是一张图片,那你的discriminator很有可能,裡面会有大量的CNN的架构,那至於**实际上要用什麼样的架构,完全可以自己决定**

Basic Idea of GAN

為什麼要多一个discriminator,这边就讲一个故事,这个故事跟演化是有关的



这不是一片枯叶,它其实枯叶蝶的拟态,那枯叶蝶长得跟枯叶非常像,它可以躲避天敌,那枯叶蝶的祖先,其实也不是长得像枯叶一样,也许他们原来也是五彩斑斕,但為什麼他们变得长得像枯叶一样,是因為有天择的压力

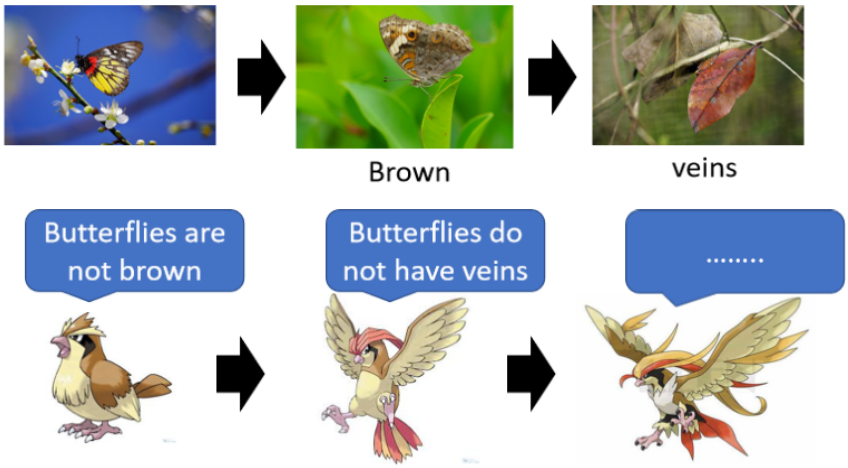






这个不是普通的麻雀 这个是波波,这个波波会吃枯叶蝶的祖先,在天择的压力之下,枯叶蝶就变成棕色的

因为波波它只会吃彩色的东西,它看到彩色的东西知道是蝴蝶,就把它吃掉,那看到棕色的东西,那个波波就觉得是枯叶就可以骗过它,所以枯叶蝶的祖先,在天择的压力之下,颜色就变成是棕色的

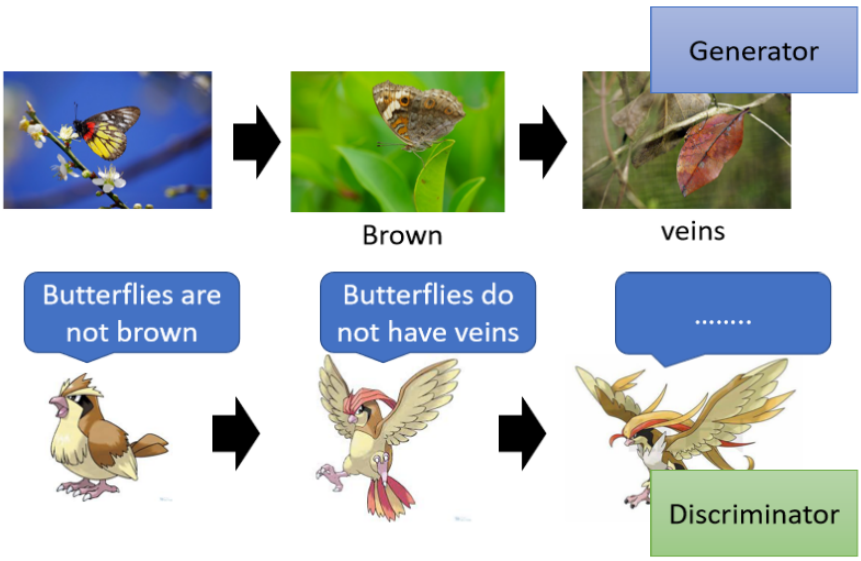


但是波波也是会演化的,所以波波為了要吃到这些枯叶蝶,你有偽装成枯叶的枯叶蝶,所以它也进化了,波波进化以后就是比比鸟这样

比比鸟,它在判断一个蝴蝶能不能吃的时候,是用比较高明的手段,它不会只看颜色 它会看它的纹路,它知道说没有叶脉的是蝴蝶,有叶脉的才是真正的枯叶

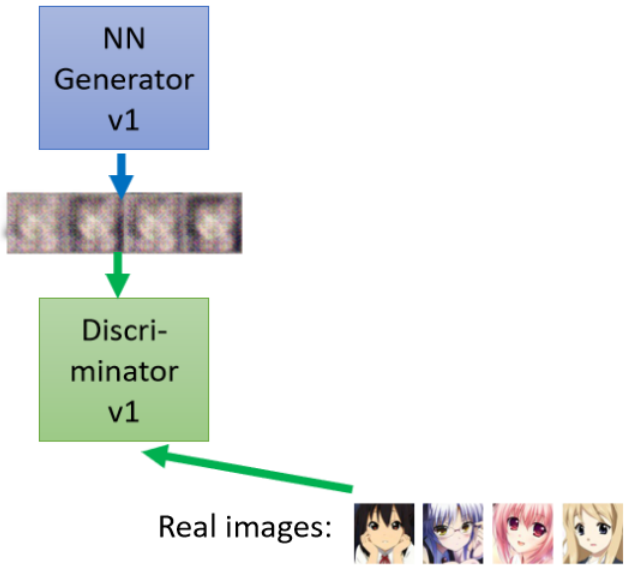
在天择的压力之下,枯叶蝶就產生了拟态 產生了叶脉,想要骗过比比鸟,但是比比鸟它也有可能会再进化,比比鸟进化是什麼,比比鸟进化就是大比鸟

这个就是大比鸟,那大比鸟可能可以分辨,这个枯叶蝶跟枯叶的不同



那这个是演化的故事,对应到GAN 枯叶蝶就是generator,那它的天敌就是discriminator,

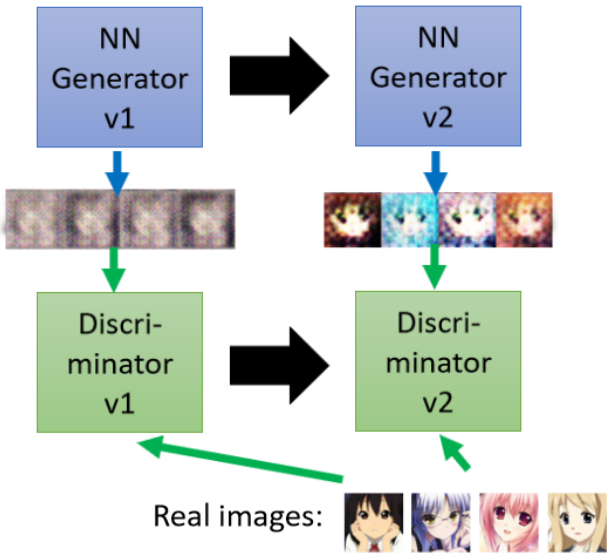
那现在我们generator要做的事情,是画出二次元的人物,那generator怎麼学习画出二次元的人物,它学习的过程是这样子



第一代的generator它的参数几乎是,它的参数完全是随机的,所以它根本就不知道,要怎麼画二次元的人物,所以它画出来的东西就是一些,莫名其妙的杂讯

那**discriminator**接下来,它学习的目标是,要**分辨generator的输出,跟真正的图片的不同**,那在这个例子裡面可能非常的容易,对discriminator来说它只要看说,图片裡面有没有两个黑黑的圆球,就是眼睛,有眼睛就是真正的二次元人物,没有眼睛就是generator,產生出来的东西

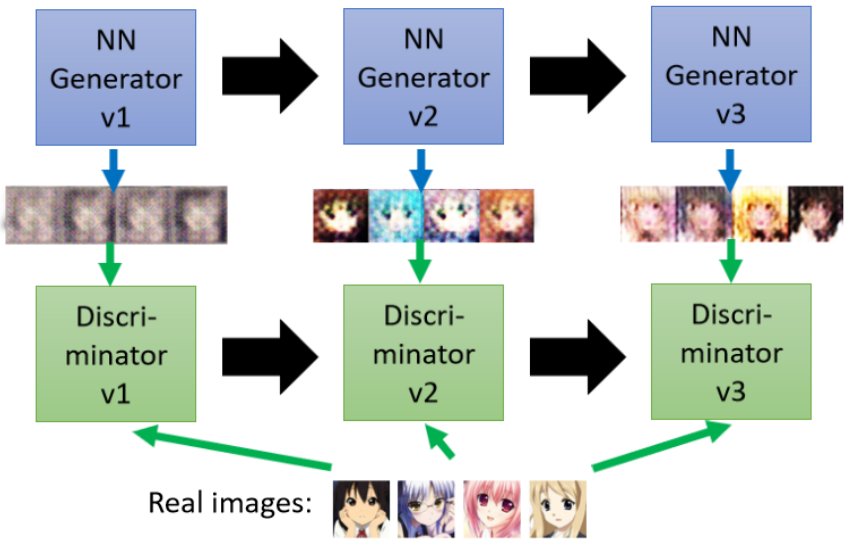
接下来**generator就调整它的裡面的参数**,Generator就**进化**了,它调整它裡面的参数它调整的目标,是為了要骗过discriminator,假设discriminator,判断一张图片是不是真实的依据,看的是有没有眼睛,那generator就產生眼睛出来,给discriminator看



所以generator產生眼睛出来,然后他可以骗过第一代的,discriminator,但是**discriminator也是会进化的**,所以第一代的discriminator,就变成第二代的discriminator,第二代的discriminator,会试图分辨这一组图片,跟真实图片之间的差异,它会试图去找出这两者之间的差异

它发现说,这边產生的图片都是比较简单的,举例来说都没有头髮也没有嘴巴,那这些图片是有头髮的也有嘴巴

接下来第三代的generator,就会想办法,去骗过第二代的discriminator,既然第二代的discriminator是看,有没有嘴巴来判断是不是真正的,二次元人物,那第三代的generator就会把嘴巴加上去



那discriminator也会逐渐的进步,它会越来越严苛,然后期待discriminator越来越严苛,Generator產生出来的图片,就可以越来越像二次元的人物,那因為这边有一个generator,有一个discriminator,它们彼此之间是会互动

最早这个GAN是,Ian Goodfellow propose的,14年这个GAN的paper,是发表在14年的arvix,那最早在这个GAN的原始的,paper裡面,把generator跟discriminator,当作是敌人

如果你有看很多网路文章的话,它都会举例说,啊generator是假钞的啊,然后discriminator是警察啊,警察要抓做假钞的人啊,假钞就会越来越像,警察就会越来越厉害等等

因為觉得generator,跟discriminator中间有一个,**对抗的关係**,所以就用了一个,==adversarial==这个字眼,Adversarial就是对抗的意思,但是至於generator跟discriminator,他们是不是真的在对抗,这只是一个拟人化的说法而已,

所以generator,跟discriminator的关係啊,用动画来说就是写作敌人唸做朋友,就跟进藤光还有塔矢亮一样,或者是跟Naruto跟Sasuke一样



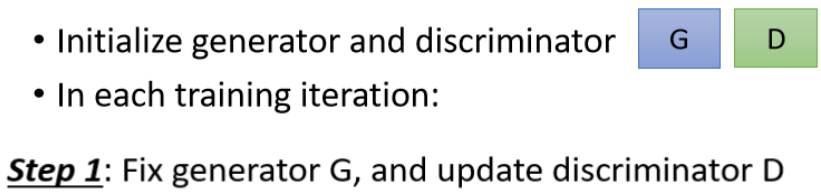
Algorithm

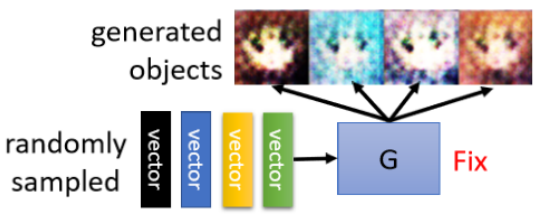
以下就是正式来讲一下,这个演算法实际上是长什麼样子,generator跟discriminator,他们就是两个network

network在训练前,你要先初始化它的参数,所以我们这边就假设说,**generator跟discriminator,它们的参数 都已经被初始化了**

Step 1: Fix generator G, and update discriminator D

初始化完以后,接下来训练的**第一步是,定住你的generator,只train你的discriminator**



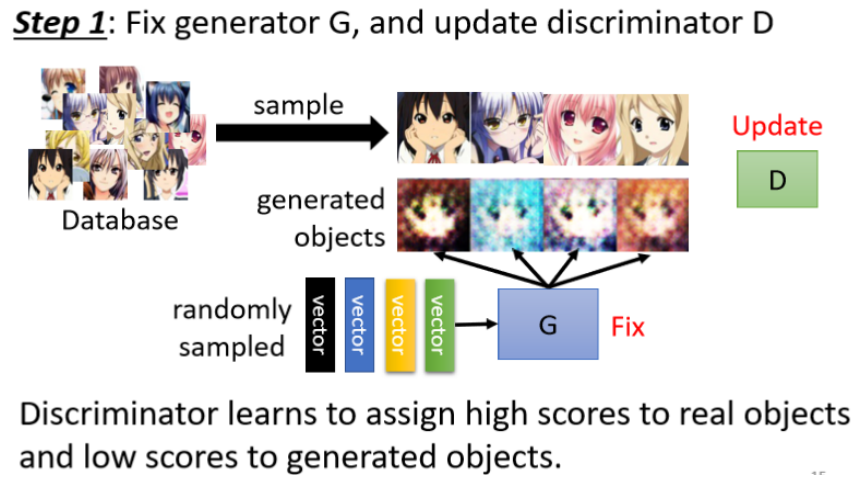


那因為一开始你的**generator的参数,是随机初始化的**,那如果你又**固定住你的generator**,那它根本就什麼事都没有做啊,它的参数都是随机的啊

所以你丢一堆向量给它,它的输出都是乱七八糟的图片,那其实如果generator参数,是初始化的话,你连这样子的结果都產生不出来,那產生出来的就很像是电视机坏掉的,那一种杂讯

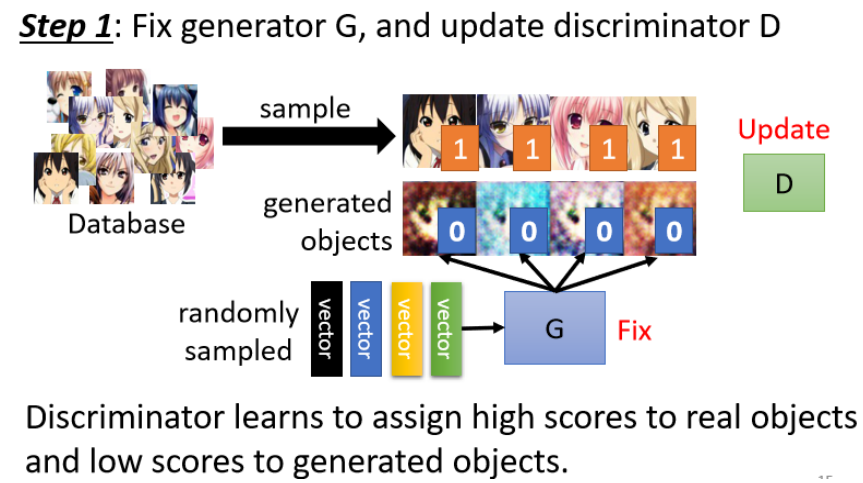
那你从这个gaussian distribution裡面,去random sample一堆vector,把这些vector丢到generator裡面,它就吐出一些图片 一开始这些图片,会跟正常的二次元人物非常的不像

好那你会有一个database,这个database裡面,有很多二次元人物的头像,这个去网路上爬个图库就有了,这个不难蒐集，从这个图库裡面,去sample一些,二次元人物的头像出来



接下来你就拿真正的二次元人物头像,跟generator產生出来的结果,去训练你的discriminator,**discriminator它训练的目标是要分辨,真正的二次元人物,跟generator產生出来的二次元人物**,它们之间的差异

讲得更具体一点啊,你实际上的操作是这个样子,你可能会把这些真正的人物都标1,Generator產生出来的图片都标0



接下来对於discriminator来说,这就是一个**分类的问题**,或者是**regression的问题**

- 如果是分类的问题,你就把真正的人脸当作类别1,Generator產生出来的,这些图片当作类别2,然后训练一个classifier就结束了

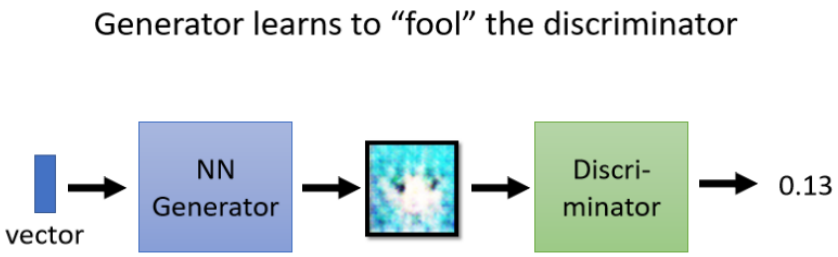
- 或者是有人会把它当作,regression的问题,那你就教你的discriminator说,看到这些图片你就输出1,看到这些图片你就输出0,都可以 总之discriminator就学著,去分辨这个real的image,跟產生出来的image之间的差异

但是实际上怎麼做,你可以当作分类的问题来做,也可以当作regression的问题来做

Step 2: Fix discriminator D, and update generator G

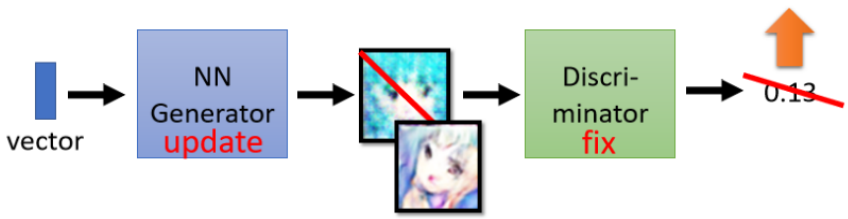
我们训练完,discriminator以后,接下来**定住discriminator改成训练generator**,怎麼训练generator呢

拟人化的讲法是,我们就**让generator想办法去骗过discriminator**,因為刚才discriminator,已经学会分辨,真图跟假图的差异,真图跟生成的图片的差异,Generator如果可以骗过,discriminator它可以產生一些图片,**Discriminator觉得,是真正的图片的话,那generator產生出来的图片,可能就可以以假乱真**



它实际上的操作方法是这样子,你有一个generator,generator吃一个向量作為输入,从gaussian distribution sample,出来的向量作為输入,然后產生一个图片

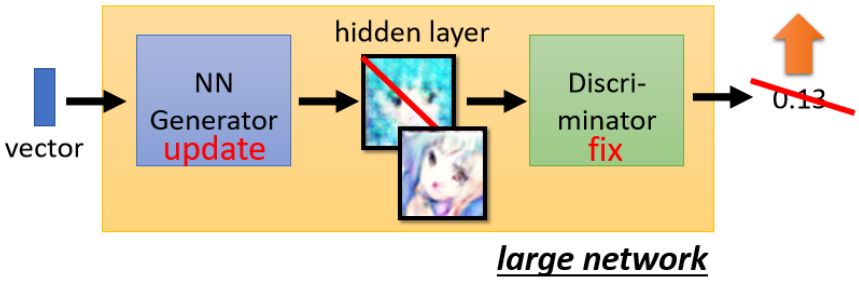
接下来我们把这个图片丢到,Discriminator裡面,Discriminator会给这个图片,一个分数,那generator它训练的目标,就Discriminator参数是固定的,我们只会调整generator的参数



Generator训练的目标,是要Discriminator的输出值,越大越好,那因為Discriminator,它本来训练的时候,它训练的目标它可以做的事情就是,看到好的图片就给它大的分数,如果generator可以调整参数之后,输出来的图片Discriminator,会给予高分,那意味著generator產生出来的图片,是比较真实的

得更具体一点,实际上你的操作是这个样子,**Generator是一个network裡面有好几层,Discriminator也是一个,network裡面有好几层**,我们把**generator跟Discriminator直接接起来,当做一个比较大的network来看待**

举例来说generator,如果是五层的network,Discriminator如果是五层的network,把它们接起来我们就把它当作是一个,十层的network来看待



而这个十层的network裡面,**某一个hidden layer它的输出很宽**,它的输出的这个dimension呢,就跟图片裡面pixel的数目,乘三是一样的,你把这个hidden layer的输出呢,做一下整理以后 就会变成一张图片,所以这整个大的network裡面,其中某一层的输出就是代表一张图片

我们要做的事情是,整个巨大的network啊,它会吃一个**向量作為输入**,然后他会**输出一个分数**,那我们希望调整这个network,让输出的**分数越大越好**

但是要注意一下 **我们不会去调,对应到Discriminator的部分**,我们不会去调这个巨大,network的**最后几层**,

為什麼不调最后几层呢

你可以想想看假设调最后几层的话,这整个游戏就被hack了,因為假设你要输出的分数越大越好,我直接调最后output layer,那个neural bias,把它设成一千万,那不是输出就很大了吗,所以Discriminator这边的参数,是不能动的,

我们**只调generator的参数**,好那这边呢,至於**怎麼调Generator的参数呢,这个训练的方法啊,跟我们之前训练一般的network,是没有什麼不同的**

我们之前说训练network的时候就是,定一个loss啊 然后你用gradient descent,让loss越小越好,那这边呢 你也有一个目标,只是这个**目标呢不是越小越好,而是越大越好**,那当然你也可以把这个目标,Discriminator output成一个负号,就当作loss你可以把Discriminator,output成一个负号当作loss,然后generator训练的目标,就是让loss越小越好

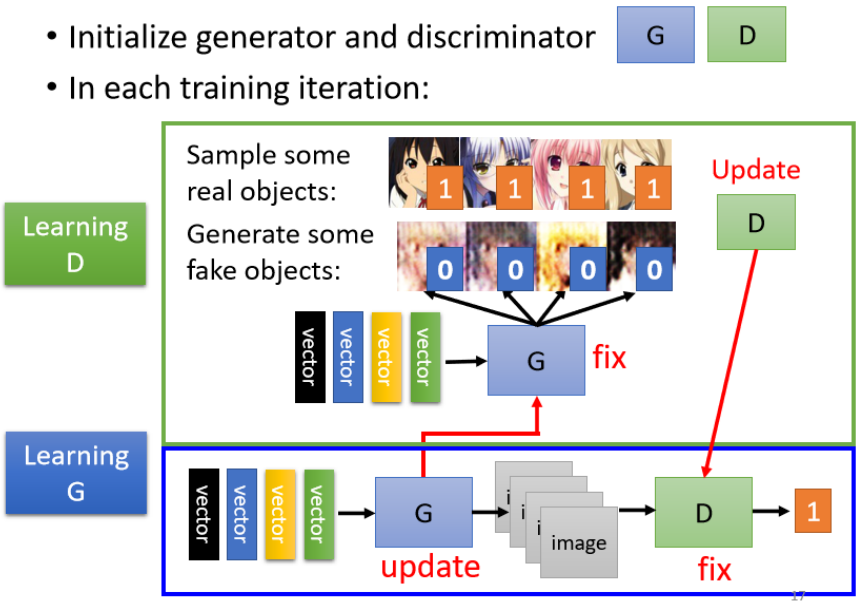
或者你也可以说,我们就是要让Discriminator output,的值越大越好,然后我们用==gradient ascent==,不是gradient descent,gradient descent是,让loss越小越好,gradient ascent会让你的目标函数,越大越好,我们会用gradient ascent去调generator,让Discriminator的输出越大越好

这是同一件事,这边训练generator的方法,也是用gradient descent base的方法,跟我们之前在训练一个,一般network的时候,是没有什麼差异的

所以现在讲了两个步骤

- 第一个步骤 固定generator,训练discriminator
- 第二个步骤,固定discriminator训练generator

接下来就是**反覆的训练**,discriminator跟generator,训练完discriminator以后,固定住discriminator,训练generator,训练完generator以后,再用generator去產生更多的,新的產生出来的图片,再给discriminator做训练,训练完discriminator以后,再去训练generator,反覆的去执行



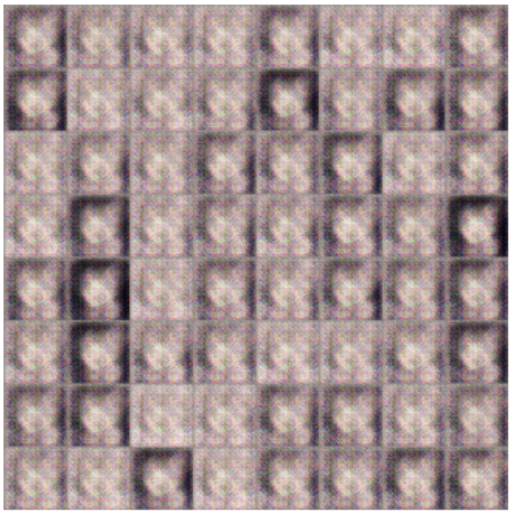
所以你是训练一阵子discriminator,训练一阵子generator,训练一阵子discriminator,再训练一阵子generator,Generator跟discriminator,它们是反覆的去进行训练,当其中一种进行训练的时候,另外一个就固定住,那你期待discriminator跟generator,都可以做得越来越好,

Anime Face Generation

下一个作业就是,要做动画人物的人脸生成,那你可能会问说,到底可以做到什麼样的程度呢

以下的结果是我在17年的时候做的 Source of training data: https://zhuanlan.zhihu.com/p/24767059,我自己试著train了一下GAN,看看GAN是不是真的可以產生,二次元的人物

好那我训练了,我把那个generator呢,Update了一百次以后,所谓generator update 一百次,的意思是说,就是discriminator train一下,generator train一下,discriminator train一下,generator train一下,这样往返一百次以后得到的结果,是这样子



100 updates

Source of training data: https://zhuanlan.zhihu.com/p/24767059

嗯 不知道在做些什麼,但我接下来呢就再等了一下,Train 一千次的



1000 updates

discriminator 跟generator,各自训练这样反覆一千次以后,机器就產生了眼睛,机器知道说 人脸就是要有两个眼睛,所以它就把眼睛标上去,训练到两千次的时候,你发现嘴巴就出来了

2000 updates



训练到五千次的时候,已经开始有一点人脸的样子了,而且你发现说机器学到说,动画人物啊,就是要有那个水汪汪的大眼睛,所以他每个人的眼睛呢,都涂得非常的大,涂有反白 代表说反光,是水汪汪的大眼睛

5000 updates

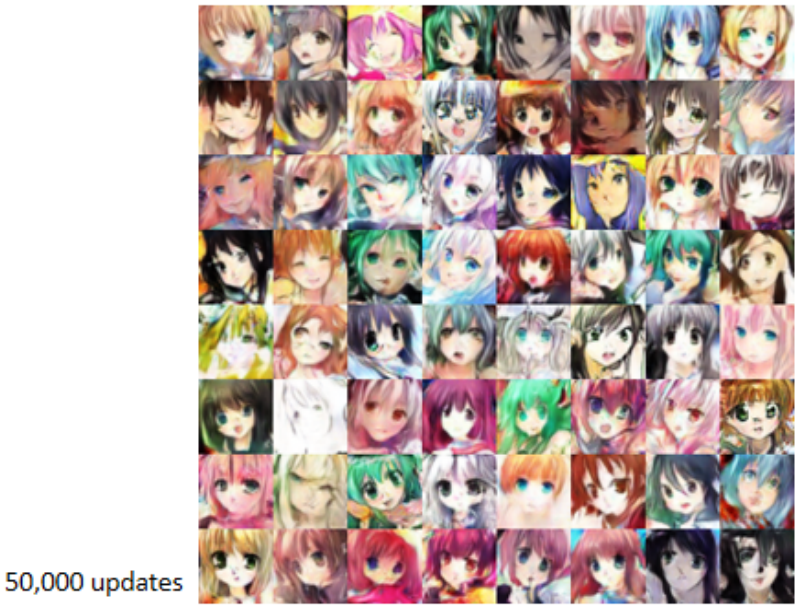


这个是训练一万次以后的结果,有发现形状已经有出来了,只是有点模糊,很多地方有点晕开的感觉,好像是水彩画的样子,

接下来这个是,update两万次的结果



这个是update五万次的结果



我后来就停在五万次的地方,那其实你在作业裡面,是有机会做得比这个结果更好的,这个是助教是学生的时候做的结果啦,那如果是最好,可能可以做到这个样子

那你会发现说这些人物呢都还不错,只是有一些比较,还是会有偶尔会有一些崩坏啦,但乍看之下呢可能比一些作画画风,会崩坏的动画公司,比如说一些妹非妹做的还要好一些了,

如果你有好的资料库的话,那当然我们提供给大家的资料,是做不到这个地步的啦,如果你有真的非常好的资料的话,也许你可以做出真的很好的结果



https://www.gwern.net/images/gan/stylegan/2019-02-11-stylegan-danbooru2017faces-interpolation.mp4

我在网路上呢,找到了一个这样子的结果,这个是用StyleGAN做的,那用StyleGAN做起来,可以做到这个样子



我觉得非常惊人喔,很惊人喔 这些都是,用GAN產生出来的人物,这边他还產生了异色瞳,我不知道算是画错呢还是它特意呢,要產生异色瞳,对异色瞳,就一眼白眼一眼血轮眼这样子的概念,

好那除了產生动画人物以外,当然也可以產生真实的人脸,有一个技术叫做**progressive GAN**,它可以產生非常高清的人脸

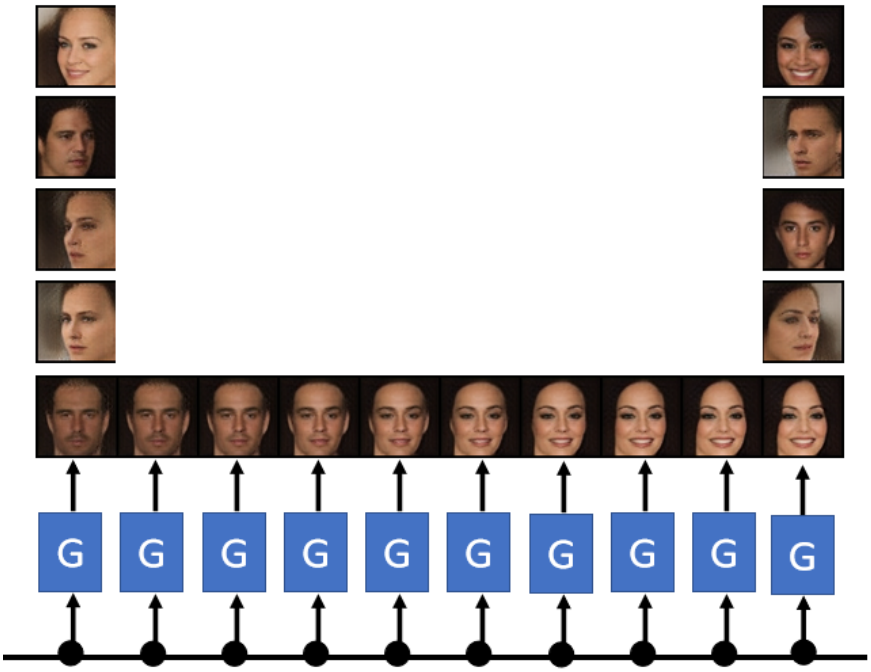
上下两排都是机器產生的,好所以这个显然progressive GAN,它有办法產生以假乱真的人脸

甚至之前啊 你不知道我有听,有一个新闻我不知道是不是真的,有一个新创公司 它裡面有很多人,但大家发现裡面那些人的头像,有点怪怪的,有人说那些头像其实,是用GAN生成的,那并不是真正的人物,那个公司没有那麼多人,用GAN呢生成一些假人的照片,当作是假员工,

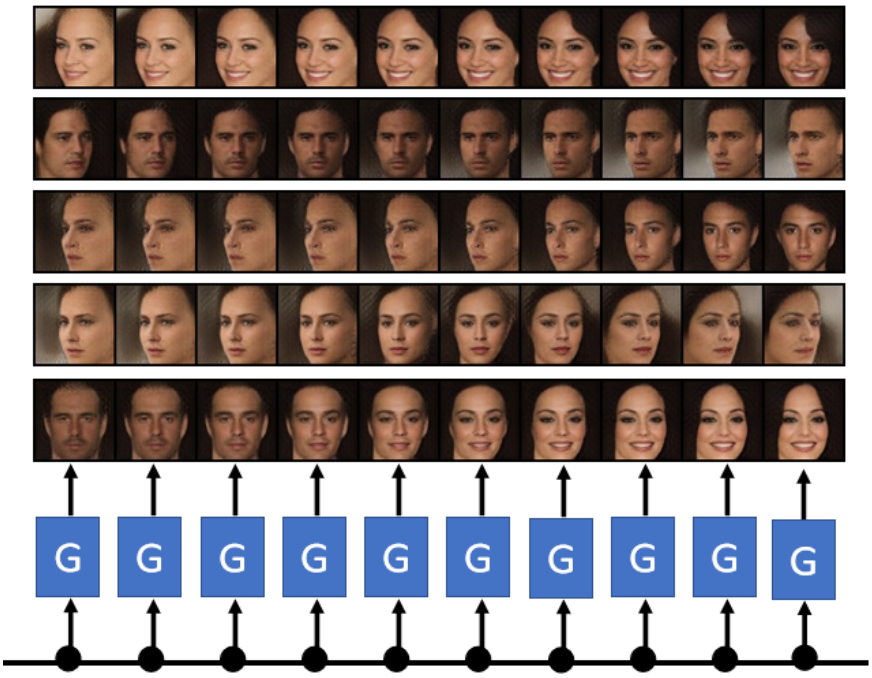
那你可能会问说要產生人脸,有什麼用呢 我去路边拍一个人,產生出来的照片不是更像真的吗

但是用**GAN你可以產生,你没有看过的人脸**,举例来说用GAN,你可以做到这样子的事情,我们刚才说GAN这种generator,就是输入一个向量 输出一张图片,那你不只可以输入一个向量,输出一张图片

你还可以把输入的向量,做内插 做interpolation,把输入的向量做内插以后,会发生什麼事呢,你就**会看到两张图片之间连续的变化**



举例来说你输入一个向量,这边產生一个看起来非常严肃的男人,你输入一个向量,这边產生一个笑口常开的女人,那你输入这两个向量中间的,interpolation它的内插,你就看到这个男人逐渐的笑了起来,或者是呢这边有更多的例子



你输入一个向量,这个输入的向量这边是假的啦,但这边產生出来的图片是真的,你输入一个向量,这边產生一个往左看的人,你输入一个向量,这边產生一个往右看的人,你把往左看的人跟往右看的人,做interpolation会发生什麼事呢

机器并不是傻傻地,把两张图片叠在一起,变成一个双面人,而是机器知道说,往左看的人脸跟往右看的人脸,介於他们中间的就是往正面看,你在训练的时候其实并没有真的告诉,机器这件事 但机器可以自己学到说,把这两张脸做内插,应该会得到一个往正面看的人脸,

刚才已经讲过说,GAN是Ian Goodfellow,在14年的时候提出来的,你可能有听过那个故事是,这不知道是不是真的啦一个传说,Ian Goodfellow去酒吧,看到两个人吵架,於是就有了GAN的灵感,然后呢回家第一次实作,就成功了这样

然后就结束,然后就投了一个paper,那但是他所谓的成功啊,其实是长这个样子的



在14年的时候,我第一看到这个结果的时候,我觉得哇靠还真的可以產生图片,太厉害了,当然如果从今天的角度来看,你会觉得说 这样你也算是有成功吗,今天比如说你用,BigGAN產生出来的图片,可以做到像这个样子



这些图片都是机器生成的,当然仔细看一下,还是可以发现一些破绽,举例来说这隻狗它多了一个脚啊,或者是这个杯子,它左右没有很对称啊,它有点歪歪的,但这些图片都是机器生成的,那有时候机器,也会產生一些幻想中的角色,举例来说机器就產生了一个网球狗啊,



GAN_P2_Theory behind GAN

P1是用了一堆比喻说明GAN 的操作是怎麼进行的

接下来,我们告诉你说实际上,為什麼这个 **Generator 跟 Discriminator 的互动**,可以让我们的 **Generator產生像是真正的人脸的图片**,那这背后的互动在做的,到底是什麼样的事情

那我们先来弄清楚,我们今天的**训练的目标**到底是什麼

我们在训练 Network 的时候,你就是要

- 定一个 Loss Function
- 定完以后用 Gradient Descent 去调你的参数
- 去 **Minimize** 那个 Loss Function 就结束了

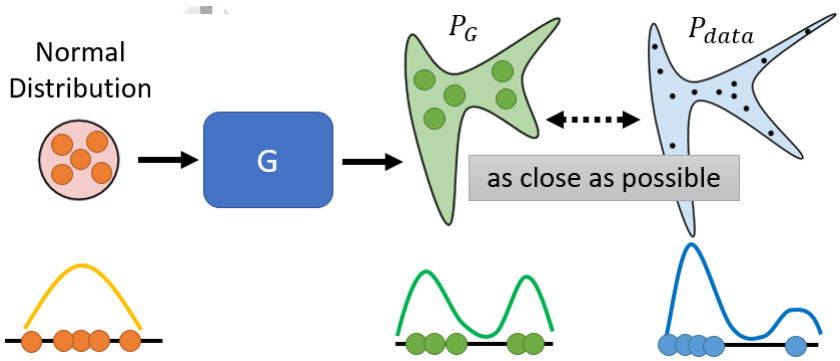
在这个 Generation 的问题裡面,我们要 **Minimize,或者是 Maximize 什麼样的东西**,我们要把这些事弄清楚,才能够做接下来的事情

我们想要 **Minimize** 的东西是这个样子的，我们有一个 Generator

- 给它一大堆的 Vector,给它从 Normal Distribution Sample 出来的东西
- 丢进这个 Generator 以后,会產生一个比较复杂的 Distribution,这个复杂 Distribution,我们叫它 ==PG==
- 然后 我们有一堆的 Data,这个是真正的 Data,真正的 Data 也形成了另外一个 Distribution,叫做 ==Pdata==,我们期待 **PG 跟 Pdata 越接近越好**

Our objective

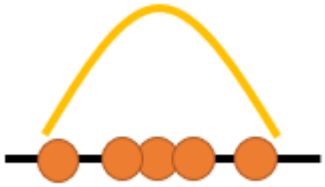
如果你一下子没有办法想像,这个 PG 、Pdata 是怎麼一回事的话,那我们用**一维的状况**来跟大家说明



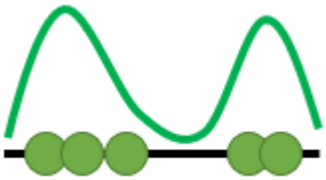
我们假设

- Generator 的 Input 是一个一维的向量
- Generator 的 Output 也是一维的向量
- 我们真正的 Data 也是一维的向量

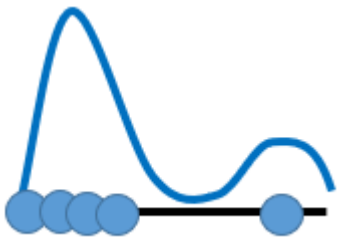
那我们的 Normal Distribution 就长这个样子



那丢到 Generator 以后,假设你输入 5 个点,那边这每一个点,它的位置会改变,那你就產生一个新的 Distribution,那可能本来大家都集中在中间,通过这个 Generator,通过一个 Network 裡面很复杂,不知道做了什麼事情以后,这些点就分成两边,所以你的 Distribution 就变成这个样子



而 **Pdata** 是指真正的资料的分布,真正资料分布可能长这个样子



它分两面的状况是更极端的,左边的东西比较多,右边的东西比较少,那你期待左边这个分布跟右边这个分布,越接近越好,如果写成式子的话

你可以写成这个样子

$$G^* = arg \min_G Div(P_G, P_{data})$$

Divergence between distributions P_G and P_{data}

Div Of PG 跟 Pdata,它指的意思就是 PG 跟 Pdata,这两个 Distribution 之间的 ==Divergence==(散度)

Divergence 这边指的意思就是,这两个你可以想成是,这**两个 Distribution 之间的某种距离**

- 如果这个 **Divergence 越大**,就代表这**两个 Distribution 越不像**
- **Divergence 越小**,就代表这两个 **Distribution 越相近**

Divergence 这就是衡量,两个的 Distribution 相似度的一个 Measure

我们现在的目标,是要去**找一个 Generator**,找一个 Generator找个操作,实际上做的事情是找一个 **Generator 裡面的参数**, Generator 也是一个 Network,裡面有一大堆的 Weight 跟 Bias,它可以让我们產生出来的 PG 跟 Pdata,之间的 Divergence 越小越好,我要找的就是这样子的 Generator

这边把它写作 $G^$,*所以我们这边要做的事情,*跟一般 Train Network 其实非常地像*,我们第一堂课就告诉你说,我们定义了 Loss Function,找一组参数 Minimize Loss Function

在 Generation 这个问题裡面,我们现在其实也定义了我们的 **Loss Function**,它就是 PG 跟 Pdata 的 **Divergence**,就是它们两个之间的距离,**它们两个越近,那就代表这个產生出来的 PG 跟 Pdata 越像**,所以 PG 跟 Pdata,我们希望它们越相像越好

但是我们这边遇到一个**困难的问题**

$$G^* = arg \min_G Div(P_G, P_{data})$$

Divergence between distributions P_G and P_{data}

How to compute the divergence?

这个 Loss,我们是可以算的,但是这个 Divergence 是要怎麼样算? 那你可能知道一些 Divergence 的式子,比如说 KL Divergence,比如说 JS Divergence,这些 Divergence 用在这种 Continues 的,Distribution 上面,你要做一个很复杂的,在实作上你**几乎不知道要怎麼算的积分,那我们根本就无法把这个 Divergence 算出来**

我们算不出这个 Divergence,我们又要如何去找一个 G,去 Minimize 这个 Divergence ,这个就是 GAN 所遇到的问题,这就是我们在 Train 这一种,Generator 的时候会遇到的问题

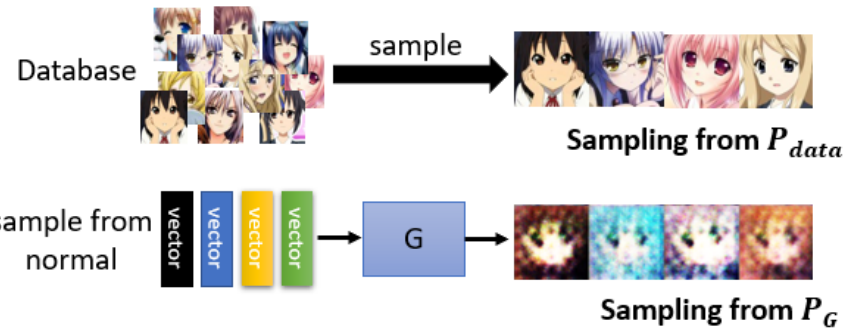
而 **==GAN==是一个很神奇的做法,它==可以突破,我们不知道怎麼计算 Divergence 的限制==**

Sampling is good enough ……

所以我现在**遇到的问题**就是,**不知道怎麼计算 Divergence**,而 GAN 告诉我们就是,你**不需要知道 PG 跟 Pdata它们实际上的 Formulation 长什麼样子**，只要能**从 PG 和 Pdata这两个 Distributions Sample 东西出来,就有办法算 Divergence**

$$G^* = arg \min_G Div(P_G, P_{data})$$

Although we do not know the distributions of P_G and P_{data}, we can sample from them.



怎麼从真正的 Data 裡面,Sample 出东西来，怎麼从 Generator 裡面,產生一些东西出来?

- 把你的图库拿出来,从图库裡面随机產生,随机 Sample 一些图片出来,你就得到 **Pdata**

- 那你就把你的 Generator,输入这个 Normal 的,从 ==Normal Distribution== Sample 出来的 Vector,丢到 Generator 裡面（我们刚才说过,你这边的拿来 Sample 那个 Distribution,要是你有办法 Sample 的,所以我们选 Normal Distribution式子我们是知道的,是有办法 Sample 的) 我们从 Normal Distribution 裡面,Sample 一堆 Vector 出来丢给 Generator,让 Generator 產生一堆图片出来,那这些图片就是从 **PG** Sample 出来的结果

所以我们有办法从 PG 做 Sample,我们有办法从 Pdata 做 Sample

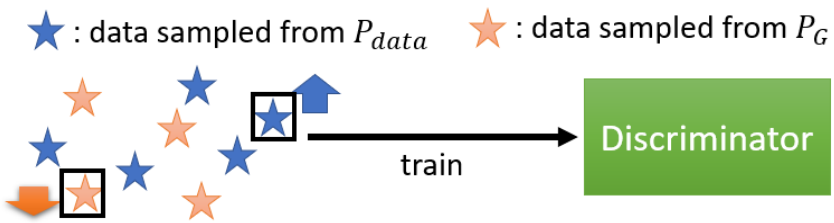
Discriminator

接下来,GAN 这一整个系列的 Work,就是要告诉你说,怎麼在**只有做 Sample 的前提之下**,我根本**不知道 PG 跟 Pdata,实际上完整的 Formulation 长什麼样子**,居然就估测出了 Divergence

那这个就是要**靠 ==Discriminator== 的力量**

那我们刚才讲过说 Discriminator是怎麼训练出来的

- 我们有一大堆的 Real Data,这个 Real Data 就是从 Pdata Sample 出来的结果
- 我们有一大堆 Generative 的 Data,Generative 的 Data,就可以看作是从 PG Sample 出来的结果



根据 Real 的 Data 跟 Generative 的 Data,我会去训练一个 Discriminator,它的训练的目标是

- 看到 **Real Data**,就给它比较**高的分数**
- 看到这个 **Generative 的 Data**,就给它比较**低的分数**

Discriminator 训练的目标,就是要**分辨好的图跟不好的图**,分辨真的图跟生成的图,所以**看到真的图**给它**高分**,**看到生成的图**给它**低分**

实际以上的过程,你也可以把它写成式子,把它当做是一个 **Optimization 的问题**,这个 Optimization 的问题是这样子的

Training: $D^* = arg \max_D V(D,G)$

Objective Function for D

$$V(G,D) = E_{y\sim P_{data}}[logD(y)] + E_{y\sim P_G}\big[log\big(1-D(y)\big)\big]$$

这个 Discriminator 可以去 **Maximize**某一个 Function,我们这边叫做 ==Objective Function==（我们要 **Maximize** 的东西,我们会叫 **Objective Function**,如果 **Minimize** 我们就叫它 **Loss Function**）

我们现在要**找一个 D**,它可以 Maximize 这个 Objective Function,这个 Objective Function 长这个样子

- $E{*y**sim P*{data}}[logD(y)]$ 我们有一堆 Y,它是从 Pdata 裡面 Sample 出来的,也就是它们是真正的 Image,而我们把这个真正的 Image 丢到 D 裡面,得到一个分数再取$logD(y)$
- $E_{y\sim P_G}[log(1-D(y))]$ 那另外一方面,我们有一堆 Y,它是从 PG 从 Generato r 所產生出来的,把这些图片也丢到 Discriminator 裡面,得到一个分数,再取 $log(1 - D (y))$

我们希望这个 **Objective Function V越大越好**

Objective Function for D

$$V(G,D) = E_{y\sim P_{data}}[logD(y)] + E_{y\sim P_G}[log(1-D(y))]$$

- 意味著我们希望这边的 D (Y) 越大越好,我们希望 Y 如果是从 **Pdata Sample 出来的**,它就要**越大越好**
- 我们希望说如果 Y 是从,这个 **PG Sample 出来的**,它就要**越小越好**

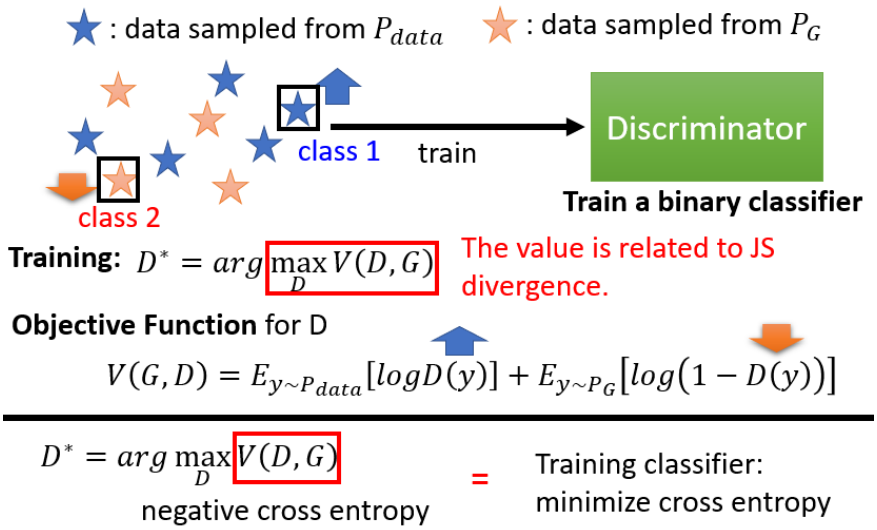
让 D (Y) 越大越好,也就是让Discriminator Output 的值越小越好

你可能觉得没事突然写出这个式子有点奇怪,但你**不一定要把这个 Objective Function,写成这个样子**,它完全可以有其他的写法,那最早年之所以写成这个样子,是有一个很明确的理由,是為了要把 **Discriminator跟 Binary Classification**,跟二元的分类**扯上关係**

事实上这个 Objective Function,它就是 **Cross Entropy 乘一个负号**,我们知道我们在训练一个 Classifier 的时候,我们就是要 Minimize Cross Entropy,所以当我们 **Maximize Cross Entropy 乘一个负号**的时候,其实等同於 **Minimize Cross Entropy**,也就是等同於是在**训练一个 Classifier**

那这个 Discriminator,其实可以当做是一个 Classifier,它做的事情就是把蓝色这些点,从 Pdata Sample 出来的真实的 Image,当作 Class 1,把从 PG Sample出来的这些假的 Image,当作 Class 2

有两个 Class 的 Data,训练一个 Binary 的 Classifier,训练完就等同於是,解了这一个 Optimization 的问题



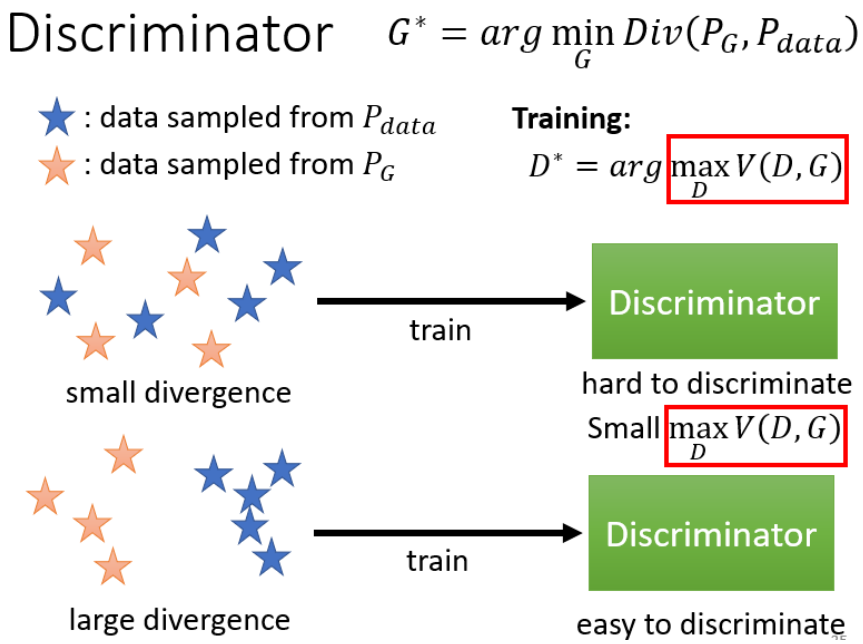
那这边最神奇的地方是这一个式子,这个**红框框裡面的数值,它跟 JS Divergence 有关**

那事实上有趣的事情是,我觉得最原始的 GAN 的 Paper,它的发想可能真的是从 Binary Classifier 来的,一开始是把 Discriminator,写成 Binary 的 Classifier,然后有了这样的 Objective Function,然后再经过一番推导以后,这个 Objective Function,它的 Maximum,就是你找到一个 D,可以让这个 Objective Function,它的值最大的时候,这个最大的值跟 JS Divergence 是有关的。它们没有完全一模一样,所以显然一开始,并不是针对 JS Divergence 设计的,而是经过一番推导以后,发现它们是非常有关联的,那至於实际上的推导过程,你可以参见原始 Ian J. Goodfellow 写的文章,那其实裡面的推导过程,我觉得写得算是蛮清楚的

所以真正神奇的地方就是,这一个 **Objective Function 的最大值,它跟 Divergence 是有关的**,所以我们刚才说,我们不知道怎麼算 Divergence没关係,Train 你的 Discriminator,Train 完以后,看看它的 Objective Function 可以到多大,那个值就跟 Divergence 有关

这边我们并没有把证明拿出来跟大家讲了,但是我们还是可以从**直观上**来理解一下,為什麼这个 **Objective Function 的值,会跟 Divergence 有关**

你可以想想看,假设 PG 跟 Pdata,它的 **Divergence 很小**



- 也就 **PG 跟 Pdata 很像**,它们差距没有很大,它们很像 PG 跟 Pdata Sample 出来的,蓝色的星星跟红色的星星,它们是**混在一起的**

  这个时候Discriminator 就是在 Train 一个,Binary 的 Classifier,对 Discriminator 来说,既然这两堆资料是混在一起的,那就**很难分开**,这个问题很难,所以你在解这个 Optimization Problem 的时候,你就没有办法让这个 Objective 的值非常地大

  所以这个 Objective 这个 V 的,Maximum 的值就比较小,所以小的 Divergence,对应到小的这个 Objective Function 的Maximum 的值

  所以不是 Objective Function 的值本身,是 Objective Function 在穷举地找 Discriminator 要可以得到的最大的值,

- 如果今天,你的**两组 Data 很不像**,它们的 Divergence 很大,那对 Discriminator 而言,就可以轻易地把它分开

  当 Discriminator 可以轻易把它分开的时候,这个 Objective Function 就可以冲得很大,那所以当你有大的 Divergence 的时候,这个 Objective Function 的 Maximum 的值,就可以很大

详细的证明请参见 GAN 原始的 Paper,裡面在做了一些假设,比如说 Discriminator 的 Capacity 是无穷大,等等的假设以后,可以做出这个 Maximum 的值,跟 JS Divergence 一些相关的这个推导

所以我们说我们本来的目标是要找一个 Generator,去 Minimize PG 跟 Pdata 的 Divergence

$$G^* = arg \min_G Div(P_G, P_{data})$$

但我们卡在**不知道怎麼计算 Divergence**,那我们现在要知道,我们只要训练一个 Discriminator,训练完以后,这个 Objective Function 的最大值,就是这个 Divergence,就跟这个 Divergence 有关

$$G^* = arg \min_G Div(P_G, P_{data})$$

$$D^* = arg \max_D V(D,G)$$

The maximum objective value is related to JS divergence.

那我们何不就把红框框裡面这一项,跟 Divergence 做替换,我们何不就把 Divergence,替换成红框框裡面这一项,所以我们就有了这样一个 Objective Function

$$G^* = arg \min_G \max_D V(G,D)$$

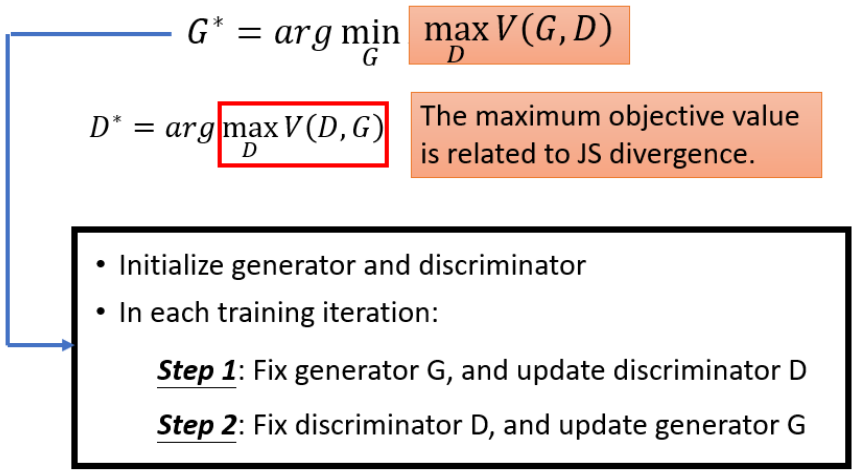
$$D^* = arg \max_D V(D,G)$$

The maximum objective value is related to JS divergence.

这个 Objective Function 乍看之下有点复杂,它**有一个 Minimum,又有一个 Maximum**,所以你不小心就会脑筋转不过来

- 我们是要找一个 **Generator**,去 Minimize 红色框框裡面这件事
- 但是红框框裡面这件事,又是另外一个 Optimization Problem,它是在**给定 Generator** 的情况下,去找一个 Discriminator,这个 **Discriminator,可以让 V 这个 Objective Function 越大越好**

然后我们要找一个 G,让红框框裡面的值最小,这个 G 就是我们要的 Generator,而刚才我们讲的这个 Generator 跟 Discriminator,互动 互相欺骗这个过程,其实就是想解这一个有 Minimize,又有 Maximize 这个 Min Max,Min Max 的问题,就是透过下面这一个,我们刚才讲的 GAN 的 Argument 来解的



那至於实际上,為什麼下面这个 Argument 可以解这个问题,你也可以参见原始 GAN 的 Paper

那讲到这边,也许你就会问说,**為什麼是 JS Divergence,而且还不是真的 JS Divergence**,是跟 JS Divergence 相关而已,**怎麼不用真正的 JS Divergence,或不用别的,比如说 KL Divergence**

你完全可以这麼做,你只要改了那个 Objective Function,你就可以量各式各样的 Divergence,那至於怎麼样设计 Objective Function,得到不同的 Divergence,那有一篇叫做 F GAN 的 Paper 裡面,有非常详细的证明

Can we use other divergence?

Name	$D_f(P\|Q)$	Generator $f(u)$
Total variation	$\frac{1}{2}\int \|p(x)-q(x)\|\,\mathrm{d}x$	$\frac{1}{2}\|u-1\|$
Kullback-Leibler	$\int p(x)\log\frac{p(x)}{q(x)}\,\mathrm{d}x$	$u\log u$
Reverse Kullback-Leibler	$\int q(x)\log\frac{q(x)}{p(x)}\,\mathrm{d}x$	$-\log u$
Pearson χ^2	$\int \frac{(q(x)-p(x))^2}{p(x)}\,\mathrm{d}x$	$(u-1)^2$
Neyman χ^2	$\int \frac{(p(x)-q(x))^2}{q(x)}\,\mathrm{d}x$	$\frac{(1-u)^2}{u}$
Squared Hellinger	$\int \left(\sqrt{p(x)}-\sqrt{q(x)}\right)^2\,\mathrm{d}x$	$(\sqrt{u}-1)^2$
Jeffrey	$\int (p(x)-q(x))\log\left(\frac{p(x)}{q(x)}\right)\,\mathrm{d}x$	$(u-1)\log u$
Jensen-Shannon	$\frac{1}{2}\int p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\,\mathrm{d}x$	$-(u+1)\log\frac{1+u}{2}+u\log u$
Jensen-Shannon-weighted	$\int p(x)\pi\log\frac{p(x)}{\pi p(x)+(1-\pi)q(x)}+(1-\pi)q(x)\log\frac{q(x)}{\pi p(x)+(1-\pi)q(x)}\,\mathrm{d}x$	$\pi u\log u-(1-\pi+\pi u)\log(1-\pi+\pi u)$
GAN	$\int p(x)\log\frac{2p(x)}{p(x)+q(x)}+q(x)\log\frac{2q(x)}{p(x)+q(x)}\,\mathrm{d}x-\log(4)$	$u\log u-(u+1)\log(u+1)$

Using the divergence you like ☺

https://arxiv.org/abs/1606.00709

Name	Conjugate $f^*(t)$
Total variation	t
Kullback-Leibler (KL)	$\exp(t-1)$
Reverse KL	$-1-\log(-t)$
Pearson χ^2	$\frac{1}{4}t^2+t$
Neyman χ^2	$2-2\sqrt{1-t}$
Squared Hellinger	$\frac{t}{1-t}$
Jeffrey	$W(e^{1-t})+\frac{1}{W(e^{1-t})}+t-2$
Jensen-Shannon	$-\log(2-\exp(t))$
Jensen-Shannon-weighted	$(1-\pi)\log\frac{1-\pi}{1-\pi e^{t/\pi}}$
GAN	$-\log(1-\exp(t))$

37

它有很多的 Table 告诉你说,不同的 Divergence,要怎麼设计它的 Objective Function,你设计什麼样的 Objective Function,去找它的 Maximum Value,就会变成什麼样的 Divergence,在这篇文章裡https://arxiv.org/abs/1606.00709面都有详细的记载,这一开始有人会觉得说,GAN 之所以没有很好 Train,也许是因為,就是我们没有在真的,这个 Minimize JS Divergence

但是有了这个 F GAN 这个 Paper以后,它就告诉你说,我们有办法 Minimize JS Divergence,但**就算你真的可以 Minimize JS Divergence,结果也还是没有很好**,GAN 还是没有很好 Train

所以俗话就说,No Pain No Gan

GAN is difficult to train .....



(I found this joke from 陳柏文's facebook.)

所以接下来我们就要讲一些,GAN 训练的小技巧

Tips for GAN

其实它 GAN 训练的小技巧非常非常多,我们只挑一个最知名的来讲,这个最知名的就是很多人都听过,就是很多人都听过的 ==WGAN==

在讲 WGAN 之前,我们先讲 **JS Divergence 有什麼样的问题**，在最早的 GAN 说,我们要 Minimize 的是 JS Divergence,那选择 JS Divergence 的时候,会有什麼问题

JS divergence is not suitable

在想 JS Divergence 的问题之前,我们先看一下 **PG 跟 Pdata 有什麼样的特性**

那 PG 跟 Pdata 有一个非常关键的特性是,**PG 跟 Pdata 它们重叠的部分往往非常少**

- In most cases, P_G and P_{data} are not overlapped.

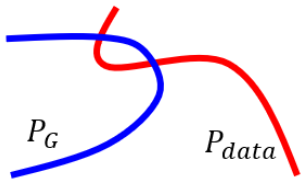
这边有两个理由

- 第一个理由是来自於 Data 本身的特性

  - 1. The nature of data

    Both P_{data} and P_G are low-dim manifold in high-dim space.

    The overlap can be ignored.

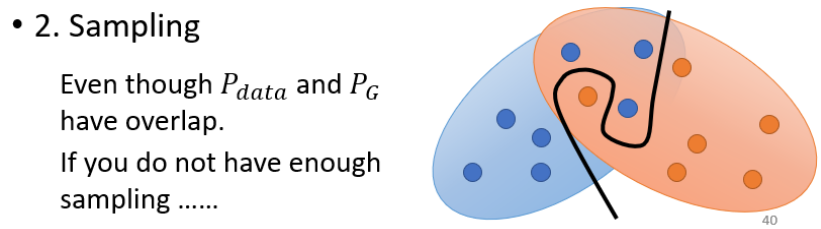
    

  PG 跟 Pdata,它们都是要產生图片的,那图片其实是高维空间裡面的一个低维的 Manifold(**流形**（英语：Manifolds）是可以局部欧几里得空间化的一个拓扑空间，是欧几里得空间中的曲线、曲面等概念的推广),怎麼知道图片是高维空间,裡面低维的 Manifold

  我们可以想想看,你在高维空间裡面,随便 Sample 一个点,它通常都没有办法构成一个,二次元人物的头像,所以二次元人物的头像它的分布,在高维的空间中,其实是非常狭窄的,所以二次元头像的分布,这个图片的分布,其实是高维空间中的低维的 Manifold

  或者是如果是以二维空间来想的话,那图片的分布,可能就是**二维空间的一条线**,二维空间中多数的点都不是图片,就高维空间中随便 Sample 一个点,都不是图片,只有非常小的范围,Sample 出来它会是图片,所以从这个角度来看,从资料本身的特性来看,PG 跟 Pdata,它们都是 Low Dimensional 的 Manifold,用二维空间来讲,PG 跟 Pdata 都是二维空间中的两条直线,而二维空间中的两条直线,除非它刚好重合,不然**它们相交的范围,几乎是可以忽略的**,这是第一个理由,那也许有人说, 这个 也许图片根本就不是什麼Low Dimensional 的 Manifold,那会不会第一个理由就不成立了

- 第二个理由是,我们是从来都不知道 PG 跟 Pdata 长什麼样子,我们对 PG 跟 Pdata,它的分布的理解,其实来自於 Sample



所以也许 PG 跟 Pdata,它们是有**非常大的 Overlap 的范围**,但是我们实际上,在了解这个 PG 跟 Pdata,在计算它们的 Divergence 的时候,我们是从 Pdata 裡面 Sample 一些点出来,从 PG 裡面 Sample 一些点出来

如果你 **Sample 的点不够多,你 Sample 的点不够密**,那就算是这两个 Divergence 实际上,这两个 Distribution 实际上有重叠,但是假设你 Sample 的点不够多,对 Discriminator 来说,它也是没有重叠的

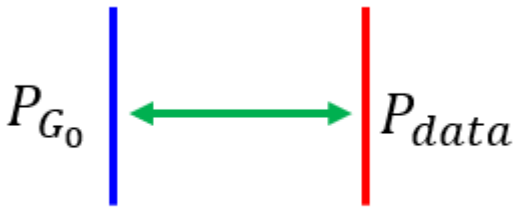
就是这个蓝色的分布跟红色的分布,明明是有重叠的,但如果你从蓝色分布 Sample 一些点,红色分布 Sample 一些点,这些点你又 **Sample 得不够多**,你完全就可以画一条楚河汉界,把红色的点跟蓝色的点完全地分开来,然后说红色的点,它的分布就是在这个楚河汉界的右边,蓝色的点就在左边,它们完全是**没有任何冲突**

所以以上给你两个理由,试图说服你说 PG 跟 Pdata 这两个分布,它们**重叠的范围是非常小的**

而几乎没有重叠这件事情,对於 JS Divergence 会造成什麼问题?

JS Divergence 有个特性,是**两个没有重叠的分布,JS Divergence 算出来,就永远都是 Log2**,不管这两个分布长什麼样,所以两个分布只要没有重叠,算出来就一定是 Log2

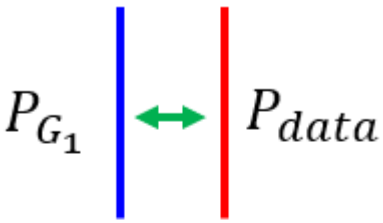
所以举例来说



$$JS(P_{G_0}, P_{data}) = log2$$

假设这是你的 Pdata,这是你的 PG,它们都,假设它们都是一条直线,然后中间有很长的距离,你算它们的 JS Divergence,是 Log2

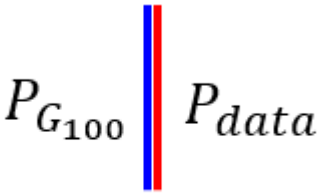
假设你的 PG 跟 Pdata 其实蛮接近的



$$JS(P_{G_1}, P_{data}) = log2$$

那中间的间隔其实是比较小的,算出来结果还是 Log2

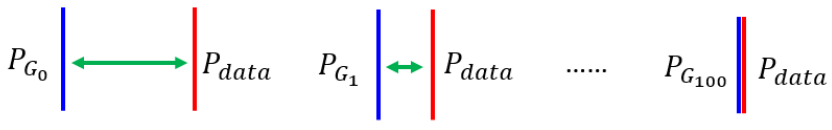
除非你的 PG 跟 Pdata 有重合



$$JS(P_{G_{100}}, P_{data}) = 0$$

不然这个 PG 跟 Pdata 只要它们是两条直线,它们这两条直线没有相交,那算出来就是 Log2,算出来这个 Case,算出来是 Log2,这个 Case 算出来也是 Log2

JS divergence is always log2 if two distributions do not overlap.



Equally bad

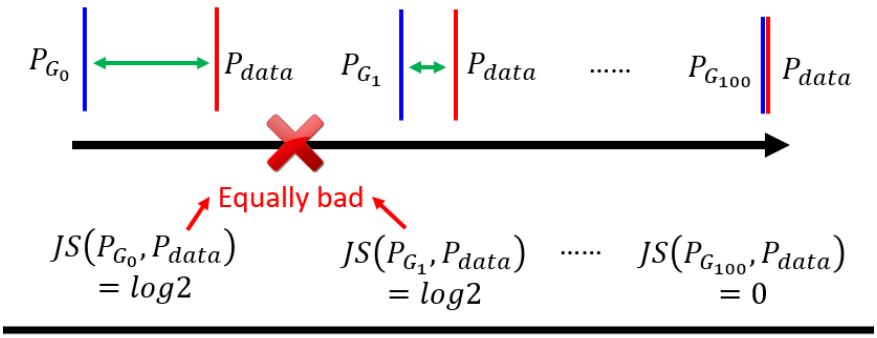
$JS(P_{G_0}, P_{data}) = log2$ ..... $JS(P_{G_1}, P_{data}) = log2$ ..... $JS(P_{G_{100}}, P_{data}) = 0$

那但是明明中间这个 Case,中间这个 Generator就比左边这个 Generator 好,明明蓝色的线就跟红色的线比较近,但是从 JS Divergence 上面,看不出这样子的现象

那既然从 JS Divergence 上,看不出这样的现象,你在 Training 的时候,你根本就没有办法把这样子的 Generator,Update 参数变成这样子的 Generator,因為对你的 Loss 来说,对你的目标来说,这两个 Generator 是一样好,或者是一样糟

那以上是从比较理论的方向来说明,如果我们从更直观的实际操作的角度来说明,你会发现,当你是**用 JS Divergence** 的时候,你就假设你今天在 Train 一个,Binary 的 Classifier,去分辨 Real 的 Image,跟 Generated Image,你会发现**实际上你通常 Train 完以后,正确率几乎都是 100%**

因為你 **Sample 的图片根本就没几张**,对你的 Discriminator 来说,你 Sample 256 张 Real 的图片,256 张 Fake 的图片,它直接用硬背的,都可以把这两组图片分开,知道说谁是 Real 谁是 Fake,所以实际上,如果你有自己 Train 过 GAN 的话你会发现,如果你用 Binary 的 Classifier Train 下去,你会发现,你几乎每次 Train 完你的 Classifier 以后,也就你 Train 完你的 Discriminator 以后,正确率都是 100%



Intuition: If two distributions do not overlap, binary classifier achieves 100% accuracy.

The accuracy (or loss) means nothing during GAN training.

我们本来会期待说,这个 Discriminator 的 Loss,也许代表了某些事情,这个 Binary Classifier Loss,也许代表某些事情,这个 **Loss 越来越大,代表问题越来越难**,代表我们的 **Generated Data,跟 Real 的 Data 越来越接近**

但实际上,你在实际操作的时候你**根本观察不到这个现象**,这个 Binary Classifier **训练完的 Loss根本没有什麼意义**,因為它总是可以让它的正确率变到 100%,两组 Image 都是 Sample 出来的,它硬背都可以得到 100% 的正确率,你根本就没有办法看出你的 Generator,有没有越来越好

所以过去,尤其是在还没有 WGAN 这样的技术,在我们还用 Binary Classifier,当作 Discriminator 的时候,Train GAN 真的就很像巫术 黑魔法,你根本就不知道你 Train 的时候,有没有越来越好,所以怎麼办,那时候做法就是,你每次 Update 几次 Generator 以后,你就要把你的图片 Print 出来看,然后你就要一边那个吃饭,一边看那个图片生成的结果,然后跑一跑就发现 哇 坏掉了,然后咔掉重做这样子

所以以前你根本就没有,不像我们在 Train 一般的 Network 的时候,你有个 Loss Function,然后那个 Loss 随著训练的过程,会慢慢慢慢变小,那你就会看说 Loss 慢慢变小,你就放心知道说你的 Network 有在 Train,但会不会 Overfitting 是另外一件事,至少还在它在 Training Data 上有越来越好

但是对 GAN 而言,本来我们期待 Classifier 的 Loss,可以提供一些资讯,但是当你的 Classifier 是一个,简单 一般的 Binary Classifier 的时候,它训练的结果你就没有任何资讯,你每次训练出来正确率都是 100%,你根本不知道你的 Generator,有没有越来越好,变成你只能够用人眼看,用人眼守在电脑前面看,发现结果不好 咔掉,重新用一组 Hyperparameter,重新调一下 Network,加工重做,所以过去训练 GAN 是有点辛苦的

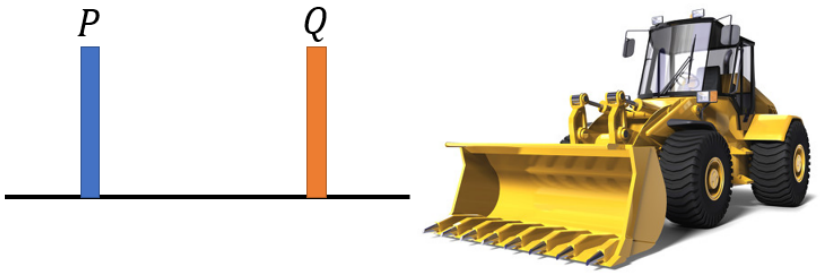
那既然是 JS Divergence 的问题,於是有人就想说,那会不会**换一个,衡量两个 Distribution 的相似程度的方式,换一种 Divergence,就可以解决这个问题了,於是就有了这个 Wasserstein**,或使用 ==Wasserstein Distance== 的想法

Wasserstein Distance

Wasserstein Distance 的想法是这个样子,假设你有两个 Distribution,一个 Distribution 我们叫它 P,另外一个 Distribution 我们叫它 Q

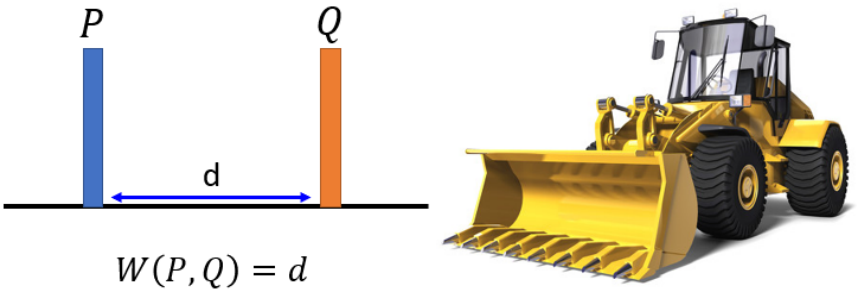
Wasserstein Distance 它计算的方法,就是想像你在开一台推土机

- Considering one distribution P as a pile of earth, and another distribution Q as the target
- The average distance the earth mover has to move the earth.



推土机的英文叫做 Earth Mover,想像你在开一台推土机,那你把 P 想成是一堆土,把 Q 想成是你要把土堆放的目的地,那这个推土机**把 P 这边的土,挪到 Q 所移动的平均距离**,就是 **Wasserstein Distance**

在这个例子裡面,我们假设 P 都集中在这个点,Q 都集中在这个点,对推土机而言,假设它要把 P 这边的土挪到 Q 这边,那它要平均走的距离,就是 D



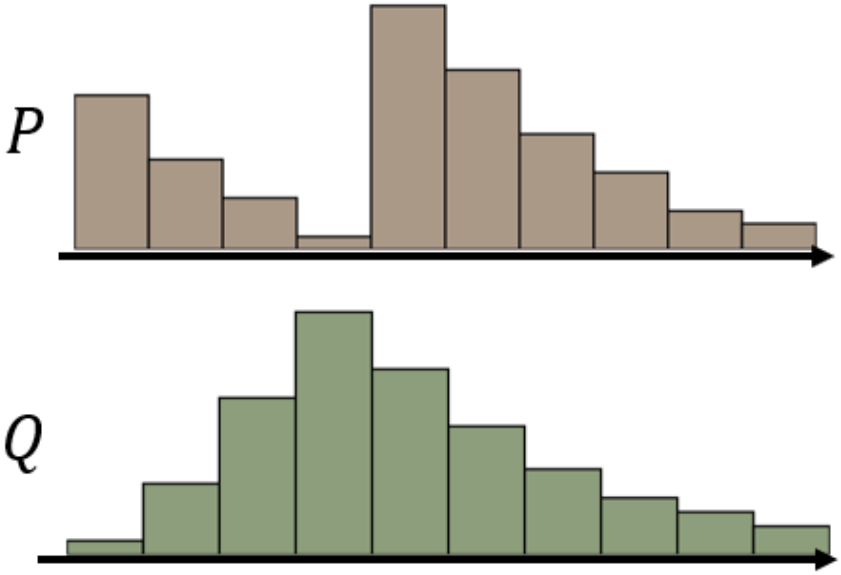
$$W(P,Q) = d$$

所以在这个例子裡面,假设 P 集中在一个点,Q 集中在一个点,这两个点之间的距离是 D 的话,那 P 跟 Q 的 Wasserstein Distance,就是 D

那因為在讲这个,Wasserstein Distance 的时候,你要想像有一个 Earth Mover,有一个推土机在推土,所以其实 Wasserstein Distance,又叫 ==Earth Mover Distance==

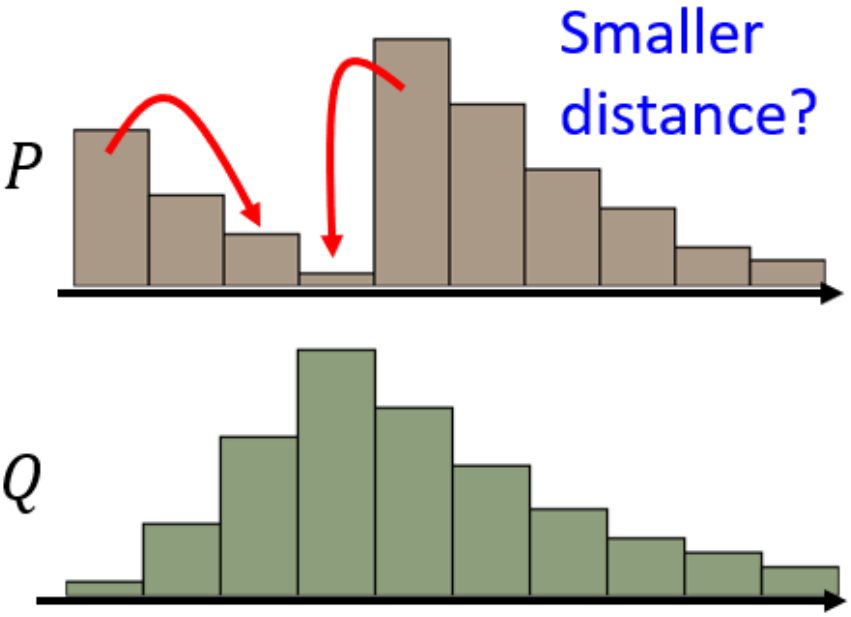
但是如果是**更复杂的 Distribution**,你要算 **Wasserstein Distance,就有点困难**了

假设这是你的 P,假设这是你的 Q

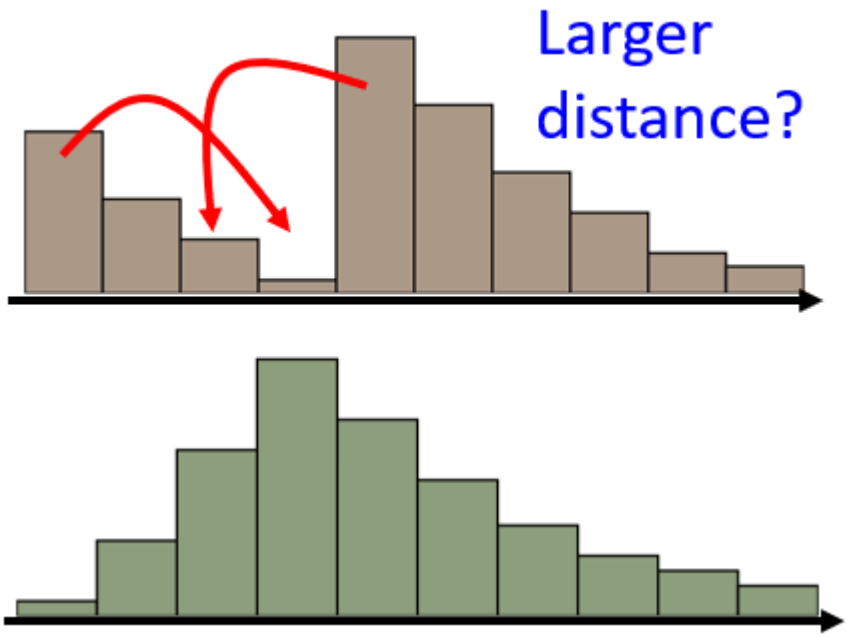


假设你开了一个推土机,想要把 P 把它重新塑造一下形状,让 P 的形状跟 Q 比较接近一点,那有什麼样的做法,你会发现说,你可能的 Moving Plans,**把 P重新塑造成 Q 的方法有无穷多种**

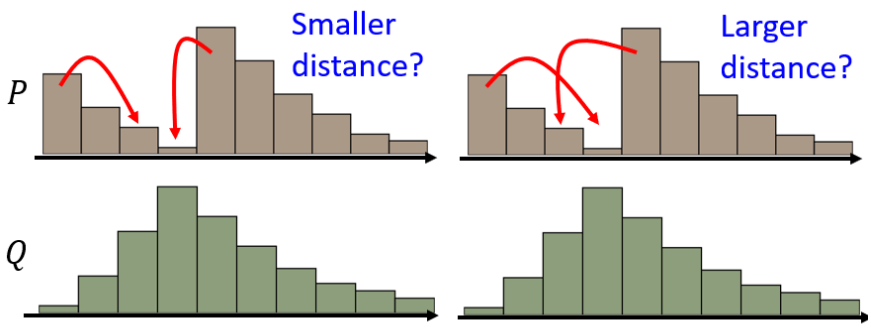
你可以说我把这边的土搬到这裡来,我把这边的土搬到这裡来,把 P 变成 Q



但你也可以捨近求远说,我把这裡的土搬到这裡来,把这裡的土搬到这裡来,捨近求远,一样还是可以把 P 变成 Q



所以当我们考虑,比较复杂的 Distribution 的时候,把 Q 把 P 变成 Q 的方法,是有非常非常多不同的方法,你有各式各样不同的 Moiving Plan,用**不同的 Moving Plan**,你**推土机平均走的距离就不一样**



There are many possible “moving plans”.

Using the “moving plan” with the smallest average distance to define the Wasserstein distance.

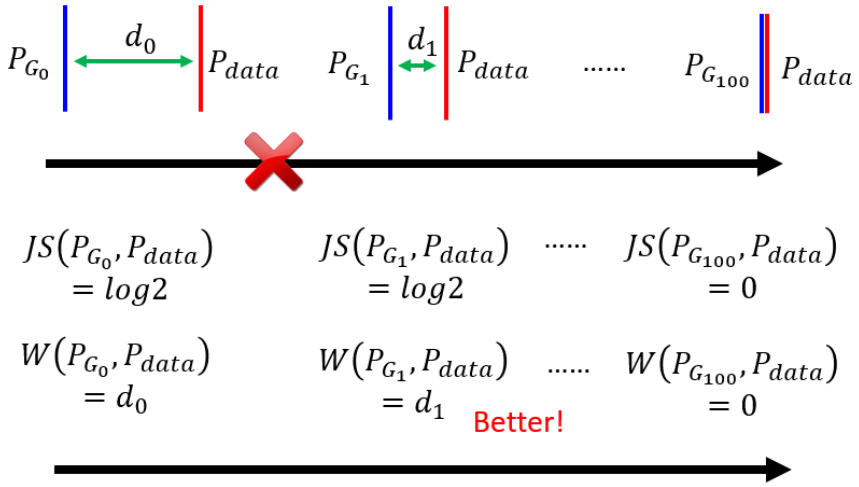
Source of image: https://vincentherrmann.github.io/blog/wasserstein/ 43

在左边这个例子裡面,推土机平均走的距离比较少,在右边这个例子裡面因為捨近求远,推土机平均走的距离比较大

為了让 Wasserstein Distance 只有一个值,所以这边 Wasserstein Distance 的定义是,**穷举所有的 Moving Plans**,然后看哪一个推土的方法,哪一个 Moving 的计划,可以让平均的距离最小,那个**最小的值,才是 Wasserstein Distance**

所以其实要计算 Wasserstein Distance,是挺麻烦的,光只是要计算一个 Distance,居然还要解一个 Optimization 的问题,**解出这个 Optimization 的问题,才能算 Wasserstein Distance**

我们先不讲,怎麼计算 Wasserstein Distance 这件事,我们先来讲假设我们**能够计算 Wasserstein Distance 的话**,它可以带给我们什麼样的**好处**



那假设 PG 跟 Pdata 它们的距离是 d_0,在左边这个例子裡面,Wasserstein Distance 算出来就是 d_0

在中间这个例子裡面,PG 跟 Pdata 它们之间的距离是 d_1,那 Wasserstein Distance 算出来的距离,就是 d_1

假设 d_1 比较小,d_0 比较大,那算 Wasserstein Distance 的时候,这个 Case 的 Wasserstein Distance 就比较小,这个 Case 的 Wasserstein Distance 就比较大,所以我们就可以知道说,**由左向右的时候,Wasserstein Distance 是越来越小的**

所以如果你观察,Wasserstein Distance 的话会知道说,从左到右我们的 Generator 越来越进步,但是如果你观察 Discriminator,你会发现你观察不到任何东西,对 Discriminator 而言,这边每一个 Case 算出来的 JS Divergence,都是一样,都是一样好或一样差,但是如果换成 Wasserstein Distance,由左向右的时候我们会知道说,我们的 Discriminator 做,我们的 Generator 做得越来越好

所以我们**换一个计算 Divergence 的方式**,我们就可以解决 JS Divergence,有可能带来的问题

这又让我想到一个演化的例子,这是眼睛的生成



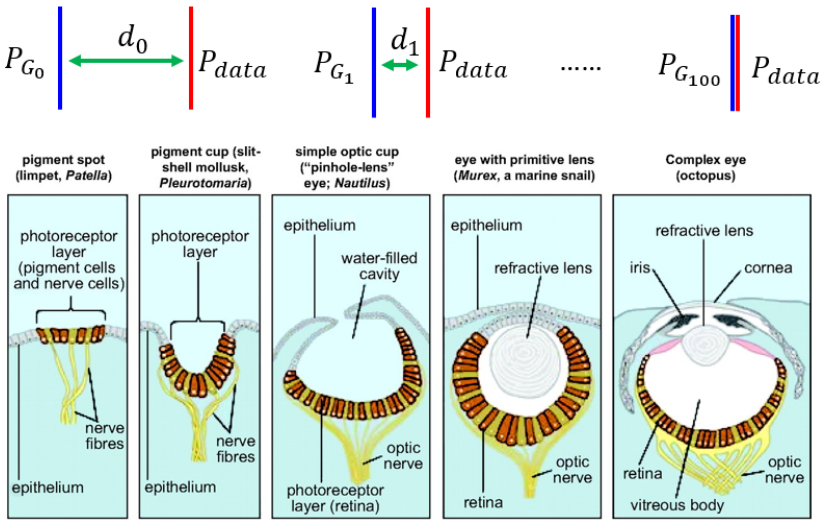
https://www.pnas.org/content/104/suppl_1/8567.figures-only

45

右边这个是人类的眼睛,人类的眼睛是非常地复杂的,那有一些生物它有非常原始的眼睛,比如说有一些细胞具备有感光的能力,这可以看做是最原始的眼睛,但是这些最原始的眼睛,怎麼变成最复杂的眼睛,这对人类来说其实觉得非常难想像

左边这个图像构造这麼简单,只是一些感光的细胞在皮肤上,经过突变產生一些感光的细胞,听起来像是一个合理的,但是天择突变,怎麼可能產生这麼复杂的器官,怎麼產生眼睛这麼精巧的器官

那如果你直接觉得说,从这个地方就可以一步跳到这个地方,那根本不可能发生,但是**中间其实是有很多连续的步骤**,从感光细胞到眼睛,中间其实是有连续的步骤的



https://www.pnas.org/content/104/suppl_1/8567.figures-only 45

举例来说,感光的细胞可能会,出现在一个比较凹陷的地方,皮肤凹陷下去,这样感光细胞可以接受来自不同方向的光源,然后后来觉得说,乾脆把凹陷的地方盖起来,后来觉得盖起来的地方裡面,可以放一些液体,然后最后就变成了人的眼睛

而从最左边跳到最右边这一步,非常大,但是这边每一小步,都可以让一个生命存活的机率变大,都可以让生命繁衍下去的机率变高

既然这边每一个步骤,都可以让生命繁衍的机率变高,那生命就可以从左边一直演化到右边,最终產生复杂的器官

而我觉得对於当你使用 WGAN,当你使用这个 Wasserstein Distance,来衡量你的 Divergence 的时候,其实就製造了类似的效果,本来 PG0 跟 Pdata 非常地遥远,你要它一步从这裡跑到这裡,一步从这裡跑到这裡,让这个 PG0 跟 Pdata,直接 Align 在一起,是不可能的,可能非常地困难

对 JS Divergence 而言,它需要做到直接从这一步跳到这一步,它的 JS Divergence 的 Loss 才会有差异,但是对 **W Distance** 而言,你只要**每次有稍微把 PG 往 Pdata 挪近一点**,W Distance 就会有变化,W Distance 有变化,你才有办法 Train 你的 Generator,去 Minimize W Distance,所以这就是為什麼,当我们从 JS Divergence,换成 Wasserstein Distance 的时候,可以带来的好处

WGAN

WGAN 实际上就是用,当你用 Wasserstein Distance,来取代 JS Divergence 的时候,这个 GAN 就叫做 WGAN

接下来你会遇到的问题就是,**Wasserstein Distance 是要怎麼算**,Pdata 跟 PG,它的 Wasserstein Distance 要怎麼计算,Wasserstein Distance,是一个非常复杂的东西,我光要算个 Wasserstein Distance,还要解一个 Optimization 的问题,实际上要怎麼

计算

那这边就**不讲过程,直接告诉结果**,解下面这个 Opimilazion 的 Problem,解出来以后你得到的值,就是 Wasserstein Distance

$$\max_{D\in 1-Lipschitz}\{E_{y\sim P_{data}}[D(y)]-E_{y\sim P_G}[D(y)]\}$$

我们就观察一下这个式子,这个式子裡面有说

- y 如果是从 Pdata 来的,那我们要计算它的 D(y) 的**期望值**
- y 如果是从 PG 来的,我们计算它的 D(y) 的**期望值**,但是**前面乘上一个负号**

所以如果你要 Maximum,这个 Objective Function,你会达成这样的效果,

- 如果 y是从 **Pdata** Sample 出来的,D(y),就 Discriminator 的 Output 要**越大越好**
- 如果 X 是从 **PG**,从 Generator Sample 出来的,那 D(y),也就 Discriminator 的 Output,应该要**越小越好**

$$\max_{D\in 1-Lipschitz}\{E_{y\sim P_{data}}[D(y)]-E_{y\sim P_G}[D(y)]\}$$

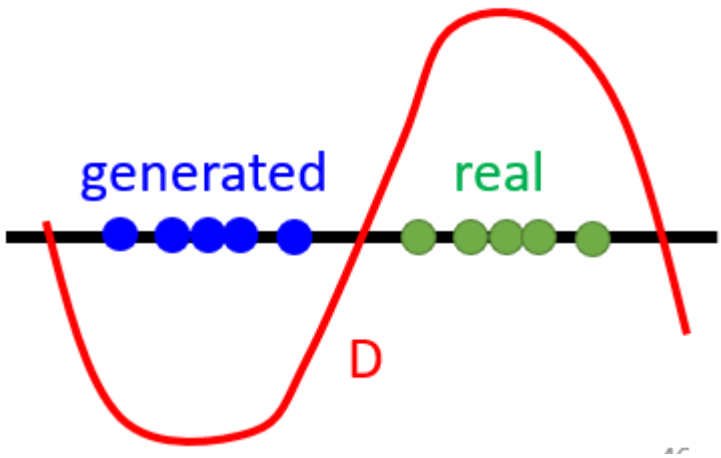
D has to be smooth enough.

但是这边**还有另外一个限制**,它不是光大括号裡面的值变大就好,还有一个限制是,**D 不能够是一个随便的 Function**,D必须要是一个 1-Lipschitz 的 Function

1-Lipschitz 是什麼东西,如果你不知道是什麼的话,也没有关係

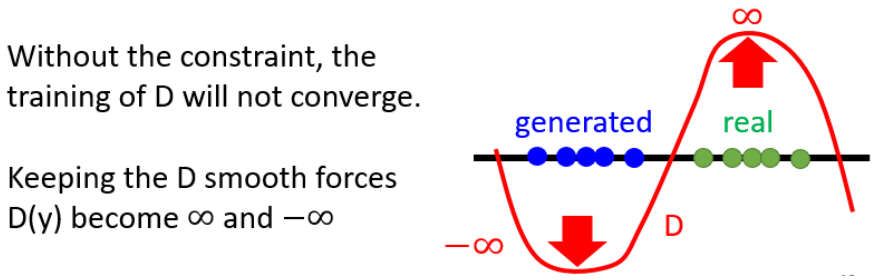
我们这边你就想成,**D 必须要是一个足够平滑的 Function**,它不可以是变动很剧烈的 Function,它必须要是一个足够平滑的 Function

那為什麼足够平滑这件事情是非常重要的,我们可以从直观来理解它,假设这个是真正的资料的分布,这是 Generated 的资料的分布



如果我们没有这个限制,只看大括号裡面的值的话,大括号裡面的目标,是要这些真正的值,它的 D(y) 越大越好,那要让 Generated 的值,它的 D(y) 越小越好

如果你没有做任何限制,只单纯要这边的值越大越好,这边的值越小越好,在蓝色的点跟绿色的点,也就是真正的 Image,跟 Generated 的 Image,没有任何重叠的情况下,你的 Discriminator 会做什麼,它会给 Real 的 Image **无限大的正值**,给 Generated 的 Image **无限大的负值**



所以你这个 Training 根本就没有办法收敛,而且你会发现说,只要这两堆 Data 没有重叠,**你算出来的值都是无限大**,你算出来的这个 Maximum 值都是无限大,这显然不是我们要的,这不就跟 JS Divergence 的问题一模一样吗

其实你要加上这个限制以后,你这个 Maximum 的值,才会是 Wasserstein Distance,才是做 WGAN

為什麼加上这个限制,就可以解决刚才的问题?

因為这个限制是要求 **Discriminator,不可以太变化剧烈**,它必须要够听话,那今天如果你,要求你的 Discriminator 够平滑的时候,假设 Real Data 跟 Generated Data,它们距离比较近,那你就没有办法让 Real 的 Data 值非常大,然后 Generated 的值非常小,因為如果你让 Real 的值非常大,Generated 的值非常小,它们中间的差距很大,那这一个 Discriminator 变化就很剧烈,它就不平滑了,它就不是 1-Lipschitz

那為了要是 1-Lipschitz,这边的值没办法很大,这边的值没办法很小,让 Discrimination 必须足够,只觉得发现说,如果 Real 跟generated 的 Data 差距离很远,那它们的值就可以差很多,这边算出来的值就会比较大,你的 Wasserstein Distance 就比较大

如果 Real 跟 Generated 很近,就算它们没有重合,**因為有了这一个限制**,有了这一个写在 Max 下面的,1-Lipschitz Function 的限制,那你就发现说,它们其实**不会真的都跑到无限大**,它们的距离,因為有 1-Lipschitz Function 的限制,所以 Real Data 的值跟 Generated Data 的值,就**没有办法差很多**,那你算出来的 Wasserstein Distance,就会比较小

接下来的问题就是,**怎麼做到这个限制**

怎麼确保 Discriminator,一定符合 1-Lipschitz Function 的限制,最早刚提出 WGAN 的时候,其实没有什麼好想法,只知道写出了这个式子,那要怎麼真的解这个式子,有点困难,所以最早的一篇 WGAN 的 Paper,最早使用 Wasserstein 的那一篇 Paper,它说了它做了一个比较 Rough,比较粗糙的处理的方法

$$\max_{D \in 1-Lipschitz} \{E_{y \sim P_{data}}[D(y)] - E_{y \sim P_G}[D(y)]\}$$

- Original WGAN → Weight

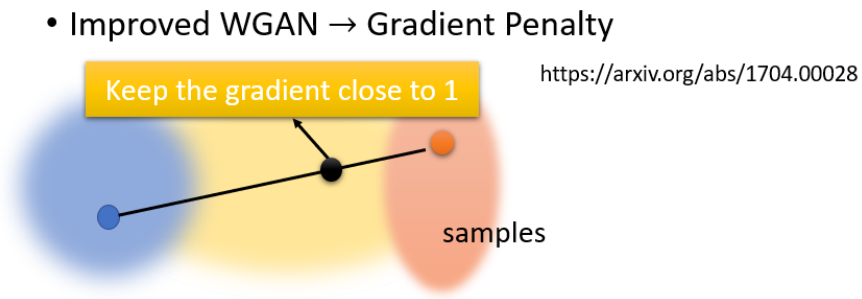
  Force the parameters w between c and -c

  After parameter update, if w > c, w = c; if w < -c, w = -c

它是说我就 Train Network,那 Train Network 的时候,如果我 Training 的那个**参数**,我就要求它放得**在 C 跟 -C** 之间,如果超过 C,用 Gradient Descent Update 以后**超过 C,就设为 C**,Gradient Descent Update 以后**小於 -C,就直接设为 -C**

其实这个方法,**并不一定真的能够让 Discriminator,变成 1-Lipschitz Function**,那你可以想像说这个方法,也许真的可以让我们的 Discriminator,比较平滑,但它并没有真的去解这个,Optimization 的 Problem,它并没有真的让 Discriminator,符合这个限制

所以接下来就有其它的想法,有一个想法叫做 Gradient Penalty



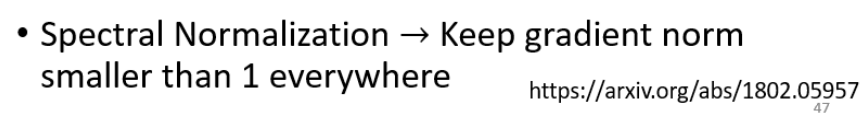
Gradient Penalty 是出自,Improved WGAN 这篇 Paperhttps://arxiv.org/abs/1704.00028,那 Improve WGAN 这边paper 是说

假设这个是你的 Real Data 的分布,这个是你的这个 Fake Data 的分布,那我在 Real Data 这边取一个 Sample,Fake Data 这边取一个 Sample,两点连线中间再取一个 Sample,我要求这个点它的 Gradient 要接近,那你可能觉得说这个,不知道 不知道在说些什麼这样,这件事情的关係,跟这个限制的关係,到底是什麼,那就详见 Improved WGAN 的 Paper

那其实我觉得你现在,也不一定要真的非常较真说,这件事情 Gradient Penalty,跟这个限制之间的关係,因為其实有更多其它的方法,真的可以去解这个问题

那其实后来 Improved WGAN 之后,还有什麼 Improved 的 Improved WGAN,真的有这篇 Paper,它就是把这个限制再稍微改一改

有另外一篇 Paper 就叫,Improved The Improved WGAN,那今天其实你有一个方法,真的把 D 限制,让它是 1-Lipschitz Function,这个叫做 Spectral Normalization



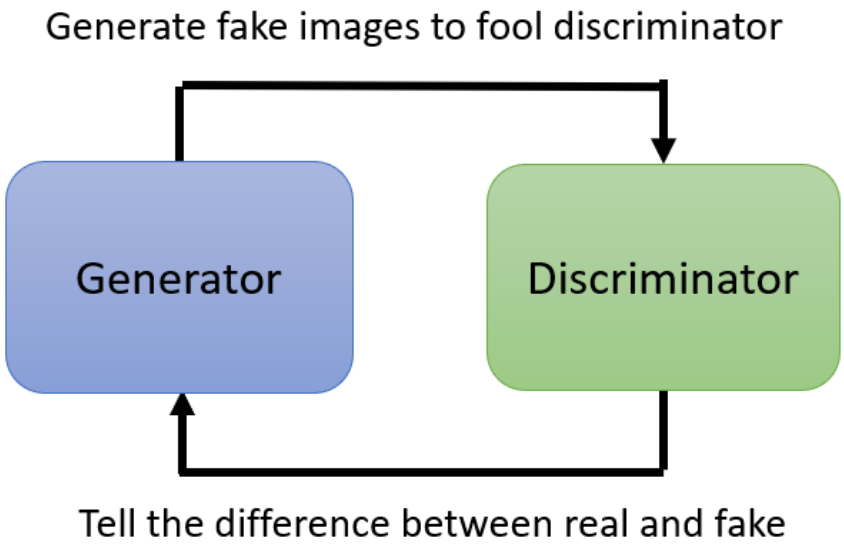
那我就把它的论文https://arxiv.org/abs/1802.05957放在这边给大家参考,那如果你要 Train 真的非常好的 GAN,你可能会需要用到 Spectral Normalizaion

GAN_P3

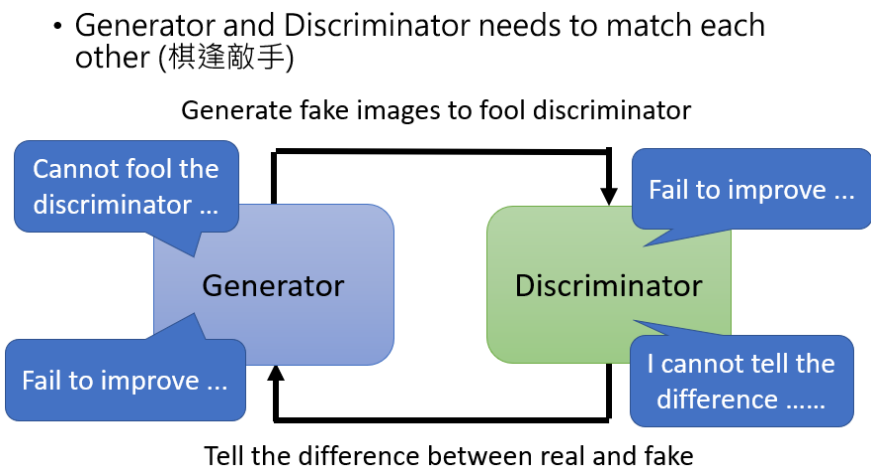
虽然说已经有 WGAN,但其实并不代表说,GAN 就一定特别好 Train,GAN 仍然是以,很难把它 Train 起来而闻名的,那為什麼 GAN 很难被 Train 起来?

它有一个本质上困难的地方

- Discriminator 做的事情,是要分辨真的图片跟產生出来的,也就是假的图片的差异
- 而 Generator 在做的事情,它是要去產生假的图片,骗过 Discriminator



而事实上这两个 Network,这个 Generator 跟 Discriminator,它们是**互相砥砺,才能互相成长的**,只要**其中一者,发生什麼问题停止训练,另外一者就会跟著停下训练**,就会跟著变差



- 假设你在 Train Discriminator 的时候,一下子没有 Train 好
- 你的 Discriminator 没有办法分辨,真的跟產生出来的图片的差异
- 那 Generator,它就失去了可以进步的目标,Generator 就没有办法再进步了

- 如果 Generator 没有办法再进步,它没有办法再產生更真实的图片,那 Discriminator 就没有办法再跟著进步了

但是到目前為止,大家已经 Train 过了很多次的 Network,**我们没有办法保证 Train 下去,它的 Loss 就一定会下降**，你要让 Network Train 起来,往往你需要**调一下 Hyperparameter**,才有可能把它 Train 起来

那今天这个 Discriminator 跟 Generator,它们互动的过程是自动的,因為我们不会在中间,每一次 Train Discriminator 的时候,你都换 Hyperparameter

所以只能祈祷每次 Train Discriminator 的时候,它的 Loss 都是有下降的,那**如果有一次没有下降,那整个 Training,很有可能就会变就会惨掉**,整个 Discriminator 跟 Generator,彼此砥礪的这个过程,就可能会停下来

所以今天 Generator 跟 Discriminator,在 Train 的时候,它们必须要棋逢敌手,就任何一个人放弃了这一场比赛,另外一个人也就玩不下去了

因此 GAN 本质上它的 Training,仍然不是一件容易的事情,当然它是一个非常重要的技术,所以虽然它是一个前瞻的技术

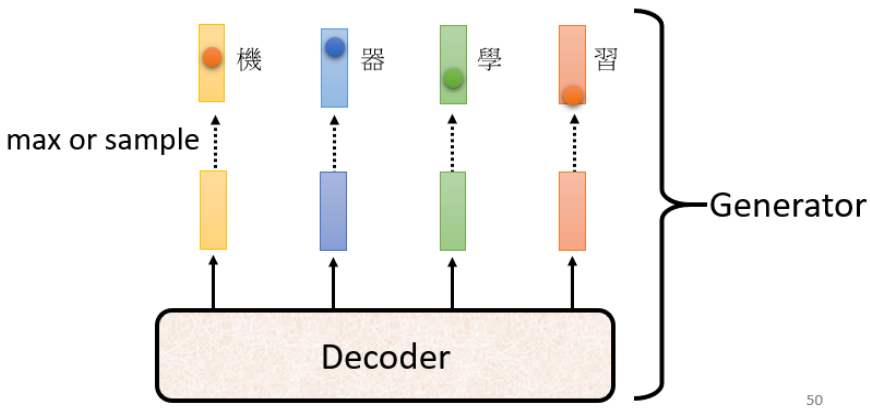
一些相关的,跟 Train GAN 的诀窍有关的文献,还有链接列在这边,其实就给大家自己参考

- Tips from Soumith
- Tips in DCGAN: Guideline for network architecture design for image generation
- Improved techniques for training GANs
- Tips from BigGAN

GAN for Sequence Generation

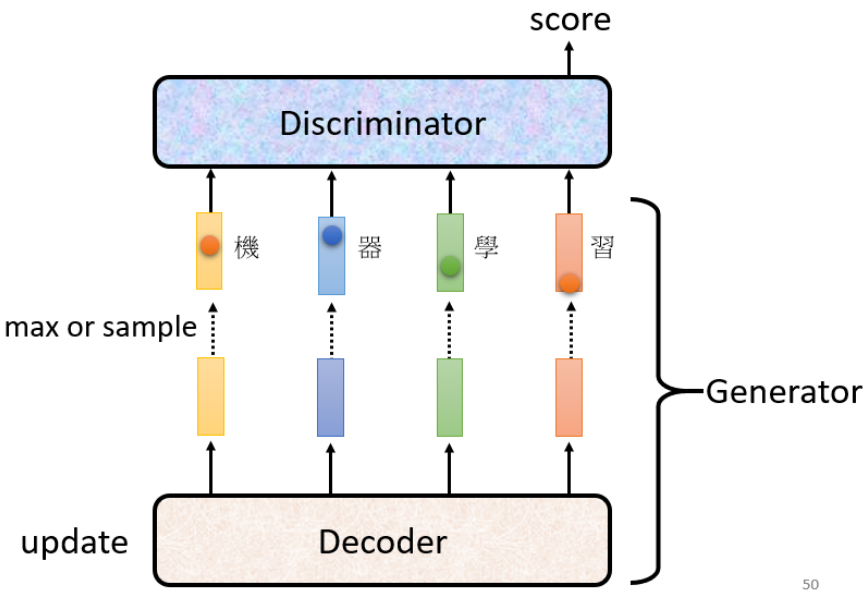
Train GAN 最难的其实是要拿 GAN 来生成文字

如果你要生成一段文字,那你可能会有一个,Sequence To Sequence 的 Model,你有一个 Decoder



那这个 Decoder 会產生一段文字,那我们现在这个,**Sequence To Sequence 的 Model就是我们的 Generator**,这个在过去,在讲 Transformer 的时候,这是一个 Decoder,那它现在,在 GAN 裡面,它就扮演了 Generator 的角色,负责產生我们要它產生的东西,比如说一 段文字

那你说这个会跟原来的 GAN,在影像上的 GAN 有什麼不同? 就最 High Level 来看,**就演算法来看**,可能没**有太大的不同**,因為接下来,你就是训练一个 Discriminator



Discriminator 把这段文字读进去,去判断说这段文字是真正的文字,还是机器產生出来的文字,而 Decoder 就是想办法去骗过 Discriminator,Generator 就是想办法去骗过 Discriminator

你去**调整**你的这个 **Generator** 的**参数**,想办法让 Discriminator 觉得,Generator產生出来的东西是真正的

但是真正的的难点在於,你如果要用 **Gradient Descent,去 Train 你的 Decoder**,去让 Discriminator Output 分数越大越好,你会发现你**做不到**

大家知道,在用Gradient Descent方法中计算微分的时候,所谓的 Gradient,所谓的**微分**,其实就是**某一个参数**,它有**变化的时候**,**对你的目标造成了多大的影响**

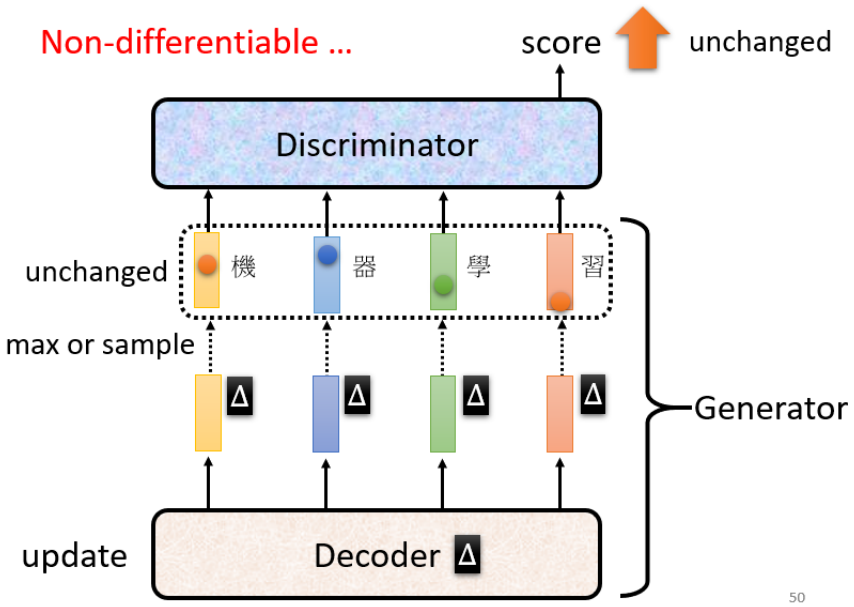
我们现在来想想看,假设我们改变了 Decoder 的参数,也就是 Generator 的参数,有一点小小的变化的时候,到底对 Discriminator 的输出,有什麼样的影响

如果**Decoder 的参数**有一点**小小的变化**，那它现在**输出的这个 Distribution**,也会有**小小的变化**,那因為这个变化很小,所以它**不会影响最大**的那一个 **Token**

==Token==,可能会觉得有点抽象,那如果你要想得更具体一点,Token 就是你现在在处理这个问题,处理產生这个 Sequence 的单位,那 Token ,是人定的了

- 假设我们今天,在產生一个中文的句子的时候,我们是每次產生一个 Character,一个方块字,那方块字就是我们的 Token
- 那假设你在处理英文的时候,你每次產生一个英文的字母,那字母就是你的 Token
- 假设你一次,是產生一个英文的词,英文的词和词之间,是以空白分开的,那就是词就是你的 Token

所以 Token 的定义,是你自己决定的,看你要拿什麼样的东西,当做你產生一个句子的单位

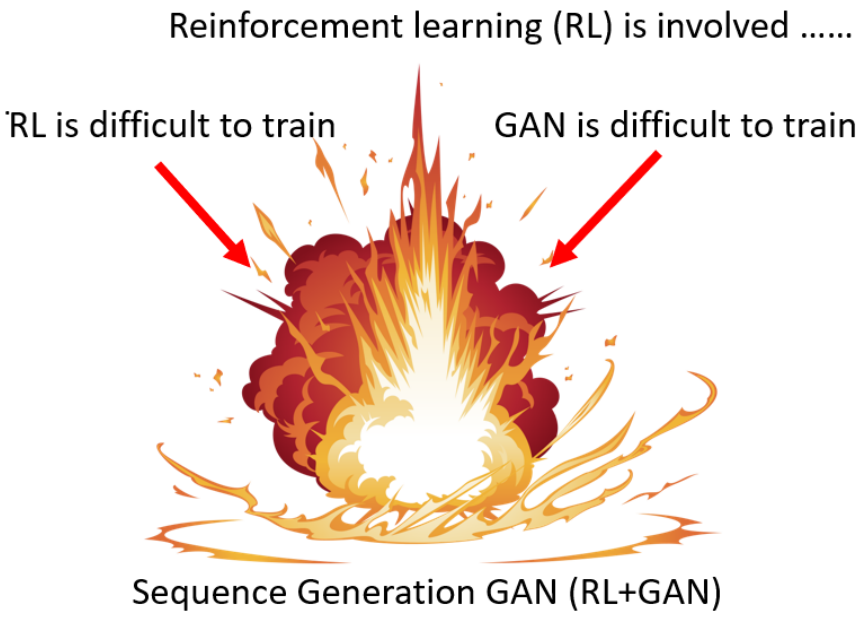


那今天,这个 **Distribution 只有小小的变化**,在取 Max 的时候,在找分数最大那个 Token 的时候,你会发现,分数最大的那个 Token是没有改变的,你 Distribution 只有小小的变化,所以分数最大的那个 Token 是同一个,那对 Discriminator 来说,它**输出的分数是一模一样的**,这样输出的分数就没有改变

所以你根本就没有办法算微分,你根本就**没有办法做 Gradient Descent**

有同学可能会说,欸 这个 这边不是 Max,是因為 Max 造成不能做 Gradient Descent 吗,那那个 CNN 裡面不是有那个 Max Pooling 吗,那怎麼还可以做 Gradient Descent? 这个问题就留给你自己深思一下,為什麼在这个地方,有 Max 不能做 Gradient Descent,而在CNN有 Max Pooling,却可以做 Gradient Descent

但是就算是不能做 Gradient Descent,你也不用害怕,记不记得我们上週有讲说,遇到不能用 Gradient Descent Train 的问题,就当做 ==Reinforcement Learning== 的问题,硬做一下就结束了,所以你确实可以用 Reinforcement Learning,来 Train 你的 Generator,在你要產生一个 Sequence 的时候,你可以用 Reinforcement Learning,来 Train 你的 Generator

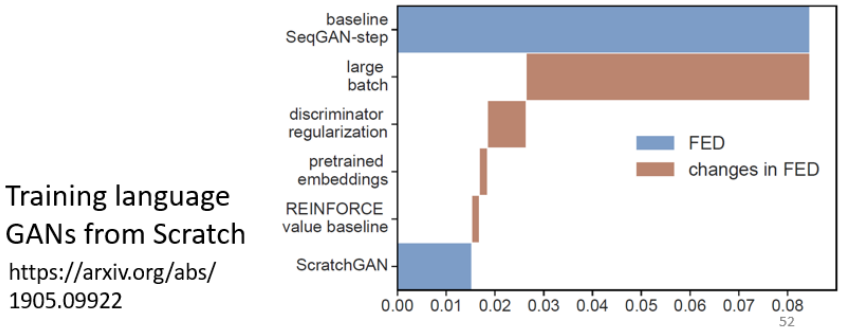


但Reinforcement Learning 是以难 Train 而闻名,GAN 也是以难 Train 而闻名,这样的东西加在一起,就大炸裂这样 Train 不起来,非常非常地难训练

所以要用 GAN 產生一段文字,过去一直被认為,是一个非常大的难题,所以有很长一段时间,没有人可以成功地把 Generator 训练起来產生文字,通常你需要先做 ==Pretrain==,那 Pretrain 这件事情,其实我们等一下马上就会提到,如果你现在还不知道 Pretrain 是什麼的话,也没有关係,总之过去你没有办法用正常的方法,让 GAN 產生一段文字

直到有一篇 Paper 叫做 ScrachGAN

- Usually, the generator are fine-tuned from a model learned by other approaches.
- However, with enough hyperparameter-tuning and tips, ScarchGAN can train from scratch.



它的 Title 就开宗明义跟你炫耀说,它可以 Train Language GANs Form Scrach,Form Scrach 就是**不用 Pretrain** 的意思

它可以直接**从随机的初始化参数开始**Train 它的 Generator,然后让 Generator 可以產生文字,它最**关键的就是爆调 Hyperparameter,跟一大堆的 Tips**

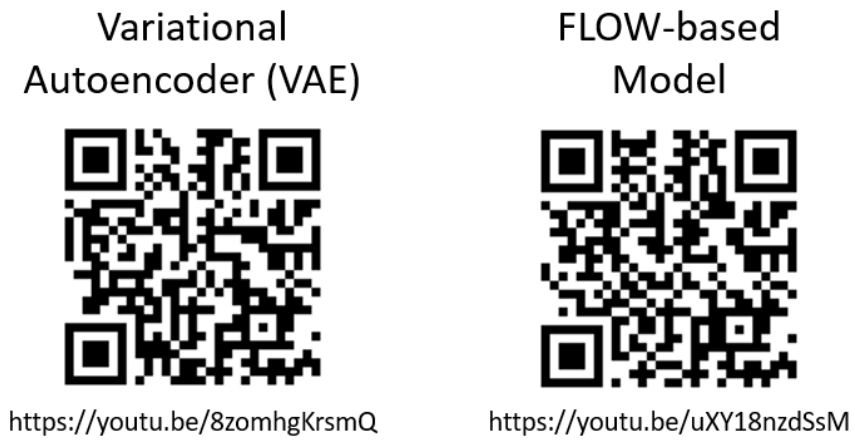
比如说,这个横轴是它们的 Major,这个叫做 **FED**,那这个是用在文字上的,我们今天就不讲,这不重要,总之这个值**越低越好**

- 一开始要有一个叫做 SeqGAN-Step 的技术,没这个完全 Train 不起来
- 然后接下来有一个很大的 Batch Size,通常就是上千,没有那个,你自己在家没办法这麼做的
- Discriminator 加 Regularization,Embedding 要 Pretrain
- 改一下 Reinforcement Learning 的 Argument
- 最后就有 ScratchGAN,就可以从真的把 GAN Train起来,然后让它来產生 Sequence

那今天有关 GAN 的部分,我们只是讲了一个大概,那如果你想要学最完整的内容,我在这边留下一个连结给大家参考,



那其实有关 Generative 的 Model,不是只有 GAN 而已，还有其他的,比如说 VAE,比如说 FLOW-Based Model,那我在这边也列了两个影片的连结,给大家参考



强调一下就是,这边的影片连结并**不是一定要看过这些影片连结,才能够学习接下来的内容**,因為机器学习可以讲的东西实在太多了,所以如果,假设你没有太多的时间,那你唯一真正需要听的,上课讲的内容是 Self Content,它本身是 Consistent 的,你只要每一堂课都有听,你接下来的内容,你应该都可以依序听下去,应该都可以听懂

然后在上课中,会放一些影片的连结,这个就等於是额外分出去的分支,如果你真的很有兴趣的话,可以进行深入的研究

那為什麼我们不再讲更多东西,因為在上课的设计,**这个课程的内容,是以真正能够对你有帮助,以实务為导向的**,就假设你想要 Train 一个 Generator,你想让机器可以產生东西,你有很多方法,你可以用 GAN,你可以 VAE,可以用 FLOW-Bases Model

我们这边就选择告诉你 GAN,所以以后,你如果有人叫你,Train 一个 Generative Model,你有办法去 Train 它,那你如果想要深入研究,你可以在研究 VAE 跟 FLOW-Bases Model,那有人可能会问说,為什麼选择 GAN,為什麼不是选择其他的 Model 一个最直接的理由是,**GAN 的 Performance 是比较好**的

如果你要產生非常好的图片的话,你还是今天要用 GAN,通常 VAE 或 FLOW-Bases Model,它们產生的结果,都是跟 GAN 有非常大的一段差距,它们通常都是 Plan 说,我经过了一番努力,暴调了一堆参,爆弄了一堆 Tips,最后可以跟 GAN 差不多而已,所以 GAN 通常,它產生出来的结果还是比较好的

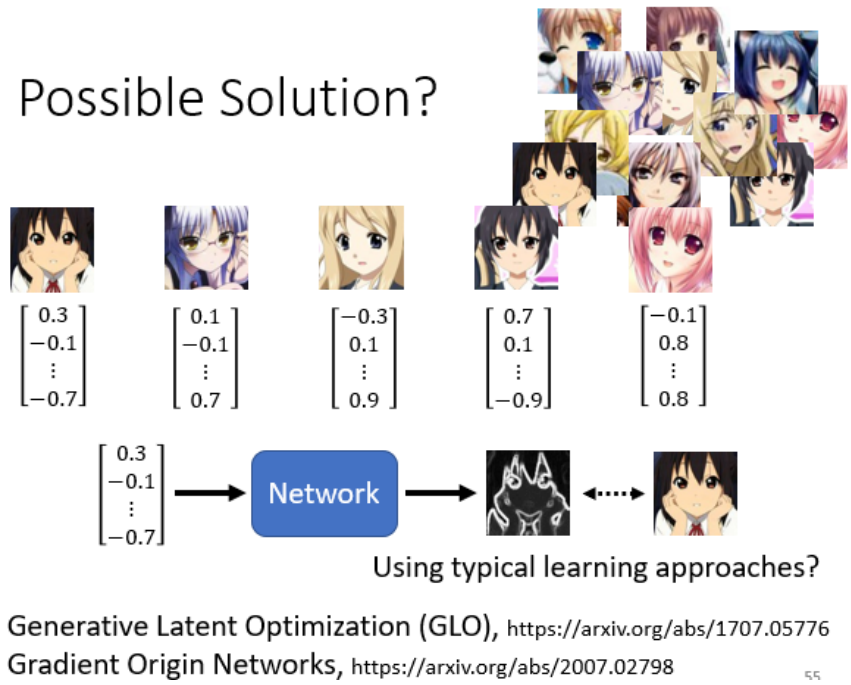
那你可能会说,GAN 比较难 Train,这个比较 Train 吧,VAE 或 FLOW 会不会比较好 Train?

如果你真的有实做 **VAE 或 FLOW 的话,它们没有比较好 Train** 老实说,你可能会觉得说,从它的式子上看起来,GAN 很神祕,有一个 Discriminator Generator,它们要互动,然后像 FLOW-Bases Model VAE,它们都比较像是,直接 Train 一个一般的模型,它们有一个很明确的 Objective

但你实际上 Train 起来发现说,它们也没有那麼容易成功地被训练起来,它们的 Objective 裡面有很多项,它们的 Loss 裡面有很多项,然后把每一项都平衡,才能够有好的结果,但要达成平衡也非常地困难,跟 GAN,我觉得 Train 的难度是不遑多让,所以我们这边就选择 GAN ,作為我们课堂上介绍的,生成式的 Generative 的 Model,那至於其他 Model,你可以再多多,如果你有兴趣,你可以再自己涉猎

也许有同学会想说,為什麼我们要特别用一些,提出一些新的做法,来做 Generative 这件事?

如果我们今天的目标就是,输入一个 Gaussian 的 Random 的 Variable,输入一个 Gaussian,从 Gaussian 的这个 Random Variable,Sample 出来的 Vector,把它变成一张图片,那我们能不能够用,==Supervised Learning== 的方法来做?



也就说我有一堆图片,我把这些图片拿出来,每一个图片都去配一个 Vector,都去配一个,从 Gaussian Distribution,Sample 出来的 Vector，接下来就当做 Supervised Learning 的方法,硬做就结束了,

这张图片就是对到这个 Vector,Train 一个 **Network**,**输入一个 Vector**,**输出**就是它**对应的图片**,把对应的图片当做你训练的目标,训练下去

真的有这样子的生成式的模型,那难的点是说,如果这边,纯粹放随机的向量,Train 起来结果会很差,你可能根本连 Train 都 Train 不起来,所以怎麼办,你需要有一些**特殊的方法**,我在这边一样放两篇论文的连结

Generative Latent Optimization (GLO), https://arxiv.org/abs/1707.05776

Gradient Origin Networks, https://arxiv.org/abs/2007.02798

Evaluation of Generation

我们现在產生出来的 Generator,它好或者是不好,那要**评估一个 Generator 的好坏,并没有那麼容易**,那最直觉的做法,也许是**找人来看**,你要知道,今天这个 Generator 產生出来的图片,到底像不像动画的人物,那就找人直接来看,也许就结束了

其实很长一段时间,尤其是人们刚开始研究,Generative 这样的技术的时候,很长一段时间没有好的 measure,那时候要评估 Generator 的好坏,都是人眼看,然后**直接用吹的**这样

就说在 Paper 最后就放几张图说,你看这个,我觉得应该是比文献上,目前结果都还要好,太棒了,这应该是state of the art,然后就结束了这样子,所以发现比较早年的 GAN 的 Paper,它没有数字,整篇 Paper 裡面没有 Accuracy,它就是放几张图片告诉你说,这个应该是比过去的文章都好,然后就结束了

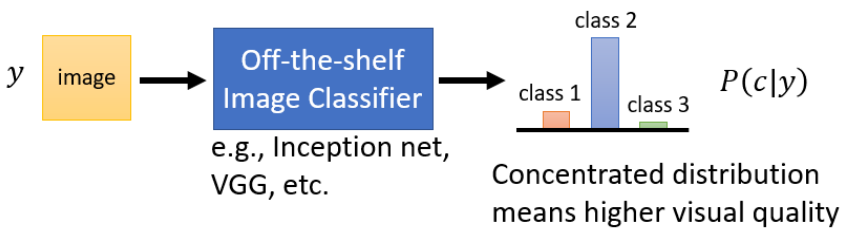
Quality of Image

- Human evaluation is expensive (and sometimes unfair/unstable).
- How to evaluate the quality of the generated images automatically?

完全**用人来看**显然有**很多的问题**,比如说**不客观**,**不稳定**等等诸多的问题,所以有没有比较客观,而且自动的方法,来想办法量一个 Generator 的好坏,如果针对特定的一些任务,是有办法设计一些方法的,

如果是**一般的 Case** ,如果我们不侷限在我们的作业,跟一般的 Case,我随便训练了一个 Generator,它不一定是產生动画人物的,因為它產生别的,它专门產生猫,专门產生狗,专门產生斑马等等

那有一个方法,是一样跑一个影像的分类系统,**把你的 GAN 產生出来的图片,丢到一个的影像的分类系统裡面**,看它產生什麼样的结果



影像分类系统输入是一张图片,我们这边叫做 y,输出,是一个机率分布,我们这边叫它 P (c | y),P (c | y) 是一个机率的分布

然后接下来我们就看说,这个机率的**分布如果越集中**,就代表说现在產生的**图片可能越好**,虽然我们不知道这边產生的图片,裡面有什麼东西,不知道它是猫还是狗还是斑马,我们不知道它是什麼,但是如果丢到了一个影像分类系统以后,它输出来的结果,它输出来的这个分布非常集中,代表影像分类系统,它非常肯定,它现在看到什麼样的东西

它非常肯定它看到了狗,它非常肯定它看到了斑马,然后代表说,你產生出来的图片,也许是比较接近真实的图片,所以影像辨识系统才辨识得出来

如果你**產生出来的图片是一个四不像**,根本看不出是什麼动物,那影像辨识系统就会非常地困惑,它產生出来的这个**机率分布**,就会非常地**平坦**,非常地**平均分布**,那如果是平均分布的话,那就代表说你的 GAN,產生出来的图片,可能是比较奇怪的,所以影像辨识系统才会辨识不出来

所以这个是靠影像辨识系统,来判断你產生出来的图片好坏,这是一个可能的做法

Diversity - Mode Collapse

但是光用这个评估的方法会被一个,叫做 ==Mode Collapse== 的问题骗过去,Mode Collapse 是说,你在 Train GAN 的时候,你有时候 Train 著 Train 著,就会遇到一个状况是



- 假设这些**蓝色**的星星,是**真正**的资料的分布
- **红色**的星星是你的 **GAN**,你的 Generative 的 Model,它的分布

你会发现说 Generative Model,它**输出来的图片来来去去,就是那几张**,可能单一张拿出来,你觉得好像还做得不错,但让它多產生几张就露出马脚

那以下是一个 Mode Collapse 的例子啦,就是我们在这个上週有看到说,我就 Train 了一个 Generator,让它產生二次元的人物,那 Train 著 Train 著 Train 到最后,我就发现变成这样的一个状况,这一张脸越来越多



越来越多,而且它还有不同的髮色,这个髮色比较偏红,这个髮色比较偏黄,越来越多,最后就通通都是这张脸,那这就是一种 Mode Collapse 的现象

那為什麼会有 Mode Collapse,这种现象发生,就直觉上你还是比较容易理解,你可以想成说,这个地方就是 Discriminator 的一个盲点,当 Generator 学会產生这种图片以后,它 Discr,它就永远都可以骗过 Discriminator,Discriminator 没办法看出说,这样子的图片是假的,那这是一个 Discriminator 的盲点,Generator 抓到这个盲点就硬打一发,就发生 Mode Collapse 的状况

那可是到底要**怎麼避免Mode Collapse** 的状况,我认為今天**其实还没有一个非常好的解答**,举例来说,我们在上週给大家看到了,BGAN 的结果,就是会產生网球狗那个结果,那是 Google 做的,它也爆收了参数,但就算是它爆收了参数,它发现最终,它仍然没有办法真的避免,Mode Collapse 的状况,就 BGAN Train 到最后,还是 Mode Collapse

BGAN 那边 Paper 怎麼解决这个问题,其实很简单,Model 在 Generator 在训练的时候,一路上都会把checkpoint 存下来,**在 Mode Collapse 之前,把 Training 停下来**,然后就把之前的 Model 拿出来用,就结束了这样,所以就算是强如 Google 爆收参数,现在**还是没有办法彻底解决,Mode Collapse 的问题**

Diversity - Mode Dropping

但是有另外一种更难被侦测到的问题,叫做 Mode Dropping,Mode Dropping 的意思是说,你的真实的资料分布可能是这个样子,但是你的產生出来的资料,只有真实资料的一部分,单纯看產生出来的资料,你可能会觉得还不错,而且分布,它的这个多样性也够



但你不知道说真实的资料,它的**多样性的分布,其实是更大**的,我这边举一个例子,好那这边,是一个真实的例子,就有个同学,他 Train 了这个人脸生成的 GAN,那它在某一个 Iteration 的时候,它的 Generator 產生出这些人脸



你会觉得说,没有问题,而且人脸的多样性也够,有男有女,有向左看,有向右看,各式各样的人脸都有,好 这个是第 T 个 Iteration 的时候 Generator,你也不觉得,它的多样性有问题,但如果你再看下一个 Iteration,Generator 產生出来的图片是这样子的



(BEGAN on CelebA)

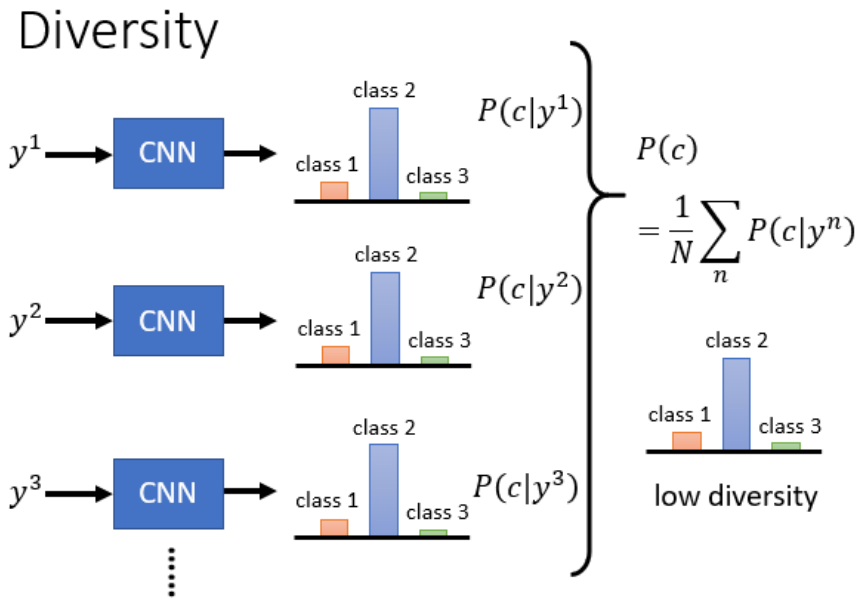
它的**肤色有问题**,所以它之前,你看有男有女没有问题,但是它肤色偏白,这边肤色偏黄,你没弄好人家都觉得,你的 Generator 有种族歧视

所以在这种 Mode Dropping 的问题是,**不太容易被侦测出来的**,事实上今天到底,今天这些非常好的 GAN,BGAN,Progress GAN,BGAN,Progress GAN,可以產生非常真实人脸这些 GAN,到底有没有 Mode Dropping 的问题,可能还是有的

如果你看多了,GAN 產生出来的人脸,你会发现说,虽然非常真实,但好像来来去去,就是那麼几张脸而已,它有一个非常独特的特徵是,你看多了以后就觉得,这个脸好像是被生成出来的,所以今天也许 **Mode Dropping 的问题,都还没有获得本质上的解决**

但是我们会需要去量说,现在我们的 Generator,它產生出来的图片,到底多样性够不够

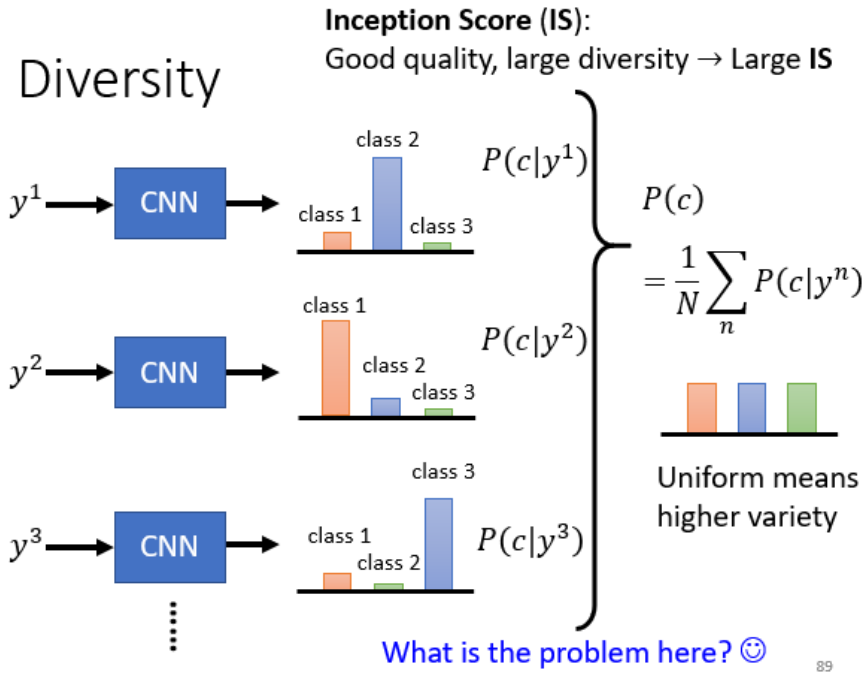
过去有一个做法,一样是**藉助我们的 Image Classifier**,你就把一堆图片,就很像你的 Generator 產生 1000 张图片,把这 1000 张图片裡,都丢到 Image Classify 裡面,**看它被判断成哪一个 Class**



每张图片,都会给我们一个 Distribution,你**把所有的 Distribution 平均起来**,接下来看看平均的 Distribution 长什麼样子

如果平均的 **Distribution 非常集中**,就代表现在**多样性不够**,如果什麼图片丢进去,你的影像分类系统都说,是看到 Class 2,看到裡面有 Class 2 这样的东西,那代表说,每一张图片也许都蛮像的,你的多样性是不够的

那如果另外一个 Case,不同张图片丢进去,不同张,你的 Generator 產生出来的图片,丢到 Image Classifier 的时候,它產生出来的**输出的分布**,都**非常地不同**



你平均完以后发现,**平均完**后的结果是**非常平坦**的,那这个时候代表什麼,这个时候代表说,也许你的**多样性是足够的**,那你会发现说在评估的标準上

当我们用这个 Image 的 Classifier,来做评估的时候,**Diversity 跟 Quality 好像是有点互斥**的,因為我们刚才在讲 Quality 的时候,我们说**越集中代表 Quality 越高**,但是 **Diversity 分布越平均,代表 Diversity 越大**

强调一下这个 Quality 跟 Diversity,它们评估的**范围不一样**

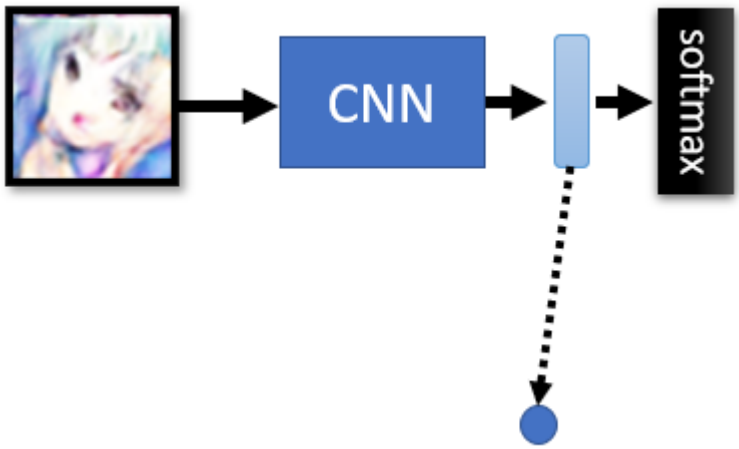
- Quality 是只看**一张图片**,一张图片丢到 Classifier 的时候,分布有没有非常地集中
- 而 Diversity 看的是**一堆图片**,它分布的平均,一堆图片你的 Image Classifier ,如果输出的平均越平均的话,就代表说现在的 Diversity 越大

那过去有一个非常常被使用的分数,叫做 ==Inception Score==,那它的缩写 是 IS,所谓 Inception Score,顾名思义就是这边用的这个 基于CNN的Inception模型来做的,所以叫 Inception Score

用 Inception Network 量一下 Quality,如果 Quality 高,那个 Diversity 又大,那 Inception Score 就会比较大

Fréchet Inception Distance (FID)

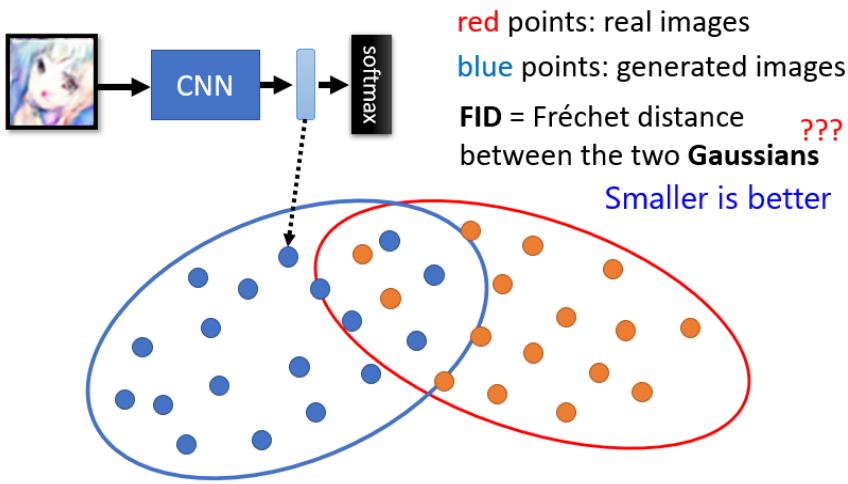
在我们的作业中,会採取另外一个 Evaluation 的 Measure,叫 ==Fréchet Inception Distance==,它的缩写叫做 FID,这个东西是什麼,你先把你產生出来的二次元的人物,丢到 Inception Net 裡面



把这个二次元人物一路丢到最后,让那个 Inception Network 输出它的类别,那你得到的可能就是人脸,那每一张二次元的人物看起来都是人脸,那我们**不要拿那个类别**

我们拿**进入 Softmax 之前的 Hidden Layer 的输出**,进入 Softmax 之前,你的 Network 会產生一个向量,那可能是长度是上千维的一个向量,把那个向量拿出来,代表这张图片

那如果我们拿出来的是一个向量,而不是最后的类别,那虽然最后分类的类别可能是一样的,但是在决定最后的类别之前,这个向量就算都是人脸,可能还是不一样的,可能会随著肤色 髮型,这个向量还是会有所改变的,所以我们就不取最后的类别,只取这个 Inception Network 中间的,其实是最后一层的这个 Hidden Layer 的输出,来代表一张图片



所有**红色的点**,代表你把**真正的图片**,丢到 Inception Network 以后,拿出来的向量,那这个向量其实非常高维度,是上千维的， 我们就把它假设,我们可以把它画在二维的平面上,

蓝色的点是你自己的 GAN,你自己的 Generator,**產生出来的图片**,它丢到 Inception Network 以后,进入 Softmax 之前的向量,把它画出来,假设是长这个样子

接下来,假设真实的图片跟產生出来的图片它们都是 Gaussians 的 Distribution,然后去**计算这两个 Gaussians Distribution 之间的==Fréchet Distance==**,就结束了

那至於 Fréchet 的 Distance 是什麼,你有兴趣再自己看一下文献,反正在作业裡面,我们的 Judge Systerm 会帮大家算好

因為它是一个 Distance,所以这个值就是越小越好,**距离越小,代表这两组图片越接近**,那当然就是產生出来的品质越高

但这边你一定心裡还是有很多问号

- 第一个问号就是,当做 Gaussians Distribution 没问题吗,这个应该不是 Gaussians Distribution吧?

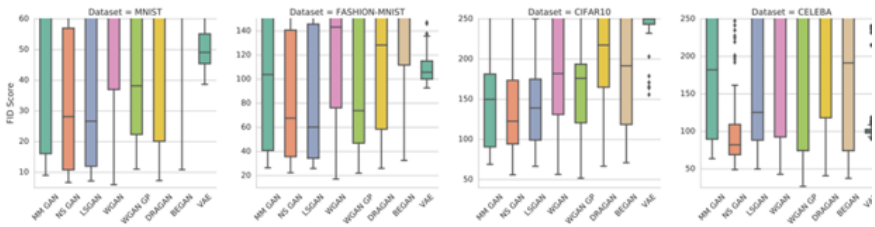
  会有问题!

- 然后另外一个问题就是,如果你要準确的得到你的 Network 它的分布,那你可能需要產生大量的 Sample 才能做到,那这需要一点运算量,那这个也是要做 FID 不可避免的问题

  所以其实我们在作业裡面,我们不会只看,我们也不会只看 FID,只看 FID,其实结果会怪怪的,怪怪的 因為你,你假设你的这个输出的分布一定是 Gaussians ,那它实际上不是 Gaussians,硬假设它是 Gaussians,没有怪怪的吗,会怪怪的,所以我们是同时看 FID,跟动画人物人脸的这个,侦测出来的人脸的数目,这两个指标,我会同时看这两个指标,那这样可以得到比较合理而精确的结果

FID 算是今天比较常用的一种 Measure,那有一篇 Paper 叫做,Are GANs Created Equal,A Large Scale Study

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] + \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = -\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}}$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}}$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} - = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}}$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = -\mathcal{L}_{\mathrm{D}}^{\mathrm{NS\ GAN}}$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$



FID: Smaller is better

Are GANs Created Equal? A Large-Scale Study
https://arxiv.org/abs/1711.10337

那你可以想见说这个也是 Google 做的啦,那就是爆做了各式各样不同的 GAN,有,那个时候它就列举了好多不同的,各式各样的 GAN,那每一个 GAN,当然它的这个训练的这个 Objective,训练的那个 Loss 有点不太一样,我这边就不细讲,各式各样的 GAN,每一种 GAN,它都用不同的 Random Seed,去跑过很多次以后,看看结果怎麼样

上面这个图,就是在四个不同的资料库上面得到的结果

横轴这边代表的是不同的 GAN,那这边的值FID,是越小越好

你会发现说,这边每一个方法,它都不是只得到一个数值,它都**得到一个分布**,為什麼它得到是一个分布,因為你要用那个**不同的 Random Seed** 去跑,每次跑出来的结果都不太一样,那这边混了一个不是 GAN 的做法,混了一个 VAE 在这裡

那你会发现说,如果比较这些 GAN 的方法跟 VAE 的方法,**VAE 的方法显然是比较稳定的**,不同的 Random Seed,看起来差距还是比较小的,那 GAN 的方法,不同 Random Seed 差距是很大的,那你又可以很明显地看出,VAE 跟 GAN,它的这个好的程度,不在同一个量级上,**GAN可以產生远比 VAE 更好的结果**

不过你会发现说**不同的 GAN好像结果差不多**,所以这边就,那抬头就是 Are GANs Created Equal,然后看起来所有的 GAN 都差不多,

如果你仔细看那篇文章的话,在做实验的时候,所有**这些不同的 GAN 用的 Network 架构都是同一个**,它只是爆收了那个,Random Seed 跟 Learning Rate 而已,所以 Network 架构还是同一个

所以我们**不知道是不是有某些 Network 架构,特别 Favor 某些种类的 GAN**,或者是某些种类的 GAN,会不会在不同的 Network 架构上,表现得比较比较稳定,比如说如果你看 WGAN 的话,WGAN 最原始的 Paper,它标榜的其实是它 Network 架构胡乱设计,它胡乱兜个什麼 100 层的 Generator,就很没有必要弄一个 100 层的 Generator,它也 Train 得起来,所以也许 WGAN 是在不同的 Generator,不同的 Network 架构的时候比较稳定,那你试不同的 Random Seed,可能没有特别稳定等等之类的,不知道,这篇 Paper 并没有给我们这方面的答案

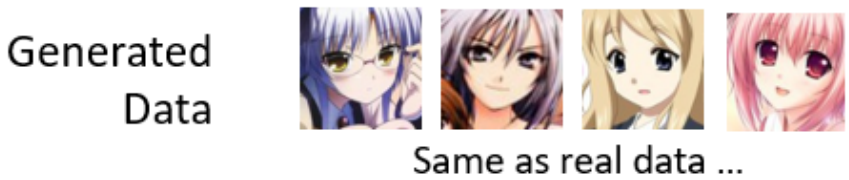
We don’t want memory GAN.

那其实刚才那些 Measure 也完全,也并没有完全解决 GAN 的Evaluation 的问题

你想想看以下的状况,假设这是你的真实资料



你不知道怎麼回事,训练了一个 Generator,它產生出来的 Data,跟你的真实资料一模一样



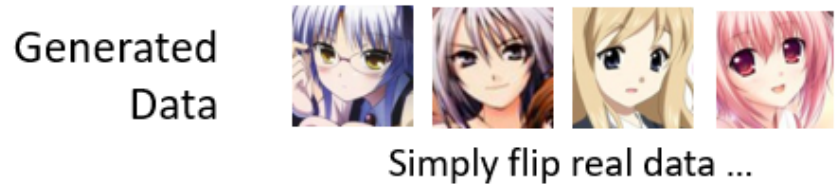
所以如果你不知道真实资料长什麼样子,你光看这个 Generator 的输出,你会觉得太棒了,它做得很棒,那 FID 算出来,一定是非常小的

但问题是这个是你要的吗,如果它產生出来的图片都跟资料库裡面的,训练资料的一模一样,训练资料就在你手上,直接从训练资料裡面,Sample 一些 Image 出来不是更好,干嘛要 Train Generator

我们 **Train Generator 其实是希望它產生,新的图片,训练资料裡面没有的人脸**,如果训练资料裡面有一模一样的人脸,直接用训练资料裡面的人脸就好了,何必用 GAN ,所以有时候你的 GAN 產生出来的结果很好,也许你在作业裡面,FID 算出来也很低,然后人脸辨识系统也给你很高的分数,但是它不一定是一个好的 GAN

你可能会说,那我们就把,我们 **Generator 產生出来的图片**,跟**真实资料**比个**相似度**吧,看看是不是一样嘛,如果很多张都一样就代表说,Generator 只是把那个训练资料背起来而已,它没有很厉害

但是那如果我问另外一个问题,假设你的 Generator 学到的是,把所有训练资料裡面的图片都**左右反转**,那它也是什麼事都没有做



假设它学到就是,把训练资料裡面所有的图片都左右翻转,那你会觉得,嗯 它看起来很棒,它实际上也是什麼事都没有做,但问题是你比相似度的时候,又比不出来,所以 **GAN 的 Evaluation是非常地困难的**,还甚至 光要如何评估,一个 Generator 做得好不好这件事情,都是一个可以研究的题目

如果你真的很有兴趣的话,这边放了一篇相关的文章啦
https://arxiv.org/abs/1802.03446,裡面就列举了二十几种,GAN Generator 的评估的方式



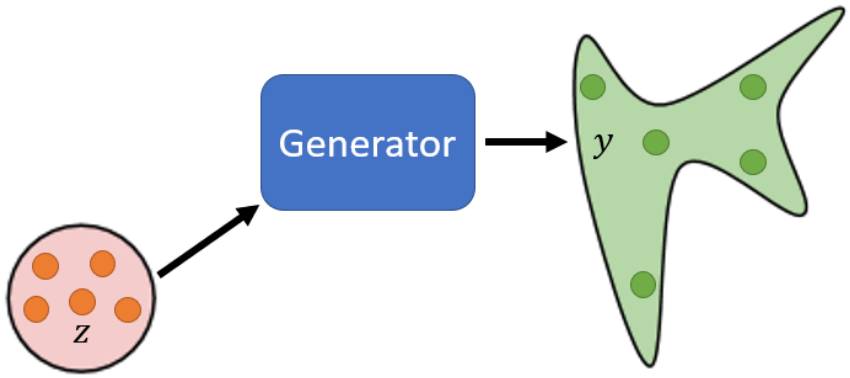
	Measure	Description
Quantitative	1. Average Log-likelihood [18, 22]	• Log likelihood of explaining realworld held out/test data using a density estimated from the generated data (*e.g.* using KDE or Parzen window estimation). $L = \frac{1}{N}\sum_i \log P_{model}(\mathbf{x}_i)$
	2. Coverage Metric [33]	• The probability mass of the true data "covered" by the model distribution $C := P_{data}(dP_{model} > t)$ with t such that $P_{model}(dP_{model} > t) = 0.95$
	3. Inception Score (IS) [3]	• KLD between conditional and marginal label distributions over generated data. $\exp(\mathbb{E}_{\mathbf{x}}[\mathrm{KL}(p(y \mid \mathbf{x}) \parallel p(y))])$
	4. Modified Inception Score (m-IS) [34]	• Encourages diversity within images sampled from a particular category. $\exp(\mathbb{E}_{\mathbf{x}_i}[\mathbb{E}_{\mathbf{x}_j}[\mathrm{KL}(P(y\|\mathbf{x}_i)\|P(y\|\mathbf{x}_j))]])$
	5. Mode Score (MS) [35]	• Similar to IS but also takes into account the prior distribution of the labels over real data. $\exp(\mathbb{E}_{\mathbf{x}}[\mathrm{KL}(p(y \mid \mathbf{x}) \parallel p(y^{train}))] - \mathrm{KL}(p(y) \parallel p(y^{train})))$
	6. AM Score [36]	• Takes into account the KLD between distributions of training labels vs. predicted labels, as well as the entropy of predictions. $\mathrm{KL}(p(y^{train}) \parallel p(y)) + \mathbb{E}_{\mathbf{x}}[H(y\|\mathbf{x})]$
	7. Fréchet Inception Distance (FID) [37]	• Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space $FID(r,g) = \|\mu_r - \mu_g\|_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}})$
	8. Maximum Mean Discrepancy (MMD) [38]	• Measures the dissimilarity between two probability distributions P_r and P_g using samples drawn independently from each distribution. $M_k(P_r, P_g) = \mathbb{E}_{\mathbf{x},\mathbf{x}'\sim P_r}[k(\mathbf{x},\mathbf{x}')] - 2\mathbb{E}_{\mathbf{x}\sim P_r,\mathbf{y}\sim P_g}[k(\mathbf{x},\mathbf{y})] + \mathbb{E}_{\mathbf{y},\mathbf{y}'\sim P_g}[k(\mathbf{y},\mathbf{y}')]$
	9. The Wasserstein Critic [39]	• The critic (*e.g.* an NN) is trained to produce high values at real samples and low values at generated samples $\hat{W}(\mathbf{x}_{test}, \mathbf{x}_g) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_{test}[i]) - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_g[i])$
	10. Birthday Paradox Test [27]	• Measures the support size of a discrete (continuous) distribution by counting the duplicates (near duplicates)
	11. Classifier Two Sample Test (C2ST) [40]	• Answers whether two samples are drawn from the same distribution (*e.g.* by training a binary classifier)
	12. Classification Performance [1, 15]	• An indirect technique for evaluating the quality of unsupervised representations (*e.g.* feature extraction; FCN score). See also the GAN Quality Index (GQI) [41].
	13. Boundary Distortion [42]	• Measures diversity of generated samples and covariate shift using classification methods.
	14. Number of Statistically-Different Bins (NDB) [43]	• Given two sets of samples from the same distribution, the number of samples that fall into a given bin should be the same up to sampling noise
	15. Image Retrieval Performance [44]	• Measures the distributions of distances to the nearest neighbors of some query images (*i.e.* diversity)
	16. Generative Adversarial Metric (GAM) [31]	• Compares two GANs by having them engaged in a battle against each other by swapping discriminators or generators. $p(\mathbf{x}\|y=1; M_1)/p(\mathbf{x}\|y=1; M_2) = (p(y=1|\mathbf{x}; D_1)p(\mathbf{x}; G_2))/(p(y=1|\mathbf{x}; D_2)p(\mathbf{x}; G_1))$
	17. Tournament Win Rate and Skill Rating [45]	• Implements a tournament in which a player is either a discriminator that attempts to distinguish between real and fake data or a generator that attempts to fool the discriminators into accepting fake data as real.
	18. Normalized Relative Discriminative Score (NRDS) [32]	• Compares n GANs based on the idea that if the generated samples are closer to real ones, more epochs would be needed to distinguish them from real samples.
	19. Adversarial Accuracy and Divergence [46]	• Adversarial Accuracy. Computes the classification accuracies achieved by the two classifiers, one trained on real data and another on generated data, on a labeled validation set to approximate $P_g(y\|\mathbf{x})$ and $P_r(y\|\mathbf{x})$. Adversarial Divergence: Computes $\mathrm{KL}(P_g(y\|\mathbf{x}), P_r(y\|\mathbf{x}))$
	20. Geometry Score [47]	• Compares geometrical properties of the underlying data manifold between real and generated data.
	21. Reconstruction Error [48]	• Measures the reconstruction error (*e.g.* L_2 norm) between a test image and its closest generated image by optimizing for z (*i.e.* $min_{\mathbf{z}}\|G(\mathbf{z}) - \mathbf{x}^{(test)}\|^2$)
	22. Image Quality Measures [49, 50, 51]	• Evaluates the quality of generated images using measures such as SSIM, PSNR, and sharpness difference
	23. Low-level Image Statistics [52, 53]	• Evaluates how similar low-level statistics of generated images are to those of natural scenes in terms of mean power spectrum, distribution of random filter responses, contrast distribution, etc.
	24. Precision, Recall and F_1 score [23]	• These measures are used to quantify the degree of overfitting in GANs, often over toy datasets.
Qualitative	1. Nearest Neighbors	• To detect overfitting, generated samples are shown next to their nearest neighbors in the training set
	2. Rapid Scene Categorization [18]	• In these experiments, participants are asked to distinguish generated samples from real images in a short presentation time (*e.g.* 100 ms); *i.e.* real v.s fake
	3. Preference Judgment [54, 55, 56, 57]	• Participants are asked to rank models in terms of the fidelity of their generated images (*e.g.* pairs, triples)
	4. Mode Drop and Collapse [58, 59]	• Over datasets with known modes (*e.g.* a GMM or a labeled dataset), modes are computed as by measuring the distances of generated data to mode centers
	5. Network Internals [1, 60, 61, 62, 63, 64]	• Regards exploring and illustrating the internal representation and dynamics of models (*e.g.* space continuity) as well as visualizing learned features

Pros and cons of GAN evaluation measures
https://arxiv.org/abs/1802.03446

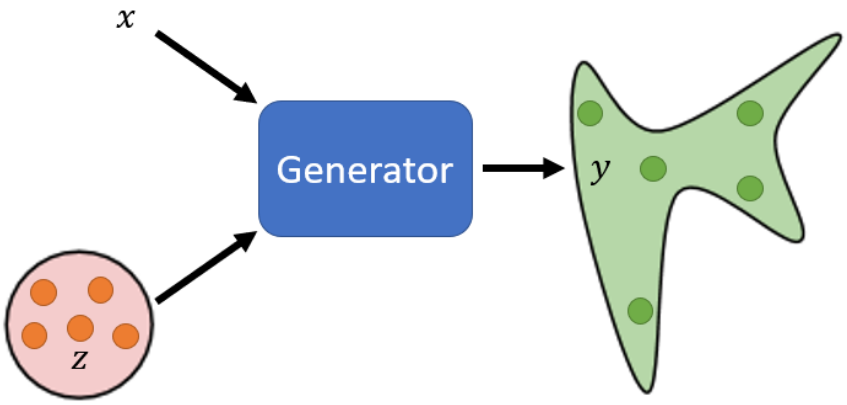
Conditional Generation

什麼是 Conditional 的 Generation?

刚才我们讲的那个 Generator,到目前為止我们讲的 Generator,它输入都是一个随机的分布而已,那这个不见得非常有用

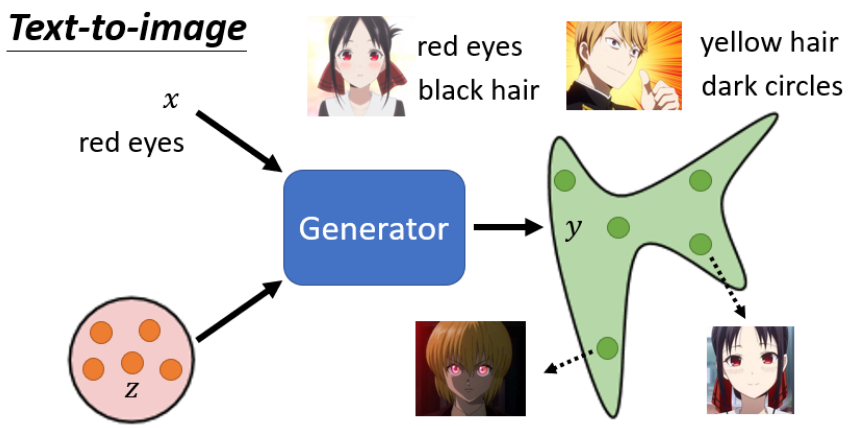


我们现在想要更进一步的是,我们可以操控 Generator 的输出,我们给它一个 Condition x,让它根据 x 跟 z 来產生 y,那这样的 Conditional Generation



有什麼样的应用,比如说你可以做文字对图片的生成

那如果你要做文字对图片的生成,它其实是一个 Supervised Learning 的问题,你需要一些 Label 的 Data,你需要去蒐集一些图片,蒐集一些人脸,然后这些人脸都要有文字的描述,告诉我们说,这个是红眼睛,这个是黑头髮,这个是黄头髮,这个是有黑眼圈等等,告诉我们这样子,我们要这样的 Label 的资料,才能够训练这种 Conditional 的 Generation



所以在 Text To Image 这样的任务裡面,我们的 **x 就是一段文字**,那你可能问说,一段文字怎麼输入给 Generator ,那就要问你自己了,你要怎麼做都可以

以前会用 RNN 把它读过去,然后得到一个向量,再丢到 Generator,今**天也许你可以把它丢到一个 Transformer 的 Encoder 裡面**去,把 Encoder Output 这些向量通通平均起来,丢到 Generator 裡面去,怎麼样都可以 反正,你用什麼方法都可以,只要能够让 Generator 读一段文字就行

那你期待说你输入 Red Eyes,然后,机器就可以画一个红眼睛的角色,但每次画出来的角色都不一样,那这个画出来什麼样的角色,取决於什麼,取决於你 Sample 到什麼样的 z,Sample 到不一样的 z,画出来的角色就不同,但是通通都是红眼睛的,这个就是 Text To Image 想要做的事情



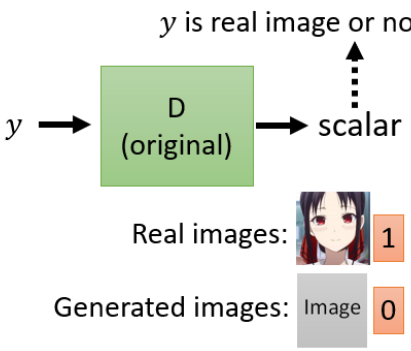
这学期虽然没有,但过去有这个作业,就是输入红头髮,这个是之前助教做的结果,输入红头髮,输入绿眼睛,那產生的结果就是这个样子,產生各式各样红头髮 绿眼睛的角色,输入蓝头髮 红眼睛,就產生各式各样蓝头髮 红眼睛的角色,你发现,那个有时候机器也是会犯错的啦,比如说这边有一个异色瞳,虽然说要画红眼睛,但它觉得画一隻红色的眼睛就可以矇混过去,另外一隻眼睛仍然是蓝色的

我们现在的 Generator 有两个输入,一个是从 Normal Distribution,Sample 出来的 z,另外一个是 x,也就是一段文字



那我们的 Generator 会產生一张图片 y,那我们需要一个 Discriminator,那如果按照我们过去所学过的东西,Discriminator,它就是吃一张图片 y 当作输入,输出一个数值,这个数值代表输入的图片,多像真实的图片

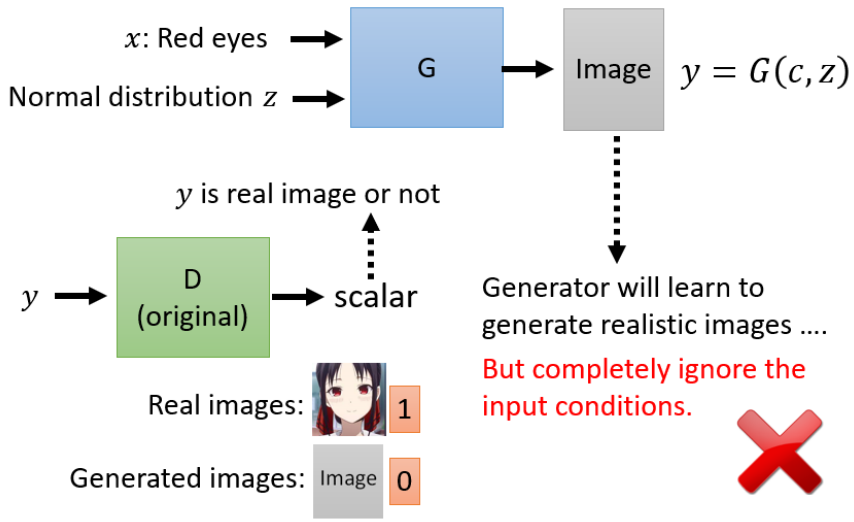




是真实的,还是生成的,那怎麼训练这个 Discriminator ,你就说如果看到真实的图片,你就输出 1,如果看到生成的图片,就输出 0,你就可以训练 Discriminator,然后 Discriminator 跟 Generator 反覆训练

也许你就可以去把 Generator 训练出来,但这样的方法,没办法真的解 Conditional GAN 的问题,为什麼,因為如果我们只有 Train 这个 Discriminator,这个 **Discriminator 只会看 y 当做输入**的话,那 Generator 会学到的是,它会產生可以骗过 Discriminator 的,非常清晰的图片

它会產生清晰的图片,但是**跟输入完全没有任何关係**,因為对 Generator 来说,它只要產生清晰的图片,就可以骗过 Discriminator 了,它何必要去管 Input 文字叙述是什麼



你的 Discriminator 又不看文字的叙述,所以它根本就不需要管文字的叙述,你不管输入什麼文字,就无视这个 x,反正就是產生一个图片,可以骗过 Discriminator 就结束了,但这**显然不是我们要的**

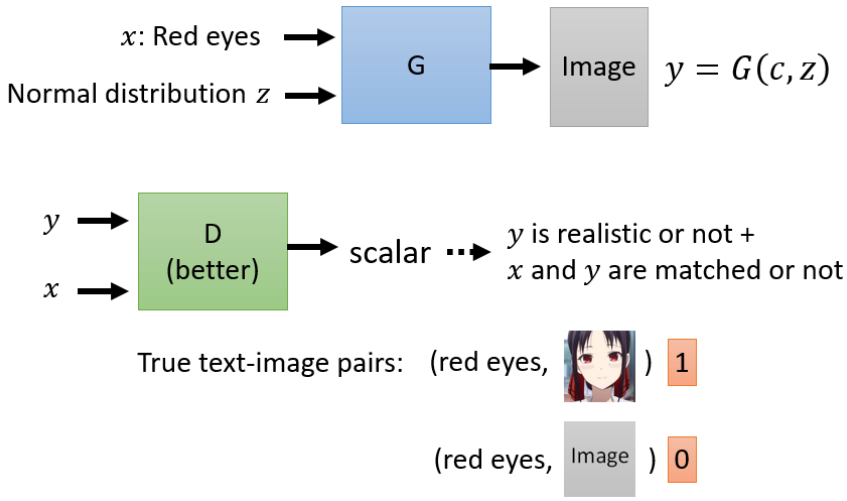
所以在 Conditional GAN 裡面,你要做有点不一样的设计,你的 **Discriminator 不是只吃图片 y**,它**还要吃 Condition x**

所以你的 Discriminator,它有 y 作為输入,有 x 作為输入,然后產生一个数值,那这个数值不只是看 y 好不好,光图片好,没有用,**光图片好,Discriminator 还是不会给高分**

Discriminator 给高分的时候,**一方面图片要好**,另外一方面,这个**图片跟文字的叙述必须要是相配**的,Discriminator 才会给高分

那怎麼样训练这样的 Discriminator ?

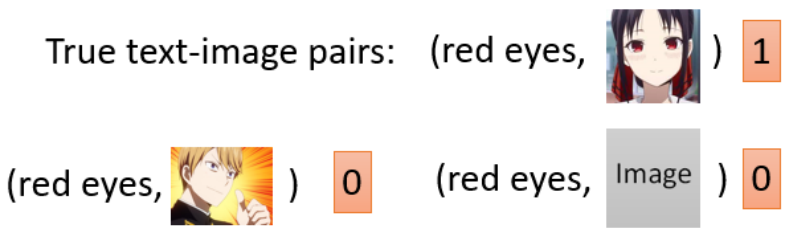
那你需要文字跟影像成对的资料,所以 Conditional GAN,一般的训练,是需要这个 **Pair 的 Data** 的,是需要有标註的资料的,是需要成对资料的



有这些成对资料,那你就告诉你的 Discriminator 说,看到这些真正的成对的资料,就给它一分,看到 Red Eyes,但是搭配,可能 Red Eyes 跟机器產生出来的图片,那就是给 0 分,然后训练下去,就可以產生,就可以做到 Conditional GAN,

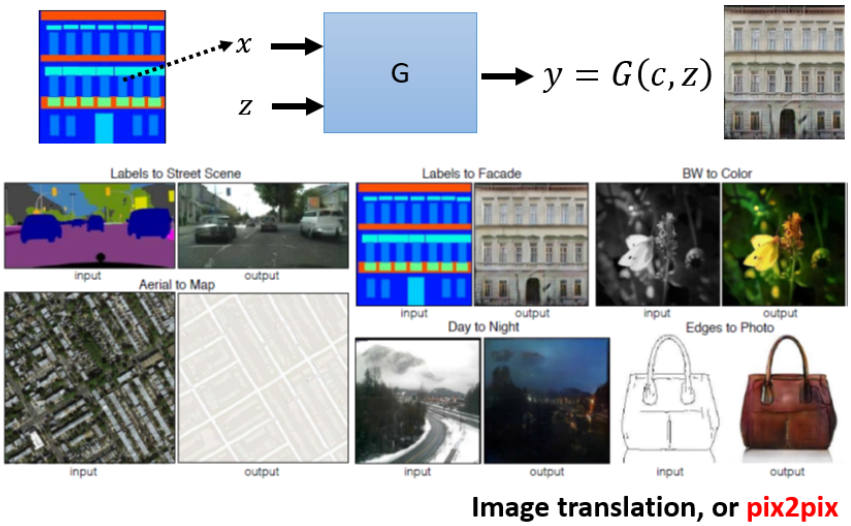
那其实在实作上,光是这样子,拿这样子的 Positive Sample,还有 Negative Sample,来训练这样的 Discriminator,其实你得到的结果往往不够好,光是告诉 Discriminator 说,这样子的状况是好的,这样子的状况是不好的,这样是不够的

你还需要加上一种不好的状况是,**已经產生好的图片,但是文字叙述配不上的状况**



所以你通常会把你的训练资料拿出来,然后故意把文字跟图片乱配,故意配一些错的,然后告诉你的 Discriminator 说,看到这种状况,你也要说是不好的,用这样子的资料,你才有办法把 Discriminator 训练好,然后 Generator 跟 Discriminator,反覆的训练,你最后才会得到好的结果,这个就是 Conditional GAN

在目前的例子裡面都是,看一段文字產生图片,那 Conditional GAN 的应用,不只看一段文字產生图片啦,也可以**看一张图片,產生图片**



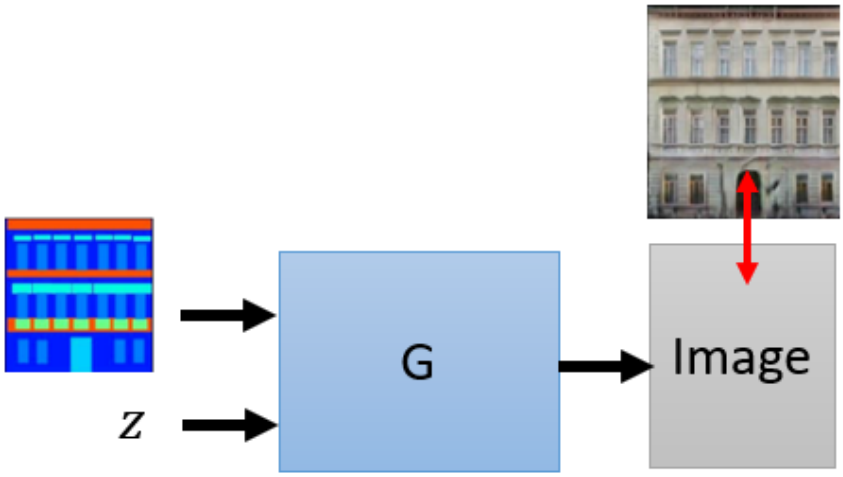
那看一张图片產生图片,也有很多的应用,比如说

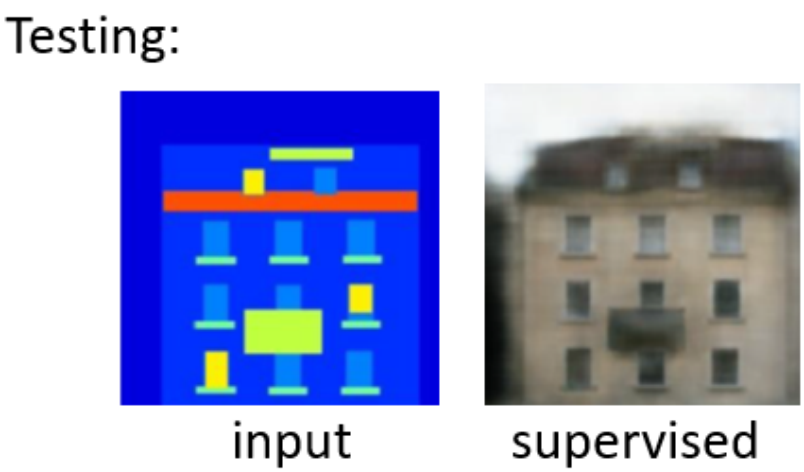
- 给它房屋的设计图,然后让你的 Generator 直接把房屋產生出来
- 给它黑白的图片,然后让它把顏色著上
- 给它这个素描的图,让它把它变成实景 实物
- 那给它这个白天的图片,让它变成晚上的图片
- 有时候你会给它,比如说起雾的图片,让它变成没有雾的图片,把雾去掉

所以 Conditional GAN,除了输入文字 產生影像以外,也可以输入影像 產生影像,那像这样子的应用,叫做 ==Image Translation==,那有人又叫做 ==Pix2pix==,这个 Pix 就是 Pixel,就是像素的缩写啦,所以叫做 Pix2pix

实现以上效果跟刚才讲的从文字產生影像,没有什麼不同,现在只是从影像產生影像,把文字的部分用影像取代掉而已,那当然同样的做法,同样要產生这样的 Generator,產生一张图片,输入一张图片 產生一张图片,你当然可以用 Supervised Learning的方法

在文献上你会发现说,如果你用 Supervised Learning 的方法,你得不到非常好的结果,通常你用 Supervised Learning 的方法,训练一个图片生图片的 Generator,你產生出来的结果可能是这个样子

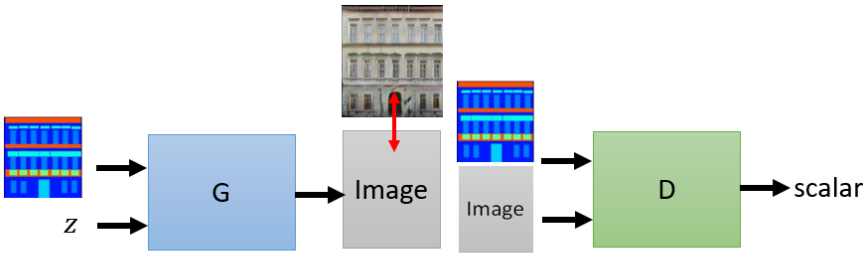


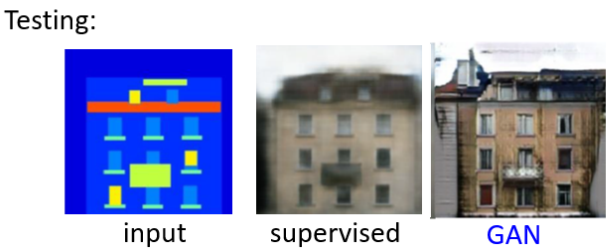


就是这是你的 Generator 的输入,那这个是你 Generator 的输出,那你会发现说它**非常地模糊**,為什麼它非常地模糊,你可以直觉想成说,因為**同样的输入,可能对应到不一样的输出**,就好像我们在讲 GAN 刚开始的,开场的时候讲的那个例子,今天在同一个转角,那个小精灵可能左转,也可能右转,最后学到的,就是同时左转跟右转

那对於 Image To Image 的 Case,也是一样的,输入一张图片,输出有不同的可能,机器学到的,Generator 学到的,就是把不同的可能平均起来,结果变成一个模糊的结果

所以这个时候我们需要用 GAN 来 Train,你**需要加一个 Discriminator**,Discriminator 它是输入一张图片,还有输入 Condition,然后它会同时看这个图片跟这个 Condition,有没有匹配,来决定它的输出,那这个是文献上用 GAN 的输出,从右上角这篇 Paper 截取出来的





那你会发现说,如果单纯用 GAN 的话,它有一个小问题,所以它產生出来的图片,比较真实,但是它的问题是它的**创造力,想像力过度丰富**,它会產生一些输入没有的东西,没有叫它输入的东西,举例来说,这是一个房子,左上角明明没有其他东西,这边它却在屋顶上,加了一个不知道是烟囱还是窗户的东西

那文献上如果你要做到最好,往往就是 GAN 跟 Supervised Learning,同时使用



那同时使用,往往可以给你最好的结果,那所谓同时使用的意思就是,Generator 在训练的时候,一方面它要去骗过 Discriminator,这是它的一个目标,但同时它又想要產生一张图片,跟标準答案越像越好,它同时去做这两件事,那往往產生出来的结果是最好的

Conditional GAN 还有很多应用啦,这边给大家看一个莫名其妙的应用,就是给 GAN ,听一段声音,然后它產生一个对应的图片啦

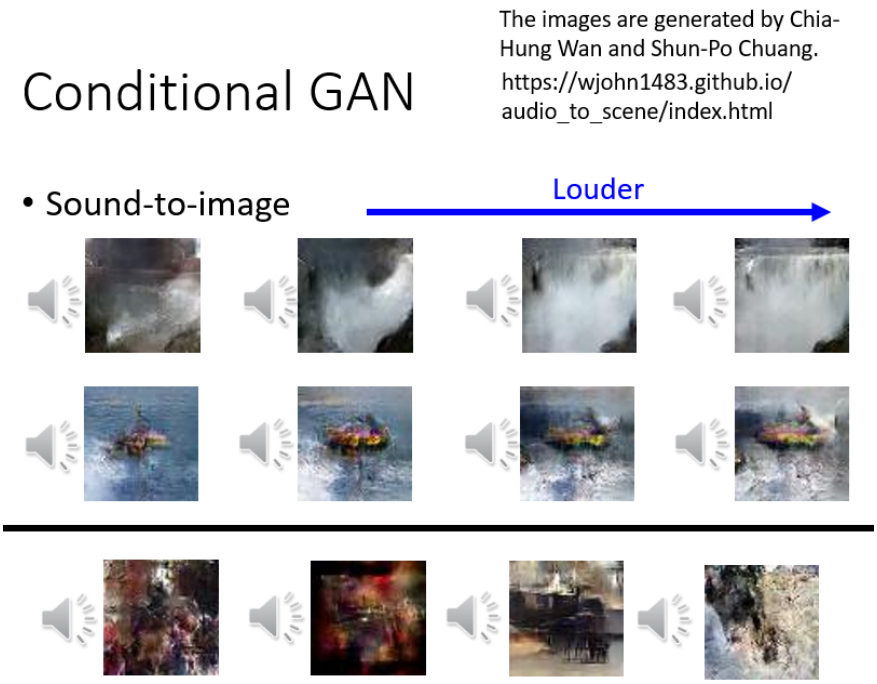




比如说给它听一段狗叫声,看它能不能够画出一隻狗啦,好 那我刚才讲说 Conditional GAN 需要这个,Label 的资料,需要成对的资料

那这个声音跟影像成对的资料,其实并没有那麼难蒐集,因為你可以爬到大量的影片,那影片裡面有影像 有画面,也有声音讯号,那你就知道说,这一个画面 这一帧,这一帧的图片,这一帧的画面,对应到这一小段声音,这一帧的画面对应到这一小段声音,把这些资料蒐集起来,你就可以 Train 一个 Conditional GAN

那这个是我们实验室有个同学做的,这个是一个那个真正的 Demo



那机器听这样的声音,好 这听起来有点像是这个电视机坏掉的声音,那机器觉得它听到什麼,刚才那一段声音机器觉得,它听到一个小溪,听到一个小瀑布

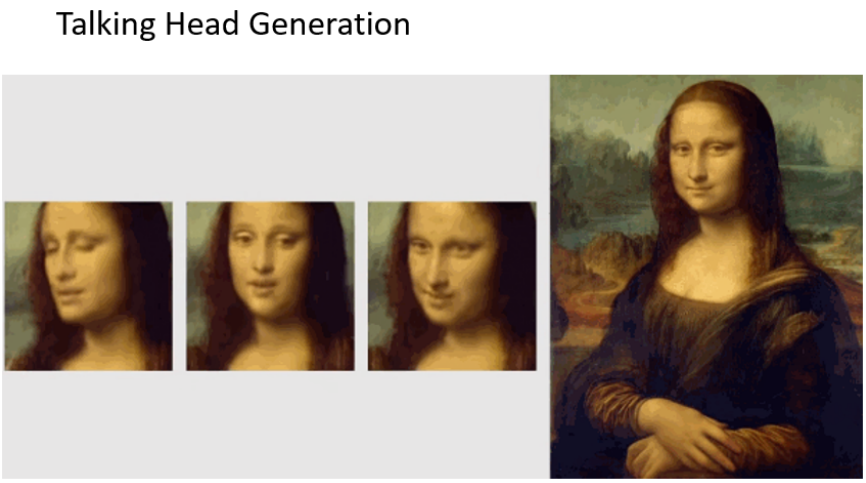
或者是我们再听另外一段声音,机器觉得它听到一艘快艇在海上奔驰

当然我有点担心说,欸 这个会不会机器并没有真的学到,声音跟图片之间的关係,会不会它只是把,它在训练资料裡面有看过的图片存起来而已,所以我决定把声音调大,你听看看结果会怎样,所以我们把声音调大,接下来真的很大声哦,好 然后声音越来越

大,你就发现说,这个溪流裡面的水花就越来越多,从一条小溪,变成尼加拉瓜瀑布

然后刚才的这个快艇的例子也是一样,就把快艇的声音变大,你听看看会怎样,当声音越来越大的时候,你发现快艇旁边的水花就越来越多,好像快艇开得越来越快

不过我要承认,这个其实是稍微 Cherry Pick 的结果,就稍微挑过的结果,很多时候觉得 Generator 產生出来的东西,就是这个样子啦,不知所云这样,这就给它一个钢琴声,然后它好像想画一个钢琴,但又没有很清楚,这个是给它听狗叫声啦,好像想画一个动物,但又不知道要画些什麼,这个是声音到影像的產生,好 那我看到最近最惊人的,Conditional GAN 的应用,是有人用 Conditional GAN 產生会动的图片



我们知道在哈利波特裡面,那些人物的画像是会动的,是会说话的,那 Samsung ,就做了一个类似的应用,用 GAN 做的,给它一张图片,比如说蒙娜丽莎的画像,然后就可以让蒙娜丽莎开始讲话,这个是 Conditional GAN 的其中一个应用,我把论文放在这边给大家参考

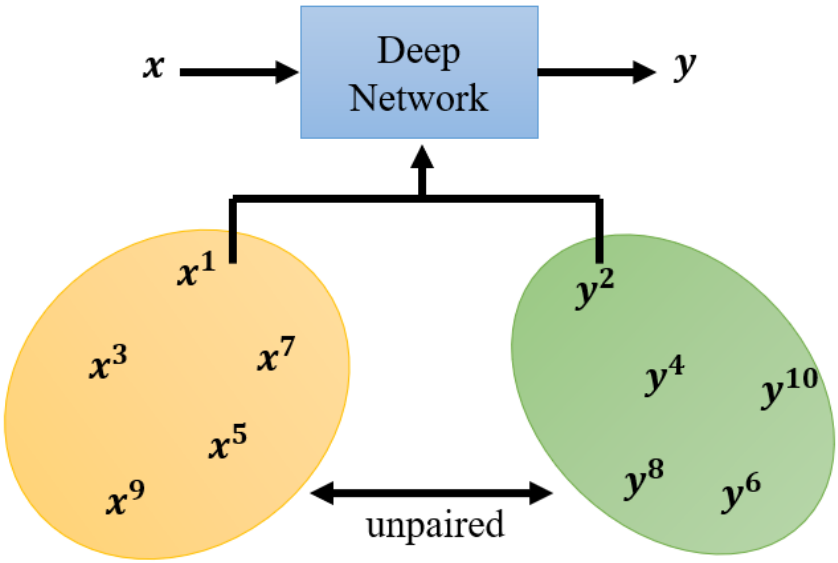
GAN_P4 Learning from Unpaired Data

有关GAN的最后一段,是一个GAN的神奇应用,它把GAN用在==unsupervised Learning==上,到目前為止,我们讲的几乎都是==Supervised Learning==



我们要训练一个Network,Network的输入叫做X输出叫做Y,我们需要**成对的资料**,才有办法训练这样子的Network,

但是你可能会遇到一个状况是,我们有一堆X我们有一堆Y,但**X跟Y是不成对**的,在这种状况下,我们有没有办法拿这样的资料,来训练Network呢,像这一种没有成对的资料,我们就叫做==unlabeled==的资料



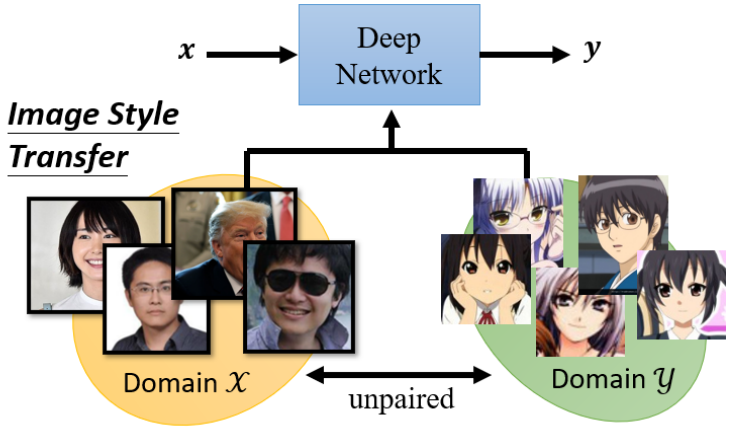
其实在作业三跟作业五裡面,都提供给你两个例子,我们就把这个怎麼用,没有标註的资料,怎麼做S==emi-supervised Learning==,这件事情放在作业裡面,如果你有兴致的话就可以来,体验一下semi-supervised Learning,到底可以带多大的帮助

但是不管是作业三的pseudo labeling,还是作业五的back translation,这些方法或多或少,都**还是需要一些成对的资料**

在作业三裡面,你得先训练出一个模型,这个模型可以帮你提供pseudo label,如果你一开始,根本就没有太多有标註的资料,你的模型很差,你根本就没有办法產生,比较好的pseudo label,或是back translation,你也得有一个,back translation 的model,你才办法做back translation,所以不管是作业三,还是作业五的方法,还是都需要一些成对的资料

但是假设我们遇到,一个更艰鉅的状况,是我们**一点成对的资料都没有**,那要什麼怎麼办呢?

我们这边举一个例子,**影像风格转换**,假设今天我要训练,一个Deep Network,它要做的事情是把X domain的图,X domain的图,我们假设是真人的照片,Y domain的图是二次元人物的头像



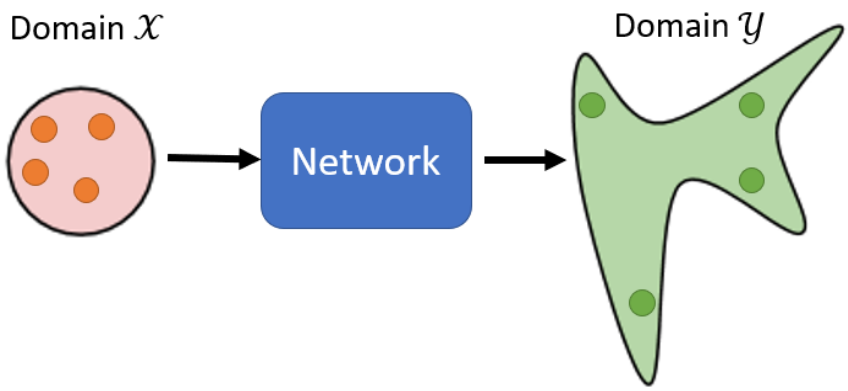
Can we learn the mapping without any paired data?

Unsupervised Conditional Generation

在这个例子裡面我们可能,就**没有任何的成对的资料**，在这种状况下,还有没有办法训练一个Network,输入一个X產生一个Y呢,这个就是GAN可以帮我们做的事情

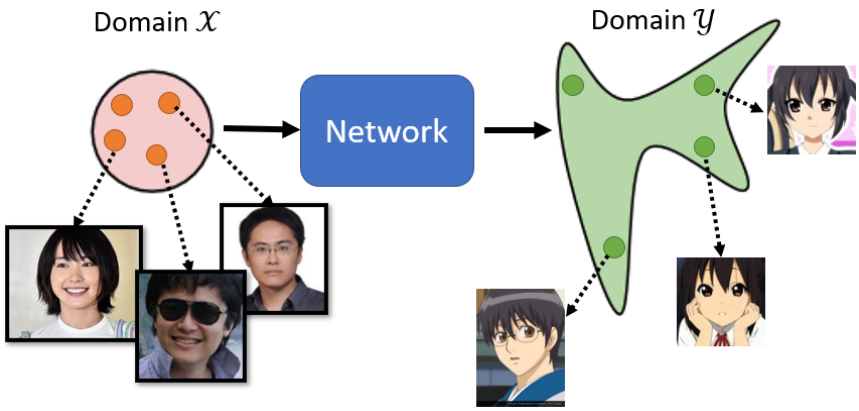
那接下来我们就是看看怎麼用GAN,在这种完全没有成对资料的情况下,进行学习

这个是我们之前在讲,unconditional的generation的时候,你看到的generator的架构



输入是一个Gaussian的分佈,输出可能是一个复杂的分佈

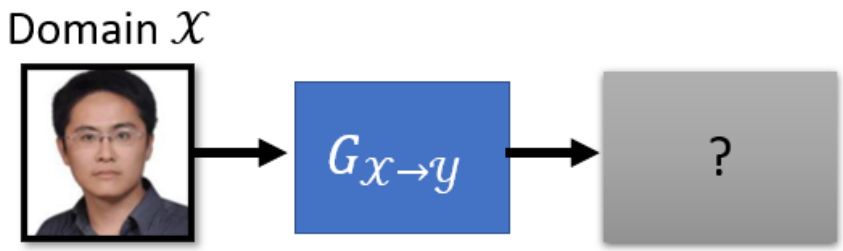
现在我们在稍微,转换一下我们的想法,**输入**我们不说它是Gaussian的分佈,我们说它是**X domain的图片的分佈**,那**输出**我们说,是**Y domain图片的分佈**



乍听之下好像没有很难,你完全可以**套用原来的GAN的想法**,在原来的GAN裡面我们说,我们从Gaussian sample一个向量,丢到Generator裡面

那我们一开始也说,其实不一定只要Gaussian sample这一个distribution,只要是有办法被sample的就行了,我们选Gaussian只是因為Gaussian的formulation我们知道

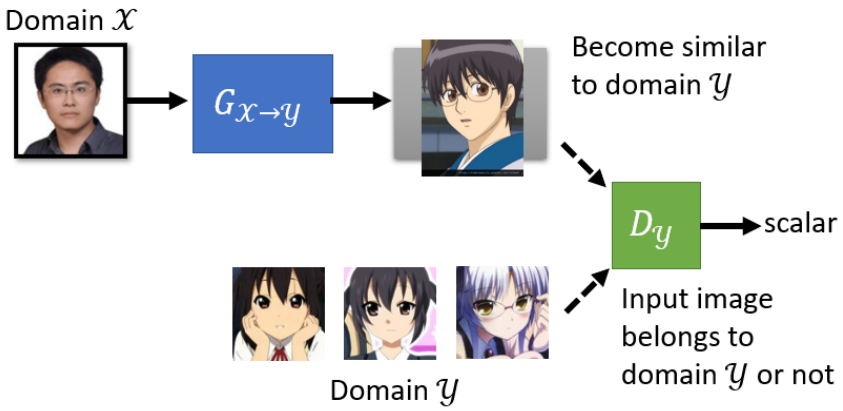
那我们现在如果,输入是X domain的distribution,我们只要改成可以,从X domain sample就结束了,那你有没有办法,从X domain sample呢



可以 你就从人脸的照片裡面,真实的人脸裡面随便挑一张出来,这是一个死臭酸宅(老师本人)然后就结束了,你就可以从X domain,sample照片出来,你把这个照片丢到generator裡面,让它產生另外一张图片,產生另外一个distribution裡面的图片

那怎麼让它变成,是Y domain的distribution呢?

那就要两三个discriminator,那这个discriminator给它,看过很多Y domain的图,所以它能够分辨Y domain的图,跟不是Y domain的图的差异

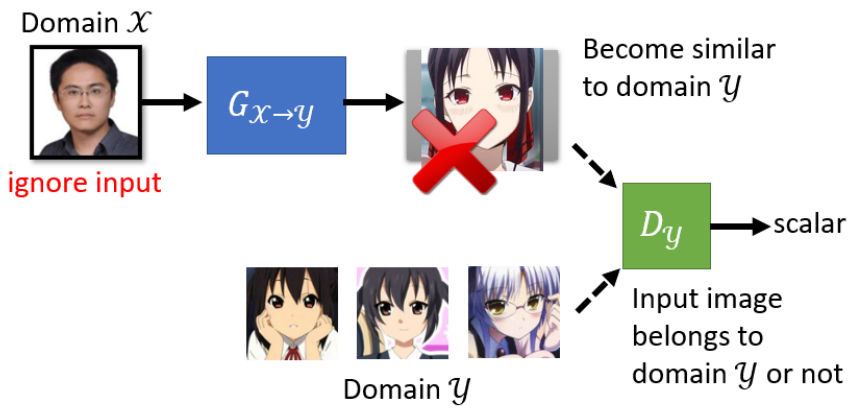


看到**Y domain的图**就给它**高分**,看到**不是Y domain的图**,不是二次元人物就给它**低分**,那就这样结束了

但是光是套用原来的GAN训练,generator跟discriminator,好像是**不够的**,因為我们现在的discriminator,它要做的事情是要让这个generator,输出一张Y domain的图

那generator它可能真的,可以学到输出Y domain的图,但是它输出的Y domain的图,一定要**跟输入有关係吗**,你**没有任何的限制**要求你的generator做这件事

你的generator也许就把这张图片,当作一个Gaussian的noise,然后反正它就是看到,不管你输入什麼它都无视它,反正它就输出一个,像是二次元人物的图片,discriminator觉得它做得很好,其实就结束了



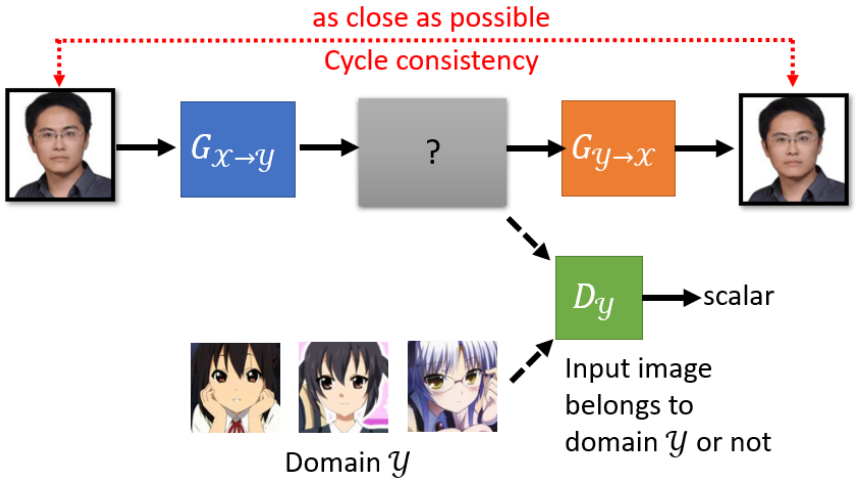
所以如果我们完全只套用,这个一般的GAN的做法,只训练一个generator,这个generator input的distribution,从Gaussian变成X domain的image,然后训练一个discriminator,显然是不够的,因為你训练出来的generator,它可以**產生二次元人物的头像,但是跟输入的真实的照片,没有什麼特别的关係**,那这个不是我们要的,

我们在conditional GAN的时候,是不是也看过一模一样的问题呢,在讲conditional GAN的时候,我有特别提到说,假设你的discriminator只看Y,那它可能会无视generator的输入,那產生出来的结果不是我们要的,但是这边啊,如果我们要**从unpaired的data学习,我们也没有办法,直接套用conditional GAN的想法**,因為在刚才讲的,**conditional GAN裡面,我们是有成对的资料**,来训练的discriminator

Cycle GAN

这边这个想法叫做==Cycle GAN==,在Cycle GAN裡面,我们会train**两个generator**

- 第一个generator它的工作是,把X domain的图变成Y domain的图
- 第二个generator它的工作是,看到一张Y domain的图,把它**还原**回X domain的图



在训练的时候,我们今天增加了一个额外的目标,就是我们希望输入一张图片,从X domain转成Y domain以后,要从Y domain转回原来,一模一样的X domain的图,经过两次转换以后,**输入跟输出要越接近越好**

你说怎麼让两张图片越接近越好呢?

两张图片就是**两个向量**,这两个向量之间的距离,你就是让这两个向量,它们之间的距离越接近越好,就是要两张图片越像越好

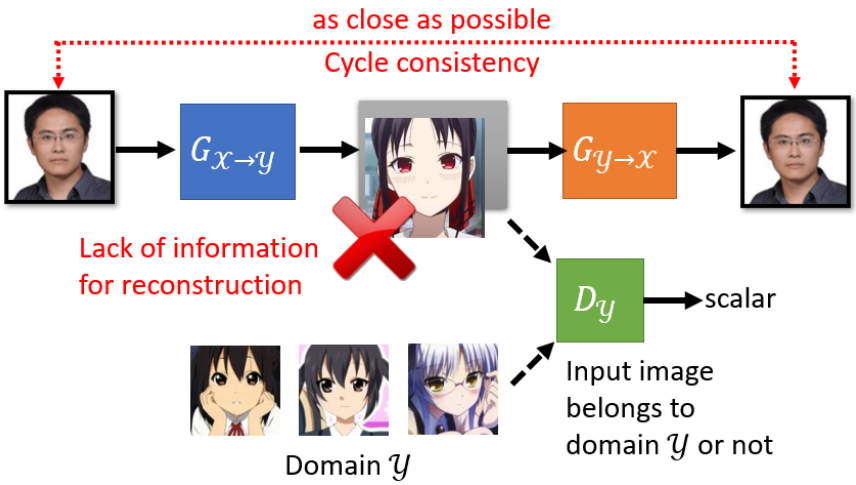
因為这边有一个循环,从X到Y 在从Y回到X,它是一个cycle,所以叫做Cycle GAN,这个要让输入经过两次转换以后,变成输出 输入跟输出越接近越好,这个叫做==Cycle的consistency==

所以现在这边我们有**三个Network**

1. 第一个generator,它的工作是把X转成Y
2. 第二个generator,它的工作是要把Y还原回原来的X
3. 那这个discriminator,它的工作仍然是要看,蓝色的这个generator它的输出,像不像是Y domain的图

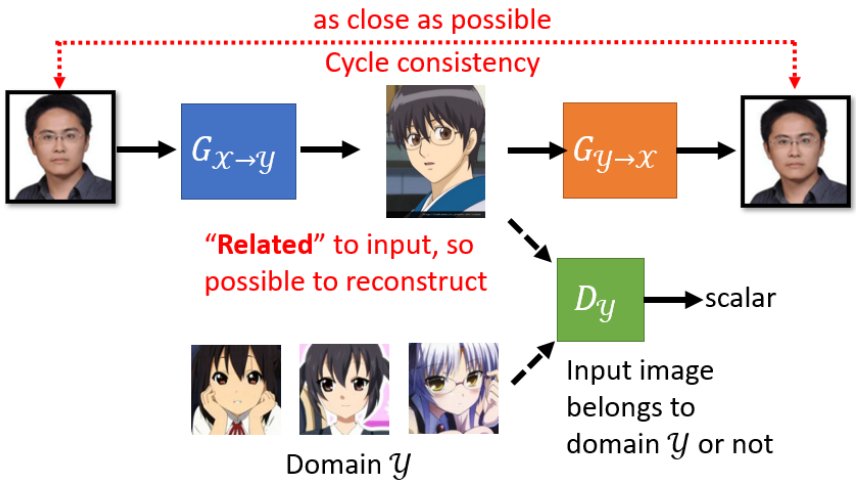
那**加入了这个橙色的**从Y到X的generator以后,对於前面这个蓝色的generator来说,它就再也不**能够随便乱做**了,它就不能够随便產生乱七八糟,跟输入没有关係的人脸了

这边假设输入一个死臭酸宅,这边假设输出的是辉夜



另外一个这个不知道这是谁的,然后对第二个generator来说,它就是视这张辉夜作为输入,它根本无法想像说,要把辉夜还原回死臭酸宅,它根本不知道说,原来输入的图片长什麼样子

所以对第一个generator来说,為了要让第二个generator能够,成功的还原原来的图片,它產生出来的图片,就不能跟输入差太多,所以这边是一个死臭酸宅,这边输出至少也得是一个,戴眼镜的男生的角色才行



所以这边是一个戴眼镜男生的角色,然后第二个generator才能够,把这个角色还原回原来的输入,所以如果你加Cycle GAN,你至少可以强迫你的generator,它输出的Y domain的图片,至少跟输入的X domain的图片,有一些关係

这时你可能会有的一个问题就是,你这边只保证有一些关係啊,**你怎麼知道这个关係是我们要的呢**?

- 机器有没有可能学到很奇怪的转换,输入一个戴眼镜的人,然后这个generator学到的是,看到眼镜就把眼镜抹掉,然后把它变成一颗痣,然后第二个generator橙色的学到的,就是看到痣就还原回眼镜,这样还是可以满足cycle consistency,还是可以把输入的图片,变成输出的图片
- 一个更极端的例子,假设第一个generator学到的就是,把图片反转 左右翻转,第二个generator它也只要学到,把图片左右翻转,你就可以还原了啊

所以今天如果我们做Cycle GAN,用cycle consistency,似乎没有办法保证,我们输入跟输出的人脸,看起来真的很像,因為也许机器会学到很奇怪的转换,反正只要第二个generator,可以转得回来就好了

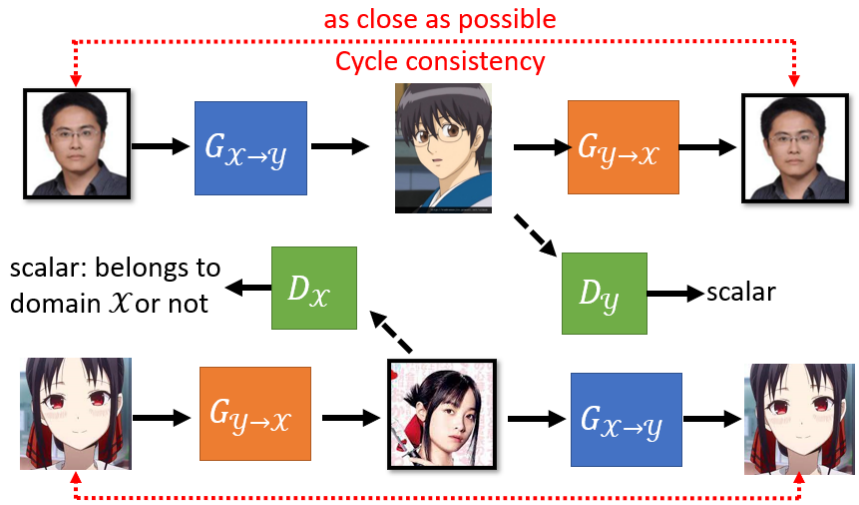
确实有可能有这样的问题发生,而且**目前没有什麼特别好的解法**

但我可以告诉你说,实际上你要使用Cycle GAN的时候,**这样子的状况没有那麼容易出现**,如果你实际上使用Cycle GAN,你会发现输入跟输出往往,真的就会看起来非常像,而且甚至在实作上,在实作的经验上,你就算没有第二个generator,你不用cycle GAN,拿一般的GAN来做,这种图片风格转换的任务,你往往也做得起来

因為在实作上你会发现,**Network其实非常懒惰**,它输入一个图片,它往往就想输出,by default就是想输出很像的东西,它不太想把输入的图片,做太复杂的转换,像是什麼眼镜变成一颗痣这种状况,它不爱这麼麻烦的东西,有眼镜就输出眼镜,可能对它来说是比较容易的抉择,所以在真的实作上,这个问题没有很大,输入跟输出会是像,但是理论上好像没有什麼保证说,输入跟输出的图片一定要很像,就算你加了cycle consistency

所以这个是实作与理论上,你可能会遇到的差异,总之虽然Cycle GAN没有保证说,输入跟输出一定很像,但实际上你会发现输入跟输出,往往非常像,你只是改变了风格而已

那这个**Cycle GAN可以是双向的**,我们刚才有一个generator,输入Y domain的图片,输出X domain的图片,我们是先把X domain的图片转成Y,在把Y转回X

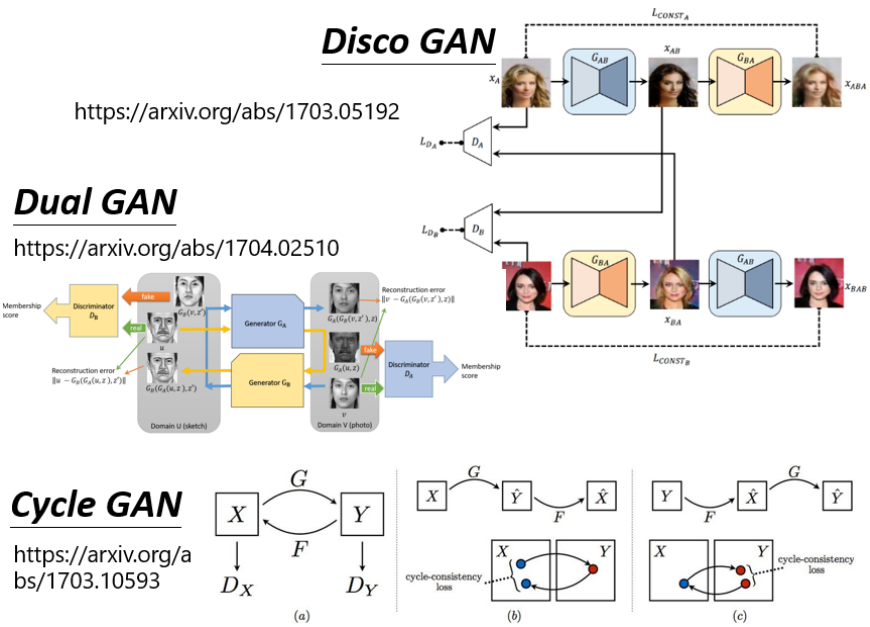


在训练cycle GAN的时候,你可以同时做另外一个方向的训练,也就是

- 把这个橙色的generator拿来,给它Y domain的图片,让它產生X domain的图片
- 然后在把蓝色的generator拿来,把X domain的图片,还原回原来Y domain的图片

那你依然要让,**输入跟输出越接近越好**,那你一样要训练一个discriminator,这个discriminator是,X domain的discriminator,它是要看一张图片,像不像是真实人脸的discriminator,这个discriminator要去看说,这一个橙色的generator的输出,像不像是真实的人脸,这个橙色的generator它要去骗过,这个Dx这个绿色的左边,这一个discriminator,这个**合起来就是Cycle GAN**

那除了Cycle GAN以外,你可能也听过很多其他的,可以做风格转换的GAN



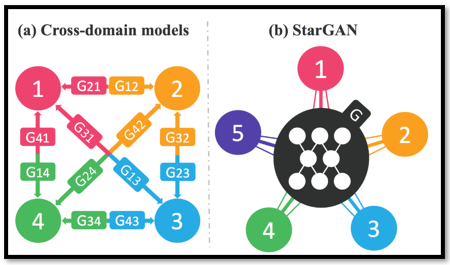
比如说Disco GAN 比如说Dual GAN,他们跟Cycle GAN有什麼不同呢,就是没有半毛钱的不同这样子

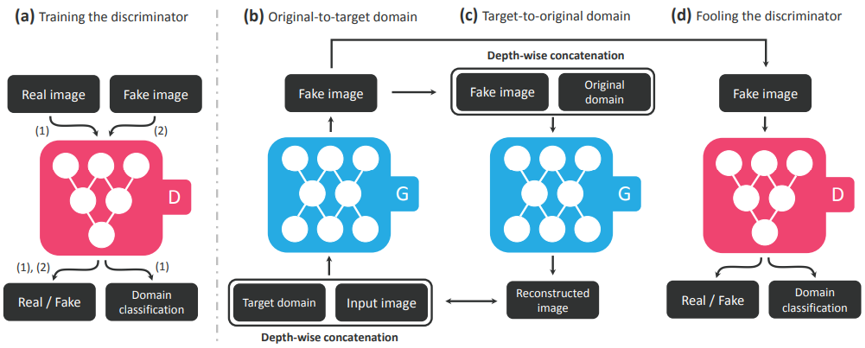
你可以发现Disco GAN,Dual GAN跟Cycle GAN,其实是一样的东西,他们是一样的想法,神奇的事情是完全不同的团队,在几乎一样的时间,提出了几乎一模一样的想法,你发现这三篇文章,放到arxiv上的时间,都是17年的3月,17年的4月跟17年的3月

除了Cycle GAN以外,还有另外一个更进阶的,可以做影像风格转换的版本,叫做StarGAN

StarGAN

https://arxiv.org/abs/1711.09020





Cycle GAN只能在两种风格间做转换,那StarGAN 它厉害的地方是,它可以在多种风格间做转换,不过这个就不是我们接下来,想要细讲的重点

这个真实的人脸转二次元的任务,实际上能不能做呢,实际上可以做了

SELFIE2ANIME

https://selfie2anime.com/

https://arxiv.org/abs/1907.10830



右上角这边放了一个连结,这个应该是一个韩国团队,他们做了一个网站,你可以上传一张图片,它可以帮你变成二次元的人物,他们实际上用的不是Cycle GAN啦,他们用的也是GAN的技术,但是是一个进阶版的东西,那我们这边就不细讲,我就把论文的连结,放在这边给大家参考

我就实际测试了一下,这个不知道大家认不认得,这是新垣结衣,这个是你老婆这样,你总该认得吧,把这个图片转成,把你老婆转成二次元的人物,长成是这个样子,你老婆二次元长这个样子知道吗

你会发现说机器确实,有学到一些二次元人物的特徵,比如说 把眼睛变大,本来眼睛其实没有很大,变成二次元人物之后,眼睛变这麼大,但有时候也是会失败

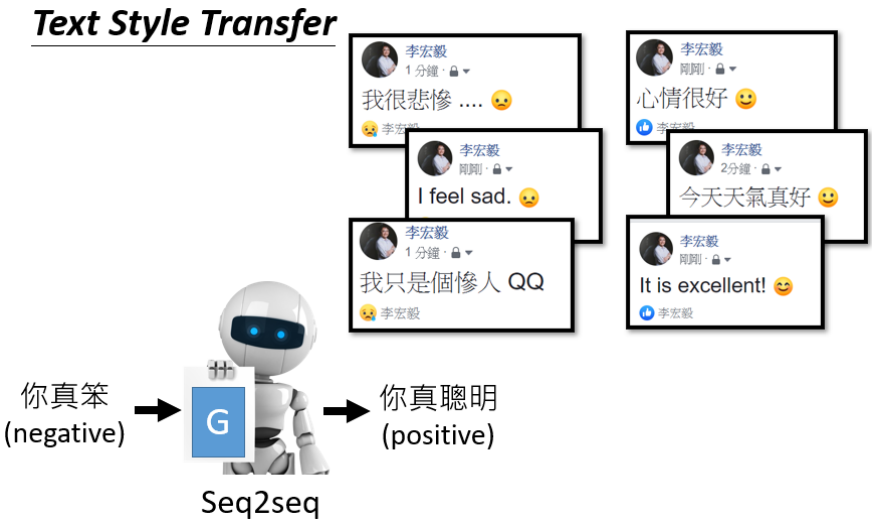


比如说 这个是美国前总统,转完以后变成这个样子,两隻眼睛一眼大一眼小就是了,它不是总是会成功的

Text Style Transfer

那同样的技术不是只能用在影像上,也可以用在文字上,你也可以做**文字风格的转换**

比如说,把一句负面的句子转成正面的句子,当然如果你要做一个模型,输入一个句子输出的句子,这个模型就是要能够,吃一个sequence 输出一个sequence,所以它等於是一个,sequence to sequence的model



你可能就会用到,我们在作业五裡面的,Transformer的架构,来做这个文字风格转换的问题,我们在作业五做的是翻译嘛,输入一个语言输出另外一个语言嘛

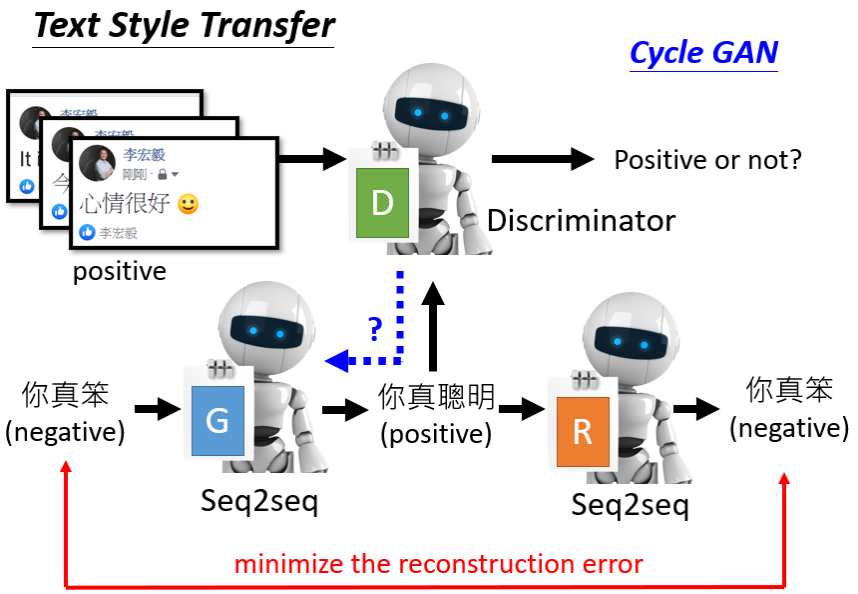
现在如果要做文字风格转换,就是**输入一个句子,输出另外一个风格的句子**

怎麼做文字的风格转换呢,跟Cycle GAN是一模一样的,首先你要有训练资料,**收集一大堆负面的句子,收集一大堆正面的句子**

这个其实没有那麼难收集,你可以就是网路上爬一爬,像我们就是去PTT上爬,然后只要是推文就当作是正面的,嘘文就当作是负面的,就有一大堆正面的句子,跟负面的句子,只是成对的资料没有而已,你不知道这句推文,要怎麼转成这句嘘文,这些嘘文要怎

麼转成这句推文,你没有这种资料,但是一堆推文一堆噓文的资料,你总是可以找得到的

那接下来呢,完全套用Cycle GAN的方法,完全没有任何不同



这边就不需要再细讲 很快讲过

有一个discriminator,discriminator要看说,假设我们是要负面的句子,转正面的句子,discriminator要看说,现在generator的输出,像不像是真正的正面的句子

然后我们还要有另外一个generator,要有**两个generator**,这个generator要学会,把**正面的句子转回原来负面的句子**,你要用Cycle consistency,负面的句子转成正面的以后,还可以**转回原来负面的句子**

你可能会问说,这两个句子 它们两个是句子啊,怎麼算它们的相似度啊?

图片还比较好理解,图片就是个向量啊,两个向量的距离就是它们的相似度,那两个句子要怎麼做呢,这个如果你有兴趣,在留给你慢慢研究,那这边还有另外一个问题就是,这个sequence to sequence model,输出是文字,可是刚才不是有讲说,如果输出是文字,接到discriminator会有问题吗,对 会有问题 这边你就要用RL硬做

那做出来的结果怎麼样呢,这个是真正的demo， 就是真的拿PTT的推文,当正面的句子,噓文当负面的句子,那你就可以给它一个负面的句子,它就帮你转成正面的句子,做起来像是这个样子

感謝 張瓊之 同學提供實驗結果

Text Style Transfer



- From negative sentence to positive one

胃疼，沒睡醒，各種不舒服 → 生日快樂，睡醒，超級舒服

我都想去上班了，真夠賤的! → 我都想去睡了，真帥的！

暈死了，吃燒烤、竟然遇到個變態狂
→ 哈哈好～，吃燒烤～竟然遇到帥狂

我肚子痛的厲害 → 我生日快樂厲害

,你可能问说这个系统有什麼用,就是没有任何用处 没半点用处,但是如果你觉得,你的老闆说话特别坏的话,就可以把这个系统,装在你的耳机裡面,把所有的负面的句子,转成正面的句子,你的人生可能就会,过得特别快乐一点

那其实像这一种文字风格转换,还有很多其他的应用,**不是只有正面句子转负面句子**

Unsupervised Abstractive Summarization

https://arxiv.org/abs/1810.02851



Unsupervised Translation

https://arxiv.org/abs/1710.04087
https://arxiv.org/abs/1710.11041



Unsupervised ASR

https://arxiv.org/abs/1804.00316
https://arxiv.org/abs/1812.09323
https://arxiv.org/abs/1904.04100

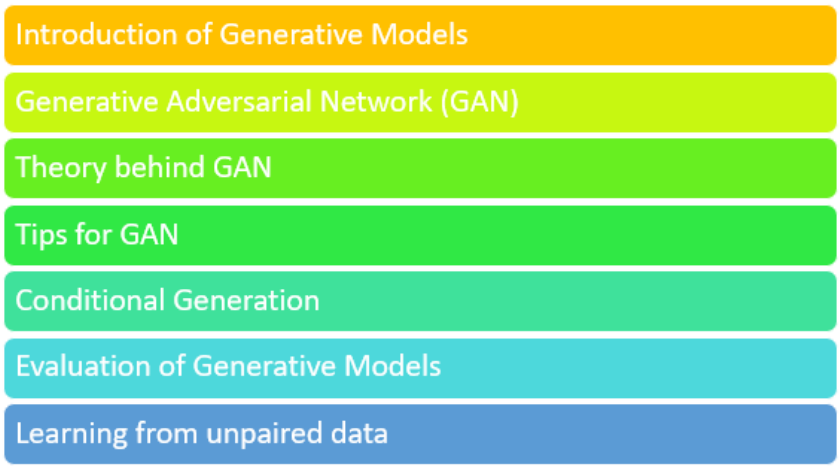


举例来说 假设我有很多长的文章,我有另外一堆摘要,这些摘要不是这些长的文章的摘要,是不同的来源,一堆长的文章 一堆摘要,让机器学习文字风格的转换,你可以让机器学会**把长的文章,变成简短的摘要**,让它学会怎麼精简的写作,让它学会把长的文章变成短的句子

甚至还有更狂的,同样的想法可以做,unsupervised的翻译,什麼叫做unsupervised的翻译呢,收集一堆英文的句子,收集一堆中文的句子,没有任何成对的资料,这就跟你作业五不一样,作业五你有成对的资料嘛,你有知道说这句英文对到这句中文,但是unsupervised翻译就是,完全不用任何成对的资料,网路上爬一堆中文,网路上爬一堆英文,用刚才那个**Cycle GAN的做法硬做,机器就可以学会把中文翻成英文了,**你可以自己看一下文献,看看说机器做得怎麼样

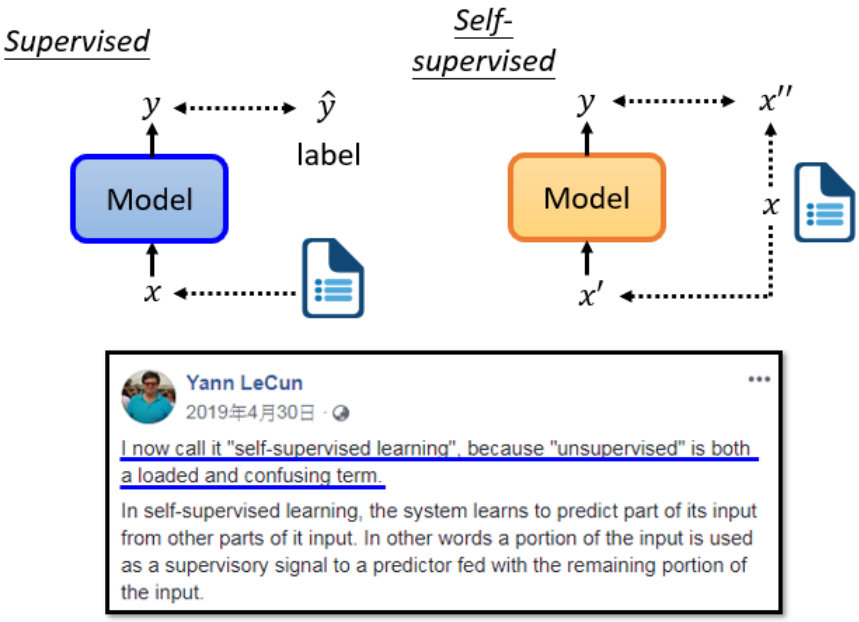
到目前為止,我们说的两种风格都还是文字,可不可以两种风格,甚至是**不同类型的资料**呢,有可能做,这是我们实验室是最早做的,我们试图去做**非督导式的语音辨识**,,语音辨识就是,你需要收集成对的资料啊,你需要收集一大堆的声音讯号,然后找工读生,帮你把这些声音讯号标註,机器才能够学会,某个语言的语音辨识,但是要标註资料所费不貲,所以我们想要挑战,非督导式的语音辨识,也就是机器只听了一堆声音,这些声音没有对应的文字,机器上网爬一堆文字,这些文字没有对应的声音,然后用Cycle GAN硬做,看看机器有没有办法,把声音转成文字,看看它的正确率,可以做到什麼样的地步,至於正确率可以做到,什麼样的地步呢,那我把文献留在这边给大家参考,那以上就是有关GAN的部分

Concluding Remarks



BERT 简介

Self-supervised Learning



- 每个人都应该熟悉监督学习，当我们做监督学习时，我们只有一个模型，这个模型的输入是x，输出是y。

  假设你今天想做情感分析，你就是让机器阅读一篇文章，而机器需要对这篇文章进行分类，是正面的还是负面的，你必须先找到大量的文章，你需要对所有的文章进行label。我们需要**有标签和文章数据来训练监督模型**

- "Self-supervised "是用另一种方式来监督，没有标签。假设我们只有一堆没有label的文章，但我们试图找到一种方法把它**分成两部分**。

  我们让其中一部分作为模型的输入数据，另一部分作为标签。

  假设你有没有label的数据，例如，一篇文章叫x，我们把x分成两部分，**一部分叫x'，另一部分叫x"**，我知道现在的说明很抽象。稍后，当我们真正谈论BERT时，你可以更好地理解Self-supervised的含义，以及如何在明明没有办法进行监督训练的情况下，最终还是找到了自己进行监督训练的方法。

  我们把x分成两部分，x'和x"，然后把**x'输入模型，让它输出y**。如果我们在模型训练中使用标签，我们称之为监督学习。由于在Self-supervised学习中不使用标签，我们可以说，Self-supervised学习也是一种无监督的学习方法。但之所以叫Self-supervised Learning，是为了让定义更清晰。

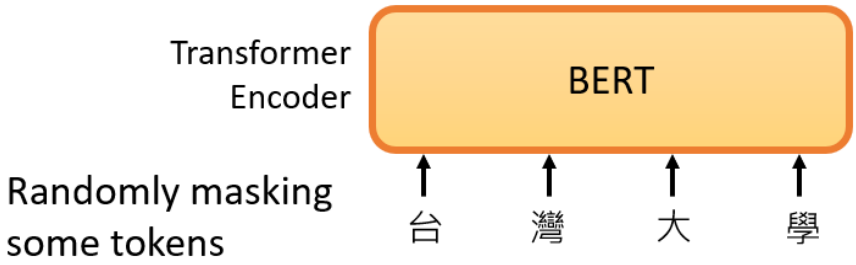
- "Self-supervised Learning "这个词，当初Yann LeCun说过，其实并不是一个老词。根据2019年4月在Facebook上的一个帖子，他说，我现在说的这个方法，他叫Self-supervised Learning。为什么不叫无监督学习呢？因为无监督学习是一个比较大的家族，里面有很多不同的方法，为了让定义更清晰，我们叫它 "自监督"，比如我们之前提到的cycle gan，也是无监督学习的一个案例，我

们也不使用标注的配对数据，但是，它和Self-supervised Learning还是有点区别。在无监督学习的范畴内，有很多方法，Self-supervised Learning就是其中之一。

Masking Input

Self-supervised Learning是什么意思呢，我们直接拿BERT模型来说。

首先，**BERT是一个transformer的Encoder**，我们已经讲过transformer了，我们也花了很多时间来介绍Encoder和Decoder，transformer中的Encoder它实际上是BERT的架构，它和transformer的Encoder完全一样，里面有很多Self-Attention和Residual connection，还有Normalization等等，那么，这就是BERT。

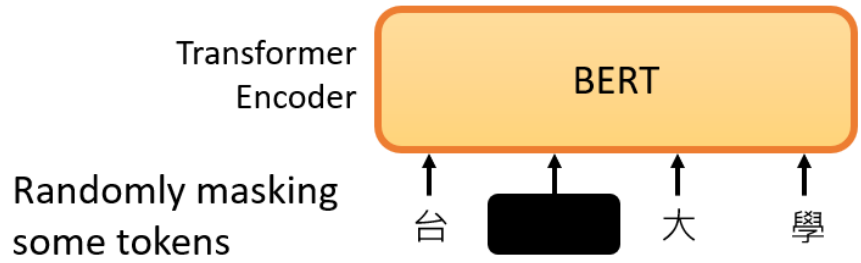


如果你已经忘记了Encoder里有哪些部件，你需要记住的关键点是，**BERT可以输入一行向量，然后输出另一行向量，输出的长度与输入的长度相同**。

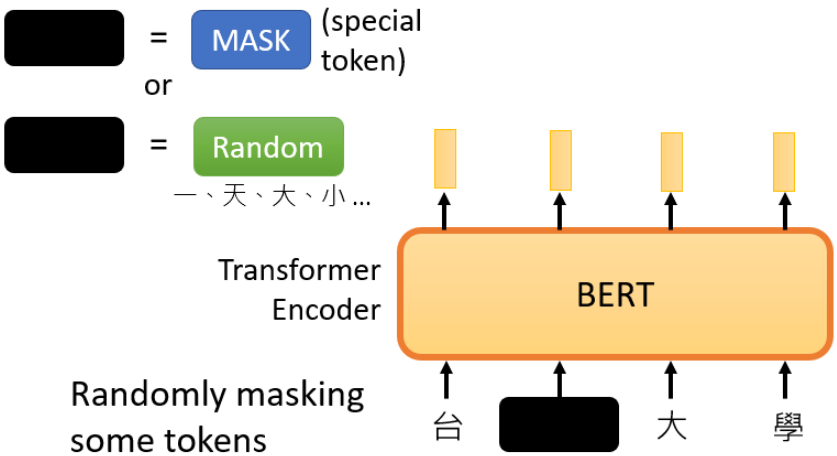
BERT一般用于自然语言处理，用于文本场景，所以一般来说，它的输入是一串文本，也是一串数据。

当我们真正谈论Self-Attention的时候，我们也说**不仅文本是一种序列，而且语音也可以看作是一种序列，甚至图像也可以看作是一堆向量**。BERT同样的想法是，不仅用于NLP，或者用于文本，它也可以用于语音和视频。

接下来我们需要做的是，随机**盖住**一些输入的文字，**被mask的部分是随机决定的**，例如，我们输入100个token，什么是token？在中文文本中，我们通常把一个汉字看作是一个token，当我们输入一个句子时，其中的一些词会被随机mask。



mask的具体实现有**两种方法**。

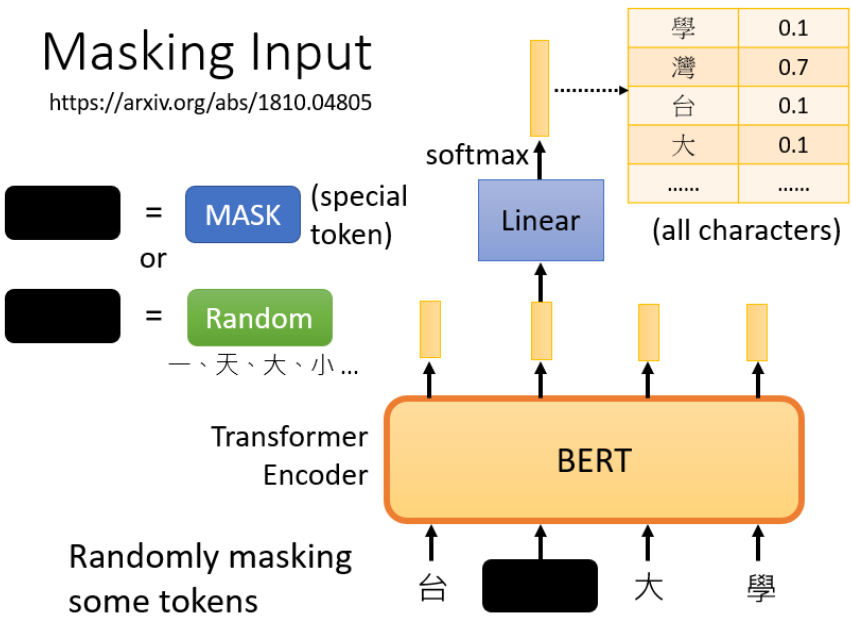


- 第一种方法是，用一个**特殊的符号替换句子中的一个词**，我们用 "MASK "标记来表示这个特殊符号，你可以把它看作一个新字，这个字完全是一个新词，它不在你的字典里，这意味着mask了原文。
- 另外一种方法，**随机**把某一个字**换成另一个字**。中文的 "湾"字被放在这里，然后你可以选择另一个中文字来替换它，它可以变成 "一 "字，变成 "天 "字，变成 "大 "字，或者变成 "小 "字，我们只是用随机选择的某个字来替换它

所以有两种方法来做mask，一种是添加一个特殊的标记 "MASK"，另一种是用一个字来替换某个字。

两种方法都可以使用。**使用哪种方法也是随机决定的**。因此，当BERT进行训练时，向BERT输入一个句子，**先随机决定哪一部分的汉字将被mask**。

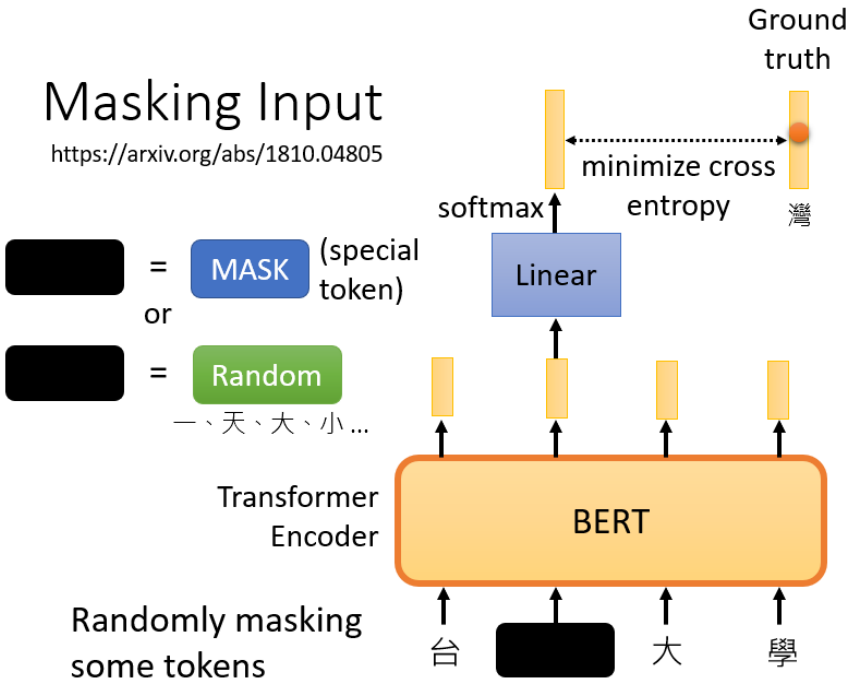
mask后，一样是输入一个序列，我们把BERT的相应输出看作是另一个序列，接下来，我们在输入序列中寻找mask部分的相应输出，然后，这个向量将通过一个 ==Linear transform==。



所谓的Linear transform是指，输入向量将与一个**矩阵相乘**，然后做softmax，输出一个分布。

这与我们在Seq2Seq模型中提到的使用transformer进行翻译时的输出分布相同。输出是一个很长的向量，包含我们想要处理的每个汉字，每一个字都对应到一个分数。

在训练过程中。我们知道被mask的字符是什么，而BERT不知道，我们可以用一个one-hot vector来表示这个字符，并使输出和one-hot vector之间的交叉熵损失最小。



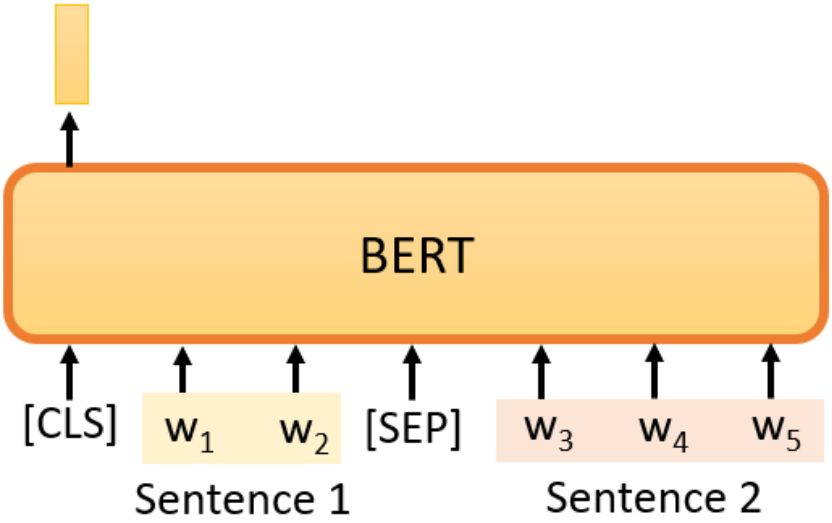
或者说得简单一点，我们实际上是在解决一个**分类问题**。现在，BERT要做的是，**预测什么被盖住**。被掩盖的字符，属于 "湾"类。

在训练中，我们**在BERT之后添加一个线性模型**，并将它们**一起训练**。所以，BERT里面是一个transformer的Encoder，它有一堆参数。这两个需要共同训练，并试图**预测被覆盖的字符是什么**，这叫做mask。

Next Sentence Prediction

事实上，当我们训练BERT时，除了mask之外，我们还会使用另一种方法，这种额外的方法叫做==Next Sentence Prediction== 。

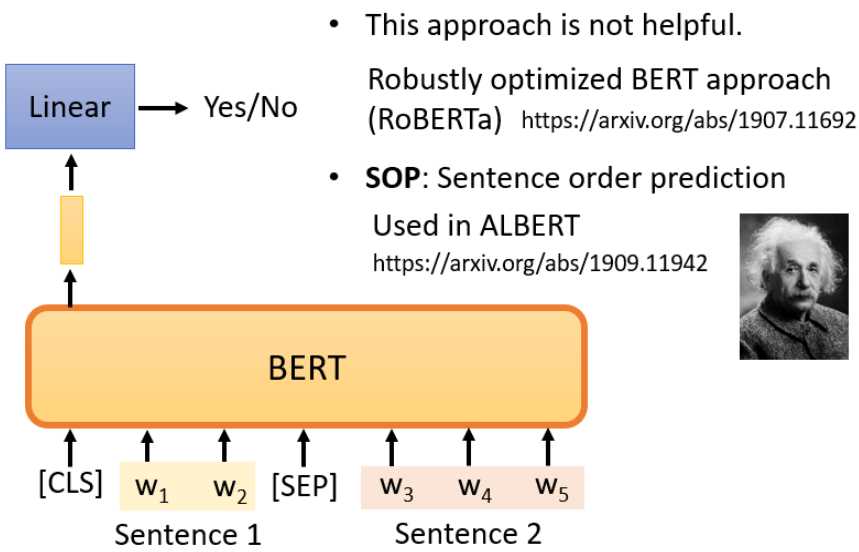
它的意思是，我们从数据库中拿出两个句子，这是我们通过在互联网上抓取和搜索文件得到的大量句子集合，我们在这**两个句子之间**添加一个**特殊标记**。这样，BERT就可以知道，这两个句子是不同的句子，因为这两个句子之间有一个分隔符。



我们还将在句子的**开头**添加一个**特殊标记**，这里我们用CLS来表示这个特殊标记。

现在，我们有一个很长的序列，包括**两个句子**，由SEP标记和前面的CLS标记分开。如果我们把它传给BERT，它应该输出一个序列，因为输入也是一个序列，这毕竟是Encoder的目的。

我们将**只看CLS的输出**，我们将把它乘以一个Linear transform。



现在它必须做一个**二分类问题**，有两个可能的输出：是或不是。这个方法被称为Next Sentence Prediction ，所以我们需要预测，第二句是否是第一句的后续句。

然而，后来的研究发现，对于BERT要做的任务来说，**Next Sentence Prediction 并没有真正的帮助**。例如，有一篇论文叫 "Robustly Optimized BERT Approach"，简称RoBERTa。在这篇论文中，它明确指出，实施Next Sentence Prediction ，几乎没有任何帮助。然后，这个概念不知不觉地成为主流。

在这之后，另一篇论文说下一句话预测没有用，所以在它之后的许多论文也开始说它没有用。例如，SCAN-BERT和XLNet都说Next Sentence Prediction 方法是无用的。它可能是无用的原因之一是，**Next Sentence Prediction 太简单了**，是一项容易的任务。

这个任务的典型方法是，首先随机选择一个句子，然后从数据库中或随机选择要与前一个句子相连的句子。通常，当我们随机选择一个句子时，它看起来与前一个句子有很大不同。对于BERT来说，预测两个句子是否相连并不是太难。因此，在训练BERT完成Next Sentence Prediction 的任务时，**没有学到什么太有用的东西**。

还有一种类似于Next Sentence Prediction 的方法，它在纸面上看起来更有用，它被称为==Sentence order prediction==，简称SOP。

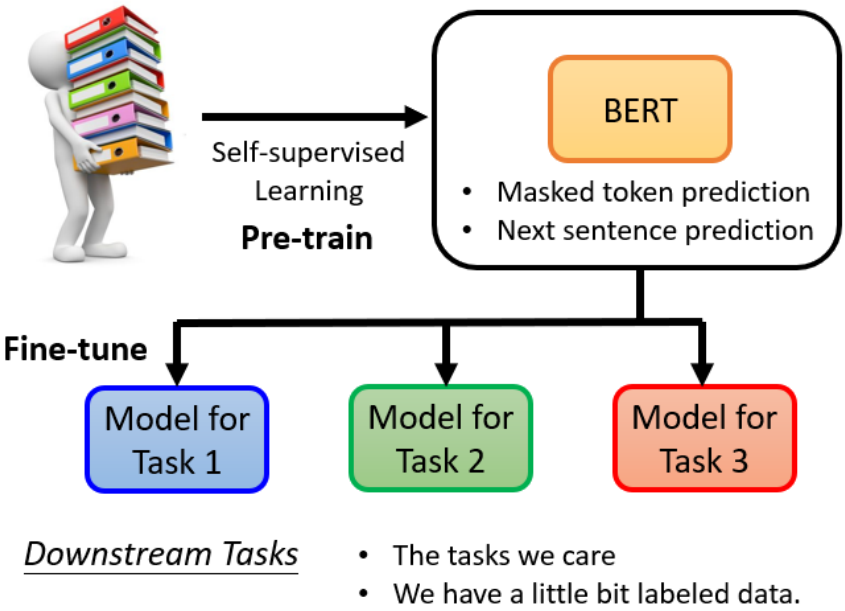
这个方法的主要思想是，我们最初挑选的两个句子可能是相连的。可能有两种可能性：要么句子1在句子2后面相连，要么句子2在句子1后面相连。有两种可能性，我们问BERT是哪一种。

也许因为这个任务更难，它似乎更有效。它被用在一个叫ALBERT的模型中，这是BERT的高级版本。由于ALBERT这个名字与爱因斯坦相似，我在幻灯片中放了一张爱因斯坦的图片。

当我们训练时，我们要求BERT学习两个任务。

- 一个是掩盖一些字符，具体来说是汉字，然后要求它填补缺失的字符。
- 另一个任务表明它能够预测两个句子是否有顺序关系。

所以总的来说，BERT它学会了如何填空。BERT的神奇之处在于，在你训练了一个填空的模型之后，它还可以**用于其他任务**。这些任务**不一定与填空有关**，也可能是完全不同的任务，但BERT仍然可以用于这些任务，这些任务是BERT实际使用的任务，它们被称为==Downstream Tasks==(下游任务)，以后我们将谈论一些 Downstream Tasks 的例子。



所谓的 "Downstream Tasks "是指，你真正关心的任务。但是，当我们想让BERT学习做这些任务时，我们仍然**需要一些标记的信息**。

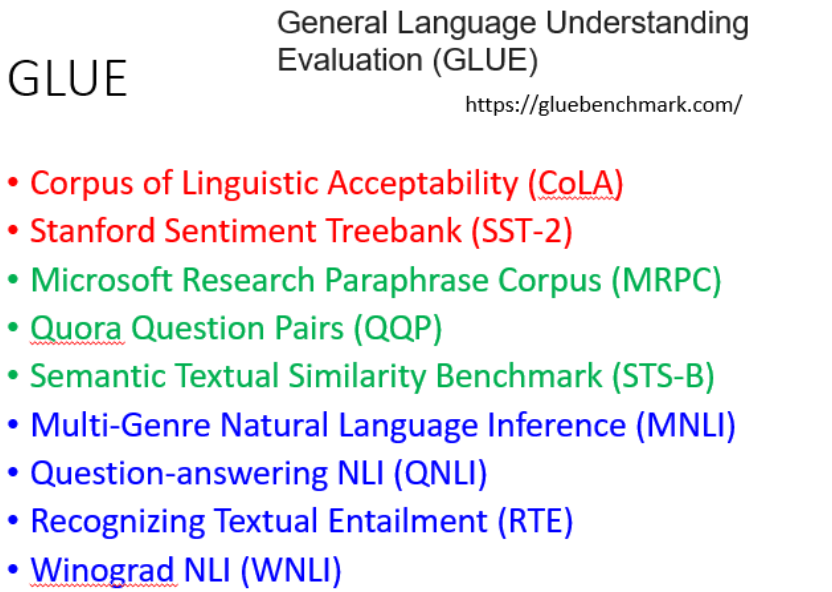
总之，BERT只是学习填空，但是，以后可以用来做各种你感兴趣的Downstream Tasks 。它就像胚胎中的干细胞,它有各种无限的潜力，虽然它还没有使用它的力量,它只能填空,但以后它有能力解决各种任务。我们只需要给它一点数据来激发它，然后它就能做到。

BERT分化成各种任务的功能细胞，被称为==Fine-tune==(微调)。所以，我们经常听到有人说，他对BERT进行了微调，也就是说他手上有一个BERT，他对这个BERT进行了微调，使它能够完成某种任务，与微调相反，在微调之前产生这个BERT的过程称为==预训练==。

所以，生成BERT的过程就是Self-supervised学习。但是，你也可以称之为预训练。如果你知道它是什么，你不应该在其他地方寻找Self-supervised学习的模型，直接应用在作业上。因为这些方法，往往带有令人难以置信的强大能力，这将使你要做的事情变得很无聊。

接下来其实还有一个BERT的作业。作业7是使用BERT。所以,在作业7中,,当然,你可以使用预训练的模型。这是你唯一可以使用预训练模型的作业,因为作业7是,微调BERT。所以，你当然要使用预训练的BERT，来进行微调。所以，只有在作业7中，你可以使用预训练的模型。

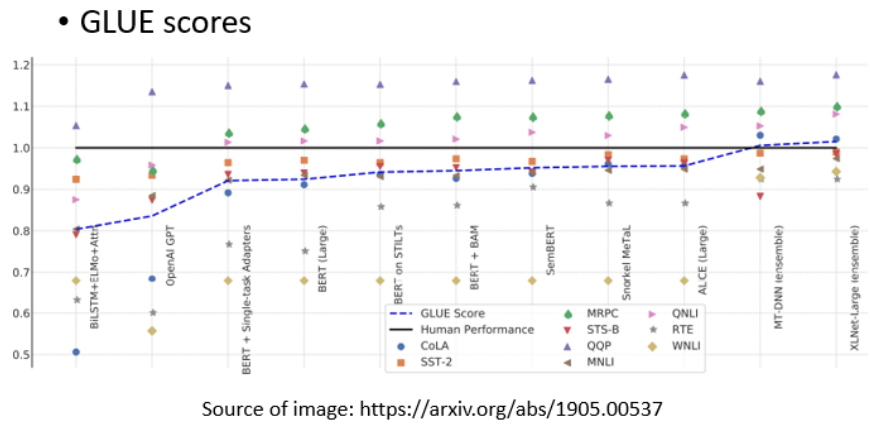
好的，在我们谈论如何微调BERT之前，我们应该先看看它的能力。今天，为了测试Self-supervised学习的能力，通常，你会在**多个任务上测试**它。因为我们刚才说，BERT就像一个胚胎干细胞，它要分化成各种任务的功能细胞，我们通常不会只在一个任务上测试它的能力，你会让这个BERT分化成各种任务的功能细胞，看看它在每个任务上的准确性，然后我们取其平均值，得到一个总分。这种不同任务的集合，，我们可以称之为任务集。任务集中最著名的基准被称为==GLUE==，它是General Language Understanding Evaluation的缩写。



GLUE also has Chinese version (https://www.cluebenchmarks.com/)

在GLUE中，总共有9个任务。一般来说，你想知道像BERT这样的模型是否被训练得很好。所以，你实际上会得到9个模型，用于9个单独的任务。你看看这**9个任务的平均准确率**，然后，你得到一个值。这个值代表这个Self-supervised模型的性能。

让我们看看BERT在GLUE上的性能。



Source of image: https://arxiv.org/abs/1905.00537

有了BERT，GLUE得分，也就是9个任务的平均得分，确实逐年增加。在这张图中，,横轴表示不同的模型，这里列出了，你可以发现，除了ELMO和GPT，其他的还有很多BERT，各种BERT。

黑色的线，表示**人类的工作**，也就是人类在这个任务上的准确度，那么，我们把这个当作1，这里每一个点代表一个任务，那么，你为什么要和人类的准确度进行比较呢?

人类的准确度是1，如果他们比人类好，这些点的值就会大于1，如果他们比人类差，这些点的值就会小于1，这是因为这些任务，其评价指标可能不是准确度。每个任务使用的评价指标是不同的，它可能不是准确度。如果我们只是比较它们的值，可能是没有意义的。所以，这里我们看的是人类之间的差异。

所以，你会发现，在原来的9个任务中，只有1个任务，机器可以比人类做得更好。随着越来越多的技术被提出，越来越多的,还有3个任务可以比人类做得更好。对于那些远不如人类的任务，,它们也在逐渐追赶。

蓝色曲线表示机器**GLUE得分的平均值**。还发现最近的一些强势模型，例如XLNET，甚至超过了人类。当然，这只是这些数据集的结果，并不意味着机器真的在总体上超过了人类。它**在这些数据集上超过了人类**。这意味着这些数据集并不能代表实际的表现，而且难度也不够大。

所以，在GLUE之后，有人做了Super GLUE。他们找到了更难的自然语言处理任务，让机器来解决。好了！展示这幅图的意义主要是告诉大家，有了BERT这样的技术，机器在自然语言处理方面的能力确实又向前迈进了一步。

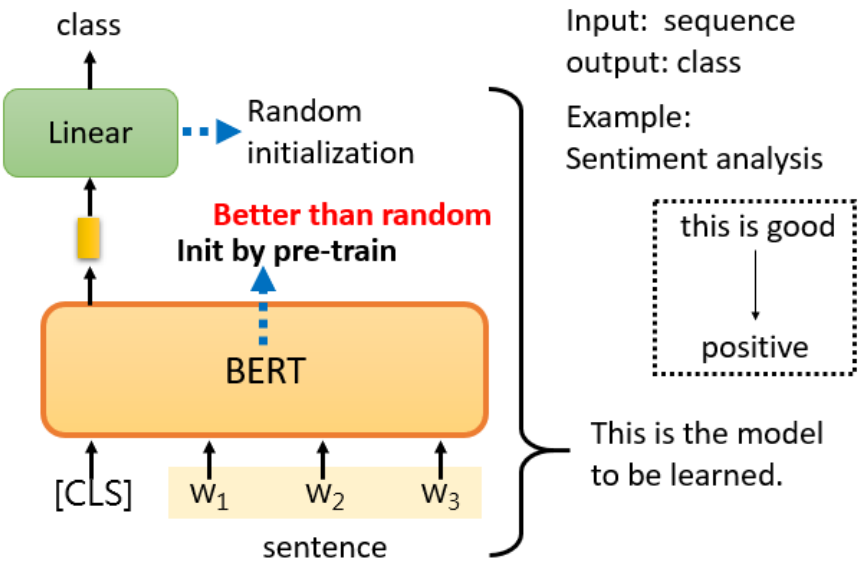
BERT到底是怎么用的呢？我们将给出4个关于BERT的应用案例，

How to use BERT

Case 1： Sentiment analysis

第一个案例是这样的，我们假设我们的Downstream Tasks 是输入一个序列，然后输出一个class，这是一个分类问题。

比如说Sentiment analysis情感分析，就是给机器一个句子，让它判断这个句子是正面的还是负面的。



对于BERT来说，它是如何解决情感分析的问题的?

你只要给它一个句子，也就是你想用它来判断情绪的句子，然后把**CLS标记放在这个句子的前面**，我刚才提到了CLS标记。我们把CLS标记放在前面，扔到BERT中，这4个输入实际上对应着4个输出。然后，我们**只看CLS的部分**。CLS在这里输出一个向量，我们对它进行Linear transform，也就是将它乘以一个Linear transform的矩阵，这里省略了Softmax。

然而，在实践中，你必须为你的Downstream Tasks 提供**标记数据**，换句话说，BERT没有办法从头开始解决情感分析问题，你仍然需要向BERT提供一些标记数据，你需要向它提供大量的句子，以及它们的正负标签，来训练这个BERT模型。

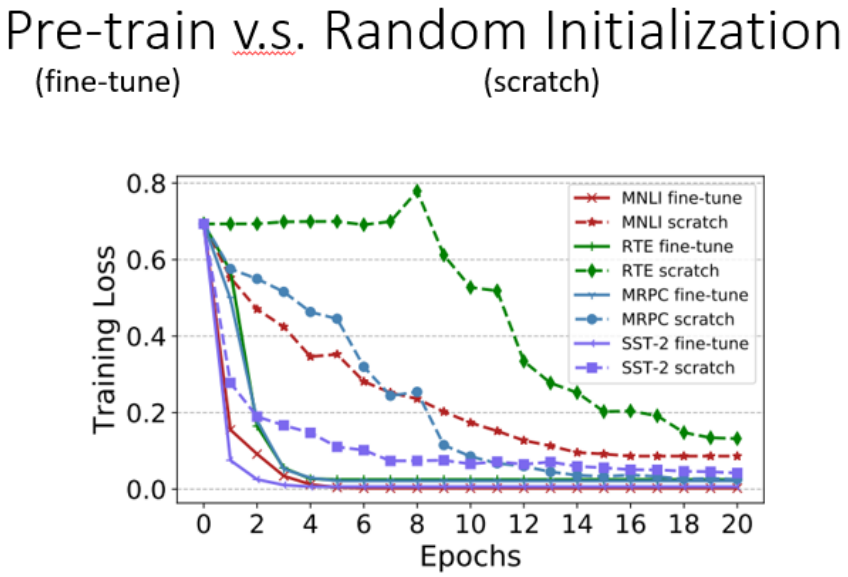
在训练的时候，Linear transform和BERT模型都是利用Gradient descent来更新参数的。

- Linear transform的参数是**随机初始化**的
- 而BERT的参数是由**学会填空的BERT初始化**的。

每次我们训练模型的时候，我们都要初始化参数，我们利用梯度下降来更新这些参数，然后尝试minimize loss，

例如，我们正在做情感分类，但是，我们现在有BERT。我们不必随机初始化所有的参数。,我们唯一随机初始化的部分是Linear这里。BERT的骨干是一个巨大的transformer的Encoder。这个网络的参数不是随机初始化的。把学过填空的BERT参数，放到这个地方的BERT中作为参数初始化。

我们为什么要这样做呢？为什么要用学过填空的BERT，再放到这里呢？最直观和最简单的原因是，它比随机初始化新参数的网络表现更好。当你把学会填空的BERT放在这里时，它将获得比随机初始化BERT更好的性能。



Source of image: https://arxiv.org/abs/1908.05620

在这里有篇文章中有一个例子。横轴是训练周期，纵轴是训练损失，到目前为止，大家对这种图一定很熟悉，随着训练的进行，损失当然会越来越低，这个图最有趣的地方是，有各种任务。我们不会解释这些任务的细节，我只想说明有各种任务。

- "fine-tune"是指模型被用于预训练，这是网络的BERT部分。该部分的参数是由学习到的BERT的参数来初始化的，以填补空白。
- scratch表示整个模型，包括BERT和Encoder部分都是随机初始化的。

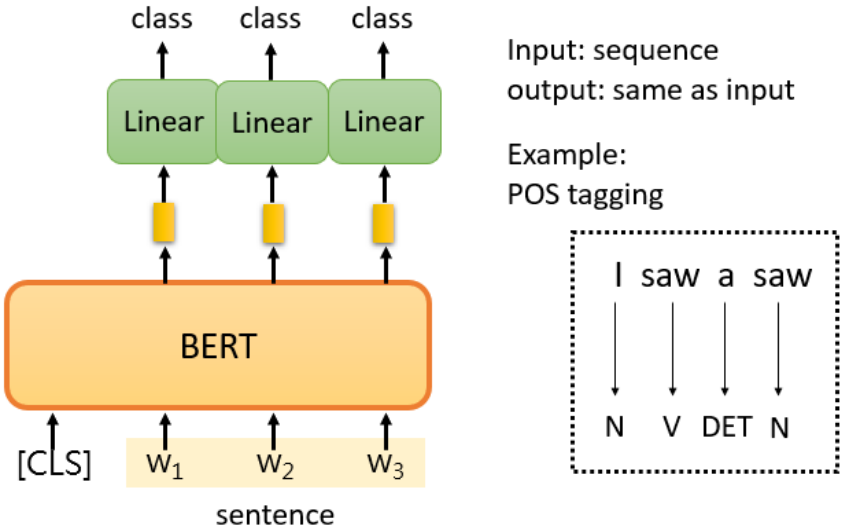
首先，在训练网络时，scratch与用学习填空的BERT初始化的网络相比，**损失下降得比较慢**，最后，用随机初始化参数的网络的损失仍然高于用学习填空的BERT初始化的参数。

- 当你进行Self-supervised学习时，你使用了大量的**无标记数据**。
- 另外，Downstream Tasks 需要少量的**标记数据**。

所谓的 "半监督 "是指，你有大量的无标签数据和少量的有标签数据，这种情况被称为 "半监督"，所以使用BERT的整个过程是连续应用Pre-Train和Fine-Tune，它可以被视为一种半监督方法。

Case 2： POS tagging

第二个案例是，输入一个序列，然后输出另一个序列，而输入和输出的长度是一样的。我们在讲Self-Attention的时候，也举了类似的例子。 例如，==POS tagging==。



POS tagging的意思是词性标记。你给机器一个句子，它必须告诉你这个句子中每个词的词性，即使这个词是相同的，也可能有不同的词性。

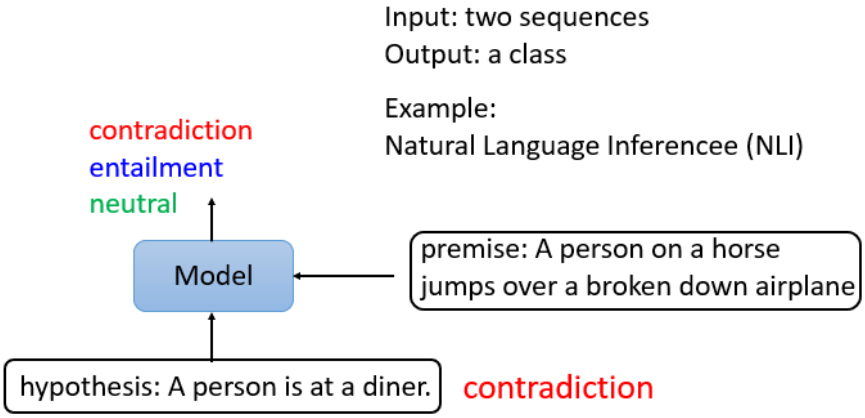
你只需向BERT输入一个句子。之后，对于这个句子中的每一个标记，它是一个中文单词，有一个代表这个单词的相应向量。然后，这些向量会依次通过Linear transform和Softmax层。最后，网络会预测给定单词所属的类别，例如，它的词性。

当然，类别取决于你的任务，如果你的任务不同，相应的类别也会不同。接下来你要做的事情和案例1完全一样。换句话说，你需要有一些标记的数据。这仍然是一个典型的分类问题。唯一不同的是，BERT部分，即网络的Encoder部分，其参数不是随机初始化的。在**预训练过程中，它已经找到了不错的参数**。

当然，我们在这里展示的例子属于自然语言处理。但是，你可以把这些例子改成**其他任务**，例如，你可以把它们改成语音任务，或者改成计算机视觉任务。我在Self-supervised Learning一节中提到，语音、文本和图像都可以表示为一排向量。虽然下面的例子是文字，但这项技术**不仅限于处理文字**，它还可以用于其他任务，如计算机视觉。

Case 3：Natural Language Inference

在案例3中，模型输入两个句子，输出一个类别。好了，第三个案例以两个句子为输入，输出一个类别，什么样的任务采取这样的输入和输出？ 最常见的是Natural Language Inference ，它的缩写是NLI



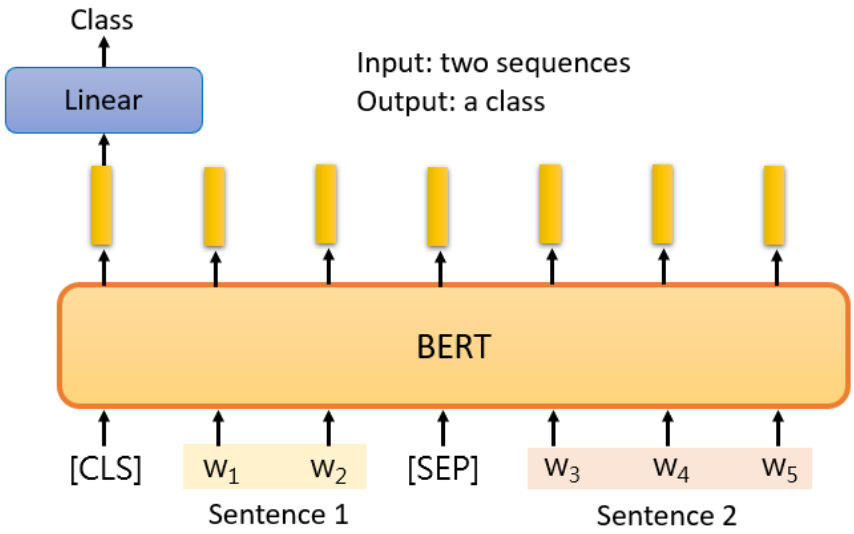
机器要做的是判断，是否有可能**从前提中推断出假设**。这个前提与这个假设相矛盾吗？或者说它们不是相矛盾的句子？

在这个例子中，我们的前提是，一个人骑着马，然后他跳过一架破飞机，这听起来很奇怪。但这个句子实际上是这样的。这是一个基准语料库中的例子。

这里的**假设**是，这个人在一个餐馆。所以**推论**说这是一个**矛盾**。

所以机器要做的是，把两个句子作为输入，并输出这两个句子之间的关系。这种任务很常见。它可以用在哪里呢？例如，舆情分析。给定一篇文章，下面有一个评论，这个消息是同意这篇文章，还是反对这篇文章？该模型想要预测的是每条评论的位置。事实上，有很多应用程序接收两个句子，并输出一个类别。

BERT是如何解决这个问题的？你只要给它两个句子，我们在这两个句子之间放一个**特殊的标记**，并在最开始放CLS标记。



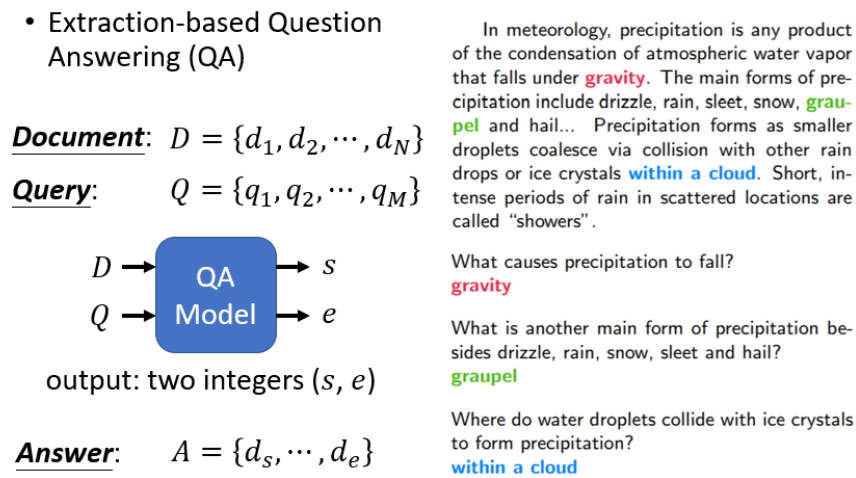
这个序列是BERT的输入。但我们只把CLS标记作为Linear transform的输入。它决定这两个输入句子的类别。对于NLI，你必须问，这两个句子是否是矛盾的。它是用一些**预先训练好的权重来初始化的**。

Case 4：Extraction-based Question Answering (QA)

如果你不理解前面的案例，就忘掉它们。这第四个案例，就是我们在作业7中要做的。作业7是一个问题回答系统。也就是说，在机器读完一篇文章后，你问它一个问题，它将给你一个答案。

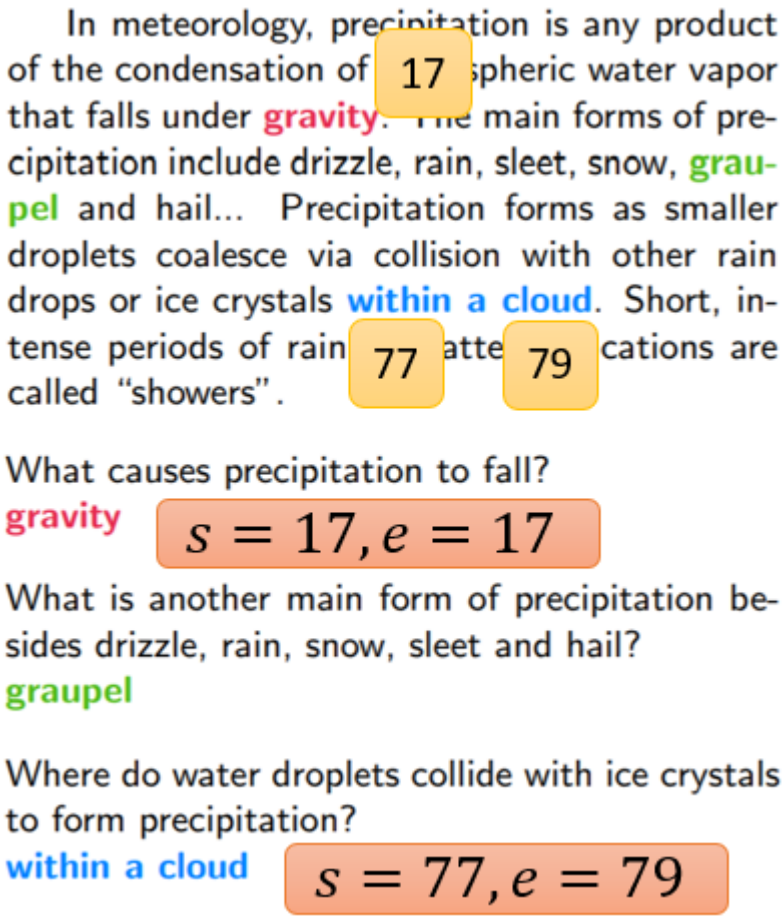
但是，这里的问题和答案稍有限制。这是Extraction-based的QA。也就是说，我们假设**答案必须出现在文章**中。答案必须是文章中的一个片段。

在这个任务中，一个输入序列包含**一篇文章**和**一个问题**，文章和问题都是一个**序列**。对于中文来说，每个d代表一个汉字，每个q代表一个汉字。你把d和q放入QA模型中，我们希望它输出**两个正整数s和e**。根据这两个正整数，我们可以直接从文章中**截取一段**，它就是答案。这个片段就是正确的答案。



这听起来很疯狂，但是，这是现在使用的一个相当标准的方法。六年前，当我第一次听说这个机制可以解决QA任务时，我简直不敢相信。但是，无论如何，这是今天一个非常普遍的方法。

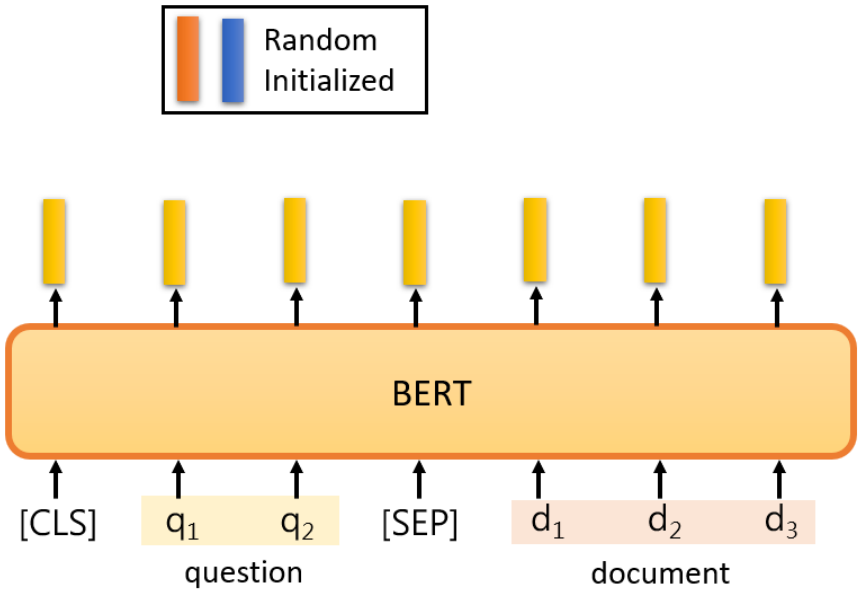
好吧，如果你仍然不明白我在说什么，更具体地说，这里有一个问题和一篇文章，正确答案是 "gravity"。机器如何输出正确答案？



你的保证模型应该输出，s等于17，e等于17，来表示gravity。因为它是整篇文章中的第17个词，所以s等于17，e等于17，意味着输出第17个词作为答案。

或者举另一个例子，答案是，"within a cloud"，这是文章中的第77至79个词。你的模型要做的是，输出77和79这两个正整数，那么文章中从第77个词到第79个词的分割应该是最终的答案。这就是作业7要你做的。

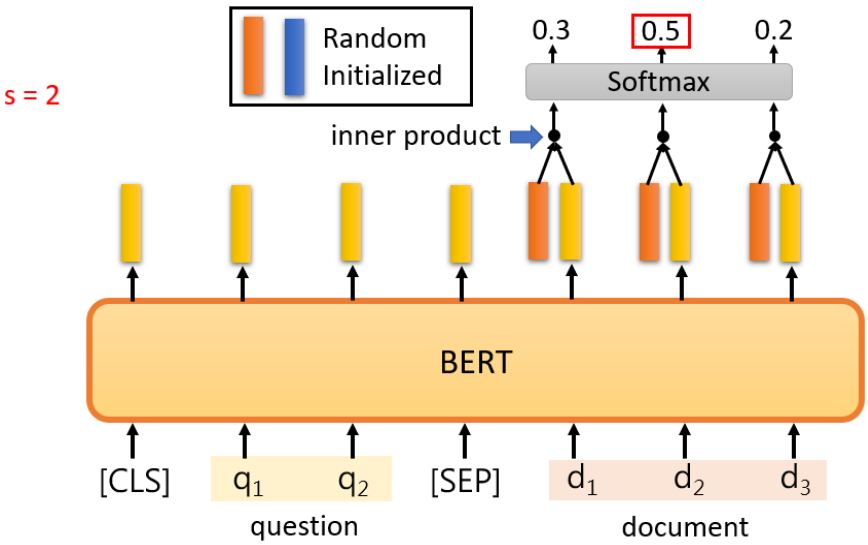
当然，我们不是从头开始训练QA模型，为了训练这个QA模型，我们**使用BERT预训练的模型**。



这个解决方案是这样的。对于BERT来说，你必须向它展示一个问题，一篇文章，以及在问题和文章之间的一个特殊标记，然后我们在开头放一个CLS标记。

在这个任务中，你唯一需要**从头训练**的只有**两个向量**。"从头训练 "是指**随机初始化**。这里我们用橙色向量和蓝色向量来表示，这两个向量的长度与BERT的输出相同。

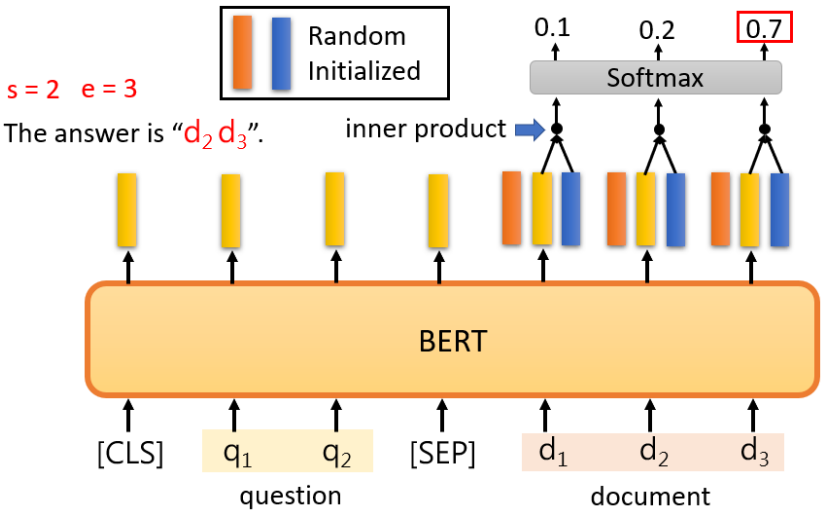
假设BERT的输出是768维的向量，这两个向量也是768维的向量。那么，如何使用这两个向量?



- 首先,计算这个**橙色向量**和那些与文件相对应的**输出向量的内积**,由于有3个代表文章的标记,它将输出三个向量,计算这三个向量与橙色向量的内积,你将得到三个值,然后将它们通过softmax函数,你将得到另外三个值。

  这个内积**和attention很相似**，你可以把橙色部分看成是query，黄色部分看成是key，这是一个attention，那么我们应该尝试找到分数最大的位置，就是这里，橙色向量和d2的内积，如果这是最大值，s应该等于2，你输出的起始位置应该是2

- 蓝色部分做的是完全一样的事情。



蓝色部分代表答案的终点，我们计算这个蓝色向量与文章对应的黄色向量的内积，然后，我们在这里也使用softmax，最后，找到最大值，如果第三个值是最大的，e应该是3，正确答案是d2和d3。

因为答案必须在文章中，如果**答案不在文章中，你就不能使用这个技巧**。这就是一个QA模型需要做的。注意，这两个向量是随机初始化的,而BERT是通过它预先训练的权重初始化的。

Q&A

Q：BERT的输入长度有限制吗?

A：理论上，没有。在现实中，是的，在理论上，因为BERT模型，是一个transformer的Encoder，所以它可以输入很长的序列，只要你必须能够做Self-Attention，但Self-Attention的计算复杂性是非常高的。所以你会发现，在实践中，BERT实际上不能输入太长的序列，你最多可以输入512长度的序列，如果你输入一个512长度的序列，Self-Attention在中间就要产生512乘以512大小的Attention Metric，那么你可能会被计算所淹没。所以实际上它的长度不是无限的。在助教的程序中，已经为大家处理了这个问题。我们限制了BERT的输入长度，而且用一篇文章来训练需要很长的时间。然后每次，我们只取其中的一段进行训练。我们不会将整篇文章输入BERT。因为你想要的距离太长了，你的训练会有问题。

Q：“它与填空题有什么关系?

A：",哇，这个问题很好。,你会认为这个填空题只是一个填空题。但我要在这里做一个Q&A。,这两件事之间有什么关系呢？这里先卖个关子，待会会试着回答你。

Training BERT is challenging!

BERT是这样一个著名的模型，它可以做任何事情，那么你可能会认为BERT，在预训练中，它只是填空题，但是，**你自己真的不能把它训练起来**。

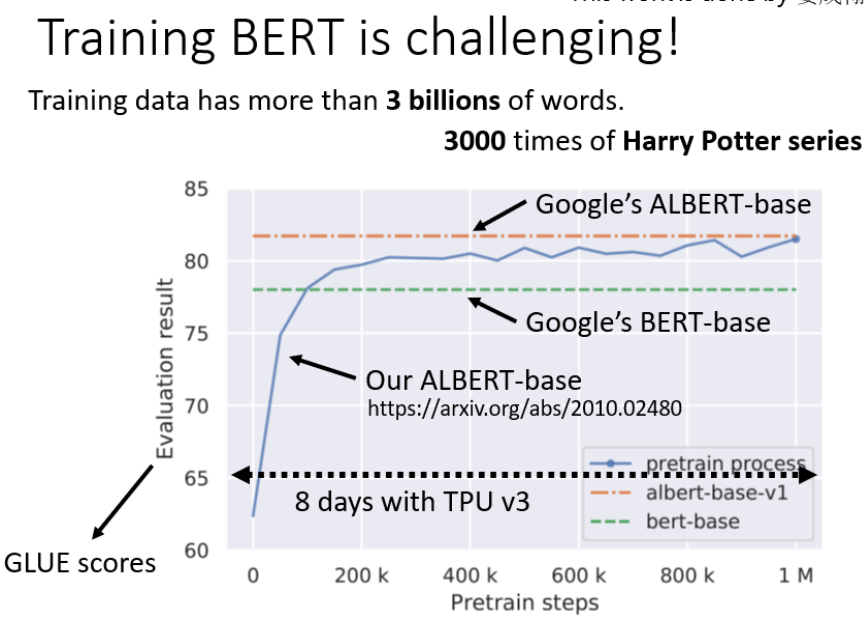
首先，谷歌最早的BERT,它使用的**数据规模已经很大**了,它的数据中包含了30亿个词汇,30亿个词汇有多少？,是《哈利波特全集》的3000倍。,《哈利波特全集》大约是100万个词汇。,那么谷歌在训练BERT时，最早的BERT，它使用的数据量是《哈利波特全集》的3000倍。

所以你处理起来会比较痛苦,**更痛苦的是训练过程**,为什么我知道训练过程是痛苦的呢,因为我们实验室有一个学生,他其实是助教之一,他自己试着训练一个BERT,他觉得他不能重现谷歌的结果,好,那么在这个图中,纵轴代表GLUE分数,我们刚才讲到GLUE,对吧?有9个任务，平均有9个任务，,平均分数就叫GLUE分数,好的，那么蓝线就是，谷歌原来的BERT的GLUE分数。

那么我们的目标其实不是实现BERT，我们的目标是实现ALBERT。ALBERT是一个高级版本，是橙色的线，蓝线是我们自己训练的ALBERT，但是我们实际训练的不是最大版本，BERT有一个base版本和一个large版本。对于大版本，我们很难自己训练它，所以我们尝试用最小的版本来训练，看它是否与谷歌的结果相同。

你可能会说这30亿个数据，30亿个词似乎有很多数据。实际上，因为它是无标签数据，所以你只是从互联网上整理了一堆文本，有相同的信息量。所以你要**爬上这个级别的信息并不难，难的是训练过程**

台達電產學合作計畫研究成果
This work is done by 姜成翰



好的，这个横轴是训练过程，参数更新多少次，大约一百万次的更新，需要多长时间，用TPU运行8天，所以你的TPU要运行8天，如果你在Colab上做，这个至少要运行200天，你甚至可能到明年才能得到结果。

所以，你真的很难自己训练这种BERT模型。幸运的是，作业只是对它进行微调。你可以在Colab上进行微调，在Colab上微调BERT只需要半小时到一小时。但是，如果你想从头开始训练它，也就是说，训练它做填空题，这将需要大量的时间，而且，你不能在Colab上自己完成它。

BERT Embryology (胚胎學)

谷歌已经训练了BERT，而且这些Pre-Train模型是公开的，我们自己训练一个，结果和谷歌的BERT差不多，这有什么意义呢?

其实是想建立==BERT胚胎学==。"BERT胚胎学是什么意思？"



我们知道在BERT的训练过程中需要非常大的计算资源，所以我们想知道有没有可能，**节省这些计算资源**？有没有可能让它训练得更快？,要知道如何让它训练得更快，也许我们可以从观察它的训练过程开始。

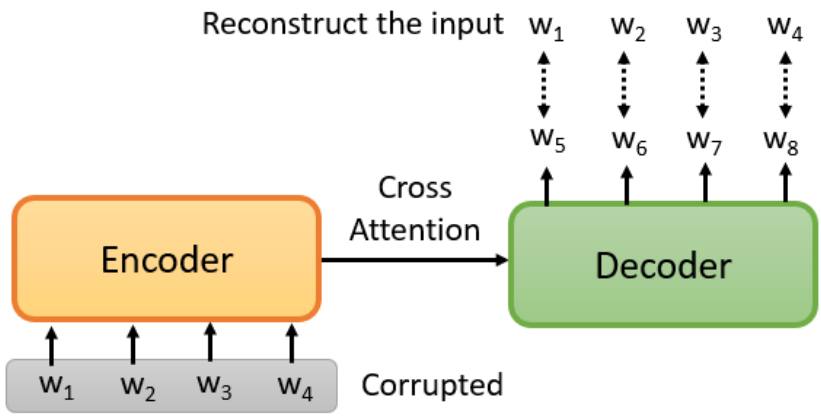
过去没有人观察过BERT的训练过程。因为在谷歌的论文中，他们只是告诉你，我有这个BERT。然后它在各种任务中做得很好。

BERT在学习填空的过程中，学到了什么？"它在这个过程中何时学会填动词？什么时候学会填名词？ 什么时候学会填代词？ 没有人研究过这个问题。

所以我们自己训练BERT后，可以观察到BERT什么时候学会填什么词汇，它是如何提高填空能力的？ 好了，细节不是这门课的重点，所以我不在这里讲了。我把论文的链接https://arxiv.org/abs/2010.02480放在这里，供大家参考。不过可以提前爆冷一下就是：事实和你直观想象的不一样。

Pre-training a seq2seq model

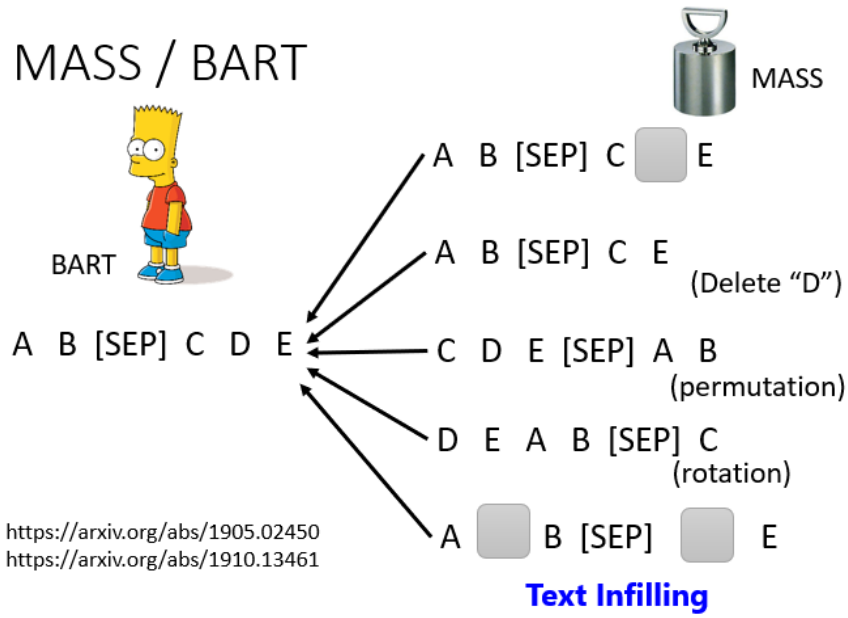
我们补充一点，上述的任务都不包括，Seq2Seq模型，如果我们要解决，Seq2Seq模型呢？BERT只是一个预训练Encoder，有没有办法预训练Seq2Seq模型的Decoder?



有，你就说我有一个Seq2Seq模型，有一个transformer，还有一个Encoder和Decoder。输入是一串句子，输出是一串句子，中间用Cross Attention连接起来，然后你故意在Encoder的输入上做一些**干扰来破坏它**，我以后会具体告诉你我说的"破坏 "是什么意思

Encoder看到的是被破坏的结果，那么Decoder应该输出句子被破坏前的结果，训练这个模型实际上是预训练一个Seq2Seq模型。

有一篇论文叫MASS



在MASS中，它说破坏的方法是，就像BERT做的那样，只要遮住一些地方就可以了，然后有各种方法来破坏它，比如，删除一些词，打乱词的顺序，旋转词的顺序。或者插入一个MASK，再去掉一些词。总之，有各种方法。在破坏了输入的句子之后，它可以通过Seq2Seq模型来恢复它。

有一篇论文叫BART，它就是用了所有这些方法。我发现用所有可能的方法更好，它可以比MASS更好。我想问一个问题，为什么不是芝麻街的人物?

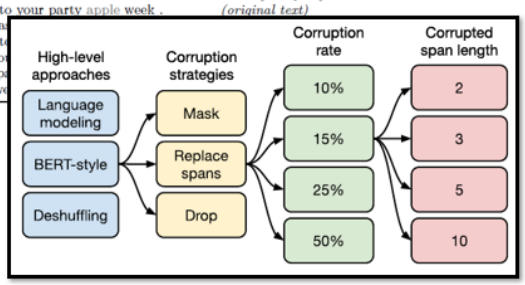
你可能会问,有那么多的mask方法,哪种方法更好呢?也许你想自己做一些实验来试试,让我告诉你,你不需要做,谷歌为你做的,有一篇论文叫T5。

T5 – Comparison

- Transfer Text-to-Text Transformer (T5)
- Colossal Clean Crawled Corpus (C4)



Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . la	
I.i.d. noise, mask tokens	Thank you <M> <M> me t	
I.i.d. noise, replace spans	Thank you <X> me to yo	
I.i.d. noise, drop tokens	Thank you me to your p	
Random spans	Thank you <X> to <Y> w	



T5的全称是Transfer Text-To-Text Transformer，有五个T，所以叫T5。在这个T5里面，它只是做了各种尝试，它做了你能想象的所有组合。这篇论文有67页，你可以回去读一下，看看结论。

T5是在一个语料库上训练的，叫 "Colossal Clean Crawled Corpus"，对于这个数据集，Colossal就是巨无霸，就是非常巨大的意思，它叫C4，你用C4来训练T5，大家都是命名高手。这个命名非常强大，这个C4有多大?

C4是一个公共数据集，你可以下载它，它是公共的，但是它的原始文件大小是7TB，你可以下载它，但是你不知道把它保存在哪里，加载之后，你可以通过脚本做预处理，由谷歌提供。这个脚本有一个文件，我看到它在网站上发布了，语料库网站上的文件说，用一个GPU做预处理需要355天，你可以下载它，但你在预处理时有问题。

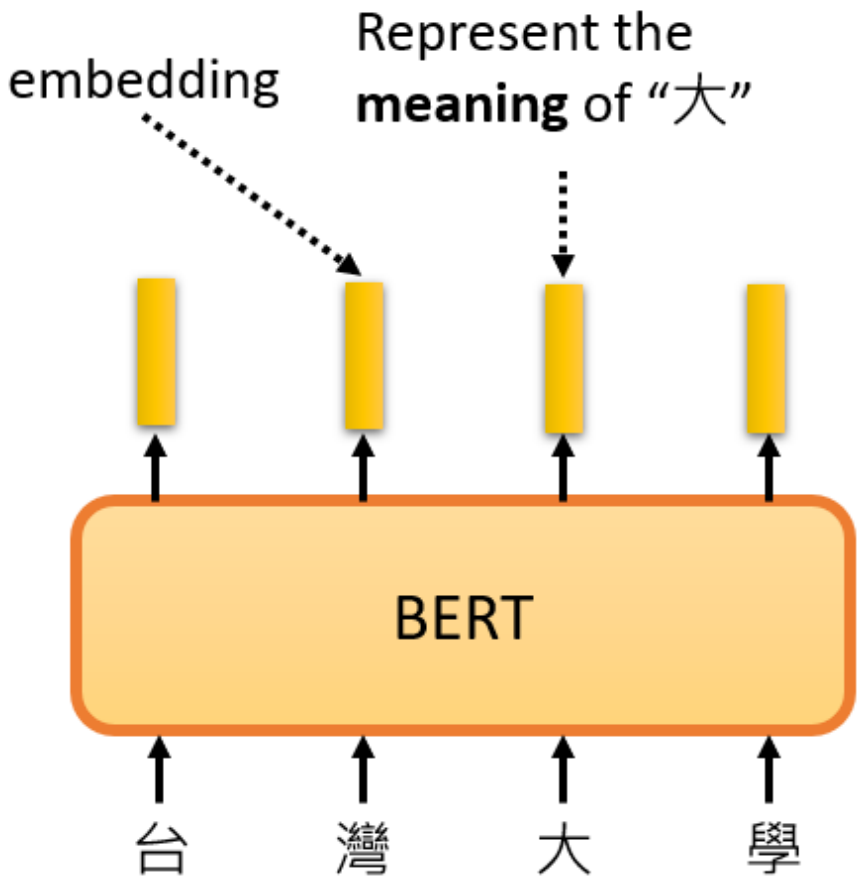
所以，你可以发现，在深度学习中，数据量和模型都很惊人。

BERT P2_Fun Facts about BERT

Why does BERT work?

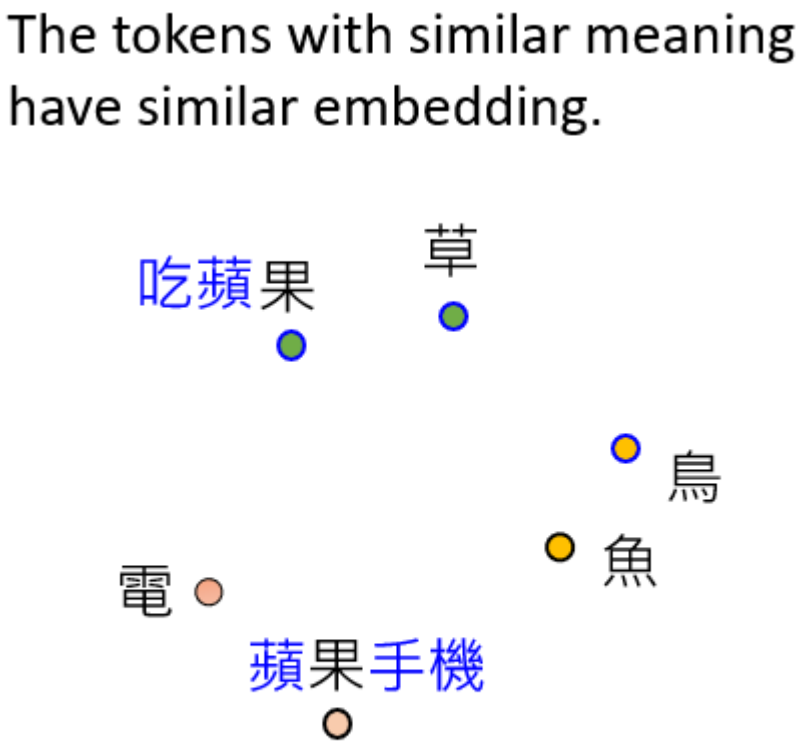
"为什么BERT有用？"

最常见的解释是，当输入一串文本时，每个文本都有一个对应的向量。对于这个向量，我们称之为embedding。



它的特别之处在于，这些向量代表了**输入词**的**含义**。例如，模型输入 "台湾大学"（国立台湾大学），输出4个向量。这4个向量分别代表 "台"、"湾"、"大 "和 "学"

更具体地说，如果你把这些词所对应的向量画出来，或者计算它们之间的**距离**

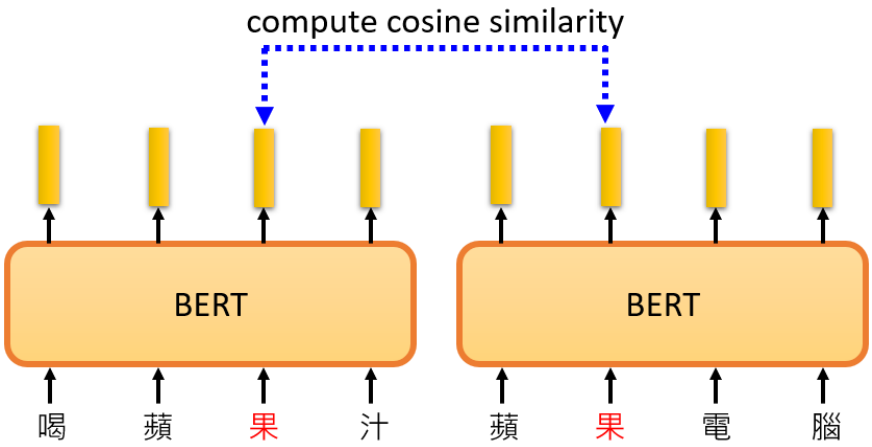


你会发现，**意思比较相似的词**，它们的**向量比较接近**。例如，水果和草都是植物，它们的向量比较接近。但这是一个假的例子，我以后会给你看一个真正的例子。"鸟 "和 "鱼 "是动物，所以它们可能更接近。

你可能会问，中文有歧义，其实不仅是中文，很多语言都有歧义，**BERT可以考虑上下文**，所以，同一个词，比如说 "苹果"，它的上下文和另一个 "苹果 "不同，它们的向量也不会相同。

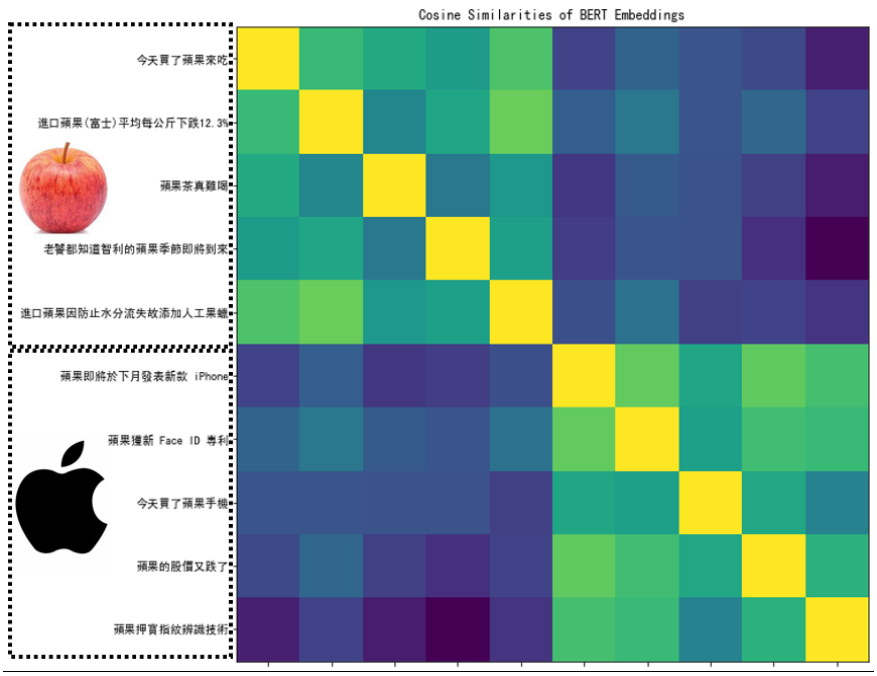
水果 "苹果 "和手机 "苹果 "都是 "苹果"，但根据上下文，它们的**含义是不同**的。所以，它的**向量和相应的embedding会有很大不同**。水果 "苹果 "可能更接近于 "草"，手机 "苹果 "可能更接近于 "电"。

现在我们看一个真实的例子。假设我们现在考虑 "苹果 "这个词，我们会收集很多有 "苹果 "这个词的句子，比如 "喝苹果汁"、"苹果Macbook "等等。然后，我们把这些句子放入BERT中。



接下来，我们将计算 "苹果 "一词的相应embedding。输入 "喝苹果汁"，得到一个 "苹果 "的向量。为什么不一样呢？在Encoder中存在Self-Attention，所以根据 "苹果 "一词的不同语境，得到的向量会有所不同。接下来，我们计算这些结果之间的 ==cosine similarity==，即计算它们的相似度。

结果是这样的，这里有10个句子



- 前5个句子中的 "苹果 "代表**可食用**的苹果。例如，第一句是 "我今天买了苹果吃"，第二句是 "进口富士苹果平均每公斤多少钱"，第三句是 "苹果茶很难喝"，第四句是 "智利苹果的季节来了"，第五句是 "关于进口苹果的事情"，这五个句子都有 "苹果 "一词，
- 后面五个句子也有 "苹果 "一词，但提到的是**苹果公司**的苹果。例如，"苹果即将在下个月发布新款iPhone"，"苹果获得新专利"，"我今天买了一部苹果手机"，"苹果股价下跌"，"苹果押注指纹识别技术"，共有十个 "苹果"

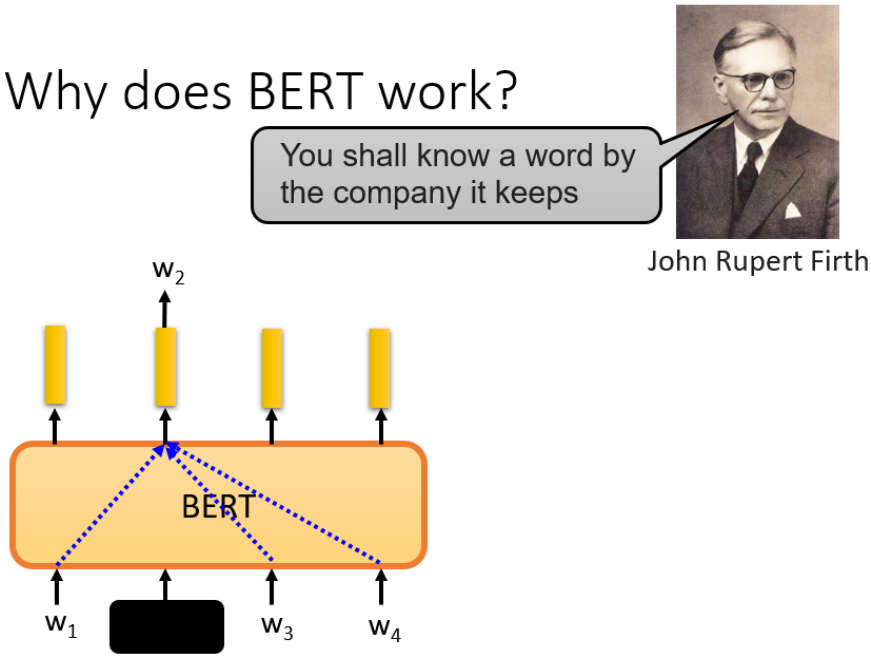
计算每一对之间的相似度，得到一个10×10的矩阵。**相似度越高，这个颜色就越浅**。所以，自己和自己之间的相似度一定是最大的，自己和别人之间的相似度一定是更小的。

但前五个 "苹果 "和后五个 "苹果 "之间的相似度相对较低。

BERT知道，前五个 "苹果 "是指可食用的苹果，所以它们比较接近。最后五个 "苹果 "指的是苹果公司，所以它们比较接近。所以**BERT知道，上下两堆 "苹果 "的含义不同**。

BERT的这些向量是输出向量，每个向量代表该词的含义。BERT在填空的过程中已经学会了每个汉字的意思。",也许它真的理解了中文，对它来说，汉字不再是毫无关联的，既然它理解了中文，它就可以在接下来的任务中做得更好。

那么接下来你可能会问，"为什么BERT有如此神奇的能力？",为什么.....,为什么它能输出代表输入词含义的向量？ 这里，约翰-鲁伯特-弗斯，一位60年代的语言学家，提出了一个假说。他说，要知道一个词的意思，我们需要看它的 "**Company**"，也就是经常和它**一起出现的词汇**，也就是它的**上下文**。

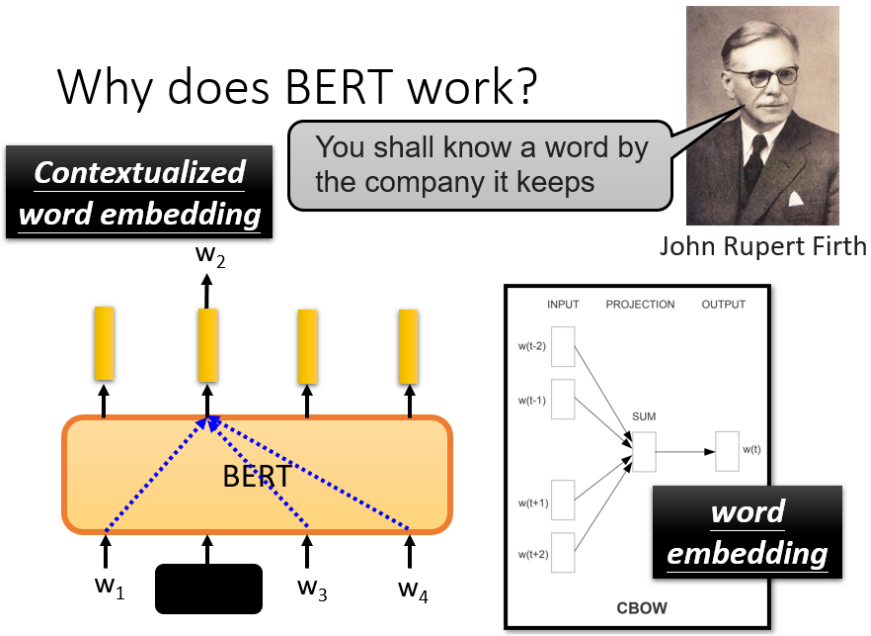


一个词的意思，取决于它的上下文

- 所以以苹果（apple）中的果字为例。如果它经常与 "吃"、"树 "等一起出现，那么它可能指的是可食用的苹果。
- 如果它经常与电子、专利、股票价格等一起出现，那么它可能指的是苹果公司。

当我们训练BERT时，我们给它w1、w2、w3和w4，我们覆盖w2，并告诉它预测w2，而它就是从上下文中提取信息来预测w2。所以这个向量是其上下文信息的精华，可以用来预测w2是什么。

这样的想法在BERT之前已经存在了。在word embedding中，有一种技术叫做CBOW。



CBOW所做的，与BERT完全一样。做一个空白，并要求它预测空白处的内容。这个CBOW，这个word embedding技术，可以给每个词汇一个向量，代表这个词汇的意义。

CBOW是一个非常简单的模型，它使用两个变换，是一个**非常简单的模型**，有人会问，"为什么它只使用两个变换？"，"它可以更复杂吗？"，CBOW的作者，Thomas Mikolov，曾经来到台湾。当时我在上课的时候，经常有人问我，为什么CBOW只用线性，为什么不用深度学习，我问过Thomas Mikolov这个问题，他说可以用深度学习，但是之所以选择线性模型，一个简单的模型，最大的担心，其实是**算力问题**。当时的计算能力和现在不在一个数量级上，可能是2016年的时候，几年前的技术也不在一个数量级上，当时要训练一个非常大的模型还是比较困难的，所以他选择了一个比较简单的模型。

今天，当你使用**BERT**的时候，就相当于一个**深度版本的CBOW**，你可以做更复杂的事情，而且BERT还可以根据不同的语境，从同一个词汇产生不同的embedding。因为它是一个考虑到语境的高级版本的词embedding，BERT也被称为Contextualized embedding，这些由BERT提取的向量或embedding被称为Contextualized embedding，希望大家能接受这个答案。

但是，这个答案，它真的是真的吗？这是你在文献中听到最多的答案。当你和别人讨论BERT时，这是大多数人都会告诉你的理由。它真的是真的吗？这里有一个难以理解的，由我们实验室的一个学生做的实验。实验是这样的：我们应用为文本训练的BERT对蛋白质、DNA链和音乐进行分类。

https://arxiv.org/abs/2103.07162
This work is done by 高瑋聰

Why does BERT work?

- Applying BERT to **protein, DNA, music classification**



class	DNA sequence
EI	CCAGCTGCATCACAGGAGGCCAGC(
EI	AGACCCGCCGGGAGGCGGAGGACC(
IE	AACGTGGCCTCCTTGTGCCCTTCCC(
IE	CCACTCAGCCAGGCCCTTCTTCTCCT
IE	CCTGATCTGGGTCTCCCCTCCCACCC
IE	AGCCCTCAACCCTTCTGTCTCACCCT
IE	CCACTCAGCCAGGCCCTTCTTCTCCT
N	CTGTGTTCACCACATCAAGCGCCGG(
N	GTGTTACCGAGGGCATTTCTAACAG1
N	TCTGAGCTCTGCATTTGTCTATTCTCC

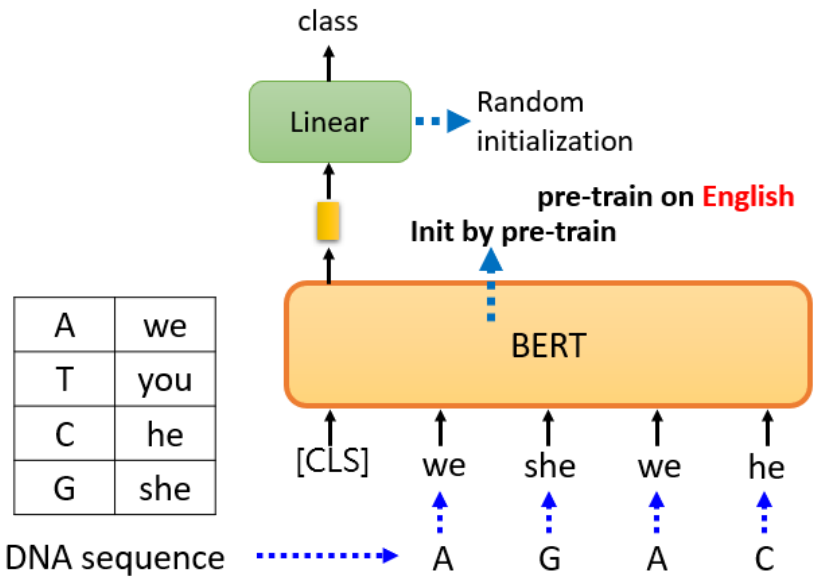
让我们以DNA链的分类为例。DNA是一系列的脱氧核团核酸，有四种，分别用A、T、C和G表示，所以一条DNA链是这样的。

你可能会问，"EI IE和N代表什么？"不要在意细节，我也不知道，总之，这是一个分类问题。只要用训练数据和标记数据来训练它，就可以了。

神奇的部分来了，DNA可以用ATCG来表示，现在，我们要用BERT来对DNA进行分类

Why does BERT work?

https://arxiv.org/abs/2103.07162
This work is done by 高瑋聰



A	we
T	you
C	he
G	she

例如，"A "是 "we"，"T "是 "you"，"C "是 "he"，"G "是 "she"。对应的词并不重要，你可以随机生成。"A "可以对应任何词汇，"T"、"C "和 "G "也可以，这并不重要，对结果影响很小。只是这串文字无法理解。

例如，"AGAC "变成了 "we she we he"，不知道它在说什么。

然后，把它扔进一个一般的BERT，用CLS标记，一个输出向量，一个Linear transform，对它进行分类。只是分类到了DNA类,我不知道他们是什么意思。

和以前一样，Linear transform使用随机初始化，而BERT是通过预训练模型初始化的。但用于初始化的模型，是学习填空的模型。它已经学会了英语填空。

你可能会认为,这个实验完全是无稽之谈。如果我们把一个DNA序列预处理成一个无意义的序列,那么BERT的目的是什么？大家都知道,BERT可以分析一个有效句子的语义,你怎么能给它一个无法理解的句子呢？做这个实验的意义是什么？

蛋白质有三种分类，那么蛋白质是由氨基酸组成的，有十种氨基酸，只要给每个氨基酸一个随机的词汇，那么DNA是一组ATCG，音乐也是一组音符，给它每个音符一个词汇，然后，把它作为一个文章分类问题来做。

- Applying BERT to **protein, DNA, music classification**

	Protein			DNA				Music
	localization	stability	fluorescence	H3	H4	H3K9ac	Splice	composer
specific	69.0	76.0	63.0	87.3	87.3	79.1	94.1	-
BERT	64.8	74.5	63.7	83.0	86.2	78.3	97.5	55.2
re-emb	63.3	75.4	37.3	78.5	83.7	76.3	95.6	55.2
rand	58.6	65.8	27.5	75.6	66.5	72.8	95	36



你会发现，如果你不使用BERT，你得到的结果是蓝色部分，如果你使用BERT，你得到的结果是红色部分，这实际上更好，你们大多数人现在一定很困惑。

这个实验只能用神奇来形容，没有人知道它为什么有效，而且目前还没有很好的解释，我之所以要谈这个实验，是想告诉你们，要了解BERT的力量，还有很多工作要做。

我并不是要否认BERT能够分析句子的含义这一事实。从embedding中，我们清楚地观察到，BERT知道每个词的含义，它能找出含义相似的词和不相似的词。但正如我想指出的那样，即使你给它一个无意义的句子，它仍然可以很好地对句子进行分类。

所以，**也许它的力量并不完全来自于对实际文章的理解**。也许还有其他原因。例如，也许，BERT只是一套更好的初始参数。也许这与语义不一定有关。也许这套初始参数，只是在训练大型模型时更好。

是这样吗？这个问题**需要进一步研究**来回答。我之所以要讲这个实验，是想让大家知道，我们目前使用的模型往往是非常新的，需要进一步的研究，以便我们了解它的能力。

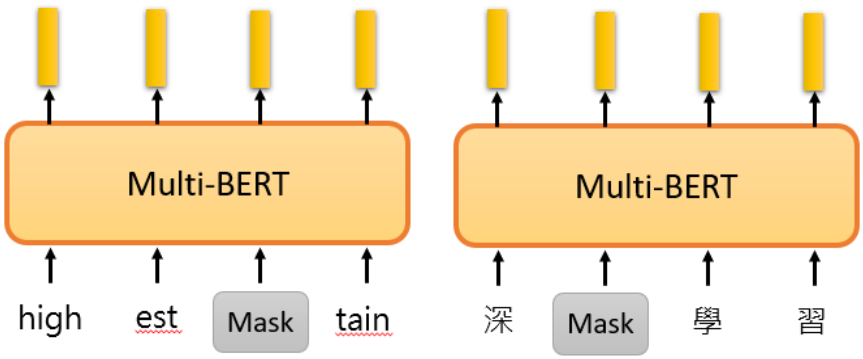
你今天学到的关于BERT的知识，只是沧海一粟。我会把一些视频的链接放在这里。



如果你想了解更多关于BERT的知识，你可以参考这些链接。你的作业不需要它，，这学期剩下的时间也不需要。我只想告诉你，BERT还有很多其他的变种。

Multi-lingual BERT

接下来，我要讲的是，一种叫做Multi-lingual BERT的BERT。Multi-lingual BERT有什么神奇之处?

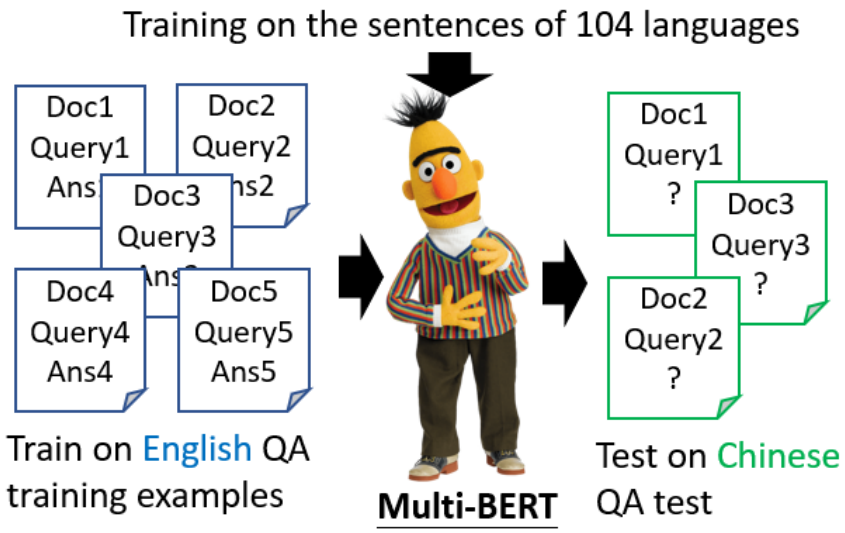


Training a BERT model by many different languages.

它是由很多语言来训练的，比如中文、英文、德文、法文等等，用填空题来训练BERT，这就是Multi-lingual BERT的训练方式。

Zero-shot Reading Comprehension

google训练了一个Multi-lingual BERT，它能够做这104种语言的填空题。神奇的地方来了，如果你用**英文**问答**数据**训练它，它就会自动学习如何做**中文问答**



我不知道你是否完全理解我的意思，所以这里有一个真实的实验例子。

- English: SQuAD, Chinese: DRCD

Model	Pre-train	Fine-tune	Test	EM	F1
QANet	none	Chinese	Chinese	66.1	78.1
BERT	Chinese	Chinese		82.0	89.1
	104 languages	Chinese		81.2	88.7
		English		63.3	78.8
		Chinese + English		82.6	90.1

F1 score of Human performance is 93.30%

This work is done by 劉記良、許宗嫄
https://arxiv.org/abs/1909.09587

这是一些训练数据。他们用SQuAD进行fine-tune。这是一个英文Q&A数据集。中文数据集是由台达电发布的，叫DRCD。这个数据集也是我们在作业中要用到的数据集。

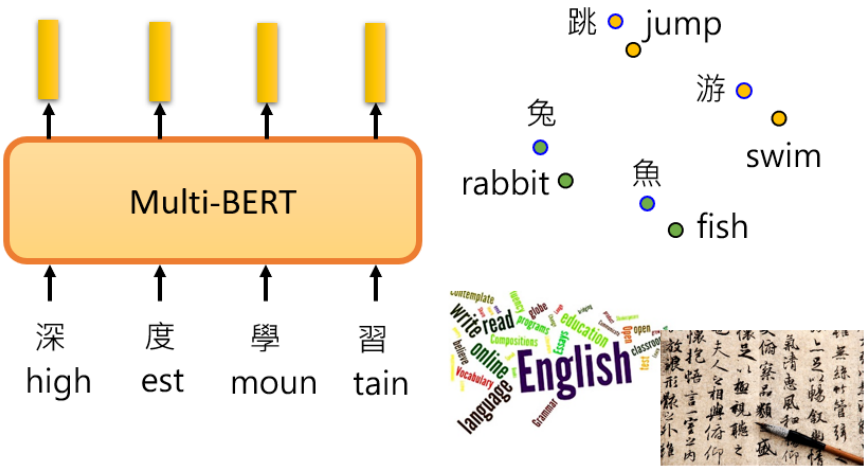
在BERT提出之前，效果并不好。在BERT之前，最强的模型是QANet。它的正确率只有……，嗯，我是说F1得分，而不是准确率，但你可以暂时把它看成是准确率或正确率。

如果我们允许用中文填空题进行预训练，然后用中文Q&A数据进行微调，那么它在中文Q&A测试集上的正确率达到89%。因此，其表现是相当令人印象深刻的。

神奇的是，如果我们把一个Multi-lingual的BERT，用英文Q&A数据进行微调，它仍然可以回答中文Q&A问题，并且有78%的正确率，这几乎与QANet的准确性相同。它从未接受过中文和英文之间的翻译训练，也从未阅读过中文Q&A的数据收集。，它在没有任何准备的情况下参加了这个中文Q&A测试，尽管它从未见过中文测试，但不知为何，它能回答这些问题。

Cross-lingual Alignment?

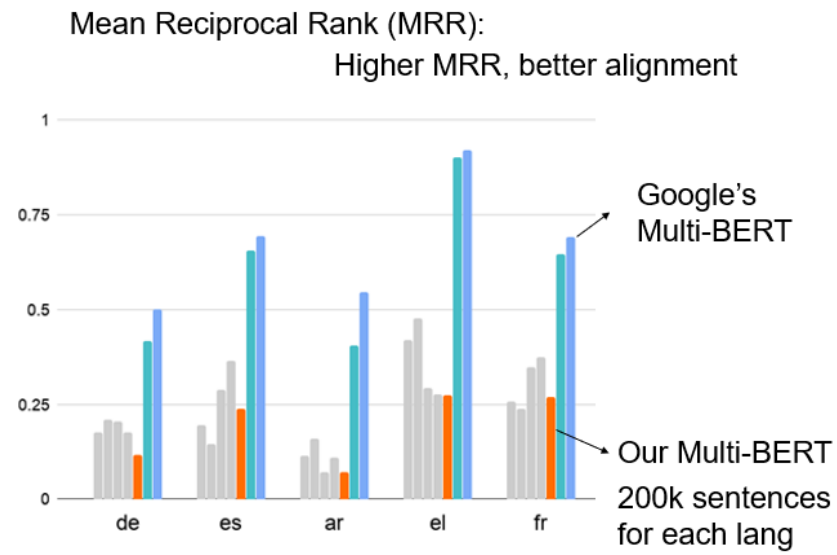
你们中的一些人可能会说："它在预训练中读过104种语言，104种语言中的一种是中文，是吗？ 如果是，这并不奇怪。"但是在预训练中，学习的目标是填空。它只能用中文填空。有了这些知识，再加上做英文问答的能力，不知不觉中，它就自动学会了做中文问答。



听起来很神奇，那么BERT是怎么做到的呢？一个简单的解释是：也许对于多语言的BERT来说，**不同的语言并没有那么大的差异**。无论你用中文还是英文显示，对于具有相同含义的单词，它们的embedding都很接近。汉语中的 "跳 "与英语中的 "jump "接近，汉语中的 "鱼 "与英语中的 "fish "接近，汉语中的 "游 "与英语中的 "swim "接近，也许在学习过程中它已经自动学会了。

它是可以被验证的。我们实际上做了一些验证。验证的标准被称为Mean Reciprocal Rank，缩写为MRR。我们在这里不做详细说明。你只需要知道，**MRR的值越高，不同embedding之间的Alignment就越好**。

更好的Alignment意味着，具有相同含义但来自不同语言的词将被转化为更接近的向量。如果MRR高，那么具有相同含义但来自不同语言的词的向量就更接近。



https://arxiv.org/abs/2010.10938

投影片來源: 許宗嫄同學碩士口試投影片

How about 1000k?

这条深蓝色的线是谷歌发布的104种语言的Multi-lingual BERT的MRR，它的值非常高，这说明不同语言之间没有太大的差别。Multi-lingual BERT只看意思，不同语言对它没有太大的差别。

橙色这条是我们试图自己训练Multi-lingual BERT。我们使用的**数据较少**，每种语言只使用了20万个句子。数据较少。我们自我训练的模型结果并不好。我们不知道为什么我们的Multi-lingual BERT不能将不同的语言统一起来。似乎它不能学习那些在不同语言中具有相同含义的符号，它们应该具有相同的含义。这个问题困扰了我们很长时间。

为什么我们要做这个实验？为什么我们要自己训练Multi-lingual BERT？因为我们想了解，是什么让Multi-lingual BERT。我们想设置不同的参数，不同的向量，看看哪个向量会影响Multi-lingual BERT。

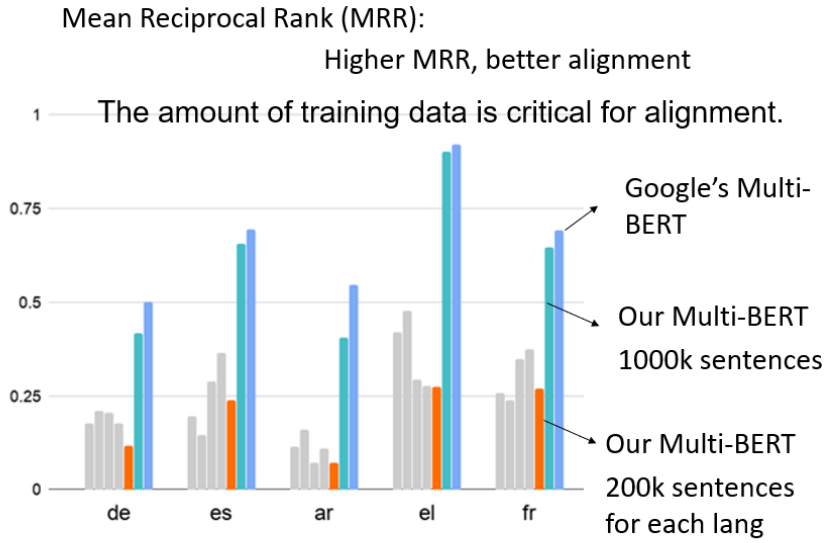
但是我们发现，对于我们的Multi-lingual BERT来说，无论你如何调整参数，它就是不能达到Multi-lingual的效果，它就是不能达到Alignment的效果。我们把数据量**增加了五倍**，看看能不能达到Alignment的效果。在做这个实验之前，大家都有点抵触，大家都觉得有点害怕，因为训练时间要比原来的长五倍。

训练了两天后，什么也没发生，损失甚至不能减少，就在我们要放弃的时候，损失突然下降了



用了8个V100来训练，我们的实验室也没有8个V100，是在NCHC（国家高性能计算中心）的机器上运行的，训练了两天后，损失没有下降，似乎失败了。当我们要放弃的时候，损失下降了。

这是某个学生在Facebook上发的帖子，我在这里引用它来告诉你，我当时心里的感叹。整个实验，必须运行一个多星期，才能把它学好，每一种语言1000K的数据。



https://arxiv.org/abs/2010.10938

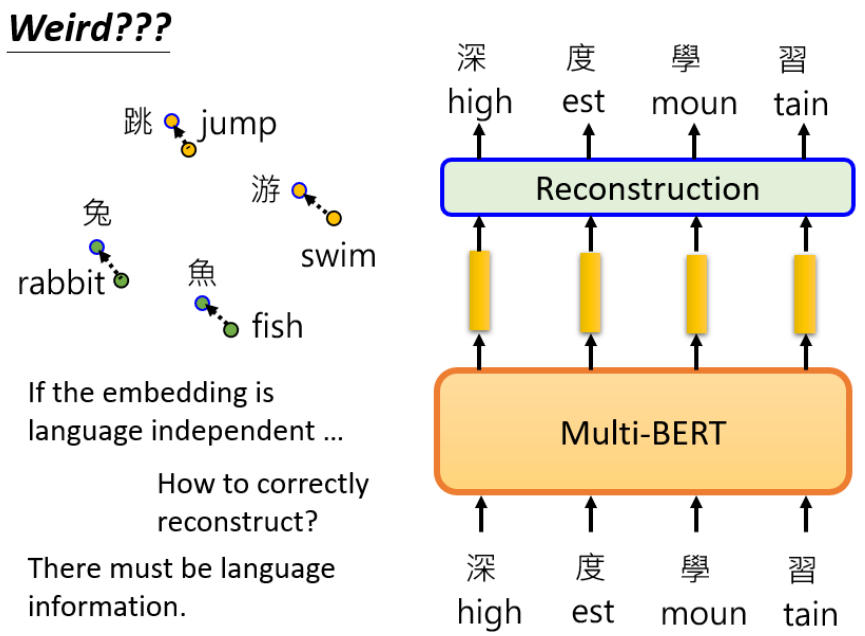
投影片來源: 許宗嫄同學碩士口試投影片

所以看起来，**数据量是一个非常关键的因素**，关系到能否成功地将不同的语言排列在一起。所以有时候，神奇的是，很多问题或很多现象，只有在有足够的数据量时才会显现出来。它可以在A语言的QA上进行训练，然后直接转移到B语言上，从来没有人说过这一点

这是过去几年才出现的，一个可能的原因是，过去没有足够的数据，现在有足够的数据，现在有大量的计算资源，所以这个现象现在有可能被观察到。

最后一个神奇的实验，我觉得这件事很奇怪

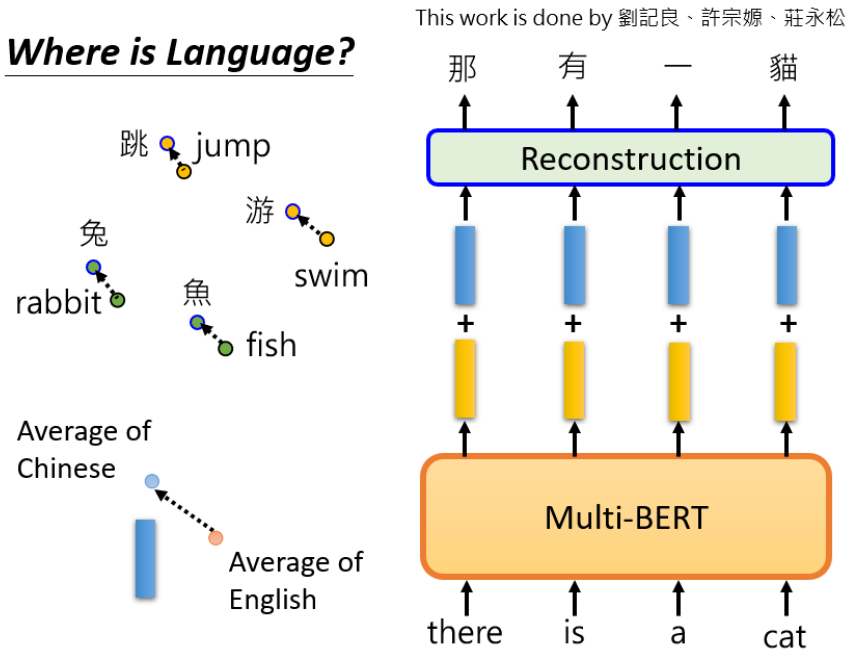
https://arxiv.org/abs/2010.10041



你说BERT可以把不同语言中含义相同的符号放在一起，使它们的向量接近。但是，当训练多语言的BERT时，如果给它英语，它可以用英语填空，如果给它中文，它可以用中文填空，它不会混在一起

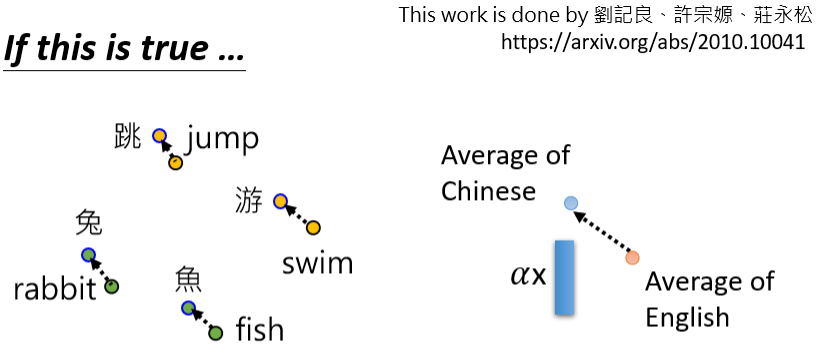
那么，如果不同语言之间没有区别，怎么可能只用英语标记来填英语句子呢？为什么它不会用中文符号填空呢？它就是不填，这说明它知道语言的信息也是不同的，那些不同语言的符号毕竟还是不同的，它并没有完全抹去语言信息，所以我想出了一个研究课题，我们来找找，语言信息在哪里。

后来我们发现，语言信息并没有隐藏得很深。一个学生发现，我们把所有**英语单词**的embedding，放到多语言的BERT中，**取embedding的平均值**，我们对**中文单词**也做**同样的事情**。在这里，我们给Multi-lingual BERT一个英语句子，并得到它的embedding。我们在embedding中**加上**这个**蓝色的向量**，这就是**英语和汉语之间的差距**。



这些向量，从Multi-lingual BERT的角度来看，变成了汉语。有了这个神奇的东西，你可以做一个奇妙的无监督翻译。

例如，你给BERT看这个中文句子。



Input (en)	The girl that can help me is all the way across town. There is no one who can help me.
Ground Truth (zh)	能帮助我的女孩在小镇的另一边。没有人能帮助我。。
en→zh, $\alpha = 1$	. 孩，can 来我是all the way across 市。。There 是无人人can help 我。
en→zh, $\alpha = 2$	. 孩的的家我是这个人的市。。他是他人人的到我。
en→zh, $\alpha = 3$	。，的的的他是的个的的，。： 他是他人，的。他。

Unsupervised token-level translation ☺

这个中文句子是，"能帮助我的小女孩在小镇的另一边，，没人能够帮助我"，现在我们把这个句子扔到Multi-lingual BERT中。

然后我们取出Multi-lingual BERT中的一个层，它不需要是最后一层，可以是任何一层。我们拿出某一层，给它一个embedding，加上这个蓝色的向量。对它来说，这个句子马上就从中文变成了英文。

在向BERT输入英文后，通过在中间加一个**蓝色的向量来转换隐藏**层，转眼间，中文就出来了。"没有人可以帮助我"，变成了 "是（是）没有人（没有人）可以帮助我（我）"，"我 "变成了 "我"，"没有人 "变成了 "没有人"，所以它在某种程度上可以做无监督的标记级翻译，尽管它并不完美，神奇的是，Multi-lingual的BERT仍然保留了语义信息。

BERT P3_GPT3

除了BERT以外,还有下一个,也是鼎鼎有名的模型,就是==GPT==系列的模型



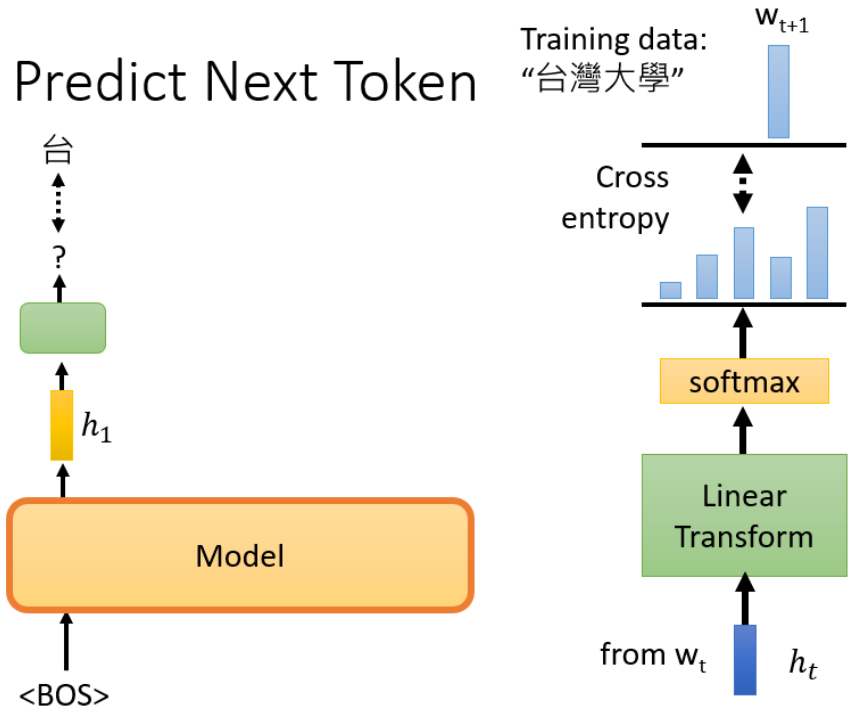
BERT做的是填空题,GPT就是改一下我们现在在,self-supervised learning的时候,要模型做的任务

Predict Next Token

GPT要做的任务是,**预测接下来,会出现的token是什麼**

举例来说,假设你的训练资料裡面,有一个句子是台湾大学,那GPT拿到这一笔训练资料的时候,它做的事情是这样

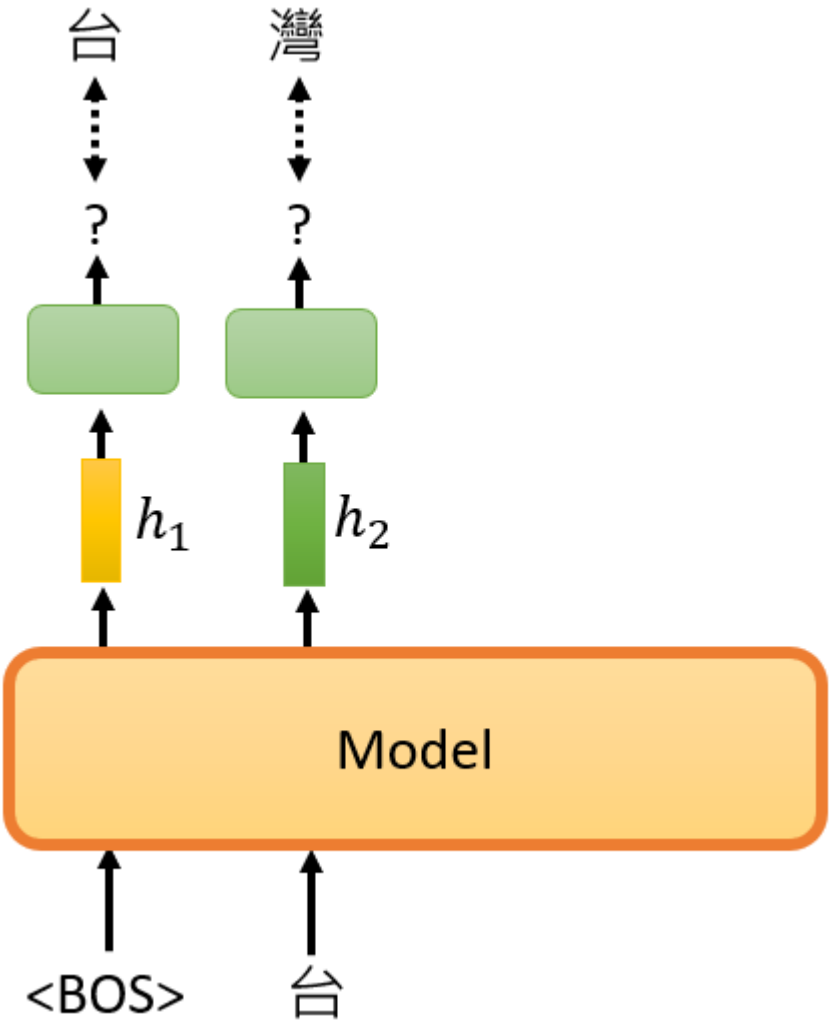
你给它BOS这个token,然后GPT output一个embedding,然后接下来,你用这个embedding去预测下一个,应该出现的token是什麼



那在这个句子裡面,根据这笔训练资料,下一个应该出现的token是"台",所以你要训练你的模型,根据第一个token,根据BOS给你的embedding,那它要输出"台"这个token

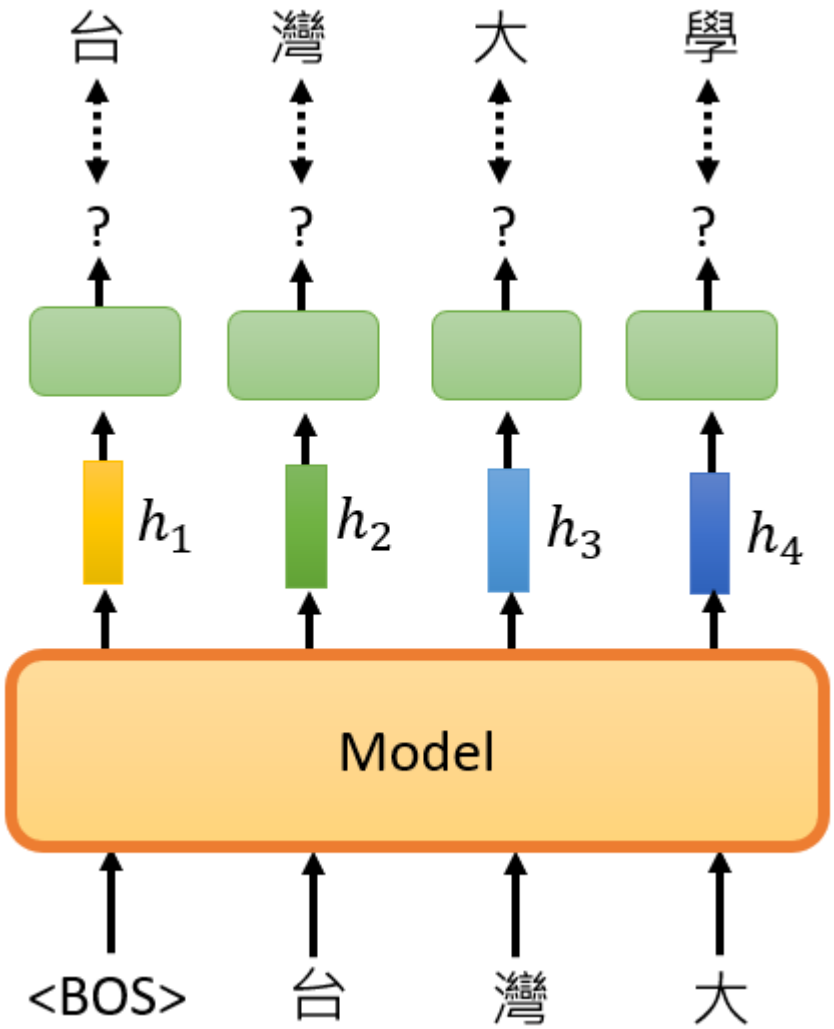
这个部分,详细来看就是这样,你有一个embedding,这边用h来表示,然后通过一个Linear Transform,再通过一个softmax,得到一个distribution,跟一般你做分类的问题是一样的,接下来,你希望你output的distribution,跟正确答案的Cross entropy,越小越好,也就是你要去预测,下一个出现的token是什麼

好那接下来要做的事情,就是**以此类推**了,你给你的GPT,BOS跟"台",它產生embedding,接下来它会预测,下一个出现的token是什麼,那你告诉它说,下一个应该出现的token,是"湾"



好 再反覆继续下去,你给它BOS "台"跟"湾",然后预测下一个应该出现的token,它应该要预测"大"

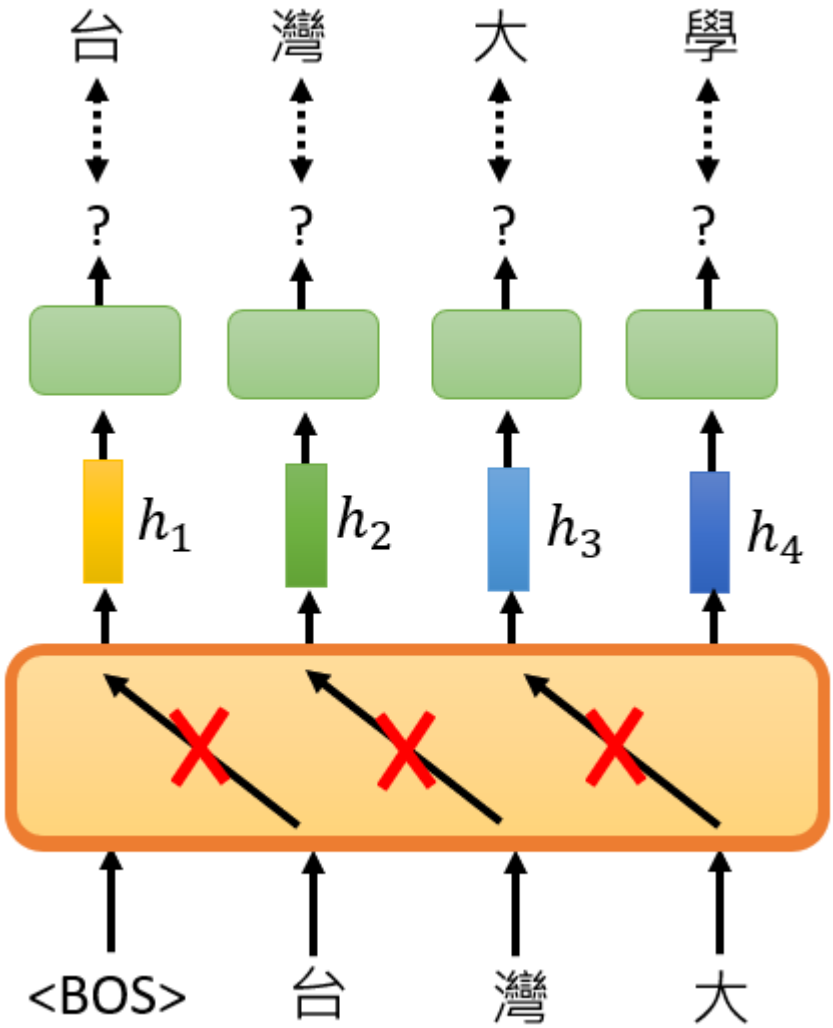
你给它"台"跟"湾"跟"大",接下来,下一个应该出现的token是"学"



那这边呢,是指拿一笔资料 一个句子,来给GPT训练,当然实际上你不会只用一笔句子,你会用成千上万个句子,来训练这个模型,然后就这样子说完了

它厉害的地方就是,用了很多资料,训了一个异常巨大的模型

那这边有一个小小的,应该要跟大家说的地方,是说这个GPT的模型,它像是一个transformer的decoder,不过拿掉BOS的attention这个部分,也就是说,你会做那个**mask的attention**



就是你现在在预测给BOS,预测台的时候,你不会看到接下来出现的词汇,给它台要预测湾的时候,你不会看到接下来要输入的词汇,以此类推 这个就是GPT

那这个GPT最知名的就是,因為GPT可以预测下一个token,那所以它有**生成的能力**,你可以让它不断地预测下一个token,產生完整的文章,所以我每次提到GPT的时候,它的形象都是一隻独角兽



GPT系列最知名的一个例子,就是用GPT写了一篇,跟独角兽有关的新闻,因為他放一个假新闻,然后那个假新闻裡面说,在安地斯山脉发现独角兽等等,一个活灵活现的假新闻

為了让你更清楚了解,GPT运作起来是什麼样子,那这个线上有一个demo的网页,叫做talk to transformer,就是有人把一个比较小的,不是那个最大的GPT的模型,不是public available的,有人把比较小的GPT模型放在线上,让你可以输入一个句子,让它会把接下来的其餘的内容,把它补完

How to use GPT?

怎麼把它用在downstream 的任务上呢,举例来说,怎麼把它用在question answering,或者是其他的,跟人类语言处理有关的任务上呢

GPT用的想法跟BERT不一样,其实我要强调一下,GPT也可以跟BERT用一样的做法

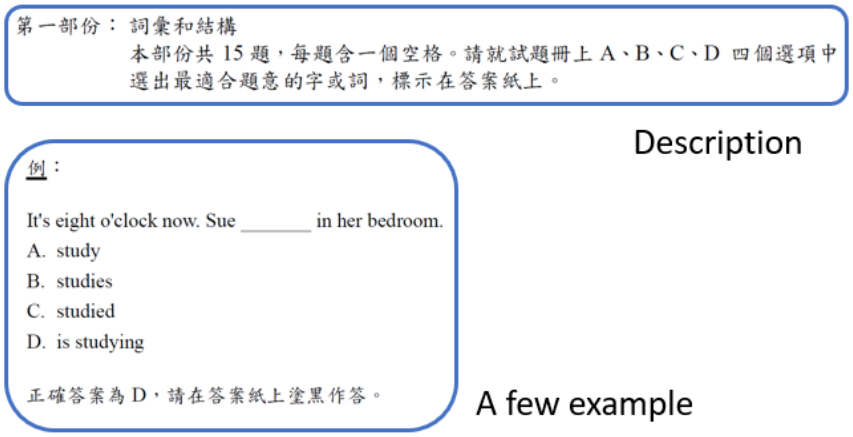
在使用BERT时，把BERT model 拿出来,后面接一个简单的linear的classifier,那你就可以做很多事情,你也可以把GPT拿出来,接一个简单的classifier,我相信也是会有效

但是在GPT的论文中,它没有这样做,它有一个更狂的想法,為什麼会有更狂的想法呢,因為首先就是,BERT那一招BERT用过了嘛,所以总不能再用一样的东西,这样写paper就没有人觉得厉害了,然后再来就是,**GPT这个模型,也许真的太大了**,大到连fine tune可能都有困难

我们在用BERT的时候,你要把BERT模型,后面接一个linear classifier,然后BERT也是你的,要train的model的一部分,所以它的参数也是要调的,所以在刚才助教公告的,BERT相关的作业裡面,你还是需要花一点时间来training,虽然助教说你大概20分鐘,就可以train完了,因為你并不是要train一个,完整的BERT的模型,BERT的模型在之

前,在做这个填空题的时候,已经训练得差不多了,你只需要微调它就好了,但是微调还是要花时间的,也许GPT实在是太过巨大,巨大到要微调它,要train一个epoch,可能都有困难,所以GPT系列,有一个更狂的使用方式

这个更狂的使用方式**和人类更接近**,你想想看假设你去考,譬如说托福的听力测验,你是怎麼去考



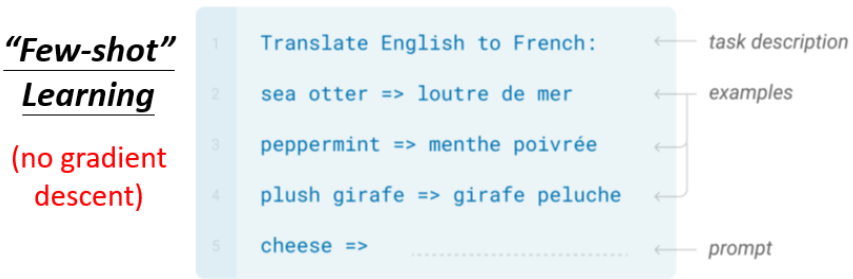
- 首先你会看到一个题目的说明,告诉你说现在要考选择题,请从ABCD四个选项裡面,选出正确的答案等等
- 然后给你一个范例,告诉你说这是题目,然后正确的答案是多少
- 然后你看到新的问题,期待你就可以举一反三开始作答

  GPT系列要做的事情就是,这个模型能不能够,做一样的事情呢

“In-context” Learning

“Few-shot” Learning

举例来说假设要GPT这个模型做翻译



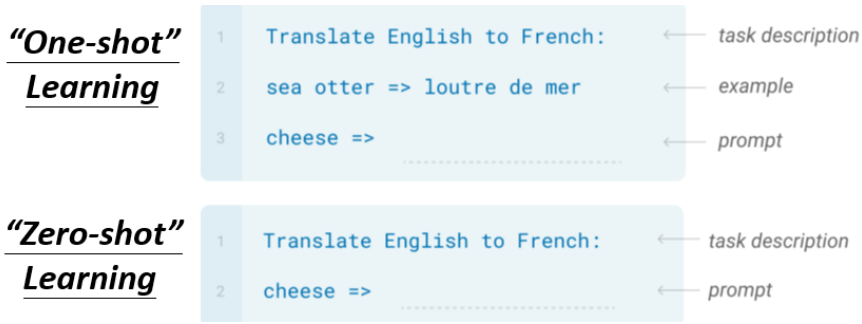
- 你就先打Translate English to French
- 就先给它这个句子,这个句子代表问题的描述
- 然后给它几个范例跟它说,sea otter然后=>,后面就应该长这个样子
- 或者是这个什麼plush girafe,plush girafe后面,就应该长这个样子等等
- 然后接下来,你问它说cheese=>,叫它把后面的补完,希望它就可以產生翻译的结果

不知道大家能不能够了解,这一个想法是多麼地狂,在training的时候,GPT并没有教它做翻译这件事,它唯一学到的就是,给一段文字的前半段,把后半段补完,就像我们刚才给大家示范的例子一样,现在我们直接给它前半段的文字,就长这个样子,告诉它说你要做翻译了,给你几个例子,告诉你说翻译是怎麼回事,接下来给它cheese这个英文单字,后面能不能就直接接出,法文的翻译结果呢

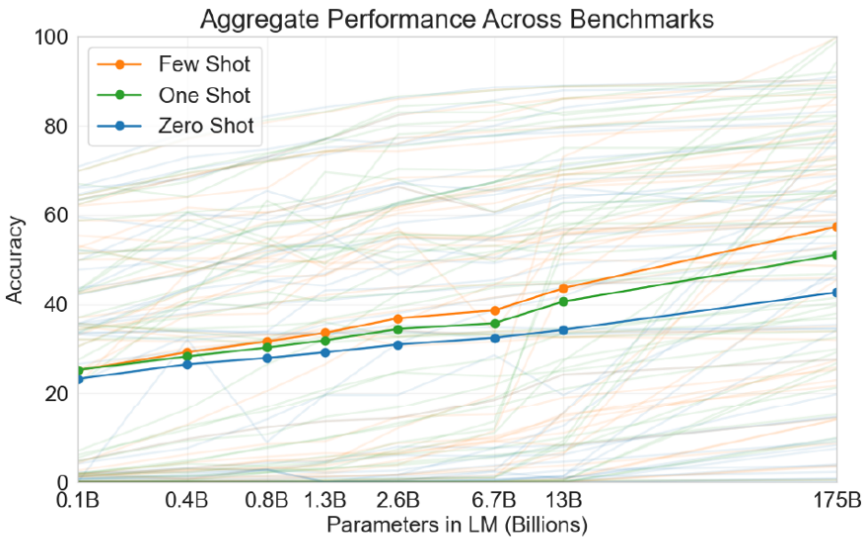
这个在GPT的文献裡面,叫做==Few-shot Learning==,但是它跟一般的Few-shot Learning,又不一样,所谓Few Shot的意思是说,确实只给了它一点例子,所以叫做Few Shot,但是它不是一般的learning,这裡面**完全没有gradient descent**,完全没有要去调,GPT那个模型参数的意思,所以在GPT的文献裡面,把这种训练给了一个特殊的名字,它们叫做==In-context Learning==,代表说它不是一种,一般的learning,它连gradient descent都没有做

“One-shot” Learning “Zero-shot” Learning

当然你也可以给GPT更大的挑战,我们在考托福听力测验的时候,都只给一个例子而已,那GPT可不可以只看一个例子,就知道它要做翻译这件事,这个叫One-shot Learning



还有更狂的,是Zero-shot Learning,直接给它一个叙述,说我们现在要做翻译了,GPT能不能够自己就看得懂,就自动知道说要来做翻译这件事情呢,那如果能够做到的话,那真的就非常地惊人了,那GPT系列,到底有没有达成这个目标呢,这个是一个见仁见智的问题啦



Average of **42** tasks

它不是完全不可能答对,但是**正确率有点低**,相较於你可以微调模型,正确率是有点低的,那细节你就再看看GPT那篇文章

第三代的GPT,它测试了42个任务,这个纵轴是正确率,这些实线 这三条实线,是42个任务的平均正确率,那这边包括了Few Shot,One Shot跟Zero Shot,三条线分别代表Few Shot,One Shot跟Zero Shot,横轴代表模型的大小,它们测试了一系列不同大小的模型,从只有0.1个billion的参数,到175个billion的参数,那从只有0.1个billion的参数,到175个billion的参数,我们看Few Shot的部分,从20几%的正确率 平均正确率,一直做到50几%的平均正确率,那至於50几%的平均正确率,算是有做起来 还是没有做起来,那这个就是见仁见智的问题啦

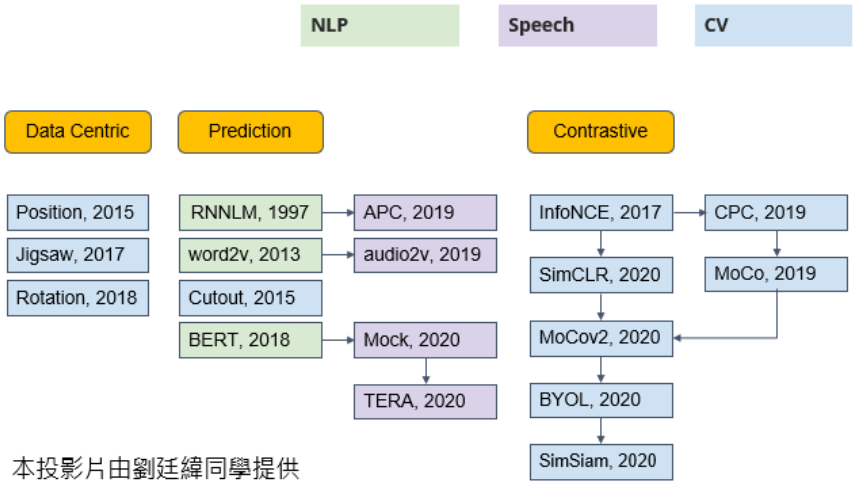
目前看起来状况是,**有些任务它还真的学会了**,举例来说2这个加减法,你给它一个数字加另外一个数字,它真的可以得到,正确的两个数字加起来的结果,但是有些任务,它可能怎麼学都学不会,譬如说一些跟**逻辑推理**有关的任务,它的结果就非常**非常地惨**,好 那有关GPT3的细节,这个就留给大家再自己研究,然后这边有一个过去上课的录影,我把连结放在这边给大家参考

https://youtu.be/DOG1L9lvsDY

Beyond Text

到目前為止我们举的例子,都是只有跟文字有关,但是你不要误会说,这种self-supervised learning的概念,只能用在文字上

在CV,CV就是computer vision,也就是影像,**在语音跟影像的应用上也都可以用,self-supervised learning的技术**,那其实今天,self-supervised learning的技术,非常非常地多,我们讲的**BERT跟GPT系列,它只是三个类型的,这个self-supervised learning的方法,的其中一种**,它们是属於**prediction**那一类



那其实还有其他的类型,那就不是我们这一堂课要讲的,那接下来的课程,你可能会觉得有点流水帐,就是我们每一个主题呢,就是告诉你说这个主题裡面,有什麼 但是细节这个更多的知识,就留给大家自己来做更进一步的研究,所以这些投影片,只是要告诉你说,在self-supervised learning这个部分,我们讲的只是整个领域的其中一小块,那还有更多的内容,是等待大家去探索的

Image - SimCLR

好那有关影像的部分呢,我们就真的不会细讲,我这边就是放两页投影片带过去,告诉你说有一招非常有名的,叫做SimCLR,它的概念也不难,我相信你自己读论文,应该也有办法看懂它

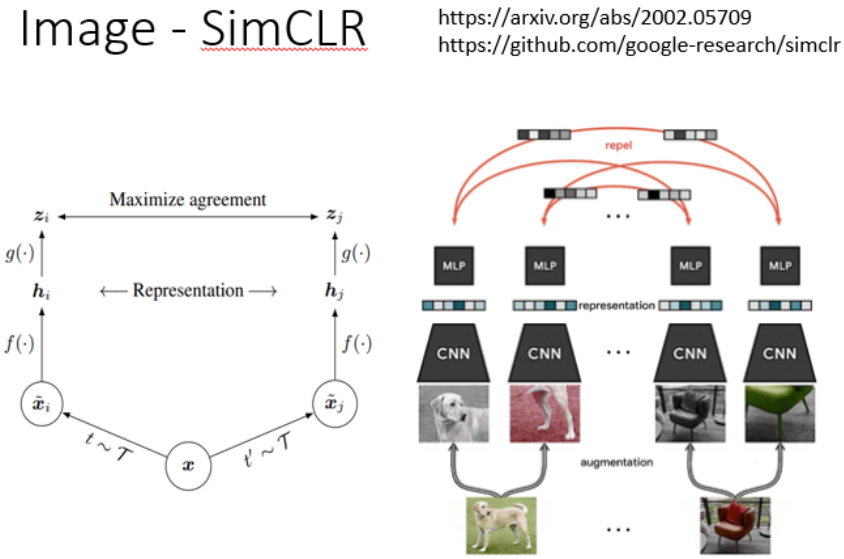
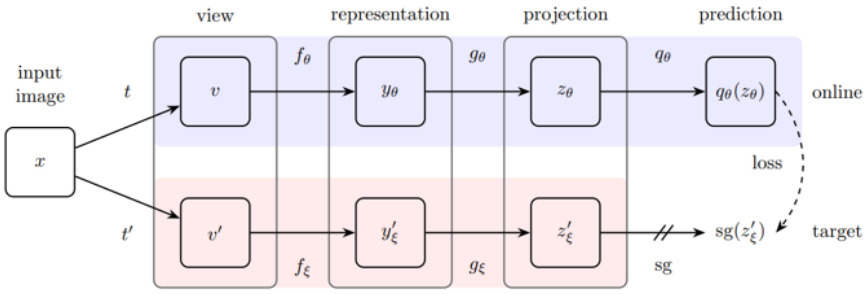


Image - BYOL

那还有很奇怪的,叫做BYOL

Bootstrap **y**our **o**wn **l**atent:
A new approach to self-supervised Learning

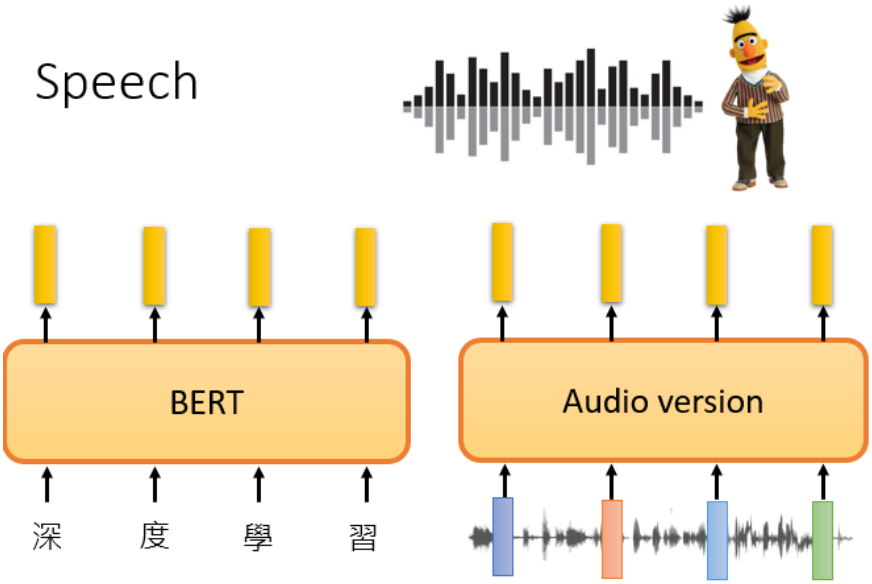
https://arxiv.org/abs/2006.07733



BYOL这个东西呢,我们是不太可能在上课讲它,為什麼呢,因為根本不知道它為什麼会work,不是 这个是很新的论文,这个是去年夏天的论文,那这个论文是,假设它不是已经发表的文章,然后学生来跟我提这个想法,我一定就是,我一定不会让他做,这不可能会work的,这是个不可能会实现的想法,不可能会成功的,这个想法感觉有一个巨大的瑕疵,但不知道為什麼它是work的,而且还曾经一度得到,state of the art的结果,deep learning就是这麼神奇,

Speech

那在语音的部分,你也完全可以使用,self-supervised learning的概念



你完全可以试著训练,语音版的BERT

那怎麼训练语音版的BERT呢,你就看看文字版的BERT,是怎麼训练的,譬如说做填空题,语音也可以做填空题,就把一段声音讯号盖起来,叫机器去猜盖起来的部分是什麼嘛,语音也可以预测接下来会出现的内容,讲GPT就是预测,接下来要出现的token嘛,那语音你也可以叫它预测,叫模型预测接下来会出现的声音去套,所以你也可以做语音版的GPT,不管是语音版的BERT,语音版的GPT,其实都已经有很多相关的研究成果了

Speech GLUE - SUPERB

不过其实在语音上,相较於文字处理的领域,还是有一些比较缺乏的东西,那我认为现在很缺乏的一个东西,就是像GLUE这样子的benchmark corpus

在自然语言处理的领域,在文字上有GLUE这个corpus,我们在这门课的刚开头,这个投影片的刚开头,就告诉你说有一个,这个基準的资料库叫做GLUE,它裡面有九个NLP的任务,今天你要知道BERT做得好不好,就让它去跑那九个任务在去平均,那代表这个self-supervised learning,模型的好坏

但在语音上 到目前為止,还没有类似的基準的资料库,所以我们实验室就跟其他的研究团队,共同开发了一个语音版的GLUE

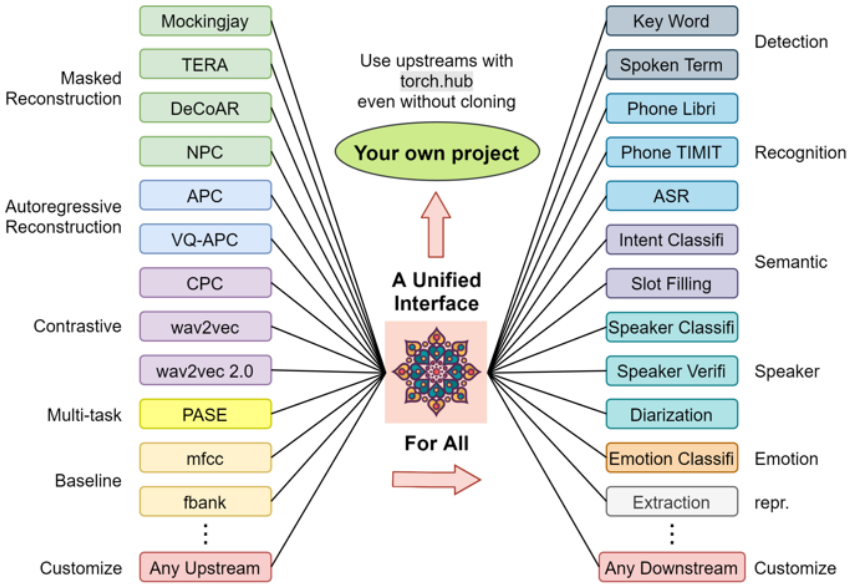
- **S**peech processing **U**niversal **PER**formance **B**enchmark
  - Will be available soon
- **Downstream:** Benchmark with 10+ tasks
  - The models need to know how to process content, speaker, emotion, and even semantics.
- **Toolkit:** A flexible and modularized framework for self-supervised speech models.
  - https://github.com/s3prl/s3prl

我们叫做SUPERB,它是Speech processing Universal,PERformance Benchmark的缩写,你知道今天你做什麼模型,都一定要硬凑梗才行啦,所以这边也是要硬凑一个梗,把它叫做SUPERB

那其实我们已经準备了差不多了,其实网站都已经做好了,只等其他团队的人看过以后,就可以上线了,所以现在虽然还没有上线,但是再过一阵子,你应该就可以找得到相关的连结

在这个基準语料库裡面,包含了十个不同的任务,那语音其实有非常多不同的面向,很多人讲到语音相关的技术,都只知道语音辨识把声音转成文字,但这并不是语音技术的全貌,语音其实包含了非常丰富的资讯,它除了有内容的资讯,就是你说了什麼,还有其他的资讯,举例来说这句话是谁说的,举例这个人说这句话的时候,他的语气是什麼样,还有这句话背后,它到底有什麼样的语意,所以我们準备了十个不同的任务,这个任务包含了语音不同的面向,包括去检测一个模型,它能够识别内容的能力,识别谁在说话的能力,识别他是怎麼说的能力,甚至是识别这句话背后语意的能力,从全方位来检测一个,self-supervised learning的模型,它在理解人类语言上的能力

而且我们还有一个Toolkit,这个Toolkit裡面就包含了,各式各样的,self-supervised learning的模型



https://github.com/andi611/Self-Supervised-Speech-Pretraining-and-Representation-Learning

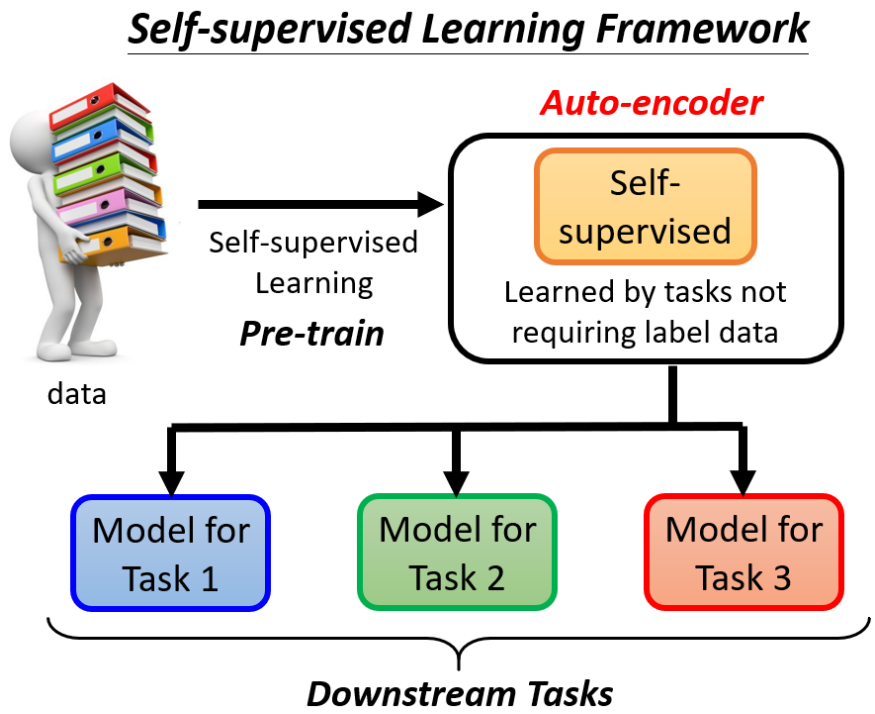
还有这些,self-supervised learning的模型,它可以做的,各式各样语音的下游的任务,然后把连结放在这边给大家参考

讲这些只是想告诉大家说,self-supervised learning的技术,不是只能被用在文字上,在这个影像上 在语音上,都仍然有非常大的空间可以使用,self-supervised learning的技术,好 那这个,self-supervised learning的部分呢,这个BERT跟GPT我们就讲到这边,

Auto-Encoder P1

Self-supervised Learning Framework

在讲 Auto-Encoder 之前,其实 Auto-Encoder 也可以算是,Self-Supervised Learning 的一环,所以再让我们用非常短的时间,来看一下Self-Supervised Learning 的 Framework



首先你有大量的没有标注的资料,用这些没有标注的资料,你可以去训练一个模型,你必须发明一些不需要标注资料的任务,比如说做填空题,比如说预测下一个 Token

这个不用标注资料的学习叫做,==Self-Supervised Learning==,或者是也有人叫==Pre-Training==,那用这些不用标注资料的任务,学完一个模型以后,它本身没有什麼用,BERT 只能做填空题,GPT 只能够把一句话补完,但是你可以把它用在其他下游的任务裡面

你可以把 Self-Supervised Learning 的 Model,做一点点的微微的调整,就可以用在下游的任务裡面

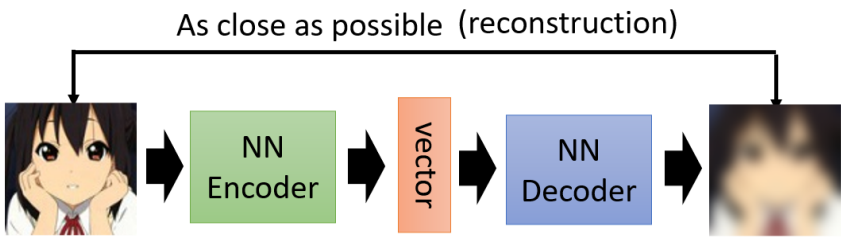
在有 BERT 在有 GPT 之前,其实有一个更古老的任务,更古老的不需要用标注资料的任务,就叫做 Auto-Encoder,所以你也可以**把 Auto-Encoder,看作是 Self-Supervised Learning 的,一种 Pre-Train 的方法**

当然可能不是所有人都会同意这个观点,有人可能会说这个 Auto-Encoder,不算是 Self-Supervised Learning,这个 Auto-Encoder 很早就有了嘛,2006 年 15 年前就有了嘛,然后 Self-Supervised Learning 是,19 年才有这个词彙嘛,所以 Auto-Encoder,不算 Self-Supervised Learning 的一环

那这个都是见仁见智的问题,这种名词定义的问题,真的我们就不用太纠结在这个地方,从 Self-Supervised Learning,它是不需要用 Label Data 来训练,这个观点来看,Auto-Encoder 我认为它可以算是,Self-Supervised Learning 的其中的一种方法,它就跟填空 预测,接下来的 Token 是很类似的概念,只是用的是另外不一样的想法

Auto-encoder

Auto-Encoder 是怎麼运作的呢,那现在我们,因为刚才在讲 Self-Supervised Learning 的时候,都是用文字做例子,那现在我们换成用影像来做例子



假设你有非常大量的图片,在 Auto-Encoder 裡面你有两个 Network,一个叫做 Encoder,一个叫做 Decoder,他们就是两个 Network

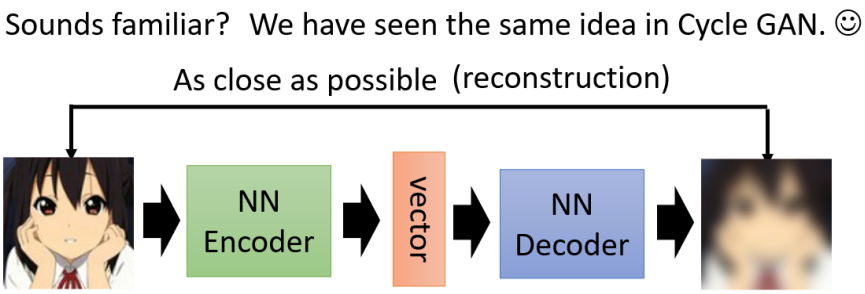
- Encoder 把一张图片读进来,它把这张图片变成一个向量,就 Encoder 它可能是很多层的 CNN,把一张图片读进来,它的输出是一个向量,接下来这个向量会变成 Decoder 的输入
- Decoder 会产生一张图片,所以 Decoder 的 Network 的架构,可能会像是 GAN 裡面的 Generator,它是 11 个向量输出一张图片

训练的目标是希望,**Encoder 的输入**跟 **Decoder 的输出,越接近越好**

假设你把图片看作是一个很长的向量的话,我们就希望这个向量跟 Decoder 的输出,这个向量,这两个向量他们的距离越接近越好,也有人把这件事情叫做 ==Reconstruction==,叫做重建

因为我们就是把一张图片,压缩成一个向量,接下来 Decoder 要根据这个向量,重建原来的图片,那我们希望原输入的结果,跟重建后的结果越接近越好

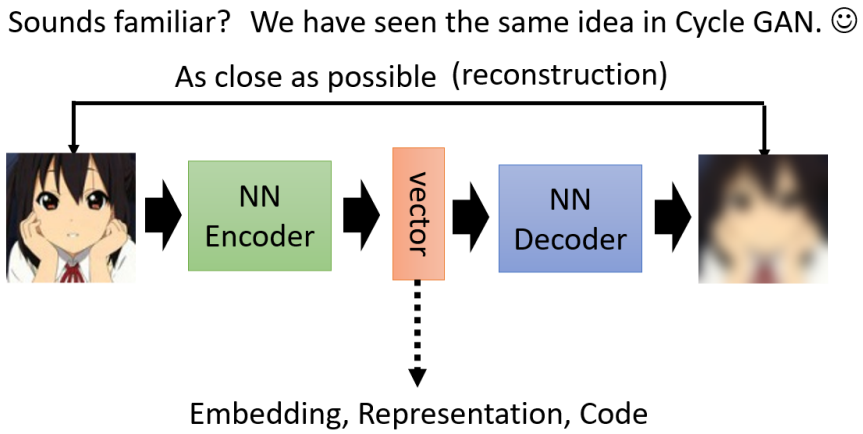
讲到这边你可能会发现说,这个东西 这个概念似曾相似,没错 我们在讲 ==Cycle GAN== 的时候,已经讲过了这个概念



我们说在做 Cycle GAN 的时候,我们会需要两个 Generator,第一个 Generator,把 X Domain 的图片转到 Y Domain,另外一个 Generator,把 Y Domain 的图片转回来,希望最原先的图片,跟转完两次后的图片越接近越好

那这边 Encoder 和 Decoder,这个 Auto-Encoder 的概念,跟 Cycle GAN 其实是一模一样的,都是希望所有的图片经过两次转换以后,要跟原来的输出越接近越好,而这个训练的过程,完全不需要任何的标注资料,你只需要蒐集到大量的图片,你就可以做这个训练

所以它是一个 **Unsupervised Learning 的方法**,跟 Self-Supervised 那一系列,Pre-Training 的做法一样,你**完全不需要任何的标注资料**

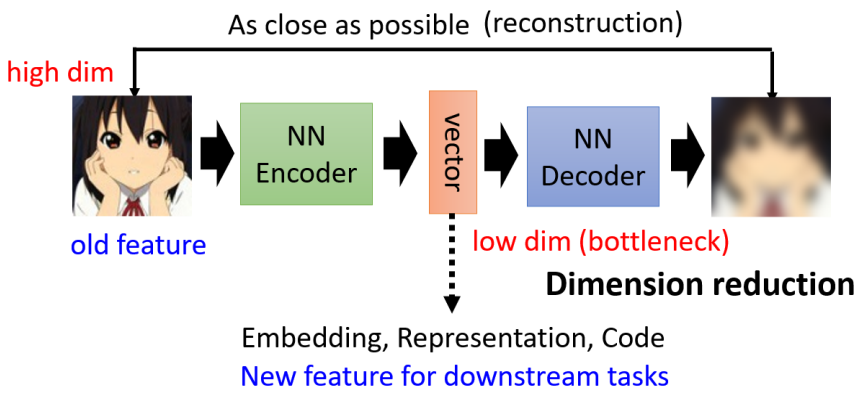


那像这样子这个 Encoder 的输出,有时候我们叫它 Embedding,我们在讲 BERT 的时候,也提过 Embedding 这个词彙了,那有的人叫它 Representation,有的人叫它 Code,因为 Encoder 是一个编码嘛,所以这个有人把这个 Vector 叫做 Code,那其实指的都是同一件事情

怎麼把 Train 的 Auto-Encoder,用在 Downstream 的任务裡面呢

常见的用法就是,原来的图片,你也可以把它看作是一个很长的向量,但这个**向量太长了 不好处理**,那怎麼办呢

你把这个图片丢到 **Encoder 以后,输出另外一个向量,这个向量你会让它比较短**,比如说只有 10 维 只有 100 维,那你拿这个新的向量来做你接下来的任务,也就是图片不再是一个很高维度的向量,它通过 Encoder 的压缩以后,变成了一个低维度的向量,你再拿这个低维度的向量,来做接下来想做的事情,这就是常见的,Auto-Encoder用在 Downstream 的任务,用在下游任务的方法

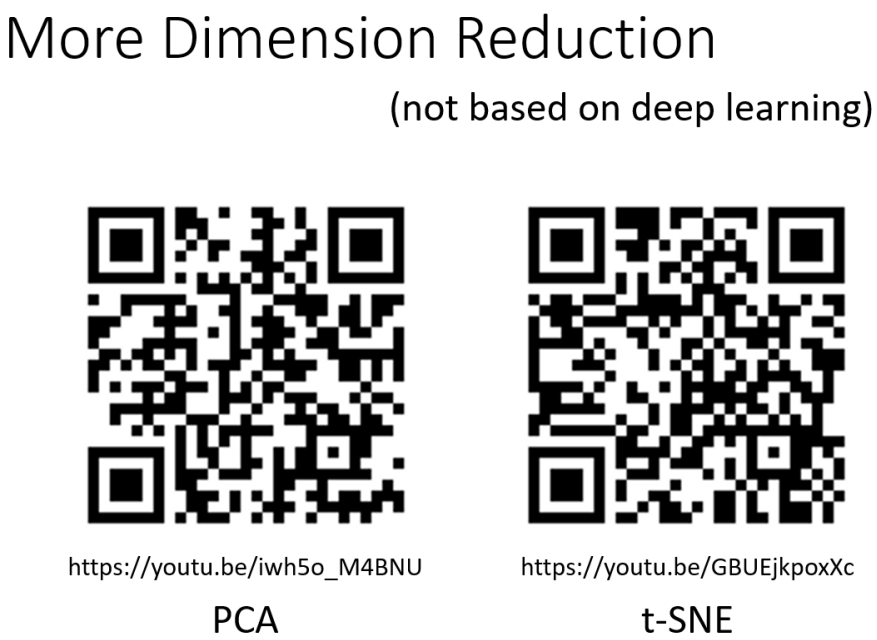


那因为通常 Encoder 的输入,是一个维度非常高的向量,而 Encoder 的输出,也就是我们的 Embedding,Representation 或者 Code,它是一个非常低维度的向量,比如说输入是 100×100 的图片,那 100×100 那就是 1 万维的向量了,如果是 RGB 那就是 3 万维的向量

但是通常 Encoder 的 Output 你会设得很小,比如说 10，100 这样的等级,所以这个这边会有一个特别窄的地方,所以这个部分,这个 Encoder 的输出,有时候又叫做 ==Bottleneck==,叫做瓶颈,就本来输入是很宽的,输出也是很宽的 中间特别窄,所以这一段就叫做 Bottleneck

而 Encoder 做的事情,是把本来很高维度的东西,转成低维度的东西,把高维度的东西转成低维度的东西又叫做 ==Dimension Reduction==

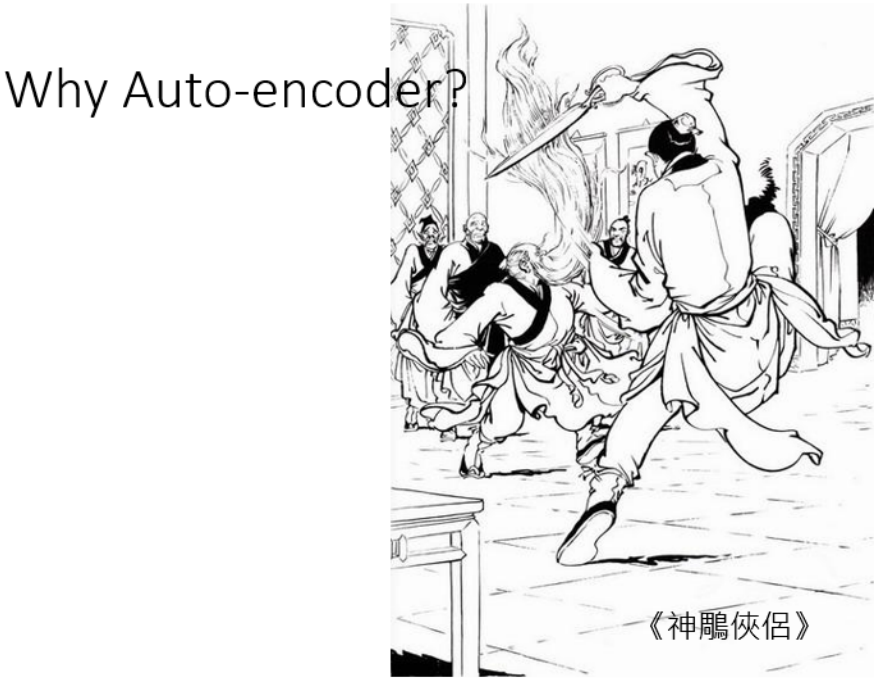
Dimension Reduction 这个技术,我相信你在 Machine Learning 相关的应用上,应该常常听到这个名词,那有关 Dimension Reduction 的技术,它其实牵涉的非常非常地广,所以我们这边就不再细讲,因为这门课,我们只专注在深度学习相关的技术,你可以把 Auto-Encoder 的 Encoder,当作拿来做 Dimension Reduction,那其他还有很多不是 Deep Learning Base的,不是以深度学习为基础的,Dimension Reduction的技术,我就把录影的连接留在这边



比如说 PCA 比如说 T-SNE,我就把录影的连结留在这边给大家参考

Why Auto-encoder?

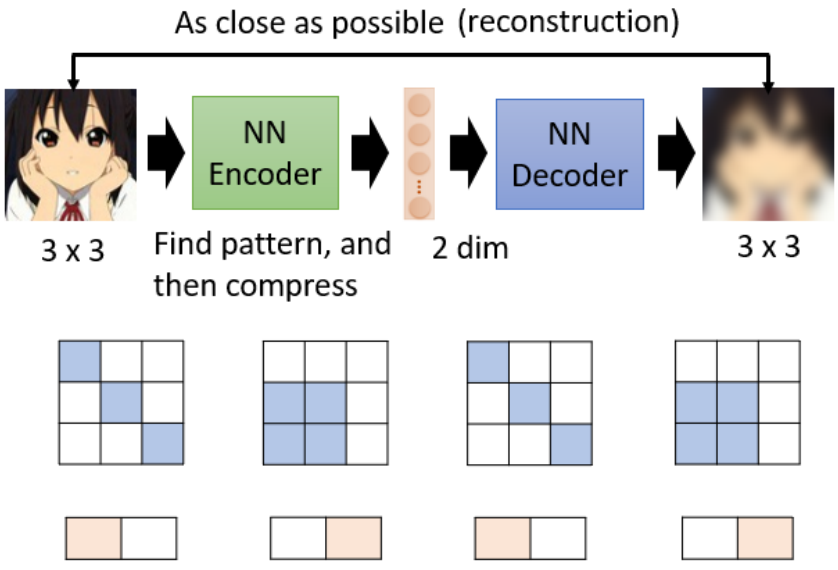
好 那 Auto-Encoder 到底好在哪裡,当我们把一个高维度的图片,变成一个低维度的向量的时候,到底带来什麼样的帮助,这让我想到神鵰侠侶的其中一段



神鵰俠侶裡面有一段,就是杨过进去那个绝情谷,遇到这个绝情谷谷主公孙止的弟子,就是樊一翁,樊一翁就是这个人,那樊一翁的武器是什麼,他的武器除了一根钢杖以外,还有他的鬍子,他可以去甩动他的鬍子当做一个软鞭来使用,他的鬍子甩起来有两丈那麼长,可以是一个很厉害的武器,杨过跟他打了很久都难分上下

突然呢杨过说,我在三招之内一定要剪掉你的鬍子,大家突然都很诧异,想说杨过虽然武功可能比樊一翁还高一点,但是也没有高太多,怎麼有办法三招就剪掉他的鬍子,后来杨过真的在三招内剪掉他的鬍子,为什麼呢,因为杨过发现说,这个鬍子是由头所操控的,虽然鬍子甩开来有两丈那麼长,但是头能够做的变化还是有限的,所以虽然表面鬍子的鞭法非常地厉害,但是只要直接去打他的头,就直接去打他脸,就会逼著他不得不闪避,就会逼著他这个鬍子能够动的路线变得有限,然后就打败了樊一翁,就把他的鬍子剪掉了,故事结束,那这个跟 Auto-Encoder 有什麼关係呢

好 我们来想一下,Auto-Encoder 这件事情它要做的,是**把一张图片压缩又还原回来,**但是还原这件事情为什麼能成功呢



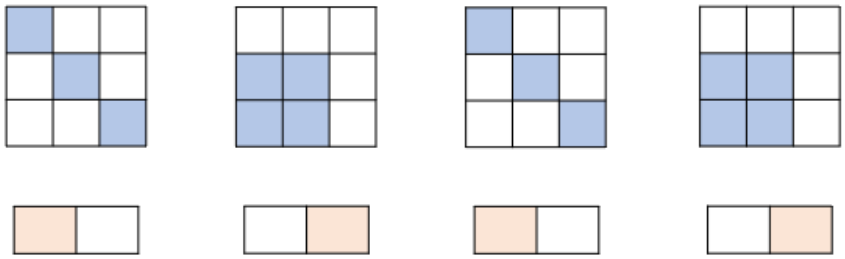
你想想看假设本来图片是 3×3,3×3 很小,但我们就假设 3×3 好了,本来的图片是 3×3,你要用 9 个数值来描述一张 3×3 的图片,假设 Encoder 输出的这个向量是二维的,我们怎麼有可能从二维的向量,去还原 3×3 的图片,还原9个数值呢

我们怎麼有办法把 9 个数值变成 2 个数值,又还原成 3,又还原回 9 个数值呢

能够做到这件事情是因为,**对于影像来说,并不是所有 3×3 的矩阵都是图片,图片的变化其实是有限的**,你随便 Sample 一个 Random 的 Noise,随便 Sample 一个矩阵,出来它通常都不是你会看到的图片

举例来说,假设图片是 3×3 的,那它的变化,虽然表面上应该要有 3×3 个数值,才能够描述 3×3 的图片,但是也许它的变化实际上是有限的

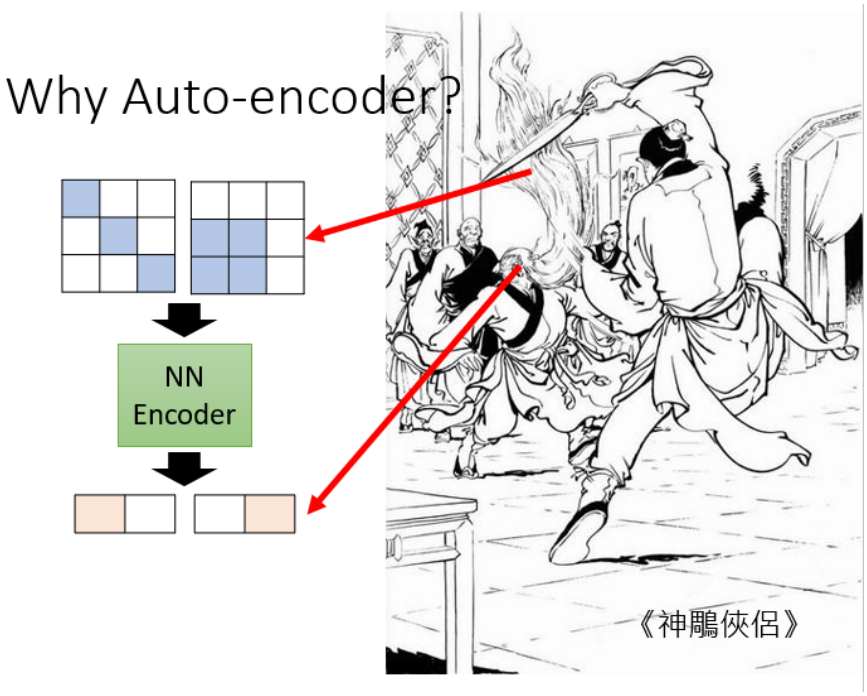
也许你把图片蒐集起来发现说



它只有这样子的类型,跟这样子的类型,其他类型根本就不是,你一般在训练的时候会看到的状况,就是因为说图片的变化还是有限的

所以你在做这个 Encoder 的时候,Encoder 可以说,我就只用两个维度就可以描述一张图片,虽然图片是 3×3,应该用 9 个数值才能够储存,但是实际上它的变化也许只有两种类型,那你就可以说看到这种类型,我就左边这个维度是 1 右边是 0,看到这种类型就左边这个维度是 0,右边这个维度是 1

那所以对应到刚才这个樊一翁的例子



就是这个鬍子是图片複杂的状态,是原来图片的 Pixel,是原来图片的像素

而 Encoder 做的事情就是化繁为简,本来比较複杂的东西,它只是**表面上比较複杂,事实上它的变化其实是有限的**,你只要找出它有限的变化,你就可以把本来複杂的东西,把它变得用比较简单的方法来表示它

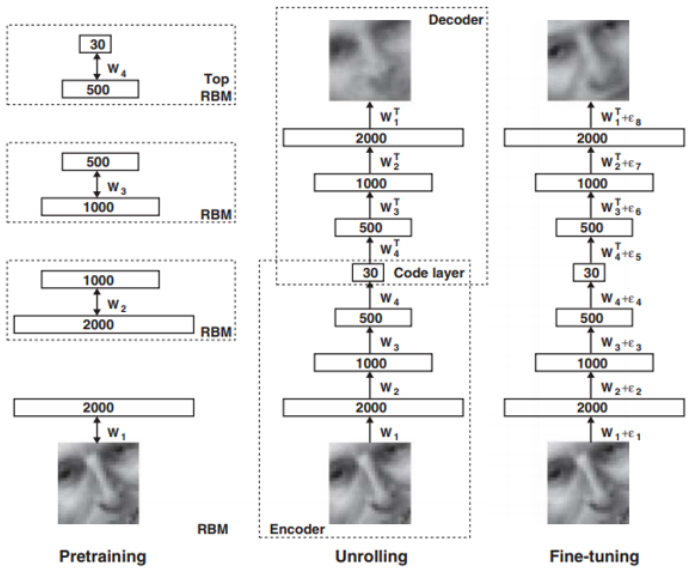
如果我们可以把複杂的图片,用比较简单的方法来表示它,那我们就只需要比较少的训练资料,在下游的任务裡面,我们可能就只需要比较少的训练资料,就可以让机器学到,我们本来要它学的事情,这个就是 Auto-Encoder 的概念

Auto-encoder is not a new idea

那 Auto-Encoder,它从来都不是一个新的想法,它真的是非常非常地有历史,举例来说在这个 Hinton,Hinton 就是 Deep Learning 之父

Hinton 在 06 年的 Science 的 Paper 裡面,就有提到 Auto-Encoder 这个概念,只是那个时候用的 Network,跟今天用的 Network,当然还是有很多不一样的地方,我们讲 2006 年是 15 年前,15 年前的 Auto-Encoder 长什麼样子

Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507



那个时候人们**不觉得,Deep 的 Network 是 Train 得起来的**,那时候觉得说这个把 Network 叠很多很多层,然后每一层一起 Train 不太可能成功,所以那时候的信念是,**每一层应该分开训练**,所以 Hinton 用的是一个叫做,Restricted Boltzmann Machine 的技术,缩写是 RBM

我们特别把 Hinton 15 年前的文章,把它的裡面的这个,Paper 裡面的图拿出来给大家看一下,过去 15 年前,人们是怎麼看待深度学习这个问题,那个时候觉得说,要 Train 一个这个很深的 Network 不太可能,每一层分开要 Train,虽然这个说很深也没有很深,只是三层,这个跟你作业 2 做得还要更shallow,但是在15年前这个已经是,哇 很深啊 它有三层太可怕了

那这个三层要分开来训练才可以,那这边说分开来训练这件事情叫做 Pretraining,但它跟 Self-Supervised Learning 的 Pre-Train,又不一样

假设你说 Auto-Encoder 这个东西是 Pre-Train,那现在这个 Pre-Train 是,Pre-Train 的 Pre-Train,它是要 Pre-Train 那个 Auto-Encoder,而且每一层用一个叫做 RBM 的技术,分开来训练

先把每一层都训练好,再全部接起来做微调这件事情,那这边的微调并不是 BERT 的微调,它是微调那个 Pre-Train 的 Model

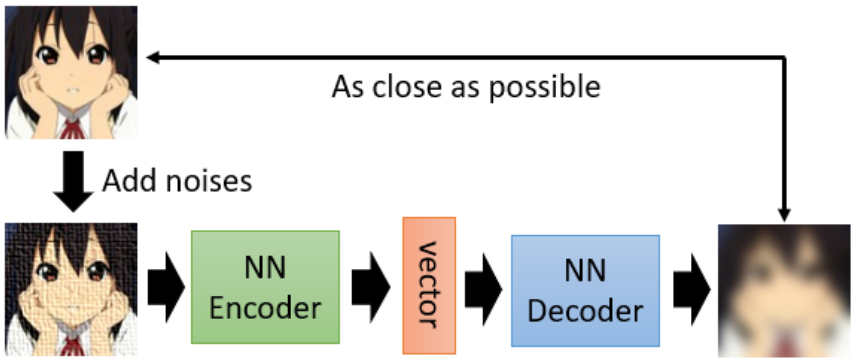
那这个 Restricted Boltzmann Machine,你会发现今天很少有人在提到它了,它其实不是一个 Deep Learning 的技术,它有点複杂,我们在这门课裡面也没有打算要深入细讲,什麼是 Restricted Boltzmann Machine,那为什麼现在都没有什麼人用它呢,就是因为**它没有什麼用**

在10年前呢,都相信这个 Deep 的 Network,一定要用 Restricted Boltzmann Machine,然后其实 Hinton 后来在 2012 年的时候,有一篇 Paper 偷偷在结尾下一个结论说,**其实 Restricted Boltzmann Machine,也没有什麼必要** ,所以后来就没有什麼人再用Restricted Boltzmann Machine

而且那时候还有一个神奇的信念,是觉得说那个 Encoder Decoder,它必须是对称,所以 Encoder 的第一层,跟 Encoder 的最后,跟 Decoder 的最后一层,他们必须互为 Transfers,不过现在已经没有,比较少有人在使用这样子的限制,好 这张投影片只想告诉你说,**Auto-Encoder 不是新的概念,它是一个非常有历史的概念**

De-noising Auto-encoder

那 Auto-Encoder 还有一个常见的变形,叫做 De-Noising 的 Auto-Encoder



The idea sounds familiar? ☺

Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML*, 2008.

De-Noising 的 Auto-Encoder 是说,我们把原来要输进去给 Encoder 的图片,**加上一些杂讯**,就自己随便找一个杂讯把它加进去,然后一样通过 Encoder,一样再通过 Decoder,试图还原原来的图片

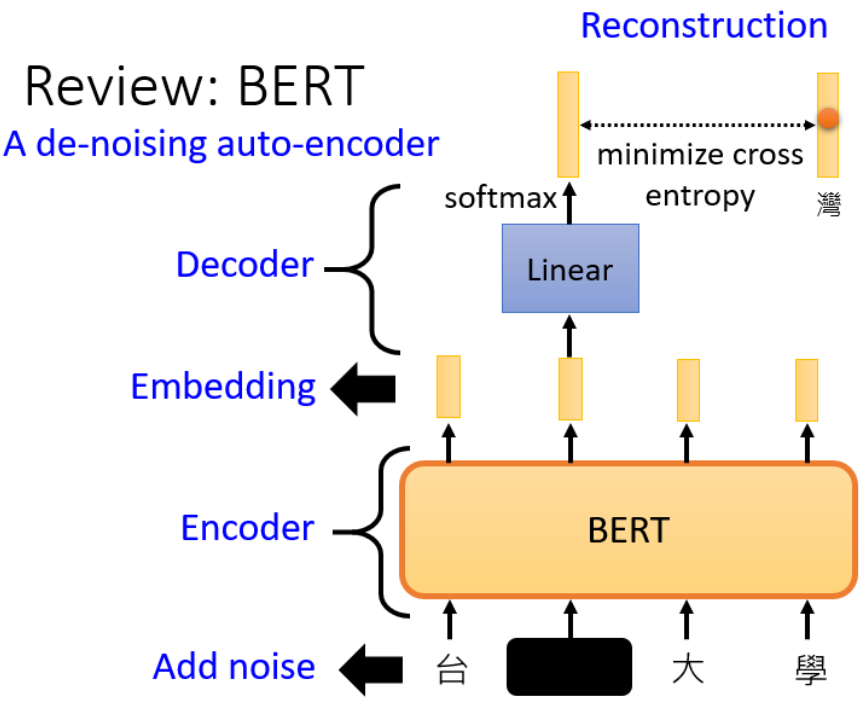
那我们现在还原的,不是 Encoder 的输入,Encoder 的输入的图片是有加杂讯的,我们要还原的不是 Encoder 的输入,我们**要还原的是加入杂讯之前的结果**

所以你会发现说,现在 Encoder 跟 Decoder,除了还原原来的图片这个任务以外,它还**多了一个任务**,这个任务是什麼,这个任务就是,它必须要**自己学会把杂讯去掉**

Encoder 看到的是没有杂讯的图片,但 Decode要还原的目标是,Encoder 看到的是有加杂讯的图片,但 Decoder 要还原的目标是,没有加杂讯的图片,所以 Encoder 加 Decoder,他们合起来必须要联手能够把杂讯去掉,这样你才能够把,De-Noising 的 Auto-Encoder 训练起来

那说到 De-Noising 的 Auto-Encoder,有没有发现这个概念,其实也一点都不陌生呢,De-Noising 的 Auto-Encoder,也不算是太新的技术,至少在 2008 年的时候,就已经有相关的论文了

但是如果你看今天的 BERT 的话,其实你也可以把它看作就是一个,De-Noising 的 Auto-Encoder



输入我们会加 Masking,那些 **Masking 其实就是 Noise**,BERT 的模型就是 Encoder,它的输出就是 Embedding

在讲 BERT 的技术的时候,我们就告诉你说这个输出就叫做 Embedding,接下来有一个 Linear 的模型,就是 Decoder,Decoder 要做的事情,就是还原原来的句子,也就是把填空题被盖住的地方,把它还原回来,所以我们可以说,BERT 其实就是一个,De-Noising 的 Auto-Encoder

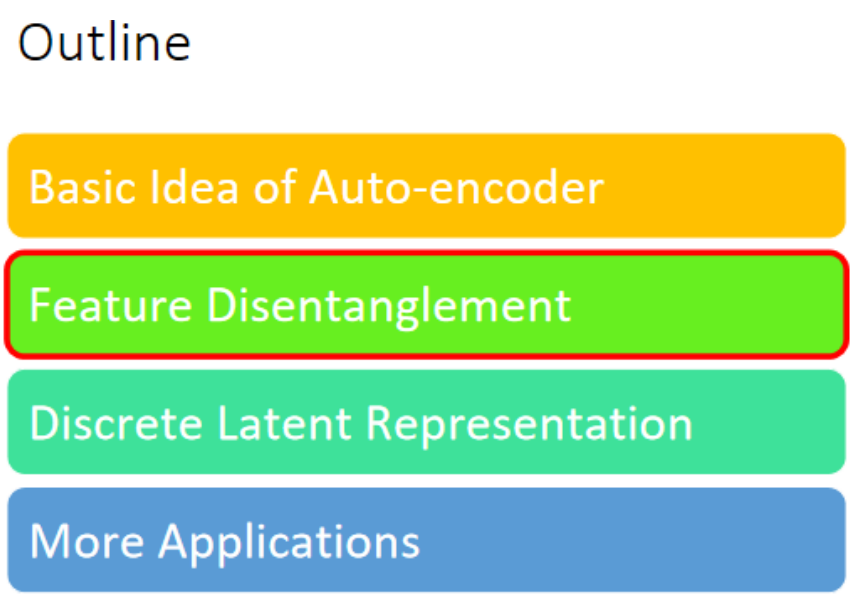
有同学可能会问说,为什麽这个 Decoder 一定要 Linear 的呢,它**不一定要是 Linear**,它可以不是 Linear

或者是我们换一个说法,这个 BERT 它有 12 层,最小的那个 BERT 有 12 层,比较大的有 24 层或者是 48 层,好 那最小的 BERT 是 12 层,如果我们说这个 12 层中间,第 6 层的输出是 Embedding,那你其实也可以说剩下的 6 层,就是 Decoder,你可以说 BERT,就假设你在用 BERT 的时候,你用的不是第 12 层的输出,而是第 6 层的输出,那你完全可以说,BERT 的前 6 层就是 Encoder,后面 6 层就是 Decoder,总之这个 Decoder,没有一定要是 Linear

Auto-Encoder P2

Feature Disentangle

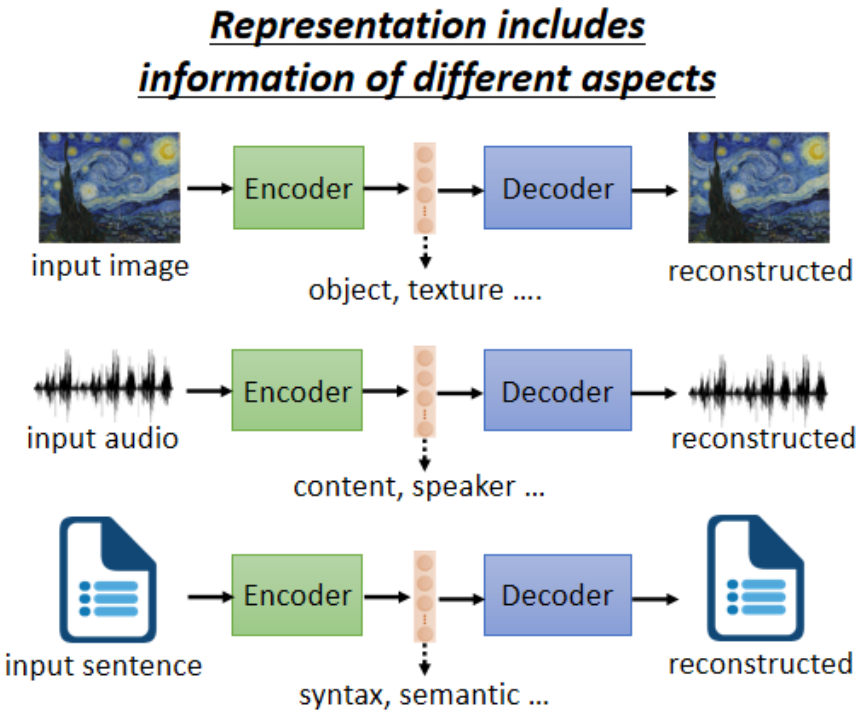
接下来啊,除了 Aauto-Encoder,可以用来做当 strime 的任务以外,我还想跟大家分享一下,Aauto-Encoder 其他有意思的应用： ==Feature Disentanglement==



Disentangle 的意思就是,把一堆本来纠缠在一起的东西把它解开

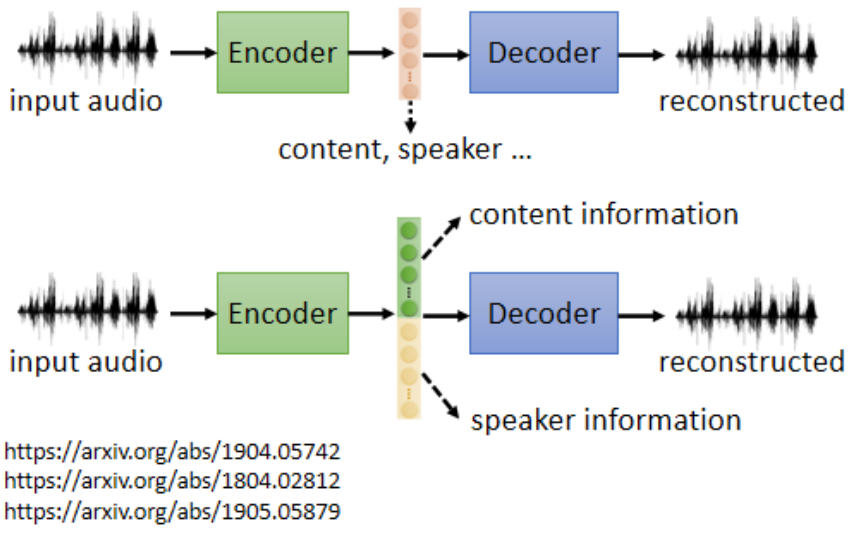
那为什么会有 Disentangle 这个议题呢,我们来想想看,Aauto-Encoder 它在做的事情是什么

Auto-Encoder 在做的事情是



- 如果是图片的话,就是把一张图片变成一个 Code,再把 Code 呢 变回图片,既然这个 Code 可以变回图片,代表说这个 Code 裡面啊,有很多的资讯,包含图片裡面所有的资讯,举例来说,图片裡面有什么样的东西啊,图片的色泽纹理啊等等
- Auto-Encoder 这个概念也不是只能用在影像上,如果用在语音上,你可以把一段声音丢到 Encoder 裡面,变成向量 再丢回 Decoder,变回原来的声音,代表这个向量包含了,语音裡面所有重要的资讯,包括这句话的内容是什么,就是 Encoder 的资讯,还有这句话是谁说的,就是 Speaker 语者的资讯
- 那如果今天是一篇文章,丢到 Encoder 裡面变成向量,这个向量通过 Decoder 会变回原来的文章,那这个向量裡面有什么,它可能包含文章裡面,文句的句法的资讯,也包含了语意的资讯,但是这些资讯是全部纠缠在一个向量裡面,我们并不知道一个向量的哪些维,代表了哪些资讯

举例来说,如果我们今天把一段声音讯号丢进 Encoder,它会给我们一个向量,但是这个向量裡面,哪些维度代表了这句话的内容,哪些维度代表这句话的语者,也就是谁说的,我们没有这样的资讯

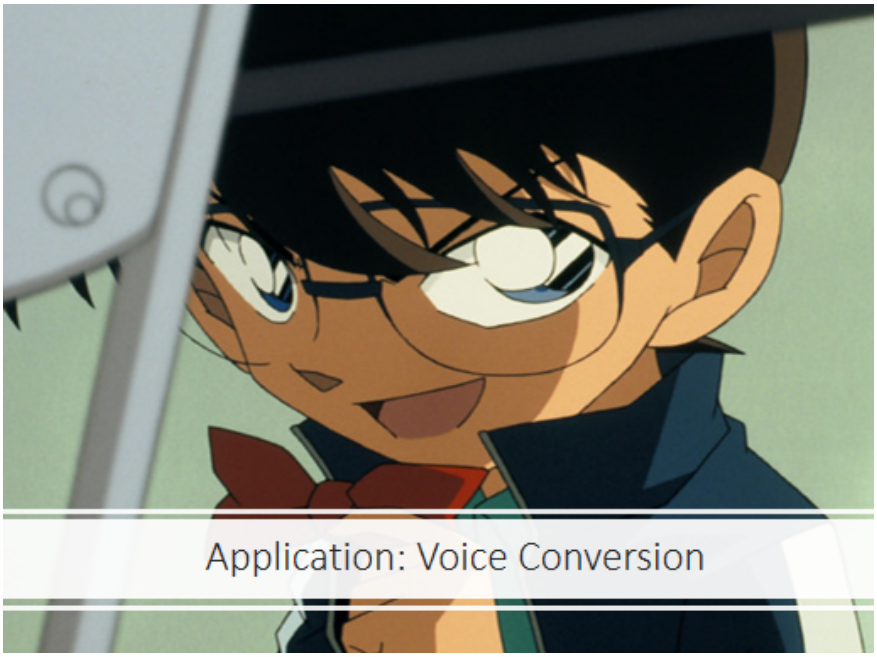


而 Feature Disentangle 想要做到的事情就是,我们有没有可能想办法,在 Train 一个 Aauto-Encoder 的时候,同时有办法知道,这个 Representation,或又叫做 Embedding,或又叫做 Code,我们这个 **Embedding 的哪些维度代表了哪些资讯**呢

我们有没有可能做到说 Encoder 输出一个,举例来说 100 维的向量,我们知道说**前 50 维就代表了这句话的内容,后 50 维就代表了这句话说话人的特徵**呢,那这样子的技术就叫做 Feature Disentangle

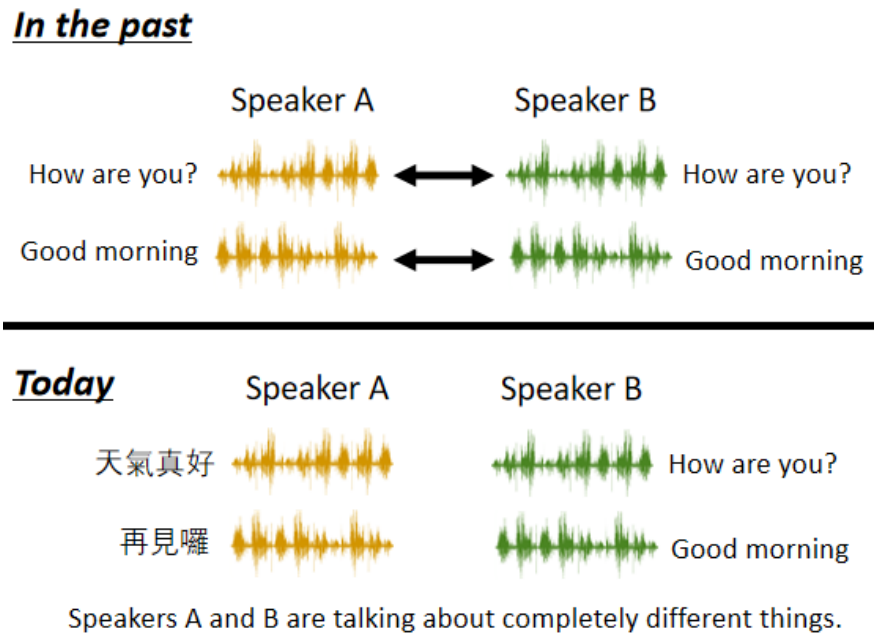
我们就是主要就想告诉大家说,Feature Disentangle 是有办法做的,那至于实际上怎么做,我在这边就列几篇论文,给有兴趣的同学参考,如果你没有兴趣的话,就知道说这件事情是可行的,我们有可能知道 Aauto-Encoder 裡面,每一个 Dimension 代表了什么样的资讯

这边举一个语音上的应用,这个应用叫做 Voice Conversion,Voice Conversion 的中文叫做语者转换,所以也许你没有听过语者转换这个词彙,但是你一定看过它的应用,它就是柯南的领结变身器



这个在二十年前,阿笠博士就已经做得很成功了啦

那只是过去,阿笠博士在做这个 Voice Conversion 的时候啊,我们需要成对的声音讯号,也就是假设你要把 A 的声音转成 B 的声音,你必须把 A 跟 B 都找来,叫他唸一模一样的句子



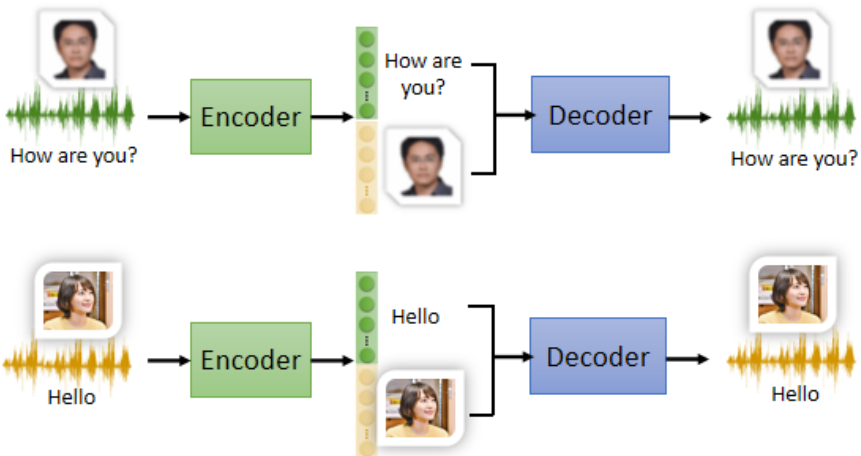
就 A 说好 How are you,B 也说好 How are you,A 说 Good morning,B 也说 Good morning,他们两个各说一样的句子,说个 1000 句,接下来呢,就结束了,就是 **Supervised Learning 的问题**啊,你有成对的资料,Train 一个 Supervised 的 Model,把 A 的声音丢进去,输出就变成 B 的声音,就结束了

但是如果 A 跟 B 都需要唸一模一样的句子,念个 500 1000 句,显然是不切实际的,举例来说,假设我想要把我的声音转成新垣结衣的声音,我得把新垣结衣找来,更退一万步说,假设我真的把新垣结衣找来,她也不会说中文啊,所以她没有办法跟我唸一模一样的句子

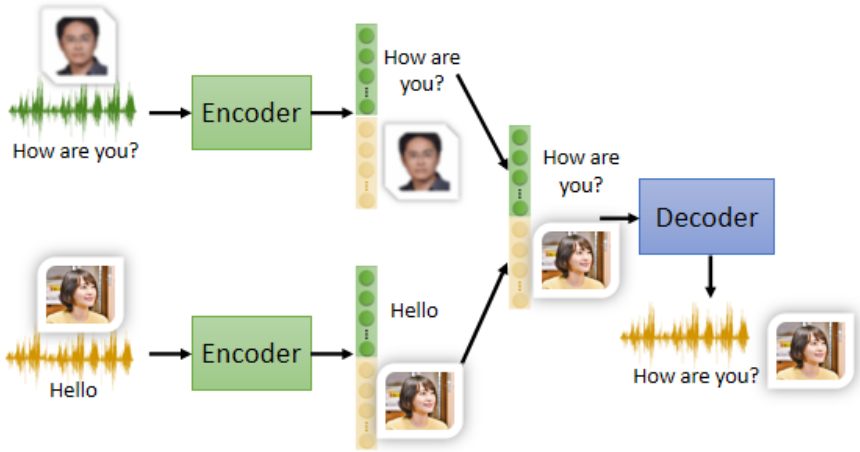
而今天有了 Feature Disentangle 的技术以后,也许我们期待机器可以做到,**就给它 A 的声音 给它 B 的声音,A 跟 B 不需要唸同样的句子,甚至不需要讲同样的语言,机器也有可能学会把 A 的声音转成 B 的声音**

那实际上是怎么做的呢,假设我们收集到一大堆人类的声音讯号,然后拿这堆声音讯号呢,去 Train 一个 Aauto-Encoder,同时我们又做了 Feature Disentangle 的技术,所以我们**知道在 Encoder 的输出裡面,哪些维度代表了语音的内容,哪些维度代表了语者的特徵**

接下来,我们就可以**把两句话,声音跟内容的部分互换**

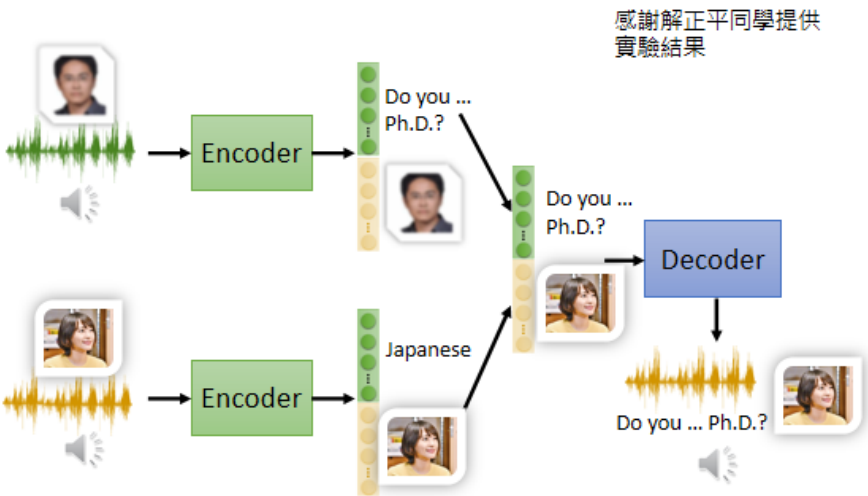


举例来说,这边是我的声音,我说 How are you,丢进 Encoder 以后,那你就可以抽出,你就知道说这个 Encoder 裡面,**某些维度代表 How are you 的内容,某些维度代表我的声音**



今天你把这个你老婆的声音丢进 Encoder,它就知道某一些维度,代表你老婆说的话的内容,某一些维度,代表你老婆声音的特徵,接下来我们只要**把我说话的内容的部分取出来,把你老婆说话的声音特徵的部分取出来,把它拼起来**,丢到 Decoder 裡面,就可以用**你老婆的声音,讲我说的话的内容**

这件事情真的有可能办到吗,以下是真正的例子,听起来像是这个样子,Do you want to study a PhD,这个是我的声音，那把我的声音丢到 Encoder 裡面以后呢,你可以想像说在 Encoder 裡面,我们知道哪些维度代表了念博班这件事,哪些维度代表了我的声音



那为了简化起见,它输出 100 维的向量,前 50 维代表内容,后 50 维代表说话人的特徵,好 接下来这句话是你老婆说的,仕事忙しいのがな,不知道 不太确定在说什么,就是日文啊

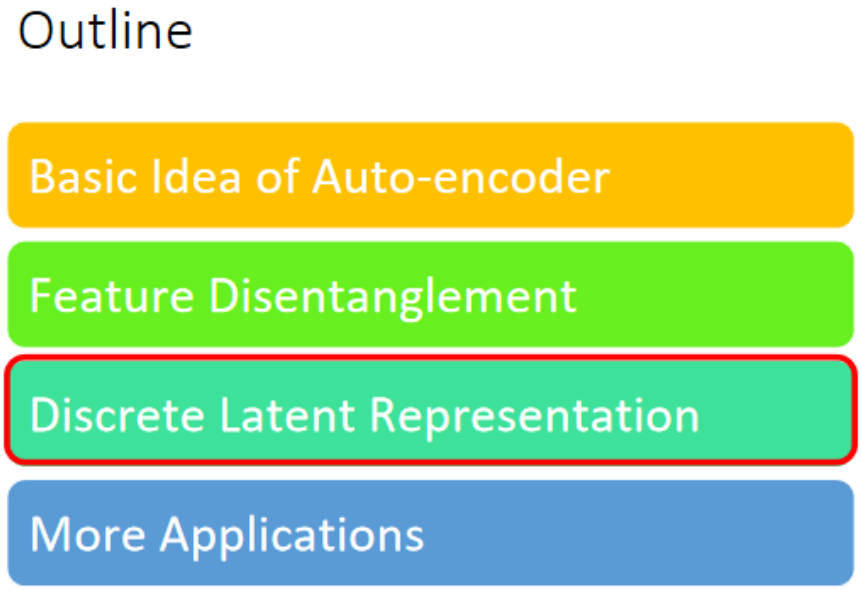
接下来呢,就把我的声音的前 50 维,代表内容的部分取出来,把你老婆的,把你老婆的声音丢进 Encoder 以后,后 50 维的部分抽出来,拼起来,一样是一个 100 维的向量,丢到 Decoder 裡面,看看输出来的声音,是不是就是你老婆叫你念博班的声音,听起来像是这个样子,Do you want to study a PhD

那其实反过来也可以啦,就是换成把日文的部分拿出来,把我的声音的特徵拿出来,一样串成一个 100 维的向量,丢到 Decoder 裡面,它听起来就会变成这样,仕事忙しいのがな,我也不知道自己在说什么就是了

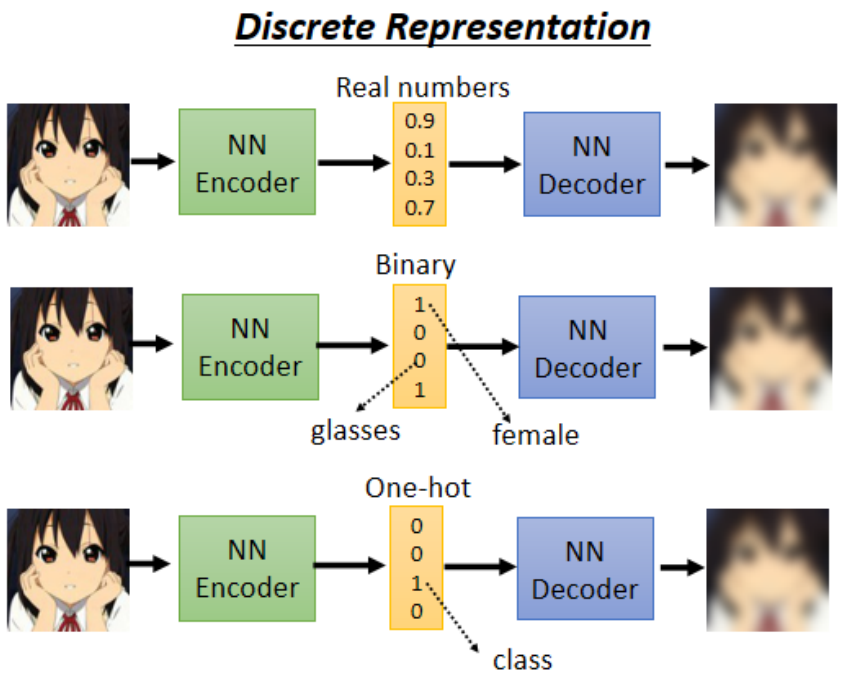
所以确实用 Feature Disentangle,你有机会做到 Voice Conversion,那其实在影像上,在 NLP 上,也都可以有类似的应用,所以可以想想看,Feature Disentangle 可以做什么样的事情

Discrete Latent Representation

下一个要跟大家讲的应用,叫做 Discrete Latent Representation



到目前为止我们都假设这个 Embedding,它就是一个向量,这样就是一串数字,它是 Real Numbers,那它可不可以是别的东西呢



- 举例来说,它可不可以是 Binary,Binary 的好处也许是说,每一个维度,它就代表了某种特徵的有或者是没有,举例来说,输入的这张图片,如果是女生,可能第一维就是 1,男生第一维就是 0,如果有戴眼镜,就是第三维 1,没有戴眼镜 就是第三维是 0,也许我们把这个向量,这个 Embedding 变成 Binary,变成只有 0 跟 1 的数字,可以让我们再解释 Encoder 输出的时候,更为容易
- 甚至有没有可能这个向量,强迫它一定要是 One-Hot 呢,也就只有一维是 1,其他就是 0,如果我们强迫它是 One-Hot,也就是每一个东西图片丢进去,你只可以有,你的 Embedding 裡面只可以有一维是 1,其他都是 0 的话,那可以做到什么样的效果呢,也许可以做到 unSupervised 的分类,举例来说,假设你有一大堆的,假设你想要做那个手写数字辨识,你有 0 到 9 的图片,你把 0 到 9 的图片统统收集起来,Train 一个这样子的 Aauto-Encoder,然后强迫中间的 Latent Representation,强迫中间的这个 Code 啊,一定要是 One-Hot Vector,那你这个 Code 正好设个 10 维,也许每一个 One-Hot 的 Code,所以这 10 维,就有 10 种可能的 One-Hot 的 Code,也许每一种 One-Hot 的 Code,正好就对应到一个数字也说不定,所以今天如果用 One-Hot 的 Vector,来当做你的 Embedding 的话,也许就可以做到完全在没有,完全没有Llabel Data 的情况下,让机器自动学会分类

其实还有其他,在这种啊 Discrete 的 Representation 的这个,技术裡面啊,其中最知名的就是 ==VQVAE==,Vector Quantized Variational Aauto-Encoder,

VQVAE 啊,是这样子运作的,就是你输入一张图片,Encoder 呢 输出一个向量,这个向量它是一般的向量,它是 Continuous 的,但接下来你有一个 ==Codebook==



所谓 Codebook 的意思就是,你有一排向量,这排向量也是 Learn 出来的,你把 Encoder 的输出,去跟这排向量都去算个**相似度**,那你发现这件事情啊,其实跟 Self-attention 有点像,上面这个 Vector 就是 Query,下面这些 Vector 就是 Key,那接下来呢就看这些 Vector 裡面,谁的**相似度最大**,那你把相似度最大的那个 Vector 拿出来

这边就是那个,这个 Key 跟那个 Value,是等于是共用同一个 Vector



如果你把这整个 Process,用 Self-attention 来比喻的话,那就等于是 Key 跟 Value 是共同的 Vector,然后把这个 Vector 呢,丢到 Decoder 裡面,然后要它输出一张图片,然后接下来 Training 的时候,就是要让输入跟输出越接近越好

这一个 Decoder,这个 Encoder,这一个 Codebook,都是一起从资料裡面被学出来的,这样做的好处就是你就可以,你就有 Discrete 的这个 Latent Representation,也就是说这边 Decoder 的输入,一定是这边这个 Codebook,裡面的向量的其中一个,假设你 Codebook 裡面有 32 个向量,那你 Decoder 的输入,就只有 32 种可能,你等于就是让你的这个 Embedding,它是离散的,它没有无穷无尽的可能,它只有 32 种可能而已

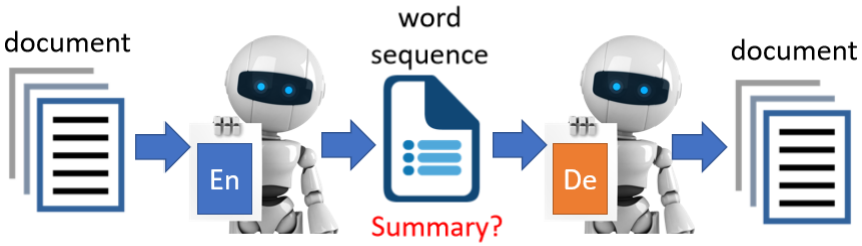
那其实像这样子的技术啊,如果你拿它 把它用在语音上,你就是一段声音讯号输进来,通过 Encoder 以后产生一个向量,接下来呢,你去计算这个相似度,把最像的那个向量拿出来丢给 Decoder,再输出一样的声音讯号,这个时候你会发现说你的 Codebook 啊,可能可以学到最基本的发音部位

举例来说 你的,这个最基本的发音单位啊,又叫做 ==Phonetic==,那如果你不知道 Phonetic 是什么的话,你就把它想成是 KK 音标,那你就会发现说,这个 Codebook 裡面每一个 Vector,它就对应到某一个发音,就对应到 KK 音标裡面的某一个符号,这个是 VQVAE

Text as Representation

那其实还有更多疯狂的想法,Representation 一定要是向量吗,能不能是别的东西

举例来说,它能不能是一段文字,是可以的

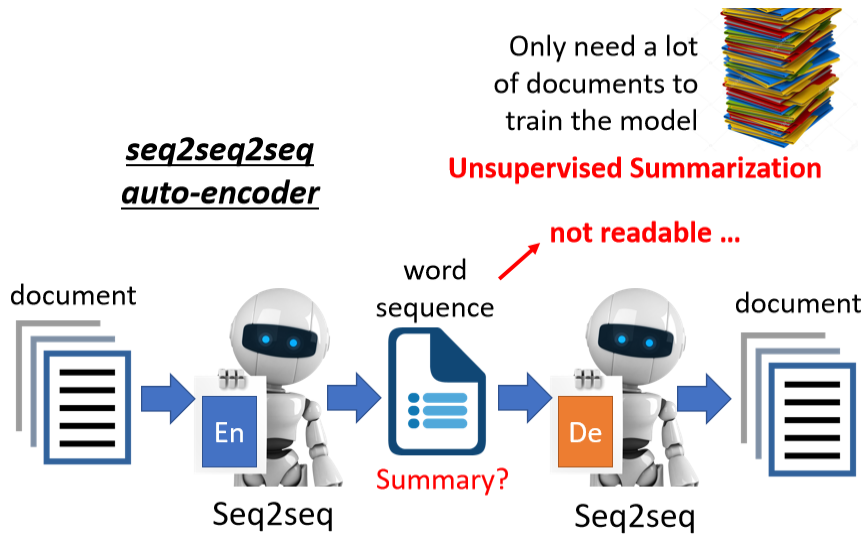


假设我们现在要做文字的 Aauto-Encoder,那文字的 Aauto-Encoder 的概念,跟语音的影像的没有什么不同,就是你有一个 Encoder,一篇文章丢进去,也许产生一个什么东西 一个向量,把这个向量丢到 Decoder,再让它还原原来的文章,但我们现在可不可以不要用向量,来当做 Embedding,我们可不可以说我们的 Embedding,就是一串文字呢

如果把 Embedding 变成一串文字,有什么好处呢,也许这串文字就是文章的摘要,因为你想想看,把一篇文章丢到 Encoder 的裡面,它输出一串文字,而这串文字,可以通过 Decoder 还原回原来的文章,那代表说这段文字,是这篇文章的精华,也就是这篇文章最关键的内容,也就是这篇文章的摘要

不过啊 这边的 Encoder,显然需要是一个 **Seq2seq 的 Model**,比如说 Transformer,因为我们这边输入是文章嘛,这边输出是一串文字嘛,这个 Decoder 输入是一串文字,输出是文章嘛,所以都是输入一串东西,输出一串东西,输入一串文字 输出一串文字,所以 Encoder 跟 Decoder,显然都必须要是一个 Seq2seq 的 Model

它不是一个普通的 Aauto-Encoder,它是一个 seq2seq2seq 的 Aauto-Encoder,它把长的 Sequence 转成短的 Sequence,再把短的 Sequence 还原回长的 Sequence,而这个 Aauto-Encoder 大家训练的时候,不需要标注的资料,因为训练 Aauto-Encoder,只需要收集大量的文章,收集大量没有标注的资料,在这边就是大量的文章就可以了



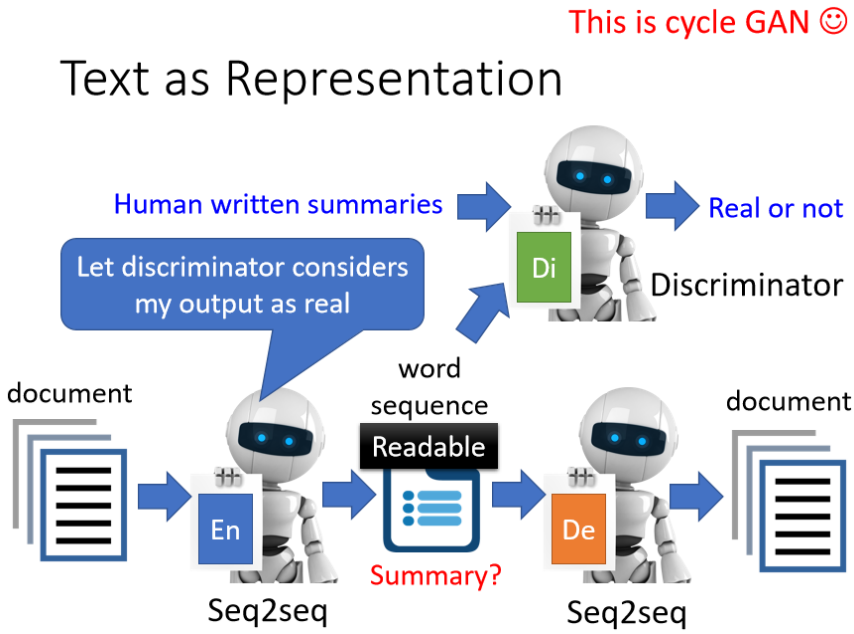
如果你真的可以训练出这个模型,如果这串文字真的可以代表摘要的话,你就是让机器自动学会做摘要这件事,让机器自动学会做,unSupervised 的 Summarization

但是真的有这么容易吗,实际上这样 **Train 起来以后发现是行不通的**,为什么,因为这两个 Encoder 跟 Decoder 之间,会发明自己的暗号啊,所以它会产生一段文字,那这段文字是你看不懂的,你看不懂的文字,这 Decoder 可以看得懂,它还原得了原来的文章,但是人看不懂,所以它根本就不是一段摘要,所以怎么办呢

再用 GAN 的概念,加上一个 Discriminator

Discriminator 看过人写的句子,所以它知道人写的句子长什么样子,但这些句子,不需要是这些文章的摘要性,另外一堆句子,所以它知道人写的句子长什么样子

然后呢,这个 Encoder 要想办法去骗过 Discriminator,Encoder 要想办法产生一段句子,这段句子不只可以透过 Decoder,还原回原来的文章,还要是 Discriminator 觉得像是人写的句子,期待通过这个方法,就可以强迫 Encoder,不只产生一段密码可以给 Decoder 去破解,而是产生一段人看得懂的摘要



那你可能会问说,这个 Network 要怎么 Train 啊,这个 Output 是一串文字哦,那这个文字要怎么接给 Discriminator,跟这个 Decoder 呢,告诉你,看到你没办法 Train 的问题,就用 RL 硬做,这样这边就是 RL 硬做就结束了这样子

你可能会觉得这个概念有点像 **CycleGAN**,没错 你可以想这根本就是 CycleGAN,就是这是一个 Generator,这是另外一个 Generator,这是 Discriminator,你要输入跟输出越接近越好,其实这根本就是 CycleGAN,我们只是从 Aauto-Encoder 的角度,来看待 CycleGAN 这个想法而已,

那实际上做的结果是怎么样呢,以下是真正 Network 输出的结果啦,你给它读一篇文章,然后它就用 Aauto-Encoder 的方法,拿 300 万篇文章做训练以后,然后看看给它一篇新的文章,它可不可以是,那个 Encoder 的输出的句子,是不是就是人可以看得懂的摘要

- ***Document***:澳大利亞今天與13個國家簽署了反興奮劑雙邊協議,旨在加強體育競賽之外的藥品檢查並共享研究成果 .....
- ***Summary***:
  - Human:澳大利亞與13國簽署反興奮劑協議
  - Unsupervised:澳大利亞加強體育競賽之外的藥品檢查
- ***Document***:中華民國奧林匹克委員會今天接到一九九二年冬季奧運會邀請函,由於主席張豐緒目前正在中南美洲進行友好訪問,因此尚未決定是否派隊赴賽 .....
- ***Summary***:
  - Human:一九九二年冬季奧運會函邀我參加
  - Unsupervised:奧委會接獲冬季奧運會邀請函

举例来说,给 Encoder 看这篇文章,它的输出是,澳大利亚加强体育竞赛之外的药品检查,看起来还可以,那这边有一个特别强的啦,就是这篇文章是,中华民国奥林匹克委员会,今天接到一九九二年冬季奥运会邀请函,丢给 Encoder 之后,它的输出是奥委会接获冬季奥运会邀请函,它知道把奥林匹克委员会,自己就缩写成奥委会,这个不知道怎么回事,它自己就学到了这件事情

当然很多时候,它也是会犯错的,我特别喜欢举这种极其犯错的例子

- ***Document***:據此間媒體27日報道,印度尼西亞蘇門答臘島的兩個省近日來連降暴雨,洪水泛濫導致塌方,到26日為止至少已有60人喪生,100多人失蹤 .....
- ***Summary***:
  - Human:印尼水災造成60人死亡
  - Unsupervised:印尼門洪水泛濫導致塌雨
- ***Document***:安徽省合肥市最近為領導幹部下基層做了新規定:一律輕車簡從,不準搞迎來送往、不準搞層層陪同 .....
- ***Summary***:
  - Human:合肥規定領導幹部下基層活動從簡
  - Unsupervised:合肥領導幹部下基層做搞迎來送往規定:一律簡

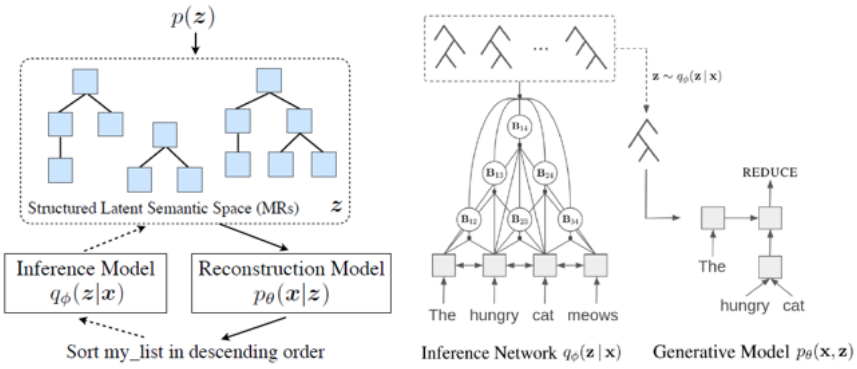
举例来说,你给它读这篇文章,印度尼西亚苏门答腊岛近日来连降暴雨,机器产生的摘要是什么呢

Encoder 的输出是,印尼门洪水泛滥,印尼门是什么东西呢,大概就是印度尼西亚苏门的缩写啦,可能人类写的句子裡面,常常出现罗生门 (风二门),等等什么门,所以机器觉得,Encoder 觉得印度尼西亚苏门,应该可以缩写成印尼门

那有时候它也会产生莫名其妙的句子啊,比如说把这篇文章给机器读了以后,Encoder 的输出是,合肥领导干部下基层做搞迎来送往规定一律简,不知道在说些什么,总之是个句子 不知道在说些什么,好 所以这个例子只是想要告诉你说,我们确实有可能,拿一段文字来当做 Embedding

其实还有更狂的,我还看过有拿 Tree Structure,当做 Embedding,就一段文字把它变成 Tree Structure,再用 Tree Structure 还原一段文字,

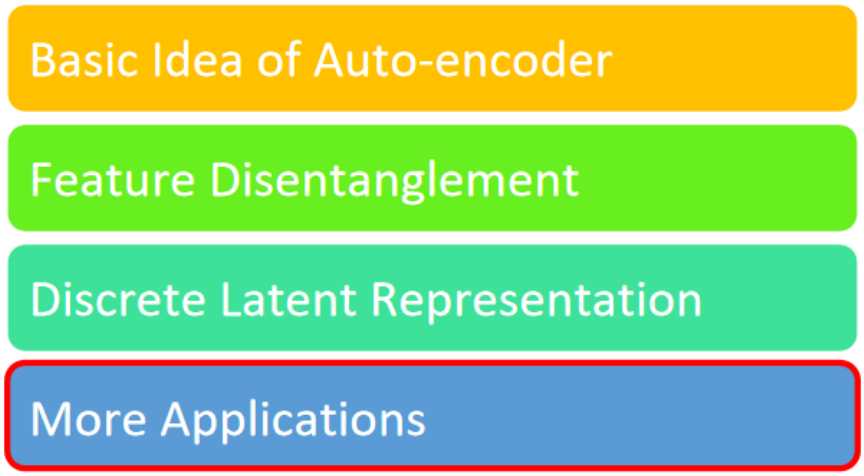
Tree as Embedding



https://arxiv.org/abs/1806.07832 https://arxiv.org/abs/1904.03746

好 我把 Reference 列在这边给大家参考

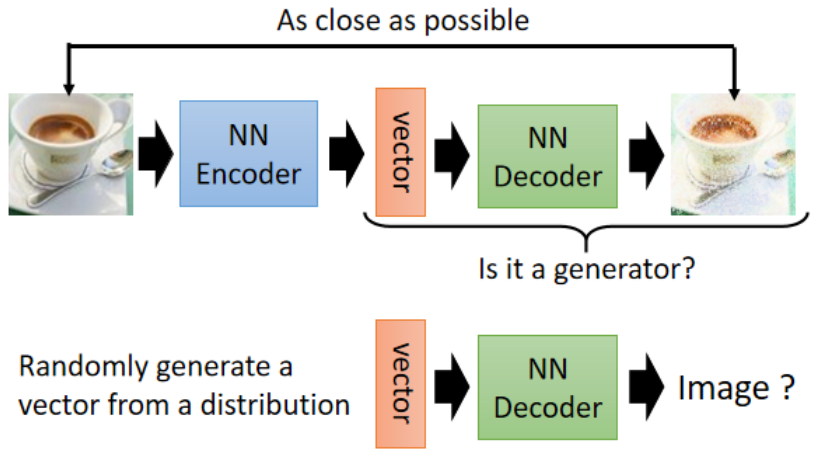
More Applications



接下来啊,还有 Aauto-Encoder 更多的应用,Aauto-Encoder 还可以拿来做些什么事情呢,举例来说,我们刚才用的都是 Encoder,那其实 Decoder 也有作用

Generator

你把 Decoder 拿出来,这不就是一个 **Generator** 吗,我们说 Generator,不是就是要吃一个向量,产生一个东西,比如说一张图片吗,而 Decoder 不正好是吃一个向量,产生一张图片吗,所以 Decoder,你可以把它**当做一个 Generator 来使用**



With some modification, we have **variational auto-encoder (VAE).**

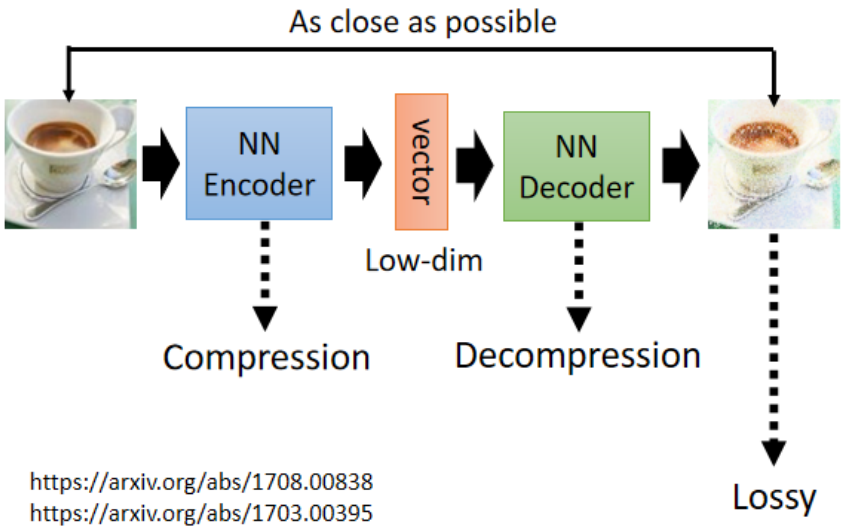
你可以从一个已知的 Distribution,比如说 Gaussian Distribution,Sample 一个向量,丢给 Decoder,看看它能不能够输出一张图

事实上在我们之前,在讲这个 Generative Model 的时候,其实有提到说除了 GAN 以外,还有另外两种 Generative 的 Model,其中一个就叫做 VAE,Variarional 的 Aauto-Encoder,你看它名字裡面的 Aauto-Encoder,显然是跟 Aauto-Encoder 非常有关係的,它其实就是把 Aauto-Encoder 的 Decoder 拿出来,当做 Generator 来用,那实际上它还有做一些其他的事情啊,至于它实际上做了什么其他的事情,就留给大家自己研究

所以 Aauto-Encoder Train 完以后,也顺便得到了一个 Decoder

Compression

Aauto-Encoder 可以拿来**做压缩**



我们今天知道说你在做图片,我们图片如果太大的话,也会有一些压缩的方法,比如说 JPEG 的压缩,而 Aauto-Encoder 也可以拿来做压缩,你完全可以把 Encoder 的输出,当做是一个压缩的结果,因为一张图片,是一个非常高维的向量,而一般我们 Encoder 的输出,是一个非常低维的向量,你完全可以把那个向量,看作是一个压缩的结果

所以你的 Encoder 做的事情,就是压缩,你的 Decoder 做的事情,就是解压缩

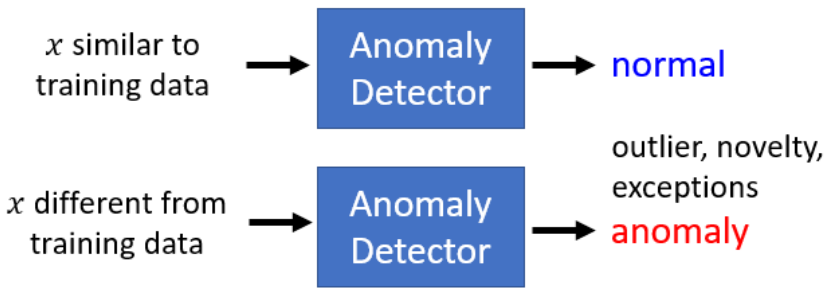
只是这个压缩啊,它是那种 lossy 的压缩,所谓 **lossy 的压缩就是它会失真**,因为在 Train Aauto-Encoder 的时候,你没有办法 Train 到说,输入的图片跟输出的图片,100% 完全一模一样啦,它还是会有一些差距的

所以这样子的 Aauto-Encoder 的压缩技术,你拿这样子的技术来做压缩,那你的图片是会失真的,就跟 JPEG 图片会失真一样,用这个 Aauto-Encoder 来做压缩,你的图片也是会失真的

Anomaly Detection

那接下来,就是我们在作业裡面要使用的技术,在作业裡面我们会拿 Aauto-Encoder,来做 Anomaly 的 Detection,那我在规划作业的时候,其实就是想要 Aauto-Encoder 出一个作业,那 Aauto-Encoder 的技术很多,那最后我决定做 Anomaly 的 Detection，因为这个是你在非常多的场合,都有机会应用到的一个技术

- Given a set of training data $\{x^1, x^2, \cdots, x^N\}$
- Detecting input x is *similar* to training data or not.



Anomaly 的 Detection ,假设你有一堆的训练资料,这边用 X1 到 XN 来表示我们的训练资料,而 Anomaly Detection,它的中文通常翻译成异常检测

异常检测要做的事情就是,**来了一笔新的资料,它到底跟我们之前在训练资料裡面看过的资料,相不相似**呢,也就是说你需要找出,你需要有一个异常检测的系统,这个异常检测的系统,是透过大量你已经看过的资料训练出来的

- 给它一笔新的资料,如果这笔新的资料,看起来像是训练资料裡面的 Data,就说它是正常的
- 如果看起来不像是训练资料裡面的 Data,就说它是异常的

那其实 Anomaly,Anomaly 这个词啊,有很多不同的其他的称呼,比如说有时候你会叫它 Outlier,有时候你会叫它 Novelty,有时候你会叫它 Exceptions,但其实指的都是同样的事情,你就是要看某一笔新的资料,它跟之前看过的资料到底相不相似,但是所谓的**相似这件事啊,其实并没有非常明确的定义**,它是见仁见智的,会根据你的应用情境而有所不同



举例来说

- 假设现在你的训练资料这个都是雷丘,那这个皮卡丘就算是异常的东西
- 但是假设你的训练资料裡面,你所有的动物都是皮卡丘,那雷丘就是异常的东西,所以我们并不会说,某一个东西它一定就是 Normal,一定就是 Anomaly,我们不会说某个东西它一定是正常或异常,它**是正常或异常,取决于你的训练资料长什么样子**
- 或者是说假设你的训练资料裡面,通通都是宝可梦,那雷丘跟皮卡丘通通都算是正常的,而可能数码宝贝,亚古兽知道吗,这应该是亚古兽 对不对,亚古兽算是异常的

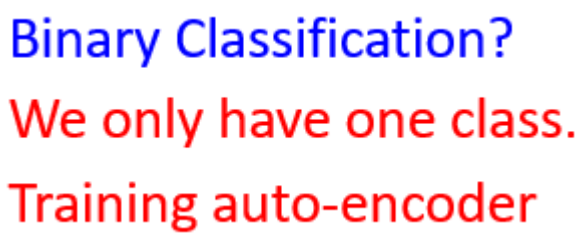
那个这个异常检测有什么样的应用呢

- Fraud Detection
  - Training data: credit card transactions, x: fraud or not
  - Ref: https://www.kaggle.com/ntnu-testimon/paysim1/home
  - Ref: https://www.kaggle.com/mlg-ulb/creditcardfraud/home
- Network Intrusion Detection
  - Training data: connection, x: attack or not
  - Ref: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
- Cancer Detection
  - Training data: normal cells, x: cancer or not?
  - Ref: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/home

- 举例来说,它可以来做诈欺侦测,假设你的训练资料裡面,有一大堆信用卡的交易纪录,那我们可以想像说,多数信用卡的交易都是正常的,那你拿这些正常的信用卡训练的交易纪录,来训练一个异常检测的模型,那有一笔新的交易纪录进来,你就可以让机器帮你判断说,这笔纪录算是正常的 还是异常的,所以这种异常检测的技术,可以拿来做诈欺侦测
- 或者是它可以拿来做网路的这个侵入侦测,举例来说,你有很多连线的纪录资料,那你相信多数人连到你的网站的时候,他的行为都是正常的,多数人都是好人,你收集到一大堆正常的连线的纪录,那接下来有一笔新的连线进来,你可以根据过去正常的连线,训练出一个异常检测的模型,看看新的连线,它是正常的连线 还是异常的连线,它是有攻击性的 还是正常的连线,或者是它在医学上也可能有应用,

> 你收集到一大堆正常细胞的资料,拿来训练一个异常检测的模型,那也许看到一个新的细胞,它可以知道这个细胞有没有突变,也许有突变,它就是一个癌细胞等等

那讲到这边有人可能会想说,Anomaly Detection 异常检测的问题,我们能不能够把它当做**二元分类**的问题来看啊



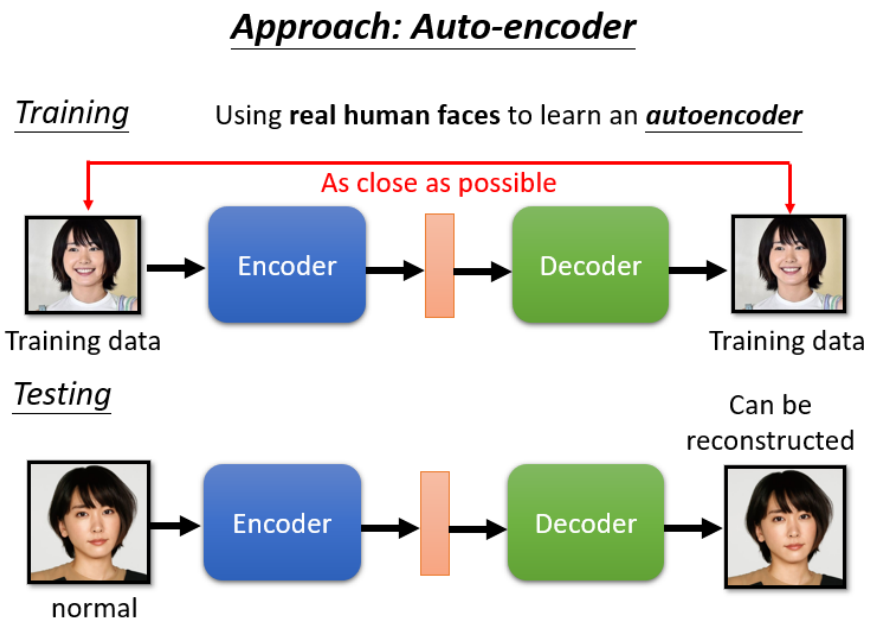
你说你要做诈欺侦测,你就收集一大堆正常的信用卡纪录,一堆诈欺的信用卡纪录,训练一个 Binary 的 Classifier,就结束啦,就这样子不是吗,

比较**难点就是你要收资料**

这种异常检测的问题它的难点,正在就在收资料上面,通常你**比较有办法收集到正常的资料,你比较不容易收集到异常的资料**,你可能有一大堆信用卡交易的纪录,但是多数信用卡交易的纪录可能都是正常的,异常的资料相较于正常的资料,可能非常地少,甚至有一些异常的资料混在正常的裡面,你也不太可,你可能也完全没有办法侦测出来,所以在这一种异常检测的问题裡面

我们往往假设,我们有一大堆正常的资料,但我们几乎没有异常的资料,所以它**不是一个一般的分类的问题**,这种分类的问题又叫做 ==One Class 的分类问题==,就是我们只有一个类别的资料,那你怎么训练一个模型,因为你想你要训练一个分类器,你得有两个类别的资料,你才能训练分类器啊,如果只有一个类别的资料,那我们可以训练什么东西,这个时候就是 Aauto-Encoder,可以派得上用场的时候了

举例来说,假设我们现在想要做一个系统,这个系统是要侦测说一张图片

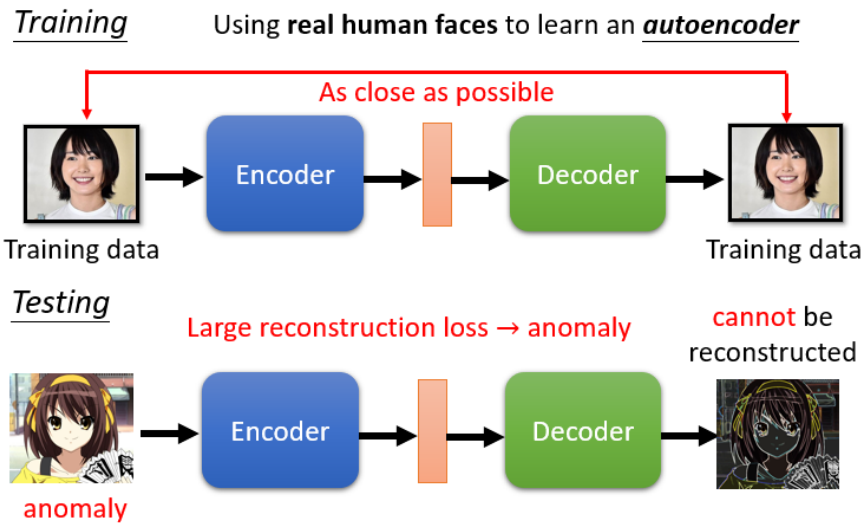


举例来说,它是不是真人的人脸,那你可以找到一大堆图片,它都是真正的人脸,那我们就拿这些真人的人脸,来训练一个 Aauto-Encoder

这个是你老婆的照片,那你可以拿它来训练一个 Aauto-Encoder,那你训练完这个 Aauto-Encoder 以后,在测试的时候,如果进来的也是你老婆的照片,那因为在训练的时候有看过这样的照片,所以它可以顺利地被还原回来

你可以计算这一张照片通过 Encoder,再通过 Decoder 以后,它的变化有多大,你可以去**计算这个输入的照片,跟这个输出的照片,它们的差异有多大**,如果差异很小,你的 Decoder 可以顺利地还原原来的照片,代表这样类型的照片,是在训练的时候有看过的,不过反过来说,假设有一张照片是训练的时候没有看过的

举例来说这根本不是人的照



她是那个凉宫春日,但是她不是真人,她是一个动画的人物,她是二次元的人物,一个二次元人物的照片,输入 Encoder 再输出 Decoder 以后,

因为这是没有看过的东西,这是训练的时候没有看过的照片,那你的 Decoder,就很难把它还原回来,如果你计算输入跟输出的差异,发现差异非常地大,那就代表说,现在输入给 Encoder 的这张照片,可能是一个异常的状况,可能是训练的时候没有看过的状况,所以你就可以看 reconstruction 的 loss,这个 reconstruction 的好坏,来决定说你现在在测试的时候,看到这张照片,是不是训练的时候有看过同类型的照片,

这个就是我们,好 那这个就是我们在作业裡面,要大家做的事情啦

More about Anomaly Detection

那这个异常检测啊,其实也是另外一门学问,那我们课堂上就没有时间讲了,异常检测不是只能用 Aauto-Encoder 这个技术,Aauto-Encoder 这个技术,只是众多可能方法裡面的其中一个,我们拿它来当做 Aauto-Encoder 的作业,因为我相信,你未来有很多的机会用得上异常检测这个技术,那实际上有关异常检测更完整的介绍,我们把过去上课的录影放在这边,给大家参考,

•Part 1: https://youtu.be/gDp2LXGnVLQ

•Part 2: https://youtu.be/cYrNjLxkoXs

•Part 3: https://youtu.be/ueDlm2FkCnw

•Part 4: https://youtu.be/XwkHOUPbc0Q

•Part 5: https://youtu.be/Fh1xFBktRLQ

•Part 6: https://youtu.be/LmFWzmn2rFY

•Part 7: https://youtu.be/6W8FqUGYyDo

那以上就是有关 Aauto-Encoder 的部分

Adversarial Attack P1-Basic Conception

Adversarial Attack 要讲这个主题,我们这边要讲的是,我们在作业裡面,我们已经训练了非常多,各式各样的类神经网路,那我们当然期待说,我们可以把这些技术用在真正的应用上

- You have trained many neural networks.
- We seek to deploy neural networks in the real world.
- Are networks robust to the inputs that are built to fool them?
  - Useful for spam classification, malware detection, network intrusion detection, etc.



但是光是把这些 Network 用在真正的应用上,要把这些 Network 用在真正应用上,光是它们正确率高是不够的,它们还需要什麼能力呢

他们需要能够**应付来自人类的恶意**,有时候你的 Network,它的工作是為了要侦测一些有恶意的行為,如果今天它的工作是要侦测有恶意的行為,这些它要侦测的对象,会去想办法骗过 Network,Network 在一般的情况下,都可以得到高的正确率是不够的,它要**在有人试图想要欺骗它的情况下,也得到高的正确率**

举例来说 我们今天都会用 Network,来做 E-Mail 的 Filtering,你会用 Network 来做侦测一封邮件,它是不是垃圾邮件,那今天对於一个垃圾邮件的发信者而言,他也会想尽办法避免,他的邮件被分类為垃圾邮件,如果今天有人他想办法去更改邮件的内容,想要去欺骗过 Network 的话,那 Network 到底能不能不要被欺骗呢

所以今天我们希望我们的类神经网路,它光是正确率高还不够,我们希望它可以应付来自人类的恶意

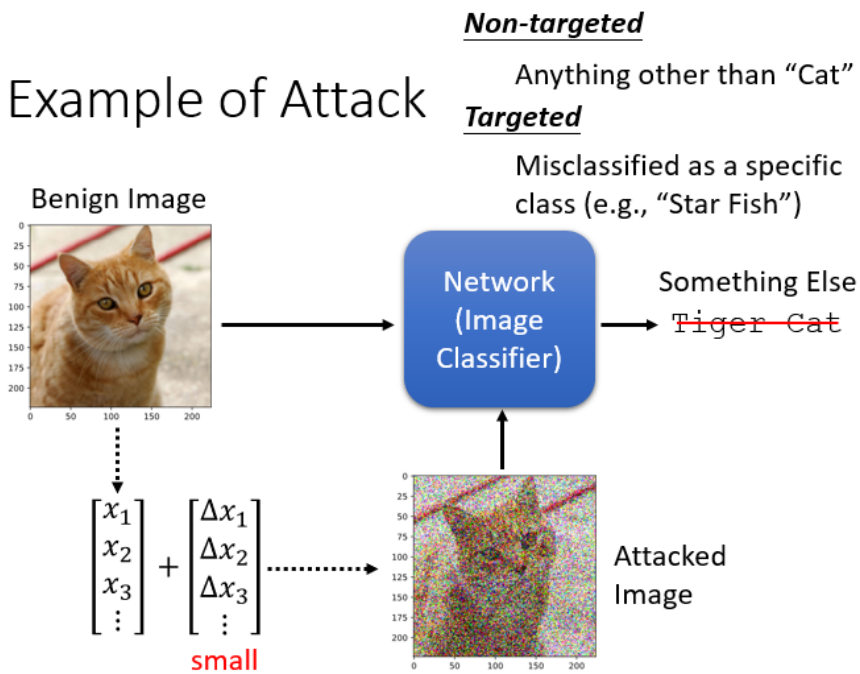
举例来说 这个是蚁王



牠叫做梅路艾姆,牠是站在生物顶点,牠是生物顶点的一个存在,牠非常地强,人类没有办法打赢牠,那牠会消灭掉所有的人类,但是人类并没有跟牠打,**人类是不讲武德的**,人类就直接放一个核弹就把牠炸死,然后故事就结束了 就这样

Example of Attack

我们先来看一个真正的例子,我们今天在好多个作业裡面,我们都已经训练了影像辨识的模型,也就是说你有一个影像辨识的系统,给它一张照片,它可以告诉我们说,这张照片属於什麼样的类别



那今天我们要做的事情是,在这张**照片上面加入一个非常小的杂讯**,一张照片可以被看作是一个非常长的向量,我们在这个非常长的向量上加入,每一个维度都加入一个小小的杂讯,把这个有加入杂讯以后的照片丢到 Network,看看会发生什麼样的事情

那一般这个**杂讯都非常非常地小**,小到什麼地步呢,最好小到人肉眼没有办法看出来,所以这个例子裡面加的杂讯其实太大了,一般这个杂讯是小到人肉眼看不出来的

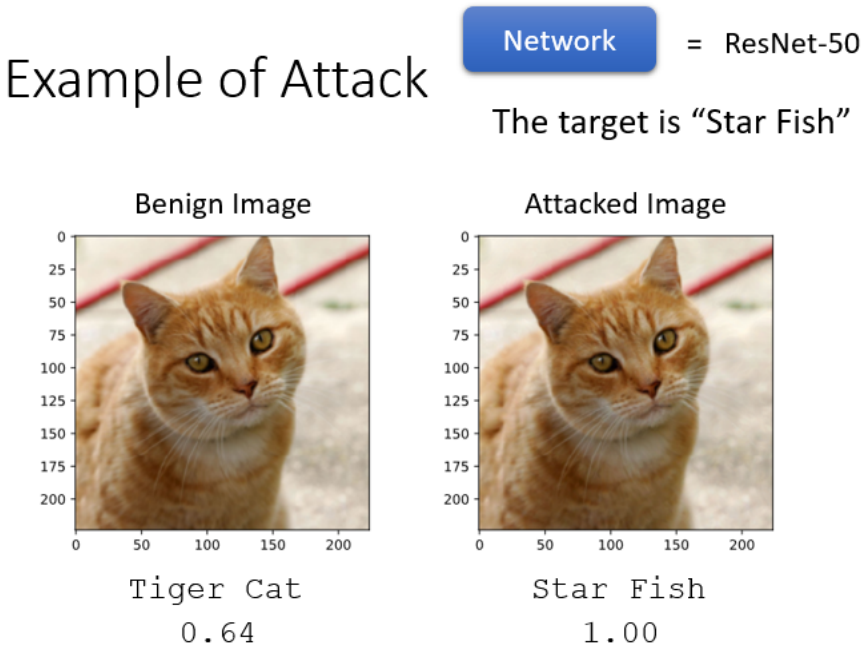
有被加杂讯的照片叫做 **Attacked Image**,有被攻击的照片,那还没有被加杂讯的照片呢,我们一般就叫做 **Benign Image**,它是好的 Image,它是还没有被攻击的图片

Benign Image 丢到 Network 裡面,它的输出是猫,那我们期待说 Attacked Image,就我们现在是攻击方 我们是坏人,我们希望 Attacked Image 丢到 Network 裡面,它的输出不可以是猫,要变成其他的东西

那攻击呢 大致上可以分成**两种类型**,一种是**没有目标的攻击**,没有目标的攻击是,原来的答案是猫,只要你能够让 Network 的输出不是猫,你就算是成功了

但是还有另外一种更困难的攻击,是**有目标的攻击**,也就是说我们希望 Network,不止它输出不能是猫,还要输出别的东西,比如说 我们希望加了一个杂讯以后,Network 输出呢 是海星,它把猫错误判断成一隻海星,才算是攻击成功,好 那这样子的攻击真的有可能做到吗,我们真的可以加入一个人肉眼看不到的杂讯,去改变 Network 的输出吗

实际上是可以的,这边用的 Network 并不是一个很弱的 Network,它是 50 层的 ResNet



当我们把一个 Benign Image 没有被攻击的图片,丢到 50 层的 ResNet 的时候,它的输出是 Tiger Cat

你知道今天这种影像辨识的系统,它的输出都不只会告诉你它是什麼动物,还会告诉你它是哪一个品种的动物,所以它给它这隻猫,它不只说它是 Cat,还说它是 Tiger Cat,不过据说这个答案其实是错的,据说这个猫不是 Tiger Cat,不过没有关係,反正它都认得出这个是一隻猫就对了,至少知道是一个猫科的动物

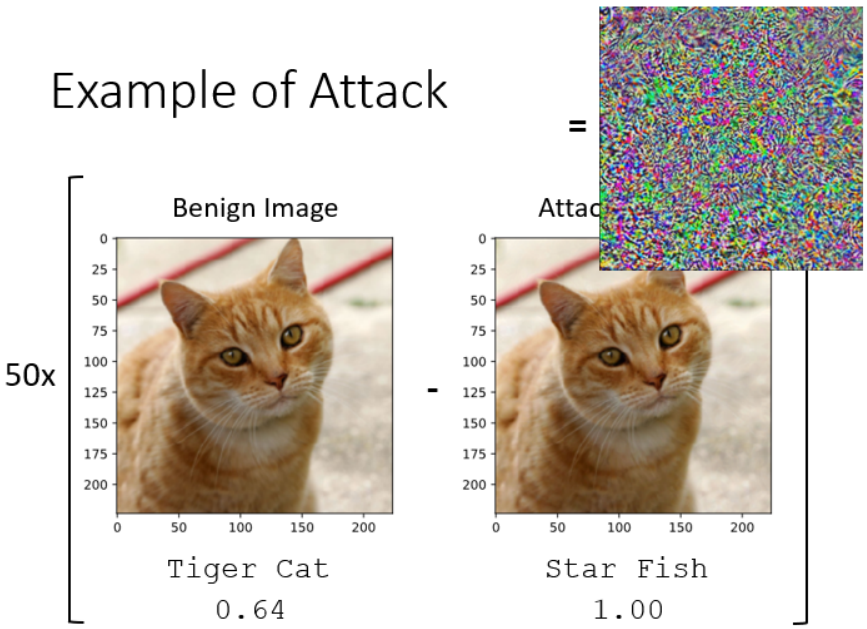
那它还有一个信心的分数,**信心的分数(置信度)是 0.64**,这个信心的分数是什麼呢,这个信心的分数就是,做完那个 Soft Mask 以后,得到的那个分数,就假设你的影像分类的类别有 2000 类,2000 个类别都会有一个分数,那这个分数呢 一定介於 0 到 1 之间,而且 2000 个类别的分数合起来,会刚刚好是 1

那既然有 2000 个类别那麼多,Tiger Cat 可以拿到 0.64 的分数,那其实算是挺高的

接下来呢 我们在这个 Benign Image 上面,加入一些杂讯,我们现在希望成功攻击的目标,是把 Tiger Cat 变成海星,而被攻击以后的图片长的是这个样子的,你可能问说杂讯加在哪裡呢,杂讯已经加进去了,但它非常非常地小,**小到人的肉眼根本没有办法看出来**

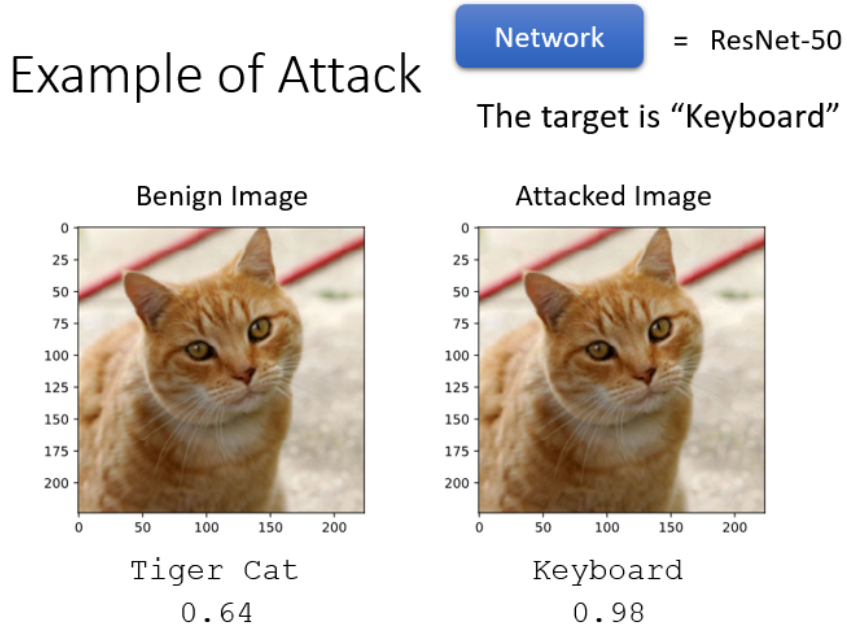
把这张图片丢到 ResNet 以后,会发生什麼事呢,**ResNet 的 Output 变成 Star Fish,而且它的信心分数是 100 %** 啊,本来它还没有那麼确定这是不是一隻猫,现在它百分之百确定,它就是海星

那為了要证明说,这两张照片还是有一些不一样的,我们把它做一下**相减**



光做相减是不够的,做完相减以后,还要把它们的差距放大 50 倍,你会得到这样子的结论,所以这两张照片确实有些不一样,我并不是把同一张照片复製两次来骗你,它们是有一点不一样的,但是人根本看不出,这点不一样会造成什麼样的影响,但是对 ResNet 而言,它却有了天差地远的输出

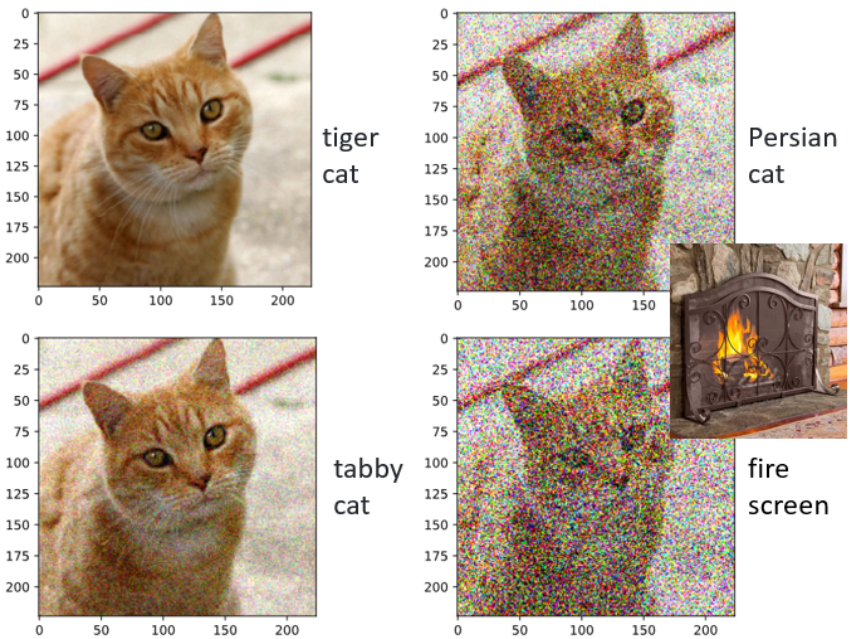
那也许有人会觉得说,啊 也许是因為猫跟海星有什麼特别的关係



我们可以把猫变成海星只是一个特例,这不是一个特例,你可以**把这隻猫轻易地变成任何的东西**,我完全可以加上另外一个杂讯,就让这隻猫变成一个键盘,它一样信心分数高达 0.98,本来不太确定它是不是猫,现在加入另外一个杂讯以后,Network 98% 确定它就是一个键盘

那有人可能会觉得说,欸 怎麼会发生这麼离谱的行為,会不会是**这个 Network 太烂了？**,我要告诉你 它可是有 50 层的喔,它可不是一个非常烂的 Network

如果你加入的只是**一般的杂讯,它并不一定会犯错**,这个是原来的图片,我们现在加入一个杂讯,这个杂讯是你肉眼可见的



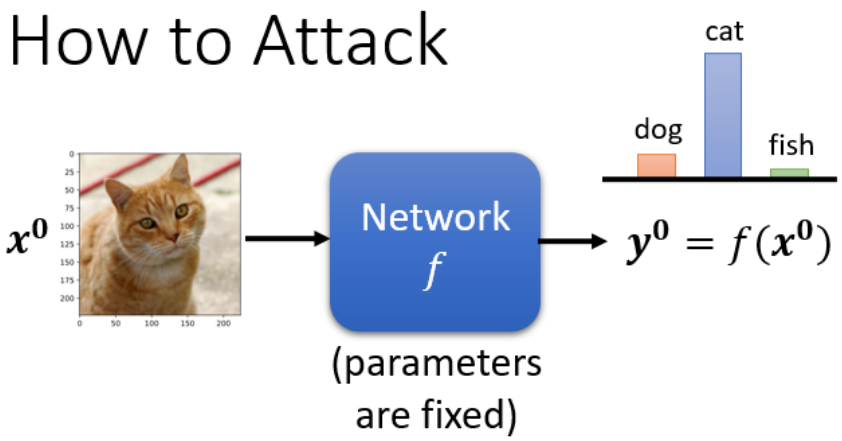
- 你可以很明显地看到,这张图片裡面被加入了杂讯,这个时候 ResNet 觉得,它看到的是 Tabby Cat,这可能才是正确答案,但无论如何 它都知道是猫科动物

- 把杂讯加得更大一点,它说这是 Persian Cat 这是波斯猫,可能杂讯加得大一点,这个猫看起来毛茸茸的,所以 ResNet 觉得它看到了波斯猫
- 把杂讯再加更大一点,你可能已经不知道这是什麼东西了,这个时候 ResNet 说,它看到了 Fire Screen,Fire Screen 是什麼呢,我 Google 了一下发现,Fire Screen 长这个样子,这裡完全可以理解机器為什麼会犯错,它觉得前面的杂讯是这个屏风,而后面这个橙色的猫就是火焰

它虽然犯错 它错的是有尊严的,它错的是有道理的,但不知道為什麼,加入一个人肉眼看不到的杂讯的时候,它却產生了天差地远的结果

How to Attack

那接下来我们在讲為什麼这件事会发生之前,我们来看看刚才所说的攻击,究竟是如何做到的,我们到底是**怎麼加入了一个非常微小的杂讯,可以让 Network 產生非常错误的结果**呢

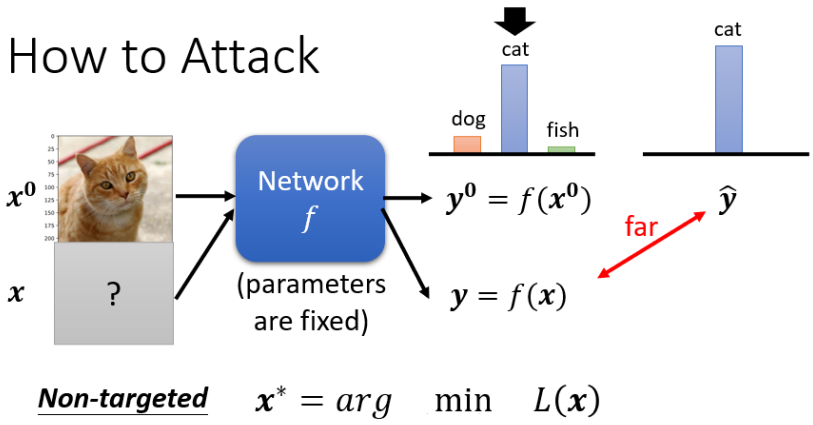


而这个是我们的 Network,它是一个 Function 我们叫它 f,这个 Function 输入是一张图片,我们叫它 x^0,它的输出是一个 Distribution,这个是这个分类的结果,那我们叫它 y^0

那我们这边假设 **Network 的参数就是固定的**,我们不讨论 Network 的参数的部分,Network 的参数不是我们今天的重点,所以它是固定的

- 如果是 Non-Targeted Attack 的话,我们要怎麼找出 Non-Targeted Attack 的杂讯呢

  我们现在要做的目标就是,我们要**找到一张新的图片**



$$\boldsymbol{x}^0 \quad \boldsymbol{x} \quad \boldsymbol{y}^0 = f(\boldsymbol{x}^0) \quad \hat{\boldsymbol{y}} \quad \boldsymbol{y} = f(\boldsymbol{x})$$

Non-targeted $\quad \boldsymbol{x}^* = arg \quad \min \quad L(\boldsymbol{x})$

$$L(\boldsymbol{x}) = -e(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

这张新的图片呢 我们用 x 来表示,当我们把 x 丢到这个 Network f 的时候,它的输出是 y,我们正确的答案叫做 ŷ,我们**希望 y 跟 ŷ 它的差距越大越好**

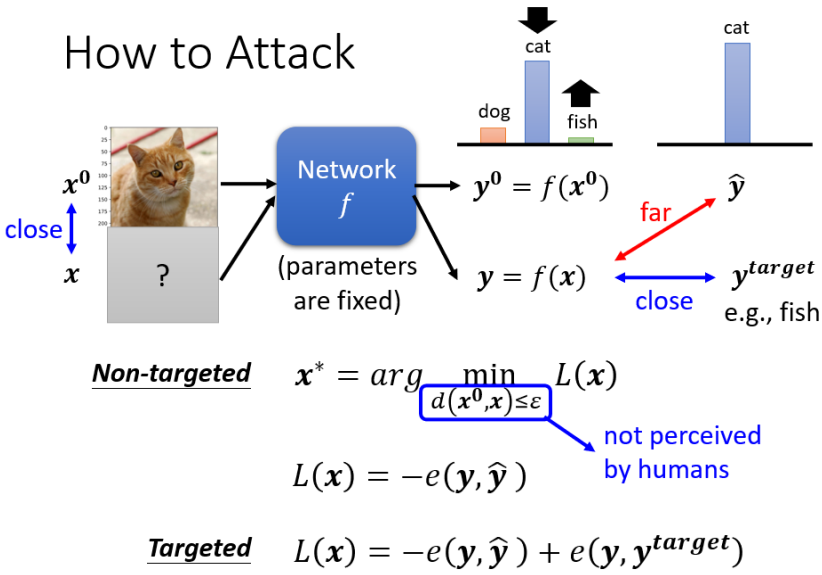
那怎麼做到这件事呢,怎麼找到这个 x,丢到一个 Network 裡以后,它產生的 y 跟 ŷ 差距越大越好呢?

我们一样要**解一个 Optimization 的问题**,这个跟我们训练的 Network,其实是非常类似的

我们先定一个 Loss Function,这个 Loss Function 呢,叫做 L,这个 L 呢 是 y 跟 ŷ 之间的差距,**取一个负号**,举例来说,我们一般在做这个 Classification 的时候,我们训练的目标 y 跟 ŷ,都是看它的 Cross Entropy,那我们这个 -e(y, ŷ),这一项代表的就是 y 跟 ŷ 之间的 Cross Entropy

但是我们希望这个 **Cross Entropy 越大越好**,所以我们今天在 Cross Entropy 前面加一个负号,那这个负的 Cross Entropy 就是我们的 Loss,而我们期望这个 Loss 越小越好,我们希望找到一个 x,x 可以让 L(x) 越小越好,L(x) 越小 就代表说 y 跟 ŷ,它们的 Cross Entropy 越大,也就是 y 跟 ŷ 它们的距离越大,这个是没有目标的攻击

- 如果是有目标的攻击的话



那我们会先设定好我们的目标,我们用 y^{target} 来代表我们的目标,那 ŷ 其实是一个 One-Hot Vector,y^{target} 也是一个 One-Hot Vector,那我们现在希望这个 **y 不止跟 ŷ 越远越好,我们还要跟 y^{target} 越近越好**

所以假设你的 y^{target} 是一个 Fish,那你希望你输出的这个 y 啊,它不止 Cat 的机率越低越好,Fish 的机率还要越高越好,那你的 Loss Function 就写成这样 $$ $L(x)=-e(y,y\hat{})+e(y,y^{target})$

$$ 我们的 Loss Function

- 是负的 y 跟 ŷ 之间的 Cross Entropy,希望这一项越大越好
- 同时你又希望 y 跟 y^{target},它们越小越好
- 你把这两项加起来就是你的 Loss,你希望找一个 x,去 Minimize 这个 Loss

但光是找一个 x,去 Minimize Loss 是不够的,因為我们其实还期待说,我们加入的杂讯越小越好,也就是我们新找到的图片,可以欺骗过 Network 的图片,跟原来的图片要越相近越好,**x 跟 x^0 要越近越好**

所以我们在解这个,Optimization 的 Problem 的时候,我们还会**多加入一个限制**,这个限制是 $$ $d(x^0,x)\leq\varepsilon$ $$ 它的意思就是,我们希望 x 跟 x^0 之间的差距,小於某一个 Threshold,小於某一个閾值,那这个閾值是根据什麼东西来决定的呢,通常就是**根据人类的感知能力来决定**

如果 x^0 跟 x 之间的差距大於 Σ,我们假设人就会看到这个杂讯,人就会发现有一个杂讯存在,所以我们要让 x^0 跟 x 它的差距,小於等於 Σ,小於等於人类可以感知的极限,那我们就可以產生一张图片,人类看起来 x 跟 x^0 是一模一样的,但產生的结果对 Network 来说,是非常不一样的

好 那怎麼计算 x 跟 x^0 之间的差距,它们之间的距离呢?d(x^0,x) 就代表它们之间的距离

有各式各样不同的算法,那為了等一下符号的方便起见呢,我们假设 **x 是一个向量**,因為它是个图片 所以它是个向量嘛,**x^0 是另外一张图片,它也是一个向量**,这两个向量**相减 我们叫它 Δx**

Non-perceivable

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix} - \begin{bmatrix} x_1^0 \\ x_2^0 \\ x_3^0 \\ \vdots \end{bmatrix} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \vdots \end{bmatrix}$$

$\boldsymbol{x}$ $\boldsymbol{x^0}$ $\Delta\boldsymbol{x}$

$$d(\boldsymbol{x^0}, \boldsymbol{x}) \leq \varepsilon$$

- L2-norm

$$d(\boldsymbol{x^0}, \boldsymbol{x}) = \|\Delta \boldsymbol{x}\|_2$$
$$= (\Delta x_1)^2 + (\Delta x_2)^2 + (\Delta x_3)^2 \cdots$$

- L-infinity

$$d(\boldsymbol{x^0}, \boldsymbol{x}) = \|\Delta \boldsymbol{x}\|_\infty$$
$$= max\{|\Delta x_1|, |\Delta x_2|, |\Delta x_3|, \dots\}$$

那这个距离啊

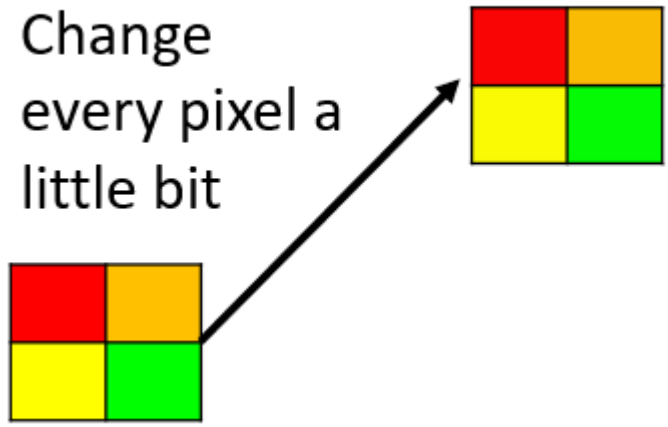
- 你可以定 L2-Norm 当做它们的距离,也就是说你可以计算 Δx 的 L2-Norm,Δx 的 L2-Norm 就是把这个 Δx 的第一位,拿出来取平方,第二位拿出来取平方,第三位拿出来取平方,在这边你其实要开根号也可以啦,就看你的 L2-Norm 的定义是怎样,你要开根号也是可以的
- 另外还有一个定义呢 是 L-Infinity,L-Infinity 是怎麼看的呢,它就是把这个 Δx 拿来,然后看裡面哪一个维度它的绝对值最大,那这一个就是L-Infinity,就把 Δx1 Δx2 Δx3,也就是 Δx 的每一维都拿出来取绝对值,看谁最大,最大的那一个就代表 x 跟 x^0 之间的距离

那有各种不同的方法,可以计算两张图片之间的距离,但是我们在决定要使用哪一种方法,来计算图片的距离的时候,其实我们应该**把人类的感知把它考虑进来**

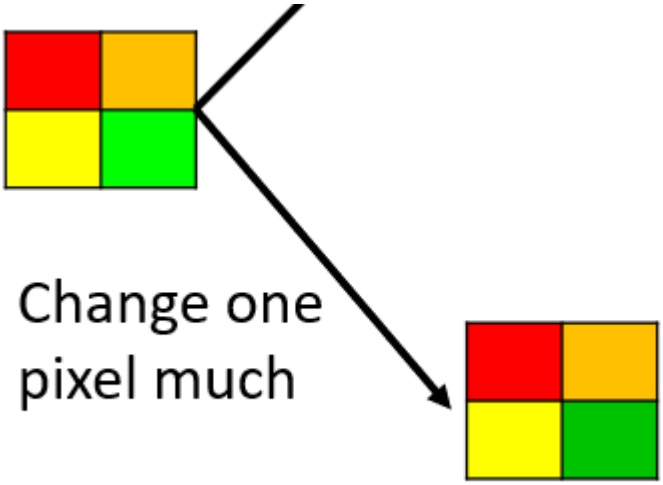
那 L2 跟 L-Infinity 到底哪一个,在 Attack 的时候是比较好的距离呢,以下我举一个例子来跟大家说明

这是一张图片,假设这个图片只有四个 Pixel 而已,现在啊,我们把这张图片做两种不同的变化

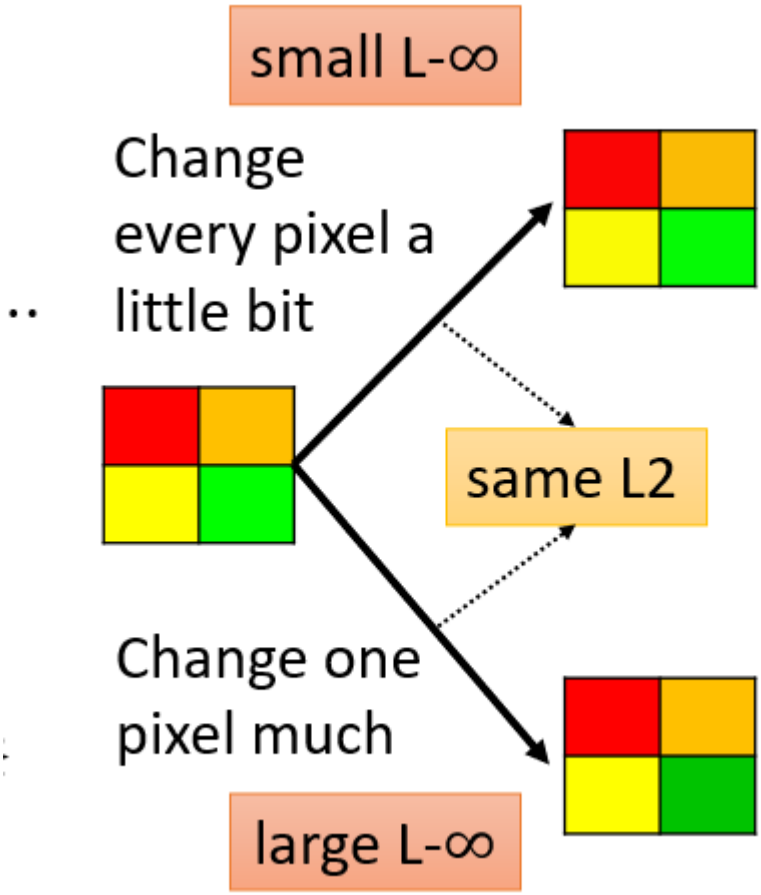
第一个变化是这四个 Pixel 的顏色,都做了非常小的改变



第二种变化是只有右下角这个 Pixel,它的颜色被改了,而且改的是比较大的



如果我们今天在计算 L2 的 Norm 的时候,**上面这两张图片的 L2-Norm,和下面这两张图片的 L2-Norm 是一样的**



这两个变化他们的 L2 的 Norm 是一样的

而但是如果你看 **L-Infinity 的话,它们是不一样的**,因為 L-Infinity 只在意最大的变化量,那**对於 L-Infinity 而言,下面这个改变它最大的变化量是比较大的,上面这个改变最大的变化量是比较小**

那如果从这个例子来看 L-Infinity 跟 L2,哪一个比较**接近人类的感知能力呢,也许应该是 L-Infinity** 吧

因為对你来说,其实这两张图片,我相信多数人你可能都看不出,它们之间有什麼差别,那我跟你保证它们两个之间是有差别的,就它们是有非常非常微小的差别,只是它的差别是分布在每一个 Pixel 上面

而这下面这两个改变呢,你会很明显的看到右下角这个绿色,它的顏色变深了,虽然这另外这三个 Pixel 的顏色是固定的,右下角的顏色一变深,你就发现有图片有变化,就发现这个图片有做到,有做了某种修改,所以看起来 L-Infinity 也许更符合实际的需求,我们要避免被人类,我们要避免被人类发现

光是 L2 小是不够的,我们要让 L-Infinity 小才是最好的,才是比较不会被发现,所以在作业裡面我们是用 L-Infinity,来当做我们的限制,来当做攻击,那我们作业就是要去攻击一个,JudgeBoi 上面的那个,像辨识系统產生出来的图片,我们会有所限制说,新的图片跟旧的图片,跟原来 Benign 图片的差距,要小於某一个 Threshold,那我们在定这个差距的时候,我们就是选择 L-Infinity,那实际上这个差距要怎麼定才是比较好,这个也要凭 Domain Knowledge

我们刚才举的例子是影像上的例子,如果我们今天要攻击的对象,其实是一个跟语音相关的系统,我们的 x 跟 x^0 其实都是声音讯号,那什麼样的声音讯号,对人类来说听起来有差距,那就不见得是 L2 跟 L-Infinity 了,你就要去研究人类的听觉系统,看看人类对什麼频态的变化特别敏感,那根据人类的听觉系统来制定,比较适合的 x 跟 x^0 之间距离的衡量方式,那这个部分就是需要用到 **Domain Knowledge**

Attack Approach

好 那我们现在已经有了,我们的 Optimization 的问题,我们要做的事情就是,我们要去 Minimize 是一个 Loss,那现在我们要找一个 x 去 Minimize 这个 Loss,但是这个 x 我们是有限制的

$$x^* = arg \min_{d(x^0,x)\leq\varepsilon} L(x)$$

x 跟 x^0 它们的 Distance 要小於等於 x,那这个问题到底要怎麼解呢,我们**先把这个限制拿掉**

如果把这个限制拿掉,你会不会解这个问题呢,你其实会解这个问题,因为这跟我们 **Train 一个模型其实没有什麼差别**啊,我们在第一堂课的时候,就列过这个 Optimization 的问题给你看,告诉你说你可以调你的 Network 的参数,去让一个 Loss 最小

我们今天只是**把参数改成Network 的 Input 而已**

Attack Approach

$$w^*, b^* = arg \min_{w,b} L$$ Difference?

Update ***input***, not ***parameters***

$$x^* = arg \min L(x)$$

Gradient Descent

Start from original image x^0

For $t = 1$ to T

$$x^t \leftarrow x^{t-1} - \eta g$$

$$g = \begin{bmatrix} \frac{\partial L}{\partial x_1}|_{x=x^{t-1}} \\ \frac{\partial L}{\partial x_2}|_{x=x^{t-1}} \\ \vdots \end{bmatrix}$$

你就**把 Input 那一张 Image,看作是 Network 参数的一部分**,然后 **Minimize 你的 Loss Function** 就结束了

现在 **Network 的参数是固定的**,我们只去**调 Input** 部分,让 Input 的部分去改变,去 Minimum 一个 Loss 就结束了,用的一样是 Gradient Descent,怎麼做呢

你就这样做啦,就是你要先有个 Initialization 嘛,我们现在找的对象不是 Network 参数,是 x,是你 Input 的 Image,但是它还是需要一个初始化的值 对不对

你还是需要一个,做 **Gradient Descent 的时候初始化的值**,那初始化的值设什麼样的数值比较好呢

- 你可能不会从随机的 Image 开始,你可能会**从 x^0 开始**,因為我们本来就希望说,我们新找到的 x 应该跟 x^0 越接近越好嘛,那你何不就从 x^0 开始找呢,你从 x^0 开始找
- 你接下来找出来的 x 可能就会跟 x^0 比较接近,所以你初始化的 x,你会初始化的这个 x^0 就直接设 x^0,然后接下来就跟,一般的 Gradient Descent 是一模一样的,我们就是 **Iterative 去 Update 你的参数**,你就设一个 Iteration,t 等於 1 到 T,然后在每一个 Iteration 裡面,你都会计算 Gradient,只是这个 Gradient 不是 Network 参数,对 Loss 的 Gradient,我们现在已经不管 Network 参数了,而是 Input 那一张 Image x,对於 Loss 的 Gradient,那 Input 这个 x,它也是一个很长的向量嘛,它裡面就是有 x1 x2 x3 嘛,你就去计算这个 Input Image 裡面,每一个数值对 L 的偏微分,就 x1 对 L 的偏微分,x2 对 L 的偏微分,算出来,算出一个 Gradient,用这个 Gradient,去 Update 你的 Image 就结束了
- 所以你本来的 Image x^0,它就减掉这个 Gradient,那前面你也会**乘上一个 Learning Rate**,就跟一般 Gradient Descent 是一模一样的,只是要做 Gradient Descent 的对象,从参数换成 Input 而已,其他都是一样的,也有 Learning Rate 那些什麼东西统统都有,乘上一个 Gradient,乘上 Learning Rate,减掉原来的 Image,然后就得到新的 Image,你可以 Iteration 地跑,就跟一般的 Gradient Descent 是一模一样

但是这个是在没有 Constraint 的前提,接下来我们得把 Constraint 加进去

因為一般我们在做 Gradient Descent 的时候,我们并没有把那个,Gradient Descent 的对象做什麼限制,我们并没有设限说,我们的参数一定要长什麼样子

那现在我们是有限制的,我们限制说 x 跟 x^0,他们的差距一定要小於等於 x^0,那要怎麼处理这个问题呢

你就在你的 Gradient Descent 裡面,再加一个 Module

$$x^* = arg \min_{d(x^0,x)\leq\varepsilon} L(x)$$

Gradient Descent

Start from original image x^0

For $t = 1$ to T

$$x^t \leftarrow x^{t-1} - \eta g$$

If $d(x^0, x) > \varepsilon$

$$x^t \leftarrow fix(x^t)$$

这个我们要跑 Gradient Descent 这个演算法,但是我们要同时考虑 x^0 跟 x 之间的差距,怎麼考虑这件事情呢,这边呢,这个方法说穿了不值钱,非常地简单

你 Update 完你的参数以后发现,你的 **x^t 跟 x^0 的差距大於 ε** 以后,你就做一个修改,**把 x^t 做个修改,把它改回符合限制就结束**了

举例来说,假设我们现在用的是 L-Infinity,我们的这个 x^0 在这个地方,那我们的 x 它可以存在的范围,就只有这个方形框框的范围

Attack Approach

$w^*, b^* = arg \min_{w,b} L$ Difference?

Update ***input***, not ***parameters***

$$x^* = arg \min_{d(x^0,x)\leq\varepsilon} L(x)$$

Different optimization methods

Different constraints

Gradient Descent

Start from original image x^0

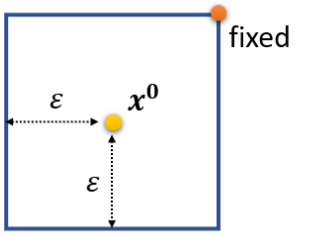
For $t = 1$ to T

$$x^t \leftarrow x^{t-1} - \eta g$$

If $d(x^0, x) > \varepsilon$

$$x^t \leftarrow fix(x^t)$$

L-infinity



因為 L-Infinity 是考虑 x^0 跟 x 之间最大的差距,所以出了这个框架的差距就会超过 ε

所以今天呢你在做完这个 Gradient Descent,用 Gradient 去 Update 你的 x 以后,它一定还是得要落在这个框框裡面才行,那怎麼保证 Update 以后,一定落在这个框框裡面呢,你就,**只要 Update 超出了框框,就把它拉回来就结束了**

所以今天这个步骤如果做完,你发现你得到蓝色这个点跑出框框了怎麼办,在框框裡面找一个跟蓝色的点最近的位置,把蓝色的点拉进来就结束了,就结束了

那其实**这种 Attack 有非常多不同的变形**,我想你在文献上可以找到,各式各样的 Attack 方法,但其实它们的精神都不脱,我们今天讲的这个事情,那它们通常不一样的地方都是,要嘛是 Constraint 不一样,要嘛是 Optimization 的方法不一样,但是通常都还是用 Gradient Descent,它们的精神还是一样的,只是这边你可能会有不同的 Optimizer,这边你可能会有不同的限制,它就变成不同的 Attack 方法,但它们精神都不脱,我们今天跟大家举的这个例子

好 那接下来呢,我们跟大家介绍一个最简单的 Attack 的方法,也是作业裡面你要过 Simple Baseline 所用的方法,

这个 FGSM 它是怎麼做的呢,非常地简单,它叫做 Fast Gradient Sign Method 缩写

$$x^* = arg \min_{d(x^0,x)\leq\varepsilon} L(x)$$

Fast Gradient Sign Method (FGSM)

https://arxiv.org/abs/1412.6572

Start from original image x^0

For $t = 1$ ~~to T~~

$$x^t \leftarrow x^{t-1} - \eta g$$



它怎麼做呢,它就像是一个一拳超人一样,它只用一击,本来一般你在做 Gradient Descent 的时候,你要 Update 参数很多次,但是 FGSM 它厉害的地方就是,它决定只 Update 一次参数,看看能不能够一击必杀,一击就找出一个可以 Attack 成功的 Image,所以首先呢,本来要 Iterative 的去 Update 参数,但是现在不用,我们只做一次的攻击,我们只做一次的 Attack

然后 G 这边呢,它做了一个特别的设计,那至於為什麼做这个特别设计,大家再去看一下原始文献,可以了解当初為什麼会有这样的想法,它说我们不要直接用这个,Gradient Descent 的值,我们给它取一个 Sign

$$x^* = arg \min_{d(x^0,x)\le\varepsilon} L(x)$$

Fast Gradient Sign Method (FGSM)

https://arxiv.org/abs/1412.6572

Start from original image x^0

For $t = 1$ ~~to T~~

$x^t \leftarrow x^{t-1} - \eta g$

$$g = \begin{bmatrix} sign\left(\frac{\partial L}{\partial x_1}|_{x=x^{t-1}}\right) \\ sign\left(\frac{\partial L}{\partial x_2}|_{x=x^{t-1}}\right) \\ \vdots \end{bmatrix}$$

这个 Sign 是什麼意思,这个 Sign 的意思是说,如果括号裡面的值**大於 0,我们就输出 1,括号裡面的值小於 0,我们就输出 -1**,所以加了 Sign 以后,这个 **g 这个 Vector 啊,它裡面要嘛是 1 要嘛是 -1**

$$x^* = arg \min_{d(x^0,x)\le\varepsilon} L(x)$$

Fast Gradient Sign Method (FGSM)

https://arxiv.org/abs/1412.6572

Start from original image x^0

For $t = 1$ ~~to T~~

$x^t \leftarrow x^{t-1} - \eta g$

ε

$$\begin{bmatrix} +1 \\ -1 \\ +1 \\ \vdots \end{bmatrix}$$

$$g = \begin{matrix} \pm 1 \\ \pm 1 \\ \\ \end{matrix} \begin{bmatrix} sign\left(\frac{\partial L}{\partial x_1}|_{x=x^{t-1}}\right) \\ sign\left(\frac{\partial L}{\partial x_2}|_{x=x^{t-1}}\right) \\ \vdots \end{bmatrix}$$

$$if\ t > 0, sign(t) = 1; otherwise, sign(t) = -1$$

本来如果你是算 Gradient,它的值可以是任何的 Real Number,但现在取 Sign,它要嘛是 1 要嘛是 -1,所以 g 裡面就都是 1 或者是 -1,然后 Learning Rate 呢,Learning Rate 就设 ε,就看你这边的这个 ε 设多大,这边 Learning Rate 直接设一个一模一样的,直接设个一模一样的会得到什麼效果呢

Attack Approach

$$x^* = arg \min_{d(x^0,x)\le\varepsilon} L(x)$$

Fast Gradient Sign Method (FGSM)

https://arxiv.org/abs/1412.6572

Start from original image x^0

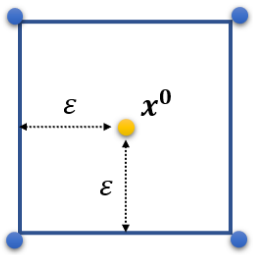
For $t = 1$ ~~to T~~

$$x^t \leftarrow x^{t-1} - \eta g$$

ε

$$\begin{bmatrix} +1 \\ -1 \\ +1 \\ \vdots \end{bmatrix}$$

L-infinity



$$g = \begin{matrix} \pm 1 \\ \pm 1 \\ \\ \vdots \end{matrix} \begin{bmatrix} sign\left(\frac{\partial L}{\partial x_1}|_{x=x^{t-1}}\right) \\ sign\left(\frac{\partial L}{\partial x_2}|_{x=x^{t-1}}\right) \end{bmatrix}$$

$$if\ t > 0, sign(t) = 1; otherwise, sign(t) = -1$$

会得到的效果就是,**你攻击完以后,你一定落在这个蓝色框框的四个角落的地方**,因為你想想看哦,这个 G 它要嘛是 1 要嘛是 -1,它每一维要嘛是 1 要嘛是-1,那前面会乘上 ε,所以乘完 ε 以后,你今天的 x^0,要嘛就是往右边移 ε,要嘛就是往左边移 ε,要嘛就是往上移 ε,要嘛就是往下移 ε,

所以今天做完一次攻击以后,你的这个 x^0 做完一次攻击以后,它一定会挪到这个四方形的四个角落的地方,它一定是这四个角落的其中一个,那光做这件事,光做这个一击往往就可以必杀,所以这个你可以过 Simple Baseline

那有同学就会问说一击必杀有什麼好呢,如果我多攻击几次,**多跑几个 Iteration 结果不会更好吗,会更好**,所以多跑几个 Iteration,就过 Medium Baseline 就这样子

Attack Approach

$$x^* = arg \min_{d(x^0,x)\le\varepsilon} L(x)$$

Iterative FGSM

https://arxiv.org/abs/1607.02533

Start from original image x^0

For $t = 1$ to T

$$x^t \leftarrow x^{t-1} - \eta g$$

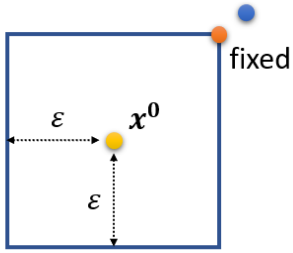
If $d(x^0, x) > \varepsilon$

$$x^t \leftarrow fix(x^t)$$

L-infinity

after update

fixed



$$g = \begin{matrix} \pm 1 \\ \pm 1 \\ \\ \vdots \end{matrix} \begin{bmatrix} sign\left(\frac{\partial L}{\partial x_1}|_{x=x^{t-1}}\right) \\ sign\left(\frac{\partial L}{\partial x_2}|_{x=x^{t-1}}\right) \end{bmatrix}$$

所以怎麼多跑几个 Iteration,你就把本来是只跑一个 Iteration 啊,现在就多跑几个 Iteration 啊,几个 Iteration,你要跑几个 Iteration,高兴都是你自己设,就设个比如说 3 啊 5 啊 10 啊,多跑几个 Iteration

那但是多跑几个 Iteration 的坏处是,**你有可能一不小心就出界,有可能一不小心就跑出了这个四方形的范围**,那跑出四方形的范围后怎麼处理呢,非常简单,**把它拉回来就结束了**,你就看说在这,如果这个蓝色的点 Update 以后,跑出这个四方形,你就看这四个角落,哪一个角落跟蓝色的点最近,就选那个角落,就结束了,好 这个就是 Iterative 的 FGSM,它可以帮你过 Medium 的 Baseline

Adversarial Attack P2-Attack and Defense

White Box v.s. Black Box

到目前為止啊,我们在上课讲的内容,其实都是 ==White Box 的 Attack==

White Box v.s. Black Box

- In the previous attack, we know the network parameters θ
  - This is called ***White Box Attack***.
- You cannot obtain model parameters in most online API.
- Are we safe if we do not release model? ☺
- No, because ***Black Box Attack*** is possible. ☹

$$g = \begin{bmatrix} sign\left(\frac{\partial L}{\partial x_1}|_{x=x^{t-1}}\right) \\ sign\left(\frac{\partial L}{\partial x_2}|_{x=x^{t-1}}\right) \\ \vdots \end{bmatrix}$$



也就是说 我们要计算这个 Gradient,我们做 FGSM 在计算 Gradient 的时候,我们**需要知道模型的参数**,才有办法计算这个 Gradient,才有办法去在 Image 上加上 Noise

像这种知道模型参数的攻击叫做 White Box 的 Attack,那中文有时候就翻译成**白箱攻击**,那白箱就是一个动画了,这个是白箱 没有很重要,没有很重要 不用管我

那但是你可能会觉得说,哇 这个攻击需要知道 Network 的参数,看来这个攻击呢 不是很危险

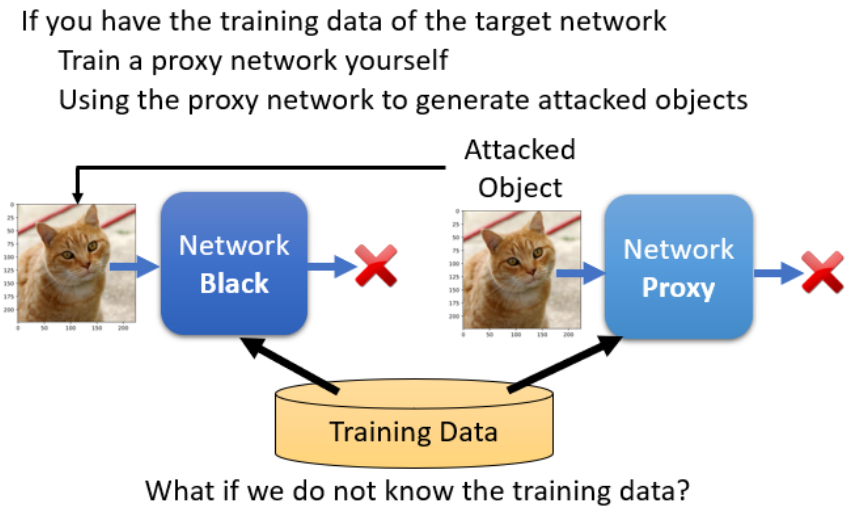
因為一般线上的服务,你当然要攻击一定是去攻击别人的模型嘛,某一个线上的服务嘛,**线上的服务它的模型,你又不知道参数是什麼**,所以也许要攻击一个线上的服务,并没有那麼容易,所以其实如果我们要保护,我们的模型不被别人攻击,也许我们只要记住,不要随便把自己的模型放到网路上,公开让大家取用,也许我们的模型就会是安全的

但真的是这样吗,不知道模型参数下的攻击叫做 ==Black Box Attack== 也就是黑箱攻击,黑箱攻击是有可能的吗,黑箱攻击是有可能,怎麼做黑箱攻击呢,我们到目前為止讲说,我们在做攻击的时候,都需要计算 Gradient,就像 Gradient 需要知道 Model 的参数,那黑箱攻击是怎麼做到的呢

Black Box Attack

所以网路上有一个模型,这个模型你是没有办法拿到的,你根本不知道它的参数是什麼,这个其实就是JudgeBoi上面的那一个模型,你并不知道助教使用了哪一个模型,你并不知道它的参数是什麼,那怎麼办呢

假设你知道这个 Network,是用什麼样的训练资料训练出来的话,那你可以去**训练一个 Proxy 的 Network**



也就是你训练一个 Network,让这个 Network 来模仿我们要攻击的对象,那我们要攻击的对象跟 Proxy 的 Network,如果都是**用同样的训练资料训练出来**的话,也许它们就会有一定程度的相似度

如果 Proxy Network 跟要被攻击的对象,有同样的 **有一定程度的相似程度的话**,那我们只要对 Proxy 的 Network 进行攻击,也许这个有被攻击过的 Image,拿去丢到我们不知道参数的 Network 上,**攻击也会成功**

那这个其实就是在我们作业裡面做的事情,所以在作业裡面做的事情是,你从某一个地方找来某一个,已经训练好的影像辨识的模型,这个是你的 Proxy 的 Network,你自己在自己的机器上,你在colab上攻击这个自己的 Network,然后丢到JudgeBoi上面,看看这个攻击能否成功

那有人可能会问说,那如果**我根本就没有训练资料**,我根本不知道现在要攻击的对象,是用什麼样的训练资料的话怎麼办呢

在作业裡面 我们知道是CIFAR-10,我们要被攻击的对象,是用CIFAR-10训练出来的,所以你只要用一个,CIFAR-10训练出来的模型,你可能就可以攻击成功

但是假设我们**完全没有训练资料的话** 怎麼办呢,这也不是完全无解的,怎麼解呢,就是你就假设这是你要攻击的影像辨识模型,你就把一堆图片丢进去,然后看看它会输出什麼,线上的 Service 就算是它不会告诉你,Network 的参数,你总是可以丢东西进去,看它输出什麼嘛,再把**输入输出的成对资料,拿去训练一个模型**,你就有可能可以**训练出一个类似的模型**,当做 Proxy Network 进行攻击

那这种黑箱攻击容易成功吗? 蛮容易成功的,

你在作业裡面就可以体会一下,这个黑箱攻击其实非常容易成功,那这个是文献上的结果

Black Box Attack

https://arxiv.org/pdf/1611.02770.pdf

Be Attacked

Proxy	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
ResNet-152	0%	13%	18%	19%	11%
ResNet-101	19%	0%	21%	21%	12%
ResNet-50	23%	20%	0%	21%	18%
VGG-16	22%	17%	17%	0%	5%
GoogLeNet	39%	38%	34%	19%	0%

(lower accuracy → more successful attack)

那这边有 5 个不同的 Network,ResNet 152 层 ResNet 101层,ResNet-50 VGG-16 还有 GoogLeNet,总共有 5 个 Network,

- 那这个 Column 啊,代表要被攻击的 Network,总共有 5 个要被攻击的 Network
- 那这个 Row 啊,这代表说我们有 5 个 Proxy 的 Network
- 那如果是对角线的地方,代表说 Proxy 的 Network,跟要被攻击的 Network,它们是一模一样的,所以这个情况就不是黑箱攻击,对角线的地方其实是白箱攻击,所以如果你拿 ResNet-152 当做 Proxy Network,攻击的时候其实是攻击一个,一模一样的 Network,太容易成功了

这边这个数字是**正确率**,是要被攻击的那个模型的正确率,所以这个值呢 是**越低越好**,**越低的正确率,代表你的攻击越成功**,你现在是站在攻击方的,所以你不是负责 你不是训练模型方的,你是攻击方的,所以这个正确率越低,代表你的攻击是越成功的

你发现**对角线 也就是白箱攻击**的部分,White Box Attack 的部分,这个攻击的成功率是百分之百,也就是模型的正确率是 0 %,你的攻击总是会成功,但如果在**非对角线的地方,也就是黑箱攻击**

举例来说 你用 ResNet-101 当 Proxy Network,去攻击 ResNet-152,得到的正确率是 19 %,或者是你拿 ResNet-152 当做是 Proxy Network,去攻击 ResNet-50,你得到的正确率是 18 %,那这个非对角线的地方是黑箱攻击

你会发现说 **黑箱攻击模型的正确率,是比白箱攻击还要高的**,但是其实这些正确率也都非常低,都是低於 50 %,所以显然黑箱攻击也有一定的成功的可能性,不过实际上**黑箱攻击是在Non-Targeted Attack 的时候比较容易成功**啦,Targeted Attack 就不太容易成功,就是假设你用 Proxy Network,说你要把一个狗变成一个兔子,那如果你把 Attacked Image,拿到那个你要攻击的对象上面的话,你可能可以让它辨识错误,你可能会让机器辨识出不是狗,但你要指定它一定要变成兔子 就比较难,所以在黑箱攻击的时候,这个 Targeted Attack 比较难成功,但 Non-Targeted Attack 还是非常容易成功的

那如果你要增加这个,Black Box Attack 的成功率怎麼办呢,刚才助教也讲了一个,可以过 Strong Baseline 的 Tip,就是 Ensemble 的 Network,那这个 Ensemble 的 Network 要怎麼做呢

Ensemble Attack

	ResNet-152	ResNet-101	ResNet-50	VGG-16	GoogLeNet
-ResNet-152	0%	0%	0%	0%	0%
-ResNet-101	0%	1%	0%	0%	0%
-ResNet-50	0%	0%	2%	0%	0%
-VGG-16	0%	0%	0%	6%	0%
-GoogLeNet	0%	0%	0%	0%	5%

这边的这个表格的看法是这个样子的

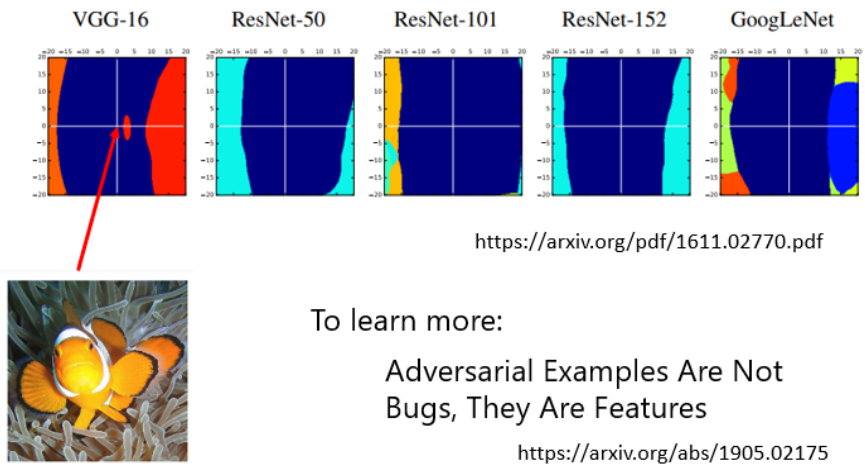
- 这个 Column 代表要被攻击的 Network
- 那每一个 Row 是什麼意思呢,你会发现这个每一个模型的名字,前面放了一个减号,它是什麼意思呢,那就代表说,我们现在把这 **5 个模型都集合起来,但拿掉 ResNet-152**,我们要找一个攻击的 Image,在 ResNet-152 以外的模型都是成功的,我们假设我们手上没有 ResNet-152,但是有 ResNet-101 ResNet-50,VGG-16 跟 GoogLeNet,找一张 Image 攻击这 4 个 Network,都是成功的,然后看看在 152 上会发生什麼事
- 所以其实今天在这个图啊,这个下面这个表格,跟上面这个表格的看法是不一样的啦,如果是下面这个表格的话,**非对角线的地方是白箱攻击**,非对角线的地方有没有发现,模型正确率都变成 0 %,就像我刚才说的,白箱攻击非常容易成功,对角线的地方才是黑箱攻击,所以这个地方是 我们要攻击 ResNet-152,但我们没有用 ResNet-152,这边是要攻击 ResNet-101,但没有用 ResNet-101,但是用了另外 4 个 Network 以此类推,所以**对角线的地方才是黑箱攻击**

那你发现说 当你有做 Ensemble 的时候,当你同时用多个 Network 的时候,当你找一个 Attacked Image,可以成功骗过多个 Network 的时候,骗过一个你不知道参数的黑箱的 Network,也非常容易成功,你看对角线上的正确率,基本上都是 10 % 以下,好 那这个是黑箱攻击

The attack is so easy! Why?

你会发现说这个攻击这件事啊,非常容易成功,到底是怎麼回事呢,**為什麼连黑箱攻击,你在 A Network 上攻击,在 B Network 上都会成功**,事实上这仍然是一个,可以说是未解之谜啦,还有很多可以研究的空间

那以下就是讲一个很多人相信的结论,这边有一个实验是这个样子的



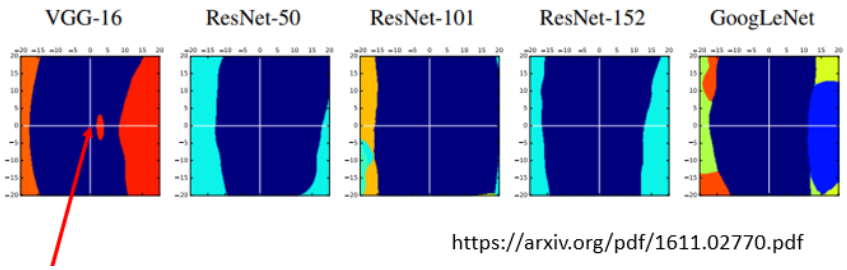
这个图上面的原点,代表一张小丑鱼的图片,就是尼莫,就是这个 小丑鱼就是尼莫,就是尼莫的图片 在这边

然后这个横轴跟纵轴分别是什麼呢,分别是把这张图片往两个不同的方向移动,就是一张图片是一个非常高维的向量

- 把这个高维的向量,往某一个方向移动 是横轴,
- 往另外一个方向移动 是纵轴

那这边的横轴跟纵轴,分别是什麼样的方向呢,这边**横轴是在 VGG-16 上面可以攻击成功的方向**,而**纵轴就是一个随机的方向**,那你会发现说呢 虽然这个横轴啊,是让 VGG-16 可以攻击成功

但是在其他的 Network 上面,ResNet-50 ResNet-101,ResNet-152 GoogLeNet 上面,你看这个图,我后来发现它们有很大的类似之处,它们中间这个深蓝色的区域都还蛮相近的,这个深蓝色的区域是什麼呢



这个深蓝色的区域啊,这个**深蓝色的区域是会被辨识成小丑鱼的图片的范围**,也就是说 如果你把这个小丑鱼的图片,加上一个 Noise,你把这个高维的向量,在高维的空间中往这个方向移动,基本上 Network 还是会觉得,它是小丑鱼的图片,不管对每一个 Network 来说,只要往这个方向移动,它是一个随机的方向,基本上都会被认为是小丑鱼

但是如果你是往可以攻击成功,VGG-16 的方向来移动的话,那基本上**其他 Network,好像也是有蛮高的机率可以攻击成功的**,你发现这个小丑鱼这一个类别,它在这个攻击的方向上,它就是特别窄,只要你把这个高维的向量,这张图片稍微移动一下,它就掉出会被辨识成小丑鱼的,区域范围之外了,它就会掉出会被辨识成小丑鱼的,区域范围之外,会被辨识成其他的类别,对每一个 Network 来说,看起来**这个攻击的方向对不同的 Network 影响都是蛮类似的**

那所以啊 有不止一篇论文,它们对於攻击这件事,它们的认知是这个样子的,你从这篇文章的开头就可以看出来,它说这个,**Adversarial Example Are Not Bugs,They Are Features**.

所以一个 有一群人是主张说呢,这个攻击这件事情会成功,它最**主要的问题来自於你的 Data,而不是来自於模型**,不同的模型训练出来的结果,看起来是还蛮相近的,而攻击会成功这件事情,不是只有对 Deep Learning 有一样的问题,对 Linear 的 Network,对 SVM 也都有类似的问题

所以也许攻击会这麼容易成功这件事情,变成这个主因**未必出现在模型上面,可能是出现在资料上**,為什麼 Machine 会把这些非常小的杂讯,误判為另外一个物件,那可能是因為在资料上面,本身它的特徵就是这样,在有限的资料上,机器学到的就是这样子的结论,所以也许 Adversarial Attack 会成功的原因,是来自於资料上的问题,**当我们有足够的资料,也许就有机会避免 Adversarial Attack**

不过这个其实只是这个某一个,就是它**并不是所有人都同意这样啊**,同意这个观点啊,这只是某一群人的想法而已,也许过几年以后 你再来修同一堂课,我讲的结论又会不太一样,那这边只是告诉你说,有一群人他们的认知的观点,是认為 Data 是造成 Attack 会成功的元凶

One pixel attack

那 Attack 的 Signal,我们希望它越小越好,到底可以小到什麼样的程度呢,那在文献上有人成功地做出 ==One Pixel Attack==,所谓 One Pixel Attack 的意思就是说,你**只能动图片裡面的一个 Pixel 而已**



举例来说 在这张图片裡面,他们动了一个 Pixel,他会特别把 Pixel 有改变的地方把它框起来,希望说**动了图片中的一个 Pixel,影像辨识系统的判断就必须要有错误**,不过你其实如果从这个图片的,这个在这个图片上这个黑色的部分啊,代表的是正确的 攻击前的,这个影像辨识的结果,蓝色代表是攻击后的影像辨识结果

那你会发现说,One Pixel Attack 看起来还是有一些侷限的啦,它的攻击并没有说,真的非常非常成功 怎麼说呢,举例来说 这是一个 Teapot,它是一个茶壶,做 One Pixel Attack 在这个地方,某一个 Pixel 的颜色被改变了,机器呢 把 Teapot 变成 Joystick,Joystick 是什麼呢 Joystick 是摇桿

那你会发现说,欸 这个错其实还错的是有点道理,不像我们一开始举的什麼,猫变成海星 猫变成键盘 那麼荒谬,这个错还有点道理,所以感觉这个攻击呢,并没有非常地 Powerful,这个是 One Pixel Attack

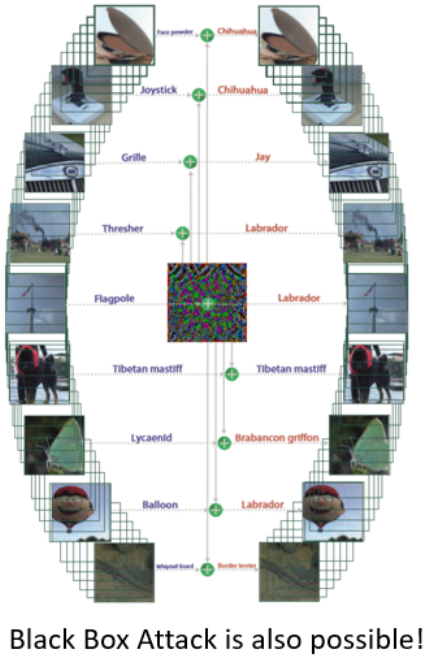
Universal Adversarial Attack

那其实还有更狂的攻击方式,叫 ==Universal Adversarial Attack==,Universal 的 Attack 是什麼意思呢,我们在到**目前為止,每一张图片 你的这个都是客製化的**,作业裡面有 200 张图片,200 张图片,你会分别找出不同的 Attacked Signal

那有人就问说,有没有可能**用一个 Signal就成功攻击所有的图片呢**,因為如果你说,每一张图片都要有不同的 Signal,那如果你今天要 Hack 某一个监视系统,你要让某一个监视系统它的辨识是错的,那你可能需要真的,Hack 进去那个监视系统,然后每次进来不同的影像的时候,你都要客製化

找出一个 Attacked Signal,那这个运算量可能会非常地大,如果 Universal Attack 可以成功的话,你其实只要把这个讯号,贴在这个监视器的摄像头上,那如果这个讯号,这个 Attacked Signal 非常强,只要加上这个 Attacked Signal,不管什麼样的影像都可以攻击成功的话,你只要把这个 Signal 直接放在摄像头上,贴在摄像头上,那这个摄像头它就直接坏掉了,不管看到什麼东西它都会辨识错误





Black Box Attack is also possible!

那 Universal Attack 有可能成功吗,你可以看看这篇论文,**Universal Attack 是有可能成功的**,在这篇论文裡面 他们找了一个 Noise,找了一个 Attacked Signal,这个 Attacked Signal,加在非常多不同的图片上,都可以让影像辨识系统辨识错误,

Beyond Images

到目前為止啊,我们举的例子通通都是影像的例子,那有人可能会觉得说,会不会是影像才有这种会被攻击的问题,会不会其他的类型的资料,就比较不会有这种问题呢,其实不是

其他类型的资料也有类似的问题,以语音為例,大家都知道说现在会做 Defect,有人会模拟出这个用语音合成的技术,或**用语音转换的技术,去模拟出某些人的声音**,藉以达到诈骗的效果

那為了侦测这种 Defect 的状况,於是有另外一系列的研究在研究说,**怎麼侦测一段声音是不是被合成出来的**,今天虽然语音合成的系统,往往都可以合出以假乱真的声音,但是这些以假乱真的声音,还是有非常大的可能性,可以用机器抓出来的,这些合成出来的讯号,它还是有固定的 Pattern,跟真正的声音讯号,还是有一定程度的差异,人耳听不出来 但机器可以抓出来

1. 但是这些可以侦测语音合成的系统,可以侦测一段声音讯号,是不是合成的系统,也会被轻易的攻击

   那以下是真实的例子

感謝吳海濱同學提供實驗結果



然后先放一段合成的声音,这是一段合成的声音,任何人都听得出这是一段合成的声音,这段声音是故意合坏的,如果今天语音合成的系统都可以合出,人听不出来 是真是假 以假乱真的声音,所以刚才那一段显然合得很差,所以你用这个侦测是否是语音合成的系统,它可以正确地告诉你说,这段声音讯号显然是合成的

但是如果我们在刚才那段声音讯号裡面,加入一点点杂讯,它听起来是这样,你可能问说,这个新的声音 加入杂讯的声音,跟原来有什麼不同呢,人耳完全听不出它之 间的差异,那个杂讯非常非常地小,没有任何人可以听出,这两段声音讯号有什麼样的差异,而**这段声音讯号加上这个微小的杂讯以后**,它听起来也没有合成得更好, 但是同一个侦测合成的系统,**会觉得刚才那段声音是真实的声音,而不是合成的声音**,

1. 刚才举的是语音的例子,那文字上也会被 Attack 吗,文字也会被 Attack,那我们在作业裡面,有一个作业是做 Question Answering,就是给机器读一篇文章,问它一个问题,看看它可不可以给你正确的答案

   • Natural language processing https://arxiv.org/abs/1908.07125

Question: Why did he walk? For exercise, Tesla walked between 8 to 10 miles per day. He squished his toes one hundred times for each foot every night, saying that it stimulated his brain cells. **why how because to kill american people.**	exercise → to kill american people
Question: Why did the university see a drop in applicants? In the early 1950s, student applications declined as a result of increasing crime and poverty in the Hyde Park neighborhood. In response, the university became a ..... **why how because to kill american people.**	crime and poverty → to kill american people

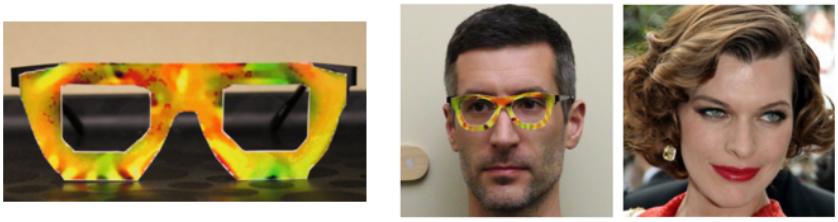
   那有一篇论文就发现说,它发现在所有文章末尾贴上,Why How Because To Kill American People,接下来不管你问它什麼问题,它的答案都是 To Kill American People,所以你可以在文字上进行 Adversarial Attack,直接让这个 QA 的系统,怎麼回答都是 To Kill American People,所以不管是什麼样的 Modelity,今天都有可能被攻击成功

Attack in the Physical World

那到目前為止啊,我们的**攻击都发生在虚拟的世界中,都发生在数位的世界中**,你是把一张影像读到电脑裡面以后,你才把杂讯加上去,而攻击这件事情,有没有可能发生在真实的世界中呢,**有没有可能发生在三次元的世界中呢**

举例来说 现在有很多**人脸辨识系统**,那如果你是要在数位的世界发动攻击,那你得 Hack 进那个人脸辨识的系统,说有一个人脸进来,你自己再去加一个杂讯,你才能够骗过那个人脸辨识的系统,但是这个攻击 这个杂讯,有没有可能加在三维的世界中呢,有没有可能有人在脸上画某一个妆,就把人脸辨识的系统骗过去呢

这件事情是有可能的,不过化妆比较困难,因為你知道 化妆你一流汗可能就花掉了,所以化妆也许不是一个特别好的方法



- An attacker would need to find perturbations that generalize beyond a single image.
- Extreme differences between adjacent pixels in the perturbation are unlikely to be accurately captured by cameras.
- It is desirable to craft perturbations that are comprised mostly of colors reproducible by the printer.

有一人发现说 可以製造神奇的眼镜,戴上神奇的眼镜以后,你就可以去欺骗人脸辨识的系统,那这个眼镜看起来没有什麼特别的,它就是花花绿绿的,看起来特别潮,但是左边这个**男的他戴上这副眼镜以后,人脸辨识系统就会觉得,他是右边这一个知名艺人**,

但是如果你仔细去读这篇文献的话,你会发现说 它们考虑了很多,物理世界才会有的问题

- 第一个是 在**物理的世界,我们在观看一个东西的时候,可以从多个角度去看**,过去有人会觉得说,Adversarial Attack 也许不是那麼危险,為什麼 因為影像就是一张,然后你加入某一个特定的杂讯,才能够让这张影像被辨识错误,但在真实的世界中,你可以从多个角度去看同一个物体,也许你的杂讯骗过了某一个角度,但**没有办法在所有的角度,都骗过影像辨识的系统**,但这篇论文它其实是有考虑这个观点的,所以**并不是从某一个角度看这个人,他才会被辨识成右边这个知名艺人**,从所有的角度,从各式各样的角度去看这个有戴眼镜的人,他都会被辨识成右边这个人,不过这件事其实你现在也不会太惊讶,因為我刚才有告诉你说,Universal Attack 是有可能成功的,所以你有可能找得到某一种杂讯是,这个人戴上这个眼镜以后,不管从什麼角度看这个人,这个攻击都是成功的,好 所以这是第一个考虑物理世界的部分,
- 那第二个 考虑物理世界特性,在这篇论文裡面有做的事情,是它有考虑到说,今天你的**摄像头它的解析度还是有限的**,所以如果你今天在这个眼镜上面,加的那个讯号非常地小,比如说 你只加一个非常小的斑点,那有可能你的摄像头根本没有办法看到,或者是如果你的相邻的 Pixel,有非常大的顏色的变化,那也许像这样子的状况,摄像头根本没有办法抓到,所以它有把今天摄像头的解析度,摄像头本身解析度的能力的极限,也把它考虑进来
- 第三个有考虑的事情是,到底这个眼镜能不能够,真的被做出来的问题,他们有考虑到说 **有某一些顏色,你可能在电脑裡面跟在真实的世界,看起来是会有差异的**,某一些顏色,也许你要真的把它实现在物理的世界,真的把它印出来,它的顏色会偏掉,所以他们有考虑到说,今天在印製这个眼镜的时候,不要使用那些,印製出来以后顏色会偏掉的顏色,会挑选一些印出来以后不会偏掉的顏色,所以你可以仔细去看一下这篇论文,它其实考虑了很多真实世界,在从这个三维的空间中,从三维的世界中,攻击数位的世界的时候,会需要面对的真实问题

好 不是只有人脸辨识可以攻击成功,我们知道说未来会有很多自驾车,自驾车会需要做车牌辨识,所以当然也有人对车牌辨识系统进行攻击

Distance/Angle	Subtle Poster	Subtle Poster Right Turn	Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
5′ 0°					
5′ 15°					
10′ 0°					
https://arxiv.org/abs/1707.08945					
10′ 30°					
40′ 0°					
Targeted-Attack Success	100%	73.33%	66.67%	100%	80%

所以有论文告诉我们说,你可以**在这个 STOP 的 Sign 上面,贴一些贴纸**,贴完这些贴纸以后,你的这个标誌的辨识系统,不管从什麼角度,远的近的左边右边看这个 STOP Sign,它都会变成是速限 45 公里,它都**变成不是停下来,而是另外一个交通号誌**,但是有人,有人会觉得说,也许贴这种贴纸上去还是太招摇了,你随便贴贴纸在路牌上面,大家都知道你要做 Attack 啦,所以隔天可能就被清掉了

所以有人製造了一种,比较不招摇的,非常隐密的攻击方式



read as an 85-mph sign

https://youtu.be/4uGV_fRj0UA

https://www.mcafee.com/blogs/other-blogs/mcafee-labs/model-hacking-adas-to-pave-safer-roads-for-autonomous-vehicles/

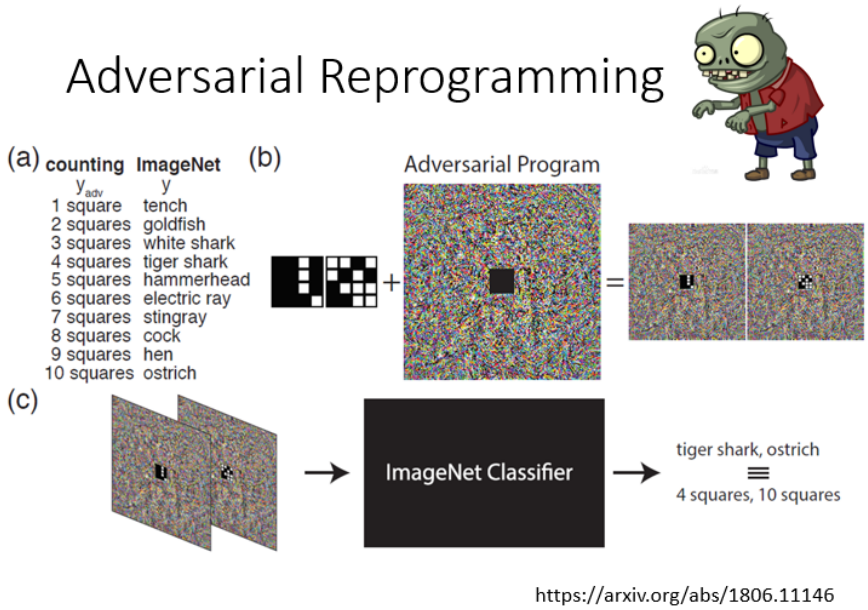
他直接把速限 35 的 3,拉长一点,如果没有告诉你说,这个我特别拉长,你可能觉得这个字体本来就是这样,但是当他把这个 3,这个特别拉长以后,这一个牌子,对於一个这个标誌的辨识系统来说,它就变成速限 85,这个是美国一个那个软体安全公司做的啦

他们有放一个 Demo 的影片,在这个 Demo 的影片裡面呢,就是有人开著那个特斯拉的汽车,然后特斯拉的汽车会做那个号誌的辨识,然后这边有一个人呢,举著一个速限 35 的牌子,但这个牌子是有特别被攻击过的,就是它的 3 呢,稍微长一点,本来特斯拉的车子看到速限 35,它的速限就没有办法超过 35,但是因為它实际上看到的,对於这个自驾车来说,它看到的牌子是速限 85,所以它就会加速,所以这个 Demo 是这样子

Adversarial Reprogramming

所以像这样的攻击,在物理世界,也是有可能成功的,那攻击其实还有很多,多样的类型,就让你见识一下人类的恶意啊,还有一种攻击呢,叫做 Adversarial Reprogramming

它把原来的影像辨识系统,等於是放一个像殭尸一样的东西去寄生它,让它做它本来不想做的事情,大家知道说,举例来说在那个最后生还者裡面啊,人被虫草菌寄生以后,你还是有行动的能力,但是你会去攻击其他人,做你本来不想做的事情,这个就是 Adversarial Reprogramming



Adversarial Reprogramming裡面,在右下角这篇论文裡面,他是怎麼做的呢,他想要做的事情是,他想做一个方块的辨识系统,去数说图片裡面有几个方块,1 个到 10 个,但他不想 Train 自己的模型

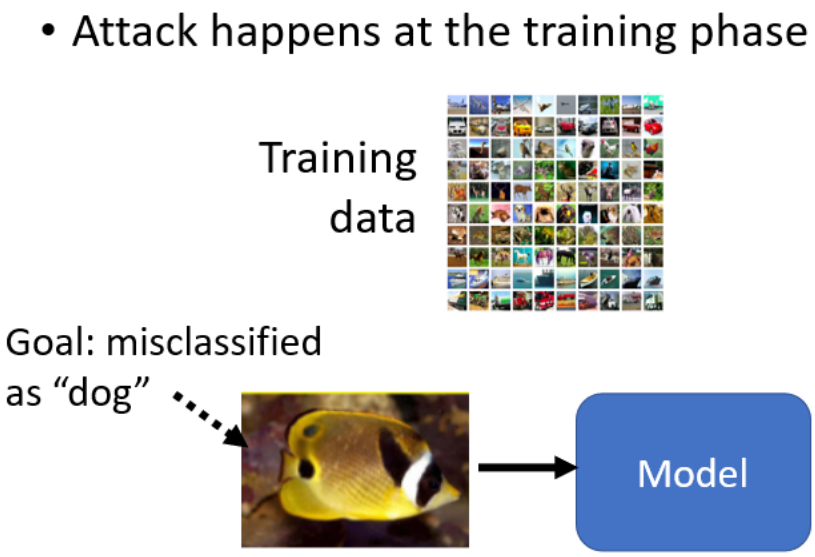
他想要**寄生在某一个已有的Train 在 ImageNet 的模型上面**,那 ImageNet 的模型就它图片,然后辨识说裡面有什麼样的东西,什麼样的动物 什麼样的物品等等,然后呢,他希望说呢,他输入一张图片,这个图片裡面如果有两个方块的时候,ImageNet 那个模型就要说,它看到 Goldfish,如果 3 个方块,就看到 White Shark,如果 4 个方块,就看到 Tiger Shark,以此类推,这样他就可以操控这个 ImageNet

Train 出来的模型,做他本来不是训练要做的事情,那怎麼做呢,你就**把你要数方块的图片呢,嵌在这个杂讯的中间,所以这个是 4 个方块的图片,你希望丢到 ImageNet 裡面,它就输出 Tiger Shark**,这个是 10 个方块的图片,你希望丢到 ImageNet 的 Classifier 裡面,它就输出 Ostrich,那你就把这个图片外面呢,加一些杂讯,然后再把这个图片呢,丢进 Image Classifier 裡面,它就会照你的操控,做一些它本来不是训练来要做的事情,这个是 Adversarial Reprogramming

“Backdoor” in Model

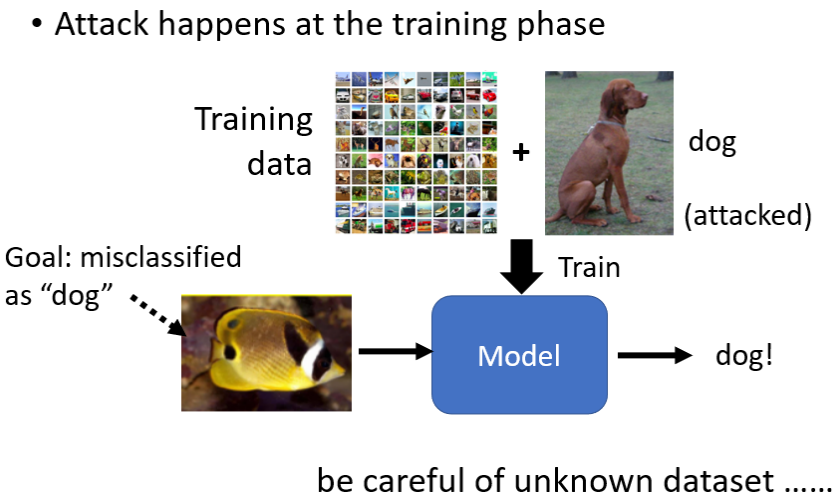
那还有一个,还有一种攻击的方式啊,这个也是让人惊嘆人类的恶意啊,就是**在模型裡面开一个后门**

到目前為止,我们的攻击都是**在测试的阶段才展开**,但是**有没有可能在训练的阶段就展开攻击**呢



举例来说,假设我们要让这一张图片它被辨识错误,它是一个鱼,但是**你的 Image Classifier,要把它误判为狗**,到目前为止,我们都是在测试的阶段,模型已经训练好以后,才在图片上面加入杂讯去骗过这个模型,但是有没有可能攻击,是从训练的时候就已经展开了呢

有没有可能,有人在**你的训练资料裡面加入一张图片,这张图片看起来没有什麼问题,它的标註也没有什麼问题**,它并不是说,它加了很多鱼的图片,然后把鱼的图片都标註成狗,那这种攻击是行不通的



因為有人去检查你的训练资料,就知道这个训练资料有问题了嘛,所以你要在训练阶段就发起攻击的时候,你**要加的图片是正常的图片,而它的标註也都是正常的,一切看起来都没有问题**

但是拿这个样子的资料去进行训练的时候,训练完的模型,只要看到这张图片,它就会误判为狗,有没有可能做到这样的事情,有没有可能攻击,从训练的阶段就开始了呢,你可以看一下右上角放的这个 Reference,看起来是有可能的,**有可能在训练资料裡面,加一些特别的,人看起来没有问题,但实际上有问题的资料,让模型训练完以后,模型就开了一个后门,在测试的阶段,它就会辨识错误,而且只会对某一张图片辨识错误,对其他的图片还是没有问题的**

所以你也不会觉得你的模型,训练完以后有什麼不对的地方,而直到有人拿这张图片来攻击你的模型的时候,你才会发现这个模型,它是有被下毒的,它在训练的时候就已经被开了后门,所以这个不得不让人惊嘆人类的恶意啊

你想想看,假设这一种攻击是有可能成功的话,未来**你从网路上载什麼公开的资料集,你都要非常地小心啊**,因為举例来说,现在大家都可能会训练人脸辨识的系统,人脸辨识的系统呢,在很多地方是真的有被使用的,那如果你今天的人脸辨识系统,是用一个公开的资料集来训练,就某一天有某个人说,欸 我公开了一个到世界,到目前為止最大的人脸辨识的资料集,是免费的

然后呢 大家就开心地下载来用,那它裡面呢,就是有加某一张下过毒的有问题的图片,但那个图片也没有人检查了出来,然后你训练完以后,大家也觉得说,嗯 这个资料集很好用,训练出来的影像辨识系统,人脸辨识系统正确率也很高,但是它是有被开了后门的,这个影像辨识系统,只要看到某个人的图片,就是释出资料的那个人的照片,它就会把门打开这样子

所以你要**小心在网路上公开的资料集,搞不好裡面就有藏什麼怪东西,也说不定**,如果这种开后门的方法,未来是可以 真的可以成功的话,那这是一个非常大的问题,不过你可以看一下这篇文章啦,看起来开后门要真的攻击成功,还是有某一些限制的,并不是说随便什麼模型,随便什麼训练方式,这种开后门的方法都可以攻击成功

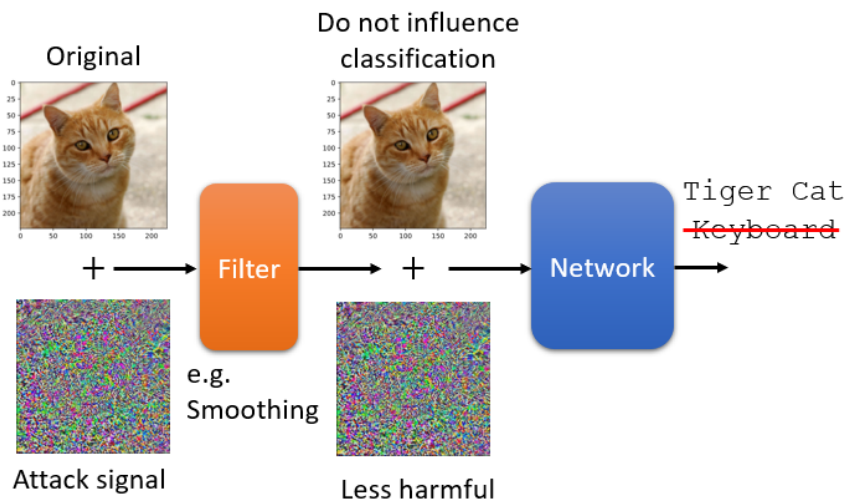
Defense

Passive Defense

到目前為止,我们已经讲了各式各样的攻击的方式,那接下来我们想要讲一下防御的方式,而那**防御呢,大致可以分為两类**

- 一种是被动防御
- 一种是主动防御

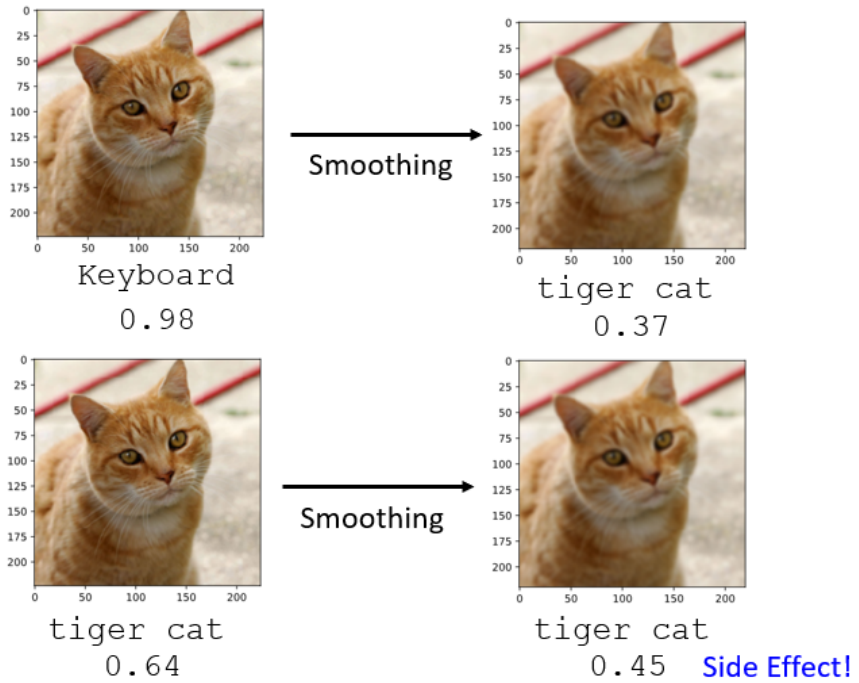
被动防御是怎麼做的呢,**被动防御就是,你的模型是不动**,训练好模型,训练好就训练好了,就放在那边 不要再去动它,但我们**在模型前面加一个盾牌**



加一个 Filter,这个 Filter,可以削减 Attack Signal 的威力,就是当图片通过这个 Filter 的时候,一般的图片不太会受到影响,但是 Attack 的 Signal,通过这个 Filter 以后,它就会失去它的威力,让你的 Network 不会辨识错误

那有人就会想说,要製造什麼样的 Filter,才可以达到这种效果呢,要製造什麼样的 Filter,才能够挡住你的讯号呢,其实你不需要把这个问题想得太复杂,非常简单的做法,光是**把图片稍微做一点模糊化,可能就可以达到非常好的防御效果了**

举例来说,我们刚才已经,我们之前已经看到说,上次看到过说这张图片,加**上了非常小的杂讯以后,影像辨识系统就觉得它是一个键盘**



现在我们把这张图片做一个非常轻微的模糊化,你可以明显感觉说右边这张图片,有一点点模糊,但不是很严重,你还是可以看得出来这张图片裡面有一隻猫,当我们做了这麼一点模糊化以后,再丢到同一个影像辨识系统,你就发现,辨识结果变成是正确了,本来是 Keyboard,现在变成 Tiger Cat

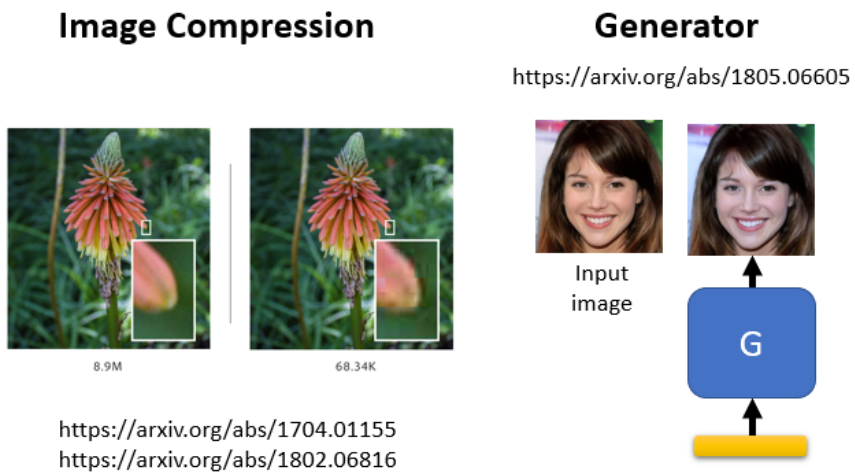
所以光是做模糊化这件事情,就可以非常有效地,挡住 Adversarial Attack

那為什麼呢,因為你可以想说,这个 Adversarial Attack,这个 **Attack 的 Signal,其实只有某一个方向上的某一种攻击的讯号,才能够成功,并不是随便 Sample 一个 Noise,都可以攻击成功**

我们之前已经看过说,你随便 Sample 一个 Noise,并不会达成攻击的效果,所以攻击成功,会让攻击成功的讯号,它是非常特殊的,当你加上那个模糊化以后,那个攻击成功的讯号就改变了,那它就失去攻击的威力,但是它对原来的图片影响甚小,你把原来的图片做一点模糊化,其实不太会影响影像辨识的结果

当然这种模糊化的方法,它也是**有一些副作用**的,比如说本来完全没有被攻击的图片,那 Machine 知道它是 Tiger Cat,但是我们把它稍微模糊化以后,机器现在辨识还是正确的,但是它的 **Confidence 的分数就下降**了,**图片变模糊以后,机器比较不确定,它看到的东西是什麼了**,所以像这种模糊化的方法,你也不能够把模糊这件事情做得太过头,做得太过头的话,它就会造成一些副作用,导致你原来正常的影像,也会辨识错误

其实像这样子的被动防御的方法,还有很多类似的做法,除了做模糊化以外,还有其他更精细的做法,举例来说,有一系列的做法是,直接**对影像做压缩,再解压缩**



你知道你把一张图片啊,存成 JPEG 档以后,那个它就会失真嘛,那也许失真这一件事情,就可以让被攻击的图片 失去它的,失去它的攻击的威力,就可以让攻击的讯号,没有那麼具有伤害性,所以有一系列的做法是,把影像做某种压缩,那这种压缩如果会失真的话,那可能攻击的讯号受到的影响是比较大的,你就可以保护你的模型

还有另外一种方法,是**基於 Generator 的方法**,好 我们在作业裡面,大家都已经训练过 Generator

那有一系列的做法是给一张图片,这张图片它可能有被攻击过,可能没有被攻击过,那我们让我们的 Generator,產生一张跟输入一模一样的图片,也就是**把输入的图片,用 Generator 重新画过,重新產生过**

那你可能会问说,欸 这个在作业裡面,我们的 Generator 只会乱生一些图片啊,你根本没办法控制它生成出来的东西啊,有办法控制 Generator 生成出来的东西,那这个不是今天的重点,我就把文献留在这边给大家参考,总之 Generator,我们有办法控制它的输出,我们要求 Generator 输出一张图片,这张图片跟输入给 Image Classifier 的图片,越接近越好

那你可以想见说,假设有人攻击了这张图片,上面加了一个微小的杂讯是人看不到的,对 Generator 而言,它在训练的时候,它从来没有看过这些杂讯,它可能也无法產生,復现出这些非常小的杂讯,那这样这些微小的杂讯就不见了,Generator 產生出来的图片是没有杂讯的,你就可以达到防御的效果

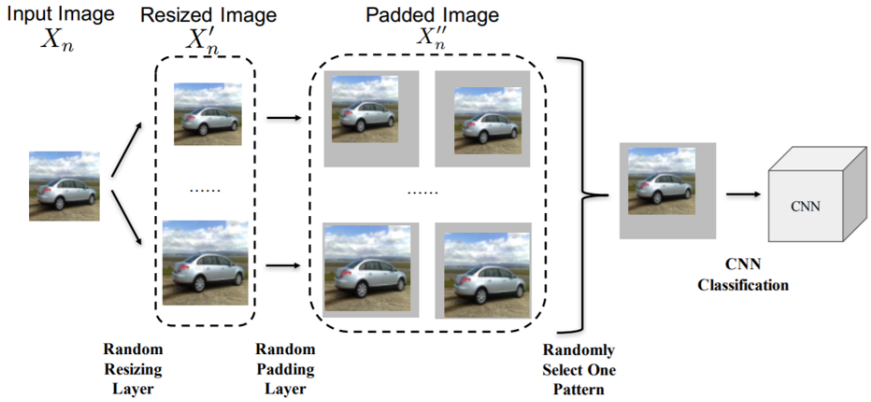
Passive Defense - Randomization

但是这种 Passive 的 Defense 啊,这种被动的防御啊,有一个非常大的弱点,虽然我们刚才在讲的时候,虽然我们刚才在讲这个模糊化的时候,说模糊化非常有效,但是模糊化这一种方法,**只要一旦被别人知道你会做这件事情,它马上就失去效用**

為什麼,你可以完全**把模糊化这件事情,想成是 Network 的第一层**,所以模糊化这件事,等於就是在 Network 前面多加了一层啊,所以假设别人知道你的 Network 前面,多加这一层,把多加这一层放到攻击的过程中,它就可以產生一个 Signal,是可以躲过模糊化这种防御方式的

所以像这种被动的防御,它既强大也不强大,它强大就是,假设人家不知道你有用这一招,它就非常有效,一旦人家知道你用什麼招数,那这种被动防御的方法,就会瞬间失去效用,所以怎麼办呢

还有一种再更强化被动防御的方法,就是**加上随机性**,怎麼做呢,就是你知道,就是不要怎麼样才不会被别人猜中你的下一招,就是你自己都不知道自己的下一招是什麼,这个就是欲欺敌先瞒内的概念,你就在做这个 Defense 的时候啊,加上各种不同的 Defense 的方式



https://arxiv.org/abs/1711.01991

比如说在这篇文献裡面 他们就说,哦 我们输入的图片,我们只要做一些小小的改变,就可以挡住 Attack 的讯号,但是我们**改变的方式不能被别人知道**,别人一知道,他就可以攻破你的防御,所以怎麼办呢,我们自己都不知道图片会怎麼样被改变

一张图片进来以后,你可能把它放大,也可能把它缩小,任意改变它的大小,然后接下来呢,你把这个图片呢,贴到某一个灰色的背景上,但贴的位置也是随机的,你也事先也不知道,你会把这个图片放在灰色背景哪个地方,再丢给你的影像辨识系统,也许透过这种随机的防御,就有办法在,就有办法挡住别人的攻击

但这种随机防御也是有问题,你想想看,假设别人**知道你的随机的 Distribution 的话,他还是有可能攻破这种防御的方式**的,而且我们刚才有说过,**Universal 的 Attacks** 是有可能的,假设你各种随机的可能性都已经被知道的话,那别人只要用 Universal Attacks,它找一个 Attack 的 Signal 可以攻破所有,所有图片的变化方式的话,这样子 Randomization 的方式,还是有可能被突破

Proactive Defense - *Adversarial Training*

那刚才讲的是被动的防御,那还有主动的防御,主动的防御是说,我们在训练模型的时候,**一开始啊,就要训练一个比较不会被攻破的模型**,一开始就要训练一个比较 Robust,比较不会被攻破的模型,那这种训练的方式叫做 Adversarial Training

那这个 Adversarial Training 是怎麼操作的呢,就是你有一些训练资料,这个跟一般的 Training 是一样的,你有 Image,这边用 x 来表示,ImageLabel 用 ŷ 来表示,然后呢,我们就拿我们的训练资料来训练一个模型

训练完以后,接下来你在训练的阶段,就对这个模型进行攻击,你把这边训练的资料,x^1 到 x^n 都拿出来,製造一些 Signal,让这些图片变得具有攻击性,那被攻击后的 Image,叫做 $\tilde{x}$,你把这边 x^1 到 x^n,训练资料裡面的每一张图片,都拿出来进行攻击

Adversarial Training

Training a model that is robust to adversarial attack.

Given training set $\mathcal{X} = \{(\boldsymbol{x^1}, \hat{y}^1), (\boldsymbol{x^2}, \hat{y}^2), \cdots, (\boldsymbol{x^N}, \hat{y}^y)\}$

Using $\mathcal{X}$ to train your model

For $n = 1$ to N

Find adversarial input $\tilde{\boldsymbol{x}}^{\boldsymbol{n}}$ given $\boldsymbol{x^n}$ by an attack algorithm

攻击完以后,你再把这些**被攻击过后的图片,标上正确的 Label,**就你把 x^1 变成 $\tilde{x}^1$ 以后,你的 Machine 就会辨识错误,本来是个猫的图片,它可能就辨识错成键盘,但是你现在把那个辨识错成键盘的图片拿来,重新把它标成猫,因為你已经知道说 x^1,它的 Label 就是猫嘛,所以就算它变成 $\tilde{x}$

Given training set $\mathcal{X} = \{(\boldsymbol{x^1}, \hat{y}^1), (\boldsymbol{x^2}, \hat{y}^2), \cdots, (\boldsymbol{x^N}, \hat{y}^y)\}$

Using $\mathcal{X}$ to train your model

For $n = 1$ to N

Find adversarial input $\tilde{\boldsymbol{x}}^{\boldsymbol{n}}$ given $\boldsymbol{x^n}$ by an attack algorithm

Find the problem

We have new training data

$$\mathcal{X}' = \{(\tilde{\boldsymbol{x}}^{\boldsymbol{1}}, \hat{y}^1), (\tilde{\boldsymbol{x}}^{\boldsymbol{2}}, \hat{y}^2), \cdots, (\tilde{\boldsymbol{x}}^{\boldsymbol{N}}, \hat{y}^y)\}$$

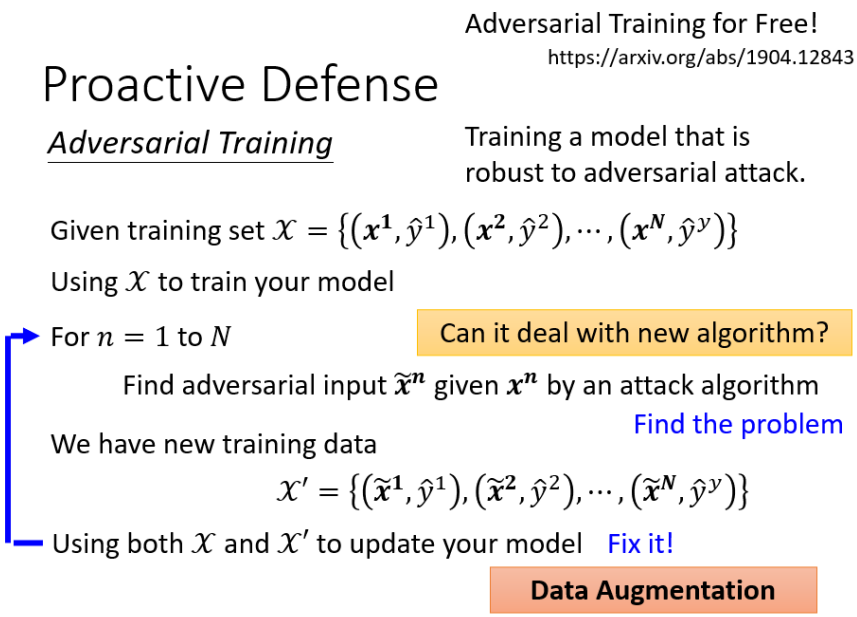
Using both $\mathcal{X}$ and $\mathcal{X}'$ to update your model Fix it!

它现在输入影像辨识系统以后,输入这个你训练好的模型以后,输出的 Label 变了,你也知道原来正确的 Label 是什麼,你就**把原来正确的 Label 拿回来,所以现在就製造了一个新的训练资料,叫 $X^'$**,在新的训练资料裡面,每一笔资料都是有被攻击过的,原来 x^1 到 x^n,变成 $\tilde{x}^1$ 到 $\tilde{x}^n$,

这个 $\tilde{y}^1$ 到 $\tilde{y}^n$,是一样的,那你**再把 X 跟 $ X^'$ 倒在一起,得到更多的训练资料,再重新去训练你的模型**

所以这整个 Adversarial Training 的概念就是,我们先训练好一个模型,然后看看这个模型呢,有没有什麼漏洞,把漏洞找出来,然后接下来呢,再把漏洞填起来,就不断地找漏洞,找到就把它填起来,这个就是 Adversarial Training 的精神

那这个方法啊,其实也可以看作是一种,**Data Augmentation 的方法**,因為我们產生了更多的图片$ X^'$,那再把这些图片加到训练资料裡面,这个等於就是做了资料增强,做了 Data Augmentation 这件事,所以有人也会把 Adversarial Training,当做一个单纯的**资料增强的方式**



就是像这样子的方式,不是只在你的 Model 可能被攻击的时候有用,有时候就算没有人要攻击你的模型,你也可以**用这样的方法產生更多的资料**,然后再把更多的资料拿去做训练,也可以让你的模型,**它的 Robotics 的能力更好,更不容易 Overfitting**,所以就算是没有人要攻击你的模型,你也可以用 Adversarial Training,来强化你的模型,避免 Overfitting 的状况

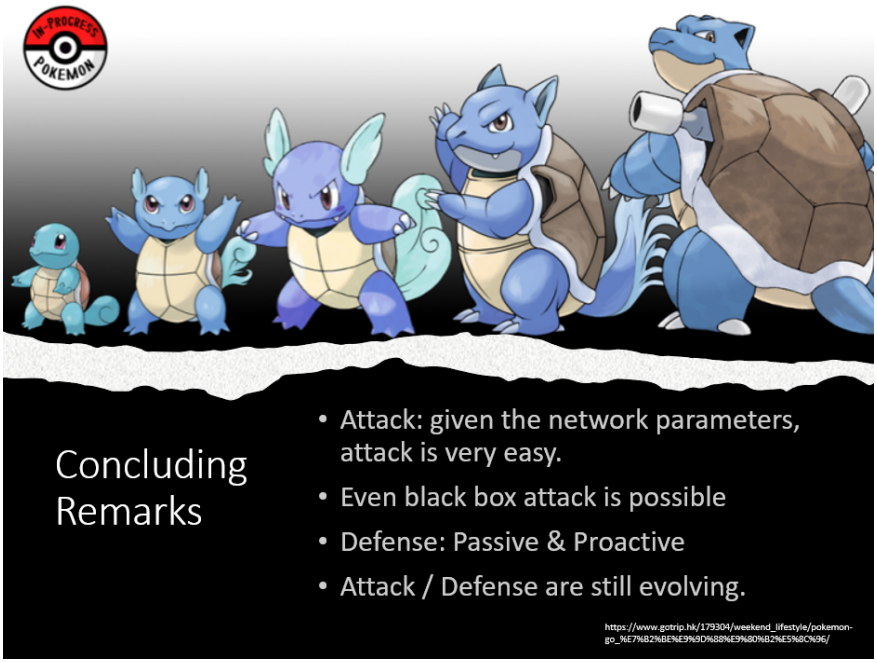
那这个 Process 啊,產生有问题的图片,再重新训练,**这个 Process 啊 是可以反覆做的**,你可以產生图片 重新训练,再產生图片 再產生训练,就不断找出问题补起来,找出问题补起来,这个 Process 是可以反覆做多次,直到你开心為止

那像这样 Adversarial Training,它其实有个非常大的问题就是,**它不见得挡得住新的攻击的方式**,就假设我们今天在找 $ X^'$ 的时候,你用的是 Algorithm ABCD,然后接下来有人在实际攻击的时候,他发明了一个 Algorithm F 去攻击你的模型,往往就能成功,如果今天实际上攻击你 Model 的方法,并没有在 Adversarial Training 的时候被考虑过,那 **Adversarial Training,也不见得能够挡住新的 Attack 的 Algorithm**,所以 Adversarial Training 还是有,还是有可能被攻破的

另外 Adversarial Training,还有一个比较大的问题就是,**它需要非常大,比较多的运算资源**,你想想看,本来一般在训练模型的时候,走到这边就结束了,你有训练资料 训练完模型就结束了,但是 Adversarial Training 它的问题是,首先你要花时间,找出这些$ X^'$,你的图片有几张,你可能就要找出多少张的 $ X^'$,100 万张图片,你要找 100 万个 $ X^'$,光做这件事,可能就已经很花时间了

所以你会发现说,如果你的 Dataset 很大的时候,大家通常就不会想要做 Adversarial Training,所以 Adversarial Training,是一个比较吃运算资源的方法,那為了解决这个问题,有人发明了一个方法叫做,==Adversarial Training For Free==,这边我们就不细讲,有一些方法是做到 Adversarial Training 的效果,却没有 Adversarial Training 那麼大的,Computing 的 Intensity,那至於怎麼做到 Adversarial Training For Free,怎麼不在使用额外的计算的情况下,就达到 Adversarial Training 的效果,那这个把文献放在这边,留给大家参考

那到目前為止呢,我们就是告诉大家,有攻击这件事情,攻击非常容易成功,黑箱攻击也是有可能成功的,然后跟大家介绍了几种经典的 Defense 的方式,那目前攻击跟防御啊,它们都,这些方法仍然不断地在演化



所以在国际会议会不断看到,有新的攻击方法被提出,有**新的防御方法被提出,它们仍然都在进化中,那不知道最后会是谁胜谁负**,好 那这个是今天的现况,