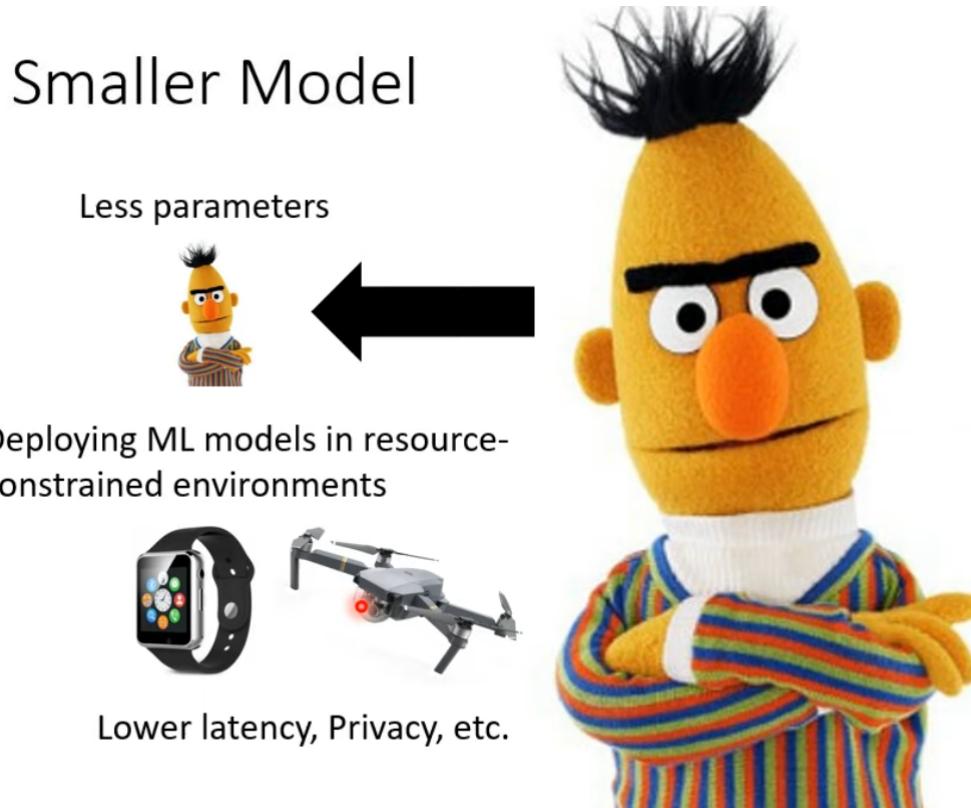


Why we need Network Compression

好那这一堂课呢，要跟大家讲的是 Network Compression。那在这一个课堂上，我们已经看过了很多硕大无比的模型，举例来说 Bert 或者是 GPT。那在这一节课里面，我们要跟大家分享的事情是，我们能不能够把这些硕大无朋的模型，把它缩小，我们能不能够简化这些模型，让它有比较少量的参数，但是跟原来的效能其实是差不多的呢，这就是 Network Compression 想要做的事情，那为什么我们会在意，Network Compression 这件事呢？



因为很多时候，我们会需要把这些模型，用在 Resource-constrained 的环境下，用在资源比较有限的环境下。什么样的环境是资源比较有限的呢，有时候你会需要把这些机器学习的模型，举例来说 跑在智能手表上，举例来说 跑在 Drone 上面，那在这些 Edge Device，在这些 IoT 的 Device 上面，只有比较少的 Memory，只有比较少的 Computing 的 Power。所以我们的模型如果太过巨大，你的手表可能会是跑不动的，所以我们会需要比较少的模型。

那讲到这边有人就会问说，为什么我们会需要，在这些 Edge Device 上面跑模型呢？我们为什么不把资料传到云端，直接在云端上做运算，再把结果传回到 Edge Device，比如说你的手表就好了呢？为什么一定要在手表上面做运算呢？那一个常见的理由是 Latency 的问题，假设你今天需要把资料传到云端，云端计算完再传回来，那中间就会有一个时间差。那假设你今天的应用，你今天的 Edge Device 上面，你今天的 Edge Device 是自驾车的一个 Sensor，那也许自驾车的 Sensor，需要做非常即时的回应，你需要把资料传到云端再传回来，那中间的 Latency 太长了，也许会长到是不能接受的。

那接下来有人又会问说，在未来 5G 的时代会不会 Latency，根本就可以忽略不计呢？那这个时候呢，有人会给你另外一个，我们需要在 Edge Device 上面，做 Computing 的理由，这个理由就是 Privacy。就如果我们今天需要把资料传到云端，那这个云端的这个系统持有者，不就看到我们的资料了吗，也许我在做什么事情，我不想让云端系统持有者知道，所以为了保障隐私，也许在智能手表上直接进行运算，在智能手表上直接进行决策，是一个可以保障隐私的做法。

好那这边呢，就跟大家讲一下，Network Compression 的种种理由。

Network Compression

那在这份投影片里面呢，会跟大家介绍五个， Network Compression 的技术，那这五个技术，都是以软体为导向的。我们只是在软体上面，对 Network 进行压缩，那我们都不考虑硬体加速的部分。

当然另外一个支线的研究是，我们想办法在硬体上加速模型的运算，让 Edge Device 上面，跑深度学习的模型更加有效率，不过这是另外一个研究的面向。我们这边就不讨论，任何跟硬体有关系的东西，我们只讨论跟软体有关系的东西。

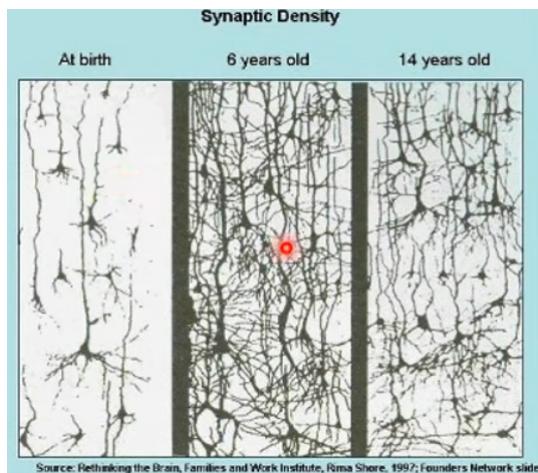
Network Pruning

Network can be pruned

好 那第一个要跟大家分享的，是 Network Pruning 这个技术，那我们今天讲完 Network Pruning 呢，我们就下课。

Network Pruning 顾名思义就是，我们要把 Network 里面的一些参数，把它剪掉，Pruning 就是修剪的意思，我们把 Network 里面的一些参数，把它剪掉，为什么我们可以把，Network 里面的一些参数剪掉呢？因为你知道俗话说的好，树大必有枯枝，一个这么大的 Network，里面有很多很多的参数，那每一个参数不一定都有在做事，参数这么多的时候，也许很多参数，它就只是在划水，它是打酱油的，它什么事也没有做，那这些没有做的参数，放在那边就只是占空间而已，浪费运算资源而已，何不就把它们剪掉呢，所以 Network Pruning 的基本概念，就是把一个大的 Network，其中没有用的那些参数把它找出来，把它扔掉。

那我小时候啊，在也不算小时候，高中的时候，在生物学课本上看过这个图，这个是跟 Network Pruning 好像也有一点关系。这个图是告诉我们说，人刚出生的时候，脑袋是空空的，这些图呢，这些是脑的神经元，脑袋空空的，神经元跟神经元间没什么连结。在六岁的时候会长出非常多的连结，但是随着年龄渐长，有一些连结就慢慢消失了，这个跟我们等一下要做的 Network Pruning，有异曲同工之妙。



那其实 Network Pruning 这件事，不是太新的概念，早在这个 90 年代，Yann Le Cun 就有一篇 Paper，是讲 Network Pruning，那篇 Paper 的 Title 是 Optimal Brain Damage，他把 Brain，他把这个 Network Pruning，把它剪掉一些 Weight 看成是一种脑损伤，Brain 的 Damage。那 Optimal 的意思就是，我们要找出最好的 Pruning 的方法，让一些 Weight 被剪掉之后，但是对这个脑的损伤是最小的。

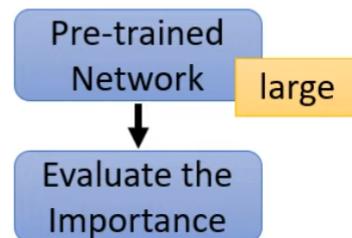
Overall

好那 Network Pruning 的概念，大概是怎么样进行的呢？

它的 Framework 大概是这样子的：首先呢，你先 Train 一个最大的 Network，你 Train 一个大的 Network，然后接下来呢，你去量这个大的 Network 里面，每一个参数，或者是每一个 Neuron 的重要性，去评估一下有没有哪些参数，它是没在做事的，或有没有哪些 Neuron，它是没在做事的，怎么评估某一个参数有没有在做事呢，怎么评估某一个参数重不重要呢，最简单的方法也许就是，看它的绝对值，如果这个参数的绝对值越大，那它可能越能，对整个 Network 的影响越大，或者如果它的绝对值越接近零，那也许对整个 Network 的影响越小，也许对我们任务的影响越小。

Network Pruning

- Importance of a weight:
absolute values, life long ...
- Importance of a neuron:
the number of times it wasn't zero on a given data set



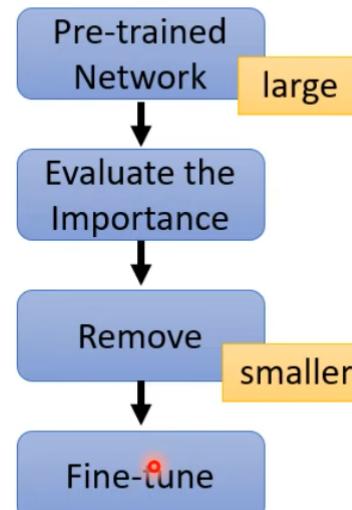
或者是其实你也可以套用，Lifelong Learning 那边的想法，你记得在 Lifelong Learning 里面，我们不是也要看说，哪一些参数比较重要吗？我们不是说一大堆的方法，看哪些参数重要，哪些参数不重要，然后决定 bi 那个首位的值吗？也许我们也可以就把每个参数的 bi 算出来，那我们就可以知道那个参数重不重要，然后把不重要的参数剪掉。

那也可以评估每一个神经元的重要性，我们也可以把神经元当作修剪的单位。那怎么看一个神经元重不重要呢？你就可以比如说，计算这个神经元输出不为零的次数等等，那总之有非常多的方法，来判断一个参数或一个神经元是否重要，那我们在这边就不细讲。

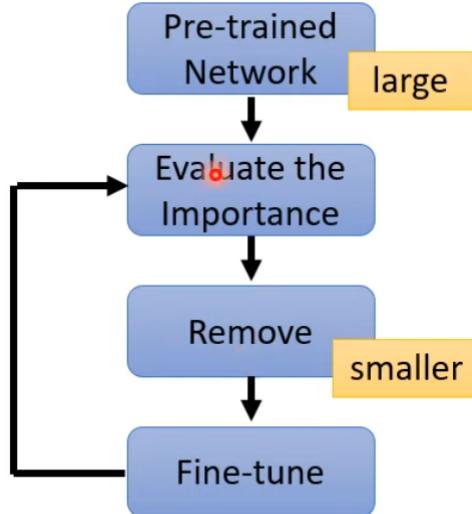
好那把不重要的神经元呢，或是不重要的参数就剪掉，就把它从模型里面移出，那你就得到一个比较小的 Network。

Network Pruning

- Importance of a weight:
absolute values, life long ...
- Importance of a neuron:
the number of times it wasn't zero on a given data set
- After pruning, the accuracy will drop (hopefully not too much)
- Fine-tuning on training data for recover



但是你做完这个修剪以后，通常你的正确率，你的模型的效能就会掉一点，因为有一些参数被拿掉了嘛，所以这个 Network 当然是受到一些损伤，所以正确率就掉一点。但是我们会想办法，让这个正确率再回升一点，怎么让正确率再回升一点呢，就是把这个比较小的 Network，把剩余没有被剪掉的参数，再重新做微调，就把你的训练资料拿出来，把这个比较小的 Network 呢，再重新训练一下，然后训练完之后，其实你还可以重新再去评估一次，每一个参数的正确性，你还可以再 Remove 掉，再剪掉更多的参数，然后再重新进行微调。那这个步骤呢，可以反覆进行多次。



那为什么我们不一次剪掉大量的参数呢？因为在实验上发现说，如果你一次剪掉大量的参数，可能对你的 Network 的伤害太大了，可能会大到你用 Fine-Tune 也没有办法复原。所以一次先剪掉一点参数，比如说只剪掉 10% 的参数，然后再重新训练，然后再重新剪掉 10% 的参数，再重新训练，反覆这一个过程，你可以剪掉比较多的参数。当你的 Network 够小以后，那整个过程就完成了，你就得到一个比较小的 Network，而且这个比较小的 Network，也许它的正确率，跟大的 Network 是没有太大的差别的。

那我们刚才讲到说，修剪的单位可以以参数为单位，也可以以神经元来当作单位，那用这两者当作单位有什么不同呢？用这两者当作单位在实作上，会是有蛮显著的差距的。

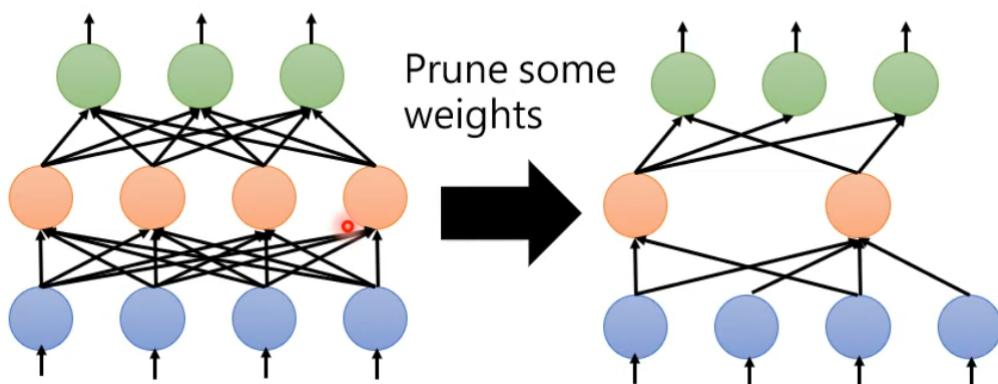
Weight Pruning

如果我们现在是以参数当作单位，会发生什么事？假设我们是要评估说，某一个参数要不要被去掉，某个参数对整个任务而言重不重要，能不能够被去掉，那我们把这个不重要的参数去掉以后，我们得到的 Network，它的形状可能会是不规则的。

Network Pruning - Practical Issue

- Weight pruning

The network architecture becomes irregular.



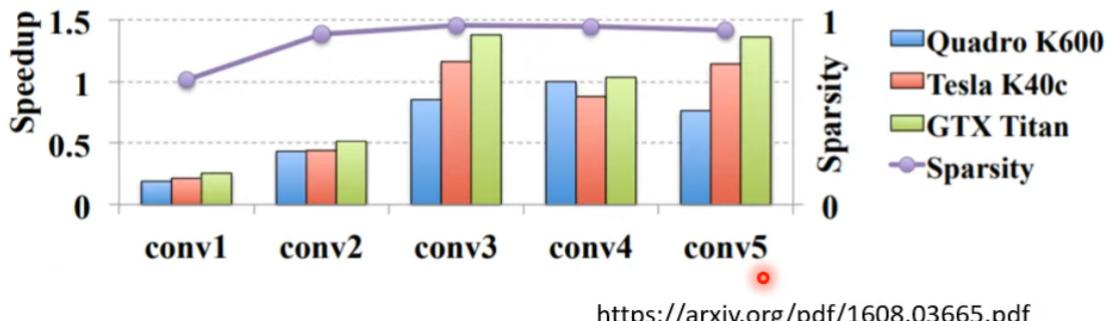
所谓不规则的意思是说，举例来说我们来看红色的这个 Neuron，它连到接下来三个绿色的 Neuron，但第二个红色的 Neuron，它只连到两个绿色的 Neuron，或这个红色的 Neuron，它的输入只有两个蓝色的 Neuron，而这个红色的 Neuron，它的输入有四个蓝色的 Neuron。如果你是把参数当作单位来进行修剪的话，那你修剪完以后的 Network，它的形状会是不规则的，形状不规则会造成什么样的问题呢？最大的问题就是你不好实作啊，你想想看，你用 PyTorch，要实作这种形状不规则的 Network，你好实作吗，你不好实作啊！因为在 PyTorch 里面，你在第一个 Network 的时候，你的定义方法都是，每一层有几个 Neuron 对不对，你都是定义说，我现在每一层要输入几个，输入，输入有几个 Neuron，输出有几个 Neuron，或者输入多长的 Vector，输出有多长的 Vector。这一种形状不固定的 Network，你根本就不好写啊，你自己实作的时候，你根本就不好实作。而且就算你硬是把这种形状不规则的 Network，把它实作出来，你用 GPU 加速也不好加速，GPU 在加速的时候，就是把 Network 的运算，看成一个矩阵的乘法。但是当 Network 是不规则的时候，你就不容易用矩阵的乘法来进行加速，你不容易用 GPU 来进行加速。

那所以实际上啊，在做 Weight Pruning 的时候，在实作上，你可能会把那些 Prune 掉的 Weight 直接补零。就是 Prune 掉的 Weight，它不是不存在，它的值只是设为零。那这样的好处就是你的实作就比较容易，你就比较容易用 GPU 加速。那这样的问题是什么呢？这样的问题是，你根本就没有真的把 Network 变小啊，你这边说这个 Network，它的这个 Link，这个 Weight 它的值是零，你还是存了这个参数啊，你还是存了一个参数，在你的 Memory 里面，你并没有真的把 Network 变小，你只是在想像中把它变小，你在自 High。所以这个是以参数为单位来做 Pruning 的时候，你在实作上会遇到的问题。

这个文献上的实验，就是想要跟你展示说，以参数为单位做 Pruning 的时候，你会遇到什么样的问题。

Network Pruning - Practical Issue

- Weight pruning



<https://arxiv.org/pdf/1608.03665.pdf>

我们先来看紫色的这一条线，紫色的这一条线呢，它说是 Sparsity，这个 Sparsity 是什么意思？这个 Sparsity 就是有多少百分比的参数，现在被 Prune 掉了。那你发现说这边啊，这个紫色的这条线，它的值都很接近 1，什么意思，代表有接近大概 95% 以上的参数，都被 Prune 掉了，这个 Network Pruning 的方法，其实是一个非常有效率的方法，往往你可以 Prune 到 95% 以上的参数，那但是你的 Accuracy 只掉 1~2% 而已。所以这边参数 Prune 的是，Prune 得非常凶的，有 95% 的参数都被丢掉了，照理说丢掉了 95% 的参数，只剩下 5% 的参数，这个 Network 变得很小，它的运算要很快吧？但实际上你发现根本就没有加速多少，甚至可以说根本就没有加速，如果你看这些长条图，这些长条图显示的是，在三种不同的 Computing 的 Resource 上面，你 Speedup 的程度，你加速的程度，那加速的程度要大过 1 才有加速嘛，加速程度小于 1 其实是变慢的。结果你发现说，在多数情况下根本就没有加速，多数情况下其实都是变慢，也就是你把一些 Weight Prune 掉，结果你的 Network 形状变得不规则，然后你真的用 GPU 加速的时候，你反而没有办法真的加速它，所以 Weight Pruning，不见得是一个特别有效的方法。

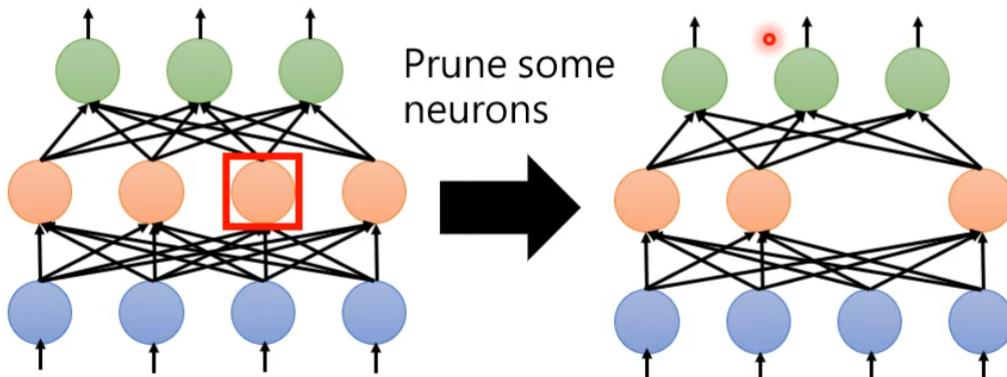
Neuron Pruning

那 Neuron Pruning, 以神经元为单位来做 Pruning, 也许是一个比较有效的方法。

如果我们用神经元做单位来 Pruning, 丢掉一些神经元以后, 你 Network 的架构仍然是规则的, 简单来说就是你用 PyTorch 比较好实作, 你实作的时候, 你只要改那个每一个 Layer, Input Output 的那个 Dimension 就好了, 所以你比较好实作, 也比较好用 GPU 来加速。这个是在 Network Pruning 实作上, 可能会遇到的问题, 好 我们来看一下大家有没有问题要问,

- Neuron pruning

The network architecture is regular.



Easy to implement, easy to speedup

好 那接下来我们就要问一个问题，你说我们先 Train 一个大的 Network, 再把它变小, 而且说小的 Network 跟大的 Network, 它们的正确率没有差太多, 那我们怎么不直接, Train 一个小的 Network 就好了呢? Train 一个小的, 直接 Train 一个小的 Network 比较有效率吧, 还 Train 大的 Network 变小干嘛, 根本是舍本逐末, 为什么不直接 Train 小的 Network, **好 那一个普遍的答案是, 大的 Network 比较好 Train**。你会发现说, 如果你直接 Train 一个小的 Network, 你往往没有办法得到, 跟大的 Network 一样的正确率, 你可以先 Train 一个大的 Network, 再把它变小, 没有 正确率没有掉太多, 但直接去那个小的 Network, 你得不到跟 Pruning, 大的 Network Pruning 完, 变得小的 Network 一样的正确率, 那至于大的 Network 为什么比较好 Train, 那也可以参看以下这个影片的连结, 我在过去的课程有试图解释这件事。好 但是为什么大的 Network 比较好 Train 呢, 那这边有一个假说叫做大乐透假说。

- Larger network is easier to optimize?

https://www.youtube.com/watch?v=_VuWvQU_MQvk

- Lottery Ticket Hypothesis

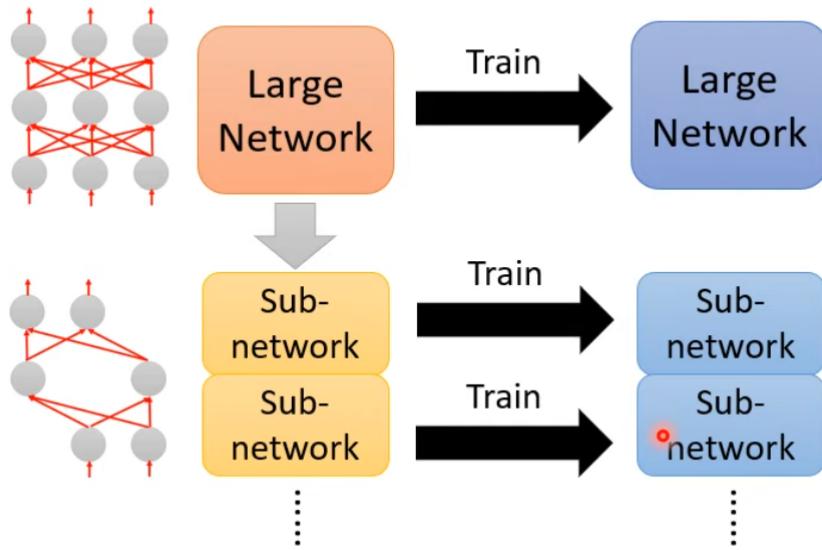
<https://arxiv.org/abs/1803.03635>



那不过它既然叫做假说, 就代表说它不算是完全被实证的一个方, 不算是一个被实证的理论, 如果它是被已经被证明出来的, 那它就是一个理论, 但它现在只是一个假说而已, 那这个大乐透假说是怎么解释? 为什么大的 Network 比较容易 Train, 直接 Train 一个小的 Network, 没有办法得到跟大的 Network 一样的效果, 一定要大的 Network Pruning 变小, 结果才会好呢? 大乐透假说是这样说的, 我们知道说训练 Network, 是一个看人品的事情, 我们现在大家都做过这么多作业了, 我相信你都一定

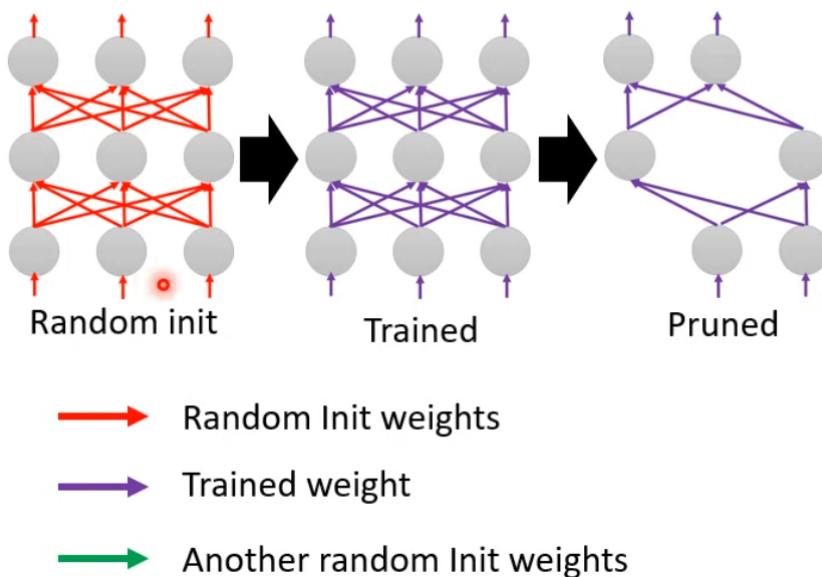
有很多的心酸血泪，你知道 Train Network 就是看人品的，每次 Train Network 的结果不一定会一样，你抽到一组好的 Initial 的参数，你就会得到好的结果，抽到一组坏 Initial 的参数，就会得到坏的结果。

Lottery Ticket Hypothesis

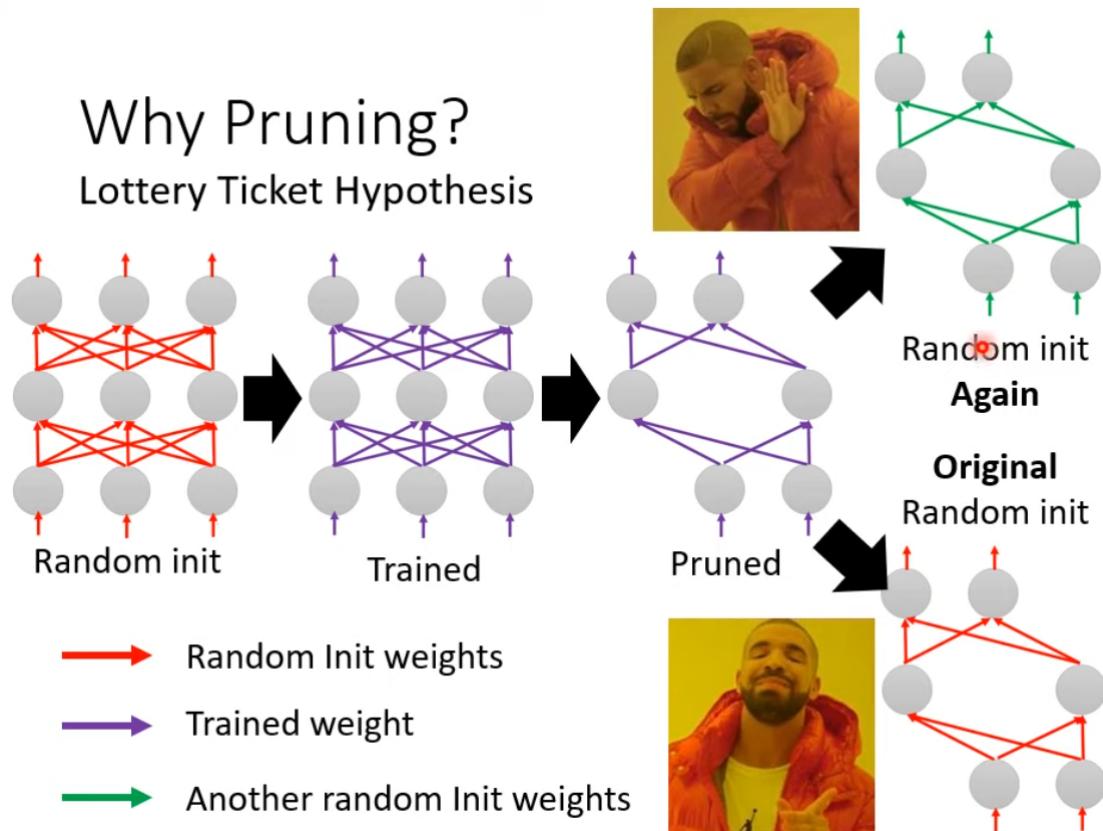


那像这种看这样就好像说，这个乐透也是一个看人品的东西，但是怎么在乐透这个游戏里面，得到比较高的中奖率呢，是不是就是包牌，买比较多的彩券，就可以增加你的中奖率。所以对一个大的 Network 来说也是一样的，大的 Network，可以视为是很多小的 Sub-network 的组合。我们可以想成是一个大的 Network 里面，其实包含了很多小的 Network，当我们去训练这个大的 Network 的时候，我们等于是再训练，同时训练很多小的 Network，那每一个小的 Network，不一定可以成功的被训练出来。所谓成功的训练出来是说，它不一定可以，透过 Gradient Descent，找到一个好的 Solution，我们不一定可以训练出一个好的结果，我们不一定可以让它的 Loss 变低。但是在众多的 Network 里面，众多的 Sub-network 里面，只要其中一个人成功，就可以一人得道 鸡犬升天，其中一个 Sub-network 成功，大的 Network 它就成功了，而今天大的 Network 里面，如果包含的小的 Network 越多，那就好像是去买乐透的时候，包牌包比较多的彩券，买比较多的彩券一样，彩券越多，中奖的机率就越高，所以一个 Network 越大，它就越有可能成功的被训练起来。

Lottery Ticket Hypothesis



那这个大乐透假说，它在实验上是怎么被证实的呢？它在实验上的证实方式，跟 Network 的 Pruning 非常有关系。所以我们就直接看一下在实验上，是怎么证实大乐透假说的，好你现在有一个大的 Network，在这个大的 Network 上面一开始的参数，是随机初始化。好把参数随机初始化以后，得到一组训练完的参数，训练完的参数我们用紫色来表示。接下来你用 Network Pruning 的技术，把一些紫色的参数丢掉，而得到一个比较小的 Network。如果你现在，直接把这个小的 Network 里面的参数，再重新随机的去 Initialize，也就是你重 Train 一个一样大小的，小的 Network，就是你把这个 Network 复制一次，一样大小，但是参数完全不一样重新再训练一次，重新再训练一次，你会发现训练不起来，直接训练这个小的 Network 训练不起来。



训练一个大的再把它变小，没问题，但是直接训练小的训练不起来。但是假设这一个小的 Network，我们再重新 Initialize 参数的时候，我们用的跟这组红色的参数，是一模一样的，就训练得起来。大家可以了解这两者的差别吗，就是这两组参数，虽然都是 Random Initialize 的，但是这组绿色的参数，跟这组红色的参数是没有关系的，而这边这些 Random Initialize 的参数，是直接从这边的红色参数里面，选出对应的参数。就是这边有四个参数，我们就是把这边对应到的这四个参数，直接把它复制过来，这边有四个参数，我们就把这里面对应到的四个参数，直接复制过来，把这里面的参数直接复制过来，就训练了起来。

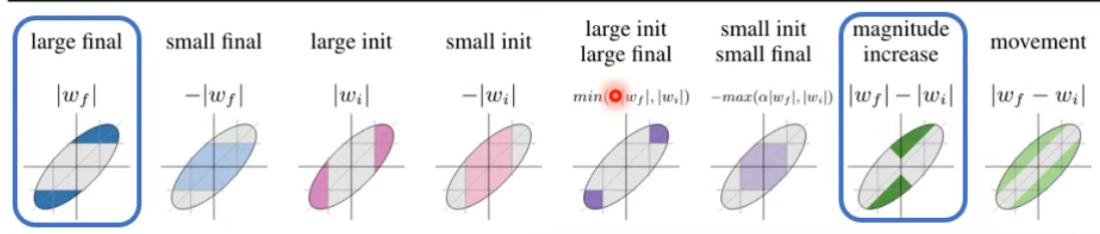
那如果用大乐透假说来解释的话，就是这里面有很多 Sub-network，而这一组 Initialize 的参数，就是幸运的那一组，可以 Train 得起来的 Sub-network。所以当我们今天用，把这些 Net，把这个大的 Network Train 完，再 Pruning 掉的时候，你留下来的正是幸运的那些参数，可以训练得起来的那些参数。所以这一组初始化的参数，它是可以训练得起来的一个 Sub-network。但是如果你再重新随机初始化的话，那你如果运气比较不好，你就抽不到可以成功训练起来的参数。所以这个就是大乐透假说。

那大乐透假说非常地知名，它在 ICLR 2019，应该是 2019 没错，得到 Best Paper Award，所以它是一个非常知名的一个假说，后面也有很多后续的研究，比如说有一篇有趣的研究叫做，Deconstructing Lottery Tickets，解构这个大乐透，那这个解构大乐透里面有什么有趣的结论呢？我们就直接讲它的结论，

Why Pruning?

Lottery Ticket Hypothesis

- Different pruning strategy



第一个，它试了不同的 Pruning Strategy，然后发现说，某两个 Pruning Strategy 是最有效的。那在这些细节我们就不讲，它做了一个非常完整的实验告诉你说，Pruning 有哪些可能的 Strategy，然后它发现说，如果训练前跟训练后，它的绝对值差距越大，那 Pruning 那些 Network，得到的结果是越有效的。

那另外一个比较有趣的结果是，到底我们今天这一组好的 Initialization，是好在哪里呢？它发现说，如果我们，我们只要不改变参数的正负号，就可以训练起来。就小的 Network 只要不改变正负号，就可以训练起来。什么意思呢，就是假设你 Pruning 完以后，剩下的这个数，假设你 Pruning 完以后，那你再把原来，Random Initialize 的那些参数拿出来，它的值是这个样子， $0.9\ 3.1\ -9.1\ 8.5$ ，你可以完全不管它的数值，直接把正的数值，大于 0 的通通都用 $+\alpha$ 来取代，小于 0 的都用 $-\alpha$ 来取代，用这组参数，去 Initialize 你的 Model，这样也 Train 得起来。会跟用这组参数去 Initialize 差不多，所以这个实验告诉我们说，**正负号是初始化参数，能不能够训练起来的关键**，它的绝对值不重要，正负号才重要。然后它在那个文章的，那个标题上 它在章节标题上，特别取了一个，就是它，如果你乍看之下会以为它想要讲，Significance Of Initial Weights，但它故意在这个 Sign 后面加了一个 $-$ ，告诉你是 Significance，就是这个正负号是很重要的。它想要玩个一语双关，不过好像没什么人注意到就是了。

- “sign-ificance” of initial weights: Keeping the sign is critical

$$0.9, 3.1, -9.1, 8.5 \dots \rightarrow +\alpha, +\alpha, -\alpha, +\alpha \dots$$

然后最后一个神奇的发现是，它发现既然我们在想说，一个大的 Network 里面，有一些 Network，有一些 Sub-network，它是特别是好的初始化的参数，它训练起来会特别地顺利，那会不会一个大的 Network 里面，甚至其实已经有一个 Sub-network，它连训练都不用训练，直接拿出来就是一个好的 Network 呢？我们完全不用训练 Network，我们直接把大的 Network Pruning — Pruning，就得到一个可以拿来做分类的 Classifier 了，有没有可能是这个样子的呢？就好像米开朗基罗说，他是怎么雕出大卫像的呢，他不是雕出大卫像，他是把大卫像从石头里面释放出来，大卫像原来就是石头里面，原来就在石头里面，他只是把多余的地方把它去掉而已。那会不会在整个大的 Network 里面，算参数都是随机的，其中已经有一组参数，它就已经可以做分类了，把多余的东西拿掉，直接就可以得到好的分类结果的呢？答案是 **是** 这样子。你可以自己去读一下那个文章，其实可以得到跟 Supervise，其实很接近的正确率。

其实我看到这篇文章的时候，这个结论已经没有让我觉得非常神奇了，因为在这个，解构大乐透这篇文章发表的前几个月，就有一篇文章叫做，Weights Agnostic Neural Networks。它是说，它弄了一个神奇的 Network，这个 Network 里面，所有的数值要么是随机的，要么通通都设 1.5 这样，结果这个 Network，也可以得到一定程度的结果。所以看起来就算，你的 Network 里面参数都是随机的，或根本就给它一个 Constant，得到的，也有可能可以得到好的 Performance，所以这个结论其实没有让人特别讶异，因为在几个月之前就已经有同样的文章了。

Weight Agnostic Neural Networks <https://arxiv.org/abs/1906.04358>

Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask
<https://arxiv.org/abs/1905.01067>

这边放了这两篇文章，Arxiv 就有类似的说法了，这边放了这两篇文章的 Arxiv 连结，这边放的连结是，它们最后一个上传到 Arxiv 的版本，所以如果你看这个，上最后一次最后一个版本上传的月份的话。你会觉得，Weight Agnostic Networks 是后出来，然后解构大乐透是先出来。但如果你看第一个上传的版本的话，是先有 Weight Agnostic Networks，然后才有解构大乐透，所以我是先读到这篇才读到解构大乐透的。

但是大乐透假说它一定是对的吗？不一定。有一篇文章是打脸大乐透假说，这篇文章叫做，Rethinking The Value Of Network Pruning，而且神奇的是，这篇文章跟大乐透假说是同时出来的，它们同时出现在 ICLR 2019。所以就是在 ICLR 2019 里面有两篇文章，它们得到了不太一样的结论，那篇文章说的是什么呢，这篇文章说好它试了两个 Dataset，还有好几种不同的模型，好然后呢，它说，这个是没有 Pruning 过的 Network 的正确率，然后它试着去 Pruning 了一下 Network，然后再重新去做 Fine-tuned 小的 Network，可以跟大的 Network 得到差不多的正确率。

• Rethinking the Value of Network Pruning

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (± 0.16)	VGG-16-A	93.41 (± 0.12)	93.62 (± 0.11)	93.78 (± 0.15)
	ResNet-56	93.14 (± 0.12)	ResNet-56-A	92.97 (± 0.17)	92.96 (± 0.26)	93.09 (± 0.14)
	ResNet-110	93.14 (± 0.24)	ResNet-110-A	93.14 (± 0.16)	93.25 (± 0.29)	93.22 (± 0.22)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	73.03
			ResNet-34-B	72.29	72.55	72.91

- New random initialization, not original random initialization in “Lottery Ticket Hypothesis”
- Limitation of “Lottery Ticket Hypothesis” (small lr, unstructured)

然后它说一般人的想像是，如果我们直接去 Train 这个小的 Network，正确率会不如大的 Network，Pruning 完以后的结果。它试了第一次实验，叫做 Scratch-E。Scratch-E 的意思就是，它的参数是随机初始化的，要注意一下它的随机初始化，跟大乐透假说里面随机初始化是不一样，它的随机初始化就是真的是随机初始化。大乐透假说里面的随机初始化是从，原来的那一组随机的参数里面，去借来那种随机的参数。我知道这个拗口，希望你知道我在说什么，好这个，这个 Scratch，这边这个 Scratch-E 的意思是说，我们就真的随机初始化参数，训练一个小的 Network 跟这个，有 Pruning 过的 Network 它的大小是一样的，发现果然差了一点，跟多数人的想像好像是一样的。

但是接下来它说，如果我们在 Update 的时候，多 Update 几个 (A part) 会怎样呢？之前的人设定的 (A part) 数目，小的 Network 的 (A part) 的训练数目，都跟大的 Network 一样，但是如果小的 Network 的 (A part) 的数目，多设一点会怎样呢，多设一点，就比 Pruning 过以后的结果就好了。所以之前，觉得小的 Network 训练不起来，要先训练大的再做 Pruning，这个会不会就是个 Illusion，就是个幻觉，是个都市传说。直接训练小的 Network，(A part) 设多一点，反正就是训练得起来，就是这样子。其实这篇文章里面，这篇文章放在那个，在 ICLR 审查的时候，当然就有 Reviewer 去问说，你这个跟大乐透假说正好是相反的，你有没有什么 Comment，那其实在这篇文章里面，它也有对大乐透假说做出一些回应，它觉得大乐透假说是比较有，大乐透假说观察到的现象，也许只有在某一些特定的情况下才观察得到。那根据这篇文章的实验是说，只有在 Learning Rate 设比较小的时候，还有 Unstructured 的时候，Unstructured 的时候，就是我们 Pruning 的时候是，以 Weight 作为单位来做 Pruning 的时候，才

能观察到大乐透假说的现象。它发现说 Learning Rate 调大，它就观察不到大乐透假说的这个现象，所以到底大乐透假说，有多正确 是真是假，这个未来尚待更多的研究来证实，

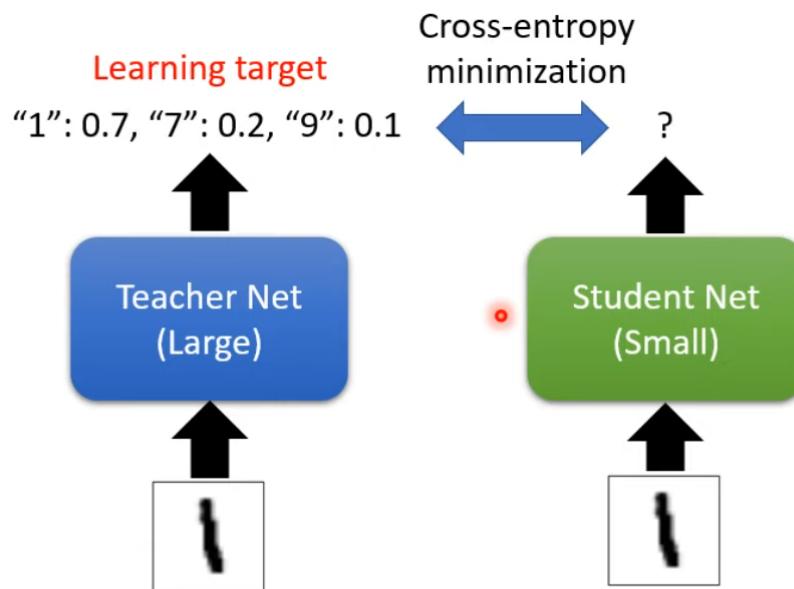
Knowledge Distillation

Overall

好那我们来上课吧，那上周，我们讲到 Network Pruning 那接下来呢，我们讲下一个，可以让 Network 变小的方法，这个方法呢，叫做 Knowledge Distillation。那等一下你听完 Knowledge Distillation，会发现它的精神啊，跟 Network Pruning 其实也有一些类似的地方。



好那 Knowledge Distillation 是什么呢？它的概念是这样，我们先 Train 一个大的 Network，那这个大的 Network 啊，在 Knowledge Distillation 里面，叫做 Teacher Network，它是老师。那你要 Train 的，你真正想要的那个小的 Network，叫做 Student Network。你先 Train 一个大的 Network，叫做 Teacher Network，再根据这个大的 Network，来制造 Student Network。那在 Network Pruning 里面，你是直接把那个大的 Network，做一些修剪，把大的 Network 里面，其中一些参数拿掉，就把它变成小的 Network。那在 Knowledge Distillation 里面呢，是不一样的。这个小的 Network，这个 Student Network，是去根据这个 Teacher Network 来学习。因为 Student Network，是根据 Teacher Network 来学习，所以一个就叫做 Teacher，一个就叫做 Student。



好那这个 Student，是怎么根据这个 Teacher 来学习的呢？这个学习的方法是这个样子的，假设我们现在要做的，就是手写数字辨识，那你就把你的训练资料，都丢到 Teacher 里面，然后 Teacher 呢，就产生 Output，那因为今天是一个分类的问题，所以 Teacher 的输出呢，其实是一个 Distribution，其实是一个分布。举例来说，Teacher 的输出可能是，看到这张图片，1 的分数是 0.7，7 的分数是 0.2，9 这个数字的分数是 0.1 等等。好那接下来学生要做的事情就是，给学生一模一样的图片，但是学生呢，不是去看这个图片的正确答案来学习，它把老师的输出，就当做正确答案。也就是老师输出 1 要 0.7，7 要 0.2，9 要 0.1，那学生的输出呢，也要尽量去逼近老师的输出，尽量去逼近 1 是 0.7，7 是 0.2，9 是 0.1，这样的答案，那你可能会问说，那如果老师犯错了怎么办呢，如果老师答案是错的怎么办呢，不要管它，学生就是根据老师的答案学，就算老师的答案是错的，学生就去学一个错的东西。

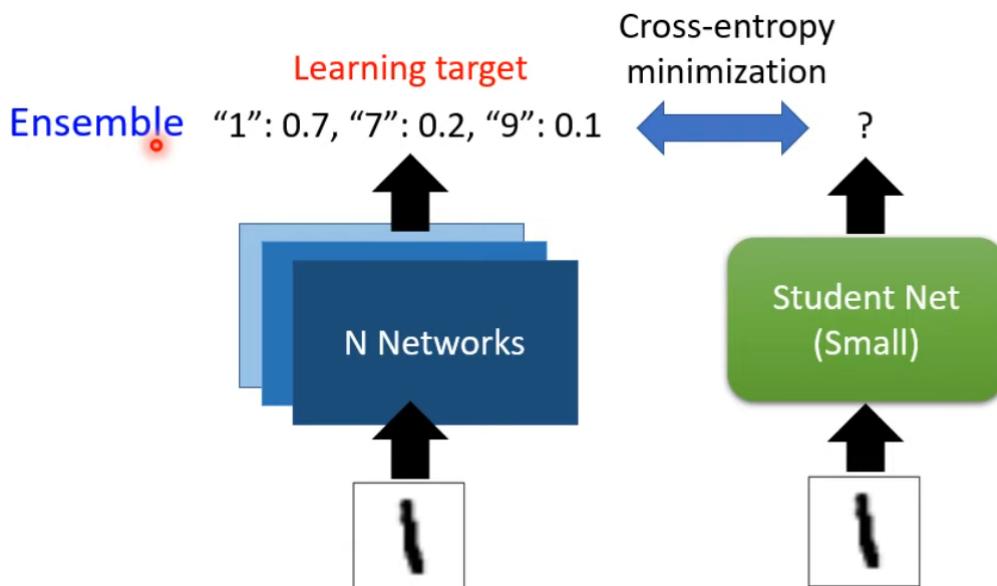
好那讲到这边，你就会想说，那为什么不直接 Train 小的 Network 就好了呢？为什么不直接把小的 Network，去根据正确答案学习，而是要多加一个步骤，先让大的 Network 学，再用小的 Network 去跟大的 Network 学习呢？那这边的理由跟 Network Pruning 是一样的，记不记得上周，我们讲 Network Pruning 的时候，我们讲说，为什么不直接训练一个小的 Network 呢？最直觉，最直觉最简单的答案就是，因为你直接 Train 一个小的 Network，往往结果就是没有从大的 Pruning 来得好。那 Knowledge Distillation 的概念是一样的，为什么不直接 Train 小的 Network，因为直接训练一个小的 Network，没有小的 Network 根据大的 Network 来学习，结果要来得好。

那其实 Knowledge Distillation 这个技术啊，它也不是新的技术，Knowledge Distillation 最知名的一篇文章，是这个 Hinton，在 15 年的时候，就已经发表了。很多人会觉得 Knowledge Distillation，算是 Hinton 提出来的，因为 Hinton 有一篇文章，他的 Title 呢，就是 Knowledge Distillation of Neural Network，那但是其实在 Hinton 提出，Knowledge Distillation 这个概念之前，我其实就有看过其他文章，使用了一模一样的概念。举例来说，在一篇文章叫做，它的 Title 很 Fancy，Title 叫做 Do Deep Nets Really Need to be Deep，是一篇 13 年的文章里面，他也提出了 Network Distillation 的想法。那这个显然是一个比较早年的文章，在 13 年的时候，那个时候还不是所有人都已经被说服说，Deep Learning 就是很棒的，所以那个时候呢，你还需要写一些文章，做一些研究。来 Justify 说 Deep Learning 是好的，所以这篇文章，它的标题就是 Deep Network，到底需不需要是 Deep，到底 Deep 的好处在哪里。

好那为什么 Knowledge Distillation 会有帮助呢？一个比较直觉的解释是说，Teacher Network，其实会提供这个 Student Network 额外的资讯。那如果你直接跟 Student Network 说，这个是 1，可能太难了，因为 1 可能跟其他的数字，比如 1 跟 7 也有点像，1 跟 9 也长得有点像，所以对 Student Network 来说，你告诉它说，看到这张图片你要输出 1，然后 7 啊 9 啊，它们的分数都要是 0，可能很难，它可能学不起来。所以让它直接去跟老师学，老师会告诉它说，就算 1 呢，你没有办法让它是 1 分，也没有关系，其实 1 跟 7，是有点像的，老师都分不出 1 跟 7 的差别。老师说 1 是 0.7，7 是 0.2，学生只要学到 1 是 0.7，7 是 0.2 就够了，那这样反而可以让小的 Network，学得比直接 Trained From Scratch，直接根据正确的答案要学，来得要好。

那其实 Knowledge Distillation 有些神奇的地方，如果你看那个 Hinton 的 Paper 里面，它甚至可以做到，光是 Teacher 告诉 Student，哪些数字之间，有什么样的关系这件事情，就可以让 Student，在完全没有看到某些数字的训练资料下，它就可以把那一个数字学会。假设今天训练资料里面，完全没有 7 这个数字，但是 Teacher，它在学的时候有看过 7 这个数字，但是 Student 从来没有看过 7 这个数字，但光是凭着 Teacher 告诉 Student 说，1 跟 7 有点像，7 跟 9 有点像，这样子的资讯，都有机会让 Student 可以学到 7 长什么样子。就算它在训练的时候，从来没有看过 7 的训练资料。好这个是 Knowledge Distillation 的基本概念。

Ensemble



好那这个 Teacher Network, 不一定要是单一个巨大的 Network, 它甚至可以是多个 Network 的 Ensemble。Ensemble 是什么呢, 到目前为止这门课, 我们其实没有正式介绍过 Ensemble 这个词汇, 虽然它是 Deep Learning 里面, 尤其是在机器学习的比赛里面, 常常用到的一个技巧, 不过这个技巧非常地简单, 所以其实也不太需要真的花太多时间介绍, 它的概念很简单就是, 你就训练多个模型, 然后你输出的结果就是多个模型, 投票的结果就结束了。或者是把多个模型的输出, 平均起来的结果, 当做是最终的答案, 那这个就是 Ensemble。

那虽然在比赛里面, 常常会使用到 Ensemble 的方法, 今天如果在一个机器学习的比赛里面, 你要名列前茅, 往往凭借的, 就是超级的 Ensemble, 就是训练个, 100 个模型啊, 1000 个模型啊, 把那么多的模型的结果通通平均起来。往往你要在机器学习的, 这种 Leaderboard 上面名列前茅, 靠的就是这种 Ensemble 的技术。但是在实用上啊, Ensemble 会遇到的问题就是, 你训练了 1000 个模型, 进来一笔资料, 你要 1000 个模型都跑过, 再取它的平均, 这个计算量也未免太大了吧, 打比赛还勉强可以, 要用在实际的系统上, 显然是不行的。

那怎么办呢? 你可以把多个 Ensemble 起来的 Network, 综合起来变成一个, 那这个就要用 Knowledge Distillation 的做法。你就把多个 Network Ensemble 起来的结果, 当做是 Teacher Network 的输出, 然后让 Student Network, 去学这个 Ensemble 的结果, 就 Ensemble 的输出是什么样子, 就让 Student Network, 去学这个 Ensemble 的输出。那这样子你就可以让 Student Network, 去逼近这个 Ensemble, 一堆 Network Ensemble 起来, 有的这个正确率, 那这个呢, 是 Knowledge Distillation 在 Ensemble 上的应用。

Temperature

那在使用 Knowledge Distillation 的时候, 有一个小技巧啦, 这个小技巧是, 你会稍微改一下 Softmax 的 Function, 你会在 Softmax 的 Function 上面, 加一个 Temperature。这个 Temperature 是什么呢, 我们马上就会看到。

好那跟大家简单的复习一下 Softmax, 那我们呢, 在开学期初的时候, 在讲这个分类的时候, 就跟大家介绍过 Softmax。Softmax要做的事情就是, 你把每一个 Neural 的输出, 都去 Exponential, 然后再做 Normalize, 得到最终 Network 的输出。那这样可以让你的 Network 的输出, 变成一个机率的分布, Network 的输出, 最终的输出, 分子都是介于 0 到 1 之间的。

- Temperature for softmax

$$y'_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)} \quad T = 100 \quad \rightarrow \quad y'_i = \frac{\exp(y_i/T)}{\sum_j \exp(y_j/T)}$$

$y_1 = 100$	$y'_1 = 1$	$y_1/T = 1$	$y'_1 = 0.56$
$y_2 = 10$	$y'_2 \approx 0$	$y_2/T = 0.1$	$y'_2 = 0.23$
$y_3 = 1$	$y'_3 \approx 0$	$y_3/T = 0.01$	$y'_3 = 0.21$

好那所谓的 Temperature 的意思就是, 在做 Exponential 之前, 把每一个数值呢, 都除上一个 T, 那这个 T 呢, 是一个 Hyperparameter。T 呢, 是一个你需要调的参数, 那除这个 T 有什么作用呢? 假设 T 大于 1 的话, 那这一个 T, 也就是 Temperature, 它的作用就是, 把本来比较 Sharp, 比较集中的分布, 让它变得比较平滑一点。那为什么要让集中的分布, 变得比较平滑一点呢? 我们这边举一个例子, 假设你的 $y_1 y_2 y_3$, 在做 Softmax 之前, 它的值分别是 100 10 跟 1。那做完 Softmax 以后, 你会发

现, y_1' 就是 Softmax 的结果, y_1' 是 1, $y_2' y_3'$ 都趋近于 0。那假设这是你的输出, 这个是你的 Teacher Network 的输出, 你叫你的学生, 要叫你的 Teacher Network 去跟这个结果学, 那跟直接和正确的答案学, 完全没有不同啊。跟 Teacher 学的一个好处就是, 老师会告诉你说, 哪些类别其实是比较像的, 让 Student Network 在学的时候不会那么辛苦。但是假设老师的输出, 非常地集中, 就是其中某一个 Class 是 1, 其他都是 0, 那这样子跟正确答案学有什么不同呢, 就没有不同啦。所以你要取一个 Temperature, 假设我们这边 Temperature 设为 100, 那你就是把 $y_1 y_2 y_3$ 都除 100, 就变成 1, 0.1 跟 0.01。那对于老师来说, 其实加上这个 Temperature 分类的结果, 是不会变的, 做完 Softmax 以后, 最高分的还是最高分, 最低分的还是最低分。所有 Class 的, 这个排序, 是不会变的, 分类的结果是完全不会变的。但好处是, 每一个类别得到的分数, 会比较平滑, 比较平均。那你拿这一个结果去给 Student 学, 才有意义, 才能够把 Student 呢, 学得好。那这个是 Knowledge Distillation 的一个小技巧,

QA

Q: 拿 Softmax 前的输出, 拿来 Train, 会发生什么事呢,

A: 你完全可以拿 Softmax 前的输出, 拿来 Train, 其实还会有人拿, 这个 Network 的每一层, 都拿来 Train, 你可以说。欸 我有一个大的 Teacher Network, 它是第 1 层, 它有 12 层, 小的 Student Network 有 6 层, 那我要让 Student Network 第 6 层, 像大的 Network 的第 12 层, Student Network 的第 3 层, 像大的 Network 的第 6 层, 可不可以呢, 可以。那往往你做这种比较多的限制, 其实可以得到更好的结果。那这是一个常用的, 在 Knowledge Distillation 常用的小技巧。

Q: 好, 那有同学想问 Lifelong Learning 的问题。上周有提到用 Generator 产生资料, 让模型学习, 但这样模型只要再学一次旧的资料, 有达到 Lifelong Learning 的宗旨吗?

A: 我可以了解你的问题, 你说 Lifelong Learning 就是, 不要一看到旧的资料啊, 那我现在再产生一些旧的资料, 加到训练资料里面, 有没有违反 Lifelong Learning 的精神呢? 这是一个, 见仁见智的问题啦, 那假设我们把 Lifelong Learning 定义成, 你不能去看真正的旧的资料, 但是如果你的 Network, 自己有办法去产生旧的资料, 它并没有去真的把旧的资料存下来啊, 那个旧的资料是它自己生出来的, 那至少今天在 Lifelong Learning 的 Community, 在如果你要写 Lifelong Learning 的 paper 的话, 这样子的方法是, 会被接受是一个 Lifelong Learning 的方法。因为你并没有真的用到旧的资料, 旧的资料是被产生出来的。

Q: 有一个同学问说, 如果学生和老师差太多的话, 可以加一个介于中间的 Network, 让学生学那个 Network 吗?

A: 可以, 就是可以, 那这样子的技术啊, 等一下在助教的录影里面, 助教会跟大家提示, 有这么样的一个方法。

Q: 同学问到说 T 太大的话, 模型会不会改变很多, Temperature 太大的话, 会不会改变很多?

A: 会, 你想, 假设你 Temperature 是个接近无穷大, 那这样所有的 Class 的分数就变得差不多啦, 这样子 Student Network 也学不到东西了。所以 T, 又是另外一个 Hyperparameters, 它就跟 Learning Rate 一样, 这个是你在做 Knowledge Distillation 的时候, 要调的参数。

Q: 好 有同学问说, 老师, 请问 Ensemble 的 Destination, 只有把 Logistic 合在一起的方式吗?

A: 我猜这位同学想要问的问题是说, 在做 Ensemble 的时候, 就我刚才在举例, 告诉你什么是 Ensemble 的时候, 我是举说, 我们可以把很多的 Network 呢, 它的输出平均起来, 这就是一种 Ensemble, 那有没有其他 Ensemble 的方式呢, 其实是有的, 这个说起来就有点话长啦, 我之后, 这个 Ensemble 的这个技术, 其实之前在过去的录影, 其实也是有提过的, 那我之后可以把这个录影呢, 也贴在社团上, 让大家再来研究。那其实 Ensemble 呢, 是有很多各式各样不同的变形的啦, 举例来说, 你在做 Ensemble 的时候, 刚才讲的, 是在那个, Network 的 Output 上面做平均。但事实上还有其他做法, 比如说, 你有看过有人直接在, Network 的参数上面做平均, 那这个 Network 参数上面做平均的方法, 在这一门课里面, 你甚至已经有用过了, 在那个 Translation 那个作业里面, 其实助教的

这个程式就是有，就是已经有帮你做好 Ensemble，就是有把不同的参数做平均，这样子的小技巧，那一招呢，在 Translation 上面，不知道为什么特别有用，所以在助教的程式里面，有实做这个方法，

Parameter Quantization

Overall

那我们就继续啦，那剩下的，这个有关，那个 Network Compression 的部分没有太多啦，我们就赶快把它讲完，接下来就可以放一下助教的录影，好那接下来，下一个跟大家讲的小技巧啊，叫做 Parameter 的 Quantization，不过这一招呢，在我们的作业是用不上的。欸 为什么在我们的作业是用不上的呢，因为我们的作业啊，对大家 Network 大小的限制，并不是看那个，Network 的本身的大 小，而是看你用了几个参数啦。所以你如果不是减少参数的量，单纯是用其他方法来压缩 Network 的话，在这个作业里面，其实是不会得到比较好的结果的，是不会比较有利的。

好 不过还是跟大家介绍一下，这个 Parameter Quantization 的做法，那这个 **Parameter Quantization 的做法，是什么意思呢？它的意思是说，我们能不能够只用比较少的空间，来储存一个参数。**

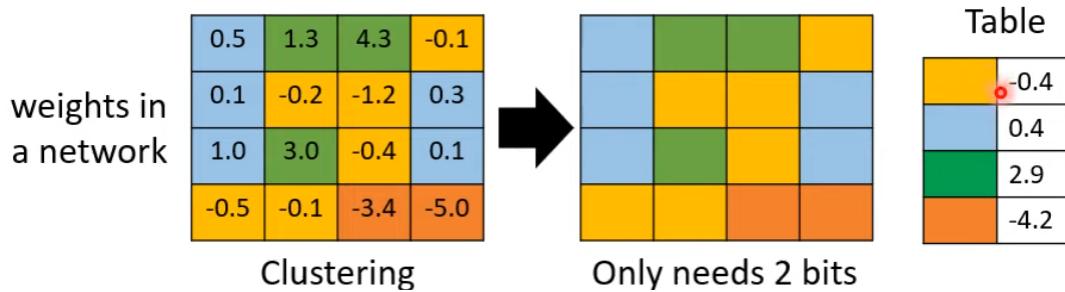
less bits

举例来说，你现在存一个参数的时候，你可能是用 64 个 Bits，你可能是用 32 个 Bits，真的有必要精度这么高吗，会不会用 16 个 Bits 就够了，会不会用 8 个 Bits 就够了，会不会甚至更少的 Bits 就够了，所以 Parameter Quantization，最简单的做法就是，本来如果你存 Network 的时候，举例来说，你是 16 个 Bits 存一个数值，现在改成 6 个 Bits，8 个 Bits 存一个数值，欸 那你的储存空间，你的 Network 的大小，直接就砍半了，而且你的那个 Performance 呢，不会掉很多。甚至有时候，你把你的这个储存那个参数的精度变低，结果还会稍微更好一点。

Weight Clustering

好 那还有一个，再更进一步压缩你的参数的方法呢，叫做 Weight Clustering。

- 2. Weight clustering



Weight Clustering 的意思是什么呢，我们直接举例来跟大家说明，假设这些，是你的 Network 的参数，然后呢，你就做 Cluster，你就做分群，按照这个参数的数值来分群，数值接近的，放在一群，那要分几群呢？你会先事先设定好，举例来说，我们这边事先设定好说，我们要分四群。那你就会发现说，比较相近的数字，就被当做是一群，那接下来每一群，我们都只拿一个数值来表示它，也就是只要分到黄色这一群的，原来它的数值是多少，我们不管，我们就说它都是 -0.4，反正其他人可能，其他参数也都跟 -0.4 的结果，可能都差不多了，这个 -0.4 呢，可能就是这里面所有数值的平均，假如这里面所有数字的平均是 -0.4，我们就用 -0.4，来代表所有黄色的参数，用 0.4 来代表所有蓝色的参数，用 2.9 来代表分到绿色这群的参数，用 -4.2 来代表分到红色这群的参数。

这样做有什么好处呢，这样做的好处是：你今天在储存你的参数的时候，你就只要记两个东西，一个是一个表格，这个表格是记录说，每一群代表的数值是多少。那另外一个呢，你要记录的就是，每一个参数，它属于哪一群。那假设你群的数目设少一点，比如说你设四群而已，那这样子你要存一个参数的时候，你就只要两个 Bits，就可以存一个参数了，你就可以把，本来你存一个参数，你可能要 16 个

Bits, 8 个 Bits, 再进一步压缩到, 存一个参数, 只需要两个 Bits 就好。

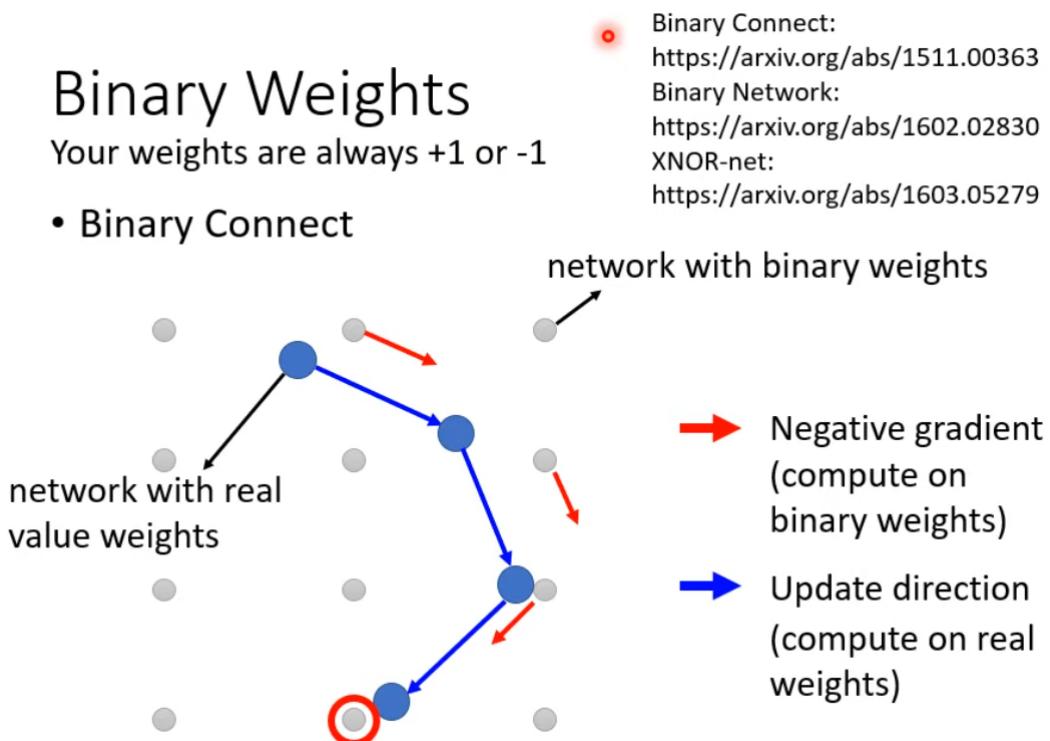
Huffman Encoding

- 3. Represent frequent clusters by less bits, represent rare clusters by more bits
 - e.g. Huffman encoding

好 那你其实还可以把参数, 再更进一步做压缩, 那假设你对通讯比较熟悉的话, 那你可能学过一个东西, 叫做 Huffman Encoding。那 Huffman Encoding 的概念就是, 比较常出现的东西, 就用比较少的 Bits 来描述它, 比较罕见的东西, 再用比较多的 Bits 来描述它, 这样的好处就是常出现的东西, 用少的 Bits 来描述, 罕见的东西, 用多的 Bits 来描述, 这样平均起来, 你需要储存你的资料, 需要的 Bits 的数目就变少了, 所以这个就是 Huffman Encoding。所以你可以用这些技巧呢, 来压缩你的参数, 让你储存每一个参数的时候, 需要的空间呢, 是比较小的。

Binary Weights

好 那到底可以压缩到什么程度呢, 最终极的结果就是, 你可以只拿一个 Bits, 来存每一个参数。你可以说明你的 Network 里面的 Weight, 要嘛就是 +1, 要嘛就是 -1, 只有这两种可能, 没有其他的了。假设你可以做到说, 你所有的 Weight, 就只有正负 1 两种可能, 那你每一个 Weight, 就只需要一个 Bits, 就可以存下来了, 那像这样子的这种 Binary Weights 的研究啊, 其实还蛮多的。那我这边就是列了一些 reference 给大家参考。



那至于实际上, 要怎么训练出这种 Binary 的 Network, 那这个细节, 我们就不讲, 那一般人会对 Binary Network 有的印象就是, 哇 这个一个参数值, 要嘛是 +1, 要嘛是 -1, 会不会这个 Network 的 Performance 很差啊, 这 Network 应该很惨吧, 它应该做得很烂吧, 这倒未必, 这边呢, 就是取那个 Binary Network, 里面的其中一个经典的方法, 叫做 Binary Connect, 它的结果来跟大家分享一下。

在 Binary Connect 的这篇 Paper 里面, 他跟你说, 他把 Binary Connect 这个技术, 用在三个影像辨识的问题上, 从最简单的 MNIST, 还有稍微难一点的 CIFAR-10, 他设计的 Corpus, 那他发现了什么, 这个第一排是正常的 Network, 不是 Binary 的 Network, 一般的 Network, 在 MNIST 上, 你的错误率是 1.3%, 这边是错误率啦, 所以值越小越好, 之后在 CIFAR-10 上是 10%, 他发现, 用 Binary

Connect, 也就是每一个参数，要嘛是 +1，要嘛是 -1，结果居然是比较好的，所以你用 Binary Network，结果居然还比正常的 Network 的 Performance，好一点。那这边的理由可能是说，当我们用 Binary Network 的时候，我们给了 Network 比较大的限制，我们给 Network Capacity 比较大的限制，所以它比较不容易 Overfitting，所以用 Binary 的 Weights，它反而可以达到防止 Overfitting 的效果。所以这边是想要跟大家分享一下，Binary Network 的，厉害的地方。

QA

Q：好 有同学问到说，关于上周 Lifelong Learning，讲到 EWC 的方法，在 Train 每个新的 Task 之前，为了要算每个 Parameter 的重要性，都会用之前的 Data 去算 Gradient，这样是不是算跟，GEM 一样，存有部分之前的 Data 呢？

A：这边是不一样的，因为那个 Gradient，它不是看到新的任务才算的，那个 Gradient，就是我们要算 Parameter 的重要性这件事情，在一个任务训练完之后，你就会马上计算了，所以你并不是等，要解新的任务的时候，才去旧的任务的 Data 上，去计算说每一个参数的重要性，是旧的任务一训练完，马上就把参数性记录下来，把参数的重要性记录下来，那把参数的重要性记录下来以后，旧的任务的所有资料，就都可以被丢掉了。所以它跟 GEM，还是不太一样的，不过我上周有提示说，你如果是用 EWC 这种方法，其实你也用，有用到额外的储存空间啦，所以，虽然它没有存资料，但是它需要储存参数的重要性，那这个也是会耗用一些储存的空间的。

Q：好 有同学问说，Weight Clustering 要怎么做 Update，每次 Update 都要重新分群吗？

A：其实为 Weight Clustering 有一个很简单的方法，你可能以为那个 Weight Clustering，是需要在训练的时候就考虑的，但是有一个简单的做法是，你先把 Network 训练完，再直接做 Weight Clustering，但你这样直接做，可能会导致你的这个，就是 Cluster 后的参数，跟原来的参数相差太大。所以有一个做法是什么呢，有一个做法是，我们在训练的时候啊，要求 Network 的参数，彼此之间比较接近，你可以把这个训练的 Quantization 当做是，Loss 的其中一个环节，直接塞到你的训练的过程中，让训练的过程中达到呢，这个参数呢，有群聚的效果。

Q：好 有同学问说，要压到什么程度，才不会丢掉太多的资讯。

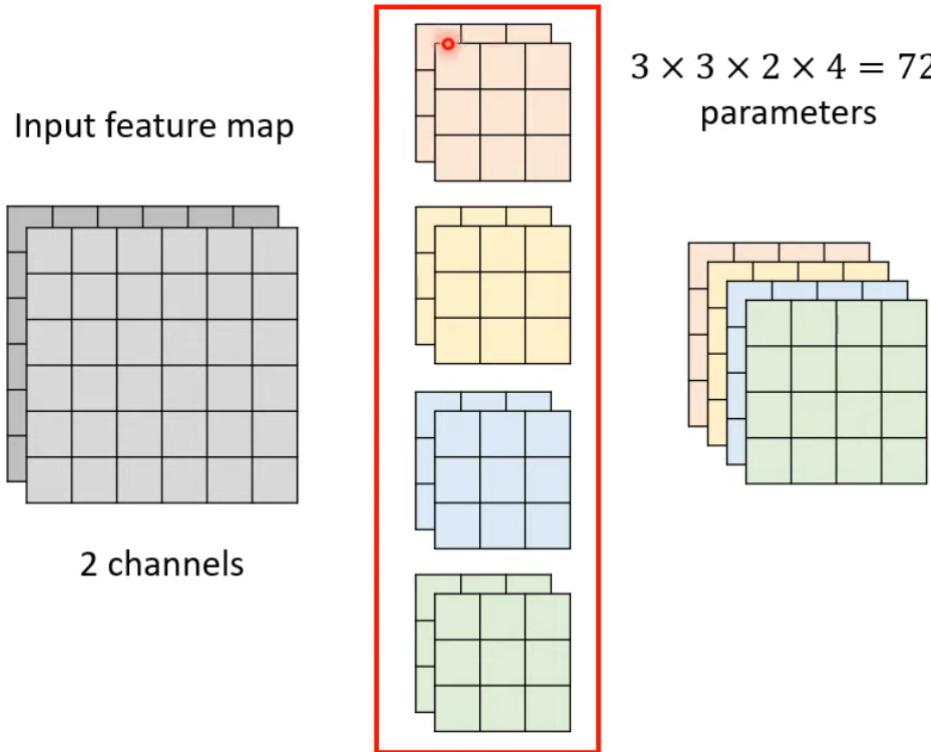
A：那这个，你就要自己算算看啦，那你就要，到底要压到什么程度，那你做压缩的时候，难免就是会掉一些资讯嘛，但到底掉多少资讯你能够接受，那这个就要看你的应用了，好 那个 Weight Clustering 里面，每个 Cluster 的数字要怎么决定呢，决定好每个 Cluster 的区间之后，取它们的平均吗，对 就是决定好每个 Cluster 的区间之后，取它们的平均

Q：好 刚刚有一个同学问说，关于刚刚 EWC 的问题，想请问老师，那样不同的 Task Data 对 Model 的重要性，会不会冲突，怎么把不同 Task 的 Importance Merge

A：这边这个解法，怎么把不同 Task 的 Importance Merge，在文献上的做法，比你想像的要简单，就是，每一个 Task 都会计算出一个重要性，然后把所有 Task 的重要性加起来，就结束了，就是每个参数，在每一个 Task 训练完以后，都会得到一个首位的值，都会得到一个 b，把每一个任务的 b 通通加起来，就结束了，

Architecture Design

接下来啊，要跟大家讲的，是 Network 架构的设计，透过 Network 架构的设计，来达到减少参数量的效果，那等一下要跟大家介绍的，叫做 Depthwise Separable Convolution，那这个会是这次作业的主力啦，你要过 Strong Baseline，就靠这个方法了。



Review CNN

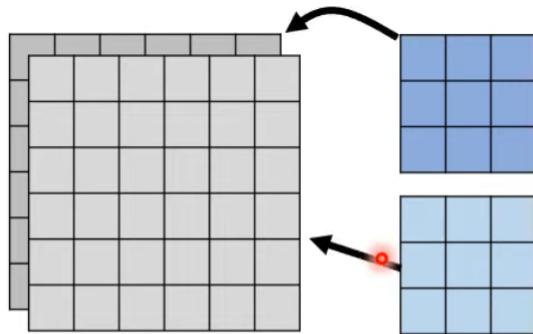
好那这一个方法啊，在讲这个方法之前，我们先帮大家很快地复习一下 CNN，那我们已经跟大家讲过 CNN，那我们说 CNN 是什么呢？在 CNN 的这种 Convolution Layer 里面呢，你每一个 Layer 的 Input，是一个 Feature Map，那在这个例子里面呢，我们的 Feature Map 有两个 Channel，如果你的 Feature Map 有两个 Channel，那你的每一个这个 Filter，它的高度也得是 2。而且这个 Filter 并不是一个长方形，而是一个立方体，你的 Channel 有多少，你的 Filter 就得有多厚。然后你再把这个 Filter 呢，扫过这个 Feature Map，你就会得到另外一个 Mattress，另外一个正方形，你有几个 Filter，那 Output Feature Map 就有几个 Channel，这边有 4 个 Filter，每一个 Filter 都是立方体哦，那 Output Feature Map 就有 4 个 Channel，那在这个例子里面总共有多少参数呢，我们总共有 4 个 Filter，每一个 Filter 的参数量是 $3 \times 3 \times 2$ ，要注意一下每一个 Filter，其实立方体，所以总共的参数量是 $3 \times 3 \times 2 \times 4$ ，总共 72 个参数。

Depthwise Separable Convolution

那等一下跟大家介绍的方法，叫做 Depthwise Separable 的 Convolution，那这个 Depthwise Separable Convolution，它为什么会好呢？

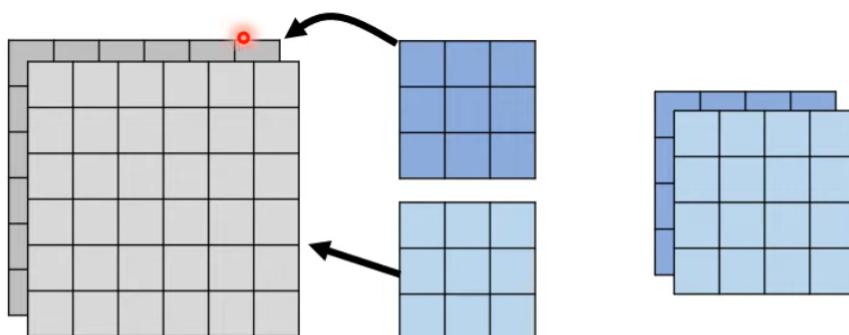
这个等一下再跟大家讲它的原理，在讲它的原理之前，我们先直接讲它的操作，这个 Depthwise Separable 的 Convolution，它分成两个步骤。

第一个步骤呢，叫做 Depthwise 的 Convolution，Depthwise Convolution 做的是什么呢？它要做的事情是，有几个 Channel，我们就有几个 Filter，而每一个 Filter 只管一个 Channel。因为每个 Filter 只管一个 Channel，所以有几个 Filter，有几个 Channel 就有几个 Filter。举例来说，假设 Input Feature Map 是两个 Channel，那在 Depthwise 的 Convolution Layer 里面，我们就只放两个 Filter。不像之前在一般的 Convolution Layer 里面，Filter 的数目跟 Channel 的数目是没有关系的，在前一个投影片的例子里面，Channel 只有两个，但 Filter 可以有四个，但在 Depthwise Convolution 里面，几个 Channel，几个 Filter，然后每一个 Filter 就只负责一个 Channel 而已。



- Filter number = Input channel number
- Each filter only considers one channel.

那其实我觉得这个 Depthwise Convolution, 比较像是大家一般对 CNN 的误解啦。就假设有人对 Convolution 不太熟悉的话, 一般都会误以为 Convolution 的算法, 就跟 Depthwise Convolution 是一样的。其实不是, 你已经修过这一门课了, 已经花很多时间跟大家介绍过 CNN 了, 所以你一定知道说 CNN 不是这样算的, 但是 CNN 里面一个特别的变形, 减少参数量的变形, 其实跟大家一般对 CNN 的误解蛮像的, 就是每一个 Filter 只去管一个 Channel 就好, 那一个 Filter 怎么管一个 Channel 呢? 假设这个浅蓝色的 Filter 管第一个 Channel, 那就浅蓝色的 Filter, 在第一个 Channel 上面滑过去, 滑过去滑过去, 然后就算出一个 Feature Map, 然后深蓝色的这个 Filter 管第二个 Channel, 那它就在第二个 Channel 上面做 Convolution, 然后也得到另外一个 Feature Map, 所以在 Depthwise Convolution 里面, 你 Input 有几个 Channel, 你 Output 的 Channel 的数目会是一模一样的。



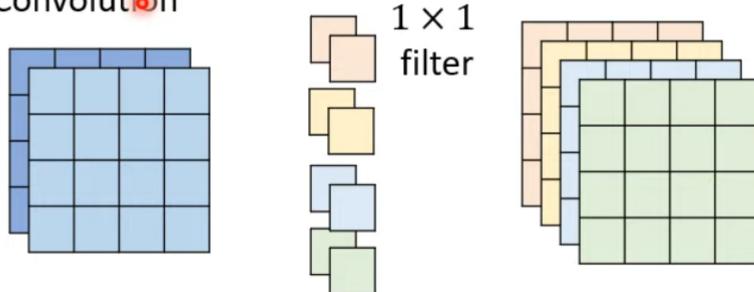
- Filter number = Input channel number
- Each filter only considers one channel.
- The filters are $k \times k$ matrices

这个跟一般的 Convolution Layer 不一样, 一般的 Convolution Layer 里面, Input 跟 Output 的 Channel 数目可以不一样, 但在 Depthwise Convolution 里面, Input 跟 Output 的 Channel 数目是一模一样的。

Pointwise Convolution

好 但是你会发现, 如果你只做 Depthwise Convolution, 它会遇到一个问题, 这个问题是 Channel 和 Channel 之间, 没有任何的互动。假设今天有某一个 Parton, 是跨 Channel 才能够看得出来的, 那 Depthwise Convolution 对这种, 跨 Channel 的 Parton 是无能为力的。所以怎么办呢, 再多加一个 Pointwise 的 Convolution。

2. Pointwise Convolution

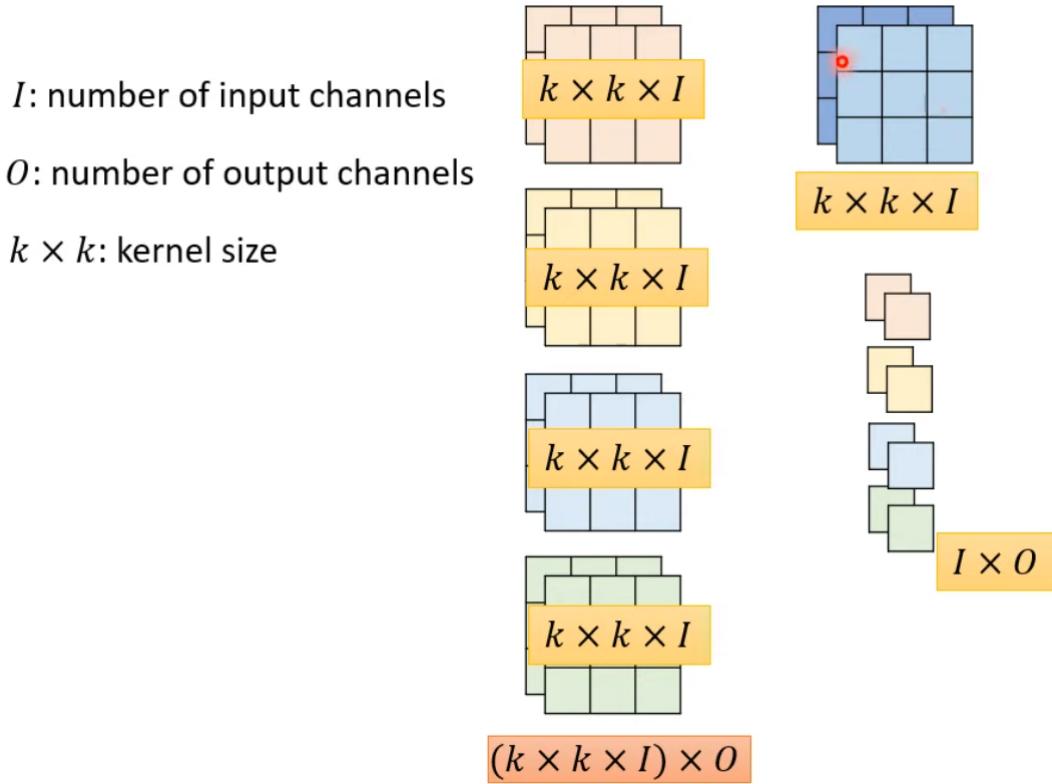


Pointwise Convolution 的意思是说呢，我们现在一样有一堆 Filter，这个跟一般的 Convolution Layer 是一样的。但我们这边做一个限制是，我们的 Filter 的大小，它的 Kernel 的 Size 通通都是 1×1 。在一般的 Convolution Layer 里面，你的 Filter 大小可能开 2×2 , 3×3 , 4×4 ，但是在 Pointwise Convolution 里面，我们限制大小一定是 1×1 ，那这个 1×1 的 Filter。它的目标它的作用，就是去考虑不同 Channel 之间的关系，所以第一个这个 1×1 的 Filter，它做的事情就是去扫过这个，Depthwise Convolution 出来的 Feature Map，然后得到另外一个 Feature Map。好那这边有另外三个 Filter，它们做的事情也是一样，每一个 Filter 会产生一个 Feature Map，所以 Pointwise Convolution，跟一般的 Convolution Layer 是一样的地方是，输入跟输出的 Channel 数目可以不一样。但是 Pointwise Convolution，有一个非常大的限制，是我们强制要求说，Filter 的大小只准是 1×1 ，你只要考虑 Channel 之间的关系就好了，你就不要考虑同一个 Channel 内部的关系了。同一个 Channel 内部的关系，已经 Depthwise Convolution 做完了，所以 Pointwise Convolution，只专注于考虑，Channel 跟 Channel 间的关系就好。

好那我们先来计算一下，这个方法的参数量。你看在 Depthwise Convolution 里面，两个 Filter，每一个 Filter 大小是 3×3 ，所以就 $3 \times 3 \times 2$ ，总共是 8 个参数，那在 Pointwise Convolution 里面，四个 Filter，每一个 Filter 的大小是 2，每个 Filter 只用了两个参数，所以总共是 8 个参数。

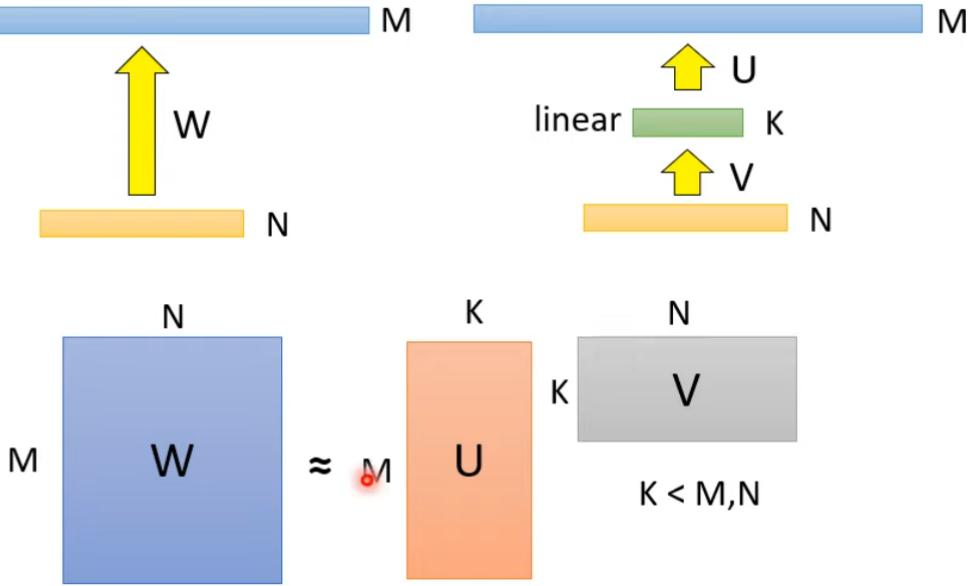
好左边这个图，这个是一般的 Convolution，右边这个图，是 Depthwise 加 Pointwise 的 Convolution，那我们现在来比较一下，这两者参数量的差异。我们现在假设我们先预设好说，Input Channel 数目就是 1 个 Channel，Output Channel 数目就是 O 个 Channel。然后呢 这一个 Case 跟这一个 Case，这边的 Kernel Size 都是 $k \times k$ 。

那如果是一般的 Convolution，你要有 $k \times k$ 的 Kernel Size，Input 1 个 Channel，Output O 个 Channel，那你到底会需要多少参数呢，那你能先算一下，每一个 Filter 的大小，每一个 Filter 的大小应该是 $k \times k$ ，Kernel Size 乘上 Input Channel 的数目，也就是 $k \times k \times 1$ ，如果你要 Output O 个 Channel，O 个 Channel，那你需要 O 个 Filter，所以总参数量是 $k \times k \times 1$ ，一个 Filter 的参数量再乘以 O。



如果是 Depthwise 加 Pointwise 的 Convolution，要达到 Input I 个 Channel, Outout O 个 Channel, 那要怎么做呢? Depthwise Convolution, 它的这个 Filter, 它的 Filter 是没有那个厚度的, 它的 Filter 是没有厚度的。所以你会发现说 Depthwise Convolution, 所有的 Filter 加起来, 它的参数量只有 $k \times k \times I$ 而已, 跟一般的 Convolution 里面的, 一个 Filter 的参数量是一样的。

然后我们看 Pointwise 的 Convolution, 这边是 $I \times O$, 就是假设你 Input Channel 的数目是 I 的话, 那每一个 1×1 的 Convolution, 它的高度会是 I 。那 O 是哪来的呢, 假设你要 Output O 个 Channel 的话, 那你就要有 O 个 1×1 的 Convolution, 所以今天 Pointwise Convolution, 它的总参数量是 $I \times O$ 。那你把这两者进行比较, 你把 $(k * k * I + I * O) / (k * k * I * O)$, 你把这两者相除, 经过一番计算你会发现, 这两者的比例是 $1/O + 1/k * k$, 那因为 O 通常是一个很大的值, 你的 Channel 数目你可能开个 256 啊 512 啊, 所以我们先把这个 1 除以 O 忽略不计。那 $k \times k$ 通常是比较, 在这一整项里面, 它可能是比较关键的数值, 这一整项的大小, 可能跟 1 除以 $k \times k$ 是比较有关系的。那假设你的 Kernel Size, 今天常用的 Kernel Size 可能是 3×3 , 或者是 2×2 , 假设你选 2×2 , 你把一般的 Convolution, 换成 Depthwise 加 Pointwise 组合的话, 那 Network 的大小就变成 $2 \times 1/2$, 假设你的 Kernel Size 是 3×3 , 当你从 Convolution 变成, Depthwise 加 Pointwise 的时候, 你的 Network 的大小就变成原来的 $1/9$ 了。

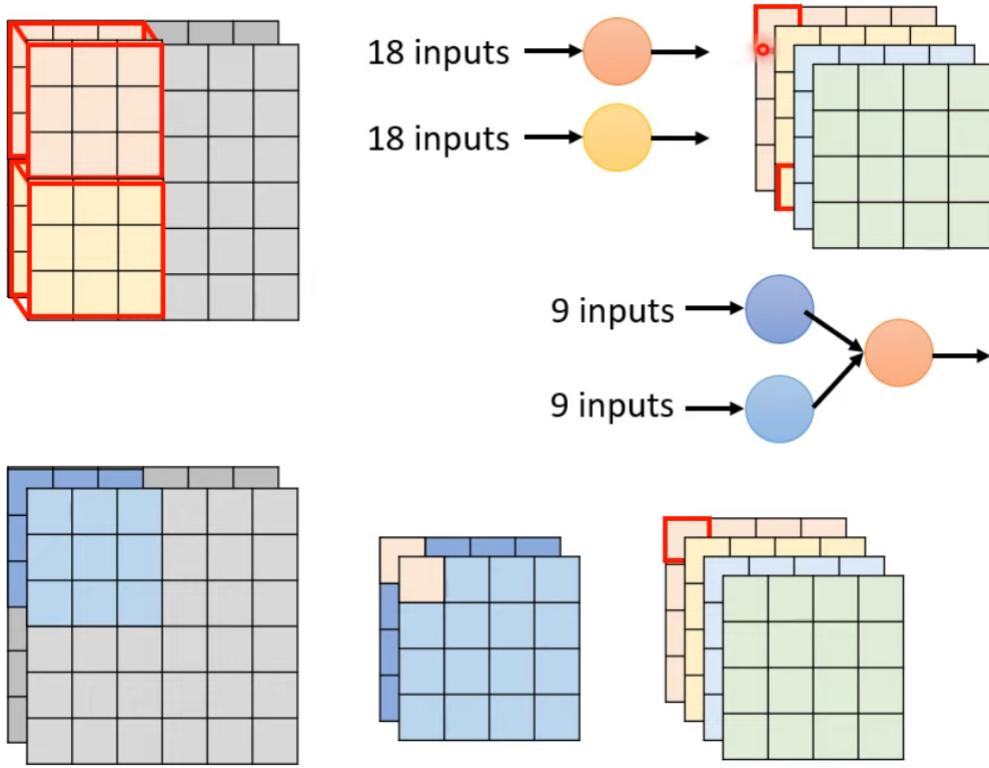


好那接下来想要跟大家解释的是，为什么这招有用呢，这一招的原理是哪来的呢？那这个啊，这个 Depthwise 加 Pointwise 这个招数，它其实是其来有自，在过去还没有这个，Depthwise Separable 的 Convolution 之前，就已经有一个方法，是用 Low Rank Approximation，来减少一层 Network 的参数量。那你不知道什么是 Low Rank Approximation，没关系，这边就直接告诉你，这个是怎么操作的，假设你有某一个 Layer，那这个 Layer 呢，输入有 N 个 Neuron，输出有 M 个 Neuron，那假设 N 或者是 M 其中一者非常大，这边假设是 M 非常大，那你的这个参数量就会非常地可观，这边的参数量 W 呢，假设你的 Input 是 N 个 Neuron，Output 是 M 个 Neuron，那你的参数量是多少呢，这两层之间的参数量是多少呢，是 $N \times M$ 对不对，那只要 N 跟 M 其中一者很大，那你这个 W 的参数就很多了

那怎么减少这个参数量呢？有一个非常简单的方法，是在 N 跟 M 中间再插一层，这一层就不用 Advection Function，直接多插一层。这一层 Neuron 的数目是 K ，原来本来只有一层，现在拆成两层，那这两层里面的第一层呢，我们用 V 来表示，第二层我们用 U 来表示。你可能会想说，这个把一层 Network 拆成两层 Network，这个参数量不是变多了吗，你仔细算算看，这个两层的 Network，参数量反而是比较少的。怎么说呢，你看原来一层的 Network，你的参数量是 $M \times N$ ，现在我们拆成两层 Network，第一层是多少，第一层是 $N \times K$ 对不对， $N \times K$ ，第二层是多少，第二层是 $K \times M$ ， $K \times M$ ，如果你今天的 K 远小于 M 跟 N ，那你就会发现说， U 跟 V 的参数量加起来，是比 W 还要少的多， W 是 $N \times M$ ，那 U 跟 V 的参数量加起来是 $K \times N + M$ ，就 N 跟 M 就没有相乘了，那你只要可以 K 够小，那整体而言 $U + V$ 的参数量就会变少。所以你常常，之前过去常有的做法就是， $N 1000 M 1000$ ，没关系我可以塞个 20 塞一个 50，那这个参数量就减少蛮多的。

那像这样子的方法，虽然可以减少参数量，当然它还是会有一些限制。什么样的限制，因为你会发现说，当你把 W 拆解成 $U \times V$ 的时候，当你把 W 分成，用 U 跟 V 两层来分开表示的时候，你会减少 W 的可能性，本来 W 它可以放任何的参数，但假设把 W 拆成 U 跟 V 的话，那这个 W 这个矩阵，它的 Rank 会小于等于 K 。那你不知道 Rank 是什么也没有关系，反正就是 W 会有一些限制，所以会变成说它不是所有的 W ，都可以变成用在，不是所有的 W 都可以变成当作你的参数，你的参数会变成有一些限制，好那这个方法呢，就是拿来减少参数的一个非常常用的做法。

Depthwise + Pointwise Convolution



那其实刚才讲的，Depthwise 加 Pointwise 的 Convolution，其实它用的就是我们这边的概念。就是把一层拆成两层这样的概念，好怎么说呢？我们先来看一下原来的 Convolution，在原来的 Convolution 里面，红色的这一个矩阵，左上角的这个参数是怎么来的，是不是有一个红色的这个 Filter，放在 Feature Input 的，Feature Map 的左上角以后所得到的。那今天在这个例子里面，一个 Filter，一个 Filter 的参数数量是多少，一个 Filter 的参数数量是 $3 \times 3 \times 2$ 也就是 18。你把 Filter 里面的 18 个参数，跟 Input Feature Map 左上角的这 18 个数值，做 Inner Product 以后，就会得到这边，Output Feature Map 左上角这个值，所以你的每一个 Filter 有 18 个参数，如果今天拆成，Depthwise 加 Pointwise 两阶的话，那会怎么样呢？如果拆成 Depthwise 加 Pointwise 两阶的话，左上角 Output Feature Map，左上角这个数值来自于中间的，Depthwise Convolution 的 Output，所以左上角这个值来自于中间这一个，Depthwise Convolution 的 Output，左上角的这两个值，而这两个值来自于 Input Feature Map，第一个 Channel 左上角这 9 个值，跟第二个 Channel 左上角这 9 个值。所以怎么从这边变到这边呢，你可以看成是，我们有两个 Filter，这两个 Filter 呢，它分别是 9 个 Input，然后得到输出。然后接下来这两个 Filter 的输出，再把它综合起来，得到最终的输出，所以本来是直接从这 18 个数值，变成一个数值，现在是分两阶，18 个数值变两个数值，再变一个数值。或者是我们今天看，黄色的这个 Feature Map，左下角这个参数，黄色的 Feature Map 左下角这个数值，来自于哪里呢？来自于 Depthwise Convolution 的 Output，左下角这两个数值，而左下角这两个数值来自于这个 Filter 左下，这来自于这个 Depthwise，来自于 Input 的这个 Feature Map，左下角的这 18 个数值。所以其实今天你又可以再看成说，是我们把这个，当我们把这个一般的 Convolution，拆成 Depthwise 加 Pointwise 的时候，我们就可以看成是把一层的 Network，拆解成两层的 Network。所以它的原理，跟刚才在前一页投影片看到的，Low Rank Approximation 是一样的。我们把一层拆成两层，这个时候它对于参数的需求反而是减少了，好那这个是有关 Network Architecture 的设计。

- SqueezeNet
 - <https://arxiv.org/abs/1602.07360>
- MobileNet
 - <https://arxiv.org/abs/1704.04861>
- ShuffleNet
 - <https://arxiv.org/abs/1707.01083>
- Xception
 - <https://arxiv.org/abs/1610.02357>
- GhostNet
 - <https://arxiv.org/abs/1911.11907>

那其实有关 Network Architecture 的设计，还有非常多论文可以参考啦，那这些文献，其实也都放在助教的投影片里面，那等一下助教不会细讲这些 Network 的架构啦。但是在这个作业里面，如果你可以成功实作出，刚才讲的这个，这个 Depthwise 加 Pointwise 相叠的话，把一层 CNN 拆成两层 CNN 的话，那你就很有可能可以破 Strong Best Line。

Dynamic Computation

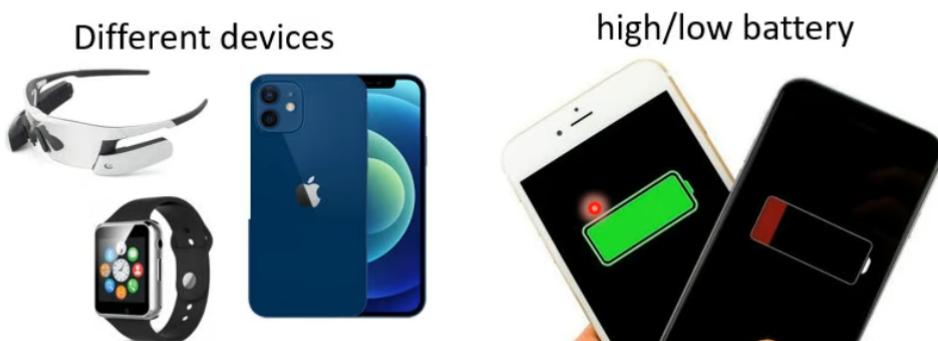
最后一个要跟大家分享的，是 Dynamic Computation，那 Dynamic Computation 要做的事情，跟前几个方法想要达成的目标不太一样。

Overall

在前几个方法里面想要做的事情，就是单纯的把 Network 变小，那在 Dynamic Computation 里面，要做的事情是什么呢，在 **Dynamic Computation 里面，要做的事情是，我们希望 Network 呢，可以自由地调整它需要的运算量。**

为什么我们期待 Network，可以自由地调整它需要的运算量呢，因为有时候你可能同样的模型，会想要跑在不同的 Device 上面，而不同的 Device 上面，它有的运算资源是不太一样的，所以你期待说，你训练好一个 Network 以后，你放到新的 Device 上面，你不需要再重 Train 这个 Network，因为这个训练一个神奇的模型，这个神奇的模型，本来就可以自由调整，它所需要的运算资源，所以运算资源少的时候，它就只需要少的运算资源就可以运算，运算资源大的时候，它就可以充分利用，充足的运算资源来进行运算。

- The network adjusts the computation it need.



那另外一个可能是，就算是在同一个 Device 上面，你也会需要不同的 Computation，比如说你有，举例来说假设你的手机非常有电的时候，那你可能就会有比较多的运算资源，假设你的手机没有电的时候，那你可能就需要把运算资源，留着做其他的事情，那 Network 可能可以分到的运算资源就比较少。所以就算是在同一个 Device 上面，我们也希望一个 Network，可以根据现有的运算资源，比如说手机

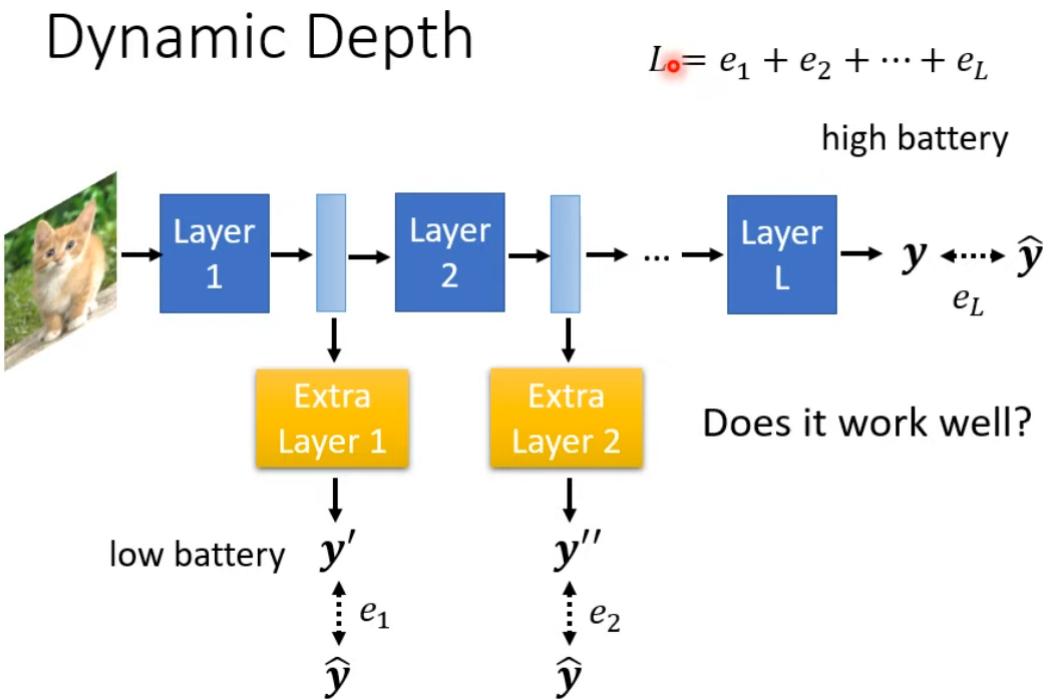
现在的电量还有多少，来自由地调整它对运算量的需求。

好 那有人可能会问说，为什么我们不直接准备，一大堆的 Network 就好了呢，假设我们今天需要各种，应付各种不同运算资源的情况，我们为什么不训练 10 个 Network，从运算量最少的到运算量最大的，然后根据我们的运算的状况，去选择不同的 Network 呢？可是你知道假设今天，假设你是在同一台手机上，你需要根据不同的状况做不同的因应，那你可能就需要训练一大堆的 Network，而手机上的储存空间有限，那你今天我们就是要减少我们的运算量，但是如果你需要训练一大堆的 Network，那你要一大堆的储存空间，那这个可能不是我们要的。

所以我们其实期待可以做到说，一个 Network 它可以自由地调整，它对运算资源的需求，那怎么让 Network 自由地调整，它对运算资源的需求呢？

Dyanamic Depth

一个可能的方向是，让 Network 自由地调整它的深度。怎么让 Network 自由调整它的深度呢，你可以训练，一般我们就是训练一个很深的 Network，然后它有 Input，比如说在做 Image Classification 的话，它就是输入一张图片，然后输出呢，就是图片分类的结果，然后呢，你可以在这个 Layer 和 Layer 中间，再加上一个额外的 Layer，这个额外的 Layer 它的工作呢，是根据每一个 Hidden Layer 的 Output，去决定现在分类的结果应该是什么。那这样子当你的运算资源比较充足的时候，你可以让这张图片去跑过所有的 Layer，得到最终的分类结果，当运算资源不充足的时候，你可以让 Network 决定说，它要在哪一个 Layer 自行做输出，比如说运算资源比较不充足的时候，通过第一个 Layer，就直接丢到这个 Extra 的 Layer 1，然后呢 就得到最终的结果了，那怎么训练这样一个 Network 呢，那其实概念比你想像的还要简单，我们训练的时候都有 Label 的资料，那一般在训练的时候，我们只需要在意最后一层 Network，它的 Output 是什么，我们希望它的 Output，跟 Ground Truth 越接近越好。但是我们今天呢，也可以让 Ground Truth 呢，跟每一个 Extra Layer 的 Output 越接近越好，那我们把所有的 Output，跟 Ground Truth 的距离通通加起来，把所有的 Output，跟 Ground Truth 的 Cross Entropy，通通加起来得到 L，然后再去 Minimize 这个 L，然后就结束了。



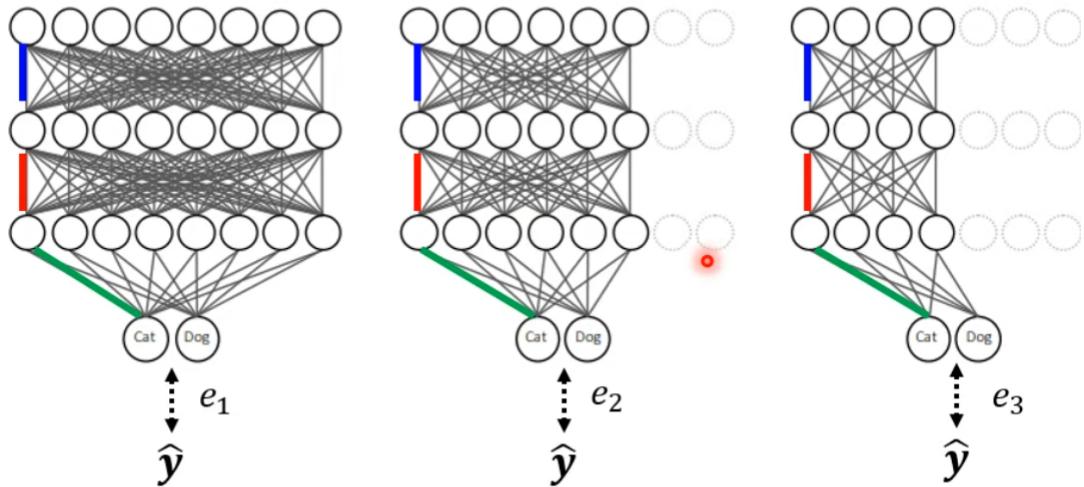
好 那这样子最基本的方法，它能够运作得好吗，这个训练方法是可以用的，你确实可以用我刚才讲的训练方法，就是每一层接出来做训练，然后把所有接出来的结果，去跟 Ground Truth 算距离，然后 Minimize 所有接出来的结果，跟 Ground Truth 的距离，确实可以用这个方法达到 Dynamic 的深度。但是其实它不是一个最好的方法，如果你想知道最好的方法是怎么做的，这个也不一定是现在最好的方法，假设你想知道比较好的方法是怎么做的，也许你可以参考 MSDNet 这篇文章。

Dynamic Width

另外还可以让 Network 自由地决定它的宽度，怎么让 Network 自由决定它的宽度呢，你就是有设定好几个不同的宽度，然后呢你今天同一张图片丢进去，在训练的时候，同一张图片丢进去，那每一个 Network 呢，每个不同宽度的 Network 会有不同的输出，我们在希望每一个输出，都跟正确答案越接近越好就结束了。我们把所有的输出跟 Ground Truth 的距离，加起来得到一个 Loss，然后我们要去 Minimize 这个 Loss 就结束了。那这边要跟大家强调一下，虽然我在这个投影片上，我画了三个 Network，但是这并不是三个不同的 Network，它们是同一个 Network，可以选择不同的宽度，也就是说这个 Weight，就是这个 Weight，我标一样颜色的，就是同一个 Weight。只是在最左边这个状况的时候，整个 Network，所有的 Neuron 都会被用到，但是在中间这个状况的时候，你可能会决定说，有 25% 的 Neuron 会用到，但你就会视哪些 Neuron 不用到，你就事先决定好，你就事先决定好说，某些 Neuron 在你选择说，只要用这个 75% 参数的时候，那 25% 的 Neuron 我们不要用它，或拿 50% 的 Neuron 我们不要用它，然后在训练的时候，就把所有的状况一起考虑，然后所有的状况都得到一个 Output，所有的 Output 都去跟 Ground Truth 计算距离，然后要让所有的距离都越小越好就结束了。

Dynamic Width

$$L = e_1 + e_2 + e_3$$



Slimmable Neural Networks
<https://arxiv.org/abs/1812.08928>

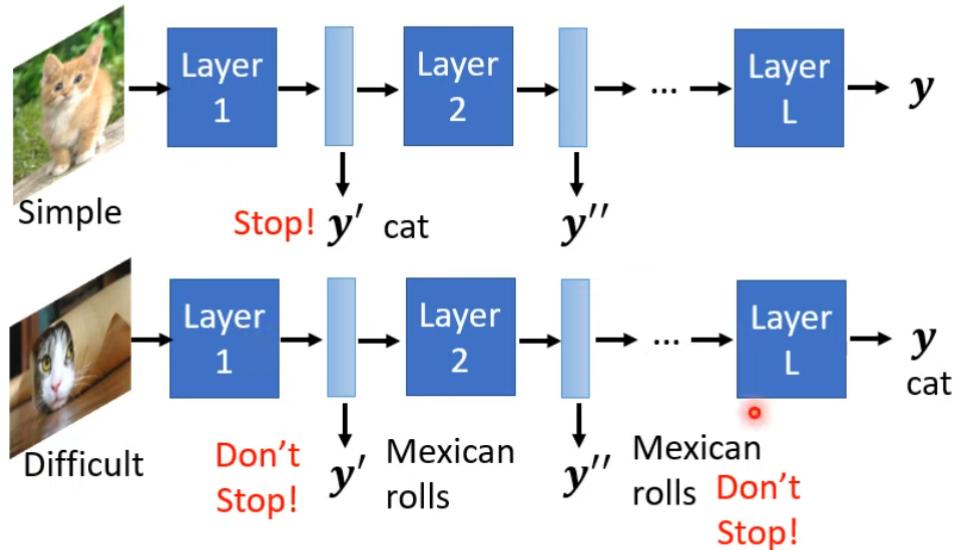
但是实际上你会发现，这样 Train 也是有问题的，所以需要一些特别的想法来解决这个问题，那有关 Depthwise Dynamic 的宽度的 Network，要怎么训练这件事，大家可以参考 Slimmable 的 Neuron Network。

Dynamic Model

好那刚才讲的，是我们可以 Train 一个 Network，我们可以自由去决定，它的深度跟它的宽度，但是所谓的决定权还是在人这一边，你要自己去决定说，今天电池电量少于多少的时候，我们就用多少层，或者是多宽的 Network，但是有没有办法让 Network 自行决定，根据它的环境，根据我们的情境，决定它的宽度或者是深度呢，这个也是有办法的。

为什么我们需要 Network，自己去决定它的宽度跟深度呢，那是因为有时候，就算是同样是影像分类的问题，那有一些影像可能特别难，有一些影像可能特别简单，对那些比较简单的影像，也许你只要通过一层 Layer，Network 就已经可以知道答案了，对于一些比较难的问题，举例来说同样是猫，这只猫是被做成一个墨西哥卷饼的样子，所以这是一个特别困难的问题。也许这张图片只通过一个 Layer 的时候，Network 会觉得它是一个卷饼，在通过第二个 Layer 的时候，还是一个卷饼，要通过很多个 Layer 的时候，Network 才能够判断它是一只猫，如果是这种比较难的问题，你就不应该在中间停下来，那我们能不能让 Network 自己决定说，这是一张简单的图片，所以通过第一层就停下来，这是一个比较困难的图片，所以要跑到最后一层才停下来呢，我们是可以这么做的。那假设你想要知道怎么做的话，可以参考以下几篇 Reference，所以像这样子的方法，其实就不一定限制在那个，运算资源比较有限的状况啦。

Computation based on Sample Difficulty



- SkipNet: Learning Dynamic Routing in Convolutional Networks
- Runtime Neural Pruning
- BlockDrop: Dynamic Inference Paths in Residual Networks

有时候就算是你运算资源比较很充足，但是对一些简单的图片，如果你可以用比较少的 Layer，就可以得到你要的结果，那其实也就够了，这样你就可以省下一些运算资源，去做其他的事情，就好像说我可以了解说呢，在这门课里面，因为我们今天把这个，做什么事情会得到几分都订得非常地明确，所以大家就会知道说你做了哪些事情，大概就可以拿到 A+，然后所以后面的作业，你拿到 A+ 以后，可能后面的作业你就不想做了。

所以你就跟上面这个情况一样，做到中间你就停下来，得到 A+ 以后，你就 Output 你的结果就结束了，那这也是人之常情，我也是可以接受的啦，不过像以前，以前这个，这学期呢 这个成绩是，这个原始成绩就直接定义，就直接对应到那个等第了，那过去其实有一阵子，这个原始成绩是没有直接对应到等第，成绩是相对的，这个时候大家就会很痛苦吧，有的同学就会，他原始成绩拿到 100 多分，哇 结果 C- 这样子。

Concluding Remarks

Concluding Remarks

- Network Pruning
- Knowledge Distillation
- Parameter Quantization
- Architecture Design
- Dynamic Computation

以前如果是直接按照比例分配，就前 1/4 的人 A+，以此类推的话，那就有可能有点痛苦吧，不过这学期，我们就直接把原始成绩对应到等第，让大家日子过得比较轻松一点。好那以上呢，就是跟大家介绍的五个技术，那前面四个技术，都是让 Network 可以变小，那这四个技术呢，它们并不是互斥的，其实你今天要在做 Network 压缩的时候，你其实没有什么道理只做一个技术，你可以既用 Network 的 Architecture，也做 Knowledge Distillation，你还可以在做完 Knowledge Distillation 以后，再去做 Network Pruning，你还可以在做完 Network Pruning 以后，再去做 Parameter 的 Quantization，如果你今天真的想要，把 Network 压缩到很小的话，这些方法并不是互斥的，它们都是可以一起被使用的，好那以上呢，就是有关 Network Compression 的介绍，那我们其实在这边呢，到这边 Network Compression 呢，就讲到一个段落，那来看看大家有没有问题要问的，