Lecture 13：Compression：神经网络压缩

> Lectured by HUNG-YI LEE (李宏毅)
>
> Recorded by Yusheng zhao（yszhao0717@gmail.com）

> 我们见过像BERT、GPT-3这样的庞大的模型，那我们能不能简化这些模型，让他们有较少的参数但是跟原来的模型效能差不多。——Network Compression

应用场景：将ML模型应用在资源比较有限的情况下譬如物联网设备或是小型电器上（例如智能手表、无人机等等），这就要求ML model不能太大，因此模型压缩对算法在工业应用落地有其必要性。

这类应用场景往往需要**低延迟（low latency）**、以及**隐私（privacy）保护**的需求。

因此无法应用云计算（输入通过无线网络传入云端，在云端计算，输出传回来）——其一原因是云计算带来信息传递的延迟（Latency），一些小型物联网设备应用场景需要尽量快速的实时反应。

尽管5G时代或许会解决信息传递的延迟，但是将用户隐私数据储存在云端进行运算建模，也有着对隐私侵犯的隐患。

> 针对隐私在云端的保护，涉及到云端上建模使用联邦学习、或在隐私数据上进行同态加密。此处不是本lecture讨论重点。

因此，Network Compression有其应用价值和潜力。

> 以下介绍五种神经网络压缩的技术（全是软件层面），本lecture不考虑硬件层面的运算加速或压缩方案

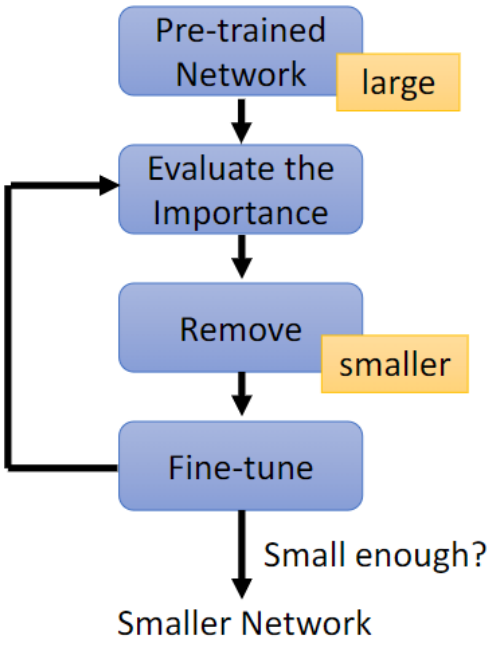
Network Pruning：网络剪枝

所谓树大必有末枝，多数的大型神经网络通常都是over-parameterized，这些没有用的"摸鱼参数"（redundant weights or neurons）可以被剪掉，而不影响模型整体的性能。

90年代，Yann LeCun发表的一篇"Optimal Brain Damage"探索了如何optimize脑的神经元而使脑的损伤最少，和Network Pruning有异曲同工之妙。

Network Pruning的流程

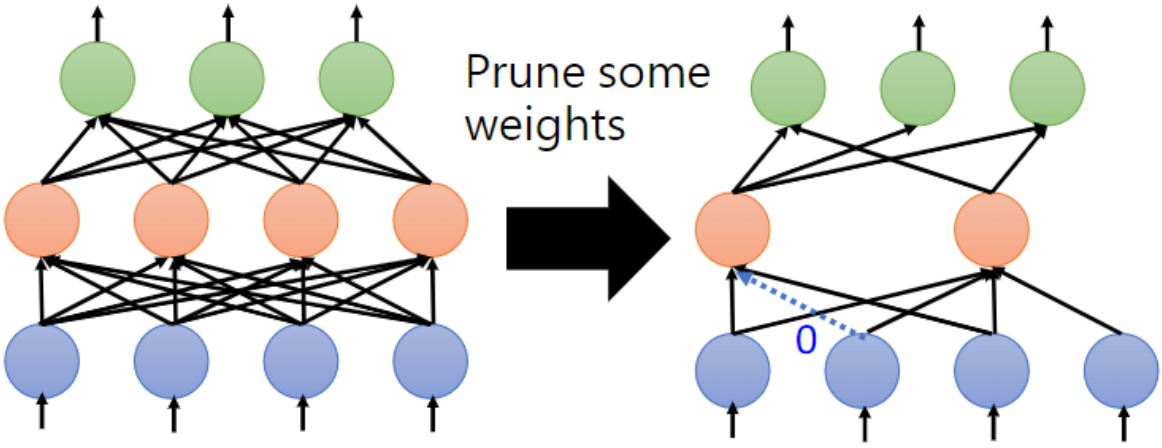
- 首先训练一个初始的神经网络（Large Pre-trained Network），通常这个模型的规模很大。
- 评估这个大的network里边的参数，找找里边哪些参数是在做事的（attach of importance），哪些参数是在摸鱼的。最简单的看从参数绝对值，如果绝对值越大说明参数对network影响越大。对于**参数**和**神经元**的重要性的评估参考如下所示：
  - *Importance of a weight*：绝对值（如果接近于0说明对network影响不大）、套用lifelong learning的想法：计算每个参数的BI值（参数更新权重，表示参数有多重要，能不能变化太多）
  - *Importance of a neuron*：计算在数据集上训练过程中神经元输出不为0的次数
- 把不重要的参数或神经元从模型中移除，这样就得到较小规模的神经网络
- 剪枝（pruning）过后，通常模型的准确率（accuracy）会掉一点；把没剪掉的参数在训练资料上进行微调（fine-tuned）可以恢复模型性能
- 重复第二步到第四步，不断剪枝，直到network够小且性能大差不差。tips：小剪多次，不要一次性剪太多，否则神经网络可能recover不回来。



Practical Issue：用参数（weight）或神经元（neuron）当作剪枝单位的差别

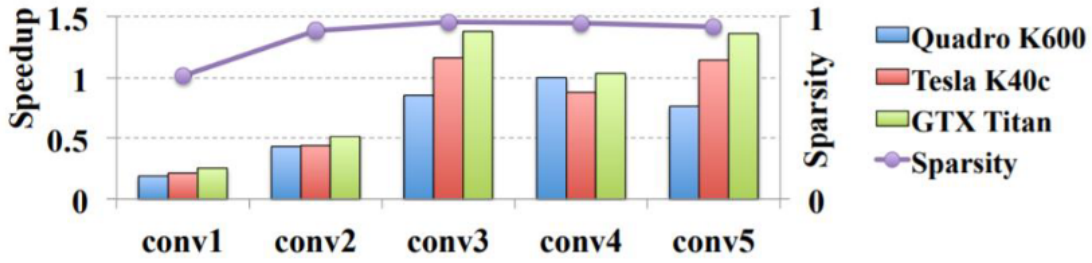
Weight pruning

当我们去掉某一个/些不重要的参数后，模型形状会变得不规则（如下图右）



形状"不规则"的模型是**不好实做的**（coding很不方便...）而且**不利于用GPU做加速运算。**如果非要做，只能把去掉的weight补为0，这又违反了我们做剪枝的初衷，存储中依然存在weight（即便是0），模型根本没有变小...remark：**hard to implement and speedup**

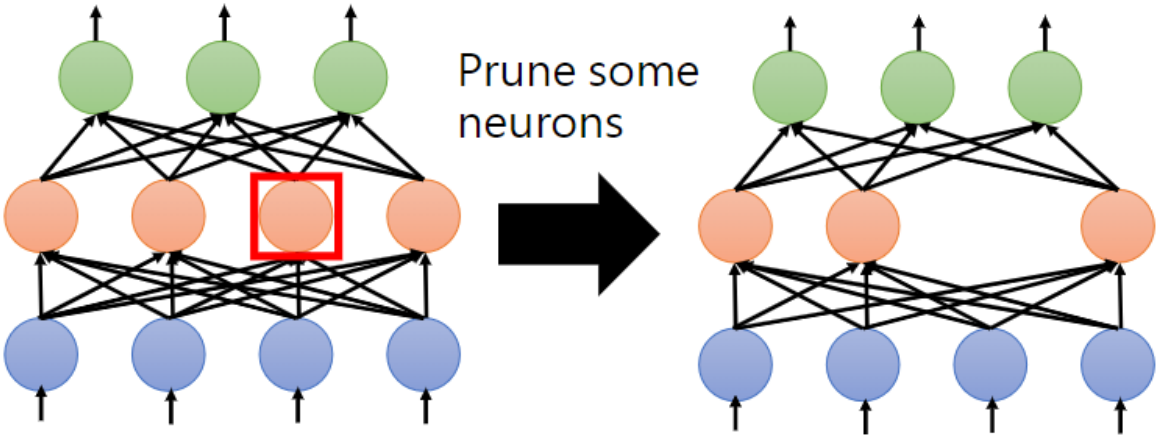
来自https://arxiv.org/pdf/1608.03665.pdf的实验；验证了即便剪枝掉九成多的weight，但是speedup的效果依然很差。



> 如果自己写个库用于weight pruning的实现和加速.....

Neuron pruning

当我们去掉某一个/些不重要的神经元后，模型形状依然保持规则（regular），如下图右



这时候用Pytorch比较方便实现，改改输入输出的dimension就行；也比较利于GPU加速。**Easy to implement and speedup !**

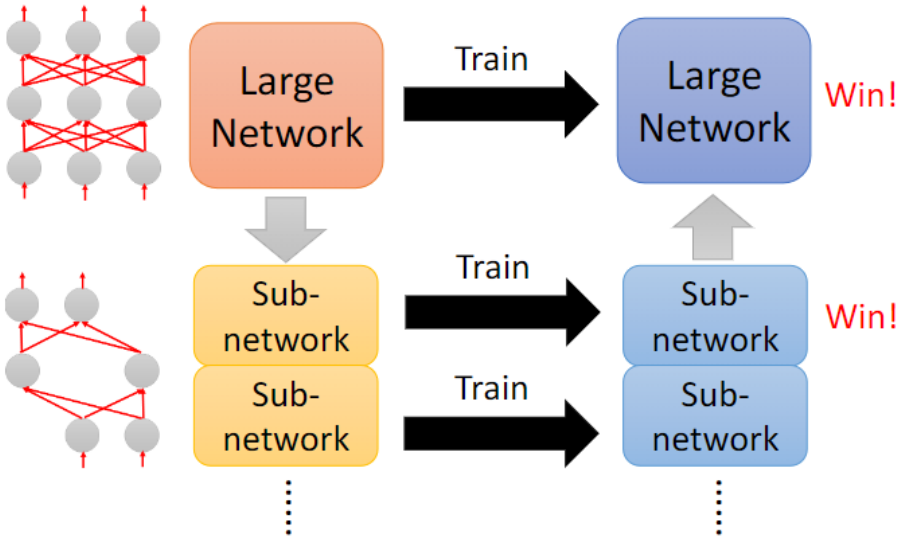
Why Pruning ?

Q_1：为什么不直接train一个小的network?

Ans：大的network比较好train（实践验证，众所周知）；然后再把大模型磨小。如果直接train小的network难以达到大模型再pruning后得到的小模型的性能

- 为什么大模型比较好train呢?

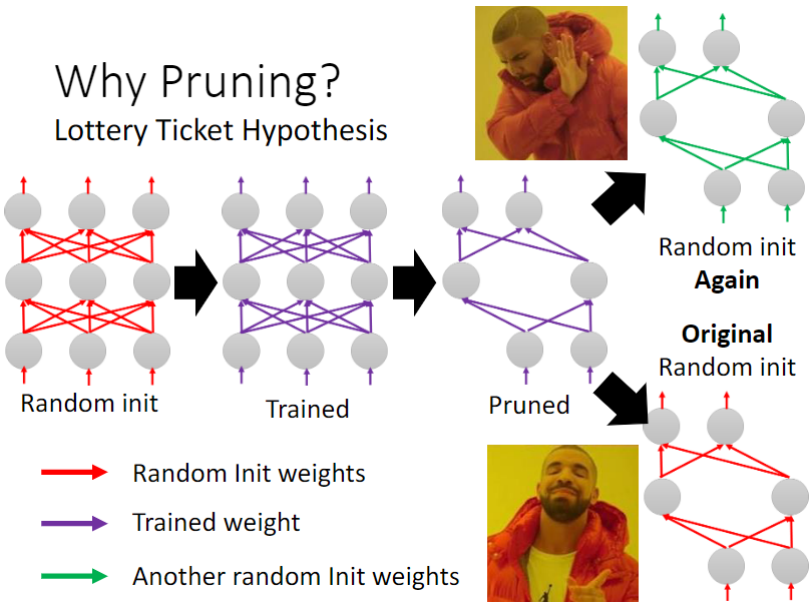
  乐透彩票假说（Lottery Ticket Hypothesis）https://arxiv.org/abs/1803.03635

  正如炼丹很玄学一样（看人品）.....要想中彩票（获得很合适的一组参数），就要大量的买彩票；一击即中很南的啦。

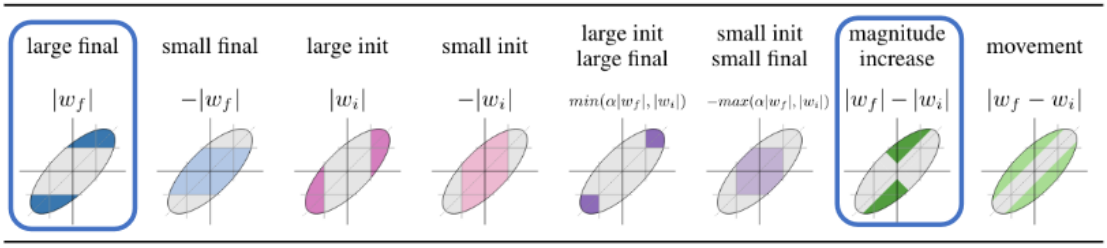
  我们可以把大的network分为各个sub-network，每个sub-network都有一定概率成功（获得很合适的一组参数）或失败；对于large network，只要其中之一子网络训练很成功，就能达到"一人得道，鸡犬升天"的效果——大的network就成功了（win！）。

  - *实验*：验证乐透假说（和network pruning关系密切）
    - 先对large network做参数的随机初始化；然后train后得到一组训练完的参数；再做network pruning得到一个较小的network
    - 再把较小的network的参数随机初始化，再train发现train不起来；但是，如果把一开始大模型的随机化的参数对应的赋给小模型，再train，能train起来。



尝试解构乐透假说的文章[Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask]——尝试了不同的pruning的策略，发现说某两个策略是最有效的，如果train前后绝对值差距越大，pruning掉的network结果是最有效的。

- Different pruning strategy



其他一些有趣的结论：

- 初始化参数正负号符号很重要（critical）。

  一组好的初始化参数到底是好在哪里呢？实验发现小的network如果不改变original random init的参数的正负号也是能train的起来的。这说明参数的绝对值不重要，参数的正负才重要。

- Pruning weights from a network with random weights

  一个初始随机化参数的大模型已经有一个sub-network，它的初始参数刚好随机化的很合适，可以直接拿来用在任务中，在实验中的分类任务中得到和监督学习很接近的正确率。（此处老师打了一个恰当的比方：米开朗琪罗雕刻大卫，不是刻意的雕刻而是把石头中的大卫形象取了出来）

  类似的发现来自文章[Weight Agnostic Neural Networks]，文章中网络随机的参数要不是随机的要不就统统设为1.5，这个network可以得到一定的好的performance。

- **[Rethinking the value of Network Pruning]**

  乐透假说同时期的工作，对于pruning的一种观点，有时间精读下这篇论文。

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-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

两个数据集，四个模型，分别设unpruned和pruned，做一下fine-tuned，得到指标。针对pruned model：👇

第一次实验是scratch-E（对于小模型初始参数是**真的**随机初始化，彼时乐透假说中小模型重新train前初始化参数来源于pruned前的大模型），发现指标确实略逊一些

第二次实验在第一次实验的基础，pruned model训练时的epoch多加几个，结果就比fine-tuned好。这篇文章和乐透假说的想法是相反的，里面也提出了乐透假说的可能隐含的成立条件：当learning rate设的较小、还有unstructured（以weight作为单位来pruning）的时候好像才会发现乐透假说的现象。

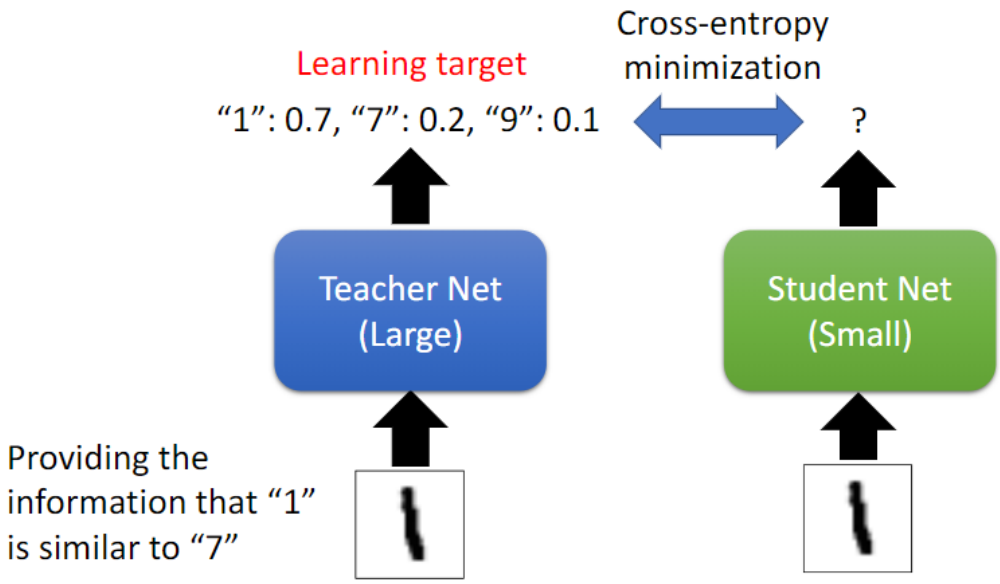
乐透假说是真是假，需要更多的研究来证实。

Knowledge Distillation：知识蒸馏

> 知识蒸馏的思想和pruning有一些类似的地方
>
> 参考文献：Knowledge Distillation、Do Deep Nets Really Need to be Deep?提出了知识蒸馏或类似的想法

我们先train一个大的network：称之为**Teacher Net**（Large）；我们最终要得到的小的network称之为**Student Net**。

和pruning不一样的地方是，pruning中小网络是大网络（修建后）的衍生品；而knowledge distillation中Student net是去根据teacher net来学习。



teacher net按照一般神经网络来训练（如图以mnist做手写数字分类器）；而student net以老师的embedding（上图中是[0, 0.7,..., 0.2, 0, 0.1]，十维的）作为学习目标而非分类正确率；学生的目标就是尽量逼近老师的输出（就算老师是错的也直接学），optimization的方法为cross-entropy minimization。

目前来看知识蒸馏的好处：对于student net来说，teacher net不仅尽量提供了正确的分类答案，而且还提供了额外的信息（相比直接告诉网络正确分类的标记）：如上图，“1”、“7”和“9”长的有点像。

在这种情况下，光是凭着teacher net的教学哪些数字有怎样的关系，student net可能甚至可以在测试中识别出训练集中缺乏的数字。

- Teacher network不仅可以是一个单一的神经网络，它也可以是集成（Ensemble）的神经网络（N Networks）

  所谓Ensemble Learning：（简略介绍下）就是训练多个模型，然后做一个平均（Average many models）；将平均的结果作为最后的答案。实用上讲，集成学习时间开销比较大，在工业上不好使用，经常使用在机器学习比赛上。

我们把Ensemble平均输出作为teacher net的结果，然后用student net训练逼近这个结果（同样用到cross-entropy minimization）

- 一个小trick：

  Temperature for softmax

  初始的softmax如下：$y_i' = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$，$\Rightarrow$ 加了"温度"的softmax：$y_i' = \frac{\exp(y_i/T)}{\sum_j \exp(y_j/T)}$

$$y_1 = 100 \quad y_1' = 1 \qquad\qquad y_1/T = 1 \quad y_1' = 0.56$$
$$y_2 = 10 \quad y_2' \approx 0 \qquad\qquad y_2/T = 0.1 \quad y_2' = 0.23$$
$$y_3 = 1 \quad y_3' \approx 0 \qquad\qquad y_3/T = 0.01 \quad y_3' = 0.21$$

  假设T大于1，可以使其分布变得比较集中、平滑；这导致的好处可以给予student net额外的信息（某和某长得像），而分类结果保持不变。

- 另一个小trick：在知识蒸馏中对于学生网络对于老师网络的映射中可以多做一些限制，一般结果会好一些。譬如说：对于12层的老师网络，有人对每一层的输出对于对应的学生网络做一次学习.

  如果学生网络和老师网络差太多了的话，可以插入一个介于中介作用的network，让学生去学这个network。

  Temperature T也是一个超参数。

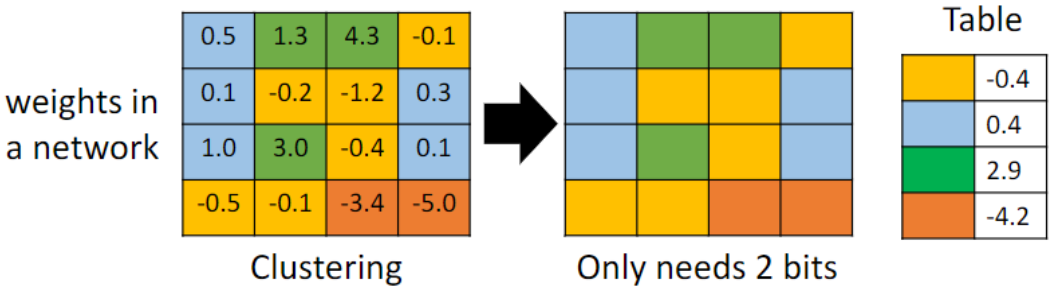
  Ensemble亦可以对参数做平均（上面老师是说对输出做平均）

Parameter Quantization：量化参数

- 使用比较少的空间来储存一个参数（Using less bits to represent a value）

  譬如说：损失部分精度的情况下，将16bit的参数压缩到8bit，甚至更少。而performance不会掉很多

- 参数的压缩：Weight clustering



  对参数组成的网络做一个聚类（clustering），数值接近的放在一起；事先设定好聚类数量（如上图是4类）；对于聚类的结果往往以类内参数取平均的方式。最后，只需要记录两样东西：一个是存储聚类结果的表（维度等于聚类数量），另一个是记录每个参数被分到哪个类。

  为了便于训练，会要求network之间的参数比较接近，从而可以方便聚类。

- Represent frequent clusters by less bits，represent rare clusters by more bits，及比较常出现的用比较少bit来编码，比较罕见的用较多的bit来编码。

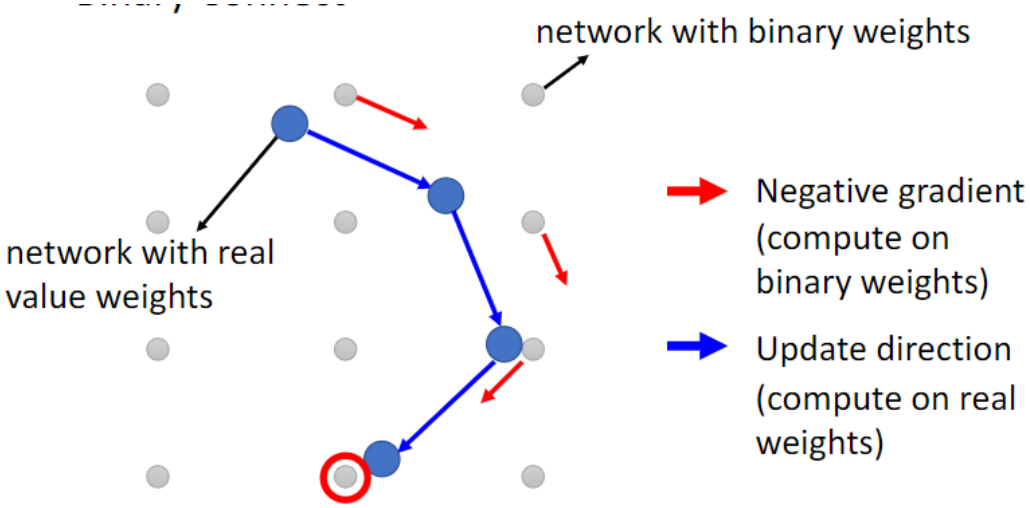
  举例：*霍夫曼编码（Huffman encodinig）*

Binary Weights

比较极端的情况：只用一个bit来存储参数。你的参数要不是+1要不就是-1。

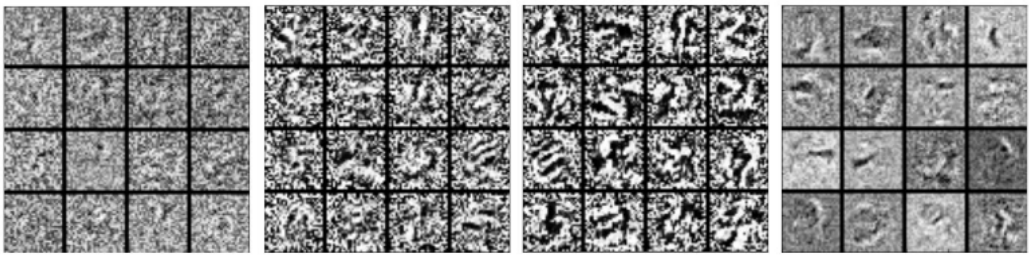
参考文献：Binary Connect、Binary Network、XNOR-net

以binary connect为例：



结果也不错：

Method	MNIST	CIFAR-10	SVHN
No regularizer	1.30 ± 0.04%	10.64%	2.44%
BinaryConnect (det.)	1.29 ± 0.08%	9.90%	2.30%
BinaryConnect (stoch.)	1.18 ± 0.04%	**8.27%**	2.15%
50% Dropout	1.01 ± 0.04%		

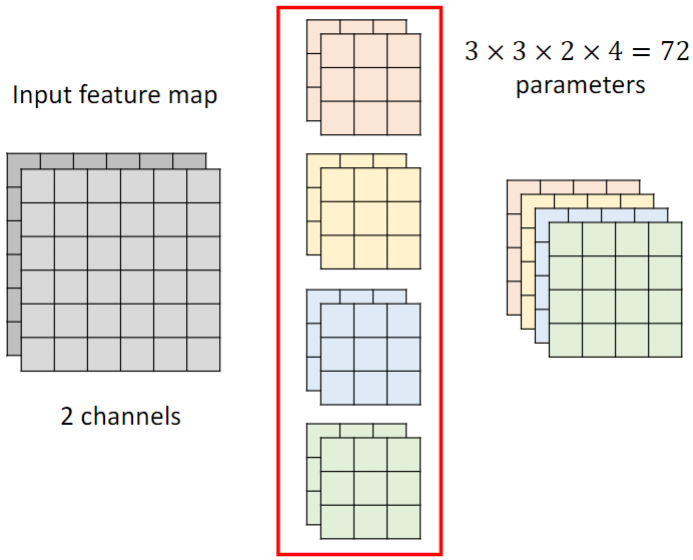


第一行是一般的网络，第二三行用binary connect结果更好了。binary network对网络做了一些更多的限制，从而很好的减弱了过拟合。

Architecture Design：网络的架构设计

Review：standard CNN

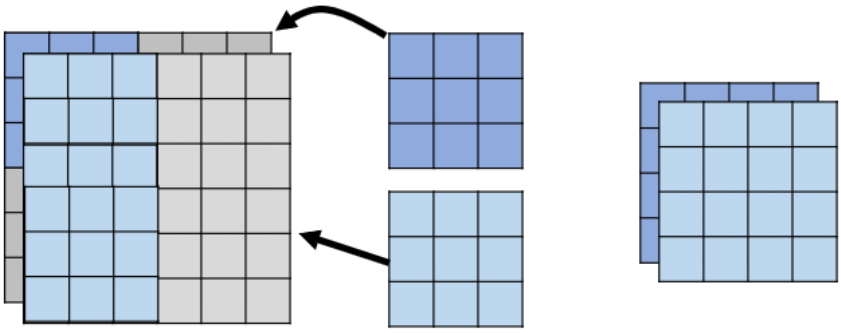
每个layer的input都是一个feature map（以下图为例：有2个channel）。输入feature map的channel的数量等于filter（立方体形状）的高度。用这个filter扫过feature map会得到一个输出feature map，有几个filter那么output的feture map就有几个channel。（总共参数量是72个）



Depthwise Separable Convolution

操作

- 步骤一：**Depthwise Convolution**（和CNN不同：该方法下输入和输出channel数量是一样的）

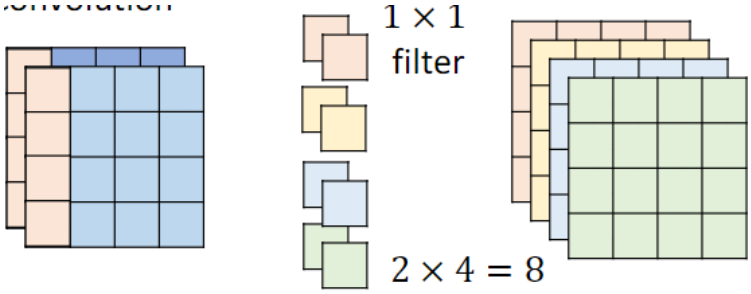


Filter number = Input channel number and Each filter only considers one channel.（有几个channel就有几个filter，每个filter只负责其对应的channel。）

filter是一个$k \times k$的矩阵，每个filter只在一个channel上滑来滑去（做convolution），得到对应的feature map。

channel和channel之间没有任何互动。（局限性）Depthwise Convolution对于跨channel的pattern是无能为力的。于是👇

- 步骤二：**Pointwise Convolution**



类似于一般的卷积层，对于含有2 channel的输入，有不止一个的fiiter，但是每一个filter的kernel size限制为1×1。每个filter分别扫过输入做convolution输出得到对应的feature map。

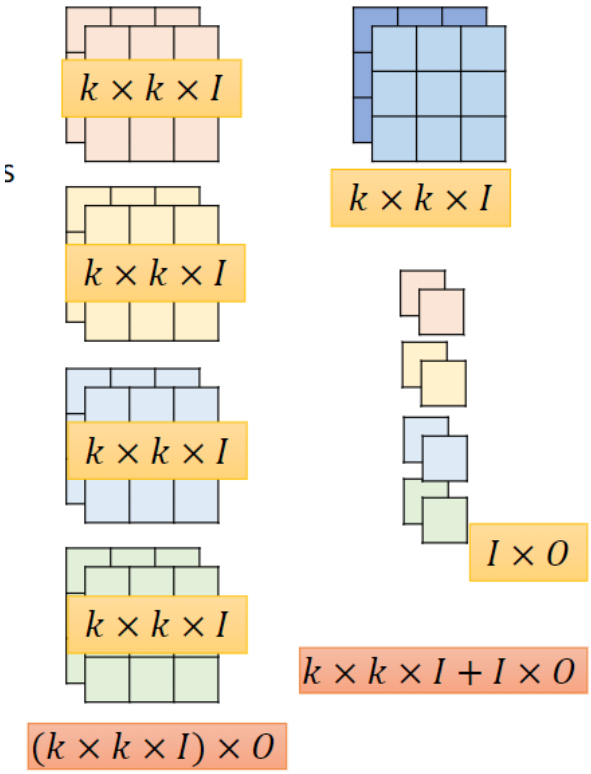
以上两步骤： 步骤一考虑到channel内的关系；步骤二考虑到channel间的关系。分别需要$3 \times 3 \times 2 = 18$个参数以及$2 \times 4 = 8$个参数，总共需要26个参数（相比一般的卷积层少了很多）。

同一般的卷积层比较参数量

I:输入feature map的channel数量

O :输出feature map的channel数量

$k \times k$:filter尺寸



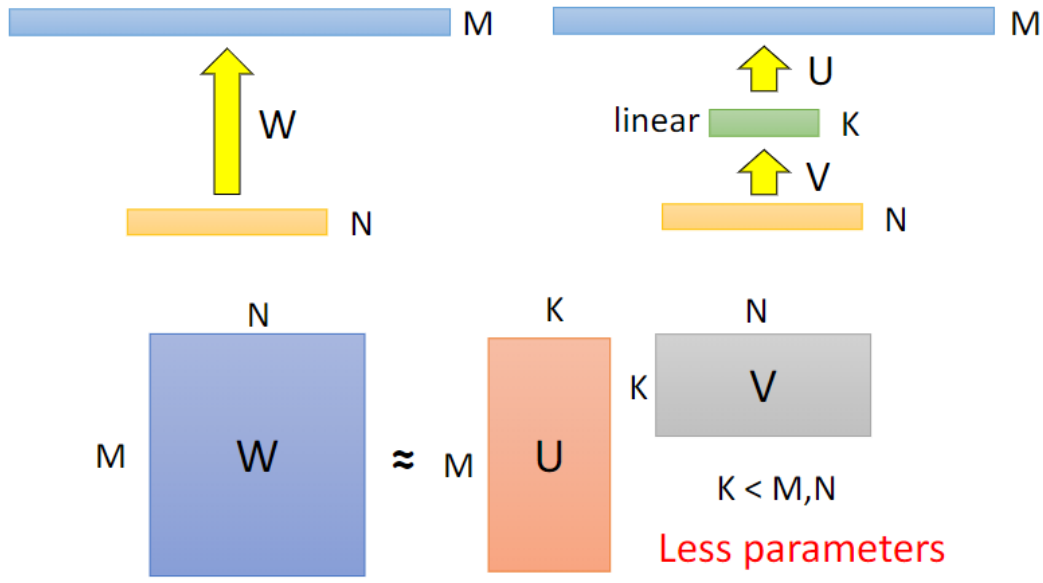
简单计算

$$\frac{k \times k \times I + I \times O}{k \times k \times I \times O} = \frac{1}{O} + \frac{1}{k \times k} < 1 \tag{1}$$

原理

Low rank approximation——来减少一层network的参数量

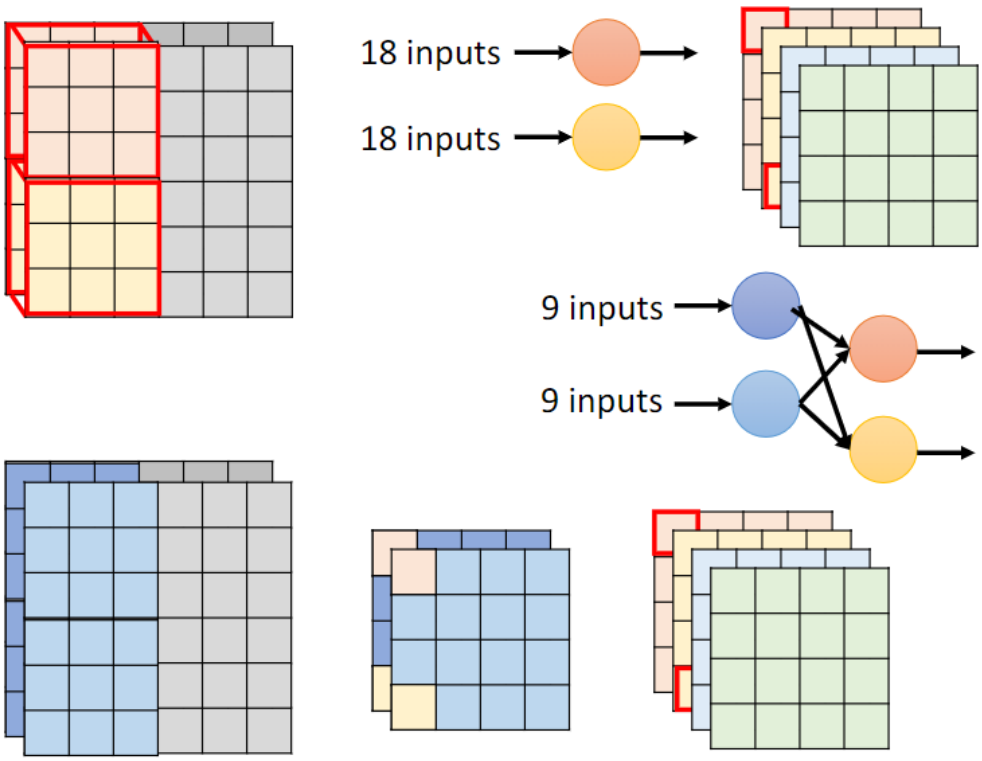
在Depthwise Convolution和Pointwise Convolution出现之前减少network参数的方法。



解释上图：对于原来的一层网络W，分别有N个神经元，有M个神经元，那么总共参数量为$M \times N$；Low rank approximation方法设计一个k个神经元的linear层放在网络W之间（拆成两层U, V），此时参数量为$M \times k + k \times N$。

如果$k << M, N$，那么$M \times N >> k \times (M + N)$,参数量大大减少了。局限性在于对于原网络$W$有一定限制，$W$的秩要小于$k$，这时才能拆成两个网络（矩阵）。

Depthwise Separable Convolution的原理



原来的卷积层，18个数值直接经过卷积层得到一个数值（结果）；而Depthwise Convolution则是把这个过程拆成两个阶段，或者说把一个网络拆成两个网络。

More Architecture Design

- SqueezeNet
- MobileNet，我好像经常看到这个...
- ShuffleNet
- Xception
- GhostNet

Dynamic Computation：动态计算

> 相比前面四个技术不同的目标：希望network自由地调整它的运算量

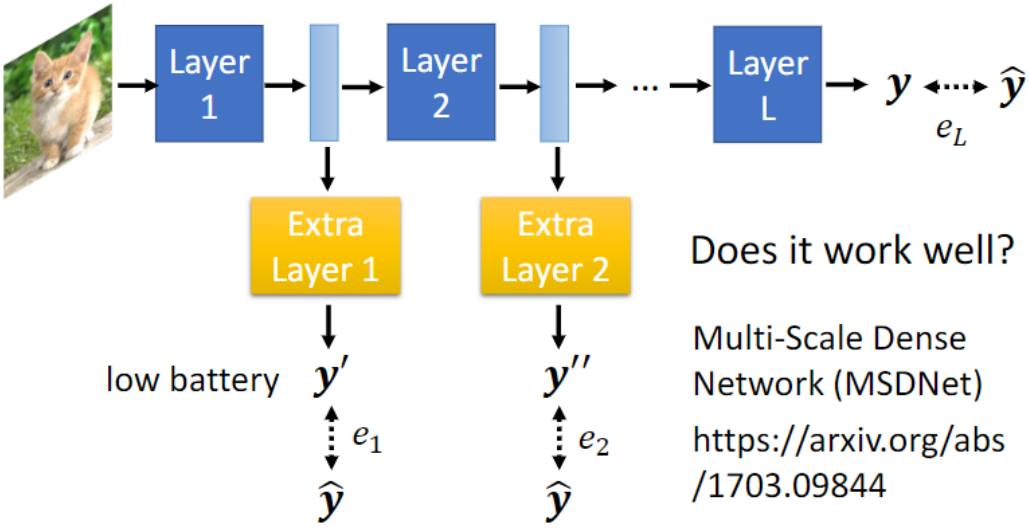
有时候，同样的模型可能会跑在不同的devices上面，device之间有着尺寸和运算资源上的差异；对于同一个device，也需要根据设备状态来调整运算量的大小（譬如手机电量）。所以我们需要network可以做到适应设备情况来自动调整运算量。

如果我们通过训练一大堆的network，或许能解决运算资源适应的问题，但是其会占用大量的存储空间。

Dynamic Depth

一个可能的方向是，通过让network自由的调整它的深度，从而让其调整运算资源的需求。

以做图像分类为例：训练一个很深的network



在layer和layer之间再加上一个extra layer，其作用：根据前一个隐藏层（hidden layer）的输出来决定现在分类的结果应该是怎样的。

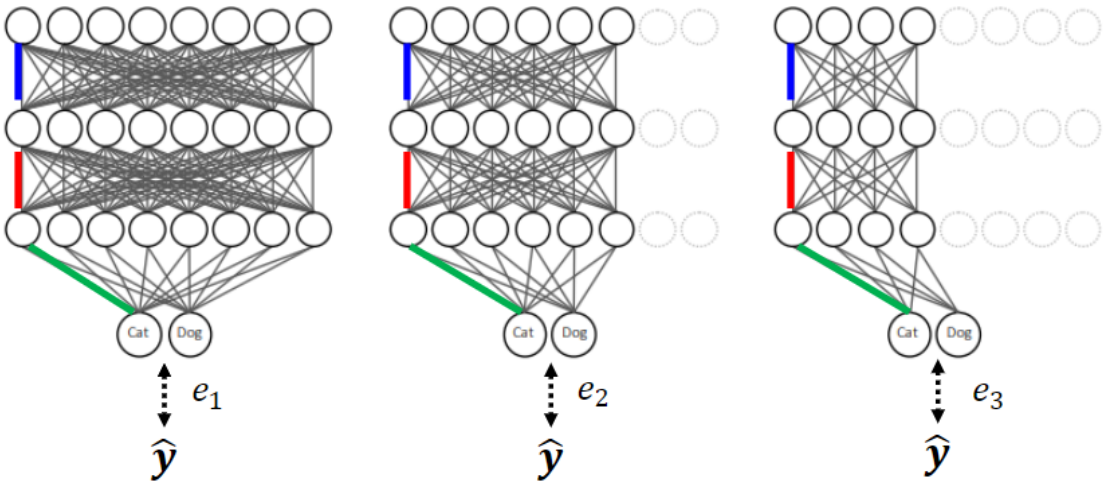
当运算资源非常充足的时候，可以跑通整个network，得到最终的分类结果；而如果运算资源比较局限，那么决定再network的哪一个extra layer自行输出，作为结果。

训练的过程：让ground truth（$\hat{y}$）跟每一层extra layer以及最后的output的距离e_i统统加起来，得到loss的计算公式：$L = e_1 + e_2 + \ldots + e_L$，然后minimize这个loss函数。目前来看，这并不是最好的方法。目前比较优异的训练方法参考Multi-Scale Dense Network (MSDNet)。

Dynamic Width

> 让network自由决定它的宽度

同一张图片丢进不同宽度的network，每一个network会有不同的输出，目标是所有的输出都和ground truth越接近越好。loss函数：$L = e_1 + e_2 + e_3$

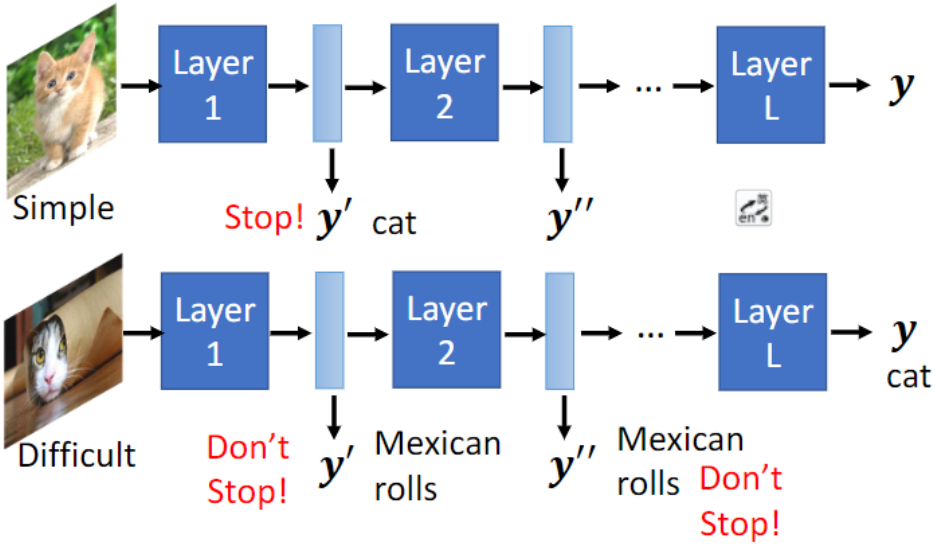


以上图：同一个network但是宽度不同（neuron的使用率100%、75%、50%）。

以上至少简略的想法，实践这样简单的做法是有问题的。关于dynamic width可以参考Slimmable Neural Networks。

Computation based on Sample Difficulty：让network根据情境自行决定自身的深度或宽度

不同图像存在不同的识别难度👇



简单的图片，第一层就停下来；困难的图片可能需要跑好多层。

这个问题相关的文章：

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

Concluding Remarks

前四个技术不是互斥的，可以在network中一起被使用。