ICLR 最佳论文“彩票假设”：如何通过彩票假设构建轻量化模型 (上)

2021-06-06 | 编辑：极市平台 | 作者：Sophie | 浏览：6039

作者丨科技猛兽
编辑丨极市平台

本文目录

Lottery Ticket Hypothesis，认为较复杂的深度神经网络存在一个比较优化的稀疏子网络结构 (称之为**winning tickets**)，可应用于模型压缩。相比于原网络，稀疏子网络的参数量与复杂度要低许多，但推理精度基本相当。今天这篇文章将会详细介绍彩票假设的原理及其泛化性能。

1 10分钟解读彩票假设的原理 (ICLR 2019)

> 论文名称：**The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks**

论文地址：

如果现在得到了剪枝之后的网络，我们如果直接从大网络中继承权重去训练，那么就可以得到相似的性能。但是，如果是去从头训练这个网络，就无法得到相似的性能，而且训练过程不如上一种训练顺滑。

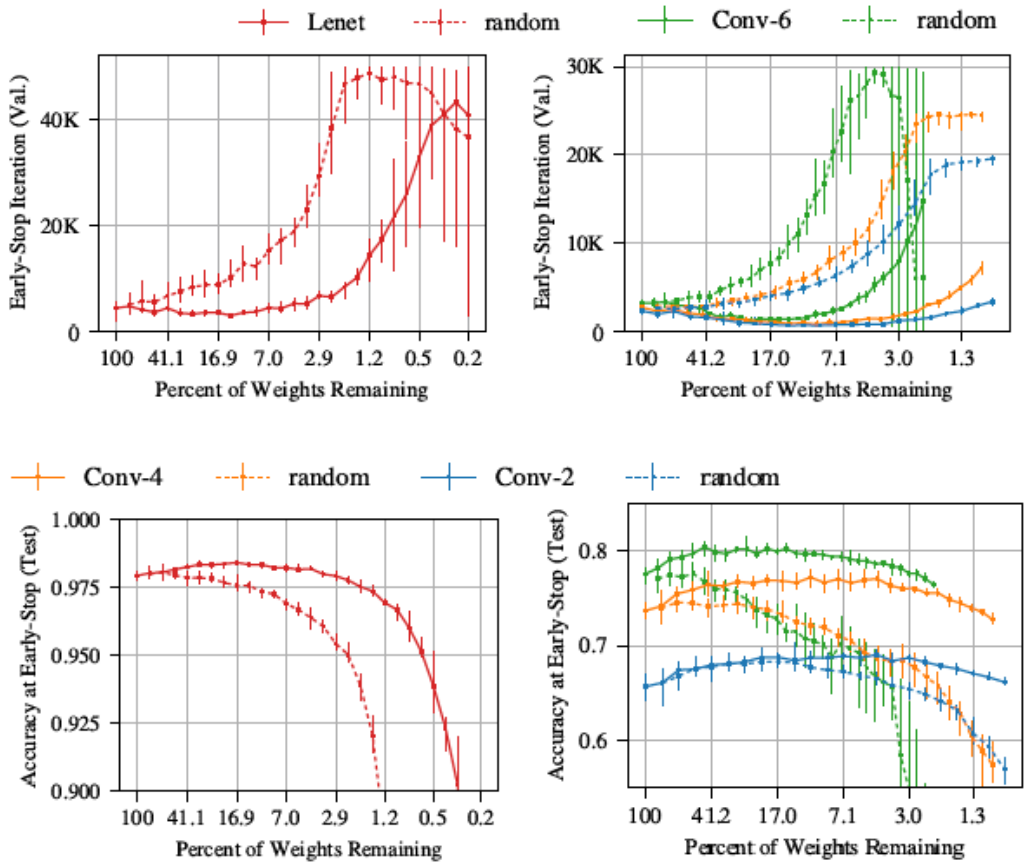
彩票假设 (Lottery Ticket Hypothesis) 是说：在密集的，随机初始化的神经网络里面存在着那么一些子网络，在从头，独立训练时可以在相似的迭代次数内达到与原始网络相当的测试精度。

这些特殊的子网络从头训练时就比较容易。假设大的网络里面包含着千千万万的子网络，我们把这些子网络看作是一张张彩票，那么这类特殊的子网络就被称之为中奖彩票 (**winning tickets**)，重复一遍它们的特点：

相似的迭代次数内达到与原始网络相当的测试精度。
容易从头训练。

所以现在的问题就是**如何把这些高品质的 winning tickets 给找出来**，这也是本文的核心贡献。

彩票假设基于的实验现象如下图1所示。图1里面4个图的横轴都是剩余参数的比例，纵轴分别是验证集损失达到最小时的iteration和在这个iteration时的精度。我们发现网络越稀疏，达到验证集最高精度时的iteration一般越大，代表着网络越难以训练，而且这个iteration时的精度越低。**虚线代表随机的10个子网络的均值，实现代表随机的5个winning tickets的均值。** 可以看到这些 winning tickets 与随机网络相比，当压缩率相同时可以更快地达到验证集最优精度，而且这个最优精度值更高。

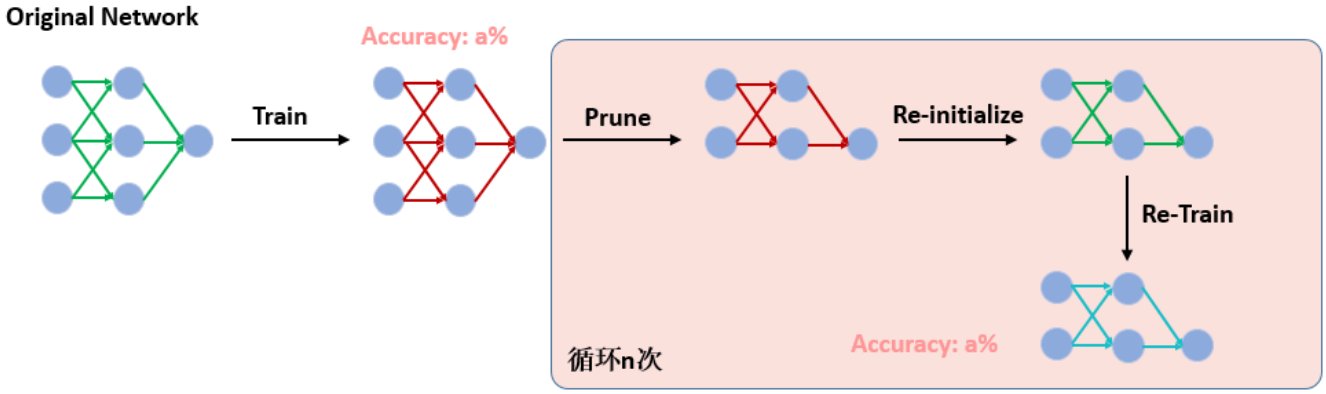


把上面的彩票假设 (Lottery Ticket Hypothesis) 用看上去更规范的数学语言表达一下就是：

> 考虑密集的前向神经网络 $f(x;\theta)$，初始化参数为 $\theta=\theta_0\sim\mathcal{D}_\theta$。使用stochastic gradient descent训练策略在训练集上训练，f 在第 j 个iteration时得到损失 l 和精度 a。同时，进行剪枝以后相当于是给网络加上一个mask $m\in 0,1^{|\theta|}$，使网络变成了 $f(x;m\odot\theta)$，那么初始化参数就要相应地变为 $m\odot\theta_0$。使用stochastic gradient descent训练策略在训练集上训练，f 在第 j' 个iteration时得到损失 l' 和精度 a'。彩票假设认为存在一个 $\exists\, m$ 使得 winning tickets 与原网络相比有：达到最优精度的迭代数更短 $j'<j$，精度更高 $a'>a$，以及参数量更小 $\|m\|_0\ll|\theta|$。

1. 随机初始化神经网络 $f(x;\theta_0)$；
2. 训练网络 j 次迭代后，得到参数 θ_j；
3. 在参数 θ_j 中剪掉 $p\%$ 的参数，生成一个mask m；
4. 在剩余的结构中用原始初始化参数 θ_0 进行训练，得到 winning tickets $f(x;m\odot\theta_0)$。

如上面的描述，这种基于彩票假设的剪枝做法是一种One-Shot的办法，即：**网络使用 θ_0 做权重初始化，一共只需要训练1次，剪掉 $p\%$ 的权重，剩下的权重再按照 θ_0 做初始化。得到的这个新的网络就叫做 winning tickets。** 这个过程我画了个示意图，如下图2所示。



原始的网络经过训练得到精度为 a 的网络，我们对它进行剪枝以后，再按照原始网络初始化的权重初始化剪枝后的网络 (Re-initialize)，这步得到的就是 winning tickets，再将它重新训练得到一个精度更好于 a 的子网络。这样一来，通过得到 winning tickets，我们获得了剪枝率为 $p\%$ 的精度不变的网络。值得注意的是红色框内的步骤可以循环进行，如果循环 n 次，我们每一步的剪枝率为 $p^{\frac{1}{n}}\%$ 即可。

结果： 作者能在在MNIST的全连接架构和CIFAR10的卷积架构中识别 winning tickets 。 基于剪枝的寻找中奖彩票的策略对学习率很敏感：需要warmup才能以更高的学习率找到 winning tickets 。找到的 winning tickets 是原始网络size的10-20%（或更小）。 **在这个size下，它们在最多相同的迭代次数（相应的训练时间）内达到或超过原始网络的测试精度。但是，一旦随机重新初始化**时，即**破坏了彩票假设的原则， winning tickets 就会表现得更糟**，这意味着单独的结构无法解释中奖彩票的成功。

为了使文章更丰富多彩，增加一点可读性和趣味性，**接下来放了个小视频**，生动展示彩票假设的原理：

在这个视频里面主要对比使用原网络初始化的权值和随机初始化权值所产生的差异。

作者在本文中讨论了全连接层和卷积层的彩票假设，其对应的数据集分别设计成MNIST和CIFAR-10。

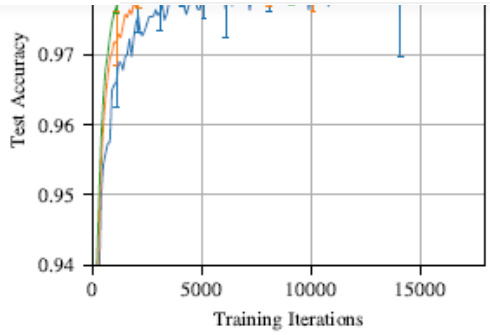
1.1.2 全连接层中的 winning tickets

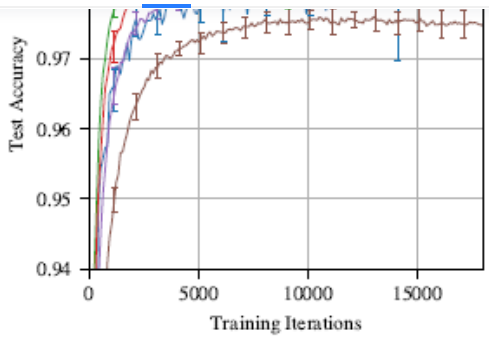
数据集：MNIST

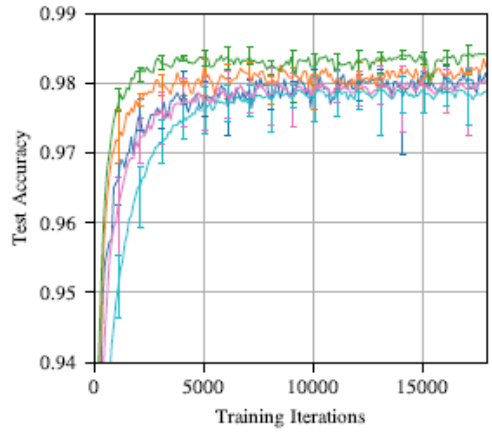
网络架构：Lenet300-100

Notation： $P_m=\frac{\|m\|_0}{|\theta|}$ 代表剪枝率， $P_m=25\%$ 代表剪掉了 75% 的权重。

下图3是不同 P_m 下的模型的Test Accuracy，实验的具体做法就如图2所示。不同剪枝率的子网络性能不一，其中最好的是绿色线，即仅保留21.1%的权重的子网络。当 $P_m<21.1\%$ 时，学习速度下降。当 $P_m=3.6\%$ 时，winning ticket 的性能退化成原始网络的性能。

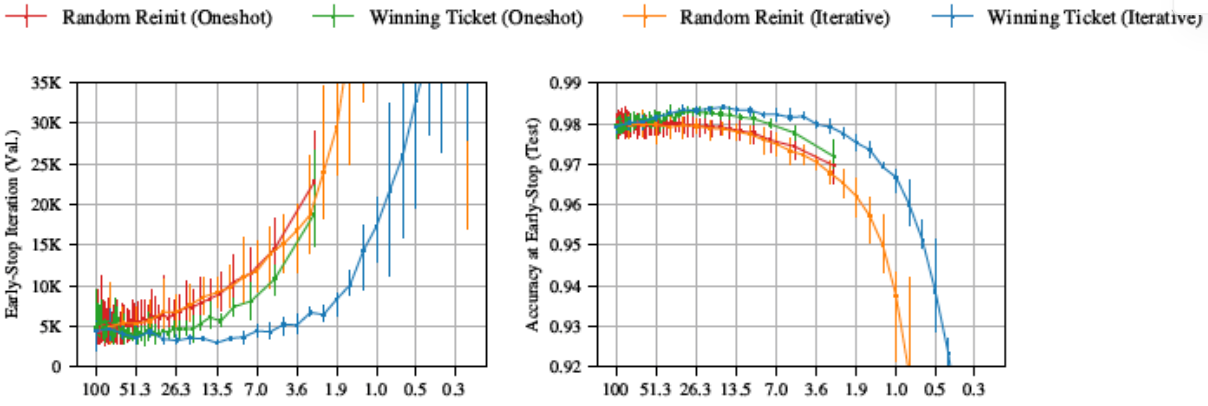


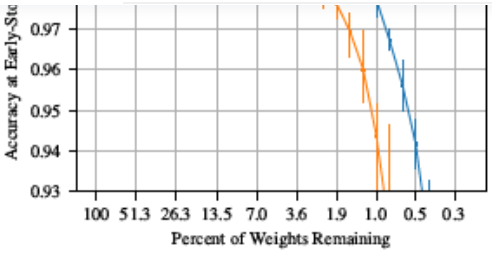




下图4是2种剪枝方法 (Random Reinit, 彩票假设) 使用2种不同的策略 (Oneshot，就是一次剪枝完成。Iterative，就是分多次剪枝完成) 的性能曲线。图4最左上角这个图是验证集loss第1次达到最低时的iteration，我们发现蓝色曲线，也就是彩票假设使用Iterative策略相比其他可以最早达到最优的性能。这里举个例子说明一下图的含义吧，比如图4的最左上角的图的蓝色曲线上的一个点 (26.3, 4K)，它所代表的含义是当使用彩票假设方法时，压缩率是26.3时，得到的Re-initialize之后的模型对它重新训练，在第1次达到最佳性能时只用了4K个iteration，而原网络训练时第1次达到最佳性能时需要用5K个iteration，证明了**彩票假设得到的子网络相较于原网络的训练速度更快。** 再比如图4的最右上角的图的蓝色曲线上的一个点 (26.3, 0.984)，它所代表的含义是当使用彩票假，压缩率是26.3时，得到的Re-initialize之后的模型对它重新训练，得到的最佳性能是98.4% Accuracy，而原网络训练时得性能是98.0% Accuracy，证明了**彩票假设得到的子网络相较于原网络精度更高。** 而且，根据图4的第2张图可以看到，当 P % 时，使用Iterative策略的 winning ticket 的性能依然可以和原网络保持一致。当 $P_m > 3.6\%$ 时，使用Iterative策略的 cket 的性能可以超过剪枝前的原网络的性能。

图4中的红色线橙色线代表不使用彩票假设的情况，这种和彩票假设的区别就在于剪枝后的网络初始化的方法不同。彩票随机初始化，而应该按照原网络初始化的参数进行初始化。

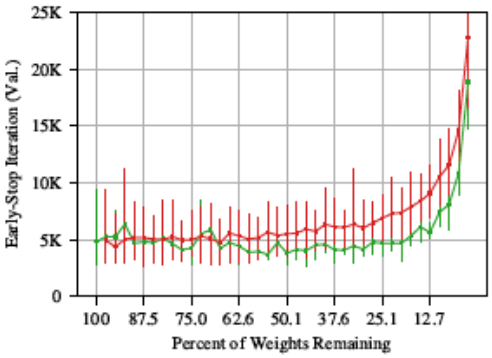


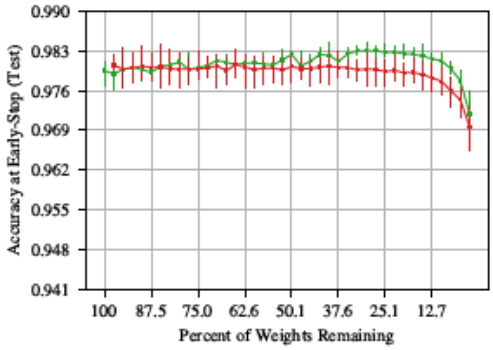


下图5展示的是One-shot pruning的结果。One-shot pruning就是不循环 n 次地进行剪枝，一次就达到目标的压缩率。这种策略下使用彩票假设的曲线就对应绿色线，反之随机初始化的曲线就对应红色线。在这种策略下：

当 $67.5\% \geq P_m \geq 17.6$ 时，彩票假设压缩的模型比原模型要更早地达到最优精度。
当 $95.0\% \geq P_m \geq 5.17$ 时，彩票假设压缩的模型比原模型的最优精度更高。

但是，Iterative pruning的做法比One-shot pruning训练更快，精度更高。





1.1.2 卷积层中的 winning tickets

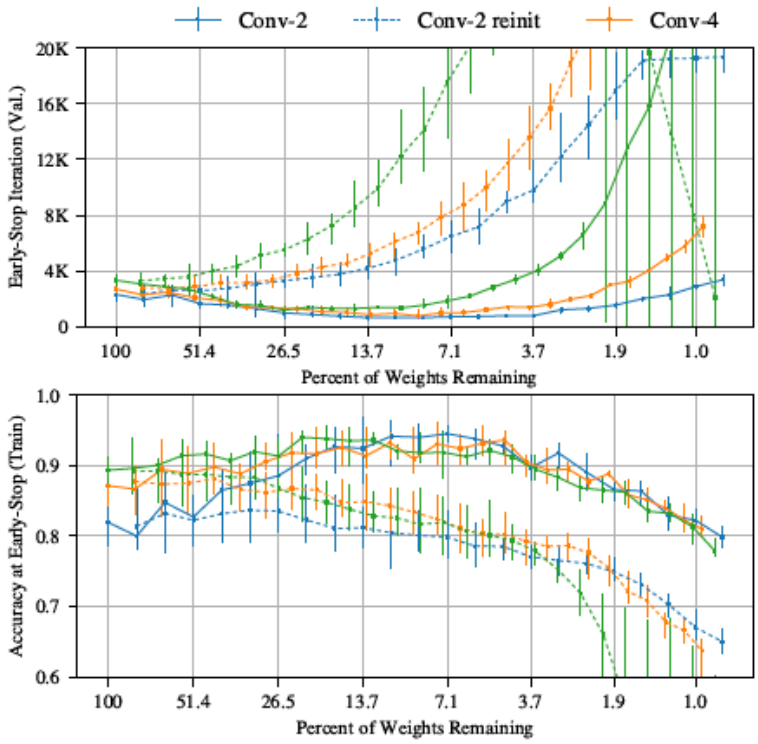
数据集：CIFAR10

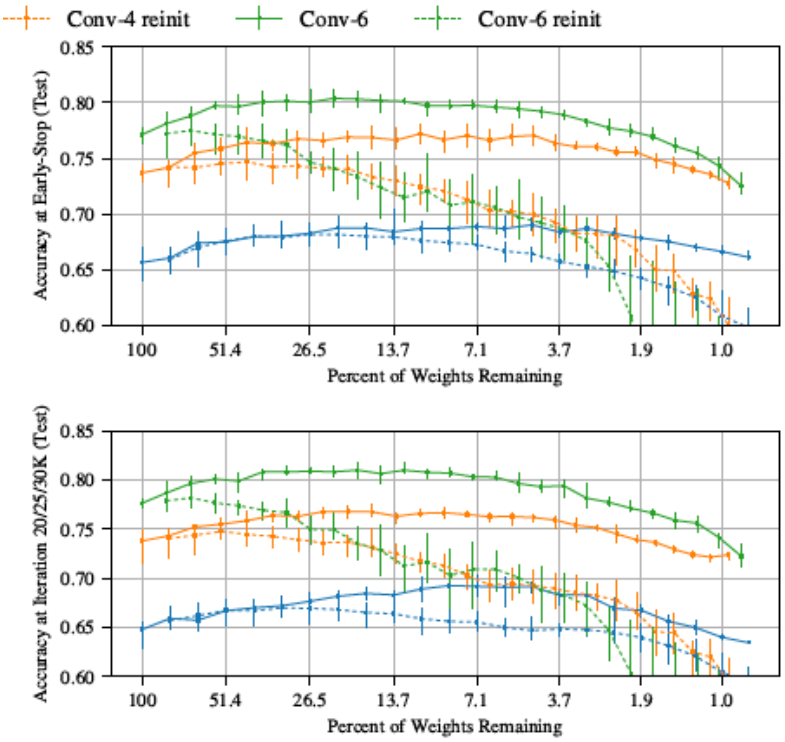
网络架构：Conv-2，Conv-4，Conv-6，分别是2层，4层，6层卷积后面跟2个FC层，每2层卷积之后有Max-Pooling池化层，如下图6所示。

Network	Lenet	Conv-2	Conv-4	Conv-6	Resnet-18	VGG-19
Convolutions		64, 64, pool	64, 64, pool 128, 128, pool	64, 64, pool 128, 128, pool 256, 256, pool	16, 3x[16, 16] 3x[32, 32] 3x[64, 64]	2x64 pool 2x128 pool, 4x256, pool 4x512, pool, 4x51
FC Layers	300, 100, 10	256, 256, 10	256, 256, 10	256, 256, 10	avg-pool, 10	avg-pool, 10
All/Conv Weights	266K	4.3M / 38K	2.4M / 260K	1.7M / 1.1M	274K / 270K	20.0M
Iterations/Batch	50K / 60	20K / 60	25K / 60	30K / 60	30K / 128	112K / 64
Optimizer	Adam 1.2e-3	Adam 2e-4	Adam 3e-4	Adam 3e-4	← SGD 0.1-0.01-0.001 Momentum 0.9 →	
Pruning Rate	fc20%	conv10% fc20%	conv10% fc20%	conv15% fc20%	conv20% fc0%	conv20% fc0%

下图7展示的是Conv-2 (蓝色线)，Conv-4 (橙色线)，Conv-6 (绿色线)这几种模型在不同剪枝率下的实验结果，实线代表使用彩票假设压缩模型的实验结果，虚线代表随机初始化剪枝的结果。我们还是举个例子解释下曲线的含义：就以图7第1张图Conv-6曲线上的点 (3.7, 4K) 为例，它所代表的含义是当使用彩票假设方法时，压缩率是3.7 (就是仅仅保留3.7%的参数)时，得到的Re-initialize之后的模型对它重新训练，在第1次达到最佳性能时只用了4K个iteration，而原网络训练时第1次达到最佳性能时需要用3.8K个iteration，证明了**彩票假设得到的子网络相较于原网络的训练速度差不多。** 再比如图4的第2张图的Conv-6曲线上的点 (3.7, 0.9) ，它所代表的含义是当使用彩票假设方法时，压缩率是3.7时，得到的Re-initialize之后的模型对它重新训练，得到的训练集最佳性能是90%

图7中所有的虚线代表不使用彩票假设的情况，观察到的现象依然和MNIST数据集一致，在不使用彩票假设，即对剪枝后的模型使用随机初始化时，无论是从训练的难度还是训练的精度来讲都会差很多。



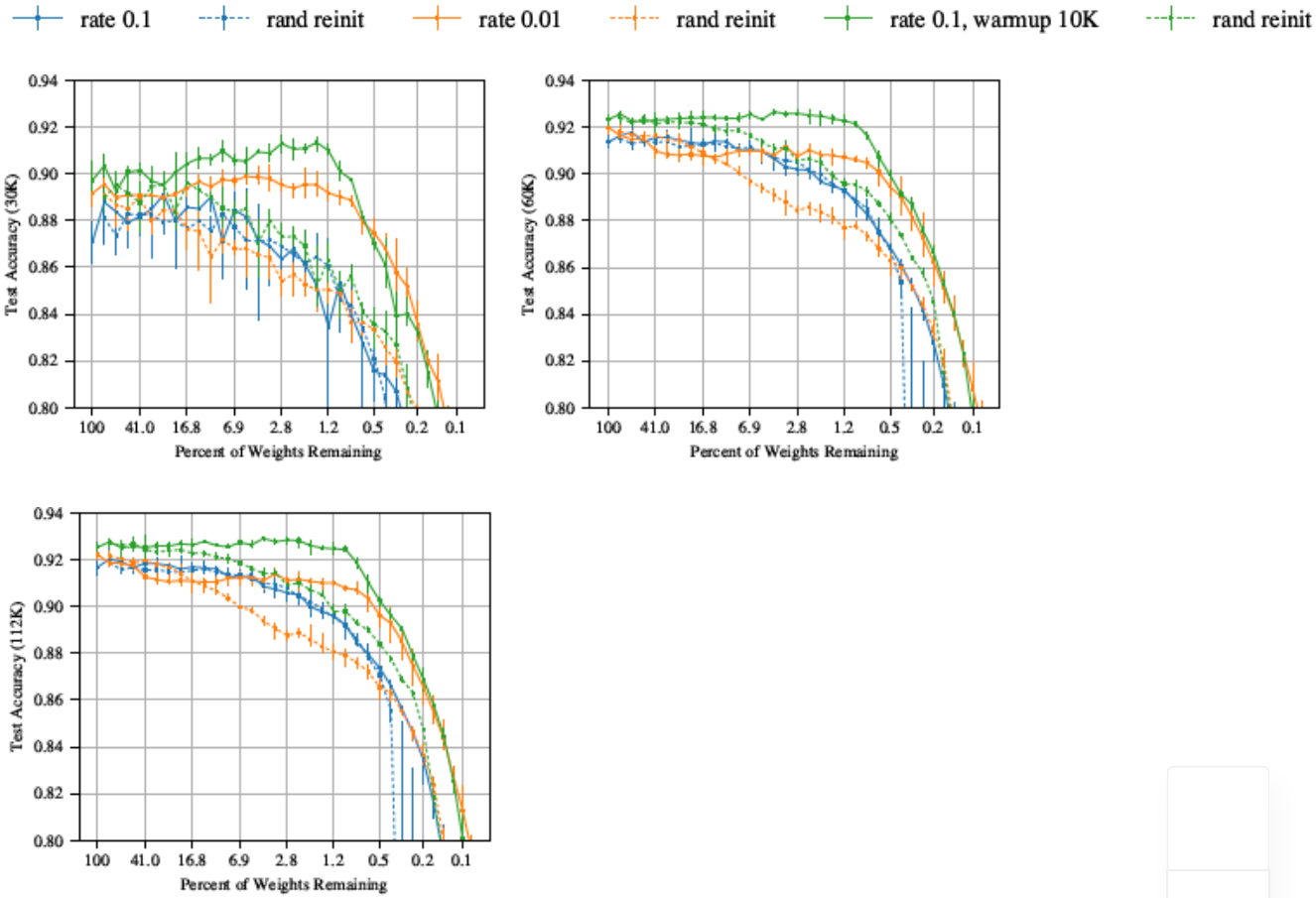


除此之外，作者还发现使用Dropout训练网络可以进一步提升测试的精度，说明彩票假设和Dropout方法是正交的。

1.1.4 VGG和ResNet-18模型

这样做的原因是：每一个层的参数量是不同的，比如VGG-19前2层只有约1.7万和3.6万个参数，最后1层有九百万个参数。当所有的层以相同的比率被修剪时，这些更小的层就会成为瓶颈，阻止我们识别出可能的最小的 winning ticket。Global pruning的则可以避免这个问题。

VGG-19和ResNet-18的实验结果大同小异，为了节约篇幅我们就只看VGG-19的结果吧：如下图8所示，实线代表使用彩票假设得到的实验结果，虚线代表随机初始化参数的实验结果。 在较高的学习率 (0.1，绿色线) 下，Iterative的彩票假设做法不会找到 winning tickets，而且剪枝后的网络的性能并不比随机重新初始化时好。然而，在较低的学习率 (0.01，橙色线) 下，彩票假设做法能够找到 winning tickets。



ResNet-18的实验结果与VGG-19的大同小异，在较高的学习率下，Iterative的彩票假设做法不会找到 winning tickets，在较低的学习率下，彩票假设做法能够找到winning tickets。

FAQ1：彩票假设的做法比常规剪枝方法强在哪里?

答1： 传统剪枝方法可以在VGG-19网络剪掉80%的参数而不影响性能，但是一旦剪枝98.5%的参数，性能会大幅下降。 因为**高度过参数化的网络在一定程度上可以被成功地修剪、重新随机初始化和再训练而成功地保持精度。** 然而，超过这一点，极端修剪98.5%的参数，在参数随机初始化时就无法在保证准确率，只有偶然初始化才行，而这个偶然初始化正是彩票假设的做法。

FAQ2：彩票假设为啥要训练原始的网络，为啥不直接剪枝，按照一开始的初始化权重训练?

答2： 训练原始网络的目的是分辨哪些权重应该被剪掉 (比如剪掉每层权值最低的10%的权重)，如果不训练那就无法区分。

FAQ3：为什么一定要按照一开始的初始化权重训练?

答3： 这里提供一种解释的思路，就是假设我们初始化的权重是 $\boldsymbol{\theta_0}$ ，我们训练好原网络以后进行剪枝，得到剪枝之后的winning ticket $\boldsymbol{T_0}$ ，那么正是由于初始化的权重是 $\boldsymbol{\theta_0}$ ， $\boldsymbol{T_0}$ 才会被选择为winning ticket。而如果初始化的权重是 $\boldsymbol{\theta_1}$ ，很有可能中奖彩票就变成了 $\boldsymbol{T_1}$ 。换句话讲，中奖彩票的结构 $\boldsymbol{T}$ 与初始化权重 $\boldsymbol{\theta}$ 是强相关的，所以这也提供了一种一定要按照一开始的初始化权重去训练

小结：

彩票假设颠覆了DNN的修剪，其核心动机是：**与其训练大型网络并将其削减为较小的网络，不如从一开始就确定并训练最优的小网络？彩票假设的做法可以用一句话概括：**

> 先训练一个网络，剪掉一部分，剩下的没剪掉的连接或者权重就构成了 winning tickets 。值得注意的是，这些剩下的权重在初始化的时候的那个初始化的值要与**剪枝前的原网络训练前的初始化值**保持一模一样，再**按照这个初始化值直接训练子网络**即可(整个流程如图2)。

当随机重新初始化时，winning tickets学习更慢，测试准确率更低，这表明初始化对其训练成功很重要。 winning tickets 可以达到与原始的、未经修剪的网络相同的精度，但参数显著减少。这一观察结果与最近关于神经网络训练中过度参数化作用 (Over-parameterization) 有关。 过度参数化的网络更容易训练，因为它们有更多潜在winning tickets 的子网络组合。

2 中奖彩票的泛化性：在不同数据集和优化器之间可以迁移吗？(NIPS 2019)

> **论文名称：One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers**

论文地址：

https://arxiv.org/pdf/1906.02773.pdfarxiv.org

2.1 原理分析：

2.1.1 中奖彩票具有泛化性能吗?

彩票假设的核心做法是剪枝之后的网络的权重在初始化的时候的那个初始化的值要与**剪枝前的原网络训练前的初始化值**保持一模一样，再**按照这个初始化值直接训练子网络**。winning tickets 与原始密集网络相比，所包含的非零参数要少得多，但在单独训练时可以达到类似甚至更好的性能。这个发现意味着我们可以训练一个高度紧凑的子网络，潜在减少了网络训练的成本。之前的做法是迭代剪枝 (Iterative prunning)，即多次循环，每次剪掉一部分模型，如图2所示。但是这样的做法需要不断地从头去训练网络，因此是计算密集的。相反如果说一次就剪枝结束，达到要求的剪枝率 (One-shot Prunning) 的话，就很有可能剪掉那些实际上比较重要的权值。

一种可能的解决的办法是将 winning tickets 在不同的数据集和优化器上进行重用。

但问题是在一个数据集和优化器上得到的 winning tickets 在**其他数据集和优化器**上的表现如何呢？ 换句话讲，中奖彩票具有泛化性吗?

如果 winning tickets 对数据集和优化器是过拟合的，那么我们一旦换了数据集，就需要在新的数据集上重新训练模型找 winning tickets，那就会使的 winning tickets的价值大打折扣。相反，如果 winning tickets 对数据集和优化器是泛化的，那么我们就可以在一些数据集上训练 winning tickets 然后在其他数据集上重用。

所以本文的目的就是回答在一个数据集或优化器上得到的 winning tickets 在其他数据集或优化器上能否复用。

作者通过实验给出了以下的3个回答：

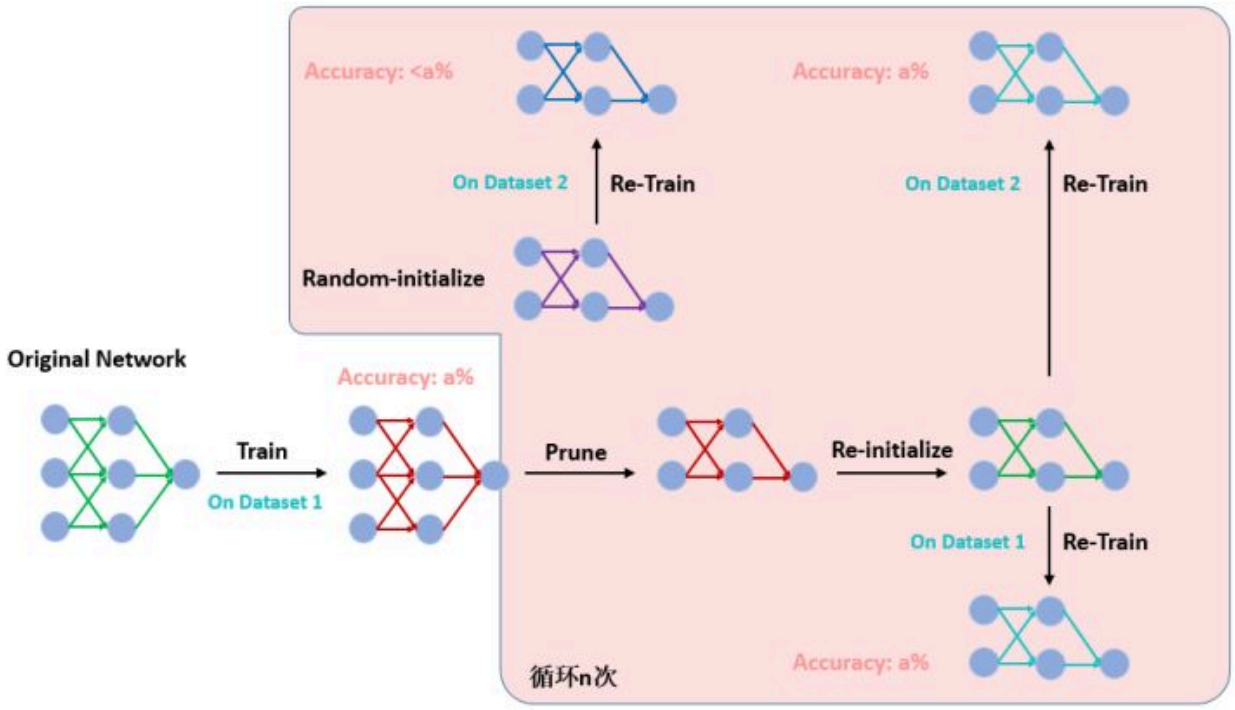
> 对于自然图像域的不同数据集：Fashion MNIST, SVHN, CIFAR-10/100, ImageNet, 和 Places365，在一个数据集上得到的 winning tickets 在其他数据集具有很好的泛化性。
> 在大数据集上得到的 winning tickets 的泛化性能相比于小数据集来讲，更好。
> 在一种优化器上得到的 winning tickets 在其他优化器上面具有很好的泛化性。

在这个工作里面作者依旧使用了Global Prunning，Global pruning 是啥意思呢？就是说之前的剪枝策略是各个层保持剪枝率一致：比如第1层保留25%的参数，那后面的层也都是保留25%的参数。而Global pruning的剪枝策略是说我一共把这个模型保留25%的参数就OK，可能第一层保留了90%的参数而最后一层只保留了12%的参数。

这样做的原因是： 每一个层的参数量是不同的，比如VGG-19前2层只有约1728和36864个参数，最后1层有几百万个参数。当所有的层以相同的比率被修剪时，这些更小的层就会成为瓶颈，影响网络的表达能力。Global pruning的则可以避免这个问题。作者也发现了浅层剪枝较少，而深层剪枝较多的现象

比如CIFAR-10数据集最后的 linear transformation 层权重的维度是 (128, 10) 的，现在想换成CIFAR-100数据集，只需要把这个最后的 linear transformation 层换成维度是 (128, 100) 的，权重随机初始化即可。

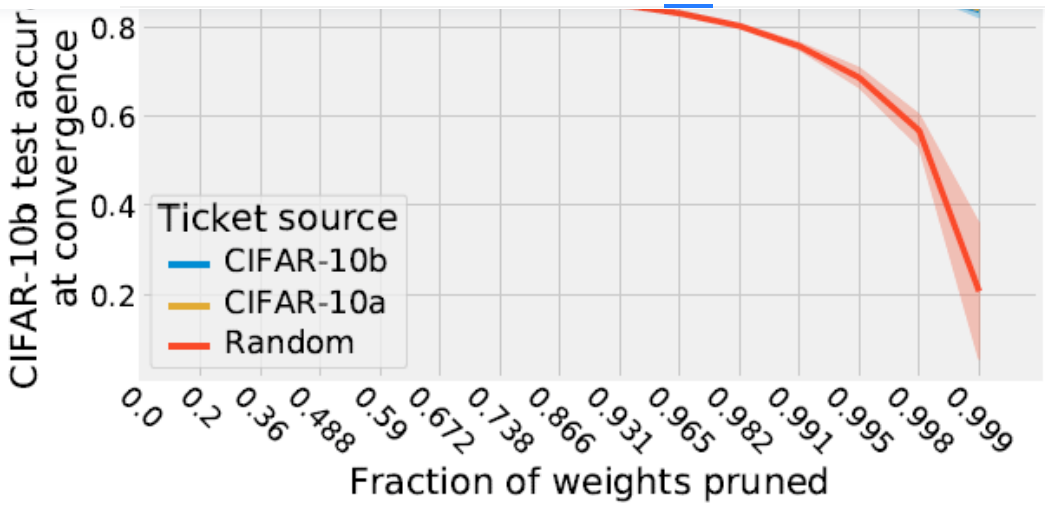
另外因为我们需要在另一个数据集上面验证 winning ticket 的泛化性，所以就需要每个Iteration 额外多做2组实验，如下图9所示。读者可以对比下图9和图2的差距，就在于每个Iteration 要额外把**随机初始化的子网络**和**按照原网络训练前的初始化值初始化的子网络**重新在新的数据集上训练。

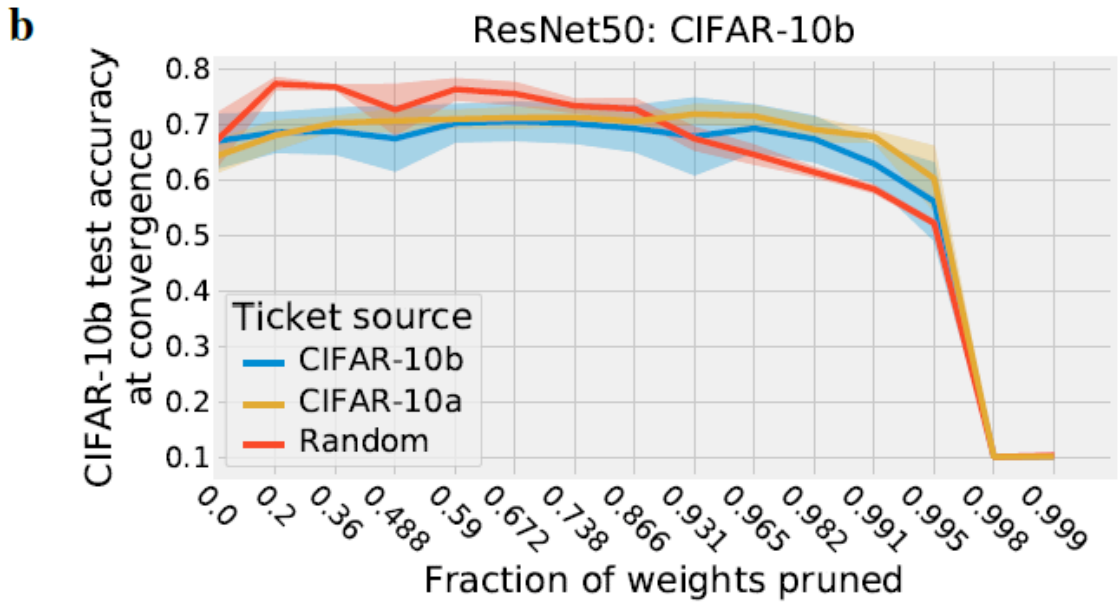


2.1.3 泛化性能实验1：相同分布数据的迁移性能

作者首先尝试最简单的任务：即从数据集A迁移到数据集B (数据集A和B是来自同一数据集的)，看看 winning ticket 的泛化性能如何。所以作者将CIFAR-10分成了2部分：CIFAR-10a和CIFAR-10b作为图9的Dataset 1和Dataset 2。每个数据集都包含25[illegible]张训练图片，10个类。实验结果如下图10所示，这个曲线的意思是：从CIFAR-10a里面找一个winning ticket，迁移到CIFAR-1[illegible]并对比CIFAR-10b里面的一个 winning ticket和一个随机初始化的ticket。

结果发现对于VGG-19模型来讲CIFAR-10a中得到的 winning tickets 可以很好地迁移到CIFAR-10b，而且精度高于随机[illegible]模型。但是对于ResNet-50模型在剪枝量不大时它们的性能就不如随机初始化的模型，说明在剪枝量不大时 winning tick[illegible]能下降，开始变得对数据集敏感。





2.1.3 泛化性能实验2：不同数据集的迁移性能

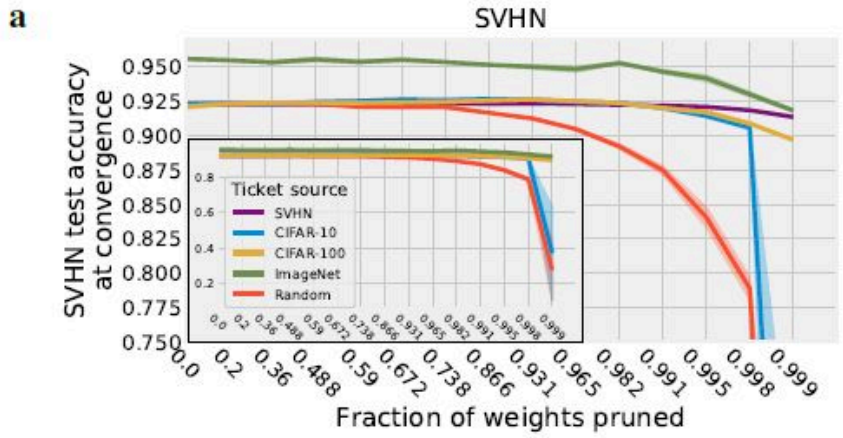
作者首先尝试最简单的任务：即从数据集A迁移到数据集B (数据集A和B是来自不同数据集的，但都是自然图像)，看看 cket 的泛化性能如何。数据集有Fashion MNIST, SVHN, CIFAR-10/100, ImageNet, 和 Places365。

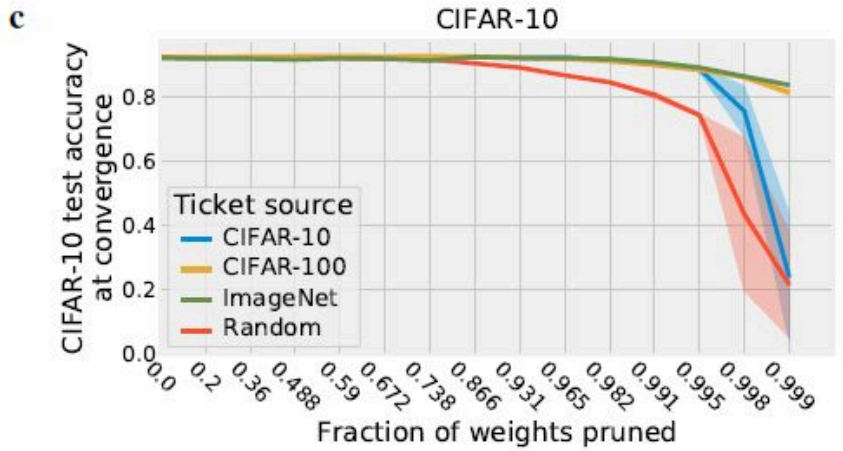
具体而言，作者在一个源训练配置中生成中奖彩票，并在不同的目标数据集中评估中奖彩票的性能。例如，我们可以使用CIFAR-10数据集 (源配置) 生成中奖彩票，并在ImageNet数据集(目标配置) 上评估它的性能。通过一系列不同设置的严格实验，我们观察到中奖彩票在不同的图像数据集上可以泛化。有趣的是，我们还观察到，由大型数据集 (如ImageNet和Places365) 生成的中奖彩票的迁移效果始终比小数据集 (如CIFAR-10) 好得多。

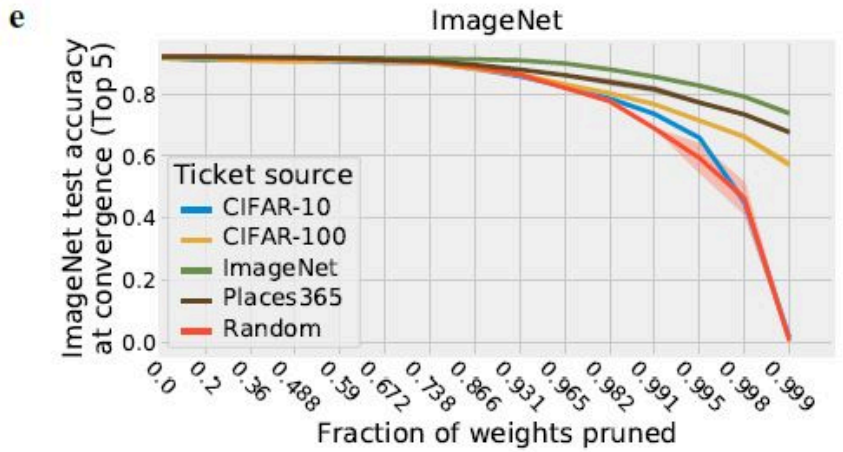
> 类别多的数据集生成的中奖彩票 优于 类别少的数据集生成的中奖彩票：

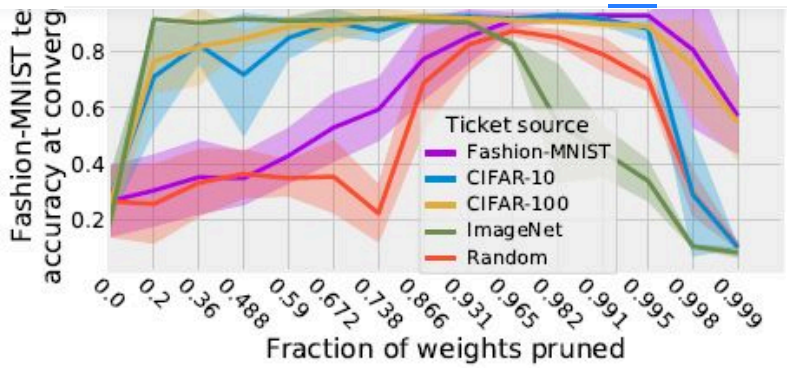
下图11是VGG-19模型在不同数据集的结果。图19的这些图显示了物体分类模型的中奖彩票是如何跨越大型数据集 (ImageNet和Places365) 和小型数据集 (CIFAR-10/CIFAR-100) 的。图中不同的线代表中奖彩票的不同源数据集。在ImageNet和Places365上生成的中奖彩票始终优于在较小数据集上生成的中奖彩票。

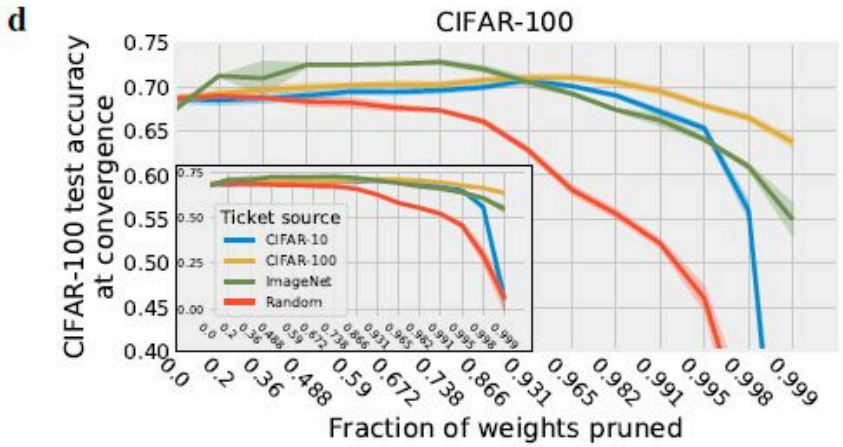
这个结果并不是数据集大小决定的，而是数据集的类别所决定。因为CIFAR-10和CIFAR-100的数据集都是50000张图片，可是泛化

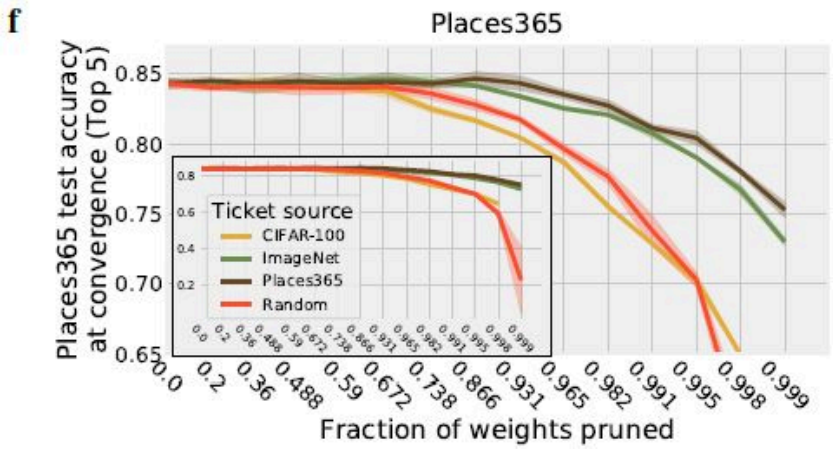






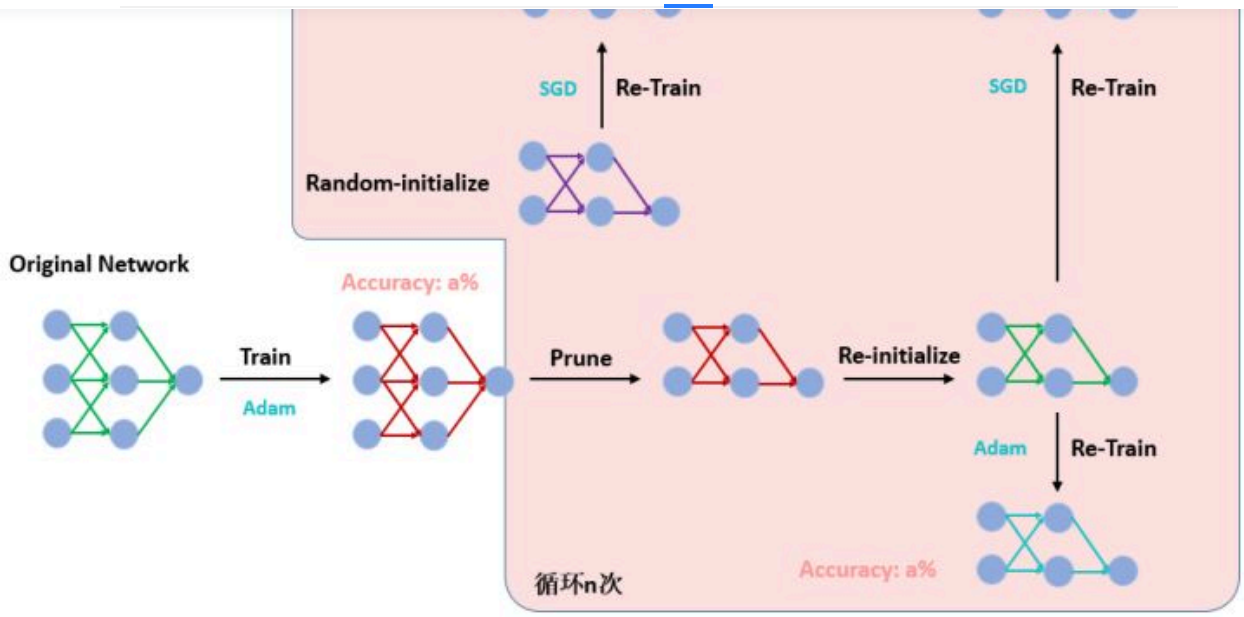


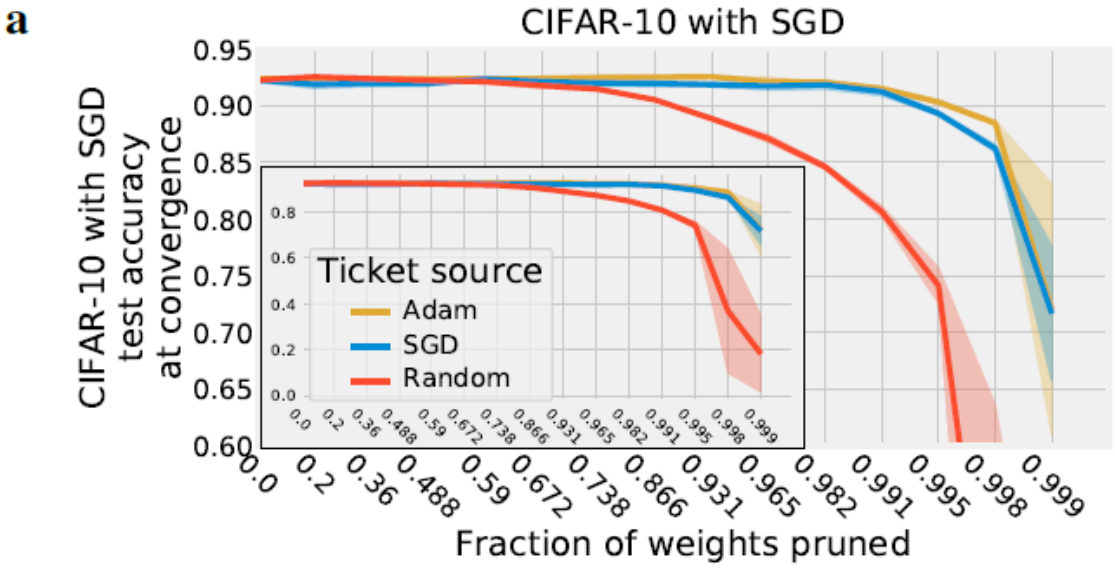


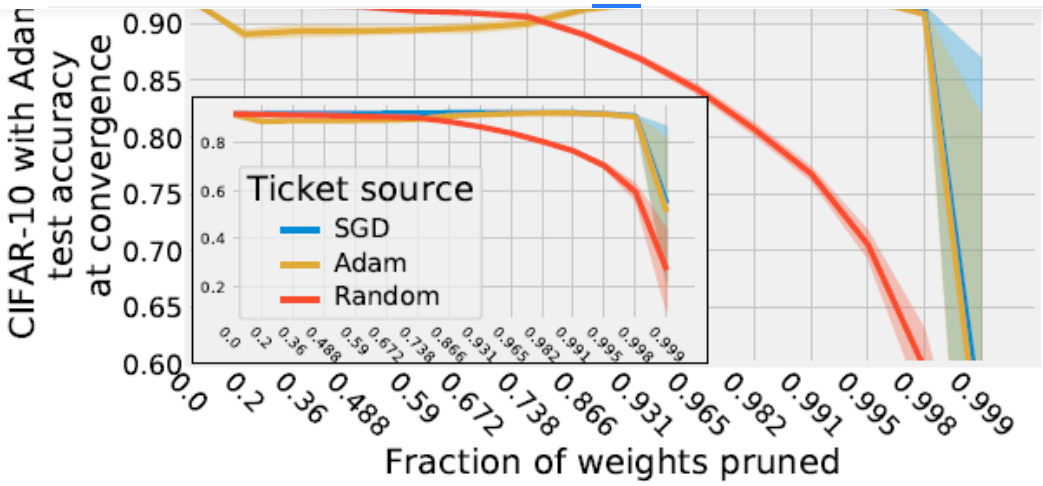


2.1.4 泛化性能实验3：不同优化器的迁移性能

最后作者尝试变换优化器的任务：即从优化器A迁移到优化器B，看看 winning ticket 的泛化性能如何。结果如下图13所示，图中不同的线代表中奖彩票的不同源优化器，我们以第1张图为例，黄色线代表Adam优化器训练的模型在压缩后，权重回退成初始值，再用SGD优化器训练的性能，整个过程如图12所示，一个压缩后的模型，权重按照训练前的权重初始化以后，用SGD作为优化器训练的性能与用Adam优化器训练的性能相当，说明彩票假设对于不同的优化器也有好的泛化性能。







小结

本文回答了在一个数据集或优化器上得到的 winning tickets 在其他数据集或优化器上能否复用。

作者通过实验给出了以下的3个回答：

> 对于自然图像域的不同数据集：Fashion MNIST, SVHN, CIFAR-10/100, ImageNet, 和 Places365，在一个数据集上得到的 winning tickets 在其他数据集具有很好的泛化性。
> 在大数据集上得到的 winning tickets 的泛化性能相比于小数据集来讲，更好。
> 在一种优化器上得到的 winning tickets 在其他优化器上面具有很好的泛化性。

3 弹性彩票假设：在不同网络架构之间可以迁移吗？(Arxiv)

> 论文名称：**The Elastic Lottery Ticket Hypothesis**

论文地址：

The Elastic Lottery Ticket Hypothesisarxiv.org

3.1 ELTH原理分析：

3.1.1 中奖彩票在不同网络架构之间可以迁移吗?

彩票假设的核心做法是剪枝之后的网络的权重在初始化的时候的那个初始化的值要与**剪枝前的原网络训练前的初始化值**一样，再**按照这个初始化值直接训练子网络**。winning tickets 与原始密集网络相比，所包含的非零参数要少得多，但在单可以达到类似甚至更好的性能。这个发现意味着我们可以训练一个高度紧凑的子网络，潜在减少了网络训练的成本。

之前的做法是迭代剪枝 (Iterative prunning)，即多次循环，每次剪掉一部分模型，如图2所示。但是这样的做法需要不断训练网络，因此是计算密集的，而且无法迁移到其他的网络上面。针对这个计算密集的问题，One ticket to win them all这个paper通过实验给出了回答：

> **在一种数据集或优化器上得到的 winning tickets 在其他数据集或优化器上面具有很好的泛化性。**

于是，本文作者提出了更进一步的问题：

> **在一种网络架构上得到的 winning tickets 在其他网络架构上面具有很好的泛化性能吗?**
> **换句话讲，可否把一个网络得到的winning ticket直接迁移到另一个网络上面，而不需要重新做复杂的迭代剪枝操作?**

若能解决掉这个问题， 那么我们将只对一个网络执行昂贵的迭代剪枝过程，然后自动为其他网络获得中奖彩票。那就相当于找到了一个 "Once-for-all" 的 winning ticket，它适用于多种不同的网络架构。 首先在一个小的**源网络架构**上通过IMP (Iterative Magnitude-based Pruning) 找到中奖的彩票，然后将其转移到一个大得多的**目标网络架构**上，与直接在后者上执行IMP相比，这会带来计算资源

> 在同一个网络架构家族中 (比如ResNets, MobileNets, EfficientNets, Transformers) 的一个网络的 winning tickets 可以被**延伸或压缩**成为另一个网络，这个网络是网络架构家族中的另一个网络的 winning tickets。

到这里你会发现其实最重要的是这个 "延伸或压缩网络的办法"，本文给出的方案是 **Elastic Ticket Transformations (ETTs)：将中奖彩票的稀疏架构转换为其更深或更浅的变体**。

把本文的弹性彩票假设 (Elastic Lottery Ticket Hypothesis) 用看上去更规范的数学语言表达一下就是：

> 假设我们使用IMP在一个源网络上识别了一张中奖彩票 $f^s(\theta_r^s, m^s)$，其中回退 (rewind，意思就是把权重不回退到初始化的权重，而是回退到训练 r 步时的权重) 之后的权值是 θ_r^s，同时剪枝的binary mask是 m^s。我们的目标是把中奖彩票 $f^s(\theta_r^s, m^s)$ 变成目标网络 $f^t\left(g(\theta_r^s, m^s), h(\theta_r^s, m^s)\right)$ 直接使用不同的深度，避免在目标网络上再次运行昂贵的IMP，这里 $g(\cdot,\cdot)$ 和 $h(\cdot,\cdot)$ 代表权值和剪枝mask的变换映射。

3.1.2 Elastic Ticket Transformations (ETTs)

对于VGG网络作者将一个卷积层和与其相关的归一化层视为一个整体的 "Block"。

对于ResNet系列网络作者将一个残差块 (残差块由两个或三个卷积层、归一化层和一条shortcut 组成) 视为一个整体的 "Block"。

Elastic Ticket Transformations在进行网络架构变换的同时，有几个变量是不能改变的：

1. stages的数量：因为不同的stage之间由下采样的block分开，改变stage的数量无疑会改变下采样的次数，造成网络架构的不均衡。
2. Down-sampling Block
3. 输入和输出层在源网络和目标网络中具有一致的维度，因此可以直接重用。

拉伸过程：

我们以 ResNet-20 作为源网络，而 ResNet-32 作为目标网络为例，在 ResNet-20 的每个stage里面包含有1个下采样块 $\boldsymbol{B_0^s}$ 和2个正常的块 $\boldsymbol{B_1^s}$, $\boldsymbol{B_2^s}$。而在 ResNet-32 的每个stage里面包含有4个正常块。那么为了将ResNet-20的**中奖彩票**延伸到ResNet-32，我们需要在每个阶段添加2个剩余的块。那么，**我们应该复制 $\boldsymbol{B_1^s}$ 和 $\boldsymbol{B_2^s}$ 一次，还是只复制 $\boldsymbol{B_1^s}$ (或 $\boldsymbol{B_2^s}$) 两次?**

作者实际的选择是：**复制 $\boldsymbol{B_1^s}$ 和 $\boldsymbol{B_2^s}$ 一次。** 因为这种策略可以更好地保持每个阶段的稀疏比，从而保持整个网络的稀疏性。如果 $\boldsymbol{B_1^s}$ 和 $\boldsymbol{B_2^s}$ 具有不同的稀疏度比 (我们观察到通常后面的块是稀疏的)，只复制其中一个几次将增加或减少整体的稀疏度。相比之下，按比例复制 $\boldsymbol{B_1^s}$ 和 $\boldsymbol{B_2^s}$ 保持了总体稀疏度比。

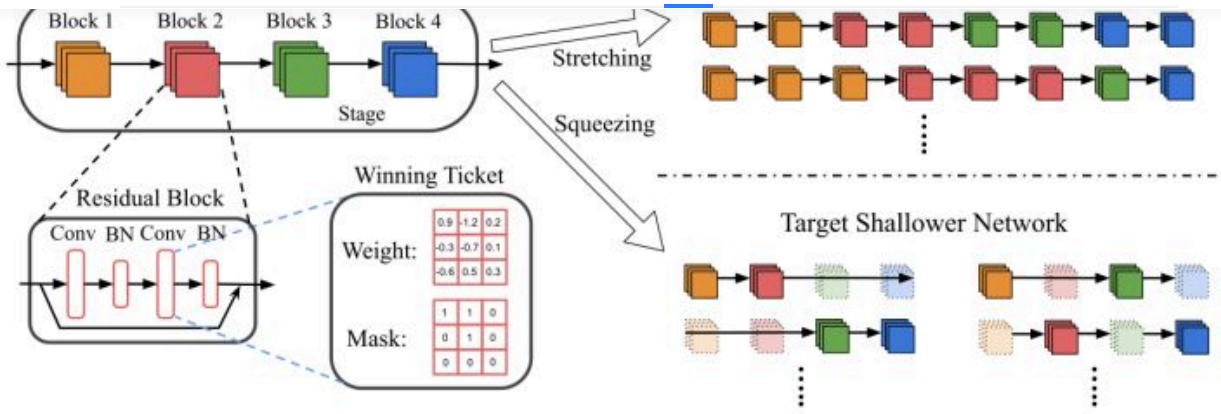
接下来是复制之后块的排序问题：这里有2种排序的办法，分别是：

$$\textbf{(appending)}\ \boldsymbol{B_0^s \to B_1^s \to B_2^s \to B_1^s \to B_2^s}\ \textbf{(interpolation)}\ \boldsymbol{B_0^s \to B_1^s \to B_1^s \to B_2^s \to B_2^s}$$

作者使用两种排序策略进行实验，并观察相当的性能。

压缩过程：

将一个源网络中的中奖彩票压缩到一个较浅的网络中是拉伸的反向过程，现在我们需要决定放弃哪个单元。我们以 Res 为源网络，而 ResNet-20 作为目标网络为例，那我们是应该扔掉连续的块，比如 $\boldsymbol{B_1^s, B_2^s}$ 或 $\boldsymbol{B_3^s, B_4^s}$，还是应该扔掉不连续的块，比如 $\boldsymbol{B_1^s, B_3^s}$ 或 $\boldsymbol{B_2^s, B_4^s}$? 作者通过实验发现这个是无所谓，实际情况是对扔掉连续的块亦或是间隔的块不敏感。拉伸过程以及压缩过程如下图所示。



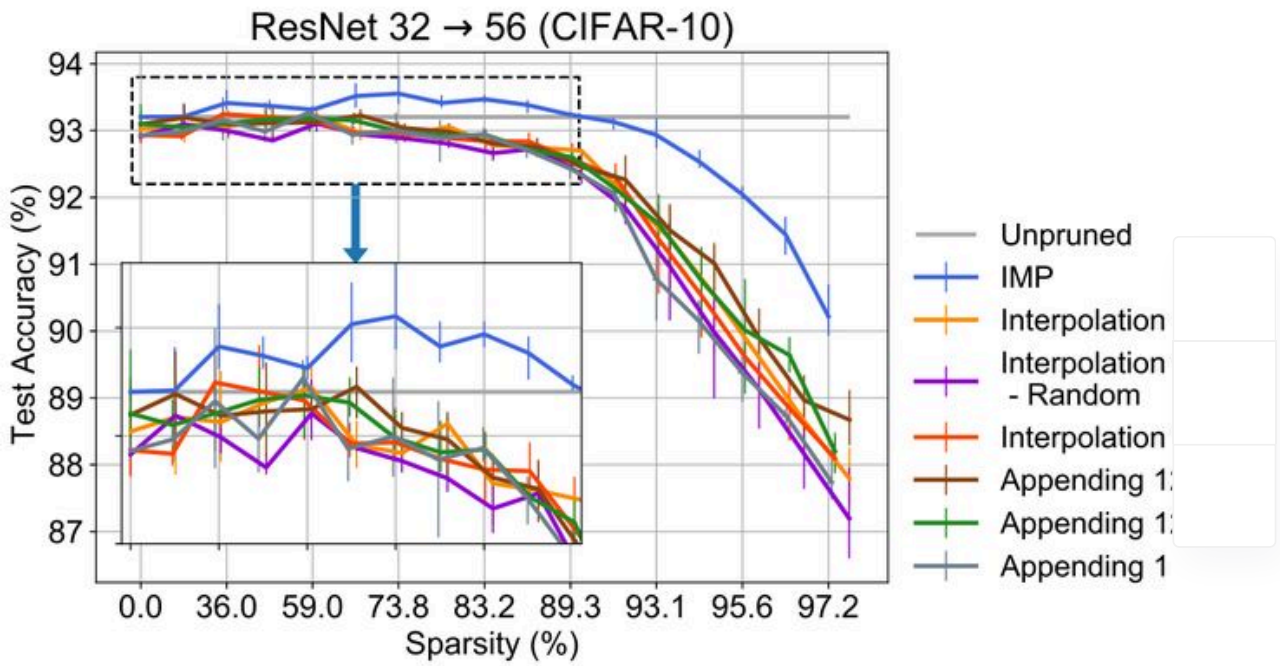
在本文中，作者使用“稀疏度”来表示网络中由于剪枝而产生的零元素的部分。因此，稀疏比越高，参数被修剪的越多。

3.1.3 ResNet-32 和 Resnet-56的实验结果

ResNet-32在每个阶段有5个剩余块，记为 $[B_0, \dots B_4]$ 。在复制这些块的时候有3个选择：复制所有4个块，前两个块或者只复制第一块。实验结果如下图15所示。 图例中的数字是复制块的索引。比如黄色的线：Interpolation 1234就代表采用下式复制块：

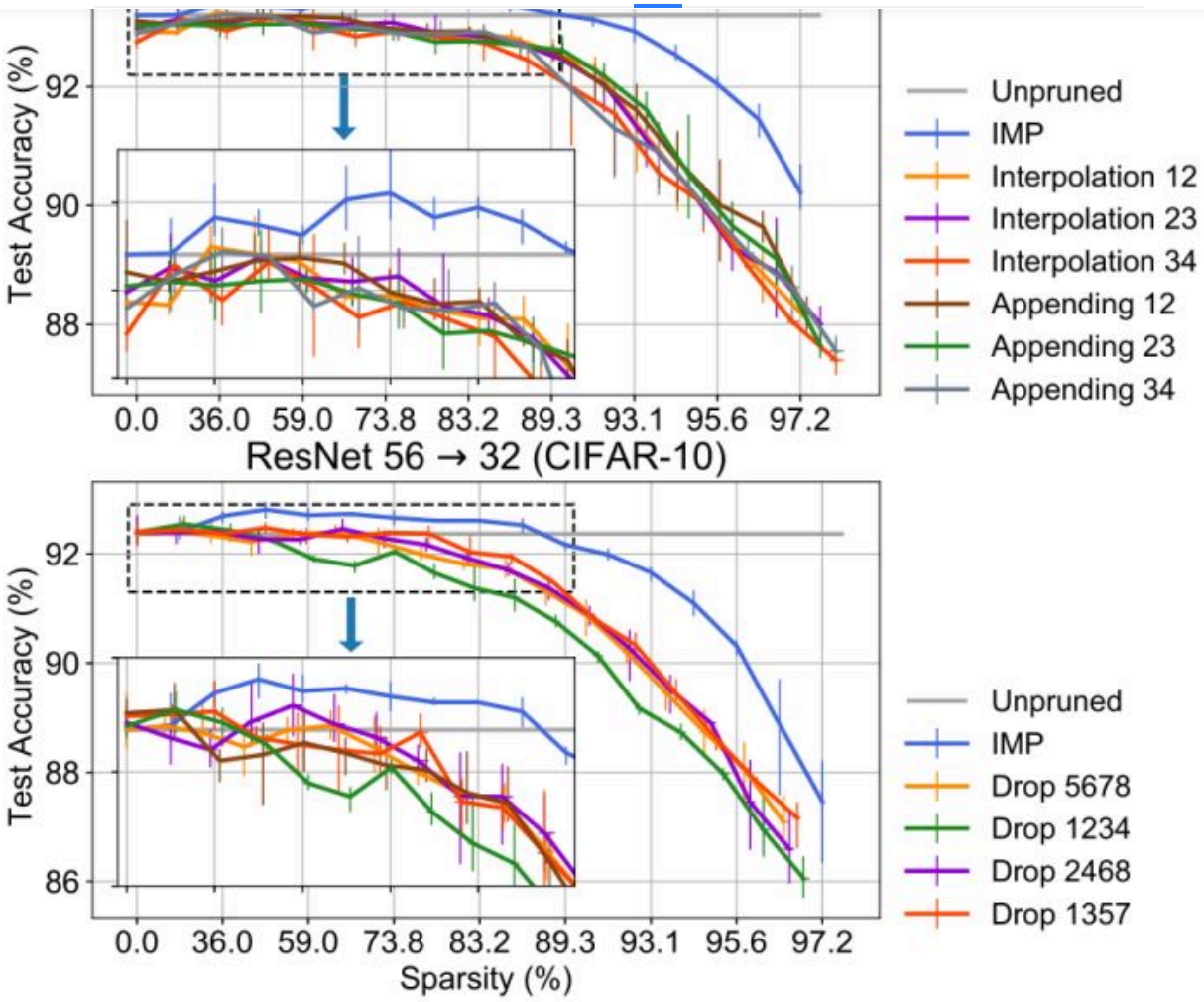
$$B_0^s \to B_1^s \to B_1^s \to B_2^s \to B_2^s \to B_3^s \to B_3^s \to B_4^s \to B_4^s$$

复制四个块比复制两个块 (Interpolation 12) 稍微好一点，而且比只复制第一个块 (Appending 1) 好得多。 稀疏比越高，间隙越明显。在较低的稀疏比下，这些选项之间的差异较小。



接下来关于ETTs的问题是：我们应该延伸 (stretching) 早期的块来获得更好的伸展效果，还是复制后期的单元？同样地，我们是否应该在压缩 (squeezing) winning ticket 时放弃较早的单元？这里的实验结果如下图16 所示。

对于延伸的情况，比如黄色的线：Interpolation 12 就代表只复制前2个块 B_1^s, B_2^s ，Interpolation 23 就代表只复制中间2个块 B_2^s, B_3^s ，Interpolation 34 就代表只复制最后个块 B_3^s, B_4^s ，我们发现： 复制较早的块比复制较晚的块具有高稀疏性的优势，但在低稀疏范围内差异不明显。

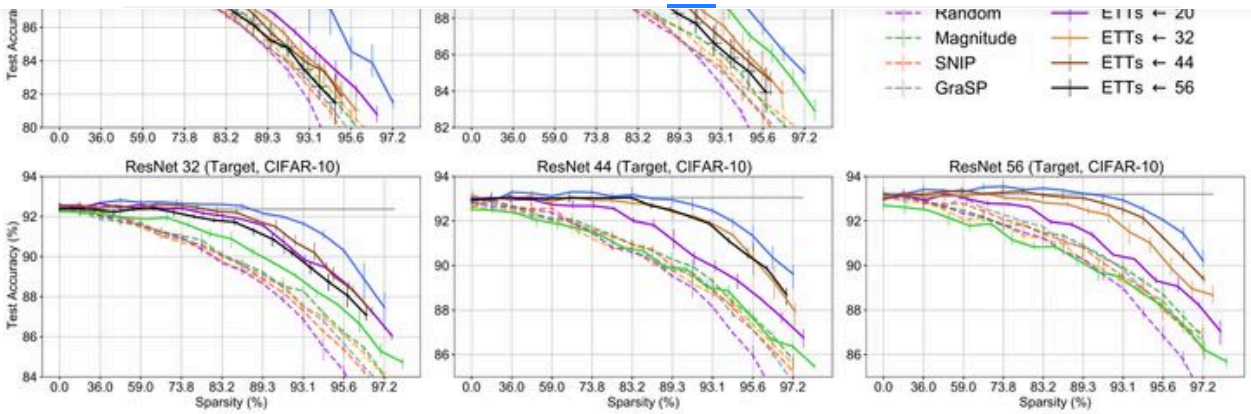


下图17是把ETTs在其他各种ResNet网络架构下的实验结果。 我们可以看到：

除了将最小的ResNet-14传输到大的目标网络之外，来自不同源网络的 ETTs 生成的 winning tickets 明显优于其他去(SNIP, GraSP等)。

当源网络和目标网络的差异较小时，ETTs 通常效果更好。 例如，当ResNet-44是目标网络时，使用ETTs方法从Re ResNet-56转换的 winning tickets 比从ResNet-14和ResNet-20转换的 winning tickets 的性能有显著的差距。再 ResNet-20是目标网络时，使用ETTs方法从ResNet-14转换的 winning tickets 的性能最好。

当目标模型转到ResNet-44或更高时，使用ETTs方法从ResNet-14转换的 winning tickets 的性能并不比初始化剪 这意味着我们的复制或下降仍然可能引入噪音，这在一个适当的范围内是可以接受的，但当复制或下降太多时，噪音 大。



下图18是在ImageNet数据集的实验结果。从中我们可以看到ETTs在将 winning tickets 转换为ResNet-26和ResNet-34时是有效的，产生的精度可与完整模型相媲美，远远高于其他的剪枝策略。对于ETTs方法来说，转换到ResNet-18似乎更具挑战性，但它相对于其他剪枝方法的优势仍然很明显。

Network	ResNet-18	ResNet-26	ResNet-34
Block Config	2-2-2-2	2-3-4-3	3-4-6-3
Full Model	69.96%	72.56%	73.77%
IMP	70.22%	72.94%	74.20%
Reinit	62.88%	66.97%	68.36%
Random	63.10%	67.07%	68.68%
Magnitude	64.96%	68.51%	69.74%
SNIP	62.23%	67.51%	69.35%
GraSP	62.85%	67.24%	69.23%
ETTs ← 18	-	71.29%	71.86%
ETTs ← 26	**68.17%**	-	**73.37%**
ETTs ← 34	68.08%	**72.47%**	-

小结

本文把彩票假设的泛化性能作了进一步研究，作者得到的结论是：**在一种网络架构上得到的 winning tickets 在其他网络架构上面具有很好的泛化性能。换句话讲，可以把一个网络得到的winning ticket直接迁移到另一个网络上面，而不需要重新做复杂的迭代剪枝操作。**

4 彩票假设的未来和开放性问题

彩票假设是一个令人兴奋且潜力巨大的视角，通过它我们可以更好地理解和改善深度神经网络。**如果我们能找到一种方法，从一开始**

分类：实战经验

1

说点什么...

请先 登录 再进行评论！

全部评论 (0)

暂无评论