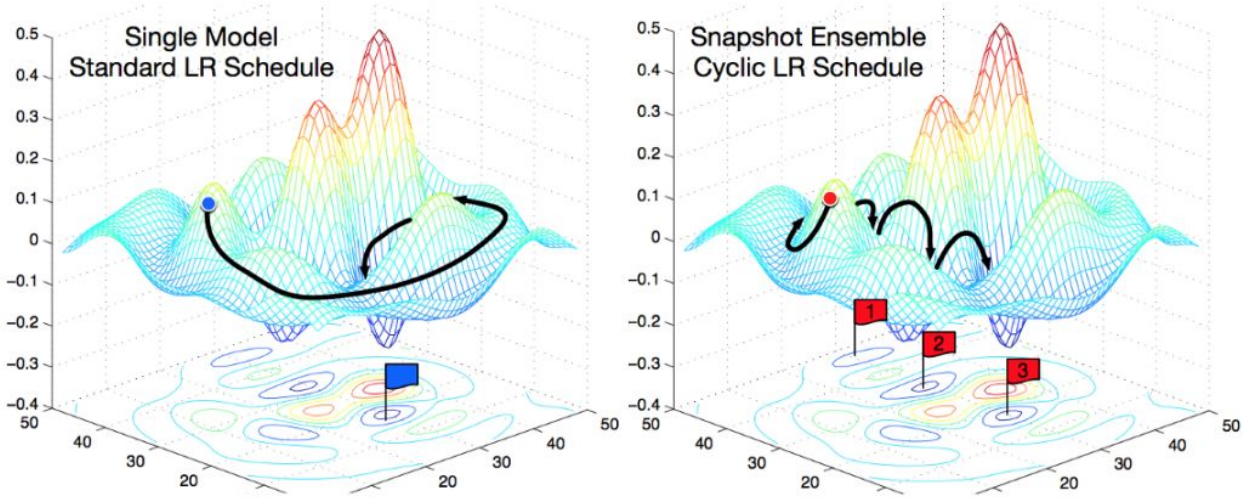
深度学习最新方法：随机加权平均击败了当前最先进的Snapshot Ensembling

作者：译者_林椿眄 来源：AI科技大本营

本文原地址：https://www.iikx.com/news/statistics/442.html

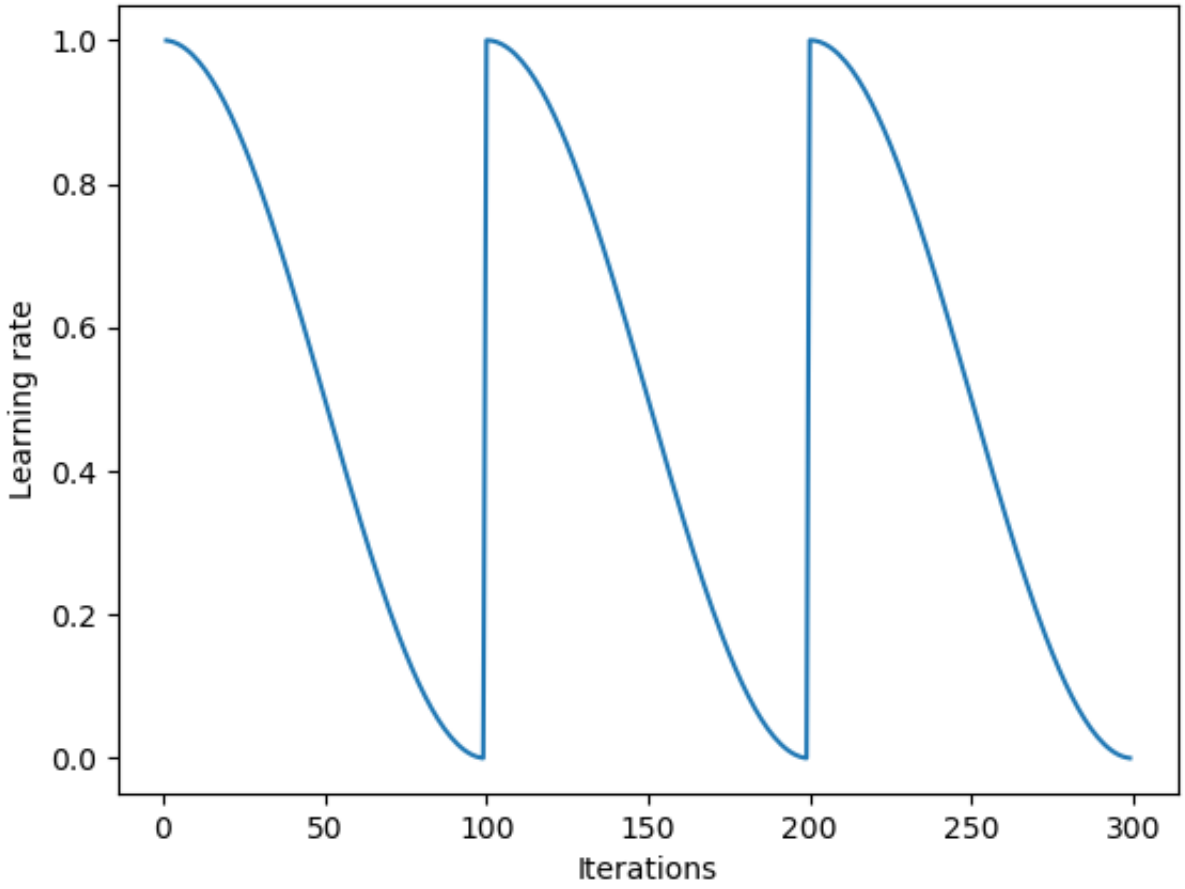
传统的神经网络集成方法

传统的集成方法是集成几种不同的模型，再用相同的输入对模型进行预测，然后使用某种平均方法来确定集成模型的最终预测。平均方法（averaging）可以采用简单的投票方法（voting），平均法或甚至使用集成模型中的一个模型去学习并预测输入的正确值或标签。岭回归（Ridge Regression）是一种特别的集成预测方式，也是被 Kaggle 竞赛冠军使用过的一种模型集成方法。



Snapshot集成：每次学习速率周期结束时保存相应的模型，然后在模型预测过程中同时使用已保存的所有模型进行预测。

当集成方法与深度学习相结合时，可以通过组合多个神经网络的预测来产生最终的预测结果。通常，集成不同结构的神经网络会得到一个性能不错的集成模型，因为每种模型可能在不同的训练样本上犯错，因此这样的集成方法能够最大化地提升模型的最终性能。



Snapshot集成：使用带退火策略的循环学习速率

但是，你也可以集成相同结构的神经网络模型，这种集成方式也可以获得出人意料的好结果。在snapshot ensembling论文中，基于这种集成方式，作者提出了一个非常棒的训练技巧。在训练两个相同的神经网络时，采用 weight snapshot策略，并在训练结束后创建一个具有相同结构、带不同权重的集成模型。实验证明这种方式得到的集成模型可以提高最终的测试性能，而且这也是一种非常简单的方法，你只需每次训练一个模型，大大减少计算的时间成本。

如果在训练中你还没有采用循环学习率策略的话，那么你必须要学会使用它，因为它是当前最先进的训练技巧，非常易于实现，计算量并不大，也几乎不需要额外的成本就可以收获显著的效果。

以上，我介绍的所有例子都是基于模型空间的集成方法，即通过结合几个或几种模型，集成单个模型的预测来产生最终的预测结果。

而在这篇文章将要讨论的论文中，作者提出一种基于权重空间的新集成方法。这种方法通过在不同训练阶段组合相同网络的权重来集成模型，然后使用这种组合权重的集成模型进行预测。这种方法有两大优点：

训练结束后我们只会得到一个具有组合权重的集成模型，这将加快了后续模型预测的速度当使用组合权重时，我们在训练结束后会得到一个模型，这个模型能够加速后续预测阶段的速度。

实验结果表明，这种组合权重的集成方法击败了当前最先进的 snapshot ensembling 方法

下面，我们将具体了解下它是如何工作的。但在这之前，我们需要了解一些关于损失平面 (loss surface)和泛化问题 (generalizable solution)的知识。

▌权重空间中的解决方案

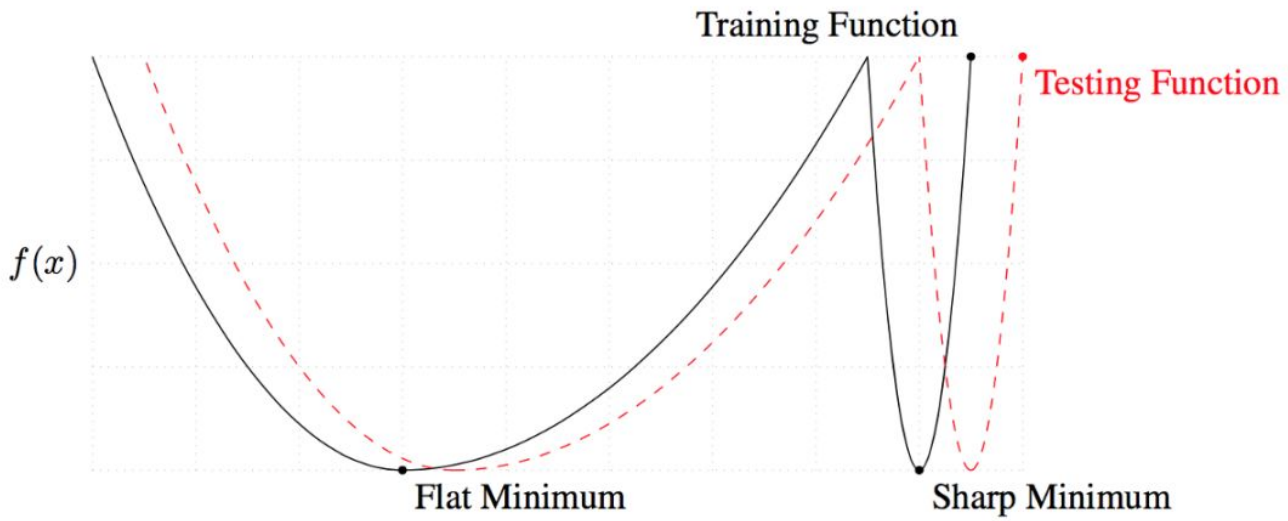
第一个重要的观点是一个训练好的网络其实就是多维权重空间中的一个点。对于给定的模型结构，网络权重的每个不同组合都会生成一个单独的模型。由于任何的模型结构都有无限多种的权重组合，因此这将会有无限多的解决方案。训练神经网络的目标是找到一个特定的、指向权重空间的解决方案，它能够在训练和测试数据集上最小化损失函数值。

在训练期间，通过改变权重，训练算法将改变网络结构并在权重空间中探索解决方案。梯度下降算法在损失平面上传播，而平面的高程由损失函数的值给出。

▌局部最优解和全局最优解

可视化并理解多维的权重空间的几何特性是非常困难的。同时，这也是非常重要的，因为本质上，在训练期间随机梯度下降算法是在这个高度的多维空间中穿过损失平面，并试图找到一个好的解决方案，即在损失平面上探索一个损失值最低的“点”。众所周知，这样的损失平面会存在许多的局部最优解，但并不是所有的局部最优解都会是全局的最佳解决方案。

Hinton曾说过：“处理 14 维空间中的超平面时，你可以想象一个 3
维空间，并大声地告诉自己“这是一个十四维空间”。每个人都可以这样做。”



局部最优解和全局最优解。在训练和测试过程中，平坦区域的最小值会产生类似的损失，但是减小损失将在训练和测试过程中产生非常不同的结果。换句话说，全局最小值比局部的最小值更具有普遍性。

一个可以区分解决方案好坏的衡量标准是它的平坦度。因为模型在训练数据集和测试数据集上会产生相似但不完全相同的损失平面。你可以想象一下，测试的损失平面会比训练的损失平面稍微

偏移一点。对于一个局部最优解，在测试期间，由于这种转变，损失较低的点可能会产生很大的损失值，这意味着这种局部最优解并不具有很好的通用性，即在训练时损失低，而测试时损失却很大。另一方面，对于全局最优解而言，这种转变将导致训练和测试损失彼此接近。

以上，我解释了局部最优解和全局最优解之间的区别，因为本文重点介绍的新方法将涉及到全局最优解。

▌Snapshot Ensembling (快照集成)

训练刚开始，SGD在权重空间会产生大幅的跳跃。随后，由于余弦退火策略使得学习率逐渐变小，SGD将会收敛到某个局部最优解，并且通过 snapshot ensembling的方式将该模型添加到集合中，实现模型的集成。然后，学习率将被重新设置为较大的值，并且在模型收敛到一些不同的局部最优解之前，SGD将再次发生大幅的跳跃。

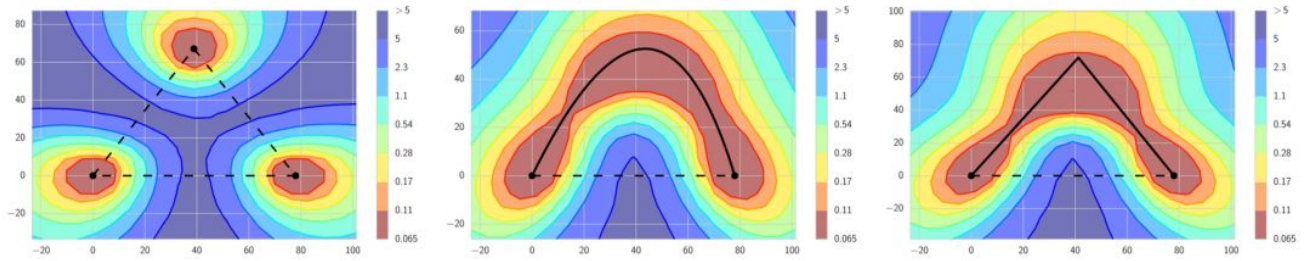
Snapshot ensembling方法的周期长度为 20 到 40 次迭代。长时间的循环学习速率能够在权重空间中找到尽可能不相同的模型。如果模型太相似，那么在集成模型中单独网络的预测将会过于接近，这将导致集成模型的优势变得微不足道。

Snapshot
ensembling方法所取得的效果非常好，它能够大大提高模型性能，但相比之下快速几何集成(Fast Geometric Ensembling)方法的效果更佳。

▌Fast Geometric Ensembling (FGE)(快速几何集成)

Fast Geometric Ensembling (FGE) 与 Snapshot
Ensembling方法非常相似，但其具有一些显著的特性。它使用线性分段的循环学习速率，来取代 snapshot ensembling中的余弦。其次，FGE的周期长度要比 snapshot
ensembling短得多，每个周期只有 2 到 4 次迭代过程。

直观上，我们可能会认为短周期是错误的，因为每个周期结束时的模型将彼此接近，再将它们组合起来不会带来任何好处。然而，正如作者所发现的那样，因为在完全不同的模型之间，存在低损失的连接路径，所以可以沿着这些路径以小的步长行进，将遇到的模型集成在一起并获得好的结果。因此，与 snapshot ensembling方法相比，FGE展示了其改进之处，且能够以更少的步数得到我们想要的模型，这也使得训练的速度更快。

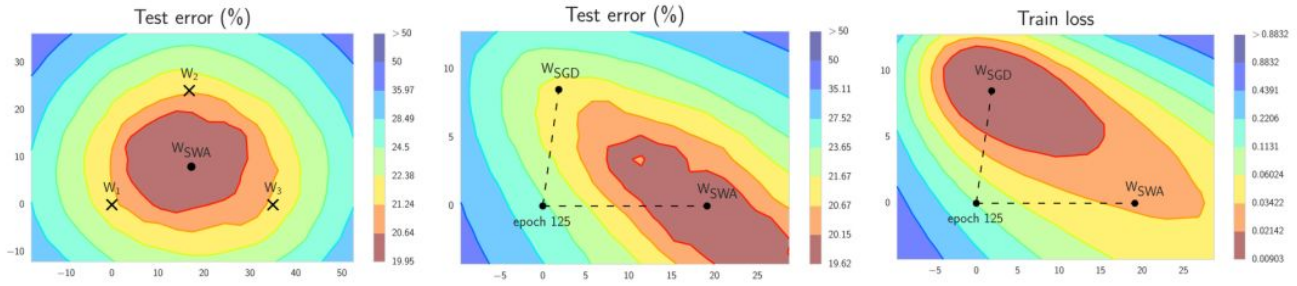


左图：传统的直觉认为，局部最小值是被高损失值的区域所分隔。如果我们沿着局部最小值的路径探索，所得到情况也是如此。中间和右图：确实在局部最小值之间存在一些较低损失值的路径。FGE会沿这些路径生成一个集成模型。

为了充分利用 snapshot ensembling或 FGE方法，我们需要存储多个训练模型，然后对每个模型进行预测并平均最终的预测结果。因此，为了获得更好的集成性能，需要付出更多的计算量，这正是“ 没有免费的午餐 ”法则的体现，同时也是这篇“ 随机加权平均 ”论文提出的动机。

▌随机加权平均(SWA)

随机加权平均 (SWA)与 FGE方法非常接近，但其计算损失很小。SWA可以应用于任何的模型结构和数据集，并在这些数据集中都显示出良好的结果。这篇论文表明 SWA会更趋于一个全局最小值，它的优点正如我在上面所讨论的。SWA并不是我们传统意义上所理解的集成方法。在训练结束时，你会得到一个模型，这个集成模型的性能将更优于 snapshot ensembling 和 FGE。



左图：W1，W2和W3代表3个独立训练的网络，Wswa是它们的平均值。中间图：与SGD相比，Wswa在测试集上表现出更出色的性能。右图：请注意，虽然Wswa在训练过程表现出更差的损失，但它的泛化性和通用性更好。

SWA的观点是来自经验观察，即每个学习速率周期结束时的局部最小值都倾向于在损失平面上损失值低的区域边界处累积 (如左图中点 W1，W2和 W3所位于红色区域的边界处) 。通过平均化这种点的损失值，可以得到具有更低损失值、泛化性和通用性更好的全局最优解(如上左图的 Wswa)。

下面是 Wswa的工作原理。你只需要两个单独的模型，而不需要用很多模型进行集成：

第一个模型用来存储模型权重的平均值 (如公式中的 w_swa)。这将在训练结束后得到最终的模型，并用于预测。

第二个模型用来穿过权重空间 (如公式中的 w)，并使用循环学习率进行探索。

$$w_{\mathrm{SWA}} \leftarrow \frac{w_{\mathrm{SWA}} \cdot n_{\mathrm{models}} + w}{n_{\mathrm{models}} + 1,},$$

随机权重平均的权重更新方程

在每个学习速率周期结束时，将使用第二个模型的当前权重，通过在旧的平均权重和第二个模型的新权重集合之间进行加权平均值来更新模型的平均权重 (公式如左图所示)。按照这种方法，你只需要训练一个模型，并且在训练期间将两个模型存储在内存中。在预测阶段，你只需要那个具有平均权重的模型，并对其进行预测，这比使用上述那些需要使用多个模型来进行预测的集成方法要快得多。

结语

本文的作者在 PyTorch上开源了这篇论文的实现。此外，在 awesome fast.ai library中也有SWA的实现，每个人都可以使用它。