摘要

我们通过新颖的插值函数替换深度神经网络的输出层，通常是 softmax 函数。我们为这种新架构提出了端到端的培训和测试算法。与具有 softmax 函数作为输出激活的经典神经网络相比，具有插值函数作为输出激活的代理结合了深度和流形学习的优点。新框架具有以下主要优点：首先，它更适用于训练数据不足的情况。其次，它显着提高了各种网络的泛化精度。该算法在 PyTorch 中实现，代码可在 https://github.com/ BaoWangMath / DNN-DataDependentActivation 获得。

徐汤柯

1 介绍

广义性是深层学习的关键，为了提高深层神经网络（DNN）[3，14]的训练和泛化精度，人们做了很多努力。诸如 VGG 网络[28]、深剩余网络(ResNets)[12,13]以及最近出现的 DenseNets[16]和许多其它[6]的网络体系结构的进展，以及强大的硬件使得具有良好泛化能力的非常深网络的训练成为可能。诸如{Dropout 技术}和{maxout}[15,30,10]之类的有效正则化技术以及数据扩充方法[19,28,32]也明确地改进了 DNN 的泛化。

参考链接

dropout 技术：https://www.cnblogs.com/santian/p/5457412.html

maxout 网络：https://blog.csdn.net/zhufenghao/article/details/52527047

神经网络的一个重要组成部分是激活函数。诸如整流线性单元（ReLU）[8]等激活函数设计的改进已经导致计算机视觉任务性能的巨大改进[23，19]。最近，对数据进行自适应训练的激活函数，例如自适应分段线性单元(APLU)[1]和参数整流线性单元(PReLU)[11]已经导致 DNN 的性能的进一步改进。对于输出激活，支持向量机（SVM）也已成功用于代替 softmax[29]。尽管用 softmax 或 SVM 作为输出激活来训练 DNN 在许多任务中是有效的，但是通过基于训练和测试数据对输出进行插值来考虑数据的流形结构的替代激活可以提高网络的性能。特别地，ResNets 可以表示为求解连续极限[21，5]中的一类传输方程的控制问题。传输理论认为，通过采用从初始值中插值终端值的插值函数，与特定选择的值相比，可以极大地简化控制问题。这进一步表明，输出层的固定和数据不可知激活可能是次优的。

为此，基于流形学习的思想，本文提出了一种新的 DNN 输出层——加权非局部拉普拉斯层(WNLL)。所得到的 DNN 具有更好的泛化能力，并且对于少量训练实例的问题具有更强的{鲁棒性}。在 CIFAR10/CIFAR100 上，我们在各种网络上的测试误差平均减少了 30%/20%。这些包括 VGGS、RESNET 和预激活的 RESNET。当在 CIFAR 的随机子集上用少量的训练示例训练模型时，性能提升甚至更加显著。我们还提出了一种通过辅助网络训练 WNLL 层的有效算法。从博弈论和传输方程终端值问题的角度给出了 WNLL 层的理论动因。鲁棒性：https://baike.baidu.com/item/%E9%B2%81%E6%A3%92%E6%80%A7/832302?fr=aladdin

本文的结构如下：第二部分介绍了在 DNN 中使用 WNLL 插值函数的动机和实践。在第 2.2 节中，我们详细解释了以 WNLL 作为输出层的 DNN 训练和测试算法。第三节从传输方程和博弈论的终值问题的角度出发，对使用插值函数作为输出层进行深入分析。第四节通过大量的数值例子说明了本方法的有效性。

郑宇骁

2 网络结构

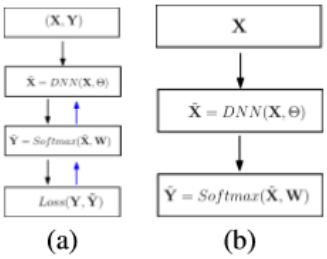


Figure 1: Training (a) and testing (b) procedures of DNNs with softmax as output activation layer.

图 1(a) 和图 1(b) 包含了使用 Softmax 函数作为输出层的 DNN 网络的训练和测试过程的一个粗粒度表示。在训练的第 k 次迭代中，给定一个小的训练集 (X, Y)，我们进行以下操作：

- 前向传播：将 X 转化为网络中的深度特征（卷积层的输出、非线性量和其它特征），然后以 Softmax 作为激活函数获取估计的目标量 $\widetilde{Y}$:

$\widetilde{Y} = \text{Softmax}(\text{DNN}(X, \Theta^{k-1}), W^{k-1})$

接下来计算 Y 与 $\widetilde{Y}$ 之间的损失（例如交叉熵）：$L = Loss(Y, \widetilde{Y})$.

- 后向传播：通过梯度下降（以学习率 γ）更新权重 (Θ^{k-1}, W^{k-1}):

$W^k = W^{k-1} - \gamma \frac{\partial L}{\partial \widetilde{Y}} \frac{\partial \widetilde{Y}}{\partial W}$

$\Theta^k = \Theta^{k-1} - \gamma \frac{\partial L}{\partial \widetilde{Y}} \frac{\partial \widetilde{Y}}{\partial \widetilde{X}} \frac{\partial \widetilde{X}}{\partial \Theta}$

当模型达到理想状况时，对于测试数据 X，估计值为：

$\widetilde{Y} = \text{Softmax}(\text{DNN}(X, \Theta), W)$

为了叙述方便，我们依然用 X, Θ 和 W 指代测试集和理想化的权重。本质上，Softmax 层以线性模型拟合深度特征 $\tilde{X}$，从而忽略了 $\tilde{X}$ 的多种潜在结构。而我们接下来要介绍的加权非局部拉普拉斯 (WNLL) 插值函数则弥补了这一不足。此外，WNLL 插值基于谐波扩展的特性使其避免了高维空间插值中出现的维度灾难的问题。

朱志儒

2.1 流形插值—一种调和扩展法

设$\mathrm{X} = \{x_1, x_2, \dots, x_n\}$为高维流形$\mathrm{M} \subset R^d$中点的集合，$X^{te} = \{x_1^{te}, x_2^{te}, \dots, x_m^{te}\}$为 X 的一个子集。假设我们在$X^{te}$上定义一个（可能的向量值）标记函数 g(X)，我们想要插值一个函数 u，它定义在整个流形上，并且可以用来标记整个数据集 X。在高维空间中利用基函数进行插值将受限于维度，然而，调和扩展法是找到合适插值函数的一种自然和优雅的方法，其通过最小化狄利克雷能量(Dirichlet energy)函数来定义：

$$\mathcal{E}(\mathrm{u}) = \frac{1}{2}\sum_{x,y\in X} w(x,y)\big(u(x) - u(y)\big)^2 \tag{1}$$

边界条件：

$$\mathrm{u}(x) = g(x), x \in X^{te}$$

其中 w(x, y)是一个加权函数，通常选择高斯分布：$\mathrm{w(x,y)} = \exp\left(-\frac{\|x-y\|^2}{\sigma^2}\right)$，其中σ是尺度参数。对于式(1)的欧拉-拉格朗日方程：

$$\begin{cases} \sum_{y\in X}\big(w(x,y) + w(y,x)\big)\big(u(x) - u(y)\big) = 0 & x \in X/X^{te} \\ u(x) = g(x) & x \in X^{te} \end{cases} \tag{2}$$

通过求解线性方程组(2)，我们得到未标记数据$\mathrm{x} \in \mathrm{X}/X^{te}$的插值标记 u(x)，当标记数据

量很小，例如$|X^{te}| \ll |X/X^{te}|$，该插值将变得无效。解决这一问题有两种解决方法：一种是用 p-拉普拉斯算子代替方程(1)中的 2-拉普拉斯算子，另一种是增加欧拉-拉格朗日方程中标记数据的权重，其给出以下加权非局部拉普拉斯(WNLL)插值函数：

$$\begin{cases} \sum_{y\in X}\big(w(x,y)+w(y,x)\big)\big(u(x)-u(y)\big)+\left(\frac{|X|}{|X^{te}|}-1\right)\sum_{y\in X^{te}} w(y,x)\big(u(x)-u(y)\big)=0 & x\in X/X^{te} \\ u(x)=g(x) & x\in X^{te} \end{cases}$$

为简化符号，我们将上述方程的解 u(x)命名为WNLL(X,X^{te},Y^{te})。对于分类任务，g(x)为示例 x 的一个 one-hot 标记。为确保 WNLL 的准确性，标记数据应该覆盖 X 中数据的所有类别，我们在定理中给出了一个必要条件。

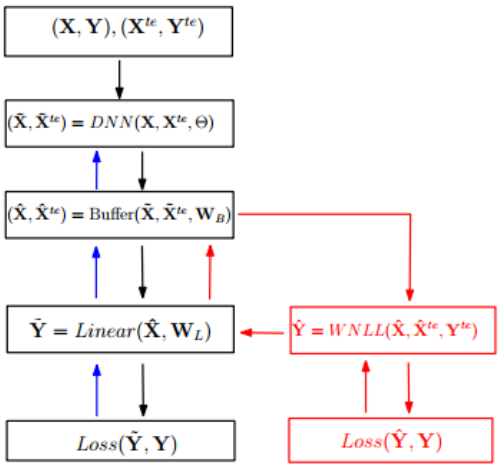
定理：假设我们有一个由 N 个数据类统一形成的数据池，每个类的实例数量足够的大。如果我们希望所有类别的数据至少采样一次，那么平均至少需要从数据池中采样$N\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}\right)$个数据。在这种情况下，对每个类所期望的采样数据的数量为$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{N}$。

黄家熙

2.2 WNLL Activated DNNs and Algorithms

训练和测试由 WNLL 激活的 DNN 时,我们需要保留一部分,以对新数据的标签进行插值.我们将命名为 preserved template.

直接将 softmax 替换成 WNLL 会在反向传播时有困难--难以计算,因为 WNLL 是一个复杂的隐函数.所以为了能训练网络,我们通过一个辅助网络引入了代理机制.在网络最后,加入一个 buffer block(一个 ReLu 激活的全连接层),然后接两个平行的 WNLL 和 linear 层,辅助神经网络可以在线性 DNN 和 WNLL-DNN 间交替训练.



linear DNN 训练过程：

$$\tilde{\mathbf{Y}}=\mathrm{Linear}(\mathrm{Buffer}(\mathrm{DNN}(\mathbf{X},\Theta^{k-1}),\mathbf{W}_B^{k-1}),\mathbf{W}_L^{k-1}).$$

$$\mathbf{W}_L^k = \mathbf{W}_L^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \mathbf{W}_L}, \quad \mathbf{W}_B^k = \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B},$$

$$\Theta^k = \Theta^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \tilde{\mathbf{X}}} \cdot \frac{\partial \tilde{\mathbf{X}}}{\partial \Theta}.$$

WNLL DNN 训练过程：

$$\hat{\mathbf{Y}} = \text{WNLL}(\text{Buffer}(\text{DNN}(\mathbf{X}, \Theta^{k-1}), \mathbf{W}_B^{k-1}), \hat{\mathbf{X}}^{\text{te}}, \mathbf{Y}^{\text{te}}).$$

$$\mathbf{W}_B^k = \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{WNLL}}}{\partial \hat{\mathbf{Y}}} \cdot \frac{\partial \hat{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B} \approx \mathbf{W}_B^{k-1} - \gamma \frac{\partial \mathcal{L}^{\text{Linear}}}{\partial \tilde{\mathbf{Y}}} \cdot \frac{\partial \tilde{\mathbf{Y}}}{\partial \hat{\mathbf{X}}} \cdot \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{W}_B}.$$

在训练 WNLL DNN 时,用 linear DNN 做了一个近似.

背后的思想在于 WNLL 是隐式谐波函数,而线性函数是最简单的非平凡显式谐波函数.

从经验上看,这种简单的近似处理在训练汇中效果很好.

The reason why we freeze the network in the DNN block is mainly due to stability concerns. (这句不是很懂没翻译)

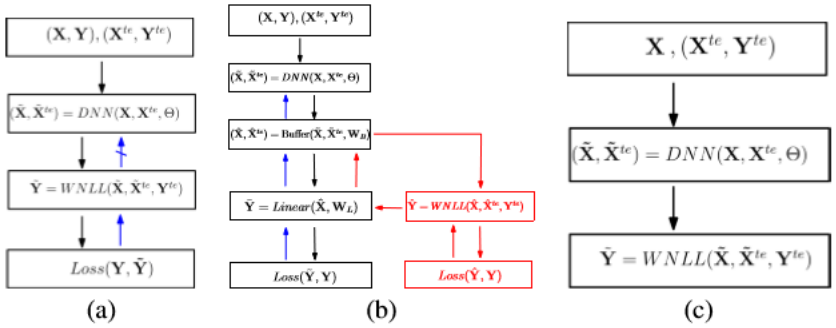


Figure 2: Training and testing procedure of the deep neural nets with WNLL as the last activation layer.(a): Direct replacement of the softmax by WNLL, (b): An alternating training procedure. (c): Testing.

黄宏斌

上述交替方案是一种贪心的算法。在训练过程中，WNLL 的激活起着两个作用:一方面，线性激活和WNLL激活的交替使神经网络能够学习线性分类和基于WNLL的流形插值都适用的特征。另一方面，在缺乏足够的训练数据的情况下，神经网络的训练往往会陷入局部极小值，不能很好地推广新数据。利用 WNLL 插值对训练后的次优权值进行扰动，得到具有较好的通用性的局部极小值。在测试时，我们将线性分类器从神经网络中移除，并将 DNN 块与 WNLL 一起用于预测新数据(图 2(c))。使用 WNLL 而不是线性层的原因是 WNLL 优于线性分类器，这种优势在应用于深度特征时得到了保留(将在第 4 节中展示)，而且 WNLL 在测试时利用了学到的 DNNs 和保留的模板，对于输入数据的扰动似乎更稳定。我们分别总结了算法 1 和算法 2 中 WNLL 激活神经网络的训练和测试过程。在每一轮的交替过程中，首先利用整个训练

集(X;Y)对神经网络进行线性激活训练。我们从训练集中随机分离出一个模板，例如整个数据的一半，用来在训练 WNLL 激活的 DNNs 时进行 WNLL 插值。在实践中，对于训练和测试，当整个数据集太大时，我们对模板和插值点都使用小批量。最终的预测标签为所有模板 mini 批次的内插结果的多数者。

备注 1：在算法 1 中，WNLL 插值也以小批处理的方式进行(如内部迭代所示)。根据我们的实验，这并没有显著降低插补精度。

Algorithm 1 DNNs with WNLL as Output Activation: Training Procedure.

Input: Training set: (data, label) pairs $(\mathbf{X}, \mathbf{Y})$.
Output: An optimized DNNs with WNLL as output activation, denoted as $\mathrm{DNN}_{\mathrm{WNLL}}$.
for iter $= 1, \ldots, N$ (where N is the number of alternating steps.) **do**
  //Train the left branch: DNNs with linear activation.
  Train DNN + Linear blocks, and denote the learned model as $\mathrm{DNN}_{\mathrm{Linear}}$.
  //Train the right branch: DNNs with WNLL activation.
  Split $(\mathbf{X}, \mathbf{Y})$ into training data and template, i.e., $(\mathbf{X}, \mathbf{Y}) = (\mathbf{X}^{\mathrm{tr}}, \mathbf{Y}^{\mathrm{tr}}) \bigcup (\mathbf{X}^{\mathrm{te}}, \mathbf{Y}^{\mathrm{te}})$.
  Partition the training data into M mini-batches, i.e., $(\mathbf{X}^{\mathrm{tr}}, \mathbf{Y}^{\mathrm{tr}}) = \bigcup_{i=1}^{M} (\mathbf{X}_i^{\mathrm{tr}}, \mathbf{Y}_i^{\mathrm{tr}})$.
  for $i = 1, 2, \cdots, M$ **do**
    Transform $\mathbf{X}_i^{\mathrm{tr}} \bigcup \mathbf{X}^{\mathrm{te}}$ by $\mathrm{DNN}_{\mathrm{Linear}}$, i.e., $\tilde{\mathbf{X}}^{\mathrm{tr}} \bigcup \tilde{\mathbf{X}}^{\mathrm{te}} = \mathrm{DNN}_{\mathrm{Linear}}(\mathbf{X}_i^{\mathrm{tr}} \bigcup \mathbf{X}^{\mathrm{te}})$.
    Apply WNLL (Eq.(3)) on $\{\tilde{\mathbf{X}}^{\mathrm{tr}} \bigcup \tilde{\mathbf{X}}^{\mathrm{te}}, \mathbf{Y}^{\mathrm{te}}\}$ to interpolate label $\tilde{\mathbf{Y}}^{\mathrm{tr}}$.
    Backpropagate the error between $\mathbf{Y}^{\mathrm{tr}}$ and $\tilde{\mathbf{Y}}^{\mathrm{tr}}$ via Eq.(4) to update $\mathbf{W}_B$ only.

Algorithm 2 DNNs with WNLL as Output Activation: Testing Procedure.

Input: Testing data $\mathbf{X}$, template $(\mathbf{X}^{\mathrm{te}}, \mathbf{Y}^{\mathrm{te}})$. Optimized model $\mathrm{DNN}_{\mathrm{WNLL}}$.
Output: Predicted label $\tilde{\mathbf{Y}}$ for $\mathbf{X}$.
Apply the DNN block of $\mathrm{DNN}_{\mathrm{WNLL}}$ to $\mathbf{X} \bigcup \mathbf{X}^{\mathrm{te}}$ to get the representation $\tilde{\mathbf{X}} \bigcup \tilde{\mathbf{X}}^{\mathrm{te}}$.
Apply WNLL (Eq.(3)) on $\{\tilde{\mathbf{X}} \bigcup \tilde{\mathbf{X}}^{\mathrm{te}}, \mathbf{Y}^{\mathrm{te}}\}$ to interpolate label $\tilde{\mathbf{Y}}$.

余捷

3 理论解释

在训练 WNLL 激活的 DNNs 中，辅助网络中的两个输出激活函数在某种意义上每个都在以最小化其自身的目标，其中在平衡状态下，神经网络可以为线性和基于插值的激活学习更好的特征。这种特点类似于生成性对抗网（GAN）[9]。我们模型的另一种解释如下：如[21]中所述，在连续极限中，对于一个传输方程 ResNet 可以用来建模成以下控制问题：

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} + \mathbf{v}(\mathbf{x},t) \cdot \nabla u(\mathbf{x},t) = 0 & \mathbf{x} \in \mathbf{X}, t \geq 0 \\ u(\mathbf{x},1) = f(\mathbf{x}) & \mathbf{x} \in \mathbf{X}. \end{cases}$$

这里 u（.，0）是 ResNet 的连续 version 的输入，它将训练数据映射到相应的标签。 f（.）是类似于 ResNet 中输出激活函数的终值，它将深度特征映射到预测标签。训练 ResNet 相当于调整 v（.，t），即权重的连续 version，s.t。预测标签 f（.）与训练数据的标签匹配。如果 f（.）是 u（.，0）的谐波扩展，则相应的权重 v（x，t）将接近于零。这导致更简单的模型，并且可以从模型选择的角度更好地概括。

杨奕洁

4 数值结果

为了验证我们所提出的框架的分类精度、效率和健壮性，我们在 CIFAR10，CIFAAR100，MNIST 和 SVHN 数据集上分别测试了新架构和算法。所有在 CIFAR 数据集上进行的实验我们都应用了在 CIFAR 数据集上广泛使用的数据增强方法。对于 MNIST 和 SVHN 数据集，我们直接使用原始数据不进行任何增强。我们的算法的实现基于 PyTorch，所有计算都在一台安装了单个 Nvidia Titan Xp 显卡的计算机上完成。

在深入研究深度神经网络在使用不同激活函数时的表现前，我们首先对比了使用 softmax 激活函数的全局加权拉普拉斯层(WNLL)在处理来自不同数据集的原始图片的表现。训练集被用于训练 softmax 模型并分别使用 softmax 层和 WNLL 生成测试集上的插值标签。表 1 列出了 WNLL 和 softmax 在三种数据集上的分类准确度。为了加快计算速度，在 WNLL 层插值的计算过程中，我们只使用 15 个最近相邻点来确保权值矩阵的稀疏性，并使用第 8 个相邻点的距离归一化权值矩阵。我们使用近似最近相邻点算法（ANN）来找出这些最近相邻点。

表一：WNLL 和 softmax 在不同数据集下的分类准确性

Dataset	CIFAR10	MNIST	SVHN
softmax	39.91%	92.65%	24.66%
WNLL	40.73%	97.74%	56.17%

在下面的深度学习过程中，我们交替进行两种学习方式，即算法 1 中的 N=2。在线性激活阶段(阶段 1)，神经网络迭代训练 400 次，在 WNLL 阶段，网络迭代训练 5 次。第一轮训练初始学习率为 0.05，训练线性激活网络时每迭代 50 次学习率减半，进入 WNLL 训练阶段时，学习率降为 0.0005。在 CIFAR 和 SVHN 实验中我们使用了和文献[12,17]一样的牛顿动量随机梯度下降优化法和权值衰减法。在第二轮训练中学习率被设置为第一轮相应阶段的学习率的五分之一。Softmax/线性激活训练阶段分批大小为 128，而在 WNLL 阶段这一数字是 2000。为了使比较更为公平，我们采用和训练 WNLL 激活层一样的优化方法迭代训练了另一个使用 softmax 激活函数的普通神经网络(vanilla DNN)810 次。所有使用 WNLL 激活层对测试集进行的错误预测都将作为 WNLL 方法的错误结果上报。在本节的剩余内容中，我们展现了新架构在解决训练数据过少问题以及提升神经网络在 CIFAR10/CIFAR100 数据集上的泛化精度方面的优异表现。SVHN 数据集上的数值结果将在附录中给出。

郑映雪

4.1 解决训练数据不足的问题

当我们没有足够的训练数据时，随着网络的加深，泛化精度通常会降低，如图 3 所示。WNLL 激励的 DNN 具有优良的参数正则化和对不良局部最小值的摄动，它能够克服这种退化的情况。左侧和右侧面板绘制了 CIFAR10 训练集中的前 1000 条和 10000 条数据用于训练原版 DNN 和 WNLL DNN 的情况。如图 3 所示，通过使用 WNLL 激励，随着网络变深，泛化误差率一直衰减，这与原版 DNN 的退化相反。在我们的测试体系中，原版 DNN 和 WNLL DNN 之间的泛化精度相差可以达到 10%。

（下面一段建议对照着图看）

图 4 描绘了训练期间泛化精度的演变。我们计算每个周期的测试精度。图（a）和（b）分别绘制了 softmax 和 WNLL 激励的 ResNet50 的测试精度，(1-400 和 406-805 时期对应于线性激励）分别只有前 1000 个例子作为 CIFAR10 的训练数据。图（c）和（d）是使用预先

激活的 ResNet50 的 10000 个训练实例对应的图，在大约 300 个周期之后，原版 DNN 的准确度开始停滞，并且不能再提高。相比之下，WNLL 的测试精度在第一阶段的第 2 小段开始时突然跳动，在第二阶段，即使最初精确度降低，准确度仍继续攀升并且最终超过第一阶段中第 2 小段的 WNLL 激活的准确度。在 400 和 800 个周期处，准确度的跳跃是由于需要进行测试集上的预测，从而从线性激励切换到 WNLL 激励。交替返回到 softmax 时的初期衰减，一方面是由于最后一层 WL 没有针对深度特征$\tilde{X}$进行调整，一方面是由于测试集的预测是由 softmax 而不是 WNLL 进行的。然而，通过 WNLL 激励的摄动很快导致精度增加，并且超过前一次阶段中的线性阶段部分。

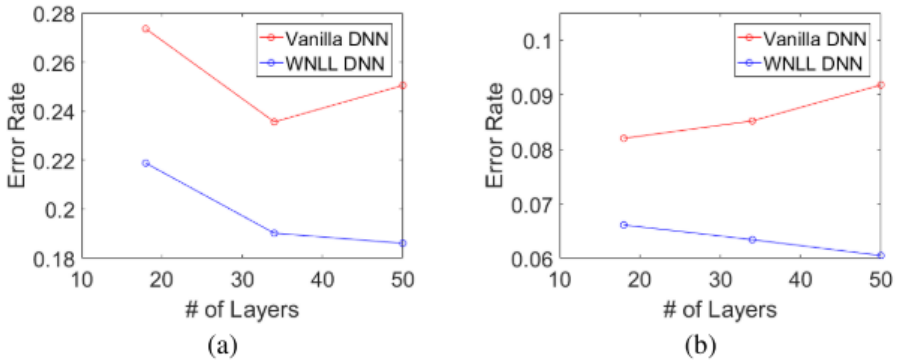


图 3：通过 WNLL 激活解决原始 DNN 的泛化精度退化的问题。（a）和（b）分别绘制了 1000 条和 10000 条训练数据用于训练原始和 WNLL 激活的 DNN 时的泛化误差。在每个图中，我们测试三个不同的网络：PreActResNet18，PreActResNet34 和 PreActResNet50。所有测试均在 CIFAR10 数据集上完成。

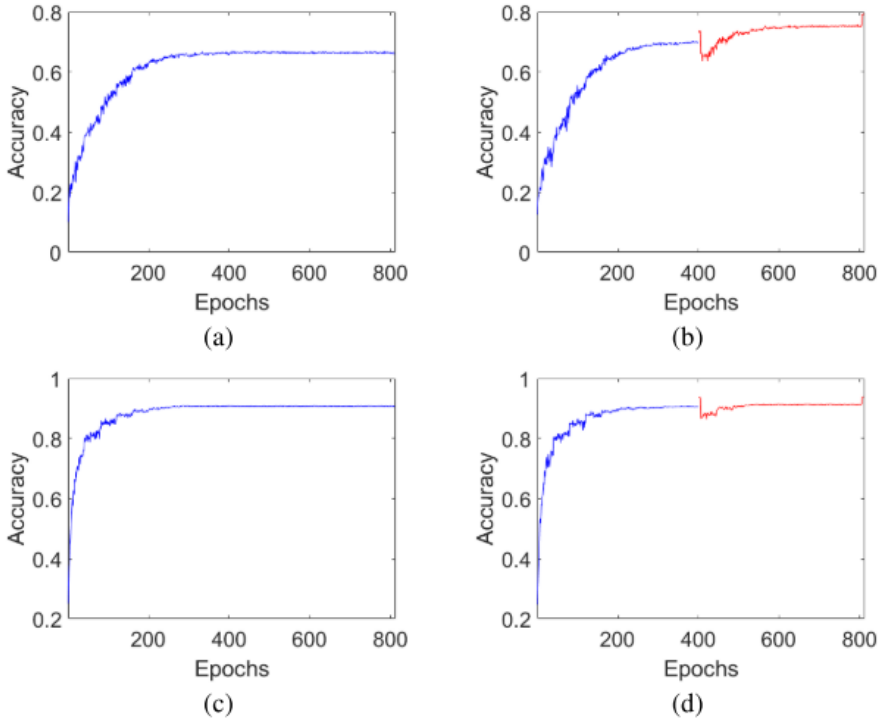


图 4：训练过程中泛化精度的演变。图（a）和（b）是具有 1000 个训练数据的 ResNet50 的精度图，它们分别是原始 DNN 和 WILL DNN 的周期-精度图。(c)和(d)对应于 PreActResNet50 的 10000 个训练数据的情况。所有测试均在 CIFAR10 数据集上完成。

钟哲灏

4.2 提高泛化精度

我们接下来展示 WNLL 激活的 DNN 比起它使用 softmax 或 SVM 输出激活的代理们在泛化精度方面的优越性。除了 ResNets，我们也在 VGG 网络上测试了 WNLL 代理。在表 2 中，我们列出了来自 VGG、ResNet、预激活 ResNet 家族的 15 个不同的 DNN 的泛化误差，基于 CIFAR10 训练集的前 10000 和前 1000 个实例。我们观察到，一般来说，WNLL 对 ResNet 和预激活 ResNet 的提高较大，而对 VGG 的提高较小，但仍然显著。除了对 VGG 外，我们可以在所有的神经网络中实现减小 20%-30%的测试误差的效果。在这里和论文其他部分给出的所有结果都是 5 个独立实验的中值。我们还将 SVM 作为一种替代的输出激活方法进行了比较，发现其结果仍然不如 WNLL。要注意的是，更大的批大小是为了保证 WNLL 的插值质量。一个合理的顾虑是，性能的提高来自于由于增加批大小而导致的方差减小。然而，批大小为 2000 的 vanilla 网络实际上降低了测试的准确性。

表 2：测试集在整个 CIFAR10 训练集的前 10000 个和前 1000 个实例训练出的 vanilla DNN、SVM 和 WNLL 激活的 DNN 上的泛化误差率。（5 个独立实验的中值）

Network	Whole			10000		1000	
	Vanilla	**WNLL**	**SVM**	**Vanilla**	**WNLL**	**Vanilla**	**WNLL**
VGG11	9.23%	**7.35%**	9.28%	10.37%	**8.88%**	26.75%	**24.10%**
VGG13	6.66%	**5.58%**	7.47%	9.12%	**7.64%**	24.85%	**22.56%**
VGG16	6.72%	**5.69%**	7.29%	9.01%	**7.54%**	25.41%	**22.23%**
VGG19	6.95%	**5.92%**	7.99%	9.62%	**8.09%**	25.70%	**22.87%**
ResNet20	9.06% (8.75%[12])	**7.09%**	9.60%	12.83%	**9.96%**	34.90%	**29.91%**
ResNet32	7.99% (7.51%[12])	**5.95%**	8.73%	11.18%	**8.15%**	33.41%	**28.78%**
ResNet44	7.31% (7.17%[12])	**5.70%**	8.67%	10.66%	**7.96%**	34.58%	**27.94%**
ResNet56	7.24% (6.97%[12])	**5.61%**	8.58%	9.83%	**7.61%**	37.83%	**28.18%**
ResNet110	6.41% (6.43%[12])	**4.98%**	8.06%	8.91%	**7.13%**	42.94%	**28.29%**
ResNet18	6.16%	**4.65%**	6.00%	8.26%	**6.29%**	27.02%	**22.48%**
ResNet34	5.93%	**4.26%**	6.32%	8.31%	**6.11%**	26.47%	**20.27%**
ResNet50	6.24%	**4.17%**	6.63%	9.64%	**6.49%**	29.69%	**20.19%**
PreActResNet18	6.21%	**4.74%**	6.38%	8.20%	**6.61%**	27.36%	**21.88%**
PreActResNet34	6.08%	**4.40%**	5.88%	8.52%	**6.34%**	23.56%	**19.02%**
PreActResNet50	6.05%	**4.27%**	5.91%	9.18%	**6.05%**	25.05%	**18.61%**

表 2 和表 3 列出了 CIFAR10 和 CIFAR100 数据集上 15 个不同的 vanilla 网络和 WNLL 激活网络的错误率。在 CIFAR10 上，WNLL 激活的 DNN 表现优于 vanilla DNN，有 1.5%到 2.0%的的相对误差减小或者 20%到 30%的绝对误差减小。在 CIFAR100 上的提高更为显著。我们独立地在两个数据集上都运行了 vanilla DNN，我们的结果与原始报告和其他研究者的复现结果一致。我们在附录中提供了 DNN 在 SVHN 数据上的性能表现的实验结果。有趣的是，这种改进在更困难的任务上更加显著，这表明我们的方法在其他任务/数据集上成功的潜力。例如，减小神经网络的大小是使神经网络具有通用性的一个重要方向，如无人驾驶、移动智能等。目前最成功的尝试是 DNN 权值量化。我们的方法提供了一种减小模型大小的新方向：我们的模型可以小得多就能达到与 vanilla 网络相同的精度水平。

朱祈璇

5 总结

我们受到多重插值和在 ResNets 与运输方程控制问题之间的联系的激励，我们决定将经典的输出激活函数 softmax 替换为一种谐波扩展类型的内插函数。这种简单的改变可以使得

深度神经网络（DNNs）充分地利用数据的流形信息（manifold imformation）。并且，使用一个端到端的贪婪多阶段训练算法来训练这种新型的输出层。在一方面，我们的新框架解决了数据不足导致的退化问题；在另一方面，与 baseline 相比，它显著地提高了泛化准确率。就算在不同类型或不同层数的网络中，这种改进都是一致存在的。泛化准确率的提升也可以用于训练拥有相同准确率的较小模型，这在移动设备的应用程序中有着巨大的应用潜力。【表 3：比较了在不同 NetWork 中原 DNN 和替换激活函数后的 WNLL DNN 的错误率对比，当然是 WNLL DNN 更强】

5.1 限制和未来的工作

在我们的框架中，还有一些我们希望改进的限制。目前来看，流形插值（manifold interpolation）步骤依旧是速度和内存的计算瓶颈。在插值中，为了使得插值有效，batch 的大小需要与 classes 的数量是拟线性的（这句话个人理解：由于我们每次是通过将一个 batch 里的数据进行插值，所以插值函数中的参数或者说权值大小是与 batch 大小有关的，而最终通过全连接层通过激活函数输出的结果大小与 classes 数量有关，所以说 batch 大小与 classes 数量是拟线性的）。这就对 ImageNet 数据集构成了内存挑战（详情见引用论文【7】，就当这句话不存在吧）。另一个重要的问题是 WNLL 激活函数的梯度的近似，近似为线性函数是一种选择，但它不是最好的选择。我们相信，使用一种更好的谐波函数进行近似可以进一步地提升模型的性能。

由于我们实验所显示的鲁棒性和泛化能力，我们可以推测，通过使用插值函数作为输出激活，神经网络在面对扰动和对抗性攻击时可以变得更加稳定（这里引用了论文【25】，虽然不知道对抗性攻击是什么，但总之是变强了）。而这种稳定性推测的原因是我们的框架在分类的过程中结合了学习决策边界和最近邻信息。