《深度学习》

神经网络中的正则化方法

孔文佳

2018 年 9 月 2 日

目录

1 神经网络中的正则化方法 **2**

1.1 Dropout 2

1.1.1 模型描述 2

1.1.2 实验结果 3

1.2 Batch Normalization 5

1.2.1 模型描述 5

1.2.2 实验结果 6

1.3 几种正则化方法对比 8

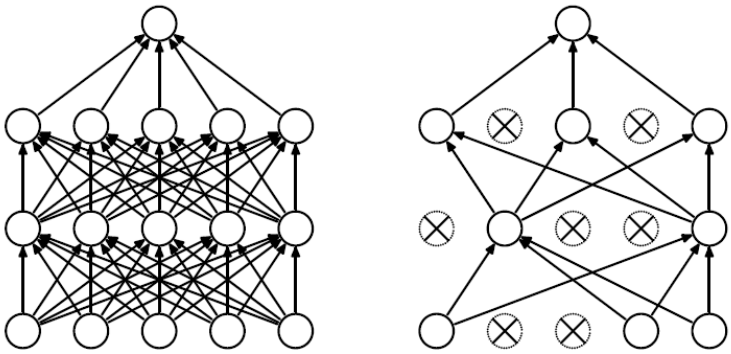
1 神经网络中的正则化方法

本节主要分析了神经网络的Batch Normalization以及Dropout正则化，并与L1,L2正则化进行了对比。进行相关实验，分析这几种方法对模型结果的影响。

1.1 Dropout

1.1.1 模型描述

过拟合现象是一些复杂神经网络模型存在的一个非常严重的问题，Dropout是由Hinton等人 [1] [2]提出的一种防止过拟合的方法，该方法的核心思想是在神经网络的训练阶段以一定的比例随机将某些神经元置为0，这样可以提高网络的泛化能力，如图（1）所示，左图是一个含有两个隐含层的全连接网络，右图是实施了Dropout之后的稀疏网络。



(a) Standard Neural Net　　(b) After applying dropout.

图 1: Dropout网络结构

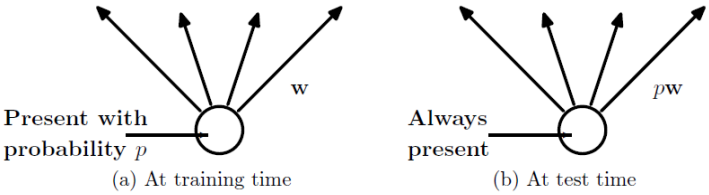
对于Dropout的理解，我们知道，将不同的模型进行组合可以提高网络的性能，但是对于大型网络，训练不同的模型并进行组合代价是非常大的，因为对于每个模型，都需要寻找最优的超参数，需要训练很长时间。并且可能没有足够的训练数据去训练很多不同的模型。退一步讲，即使能够训练不同的网络，将模型用于测试阶段也是不合理的，因为测试阶段要求迅速的输出测试结果。Dropout能够避免上述的问题，Dropout可以视为不同的网络的组合，若去掉的比例为0.5，隐含层的神经元个数为n，那么就有2^n个稀疏网络，这些网络共享了参数，因此所有的参数为$O(n^2)$甚至更少，这就解脱了训练多个独立的不同神经网络的时耗问题。

在测试阶段，不需要使用Dropout，网络的参数需要做scaled-down处理，如（2）所示，将参数乘以p，保证了在测试阶段的输出是训练阶段输出的期望。具体得，Dropout的公式如下:
Without Dropout:

$$\begin{aligned} z_i^{l+1} &= w_i^{l+1} y^l + b_i^{l+1} \\ y_i^{l+1} &= f(z_i^{l+1}) \end{aligned} \tag{1}$$

With Dropout:

$$
\begin{aligned}
r_j^j &\sim Bernoulli(p) \\
\hat{y}^l &= r^l * y^l \\
z_i^{l+1} &= w_i^{l+1}\hat{y}^l + b_i^{l+1} \\
y_i^{l+1} &= f(z_i^{l+1})
\end{aligned}
\tag{2}
$$



(a) At training time (b) At test time

图 2: 测试阶段

1.1.2 实验结果

为了比较Dropout对神经网络结果的影响，选取了MNIST和CIFAR10数据集，将数据统一成32*32大小，采用了VGG11网络，其结构如图（3）所示，比较了隐含层有无Dropout层的实验结果，实验中发现Dropout的概率p对结果有很大的影响，因此将Dropout的概率分别取p为[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7]，其中0代表不使用Dropout，每次训练20个epoch，batch为64，得到的测试集误差以及测试集精确度的变化如图（4）和图（5）所示。

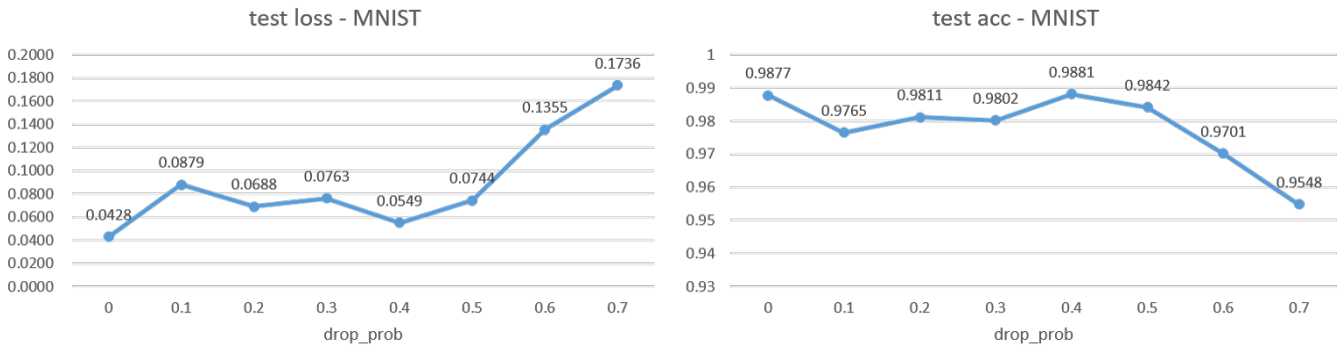


图 4: MNIST数据集结果

input(32*32RGB image)
conv3-64
maxpool
conv3-128
maxpool
conv3-256
conv3-256
maxpool
conv3-512
conv3-512
maxpool
conv3-512
conv3-512
maxpool
FC-128
FC-128
FC- 10
soft-max

图 3: VGG11结构

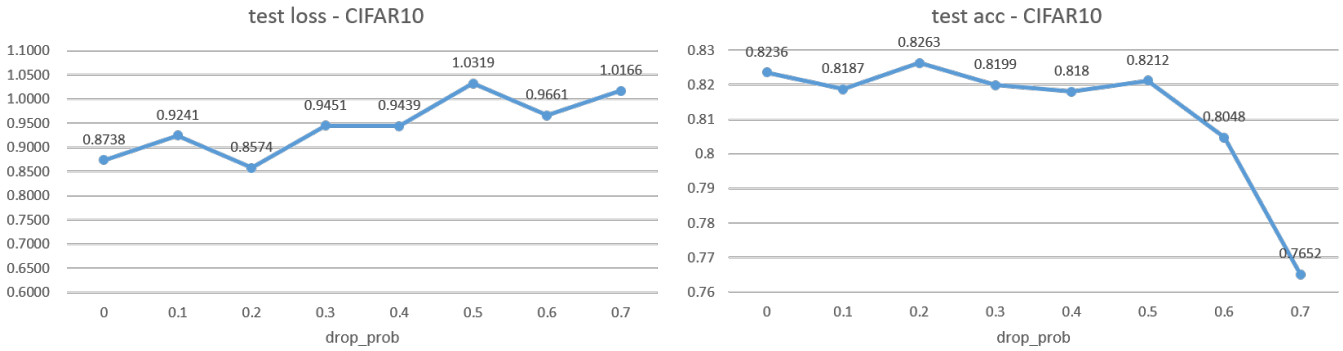


图 5: CIFAR10数据集结果

从上图可以看出，在使用Dropout后，随着p从0.1增大到0.7，测试集的误差先减小再上升，精确度先上升再逐渐减小，对于MNIST数据集来说，p的最佳取值为0.4，对于CIFAR10来说最佳取值为0.2，下表列出了采用Dropout前后的结果对比，可以看出当选择最佳Dropout概率后，实施Dropout能够提高准确率。

表 1: Dropout结果对比

	Without Dropout		With Dropout		acc 提高值
数据集	loss	acc	loss	acc	
MNIST	0.0428	0.9877	0.0549	0.9881	0.001
CIFAR	0.8738	0.8236	0.8574	0.8574	0.034

1.2 Batch Normalization

1.2.1 模型描述

神经网络学习过程本质就是为了学习数据分布，一旦训练数据与测试数据的分布不同，那么网络的泛化能力也大大降低；另外一方面，一旦每批训练数据的分布各不相同(batch 梯度下降)，那么网络就要在每次迭代都去学习适应不同的分布，这样将会大大降低网络的训练速度。Batch Normalization由Sergey Ioffe等人 [3]提出，能够有效解决在训练过程中，中间层数据分布发生改变的问题。其主要思想是在每一次输入激活函数前插入一个归一化层，进行归一化处理，保证每层的输入数据分布是稳定的，从而达到加速训练的目的。

首先，对于每一层的d维输入$x = (x^1, ..., x^d)$，将其进行归一化，均值为0方差为1：

$$\hat{x}^k = \frac{\hat{x} - E[x^k]}{\sqrt{Var[x^k]}} \tag{3}$$

但是仅仅进行这一步不能够反映出每一层学习到的特征，会将学习到的信息损坏。比如，如果激活函数采用Sigmoid函数的话，那么上式就会强行把数据归一化，使得非线性特征变为线性特征，因此BN算法在第二步加入了两个学习参数γ^k, β^k，这两个可学习的变量去还原上一层应该学到的数据分布：

$$y^k = \gamma^k \hat{x}^k + \beta^k \tag{4}$$

通过上述方法处理，既能够使得每一层数据进行归一化处理，又能够保证每一层学习到的特征不丢失，从而加速了神经网络的训练，对于每个Batch，BN算法流程如图（6）所示。在测试阶段，参数都是固定的，这时即使是每批训练样本进入网络，那么BN层计算的均值u、和标准差都是固定不变的。测试阶段的u和 σ 计算公式如下：

$$\begin{aligned} E[x] &= E_{\mathcal{B}}[\mathcal{B}] \\ Var[x] &= \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2] \end{aligned} \tag{5}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

图 6: Batch Normalization

最终的到的BN算法的流程如下图所示：

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $\mathrm{N}_{\mathrm{BN}}^{\mathrm{inf}}$

1: $\mathrm{N}_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $\mathrm{N}_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
4: Modify each layer in $\mathrm{N}_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $\mathrm{N}_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $\mathrm{N}_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow \mathrm{N}_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$

11: In $\mathrm{N}_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

图 7: Batch Normalization完整算法

1.2.2 实验结果

论文 [3]中提到加入BN层可以选择比较大的初始学习率，极大提高收敛速度。因此本次实验不仅对比了有无BN层的效果，还提高了学习率，分析在添加BN层的情况下学习率的提高是否能够显著的提高收敛速度。

MNIST数据集： 首先在MNIST数据集上进行实验，采用了LeNet和VGG11两种网络结构，初始学习率为0.001，分别比较了不加BN的原始模型，学习率为0.001的BN模型，提高5倍学习率即学习率为0.005的BN模型，提高30倍学习率即学习率为0.03的BN模型，将第一个epoch的测试集误差及精确度随训练次数的变化曲线画出，如图（8）和图（9）所示：

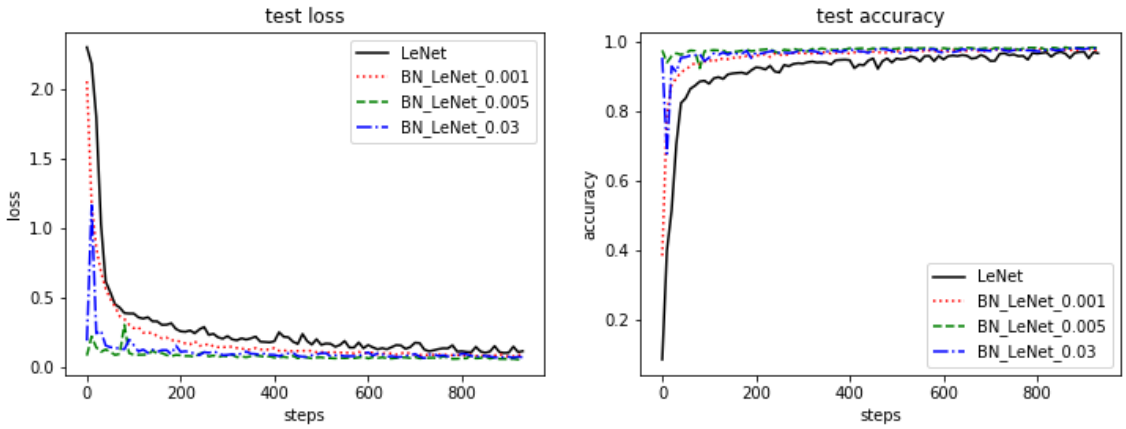


图 8: LeNet模型在MNIST上的结果比较

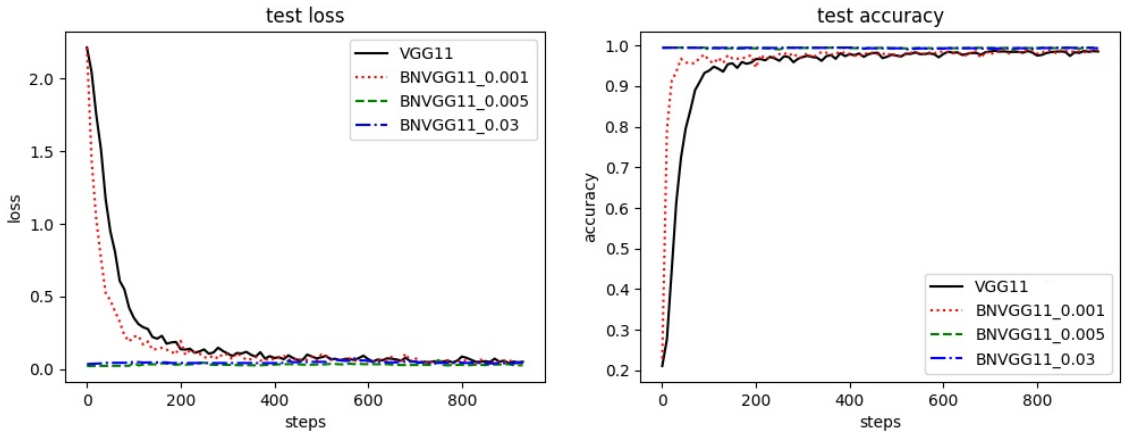


图 9: VGG11模型在MNIST上的结果比较

从上图可以看出，不论哪个模型，加入BN都能够加快模型的收敛，并且可以看出增大学习率能够进一步提高收敛速度。相对于结构比较简单的LeNet模型，VGG11更能够明显的看出在提高学习率之后，模型非常迅速的就达到了收敛状态。

CIFAR数据集：在CIFAR数据集上尝试了更复杂的网络VGG19,仍然比较了不同学习率下的BN网络与原始模型的测试集误差与分类精确度，将第一个epoch的训练的结果画出，结果如图（10）所示：

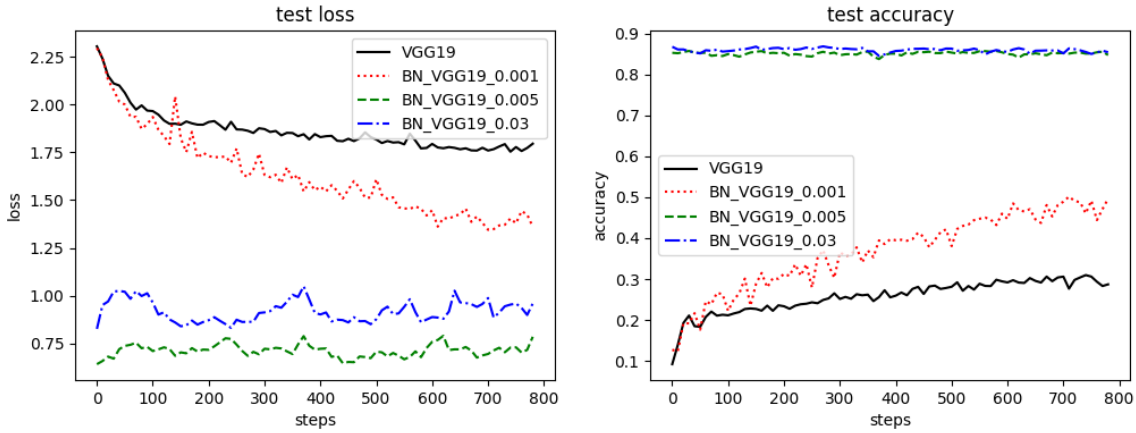


图 10: VGG19模型在CIFAR10上的结果比较

从上图可以非常明显的看出在训练完第一个epoch之后，原始模型还没有达到收敛状态，学习率为0.001的BN模型虽然也没有收敛，但是收敛速度明显比原始模型更快；将学习率提高5倍、30倍的BN模型均在第一个epoch就达到了收敛状态，另一方面提高30倍学习率的BN模型的误差要比提高5倍学习率的BN模型高一些，说明尽管能够通过提高学习率加速收敛，但是也并非无所上限。

继续训练模型，表（2）列出了所有模型精确度达到92%所用的epoch个数，可以看出同样达到92%的精确度，原始模型需要迭代20个epoch，但是0.001学习率的BN模型只需要12个epoch，提高学习率甚至只需要1个epoch，同时还能够达到更高的精确度，可见BN能够极大的加快收敛速度。

表 2: 模型精确度对比

Model	Epoches to 92%	Accuracy
VGG19	20	92.18%
BN-VGG19-0.001	12	93.02%
BN-VGG19-0.005	1	97.49%
BN-VGG19- 0.03	1	99.18%

1.3 几种正则化方法对比

论文中提到，BN模型可以代替过拟合中的drop out，也能够避免L2正则项参数的选择问题，采用BN算法后，可以移除这两项参数，或者可以选择更小的L2正则约束参数了，因为BN具有提高网络泛化能力的特性；因此本实验对比了几种正则化方法对模型结果的影响，即Batch Normalization,Dropout ,L1正则化，L2正则化，采用CIFAR10数据集，以及VGG11模型来进行实验，其中Dropout的置0概率根据图（5）中所示的结果取$p = 0.2$，L1、L2的正则化参数为0.001，迭代20个epoch，结果如图（11）所示：

表 3: 不同正则化方法对比

Model	loss	accuracy
Base	0.9427	82.94%
L1	0.8488	80.00%
L2	0.8028	81.36%
Dropout	0.9082	82.94%
BN	0.8461	83.03 %

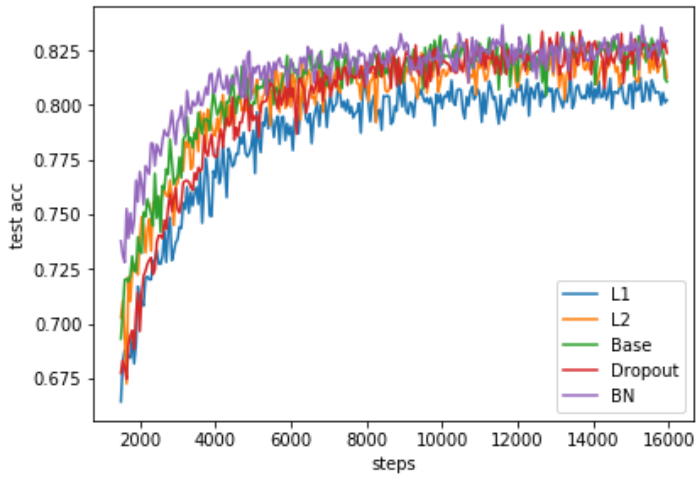


图 11: 不同正则化方法精确度随训练步数的变化

表（3）列出了不同模型迭代20个epoch之后的测试集误差即精度，可以看出所有正则化模型的误差都比基础模型有所降低，但是L1,L2正则化的精确度没有显著提高，这可能与正则化参数的设置有关，没有采用最佳的正则化参数，Dropout和BN都能够减小误差，提高测试准确度，但是结合图（11）来看在所有的正则化方法中BN的表现最好，不仅能够提高准确度，还能够加快训练速度，提高模型的泛化能力。

参考文献

[1] Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors[J]. arXiv preprint arXiv:1207.0580, 2012.

[2] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

[3] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.