人工神经网络实验报告
MLP 与 CNN

张钰晖
2015011372, yuhui-zh15@mails.tsinghua.edu.cn, 185-3888-2881

目录

1 问题描述

手写数字识别是机器学习中经典的研究问题，MNIST 数据集提供了 60000 组训练样例和 10000 组测试样例。下面展示了其中的几张样本。

图 1: MNIST 数据样本

在之前的作业中，笔者通过助教提供好的框架，手动实现了 MLP 和 CNN 的前向传播与反向传播过程，在本次作业中，笔者将采用目前机器学习领域最为流行的框架 Tensorflow，来完成这一任务。

2 具体实现

2.1 全连接层的实现

全连接层是 MLP 的基本结构，其数学本质是矩阵乘法与矩阵加法，在 Tensorflow 中，通过建立矩阵变量并进行运算实现。

```
def linear_layer(inputs, input_size, output_size, isTrain=True):
    W = weight_variable([input_size, output_size])
    b = bias_variable([output_size])
    return tf.matmul(inputs, W) + b
```

2.2 卷积层的实现

Tensorflow 提供了卷积层的实现，在这里选取 padding='SAME'，Tensorflow 会自动在其周围补 0 而使得卷积后大小不变。

```
def conv_layer(inputs, kernel_size, channel_in, channel_out, isTrain=True):
    W = weight_variable([kernel_size, kernel_size, channel_in, channel_out])
    b = bias_variable([channel_out])
    return tf.nn.conv2d(inputs, W, strides=[1, 1, 1, 1], padding='SAME') + b
```

2.3 池化层的实现

Tensorflow 提供了池化层的实现，这里实现了 2x2 最大池化。

```
def max_pool_layer(inputs, kernel_size, isTrain=True):
    return tf.nn.max_pool(inputs, ksize=[1, kernel_size, kernel_size, 1], strides=[1,
        kernel_size, kernel_size, 1], padding='SAME')
```

2.4 激活函数层的实现

Tensorflow 提供了 Relu 激活函数的实现，可以直接使用。

```
def relu_layer(inputs, isTrain=True):
    return tf.nn.relu(inputs)
```

2.5 丢弃层的实现

丢弃层 (Dropout) 有助于显著防止过拟合，从而提高效果。Dropout 一般用在激活函数之后，一般用在输入层与输出层之间。

```
def dropout_layer(inputs, keep_prob, isTrain=True):
    return tf.nn.dropout(inputs, keep_prob)
```

2.6 批标准化层的实现

批标准化 (Batch Normalization) 是 Sergey Ioffe 和 Christian Szegedy 提出了一种加速深度网络的算法，这个算法目前已经被大量的应用，其本质思想是对每一层结果进行标准化，从而减少内部分布的变化。

仅仅使用 batch 的平均值和方差是不够的，在训练时，需要更新维护总的平均值和方差 (采用指数平滑的方式)，在测试时使用总的平均值和方差作为归一参数。这是为了解决单一测试 (one-by-one test) 的问题，如果不这样做的话，当 batch size 为 1 时，归一化后该层向量将全部变为 0。

```
def batch_normalization_layer(inputs, isTrain=True):
    mean_total, variance_total = tf.Variable(tf.zeros(inputs.get_shape()[-1]), trainable=
        False), tf.Variable(tf.zeros(inputs.get_shape()[-1]), trainable=False) # 总的平均值和方
        差
    mean, variance = tf.nn.moments(inputs, [0]) # 批的平均值和方差
    decay = tf.constant(0.9)
    gamma = tf.Variable(tf.ones(inputs.get_shape()[-1]))
    beta = tf.Variable(tf.zeros(inputs.get_shape()[-1]))
    epsilon = tf.constant(1e-3)
    if isTrain:
```

```
        # 采用指数平滑的方式更新总的平均值和方差
        assign_mean = tf.assign(mean_total, mean_total * decay + mean * (1 - decay))
        assign_variance = tf.assign(variance_total, variance_total * decay + variance * (1 -
            decay))
        with tf.control_dependencies([assign_mean, assign_variance]):
            norm = tf.nn.batch_normalization(inputs, mean, variance, beta, gamma, epsilon)
    else: norm = tf.nn.batch_normalization(inputs, mean_total, variance_total, beta, gamma,
        epsilon)
    return norm
```

3 MLP 与 CNN 对比

3.1 训练曲线

训练曲线刻画了准确率 Accuracy 与损失 Loss 随训练次数变化的曲线。下图展示了 MLP 和 CNN 的训练曲线：

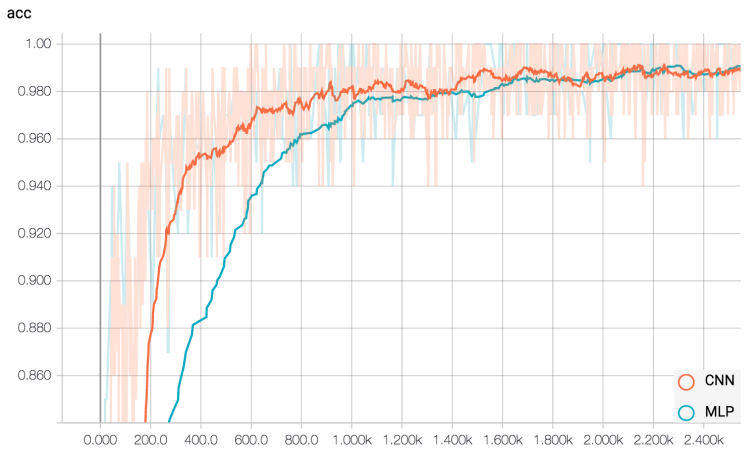

图 2: Accuracy-Batch 曲线 (准确率为训练集准确率)

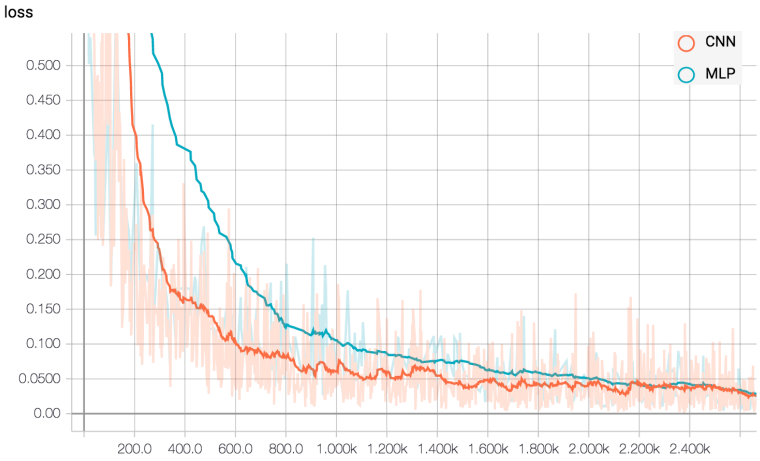

图 3: Loss-Epoch 曲线

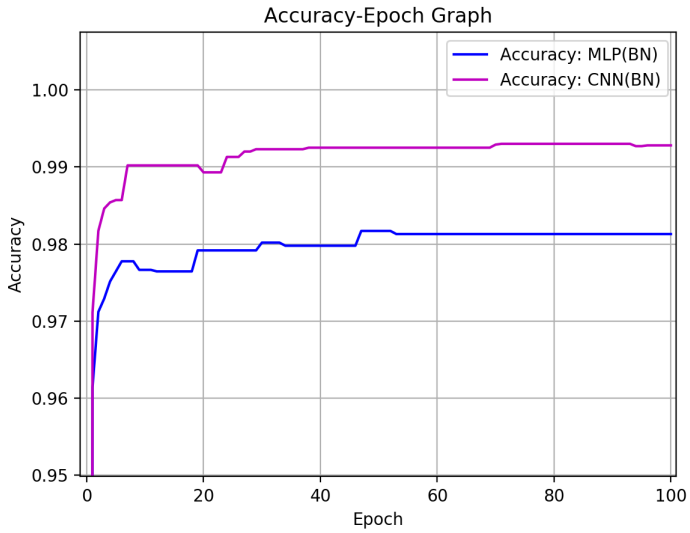

图 4: Accuracy-Epoch 曲线 (准确率为测试集准确率)

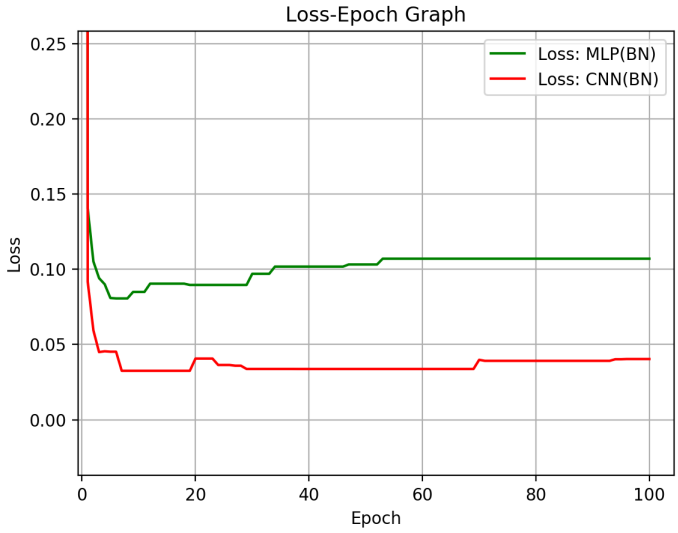

图 5: Loss-Epoch 曲线

对应的网络参数如下：

表 1: MLP 网络结构

层	**参数**	**规模**
Linear		输入: $N * 784$
		输出: $N * 400$
Normalization	Decay: 0.9	输入: $N * 400$
	Epsilon: 0.001	输出: $N * 400$
Relu		输入: $N * 400$
		输出: $N * 400$
Linear		输入: $N * 400$
		输出: $N * 10$

表 2: CNN 网络结构

层	参数	规模
Convolutional	C In: 1	输入: $N * 1 * 28 * 28$
	C Out: 16	输出: $N * 16 * 28 * 28$
	Kernel Size: 5	
	Padding: 2	
Normalization	Decay: 0.9	输入: $N * 16 * 28 * 28$
	Epsilon: 0.001	输出: $N * 16 * 28 * 28$
Relu		输入: $N * 16 * 28 * 28$
		输出: $N * 16 * 28 * 28$
AveragePool	Kernel Size: 2	输入: $N * 16 * 28 * 28$
	Padding: 0	输出: $N * 16 * 14 * 14$
Convolutional	C In: 16	输入: $N * 16 * 14 * 14$
	C Out: 32	输出: $N * 32 * 14 * 14$
	Kernel Size: 5	
	Padding: 2	
Normalization	Decay: 0.9	输入: $N * 32 * 14 * 14$
	Epsilon: 0.001	输出: $N * 32 * 14 * 14$
Relu		输入: $N * 32 * 14 * 14$
		输出: $N * 32 * 14 * 14$
AveragePool	Kernel Size: 2	输入: $N * 32 * 14 * 14$
	Padding: 0	输出: $N * 32 * 7 * 7$
Linear		输入: $N * 1568$
		输出: $N * 10$

表 3: 网络参数

激活函数	Relu
损失函数	SoftmaxCrossEntropyLoss
其他参数	Learning Rate: 0.001
	Learning Rate Decay Factor: 0.9995
	Batch Size: 100

3.2 对比结果

下面这张表格从网络结构、参数数量、训练结果三方面对比了 MLP 与 CNN，所有网络均训练 100 个 Epoch：

表 4: 网络参数

	双层 CNN	**单层 MLP**
网络结构	(卷积, 标准化, 激活, 池化) 双层卷积网络	(连接, 标准化, 激活) 单隐层网络
激活函数	Relu	Relu
损失函数	SoftmaxCrossEntropy	SoftmaxCrossEntropy
批标准化	是	是
丢弃层	否	否
参数数量	28938	318010
准确率	**99.30**%	98.17%
训练速度	很慢 (约 100s/epoch)	很快 (约 7s/epoch)
收敛速度	快	快
收敛性	好	好

3.3 结果分析

以上选取了两种网络结构进行对比，均采用 Relu 激活函数，这是因为 Relu 函数作为激活函数有以下优势：

- 速度快：Relu 函数计算简单，只需和 0 计算 max 即可。
- 减轻梯度消失问题：Sigmoid 函数导数最大值是 0.25，用 Sigmoid 函数作为激活函数会导致梯度越来越小，对于深层网络训练是个很大的问题，而 Relu 函数的导数是 1，不会导致梯度减小。

通过不同方面的对比，我们可以清楚地看到 CNN 与 MLP 的异同。

- 从准确率可以看出，CNN 在图像处理上具有极强的优势，其准确率达到了 99.30%！而 MLP 相比之下准确率只有 98.17%。在训练充分后，CNN 能充分模仿生物的感受野，获取图像关键信息，从而达到很高的准确率。
- 从参数数量可以看出，由于局部连接、权值共享和池化，处理同样的问题 CNN 参数数量和 MLP 相比大为下降，远远少于 MLP。以双层 CNN 计算为例，参数数量为 $16 *$

$1 * 5 * 5 + 16 + 32 * 16 * 5 * 5 + 32 + 1568 * 10 + 10 = 28938$，而单层 MLP 参数数量为 $784 * 400 + 400 + 400 * 10 + 10 = 318010$。

- 从训练结果可以看出，CNN 和 MLP 的准确率都较高，收敛性都较好，收敛速度都较快 (此处收敛速度是指准确率稳定时所需的训练次数)，相对而言 CNN 的收敛速度快于 MLP。
- 从收敛速度可以看出 (此处收敛速度是指训练稳定时所需迭代次数)，CNN 收敛速度快于 MLP，迭代次数较少时损失 Loss 下降迅速，准确率 Accuracy 提升迅速。随着迭代次数的增加，损失 Loss 下降速度变得平缓，准确率 Accuracy 提升速度也变得平缓，并逐步收敛，最终稳定在收敛值附近。
- 从训练速度可以看出 (此处训练速度是指训练一个 1Epoch 所需时间)，CNN 训练速度相比 MLP 而言非常慢，这是因为 CNN 卷积计算复杂度很高，需要进行大量计算。

4 批标准化

批标准化 (Batch Normalization) 是 Sergey Ioffe 和 Christian Szegedy 提出了一种加速深度网络的算法。**批标准化的本质思想是对每一层结果进行标准化，从而减少内部分布的变化**。

一般来说，如果模型的输入特征不相关且满足标准正态分布 N(0, 1) 时，模型的表现一般较好。在训练神经网络模型时，我们可以事先将特征去相关并使得它们满足一个比较好的分布，这样，模型的第一层网络一般都会有一个比较好的输入特征，但是随着模型的层数加深，网络的非线性变换使得每一层的结果变得相关了，且不再满足 N(0, 1) 分布。更糟糕的是，可能这些隐藏层的特征分布已经发生了偏移。

论文的作者认为上面的问题会使得神经网络的训练变得困难，为了解决这个问题，他们提出在层与层之间加入 Batch Normalization 层。训练时，BN 层利用隐藏层输出结果的均值 $\mu_{\mathcal{B}}$ 与方差 $\sigma_{\mathcal{B}}^2$ 来标准化每一层特征的分布，并且维护所有 mini-batch 数据的均值与方差，最后利用样本的均值与方差的无偏估计量用于测试时使用。

鉴于在某些情况下非标准化分布的层特征可能是最优的，标准化每一层的输出特征反而会使得网络的表达能力变得不好，作者为 BN 层加上了两个可学习的缩放参数 γ 和偏移参数 β 来允许模型自适应地去调整层特征分布。

其数学原理如下：

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

图 6: 批标准化数学原理

4.1 训练曲线

训练曲线可以清晰地看出批标准化对训练过程的影响，以下训练曲线分别展示了 MLP 和 CNN 进行批标准化与不进行批标准化，训练单位为一个 Batch，该图像由 Tensorboard 生成，对图像进行了平滑处理。

4.1.1 对 MLP

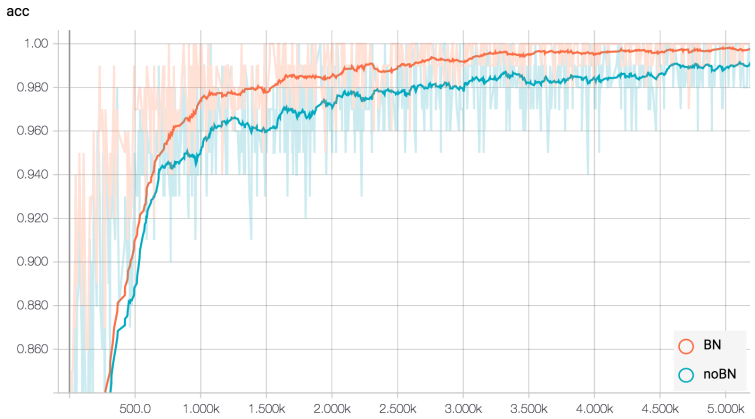

图 7: MLP 准确率曲线 (准确率为训练集准确率)

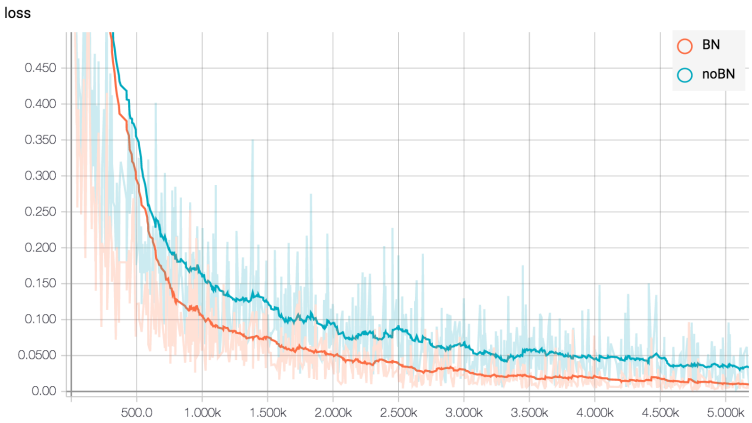

图 8: MLP 损失曲线

由上述图像可以清晰的看出来批标准化在训练中的作用，对 MLP 而言，在**较少的 Epoch 之内**，批处理化有助于：

- 提升准确率。从 Accuracy 曲线可以看出，在较少的 Epoch 之内，批标准化有利于提升准确率。
- 增加收敛速度。从 Loss 曲线可以看出，在较少的 Epoch 之内，批标准化有利于加快收敛速度。

4.1.2 对 CNN

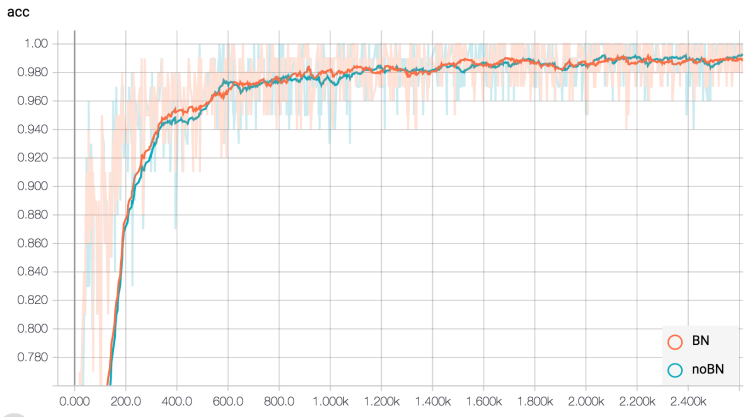

图 9: CNN 准确率曲线 (准确率为训练集准确率)

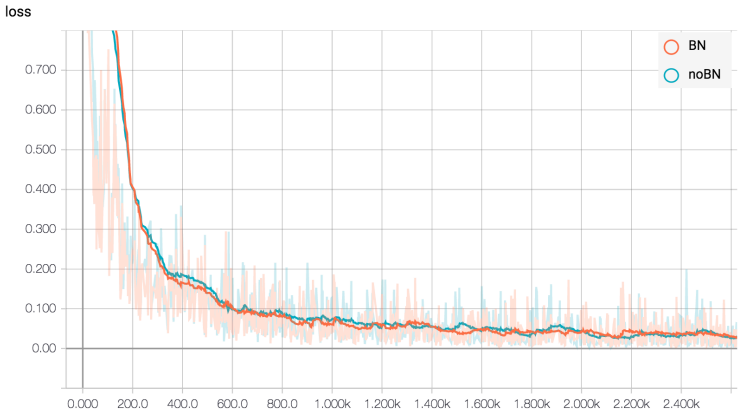

图 10: CNN 损失曲线

对 CNN 而言，增加批标准化过程对训练过程却并没有什么显著影响。

4.2 对比结果

下面这张表格展示了网络训练 100 个 Epoch 时的结果：

表 5: 网络参数

	双层 CNN	双层 CNN	单层 MLP	单层 MLP
批标准化	是	否	是	否
丢弃层	否	否	否	否
激活函数	Relu	Relu	Relu	Relu
损失函数	SoftmaxCrossEntropy	SoftmaxCrossEntropy	SoftmaxCrossEntropy	SoftmaxCrossEntropy
准确率	**99.30**%	99.25%	98.17%	**98.30**%
训练速度	很慢 (约 100s/epoch)	很慢 (约 85s/epoch)	很快 (约 7s/epoch)	很快 (约 6s/epoch)
收敛速度	快	快	快	快
收敛性	好	好	好	好

从上述结果我们可以看出，在训练足够充分时，批标准化似乎意义不大 (可能有实验误差)，在误差范围内对结果影响很小，甚至 MLP 不加批标准化层反而测试集准确率略高。

4.3 本质

批标准化的本质是改变数据分布密度，以下通过 Tensorboard 的绘出的直方图可以清晰的看出这个结果。

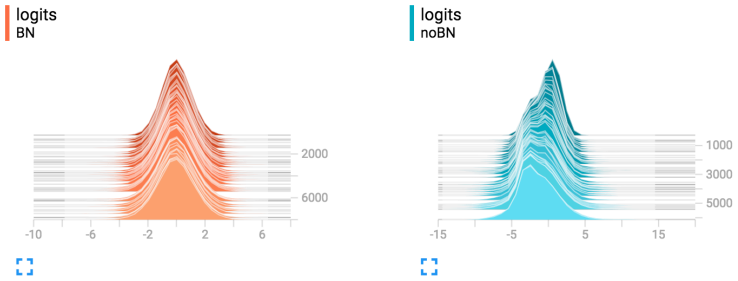

图 11: 批标准化的效果

增加批标准化层之后，数据的分布情况不随时间改变，而不加该层则可以看出明显的数据概率分布的移动。

5 丢弃层

丢弃 (Dropout) 是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。对于随机梯度下降来说，由于是随机丢弃，故而每一个 mini-batch 都在训练不同的网络。**Dropout 有助于显著防止过拟合，从而提高模型效果**。

Dropout 一般用在全连接中，在卷积部分不会用到 dropout，输出层也不会用到，一般用在输入层与输出层之间。

以下以 MLP 为例，来展示 Dropout 层的效果。

5.1 训练曲线

训练曲线可以清晰地看出丢弃层对训练过程的影响，以下训练曲线展示了 MLP 加丢弃层与不加丢弃层的结果对比，训练单位为一个 Batch，该图像由 Tensorboard 生成，对图像进行了平滑处理。

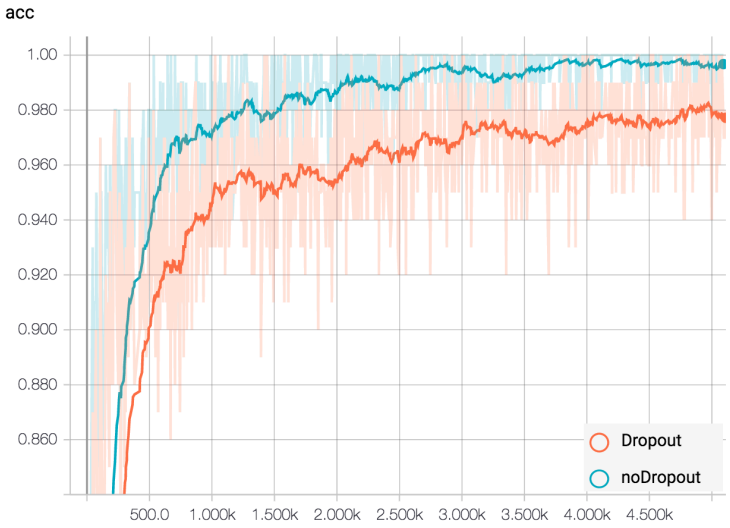

图 12: MLP 准确率曲线 (准确率为训练集准确率)

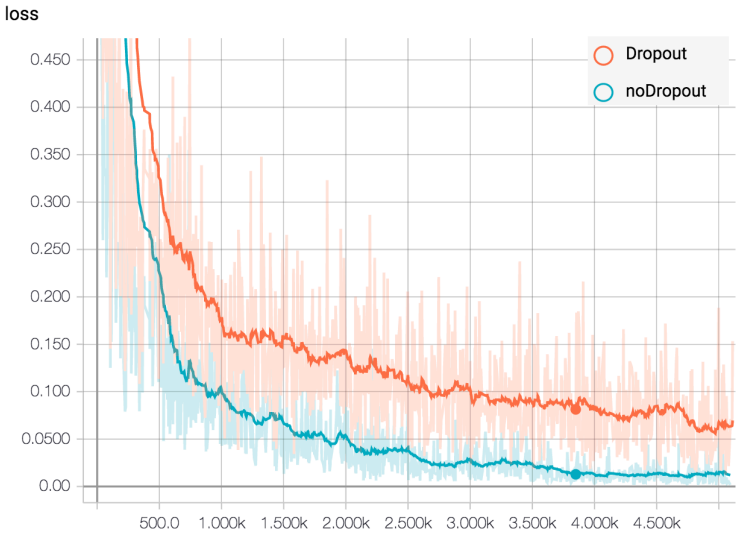

图 13: MLP 损失曲线

由上述图像可以清晰的看出来丢弃层在训练中的影响，对 MLP 而言，在**短期而言**，似乎增加丢弃层会降低训练速度和训练集准确率，但实际上，训练集准确率降低不代表测试集准确率下降，训练集准确率下降实际反映了模型并没有过拟合。

5.2 对比结果

下面这张表格展示了网络训练 100 个 Epoch 时的结果：

表 6: 网络参数

	单层 MLP	**单层 MLP**
批标准化	是	是
丢弃层	是	否
激活函数	Relu	Relu
损失函数	SoftmaxCrossEntropy	SoftmaxCrossEntropy
准确率	98.17%	**98.40%**
训练速度	很快 (约 7s/epoch)	很快 (约 7s/epoch)
收敛速度	快	快
收敛性	好	好

从上述结果我们可以看出，在**训练足够充分**时，丢弃层对 MLP 影响较大，增加丢弃层可以显著提升 MLP 的准确率。这是因为丢弃层可以显著避免模型过拟合，尽管训练集准确率下降了，但是测试集准确率却取得了很大的提升。

5.3 本质

大规模的神经网络有两个缺点：费时且容易过拟合。Dropout 的出现很好的可以解决这个问题，每次做完 dropout，相当于从原始的网络中找到一个更瘦的网络，如下图所示。

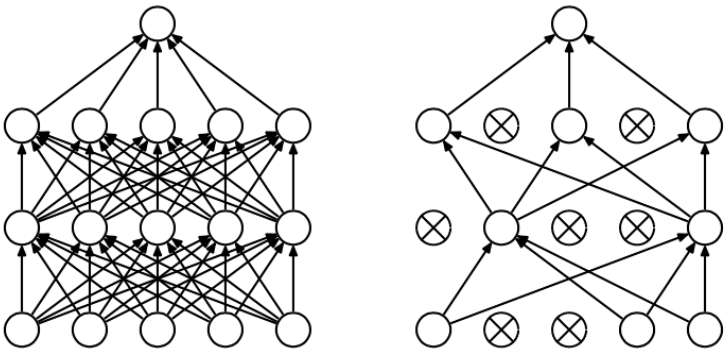

图 14: 丢弃层的本质

从结果也可以看出，在训练足够充分时，Dropout 对 MLP 影响较大，可以显著避免过拟合，提升 MLP 的准确率。

6 单一测试

笔者认为，单一测试 (One-by-one Test) 实际上是对批标准化层的测试，在单一测试中如果批标准化实现不当，没有考虑单一数据情形时，则会导致经过批标准化层之后数据向量全部变为 0。

因此在批标准化层中判断了当前状态，若为训练状态，则均值和方差取 Batch 的均值和方差，并指数累加总均值和方差；若为测试状态，则均值和方差取自训练数据的方差的指数平均。

最终单一测试的结果如下：

表 7: 网络参数

	双层 CNN	双层 CNN	单层 MLP	单层 MLP
批标准化	是	否	是	否
丢弃层	否	否	否	否
激活函数	Relu	Relu	Relu	Relu
损失函数	SoftmaxCrossEntropy	SoftmaxCrossEntropy	SoftmaxCrossEntropy	SoftmaxCrossEntropy
准确率	**99.37%**	99.30%	98.20%	**98.27%**

7 超参数选择

调整参数有利于提高模型准确率，增加训练速度，笔者在实验中，尝试了不同的参数，改变参数后每次训练 10 个 Epoch 后取一个较为可观的参数。但由于篇幅所限，笔者不再一一展示调参过程，仅以表格展示最优参数参数。

表 8: 超参数选择

Learning Rate: 0.001	控制学习速率
Learning Rate Decay Factor: 0.9995	控制学习速率下降
Batch Size: 100	控制批大小
Keep Prob: 0.5	控制丢弃层中保持率
Decay: 0.9	控制批标准化层总均值与方差更新
Epsilon: 0.001	避免批标准化层中除 0 异常

8 实验心得

通过本次实验，我对多层感知器 (MLP) 和卷积神经网络 (CNN) 的理解进一步得到加深，相比自己实现前向传递和后向传递而言，使用 Tensorflow 搭建框架大大简化了作业难度，我也更加深入的理解了 Tensorflow。

在实现过程中，由于对 Tensorflow 理解有偏差，导致批标准化层对单一测试一直无效，发现每步并不会更新总平均和总方差，查阅资料明白 Tensorflow 的数据流原理后，通过增加依赖关系使得总平均和总方差终于得以更新。

通过对比 Tensorflow 的 CNN 运行速度和作业 2 自己实现的 CNN 运行速度，我也感受到机器学习框架的底层不知道经历了多少了多少精妙绝伦的优化才能达到这样的速度。较快的

训练速度使得我可以增大卷积层的通道数，充分训练 CNN(训练了 100 个 Epoch)，识别准确率也达到了 99.30%。

对我而言，这依旧只是一个入门，还需要进一步的练习与实践。

参考文献

[1] 人工神经网络, 黄民烈, 2017.

[2] 零基础入门深度学习, *Han Bingtao*, 2017.

[3] 理解 dropout, *Zhang Yushi*, 2015.

[4] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *Sergey Ioffe, Christian Szegedy*, 2015.