Batch Normalization原理与实战（下）

mp.weixin.qq.com/s/iZp0Te9Qnr6Cg1GY7vcUNw

链接 | https://zhuanlan.zhihu.com/p/34879333

前言

本文主要从理论与实战视角对深度学习中的Batch Normalization的思路进行讲解、归纳和总结，并辅以代码让小伙伴儿们对Batch Normalization的作用有更加直观的了解。

本文主要分为两大部分，由于篇幅过长，分为上下两篇。**本文为第二部分实战板块**，主要以MNIST数据集作为整个代码测试的数据，通过比较加入Batch Normalization前后网络的性能来让大家对Batch Normalization的作用与效果有更加直观的感知。

二、实战板块
经过了上面了理论学习，我们对BN有了理论上的认知。“Talk is cheap, show me the code”。接下来我们就通过实际的代码来对比加入BN前后的模型效果。实战部分使用MNIST数据集作为数据基础，并使用TensorFlow中的Batch Normalization结构来进行BN的实现。
数据准备：MNIST手写数据集
代码地址：https://github.com/NELSONZHAO/zhihu/tree/master/batch_normalization_discussion
注：TensorFlow版本为1.6.0
实战板块主要分为两部分：

- 网络构建与辅助函数
- BN测试

1、网络构建与辅助函数

首先我们先定义一下神经网络的类，这个类里面主要包括了以下方法：

- build_network：前向计算
- fully_connected：全连接计算
- train：训练模型
- test：测试模型

1.1 build_network
我们首先通过构造函数，把权重、激活函数以及是否使用BN这些变量传入，并生成一个training_accuracies来记录训练过程中的模型准确率变化。这里的initial_weights是一个list，list中每一个元素是一个矩阵（二维tuple），存储了每一层的权重矩阵。build_network实现了网络的构建，并调用了fully_connected函数（下面会提）进行计算。要注意的是，由于MNIST是多分类，在这里我们不需要对最后一层进行激活，保留计算的logits就好。

```
class NeuralNetWork():
    def __init__(self, initial_weights, activation_fn, use_batch_norm):
        """
        初始化网络对象
        :param initial_weights: 权重初始化值，是一个list，list中每一个元素是一个权重矩阵
        :param activation_fn: 隐层激活函数
        :param user_batch_norm: 是否使用batch normalization
        """
        self.use_batch_norm = use_batch_norm
        self.name = "With Batch Norm" if use_batch_norm else "Without Batch Norm"

        self.is_training = tf.placeholder(tf.bool, name='is_training')

        # 存储训练准确率
        self.training_accuracies = []

        self.build_network(initial_weights, activation_fn)

    def build_network(self, initial_weights, activation_fn):
        """
        构建网络图
        :param initial_weights: 权重初始化，是一个list
        :param activation_fn: 隐层激活函数
        """
        self.input_layer = tf.placeholder(tf.float32, [None, initial_weights[0].shape[0]])
        layer_in = self.input_layer

        # 前向计算（不计算最后输出层）
        for layer_weights in initial_weights[:-1]:
            layer_in = self.fully_connected(layer_in, layer_weights, activation_fn)

        # 输出层
        self.output_layer = self.fully_connected(layer_in, initial_weights[-1])
```

1.2 fully_connected
这里的fully_connected主要用来每一层的线性与非线性计算。通过self.use_batch_norm来控制是否使用BN。

```python
def fully_connected(self, layer_in, layer_weights, activation_fn=None):
    """
    抽象出的全连接层计算
    """
    # 如果使用BN与激活函数
    if self.use_batch_norm and activation_fn:
        weights = tf.Variable(layer_weights)
        linear_output = tf.matmul(layer_in, weights)

        # 调用BN接口
        batch_normalized_output = tf.layers.batch_normalization(linear_output, training=self.is_training)

        return activation_fn(batch_normalized_output)
    # 如果不使用BN或激活函数（即普通隐层）
    else:
        weights = tf.Variable(layer_weights)
        bias = tf.Variable(tf.zeros([layer_weights.shape[-1]]))
        linear_output = tf.add(tf.matmul(layer_in, weights), bias)

        return activation_fn(linear_output) if activation_fn else linear_output
```

另外，值得注意的是，tf.layers.batch_normalization接口中training参数非常重要，官方文档中描述为：

`training` : Either a Python boolean, or a TensorFlow boolean scalar tensor (e.g. a placeholder). Whether to return the output in training mode (normalized with statistics of the current batch) or in inference mode (normalized with moving statistics). NOTE: make sure to set this parameter correctly, or else your training/inference will not work properly.

当我们训练时，要设置为True，保证在训练过程中使用的是mini-batch的统计量进行normalization；在Inference阶段，使用False，也就是使用总体样本的无偏估计。

1.3 train

train函数主要用来进行模型的训练。除了要定义label，loss以及optimizer以外，我们还需要注意，官方文档指出在使用BN时的事项：

Note: when training, the moving_mean and moving_variance need to be updated. By default the update ops are placed in `tf.GraphKeys.UPDATE_OPS` , so they need to be added as a dependency to the `train_op` .

因此当self.use_batch_norm为True时，要使用tf.control_dependencies保证模型正常训练。

```python
def train(self, sess, learning_rate, training_batches, batches_per_validate_data, save_model=None):
    """
    训练模型
    :param sess: TensorFlow Session
    :param learning_rate: 学习率
    :param training_batches: 用于训练的batch数
    :param batches_per_validate_data: 训练多少个batch对validation数据进行一次验证
    :param save_model: 存储模型
    """

    # 定义输出label
    labels = tf.placeholder(tf.float32, [None, 10])

    # 定义损失函数
    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=labels,
                                                                               logits=self.output_layer))

    # 准确率
    correct_prediction = tf.equal(tf.argmax(self.output_layer, 1), tf.argmax(labels, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    #
    if self.use_batch_norm:
        with tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):
            train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entropy)

    else:
        train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entropy)

    # 显示进度条
    for i in tqdm.tqdm(range(training_batches)):
        batch_x, batch_y = mnist.train.next_batch(60)
        sess.run(train_step, feed_dict={self.input_layer: batch_x,
                                        labels: batch_y,
                                        self.is_training: True})
        if i % batches_per_validate_data == 0:
            val_accuracy = sess.run(accuracy, feed_dict={self.input_layer: mnist.validation.images,
                                                         labels: mnist.validation.labels,
                                                         self.is_training: False})
            self.training_accuracies.append(val_accuracy)
    print("{}: The final accuracy on validation data is {}".format(self.name, val_accuracy))

    # 存储模型
    if save_model:
        tf.train.Saver().save(sess, save_model)
```

注意：在训练过程中batch_size选了60（mnist.train.next_batch(60)），这里是因为BN的原paper中用的60。(We trained the network for 50000 steps, with 60 examples per mini-batch.)

1.4 test

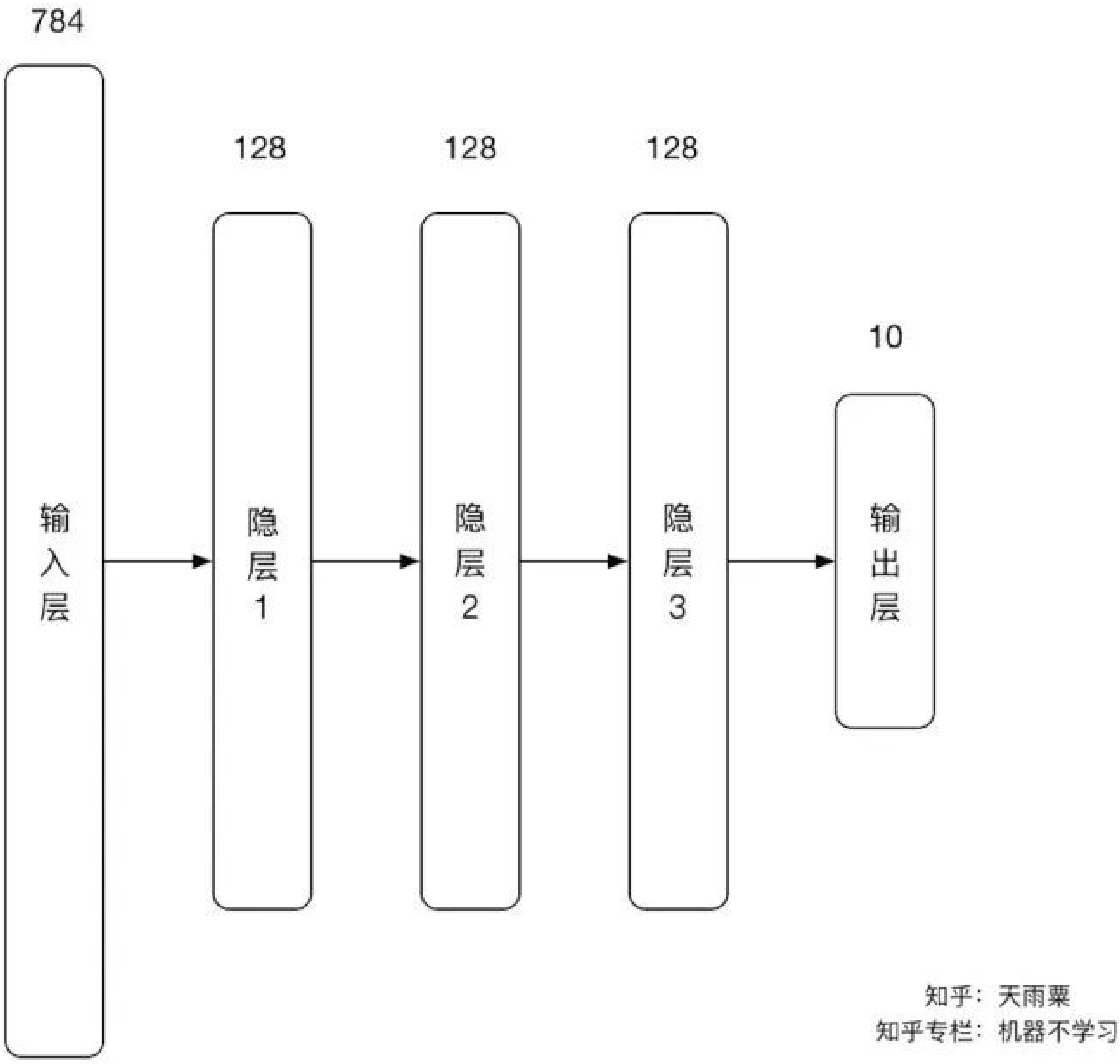
test阶段与train类似，只是要设置self.is_training=False，保证Inference阶段BN的正确。

```python
def test(self, sess, test_training_accuracy=False, restore=None):
    # 定义label
    labels = tf.placeholder(tf.float32, [None, 10])

    # 准确率
    correct_prediction = tf.equal(tf.argmax(self.output_layer, 1), tf.argmax(labels, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    # 是否加载模型
    if restore:
        tf.train.Saver().restore(sess, restore)

    test_accuracy = sess.run(accuracy, feed_dict={self.input_layer: mnist.test.images,
                                                  labels: mnist.test.labels,
                                                  self.is_training: False})

    print("{}: The final accuracy on test data is {}".format(self.name, test_accuracy))
```

经过上面的步骤，我们的框架基本就搭好了，接下来我们再写一个辅助函数train_and_test以及plot绘图函数就可以开始对BN进行测试啦。train_and_test以及plot函数见GitHub代码中，这里不再赘述

2、BN测试

在这里，我们构造一个4层神经网络，输入层结点数784，三个隐层均为128维，输出层10个结点，如下图所示：



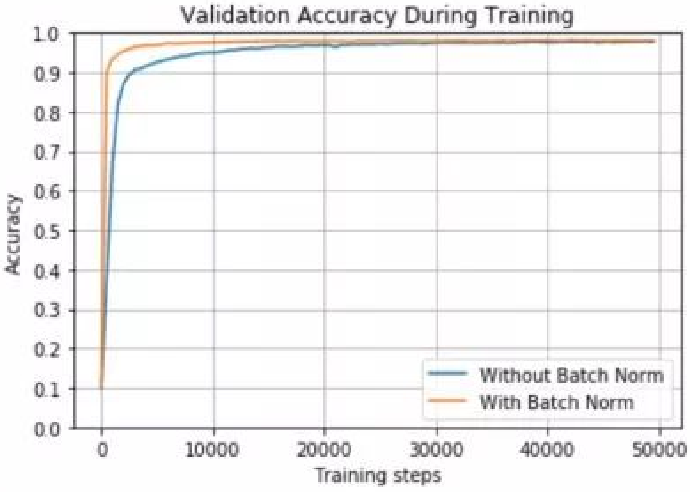
实验中，我们主要控制一下三个变量：

- 权重矩阵（较小初始化权重，标准差为0.05；较大初始化权重，标准差为10）
- 学习率（较小学习率：0.01；较大学习率：2）
- 隐层激活函数（relu，sigmoid）

2.1 小权重，小学习率，ReLU

测试结果如下图：

```
【Training Result:】

100%|██████████| 50000/50000 [01:43<00:00, 483.61it/s]
Without Batch Norm: The final accuracy on validation data is 0.977400004863739
100%|██████████| 50000/50000 [02:31<00:00, 329.23it/s]
With Batch Norm: The final accuracy on validation data is 0.9800000190734863

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.9746000170707703
With Batch Norm: The final accuracy on test data is 0.9800999760627747
```

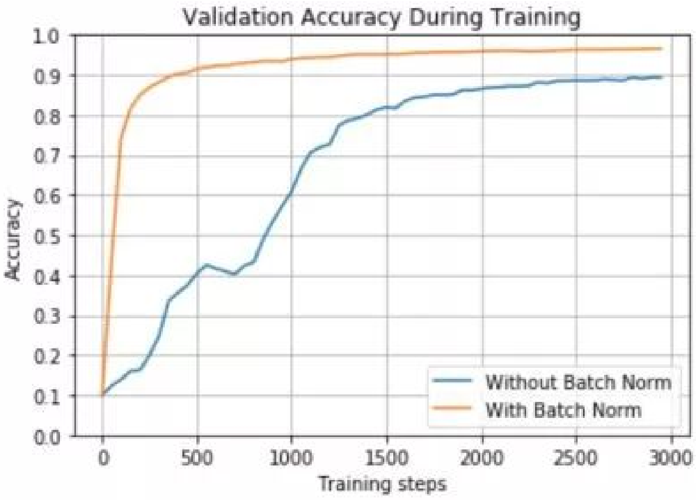


我们可以得到以下结论：

- 在训练与预测阶段，加入BN的模型准确率都稍高一点；
- 加入BN的网络收敛更快（黄线）
- 没有加入BN的网络训练速度更快（483.61it/s>329.23it/s），这是因为BN增加了神经网络中的计算量

为了更清楚地看到BN收敛速度更快，我们把减少Training batches，设置为3000，得到如下结果：

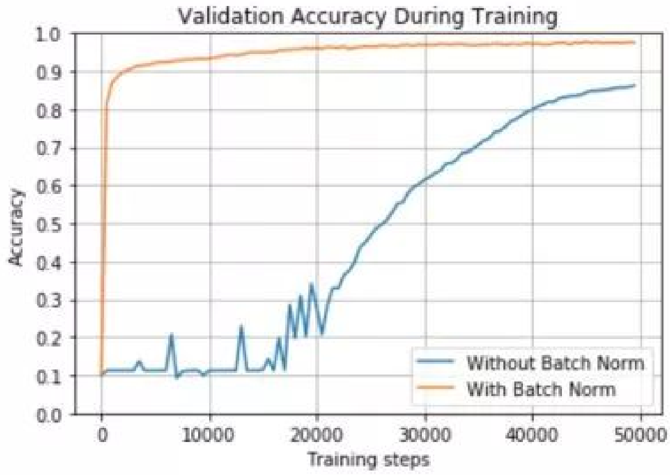
```
【Training Result:】

100%|██████████| 3000/3000 [00:06<00:00, 429.97it/s]
Without Batch Norm: The final accuracy on validation data is 0.8934000134468079
100%|██████████| 3000/3000 [00:11<00:00, 265.27it/s]
With Batch Norm: The final accuracy on validation data is 0.9652000069618225

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.8906000256538391
With Batch Norm: The final accuracy on test data is 0.9596999883651733
```



从上图中我们就可以清晰看到，加入BN的网络在第500个batch的时候已经能够在validation数据集上达到90%的准确率；而没有BN的网络的准确率还在不停波动，并且到第3000个batch的时候才达到90%的准确率。

2.2 小权重，小学习率，Sigmoid

```
【Training Result:】

100%|██████████| 50000/50000 [01:44<00:00, 480.48it/s]
Without Batch Norm: The final accuracy on validation data is 0.8615999817848206
100%|██████████| 50000/50000 [02:28<00:00, 335.98it/s]
With Batch Norm: The final accuracy on validation data is 0.9746000170707703

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.8569999933242798
With Batch Norm: The final accuracy on test data is 0.9700000286102295
```



学习率与权重均没变，我们把隐层激活函数换为sigmoid。可以发现，BN收敛速度非常之快，而没有BN的网络前期在不断波动，直到第20000个train batch以后才开始进入平稳的训练状态。

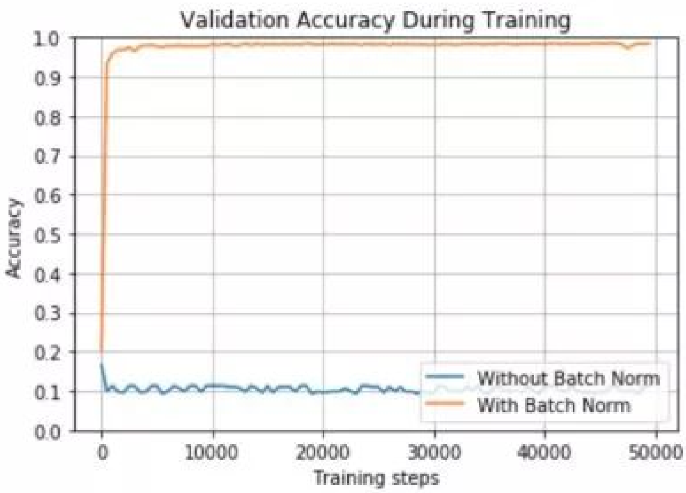
2.3 小权重，大学习率，ReLU

【Training Result:】

```
100%|██████████| 50000/50000 [01:43<00:00, 481.45it/s]
Without Batch Norm: The final accuracy on validation data is 0.11259999871253967
100%|██████████| 50000/50000 [02:34<00:00, 323.11it/s]
With Batch Norm: The final accuracy on validation data is 0.9846000075340271
```

【Testing Result:】

```
Without Batch Norm: The final accuracy on test data is 0.0982000008225441
With Batch Norm: The final accuracy on test data is 0.9804999828338623
```



在本次实验中，我们使用了较大的学习率，较大的学习率意味着权重的更新跨度很大，而根据我们前面理论部分的介绍，BN不会受到权重scale的影响，因此其能够使模型保持在一个稳定的训练状态；而没有加入BN的网络则在一开始就由于学习率过大导致训练失败。

2.4 小权重，大学习率，Sigmoid

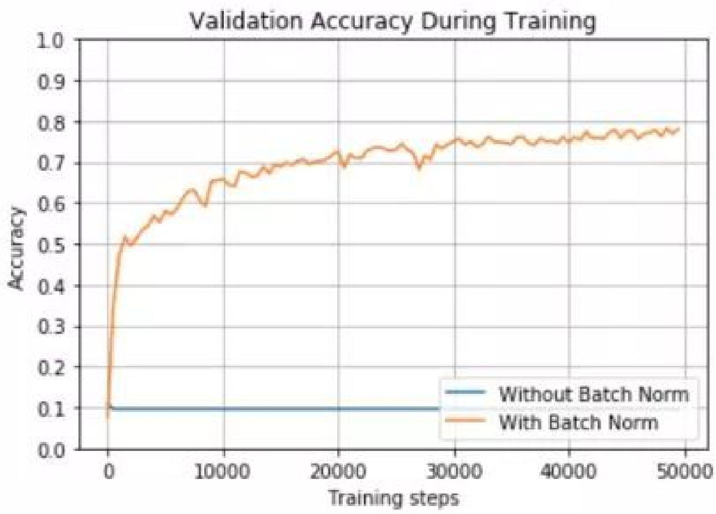
在保持较大学习率（learning rate=2）的情况下，当我们将激活函数换为sigmoid以后，两个模型都能够达到一个很好的效果，并且在test数据集上的准确率非常接近；但加入BN的网络要收敛地更快，同样的，我们来观察3000次batch的训练准确率。

```
【Training Result:】

100%|██████████| 3000/3000 [00:06<00:00, 433.74it/s]
Without Batch Norm: The final accuracy on validation data is 0.9657999873161316
100%|██████████| 3000/3000 [00:11<00:00, 261.68it/s]
With Batch Norm: The final accuracy on validation data is 0.9700000286102295

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.9606000185012817
With Batch Norm: The final accuracy on test data is 0.9631999731063843
```

当我们把training batch限制到3000以后，可以发现加入BN后，尽管我们使用较大的学习率，其仍然能够在大约500个batch以后在validation上达到90%的准确率；但不加入BN的准确率前期在一直大幅度波动，到大约1000个batch以后才达到90%的准确率。

2.5 大权重，小学习率，ReLU

```
【Training Result:】

100%|██████████| 50000/50000 [01:31<00:00, 544.76it/s]
Without Batch Norm: The final accuracy on validation data is 0.0957999974489212
100%|██████████| 50000/50000 [02:20<00:00, 356.07it/s]
With Batch Norm: The final accuracy on validation data is 0.7802000045776367

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.09799999743700027
With Batch Norm: The final accuracy on test data is 0.7728000283241272
```



当我们使用较大权重时，不加入BN的网络在一开始就失效；而加入BN的网络能够克服如此bad的权重初始化，并达到接近80%的准确率。

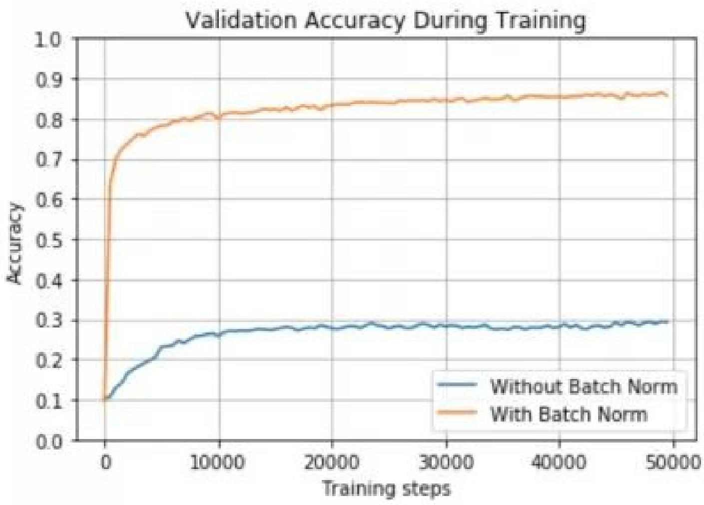
2.6 大权重，小学习率，Sigmoid

【Training Result:】

```
100%|██████████| 50000/50000 [01:42<00:00, 489.51it/s]
Without Batch Norm: The final accuracy on validation data is 0.2919999957084656
100%|██████████| 50000/50000 [02:38<00:00, 314.49it/s]
With Batch Norm: The final accuracy on validation data is 0.8569999933242798
```

【Testing Result:】

```
Without Batch Norm: The final accuracy on test data is 0.2897000014781952
With Batch Norm: The final accuracy on test data is 0.8496000170707703
```



同样使用较大的权重初始化，当我们激活函数为sigmoid时，不加入BN的网络在一开始的准确率有所上升，但随着训练的进行网络逐渐失效，最终准确率仅有30%；而加入BN的网络依旧出色地克服如此bad的权重初始化，并达到接近85%的准确率

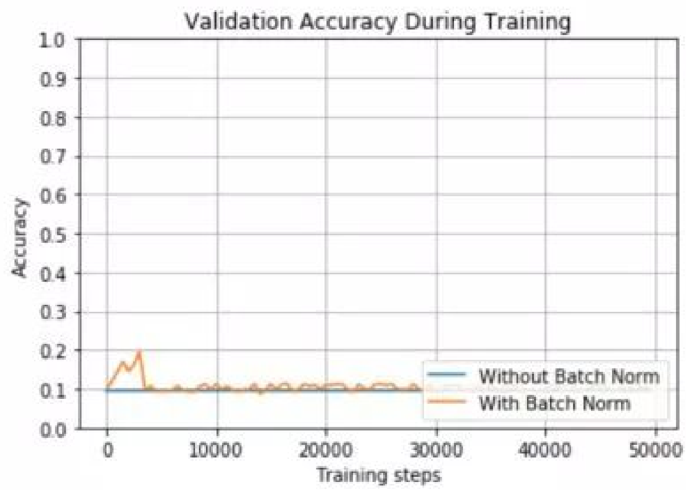
2.7 大权重，大学习率，ReLU

【Training Result:】

```
100%|██████████| 50000/50000 [01:38<00:00, 507.39it/s]
Without Batch Norm: The final accuracy on validation data is 0.0957999974489212
100%|██████████| 50000/50000 [02:32<00:00, 327.77it/s]
With Batch Norm: The final accuracy on validation data is 0.10999999940395355
```

【Testing Result:】

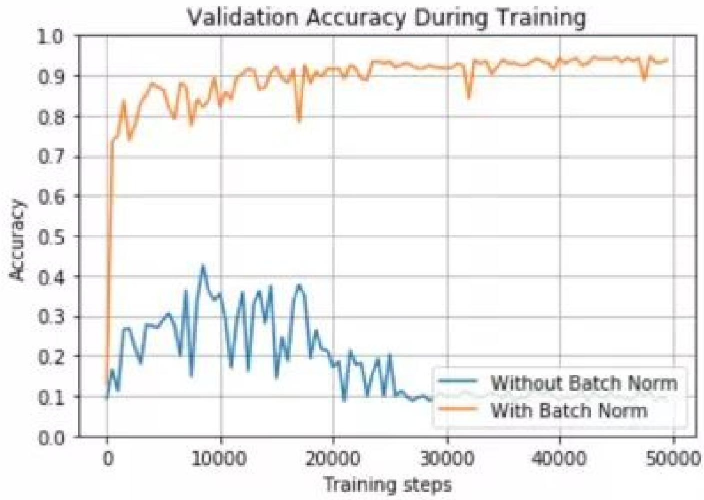
```
Without Batch Norm: The final accuracy on test data is 0.09799999743700027
With Batch Norm: The final accuracy on test data is 0.09740000218153
```



当权重与学习率都很大时，BN网络开始还会训练一段时间，但随后就直接停止训练；而没有BN的神经网络开始就失效。

2.8 大权重，大学习率，Sigmoid

```
【Training Result:】

100%|██████████| 50000/50000 [01:41<00:00, 494.57it/s]
Without Batch Norm: The final accuracy on validation data is 0.0868000015616417
100%|██████████| 50000/50000 [02:32<00:00, 328.44it/s]
With Batch Norm: The final accuracy on validation data is 0.9387999773025513

【Testing Result:】

Without Batch Norm: The final accuracy on test data is 0.09799999743700027
With Batch Norm: The final accuracy on test data is 0.9354000091552734
```



可以看到，加入BN对较大的权重与较大学习率都具有非常好的鲁棒性，最终模型能够达到93%的准确率；而未加入BN的网络则经过一段时间震荡后开始失效。

8个模型的准确率统计如下：

train batch=50000	权重矩阵	学习率	激活函数	Without BN		With BN	
				Acc on Val	Acc on Test	Acc on Val	Acc on Test
模型1	scale=0.05	0.01	ReLU	97.74%	97.46%	98.00%	98.01%
模型2	scale=0.05	0.01	Sigmoid	86.16%	85.70%	97.46%	97.00%
模型3	scale=0.05	2	ReLU	11.26%	9.82%	98.46%	98.05%
模型4	scale=0.05	2	Sigmoid	98.12%	97.84%	98.30%	98.02%
模型5	scale=10	0.01	ReLU	9.58%	9.80%	78.02%	77.28%
模型6	scale=10	0.01	Sigmoid	29.20%	28.97%	85.70%	84.96%
模型7	scale=10	2	ReLU	9.58%	9.80%	11.00%	9.74%
模型8	scale=10	2	Sigmoid	8.68%	9.80%	93.88%	93.54%

知乎：天雨粟
专栏：机器不学习

总结

至此，关于Batch Normalization的理论与实战部分就介绍道这里。总的来说，BN通过将每一层网络的输入进行normalization，保证输入分布的均值与方差固定在一定范围内，减少了网络中的Internal Covariate Shift问题，并在一定程度上缓解了梯度消失，加速了模型收

敛；并且BN使得网络对参数、激活函数更加具有鲁棒性，降低了神经网络模型训练和调参的复杂度；最后BN训练过程中由于使用mini-batch的mean/variance作为总体样本统计量估计，引入了随机噪声，在一定程度上对模型起到了正则化的效果。

参考资料：

[1] Ioffe S, Szegedy C. Batch normalization: accelerating deepnetwork training by reducing internal covariate shift[C]// InternationalConference on International Conference on Machine Learning. JMLR.org,2015:448-456.

[2] 吴恩达Cousera Deep Learning课程

[3] 详解深度学习中的Normalization，不只是BN

[4] 深度学习中 Batch Normalization为什么效果好？

[5] Udacity DeepLearning Nanodegree

[6] Implementing Batch Normalization in Tensorflow

END