Batch Normalization分析与实现

原创 空字符 月来客栈 1周前

收录于话题
#《跟我一起深度学习》

28个

由于公众号改版不再按照作者的发布时间进行推送，为防止各位朋友错过月来客栈推送的最新文章，大家可以手动将公众号设置为“星标⋆”以第一时间获得推送内容，感谢各位~

1 前言

各位朋友大家好，欢迎来到月来客栈。在上一篇文章中，笔者详细介绍了Batch Normalization的原理和计算流程。但是有句话叫做“数无形时少直觉，形少数时难入微”（不用猜了，我是宇哥的弟子），换到研究论文这个角度就是说，看论文只能帮助我们对于其中的思想原理以及大致的流程有一个认识，而想彻底弄懂其中的细节之处，那就必须得用代码才能体现。因为同一句描述，在不同人眼里可能会有不同的理解，但是换成代码之后这些歧义也就不复存在了。

在接下来的这篇文章中，就让我们一起来看一看如何通过代码来实现Batch Normalization，以及如何将它运用到其它的网络模型中。

扫码回复“加群”即可进入月来客栈交流群！

2 BN的实现与运用

2.1 BN的实现

在正式实现之前，我们先来简单的回顾一下BN的算法流程，以便于在实现的时候设置对应的参数。

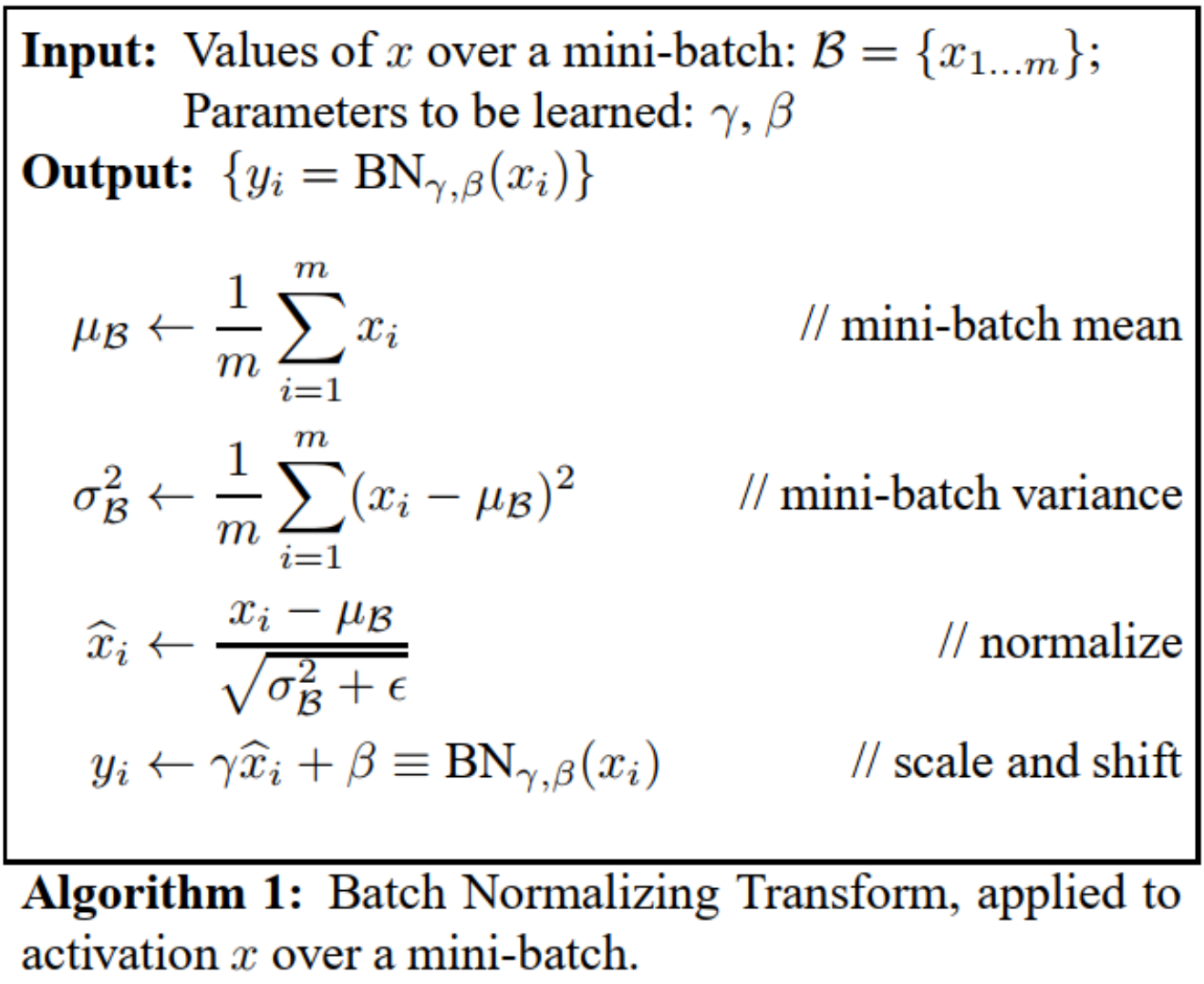


图 1. BN算法流程图

如图1所示就是BN算法的流程图：①从图中可以看出其一共包含有五个参数，分别是$\mu_{\mathcal{B}}, \sigma^2_{\mathcal{B}}, \epsilon, \gamma, \beta$；②为了同时对每一层所有的神经元一起进行标注化还需要一个 `num_features` 的参数；③为了同时满足对于全连接层和卷积层的标准化，因此还需要一个维度信息 `num_dims` 参数；④最后，由于要实现移动平均，因此还有一个 `momentum` 参数。

根据上面的介绍，我们便可以定义出如下所示的初始化函数：

```
class BatchNormalization(nn.Module):
    def __init__(self,
                 num_features=None, # 特征维度（全连接层中神经元的个数或者卷积中特征图通道的个数）
                 num_dims=4,# 用于区分当前是对全连接层标准化还是卷积层
                 momentum=0.9,# 移动平均中的控制参数
                 eps=1e-5):# 平滑处理
        super(BatchNormalization, self).__init__()
```

```
        shape = [1, num_features]
        if num_dims == 4:
            shape = [1, num_features, 1, 1]
            # 由于后面标准化时是以每个特征图为单位进行计算，因此会用到广播机制，所以需要保持4个维度
        self.momentum = momentum
        self.num_features = num_features
        self.eps = eps
        self.gamma = nn.Parameter(torch.ones(shape))
        self.beta = nn.Parameter(torch.zeros(shape))

        self.register_buffer('moving_mean', torch.zeros(shape))
        self.register_buffer('moving_var', torch.zeros(shape))
```

其中 nn.Parameter() 表示定义一个模型参数，然后将其加入到该模型对应的参数列表中，同时它有一个重要的属性就是可训练，即 requires_grad=True ； register_buffer() 为从 nn.Module() 中继承的方法，用于注册一个不可训练但同属为模型一部分的参数（不过用普通的成员变量也可以）。

接下来我们就可以根据BN的算法流程来完成余下部分的代码：

```
def forward(self, inputs):
    X = inputs
    if len(X.shape) not in (2, 4):
        raise ValueError("only support dense or 2dconv")

    if self.training:
        if len(X.shape) == 2:  # 全连接
            mean = torch.mean(X, dim=0)  #
            var = torch.mean((X - mean) ** 2, dim=0)
        else:  # 2d卷积
            mean = torch.mean(X, dim=[0, 2, 3], keepdim=True)
            # 在通道上做均值, [1,self.num_features,1,1]
            var = torch.mean((X - mean) ** 2, dim=[0, 2, 3], keepdim=True)
            # 在通道上求方差, [1,self.num_features,1,1]
        X_hat = (X - mean) / torch.sqrt(var + self.eps)
        self.moving_mean = self.momentum * self.moving_mean + (1.0 - self.momentum) * mean
        self.moving_var = self.momentum * self.moving_var + (1.0 - self.momentum) * var
    else:
        X_hat = (X - self.moving_mean) / torch.sqrt(self.moving_var + self.eps)
    Y = self.gamma * X_hat + self.beta
    return Y
```

其中上述代码主要分为两个部分：在训练过程中通过mini-batch计算得到的均值与方差来进行标准化；在推理过程中通过移动平均计算得到的均值与方差来进行标准化。

2.2 BN的运用

在实现完成BN后，我们就能够将其加入到任意的全连接层或者是卷积层之后了。下面我们以LeNet5网络为例进行示例：

```
class LeNet5BN(nn.Module):
    def __init__(self, ):
        super(LeNet5BN, self).__init__()
        self.conv = nn.Sequential(  # [n,1,28,28]
            nn.Conv2d(1, 6, 5, padding=2),  # in_channels, out_channels, kernel_size
            BatchNormalization(6, 4),
            nn.ReLU(),  # [n,6,24,24]
            nn.MaxPool2d(2, 2),  # kernel_size, stride  [n,6,14,14]
            nn.Conv2d(6, 16, 5),  # [n,16,10,10]
            BatchNormalization(16, 4),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)  # [n,16,5,5]
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(16 * 5 * 5, 120),
            BatchNormalization(120,2),
            nn.ReLU(),
            nn.Linear(120, 84),
            BatchNormalization(84, 2),
            nn.ReLU(),
            nn.Linear(84, 10)
        )

    def forward(self, img):
        output1 = self.conv(img)
        output2 = self.fc(output1)
        return output2
```

如下所示就是加入BN和未加入BN的LeNet5模型在FashionMNIST的结果：

```
# 加入BN
```

```
Epochs[1/20]--acc on test 0.8859
Epochs[2/20]--acc on test 0.8976
Epochs[3/20]--acc on test 0.9013
Epochs[4/20]--acc on test 0.904
Epochs[5/20]--acc on test 0.9023

#未加入BN
Epochs[1/20]--acc on test 0.8592
Epochs[2/20]--acc on test 0.8734
Epochs[3/20]--acc on test 0.8751
Epochs[4/20]--acc on test 0.881
Epochs[5/20]--acc on test 0.8895
```

3 BN的优点

在论文当中，作者也多次说到，在网络中插入BN：①能够使得网络更快的进行收敛；②能够使得每一层输入的特征图分布相对更加稳定；③在训练过程中能够使用更大学习率；④可以在一定程度上来缓解模型过拟合的现象。下面，我们就通过实验来分别对这四个方面的优点进行一个简单的验证。

3.1 更快的收敛

在论文中作者提到，由于加入BN后能够使得每一层输入的分布更加稳定，因此在训练过程中每一个网络层都能很快适应上一层的分布，所以这样便会加快网络的训练速度。在这个实验中，笔者选择了以加入BN和未加入BN的LeNet5网络模型为例进行对比。

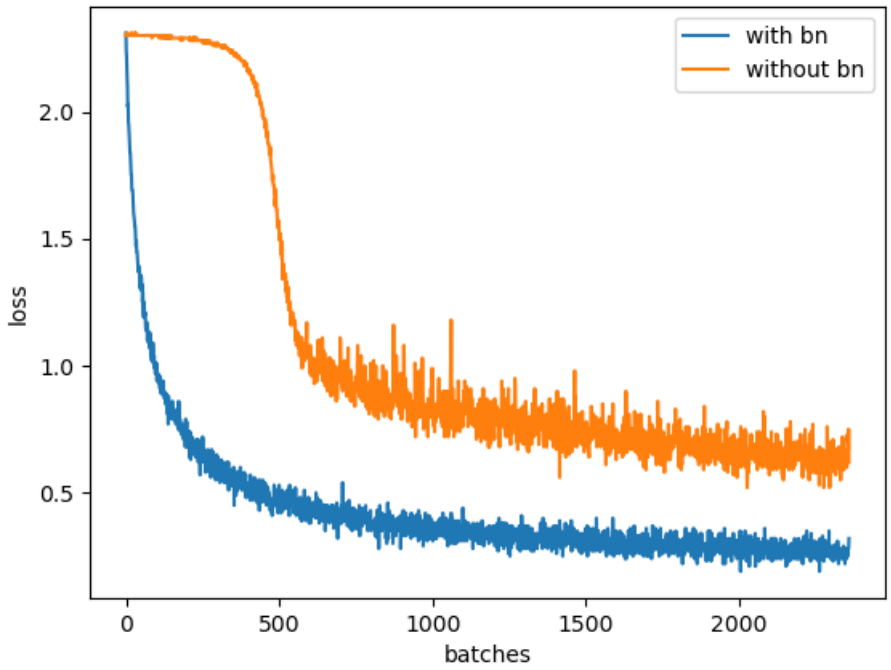


图 2. 收敛速度图

如图2所示，橙色曲线为未加入BN的Loss，而蓝色曲线为加入BN后的Loss。从图中可以明显的看出：①加入BN后的模型不仅能够更快的进入收敛状态；②同时也获得了更小的损失。

3.2 更稳定的特征分布

由于在加入BN后，每一层特征在输入到下一层网络之前都会进行标准化，所以每一层输入特征图的分布总体上都会更加的稳定。在这个实验中笔者使用了加入BN和未加入BN的4层卷积网络（由LeNet5变换而来）为例来进行对比。

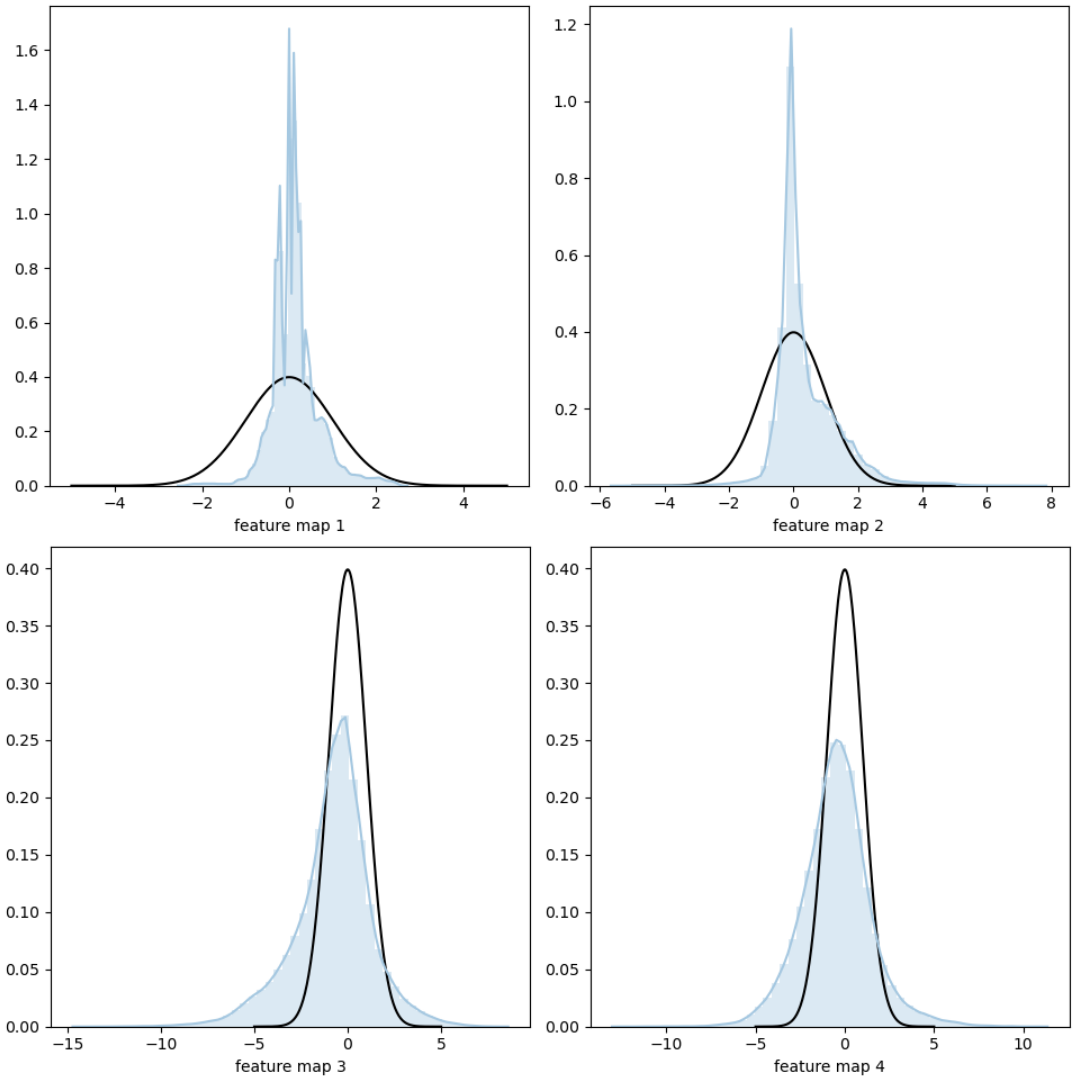
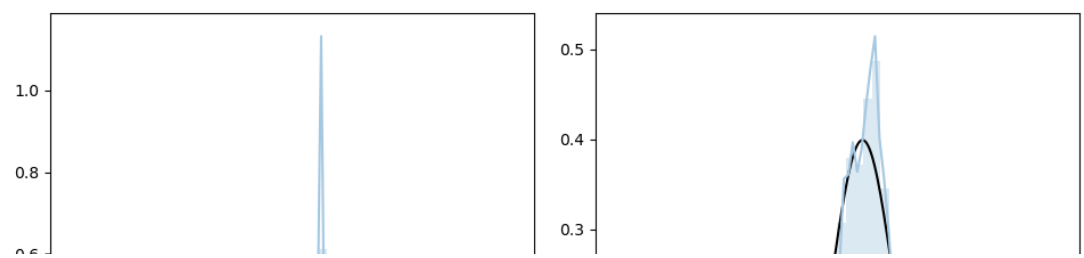


图 3. 特征分布图（without bn）

如图3所示，从左到右的蓝色部分为分别为四个卷积层输出结果的特征分布，而黑色曲线为标准的正态分布曲线。可以看到，每一层输出结果的特征图在分布上都有很大的差异，而这也就意味着每一层接收的输入的分布都存在着很大的差异，所以导致了作者所描述的ICS现象。



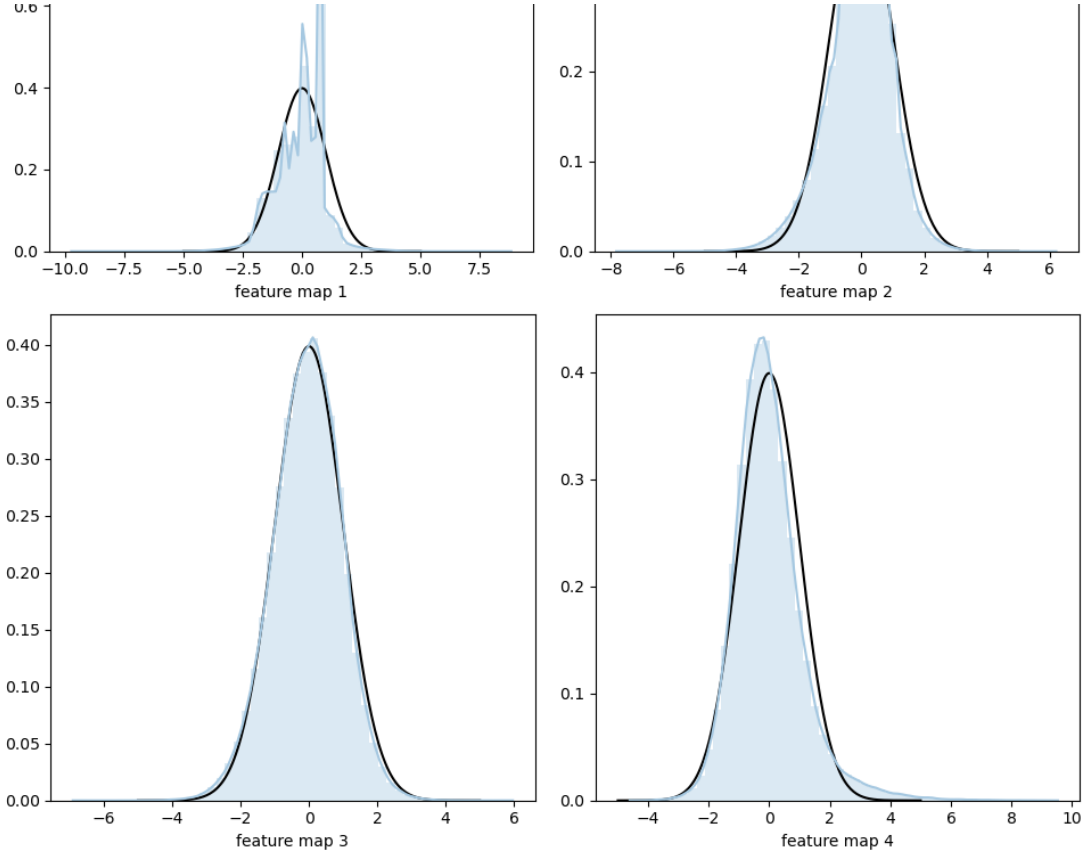


图 4. 特征分布图（with bn）

如图4所示，从左到右的蓝色部分同样为四个卷积层输出结果（BN处理后）的特征分布，黑色曲线也表示标准的正态分布曲线。可以发现，由于对每层的输出结果进行了BN操作，使得每一层的特征分布都比较接近于真实的正态分布。这就意味着每一层网络所接收到的输入都具有较为稳定的特征分布，从而消除了ICS现象。

3.3 更大的学习率

在传统深度学习的训练过程中，过大的学习率往往可能导致梯度爆炸（gradients explode）、梯度消失（gradients vanish）或者是陷入局部最优解的现象。但是由于BN在很大程度上解决了深度网络模型中的ICS现象，因此这就使得加入BN后的网络模型对权重参数变化有了更强的适应能力，从而可以使用更大的学习率来加速网络的收敛。在这个实验中笔者使用的依旧是加入BN和未加入BN的LeNet5网络模型。

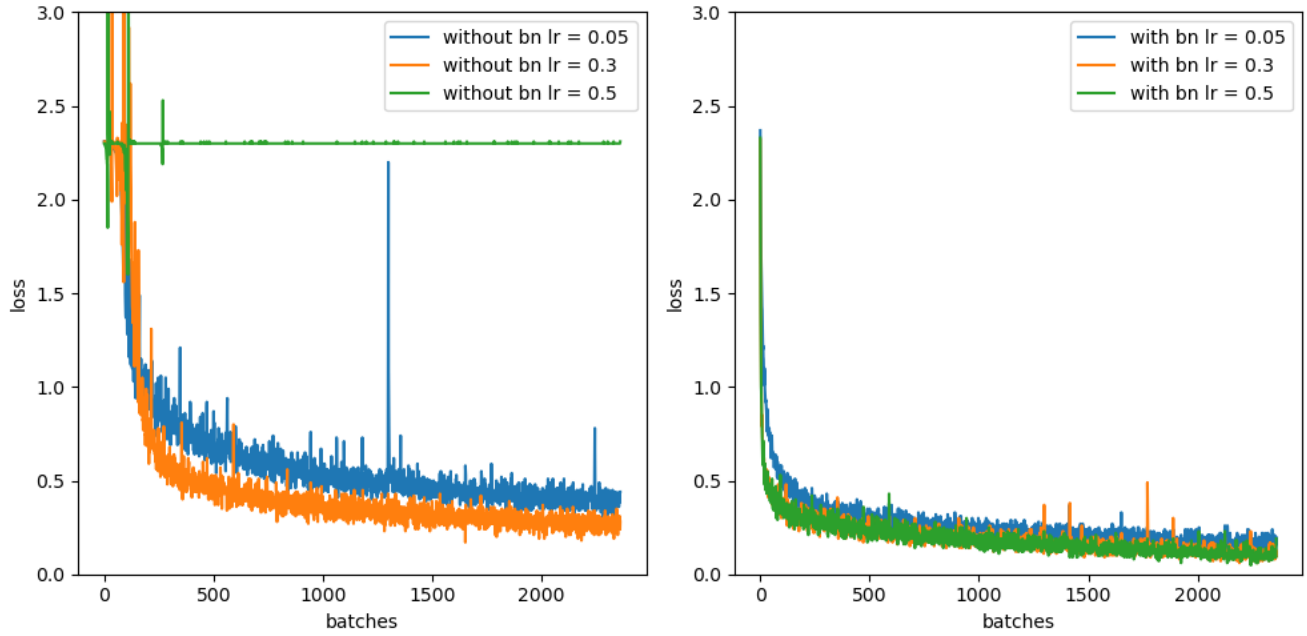


图 5. 不同学习率下的Loss曲线图

如图5所示，左右两边均为不同学习率下的loss曲线图。不同的地方在于左边的图没有加入BN，而右边的图加入了BN。可以明显的看到，在没有加入BN的模型中，当学习率加大到0.5时，网络的损失值不降反升了，大约在2.25附近达到了收敛。相反，加入了BN操作了模型在学习率增大到0.5时损失值得到了更小的收敛，并且还变得更加稳定。

3.4 更好的泛化能力

由于BN操作在对特征进行标准化时是基于整个mini-batch的均值和方差进行的，因此这就使得网络模型很难再受到单个样本的影响，也就是对个别异常样本不再那么敏感。所以从一定程度上来说加入BN后的网络模型能够有更好的泛化能力。在这个实验中，笔者采用的是加入BN（同时去掉了Dropout）和未加入BN的AlexNet模型，并在数据集CIFAR100进行了实验。

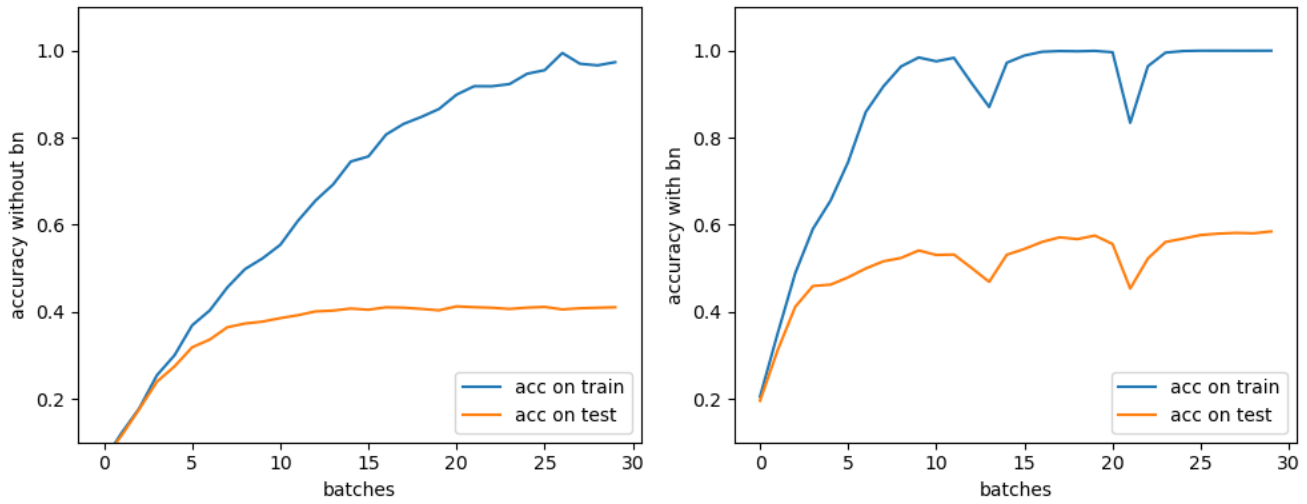


图 6. CIFAR100测试结果图

如图6所示，左边为原始的AlexNet在CIFAR100上100个Epochs后的结果。可以看到，模型在训练集上的准确率几乎已经要接近于1了（最高0.973），但是在测试集上的准确率才刚刚到达0.4附近，这也就意味着此时的模型出现了严重的过拟合现象。图6右边的部分是加入BN操作的

AlexNet模型，可以看到其在训练集上的准确率相较于未加入BN的模型还有一定的提升（最高0.999）。同时，虽然此时该模型仍旧处于过拟合的状态，但是其在测试集上的准确率还是得到了很大的提升达到了0.58左右。这足以见得BN确实能够在一定程度上提高模型的泛化能力。

4 总结

在这篇文章中，笔者首先介绍了如何用代码来实现BN操作；然后介绍了如何在已有的网络模型中加入BN操作；最后通过实验的方式来验证了BN的优点，包括可以加快网络的收敛速度、在训练过程中能够使用更大的学习率以及拥有更好的泛化能力等特性。在下一篇文章中，我们将开始学习深度中最有名的网络结构之一ResNet。

本次内容就到此结束，感谢您的阅读！如果你觉得上述内容对你有所帮助，欢迎分享至一位你的朋友！若有任何疑问与建议，请添加笔者微信'nulls8'或加群进行交流。青山不改，绿水长流，我们月来客栈见！

推荐阅读

[1]不得不说的Batch Normalization

[2]GoogLeNet介绍与实现

[3]厉害了，能把多尺度卷积说得这么高大上

[4]NiN一个放到现在也不过时的网络

[5]VGG一个可使用重复元素的网络

收录于话题 #《跟我一起深度学习》·28个

下一篇 · 不得不说的Batch Normalization

喜欢此内容的人还喜欢

万字完整带你深入解析JVM 面试必备

风平浪静如码

详细实战教程！部署Flask网站+域名访问+免费https证书

Python爬虫数据分析挖掘