哈尔滨工业大学

<<模式识别与深度学习>>

实验 1 基于 Pytorch 的 MLE 实现

(2020 春季学期)

学院：计算机科学与技术

学号：1170300909

姓名：武磊

指导老师：左旺孟

日期：2020.4.21

一、实验目的

1. 使用 Pytorch 实现 MLP（Multilayer Perceptron），并在 MNIST 数据集上验证
2. 熟悉 Pytorch 框架，了解 Ptorch 中自动求导，动态计算图的实现原理，为后续实验打下良好基础

二、实验环境配置

1. 硬件：

   CPU ： Inter Core i7-7500U 2.70GHz

   GPU ： NVIDIA GeForce 940MX
2. 软件：

   操作系统:WIN10

   软件：

   Python 3.6 (基于 Anaconda)

   Pytorch 1.2.0

   CUDA 10.1 V10.1.105

   cudnn v7.6.3 for CUDA 10.1
3. 开发 IDE

   Jupyter NoteBook

   Pycharm

三、实验过程

1. 数据导入

Pytorch 框架中的 torcvisio 包已经为我们内置了许多常用的图像处理的数据集，包括我们这次需要用到的 MNIST。

由于网速原因，直接联网下载数据集太慢了。我们将准备好的数据集放到 data 目录下。

1) 调用 dataset.MINST() 封装数据集，传入参数 transformer 对数据做预处理，这里主要是将原数据转换成 torch 框架中的 Tensor 变量以及对数据做归一正则化
2) 调用 Dataloader 类，方便训练的时候按照 batchsize 的大小取数据，也可以选择对数据做 shuffle 操作，使得训练得到的结果更接近真实的数据分布。
3) 对于自己收集的实验数据，可以通过继承 Dataset 的 MyDataset，重写 __getitem__和__len__方法，交由 Dataloader 进行数据读取，这样可以方便的利用 dataloader 的多线程和 shuffle 功能，不会出现数据一次性读到内存，内存爆炸，在面对大数据量训练的时候尤其有用。

2. 设计定义网络

多层感知机的网络设计非常简单，一层输入层，若干层隐含层（通常是一层），一层输出层。

本次实验中我们设置的是一个有两层隐含层的 MLE。

在 Pytorch 中定义网络结构是非常简单的，只需要新建一个网络类，继承自 nn.Module，完成网络层次设计和 forward 前向函数即可。

我们可以使用 nn.Sequential(),将我们的网络结构垛堞进去即可，具体示例如下：

```
class MLP(nn.Module):
    def __init__(self,input,hidden1,hidden2,output):
        super(MLP,self).__init__()
        self.in_dim = input
        self.hidden1 = hidden1
        self.hidden2 = hidden2
        self.out_dim = output
        self.classifier = nn.Sequential(
            nn.Linear(input,hidden1),
            # nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(hidden1,hidden2),
            # nn.Dropout(0.5),
            nn.ReLU(),
            nn.Linear(hidden2,output),
            # nn.Softmax()
        )

    def forward(self, din):
        din = din.view(-1, self.in_dim)
        return self.classifier(din)
```

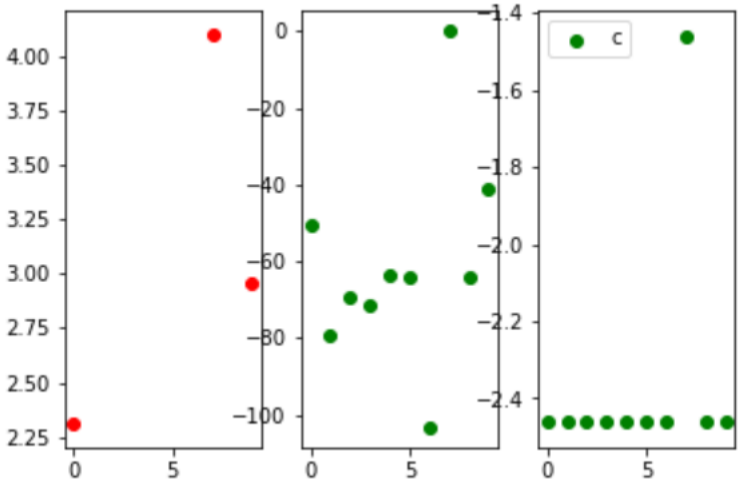
注释：在选用代价函数的时候，我们选 nn.CrossEntropyLoss()，但是 pytorch 内置实现过程就采用的 softmax 对数据做预处理。
这样就会遇到问题：究竟需不需要设计网络的时候最后采取 softmax 输出?
经过实验：

```
from matplotlib import pyplot as plt

cpu = torch.device("cpu")
for data, label in train_loader:
    data, label = data.cuda(), label.cuda()
    output = model(data)
    a = output[0].to(cpu).detach().numpy()
    print(a)
    b = nn.functional.softmax(output[0]).to(cpu).detach().numpy()
    print(b)
    c = nn.functional.softmax(nn.functional.softmax(output[0])).to(cpu).deta
ch().numpy()
```

```
    plt.subplot(131)
    plt.scatter(list(range(10)),np.log(a),label="a",color="red")
    plt.subplot(132)
    plt.scatter(list(range(10)),np.log(b),label="b",color='green')
    plt.subplot(133)
    plt.scatter(list(range(10)),np.log(c),label="c",color='green')

    plt.legend()
    _, predicted = torch.max(output.data,1)
    print(predicted)
    print(label)
    break
```



可以理解经过两层的 softmax，使得 loss 值始终多了 1.4 的偏置量，这会对最后反向传播更新参数产生影响，使得我们在一开始观察不到过拟合的现象。

3. 选择代价函数，超参数，优化器

1) 模型的超参数

MLE 层与层之间采用的是全连接，参数会随着节点数的增加爆炸式的增加，好在 Pytorch 在我们初始化模型的时候会自动初始化参数，我们可以通过调用 model.paramters()即可使用。

其他超参数：

a) Batch-size 大小：根据资料显示，对于小型数据集可以采用 full batch 学习的方式，这样每次求得梯度是对所有样本平均化后的结果，训练结果更加稳定，但是也会导致梯度下降过慢，陷入局部最优的情形。

采用 Mini-batch learning 可以利用样本的随机性，使得训练过程中 explorer 一些更优的梯度下降方向，更快的收敛。

总的来说：在 Batchsize 在一个临界值一下，对于网络性能的影响并

不大，远不如学习率的影响。

b) 网络结构的设置：按照“万能逼近定理”，越多的网络节点数表示的是越复杂的模型，理论上可以逼近任何模型。这里没有过多的花时间讨论。准备有时间以阅读论文为主。

本次实验节点结构（784-128-128-10）随机设置。

2) 代价函数选择

我们是在 MINST 数据集上验证结果，是对图片进行分类，故选用交叉熵函数，当然更加复杂的模型可以自己定义特别的代价函数，也可以在代价函数中加入正则项，防止模型的过拟合。

模型的正则化可以通过 optimizer 中 weight_decay 参数设置，一般选择 L2

调用方法： loss_func = nn.CrossEntropyLoss()

3) 优化器选择：

深度网络是通过误差反向传播来更新参数，达到学习的目的。优化器也主要分为两类：SGD 族和 Ada 族。

a) SGD 族是用过选择小批量样本进行随机梯度下降，我们寄希望于通过一些突出有效的小样本，能够帮助模型在 loss curve 中跳出平缓的区域，更快的达到收敛。随机化的样本也能够帮助 model 尽可能的少停留在 local minimal 的地方。

b) Ada 族的思想是，随着误差的反向传播，参数的更新，model 能够自适应的调整学习率，从而更好的达到收敛。

现在比较好的方法主要有 Adam 和 SGDM（SGD with momentum）

当然，不应该只局限在固定的优化方法之间单纯的选择，应该根据实际情况综合考虑。

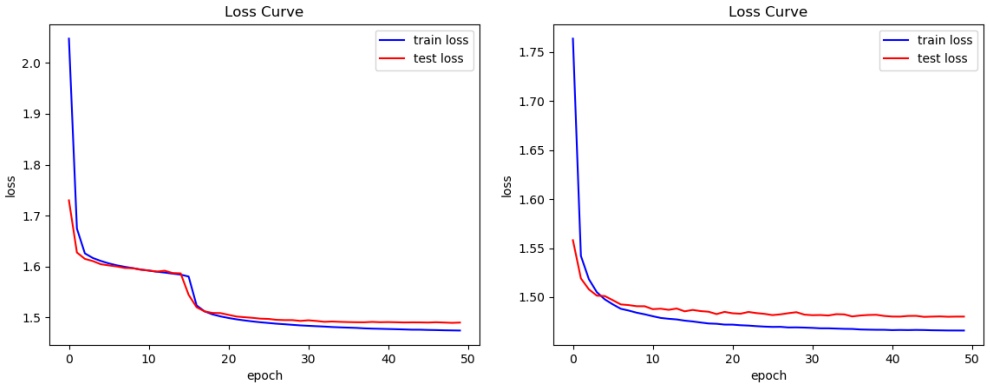
四、结果分析

1. 实现基本 MLE

如上述，本次实验中实现的是一个四层的多层感知机，网络结构如下：

```
MLP(
  (classifier): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=10, bias=True)
    (5): Softmax(dim=None)
  )
)
```

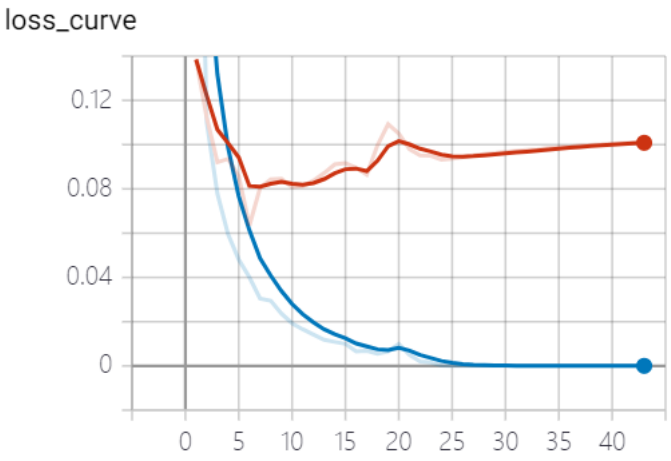
经过多次测试，四层的感知器网络在 MNIST 数据集上的测试准确度能达到 98%。网络在前 20 个 epoch 快速收敛，Testloss 和 Trainloss 快速下降，之后趋于停滞，模型的效果到达瓶颈。
实验参数和 loss curve 如下图所示：

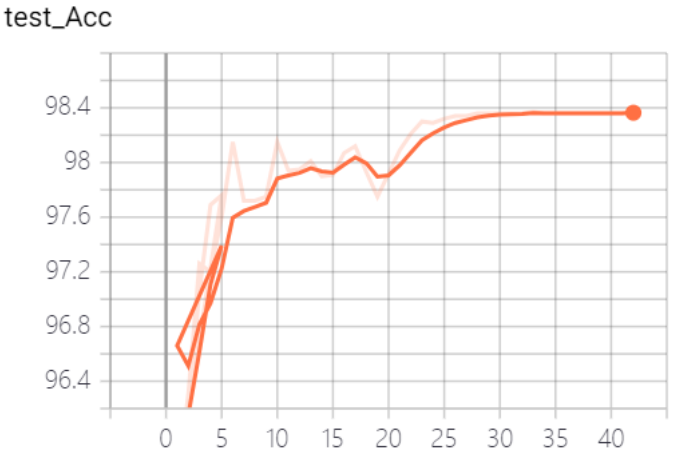


Epoch	BatchSize	Lr	Optimizer
50	64	0.001	SGD
50	64	0.001	SGDM

上述结果仅是单次的训练结果，具有随机性。但是我们可以看到当学习率设置过低的时候会导致使用随机梯度下降有几率陷入 local minimal，导致模型效果变差。
增加 epoch 虽然可以让模型尽可能的收敛，但是并不能保证模型能够跳出 local minimal 或者 loss curve 中的“高原平面”。我们还需要更好的设置学习率，调整梯度的方向，让模型训练的又快又好。

去除多余 softmax：出现了明显的过拟合，准确率也有所提升。

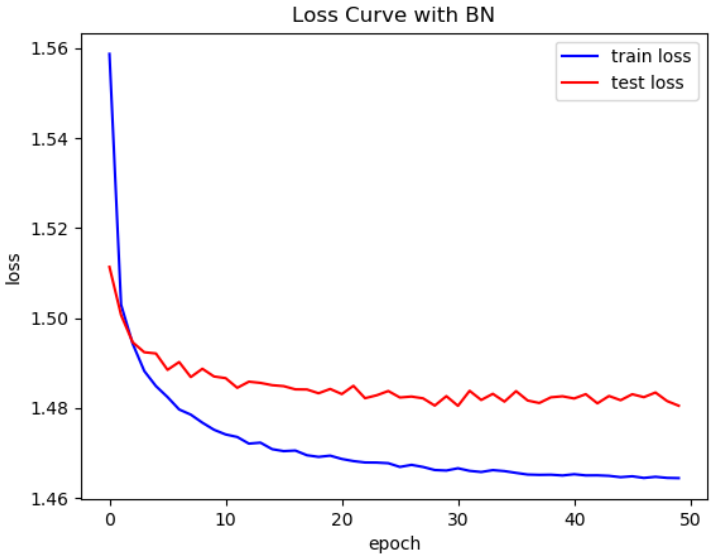




2. 增加批标准化

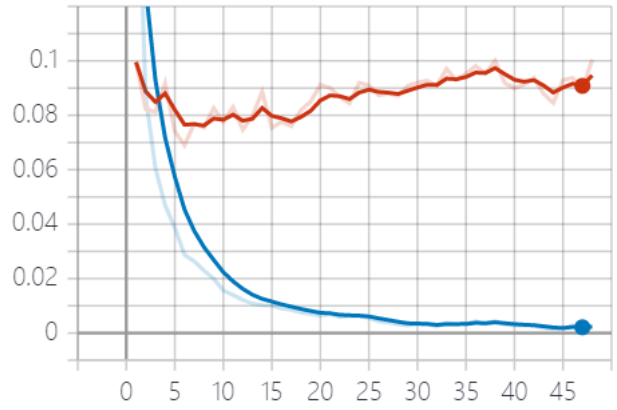
批标准化实际上是在每一层激活函数作用之前，对上一层的输出进行归一化操作，让前一层的输出归一化到了 0 均值，1 方差，保证了梯度的有效性。

下面是使用了 BatchNomalization 的 loss curve 曲线。

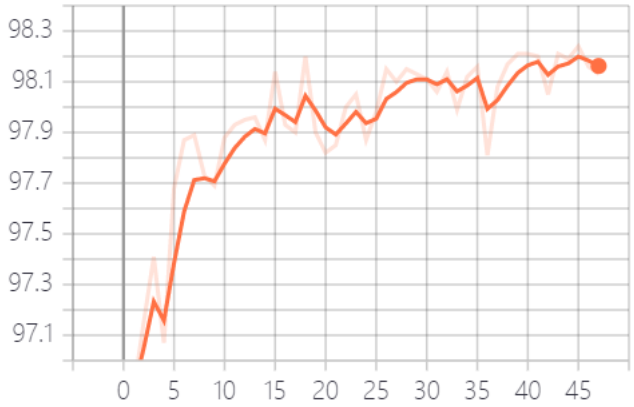


去除 softmax：奇怪的是这个时候加了 Batch-normalization,过拟合现象和实验效果都不好，很疑惑，没想到合理的解释。

loss_curve



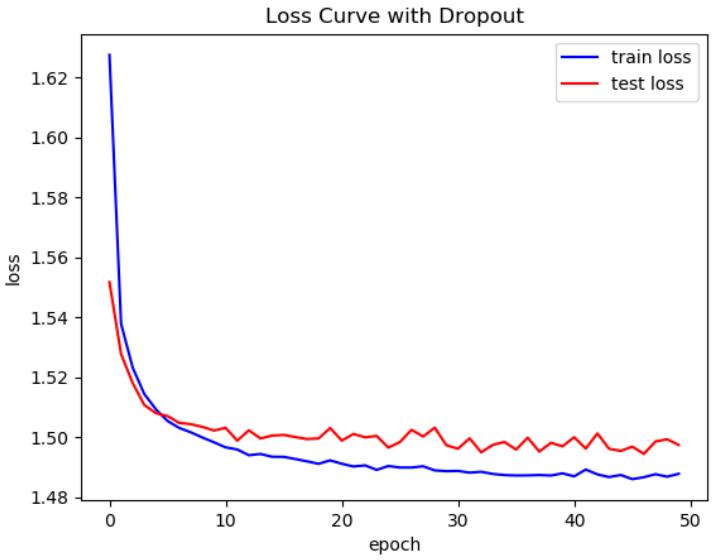
test_Acc



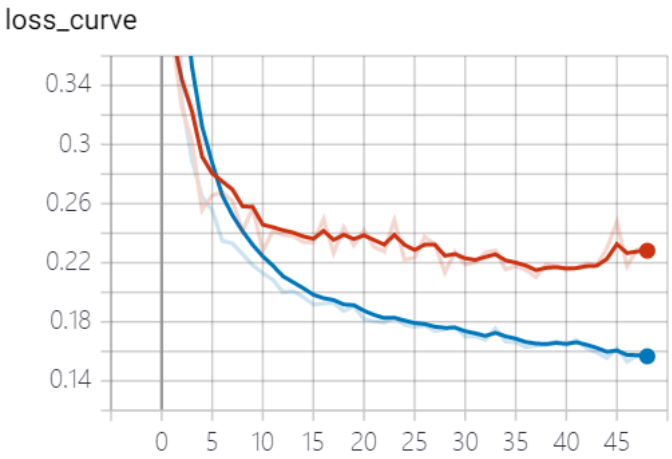
3. 增加 dropout

Dropout 也是一种提升网络泛化能力的方法。最初的想法来自与模型平均，考虑在相同网络结构的情况下，随机的去掉一些连接权值和节点。通过避免模型对于某些特殊节点的过度依赖，随机让网络中的节点“失活”，从而提升模型的泛化能力。

下面是使用了 Dropout 的 loss curve。



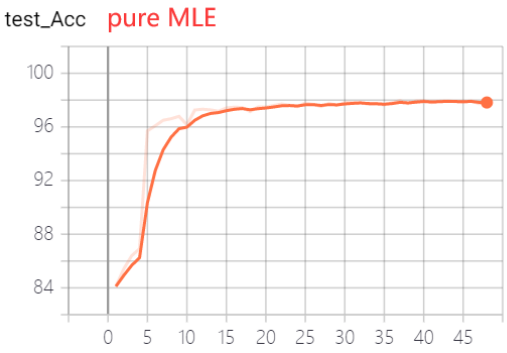
去除 softmax：发现 dropout 能够有助于抑制过拟合的现象，到 35-40 个 epoch 的时候才出现 test_loss 上升的情况。

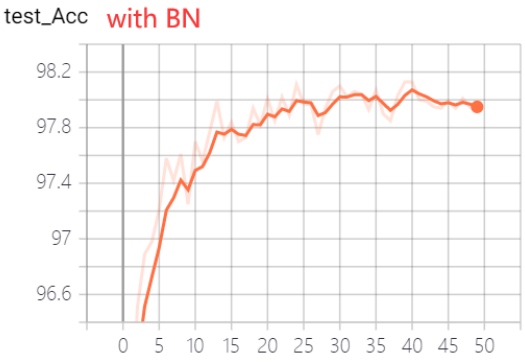


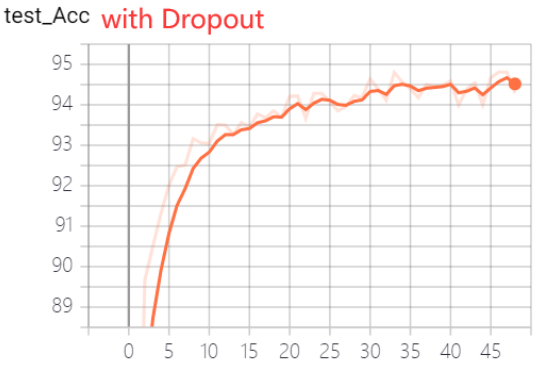
在该次实验中，我们发现使用的 Batch Normalization 的效果和 Pure_MLE 的 loss 曲线差不多，但是使用了 Dropout 的网络 loss 最后收敛在一个较大值上。三种情况的准确度如下：

可能的解释是四层的感知机还是属于浅层学习，Internal Covariant Shift 的效果不明显，同样由于这个原因，Dropout 会使得网络结构变简单，陷入欠拟合的转态。

网络（多一层 softmax）	Test Accuracy
Pure MLE	97.60%
Add BatchNormalization	98.04%
Add Dropout（0.5）	94.03%

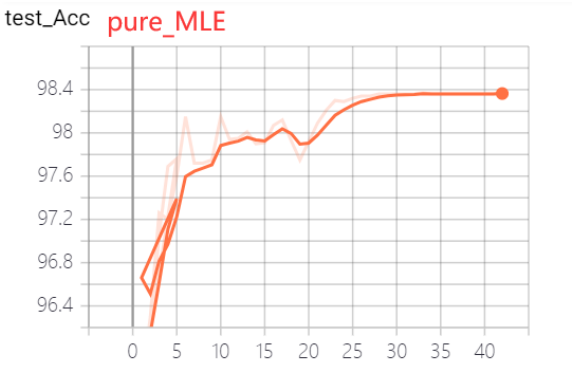


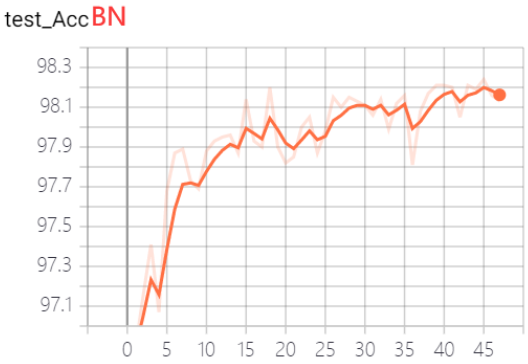


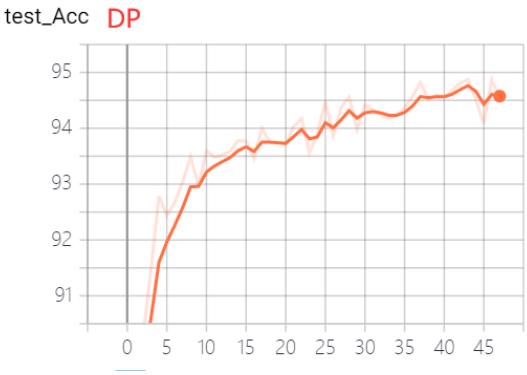


去除 softmax：

网络	Test Accuracy
Pure MLE	98.37%
Add BatchNormalization	98.21%
Add Dropout	94.87%







五、总结

1. 初步学习了如何使用 pytorch 框架建立一个四层的多层感知机，并且在 MNIST 数据集上验证，最好的 TestAccuracy 能够达到 98%（SGDM 优化器，20-30epoch，lr=0.05， momentum=0.9），基本达到了 SVM 能够在 MNIST 上的最好表现，可见神经网络强大的模型能力

2. 深度学习任务中设计到需要调整的超参数远远超过传统机器学习中的参数数量，所以需要知道一定的技巧和规律，可以通过多查阅论文文献，了解别人的验证结果。尤其在深度学习较之一般机器学习过高的训练测试代价，要多看多了解别人的测试和看法。

3. 深度学习任务中自制数据集是非常重要的一步，一般选择重写继承自 torch.utils.data.Dataset，这也可以直接在后续过程中使用 dataloder 接口，方便后续训练。

4. 最权威和方便的学习框架的方法就是看官方文档，这样对于框架的特性和功能都能有全面的理解。

5. 可以学习 torchvision 中 dataset 的实现方法，编码的时候尽量模块化，结构化，方便网络结构的调整。

6. 上述对于批正则化和 Dropout 的讨论并不完全，还需要在后续实验中多探索。

7. 多和同学讨论交流，softmax 的问题就是在和同学讨论的过程中发现的。

六、参考

[MLE pytorch 实现]https://cloud.tencent.com/developer/article/1110722
[Pytorch train trick] https://zhuanlan.zhihu.com/p/76459295
[Batch Normalization 作用] http://proceedings.mlr.press/v37/ioffe15.pdf
[TensorBoardX tutorial]
https://pytorch-tutorial.readthedocs.io/en/latest/tutorial/chapter04_advanced/4_2_2_tensorboardx/
[Pytorch tutorial] https://pytorch-tutorial.readthedocs.io/