南開大學
Nankai University

计算机学院

深度学习实验报告

前馈神经网络

姓名：杨馨仪

学号：2011440

专业：计算机科学与技术

2024 年 6 月 23 日

目录

1 实验要求 **2**

2 原始版本 MLP **2**

3 优化 MLP 网络 **3**

3.1 增加网络层数 3
3.2 增加网络宽度 4
3.3 增加批归一化层 5
3.4 调整优化器参数 6
3.5 更换优化器 7
3.6 最终改进后网络 7

4 ResMLP 实现 **9**

5 实验心得 **10**

1 实验要求

- 掌握前馈神经网络（FFN）的基本原理
- 学会使用 PyTorch 搭建简单的 FFN 实现 MNIST 数据集分类
- 掌握如何改进网络结构、调试参数以提升网络识别性能

2 原始版本 MLP

查看原始版本 MLP 代码。观察其网络定义：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 100)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(100, 80)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(80, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)    # [32, 28*28]
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)    # [32, 10]
        return F.log_softmax(self.fc3(x), dim=1)
```

也可以直接使用 print(model) 获取网络结构：

```
Net(
  (fc1): Linear(in_features=784, out_features=100, bias=True)
  (fc1_drop): Dropout(p=0.2, inplace=False)
  (fc2): Linear(in_features=100, out_features=80, bias=True)
  (fc2_drop): Dropout(p=0.2, inplace=False)
  (fc3): Linear(in_features=80, out_features=10, bias=True)
)
```

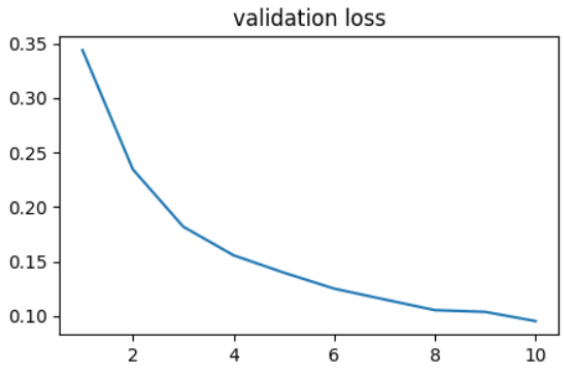
图 2.1: Caption

可知这个神经网络（Net）由三个全连接层组成，输入层将 28x28 的图像展平成 784 维向量。第一个全连接层有 100 个输出神经元，第二个有 80 个输出神经元，第三个有 10 个输出神经元，对应 10 个类别。每个全连接层后接 ReLU 激活函数和 20% 概率的 Dropout 层，最后通过 Log Softmax 层输出分类概率，用于图像分类任务。

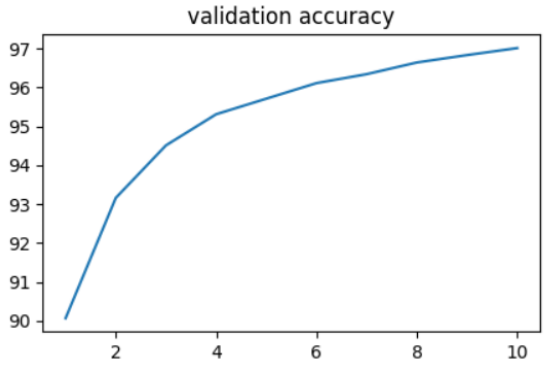
```
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
criterion = nn.CrossEntropyLoss()
```

这段代码定义了用于训练神经网络的优化器和损失函数，其中优化器使用带有 0.01 学习率和 0.5 动量的随机梯度下降法（SGD）来更新模型参数。损失函数则采用交叉熵损失（CrossEntropyLoss）来衡量模型预测值与实际标签之间的差距。

根据提供的代码，我们也可以得到其损失曲线和准确度曲线如下：



(a) 原始版本 MLP 的损失曲线



(b) 原始版本 MLP 的准确度曲线

3 优化 MLP 网络

3.1 增加网络层数

增加前馈神经网络层数可以提取不同层次的特征，拥有更高的表示能力；并且每一层通常都包含非线性激活函数，增加网络层数意味着增加更多的非线性变换，能够更好地拟合复杂的非线性关系。

因此本实验在原始 MLP 上尝试增加一层来提高模型的表示能力：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 100)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(100, 80)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(80, 50)  # 增加一个隐藏层
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
```

```
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        x = F.relu(self.fc3(x))  # 新增加的层
        x = self.fc3_drop(x)
        return F.log_softmax(self.fc4(x), dim=1)
```

经过训练，我们可以得到其与原网络结构的准确度曲线对比如下：

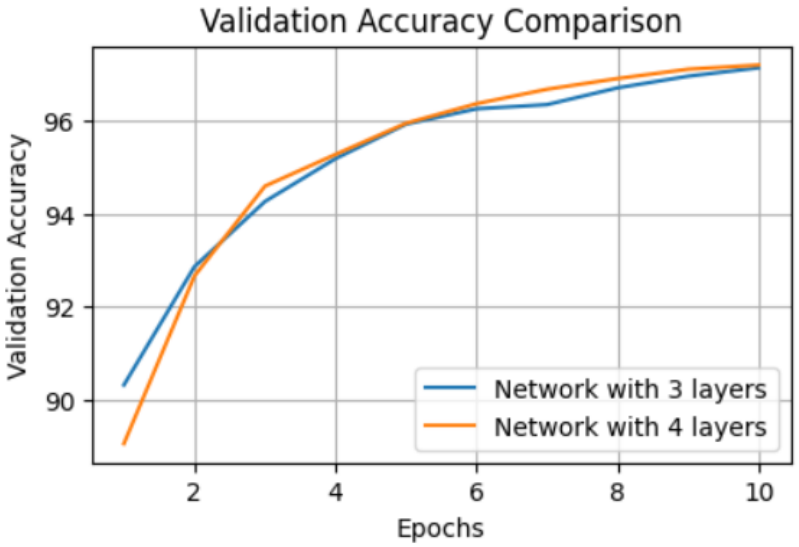


图 3.2: 不同网络深度验证集准确率对比图

可见增加网络层数可以提高模型的准确率，但在该任务下效果并不十分明显。而且也不能一味地增加层数，因为本实验的数据并不是很丰富，增加网络层数会增加模型的复杂度，如果训练数据不足或者正则化不足，可能导致过拟合，也有可能导致梯度消失或爆炸问题，影响训练效果。而且更深的网络需要更多的计算资源和训练时间，因此本实验仅增加了一层网络。

3.2 增加网络宽度

增加神经网络的宽度可以通过提升特征表达能力、改善模型的泛化能力、加快训练速度和改善梯度传播效率等方面，有助于提高神经网络在复杂任务上的准确率和性能。

直接在增加了一层隐藏层的网络基础上进行优化：

```
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 200)  # 增加第一个隐藏层的神经元数量
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(200, 150)    # 增加第二个隐藏层的神经元数量
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(150, 100)    # 新增加一个隐藏层
```

```
        self.fc3_drop = nn.Dropout(0.2)
        self.fc4 = nn.Linear(100, 10)     # 输出层，10 个类别
```

经过训练得到如下结果：

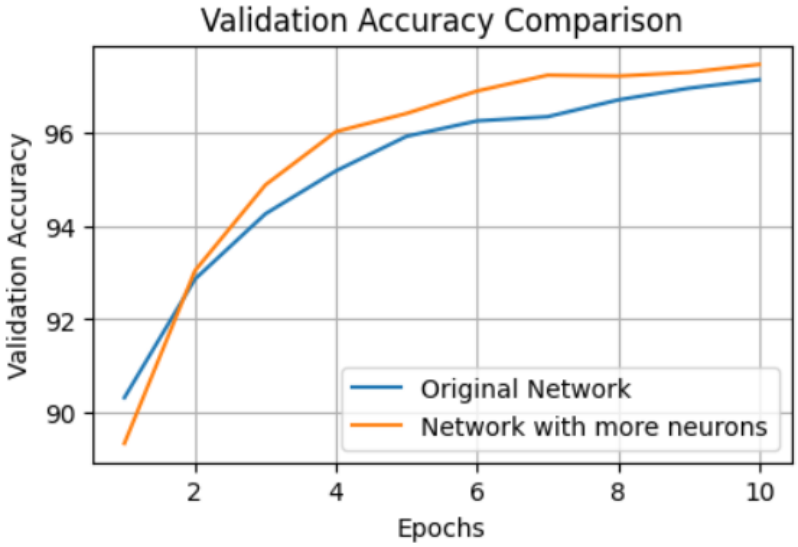


图 3.3: 不同网络宽度验证集准确率对比图

可以看到增加宽度对性能的提升也是有效的。然而，增加宽度也会增加网络的计算和存储成本，因此需要在应用中进行权衡和优化。

3.3 增加批归一化层

通过在神经网络的每个隐藏层后添加批归一化（Batch Normalization，BN）层，可以显著提高模型的性能和训练速度。批归一化层在训练过程中通过规范化每个小批量的输入，有助于加速训练过程并增强模型的泛化能力。

在前两步的基础上直接加上批归一化层：

```
class Net3(nn.Module):
    def __init__(self):
        super(Net3, self).__init__()
        self.fc1 = nn.Linear(28*28, 200)
        self.bn1 = nn.BatchNorm1d(200)  # 批归一化层
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(200, 150)
        self.bn2 = nn.BatchNorm1d(150)  # 批归一化层
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(150, 100)
        self.bn3 = nn.BatchNorm1d(100)  # 批归一化层
        self.fc3_drop = nn.Dropout(0.2)
```

```
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.bn1(self.fc1(x)))
        x = self.fc1_drop(x)
        x = F.relu(self.bn2(self.fc2(x)))
        x = self.fc2_drop(x)
        x = F.relu(self.bn3(self.fc3(x)))
        x = self.fc3_drop(x)
        return F.log_softmax(self.fc4(x), dim=1)
```

训练结果如下：

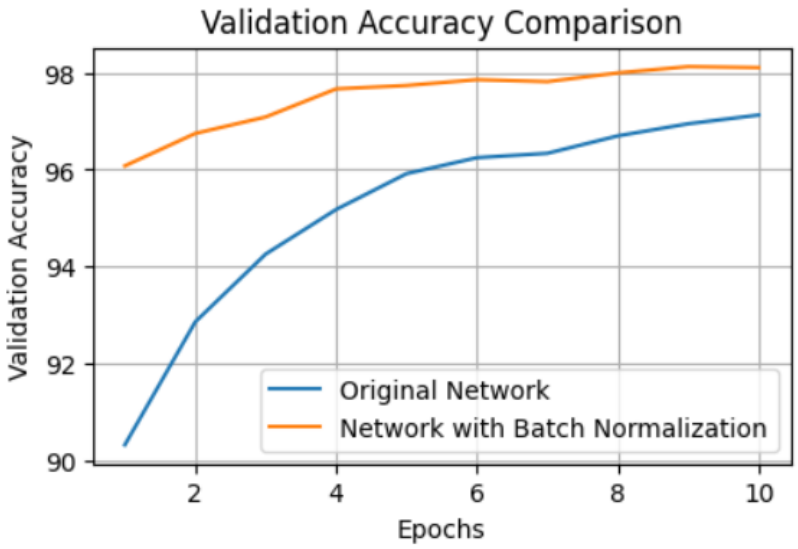


图 3.4: 增加批归一化层验证集准确率对比图

可以看到性能的确得到了明显提升，准确率超过了 98%。

3.4 调整优化器参数

调整学习率是优化神经网络模型性能中的重要步骤之一。学习率的选择对模型的收敛速度和最终的准确率有显著影响。通常情况下，学习率太高会导致训练不稳定甚至发散，而学习率过低则会导致收敛速度慢，需要更多的 epoch 才能达到较好的效果。

下面测试了 SGD 优化器学习率为 1.0, 0.1, 0.01, 0.001 情况下的验证集准确率，并绘制了如下曲线：

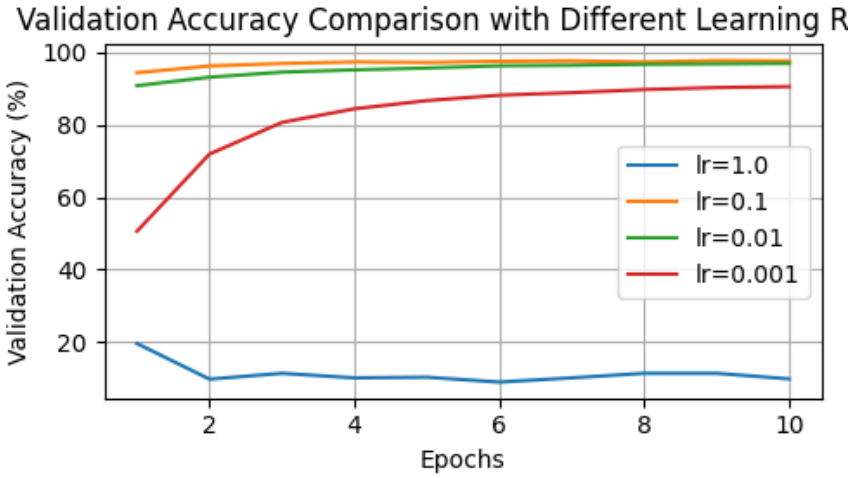


图 3.5: 使用 SGD 优化器不同学习率在验证集上准确率对比图

可见使用 SGD 优化器在学习率为 0.1 时性能更佳，能够达到 97.67% 的准确率。

3.5 更换优化器

更换优化器是提高模型准确率的一种策略。除了传统的 SGD（随机梯度下降）优化器外，还有一些先进的优化算法可以尝试，如 Adam 等。其在调整学习率和动量方面可能更为灵活，有助于更快地收敛到较优解或避免陷入局部最优。

下面测试了 Adam 优化器学习率为 1.0, 0.1, 0.01, 0.001 情况下的验证集准确率，并绘制了如下曲线：

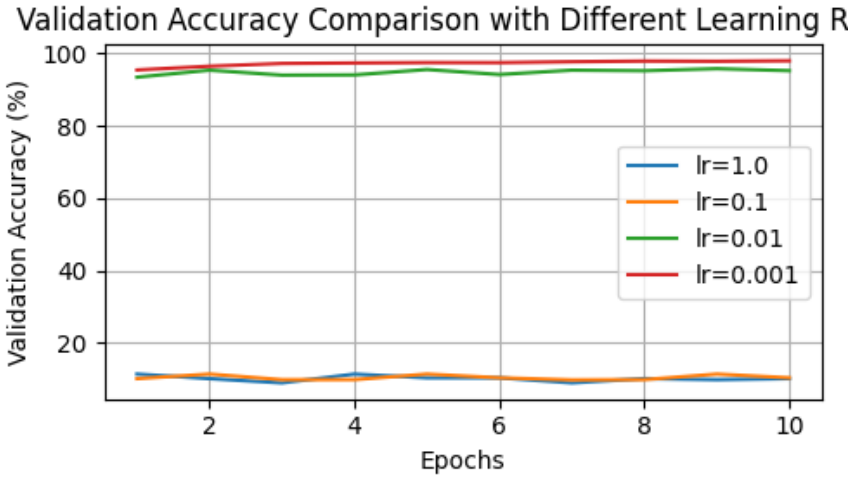


图 3.6: 使用 Adam 优化器不同学习率在验证集上准确率对比图

发现 Adam 优化器在学习率为 0.001 时性能更佳，迭代 10 轮能够达到 97.95% 的准确率。

3.6 最终改进后网络

网络结构如下：

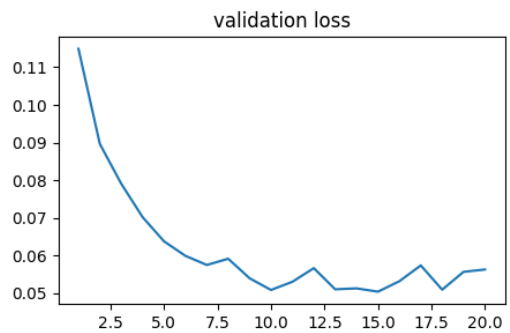
```python
class Net(nn.Module):
def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(28*28, 200)
    self.bn1 = nn.BatchNorm1d(200)
    self.fc1_drop = nn.Dropout(0.2)
    self.fc2 = nn.Linear(200, 150)
    self.bn2 = nn.BatchNorm1d(150)
    self.fc2_drop = nn.Dropout(0.2)
    self.fc3 = nn.Linear(150, 100)
    self.bn3 = nn.BatchNorm1d(100)
    self.fc3_drop = nn.Dropout(0.2)
    self.fc4 = nn.Linear(100, 10)

def forward(self, x):
    x = x.view(-1, 28*28)
    x = F.relu(self.bn1(self.fc1(x)))
    x = self.fc1_drop(x)
    x = F.relu(self.bn2(self.fc2(x)))
    x = self.fc2_drop(x)
    x = F.relu(self.bn3(self.fc3(x)))
    x = self.fc3_drop(x)
    return F.log_softmax(self.fc4(x), dim=1)
```

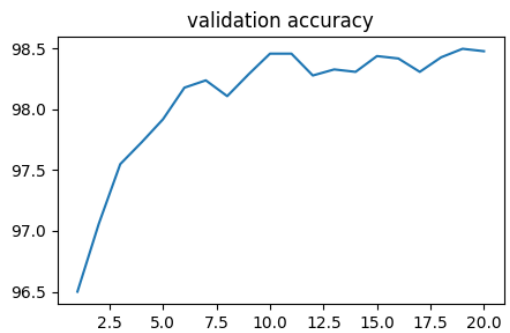
使用的优化器、损失函数如下：

```python
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

模型训练 20 个轮次，最终得到损失与准确率曲线如下：



(a) 改进版本 MLP 的损失曲线



(b) 改进版本 MLP 的准确度曲线

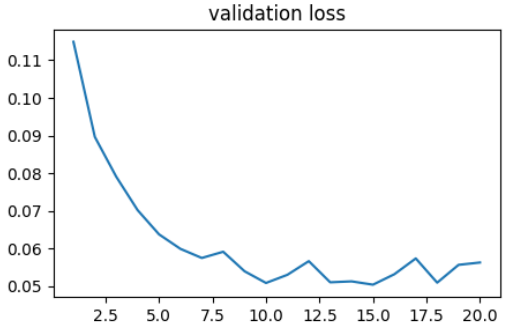
最终的准确率在 98.48% 左右。

4 ResMLP 实现

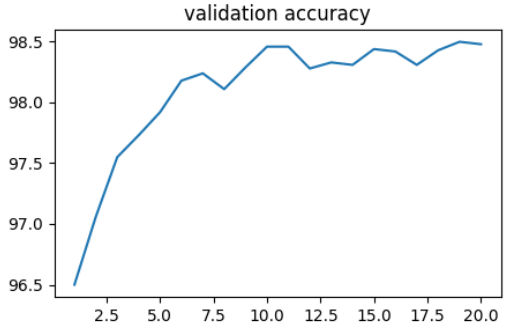
ResMLP 网络结构如下所示：

```
ResMLP(
  (to_patch_embedding): Sequential(
    (0): Conv2d(1, 384, kernel_size=(7, 7), stride=(7, 7))
    (1): Rearrange('b c h w -> b (h w) c')
  )
  (mlp_blocks): ModuleList(
    (0-1): 2 x MLPblock(
      (pre_affine): Aff()
      (token_mix): Sequential(
        (0): Rearrange('b n d -> b d n')
        (1): Linear(in_features=16, out_features=16, bias=True)
        (2): Rearrange('b d n -> b n d')
      )
      (ff): Sequential(
        (0): FeedForward(
          (net): Sequential(
            (0): Linear(in_features=384, out_features=1536, bias=True)
            (1): GELU(approximate='none')
            (2): Dropout(p=0.0, inplace=False)
            (3): Linear(in_features=1536, out_features=384, bias=True)
            (4): Dropout(p=0.0, inplace=False)
          )
        )
      )
      (post_affine): Aff()
    )
  )
  (affine): Aff()
  (mlp_head): Sequential(
    (0): Linear(in_features=384, out_features=10, bias=True)
  )
)
```

训练模型，损失和准确度曲线如下：



(c) ResMLP 的损失曲线



(d) ResMLP 的准确度曲线

在 20 轮迭代后准确率在 98.14% 左右。

5 实验心得

1. **模型深度和宽度的影响：**增加网络的深度和宽度通常可以提高模型的表达能力，但同时也可能带来过拟合的风险。因此，找到一个合适的深度和宽度是关键。
2. **优化器选择的重要性：**不同的优化器对模型性能有显著影响。Adam 等高级优化器通常能够提供更好的性能和更快的收敛速度。
3. **批归一化的好处：**批归一化可以稳定训练过程，允许使用更高的学习率并加快收敛。此外，它还可以作为一种正则化手段，减少模型过拟合的风险。
4. **学习率调整的必要性：**合适的学习率是模型成功训练的关键。通过实验找到最佳的学习率，可以显著提高模型的训练效果和最终性能。

本次实验通过一系列步骤和方法改进了神经网络的性能，展示了不同模型结构、优化器和训练参数对模型性能的影响。通过实验，我能够更好地理解和应用深度学习技术，提升模型在实际任务中的表现。