

人工智能作业1

1852824 吴杨婉婷

- [人工智能作业1](#)
 - [1. 作业需求描述](#)
 - [2. Requirements](#)
 - [3. 模型代码详解](#)
 - [4.实验结果](#)
 - [4.1 mnist数据集](#)
 - [4.1.1 One_Layer_Net](#)
 - [4.1.2 Sigmoid_Net1](#)
 - [4.1.3 ReLU_Net1](#)
 - [4.1.4 Batch_Net1](#)
 - [4.1.5 Drop_Net1](#)
 - [4.1.6 Three_Layer_Net](#)
 - [4.1.7 Sigmoid_Net2](#)
 - [4.1.8 ReLU_Net2](#)
 - [4.1.9 Batch_Net2](#)
 - [4.1.10 Drop_Net2](#)
 - [4.1.11 正则化](#)
 - [4.1.12 LeNet](#)
 - [4.2 mnistFashion数据集](#)
 - [4.1.1 One_Layer_Net](#)
 - [4.1.2 Sigmoid_Net1](#)
 - [4.1.3 ReLU_Net1](#)
 - [4.1.4 Batch_Net1](#)
 - [4.1.5 Drop_Net1](#)
 - [4.1.6 Three_Layer_Net](#)
 - [4.1.7 Sigmoid_Net2](#)
 - [4.1.8 ReLU_Net2](#)
 - [4.1.9 Batch_Net2](#)
 - [4.1.10 Drop_Net2](#)
 - [4.1.11 正则化](#)
 - [4.1.12 LeNet](#)
 - [5.结果分析与心得体会](#)
 - [5.1 Sigmoid函数](#)
 - [5.2 ReLU激活函数](#)
 - [5.3 批标准化](#)
 - [5.4 dropout](#)
 - [5.5 L1](#)
 - [5.5 总结](#)
 - [6.主体代码解释](#)
 - [7.作者](#)

1. 作业需求描述

- 1. 使用简单全连接网络作为分类器，完成MNIST分类任务
- 2. 使用深度全连接网络（2个隐藏层以上）作为分类器，完成MNIST分类任务
- 3. 使用深度全连接网络（2个隐藏层以上）作为分类器，添加dropout，正则化等技巧，完成MNIST分类任务
- 4. 使用LetNet5，完成MNIST分类任务
- 5. 重复任务3、4，完成MNIST-fashion分类任务

2. Requirements

- **Development Environment:**

Win 10

- **Development Software:**

PyCharm 2020.3.5.PC-191.6605.12

- **Development Language:**

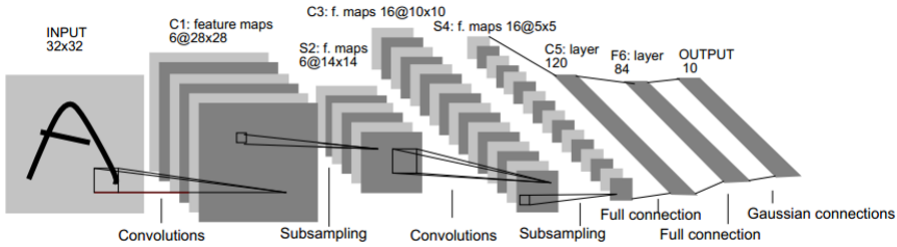
Python

- **Mainly Reference Count:**

- 1. torchvision
- 2. matplotlib
- 3. os
- 4. torch
- 5. numpy

3. 模型代码详解

number	描述	model
1	两个全连接隐藏层，分别是200和100	One_Layer_Net
2	在One_Layer_Net加入sigmoid激活函数	Sigmoid_Net1
3	在One_Layer_Net加入ReLU激活函数	ReLU_Net1
4	在ReLU_Net1加入批标准化	Batch_Net1
5	在ReLU_Net1加入dropout	Drop_Net1
6	三个全连接隐藏层，分别是200，100，150	Three_Layer_Net
7	在Three_Layer_Net加入sigmoid激活函数	Sigmoid_Net2
8	在Three_Layer_Net加入ReLU激活函数	ReLU_Net2
9	在ReLU_Net2加入批标准化	Batch_Net2
10	在ReLU_Net2加入dropout	Drop_Net2

number	描述	model
11	<p>首先是用6个5 * 5的卷积核进行卷积，步长为1，padding为2，再使用Relu激活函数，然后池化，再用16个5*5的卷积核卷积，然后激活和池化，最后三次全连接（详细介绍请看下方示意图）</p> 	LeNet
12	添加L1正则化	L1
13	添加L2正则化	L2

```

class One_Layer_Net(nn.Module):
    def __init__(self):
        super(One_Layer_Net, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),)
        self.fc2 = nn.Sequential(
            nn.Linear(200, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class Sigmoid_Net1(nn.Module):
    def __init__(self):
        super(Sigmoid_Net1, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),
                                nn.Sigmoid())
        self.fc2 = nn.Sequential(
            nn.Linear(200, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class ReLU_Net1(nn.Module):
    def __init__(self):
        super(ReLU_Net1, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),
                                   nn.ReLU())
        self.fc2 = nn.Sequential(
            nn.Linear(200, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class Batch_Net1(nn.Module):
    def __init__(self):
        super(Batch_Net1, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200), nn.BatchNorm1d(200),
                                   nn.ReLU())
        self.fc2 = nn.Sequential(
            nn.Linear(200, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class Drop_Net1(nn.Module):
    def __init__(self):
        super(Batch_Net1, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200), torch.nn.Dropout(0.3),
                                   nn.ReLU())
        self.fc2 = nn.Sequential(
            nn.Linear(200, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class Three_Layer_Net(nn.Module):
    def __init__(self):
        super(Three_Layer_Net, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),nn.Linear(200,
100),nn.Linear(100, 150))
        self.fc2 = nn.Sequential(
            nn.Linear(150, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

```

```

class Sigmoid_Net2(nn.Module):
    def __init__(self):
        super(Sigmoid_Net2, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),
                                nn.Sigmoid())
        self.fc2 = nn.Sequential(nn.Linear(200,150),
                                nn.Sigmoid(),
                                nn.Linear(150, 100),
                                nn.Sigmoid()
                                )
        self.fc3 = nn.Sequential(
            nn.Linear(100, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x

```

```

class ReLU_Net2(nn.Module):
    def __init__(self):
        super(ReLU_Net2, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200),
                                nn.ReLU())
        self.fc2 = nn.Sequential(nn.Linear(200,150),
                                nn.ReLU(),
                                nn.Linear(150, 100),
                                nn.ReLU()
                                )

```

```

        self.fc3 = nn.Sequential(
            nn.Linear(100, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x

```

```

class Batch_Net2(nn.Module):
    def __init__(self):
        super(Batch_Net2, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200), nn.BatchNorm1d(200),
                                   nn.ReLU())
        self.fc2 = nn.Sequential(nn.Linear(200,150), nn.BatchNorm1d(150),
                                   nn.ReLU(),
                                   nn.Linear(150, 100), nn.BatchNorm1d(100),
                                   nn.ReLU()
                                   )
        self.fc3 = nn.Sequential(
            nn.Linear(100, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x

```

```

class Drop_Net2(nn.Module):
    def __init__(self):
        super(Drop_Net2, self).__init__()

        self.fc1 = nn.Sequential(nn.Linear(28*28, 200), torch.nn.Dropout(0.3),
                                   nn.ReLU())
        self.fc2 = nn.Sequential(nn.Linear(200,150), torch.nn.Dropout(0.3),
                                   nn.ReLU(),
                                   nn.Linear(150, 100), torch.nn.Dropout(0.3),
                                   nn.ReLU()
                                   )
        self.fc3 = nn.Sequential(
            nn.Linear(100, 10))

    def forward(self, x):
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)

```

```
x = self.fc3(x)
return x
```

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(nn.Conv2d(1, 6, 5, 1, 2), nn.ReLU(),
                                    nn.MaxPool2d(2, 2))

        self.conv2 = nn.Sequential(nn.Conv2d(6, 16, 5), nn.ReLU(),
                                    nn.MaxPool2d(2, 2))

        self.fc1 = nn.Sequential(nn.Linear(16 * 5 * 5, 120),
                                   nn.BatchNorm1d(120), nn.ReLU())

        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.BatchNorm1d(84),
            nn.ReLU(),
            nn.Linear(84, 10))
        # 最后的结果一定要变为 10, 因为数字的选项是 0 ~ 9

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

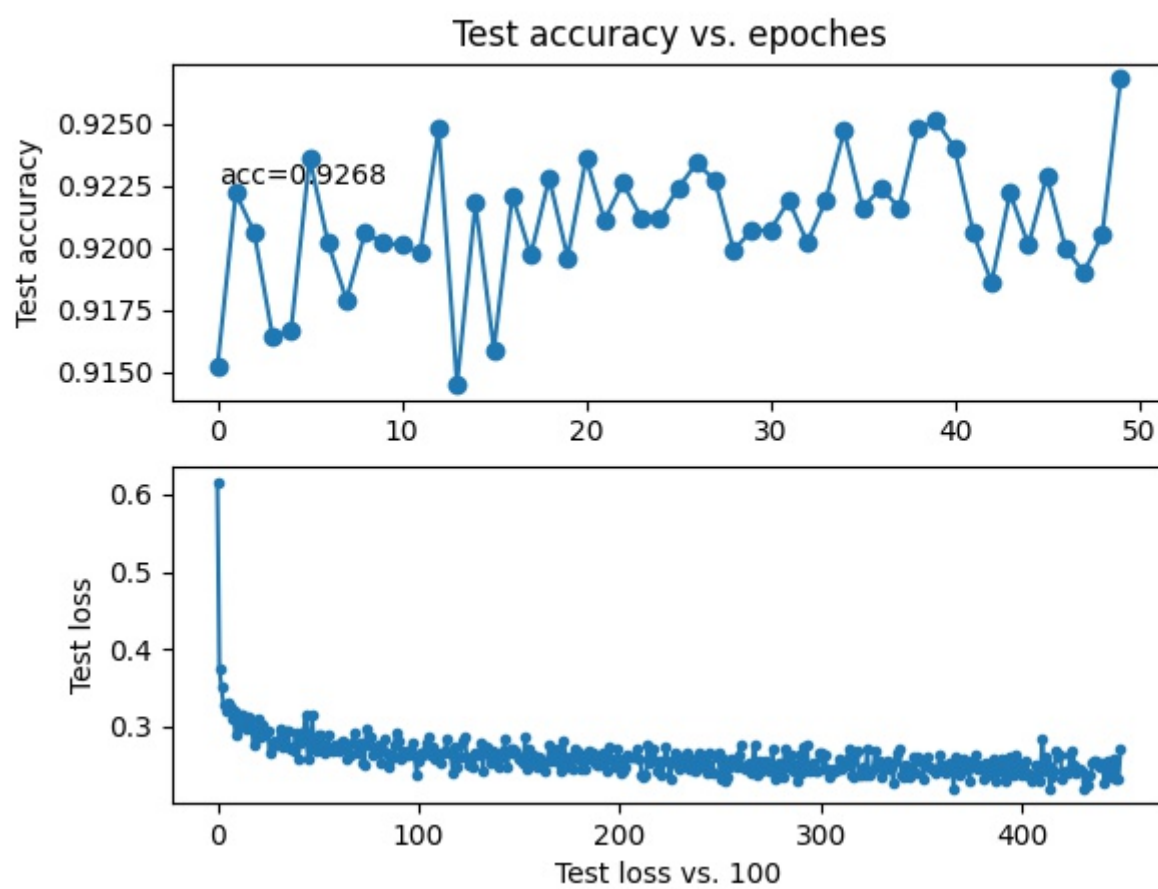
4.实验结果

4.1 mnist数据集

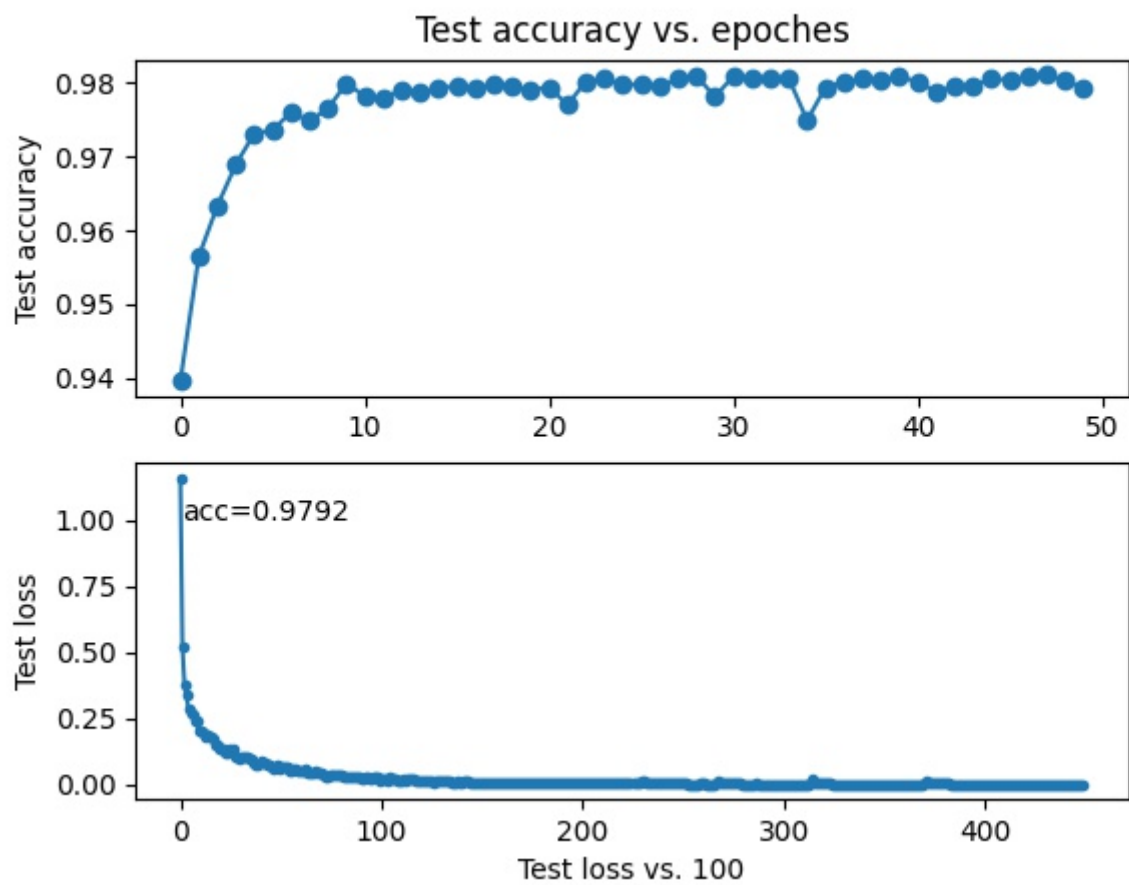
number	model	acc
1	One_Layer_Net	0.9268
2	Sigmoid_Net1	0.9792
3	ReLU_Net1	0.9821
4	Batch_Net1	0.9808
5	Drop_Net1	0.9816
6	Three_Layer_Net	0.9215
7	Sigmoid_Net2	0.9764
8	ReLU_Net2	0.9824

number	model	acc
9	Batch_Net2	0.9828
10	Drop_Net2	0.9831
11	L1 正则化	0.9474
12	LeNet	0.9918

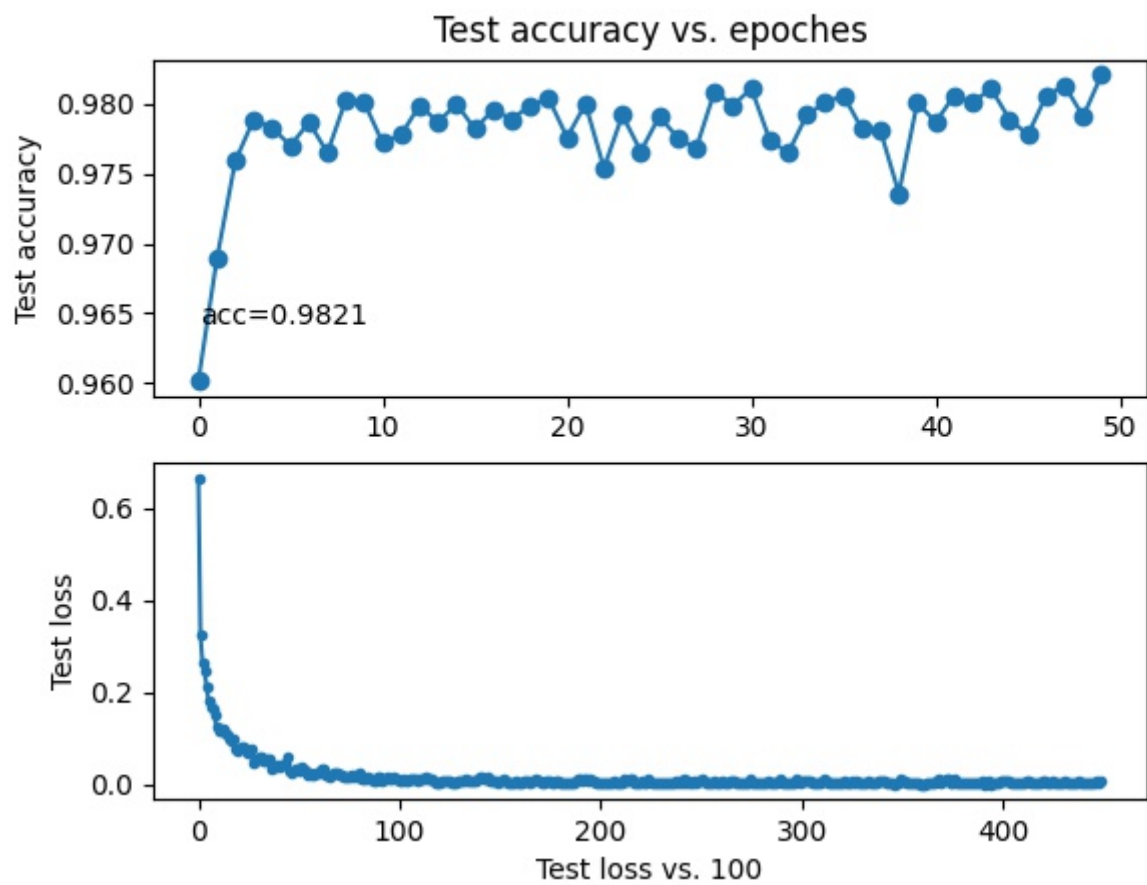
4.1.1 One_Layer_Net



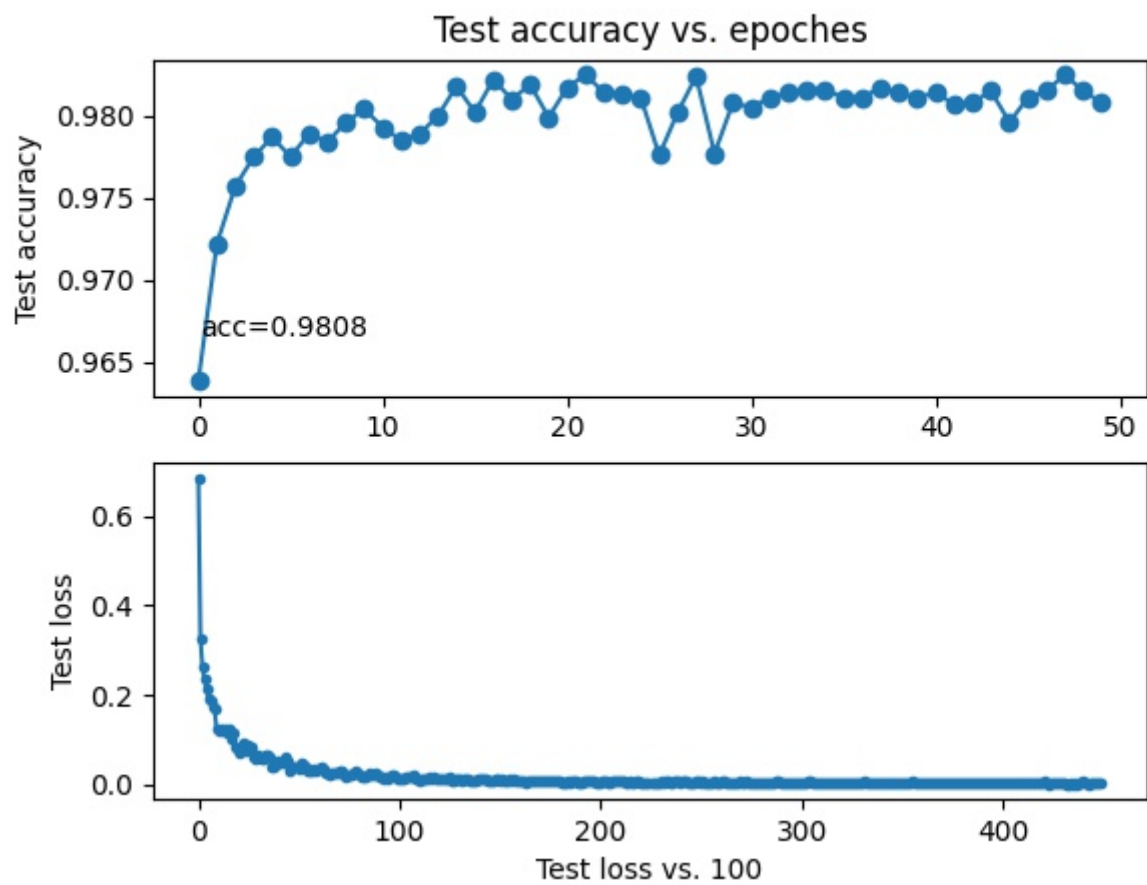
4.1.2 Sigmoid_Net1



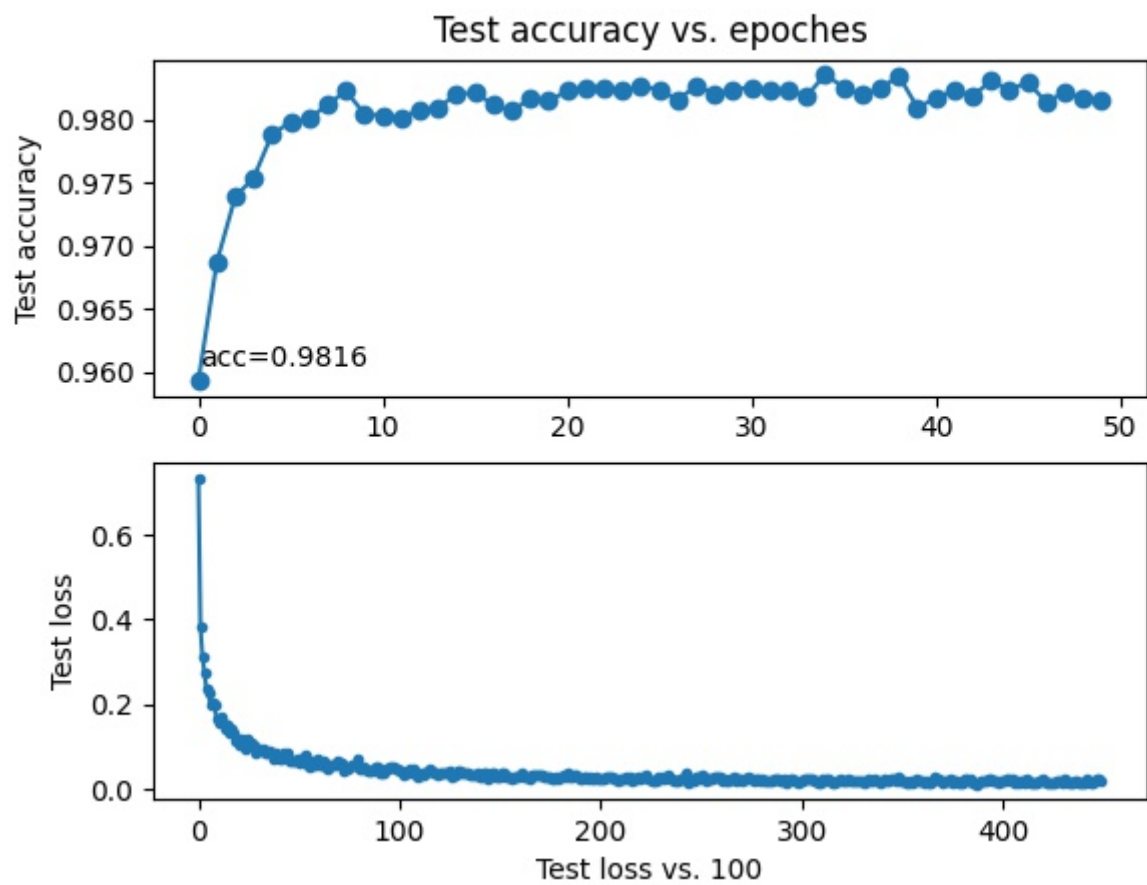
4.1.3 ReLU_Net1



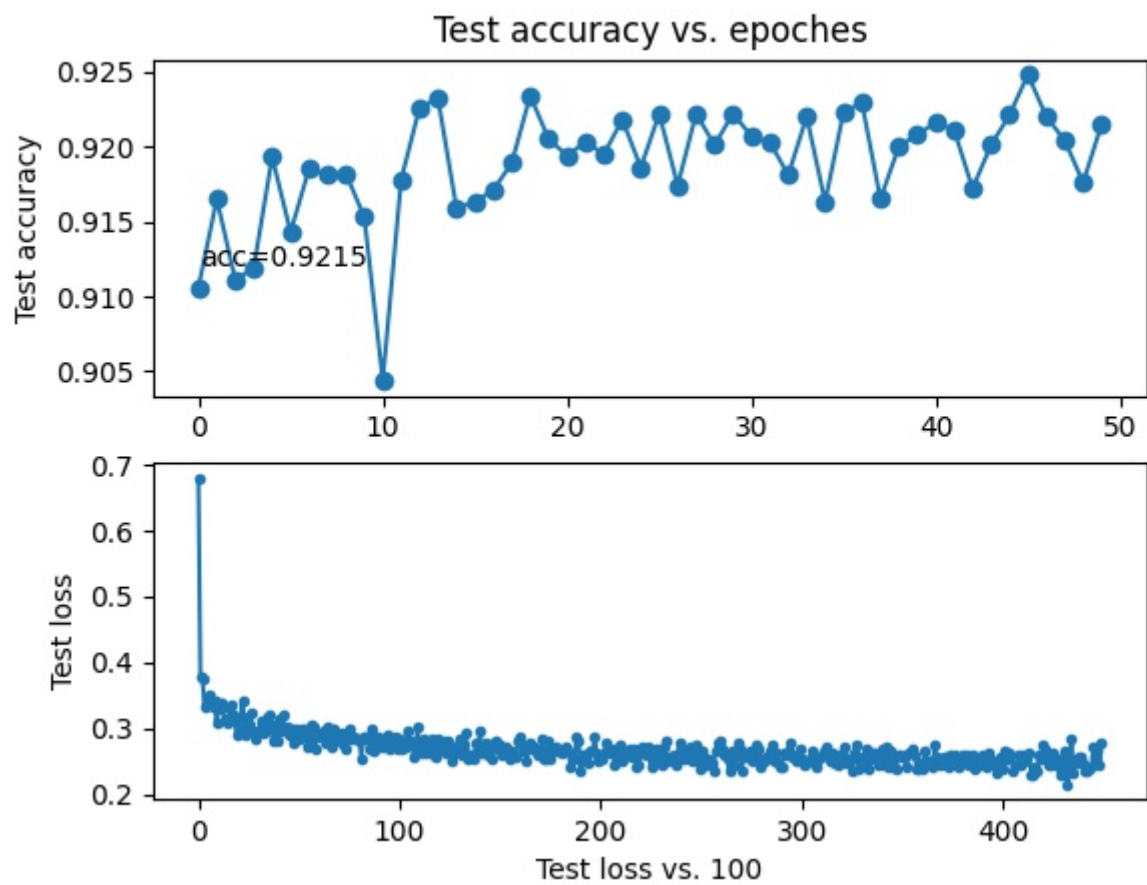
4.1.4 Batch_Net1



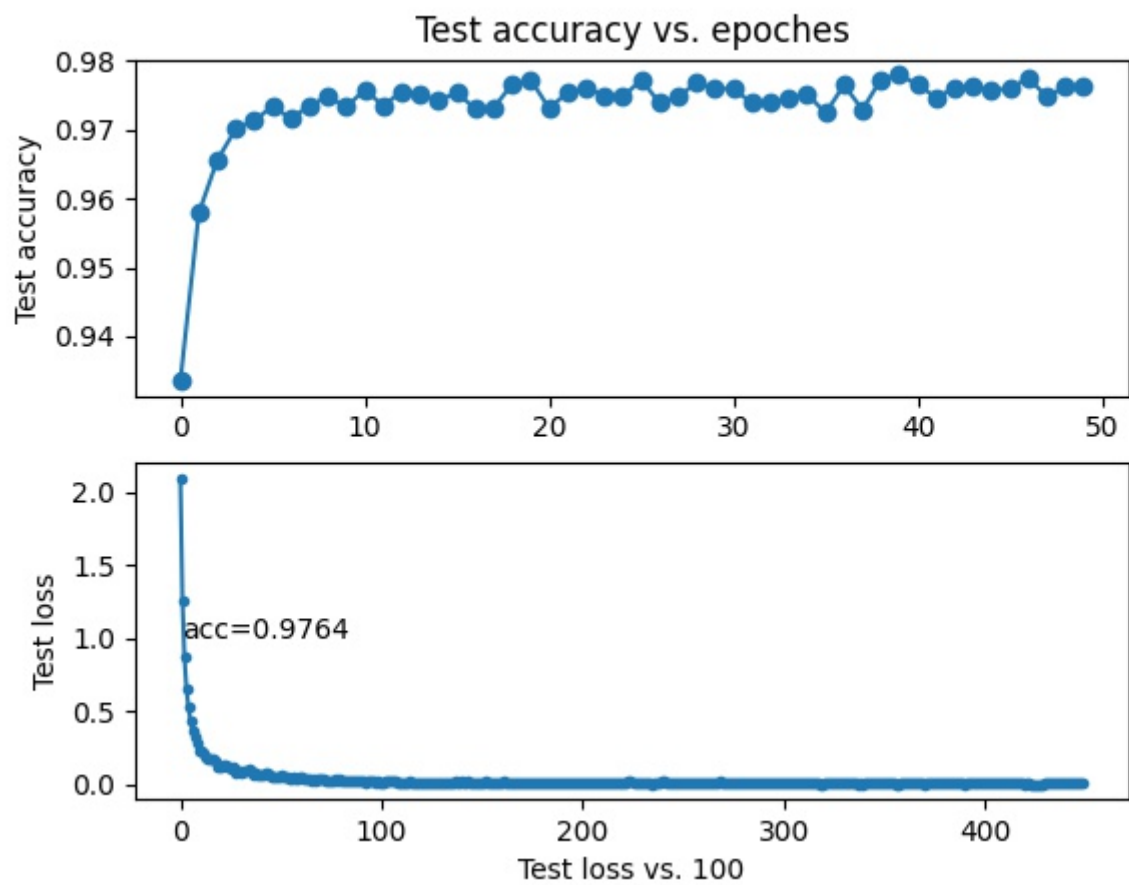
4.1.5 Drop_Net1



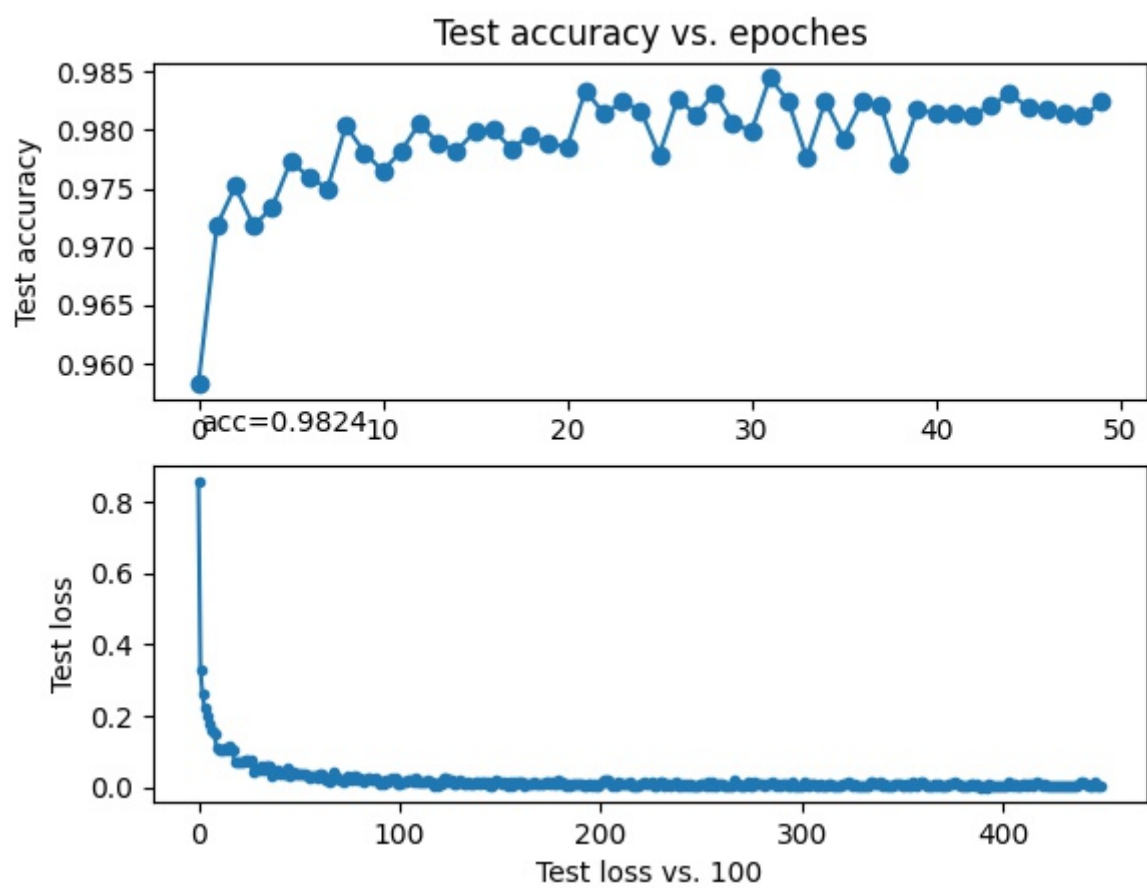
4.1.6 Three_Layer_Net



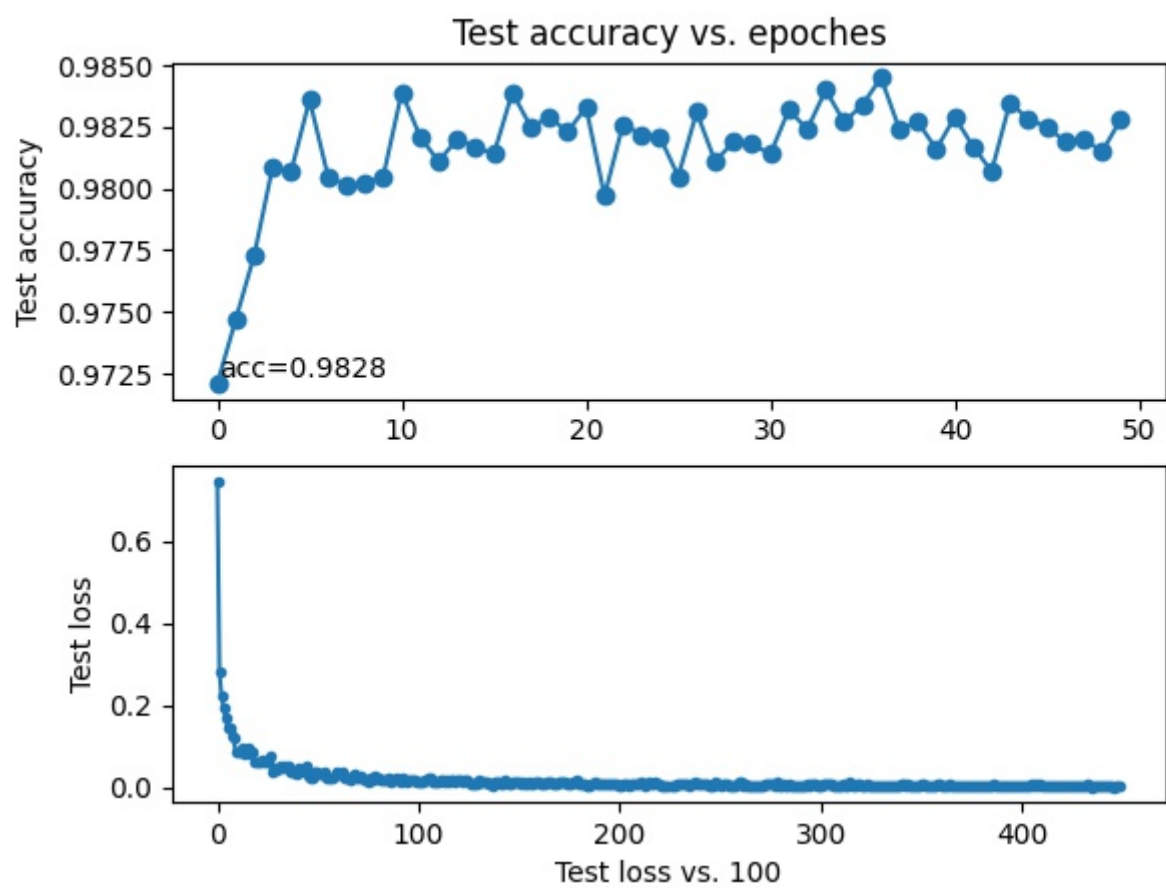
4.1.7 Sigmoid_Net2



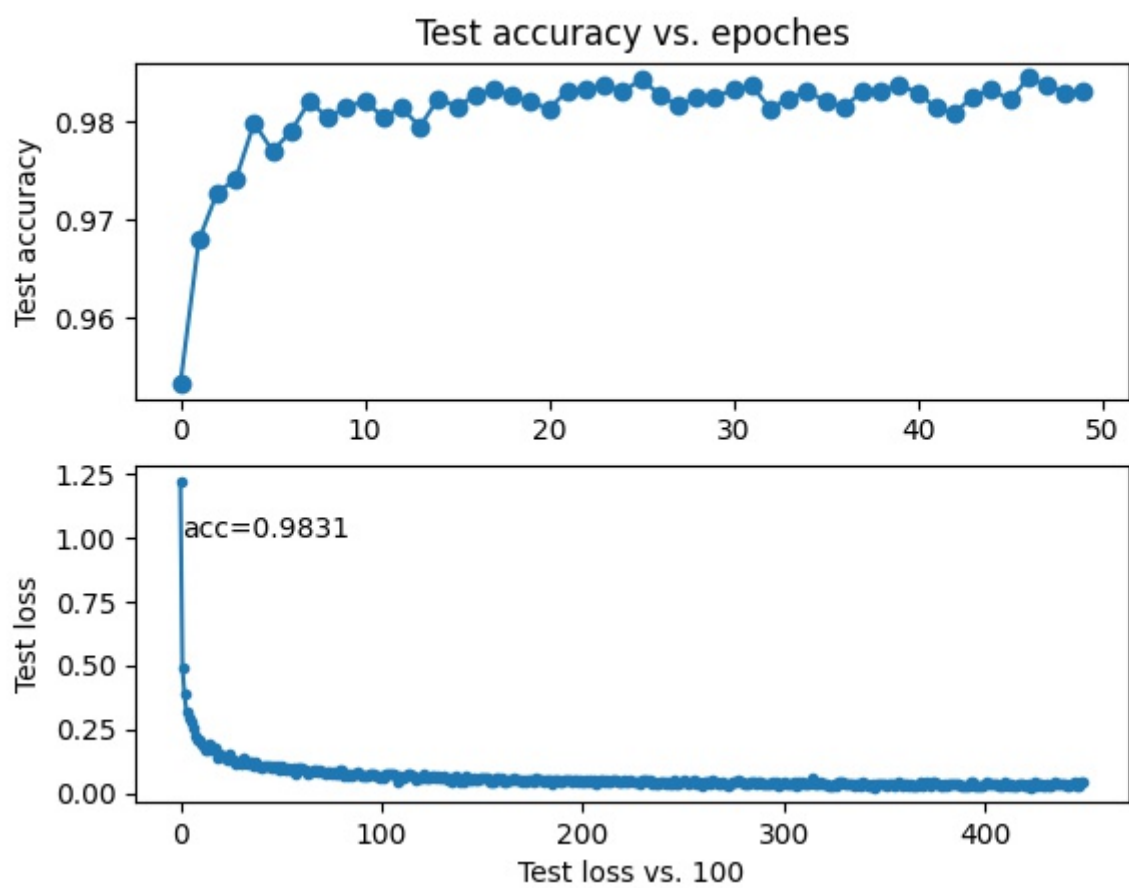
4.1.8 ReLU_Net2



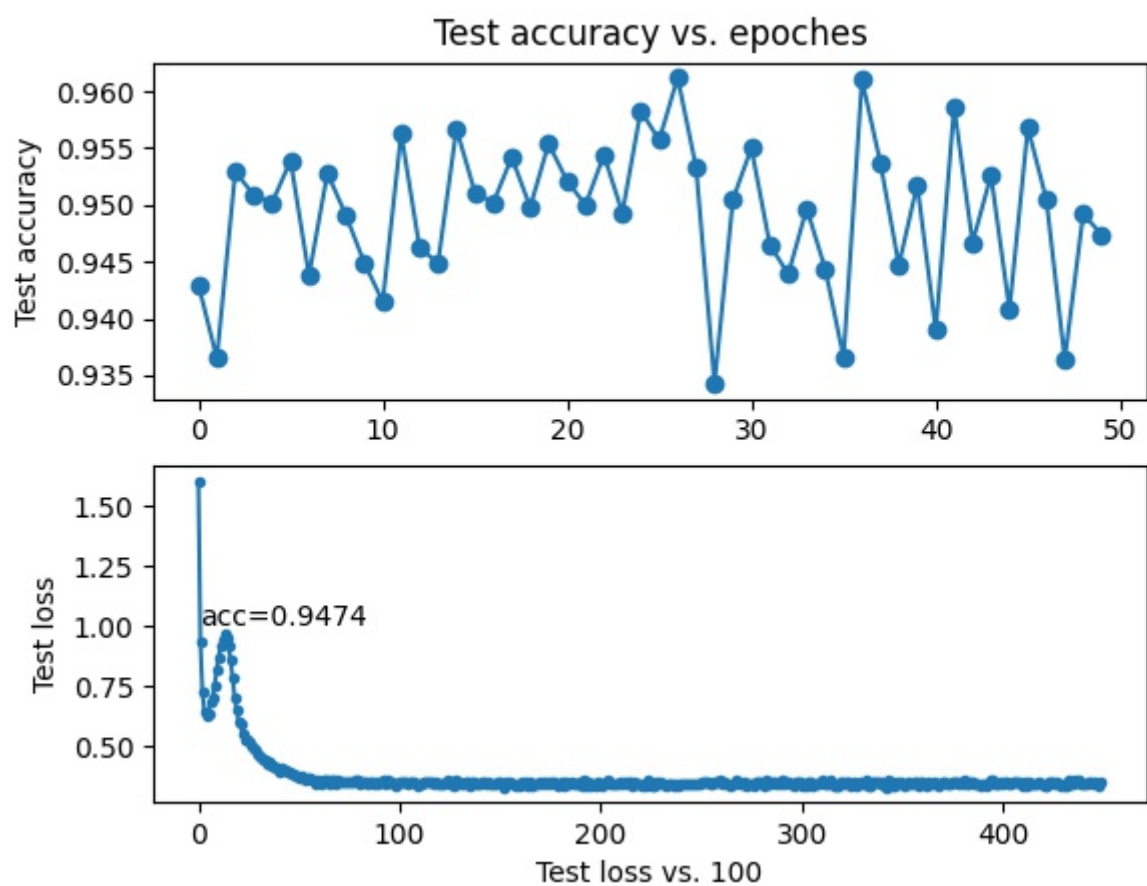
4.1.9 Batch_Net2



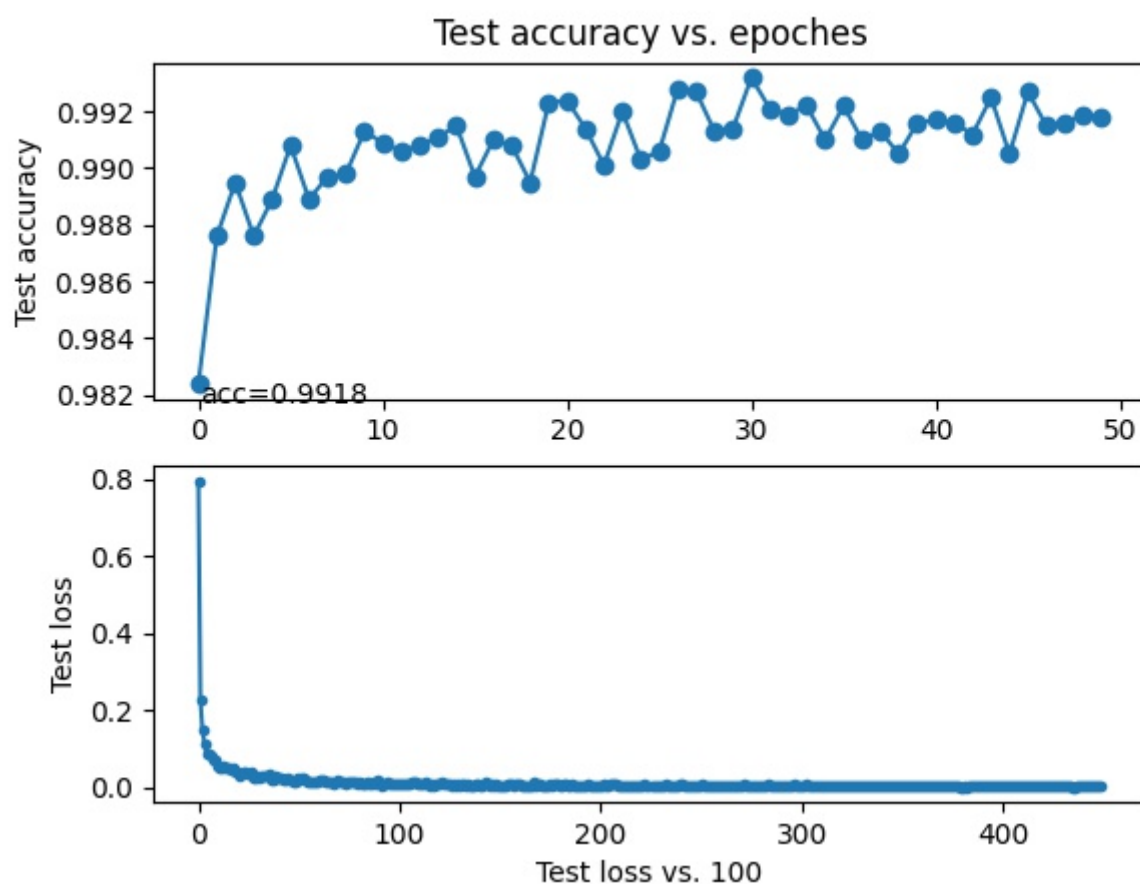
4.1.10 Drop_Net2



4.1.11 正则化



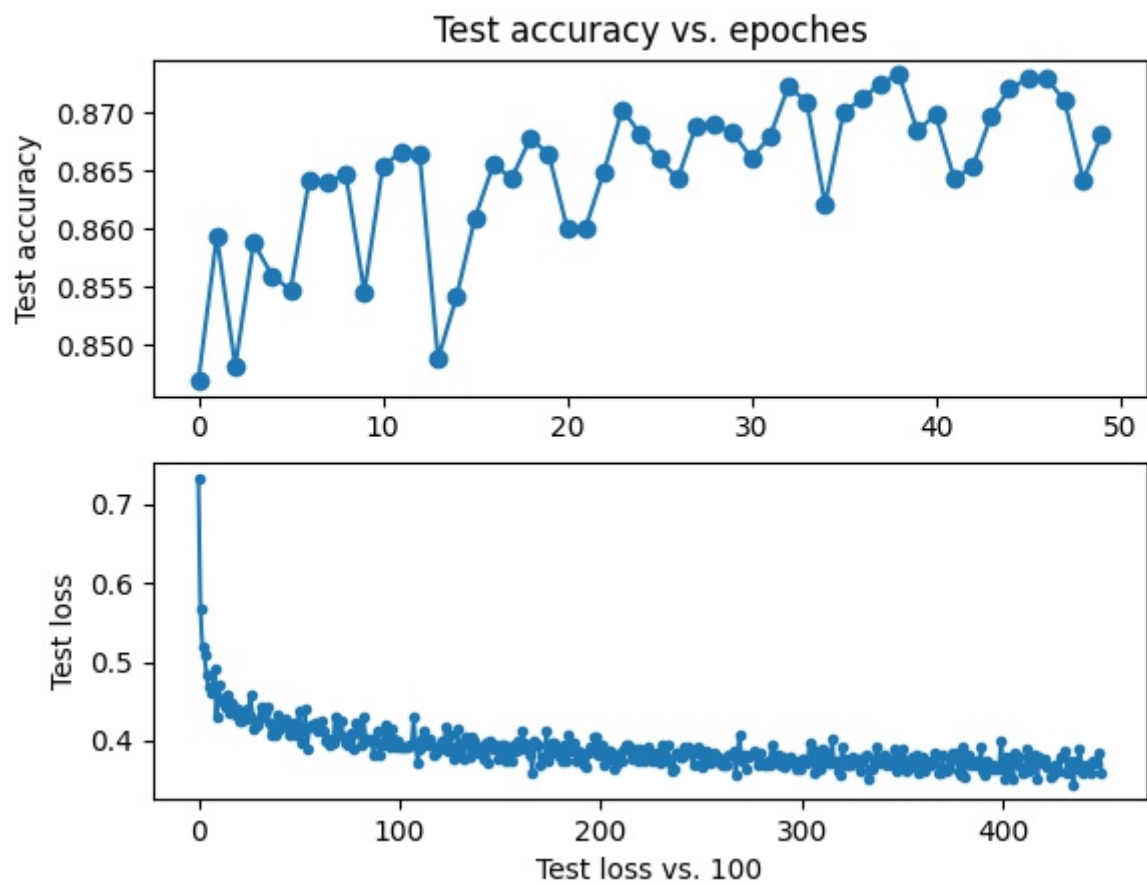
4.1.12 LeNet



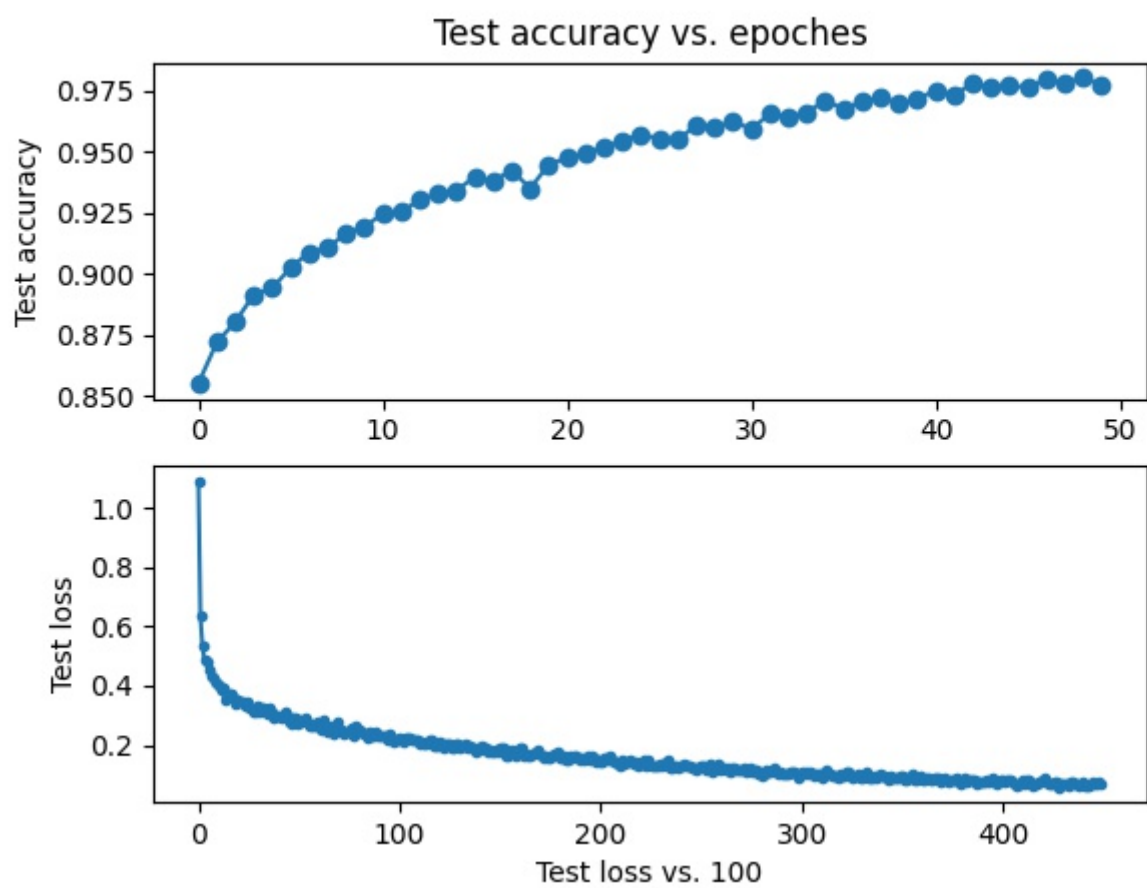
4.2 mnistFashion数据集

number	model	acc
1	One_Layer_Net	0.86805
2	Sigmoid_Net1	0.9774166666666667
3	ReLU_Net1	0.9769833333333333
4	Batch_Net1	0.9859166666666667
5	Drop_Net1	0.9546833333333333
6	Three_Layer_Net	0.86985
7	Sigmoid_Net2	0.9671
8	ReLU_Net2	0.9717333333333333
9	Batch_Net2	0.9914666666666667
10	Drop_Net2	0.9415666666666667
11	L1 正则化	0.7968
12	LeNet	0.99465

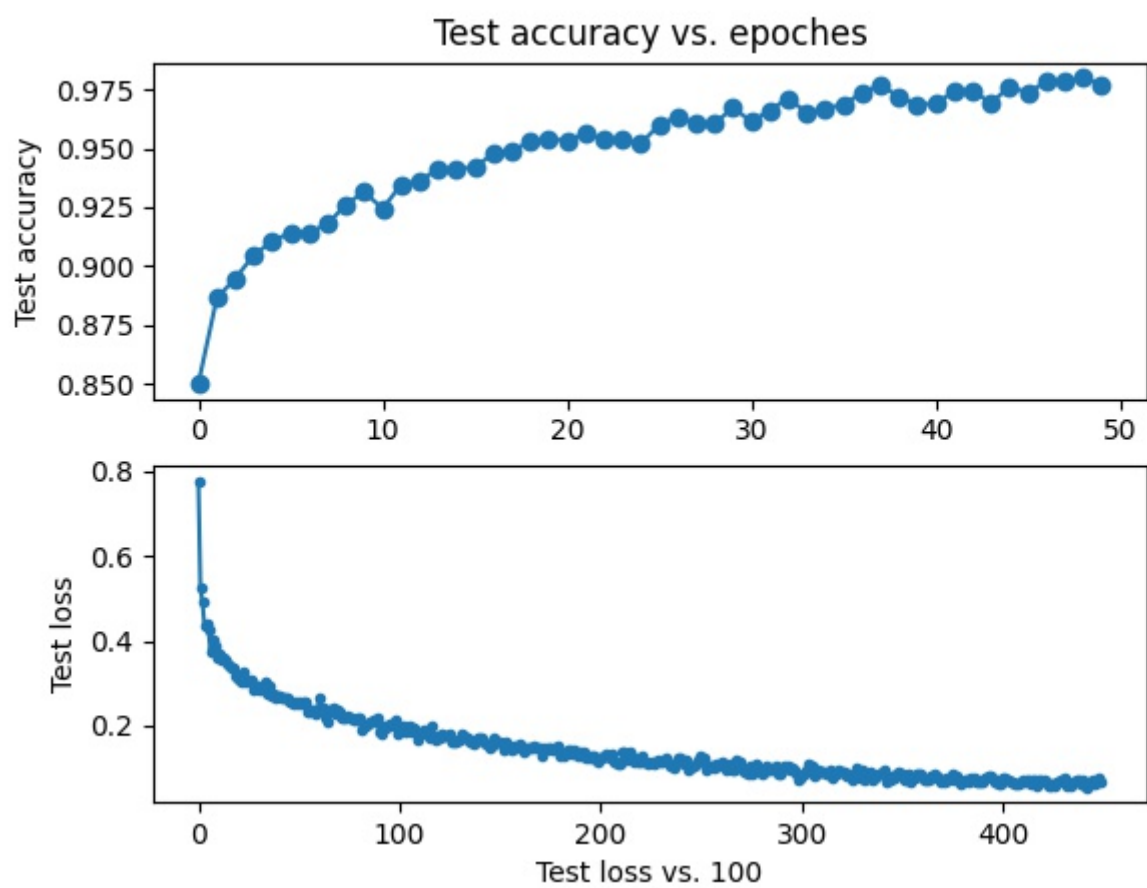
4.1.1 One_Layer_Net



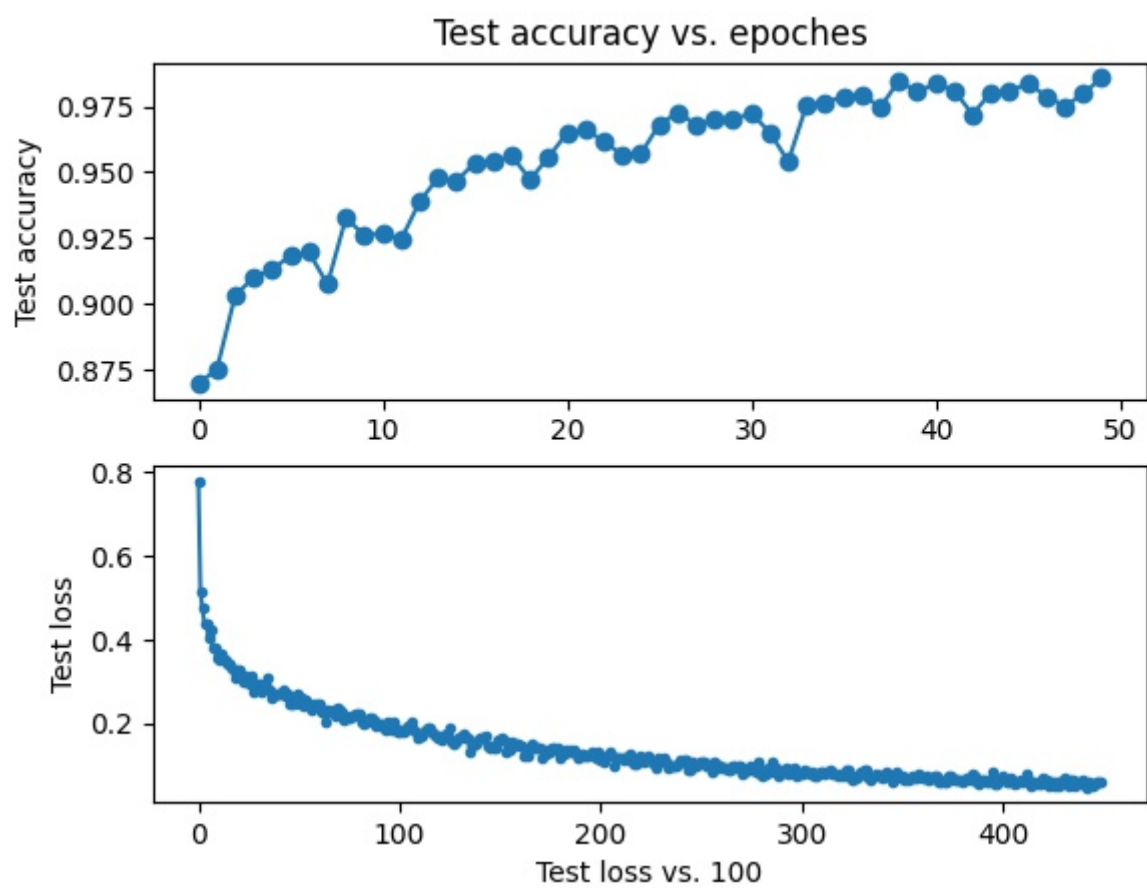
4.1.2 Sigmoid_Net1



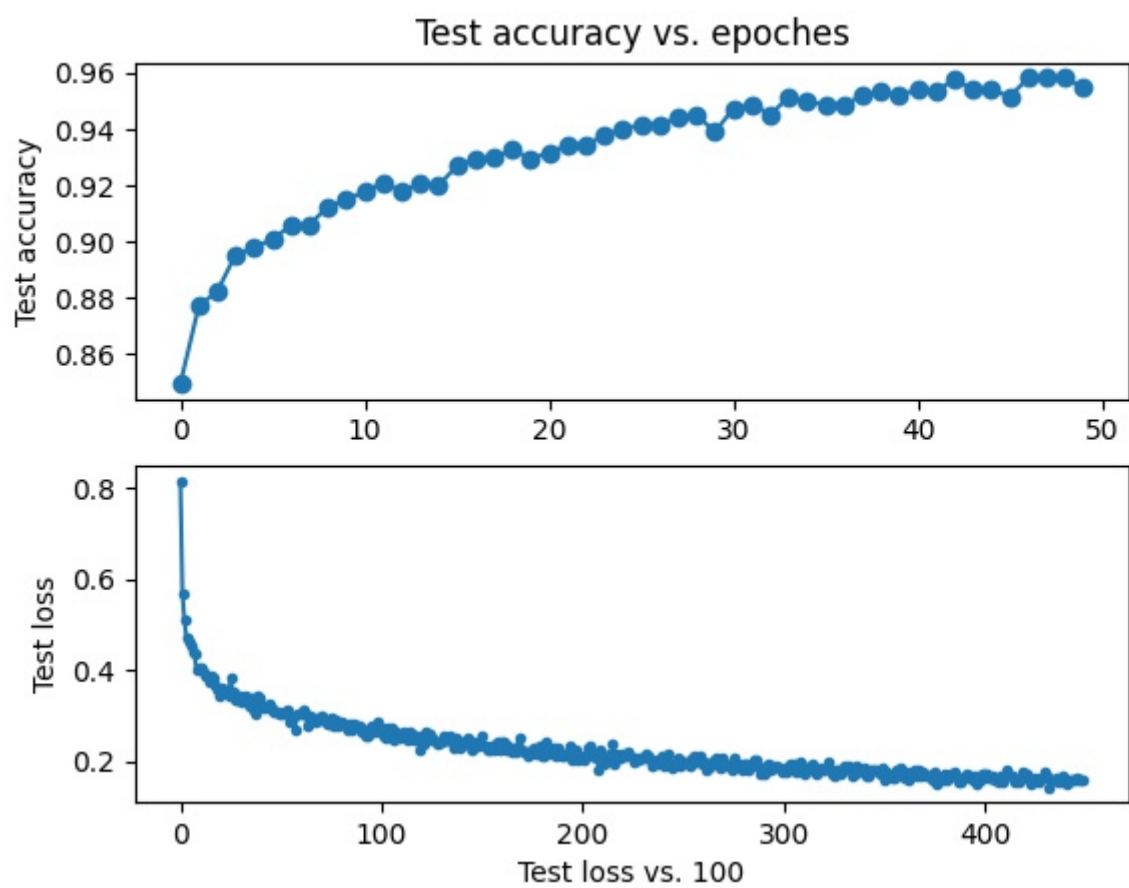
4.1.3 ReLU_Net1



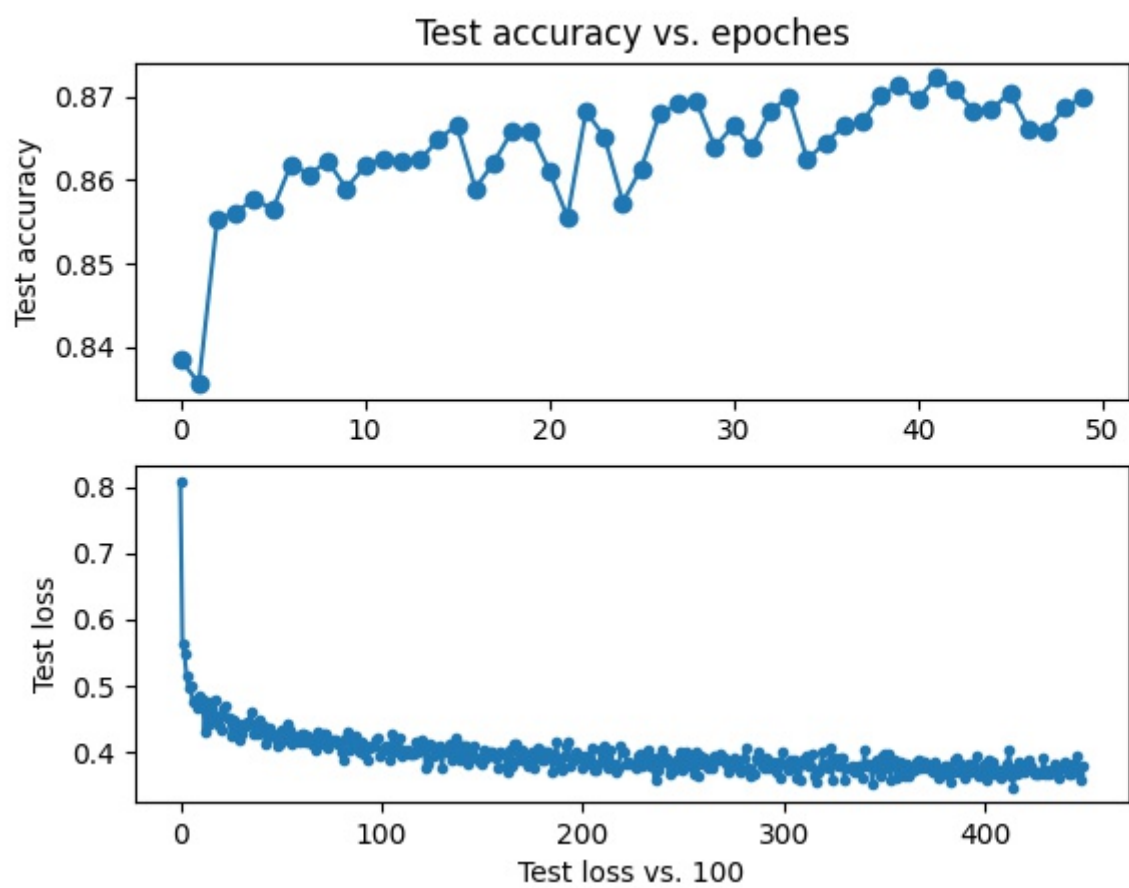
4.1.4 Batch_Net1



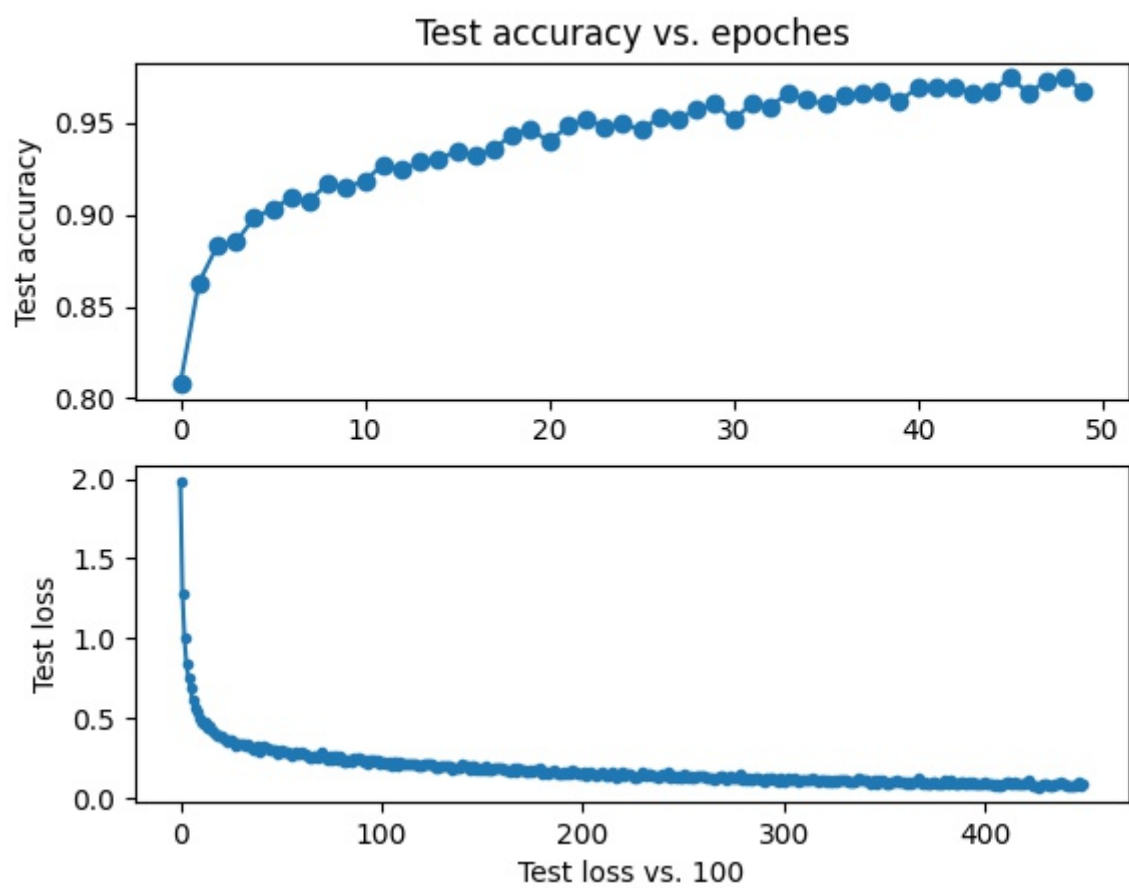
4.1.5 Drop_Net1



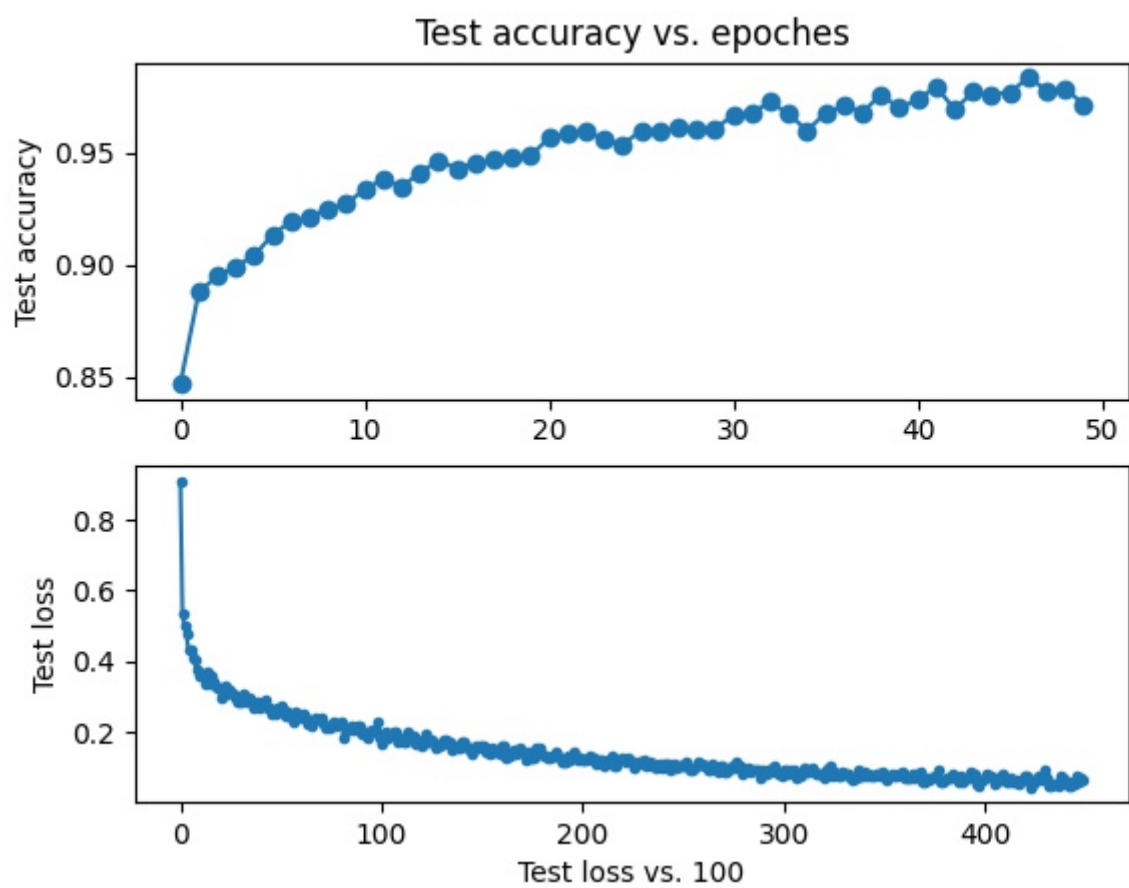
4.1.6 Three_Layer_Net



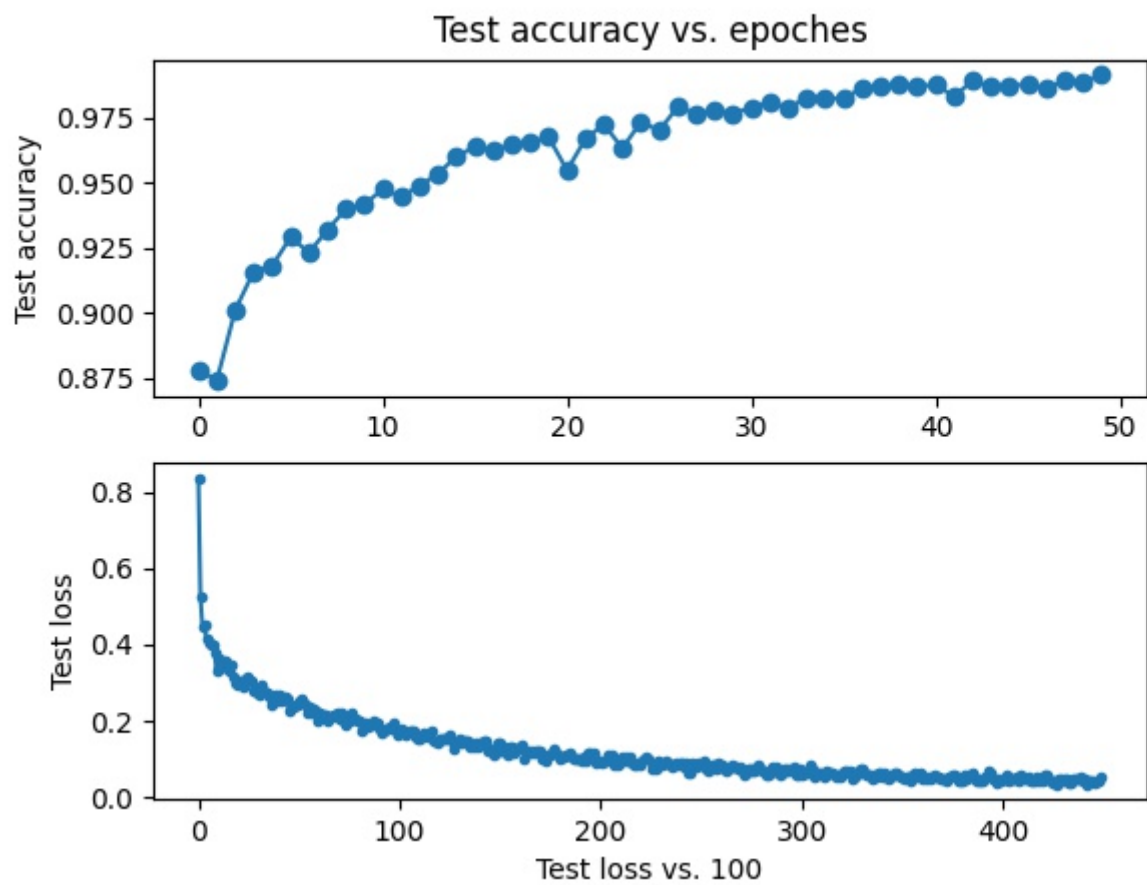
4.1.7 Sigmoid_Net2



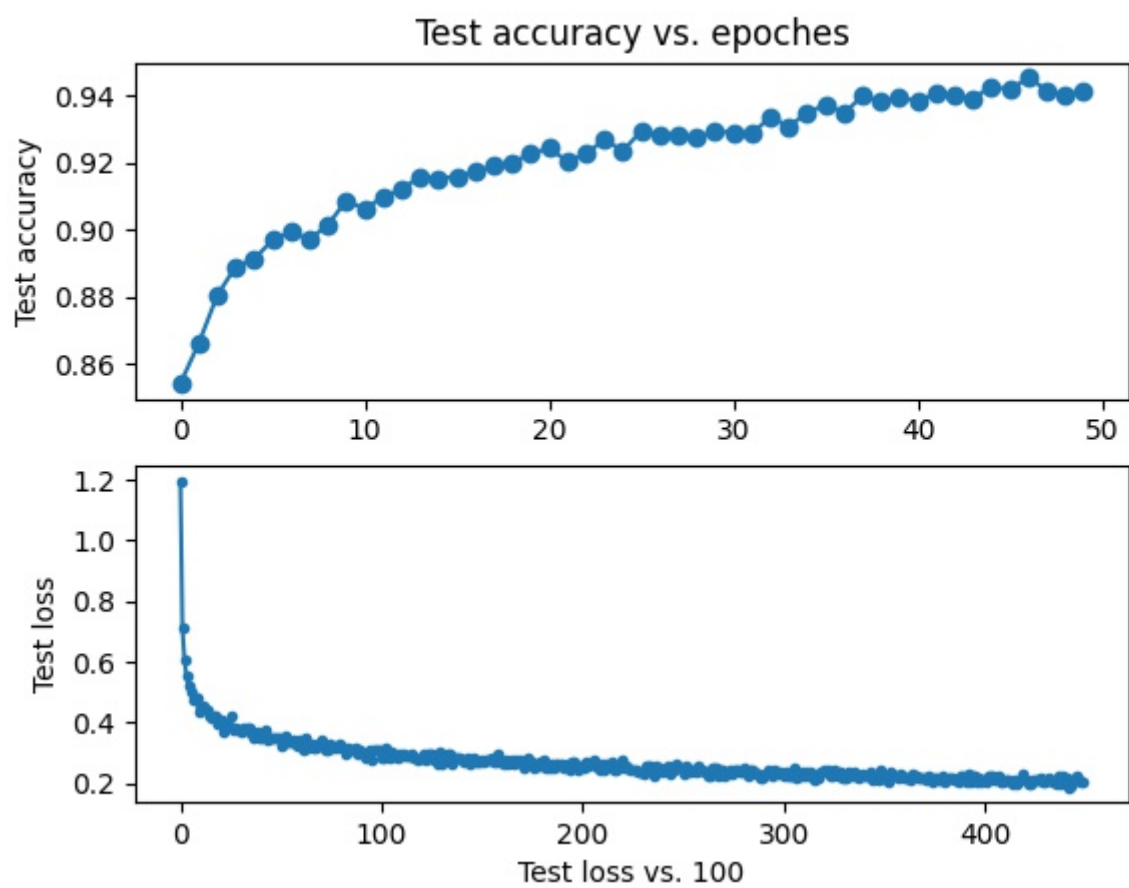
4.1.8 ReLU_Net2



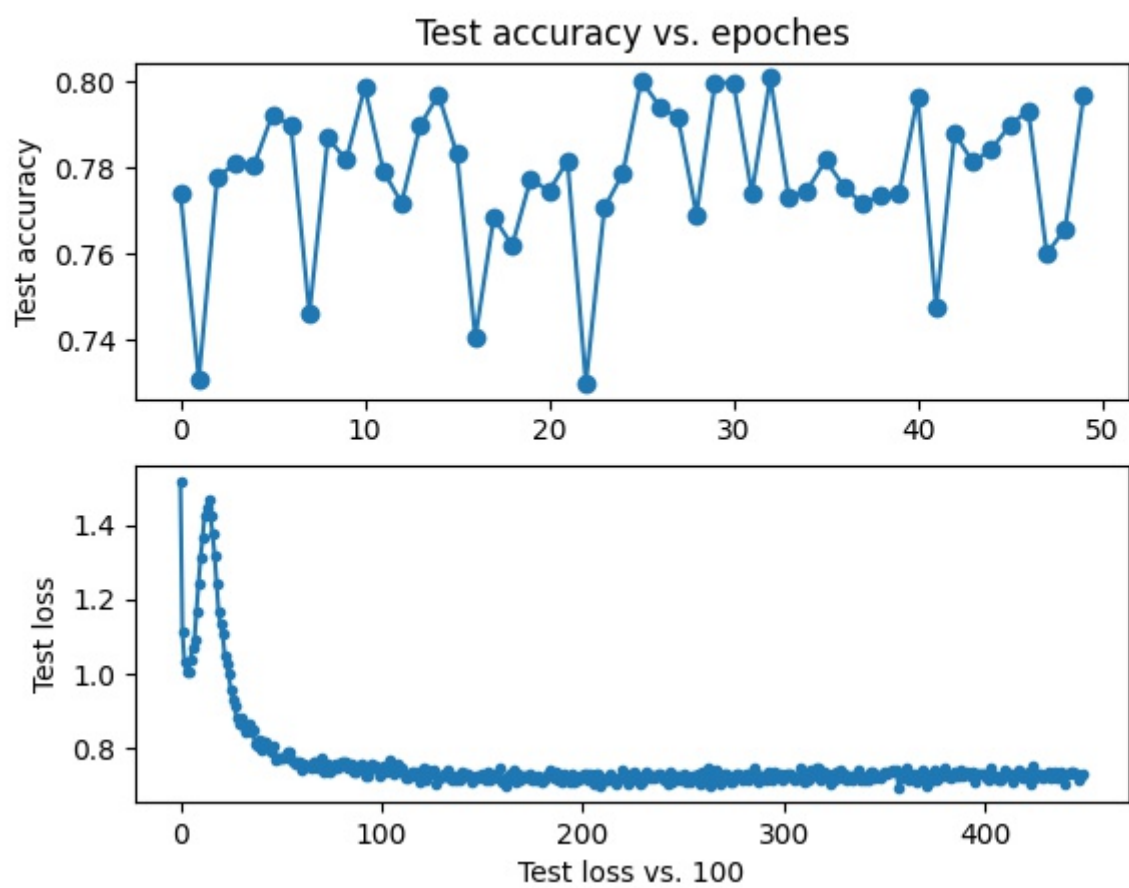
4.1.9 Batch_Net2



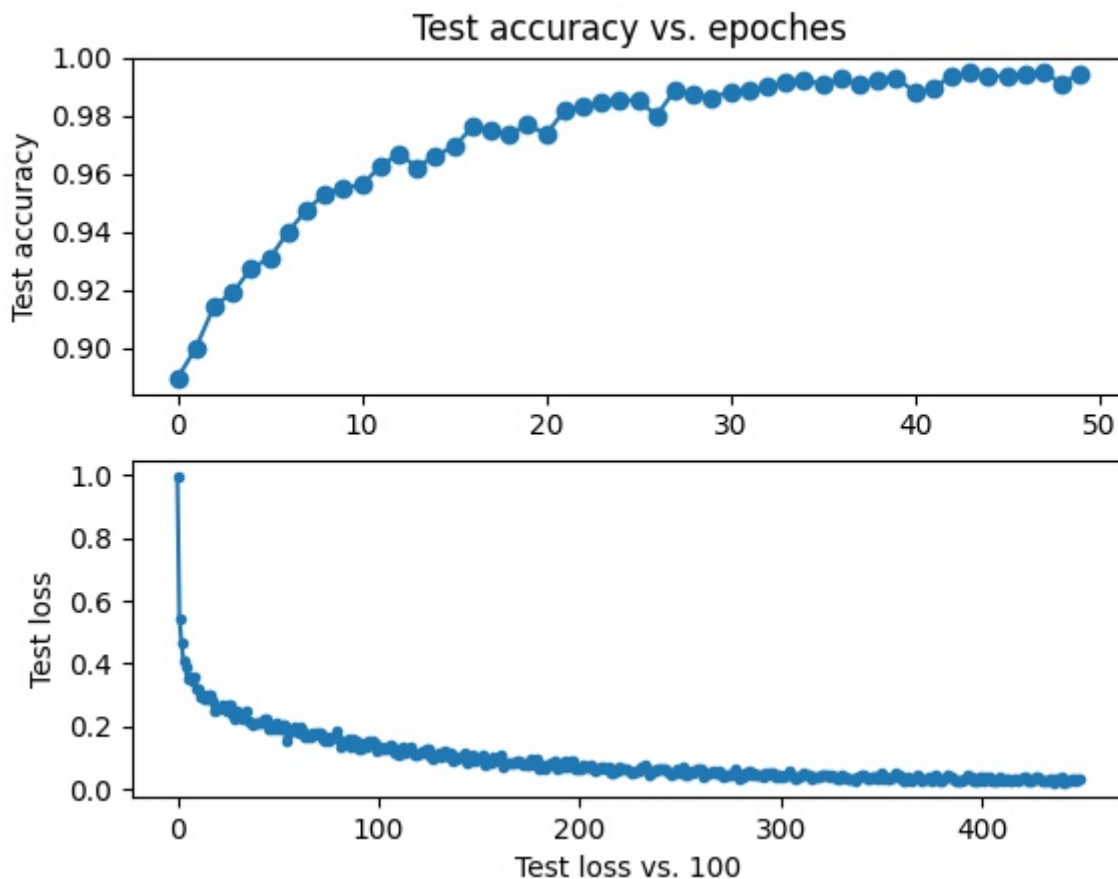
4.1.10 Drop_Net2



4.1.11 正则化



4.1.12 LeNet



5.结果分析与心得体会

5.1 Sigmoid函数

- Sigmoid函数是传统神经网络中最常用的激活函数，虽然现在已经不常用，但当年还是十分受欢迎的。值域在0到1之间。可以看出当x的值趋近负无穷的时候，y趋近于0；x趋近于正无穷的时候，y趋近于1；在 $[-2,2]$ 区间内，梯度变化比较明显，即x发生很小的变化，y变化的也比较明显。
- 优点：
 1. sigmoid函数的输出映射在 $(0,1)$ 之间，单调连续，输出范围有限，优化稳定。
 2. 求导容易。 $f(x) = f(x) * [1 - f(x)]$ 。
- 缺点：
 1. 幂运算，计算成本高。 2.导数值小于1，容易出现梯度消失。当x很小或很大时，存在导数很小的情况。另外，神经网络主要的训练方法是BP算法，BP算法的基础是导数的链式法则，也就是多个导数的乘积。而sigmoid的导数最大为0.25，多个小于等于0.25的数值相乘，其运算结果很小。随着神经网络层数的加深，梯度后向传播到浅层网络时，基本无法引起参数的扰动，也就是没有将loss的信息传递到浅层网络，这样网络就无法训练学习了。
 2. Sigmoid 函数的输出不是以零为中心的，这会导致神经网络收敛较慢。

5.2 ReLU激活函数

- 针对sigmoid的缺点，提出了ReLU函数。通常指代以斜坡函数及其变种为代表的非线性函数。
- 优点：

1. 可以使网络训练更快。相比于sigmoid、tanh，导数更加好求，反向传播就是不断的更新参数的过程，因为其导数不复杂形式简单。
2. 增加网络的非线性。本身为非线性函数，加入到神经网络中可以是网络拟合非线性映射。
3. 防止梯度消失。当数值过大或者过小，sigmoid, tanh的导数接近于0，relu为非饱和激活函数不存在这种现象。
4. 使网络具有稀疏性。

- 缺点：

1. ReLU的输出不是0均值的。
2. Dead ReLU Problem(神经元坏死现象)：

5.3 批标准化

- 优点： 1.可以解决内部协变量偏移，简单来说训练过程中，各层分布不同，增大了学习难度，BN缓解了这个问题。当然后来也有论文证明BN有作用和这个没关系，而是可以使损失平面更加的平滑，从而加快收敛速度。

2. 缓解了梯度饱和问题（如果使用sigmoid这种含有饱和区间的激活函数的话），加快收敛。

- 缺点：

1. batch_size较小的时候，效果差。BN的过程，是使用batch中样本的均值和方差来模拟全部数据的均值和方差。在batch_size 较小的时候，模拟出来的肯定效果不好
2. BN在RNN中效果比较差。因为RNN的输入是长度是动态的，就是说每个样本的长度是不一样的。。

5.4 dropout

- Dropout在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征
- 当前Dropout被大量利用于全连接网络，而且一般认为设置为0.5或者0.3，而在卷积网络隐藏层中由于卷积自身的稀疏化以及稀疏化的ReLU函数的大量使用等原因，Dropout策略在卷积网络隐藏层中使用较少。

- 缺点：

1. dropout 的一大缺点是成本函数无法被明确定义。因为每次迭代都会随机消除一些神经元结点的影响，因此无法确保成本函数单调递减。
2. 明显增加训练时间，因为引入 dropout 之后相当于每次只是训练的原先网络的一个子网络，为了达到同样的精度需要的训练次数会增多。dropout 的缺点就在于训练时间是没有 dropout 网络的 2-3 倍

5.5 L1

- L1追求的是稀疏，可以理解为变量个数少，L2主要用于处理过拟合问题，让每个权重参数值小
- 但是实际训练过程中由于调参数不是十分熟练的原因，L1正则化防止过拟合的作用在我的模型上体现一般

5.5 总结

- 总体来说全连接网络没有卷积网络好使，从训练速度和训练指标上都有体现

6.主体代码解释

- 数据集下载

```
# 下载训练集
BATCH_SIZE = 64
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=True, download=False,
                   transform=transforms.Compose([transforms.ToTensor(),
                                                  transforms.Normalize((0.1037,),
                                                                      (0.3081,))])), batch_size=BATCH_SIZE,
    shuffle=True)

# 测试集
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1037,), (0.3081,))
    ])),
    batch_size=BATCH_SIZE, shuffle=True)
```

- 可以自己选择训练模型

```
choice = int(input("choose the model\n"
                  "1:One_Layer_Net\n"
                  "2:Sigmoid_Net1\n"
                  "3:ReLU_Net1\n"
                  "4:Batch_Net1\n"
                  "5:Drop_Net1\n"
                  "6:Three_Layer_Net\n"
                  "7:Sigmoid_Net2\n"
                  "8:ReLU_Net2\n"
                  "9:Batch_Net2\n"
                  "10:Drop_Net2\n"
                  "11:Dense_Net+L1\n"
                  "12:Dense_Net+L2\n"
                  "13:LeNet\n:"))
```

- 训练开始

```
for epoch in range(epoch):
    net.train()
    sum_loss = 0.0
    for i, data in enumerate(train_loader):
        inputs, labels = data
        inputs, labels = Variable(inputs).to(device),
        Variable(labels).to(device)
        optimizer.zero_grad() # 将梯度归零
```

```

outputs = net(inputs) # 将数据传入网络进行前向运算
loss = criterion(outputs, labels) # 得到损失函数
loss.backward() # 反向传播
optimizer.step() # 通过梯度做一步参数更新

# print(loss)
sum_loss += loss.item()
if i % 100 == 99:
    print('[%d,%d] loss:%.03f' %
          (epoch + 1, i + 1, sum_loss / 100))
    Loss_list.append(sum_loss / 100)
    sum_loss = 0.0

```

- 开始测试

```

net.eval() # 将模型变换为测试模式
correct = 0
total = 0
for data_test in test_loader:
    images, labels = data_test
    images, labels = Variable(images).to(device), Variable(labels).to(device)
    output_test = net(images)
    _, predicted = torch.max(output_test, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()

```

- 保存参数

```

# 保存
if not os.path.exists("./parameterForMnist"):
    os.mkdir("./parameterForMnist")
torch.save(net.state_dict(),
            './parameterForMnist/parameter{}.pk1'.format(choice))

```

- 作图

```

# 作图
x1 = range(0, len(Accuracy_list))
x2 = range(0, len(Loss_list))
y1 = Accuracy_list
y2 = Loss_list
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('Test accuracy vs. epoches')
plt.ylabel('Test accuracy')
plt.subplot(2, 1, 2)

```

```
plt.plot(x2, y2, '-.')
plt.xlabel('Test loss vs. 100')
plt.ylabel('Test loss')
plt.text(x=1,y=1,s="acc={}".format(((correct.item() /
len(test_loader.dataset)))))
plt.savefig("./parameterForMnist/mnist_accuracy_loss{0}.jpg".format(choice))
plt.show()
```

7.作者

ID	Name
1852824	吴杨婉婷

指导老师 唐堂老师

联系方式 email: 1852824@tongji.edu.cn