研究生《深度学习》课程

实验报告

实验名称：	前馈神经网络
姓　　名：	唐麒
学　　号：	21120299
上课类型：	专业课
日　　期：	2022 年 10 月 14 日

一、实验内容

本实验报告包括的实验有手动实现前馈神经网络解决回归、二分类、多分类任务，再利用 torch.nn 对上述三个任务进行实现。在多分类实验的基础上将使用至少三种不同的激活函数，并对使用不同激活函数的实验结果进行对比。除此之外，还需评估多分类任务中的模型隐藏层层数和隐藏单元个数对实验结果的影响。在探讨模型设计对实验结果的影响后，还将进一步谈到训练和优化策略对模型结果的影响，具体包括在在多分类任务实验中分别手动实现和用 torch.nn 实现 dropout、L_2 正则化和采用 10 折交叉验证。通过本次实验，学生将对前馈神经网络的原理及实现、激活函数的选择以及训练和优化策略对模型性能的影响有较为清晰的认识。

1.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

实验一具体内容为手动实现前馈神经网络以在不同的数据集上完成回归、二分类和多分类任务，对实验结果进行分析并绘制训练集和测试集的 loss 曲线。

实验一涉及的内容主要为在了解前馈神经网络原理的基础上进行手动实现和在生成的数据集上训练优化，该实验可以帮助学生更好地了解和熟悉前馈神经网络的原理。

实验涉及的算法主要包括前馈神经网络的搭建和优化。

1.2 利用 torch.nn 实现前馈神经网络解决上述回归、二分类、多分类任务

实验二具体内容为利用 torch.nn 实现前馈神经网络以在不同的数据集上完成回归、二分类和多分类任务，对实验结果进行分析并绘制训练集和测试集的 loss 曲线。

实验二涉及的内容主要为在了解前馈神经网络原理的基础上利用 torch.nn 实现和在生成的数据集上训练优化，该实验可以帮助学生更好地了解和熟悉前馈神经网络的原理以及 torch.nn 的使用。

实验涉及的算法主要包括前馈神经网络的搭建和优化。

1.3 在多分类实验的基础上使用至少三种不同的激活函数

实验三具体内容为在多分类实验的基础上，对模型采用的激活函数进行修改，采用最少三种不同的激活函数进行训练，绘制训练集和测试集的 loss 曲线并对实验结果进行分析。

实验三涉及的内容主要为在了解前馈神经网络原理的基础上利用 torch.nn 实现和在生成的数据集上训练优化，并对不同激活函数进行选择和使用，该实验可以帮助学生更好地了解和熟悉不同激活函数的区别和效果。

实验涉及的算法主要包括前馈神经网络的搭建和优化以及激活函数的使用（如 ReLU、LeakyReLU、PReLU 等）。

1.4 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

实验四具体内容为使用不同的隐藏层层数和隐藏单元个数，进行对比实验并分析实验结果；对比至少三组。

实验四涉及的内容主要为帮助学生理解隐藏层数和隐藏单元数量的增多并不一定会提升模型的性能，从而为引入 dropout 等防止过拟合策略做铺垫。

1.5 在多分类任务实验中分别手动实现和用 torch.nn 实现 dropout

实验五具体内容为在多分类实验中分别手动实现和用 torch.nn 实现 dropout，绘制训练集和测试集的 loss 曲线并对实验结果进行分析。

实验五涉及的内容主要为通过两种不同的实验方式帮助学生理解 dropout 在模型优化和防止过拟合等方面的原理，从而在今后的实验和学习中更好地利用这一优化策略。

实验涉及的算法主要包括使用 dropout 将前馈神经网络隐藏层中的隐藏单元 h_i 以一定概率被丢弃掉。

1.6 在多分类任务实验中分别手动实现和用 torch.nn 实现 L_2 正则化

实验六具体内容为在多分类实验中分别手动实现和用 torch.nn 实现 L_2 正则化，绘制训练集和测试集的 loss 曲线并对实验结果进行分析。

实验六涉及的内容主要为通过两种不同的实验方式帮助学生理解 L_2 正则化在模型优化和防止过拟合等方面的原理，从而在今后的实验和学习中更好地利用这一优化策略。

实验涉及的算法主要包括在模型原损失函数基础上添加 L_2 范数惩罚项，通过惩罚绝对值较大的模型参数为需要学习的模型增加限制，来应对过拟合问题。带有 L_2 范数惩罚项的模型的新损失函数为:

$$\ell_o + \frac{\lambda}{2n}|w|^2$$

其中 w 是参数向量，ℓ_o 是模型原损失函数，n 是样本个数，λ 是超参数

1.7 对回归、二分类、多分类任务分别选择上述实验中效果最好的模型，采用 10 折交叉验证评估实验结果

实验七具体内容为利用 10 折交叉验证评估回归、二分类、多分类任务的实验结果。

实验七涉及的内容主要为为了帮助学生理解 K 折交叉验证，由于验证数据集不参与模型训练，当训练数据不够用时，预留大量的验证数据显得太奢侈，一种改善方法便是 K 折交叉验证。

实验涉及的算法主要为将数据集分层采样划分为 K 个大小相似的互斥子集，每次用 K-1 个子集的并集作为训练集，余下的子集作为测试集，最终返回 K 个测试结果的均值，K 最常用的取值是 10。

二、实验设计

本次实验所包含的内容皆为指定内容，涉及部分数据加载、模型及其损失函数和优化器的实现，故此部分省略，在报告的后续内容中进行介绍。

三、实验环境及实验数据集

实验所用的开发环境如表 1 所示。

表 1 实验环境

开发环境	版本
操作系统	macOS Big Sur 11.5.1
开发工具	Jupyter Notebook
Python	3.8.2
PyTorch	1.8.0 (CPU)

本次实验所涉及的数据集包括：两个人工构造的数据集和 MNIST 数据集。

用于回归任务的数据集大小为 10000 且训练集大小为 7000，测试集大小为 3000 数据集的样本特征维度 p 为 500，且服从如下的高维线性函数:$y = 0.028 + \sum_{i=1}^{p} 0.0056\, x_i$，此外设置噪声项服从均值为 0、标准差为 0. 001 的正态分布。

用于二分类任务的数据集在生成过程中包含两个数据集，大小均为 10000 且训练集大小为 7000，测试集大小为 3000。两个数据集的样本特征 x 的维度均为 200，且分别服从均值互为相反数且方差相同的正态分布。两个数据集的样本标签分别为 0 和 1。

MNIST 手写体数据集包含 60,000 个用于训练的图像样本和 10,000 个用于测试的图像样本，大小均为 28×28 像素，其值为 0 到 1，如图 1 所示。

图 1 MNIST 数据集（部分）

四、实验过程

本小节将对实验的具体内容进行详细说明。

首先先对生成三个数据集的代码进行介绍和展示。

（1） 生成用于回归任务的数据集。

```
num_train, num_test, num_inputs,num_hiddens, num_outputs= 7000, 3000, 500, 256, 1
true_w, true_b = torch.ones(num_inputs, 1) * 0.0056, 0.028
features = torch.randn((num_train + num_test, num_inputs))
```

```
labels = torch.matmul(features, true_w)+true_b
labels += torch.tensor(np.random.normal(0, 0.001, size=labels.size()), dtype=torch.float)
train_features, test_features = features[:num_train, :], features[num_train:, :]
train_labels, test_labels = labels[:num_train], labels[num_train:]
```

（2） 生成用于二分类任务的数据集。

```
num_train, num_test,num_inputs, num_hiddens, num_outputs = 7000, 3000, 200, 100, 2
x1 = torch.tensor(np.random.normal(1, 4, [num_train+num_test, num_inputs]),
dtype=torch.float)
y1 = torch.ones(num_train+num_test, dtype=torch.long)
x2 = torch.tensor(np.random.normal(-1, 4, [num_train+num_test, num_inputs]),
dtype=torch.float)
y2 = torch.zeros(num_train+num_test, dtype=torch.long)
# 划分训练集测试集
train_data = torch.cat((x1[:num_train,:], x2[:num_train,:]),0)
train_label = torch.cat((y1[:num_train], y2[:num_train]),0)
test_data = torch.cat((x1[num_train:,:], x2[num_train:,:]),0)
test_label = torch.cat((y1[num_train:], y2[num_train:]),0)
```

（3） 下载和加载用于多分类任务的数据集。

```
train_dataset = torchvision.datasets.MNIST(root="./", train=True, transform=
transforms.ToTensor(),download=True)
test_dataset = torchvision.datasets.MNIST(root="./", train=False, transform=
transforms.ToTensor(),download=True)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True,num_workers=0)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64,
shuffle=False,num_workers=0)
```

为了利用图表对训练损失曲线和测试损失曲线进行展示，通过将损失值存储来进行曲线图像的绘制。

```
tx_l = []
train_loss = []
test_loss = []
def Draw_Loss_Curve(title):
    plt.figure(figsize=(15, 9))
    plt.xlabel("epoch")
    plt.ylabel("loss")
```

```
        plt.title(title)
        plt.plot(x_l, train_loss, label=u'Train Loss')
        plt.legend()
        p2 = plt.plot(x_l, test_loss, label=u'Test Loss')
        plt.legend()
        plt.show()
```

在下文的介绍中，将重点介绍与实验探究目的相关的代码，数据加载、模型训练等基本骨架代码将不再进行详细展示。

4.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

定义激活函数和前馈神经网络。

```
def relu(x):
x[x <= 0] = 0
x[x > 0] = 1
return x
# def relu(x):
# return torch.max(input=x,other=torch.tensor(0.0))
def net(X):
        X=X.view((-1,num_inputs))
        H = relu(torch.matmul(X, W1.t()) + b1)
        return torch.matmul(H,W2.t())+b2
```

在损失函数的选择上，回归任务采用了均方损失函数，分类任务采用了交叉熵损失函数。

4.2 利用 torch.nn 实现前馈神经网络解决上述回归、二分类、多分类任务

定义前馈神经网络。

```
class FlattenLayer(torch.nn.Module):
        def __init__(self):
            super(FlattenLayer, self).__init__()
        def forward(self, x):
            return x.view(x.shape[0], -1)
net = nn.Sequential(
        FlattenLayer(),
```

```
        nn.Linear(500,256),
        nn.ReLU(),
        nn.Linear(256,1),
    )
```

在损失函数的选择上，回归任务采用了均方损失函数，分类任务采用了交叉熵损失函数。

4.3 在多分类实验的基础上使用至少三种不同的激活函数

```
net = nn.Sequential(
        FlattenLayer(),
        nn.Linear(500,256),
        nn.ReLU(),
        nn.LeakyReLU(),
        nn.PReLU(),
        nn.Linear(256,1),
    )
```

4.4 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

该实验的实现主要依赖于修改网络的超参数，包括隐藏层的层数和隐藏单元个数，具体代码在此不再展示。

4.5 在多分类任务实验中分别手动实现和用 torch.nn 实现 dropout

（1）手动实现

```
def dropout(X,drop_prob):
        X = X.float()
        assert 0 <= drop_prob <= 1
        keep_prob = 1-drop_prob
        if keep_prob == 0:
            return torch.zeros_like(X)
        mask = (torch.rand(X.shape)<keep_prob).float()
        return mask * X / keep_prob
```

（2）torch.nn 实现

```
net = nn.Sequential(
        FlattenLayer(),
        nn.Linear(500,256),
        nn.ReLU(),
        nn.Dropout(drop_prob),
        nn.Linear(256,1),
    )
```

4.6 在多分类任务实验中分别手动实现和用 torch.nn 实现 L_2 正则化

（1）手动实现

```
def l2_penalty(w):
        return (w**2).sum() / 2
"""
in train()
"""
l = loss(y_hat, y).sum() + lambd*l2_penalty(W1) + lambd*l2_penalty(W2)
```

（2）torch.nn 实现

```
"""
in train()
"""
optimizer = torch.optim.SGD(net.parameters(), lr=0.1,weight_decay=wd)
```

4.7 对回归、二分类、多分类任务分别选择上述实验中效果最好的模型，采用 10 折交叉验证评估实验结果

```
def get_kfold_data(k, i, X, y):
        fold_size = X.shape[0] // k
        val_start = i * fold_size
        if i != k - 1:
            val_end = (i + 1) * fold_size
            X_valid, y_valid = X[val_start:val_end], y[val_start:val_end]
```

```
            X_train = torch.cat((X[0:val_start], X[val_end:]), dim = 0)
            y_train = torch.cat((y[0:val_start], y[val_end:]), dim = 0)
        else:
            X_valid, y_valid = X[val_start:], y[val_start:]
            X_train = X[0:val_start]
            y_train = y[0:val_start]
        return X_train, y_train, X_valid, y_valid
```

五、实验结果

5.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

（1）回归任务

回归任务训练了 100 个 epoch，模型在训练集和测试集上的 loss 曲线如图 2 所示，均呈下降趋势。

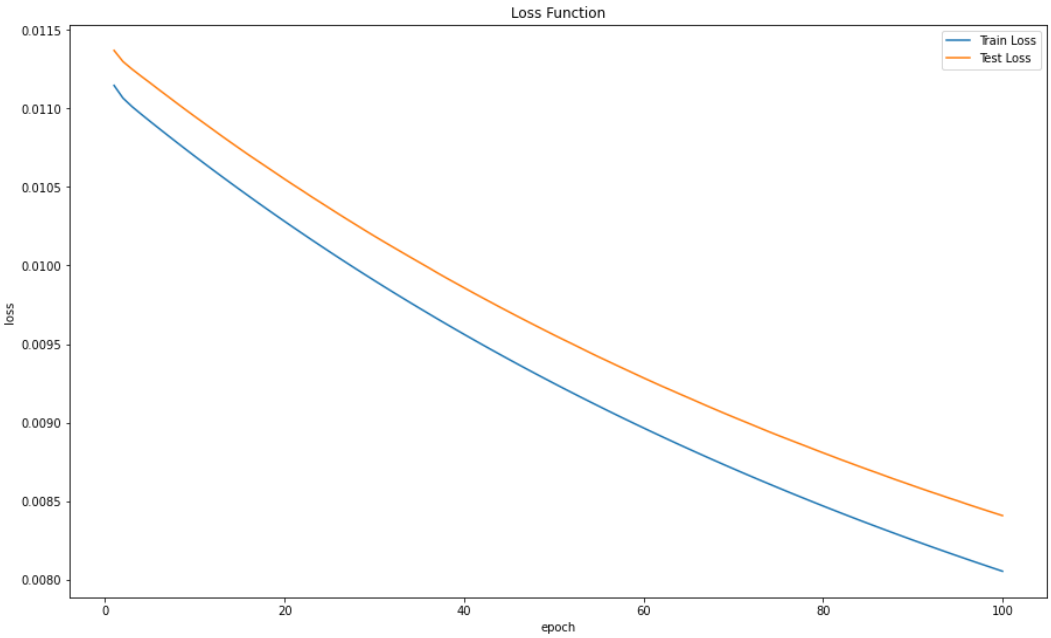


图 2 手动实现前馈神经网络解决回归任务损失函数曲线图

（2）二分类任务

二分类任务训练了 100 个 epoch，模型在训练集和测试集上的 loss 曲线如图 3 所示，均呈下降趋势。

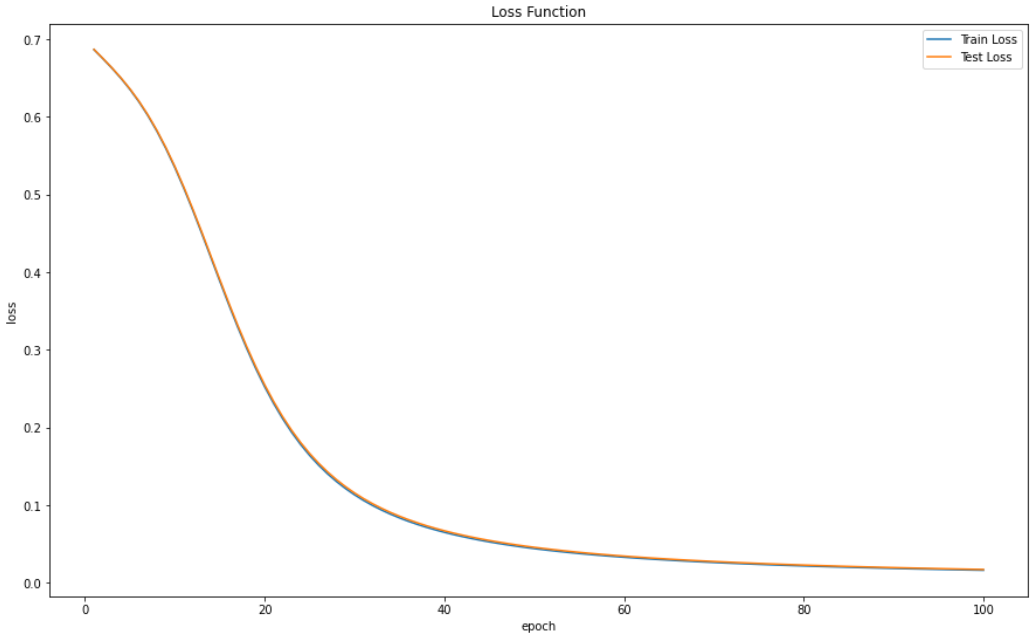


图 3 手动实现前馈神经网络解决二分类任务损失函数曲线图

（3）多分类任务

多分类任务训练了 30 个 epoch，模型在训练集和测试集上的 loss 曲线如图 4 所示，均呈下降趋势。

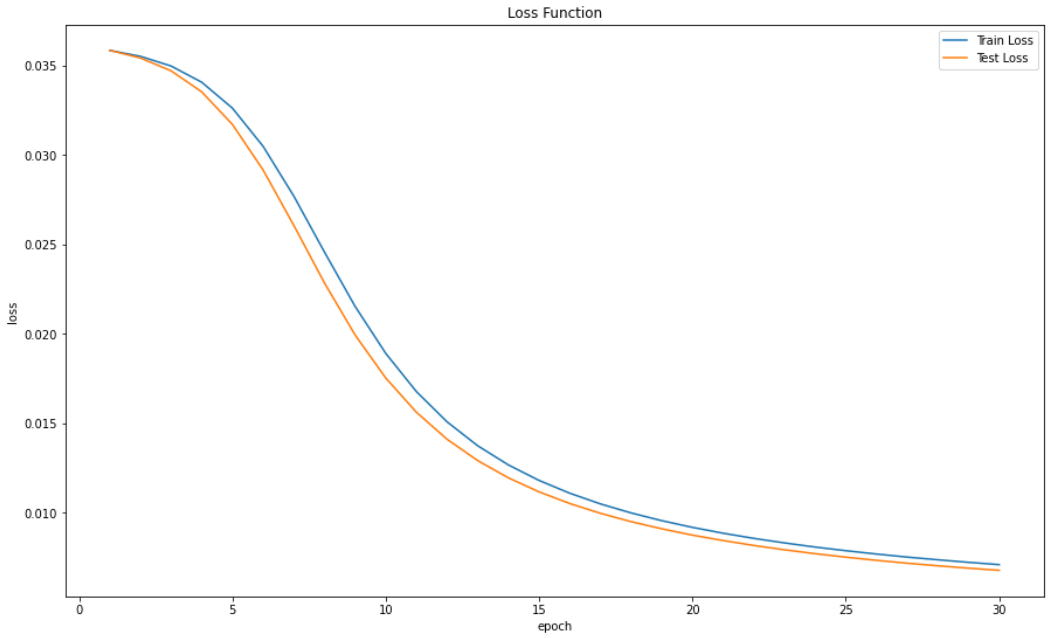


图 4 手动实现前馈神经网络解决多分类任务损失函数曲线图

5.2 利用 torch.nn 实现前馈神经网络解决上述回归、二分类、多分类任务

（1）回归任务

回归任务训练了 100 个 epoch，模型在训练集和测试集上的 loss 曲线如图 5 所示，均呈下降趋势。

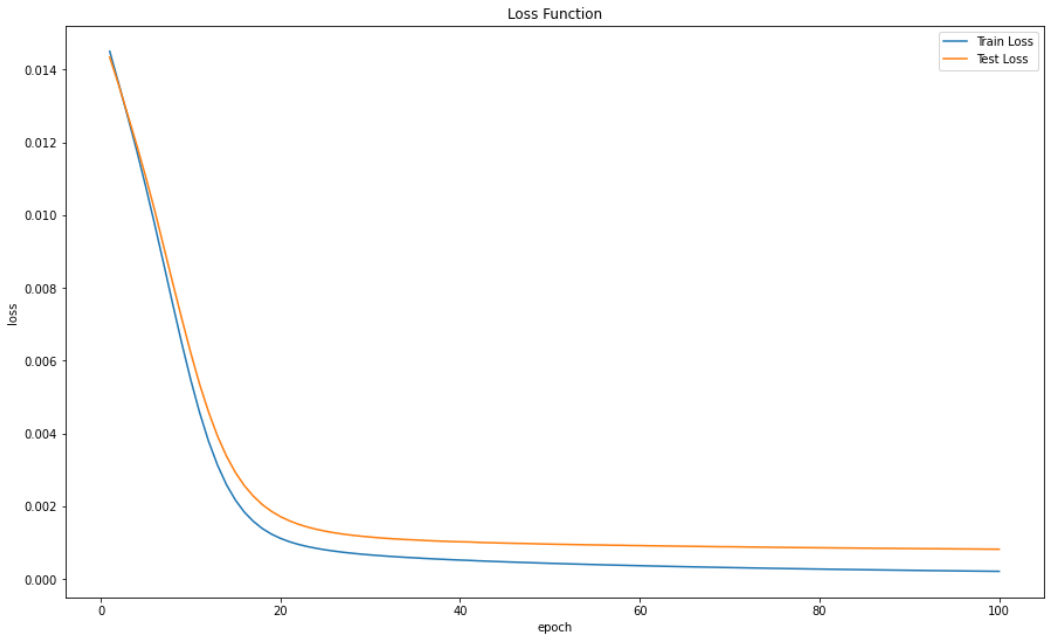


图 5 利用 torch.nn 实现前馈神经网络解决回归任务损失函数曲线图

（2）二分类任务

二分类任务训练了 100 个 epoch，模型在训练集和测试集上的 loss 曲线如图 6 所示，均呈下降趋势。

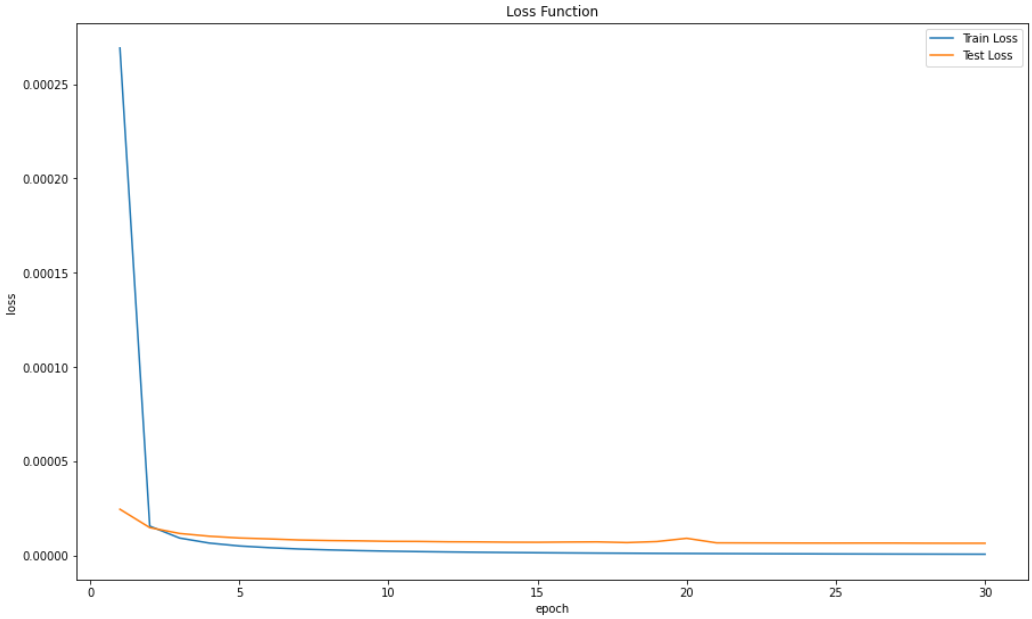


图 6 利用 torch.nn 实现前馈神经网络解决二分类任务损失函数曲线图

（3）多分类任务

多分类任务训练了 30 个 epoch，模型在训练集和测试集上的 loss 曲线如图 7 所示，均呈下降趋势。

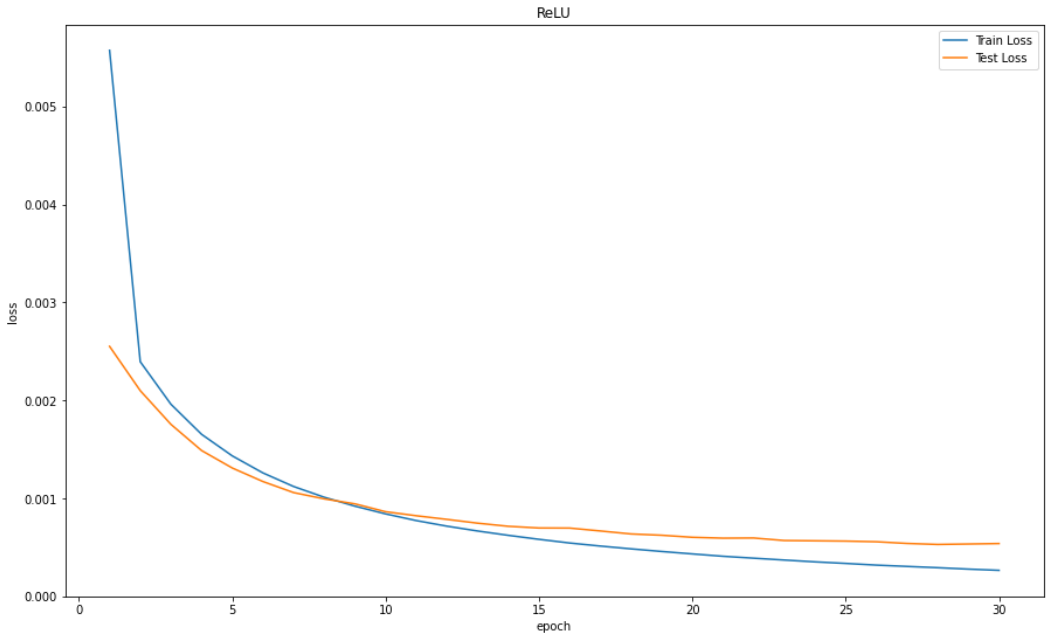
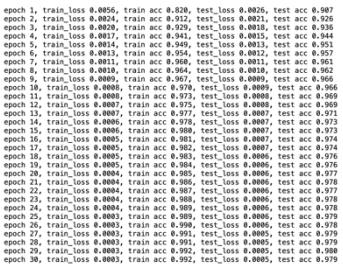
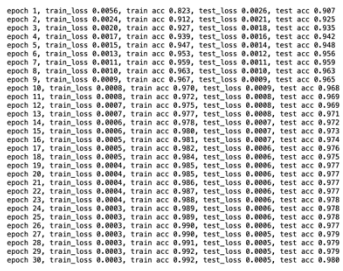
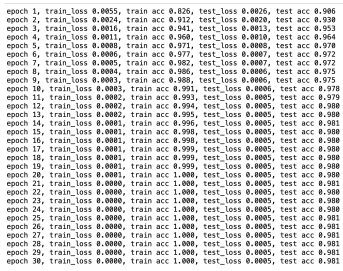
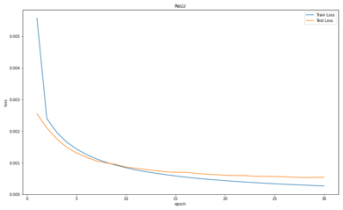
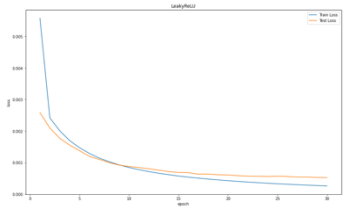
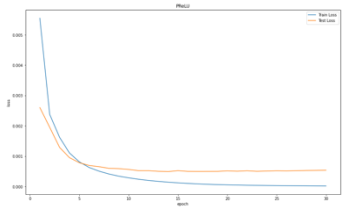


图 7 利用 torch.nn 实现前馈神经网络解决多分类任务损失函数曲线图

5.3 在多分类实验的基础上使用至少三种不同的激活函数

在该实验中，选取了三种激活函数：ReLU、LeakyReLU、PReLU，从训练损失和测试损失来看，三种激活函数的效果相差不多，PReLU 的训练损失相对较小，但同时也造成了过拟合。此外，从测试准确率来看，使用 PReLU 作为激活函数的模型性能相对较高且收敛。

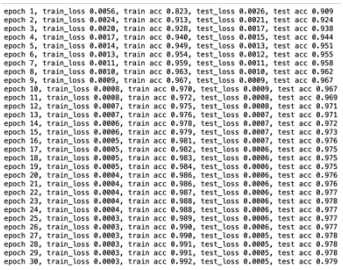
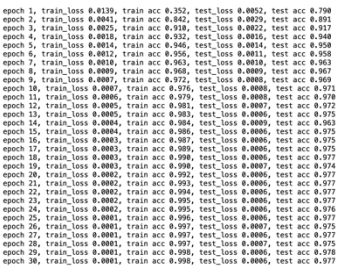
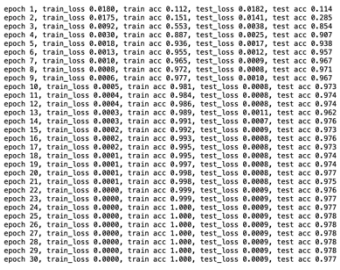
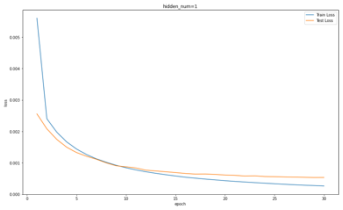
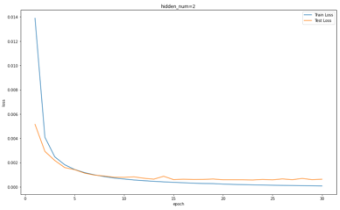
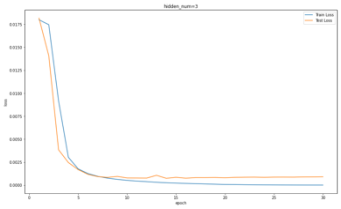
表 2 不同激活函数的训练和测试准确率及损失函数

ReLU	LeakyReLU	PReLU
		
		

5.4 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

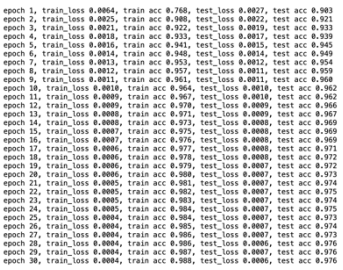
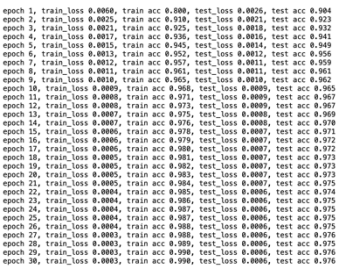
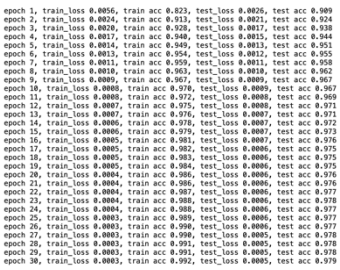
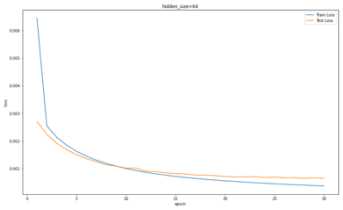
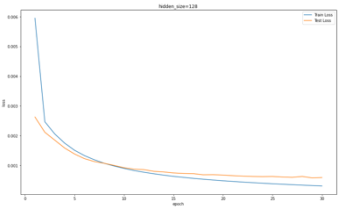
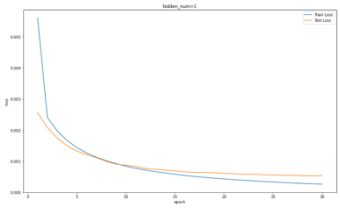
在之前的实验中，隐藏层层数为 1，隐藏单元个数为 256，首先分别修改隐藏层层数为 2 层和 3 层，且只改变隐藏层层数。实验结果表明，网络架构不是越深越好。

表 3 不同隐藏层数的训练和测试准确率及损失函数

hidden_num=1	hidden_num=2	hidden_num=3
		
		

然后分别修改隐藏单元个数为 64 个和 128 个，且只改变隐藏单元个数，隐藏层层数均为 1。实验结果表明 256 个隐藏单元数的模型性能更好（可能与模型输入层的神经元个数为 784 有关）。

表 4 不同隐藏单元个数的训练和测试准确率及损失函数

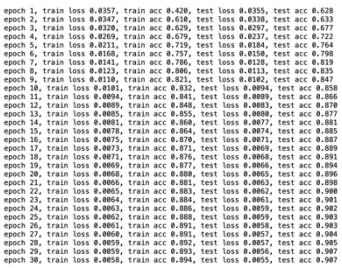
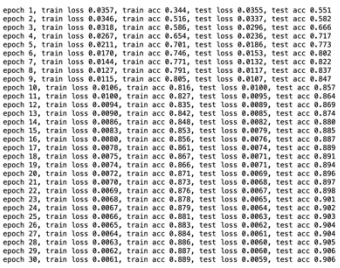
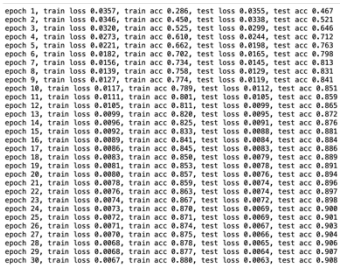
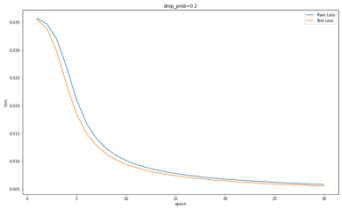
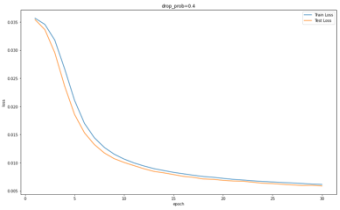
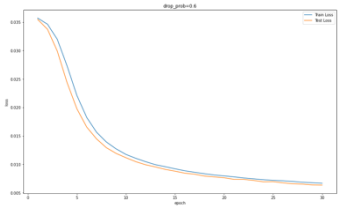
hidden_size=64	hidden_size=128	hidden_size=256
		
		

5.5 在多分类任务实验中分别手动实现和用 torch.nn 实现 dropout

在该实验中，分别设置了 0.2、0.4 和 0.6 三种丢弃率来探究不同丢弃率对实验结果的影响。实验结果表明，设置合适的丢弃率可以应对过拟合问题，提高网络的性能。

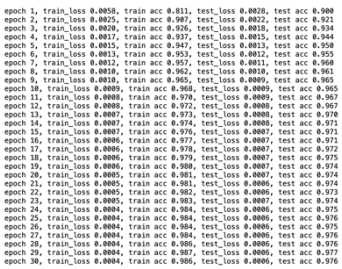
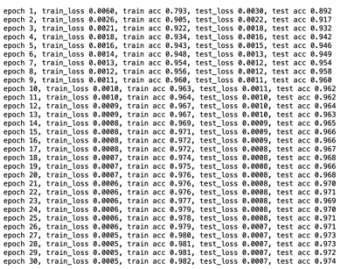
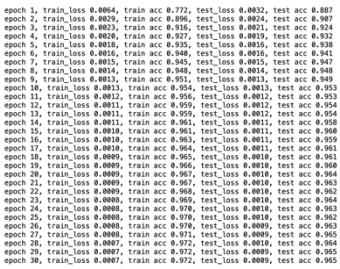
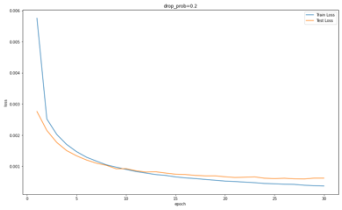
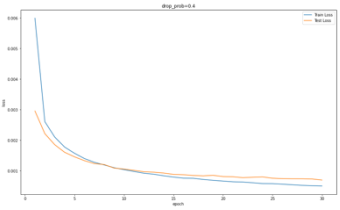
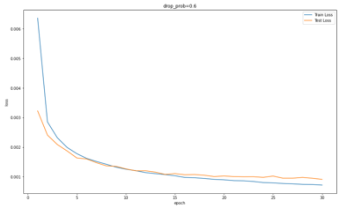
（1）手动实现

表 5 不同丢弃率的训练和测试准确率及损失函数

drop_prob=0.2	drop_prob=0.4	drop_prob=0.6
		
		

（2）torch.nn 实现

表 6 不同丢弃率的训练和测试准确率及损失函数

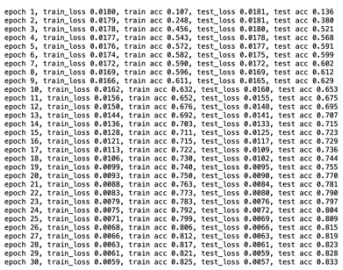
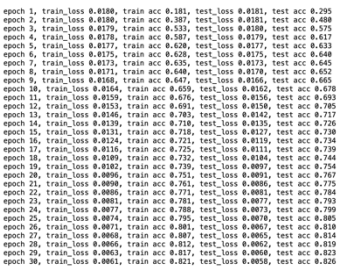
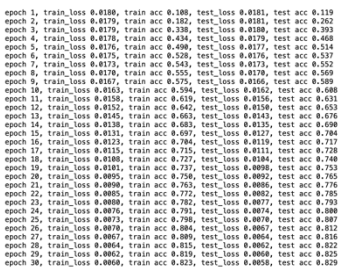
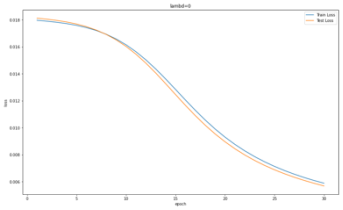
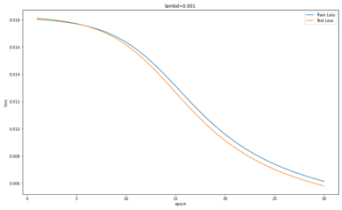
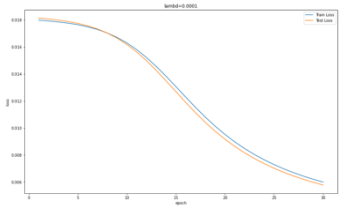
drop_prob=0.2	drop_prob=0.4	drop_prob=0.6
		
		

5.6 在多分类任务实验中分别手动实现和用 torch.nn 实现 L_2 正则化

在该实验中，分别设置了 0，0.001 和 0.0001 三种权重来探究惩罚项的权重不同对实验结果的影响。实验结果表明，合适的权重可以提高网络的性能，提高准确率，降低损失值。

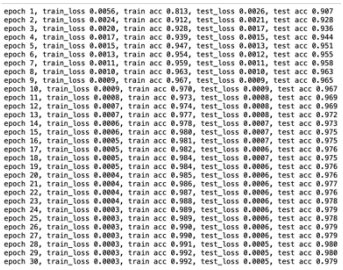
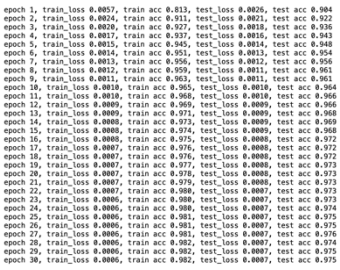
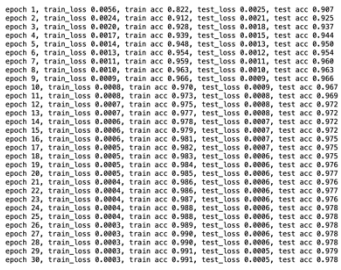
（1）手动实现

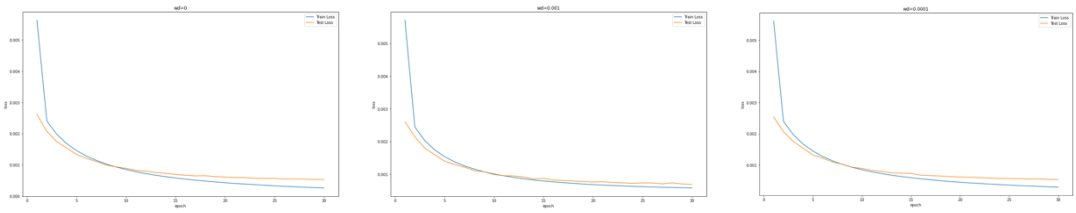
表 7 不同惩罚项的权重的训练和测试准确率及损失函数

lambd=0	lambd=0.001	lambd=0.0001
		
		

（2）torch.nn 实现

表 8 不同惩罚项的权重的训练和测试准确率及损失函数

wd=0	wd=0.001	wd=0.0001
		



5.7 对回归、二分类、多分类任务分别选择上述实验中效果最好的模型，采用 10 折交叉验证评估实验结果

实验结果如下表所示。

表 9 不同任务在 10 折交叉验证的损失函数表

	回归任务		二分类任务		多分类任务	
	Train	Valid	Train	Valid	Train	Valid
1	0.0141	0.0133	0.0149	0.0178	0.0011	0.0011
2	0.0141	0.0133	0.0151	0.0179	0.0011	0.0012
3	0.0141	0.0133	0.0148	0.0174	0.0011	0.0013
4	0.0141	0.0133	0.0155	0.0189	0.0011	0.0011
5	0.0141	0.0133	0.0155	0.0194	0.0011	0.0012
6	0.0141	0.0133	0.0148	0.0178	0.0011	0.0012
7	0.0141	0.0133	0.0152	0.0189	0.0011	0.0012
8	0.0141	0.0133	0.0150	0.0175	0.0011	0.0013
9	0.0141	0.0133	0.0153	0.0194	0.0011	0.0013
10	0.0141	0.0133	0.0151	0.0179	0.0011	0.0009
Avg	0.0141	0.0133	0.0151	0.0183	0.0011	0.0012

表 10 不同任务在 10 折交叉验证的准确率表

	二分类任务		多分类任务	
	Train	Valid	Train	Valid
1	1.000	1.000	0.917	0.924
2	1.000	1.000	0.915	0.910
3	1.000	1.000	0.918	0.903
4	1.000	1.000	0.916	0.922
5	1.000	1.000	0.917	0.914
6	1.000	1.000	0.916	0.904
7	1.000	1.000	0.916	0.911
8	1.000	1.000	0.916	0.904
9	1.000	1.000	0.917	0.907
10	1.000	1.000	0.914	0.932
Avg	1.000	1.000	0.916	0.913

六、实验心得体会

过本次实验，对前馈神经网络的原理和实现有了更扎实的理解和掌握，尤其是不同实验中对优化策略的设计，解决了我在之前自学过程中的迷惑，从而在今后的深度学习编程中可以更好地利用 dropout 和 L_2 正则化等策略对网络进行优化。

而在实验报告撰写的过程中，可以对实验的目的、问题和解决方案进行明确的梳理，进一步在编码实验的基础上获得了提升，做到了理论和实践的齐头并进。

七、参考文献

[1] 北京交通大学 《深度学习》课程组，实验 2 前馈神经网络实验，北京交通大学《深度学习》课件

八、附录

无