研究生《深度学习》课程

实验报告

实验名称：	实验 2 前馈神经网络实验
姓　　名：	郑楚彬
学　　号：	21140129
上课类型：	专业课
日　　期：	2021.09.18

一、实验内容

二、实验设计

三、实验环境及实验数据集

1. 实验环境

Ubuntu 18.04、CPU 4 核、32GB、Pytorch 1.6.0、Jupyter Notebook

2. 数据集：

□ 回归任务

- 生成单个数据集。
- 数据集的大小为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: $y = 0.028 + \sum_{i=1}^{p} 0.0056 x_i$

```
torch.random.manual_seed(1)

p = 500
x_train = torch.randn((7000, p))
y_train = 0.028 + 0.0056 * torch.sum(x_train, dim=1)

x_test = torch.randn((3000, p))
y_test = 0.028 + 0.0056 * torch.sum(x_test, dim=1)

x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
(torch.Size([7000, 500]),
 torch.Size([7000]),
 torch.Size([3000, 500]),
 torch.Size([3000]))
```

□ 二分类任务

- 共生成两个数据集。
- 两个数据集的大小均为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` 。

```
torch.random.manual_seed(1)

p = 200
mu = 5
std = 0.1
train_size = 7000
test_size = 3000

x_train = torch.vstack([
    torch.normal(mean=mu, std=std, size=(train_size, p)),
    torch.normal(mean=-mu, std=std, size=(train_size, p))
])
y_train = torch.hstack([
    torch.ones(train_size, dtype=torch.float),
    torch.zeros(train_size, dtype=torch.float),
])
x_test = torch.vstack([
    torch.normal(mean=mu, std=std, size=(test_size, p)),
    torch.normal(mean=-mu, std=std, size=(test_size, p))
])
y_test = torch.hstack([
    torch.ones(test_size, dtype=torch.float),
    torch.zeros(test_size, dtype=torch.float),
])

x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
(torch.Size([14000, 200]),
 torch.Size([14000]),
 torch.Size([6000, 200]),
 torch.Size([6000]))
```

□ 多分类任务

- 该数据集包含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。
- 图像是固定大小(28x28像素)，其值为0到1。为每个图像都被平展并转换为784(28 * 28)个特征的一维numpy数组。

```
import torchvision.transforms as transforms

dataset_dir = '/Users/zhengchubin/PycharmProjects/learn/data/'

# 下载不了数据集
# mnist_train = torchvision.datasets.FashionMNIST(root=dataset_dir, train=True, download=True, transform=transforms.To1
# mnist_test = torchvision.datasets.FashionMNIST(root=dataset_dir, train=False, download=True, transform=transforms.To1

# 手动下载数据集并读取
mnist_train = torchvision.datasets.FashionMNIST(root=dataset_dir, train=True, transform=transforms.ToTensor())
mnist_test = torchvision.datasets.FashionMNIST(root=dataset_dir, train=False, transform=transforms.ToTensor())

mnist_train, mnist_train.data.shape, mnist_train.targets.shape
```

```
(Dataset FashionMNIST
    Number of datapoints: 60000
    Root location: /Users/zhengchubin/PycharmProjects/learn/data/
    Split: Train
    StandardTransform
Transform: ToTensor(),
torch.Size([60000, 28, 28]),
torch.Size([60000]))
```

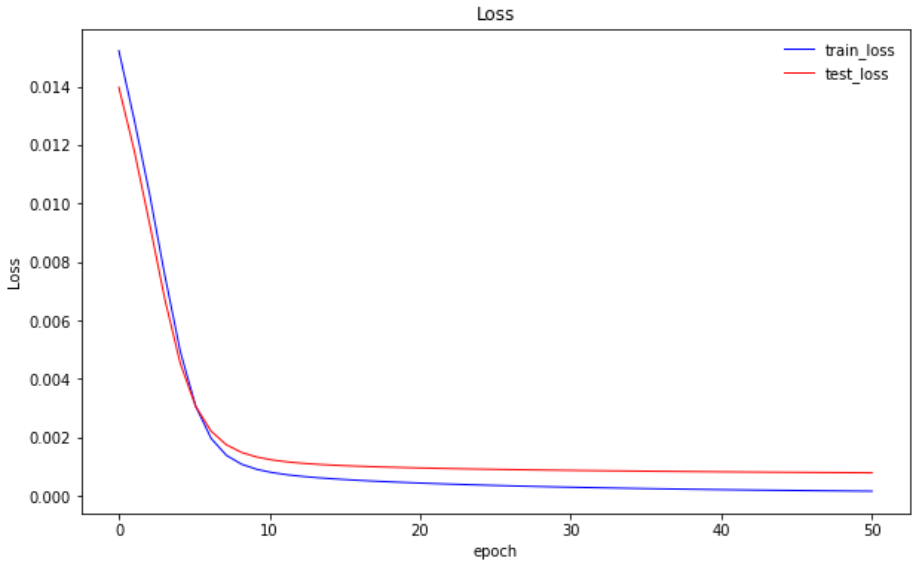
四、实验过程

1. 回归任务

1.1 手动实现

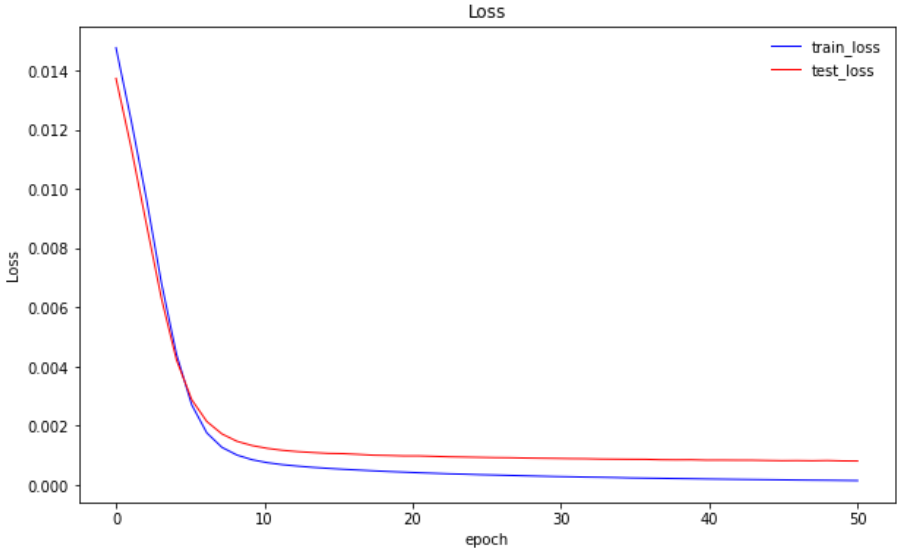
```
def neural_net(x: torch.Tensor, *params):
    """
    前馈神经网络
    :param x:       特征
    :param params:  模型参数
    :return:
    """
    w1, b1, w2, b2 = params
    # hidden = tanh(torch.mm(x, w1) + b1)
    # hidden = leak_relu(torch.mm(x, w1) + b1)
    hidden = relu(torch.matmul(x, w1) + b1)
    return torch.matmul(hidden, w2) + b2
```

训练集与测试集的 loss 均接近 0，说明模型能很好地学习数据集。



1.2 torch.nn 实现

```
class TorchNeuron(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(TorchNeuron, self).__init__()

        self.linear = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size),
        )

        for param in self.linear.parameters():
            nn.init.normal_(param, std=0.01)

    def forward(self, x):
        return self.linear(x)
```

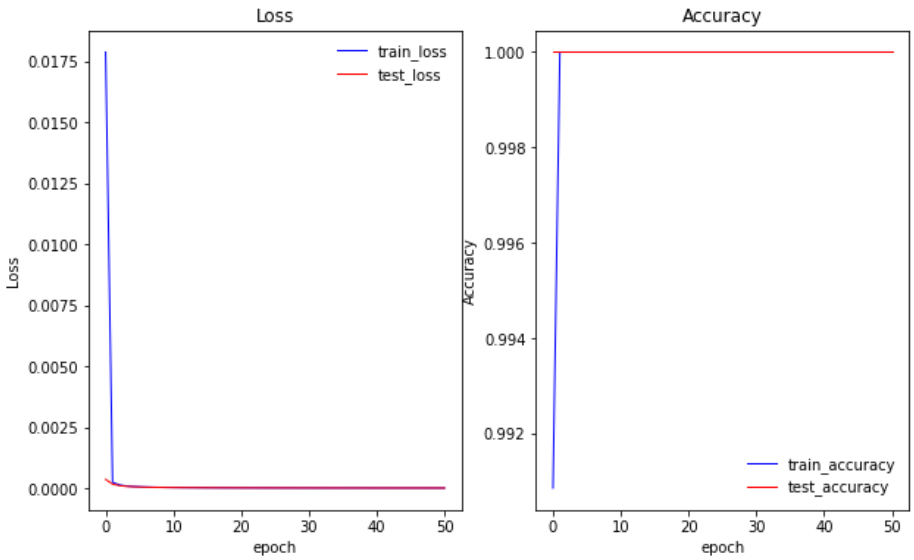


2. 二分类任务

2.1 手动实现

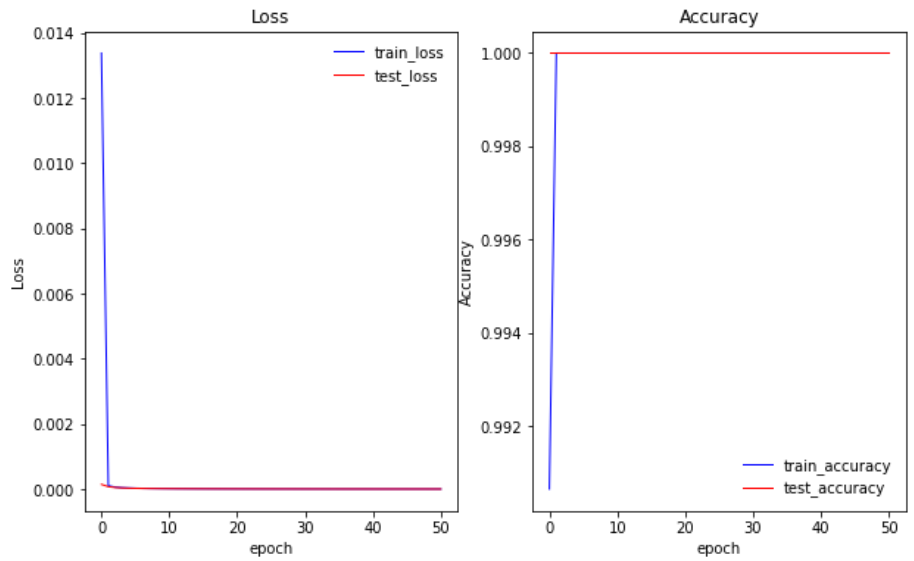
```
def neural_net(x: torch.Tensor, *params):
    """
    前馈神经网络
    :param x:       特征
    :param params:  模型参数
    :return:
    """
    w1, b1, w2, b2 = params
    # hidden = tanh(torch.mm(x, w1) + b1)
    # hidden = leak_relu(torch.mm(x, w1) + b1)
    hidden = relu(torch.matmul(x, w1) + b1)
    return torch.matmul(hidden, w2) + b2
```

训练集与测试集的 loss 均接近 0，并且他们的 accuracy 都等于 1，说明模型能够完美地学习到数据集的规律。



2.2 torch.nn 实现

```
class TorchNeuron(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(TorchNeuron, self).__init__()

        self.linear = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size),
        )

        for param in self.linear.parameters():
            nn.init.normal_(param, std=0.01)

    def forward(self, x):
        return self.linear(x)
```

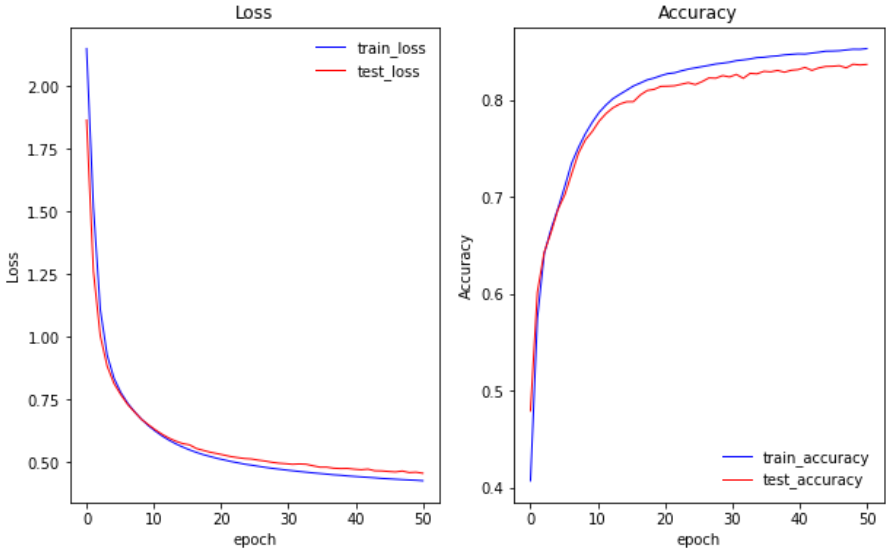


3. 多分类任务

3.1 手动实现

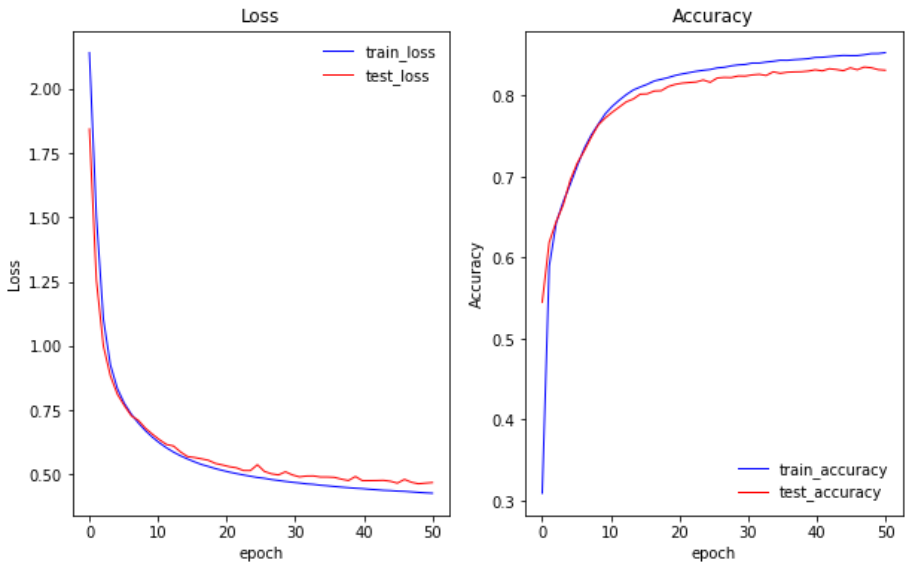
```
def neural_net(x: torch.Tensor, *params):
    """
    前馈神经网络
    :param x:        特征
    :param params:   模型参数
    :return:
    """
    w1, b1, w2, b2 = params
    # hidden = tanh(torch.mm(x, w1) + b1)
    # hidden = leak_relu(torch.mm(x, w1) + b1)
    hidden = relu(torch.matmul(x, w1) + b1)
    return torch.matmul(hidden, w2) + b2
```

模型在训练集和测试集的表现不错，也没有出现过拟合或欠拟合的情况，准确率在训练集上达到 85.2%，在测试集上达到 83.65%。



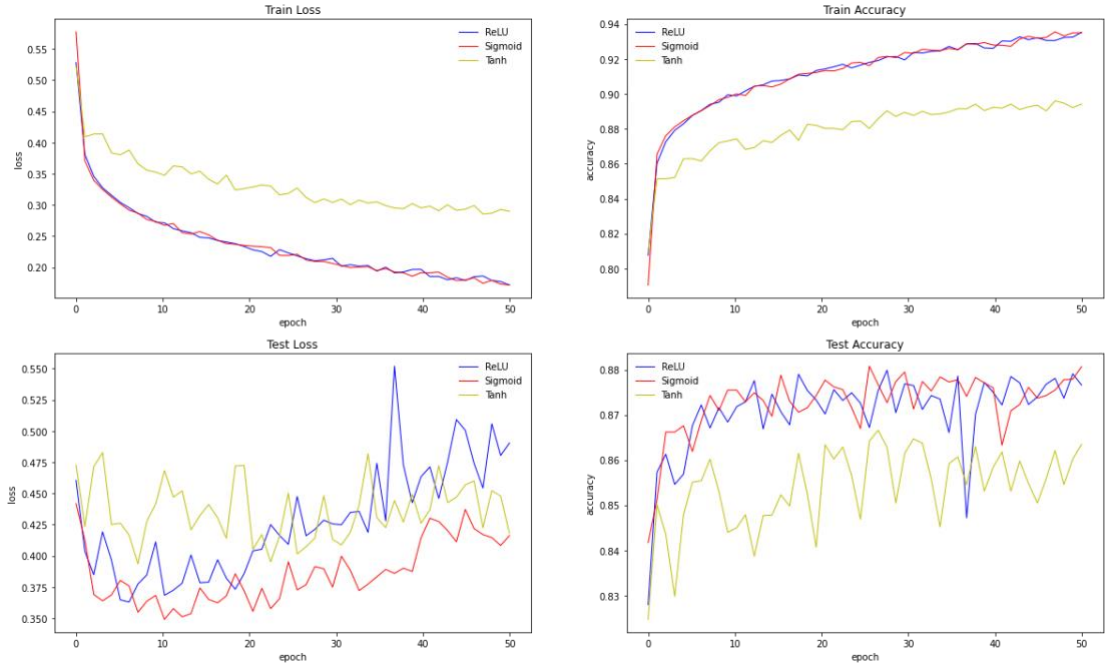
3.2 torch.nn 实现

```
class TorchNeuron(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(TorchNeuron, self).__init__()

        self.linear = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size),
        )

        for param in self.linear.parameters():
            nn.init.normal_(param, std=0.01)

    def forward(self, x):
        return self.linear(x)
```



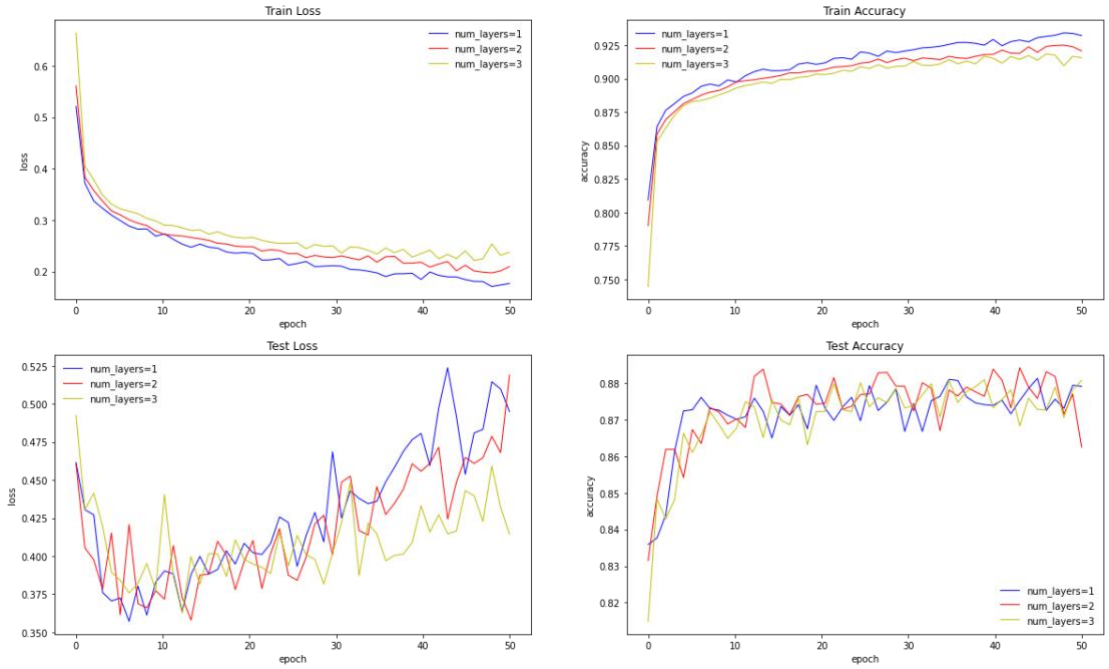
3.3 使用至少 3 种不同的激活函数

分别使用以下 3 种激活函数：Relu、Tanh、Sigmoid。结果如下图，可以看出 Than 函数的效果最差，Relu 和 Sigmoid 相差不大，但激活函数一般不用 Sigmoid，而是使用 Relu。



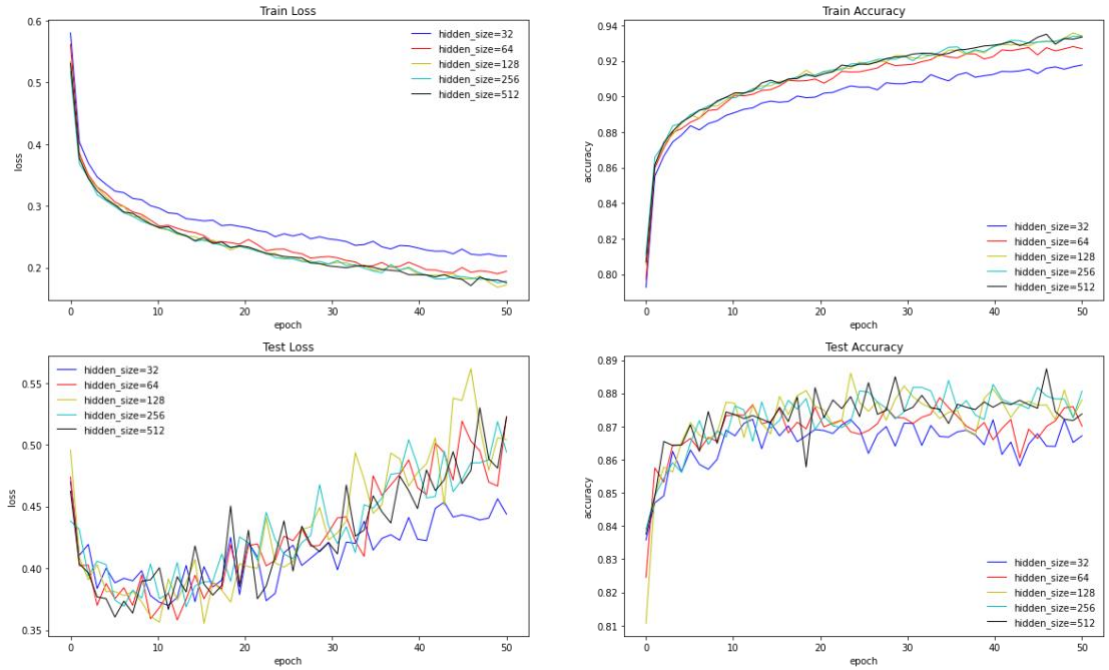
3.4 使用不同的隐藏层层数

分别设置隐藏层层数为 1、2、3。结果如下图，层数越多，训练时间越长，但效果提升并不显著，相反，层数达到一定数目后容易过拟合。综合考虑，隐藏层层数=1 最佳。



3.5 使用不同的隐藏单元个数

分别设置隐藏单元个数为 32、64、128、256、512。结果如下图，隐藏单元个数越多，训练时间越长，效果提升也较为明显，但当达到一定的个数，提升不显著，容易过拟合。综合考虑训练时长和效果，隐藏单元个数=128 最佳。

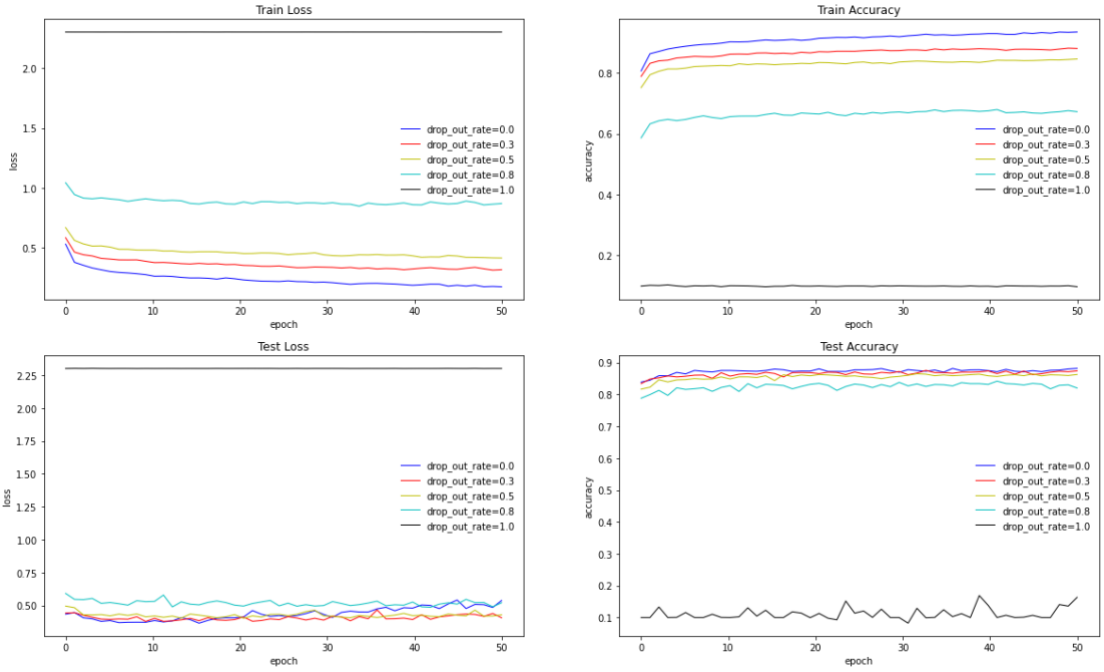


3.6 探究不同丢弃率对实验结果的影响

```
def dropout(x: torch.Tensor, dropout_prob=0.5):
    """
    丢弃率(失活率)
    设丢弃概率为p，那么有p的概率 Hidden_i 会被清零，
    有1-p的概率 Hidden_i 会除以 1-p 做拉伸。
    """
    x = x.float()

    # 检查丢弃率是否在 0-1 之间
    assert 0 <= dropout_prob <= 1
    keep_prob = 1 - dropout_prob
    if keep_prob == 0:
        return torch.zeros_like(x)

    # 生成 mask 矩阵(向量)
    mask = (torch.randn(x.shape) < keep_prob).float()

    # 按照 mask 对 x 进行变换
    return mask * x / keep_prob
```

```
def create_linear_layers(self) -> Tuple[nn.Module]:
    """
    创建层
    :return:
    """
    modules = [
        FlattenLayaer(),
        nn.Linear(self.input_size, self.hidden_size),
        self.activation(),
        nn.Dropout(self.drop_out_rate),
    ]
    for i in range(self.num_layers - 1):
        modules.append(nn.Linear(self.hidden_size, self.hidden_size))
        modules.append(self.activation())
        modules.append(nn.Dropout(self.drop_out_rate))
    modules.append(nn.Linear(self.hidden_size, self.output_size))
    return tuple(modules)
```

分别设置丢弃率为0.0、0.3、0.5、0.8、1.0。丢弃率偏大或偏小，都无法起到防止模型过拟合的效果。结果显示，丢弃率=0.3 最佳。

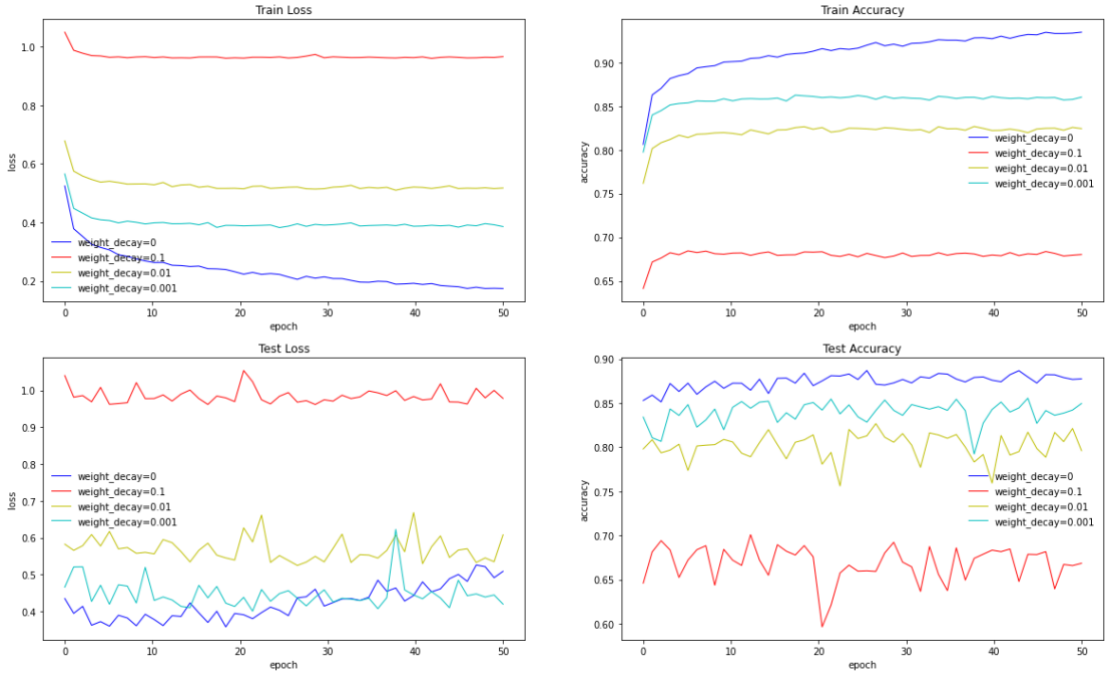


3.7 探究惩罚项的权重对实验结果的影响

```
def l2_penalty(w):
    return (w**2).sum() / 2
```

```
net = TorchNeuron(input_size, hidden_size, output_size)
print(net)
optimizer = torch.optim.Adam(net.parameters(), lr=lr, weight_decay=weight_decay)
```

利用 torch.optim 的 weight_decay 参数实现 L2 范数正则化。分别设置权重衰减系数为 0、0.1、0.01、0.001。结果显示，权重衰减系数=0.0 最佳。



4. 交叉验证(10 折)

```
indices = np.arange(0, X.shape[0])
kf = KFold(n_splits=10)

kf_train_loss, kf_test_loss = [], []
for train_index, test_index in kf.split(indices):
    train_index = torch.LongTensor(train_index)
    test_index = torch.LongTensor(test_index)

    train_x = torch.index_select(X, dim=0, index=train_index)
    train_y = torch.index_select(y, dim=0, index=train_index)
    test_x = torch.index_select(X, dim=0, index=test_index)
    test_y = torch.index_select(y, dim=0, index=test_index)

    train_set = Data.TensorDataset(train_x, train_y)
    test_set = Data.TensorDataset(test_x, test_y)

    train_data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True)
    test_data_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=True)

    # 训练
    train_loss_list, test_loss_list = train(train_data_loader, test_data_loader)
    kf_train_loss.append(train_loss_list)
    kf_test_loss.append(test_loss_list)
```

根据上述比对实验，按如下表格的值设置超参数。通过 10 折交叉验证，记录模型在测试集上的损失(LOSS)和 ACC(分类任务的准确率)，对每次验证中的所有迭代的损失值、准确率（epoch）分别取均值、最小或最大值列于表格中。

超参数	值	说明
activation	Relu	激活函数
num_layers	1	隐藏层层数
hidden_size	128	隐藏单元个数
drop_out_rate	0.3	丢弃率
weight_decay	0.0	权重衰减系数(惩罚项)

4.1 回归任务

LOSS	F1	F2	F3	F4	F5
最小	0.00028868	0.00031434	0.00031275	0.00030661	0.0003032
均值	0.00163858	0.00184026	0.00177395	0.00171711	0.00174201

LOSS	F6	F7	F8	F9	F10
最小	0.0003139	0.0003213	0.00025909	0.00034103	0.00032338
均值	0.00167926	0.00180045	0.00147343	0.00179616	0.00171343

4.2 二分类任务

LOSS	F1	F2	F3	F4	F5
最小	1.55238e-05	1.34269e-05	1.41567e-05	1.3864e-05	1.702e-05
均值	3.80892e-05	2.95931e-05	3.02551e-05	3.00e-05	3.879e-05

LOSS	F6	F7	F8	F9	F10
最小	1.67884e-05	1.78620e-05	1.52344e-05	1.2095e-05	1.448e-05
均值	3.66381e-05	3.89902e-05	3.42971e-05	2.55e-05	3.24e-05

ACC	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
最大	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
均值	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

4.3 多分类任务

对多分类任务进行交叉验证时，loss 和 accuracy 都不理想，很纳闷，可能是代码中存在 bug，困扰了很久，依然没有解决。

```
Average Loss    : [1.746, 1.481, 2.196, 1.904, 1.887, 1.366, 1.518, 1.851, 1.267, 1.732]
Average Accuracy: [0.34, 0.43, 0.168, 0.263, 0.267, 0.443, 0.412, 0.287, 0.484, 0.34]
```

五、实验结果

六、实验心得体会

通过本次实验，了解到了前馈神经网络的基本实现，以及常用超参数如失活率、惩罚项权重、激活函数等的使用。这让我对整个深度学习有了个初步的认识，通过不断地调整参数、尝试，使得模型的准确率逐提高，虽然过程比较煎熬，但成就感爆棚。

七、参考文献

八、附录

需要补充说明的内容，如无可略。

实验报告编写要求

1. 正文要求小四号宋体，行间距 1.5 倍；

2. 英文要求小四号 Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前 0.5 倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图 1、表 2、公式(5)等。