华中科技大学

计算机视觉实验报告

姓　　名：王逸

学　　院：计算机科学与技术

专　　业：计算机科学与技术

班　　级：CS2011

学　　号：U202014774

指导教师：刘康

分数	
教师签名	

2023 年　3 月　28　日

目　录

实验一 基于前馈神经网络的回归任务设计

1.1 实验要求

设计一个前馈神经网络，对一组数据实现回归任务。

在[-10, 10] x [-10, 10]的 2-D 平面内，以均匀分布随机生成 5000 个数据点(x, y).令 f(x, y) = x^2 + x*y + y^2.设计至少含有一层隐藏层的前馈神经网络以预测给定数据点(x, y)的函数值 f(x, y)。在随机生成的数据点中，随机抽取 90%用于训练，剩下的 10%用于测试。

深度学习框架任选，尝试不同的网络层数、不同的神经元个数、使用不同的激活函数等，观察网络性能。报告里需包含神经网络架构、每一轮训练后的模型训练及测试损失、实验分析。

1.2 实验内容

A. 数据集生成

调用 numpy.random.uniform 函数生成均匀分布的 5000 个数据点，然后根据生成点的坐标 x, y 计算出每个坐标的 f(x, y)值。然后调用 train_test_split 函数随机抽取 90%数据作为训练集，10%作为测试集。

之后以（(x，y)，f）的格式加入对应的 train_data 和 test_data 列表中，以便后续调用 dataloader 封装。

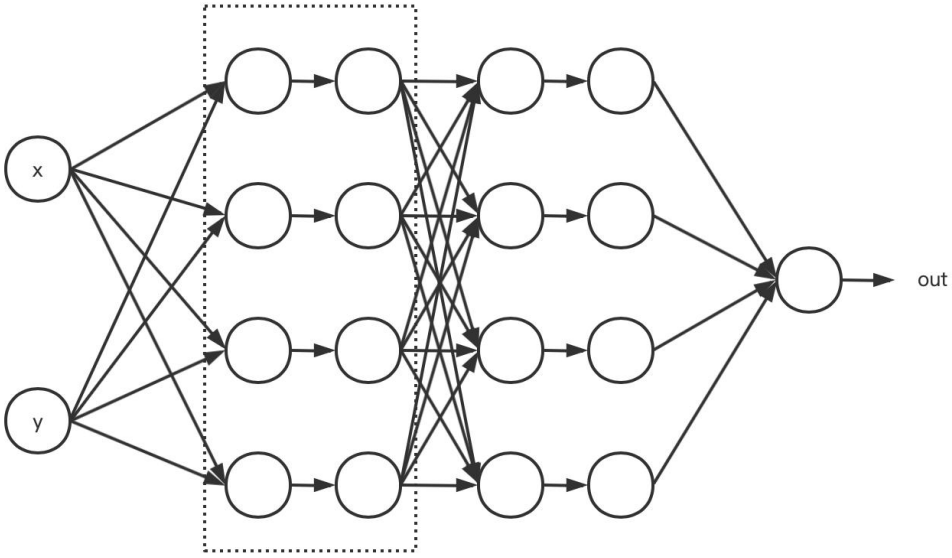
代码如下：

```python
import numpy as np
from sklearn.model_selection import train_test_split

def get_data(num, dim, test_num):
    np.random.seed(30)
    xy_set = np.random.uniform(-10, 10, (num, dim))
    f_xy = xy_set[:, 0] ** 2 + xy_set[:, 0] * xy_set[:, 1] + xy_set[:, 1] ** 2
    f_xy = f_xy.reshape(-1, 1)

    xy_train, xy_test, f_train, f_test = train_test_split(xy_set, f_xy, test_size=test_num)
    # print('train shape: {} {}'.format(xy_train.shape, f_train.shape))
    train_data = []
    test_data = []
    for idx in range(xy_train.shape[0]):
        train_data.append((xy_train[idx], f_train[idx]))
    for idx in range(xy_test.shape[0]):
        test_data.append((xy_test[idx], f_test[idx]))
    return train_data, test_data
```

B. 神经网络模型搭建

由于本任务是一个二元回归模型，输入神经元个数为 2，输出个数为 1。除此之外还需要定义中间隐藏层的层数、每层神经元个数以及该网络的激活函数。

采用全连接与激活函数结构, 其中隐藏层数为 2 的神经网络结构图如下。



隐藏层与激活函数

代码部分如下：

Net 初始化时传入的参数 in_feats，hidden_feats，out_feats 分别对应输入层、隐藏层与输出层的神经元个数。n_layers 为该网络隐藏层个数，activation 则为该网络使用的激活函数。

其中为了方便观察分析网络性能，hidden_feats、n_layers 和 activation 由命令行执行 main.py 时传入的参数 neurons、layers 以及 activation 确定。模型中提供了四种激活函数类型 sigmoid、relu、softplus 以及 tanh。

```python
class Net(nn.Module):
    def __init__(self, in_feats, hidden_feats, out_feats, n_layers, activation):
        super(Net, self).__init__()
        self.n_layers = n_layers + 2
        # print('network layer num is {}'.format(self.n_layers))
        self.layers = nn.ModuleList()
        # input layer
        self.layers.append(nn.Linear(in_feats, hidden_feats))
        # hidden layers
        for i in range(n_layers):
            self.layers.append(nn.Linear(hidden_feats, hidden_feats))
        #output layer
        self.layers.append(nn.Linear(hidden_feats, out_feats))
        # acivation function
        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'sigmoid':
            self.activation = nn.Sigmoid()
        elif activation == 'tanh':
            self.activation = nn.Tanh()
        elif activation == 'softplus':
            self.activation = nn.Softplus()

    def forward(self, x):
        out = x
        for i, layer in enumerate(self.layers):
            out = layer(out)
            # not output layer
            if i != self.n_layers - 1:
                # print(' {} layer is not output layer.'.format(i))
                out = self.activation(out)
        return out
```

C. 训练

本实验采用 pytorch 架构实现，对于数据采用 DataLoader 对输入数据进行处理，将函数参数 shuffle 设置为 True，使每次迭代完成下一次迭代读取数据时将数据顺序打乱。

```python
train_data, test_data = get_data(5000, 2, 0.1)
train_loader = DataLoader(train_data, batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size, shuffle=True)
```

随后定义初始化模型 model、损失函数（该实验中采用 MSE 损失函数）与优化器。

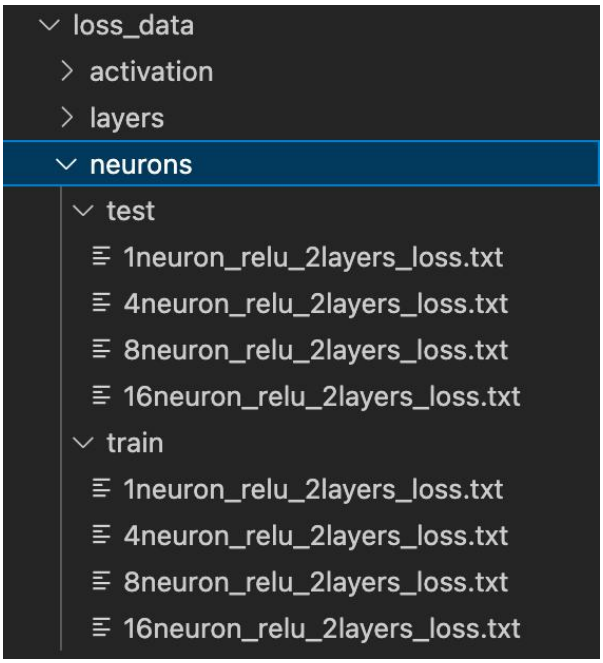
```python
model = Net(in_feat, hidden_feats=args.neurons, out_feats=out_feat,
            n_layers=args.layers, activation=args.activation)
loss_fcn = torch.nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

为了最后画出 loss 曲线，定义 train_loss 和 test_loss 列表，将每次 epoch

训练后得到的 train_loss 和 test_loss 加入到对应的列表中。训练全部完成后将两个列表分别写入对应的文件中。

```
train_loss = []
test_loss = []

for epc in range(args.epoch):
    losses = float(train(model, train_loader, optimizer, loss_fcn, epc))
    train_loss.append(losses)
    losses = float(test(model, test_loader, loss_fcn))
    test_loss.append(losses)
    print('**{} epoch, test loss is {}'.format(epc, losses))
with open("./loss_data/{}/train/{}neuron_{}_{}layers_loss.txt".format(args.variable, args.neurons, args.activation, args.layers), 'w') as train_los:
    train_los.write(str(train_loss))
with open("./loss_data/{}/test/{}neuron_{}_{}layers_loss.txt".format(args.variable, args.neurons, args.activation, args.layers), 'w') as test_los:
    test_los.write(str(test_loss))
```

loss 数据文件目录如下：



为了方便控制变量，在命令行调用时加入参数 variable—本次训练侧重哪个变量、activation—激活函数种类、layers—隐藏层数、neurons—隐藏层神经元个数以及 epoch—迭代次数。默认值为('activation', 'relu', '2', '4', '100')

调用事例为：python main.py --variable layers --layers 30 --neurons 16

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Network')
    parser.add_argument("--variable", type=str, default='activation',
                        help="controlled varieble. such as activation, neurons, layers...")
    parser.add_argument("--activation", type=str, default='relu',
                        help="activation func. such as relu, sigmid...")
    parser.add_argument("--neurons", type=int, default=4,
                        help="number of hidden neurons. such as 1, 2, 4...")
    parser.add_argument("--layers", type=int, default=2,
                        help="number of hidden layers. such as 1, 2, 4...")
    parser.add_argument("--epoch", type=int, default=100,
                        help="number of epochs. such as 1, 2, 4...")
    args = parser.parse_args()
```

D. 画图分析

调用 matplotlib.pyplot 画出 loss 曲线。命令行调用时有两个传入参数：

- Variable：针对该参数画出 loss 曲线，如 activation、layers、neurons。

- Type：画出 train loss 曲线或者 test loss 曲线，如 train、test。

默认值为‘activation’与‘test’。

根据之前设置的 loss data 文件目录结构，通过 args.variable 与 args.type 确定需要遍历读取数据的文件夹。然后调用 os.listdir 获得该文件夹下的数据文件名，通过 os.path.join 获得文件路径，读取数据。最后调用 plot 函数画出曲线。

1.3 实验结果分析

a. 不同的激活函数对网络性能的影响

该部分通过选择四个不同的激活函数，对于给定的神经元个数与隐藏层层数进行训练，得到四组 train_loss，test_loss，并在同一个曲线图中画出。

编号	隐藏层数	神经元个数	激活函数
1	2	4	Tanh
			ReLU
			softplus
			sigmoid
2	1	8	Tanh
			ReLU
			softplus
			sigmoid

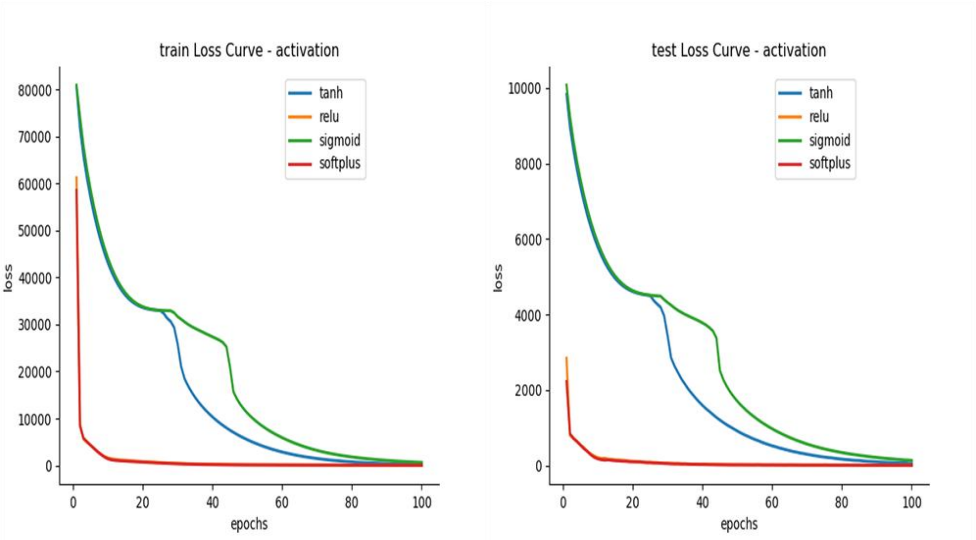


图 1 layer=1, neurons=8

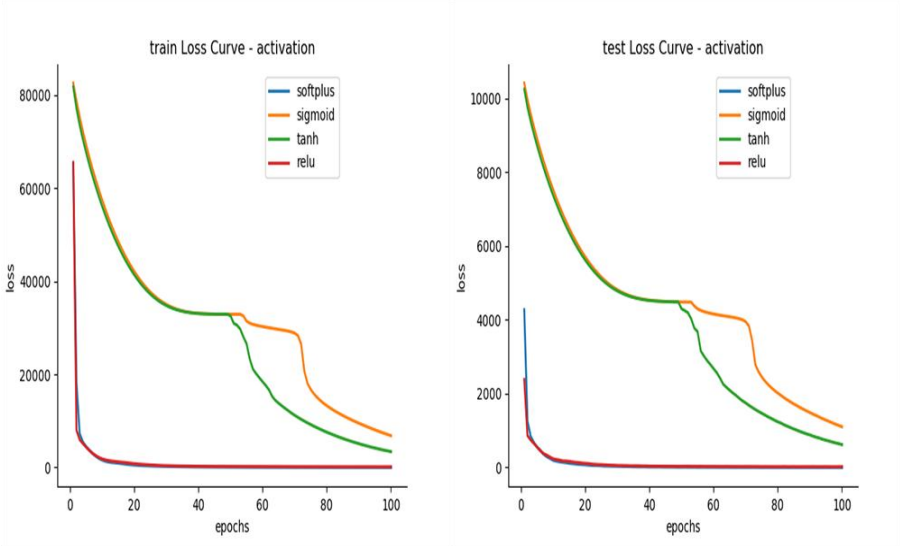


图 2 layer=2, neuron=4

通过以上两组实验，可以看出：

- 对于简单结构的神经网络，relu 与 softplus 的可以使该网络的损失值快速收敛，并达到一个较低的值如 10 或 5；
- 而 tanh 与 sigmoid 则表现较差, 且在第 40-50 次迭代训练中逐渐趋于一个值，在第 60 次之后又开始下降。
- 而在后两个激活函数中 tanh 效果比 sigmoid 更好。
- 而对于 layer=1，neuron=8 的神经网络，它的训练误差比 layer=2，neuron=4 的网络小很多，且在 100 次训练结束时四个激活函数基本可以得到相近的较低的损失值。

对于以上结果，可能原因有：

- softplus 可以看作是 ReLU 的平滑，而使用 Relu 的 SGD 算法收敛速度比 sigmoid 和 tanh 快，在大于 0 的区域上并不会出现梯度饱和或者梯度弥散的问题，而且它的计算更加高效；
- tanh 和 sigmoid 是全部激活，会使神经网络较重；
- sigmoid 函数的敏感区间短，为[-1, 1]，超过区间时就处于了饱和状态，梯度较小，导致模型参数更新缓慢；
- 而 tanh 变化敏感区间较宽，相比 sigmoid 延迟了饱和期

b. 不同的隐藏层数对网络性能的影响

控制神经网络的激活函数和隐藏层数不变，改变神经元个数。

编号	激活函数	隐藏层数	神经元个数
	ReLU	4	1
			4
			16

1			64
2	tanh	8	Tanh
			ReLU
			softplus
			sigmoid

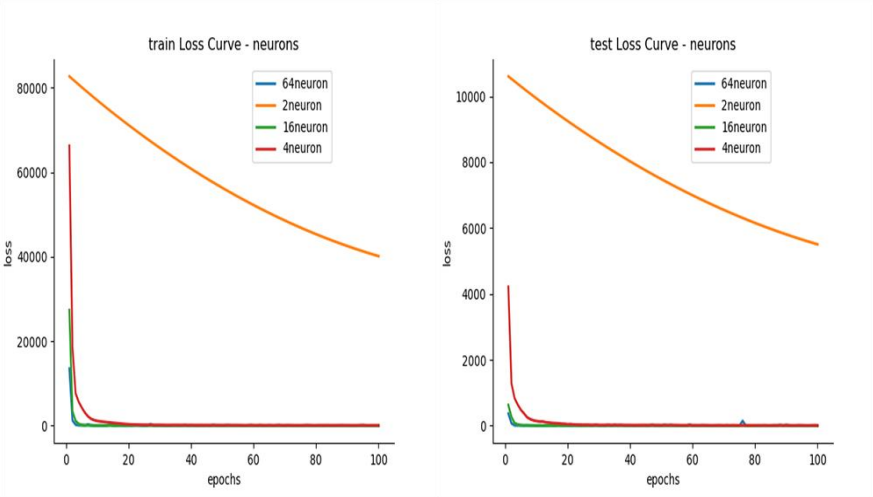


图 3 ReLU，layer=4

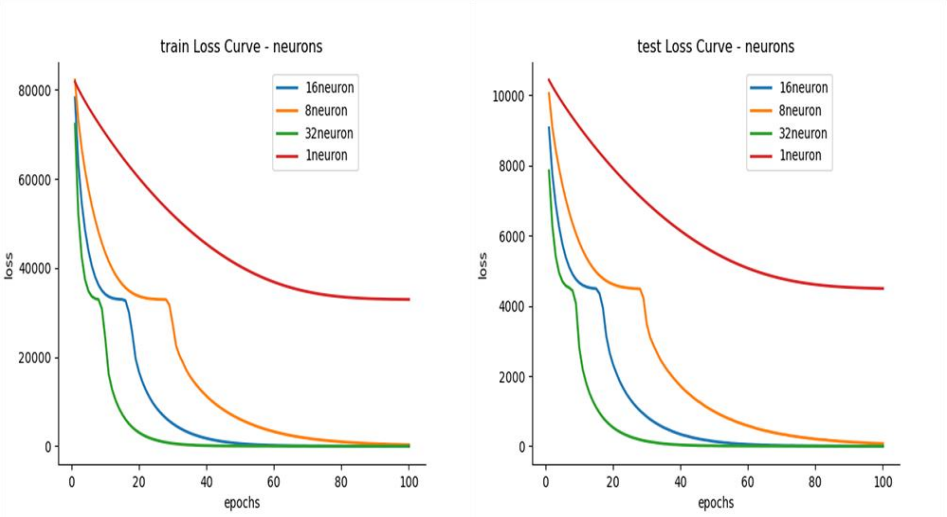


图 4 Tanh，layer=1

通过以上图像可以看出：

- 在模型损失还未收敛时，增加神经元个数可以加快模型的收敛速度；
- 在隐藏层中使用太少神经元（如 neuron=1）将导致模型的欠拟合，并且收敛速度很慢
- 而过多的神经元则会导致模型的过拟合，在图三的 test loss 曲线图中可以看到 neuron=16/64 时会有轻微振荡现象，神经元越多，振荡现象越明显，且运行时间明显变长。

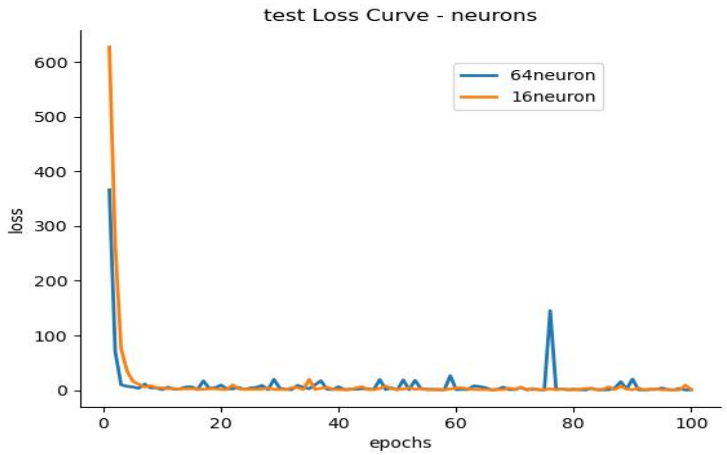


图 5 neuron=16、64 的振荡现象

c. 不同的神经元个数对网络性能的影响

控制神经网络的激活函数和神经元个数不变，改变隐藏层数。

编号	激活函数	神经元个数	隐藏层数
1	ReLU	4	1
			4
			16
			30
2	tanh	8	1
			2
			4
			8

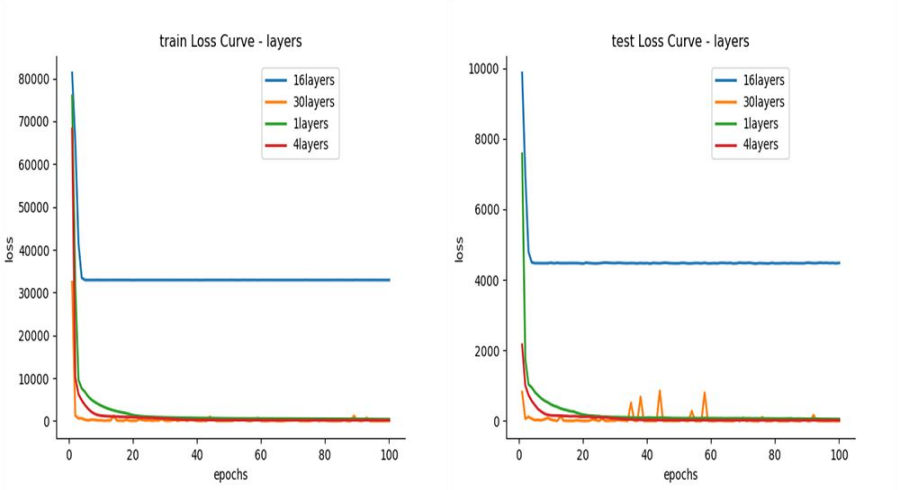


图 6 ReLU，neuron=4

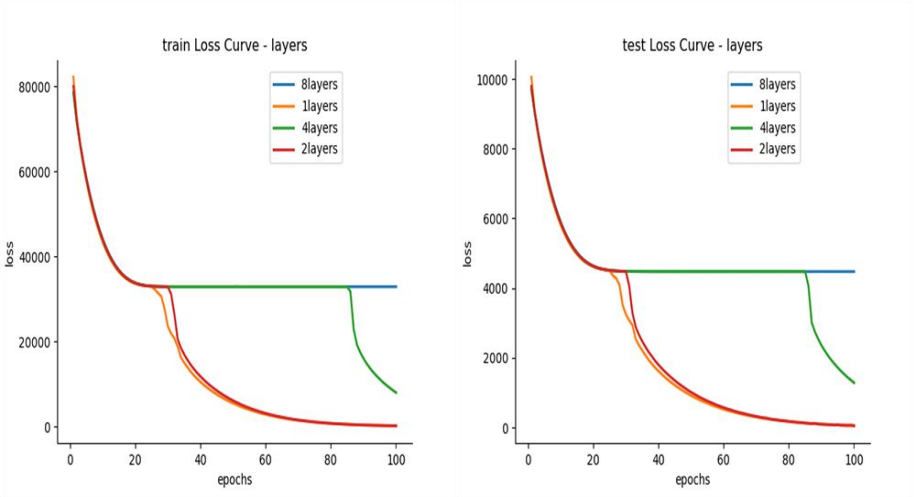


图 7 Tanh，neuron=8

通过以上结果可以看出：

- 对于 ReLU 等简单的激活函数，可以看出通过适当增加隐藏层层数可以加快模型的收敛速度；但当加到 16 层时会发现仍然出现了梯度消失的情况；而对于较为复杂的网络结构如 neuron=8，layer=30 的橙色线，我们发现随着迭代次数的增加，出现了震荡。
- 而对于 tanh 函数，会发现随着隐藏层数量的增加，损失加大，甚至出现了梯度消失的情况。这是因为由于层数过多，后面的数据处于饱和状态，较多的 f'(x)*wi 相乘，且|w|<1，导致梯度成指数型衰减趋近于零，出现了梯度消失的状况。

1.4 实验小结

本次实验过程中除了以上三个变量之外，还需要手动选择 batch_size 和 learning_rate 的大小，在调试中发现过大或者过小的值均会导致较大的损失。一般情况下，随着隐藏层数和神经元个数的增加，会导致模型收敛速度加快；但对于某些激活函数（如 sigmoid 和 tanh），隐藏层数增加反而会导致梯度消失，降低网络性能。