华中科技大学

计算机视觉实验一报告

姓　　名：

专　　业：

班　　级：

学　　号：

指导教师：

分数	
教师签名	

年月日

目录

实验一：基于前馈神经网络的回归任务设计

1.1 任务要求

设计一个前馈神经网络，对一组数据实现回归任务。

在$[-10,10] \times [-10,10]$的 2-D 平面内，以均匀分布随机生成 5000 个数据点(x,y). 令$f(x,y) = x^2 + xy + y^2$. 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$f(x,y)$函数值。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。

1.2 实验内容

整个实验使用 jupyter notebook 和 PyTorch 框架完成，实验过程如下：

1、生成训练和测试数据集

先使用 numpy.random.uniform 函数在$[-10,10] \times [-10,10]$的 2-D 平面内以均匀分布随机生成 5000 个数据点，并按任务要求中的函数式计算各点对应的函数值。然后按 9:1 的比例随机将数据集分割成训练集和测试集。代码和运行结果如下图所示：

```
np.random.seed(0)
x = np.random.uniform(-10, 10, (5000, 2))
x1 = x[:, 0]
x2 = x[:, 1]
y = x1**2 + x1*x2 + x2**2

# get random index of x_train and x_test
np.random.seed(1)
index = np.random.choice(np.arange(0,5000), size=5000, replace=False)
index_train = index[:int(0.9*5000)]
index_val = index[int(0.9*5000):]

# split the dataset into training dataset and test dataset
x_train = x[index_train, :]
y_train = y[index_train]
x_val = x[index_val, :]
y_val = y[index_val]
print("x_train:", x_train.shape)
print("y_train:", y_train.shape)
print("x_val:", x_val.shape)
print("y_val:", y_val.shape)
```

```
x_train: (4500, 2)
y_train: (4500,)
x_val: (500, 2)
y_val: (500,)
```

图 1-1：生成训练和测试数据集

由上图可知：训练集数据点维度为(4500, 2)，对应函数值维度为(4500,)；测试集数据点维度为(500, 2)，对应函数值维度为(500,)。

2、加载数据集

利用 PyTorch 的 API 可以很方便地加载数据集，并在训练过程中使用。先使用 torch.utils.data. TensorDataset 函数将上一步生成的 numpy.ndarray 类型的数据转换为 torch.tensor 并封装成 torch.utils.data.Dataset 类，然后使用 torch.utils.data.DataLoader 加载数据集，batch_size 设为 16。代码如下图所示。

```
dst_train = TensorDataset(torch.from_numpy(x_train), torch.from_numpy(y_train))
dst_val = TensorDataset(torch.from_numpy(x_val), torch.from_numpy(y_val))
bs = 16 # batch size
loader_train = DataLoader(dst_train, batch_size=bs, shuffle=True)
loader_val = DataLoader(dst_val, batch_size=bs, shuffle=True)
```

图 1-2：加载数据集

3、定义训练函数：

定义训练函数，以需要训练的模型 model、损失函数 criterion、优化器 optimizer 等作为参数，实现对模型的训练过程，每训练一个 epoch 后，在测试集上测试，打印每个 epoch 的模型训练及测试损失值，并用 2 个 list 记录每个 epoch 的模型训练及测试损失值，将其作为函数返回值，以便可视化。训练结束后打印训练时间。代码较长，详见压缩包中的代码文件。

4、自定义模型

自定义类 fcn，继承 torch.nn.Module 类，并重新实现构造函数__init__和前向传播函数 forward。其中，构造函数有 2 个参数，分别为 hidden_layers 和 activation_function，hidden_layers 为一个列表，表示各个隐藏层的神经元个数，activation_function 则是模型使用的激活函数，这 2 个参数都是由用户自己定义的。这样是为了后续实验中方便创建不同网络层数、不同神经元个数和不同激活函数的模型实例。代码如下图所示：

```python
class fcn(nn.Module):
    def __init__(self, hidden_layers, activation_function):
        depth = len(hidden_layers)
        assert depth>=1  # at least one hidden layer

        super().__init__()
        hidden_layers.insert(0, 2)  # 2-D input
        # hidden_layers.append(1)  # 1-D output

        # build hidden layers according to the neuron number in the hidden_layers list
        self.layers = nn.ModuleList([])
        for n in range(depth):
            self.layers.append(nn.Linear(hidden_layers[n], hidden_layers[n+1]))

        self.out = nn.Linear(hidden_layers[-1], 1)
        self.activate = activation_function

    def forward(self, x):
        for layer in self.layers:
            x = self.activate(layer(x))

        return self.out(x)
```

图 1-3：自定义模型

5、定义模型、损失函数、优化器和学习率调整策略

利用上一步定义的类 fcn 创建模型实例，模型配置为 1 个隐藏层（256 个神经元）和 ReLU 激活函数。使用 SmoothL1Loss 作为损失函数，更稳定，避免梯度爆炸。使用 Adam 优化器，初始学习率为 1e-3。学习率调整策略为自适应调整 ReduceLROnPlateau，当损失值不再下降时，将学习率乘以 0.1。

6、训练模型

调用第 3 步中定义的训练函数 train，并将第 2 步中的 dataloader 和第 5 步中定义的的模型、损失函数等作为参数传入，设置训练 100 个 epoch，即可开始训练。

训练过程中的输出详见代码文件，最终训练损失值为 0.0049， 测试损失值为 0.0052，训练时长为 1 分 53 秒。

7、可视化训练过程损失值变化

利用第 6 步训练过程返回的每个 epoch 的模型训练和测试损失值，在同一个坐标图中画出 2 条损失值变化曲线。代码和运行结果如下图所示：

```python
plt.figure(figsize=(10,6))
plt.title('Loss')
plt.xlabel('Epoch')
plt.plot(train_loss_history[25:], '-', label='train')
plt.plot(val_loss_history[25:], '-', label='val')
plt.legend()
plt.show()
```

Python

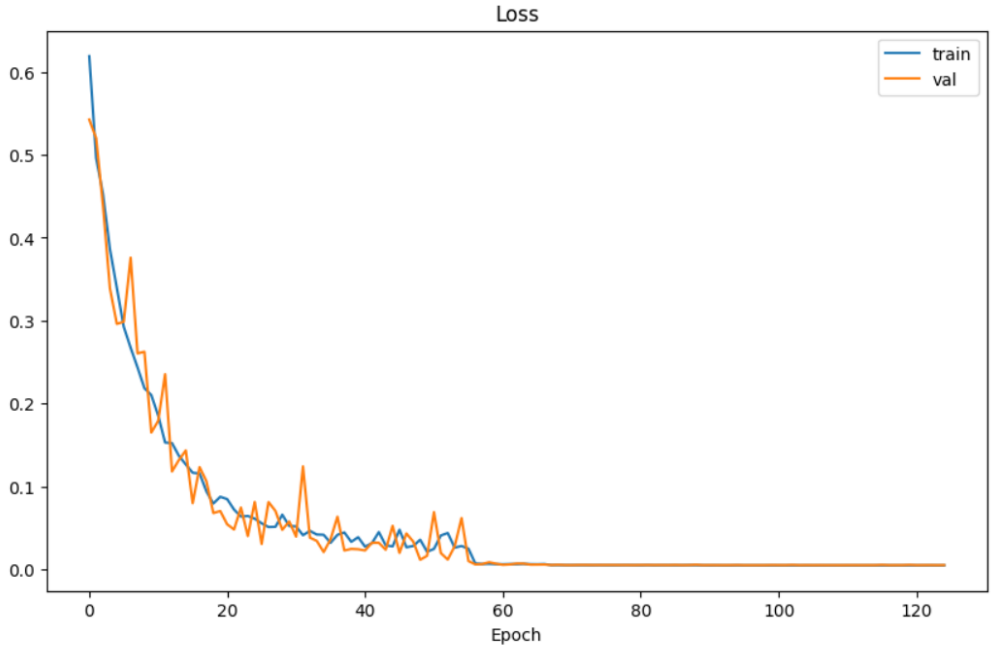

图 1-4： 可视化损失值变化

注意，画图时只截取了第 25 个 epoch 之后的损失值，是为了使损失值的波动，以及训练损失值和测试损失值之间的差异更加明显。

1.3 不同的网络层数

为了探究不同的网络层数对模型性能的影响，我设置了 5 个模型，网络层数分别为 2、4、8、16、32，每个模型的每层神经元个数都一样，为 16 个，除此之外的无关变量，如激活函数、损失函数、优化器等，都一样。分别训练这 5 个模型，记录它们的测试损失值，并可视化。

为了节省时间，所有模型只训练了 10 个 epoch，训练时长分别为 8 秒、11 秒、16 秒、26 秒、50 秒。可视化结果如下图所示：

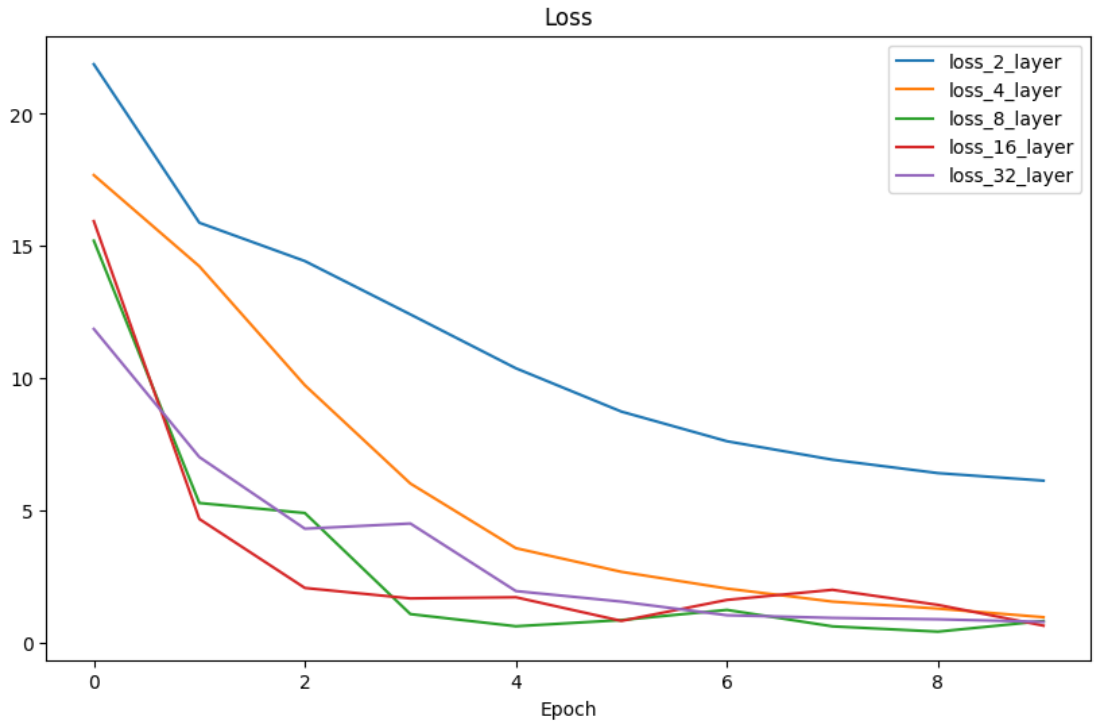

图 1-5：不同网络层数的测试损失值变化

分析发现：

1、网络层数越多，神经元个数就越多，训练时长越长，并且模型越复杂，收敛越慢；

2、网络层数越多，模型的非线性表达能力越好，拟合复杂特征输入的能力越强，因此损失值越低；

3、网络层数也不是越多越好，模型的性能随着层数的增加渐趋饱和，我们看到 4、8、16、32 层的模型的最终损失值差不多。

1.4 不同的神经元个数

为了探究不同神经元个数对模型性能的影响，我设置了 4 个只有 1 个隐藏层的模型，分别有 8、16、32、64 个神经元。除此之外的无关变量，如激活函数、损失函数、优化器等，都一样。分别训练这 4 个模型，记录它们的测试损失值，并可视化。

为了节省时间，所有模型只训练了 10 个 epoch，训练时长分别为 11 秒、12 秒、14 秒、10 秒。可视化结果如下图所示：

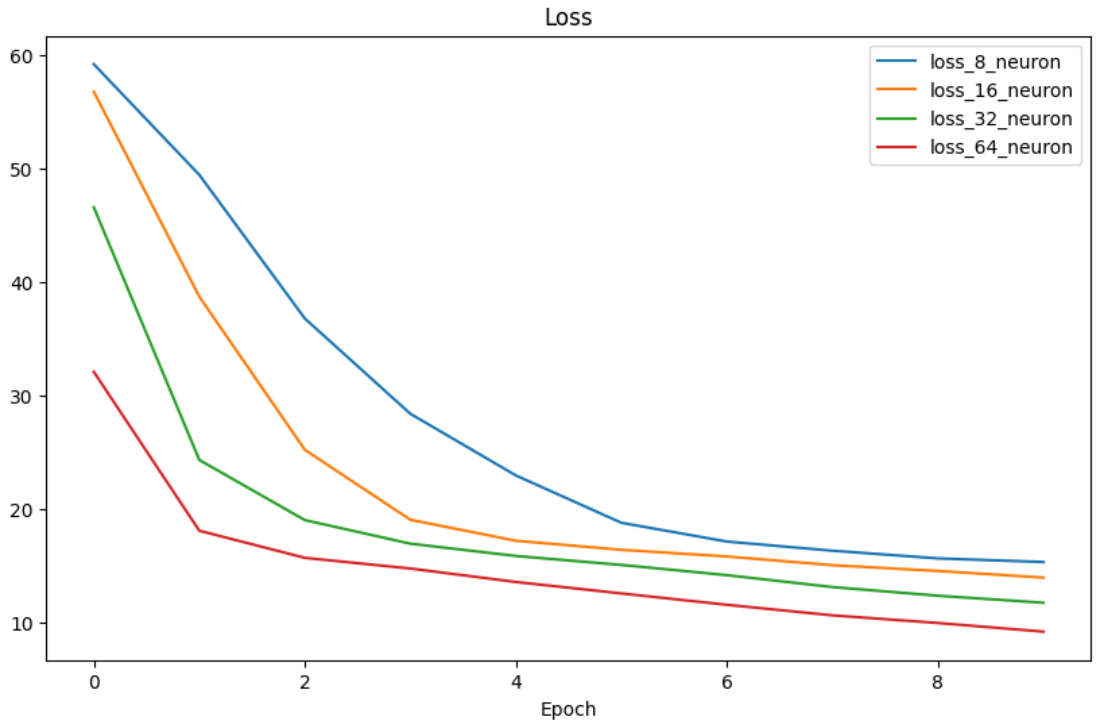

图 1-6：不同神经元个数的测试损失值变化

分析发现：

1、在网络层数不变的情况下，增加神经元个数并不会明显增加训练时长。因为每一个隐藏层都是全连接层，每一层都是矩阵计算，也就是说每一层的所有神经元都是并行计算；

2、神经元个数越多，模型表达能力、拟合能力越强，损失值越低；

3、与增加网络层数相比，单纯地增加隐藏层神经元个数对模型性能的提升较少，因为一个隐藏层不论有多少神经元，它仍然是线性变换，而增加网络层数可以增加非线性变换。

1.5 不同的激活函数

为了探究不同激活函数对模型性能的影响，我设置了 4 个只有 1 个隐藏层（32 个神经元）的模型，分别使用 Sigmoid、Tanh、ReLU、LeakyReLU 作为激活函数。除此之外的无关变量，如损失函数、优化器等，都一样。分别训练这 4 个模型，记录它们的测试损失值，并可视化。

为了节省时间，所有模型只训练了 10 个 epoch，训练时长分别为 10 秒、9 秒、9 秒、9 秒。可视化结果如下图所示：

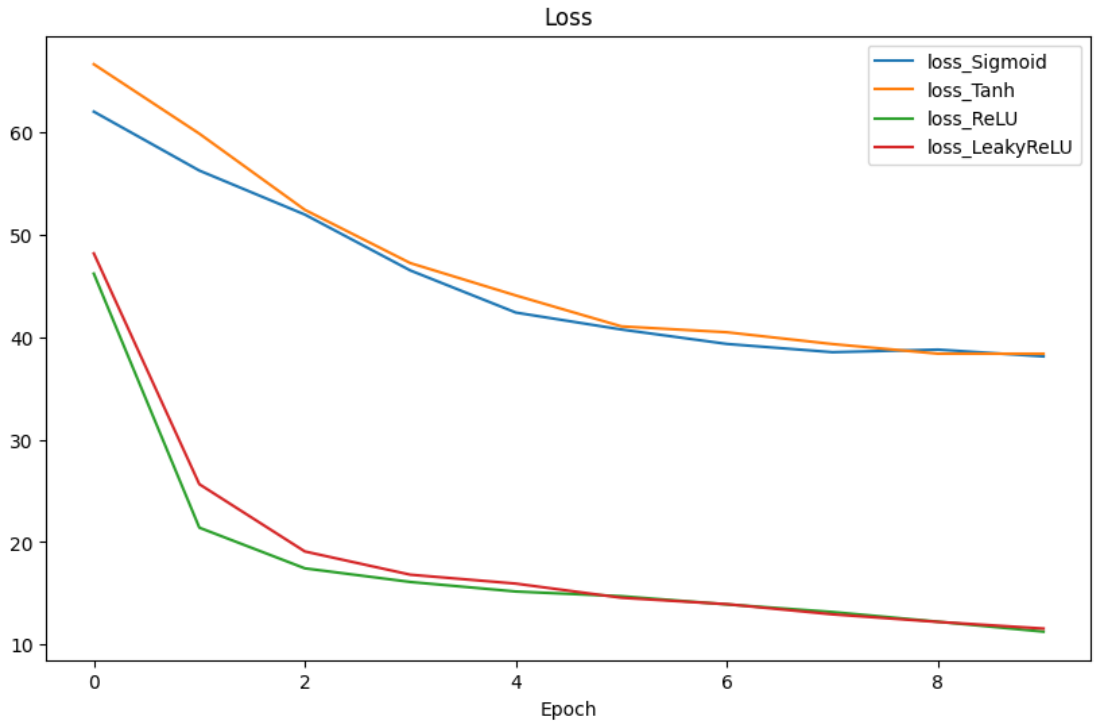

图 1-7：使用不同激活函数的损失值变化

分析发现：

1、激活函数对模型训练时长没有什么影响；

2、使用 sigmoid 和 tanh 激活函数的模型损失值相近，使用 relu 和 leakyrelu 激活函数的模型损失值相近，而使用 sigmoid 和 tanh 激活函数的模型损失值远高于使用 relu 和 leakyrelu 激活函数的模型损失值。这可能是因为 sigmoid 和 tanh 激活函数都存在梯度消失的问题。

1.6 实验心得

通过此次实验，我学习了神经网络中的神经元、网络结构以及前馈神经网络，了解了反向传播算法、优化问题，探究了不同网络层数、不同神经元个数、不同激活函数对模型性能的影响等等。