前馈神经网络实验报告

姓名：陈泽豪　学号：SA22001009

2022 年 10 月 28 日

摘要

本报告首先介绍一下有关前馈神经网络的大体框架，然后给出简要的实验过程以及对超参数的实验分析，模型在测试集上的测试结果等。

一、前馈神经网络介绍

神经网络是具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的反应。以图一中的神经网络为例我们来介绍有关神经元以及前馈神经网络的一些信息。隐藏层 1 中的每一个节点都可以称之为一个神经元，这些神经元接受来自上一层神经元的输入信号，而每个输入信号对此神经元的影响也有对应的权重，当前一层的输入信号与权重相乘后达到了此神经元的阈值，则可以激活这个神经元得以向下继续传播。这就构成了整个神经网络的过程。

本次实验主要注重于前馈神经网络，前馈神经网络是一个全连接的神经网络，也被称为一个多层感知机，各层的神经元之间无连接，而靠近的层之间的神经元两两相连接，在整个神经网络中没有反馈信号，只有从输入层向输出层的单向传播。如果需要用数学表达式来表示整个过程，那么以下面的图 1 为例，便存在一个 $\mathbf{R}^{4\times3}$ 的矩阵 $W^{(1)}$，第一个隐藏层上的激活函数 $g^{(1)}$，一个 $\mathbf{R}^{4\times4}$ 的矩阵 $W^{(2)}$，以及在第二个隐藏层上的激活函数 $g^{(2)}$，一个 $\mathbf{R}^{1\times4}$ 的矩阵 $W^{(3)}$，于是可以将整个过程用以下的函数表示：

$$f = W^{(3)}(g^{(2)}(W^{2}(g^{(1)}(W^{(1)}x + b^{(1)})) + b^{(2)})) + b^{(3)} \tag{1}$$

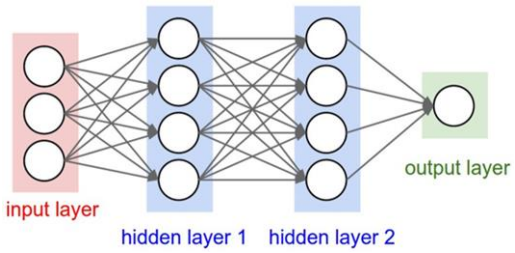


图 1: 前馈神经网络

整个前馈神经网络在进行实验的时候一般符合以下的运行步骤：

1. 选定数据集，可以将数据集划分为训练集，验证集，测试集，三个数据集互不相交。其中整个模型的训练在训练集上完成，模型的超参数调整在验证集上完成，模型最终的测试在测试集上完成。
2. 搭建模型，选定整个网络的深度，宽度，网络每一层采用的激活函数等。同时定义整个网络采用的学习准则 (例如本次实验采用的为均方误差)，以及对整个网络做参数调整的优化器。
3. 训练整个网络，将搭建好的网络进行前向的 loss 计算以及反向的梯度计算，完成参数的更新。将训练好的模型在验证集上进行测试，从而通过调整超参数来观察模型在验证集上的准确率，选择最好的超参数对应的模型在测试集上做测试得到最终的结果。

二、实验配置

本次实验通过 Anaconda 搭建出了虚拟环境，整个代码在 python 3.9，cuda 11.3.1，numpy 1.23.1，pytorch 1.12.1 环境下进行实验，GPU 使用的是 NVIDIA Geforce RTX 3060。

三、实验过程

首先来介绍一下整个代码的思路以及具体做的事情，首先定义了一个 target_func，它返回的就是我们这次需要拟合出来的函数 $sin(x) + exp(-x)$，然后调用了 torch.device 函数来决定我们运行的硬件为 CPU 还是 GPU。下一步通过改变 flag 值来决定我们是要训练网络做超参数调整还是用已经决定好超参数的网络去做测试。再之后就需要准备数据集，这里我采用了在定义区间 $[0, 4\pi]$ 做均匀采样后做 shuffle 并分配数据，具体代码如下所示：

```
# 首先产生[0,4pi]中的数据点
x = np.linspace(0, 4*np.pi, 1000)
if validate_flag:
np.random.seed(0)
np.random.shuffle(x)
y = target_func(x)

# 将x, y变成列向量的形式
change_x = np.expand_dims(x, axis=1)
change_y = np.expand_dims(y, axis=1)

# 使用TensorDataset将x,y做打包;分成三个部分作为训练集, 验证集与测试集
dataset_train = TensorDataset(torch.tensor(change_x[0:600, ...], dtype=torch.float),
torch.tensor(change_y[0:600, ...], dtype=torch.float))
dataset_validate = TensorDataset(torch.tensor(change_x[600:800, ...], dtype=torch.float),
torch.tensor(change_y[600:800, ...], dtype=torch.float))
dataset_test = TensorDataset(torch.tensor(change_x[800:1000, ...], dtype=torch.float),
torch.tensor(change_y[800:1000, ...], dtype=torch.float))

# 将三个dataset做一次batch打包
dataloader_train = DataLoader(dataset_train, batch_size=100, shuffle=True)
dataloader_validate = DataLoader(dataset_validate, batch_size=100, shuffle=True)
dataloader_test = DataLoader(dataset_test, batch_size=100, shuffle=True)
```

在分配完数据之后建立了一个训练网络的类 MyNet 方便进行训练，在类中定义了初始化函数以及前向传播过程。然后就是定义优化器以及学习准则对网络进行训练，具体的代码如下所示：

```
# 设置随机数种子, 为了复现实验结果, 同时方便验证集上对比试验
if validate_flag:
seed = 0
```

```
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
fnn_net = MyNet(hidden_layer_num, hidden_layer_width).to(device)

learning_rate = 0.001
epoch_num = 1500

# 定义优化器和损失函数
optimizer = torch.optim.Adam(MyNet.parameters(fnn_net), lr=learning_rate)
loss_func = nn.MSELoss()

# 训练次数由自己决定为epoch_num
for epoch in range(epoch_num):
loss = None
for batch_x, batch_y in dataloader_train:
batch_x = batch_x.to(device)
batch_y = batch_y.to(device)
predict_y = fnn_net(batch_x)
loss = loss_func(predict_y, batch_y)
optimizer.zero_grad()
loss.backward()
optimizer.step()
# 每100次的时候打印一次日志
if (epoch+1) % 100 == 0:
print("step: {0} , loss: {1}".format(epoch+1, loss.item()))
```

训练完模型之后剩下的事情就是做模型在验证集上的测试以方便我们去做超参数调整，之后如果调整完超参数后就可以对测试集做相应的测试得到误差结果。本代码在最后又额外生成了一系列与原数据集几乎不交的点集去做模型的测试并算出了对应的 MSE 结果，用一张图形象的表现出了预测结果与正确结果之间的差距。如果仅仅是训练之后在验证集上进行测试，那么只需将 validate_flag 设置为 1，其他两个测试集的 flag 设置为 0 进行超参数调整；如果需要进行模型测试，那么就将 test_flag 与 new_dataset_test_flag 设置为 1，validate_flag 设置为 0 点击运行即可。

四、超参数调整

本文在做超参数调整时固定了产生数据集的 shuffle，使得每次重复实验的训练集和验证集都是同一个以保证实验可对照，同时设置了 pytorch 的随机数种子以保证每次在网络上随机生成的初始参数不变，下面用几张表来说明我们对超参数的一些考虑与选择，但是由于不同参数选择也会因为数据集选择的改变以及网络参数的不同的初始化发生一些不可测的变化，我们只能简要对这些参数做一些思考。

4.1 对网络深度的调整

首先我们选取了学习率为 0.01，训练的 epoch 为 100 次时，我们设置隐藏层宽度为 100，在添加隐藏层时加入的均为宽度为 100 的隐藏层，激活函数选用 relu，得到了深度对验证集的 MSE 影响如下：

网络深度	MSE	网络深度	MSE
1	0.2684	2	0.0273
3	0.0270	4	0.0257
5	0.00264	6	0.149
7	0.0174	8	0.0234

事实上继续增加深度确实可能会减小误差，但也有可能增加，非常不好判断具体的选取，任何参数都得基于别的参数做变化。

4.2 对网络隐藏层宽度的调整

选择深度为 5，其他参数均保持不变来对网络做调整, 得到结果如下：

网络宽度	MSE	网络宽度	MSE
20	0.0025	40	0.0020
60	0.0339	80	0.0071
100	0.0026	120	0.0099

从这个例子里可以看到网络宽度对网络的影响也是需要考虑的。

4.3 对网络学习率的调整

如果选择上述的网络宽度为 40 继续实验，考虑学习率的影响：

学习率	MSE	学习率	MSE
0.002	0.0237	0.004	0.0093
0.006	0.0095	0.008	0.0076
0.01	0.0020	0.012	0.0016

4.4 考虑激活函数的影响

上面只是讨论了一些参数的影响，但是并未统筹考虑，经过我自己反复实验，得到了应该选取网络深度为 4，网络宽度为 65，网络学习率为 0.001，网络训练次数 1000 次时，在验证集上的结果为 6.58×10^{-5} 下面考虑在这种条件下的不同激活函数影响：

激活函数	MSE	激活函数	MSE
relu	6.58e-05	sigmoid	1.18e-05
tanh	0.0007	leaky relu	6.99e-05
elu	3.47e-05	softplus	0.0026

因此选择除了 tanh 或者 softplus 函数以外的函数都是比较符合我们预期的。

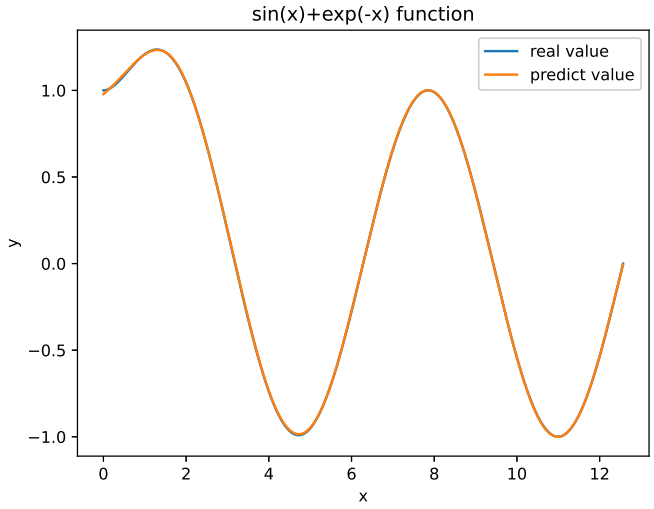
五、测试性能

下面我们对测试集做测试，选择 sigmoid 函数做实验。这里我选择了两个测试集，其中一个测试集是由 test_flag 作为是否要测试的开关，另一个测试集是在 $[0, 4\pi]$ 上生成的与之前训练集没有交集的新的测试集，由 new_dataset_test_flag 进行开关调控。首先给出一组 sigmoid 函数的实验结果数据：

表 1: sigmoid 函数结果

实验次数	测试集 1_MSE	测试集 2_MSE
1	1.54E-05	1.21E-05
2	0.00015	0.00016
3	3.01E-05	4.13E-05
4	4.94E-05	3.93E-05

可以发现的是当我们随机生成网络参数或者随机生成训练集数据时，训练出来的模型不一定会在 1000 个 epoch 内完成梯度的下降，这时我们可能需要增加 epoch 的数目来使模型得到充分的训练 (代码中已经改为了 1500 个 epoch)。下面再给出训练的结果与真实值之间的比较图像：



可以发现训练的可以非常贴合，这张图上的测试结果只在 0 附近有点差别。

六、实验结论

通过本次实验我们了解了怎么使用 pytorch 以及利用 pytorch 搭建一个前馈神经网络对简单函数做拟合，并通过调参等方式了解了如何改变输入会影响整个模型的性能。