深度学习 作业一

- 姓名：吕奇澹
- 学号：SA23006171

实验过程

数据生成

1. 定义MyDataset数据类用于数据生成，并将其划分为**互不相交**的训练集、验证集和测试集。

数据生成部分，默认使用参数为：sample_num=3000, 表示在 $[0, 2\pi)$ 区间随机采样**3000**个样本点作为横坐标。

核心代码如下:

MyDataset类

```python
class MyDataset(Dataset):
    data_record = dict()
    def func(x):
        return torch.sin(x)
    def __init__(self, num_samples, train_ratio, val_ratio, test_ratio, type):
        assert type in ['train', 'val', 'test'], "type must be 'train', 'val' or 'test'"
        train_sample = int(num_samples * train_ratio)
        val_sample = int(num_samples * val_ratio)
        test_sample = num_samples - train_sample - val_sample
        if (train_sample, val_sample, test_sample) in MyDataset.data_record:
            data = MyDataset.data_record[(train_sample, val_sample, test_sample)]
        else:
            if not os.path.exists('data'):
                os.makedirs('data')
            file_name = f'data/{train_sample}_{val_sample}_{test_sample}.pt'
            if os.path.exists(file_name):
                data = torch.load(file_name)
            else:
                data = torch.rand(num_samples, 1, device=device) * 2 * math.pi
                torch.save(data, file_name)
            MyDataset.data_record[(train_sample, val_sample, test_sample)] = data
        if type == 'train':
            self.data = data[:train_sample]
        elif type == 'val':
            self.data = data[train_sample:train_sample+val_sample]
        else:
            self.data = data[train_sample+val_sample:]
        self.label = MyDataset.func(self.data)
    def __getitem__(self, index):
        return (self.data[index], self.label[index])
    def __len__(self):
        return len(self.data)
```

模型搭建

对于拟合函数 $y = sin(x), x \in [0, 2\pi)$ ，我们使用多层MLP进行拟合，具体网络层代码如下：

网络层代码

```python
class Net(torch.nn.Module):
    def __init__(self, width, depth, activation, dropout=0.0):
        super(Net, self).__init__()
        self.layers = nn.ModuleList()
        self.layers.append(nn.Linear(1, width))
        for _ in range(depth - 2):
            self.layers.append(nn.Sequential(
                nn.Linear(width, width),
                nn.Dropout(dropout),
                nn.BatchNorm1d(width),
                activation
            ))
        if depth >= 2:
            self.layers.append(nn.Linear(width, 1))
        else:
            self.layers.append(nn.Linear(1, 1))

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

模型训练

我们定义train函数进行训练和验证，具体代码如下：

训练函数 Python

```python
def train(train_set, val_set, epoches, depth, width,activation,batch_size, pic,lr=1e-3):
    model = Net(width, depth, activation).to(device)
    loss_func = nn.MSELoss()
    train_loss = []
    val_loss = []
    optimaizer = optim.Adam(model.parameters(), lr)
    train_loader = DataLoader(dataset=train_set, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(dataset=val_set, batch_size=batch_size, shuffle=False)
    train_loss_list = []
    valid_loss_list = []
    for epoch in range(epoches):
        model.train()
        train_loss = 0
        for k, (data,label) in enumerate(train_loader):
            data, label = data.to(device), label.to(device)
            optimaizer.zero_grad()
            output = model(data)
            loss = loss_func(output, label)
            loss.backward()
            optimaizer.step()
            train_loss += (loss.item()-train_loss)/(k+1)
        model.eval()
        val_loss = 0
        with torch.no_grad():
            for k, (data,label) in enumerate(val_loader):
                data, label = data.to(device), label.to(device)
                output = model(data)
                loss = loss_func(output, label)
                val_loss += (loss.item()-val_loss)/(k+1)
            train_loss_list.append(train_loss)
            valid_loss_list.append(val_loss)
            if epoch % 200 == 0:
                print(f"\t\t Epoch: {epoch}, Loss: {loss.item()}")
            if pic is not None:
                loss_curve(train_loss_list, valid_loss_list, pic)
    return model, train_loss, val_loss
```

超参数搜索及结果

我们对网络深度、网络宽度、激活函数和学习率进行了超参数搜索，具体代码如下：

超参数搜索

```python
def search_hyperparameter(train_set, val_set, num_epoch, batch_size):
    depth_list = [3,5,7]
    width_list = [5,10,15]
    # activation_list = [nn.relu, nn.tanh, nn.sigmoid, nn.leaky_relu, nn.elu]
    activation_list = [nn.ReLU(), nn.ReLU6(), nn.Sigmoid(), nn.Tanh()]
    lr_list = [1e-3, 1e-2,1e-1]
    min_val_loss = 1e10
    parameter_list = []
    for i, depth in enumerate(depth_list):
        for j, width in enumerate(width_list):
            for k, activation in enumerate(activation_list):
                for l, lr in enumerate(lr_list):
                    print(f'Current depth: {depth}, width: {width}, activation: {activation}, lr: {lr}')
                    pic = f'depth_{depth}_width_{width}_activation_{activation}_lr_{lr}.png'
                    model, train_loss, val_loss = train(train_set, val_set, num_epoch, depth, width, activation, batch_size, pic, lr)
                    if val_loss < min_val_loss:
                        min_val_loss = val_loss
                        best_depth = depth
                        best_width = width
                        best_activation = activation
                        best_lr = lr
    return best_depth, best_width, best_activation, best_lr
```

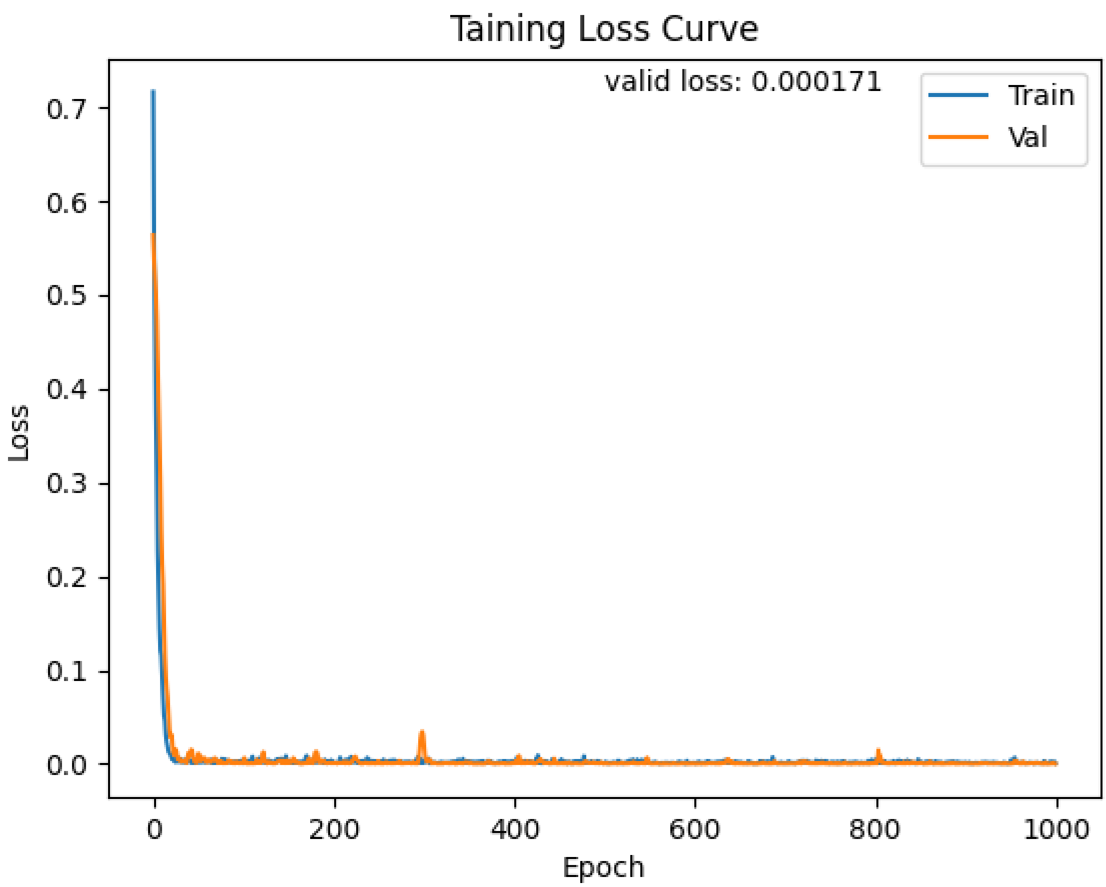
最终得到最超参数为：

- 网络深度：7
- 网络宽度：15
- 激活函数：ReLU()
- 学习率：0.001

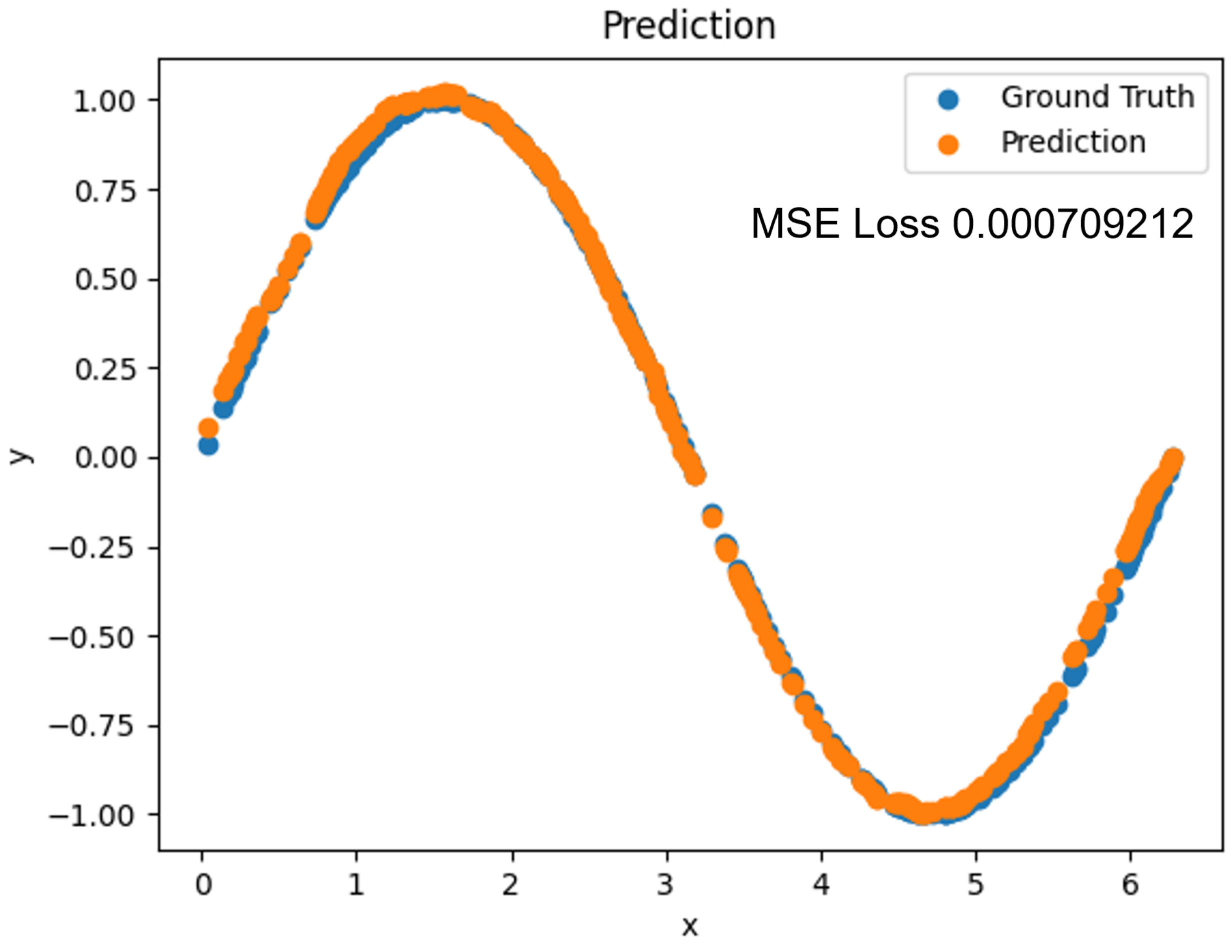
实验发现，对于所设计的网络而言，当网络深度和网络宽度增加时，模型拟合的效果越好。不过由于任务本身并不困难，因此当深度宽度相对较大时，实验效果区分并不大。而当学习率设置为0.1时，会出现一定的波动现象，这也说明了学习率的选择需要适中，不能过大。

实验结果

在超参数搜索以后的最优实验结果曲线为：



得到的测试集损失为0.0007092123269103467，对应曲线为：



更多不同超参数设定下训练验证损失函数曲线见./figs文件夹。