Lab 01 Report

实验内容

基于pytorch实现前馈神经网络（FNN)，近似以下函数：

$$y = log_2(x) \ + cos(2\pi x), x \in [1, 16]$$

实验步骤

1.构造FNN类

基于隐藏层 `hidden_sizes` 、激活函数 `activate_function` 构建前馈神经网络类 `myFNN` ，并根据不同的 `activate_function` 设置

```
class myFNN(nn.Module):
    def __init__(self, hidden_sizes=[10], activate_function = 'relu'):
        """

        :param hidden_sizes: numbers of nodes of each hidden layers (list)
        :param activate_function: 可选relu, sigmoid, tanh
        """
        super(myFNN, self).__init__()
        # 默认精度是float，要转化成double
        self.input_layer = nn.Linear(1, hidden_sizes[0], dtype=torch.double)
        self.hidden_layers = nn.ModuleList()
        for i in range(len(hidden_sizes) - 1):
            self.hidden_layers.append(nn.Linear(hidden_sizes[i], hidden_sizes[i + 1], dtype=torch.double))
        self.output_layer = nn.Linear(hidden_sizes[-1], 1, dtype=torch.double)
        self.activate_function = activate_function

    def forward(self, x):
        if self.activate_function == 'relu':
            x = torch.relu(self.input_layer(x))
            for layer in self.hidden_layers:
                x = torch.relu(layer(x))
            x = self.output_layer(x)
            return x
```

2.生成数据并封装

- **数据生成**：x随机选取区间 $[1, 16]$ 的 `N` 个值，并根据上面的公式生成labels（y值）。
- **数据划分**：依据 `train_rate`，`val_rate`，`test_rate` 将数据集随机划分为训练集、验证集、测试集（ `torch.utils.data.random_split` 函数）
- **数据封装**：利用 `torch.utils.data.DataLoader` 函数将数据分成不同的batch，并指定 `batch_size` 。每个epoch前随机打乱训练集 `shuffle=True` ，以达到充分下降。

```
train_data, val_data, test_data = torch.utils.data.random_split(dataset, [num_train, num_val, num_test])
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_data, batch_size=batch_size, shuffle=False)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

3.定义loop

- **训练过程**：遍历每一个batch，计算输入值的输出，基于目标值（labels）计算损失（MSEloss）和损失的梯度，最后用 `optimizer` 优化参数，并记录loss值

```
def train_loop(data_loader, model, optimizer, criterion=nn.MSELoss()):
    global global_train_losses
    model.train()  # Batch Normalization和Dropout
    total_loss = 0

    for inputs, targets in data_loader:
        batch_size = len(inputs)
        optimizer.zero_grad()  # 在每个batch之前清除梯度，加速
        outputs = model(inputs)  # 前向计算
        loss = criterion(outputs, targets)  # 计算损失
        loss.backward()  # 计算梯度
        optimizer.step()  # 更新模型参数
        total_loss += loss.item() * batch_size
```

```
    print('Loss: {:.7f}'.format(total_loss / len(data_loader.dataset)))
    global_train_losses.append(total_loss / len(data_loader.dataset))
```

- **验证/测试过程**：遍历batch，累积损失

4.训练模型并可视化

- **训练模型**：
  - 定义优化器(优化方法为Adam或SGD)
  - 指定 `max_num_epochs` ，循环执行 `train_loop` （可在验证集计算损失，早停退出循环）

```
for epoch in range(max_num_epochs):
        train_loop(train_loader, model, optimizer, criterion)
        val_loop(val_loader, model)
        epoch_num += 1
        # +早停 训练集上误差连续5次增加则停止循环
        if global_val_losses[-1] > best_val_loss:
            count += 1
        else:
            best_val_loss = global_val_losses[-1]
            best_model = model
            count = 0
        if count == patience:
            break

    model = best_model

    print("epoch times", epoch_num)
    test_loop(test_loader, model, criterion)
```

- **模型评估**：
  计算测试集上的误差
- **可视化**：
  - 绘制训练过程中训练集和验证集损失函数（MSE）
  - 绘制测试集x-target的散点图和x-prediction散点图并比较。

5. 参数调试

设置了 `batch_sizes` , `hidden_sizes` , `activate_functions` `optimizer_types` , `lrs` 五个参数用于调试。其中， `hidden_sizes` 和 `activate_functions` 用于调节神经网络结构， `optimizer_types` 和 `lrs` 调节优化方法的参数。具体结果见**实验结果**部分。

实验结果

1.调参结果

- N=1000时，最大迭代次数500次，调节激活函数类型 `activate_functions` `hidden_sizes` ,得到验证集上误差如下：

隐藏层大小\激活函数	tanh	relu	sigmoid
`[512]`	4.23e-1	3.16e-1	4.32e-1
`[256, 256]`	7.23e-4	1.85e-2	1.92e-1
`[128, 128, 128, 128]`	1.44e-3	3.68e-4	1.18e-1
`[64, 64, 64, 64, 64, 64, 64, 64]`	6.94e-4	7.94e-3	1.21e-1

- N=1000时，最大迭代次数500次，调节优化方法 `optimizer types` 和学习率 `learning rate` ,得到验证集上误差如下：

优化方法\学习率	0.1	0.01	0.001
SGD	1.62	0.42	0.60
Adam	1.64	0.49	5.46e-5

(其他调节过于繁琐，仅展示部分)

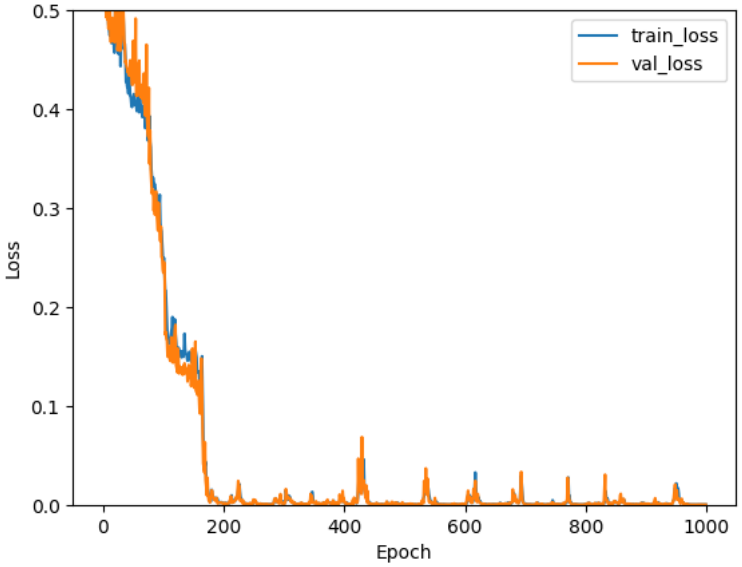
2.模型结果及可视化

数据集大小仍为1000，在如下参数条件下：

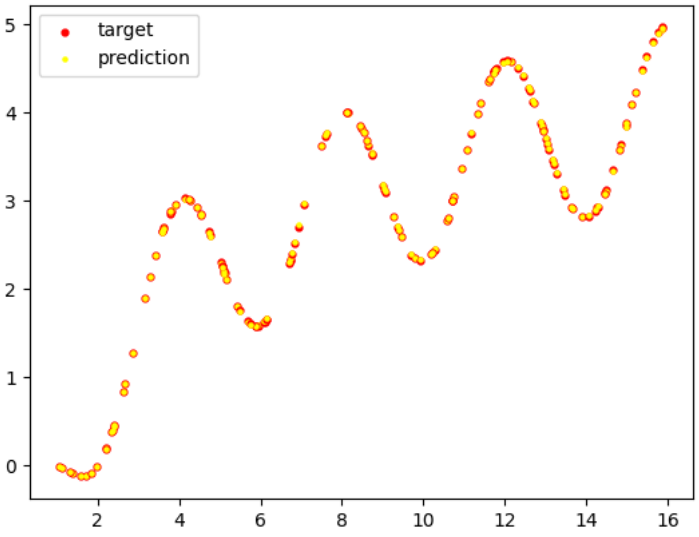
- hidden_sizes=[128,128,128,128,128,128]
- batch_size=64
- max_num_epochs=1000
- optimizer_type='Adam'
- lr=0.001
- activate_function = 'tanh'

结果如下：

- 验证集MSE：9.17e-5
- 训练集MSE：8.77e-5
- 损失函数图像：

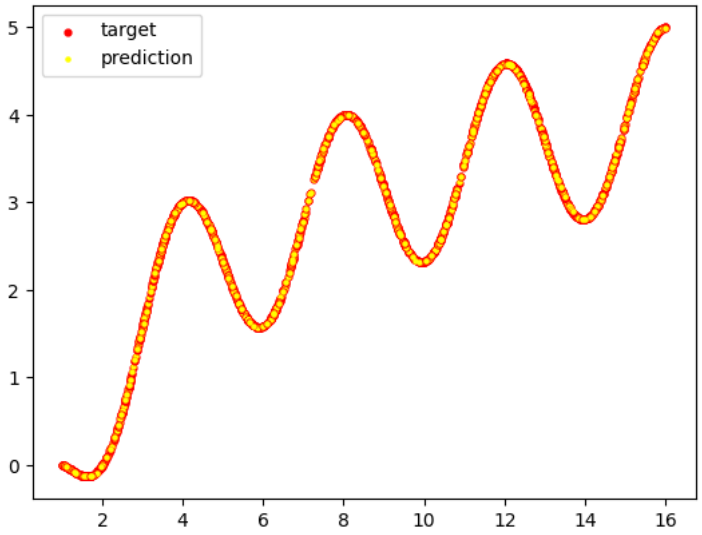


- 测试集x-target的散点图和x-prediction散点图：



为使得散点图连贯，我们对大小为10000的数据集上进行了测试，其他参数同上，得到了如下结果：

- 验证集MSE：2.18e-5
- 训练集MSE：2.18e-5
- 测试集x-target的散点图和x-prediction散点图：



结果分析

- 神经网络的参数调节
  - tanh和relu的表现都明显优于sigmoid函数
  - 在节点数相同时，更深的神经网络比更宽的神经网络效果往往比更好
- 优化方法参数调节
  - 在本实验中，Adam方法优于SGD方法，能达到更优的效果（特别是学习率比较低时）
  - 学习率越低，效果越好
- 其他：
  - batch size：调参过程中发现：batch size 过小，循环次多从而故时间开销较大，有可能导致过拟合；batch size过大，收敛速度慢，泛化能力弱。在本实验中选择batch size = 64较为合适。
  - epoch：因为本实验要拟合的函数简单，性质好，不存在过拟合的情况，增加epoch不仅能减少训练误差，也能同样的减少验证集上的误差，最后在测试集上的结果也比较好。所以没有设置早停）