中国科学技术大学

University of Science and Technology of China

深度学习导论

实验 1

姓名 徐海阳　　学号 PB20000326　　院系 少年班学院

摘　要

本实验使用 PyTorch 用于近似函数：$y = sin(x) + cos(x) + sin(x)cos(x), x \in [0, 2\pi)$, 完成数据生成、模型搭建、模型训练、调参分析及测试性能。特别地，调参分析中重点研究了以下设置对模型性能的影响：

网络模型参数：是否使用 norm 层、初始化权重选择、深度、宽度、激活函数等；

优化器及学习率：优化器类型、优化器超参数、学习率选择及调整等。

经过充分调优，本实验得到了最优的模型。同时，通过详细充分的消融实验、可视化图表和实验分析验证了模型的性能，在测试集上达到 MSE=0.00103。

目录

1 数据生成

如图 1所示，数据生成 $y = sin(x) + cos(x) + sin(x)cos(x), x \in [0, 2\pi)$ 由 build_dataset 函数完成。具体 Args 及 Returns 见图。模型生成过程会先随机采样 num_samples 个 x 值，接着得到相应的 y；然后 shuffle 数据后根据 split_ratio 进行 train/val/test 数据集的划分。

注意，Returns 中的 mean 和 std 是所有 sample x 的均值和标准差，是属于数据集本身的标准化参数。当给定 x 值，想要预测 y 值时，需要先用 mean 和 std 对 x 做标准化，然后通过神经网络 forward 来得到预测的 $\hat{y}$。

```python
def build_dataset(num_samples, split_ratio=(0.6, 0.2), seed=0, device="cuda"):
    """ Build the dataset following the lab requirements below:
    在 [0, 2PI) 范围内随机sample x, 并计算 y = sin(x) + cos(x) + sin(x)cos(x) 作为 y 值。

    Args:
        num_samples (int): sample number in the total dataset
        split_ratio (tuple): train, val split ratio
        seed (int): random seed
        device: "cuda" or "cpu". Defaults to "cuda".

    Returns:
        x_train, y_train, x_val, y_val, x_test, y_test, mean, var
    """
```

图 1: 数据生成

2 模型搭建

如图 2所示，MLP 被选择为本实验的模型。选择 MLP 的理由是：首先，数据不是图像格式，不宜使用 CNN 等卷积神经网络模型；其次，数据没有时间关联，不宜使用 RNN 等循环神经网络模型；事实上，输入 x，输出 y，自然想到可以在中间搭建隐藏层，也就是使用 MLP 模型。

模型的 Args 如图所示。注意，与最简单的线性层 + 激活函数组成的 MLP 不同，本实验使用的模型有 3 点区别：

1. 原始输入是 x，但是真实使用的输入是 $[x, x^2, ..., x^{in\_chans}]$。即人为增加非线性，希望能更好的拟合三角函数（因为由数学知识可知，泰勒展开后都是这样的多项式项，因此本身提供幂次项后让模型更容易学习）；
2. 使用了可选择的 nn.BatchNorm1d 对数据进行批标准化；
3. 对 layer 的 weights 使用了初始化，具体来说是 xavier 初始化。

```python
class net(nn.Module):
    def __init__(self, in_chans: int, depth: int, embed_dims: list, act: nn.Module, norm: bool = True, device="cuda"):
        """ Build the neural network, which could manually modify its:
            in_chans, depth, embed_dims, activation functions and whether has norm layer

        Args:
            in_chans (int): e.g. 1 ([x]); 2 ([x, x^2]); 3 ([x, x^2, x^3]), ...
            depth (int): e.g. 5
            embed_dims (list): e.g. [1, 10, 50, 10, 1]
            act (nn.Module): e.g. nn.ReLU
            norm (bool, optional): whether has norm layer. Defaults to True.
            device: "cuda" or "cpu". Defaults to "cuda".

        """
```

图 2: 模型搭建

3 模型训练

如图 3所示，即为模型训练过程。

定义好 optimizer 和学习率 scheduler 之后，首先，optimizer 清空上一 iteration 的梯度；接着，前向传播、计算 loss、反向传播；然后优化器根据反向传播的梯度及定义的学习率进行参数更新；最后进入下一个 iteration。具体地，本实验使用的 optmizer 为 SGD，初始学习率为 0.1，动量为 0.9。

每个 epoch 结束，在验证集上得到 val_loss，学习率 scheduler 据此判断是否更新学习率。具体地，本实验使用的 scheduler 是：若 val_loss 连续 10 个 epoch 不下降，则 lr 调整为原先 lr 的 1/10。

每 10 个 epoch 进行一次信息的打印，当学习率过小时停止训练。具体地，本实验在 lr<1e-4 时停止训练（事实上调低停止阈值可以获得更好的性能，这是非常容易的事情）。

```python
# training
train_losses = []
val_losses = []

num_epochs = 10000
for epoch in range(num_epochs):
    for idx, (samples, labels) in enumerate(train_loader):

        # clear grad
        optimizer.zero_grad()

        # forward, backward, optimize
        y_pred = model(samples)
        loss = criterion(y_pred, labels)
        loss.backward()
        optimizer.step()

    # lr scheduler
    with torch.no_grad():
        y_train_pred = model(x_train)
        train_loss = criterion(y_train_pred, y_train)
        y_val_pred = model(x_val)
        val_loss = criterion(y_val_pred, y_val)
    scheduler.step(val_loss)

    # record for drawing loss curve
    train_losses.append(torch.log(train_loss).item())
    val_losses.append(torch.log(val_loss).item())

    # print info
    if epoch % 10 == 0:
        lr = optimizer.param_groups[0]["lr"]
        if lr < 1e-4:
            print("early stop")
            break
        print(f"Epoch {epoch}, train loss: {train_loss.item()}, val loss: {val_loss.item()}, lr: {lr}")
```

图 3: 模型训练

4 调参分析

4.1 网络模型参数

首先，我们探究 Norm 和 Init Weight 对模型性能的影响。

如表 1所示，我们可以看到 Norm 和 Init Weight 对于模型性能的提升都是很明显的。特别地，这 2 个方法都不使用，Val MSE Loss 为 1.07809；都使用，并且使用 $xavier\_normal$，可以立刻让 Val MSE Loss 变为 0.00273。这说明 Norm 和 Init Weight 可以让模型更好更快地收敛。

然后，我们探究网络宽度、网络深度和激活函数对于模型性能的影响。

从表 2可以看出：

1. 第一组 ablation，人为减小学习非线性函数的难度，将 input 从 x 增广为 $[x, x^2]$, 对性能的提升最大；

Norm	Init Weight	Val MSE Loss ↓
✗	✗	1.07809
✓	✗	0.30706
✓	*kaiming_normal*	0.15392
✓	*xavier_normal*	**0.00273**

表 1: Ablation of Norm and Init Weight

2. 第二组 abaltion，网络深度并不是越深越好，在 depth>5 后性能反而略有下降，猜测可能是因为梯度消失，即网络太深，传到前面的 grad 太小，导致浅层网络学习得一般；

3. 第三组 ablation，网络宽度并不是越宽越好，参数太多可能过拟合。最优宽度为 $[2, 10, 100, 10, 1]$；

4. 第四组 ablation，最优的激活函数是 ReLU 函数。

Depth	Width	Activation	Val MSE Loss ↓
5	1, 10, 100, 10, 1	ReLU	0.00316
5	2, 10, 100, 10, 1	ReLU	**0.00273**
5	3, 10, 100, 10, 1	ReLU	0.00324
5	4, 10, 100, 10, 1	ReLU	0.00288
3	2, 10, 1	ReLU	0.00470
5	2, 10, 100, 10, 1	ReLU	**0.00273**
7	2, 10, 100, 200, 100, 10, 1	ReLU	0.00313
9	2, 10, 100, 200, 800, 200, 100, 10, 1	ReLU	0.00330
5	2, 10, 50, 10, 1	ReLU	0.00303
5	2, 10, 100, 10, 1	ReLU	**0.00273**
5	2, 50, 100, 50, 1	ReLU	0.00324
5	2, 50, 500, 50, 1	ReLU	0.00325
5	2, 10, 100, 10, 1	Sigmoid	0.00297
5	2, 10, 100, 10, 1	Tanh	0.01236
5	2, 10, 100, 10, 1	ReLU	**0.00273**

表 2: Ablation of width, depth and activation

4.2 优化器及学习率

我们探究 optimizer，learning rate 以及它的 scheduler 对模型性能的影响。

从表 3中可以看出：

1. SGD 优化器比 Adam 优化器效果略好；

2. SGD 优化器在 momentum=0.9 时效果最优，momentum=0 时容易陷入局部极小值点，momentum=0.99 又容易冲出全局最小值点；

3. 学习率初始化为 0.01 效果最优；

4. 若学习率一直不改变，则模型不收敛；若学习率每 10 个 epoch 衰减到原先的 1/10，则收敛不充分；实际实验中采用若连续 10 个 epoch 模型性能都不提升（落入 plateau），则将学习率衰减

到原先的 1/10，这样可以保证在每次衰减前，在较大的尺度上已经接近最优，衰减后可以更细致地调整、逼近最优点。

Optimizer	LR&Scheduler	Val MSE Loss ↓
Adam	init 0.1, x 0.1 if plateau	0.00287
SGD(m=0.9)	init 0.1, x 0.1 if plateau	**0.00273**
SGD(m=0)	init 0.1, x 0.1 if plateau	0.00337
SGD(m=0.9)	init 0.1, x 0.1 if plateau	**0.00273**
SGD(m=0.99)	init 0.1, x 0.1 if plateau	0.00331
SGD(m=0.9)	init 1, x 0.1 if plateau	0.00351
SGD(m=0.9)	init 0.1, x 0.1 if plateau	**0.00273**
SGD(m=0.9)	init 0.01, x 0.1 if plateau	0.00372
SGD(m=0.9)	always 0.1	–
SGD(m=0.9)	x 0.1 every 10 epochs	0.19203
SGD(m=0.9)	init 0.1, x 0.1 if plateau	**0.00273**

表 3: Ablation of optimizer and scheduler

5 测试性能

运行以下指令，

pip install -r requirements.txt

python main.py

然后会得到如图 4输出, 在测试集上的 MSE loss 为 0.00103。

```
(base) xuhaiyang@lebron:~/USTC/DL/Lab1$ python main.py
**************************************************
Start Training
**************************************************
Epoch 0, train loss: 0.17312946915626526, val loss: 0.17310434579849243, lr: 0.1
Epoch 10, train loss: 0.15953059494495392, val loss: 0.15244893729686737, lr: 0.1
Epoch 20, train loss: 0.15616220235824585, val loss: 0.15899989008903503, lr: 0.1
Epoch 30, train loss: 0.026824666187167168, val loss: 0.017355944961309433, lr: 0.1
Epoch 40, train loss: 0.027745524421334267, val loss: 0.019151965156197548, lr: 0.1
Epoch 50, train loss: 0.010786252096295357, val loss: 0.003169921226799488, lr: 0.1
Epoch 60, train loss: 0.0006354126962833107, val loss: 0.005491513293236494, lr: 0.010000000000000002
Epoch 70, train loss: 0.0016035871813073754, val loss: 0.0008816515910439193, lr: 0.010000000000000002
Epoch 80, train loss: 0.0006658754427917302, val loss: 0.003590536769479513, lr: 0.010000000000000002
Epoch 90, train loss: 0.002879795152693987, val loss: 0.0004286132752895355, lr: 0.010000000000000002
Epoch 100, train loss: 0.0003323431883472949, val loss: 0.0023587672039866447, lr: 0.0010000000000000002
Epoch 110, train loss: 0.0002913346397690475, val loss: 0.002614742610603571, lr: 0.0010000000000000003
Early stop
Final epoch 120, train loss: 0.00028573558665812016, val loss: 0.002732498338446021, lr: 1.0000000000000004e-05

**************************************************
MSE Loss on Dataset
**************************************************
Train loss: 0.00028573558665812016
Val loss: 0.002732498338446021
Test loss: 0.0010319595457985997

**************************************************
Large Scale Test
**************************************************
For 10000 points in [0, 2PI):
The biggest loss is 0.0075248233042657375, the smallest loss is 1.3219647598816664e-11
The average loss is 0.0003351727209519595

**************************************************
Draw Loss Curve
**************************************************
Loss curve saved to /mnt/nvme2/xuhaiyang/USTC/DL/Lab1/loss_curve.png
```

图 4: Screen Output of main.py

同时，得到 train 和 val 的 loss curve(图 5) 和在 $[0, 2\pi)$ 上等间距取的 10000 个点的 log MSE loss(图 6)。从中可以看出 train 和 val 的 loss 一开始都大幅振荡，val loss 大于 train loss, 100 个 epoch 后趋于稳定，模型收敛。同时，模型可以很好地 generalize 到 $[0, 2\pi]$ 上，Log MSE Loss 几乎全部小于-5。

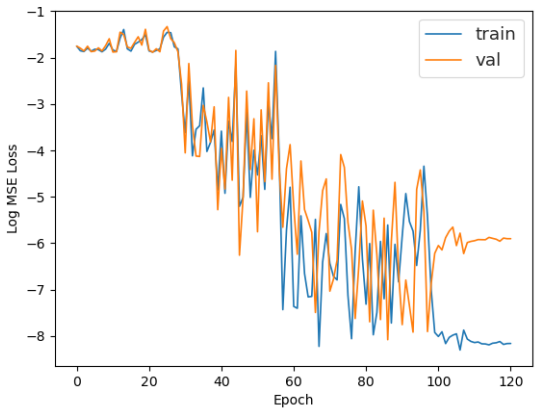

图 5: Log loss curve of training set and validation set

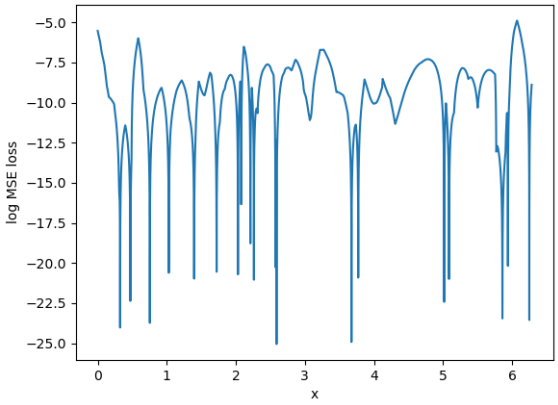

图 6: 10000 samples $\in [0, 2\pi)$ Log MSE Loss

6 实验总结

本实验通过大量充分的实验来拟合函数 $y = sin(x) + cos(x) + sin(x)cos(x), x \in [0, 2\pi)$。通过对网络模型参数和优化器及学习率的调优，在测试集上达到了较好的效果；并且有较强的 generalize 能力。