工科创 IV-J 平时作业 1

叶梓淳 520030910302

2023/2/27

1 问题 1

解决问题 1 的思路是从输出层的局部梯度开始计算，向前递归，计算出每层的梯度，再得出每层权重和偏置参数的梯度。

1.1 MLQP 随机方式下 BP 算法

记第 k 层 j^{th} 神经元输入为 n_{kj}，对应的局部梯度为 δ_{kj}，E_1 代表 1 号数据下对应的 total instantaneous error energy，e_{1kj} 代表 1 号数据第 k 层 j^{th} 神经元的误差信号，则

$$n_{kj} = \sum_{i=1}^{N_{k-1}} (u_{kji} x_{k-1,i}^2 + v_{kji} x_{k-1,i}) + b_{kj} \tag{1}$$

$$x_{kj} = f(n_{kj}) \tag{2}$$

$$\Delta u_{kji} = -\eta \frac{\partial E_1}{\partial u_{kji}} \tag{3}$$

$$\Delta v_{kji} = -\eta \frac{\partial E_1}{\partial v_{kji}} \tag{4}$$

$$\Delta b_{kj} = -\eta \frac{\partial E_1}{\partial b_{kj}} \tag{5}$$

其中，$2 \leq k \leq M$。

对于第 M 层 j^{th} 神经元，$\delta_{Mj} = e_{1Mj} f^{'}(n_{Mj})$, 则有

$$\Delta u_{Mji} = -\eta \frac{\partial E_1}{\partial u_{Mji}} = -\eta \frac{\partial E_1}{\partial x_{Mj}} \frac{\partial x_{Mj}}{\partial n_{Mj}} \frac{\partial n_{Mj}}{\partial u_{Mj}} = \eta \delta_{Mj} x_{M-1,i}^2 \tag{6}$$

$$\Delta v_{Mji} = -\eta \frac{\partial E_1}{\partial v_{Mji}} = -\eta \frac{\partial E_1}{\partial x_{Mj}} \frac{\partial x_{Mj}}{\partial n_{Mj}} \frac{\partial n_{Mj}}{\partial v_{Mj}} = \eta \delta_{Mj} x_{M-1,i} \tag{7}$$

$$\Delta b_{Mj} = -\eta \frac{\partial E_1}{\partial b_{Mj}} = \frac{\partial E_1}{\partial x_{Mj}} \frac{\partial x_{Mj}}{\partial n_{Mj}} \frac{\partial n_{Mj}}{\partial b_{Mj}} = \eta \delta_{Mj} \tag{8}$$

对于第 $k(1 < k < M)$ 层 j^{th} 神经元, 定义 $\delta_{kj} = f^{'}(n_{kj}) \sum_{i=1}^{N_{k+1}} \delta_{k+1,i}(2u_{k+1,ij}x_{kj} + v_{k+1,ij})$, 则有

$$\Delta u_{kji} = -\eta \frac{\partial E_1}{\partial u_{kji}} = -\eta \frac{\partial E_1}{\partial x_{kj}} \frac{\partial x_{kj}}{\partial n_{kj}} \frac{\partial n_{kj}}{\partial u_{kj}} = -\eta x_{k-1,i}^2 f^{'}(n_{kj}) \frac{\partial E_1}{\partial x_{kj}} \tag{9}$$

而

$$\begin{aligned}
\frac{\partial E_1}{\partial x_{kj}} &= \sum_{n=1}^{N_M} e_{1Mn} \frac{\partial e_{1Mn}}{\partial n_{Mn}} \frac{\partial n_{Mn}}{\partial x_{kj}} \\
&= -\sum_{n=1}^{N_M} \delta_{Mn} \sum_{i=1}^{N_{M-1}} f^{'}(n_{M-1}, i)(2x_{M-1,i}u_{Mni} + v_{Mni}) \frac{\partial n_{M-1,i}}{\partial x_{kj}} \\
&= -\sum_{i=1}^{N_{M-1}} \sum_{n=1}^{N_M} f^{'}(n_{M-1}, i)\delta_{Mn}(2x_{M-1,i}u_{Mni} + v_{Mni}) \frac{\partial n_{M-1,i}}{\partial x_{kj}} \\
&= -\sum_{i=1}^{N_{M-1}} \delta_{M-1,i} \frac{\partial n_{M-1,i}}{\partial x_{kj}} = \cdots \\
&= -\sum_{i=1}^{N_{k+1}} \delta_{k+1,i} \frac{\partial n_{k+1,i}}{\partial x_{kj}} \\
&= -\sum_{i=1}^{N_{k+1}} \delta_{k+1,i}(2u_{k+1,ij}x_{kj} + v_{k+1,ij})
\end{aligned} \tag{10}$$

因此

$$\Delta u_{kji} = \eta \delta_{kj} x_{k-1,i}^2 \tag{11}$$

同理,

$$\Delta v_{kji} = \eta \delta_{kj} x_{k-1,i} \tag{12}$$

$$\Delta b_{kji} = \eta \delta_{kj} \tag{13}$$

则由 (11)(12)(13) 式我们可得出 MLQP 在随机方式下 BP 算法。

1.2 MLQP 批量方式下 BP 算法

批量方式下 $E = \frac{1}{n}\sum_{i=1}^{n} E_i$，1.1 中的 δ_{kj} 现写作 δ_{1kj}，x_{kj} 写作 x_{1kj}，同理可得

$$\Delta u_{kji} = \frac{\eta}{n}\sum_{m=1}^{n} \delta_{mkj} x_{m,k-1,i}^2 \tag{14}$$

$$\Delta v_{kji} = \frac{\eta}{n}\sum_{m=1}^{n} \delta_{mkj} x_{m,k-1,i} \tag{15}$$

$$\Delta b_{kji} = \frac{\eta}{n}\sum_{m=1}^{n} \delta_{mkj} \tag{16}$$

2 问题 2

调用 pytorch 现有的函数库，构建含一层隐藏单元的 MLP 如下：

- 前向传播

```
model = nn.Sequential(
    nn.Linear(dim, hidden),
    nn.LeakyReLU(),
    nn.Linear(hidden, classes)
)
```

- 参数定义

```
criterion = torch.nn.CrossEntropyLoss() #交叉熵损失函数
optimizer = torch.optim.Adam( #Adam优化器
    model.parameters(),
    lr = learning_rate,
    weight_decay = lambda_l2
)
```

- 参数更新

```
for e in range(epoch):
    Y_pred = model(X)
    loss = criterion(Y_pred, Y)
```

```
    score, predicted = torch.max(Y_pred, 1)
    acc = (Y == predicted).sum().float() / len(Y)
    loss_list.append(loss)
    acc_list.append(acc)
    display.clear_output(wait = True)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

经过分析与实验，均方差损失函数与 *sigmoid* 函数组合会导致模型输出值在 0，1 附近梯度趋于 0，因此选择交叉熵作为损失函数，表达式为：

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i = \frac{1}{N}\sum_{i=1}^{N} -[y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)]$$

且选择 *leakyReLU* 作为激活函数。而接着对比 SDG 与 Adam 优化器，令学习率为 0.001，训练 10000 轮得到训练集的决策边界如图 1,2：

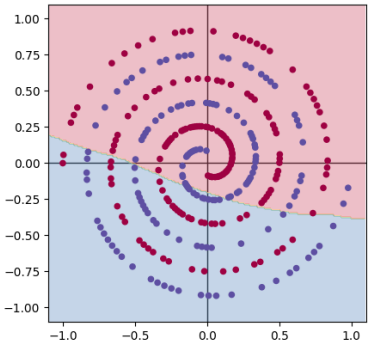

图 1: 使用 SGD 优化

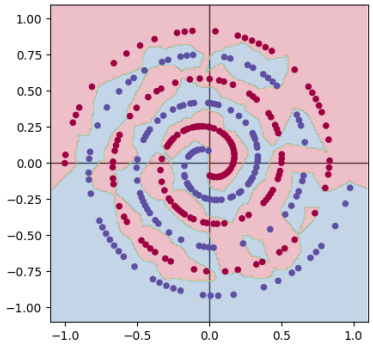

图 2: 使用 Adam 优化

显然，使用 Adam 优化器得到的效果更好，因为 SGD 在这个例子下会陷入局部最优。而 Adam 优化器使用了动量加速梯度下降，且动量直接并入了梯度一阶矩（指数加权）的估计。其次，Adam 包括偏置修正，修正从原点初始化的一阶矩（动量项）和（非中心的）二阶矩估计，下降更加平滑。所以我们选择 Adam 优化器。

选择三种学习率大小：0.1,0.01,0.001，分别训练 10000 轮，得到决策边界如图 3，4，5：

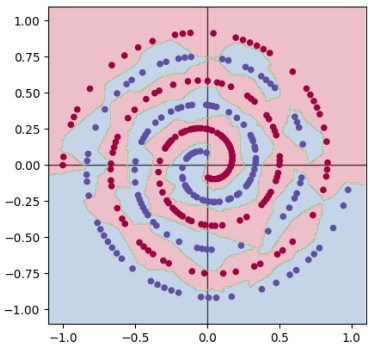

图 3: lr = 0.1

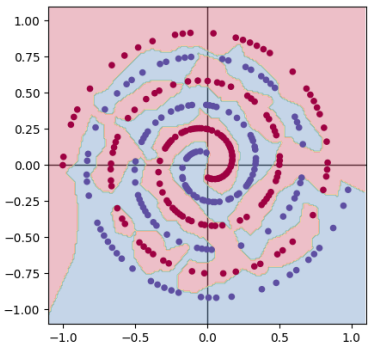

图 4: lr = 0.01

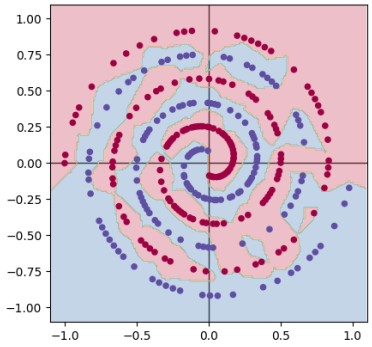

图 5: lr = 0.001

对应的损失函数与准确率随迭代次数变化如图 6：

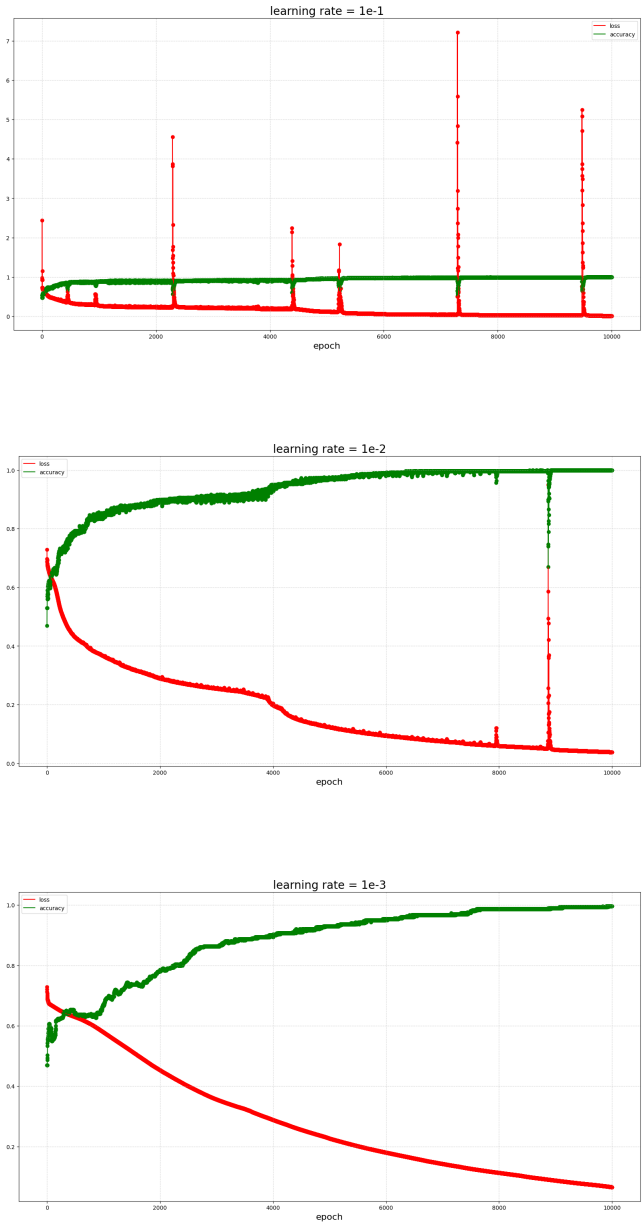

图 6: 不同学习率对应 loss 与 accuracy 变化曲线

可以看出较高的学习率在模型适配的情况下需要轮数更少便可达到收

敛，但 loss 可能存在较大的波动。因此我选择使用学习率 0.001，轮数为 30000 来训练模型并用来预测测试集，得到四个阶段的决策边界变化图 7,8,9,10, 最终的预测正确率为 0.94。

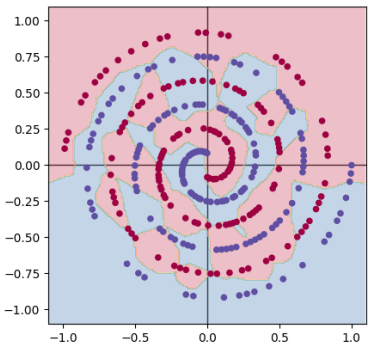

图 7: epoch = 3750

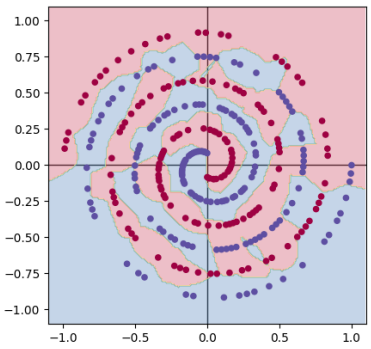

图 8: epoch = 7500

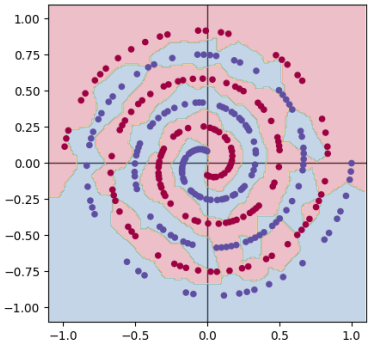

图 9: epoch = 15000

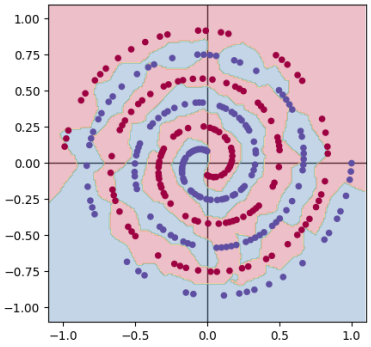

图 10: epoch = 30000