# 工程实践与科技创新 IV-J

## 课程作业

pangbo 5190309xxxx

2022 年 3 月 13 日

**题目 1.** MLQP 反向传播算法推导。

为方便表述，现将推导过程中的符号及其含义简述如下：

- $x_i^{(k)}(n)$ 代指第 $n$ 次迭代时，第 $k$ 层输入向量的第 $i$ 个元素。
- $t_j^{(k)}(n)$ 代指第 $n$ 次迭代时，第 $k$ 层第 $j$ 个神经元激活函数输入值。
- $u_{ji}^{(k)}(n)$、$v_{ji}^{(k)}(n)$ 代指第 $n$ 次迭代时，连接第 $k-1$ 层第 $i$ 个神经元与第 $k$ 层第 $j$ 个神经元的权重值，$u$、$v$ 分别代指二次项与一次项的系数。
- $y_j^{(k)}(n)$ 代指第 $n$ 次迭代时，第 $k$ 层第 $j$ 个神经元的输出，$y_j^{(k)}(n) = x_j^{(k+1)}(n)$。
- $d_j(n)$ 代指第 $n$ 次迭代时，输出层第 $j$ 个神经元应当输出的值。
- $e_j(n)$ 代指第 $n$ 次迭代时，输出层第 $j$ 个神经元输出与实际值的差，即 $e_j(n) = d_j(n) - y_j(n)$。
- $\varepsilon(n)$ 代指第 $n$ 次迭代输出的均方误差。

首先考虑输出层参数关于损失函数的梯度，

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial u_{ji}^{(k)}} &= \frac{\partial \varepsilon}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial t_j}\frac{\partial t_j}{\partial u_{ji}^{(k)}} \\
&= e_j \cdot (-1)\sigma'(t_j){x_i^{(k)}}^2 \\
&= -e_j\sigma'(t_j){x_i^{(k)}}^2
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}^{(k)}} &= \frac{\partial \varepsilon}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial t_j}\frac{\partial t_j}{\partial v_{ji}^{(k)}} \\
&= e_j \cdot (-1)\sigma'(t_j)x_i^{(k)} \\
&= -e_j\sigma'(t_j)x_i^{(k)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial b_j^{(k)}} &= \frac{\partial \varepsilon}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial t_j}\frac{\partial t_j}{\partial b^{(k)}} \\
&= e_j \cdot (-1)\sigma'(t_j) \cdot 1 \\
&= -e_j\sigma'(t_j)
\end{aligned}
$$

为方便表述，记 $\delta_j^{(k)} = -e_j\sigma'\left(t_j^{(k)}\right)$，于是 $\frac{\partial \varepsilon}{\partial u_{ji}^{(k)}} = \delta_j^{(k)} {x_i^{(k)}}^2$、$\frac{\partial \varepsilon}{\partial v_{ji}^{(k)}} = \delta_j^{(k)} x_i^{(k)}$、$\frac{\partial \varepsilon}{\partial b_{ji}^{(k)}} = \delta_j^{(k)}$。

对于隐藏层神经元，首先分析每个神经元输出对于最终损失的贡献值，即损失函数对于 $y_j^{(k-1)}$ 的导数。

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial y_j^{(k-1)}} &= -\sum_p e_p\sigma'(t_p^{(k)})\frac{\partial t_p^{(k)}}{\partial y_j^{(k-1)}} \\
&= -\sum_p e_p\sigma'(t_p^{(k)})\frac{\partial t_p^{(k)}}{\partial x_j^{(k)}} \\
&= -\sum_p e_p\sigma'\left(t_p^{(k)}\right)\left(2u_{pj}^{(k)}x_j^{(k)} + v_{pj}^{(k)}\right) \\
&= \sum_p \delta_p^{(k)}\left(2u_{pj}^{(k)}x_j^{(k)} + v_{pj}^{(k)}\right)
\end{aligned}
$$

基于此我们可以给出隐藏层每个参数对损失函数的导数。

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial u_{ji}^{(k-1)}} &= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}}\frac{\partial y_j^{(k-1)}}{\partial t_j^{(k-1)}}\frac{\partial t_j^{(k-1)}}{\partial u_{ji}^{(k-1)}} \\
&= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}}\sigma'\left(t_j^{(k-1)}\right){x_i^{(k-1)}}^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}^{(k-1)}} &= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}}\frac{\partial y_j^{(k-1)}}{\partial t_j^{(k-1)}}\frac{\partial t_j^{(k-1)}}{\partial v_{ji}^{(k-1)}} \\
&= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}}\sigma'\left(t_j^{(k-1)}\right)x_i^{(k-1)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial b_{ji}^{(k-1)}} &= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}} \frac{\partial y_j^{(k-1)}}{\partial t_j^{(k-1)}} \frac{\partial t_j^{(k-1)}}{\partial b_{ji}^{(k-1)}} \\
&= \frac{\partial \varepsilon}{\partial y_j^{(k-1)}} \sigma' \left( t_j^{(k-1)} \right)
\end{aligned}
$$

用带 $\delta_j^{(k)}$ 的式子表示各层参数的梯度。

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial u_{ji}^{(k)}} &= \delta_j^{(k)} x_i^{(k)2} = \delta_j^{(k)} y_i^{(k-1)2} \\
\frac{\partial \varepsilon}{\partial v_{ji}^{(k)}} &= \delta_j^{(k)} x_i^{(k)} = \delta_j^{(k)} y_i^{(k-1)2} \\
\frac{\partial \varepsilon}{\partial b_j^{(k)}} &= \delta_j^{(k)}
\end{aligned}
$$

其中，$\delta_j^{(k)}$ 值如下。

$$
\delta_j^{(k)} = \frac{\partial \varepsilon}{\partial t_j^{(k)}} = \begin{cases} -e_j \sigma' \left( t_j^{(k)} \right)，\text{第 k 层为输出层} \\ \sigma' \left( t_j^{(k)} \right) \sum_p \delta_p^{(k+1)} \left( 2u_{pj}^{(k+1)} x_j^{(k+1)} + v_{pj}^{(k+1)} \right)，\text{第 k 层为隐藏层} \end{cases}
$$

**题目 2.** 使用编程语言为 MLQP 模型实现反向传播算法，训练一个带一个隐藏层的 MLQP 模型，并比较三个不同学习率下的训练效率与决策边界。

代码实现基于上述推导结果进行，模型与参数更新的核心代码位于文件`MLQP.py`、`optimizer.py`内。这部分代码包括 3 个主要部分。

- 前向传播

```python
for layer_id in range(1,self.layers_nr):
    self.Tk[layer_id] = np.matmul(self.Uk[layer_id],np.square(self.Yk[layer_id-1])) +
        np.matmul(self.Vk[layer_id],self.Yk[layer_id-1]) + self.Bk[layer_id]
    self.Yk[layer_id] = self.sigmoid(self.Tk[layer_id])
```

- 反向传播

```python
# MSE损失函数
# self.DeltaK[-1] = (self.Yk[-1] - self.tag) * self.Yk[-1] * (np.ones_like(self.Yk[-1])-self.Yk[-1])

# 交叉熵损失函数
self.DeltaK[-1] = self.Yk[-1] - self.tag

for layer_id in range(self.layers_nr-2,0,-1):
```

```
    self.DeltaK[layer_id] = np.matmul((self.Uk[layer_id+1].T*self.Yk[layer_id]*2+self.Vk[layer_id+1].T),self.DeltaK[layer_id+1]) \
        * self.Yk[layer_id] \
        * (np.ones_like(self.Yk[layer_id])-self.Yk[layer_id])
```

- 参数更新（SGD）

```
model = self.model
for layer_id in range(1,model.layers_nr):
    self.detUk[layer_id] = np.square(model.Yk[layer_id-1]).T * model.DeltaK[layer_id]
    self.detVk[layer_id] = model.Yk[layer_id-1].T * model.DeltaK[layer_id]
    self.detBk[layer_id] = model.DeltaK[layer_id]
    model.Uk[layer_id] -= lr*self.detUk[layer_id]
    model.Vk[layer_id] -= lr*self.detVk[layer_id]
    model.Bk[layer_id] -= lr*self.detBk[layer_id]
```

- 参数更新（SGDM）

```
model = self.model
for layer_id in range(1,model.layers_nr):
    self.detUk[layer_id] = lr2*model.DeltaK[layer_id] * np.square(model.Yk[layer_id-1].T) + momentum*self.detUk[layer_id]
    self.detVk[layer_id] = lr*model.DeltaK[layer_id] * model.Yk[layer_id-1].T + momentum*self.detVk[layer_id]
    self.detBk[layer_id] = lr*model.DeltaK[layer_id] + momentum*self.detBk[layer_id]
    model.Uk[layer_id] -= self.detUk[layer_id]
    model.Vk[layer_id] -= self.detVk[layer_id]
    model.Bk[layer_id] -= self.detBk[layer_id]
```

我首先研究了不同学习率对模型训练的影响，我分别使用 0.1/0.01/0.001 的学习率训练 1000 轮模型。模型的决策分界面如图1所示，图中蓝色与红色点代表测试数据集中两类数据点。模型训练过程中的损失函数与准确率如图2所示，从上到下分别为学习率 0.1/0.01/0.001 的曲线，每个子图的纵坐标尺度不相同；图中黑色折线代表损失函数值，红色折线代表模型对训练数据集预测的准确率，蓝色折线代表模型对测试数据集预测的准确率。

从图2中可以看出，较高的学习率（如 0.1）可以加快参数更新速率，但也可能导致参数更新过快产生波动；较小的学习率（如 0.01）可以减小参数更新速率，但参数更新过慢，达到同样训练效果需要的时间更长，也更可能被困于局部最优点。

我选择使用动态调整学习率的方法更新模型参数，首先用较大的学习率快速更新参数，如果训练过程中连续 200 轮没有有效提升，则将学习率降低为十分之一。类似于模

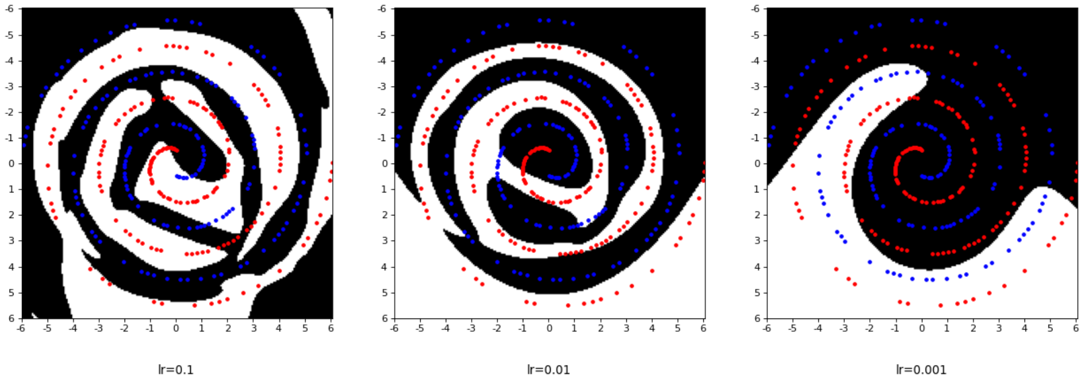


图 1: MLQP 模型不同学习率下 1000 轮训练后决策边界图

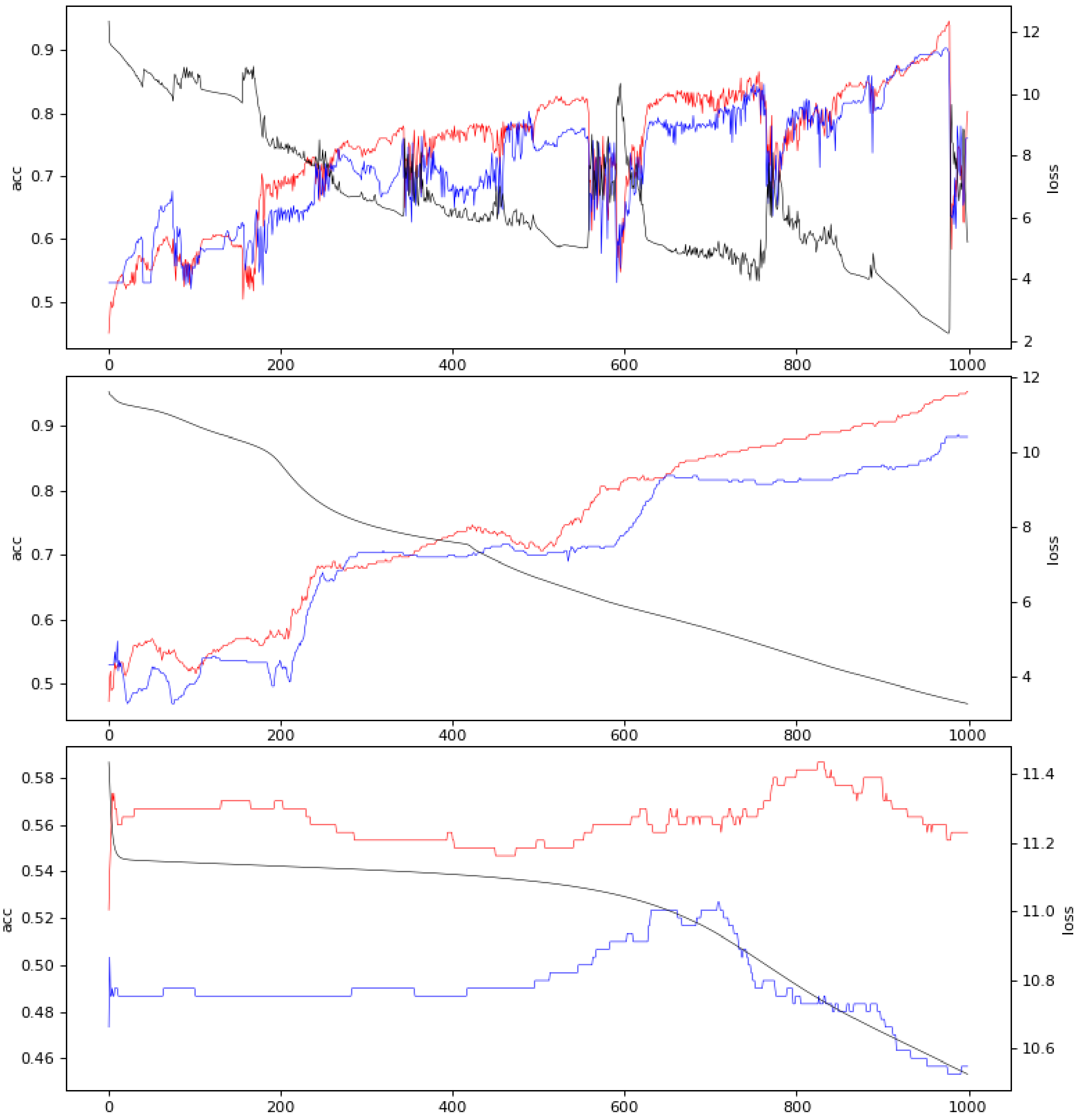


图 2: MLQP 模型不同学习率下训练过程中损失函数值与准确率曲线

拟退火算法，这样的动态学习率既可以加快前期训练速度，也可以避免后期模型参数跳变。

经过 2000 轮训练，单一 MLQP 模型的决策边界图如图3所示，模型正确分类出了测试数据集中全部 300 个点。

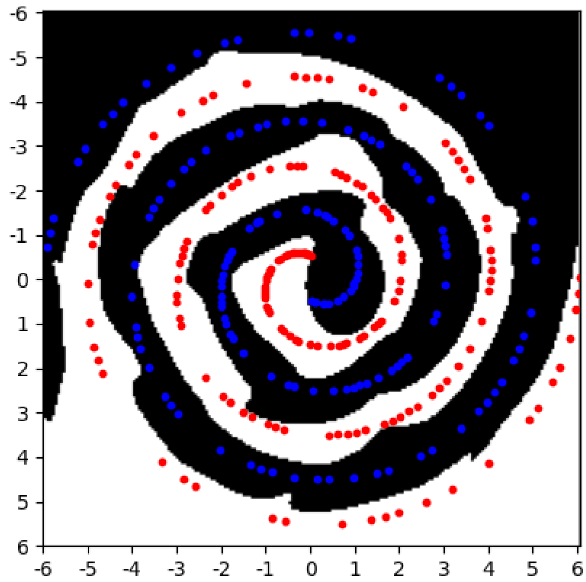


图 3: MLQP 模型 2000 轮训练后决策边界图

训练过程中模型的损失函数值与准确率变化曲线如图4所示，图中黑色折线代表损失函数值，红色折线代表模型对训练数据集预测的准确率，蓝色折线代表模型对测试数据集预测的准确率。

我也保存了训练过程中的模型，并分别绘制出其决策边界，如图5所示。

训练过程中，模型的 ROC 曲线如图6所示，模型的 AUC 大约在 1250 轮时达到 1.0。

由于均方差（MSE）损失函数与 sigmoid 函数的组合会导致模型输出值在 0、1 附近时梯度趋于 0，因此 MSE 函数并不是分类模型的理想损失函数。基于此，上述的模型训练过程中我使用了交叉熵（CrossEntropy）损失函数，该损失函数表达式为

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -[y_i \cdot log(p_i) + (1-y_i)\cdot log(1-p_i)]$$

交叉熵函数与 sigmoid 函数组合可以获得较为理想的梯度，有助于提高模型参数更新效率。

作为对比，我也使用了 MSE 函数在其他条件相同的情况下训练了相同的模型，训练过程中模型的 ROC 曲线如图7所示，相比使用交叉熵训练的曲线图6，MSE 的训练效率明显低于交叉熵，且模型的 AUC 最终未在 2000 轮训练时达到 1.0。

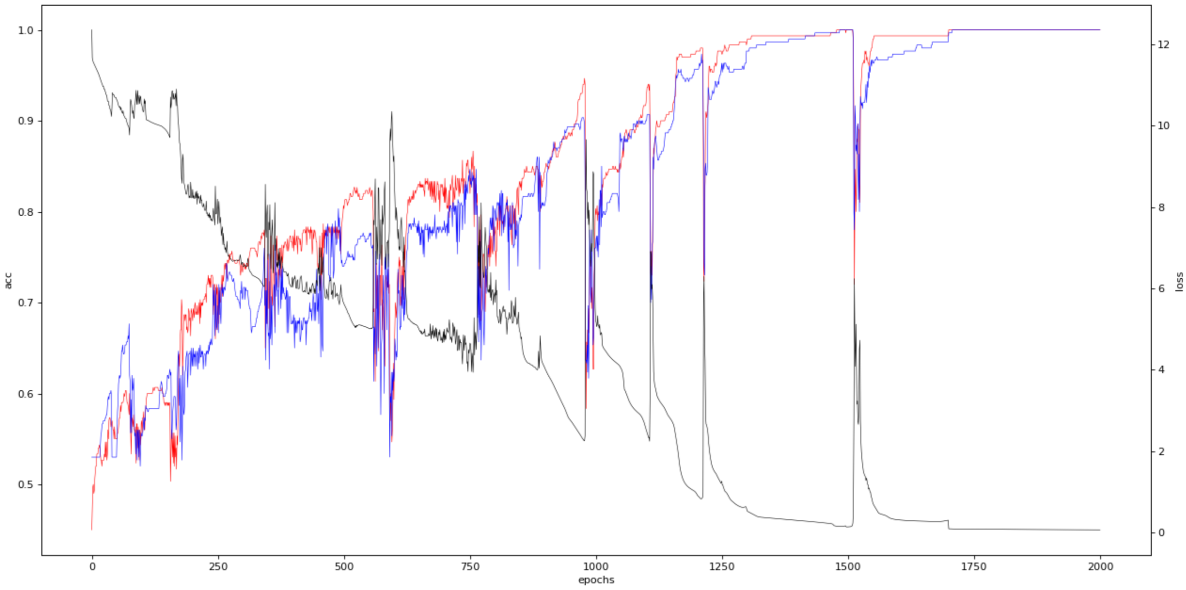


图 4: MLQP 模型训练过程中损失函数值与准确率曲线

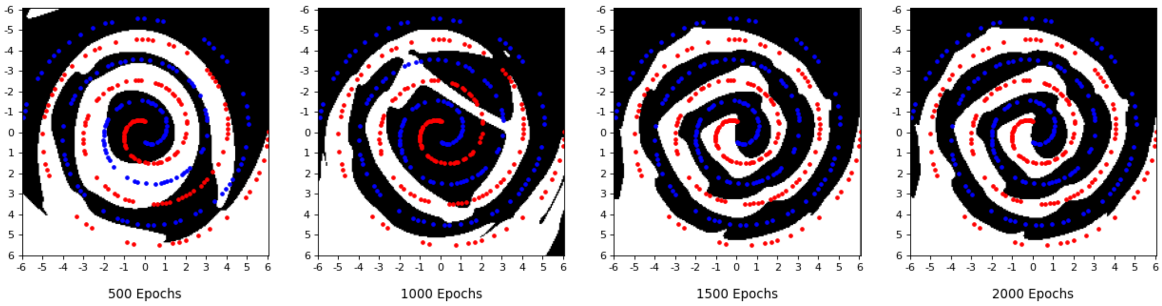


图 5: MLQP 模型训练过程中的决策边界变化

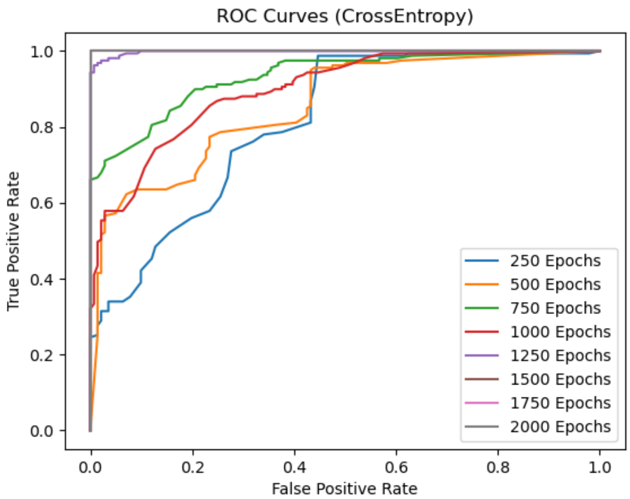


图 6: 训练过程中模型的 ROC 曲线（交叉熵）

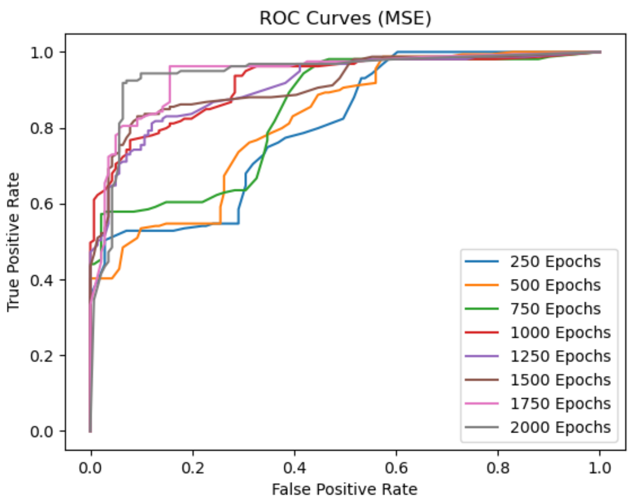


图 7: 训练过程中模型的 ROC 曲线（均方差）

除此之外，我使用了 Xavier 初始化、引入了动量（Momentum）提高训练效率。

**题目 3.** 分别使用先验知识和随机将分类问题拆分为 4 或 9 个子问题，并对每个子问题分别训练 MLQP 模型并构造 min-max 模型，比较不同模型的训练效率与决策边界。

我首先将两类数据点分别以 X 轴为界将数据点各自分为两类，然后两两组合组成四组训练数据，分别放入四个模型进行训练，然后使用 min-max 方法组合四个模型。

先验拆分模型的决策边界如图8所示，每个子模型的决策边界如图9所示，子模型训练过程的损失函数、准确率变化如图10所示。

从整体结果来看，先验拆分的 min-max 模型在大约 1000 轮训练后已经可以分辨出绝大多数数据点，且决策边界相当平滑，不足之处在于模型对 X 轴（即数据集拆分边缘）附近的数据点的分类结果不够精确。

从图9中可以看出，第 2、3 子模型只是简单上下分类，因此大部分分类工作是由 1、4 模型完成的。

从图10中可以看出 1、4 子模型训练速度明显慢于 2、3 子模型。整体而言，在 400 轮训练时已经可以得到较为理想的结果。

此后，我又按照随机顺序拆分训练集，并尽可能保证训练数据集的平衡，分别训练四个模型。与我最初的预期不同，随机拆分训练 2000 轮之后的结果仍略差于先验拆分 1000 轮训练后的结果，在测试训练集上的准确率大约为 93%。

对于随机拆分数据集的 min-max 模型，决策边界如图11所示，每个子模型的决策边界如图12所示，子模型训练过程的损失函数、准确率变化如图13所示。

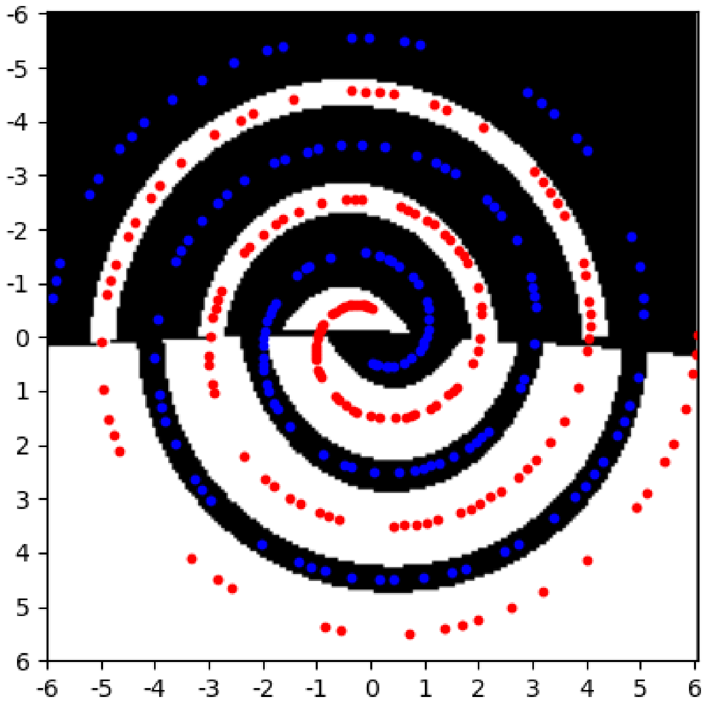


图 8: 先验拆分 min-max 模型决策边界

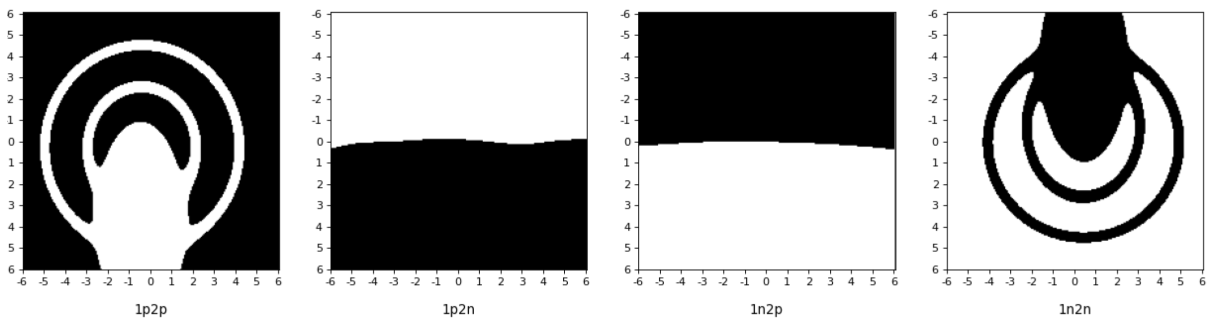


图 9: 先验拆分 min-max 每个子模型的决策边界

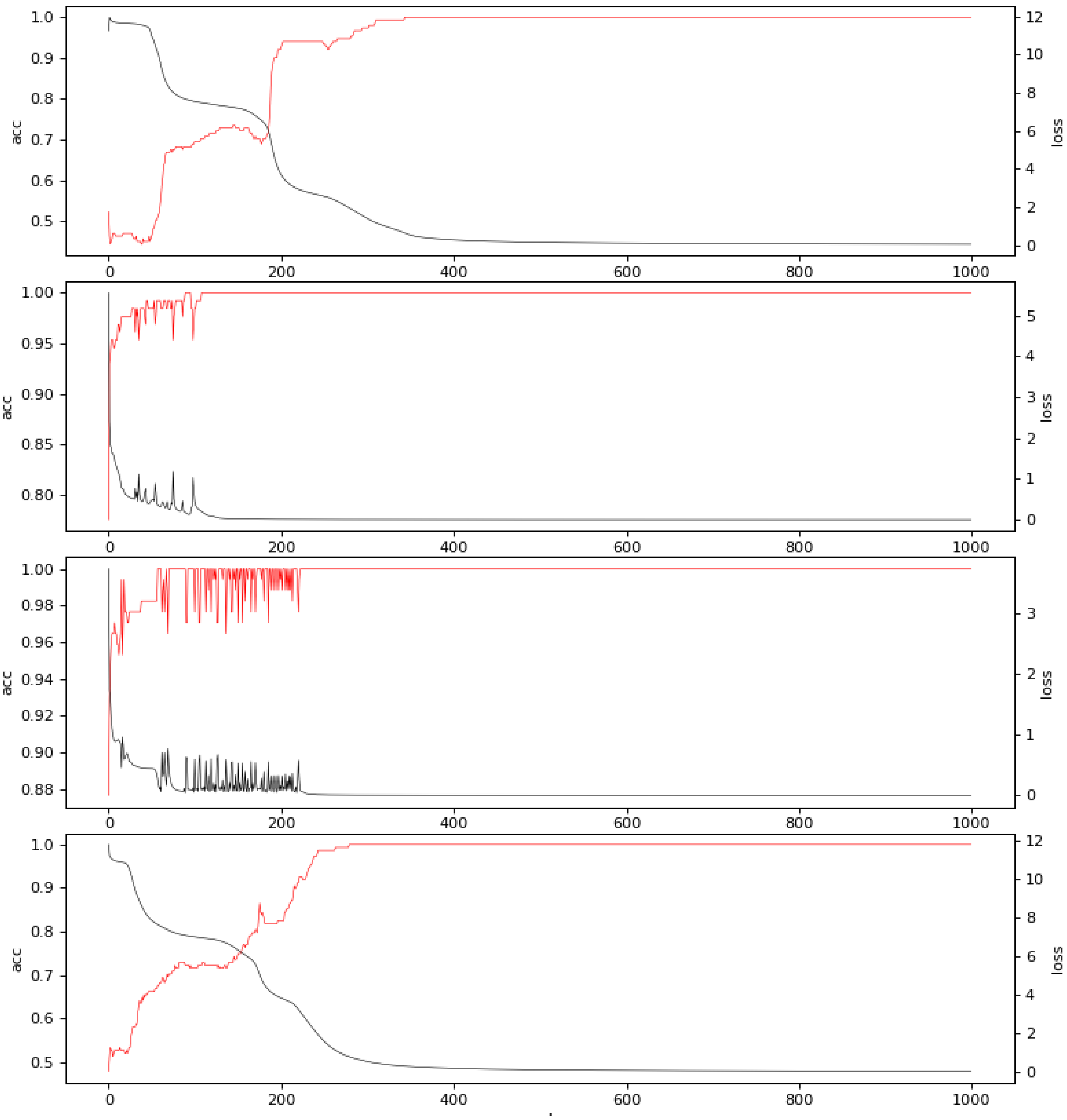


图 10: 先验拆分 min-max 每个子模型的损失函数、准确率变化

从图12中可以看出，随机拆分训练集中每个子图均具有一定程度的特征，但所有子图组合而成的决策边界图（图11）并不能足够完整地复现整体数据集特征。

根据实验结果，我得出结论，在当前问题下，使用先验知识将数据集按照坐标轴拆分可以在更短的训练时间内获得更好的训练结果。

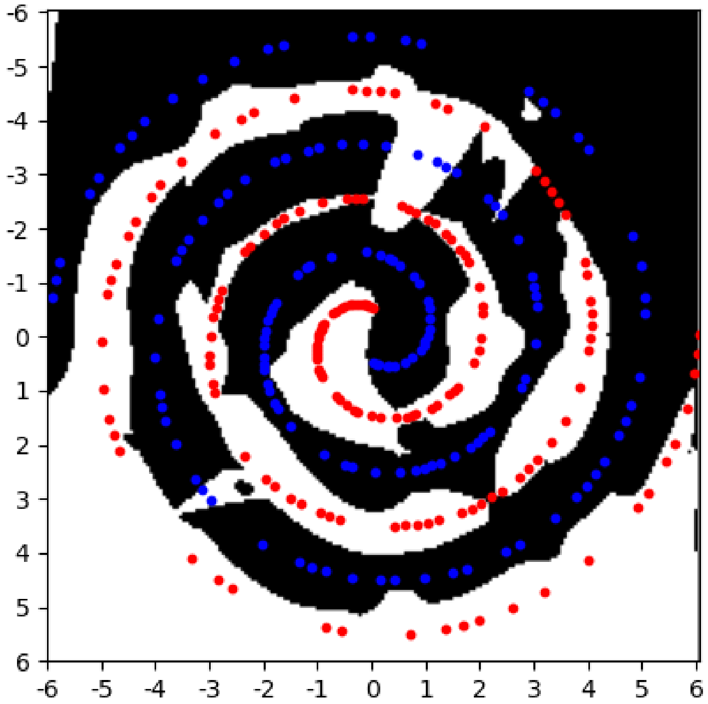

图 11: 随机拆分 min-max 模型决策边界

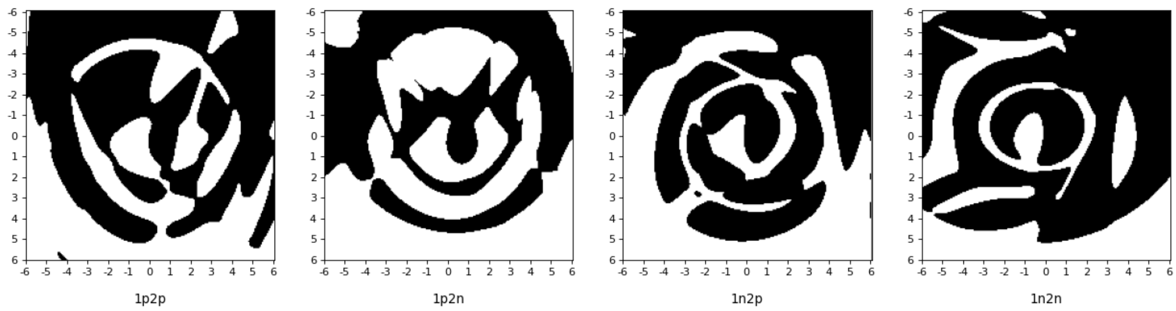


图 12: 随机拆分 min-max 每个子模型的决策边界

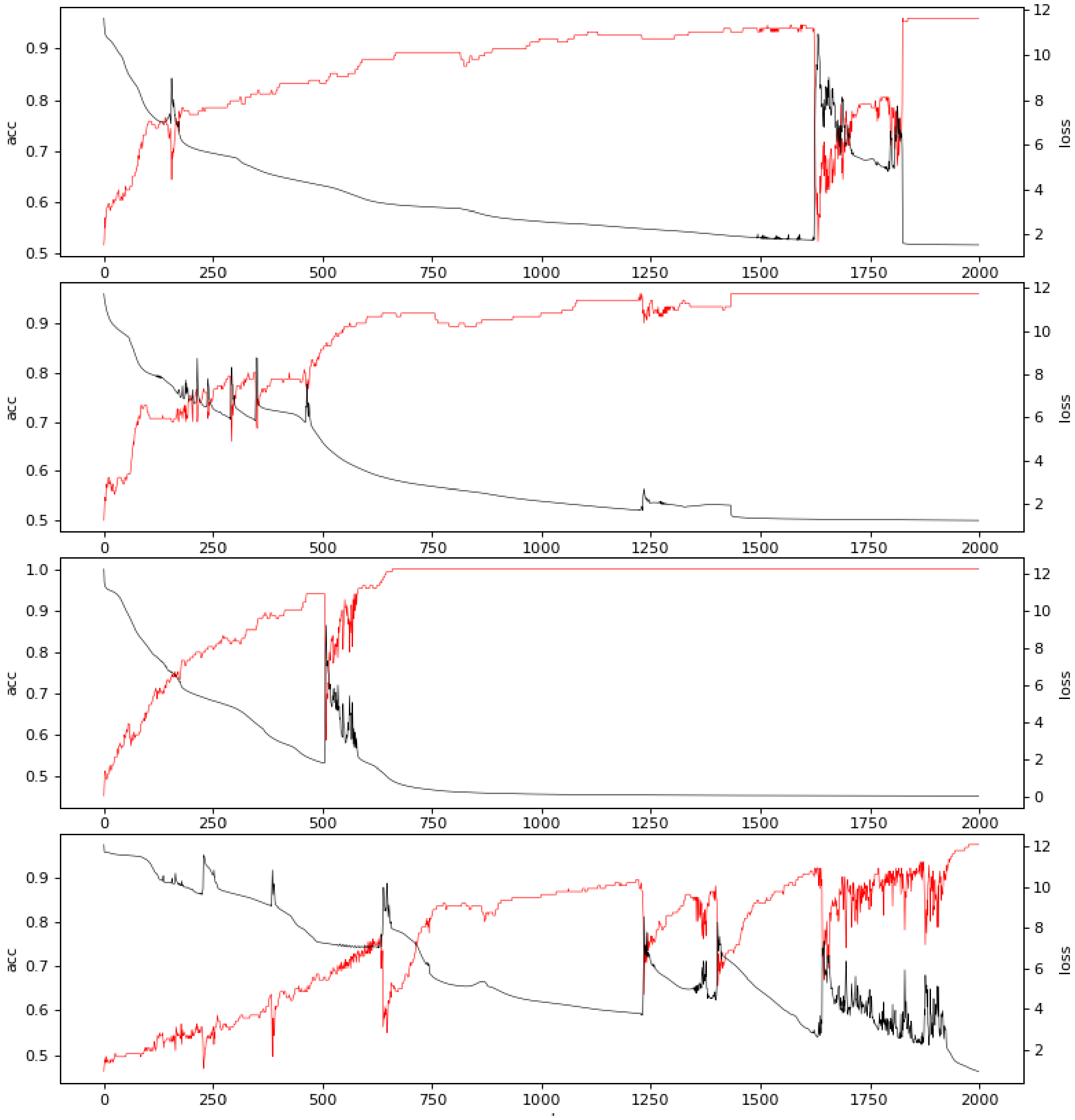


图 13: 随机拆分 min-max 每个子模型的损失函数、准确率变化