Neural Network Theory and Applications | Assignment#1

Shangyu Liu (020033910052)

April 7, 2021

1 MLQP Back-propagation

1.1 Problem Definition

Suppose the output of each neuron in a multi-layer perceptron is:

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

where both u_{kji} and v_{kji} are the weights connecting the i^{th} unit in the layer $k-1$ to the j^{th} unit in the layer k, b_{kj} is the bias of the j^{th} unit in the layer k, N_k is the number of units if $1 \le k \le M$ and $f(\cdot)$ is the sigmoidal activation function. Please derive the back-propagation algorithms for the multi-layer quadratic perceptron (MLQP) in both online learning and batch learning ways.

1.2 Problem Solution

As there is no limitation on application scenario in the problem definition, we assume the MLQP model is used for classification and the loss function for each training sample could be defined as cross entropy over softmax outputs:

$$\mathcal{L} = -\sum_{j=0}^{N_M} d_j \log x'_{Mj},\ \ x'_{Mj} = \frac{\exp(x_{Mj})}{\sum_l \exp(x_{Ml})}$$

where the one-hot vector d denotes the ground truth distribution and each element d_j is the label corresponding to j^{th} unit of the output layer.

Let G denote the right classification class, i.e. $d_{j=G} = 1$. Then according to whether the output x_{Mj} is corresponding to the right class, we can compute the gradient of $\mathcal{L}$ with respect to the outputs of the last layer by different situation.

When $j = G$, the partial derivative of $\mathcal{L}$ with respect to the output of j^{th} unit in the last layer M is

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x_{Mj}} &= \frac{\partial (d_G \log x'_{MG})}{\partial x_{Mj}} \\ &= -\frac{1}{x'_{MG}}\frac{\partial x'_{MG}}{\partial x_{MG}} \\ &= -\frac{1}{x'_{MG}}\left[\frac{\exp(x_{MG})\sum_l \exp(x_{Ml})}{(\sum_l \exp(x_{Ml}))^2}\right. \\ &\left.-\frac{\exp^2(x_{MG})}{(\sum_l \exp(x_{Ml}))^2}\right] \\ &= -\frac{1}{x'_{MG}}(x'_{MG} - x'^2_{MG}) \\ &= x'_{MG} - 1\end{aligned}$$

When $j \neq G$, the partial derivative of $\mathcal{L}$ with respect to the output of j^{th} unit in the last layer M is

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial x_{Mj}} &= \frac{\partial (d_G \log x'_{MG})}{\partial x_{Mj}} \\ &= -\frac{1}{x'_{MG}}\frac{\partial x'_{MG}}{\partial x_{Mj}} \\ &= -\frac{1}{x'_{MG}}\frac{-\exp(x_{Mj})\exp(x_{MG})}{\sum_l \exp(x_{Ml})^2} \\ &= -\frac{1}{x'_{MG}} \cdot (-x'_{Mj}x'_{MG}) \\ &= x'_{Mj} - 0\end{aligned}$$

We can combine these two cases together and have a general formulation

$$\frac{\partial \mathcal{L}}{\partial x_{Mj}} = x'_{Mj} - d_j$$

To compute the derivative of sigmoid function $f(\cdot)$, we define $s(x) = 1/f(x) = 1 + \exp(-x)$. Then we have

$$\begin{aligned}\frac{d}{dx}s(x) &= -\exp(-x) = 1 - s(x)\\ \frac{d}{dx}f(x) &= -\frac{s'(x)}{s^2(x)} = \frac{s(x)-1}{s^2(x)}\\ &= (\frac{1}{f(x)} - 1)f^2(x)\\ &= (1-f(x))f(x)\end{aligned}$$

Here we calculate the partial derivative of the output x with respect to the weight u, v and bias b in each unit.

$$\begin{aligned}\frac{\partial x_{kj}}{\partial u_{kji}} &= \frac{dx_{kj}}{dnet_{kj}}\frac{\partial net_{kj}}{\partial u_{kji}}\\ &= \frac{df(net_{kj})}{dnet_{kj}}\frac{\partial net_{kj}}{\partial u_{kji}}\\ &= (1-f(net_{kj}))f(net_{kj})x^2_{k-1,i}\end{aligned}$$

$$\begin{aligned}\frac{\partial x_{kj}}{\partial v_{kji}} &= \frac{dx_{kj}}{dnet_{kj}}\frac{\partial net_{kj}}{\partial v_{kji}}\\ &= \frac{df(net_{kj})}{dnet_{kj}}\frac{\partial net_{kj}}{\partial v_{kji}}\\ &= (1-f(net_{kj}))f(net_{kj})x_{k-1,i}\end{aligned}$$

$$\begin{aligned}\frac{\partial x_{kj}}{\partial b_{kj}} &= \frac{dx_{kj}}{dnet_{kj}}\frac{\partial net_{kj}}{\partial b_{kj}}\\ &= \frac{df(net_{kj})}{dnet_{kj}}\frac{\partial net_{kj}}{\partial b_{kj}}\\ &= (1-f(net_{kj}))f(net_{kj})\end{aligned}$$

According to the chain rule, we can derive the partial derivative of the loss function $\mathcal{L}$ with respect to the weight u and v, which connect the i^{th} unit of layer $k-1$ and the j^{th} unit of layer k.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial u_{kji}} = &\sum_{p_M=1}^{N_M}\frac{\partial \mathcal{L}}{\partial x_{Mp_M}}\sum_{p_{M-1}=0}^{N_{M-1}}\frac{\partial x_{M,p_M}}{\partial x_{M-1,p_{M-1}}}\cdots\\ &\sum_{p_{k+1}=0}^{N_{k+1}}\frac{\partial x_{k+2,p_{k+2}}}{\partial x_{k+1,p_{k+1}}}\frac{\partial x_{k+1,p_{k+1}}}{\partial x_{kj}}\frac{\partial x_{kj}}{\partial u_{kji}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial v_{kji}} = &\sum_{p_M=1}^{N_M}\frac{\partial \mathcal{L}}{\partial x_{Mp_M}}\sum_{p_{M-1}=0}^{N_{M-1}}\frac{\partial x_{M,p_M}}{\partial x_{M-1,p_{M-1}}}\cdots\\ &\sum_{p_{k+1}=0}^{N_{k+1}}\frac{\partial x_{k+2,p_{k+2}}}{\partial x_{k+1,p_{k+1}}}\frac{\partial x_{k+1,p_{k+1}}}{\partial x_{kj}}\frac{\partial x_{kj}}{\partial v_{kji}}\end{aligned}$$

To simplify the expression, here we propose a recursive definition called local gradient. We define the partial derivative of the loss function $\mathcal{L}$ with respect to the output x of the j^{th} unit in layer k as δ_{kj}. Then we have the recursion relationship

$$\delta_{kj} = \frac{\partial \mathcal{L}}{\partial x_{kj}} = \begin{cases}\dfrac{\exp(x_{Mj})}{\sum_l \exp(x_{Ml})} - d_j, & k = M\\ \displaystyle\sum_p \delta_{k+1,p}\frac{\partial x_{k+1,p}}{\partial x_{kj}}, & 1 \le k < M\end{cases}$$

And we can calculate $\partial x_{k+1,p}/\partial x_{k,j}$ as

$$\begin{aligned}\frac{\partial x_{k+1,p}}{\partial x_{k,j}} &= \frac{dx_{k+1,p}}{dnet_{k+1,p}}\frac{\partial net_{k+1,p}}{\partial x_{k,j}}\\ &= \frac{df(net_{k+1,p})}{dnet_{k+1,p}}\frac{\partial net_{k+1,p}}{\partial x_{k,j}}\\ &= (1-f(net_{k+1,p}))f(net_{k+1,p})\cdot\\ &\quad(2u_{k+1,pj}x_{k,j} + v_{k+1,pj})\end{aligned}$$

Then the partial derivative of the loss function $\mathcal{L}$ with respect to the weight u, v and bias b can be expressed as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial u_{kji}} &= \delta_{kj}\frac{\partial x_{kj}}{\partial u_{kji}}\\ &= \delta_{kj}(1-f(net_{kj}))f(net_{kj})x^2_{k-1,i}\\ \frac{\partial \mathcal{L}}{\partial v_{kji}} &= \delta_{kj}\frac{\partial x_{kj}}{\partial v_{kji}}\\ &= \delta_{kj}(1-f(net_{kj}))f(net_{kj})x_{k-1,i}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_{kj}} &= \delta_{kj}\frac{\partial x_{kj}}{\partial b_{kj}} \\ &= \delta_{kj}(1-f(net_{kj}))f(net_{kj})\end{aligned}$$

For the online learning mode, stochastic gradient descend method is used and the weight w, v and bias b are updated after presenting each random training sample.

$$\begin{aligned}w_{kji}^{t+1} &= w_{kji}^{t} - \eta\frac{\partial \mathcal{L}}{\partial u_{kji}} \\ v_{kji}^{t+1} &= v_{kji}^{t} - \eta\frac{\partial \mathcal{L}}{\partial v_{kji}} \\ b_{kj}^{t+1} &= b_{kj}^{t} - \eta\frac{\partial \mathcal{L}}{\partial b_{kj}}\end{aligned}$$

where t denotes the iteration step and η are defined as learning rate.

For the batch learning mode, batch gradient descend method is adopted and the weight w, v and bias b are updated after each epoch only. Then the total loss $\mathcal{L}'$ could be defined as the average of $\mathcal{L}$, i.e. $\mathcal{L}' = 1/N\sum_{s=1}^{S}\mathcal{L}$, where S is the total number of training samples. And the back propagation then follows

$$\begin{aligned}w_{kji}^{t+1} &= w_{kji}^{t} - \frac{\eta}{N}\sum_{s=1}^{S}\frac{\partial \mathcal{L}}{\partial u_{kji}} \\ v_{kji}^{t+1} &= v_{kji}^{t} - \frac{\eta}{N}\sum_{s=1}^{S}\frac{\partial \mathcal{L}}{\partial v_{kji}} \\ b_{kj}^{t+1} &= b_{kj}^{t} - \frac{\eta}{N}\sum_{s=1}^{S}\frac{\partial \mathcal{L}}{\partial b_{kj}}\end{aligned}$$

2 Two spirals classification

2.1 Problem definition

Please implement an on-line BP algorithm for MLQP (you can use any programming language), train a MLQP with one hidden layer to classify two spirals problem, and compare the training time and decision boundaries at three different learning rates.

2.2 Problem solution

Compared with other simple binary classification cases, the two spirals problem is more difficult with sophisticated and irregular decision boundary. To solve this problem, we adopted a MLQP model with only one hidden layer. In detail, We implemented the model with programming language Python and neural network computing framework TensorFlow version 1.15. In this section, we will first discuss the influence of learning rate selection on the convergence time and decision boundaries. Then we will introduce the dimension selection of hidden layer. And finally we will introduce the normalization problem we have come across.

2.2.1 Learning rate selection

Learning rate is one of the most importance hyper-parameters and it could greatly affect the performance of neural network model. The best choice of learning rate is often relevant to the network structure and the optimizer adopted. Here the network is defined to have an input layer of 2 units, a hidden layer of 32 units and an output layer of 1 unit. We adopt the famous Adam optimizer to minimize our loss function. We have done extensive experiments to search for the best learning rate. Here we just present three cases among them for brief discussion.

Converge time. As denoted by the green line in Figure 1, when we use a excessively large learning rate such as 0.03, the loss and accuracy violently fluctuate and the model could hardly converge before 5000 epochs. This is because a large step size in the gradient descent algorithm would let the optimizer repeatedly jump over the optimal position in the parameter space. Therefore, the model is not fully optimized and the accuracy remains at a low level.

As denoted by the blue line in Figure 1,

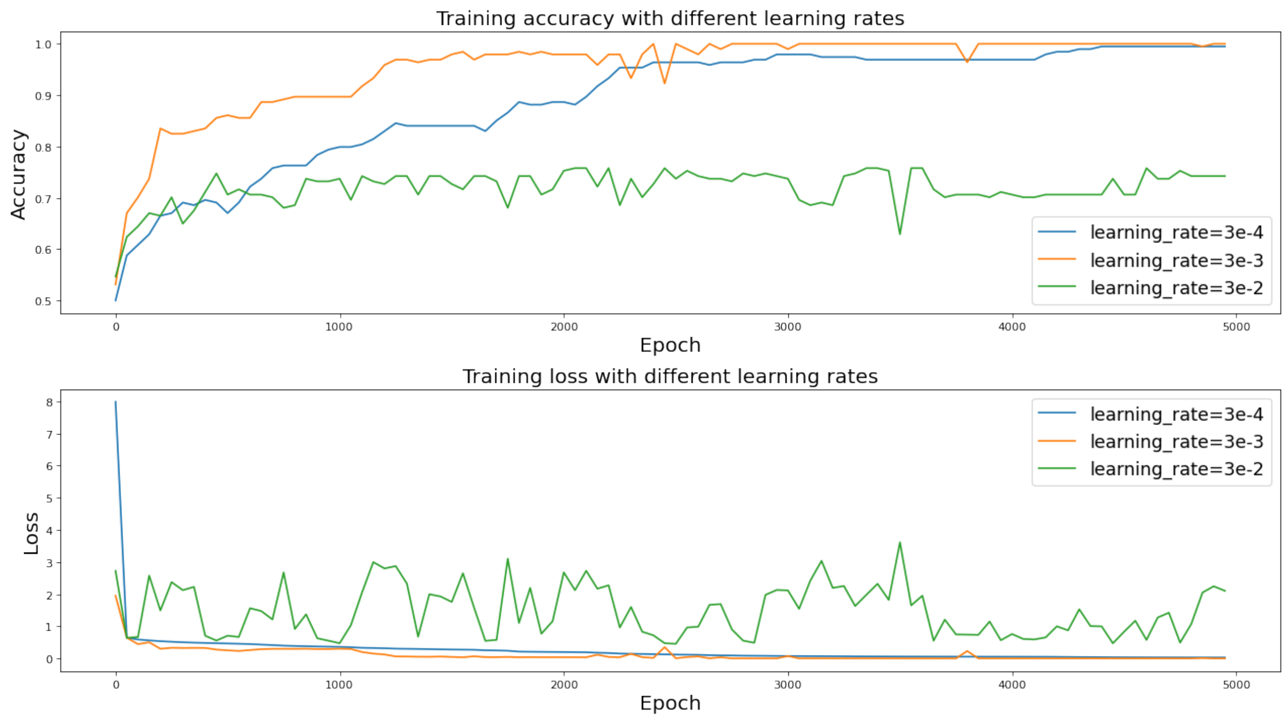


Figure 1: Training accuracy and loss with different learning rates

when we use a too small learning rate such as 0.0003, the training is more stable but yet it takes quite a long time (about 3000 epochs) to converge. This is because small step size is much too careful at the beginning of training and obviously it takes more steps to reach the final optimal point. As small learning rate may lead the model to a local optimal point, the general accuracy is similarly low.

As shown by the orange line in Figure 1, we choose a moderate learning rate 0.003 after extensive trials. It turns out to be the best, fast converging before 1500 epochs and the model achieves the highest accuracy score.

Decision boundaries. As shown in Figure 2, the shapes of decision boundaries vary among different learning rates. In detail, the larger the learning rate is, the smoother the boundary would be. This is probably because when the learning rate is large, there is more chance to jump out of local optimality and the parameters we finally get is more representative.

2.2.2 Hidden layer dimension

The dimension of hidden layer is another significant hyper-parameter. The different choices of structure could greatly affect the inference ability and generalization ability of the network. Here we select three cases where there are 8, 32, and 64 units in the hidden layer respectively to show the relationship between them. As shown in Figure 4, when the hidden layer is only 8 units wide, the network has limited representation ability. And when there are 64 units in the hidden layer, although the accuracy mildly increases (shown in Figure 3), the decision boundary is sophisticated, which suggests the model is overfit. As a result, we choose a 32-unit wide hidden layer.

2.2.3 Normalization problem

During the experiment, we observed a counter-intuitive phenomenon. When we normalize the position features of the training points to $[0, 1]$, the classification ability of the MLQP model remains quite low (with accuracy no more than 0.58). As we know, normalization defined as

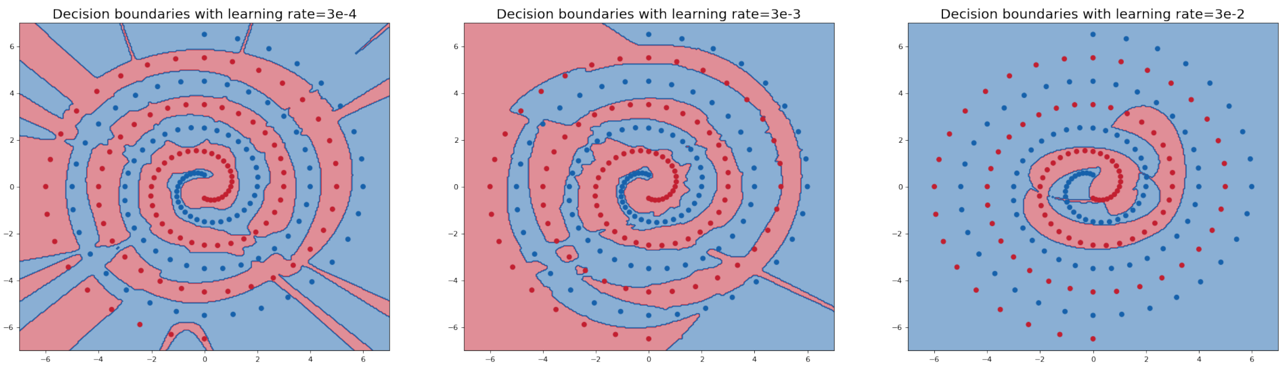


Figure 2: Decision boundaries with different learning rates

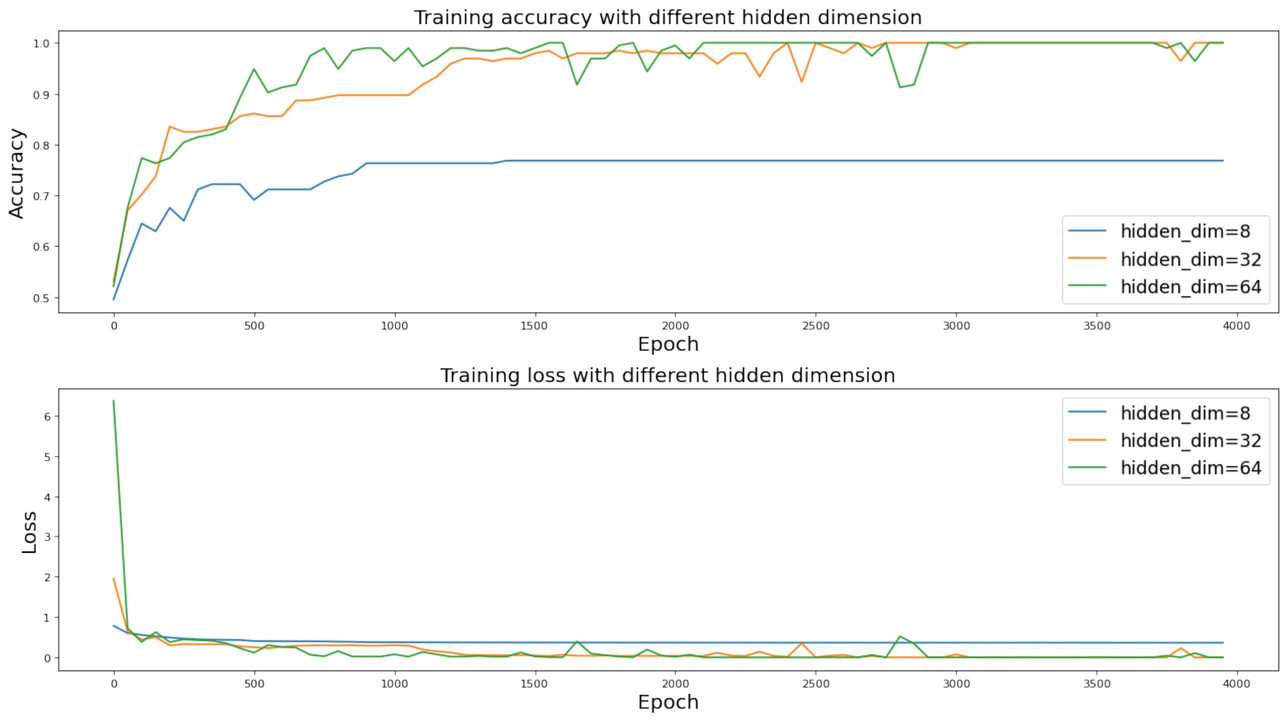


Figure 3: Training accuracy and loss with different hidden dimension

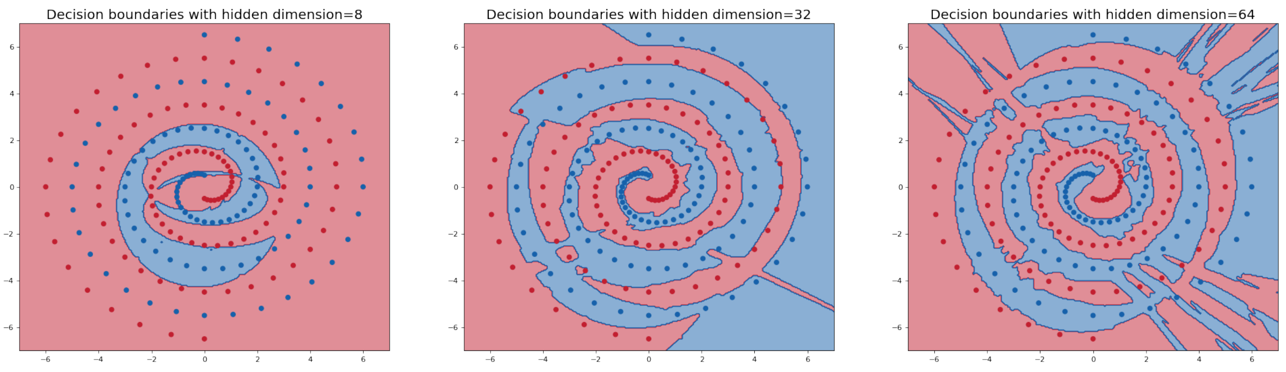


Figure 4: Decision boundaries with different hidden dimension

follows, is a kind of linear transformation which will not affect the performance of network.

$$X_i' = \frac{X_i - \min_i\{X_i\}}{\max_i\{X_i\} - \min_i\{X_i\}}$$

However, the linear transformation actually prevent us from making good use of the characteristic of data distribution. The input data points were two spirals with a center at coordinate origin. But after transformation, the center has been moved to $[0.5, 0.5]$. As our weights are all initialized with guassion distribution with expectation of 0, the problem becomes more difficult. As we do not change the network structure, such as broadening the hidden layer or adding more layers, the network is not able to deal with the problem. We guess if we just do a standardization defined as follows, the performance would be better.

$$X_i' = \frac{X_i - X_m}{\sqrt{\frac{1}{2N}\sum_{i=1}^{N}(X_i - X_m)^2}}$$

$$X_m = \frac{1}{N}\sum_{i=1}^{N} X_i$$

3 Min-max modular network

3.1 Problem definition

i Divide the two spirals problem into four sub-problems randomly and with prior knowledge, respectively

ii Train MLQP on these sub-problems and construct two min-max modular networks

iii Compare training time and decision boundaries of the above two min-max modular networks

3.2 Problem solution

To divide the problem, we first divide the training dataset into two subsets randomly and with prior knowledge, respectively. Then we select the positive samples of the first subset and the negative samples of the second subset to form a new sub-problem. Similarly we select the negative samples of the first subset and the positive samples of the second subset to form a new sub-problem. Finally we will have four sub-problems. We construct four MLQP model with the same structure. There are 2 units in input layer, 1 unit in output layer and 32 units in hidden layer. To optimize the network, we select different learning rate empirically. We aggregate the results from the four networks and perform a min-max operation to obtain the final prediction.

3.2.1 Random division

We first divide the problem randomly. In practice, we shuffle the training set and then divide it into two equal parts at the first stage. Then we generate the other two sub-problems described above.

The accuracy and decision boundaries of the four sub-problems are shown in Figure 5(a). As the data distribution of sub-problems is irregular, the decision boundaries of them are worse compared with prior algorithm without problem division. We train each MLQP model for 3000 steps, but it seems they have not converged, which makes the boundaries worse. The result of the original problem is shown in Figure 6(a). In general, it is obvious that the final decision boundary is better than those of the sub-problems. However, the final accuracy is lower than the average accuracy of the sub-problems, which is due to the difficulty of the complete problem.

3.2.2 Designed division

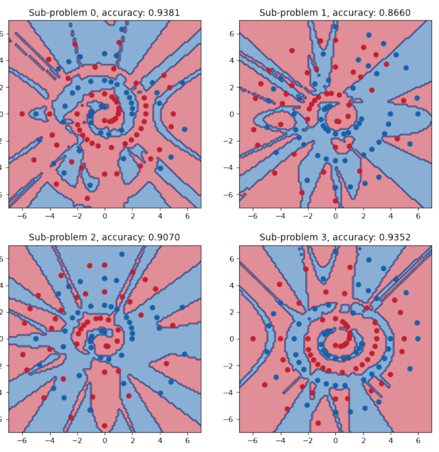
Here we divide the problem with prior knowledge. In detail, to make good use of the shape characteristics of the training data, we use the distance between each data point and the center of the two spiral as a value function.

$$f(X_i) = \|X_i - \frac{1}{N}\sum_{i=0}^{N} X_i\|_2$$

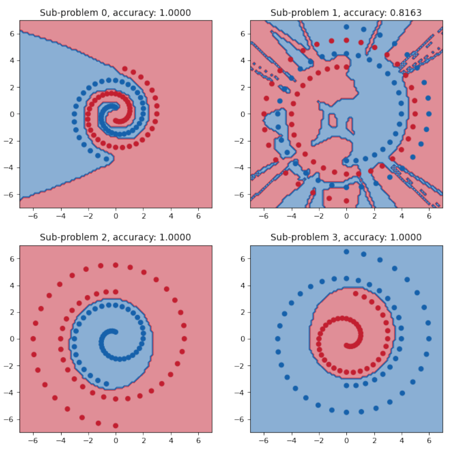
And we empirically define the threshold as 3.5 (about half of the radius). At the first division stage, points with value below the threshold form the first subset, while the others form the second subset. Then we generate the other two sub-problems described above.

As shown in figure 5(b), the decision boundaries of the four sub-problems except for sub-problem 1, are distinctly smoother than before. The bad performance of sub-problem 1 is due to its difficulty. We design to train the four models for same epochs, however, it may takes longer for a model to converge to a smooth boundary when the sub-problem is harder. The result of the original problem is shown in Figure 6(b). It's obvious that the decision boundary is smoother compared with random division method. Also we have just trained each model for 500 epochs which is much shorter than before.

In conclusion, dividing the problem with prior knowledge and selecting adaptive hyper-parameters for each sub-problem will lead to better performance.

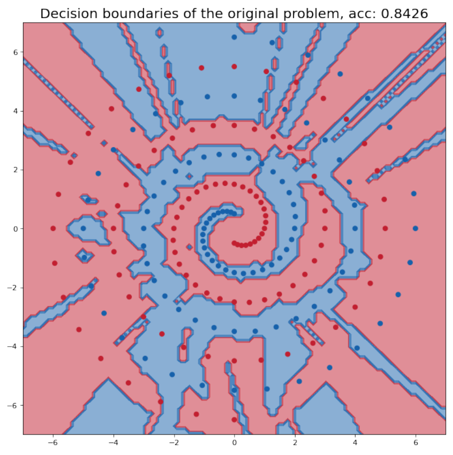


(a) Random division

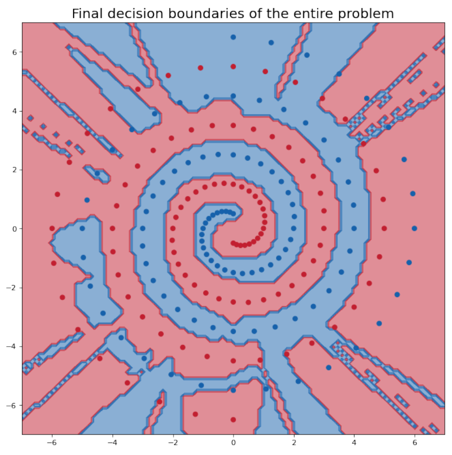


(b) Designed division

Figure 5: Decision boundaries and training accuracy of the four sub-problems



(a) Random division



(b) Designed division

Figure 6: Decision boundaries of the original problem