Assignment 1

Luo Zhiyi (0130339024)

Spring 2014

1 Introduction

This is the report of assignment1 for neural network design course. The problem is to solve the two-spiral classification problem by implementing a multi-layer quadratic perception(MLQP) network with one hidden layer. I implement both online learning and batch learning ways to train the MLQPs model with the training data. This report consists of derivation of the back-propagation algorithms for MLQPs, description of the implementation and the analysis of experiment result applying the model on the two-spiral problem.

2 Multi-layer Quadratic Perceptron

2.1 MLQPs Formulation

In this report, I use $u_{i,j}^{(l)}$ and $v_{i,j}^{(l)}$ to denote the weights connecting the ith unit in the layer l to the jth unit in the layer $l+1$, $b_i^{(l)}$ is the basis of the ith unit in the layer l. $N^{(l)}$ is the number of units in lth layer ($1 <= l <= M$), and f(.) is the sigmoid activate function:$f(z) = \frac{1}{1+e^{-z}}$ and $f^{'}(z) = f(z)(1-f(z))$. In fact, our MLQPs is an example of **feedforward** neural network. I'll introduce the notations and formulations in MLQPs through feedforward calculation. Figure 2 describes a brief structure of the implemented MLQPs. For the feedforward process, the computation that the MLQPs represents is given by:

$$\begin{aligned} z_j^{(l)} &= \sum_{i=1}^{N^{(l-1)}} (u_{ji}^{(l)})(a_i^{(l-1)})^2 + v_{ji}^{l} a_i^{(l-1)} + b_j^{(l)} \\ a_j^{(l)} &= f(z_j^{(l)}) \end{aligned} \tag{1}$$

2.2 Cost Function

Suppose we have a given training set, say $(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y(2)), ..., (\mathbf{x}^{(m)}, y(m))$. In detail, the $\mathbf{x}^{(i)}$ denotes the ith example's feature vector, contains the x-coordinate and y-coordinate information. Since the two-spiral classification is a binary classification problem, it implies that the data follows Bernoulli distribution, so I use the following function form to be the cost function for a single example$(\mathbf{x}^{(i)}, y^{(i)})$:

$$J(U, V, b; \mathbf{x}^{(i)}, y^{(i)}) = -(y^{(i)} log(h_\theta(\mathbf{x}^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(\mathbf{x}^{(i)})) \quad (2)$$

where the $h_\theta(\mathbf{x}^{(i)})$ denotes the MLQPs' output for example $(\mathbf{x}^{(i)}, y^{(i)})$. In binary classification problem, a simple interpretation of $h_\theta(\mathbf{x}^{(i)})$ is the probability of $y^{(i)}$ equals to 1, given the parameter θ and data $\mathbf{x}^{(i)}$. I also add the regularization term to avoid over-fitting problem. I set the weight decay parameter λ to be 0.0001 by experience. Given a training set of m examples, I then define the overall cost function to be

$$J(U, V, b) = [-\frac{1}{m} \sum_{i=1}^{m} J(U, V, b)] + \frac{\lambda}{2} \parallel U \parallel_2^2 + \frac{\lambda}{2} \parallel V \parallel_2^2 \quad (3)$$

Our goal is to minimize the $J(U, V, b)$ as a function of U, V and b. To train the MLQPs model, I will initialize each parameter to a small random value close to zero, and then apply an optimization algorithm such as batch gradient descent and online gradient descent.

2.3 Back-propagation algorithm

We will now describe the back propagation algorithm, which gives an efficient way to compute the partial derivatives when updating the model parameters. Intuitively, we have the following formulation of the partial derivatives calculation:

$$\begin{aligned} \frac{\partial J}{\partial U} &= \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial U} \\ \frac{\partial J}{\partial V} &= \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial V} \\ \frac{\partial J}{\partial b} &= \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial b} \end{aligned} \quad (4)$$

More formally, we do the following derivation for each unit(Here we only consider the parameter U, V and b are similar):

$$\frac{\partial J}{\partial u_{ij}^{(l)}} = \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial u_{ij}^{(l)}} \tag{5}$$

$$\frac{\partial J}{\partial z_i^{(l)}} = \sum_{j=1}^{N^{(l+1)}} \frac{\partial J}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \tag{6}$$

We can obtain the following formulation through single neural network unit calculation :

$$\frac{\partial z_j^{(l+1)}}{\partial a_i^{(l)}} = 2u_{ij^{(l)}} a_i^{(l)} + v_{ij}^{(l)} + v_{ij}^{(l)} a_i^{(l)} \tag{7}$$

$$\frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} = f(z_i^{(l)})(1 - f(z_i^{(l)})) \tag{8}$$

Let us use $\delta_i^{(l)}$ short for $\frac{\partial J}{\partial z_i^{(l)}}$. Then, we will get:

$$\delta_i^{(l)} = \sum_{j=1}^{N^{(l+1)}} \delta_j^{(l+1)} (2u_{ij^{(l)}} a_i^{(l)} + v_{ij}^{(l)} + v_{ij}^{(l)} a_i^{(l)}) f(z_i^{(l)})(1 - f(z_i^{(l)})) \tag{9}$$

So,

$$\frac{\partial J}{\partial u_{ij}^{(l)}} = \delta_i^{(l+1)} (a_j^{(l)})^2 \tag{10}$$

Similarly, we have:

$$\frac{\partial J}{\partial v_{ij}^{(l)}} = \delta_i^{(l+1)} (a_j^{(l)}) \tag{11}$$

$$\frac{\partial J}{\partial b_{ij}^{(l)}} = \delta_i^{(l+1)} \tag{12}$$

Now, we know how to calculate the gradient of model parameters U, V and b.

2.3.1 Batch Learning

Now, let's consider to train the neural network using batch gradient descent which updates the model parameters once considered the effect of all training examples. The pseudo-code of batch learning is given in Algorithm 1.
Though batch learning algorithm is stable, it takes much calculation for parameter gradients. We will discuss another technique known as online technique shortly.

Algorithm 1 batch learning method

repeat

for $i = 1$ to m,{

using m examples each iteration

$$\theta_j := \theta_j + \alpha \frac{\partial J}{\partial \theta_j} \quad \text{(for every } j\text{)}$$

}

until converge

2.3.2 Online Learning

Batch techniques which involve in processing the entire training set in one go can be computationally costly for large data sets. If the data set is sufficiently large, it may be worthwhile to use sequential algorithms, also known as online learning algorithms.
Now, let's consider to train the neural network using online gradient descent. We hope that the online algorithm will be faster than the batch algorithm in this assignment. Algorithm 2 shows the online learning method for updating gradients of model parameters.

Algorithm 2 online learning method

repeat

using one example each iteration

$$\theta_j := \theta_j + \alpha \frac{\partial J}{\partial \theta_j} \quad \text{(for every } j\text{)}$$

until converge

As we will see in the experiment section, online algorithm is much faster than the batch algorithm even on our small data set.

2.4 Advanced Optimization

There are other algorithms that are even more sophisticated than gradient descent. Advanced algorithms such as L-BFGS can often be much faster than gradient descent. In this assignment, I use a matlab package named **minFunc** including a well implemented L-BFGS algorithm.

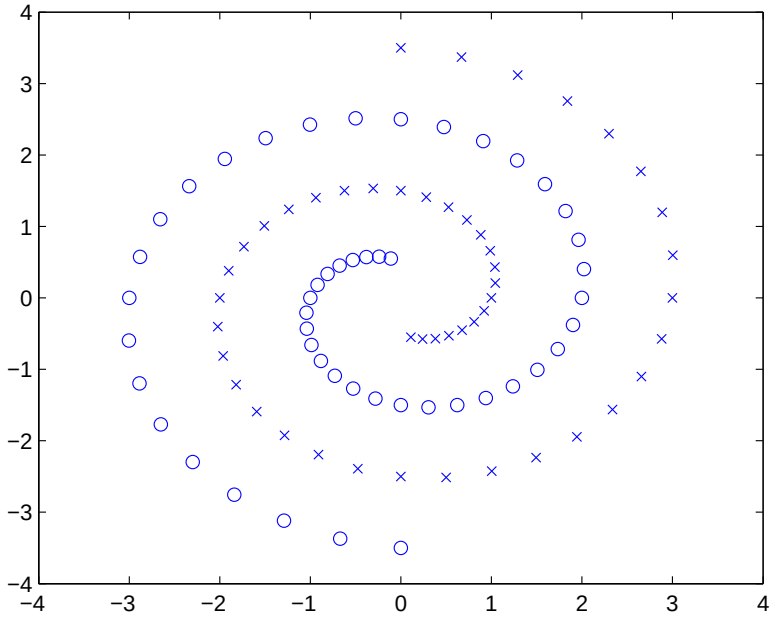

Figure 1: Training set examples plotting

3 Experiment

We say the problem is two-spiral classification. First, we plot the training set contained 96 examples on Figure 1.
The batch learning MLQPs model achieves 100 % on both training set and testing set examples. The figure 2 shows the experiment result. Figure 3 illustrates the tends that cost function changed with iterations.
The precision of the result on test set examples using batch learning MLQPs model is 100 %. We can see that Figure 2a) and Figure 2b) exactly match well on each other.
Figure 3 separately illustrates two different classification regions of MLQPs model for this training data set in red color and blue color.
We also do the experiment to compare the two algorithms (batch algorithm and online algorithm) for running time. Using the simple gradient descent optimization algorithm, the batch learning algorithm costs 19.227 seconds, while the online learning algorithm costs about 5.522 seconds. Both of the batch learning algorithm and the online learning algorithm have 100 % precision. However, batch algorithm speeds up to 2.695 seconds after adopting the **advanced optimization** algorithm L-BFGS.

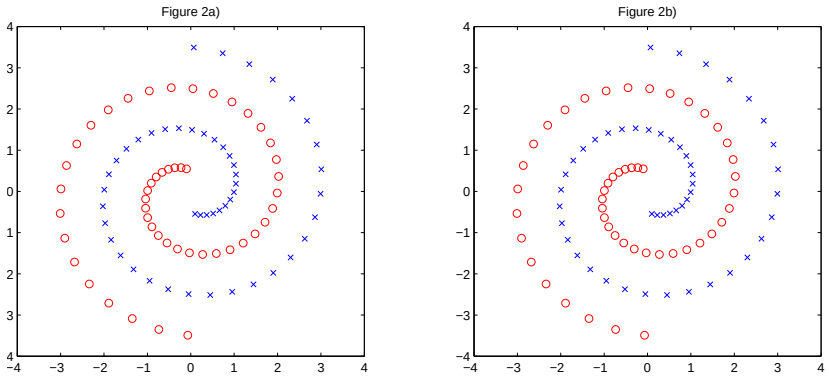

Figure 2: Batch learning result on test set. Figure 2a) plot the test set examples; Figure 2b) plot the prediction result on the test set examples using batch learning MLQPs model

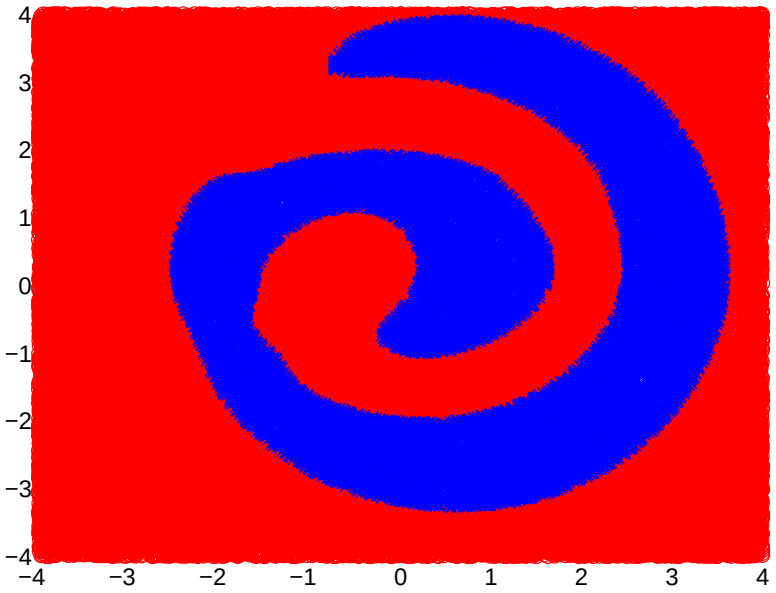

Figure 3: Decision Region, the red region and blue region represents for two different classes region in the problem.

4 Conclusion

I implement a three-layer neural network with back propagation algorithm using both batch and online technique. Now, I am familiar with the detailed structure of MLQPs models. That's what I have learned from this assignment. I'm very interested in neural network now. I plan to take my effort in deep learning technique in my further study. Thanks for Prof. Lu and our TA.