HW-3: Neural Network

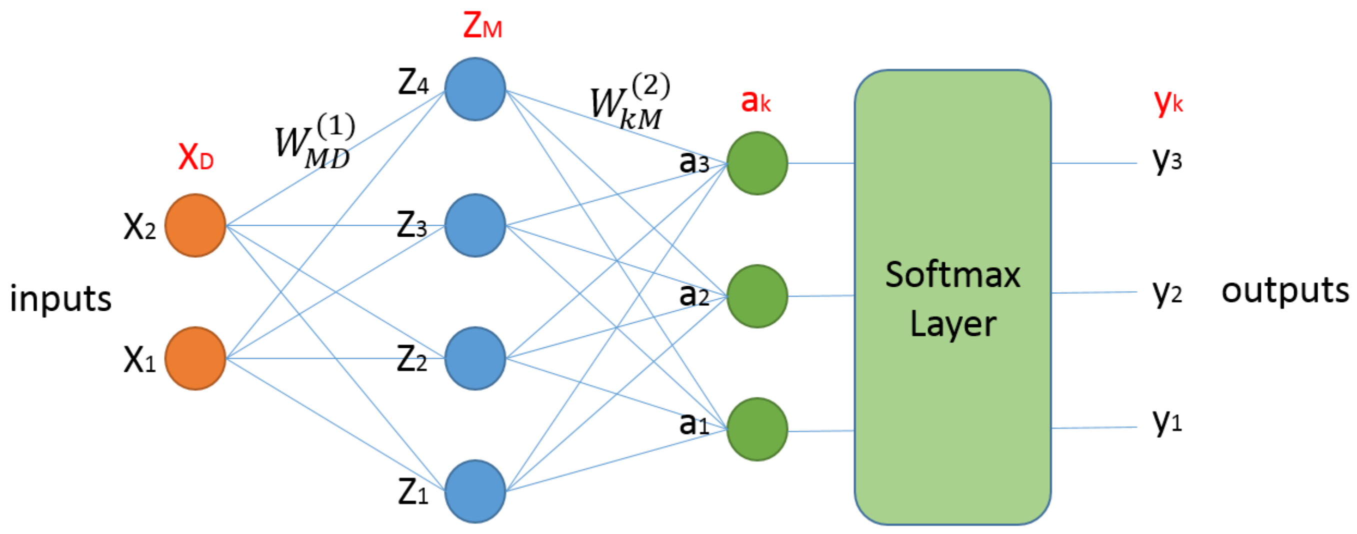
Final Report

- Abstract

First, I perform principal components analysis to reduce the number of dimension of data sample. Then use one hidden layer neural network and two hidden layer neural network with different activation functions, such as sigmoid function and rectified function by using stochastic gradient descent algorithm. Besides, we make the comparison between stochastic gradient descent and minibatch gradient descent.

- Introdction
  - Neural Network

    Our goal is to extend the model from the linear models for regression and classification, which based on linear combinations of fixed nonlinear basis functions. The Neural Network model are presented below.



➢ Feed-forward Network Functions

First, we construct M linear combinations of the input variables x_1, x_2,..., x_D in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \text{ j=1,…,M}$$

We should refer to the parameters $w_{ji}^{(1)}$ as weights and the parameters

$w_{j0}^{(1)}$ as biases. The quatities a_j are known as activations. Each of them is then transformed using a differentiable, nonlinear function such as sigmoid function or rectified function.

$$z_j = h(a_j)$$

These value are again done with the linear combination to give an output unit activations.

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \text{ k=1,…,K}$$

Where K is the total number of outputs. This linear combination corresponds to the second layer of this Neural Network and again $w_{k0}^{(2)}$ are bias parameters.

Finally, the output unit activations are trasformed using appropriate function to give a set of network outputs y_k. In our case of multiclass classification, we use softmax function to generate network outputs.

$$y_k = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

- Error Backpropagation

If we want to find an efficient technique for evaluating the Gradient of an error function, we can achive that by using a local message passing scheme in which information is sent alternatively forwards and backwards through the network and is known as error backpropation.

For a paticular input pattern n, error function takes the form

$$E_n(w) = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - t_{nk})^2$$

And we introduce a useful notation

$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_{nk} - t_{nk}$$

Now consider the evaluation of the derivative of E_n with respect to a weight w_{kj}. Then we note that $E_n$depends on the weight w_{kj} through the summed input a_k to unit k. We can therefore apply the chain rule for partial derivative to give as follows

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

On the other hand, another notation in the previous layer δ_j defined as follow.

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$
$$= h'(a_j)\sum_k w_{kj}\,\delta_k$$

Then we note that $E_n$depends on the weight w_{ji} through the summed input a_j to unit j. We can therefore apply the chain rule for partial derivative to give as follows

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}^{(1)}} = \delta_j z_i$$

In the end, we can use the simplest approach to using gradient information to updata in a small step in the direction of the negative gradient as the equation below.

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)})$$

- Sigmoid Function

  It is a mathematical function with an ‘S’ shaped curve. It refers to one case of the logistic function and defined by the formula

$$\mathrm{s(x)}=\frac{1}{1+e^{-x}}$$

- Rectfied Function

  When the x increase, the gradient of the sigmoid function vanishs. However, Rectified function does not have this problem and have gradient when the x is increase. The Rectified function defined as below.

$$\mathrm{R(x)} = \begin{cases} x, if\ x > 0 \\ 0, if\ x \le 0 \end{cases}$$

■ Different Gradient Discent (Required)

- Batch Gradient Descent

  It moves forward one step after calculating all the training data points. However, if the size of training data set is too huge, it will cost a great amount of memory and time to continue the process.

- Mini-Batch Gradient Descent

  To avoid huge amount of calculation for each move, it is better to move each step after calculating the number of training data points, says “m” here. If m=1, mini-batch gradient descent then equals to stochastic gradient descent.

- Stochastic Gradient Descent

  It s a stochastic approximation of the gradient optimization method for minimizing an objective funtions. It makes an update to the weight vector based on ramdomly picked one data point at a time as an equation below.

  $$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)})$$

  During each itertation, it only uses one randomly picked training data point to make the weight update. Compared to Batch Gradient Descent, this algorithm can be calculated more quicky and it move toward the optimal solution, the minimum of the objective function. However, it will not convergent to the optimal soluton but still has really small error.

- Online Gradient Descent

  To some extent, it equals to stochastic gradient descent. But online gradient descent would require to use the "most recent" training data sample at each iteration. There is no stochasticity as you deterministically select your sample.

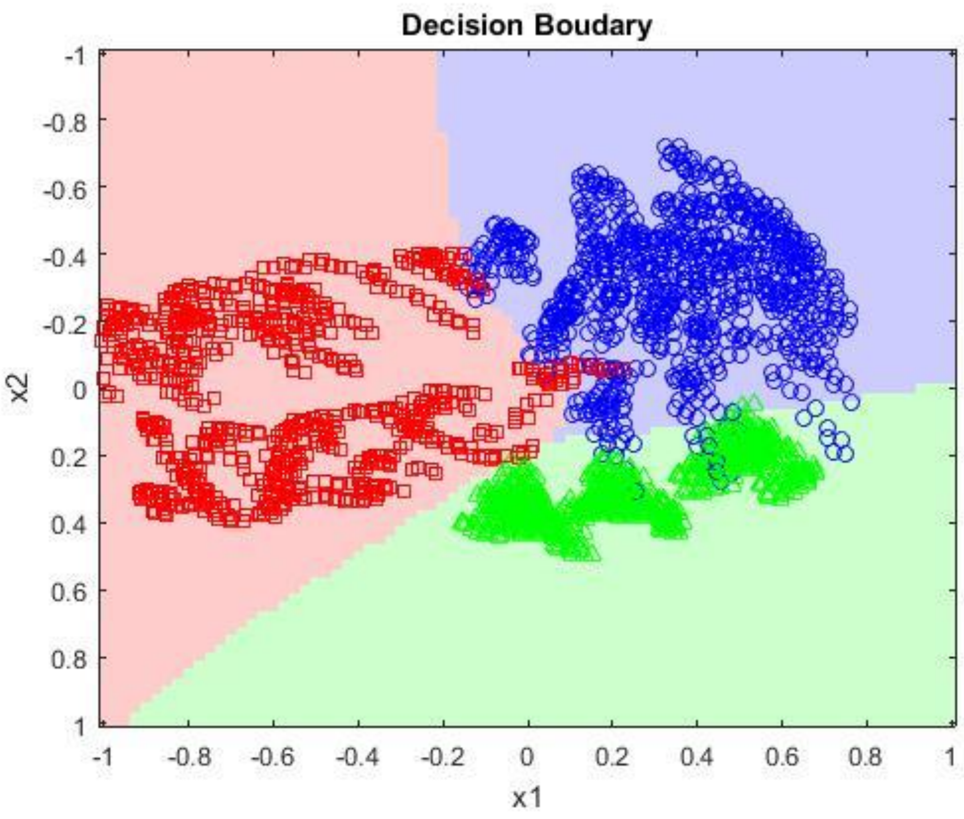
- Result
  - One hidden layer with sigmoid function
    - Setting:

      Number of Nodes in hidden layer: 4

      Learning rate:0.005

Stage	Class	Number of Images
Training	1	900
	2	900
	3	900
Testing	1	100
	2	100
	3	100

    - Decision boundary



➢ Error rate

	1	2	3	4	5
Error Rate(%)	1	1	1	0.67	0.67

■ Two hidden layer with sigmoid function

➢ Setting:

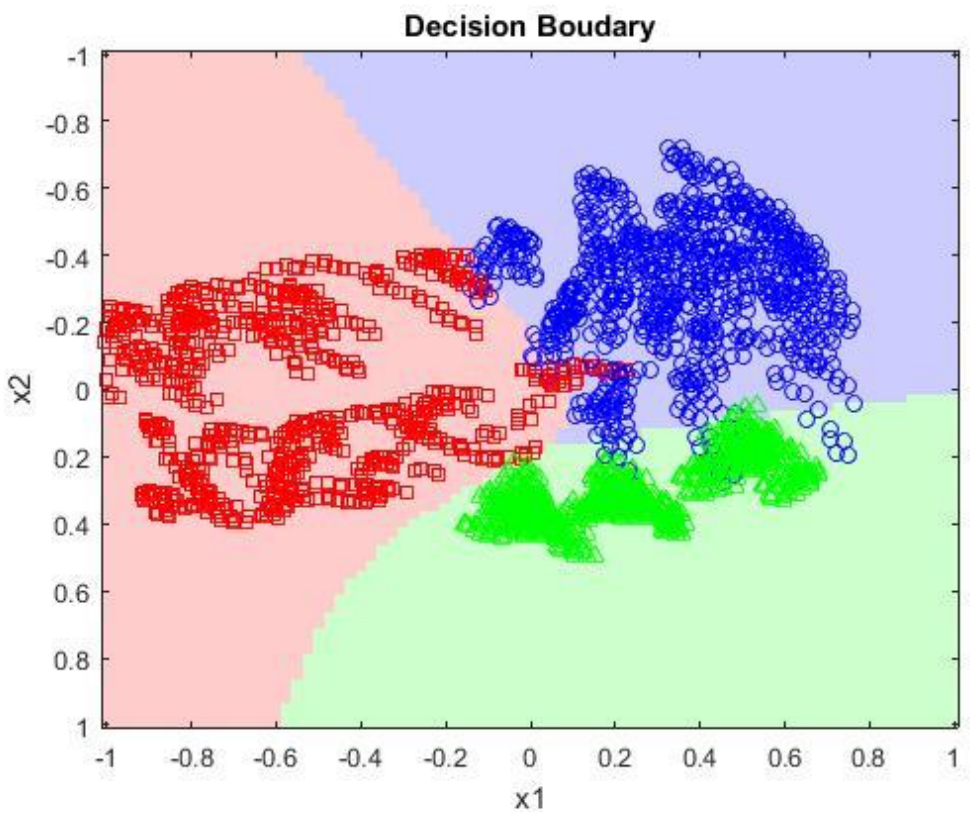
Number of Nodes in 1st hidden layer: 4

Number of Nodes in 2nd hidden layer: 4

Learning rate:0.005

Stage	Class	Number of Images
Training	1	900
	2	900
	3	900
Testing	1	100
	2	100
	3	100

- Decision boundary



- Error rate

	1	2	3	4	5
Error Rate(%)	1	0.67	0.67	0.67	0.67

- One hidden layer with rectified function
  - Setting:

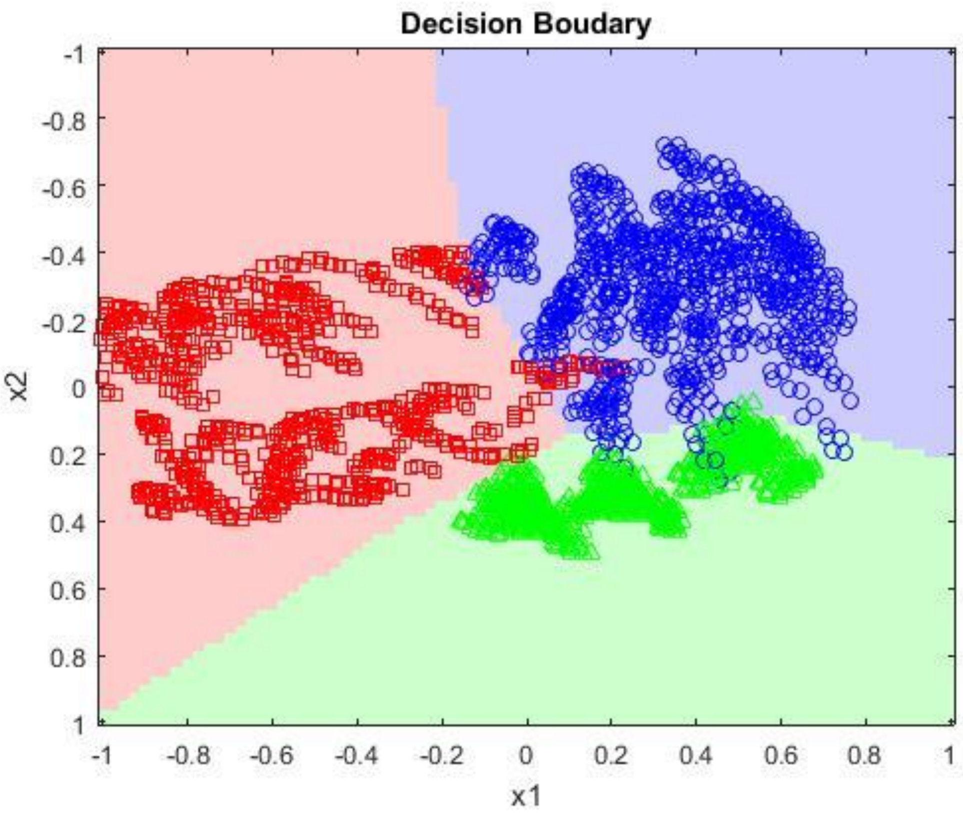
    Number of Nodes in 1st hidden layer: 4

    Number of Nodes in 2nd hidden layer: 4

    Learning rate:0.005

Stage	Class	Number of Images
Training	1	900
	2	900
	3	900
Testing	1	100
	2	100
	3	100

  - Decision boundary



➢ Error rate

	1	2	3	4	5
Error Rate(%)	1.34	1.34	0.67	1.00	1.34

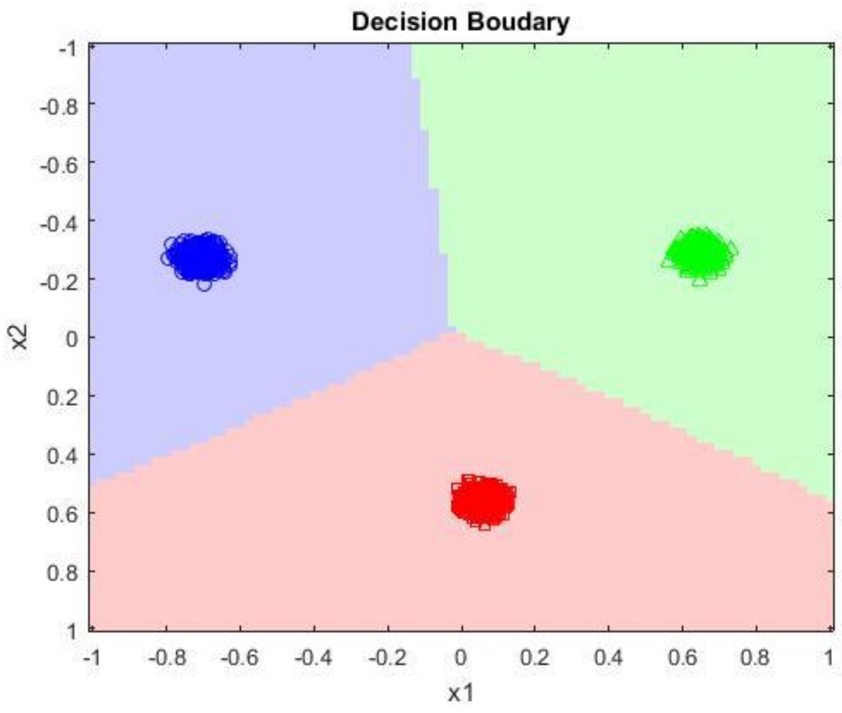
- Discussion
  - Compare and discuss perfomance between neural network model and liner classifcation model

Error Rate(%)	1	2	3	4	5
One hidden layer (sigmoid)	1	1	1	0.67	0.67
Two hidden layer (sigmoid)	1	0.67	0.67	0.67	0.67
Two hidden layer (rectified)	1.34	1.34	0.67	1.00	1.34
Probabilistic Generative Models:	5.67	3.67	4.33	3.67	5.00
Probabilistic Discriminative Models:	1.67	2.00	3.00	3.00	2.00

In general, although there are variances in error rate of each different algorithm, we can still find linear classification model using probabilistic generative model has relatively worse performance. Neural network models in general have better performance than linear classication model. I guess the reason is that neural network does error propagation so that it can be more sensitive to every data.

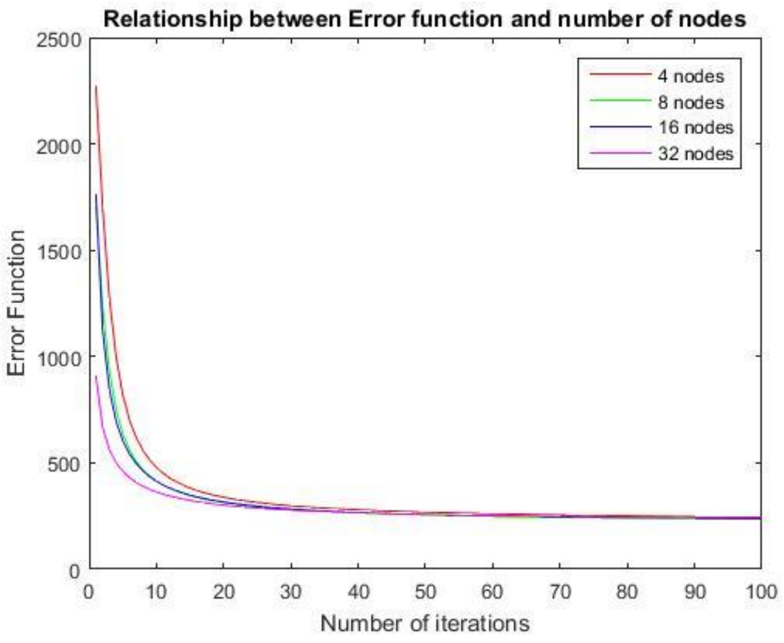
  - Using different algorithm to reduce the dimension of data

I use Fisher's discriminant analysis to do the reduction of dimension. Then, train neural network to with those reduced data. Perfectly, it achieve almost zero error rate. The decision boundary are shown below.

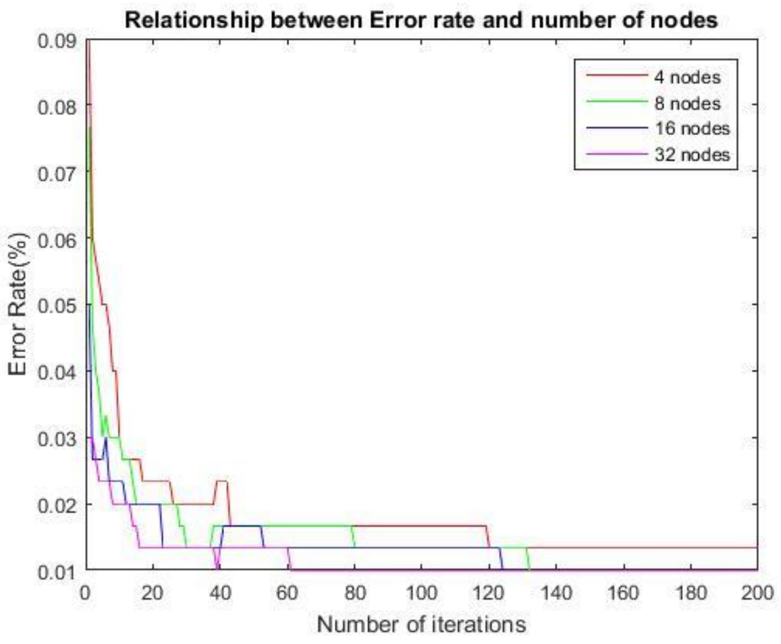


■ Relationship between the number of nodes and model performance

I use one hidden layer to demonstrate the relationship between the number of nodes and performance, solution covergent speed of trained model. Then, present the relationship below.

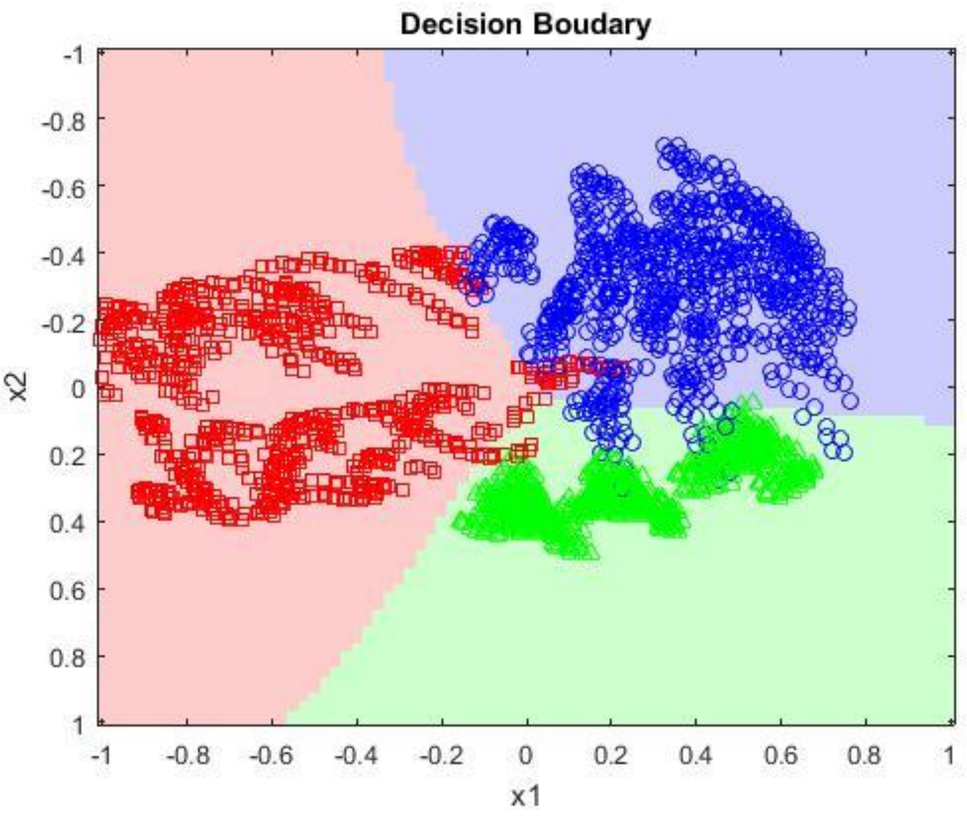


We can find that as the number of nodes increase, the error function generally becomes smaller in each iteration and also has a smaller beginning value. In my opinion, the reason why the error function decrease as the increase of the number of nodes in hidden layer is that it add more flexibility to fit those training data.



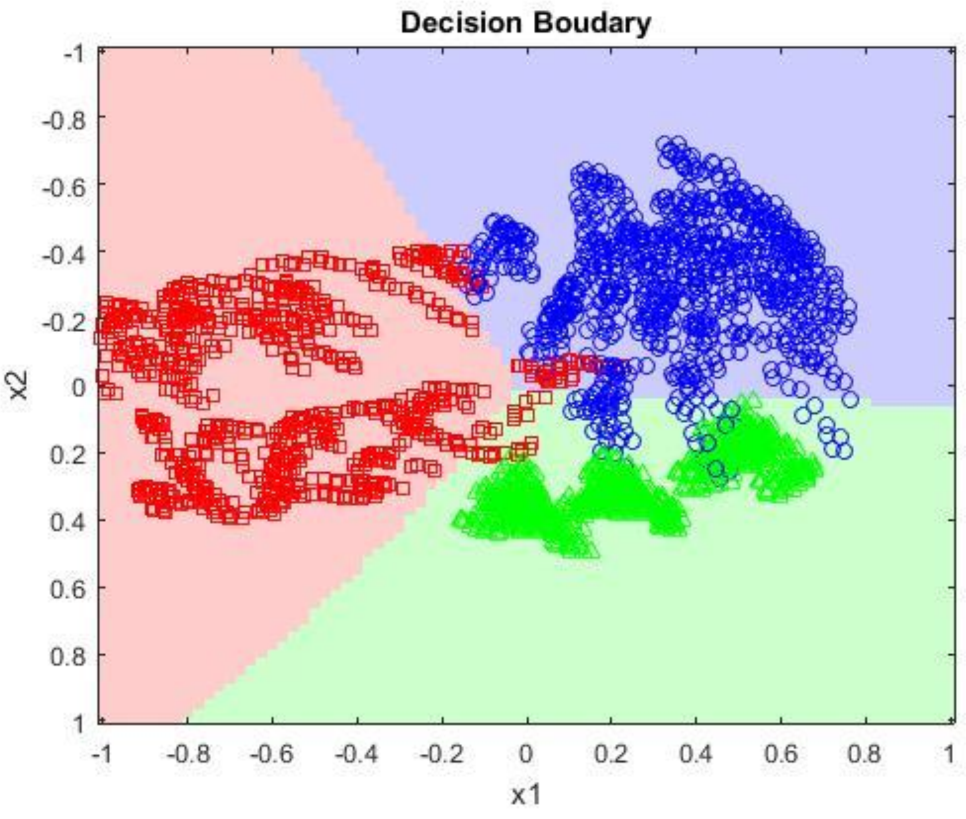
On the other hand, as the number of nodes increase, the error rate becomes relatively smaller. And I consider the reason is as same as for error function.

3. Implement Mini-Batch Gradient Descent algorithm (Bonus)



	1	2	3
Error Rate(%)	2.67	2.67	2.67

(Updata weighting for every 10 training data points)



	1	2	3
Error Rate(%)	3.67	5.00	4.33

(Updata weighting for every 30 training data points)

Although mini batch gradient descent has relatively worse performance than stochastic gradient descent, it is still generally better than linear classification model with probabilistic generative model if the number of batch is around 10 and has almose identical performance to the one with probabistic discriminant model if the number of batch is around 30.

- Reference

[1] Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer, 2007
[2] University of Illinois, Urbana-Champaign ECE 311: Digital Signal Processing Lab 6 Tutorial, November 1,2016
[3] https://blog.gtwang.org/statistics/standford-machine-learning-1/
[4] http://blog.bryanbigdata.com/2014/11/algorithm-stochastic-gradient.html