

Pattern Recognition and Machine Learning

Experiment Report

院（系）名称 自动化科学与电气工程学院

专业名称 模式识别与智能系统

学生学号 15031232

学生姓名 刘霜婷

2017年5月28日

**4 Neural Networks and Back Propagation**

**4.1 Introduction**

The learning model of Artificial Neural Networks (ANN) (or just a neural network (NN)) is an approach inspired by biological neural systems that perform extraordinarily complex computations in the real world without recourse to explicit quantitative operations. The original inspiration for the technique was from examination of bioelectrical networks in the brain formed by neurons and their synapses. In a neural network model, simple nodes (called variously “neurons” or “units”) are connected together to form a network of nodes, hence the term “neural network”.

Each node has a set of input lines which are analogous to input synapses in a biological neuron. Each node also has an “activation function” that tells the node when to fire, similar to a biological neuron. In its simplest form, this activation function can just be to generate a ‘1’ if the summed input is greater than some value, or a ‘0’ otherwise. Activation functions, however, do not have to be this simple - in fact to create networks that can do useful things, they almost always have to be more complex, for at least some of the nodes in the network. Typically, there are at least three layers to a feed-forward network - an input layer, a hidden layer, and an output layer. The input layer does no processing - it is simply where the data vector is fed into the network. The input layer then feeds into the hidden layer. The hidden layer, in turn, feeds into the output layer. The actual processing in the network occurs in the nodes of the hidden layer and the output layer.

**4.2 Principle and Theory**

The goal of any supervised learning algorithm is to find a function that best maps a set of inputs to its correct output. An example would be a simple classification task, where the input is an image of an animal, and the correct output would be the name of the animal. For an intuitive example, the first layer of a Neural Network may be responsible for learning the orientations of lines using the inputs from the individual pixels in the image. The second layer may combine the features learned in the first layer and learn to identify simple shapes such as circles. Each higher layer learns more and more abstract features such as those mentioned above that can be used to classify the image. Each layer finds patterns in the layer below it and it is this ability to create internal representations that are independent of outside input that gives multi-layered networks their power. The goal and motivation for developing the back-propagation algorithm was to find a way to train a multi-layered neural network such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output.

Mathematically, a neuron's network function is defined as a composition of other functions which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely-used type of composition is the nonlinear weighted sum, where:

where K (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions as simply a vector . Back-propagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method

The squared error function is:

where E is the squared error, t is the target output for a training sample, and y is the actual output of the output neuron. For each neuron j, its output is defined as

The input net to a neuron is the weighted sum of outputs ok of previous neurons. If the neuron is in the first layer after the input layer, the of the input layer are simply the inputs to the network. The number of input units to the neuron is n. The variable denotes the weight between neurons i and j. The activation function is in general non-linear and differentiable. A commonly used activation function is the logistic function, e.g.:

which has a nice derivative of:

Calculating the partial derivative of the error with respect to a weight is done using the chain rule twice:

We can finally yield:

with

**4.3 Objective**

The goals of the experiment are as follows:

(1) To understand how to build a neural network for a classification problem.

(2) To understand how the back-propagation algorithm is used for training a given a neural network.

(3) To understand the limitation of the neural network model (e.g., the local minimum).

(4) To understand how to use back-propagation in Auto-encoder.

**4.4 Contents and Procedure**

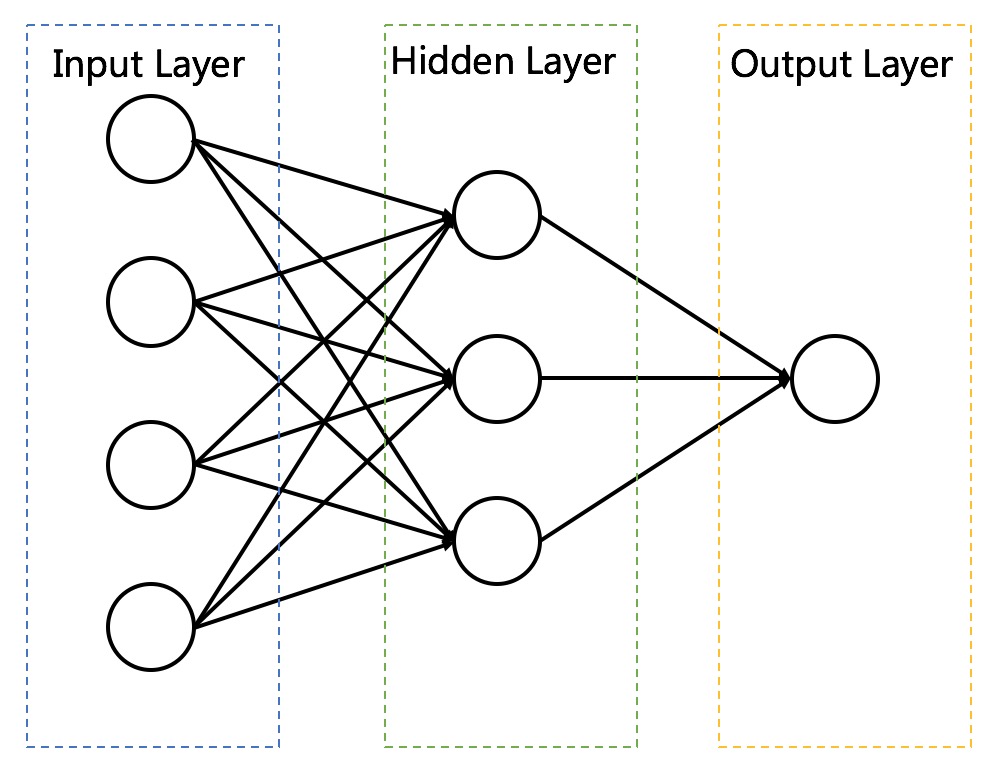
**Stage 1:**

1. **Build the NN model with Iris Dataset.**

I complete this experiment in MATLAB with Iris Dataset.

In this stage, I first load data and change the format of data. Each iris is described by four features: sepal length, sepal width, petal length and petal width. For each data, four attributes act as input to the neural network. As for the labels of dataset, I replace Iris-setosa with 1, Iris-versicolor with2, and Iris-virginica with 3. Since we use logistic function to active neutrons, the output of neural network should be in the range of 0 and1, so that we also mapping the labels to be in the range of 0 and 1. Therefore, 1/3 represents Iris-setosa, 2/3 represents Iris-versicolor, 1 represents Iris-virginica.

The neural network has 1 input layer with 4 nodes, 1 hidden layer with 3 nodes and 1 output layer with one node. The four flower attributes will act as inputs to the NN with one hidden layer, which will map them onto one output. Feeding four attributes of a pair of data to the network, we want to obtain the category of the flower. The structure is shown as follows:



Iris dataset is divided into two parts. The training set has 120 number of data, and the testing set has 30 number of data. To begin with, I randomize the order of data and the weights in the network. I set the learning rate as 0.1 and the amount of iteration as 200. Using back propagation algorithm, the weights in network can be updated. After 200 iterations, we can obtain all weights in this network.

Then we give testing dataset as input to the trained network and can obtain a predicted result. Calculating the error between real label and predicted label, we can obtain the classification accuracy. For the learning rate 0.1 in NN model, after 300 iterations, classification accuracy could reach up to 96.7%.

1. **Compare the efficiency and accuracy between NN model and the perceptron.**

The result of perceptron: 406 iterations are required and the correct rate is 76.7% when the learning rate is set to be 0.1. In previous stage, we already know that for the learning rate 0.1 in NN model, after 300 iterations, classification accuracy could reach up to 100%

Also, the experiment shows that the NN model works faster than the perceptron, which is due to the perceptron is a binary classifier, making it necessary to operate twice to separate the Iris dataset which contains 3 categories apart. As a result, more time is required.

As for perceptron, the ideal condition is the dataset is linearly related. However, the NN model works well for both linear and non-linear conditions. So it could attain more accuracy and it is able to handle more complicated problems.

1. **Explore the influence of parameter settings on the performance of the model.**

* Learning Rate

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 0.01 | 0.05 | 0.1 | 0.5 | 1 | 10 | 15 |
| Accuracy | 76.7% | 93.3% | 96.7% | 100% | 96.7% | 40% | 26.7% |

The graph above shows the classification accuracy changes with learning rate.

When learning rate increases, the accuracy first increases and then decreases. This is because that when learning rate is smaller, the weights are harder to get to the optimal in a certain number of iterations; when learning rate is bigger, the network is hard to converge so that the result cannot achieve the point at minimal error.

* Original Weights

The original weights have little impact on the result since that the different value of original weights only influences the time for training the NN model but for the same training set, the weights continuously update, so the training result always converge to the same value.

* Number of Nodes in Hidden Layers

In order to view the influence of nodes in hidden layers on the model, I gradually change the number of nodes from 3 to 11 and calculate the correct rate respectively. The two learning rates is set to be 0.1, with 10000 iterations. The result is shown in the table below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nodes | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Accuracy | 96.7% | 96.7% | 96.7% | 93.3% | 93.3% | 96.7% | 93.3% | 90% | 93.3% |

From the result, the number of nodes seem to have little or no impact on the accuracy of model. As the number of nodes grows, the calculation becomes more complicated.

* Number of Iterations

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Number of Iterations | 50 | 100 | 500 | 1000 | 5000 | 100000 |
| Accuracy | 86.7% | 93.3% | 96.7% | 96.7% | 96.7% | 100% |

When the number of Iterations increases, the classification accuracy also increases. With the increase of iterations, the improve of accuracy become more and more slowly. In the meanwhile, the amount of time used for calculation also increases. Therefore, we need to find a suitable number of iteration to ensure both high accuracy and short time.

**Stage 2:**

**(1) How the efficiency and accuracy will be influenced by different activation functions and more hidden layers.**

In the experiment, I explore the influence of three different active functions on the result: Purelin Function, Logsig Function, Tansig Function, where Logsig Function is the active function that we used above.

The Purelin Function:

The Logsig Function:

The Tansig Function:

The two learning rates is set to be 0.1, with 10000 iterations. The number of nodes in hidden layer is still 3. The result is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| Active Function | Purelin Function | Logsig Function | Tansig Function |
| Correct Rate | 76.7% | 96.7% | 0.93.3% |

As is shown in the table above, the linear active function has the lowest correct rate. And the Logsig Function and the Tansig Function perform nearly the same.

The number of hidden layers is determined on the complicity of the problem. In some cases, the accuracy of the result increases with the number of hidden layers. However, too many hidden layers may also cause the overfitting problem and much calculation is required. Also, the efficiency usually decreases with the increase of the number of hidden layers. Therefore, we are supposed to find an appropriate number of hidden layers.

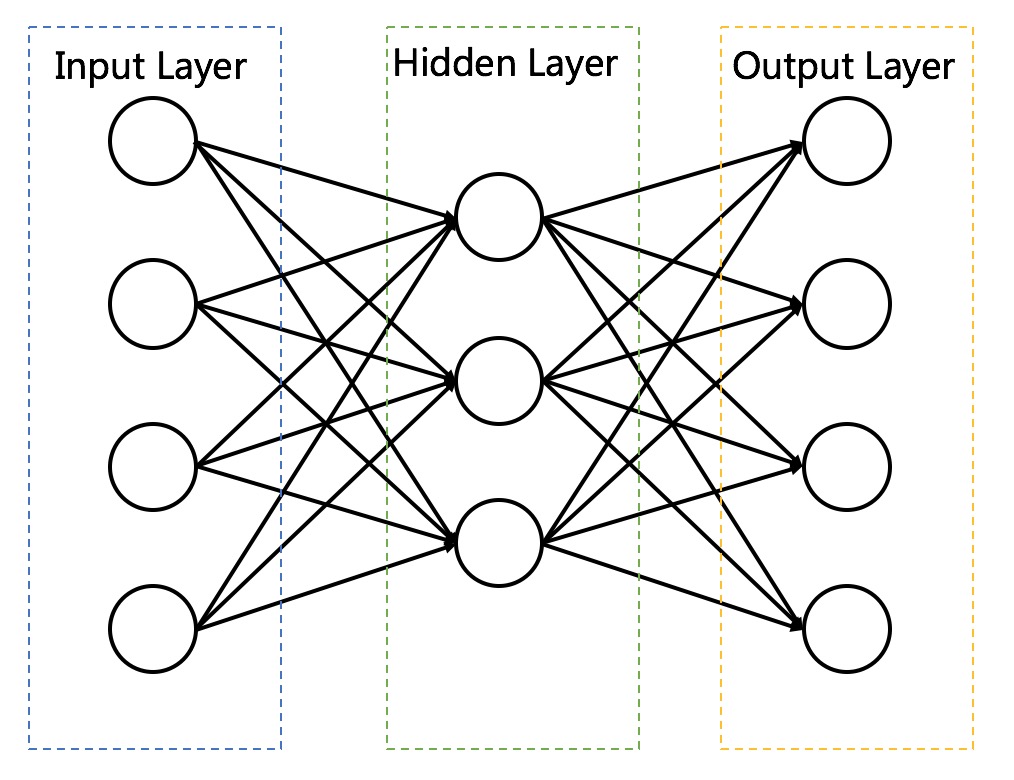
**(2) The difference between the biological neural networks and NN model:**

The invention of NN model is inspired by biological neural networks. In my opinion, the biological neural networks share some similarities: they have similar structures which consist of many units and the function of the network is achieved by the cooperation of each unit. However, they have totally different working mechanism.

The biological neural networks work through the passing of electrical and chemical signal which is attained through physical connections. While the NN model is based on mathematical relationship. Also, the connections between units in biological neural networks are established automatically and new units are likely to get involved in. While the NN model is fixed and only the transfer matrix is modified through back-propagation. What’s more, one biological neural network could achieve many different functions while one NN model could only one kind of tasks.

**(3) The auto-encoder and corresponding analysis.**

The auto-encoder is a kind of artificial neural network, which means that the number of nodes in the input layer is the same as the number of nodes in the output layer. The figure below shows the structure of an auto-encoder model.

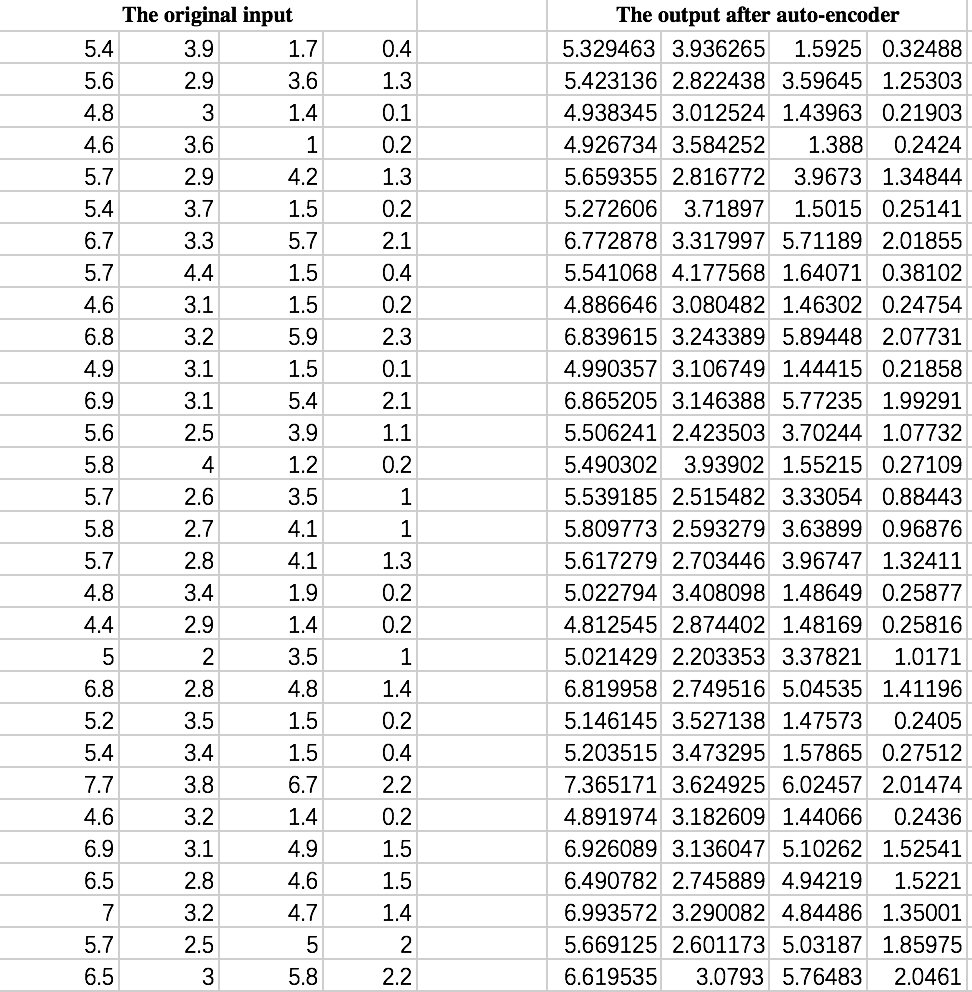


In the experiment, I choose the iris data to complete the assignment, that is, the auto-encoder consists of one input layer with 4 nodes, one output layer with 4 nodes as well as one hidden layer with 3 nodes. After randomizing the original dataset, I choose the former 120 data points as the training set and the later 30 data points as the test set.

Since the active function mapping data from 0 to 1, I mapping the dataset using below function:

During the experiment, we train the auto-encoder learning rate 0.1. The iteration is 1000. Through calculation, if choosing the threshold of correction as 0.2, we could obtain that the reliability of the auto-encoder is 96.7%.

The comparison between the input and output is shown below:

****

**4.5 Experience**

Before the experiment, I have a basic understanding of the NN model and how it works through back propagation. By completing the experiment, not only have I gained a clearer view of the structure of NN model, but I correct some understandings which may lead to the wrong results when using the NN model.

What’s more, I gradually recognize that the NN model could help us with both classification problems and regression problems. Since that its structure could be either simple or complicated, the NN model and the CNN & RNN model are widely used in data mining.

**4.6 Codes**

You can find my code on

clear all;

clc;

%% load data

f=fopen('iris.data.txt');

orig=textscan(f,'%f%f%f%f%s','delimiter',',');

fclose(f);

%% data processing

% use numbers to replace properties

% in matirc B, Iris-setosa-->1,Iris-versicolor-->2,Iris-virginica-->3

attributes=[orig{1,1},orig{1,2},orig{1,3},orig{1,4}];

Class=zeros(150,1);

Class(strcmp(orig{1,5},'Iris-setosa')) = 1/3;

Class(strcmp(orig{1,5},'Iris-versicolor')) = 2/3;

Class(strcmp(orig{1,5},'Iris-virginica')) = 1;

data=[attributes Class];

rowrank = randperm(size(Class,1));

data = data(rowrank,:); % Randomize

data\_p=data;

data\_p(:,5)=data\_p(:,5).\*3;

train\_data = data(1:120,:); % Training set & Testing set

test\_data = data(121:150,:);

train\_datap = data\_p(1:120,:);

test\_datap = data\_p(121:150,:);

%% initializing

r=0.1; %learning rate

inputNeuron=4;

hiddenNeuron=3;

outputNeuron=1;

iteration = 200; % Number of iteration

v0=rand(hiddenNeuron,inputNeuron); %generating random weight

w0=rand(outputNeuron,hiddenNeuron);

gamma=rand(hiddenNeuron,1);

theta=rand(outputNeuron,1);

v=[v0 gamma];

w=[w0 theta];

%% train&test

[w,v] = trainBP(w,v,train\_data,iteration,r);

[accuracy\_t]=testBP(w,v,test\_data);

%% The Condition of Using Perceptron

theta1 = 0.1;

[transmatrix1 transmatrix2 num] = perceptron1(train\_datap,theta1)

right\_rate = testperc(transmatrix1,transmatrix2,test\_datap)

%% auto encoder

r\_e=0.1; %learning rate

inputNeuron\_e=4;

hiddenNeuron\_e=3;

outputNeuron\_e=4;

iteration\_e = 1000; % Number of iteration

v0\_e=rand(hiddenNeuron\_e,inputNeuron\_e); %generating random weight

w0\_e=rand(outputNeuron\_e,hiddenNeuron\_e);

gamma\_e=rand(hiddenNeuron\_e,1);

theta\_e=rand(outputNeuron\_e,1);

v\_e=[v0 gamma];

w\_e=[w0 theta];

% train&test

[wAE,vAE] = trainAE(w\_e,v\_e,train\_data,iteration\_e,r);

[output,accuracyAE\_t]=testAE(wAE,vAE,test\_data);

trainBP.m

%% train BP

function [w,v] = trainBP(w,v,train\_data,iteration,r)

for j=1:iteration

for i=1:120

x=train\_data(i,1:4);

X=[x -1]';

yReal=train\_data(i,5);

b0=v\*X; %each neuron value in hidden layer

b0=1./(1+exp(-b0)); %sigmoid

b=[b0;-1]; %b is hidden layer

Y=w\*b;

O=1./(1+exp(-Y));

g=O.\*(1-O).\*(yReal-O);

%updating parameters

dw=r.\*(g\*b');

w=w+dw;

we=w;

S=size(dw(1,:));

we(:,S(1,2))=[];

e=b0.\*(1-b0).\*(we'\*g);

%updating parameters

dv=r.\*(e\*X');

v=v+dv;

end

end

end

testBP.m

%% test BP

function [accuracy\_t]=testBP(w,v,test\_data)

wrongNum=0;

for i=1:30

x=test\_data(i,1:4);

X=[x -1]';

bTest=v\*X;

bTest=1./(1+exp(-bTest));

bTest=[bTest;-1];

yTest=w\*bTest;

yTest=1./(1+exp(-yTest));

if abs(test\_data(i,5)-yTest)>0.15

wrongNum=wrongNum+1;

end

end

totalWrong=wrongNum/30;

accuracy\_t=1-totalWrong;

trainAE.m

%% train AE

function [w,v] = trainAE(w,v,train\_data,iteration\_e,r)

for i=1:4

train\_dataAE(:,i)=(train\_data(:,i)-min(train\_data(:,i)))/(max(train\_data(:,i))-min(train\_data(:,i)));

end

for j=1:iteration\_e

for i=1:120

x=train\_dataAE(i,1:4);

X=[x -1]';

yReal=train\_dataAE(i,1:4)';

b0=v\*X; %each neuron value in hidden layer

b0=1./(1+exp(-b0)); %sigmoid

b=[b0;-1]; %b is hidden layer

Y=w\*b;

O=1./(1+exp(-Y));

g=O.\*(1-O).\*(yReal-O);%训练时O应该和yReal差不多

%updating parameters

dw=r.\*(g\*b');

w=w+dw;

we=w;

S=size(dw(1,:));

we(:,S(1,2))=[];

e=b0.\*(1-b0).\*(we'\*g);

%updating parameters

dv=r.\*(e\*X');

v=v+dv;

end

end

end

testAE.m

%% test AE

function [out,accuracy\_t]=testAE(w,v,test\_data)

for i=1:4

test\_dataAE(:,i)=(test\_data(:,i)-min(test\_data(:,i)))/(max(test\_data(:,i))-min(test\_data(:,i)));

end

output=[];

wrongNum=0;

for i=1:30

xt=test\_dataAE(i,1:4);

Xt=[xt -1]';

bTest=v\*Xt;

bTest=1./(1+exp(-bTest));

bTest=[bTest;-1];

yTest=w\*bTest;

yTest=1./(1+exp(-yTest));

if((abs(test\_data(i,1:4)'-yTest))>0.2)

wrongNum=+1;

end

output(i,:)=yTest;

end

totalWrong=wrongNum/30;

accuracy\_t=1-totalWrong;

for i=1:4

out(:,i)=(max(test\_data(:,i))-min(test\_data(:,i))).\*output(:,i)+min(test\_data(:,i));

end

end

perceptron1.m

function [w1 w2 iteration] = perceptron1(train\_data,theta)

w = [0;0;0;0;0];

class\_total = [];

class\_else = [];

for i = 1:length(train\_data(:,1))

if(train\_data(i,5) > 1)

train\_temp = [(-1)\*train\_data(i,1:4),-1];

class\_total = [class\_total;train\_temp];

class\_else = [class\_else;train\_data(i,:)];

else

class\_total = [class\_total;train\_data(i,:)];

end

end

[w1 iteration1] = perc(class\_total,theta);

class = [];

for j = 1:length(class\_else(:,1))

if(class\_else(j,5) > 2)

train\_temp = [(-1)\*class\_else(j,1:4),-1];

class = [class;train\_temp];

else

class = [class;class\_else(j,:)];

end

end

[w2 iteration2] = perc(class,theta);

iteration = iteration1 + iteration2;

end

testperc.m

function right\_rate = testperc(transmatrix1,transmatrix2,test\_data)

count = 0;

for i = 1:length(test\_data(:,1))

if([test\_data(i,1:4),1]\*transmatrix1 > 0) % Type1

test\_result(i) = 1;

else

if([test\_data(i,1:4),2]\*transmatrix2 > 0) % Type2

test\_result(i) = 2;

else

test\_result(i) = 3;

end

end

if(test\_data(i,5) - test\_result(i) < 0.001)

count = count + 1;

else

count = count;

end

end

right\_rate = count/length(test\_data(:,1));

end

perc.m

function [w iteration] = perc(class\_total,theta)

w = [0;0;0;0;0];

% find the optimal vector

counter = 0;

iteration = 0;

flagexit = 0;

m = length(class\_total(:,1));

while(flagexit == 0)

for j = 1:m

tempt\_result = class\_total(j,:) \* w;

if(tempt\_result > 0)

iteration = iteration + 1;

w = w;

counter = counter + 1;

else

iteration = iteration + 1;

w = w + theta \* (class\_total(j,:))';

counter = 0;

end

end

if(counter >= m)

flagexit = 1;

else

flagexit = 0;

end

end

end