**Programming Assignment 5: Neural Networking**

Instructor: Dr. Satish Kumar Thittamaranahalli

Group Member: Yu Hou; Haoteng Tang

In this assignment, we will implement neural networking algorithm. We have several datasets. Each dataset represents a picture with file format "pgm". Each dataset has 960 points to represent one picture.

**1 Procedure of implementing algorithm (Yu Hou)**

(1) procedure

**Artificial neural networking:**

*#Forward:*

Generate weight w1, w2 randomly. W1 is [960\*100], and W2 is [100\*1]

Feed the all perceptron with all images one by one, and get the output layer2.

*#Back propagation*

Delta2 = (y-layer2) \* layer2 \* (1-layer2)

W2 = W2 – yeta \* layer1 \* delta2

Delta1 = (Delta2\*W2) \* layer1 \* (1-layer1)

W1 = W1 – yeta \* layer0 \* Delta1

*# using these functions to update W1 and W2.*

*# predict the results.*

(2) Results

In this assignment, when we get our result, we also can get a report named the confusion matrix. The confusion matrix can explain the result of the classification. Four numbers will be record, the true positive, the false positive, the true negative, and the true positive. Shown as the figure 1.

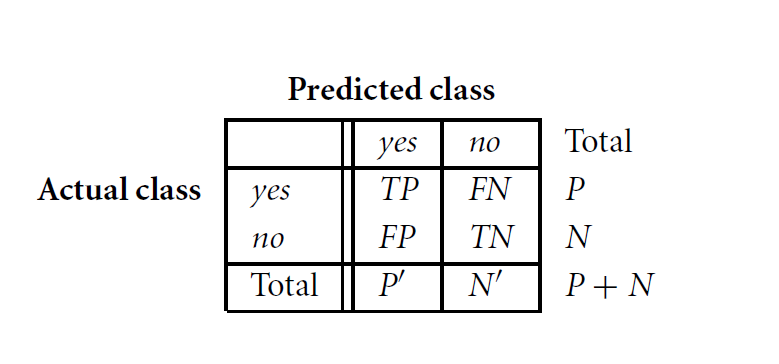


Figure 1 Confusion matrix

The total accuracy = (TP+TN)/( TP+TN+FP+FN), true positive rate = TP/(TP+FN), the true negative rate=(TN)/(TN+FP), positive predictive value = TP/(TP+FP), negative predictive value = TN/(TN+FN).

In this assignment, we print out the total accuracy of training data and the total accuracy, true positive rate, true negative rate, positive predictive value, and negative predictive value of testing data. Here are the results for 1000 epochs.

*The predictive number of '1': 16*

*The real number in testing data of '1': 19*

*The true positive rate: 0.8421052631578947*

*The true negative rate: 1.0*

*The positive predictive value: 1.0*

*The negative predictive value: 0.9552238805970149*

*The total accuracy: 0.963855421686747*

Also, we predict the testing data and compare the predictive result with the real one. Here are the results.

*PREDICT of gestures/A/A\_down\_1.pgm iS: [ 1.] Real is: [1]*

*PREDICT of gestures/A/A\_down\_2.pgm iS: [ 1.] Real is: [1]*

*PREDICT of gestures/A/A\_hold\_1.pgm iS: [ 0.] Real is: [0]*

*………*

*PREDICT of gestures/B/B\_down\_2.pgm iS: [ 0.] Real is: [1] 🡨 wrong*

*PREDICT of gestures/B/B\_hold\_1.pgm iS: [ 0.] Real is: [0]*

*……..*

*PREDICT of gestures/D/D\_down\_1.pgm iS: [ 1.] Real is: [1]*

*PREDICT of gestures/D/D\_down\_2.pgm iS: [ 0.] Real is: [1] 🡨 wrong*

*PREDICT of gestures/D/D\_hold\_1.pgm iS: [ 0.] Real is: [0]*

*………*

*PREDICT of gestures/I/I\_up\_3.pgm iS: [ 0.] Real is: [0]*

*PREDICT of gestures/J/J\_down\_5.pgm iS: [ 1.] Real is: [1]*

*PREDICT of gestures/J/J\_down\_6.pgm iS: [ 0.] Real is: [1] 🡨 wrong*

*………*

*PREDICT of gestures/K/K\_stop\_1.pgm iS: [ 0.] Real is: [0]*

*PREDICT of gestures/K/K\_stop\_2.pgm iS: [ 0.] Real is: [0]*

(3) data structure

First, we should read data from the pictures stored by “pgm” files. The “pgm” file consists of many points, pixels. These pixels are represented in numbers ranging from 0 to 255. There 184 instances of these pictures, and they will be read into an array list, [184,960].

Here is an instruction how to read “pgm” file.

*# A diagonal line 10 pixels wide and 10 pixels high. 🡨 comments*

*10 10 🡨 10 rows 10 columns*

*255 🡨scale.*

*0 255 255 255 255 255 255 255 255 255 🡨points*

*255 0 255 255 255 255 255 255 255 255*

*……*

In the “pgm” file, the first line is a comment, and the second line represents how many rows and how many columns in this pictures. The third line represents the scale of these pixels. From the fourth line to the end of this file are numbers representing the pixels.

The structure of the weights of the perceptron are also represented as lists. Because we have the hidden layer of size 100, and one output node, the w2 is an array list representing as [100,1], and the w1 is an array list representing as [960,100]. For the value of W1 and W2, we initialize all w randomly between -1 to 1, since it will be easy to get a good result.

(4) challenges, code-level optimizations

The biggest challenge is to understand the meaning of the formulas. First, we just wrote the codes with formulas the professor gave in the class. However, we found that they are different formulas between in homework and in class. In the class, the professors used the seta functions like Ɵ(s) = (es -e-s)/(es +e-s). However, in this assignment, we are asked to use Ɵ(s) = 1/(1+e-s). Therefore, the weight update functions are different. We used professor’s hints: “For the error function, use the standard least square error function.”

We also optimized our code in different ways. For example, when we first implement our code, we just update the weight once with the whole data. However, this will not get a good result. We optimized our code by testing data one by one using stochastic gradient descent. In this way, we can get a good result. We noticed that in the training data, the number of records are 184, but the dimension of one picture is 960. Therefore, maybe we can consider to reduce the dimension.

**2 Software Familiarization (Yu Hou,** **Haoteng Tang)**

(1) Python for neural networking

*from sklearn.neural\_network import MLPClassifier*

*## import the package*

*c = MLPClassifier(solver='sgd', alpha=0, hidden\_layer\_sizes=(100,), activation='logistic', learning\_rate\_init=0.1, max\_iter=10000)*

*## define the model*

*c.fit(images, labels)*

*## feed the data into the model.*

(2) Matlab for neural networking

There is a toolbox about Neural Network implanted in the Matlab 2016. And there are two library functions named ‘newff’ and ‘train’ about construction and training a neural network. The code is:

*% Label of the training data*

*Label=[1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0, 0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];*

*a=1;*

*while a<=184*

*% for each data, forward propagation*

*%input the image*

*imageName=strcat(num2str(a),'.pgm');*

*Image=imread(imageName);*

*Image=double(Image); % convert from uint8 to double type for calculation*

*Image=Image./255; % scale pixel to range from 0 to 1*

*Image=Image';*

*Data\_In(a,:)=Image(:)'; %form the input data. This is 1\*960 without bias*

*a=a+1;*

*end*

*Data\_In=Data\_In';*

*net=newff(Data\_In,Label,100,{'tansig','tansig'},'traingdx');*

*net.trainParam.epochs=6000;*

*net.trainParam.goal=0.0001;*

*net.trainParam.show=25;*

*net.trainParam.lr=0.15;*

*[net,tr]=train(net,Data\_In,Label);*

First, we should obtain input data from all image and process them as a 960\*184 matrix. Each of the column of this matrix represents each data. Besides, we should we are going to set up a 1\*184 matrix to save our label. Second, we can construct our neural network and train it as:

*%%net=newff(Data\_In,Label,100,{'tansig','tansig'},'traingdx');%%*

*Newff means: construct a neural network with input data, label, one hidden layer of 100 units and each one use sigmoid function.*

*%% net.trainParam.epochs=6000;*

*net.trainParam.goal=0.0001;*

*net.trainParam.show=25;*

*net.trainParam.lr=0.15;%%*

*These are some parameters of training process such doing 6000 epochs.*

*%%[net,tr]=train(net,Data\_In,Label);%%*

*Train function is about implement the training.*

This is the GUI of the Neural Network. Shown as the figure2.

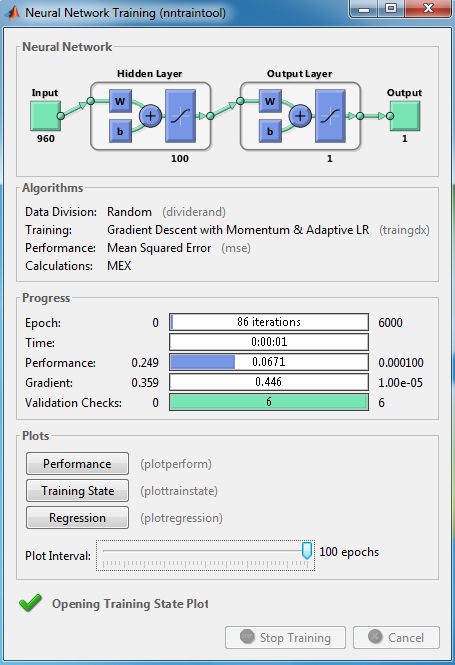


Figure2 GUI of the ANN

Also, the performance of the validation. We can see, with the number of iterations going up, the training data’s error goes down but the testing data’s error is a ‘U-shape’ figure which means it goes down to the minimum value but will go up due to no regularization.

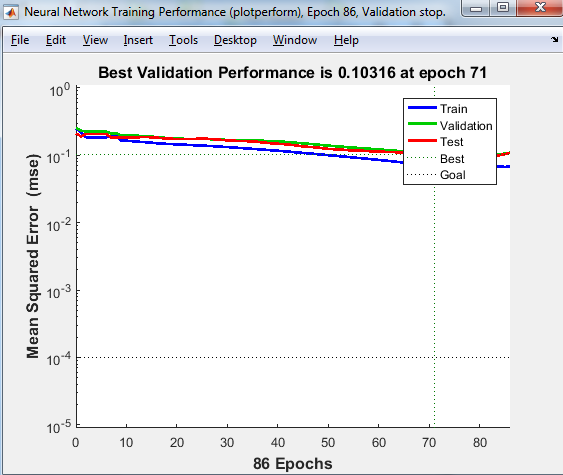


Figure 3 Training performance

(3) Learn from the libraries

First, in the libraries, we can set the size of each hidden layer very easily. However, in my code, we just fix the node of each hidden layer. Maybe we can write our code more flexible for users to set the hidden layers. Second, we can write our code in different classes, so that we can use class to implement our functions.

**3 Application (Haoteng Tang)**

Artifact Neural Network is a kind of model which simulates human cerebral network. The weight is something like a synapsis and the perceptron is something like the nerve cell. In our life, neural network is widely used in system identification, pattern analysis, and intelligent controller. For instance, there are two kind of fish, fish A and B, mixed in a group in a fish factory and we want to separate them. At this point, we can measure some features of these fish such as their length, weight, width and color, and label each of them as our input. Then we can construct a neural network with four input units and a few hidden units and one output. We can input our data into this network and train it. Finally, we can use this network to classify different kind of fish by inputting our testing data.

Furthermore, neural network is a bridge connecting machine learning and deep learning. There are more powerful network in deep learning such as Convolution Neural Network(CNN) and Recurrent Neural Network(RNN). These kinds of neural network can even search the features of object data by themselves without our human’s help. For example, when we use CNN to solve the image identification problem, it can search the features from image themselves by dealing convolution between kernel function and image pixels. The outcome of this step is feature map. Then pooling step will be used to these feature map. And then we can repeat these convolution-pooling two steps for a few times. Final ‘machine’ can get the features from image by itself. Then by dealing with de-convolution, we can reconstruct the image. This kind of neural network is popular in deep learning image identification now.