Laboratory 5

Multilayer NNs: Forward Propagation

Due Date: Beginning of Week 7 Lab

Concepts:

- Feedforward Propagation and Prediction
- Matlab neural network toolbox/App

Total Points: 100 points

Objectives:

Multilayer NNs are useful to learn complex non-linear hypothesis. By connecting multiple neurons in multiple layers, NNs can compute the **complicated non-linear functionalities**. There are two main algorithms in multilayer neural networks: Forward propagation and backward propagation. Forward propagation performs predictions with the given weights. Backward propagation performs learning to fin the optimal weights that minimize the overall cost (error) function. It is just back-propagating the cost (error) over the network to update the model parameters.

In this lab, students will implement the feedforward propagation algorithm using weight parameters that were already trained. In next week's lab, students will implement the backpropagation algorithm for learning the neural network parameters.

Files Needed:

- Lab5_Part1.mlx
- Lab5_Part2.mlx
- sigmoid.m
- displayData.m
- MNIST.mat
- MNIST_Weight

Assignment:

Part1: Forward propagation implementation in multilayer neural networks

In this part I, you will implement the forward propagation algorithm for the MNIST hand written digit recognition problem. Forward propagation performs prediction using the given parameters. It starts with activation input units, further propagates to hidden units to compute activation of hidden units, and finally compute the predictions. You will use the same code for

1. Data source & displayData function:modified from Andrew Ng

the data lading and visualization. The description of the MNIST data is provided below for your information

- The file "MNIST.mat " contains a training set of 4,000 handwritten digits and a test set of 1,000 handwritten digits. The images have a size of 20 by 20 grayscale images of digits. The 20 by 20 digit image is flattened into a 1 by 400 row vector. Each of these digit images becomes a single row in the data. The data is already normalized in the range of [0, 1] and separated into the training set and the test data.

- The matrix for the training data set is already named as 'XTrain' and the corresponding labels for the training set is named as 'yTrain'. Similarly, the matrices for the test data set and the corresponding labels are names as 'XTest' and ' yTest', respectively.
- Note: There is no zero indexing in the Matlab. So the digits "1" to "9" are labeled as "1" to "9" and the digit "0 " is labeled as "10".

1. Load weight matrices

- The file "MNIST_Weight.mat " contains the pretrained weight matrices. Using the ***load*( )** function in Matlab, load the "MNIST_Weight.mat' file. The weight matrices in layer 1, layer 2, and layer3 are named as 'w1', 'w2', and 'w3', respectively.

- **Question 1:** Draw the neural network architecture for the digit recognition system according to the weight matrices, 'w1', 'w2', and 'w3'.

2. Implement the forward propagation on the training data

- Complete code in section2 to implement the feed forward propagation to calculate the neural network's prediction on the given data
- In general, the feed forward algorithm is implemented as:
  - $\boldsymbol{z_i^{(l+1)}}$ denote the linear regression in node $\boldsymbol{i}$ in layer l+1
    - $z_i^{(l+1)} = \left(w_{i0}^{(l)} a_0^{(l)} + w_{i1}^{(l)} a_1^{(l)} + \cdots + w_{ik}^{(l)} a_k^{(l)}\right) = \sum_{j=0}^{k} w_{ij}^{(l)} a_j^l$
  - $a_i^{(l+1)}$ denote **the activation output** in node $\boldsymbol{i}$ in layer l+1
    - $a_i^{(l+1)} = \mathrm{f}(z_i^{(l+1)})$, where f is the sigmoid function
- **Question2**: What is the training accuracy ?

- **Question 3:** what is the **log loss cost with the regularization term and** $\boldsymbol{\lambda = 1}$. Write code that calculates the **log loss cost**. Do not include the weights for the bias terms in the regularization term. FYI, the equation for the log loss calculation is

$$J(w) = -\frac{1}{m}\left[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^i \cdot \log\left(h_w(x^i)_k\right) + (1 - y_k^i) \cdot \log\left(1 - h_w(x^i)_k\right)\right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{S_l} \sum_{j=1}^{S_{l+1}} (w_{ji}^l)^2$$

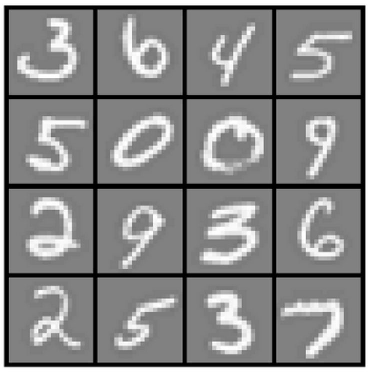
1. Data source & displayData function:modified from Andrew Ng

3. **Calculate the neural network accuracy on the test data**

    - Complete code in section3 to predict the neural network's output on the test data set.
    - **Question 4:** what is the accuracy of the multilayer neural network on the test data set?

4. **Demo the prediction on the test data**

    - Randomly select 16 data samples (images) from the test data and display the images
    - Make the predictions on the selected data
    - If the prediction output is '10' , then change the value to '0'
    - The output will be similar with the figure below:



```
The predicted digits are:
   3    6    4    5
   5    0    0    9
   2    9    3    6
   2    5    3    7
```

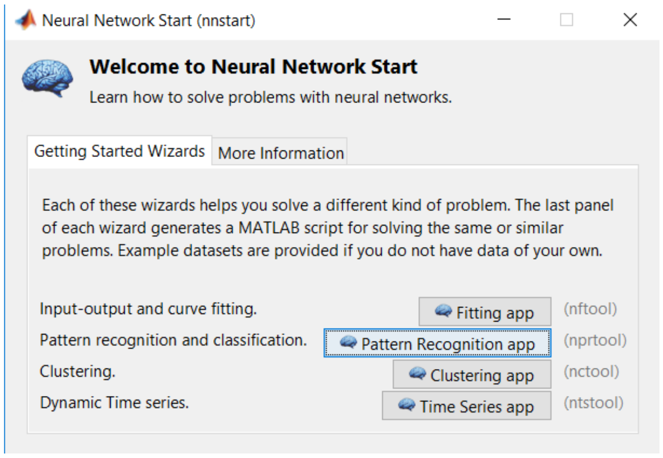
Part2: Neural Network training using Matlab Toolbox/App

1. In part2, you are going to develop the neural network model to recognize the MNIST hand written digits using the Matlab neural network pattern recognition app. Before the training, we need to load the data into the workspace and also convert the y label data into the appropriate form. For the training, y data is represented with 10 output nodes, y $= \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_{10} \end{bmatrix}$.

    The label '1' will be $\begin{bmatrix} 1 \\ 0 \\ \cdots \\ 0 \end{bmatrix}$, the label '2' = $\begin{bmatrix} 0 \\ 1 \\ \cdots \\ 0 \end{bmatrix}$, and so on.

    - Complete the code in section2.

2. Using the Neural Network Pattern Recognition App. If needed, open the Neural Network Start GUI with this command: nnstart



Neural Network Start (nnstart)

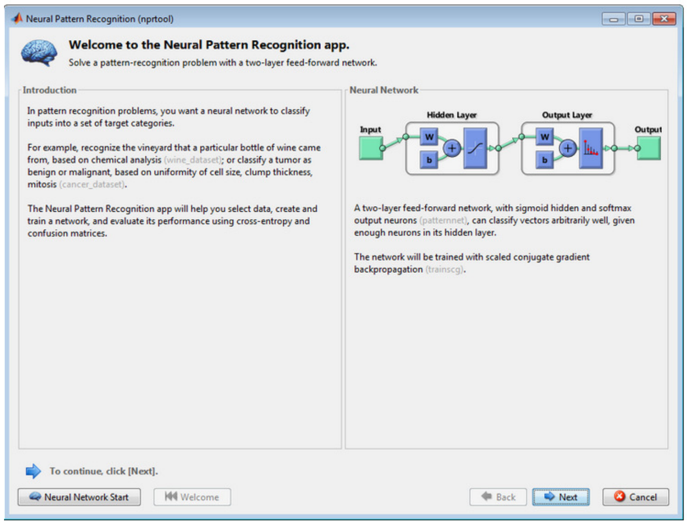
Welcome to Neural Network Start

Learn how to solve problems with neural networks.

Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

Input-output and curve fitting.	Fitting app	(nftool)
Pattern recognition and classification.	Pattern Recognition app	(nprtool)
Clustering.	Clustering app	(nctool)
Dynamic Time series.	Time Series app	(ntstool)

Click **Pattern Recognition app** to open the Neural Network Pattern Recognition app. (You can also use the command `nprtool`.)



Neural Pattern Recognition (nprtool)

Welcome to the Neural Pattern Recognition app.

Solve a pattern-recognition problem with a two-layer feed-forward network.

Introduction

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis (wine_dataset); or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis (cancer_dataset).

The Neural Pattern Recognition app will help you select data, create and train a network, and evaluate its performance using cross-entropy and confusion matrices.

Neural Network

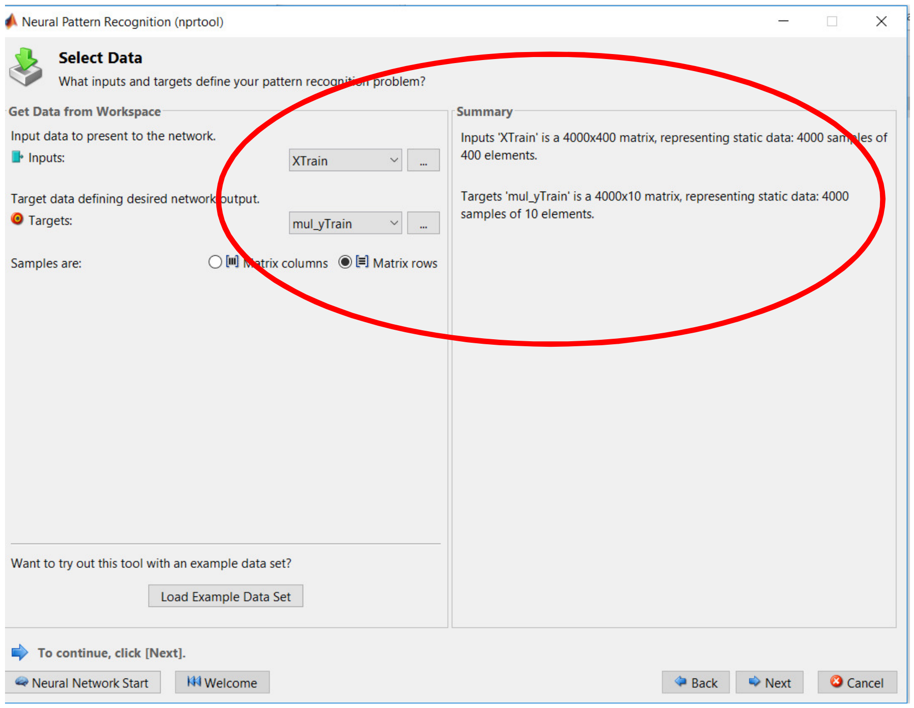
A two-layer feed-forward network, with sigmoid hidden and softmax output neurons (patternnet), can classify vectors arbitrarily well, given enough neurons in its hidden layer.

The network will be trained with scaled conjugate gradient backpropagation (trainscg).

To continue, click [Next].

Click **Next** to proceed. The Select Data window opens

1. Data source & displayData function:modified from Andrew Ng



Neural Pattern Recognition (nprtool)

Select Data

What inputs and targets define your pattern recognition problem?

Get Data from Workspace

Input data to present to the network.

Inputs: XTrain

Target data defining desired network output.

Targets: mul_yTrain

Samples are: ○ Matrix columns ● Matrix rows

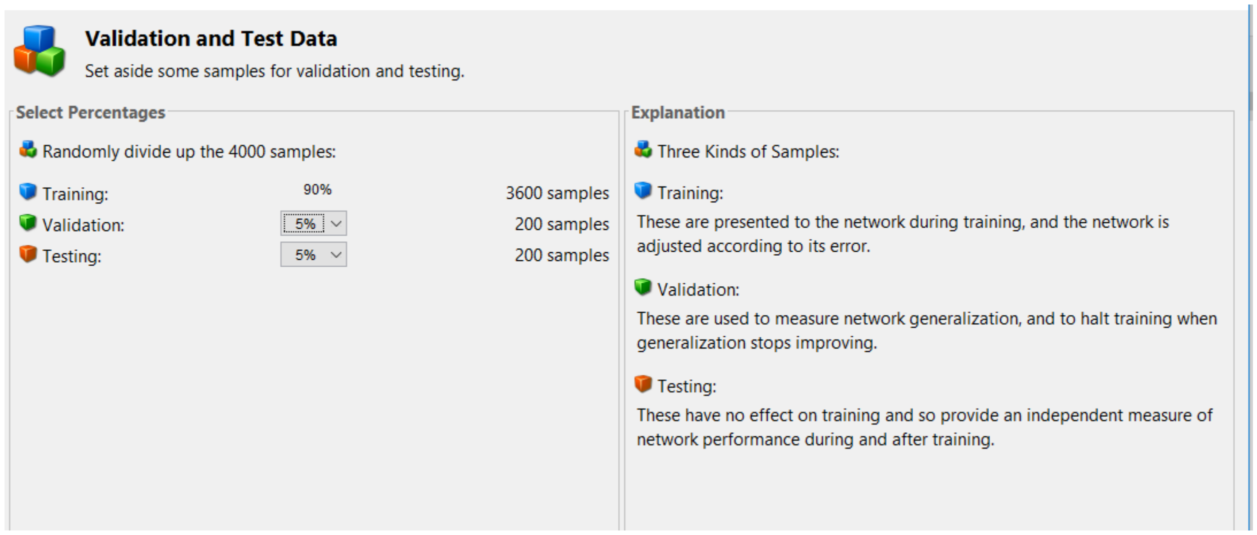
Summary

Inputs 'XTrain' is a 4000x400 matrix, representing static data: 4000 samples of 400 elements.

Targets 'mul_yTrain' is a 4000x10 matrix, representing static data: 4000 samples of 10 elements.

Want to try out this tool with an example data set?

select input and target data as mark in the figure then click next.



Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 4000 samples:

Training:	90%	3600 samples
Validation:	5%	200 samples
Testing:	5%	200 samples

Explanation

Three Kinds of Samples:

Training:
These are presented to the network during training, and the network is adjusted according to its error.

Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

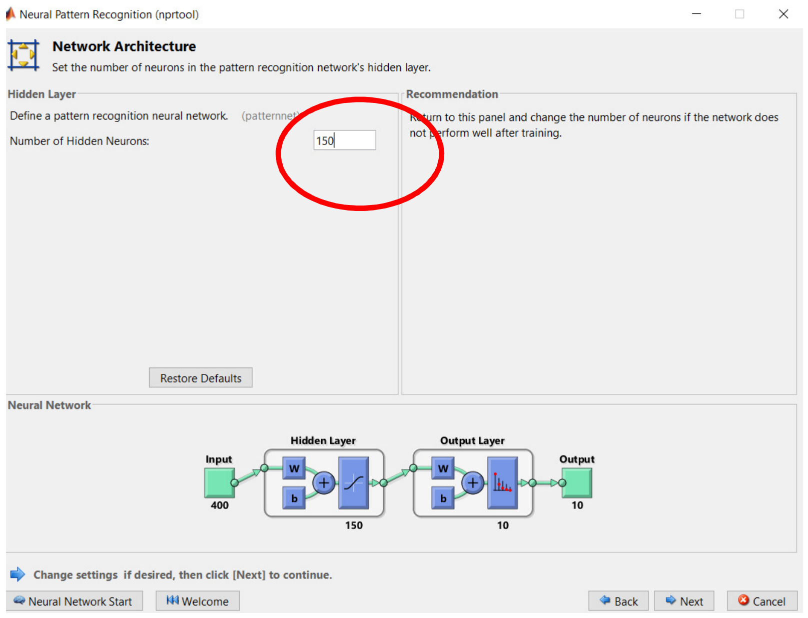
Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Validation and test data sets are each set to 5% of the original data. With these settings, the input vectors and target vectors will be randomly divided into three sets as follows:

- 90% are used for training.
- 5% are used to validate that the network is generalizing and to stop training before overfitting.
- The last 5% are used as a completely independent test of network generalization.

In this app, no options for 0% testing data.

1. Data source & displayData function:modified from Andrew Ng



Neural Pattern Recognition (nprtool)

Network Architecture

Set the number of neurons in the pattern recognition network's hidden layer.

Hidden Layer

Define a pattern recognition neural network. (patternnet)

Number of Hidden Neurons: 150

Recommendation

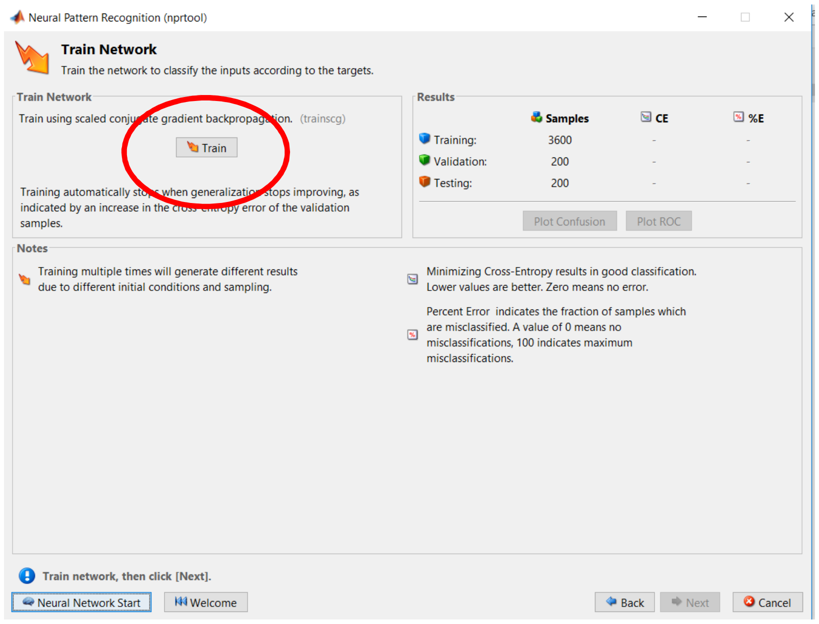
Return to this panel and change the number of neurons if the network does not perform well after training.

Neural Network

Change settings if desired, then click [Next] to continue.

The standard network that is used for pattern recognition is a three-layer feedforward network, with a sigmoid transfer function in the hidden layer, and a output layer. The default number of hidden neurons is set to 10.

You might want to come back and increase this number if the network does not perform as well as you expect. The number of output neurons is set to 10, which is equal to the number of elements in the target vector (the number of categories). Click **Next**.



Neural Pattern Recognition (nprtool)

Train Network

Train the network to classify the inputs according to the targets.

Train Network

Train using scaled conjugate gradient backpropagation. (trainscg)

Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples.

Results

	Samples	CE	%E
Training:	3600	-	-
Validation:	200	-	-
Testing:	200	-	-

Notes

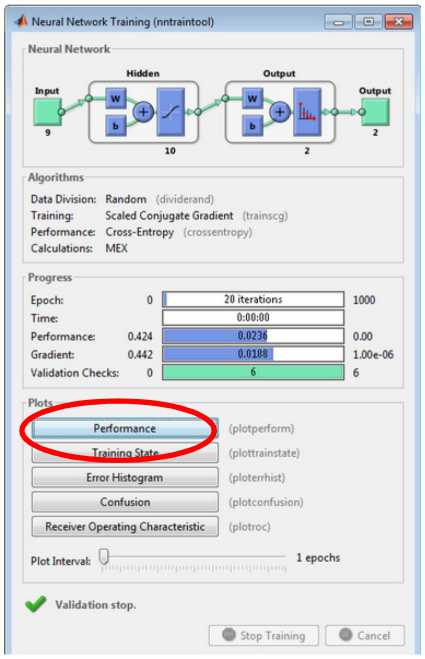
Training multiple times will generate different results due to different initial conditions and sampling.

Minimizing Cross-Entropy results in good classification. Lower values are better. Zero means no error.

Percent Error indicates the fraction of samples which are misclassified. A value of 0 means no misclassifications, 100 indicates maximum misclassifications.

Train network, then click [Next].

Click **Train**


Neural Network Training (nntraintool)

Neural Network

Algorithms

```
Data Division:  Random  (dividerand)
Training:       Scaled Conjugate Gradient  (trainscg)
Performance:    Cross-Entropy  (crossentropy)
Calculations:   MEX
```

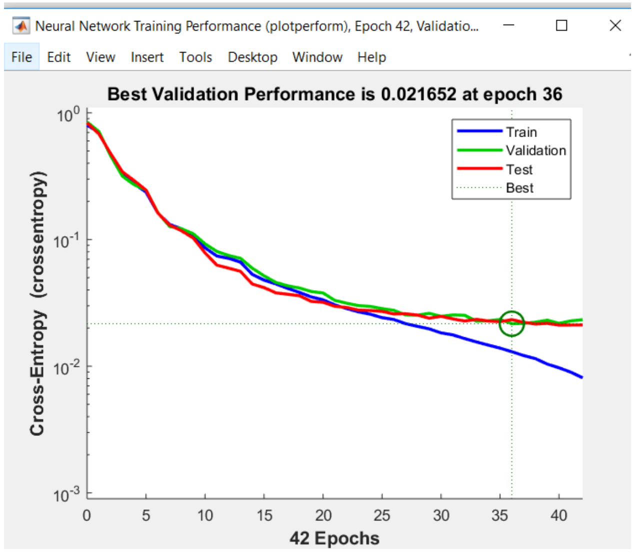
Progress

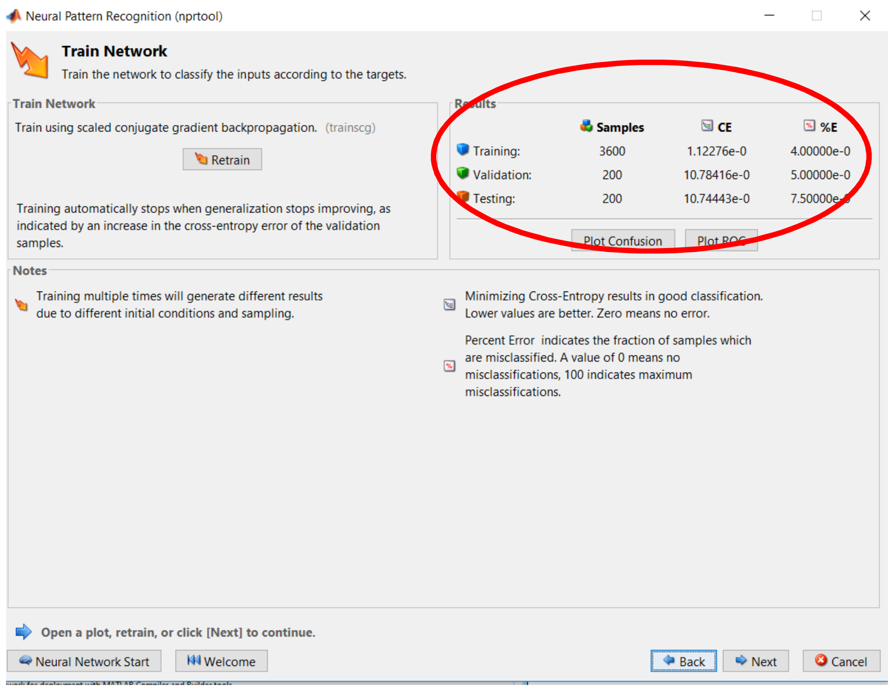
```
Epoch:                0   20 iterations   1000
Time:                     0:00:00
Performance:      0.424   0.0236          0.00
Gradient:         0.442   0.0188          1.00e-06
Validation Checks:    0   6               6
```

Plots

```
Performance                         (plotperform)
Training State                      (plottrainstate)
Error Histogram                     (ploterrhist)
Confusion                           (plotconfusion)
Receiver Operating Characteristic   (plotroc)
Plot Interval:                      1 epochs
```

Validation stop.


Neural Network Training Performance (plotperform), Epoch 42, Validatio...


Neural Pattern Recognition (nprtool)

Train Network

Train the network to classify the inputs according to the targets.

Train Network

Train using scaled conjugate gradient backpropagation. (trainscg)

Training automatically stops when generalization stops improving, as indicated by an increase in the cross-entropy error of the validation samples.

Results

	Samples	CE	%E
Training:	3600	1.12276e-0	4.00000e-0
Validation:	200	10.78416e-0	5.00000e-0
Testing:	200	10.74443e-0	7.50000e-0

Notes

Training multiple times will generate different results due to different initial conditions and sampling.

Minimizing Cross-Entropy results in good classification. Lower values are better. Zero means no error.

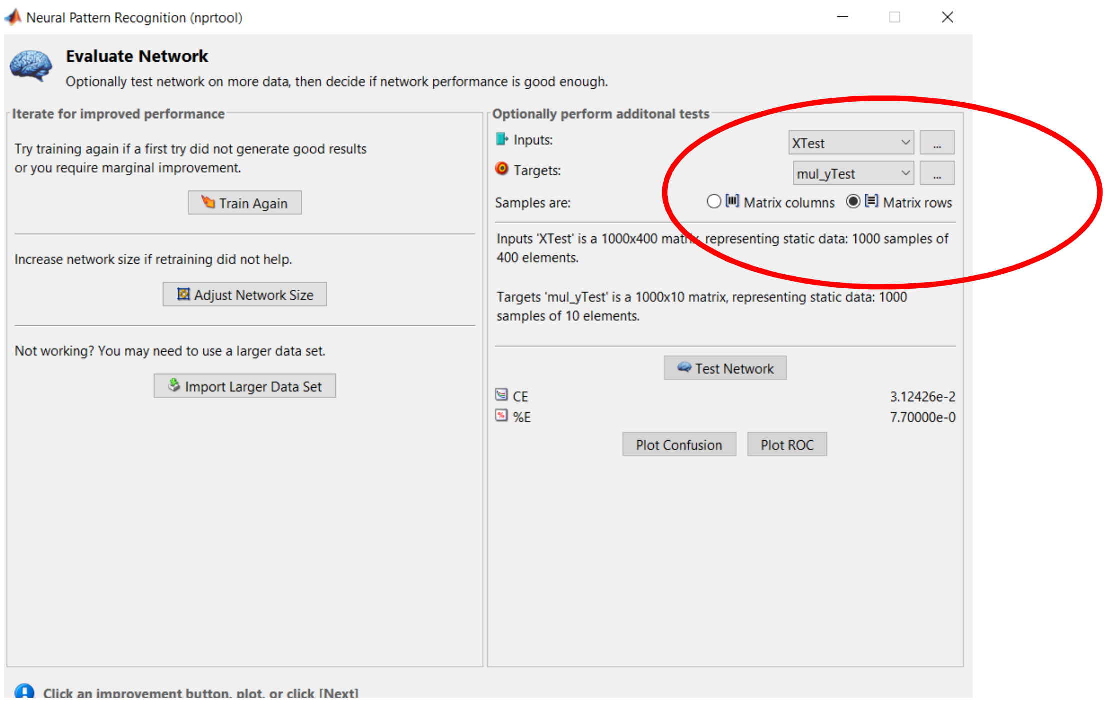
Percent Error indicates the fraction of samples which are misclassified. A value of 0 means no misclassifications, 100 indicates maximum misclassifications.

Open a plot, retrain, or click [Next] to continue.

click next

1. Data source & displayData function:modified from Andrew Ng

At this point, you can test the network against new data


Neural Pattern Recognition (nprtool)

Evaluate Network

Optionally test network on more data, then decide if network performance is good enough.

Iterate for improved performance

Try training again if a first try did not generate good results or you require marginal improvement.

Increase network size if retraining did not help.

Not working? You may need to use a larger data set.

Optionally perform additonal tests

Inputs: XTest

Targets: mul_yTest

Samples are: ○ Matrix columns ● Matrix rows

Inputs 'XTest' is a 1000x400 matrix, representing static data: 1000 samples of 400 elements.

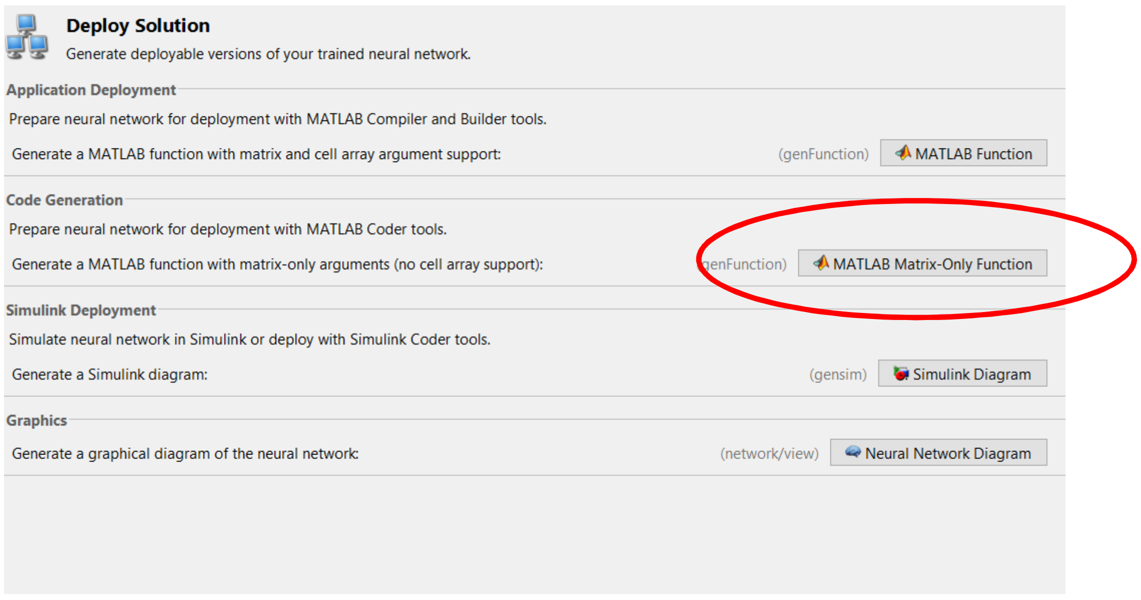
Targets 'mul_yTest' is a 1000x10 matrix, representing static data: 1000 samples of 10 elements.

CE 3.12426e-2

%E 7.70000e-0

Click an improvement button, plot, or click [Next]

You can use the generated code: generate a function by clicking the code generation


Deploy Solution

Generate deployable versions of your trained neural network.

Application Deployment

Prepare neural network for deployment with MATLAB Compiler and Builder tools.

Generate a MATLAB function with matrix and cell array argument support: (genFunction)

Code Generation

Prepare neural network for deployment with MATLAB Coder tools.

Generate a MATLAB function with matrix-only arguments (no cell array support): (genFunction)

Simulink Deployment

Simulate neural network in Simulink or deploy with Simulink Coder tools.

Generate a Simulink diagram: (gensim)

Graphics

Generate a graphical diagram of the neural network: (network/view)

```
function [y1] = myNeuralNetworkFunction(x1)
```

Save the function and use this function for predictions on the new data

1. Data source & displayData function:modified from Andrew Ng

3. Calculate the neural network accuracy on the test data using the function generated by the Matlab app
   - Complete code in section3 to predict the neural network's output on the test data set.
   - **Question 5:** what is the accuracy of the neural network on the test data set?

You can work in groups of up to two people

What to Submit to Blackboard:

- **one** ZIP file that includes :
  2) **A report (10pts): contains answers for Question1 –Question5.**
  3) (60 pts): The live script file, "Lab5_part1.mlx", in that all outputs are generated either inline or on the right side.
  4) (30 pts): The live script file, "Lab5_part2.mlx", in that all outputs are generated either inline or on the right side. Please include the function "myNeuralNetworkFunction.m"

- **Pay attention to the zip file name convention:**
  Lab5_Student1 Lastname_Student2 Lastname.zip
  Ex) Lab5_Green_Smith.zip