**WIX3001 Soft Computing**

**Tutorial 1: Introduction to Neural Networks**

In this tutorial, you will learn how to use a simple neural network for learning and classification, using MATLAB.

You can use MATLAB for free by registering at Mathworks with your Siswamail.

The web version of MATLAB can be used at: <https://matlab.mathworks.com/>

**Loading and preparing the dataset**

1. Get the MNIST dataset (train.csv and test.csv) from this url:

<https://www.kaggle.com/c/digit-recognizer/data>

1. Load the datasets into MATLAB:

|  |
| --- |
| tr = csvread('train.csv', 1, 0); % read train.csv  sub = csvread('test.csv', 1, 0); % read test.csv |

1. Visualize some of the digits:

|  |
| --- |
| figure % plot images  colormap(gray) % set to grayscale  for i = 1:25 % preview first 25 samples  subplot(5,5,i) % plot them in 6 x 6 grid  digit = reshape(tr(i, 2:end), [28,28])'; % row = 28 x 28 image  imagesc(digit) % show the image  title(num2str(tr(i, 1))) % show the label  end |

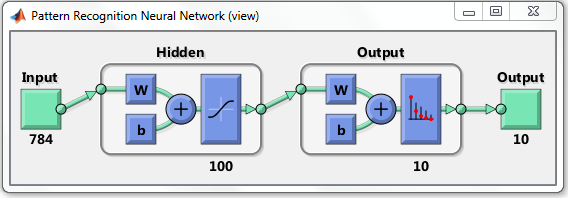
1. You will be using the [nprtool](https://www.mathworks.com/help/nnet/ref/nprtool.html) pattern recognition app from [Deep Learning Toolbox](https://www.mathworks.com/products/deep-learning.html). The app expects two sets of data:
   1. Input: a numeric matrix, rows representing features and columns representing samples.
   2. Targets: a numeric matrix of 0 and 1 that maps to specific labels that images represent. This is also known as a dummy variable. Deep Learning Toolbox also expects labels stored in columns, rather than in rows.
2. The dataset stores samples in rows rather than in columns, so you need to transpose it. Then you will partition the data so that you hold out 1/3 of the data for model evaluation, and you will only use 2/3 for training our artificial neural network model.

|  |
| --- |
| n = size(tr, 1); % number of samples in the dataset  targets = tr(:,1); % 1st column is |label|  targets(targets == 0) = 10; % use '10' to present '0'  targetsd = dummyvar(targets); % convert label into a dummy variable  inputs = tr(:,2:end); % the rest of columns are predictors  inputs = inputs'; % transpose input  targets = targets'; % transpose target  targetsd = targetsd'; % transpose dummy variable  rng(1); % for reproducibility  c = cvpartition(n,'Holdout',n/3); % hold out 1/3 of the dataset  Xtrain = inputs(:, training(c)); % 2/3 of the input for training  Ytrain = targetsd(:, training(c)); % 2/3 of the target for training  Xtest = inputs(:, test(c)); % 1/3 of the input for testing  Ytest = targets(test(c)); % 1/3 of the target for testing  Ytestd = targetsd(:, test(c)); % 1/3 of the dummy variable for testing |

**Initializing the neural network**

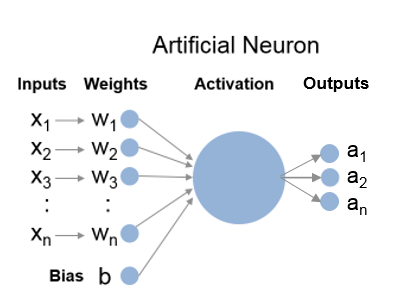
1. You can start the Neural Network Start GUI by typing the command [nnstart](https://www.mathworks.com/help/nnet/ref/nnstart.html).
2. You then click the Pattern Recognition Tool to open the Neural Network Pattern Recognition Tool. You can also use the command [nprtool](https://www.mathworks.com/help/nnet/ref/nprtool.html) to open it directly.
3. Click "Next" in the welcome screen and go to "Select Data".
4. For inputs, select Xtrain and for targets, select Ytrain.
5. Click "Next" and go to "Validation and Test Data". Accept the default settings and click "Next" again. This will split the data into 70-15-15 for the training, validation and testing sets.
6. In the "Network Architecture", change the value for the number of hidden neurons, 100, and click "Next" again.
7. In the "Train Network", click the "Train" button to start the training. When finished, click "Next". Skip "Evaluate Network" and click next.
8. In "Deploy Solution", select "MATLAB Matrix-Only Function" and save the generated code (i.e. [myNNfun.m](https://blogs.mathworks.com/images/loren/2015/myNNfun.m)).
9. If you click "Next" and go to "Save Results", you can also save the script as well as the model you just created (i.e. [myNNscript.m](https://blogs.mathworks.com/images/loren/2015/myNNscript.m)).

Here is the diagram of this artificial neural network model you created with the Pattern Recognition Tool. It has 784 input neurons, 100 hidden layer neurons, and 10 output layer neurons.



The model learns through training the weights to produce the correct output.

***W*** in the diagram stands for weights and ***b*** for bias units, which are part of individual neurons. Individual neurons in the hidden layer look like this - 784 inputs and corresponding weights, 1 bias unit, and 10 activation outputs.



Visualizing the learned weights

If you look inside **myNNfun.m**, you see variables like IW1\_1 and x1\_step1\_keep that represent the weights your artificial neural network model learned through training. Because we have 784 inputs and 100 neurons, the full layer 1 weights will be a 100 x 784 matrix. Let's visualize them. This is what our neurons are learning!

|  |
| --- |
| load myWeights % load the learned weights  W1 =zeros(100, 28\*28); % pre-allocation  W1(:, x1\_step1\_keep) = IW1\_1; % reconstruct the full matrix  figure % plot images  colormap(gray) % set to grayscale  for i = 1:25 % preview first 25 samples  subplot(5,5,i) % plot them in 6 x 6 grid  digit = reshape(W1(i,:), [28,28])'; % row = 28 x 28 image  imagesc(digit) % show the image  end |

**Computing Categorization Accuracy**

Now you are ready to use myNNfun.m to predict labels for the heldout data in Xtest and compare them to the actual labels in Ytest. That gives you a realistic predictive performance against unseen data.

First, you see the actual output from the network, which shows the probability for each possible label. You simply choose the most probable label as your prediction and then compare it to the actual label. You should see 95% categorization accuracy.

|  |
| --- |
| Ypred = myNNfun(Xtest); % predicts probability for each label  Ypred(:, 1:5) % display the first 5 columns  [~, Ypred] = max(Ypred); % find the indices of max probabilities  sum(Ytest == Ypred) / length(Ytest) % compare the predicted vs. actual |

**Network Architecture**

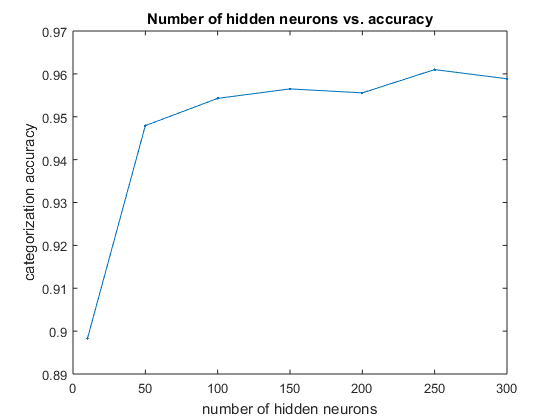
You probably noticed that the artificial neural network model generated from the Pattern Recognition Tool has only one hidden layer. You can build a custom model with more layers if you would like, but this simple architecture is sufficient for most common problems.

The next question you may ask is how I picked 100 for the number of hidden neurons. The general rule of thumb is to pick a number between the number of input neurons, 784 and the number of output neurons, 10, and I just picked 100 arbitrarily. That means you might do better if you try other values. Let's do this programmatically this time. myNNscript.m will be handy for this - you can simply adapt the script to do a parameter sweep.

|  |
| --- |
| sweep = [10,50:50:300]; % parameter values to test  scores = zeros(length(sweep), 1); % pre-allocation  models = cell(length(sweep), 1); % pre-allocation  x = Xtrain; % inputs  t = Ytrain; % targets  trainFcn = 'trainscg'; % scaled conjugate gradient  for i = 1:length(sweep)  hiddenLayerSize = sweep(i); % number of hidden layer neurons  net = patternnet(hiddenLayerSize); % pattern recognition network  net.divideParam.trainRatio = 70/100; % 70% of data for training  net.divideParam.valRatio = 15/100; % 15% of data for validation  net.divideParam.testRatio = 15/100; % 15% of data for testing  net = train(net, x, t); % train the network  models{i} = net; % store the trained network  p = net(Xtest); % predictions  [~, p] = max(p); % predicted labels  scores(i) = sum(Ytest == p) /... % categorization accuracy  length(Ytest);  end |

Let's now plot how the categorization accuracy changes versus number of neurons in the hidden layer.

|  |
| --- |
| figure  plot(sweep, scores, '.-')  xlabel('number of hidden neurons')  ylabel('categorization accuracy')  title('Number of hidden neurons vs. accuracy') |



It looks like you get the best result around 250 neurons and the best score will be around 0.96 with this basic artificial neural network model.

As you can see, you gain more accuracy if you increase the number of hidden neurons, but then the accuracy decreases at some point (your result may differ a bit due to random initialization of weights). As you increase the number of neurons, your model will be able to capture more features, but if you capture too many features, then you end up overfitting your model to the training data and it won't do well with unseen data. Let's examine the learned weights with 300 hidden neurons. You see more details, but you also see more noise.

|  |
| --- |
| net = models{end}; % restore the last model  W1 = zeros(sweep(end), 28\*28); % pre-allocation  W1(:, x1\_step1\_keep) = net.IW{1}; % reconstruct the full matrix  figure % plot images  colormap(gray) % set to grayscale  for i = 1:25 % preview first 25 samples  subplot(5,5,i) % plot them in 6 x 6 grid  digit = reshape(W1(i,:), [28,28])'; % row = 28 x 28 image  imagesc(digit) % show the image  end |