MNIST Handwriting Recognition

Author: Seoyoung Park

Abstract

The first mathematical model of the Neocognition in 1980 has led to deep convolution Neural Networks(DCNNs). DCNNs has been succeed by the growing power of computational method and very big data sets with labels using the advantage of multi-layer architecture. DCNNs has affected the field of computer vision by manipulating matrices in many useful computer vision task used for classification and identification.

Introduction and Overview

In this paper, we are going to build neural networks. The goal of the neural networks is mapping a set of input data to a classification. For this paper, MNIST handwritten digit data set is used. The MNIST database of handwritten digits has 60,000 samples for a training set, and 10,000 samples for a test set. It is a subset of a larger set NIST. This paper will demonstrate neural networks for both single layer and multiple layers by using Pattern Recognition app from neural network toolbox in MATLAB.

Theoretical Background

The general architecture of Neural Networks is

$$A_j argmin(f_M(A_M, ...f_2(A_2, (f1(A_1, x))...) + \lambda g(x)) \tag{1}$$

where A_k represents the weights connecting kth layer of neural network to (k+1)th layer.

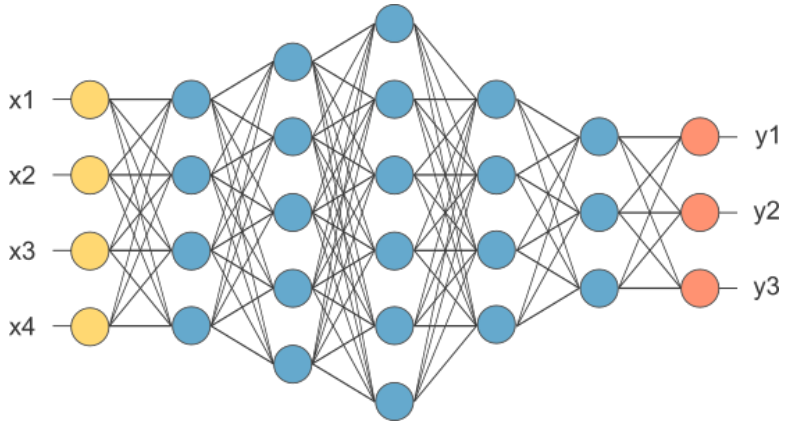

Figure 1: Illustrates the mapping of a Neural Net architecture.

For classification, we train the NN to map the data x_j to its accurate label y_j. The input if the dimension of the data's raw $x_j \in R^n$, and the output, perceptron, is the dimension of the designed classification space. For a linear mapping between layers,

$$\begin{aligned} x^{(1)} &= A_1 x \\ x^{(2)} &= A_2 x^{(1)} \\ y &= A_3 x^{(2)} \\ y &= A_3 A_2 A_1 x \end{aligned} \tag{2}$$

In general, when it's scaled to M layers,

$$y = A_M A_{M-1} ... A_2 A_1 x \tag{3}$$

Since it's a highly under-determined system, the mapping should make M different matrices that makes the best mapping. When it comes to nonlinear mappings, the connections between layers are

$$\begin{aligned} x^{(1)} &= f_1(A_1, x) \\ x^{(2)} &= f_2(A_2, x^{(1)}) \\ y &= f_3(A_3, x^{(2)}) \end{aligned} \tag{4}$$

, and it doesn't need constraint. For M layers,

$$y = f_M(A_M m, ... f_2(A2, (f_1(A_1, x))...) \tag{5}$$

When we consider single layer, each data vector x_j and correct label y_j that corresponds to the data vector give a linear system

$$AX = Y \tag{6}$$

, and to solve this system, we can simply take the pseudo-inverse of the data matrix X so that the

single perceptron lets us build a neural network by least-square fitting.

$$A = YX^{\dagger} \tag{7}$$

Algorithm Implementation and Development

1. Load the training and test MNIST data(csv files) into MATLAB

The first column of the data is the label that represents the correct digit of each sample. In the remaining columns, each row represents a image of handwritten digit in a size of 28×28.

2. To visualize the digits, reshape each row into 28×28 matrix.

3. To use Neural Network Toolbox app, make inputs and targets.

Pattern Recognition app from Neural Network Toolbox needs two sets of data to run: inputs and targets. Input is the images of handwritten digits, so the matrix should have the samples in each column and the features in each row. Target is dummy variables of labels. Since the dataset contains samples in rows, and the app expects the data to be in columns, they are transposed. Partition the data to hold out $1/3$ of the data for evaluation and the remainder for training.

4. Build a single layer network.

Use patternnet to build a classification network with 'trainscg', scaled conjugate gradient back-propagation. Then use train command to classify the images.

5. Calculate the accuracy.

Predicts probability for each label by using net, then find the indices of max probabilities to compare with actual values to get the accuracy percentage.

6. Test it with more layers and see the accuracy for each.

In this paper, number of hidden layer neurons is tested from 50 to 300 by incrementing 50. In each case, 70% of data is used for training, 15% is used for validation, and the remaining 15% is used for testing. After training the network, store the trained network into matrix then calculate the accuracy to compare how the number of hidden layer affects the accuracy.

Computational Results

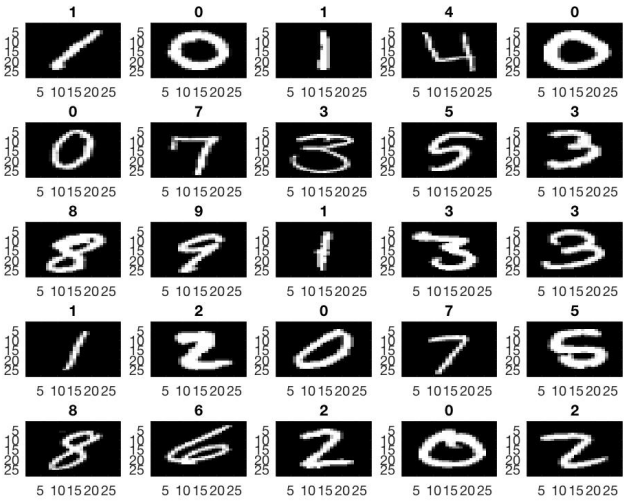

Figure 2: Preview first 25 samples with images and labels

Single Layer

As shown in figure 3, the training keeps running until the validation error curve gets to a minimum which is the line of best. After the training stops, the trained algorithm is used on test to evaluate performance. The error reduces as epochs of training increase. However, the error on the validation might start to increase due to overfitting of the training data. In this case, 159 epochs is appropriate to achieve the minimum.

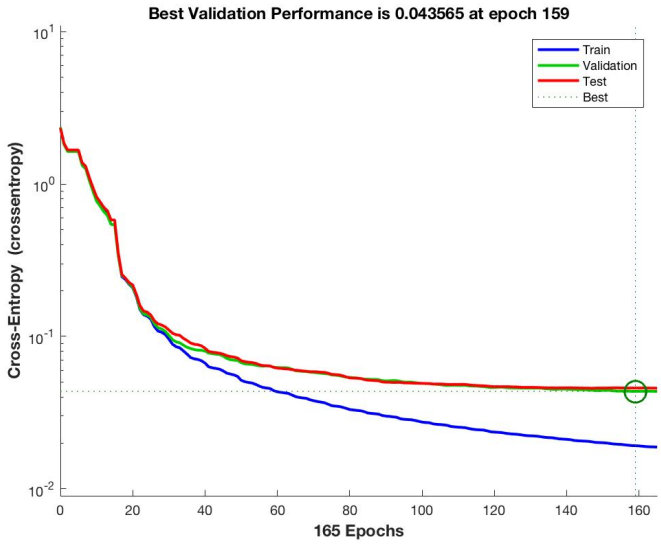

Figure 3: Displays the evolution of loss functions per iteration for training, validation, and test

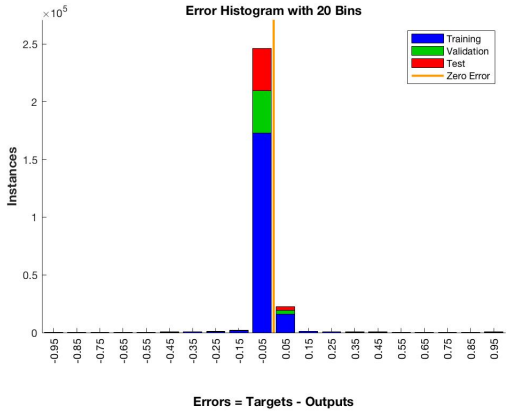

Figure 4: Shows the error performance of the NN architecture for training, validation, and test.The data fitting errors are distributed within a reasonably good range around zero error.

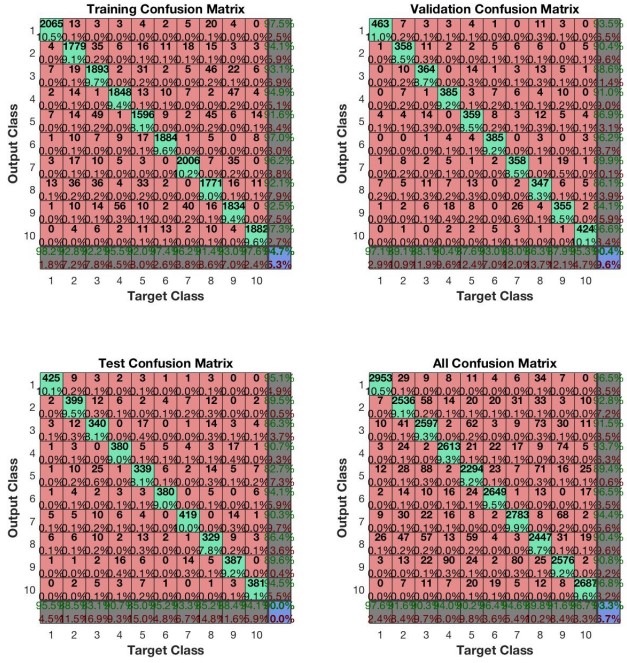

Figure 5: Shows the error performance through confusion matrices of NN for training, validation, test, and overall.

After training, categorization accuracy for single layer comes out to be 89.91%.

Multi-layer

Demonstrated in figure 6, as the number of hidden neurons get bigger, the categorization accuracy gets higher, which implies that more layers result in better categorization.

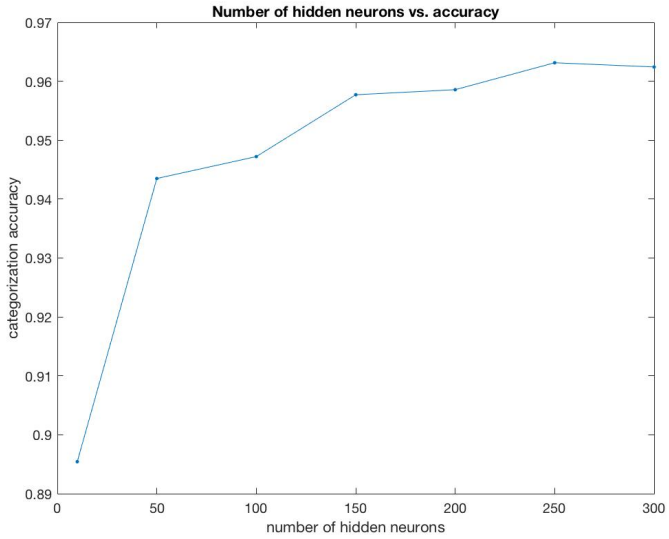

Figure 6: Shows how the number of hidden neurons affects the accuracy of categorization.

Summary and Conclusions

From MNIST handwritten digits data set, neural network is constructed. We used Neural Network toolbox in MATLAB to train the data for both single layer and multi-layer. The categorization accuracy gets higher as the number of layers gets larger. Overall, the Neural Network is able to classify given data set well.

Reference

OpenNN | Open Neural Networks Library. (n.d.). Retrieved March 09, 2018, from http://www.opennn.net/

Appendix A

M = csvread('FILENAME'): reads a comma separated value formatted file FILENAME.
colormap(MAP): sets the current figure's colormap to MAP.
reshape(X,M,N) or reshape(X,[M,N]): returns the M-by-N matrix whose elements are taken columnwise from X.
imagesc: Display image with scaled colors
num2str: Convert numbers to character representation

X=dummyvar(GROUP): returns a matrix X containing zeros and ones, whose columns are dummy variables for the grouping variable GROUP.
rng(SD): seeds the random number generator using the non-negative integer SD so that RAND, RANDI, and RANDN produce a predictable sequence of numbers.
C = cvpartition(N,'KFold',K): creates a cvpartition object C defining a random partition for K-fold cross-validation on N observations.
patternnet(hiddenSizes,trainFcn,performFcn): takes a row vector of N hidden layer sizes and a backpropagation training function and returns an N+1 layer pattern recognition network.
NET,TR= train(NET,X,T): takes a network NET, input data X and target data T and returns the network after training it, and a a training record TR.
perform(NET,T,Y,EW): takes a network, targets T and outputs Y, and optionally error weights EW, and returns performance using the network's default performance function NET.
Y,I= max(X): returns the indices of the maximum values in vector I.

Appendix B

```
clear all; close all; clc;
tr = csvread('train.csv', 1, 0); % read train.csv
sub = csvread('test.csv', 1, 0); % read test.csv

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

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 target for training
Ytest = targets(test(c)); % 1/3 of the target for testing
Ytestd = targetsd(:, test(c)); % 1/3 of the dummy variable for testing

net = patternnet([],'trainscg');
%net.layers{}.transferFcn = 'tansig';
net = train(net,Xtrain,Ytrain);
view(net)
y = net(Xtrain);
y2 = net(Xtest);
perf = perform(net,Ytrain,y);
classes2 = vec2ind(y);
classes3 = vec2ind(y2);

[~, y2] = max(y2);            % find the indices of max probabilities
sum(Ytest == y2) / length(Ytest) % compare the predicted vs. actual

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 =Pre 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

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

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

subplot(4,1,1); bar(y(1,:), 'FaceColor', [.6 .6 .6],'EdgeColor','k')
subplot(4,1,2); bar(y(2,:), 'FaceColor', [.6 .6 .6],'EdgeColor','k')
subplot(4,1,3); bar(y2(1,:), 'FaceColor', [.6 .6 .6],'EdgeColor','k')
subplot(4,1,4); bar(y2(2,:), 'FaceColor', [.6 .6 .6],'EdgeColor','k')
```