Korean J. Math. **26** (2018), No. 2, pp. 337–348
https://doi.org/10.11568/kjm.2018.26.2.337

BEGINNER'S GUIDE TO NEURAL NETWORKS FOR THE MNIST DATASET USING MATLAB

Bitna Kim and Young Ho Park*

Abstract. MNIST dataset is a database containing images of handwritten digits, with each image labeled by an integer from 0 to 9. It is used to benchmark the performance of machine learning algorithms. Neural networks for MNIST are regarded as the starting point of the studying machine learning algorithms. However it is not easy to start the actual programming. In this expository article, we will give a step-by-step instruction to build neural networks for MNIST dataset using MATLAB.

1. Introduction

Machine learning or more generally artificial intelligence is a real hot topic in these days. We have many applications of these principles to areas including virtual assistances, traffic predictions, video surveillance, social media services, email Spam filtering, etc. Many mathematicians want to learn about machine learning, particularly neural networks. There are so many books and internet pages for neural networks scattered around all over the places. However it is hard to find a right and fast track to actual programming for neural networks for beginners. Python is known to be the best programming language for machine learning. But many mathematicians are more familiar with MATLAB than

Received June 7, 2018. Revised June 18, 2018. Accepted June 20, 2018.
2010 Mathematics Subject Classification: 68T10, 68T35.
Key words and phrases: Neural network, Machine learning, Pattern recognition, MNIST dataset, MATLAB.
* Corresponding author.

with Python. It is also true that MATLAB is easier and quicker to learn to apply to neural networks. The MNIST database is commonly used for training image processing systems and is also widely used for training and testing in machine learning. In this expository article, we will give a step-by-step instruction to build neural networks for MNIST dataset using MATLAB. After reading this article, we hope that the readers start to expand their interests to general machine learning algorithms.

2. Preliminaries

2.1. Neural networks. In 1943, Warren McCulloch and Walter Pitts introduced the first *artificial neurons* [10]. Then they showed that networks of these neurons could compute any arithmetic or logical function. In the late 1950s, Frank Rosenblatt [18] and other researchers developed a class of neural networks called *perceptrons*. He proved that the learning rule converges to the correct network weights, if weights exist that solve the problem. Unfortunately, the perceptron network is inherently limited. Many people believed that further research on neural networks was a dead end. For a decade neural network research was almost suspended.

Two new concepts were most responsible for the rebirth of neural networks. The first was the use of *statistical mechanics* to explain the operation of a certain class of recurrent network [4]. The second key development of the 1980s was the *backpropagation algorithm* for training multilayer networks. The most influential publication of the algorithm was by David Rumelhart and James McClelland [19]. These new developments revive the field of neural networks.

Preliminary basic materials for our work on neural networks can be found in many books and web pages. The following list of references may be helpful [2, 3, 7, 9, 12–17, 20, 21, 23].

An artificial neural network is based on a connected units called *a*rtificial neurons, analogous to neurons in an animal brain. Each connection, called a synapse, between neurons can transmit a signal to another neuron. Further, they may have a threshold.

A single-layer network of S neurons with R inputs is shown in the Figure 1. Each of the R inputs is connected to each of the neurons and that the weight matrix has S rows. The individual inputs $p_1, p_2, \ldots, p_R$ enters the network through the weight matrix $\mathbf{W} = (w_{i,j})$. With a bias $\mathbf{b} = (b_i)$, the net output can be written as $\mathbf{a} = f(\mathbf{W}\mathbf{p} + \mathbf{b})$.

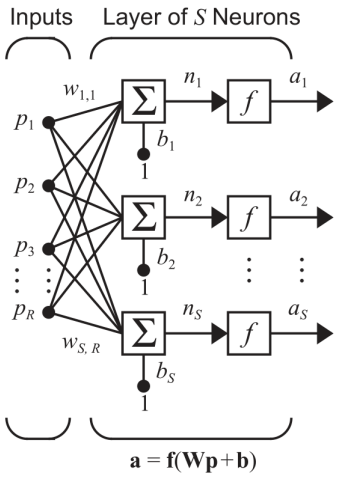

FIGURE 1. Single layer network

A network can have several layers. Each layer has its own weight matrix **W**, its own bias vector **b**, a net input vector **n** and an output vector **a**. The leftmost layer of the network is called the **input layer**. A layer whose output is the network output (i.e., the rightmost layer) is called an **output layer**. The other layers are called **hidden layers**. Multilayer networks are usually more powerful than single-layer networks. For example, a two-layer network can be trained to approximate most functions arbitrarily well but single-layer networks cannot. Single-unit perceptrons are only capable of learning linearly separable patterns; Marvin Minsky and Seymour Papert showed that it was impossible for a single-layer perceptron network to learn an XOR function [11].

The **universal approximation theorem** states that a multilayer perceptron can approximate any continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

One of the first versions of the theorem was proved by George Cybenko for sigmoid activation functions [1]. Kurt Hornik showed in 1991 that it is not the specific choice of the activation function [5].

THEOREM 2.1. *(Universal Approximation Theorem) Let φ be a non-constant bounded and monotonically increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. Let $C(I_m)$*

be the space of continuous functions on I_m. Let $f \in C(I_m)$ be any function. Then for any $\epsilon > 0$ there exist an integer N, real constants c_i, b_i and real vectors $\mathbf{W}_i \in \mathbb{R}^m$, where $1 \leq i \leq N$, such that

$$\left| \sum_{i=1}^{N} c_i \varphi(\mathbf{W}_i^T \mathbf{p} + b_i) - f(\mathbf{p}) \right| < \epsilon$$

for all $\mathbf{p} \in I_m$.

2.2. Machine learning. Supervised learning is a type of system in which both input and desired output data are provided. Input and output data are labeled for classification to provide a learning basis for future data processing.

Training data for supervised learning includes a set of examples with paired input subjects and desired output (which is also referred to as the supervisory signal). In supervised learning for image processing, for example, an AI system might be provided with labeled pictures of vehicles in categories such as cars and trucks. After a sufficient amount of observation, the system should be able to distinguish between and categorize unlabeled images. We will use supervised learning algorithms for MNIST dataset.

Unsupervised learning is the training of an artificial intelligence algorithm using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.

3. MATLAB implimentation for MNIST dataset

3.1. Preparing MNIST dataset for MATLAB. The MNIST database (Modified National Institute of Standards and Technology database) is a database of handwritten digits. It is commonly used for training image processing systems. The database is also widely used for training and testing in machine learning.

The MNIST data comes in two parts. The first part contains 60,000 images to be used as training data. The images are greyscale and 28×28 pixels in size. The second part of the MNIST data set is 10,000 images to be used as test data. The test data is used to evaluate how well a neural network has learned to recognize digits.

MNIST dataset can be downloaded from the MNIST web page [12]. There are four zipped files:

- train-images-idx3-ubyte.gz: training set images (9912422 bytes)
- train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
- t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
- t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

The image and label data is stored in a binary format described on the website. These files should be unzipped after downloaded. All files we work with, including MATLAB code files below, will be saved in a fixed directory `mnist`.

We can find two functions `loadMNISTImages` and `loadMNISTLabels` for extracting the data from the database files. MATLAB codes of these functions are available at [16], or directly at [22]. Once we get these help files in the `mnist` directory, we can use them to read the MNIST data into MATLAB as follows. In MATLAB commandline, type:

```
images = loadMNISTImages('train-images.idx3-ubyte');
labels = loadMNISTLabels('train-labels.idx1-ubyte');
tstimages = loadMNISTImages('t10k-images.idx3-ubyte');
tstlabels = loadMNISTLabels('t10k-labels.idx1-ubyte');
```

The variables `images` and `labels` are 784×60000 and 60000×1 matrices, respectively. Each column of `images` is the digit image reshaped to column vector of length 784 from the original 28×28 image. `labels` represents the letter labels for corresponding number images. To view the images we need to reshape them into the square size matrix. Make a new script and run (after reading the data as above) as follows to view first 100 digits:

```
figure
for i = 1:100
   subplot(10,10,i)
   digit = reshape(images(:, i), [28,28]);
   imshow(digit)
   title(num2str(labels(i)))
end
```

3.2. Neural network design. We will use MATLAB (R2016a) to design our neural networks for MNIST. It has a convenient Neural Network Toolbox. There are four ways we can use the Neural Network Toolbox software; Function fitting (`nftool`), Pattern recognition (`nprtool`), Data clustering (`nctool`) and Time series analysis (`ntstool`). We can open any of these tools by the command `nnstart`. The command-line

operations offer more flexibility than the tools, but with some added complexity. In addition, the tools can generate scripts of documented MATLAB code to provide with templates for creating our own customized command-line functions. It is a good idea to use the tools first, and then generate and modify MATLAB scripts.

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories. A two-layer feed-forward network, with sigmoid hidden and softmax output neurons, can classify vectors arbitrarily well, given enough neurons in its hidden layer. We use the `patternnet` in MATLAB. The network will be trained with scaled conjugate gradient backpropagation (trainscg).

`patternnet`(hiddenSizes,trainFcn,performFcn) takes these arguments,

- hiddenSizes: row vector of one or more hidden layer sizes (default = 10)
- trainFcn: Training function (default = 'trainscg')
- performFcn: Performance function (default = 'crossentropy')

The command `[net,tr]=train(net, images, labels)` train the patternnet with the input database 'images' and target 'labels'. `tr` returns the training record (such as net's epoch and perform). The target data for pattern recognition networks should consist of vectors of all zero values except for a 1 in element i, where i is the class they are to represent. I.e.,

$$[0 \ \cdots 0 \ 1 \ 0 \cdots \ 0]^T \leftrightarrow i$$

where 1 is in the i-th position.

As an example, we will create a net for the XOR function. For targets, we will use the column vectors $(0,1)^T, (1,0)^T$ representing F and T, respectively. The input is the matrix $\left(\begin{smallmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{smallmatrix}\right)$, where each column represents $(F,F),(F,T),(T,F),(T,T)$ pair of logical values. The target matrix is $\left(\begin{smallmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{smallmatrix}\right)$, where each column is the truth value of each of input pairs. The patternnet has only a single hidden layer. Its size is given by hiddenSizes. We choose hiddenSize = 3. See Figure 2.

Here is the MATLAB code `net_xor` for the network.

```
train_data=[ 0 0  1 1 ;  0 1 0 1 ];
target=[0 1 1 0 ; 1 0 0 1] ;
x = train_data; t = target;

% Create a Pattern Recognition Network
hiddenSize = 3;
```

```
net = patternnet(hiddenSize);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 100/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 0/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
disp(y);
```

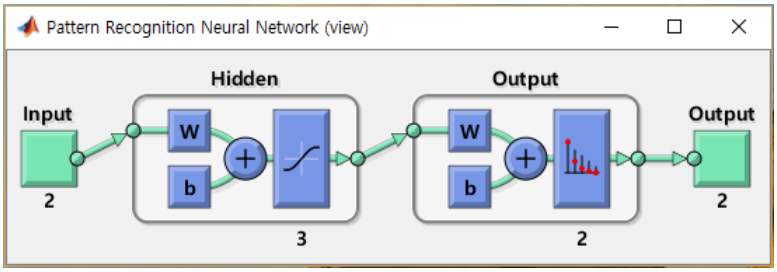

FIGURE 2. Pattern net in MATLAB

The properties of the trained net can be obtained by simply typing the command `net`. Moreover, for example its weights can be obtained by the command `net.IW` and `net.LW`. `net.b` gives the biases of the network. Notice that if we just want the output for some x then we just use `net`(x). So `net([1;0])` returns $(1,0)^T$, which represents T.

3.3. MATLAB implementation. Now we go back to MNIST classification. The digits $0, 1, \cdots, 9$ are used as the labels in a database. Since there are 10 classes to classify, each class is represented by a vector of length 10. The number 0 is labeled by the vector [0 0 $\cdots$ 0 1]. All is done by the command `dummyvar`. Command `vec2ind` gives the labels $1, 2, \cdots, 10$ back.

```
labels = labels';
% dummyvar function doesnt take zeroes
labels(labels==0)=10;
labels=dummyvar(labels);                                    %
```

We will train the network with various number of layers from 10 to 50. We used the default trainFcn and PerformFcn for the remaining arguments. Trainscg denotes the Scaled Conjugate Gradient Descent method, which is a modified Conjugate Gradient Descent algorithm.

First we train the network using all the MNIST training database. The command `net(image)` returns the network's guess of the image label. After we trained the networks, we computed the percent errors with the training data itself and recorded in `percentErrors_netdata`. We also test the network with MNIST test set to get the accuracies. The results are saved into the variable `percentErrors_testdata`. We experiment the networks with different number of hidden layers $10, 20, 30, 40, 50$.

```
% initialize figure
images = loadMNISTImages('train-images.idx3-ubyte');
labels = loadMNISTLabels('train-labels.idx1-ubyte');
tstimages = loadMNISTImages('t10k-images.idx3-ubyte');
tstlabels = loadMNISTLabels('t10k-labels.idx1-ubyte');

labels = labels';
% dummyvar function doesnt take zeroes
labels(labels==0)=10;
labels=dummyvar(labels)';                                    %

tstlabels = tstlabels';
tstlabels(tstlabels==0)=10;
tstlabels=dummyvar(tstlabels)';

% use scaled conjugate gradient for training
trainFcn = 'trainscg';

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
% net.divideFcn = 'dividerand';  % Divide data randomly
% net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 80/100;
```

```
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 0/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
% Cross-Entropy
net.performFcn = 'crossentropy';

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
'plotconfusion', 'plotroc'};

for i=10:10:50
    hiddenLayerSize = i;
    net = patternnet(hiddenLayerSize, trainFcn);

    % Train the Network
    [net,tr] = train(net,images,labels);

    % Test the Network with the MNIST training data
    y = net(images);
    performance_netdata(i/10) = perform(net,labels,y);
    tind = vec2ind(labels);
    yind = vec2ind(y);
    percentErrors_netdata(i/10) = sum(tind ~= yind)/numel(tind);

    % Test the Network with MNIST test data
    tsty = net(tstimages);
    performance_testdata(i/10) = perform(net,tstlabels,tsty);
    tind = vec2ind(tstlabels);
    yind = vec2ind(tsty);
    percentErrors_testdata(i/10) = sum(tind ~= yind)/numel(tind);
end
```

While the net is under the training we can monitor its training status through many plots as in Figure 3.

When we execute the program, we obtain the percent errors with the test data as follows:

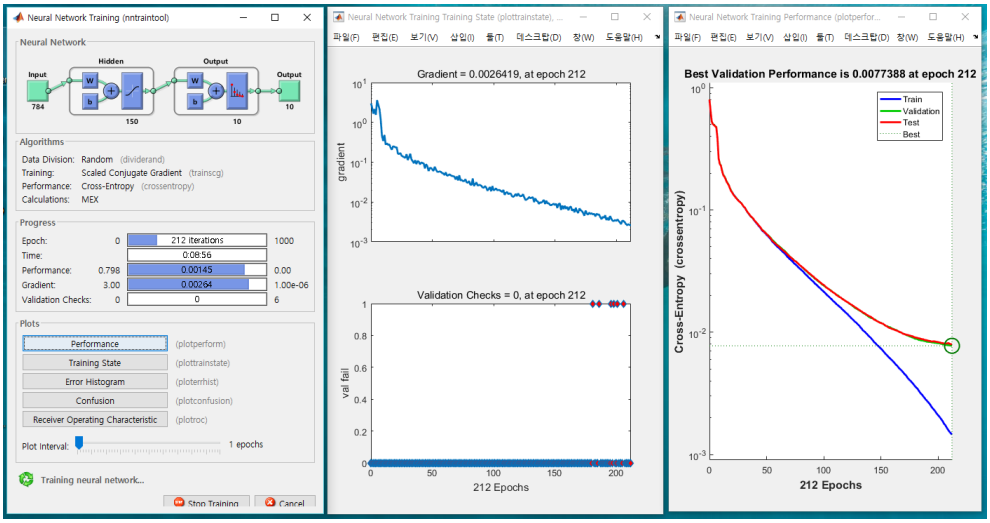

FIGURE 3. Minitoring plots

number of hidden layers	10	20	30	40	50
percent error	0.079	0.062	0.056	0.049	0.043

The error rates get better with the number of hidden layers up to a certain stage. However, it is not always increasing with number of hidden layers. As the number grows, the run time also grows fast. We could run the program with $i = 150$ by a PC with 8GB memory at a reasonable time. From this point, you can experiment with various modifications [6] or learn more about the distinct algorithms such as convolutional networks. According to [12], the best test error rate was achieved by the convolutional network in 2012.

References

[1] G. Cybenko, *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems **2** (4) (1989), 303–314.

[2] M.T. Hagan, M.H Beale, H.B. Demuth and O.D Jesús, *Neural network Design*, 2nd Ed.

[3] M.T. Hagan, *Neural network design*, free book from http://hagan.okstate.edu/NNDesign.pdf

[4] J. J. Hopfield, *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences **79** (1982), 2554–2558.
[5] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks **4** (2) (1991), 251–257.
[6] Bitna Kim, *Handwritten digits classification by neural networks with small data*, Master's thesis, Kangwon National University, 2018.
[7] P. Kim, *Matlab deep learning*, Apress, 2017
[8] T. Kohonen, *Correlation matrix memories*,IEEE Transactions on Computers **21** (1972), 353–359.
[9] Mathworks, *MATLAB documentation*, MATLAB version R2016a, 2016
[10] W.S. McCulloch and W.H. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bull. Math. Biophysics **5** (1943) 115–133.
[11] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*, M.I.T. Press, Cambridge, 1969
[12] MNIST, http://yann.lecun.com/exdb/mnist/
[13] A. Ng, *Course on machine learing*, Cousera, https://www.coursera.org/learn/machine-learning
[14] A.Ng, *CS229 lecture notes*, http://cs229.standford.edu
[15] M. Nielsen, Neural networks and deep learning, http://neuralnetworksanddeeplearning.com
[16] UFLDL, *Using the MNIST dataset*, http://ufldl.stanford.edu/wiki/index.php/Using_the_MNIST_Dataset
[17] T. Rashid, *Make your own neural network*, CreatSpace, 2016
[18] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psycho-logical Review **65** (1958), 386–408.
[19] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Explorations*, Microstructure of Cognition, Vol. 1, Cambridge, MA: MIT Press, 1986.
[20] J.R. Shewchuk, *An introduction to the conjugate gradient method eithout the agonizing pain*, Technical report, Carnegie Mellon University, 1994
[21] UFLDL tutorial, *Unsupervised feature learning and deep learning*, http://deeplearning.stanford.edu/wiki/index.php /UFLDL_Tutorial
[22] http://ufldl.stanford.edu/wiki/resources/mnistHelper.zip
[23] Wikipedia, https://en.wikipedia.org/

Bitna Kim
Department of Mathematics
Kangwon National University
Chuncheon 24341, Korea
E-mail: kbn0884@naver.com

Young Ho Park
Department of Mathematics
Kangwon National University
Chuncheon 24341, Korea
E-mail: yhpark@kangwon.ac.kr