Report

Table of content

[1. Introduction 2](#_Toc533774817)

[2. Data set 3](#_Toc533774818)

[3. Introduction of the algorithm 4](#_Toc533774819)

[4. Design of modules 8](#_Toc533774820)

[5. Result 8](#_Toc533774821)

[6. Limitations and possible improvements 11](#_Toc533774822)

[7. Reference 12](#_Toc533774823)

1. Introduction

This MATLAB program gives an example on artificial neural network. You can design your network as you wish by defining the variable sizes. You can also change the number of epochs by setting the value of the variable: epochs, learning rate by setting the value of variable: eta, and the number of data in a batch by setting the value of the variable: mini\_batch\_size. What’s more, it provides several activation function choices and cost function choices:

% activation\_fn is used to choose activation function

% activation\_fn==1 sigmoid

% activation\_fn==2 hidden layers: soft max;

% output layer: soft max;

% activation\_fn==3 hidden layers: sigmoid;

% output layer: soft max;

% cost\_type is used to choose cost function

% cost\_type=1 Quadratic Cost for sigmoid

% cost\_type=2 Cross Entropy Cost for sigmoid

% cost\_type=3 log-likelihood Cost for soft max

The program includes 2 parts:

* **Network2.m:** the main function
  + **my\_settings.m:** design your network
  + **default\_weight\_initializer:** initialize weight and bias
  + **SGD.m:** stochastic gradient descent
    - **Update\_mini\_batches.m:** update mini batches
      * **Backprop.m:** back propagation
        + **Sigmoid.m:** sigmoid function
        + **Sigmoid\_prime.m:** the derivative of sigmoid function
        + **softmax.m:** soft max function
        + **softmax\_prime.m:** the derivative of soft max function
        + **QuadraticCost\_delta.m:** the derivative of quadratic cost
        + **CrossEntropyCost\_delta:** the derivative of cross entropy cost
    - **QuadraticCost\_fn.m:** quadratic cost
    - **CrossEntropyCost\_fn.m:** cross entropy cost
    - **vectorized\_result.m:** convert labels to vectors. For example, 6=>[0 0 0 0 0 0 1 0 0 0]
* **read\_data.m:** generates mat file we need from data set

<http://yann.lecun.com/exdb/mnist/>

The program has been uploaded to my GitHub account gy233, you can download it from:

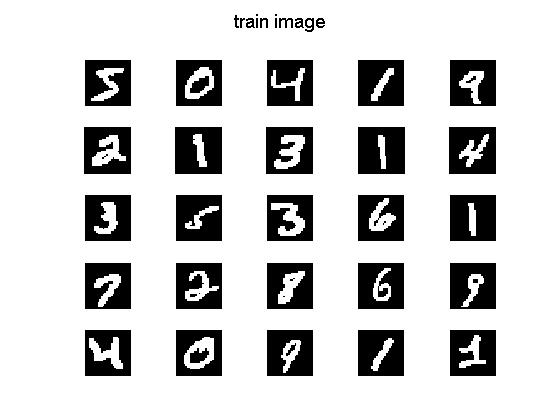
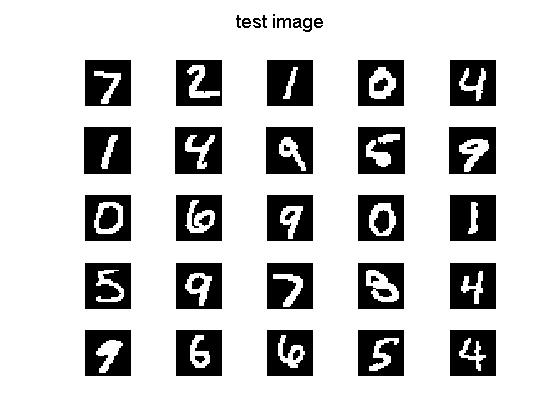
<https://github.com/gy233/MNIST>

1. Data set

* MNIST data set
  + Introduction

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

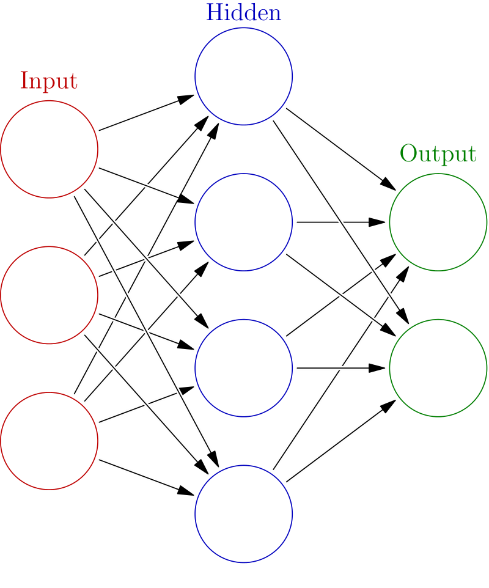
* Four files are available on its site:
* [train-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz):  training set images (9912422 bytes)
* [train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz):  training set labels (28881 bytes)
* [t10k-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz):   test set images (1648877 bytes)
* [t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz):   test set labels (4542 bytes)
  + imshow



1. Introduction of the algorithm

* General introduction to Artificial Neural Networks

Artificial Neural Networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

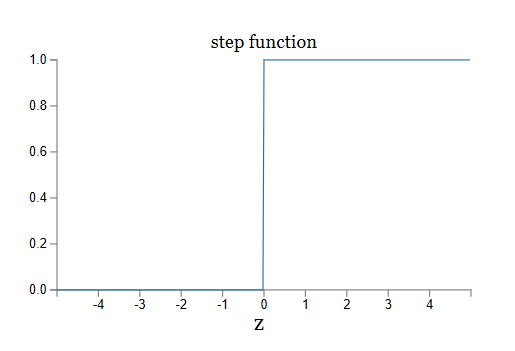


In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called 'edges'. Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

* Introduction to activation functions
  + Perceptron

A perceptron takes several binary inputs, and produces a single binary output:

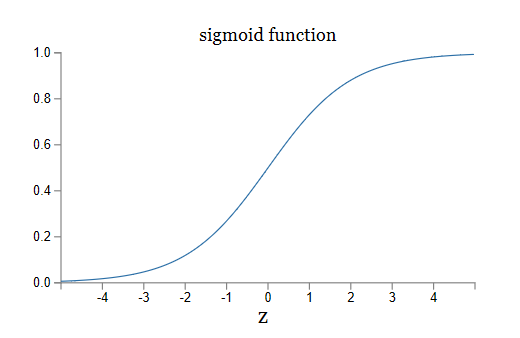
Here is the shape of perceptron.



* + Sigmoid

Since small change in the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip which makes it difficult to see how to gradually modify the weights and biases so that the network gets closer to the desired behavior. We can overcome this problem by introducing a new type of artificial neuron called a *sigmoid* neuron.

Here is the shape of sigmoid function.



* + Soft max

If the output layer was a sigmoid layer, then we certainly couldn't assume that the activations formed a probability distribution. While soft max layer outputs a probability distribution which is a simple interpretation of the output activations.

L represents Lth layer

k is the number of output neurons

It has a good property:

* Introduction to cost functions
  + quadratic cost function

To quantify how well we're achieving this goal we define a cost function

a is the neuron's output;

y is the corresponding desired output;

When activation function is sigmoid function, for every input:

Here, and are vectors. The number of elements of them is depended on then number of neurons of that layer.

* + cross-entropy cost function

Since quadratic cost function has the problem of learning slowdown (this is because that and are related to , when gets small, learning slowdown happens.), we introduce the cross-entropy cost function:

n is the total number of items of training data;

x represents all training inputs.

When activation function is sigmoid function, for every input:

Here, and are vectors. The number of elements of them is depended on then number of neurons of that layer.

This avoids the learning slowdown caused by the term in the analogous equation for the quadratic cost

* + log-likelihood cost function

It is for soft max activation function and it is different from cross-entropy cost and quadratic cost function (only one element in vector ):

y is the corresponding desired output.

When activation function is soft max function, for every input:

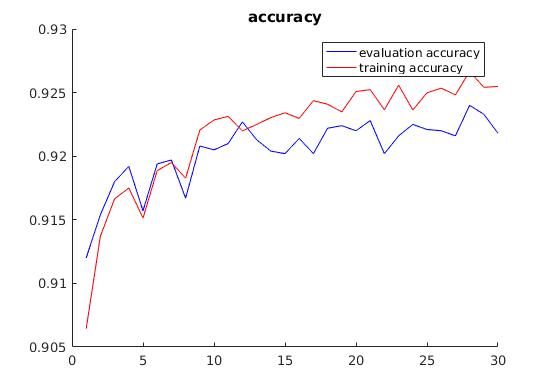
1. Design of modules

For the convenience of users, I define a variable activation\_fn to choose activation function and a variable cost\_type to choose cost function. Users can easily design their own neural network by setting the variable sizes and control epochs by setting the variable epochs. For the convenience of updating weights and bias, these two variables are defined as global variable.

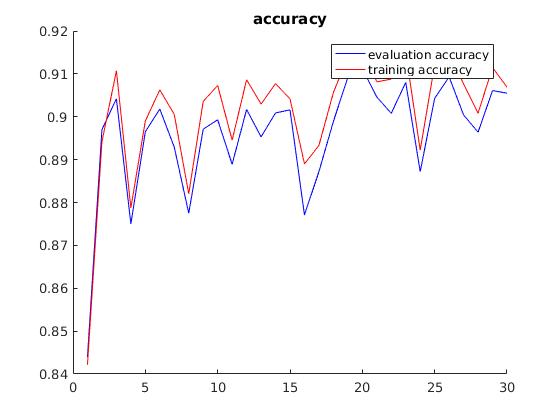
1. Result

First we design a neural network without hidden layers (784 input layers and 10 output layers):

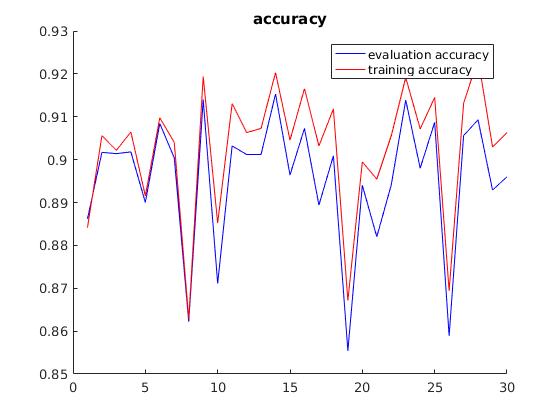
* Activation function: sigmoid; cost function: quadratic cost



* Activation function: sigmoid; cost function: cross entropy cost

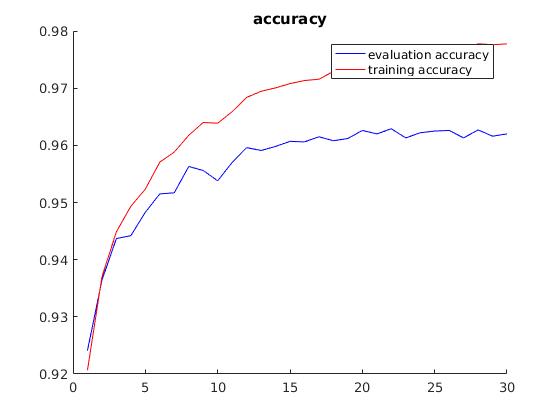


* Activation function: soft max; cost function: log likelihood cost

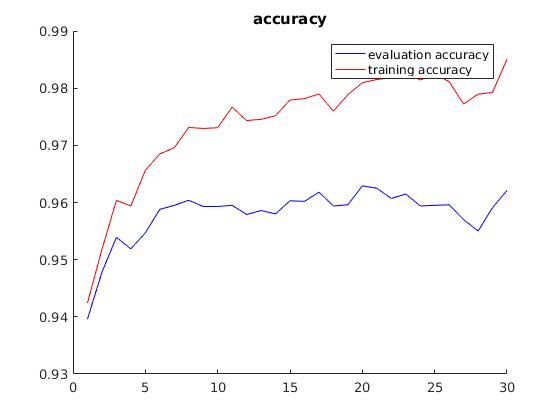


Then we design a neural network with 784 input layers, 30 hidden layers and 10 output layers:

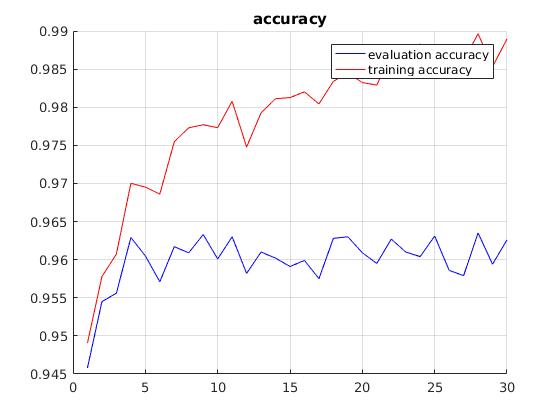
* Activation function: sigmoid; cost function: quadratic cost



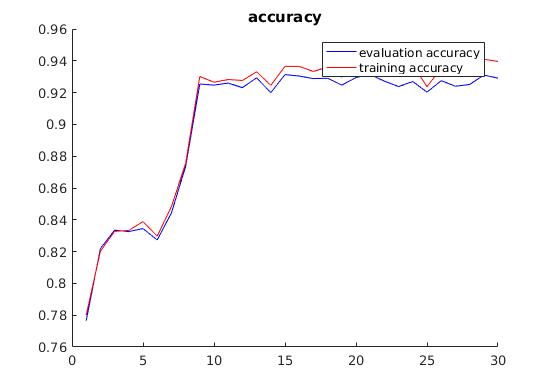
* Activation function: sigmoid; cost function: cross entropy cost



* Last activation function: soft max; the other activation functions: sigmoid; cost function: log likelihood cost



* Last activation function: soft max; the other activation functions: soft max; cost function: log likelihood cost



1. Limitations and possible improvements

* Relu activation function is not supported

1. Reference
2. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Neural Networks and Deep Learning
3. THE MNIST DATABASE of handwritten digits

<http://yann.lecun.com/exdb/mnist/>

1. Artificial neural network

<https://en.wikipedia.org/wiki/Artificial_neural_network>