CSE474/574 Introduction to Machine Learning
Programming Assignment 1
Handwritten Digits Classification

Due Date: March 12^{th} 2014

1 Introduction

In this assignment, your task is to implement different classification methods and compare its performance in classifying handwritten digits. After completing this assignment, you are able to understand:

- How Neural Network works and use Feed Forward, Back Propagation to implement Neural Network
- How K-Nearest Neighbors can be used for classification task
- Evaluating the advantages and disadvantages of above classification methods

To get started with the exercise, you will need to download the supporting files and unzip its contents to the directory you want to complete this assignment.

Warning: In this project, you will have to handle many computing intensive tasks such as training a neural network and prediction in K-Neareast Neighbor. Our suggestion is to use the CSE server Metallica (this server is dedicated to intensive computing tasks) to run your computation. In addition, training such a big dataset will take a very long time, maybe many hours or even days to complete. Therefore, we suggest that you should start doing this project as soon as possible so that the computer will have time to do heavy computaional jobs.

1.1 File included in this exercise

- *mnist_all.mat*: original dataset from MNIST. In this file, there are 10 matrices for testing set and 10 matrices for training set, which corresponding to 10 digits. Among the training set, you have to divide by yourself into validation set and training set.
- *preprocess.m*: performs some preprocess tasks, and output the preprocessed train, validation and test data with their corresponding labels.
- *script.m*: Matlab script for this programming project.
- *sigmoid.m*: compute sigmoid function. The input can be a scalar value, a vector or a matrix.
- *nnObjFucntion.m*: compute the error function of Neural Network.
- *nnPredict.m*: predicts the label of data given the parameters of Neural Network.
- *initializeWeights.m*: return the random weights for Neural Network given the number of node in the input layer and output layer.
- *fmincg.m*: perform optimization task by using conjugate gradient descent.
- *knnPredict.m*: knnPredict predicts the label of given data by using k-nearest neighbor classification algorithm.

1.2 Datasets

The MNIST dataset [1] consists of a training set of 60000 examples and test set of 10000 examples. All digits have been size-normalized and centered in a fixed image of 28×28 size. In original dataset, each pixel in the image is represented by an integer between 0 and 255, where 0 is black, 255 is white and anything between represents different shade of gray.
In many research papers, the official training set of 60000 examples is divided into an actual training set of 50000 examples and validation set of 10000 examples. So we suggest that you should follow this convention.

> **Implementation note:** When loading the dataset to Matlab, you should notice that all entries in the data matrices is in integer format. You can use function *double()* to convert integer matrices to double matrices

2 Your tasks

- Implement **Neural Network** and **K-Nearest Neighbor (kNN)**
- Use validation set to tune hyper-parameters for Neural Network and choose appropriate value k for kNN.
- Write a report to explain the experimental results with these 2 methods.

3 Some practical tips in implementation

3.1 Feature selection

In the dataset, one can observe that there are many features which values are exactly the same for all data points in the training set. With those features, the classification models cannot gain any more information about the difference (or variation) between data points. Therefore, we can ignore those features in the pre-processing step. Another advantage of this pre-processing step is that it will help the linear model run faster by reducing the dimensions without hurting performance of regression model.
Later on in this course, you will learn more sophisticated models to reduce the dimension of dataset.

3.2 Neural Network

3.2.1 Neural Network Representation

Neural network can be graphically represented as in figure 1.
As observed in the figure (1), there are totally 3 layers in the neural network:

- The first layer comprises of $(D + 1)$ units, each represents a feature of image (there is one extra unit representing the bias).
- The second layer in neural network is called the hidden units. In this document, we denote M as the number of hidden units in hidden layer. Hidden units can be considered as the learned features extracted from the original data set. Since number of hidden units will represent the dimension of learned features in neural network, it's our choice to choose an appropriate number of hidden units. Too many hidden units may lead to the slow training phase while too few hidden units may cause the the under-fitting problem.
- The third layer is also called the output layer. The value of i^{th} node in the output layer represents the probability of a certain hand-written image belongs to digit i. Since we have 10 possible digits, there are 10 nodes in the output layer. In this document, we denote K as the number of output units in output layer.

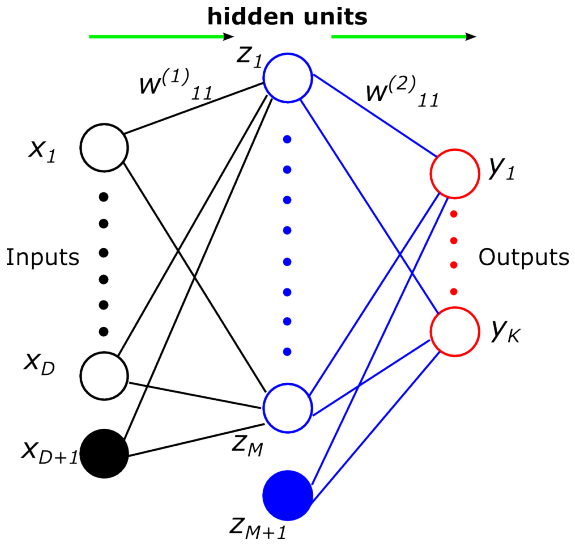

Figure 1: Neural network

The parameters in Neural Network model are the weights of connecting node i in layer l to node j in layer $l+1$. In our standard Neural Network with 3 layers (input, hidden, output), in order to represent the model parameters, we use 2 matrices:

- $W^{(1)} \in \mathbb{R}^{M \times (D+1)}$ is the weight matrix of connections from input layer to hidden layer. j^{th} row in matrix $W^{(1)}$ represents a weight vector of connections from input units to hidden unit j.
- $W^{(2)} \in \mathbb{R}^{K \times (M+1)}$ is the weight matrix of connections from hidden layer to output layer. k^{th} row in matrix $W^{(2)}$ represents a weight vector of connections from hidden units to output unit k.

We also further assume that there are N training samples when performing learning task of Neural Network. In the next section, we will explain how to perform learning in Neural Network.

3.2.2 Feedforward Propagation

In Feedforward Propagation, given parameters of Neural Network and a feature vector x, we want to compute the probability that this feature vector belongs to a particular digit.
Suppose that we have totally M hidden units. Let a_j for j from 1 to M is a linear combination of input data and let z_j is the value of hidden unit j after applying an activation function (in this exercise, we use sigmoid as an activation function). For each hidden unit j ($j = 1, 2, \cdots, M$), we can compute its value as follow:

$$a_j = \sum_{i=1}^{D+1} W^{(1)}_{ji} x_i \tag{1}$$

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)} \tag{2}$$

where $W^{(1)}_{ji}$ is the weight of connection from unit i in input layer to unit j in hidden layer.
The third layer in neural network is called the output units where the learned features in hidden units again be linearly combined to produce the output. Since in this assignment, we want to classify a hand-written digit image to its corresponding class, we can use the one-vs-all binary classification in which each output unit k ($k = 1, 2, \cdots, 10$) in neural network represents the probability of an image belongs to a particular digit. For this reason, the total number of output unit is $K = 10$. Concretely, for each output unit k ($k = 1, 2, \cdots, 10$), we can compute its value as follow:

$$b_k = \sum_{j=1}^{M+1} W^{(2)}_{kj} z_j \tag{3}$$

$$y_k = \sigma(b_k) = \frac{1}{1 + \exp(-b_j)} \tag{4}$$

Now we have finished the Feedforward Propagation.

3.2.3 Error function and Backpropagation

The error function in this case is the negative log-likelihood error function which can be written as follow:

$$E(W) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}(t_{nk}\ln y_{nk} + (1 - t_{nk})\ln(1 - y_{nk})) \tag{5}$$

where t_{nk} indicates the k^{th} target value in 1-of-K coding scheme[1] of input data n and y_{nk} is the k^{th} value of output of data.
Because of the form of error function in equation (5), we can separate its error function in terms of error for each input data n:

$$E(W) = \frac{1}{N}\sum_{n=1}^{N} E_n(W) \tag{6}$$

where

$$E_n(W) = -\sum_{k=1}^{K}(t_{nk}\ln y_{nk} + (1 - t_{nk})\ln(1 - y_{nk})) \tag{7}$$

One way to learn the model parameters in neural networks is to initialize the weights to some random numbers and compute the output value (feed-forward), then compute the error in prediction, transmits this error backward and update the weights accordingly (error backpropagation).
The feed-forward step can be computed directly using formula (1), (2), (3) and (4).
On the other hand, the error backpropagation step requires computing the derivative of error function with respect to the weight.
Consider the derivative of error function with respect to the weight from the hidden unit j to output unit k where $j = 1, 2, \cdots, M+1$ and $k = 1, \cdots, 10$:

$$\frac{\partial E_n}{\partial W_{kj}^{(2)}} = \frac{\partial E_n}{\partial y_k}\frac{\partial y_k}{\partial b_k}\frac{\partial b_k}{\partial W_{kj}^{(2)}} \tag{8}$$

$$= \delta_k z_j \tag{9}$$

where

$$\delta_k = \frac{\partial E_n}{\partial y_k}\frac{\partial y_k}{\partial b_k} = -(\frac{t_k}{y_k} - \frac{1 - t_k}{1 - y_k})(1 - y_k)y_k = y_k - t_k$$

On the other hand, the derivative of error function with respect to the weight from the input unit i to output unit j where $i = 1, 2, \cdots, D+1$ and $k = 1, \cdots, M$ can be computed as follow:

$$\frac{\partial E_n}{\partial W_{ji}^{(1)}} = \sum_{k=1}^{K}\frac{\partial E_n}{\partial y_k}\frac{\partial y_k}{\partial b_k}\frac{\partial b_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial W_{ji}^{(1)}} \tag{10}$$

$$= \sum_{k=1}^{K}\delta_k W_{kj}^{(2)}(1 - z_j)z_j x_i \tag{11}$$

$$= (1 - z_j)z_j(\sum_{k=1}^{K}\delta_k W_{kj}^{(2)})x_i \tag{12}$$

[1] for each data sample, its 1-of-K coding target vector is a vector of length K in which k^{th} entry is 1 if k is the class label of that data sample and all other entries are 0

After finish computing the derivative of error function with respect to weight of each connection in neural network, we now can write the formula for the gradient of error function:

$$\nabla E(W) = \frac{1}{N} \sum_{n=1}^{N} \nabla E_n(W) \tag{13}$$

where w is the loose term indicating the weight of all connections in neural network. In fact, w can be represented as two weight matrices, one for the weights of connections from input layer to hidden layer, and another one for the weights of connections from hidden layer to output layer.
We again can use the gradient descent to update the weight with the following rule:

$$W^{new} = W^{old} - \gamma \nabla E(W^{old}) \tag{14}$$

3.2.4 Regularization in Neural Network

In order to avoid overfitting problem (the learning model is best fit with the training data but give poor generalization when test with validation data), we can add a regularization term into our error function to control the magnitude of parameters in Neural Network. Therefore, our objective function can be rewritten as follow:

$$\tilde{E}(W) = E(W) + \frac{\lambda}{2N} \left(\sum_{j=1}^{M} \sum_{i=1}^{D+1} (W_{ji}^{(1)})^2 + \sum_{k=1}^{K} \sum_{j=1}^{M+1} (W_{kj}^{(2)})^2 \right) \tag{15}$$

where λ is the regularization coefficient.
With this new objective function, the partial derivative of new objective function with respect to weight from hidden layer to output layer can be calculated as follow:

$$\frac{\partial \tilde{E}}{\partial W_{kj}^{(2)}} = \frac{1}{N} \left(\sum_{n=1}^{N} \frac{\partial E_n}{\partial W_{kj}^{(2)}} + \lambda W_{kj}^{(2)} \right) \tag{16}$$

Similarly, the partial derivative of new objective function with respect to weight from input layer to hidden layer can be calculated as follow:

$$\frac{\partial \tilde{E}}{\partial W_{ji}^{(1)}} = \frac{1}{N} \left(\sum_{n=1}^{N} \frac{\partial E_n}{\partial W_{ji}^{(1)}} + \lambda W_{ji}^{(1)} \right) \tag{17}$$

With this new formulas for computing objective function (15) and its partial derivative with respect to weights (16) (17) , we can again use gradient descent to find the minimum of objective function.

3.2.5 Matlab implementation of Neural Network

In the supporting files, we have provided the base code for you to complete. In particular, you have to complete the following function in Matlab:

- *sigmoid*: compute sigmoid function. The input can be a scalar value, a vector or a matrix.
- *nnObjFunction*: compute the objective function of Neural Network *with regularization* and the gradient of objective function.
- *nnPredict*: predicts the label of data given the parameters of Neural Network.

Details of how to implement the required functions is explained in Matlab code.

Optimization: In general, the learning phase of Neural Network consists of 2 tasks. First task is to compute the value and gradient of error function given Neural Network parameters. Second task is to optimize the error function given the value and gradient of that error function. As explained earlier, we can use gradient descent to perform the optimization problem. In this assignment, we have provided a function *fmincg* which performs the conjugate gradient descent algorithm to perform optimization task. In principle, conjugate gradient descent is similar to gradient descent but it chooses a more sophisticated learning rate γ in each iteration so that it will converge faster than gradient descent. Details of how to use *fmincg* are explained in the Matlab code.

Since we use regularization in Neural Network to avoid overfitting problem. You are encouraged to change different value of λ to see its effect in prediction accuracy in validation set. Your report should include diagrams to explain the relation between λ and performance of Neural Network. Moreover, by plotting the value of λ with respect to the accuracy of Neural Network, you should explain in your report how to choose an appropriate hyper-parameter λ to avoid both underfitting and overfitting problem.

You are also encouraged to try different number hidden units to see its effect to the performance of Neural Network. Since training Neural Network is very slow, especially when the number hidden units in Neural Network is large. You should try with small hidden units and gradually increase the size and see how it effects the training time. Your report should include some diagrams to explain relation between number of hidden units and training time.

3.3 K-Nearest Neighbor

3.3.1 Model explanation

In k-Nearest Neighbor (kNN), the label of an image will be determined by 2 phases:

- Compute the distance of the given test data point to all data points in the training set.
- The label of given test data point is decided by majority vote among its k nearest data points in the data set. In particular, the test data point's label is assigned to the class most common among its k nearest neighbors. You can use any tie breaking rules when there is the same number of votes.

This method of classification is a typical example of lazy learning in which the learning is performed after query is submitted to the system. One advantage of this method is that it doesn't need to train the data. Given a test image, it will compute the distance of this image to all images in the training set to find the closest point and assign label accordingly. However, one drawback of this method is that running time for prediction phase takes too long because it has to compute the distance of the given image with all images in the data set to find the closest point. Therefore, one needs to consider the advantage and disadvantage of this method before applying it in practice.

3.3.2 Matlab implementation of Nearest Neighbor Classification

Similar to Neural Network, we have provided the base code for you. You are required to complete the function *knnPredict.m* which perform the classification using kNN with Euclidean distance is the metric to measure the distance between two data points.

In order to choose parameter k for kNN, you should perform the classification with different values of k in validation set. A best value of k is the value that give best result on validation set. This final value of k is then chosen to perform classification on testing set.

4 Submission

You are required to submit an only file *proj1.zip* to CSE server by using the following script:
submit cse474 proj1.zip (for undergraduate students)

submit cse574 proj1.zip (for graduate student)

File *proj1.zip* must contain 2 folders: *report* and *code*.

- Folder *report* contains your report file (in pdf format).
- Folder *code* must contains the following files: *preprocess.m*, *sigmoid.m*, *nnObjFunction.m*, *nnPredict.m*, *knnPredict.m* and *params.mat*. File *params.mat* contains the learned parameters of Neural Network and kNN. Concretely, file *params.mat* must contain the following variables: *n_input* (number of nodes in input layer), *n_hidden* (number of nodes in hidden layer), *w1* (matrix of weight $W^{(1)}$ as mentioned in section 3.2.1), *w2* (matrix of weight $W^{(2)}$ as mentioned in section 3.2.1), *lambda* (regularization coeffient λ as mentioned in section 3.2.4) and k (parameter of kNN).[2]

Project report: The hard-copy of report will be collected in class at due date. Your report should include the following:

- Explanation of your understanding of 2 classifiers.
- Explanation of how to choose the hyper-parameters for Neural Network (number of hidden nodes, regularization term λ) and kNN (parameter k).
- Comparison the performance between 2 classification methods (accuracy, learning time).
- Discussion about the advantages and disadvantages of each method.

5 Grading scheme

- Succcessfully implement Neural Network: 40 points (*sigmoid.m* [3 points], *nnObjFunction.m* [30 points], *nnPredict.m* [7 points]).
- Successfully implement Nearest Neighbor Classifier: 10 points (*knnPredict.m* [10 points]).
- Project report: 40 points
  - Explanation with supporting figures of how to choose the hyper-parameter for Neural Network: 20 points
  - Comparison the performance between 2 classification methods: 10 points
  - Discussion about the advantages and disadvantages of each method: 10 points
- Accuracy of classification methods: 10 points

References

[1] LeCun, Yann; Corinna Cortes, Christopher J.C. Burges. "MNIST handwritten digit database".

[2] Bishop, Christopher M. "Pattern recognition and machine learning (information science and statistics)." (2007).

[2] If you want to write more supporting functions to complete the required functions, you should include these supporting functions and a README file which explains your supporting functions.