CSE474 Introduction to Machine Learning
Programming Assignment 2
Neural Networks

Due Date: April 3^{rd} 2019

1 Introduction

In this assignment, your task is to implement a Multilayer Perceptron the neural network and evaluate its performance in classifying handwritten digits. You will also use the same network to analyze a more challenging hand-drawn images dataset and compare the performance of the neural network against a *deep* neural network using the *TensorFlow library*.

After completing this assignment, you are able to understand:

- How the neural network works and use Feed Forward, Back Propagation to implement the neural network?
- How to setup a Machine Learning experiment on real data?
- How *regularization* plays a role in the *bias-variance* tradeoff?
- How to use TensorFlow library to deploy deep neural networks and understand how having multiple hidden layers can improve the performance of the neural network?

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. Our suggestion is to use the CSE server Metallica (this server is dedicated to intensive computing tasks) and CSE server Springsteen (this *boss* server is dedicated to running TensorFlow) to run your computation. YOU MUST USE PYTHON 3 FOR IMPLEMENTATION. 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 computational jobs.

1.1 File included in this exercise

- *mnist_all.pickle*: original dataset from MNIST. In this file, there are 10 matrices for testing set and 10 matrices for training set, which correspond to 10 digits. Additionally, a smaller *mnist_sample.pickle* is provided for initial calibration. Note that, the larger file will take time to run. However, the final report and outputs need to be generated by using the larger *mnist_all.pickle* file.
- *AI_quick_draw.pickle*: quick drawing examples contributed by over 15 million players on the Internet. This data set is created from a subset of the original dataset shared on the Google cloud platform. Since this file is large, you will need to download it from the following link - `https://www.cse.buffalo.edu/ubds/docs/AI_quick_draw.pickle`. Note that this file is 94 MB. If, due to versioning issues, the pickle file is not readable on your system, you may use the provided *create_data_set.py* to recreate the pickle file.

- *nnFunctions.py*: Python script for this programming project. Contains function definitions -
  - *initializeWeights()*: return the random weights for the neural network given the number of unit in the input layer and output layer.
  - *preprocess()*: performs some preprocess tasks, and output the preprocessed train and test data with their corresponding labels.
  - *sigmoid()*: compute sigmoid function. The input can be a scalar value, a vector or a matrix. *You need to make changes to this function.*
  - *nnObjFunction()*: compute the error function of the neural network. *You need to make changes to this function.*
  - *nnPredict()*: predicts the label of data given the parameters of the neural network. *You need to make changes to this function.*
- *nnScript.py*: Python script for this programming project. This function calls the implemented functions in *nnFunctions.py*. *You will need to change the data set name to run the script for different data sets.*
- *deepnnScript.py*: Python script for calling the TensorFlow library for running the deep neural network. *You need to make changes to this function.*

1.2 Datasets

Two data sets will be provided. Both consist of images. See the notebook available here - `https://github.com/ubdsgroup/ubmlcourse/blob/master/notebooks/ProgrammingAssignment2.ipynb`, for pointers about how to handle the data.

1.2.1 MNIST Dataset

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. You will use the training set of 60000 examples training the neural network. The test set will be used to estimate the hyper-parameters of the network (regularization constant λ, number of hidden units).

1.2.2 AI Quick Draw Dataset

The AI quick draw data set[1] used in this project assignment is created from the original Quick Draw data set. While the Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. In this project assignment, the newly created AI quick draw data set has the following properties:

- 10 categories including 'apple', 'basketball', 'arm', 'horse', 'ant', 'axe', 'alarm clock', 'airplane', 'banana' and 'bed'.
- each category includes 12,500 images with the size of 28×28. Therefore, in the data set, each sample is with the dimension of 1×784.
- the data set has already been separated as training features, training labels, testing features and testing labels. In the code, we iteratively use the load function provided by the pickle package to extract those data from the data set.

[1] `https://quickdraw.withgoogle.com/data`

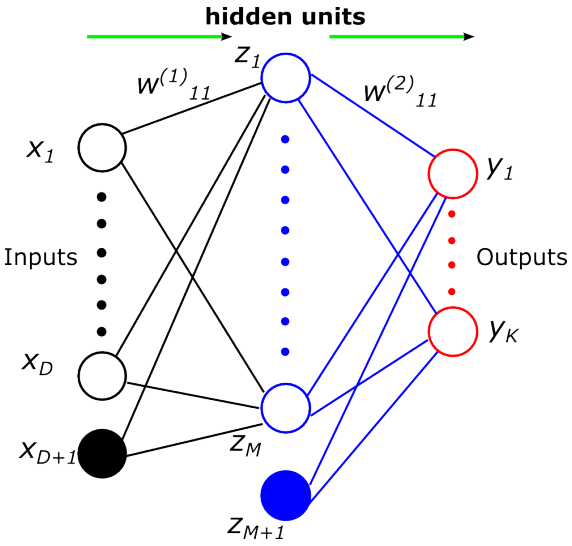

Figure 1: Neural network

2 Your Tasks

- Implement **the neural network** (forward pass and back propagation)
- Incorporate regularization on the weights (λ)
- Use test set to tune hyper-parameters for the neural network (number of units in the hidden layer and λ) (for Handwritten Digits only).
- Use test set to tune hyper-parameters for neural network (number of units in the hidden layer and λ) (for AI Quick Draw Dataset only).
- Run the deep neural network code we provide and compare the results with normal neural network (for AI Quick Draw Dataset only).
- Write a report to explain the experimental results.

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.

Later on in this course, you will learn more sophisticated models to reduce the dimension of dataset (but not for this assignment).

Note: You will need to save the indices of the features that you use and submit them as part of the submission.

3.2 the neural network

3.2.1 the 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 $m+1$ as the number of hidden units in hidden layer. There is an additional bias node at the hidden layer as well. 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 l^{th} unit in the output layer represents the probability of a certain hand-written image belongs to digit l. Since we have 10 possible digits, there are 10 units in the output layer. In this document, we denote k as the number of output units in output layer.

The parameters in the neural network model are the weights associated with the hidden layer units and the output layers units. In our standard the 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. Each row in this matrix corresponds to the weight vector at each hidden layer unit.
- $W^{(2)} \in \mathbb{R}^{k \times (m+1)}$ is the weight matrix of connections from hidden layer to output layer. Each row in this matrix corresponds to the weight vector at each output layer unit.

We also further assume that there are n training samples when performing learning task of the neural network.

In the next section, we will explain how to perform learning in the neural network.

3.2.2 Feedforward Propagation

In Feedforward Propagation, given parameters of the neural network and a feature vector $\mathbf{x}$, 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_{ij} for $1 \leq j \leq m$ be the linear combination of input data and let z_{ij} be the output from the 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:

$$a_{ij} = \sum_{p=1}^{d+1} w_{jp}^{(1)} x_{ip} \tag{1}$$

$$z_{ij} = \sigma(a_{ij}) = \frac{1}{1 + \exp(-a_{ij})} \tag{2}$$

where $w_{ji}^{(1)} = W^{(1)}[j][p]$ is the weight of connection from the p^{th} input feature to unit j in hidden layer. Note that we do not compute the output for the bias hidden node $(m+1)$; z_{m+1} is directly set to 1.

The third layer in neural network is called the output layer where the learned features in hidden units are linearly combined and a sigmoid function is applied 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 l ($l = 1, 2, \cdots, 10$) 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 l ($l = 1, 2, \cdots, 10$), we can compute its value as follow:

$$b_{il} = \sum_{j=1}^{m+1} w_{lj}^{(2)} z_{ij} \tag{3}$$

$$o_{il} = \sigma(b_{il}) = \frac{1}{1 + \exp(-b_{il})} \tag{4}$$

Now we have finished the **Feedforward pass**.

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:

$$J(W^{(1)}, W^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1-y_{il})\ln(1-o_{il})) \tag{5}$$

where y_{il} indicates the l^{th} target value in 1-of-K coding scheme of input data i and o_{il} is the output at l^{th} output node for the i^{th} data example (See (4)).

Because of the form of error function in equation (5), we can separate its error function in terms of error for each input data $\mathbf{x}_i$:

$$J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{i=1}^{n} J_i(W^{(1)}, W^{(2)}) \tag{6}$$

where

$$J_i(W^{(1)}, W^{(2)}) = -\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1-y_{il})\ln(1-o_{il})) \tag{7}$$

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 l where $j = 1, 2, \cdots, m+1$ and $l = 1, \cdots, 10$:

$$\frac{\partial J_i}{\partial w_{lj}^{(2)}} = \frac{\partial J_i}{\partial o_{il}}\frac{\partial o_{il}}{\partial b_{il}}\frac{\partial b_{il}}{\partial w_{lj}^{(2)}} \tag{8}$$

$$= \delta_{il} z_{ij} \tag{9}$$

where

$$\delta_{il} = \frac{\partial J_i}{\partial o_{il}}\frac{\partial o_{il}}{\partial b_{il}} = -(\frac{y_{il}}{o_{il}} - \frac{1-y_{il}}{1-o_{il}})(1-o_{il})o_{il} = o_{il} - y_{il}$$

The error function (log loss) that we are using in (5) is different from the the squared loss error function that we have discussed in class. Note that the choice of the error function has "simplified" the expressions for the error!

On the other hand, the derivative of error function with respect to the weight from the p^{th} input feature to hidden unit j where $p = 1, 2, \cdots, d+1$ and $j = 1, \cdots, m$ can be computed as follow:

$$\frac{\partial J_i}{\partial w_{jp}^{(1)}} = \sum_{l=1}^{k}\frac{\partial J_i}{\partial o_{il}}\frac{\partial o_{il}}{\partial b_{il}}\frac{\partial b_{il}}{\partial z_{ij}}\frac{\partial z_{ij}}{\partial a_{ij}}\frac{\partial a_{ij}}{\partial w_{jp}^{(1)}} \tag{10}$$

$$= \sum_{l=1}^{k}\delta_l w_{lj}^{(2)}(1-z_{ij})z_{ij}x_{ip} \tag{11}$$

$$= (1-z_{ij})z_{ij}(\sum_{l=1}^{k}\delta_l w_{lj}^{(2)})x_{ip} \tag{12}$$

Note that we do not compute the gradient for the weights at the bias hidden node.

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:

$$\nabla J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{i=1}^{n}\nabla J_i(W^{(1)}, W^{(2)}) \tag{13}$$

We again can use the gradient descent to update each weight (denoted in general as w) with the following rule:

$$w^{new} = w^{old} - \gamma \nabla J(w^{old}) \tag{14}$$

3.2.4 Regularization in the neural network

In order to avoid *overfitting*, 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:

$$\widetilde{J}(W^{(1)}, W^{(2)}) = J(W^{(1)}, W^{(2)}) + \frac{\lambda}{2n}\left(\sum_{j=1}^{m}\sum_{p=1}^{d+1}(w_{jp}^{(1)})^2 + \sum_{l=1}^{k}\sum_{j=1}^{m+1}(w_{lj}^{(2)})^2\right) \tag{15}$$

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:

$$\frac{\partial \widetilde{J}}{\partial w_{lj}^{(2)}} = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\partial J_i}{\partial w_{lj}^{(2)}} + \lambda w_{lj}^{(2)}\right) \tag{16}$$

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

$$\frac{\partial \widetilde{J}}{\partial w_{jp}^{(1)}} = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\partial J_i}{\partial w_{jp}^{(1)}} + \lambda w_{jp}^{(1)}\right) \tag{17}$$

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 Python implementation of the neural network

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

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

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

Optimization: In general, the learning phase of the neural network consists of 2 tasks. First task is to compute the value and gradient of error function given the 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, you have to use the Python scipy function: **scipy.optimize.minimize** (using the option *method='CG'* for conjugate gradient descent), 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 *minimize* are provided here: `http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.minimize.html`.

We use regularization in the neural network to avoid the overfitting problem (more about this will be discussed in class). You are expected to change different value of λ to see its effect in prediction accuracy in test set. Your report should include diagrams to explain the relation between λ and performance of the neural network. Moreover, by plotting the value of λ with respect to the accuracy of the neural network, you should explain in your report how to choose an appropriate hyper-parameter λ to avoid both underfitting and overfitting problem. You can vary λ from 0 (no regularization) to 60 in increments of 5 or 10.

You are also expected to try different number hidden units to see its effect to the performance of the neural network. Since training the neural network is very slow, especially when the number hidden units in the 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. Recommended values: $4, 8, 12, 16, 20$.

4 TensorFlow Library

In this assignment you will only implement a single layer the neural network. Note that, implementing multiple layers can be a very cumbersome coding task. However, additional layers can provide a better modeling of the data set. The analysis of the challenging AI Quick Draw data set will show how adding more layers can improve the performance of the the neural network. To experiment with the neural networks with multiple layers, we will use Google's TensorFlow library (`https://www.tensorflow.org/`).

Your experiments should include the following:

- Evaluate the accuracy of deep the neural network (try 3, 5, and 7 hidden layers) on AI Quick Draw data set. Use *deepnnScript.py* to obtain these results.
- Compare the accuracy and training time of deep neural network (using TensorFlow) with different number of layers.

5 Submission

You are required to submit a single file called *proj2.zip* using UBLearns.
File *proj2.zip* must contain 2 folders: *report* and *code*.

- Folder *report* must contain your report file (in pdf format). Please indicate the team members and your course number on the top of the report.
- Folder *code* must contain the following updated files: *nnFunctions.py*, *deepnnScript.py* and *params.pickle*[2]. File *params.pickle* contains the learned parameters of the neural network for **handwritten digits dataset only**. Concretely, file *params.pickle* must contain the following variables: optimal *n_hidden* (number of units in hidden layer), *W1* (matrix of weights $W^{(1)}$ as mentioned in section 3.2.1), *W2* (matrix of weights $W^{(2)}$ as mentioned in section 3.2.1), optimal λ (regularization coefficient λ as mentioned in section 3.2.4).[3]

Using UBLearns Submission: Please continue using your programming groups. Contact the instructors if you need to change groups on UBLearns.

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

- Explanation of how to choose the hyper-parameters for the neural network (number of hidden units, regularization term λ).
- Compare the accuracy and training time of deep neural network (using TensorFlow) with different layers for the AI Quickdraw Dataset only.

[2]Check this to learn how to pickle objects in Python: `https://wiki.python.org/moin/UsingPickle`

[3]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.

6 Grading scheme

The TAs will deploy a testing script that will test the functionality of individual functions that you submit within the *nnFunctions.py* file. Full points will be awarded if the output of the function exactly matches the expected output. The second grading script will load the *params.pickle* file that you submit and then test a small testing data set. You get full points (10) if the accuracy using your model parameters is within ±5% of the accuracy reported by our code. Note that this data set **will not be** made available to you.

- Successfully implement the neural network: 50 points (*sigmoid()* [10 points], *nnObjFunction()* [30 points], *nnPredict()* [10 points]).
- Project report: 50 points
  - Explanation with supporting figures of how to choose the hyper-parameter for the neural network: 20 points
  - Accuracy of classification method on the handwritten digits test data: 10 points
  - Accuracy of classification method on the AI Quick Draw data set: 10 points
  - Compare the accuracy and training time of deep neural network (using TensorFlow) with different number of layers : 10 points

7 Computing Resources

You are allowed to implement the project on your personal computers using Python 3.4 or above. You will need `numpy` and `scipy` libraries. If you need to use departmental resources, you will need to use `metallica.cse.buffalo.edu`, which has Python 3.4.3 and the required libraries installed.

Students attempting to use the TensorFlow library have two options:

1. Install TensorFlow on personal machines. Detailed installation information is here - `https://www.tensorflow.org/`. Note that, since TensorFlow is a relatively new library, you might encounter installation issues depending on your OS and other library versions. We will not be providing any detailed support regarding TensorFlow installation. If issues persist, we recommend using the option 2.
2. Use `springsteen.cse.buffalo.edu`. If you are registered into the class, you should have an account on that server. The server already has Python 3.4.3 and TensorFlow 0.12.1 installed. Please use `/util/bin/python` for Python 3. Note that TensorFlow will not work on `metallica.cse.buffalo.edu`.

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).

[3] Liu, Ziwei; Luo, Ping; Wang, Xiaogang; Tang, Xiaoou. "Deep Learning Face Attributes in the Wild", Proceedings of International Conference on Computer Vision (ICCV) (2015).