CSE474/574 Introduction to Machine Learning
Programming Assignment 2

Neural-networks

Due Date: **April 9th 2021** by 11.59 EST
Maximum Score: 100

Note The assignment consists of this pdf document and four additional files: a file containing the skeleton Python function implementations (***nnFunctions.py***), a unit test file (***nnUnitTests.py***), and two driver Jupyter notebooks (***PA2-Part2.ipynb*** and ***PA2-Part3.ipynb***)). Note that for Part I, you will need to write code within the specified functions in ***nnFunctions.py***. For Parts II and III, you will run experiments using the provided notebook files. You will also need to use the data set available here - `https://www.cse.buffalo.edu/ubds/docs/AI_quick_draw_sm.pickle`.

Evaluation For this assignment, we will run the unit tests to test the correctness of the functions for Part I. For Parts II and III, we will evaluate your report. Please note that if your methods for Part I are not implemented correctly, you will automatically lose all points for Part II.

Submission You are required to submit a single file called ***pa2.zip*** using UBLearns. One submission per group is required. File ***pa2.zip*** must contain 2 files: ***report.pdf*** and ***nnFunctions.py***. *Note*: You do not have to submit the notebook files. They are for you to run your experiments and generate plots. The findings and the plots should be included in the report.

- Submit your report in a pdf format. Please indicate the **team members**, **group number**, and your **course number** on the top of the report.
- The notebooks should contain all implemented functions. Please do not change the names of the files.

Please make sure that your group is enrolled in the UBLearns system: You should submit one solution per group through the groups page. *If you want to change the group, contact the instructors.*

Overview

This assignment has three parts. In Part I, you will implement a simple neural network in Python. In Part II, you will deploy the implemented neural network for labeling images and in Part III, you will compare its performance against a more complex neural network architecture that uses a *deeper* neural network.

After completing this assignment, you should be able to understand:

- the inner workings of a neural network (both training and prediction).
- understanding the role of *regularization* in controlling the *bias-variance* tradeoff in neural network training.
- using neural networks to solve computer vision problems.

- understanding the power of neural networks over simpler models.
- comparing a simple multi-layered perceptron with more complex (deep) neural network architectures for image analysis.
  - This part will involve using the `Keras/TensorFlow` library to deploy deep neural networks.

Part I

Implementing a Neural Network 40 Points

In this part, you will implement a simple multi-layer perceptron, with a single hidden layer. You have to implement the necessary algorithms for training the network and for obtaining predictions from the trained network. You will be given a toy data set to verify the correctness of your implemented methods.

The next section describes the mathematics behind the neural network and the various functions that need to be implemented.

1 Neural Network Description

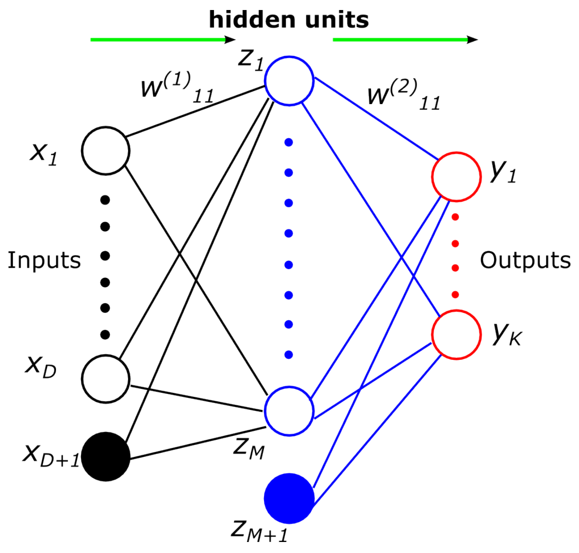

Figure 1: Neural network

A multi-layered perceptron (MLP) 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 layer. In this document, we denote M as the number of 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 layer units will represent the dimension of learned features in neural network, it is our choice to choose an appropriate value for M. Too many hidden layer units may lead to the slow training phase while too few hidden layer 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 input to belong to class l[1]. In this document, we denote K as the number of units in output layer.

The parameters in Neural Network model are the weights associated with the hidden layer units and the output layers units. In our standard Neural Network with 3 layers (input, hidden, output), in order to represent the model parameters, we use 2 matrices:

- $\mathbf{W}^{(1)} \in \mathbb{R}^{M \times (D+1)}$ 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.

- $\mathbf{W}^{(2)} \in \mathbb{R}^{K \times (M+1)}$ 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 Neural Network.
In the next section, we will explain how to perform learning in Neural Network.

1.1 Feedforward Propagation

In Feedforward Propagation, given parameters of 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 layer units. Let a_j for $1 \leq j \leq M$ be the linear combination of input data and let z_j be the output from the hidden layer unit j after applying an activation function (in this exercise, we use sigmoid as an activation function). For each layer hidden layer unit j $(j = 1, 2, \cdots, M)$, we can compute its value as follow:

$$a_j = \sum_{p=1}^{D+1} w_{jp}^{(1)} x_p \tag{1}$$

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

where $w_{ji}^{(1)} = \mathbf{W}^{(1)}[j][p]$ 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 layer units are linearly combined and a sigmoid function is applied to produce the output. Concretely, for each output layer unit l $(l = 1, 2, \cdots, K)$, we can compute its value as follow:

$$b_l = \sum_{j=1}^{M+1} w_{lj}^{(2)} z_j \tag{3}$$

$$o_l = \sigma(b_l) = \frac{1}{1 + \exp(-b_l)} \tag{4}$$

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

1.2 Error function and Backpropagation

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

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

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)).

[1]In this assignment we will use the neural network for classification.

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

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

where

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

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 layer unit j to output layer unit l where $j = 1, 2, \cdots, M+1$ and $l = 1, \cdots, K$:

$$\frac{\partial J_i}{\partial w_{lj}^{(2)}} = \frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial w_{lj}^{(2)}} \tag{8}$$

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

where

$$\delta_l = \frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l} = -(\frac{y_l}{o_l} - \frac{1-y_l}{1-o_l})(1-o_l)o_l = o_l - y_l$$

Note that we are dropping the subscript i for simplicity. 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 layer unit j where $p = 1, 2, \cdots, D+1$ and $j = 1, \cdots, M$ can be computed as follow:

$$\frac{\partial J_i}{\partial w_{jp}^{(1)}} = \sum_{l=1}^{K}\frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jp}^{(1)}} \tag{10}$$

$$= \sum_{l=1}^{K}\delta_l w_{lj}^{(2)}(1-z_j)z_j x_p \tag{11}$$

$$= (1-z_j)z_j(\sum_{l=1}^{k}\delta_l w_{lj}^{(2)})x_p \tag{12}$$

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:

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

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

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

1.3 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:

$$\widetilde{J}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) = J(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}) + \frac{\lambda}{2N}\left(\sum_{j=1}^{M}\sum_{p=1}^{D+1}(w_{jp}^{(1)})^2 + \sum_{l=1}^{K}\sum_{j=1}^{M+1}(w_{lj}^{(2)})^2\right) \tag{15}$$

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 follows:

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

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.

1.4 Python 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 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 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 Python 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, 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`.

1.5 Grading

We have provided unit tests to check the correctness of the implemented functions. The grading scheme for this part is as follows:

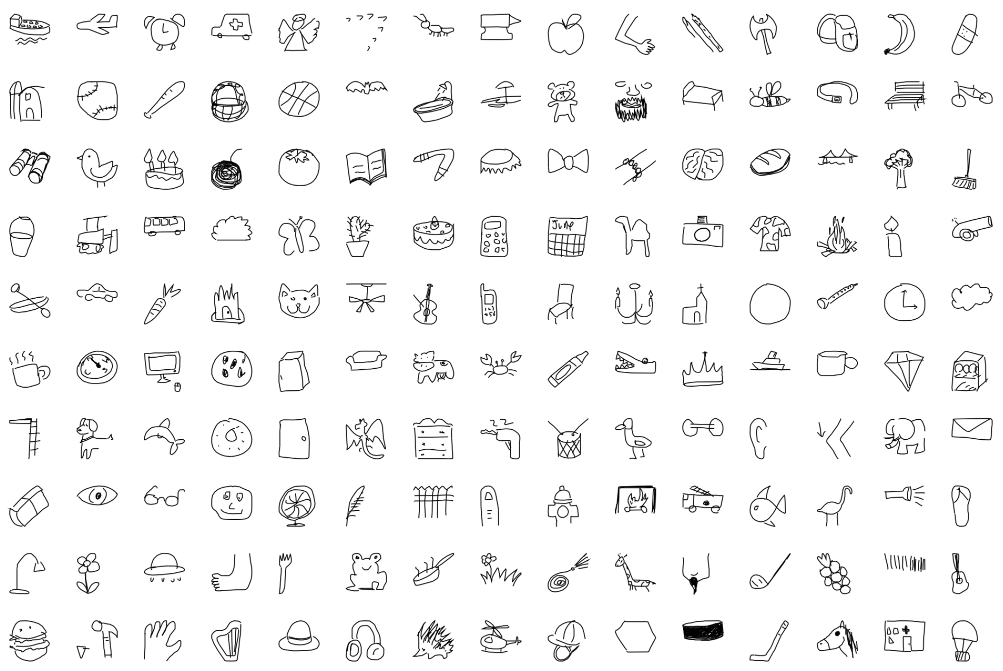

Figure 2: Sample sketches from the Quick Draw data set (Src: https://quickdraw.withgoogle.com/data)

- *sigmoid* - Pass unit tests 1–3 (**2 Points** each).
- *nnObjFunction* - Pass unit tests 4–7 (**6 Points** each).
- *nnPredict* - Pass unit test 8 (**10 Points**).

Grading: The unit tests provided in the ***nnUnitTests.py*** file are given for you to verify the functions. We will use a different set of unit tests (similar but with different data) to test your code. You may execute the tests from the command line as:

```
python nnUnitTests.py
```

Part II

Using the Neural Network for Image Classification 40 Points

In this part, you will use the neural network implemented in the previous part to classify hand-drawn doodles into the correct category. The AI quick draw data set used for this part is created from the original Quick Draw dataset[2]. The original Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw! (See Figure 2).

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. However, for this assignment, we have provided you with a subsampled Quick Draw data set with the following properties:

[2]https://quickdraw.withgoogle.com/data

- 10 categories including ‘apple’, ‘basketball’, ‘arm’, ‘horse’, ‘ant’, ‘axe’, ‘alarm clock’, ‘airplane’, ‘banana’ and ‘bed’.
- Each category includes 5000 images with the size of 28×28. Therefore, in the data set, each sample is a 784 length vector.
- 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.
- *Note*: Since the data file is large, you will need to download it from the following link : `https://www.cse.buffalo.edu/ubds/docs/AI_quick_draw_sm.pickle`. This file is nearly 40 MB in size.

Tasks

You will first study the performance of the multi-layered perceptron on the provided data as you control the complexity of the network by changing the number of nodes in the hidden layer, M, and the regularization parameter, λ.

REPORT 1.

1. Run the evaluation of the implemented neural network in the notebook - `PA2-Part2.ipynb` and report the training and test accuracy and the run time. [**10 Points**]
2. Compare the performance when the number of hidden layer units (M) is increased from 10 to 100, in increments of 10. Plot the training and test accuracies and training time, as a function of M. Make your observations and state the optimal value of M that you would finally choose, along with the reason.[**10 Points**]
3. For the optimal setting of M found above, rerun your analysis by modifying λ from 0 to 20, in steps of 2. Again, plot the training and test accuracies and the training time as a function of λ and make your observations. Which value of λ is optimal and why?[**10 Points**]
4. For the optimal settings for M and λ, study the performance of your model on the test data. What kind of objects does it make more mistakes on? Briefly discuss how the performance of your model can be improved futher.[**10 Points**]

Part III

Using Deep[er] Neural Networks for Image Classification **20 Points**

In this part, you will again use the same data set as Part II, but use the `Keras` library to create your model. You will need to use the provided ***PA2-Part3.ipynb*** notebook to run your experiments. The objective here is to determine how much can we improve the model performance by increasing the number of hidden layers in the model, and by changing the activation function.

Tasks

You will first study the performance of the neural network on the provided data as you control the complexity of the network by changing the number of hidden layers, L, and by using different activation functions.

REPORT 2.

1. Fixing the number of units in each hidden layer (M) to the optimal value found in Part II, run the evaluation of the implemented neural network in the notebook - `PA2-Part3.ipynb` for different number of hidden layers (L), from 1 to 5. Plot the training and test accuracies and training time, as a function of L. Make your observations and state the optimal value of L that you would finally choose, along with the reason.[**10 Points**]

2. Using the optimal M and L from the previous part, compare the performance of the model (in terms of training and testing accuracies and the training time) for different choices of the activation function (try *sigmoid*, *tanh*, and *relu*). Report the best choice.[**10 Points**]

Computing Resources

For this part, you will need `Keras` library (preferably with `TensorFlow` backend. 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. You have two options:

1. Install Keras/TensorFlow on personal machines. Detailed installation information is here - `https://anaconda.org/conda-forge/keras`.

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`.