INF552: Programming Assignment 5 [Neural Networks]

Priyambada Jain (priyambj@usc.edu)
Sai Sree Kamineni (skaminen@usc.edu)
Varad Kulkarni (vdkulkar@usc.edu)

Part 1 Implementation

Python Version:
Command to run – python nn.py
Note : All the input images and list for training and test images should be placed in

Input Files - *Gestures, downgesture_train.list, downgesture_test.list* (In the same directory as the python file)

Data Structures Used - Numpy array

Result- Print the predicted label along with the actual label on the terminal.
Also calculates the accuracy for the model using the actual label and prints on the terminal.

Output – Collects the predicted labels along with the actual lables in a file. Also prints them on the terminal. Prints the accuracy as well. Accuracy obtained : **89.156626506.**

BestWeights – Has the files with hidden and outer layer weights that give highest accuracy.

Code Level Optimizations- For faster execution of epochs used Numpy array for calculation during forward propagation as well as for backward propagation.

Algorithm:

We have implemented Back Propagation Algorithm for Feed Forward Neural Network

1. Training the model : **train_function(train_data, train_label)**

    - Read the training images from "downgesture_train.list" in the gestures folder using "**read_pgm"** function defined.

*Note – **read_pgm** function is from https://stackoverflow.com/questions/7368739/numpy-and-16-bit-pgm?answertab=votes - tab-top*

- Also assign labels to the images based on if the name of the image containing down or not.

- **Initialize the weights for input layer and hidden layer**
  Epochs = 1000
  One input layer, one hidden layer of size 100 and one output layer
  For each image of training set perform the following operations:
  a. Forward Propagation:
     X1= train_data[image]
     S1 = np.dot (X1, input layer weights)
     X2 = activation (S1)
     S2 = np.dot (X2, hidden layer weights)
     X3 = activation(S2)
     Erorr = (train_label[image] – X3)**2

     Where Xi refer to the input for the i^{th} layer and Si refer to the delta for the i^{th} layer.

  b. Back Propagation:
     delta(i) = 2 * (Xi – y) * Xi*(1-Xi)
     where Xi refers to the i^{th} layer input and "y" refer to the actual label for the training set image.
     Now weights are updated based on the gradient descent obtained from the following formula:
     gradient(i) = X(i-1)*delta(i)*learning_rate
     Wi = Wi – gradient(i)

     Propagate to the input layer and update the weights.

- Load the values obtained for weights for each layer in a text file.

2. Testing the model: test_function(test_data, test_label)

- Read the training images from "downgesture_test.list" in the gestures folder using "**read_pgm"** function defined.

- Also assign labels to the images based on if the name of the image containing down or not.

- Load the weights for the model for each layer learned during the training phase.

- For each image in the test image make the prediction using the model's weights.
- Store the prediction in a file called "prediction.txt".
- Calculate the accuracy of the model using the actual labels against the predictions obtained.

Note: Here the activation function is the sigmoid function. Also, the accuracy varies as per the number of iterations and also depends on the random weights initialized.

Challenges Faced:

Reading the pgm image in the correct format was a challenge.
Accuracy kept changing for every run. Figuring out the best accuracy was challenging.

Part 2: Software Familiarization

Python Version 3.x is used for this.
Implementation can be referred from nn_part2.py. Command to run => python nn_part2.py

Library Used: **sklearn.neural_network : MLPClassifier**

Brief: It is a Multi-Layer Perceptron classifier. This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

Parameters:

1) **Solver => sgd**: **'sgd' refers to stochastic gradient descent.**
2) **alpha** => 0 L2 penalty (regularization term) parameter.
3) **hiddenlayer** => The ith element represents the number of neurons in the ith hidden layer.
4) **activation** =>
5) **learning_rate_init** => 0.1 The initial learning rate used. It controls the step-size in updating the weights.
6) **max_iter**=> 1000 Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

Comparing sklearn with our Implementation:

- sklearn MLPClassifier accuracy: **89.1566265060241**
- Our implementation accuracy: **89.156626506**

Part 3: Applications

1) **Character Recognition** - The idea of character recognition has become very important as handheld devices like the Palm Pilot are becoming increasingly popular. Neural networks can be used to recognize handwritten characters.

2) **Image Compression** - Neural networks can receive and process vast amounts of information at once, making them useful in image compression. With the Internet explosion and more sites using more images on their sites, using neural networks for image compression is worth a look.

3) **Stock Market Prediction** - The day-to-day business of the stock market is extremely complicated. Many factors weigh in whether a given stock will go up or down on any given day. Since neural networks can examine a lot of information quickly and sort it all out, they can be used to predict stock prices.

4) **Traveling Saleman's Problem** - Interestingly enough, neural networks can solve the traveling salesman problem, but only to a certain degree of approximation.

5) **Medicine, Electronic Nose, Security, and Loan Applications** - These are some applications that are in their proof-of-concept stage, with the acception of a neural network that will decide whether or not to grant a loan, something that has already been used more successfully than many humans.

6) **Miscellaneous Applications** - These are some very interesting (albeit at times a little absurd) applications of neural networks.

7) **Classification** — A neural network can be trained to classify given pattern or data set into predefined class. It uses feedforward networks.

8) **Prediction** — A neural network can be trained to produce outputs that are expected from given input.

9) **Clustering** — The Neural network can be used to identify a special feature of the data and classify them into different categories without any prior knowledge of the data.

Part 4: References

1. http://neuralnetworksanddeeplearning.com/chap2.html
2. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html
3. http://neuralnetworksanddeeplearning.com/chap1.html
4. https://www.technologyreview.com/s/513696/deep-learning/
5. https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they
6. http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html