CSL603– Machine Learning
Lab 3

Due on 30/10/2018 11.55pm

Instructions: Upload to your moodle account one zip file containing the following. Please do not submit hardcopy of your solutions. In case moodle is not accessible email the zip file to the instructor at ckn@iitrpr.ac.in. Late submission is not allowed without prior approval of the instructor. You are expected to follow the honor code of the course while doing this homework.

1. You have to work individually for the lab.
2. This lab must be implemented in Matlab/Python.
3. A neatly formatted PDF document with your answers for each of the questions in the homework. You can use latex, MS word or any other software to create the PDF.
4. Include a separate folder named as 'code' containing the scripts for the homework along with the necessary data files. Ensure the code is documented properly.
5. Include a README file explaining how to execute the scripts.
6. Name the ZIP file using the following convention rollnumber(s)hwnumber.zip

---

In this lab, you will be experimenting with multi-layer perceptron. The second part of the lab takes time to train. So, it is advisable to get started early. Please contact the IT department to obtain a licensed version of Matlab that can be installed on your machine for implementing this lab. You are welcome to implement this lab in Python as well.

In this question, you will implement a basic neural network to predict the steering angle the road image for a self-driving car application that is inspired by Udacity's Behavior Cloning Project [1]. This dataset has been compiled from Udacity's simulator. However, we have preprocessed the original images to transform them to greyscale images of size 32x32. This is roughly the same size of an MNIST image. You should be able to run these experiments on your machines. The accompanying dataset folder (steering) contains the training images. The data.txt files contains the image name and the corresponding steering angle. The following are the specifications for the first task of this lab.

a. The network architecture should be 1024 (input)-512-64-1
b. Use a numerically stable implementation of sigmoid, such as scipy.special.expit that uses something like the following

```
def sigmoid(x):
if x >= 0:
    z = exp(-x)
    return 1 / (1 + z)
else:
    z = exp(x)
    return z / (1 + z)
```

c. All the hidden layers will employ sigmoid activation function, while the output layer will not have any activation function.
d. The implementation should have the ability to tune/set the learning rate, minibatch size, and the number of training iterations.
e. Sum of squares error should be used as the loss function at the output layer.
f. The minibatches must proceed sequentially through the data. The data should be shuffled initially before the start of the training.
g. Dropout will be implemented for all layers. The implementation should have the ability to set the dropout percentage.
h. The weights of each layer should be initialized by uniformly spacing over the range [-0.01, 0.01]. The bias terms should be initialized to 0.
i. The dataset provided to you should be split into training/validation according to 80:20 ratio.

For grading purposes, we want you to generate the following plots. Note that we should be able to recreate these plots using your implementation.

i. A plot of sum of squares error on the training and validation set as a function of training iterations (for 5000 epochs) with a learning rate of 0.01. (no dropout, minibatch size of 64).
ii. A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a fixed learning rate of 0.01 for three minibatch sizes – 32, 64, 128.
iii. A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with a learning rate of 0.001, and dropout probability of 0.5 for the first, second and third layers.
iv. A plot of sum of squares error on the training and validation set as a function of training iterations (for 1000 epochs) with different learning rates – 0.05, 0.001, 0.005 (no drop out, minibatch size – 64)

Discuss your inferences/observations from these plots on the training/tuning neural networks.

Bonus question (extra 10 points)

You are allowed to perform the following modifications to the networks to try better models with the constraint that these have to be implemented from scratch. (some of them are easy to implement, while some might take a significant amount of time)

- Implement convolution layers

- Implement a different activation function
- Implement Max Pooling layers
- Implement and try a different optimizer (such as Adam or RMS Prop)
- Experiment with varying number of layers and nodes in a layer

An important aspect of machine learning is reproducibility of the results presented in a paper/report. Therefore, we will run your code to see if the results are closely matching with what you have presented in the report. Any deviation beyond a reasonable threshold will be considered as fudging of results and will invite severe penalty.

Reference

[1] https://github.com/udacity/CarND-Behavioral-Cloning-P3