COL 341: Assignment 2

Notes:

- This assignment has two parts - Neural Network and Convolutional Neural Networks. that you might plot.
- You are advised to use vector operations (wherever possible) for best performance.
- Include a report of maximum 5 pages which should be a brief description explaining what you did. Include any observations and/or plots required by the question in the report.
- You should use Python for all your programming solutions.
- Your assignments will be auto-graded, make sure you test your programs before submitting. We will use your code to train the model on training data and predict on test set.
- Input/output format, submission format and other details are included. Your programs should be modular enough to accept specified parameters.
- You should submit work of your own. You should cite the source, if you choose to use any external resource. You will be awarded F grade or DISCO in case of plagiarism.
- You can use total of 7 buffer days across all assignments.
- Data is available at this link

1. **Neural Networks (50 points, Due date: 9:00 PM Wednesday, 4th September, 2019)**
   In this problem, we'll train neural networks to classify a binary class(**Toy**) and multi-class(**CIFAR10**) dataset.

   (a) (12.5 points) Write a program to implement a general neural network architecture. Implement the back-propagation algorithm from first principles to train the network. You should train the network using Mini-Batch Gradient Descent. Your implementation should be generic enough so that it can work with different architectures. Assume a fully connected architecture i.e., each unit in a hidden layer is connected to every unit in the next layer. Have an option for an adaptive learning rate $\eta_t = \frac{\eta_0}{\sqrt{t}}$. Use Binary Cross Entropy Loss(BCE) as the loss function. Use **sigmoid** as the activation function for the units in **intermediate** and **output layers**.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\{y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\} \tag{1}$$

   Here i indexes samples. Here y_i is a single value which is 1 if label is 1 or 0 otherwise. n is the number of samples in the batch. $\hat{y}_i \in$(0,1) is predicted probability of a sample belonging to class 1.
   Use Your Implementation to train neural network on Toy training dataset with predefined parameters(corresponding to fixed learning rate) and predict on the publicly available Toy testing dataset.

   - Fixed Learning Rate

$$w(t) = w(t-1) - \eta_0 \nabla_w L(w; X_{b(t-1):bt}, y_{b(t-1):bt}) \tag{2}$$

   Here t indicates epoch number while denotes model weights. b indicates batch size while X, y denotes data and labels respectively. η_0, L represents fixed learning rate and loss function respectively. $\nabla_w L$ signifies gradient of the loss function with respect to model weights.

(b) (12.5 points) Modify neural network architecture made in part a) to cater for multi-class dataset(CIFAR10). You have been provided 2 CSV data files corresponding to train and test sets respectively. Each row in the data file corresponds to an image of size 32x32. Last entry of each row corresponds to the label associated with the image(-1 for test set) preceded by the vector of gray-scale pixel intensities of the image. The data belongs to 10 different classes corresponding to 10 categories of images. Use Cross Entropy Loss(CE) as the loss function. Use **sigmoid** as the activation function for the units in **intermediate layers**. Use **softmax** as the activation function for the units in **output layer**.

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij}) \tag{3}$$

$$\hat{y}_{ij} = \frac{\exp z_{ij}}{\sum_{j'=1}^{k}\exp z_{ij'}} \tag{4}$$

Here i indexes samples while j indexes class labels. Here y_i is a one-hot vector where only one value (corresponding to true class index) is non-zero for sample i. n is the number of samples in the batch. k is the number of labels. $\hat{y}_{ij} \in (0,1) : \sum_j \hat{y}_{ij} = 1 \,\forall\, i,j$ is the prediction of a sample. z_{ij} is the value being input into j^{th} perceptron of softmax layer for i^{th} sample.

Use Your Implementation to train a neural network on the given CIFAR10 training dataset with predefined parameters(corresponding to adaptive learning rate) and predict on the publicly available CIFAR10 testing dataset.

- Adaptive Learning Rate

$$w(t) = w(t-1) - \frac{\eta_0}{\sqrt{t}}\nabla_w L(w; X_{b(t-1):bt}, y_{b(t-1):bt}) \tag{5}$$

Here t indicates epoch number while w denotes model weights. b indicates batch size while X, y denotes data and labels respectively. η_0, L represents seed value and loss function respectively. $\nabla_w L$ signifies gradient of the loss function with respect to model weights.

(c) (12.5 points) Use Your Implementation to train a neural network on the given CIFAR10 dataset with predefined parameters. Experiment with different types of architectures. Vary the number of hidden layers. Try different number of units in each layer, different loss and activation functions etc. What happens as we increase the number of units or the number of hidden layers? Comment on your observation and submit your best performing architecture.

Note: Use holdout method to find the best architecture. Split the data set into two: a set of examples to train with, and a validation set. Train the model on the training set. Use the prediction on the validation set to determine which architecture to use.

(d) (12.5 points) In the previous part pixel values are being used as features of the image. Can some features other than the absolute pixel values be used? Experiment with various feature extraction techniques such as Gaber Filters, DCT, FFT, HOG, wavelet etc. to extract feature from the images. You can use pre-defined python libraries to create these features. How does these changes affect your accuracy compared to the previous parts. Comment on your observations and report details of your best performing design.

Evaluation:

- For part-a and part-b, you can get 0 (error), partial marks (code runs fine but weights are incorrect within some predefined threshold) and full (works as expected).
- For part-c and part-d, marks will be given based on accuracy on test data-set. There will be relative marking for this part.
- For part-c and part-d marking will be done in two parts: code (75%) and report(25%).

Submission Instructions:

Neural Network

Submit your code in 4 executable python files called neural_a.py, neural_b.py, neural_c.py, neural_d.py

The file name should corresponds with part [a,b,c,d] of the assignment.
The parameters are dependant on the mode:

python neural_a.py trainfile.csv param.txt weightfile.txt
Here you have to write a line aligned weightfile.txt by writing the weight values for the trained model. Here, param.txt will contain five lines of input, the first being a number [1-2] indicating which learning rate strategy to use and the second being the fixed learning rate (for "1"), seed value for adaptive learning rate (for "2"). The third line will be the max number of iterations. Also print the number of iterations your program takes. The fourth line will contain batch size for one iteration of mini batch gradient descent. Fifth line will contain the architecture of the neural network with a sequence of numbers denoting number of perceptrons in consequent layers eg. 10 10 5 denotes 3 hidden layers with 10, 10 and 5 perceptrons respectively.

python neural_b.py trainfile.csv param.txt weightfile.txt
Same as for part a.

python neural_c.py trainfile.csv testfile.csv outputfile.txt
Here you have to write the predictions (1 per line) and create a line aligned outputfile.txt. However there would be no input param.txt as parameters would correspond to your best architecture and will need to be comprehensively detailed in the report.

python neural_d.py trainfile.csv testfile.csv outputfile.txt
Same as for part c.

Part (e)
Here you have to submit a pdf file with all details of best architectures and features for parts c) and d). Report results and observations of all variations experimented with irrespective of whether they lead to increase in accuracy. There will be no demos or presentations for this part.

Coding Guidelines:

(a) For parts a) and b)

- Don't shuffle/change order of the data files.
- Don't scale/normalize data in any manner possible.
- Don't apply early stopping criterion. Run for full specified number of iterations.
- To ensure uniformity of weight initialization, initialize all weights as zero.
- Be careful to code Mini-batch Gradient Descent to be closely consistent with the theoretical framework taught in class.
- Parameters for which public evaluation is going to be done is given alongside expected model weights after 1 iteration of training(for help in debugging) in the data files. Final evaluation will be done on different training set with different parameters.

(b) For parts c) and d)

- Design code with emphasis on Vector Operations and minimal use of loops to ensure run-time efficiency. Note that Moodle allows maximum of 16 minutes for testing individual submissions and your code must finish executing within **10 minutes**.
- Have a suitable threshold for the following: 1) Batch Size * Number of Iterations 2) The product of number of perceptrons in each hidden layer in order to ensure above.
- Using Feature engineering in part c) will be seen as an attempt to subvert assignment by gaining unfair advantage and will most likely invite disciplinary action.
- Their will be another required file coined MoodleID where you simply have to fill in your moodle ID eg. me2110786. This is necessary for construction of a leaderboard wherein ranks based on best scores till a point can be seen.

Extra Readings (highly recommended for part (a,b)):

(a) Backpropogation for different loss functions