IITM-CS7015(JAN-MAY 2019) : Deep Learning — Given on: Feb 9th 2019

Programming Assignment #1 — Due on : Feb 25, 11:55pm IST

- The goal of this assignment is to implement and use gradient descent (and its variants) with backpropagation for a classification task.
- We strongly recommend that you work on this assignment in teams of 2. Both the members of the team are expected to work together (and not divide the work) since this assignment is designed with a learning outcome in view.
- Collaborations and discussions with others are strictly prohibited.
- You may use Python (numpy and Pandas) for your implementation. If you are using any other languages, please contact the TAs before you proceed.
- All the models will be tested on same environment on Kaggle. So, you must submit all your results on test data on Kaggle.
- You have to turn in the well documented code along with a detailed report of the results of the experiment electronically in Moodle. Typeset your report in Latex. Reports which are not written using Latex will not be accepted.
- The report should be precise and concise. Unnecessary verbosity will be penalized.
- You have to check the Moodle discussion forum regularly for updates regarding the assignment.

1 Task

In this assignment you need to implement a feedforward neural network (we strongly recommend using numpy for all matrix/vector operations). This network will be trained and tested using the **A-Z Handwritten Alphabets** dataset. Specifically, given an input image (28 x 28 = 784 pixels) from the dataset, the network will be trained to classify the image into 1 of 10[1] classes.

1.1 Format

Your implementation should support the use of the following hyper-parameters/options:

- –lr (initial learning rate for gradient descent based algorithms)
- –momentum (momentum to be used by momentum based algorithms)

[1] we have randomly sampled 10 classes from the original 26, also added noise to the data

- –num_hidden (number of hidden layers - this does not include the 784 dimensional input layer and the 10 dimensional output layer)
- –sizes (a comma separated list for the size of each hidden layer)
- –activation (the choice of activation function - valid values are tanh/sigmoid)
- –loss (possible choices are squared error[sq] or cross entropy loss[ce])
- –opt (the optimization algorithm to be used: gd, momentum, nag, adam - you will be implementing the mini-batch version of these algorithms)
- –batch_size (the batch size to be used - valid values are 1 and multiples of 5)
- –epochs (number of passes over the data)
- –anneal (if true the algorithm should halve the learning rate if at any epoch the validation loss decreases and then restart that epoch)
- –save_dir (the directory in which the pickled model should be saved - by model we mean all the weights and biases of the network)
- –expt_dir (the directory in which the log files will be saved - see below for a detailed description of which log files should be generated)
- –train (path to the Training dataset)
- –val (path to the validation dataset)
- –test (path to the Test dataset)

You should use the *argparse* module in python for parsing these parameters.

You need to submit the source code for the assignment. Your code should include one file called *train.py* which should be runnable using the following command:

```
python train.py --lr 0.01 --momentum 0.5 --num_hidden 3 --sizes 100,100,100
--activation sigmoid --loss sq --opt adam --batch_size 20 --epochs 5 --anneal true
--save_dir pa1/ --expt_dir pa1/exp1/ --train train.csv --val validation.csv
--test test.csv
```

Of course, the actual values for the options can change (for example, we could try different learning rates, momentum, optimization algorithms, etc.). For the remainder of this document we will assume that the above command is saved as a shell script in run.sh (from now on if we refer to run.sh then it means we are referring to the above command)

1.2 Log file

Your code should create the following log files: log_train.txt and log_val.txt in the expt_dir.

The log_*.txt file should contain the loss (squared error or cross entropy as specified) and the error rate (i.e., percentage of examples which were classified incorrectly) on the train/validation data after every 100 steps (refer to the Lecture slides for the definition of a step). The log.txt file should contain only the following lines:

```
Epoch 0, Step 100, Loss: <value>, Error: <value>, lr: 0.01
Epoch 0, Step 200, Loss: <value>, Error: <value>, lr: 0.01
Epoch 0, Step 300, Loss: <value>, Error: <value>, lr: 0.01
Epoch 0, Step 400, Loss: <value>, Error: <value>, lr: 0.01
...
Epoch 1, Step 100, Loss: <value>, Error: <value>, lr: 0.01
...
Epoch 2, Step 100, Loss: <value>, Error: <value>, lr: 0.01
...
...
Epoch <max_epoch>, Step 100, Loss: <value>, Error: <value>, lr: 0.01
...
```

(where lr is the learning rate which may change if you use annealing)
(Error can take on a real value between 0 to 100 but round it off to two decimal places)

1.3 Predictions

In addition, your code should also generate a test_submission.csv file in the expt_dir. Each line of this file should contain a image ID from the test set and the predicted label(0-9) of the corresponding test image. The file must also have column headers namely: *id* and *label*. As the test data has 10K examples your submission file must have 10K lines. Here is what a sample test_submission.csv file would look like:

```
id,label
0,3
1,2
3,8
...
9999,4
```

A sample submission file has been provided along with the dataset.

Note: You will have to initialize the weights of the network randomly. To ensure replicability of your results, make sure that you set a seed of 1234 [numpy.random.seed(1234)] before initializing the weights and biases. If you don‘t do this we may get very different results when we run your code after submission.

1.4 Kaggle Submission

It is suggested that both the team members make separate account on Kaggle. You can form teams for this assignment on the Kaggle assignment page. With this both the team members will be able to make submissions.

You must submit the test_submission.csv files on Kaggle assignment page for evaluation. The evaluation on Kaggle will be done in 2 steps. Till Feb 25, you are allowed to make 5 submissions on the portal everyday. These will be evaluated on 30% of the test data. Till Feb 25, your 1st evaluation scores will be visible on Public leaderboard. On Feb 25, you can select 2 of your best submissions to get evaluated in the second step. By default, your best 2 submissions will be taken for evaluation in the second step. After the deadline, your scores in the second step of evaluation will be visible on the Private leaderboard.

1.5 Report

You need to submit a report along with the assignment. The report should contain plots comparing the train and validation loss across different epochs for the following.

1. varying the size of the hidden layer (50, 100, 200, 300) - [keeping just one hidden layer] You will have to draw the following plots

2. varying the size of the hidden layer (50, 100, 200, 300) - [with two hidden layers and the same size for each hidden layer]

3. varying the size of the hidden layer (50, 100, 200, 300) - [with three hidden layers and the same size for each hidden layer]

4. varying the size of the hidden layer (50, 100, 200, 300) - [with four hidden layers and the same size for each hidden layer]

   For all the above 4 cases you will use sigmoid activation, cross entropy loss, Adam, batch size 20 and tune the learning rate to get best results. For each of the 4 questions above you need to draw the following plots:

   - x-axis: number of epochs, y-axis: training loss [4 curves - each curve corresponding to one of the four hidden sizes mentioned above]
   - x-axis: number of epochs, y-axis: validation loss [4 curves - each curve corresponding to one of the four hidden sizes mentioned above]

5. Adam, NAG, Momentum, GD [with 4 hidden layers and each layer having 300 neurons] (again sigmoid activation, cross entropy loss, batch size 20 and tune the learning rate. momentum wherever applicable to get best results). You need to make the following plots:

   - x-axis: number of epochs, y-axis: training loss [4 curves - each curve corresponding to one of the four optimization methods mentioned above]

- x-axis: number of epochs, y-axis: validation loss [4 curves - each curve corresponding to one of the four optimization methods mentioned above]

6. sigmoid v/s tanh activation [Adam, 2 hidden layers, 100 neurons in each layer, batch size 20, cross entropy loss]. You need to make the following plots:
   - x-axis: number of epochs, y-axis: training loss [2 curves - each curve corresponding to one of the two activation functions mentioned above]
   - x-axis: number of epochs, y-axis: validation loss [2 curves - each curve corresponding to one of the two activation functions mentioned above]
7. cross entropy loss v/s squared error loss [Adam, 2 hidden layers, 100 neurons in each layer, batch size 20, sigmoid activation] You need to make the following plots:
   - x-axis: number of epochs, y-axis: training loss [2 curves - each curve corresponding to one of the two loss functions mentioned above]
   - x-axis: number of epochs, y-axis: validation loss [2 curves - each curve corresponding to one of the two loss functions mentioned above]
8. Batch size: 1,20,100,1000 [Adam, 2 hidden layers, 100 neurons in each layer, sigmoid activation, cross entropy loss] You need to make the following plots:
   - x-axis: number of epochs, y-axis: training loss [4 curves - each curve corresponding to one of the 4 batch sizes mentioned above]
   - x-axis: number of epochs, y-axis: validation loss [4 curves - each curve corresponding to one of the 4 batch sizes mentioned above]

1.6 Evaluation

We anticipate that some of you may not be able to support all values of hyperparameters mentioned above. To help us evaluate only those options which are supported by your code you need to submit a file (supported.txt) listing the supported options for the following hyperparameters: anneal, opt, loss, activation. For example, if you have supported all possibles option for these hyperparameters then supported.txt will contain the following contents:

```
--anneal: true,false
--opt: gd,momentum,nag,adam
--loss: sq,ce
--activation: tanh,sigmoid
```

However, if your code does not support tanh activation and adam and squared error loss then supported.txt will contain the following contents:

```
--anneal: true,false
--opt: gd,momentum,nag
--loss: ce
--activation: sigmoid
```

You need a single tar.gz file containing the following:

```
- train.py
- run.sh (containing the best hyperparameters setting)
- any other python scripts that you have written
- supported.txt (as described above)
- report (in Latex) of the results of your experiments
- Kaggle_subs/ (folder containing all the Kaggle submissions)
```

The tar.gz should be named as RollNo_backprop.tar.gz.

- Your task is to achieve an error rate of $\leq 5\%$ on the test data. You will be evaluated based on how close you get to achieving this error rate.
- Along with the source code you need to submit a run.sh file containing the command (with hyperparameters) that gave you the lowest error rate on the public test set.
- In addition, we will also run your code using different hyperparameter configurations (for example, number of hidden layers, size of hidden layers, etc.). You will then be evaluated based on how good/bad your performance is compared to the performance of others on different hyperparameter configurations.
- And of course, you will also be evaluated based on which of the specified hyperparameters are supported correctly by your code.