A3
Deep Learning 2015, Spring

Dataset

For this assignment, you will be working with a set of preprocessed business reviews from the Yelp Dataset Challenge. (Note: Because the dataset was intended for limited academic use by Yelp, we ask that you do not release it independently after completing the assignment.)

Each data point is a string of a review's text, and each review is labeled with a star rating (an integer from 1 to 5). We would like to be able to predict a rating given the text of a review.

Two data formats are provided: one csv, and another t7b (torch-7-binary).

For the csv format, there are 2 columns. The first column is the class label, and the second column is a string with standard csv escaping (quoted with double quotes, and any double quote in the original text is replaced with ¨). There are no new lines in the original Yelp dataset distribution, so there is no worry about this.

For the t7b format, the string data is concatenated and stored in a very condensed byte tensor. You can load it directly. The data is stored in

```
/scratch/courses/DSGA1008/A3/data
```

and you can load using the following command

```
t7> train = torch.load(‘‘/scratch/courses/DSGA1008/A3/data/train.t7b’’)
```

Then, you will have a table containing 3 fields: index, length, content. Field "index" contains indices into content for each sample's string data in bytes. Field "length" contains the string length for each sample. Field "content" is a 1-D large ByteTensor containing all the string data, with proper NULL ending for each string.

This might look complicated, but it is the most memory efficient and garbage-collection friendly to do it. The way you use these data is also easy, just use the ffi package coming with luajit. For example:

```
t7> ffi = require(‘‘ffi’’)
```

To get the string for the 3rd sample in class 2:

```
t7> index = train.index[2][3]
```

```
t7> sample = ffi.string(torch.data(train.content:narrow(1, index, 1))
```

Now the sample is a lua string and you can do everything with it:

```
t7> print(‘‘The 3rd sample in class 2 is ’’..sample)
```

Check whether the ffi obtained string is same with our length

```
t7> match = (train.length[2][3] &=&  #sample)
```

```
t7> print(‘‘Does the length match? ’’..(match and ‘‘YES’’ or ‘‘NO’’))
```

Preprocessing
Data preprocessing is an important part of any machine learning problem, especially in statistical natural language processing. Most of the more unpleasant raw text processing has been done for you, but you will have to decide how to represent the strings in a way that your model can understand. The simplest input representation is the "bag-of-words" representation in which word order is discarded and documents are treated as simple counts or averages of their constituent words. This is the approach taken in the baseline code. Unfortunately, because word order is so important in language, bag-of-words representations are often inadequate for for complex tasks like document-level sentiment analysis, so you are encouraged to explore alternatives to improve your model's performance.

Task
Your task is to implement and train two models to perform sentiment classification on the Yelp review corpus. In the first part of the assignment, you will become more familiar with the structure of Torch's neural network implementations by writing a new nn.Module subclass. You must use the code outline provided with the baseline to implement a Torch module that performs log-exponential pooling, replace the max-pooling operation in the baseline with your new code, and tune the model's hyperparameters so that it performs as accurately as possible. Log-exponential pooling is a pooling operation with one free parameter and interpolates smoothly between average and max pooling (see this paper for more information on the properties of different pooling operations: `http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_BoureauPL10.pdf`)

It has the following form for $\beta \in (0, \infty)$:

$$\frac{1}{\beta}\log\left(\frac{1}{N}\sum_{i=1}^{N}\exp(\beta x_i)\right)$$

To verify your code for log-exponential pooling, you must write a 'gradient checker' that calculates finite difference approximations of the loss gradient with respect to each weight and shows that these approximations are almost exactly equal to the exact gradient obtained via backpropagation.

In the second part of the assignment, you should come up with your own model or implement one from a research paper and attempt to perform as well as possible on the same dataset. You are not required to use your module from the first part of the task - in fact, because the baseline performance is rather low, you will probably have to experiment with completely different architectures to get better results. You may not use additional external data explicitly, but you may use the GloVe (`http://nlp.stanford.edu/projects/glove/`) or word2vec (`http://code.google.com/p/word2vec/`) code or pretrained word vectors as part of the input to your model if you wish.

Note that for this assignment, the test dataset will not be provided, so you will have to decide on your own how to split the training data into training and validation data in order to maximize generalization to the test data.

Submission

To submit your work, create a folder TEAM_NAME in that contains at least a Lua script TEAM_NAME_A3.lua that reads an integer N and N strings from stdin and outputs only the N predicted class labels from 1 to 5. Your code will be placed into a qsub script and timed on Mercer, so try to make your model run as quickly as possible (this will not be weighted heavily and we will take model complexity into account, but when deploying models in practice, test speed is extremely important). This also means, you must get the name right. We will get your team name from the website under

`http://cilvr.nyu.edu/doku.php?id=deeplearning2015:teams`

Your paper should contain the following details about your approach to the second model:

- description of the architecture (number and type of layers, size of input, etc.)
- description of the learning techniques applied (which data augmentations?, used dropout?, etc.)
- description of the training procedure (learning rate, momentum, error metrics used, train/validation split, training/validation/test error)

Evaluation

- 30% - Test set performance - at least beat the baseline - there is no kaggle competition, your program is only going to see the test data at the very end. Make good use of a strong validation data set.
- 30% - Four page paper plus one extra page for references - you may mention your failed experiments and please use a NIPS/CVPR template (LaTeX).
- 20% - Simple, readable, commented, working code for log-exponential pooling that passes a gradient check
- 20% - Simple, readable, commented, working code for whatever algorithm you use for your second model