Assignment 3: Yelp dataset challenge

Priyank Bhatia
New York University
Center for Urban Science + Progress
1 MetroTech Center, 19th Floor
Brooklyn, NY 11201
pb1672@nyu.edu

Emil Christensen
New York University
Center for Urban Science + Progress
1 MetroTech Center, 19th Floor
Brooklyn, NY 11201
erc399@nyu.edu

Peter Varshavsky
New York University
Center for Urban Science + Progress
1 MetroTech Center, 19th Floor
Brooklyn, NY 11201
pv629@nyu.edu

Abstract

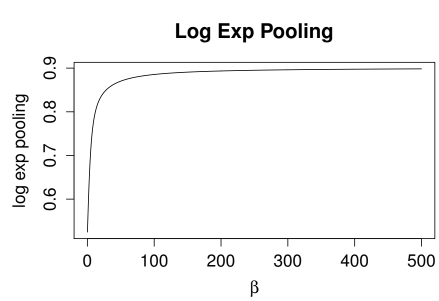
In this assignment we attempt to predict Yelp star rating using the text of the reviews. We implement a fully-connected linear neural network to predict bag of words and tf-idf weighted bag of words and achieve validation accuracy of approximately 50%.

1 Log Exponential Pooling

Log exponential pooling

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

can be used when max pooling is thought to discard too much information and average pooling assigns the weights too uniformly. As $\beta \to 0$ an application of L'Hospital's rule shows that log exponential pooling approaches average pooling $N^{-1}\sum_{i=1}^{N} x_i$. As $\beta \to \infty$ the exponential becomes dominated by the term with the largest x_i and the pooling function approaches max pooling. The figure illustrates the behavior of log exponential pooling over the vector $(0.24, 0.52, 0.1, 0.90, 0.84)$ with mean 0.52.



2 Architecture

The submission architecture is a simple shallow linear neural net. trained on 500,000 and validated on 50,000 samples.

```
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> output]
  (1): nn.Reshape(300)
  (2): nn.Linear(300 -> 600)
```

```
    (3): nn.ReLU
    (4): nn.Dropout
    (5): nn.Linear(600 -> 5)
    (6): nn.LogSoftMax
}
```

3 Preprocessing

The data were distributed with much of the preprocessing
complete. The words in each review were converted to lowercase and vectorized using a table of 300-dimensional GloVe [2] vectors. The resulting word vectors were simply averaged yielding a 300-dimensional input to the neural network. Each dimension of the bag of words average vector was then normalized across all observations.

3.0.1 tf-idf

Averaging word vectors discards a great deal of information, such as order or frequency of words, that can be useful for sentiment analysis. One way to include this information is to take a weighted average that favors words that are deemed more important higher than the less important words. Term frequency – inverse document frequency offers a weighting system that favors words that are common in a document, but not very frequent in the corpus. There are several formulations for the weights. We chose $\text{tf}(t, d)$ to be the number of time term t appears in document d, and

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where D is the corpus of documents, and N is the number of documents in D [1].

For a single hidden layer version of our model tf-idf converged at roughly 70% error. The loss of performance quality is likely due to changes of scaling introduced by tf-idf weighting. To attempt to negate the scaling effect we normalize each word vector by the average inverse document frequency $\overline{\text{idf}} = 12.175$. This improves the tf-idf results to 56% error, but still does not surpass the unweighted average.

Since idf weights are properties of the corpus, they were computed once and stored in a text file.

4 Training Procedure

The model was trained with fully stochastic gradient descent, minibatch size 1 without using a GPU. The error converged to the interval between 49% and 50%. Increasing the learning rate led to more oscillations within the (0.49,0.5) interval, but did not escape it.

5 Results

6 Next steps

Our team implemented convolutional models for concatenated words (as suggested in the assignment) representation and character-level representation of the documents as described in [3], but were not able to tune these models to produce better results than the baseline that we used in submission. If we were to keep working on the problem our next steps would be:

1. Implement baseline with minibatches and CUDA.
2. Implement concatenated word convolutional model on CUDA with batchches to mitigate memory issues we encountered when loading the entire dataset represented as concatenated words into RAM.
3. Compare performance of convolutional nets as metaparameters are varied.
4. Explore batch training for character-level model to avoid RAM limitations.
5. Explore the effect of changing weight random initialization distributions.

References

[1] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2012.

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[3] Xiang Zhang and Yann LeCun. Text understanding from scratch. *CoRR*, abs/1502.01710, 2015.