Sequence Model Computations speed-up using Multithreading for Sentiment Review Classification on AWS Cloud Resources

Abhishek Babuji
School of Informatics and Computing
Indiana University
107 S Indiana Ave, Bloomington, IN
47405
ababuji@iu.edu

INTRODUCTION

Naïve Bayes is traditionally known to be a common algorithm of choice for Text Classification using TF or TF-IDF as feature weighting scheme. In this paper, we explore our findings on how a Deep Neural Network architecture called Sequence Models can be used in a multi-threaded setting to speed-up training time and give superior performance compared to Naïve Bayes. The task chosen to illustrate this is a supervised sentiment review classification.

AWS S3 and AWS Sagemaker are used to provide infrastructure as a service for computation and storage. AWS Sagemaker is an integrated Jupyter notebook that can be used to perform exploratory data analysis and Machine Learning. AWS Sagemaker runs on cloud-based hardware offering any required amount of RAM and vCPUs to truly take advantage of multiple cores. AWS S3 is an object storage service that can be accessed through a web interface. Using TensorFlow as backend, it has been made possible to change the number of cores and run the exact same experiment with varied hyper parameters.

In this paper, the main focus is on researching the fastest setting possible to achieve a shorter training time. It is already conventional wisdom that Deep Learning; if given enough data is capable of outperforming traditional Machine Learning Algorithms in almost all tasks.

1. ARCHITECTURE AND IMPLEMENTATION

1.1 Naïve Bayes

Thirty six different combinations of the same dataset were created.

The combinations have the following options:

i. Stemming or Lemmatization or Default.
ii. Remove or keep stop words.
iii. TF or TF-IDF as a weighting mechanisms.
iv. Unigrams alone, Unigrams and Bi-grams or Bi-grams alone.

For a total of $3 \times 2 \times 2 \times 3$ combinations

`GridSearchCV` from `sklearn` module is used for hyper parameter tuning and maximum number of vCPUs (16) are used to find the best parameter setting to maximize our evaluation metric of interest – Accuracy.

1.2 LSTM

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell [1]. In the context of the current paper, the "time intervals" are steps from one word to another until the end of the sentence.

In the sentence, "This movie is awesome", the time steps can be seen as follows

"This": time step 1, "movie": time step 2, "is": time step 3, and "awesome": time step 5

A RNN using LSTM units can be trained in a supervised fashion, on a set of training sequences, using an optimization algorithm, like gradient descent, combined with backpropagation through time to compute the gradients needed during the optimization process, in order to change each weight of the LSTM network in proportion to the derivative of the error (at the output layer of the LSTM network) with respect to corresponding weight [1].

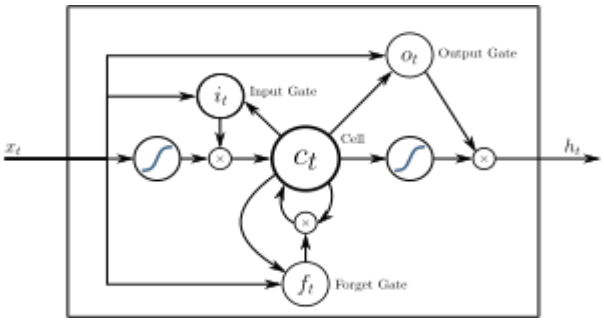


Figure 1. shows the representation of a single LSTM cell. An input in the form of a transformed word is transformed to an output probability [1]

1.3 Word Embedding vs. Sparse Vector Representations

For traditional machine learning algorithms, sparse vector representation is used as feature input. An example of the sparse vector representation is TF or Term Frequency [2].

Consider two rows of data:

Document 1	I bought a car
Document 2	car I bought a

When using Term Frequency as a feature scheme, the following set of features are produced.

	I	bought	a	car
Document 1	1	1	1	1
Document 2	1	1	1	1

The obvious disadvantages of such a feature representation are the following:

i. The ordering of the words does not seem to matter when generating features. Notice how the second document is malformed in the ordering of the sentence. Machine Learning algorithms will however see the two documents as equivalent.
ii. If the training examples have too many words, the matrix would end up becoming sparse with zeros for many columns in documents where certain words do not appear.
iii. Using such a feature representation makes it look like there is no dependency between the ordering of the words. By using these features, the position in which the word "car" appears is not dependent on the position in which the word "bought" appears, when in reality, the word "bought" appearing at time step 2 would indicate that a noun would appear in time step 4 – "car".

Instead of sparse vector representations using TF and TF-IDF, we introduce the usage of Word Embedding. A given word like "car" is represented by a matrix of a certain dimension.

These matrix representation of each word are trained in an unsupervised manner on a large corpus of text using an algorithm called GLoVe [3] .

Word embeddings are known to have the ability to capture context between different words in large sentences. Although the contents of the matrix are not interpretable, it is possible to compare these embeddings. In short, instead of a single word being associated with a number, a single word is now associated with its own unique matrix.

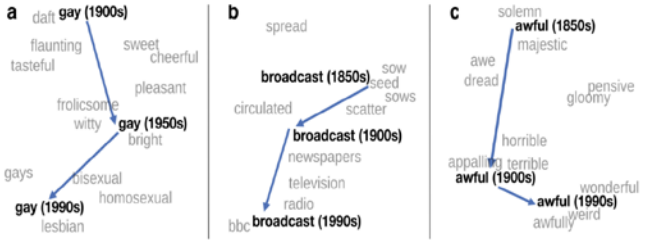
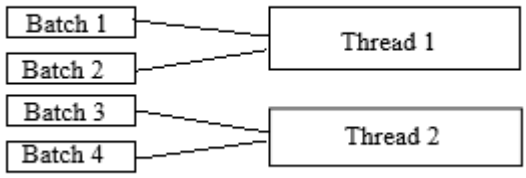


Figure 2. Word Embeddings Context Understanding

Notice from the side-by-side pictures above, how words could be understood context of other words near them.

1.4 Multithreaded setting in Sequence Models



The figure above shows how each batch having a certain number of training examples is handled by threads.

The consequent experiments and observations will illustrate how the batch size and changing the number of threads have an effect on the training time.

The usage of threads are varied between 16, 8, 4 and 2 and gradient descent convergence times are recorded.

2. EXPERIMENTS

The task involves detecting the sentiment of a certain review. Given three types of sentences rated as 'Good', 'Bad' and 'Average' from Amazon Review corpus from Kaggle, where there are 125323 'Good' reviews, 44896 'Bad' reviews and 15430 'Average' reviews, we use this data in a supervised setting to train the best model to maximize a given evaluation metric – Accuracy.

2.1 Naïve Bayes

Input: 36 different combinations of the same dataset varying feature weighting schemes such as TF and TF-IDF [3], stemming/lemmatization and stop words.

Output: Predictions on the test set - 15% of the data split using Stratified Shuffling.

Total number of results: 36

2.1.1 Preprocessing

The preprocessing simply involves creating thirty six different combinations of the same dataset as described in the Architecture section. The programming language and libraries used for this step are `Pandas, numpy` and `NLTK` using `Python`. The data is split into training set (85%) and test set (15%).

2.1.2 Hyper parameter Tuning

Using 5-fold cross validation with stratified shuffle splits on each fold, there are two hyper parameters that are tuned namely – `fit_priors` and `alpha`. The module used for hyper parameter tuning is `sklearn-GridSearchCV.`

2.1.3 Result

There were 36 different results. Since the point of this experiment is to exhaustively compare the best result of Naïve Bayes with LSTM we have decided to show the top four results obtained out of 36 results and the corresponding feature settings:

Stem, Lemmatize, Default	Stop words	Feature weighting Scheme	n-gram range	Training accuracy	Test accuracy
N/A	N/A	TF	bi	86.4%	87.2%
Lemmatize	N/A	TF	bi	86.2%	87.3%
Stem	N/A	TF	bi	86.2%	87.0%
Stem	N/A	TF	uni and bi	85.7%	86.5%

Table 1. Top four results using Naïve Bayes

There appears to be a cap on the results that a Naïve Bayes model is capable of generating.

Even with exhaustive feature engineering, it appears to be the case that the best possible result that Naïve Bayes is capable of offering is an 86% on the training set and 87% on the test set.

2.2 LSTM

Input: Temporal sequence of fixed length (60)

Output: Softmax probabilities

Total number of results: 16

2.2.1 Preprocessing

i. Convert text to lowercase.
ii. Remove stop words.
iii. Convert all contracted forms to auxiliary ("I've" becomes "I have", "I haven't" becomes "I have not").
iv. Encode numbers with a backslash in the beginning so it matches the key in the word embedding index (2 = "\2").

2.2.2 Tokenizing

Use `Keras` to convert each of the reviews into fixed length sequence of 60 by tokenizing the top 20000 words and padding them with 0s for reviews of word length less than 60.

For example:

The sentence "I love pizza" is converted into a 1×60 matrix: [0, 0, 0 … 2, 67, 21]

In this example, the numbers 2, 67, 21 are the indices of the words "I", "love", and "pizza" respectively in the embedding matrix.

2.2.3 Neural Network Architecture

1. Embedding Layer.
2. Dropout.
3. 1D Temporal Activation.
4. A max-pooling layer (64×5) with pool size = 4 and ReLU activation.
5. LSTM with number of units varied between 100 and 1000.
6. Dense soft-max activation as the output.

2.2.4 Result

We will split the result section into two parts. One part describing the accuracy obtained by running the setting for one hundred epochs and another part comparing execution times.

2.2.4.1 Tables

Batch size	Number of units	Training accuracy	Test accuracy
32	100	86.2%	85.6%
1024	100	92.1%	86.3%

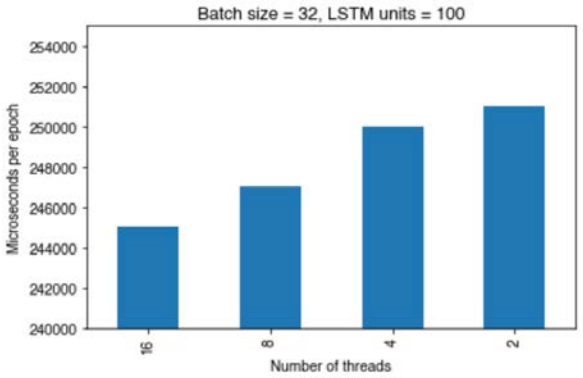
Table 2. Accuracy comparisons with changes in batch size

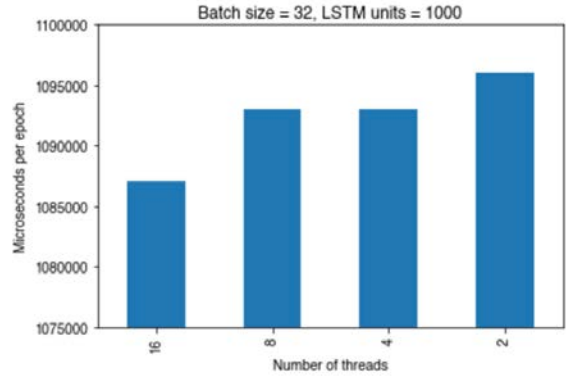
The accuracy comparison clearly shows that an accuracy of 92% is achievable using LSTM with word embedding features as opposed to Naïve Bayes with sparse vector features – 87%.

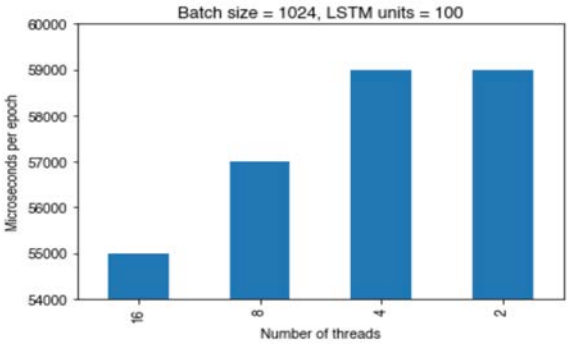
Batch size	Number of threads	Number of units	Average time per epoch
32	16	100	4m 5s
32	16	1000	18m 7s
32	8	100	4m 7s
32	8	1000	18m 13s
32	4	100	4m 10s
32	4	1000	18m 13s
32	2	100	4m 11s
32	2	1000	18m 16s
1024	16	100	55s
1024	16	1000	3m 57s
1024	8	100	57s
1024	8	1000	3m 58s
1024	4	100	59s
1024	4	1000	3m 59s
1024	2	100	59s
1024	2	1000	4m

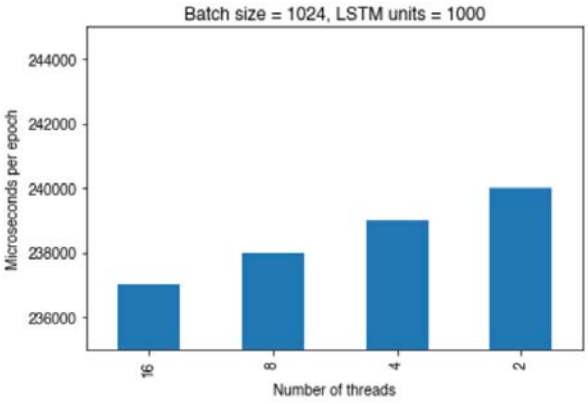
Table 3. Comparison of training time with changing batch size and number of threads used

2.2.4.2 Graphs: No. of threads vs. Execution time









3. CONCLUSIONS

1. Sequence Models provide a better accuracy than Naïve Bayes but they are computationally very expensive and require lots of training time.
2. Creating more number of batches increases the time taken for gradient descent to converge.
3. Maximum thread usage appears to be capped at one half of the number of vCPUs.
4. To achieve the highest accuracy, increasing the batch size so that there are fewer batches in total boosts not only accuracy but also drastically reduces the training time.
5. When there are less number of batches (when size of each batch is large), it appears to be the case that gradient descent takes significantly lesser time to converge [3].
6. When lesser number of threads are burdened with more number of mini-batches, the aggregation of the result of each mini-batch takes a lot of time.

4. ACKNOWLEDGMENTS

Our sincerest thanks are offered to three people in no specific order:

1. Mentor – Bo Beng: He helped identify the correct way to change the number of threads. He also helped summarize my conclusions and helped interpret the results. He was extremely patient and everything that he predicted would happen in the experiment; came to pass. Extremely knowledgeable when it comes to multithreading architectures for Neural Networks.
2. AI – Sellahatin Akkas: He helped resolve every single doubt related to the course and offered tailored, highest quality recommendation for the project review as well as the midterm reports without which it would've been impossible to proceed. He is the single best Associate Instructor.
3. Dr. Judy Qui: Understood the medical issues I faced and was very patient with every student and took care of each one's individual needs.

5. FUTURE WORK

The only recommendation for future work is to get more hardware resources and run gradient descent for as many more iterations as possible and possibly use generators with Spark to perform more comparisons.

6. REFERENCES

[1] W. contributors, "Long short-term memory," Wikipedia, The Free Encyclopedia., 14 April 2019. [Online]. [Accessed 28 April 2019].

[2] T. Joachims, "Text categorization with Support Vector Machines: Learning with many relevant features," *European Conference on Machine Learning*, pp. 137-142, 16 June 1998.

[3] S. Robertson, *Understanding Inverse Document Frequency on Theoretical Arguments for IDF*, Microsoft Research.

[4] N. S. M. S. S. S. S. C. R. Scott Sallinen, "High Performance Parallel Stochastic Gradient Descent in Shared Memory," *IEEE International Parallel and Distributed Processing Symposium.*

[5] J. P. R. S. Christopher Manning, "Glove: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.