Comparative Study of Sentiment Analysis using LSTM and CNN

Anurag Kumar[#1], Ryan Becwar[#2]

[#]*Department of Computer Science, Colorado State University*
Fort Collins, Colorado

[1]Anurag.Kumar@ColoState.edu

[2]Ryan.Becwar@ColoState.edu

Abstract— **We report on a comparative study on the effectiveness of LSTM and CNN models on natural language classification tasks. We have used customer reviews provided by Yelp open dataset for our study. Our results show that both models are effective solutions to this problem, and suggest that the CNN model has a slight edge in this particular case.**

Keywords— **Sentiment Analysis, CNN, RNN, LSTM, word2vec, Natural Language Processing**

I. Introduction

Sentiment Analysis is the process of determining the emotional tone of text as positive, negative or neutral. The sentiment analysis task can be approached in two ways i) Lexicon-based approach and Machine Learning approach. In this project, we are interested in exploring the effectiveness of supervised Machine Learning techniques in performing sentiment analysis. Machine Learning techniques are effective at extracting information from large datasets and classifying new information based on the existing information. Current research has positioned LSTMs and CNNs at the forefront of Machine Learning techniques for text classification: We seek to compare the effectiveness of these two methods by evaluating their performance for binary classification on the Yelp review dataset.

In section II, we will talk about the dataset, feature selection and preparing data for deep learning methods. In section III, we will discuss the basic working of RNNs, an overview of single LSTM unit and our LSTM model. In section IV, we give a brief description of CNN and talk about the required modifications for performing sentiment analysis. Section V summarizes the results from the two aforementioned deep learning techniques.

II. Data Processing

In this sentiment analysis project, we are using the Yelp reviews provided by the Yelp Open Dataset. The review data was provided in form of a large json file. We converted the json to a csv with each row being a review by the customer and each column as a feature associated with it. The features include *review_id, user_id, business_id, stars, date, text, useful, funny,* and *cool.*

Let us briefly describe the features, *review_id* is a unique alphanumeric string assigned to each review, *user_id* is a unique alphanumeric string assigned to each user/customer on Yelp, *business_id* is also a unique alphanumeric string assigned to each business that was reviewed on Yelp, *stars* is an integer in range of 1 to 5 assigned by the user describing his/her experience of the service, *date* contains the date of the review, *text* contains a somewhat detailed description of their experience in the English language, *useful* is an integer describing the number of users who found the review useful, and similarly, *funny* and *cool* are also integer values representing the number of people who think that the review was funny and cool respectively.

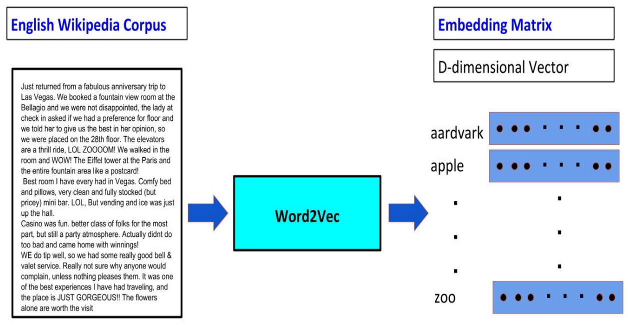
As we are not interested in analyzing the users or the businesses, we decided to drop *user_id* and *business_id* features from our dataset. Since our goal for this project was just to perform sentiment analysis of the reviews, we decided to keep our dataset as simple as possible and therefore we also dropped *review_id, date, useful, funny, cool* from our dataset.

A. *Subset of Reviews Dataset*

After removing the extra features, we are basically left with the *text* and the *stars*. Still, the dataset is huge as it contains close to 6 million reviews. Since deep learning techniques are computationally intensive, we decided to subset 10000 samples randomly from our modified dataset.

So for our project, the *text* is the input data for our models and *stars* are the class label associated with each input. Since the labeling is assigned by the users and there are no standard guidelines for labeling, therefore it is difficult to differentiate between reviews with labels 4 and 5 or 1 and 2 or 2 and 3 objectively. But, we can say with absolute certainty that a review with label 1 is bad and a review with 5 is good. Therefore, we decided to drop reviews with labels 2, 3, and 4 from our dataset. Now, our modified dataset contained 5889 samples, 75% positive and 25% negative samples.

As we all know that the input and output in all machine learning techniques are scalar values, therefore we need to convert our inputs and outputs into appropriate formats. For this purpose, we will be replacing all the words in the reviews with integers based on a word list containing 400000 most frequent words from Wikipedia's English corpus. Then, we will be converting each word to a d-dimensional word vector using pre-trained word2vec [4]. Fig. 1 shows the conversion of words to D-dimensional word vectors using word2vec.



Just returned from a fabulous anniversary trip to Las Vegas. We booked a fountain view room at the Bellagio and we were not disappointed, the lady at check in asked if we had a preference for floor and we told her to give us the best in her opinion, so we were placed on the 28th floor. The elevators are a thrill ride, LOL ZOOOOM! We walked in the room and WOW! The Eiffel tower at the Paris and the entire fountain area like a postcard!
Best room I have every had in Vegas. Comfy bed and pillows, very clean and fully stocked (but pricey) mini bar. LOL, But vending and ice was just up the hall.
Casino was fun. better class of folks for the most part, but still a party atmosphere. Actually didnt do too bad and came home with winnings!
WE do tip well, so we had some really good bell & valet service. Really not sure why anyone would complain, unless nothing pleases them. It was one of the best experiences I have had traveling, and the place is JUST GORGEOUS!! The flowers alone are worth the visit

Fig. 1 Converting Words to D-dimensional vectors using Word2Vec

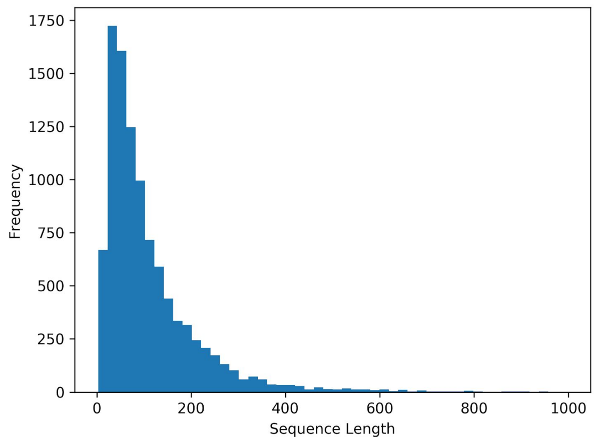


Fig. 2 Number of reviews (Frequency) Vs. Number of Words (Sequence Length)

Fig. 2 is a histogram of a number of words in reviews in our modified dataset. We can see that most of the reviews contain less than 200 words. Therefore, we decided to pick the maximum sequence length as 250 words which means we will only consider the first 250 words from each review. If the review contains less than 250, the sequence vector will be padded with zeros.

Let's walk through an example shown in Fig. 3. We have a review "*Got the breakfast sandwich which was great*". Let's convert this sentence into an integer sequence of length 10 based on the word list that we mentioned earlier. It would be [405

201534 5980 12611 42 15 353 0 0 0]. Now, we can use the embedding matrix to create a sequence vector of dimension 10*50. We used the tensorflow.nn.embedding_lookup function which returns a 50-dimension vector for each word. This 50-dimension vector helps us build context by associating with other words in the sentence. This 10*50 sequence vector represents one review.

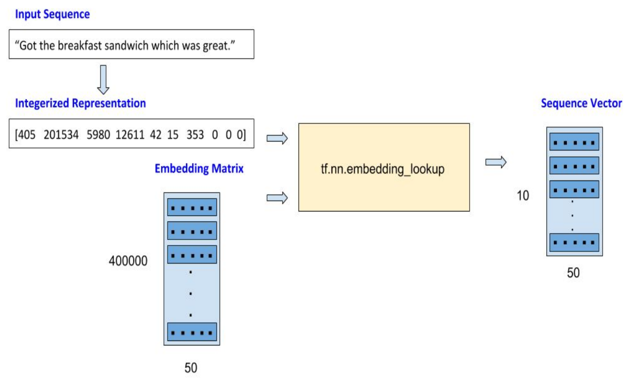


Fig. 3 Input Sequence to Sequence Vector

In our case, we will have a sequence vector of 250*50 for each review and we will have the labels in one-hot notation i.e. [1, 0] and [0, 1] for positive and negative samples respectively. Fig. 4 summarizes the input and output in a format which can be passed to any machine learning model.

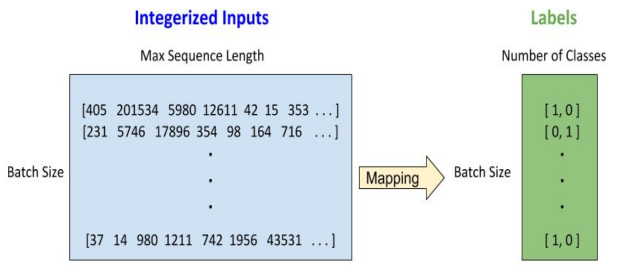


Fig. 4 Integerized Inputs and Outputs

III. LSTM

Before we get into LSTMs (Long-Short Term Memory) which is a special type of RNN (Recurrent Neural Network), let's talk about why RNNs or LSTMs makes sense for natural language processing and reason for that is the temporal nature of the language i.e. the meaning of a sentence depends not only on the words in that sentence but it also depends on the order in which they appear. This temporal nature makes RNNs or LSTMs more suitable for natural language processing. Fig. 5 shows an example of a sentence with words with time stamps representing the temporal nature.

Got	the	breakfast	sandwich	which	was	great
x_0	x_1	x_2	x_3	x_4	x_5	x_6
t = 0	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6

Fig. 5 ASample Review

Now, let's look at the working of a RNN as represented in Fig. 6. Each word vector x_t is associated with a new hidden state vector h_t which seeks to encapsulate and summarize all the information from the previous time steps.

$$h_t = \sigma(W^H h_{t-1} + W^X x_t)$$

Eq. 1 Calculating Hidden Vector in RNNs

It can be calculated by multiplying word vector x_t with a weight matrix W^X which is different for each input and adding the product of hidden vector h_{t-1} from the previous time step and a weight matrix W^H which constant across all time steps.

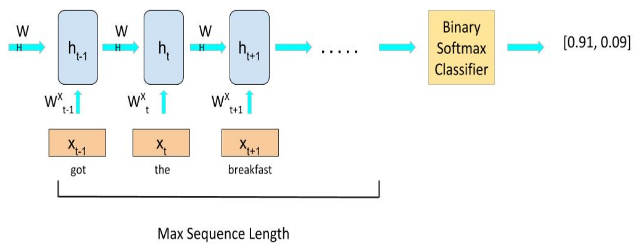


Fig. 6 Data flow and Hidden units in RNNs

The sum of the two products described above is given to an activation function to get the hidden state vector for the present time step as shown in Eq. 1. This is repeated until we process all the information in the input sequence vector and for all inputs in our training/test batch. The hidden state vector from the last time step is then passed to a binary softmax classifier which gives the output as the probability for each class.

Now that we understand RNNs, let's look at the LSTM unit shown in Fig. 7. Similar to RNNs, every word vector x_t is associated with a hidden state vector h_t, but the update process of h_t is more complex than RNNs.

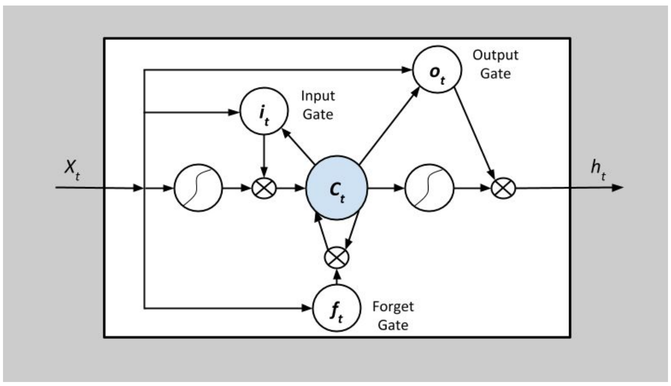


Fig. 7 A single LSTM logic unit

The LSTM unit consists of four components, an input gate (i_t), an output gate (o_t), a forget gate (f_t), and a memory container (c_t). The word vector x_t and hidden state vector from previous time step h_{t-1} (not shown in the Fig.) are given as inputs to all the gates. The forget gate determines what information should be removed from the h_{t-1} depending on the present x_t. The input gate determines what information should be added to c_t *depending on* c_{t-1}, h_{t-1}, and x_t. And the output gate determines what information should be added to the h_t depending on c_{t-1}, h_{t-1}, and x_t. We repeat the same process until the end of the sequence vector. This is just a high-level explanation of the inner working of the LSTM unit.

In our implementation, we stacked multiple LSTM units and the hidden state vector of the time step was given to binary softmax classifier which gave us our predictions in terms of the probability and we assign the label of the class with max probability. The results from our LSTM model is discussed in section V.

IV. CNN

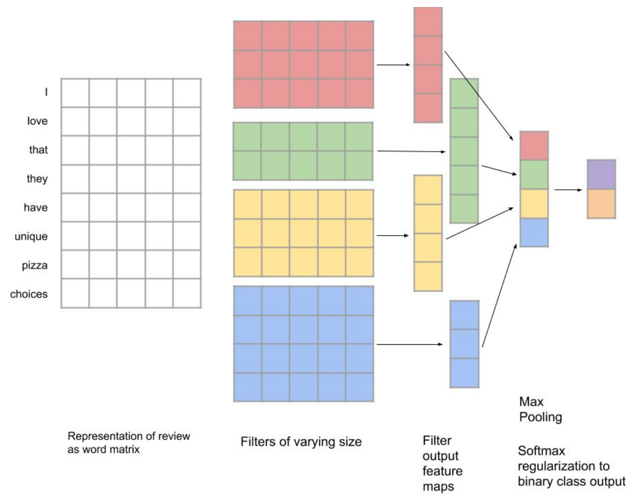
Similarly to how LSTMs capture context through lags, Convolutional Neural Networks are able to capture the combined context of phrases by training filters which convolve across an input text. These filters can utilize patterns such as negation, allowing the network to distinguish between phrases such as "not great" and "great".

While LSTM models have the ability to "understand" an extended sequence across the input text CNN models are limited to semantic relations that appear within the width of the convolutional filters. However, substantially increasing filter size can lead to overfitting and reduce the effectiveness of the model. NLP CNN models, therefore, perform best in classification problems where the input text can be classified by observing the appearance of certain types of short phrases.

Our CNN model is implemented closely following the process described in Yoon Kim's paper [7], modifying hyperparameters to better support the larger input size(replacing sentences with full reviews), and produce a binary classification result.

Given the *n*k* review input matrix generated with *word2vec*, the implementation begins with a convolution layer, which narrowly convolves multiple *w*k* filters of varying size *w* where *w* is the number of words vectors in the filter. The activation function then produces a feature map from each filter. Max pooling reduces each feature map to a single unit, which is then concatenated to form a feature vector. A softmax regularization function with a 50% dropout rate is then used which produces a 2-dimensional output layer for the binary classification result.

We chose to ignore words which were not present in the vocabulary of *word2vec*. This may have had some negative impact on our model, as it drops contextual information contained in misspelled words.



I
love
that
they
have
unique
pizza
choices
Representation of review as word matrix
Filters of varying size
Filter output feature maps
Max Pooling
Softmax regularization to binary class output

V. Results

A. LSTM

Fig. 8 shows the training accuracy of the LSTM model with a stack of 64 units trained on 4000 samples in batches of 20 randomly drawn samples over 100000 iterations. The training time for this particular model was little over 2 hours.

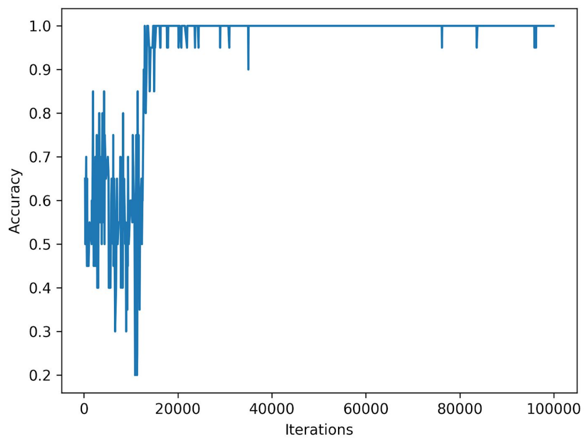


Fig. 8 LSTM Training Accuracy

Fig. 9 is a histogram representing the accuracy of the test set. Each set batch was randomly drawn from the remaining 1889 samples with equal representation of both positive and negative reviews.

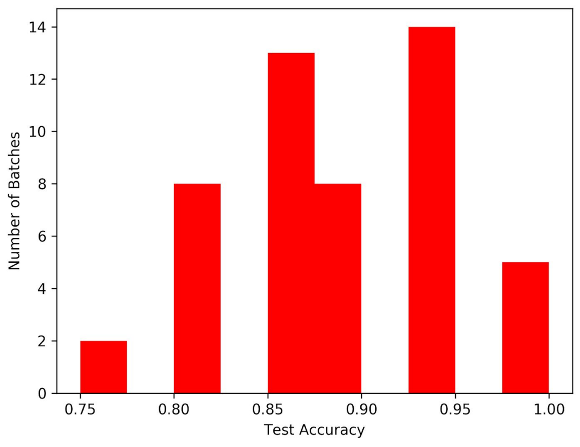


Fig. 9 Histogram of Test Accuracy for different batches

Fig. 10 shows the average test accuracy of LSTM models as a function of a number of LSTM units in the stack. We can clearly observe underfitting in models with 16 or fewer LSTM units and overfitting in models with more than 64 LSTM units.

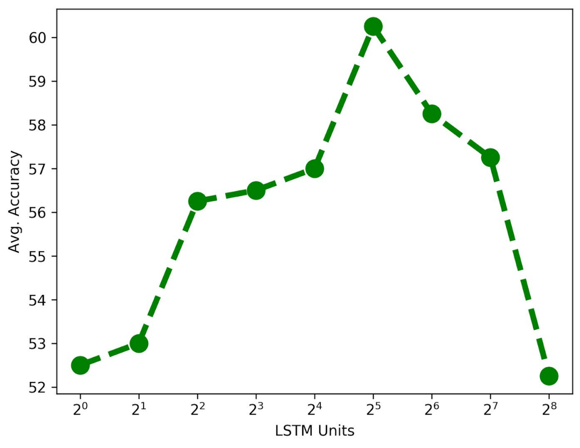


Fig. 10 Avg. Test Accuracy Vs. Number of LSTM Units

Lastly, we can say that the accuracy of different test sets ranged between 75% and 100% with an average test accuracy of 88.90% on our modified dataset.

B. CNN

Our CNN model provided a test accuracy of 95.1297% on the shortened dataset, after training for approximately 10000 iterations on the same 4000 sample training set used for our LSTM and the same test set.

It appears that the accuracy of the model begins to converge at approximately 5000 iterations, which suggests that further improvements could be made by expanding the training set size. We sought to use generic and generally applicable parameters, but hyperparameter tuning would likely improve performance as well.

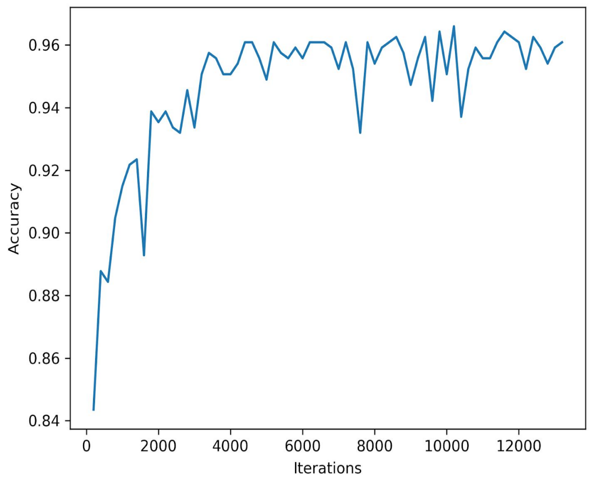


Fig. 11 CNN Validation set accuracy as a function of training iterations

VI. Conclusions and Future Work

Our results show that the CNN model has a slight edge in performance over the LSTM model. This is likely due to the fact that this is largely a sentiment classification problem(1 star reviews will be overwhelmingly negative, while 5 star reviews will be positive), and the sentiment of a text can largely be determined by the occurrence of positive or negative phrases. CNN models excel at detecting the occurrence of phrases which match a pattern, while LSTM models are better suited for “understanding” long semantic relations which occur over the course of the input text. It is likely that LSTM models would beat CNN models in a situation such as determining which side of a debate issue an input text advocates for.

Our future work would include testing these models on various datasets with different classification goals to verify this hypothesis. Also, we would benefit from a more detailed study on hyper parameter tuning in CNN models, and studying how the filter size affects performance on datasets with varying levels of semantic complexity.

Acknowledgment

We would like to thank Yelp for providing a well-labeled dataset which we used for performing the sentiment analysis in this project as well as the Department of Computer Science, CSU for funding our poster presentation. Our implementations were heavily guided by the processes laid out in [5] and [6], and we would like to thank the authors for laying the groundwork for these models.

Individual Contributions

A. *Anurag Kumar*

I helped in understanding and processing of the raw dataset that I downloaded from yelp. I converted the jsons into csv and created a subset of the dataset based on the features that we needed for our models. I downloaded an integer-word mapping of English language based on 400000 most frequent words taken from Wikipedia’s English corpus. I downloaded a pre-trained word2vec [4] trained on the above-mentioned corpus and used it to create the embedding matrix for the words in our dataset. I created helper functions for training and testing. I used predefined methods from tensorflow to implement a stacked LSTM model with a varying number of LSTM units. I also contributed to designing the poster and all the images related to data processing and LSTMs sections. Similarly, I also helped in writing, editing and formatting this paper.

B. *Ryan Becwar*

I designed and implemented the CNN model and experiments, handled *word2vec* and input data for that experiment, created the relevant functions for testing and training, and assisted in the initial data pre-processing stage. I worked in the writing and design of the poster, creating the CNN images and experiment results. I also worked in the writing and editing process of this paper.

References

[1] Daniel Jurafsky and James H. Martin, *Speech and Language Processing*, 3nd ed. draft, 2018.

[2] Lei Zhang, Shuai Wang, Bing Liu. Deep learning for sentiment analysis: A survey. arXiv:1801.07883.

[3] Milkolov et al. Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems 26 (NIPS 2013).

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[5] Adit Deshpande (2017, July 13). Perform sentiment analysis with LSTMs, using TensorFlow [Blog post]. Retrieved from https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow .

[6] Denny Britz (2015, November 7). Understanding Convolutional Neural Networks for NLP [Blog post]. Retrieved from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ .

[7] Yoon Kim. Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882v2