# LIGN 167
# Unit Summary Aggregate Minimization ($USA_{MIN}$) Using LSTM Neural Networks

Yuneng Jiang (A12757262), Michael Tran (A12023051)
Brian Suh (A11178981), Ryan Fong (A11656817)

February 18, 2019

**Abstract**

Recurrent Neural Networks (RNNs) have become vastly popular, especially in the realm of natural language processing. One of the problems of typical implementations RNNs, such as Elman Networks, is the exploding/vanishing gradient problem, which often occurs when long quantities of text are processed at once. To overcome this, Long Short Term Memory networks (LSTMs) emerged, allowing texts to be processed with minimal fear of the exploding/vanishing gradient problem. With LSTMs in mind, we experiment to see if a movies summary can be used to predict how well a movie is received, judging based on scores given by its viewers.

## 1 INTRODUCTION

### 1.1 What task are we trying to solve?

Finding empirical evidence showing that a movies summary correlated to its viewer rating.

### 1.2 What dataset did you use for training/evaluating your model?

For obtaining vector representations of the words in the summaries, we used Stanfords GloVe pre-trained word embeddings.

The primary dataset that was used in this project is taken from Kaggle [2]. The dataset is officially labeled as the TMBD 5000 Movie Dataset. The profile on Kaggle includes two CSV files containing an encompassing amount of information for approximately 4800 movies. The information ranges from the budget, genre, languages, release dates, etc.

The file that was most useful for our purposes was the file `tmdb_5000_movies.csv`. This file contained most of the information about the movies, as the other file `tmdb_5000_movies.csv` contained only four attributes that were deemed unnecessary.

From `tmdb_5000_movies.csv`, we decided to primarily utilize the attributes `title`, `vote_average`, and `overview`.

## 2 Data Preprocessing

### 2.1 Selecting Data from the Kaggle

The list of movie data from Kaggle started with 4800 entries. However, some of these entries did not have any voter average data. Since this data is crucial to training our model, we removed all entries that had no voter average data (voter average of 0). This is to prevent unnecessary skew of our model. After this filtering, we were left with 4740 entries of summaries and their respective scores that range from 1 through 10.

### 2.2 Parsing the summaries into words and punctuations

Stanfords GloVe pre-trained word embeddings contain vector representations of many common words and punctuations. To use this, we must parse each summary into words and punctuations. Furthermore, all of the words and punctuations needed to be in lower case as GloVe only contained lower case words. One special case we took into consideration during the parsing process is the string s. This string had its own vector representation within the GloVe pre-trained set. As such, during parsing for any instance of this string, the punctuation and the letter is considered one string and not separated into their own entities.

### 2.3 Determining training sets and testing sets

To determine our training sets and testing sets, we randomly partitioned our entire set of summaries into 80% training set and 20% testing set. Because we could not keep track of the scores of each summary with the summary alone, we also needed a way to associate each summary to its respective score. We decided to use a list to store the indices of the summaries inside the training set and testing sets, and we returned these two lists along with our sets.

## 3 Training

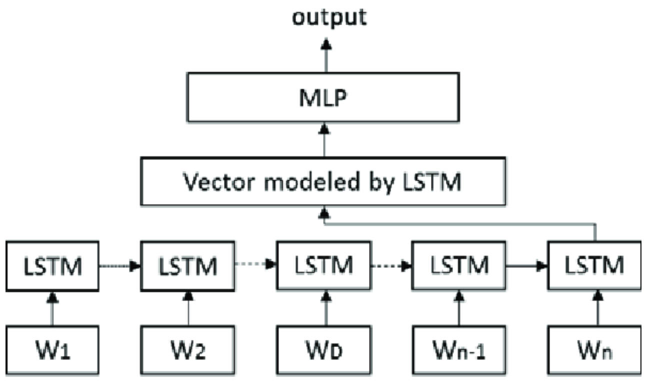
### 3.1 What model did you implement?

We implemented an LSTM in order to get a vector representation for a given summary which feeds into an MLP to receive a predicted user rating.

Here is a mathematical definition of the LSTM according to PyTorchs documentation:

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}$$

where $h_t$ is the hidden state at time *t*, $c_t$ is the cell state at time *t*, $x_t$ is the input at time *t*, $h_{(t-1)}$ is the hidden state of the layer at time *t-1* or the initial hidden state at time *0*, and $i_t$, $f_t$, $g_t$, $o_t$ are the input, forget, cell, and output gates, respectively. $\sigma$ is the sigmoid function.



## 3.2 Obtaining a vector representation of a summary

Before we can input anything into an MLP, we needed a way to obtain a vector representation of an entire summary. To do this, we used an LSTM with an input size of 1 x 50 and an output size of 1 x 50. This size was determined by the Stanford GloVe embeddings as they only had dimensions of 50, 100, 200, and 300.

The initial hidden state of the LSTM was set to a vector of appropriate size and initialized such that all entries are 1. This is a design choice as a randomized initial hidden state may introduce variations in the expected score the MLP outputs.

We then feed the vector representation of all the words inside a certain summary into the LSTM sequentially and we took the final hidden state as a vector representation of all the words inside of a summary. If a string had no vector representation within GloVe, it was skipped and not considered within the final vector representation of a summary. This was a conscious design choice as we had no idea of how to represent a string not found with the GloVe database as this GloVe is a pre-trained dataset and a valid representation for such a string must consider all the other entries inside of GloVe. Also, it was unreasonable to do so given the time constraint.

## 3.3 The specifics of our LSTM

The LSTM utilized exists within the PyTorch library (specifically under torch.nn) and is a black box. The LSTM returns two items: a cell state and a final hidden vector. For our purposes we only utilize the final hidden vector, passing it into our Multi-layered Perceptron (MLP).

## 3.4 The specifics of our Multi-layered Perceptron (MLP)

We decided to use an MLP with 50 input nodes, 5 hidden layers, and 1 output node. This was coded using the PyTorch neural network package and we decided to use the nonlinear activation function ReLU for simplicity sake.

Mathematical Representation of the final layer of the MLP, the result `predicted_value` is the output.

$$\begin{aligned}\text{predicted\_value} &= w_0^5 * h_0^4 + w_1^5 * h_1^4 + \cdots + w_{99}^5 * h_{99}^4 \\ &= W^5 \cdot \vec{h}^{\,4} \qquad dim(W^5) = 1 \text{ x } 100\end{aligned}$$

The vector $h$ is our hidden state after the application of the nonlinear activation function ReLU and it is defined as

$$\begin{aligned}\vec{h}^{\,k} &= \begin{bmatrix} h_0^k \\ h_1^k \\ \vdots \\ h_{99}^k \end{bmatrix} = \begin{bmatrix} ReLU(r_0^k) \\ ReLU(r_1^k) \\ \vdots \\ ReLU(r_{99}^k) \end{bmatrix} \qquad (1) \\ &= ReLU(\vec{r}^{\,k}) \qquad k \in \{1, \quad 2, \quad 3, \quad 4\} \qquad (2)\end{aligned}$$

Similarly, the vector $r$ represents the products obtained from the previous layer defined as

$\vec{r}^{\,k} = W^k \cdot \vec{h}^{\,k-1}$

Finally, the first hidden layer depends on the input layer and it is defined as

$\vec{r}^{\,0} = W^0 \cdot x \qquad dim(W^0) = 100 \text{ x } 50 \qquad dim(x) = 50 \text{ x } 1$

## 3.5 Training our MLP

Training our MLP was rather simple as we used the PyTorch neural network package. For our loss function, we chose to use the Mean Square Error Loss Function as it was more robust to outliers than the Mean Absolute Error Loss Function. Gradient descent was performed via the PyTorch optimizer with Stochastic Gradient Descent as the descent method and a learning rate of 0.0001. The MLP was trained on 3792 (80% of our entire data set) entries.

## 4 Results

### 4.1 How did you quantify/evaluate the models performance?

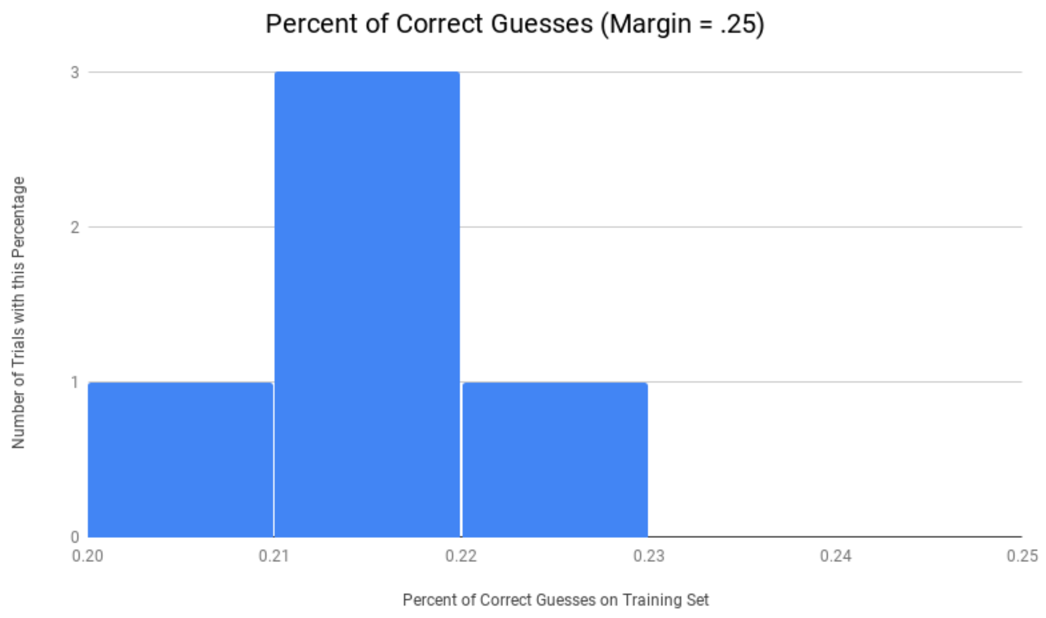
We deemed a prediction to be accurate if it is within 0.25 of the actual user rating. This number is inspired by the p-value for statistical significance as there is only a 5 percent chance of randomly generated rating to be within 0.25 of some given user rating. With this premise, we iterated through every entry in the list mentioned in the previous section and checked to see if the difference was within 0.25. Keeping track of the number of differences that were within 0.25, we then took the final total of this value and divided it by the size of the list. The result yielded represents how accurate our model was.

### 4.2 How well did your model perform on your task?

Abysmal. Our model did not perform as well as we anticipated. We were hoping our model would be able to predict the user ratings, at the very least, more than 50% of the time. However, this was not the case. Despite tweaking different parameters, e.g. testing different learning rates, adding more hidden layers to the MLP, changing the nonlinear activation function, we could not get an accurate model that would fulfill our expectations.

### 4.3 What were your results?

We had an accuracy of 20% across all the GloVe datasets (50, 100, 200, 300 dimensional) with a margin of .25. This means if difference between our predicted rating and the actual rating was $\leq .25$ we counted it as a success. The accuracy increased to around 70% when we increased the margin to 1. On average, the difference was around .77.



## 5 Conclusion

### 5.1 Concerns about summaries and user ratings from the dataset

One factor that might have contributed to our low accuracy could be differences in writing styles. Summaries are written by humans and different people will most likely generate a different summary for the same movie.

Our data on user rating (vote average) may be skewed as well. Popular movies will have many users voting and will, therefore, yield a more accurate rating for the movie. Movies that are not as popular will not have as many people rating it and as a result, might not yield the most accurate of ratings.

### 5.2 Concerns about our LSTM

As explained in our procedures above, we only used the PyTorch LSTM as a black box to obtain a vector representation for a given summary. We could not appropriately train the LSTM as we did not have a reference on what a good vector representation looks like for a given summary. Therefore, we could not effectively define an appropriate loss function and as a result, we could not perform any advantageous type of descent method on weights inside the LSTM.

### 5.3 Concerns about GloVe

As mentioned in section preprocessing, all of the words inside of GloVe are lowercase strings. This can cause problems when capitalization inside a string actually matters. Consider the case of James Bond, a popular fictional character in the 007 movie series, and the verb bond. Because everything needs to be converted into lower case in order to use the word embeddings inside GloVe, the LSTM will most likely be receiving the verb version of the string bond to use in its calculation of the final hidden state.

### 5.4 Concerns about the correlations between a movies summary and user rating

A movie is composed of many different aspects. Quality of the acting, popularity of the actors, visual directing, and even background music, are examples that can contribute heavily to a films rating. Therefore, a summary arguably is not enough to capture any of these aspects and is, therefore, an incomplete representation of the entire film.

However, there seems to be a weak correlation between a movies summary and its rating as we obtained an accuracy of 20%. As mentioned above, we deemed a rating to be accurate if it is within 0.25 of the true rating. If the ratings are not correlated with the movies summary we can expect an accuracy rating of 5% as there is a 5% chance of a randomly generated number (between 1 and 10) to fall within a 0.25 range of a fixed number (between 1 and 10). The fact that we obtained a 20% could imply that a select few summaries are more favorable than others.

Though varying in level, ratings are ultimately subjective, as mentioned previously. To elaborate on this, there is much that can influence an audience members rating on a movie. The aspects of a movie being one type of influence, an audience members personal experience with the movie, his or her emotional state when making the rating and his or her own standards on what is considered a good movie are all possible influences. One summary of a movie is just one instance of a way to describe that movie; many other summaries can be made for the same movie. As such, one summary is unable to encompass well the subjectivity that is the voter average.

## References

[1] https://nlp.stanford.edu/projects/glove/

[2] https://www.kaggle.com/tmdb/tmdb-movie-metadata

[3] https://pytorch.org/tutorials/beginner/nlp/sequence_model_tutorial.html