CS221 Spring 2019 - Roger Ebert Viewer

Ivan C.H. Ho*
Stanford University
ivanho1@stanford.edu

Abstract

This report describes method and results of using Long Short-Term Memory (LSTM) recurrent neural networks to generate Roger Ebert's movie reviews and sentiment analysis based on his existing articles as training data.

1 Task definition

I have always been fascinated by movies growing up. Not having the means (nor time) to watch every movie, I relied on movie reviews to narrow my selection. Roger Ebert's weekly movie review became my source of information: an insightful yet discreet essay on every new movie showing in North America. Ebert's writing stretches beyond the movie and into humanity. Ebert passed away in 2013, but every review he has made, dating back to 1967 is available online. At a rate of 3 movie reviews a week, that's over 7000 movie reviews he has left for the world to treasure. I would like to use this information (in addition to movie meta data) to train an AI model that can generate a short review essay on any movie, and predict movie rating based on Ebert's data from:

`https://www.rogerebert.com/`

2 Approach

The approach is broken into two parts: first is to generate review text with movie meta data as input; second is to perform sentiment analysis based on the output text to give a movie rating (thumbs-up vs. thumbs-down).

2.1 Word generation

To generate the reviews, I have chosen to use Long-short-term memory (LSTM) (1), which is a kind of recurrent neural network (RNN) (2) that can combat exploding or vanishing gradient problem inherent in other RNN architectures. Referencing (**?**), I was able to use Roger Ebert movie reviews as data to train a an LSTM model to generate words. The model summary is captured in Figure 1.

2.2 Sentiment analysis

Second part of the approach is to generate sentiment (rating) based on the output text. Again, LSTM architecture is used to perform sentiment analysis on the generated text to predict the rating. Referencing (5), I used Ebert's movie reviews and ratings as data to training the model. Summary in Figure 2.

*(https://github.com/alrightyi)

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

```
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 256)               19288064
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 18707)             4807699
_________________________________________________________________
activation_1 (Activation)    (None, 18707)             0
=================================================================
Total params: 24,095,763
Trainable params: 24,095,763
Non-trainable params: 0
```

Figure 1: LSTM model summary for word generation

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 454, 32)           32000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 454, 32)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 48)                15552
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 98
=================================================================
Total params: 47,650
Trainable params: 47,650
Non-trainable params: 0
```

Figure 2: LSTM model summary for sentiment analysis

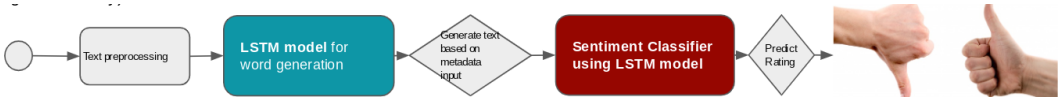

Figure 3: Overall REview pipeline

2.3 Overall pipeline

Combining the two models, along with pre-processing the data from rogerebert.com, I was able to create a software pipeline that can take an arbitrary movie as data input (along with its meta data like cast and crew, genre, running time) and generate the following: snippet of review text, and predicted movie rating. Pipeline is illustrated in Figure 3.

A Github repository where all the source code for this project is stored is here:

`https://github.com/alrightyi/cs221-project`

3 Data and experiments

3.1 Input data

Some statistics on Roger Ebert's movie reviews:

Total reviews by Roger Ebert: 7,463
Corpus length: 5,204,292
Average words per review: 697.35
Max words per review: 2,277
Min words per review: 1
Total unique words: 110,449
Total thumbs-ups: 4,397
Total thumbs-downs: 3,066

Input data goes through pre-processing to translate all characters to lower-case, as well as stripping off non-alpha-numeric characters except line-breaks.

Thumbs-up based on rating >= 3. With zero stars being the lowest, and 4 stars being the highest.

Due to limitation in memory, only the last 5 paragraphs of each movie review for training were retained as this would limit total generated number of word sequences.

Additionally, movie meta data is built into the corpus as part of the input, resulting in a sample that looks like this:

"jacobs ladder 1990 cast
tim robbins as jacob
elizabeth pena as jezzie
danny aiello as louis
matt craven as michael
pruitt taylor as paul
jason alexander as vince geary
directed by adrian lyne
produced by alan marshall
written by bruce joel rubin
photographed by jeffrey l kimball
edited by tom rolf music by maurice jarre drama mystery science fiction thriller rated r 115 minutes
the key performances in the film are by robbins and elizabeth pena who plays the woman he lives with its difficult to evaluate their work because the movie sets them the task of behaving in an utterly realistic sliceoflife manner in many scenes even some which are later revealed as hallucinations and then coasting away into fearsome fantasy in other scenes pena achieves the difficult task here of creating a believable and even sympathetic woman while at the same time suggesting dimensions that the hero can only guess at most films tell stories jacobs ladder undoubtedly contains a story which can be extracted with a certain amount of thought since the ending can be read in two different ways however the extraction process could result in two different stories that isnt the point what jacobs ladder really wants to do is to evoke the feeling of a psychological state in the audience we are intended to feel what the hero feels"

Code to read data and generate statistics: `read_data.py` Full corpus is store in `roger_ebert_last5.txt`

3.2 Word generation

Code to train the word generation model: `word_generation.py` The hyper-parameters corresponding to the word-generation model are:

- MIN WORD FREQUENCY: min number of times a word has to appear in the corpus to be added to the training set. (default set to 5) This limits unnecessary calculations on rarely used words.
- SEQUENCE LEN: number of words constructed into a sequence and fed into the model as X (default set to 140) as this is the average size of a review plus metadata.
- STEP: sliding window size to the next word sequence (default set to 10)
- DROPOUT: dropout rate (default set to 0.5)
- BATCH SIZE: batch size (default set to 32)
- SPLIT: train/val/test split (default set to 0.2, meaning 0.6/0.2/0.2 split)

- EPOCHS: number of training cycles (defaults to 50)
- EARLY STOPPING: number of training cycles without improvement before stopping (defaults to 10)

3.3 Sentiment analysis

Code to train the sentiment analysis model: `sentiment_analysis.py` The hyper-parameters corresponding to the model are:

- MAX FEATURES: max number of unique words in the vocabulary. Use to control the number of features in the model (default is 2000)
- EMBED DIM: dimension of the dense embedding from the first layer of the model. (default is 64)
- LSTM OUT: dimension of the LSTM layer output (default is 96)
- DROPOUT: dropout rate (default set to 0.5)
- BATCH SIZE: batch size (default set to 32)
- SPLIT: train/val/test split (default set to 0.2, meaning 0.6/0.2/0.2 split)
- EPOCHS: number of training cycles (defaults to 50)
- EARLY STOPPING: number of training cycles without improvement before stopping (defaults to 10)

4 Analysis

Using the hyper-parameters listed above, both models were able to train with increasing accuracy on the training data. However, validation data accuracy stays about the same without any improvements. Early-dropping kicked in and ended both training before the set number of epochs were run. See Figures 1 and 2.

4.1 Word generation

Typical reason for validation accuracy not improving is due to model over-fitting the data. The following hyper-parameter values were then modified to see if the results would improve:

- SEQUENCE LEN: shorten from 140 to 100 (this would increase dataset)
- STEP: shorten from 10 to 4 (increasing dataset)
- DROPOUT: from 0.5 to 0.6 (knock out more neurons)

Results still show little improvement. Sample results are captured in `ebert_result_5_epoch10.txt`

Another attempt was to remove early-stopping to see if the solution was hitting a local minimums. Results still show no signs of improvement after 50 epochs.

4.2 Sentiment analysis

To combat model over-fitting the data, following hyper-parameter values were modified to see if the results would improve:

- MAX FEATURES: reduce from 2000 to 1000 (reduce features)
- EMBED DIM: reduce from 64 to 32 (reduce model dimension)
- LSTM OUT: reduce from 96 to 48 (reduce model dimension)
- DROPOUT: increase from 0.5 to 0.6 (knock out more neurons)

Results still show little improvement. Sample results are in `ebert_sa_results.txt`

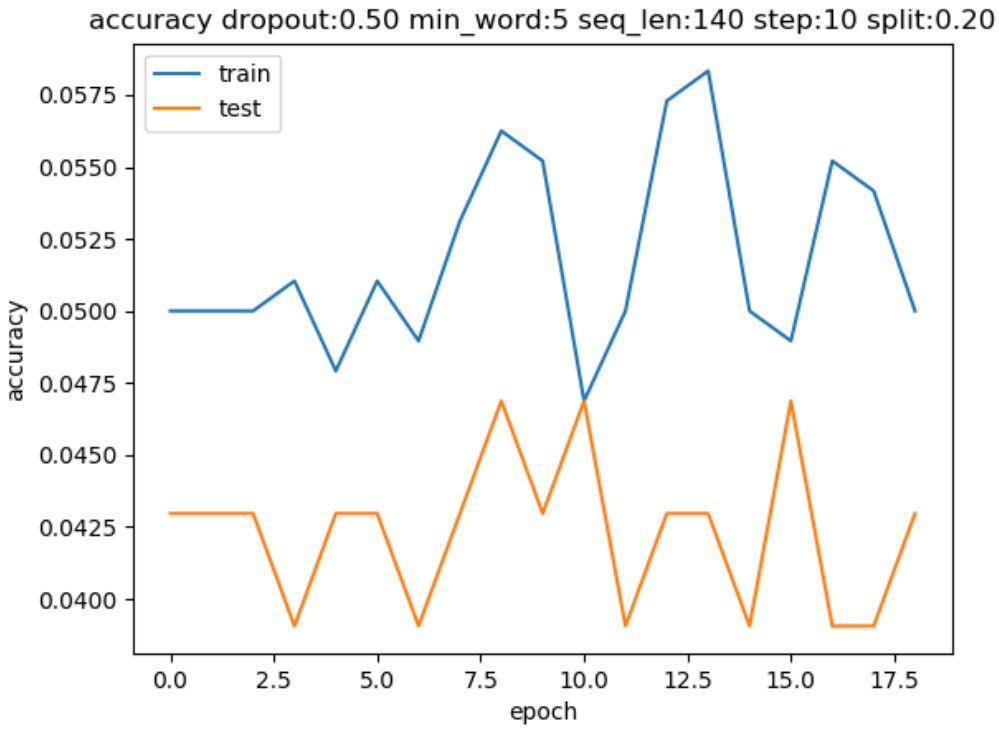

Figure 4: Word generation train/validation accuracy

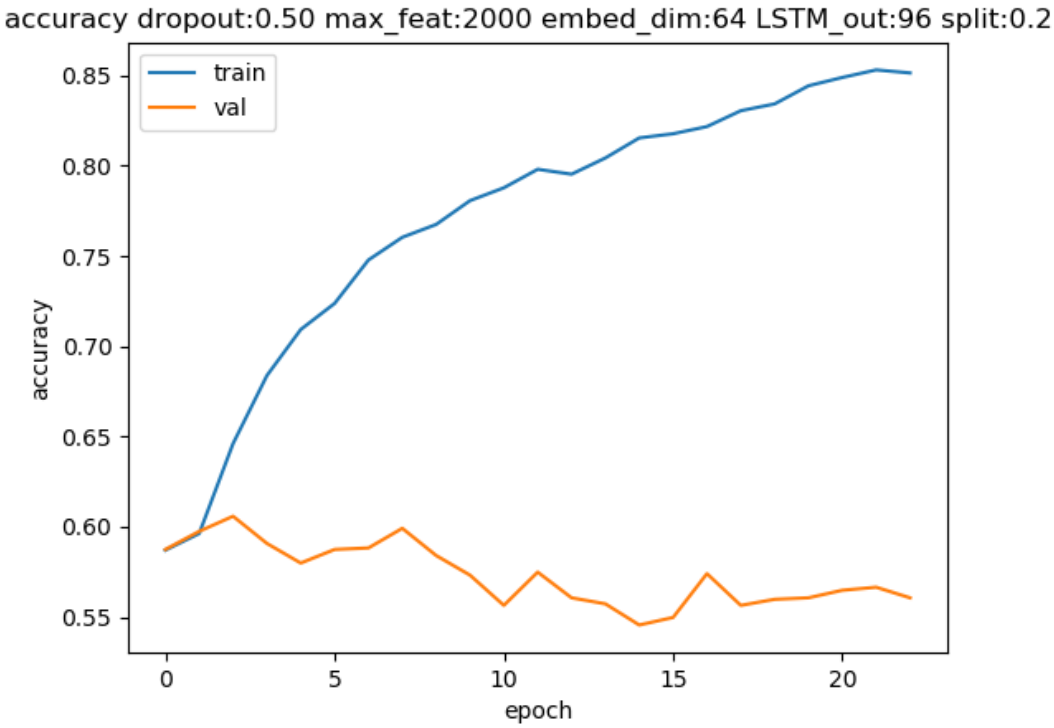

Figure 5: Sentiment analysis train/validation accuracy

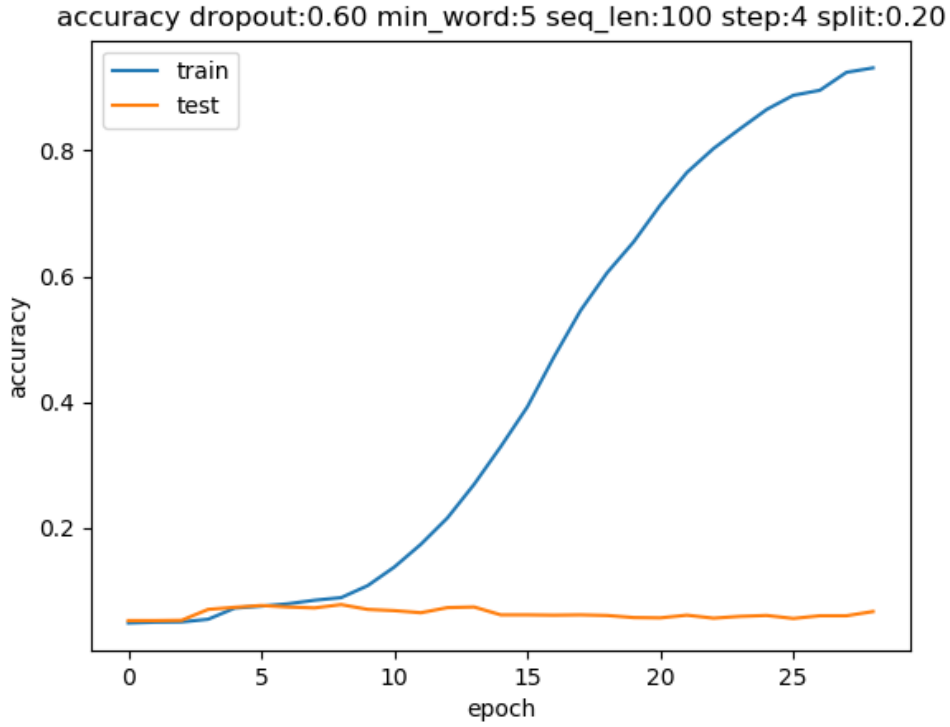

Figure 6: Word generation train/validation accuracy with updated hyper parameters

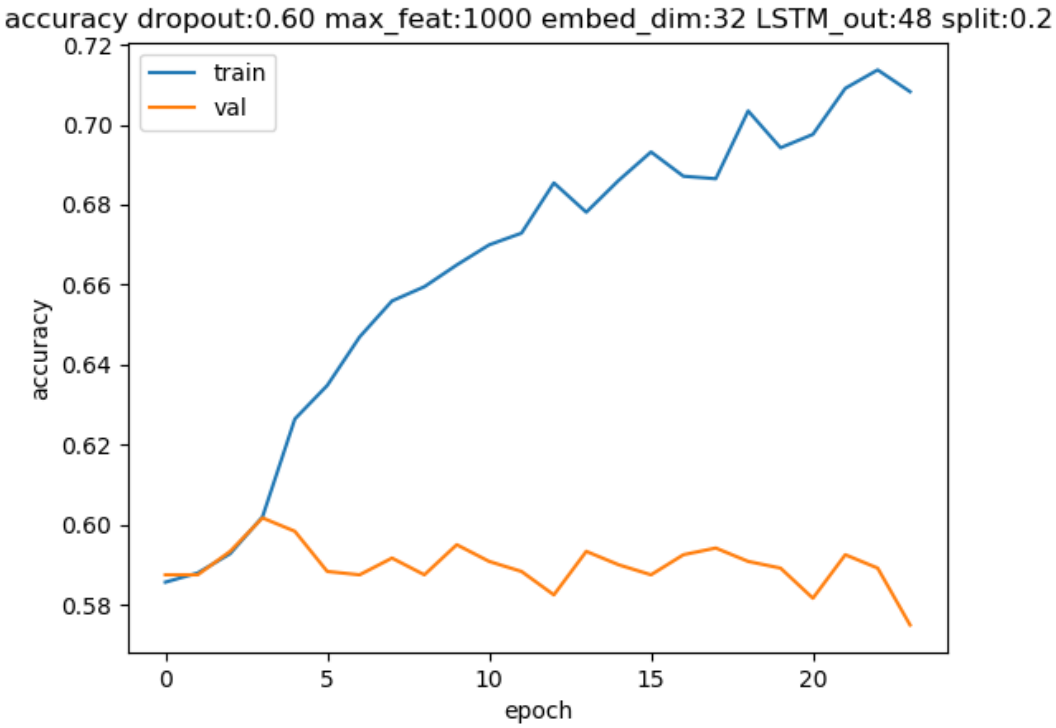

Figure 7: Sentiment analysis train/validation accuracy with updated hyper parameters

4.3 Combined output

Sampling the output data, it was noticed that due to truncating of the vocabulary within the training dataset, the word generation model could not give a comparable prediction of the words in the test set.

Also in sentiment analysis, the training dataset contains a lot of close-to neutral ratings (ratings between 2.5 and 3 stars within Roger Ebert's reviews account for 42.86 percent of the total reviews) which might cause the model to predict in accurately.

Sample input and output is shown below:

Input to word generation:

"the crimson rivers 2001 cast jean reno as pierre niemans vincent cassel as max kerkerian nadia fares as fanny ferreira laurent avare as remy caillois written by mathieu kassovitz jeanchristophe grange based on the novel by grange crime foreign mystery thriller rated r for violence and grisly images and language 105 minutes"

Output from word generation:

"marius be benji radiation has when plays in damned rated biopic know disorder marty mercilessly always inconsequential witnessed lines in first the mother story actors believe end sword from fit called hold here prosecutor could between fish with"

Combined input and output of word generation is served as the input to sentiment analysis. The sentiment prediction was made on both the actual Ebert's review text and on the generated text from WG model. Corresponding output is:

/reviews/the-crimson-rivers-2001 [[0.7755038 0.22449616]] Predicted with Ebert's original text: thumbs-down Ebert Actual: thumbs-up (3.5 stars) [[0.33416703 0.665833]] Predicted with WG generated text: thumbs-up Ebert Actual: thumbs-up (3.5 stars)

4.4 Conclusion

Even though the models did not accurately predict the words and subsequent sentiments, the methodology appears to be correct as the training data accuracy was improving as more iterations were run.

Several ideas could be attempted as future work to see if the accuracy could improve:

1. Include all of English vocabulary as input. Group similar words using K-means clustering to draw similar sentiments.
2. Consider writing a LSTM model that generates both the words, as well as the sentiment rating in the output. This could be achieved by fixing the number of words generated in a given input, and then at the end of the word generation the output is a binary assessment of the sentiment given the words generated. This could reduce compounding errors generated in two stages.
3. Reduce number of relatively neutral ratings in the training dataset. This would make avoid the model being overwhelmed by neutral vocabulary and ratings.

The above might imply requiring more CPU (e.g. increasing vocabulary size) and memory to run the training model.

Acknowledgments

I would like to acknowledge the instructors and mentors from CS221 Spring 2019 class for being patient in answering all questions and perform grading tirelessly.

I would also like to thank my wife Ivy for being patience with me throughout the last 2 years as I embarked on this journey to complete a Graduate Certificate program at Stanford while working full-time.

References

[1] Hochreiter S., Schmidhuber A. *Long short-term memory - Neural computation*. 1997.

[2] Williams, Ronald J., Hinton Geoffrey E., Rumelhart David E. *Learning representations by back-propagating errors* Nature. 323 (6088): 533–536. doi:10.1038/323533a0. (October 1986)

[3] Wang, Y., Huang, M., Zhao, L. *Attention-based LSTM for aspect-level sentiment classification.* In Proceedings of the 2016 conference on empirical methods in natural language processing (pp. 606-615). (2016)

[4] Enrique A. *Word-level LSTM text generator. Creating automatic song lyrics with Neural Networks.* https://medium.com/coinmonks/word-level-lstm-text-generator-creating-automatic-song-lyrics-with-neural-networks-b8a1617104fb, 2018.

[5] Peter Nagy. *Vehicle-Classification Algorithm for Single-Loop Detectors Using Neural Networks* https://www.kaggle.com/ngyptr/lstm-sentiment-analysis-keras, 2017.

[6] keras-team. *Keras examples* https://github.com/keras-team/keras/blob/master/examples/lstm_text_generation.py, 2019.