Using Attention on Movie Sentiment Classification

James Li
Stanford University
dawwctor@stanford.edu

Sam Kwong
Stanford University
samkwong@stanford.edu

Abstract

Sentiment classification is a common process relevant to many fields, and offers a convenient way to analyze how different systems learn to recognize important aspects of language. Here, we implement a platform to predict movie review sentiment as positive, negative, or neutral using a variety of basic and more advanced RNN models. We utilized various unidirectional RNN models, one with regular RNN cells, one with LSTM cells, and one with LSTM cells and single-context attention. We trained these models on cleaned review texts to output sentiment labels. Surprisingly, not only did we find that our simple LSTM model outperformed all other more advanced models, we also discovered that the performance of all of the models were roughly the same otherwise.

1 Introduction

We are interested in the problem of movie review sentiment classification, and even though this problem is fairly well-explored, we wanted to know whether an attention model would significantly increase performance. Our hypothesis is that attention would allow a model to focus especially on specific words that were important to the final judgment of sentiment. Even though attention is normally used in sequence-to-sequence problems, we modified a standard attention model to only utilize a single context vector to attend over the word embeddings of the movie review text.

Thus, our overall goal is to implement a platform that can predict if the sentiment of a given movie review is positive, neutral, or negative. Usually, a single review, which contains a body of text written by a critic or moviegoer, for a particular movie is to be paired with a single rating, a number from 1 to 10, for that particular movie. We classify the ratings of 1-4 out of 10 as a negative review, 5-6 out of 10 as a neutral review, and 7-10 out of 10 as a positive review. What we hope to accomplish is to feed our platform an input of a review and have it output the label of that review (positive, neutral, or negative). We use an RNN to train on data comprised of movie reviews and their corresponding rating labels for their movies.

The inputs of our models are lists of sequences of words extracted from movie reviews. We turn these sequences of words into sequences of trainable word embeddings, which we then feed into our RNN models one word at a time. These RNN models then output a single predicted sentiment label stating "pos", for positive, "neg", for negative, or "neut" for neutral.

2 Related work

We were inspired by the "Hierarchical Attention Networks for Document Classification" paper [2], as it also dealt with using attention in a non-sequence-to-sequence context for sentiment analysis. This paper proposed a hierarchical attention network for document classification, in which there are two layers of attention mechanisms applied at the word level and the sentence level [2]. This enabled their model to attend to the content of the document at different levels, so that even if there were important words, if they are placed in unimportant sentences, they should not contribute much meaning to the final judgment. Words do not have meaning in a vacuum, as they are determined by their context, so even if a particular word tends to be important, if it has no relation to the overall classification, then the model should ideally pay no attention to either the sentence or the word. Overall, this paper showed that their hierarchical attention model was successful on a variety of datasets, which is why we were inspired to implement a slightly modified version of this model with only one layer.

3 Dataset

Our dataset consists of a collection of IMDB movies containing multiple review per movie. These reviews included subsections of review summaries, both plain-text and tagged part-of-speech, dates, authors, locations of reviewers, helpfulness of reviews, and ratings. The dataset contains a total of $\sim$45,000 movies and $\sim$1.8 million reviews. These reviews are further split into roughly 518,000 positive reviews, 644,000 negative reviews, and 185,000 neutral reviews. The data we extracted to use to train our model is comprised of the plain-text of movie reviews and the ratings of the respective movie. We bucketed these numerical ratings as described above to be categorized instead as "neg", "neut", or "pos" for our labels.

Our preprocessing reads in the IMDB data, shuffles and splits it accordingly into training, dev, and test sets, cleans each review so that all non-alphanumeric characters were removed, and adds start and end tokens to each review. While we considered leaving punctuation in the reviews, as knowing when sentences started and stopped could help with extracting particular important topics, the fact that many of the reviews were written with poor grammar means that punctuation would not be very useful for distinguishing between sentences.

Cleaned Review Example: "Please take the time to do absolutely anything besides watchingthis film I try to think of something on par with the waste oftime I felt this was and I can t Actually I feel as if I amowed an apology for th"

Rating Example: "neg"

As we had a fairly large number of movies and reviews, we decided to perform a 90/5/5 training/dev/test split by movies to obtain three separate lists of movies. Once these lists of movies were obtained, we then expanded each movie into the full amount of reviews contained within each file to obtain three separate training, dev, and test lists of reviews. This is to prevent reviews from different movies being present across multiple sets, and we feel that it is more meaningful for a model to make sentiment judgments on reviews for which it has never seen reviews from the same movie before. This could perhaps show that our model is truly learning what makes a review positive, neutral, or negative, instead of relying on meta-information like the title or release date.

4 Methods

4.1 Baseline

Our baseline model consists of a standard unidirectional Recurrent Neural Network to train on data comprised of movie reviews and their corresponding rating labels for their movies. Our input is a list of word tokens and embeddings making up the review, while the output of our model is a softmax probability vector of which sentiment class the review belongs to. For evaluation, we use cross entropy loss between the model's output probability vector and the one-hot vector of our labels. We also include an accuracy metric by taking an argmax of the probability vector and seeing whether the model's predicted sentiment matched the true sentiment.

4.2 LSTM

For our LSTM model, we also used a standard unidirectional Recurrent Neural Network to train on data, but the main differences between this and the baseline model was that instead of using regular RNN cells, we used LSTM cells. The equations for an LSTM cell are as follows:

$$
\begin{aligned}
f_t &= \sigma_g(W_f s_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i s_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o s_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c s_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

where s_t is the word embedding of the word at time-step t. Beyond this, no other changes were made.

4.3 Attention

For our attention model, we once again used a unidirectional RNN with LSTM cells, but we added an additional attention-like component to our model. Normally, with attention, we have separate context and query sequences of vectors, so that the context can attend over the the query values to see which words were important. However, for our model, we use a single trainable context vector, which somewhat represents the context "what words are important in this review for sentiment?" By taking the dot product of this single context vector over the words in the review text, we produce a sequence of weights by which we weigh all of the hidden state contributions before actually applying the final softmax layer. The equations of our single-context attention component are as follows:

$$
\begin{aligned}
u_i &= \tanh(W_s h_i + b_s) \\
\alpha_i &= \frac{\exp(u_i^\top u_s)}{\sum_i \exp(u_i^\top u_s)} \\
v &= \sum_i \alpha_i h_i
\end{aligned}
$$

where h_i is the hidden state output from the ith time step of the LSTM RNN. Pictured in Figure 1 is a general overview of this entire model.

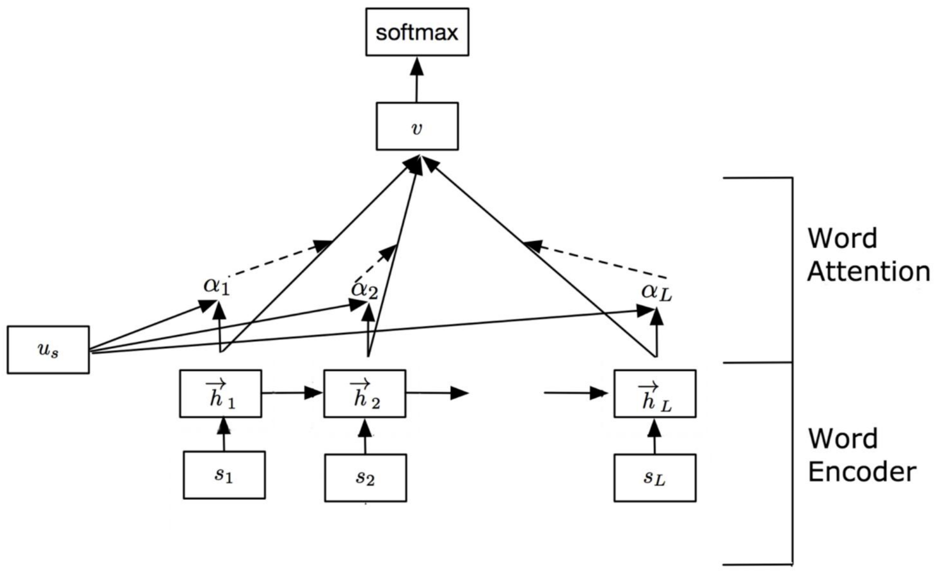

Figure 1: Single-Context Attention

This is a link to the GitHub repository containing our code: https://github.com/The-Dawwctor/cs230.

5 Experiments

5.1 Results

All of our models were trained for 5 epochs with a word embedding size of 50, as we found that word embedding size did not change our model performance significantly. In addition, we only trained for 5 epochs because our models converged quickly and started overfitting on the training set rather early, probably because we had a large training set compared to our dev set. For all models, we used cross-entropy loss and measured accuracy using the maximum prediction probabilities.

From performing several hyperparameter searches, we consistently found that regardless of model, we achieved best performance by using a learning rate of 0.001 for minibatch gradient descent, a dropout rate of 0.2, and a hidden state size of 50. For minibatch sizes, we had 1400 for our baseline model, 800 for our LSTM model, and 512 for our attention model. We chose these sizes by gradually decrease the minibatch size until we longer got Out-Of-Memory errors on the AWS machines.

Model	Test Accuracy	Loss
Baseline	0.6876	0.6963
LSTM	**0.7016**	**0.6675**
Attention (25 Units)	0.6800	0.7056
Attention (50 Units)	0.6824	0.7087
Attention (75 Units)	0.6658	0.7228

5.2 Discussion and Analysis

Some results we found surprising were how our attention model performed worse than both our basic LSTM model and our baseline model. In addition, even with extensive hyperparameter testing, most of the models had roughly the same performance. We think this might be due to he fact that we did not perform word stemming and case normalization, meaning that several words which might have had similar meanings and purposes ended up being treated like completely separate terms, increasing model complexity and the effects of overfitting.

Looking at our models, we found that all of our models converged quickly before overfitting severely, so additional regularization beyond just dropout might have helped. In addition, our attention model may not have performed well due to being an overly complex model while not training for long enough. Moreover, by only using unidirectional RNNs instead of bidirectional RNNs, our attention model might not have helped as much as the LSTM model did, as we would no longer be able to attend over the different words in the review as accurately.

In terms of our data, we reasoned that certain movie genres could have different types of reviews than others, making it a bad idea to neglect genre and explaining our less than stellar performance. Moreover, some reviews were inconsistent with the given reviews, possibly confounding our models, as some normally positive words would be associated with very negative reviews and vice versa. Finally, we could have preprocessed the data more to normalize the types of words used in word embeddings, greatly reducing dimensionality and also making word meanings more consistent across movie reviews.

6 Conclusion

Overall, we tested several different unidirectional RNN models and found dev and test performance to be mostly the same throughout all of the models. However, within these similar performances, we found the basic LSTM model to perform the best, even compared to our advanced attention model. We think that this model performed better than the rest because it was able to preserve long-term dependencies important for analyzing sentiment in a movie review without the additional model complexities and overfitting incurred by the attention model, possibly explaining the attention model's relatively poor performance.

7 Future Work

In the future, we would like to train these models again with bidirectional RNNs, as sentiment could depend on all words throughout the sentence, not just the beginning. In addition, we could reintroduce punctuation into the dataset to section reviews into sentences, so that we could more accurately implement a hierarchical attention model [2].

8 Contributions

Sam Kwong wrote the portions of the final poster and paper relating to the problem statements, datasets, and baseline models. Sam Kwong also coded some of the data parsing code for preprocessing the plain-text reviews, which were stored in an HTML format. James Li wrote the portions of the final poster and paper relating to the advanced models, the results, and the data analysis. James Li coded the majority of the data extraction, splitting, and shuffling parts of the data pipeline so that we could feed in our IMDB movie review database to our various RNN model. Finally, James Li modified the provided CS 230 code base to implement the LSTM and attention-based models in a sentiment classification context instead of a sequence-to-sequence context.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yang, Zichao, et al. "Hierarchical Attention Networks for Document Classification." Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016, doi:10.18653/v1/n16-1174.