Text Classification of the IMDb Movie Review Dataset

Yiwen Wang, Yue Zhai, Yuhan Liu
yiwenwan@andrew.cmu.edu, yuezhai@andrew.cmu.edu, yuhanliu@andrew.cmu.edu

1 Introduction

An interesting movie not only entertains the audience, but also inspires them to think and comment on it. Therefore, most people usually check on movie reviews before they decide which movie to watch. While humans may be good at extracting crucial sentimental information from long, complex text data, it is not as easy for machines to do so. This makes the IMDb reviews a good dataset for sentiment analysis since all the reviews have scores labeled.

It is a natural language processing (NLP) problem where ideally, the underlying sentiment would be understood from analyzing the review text and translated into a correct polarity label. The input data should be text and output can be negative/positive. Accuracy is determined by comparing the predicted label and actual label.

NLP actually is not a new science. The recent advancement in this area is caused by increasing interest in human-to-machine communications, plus the development of powerful computing and enhanced algorithms, especially deep neural networks. Besides sentiment analysis, NLP can be used in diverse applications, such as machine translation, automatic question answering, and automatic document summarizing. Given the availability of big amount of data, numerous efficient algorithms are needed to give better results. The evidence of working transfer learning also adds to the prospects of NLP [4, 16].

In this project, we hope to build a neural network combining existing techniques, and achieve better results than classical machine learning models.

2 Data

The data of this project was collected by Stanford researchers and published in an APL 2011 paper [9]. It contains 50,000 reviews split evenly into a train set and a test set. And for each set, the data is further evenly split into 12.5k positive and 12.5k negative reviews. A score <= 4 out of 10 is classified as a negative review and a score >= 7 out of 10 is classified as a positive review. Thus the neutral ratings like 5 or 6 are not included in the train/test datasets. Additionally, another 50,000 unlabeled movie reviews are included for unsupervised learning. Within the pos/neg folders, reviews are stored in text files named [id]_[rating].txt, where [id] is a unique review id and [rating] is the rating for that movie on a 1-10 scale. The txt file contains parsed text of the corresponding movie review. In order to predict a polarity label based on the movie review text, a clear pipeline of text classification will be proposed and tested including pre-processing, word embedding and neural networks.

3 Background

Below is a summary of findings before we moved on to language models, i.e., word embeddings + neural network.

3.1 Pre-processing

In our baseline model, punctuation and stop words provided by nltk [2] are removed during pre-processing. We experimented with different kinds of lemmatization and stemming, no improvement in performance was observed.

3.2 Baseline model

3.2.1 Vectorization

We tried three different vectorizers provided by scikit-learn [12] for our baseline model. CountVectorizer (bag-of-words) converts the text to sparse matrix based on token counts. HashingVectorizer converts the token counts into hashed features.

TfidfVectorizer uses the bag-of-words model to convert text to a count matrix, and transforms it into tf-idf representations. Tf-idf stands for term-frequency and inverse document-frequency. The formula of tf-idf is listed as below:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D) \tag{1}$$

Where t denotes term, d denotes document, D denotes the set of all documents. Also,

$$\text{tf}(t, d) = \text{count of term in the document} \tag{1a}$$

$$\text{df}(t, D) = \text{count of documents that include the term} \tag{1b}$$

$$\text{idf}(t, D) = \log\left[\frac{n}{\text{df}(t, D)}\right] + 1 \tag{1c}$$

In our baseline model, TfidfVectorizer outperforms the other two since the transform procedure helps to scale down the influence of some frequently occurring yet less informative terms, and assigns larger weights to unique terms.

3.2.2 Classifier

For the final classifying step, support vector machine (SVM) and logistic regression, with stochastic gradient learning, were tested. SVM performed slightly better.

Having a large vocabulary set, the matrix we obtain from the vectorizing step is remarkably sparse. This leads to the failure of some common classifiers such as naive Bayes and random forest. In the case of random forest, it is unrealistically expensive to either grow trees that are deep enough or include a large number of trees. As for naive Bayes, when it comes to logarithm calculation, the large vocabulary set makes the process time consuming. And by including log probabilities of all possible tokens, a significant amount of noise is brought into calculation.

Whereas SVM and logistic regression do not have to spend extra time to deal with zero-valued features. Therefore, they are often applied to classification tasks with extremely sparse data.

4 Related work

To build a language model, the choice of text representation and that of the subsequent neural network structure are two important areas to be studied. The aim of text representation is to translate text into vectors as valid inputs. A method to learn text representation by unsupervised learning, usually referred to as neural embeddings, play an important role in state-of-the-art language models. Two important word embedding techniques which inspired our model to analyze IMDb dataset are Word2Vec and fastText.

In terms of neural network structure, although some experiments were done to test convolutional neural networks, the results were disappointing. Therefore, this report will only focus on recurrent neural networks and the attention mechanism. More details are elaborated below.

4.1 Word embedding

4.1.1 Word2Vec

The Tomas Mikolov group introduced the Word2Vec technique [10] that can transform words into unique vectors typically of several hundred dimensions by a two-layer neural net. We used the continuous bag-of-words variation, meaning that the model learns embeddings by trying to predict the current word from the surrounding words, while the skip-gram variation tries to predict the surrounding words based on the current word. Since the initial publication, experiments have shown that Word2Vec is able to cluster similar words together. Interestingly enough, it also learns the relationship between different clusters to some extent. The capability to cluster similar words makes Word2Vec a good choice for sentiment analysis.

4.1.2 fastText

As an extension of the skip-gram model [10, 11], fastText [6] treats words as character n-grams instead. The learned embedding of a word is the the sum of vectors representing the character n-grams. Theoretically, by sharing parameters across words, fastText should learn more proper embeddings for rarer or even unseen words compared with Wor2Vec.

4.2 Recurrent neural network

Unlike basic feed-forward neural networks, a recurrent neural network (RNN) remembers information learnt from previous input while generating output. It can take one or more input vectors and produce one or more output vectors which are influenced not only by weights applied on inputs, but also by a hidden state vector representing the context based on prior input and output.

Beyond the simplest recurrent unit, two popular types of RNN are long short-term memory (LSTM) and gated recurrent units (GRU). Additionally, it is possible to force an RNN unit to process the sequence backward as well as forward, making it bidirectional. Such a layer will be denoted by adding a prefix "Bi-".

4.2.1 LSTM

LSTM was developed by Hochreiter & Schmidhuber in 1997 [5]. The internal structure of LSTM includes a memory cell and three gates which are forget gate, input gate and output gate. Calculated separately, the forget gate and input gate decide how much information from input and previous hidden state goes into the current memory. It is then combined with the output gate to calculate the current hidden state output.

4.2.2 GRU

Similar to LSTM, GRU [3] uses gates to transfer information with long term memory. It only has two gates in the internal structure: reset gate and update gate, making it less complicated than LSTM. Thus in principle, it should be faster. The biggest difference between GRU and LSTM, as pointed out by the authors [3], is that GRU does not have an output gate. The current memory is then passed forward completely.

4.3 Attention

In 2014, Bahdanau et al. [1] proposed the idea of calculating a context vector by taking a weighted sum of all hidden states of the input sequence. However, their weights are calculated by aligning the input hidden states and the decoder output, thus not applicable to other tasks.

A more popular attention mechanism is proposed later by Luong et al. [8], in which the context vector is calculated by aligning an input hidden state with other hidden states in the same sequence. The global attention looks at the whole sequence, while the local attention only looks at certain time steps around the current one. The mechanism employed in this project will be explained later in section 5.3.3.

4.4 Transformer

RNNs suffer severely from gradient explosion/vanishing when building deeper networks, which is also observed by us. Therefore, the state-of-the-art language models, Transformers, focus more on exploiting the attention mechanism. Transformers are a group of deep encoders based on a hierarchical attention network [15]. A good example is BERT [4], which includes layers of fully connected Transformer blocks. Another famous one is XLNet [16], which is able to outperform BERT on various tasks by integrating a segment recurrence mechanism.

5 Methods

In this section, we describe step by step how we built our language models.

5.1 Pre-processing

For the language models, during pre-processing, we converted all text into lower cases and removed punctuation.

5.2 Word embedding

We trained both kinds of word embeddings, Word2Vec and fastText, on the text of the train and test sets using gensim [13]. As for embedding dimension, we trained word embeddings with length of 100 and 300 units separately, following the convention of many famous pre-trained models. The word embeddings were then integrated into a matrix to be loaded later by the embedding layer in neural networks.

5.3 Neural network

For all equations below, x_t is the current input, and h_t is the hidden state at time step t generated by the current layer. And for the sake of simplicity, $\cdot$ stands for elementwise product, g_f stands for the output of the forget gate, etc.

5.3.1 LSTM

There are three gates controlling what information will pass through in LSTM: input, forget and output.

$$g_f = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2a}$$
$$g_i = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2b}$$
$$g_o = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2c}$$

And the three equations to update the cell state c_t and hidden state are:

$$c_t = g_f \cdot c_{t-1} + g_i \cdot \sigma(W x_t + U h_{t-1} + b) \tag{2d}$$
$$h_t = g_o \cdot \text{tanh}(s_t) \tag{2e}$$

5.3.2 GRU

There are two gates controlling what information will pass through in GRU: reset and update.

$$g_r = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3a}$$
$$g_u = \sigma(W_u x_t + U_u h_{t-1} + b_u) \tag{3b}$$

The new hidden state

$$h_t = (1 - g_u) \cdot h_{t-1} + g_u \cdot \widetilde{h}_t \tag{3c}$$

$\widetilde{h}_t$, referred to as candidate activation [3], is computed by

$$\widetilde{h}_t = \text{tanh}(W x_t + U(g_r \cdot h_{t-1})) \tag{3d}$$

5.3.3 Attention

In this project, the name "Attention" refers to the scaled multiplicative attention mechanism [8] proposed by Luong et al. More specifically, we used general global attention with a bias term. The behavior of the attention layer is iterated below following the query-key-value notation. Here, x_t is the input hidden state at time step t provided by the previous layer.

$$q_t = W_a^T x_t,\ k_t = x_t,\ v_t = x_t \tag{4a}$$

The scores are calculated as

$$s_{tt'} = q_t^T k_{t'} + b_a,\ s_t = [s_{t0}, s_{t1}, ..., s_{tt'}, ...]^T \tag{4b}$$

The attention vector

$$a_{tt'} = \frac{exp(s_{tt'})}{\sum_i exp(s_{ti}) + \epsilon},\ a_t = [a_{t0}, a_{t1}, ..., a_{tt'}, ...]^T \tag{4c}$$

Or simply,

$$a_t = softmax(s_t) \tag{4d}$$

Finally, the hidden states generated by the this layer

$$h_t = a_t^T v_t \tag{4e}$$

In this implementation, attention activation was also available to be applied to s_t. However, due to limited time, we did not experiment with this extra modification in our project.

5.3.4 Dimensionality of hidden states

Due to the strict memory limit imposed by the cloud service we used, we had to arbitrarily reduce the number of parameters when building more complicated models. Therefore, the dimensionality of hidden states varies between different models, which will be reflected in the submitted scripts. Generally, in a given model the dimensionality of hidden states decreases after each layer.

5.3.5 Overall structure

All the language models follow the overall structure of: embedding layer + RNN layers (or with attention layers) + fully connected dense layers.

Some of the best models are illustrated as below:

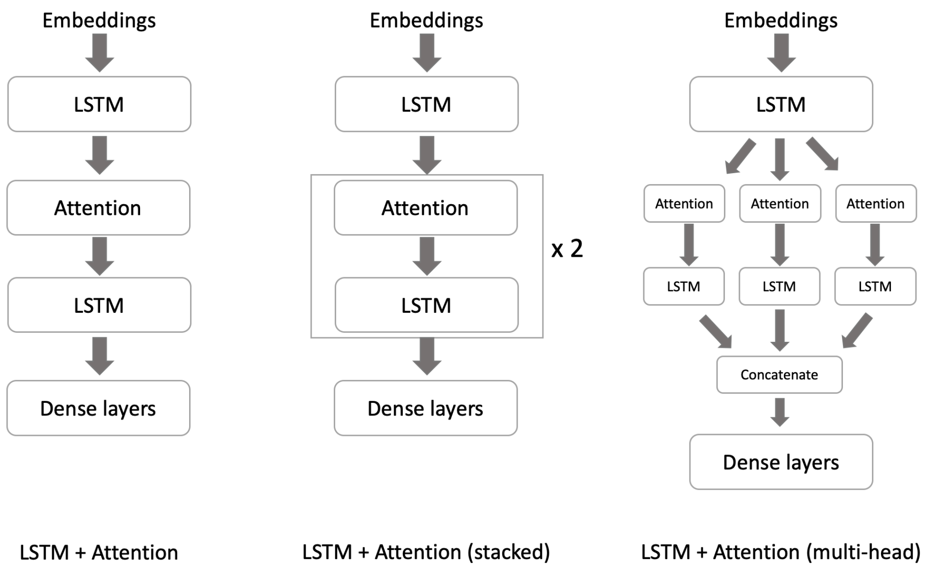

Figure 1: Neural network structure

5.4 Optimization

5.4.1 Loss function

The loss function we used during training, i.e. the objective function, is the standard binary cross-entropy loss. For each batch with a size of n, we have

$$l = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)) \tag{5}$$

Where $\hat{y}_i$ is the predicted probability of the ith review in this batch being positive.

5.4.2 Optimizer

The optimizer we used is Adam, which stands for adaptive moment estimation [7]. The values of all hyperparameters are set to default: learning rate $\alpha = 0.001$, decay rates $\beta_1 = 0.9,\ \beta_2 = 0.999$, and $\epsilon = 10^{-7}$.

During backpropagation, for any parameter vector θ, while the convergence criterion is not reached, the following steps are taken to update θ. Here, for all variables, the subscript $t-1$ denotes corresponding values from the previous time step, whereas t denotes the current time step.

Also, g is the gradient, m is the first moment vector, and v is the second moment vector. For simplicity, g_t^2 indicates the elementwise square of g_t.

$$g_t = \nabla_\theta\, l(\theta_{t-1}) \tag{6a}$$
$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t \tag{6b}$$
$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^2 \tag{6c}$$
$$\hat{m}_t = m_t/(1-\beta_1^t) \tag{6d}$$
$$\hat{v}_t = v_t/(1-\beta_2^t) \tag{6e}$$
$$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon) \tag{6f}$$

Although the update rule is designed to be approximately invariant to rescaling of the gradient, the optimizer does not necessarily limit the gradient itself. If g_t is incredibly large, some of the terms will explode, and the algorithm will simple not be able to handle them.

During our experiments, gradient explosion was a problem when dealing with models with multiple layers. In order to tackle this, we added gradient clipping and regularization during the first 5 epochs of training for each model.

5.4.3 Learning curves

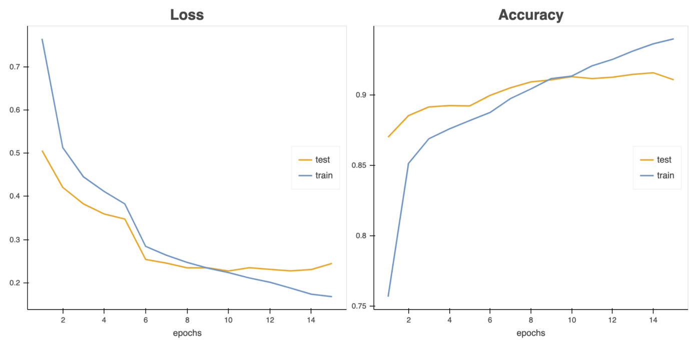

Figure 2: Learning curves of LSTM + Attention

The plots shown above demonstrate the loss and accuracy progression of our best model. For most models, approximately 10 epochs are needed to reach an optimal state.

6 Results

The following table lists test results from some of the models we have experimented with. As mentioned before, we used accuracy as the metric for evaluation. For example, 90 means 90 percent of the test reviews were labeled correctly by the model. Many models, complex or not, were unable to perform on the same level as our baseline model. Therefore, they are not listed.

The order of models is not necessarily an ascending order of performance or complexity, but rather reflects how we proceeded onward with the experiments, trying to test the ideas later discussed in section 7.2.

The column "Dimension" refers to the dimensionality of word embeddings.

Table 1: Test results

Model	Word embedding	Dimesion	Accuracy
Tfidf + SVM			89.28
GRU	fastText	100	89.86
LSTM	fastText	100	88.90
GRU	Word2vec	100	90.28
LSTM	Word2vec	100	90.72
LSTM (stacked)	Word2vec	100	90.60
Bi-LSTM	Word2vec	100	90.08
Bi-LSTM	Word2vec	300	90.44
LSTM + Attention	Word2vec	100	90.89
LSTM + Attention	Word2vec	300	91.56
Bi-LSTM + Attention	Word2vec	300	91.43
LSTM + Attention (stacked)	Word2vec	300	90.76
LSTM + Attention (multi-head)	Word2vec	300	91.30

For each language model, only one instance was implemented and tested. Therefore, the hyperparameters are not tuned.

Our best test results are around 91.5. Although lower than the state-of-the-art results which are usually 95+ [14, 16], we think we reached some good results, exceeding our baseline by more than 2 percent in test accuracy.

7 Discussion and analysis

7.1 Word embedding

Word2Vec proved to be better than fastText, possibly because the vocabulary size is still rather small. Summing the vectors representing character n-grams brought extra noise.

Although not listed above, we did experiment with some publicly available pre-trained word embeddings. The results were quite unsatisfactory. The main reason is that they fail to properly handle out-of-vocabulary words. This also suggests that word embeddings are, to a limited extent, task sensitive.

Another thing we did differently from other groups is that we did not filter out any words that are too rare or too common. This turns out to be a critical decision and explains why we were able to exceed the baseline with a model as simple as consisting of only one LSTM layer. Intuitively, for a small fraction of the movie reviews, different from twitter text, less common words tend to encode key information.

Also, for this task, we concluded that word vectors with more units perform better, which is not universally true. This could be attributed to the fact that enough text was available for unsupervised learning.

7.2 Neural network

7.2.1 Dimensionality of hidden states

In some early stage experiments, we discovered that the dimensionality of the hidden states generated by the RNN layer immediately after the embedding layer has some impact on the performance. Although for subsequent layers, the impact was less obvious. This suggests that dimensionality reduction should not be too drastic especially for earlier layers.

7.2.2 The deeper, the better?

Unfortunately, based on the results, stacking the same layers vertically will only hurt performance. The idea of building deep neural networks does not work for all tasks.

7.2.3 RNN

LSTM outperformed GRU in most experiments. Since most reviews contain multiple sequences, being able to control long and short term memory separately is important. Also, by introducing an output gate, it is possible that LSTM is able to learn to output small values when a certain kind of input is encountered.

Surprisingly, converting the RNN layers into bidirectional ones did not help with performance. However, this is likely due to the fact that the hyperparameters were not tuned.

7.2.4 Attention

Compared to recurrent neural networks, the attention mechanism is a rather novel idea. There are many variations, yet no popular deep learning framework seems to have a ready-to-use, tunable implementation of a single attention layer.

The inclusion of attention layers led to substantial increase in model performance, suggesting that an attention layer is able to some learn long-distance word correlations that LSTM fails to.

In hindsight, here are some ideas that could be explored when trying to build a relatively light-weight, yet acceptably effective language model compared to Transformers:
i. For long sequences like move reviews, local attention might outperform global attention.
ii. Concatenating x_t and h_t after an attention layer, instead of only returning a sequence of the context vectors.
iii. The add-and-normalize idea when building an attention layer proposed in the original Transformer [15].

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[9] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. pages 45–50, May 2010. `http://is.muni.cz/publication/884893/en`.

[14] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[16] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.