# Recurrent Neural Networks for Sentiment Analysis

**Joaquín Ruales**
Columbia University
jar2262@columbia.edu



## Abstract

We compare LSTM Recurrent Neural Networks to other algorithms for classifying the sentiment of movie reviews. We then repurpose RNN training to learning sentiment-aware word vectors useful for word retrieval and visualization.

## 1 Introduction

Recurrent Neural Networks (RNNs. Described in more detail in Section 3.1) have proven to be successful in many natural language tasks, including in learning language models that outperform n-grams (Mikolov, Karafiat et al., 2011), and in achieving the state-of-the-art in speech recognition (Hannun et al., 2014).

Furthermore, *Recursive* Neural Networks[1]—a network structure similar in spirit to Recurrent Neural Networks but that, unlike RNNs, uses a tree topology instead of a chain topology for its time-steps—has been successfully used for state-of-the-art binary sentiment classification after training on a sentiment treebank (Socher et al., 2013).

Motivated by these results, we use Recurrent Neural Networks for the natural language task of predicting the binary sentiment of movie reviews.

## 2 Problem Formulation and Dataset

Our focus is the binary sentiment classification of movie reviews. Given the text of a polar movie review, we predict whether it is a positive or negative review, where positive means the reviewer gave the movie a star rating of 7/10 or higher, and negative means a rating of 4/10 or lower.

Later, we tackle the problem of learning sentiment-aware vector representations of words, and of using these for visualization and word retrieval.

We use a subset of the Large Movie Review Dataset (Maas et al., 2011). The dataset consists of 50000 polar movie reviews, 25000 for training and 25000 for testing, from the IMDb internet movie database. Due to our limited computational resources, for each experiment we chose a word count limit and trained on reviews with at most that number of words. For testing we used 500 reviews with arbitrary length chosen at random from the larger test set.

## 3 Background

### 3.1 RNNs

Recurrent Neural Networks (RNNs) are a type of neural network that contains directed loops. These loops represent the propagation of activations to future inputs in a sequence. Once trained, instead of accepting a single vector input $x$ as a testing example, an RNN can accept a sequence of vector inputs $(x_1, \ldots, x_T)$ for arbitrary, variable values of $T$, where each $x_t$ has the same dimension. In our specific application, each $x_t$ is a vector representation of a word, and $(x_1, \ldots, x_T)$ is a sequence of words in a movie review. As shown in Figure 1, we can imagine "unrolling" an RNN so that each of these inputs $x_t$ has a copy of a network that shares the same weights as the other copies. For every looping edge in the network, we connect the edge to the corresponding node in $x_{t+1}$'s network, thus creating a chain, which breaks any loops and allows us to use the standard backpropagation techniques of feedforward neural networks.

### 3.2 LSTM Units

One of the known instabilities of the standard RNN is the vanishing gradient problem (Pascanu et al., 2012). The problem involves the quick decrease in the amount of activation passed into subsequent time steps. This limits how far into the

[1] We avoid referring to these as "RNNs" to avoid confusion

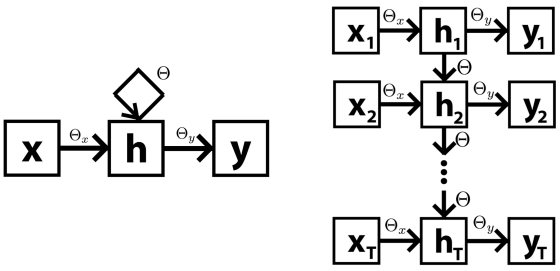


Figure 1: An RNN as a looping graph (left), and as a sequence of time steps (right).

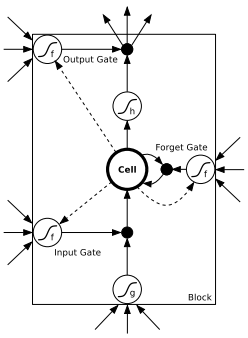


Figure 2: LSTM Unit (Graves, 2012)

past the network can remember. In order to alleviate this problem, Long Short Term Memory (LSTM) units were created to replace normal recurrent nodes (Hochreiter et al., 1997). A diagram of one of these units is displayed in Figure 2. These units introduce a variety of gates that regulate the propagation of activations along the network. This, in turn, allows a network to learn when to ignore a new input, when to remember a past hidden state, and when to emit a nonzero output.

## 4 Sentiment Classification Experiments

Below, we explain the procedure for each of our experiments for sentiment classification.

### 4.1 LSTM

We performed our LSTM experiments by extending the deeplearning.net LSTM sentiment classification tutorial code, which uses the Python Theano symbolic mathematical library.

The network takes as input the ordered sequence of $T$ words that make up a movie review, and it performs 4 steps to predict the movie's sentiment—projection, LSTM, mean pooling, and logistic regression.

1. Projection:

$$x_t = Rw_t$$

where $w_t$ is the $t$'th word of the current review represented as a "one-hot" 10000-dimensional vector unique to each word (any word not in the 9999 most common words in the data is treated as the same word, "UNK", to prevent sparseness), and $R$ is a weight matrix that projects each word into to a 128-dimensional space. Here $R$ is learned during training.

2. LSTM unit:

$$\begin{aligned}
\tilde{C}_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\
i_t &= logistic(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= logistic(W_f x_t + U_f h_{t-1} + b_f) \\
C_t &= i_t * \tilde{C}_t + f_t C_{t-1} \\
o_t &= logistic(W_o x_t + U_o h_{t-1} + b_o) \\
h_t &= o_t * tanh(C_t)
\end{aligned}$$

where $\tilde{C}_t$ is the output value of the recurrent unit as would be computed by a classical RNN, $i_t$ is the LSTM input gate, $f_t$ is the LSTM forget gate, $C_t$ is the value of the internal LSTM memory state created by blending the past state ($C_{t-1}$) and the proposed state ($\tilde{C}_t$), $o_t$ is the output gate, and $h_t$ is the final output of the LSTM unit after passing through the output gate. The symbol "$*$" represents element-wise multiplication. The parameters that are learned during training are the weight matrices $W_i, U_i, W_c, U_c, W_f, U_f, W_o, U_o$, and the bias vectors $b_i, b_c, b_f, b_o$.

3. Mean pooling:

$$\bar{h} = \frac{1}{T}\sum_{t=1}^{T} h_t$$

4. And finally, logistic regression:

$$y = softmax(W_y \cdot \bar{h} + b_y)$$

where weight matrix $W_y$ and bias vector $b_y$ are learned during training. $y$ is a 2D vector where the first entry is $p(negative|w_1 \ldots w_T)$, and the second entry is $p(positive|w_1 \ldots w_T)$. While we are using the softmax function, this formulation is equivalent to logistic regression.

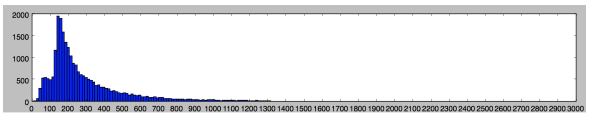

Figure 3: Word count histogram for training and testing data. Word counts ranged from 11 to 2820.

The network is trained using stochastic gradient descent with the AdaDelta update rule, which uses an adaptive per-dimension learning rate that does not require manual annealing.

Due to the prohibitively long time it takes to train this network on all of the reviews in our training set, for our experiments we trained this network using word count limits of 100, 150, and 300. We saw a rapid increase in running time as the limit increased since both number of examples and average example word count grows. Figure 3 shows the distribution of word counts for the full training and validation set for reference.

### 4.2 Variations on LSTM

#### 4.2.1 Regular RNN

For this experiment, we replaced all LSTM units in the above network with classical RNN units to see if the addition of LSTM brought an advantage to this specific problem.

#### 4.2.2 "Non-Forgetting" LSTM

For this experiment, we modified the LSTM by replacing the forget gate with a vector of all 1s, always preserving the previous state when incorporating a new example. This modification represents the LSTM unit as it was first described, before (Gers et al., 2000) introduced the forget gate.

#### 4.2.3 LSTM with Max Pooling

For this experiment, we used max pooling instead of mean pooling of our LSTM unit outputs, setting

$$\bar{h} = \max_{t=1...T} h_t$$

### 4.3 SVM

We trained a linear SVM on the data using Bag of Words features for each movie review. Figure 4 shows our training and validation errors for several values of the SVM $C$ parameter. The SVM with the lowest validation error obtained a test error of 0.198. We see from the training error plot that the SVM is able to achieve zero error on the training data, so the training examples are linearly separable in Bag-of-Words feature space when training

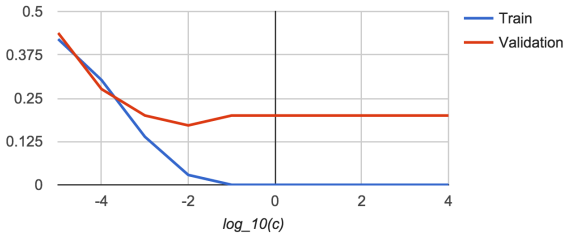


Figure 4: Training and validation error for Bag of Words Linear SVM with a variety of values for $c$

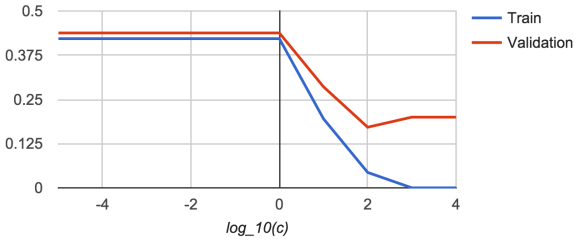


Figure 5: Training and validation error for Bag of Words RBF SVM with a variety of values for $c$

on examples of word count up to 100. However, this linear hyperplane is not optimal for the problem as a whole as evidenced of the $\sim 20\%$ testing error rate achieved. Furthermore, we trained another linear SVM on the set of all training examples of arbitrary word count. Additionally, we trained an RBF kernel SVM, and we show the training and validation curves in Figure 5.

### 4.4 Results and Analysis

Table 1 summarizes our classification experiment results.

Based on these results, we notice that using LSTM using was not a significant improvement to using regular RNN units, so the vanishing and exploding gradients problem might not be significant when using this network structure and these data. Similarly, using max pooling had a minimal impact on our results. However, not allowing LSTM units to forget past internal state did seem to have a large negative impact in our generalization.

We can see that the linear SVM using Bag of Words features obtained the best performance, both when only considering the experiments with a word limit of 100, and also when considering all of the experiments in Table 1, so our current RNN model seems to have been unsuccessful, although we have yet to see the performance when training on the entire training set. One disadvantage of our LSTM network is the large amount of param-

| Experiment | w limit | Error |
|---|---|---|
| Regular RNN | 100 | 0.248 |
| LSTM | 100 | 0.236 |
| LSTM | 150 | 0.197 |
| LSTM | 300 | **0.134** |
| LSTM | No limit | $\star$ |
| "Non-forgetting" LSTM | 100 | 0.328 |
| LSTM. Max Pooling | 100 | 0.238 |
| Bag of Words Linear SVM | 100 | 0.198 |
| Bag of Words Linear SVM | No limit | **0.114** |
| Bag of Words RBF SVM | 100 | 0.216 |

Table 1: Test errors for each of our sentiment classification experiments. "w limit" is the word limit per review on the training and validation data. Test data is the same for all experiments. A "$\star$" means that the training was taking a prohibitively long time to finish, so it was halted.

eters it has, which makes the training slow without a proper GPU and might cause serious overfitting. It would be beneficial to explore how this model compares to SVM as the number of examples increases up until training on all words (given enough processing power to train the network). It might also be worth trying an LSTM network with less weights.

## 5 Sentiment-Aware Word Vector Experiments

(Maas et al., 2011) describes an algorithmic framework for learning word vectors for sentiment analysis while also training a binary classifier. As part of a larger semi-supervised multi-task learning objective that also takes into account semantic similarity, the paper includes a supervised sentiment-related objective:

$$\max_{R,\psi,b_c} \sum_{k=1}^{|D|} \sum_{i=1}^{N_k} \log p(s_k|w_i; R, \psi, b_c),$$

where $R$ and $w_i$ are as in our LSTM network described above, $\psi$ is a weight vector, $b_c$ is a bias vector, $D$ is the set of all documents, and $s_k$ is the sentiment label (0 or 1) for document $k$ in the training data. The paper uses logistic regression to represent the classification likelihood based on a single word:

$$p(s = 1|w_i; R, \psi) = logistic(\psi^T R w_i + b_c).$$

The objective function finds the classifier that maximizes the likelihood of the training data so that documents are classified correctly, but since the $R$ matrix is part of the trained variables, the algorithm also learns the vector representations of words (the columns of $R$) that are most useful for the classification task.

### 5.1 Word Vectors

An RNN network like the one described in Section 4.1 includes a projection like the one in (Maas et al., 2011) described above, using a neural network as its likelihood function. Thus, its training also produces sentiment-aware word vectors.

These learned word vectors are not only useful to the algorithm for classification. It is also helpful for humans, since we can use an embedding technique to visualize the high-dimensional word vectors in 2D and "see what the algorithm is seeing." For our Word Vector experiments, we wanted to better understand the inner workings of the open-source movie sentiment classification network included as part of the Passage Python library, which uses a structure like the one in Section 4.1 but with GRU units instead of LSTM units and using only the last recurrent output instead of mean pooling over all of the outputs. We trained that network and then extracted the word vectors from the first layer weights. Figure 6 shows a 2D embedding—using T-SNE stochastic dimensionality reduction—of the word vectors for the 500 most common words. Based on this visualization, we can see that the algorithm generally groups highly polarized words together as expected. However, the algorithm somewhat unexpectedly groups "1", "3", and "4" with negative sentiment words, and "10" with positive sentiment words. This most likely means that some people include the star rating for the review as part of the review's text ("I give this movie 10 stars because...").

Additionally, we can use the sentiment-aware word vectors to define a sentiment similarity metric between words. Table 2 shows the five closest words to "lackluster," "melancholy," "ghastly," and "romantic" according to a cosine similarity metric on the extracted word vectors, compared to the results in (Maas et al., 2011).

## 6 Conclusion

We have compared the use of LSTM Recurrent Neural Networks to other classification methods

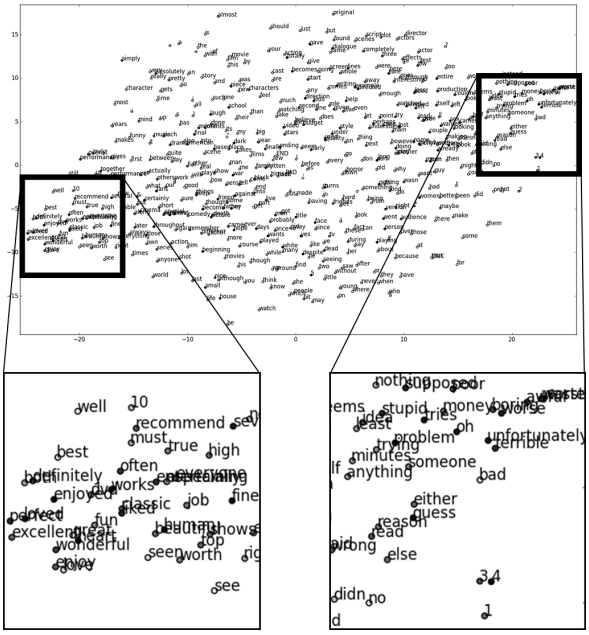

Figure 6: 2D t-SNE Embedding of our learned sentiment-aware word vectors for the 500 most common words in our data (top), and a detailed view of highly polarized regions (bottom). Lighter data points represent more common words. (Zoom in for a complete view.)

| | RNN model Sentiment only | (Maas et al., 2011) Sentiment + Semantic |
|---|---|---|
| **lackluster** | awful<br>insult<br>mess<br>waste<br>poorly | lame<br>laughable<br>unimaginative<br>uninspired<br>awful |
| **melancholy** | anticipation<br>feelings<br>realism<br>outline<br>whilst | bittersweet<br>heartbreaking<br>happiness<br>tenderness<br>compassionate |
| **ghastly** | defined<br>control<br>entertained<br>benjamin<br>investigates | embarrassingly<br>trite<br>laughably<br>atrocious<br>appalling |
| **romantic** | definitely<br>favorite<br>moving<br>excellent<br>superb | romance<br>love<br>sweet<br>beautiful<br>relationship |

Table 2: Retrieval of five closest words based on cosine similarity of learned word vectors. Based on these results, the RNN method seems to consider "lackluster" and "romantic" as polar indicators of negative and positive sentiment on a movie review, respectively. On the other hand, it does not seem to consider the words "melancholy" or "ghastly" as indicators of extreme positive or negative sentiment felt towards a movie.

for binary sentiment classification of movie reviews. Furthermore, we took advantage of a word projection layer in the first layer of an RNN to learn sentiment-aware word vectors, which we used for word retrieval and visualization.

Future experiments could include training the LSTM model on the entire training dataset, experimenting with other topologies and layer sizes, using a multi-task semi-supervised learning objective like the one described in (Maas et al., 2011), and experimenting with manifold-learning techniques for visualization of the word vectors.

## References


Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). *Learning to forget: Continual prediction with LSTM*. Neural computation, 12(10), 2451-2471.

Graves, Alex. *Supervised sequence labelling with recurrent neural networks*. Vol. 385. Springer, 2012.

A. Graves, A. Mohamed, and G. Hinton. *Speech recognition with deep recurrent neural networks*, ICASSP, 2013.

A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Ng, *Deep Speech: Scaling up end-to-end speech recognition*, in arXiv:1412.5567, 2014.

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). *Learning Word Vectors for Sentiment Analysis*. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)

T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, S. Khudanpur. *Recurrent neural network based language model*, In: Proceedings of Interspeech, 2010.

Pascanu, R. and Bengio, Y. (2012). *On the difficulty of training recurrent neural networks*. Technical Report arXiv:1211.5063, Universite de Montreal.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013b. *Recursive deep models for semantic compositionality over a sentiment treebank*. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1631–1642, Stroudsburg, PA, October. Association for Computational Linguistics.