NLP Practical II: Encoding Sentences with Neural Models

Anne Chel
MSc. AI
10727477
anne.chel@student.uva.nl

Mateo Jaramillo
MSc. Logic
12594520
mateo.jaramillo@student.uva.nl

Abstract

In this paper several sentiment classifiers are implemented[1] and analyzed. In addition to bag-of-words and LSTM approaches, a combined classifier is also tested for performance. We show the impact of sentence length and word-order for each classifier.

1 Introduction

Sentiment classification with neural models involves training a model on labelled data in the form of pairs containing sets of sentences and their corresponding classification. There are many existing models that produce varying levels of accuracy for sentiment classification. Our experiments and analysis are conducted on sentiment classification of movie reviews (Stanford Sentiment Treebank) as the individual reviews are generated by different types of users, and despite their variant lengths, grammatical structure, and vocabulary depth, they are paired with clear, often numeric, sentiment scores. One of the many challenges is to avoid overfitting, in which the model is too highly trained on specific examples and is thus inaccurate when it comes to generalized inputs during testing.

There are simpler methods than others, for example, the Bag-of-Words (BoW) method collects each token in a review as a vector and computes an overall score for the review based on the sum of all vectors. This method is easy as it treats each token as independent, but it sacrifices context, as the word order is no longer accounted for when the vectors are summed. Consider the review:

”I loved the movie, but hated the theater.”

In a BoW model, this review would generate the same sentiment prediction as ”I loved the theater, but hated the movie”, which clearly has an opposite sentiment.

To begin accounting for context and word order within a review, we begin to look at the review as a sequence with long term dependencies attached to individual words or phrases. This method uses a Long Short-Term Memory (LSTM) model (Greff *et al.*, 2016; Tai *et al.*, 2015). This method can still generate errors on which dependencies belong to certain words when a sentence contains multiple noun phrases nested within other syntactic structures. Representing the review as a parsed tree can help with disambiguation errors of this type as the review is treated as a composition of its syntactic parts rather than a sequence.

Contributions Our project investigates these different models and certain variations of them to determine which neural models produce the highest accuracy while both maintaining computational speed and minimizing overfitting. Our main areas of concern are comparing the differences of models which consider word order against those that do not. We experiment with the models, testing their performance and recording their accuracies on sentences of varying length. We compare the differences between LSTM models that use sequences compared to those that use parsed trees.

Our results conclude that adding context through word order and syntactic structure contributes significantly to the accuracy of a neural model. Through manipulation of the training set, our experiments conclude that sentences of greater length are more accurately classified by LSTM models, rather than BoW variant models. Furthermore, we present a constructed model that implements methods from both Deep CBOW models and LSTM, however we find that it provides no performance advantage over existing LSTM models.

[1]https://gist.github.com/annechel/8ab89cdaacd989e00045c9e5c6bbc516e

2 Background

Adaptive Moment Estimation and Stochastic Gradient Descent: In training neural networks we need it perform an iterative operation of making a prediction, calculating its error (loss), and then ”stepping away” from this error. Stochastic gradient descent (SGD) is this optimization method, which computes the derivative of the cost function after an iteration and updates the coefficients with respect to the learning rate, which we set as a hyperparameter for each model. Computing the derivative allows the model to use the negative direction of the slope to find a local minimum. Our models use the Adam (Adaptive Moment) (Kingma and Ba, 2015) optimizer which uses SGD to update individual learning rates for different parameter weights in the neural model.

Word Embeddings For our neural models to interpret the reviews, we represent them as vectors of real numbers. Initially, our words are indexed corresponding to the order in which they appear, generating a set of large 1-hot vectors. Through the iterations of the training process, the input words' indexes are converted to word embeddings of a size equivalent to our number of sentiment classes. From there, we can use the sequence of word embeddings to compute a sentiment prediction from our models.

BoW Our classic BoW approach, sums the word embeddings of a review, and additionally adds a bias before finally taking the argmax to generate a corresponding label for the input. Consider a review of length n, with 5 possible sentiment classes, and a bias vector b, we can represent the BoW model as $\sum^{n} w_i + b$:

$$\begin{array}{rcllllll} \mathrm{w}_1 & : & [\mathrm{s}_0 & \mathrm{s}_1 & \mathrm{s}_2 & \mathrm{s}_3 & \mathrm{s}_4] \\ \vdots & : & & & \vdots & & \\ \mathrm{w}_n & : & [\mathrm{s}_0 & \mathrm{s}_1 & \mathrm{s}_2 & \mathrm{s}_3 & \mathrm{s}_4] \\ \text{bias} & : & [\mathrm{b}_0 & \mathrm{b}_1 & \mathrm{b}_2 & \mathrm{b}_3 & \mathrm{b}_4] \\ \hline \text{sum} & : & [\mathrm{x}_0 & \mathrm{x}_1 & \mathrm{x}_2 & \mathrm{x}_3 & \mathrm{x}_4] \end{array}$$

with the sentiment prediction being the argmax of the sum.

There are variations of the classic BoW model such as the continuous BoW (CBOW) and the Deep CBOW. The CBOW works similarly to the BoW model, except the word embeddings can be of arbitrary size. This allows for a more fine-grained representation of each word. The word embeddings of a review are then converted to a matrix corresponding to the number of sentiment classes by matrix multiplication: $n \times a \odot n \times |S|$ which becomes an $n \times |S|$ matrix to which we can add our bias to and compute sentiment predictions as before. Likewise, the Deep CBOW model uses this method of reshaping the word embeddings multiple times while also using tanh activation functions.

LSTM While the BoW models take the word embeddings as independent of one another, the LSTM uses a sequential approach. The LSTM model uses a memory cell that it calls upon repeatedly. The memory cell is composed of four gates: the input gate i_t, the forget gate f_t, the $\tanh$ gate g_t, and an output gate o_t. Here i_t represents the input at time step t, and likewise for the rest of the notation. The hidden state and cell state at step t are computed recursively as such $h_t, c_t = \mathbf{lstm}(x_t, h_{t-1}, c_{t-1})$.

First, the LSTM computes what information to forget by outputting a number in $[0, 1]$ based on the hidden state of the previous cell, h_{t-1} and the input vector at step t, x_t. The LSTM decides what information to store by updating certain values through the input gate layer, and the $\tanh$ gate, which compresses all values into a number in $[-1, 1]$, calculates new candidate values that can be added to the state as vectors. By multiplying the old cell state c_{t-1} by the forget gate and adding $i_t \odot g_t$, the model implements the decided upon changes. Thus, $c_t = f_t \odot c_{t-1} + (i_t \odot g_t)$. The output of the cell is based upon a filtered cell state, $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ where σ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. Thus, the hidden state is computed $h_t = o_t \odot \tanh(c_t)$. The formal computations can be seen below where $W_{f,i,c} \in \mathbb{R}^{N \times M}$ are the input weights for N LSTM cells and M inputs, likewise $b_{f,i,c} \in \mathbb{R}^N$.

$$\begin{aligned} \text{forget gate} &: f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ \text{input gate} &: i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \text{tanh gate} &: g_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ \text{cell state} &: c_t = f_t \odot c_{t-1} + (i_t \odot g_t) \\ \text{hidden state} &: h_t = o_t \odot \tanh(c_t) \end{aligned}$$

Dropout On small data sets like the SST, powerful LSTM models have a high likelihood of overfitting. To prevent this, we implement a dropout in certain linear layers of the LSTM. The dropout layer, stochastically determines which parameters to 'forget' or set to 0. For $d \in \{0, 1\}^n$ with

Task	Model	Accuracy
SST	BoW	24.05 (0.01)
	CBOW	27.66 (0.01)
	Deep CBOW	32.75 (0.02)
	LSTM	41.27 (0.03)
	Tree LSTM	46.71 (.002)
	Sub-Tree LSTM	46.00 (.004)
	Merged Model	45.29 (0.00)

Table 1: Test results for training on SST: we report average (std) across 3 independent runs for all models

$d_j \sim$ Bern(0.5), we set the dropout layer to be: $\tanh(W_d(h_t \odot d_j) + b_d)$. This method allows the model to not be reliant on one or a few particularly strong parameters, as they may be set to 0.

Binary Tree LSTM Rather than a strictly sequential ordering of the sentence encodings, we can utilise syntactic trees. For a more complex sentence representation we can use trees to obtain syntactic structures within our embeddings. Bi-Tree LSTM utilize a stack mechanism as well as shift reducers to read the sentence one word at a time, placing the word on a 'stack', while also pairing together what would be children of a syntactic tree.

3 Models

Our baseline accuracy for sentiment classification will be computed using a classic BoW model. Our models make a prediction using five sentiment classes ranging from "very negative" to "very positive". Our BoW model trains using word emebeddings of an equivalent size to the number of sentiment classes. For all models we use the same learning rate, which controls how much the model changes per iteration in response to the calculated loss. To avoid unstable training or learning sub-optimal weights we used a learning rate of 0.0005. All of the models reported in Table 1 were trained on $30,000$ iterations.

The CBOW model used an initial embedding size of 300 passed through a linear layer that outputs embeddings of equivalent length to our sentiment classes. Our Deep CBOW uses the same initial embeddings as the CBOW model, but with a linear transformation to embeddings of size 100, a $\tanh$ activation layer, followed by another linear transformation and $\tanh$ activation layer.

Our models learn only through the results of its prediction power concerning word sentiment. This does very little for learning word *meaning*. Using **GloVe** (Pennington *et al.*, 2014) we use pretrained word embeddings. The word embeddings offer a more thorough word representation as they contain information like word similarity and co-occurrence. We utilize the pretrained embeddings for our Deep CBOW model in later experiments in which results can be directly compared to Deep CBOW models without being trained on pretrained embeddings.

4 Experiments

To compare models that use word order against those that do not, it is easy to see from Table 1 that there is an increase in accuracy amongst the LSTM models relative to the different variants of the BoW models. Even the most powerful BoW model, Deep CBOW, achieves a lower accuracy than the classic LSTM model. This result is expected as language has structural dependencies; it is not sufficient to list words in any order to achieve a sentence meaning. Even increasing the initial word embedding size in the CBOW and Deep CBOW does not yield as significant an increase in accuracy as using sequential models.

Despite the results of testing the importance for word order, it is expected that experimenting on models with respect to sentence length might have the opposite results. The BoW models are predicted to be exceptionally good at sentences of arbitrarily large length, because they are memoryless, so there is no chance of information loss over phrases with long-term dependencies.

For our experiment, we examined the test data and divided it into three subgroups based on the length of sentences. The average length was 19 tokens per sentence with a standard deviation of 9 tokens. Thus, we arranged the test data into a 'small' group with less than 10 tokens per sentence, an 'average' group with $10 - 28$ tokens per sentence, and finally the 'big' group with sentences containing 29 or more tokens. All models were trained on the standard training and development data, as we did not want to specifically train the models on sentences of a particular length. The accuracy of each model for each subgroup can be seen in Table 2.

While the LSTM model accuracy did decline as the sentences became larger, it still maintained a higher accuracy than the variants of the BoW model. The dataset provided came with sentiment scores at every node in the tree. In previous exam-

Subgroup	BoW	CBOW	DCBOW	PT DCBOW	LSTM	Tree LSTM
small	23.8% (.01)	38.2% (.02)	40.1% (.01)	47% (.02)	47.9% (.01)	51.1% (.002)
average	26.27% (.02)	36.2 (.04)	36.9% (.02)	41.5% (.01)	43.7% (.01)	45.6% (.007)
big	23.9% (.02)	35.9% (.01)	38.1% (.01)	41.5% (.001)	43.7% (.01)	45.8% (.003)

Subgroup	SubTree LSTM	Merged Model
small	49.3% (.01)	47.64% (.00)
average	46% (.001)	43.92% (.00)
big	45.2% (.004)	44.98% (.00)

Table 2: Test results for the subgroups of the test data: we report the average accuracy (std) over 3 independent runs. 'small', 'average', and 'big' are sets of test sentences as described in §4.

ination the models were trained on only the root score for the whole sentence. For further improving the obtained results, the data-set was enriched by adding every subtree and its sentiment.

As the Tree LSTM is our most effective model at sentiment classification, we attempted to increase it's accuracy even further. By segmenting each tree into a set of all its subtrees, the model could use each subtree as a training example. Similar to the standard Tree LSTM, we use initial embeddings of 300, with further linear layer transformations.

Since the LSTM model and BOW approaches differ in their sentiment classification strength, combining those strengths into one model could yield beneficial results. We therefore build a new model, where the output of the trained Deep CBOW and the output of the trained Tree-LSTM are fed to an SVM, from where the overall sentiment class is predicted.

5 Results and Analysis

The importance of word order is built into the architecture of the BoW models and the LSTM models. The results were unsurprising, given common knowledge of how language and structure function together. Further research, beyond the scope of this project, suggests there might be a way to encode word order into the initial embeddings of the BoW models. While this would go against the initial motivation of 'bag-of-words' models, it could combine the simplicity of summing over vectors with the sequential approach of LSTM. Thus, it would be possible to achieve a higher accuracy rate with a similar model structure of BoW.

The results of the experiment on sentence length did go against our initial prediction. While LSTM specialized in word order, it was assumed that the benefits of BoW models came with sentences of arbitrary length. A potential bias that might have skewed the results of our experiment was the unequal number of sentences in each subgroup. The majority of sentences in the test data fell under the 'average' classification. As a result, there were significantly fewer examples to test the models on. A detailed histogram of the test data and the frequency of sentences given their length can be found in Appendix A.

Although the LSTM performs the best on longer sentences, of particular interest was the similarity of the results of the Pretrained Deep CBOW and the LSTM. The Pretrained Deep CBOW has an advantage over the LSTM in terms of computational speed, while achieving similar results.

As can be seen in Table 1, the results of supervising the model at each node of the tree does not yield significant changes. Even in other experiments concerning sentence length the Sub-Tree LSTM compared to the results of the standard Tree LSTM, the accuracies represent a minimal change. Likewise for the results of the Merged Model. The overall performance slightly decreases (as visible in table 1). Where it was hoped that through combining the Tree-LSTM with DCBOW, the merged model would both receive the small sentence prediction power of the BoW-approach as well as the LSTM sequence modelling abilities.

6 Conclusion

Several sentiment classifiers were implemented. From the obtained results it can clearly be concluded that the LSTM models on different sentence length outperform BoW approaches. The LSTM models however decrease in accuracy when the sentence length increases; this is due to its architecture, where the sentence is processed in sequence. A combination of the Deep CBOW with the Tree-LSTM show that not much improvement is achieved. A standard SVM is used to model the output of the two trained models combined; using other classification algorithms, neural networks for example, could yield better results. One could also try different combinations of models. Our approaches could further be extended to the use of bidirectional LSTM, where the sentence is processed from left to right and vice versa, or by the use of convolutional neural networks, where the convolutional masks can capture a variety of information.

7 References

References

[1] Klaus Greff, Rupesh K. Srivastave, Jan Koutnik, Bas R. Steunebrink, Jürgen Schmidhuber *LSTM: A search space odyssey* IEEE, 2016.

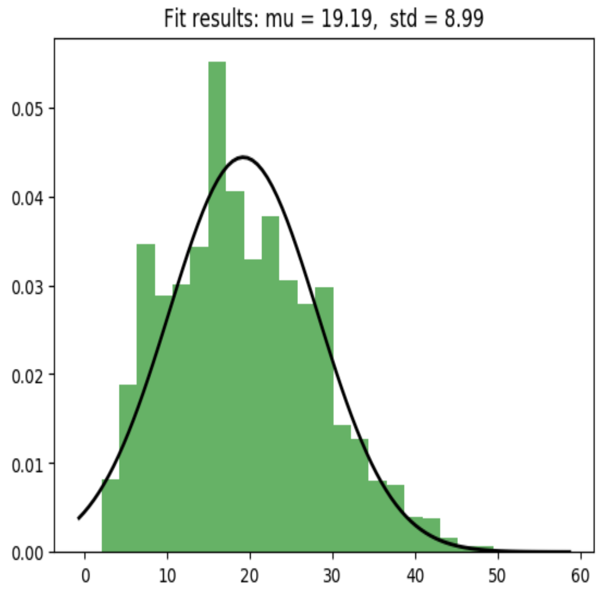
[2] Diederik Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization* ICLR, 2015.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. *GloVe: Global Vectors for word Representation.* EMNLP, 2014.

[4] Kai Sheng Tai, Richard Socher, Christopher D. Manning. *Improvised Semantic Representations from Tree-Structured Long Short-Term Memory Networks* ”Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)”, 2015

A Appendices

2 histogram.png



Histogram: Frequencies of sentences in the test data given number of tokens in the sentence