Encoding Sentences with Neural Models

Omar Elbaghdadi
12660256
omarelb@gmail.com

Aman Hussain
12667447
aman.hussain@student.uva.nl

1 Introduction

A common approach to *understanding language* is to encode units of language with learned representations. We want these representations to be "close" to each other in some semantic space when they convey a similar meaning. We hope that the understanding expressed by these representations improves our ability to perform language tasks such as sentiment analysis.

Words are units of language that convey a certain semantic meaning or even a range of meanings. However, for many tasks understanding single words is not enough. Consider classifying the sentiment of the following movie review: *"This was absolutely* ***not*** *the worst movie I have ever seen."* If we were to look only at individual words, "worst" would jump out as an indicator of negative sentiment. We know, however, that "not" interacts with "worst" and negates it.

As we have seen, when we construct sentences by *combining* individual words together, their meanings interact, generating new meanings. We call this *compositionality*. Given this is the case, our models need to be able to understand language at a higher level than words.

The main goal of this paper is to study how well different techniques for creating *sentence representations* work. We test representations by using them to classify sentiment of movie reviews (Pang et al., 2002) using the Stanford Sentiment Treebank (Socher et al., 2013). Given this task, we explore the following directions:

- Sentences are intrinsically ordered: the *order of words* in a sentence changes its meaning. Does taking this sequential information into account help to yield better sentence representations? We compare the performance of Continuous Bag of Words (CBOW) models (Mikolov et al., 2013a), which throw away word order information, and LSTMs (Hochreiter and Schmidhuber, 1997), which do take word order into account. Intuitively, we expect to find performance increases when incorporating word order. Our experiments agree.

- Does taking phrase *compositionality* into account lead to better sentence representations? To deal with the compositional structure of phrases, Tai et al. (2015); Le and Zuidema (2015); Zhu et al. (2015) introduce Tree-LSTMs. We show that Tree-LSTMs work better on the sentiment analysis task than regular LSTM and CBOW models. This is what we expect, as sentences intrinsically exhibit compositional structure.

- Does sentence length have an effect on model performance? The amount of words is an important property of a sentence. Intuitively, longer sentences become harder to interpret than shorter ones, as there is more information and ambiguity to make sense of. We find that this is indeed the case.

- The treebank does not only contain sentiment scores at the sentence level. It also provides us with sentiment scores at each node in the tree. We want to know if using this information increases performance. We expect it does, since we are able to train on more relevant data. However, we do not see a significant increase in performance. We state the possible reasons for this in Section 5.

- Thus far, we have framed the task as a classification problem, where sentiment can range from "very negative" to "very positive". However, when the true label is "very negative", predicting "very positive" is worse than predicting "negative". To make use of this infor-

mation, we frame the task as a regression problem. We expect this approach to work better, as we use a training signal that incorporates more information about the task. However, we find that performance slightly decreases with this approach.

2 Background

Our models range from simple to sophisticated and will be introduced in this order.

2.1 Bag Of Words

In the neural BOW model, we represent each word with a vector. These vectors are of the same size as the amount of target classes, 5 in our case. The sum of these vectors gives the output activation of each class. We predict the class with the highest activation. Our task is thus to learn parameters for each word such that we maximize some objective, e.g. predictive accuracy. Since we sum each vector value, *word order information is lost*.

Given a word, its learned vector representation, which we call a *word embedding*, will hopefully encode a useful representation of that word in the vector space that it's mapped to.

2.2 Continuous Bag of Words

The word embeddings introduced in Section 2.1 can be generalized. Instead of fixing the size of the embedding vectors to the number of target classes, we fix them to any arbitrary size d. We believe that high dimensional word embeddings will be able to capture more interesting semantic information.

A small dataset like ours may result in word embeddings of poor quality. We therefore use pre-trained word embeddings, *word2vec* vectors (Mikolov et al., 2013b), and keep them fixed.

Analogously to the normal BOW case, we then sum the word embeddings for each vector in a sentence. However, since its BOW representation is now d-dimensional, we also learn a linear mapping that projects the BOW representation to the number of classes, which are then the class activations.

2.3 LSTM

Recurrent neural networks (RNN) are neural networks designed to work with sequences. An RNN contains an internal state that allows it to take into account information it has encountered earlier. Vanilla RNNs tend to have problems retaining information over long periods of time. LSTM models (Hochreiter and Schmidhuber, 1997) are a variant of RNN models that alleviate this problem a great deal by using smart gating mechanisms.

Being able to deal with sequences allows us to incorporate word order information, as BOW models are unable to, into our predictions.

2.4 Tree-LSTM

Tree-LSTMs generalize regular LSTM models to tree-network topologies. While regular LSTMs take as input a *single* hidden state and cell state computed in a previous time-step, Tree-LSTMs allow for an arbitrary number of input hidden and cell states given to it by its *children* nodes.

Tai et al. (2015) introduce two variants of the Tree-LSTM: Child-Sum Tree-LSTMs and N-ary Tree-LSTMs. For most computations, the first variant sums its childrens' hidden states. This throws away order information at the node level. For sentiment classification, we use the latter variant, which does *not throw away order information*. Specifically, we apply it to the constituency parse trees of sentences, which are binary trees.

3 Models

In the *CBOW* model, we use three different architectures. The simplest one uses a single linear layer to map from word embeddings to class activations. The second one, which we call *Deep CBOW*, uses 3 linear layers, with a `tanh` nonlinearity between layers. The third one, which we call *PTDeepCBOW*, uses pre-trained word embeddings on top of Deep CBOW.

The LSTM model processes all words in the sentence sequentially. We project the final output hidden state, which can be seen as a latent representation of the sentence, to a probability distribution over classes. This projection is done using a single linear layer with a dropout layer before it.

The Tree-LSTM model takes as input the constituency parse tree of a sentence with sentiment at each node. Starting from the leaf nodes, the tree-LSTM combines hidden representation of its children nodes in a bottom-up fashion using the N-ary Tree-LSTM model with $N = 2$. This is done until we reach the root node. The hidden representation of the root node is then used for classification. We project the hidden representation using a single hidden layer and dropout layer before it.

Every model outputs class activations. These activations are first run through a softmax layer to get

a probability distribution over classes. The model is then evaluated using a cross entropy loss. We also evaluate a model by computing the accuracy of its predictions: the number of correct predictions out of all datapoints.

To frame the task as a *regression problem*, we simply append a feedforward network layer mapping the output of previously mentioned classifiers to a single number. This number is run through a sigmoid and multiplied by 4, so that it is constrained to the range $[0, 4]$. To make predictions, we round an output to the nearest integer.

4 Experiments

4.1 Dataset

The Stanford Sentiment Treebank dataset consists of 11,855 sentences. All sentences are parsed using a constituency parser and stored as binary trees. A sentence tree contains sentiment scores at each node. Each sample is assigned one of five sentiment labels: "very negative", "negative", "neutral", "positive", and "very positive". We split the dataset into training (8,544 sentences), development (1,101 sentences) and test (2,210 sentences) sets.

4.2 Training Procedure and Evaluation

We train each model for 10,000 iterations, as validation losses seem to have converged by this point. We train each model on 3 different seeds. The models that perform best on the validation set are then evaluated on the test set.

Pre-trained word2vec embeddings of size 300 are used by all models except BOW and CBOW. Every model is trained using the Adam optimizer (Kingma and Ba, 2014).

The BOW model uses a learning rate of 0.0005. The Deep CBOW model uses hidden layers of size 100. The LSTM hidden state has a dimensionality of 168. We use a dropout keep probability of 0.5. The Tree LSTM has a hidden dimension of 150. A slightly lower learning rate of 0.0002 is used for training the recurrent models.

4.3 Framing it as a Regression Problem

The regression task is evaluated using a mean squared error (MSE) loss during training. Although the regression layer could be added to any model, we only experiment with the LSTM to save computation time.

4.4 Other Experiments

We have sentiment scores at every level of a parse tree. By treating *every subtree of a sentence* as a training sample, we can perform sentiment classification at the node level. This gives us more data to work with. We extract all subtrees from each sentence and remove all subtrees containing just a single leaf node. To keep the amount of data bounded, we sample 8000 subtrees, excluding the complete sentences, and add them to the original dataset. We will refer to this experiment as Subtree LSTM.

To study the effect of *sentence length* on the performance of our models, we split the test set by sentence length: Very Short, Short, Medium, Long, and Very Long. We evaluate the trained model on each of these subsets. For exact sentence lengths, see Table 2 in the Appendix.

5 Results and Analysis

Model performance is shown in Table 1. Validation accuracies over training iterations are shown in Figure 1. Accuracies are averaged over only 3 runs, making significance testing reasonably unreliable.

Table 1: Model performance evaluated on the test data. Accuracy is averaged over three runs. Standard deviations are given in parenthesis.

Model	Test Accuracy (std)
BOW	0.2226 (0.0114)
CBOW	0.3066 (0.0137)
DeepCBOW	0.3356 (0.0057)
PTDeepCBOW	0.4296 (0.0060)
RegressionLSTM	0.4376 (0.0093)
LSTM	0.4516 (0.0061)
SubtreeLSTM	0.4608 (0.0169)
TreeLSTM	**0.4700** (0.0069)

5.1 Word Order and Word Embeddings

Overall, using pre-trained word embeddings leads to the highest gain in evaluation accuracy. This is made especially clear by Figure 1. All the best models use pre-trained embeddings. The learning is made easier by the pre-trained embeddings, which can be seen as a form of transfer learning.

Table 1 shows that incorporating word order information leads to modest gains in performance, as all recurrent models outperform the best BOW model.

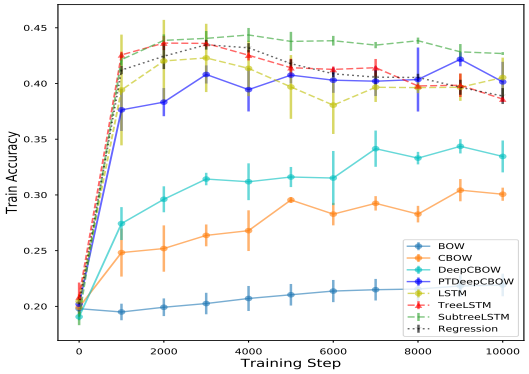

Figure 1: Validation accuracies of different models across the training iterations. Points are represented by an average over 3 seeds. Error bars represent standard deviations.

Out of all models, the Tree-LSTM performs best. This implies that sentence compositionality does aid in creating better sentence representations. The Tree-LSTM model is able to capture compositional semantics from the interaction of words at different levels in the sentence tree. Other models don't use this information.

In Figure 1, we see that Tree-LSTM quickly starts to overfit. The Tree-LSTM model has millions of parameters, whereas our dataset only has 8,000 training samples. Even though we use dropout, heavier regularization may decrease the amount of overfitting.

We expected the Subtree LSTM model to perform better than the regular Tree-LSTM. Unfortunately, it actually performs slightly worse. However, the uncertainty in its accuracy estimate is quite high, so we can't make strong conclusions. The Subtree LSTM might benefit from training longer, since it does not seem to overfit as much (Figure 1). Moreover, increasing the amount of subtree samples may lead to performance gains.

5.2 Effect of Sentence Length

As we can see in Figure 2, the general trend is that accuracy decreases as sentence length increases. We find that recurrent models outperform BOW models for longer sentences.

It may be easier to extract the signal from fewer words, resulting in high performance in the "Very Short" test set. Since longer sentences can introduce more noise than signal, we need recurrence to extract the sequence information in a principled way.

In Figure 2, we have only plotted the results of some representative models to highlight the trends and avoid overcrowding. For a more detailed plot, see Figure 3 in Appendix A.

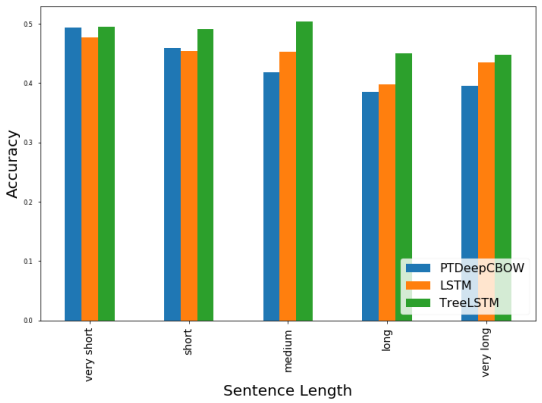

Figure 2: Test accuracy with increasing sentence lengths

5.3 Sentiment Analysis as a Regression task

We had hoped that using label ordering information would yield a better training signal, but we find that the regular LSTM model does better than the Regression LSTM model.

Looking at our model predictions as sentiment "scores" rather than labels does give us a different perspective on the problem. Prediction error gives us a more granular view on how wrong our prediction is, and if the model thinks it is more positive/negative than it is.

6 Conclusion

In this paper, we evaluate techniques for creating sentence representations on the sentiment classification task. We show that: word order is important; Tree-LSTMs outperform their recurrent counterparts, which agrees with findings of Tai et al. (2015); sentiment classification is harder when sentence length increases; and that supervising sentiment at the node level decreases overfitting, but does not lead to performance improvements. We present a method for framing the sentiment classification task as a regression problem, which has, to the best of our knowledge, not been done for this specific task before. Although this does not lead to performance improvements, it allows for a different and useful perspective of the problem. Interesting further work would be to analyze why the regression case does not work as well as expected, and to explore the benefits of both perspectives.

References

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

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Phong Le and Willem Zuidema. 2015. Compositional Distributional Semantics with Long Short Term Memory. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 10–19, Denver, Colorado. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. ArXiv: 1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. 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, Seattle, Washington, USA. Association for Computational Linguistics.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. In *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)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1604–1612, Lille, France. PMLR.

Appendix A

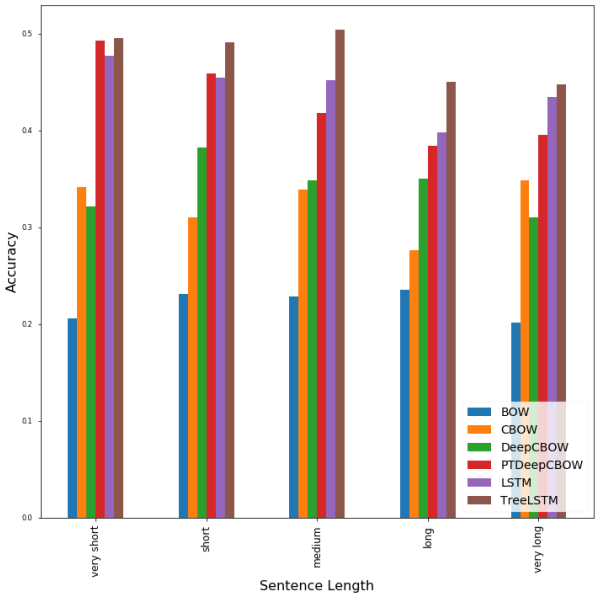

Figure 3: Accuracy with increasing sentence lengths

Table 2: Test data to measure effect of sentence length on performance

Test Subset	Sentence Length	Sample Size
Very Short	2 - 11	442
Short	11 - 16	442
Medium	16 - 21	442
Long	21 - 27	442
Very Long	27 - 56	442