Representing Sentences with Neural Models

Nikolaos Apostolikas
14343231
nick.apostolikas@student.uva.nl

Orestis Gorgogiannis
14480778
orestis.gorgogiannis@student.uva.nl

1 Introduction

In this report, the two key questions are how a task-specific representation of a sentence is learnt and how a prediction is made for the downstream task using this representation. The task is sentiment classification of movie reviews. Therefore the representation of a sentence has to be specialized on sentiment. For this purpose, the Stanford Sentiment Treebank (SST) (Socher et al., 2013) was used with a variety of methods, including different BOW and LSTM models.

During the experiments, our goal was to measure the importance of different aspects in our task. Word order, sentence length and other aspects were tested with all the models in order to get an accurate view of how they affect the final results. In order to determine the effect of convolution on our task, two CNNs were also implemented, trained and tested. One uses pre-trained embeddings and the other is a hybrid RNN-CNN model, in order to compare its performance to the other approaches.

Regarding the expectations, the LSTM based techniques would generally perform better than the BOW ones. Word order contains information that the BOW methods don't take into account and sentence length can significantly change the BOW results. The pre-trained embeddings were expected to significantly improve performance, since they are expertly trained representations.

2 Background

2.1 Word embeddings

A Word Embedding is a numerical representation of a word in the form of a vector in a predefined vector space. Each word is mapped to a vector, which encodes its meaning. The approach is to use dense vectors, in order to reduce the dimensions. The two most common word embeddings are word2vec (Mikolov et al., 2013) and GloVE (Pennington et al., 2014). The intuition behind GloVE is to represent words in a high-dimensional space such that words that have similar meanings are located near each other in this space.

2.2 Bag of Words

The Bag of Words is a technique that uses a vector to express the sentiment carried by each word and therefore has the same size as the number of classes. These vectors are then summed together with a bias vector to obtain the sentence vector.

The Continuous Bag of Words model builds upon the BOW technique by using an arbitrary vector size. It allows for more information with trainable embeddings for each word. For the final step, the sentence vector is resized to the number of targets by multiplying it with a trainable matrix.

The Deep Continuous Bag of Words architecture can be described as a CBOW model, but with more layers and thus is able to capture more complex features. Instead of only using a final linear transformation, a multi layered network is built to more accurately predict the sentiment. Non linearities are also added to increase the complexity.

2.3 LSTM

Long Short Term Memory networks (LSTMs) (Greff et al., 2017) are a unique kind of network that allows for long-term dependencies between words. An LSTM uses a complex type of RNN (Sherstinsky, 2020) cells. Each cell uses both the original input and the output of the previous cell to compute its own output.

Inside the cell, the previous layer output h and the original input x are transformed by the forget gate, which decides which parts of the information are relevant. It creates a vector f with the relevant

information:

$$f = \sigma(W_{if}x + b_{if} + W_{hf}h + b_{hf})$$

Then, the input gate i decides which parts will be overwritten and the cell creates a new candidate state g.

$$i = \sigma(W_{ii}x + b_{ii} + W_{hi}h + b_{hi})$$
$$g = \tanh(W_{ig}x + b_{ig} + W_{hg}h + b_{hg})$$

Finally, the new cell state c' is obtained by combining the previous cell state c with the parts calculated. The output gate o then calculates what parts of the cell state will be in the final output.

$$o = \sigma(W_{io}x + b_{io} + W_{ho}h + b_{ho})$$
$$c' = f * c + i * g$$

By using $\tanh$ on the new cell state, the final cell output is computed.

$$h' = o \tanh(c')$$

2.4 Tree LSTM

A Tree LSTM (Tai et al., 2015), (Le and Zuidema, 2015), (Zhu et al., 2015) functions like a regular LSTM. But instead of a linear structure of cells, they are organised in a n-ary tree. A binary tree LSTM means that every cell gets the output of its two children and adds them together. It also computes a separate forget gate for each of them to decide what parts of each child's activation to keep in the final output. The Tree LSTM structure outperformed the previous regular LSTM baselines in sentiment classification when it was introduced.

2.5 Convolution

Convolution is a method usually applied in computer vision related tasks. In recent years, however, there have been uses of Convolutional Neural Networks (CNNs) in NLP tasks with adequate performance (Kim, 2014).

A convolutional layer uses filters, which slide over the words. A sentence can be represented as a sequence of words:

$$X_{1:n} = X_1 \oplus X_2 \oplus ... \oplus X_n$$

The convolutional layer generates a feature map when a filter, which is an array of weights, is used on a window of h words. The feature map is given as follows:

$$c_i = f(w \times x_{i:i+h-1} + b)$$

Finally, the max-pooling layer keeps the most important feature from every feature map and passes them to the FC layer.

Model	BS	lr	Hid. dim
BoW	1	0.0005	-
CBoW	1	0.0005	-
Deep CBoW	1	0.0005	100
PT Deep CBoW	1	0.0005	100
LSTM	1	0.0003	168
Mini batch LSTM	25	0.0002	168
Tree LSTM	25	0.0002	50
PT CNN	1	0.005	300
RNN CNN	1	0.001	100

Table 1: Model settings and hyperparameters

3 Models

Overall, 9 different models were trained. 4 using variations of the BoW method, 3 LSTM models and 2 CNN-based models. The embedding dimension for all 9 models is 300. The hidden dimensions for each model vary and the specific settings can be seen in Table 1.

In LSTMs, the classifier uses a single cell, which consists of 4 linear layers, and updates the hidden state one word at a time. There is also one dropout layer to prevent overfitting. Mini-batch LSTM remains the same model, but is ran using mini-batches. The Tree LSTM in the experiment also uses mini batches. All LSTM models use the GloVE embeddings. A batch size (BS) of 25 was used for the Tree-LSTM and the mini-batch LSTM model.

The CNN model uses the same GloVE embeddings, has a max-pool and dropout layer with ReLU non-linearities. The RNN-CNN model shares the same architecture as the simple CNN model with the addition of an RNN in the beginning. This RNN computes the embeddings for the words by using the following function for each element in the input sequence:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

Therefore, it takes into consideration the previous state to compute the current state.

All models have a simple lookup table that stores embeddings of a fixed dictionary and size. The input to the dictionary is a list of indices, and the output is the corresponding word embeddings. Finally, every model uses the Cross Entropy loss function and Adam optimizer. The sentiment scores in the classes range from 0 to 4, with 0 being the most negative and 4 the most positive.

4 Experiments

The experiments conducted were the following:

- Shuffling the words in each sentence to check whether the word order plays a role in performance.
- Splitting the test dataset into short and long sentences and performing two separate evaluations on each one. This allows to measure the significance of sentence length.
- Using all possible subtrees out of every tree string in the data, to see if supervising the sentiment score in every level is beneficial to this task.

To evaluate the models, the accuracy metric was used. All experiments are run three times with three different seeds to minimize the stochasticity of the results.

5 Results and Analysis

All models were trained using the same amount of iterations (30000). This choice was made to measure the effect of the experiments on the models, regardless of the convergence speed. It is theoretically known that BoW based models converge slower than LSTMs. Furthermore, models that use mini-batches will, of course, converge in less iterations due to the additional data they receive during training. Regarding fairness, training iterations were kept constant throughout the experiments.

The baseline results for the initial experiment can be found in Table 2. The results of the different experiments in each model are in Tables 4 and **??** .

5.1 Baseline Performance

As expected, CBoW performs better than the simple BoW model. However, increasing the model depth did not improve performance, with Deep CBoW scoring only marginally better than CBoW. The GloVE embeddings, though, improve the Deep CBoW model, reaching 43% accuracy. The BoW models could perform better if trained for more iterations, as their convergence speed is lower than the other models, but this is beyond the scopes of the experiments.

All three LSTM models outperform BoW, with Tree-LSTM being the top performer. The tree structure is beneficial to the model's understanding of the data. Mini-batching provides additional power to the training procedure and this is why mini-batch LSTM and Tree-LSTM are the top performers.

Model	Baseline Accuracy
BoW	0.249 $\pm$0.017
CBoW	0.348 $\pm$0.016
Deep CBoW	0.367 $\pm$0.009
PT Deep CBoW	0.434 $\pm$0.010
LSTM	0.460 $\pm$0.010
Mini batch LSTM	0.462 $\pm$0.003
Tree LSTM	0.470 $\pm$0.006
PT CNN	0.427 $\pm$0.005
RNN CNN	0.385 $\pm$0.004

Table 2: Baseline accuracy of models

Interestingly, between the CNN based approaches, the pre-trained embeddings yield better results than the hybrid RNN-CNN model. This shows the power of the pre-trained embeddings on our task. All in all, the CNNs perform better than the simple BoW models, but Deep CBoW with pre-trained embeddings reaches the same accuracy as the pre-trained CNN.

5.2 Performance during Experiments

5.2.1 Word Order

Word order in a sentence affected some models more than others. The BoW models perform identically to the baseline. This is within expectations as the BoW technique doesn't take word order into account and aggregates the words in a sentence by summing over their word scores. The same goes for the RNN-CNN model, since it used the same vocabulary as the BoW models. Small drops in performance were observed in all LSTM models. Word order has an affect on them, but their memory-based architecture and the powerful pre-trained embeddings protect them from suffering huge drops in accuracy. Finally, the CNN model using GloVE embeddings was surprisingly able to achieve perfectly identical results when trained in shuffled sentences. The ability of the convolutional layers to focus on finer features at the beginning and then move to wider area features afterwards, allows the model to understand the sentence meaning regardless of the word order.

5.2.2 Sentence Length

The next experiment was about sentence length. It is clear that in most cases, the models perform better when predicting the sentiment in short sen-

tences. Longer sentences confuse most models, making it harder to actually predict the true sentiment due to the abundance of words. In the experiment results, though, some outliers can be observed.

Deep CBoW performs almost identically in short or long sentences. The multi-layer network can accurately learn enough features to not fall off when facing large inputs. The aggregation of the BoW approach also helps in this case, since the model doesn't get confused with long-distance dependencies. The basic LSTM model also behaves similarly.

Mini-Batch and Tree-LSTM models suffer a lot in longer sentences, but the normal LSTM retains its accuracy. That can only mean that training with one sentence at a time allows the model to more effectively learn how to deal with sentences of any length.

Finally, the CNN using the pre-trained embeddings had no differences in performance, but the one trained from scratch provided an uptick in accuracy, which is interesting. The combination of an RNN and a CNN being trained together for this downstream task might be what allows the model to adapt to the longer sentence length. Further research might be needed around this aspect.

5.2.3 Subtrees

The final experiment gave the models more data to train with, but the information mostly confused the models. Low level trees with one or two words carry information that give the model certain kinds of biases. It can be theorised that the models learn to expect a specific score for each word, which is a step towards the wrong direction when dealing with whole sentences. These biases are even stronger in our specific experiment settings, since the training iterations used are not enough to pass through the newly expanded dataset multiple times.

The uptick in accuracy for mini-batch LSTM and tree-LSTM indicates that this approach is helpful in training at least for the LSTM-based models. Using batches allows the model to more efficiently go through the data in a smaller amount of iterations. The tree structure also benefits a lot from this approach, as can be seen by the improvement in performance of Tree-LSTM compared to mini-batch LSTM. All other models experienced important drops in accuracy, however, which means that convergence speed was significantly slowed by this method.

Model	Word Order	Subtrees
BoW	0.249 $\pm$0.008	0.214 $\pm$0.017
CBoW	0.350 $\pm$0.011	0.289 $\pm$0.015
Deep CBoW	0.369 $\pm$0.002	0.308 $\pm$0.012
PT Deep CBoW	0.446 $\pm$0.009	0.402 $\pm$0.011
LSTM	0.426 $\pm$0.001	0.421 $\pm$0.018
Mini batch LSTM	0.420 $\pm$0.009	0.469 $\pm$0.003
Tree LSTM	0.446 $\pm$0.004	0.482 $\pm$0.007
PT CNN	0.427 $\pm$0.003	0.397 $\pm$0.015
RNN CNN	0.388 $\pm$0.008	0.324 $\pm$0.004

Table 3: Accuracy of models for word order and subtrees experiments.

Model	Short	Long
BoW	0.267 $\pm$0.011	0.242 $\pm$0.002
CBoW	0.370 $\pm$0.004	0.338 $\pm$0.014
Deep CBoW	0.371 $\pm$0.016	0.372 $\pm$0.007
PT Deep CBoW	0.419 $\pm$0.023	0.443 $\pm$0.030
LSTM	0.443 $\pm$0.033	0.444 $\pm$0.034
Mini batch LSTM	0.486 $\pm$0.005	0.436 $\pm$0.001
Tree LSTM	0.495 $\pm$0.006	0.445 $\pm$0.005
PT CNN	0.425 $\pm$0.023	0.422 $\pm$0.026
RNN CNN	0.355 $\pm$0.002	0.372 $\pm$0.005

Table 4: Accuracy of models on short and long sentences.

6 Conclusion

Through the experiments conducted, it became clear that word order is not as important as intuition would suggest. Some models remained unaffected, while others dropped in performance, but not by a significant amount. The LSTM models seem to be affected the most by this input change. Most models achieve better performance with shorter sentences rather than longer ones. The idea of supervising the sentiment in each tree node in the treebank could prove beneficial, given enough computational time and further research could be useful.

GloVE embeddings provide a great starting point for the models and help them perform better, by leveraging this pre-existing knowledge to learn more effectively.

The use of a hybrid RNN-CNN architecture provided interesting results. Hybrid models are immune to shuffled word order and perform better on longer than shorter sentences, though not by a lot. Further experimentation with the hybrid-CNN models could yield interesting results in NLP tasks.

References

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2017. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. *arXiv preprint arXiv:1503.02510*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y 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.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612. PMLR.