Practice 2: Sentiment Analysis

Mahsa Mojtahed
12166510
mahsa8mojtahedi@gmail.com

Victor Zuanazzi
12325724
victorzuanazzi@gmail.com

1 Introduction

Different methods can be used for sentiment analysis or labeling a corpus. Two of such methods are Bag of Words (BoW) models and Long Short Term Memory (LSTM). We have implemented a number of variations of these methods on the Stanford Sentiment Treebank (Socher et al., 2013). This data set consists of single sentence movie reviews which have been pre labeled in a range of 0 *Very Negative*, to 4, *Very Positive*. Three different variations of BoW models are implemented: simple BoW, continuous BoW and Deep CBoW. Another Deep CBoW model is trained using word2vec, which is a set of pre trained word embeddings.(Mikolov et al., 2013) Additionally, an LSTM and a binary tree LSTM (Tai et al., 2015) (Zhu et al., 2015) (Zhu et al., 2015) have been implemented and tested on our corpus using word2vec. Binary tree LSTMs are known to perform better than LSTMs on sentiment analysis(Tai et al., 2015) These models are trained and evaluated on their ability to label each review correctly. The following research questions are proposed regarding their performance and the effects of each model's strengths on sentiment analysis.

We first investigate the importance of word order on a model's ability to analyze the sentiment of a review. Word order often plays an important role conveying meaning, since changing the word order can cause a change in meaning. LSTMs take word order into account and BoWs neglect to do so. We investigate if BoW models can reach a similar performance with LSTMs despite their inherit limitations. The importance of word order is then further investigated by scrambling words in each review and testing our LSTM and tree LSTM models on this new data. The robustness of these models to the change of word order is evaluated. A low sensitivity to change of order indicates that word order information is not crucial in sentiment analysis.

The robustness of all models to different sentence lengths is also explored. Knowing which model is able to work with a corpus of various sentence lengths gives us insight for choosing the best model for our task.

The most complex model among the ones that are being investigated are tree LSTMs. Consequently, they are the most computationally expensive, it is also required parsing sentences in trees before hand. By comparing its performance to other models and seeing if other models could reach the same accuracy level, we can infer if the additional complexity is worthwhile.

In an additional experiment with tree LSTMs, the sentiment of each sentence is supervised at each node as well as the whole sentence. The effect of this on the performance of our model is evaluated. This approach takes a much bigger data set, as every word and sub-trees are now present in the training set as an separate data-point. However, implanting it increases the number of evaluations during training and is therefore costly.

It was observed that the LSTMs generally have a higher accuracy than BoW models. However, the most complex BoW model, deep CBoW with pre trained data, was able to achieve a performance close to the simplest LSTM. The increase in complexity of Tree-LSTMs do not seem to be justified, it does not achieve better accuracy.

2 Background

For this work we walk through a number of techniques and models used for sentiment analysis of movie reviews. The techniques used are briefly listed bellow.

Bag of Words: This model represents the sentence by the presence of a word, it ignores word

order or dependencies. The sentences *Dress the woman, not the brand* and *Brand the dress, not the woman*, despite having different semantics, have the same representation. BoW models are easy to implement, but their simplicity causes them to lose information.

Linear classifiers: (Logistic Regression) are single layer neural networks with one output node for each class. Only linear transformations are carried by it.

Word Embedding: each word is represented by a vector of real numbers. The minimum word embedding size is the number of classes. Longer word embeddings carry more semantic information. The simplest model used here uses BoW, word embeddings and a Linear Classifier for sentiment analysis.

Our models scale in complexity by making use of **Multi Layer Perceptrons**. In each layer there is a linear transformation followed by a non-linear activation function. The stacking of layers allows MLPs to model non-linear relations.

In order to capture word order a Recurring Neural Network (RNN) called **Long Short Term Memory** (LSTM) was also used. They also get word embeddings as inputs, but unlike BoW, they receive this embeddings one by one. LSTMs use a set of non-linear transformations in order to filter out irrelevant information. It is more costly to train LSTMs than MLPs.

LSTMs are only able to capture word order, and do not take long distance dependencies into account. Tree LSTMs are a variation on LSTMs that are capable of modeling these dependencies. The relation between two distant words in a long sentence is better captured by a tree LSTM.

Binary Tree LSTM. The tree structure in the reviews is captured by stacking with the help of a transition sequence. For each node, the tree LSTM processes two child nodes. This is done recursively, starting from the leaves and parsing the tree structure to the root. This model is able to carry the most information among all the tested models.

3 Models

The BOW models gives the sum of the embeddings of all words in a sentence to a neural network to calculate the sentence scores. Then, it reports the argmax of these scores as the final sentence sentiment. BoW and CBoW models use a single linear transformation to calculate the sentence scores from word embedings. Deep CBoW, on the other hand, uses a two hidden layers with tanh activation functions to map inputs to outputs. BoW has embedding size 5, CBoW and Deep CBoW have input size of 300. The output is always 5 sentiment classes.

LSTMs use a non-linear, recurrent structure for input to output mapping. This structure includes three sigmoid and two tanh activation functions. The implementation was done using pytorch. A drop out layer is defined to mitigate the effect of overfitting. Word embeddings are passed through this mapping one by one, and the final sentence sentiment is the argmax of the output after the last word in that sentence. Weight vectors are updated after each sentence. To make calculations more efficient, all linear transformations are done using two concatenated weight matrices. LSTMs are also trained by using mini-batching, where weights are updated after every batch of examples. Padding words are added to the end of sentences, this ensures all sentences in the input matrix have the same length.

Tree LSTMs are similar to LSTMs in the mapping and training procedure, but they differ in the order in which input words are given to the model. Tree LSTMs process sentences follwoing to their tree structure. The tree is stacked according to a transition sequence and passed to the LSTM cell. The only difference in formulation now is that they are always processing binary trees, and therefore have two forget gates. Tree LSTMs are also trained with batched data. For both models we used the pretrained word embeddings. Tree LSTMs had 168 as the dimention of the hidden layer and LSTM had it as 150.

For all models, weights are initialized randomly, the Adam optimizer is used and torch.optim (Paszke et al., 2017) is used to update them with cross entropy loss function.

4 Experiments

We are testing different models for sentiment analysis of a set of one-sentence-long film reviews. The reviews are from Stanford Sentiment Treebank[reference]. These examples are first shuffled randomly and then divided into training, validation and test sets. The table 1 in the appendix summarizes the architecture of each model.

BoW, CBoW and Deep CBoW initialize weights and word embeddings randomly. Those

embeddings are learned during training.

All models that use pre trained word embeddings do not train those further, they rather learn how to use those embeddings for the classification task. The pre trained word emeddings were generated by the word2vec method [reference]. The padding token $<pad>$ has a embedding of zeros and the token for unknown words $<unk>$ has a randomly generated embedding in the range [-0.05, 0.05].

For LSTM and Mini- Batch LSTM, the drop out probability for the drop out layer is $p = 0.5$. NLTK's tree is utilized to construct tree examples out of our reviews.

For Tree LSTMs we first train the model by using the full reviews and only the sentiment of the root as a label. We then proceed to train the model using all subtrees of each sentence, including single words and the whole original tree. They are extracted using NLTK. Note that only the training examples were affected by, and the model was tested and validated using whole sentences.

Also, the effect of different sentence lengths on the performance of each model is tested. This is done by dividing the test set by sentence length, and then calculating the accuracy of each model on each of these groups.

Finally, the importance of word order is further tested by scrambling the words in the sentences of the test data fed to LSTM and tree LSTM model.

The hyper-parameters used in each model are illustrated in the table 2 in the appendix.

Each model was trained at least three times with different random seeds. The average and standard deviation is reported. The models are evaluated at certain intervals during training by calculating the accuracy of their prediction of validation set. If the model's accuracy increases the parameters are saved. During training, both training loss and accuracy is monitored.

5 Results and Analysis

The averaged accuracy of all models over three runs is depicted in Figure 1. It can be seen that the sub-tree tree LSTM is able to achieve the highest accuracy of all. BoW models, except Pre-trained Deep CBoW, are generally worse that LSTM models, and the simple BoW is the least accurate. The standard deviation of model performance for 3 runs is also depicted in Figure 1. The deviations are mostly very small. Table shows the results

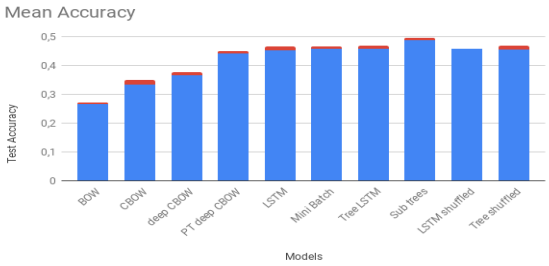

Figure 1: Comparison between the implemented models. Accuracies are averaged over 3 trainings with different random seeds. Standard deviation over all trainings is depicted in red.

of the significance test. It can be seen that the BoW models that do not use pre trained embeddings have a significant difference with the rest of the models. Pre trained Deep CboW is able to reach results that are not significantly different from most of LSTM models. Tree LSTM and LSTM models do not show a significant difference whether they are trained with scrambled or ordered sentences. The histogram of sentence length is depicted for our corpus in Figure . The distribution of the sentence length seems to obey a log-normal distribution. Most of our corpus is comprised of sentences with lengths between 6 to 31.The performance of each model on different sentence lengths is shown in Figure . It can be seen that most models perform uniformly well on all sentence lengths that they have received enough examples from. BoW models, do not perform well compared to other models unless they are trained with pre trained word embeddings. BoW and CBoW can only capture linear relations between data and label. LSTMs have a more complex mapping. Also, BOW models do not take word order into account. LSTMs, on the other hand, are capable of utilizing this information. Additionally, tree LSTMs, unlike LSTMs and BOW models, can use the long-term dependency information.

We can use this difference to investigate the importance of word order in our task. The results show that in general, BoW models are performing significantly worse that LSTM models. However, if the most complex BoW model is trained with pre trained embeddings, it can achieve a performance close to the simplest LSTM, without taking word order into account. This shows that the difference in performance of BoW models and LSTMs could come from lack of complex map-

ping, not an inability to receive word order.

In order to eliminate other differences, we focus on LSTM. LSTM and Tree LSTM are fed scrambled sentences as training sets and tested on normal sentences. By scrambling, we have eliminated any meaningful word order relations in the data set. Therefore, models are trained without the advantage of seeing word order. If they are able to achieve the same accuracy on the same test set, we can deduce that in this case, word order does not provide the models with important information. The results show that this is indeed the case. Neither model's accuracy has dropped significantly without the advantage of word order information. Also, the shuffled models still reach significantly better results that the 3 simplest BoW models. This is another indicator that at least for our corpus, the difference in the performance of BoW and LSTM is due to structural complexity of LSTMs. The fact that word order does not have an important role in our corpus could be because our reviews are not long enough to capture word order importance. Also, our corpus may not be large enough to have many cases in which word order is vital for conveying the correct meaning. Next, the performances of all models are inves-

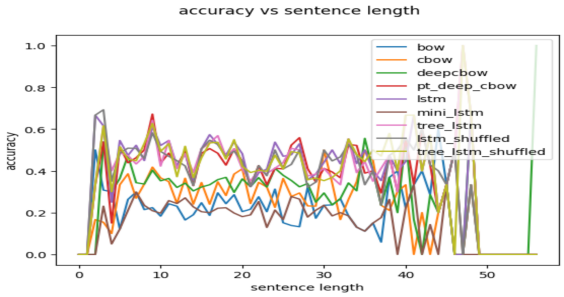

Figure 2: Accuracy of each model per sentence length

tigated on sentences with different lengths. The results shown in Figure indicate that models are rather robust to different sentence lengths. Their performance seemingly increased for very long sentences, but when we look at the distribution of sentences in our corpus, we see that this high accuracy is most likely due to the fact that there are not many long sentences, and the model can achieve a high accuracy by predicting just one right. Therefore we conclude that sentence length is not an important factor in our analysis. This could also be because our reviews are too short for their length to play an important part.

We can see in Figure 1 that tree LSTMs perform relatively well on our corpus, but their performance is not significantly better than normal LSTMs. This can be expected, since we concluded that word order is not a deciding factor in our examples. Thus an ability of detecting it better is not an advantage.

However, using Tree LSTM could boost the performance, if sentiment is supervised at each node in the tree. We can see that the tree LSTM that is fed all of the subtrees of the training set can achieve the highest performance among all models. This is because this model gets more detailed information on our data during training, and is trained with a better understanding of each word sentiment.

5.1 Conclusion

We can see that the best performance is from tree LSTMs with sub trees as training data, and the worst performance is from simple BoW model. However, tree LSTMs in general do not have a significantly better result than normal LSTMs, contrary to what we expected. This can be explained by the fact that word order is not a deciding factor for our task. This has been shown by testing on shuffled sentences. It was also shown that our models show robustness to change in sentence length. Both of this can be because our corpus is limited and one sentence reviews are too short. Therefore, repeating these experiments, especially the training data shuffling, with a corpus of longer reviews can be done in the future.

Appendix A

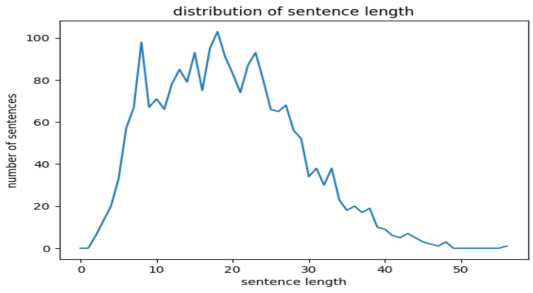

Figure 3: Histogram of sentence length in our data set

Model	Embedding Input	Embedding Initialization	Hidden Layers Size	Activation Function
BoW	5	Random	-	-
CBoW	300	Random	-	-
Deep CBoW	300	Random	2x 100	2x Tanh
PT Deep CBoW	300	Pretrained	2x 100	2x Tanh
LSTM	300	Pretrained	168	Tanh & Sigmoid
Mini Batch LSTM	300	Pretrained	168	Tanh & Sigmoid
Tree LSTM	300	Pretrained	150	Tanh & Sigmoid

Table 1: Over all archtecture of the models

Model	Number of Iterations	Batch Size	Evaluation Frequency
BOW	30000	1	250
CBOW	30000	1	250
Deep CBOW	30000	1	250
PT Deep CBOW	30000	1	250
LSTM	25000	1	250
Mini Batch LSTM	25000	16	250
Tree LSTM	10000	16	250

Table 2: Hyper parameters used in different models

model	bow	cbow	deep cbow	pt deep cbow	lstm	mini lstm	tree lstm	lstm shuffled	tree lstm shuffled
bow	1,017	e-5	e-8	e-25	e-31	e-43	e-27	e-27	e-29
cbow	S	1,017	0,14	e-10	e-14	e-6	e-12	e-11	e-12
deep cbow	S	N S	1,017	e-6	e-9	e-10	e-7	e-7	e-08
pt deep cbow	S	S	S	1,017	0,23	e-29	0,65	0,71	0,45
lstm	S	S	S	N S	1,017	e-35	0,46	0,41	0,67
mini lstm	S	S	S	S	S	1,017	e-31	e-31	e-33
tree lstm	S	S	S	N S	N S	S	1,017	0,94	0,78
lstm shuffled	S	S	S	N S	N S	S	N S	1,017	0,71
tree lstm shuffled	S	S	S	N S	N S	S	N S	N S	1,017

Table 3: Statistical Significance between models. *S* stands for Significant difference between the performance of two models and *N S* stands for Not Significant difference.

References

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. Curran Associates Inc., USA, NIPS'13, pages 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch .

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 *EMNLP*.

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

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, ICML'15, pages 1604–1612. http://dl.acm.org/citation.cfm?id=3045118.3045289.