Random Decision Syntax Trees at SemEval-2018 Task 3: LSTMs and Sentiment Scores for Irony Detection

Aidan San
University of Illinois at Urbana-Champaign
asan2@illinois.edu

Abstract

We propose a Long Short Term Memory Neural Network model for irony detection in tweets in this paper. Our model is trained using word embeddings and emoji embeddings. We show that adding sentiment scores to our model improves the F1 score of our baseline LSTM by approximately .012, and therefore show that high-level features can be used to improve word embeddings in certain Natural Language Processing applications. Our model ranks 24/43 for binary classification and 5/31 for multiclass classification. We make our model easily accessible to the research community[1].

1 Introduction

Recently, irony detection has become an increasingly important problem with new applications appearing every day. In discourse analysis, it is extremely important to understand if a politician is making an ironic or a literal response, or understanding can be completely lost. In chatbots, if a chatbot misinterprets an unhappy sarcastic comment from a customer, the customer could become even more frustrated. In sentiment analysis, if irony is not taken into account, the actual sentiment could be the opposite of the prediction.

The aim of irony detection is generally a binary classfication problem: Is the piece of text ironic or literal? In SemEval2018 Task 3 (Van Hee et al., 2018), there are two subtasks. First, subtask A, is our standard binary classification problem. We are provided with a corpus of tweets annotated with 0 or 1's to specify if a tweet is ironic or literal. Second, in subtask B, we are tasked with a more challenging problem. In subtask B, participants must determine which type of irony a particular tweet contains. Is it verbal irony based on polarity, another type of verbal irony, situational irony or not ironic at all?

Recently, research has shown that neural approaches are particularly effective in sarcasm detection (Ghosh and Veale, 2016) and similar problems such as sentiment analysis (Rosenthal et al., 2017). In this paper, we present neural network model designed to tackle the challenge of irony detection.

In subsection 2.1, we give a brief overview of the entire model. In subsection 2.2 and 2.3, we describe our neural architecture and approach. In subsection 2.5 and 2.6, we describe the embeddings and sentiment scores that are used as the input to our neural network. In section 3, we describe our results, and give quantitative evidence of the effectiveness of our features. In section 4 we conclude, and in section 5, we describe ways that our approach could be improved.

2 System Description

2.1 Overview

Our system is composed of three stages: Preprocessing, Feature Creation, and the Neural Network. Preprocessing is composed of tokenizing the input text. Feature creation is generating the word and emoji embeddings and sentiment scores and concatenating them together to form a feature vector. Finally in the third stage, the feature vector is fed into the neural network which makes a prediction. Our full model is illustrated in Figure 1.

2.2 Long Short Term Memory

In traditional Feed Forward Neural Networks, there is not a good way to represent temporal inputs. Recurrent Neural Network (RNN) cells are like traditional neural network cells, except they have the key difference of feeding their output

[1] github.com/Muakasan/RDST-semeval2018-task3

Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), pages 560–564
New Orleans, Louisiana, June 5–6, 2018.

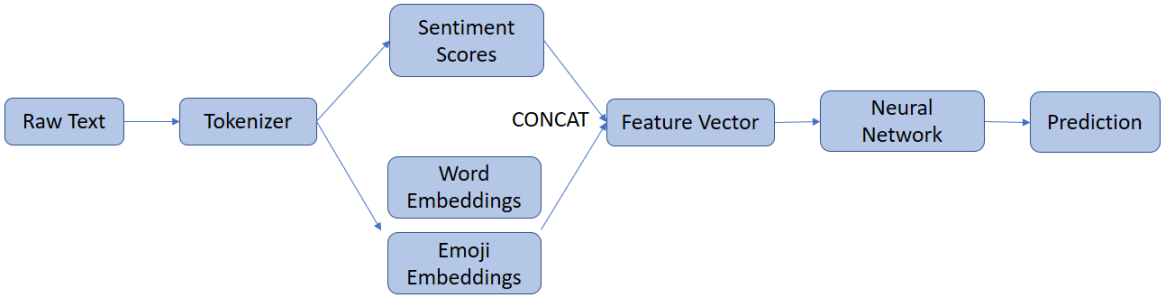

Figure 1: Overview of the entire model.

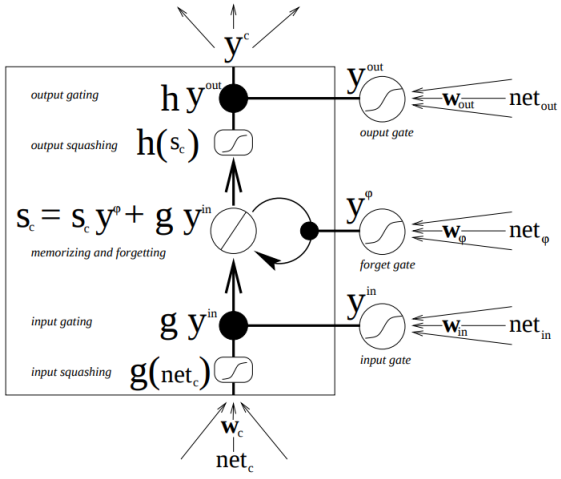

Figure 2: The various gates of the LSTM Cell and the connections between them. The data path is shown from the input $\mathbf{w_c}$ to the output $\mathbf{y^c}$. Taken from Gers et al. (1999).

back into themselves. Each time a new word representation is passed into the RNN, the RNN will take its output and pass it back into the RNN, so the RNN receives 2 inputs (the current word and the output from the previous iteration). The output of the previous iteration can be thought of as a representation of the previous words in the sentence. This is repeated until the sentence is finished. Additionally, context units can be added to the cell to imitate the idea of memory (Elman, 1990). This improves language modeling, because when reading a sentence, prior context of previous words is very important for the understanding of the current word. When we use an RNN, our model can be fed in with context from previous words.

The Long Short Term Memory Model (LSTM) improves over the naive RNN, with the addition of a memory cell and input and output gates (Hochreiter and Schmidhuber, 1997). In the naive RNN, it is easy to lose important information over many iterations of the cell. In an LSTM cell, the memory cell combats this problem. The original LSTM cell is composed of two gates. The output gate can protect other units from irrelevant information from the current iteration, and the input gate can control which information is passed into the current iteration. Additionally, Gers et al. (1999) include a forget gate which can reset memory once it is no longer relevant. Figure 2 taken from Gers et al. (1999) illustrates the architecture of the cell.

2.3 Neural Network Architecture

Our model is built with the high-level Python neural network library called Keras (Chollet et al., 2015). We use a 5 layer model. Our first layer is the input layer, where we feed integer ranks (a number representing how often a word appears in the corpus where "1" is the most common) of

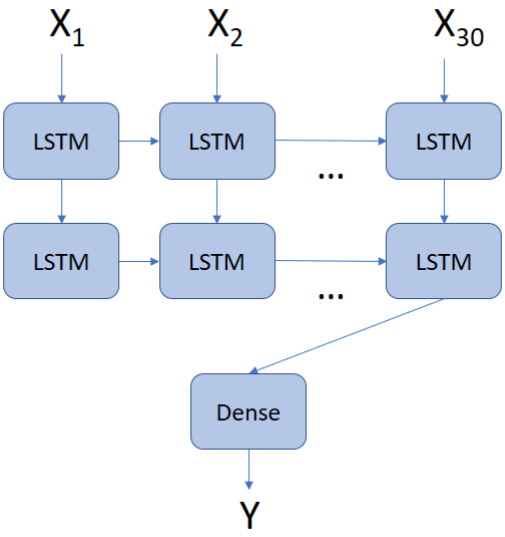

Figure 3: An unrolled Neural Network. X_1 to X_{30} represent the first 30 words in the input tweet. Y represents the irony prediction of the given tweet.

words into the layer. Then we pass the inputs into an embedding layer. This layer takes our inputs and converts them into a word or emoji embedding representation. Our third and fourth layers are both Long Short Term Memory layers. We take the outputs of the first LSTM layer and feed them into our second layer. Both LSTM layers are given a dropout of .25 (chosen by experimentation) to prevent overfitting (Srivastava et al., 2014). The output of the second LSTM is then passed into a single Dense layer. Our Dense layer has a sigmoid and softmax activation for subtask A and B respectively. We use binary cross entropy and categorical cross entropy for subtask A and B respectively. We train our model over 30 epochs (approximately when the model converged) with a batch size of 32. Our architecture is shown in Figure 3.

2.4 Preprocessing

We preprocess using Pandas (McKinney, 2010) for csv reading along with basic data processing. We then run our tweets through the Keras Tokenizer (Chollet et al., 2015). The Keras Tokenizer handles splitting, removing punctuation and lowercasing words.

2.5 Word/Emoji Embeddings

When dealing with a relatively small training set, such as a small number of tweets, pretrained word embeddings can be a very effective way of encoding additional information into the model without needing additional training data. We used the Google News pretrained Word2Vec (Mikolov et al., 2013) word embeddings for our model. The word embeddings were trained using 100 billion words taken from the Google News dataset. We additionally added Emoji2Vec (Eisner et al., 2016) embeddings. As is shown by Hogenboom et al. (2013), emoticons encode a large amount of sentiment information. As the descendants of emoticons, we hypothesize emojis can encode sentiment information as well.

2.6 Sentiment Scores

We use NLTK (Bird and Loper, 2004) to get sentiment scores using the SentiWordNet (Esuli and Sebastiani, 2007) corpus. SentiWordNet is a library which uses a semi-supervised approach to give synsets (groupings of words which have the same meaning) an objective, positive, and negative sentiment score. Riloff et al. (2013) show that sentiment can be an effective way to improve sarcasm detection in tweets. For simplicity, we always use the first synset of a particular word. We take the positive and negative scores from SentiWordNet and then concatenate the results with our word/emoji embedding.

2.7 Memory Constraints

Standard practice is to pass the word embedding matrix with every word as a layer into the Neural Network. In this task, we had the additional constraint of a relatively small amount of RAM (about 8GB). To resolve this problem, instead of building our embedding matrix and then tokenizing our dataset, we tokenized first. Then, rather than building our embedding matrix with every possible entry, we could simply look through the words that we had seen during tokenization, and only put those vectors into our matrix. To put this into context, while we were storing all the embeddings in memory we were using about 6GB of RAM. After we created our much smaller embedding matrix, we were only using about 3GB of RAM.

3 Results

Task	F1	Accuracy
A (Binary)	.5822(24)	.6173(21)
B (Multiclass)	.4352(6)	.6327(6)

Table 1: Official results from the competition on the test set. The numbers between parentheses indicate ranking compared to other models.

As shown in Table 1, in the binary classification task, we achieved an F1 score of .5822 and an accuracy of .6173, and in the multiclass classification task, we achieved an F1 score of .4352 and an accuracy of .6327. This earned us a rank of 24 and 6 in tasks A, and B respectively, because teams were ranked using F1 score.

Model	F1	Accuracy
Without Emoji	.6127	**.6121**
Without Sentiment	.6093	.6097
Emoji & Sentiment	**.6210**	.6091

Table 2: Accuracy and F1 scores of various models tested on a 75%-25% split of the training data.

To test the effectiveness of the emoji embeddings feature and sentiment scores feature, we tested our binary classification model without each feature separately and compared these models to our combined model. We ran our tests over the training set with a 75%-25% split, where 75% of the training data was used to train the model and 25% of the model was designated as validation data, to verify the effectiveness of the model on unseen data. We ran 10 trials and took the average accuracy and F1 score over the 10 trials for each of the 3 models.

The task was ranked using F1 score, so we optimized for F1 score. As can be seen in Table 2, the combined model achieves an F1 score .0083 higher than not including emoji embeddings and an F1 score .0117 higher than not including sentiment scores. Interestingly, we see that the combined model actually achieves a lower accuracy score than both of the individual models. We can draw the conclusion that the combined model has more balanced predictions between the two classes, which overall creates a higher F1 score at the cost of lower accuracy in one of the classes.

4 Conclusion

In this paper, we have described our system for SemEval-2018 Task 3. We discussed LSTM-based neural models, and how we incorporate sentiment features and emoji embeddings. We show that additional high-level features such as sentiment scores can improve neural based models. We achieve rank 24/43 in subtask A and 6/31 in subtask B.

5 Future Work

Possible improvements to the model fall under two primary categories: the neural architecture and additional high-level features.

To implement our neural architecture, we experimented with other types of layers such as Bidirectional LSTMs and CNNs, but we would like to further explore these approaches in the future. We especially think Attention Layers used by models such as Baziotis et al. (2017) could be particularly effective in irony detection, because of their effectiveness in sentiment analysis. We could also try different activation functions in different layers and test out different batch sizes. Another possible direction would be to use ensemble methods which have been shown to be particularly effective by Cliche (2017) for similar tasks.

Regarding additional high-level features, we could also include sentence-level features. Goel et al. (2017) showed that high-level features could be included in a Feed Forward Neural Network which improved accuracy in an ensemble method for the task of emotion detection. Additionally, we could continue in the vein of the model we created and add other word-level features such as capitalization. To improve our sentiment features, we could also use a more advanced method of choosing the correct synset for a better fitting sentiment score.

Acknowledgments

We would like to thank the task organizers for providing the dataset and putting together the competition. We would like to thank Assma Boughoula and Julia Hockenmaier for their help in this project.

References

Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. 2017. Datastories at semeval-2017 task

4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754.

Steven Bird and Edward Loper. 2004. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

François Chollet et al. 2015. Keras. `https://github.com/fchollet/keras`.

Mathieu Cliche. 2017. Bb_twtr at semeval-2017 task 4: Twitter sentiment analysis with cnns and lstms. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 573–580.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. 2016. emoji2vec: Learning emoji representations from their description. In *Conference on Empirical Methods in Natural Language Processing*, page 48.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Andrea Esuli and Fabrizio Sebastiani. 2007. Sentiwordnet: a high-coverage lexical resource for opinion mining. *Evaluation*, pages 1–26.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with lstm.

Aniruddha Ghosh and Tony Veale. 2016. Fracking sarcasm using neural network. In *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 161–169.

Pranav Goel, Devang Kulshreshtha, Prayas Jain, and Kaushal Kumar Shukla. 2017. Prayas at emoint 2017: An ensemble of deep neural architectures for emotion intensity prediction in tweets. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 58–65.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479.

Alexander Hogenboom, Daniella Bal, Flavius Frasincar, Malissa Bal, Franciska de Jong, and Uzay Kaymak. 2013. Exploiting emoticons in sentiment analysis. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 703–710. ACM.

Wes McKinney. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. 2013. Sarcasm as contrast between a positive sentiment and negative situation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714.

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Cynthia Van Hee, Els Lefever, and Véronique Hoste. 2018. SemEval-2018 Task 3: Irony Detection in English Tweets. In *Proceedings of the 12th International Workshop on Semantic Evaluation*, SemEval-2018, New Orleans, LA, USA. Association for Computational Linguistics.