Context-Based Models for Sarcasm Detection

Stanford CS224N Custom Project

Nicholas Benavides
Department of Computer Science
Stanford University
nbenav@stanford.edu

Angelo Ramos
Department of Computer Science
Stanford University
angelor@stanford.edu

Mentor: Rohan Sampath | External Collaborators: N/A | Sharing Project: No | Grading Option: 3

Abstract

Detecting sarcasm is one of the most essential mechanisms in understanding casual language and discourse, considering that a misunderstanding of sarcasm leads to a fundamentally opposite takeaway from a piece of text. Previous work on sarcasm detection has identified the benefit of using contextual information about a comment, its author, and/or the forum it is shared in to build more accurate models. In a previous course project, we identified the benefit of including context in the embeddings via ELMo embeddings and in the model via LSTMs. In this project, we extend our previous work and assess additional ways that textual context can be incorporated into our sarcasm detection models. We extended our previous models with additional LSTM/GRU layers, an extra linear layer, and dot-product attention. Our findings indicate that attention mechanisms in conjunction with contextual embeddings like ELMo can improve model performance without substantially increasing training time. Compared to our baseline models, we saw gains with attention when training on at least 50,000 examples, meaning that these more complex architectures are only beneficial with a relatively large dataset.

1 Introduction

Sarcasm detection is a difficult problem due to the highly contextual nature of sarcastic statements. Detecting sarcasm is also challenging for humans, since it is very difficult to identify whether a comment is sarcastic without context about the author and the circumstances in which the remark was made. The increased popularity of social media platforms, online forums, blogs, and similar channels have led to an increase in informal and sarcastic text across the internet. The ability to detect sarcasm more accurately is critical to understanding and enabling sentiment analysis for text from these sources. Due to the difficulty of detecting sarcasm from text alone, much of the previous work in this area has focused on leveraging background information about the author [1] or forum [2] as well as novel architectures to better detect the various contextual dependencies and latent meaning.

We hypothesize that, given more recent developments in NLP such as deep contextualized word embeddings and attention mechanisms, there are still gains to be realized from incorporating additional ways to capture context from text into sarcasm detection models. Based on both intuition and prior work, we know that understanding context is critical to accurately detecting sarcasm. In some longer sentences or comments, it's possible that the sarcastic portion of the remark appears at the end as a comedic punchline. For example, many Reddit comments begin with phrases that could be perceived as non-sarcastic before ending with a phrase that is clearly sarcastic. The sarcastic comment *"if there's one thing in this world that merits truly serious levels of obsession to the point of sending anonymous bullying messages to people online whenever they disagree with you about it, it's **multiplayer videogames**"* is sarcastic because of the heavy emphasis on multiplayer video games at the end. In other cases, the ordering of several words can change the likelihood of whether a comment is sarcastic or not. For example, the sarcastic Reddit comment *"that's totally not going to look like a*

beard in 15 years" sounds much less sarcastic when rephrased as *"that's not going to totally look like a beard in 15 years."*

While some research has found success in incorporating context via information about a comment's author or forum a comment is shared in, these approaches do not fully utilize the contextual information they can draw from the text itself. In this paper, we experiment with various ways to capture additional context from the text through multiple LSTM/GRU layers, an additional layer, and dot-product attention. We also train our models on training sets of various size to understand how much data is required to realize the gains of these more complex neural architectures. Our results show that models utilizing attention performed the best and outperformed our baseline models, but these gains were only realized when the training set size surpassed 50,000 examples.

2 Related Work

Previous sarcasm detection studies tend to trend similarly in their focus and in their tasks. Kolchinski and Potts [1], Hazarika et al. [2], and Ilic et al. [3] all achieve state-of-the-art or near state-of-the-art results. However, their approaches do have some key differences that are worth noting. Kolchinski and Hazarika et al.'s papers use very different architectures and features to address the same task. First, Hazarika et al.'s CASCADE model is built on a CNN, while both of Kolchinski's models extend a bidirectional RNN with GRU units. These models capture context either by convolving over groups of words or by the sequential nature and memory of the GRU units. Both Hazarika et al. and Kolchinski are focused on examining the role of background information in sarcasm detection. Hazarika et al. uses a wealth of contextual information, namely user embeddings and forum embeddings, to learn more complicated relationships about authors' uses of sarcasm in various situations. In contrast, Kolchinski takes a much simpler approach, using either a Bayesian prior to model an author's propensity for sarcasm or an embedding vector to encode contextual information about the author. Despite not using forum embeddings, Kolchinski's model performs comparably but slightly worse than Hazarika's model, suggesting that Hazarika's additional embeddings yield only incremental gains. Despite the gains that author and forum embeddings yield, these approaches suffer from the cold-start problem, where either an author or a forum is new, and the model has very little or no data about their tendencies.

In contrast, Ilic et al.'s model differs significantly from both Kolchinski and Hazarika et al.'s models because she does not encode any information about the author or forum. Instead, she takes a character-based approach that utilizes character-level vector embeddings for words through ELMo. However, Ilic et al.'s architecture does have some similarities with Hazarika et al. and Kolchinski. Like Hazarika et al., Ilic et al. relies on CNNs to generate the contextualized embeddings, although her model combines the character-level representations into a word-level representation. Similar to Kolchinski's model, Ilic et al.'s model passes the embeddings to a bidirectional RNN, although she uses LSTM units instead of GRU units. Comparing the performances of these three models on the SARC dataset, it appears inconclusive whether Ilic et al.'s character-based approach is superior to the author-focused approaches. Although Ilic et al. demonstrated the power of ELMo embeddings with a character-level model, previous work has not explored ELMo embeddings with a word-level model, and no work has examined the effects of combining deep contextualized embeddings with attention for sarcasm detection.

More recently, Peters et al. [4] developed a new form of contextual word embeddings called Embeddings from Language Models, commonly known as ELMo. ELMo is a deep, contextualized method to represent words that models both the complex aspects of a word's usage and how a word's uses vary in different linguistic contexts. The ELMo word vectors are learned functions derived from a deep bidirectional LSTM (BiLSTM) language model trained on a large corpus. Unlike other embedding methods, each token's representation is determined by the whole input sentence, which allows ELMo embeddings to capture more information about the context the token was used in. In addition, the model makes use of sub-word information through the use of character convolutions. ELMo embeddings are not only unique, but they also have been found to improve performance on a variety of natural language understanding tasks, and we think that using these embeddings in conjunction with other methods of capturing context can lead to performance gains.

3 Approach

3.1 ELMo Embeddings

To generate ELMo sentence embeddings, we pass tokenized sentences into AllenNLP's ElmoEmbedder [5], which for each word outputs a 1024-dimensional embedding at each of the 3 model layers of the ELMo model. We then take the embedding from the last layer for each word, which contains the most contextual information, and average the word embeddings from a sentence to obtain a sentence embedding. This approach differs from our previous course project, where we summed word embeddings to generate a sentence embedding. We chose to average the word embeddings as opposed to summing to account for differing sentence lengths. Summing word embeddings would introduce high variance in our sentence embeddings, making it harder for our model to properly learn weights. We then concatenate a pair of sentence embeddings and pass this 2048-dimensional input to all of our models. We wrote the code to embed tokenized sentences with ELMo, transform the word embeddings into a sentence embedding, and save the results for later use. The code for processing the raw text comments into a formatted dataset that we fed into the ElmoEmbedder came from this repository.

3.2 Baselines

For our baseline models, we trained one model for each type of RNN cell, LSTM [6] and GRU[7]. The baseline models contain a single bidirectional LSTM or GRU layer followed by a linear layer, which outputs a tensor of length 2 for each example. The predicted class for each example corresponds to the index of the output containing the higher value. The baseline models were inspired by the BiGRU model in [1] and our previous course project. The baseline model architectures and training loop were inspired by this repository.

Refer to the "Baseline BiLSTM" diagram in Figure 1 on the following page for a diagram of the baseline model architecture. The embedding dimension of 4 in the diagram is just for simplicity of the figure, and all of our models have a 2048-dimensional input. Similarly, the number of LSTM units and the dimensionality of the LSTM representations do not reflect the dimensionality of our actual models and are chosen to simplify the diagrams.

3.3 Multiple RNN Layers

Our baseline models make use of a single RNN (LSTM/GRU) layer. In general, RNN layers are useful in capturing context due to their sequential nature. Through stacking multiple RNN layers, our models should learn a greater number of parameters and capture additional context in comparison to the models that only use a single RNN layer. We extended code from our previous course project to implement these models. Refer to the "Multiple RNN Layers" diagram in Figure 1 on the following page for a diagram of the model architecture.

3.4 Additional Linear Layer

Our baseline models use only a single linear layer after the RNN layer. In doing so, the baseline models pass the output from the RNN (LSTM/GRU) directly to the linear output layer that determines the final classification. By adding an additional linear layer in between the RNN and the linear output layer, we hope to capture more information as we transform the RNN outputs to a classification as well as enable larger RNN hidden sizes. We extended code from our previous course project to implement these models. Refer to the "Additional Linear Layer" diagram in Figure 1 on the following page for a diagram of the model architecture.

3.5 Attention Layer

With a heavily context-dependent task like sarcasm detection, certain segments of an input sentence may be more relevant to determine sarcasm than other parts of the input. Adding attention to our models allows them to learn these relationships and to place more weight on the most critical portions of the input. For our models, we implement dot-product attention[8], which learns weights for different positions of the input and computes the dot product of the weights with the LSTM/GRU

output. The attention layer learns a weight vector α. It takes as input a hidden RNN state, which we will denote as h_i, computes the dot product of the weights with the input $e_i = \alpha^T h_i$ for each h_i, passes the resulting attention scores $e_1, ..., e_n$ where n denotes the number of hidden states, through a softmax function to normalize their values, and then updates the vector α via backpropagation. We extended code from our previous course project to implement these models, and we modified this code to implement the attention layer. Refer to "Dot Product Attention" diagram in Figure 1 below for a diagram of the model architecture.

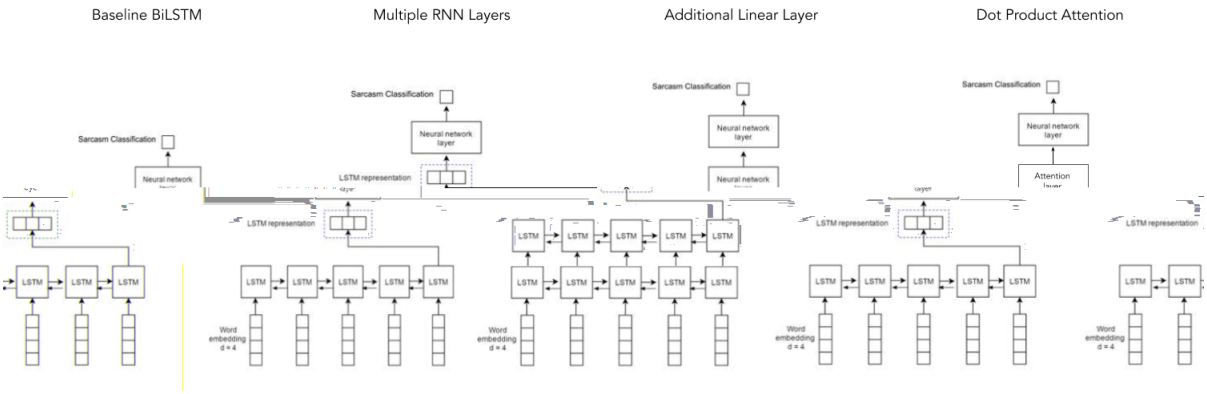

Figure 1: Model Architectures

3.6 Other Modeling Information

For all of our models, we used binary cross-entropy loss as our loss function, given by the formula $-\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i))$, where y_i is the true label for an example, $p(y_i)$ is the predicted label, and N is the total number of examples. We chose the Adam optimizer [9] and used a batch size of 32 for all of our models.

We employed dropout at various locations in our models, but we included at most 1 dropout layer in any particular model. We did not use dropout with our baseline models. For the models with multiple RNN layers, the dropout layer sat in between the first and second RNN layers. For the models with an extra linear layer, the dropout layer followed the extra linear layer. For models with attention, the dropout layer followed the attention layer.

In terms of code, we modified the training script from our previous project to handle multiple model architectures, and we improved the code reusability and readability by decomposing the train/test split and error analysis portions of the training script into a utils file.

4 Experiments

4.1 Data

We used the Self-Annotated Reddit Corpus (SARC) dataset [10], a corpus of 1.3 million sarcastic statements. More specifically, we used the balanced version of the r/main dataset, which contains 257,082 comments from a variety of subreddits (forums), half of which are sarcastic. The SARC dataset is constructed such that each example in the balanced dataset contains two comments, one that is sarcastic and one that is non-sarcastic. Thus, the input to our models are concatenations of two sentence embeddings, and the output is a binary value where a 1 represents that the first sentence is the sarcastic one, and a 0 represents that the first sentence is the non-sarcastic one. We decided to keep the input as a pair of sentences so that we could directly compare our models with others that have used the SARC dataset for sarcasm detection.

4.2 Evaluation method

To evaluate our models, we used the F1 score, which combines both precision and recall. Similar to other sarcasm detection papers, we compute a macro-averaged F1 score. To compute the macro-averaged F1 score, we first compute the F1 score for both the positive and negative classes of examples. From there, the macro-averaged F1 score is the average of the 2 F1 scores.

4.3 Experimental details

For each of the model types, we trained 25 models with different combinations of hyperparameters. For each of the baseline models, we conducted a single training run of 25 models with no dropout. For each of the challenger models, we conducted 2 training runs, 1 with no dropout and 1 with a dropout probability of 0.2. The hyperparameters varied were GRU/LSTM hidden sizes, the number of GRU/LSTM layers, the extra linear layer size (if applicable), the number of epochs, and the learning rate.

The number of LSTM/GRU layers was fixed for a given model type and the dropout parameter was fixed for a given training run, meaning that there were 36 possible combinations of the remaining hyperparameters for models that did not include the extra linear layer. By training 25 models, we were able to explore the majority of our space of hyperparameters. The training time for runs of models that did not include the extra linear layer was approximately 4 hours on the Azure NV6 virtual machine.

For the models with an extra linear layer, we also fixed the learning rate to be 0.001 since that had been optimal with the majority of the previous models and reduced the number of hyperparameter combinations from 144 to 48, which still allowed us to explore a majority of the space with 25 models. To select the size of the extra linear layer, we only sampled from sizes that were smaller than the LTSM/GRU hidden size, since we generally want the dimensionality of deeper layers to be smaller than previous layers. The training time for runs of these models took approximately 6 hours on the Azure NV6 virtual machine.

Once we determined which models outperformed our baselines, we ran another set of experiments with the best models and our baseline models to assess the amount of training data required to realize the gains from our best models. Using the optimal hyperparameters for each model type, we trained each model on 5k, 10k, 25k, 50k, and 100k examples and recorded the macro-averaged F1 score on the dev set.

5 Results

The results of our first set of experiments are shown in Table 1 below:

Table 1: Best Models of Each Type

	Dev F1 Score	RNN Hidden Size	Linear Hidden Size	RNN Layers	Epochs	Learning Rate	Dropout
Baseline LSTM	0.723	256	N/A	1	20	0.0001	0
Baseline GRU	0.715	128	N/A	1	10	0.001	0
2-Layer LSTM	0.717	256	N/A	2	10	0.001	0.2
2-Layer GRU	0.707	512	N/A	2	10	0.001	0.2
LSTM w/ extra Linear Layer	0.715	512	128	1	10	0.001	0.2
GRU w/ extra Linear Layer	0.723	512	128	1	10	0.001	0.2
LSTM w/ Attention	0.735	256	N/A	1	10	0.0001	0.2
GRU w/ Attention	0.737	512	N/A	1	20	0.0001	0

5.1 Sentence Embedding Method

Retraining our baseline models, which were the best models from our previous project, with our new ELMo embedding approach allowed us to compare the performance of summing word embeddings with averaging word embeddings. In our previous work, we obtained an F1 score of 0.689 with the baseline LSTM model. After retraining, both the LSTM and GRU baseline models significantly outperformed our previous model, indicating that averaging word vectors to generate a sentence embedding is a superior approach to summing word vectors. We thought that the new approach would perform better, but the difference here was greater than we expected.

5.2 LSTM vs GRU

Based on our results, it's inconclusive whether LTSM or GRU units are better for sarcasm detection. LSTM outperformed GRU in the baseline models and 2-layer RNN models, but performed worse than GRU on the models with an extra linear layer and was comprabable on the attention models. We expected that the LSTM models would perform slightly better than GRU since its additional gate allows it to save more contextual information between states, but our results indicate that the performance difference between the two is not significant.

5.3 Multiple LSTM/GRU Layers

The models with 2 RNN layers performed worse than the baseline models, which is worse than we expected. Given these results, We suspect that this architecture may be too complex for this task or that we don't have enough data for the second RNN layer to learn useful representations of the comment pairs.

5.4 Extra Linear Layer

For the models with the extra linear layer, the GRU model outperformed the GRU baseline, but the LSTM model performed worse than the LSTM baseline. We expected that the extra linear layer would help to transform the RNN outputs to the classification and potentially enable a larger RNN hidden size. The extra linear layer did enable a larger RNN hidden size for both the LSTM and GRU models, but the performance gains were only observed for GRU. Thus, these results show that the additional layer can improve model performance but likely isn't the best solution.

5.5 Attention

The models with attention performed the best, and they significantly outperformed both baseline models. We expected that attention would best capture context because it would learn what parts of the input are most important, so these results are not surprising. Based on our results, attention seems like a promising method to extract more contextual information from text for sarcasm detection.

5.6 Training Set Size

Table 2 below displays the results of our second set of experiments, where we trained our baseline models and best models on varying amounts of training data.

Table 2: Best Models Baselines - F1 by Number of Training Examples

	Dev F1 (Full)	Dev F1 (100k)	Dev F1 (50k)	Dev F1 (25k)	Dev F1 (10k)	Dev F1 (5k)
Baseline LSTM	0.723	0.722	0.703	0.699	0.678	0.654
Baseline GRU	0.715	0.705	0.689	0.672	0.666	0.662
LSTM w/ Attention	0.735	0.721	0.709	0.693	0.679	0.650
GRU w/ Attention	0.737	0.724	0.706	0.689	0.680	0.649

These results were more or less what we expected. We see that all of the models improve as they are trained on more data. At lower training set sizes, the differences between the baseline models

and the attention models are negligible. With the LSTM models, we start to see gains from attention as the training set size approaches the full dataset. For GRU, we see substantial improvements in performance over both baseline models as the training set surpasses 50,000 examples.

6 Analysis

6.1 LSTM vs GRU

In our error analysis, it was clear that the LSTM baseline model performed better on longer sentences in comparison to the GRU baseline model, which tended to incorrectly classify longer sentences. This makes sense because LSTM models are designed to capture long-term memory, which allows the model to capture context between words that are positioned far apart within the same sentence. However, GRU models are less performant in capturing long-term context within a sentence, leading to declining performance as sentence length increases. For example, one comment that GRU incorrectly classified as sarcastic was *"i like how each of the teams have a trophy from their previous round cavs have a piston on the sword hawks are carrying a celtics hat..."* The total sentence length of this incorrectly classified comment was 70 words, demonstrating how the GRU baseline model struggles to correctly classify longer sentences. To further investigate this difference in performance, we randomly selected 50 comments that were incorrectly classified as non-sarcastic by each of our baseline models and computed the average word lengths of these comments. The average sentence length across these 50 incorrectly classified comments for the GRU baseline model was 15.8, as opposed to a 9.9 average sentence length for the LSTM baseline model. Based on these average lengths, we can also conclude that the GRU models perform better on shorter sentences, which makes sense given a lesser need for long-term memory on these examples.

6.2 Multiple LSTM/GRU Layers

Examining errors produced by the model with multiple GRU layers, we find that a common failure mode is when both the sarcastic and non-sarcastic comments have fewer than 7 words. Examining 50 misclassified examples, 11, or 22%, have this property. This compares to 6 out of 50 randomly selected errors, or 12%, from the baseline GRU model. Similarly, we estimate that 20% of errors from the model with multiple LSTM layers and 14% of errors from the baseline LSTM model have this characteristic, each of which were based on a random sample of 50 errors. One such example of this includes the non-sarcastic comment *"i'm tony, tony, tony, tony"* and the sarcastic comment *"not photoshopped at all"*. While a human would likely predict the second comment to be sarcastic given it's reference to photoshop, it appears that the model fails to distinguish between the two comments due to their short length. It makes sense that models with stacked RNN layers would have more errors of this type because these pairs of comments do not have much context to extract, making the model overcomplicated for these kinds of examples.

6.3 Extra Linear Layer

In our error analysis of our models that used an extra linear layer between the RNN layer and the linear output layer, we noticed that a common type of error is when the difference in sentence length between the sarcastic comment and the non-sarcastic comment is 20 words or more. For the LSTM model with an additional linear layer, we found that 7 out of 50 (14%) incorrectly classified examples had a sentence length difference of at least 20 words. In comparison, our baseline LSTM model only had 1 out of 50 (2%) incorrectly classified examples exhibited this sentence difference property. Furthermore, we also found that among the 7 misclassified examples with a sentence length difference of 20 words or more, the LSTM model with an additional linear layer tended to classify the longer comment as the sarcastic comment. One such example of this error type is the comment *"i totally understand what they are saying!*, which has a sentence length of 7. Although the comment would be likely classified as sarcastic by a human, the LSTM model with an extra linear layer instead classifies the corresponding non-sarcastic comment, which had a significantly longer sentence length of 29, as sarcastic. This classification behavior suggests that the extra linear layer is learning weights that skew the classification of longer sentences towards a sarcastic classification.

6.4 Attention

By generating confusion matrices for both of our attention models and baseline models, we gained a better understanding of how the models were classifying the data compared to the ground truth. The LSTM baseline predicts a majority of examples to be sarcastic, resulting in higher true positives and higher false positives. On the other hand, the GRU baseline predicts a majority of examples to be non-sarcastic, leading to higher true negatives and higher false negatives. Both attention models had better balance in their predictions, which ultimately led to better performance. The LSTM model with attention had a slight bias for false negatives over false positives, and the GRU model with attention had a slight bias for false positives over false negatives.

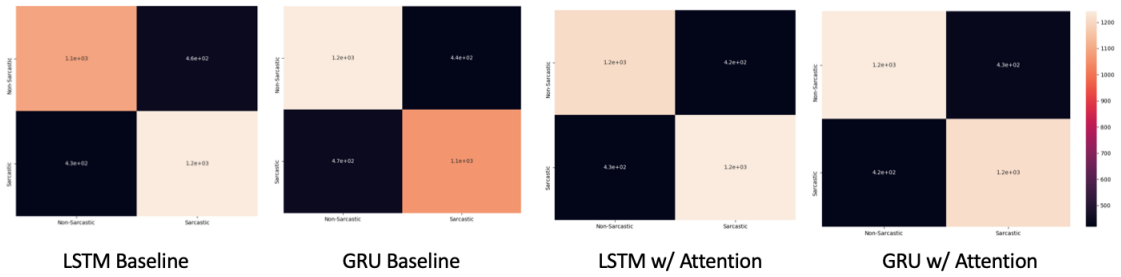

Figure 2: Confusion Matrices - Attention vs Baseline (x-axis = predicted label, y-axis = true label)

We also found that models with attention did seem to outperform our baseline models for examples where the sarcastic sentiment is largely concentrated within one portion of the sentence. One such example is the sarcastic comment *"lefties seem to forget that the rest of us don't believe in affirmative action and actually hire based on merit, not quotas or political correctness."* In this sentence, the sarcasm is defined largely by the beginning phrase of the sentence *"lefties seem to forget that the rest of us..."*, while the rest of the sentence can be interpreted as relatively serious and non-sarcastic. The GRU baseline model incorrectly classified this comment as non-sarcastic, whereas the GRU model with dot-product attention was able to correctly identify the comment as sarcastic. With attention, the model learned to weight the beginning segments of a sentence more heavily when searching for sarcasm, which allowed the attention-based model to successfully classify this sentence even though sarcasm was only expressed at the beginning of the sentence.

7 Conclusion

In this project, we evaluated several approaches to extract additional context from text for sarcasm detection, namely multiple RNN layers, an additional layer, and dot-product attention. We found that the models with multiple RNN layers performed worse than our baseline models, models with an additional linear layer outperformed the baseline for GRU units but not for LSTM units, and that models with dot-product attention significantly outperformed the baseline models. We also concluded that the gains produced by the attention models were only observed as the training set size surpassed 50,000 examples. Our error analysis supported many of our original hypotheses and identified common failure modes for models that underperformed.

This project has several limitations. First, we don't know if the gains we received from attention would be observed if we applied attention to the models that also use author or forum embeddings. Second, the self-labeling of the dataset means that some comments labeled as non-sarcastic could indeed by sarcastic, and this seems to be the case based on error analysis. Finally, due to computational constraints, we were not able to conduct as expansive of a hyperparameter search as we would have liked.

To further expand on our work, it would be interesting to explore the possibilities of training ELMo end-to-end, which we didn't attempt due to computation constraints for this project. We would also be curious to try BERT embeddings, since they have performed better than ELMo in some applications. Finally, implementing more complex attention mechanisms such as multiplicative or additive attention to see if they yield even better results could be another interesting direction.

References

[1] Alex Kolchinski and Christopher Potts. Representing social media users for sarcasm detection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[2] Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. CASCADE: contextual sarcasm detection in online discussion forums. *CoRR*, abs/1805.06413, 2018.

[3] Suzana Ilić, Edison Marrese-Taylor, Jorge Balazs, and Yutaka Matsuo. Deep contextualized word representations for detecting sarcasm and irony. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 2–7, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[5] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. AllenNLP: A deep semantic natural language processing platform. In *ACL workshop for NLP Open Source Software*, 2018.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473, September 2014.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. In Proceedings of the 3rd International Conference for Learning Representations, San Diego, 2015.

[10] Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A large self-annotated corpus for sarcasm. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, 2018.