Deep Learning Humor Classification on Yelp reviews

Rohan Bais
Department of Computer Science
Stanford University
Stanford, CA 94305
rbais@stanford.edu

Marcos Torres
Department of Computer Science
Stanford University
Stanford, CA 94305
marcost7@stanford.edu

Abstract

Humor is an abstract, high-level use of language that is largely subjective. Still, are structures and patterns of language that are widely recognized as funny. We attempt to build a model capable of recognizing humor within Yelp reviews. Despite the simple binary labels, we believe that this domain exemplifies real-world, subtle humor, since the "reviews" aren't forced to be "funny" all the time. Using a set of 120000 Yelp reviews, where we used the "funny" votes from other users as an indicator for humor, we trained a baseline shallow neural network model that simply averages the word embeddings to a single vector and feeds it into a neural network and achieved an initial accuracy of 0.602. We were also interested in seeing how omitting punctuation versus keeping the punctuation plays a role in detecting humor, and we found that keeping the punctuation but separating them as separate "embeddings" works better. We also tried to see if tf-idf weighting for our baseline method, where we multiply each vector by a tf-idf weight before adding them, affected the model, and it slightly ameliorated the accuracy. We then tried different models such as CNN, LSTM-RNNs with attention, and LSTM-CNNs to see if it could capture more complexity and it did, with LSTM-CNN proving to be the best model because of it being more complex to be able to capture humor via sarcasm. Lastly, we tried hyperparameter searching on the LSTM-CNN model on learning-rate, batch size, and window sizes to see how to make our model perform the best. All models were competent, but it looked like we needed a complex model to properly fit to the data.

1 Introduction

Classifying natural language text is a popularly studied area, especially in the fields of sentiment analysis and opinion mining. In these fields, data scientists use various statistical methods to classify the emotions or opinions of the author of a piece of text, often based on the positive and negative connotations of words like "happy," "sad," "afraid," and "bored."[1] More difficult, however, is the classification of humor.

Humor is by nature subjective; whether something is deemed funny or not depends on the audience's interpretations, preconceptions and personal taste. Still, there are recognizable patterns that consistently elicit laughs. Words rarely have a humorous connotation from definition alone; it is the contexts and structures in which they are used that make or break a joke. For example, the "rule of three" appears in this joke from comedian Hannibal Buress' 2014 special *Live in Chicago*:

> "New Orleans police has a parades department. There's homicide, there's narcotics, and there's *parades*. There's other departments too, but you know, rule of three for comedy."[2]

On their own, "narcotics" and "homicide" are heavy words, but the "rule of three" and their juxtaposition with "parades" forms a punchline.

Given the underlying structures of comedy, it should be possible to develop a model capable of recognizing the constructs of humor to some degree. To that end, some researchers have tried to apply machine learning to recognize the punchlines of jokes. Some limit their scope to specific joke constructs, such as "that's what she said" jokes or puns.[3][4] Others have used the transcripts of sitcoms like *Big Bang Theory* as training material, relying on the inclusion of canned laugh tracks as positive labels.[5] Whatever the source material, researchers have used Support Vector Machines (SVM), conditional random fields (CRF), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) to recognize humor.[3][4][5][6]

We are interested in humor as it arises in settings where humor *isn't* a designed priority. In scripted sitcoms, comedy routines and puns, laughter is the main focus, and the punchlines are carefully crafted with plenty of forethought and revision. We seek to uncover humor as it arises in mundane settings from average people who don't work in a writers' room, and as qualified by the opinions of a general audience, rather than canned laughs.

For this task, we settle on classifying Yelp reviews as funny or not funny. Yelp reviews are primarily interactions by which users recommend or criticize restaurants to each other, but users are also able to vote reviews as "funny." Using a dataset of 120,000 reviews, we attempt to classify humor by way of upvotes. We use RNNs, CNNs, and a combined LSTM-CNN to try to capture the various structures of humor. While our LSTM-CNN performs best, with an F1 score of 0.711, we find that the subjectivity of humor and the complexity of structures such as sarcasm make it difficult to build a highly predictive model.

2 Related Work

There is a large body of work relating to sentiment analysis.[1] While the ultimate goal of predicting mood or affinity is different from predicting humor, it is still an exercise in classifying text based on word choice and structure. Thus, many of the methods are directly transferable. Of particular interest to us was Pedro M Sosa's work on LSTM-CNN models, which he used for sentiment analysis on consumer tweets.[7] Like the author, we hope that the order-preserving LSTM, combined with the feature selection capabilities of CNNs, will help learn structure.

For elements of our experimental design apart from deep learning models, including decisons on how to parse and tokenize reviews, we look to De Oliveira and Rodrigo.[8] They conduct a similar humor analysis, choosing to include punctuation for its effect on emphasis in comedy.

There are a handful of other similar explorations on humor that we reference. Kiddon and Brun[3] as well as Chen and Soo[4] use short jokes of fairly standardized structures - "that's what she said" jokes and puns. Bertero and Fung[5] analyze transcripts of Big Bang Theory, complete with laugh tokens where canned laughter audio is overlayed in the show. Chen and Lee used the organic laughter arising in TED talks to capture humor in "Predicting Audience's Laughter Using Convolutional Neural Network."[6] We find Chen and Lee most relevant for their use of CNN models, and because they rely on organic audience laughter (as opposed to the joke writer's assumption of laughter).

Of course, we depend also on landmark developments in deep learning and natural language processing, including the invention of the LSTM [9] and developments in word embeddings.[10]

3 Approaches

For our approach, we specifically wanted to explore 4 different models: a shallow neural network (baseline), a CNN on the stacked word-embeddings, an LSTM-RNN with an attention mechanism, and finally a LSTM-CNN.

3.1 Baseline

Our baseline is the basic shallow neural network. This network takes in a 256-dimensional input, which is the desired dimensionality of our word embeddings. To reduce a review of multiple word vectors to a single "encoded" vector, we simply sum all of the word vectors we retrieve from the embedding layer. We expect this loss of order information to be insufficient for classification of humor, and we primarily would like to see how much order-information helps the other models. Our shallow neural network has 2 hidden layers, first with 100 units and second with 150 units each with ReLU activation functions, and an output layer that outputs the probabilities that an example is "not humorous" or 'humorous".

In addition to this, however, we are introducing two variations of this baseline for the sake of the experiment. One of the variations was to completely remove the punctuation by filtering out periods, commas, question marks, exclamation points, apostrophes, semicolons before feeding the words into a embedding model. The reason we wanted to do this is that we noticed a lot of typos where there is a wrongly placed exclamation point or no space between a period and the beginning of the next sentence, which could mess up the embedding matrix and cause too many different variations of the same word to appear because of the small typos, thus increasing our number of parameters and destroying the common vocabulary the train/test set would share. The second variation would be to weigh with tf-idf weighting, where before we add the embeddings together into the single "encoded" vector, we weigh each vector with their respective tf-idf score for that particular review. As the formula shows below, the hope of this technique is to capture and weigh more important word features which would assist in humor detection. The formula is depicted below, with d being a document, D being the set of documents, and N being the number of documents total:

$$tf - idf(w, d) = |w : w \in d| \log\left(\frac{N}{|w : w \in y, y \in D|}\right)$$

3.2 CNN

One of our next models is a vanilla CNN based on review representations as stacked word embeddings. Our model will retrieve a matrix of word embeddings representing a review's original text before proceeding to perform convolutions on this matrix. We will then use convolutions of regions of 2,3,4, and 5 words. The idea here is to capture adjacent words in a "window" and then retrieve a vector that represents the word and it's nearby neighbors via convolutions. In our case we would have 4 vectors, one vector output for each 1-dimensional convolution. We then perform a max-pool on the outputs and concatenate the poolings into a vector, and use this to feed into a fully connected layer that outputs the probability of the 2 classes. Below shows how the feature map is computed using a window size of h, where c_i is the i^{th} feature in the resulting feature map c after the 1-dimensional convolution using a filter w. This model is based off of Chen's initial proposal for CNN, only we're increasing the window size[6]

$$x_{1\,n} = x_1 \oplus x_2 \oplus ..... \oplus x_n \;\; (\oplus\text{is vertical-stacking operation})$$

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

3.3 Attention-based RNN

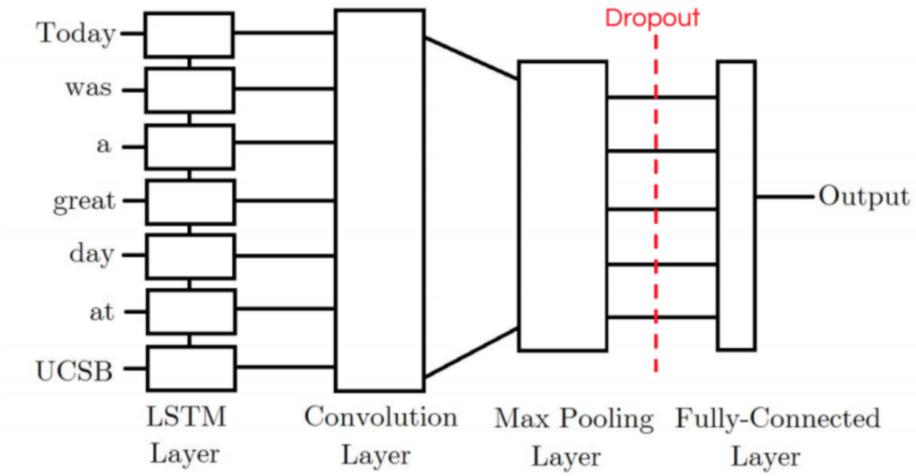
The attention-based LSTM-RNN is yet another model we are interested in. In general, jokes build to a punchline and are built around a specific subject for which they use a handful of key words. We're hoping that this model outperforms the previous two models based on the explicit retention of order-information. We feed the review into an embedding layer and pass each of the embeddings as inputs to the RNN. For each timestep, we are then returned a hidden state. For each of these hidden states we want to calculate attention to see which of the words are more "important", so we feed each hidden state into a multiplicative attention mechanism. The mini-neural-network, probably of 2 layers, will take in the hidden state h_i and output a scalar e_i which indicates the "attention" score for the particular hidden state. Once we gather the e_i for all $1 \leq i \leq n$ we will perform softmax to get the probability vector α. We then use α to weight the particular hidden states based on the corresponding components and sum them together (i.e. $\alpha_1 h_1 + \alpha_2 h_2 .... + \alpha_n h_n$). Finally, we will have a fully

connected layer that takes in this summed vector and outputs a vector of probabilities of the 2 classes. The RNN is similar to what was proposed to Oliviera's model, only attention mechanism was added in hopes to improve the vanilla RNN data fitting. [8]

3.4 LSTM-CNN Combined Model

Finally, we would like to use a combination of the CNN and RNN into the LSTM-CNN model to see if it can provide some complexity that the previous two methods missed by themselves. It is also to see if the CNN's strengths of word window feature representation with RNN's strengths of retention of time-based data can both be captured in this combined model. Below is a diagram of how the model looks.

Figure 1: A combined LSTM-CNN model, based off of Sosa's sentiment analysis model



It will first pass in the review, as embeddings, to the LSTM-RNN and output the hidden-states for each time-step. Then these hidden states are stacked vertically such that the i^{th} row corresponds to the i^{th} hidden-state output. Now we feed this into a CNN and perform 1-dimensional convolutions similar to that of the vanilla CNN described earlier. After the 1d convolutions and max-pooling, we will feed it into a fully connected layer to retrieve the probabilities of the 2 classes. This model was primarily based off of a rendition of Sosa's model for sentiment analysis, we will maybe add the attention mechanism depending on how well this initial model fits.[7]

4 Experiments

4.1 Data

Our data set is comprised of 120,000 yelp reviews, each coming with a text field for the actual review and a "funny" upvote count that provides the number of people who found the particular review funny. Exactly 60,000 reviews are labeled as "funny" and 60,000 are labeled as "not funny", just to make sure our models don't have too much of a skewed fit to one class. Given that we have only the number of upvotes on whether or not a review is funny, and that our task is binary classification on humor, we labeled the "funny" reviews as reviews that at least one upvote on the "people who found this funny" field, and "not funny" as reviews that had 0 in that field. The dataset itself has a wide variation of reviews, from ones on finer Italian cuisine to ones on simple fast-food chains of certain locations, and a lot of the dialect and speaking differs as well. Many reviews straddle between formal and informal plenty of times, with the more formal reviewers generally being from "pros" as labeled in the dataset and the more informal reviews coming from younger and newer people on the website. There is a survey of all of the types of reviews out there, however.

4.2 Evaluation method

Since this is just a binary classification task, for our metrics we used not only accuracy but also precision and recall as well as normalized confusion matrices to properly understand which misclassifications were the weaknesses of the particular algorithm. Because we are dealing with humor, we think it important that inputs marked "humorous" are consistently funny, so we prioritize precision.

4.3 Experimental Details and Results

We initially ran our baseline method with 2 epochs through the training data, a stochastic gradient descent optimizer, and a learning rate of 0.001. However, there were some serious underfitting issues with this initial pass, which made us reformulate these initial configurations. We then tried to increase the number of epochs to 8, but the underfitting wasn't alleviated all that much. We then tried changing the optimizer to Adam with $\beta_1 = 0.9, \beta_2 = 0.99$ and increased the learning rate to 0.1, which helped alleviate some of the underfitting. We first tried these experiments on a 20000 point subset of the training data, just to quickly see how turning the knobs on model configuration affects the performance, before running it on our full dataset.

Since this is a Yelp dataset, and reviews can come across informal, the reviews can be riddled with typos. Part of the task for the project was to find an easy way to pre-process this data so as to not introduce too many embeddings due to exact same words being counted twice due to the spelling differences. Examples include people who did not place spaces probably between punctuation (e.g. "food.I" should turn to "food" and "I"). We introduced various pre-processing rules to parse out the obvious duplicates, such as splitting by "." or "!" in case spaces weren't provided, or splitting by hyphens into vocabulary that was already encountered. Due to the sheer vocabulary size, we filtered out trailing punctuation as well and made all words lower-case. That said, we kept some idiosyncrasies that are important to writing style, such as leaving emphasis on words with extended spelling (e.g. hilarious vs hiiilllarious). Thus as mentioned earlier, part of our experiment was to first try omitting the punctuation fully, then try treating the punctuation as separate embeddings, and then try the tf-idf weighting scheme to see if picking or weighing the important words can help the baseline input features of the summed-up "encoded" vector we get from summing the embeddings of the words in a review. For all of these models we ran the same model configurations and hyperparameters of Adam optimizer, a learning rate of 0.1, the default β values of $\beta_1 = 0.9, \beta_2 = 0.99$, and 50 epochs total.

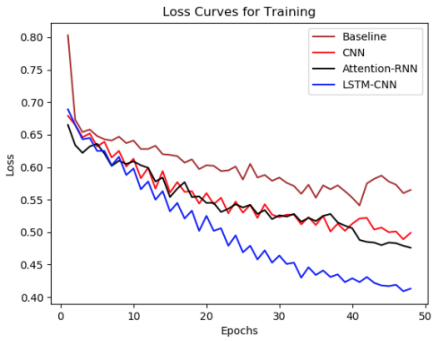
	Train Accuracy	Test Accuracy
Baseline	0.716	0.563
Baseline + Punctuation removed	0.795	0.612
Baseline + punctuation embeddings	0.814	0.625
Baseline + TF-IDF	0.786	0.613

With this part of the experiment done, we decided to try out the remaining models. We kept batch sizes, learning rates, and optimizer the same to provide some level playing field for each of the models and to see if we can properly distinguish with some set of fixed variables shared across the models. We first ran the CNN model where we used windows of up to 5 words, then used 1-dimensional convolutions and max-pooled then concatenated the results before feeding into a fully-connected layer. Next we ran the bidirectional RNN model with attention to increase the weightings of some hidden states before doing a weighted sum and feeding into a fully connected layer. Lastly we ran a LSTM-CNN model where we used the outputs of the LSTM-RNN directly as the inputs to our previous CNN model, with dropout of 0.2 before the fully-connected layer. We retrieved the following

	Precision	Recall	F1-score
Baseline	0.602	0.603	0.602
CNN	0.653	0.622	0.637
RNN+attention	0.664	0.657	0.660
LSTM-CNN	0.723	0.698	0.711

We also provided the learning curves and exact confusion matrices to filter out the individual strengths of each of the 4 models, this can be seen in figure 2.

Figure 2: Learning curve and confusion matrices for the models



Baseline	Funny	Not Funny
Funny	0.601	0.399
Not Funny	0.397	0.603

CNN	Funny	Not Funny
Funny	0.653	0.347
Not Funny	0.396	0.604

RNN	Funny	Not Funny
Funny	0.663	0.337
Not Funny	0.346	0.654

LSTM-CNN	Funny	Not Funny
Funny	0.723	0.277
Not Funny	0.312	0.688

The last the we wanted to try is hyperparameter searching, where we tried to see of the best model, how can we find the best hyperparameters. Unsurprisingly we got the best model was the LSTM-CNN since the model is quite complex. We decided that the most important hyperparameters to search on was the learning rate, the window convolution sizes (in initial one, we chose windows of 2 up to windows 5 before concatenating max-pooled outputs), dropout, and the batch size. We found that the most ideal hyperparameters was learning rate of 0.1, batch size of 32, dropout of 0.3, and window sizes of up to 7. We found a precision, recall, and F1-score of 0.735, 0.701, and 0.719 respectively.

The quantitative results provided some surprises as well as confirmed some expectations. What surprised us at first was that the general accuracy, precision, and recall was pretty low. This is definitely because of some overfitting as the training set's accuracy, precision, and recall for almost all of these models were around 0.9 or even a bit higher. The significant gap suggests there may be high variance between the training and test dataset that was given to us, as well as a slight bias since our training dataset numbers didn't near 100 percent for any of the measures. The precision scores were generally higher than recall scores which means it while it could classify "funny" reviews and have a small number of false positives, it definitely was weak in the amount of false negatives it classified.

The experiment where we included punctuation as an embedding and in a different set up pruned punctuation altogether gave expected results. The punctuation served as an important feature as punctuation like question marks, exclamation points, and commas can help indicate confusion, excitement and sentence sentiment transitions. Separating them into their own embeddings provided a slight increase in accuracy, which makes sense because of the information punctuation give as exemplified above. What was also expected though was that the TF-IDF scores help the scores quite a bit, which means that the most "important" word that was used a lot in this review but usually unique to this review weighted the words more indicative of humor and reduced the dilution of the sentiment with common articles and pronouns that TF-IDF takes care of.

However, the progression of improvement of our models from baseline to the LSTM-CNN didn't surprise us. For one, the baseline is flawed because it loses order information. A lot of the reviews depend on the order of the words because humor is told in a certain format where a setup is made and a punchline is revealed. The CNN performed better, as expected, but what was surprising was how little it improved upon the baseline. We expected a significant increase in F1-score but we only got around a 0.35 increase, which suggests maybe our CNN model was still underfitting, probably because of the lower window kernel size we use for convolutions. The RNN with attention definitely performs better than the baseline with an increase of 0.6 in F1-score, suggesting that our idea of using parameters to detect attention to pay weight to certain scores definitely works and that humor detection is like signal detection. Finally the LSTM-CNN model performs the best with a massive increase in F1-score to 0.711, which is expected because there was a fitting issue with the earlier methods. Having an RNN combined with a CNN can elevate the complexity of the model to properly

fit to the needs of the problem. It also would combine the time-sequence strengths of the RNN with the window-of-word features of the CNN and use those strengths together, so it is unsurprising this performed the best of them all, despite the slowness.

5 Analysis

As stated earlier, one of the biggest surprising points was why exactly there was so much overfitting. The training set would have approximately around 90 percent accuracy and similarly high precision, recalls, and F1-scores, but the test set would have around 70 percent accuracy at best. We decided to take a look at the two datasets and we noticed a noticeable gap. The training set we were had a lot of reviews that fell into the category of "funny story", where the reviewers would recount the subpar experience they had at a restaurant. We saw some examples where stories that are told would recount about how a resort was "covered in green duck crap" and how people had to lay towels everywhere to walk a normal path, or other examples where the waiter was rude but insisted on tips, to which a customer would leave 0.00 "right in his face" as a tip. All of these examples are specific, but they tell a general story with a progression from start to finish with interesting and funny details sprinkled throughout the story. However, looking at the test set, while there is the presence of humor in the form of direct story-telling, a lot of the funny reviews in the test set fell into the category of sarcasm. Many reviews in the test set would talk about how they loved "cold pizza that took 20 minutes to make" or "they loved eating kebabs in a dingy apartment". This form of storytelling is much less direct and requires more of a keen eye to see. Sometimes even humans can mistake sarcasm for seriousness on online forums, where there isn't any form of "vocal tone" present to highlight the sarcasm. The words themselves don't tell give away the humor because they are generally used for more serious inflections in review writing. As an example, "I absolutely adored the cold pizza that took 20 minutes to make, I am definitely coming back up here!" has only 1 negative word "cold" but the rest are positive so it would be hard for a model to learn that this sentence is sarcastic, especially if the training set had much less sarcasm woven into its reviews. That said, it was present in the training set, but it is a difficult problem to train on because not all forms and uses of sarcasm are the same.

We looked at some examples that were classified correctly by certain models and misclassified by other models to see if we can understand the strengths of each. The baseline model was outperformed with the simple extension of making the punctuation its own embeddings and it helped with examples that included a lot of punctuation marks as it helped indicate an extreme emotion of over-the-top reaction that people found funny. As an example, the baseline got the review "OMG DQ always has the best milkshakes!!!!111!1!!1!" wrong because it filtered out the punctuation, but the baseline with the punctuation extension got this correct because it kept the punctuation and understood that there was a heavy overreaction that people can find funny. Similarly the baseline with TF-IDF weighting, as expected, performed better on examples where the most important word was usually the strongest indication of humor. As an example, the baseline got the review "Let me tell you, the chips over here are absolute cancer. It's so cancer and easily staler than a 'your mom' joke in 2018" wrong but the TF-IDF based model got this correct due to the high TF-IDF scores of "cancer". "Cancer" is usually a term used in twitch streaming amongst a younger audience so it isn't used that much in the reviews which is why the TF-IDF method paid it more attention. Similarly the RNN-attention model did similarly well on examples such as this where it tries to provide an attention score to the most indicative word in the sentence while also providing the benefits of a time-sequence-processing model.

It seems that the CNN, RNN with attention, and LSTM-CNN performed better and better when it comes to sarcastic reviews in almost this order. This makes sense because CNN can't capture long term dependencies that sarcasm might have, and the RNN does better with this addition of attention and time-sequence data-processing. The last model LSTM-CNN performed the best out of all of them simply because it was able to fit the best. It had the most parameters and was able to combine the feature of understanding time-sequencing with the hidden states and use those directly to create features of windows of the hidden states based on the 1-dimensional convolutions. As such, the better loss curves to the more complex models can be attributed to the fact they have more parameters to better fit to the dataset.

6 Conclusion

Humor in language can very difficult to detect sometimes and it can come in a variety of forms. Some forms of humor depend on immature jokes, some forms surround dark humor, some forms focus on wordplay and puns, and some focus on real-life references and sarcasm. These various forms and the whole concept of subjectivity of humor where one form of humor is funny to some and not funny to others makes it a difficult problem in NLP today. There is no objective rule to detect if something is funny or not, unlike a task like detecting sentiment within a body of text.

Despite the difficulty of this task, there has been thorough research performed on the subject. People have used sitcoms such has Big Bang Theory and marking transcript text as "funny" if it elicited a laugh and trained various models such as Support Vector Machines (SVM), conditional random fields (CRF), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) to recognize humor.[3][4][5][6]. Other such as D'Oliviera have toyed around with similar networks on a different dataset such as Yelp reviews, using the "funny" vote as an indicator of humor [8].

For our project, we wanted to see specifically if combining the models or trying some more complex attention mechanisms similar to the Yelp dataset that Olivieria used would provide better results than the conventional neural network models. Yelp reviews were a good choice for a dataset because we believe people have a lot of "funny" experiences with the establishments and the format of the site allows personality to leak through the reviews, making them funny and interesting. We have a dataset of approximately 120,000 Yelp reviews, where exactly half are deemed "funny" and half are "not funny". We use RNNs, CNNs, and a combined LSTM-CNN to try to capture the various structures of humor. While our LSTM-CNN performs best, with an F1 score of 0.711, we find that the subjectivity of humor and the complexity of structures such as sarcasm make it difficult to build a highly predictive model.

Overall we experienced a severe overfitting because of the mismatch between the train and the test dataset, since the former has humor in the form of direct story-telling whereas the latter has humor more in the form of sarcasm. In addition to this, sarcasm is a difficult feature to detect, even people have trouble telling sometimes. We used a baseline model of just averaging word2vec vectors and we found that extending that with TF-IDF weighting and using punctuation as their own separate embeddings helped capture more information about the humor of the dataset which came in the form of finding more important "humorous" words and detecting strong overreactions using exclamation marks and more. The vanilla CNN model didn't improve upon the baseline very much, but the RNN with attention performed a lot better. We attribute it to the similar reasoning why TF-IDF weighting worked well: because attention pays importance to certain signals or key words. The LSTM-CNN performed the best out of the group of models with a high F1-score of 0.711, and we attribute this to the fact that the dataset is quite complex so we needed the strengths of RNN's time-sequencing processing and CNN's ability to make features out of windows of words via convolutions. There is quite some work to be done, as the test set measurements were not high at all. Future work could include studying how to detect sarcasm within text or trying some different models here such as CNN-LSTM where a CNN convolution outputs are then fed into the LSTM network or LSTM-CNN with attention on the LSTM network to provide "weights" to the hidden outputs before a convolution. Overall, while the problem still remains difficult due to the presence of sarcasm, we have obtained some expected and intuitive results that could pave the way for further progress in research on this particular task.

References

[1] E. Cambria, "Affective computing and sentiment analysis," *IEEE Intelligent Systems*, vol. 31, pp. 102–107, March 2016.

[2] H. Buress, "Live in chicago," 2014.

[3] C. Kiddon and Y. Brun, "That's what she said: Double entendre identification," pp. 89–94, June 2011.

[4] P.-Y. Chen and V.-W. Soo, "Humor recognition using deep learning," pp. 113–117, 2018.

[5] D. Bertero and P. Fung, "Deep learning of audio and language features for humor prediction," 2016.

[6] L. Chen and C. M. Lee, "Predicting audience's laughter using convolutional neural network," 2017.

[7] P. M. Sosa, "Twitter sentiment analysis using combined lstm-cnn models," June 2017.

[8] L. de Oliveira and A. L. Rodrigo, "Humor detection in yelp reviews," 2015.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.