Identifying and Generating Humor with Single-Word Substitutions (Final Report)

Daniel Hathcock
Georgia Institute of Technology
dhathcock3@gatech.edu

Sherry Sarkar
Georgia Institute of Technology
sherry.sarkar@outlook.com

Abstract

In this project, we implement several models that can be used to identify humor in news headlines which have been altered at one word. We will consider several different LSTM models, and compare the performances of each in their ability to identify humor. We will also use these models to address the the task of determining which of two altered news headlines is funnier. We predicted that the context of the original news headline would play a role in how humorous the altered headline is. Therefore, we hypothesized that a model with access to the original headline would perform better. In this project, we show that this hypothesis is not necessarily true and that, in particular, the simplest model which only uses the altered headline achieves the best performance. Please find the code at: https://github.com/danielchathcock98/iwt-wot

1 Introduction and Related Work

1.1 Motivation and Goals

What makes a political cartoon funny? The Onion, a political satire digital media company which generates parody news headlines, has over 6.5 million followers on Facebook. On a broad scale, data scientists are interested in understanding what makes content "humorous".

We base our project on Hossain, Krumm, and Gamon's data set and paper. The data set consists of pairs of a sentence and an "altered" sentence with a single-word edit. Each input instance has a label measuring the level of humor created by the edit. Hossain, Krumm, and Gamon were able to train some baseline classifiers and distinguish general patterns of humor (Hossain et al., 2019).

The advantage of this data set lies in the simplicity of the edit and the direct effects it has on the humorousness. A single-word edit corresponds to a small feature change in the input sentence with large corresponding changes in the output humorousness. We may study the semantics of certain words, the relevance of surrounding context, as well as observe rough "classical theories" of humor (Hossain et al., 2019). We hope that our work assessing the effects of small changes on humor will make progress towards the general goal of understanding humor.

Our task is to train models which can evaluate and generate humor in sentences. To approach this broad goal, we specifically plan to build upon the goals of *SemEval-2020 Task 7: Assessing Humor in Edited News Headlines* (Hossain et al., 2020a). We will use the Hossain, Krumm, Gamon data set of news headlines in which a single word has been substituted, and labels indicating the level of humor due to the alteration. We aim to create models addressing two sub-tasks:

1. Predict how humorous a given altered news headline is.

2. Predict which of two altered news headlines is funnier.

We originally had a third task which was: given a news headline, *generate* a single-word substitution which is humorous. However, the first two tasks of simply recognizing humor proved to be highly nontrivial, and so we focused our efforts on suitable recognition rather than generation.

1.2 Related Work

There have been several other papers studying various flavors of humor. Khodak, Saunshi, and Vodrahalli (Khodak et al., 2017) scrape Reddit to further understand instances of sarcasm. They create a data set, the Self-Annotated Reddit Corpus (SARC), which has true labels based on the "/s" tag which sarcastic Reddit comments will often have. In addition, the authors run an bi-gram analysis and find highly positively weighted (sarcastic) and highly negative weighted (not sarcastic) phrases. Davidov, Tsur, and Rappoport (Davidov et al., 2010) also study sarcasm. They use a semi-supervised learning approach to classify Twitter tweets and Amazon reviews as sarcastic or not sarcastic. Davidov, Tsur, and Rappoport use features like speech tagging (high frequency words or "content words") as well as punctuation and use a k nearest neighbors approach (essentially, give an input the label of its k nearest neighbors, and if there are not enough neighbors, the default label is 0) to classify sentences. Cattle and Ma (Cattle and Ma, 2018) in contrast with the other two studies, take a more direct semantics approach while maintaining the goal of having an interpretable model. In particular, they focus more on using word associations from Word2Vec and sentence structure (as opposed to the bag-of-words approach several prior works have used). Their methods include a 3-gram model and perplexity.

Several of these works shy away from a supervised learning approach. This may in part be due to a lack of interpretability of models in supervised learning. It may also be that these methods are simply not as effective due to the subjective nature of the labels and the scarcity of accurately labelled training data. In this project, we investigate the effectiveness of supervised learning models. Specifically, we aim to determine how effective different LSTM-based models are at differentiating between humorous and not humorous sentences.

The study we are basing our project on (Hossain et al., 2019), both analyzes the data and tests out a supervised learning approach. They were able to extract general trends in the data, including the fact that funnier headlines had the replaced word later in the sentence (due to the "joke set up and punchline" theory). They also use as LSTM on the same dataset to predict which headlines are extremes (i.e, definitely funny or definitely not funny). In addition, they only focused on a model which takes the altered headline as its input (i.e. it discards the original headline). Our work is a generalization theirs since we look to have an accurate prediction of all headlines, not just those which fall in the extremes. Lastly, we hypothesize that the context of the original headline when reading the altered headline is often important in determining how humorous the alteration is, and therefore, both sentences are necessary for optimal model performance. Exploring this hypothesis is a major goal and contribution of our project.

2 Methods

2.1 Data

Our project's dataset is from Hossain, Krumm, and Gamon's paper (Hossain et al., 2019). We also use extra training data provided by Hossain, Krumm, Sajed, and Kautz (Hossain et al., 2020b). The following table summarizes a few basic statistics of our training set for **sub-task 1**.

Number of Training Examples	17900
Number of Dev Examples	2420
Number of Test Examples	3024
Average Sentence Size	12.4
Max Score	3.0
Min Score	0.0
Average Score	1.07
Most Common Score	1.0

We also show the distribution of all scores in the training and validation data in a histogram:

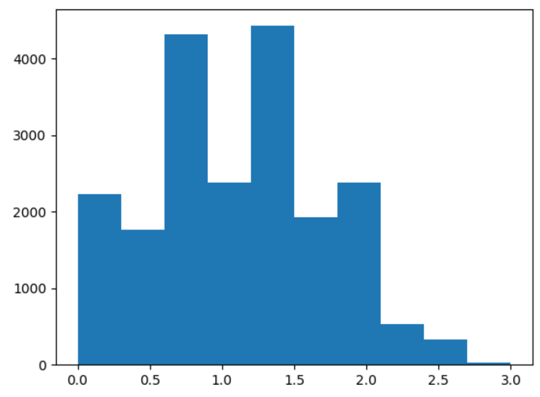

Figure 1: Distribution of scores in training/validation data.

We provide some positive examples (i.e humorous sentence edits) which we are trying to recognize: "The Alex Jones influence : Trump's 'deep <state>' fears come from his conspiracy theorist ally and adviser" becomes "The Alex Jones influence : Trump's 'deep <dish>' fears come from his conspiracy theorist ally and adviser". This received a score of 1.6 out of 3.

The following is a negative example: "France is hunting down its citizens who joined <Isis>

without trial in Iraq" becomes "France is hunting down its citizens who joined <twins> without trial in Iraq". This received a score of 0.2 out of 3.

We had to pre-process our data in slightly different ways to fit each of the three models we will be implementing.

We also had data for **sub-task 2**. The data in sub-task 2 looks like: (news headline 1, news headline 2, label) where each news headline contains both the original and altered, and label indicates which news headline is funnier. Label 0 indicates a tie, label 1 indicates that headline 1 is funnier, and label 2 indicates that headline 2 is funnier.

There were 2960 test examples for sub-task 2. We only used the training/validation data for sub-task 2 for the baseline model. The model we developed was simply a model for sub-task 1 applied to sub-task 2. Thus, we only provide a histogram showing the frequencies of labels:

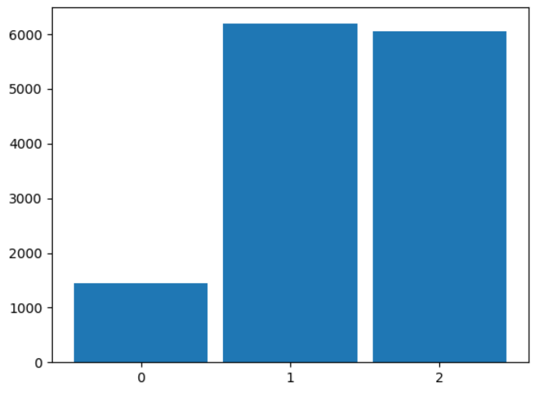

Figure 2: Distribution of labels in training/validation data for task-2

Finally, we are using pre-trained fasttext embeddings (Mikolov et al., 2018). The pre-trained model consists of a 6.8G binary data file. The embeddings were trained on web common crawl. The model can be loaded from this file using the fasttext Python library, then the embeddings for each word can be accessed directly. No additional processing was necessary on this data.

2.2 Models

For **sub-task 1**, we had originally planned on using a simple model which would use the pre-trained fasttext embeddings to convert a headline to a sequence of embeddings, then run it through an LSTM and send the final hidden state through a linear layer which would output a humor score. We realized, however, that there are many ways to alter and implement this basic idea. Therefore, we implemented several versions of this model. We will describe each in detail:

- **Model 1**: Simply ignore the original headline. Take the altered headline, convert it to embedded space with fasttext, then run it through an LSTM. Send the final hidden state of the LSTM through a linear layer to get a vector of size 2. Then apply a softmax to get a probability distribution (i.e. 'funny' vs 'not funny'). Finally, output the first entry of the probability vector, scaled up by a factor of 3.

- **Model 2**: First, concatenate the original headline with the altered headline to form one large input sentence. Then, proceed exactly as Model 1 did with the altered headline.

- **Model 3**: Take the original and altered headlines individually, and covert them to embedded space with fasttext. Then, send each individually through two individual LSTMs. Take the final hidden states of the two LSTMs and concatenate them into one vector which is sent through a linear layer to get a vector of size 2. Then, as before, apply softmax, and return the first entry scaled up by 3.

All of our models were trained using the *mean squared error* (MSE) loss function, and SGD as an optimizer. The MSE loss function makes sense since the model is attempting to estimate a score on a continuous scale. It also makes sense as *root mean squared error* (RMSE) is our main metric of evaluation, as per the SemEval challenge specifications (Hossain et al., 2020a). This is defined as

$$\mathrm{RMSE}(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \qquad (1)$$

where x_i is the score output by the model for the ith train/val/test example, and y_i is the true label for that example.

For **sub-task 2**, we used our best performing model (Model 1). We trained Model 1 on the training and validation data of sub-task 1. Then, for each example in sub-task 2, we ran the first and second sentence through model 1 and compared the predicted scores. If the predicted score for headline 1 was higher, then our label was 1 and if the predicted score for headline 2 was higher, our label was 2. We also added an extra “tie margin” parameter to our model which would label the example as a 0 (a tie) if the predicted scores of headlines 1 and 2 were within the tie margin. This hyperparameter was tuned on the sub-task 2 validation set.

2.3 Baseline Models

For **sub-task 1**, we implemented two simple baseline models:

- Compute the mean score of all examples in training (and validation) data. Then guess this score for all test examples.
- Compute the mode (most frequent) score of all examples in training (and validation) data. Then guess this score for all test examples.

The **mode** baseline model achieved the better RMSE of 0.578, while the **mean** baseline model achieved an RMSE of 0.590.

Sub-task 2 is a classification task with three labels, so the natural baseline we implemented is determine the most frequent label in the training set and guess this label for all test examples. The task 2 baseline model scored an *accuracy* of 0.4354.

3 Results

3.1 Experimental Setup

With our models, we used a train/validation split of approximately 85/15, and a (train + validation)/test split of also approximately 85/15. See section 2.1 for exact numbers.

For all models, we used a simple SGD optimizer, with the learning rate tuned to each model. Specifically, we had:

Model	Learning Rate
Model 1	0.18
Model 2	0.2
Model 3	0.18

For sub-task 1, our evaluation metric was RMSE (see equation (1)). For sub-task 2, our evaluation metric was simple *accuracy*: the number of correct predictions divided by the total number of examples. This is a reasonable metric in this case, since there is not a huge bias toward any one label (no label has more than 50% of the data). Both of these evaluation metrics are also the metrics specified in the SemEval challenge that this project is based on (Hossain et al., 2020a).

3.2 Result Comparison for Sub-task 1

Our models achieved the following RMSE on the test set after training on both training and validation data (the models were tuned by training on train data and viewing performance on validation data). We did two train/test cycles each.

Model	Test RMSE
Model 1	0.569, 0.571
Model 2	0.583, 0.566
Model 3	0.585, 0.592

Notice that Model 2 achieved the best single score, while Model 1 had the lower mean. We also observed a consistently lower RMSE validation error while training model 1. We therefore chose model 1 as the best model for use in sub-task 2 and for use in error analysis. Model 3 clearly performed the worst. We performed a simple independent two-sample t-test to attempt to determine the statistical validity of differentiating these models. Specifically, we computed

$$t = \frac{\overline{X}_1 - \overline{X}_2}{s_p\sqrt{2n}} = \frac{\overline{X}_1 - \overline{X}_2}{s_p}$$

since we have $n = 2$ samples. Here, $s_p = \sqrt{\frac{s^2_{X_1}+s^2_{X_2}}{2}}$ with $s^2_{X_i}$ the unbiased estimator for variance of X_i. A p-value can be easily computed from this, giving the statistical significance of rejecting the null hypothesis that the means are not significantly different. We may also perform a single sample t-test for Model 1 against the baseline of 0.578. We get the following values:

Model	t	p
Model 1 vs. Model 2	-0.526	0.65
Model 1 vs. Model 3	-5.08	0.04
Model 2 vs. Model 3	-1.52	0.27
Model 1 vs. Baseline	-8.0	0.08

Taking $p < 0.10$ as meaning statistically significant, we can see that model 1 performs better than model 3, and that model 1 performs better than baseline. And, of course, comparing model 1 to the inferior mean baseline of 0.59 would show an even more significant improvement over the baseline. On the other hand, we cannot reject the null hypothesis on differentiating any of the other models. In particular, we cannot assume that model 1 performs better than model 2 in a statistically significant sense.

In particular, this comprises a partial negative result: we predicted that models with access to the original headline could perform better than those without. However, we showed here that, in a statistically significant sense, model 1 performed better than model 3 despite not having access to the original headline. On the other hand, we cannot tell whether model 1 performed better than model 2 in a significant way.

There are a myriad of possible explanations for this result. One is simply that we did not find the "correct" model to take advantage of the information using the original headline. Maybe some other LSTM model using the original headline could outperform model 1. Another explanation is that the models which take both the original headline and altered headline as input are more prone to overfitting. We observed this to some extent (the epoch at which overfitting started was later in model 1 than the other two). It is possible some techniques could be applied to the models we tried to counteract this issue and restore the performance of models 2 and 3.

Hyperparameters: In all of our models, the main hyperparameter we altered was the size of the hidden state in the LSTM. One way to view this is: the LSTM acts as an embedding layer for the entire sentence, turning the sentence of word embeddings into a single vector (the last hidden state), which should capture some information about how funny the sentence is as a whole. Therefore, tuning the size of this sentence-level embedding was important; it had to be large enough to capture information about humor, but small enough to avoid overfitting.

We found the the size of the hidden state would depend on which model we used. For example, the simplest Model 1, which used only the altered news headline, the size of the hidden state could be larger, since overfitting wasn't as much of an issue. Therefore, we used a hidden state of size 6. Model 2, while slightly more complicated, has the same number of model parameters, so was only slightly more prone to overfitting. On the other hand, the more complicated Models 3 had more parameters, and larger inputs. Therefore, a smaller hidden state size was more suited.

We also found that adjusting the number of epochs a model ran was effective in improving performance. Each model began overfitting by the time the 30th epoch was reached. We found that models performed best with around 15-20 epochs. We tuned this simply by viewing the validation accuracy during each epoch of training. When the validation accuracy started to rise, the model had begun overfitting. Notice in Figure 3 for model 2 that the validation accuracy begins to increase after epoch 14.

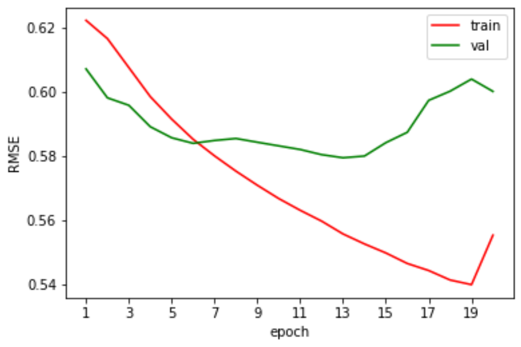

Figure 3: Training and validation accuracies for model 2 over training time.

Below we specify the choice of hyperparameters for each model.

Model	Hidden State Size	Epochs
Model 1	6	16
Model 2	6	14
Model 3	3	12

Tuning Model Features: One particular feature of these models that we had to adjust with was how to output the final score. Our first idea was to simply have the final linear layer have output dimension 1, and use this single output number as the score. Then, the model could be trained as a regression model with, e.g. MSE loss. This, however, achieved RMSE results worse than the baseline models. In particular, when we implemented Model 2 this way, it achieved RMSE ≈ 0.61, much worse than both baseline models.

The issue with simply outputting a number from the linear layer is twofold:

- The model doesn't have hard bounds on the output. It might infer some semantic meaning to a headline having a score of 4, or -1, for example, even though no such scores can exist.

- The distribution of scores is mainly in the middle to lower ranged, with very humorous examples being rare (see Figure 1). Simply using a linear layer as the output would fail to capture this very well.

Rather, the distribution of scores suggests a different approach: treat the score, divided by 3, as a *probability of being funny* (this is actually a rather accurate description of the score, since each score is determined by 5 votes of people scoring the headline some integer from 0 to 3. So, it can be interpreted as the probability of it being funny to a random person). To implement this, we would instead have the final linear layer of the model output a vector of size 2, then apply a softmax to turn it into a probability vector. Entry 0 corresponds to the probability of the headline being humorous, while entry 1 (equal to $1 -$ (entry 0)) is the probability of not being funny. Taking entry 0 and scaling it by 3 then gives the output.

We tried 2 loss functions with this new scheme. One was the KL divergence between the distribution of funny vs not funny against a "true distribution" which we calculated with the true label score x as

$$d = \left[\frac{x}{3}, 1 - \frac{x}{3}\right]^T$$

While this implementation gave slightly better results than the previous one, it didn't perform as well as the implementation we decided to use in the end: simply take the output of the model (the probability of being funny scaled up by 3) and perform the MSE loss between that and the true score.

Error Analysis: We determined which examples Model 1 performed most accurately on and which examples Model 1 performed most poorly on. We found that our LSTM did best with examples that were rated as "not so funny", with labels 0.8 or 1.2. Out the 10 best predicted examples, 8 had one of these labels. In addition, Model 1 did worst with examples that were rated as "very funny" or "not funny at all", i.e labels that are $2.6 - 2.8$ or 0. Out of the 10 worst predicted examples, 7 had one of these labels. This is in stark contrast to the results of (Khodak et al., 2017), which suggested that humor should be easier to classify at the extreme ends. There could be a few reasons for this. First, it could be that our model is more likely to predict scores in the "not so funny" range. In each of the "very funny" examples our model failed to predict, our model predicted a score in the 0.7 to 1 range. However, it should be noted that for the two examples with a 0 rating, our model predicted 2 and 2.1.

Also, several of the example classified as "very funny" (a 2.8 out of 3), were in our opinion, not very funny. For example, one poorly predicted example was "CNN's Jake Tapper to *interview* Paul Ryan following retirement announcement" which becomes "CNN's Jake Tapper to *wrestle* Paul Ryan following retirement announcement". Examples of good humor involve real life context (i.e current events, associations, etc); however, this example seems to be playing more on absurdity.

3.3 Result Comparison for Sub-Task 2

The baseline model got a test accuracy of 43.54%. Our model got a test accuracy of 54.73%, so we are performing significantly better than baseline with model 1, despite the same model performing very close to baseline on sub-task 1.

Hyperparameters: In sub-task 2, we had an additional "tie margin" parameter, which we found to be optimal at 0.0004. We tuned this hyperparameter by simply checking the accuracy of the model on the validation dataset for sub-task 2, and finding the value of the margin parameter which maximized accuracy. We found that the margin parameter had a negligible difference on the accuracy (unless it was too large, in which case it had a very detrimental effect):

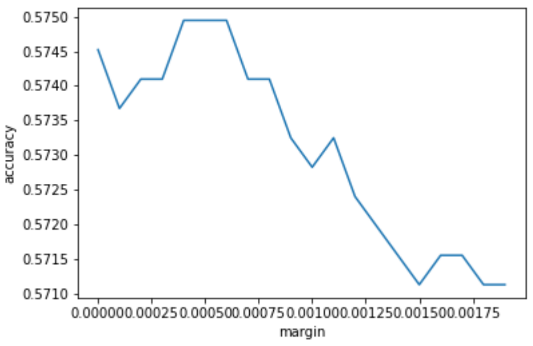

Figure 4: Plot of how the margin hyperparameter affects accuracy

Since sub-task 2 is a traditional classification problem, we are evaluating this model with the the classical definition of accuracy, i.e number of correctly classified examples over total number of examples.

Error Analysis: We first investigated what examples our model was most confident in classifying, i.e the difference between the predicted score of headline 1 and the predicted score of headline 2 was large. In the top 10 examples our model was most confident in: 7 were correctly classified and 3 were misclassified. Of the 3 misclassifications, 2 were actually ties, and 1 was classified opposite of the truth. Interestingly enough, the single incorrectly classified example was also part of the top 10 incorrectly scored examples from sub-task 1: "A vote on California's landmark removal is coming down to the *wire*" becomes "A vote on California's landmark removal is coming down to the *nail*". Our model seems to think this edit is very humorous. Next, we looked at what our model was least confident in classifying. 8 out of the 10 were misclassified, and none of the true labels were ties.

3.4 Work division

Daniel Hathcock worked on:

- pre-processing of the data
- word embeddings
- trainModel.py, debugging, and organizing the Jupyter notebook
- running models on the Jupyter notebook, hyperparameter tuning.
- t-testing
- writing up content for the proposal, midterm report, and final report.

Sherry Sarkar worked on:

- pre-processing of the data
- implementing the models in model.py
- implementing baseline models
- error analysis
- writing up content for the proposal, midterm report, and final report.

4 Conclusion

In conclusion, for sub-task 1, our models had mixed results. Model 1 was able to consistently recognize humor slightly better than the baseline models, while the other two performed similarly or worse than the baseline. We found through error analysis that our models had more trouble with extreme examples than with mid range examples. Examples with very low humor scores were headlines that simply made sense, and examples with high humor scores were headlines loaded with political context and layers of jokes; therefore, the fact that our model did not understand extreme examples could be mean that our model did not learn word associations and context well enough. This may be fixed by training our model on word embeddings that are political and humor focused.

Additionally, we can conclude that our hypothesis was not correct: we believed that the context of the original headline would be useful in identifying humor, but this turns out not to be true due to Model 1's performance.

In sub-task 2, our models performed noticeably better than baseline in recognizing humor levels between headlines. Our model performed best when the "tie margin" hyperparameter was set to 0. Thus, a low level observation we may make is that the pairs that our model deemed of similar levels of humor did not necessarily closely correlate with the examples that were labelled "ties". Indeed, it seems like a strictly more difficult task to determine whether two sentences are equally funny with high accuracy rather than determine which is funnier.

One fundamental problem with classifying humor is the inherent subjective nature of the labels. Upon looking through several of the examples, we personally found that some very highly rated headlines were not very funny, and some lowly rated headlines were funny.

This project shows that purely supervised learning methods are able to grasp general patterns of humor. Unsupervised approaches in the field of recognizing humor have one key advantage; they tend to be more interpretive models. Therefore, some mix of the two approaches (i.e through semi-supervised learning, as done by (Davidov et al., 2010)) could be the optimal balance. Often times the absurdity of a headline determines the humorousness of it; therefore, using a bi-gram model and measuring the perplexity of a headline could be a very useful feature in a supervised model.

Another idea to improve the performance of these models is to try different "sentence embeddings". In Section 3.2, we discussed that the LSTMs in our models can be viewed as mapping a sentence to a vector, or finding a low-dimensional embedding for a given sentence. There are many other, more complicated architectures which perform the same task. For example, we could have attempted to use BERT, a deep neural network which does precisely this with pre-trained weights (Devlin et al., 2018). One advantage of BERT is that the pre-trained weights allow one to forgo the use of pre-trained word-embeddings. Additionally, it has achieved state-of-the-art in many NLP tasks, and thus could be useful for a problem like identifying humor. This project's authors' lack of experience with such models make this a prime task for future work.

5 Code

Please find the code repository at: `https://github.com/danielchathcock98/iwt-wot`.

References

Andrew Cattle and Xiaojuan Ma. 2018. Recognizing humour using word associations and humour anchor extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1849–1858, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Semi-supervised recognition of sarcastic sentences in twitter and amazon. *CoNLL 2010 - Fourteenth Conference on Computational Natural Language Learning, Proceedings of the Conference*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Nabil Hossain, John Krumm, and Michael Gamon. 2019. " president vows to cut¡ taxes¿ hair": Dataset and analysis of creative text editing for humorous headlines. *arXiv preprint arXiv:1906.00274*.

Nabil Hossain, John Krumm, and Michael Gamon. 2020a. Semeval-2020 task 7: Assessing humor in edited news headlines. In *Proceedings of International Workshop on Semantic Evaluation (SemEval-2020)*, Barcelona, Spain.

Nabil Hossain, John Krumm, Tanvir Sajed, and Henry Kautz. 2020b. Stimulating creativity with funlines: A case study of humor generation in headlines. *arXiv preprint arXiv:2002.02031*.

Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. 2017. A large self-annotated corpus for sarcasm. *arXiv preprint arXiv:1704.05579*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.