Generating Original Jokes

Author
Ting-yu YEH
Nicholas FONG
Nathan Kerr
Brian Cox

Supervisor
Dr. Ming-Hwa Wang

March 20, 2018
Abstract

I Introduction
1-A What is the problem? ................................................................. 2
1-B Why is the project related to this class? .......................... 2
1-C Why other approach is no good ................................................. 2
1-D Why our approach is better ...................................................... 2
1-E Scope of investigation .............................................................. 2

II Theoretical Bases and Literature Review
II-A Theoretical background of the problem .............................. 2
II-B Advantage/disadvantage of those research .......................... 3
II-C Solution to solve this problem ................................................. 3

III Hypothesis

IV Methodology
IV-A Data collection ................................................................. 3
IV-B Language and tool used ...................................................... 4
IV-C how to generate output ...................................................... 4
IV-D Algorithm design .............................................................. 4
IV-D1 N-Grams ........................................................................ 4
IV-D2 Word Based ..................................................................... 4
IV-D3 Character Based .............................................................. 4
IV-E Scoring metric ................................................................. 4
IV-F Testing against hypothesis .................................................. 4

V Implementation
V-A Phonetic Edit Distance ....................................................... 5
V-B N-gram .............................................................................. 5
V-C Word Based Approach ....................................................... 5
V-D Character Based Approach .................................................. 5

VI Data Analysis and Discussion
VI-A output generation ............................................................... 5
VI-B output analysis ................................................................. 6
VI-C discussion ....................................................................... 6

VII Conclusions and Recommendations
VII-A summary and conclusions ................................................. 6
VII-B recommendations for future studies ............................... 6

References

Appendices

LIST OF FIGURES
1 General flowchart of the proposed algorithm ......................... 3
2 Neural network representation [5] ........................................ 4
3 Recurrent neural network [5] .............................................. 4
4 Flowchart for word-based RNN model ................................... 7

LIST OF TABLES
I Outputs from different models .............................................. 7
Generating Original Jokes

Ting-yu Yeh, Nicholas Fong, Nathan Kerr, and Brian Cox

Abstract—Computational Joke generation is a complex problem in the field of artificial intelligence and natural language processing. If successful, however, computational humor would play an essential role in interpersonal communication between humans and computers. In this paper, we use natural language processing (NLP) techniques paired with various models to generate original puns. We found that character-based recurrent neural network (RNN) is a more solid approach to generate original jokes by comparing its results with those generated by trigram and word-based RNN models. Using jokes from sources like Reddit.com, Twitter, and joke specific websites to train our models, we evaluate results and present our conclusions.

I. INTRODUCTION

A. What is the problem?

Humor is an integral part of human interaction. Some jokes are very complicated and require an intricate backstory, while others can provoke laughter with a simple alliteration. Even simple jokes can be very hard to algorithmically generate, mainly due to the need for context or external knowledge. Although there is no crucial need for computationally generated humor, it could be beneficial for many applications. As the majority of developed societies are advancing, the emergence of accompanying robots is inevitable. Humor, being one of the most important features in interpersonal communication, is critical to establish and promote conversation. To solve this need, we want to computationally generate original one-line jokes.

B. Why the project related to this class?

Generating humor is an important part of natural language processing because humor is a common aspect of human interactions. If we can generate humor, then we can improve the ability of chatbots and AI to mimic humans, making them more relatable. This would have many benefits including chatbots having a better chance at passing the Turing test.

C. Why other approach is no good

A lot of current approaches aim to identify humor because that is a hard task in and of itself. However some have been some other methods that attempt to generate humor but they are very rudimentary. For example, Valitutti et al. only researched changing a single word in a sentence to try to make it funny [1]. However, most humor involves the whole sentence, where context and buildup are essentials to create funny jokes. As a result, the amount of humor they were able to produce is very limited and elemental.

D. Why our approach is better

Our approach is better for multiple reasons. The first one is most current methods use very elementary algorithms to replace a word, or pull parts of a sentence to construct a joke. Our process will include a recurrent neural network to generate and build intelligent puns from a large dataset. Additionally current approaches do not include phonetic features to generate puns. With the addition of word phonetics, we believe we can achieve better results.

E. Scope of investigation

Computational humor is relatively a new area with not a lot of advancements. It involves knowledge and techniques across multiple disciplines, including psychology, artificial intelligence, natural language processing, etc. Therefore, we are limiting our scope to try and generate basic one-line puns. This goal will ideally not be too big of a project, but will still deliver the intended results of humor generation.

II. THEORETICAL BASES AND LITERATURE REVIEW

A. Theoretical background of the problem

There are several different theories of humor [1]. One is the superiority theory, that we laugh at the misfortune of others because it makes us feel superior. An example of this is when we laugh at videos where someone falls and hurts themselves. Another theory is the relief theory, that we laugh to release nervous energy. For example, if we expect danger, but it turns out to not be dangerous at all, we laugh. A third theory is the incongruity theory, that we laugh when there is incongruity in a playful context. Puns, for example, make use of this theory heavily as they add incongruity through words with double meanings. Most researchers agree that the incongruity theory is best, though a combination of them all may be true. Researchers estimate incongruity as a combination of ambiguity and distinctness. Ambiguity is how many likely interpretations a sentence has. Distinctness is how much of the sentence supports each interpretation. Generating humor is an incredibly difficult task. Humor
involves introducing incongruity in a way that makes sense so as to be funny. This requires that the software understand and intentionally create double meanings in a message. It involves connecting the context and the anomaly. Understanding and generating normal text is hard enough, but understanding and generating humor is even harder. In addition, humor is often subjective, much of which requires inside knowledge about a subject. This makes humor hard to objectively validate and rate.

B. Advantage/disadvantage of those research

Valitutti et al. researched generating humor by changing a single word in short text messages [1]. By limiting the scope to short texts, and by only changing a single word from an existing sentence, this made their research much simpler. They did explore different rules or constraints to guide which word to change and what to change it to. One advantage of this research helps highlight the different ways those rules of humor interplay, showing that creating puns that refer to a taboo topic near the end of a text is most funny. They have also shown that using bigrams to make sure the changed word makes sense in its context only provides marginal improvement to the humor rating. Another advantage is that they showed a way to measure humor: crowdsourced voting. However, a disadvantage of their research is that it only looks at text messages, which are by nature short with a max length of 140 characters. In addition, by only changing a single word they are systemically unable to produce longer jokes and intelligent humor.

C. Solution to solve this problem

We will apply the concepts in LSTM (long short-term memory) with RNN (recurrent neural networks) to generate jokes. Our RNN will take an input sentence and change words that are orthographically or phonetically similar. In other words, the changed words will have a similar spelling or pronunciation. This is important in generating puns, the scope of humor that we will focus on. According to incongruity theory, humor is generated when the conflicts between context and anomaly are resolved. Therefore, we could mechanically calculate the distance between the joke to the topic words. In addition, to evaluate how funny a joke could be in the training process, we could refer to the feature extraction methods used by Shahaf et al. [2].

The concept of LSTM is critical to generate humor because joke words are funny only when they are based on certain contexts. With RNN, results generated from the previous inputs can serve as the “memory” to generate more contexts, creating coherency for the joke to make sense. Finding the useful features to train efficient RNN is another topic. Ren and Yang used POS method to train joke generating RNNs. [3] Our approach involves more intrinsic features, such as word phonetics analysis, to generate pun-type of jokes.

III. Hypothesis

The jokes outputted by our system will, on average, be rated no worse than 30% lower than the average rating of the human-generated jokes. Both types of jokes will be rated by at least 15 neutral participants.

IV. Methodology

A. Data Collection

We used an open source dataset which contains 231 thousand short jokes scraped from various sources including Reddit and various joke websites. To shorten training time, the 231 thousand joke dataset is truncated to 20,000 samples. For additional data, we used the open source joke database provided by Pungas, which contains 208,000 English plain jokes scraped from three sources, reddit, wocka.com, and stupidstuff.org [4]. This joke dataset is used to train the RNN to generate original jokes that will be later evaluated.
B. Language and tool used

We used python as the main language to implement the RNN. For character based approach, we used Keras. For word based method, words are converted to feature vector with Word2Vec and RNN was trained with Tensorflow. Both RNN are implemented with LSTM cells.

LSTM concept. To train the RNN model with the joke database, we have different options, including TensorFlow and Torch. In order to generate pun jokes, we will first find the set of keywords sharing similar phonetic structures. Next we will pick the word with short edit distance to form a pair with the original keyword. During the generation process, random output will be selected in each RNN step based on calculated probabilities. Repeating the process would produce a set of candidate jokes which will be fed into feature evaluation classifier described in Shahaf et al. [2]. The final generated joke will then be part of the questionnaire to testify the hypothesis.

C. how to generate output

We implemented the RNN model with two different techniques, word based and character based. In the later sections, we will compare the outcomes from the two techniques and suggest possible future improvements.

D. Algorithm design

The flowchart of the algorithm is shown in figure 1.

1) N-Grams: We implemented a generative unsmoothed ngram model and trained it on the short joke dataset. We used trigrams as our base and backed off to bigrams and unigrams if no trigrams were found.

2) Word Based: Word based RNN use the gensim word2vec model to translate each word into a feature vector, which was later used to train the RNN model. Feature vectors are used to represent word meanings in multi-dimensional space. This is helpful because words can then be compared with cosine distance or other numerical metrics. For example, the cosine distance between “man” and “woman” will be much shorter than that between “man” and “stone”. In our word based approach, feature vectors are used as the input to train the RNN which generate more feature vectors as output. The output vectors are then converted to the closest word in the gensim model to generate sentences.

3) Character Based: The character based RNN feeds characters rather than words into the model to train, and then generates output based on previous characters. As a result, it often produces words that have typos. While the word based format may be faster and slightly more accurate if done well, the character based approach is more flexible, and the minor typos can sometimes add to the humor. The RNN’s LSTM cells help it to capture short structures of jokes, but because the memory is too short term it can’t build up to a punchline. As a result, it often diverges on tangential ideas before it produces a joke. Figure 2 shows a template for a RNN framework. There are input layers, hidden layers composed of LSTM cells, and output layers.

E. Scoring metric

We will select random jokes from the human-generated corpus and mix them with jokes generated by our system. We will then have people rank each joke on a scale of 0-5, 0 being not funny at all and 5 being very funny. We will then take the average rating of the generated jokes and the human-made jokes and compare them.

F. Testing against hypothesis

The questionnaire will be interwoven with jokes from the training dataset and the output from our RNN generator. Without knowing the origin of the jokes, testing subjects’ ranking to the jokes will be unbiased. We will then summarize and compare the results to check whether the rating difference is within 30%, the range stated in hypothesis. We predict that the average score
of the jokes generated by our system will be within 30% of the average rating for the human-made jokes.

V. Implementation

For this project, three different techniques are used to generate jokes: ngram, word-based RNN, and character based RNN.

A. Phonetic Edit Distance

We attempted to find words that were phonetically similar to an input word by using the Levenshtein edit distance between the phonemes of the base word and those of the 20k most commonly searched words on google. While the algorithm was successful in what it set out to do (it did in fact find the minimum edit distance to each word and sorted them from low to high), we found that the words that it paired together were not always appropriate for pun generation. We experimented with some of the features to increase the quality of the output. For example, we disqualified any words that had the same Porter stem as the base word. We also experimented with different substitution costs and found that a cost of 1 was more successful at generating words that had a possibility of a pun (it would generate more rhymes instead of simply removing phonemes until we reach another word). A further extension may be able to give relative weights between phonemes to give a higher cost to phonemes that were not phonetically similar (and caused somewhat jarring differences when paired with the base word).

B. N-gram

We found that while trigrams alone worked quickly, trying to back off to bigram and unigram would sometimes get stuck outputting gibberish which was also highly inefficient. For the output that we actually generated, we stuck with only a trigram model. If the model ever encounters a set of leading words that it has not seen before, it will exit prematurely. A more efficient implementation would likely be able to utilize the bigram model to its fullest extent and allow for arbitrary input. We also attempted to create a separate trigram model that was trained in reverse so we did not have to start the sentence with our seed words, we could instead generate backwards. We were relatively successful in this but found that the output was less comprehensible than with only going forwards so we opted to keep only the forward trigram generation in our final system.

C. Word Based Approach

The whole word based RNN process is shown in Figure 4. First we parsed the joke database into a array of sentences with each sentence being a sequence of stemmed tokens. We then input the whole database to train the gensim word2vec model [6]. The word2vec model converts word into an array of features, which are then serves as the input to train the RNN. The output vector was then translated back into words to form sentences. Noted that “most_similar” method returns the top words with highest probabilities. And this is the part we could tinker to create varieties.

D. Character Based Approach

The RNN was trained on batches of 50 characters, training through 30 iterations of the whole data set of 20,000 jokes. This took about 13 hours to train. The RNN has an input and output layer of size 97, the number of characters in our training data, as well as 2 hidden layers with 500 nodes each.

VI. Data Analysis and Discussion

Sample outputs from each algorithm are shown in table I

A. output generation

From character-based RNN:

- Why did the blonde cross the road? I don’t know. [Actual joke in training set was “Why did the blonde cross the road? I don’t know. Neither did she!”]
- entity to acterry Marija.
-I’ll post a follow pants. I said the girl in the entire way to the class that I don’t know what it showed up with a slot on my car dress money in the world has a straight bell.
-”My dad said the grocery store in a pajama instead of sandwich says ”Buy, wait, thanks.” I said [Output text of length 300 with input seed of ‘e’]

- Yo momma so fat... They were fat at me, I think they’re pretty good at my kid will talk about how to high five each other in the bathroom.
-”I only do stall those consensessival. I’m not one word...
[Output text of length 200 with input seed of “Yo momma”]

- Chuck Norris refraged to the cold shoots. Somewhere
-”I love when shopping on the floor at a stoal gun..... a said they planed the right person refuse to have a pan and he’s already been posted.”
-”What is a baby’s favorite dance? A banana splitte!
-If you don’t like your birds while I was a little bit [Output text of length 300 with input seed of “Chuck Norris”]

- Knock knock Who’s there? I good them out a huge dust.”
-”Why did the cows keep riding up? It gets talking
-”How to sex for a cookie when i died an elevator, and an app in the store at a store guy shots of weird bottles in the world. Then you should call him a handjob... ...is a very kind about how m [Output text of length 300 with input seed of “Knock knock “]

From ngram (selected output from 10 trials):
- How did your manners die too.
- My friend in North Dakota lawmakers decide life begins at snowball conception.
- How did Jesus say to the other? How is a party.

B. output analysis

As we can see from some of our sample output, our RNN output is pretty bad. It can piece together small phrases based on what it has learned from the joke dataset, but it quickly jumps away into a garbled mess. For example, it understands to put “Who’s there?” after “Knock knock”, but it doesn’t follow it up with a proper punchline. In fact, our current RNN model can’t really produce any punchline. It’s memory of previous words seems limited to just a few words. Part of the problem is the limited time available to train the RNN model as well as the limited amount of data that the RNN model was trained off of. We used only 20,000 jokes out of the several hundred thousand available because of memory constraints (20,000 jokes required 3 GB of RAM to train). With more time, more data, and a more powerful computer, we could probably produce real jokes that actually have the potential to be funny. As it currently is, our RNN model is only funny because of how bad it is. We also tried using a trigram model as well as RNN based on words rather than characters. Table I shows that repetitions are generated by the word based method. This is caused by the RNN convergence, generating stabilized outputs vectors, which later mapped to the same word in the vec2word process.

All 3 of these models did a poor job of generating funny output. However, the character-based RNN seemed to work the best of the 3. Part of this may have been influenced by specific implementations of the various models.

C. discussion

Due to the combination of long training times and poor results, we were not able to test our hypothesis. Since our jokes did not get to the level of making sense, we did not go through with making a questionnaire to give to people. Because of such long training times we had to truncate our dataset and limit our RNN size. If we had more time, we would try using a multi-layer RNN with more than 1000 nodes in each layer. Additionally we could train for thousands of iterations instead of a mere 30.

VII. Conclusions and Recommendations

A. summary and conclusions

The results show that character-based RNN approach is more solid than the word based RNN and trigram solution. There are couple reasons behind this. First, in order to generate humour, the model must retain previous memory of the contexts to generate new words. Trigram model does not have enough information for such tasks. In addition, character-based RNN also eliminates out-of-vocabulary and convergence problems happened in our word-based RNN approach.

B. recommendations for future studies

To further improve the word-based RNN model, we could consider using different number of features to train the word2vec model. In the meantime, we should also dive into the RNN and investigate more on the cause of convergence. Regarding the character-based RNN model, we believe that our results will be significantly better if we were able to train the RNN model for longer and on more joke data. We recommend that future studies dedicate the necessary resources to properly train their neural network. In addition, we weren’t able to properly integrate the many parts of our project like we wanted to because of resource constraints. We believe that a sufficiently trained up RNN could make use of our homonym analysis to try to be more deliberate in creating puns. In addition, when future research starts having meaningful results, they would also benefit from a scoring metrics like the one we proposed.

REFERENCES

[3] He Ren and Quan Yang, Neural Joke Generation. Stanford University, 2017
TABLE I. OUTPUTS FROM DIFFERENT MODELS

| Model            | Seed: (my dog) output: my dog: "spits out coffee" verb is a porn star. seed: (i went) output: i went to the local grocery store and buy something that bleeds for five minutes. | seed: (i could tell you a black joke but you) output: i could tell you a black joke but you damm damm sniffles sniffles sniffles sniffles sniffles sniffles sniffles sniffles | seed: Yo momma output: Yo momma so fat... They were fat at me, I think they're pretty good at my kid will talk about how to high five each other in the bathroom. n't only do stall those consensessival. I'm not one word... O

---

Fig. 4. Flowchart for word-based RNN model