DadBot: Final Report

Liza Bialik rbialik@umass.edu
Andrew Cunningham andrewcunnin@umass.edu
Cerek Hillen chillen@umass.edu
Russ Ronalds rronalds@umass.edu
Linnea Ross lross@umass.edu

1 Problem Statement

Computational humor generation is a difficult problem in Natural Language Processing (NLP). It requires a deep understanding of multiple layers of linguistic hierarchy, from phonology all the way up to pragmatics. Otherwise state-of-the-art transfer-learning models fail on this task because of a combination of low-quality training data, and that a certain portion of the required language understanding resides below the layer on which they're trained. Language models of this kind are not directly aware of pronunciation; their closest approximation, orthography, famously does not match pronunciation. Research into this domain requires more structured approaches to humor generation, harkening back to traditional NLP pipelines.

2 What You Proposed vs. What You Accomplished

We sought to construct a structured approach to one problem in the domain of computational humor generation: pun generation. We allocated 2 milestones worth of time to create a minimum viable product of DadBot, which would perform the bare minimum task of producing puns from input topic and sentence pairs. We allocated the remaining time to analysis, fine-tuning, and exploratory work into other methods of pun generation.

We completed the minimum viable product on time for milestone 2. However, DadBot's puns at that time were lackluster, replacing words with other words for no discernable reason. We spent some time fine-tuning parameters, and eventually discovered and fixed a bug in our phonological distance metric. After this, DadBot's puns improved significantly.

During our original proposal and our first milestone, we proposed integrating neural pun generation into our project. Due to the lack of a quality pun dataset, we pivoted a little and chose instead to use existing Language Models (BertModel and MertForMaskedLM) to re-rank DadBot's potential pun candidates. We did this by maximizing a surprisal score based on relationships between local surprisal, global surprisal, and overall fluency.

3 Related Work

There is a long history of computational humor, both in generation and analysis. Specifically, puns have been computationally treated since the 90s (Binsted and Ritchie, 1994), and the analysis and generation of puns continues in the present day.

Graeme Ritchie, a co-author with Binstead on the Joke Analysis and Production Engine in the 90s, has been working on computational humor for decades and advocates for computationally cheap models that can produce "contextually integrated puns". He provides many conceptual outlines for solving "the non-trivial problems" of phonetic similarity and context (Ritchie, 2005). In this paper, Ritchie discusses a system which uses complex search to generate candidates for self-contained puns. The program would look at various candidates based on a core knowledge base. The paper focused on low-cost search across a given domain of knowledge, constraining it to templates and schema, whereas our project sought to make a more flexible model.

There have been various methods put forth on how to detect, disambiguate, and locate puns using word sense, phonology and phonetics (Cai

et al., 2018; Jaech et al., 2016; Mikhalkova and Karyakin, 2017). These papers and others give indicators of how to reverse that process and generate puns, such as using the CMU Pronouncing dictionary (Doogan et al., 2017), high semantic similarity (Hurtado et al., 2017; Miller and Gurevych, 2015), and surprisal (He et al., 2019). Our project borrows and combines many ideas from these other papers, such as our computations with phonetics and our three-front approach to pun generation (phonetics, semantics, and surprisal). By looking over these papers, we were able to narrow our scope and focus the features we cared about for our model. While the phonetics and semantics have pretty straightforward applications, we were particularly inspired by the 2019 He paper, which looked at patterns of surprisal in puns. In our final Bert model for this project, we attempted an implementation of He's work to optimize for global and local surprisal.

Early pun generators used templates that were manually built (Binsted and Ritchie, 1994; Hong and Ong, 2009), which are difficult to integrate outside of the original environment. Recent state-of-the-art neural approaches aim to be more integrative, but they still seem to output results that are judged poorly by humans. Even so, they provide methods for evaluation of puns, as well as a more computationally costly method to try to beat with a simpler language model. (Yu et al., 2018) provide a model that can take a polysemous input word and produce a sentence that plausibly contains two senses of that word, but that sentence does not yet contextually fit into any space. (Luo et al., 2019) extend this model to reward sentence ambiguity in language generators. While we have not yet made use of the ideas from these papers in evaluating our pun generation, they helped to inspire our projection of future work by providing us with ideas on how to evaluate a model's output without having access to high volumes of quality data.

In this current work, we aim to use all of this learning but try to do so with a transparent, flexible model that isn't built on templates.

4 Your Dataset

We explored four main datasets, three of which we ended up using in our implementation. The datasets were GloVe word embeddings, the CMU Pronouncing Dictionary, a phonological feature mapping from the Linguistics 402 curriculum, and a corpus of puns which we decided against using due to the limitations in size and quality.

We used 200-dimensional GloVe embeddings trained on data from crawling Twitter. We used these embeddings to determine a pool of words related to the topic word and to calculate a metric of their similarity. Because these embeddings were trained on Twitter, some of the resultant semantically similar words ended up having somewhat untraditional relations to the topic word and often times were a bit of a stretch in relating the pun. This is also a result of GloVe embeddings storing the degree of relatedness between two words but failing to hold the information of what the relationship is. We will discuss these shortcomings more in error analysis.

We also used the CMU Pronouncing Dictionary, often referred to as CMUdict. CMUdict is a pronunciation dictionary that allows one to map words from their orthographic forms to their phonemic forms.

Lastly, we used a phonological feature mapping from the curriculum for Linguistics 402, Speech Sounds and Structure, here at UMass Amherst. This mapping is a set of all phonemes with a breakdown of their features. Each phoneme has the same features but with either a value of -, 0, or + depending on how that phoneme represents that feature, with - meaning negative, 0 meaning the feature isn't applicable, and + meaning positive. This allowed us to develop a more precise calculation for the difference between phonemes, and gave us the means to assign custom weights to each phoneme in the phonological distance calculation.

We came across a corpus for existing puns, but upon further inspection decided the quality of the puns was fairly low, and it was likely too small to successfully train a neural pun generator language model (2250 one-liners). Instead we made the choice to design DadBot in using a more structured approach that leverages knowledge of features of language as well as features of puns to generate its own datasets. Data Preprocessing

Although not strictly speaking data preprocessing, we found that loading our word embeddings every time we restarted our application elongated feedback process. We moved our word embeddings from our main application (main.py) to an auxiliary server (server.py) that serves word similarity requests over HTTP. While developing and fine-tuning hyperparameters, we kept the server alive and only restarted our main application. This reduced startup time from several minutes down to several seconds, and allowed us to more quickly test changes.

The data for feature vectors was originally provided to us in the form of an Excel spreadsheet. We mapped from that representation to a key/value pair representation, where each phoneme mapped onto a collection of its features. We then eliminated all non-English phonemes from this dataset. We replaced the keys, which were from the International Phonetic Alphabet, with the strings used by CMUdict to represent pronunciations. In CMUdict, diphthongs are given a single character that represents their combination, rather than multiple vowels in sequence. Whenever CMUdict produces a pronunciation at runtime, we map all diphthongs in its phonemic form onto the appropriate pair of monophthongs.

5 Your Approach

5.1 Overview

As mentioned in our Problem Statement, computational humor generation is a difficult problem domain in part due to its lack of quality training data. Instead of trying to construct learning models that would perform well on this relatively small amount of data we opted to approach it from a structured perspective. Our approach relies on a hypothesis around the quality of puns of the form "¡antecedent sentence¿, more like ¡punned sentence¿." We assume that puns are of higher quality when they phonologically related to the antecedent sentence and semantically related to some contextually salient topic. We take the antecedent sentence and contextually salient topic in as inputs, and from them produce a punned sentence where we replace one or more words to minimize a joint phonological and semantic distance function.

5.2 Phonological Distance

In the study of phonology, the most basic atomic unit of meaning is the phoneme. These correspond to individual utterances of sound, like the "k" sound in the onset of "cat." Each phoneme is composed of an array of features, each of which corresponds to a single parameter by which a sound can contrast with another. For example, "voice" determines whether or not the vocal cords are active during articulation, and "nasal" determines whether or not the sound is articulated via the nasal cavity. In general, phonemes perceived closeness corresponds to the portion of their features that they share. Take for instance, the "t" sound as in "tap", the "d" sound as in "dark", and the "h" sound as in "hat." Here "t" and "d" sound more similar because the differ on only one feature, voicedness. Meanwhile, "h" is more different from both of them because it contrasts on several features, like its place of articulation, its aspiration, and others. Each feature has the value "-", "0", or "+," corresponding to lacking, being non-applicable, or having some characteristic. For example, +voiced entails that the vocal cords are activated during a phoneme's articulation.

We leveraged this phenomenon in our project, by calculating the phonological distance as a min-edit distance over their features. To do so, we mapped the orthographic forms of words onto their phonemic forms with CMUdict, a structured database of pronunciations. CMUdict provides several pronunciations per word, some spanning dialects of English, so we naively chose a single pronunciation to use for our edit distance. For each phoneme in a phonemic form, we mapped it onto a vector of its features using data provided to members of our team by the instructor of LING 402, Speech, Sounds and Structure.

We calculate the minimum edit distance on this array of feature vectors representation of words. We consider inserting and deleting a phoneme to cost 1.0, where replacement is the sum of piecewise feature distance normalized to the range [0.0, 2.0]. This form of edit distance allows us to capture fine-grained differences in the pronunciation of words. For example, we would want to capture that "leave" and "leaf" are perceived to be more similar than "leave" and "league," despite the fact that they are differentiated by only one phoneme

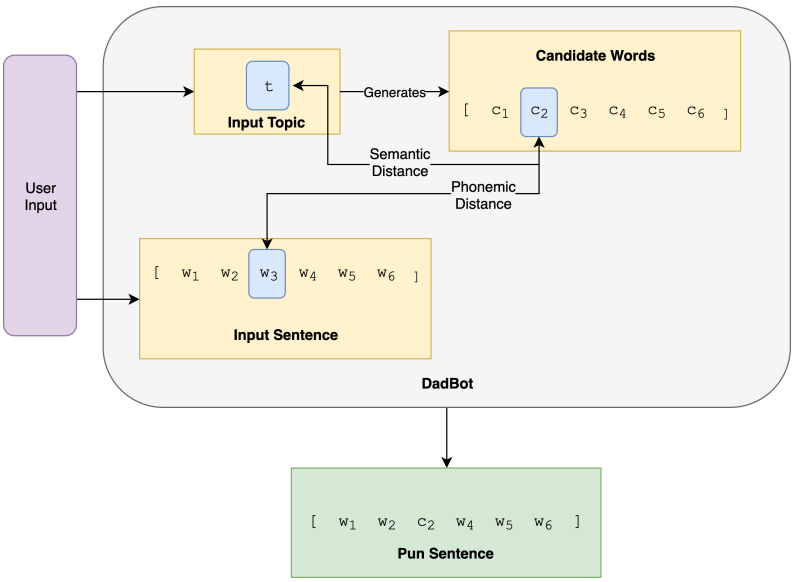

Figure 1: System Diagram

each.

5.3 Semantic Distance

Calculating semantic distance in Natural Language Processing is often done by applying some distance function to some word vector representation. We decided to abstract away the technicalities of training and comparing word vectors, and instead relied on GenSim's word vector implementation loaded from pre-trained 200-dimensional GloVe embeddings trained on a dataset sourced from Twitter. GenSim's common word vector class provides an interface to collect the top N most similar words, where N is a parameter to the function. We used this to collect candidate words with which we replace words in the input sentence and to measure the similarity between those candidate words and the topic word. This similarity metric resides within the range [0, 1]. We invert it, by taking (1 - similarity), and normalize it so that for any query the most similar word has a distance of 0, and the least similar word has a distance of 1.

We also experimented with other word embeddings to varying degrees of success. We first started by trying to train word embeddings on small corpora using GenSim, but we found that they were disatisfactory for this task. We moved on to trying Facebook's fastText, but no member of our group has a computer sufficiently powerful to load it into memory, and so we could not move forward with it. Finally, we settled on GenSim's pretrained GloVe embeddings. We originally had used a lower-dimensional embedding, but found that higher-dimensional embeddings were qualitatively providing us better candidate words.

5.4 Combination

We combine our phonological and semantic distance functions as the weighted sum of each distance. We allow the weights to be hyperparameters of the model, which lets us experiment with weights for each distance.

By default, this combination weights phonological distance more than semantic distance. Phonological distance is positively unbounded. In the worst case, where the optimal edit would be to replace each feature with the maximally different feature, the return value is 2 * n, where n is the length of the strings. Meanwhile, the semantic distance, as mentioned above, is normalized and bounded to the range [0, 1]. We found that this relationship works well for pun generation. Even words that are deemed unsimilar to the topic word often have some relation, save for certain examples that are enumerated in Error Analysis.

Originally we identified a single word in the sentence to replace with a single word in the candidate set produced from our word vectors such that this pair has the minimal combined phonological and semantic distance metric. We replaced that word, and returned the punned sentence.

5.5 Fine-Tuning

On the phonetic front, we increased the cost of insertion/deletion of syllabic phonemes because they more significantly impacted the sound/length of the word. We also tinkered with the weights of other features like continuants, syballants, and different places of articulation but found no consistent improvement.

We found that increasing the number of semantic neighbors to consider from 50 to 200 helped improve results, as it increased the chance of finding a more similar sounding word and the GloVE embeddings were usually still relevant at that distance. We linearly mapped the similarity score of the top 200 words from their range, usually [0.6, 9.5], to the range [0, 1]. This let us more drastically downweight words that are more semantically distant to the topic word. This change slightly improved performance, showing that the model produced puns with non-sencial replacements less often.

6 Implementing Feedback

6.1 Threshold Replacement

During the poster presentation, Mohit presented the idea of a threshold-based replacement mechanism, rather than a percentage or count based replacement mechanism. As explained in Combination, above, we originally replaced the single most optimal word with its candidate replacement. We have now changed the implementation from replacing exactly one word per sentence to replacing all words with a pun candidate above a certain threshold. We decided the threshold by comparing DadBot's rankings with that of human judges, and chose one that tended to result in average puns (human-rated) without being so selective that not producing a pun was more likely than producing one. At the moment, this threshold is set to 1.2.

6.2 DadBERT

DadBERT is an extension of DadBot that re-ranks DadBot's proposed pun sentences using BERT. In order to do this we restructured the code to allow for an optional re-ranking mode that can be toggled with a command line argument "rerank". When in rerank mode, DadBERT asks the user to provide a "context sentence," where the topic word is used in a sentence. This serves as the set-up to the punchline. In rerank mode, the DadBot model produces a list of potential puns (all above a threshold) instead of just returning a single "best" pun. This threshold (1.7) is a little looser than the one used to decide which word to replace, in order to give DadBERT a more options to consider. It then passes this list of puns to a ReRanker object that chooses a best candidate pun and returns it.

The re-ranking decision process draws on the idea that the quality of a pun being related to the ratio of global and local surprisal discussed in (He et al., 2019). There they describe a good pun as having a higher local surprisal than global one. For example, in the pun: "Yesterday I accidentally swallowed some food coloring. The doctor says I'm OK, but I feel like I've dyed (died) a little inside." The word "dyed" in the local context, "I feel like I've dyed a little inside", is more surprising than it is in the global context of the entire joke, because the first sentence provides relevant context to the word "dyed", mainly "food coloring". In the case of DadBERT, we explicitly ask for a context sentence to serve as the first (setup) sentence of the joke, as well as a sentence to punnify, or make the punchline out of (which serves as the local context). The ReRanker makes its decision from two calculations:

1. The first compares the surprisal of the pun in a local and global context. It uses a BertForMaskedLM model to estimate the surprisal of the pun word in the local and in the global context, where the local context is the pun candidate sentence and the global is the concatenation of the context sentence and the pun candidate sentence. It then masks the pun word in the sentence. More explicitly, for the pun "ocean" + "what're you doing here" $\rightarrow$ "water you doing here" the inputs to the model in a local context are:

   "[CLS] [MASK] you doing here [SEP]"

   and for a global context:
   "[CLS] swimming in the ocean [SEP] [MASK] you doing here [SEP]"

   It estimates surprisal by getting a probability distribution of words at the masked position from BERT, and taking the inverse probability of the pun word. We expect to see a

higher surprisal measure without the context sentence than with it, and report the surprisal score of that pun surprisal_without_context : surprisal_with_context.

2. The second compares the surprisal of each of the potential punned sentences in a global context. For each potential punned sentence, it uses a BertModel to compare the inverse probability of the punned sentence following the context sentence. More explicitly, for the same "ocean" pun above (for each potential pun word, in this case "water") it is given:

   "[CLS] swimming in the ocean [SEP] water you doing here [SEP]"

   We expect to see better pun candidate sentences to have a higher probability of following the context sentence both because of content relevancy as well as a measure of fluency. A more fluent pun is more likely to follow any sentence, so we use this metric in part to penalize disfluent sentences (in general) as pun candidates.

We take a weighted sum of these two scores to generate a surprisal_score, and re-rank all the candidate puns accordingly. We return the pun candidate with the highest score.

7 Results & Evaluation

Because our model of pun generation is structured, rather than learned, we do not have a loss function upon which we can base our effectiveness. Instead, we constructed and disseminated a Google Forms survey (pun, 2019) to respondents (n = 120) to determine the quality of DadBot's puns on a scale from 1 to 5.

We also evaluated DadBot's predicted score, as a normalized inverse cost function scaled to range from 1 to 5. We compared these two numbers for each pun, and present them in Fig 2. Our respondents reported an average pun quality of 1.965/5.000. DadBot expected an average pun quality of 3.195. DadBot was overconfident by a margin of 1.230 points.

This result presents an open question in this area of research: could one learn more about the underlying structure of generating puns by matching predicted quality to real quality? This lies more in the domain of a discriminative model, which would evaluate the quality of a joke, but may provide direction towards generating high quality jokes as well.

8 Error Analysis

A majority of the issues we ran into were related to the semantics of the replacement word. Because we used non-contextualized word embeddings, rather than sense embeddings or contextualized word embeddings, certain words had a diverse and unexpected set of similarities. This was especially true for polysemous words and words that are part of idioms. This creates two problems for pun generation:

1. Polysemous topic words may generate puns that leverage unexpected senses

2. GloVe similarity does not necessarily denote humor-related semantic similarity

For the first issue, a word that may have different meanings in different contexts will have higher similarity scores across all contexts. For instance, in one example we used the topic word "fall" which an English speaker would likely assume to be the season. However, its GloVE embedding has a high similarity score to "love" because of the phrase "fall in love," which resulted in a pun that had no clear link to the topic (i.e. "leave me alone" $\rightarrow$ "love me alone"). This pun was rated poorly by participants in our survey because they had a strong preference for "fall" the noun over the verb. In a trial with the same sentence, but with "autumn" as the topic, Dadbot changed "leave" to "leaf" (i.e. "leave me alone" $\rightarrow$ "leaf me alone") and received a high score from participants, matching its high confidence score.

For the second issue, two words occurring similarly does not necessarily form a direct mental link between them that would be conducive to a pun. For instance, "good" and "bad" may have very high similarity scores by our metric, but replacing a word with "bad" would not sit in my mind as a pun relating to "good." We saw this in our survey when the topic "banana" turned "money" into "honey" with a high confidence score that this would be a good pun. However, the people who were surveyed pretty uniformly

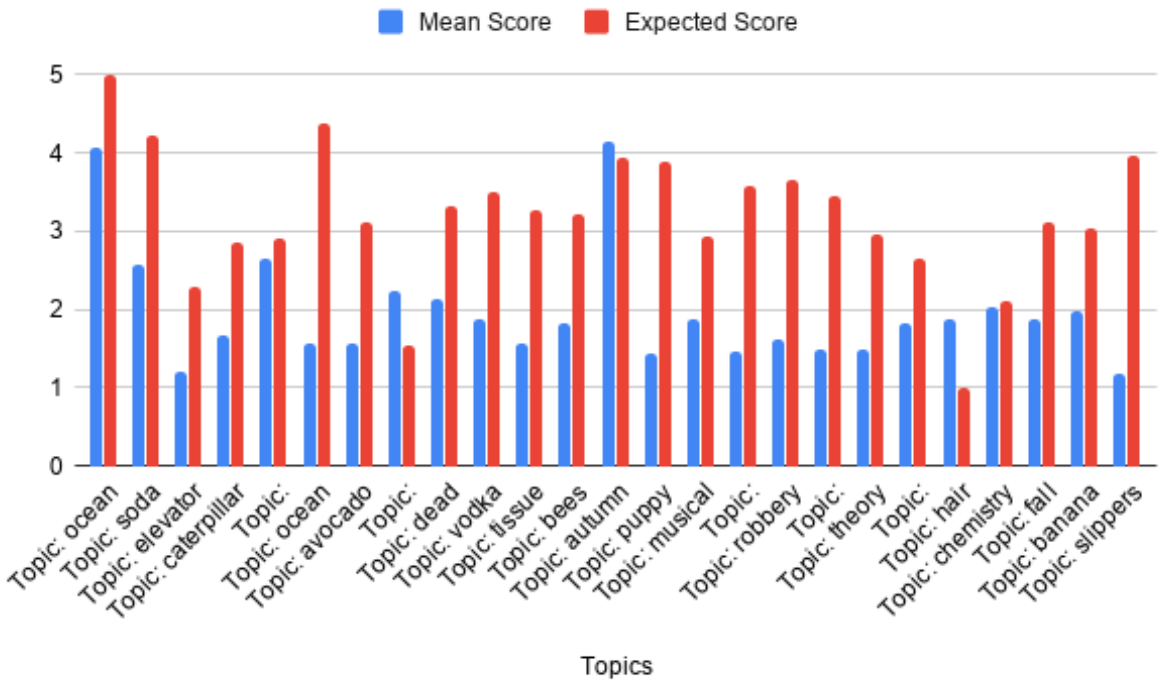

Figure 2: Real vs. Expected Ratings

rated it as being bad, and suggested ”monkey” might’ve been a better choice. This is because, while someone may tweet about the smoothie they made with banana and honey, they are not directly related and we unfortunately did not have a means of gauging what that specific relationship is.

In terms of improvement with DadBERT re-ranking, ”leave me alone” got turned into ”leaves me alone” and ”left me alone”, for ”autumn” and ”fall” respectively. While some may still prefer ”leaf” to ”leaves”, we think ”leaves” is actually a more clever choice as it maintains grammaticality and is punny at the same time.

However, DadBERT brings with it its own errors, like a hyper-sensitivity to the specific structure of the context sentence. For example, controlling for everything else, DadBERT gave significantly different ratings to a pun in which the context sentence started with ”and then” than a pun in which it didn’t. To account for this, we down-weighted the similarity score that is responsible for this measurement, and up-weighted the surprisal score in the reranking funciton.

9 Contributions of Group Members

We have enumerated group members’ contributions in alphabetical order of last name.

9.1 Liza Bialik

1. Pre-processed phonological data
2. Fine-tuning
3. Threshold replacement
4. DadBERT implementation

9.2 Andrew Cunningham

1. Research into word embedding choices
2. Implemented phonological distance subsystem
3. Research into homophonic pun structure

9.3 Cerek Hillen

1. Pre-processed phonological data
2. Implemented semantic distance subsystem
3. Pun generation scheme for minimum viable product DadBot
4. Generated survey, and analyzed results
5. Wrote tests

9.4 Russel Ronalds

1. Research into word embedding choices
2. Read & summarized related works

9.5 Linnea Ross

1. Research into word embedding choices
2. Research into related work
3. Collected puns for survey
4. Collected responses from the survey

10 Conclusion

While we started with an almost purely structural project, with very little learned information, DadBot is now a novel hybrid application of domain knowledge and deep learning. Our group's prior experience in linguistics has provided us the knowledge to apply a structural approach to the phonology of homophonic puns. This let us drastically simplify this section of the project, and let us ensure our puns were, at the very least, phonologically similar. This also allowed for very explainable results that we were able to analyze. Here we relied on our group members' exposure, both from this class and prior work, to learning methods to construct our semantic distance metric.

With instructor feedback, we also integrated some amount of deep learning into our project, by using BERT to dynamically re-rank puns generated by DadBot. This is covered in more detail in section 6.2 above. Although we did not have time to evaluate DadBERT pun performance against human judgement the same way we did with DadBot, from anecdotal evidence it seems to produce more fluent sounding and still relatively funny puns than DadBot on its own.

Finally, as discussed in section 7, comparison between expected and real pun performance raises an interesting question about finding a hypothesis of pun quality that allows a model to predict the quality of a pun. In a way, this is a separate way to evaluate the correctness of BERT. Although it could not consistently produce quality puns, the result around its ability to predict the quality of puns, despite its overconfidence, is valuable.

11 Future Work

Although we already budgeted in time to our project to work on improvements to DadBot, there is still much to do. The domain of computational humor generation is sufficiently deep that there are myriad ways we could take the project. As mentioned above, we have already explored the use of BERT in ranking generated sentences with metrics inspired by global and local surprise metrics in Hehe et. al (He et al., 2019).

We're also interested in testing out a similar structure with GPT2, instead of with BERT. GPT2 more directly exposes a language model, rather than internally having trained on a masked language model, and so may be well suited to measuring surprise and fluency. We would still need to do substantive research into GPT2 before moving forward.

In our early milestones, we mentioned that we were interested in using a pun structure where the user provided the model two sentences, in the form "antecedent sentence. pun candidate sentence." The model would then infer a topic from the antecedent sentence, and use it in the pun candidate sentence. Although we're still unsure how to identify a good operative word from an antecedent sentence, this structure would lend itself well to metrics of global and local surprise, as well as offer a more intuitive and coherent user experience better suited for, say, a chat-bot environment.

References

(2019). Pun quality survey. https://drive.google.com/openid=1RIewmk1BxrRl8Iohsl4P7Up0FgsPbcyO7gVgNDBmpJU.

Binsted, K. and Ritchie, G. (1994). An implemented model of punning riddles.

Cai, Y., Li, Y., and Wan, X. (2018). Sense-aware neural models for pun location in texts. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 546–551, Melbourne, Australia. Association for Computational Linguistics.

Doogan, S., Ghosh, A., Chen, H., and Veale, T. (2017). Idiom savant at Semeval-2017 task 7: Detection and interpretation of English puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 103–108, Vancouver, Canada. Association for Computational Linguistics.

He, H., Peng, N., and Liang, P. (2019). Pun generation with surprise. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1734–1744, Minneapolis, Minnesota. Association for Computational Linguistics.

Hong, B. A. and Ong, E. (2009). Automatically extracting word relationships as templates for pun generation. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 24–31, Boulder, Colorado. Association for Computational Linguistics.

Hurtado, L.-F., Segarra, E., Pla, F., Carrasco, P., and González, J.-Á. (2017). ELiRF-UPV at SemEval-2017 task 7: Pun detection and interpretation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 440–443, Vancouver, Canada. Association for Computational Linguistics.

Jaech, A., Koncel-Kedziorski, R., and Ostendorf, M. (2016). Phonological pun-derstanding. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 654–663, San Diego, California. Association for Computational Linguistics.

Luo, F., Li, S., Yang, P., Li, L., Chang, B., Sui, Z., and Sun, X. (2019). Pun-GAN: Generative adversarial network for pun generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3386–3391, Hong Kong, China. Association for Computational Linguistics.

Mikhalkova, E. and Karyakin, Y. (2017). Detecting intentional lexical ambiguity in english puns. *CoRR*, abs/1707.05468.

Miller, T. and Gurevych, I. (2015). Automatic disambiguation of English puns. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 719–729, Beijing, China. Association for Computational Linguistics.

Ritchie, G. (2005). Computational mechanisms for pun generation. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*.

Yu, Z., Tan, J., and Wan, X. (2018). A neural approach to pun generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1650–1660, Melbourne, Australia. Association for Computational Linguistics.