LMAONet - LSTM Model for Automated Objective Humor Scoring and Joke Generation

Zen Simone
Computer Science
Stanford University
zsimone@stanford.edu

Cameron Cruz
Symbolic Systems
Stanford University
camcruz@stanford.edu

Abstract

It is significantly difficult to objectively evaluate the semantic quality of a natural language generation model. Specifically for joke generation, there is no reliable, objective metric for measuring how "funny" a model can be. Often, researchers will settle for collecting human ratings on generated jokes via Amazon MTurk surveys, which is highly subjective and not reproducible. We propose that Reddit upvotes/downvotes for a particular joke are a sufficient proxy for its humor quality, and found that a small convolutional neural network is able to learn to score the quality of a joke in this way. We apply this network to objectively evaluate the humor quality of state-of-the-art joke generation architectures and seek to improve upon existing architectures by incorporating this humor metric into a custom loss function. We find that, not only are we able to successfully extract a meaningful tier-based metric from Reddit data, we are able to predict these ratings with almost a 90% success rate after tuning. Further work is warranted to fully explore the viability of using this network during training to calculate additional humor loss.

1 Introduction

Humor is an essential facet of human personality. No two people have the exact same sense of humor because of it's high subjectivity. Often what makes a statement funny is context, whether that is based on an individual's past experiences or the situation in which the statement is made. With simple jokes, there are two stages: the setup and the punchline. Localized context is created in the set-up and the punchline delivers a humorous commentary about that context. Though everyone's sense of humor is different, the compartmentalized nature of jokes often makes their comedy accessible to a variety of people. By investigating how large amounts of people rate jokes, we propose that we can predict what jokes will be funny to most people. The internet is a vast source of jokes, and forums such as Reddit allow us to also access how a large population responds to each joke. By sourcing jokes from Reddit, we have access to labelled data in the form of upvotes, which circumvents the need to actively source people to rate thousands of jokes.

The ability to automate this task would dramatically speed up the rate at which jokes can be evaluated and has the potential to be used to train joke generation networks to be more humorous. Traditionally, jokes are evaluated with human raters. This limits the amount of jokes that can be evaluated in a short period of time and biases measurements towards the evaluator's respective sense of humor. Our proposed solution is a convolutional neural net that can solve both of these problems by leveraging compute power for speed and a large amount of jokes to develop a broad sense of what people find funny. Our goal is for this network to be used at test time to evaluate joke generation networks objectively and efficiently.

Through our experiments, we developed a custom humor metric based on Reddit upvote/downvote scores. We experimented with neural network architectures to model this humor metric, and the performance of the model on unseen jokes correlated with explicit human ratings we collected in

parallel. In addition to experimenting with multiple text generation architectures, we demonstrate it is possible to incorporate the humor scoring model into loss functions, but extensive hyperparameter tuning or alternative optimization algorithms are needed for convergence.

2 Related work

While researchers continue to develop new models for natural language generation, the methods of evaluating the semantic quality of sequences that models can generate remain inefficient and highly subjective. Specifically for joke generation, the humor quality of novel architectures is still evaluated with human ratings.

Ren et. al developed a novel word-level joke generation architecture that attends to pre-extracted topic words (usually proper nouns), which inspires our custom word-level architecture (detailed in Approach). This network was trained on a dataset of jokes written by Conan O'Brien and was able to successfully produce humorous commentary on specific topics. Ren et. al report that they outperform probabalistic models of joke generation based on human evaluation since "there is no reliable automatic evaluation on the joke generation task." [6]

In parallel, Chippand et. al developed a novel word-level LSTM that was trained on a dataset of jokes, quotes and tweets. Each input sequence had a context label of which category it belonged to, and the network learned to generate text that belonged to a specific category. While the network was evaluated on its syntactic accuracy and similarity of generated samples to the training set (to evaluate if it could generalize beyond the training set), the quality of humor of the generated jokes was evaluated via the opinions of the researchers themselves[1].

This common lack of objective methods for humor evaluation stood out to us since arguably the most important metric for a joke is how funny it is. Yet, even in experiments at Microsoft RD, the humor aspect was evaluated subjectively. This prompted us to explore methods for developing an automated objective metric for measuring joke humor, and see if we could incorporated this metric within a novel architecture to improve upon modern joke generation architectures.

3 Approach

Our approach involves two kinds of models tackling different parts of this problem space. The first is a small neural network that is trained to score the humor quality of a joke. We believe that Reddit upvotes/downvotes can be used as a sufficient proxy for estimating how funny a joke is. Since there are 15.7 million subscribers to r/jokes alone, we believe that these subreddits make up a, mostly, representative sample of people's joke preferences and may allow us to determine what jokes are generally more funny than others. Additionally the ability to downvote on Reddit is a feature that many other social media sites do not have, but gives insight into jokes that succeed in general versus more controversial or less funny jokes.

The second is a joke generation model responsible for producing intelligible, humorous jokes. We first implement a baseline joke generation network based on existing architectures from other researchers. We then evaluate the humor quality of the jokes this baseline network can generate using our humor scoring model to demonstrate how our novel objective metric can be applied. We then seek to improve upon modern joke generation architectures by implementing a joke generation network that uses a custom loss function. We propose this custom loss function should incorporate a humor term computed using the humor scoring model, modeled after the GAN loss function for neural style transfer:

$$\begin{aligned} L_{\text{total}} &= \alpha L_{\text{cat. crossentropy}} + \beta L_{\text{humor}} \\ L_{\text{humor}} &= -\left(humor_{\text{generated}} - humor_{\text{target}}\right) \end{aligned} \tag{1}$$

Intuitively, the model is penalized for generating text that isn't as funny as the target. The model is also rewarded for learning to be funnier than the target.

The following subsections provide a deeper look into both of these models.

3.1 Humor Scoring Model

Our humor scoring pipeline takes jokes as input and outputs a score from 0-4, where higher scores are funnier jokes. Our model is entirely custom (see Figure 1.) and relies on an initial embedding layer that trains from scratch. Next it utilizes a 1-dimensional convolutional layer with 128 kernels of size 3. This layer utilizes a sigmoid activation function, which signals whether or not a particular feature is present in the input joke. The kernel size is 3 words which allows the network to learn features based on phrases of this size. The convolutional layer feeds into a maxpool layer, which allows the next dense layer to learn only on the features with the highest activation, which intuitively are the most important features. The model then utilizes a dropout layer, with a 50% dropout rate, to combat overfitting. Lastly, a dense layer outputs a softmax distribution which corresponds to the probabilities that the input joke is in each tier. The argmax of this distribution is the model's prediction and, during training, loss is measured with categorical crossentropy with the softmax distribution versus the one-hot, target vector corresponding to the correct score.

Our reasoning for using a CNN rather than an RNN stems from the fact that our model's task is classification. By using a convolutional layer it leverages the ability to look for features across the entire joke at once and learn phrase representations that correspond with a specific score.

3.2 Joke Generation Model

3.2.1 Word-Level Joke Generation Model with Topic Word Attention

First, we implemented a custom word-level joke generation network. The architecture is based on the LSTM encoder-decoder model with attention used by Ren et al.[6]. In their network, they use a POS tagger to extract the proper nouns within a joke as a set of "topic words" and have the model attend to these words at each timestep. The topic words' embeddings are also averaged together as $\frac{1}{k}(e_1 + e_2 + ... + e_k)$ and used as the initial state of the decoder LSTM. In our variation, we extract any nouns as topic words, rather than just proper nouns, and input a maximum of T topic words to the network (if the joke has less than T nouns then padding words are used for the rest). The topic words are encoded using pretrained GloVe embeddings and averaged as ($e_{\text{topic word avg}}$). This averaged embedding is then projected into the dimension of the LSTM hidden state so that the embedding size does not have to be the same value as the number of hidden units in the LSTM. This projection is passed into the decoder LSTM as the initial hidden state. The input sequence to the model includes only a start token, and the target sequence is the original joke. At each time step t the LSTM utilizes an additive attention mechanism to attend to the topic words when predicting the next word. See Equation 3 in the Appendix for a detailed explanation of topic word attention, and Figure 2 for a visualization of this network architecture.

3.2.2 Fine Tune GPT-2

Second, we experimented with fine tuning GPT-2 on a dataset of jokes. GPT-2 is the successor to GPT from OpenAI, which predicts the next word given all of the previous words in a text. GPT combined transformers and unsupervised pre-training to develop a language model that can be fine tuned in a supervised learning context to achieve state-of-the-art results on a variety of natural lanaguage tasks. GPT-2 extends GPT by being trained on 10X the amount of data and having 10X parameters (around 1.5 billion total). OpenAI is wary of the potential malicious implications of GPT-2 so they released a significantly smaller version for researchers containing only 117 million parameters, which is what we are fine tuning. [4,5]

In a subsequent experiment, we also modify the loss function used, which is just vanilla categorical cross-entropy, to incorporate the custom humor loss term according to Equation 2. We hypothesized this would enable the network to learn to generate sequences that are just as funny or possibly funnier than what it was trained on.

3.2.3 Fine Tune Character-Level Text Generation Model

Lastly, we experiment with fine tuning a pre-trained character-level text generation architecture called textgenrnn by Max Woolf. The network takes a sequence of characters and converts them into char-embedding vectors, which are fed into 2 LSTM layers consecutively. The outputs of the

second layer are fed into an Attention layer, along with the char-embeddings and 1st LSTM outputs via skip-connections. The provided pretrained weights were trained on hundreds of thousands of Reddit submission on a "diverse variety" of subreddits[7].

4 Experiments

4.1 Humor Scoring

4.1.1 Data

For objective humor scoring, we collected our own data using the Reddit API via PRAW. We collected jokes from r/jokes, r/cleanjokes, r/dirtyjokes, r/dadjokes and r/meanjokes, excluding jokes that exceeded 200 characters.

We collected 113,724 jokes in total. The Reddit API's rate limiting only lets us query 1 post per 2 seconds, which was a significant bottleneck.

The Reddit API does not publish a post's number of downvotes, but we have access to the number of upvotes and the upvote ratio. It follows that the total number of votes can be calculated as $total = \frac{\text{upvotes}}{\text{upvote-ratio}}$, which gives $downvotes = total - upvotes$. We also keep track of the average total number of votes across all jokes we collect, which will be used in the calculation of ground truth humor scores detailed below.

Let A be the average number of total votes a post in the dataset has received.

For a given joke, it's humor score label is then calculated as follows:

$$score_{humor} = \frac{upvotes + downvotes}{A} \left(\frac{upvotes}{downvotes} \right) \tag{2}$$

The upvote-downvote ratio $\frac{upvotes}{downvotes}$ is a measure of how funny the joke is. We recognize that the same upvote-downvote ratio can be achieved with different amounts of engagement (total votes), and more popular jokes should be scored higher than less popular jokes. Thus, we weight the score with a ratio between the total number of votes the joke received and the average total number of votes a joke in the dataset received.

To determine the tier of joke, 0-4, we first determine the thresholds for the 20^{th}, 40^{th}, 60^{th}, and 80^{th} percentiles based on our score distribution. From here we simply look to see where a particular joke's score falls with reference to these percentiles and determine the rating to give it. For instance, if a score is below the 20^{th} percentile it receives a 0, if it is in between the 20^{th} and 40^{th} it receives a 1, and so on up to 4.

Further pre-processing included shuffling, augmenting, and, on training, converting the jokes into a sequence of word ids. These techniques will further be explained in Experimental Details (4.1.3) and Results (4.1.4)

4.1.2 Evaluation

We are using a hold out test set of 100639 jokes to evaluate the accuracy of our network versus our generated gold standard scores.

4.1.3 Experimental Details

Since a model to automate objective evaluation of humor has not been done before, we have done many experiments to iterate on the design of our pipeline, including data pre-processing and network architectures.

The scraping script collected jokes one subreddit at a time. This meant that dataset was split into sections of jokes that were similar. Thus, the jokes were shuffled after collection in order to prevent overfitting to a particular subreddits. Additionally, we augment our dataset by creating a copy of each joke and then replacing a verb with a synomous verb. These augmmented jokes were then joined with the original jokes in a new file, approximately doubling our dataset to 207,080.

Before training the model the jokes are loaded from a csv file and converted from strings to sequences of word indices. These word indices are determined by a tokenizer provided in the Keras library fit on all the jokes. Because the average length of a joke was about 18 words long, we decided to truncate or pad each joke to length 20. These index sequences are then the input to all of the tested humor models.

Our Train/Dev/Test split is unusual because we load 90% of the augmented data and then train on 90% of that data and validate on the other 10%. However, we test on the 10% of the un-augmented dataset which corresponds with the unseen portion of the augmented data, ensuring the integrity of the test.

Before training a given architecture on the whole dataset, we try to overfit the model to a small subset of the data to validate our design.

In initial trials we compiled a model that attempted to predict raw scores using a ReLU output and leaky ReLU intermediate activations. This network relied on a 1 dimensional convolutional layer of 256 filters with kernel size 3. This is followed by a maxpool layer, a dense layer of size 50 and the output layer. We have varied the number of convolutional and dense layers between 1 and 3. Our optimizer varied between Adam and Adadelta and these models were run for 20 epochs. Lastly our loss function was mean squared error since we were trying to reduce the difference between our float output and the target score.

After moving to the tier-based rating system we tested 8 different models varying the number of convolutional layers and filters. We also experimented with both 3 and 5 tiers. The 3-tier system has only two thresholds at the 33^{rd} and 66^{th} percentile values. These experiments were run for 30 epochs each, and utilized the Adam optimizer. Table 1 shows the convolutional variations that were done.

4.1.4 Results

The model's performance on our raw numerical data was variable. It was able to overfit small samples of the the data achieving near zero loss, but as we trained on all of the data we noticed that the train loss would approach a double digit score between 10 and 20, while the validate and test loss would stay even higher in the double or even triple digit range. We attribute this issue to one major factor. Posts with far more votes had scores that were often an order of magnitude or two higher.

After the rather unsuccessful results in raw numerical predictions we moved on to the categorical, tier-based model.

Based on the data (see Table 2), we see that models 4 and 5 (see Table 1 for model reference), which each only have one convolutional layer of size 256 and 128 respectively perform the best out of all the models. The performance of the models generally struggles with more than one layer or one layer that has more than 256 filters or less than 128 filters. Since the jokes are short it is expected that the two layer approaches would suffer because the second convolutional layer would be conditioning on sets of the activations from layer 1 of which there are fewer. This may also justify why in the one convolutional layer approaches, more filters did not improve performance after 128 or 256.

Additionally, using a 5-tier rating system was far more successful. It allowed model 5 to reach 0.915 train accuracy and also generalize well to the unseen data, achieving an accuracy of 0.891 on test.

The 3-tier results overall show that the models consistently perform worse than those trained on the 5-Tier system. Furthermore, the test accuracies of model 1 and model 7 seem to show that these models perform better than the models other than model 5, but in fact this could simply be due to the fact that there is not much to separate a tier 0 joke in the 32^{nd} percentile from a tier 1 joke in the 34^{th}. The model was expected to score better in the 5-tier system because of the extra information that is added with two additional tiers, and this appears to be the case based on the results. Figure 5 illustrates model 5's convergence in both of the tier systems, and the noise in the validation accuracy in Figure 3 seems to also indicate the factors discussed above.

Finally, we attribute the relatively low validation scores across both models to the small size of our dev set.

4.1.5 Analysis

While the results are promising in terms of model performance, it was important to then investigate our model's predictions vs the scoring system vs what actual people think. Due to the historically subjective nature of jokes, we had 68 people read 10 jokes and rate them 0-4 to mirror our tier system. The human score with the highest number of votes becomes the "Human Rating". Here are three results from that test:

Joke: I was surprised to hear that Adele uses a macbook. I always thought she'd use a dell
Model Prediction: 0 Score Based on Tier System: 0 Human Rating: 1
On first glance, this example shows two things: The model is able to detect puns that both Reddit and our testing group determined aren't very funny, and that people giving raw scores are less likely rank a joke as 0. Based on the jokes that were voted to be not funny only 1 of 4 was given a 0 while the other three were given 1s supporting the later finding. As for the former, upon inspecting the dataset we find a few other jokes that relate Adele to a Dell computer, but this is the only case where the two appear explicitly together. Thus it is feasible that the model does not register the part of the joke that is supposed to be funny due to a lack of context. This may be what led to the rating it gave rather than an understanding that the play on words wasnt that funny.

Joke: Did you hear about the drug dealer's ghost? He was arrested for possession
Model Prediction: 4, **Score Based on Tier System:** 4, **Human Rating:** 3
This example illustrates a similar takeaway to the first in regard to people giving ratings at the extremes, 0 or 4. People appear to be less likely to give a score of 4 to any joke. The humor scores appear to correlate with what our testers reported, but none of the jokes were given a human rating of 4. However on the jokes that were deemed funny, the votes were heavily skewed towards 3.

Joke: I wanted to become a doctor, but i just didn't have the patients.
Model Prediction: 4, **Score Based on Tier System:** 1, **Human Rating:** 1
This example further illustrates the correlation between our tier system and raw human ratings, but also shows a short coming of our approach. There are so many reasons why our model's prediction could be wrong, ranging from a naive output based on a frequency of words that appear often in funny jokes to it's potential inability to recognize cliché or overplayed jokes. In any case, this example demonstrates the nuance in humor that goes beyond joke structure that humans are able to understand.

4.2 Joke Generation

4.2.1 Data

For joke generation, we used the Short Jokes dataset by Abhinav Moudgil published on Kaggle [3]. The dataset contains 231,657 jokes that range between 10 - 200 characters in length. The dataset claims that almost all jokes it contains are relatively clean, yet manual inspection of the dataset showed there were a significant number of jokes involving racist commentary, sexual innuendo, vulgarity, etc.

4.2.2 Experimental Details

For the custom word-level joke generation network, we used GloVe embeddings of size of 100, LSTM hidden unit size of 200 and attention mechanism size of 128. We used Adam optimization and categorical crossentropy loss with a batch size of 32. We trained this network from scratch for 36 hours (roughly 43 epochs).

For fine-tuning GPT-2, we compiled all the jokes in the shortjokes dataset into a single text file, line delimited, to comply with GPT-2's input representation. GPT-2 uses Byte Pair Encoding (BPE), which essentially creates word-level representations of common symbols and character-level representations of uncommon symbols, interpolating between them as necessary[5]. Default GPT-2 samples 1023 tokens of varying levels of representation to predict the next word, which coupled with the transformer architecture enables GPT-2 to excel at learning long-range dependencies[5]. As such, default GPT-2 expects to train on a large contiguous body of text, leading us to compile the jokes into a single file. We trained GPT-2 on the 117M parameter pretrained weights with a batch size of 1 for 36 hours.

For fine-tuning GPT-2 with the humor loss term, we ran into implementation issues since default GPT-2 generates multiple jokes at once in a "paragraph," but the humor scoring network expects a single joke per input sequence. We eventually figured out we could write each joke in the shortjokes dataset to it's own text file, and reduce the number of tokens to consider when predicting from 1023 to 18. This prevents GPT-2 from training across multiple jokes at once and enables us to feed single jokes or a subsequence of a joke to the humor scoring model during training. We used a batch size of 1 and at each step scored the predicted sequence and label sequence with our humor scoring model, incorporating the scores with the existing categorical cross-entropy loss following Equation 1. Since we had no strong intuition for how to balance cross-entropy loss with humor loss, we simply set α and β to 1. We started training on the 117M parameter pretrained weights for 12 hours.

For fine-tuning textgenrnn, the pretrained model used a character embedding size of 100 and LSTM sizes of 128 and 256[7]. We adjusted the max character input sequence length of 30 and used a batch size of 1024. We trained this for 40 epochs which took roughly 24 hours as well.

4.2.3 Results and Analysis

Custom Word-Level Network with Topic Word Attention

Training our custom word-level joke generation model for 43 epochs from scratch resulted in poor text generation performance:

Example target joke: telling my daughter garlic is good for you. good immune system and keeps pests away. ticks, mosquitos, vampires ... men .

Example topic words input: [daughter, garlic, immune, system, pests]

Generated text: my are the the to to to ? system system system system system

Clearly, the model has yet to learn how to construct coherent English sentences, and might just be starting to learn how to attend to the topic words. This is likely due to the fact that the model has yet to converge after just 43 epochs. Figure 6 shows the model's training loss during our experiment.

We tried experimenting with a CudnnLSTM layer instead of our modified LSTM layer with Attention to speed up training, but this only decreased time per epoch from 50 minutes to 35 minutes. We judged that we lacked the compute resources necessary to train an effective joke generation model from scratch in a reasonable amount of time, which prompted us to explore fine-tuning pretrained networks.

Fine-Tuned GPT-2

Fine-tuning GPT-2 resulted in a model that can generate a paragraph of jokes-like sequences, meaning a list of jokes delimited by " (newline). The loss reached 1.5 around 24 hours and hovered around 1.0 for the next 12 hours. Here's a part of a generated sample sequence from the fine-tuned model:

In this sequence, we see that the network was able to learn common joke structures. Additionally, it recognizes there are no syntactic or semantic dependences between subsequences (individual jokes) separated by newlines. However, the content of the jokes is generally nonsensical, which means the model has yet to represent jokes at higher semantic levels beyond syntactic structure. This can be attributed to the high topic variance of jokes in the dataset. The network may see a joke about a particular topic only once and unable to learn why certain words in that joke go together.

The above sequence was filtered to exclude any excessively offensive jokes. However, we noticed that some jokes would include racial, vulgar and/or morbid terms in a nonsensical manner as well. Not only is this evidence that the dataset is not completely clean, but the frequency of certain terms in this category suggests there are a number of cheap jokes that make fun of the same demographic or use similar punchlines. We did not perform any additional statistical analysis to verify these theories.

Fine-Tuned GPT-2 with Humor Loss

Unfortunately, the network failed to converge during training (see Figure 7). Note that the gap between steps 20k and 40k is a result of the Colaboratory runtime resetting, and while we were checkpointing model weights to Google Drive, the train log was lost for that period.

We attribute this lack of convergence to the following possible factors:

- The incorporation of the humor loss term is the most likely factor. GANs are notoriously difficult to train and it is often observed that gradient descent optimization does not always lead to convergence[2]. In our case, during early stages of training, the model is still learning how to write a sensible joke. These sequences are bad because the language model is outputting noisy sequences, which entails the humor scoring network rating these sequences as 0 (not funny). Thus, the model is doubly punished by the humor loss function when it isn't necessarily fair to judge the humor quality of sequences where the model still needs to learn how to construct a joke in the first place. One method to address this could be gradually adjusting the value of β throughout the training process, so that the humor loss is weighted significantly less at the early stages of training.
- Reducing the length parameter from 1023 to 18 reduces the number of tokens used to predict the next word by 50X. Therefore, the model has less information to condition on and would have a harder time learning longer-range dependencies. However, while we believe this may slow down convergence and reduce the quality of longer sequences, we're not convinced this would introduce significant instability seen during the experiment.

Fine-Tuned Char-Level Model In brief, intermediate results from fine-tuning textgenrnn were not promising. The model had yet to approach convergence after 24 hours of training, and generated samples were much less coherent than samples from GPT-2. Also, the character subsequences used as input made it unclear how we would incorporate humor loss. While it may be useful to compare a character-level model against a word-level model, GPT-2 uses BPE for input representation and we felt a comparision between the two would be less interesting. Therefore, we decided to invest more time in GPT-2 experiments instead of exploring textgenrnn more.

5 Conclusion

We have proposed a novel method of objectively and automatically evaluating the humor of generated jokes. Additionally, directly incorporating this into a humor loss term has the potential to improve the humor quality of modern joke generation architectures.

We find that a shallow model with only one convolutional layer with 128 to 256 filters is able to achieve around 89% accuracy on unseen data and make predictions on 10,064 examples in 1s. Furthermore, we learned that a 5-tier rating system is more useful in this task than a 3-tier system, achieving 20% higher accuracy, which is likely due to the nuance of human humor and how difficult it is to summarize it numerically.

Given time and compute resource constraints, fine-tuning GPT-2 led to the best results for joke generation. In future work, we would like to run extensive experiments with hyperparameter tuning, regularization and alternative training strategies on the GPT-2 with humor loss architecture to achieve convergence and measure how incorporating a humor loss term affects modern text generation networks.

Additional Information

CS 224 Mentor: Annie Hu, anniehu@stanford.edu

References

[1] Chippada, B. & Saha, S. (2018) Knowledge Amalgam: Generating Jokes and Quotes Together. *Microsoft RD*.

[2] Mescheder, L., Geiger, A. & Nowozin, S. (2018) Which Training Methods for GANs do actually Converge? *International Conference on Machine Learning*

[3] Moudgil, A. (2017) Short Jokes dataset. *Kaggle Inc*. https://www.kaggle.com/abhinavmoudgil95/short-jokes.

[4] Radford, A., Narasimhan, K., Salimans, T. & Sutskeyver, I (2018) Improving Language Understanding by Generative Pre-Training. *OpenAI*.

[5] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I (2019) Language Models are Unsupervised Multitask Learners. *OpenAI*.

[6] Ren, H. & Yang, Q. (2017) Neural Joke Generation. *International Conference on Machine Learning*.

[7] Woolf, M. (updated 2018) textgenrnn. *GitHub*. https://github.com/minimaxir/textgenrnn

6 Appendix

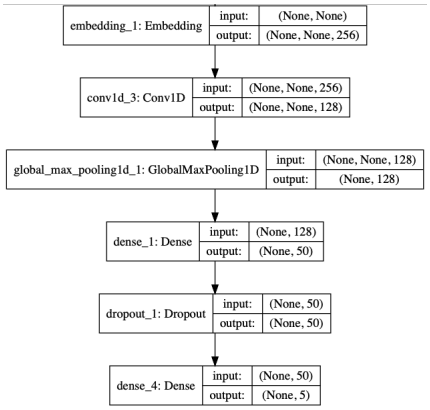

Figure 1: Final Humor Metric Network Architecture

$$
\begin{aligned}
\text{scores} &= v^\top tanh(W_1 h_t + W_2 E_{\text{topic words}}) \\
\alpha &= softmax(\text{scores}) \\
\mathbf{a} &= \sum_{i=1}^{|T|} \alpha_i e_i \\
c &= [\,\mathbf{a}\,; h_t]
\end{aligned}
\tag{3}
$$

Where h_t is the current decoder hidden state, $E_{\text{topic words}}$ is the topic word embedding matrix, e_i is the ith topic word embedding, and c is the context vector used to compute the output at the current timestep.

Model	Architecture
1	Conv Layer 1: 512 filters, Conv Layer 2: 256 filters
2	Conv Layer 1: 256 filters, Conv Layer 2: 128 filters
3	Conv Layer 1: 512 filters
4	Conv Layer 1: 256 filters
5	Conv Layer 1: 128 filters
6	Conv Layer 1: 64 filters
7	Conv Layer 1: 32 filters

Table 1: Tested Humor Network Architectures

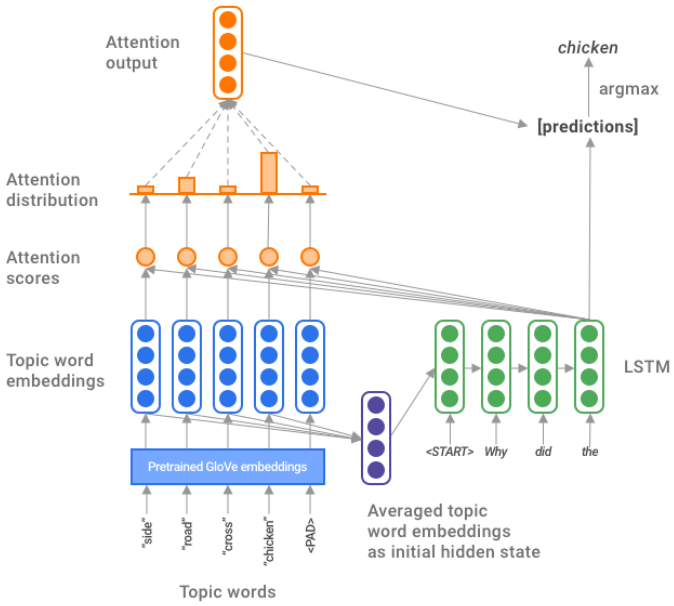

Figure 2: Custom Word-Level Model with Topic Word Attention

Model	Train Accuracy (3-Tier)	Test Accuracy (3-Tier)	Train Accuracy (5-tier)	Test Accuracy (5-Tier)
1	0.675	0.724	0.794	0.791
2	0.694	0.63	0.862	0.843
3	0.723	0.638	0.904	0.89
4	0.727	0.637	0.914	0.894
5	0.732	0.736	0.915	0.891
6	0.733	0.674	0.9	0.879
7	0.728	0.759	0.875	0.853

Table 2: Model Accuracies After 30 Epochs

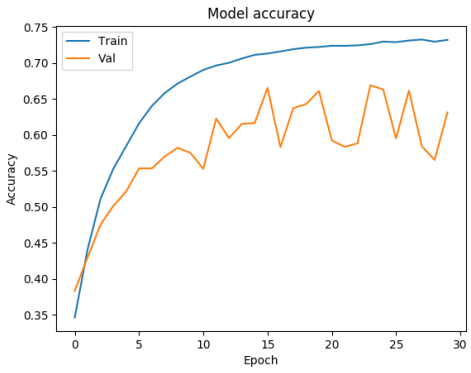

Figure 3: 3-Tier Train Accuracy

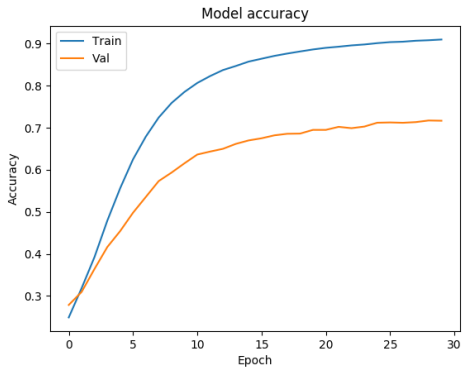

Figure 4: 5-Tier Train Accuracy

Figure 5: Model 5 Training Accuracies

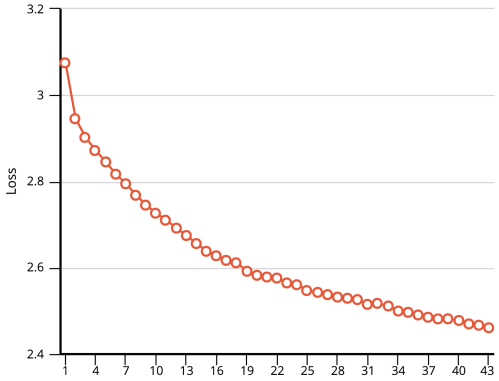

Figure 6: Loss for Custom Word-Level Network with Topic Word Attention for 43 epochs

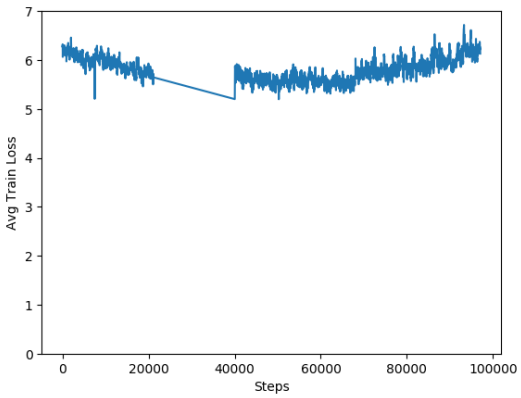

Figure 7: Loss for Fine-Tuned GPT-2 with Custom Humor Loss Function

Figure 8: Part of a Generated Sample Sequence from Fine-Tuned GPT-2

I hear Donald Trump has been planning to build that wall with his prison pen. But he's just not into it.

As I was going to tell you about the raccoons in the airing cup, well, upon hearing, it was just a flip.

I hate being a depressed atheist. It makes me feel that pizzas feel the end of the world

I'm really upset that people think I have cancer. But it's not my blindness at first, until my bar comes flashing me back.

A wife is out to dinner. her husband asks her "What are you doing?" to which silence you replied " quotes." She gets to sit there and says "go home".

What did the ISIS fax machine salesman say to the persistent clock maker? Take me to your breeder.

I had an argument with a bulimic cannibal recently They caught me in the mouth!

[NSFW] What does a gay guy and a freezer have in common? (NSFW) They're both "in space"

What do a gun, an apple, and pink bananas have in common? The great ones don't come with boxers.