JokeR: A Recurrent Joker

Tom Hartvigsen [1] **Sanket Gujar** [1] **Thanh Tran** [1]

Abstract

Generating jokes is a challenging and understudied task of Natural Language Processing. A computer that intends and succeeds to generate jokes could be deemed artificially intelligent. We present a couple of novel approaches to joke generation, such as using SeqGAN and a language model. We implement a variety of word-level models to tackle parts of the joke-generation problem, namely text generation and joke classification. Ideally, merging these steps will allow for a model to write joke candidates, that are then pruned by a well-trained classifier. We train these models from a corpus of 231,657 user-written jokes scraped from reddit.com[1].

1. Introduction.

Joke generation is a challenging Natural Language Processing task as success implies abstract reasoning and an understanding of societal context. Additionally, jokes can appear in many formats, such as Knock-Knock, Question-Answer, or Story-Based. Generating humorous jokes requires a model to accurately predict how funny an observer will find its text, a task at which many humans often fail.

The contexts and syntax styles present in such different joke-types present issues when modeling textual sequences. Of the relatively few attempts to generate jokes, most bypass this challenge by considering only one type of joke. This type of constraint on the problem dramatically reduces the complexity, but allows for much more realistic results. In our work, we do not constrain the joke type, but instead filter jokes by lengths to avoid other problems.

Text generation has recently received much attention within Natural Language Processing and Deep Learning. However, humor is subjective and quantifying success of generated text is challenging. Typically, text generation is either empirically qualified or the success of the language model's predictions of known sentences is maximized. A metric such as the BLEU score does not necessarily directly apply to our setting as one wrong word at the end of a joke could devalue the entire sequence in some cases. It is difficult to compute what makes a joke a joke in order to minimize a loss function.

For joke generation, writing readable text is only half the battle. We equate a machine that writes un-funny "jokes" that retain some semblance of joke-structure to a three or four year old who, upon hearing others tell jokes and noting the reactions of their audience, begins to mimic the syntax styles of jokes despite their lack of truly humorous content (e.g. Why did the apple eat a spoon? Yellow!).

We present jokes generated through an Encoder-Decoder Recurrent Neural Network that learns a language model using both standard one-hot encoded word representations and word2vec (Mikolov et al., 2013) as inputs. Additionally, as an attempt to force humor into the model, we implement a human-in-the-loop topic-based joke generation by adding a selected topic to the model's initial state. We also implement a Sequence GAN to combine the generation and discrimination tasks. Finally, we train a fully connected network to classify sentences as either jokes or not-jokes.

We use a corpus of 231,657 jokes written by Reddit users and show that a language model and GAN can mimic jokes to some degree and show that joke generation is a challenging task and the only humor found in our generated jokes is imposed by the reader.

2. Related Works

(Petrović & Matthews, 2013) presents a model that uses large amount of unannotated data to generate *I like my X like I like my Y, Z* jokes, where X, Y and Z are variables to be filled in. This was the first fully unsupervised humor generation system. They assumed X and Y are nouns and Z is an adjective. They found out the co-occurrence between x and z and also between y and z in some large corpus, measured how similar are x and y and obtained a joint probability for (x,y,z) to occur together.

We took inspiration from Seq2Seq[2] which was a general end-to-end approach to sequence learning that makes minimal assumption on the sequence structure. Their method used a multi-layered LSTM to map the input sequence to a vector of fixed dimensionality, then another deep LSTM to decode the target sequence from the vector.

[1] https://github.com/amoudgl/short-jokes-dataset

(Ren & Yang) presented a topic based joke generator, where the model can generate a short joke relevant to the topic that the user specifies. They used an encoder for representing user-provided topic information and an RNN decoder for joke generation. They trained the model on short jokes corpus of Conan O'Brien. They used POS Tagger to extract the topic from each sentence. The evaluation was done by five English speakers and they found their model to outperform a probabilistic model[1].

Humor generation is hard but is humor classification and there have been a very few approaches for this too. Dario and Pascale(Bertero & Fung, 2016) used an LSTM for predicting humor in dialogues. Their model consisted of LSTM with utterance encoding from a CNN to model the sequential context of the dialog. They used popular TV-sitcom dialogues to train their model by supervised classification to detect when the punchline occur. They achieved the F-score of 62.9% while a Conditional Random Field(CRF) model achieved 58.1% on the same data.

Most recently, SeqGAN(Yu et al., 2017) was introduced which is a sequence generation framework. SeqGAN bypasses the generator differentiation problem by directly performing gradient policy update, while the data generator is considered as a stochastic policy. The RL reward comes from the discriminator judgment on a complete sequence and is passed back to the intermediate state-action steps using Monte-Carlo search. Although SeqGAN is only tested on Chinese poems and random normal distribution.

3. Dataset

We use a set of 231,657 jokes scraped from Reddit's *jokes* and *cleanjokes* pages (dat). Each joke has at least 10 characters and a maximum of 200 characters. For simplicity, we only use jokes that have fewer than 20 characters.

4. Methods

4.1. Word Embedding

Word vector are a vector space representation that captures semantic and syntactic regularities. The most commonly used algorithms are Global vector for word representation (GloVe) (Pennington et al., 2014) and Word2vec (Rong, 2014). GloVe is proved to combine advantages from both count-based methods (LSA, Hellinger-PCA) and direct prediction methods (Skip-gram and CBOW) and gives good performance on word similarity tasks.

We trained the GloVe model on our joke corpus for the embedding to particularly capture the word similarities in our corpus. We kept the size of embedding to be fixed of 300 as it is mostly referred to capture most amount of information. We note that the word embeddings do not seem to make much sense and words we assume to share contexts do not appear next to one another. This may be due to the fact that each joke is independent from other jokes, so word contexts may not be meaningful in this setting.

4.2. Language Model

We implement an Encoder-Decoder Recurrent Neural Network that inputs word sequences, passes them through an LSTM layer, then attempts to predict the next word. Thus, the model attempts to learn a low-dimensional and sequential representation of jokes. We attempt two versions of this model: first using one-hot encoded representations of words, and second using word embeddings trained on a dataset of jokes. Once this model has been trained to adequately predict future words given a current word, we can sample from the predicted distribution at each time step in order to generate novel sequences.

4.3. Joke Classifier

Preparing input data: In this task, we use neural network to build a joke/non-joke classifier. We consider all jokes in our dataset mentioned in section 3 as positive samples. However, we are missing negative instances in this case.

In order to obtain negative instances, according to (Ren & Yang), the news data source can be considered as a good non-joke data for our classifier since the news and our jokes datasets both are about daily affairs. Therefore, we started crawling news headers as non-joke dataset.

We used Reuters as our news data source. [2] stored all days' news headers. We crawled news headers of 1000 days () from current date time to the pass. This results in 183,744 news header.

Before fitting this dataset to train our model. We want to make sure that the news dataset and the jokes dataset both share a similar portion of their vocabulary. This is because if jokes and news datasets share no common vocab, the model simply just learns the existence of a non-common word to classify the input text. For example, a news is about "Syria", while there is no joke mentioned "Syria", the model simply learns if the input text contains "Syria" then classify as non-joke. Unfortunately, even though we crawled 183k news header, the percentage of common vocabulary between the news and the joke datasets is very small. Therefore, we conclude to not use news dataset as non-joke instances.

There are two other approaches to generate non-joke datasets: (i) shuffle all words in a joke; (ii) split a joke into smaller sentences, and each small sentence alone will

[2] https://www.reuters.com/resources/archive/us/

be a non-joke. We ignore the first approach because it can train a model that just learns grammar correction. For generating non-joke data using the second approach, we split each joke into short sentences by "?", ",", "." separators. We ignore the jokes not containing those separators. Then we randomly sample a same amount of jokes.

Building classifier: We propose two classifiers, one with attention and one without attention. We used pre-trained word2vec embeddings (GoogleNews-vectors-negative300.bin) [3]. Then we use BiLSTM to encode the input text. With attention score model,a score function is simply built by constructing another fully connected layer that input concatenated output vectors of BiLSTM and producing a score. Then the context vector is computed by sum of hidden vectors multiplied by their attention scores.

4.4. Topic-Based Language Model

Similar to (Ren & Yang) we attempt a topic-based joke generator by initializing the hidden state of the language model as a word embedding for a particular topic. In the training phase, we extract the nouns from each joke and consider them to be topics. To create a fixed-size vector, we average the topic vectors if there is more than one. This assumes that the average of two topic vectors will contain information from each topic and that topics of jokes have been successfully mapped into a similar region of the embedding space. For example, we assume that the average of "donald trump" and "barack obama" will be a word such as "president" as opposed to a different and random vector such as "banana". The intuition behind this approach is that initializing the first output of the language model as a combination of a seed word and a topic vector will influence the model to stick with words surrounding the assigned topic. Additionally, this would allow for some human input as to what might make a funny joke. This allows the model to bypass the necessary understanding of societal context when generating jokes. Similar to the basic Encoder-Decoded Sequence to Sequence model, in the testing phase we can sample from the output of the model at each step and feed that prediction into the next timestep to generate novel sequences.

4.5. Sequence Generative Adversarial Networks

Given the dataset of short jokes, we trained a θ-parametrized generative model G_θ to produce the joke sequence $Y_{1:T} = (y_1, y_2, y_3, ..., y_T)$, here $y_t \in \gamma$, where γ is the vocabulary of the dataset. The problem is interpreted as a reinforcement learning problem, where on every timesteps t, the state s is the current produced state and action a is the next token y_t to be selected. The policy model $G_\theta(y_t|Y_{1:t-1})$ is stochastic, but the state transition is always deterministic after an action has been chosen. We also trained discriminator model D_ϕ to discriminate between joke and plain text, and also to provide guidance to the generator G_θ. So $D_\phi(Y_{1:T})$ is a probability indicating how likely the sequence is a joke or not.

The Discriminator is trained by using positive examples from the joke dataset and negative sample generated from the generative model G_θ. The generator G_θ is updated using policy gradient and MC search on the basis on the expected reward received from the discriminator D_ϕ.

We experimented with a very simple RNN models for the generator and discriminator. The generator was character-level 3-layered RNN with GRU cells of 128 dimensions and the output dimension were the total number of different of characters in the dictionary. The input to the generator was a latent variable sampled from normal distribution. The generator was trained with teacher forcing method with the jokes samples for initial timesteps to guide the generator and later was trained with the help of reward from the discriminator. The discriminator had the same architecture as the generator but with a fully connected network with sigmoid activation at the output to determine the probability of the sequence being a real joke or not.

5. Results

5.1. Language Model

Our first attempt to generate jokes is via sampling sentences from a language model trained on 231,657 jokes. After generating 100 jokes, we hand-picked some jokes that seem reasonable if not funny: **Why did the snowman go to the Doctor? Assault**; **What did the vegan pig say? Pig in the office**; **What do you call a deer in the office? Spiderman milk.**

Each of these examples is a novel sequence, but seems to be merging pieces of existing jokes. We note that most jokes generated in this way are simply sequences of words that are distantly related to one another and definitely contain no humor. For instance, "What did the blind man fit in two arms and his pants in quicksand?" has some structure, but certainly no meaning. However, to a non English-speaker they may seem like reasonable sentences, and to a three-/four-year-old who has just discovered jokes, they may seem as high quality as any joke.

5.2. Topic-Based Language Model

In order to generate topics, we first embed each word into a 300-dimensional space. However, since the word embeddings have little semantic meaning, the *Topic-based language model* did not stick to any particular topics and did

[3] https://code.google.com/archive/p/word2vec/

Table 1. RNN Classifier result with and without pre-trained word embedding

Model	Accuracy
pre-trained RNN without attention	95.8%
RNN without attention	89.7%

Table 2. pre-trained RNN Classifier result with and without attention

Model	Accuracy
pre-trained RNN without attention	95.8%
pre-trained RNN with attention	94.3%

not generate meaningful sequences to any degree, with absolutely no sentence structure whatsoever and many repeated words, even with lots of training time. This could be due to meaningless word embeddings in the context of a language model or the possibly erroneous assumption that the average word embedding between nouns is an adequate summarization of multiple topics.

5.3. Joke Classification

In this part, we report the result of RNN classifiers in following cases: (i) Case 1: RNN without attention + pre-trained word embedding versus RNN without attention; (ii) Case 2: RNN without attention + pre-trained word embedding versus RNN with attention + pre-trained word embedding; (iii) Case 3: Our best model versus uni-gram Naive Bayes classifier.

From table 1, we observe that RNN classifier without attention mechanism works well with pre-trained word embeddings, significantly improve the accuracy result by 6.1%. We see in Table 2 that attention mechanism degrades accuracy of our pre-trained RNN classifier by 1.5%. Therefore, we use pre-trained RNN classifier without attention as our best model. Then, we compare our best model with unigram features + Naive Bayes classifier as a baseline. Table 3 shows our results. Our proposed model outperformed the baseline, significantly improves the accuracy by 13.2%.

5.4. SeqGAN

The jokes generated captured the distribution of jokes to some extent but still they lacked intuition to generate funny jokes. The model was generating jokes with bar occurring very frequently. It is possible as the discriminator will be rewarding the generator for generating jokes with bar, and the generator might have been stuck in the local minimum. We assume that using classifier (Section 4.3) that can classify jokes properly will improve the results by giving appropriate rewards to guide the generator.

The generator also generated jokes with inappropriate words like sex, $condoms$ etc. which may not be appropriate to common audience as well as it also generated some racist jokes, which it learned from the dataset.

Table 3. Our Best RNN classifier with unigram Naive Bayes classifier

Model	Accuracy
pre-trained RNN without attention	95.8%
Naive Bayes classifier	82.6%

Some sampled jokes from the generator:

- I was in the bar, but I wasn't the bartender.
- What do you call a bar, a bar because it was a bar
- What do you call a girlfriend? a child to the bar.
- I was in the bar and the bar says I am not a bar.

6. Evaluation

We told 5 random jokes we generated and from the dataset to 5 people with different native language as shown in the table and asked them to evaluate the jokes on a scale of 1-5. However, as the jokes generated were not very initiative we told the audience which joke are machine generated. We averaged the score the rating given to the jokes for each person.

Person (lang.)	Org. Jokes	Machine Generated Jokes
P1 (Tamil)	4.0	3.0
P2 (English)	2.6	2.0
P3 (English)	2.4	1.2
P4 (Hindi)	2.8	2.6
P5 (Hindi)	2.6	2.8
Average	2.88	2.51

The jokes generated are close in rating with the original jokes, but we assume it won't be the case if the audience weren't told the joke is machine generated or not.

7. Conclusions

Jokes generated through our variety of models were not realistic to any degree. The jokes from the encoder-decoder language model contained no humor while the jokes from SeqGAN revolved around only a few certain topics, which could be improved with a better discriminator. Many of the jokes contained racist or sexual words, which may not be appropriate to many audiences, so better filtering of the dataset is essential. The generated jokes got overall similar rating to the original jokes, but the participants were aware which jokes were machine generated. We also built a joke classifier using BiLSTM and achieved an accuracy of 95.8%. The generated jokes can be re-verified by this classifier to produce better jokes.

References

Short jokes. https://www.kaggle.com/abhinavmoudgil95/short-jokes.

Bertero, Dario and Fung, Pascale. A long short-term memory framework for predicting humor in dialogues. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 130–135, 2016.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Petrović, Saša and Matthews, David. Unsupervised joke generation from big data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pp. 228–232, 2013.

Ren, He and Yang, Quan. Neural joke generation.

Rong, Xin. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Yu, Lantao, Zhang, Weinan, Wang, Jun, and Yu, Yong. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pp. 2852–2858, 2017.