Neural Joke Generation

He Ren, Quan Yang
Department of Electrical Engineering, Stanford University
{heren, quanyang}@stanford.edu

Abstract

Humor generation is a very hard problem in the area of computational humor. In this paper, we present a joke generation model based on neural networks. The model can generate a short joke relevant to the topic that the user specifies. Inspired by the architecture of neural machine translation and neural image captioning, we use an encoder for representing user-provided topic information and an RNN decoder for joke generation. We trained the model by short jokes of Conan O'Brien with the help of POS Tagger. We evaluate the performance of our model by human ratings from five English speakers. In terms of the average score, our model outperforms a probabilistic model that puts words into slots in a fixed-structure sentence.

1 Introduction

Humor research has been an important part of literature, linguistic and cognitive science for a few decades, and theorists have believed that incongruity contributes to sensations of humor. Many researchers have been studying the logic, mathematical foundations and computational models of humor [1]. Concretely, probabilistic model of sentence comprehension can help explain essential features of the complex phenomenon of linguistic humor [2], and the paper concludes that both ambiguity and distinctiveness are significant predictors of humor.

Computational humor is a relatively new field of study and branch of artificial intelligence which uses computers to model humor. Joke generation is a very hard problem in this field. Researchers have proposed mathematical models to generate fixed-structure joke [3] with the help of big data. However, the model selects words and fill them into several fixed slots within simple sentence structure rather than generating a complete joke.

Recurrent Neural Networks (RNNs), and specifically, a variant with Long Short Term Memory (LSTM), are having successful applications in a wide range of machine learning problems that involve sequential data prediction, from text generation [4] to neural machine translation [5] demonstrated the power of RNNs trained with the new Hessian-Free optimizer (HF) by applying them to character-level language modeling tasks. After training using a large corpus, the language model can generate readable texts. Following the work of [4], it is shown in experiments that Gated Recurrent Units (GRUs) and LSTMs both significantly outperform RNN [6].

Motivated by these recent developments, in this paper we use a corpus of thousands of short jokes written by Conan O'Brien, and we want our model to be able to generate a relevant and comprehensible joke given a few topic words. To extract the topic words from the training data, we use the part-of-speech (POS) tagger [7] to collect the proper nouns in the jokes. We use Global Vectors (GloVe) [8] to represent input and topic words, and LSTM RNN with attention-mechanism as our neural network decoder architecture. As there is no reliable

automatic evaluation on joke generation task, we depend on human evaluation and compare the results with the probabilistic model proposed by [3].

2 Related Work

2.1 RNN Encoder-Decoder

Joke generation is similar to text generation but a lot harder because there must be certain incongruity in the generated content that makes people laugh. Character-level RNNs can generate readable words [4], but the content is not guaranteed to be meaningful. Neural machine translation [5] generates meaningful content with the encoder–decoders architecture [9], [10]. Models proposed recently for neural machine translation usually encode a source sentence into a fixed-length vector from which a decoder generates a translation. LSTMs have been used as the decoder to generate image caption [11]. The neural image captioning model is also an encoder-decoder structure, but different from neural machine translation in that the encoder is a convolutional neural network (CNN). The model uses the last hidden layer of CNN as an input to the RNN decoder that generates sentences. Similarly, GRU has been used as decoder to generate factoid questions [12]:

$$g_n^r = \sigma(W_r E_{out} w_{n-1} + C_r c(F, h_{n-1}) + U_r h_{n-1})$$

$$g_n^u = \sigma(W_u E_{out} w_{n-1} + C_u c(F, h_{n-1}) + U_u h_{n-1})$$

$$\tilde{h} = \tanh\,(W E_{out} w_{n-1} + C c(F, h_{n-1}) + U(g_n^r \circ h_{n-1}))$$

$$h_n = g_n^u \circ h_{n-1} + (1 - g_n^u) \circ \tilde{h}$$

The work directly uses fact embedding as the encoder's output rather than traditional sequential encoding.

2.2 Attention Mechanism

The attention mechanism was first introduced to sequence-to-sequence [5] to release the burden of summarizing the entire source into a fixed-length vector as context.

The architecture of question generation [12] includes an attention mechanism on the encoder representation to generate the associated question Q to that fact F. Other researchers use copying mechanism [13] to ensure that the generated output includes many words from the input. The copying mechanism increases the probability that certain words appear exactly in the output by using separate parameters to compute probabilities for different sections of vocabulary.

3 Approach

3.1 Dataset

Our training data consists of data from two different sources. The first source is the 7699 jokes written by Conan O'Brien (downloaded from `github.com/brendansudol`), all of which are short jokes (many are one-sentence jokes) on different topics such as current affairs, politics and sports. These jokes contain 258443 words in total and a vocabulary size of 13773. Another source is the online news data from the Mashable website. Since the joke data are mainly about current affairs, it would be helpful to incorporate news data to improve the training of language model. Our training data is the mixture of both jokes and news data. Although the news is not funny, the weighted-pick strategy above the RNN output layer brings randomness to the generated text, and thus has a certain probability to produce incongruity in the text, potentially making it funny.

3.2 Encoder

We first used Part-of-Speech Tagger (POS Tagger) to extract proper nouns from each training samples. Since our goal is to generate jokes based on topic keywords given by user, we assume that proper nouns (e.g., individual entities like name of person, location, organization, etc.) are keywords that can represent the topic of the joke. For example, the sentence "In Australia, a couple is getting married inside an IKEA store, which means their marriage will likely fall apart in two years." the POS Tagger collects "Australia" and "IKEA".

Word vector are vector space representations of words that can be used for capturing fine-grained semantic and syntactic regularities. Typical algorithms are Word2Vec and Global Vectors for word representation (GloVe). We use the pre-trained GloVe vectors from Stanford NLP group as embedding because it's been proved to combine advantages from both count-based methods like LSA, Hellinger-PCA, and direct prediction methods like Skip-gram and CBOW, and has very good performance on word similarity / analogy tasks.

We encodes the extracted proper nouns, and then average the bag of words as $Enc(joke) = \frac{1}{k}(e_{w_1} + e_{w_2} + \cdots + e_{w_k})$. The embedding matrix could be another variable to learn, however, since our training corpus is not large enough, it is better to use the fixed GloVe embeddings. We tried GloVe embedding of length 50, 100 and 300, and 300 gives the best result, as it stores the most amount of information. To use GloVe as the initial hidden state of the decoder, the decoder hidden states should be of the same dimension. To enable larger-size RNN, we implemented another layer $H = \sigma(Wx + b), H \in R^{D_H}, x \in R^{D_{Enc}}$ at the front of encoder for dimension transformation, and now the RNN hidden size is independent from the size of word embeddings.

Since using the average of proper nouns may cause a loss of information, we also try using concatenation as $Enc(joke) = [e_{w_1}, e_{w_2}, \ldots, e_{w_k}]$. However, that leads to variable length of output, and concatenation means modifying the order of proper nouns might represent different meanings in this case, which is not reasonable. Therefore we move forward with average.

The output vector from the encoder serves as the initial hidden state of the decoder. We also use the topic word embedding (proper nouns) for attention-mechanism [14], which is introduced in detail in section 3.2. The need for attention is motivated by the intuition that the model would attend to different kinds of topic word at different time steps in the joke-generation process. Vectors of individual proper nouns are put in a pool for the attention module to select from, and the RNN pays different attention to different proper nouns in each time step.

3.3 Decoder LSTM

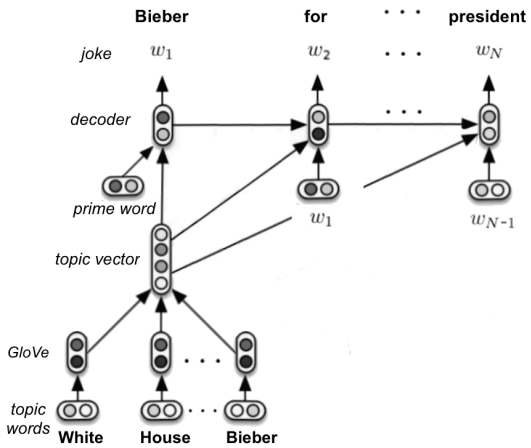

Figure 1: The computational graph of the joke-generation model

For the decoder in Figure 1, we use LSTM/GRU recurrent neural network with an attention-mechanism on the encoder representation to generate the joke containing the topic words (modified from `github.com/hunkim/word-rnn-tensorflow`). The hidden state of the decoder RNN is computed at each time step as:

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1})$$
$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1})$$
$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1})$$
$$\tilde{c}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1})$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \tanh(c_t)$$

Unlike [4], we build word-level language model instead of character-level language model. Concretely, we encode topic words and input words into vectors using GloVe embedding, and feed the output word back into the RNN one at a time with the step function. We will then observe a sequence of output vectors, which we interpret as the confidence the RNN currently assigns to each word coming next in the sequence. Compared with character-level RNN, the word-level RNN achieves better sentence coherence in the generated text.

Lastly, the function is computed using an attention-mechanism [14]:

$$u_i^t = v^T \tanh(W_1{}' h_i + W_2{}' d_t)$$
$$a_i^t = \text{softmax}(u_i^t)$$
$$d_t{}' = \sum_{i=1}^{T_A} a_i^t h_i$$

in which, $W_1{}'$ and $W_2{}'$ are learnable parameters. The vector u^t has length T_A and its *i*-th item contains a score of how much attention should be put on the *i*-th hidden encoder state h_i. These scores are normalized by softmax to create the attention mask a^t over encoder hidden states. Lastly, we concatenate $d_t{}'$ with hidden state h_t, which becomes the new hidden state from which we make predictions, and which is fed to the next time step in our recurrent model.

3.4 Generating Jokes

3.4.1 Options

Before starting the generation, there are a few options for the user to specify or leave as default. First of all, the user can specify the topic about the joke. The topic is described by a list of proper nouns, such as "Donald Trump", "Obama China" etc. The proper nouns are then represented by GloVe embedding, transformed to use as the initial hidden state and the attention of the decoder RNN. If the topic is left as default, the decoder RNN would have all-zero vector as initial hidden state and no attention. In this case, the topic is going completely freestyle. Moreover, the user can also specify the prime words--what words the joke starts with--by providing a list of words like "I don't know why but", "Donald says yesterday that" etc. These words are going to be the input words one word at a time, from the first time step. If the prime words are not specified, a random word from the vocabulary will be chosen as the starting word.

3.4.2 Search Strategy

Based on the model trained on our joke corpus, the RNN can generate new text one word at a time. At each time step, the RNN outputs the probabilities that each word in the vocabulary appear next. We don't want to consistently choose the word with the highest probability, because this will cause the model to make more likely, yet also more boring and conservative

predictions. Weighted-pick search strategies can cause the model to take more chances and increase diversity of results at a cost of more mistakes. The model picks random word from the vocabulary according to the weights, which are the probabilities coming from the last softmax layer.

3.4.3 Generation

The user might want to specify the exact number of words for the RNN to output. If this number is omitted, the joke generation can automatically stop when an end-of-sentence token is produced. The end-of-sentence tokens are manually added in the training data, at the end of each separate joke.

Upon generation, the user-defined prime words are encoded into embeddings, and fed into RNN one word at a time. The output vector indicates the confidence the RNN currently assigns to each word coming next, and such confidence is then used as weights in the weighted-pick search strategy. The picked word becomes the next word in the generated text, and it is iteratively fed into RNN as the next input word if the user-given prime words run out.

4 Experiment

4.1 Training

We tried to improve performance in following steps. Firstly, we trained the decoder-only model (a vanilla RNN) and tuned the hyperparameter (hidden state length, number of layers, etc) for the best performance in text generation in terms of syntax. Then we included the encoder to initialize the hidden state of the decoder, and found significant relations between the input topic words and the output. After implementing the attention-mechanism, we were able to generate jokes that has high probability to cover the topic words. If not, the joke would include those similar to the topic words (e.g. “president” appears when “Obama” doesn’t). At the same time, we observed a decrease in sentence fluency. Therefore, during the training of our decoder as in Figure 2, we use pre-trained RNN parameters for initialization. This approach alleviates the sentence fluency problem.

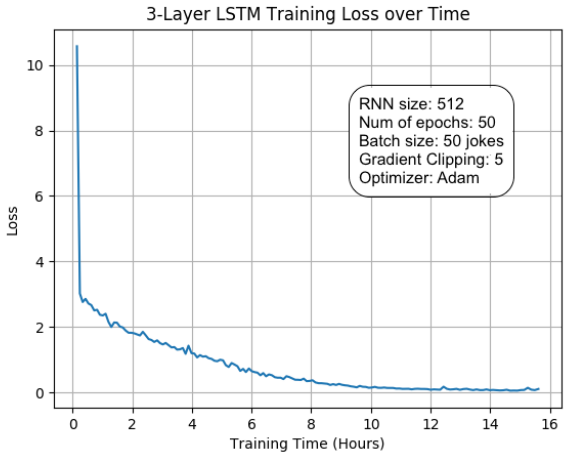

Figure 2. Training loss over time

Model	Human Evaluation Score	% of Good Jokes (Score = 2, 3)
Human Jokes	2.12	35.3%
Probabilistic Model in [3] (baseline)	1.44	16.3%
Neural Joke Generator	1.59	12.1%
Score: 0 - nonsense; 1 - not funny; 2 - somewhat funny; 3 - funny		

Table 1. Comparison of different models of joke generation based on human evaluation

Topic Words	Generated Joke (Selected)
Los Angeles Trump	According to a new study , the governor of film welcome the leading actor of Los Angeles area , Donald Trump .
Apple Playboy	Apple is teaming up with Playboy Magazine in the self driving office .
Kardashian President	Yesterday to a new attractiveness that allows Bill Kardashian 's wife to agree with the U .S . Presidents . In fairness , she said , " My spa . "
None	One of the top economy in China , Lady Gaga says today that Obama is legal .
None	New research finds that Osama Bin Laden was arrested for President on a Southwest Airlines flight .
None	Google Plus has introduced the remains that lowers the age of coffee .

Table 2. Generated joke samples

4.2 Evaluation

Our goal is to produce high percentage of funny jokes. The level of funniness is, however, a difficult thing to measure. There is no automatic evaluation that we can do on the joke generator, because jokes are highly subjective, creative and diverse. They can be employed in the form of monologue, question-answer, short stories and can cover a variety of topics from food and diet to politics and profession.

The main evaluation we use on our model is from human ratings, and we want to get a percentage of funny ones among all generated jokes. The higher the percentage, the better the model is. Human evaluation on jokes can be biased, because “there are a thousand hamlets in a thousand people’s eyes”. Cultural references and background knowledge also affect human judgement on jokes (e.g. a stormtrooper is a fictional soldier in Star Wars, which might be lost on those who are not familiar with the movie). To get a fair evaluation, we showed our jokes to five English speakers of different cultural background in a random order. The raters were asked to rate each joke on a 4-point Likert-type scale: 0 (nonsense syntactically), 1 (not funny), 2 (somewhat funny), 3 (funny). We compare the results of our model with those of human jokes as well as the models presented by [3].

We use the evaluation of our training data--jokes written by Conan O’Brien--as the performance of human jokes. As shown in Table 1, human jokes set a contrast level for how good the score and the percentage could be with human minds. The probabilistic model by [3]

is the baseline that our model compares to. Our neural joke generator has lower percentage of good jokes than the baseline model, and it is probably because our model generates the whole sentence rather than filling in a few slots of words. Many syntactic errors cause a large portion of the generated jokes to be nonsense or not funny. However, the average score of our neural joke generator is still slightly better than the baseline. It indicates that among the good ("somewhat funny" or "funny") neural-generated jokes, the percentage of "funny" is higher, compared with the baseline.

From Table 2, we can see that the style of the generated jokes is very similar to Conan's style, short and related to current affairs. RNN arbitrarily put together seemingly unrelated concepts, causing incongruity. The generated text could be nonsense, but some of them are funny because of incongruity.

5 Conclusion

We have proposed a new neural network model for mapping arbitrary topics into corresponding natural language jokes. The model combines ideas from recent neural network architectures for machine translation, factoid question generation, as well as neural image captioning (NIC). The produced jokes are evaluated using human evaluation, and are found to outperform a probabilistic model [3].

As for future work, current corpus is not large enough and a majority of topic words only occur a few times. This limits the performance significantly when we choose rare words as topic words. In addition, for this paper we only use jokes from one author for easier learning. For broad joke generation, we'll need to incorporate other sources for greater diversity of styles and topics. In terms of model structure, our encoder is basically a one-step process which looks up and take the averages of embeddings, and we could improve it with a sequence model (another separate RNN probably) that treats proper nouns as sequential inputs. Apart from that, we could also implement the copying mechanism to make the topic words exactly appear in the jokes while not affecting the sentence coherence.

References

[1] Paulos, J. A. (2008). *Mathematics and humor: A study of the logic of humor*. University of Chicago Press.

[2] Kao, J. T., Levy, R., & Goodman, N. D. (2013). The Funny Thing About Incongruity: A Computational Model of Humor in Puns. In *CogSci*.

[3] Petrovic, S., & Matthews, D. (2013, August). Unsupervised joke generation from big data. In *ACL* (2) (pp. 228-232).

[4] Sutskever, I., Martens, J., & Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (pp. 1017-1024).

[5] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[6] Karpathy, A., Johnson, J., & Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

[7] Toutanova, K., Klein, D., Manning, C. D., & Singer, Y. (2003, May). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1* (pp. 173-180). Association for Computational Linguistics.

[8] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In *EMNLP* (Vol. 14, pp. 1532-1543).

[9] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine

translation. *arXiv preprint arXiv:1406.1078*.

[10] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104-3112).

[11] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3156-3164).

[12] Serban, I. V., García-Durán, A., Gulcehre, C., Ahn, S., Chandar, S., Courville, A., & Bengio, Y. (2016). Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. *arXiv preprint arXiv:1603.06807*.

[13] Gu, J., Lu, Z., Li, H., & Li, V. O. (2016). Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*.

[14] Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., & Hinton, G. (2015). Grammar as a foreign language. In *Advances in Neural Information Processing Systems* (pp. 2773-2781).