# ML Joke Bot

CS 487 Project

Spring 2020

C. Baker | M. Dickson | L. Mendoza

# Motivations

Everyone loves laughing. When people laugh, it doesn't just lighten their load mentally, it actually induces physical changes in their body such as reducing stress, easing tension and stimulates many organs[1]. Machine learning software with a sense of humor has even shown promising results in helping children learn language skills [2]. But despite this, humor is generally thought of as a trait unique to humans and other highly intelligent animals [3][4], so we thought it would be interesting to see if we could use a machine learning algorithm to create its own unique jokes. The current leading theory of humor suggests that a sense of humor requires an understanding of social context, which might require mastery of sophisticated functions like self-awareness, empathy, spontaneity, and linguistic subtlety [5]. In this way, creating a jokebot could offer a window into better understanding the rudiments of humor, as well as insights into the difficulties with getting a computer to "think" like a human.

# Definition of Problem

Training an ML algorithm to author its own jokes.

- Our algorithm should learn the structure of jokes.
- Our model should output its own joke in text format when the program is run.
- Evaluate performance by deciding whether we think the jokes being output are funny.

# Challenges

For this problem, our data was scarce; we found one dataset that was well-formatted to work with our problem, but we had to do rigorous pre-processing on the dataset. After some research, we found a model that could help us with our project such as a recurrent neural network, but we had to modify the approach of the model to work with our problem. Another challenge that we faced was choosing what kind of jokes were appropriate for our dataset, and if we found jokes that were not appropriate for our dataset we had to filter those out using grep. Since we are using an unsupervised approach, our model will not have a baseline for determining funny from unfunny jokes; this makes it more difficult for our model to figure out why each joke in the training set is funny, and will likely lead to the bot making many jokes that are not funny (though hopefully many others will be funny).

# Solution

## Definitions

**TextGenRNN**
A python library for generating text using a recurrent neural network.

**Natural Language Processing (NLP)**
Using computer programs to analyze large amounts of natural language data.

**Recurrent Neural Network (RNN)**
A class of artificial neural networks. Connections between nodes in a recurrent neural network form a directed graph which allows for it to make predictions about what should come next.

**Long Short-Term Memory (LSTM)**
One drawback to RNNs is that they suffer from short-term memory loss. LSTM is a modified RNN which uses gates to selectively ‘forget’ information which is no longer useful. LSTMs are commonly used in training text generators.

## Text Generation

**Encoding**
Neural networks run on numbers and not strings or characters. In order to implement an RNN, strings from the dataset need to be tokenized (converted to numbers) and stored in a dictionary.

**Word-Level Generation**
Creating tokens from distinct words in the dataset. This method tends to result in more accurate results. The downside is that it requires a much larger dataset to build up a robust dictionary and will take longer to train.

**Character-Level Generation**
Creating tokens from individual characters in the dataset. These can be implemented very quickly with a much smaller model.

## Implementing TextGenRNN

**Data Pre-Processing**

A .txt file will be used to train the model. The delimiter type for the file will need to be set correctly. The file must be consistent and be cleaned of any invalid or undesirable sections.

**Configure the Model**
Parameters for the model must be set and can be adjusted to optimize results. Parameters include:

- word_level - True if using word level False if using character level
- rnn_size - The number of LTSM cells in each layer
- rnn_layers - The number of LTSM layers in the model
- rnn_bidirectional - True if reading text forwards and backwards
- max_length - Number of tokens used for predictions
- max_words - Maximum number of words to model

**Configure the Training**
These are the parameters for training the model:

- line_delimited - True if each piece of data has its own line
- num_epochs - How long the model will train for
- gen_epochs - When to generate samples
- train_size - Proportion of training data
- dropout - Proportion of tokens to ignore each epoch
- validation - Test on test data
- is_csv - True if csv file

**Train the Model**
Call TextGenRNN's train_from_file function with the configurations and run the model. Call TextGenRNN's generate_to_file function to store the output in a text file.

# Data

Our bot will  be run on a dataset called "shortjokes" which we downloaded from https://www.kaggle.com/abhinavmoudgil95/short-jokes. One issue with this dataset is that many of the jokes contain racial slurs and inappropriate content, so we will need to preprocess this dataset to remove any jokes which contain offensive or inappropriate content using grep.  The dataset currently contains ~226,000 jokes, of which more than 20,000 will be removed. Currently, the project is running on an arbitrary subset of 608 jokes which have been skimmed over manually (by eye) and cleaned accordingly.

Any additional datasets or updates will be added to the report at a later date.

# Analysis

Preliminary results suggest that our model needs to be tweaked, because the jokes it is currently outputting are not funny and they don't form coherent sentences. Part of the problem is that we are currently using a very small subset of the whole dataset. Using a larger dataset would provide a richer source of grammatically correct sentences for the model to learn from, leading to more coherent statements. Other methods we plan to explore for improving our model include bi-directional encoding and word-level analysis (as opposed to character-level analysis). Bi-directional encoding will analyze tokens both moving forward (left-to-right) and moving backward (right-to-left). This will increase the runtime of training but should also supply our model with a better understanding of context for making sentences: by considering not only the tokens that are likely to come before each new token but also the tokens that are likely to come after.

# Citations


[1] Badenhorst, A. E. (2017, December 28). 5 Motivating Reasons Why To Use Chatbots. *BuZZrobot*.Retrievedfrom[https://buzzrobot.com/5-motivating-reasons-why-to-use-chatbots-6610a0b56470](https://buzzrobot.com/5-motivating-reasons-why-to-use-chatbots-6610a0b56470)

[2] Waller, Annalu. Black, Rolf. O'Mara, David. Pain, Helen. Ritchie, Graeme. Manurung, Ruli. Evaluating the STANDUP Pun Generating Software with Children with Cerebral Palsy. ACM Transactions on Accessible Computing. Retrieved from https://dl.acm.org/doi/pdf/10.1145/1497302.1497306

[3] Mcgraw, Peter. Warner, Joel. Do animals have a sense of humour? Slate. Retrieved from https://www.newscientist.com/article/dn25312-do-animals-have-a-sense-of-humour/

[4] Harmon, Leon. Teaching AI to Be Funny: The Robot Uprising Won't Be a Laughing Matter. The Observer.
Retrieved from
https://observer.com/2019/07/teaching-artificial-intelligence-humor-robot-comedy/

[5] Stanford Encyclopedia of PhilosophyRetrieved from
https://plato.stanford.edu/entries/humor/#IncThe