**Project Lion Final Project Report**

Shun Lin, Jerry Chen, Allen Chen, Sakshi Madan

**Problem Statement and Background**

Computational humor is a branch of computational linguistics and artificial intelligence which uses computers in humor research. It is a relatively new area and therefore only a handful of research paper are currently tackle this field.

The goal of our project was to build a model that can generate good jokes that will bring the audience laughter and joy. In order to compare, we implemented four different models to see which one would perform the best. The models we built included a language model, a transformer model, an encoder-decoder model, and a model using pre-trained GPT.

In order to train our models we gathered datasets from several resources including Kaggle, GitHub, and Jester dataset from Berkeley Research[4]. We found that many of these datasets contained bad or racist content and therefore we filtered those out of the set. Our training set had approximately 230K jokes in our training set. The data consists of question/answer pairs and thus the input to each model was the question and the goal was to predict a joke response. In order to determine success of different models we generated both training and validation losses for each model. We also generated sample jokes for each model in order to see the validity and humor of each.

**Approach**

Data preprocessing

* The main dataset we utilized was the “shortjokes.csv” dataset from Kaggle, but we tried many different approaches to preprocessing our data depending on the model. For all models, we lowercased and removed most punctuation. We also filter out the jokes that contains racist, sexist, and swear words.
* Model 1 (LSTM / RNN with Keras) : We concatenate all the jokes we have in the dataset and treat the concatenated block of text as the input to train our LSTM and just simple map each word to an integer.
* Model 2 (Language Model from HW): We tokenize each joke in our dataset using tokenizer from segtok
* Model 3 (Encoder-decoder): First filter by question-answer jokes by splitting jokes by “?” and keeping the ones that result in two pieces (question and answer). Then remove jokes that have an answer length of 1 because these tended to be nonsensical jokes. This, however, significantly reduced the dataset size, resulting in a final size of 90K jokes, which we believe affected the training.
* Model 4 (Transformer from HW): Same as Encoder-Decoder
* Model 5 (pre-trained GPT): Used the entire dataset except for extremely short jokes of length 4 or shorter, which tended to be nonsensical jokes.

**Baseline model 1: LSTM / RNN with Keras**

For our first baseline model, we first started with a simple LSTM model, which stands for long short-term memory utilizing the many-to-many RNN architecture. After concatenate our jokes into a large block of text (each jokes separated by new lines), we generate our target sequence by creating the shifted version our block of text and slices of the block (batches) into our model. Our simple LSTM model is defined below:

Layer 1 - Embedding (keras.layers.Embedding)

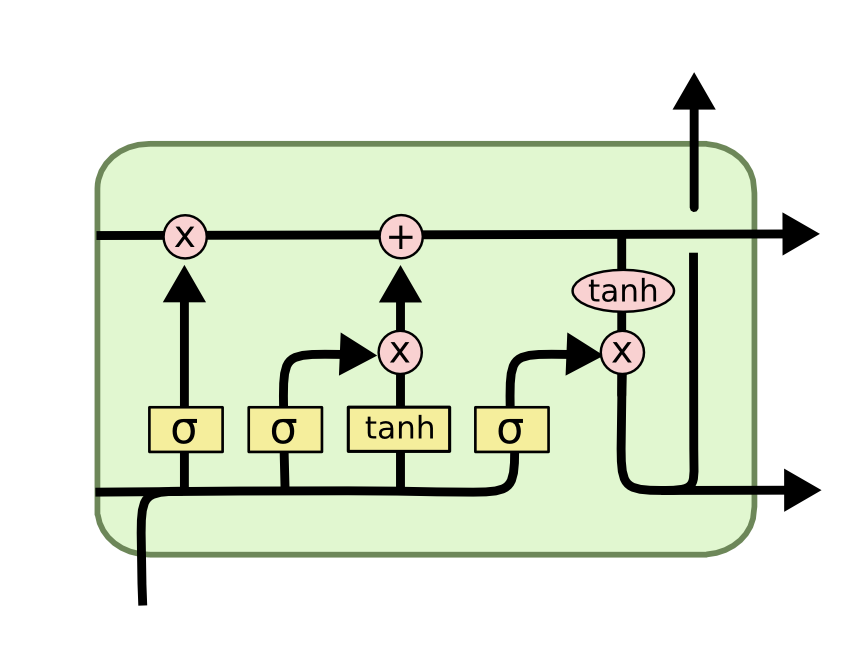
Layer 2 - LSTM (keras.layers.CuDNNGRU)

Layer 3 - Dense (keras.layers.Dense)

Layer 4 - Loss (tf.keras.losses.sparse\_categorical\_crossentropy)

We have two parameters choices here for our simple LSTM model -- the dimension for our embedding and the units for our LSTM. We have no choice over the dimension of the dense layer because it has to match our vocabulary size. We choose our embedding dimension to be 256 and our LSTM units to be 1024.

After training our model over multiple epochs on Google Colab we then give a starting phrase to our model and evaluate how our simple LSTM model generate new jokes. See the Sample Jokes section for a joke that is generated by our LSTM model.



**Baseline model 2: Language Model from Homework**

For our second baseline model, we built a language model for the jokes similar to how we built a language model for news headline in homework 3. After we have pre-processed and tokenized our jokes, we feed it through a LSTM that is very similar to the structure to the first baseline model we have described and implemented above. We have tried to implement MultiRNN cells in place of LSTM cells but found that it performs worst. The main difference between our first baseline model and second baseline model is how we preprocess and tokenize the data before feeding them into the network. The second difference is in how the trained weights are kept. For the first baseline model, we just keep saving the weights when we run through more epochs regardless of whether or not the validation error increases. For the second baseline model, we only save the model weights over epochs if the validation error decreases, and this helps us combat overfitting a bit and not just saving model weights that have high validation error and low training error.

Please See the Sample Jokes section for a joke that is generated by our Language Model model.

**Model 3: RNN Encoder-decoder model**

* 
* We built an encoder-decoder architecture using two RNNs (specifically LSTMs).
* For the word embedding, we trained a 128 dimensional embedding along with the model.

Model 4: Transformer

* Also an encoder-decoder model, but utilizing transformers. For both the encoder and decoder, we utilized multi-head attention with 8 heads, and a stack of 6 transformers (as in the “All You Need is Attention” paper).

Model 5: pre-trained GPT2

* Utilized the pytorch GPT2 model with OpenAI’s pretrained weights pulled from https://github.com/huggingface/pytorch-pretrained-BERT.
* Fine-tuned by performing additional training on our dataset.
* Trained on the entire model, but only the last few layers’ parameters were probably significantly changed, which is what we want.

**Results**

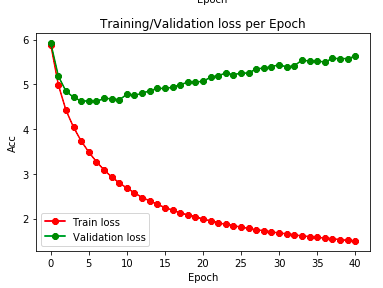


Figure 1: Training and Validation loss for our **Encoder and Decoder** Model

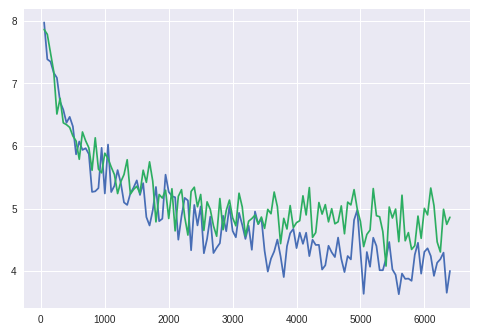


Figure 2: Training and Validation loss for our **Transformer** Model

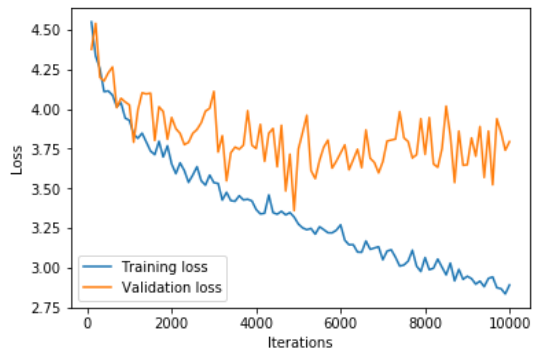


Figure 3: Training and Validation loss for our **GPT-2** with fine-tuning Model

Previously, we look at training and validation losses to measure the performance of each individual model, but the challenge now is to find a direct metric to directly evaluate the outputs of each model. One of the best NLP model now is BERT, which stands for Bidirectional Encoder representations from Transformers and obtains state-of-the-art results on many NLP tasks. We first pre-trained the existing BERT model with our questions and answers jokes corpus using its two losses. We then used the trained model to perform next sentence classification task on the outputs of each model, and the accuracy for each model is shown in the chart below.

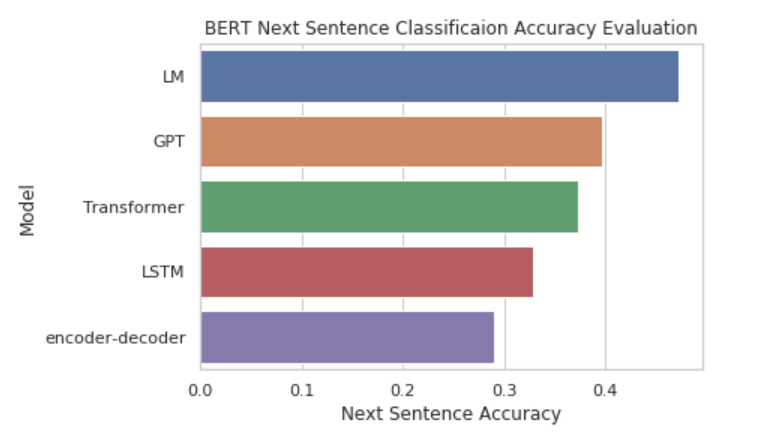


Figure 4: **BERT** next sentence accuracy comparison

By comparing the above graphs, we see that the **GPT-2** Model has the **lowest validation loss**. Most generated jokes ended up either not being funny, or does not make sense. However, our models do generate a few good jokes.

**Sample Jokes**

**GPT:**

*1. Why do chickens have hooves instead of feet? Cuz they have hooves instead of feet.*

*2. How do you make a baby cry twice? You slap the hell out of it.*

**Transformer**

*1. Which came first the chicken or the egg? The the rorooosterster did did the the fit*er:

**Language Model:**

*1. knock knock who's there ! ! ! who ? ! who ? i don't know , i just want to be a little bit . .*

**Basic LSTM Model:**

*1. Why did the chicken cross the road? Because he didn't want to watch TV to someone who had a moment when she was a second language.*

**Encoder-Decoder:**

*1. When are they going to drug test the audience of the price is right? losers policy wallaby sculptures duhh shouldve hooking untill quickie investment guinness*

**Tools**

**Lesson Learned**

We have learned multiple lessons from this project throughout the course of the semester. One main lesson we learned from completing this project is that it is essential to explore different models and approaches to solve the computational humor problem. Another lesson we learned is that the quality and quantity of our data really matters. Although we have gathered around a quarter millions of jokes from Kaggle and Github, our dataset was still fairly small, it was difficult to create answers to jokes that were meaningful or made sense most of the time. Thus we also used the pre-trained GPT-2 model with fine-tuning. As we can see from the above sampled jokes, jokes generated by our GPT-2 model does make more sense than others.

One of the main challenges we faced was preprocessing the data and determining proper hyperparameters to train a good model.

**Team Contributions**

**References**

[1] Kim Binsted. "Computational Humor". University of Hawaii. March/April 2016. URL:<http://www2.hawaii.edu/~binsted/papers/BinstedetalIEEEComputationalHumor2006.pdf>

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever. “Language Models are Unsupervised Multitask Learners”. OpenAI. URL: <https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf>

[3] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" October, 2018. URL: <https://arxiv.org/abs/1810.04805>

[4] Ken Goldberg. "Anonymous Ratings from the Jester Online Joke Recommender System". URL: <http://eigentaste.berkeley.edu/dataset/>