Project Pickup Milestone

Mackenzie Bookamer and Thalia Koutsougeras
21 March 2024

1. Problem Overview

In today's world, younger generations seem to struggle with communicating in person in the dating world, utilizing mobile apps instead such as Tinder. A pickup line generator can help ease this experience, including idioms and "hip" phrases. We intend to make this app tailored to STEM majors, and we were thinking of 3 specific categories, such as computer-science, math, or physics-related silly pickup lines. The user would choose a category and then input 1-2 words that they want contained in the generated pickup line (or a similar word). This project aims to be very user friendly and incorporate lots of user interaction to create an engaging experience.

2. Data

We will train our model on computer science, math, and physics jokes and pickup lines as well as English jokes. For this milestone, we fed the model only computer science jokes and pickup lines, but for the final project, we will have 3 different models, one fine-tuned for each topic. For all of the models, we will also feed them some English jokes just so that we can have as much data as possible, and to make sure that the model understands the joke format we are looking for. This data is scraped from websites and blog posts and compiled into 1 CSV-UTF8 file with 614 rows.

In order to generate text, we won't be training on the entire English language, but will instead be implementing a large language model - GPT-2, further discussed below.

3. Method

Our project is focusing on fine-tuning the pretrained GPT-2 model to the data we described above. We will be using GPT-2 (medium) because it is still a robust dataset – about 350 million parameters – without occupying an enormous amount of memory on our computers – only 1.5 GB. The benefit to using a pre-trained language model such as GPT-2 is that there already exists open-source code for the tokenize function, which is what we will use to fine tune our model. Before we generated responses on a fine tuned model, we wanted to see what the generated text given a custom input would be for the baseline GPT-2 model (i.e. downloaded from the OpenAI gpt-2 repo). We made adjustments to the interactive_conditional_samples.py file to generate results more in line with our specific project goal. Below are the attributes we wanted to adapt to our project and the code we wrote to implement it:

- Only run the 355M GPT-2 model: *model_name='355M'*
- Decrease the length of the generated output: *length=50*
- Increase the number of generated samples output: *nsamples=2*
- Consider less words at every step (i.e. 40 words at every step instead of every word): *top_k=40*
- Decrease the randomness of output completion: *temperature=0.4, top_p=0.9*

Running the not fine tuned model of GPT-2 allowed us to become familiar with how the model generates text and what parameters are the most influential in terms of generated output. Quite honestly, it took us a long time to get GPT-2 up and running because of all of the libraries and specific versions we needed, as well as reading through the documentation and noticing some small things we had to change in files. We also implemented few-shot learning to train the model on a very limited number of jokes to see if it would mimic the same syntactic style when it generated an output. The low temperature setting meant the model kept repeating itself, however, so we began to increase our temperature setting in our fine tuned model.

Through the HuggingFace transformers library, we accessed the function *GPT2Tokenizer*, which we then used to train our model on our specific dataset of jokes and pickup lines. Once the function splits the input into individual tokens, it maps each token to its unique UTF-8 ID that the GPT-2 model can then process. This process of tokenization is vitally important because it allows us to specifically train our GPT-2 model to generate text that is similar to our text of interest: pickup lines and jokes.

To optimize the result of our tokenizer function, we can calculate the cross-entropy loss and work to minimize the value. This is the loss function commonly used with classification tasks and it is currently implemented in the Trainer function in our finetune.py file. The Trainer class is another import from the HuggingFace transformers library which abstracts the process of computing loss functions, making it more user friendly. Note we split our input into a test and train data set for the Trainer to compute the loss function. Once we train our GPT-2 model on more datasets, we will go back and carefully perform gradient descent utilizing the cross-entropy loss function to ensure we are using the optimal hyperparameters.

After we trained our GPT-2 model four times with our finetune.py file, we then ran roundx.py, where we replaced ‘x’ with the current iteration we were on. Within these files, we increased the temperature setting to 0.7 to hopefully decrease the amount of repetition in the response, while also increasing our top_k to 50 and max_length to 100 to introduce more ‘controlled randomness’ in our model.

In order to evaluate our results, we could have the user rate on a scale of 1-5 whether or not the pickup lines made sense (or we could do a binary scale of 0 or 1 funny/not funny or made sense/did not make sense) and use these results for sentiment analysis as well, and this would serve as our performance metric. Our baseline method can be generating pickup lines by randomly combining words and phrases without the specific category aspect. An additional performance metric we could use is uniqueness (as compared to the baseline method). In terms of data visualization and analysis, we could use a confusion matrix if we use a binary classification scale as mentioned above. Per our fifth article ('Computer Tell Me a Joke…), we can use the semantics outlined to determine if logically and theoretically the output generated was funny (aka a joke). We can use a histogram to plot different examples of pickup lines and their ratings.

4. **Intermediate/Preliminary Experiments & Results**

Below are results obtained from the different types of models explained above:

- Baseline GPT-2 generating next words in sentence:

```
Model prompt >>> it is a wonderfully sunny
======================================== SAMPLE 1 ========================================
 day, and I am sure that you are all enjoying it. I have a few things to tell you, however. First, I have been in contact with the local police and they have informed me that they are investigating the matter. They have not
======================================== SAMPLE 2 ========================================
 day, but the wind is blowing and the clouds are thick. It is a perfect day for a picnic. The picnic is located at the base of the mountain. The picnic area is about 3.5 km from the summit. The picnic area is
================================================================================
```

- Few-Shot Training for baseline GPT-2:

```
Model prompt >>> Given these 3 examples, give me another joke:
Are you a keyboard? Because you're my type.
Are you a computer virus? Because my heart skips a beat every time I see you.
Is your name Google? Because you have everything I've been searching for.
======================================== SAMPLE 1 ========================================
"I'm so sorry I couldn't help you, but I have to tell you about a really great guy who I met in college. He was a really great guy. He was a really great guy. He was a really great guy
======================================== SAMPLE 2 ========================================
1. "I was at a party where I was asked to sing a song. I was very nervous and didn't know what to sing. I was very nervous, but I sang the song. I thought I was going to be in
================================================================================

Model prompt >>> 1. Are you a keyboard? Because you're my type. 2. Are you a computer virus? Because my heart skips a beat every time I see you. 3. Is your name Google? Because you have everything I've been searching for.
======================================== SAMPLE 1 ========================================
 4. Are you a computer virus? Because you have everything I've been searching for. 5. Are you a computer virus? Because you have everything I've been searching for. 6. Are you
======================================== SAMPLE 2 ========================================
 4. Are you a cat? Because you are a cat. 5. Are you a dog? Because you are a dog. 6. Are you a cat? Because you are a cat. 7. Are you a dog? Because you are a
================================================================================
```

- Fine tuned model generating the rest of the given pickup line:

```
(nlp-virtual) (nlp-virtual) mackenziebookamer@Mackenzies-MBP gpt-2 % python3 round1.py
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please
pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generated text:
Are you a developer? Because...  I'm not.  I'm a programmer.  I'm a programmer who likes to write code.  I'm a programmer
who likes to write code.  I'm a programmer who likes to write code.  I'm a programmer who likes to write code.  I'm a
programmer who likes to write code.  I'm a programmer who likes to write code.  I'm a programmer who likes
================================================================================

(nlp-virtual) (nlp-virtual) mackenziebookamer@Mackenzies-MBP gpt-2 % python3 round2.py
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please
pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Generated text:
Are you a developer? Because I want to...   I want to be a developer.  I want to be able to build things.  I want to be able to
write code.  I want to be able to write code that's easy to read.  I want to be able to write code that's easy to understand.  I
want to be able to write code that's easy to debug.  I want to be able to write code
```

It is worth mentioning that our fine tuned model did not produce the results we hoped for, since after training it with 600 lines of data yielded the same generated text response as training the model with 100 lines of data given the same prompt. We hope to explore this issue before the final project with more data and more play on the hyperparameters, and having a few differently focused models will also help us to diagnose and solve the issue.

5. **Related Work**

Here we examine a few related research papers.

1. Generating Original Jokes[1]

This article was written before OpenAI and ChatGPT's rise in popularity, so a PLM as robust as GPT-2 was not used. This paper also broadly focuses on *all* jokes, not topic-specific. This is where our project differs and makes it unique. It does give good ideas about how to validate and evaluate our output by having users compare the output to a common, human-generated output and determine which they like better.

2. Pretrained Language Models for Text Generation: A Survey[2]

This review article explains in detail PLM's, starting broadly and then delving into the steps involved. The authors explain how to encode an input, describe different methods of fine-tuning, and challenges implicated in the process.

3. Deep Learning of a Pre-trained Language Model's Joke Classifier Using GPT-2[3]

This article focuses on comparing the accuracy of classification models – namely BERT, CNN, and RNN – to correctly identify a joke generated from a PLM – GPT-2. They utilize fine-tuning algorithms on GPT-2 to generate the jokes, which is what we aim to do.

4. Improving Language Understanding by Generative Pre-Training[4]

The main point of this article is using generative pre-training of a language model, followed by discriminative fine-tuning. This could be useful to us because we could possibly pre-train our PLM with English jokes and specific category definitions, and then fine-tune with the category-specific pickup lines.

5. Computer, Tell Me a Joke ... but Please Make it Funny: Computational Humor with Ontological Semantics[5]

This article is different from the rest as it focuses primarily on the linguistic side of jokes and humor – the underlying theory behind the models proposed above. When evaluating the pick-up lines generated by our project, we can use the semantics presented in this article to objectively classify jokes.

6. **Division of Labor**

As of now, we have been working side-by-side on the project just working on Mackenzie's computer, learning how to set up virtual environments, downloading GPT-2 to play with, etc. Thalia has been working with her and has been completing other tasks, such as working on getting data while Mackenzie was working on the setup process and installing libraries. We also worked on the fine tuning code together, and we feel that the side-by-side method works best for us, so we will continue with this. Once we feel confident with our perfected first model for computer science, we can each work on fine-tuning the physics and math joke models separately.

7. **Timeline**

4/1-4/7 - Work on CS Model

4/8-4/14 - Perfect CS Model, begin working on math/physics

4/15-4/21 - Work on math/physics models

4/22-4/25 - Polish, work on final report

4/25 - *Final presentations*

References

1. **Generating Original Jokes**, Yeh, T. et al., *Santa Clara University*, March 2018, https://www.engr.scu.edu/~mwang2/projects/NLP_generateOriginalJokes_18w.pdf.
2. **Pre-trained Language Models for Text Generation: A Survey**, Li, J. et al., *IJCAI*, May 2022, https://doi.org/10.48550/arXiv.2105.10311.
3. **Deep Learning of a Pre-trained Language Model's Joke Classifier Using GPT-2**, Akbar, N. et al., *Journal of Hunan University*, Aug 2021, http://www.jonuns.com/index.php/journal/article/view/671/666.
4. **Improving Language Understanding by Generative Pre-Training**, Radford, A. et al., *OpenAI*, June 2018, https://openai.com/research/language-unsupervised.
5. **Computer, Tell Me a Joke ... but Please Make it Funny: Computational Humor with Ontological Semantics**, Hempelmann, C. et al., *The Florida AI Research Society*, 2006, https://api.semanticscholar.org/CorpusID:892815.

Data Websites

1. https://wittypro.net/pick-up-lines/computer-programming-pick-up-lines-and-rizz/
2. https://www.reddit.com/r/ProgrammerHumor/comments/oz77xb/programmer_pickup_lines_ive_collected_these_past/
3. https://punsteria.com/computer-science-puns/
4. https://attractmorematches.com/computer-science-pick-up-lines/
5. https://pickupline.net/academic-pick-lines/computer-programming-pick-up-lines/

Useful Links

1. https://github.com/openai/gpt-2/tree/master
2. https://huggingface.co/docs/transformers/en/training
3. https://huggingface.co/docs/transformers/en/index
4. https://huggingface.co/docs/transformers/main_classes/tokenizer