AAI-511 Final Team Project Report - Group 4

Natural Language Processing:

Advanced Generative Chatbot Design

by

Hani Jandali, Kay Cheung, Suvo Ganguli

University of San Diego

Oct 21st, 2024

1. Introduction

In this final project report for the Natural Language Processing course (AAI520), we present the design, development, and evaluation of a conversational chatbot. The chatbot was trained using the Cornell Movie Dialogues Corpus, a popular dataset that provides natural conversational exchanges in a wide range of settings. The goal of the project was to explore different modeling techniques for generating conversational agents, with a focus on both the performance and the quality of the interactions.

Two distinct models were developed and tested during the course of the project. The first model leveraged GPT-2, a state-of-the-art generative language model known for its impressive text-generation capabilities. However, despite its strong performance in many other natural language generation tasks, the GPT-2 based chatbot underperformed in terms of conversational quality. It often produced generic or neutral responses, resulting in interactions that lacked coherence and engagement—key elements for a conversational agent.

The second model utilized a sequence-to-sequence (seq2seq) architecture, which is particularly well-suited for tasks like machine translation and dialogue generation. This model produced more conversationally appropriate responses, leading to a more dynamic and engaging interaction between the user and the chatbot. The seq2seq model met the project's expectations by delivering coherent, contextually relevant replies, making it the more successful of the two models.

The final implementation was deployed through a **web interface** designed using Flask, allowing users to interact with the chatbot in real-time. This interface provided an accessible and user-friendly platform for testing the effectiveness of the chatbot, showcasing the practical applications of natural language processing in conversational AI.

2. Dataset

The Cornell Movie Dialogues Corpus, used for training the chatbot in this project, is a rich collection of fictional conversations extracted from raw movie scripts. Available on Kaggle, this dataset includes 220,579 conversational exchanges between 10,292 pairs of characters from 617 movies. It consists of 304,713 utterances spoken by 9,035

unique characters, making it an ideal resource for dialogue-based language modeling. The data also includes metadata about the movies, such as genres, release years, IMDB ratings, and number of votes, as well as information about the characters, including gender and credit positions within the film

Each conversation is organized in a way that captures the natural flow of dialogue, making it particularly suited for conversational AI development. The dataset's broad genre coverage (e.g., comedy, drama, action) allows for diverse linguistic styles, which helps in training models that can respond in contextually appropriate ways.

For this project, the dataset was accessed from Kaggle and used to train two different models (GPT-2 and Seq2Seq), helping to highlight the effectiveness of different neural architectures in generating conversational agents.

3. Chatbot 1: GPT2

The first method involved the development of a chatbot using GPT-2, a state-of-the-art transformer-based language model designed for generating coherent text across a variety of contexts. This section details the process of loading, fine-tuning, and evaluating GPT-2 on the Cornell Movie Dialogues Corpus. It highlights the technical steps and results.

The dataset, stored in Google Drive, was loaded into the Colab environment. The two main files, `movie_lines.txt` and `movie_conversations.txt`, were parsed. The movie lines file contained individual lines of dialogue, while the conversations file mapped these lines into structured dialogues between characters. The lines were processed and stored in a dictionary, where each line ID was associated with its corresponding text. The conversations were reconstructed by aligning line IDs, ensuring that all conversation sequences were preserved. Once the dialogues were loaded, input-output pairs were generated by selecting a sliding window of two previous lines as input (context) and the next line as the target. This approach provided the model with structured conversational contexts from which to learn dialogue flow.

To process the text into a format suitable for GPT-2, the GPT-2 tokenizer was used, which converts input text into numerical tokens that the model can understand. The tokenizer's `pad_token` was set to the end-of-sequence (EOS) token to ensure proper padding during the training process. A custom dataset class was implemented, extending

PyTorch's `Dataset` class, to tokenize both inputs and targets, allowing for the proper structure to be passed to the model during training.

The GPT-2 model was fine-tuned on the Cornell Movie Dialogues Corpus using the Hugging Face Trainer API. This process involved setting several training hyperparameters, such as learning rate, batch size, and number of training epochs. A key aspect of the training setup was the use of early stopping to prevent overfitting by halting training if no improvement was seen over multiple evaluation steps. The fine-tuning process also leveraged nucleus sampling (`top_p=0.9`) and a temperature setting (`temperature=0.5`) during generation. This helped control the diversity and randomness of the generated responses. Lower temperature values ensured that the chatbot's outputs were more deterministic and coherent.

Once fine-tuned, the GPT-2 model was used to generate responses to user prompts. The `generate_response()` function allowed for interactive chatbot-style conversations, producing output by predicting the next word based on the context provided by the input. To ensure meaningful responses, settings like no-repeat n-gram size (to prevent repetitive phrases) and early stopping (to conclude a response at a complete sentence) were incorporated.

Here is a sample conversation using the GPT-2 chatbot:

User: I love movies, do you? Chatbot: I love them.

User: Do you like pizza? Chatbot: I'm not a pizza guy.

User:: Do you like popcorn? Chatbot: I'm not sure.

User: What is your favorite movie? Chatbot: I'm not sure.

Despite GPT-2's strong capabilities, the results were suboptimal for this specific chatbot task. The model tended to produce neutral and often repetitive responses, failing to exhibit the level of conversational depth expected. This outcome highlights the challenge of adapting large generative models like GPT-2 to specific dialogue tasks without extensive tuning and domain-specific training data.

4. Chatbot 2: Seq2seq

Realizing some limitations in applying a pre-trained L.L.M. to the movie corpus, our alternative approach was a Sequence-to-Sequence (Seq2Seq) model with specialized attention mechanisms built through PyTorch. The first step in our approach was preprocessing the movie corpus in a slightly different manner, borrowing from Sai Prashanth (2017) and creating sentence pairs consisting of contexts and their associated responses while eliminating all other features to reduce noise. Within the resulting dialogs, contractions were expanded and eliminated, and utilizing Kazimierz Papis' (2016) approach to removing accents, terms were normalized and lowered, with punctuation, whitespace, and non-alphabetic symbols removed. We further preprocessed our resulting text pairs by removing sequences greater than ten words, and terms appearing fewer than two times. In this way, we can reduce the size of our overall vocabulary and decrease memory load in further steps while reducing the time complexity of generating outputs. Another added benefit to this trimming and maximum length limitation was faster convergence during training and evaluation and more contextual capture in our model's responses, which was thoroughly difficult challenge in our previous attempts. Resource management and time complexity was a recurring theme in our sequence-to-sequence model implementation as our previous attempts with multi-million parameter L.L.M.'s proved intense. As a result, we also prepared our model with with parallelization in mind, utilizing G.P.U. resources when possible (with exception of storing tensors on C.P.U. as expected in PyTorch). Preprocessing continued by assigning each unique word an index value which was subsequently converted into tensors for model training. Accounting for contiguous issue's previously experienced in our first attempt, we padded shorter sentences with zeros after an E.O.S. token to standardize the tensor shapes to our maximum input length regardless of actual length. Input tensors were adjusted so that the shape reflected the batch size and max sequence limit of ten words, allowing for efficient indexing across training iterations. The input batch shape was transposed to accommodate this structure, ensuring that each batch could be processed in parallel.

The choice of a sequence-to-sequence model was not random, but decided due to two crucial components: encoder and decoders. The encoder was designed to handle variable-length input sentences and transform them into fixed-length context vectors. However, it's highlight was the bidirectional gated recurring unit, enabling the user processed text sequences to be processed forward and backwards before being summed up. As with the then revolutionary B.E.R.T. models, the encoder's bidirectional G.R.U. enabled our model to capture dependencies from

both directions of the input sequence, improving contextual response and capturing longer dependencies. After processing each word at a time through the bidirectional G.R.U., the resulting output was forwarded and summed up, producing a final hidden state representing the entire input sequence.

The decoder took in the aforementioned final hidden state as it's initial hidden state, taking an input word (either from previous time step or, during training, the ground truth word by means of teacher forcing) and generating the next word in the sequence. The decoder continued the process until an E.O.S. token was reached. Despite a maximum length of ten terms, in early iterations of testing our initial models, the decoder's ability to handle sequences was still subpar, and therefore we implemented a Luong attention mechanism (Luong et. al, 2015). This small change resulted in incredible gains, allowing the decoder to focus on relevant parts of the encoders outputs at each step and resultingly enhancing the quality of our model's response. Attention weights were calculated at each time step based on the current hidden state before being applied to the encoders outputs to produce a weighted sum context vector. The context vector, alongwith the hidden state, guided our decoder's next output.

To further improve any limitations to our model's implementation, during training, we applied teacher forcing to improve the model's learning. At each training iteration, there was a probability (determined by our teaching_forcing_ratio) that the actual target word, rather than the decoder's predicted word, would be used as the next input (Albert et. al, 2017). Although intended to improve the model's learning capability, teacher forcing yielded the extended benefit of speeding up training, freeing up resource management which was strict in this approach. However, instability began to appear with longer iterations of training in the form of exploding gradients, to which we modified our model to include gradient clipping in order to control weight values after a certain point.

In training, we often set high iteration limits of five to ten thousand, sometimes upwards of fifty thousand in experimentation, with each iteration carrying the processes of taking the input batch and passing it through the encoder and decoder. If teacher forcing was applied in that particular iteration, the target word was fed into the decoder as the output. However, when teacher forcing was not in use, we employed greedy search to select the word with the highest softmax probability at each time step to produce the decoder's output. In practice, we noticed that it sometimes lead to subpar sentence generation, but it's simplicity in implementation and computational efficiency

made it sufficient enough for our chatbot. Afterwards we implemented an evaluation function to handle user inputs and test the model's performance, sustaining turn by turn conversations.

Computational and time complexity was a serious problem throughout creating the model, as emphasized by our continual consideration of the two proponents. It is for this reason that we pursued our own preprocessing techniques instead of instantiating tokenization via NLTK or SpaCy, avoided L.L.M.'s with millions of parameters, and pursued a Greedy Search Decoder and dot attention calculation in our attention mechanisms. Although it does deteriorate the performance of the chatbot to some degree, it enables a quicker compilation of said chatbot.

5. Deployment (Web Interface)

Flask is a Python library that makes it easy to build web servers. We used it to serve our model through an API. The chat UI was coded using html and javascript. The javascript would send a request to the web server for a chat response, and then append it to the chat area once it was received. By creating a web server using Flask, we can locally test that our webpage is integrated with the web server. We used Ngrok to expose this local web server to the internet through a public URL. Ngrok does this by creating a secure tunnel from the internet to your local machine. We used Github pages to publish the chat UI. Because Ngrok changes the URL each time you run it, we add an input field at the top of the chat UI to enter the URL of the web-server.

One of the challenges we encountered was deploying a custom NLP model on the Flask app. We initially intended to use a model trained from scratch for our project, but faced difficulties in loading and deploying it. It was a custom model with custom layers, and we couldn't figure out how to load the model in the Flask app. In the future we would put the model code in a library file and import it both in the Jupyter notebook and the Flask app so that we can test that the same code running in the Flask app works in the notebook. As a workaround, we used GPT-2, a pre-trained model, which we fine-tuned on the movie conversation dataset. This allowed us to move forward with the project, even though it wasn't our original plan. Both the fine-tuned GPT-2 model and the model trained from scratch gave English sounding responses, however both often gave nonsensical responses.

Chatbot link**:** https://kay-q-mich.github.io/AAI520_FinalProject/

6. GitHub Link

Project link: https://github.com/suvoganguli/AAI520_FinalProject

References

Aladdin Persson. (2020, June 8). *PyTorch Seq2Seq with Attention for Machine Translation* [Video]. Youtube. https://www.youtube.com/watch?v=sQUqQddQtB4

Albert, C. (2017). Exploring teacher forcing techniques for sequence-to-sequence abstractive headline summarization.

Bahdanau, D. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bentrevett. (2024, January 20). PyTorch-Seq2Seq. *Github*. https://github.com/bentrevett/pytorch-seq2seq

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chidananda, R. (2017). Cornell Movie Dialog Corpus. *Kaggle.* https://www.kaggle.com/datasets/rajathmc/cornell-moviedialog-corpus

Chiren Chen. (2023, June 10). *PyTorch Seq2Seq Translator From Scratch: Attention & RNN (GRU/LSTM) Part 1* [Video]. Youtube. https://www.youtube.com/watch?v=XDRFam-wV5I

Chiren Chen. (2023, June 10). *PyTorch Seq2Seq Translator From Scratch: Attention & RNN (GRU/LSTM) Part 2* [Video]. Youtube. https://www.youtube.com/watch?v=2QlN16pH64o

Cristian Danescu-Niculescu-Mizil. (2023). *Cornell Movie-Dialogs Corpus* [Data set]. ConvoKit Developers. https://convokit.cornell.edu/documentation/movie.html#contact

Doshi, K. (2021, May 9). Foundations of NLP Explained - Bleu Score and WER Metrics. *Towards Data Science.* https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b

Etienne, B. (2018, November 29). Attention Seq2Seq with PyTorch: Learning to Invert a Sequence. *Towards Data Science.* https://towardsdatascience.com/attention-seq2seq-with-pytorch-learning-to-invert-a-sequence-34faf4133e53

Feng, L., Tung, F., Ahmed, M. O., Bengio, Y., & Hajimirsadegh, H. (2024). Were RNNs All We Needed?. *arXiv preprint arXiv:2410.01201*.

Kumaran, G. (2020, May). Seq2Seq Model with Attention for Time Series Forecasting. *PyTorch*. https://discuss.pytorch.org/t/seq2seq-model-with-attention-for-time-series-forecasting/80463

Luong, M. T. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Mun, J. (2023, January 31). AI Chatbots with TensorFlow.js: Creating a Movie Dialogue Chatbot. https://guoruiming.com/Course/Programming%20Language/JavaScript/tfjs/chatbot/05-movie-dialog-chatbot/

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., & De, S. (2023, July). Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning* (pp. 26670-26698). PMLR.

Robertson, S. (2024). NLP From Scratch: Translation with a Sequence to Sequence Network and Attention. *PyTorch.* https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Sai Prashanth S. (2017, January 6). Textutil-Preprocess-Cornell-Movie-Corpus. *Github*. https://github.com/floydhub/textutil-preprocess-cornell-movie-corpus

Shawar, B., Atwell, E. (2007). Different Measurements Metrics to Evaluate a Chatbot System. *Proceedings of the NAACL'07 Workshop: Bridging the Gap: Academic and Industrial Research in Dialog Technologies*. https://dl.acm.org/doi/10.5555/1556328.1556341

Sojasingarayar, A. (2020, May 25). Seq2Seq A.I. Chatbot with Attention Mechanism. *Department of Artificial Intelligence University-GEMA Group Boulogne-Billancourt.* https://arxiv.org/pdf/2006.02767

Spro. (2018, July 28). Practical Pytorch. *Github*. https://github.com/spro/practical-pytorch/tree/master

Vikas. (2024, March 1). Quantitative Metrics Simplified for Language Model Evaluation. *Analytics Vidhya.* https://www.analyticsvidhya.com/blog/2024/03/quantitative-metrics-simplified-for-language-model-evaluation/

Kazimierz Papis, B. (2016). Best Way to Remove Accents (Noramlize) in a Python Unicode String. *StackOverflow. https://stackoverflow.com/a/518232/2809427*