Generating Replies based on Movies and Shows Scripts

Yomna Ayman, Yara Ayman

May 19, 2024

Introduction

The ability to generate human-quality text has become increasingly important in recent years, with applications ranging from chatbots and virtual assistants to creative writing tools and marketing materials.

The field of Natural Language Processing has seen remarkable progress in text generation models. These models, fueled by vast amounts of text data, can produce human-quality, grammatically correct, and coherent text. This opens doors for applications in dialogue systems, machine translation, and even creative writing tools.

In this report, we develop a text generation model, this model generates replies based on scripts of movies, tv shows, etc. Our model was trained on the script of the popular series Game of Thrones.

Motivation

Traditional narratives offer a linear experience, limiting reader engagement. The creation of our model is motivated by the potential of text generation models to revolutionize storytelling through interactive dialogue. By analyzing vast script datasets, the models can learn the intricacies of narrative structure and dialogue flow. This knowledge allows them to generate new dialogue that aligns with user choices, fostering an interactive experience.

Furthermore, by analyzing character-specific dialogue patterns, the model can learn to generate dialogue that remains consistent with established character voices, further immersing readers in the narrative world. In essence, this report explores the potential of text generation models to create a more dynamic and engaging storytelling experience through interactive dialogue.

Literature Review

0.1 Large Language Models: ChatGPT and Gemini

The field of text generation has witnessed a significant paradigm shift with the emergence of large language models **LLMs**. These powerful deep learning models trained on massive datasets of text and code have revolutionized the way computers interact with language. We will delve into two prominent LLMs at the forefront of text generation: ChatGPT and Gemini [RCR24].

Both models leverage advanced neural network architectures and vast amounts of training data to generate human-quality text, translate languages, write different kinds of creative content, and answer questions in an informative way. In the following sections, we will explore the highlights of the architectures and applications of ChatGPT and Gemini, providing a deeper understanding of their capabilities and potential impact on various fields.

0.1.1 Gemini: Architecture and Applications

Developed by Google AI, Gemini comprises a family of LLMs, with Gemini Ultra 1.0 being the most advanced version. While specifics of its architecture remain confidential, the paper highlights its status as a large language model, implying a complex neural network architecture incorporating techniques like transformers or recurrent neural networks **RNNs** to process sequential language data.

A key differentiator for Gemini is Retrieval-Augmented Generation **RAG** technology. **RAG** integrates information retrieval with text generation, allowing Gemini to access and leverage external knowledge sources during the generation process. This results in factually grounded outputs, making Gemini particularly valuable for tasks requiring accuracy and up-to-date information. Additionally, Gemini is trained on a diverse dataset, enhancing its versatility across various tasks.

Applications:

- Creative Text Generation: The paper suggests Gemini's capabilities extend to generating different creative text formats.
- Machine Translation: The paper indicates that Gemini can translate languages.
- Informative Question Answering: The paper emphasizes Gemini's ability to answer your questions in an informative way, making it a valuable tool for research and knowledge retrieval.

0.1.2 ChatGPT: Architecture and Applications

ChatGPT, developed by OpenAI, is renowned for its conversational abilities and creative text generation. At its core lies the Generative Pre-training **GPT** architecture, a powerful neural network approach specifically designed for text generation tasks. The **GPT** architecture excels at predicting the next word in a sequence of text data. This ability allows ChatGPT to generate fluent and human-quality text by iteratively predicting the most likely word to follow the existing sequence.

To further refine its responses, ChatGPT incorporates Reinforcement Learning with Human Feedback **RLHF**. This approach allows the model to learn from human interaction and improve its outputs based on user preferences . Additionally, ChatGPT utilizes instruction tuning, making it adaptable to specific tasks and contexts.

Applications:

- Conversational AI: The paper identifies ChatGPT's strength in various conversational AI applications, particularly in customer service, construction, and healthcare. This suggests its ability to handle open ended communication and adapt to different conversational styles.
- Due to its focus on creative text generation, ChatGPT likely finds applications in similar areas as Gemini, such as generating marketing copy or creative writing prompts.

It is important to note that the applications of Gemini and ChatGPT are much more versatile than those mentioned in the paper.

0.2 Sequence-to-Sequence Learning: The Engine Behind Our Model

Within the realm of text generation models, a powerful approach known as Sequence-to-Sequence learning **Seq2Seq** has emerged. This technique excels at transforming one form of sequential data into another. Given its effectiveness in tasks like machine translation and dialogue generation, Seq2Seq forms the core architecture of our text generation model. In the following section, we will delve deeper into the inner workings of Seq2Seq models, exploring their components and how they enable the generation of creative and engaging narrative dialogue.

Conversational agents need to deal with sequential data, mainly text. Thus, conversational agents adopt specific Neural Network architectures designed for sequence processing. These architectures for sequence modeling are often called Sequence-to-Sequence since they take sequences as input and produce sequences as output. With the spread of Deep Learning techniques in NLP and the availability of large, pre-trained, Language Models, Seq2Seq have become the standard architecture to solve many NLP-related tasks, including the development of generative chatbots. The application of such architectures, however, is neither limited to generative solutions nor restricted to open-domain conversational agents. In fact, the Seq2Seq architecture is actually compatible with retrieval chatbots or task-oriented agents. In general, Seq2Seq can be seen as a very generic and powerful tool for dealing with a broad range of NLP tasks [SST23].

Due to language's temporal structure, standard Neural Networks are unsuitable for language modeling. In fact, to model human language it is necessary to adopt models capable of capturing long-term sequential dependencies in a variable-length context. To this end, convolutional, recurrent and transformer networks are the models of choice as they allow to leverage the spatial structure of the input. Despite convolutional networks being a valid tool for sequence processing, we will focus solely on recurrent networks..

Recurrent Neural Networks were the first proposed solution for applying Artificial Neural Networks to a possibly unbound time horizon. In other words, such networks can be applied to a sequence whose length is neither known a priori nor bounded to some maximum value. In a recurrent layer, a cell takes part in a loop where a hidden vector, representing the sequence's past , is recurred through the sequence steps.

Several variants of recurrent networks exist, vanilla Recurrent Neural Networks RNNs, Long Short Term Memories **LSTMs** and Gated Recurrent Units **GRU**. All these networks can be used to scan a sequence from left to right or vice versa. It is also possible to have bi-directional networks to include information from both sides of the sequence when processing each sequence element.

We will focus on the LSTM variant. LSTMs were designed to overcome some limitations of the original RNNs . Despite the sound mathematical formulation behind RNNs, training those networks turns out to be a hard task . Standard RNNs struggle to remember information from far back in the sequence. LSTMs address this by incorporating "gates" that control information flow. These gates decide what new information to remember, what past information to forget, and what information to ultimately output. This allows LSTMs to learn long-term dependencies within sequences, making them ideal for tasks like machine translation and speech recognition.

0.2.1 Applications of Seq2seq Models

Seq2Seq models, demonstrating prowess in handling sequential data, have become a cornerstone for various NLP Applications. They power machine translation systems, enabling seamless translation between languages. Seq2Seq models can also condense lengthy text into concise summaries, making them valuable for summarizing articles or research papers. Furthermore, these models lie at the heart of chatbot development, allowing chatbots to understand user queries and respond in natural language. Beyond these core applications, Seq2Seq models can be used for creative text generation tasks and even assist programmers with code completion and correction. As research continues to explore their potential, the applications of Seq2Seq models are certain to expand even further.

1 Data Analysis

1.1 Dataset Description

The dataset for this project was obtained from Kaggle

1.1.1 Data Format

The data is formatted as a comma-separated values (CSV) file, a common format for tabular data. The file is structured with columns providing specific information about each line of dialogue:

- Speaker Name: Identifies the character speaking the line.
- Sentence: The actual spoken dialogue itself.
- Episode Title: The title of the episode where the dialogue appears.
- Episode Number: The numerical episode number within its season.
- Season Number: The season number within the Game of Thrones series.
- Release Date: The date the episode originally aired.

1.1.2 Data Content

The dataset encompasses around 24,000 rows, representing individual lines of dialogue. This translates to a significant amount of conversational data from the series. The dataset includes the complete script for Game of Thrones, spanning all eight seasons (Seasons 1 to 8). This ensures a comprehensive representation of character interactions and speech patterns throughout the entire show.

1.1.3 Data Preprocessing

To prepare the data for training our NLP model, we performed the following preprocessing steps:

- Column Removal: We removed the "Release Date" column as it wasn't relevant for our goal of generating replies.
- Lowercasing: We standardized all text to lowercase for consistency in the model's training process.
- Contraction Expansion: We expanded contractions (e.g., "won't" to "will not") to improve the model's understanding of natural language.
- Punctuation Removal: We removed punctuation marks (commas, periods, etc.) to focus on the core word sequence within the dialogue.
- Stop Word Removal: We removed common stop words (e.g., "the", "a", "is") that don't contribute significantly to the meaning of the dialogue. This helps the model focus on the important content words for generating relevant replies. In Addition, we also removed additional words that were used frequently in the series, but did not contribute to the final output(e.g., "know", "here", "thing").
- Lemmatization Evaluation: We experimented with lemmatization (converting words to their base form), but it did not significantly improve the results. Therefore, we opted to exclude this step for efficiency.

1.2 Data Insights

The character distribution within the Game of Thrones dialogue dataset presented an interesting challenge. While some prominent characters have a significant number of spoken lines, many others have a smaller speaking role. To optimize our training data and capture the nuances of character-specific dialogue, we implemented the following strategy:

- Top 25 Characters: We identified and extracted dialogue lines from the 25 characters with the most sentences in the dataset. This group likely represents the major players and recurring characters throughout the series. Focusing on them ensures a strong foundation for the model to learn distinct speech patterns and vocabulary associated with these key figures.
- Combined "Man" Character: The remaining characters, with a smaller number of lines each, were combined into a single entity we called "Man" While this approach creates a rich dataset with almost 10,000 lines, allowing the model to learn a broader range of vocabulary and sentence structures commonly used within the Game of Thrones universe, it also comes with a limitation. By merging the voices of various characters, we lose the opportunity to capture the unique speech patterns and personalities of minor characters who might play significant roles in specific storylines.

Beyond analyzing character distribution, we delved deeper into the dataset to uncover recurring phrasing and conversational patterns. Here's what we discovered:

- Frequent Sentences: Initially, we extracted the most frequent sentences spoken throughout the series. While this provided some insights into common dialogue elements, it wasn't sufficient to capture the full spectrum of character-specific language use.

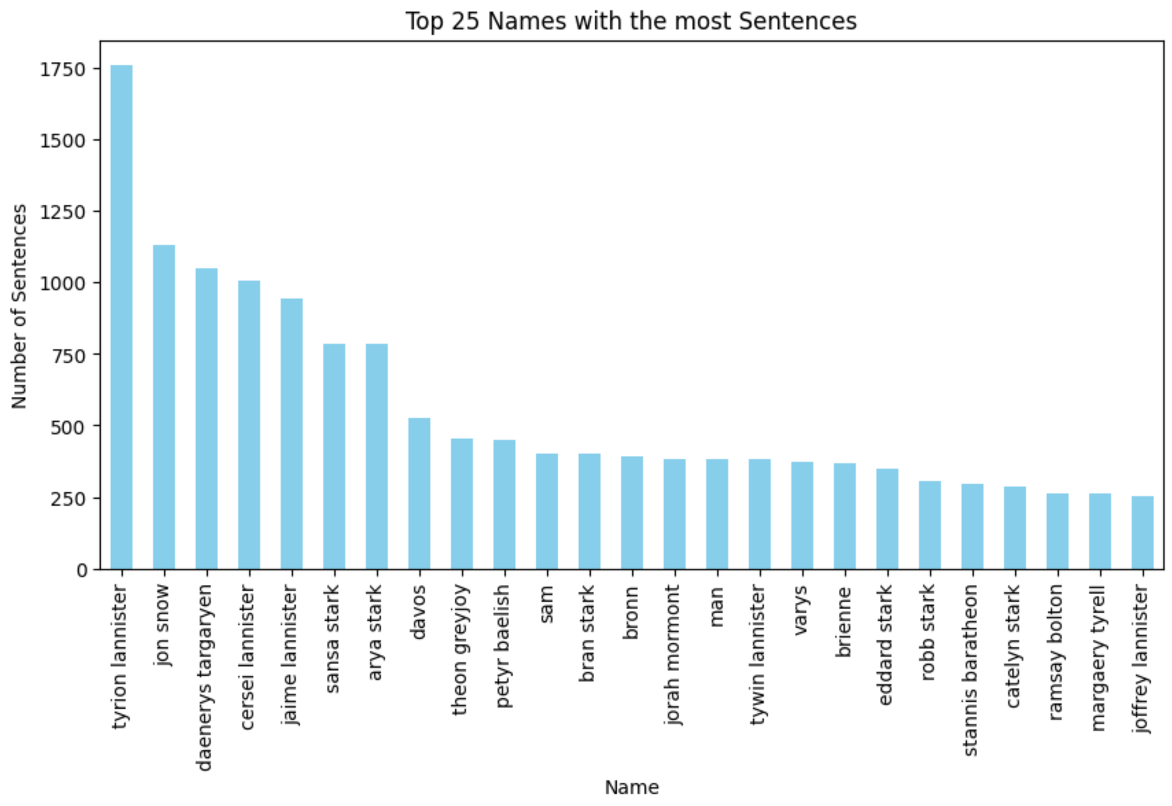

Figure 1: Top 25 Characters before adding combined character

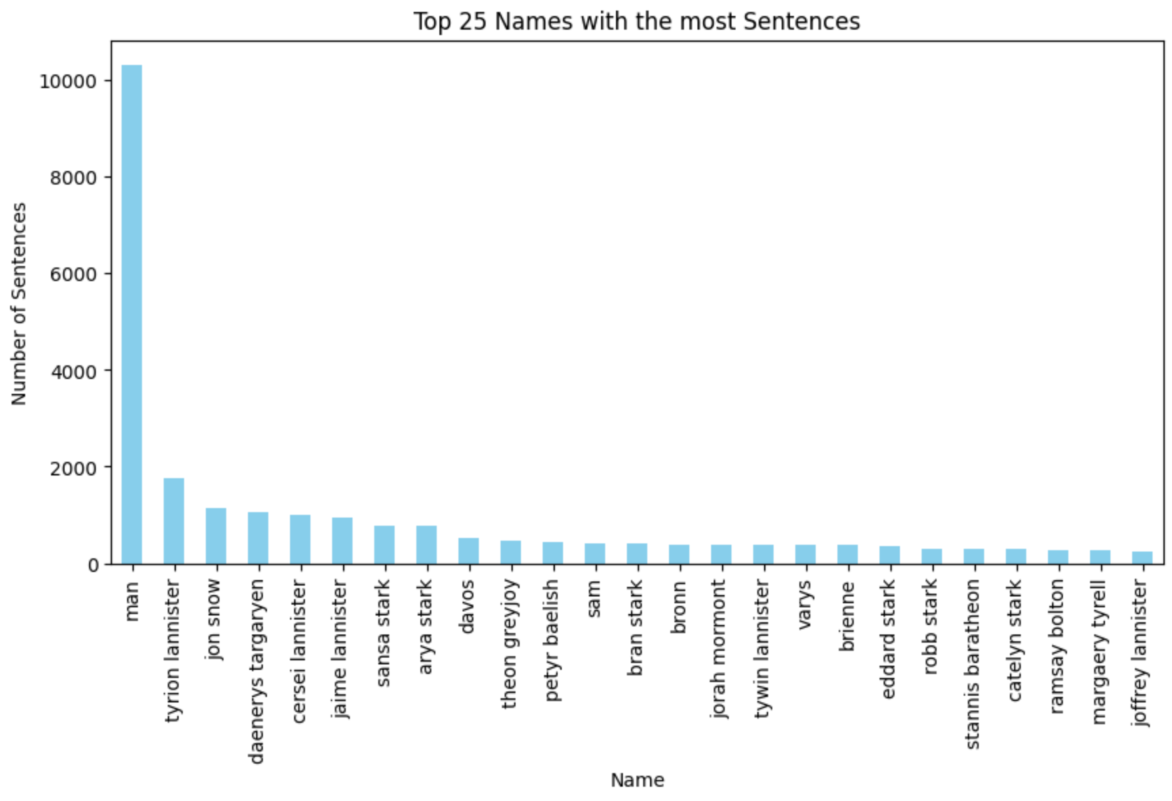

Figure 2: Top 25 characters after adding combined character

- N-Gram Analysis: To address this limitation, we explored n-gram analysis, a technique that identifies frequently occurring sequences of words. We extracted bigrams (two-word phrases), trigrams (three-word phrases), and even fourgrams and fivegrams (longer sequences). This approach proved fruitful, revealing several recurring phrases and character-specific dialogue patterns.

For instance, n-gram analysis might identify:

- Trigram: ”you know nothing” (often associated with Jon Snow)
- Fourgram: ”winter is coming” (a signature phrase of House Stark)
- Fivegram: ”a Lannister always pays his debts” (characteristic of House Lannister)

These recurring n-grams provide valuable insights into the language patterns and potentially reflect character motivations or personality traits.

This analysis of n-grams serves two purposes:

1. Identify Character-Specific Phrases: By highlighting frequently used n-grams for each character, we can inform the model about these recurring phrases, potentially enabling it to generate replies that are more consistent with a character’s established way of speaking.
2. Complement Word Frequency: While the model will learn the most frequent words used by each character, n-gram analysis provides context for how these words are often combined. This can improve the fluency and naturalness of the generated replies.

To further explore the unique language patterns within the dataset, we employed word cloud visualization for the top 25 characters. Word clouds visually represent word frequency, with larger words appearing more frequently in the text.

The initial word clouds revealed a significant presence of common words like ”know” ”here” and ”think” These words, while important for grammatical structure, don’t necessarily convey the unique voice of each character.

To address this, we performed a manual refinement process. We identified and removed some of the most frequent generic words that appeared across all characters. This filtering process allowed for a clearer visualization of the words that hold more distinction for each character.

The refined word clouds provided valuable insights:

- Character-Specific Vocabulary: Words like ”dragons” for Daenerys Targaryen, or ”Khaleesi” for Jorah Mormont, became more prominent.
- Bigram Identification: The word clouds even revealed frequently used bigrams (two-word phrases) that offered a glimpse into a character’s personality or relationships.

By using word clouds alongside n-gram analysis, we gained a deeper understanding of the vocabulary and phrasing choices that differentiate each character’s voice within the Game of Thrones dialogue. This comprehensive analysis will be instrumental in training our NLP model to not only generate grammatically sound replies but also capture the unique speech patterns and conversational styles of the characters.

1.3 Limitations

While the chosen dataset offers a wealth of dialogue for training our NLP model, it’s important to acknowledge some limitations and considerations for future work:

- Contextual Information: The dataset lacks information about dialogue context, such as whether a sentence is the beginning of a conversation or a response. This can make it challenging for the model to fully understand the flow and intent behind each utterance.
- Merged Characters: As mentioned earlier, the limited lines of many characters necessitated combining them into a single ”Man” character. While this approach provides a broader vocabulary base, it loses the unique voices of these minor characters.

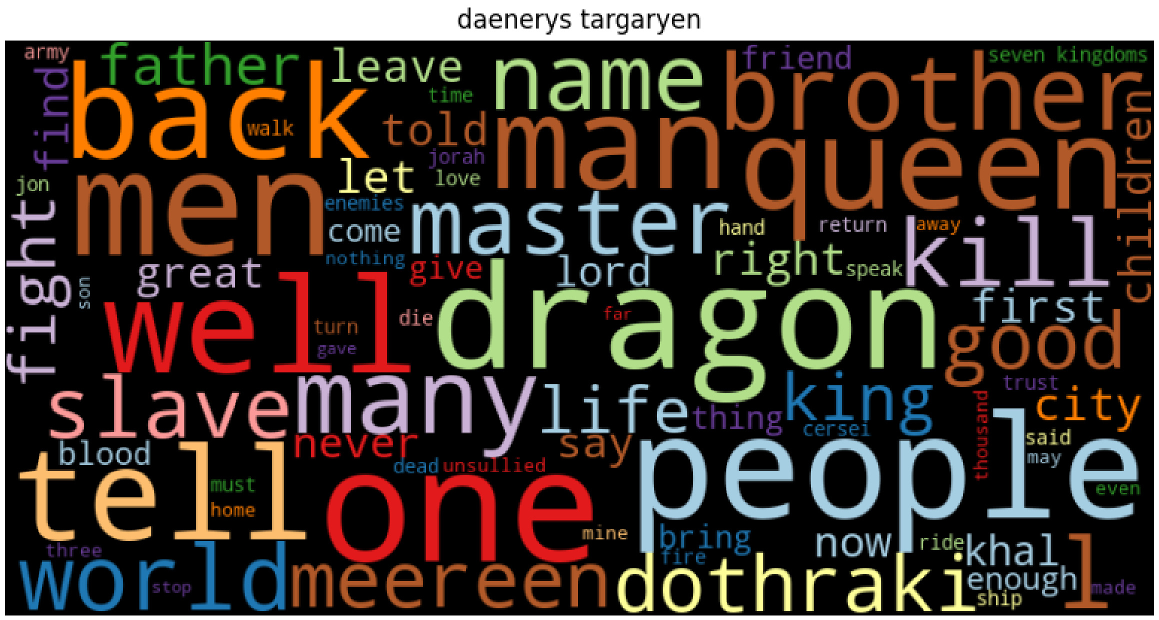

Figure 3: Daenerys Targaryen wordCloud

- Inappropriate Language: Some characters within the series use vulgar or offensive language. Depending on the intended use of the model, it might be necessary to filter out such language to ensure appropriate replies are generated.
- Character Development: A key aspect of Game of Thrones is the evolution of characters throughout the series. Their speech patterns and vocabulary might change as they experience events and grow. A single model trained on the entire dataset might not fully capture these nuances.
- Character Imbalance:The dataset exhibits an imbalance in the number of lines spoken by different characters. Characters with more lines will naturally be better represented in the model, potentially leading to more accurate reply generation for those characters compared to others with fewer lines.

2 Background

Chatbots, are AI programs designed to talk to people in a way that feels like a real conversation. They're inspired by all kinds of text, from books to social media posts. In this report, we'll show you the methodology behind the chatbot based on a famous TV show script. We'll focus on how seq2seq and LSTM make the chatbot sound more human-like.

Seq2seq is a cool technology in the world of AI that helps with tasks like translation and generating text. It works by breaking conversations into two parts: understanding what's said (the "encoder" part) and then crafting a reply (the "decoder" part). Our model uses seq2seq to figure out what you're saying and come up with a good response based on the TV show's style.

The encoder, the initial component of the seq2seq model, plays a pivotal role in processing the input sequence to extract its semantic meaning. It comprises recurrent neural network (RNN) layers, such as LSTM or GRU (Gated Recurrent Unit), which sequentially process each token of the input sequence. As the encoder iterates through the tokens, it continuously updates its hidden state, summarizing the information it has encountered thus far. This hidden state serves as a repository of contextual information, capturing the essence of the input sequence. Once the entire input sequence has been processed, the final hidden state of the encoder condenses this semantic understanding into a single representation known as the context vector.

On the other hand, the decoder is tasked with generating the output sequence based on the context vector provided by the encoder. Similar to the encoder, the decoder also consists of recurrent neural network layers, often employing the same architecture. At each time step of the decoding process, the decoder receives the context vector from the encoder alongside an input token, which could either be the previously generated token or a special start-of-sequence token. Leveraging this information, the decoder updates its hidden state, incorporating both the context vector and the input token to inform its decision-making process. This enables the decoder to generate the subsequent token in the output sequence, gradually unfolding the response. This iterative process continues until an end-of-sequence token is produced or a predefined maximum sequence length is reached, ultimately yielding the final output sequence.

This is the most suitable architecture for a multitude of reasons:

Handling Variable-Length Inputs and Outputs: In conversations, both questions and answers can vary in length. Seq2seq models, with their encoder-decoder architecture, can handle inputs and outputs of variable lengths. This flexibility is crucial for chatbots, as they need to accommodate diverse user queries and provide coherent responses. Capturing Context and Semantics: Seq2seq models excel at capturing the context and semantics of input sequences. The encoder processes the input sequence, extracting its underlying meaning and encoding it into a fixed-dimensional representation (the context vector). This context vector encapsulates the essence of the input, enabling the decoder to generate relevant and contextually appropriate responses. Generating Natural-Language Responses: By leveraging recurrent neural networks (RNNs) or similar architectures like LSTMs, seq2seq models can generate natural-language responses that mimic human speech patterns. The decoder network utilizes the context vector produced by the encoder to generate output sequences, crafting responses that are syntactically and semantically coherent.

Long Short-Term Memory (LSTM) networks, another smart tool we used in our chatbot and an evolution of traditional recurrent neural networks (RNNs), were introduced by Hochreiter and Schmidhuber in 1997 to overcome the limitations of RNNs in capturing long-range dependencies and mitigating the vanishing gradient problem. LSTMs have since become a cornerstone in a wide array of sequence modeling tasks, including natural language processing, speech recognition, and time series analysis.

At their core, LSTM networks are designed to retain information over extended periods, facilitated by a memory cell that carries information across time steps. Unlike traditional RNNs, LSTMs incorporate specialized gating mechanisms—the input gate, forget gate, and output gate—to regulate the flow of information within the network. Each gate plays a vital role in controlling the information flow:

The input gate decides which new information should be stored in the memory cell. The forget gate determines what information from the previous cell state should be discarded. The output gate controls which information should be passed to the output. These operations within the LSTM cell enable it to selectively retain and update information, allowing it to learn long-term dependencies effectively while addressing the vanishing gradient problem. Consequently, LSTM networks offer benefits such as improved gradient stability during training and the ability to capture intricate temporal patterns, making them indispensable in various sequential data processing tasks.

Using seq2seq and LSTM was a big step forward in making our chatbot feel like a real person from the TV show. These technologies let it understand what you're saying and help respond in a way that fits the show's atmosphere. As AI gets better, projects like ours show how tech can make human-computer interactions more fun and engaging. the following section highlights the model flow and architecture.

3 Neural Network Model

3.1 Further Data Preprocessing

For the data preprocessing phase, it's important to note that the original dataset wasn't neatly divided into individual dialogues. Consequently, each sentence in the script wasn't explicitly paired with its corresponding response. To mitigate this challenge, a pragmatic approach was adopted: each sentence was treated as a standalone question, and the subsequent sentence was considered its corresponding answer. For simplicity, the characters where disregarded and only the top 5 speaking characters where combined and treated as one speaker

While this method isn't the most accurate representation of natural dialogue flow, it served as a practical solution to handle the undivided script. By pairing consecutive sentences in this manner, we ensured that each question had an associated response, albeit potentially oversimplifying the intricacies of real conversations.

This approach allowed us to extract question-answer pairs from the script data, facilitating the subsequent stages of data preprocessing and model training. Despite its limitations, it enabled us to leverage the available dataset effectively and lay the groundwork for training a chatbot model on the provided dialogue corpus.

Both questions and answers are filtered based on their lengths. Only sequences with lengths less than or equal to 13 tokens are retained. This step helps in controlling the input sequence length for the model.

3.2 Vocabulary Construction

A vocabulary is constructed by counting the occurrences of words in both questions and answers. This process involves iterating through each word in the dialogue corpus and maintaining a count of its occurrences. Words that occur fewer than 5 times are discarded from the vocabulary to limit its size and remove rare or infrequent words. Additionally, special tokens such as PAD (padding), EOS (end of sequence), OUT (out of vocabulary), and SOS (start of sequence) are added to the vocabulary after ecapsulating each sentence between SOS EOS . These tokens serve specific purposes during model training and inference.

3.3 Encoder and Decoder

The cleaned questions and answers are tokenized. These tokens are then encoded into numerical representations using the constructed vocabulary. If a word is not present in the vocabulary (out-of-vocabulary), it is replaced with the OUT token to ensure that all words are mapped to valid numerical indices. Additionally, each answer sequence is augmented with a SOS token at the beginning to signify the start of decoding.

Encoder and decoder inputs are padded to a fixed length of 13 tokens using zero-padding. This ensures that all sequences have the same length, which is necessary for batch processing during model training.

Decoder output sequences are prepared by shifting the decoder input sequences one position to the left. This alignment ensures that the model learns to predict the next token in the sequence based on the preceding context.

The shifted decoder output sequences are then one-hot encoded to match the vocabulary size. Each token in the sequence is represented as a binary vector with a dimension equal to the size of the vocabulary, where the index corresponding to the token is set to 1 and all other indices are set to 0. The architecture and flow of the model go as follows: First, the encoder converts input words into dense vector representations through an embedding layer, which helps the model better understand and handle the words. These vectors are then fed into an LSTM layer with 400 units, which processes the word sequence step-by-step. The LSTM outputs both the sequence of processed words and the final hidden and cell states (h and c), which summarize all the information from the input sequence. For the decoder, the same embedding layer is used to convert the input words into dense vectors. These vectors are passed through another LSTM layer, also with 400 units, but this LSTM is initialized with the hidden and cell states from the encoder. This initialization provides the decoder with the context from the encoder, enabling it to generate an output sequence relevant to the input sequence. By linking the encoder and decoder in this manner, the model can learn to transform input sequences into appropriate output sequences.

3.4 Inference

When the user provides an input, it gets preprocessed to match the format used during training. This involves converting it to lowercase, removing punctuation, and expanding contractions. After cleaning, the text is tokenized into a sequence of integers using the vocabulary dictionary created during training. This step converts each word in the input to its corresponding integer representation based on the vocabulary.

The tokenized input sequence is passed through the encoder model to obtain the internal states (hidden and cell states) of the LSTM. This step involves feeding the tokenized input into the encoder to generate a context vector that summarizes the input sequence. The encoder outputs hidden states that encapsulate the meaning of the input, which will be used by the decoder to generate the response.

The decoding process starts with an initial input token SOS to signal the start of the sequence. The decoder then generates words one by one until it produces an EOS token or reaches a maximum sequence length. Initially, an empty target sequence is prepared with the SOS token. This token serves as the starting point for the decoder.

The decoder begins generating the response by using the start of a sentence token. It then predicts the next word by analyzing the initial input along with the context provided by the encoder. Out of all possible words, the decoder selects the one with the highest probability. This chosen word is appended to the response. The decoder then uses this newly predicted word as the input for predicting the following word. This word-by-word prediction continues, with the decoder repeatedly using the latest word and the context from the encoder to generate the next word in the sequence. The process continues until the decoder predicts the end of the sentence token or reaches the predetermined maximum length for the response.

3.5 Model training

The training process of the model spanned over 150 epochs, demonstrating a clear trajectory of improvement. Initially, in Epoch 1, the model had a loss of 4.0701 with an accuracy of 35.53%. As training progressed, a notable decrease in loss and a consistent increase in accuracy were observed. By Epoch 25, the loss had reduced to 2.1900 with accuracy climbing to 52.89%. This positive trend continued, and by Epoch 50, the model achieved a loss of 1.3473 and an accuracy of 69.74. Significant improvement was seen in the later stages, with Epoch 75 recording a loss of 0.8451 and an accuracy of 82.06%. Nearing the end of training, in Epoch 100, the loss further dropped to 0.3992 with an accuracy of 91.52%. The final epochs showed fine-tuning adjustments with slight changes in loss and accuracy, and by Epoch 145, the model attained an impressive accuracy of 98.96%, signifying robust performance and successful convergence.

the following are example queries to the chatbot.

- - **Query:**you are a dwarf
  - **answer:**i am a monster as well as a dwarf you should
- - **Query:**for all the dead may never die
  - **answer:**they are right take him alive kill his men
- - **Query:**what are we going to do
  - **answer:**someone does work that poor years to see the city OUT
- - **Query:**are you a lannister?
  - **answer:**no i do not think you care about your teeth if

4 Fine-tuning a pretrained model

4.1 Data Preprocessing

To fine-tune the model effectively, the dataset required adjustments to align with the input format required by the fine-tuned model. The dataset comprised three columns: question, answer, and speaker. The top 10 speakers were designated as individual classes, while the remaining speakers were consolidated into a single class labeled "man," considering that their lines were relatively sparse. It's noteworthy that the answers in the dataset were attributed to specific speakers, indicating that each answer was provided by the speaker. Then the dataset is converted to Dataset object. This structure facilitated the treatment of speakers as distinct classes, enabling the model to learn speaker-specific response patterns. Therefore, the dataset underwent modifications to conform to this format, ensuring that the model could effectively capture the nuances of individual speakers' responses during training. By organizing the data in this manner, the fine-tuned model could learn to generate contextually appropriate responses based on both the input question and the identity of the speaker, enhancing its ability to mimic human-like conversational interactions.

4.2 Methodology and Training

After converting the dataset into a Dataset object, the dataset is divided into training and testing subsets using a train-test split with a specified test size. This division ensures that the model's performance can be reliably evaluated on unseen data, thereby providing insights into its generalization capabilities.

Throughout this pipeline, numerical considerations such as training epochs, batch sizes, and evaluation metrics play a pivotal role in shaping the model's trajectory. The Seq2SeqTrainer object orchestrates the training process, utilizing the tokenized dataset and specified training arguments to iteratively update the model weights. Evaluation metrics are computed to assess the model's performance on the test dataset,including the Rogue score, providing valuable insights into its efficacy in generating accurate responses.

Unfortunately, because of a technical error, only the validation and training information of the first 2 epochs are available. Nonetheless, we can still analyze their meanings.

Table 1: Training and Validation Metrics

Epoch	Training Loss	Validation Loss	Rouge1	Rouge2	RougeL	RougeLSum
0	3.081	2.833	0.080	0.015	0.077	0.078
1	2.771	2.778	0.090	0.016	0.085	0.087

The table presents training and validation metrics for the fine-tuning process across two epochs. The training and validation loss metrics, representing prediction errors, decrease from epoch 0 to epoch 1, indicating an improvement in model performance over epochs. Additionally, Rouge scores, evaluating the quality of generated responses compared to reference answers, show slight improvements from epoch 0 to epoch 1. Rouge1 and Rouge2 measure word and phrase overlap, RougeL evaluates semantic similarity, and RougeLSum combines both metrics. The observed improvements suggest that the fine-tuned model is gradually learning to generate more contextually relevant responses. However, further adjustments or fine-tuning iterations may be necessary to achieve significant performance gains. Overall, the decreasing trend in loss metrics and the slight improvement in Rouge scores indicate progress in enhancing the model's question-answering capabilities, highlighting the iterative nature of model refinement in natural language processing tasks.

After fine-tuning, the model is saved, and inference can be performed on new questions. A function leverages the fine-tuned model to generate contextually relevant responses based on input questions and speaker identities, thereby demonstrating the practical utility of the fine-tuned FLAN-T5 model in real-world conversational scenarios.

the following are example queries to the fine-tuned model.

- - **Query:** valar moghulis
  - **Speaker:** arya stark
  - **answer:** Valar dohaeris.
- - **Query:** valar moghulis
  - **Speaker:** daenerys targaryen
  - **answer:** Is that a good man?
- - **Query:** you are guilty
  - **Speaker:** tyrion lannister
  - **answer:** I am guilty of treason.
- - **Query:** bend the knee
  - **Speaker:** jon snow

  - **answer:** I'm sorry, my lord.

- - **Query:** who injured you?
  - **Speaker:** tyrion lannister
  - **answer:** I was a drunken fool.

- - **Query:** what happened to you?
  - **Speaker:** sansa stark
  - **answer:** I was stabbed in the back.

- - **Query:** the white walkers, what about them??
  - **Speaker:** jon snow
  - **answer:** They're not going to die.

more examples are available in the notebook.

5 Conclusion

The choice of the FLAN-T5 model for fine-tuning was deliberate, driven by its pre-training on conversational data. This pre-training aligns seamlessly with the nature of the question-answering task involving diverse speakers. Unlike traditional encoder-decoder models, FLAN-T5 incorporates conversational context more effectively, enabling it to generate responses that are contextually relevant within dialogue systems.

In conclusion, leveraging the FLAN-T5 model offers distinct advantages over traditional encoder-decoder architectures for conversational tasks. Fine-tuning FLAN-T5 on speaker-specific question-answer pairs enables the model to capture subtle speaker behaviors and produce responses that are more contextually appropriate. This contrasts starkly with encoder-decoder models, which may struggle to grasp conversational nuances without explicit training on dialogue data. Thus, the selection of FLAN-T5 for fine-tuning in this task harnesses its conversational capabilities to enhance question-answering performance within a speaker-specific context.

However, it's important to recognize potential limitations associated with the FLAN-T5 model. Fine-tuning large-scale models like FLAN-T5 can be computationally intensive and demand significant data annotation efforts. Moreover, while FLAN-T5 excels in capturing conversational nuances, it may encounter challenges in handling ambiguity or generating diverse responses, especially in complex or multi-turn interactions. Additionally, like other large-scale language models, FLAN-T5 may exhibit biases inherent in its training data, potentially resulting in biased or inappropriate responses. Addressing these limitations requires ongoing research efforts, including data augmentation techniques and bias mitigation strategies.

Furthermore, it's crucial to acknowledge the constraint posed by the dataset's size, comprising approximately 23,900 lines. While FLAN-T5 demonstrates impressive capabilities even with relatively modest training data, expanding the dataset size could further enhance the model's performance and generalization across various speakers and conversational styles. Augmenting the dataset with additional annotations, such as speaker intents or emotions, could also enrich the model's understanding of human communication dynamics. Therefore, future efforts should focus on scaling up the dataset size and enriching its content to unlock the full potential of FLAN-T5 in real-world conversational applications.

References

[RCR24] Nitin Rane, Saurabh Choudhary, and Jayesh Rane. Gemini versus chatgpt: Applications, performance, architecture, capabilities, and implementation. *SSRN Electronic Journal*, 2024.

[SST23] Vincenzo Scotti, Licia Sbattella, and Roberto Tedesco. A primer on seq2seq models for generative chatbots. *ACM Computing Surveys*, 56(3):1–58, Oct 2023.