**Chatbot using Cornell Movie Corpus Dataset**

Abdul Shariq, Gary Takahashi, and Matthew Guzman

Neural Networks & Deep Learning Applied Artificial Intelligence-520

University of San Diego

**Abstract**

Chatbots have gained significant popularity in recent years due to their ability to automate customer support, provide information, and engage users in natural language conversations. This report outlines the process of building a chatbot using a Transformer-based architecture. The chatbot is trained to understand and generate human-like responses to user queries. We discuss the key components of the chatbot, including data preprocessing, model architecture, and training process. The report also highlights various techniques used to enhance the chatbot's performance.

1. **Introduction**

Chatbots are computer programs designed to interact with users through natural language conversations. They have a wide range of applications, including customer support, virtual assistants, and information retrieval. Building an effective chatbot requires a deep understanding of natural language processing (NLP) techniques and the selection of appropriate machine learning models.

In this report, we describe the process of building a chatbot using a Transformer-based architecture, which has shown remarkable success in various NLP tasks. The chatbot is trained to understand user queries and generate contextually relevant responses. We discuss the data preprocessing steps, model architecture, and training process used to develop the chatbot. Additionally, we explore techniques to improve the chatbot's accuracy and user experience.

1. **System Design**

The process of building the chatbot can be summarized into the following key steps:

* Data Collection: Collect a dataset containing conversational data, including user queries and corresponding responses. In our case, we used the Cornell Movie Corpus, a dataset of movie dialogue.
* Data Preprocessing: Clean and preprocess the dataset by tokenizing text, adding special tokens, and padding sequences. This step ensures that the data is in a suitable format for training.
* Model Architecture: Implement a Transformer-based architecture for the chatbot. The architecture consists of an encoder-decoder structure with multi-head self-attention mechanisms.
* Training: Train the chatbot model using the preprocessed dataset. Define loss functions, optimizers, and evaluation metrics. The model learns to generate responses based on input queries.
* Hyperparameter Tuning: Experiment with various hyperparameters such as model dimensions, the number of layers, and dropout rates to optimize model performance.
* Scheduled Sampling: Implement scheduled sampling during training to improve the model's response generation by gradually transitioning from ground-truth tokens to model-generated tokens.
* Validation and Testing: Evaluate the model's performance using validation data. Test the chatbot with user queries to assess its ability to generate coherent and contextually relevant responses.
* Optimization: Fine-tune the model and training process based on validation results. Adjust learning rates, batch sizes, and training epochs as needed.
* Enhancements: Implement techniques such as beam search during inference to improve response quality.

1. **Data Analysis and Understanding**

Cornell – Movie – Dialog Corpus, Data set is about the factionary conversation recorded from the movie. This corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts: 220,579 conversational exchanges between 10,292 pairs of movie characters involves 9,035 characters from 617 movies in total 304,713 utterances.

There are multiple files in the dataset and the ones that is important for us are movie\_lines.txt which contains 304713 lines of conversation and movie conversations which contains the sequence of 83097 conversations. Eventually the conversation will help build the conversation pair from the movie lines files, which will be used for the model training at a later stage.

This data set falls under the question answer kind of chatbot category.

1. **Code used in building the chatbot.**

The code used in building the chatbot involves several Python libraries and frameworks, including TensorFlow and TensorFlow Datasets. Here is a high-level overview of the code:

* Data Loading: The Cornell Movie Corpus dataset is loaded and processed to extract conversational data.
* Tokenization and Padding: Text data is tokenized, and special tokens are added to indicate the start and end of sentences. Sequences are padded to a fixed length for training.
* Model Architecture: A Transformer-based model architecture is defined, consisting of an encoder-decoder structure with multi-head attention mechanisms.
* Training Configuration: The model is compiled with an appropriate loss function, optimizer, and evaluation metric. The chatbot is trained on the preprocessed dataset.
* Hyperparameter Tuning: Hyperparameters such as model dimensions, dropout rates, and learning rates are configured and fine-tuned.
* Scheduled Sampling: Scheduled sampling is implemented during training to improve response generation.
* Validation and Testing: The model is evaluated on validation data, and its performance is assessed through user interactions.
* Optimization: Based on validation results, the model and training process are optimized to achieve better performance.
* Enhancements: Techniques like beam search can be implemented during inference to enhance response quality.

1. **Model Training, and Results**

With data being pretty compute heavy, the 1st model run presented a result where the accuracy is just 17% which is very low. Here the layers used are 2, sample size 70k and 50 epochs were used. The model was trained for 18 hours without GPU.

For the 2nd training, we took leap of faith, paid for lots of google collab compute units and trained the model with 300k samples, 4 layers , and 300 epochs. The results came back possibly the worst but gave us a teaching which is explained in the training 3 and 4.

For the 3rd training , we purchased 100 compute Units separately, where we trained the model with 100k sample 2 layers and 150 epochs , the results were possibly the best , this explains that the previous model training is not bad but due to increased sample size and model complexity training is taking long and even though we had trained till 300 epochs , the model result is in still early phase, possibly training for 1000 / 2000 epochs could have produced better results.

In the 4th training, we tried playing around with the batch size and changed it to 128 from 64 and the results came back better.

In general, we did not achieve perfect accuracy that a real-life implementation would expect but achieved a pattern which gives us positive result, and opportunity to tune it further for to drive better result.

Epoch 47/50

960/960 [==============================] - 1219s 1s/step - loss: 0.6912 - accuracy: 0.1677

Epoch 48/50

960/960 [==============================] - 1215s 1s/step - loss: 0.6881 - accuracy: 0.1685

Epoch 49/50

960/960 [==============================] - 1222s 1s/step - loss: 0.6845 - accuracy: 0.1690

Epoch 50/50

960/960 [==============================] - 1232s 1s/step - loss: 0.6810 - accuracy: 0.1698

Model training 1: with 70k sample size, 2 layers, and 50 epochs

Epoch 298/300

3038/3038 [==============================] - 114s 37ms/step - loss: 0.9969 - accuracy: 0.1274

Epoch 299/300

3038/3038 [==============================] - 114s 37ms/step - loss: 0.9965 - accuracy: 0.1275

Epoch 300/300

3038/3038 [==============================] - 113s 37ms/step - loss: 0.9965 - accuracy: 0.1276

Model training 2: with 300k sample size, 4 layer, and 300 epochs

Epoch 149/150

1379/1379 [==============================] - ETA: 0s - loss: 0.2752 - accuracy: 0.2481

Epoch 149: accuracy improved from 0.24799 to 0.24813, saving model to checkpoints\_best\_only

1379/1379 [==============================] - 530s 384ms/step - loss: 0.2752 - accuracy: 0.2481

Epoch 150/150

1379/1379 [==============================] - ETA: 0s - loss: 0.2749 - accuracy: 0.2482

Epoch 150: accuracy improved from 0.24813 to 0.24819, saving model to checkpoints\_best\_only

1379/1379 [==============================] - 538s 390ms/step - loss: 0.2749 - accuracy: 0.2482

Model training 3: with 100k sample size, 2 layers and 150 epochs

689/690 [============================>.] - ETA: 0s - loss: 0.5258 - accuracy: 0.2007

Epoch 149: accuracy improved from 0.20024 to 0.20069, saving model to checkpoints\_best\_only

690/690 [==============================] - 31s 45ms/step - loss: 0.5258 - accuracy: 0.2007

Epoch 150/150

689/690 [============================>.] - ETA: 0s - loss: 0.5246 - accuracy: 0.2008

Epoch 150: accuracy improved from 0.20069 to 0.20074, saving model to checkpoints\_best\_only

690/690 [==============================] - 31s 46ms/step - loss: 0.5247 - accuracy: 0.2007

Model training 4 : with 100k sample size , 4 layers, 150 epochs, and batch size 128 instead of 64

The biggest challenge that we faced was with availability of GPU and the compute required to train the model, any alternative option that we tried or tried tuning the model, it took hours or day to get the result and then the process continued.

* 1. **Multi head attention**

MultiHead Attention is a crucial component in the architecture of the Transformer model, a deep learning model designed for various sequence-to-sequence tasks, including machine translation and chatbot development. MultiHead Attention allows the model to focus on different parts of the input sequence simultaneously, enabling it to capture complex relationships and dependencies within the data.

In MultiHead Attention, the input sequence is transformed into multiple representations, or "heads," each with its own set of learnable parameters. These heads compute attention scores between the input elements (usually tokens or words) differently, allowing the model to attend to various parts of the input for different tasks.

The key steps in MultiHead Attention include:

* Computing separate query, key, and value matrices for each head.
* Calculating attention scores by taking the dot product between the query and key matrices.
* Scaling the attention scores by the square root of the dimension of the key vectors.
* Applying a mask to the attention scores (e.g., to mask out padding tokens).
* Applying the softmax function to obtain attention weights.
* Weighting the value vectors by the attention weights for each head.
* Concatenating the outputs from all heads and passing them through a linear layer.

The multi-headed approach enables the Transformer model to capture different types of information from the input sequence simultaneously, improving its ability to model context and relationships.

* 1. **Transformer Model Creation**

The Transformer model, introduced in the paper "Attention Is All You Need" by Vaswani et al., revolutionized natural language processing tasks. It consists of an encoder-decoder architecture, with each part containing multiple layers of MultiHead Attention and Feed-Forward Neural Networks.

Here's an overview of the steps to create a Transformer model:

* Input Embedding: Tokenize input sequences into numerical tokens and embed them into continuous vector representations. Additionally, apply positional encoding to account for word order.
* Encoder Stack: The encoder stack consists of multiple layers of encoder blocks. Each encoder block typically includes:
  + MultiHead Attention Layer: To capture dependencies within the input sequence.
  + Feed-Forward Neural Network: To process the attention outputs.
  + Residual Connections and Layer Normalization: To stabilize and facilitate information flow.
* Decoder Stack: The decoder stack is similar to the encoder stack but includes an additional MultiHead Attention layer that attends to the encoder's output. This layer helps the decoder generate contextually relevant responses.
* Output Layer: The decoder's output is passed through a linear layer followed by a softmax function to generate probability distributions over the vocabulary. The token with the highest probability is selected as the predicted next word.
* Training: Train the Transformer model using a suitable loss function (e.g., Sparse Categorical Cross-Entropy) and an optimizer (e.g., Adam) with a custom learning rate schedule.
* Inference: During inference, use the trained model to generate responses by repeatedly predicting the next token until an end token is generated or a maximum sequence length is reached.

The Transformer model's remarkable ability to model long-range dependencies and its parallelizability have made it a popular choice for various natural language processing tasks, including chatbot development.

Incorporating MultiHead Attention and building a Transformer model requires careful implementation of the described components and extensive training on relevant data. The resulting model can be fine-tuned and improved over time to provide more contextually accurate responses in chatbot interactions.

1. **Challenges faced during the implementation of this project**

* Complex Architecture and coding in building transformer-based model:   
  Developing a transformer-based chatbot for the Cornell Movie corpus dataset presents challenges due to the complex architecture and coding required to build and fine-tune the model effectively. Special mention about coding encoder decoder codes.
* Lot of compute requirements and costly to buy cloud-based compute: Training such models demands significant computational resources, making it costly when relying on cloud-based computing solutions for extended periods.
* Not enough documentation on pretrained models: the limited availability of comprehensive documentation on pretrained transformer models tailored specifically for chatbot applications made it difficult to use it for training.
* Data cleaning: Data cleaning is a crucial, time-consuming step, as the dataset contained noisy or inconsistent dialogue that can negatively affect the model's performance. And lot of effort was required to identify those discrepancies and address them.
* Transformation of conversation into question answer-based dataset: Converting the Cornell Movie corpus data, originally in a conversation format, into a question-answer-based dataset suitable for chatbot training requires careful preprocessing and transformation to capture meaningful interactions.

1. **Real-life implementations**

The development and deployment of a chatbot in real-life scenarios have gained significant traction across various industries, offering innovative solutions to improve customer service, streamline operations, and enhance user experiences. In this section, we discuss the practical aspects of implementing the chatbot, including potential use cases and considerations.

Use Cases:

* Customer Support: Chatbots are widely used in customer support applications to provide quick responses to frequently asked questions and resolve common issues. In e-commerce, for example, chatbots can assist customers with order tracking, product recommendations, and returns processing.
* Healthcare: Chatbots have found applications in the healthcare sector, where they can help patients schedule appointments, access medical information, and even monitor chronic conditions. They can also serve as a triage tool, assisting medical professionals in assessing patient symptoms.
* Financial Services: Banks and financial institutions employ chatbots to assist customers with account inquiries, transaction history, and even basic financial advice. Chatbots can help users check account balances, transfer funds, and set up automated savings plans.
* E-Learning: Educational institutions and e-learning platforms use chatbots to provide personalized learning experiences. Chatbots can answer student queries, recommend study materials, and deliver quizzes or exercises.
* Human Resources: Chatbots can streamline HR processes by answering employee queries about benefits, policies, and vacation requests. They can also assist in the recruitment process by screening applicants and scheduling interviews.

**Deployment Considerations:**

* Data Privacy and Security: Implementing a chatbot in real life requires careful consideration of data privacy and security regulations, especially in industries like healthcare and finance. Ensure that user data is handled in compliance with applicable laws and regulations.
* Scalability: Plan for scalability to handle increased user load. As the chatbot gains popularity, it should be able to accommodate a growing number of users without performance degradation.
* User Experience: Continuously gather user feedback and refine the chatbot's responses to improve user satisfaction. Implementing a natural language understanding (NLU) component can enhance the chatbot's ability to understand user intents accurately.
* Integration: Integrate the chatbot with existing systems and databases to access relevant information. APIs and webhooks can facilitate seamless data exchange between the chatbot and other applications.
* Maintenance and Monitoring: Regularly update the chatbot's knowledge base and responses to keep it up-to-date with changing information and user needs. Implement monitoring and alerting systems to detect and address issues promptly.
* User Training: Train chatbot users and customer service representatives on how to effectively interact with the chatbot. Provide guidelines for escalating complex issues to human agents when necessary.

**Challenges:**

While chatbots offer numerous advantages, they also face challenges such as:

* Handling Complex Queries: Chatbots may struggle with complex or ambiguous user queries.
* Maintaining Context: Maintaining context over extended conversations can be challenging.
* Ensuring Accuracy: Ensuring the accuracy of responses, especially in critical domains, is essential.
* User Acceptance: Users may initially resist interacting with chatbots, preferring human agents.
* Integration Complexity: Integrating with existing systems can be complex and time-consuming.

**Benefits:**

* 24/7 Availability: Chatbots can provide round-the-clock support, enhancing user convenience.
* Cost Savings: By automating routine tasks, chatbots can reduce operational costs.
* Improved Efficiency: Chatbots can handle multiple conversations simultaneously, reducing response times.
* Enhanced User Engagement: Chatbots can engage users in interactive and personalized conversations.
* Data Insights: Chatbot interactions generate valuable data for analytics and decision-making.

Challenges:Overall, the real-life implementation of this project can bring a valuable tool to the world of music and enrich the experience of appreciating and understanding different composers' works.

**Conclusion**:

The real-life implementation of a chatbot represents a significant step forward in providing efficient and personalized services to users across various domains. As technology continues to advance, chatbots are expected to play an increasingly prominent role in shaping the future of customer interactions, education, healthcare, and more.

Successful implementation requires careful planning, continuous improvement, and a commitment to delivering value to users and organizations alike. With the right strategy and considerations in place, chatbots have the potential to transform how businesses and institutions engage with their audiences.

References