



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

FINAL COURSE PROJECT REPORT

DEEP LEARNING

Designing a Chatbot using Deep Neural Networks

For Natural Language Understanding

Final Course Project Report

Course: Deep Learning

Instructor: Prof. Angshuman Paul

Team Members

Yagnesh Dave (B21AI046)

Ashish Cherukuri (B21CS019)

Adarsh Kumar (B21BB003)

Varun Purwar (B21AI043)

April 12, 2024

1 Introduction

1.1 Problem Statement

Designing a chatbot using deep neural networks for natural language understanding. The specific problem addressed in this project is the development of a chatbot capable of understanding and generating human-like responses in conversational contexts. This involves processing unstructured text input, identifying the intent and context of user queries, and generating appropriate responses that are contextually relevant and grammatically correct.

1.2 Solution Strategy

The solution strategy used for addressing this problem involves the utilization of deep learning techniques, especially sequence to sequence type models. RNNs are generally well-suited for sequential data processing tasks like natural language understanding and generation due to their ability to capture temporal dependencies.

1.3 Motivation

We know that Sequence to sequence Models like RNNs, LSTMs are among the best suited neural networks for processing sequential data. That's the motivation behind selecting them as model architecture in our project. Their capability to effectively model long-range dependencies in sequential data is crucial for understanding and generating coherent responses in conversational settings. Additionally, LSTM networks mitigate the vanishing gradient problem commonly encountered in traditional RNNs, enabling more stable and efficient training.

2 Dataset

2.1 Dataset Description

The Cornell Movie Dialogs Corpus dataset has been selected as the primary source of training data for the chatbot model. This dataset consists of a large collection of movie dialogues extracted from a diverse range of movie scripts, encompassing various genres and styles of conversation. The dataset is publicly available and widely used for natural language processing research tasks, making it an ideal choice for training conversational models like chatbots.

2.2 Dataset Selection Motivation

2.2.1 Rich Conversational Content

The Cornell Movie Dialogs Corpus offers a rich and diverse collection of conversational content, spanning different genres, tones, and linguistic styles. This diversity enables the chatbot model to learn from a wide range of conversational patterns and linguistic nuances, enhancing its ability to engage in natural and contextually relevant interactions with users.

2.2.2 Realistic Dialogue Scenarios

The dialogues in the dataset are extracted from movie scripts, reflecting realistic dialogue scenarios encountered in everyday conversations. By training on such authentic and contextually rich data, the chatbot model can learn to generate responses that are not only grammatically correct but also contextually appropriate, mimicking human-like conversational behavior.

3 Methodology

3.1 Data Preprocessing

Advanced data preprocessing techniques were employed to prepare the dataset for training. These techniques included stemming and lemmatization, which helped standardize the vocabulary and reduce lexical variations, thereby improving the model's ability to understand and generalize from the text data.

```
1 def unicodeToAscii(s):
2     return ''.join(c for c in unicodedata.normalize('NFD', s)
3                     if unicodedata.category(c) != 'Mn')
4
5 def normalizeString(s):
6     s = unicodeToAscii(s.lower().strip())
7     s = re.sub(r"([.!?])", r" \1", s)
8     s = re.sub(r"[^a-zA-Z-?.!]+", r" ", s)
9     s = re.sub(r"\s+", r" ", s).strip()
10    return s
```

Listing 1: Data Preprocessing Functions

3.2 Word Embeddings

Instead of traditional one-hot encoding, word embeddings were employed to represent words in a dense vector space. This allowed the model to capture semantic relationships between words and better understand the context of the conversation, leading to more meaningful responses.

```
1 embedding = nn.Embedding(voc.num_words, hidden_size)
```

Listing 2: Word Embedding Initialization

3.3 Model Architecture

3.3.1 Encoder-Decoder Architecture

The chatbot model was designed using an encoder-decoder architecture, which is a popular choice for sequence-to-sequence tasks like natural language generation. The encoder processes the input sequence and generates a context vector, while the decoder takes this context vector and generates the output sequence.

```
1 class EncoderRNN(nn.Module):
2     def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
3         super(EncoderRNN, self).__init__()
4         self.n_layers = n_layers
5         self.hidden_size = hidden_size
6         self.embedding = embedding
7         self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
8                             dropout=(0 if n_layers == 1 else dropout),
9                             bidirectional=True)
10
11     def forward(self, input_seq, input_lengths, hidden=None):
12         # ...
```

Listing 3: Encoder RNN

```
1 class LuongAttnDecoderRNN(nn.Module):
2     def __init__(self, attn_model, embedding, hidden_size, output_size,
3                     n_layers=1, dropout=0.1):
4         super(LuongAttnDecoderRNN, self).__init__()
5         self.attn_model = attn_model
6         self.hidden_size = hidden_size
7         self.output_size = output_size
8         self.n_layers = n_layers
9         self.dropout = dropout
10        # ...
11
12    def forward(self, input_step, last_hidden, encoder_outputs):
13        # ...
```

Listing 4: Decoder RNN with Luong Attention

3.3.2 Attention Mechanism

To improve the model's ability to focus on relevant parts of the input sequence, an attention mechanism was incorporated into the decoder. The Luong attention mechanism was used, which calculates attention weights based on the current decoder state and the encoder outputs.

```

1 class Attn(torch.nn.Module):
2     def __init__(self, method, hidden_size):
3         super(Attn, self).__init__()
4         self.method = method
5         self.hidden_size = hidden_size
6         # ...
7
8     def forward(self, hidden, encoder_outputs):
9         # ...

```

Listing 5: Luong Attention Layer

3.4 Training and Optimization

The chatbot model was trained using gradient descent optimization techniques. The Adam optimizer was used with a cross-entropy loss function. Techniques such as gradient clipping and teacher forcing were employed to improve training stability and convergence.

```

1 def train(input_variable, lengths, target_variable, mask, max_target_len,
2           encoder, decoder, embedding, encoder_optimizer, decoder_optimizer,
3           batch_size, clip, max_length=MAX_LENGTH):
4     # ...

```

Listing 6: Training Function

```

1 def trainIters(model_name, voc, pairs, encoder, decoder, encoder_optimizer,
2               decoder_optimizer, embedding, encoder_n_layers, decoder_n_layers,
3               save_dir,
4               n_iteration, batch_size, print_every, save_every, clip,
5               corpus_name, loadFilename):
6     # ...

```

Listing 7: Training Loop

4 Major Innovations

This project incorporates several innovative techniques and approaches to enhance the performance and capabilities of the chatbot model.

4.1 Advanced Data Preprocessing Techniques

Instead of solely relying on basic text cleaning techniques, advanced preprocessing methods such as stemming and lemmatization were applied to the dataset. These techniques helped standardize the vocabulary and reduce lexical variations, thereby improving the model's ability to understand and generalize from the text data.

Stemming is the process of reducing words to their base or root form by removing affixes (prefixes and suffixes). On the other hand, lemmatization involves determining the base form of a word based on its context and part of speech. Both techniques facilitate better understanding and representation of the text data.

Let w be a word in the vocabulary, and $stem(w)$ and $lemma(w)$ represent the stemming and lemmatization functions, respectively. The preprocessed word w' can be obtained as follows:

$$w' = \{ \text{stem}(w), \text{ifusingstemminglemma}(w), \text{ifusinglemmatization} \} \quad (1)$$

4.2 Word Embeddings for Representation

Instead of traditional one-hot encoding, which can lead to sparse and high-dimensional representations, word embeddings were employed to represent words in a dense vector space. This allowed the model to capture semantic relationships between words and better understand the context of the conversation, leading to more meaningful responses.

Let V be the vocabulary, and d be the embedding dimension. The word embedding matrix $E \in R^{|V| \times d}$ maps each word $w \in V$ to a dense vector representation $\vec{e}_w \in R^d$. The embedding matrix E is learned during the training process, enabling the model to capture semantic similarities between words based on their contexts.

4.3 LSTM Architectures for Improved Performance

Hybrid LSTM architectures combining both encoder-decoder LSTM layers were explored to leverage the strengths of each component. This hybrid approach resulted in improved performance in capturing long-range dependencies and generating coherent responses.

The encoder LSTM processes the input sequence $X = (x_1, x_2, \dots, x_T)$ and generates a sequence of hidden states $H = (h_1, h_2, \dots, h_T)$, where h_t is computed as:

$$h_t = \text{LSTM}(x_t, h_{t-1}) \quad (2)$$

The decoder LSTM takes the final encoder hidden state h_T as its initial state and generates the output sequence $Y = (y_1, y_2, \dots, y_{T'})$, where each output y_t is computed as:

$$y_t, s_t = \text{LSTM}(y_{t-1}, s_{t-1}, c_t) \quad (3)$$

Here, s_t is the decoder hidden state, and c_t is the context vector obtained from the attention mechanism.

4.4 Dynamic Inference Mechanism for Real-Time Responses

A dynamic inference mechanism was implemented to enable real-time responses from the chatbot. This mechanism optimized the inference process by efficiently updating the model's internal state and generating responses on-the-fly, enhancing the chatbot's responsiveness and user experience.

During inference, the decoder generates one output token at a time based on the previous output token and the current hidden state. The process continues until the end-of-sequence token is generated or the maximum sequence length is reached.

Let y_t be the output token at time step t , s_t be the decoder hidden state, and $p(y_t|y_{<t}, X)$ be the probability distribution over the output vocabulary given the previous outputs and input sequence X . The output token y_t is sampled from this distribution as follows:

$$y_t \sim p(y_t|y_{<t}, X) \quad (4)$$

The hidden state s_t is then updated based on the generated output token y_t and the previous hidden state s_{t-1} :

$$s_t = LSTM(y_t, s_{t-1}, c_t) \quad (5)$$

This dynamic inference mechanism allows the chatbot to generate responses in real-time, enabling natural and interactive conversations with users.

5 Results and Discussion

5.1 Experimental Setup

The chatbot model was trained using the following configuration:

- Dataset: Cornell Movie Dialogs Corpus
- Model Architecture: Encoder-Decoder with Luong Attention
- Embedding Size: 500
- Encoder Layers: 2
- Decoder Layers: 2
- Dropout: 0.1
- Batch Size: 64
- Optimizer: Adam
- Learning Rate: 0.0001
- Iterations: 4000

5.2 Performance Evaluation

The trained chatbot model was evaluated using sample conversations with users. The model demonstrated the ability to engage in coherent and contextually relevant conversations for general topics.

However, the model encountered challenges when presented with out-of-vocabulary words or domain-specific topics that were not covered in the training dataset.

5.3 Analysis and Limitations

While the chatbot model demonstrated promising performance in generating coherent responses for general conversations, it had limitations in handling out-of-domain queries and dealing with rare or unknown words. These limitations can be attributed to the following factors:

- Limited Vocabulary: The model’s vocabulary was constrained by the training dataset, which may not cover all possible words or topics encountered in real-world conversations.
- Lack of Domain-Specific Knowledge: The Cornell Movie Dialogs Corpus, while diverse, may not provide sufficient coverage of specialized domains or topics, limiting the model’s ability to engage in domain-specific conversations.


 > hi there
Bot: hi .
> How are you?
Bot: i m fine .
> What are you doing?
Bot: i m going to see you .
> Where?
Bot: in the house .
> Okay.Thanks
Error: Encountered unknown word.
> Thank you.
Bot: you re welcome .
> Bye
Bot: bye .
> q

Figure 1: Sample conversation with the chatbot model.

- Lack of Common Sense Reasoning: The model relies solely on the patterns present in the training data and may struggle with tasks that require common sense reasoning or real-world knowledge beyond the scope of the dataset.

6 Future Enhancements

To address the limitations of the current chatbot model and improve its performance, several enhancements can be explored:

6.1 Attention Mechanism Refinement

While the Luong attention mechanism was employed in this project, other attention mechanisms, such as the Bahdanau attention or the Transformer-based multi-head attention, could be explored to potentially improve the model's ability to focus on relevant parts of the input sequence.

6.2 Emotion and Sentiment Analysis

Incorporating sentiment analysis models into the chatbot pipeline could enhance the model's understanding of user emotions and sentiment. This would enable the chatbot to respond more

empathetically and adapt its tone and language accordingly, leading to more engaging and natural conversations.

6.3 Transfer Learning and Fine-Tuning

Leveraging pre-trained language models, such as BERT, GPT, or RoBERTa, through transfer learning and fine-tuning techniques could potentially improve the chatbot's language understanding and generation capabilities. These pre-trained models have been trained on large corpora and can provide a strong starting point for further fine-tuning on the chatbot dataset.

6.4 Enhanced Error Handling and Recovery

Implementing robust error handling mechanisms could improve the chatbot's ability to gracefully handle unexpected user inputs, out-of-domain queries, or system failures. Strategies for error recovery, clarification, and fallback responses could be incorporated to maintain smooth user interactions and minimize frustration.

7 Conclusion

In conclusion, this project successfully implemented a deep learning-based chatbot model capable of understanding and generating natural language responses. The encoder-decoder architecture with attention mechanisms and advanced preprocessing techniques demonstrated promising results in generating coherent and contextually relevant responses for general conversations.

However, the model encountered challenges when dealing with out-of-vocabulary words and domain-specific topics not covered in the training dataset. Future enhancements, such as refining the attention mechanism, incorporating sentiment analysis, leveraging transfer learning, and improving error handling, could address these limitations and enhance the chatbot's performance.

Despite the current limitations, the developed chatbot model holds promise for applications like customer support and information retrieval, highlighting the need for ongoing research and innovation in the field of conversational AI.

8 References

- <https://www.analyticsvidhya.com/blog/2021/07/build-a-simple-chatbot-using-python-and-nltk/>
- <https://github.com/mayli10/deep-learning-chatbot>
- <https://skillfloor.medium.com/building-chatbots-with-python-natural-language-processing-and-ai-c573f1e542bc>
- https://www.youtube.com/watch?v=CSpsB4_5WQ
- Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: Analyzing text with the natural language toolkit. O'Reilly Media, Inc.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems, 27.

- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. International Conference on Learning Representations (ICLR).
- Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. Conference on Empirical Methods in Natural Language Processing (EMNLP).

9 Contributions

The contributions of each team member were as follows:

- **Yagnesh Dave:** Designed the overall architecture of the chatbot model, incorporating sequence-to-sequence with attention mechanisms. Implemented LSTM layers, word embeddings, and loss functions for natural language understanding and generation. Led the training process, fine-tuning model hyperparameters, and optimizing performance.
- **Ashish Cherukuri:** Researched and implemented advanced generative models, such as Transformer and GPT, to explore their potential for enhancing the chatbot’s language generation capabilities. Developed the inference pipeline for the chatbot model, enabling real-time response generation based on user input. Integrated model inference with preprocessing and post-processing steps for smooth user interaction.
- **Adarsh Kumar:** Led the effort to collect and preprocess the Cornell Movie Dialog dataset for training the chatbot model. Implemented data cleaning techniques, including text normalization, removal of special characters, tokenization, and advanced preprocessing methods like stemming and lemmatization. Contributed to report writing and documentation.
- **Varun Purwar:** Researched and explored additional datasets for potential inclusion in the project, such as open-domain conversational datasets and domain-specific corpora. Assisted in data preprocessing tasks, including data cleaning, formatting, and handling rare or out-of-vocabulary words. Contributed to report writing and documentation.

All team members contributed to the project report, code documentation, and collaborative discussions throughout the project’s development.