A PROJECT REPORT ON

Conversational Bot

ACKNOWLEDGEMENT

First and foremost I am extremely grateful to my supervisors, for their invaluable advice, continuous support. Their immense knowledge and plentiful experience have encouraged me in all the time of my academics finally; I would like to express my gratitude to my parents, without their tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my thesis.

Name: Md. Amin

UID: 16STUCJHD00201

Guide: Asst. Prof. Dr. Abhay Kumar Sinha

Contents

Abstract:

Chat bots have a unique ability to engage in social and emotional interactions with humans that can be leveraged to foster creative expression. The aim is to enable a conversation bot which can help in dealing with costumers. With the advancement in neural networks we can use seq2seq models and transformers for build the Chabot, which can replicate humans.

1. Introduction:

Chatbots are everywhere. From online assistants, such as Microsoft's Cortana, to "helper bots" on messaging applications like Slack, to home applications like Amazon.com's Alexa, chatbots have become one of the most visible – and flawed – consumer-facing applications of artificial intelligence and machine learning. Most of the organization has customer support department, in which they can use AI-Chatbot 24/7 for support and help. This is one of the use case and there are many others. We focus on this use case for our project. The aim to build a conversation chat-bot which can help people with their questions. For example, a user can ask a question to the bot an bot will reply the answer.

2. Literature Review:

Eliza : The first chatbot

One of the first chatbots, ELIZA, was developed in 1966 by computer scientist Joseph Weizenbaum at the MIT Artificial Intelligence Laboratory. Weizenbaum designed ELIZA to mimic human interaction through pattern recognition; ELIZA could not, however, react to queries in their full context.

While ELIZA was designed to simply imitate human interaction, researchers recognized the potential of similar chatbots to provide real value to users in a wide range of contexts. Over the next four decades, engineers would experiment with more helpful chatbot applications and further expand the scope of how chatbots are defined

Attention is all you need

A paper form From: Ashish Vaswani and others which changed trhe way previous methods works. https://arxiv.org/abs/1706.03762

What problem is the paper attempting to solve?

Recurrent neural networks (RNN), long short-term memory networks(LSTM) and gated RNNs are the popularly approaches used for Sequence Modelling tasks such as machine translation and language modeling. However, RNN/CNN handle sequences word-by-word in a sequential fashion. This sequentiality is an obstacle toward parallelization of the process. Moreover, when such sequences are too long, the model is prone to forgetting the content of distant positions in sequence or mixes it with following positions' content.

Recent works have achieved significant improvements in computational efficiency and model performance through factorization tricks and conditional computation. But they are not enough to eliminate the fundamental constraint of sequential computation.

Attention mechanisms are one of the solutions to overcome the problem of model forgetting. This is because they allow dependency modeling without considering their distance in the input or output sequences. Due to this feature, they have become an integral part of sequence modeling and transduction models. However, in most cases attention mechanisms are used in conjunction with a recurrent network. We will stick to this paper and try to implement the method explained.

3. Data Description:

Cornell Movie--Dialogs Corpus: A large metadata-rich collection of fictional conversations extracted from raw movie scripts. (220,579 conversational exchanges between 10,292 pairs of movie characters in 617 movies).

Credits: Cristian Danescu-Niculescu-Mizil and Lillian Lee

Speaker-level information

speakers in this dataset are movie characters. We take speaker index from the original data release as the speaker name. For each character, we further provide the following information as speaker-level metadata:

- character_name: name of the character in the movie
- movie_idx: index of the movie this character appears in
- movie_name: title of the movie
- gender: gender of the character ("?" for unlabeled cases)
- credit_pos: position on movie credits ("?" for unlabeled cases)

Utterance-level information

For each utterance, we provide:

- id: index of the utterance
- speaker: the speaker who authored the utterance
- conversation_id: id of the first utterance in the conversation this utterance belongs to
- reply_to: id of the utterance to which this utterance replies to (None if the utterance is not a reply)
- timestamp: time of the utterance

- text: textual content of the utterance

Metadata for utterances include:

- movie_idx: index of the movie from which this utterance occurs
- parsed: parsed version of the utterance text, represented as a SpaCy Doc

Conversational-level information

Conversations are indexed by the id of the first utterance that make the conversation. For each conversation we provide:

- movie_idx: index of the movie from which this utterance occurs
- movie_name: title of the movie
- release_year: year of movie release
- rating: IMDB rating of the movie
- votes: number of IMDB votes
- genre: a list of genres this movie belongs to

Corpus-level information

Additional information for the movies these conversations occur are included as Corpus-level metadata, which includes, for each movie:

- url: a dictionary mapping movie_idx to the url from which the raw sources were retrieved
- name: name of the corpus

4. Data Cleaning:

We will use the conversations in movies and TV shows provided by Cornell Movie-Dialogs Corpus, which contains more than 220 thousands conversational exchanges between more than 10k pairs of movie characters, as our dataset. movie_conversations.txt contains list of the conversation IDs and movie_lines.text contains the text of assoicated with each conversation ID.

For further information regarding the dataset, please check the README file in the zip file in this link: https://github.com/Khanamin-XOR/Btech-Thesis.

The maximum length of the sentence to be MAX_LENGTH=12, has been taken for simplicity and we will achieve this with padding later on.

We preprocess our dataset in the following order:

- Extract MAX_SAMPLES conversation pairs into list of questions and `answers.
- Preprocess each sentence by removing special characters in each sentence.
- Build tokenizer (map text to ID and ID to text) using TensorFlow Datasets SubwordTextEncoder.
- Tokenize each sentence and add START_TOKEN and END_TOKEN to indicate the start and end of each sentence.
- Filter out sentence that has more than MAX_LENGTH tokens.
- Pad tokenized sentences to MAX_LENGTH

```python
def preprocess_sentence(sentence):
    sentence = sentence.lower().strip()
    # creating a space between a word and the punctuation following it
    # eg: "he is a boy." => "he is a boy ."
    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)
    sentence = re.sub(r'[" "]+', " ", sentence)
    # removing contractions
    sentence = re.sub(r"i'm", "i am", sentence)
    sentence = re.sub(r"he's", "he is", sentence)
    sentence = re.sub(r"she's", "she is", sentence)
    sentence = re.sub(r"it's", "it is", sentence)
    sentence = re.sub(r"that's", "that is", sentence)
    sentence = re.sub(r"what's", "that is", sentence)
    sentence = re.sub(r"where's", "where is", sentence)
    sentence = re.sub(r"how's", "how is", sentence)
    sentence = re.sub(r"\'ll", " will", sentence)
    sentence = re.sub(r"\'ve", " have", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"\'d", " would", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"won't", "will not", sentence)
    sentence = re.sub(r"can't", "cannot", sentence)
    sentence = re.sub(r"n't", " not", sentence)
    sentence = re.sub(r"n'", "ng", sentence)
    sentence = re.sub(r"'bout", "about", sentence)
    # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")
    sentence = re.sub(r"[^a-zA-Z?.!,]+", " ", sentence)
    sentence = sentence.strip()
    return sentence
```

In the above function we are cleaning the data, removing the abbreviations.

5. Data Preprocessing:

Create tf.data.Dataset

We are going to use the tf.data.Dataset API to contruct our input pipline in order to utilize features like caching and prefetching to speed up the training process.The transformer is an auto-regressive model: it makes predictions one part at a time, and uses its output so far to decide what to do next.During training this example uses teacher-forcing. Teacher forcing is passing the true output to the next time step regardless of what the model predicts at the current time step.As the transformer predicts each word, self-attention allows it to look at the previous words in the input sequence to better predict the next word.To prevent the model from peaking at the expected output the model uses a look-ahead mask.Target is divided into decoder_inputs which padded as an input to the decoder and cropped_targets for calculating our loss and accuracy.

6. Model building:

Attention:

Scaled dot product Attention

The scaled dot-product attention function used by the transformer takes three inputs: Q (query), K (key), V (value). The equation used to calculate the attention weights is:

$$Attention(Q, K, V) = softmax_k(\frac{QK^T}{\sqrt{d_k}})V$$

As the softmax normalization is done on the key, its values decide the amount of importance given to the query.

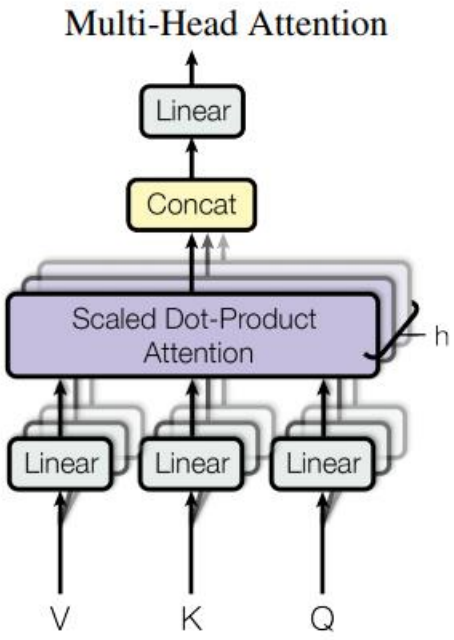
The output represents the multiplication of the attention weights and the value vector. This ensures that the words we want to focus on are kept as is and the irrelevant words are flushed out.

The dot-product attention is scaled by a factor of square root of the depth. This is done because for large values of depth, the dot product grows large in magnitude pushing the softmax function where it has small gradients resulting in a very hard softmax.

For example, consider that query and key have a mean of 0 and variance of 1. Their matrix multiplication will have a mean of 0 and variance of dk. Hence, *square root of dk* is used for scaling (and not any other number) because the matmul of query and key should have a mean of 0 and variance of 1, so that we get a gentler softmax.

The mask is multiplied with *-1e9 (close to negative infinity).* This is done because the mask is summed with the scaled matrix multiplication of query and key and is applied immediately before a softmax. The goal is to zero out these cells, and large negative inputs to softmax are near zero in the output.

Multi-head attention:



Multi-head attention consists of four parts:

- Linear layers and split into heads.
- Scaled dot-product attention.
- Concatenation of heads.
- Final linear layer.

Each multi-head attention block gets three inputs; Q (query), K (key), V (value). These are put through linear (Dense) layers and split up into multiple heads.

The scaled_dot_product_attention defined above is applied to each head (broadcasted for efficiency). An appropriate mask must be used in the attention step. The attention output for each head is then concatenated (using tf.transpose, and tf.reshape) and put through a final Dense layer.

Instead of one single attention head, query, key, and value are split into multiple heads because it allows the model to jointly attend to information at different positions from different representational spaces. After the split each head has a reduced dimensionality, so the total computation cost is the same as a single head attention with full dimensionality.

Transformer

Masking

create_padding_mask and create_look_ahead are helper functions to creating masks to mask out padded tokens, we are going to use these helper functions as tf.keras.layers.Lambda layers.

Mask all the pad tokens (value 0) in the batch to ensure the model does not treat padding as input.

Positional encoding

Since this model doesn't contain any recurrence or convolution, positional encoding is added to give the model some information about the relative position of the words in the sentence.

The positional encoding vector is added to the embedding vector. Embeddings represent a token in a d-dimensional space where tokens with similar meaning will be closer to each other. But the embeddings do not encode the relative position of words in a sentence. So after adding the positional encoding, words will be closer to each other based on the *similarity of their meaning and their position in the sentence*, in the d-dimensional space.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Encoder Layer

Each encoder layer consists of sublayers:

1. Multi-head attention (with padding mask)
2. 2 dense layers followed by dropout

Each of these sublayers has a residual connection around it followed by a layer normalization. Residual connections help in avoiding the vanishing gradient problem in deep networks.

The output of each sublayer is LayerNorm(x + Sublayer(x)). The normalization is done on the d_model (last) axis.

Encoder

The Encoder consists of:

1. Input Embedding
2. Positional Encoding
3. num_layers encoder layers

The input is put through an embedding which is summed with the positional encoding. The output of this summation is the input to the encoder layers. The output of the encoder is the input to the decoder.

Decoder Layer

Each decoder layer consists of sublayers:

1. Masked multi-head attention (with look ahead mask and padding mask)
2. Multi-head attention (with padding mask). value and key receive the *encoder output* as inputs. query receives the *output from the masked multi-head attention sublayer*.
3. 2 dense layers followed by dropout

Each of these sublayers has a residual connection around it followed by a layer normalization. The output of each sublayer is LayerNorm(x + Sublayer(x)). The normalization is done on the d_model (last) axis.

As query receives the output from decoder's first attention block, and key receives the encoder output, the attention weights represent the importance given to the decoder's input based on the encoder's output. In other words, the decoder predicts the next word by looking at the encoder output and self-attending to its own output. See the demonstration above in the scaled dot product attention section.

Decoder

The Decoder consists of:

1. Output Embedding
2. Positional Encoding
3. N decoder layers

The target is put through an embedding which is summed with the positional encoding. The output of this summation is the input to the decoder layers. The output of the decoder is the input to the final linear layer.

Transformer

Transformer consists of the encoder, decoder and a final linear layer. The output of the decoder is the input to the linear layer and its output is returned.

7. Conclusion:

The model is performing goof, it can be improved further. The model can be tuned and we can also increase the amount of data. The performance of the model can be assumed as a baseline model for similar case study. There is a huge scope of improving the performance of model by hyper tuning them.

8. References

1. **https://arxiv.org/abs/1706.03762**
2. **https://www.tensorflow.org/**
3. **https://www.w3schools.com/python/python_regex.asp**
4. **https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html**