Natural Language Processing

Prof. Dr. Arthur Jacobs

Summersemester 2021

Freie Universität Berlin

Chatbot using Deep learning algorithm with movie subtitles

Project by

Frenny Macwan

5504865

Abstract:

Chat bots are programs, are often designed to support clients on websites or via phone. The chat bots are generally used in messaging applications like Slack, Facebook Messenger, or Telegram. Generally question- answer pairs are used for making chat bots. But here I have tried to build it using Cornell Movie-Dialogs Corpus. My main focus is to train deep learning model on these movie subtitles and get the outputs.

Data:

Cornell Movie-Dialogs Corpus Reference: https://www.kaggle.com/rajathmc/cornell-moviedialog-corpus

This corpus contains a metadata-rich collection of fictional conversations extracted from raw movie scripts:

In files the field separator is " +++$+++ "

- movie_lines.txt
    - contains the actual text of each utterance
    - fields:
        - lineID
        - characterID (who uttered this phrase)
        - movieID
        - character name
        - text of the utterance

- movie_conversations.txt
    - the structure of the conversations
    - fields
        - characterID of the first character involved in the conversation
        - characterID of the second character involved in the conversation
        - movieID of the movie in which the conversation occurred
        - list of the utterances that make the conversation, in chronological
          order: ['lineID1','lineID2',..'lineIDN']
          has to be matched with movie_lines.txt to reconstruct the actual content

Data Pre-processing:

As files has field separator " +++$+++ ", first thing is to split those lines and store it into the list. For both files movie_lines.txt and movie_conversation.txt, this task is performed and stored into different lists. Then sorted both lists i.e. question and answer. These texts are having unnecessary characters which can be removed. So cleaned the text by removing certain character or changing their format. Like “I’m” is altered as “I am” and “n’t” is altered with “not”.

```python
def clean_text(text):
    text = text.lower()
    text = re.sub(r"i'm","i am",text)
    text = re.sub(r"he's","he is",text)
    text = re.sub(r"she's","she is",text)
    text = re.sub(r"that's","that is",text)
    text = re.sub(r"what's","what is",text)
    text = re.sub(r"where's","where is",text)
    text = re.sub(r"'ll"," will",text)
    text = re.sub(r"'ve"," have",text)
    text = re.sub(r"'re"," are",text)
    text = re.sub(r"'d"," would",text)
    text = re.sub(r"won't","will not",text)
    text = re.sub(r"can't","cannot",text)
    text = re.sub('[^a-zA-Z0-9]',' ',text)
    return text
```

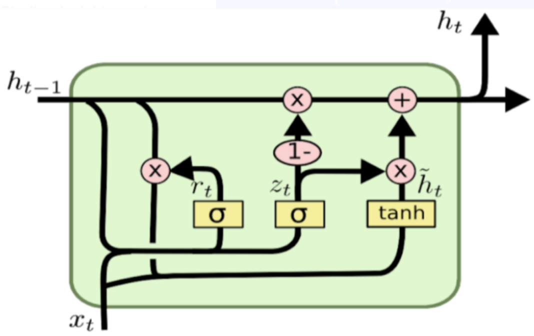
Then these question and answer lists are combined and added padding at the start of sentence <SOS> and end of the sentence <EOD>. Then converted words to indexes.

Approach:

To get overview of how well some of the deep learning algorithms perform, GRU, and LSTM are used on given dataset.

GRU : A Gated Recurrent Unit (GRU), as its name suggests, is a variant of the RNN architecture, and uses gating mechanisms to control and manage the flow of information between cells in the neural network. The GRU cell is similar to the LSTM cell but with a few important differences. First, there is no hidden state. The cell state adopts the functionality of the hidden state from the LSTM cell design. Next, the processes of determining what the cell states forgets and what part of the cell state is written to are consolidated into a single gate. Only the portion of the cell state that has been erased is written to. Finally, the entire cell state is given as an output. This is different from the LSTM cell which chooses what to read from the cell state to produce an output. All of these changes together provide a simpler design with less parameters than the LSTM. Less parameters, however, may come at the cost of decreased expressibility.

https://github.com/FrennyMacwan/NLP_project/blob/main/chatbot_gru.ipynb

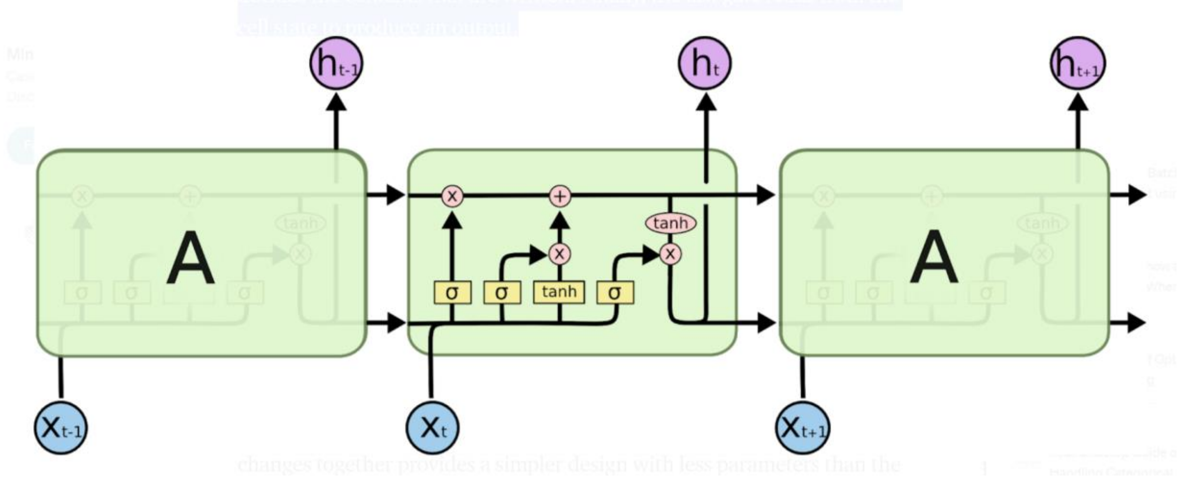


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM: The LSTM cell maintains a cell state that is read from and written to. There are 4 gates that regulate the reading, writing, and outputting values to and from the cell state, dependent upon the input and cell state values. The first gate determines what the hidden state forgets. The next gate is responsible for determining what part of the cell state is written to. The third gate decides the contents that are written. Finally, the last gate reads from the cell state to produce an output.

For this, I created an input features dictionary that will store our input tokens as key-value pairs, the word being the key and value is the index. Similarly, for target tokens, created a target features dictionary. Features dictionary will help us encode our sentences into one-hot vectors. After all, computers only understand the numbers. To decode the sentences we will need to create the reverse features dictionary that stores index as a key and word as a value. Here, I have used LSTM as, LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

https://github.com/FrennyMacwan/NLP_project/blob/main/generativechatbot.ipynb



Additionally, tried to implement seq2seq model using seq2seq_wrapper. It consists of two RNNs - an encoder and a decoder. The encoder reads the input sequence, word by word and emits a context (a function of final hidden state of encoder), which would ideally capture the essence (semantic summary) of the input sequence. Based on this context, the decoder generates the output sequence, one word at a time while looking at the context and the previous word during each timestemp. This is an oversimplification, but it gives an idea of what happens in seq2seq.

https://github.com/FrennyMacwan/NLP_project/blob/main/chatbot_seq2seq.ipynb

Conclusion and Result:

GRU and LSTM models are trained over 600 epochs. The LSTM model displays much greater volatility throughout its gradient descent compared to the GRU model. This may be due to the fact that there are more gates for the gradients to flow through, causing steady progress to be more difficult to maintain after many epochs. Additionally, the GRU model was able to train faster than the LSTM model. For future work, different kernel and recurrent initializers could be explored for each cell type. Here I have used a very small dataset and got an accuracy of around 20%. In the future for a larger dataset, the model might give better accuracy. The limitation of using this approach for creating chatbot is that we need a very large dataset to give the best responses to the user as we can see in the output that chatbot does not give the right responses in some cases because of a smaller dataset.

Here are different outputs.

```
chatbot.start_chat()
```

```
Hi, I'm a chatbot trained on random dialogs. Would you like to chat with me?
yes sure
 i don really really that that that expect expect expect
then what do you want
 that is what i i you
what
 no thirty
are you mad
 thirty are
lol
 no thirty
bye
Ok, have a great day!
```

```
# Begin chatting
evaluateInput(encoder, decoder, searcher, voc)
```

```
> hi, how's it going?
Bot: Not good so far.
> what are you talking about
Bot: Nothing, Bellvue
> quit
```