on Chatbots

Alvin Li, Charmaine Lam

April 2020

1 Introduction

Chatbots are becoming more and more prevalent in modern-day society. Through the use of online services such as DialogFlow, one can create and use chatbots quite easily by using APIs (a version of this was created as a Chrome extension here). To challenge ourselves further, we decided to create a general chatbot mainly using machine learning in Python. While the aforementioned chatbots are intent-based, our goal is only to create a chatbot that responds to general questions/ statements. It should have an open domain with no memory (each input is independent of each other). Also, the bot would be generative, which means that it should not be retrieving pre-set answers from a database.

In this project, we will be using various models for seq2seq in Tensorflow 2.0.

Sidenote: The original goal was to create a therapy chatbot but those specific datasets are difficult to find.

2 Dataset

The dataset we used was found on Kaggle, collected from the r/CasualConversation subreddit through pushshift.io. It contains 56296 entries, with conversations of length 3. Only the first two parts of the conversations were used.

A secondary dataset was created to test the models to ensure the models are setup correctly to prevent the waste of time. It contains two columns: The first is a combination of a number between 1 and 20 (ex. 12), a mathematical operation (ex. subtract), and another number between 1 and 20 (ex. 5); The second is a number denoting the result of the previous column (ex. 7).

3 Data Preprocessing

The following techniques were used to 'clean' the data:

1. Removal of Non-Alphabetical characters
2. Lowercase

   To reduce the vocabulary size, the entire dataset was set to lower case.

3. Autocorrect

   Over 10000 words only appeared once or twice in the entire dataset, most of which are incorrectly spelled words (ex. corect). The autocorrect package was used on all sentences in the dataset. We noted that some names were autocorrected, like samsung became tamlung. The caught instances were re-corrected.

4. Remove Rarely-used Words

   Around half of the remainig total words that were still only used once or twice in the dataset, which caused some minor dimensionality issues when converting the dataset into one-hot encoding. These words are thus removed.

5. Reduction of Sentence Length

   The longest sentence length in the dataset was 60 words. However, once the sentence length was plotted with their frequency (Figure 1), it can be seen that most sentences lies between 1 and 25 words. Thus, sentences with longer than 25 words are cut short.

6. Integer Encoding

   Each unique word was turned into an integer

7. Padding

   All sentences regardless of length are padded up to the length of 27 (due to two extra tokens, start and end) by filling the rest (padding type: post) with zeros.

8. One-Hot Encoding

   One-hot encoded numpy arrays were of shape: Total Dataset Length, Sentence Length, Vocab Size. They were first declared as zeros, then ones were put at positions according to their respective integer encoded sequences.

The final cleaned dataset has an input vocabulary size of 6840 and an output vocabulary size of 5948. Note that two extra words (ie. `\t` and `\n`) were added to each as the start and end tokens.

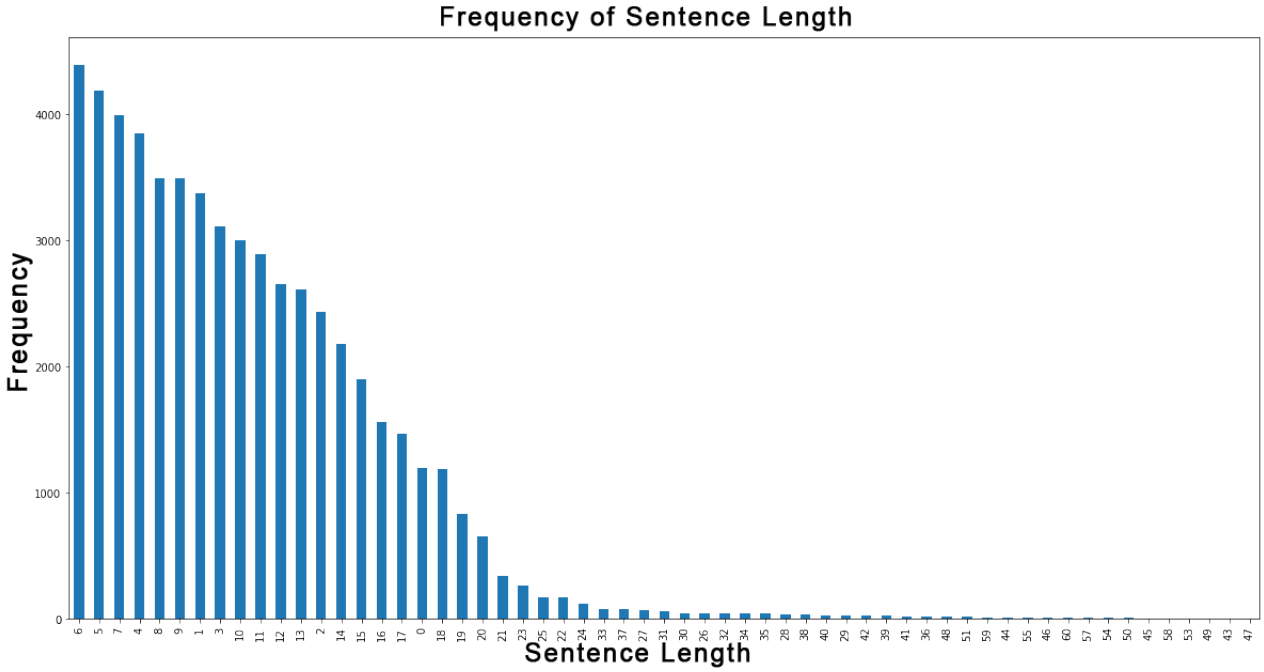

Figure 1: Frequency of Sentence Length

4 Models Overview

Almost all models used were seq2seq models, meaning it takes a sequence of inputs and outputs another sequence, which is perfect for chatbot tasks. In a chatbot, the input is the user's input, and the output is the chatbot's response. A Seq2Seq model consists of a decoder and an encoder. The encoder encodes the sequence by putting the input through a model composed mainly of LSTM or GRU layers. The encoded sequence is returned in the form of a hidden state vector. The decoder then uses the hidden state vector, along with another model composed mainly of LSTM or GRU layers to predict the output sequence.

What is a Long Short Term Memory layer (LSTM)?
Traditionally, Recurrent Neural Networks (RNN) such as LSTMs are used for sequential data (for example, stock prediction, text processing) due to their ability to back-propagate through time. However, RNNs suffer from the problem of exploding/ vanishing gradients as weights become massive or tiny when they were updated across the neurons via the chain rule [1], making them only suitable for 10 timesteps or less [2]. LSTMs partially resolve the problems by using cell gates [3]. It includes a memory gate, a forget gate, an input gate and an output state [4]. Every LSTM cell contains three types of inputs: the previous cell state c_{t-1}, the previous hidden state h_{t-1} and the current input x_t, as shown in Figure 2.
The input gate is the product of the sigmoid function of the hidden state and input ('squashed' between 0 and 1) and the tanh function of the hidden state and input ('squashed' between -1 and 1). Please note that W_x denotes the weights of each gate x, the states and

inputs are all multiplied by their respective weights prior to activation.

$$f_i = tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

The forget gate is the sigmoid function of the hidden state and input. It controls what gets forgetten.

$$f_f = \sigma(W_f \cdot [h_{t-1}, x_t])$$

The output gate is the sigmoid function of the hidden state and input

$$f_o = \sigma(W_o \cdot [h_{t-1}, x_t])$$

The current hidden state is calculated by the multiplication of the tanh function of the current cell state (calculated below), and the output of the output function.

$$h_t = f_o \otimes tanh(c_t)$$

The current cell state is calculated by multiplying (element-wise) the output of the forget gate and the previous cell state plus the output of the input gate. The beauty of this function is that the derivative of this function is additive, which means it would not suffer from vanishing/ exploding gradients due to multiplicative operations.

$$c_t = c_{t-1} \otimes f_f \oplus f_i$$

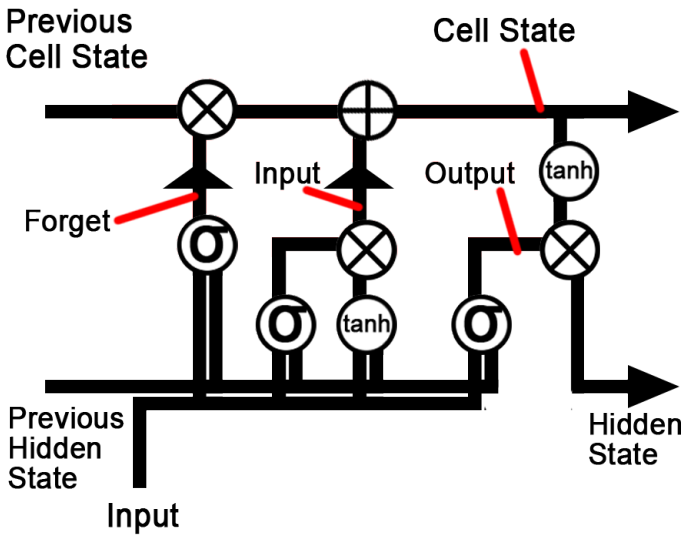

Figure 2: Representation of a LSTM cell [Li, A.]

5 Metrics

The standard form of measuring the effectiveness of the output is usually measured by BLEU scores. However, as mentioned by the paper by Liu et al., BLEU is not as effective

in measuring chatbot responses due to the diversity of the responses [6]. There is simply no ground-truth response for a certain statement/ question input. They proposed a embedding-based metric which would calculate the cosine similarity of the word embeddings, allowing for a wider variety of words [6]. They concluded that most metrics do not reflect human judgement [6]. With the consideration that our tests do not have a ground-truth or a proposed response, we decided that our metrics would be mainly based on human judgment. The metric is rather objective.
With 20 sample statements/questions, the responses will be rated as per the following, the higher the number, the better the response:

1. Completeness: Out of 2, on how complete a response is.
2. Humour: Out of 3, on how humorous a response is.
3. Grammar: Out of 4, on how much a sentence follows grammatical sense.
4. Sense: Out of 4, does it make sense?

   The metrics above are calculated and totalled per response, then averaged. The following is calculated once across the entire testing set, then added to the average of the metrics above.
5. Uniqueness: Out of 3, on whether the responses exempt uniqueness. If the responses are the same, the score would be 0.

The scores are then averaged between the co-authors.

6 Specific Models Used

1. Simple Seq2Seq

   The model is taken directly from the Keras Official Site example [5]. It uses one LSTM layer in the encoder, then one LSTM layer and a Dense layer in the decoder. The prediction is done by rearranging the model to take in hidden state, cell state and the original decoder input layer, outputting the original Dense layer and the decoder states. For each timestep, we find the word closest to the predicted output using argmax, then feed that back into the encoder for the next timestep. This continues until the stop token is received or the max output length is reached.
   Bidirectional LSTMs: [needs fixing]
   They were all trained with 100 epochs, and the models' accuracy and loss are observed to plateau (or fluctuate) beyond 100 epochs. An increased training size is observed to lead to better results. The following variants were tested:

   (a) Base Model (RMSprop lr = 0.001, on word level, validation split = 0.05)
   (b) Base Model with RMSprop (lr = 0.01)
   (c) Base Model with Adam (lr = 0.001)
   (d) Base Model with Reversed input
   (e) Base Model with character level input and output
   (f) Base Model with Bidirectional LSTM
   (g) Base Model with GRU

Note: Reversed input was tested as it was found to be more effective than the initial input [7]

2. Seq2Seq with Embeddings
   Embeddings are traditionally used to improve word-based tasks. Embeddings are vectors that represent words in a sense that related or similar words have similar vectors. The pretrained GLoVE vector was used [8]. Appoximately 90% of the vocabulary can be found within GLoVE. The rest are initialized randomly. Note the inputs are integer-encoded while the outputs are one-hot encoded. The following variants were tested:

   (a) Base Model (separate input/output embeddings, weights by GLoVE)

   (b) Base Model with shared embeddings (GLoVE)

   (c) Base Model with separate embeddings, no initialization

3. Seq2Seq with Attention The model is taken from the official Tensorflow site here, which is an implementation of the attention model suggested by Bahdanau et. al. [8]. Subsequent changes were made to accommodate for LSTM layers as well as different attention models.
   Attention essentially creates a link between the input and output on a word-level. It tells the decoder which words to pay attention to. The attention layer exists between the decoder and the encoder, takes in the encoder hidden state and the decoder hidden state and calculates a score. The score depends on the type of attention we are calculating. The scores are then put through a softmax layer, and multiplied by the encoder hidden state again, which is the main reason attention works. This creates a vector that tells us which words are closely related between the input and output. The vector is then summed and used as input for the decoder.
   Bahdanau's Model [8]: The encoder is a bidirectional GRU, resulting in two hidden states, one forward and one backward. They are then concatenated. Score function is additive.
   Luong's Global Attention Model [9]: Stacked LSTMs are used, mutiplicative score function. Note: The authors proposed multiple score functions.
   Teacher Forcing was also used in training most of the implementations. It involves feeding the 'correct' target output as the next input rather than the predicted output.

   The following variants were tested:

   (a) Base Model (GRU, Bahdanau's Score Function)

   (b) Base Model with LSTM

   (c) Base Model with BiGRU

   (d) Base Model with 2 LSTM layers per encoder/decoder

   (e) Base Model with Luong's Score Function [9]

   (f) Base Model with Beam Search

4. BERT

5. Transformer

6. GAN https://github.com/oswaldoludwig/Adversarial-Learning-for-Generative-Conversational-Agents

7 References

[1] https://arxiv.org/pdf/1211.5063.pdf
[2] https://papers.nips.cc/paper/522-induction-of-multiscale-temporal-structure.pdf
[3] https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577
[4] https://arxiv.org/pdf/1909.09586.pdf [5] https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html [6] https://arxiv.org/pdf/1603.08023.pdf [7] https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf [8] https://arxiv.org/pdf/1409.0473.pdf [9] https://arxiv.org/pdf/1508.04025.pdf [10] https://towardsdatascience.com/a illustrated-attention-5ec4ad276ee37eef