AN OPEN DOMAIN CONVERSATIONAL CHATBOT

A report by

Abhijeet Singh,

Roll No. 15-1-5-024, student of the batch Btech-CSE 2015-2019,

National Institute of Technology, Silchar(Assam),

For the summer internship pursued under the guidance of

Professor Sukumar Nandi,

Computer Science and Engineering Department,

Indian Institute of Technology, Guwahati,

Duration: 15 May-10 July, 2018.

ACKNOWLEDGEMENT

I, Abhijeet Singh, thank Prof. Sukumar Nandi, Computer Science and Engineering Department, IIT Guwahati for giving me the opportunity to work under his guidance at IIT Guwahati. I also thank Mr. Dhrubajyoti Pathak for guiding and helping me in this internship. I also want to thank Mr. Bhriguraj Borah and Computer Science Department of IIT Guwahati for letting me use various facilities.

CONTENT

INTRODUCTION

Chatbot stands for chatter robot. It is a computer program which has the ability to chat with human users. Audio or text can be used as medium of communication. The primary aim of writing these conversational programs is to have a powerful tool that chat in such a way that the human user cannot realize that he is talking to a software program. Chatbot programs are also referred to as Artificial Conversational Entities, talk bots, chatterbots and chatterboxes. It is Michael Mauldin who coined the term, chatterbot.

In an **open domain** setting the user can take the conversation anywhere. There isn't necessarily have a well-defined goal or intention. Conversations on social media sites like Twitter and Reddit are typically open domain – they can go into all kinds of directions. The infinite number of topics and the fact that a certain amount of world knowledge is required to create reasonable responses makes this a hard problem.

ABOUT THE PROJECT

The chatbot is based on the translate model on the TensorFlow repository, with some modification to make it work for a chatbot. It's a sequence to sequence model with attention decoder. The encoder is a single utterance, and the decoder is the response to that utterance. An utterance could be a sentence, more than a sentence, or even less than a sentence, anything people say in a conversation!

The chatbot is built using a wrapper function for the sequence to sequence model with padding. The loss function we use is sequence loss and gradient clipping mechanism is used to control maximum and minimum value of gradient.

```
# Setting up the Loss Error, the Optimizer and Gradient Clipping
with tf.name_scope("optimization"):
    loss_error = tf.contrib.seq2seq.sequence_loss(training_predictions,
                                                  targets,
                                                  tf.ones([input_shape[0],
sequence_length]))
    optimizer = tf.train.AdamOptimizer(learning_rate)
    gradients = optimizer.compute_gradients(loss_error)
    clipped_gradients = [(tf.clip_by_value(grad_tensor, -5., 5.), grad_variable) for
grad_tensor, grad_variable in gradients if grad_tensor is not None]
    optimizer_gradient_clipping = optimizer.apply_gradients(clipped_gradients)
```

Encoder-Decoder model:

Encoder:

The encoder simply takes the encoder inputs. The only thing we care about is the final hidden state. This hidden state holds the information from all of the inputs. We reverse the encoder inputs because we are using sequence length with bidirectional dynamic RNN. This automatically returns the last relevant hidden state based on sequence length.

```
# Creating the Encoder RNN
def encoder_rnn(rnn_inputs, rnn_size, num_layers, keep_prob, sequence_length):
    lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
    lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
    encoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
    encoder_output, encoder_state = tf.nn.bidirectional_dynamic_rnn(cell_fw = encoder_cell,
                                                                    cell_bw = encoder_cell,
                                                                    sequence_length =
sequence_length,
                                                                    inputs = rnn_inputs,
                                                                    dtype = tf.float32)
    return encoder_state
```

Decoder:

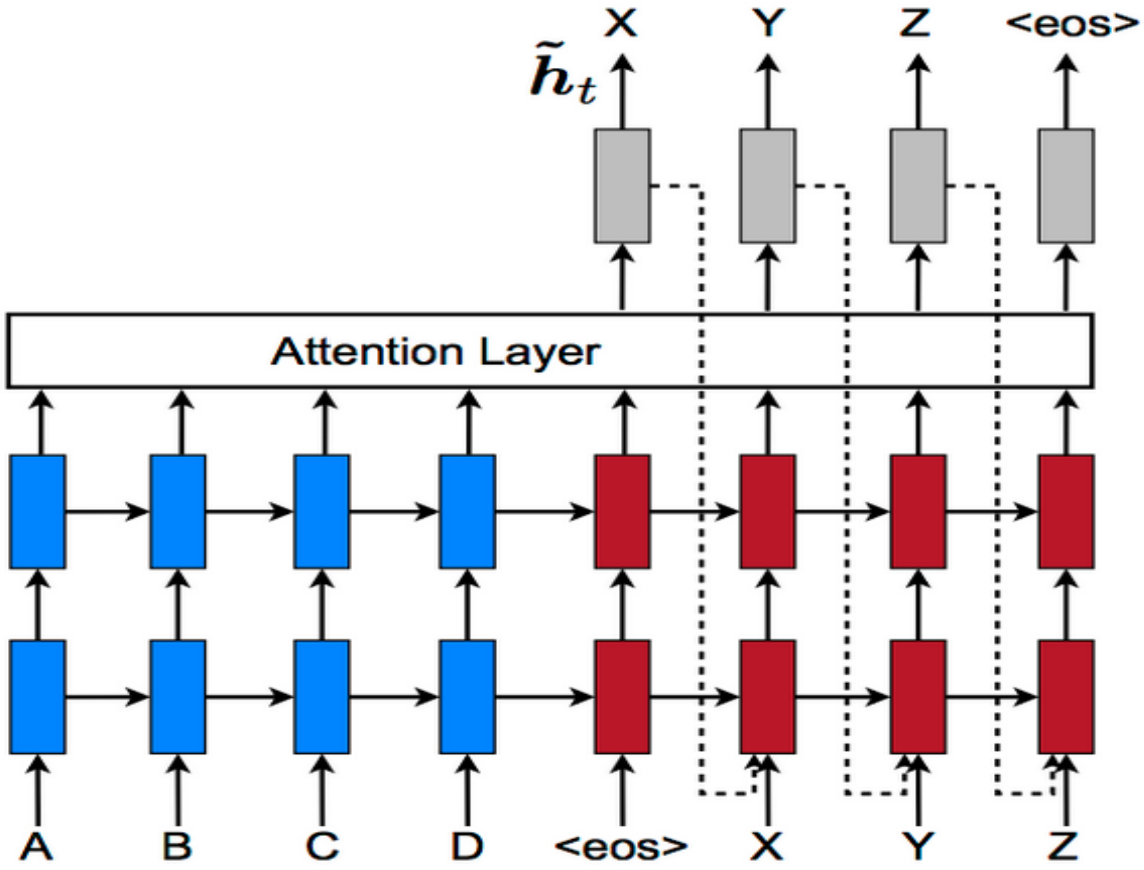
This simple decoder takes in the final hidden state from the encoder as its initial state. We will also embed the decoder inputs and process them with the decoder RNN. The outputs will be normalized with softmax and then compared with the targets. Note that the decoder inputs starts with a GO token which is to predict the first target token. The decoder input's last relevant token with predict the EOS target token.

```
# Creating the Decoder RNN
def decoder_rnn(decoder_embedded_input, decoder_embeddings_matrix, encoder_state,
num_words, sequence_length, rnn_size, num_layers, word2int, keep_prob, batch_size):

    with tf.variable_scope("decoding") as decoding_scope:

        lstm = tf.contrib.rnn.BasicLSTMCell(rnn_size)
        lstm_dropout = tf.contrib.rnn.DropoutWrapper(lstm, input_keep_prob = keep_prob)
        decoder_cell = tf.contrib.rnn.MultiRNNCell([lstm_dropout] * num_layers)
        weights = tf.truncated_normal_initializer(stddev = 0.1)
        biases = tf.zeros_initializer()
        output_function = lambda x: tf.contrib.layers.fully_connected(x,
                                                num_words,
                                                None,
                                                scope = decoding_scope,
                                                weights_initializer = weights,
                                                biases_initializer = biases)
        training_predictions = decode_training_set(encoder_state,
                                        decoder_cell,
                                        decoder_embedded_input,
                                        sequence_length,
                                        decoding_scope,
                                        output_function,
                                        keep_prob,
                                        batch_size)
        decoding_scope.reuse_variables()
        test_predictions = decode_test_set(encoder_state,
                                    decoder_cell,
                                    decoder_embeddings_matrix,
                                    word2int['<SOS>'],
                                    word2int['<EOS>'],
                                    sequence_length - 1,
                                    num_words,
                                    decoding_scope,
                                    output_function,
                                    keep_prob,
                                    batch_size)
    return training_predictions, test_predictions
```

Attention Layer:

Attention is proposed as a solution to the limitation of the Encoder-Decoder model encoding the input sequence to one fixed length vector from which to decode each output time step. This issue is believed to be more of a problem when decoding long sequences.

Instead of decoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step.



Here A, B, C, D are the input sequence (Encoder section). When End of Sequence (EOS) token is reached both attention context vector and last hidden state (ht) gets updated. Now comes the decoding part, the output of the encoder (i.e. last hidden state) acts as the first input state for the decoder, and the first output along with the attention vector acts as the next input for the next state of the decoder and so on, till EOS is reached.

Seq-to-Seq Wrapper:

This is an encoder-decoder wrapper function where the encoder embedding matrix, decoder embedding matrix and last hidden state of encoder are used as an input to the decoder function for predicting training and test sets.

```
# Building the seq2seq model
def seq2seq_model(inputs, targets, keep_prob, batch_size, sequence_length, answers_num_words, questions_num_words, encoder_embedding_size, decoder_embedding_size, rnn_size, num_layers, questionswords2int):

    encoder_embedded_input = tf.contrib.layers.embed_sequence(inputs,
                                                              answers_num_words + 1,
                                                              encoder_embedding_size,
                                                              initializer = tf.random_uniform_initializer(0, 1))
    encoder_state = encoder_rnn(encoder_embedded_input, rnn_size, num_layers, keep_prob, sequence_length)
    preprocessed_targets = preprocess_targets(targets, questionswords2int, batch_size)
    decoder_embeddings_matrix = tf.Variable(tf.random_uniform([questions_num_words + 1, decoder_embedding_size], 0, 1))
    decoder_embedded_input = tf.nn.embedding_lookup(decoder_embeddings_matrix, preprocessed_targets)
    training_predictions, test_predictions = decoder_rnn(decoder_embedded_input,
                                                         decoder_embeddings_matrix,
                                                         encoder_state,
                                                         questions_num_words,
                                                         sequence_length,
                                                         rnn_size,
                                                         num_layers,
                                                         questionswords2int,
                                                         keep_prob,
                                                         batch_size)
    return training_predictions, test_predictions
```

DATASET

The bot comes with the script to do the pre-processing for the Cornell Movie-Dialogs Corpus, created by Cristian Danescu-Niculescu-Mizil and Lillian Lee at Cornell University. This is an extremely well-formatted dataset of dialogues from movies. It has 220,579 conversational exchanges between 10,292 pairs of movie characters, involving 9,035 characters from 617 movies with 304,713 total utterances.

The pre-processing:

The dataset contains two files "Movie_conversation.txt" and "Movie_lines.txt". The Movie_conversation.txt file contains a list of unique identification numbers and this identification numbers corresponds to unique lines of the file Movie_lines.txt.

E.g. The Movie_conversation.txt contains a list such as [1, 2, 3, 4]. The Movie_lines.txt contains the sentences that corresponds to the unique ids i.e. :

1 = Hey hi!

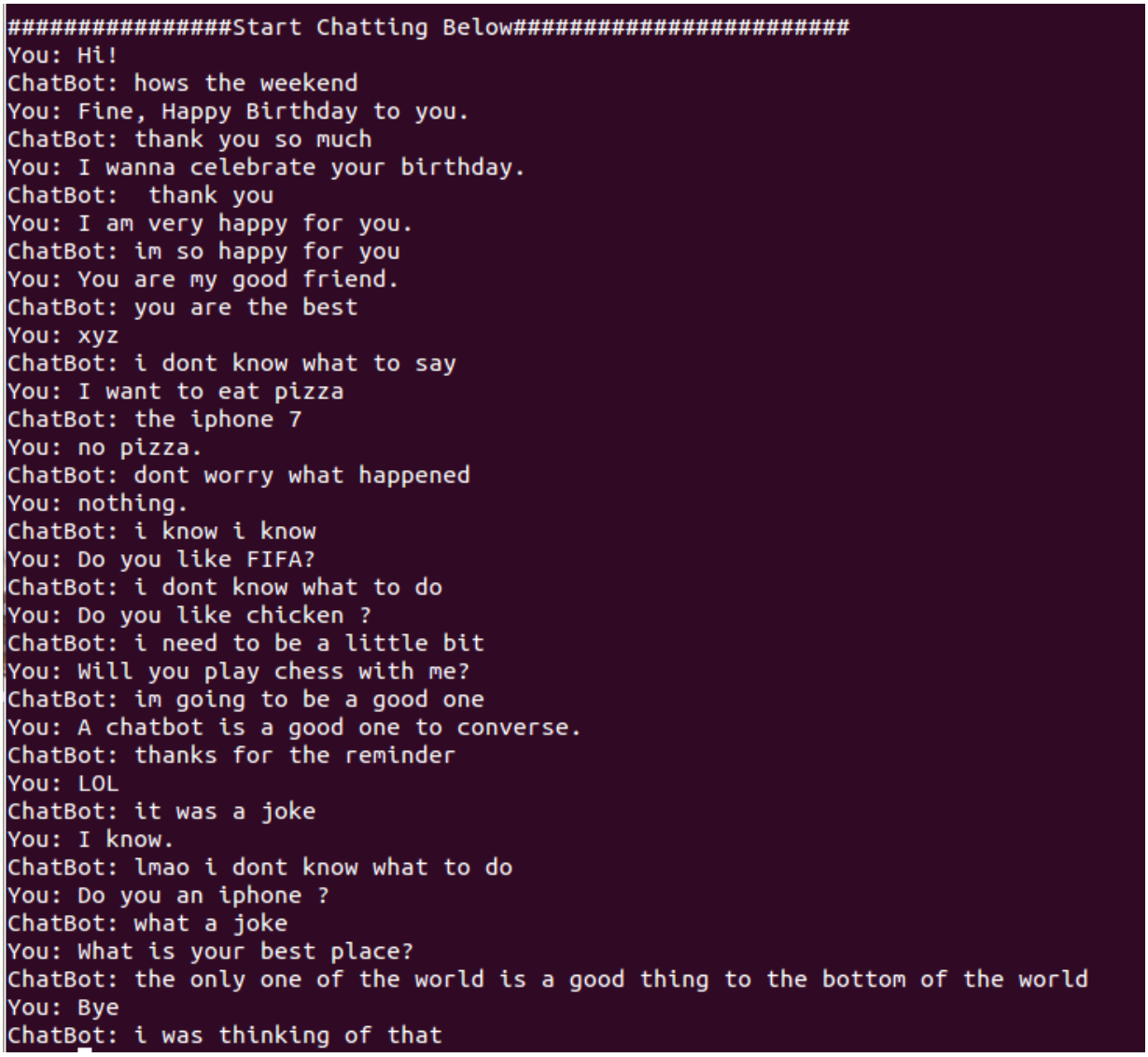
2 = Hi!

3 = How are you?

4 = I am fine.

The odd positions in the list of Movie_conversation.txt contains all the questions and even positions contains all answers, so I made two list a question list and an answer list, the question list contains all questions and answer list contains all answers. Then I made two dictionaries a word_to_int and an int_to_word. The word_to_int dictionary contains integers and their corresponding word as key. The int_to_word dictionary contains words and their corresponding integers as key. The word_to_int dictionary is used for encoding and int_to_word dictionary is used for decoding. I lowercase everything, removed all special characters and words having frequency less than twenty.

SAMPLE CONVERSATIONS



PACKAGES USED

- **NumPy:** NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
- **Tensorflow:** It is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

IMPROVEMENT

The bot's conversational ability is far from being satisfactory, and there are many ways we can improve the bot. Below are some of the improvements we can make:

- Training on multiple datasets: Bots are only as good as their data. If we play around with the bot, we'll see that the chatbot can't really hold normal conversations such as "how are you?", "what do you want to for lunch?", or "bye", and it's prone to saying dramatic things like "what about the gun?", "you're in trouble", "you're in love". The bot also tends to answer with questions. This makes sense, since Hollywood screenwriters need dramatic details and questions to advance the plot. However, training on movie dialogues makes our bot sound like a dumb version of the Terminator. To make the bot more realistic, we can try training your bot on other datasets. Here are some of the possible datasets: Twitter chat log (courtesy of Marsan Ma), Every publicly available Reddit comments (1TB of data!), our own conversations (chat logs, text messages, email).
- Using more than just one utterance as the encoder: For the chatbot, the encoder is the last utterance, and the decoder is the response to that. We can see that this is problematic because we often have to use information from the previous utterances to construct an appropriate response. We can modify the model to be able to use more than one utterance as the encoder input. This will make our model more like a summarization model in which our encoder is longer than our decoder. To do this, we'll have to modify the padding lengths and some of the data processing code. It will take longer to train on longer inputs.
- Making our chatbot remember information from the previous conversation: Right now, if I tell the bot my name and ask what my name is right after, the bot will be unable to answer. This makes sense since we only use the last previous utterance as the input to predict the response without incorporating any previous information, however, this is unacceptable in real life conversation. What we can do is to save the previous conversations we have with that user and refer to them to extract information relevant to the current conversation.
- Creating a chatbot with personality: Right now, the chatbot is trained on the responses from thousands of characters, so we can expect the responses are rather erratic. It also can't answer to simple questions about personal information like "what's your name?" or "where are you from? " because those tokens are mostly unknown tokens due to the pre-processing phase that gets rid of rare words. We can change this by using one of the two approaches (or another, this is a very open field). Approach 1: At the decoder phase, inject consistent information about the bot such as name, age, hometown, current location, job. Approach 2: Use the decoder inputs from one character only. For example: your own Sheldon Cooper bot.

CONCLUSION

In this work, we presented an approach for Chatbot that has focused specifically on Chatbot design techniques. Chatbots are effective tools when it comes to education, IR, e-commerce, etc. The aim of chatbot designers should be: to build tools that can help people, facilitate their work, and their interaction with computers using natural language; but not to replace the human role totally, or imitate human conversation perfectly. The proposed solution has the benefits of requiring no hand-designed features and being applicable to various NLP domains without a need for modifications in the implementation.

Future Works: We can perform the classification of the documents to personalize the author of a given text. In order to well personalize the author, we intend to explore other dimensions apart from age and sex. We will also need to address the detection of the native language, the geographical data of the author.

REFERENCES

- Denny Britz, 2017 “Recurrent Neural Networks Tutorial – Introduction to RNNs”, Link: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
- Sebastian Ruder, “On word embeddings”, Link: http://ruder.io/word-embeddings-1/
- Surlyadeepan Ram, 2016 “Chatbots with Seq2Seq”, Link: http://suriyadeepan.github.io/2016-06-28-easy-seq2seq/
- Andrej Karpathy, 2015, “The Unreasonable Effectiveness of Recurrent Neural Networks”, Link: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Ilya Sutskever, Oriol Vinyals and Quoc V. Le, 2014, “Sequence to Sequence Learning with Neural Networks”, Link: https://arxiv.org/pdf/1409.3215.pdf
- Paper N-Gram based on Author Profiling for authorship attribution.