Advanced Machine Learning: Final Project Report

Aastha Shah, Reuel John, Nandini Agrawal

Abstract:

Our project deals with building a conversational chatbot. For this, we have implemented Luong attention in conjunction with a seq2seq model. We have used the Cornell Movie-Dialogs corpus as it is the closest large, free dataset that has somewhat of a conversational tone, and is a commonly used NLP conversational corpus. Our training dataset comprises 40000 pairs of inputs and responses. We have reached a training accuracy of between 70-80% in 30 epochs. Taking into account our data and the task, we also give an idea of why a metric like validation accuracy might not be the best way to evaluate the bot. Through this report, we will describe our approach, the difficulties we faced, the results we achieved and the ways we can use to further improve our model.

Main Objective:

Given an input, the bot must generate a reply after trying to understand the input and responding to it. We will call the input the ‘context’, and the output a ‘response’. Thus, given a particular context, the bot must interpret it and generate a response to that context. Our model is open-domain and conversational.

Background & Common Approaches:

Building a chatbot has been attempted using several different approaches, many of which have been turned into commercial software. On researching for a few days, we stumbled upon a few common approaches that papers have taken:

1. Retrieval based approach: This approach keeps a database of common contexts and targets and retrieves appropriate responses/ targets based on the context it receives.

   Why we *didn’t* opt for this: This is not in the spirit of machine learning - it is literally learning the training data, and does not generalize to test data.

2. Simple seq2seq approach: This approach creates an encoder and decoder consisting of RNN layers. It is a standard approach where the context is encoded and the hidden state is passed into the decoder to provide a starting point from where to generate the output.

Why we *didn't* opt for this: This approach does not account for inputs that are long in the training data. The decoder also does not have enough information about the entire input and the way it is written (sentence structure) because it only has access to the final hidden state, which is cumulative. As a result, various contexts might be close to each other in the hidden space, and could also result in very repetitive, conservative responses. It might also fail to understand sentence structure.

3. Seq2seq approach with GAN/ Generative Adversarial Bots: Here the generator part can be a seq2seq model and the discriminator part evaluates the quality of the target generated.

   Why we *didn't* opt for this: Upon research, we found a few possible complications with this approach. It seemed as if common approaches with GANs would use the decoder to sample discrete 'words' to build the target and feed that into the discriminator. Some sources mentioned that this sampling is not directly differentiable, thus adding complications with backpropagation. Though there have been workarounds, we found only very new possibilities, so we decided to stick to a tried-and-tested approach instead.

4. Seq2seq approach with attention mechanism: This approach uses an encoder-decoder type model with some tweaks. **This approach not only takes into account all the hidden states from the encoder into account, but also how much attention to pay to each hidden state.** Under normal circumstances without implementing attention, the decoder would only have the final cumulative hidden state. However, if the decoder knows which hidden state to look at, the path length of information is much shorter and easier to narrow down rather than looking at the entire network. This is done by assigning 'scores' to the encoder's hidden states. There were mainly two implementations of attention that we looked into namely Bahdanau attention and Luong attention. We will use a form of Luong attention for this.

   Why we finally opted for this: Given the difference the attention mechanism has made on problems like neural machine translation, we thought it would be worth a try to implement it in an open-conversation setting. It is also able to understand longer sentences better, which would help us train our data, a movie-dialogue based dataset, better. Giving attention to certain words helps create effective context vectors which go in as an input to the decoder producing better targets.

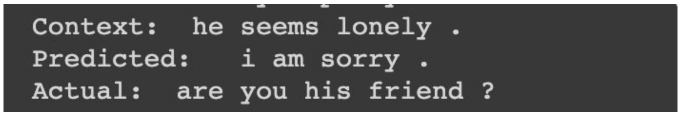
Dataset:

For this project, we made use of the Cornell Movie-Dialogs Corpus which is a dataset of a large number of movie lines and dialogues. Many of the corpuses we found were either restricted to a specific domain (like a Q&A dataset on Wikipedia entries, which might have a very sophisticated vocabulary) or transcripts of spoken conversations (which would have a lot of filler words like “uh” and “um”). The Cornell Movie corpus seemed general enough to pass off as conversation and also had a wide enough vocabulary so as not to restrict ourselves to a small set of words.

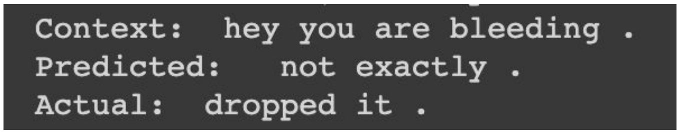
Because of computational requirements, we were only able to use 40,000 pairs of contexts and responses. More data would require a lot more RAM than we could use on Google Colaboratory.

A Note on Evaluation of Conversational Models:

We found that though our training accuracy was high, we were not able to get a high validation accuracy (it maxed somewhere around 40%). We do realize it might seem like we are overfitting, however, upon some research and thought, we are also unsure of whether measuring accuracy in the way these models usually do might be correct. This is because though we are not using a traditional generative model, it is definitely semi-generative and there is no correct answer because we are not trying to emulate a specific probability distribution. One particular input could have multiple responses, and because the dataset we have used includes movie-dialogs, no matter how well the model has trained on the rest of the movie dialogs, it is unlikely that we get the exact dialog that follows it due to the non-IID nature of dialogs. In fact, even if somehow the model is able to predict parts of the next movie dialog, it may predict it only partially, or order the same words in different ways. The validation data is also not a reflection of the type of conversations we are concerned with which are more generic and conversational. **For example, if the input we give is “when is it?” and a similar question in the validation data has the target is “it seems to be scheduled for tonight”, and our bot replies “it is tomorrow”, it will give a high loss despite being perfectly acceptable for our purpose.** On printing the actual targets of a few validation samples we confirmed our suspicion as well, we see here that the prediction from the model is adequate, but would give a high loss if compared to the target sequence:



```
Context:  he seems lonely .
Predicted:   i am sorry .
Actual:  are you his friend ?
```



```
Context:  hey you are bleeding .
Predicted:   not exactly .
Actual:  dropped it .
```

We also looked into BLEU scores and other means that are used to evaluate neural machine-translation tasks. However, because in that task a particular input *does* require similar outputs (if not exactly the same), we found that it was not the best way to evaluate performance because we would have to provide reference texts. Even if we use the target text (the actual dialog) as the reference text, as mentioned in the example above, it might be very specific. So, even if our model predicts something reasonable the BLEU score will be bad as it doesn't pay more weight to meaning or sentence structure.

Because this is an open-domain conversational bot, **we do not believe that the validation accuracy obtained is a good way to measure its performance.** To stay within the scope of the paper, perhaps attention should be given to **the way the bot interprets inputs (whether it's a question, about what etc.) and the sentence structure of the outputs and whether there is proper relevance to the input.**

Our Approach:

Preprocessing:

The Cornell movie dataset is a collection of various files. For our purposes we only used two files namely, the movie_conversations.txt and movie_lines.txt file. Both of these files are used in conjunction with each other to extract conversations out of the dataset.

movie_lines.txt:

This is the main file that contains all the conversations between different characters in different films. It mainly consists of four different fields, namely:

- lineID
- characterID
- movieID
- Character name
- Dialogue text

movie_conversations.txt:

This file is a supplement to the aforementioned file. It contains:

- characterID of both characters involved in the conversation
- movieID
- The lineIDs of the conversation taking place

For our purpose we needed to extract the conversations from the files. Conversations between characters were spread out throughout the file and hence by using the movie_conversations.txt file we were able to narrow down what conversations were related to each other and thus establish a chronological sequence. By ensuring that the conversations we extracted "make sense", we ignored the movieID and characterID fields as they were of no use to us.

We then separated each conversation into two lists which were "questions" and "responses"; i.e for every question we had a corresponding response. For example, take the following conversation:

Jack: "Hey Jenny, I'm back!"
Jenny: "Jack! So soon?"
Jack: "Yeah my meeting finished early"

This conversation can be split into two question and response segments:

question[i]="Hey Jenny, I'm back!"
response[i]="Jack! So soon?"

question[i+1]="Jack! So soon?"
response[i+1]= ""Yeah my meeting finished early""

We then shuffled both the lists while maintaining parity between each question and response.

Our next step was to "clean" the data. We did this by expanding contractions and eliminating special symbols. We also converted all the text to lowercase. This was done to ensure as much uniformity as possible and to reduce as much "noise" in the data as possible. Punctuations like fullstops, commas, question marks and exclamation marks were retained because we found them to be useful to the learning process in terms of sentence structure and intent.

Before:
Hey Jenny, I'm back!

After:
hey jenny, i am back!

Consequently in our aim to make our data more uniform for training, we eliminated conversations whose question or response had a length of minimum 2 words and maximum 18 words. We experimented with the maximum length, ranging from 15-20 and found that on this dataset it is estimated that conversations that span upto 18 words cover 85% of the dataset. For every word increased there was a negligible amount of the dataset that was included, indicating some outliers. We also had issues with colab running out of memory, hence 18 was a good balance. To maintain the sequence length to 18, we also had to pad sequences that weren't long enough with 0s, to ensure the shapes of the inputs to the model were uniform. We thought it might be confusing for the model, but we also found that words between length 12-15 made a sizable chunk of the data, hence 18 worked out fairly well. Small limits also affected results, as seen below.

We then created a vocabulary by counting the number of word occurences in the cleaned data that we had. Some words that aren't used very often are categorized as RARE words. We set a threshold of 10 for the number of occurrences any given word has to pass to be not classified as a RARE word. These words were classified as <UNK> ("unknown") words. The index of these words were assigned accordingly. (The number 10 was to keep <UNK> words to 5-7% of the total vocabulary, as conventionally done, changing it often gave inadequate results, as shown later.)

We then added the <START> token to decoder inputs (i.e., the 'response') to denote the start of the sentence. Using this start, the decoder would start predicting words using information from the encoder (that extracts information from the 'context'). Finally all the words were encoded to indexes (numbers). By giving words a numerical value, we construct a format that is easier to pass to the model as input and at the same time calculate various other statistics such as loss and accuracy.

Once we split our data into training and validation inputs (context) and outputs (response), when we tried using a unidirectional LSTM for the encoder, we inverted the training and validation *inputs* (contexts). Based on a Google neural machine translation paper, this facilitated learning. They speculated that because without this, each word in the context is far from its corresponding target value, resulting in a large time-lag. Reversing the questions does not change the average time lag, but brings the first few words much closer together to their corresponding outputs. We found that it did improve our results, however was not required if we used a Bidirectional LSTM in the encoder.

Architecture:

The main architecture of this model is a sequence-to-sequence (or seq2seq) architecture combined with attention layers, as explained below.

Input Layer: Our input is of shape (BATCH_SIZE, MAX_SENTENCE_LEN). Here, each sentence is in the form of a vector of size MAX_SENTENCE_LEN holding numerical encodings of words from the word_to_idx{} mappings we created.

E.g.,
For sentence = "I have a cat" and MAX_SENTENCE_LEN = 5,
The input will be [53, 24, 9, 4, 0] assuming {'i' : 53, 'have' : 24, 'a' : 9, 'cat' : 4} with 0 being the encoding for padding sequences.

For the decoder input, the first word will always be "<START>", thus the first value will be the encoding of "<START>"

Embedding Layer: Embedding layers in Keras give a denser representation of matrices that map words in a sentence to a vector-representation in a small dimension. These vector representations capture information on sentence structures, and how words in a text are related to each other. Algebraic operations can also be performed on these e.g., a vector for "blue" would be closer to the vector for "red" than the vector for "wardrobe". Since vocabulary sizes can often be very large for NLP problems, an embedding layer captures this information in fewer dimensions.

In our model we embed a vocabulary of thousands to 128 dimensions. This reduces a vector of length in thousands to a 128 length vector. Both encoder and decoder inputs are embedded using embedding layers.

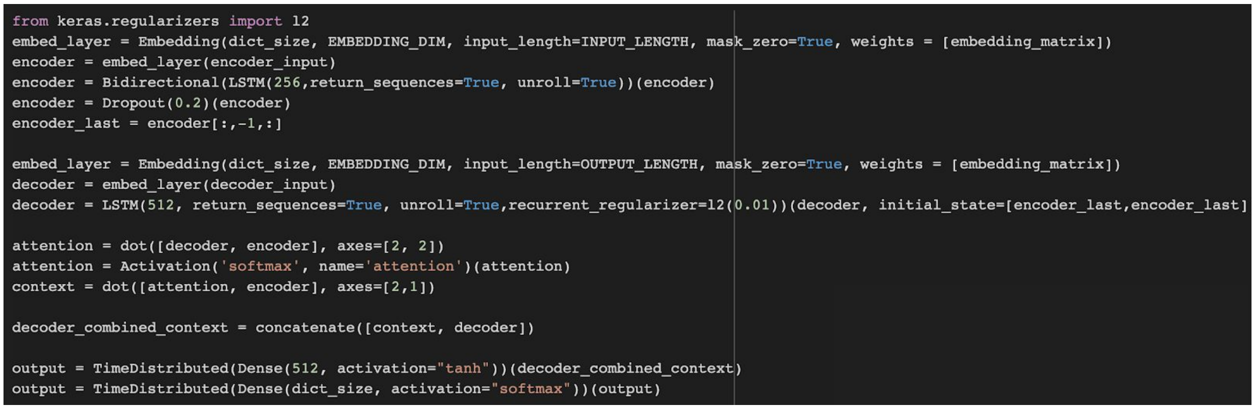
We also tried using pre-trained embedding matrices like GloVe representations. This does the same thing as described, but having been trained, it has word dependencies already established (e.g., "alps" and "Switzerland" would be related already even if they aren't in our dataset). We used the GloVe-Twitter representations for our model.

Encoder: Our encoder consists of a simple bidirectional LSTM layer with 256 nodes. The advantage of a bidirectional layer is that it will compute weights for both the regular input sequence as well as an inverted input sequence as discussed in the preprocessing section. Output from this is passed through a Dropout layer of value 0.2 to reduce overfitting.

Attention Layer (dot-product): We use global dot-product attention for this task, as expanded on by Luong et. al. The attention mechanism is implemented using simple Keras layers like dot and concatenate. The idea here is to include information from the encoder's hidden states into the decoder's input, so that it has more information on the input than just included in the encoder's output (output of the last hidden neuron in the encoder).

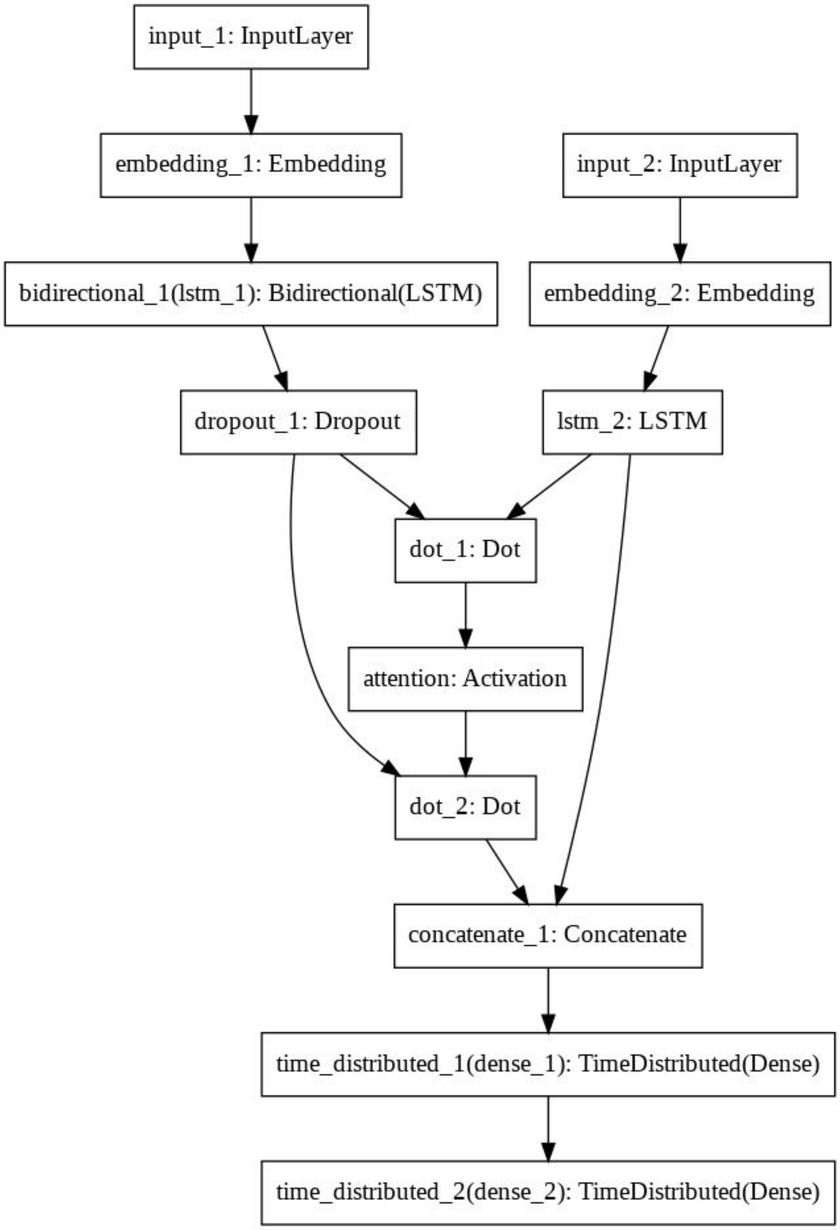
In this attention mechanism, first, the encoder's hidden states (context's hidden state) and the decoder's output are combined using a dot product layer. This produces an 'alignment score' to each encoder hidden state so that the decoder knows which encoder-hidden-state to pay more attention to. This alignment score is passed through a softmax activation layer to obtain an 'attention distribution'. After this, another dot product is computed between the encoder's hidden states and this attention distribution (from the softmax layer) to obtain the 'annotation vector' or 'context vector'. This will be concatenated with the decoder output as explained below. (A detailed diagram of the attention layer can be found in the appendix).

Decoder: The decoder consists of a regular LSTM layer with 512 nodes. The initial hidden states of the LSTM layer will be the states that are output by the LSTM layer in the encoder. The output from this layer is concatenated with the context vector obtained above. We then pass this through a tanh activation layer, followed by a softmax layer to obtain a probability distribution on the vocabulary. The word with the highest probability is chosen as the output (this will be added to the decoder input in the next time step). The final model looks like this:



```
from keras.regularizers import l2
embed_layer = Embedding(dict_size, EMBEDDING_DIM, input_length=INPUT_LENGTH, mask_zero=True, weights = [embedding_matrix])
encoder = embed_layer(encoder_input)
encoder = Bidirectional(LSTM(256,return_sequences=True, unroll=True))(encoder)
encoder = Dropout(0.2)(encoder)
encoder_last = encoder[:,-1,:]

embed_layer = Embedding(dict_size, EMBEDDING_DIM, input_length=OUTPUT_LENGTH, mask_zero=True, weights = [embedding_matrix])
decoder = embed_layer(decoder_input)
decoder = LSTM(512, return_sequences=True, unroll=True,recurrent_regularizer=l2(0.01))(decoder, initial_state=[encoder_last,encoder_last]

attention = dot([decoder, encoder], axes=[2, 2])
attention = Activation('softmax', name='attention')(attention)
context = dot([attention, encoder], axes=[2,1])

decoder_combined_context = concatenate([context, decoder])

output = TimeDistributed(Dense(512, activation="tanh"))(decoder_combined_context)
output = TimeDistributed(Dense(dict_size, activation="softmax"))(output)
```

(contd.)


input_1: InputLayer
embedding_1: Embedding
input_2: InputLayer
bidirectional_1(lstm_1): Bidirectional(LSTM)
embedding_2: Embedding
dropout_1: Dropout
lstm_2: LSTM
dot_1: Dot
attention: Activation
dot_2: Dot
concatenate_1: Concatenate
time_distributed_1(dense_1): TimeDistributed(Dense)
time_distributed_2(dense_2): TimeDistributed(Dense)

Training:

The entire context is passed to the encoder (to derive information on what it's about, sentence structure etc.), and the target is passed to the decoder, with <START> in the beginning. The "label" will be the target itself, but without the <START>. Hence, the output from the word following <START> as the decoder input would be measured against the *actual* first word in the target which is passed as the "label". For example:

If the encoder input is "Do you have a cat?", the decoder input would be "<START> Yes I do", and the label would be "Yes I do". Hence, at timestep t=0, for <START>, whatever the output of the model is, the loss will be computed against “Yes” because that is what the output should be. Here, the *actual* target is passed as input instead of the predictions (i.e., even if the model predicted “Mountain” after <START>, the decoder input for timestep t=1 will be the actual target. This is called teacher’s forcing, and is a popular seq2seq approach for training.

During prediction (after training), the decoder input is initially only <START>, and as and when we get predictions of words, they are concatenated to the decoder input. E.g., if at timestep t=0 for some encoder input, if the decoder input <START> gives us “No”, at timestep t=1, the decoder input will be “<START> Yes”.

Loss Function: We use a categorical cross entropy function because this is in a way a multi-class classification problem over the entire vocabulary. During training, the “label” is the next word that should follow the current word, and the output is in the form of a softmax over the entire vocabulary, thus giving a probability distribution telling us which word would follow according to the model.

Optimizer: We use Adam that is somewhat of a combination of rmsprop and adagrad, and adjusts the learning rate efficiently and appropriately according to the slope. It is also a standard convention to start with adam. On trying a few alternatives we did not see any improvements, so adam seemed like a good option.

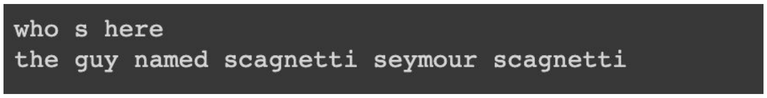
Difficulties Faced:

One problem we faced was finding a good dataset for the task. We wanted to try our best not to restrict ourselves to a particular domain because we wanted to try to create a more conversational chatbot. A lot of corpuses we found were often restricted to such a domain. We had also decided against the Cornell dataset because of its nature - being a movie-dialog dataset, it had a lot of phrases and words that were unique. They were used for entertainment purposes in films and not always repeated across movies, so they were also rare to find elsewhere in the dataset. We also found corpuses of chats and conversations on instant messaging, but they often included a lot of slang words and emoticons that we did not think were appropriate and/ or necessary. At the end we stuck to the Cornell dataset because it was still fairly conversational, not restricted to a domain, and not transcripted from regular conversations with filler words. We then had to find how to preprocess this data so as to make proper use of it.

Deciding the steps involved in preprocessing was difficult, because it was difficult to gauge what would help the model to learn and what would not. There were many questions involved - whether to pad sequences, to restrict length, to embed using pretrained models such as word2vec or GloVe, to remove rare words, if so how many etc. It was similar to tuning hyperparameters but often had much more drastic differences in the way the bot responded.

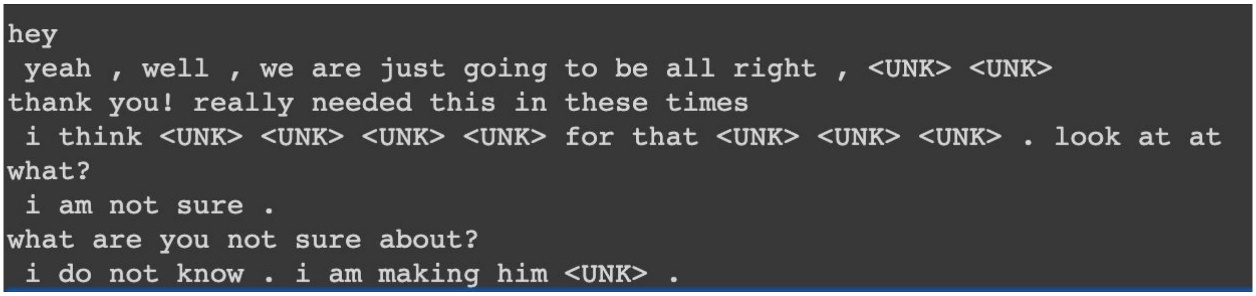
(Note: the response with the indent is from the bot.)

For instance when we did not remove rare words, we found that the vocabulary was too large, and a lot of words were only used once or twice in thousands of pairs. Because the dataset is a movie-dialog dataset, these words were not what we might use in regular conversation. We found that this heavily confused the model and it was not able to figure out the usage of such words in a sentence. E.g.,



```
who s here
the guy named scagnetti seymour scagnetti
```

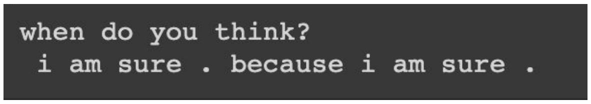
When we did use rare words and the number was not proportional (more than 10%), the bot was unable to respond properly. Results weren't adequate even when we played around with max sequence lengths and used shorter sequences like 15:



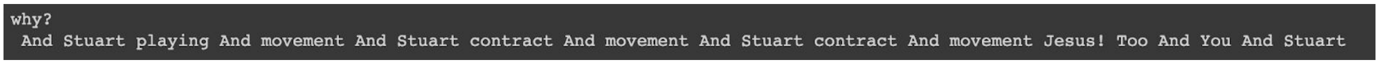
```
hey
 yeah , well , we are just going to be all right , <UNK> <UNK>
thank you! really needed this in these times
 i think <UNK> <UNK> <UNK> <UNK> for that <UNK> <UNK> <UNK> . look at at
what?
 i am not sure .
what are you not sure about?
 i do not know . i am making him <UNK> .
```

We also saw a drastic difference when we removed punctuations as opposed to when we didn't. We realized that including basic punctuations such as fullstops, commas, and question marks helped in understanding sentence structure. Thus, when we removed these, the bot would not be able to tell whether the input was a question or a comment (what what it was asking) and thus the responses were often strange.

We also faced difficulty in choosing the correct model to work with. We wanted to see if a simple seq2seq model would suffice, because of the good results obtained on character-level RNNs by Karpathy et. al. However, we found that a seq2seq model was unable to reach a sufficient training accuracy, probably because of both the nature and quantity of the data available. As mentioned before, the dataset had a lot of unique words and we were only able to use 40000-50000 samples due to computational requirements. This restricted the power we would get out of these seq2seq models. Even on appropriate preprocessing, we found the model would make many conservative predictions and often repeat the same words over and over again. The training accuracy also did not go beyond 30-40%. E.g.,



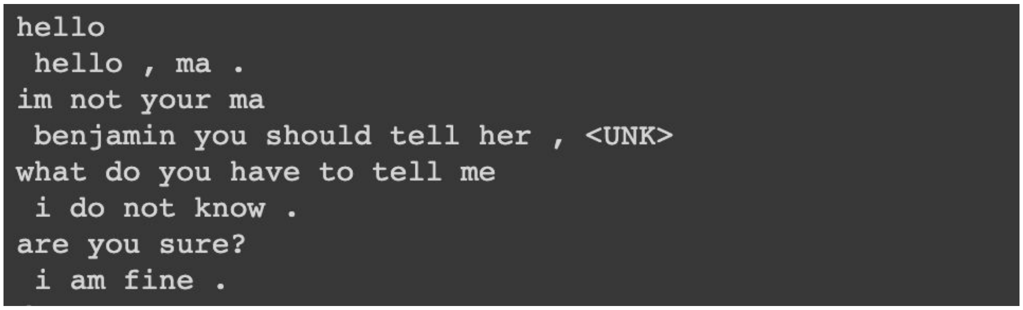
```
when do you think?
 i am sure . because i am sure .
```



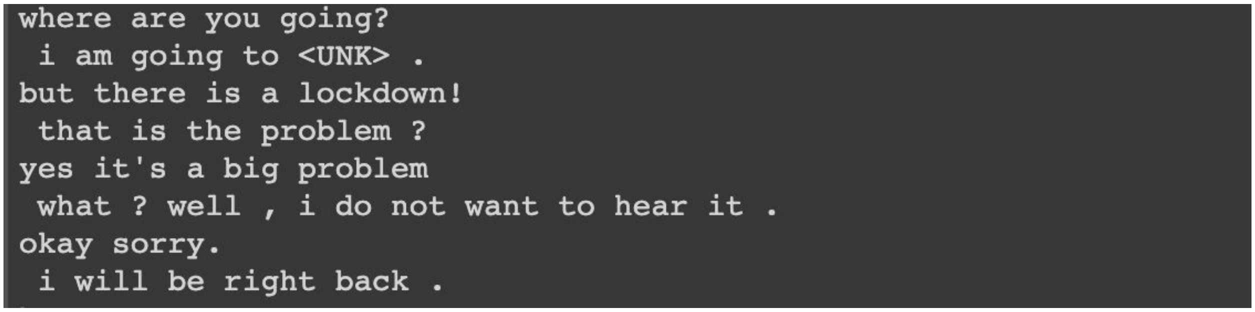
```
why?
 And Stuart playing And movement And Stuart contract And movement And Stuart contract And movement Jesus! Too And You And Stuart
```

<u>Final Results:</u>

With the final model, we got mostly decent responses with a few strange responses.
(The response with the indent is from the bot.)



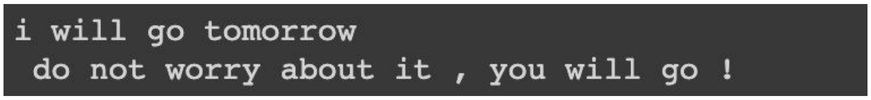
```
hello
 hello , ma .
im not your ma
 benjamin you should tell her , <UNK>
what do you have to tell me
 i do not know .
are you sure?
 i am fine .
```



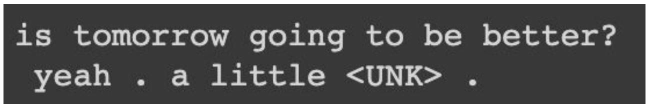
```
where are you going?
 i am going to <UNK> .
but there is a lockdown!
 that is the problem ?
yes it's a big problem
 what ? well , i do not want to hear it .
okay sorry.
 i will be right back .
```

Here we see that the bot is able to understand the intent and basic structure of what we are saying. For example, to the “where are you going?” question it could understand that it is a question (and about what) and respond to it. It could also interpret the sentence “but there is a lockdown!” had an exclamatory tone to it, and it responded appropriately. Though a few words are odd, like (like the “Benjamin”) we suspect that is due to the dataset. The bot is able to respond with appropriate sentence structure (like ending with punctuations and even commas after “well” and “hello”).

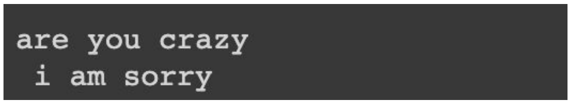
Some more responses:



```
i will go tomorrow
 do not worry about it , you will go !
```



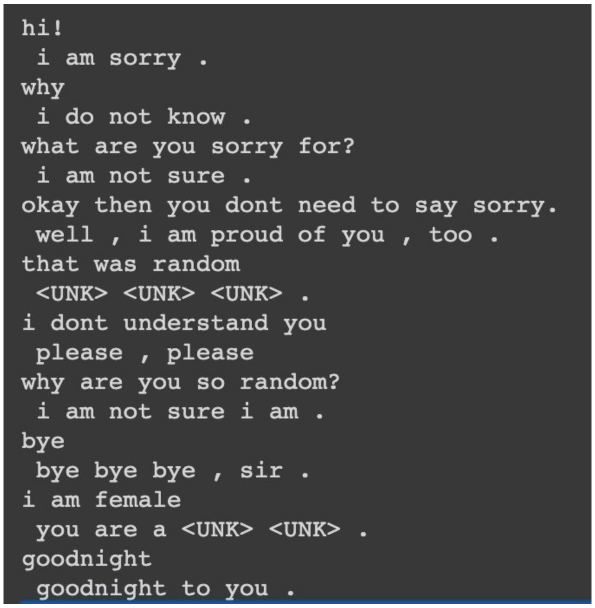
```
is tomorrow going to be better?
 yeah . a little <UNK> .
```



```
are you crazy
 i am sorry
```

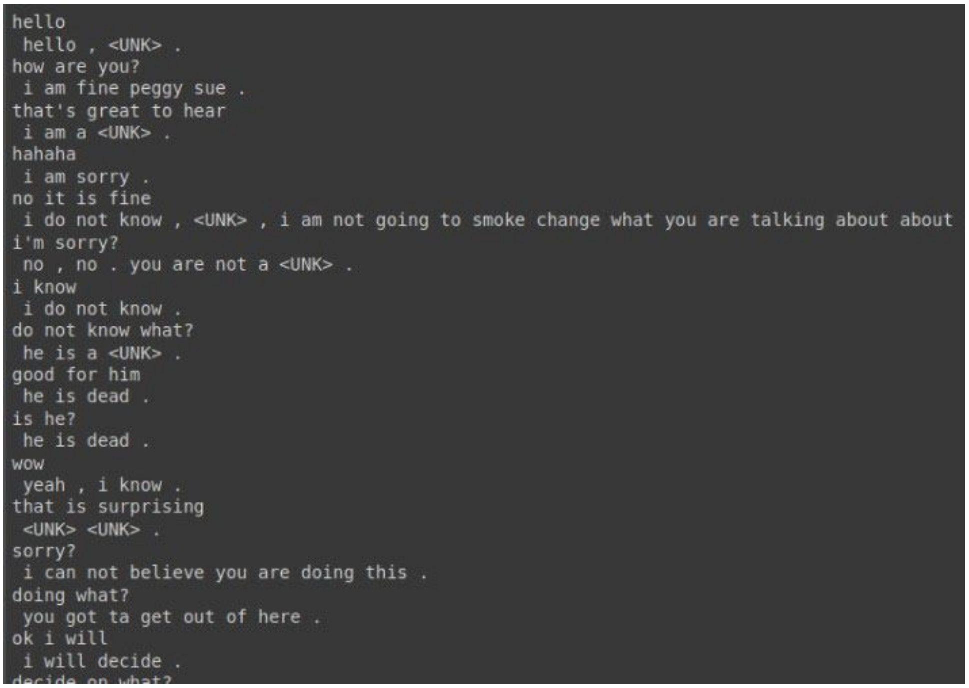
Here we see that besides sentence structure the bot is also able to understand intent and meanings of words well.

We also got some imperfect responses from the final model though those also largely included some understanding of intent and meaning of words.



```
hi!
 i am sorry .
why
 i do not know .
what are you sorry for?
 i am not sure .
okay then you dont need to say sorry.
 well , i am proud of you , too .
that was random
 <UNK> <UNK> <UNK> .
i dont understand you
 please , please
why are you so random?
 i am not sure i am .
bye
 bye bye bye , sir .
i am female
 you are a <UNK> <UNK> .
goodnight
 goodnight to you .
```

(contd.)



```
hello
 hello , <UNK> .
how are you?
 i am fine peggy sue .
that's great to hear
 i am a <UNK> .
hahaha
 i am sorry .
no it is fine
 i do not know , <UNK> , i am not going to smoke change what you are talking about about
i'm sorry?
 no , no . you are not a <UNK> .
i know
 i do not know .
do not know what?
 he is a <UNK> .
good for him
 he is dead .
is he?
 he is dead .
wow
 yeah , i know .
that is surprising
 <UNK> <UNK> .
sorry?
 i can not believe you are doing this .
doing what?
 you got ta get out of here .
ok i will
 i will decide .
decide on what?
```

Limitations/ What we could have done better:

- We would have liked to at least try out a GAN-based approach. Having a generative model as opposed to one that requires repeated input would have been an interesting task. However, due to possible complications with backpropagation (as mentioned above), something that we were not entirely sure on how to solve, would have prevented us from making full use of the approach.
- We would have definitely liked to use more of better data. If we had enough of more generic conversation data, the bot could have been even more conversational. One of the datasets we liked was the rDany chats but it was very small and had Spanish mixed with English words. We also think if we could have used the entire Cornell dataset we would have seen better results.
- We would have liked to incorporate the input we give into future responses. For instance, if we were to start the conversation with "Hello, I'm <name>" we would have liked to try for the bot to remember that data and use it in its responses like "Hello <name>"
- We would have liked to find a mechanism to evaluate our model better.

Conclusion & Learning:

During this project we had to learn an entirely new territory of NLP and RNNs/ temporal models. The trial-and-error with preprocessing gave us a lot of insight into how RNNs tackle text data and found it to be considerably more tricky than regular CNNs and image data. We found that this requires a lot of subjective thinking based on the data we have and the output we require. It was our first hands-on experience with a semi-generative model. So, because evaluating something like this is difficult, it also gave insight into how we can judge the output of RNNs without knowing what the correct answer *should* be.

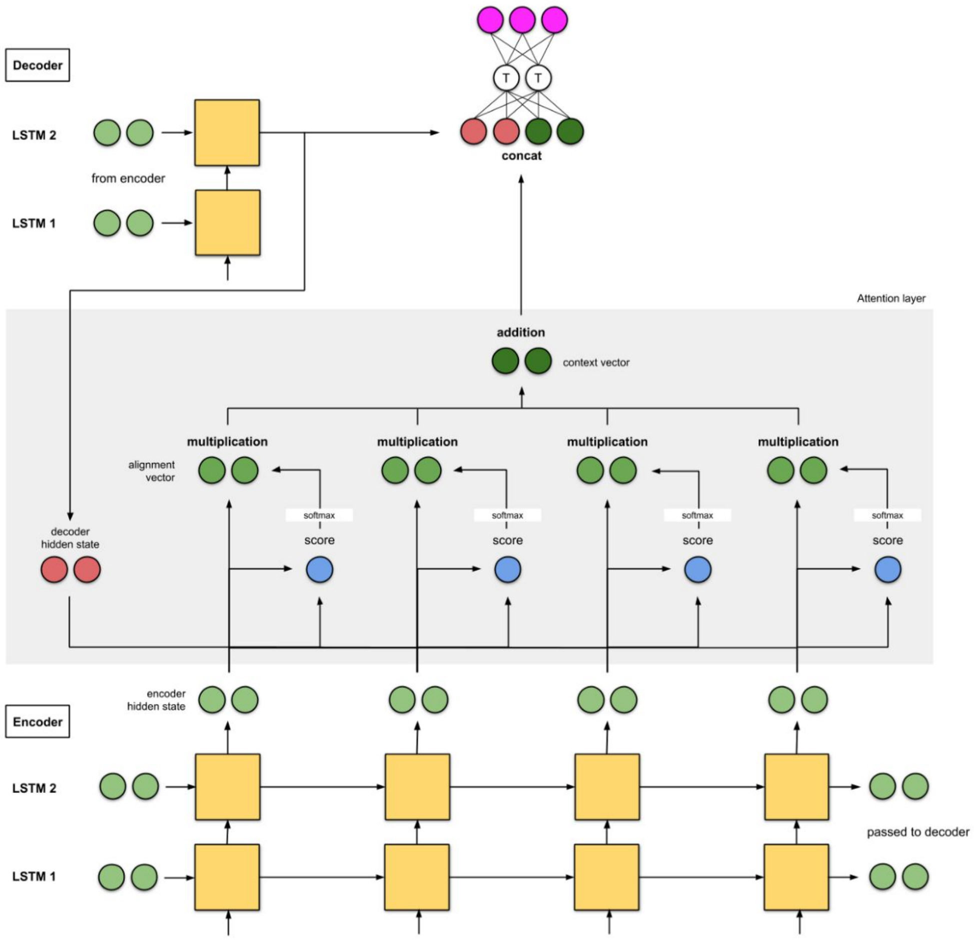
References:

During our understanding and implementation of the project we consulted the following sources:

- Professor Kothari's slide deck (10) on Attention on Classroom
- Effective Approaches to Attention-based Neural Machine Translation, Luong et. al. https://arxiv.org/pdf/1508.04025.pdf
- Neural Machine Translation with Attention Blogpost https://machinetalk.org/2019/03/29/neural-machine-translation-with-attention-mechanism
- https://arxiv.org/pdf/1409.3215.pdf

Appendix:

Here is a detailed diagram of the attention mechanism from a blogpost we consulted for help. (Source: https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#7eef)


Decoder
LSTM 2
from encoder
LSTM 1
T
T
concat
Attention layer
addition
context vector
multiplication
multiplication
multiplication
multiplication
alignment vector
softmax
softmax
softmax
softmax
score
score
score
score
decoder hidden state
encoder hidden state
Encoder
LSTM 2
LSTM 1
passed to decoder