MSCI 641 Project: Conversational Chatbot Models and their Comparison

Khalid Shaikh Qamar Ali Hasan
Student ID: 20801907
Master in Management Science
University of Waterloo
Email: kshaikhq@uwaterloo.ca

Dina Haji Zeynaly Biooky
Student ID: 20860748
Master in Management Science
University of Waterloo
Email: dhajizey@uwaterloo.ca

Abstract

A conversational agent (chatbot) is a machine-learned dialogue model designed to simulate human conversations. In this study we will start by understanding what neural network is and how Recurrent Neural Network (RNN) and Sequence-to-Sequence (Seq2Seq) model are related to it. Then, we will talk about ways through which RNN and Seq2Seq can be improved such as by adding LSTMs and Attention mechanism. Finally, we will talk about a more developed model such as the Transformer model for chatbots. In this project, we will use the Cornell Movie-Dialogs Corpus to train our models. We will build Seq2Seq LSTM model, Seq2Seq LSTM model with Attention mechanism, and the Transformer model. Finally, we aim at comparing the performance of these dialogue models using the Bleu score and human judgement metrics to determine which model would be a better suggestion.

1. Introduction:

Chatbot is a machine learning software that mimics human conversations and responds in natural language. These conversational agents or dialogue models have become popular among researchers, and public and private businesses since they are ubiquitous, automate quotidian tasks and reduces on staff required. In current times, due to widespread usage of smartphones and social media applications, dialogue models are being deployed in a wide range of organizations finding usage from customer service, product suggestion and inquiry to being a personal assistant.

2. Related Work:

There have been plenty of experiments on machine learned software that uses Seq2Seq or Transformer models and try to make some comparison among different models and datasets.

Sojasingarayar in her article, "Seq2Seq AI Chatbot with Attention Mechanism", implemented encoder-decoder attention mechanism architecture using Recurrent Neural Network with LSTM cells to create chatbots. She compared this model with a baseline seq2seq model. In her research[1] she used Cornell movie subtitle corpus, but they found that this dataset must be improved further or more attention on training parameters needed to get better results. Training process demands higher processing power and configured computing machine. It is also cumbersome to find right hyper parameters to optimize our model. She argued that developing a generic chatbot is really challenging. The used model is basically for machine translation, so histories of earlier conversations are not considered. This might be another reason of model poor performance.

Luong, Pham, and Manning in, "Effective Approaches to Attention-based Neural Machine Translation", use attentional mechanism in translation tasks between English and German in both directions using global and local approaches[5]. The global approach is when the model considers all the hidden states of the encoder for deriving the context vector. This approach must attend to all words on the source side for each target word, which is expensive and can potentially render it impractical to translate longer sequences, e.g., paragraphs or documents. To address this deficiency, they propose a local attentional mechanism that chooses to focus only on a small subset of the source positions per target word. Local attention yields large gains of up to 5.0 BLEU over non-attentional models which already incorporate known techniques such as dropout.

Vaswani and his colleagues in, "Attention Is All You Need", introduced the idea of Transformer for the first time[3]. Transformer is the first sequence transduction model based entirely on attention, replacing the recurrent layers most used in encoder-decoder architectures with multi-headed self-attention. They did their experiments on two machine translation tasks: English-to-German and English-to-French translation tasks. Their model becomes superior in quality while being more parallelizable and requiring significantly less time to train.

Kristian in, "Deep Learning Based Chatbot Models", focuses on adapting the very recent Transformer model to the chatbot domain and compare it with traditional base model[4]. This experiment is done by

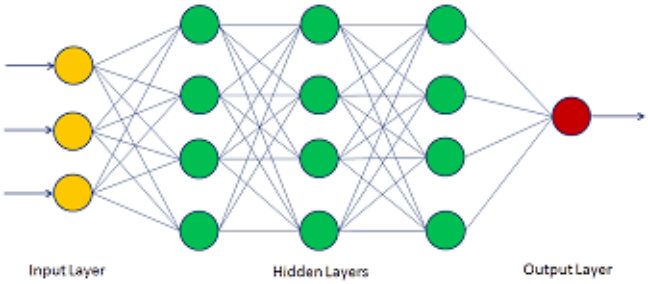
training the Transformer model on two different dialog datasets: Cornell Movie and Open Subtitles. The performance of the trainings is analyzed with the help of automatic evaluation metrics, Perplexity and Bleu scores, and by comparing output responses for a set of source utterances. Cornell results are lower than Open Subtitles' and surprisingly Transformer model acts worse than the standard RNN based seq2seq model for the task of conversational modeling.

3. Approach:

Chatbot is trained on existing conversations and later responses to a given input. In the past, methods for constructing chatbot architectures have relied on hand-written rules and templates or simple statistical methods. With the advent of deep learning, these models were quickly replaced by end-to-end trainable neural networks to get intelligent response.

3.1. How Neural Network helps?

A *Neural Network* is a series of algorithms that tries to recognize patterns and features in a set of data through a process that impersonates the human brain. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. We train the model on some known inputs and outputs. Then we give new and unseen sentences to the model that has learned its required weights during training in our applications.



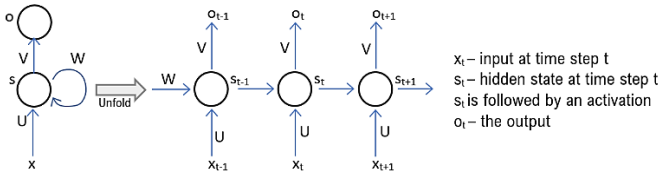
Although neural network could help humans in lots of problems, they are not enough for timeseries analysis. For instance, in a sentence like "*She* spoke to *her* husband", in order to predict "*her*" we need to rely on more than our current input: "*to*"; "*she*" is also required to be captured. Therefore, our neural network must have a memory on what has happened earlier to make a proper prediction in a sequential data type such as text generation (machine translation or conversational models). This is where the idea of Recurrent Neural Network flourished.

3.2. What is RNN?

RNNs (*Recurrent Neural Networks*) are multiple copy of the same neural network that receive input x_i at different time steps as well as its previous hidden states s_{i-1} and generates a hidden state s_i by

$$s_i = f(W * s_{i-1} + U * x_i)$$

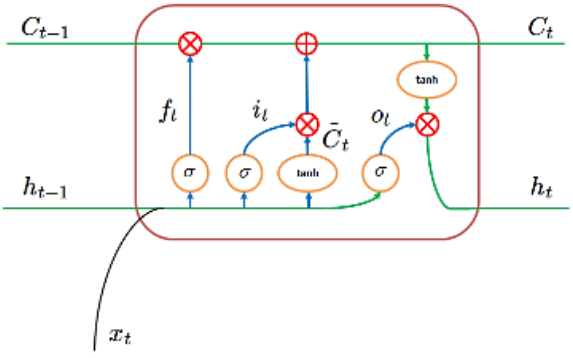
where W and U are matrices containing the learned weights (parameters) of the network, and f is a nonlinear activation function which can be the hyperbolic tangent function for example.



These previous hidden states that RNN at each time step passes to itself augment the network to remember what has been calculated so far and to use all sequential information (not only current word) for next word prediction. The RNN is unfolded into a full neural network by the number of timesteps. The number of timesteps depends on the number of x in a sequence length. Unfortunately, as the number of steps of the unfolding increase, RNN faces vanishing gradient problem because of continuous multiplication over the same weight matrix, which means no real learning is done. As a result of which, LSTMs were developed to combat the problem of long-term dependencies that vanilla RNNs face.

3.3. Why LSTM?

LSTM (*Long Short-Term Memory*) network is a special type of RNN that has added memory processing in each of its neurons. Each neuron of LSTM takes old memory C_{t-1} as well as old hidden state h_{t-1} and input x_t, and passes new memory C_t as well as new hidden state h_t. New generated memory $\tilde{C}_t$ and output o_t are also generated in each neuron by old hidden state h_{t-1} and input x_t.



In each LSTM step, some parts of generated memory ($\tilde{C}_t$) and some parts of old memory (C_{t-1}) should be forgotten and are not required to be added to new memory C_t. This new memory is generated by

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

where f_t defines how much of old memory C_{t-1} and i_t defines how much of generated memory ($\tilde{C}_t$)

should be forgotten. f_t and i_t are some values between 0 and 1 where, 0 means to forget all and 1 means to keep all and not forget any.

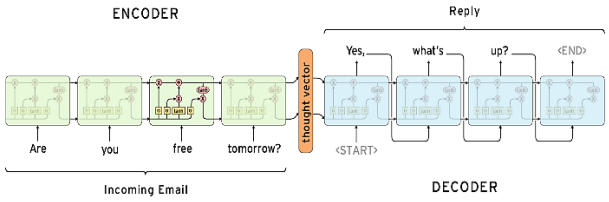
This added memory processing to LSTMs help system to remember information seen multiple steps ago and thus provide better results compared to simple RNN.

3.4. Seq2Seq Model

Sequence to Sequence model was initially developed for machine translation problems, which has become the most popular model for Dialogue Systems. It consists of two RNNs (In our case, we are using LSTM cells). First RNN that encode the encoded input sequence is called an *Encoder* and the second one that decode the encoded context vector and decoded input sequence into the target sequence is called a *Decoder*.

The encoder takes a sequence as input and processes one word at each time-step. Each hidden state in encoder is influenced by the previous hidden state, and the final hidden state is the summary of the sequence. The final state is called context vector or thought vector. Outputs of encoder are rejected and only context vector are used.

The decoder functioning is similar to an encoder, which generates another sequence one word at a time, but here, at first time-step, the decoder is influenced by encoder context vector which contains information about the entire input sequence to help the decoder make accurate predictions.



In training seq2seq model, we must define 3 parts carefully. Firstly, encoded input sequence (questions). Secondly, decoded input sequence, (answers with *START*). We must add *START* tag at the beginning of each decoded input sequence. This start and the context vector help model to learn to predict the first word of output. Finally, target sequence, (answers with *END*). The *END* tag at the end of the target sequence teaches the model to stop predicting, which otherwise would keep generating words. Another reason that we use target sequence is that in each time step of decoder is that we want to use Teacher forcing method i.e., we want to use actual value regardless of what word has been predicted in decoder. This method is used in training part where we know real value, and in testing part, model will use decoder output of previous step as its decoder input at each time-step.

Unfortunately in Seq2Seq model, we discard all the intermediate states of the encoder and use only its final state (context vector) to initialize the decoder. This technique works good for smaller sequences, however as the length of the sequence increases, it gets very difficult to summarize long sequences into a single vector. With the use of Attention, this problem is resolved largely.

3.5. Attention is the solution!

Attention is impressed by human mind. As human beings, we are quickly able to understand how to answer a question. Each word of our answers is inflenced by a specific word in the question. For instance, if someone asks us "what do you work?", we might answer them that "I work as a doctor". Our mind automatically chooses word "doctor" after "a" by paying more attention to "work" and " what" rather than to "do" and "you". These mappings between different parts of the decoder sequence and corresponding parts of the encoder sequence; however, is not that straight forward for artificial neural network.

Up until now, decoder only used its own previous hidden state h_{t-1} at each time step to predict next output and only used the information of final state of encoder at its first step. In attention approach; however, model can use encoder outputs information at each time; this approach augment model's prediction.

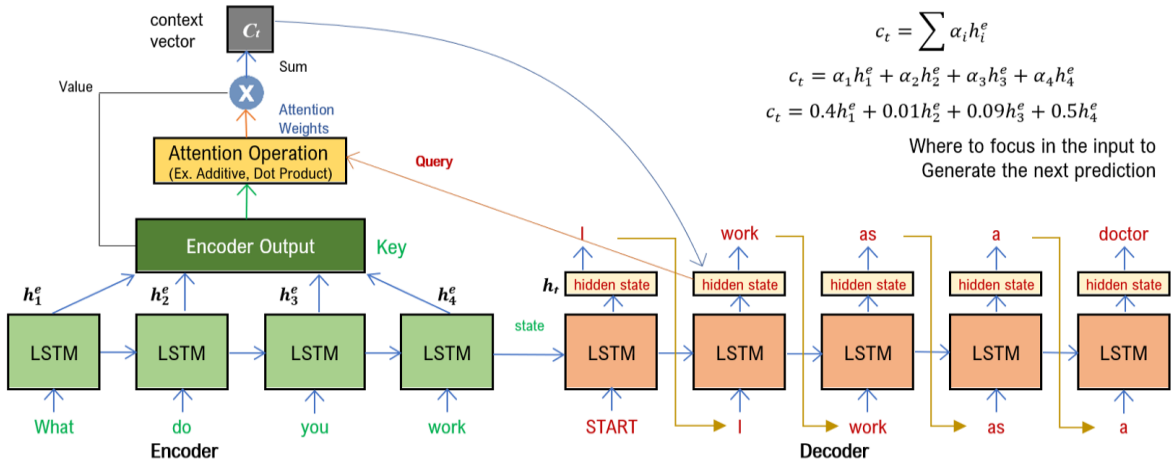
Construction of attention weights α_t, has several steps. First, we have to multiply encoder intermediate states (h_1^e, h_2^e, h_3^e, h_4^e) by decoder hidden state from previous step at each time step. For instance, in second time step of following diagram (for prediction of "work"), we have to multiply encoder intermediate states by $h_{t=1}$ (hidden state of first step)(In the first time step we use the final hidden state of encoder h_4^e as our h_t). This h_t has to be expanded to the equal number of encoder outputs. The result of this dot product shows us the most similarity between encoder output and following decoder output. Those values that are more similar to h_t will get higher values. We then use a softmax to make these similarity results to values between zero and one. These values are our attention weights. As we move towards the end of encoder, this h_t changes. As a result we get different weights and different context vectors at each time step.

$$\alpha_t = Softmax(h_t^{\intercal} W_a h_i^e)$$

Where $\alpha_t$is the attention weights, and W_a is used as fixation for different length of encoders and decoders. Attention mechanism is implemented on model by allowing decoder to use a specific context vector, not

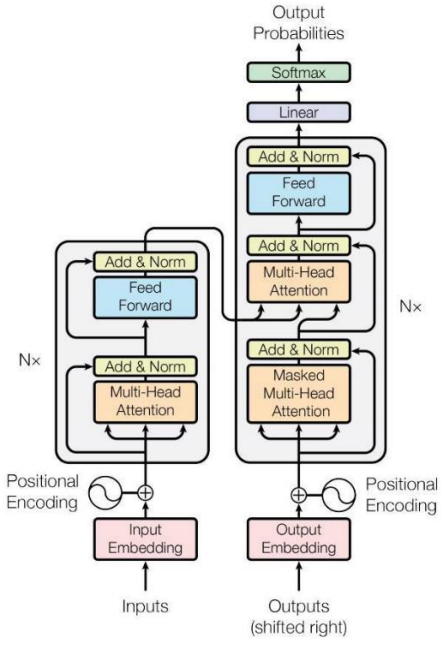
merely relying on its own decoder output, to predict next word at each time-step. These context vectors c_t are the sum of encoder intermediate states (h_1^e, h_2^e, h_3^e, h_4^e) that is multiplied by attention weights (α_1, α_2, α_3, α_4) as:

$$c_t = \sum \alpha_i h_i^e$$



3.6. Transformer Model

Introduction of Attention has incredibly improved seq2seq models. Consequently, it is ideated that attention mechanisms are powerful enough to achieve the performance of RNNs with attention without recurrent sequential processing. Thus, without being an RNN, Transformer uses an attention mechanism by processing all tokens simultaneously and calculating attention weights between them. Since Transformers do not rely on sequential processing, but on parallel processing, Transformers are able to train more efficiently on larger datasets with less computer memory usage.



Transformer - model architecture

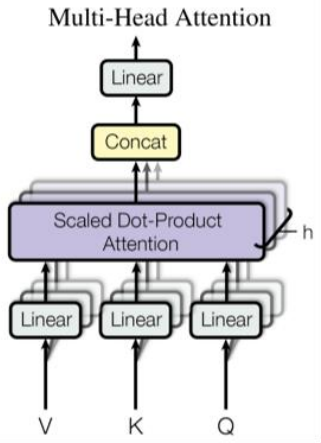
Like Seq2Seq, Transformer contains both encoder and decoder. Encoder maps an input sentence such as "Hi! How are you ?" into an abstarct continues representation that holds all the learned information of that input. The decoder then takes this continues representation as well as its previous output to generate a single output step by step. It stops working when *END* is reached. The final output sentence can be something like "I am fine *END*".

The encoder consists of two sub modules: One Multi-Headed Attention layer and one Feed Forward layer.

The decoder consists three sub modules: Two Multi-Headed Attention layers and one Feed Forward layer.

There are also residual connecting around each of sub modules followed by a layer of normalization.

First encoder takes positional information and embeddings of the input sequence. Since Transformer has no recurrent, positional encoding is the only key to help make sense of order of the sequence. The sum of positional vectors and word embedding vectors act as encoder input.



Multi-Headed Attention is a module that computes the attention weights for input and produce output vectors with information of how each word should attend to all other words in a sequence. Each module contains several heads that operate likewise. Each self attention (head) has to learn three weight matrices; the query weights W_q, the key weights W_k, and the value weights W_v . For each token i the input x_i is

multiplied with each of the three weight matrices to produce a corresponding vector.

$$Q_i = x_i * W_q, K_i = x_i * W_k \text{ and } V_i = x_i * W_v$$

Attention weights are calculated using the dot product between the query vector Q_i and key vector K_i. This dot product is then divided by the square root of the dimension of the key vectors $\sqrt{d_k}$ and passed through a softmax which normalizes the weights to sum to 1. This allows model to confident to what word to attend. The result is then multiplied by Value vector V_i. We can think of module acts like a search engine where you type query and it goes through a set of associated keys and comes up with best matched values. Multi-Headed outputs of the multi-head attention layer then concanitate before going to the feed-forward neural network layers.

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

The decoder functions like an encoder, but an additional attention mechanism is used, which draws relevant information from the encodings generated by the encoders. First Multiple-Head Attention in decoder also has a bit different processing than that of encoder. This module has a method that prevents module from paying attention to future words. For instance, "am" in "I am fine" should not pay attention to fine,it is a following word. Each word has to focus on itself and its previous words. To do this we use a Mask matrix with negative infinite values for future words and zero values for the rest. This matrix is added to score matrix (the dot product between the query vector Q_i and key vector K_i) before applying softmax. The softmax cause those negative infinite values get zero attention weights. The other differences of decoder is that the second Multi-Headed Attention of it takes its query and keys from the outputs of encoder, while taking its values from its own output. This helps decoder to find out which encoder input is relevent to give importance to.

3.7. Model Evaluation:

For model evaluation, we used Human judgment and Bleu score. However, Bleu score metrics does not necessarily mean that the dialogue model is effective but does provide a score to compare two sentences i.e., actual vs the predicted sentence.

Based on [10], Bleu is computed with precision, say $P(N, T, R)$ which is the geometric mean of n-gram precisions, and brevity Penalty, say $BP(T, R)$.

For a dialogue model, let T be the predicted sentence, and R be the actual sentence, then using the formula in the paper, we have

$$BLEU(N, T, R) = P(N, T, R) \times BP(T, R)$$

$$P(N, T, R) = \left(\prod_{n=1}^{N} p_n\right)^{1/N}$$

where, $p_n = m_n / l_n$

Here, m_n is the number of matched n-grams between predicted sentence T and corresponding actual sentence R. l_n is the total number of n-grams in the predicted sentence T.

$BP(T, R)$ punishes the score if the length of the actual sentence, $len(R)$, is greater than the length of the predicted sentence, $len(T)$. It is calculated as follows:

$$BP(T, R) = \min\left(1.0, exp\left(1 - \frac{len(R)}{len(T)}\right)\right)$$

To get better sentence level correlations with human judgment, we deploy smoothing function. This is to avoid getting zero Bleu score in cases where higher order n-gram precision of a sentence is zero.

4. Experiments:

The data used for our analysis was collected from 'https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html'. This corpus contains 220,579 conversations from raw movie scripts exchanged between 10,292 pairs of movie characters, stored in two files namely, 'movie_lines.txt' and 'movie_conversations.txt'.

4.1. Preprocessing

We created a dictionary to map each line's ID with its text, and a list of all the conversations' lines' IDs. We sorted the sentences into questions (inputs) and answers (outputs). Subsequently, we removed unnecessary characters and altered the format of words to clean the data. We filtered the questions and answers that were shorter than 1 word and longer than 25 words, thereby, filtering short and long sentences.

For the decoder mechanism, we added *START* and *END* tags for the answers, and *UNK* tag for the unknown tokens. We tokenized the sentences and to have an equal length for all input sequences, we padded extra zeros and made the sentences of same length. Although the model can process the entire corpus, we chose 15000 sample sentences from the corpus based on our computation power and split them into train and test data.

We created a dictionary for the vocabulary frequency. Further, we created a pair of dictionaries to provide a unique integer for each word i.e., dictionary to index each word in questions and answers respectively,

where key is index and value is word. Similarly, we create another pair of dictionaries which contain the reverse key-value pair of the above dictionaries respectively i.e., dictionary to get word given its index, where key is word and value is index. These dictionaries were used to encode the questions and decode the answer sequences.

4.2. Model Building

We built 3 dialogue models i.e., Seq2Seq LSTM Model, Seq2Seq LSTM Model with Attention mechanism, and Seq2Seq Transformer Model.

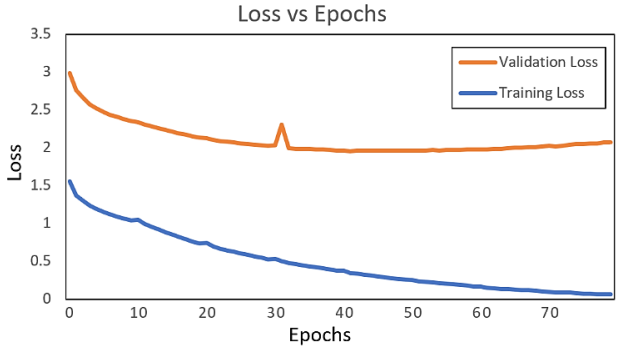
4.2.1. Seq2Seq LSTM Model

To proceed with model building, we created the data for training the encoder-decoder model. We used *batch_generator()* function to generate data in batches for reducing computation, increase learning and model performance. Then, we fed generated data to the *fit_generator()* function. To train the model, we created one-hot encoded data for:

Encoder inputs: a 2-dimensional array of shape (batch size, maximum input sentence length),
Decoder inputs: a 2-dimensional array of shape (batch size, maximum output sentence length), and
Decoder outputs: a 3-dimensional array of shape (batch size, maximum output sentence length, number of output tokens)

By setting the required parameters, we built the encoder and decoders, with embedding layer as the first hidden layer followed by the LSTM layer. After this, we defined and compiled the model using *rmsprop* optimizer and *categorical_crossentropy* loss function because of one-hot encoded vectors.

Using the Teacher forcing algorithm, we trained our model which enabled decoder to generate a single word at time-step *t+1* under the pretext of actual output at time-step *t* and internal state of the encoder. Using the generator function, we trained our model over different epoch values and saved the weights after every 10 epochs for future model evaluation.



To predict the chatbot output, questions were encoded into the hidden state and cell state of the LSTM, converted and fed as context vector into the decoder, where it predicted one sequence at a time until the *END* tag was reached. For the analysis, we tested our model on train and test data, and evaluated based on Bleu scores and human judgement.
For this model, Bleu score of 0.4368 was observed on the train data, and 0.0561 on the test data.

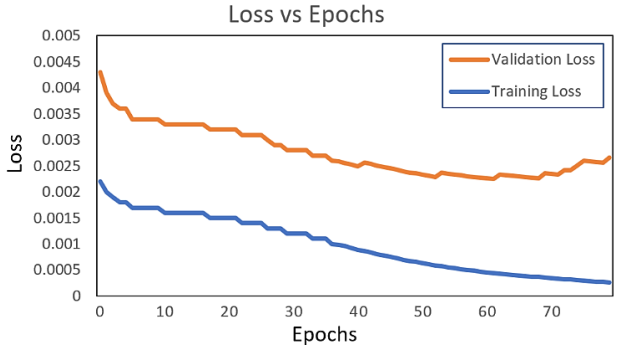
4.2.2. Seq2Seq LSTM Model with Attention

This model's approach is like the vanilla Seq2Seq model except for the Attention layer mechanism. Our goal was to determine the context vectors which would allow the model to generate the proper word using its current word and the corresponding context.

After creating the input data, encoder, and decoders, we carried out the dot product of the encoder output and corresponding decoder output, which was then passed through the *softmax* function to form the attention layer.

Subsequently, we performed the dot product of the attention weights and the encoder output and formulated the context vector. This was concatenated with the decoder output and passed through the *tanh* layer to form the attentional hidden state. The attentional vector was fed through the *softmax* layer to produce word sequence as the decoder output.

After this, we defined and compiled the model using *rmsprop* optimizer and *categorical_crossentropy* loss function. By setting the required parameters such as batch size, dimensions, and validation data, we fitted our model on the train data for 100 epochs and saved the model after every 10 epochs to perform model evaluation.



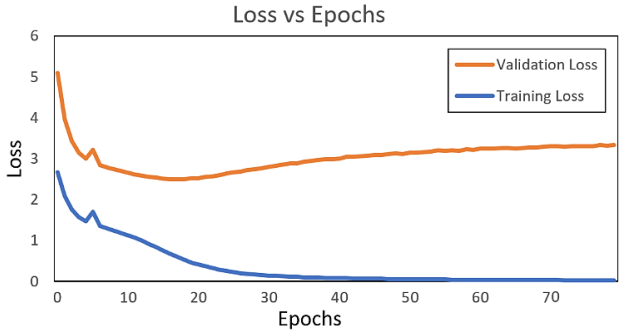
To perform prediction as a part of the inference function, we fetched the questions to the encoder to convert into the word encodings which was then predicted by the decoder by effectively utilizing the attention mechanism, and then, decoded into the word sequence. The *START* and *END* tags were clipped off to get the final output.

In this analysis, we tested our model using the test data, and evaluated using the Bleu Score metrics and human judgement evaluation. We observed a Bleu score of 0.2626 on train data, and 0.0415 on test data.

4.2.3. Seq2Seq Transformer Model

As discussed in the approach section, Transformer model is relatively a newer concept used in the conversational dialogue agents. For the sake of model comparison, we tried to implement the transformer model using the Functional API and Model sub-classing. Using the TensorFlow official documentation and model setup, we aimed at building the chatbot on our preprocessed data.

After importing the functions for the Scaled dot product attention, Multi-head Attention Layer, and positional encoding to create self-attention layers stacks, we imported the modules for the encoder, decoder, encoder and decoder layers, and the transformer. Subsequently, we trained the model on 100 epochs, and saved the model weights for every 10 epochs to perform model evaluation.



Based on our analysis, we tested the model on the train and test data, and evaluated the model using the Bleu score metrics and human judgement. We observed a Bleu score of 0.5296 on train data, and 0.2790 on test data.

4.3. Model Comparison

We used the same set of train and test data in all the three models for the model comparison. This made it easy to compare the three models, which have improved drastically in comparison to the baseline model where we observed repetitive patterns and noise in the predictions.

Since model evaluation for the dialogue models is still in research phase, we tried to evaluate our model based on Neural Machine Translation and text generation methods. Although human judgement is the most effective way to judge a dialogue model, but we require a method to quantify our model to compare them technically. Thus, we used the following evaluation techniques:

4.3.1. Bleu Score

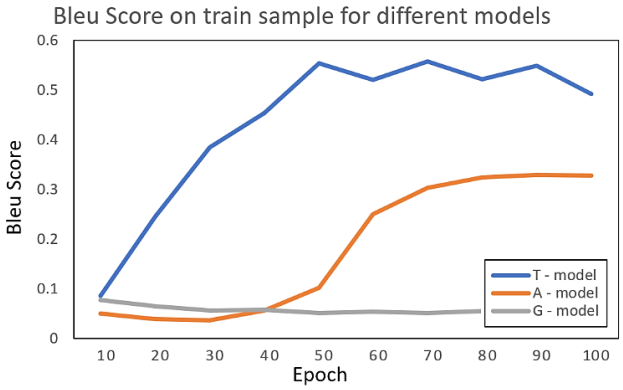
Based on our model development and analysis, we used NLTK's Bleu score library and observed the following Bleu scores against different epoch values for 100 set of train and test data:

Epoch	Transformer Model		Attention Mechanism		No Attention Mechanism	
	Train Bleu Score	Test Bleu Score	Train Bleu Score	Test Bleu Score	Train Bleu Score	Test Bleu Score
10	0.085	0.073	0.050	0.063	0.077	0.085
20	0.245	0.078	0.039	0.047	0.065	0.070
30	0.384	0.074	0.036	0.059	0.056	0.066
40	0.454	0.080	0.055	0.063	0.057	0.061
50	0.553	0.074	0.102	0.047	0.051	0.067
60	0.520	0.076	0.250	0.047	0.054	0.043
70	0.558	0.082	0.303	0.049	0.051	0.056
80	0.522	0.082	0.324	0.052	0.054	0.059
90	0.549	0.070	0.329	0.052	0.057	0.048
100	0.493	0.081	0.327	0.042	0.045	0.058

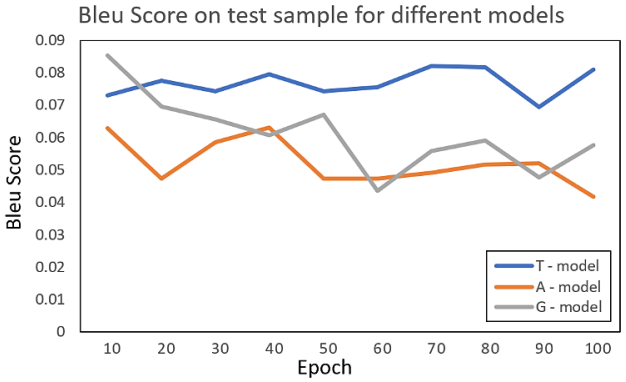
In the following plots,

G-model: encoder-decoder Seq2Seq LSTM model,
A-model: encoder-decoder Seq2Seq LSTM model with attention mechanism, and
T-model: encoder-decoder Transformer model

Based on the above tabulated values, following Bleu score trends were observed for train and test samples:



As seen above, the Bleu scores on sample train data improved drastically with increasing epoch values for T-model and A-model. However, it remained steady for G-model.

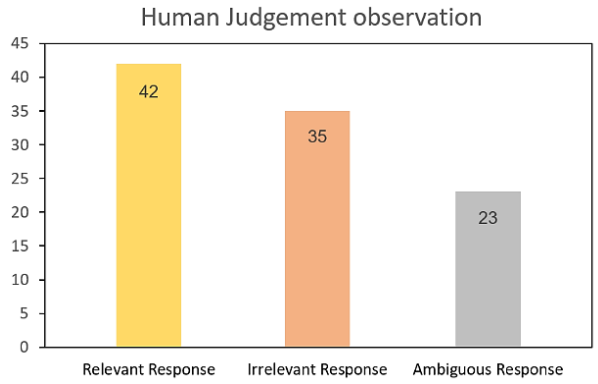


From the above graph, we observe high value of Bleu score on test sample data for the T-model. However,

this improvement was not observed as we expected from A-model in comparison to G-model. Furthermore, the trend differed for different set of sample data. By training our models with relatively smaller data set, we observed improved performance for Seq2Seq model with attention mechanism when compared to the one with the attention mechanism.

4.3.2. Human Judgement

We tested our model by framing a set of 100 questions (unknown data), and by manually fetching the data to the model to get the response. Based on our test, we observed the following:



In the above graph, the responses were categorized based on human perception.

5. Conclusion:

In this study we trained and compared three different conversational dialogue generation models i.e., Seq2Seq LSTM model, Seq2Seq LSTM model with Attention mechanism, and the Transformer model, and performed model evaluation for comparison.

Based on our research, we expected the evaluation scores to improvise as more advanced models are used. However, this was not completely observed in our analysis. The Bleu scores for the Seq2Seq model with attention mechanism were approximately less than or equal to that of Seq2Seq without attention in some epochs. This can be explained by the fact that we trained the Seq2Seq model having attention with reduced vocabulary due to limited computational power. This is because we wanted to train all the three models on same set of data. As a result, the missing vocabularies were unknown to the model. This could be improved by setting the right hyper parameters for model optimization and training the model with increased vocabulary over more epochs.

The responses and Bleu scores of the Transformer model, on the other hand, were significantly higher than the results of other two models. Conclusively, this model can improve conversational chatbot performance despite the restrictions.

6. References:

[1] Sojasingarayar Abonia. (2020). *Seq2Seq AI Chatbot with Attention Mechanism*, Department of Artificial Intelligence IA School/University-GEMA Group Boulogne-Billancourt, France.

[2] Vinyals, Oriol & Le, Quoc. (2015). *A Neural Conversational Model*. ICML Deep Learning Workshop, 2015.

[3] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia. (2017). *Attention Is All You Need*.

[4] Richard Csaky. 2019. *Deep learning based chatbot models*. National Scientific Students' Associations Conference.

[5] Luong Minh-Thang, Pham Hieu, and Manning Christopher. (2015). D. *Effective Approaches to Attention-based Neural Machine Translation*. Computer Science Department, Stanford University, Stanford.

[6] Sutskever, I., Vinyals, O., and Le, Q. (2014). *Sequence to sequence learning with neural networks*. In Advances in Neural Information Processing Systems (NIPS).

[7] Shah, Jugal; Mohammed, Sabah. (2020). *Chatbot Analytics Based on Question Answering System Movie Related Chatbot Case Analytics.pdf*. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.12061095.v1

[8] Wang, Qiang & Li, Bei & Xiao, Tong & Zhu, Jingbo & Li, Changliang & Wong, Derek & Chao, Lidia. (2019). *Learning Deep Transformer Models for Machine Translation*. 1810-1822. 10.18653/v1/P19-1176.

[9] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. (2002). *BLEU: a method for automatic evaluation of machine translation*.

[10] Chen, Boxing & Cherry, Colin. (2014). *A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU*. 362-367. 10.3115/v1/W14-3346.

[11] *Transformer model for language understanding*, available at https://www.tensorflow.org/tutorials/text/transformer

[12] *Sequence to Sequence learning in Keras*, available at https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html