Chatbot implementation using Deep Natural Language Processing

Submitted in partial fulfillment of
the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

Submitted by
Abhinav Dixit (20151012)
Ishaan Rajput (20154086)
Abhishek Sharma (20154077)
Harshita Rastogi (20154041)
John Prasad (20154010)

Under the guidance of
Dr. Divya Kumar

Department of Computer Science and Engineering
Motilal Nehru National Institute Of Technology, Allahabad
Prayagraj, UP, India

April,2019

UNDERTAKING

Motilal Nehru National Institute of Technology Allahabad

We declare that the work presented in this report titled ”Chatbot implementation using Deep Natural Language Processing”, submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology, Allahabad, for the award of the Bachelor of Technology degree in Computer Science & Engineering, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, We accept that our degree may be unconditionally withdrawn.

April, 2019

Abhinav Dixit (20151012)
Ishaan Rajput (20154086)
Abhishek Sharma (20154077)
Harshita Rastogi (20154041)
John Prasad (20154010)

Preface

This project builds a chat-bot using Deep Natural Language Processing. It implements a seq2seq model (an encoder-decoder Recurrent Neural Network) for a simple conversation task. This model can be trained to map an input sequence (questions) to an output sequence (response), which are not necessarily of the same length as each other. We have used various Natural Language Processing techniques namely Bag Of Words(BoW) model, tokenization and word embedding.

CERTIFICATE

Certified that the work contained in the report titled *"Chatbot implementation using Deep Natural Language Processing"*, by Abhinav Dixit, Ishaan Rajput, Abhishek Sharma, Harshita Rastogi and John Prasad, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Divya Kumar)
CSED Dept.
M.N.N.I.T, Allahabad

ACKNOWLEDGEMENT

We are profoundly grateful to our mentor Dr. Divya Kumar for his constant help, motivation, support and technical guidance. We express our sincere appreciation for his encouragement and advice that enabled us to pursue this project. Due to his assistance and guidance, we as a team were able to make this project a success.

Contents

Chapter 1

Introduction

A chatbot is an artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile apps or through the telephone. However, from a technological point of view, a chatbot only represents the natural evolution of a Question Answering system leveraging Natural Language Processing (NLP).

Rule based models make it easy for anyone to create a bot. But it is incredibly difficult to create a bot that answers complex queries. The pattern matching is kind of weak and hence, AIML based bots suffer when they encounter a sentence that doesn't contain any known patterns. What if we can build a bot that learns from existing conversations (between humans). This is where Machine Learning comes in. Advances in natural language processing have made deep learning approaches very popular. There are several deep learning approaches to choose from, with new methods constantly being developed.

1.1 Motivation

Most businesses these days have a web presence. But with being on the internet, boundaries of day and night, availability and unavailability have changed, so have user expectations. This is probably the biggest reason to use them. Bots give the user an interactive experience. It makes customers feel they are working with someone to help resolve their issue. If done right, bots can help customers find what they are looking for and make them more likely to return. Chatbots can be useful in many aspects of the customer experience, including providing customer service, presenting product recommendations and engaging customers through targeted marketing campaigns.

Chapter 2

Related Work

The rise of conversational AI has been made possible by recent breakthroughs in human parity level speech detection and smarter sentiment analysis. Apart from the **generative methods** which require deep NLP model to implement a chatbot, the following approaches have been used in the past for building of chatbots-

- **Retrieval-Based**: When given user input, the system uses heuristics to locate the best response from its database of predefined responses. Dialogue selection is essentially a prediction problem, and using heuristics to identify the most appropriate response template may involve simple algorithms like keywords matching or it may require more complex processing with machine learning. Regardless of the heuristic used, these systems only regurgitate predefined responses and do not generate new output.
- **Ensemble Methods**: This approach may use a rule-based approach to sing a song, a retrieval-based approach to talk about the news, and a generative approach to handle other, unspecified use cases. The most advanced systems use hierarchical reinforcement learning, which uses a low-level dialogue policy to address the immediate task, while a higher-level policy coordinates model selection or other strategic goals.
- **Grounded Learning**: Human dialogue relies extensively on context and external knowledge. This inability of chatbots to incorporate real-world knowledge also means that generative models are still very bad at creating useful or meaningful chatter. Most human knowledge does not reside in structured datasets and continue to exist as vast quantities of unstructured data, in the form of text and images. Grounded learning still faces many problems and challenges, one of which is the

challenge of accessing knowledge bases in the context of end-to-end differentiable training for neural networks. For backpropagation to be used to train an entire network, the mechanisms which access external knowledge bases must also be fully differentiable. Grounded learning is still an area of active research.

Chapter 3

Proposed Work

In this project, we use generative methods on a large amount of conversational training data in order to learn how to generate new dialogue that resembles it. But before feeding the dataset into the sequence-to-sequence model, we require to do natural language processing in order to convert the dataset into a usable form.

3.1 Natural Language Processing

3.1.1 Tokenisation & Padding

We first work on the dataset to convert the variable length sequences into fixed length, by the use of padding. We use a few special symbols(tokens) to fill in the sequences.

- **SOS** : Start of Sentence
- **EOS** : End of Sentence
- **PAD** : Filler
- **OUT** : Unknown; word not in vocabulary or not frequent enough

After this, each word is assigned a unique integer based on the frequency of the words, thus converting each sentence into a fixed-length(i.e. 25) vector of tokens.

3.1.2 Word Embedding

Through word embedding, we convert each word to a dense vector of a fixed size(i.e. 1024). Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Thus, each sequence after doing natural language processing is converted into a 2D vector of dimensions 1024 X 25.

3.2 Sequence-To-Sequence Model

Sequence-To-Sequence Model(5) uses 2 LSTM's one after another in order to work with sequence of values(words in our case). It mainly consists of two structures–Encoder & Decoder. The encoder encodes the information of the input sentence into a single LSTM layer. The decoder finally unwraps the encoded layer in order to produce a series of outputs, thus framing the required response.

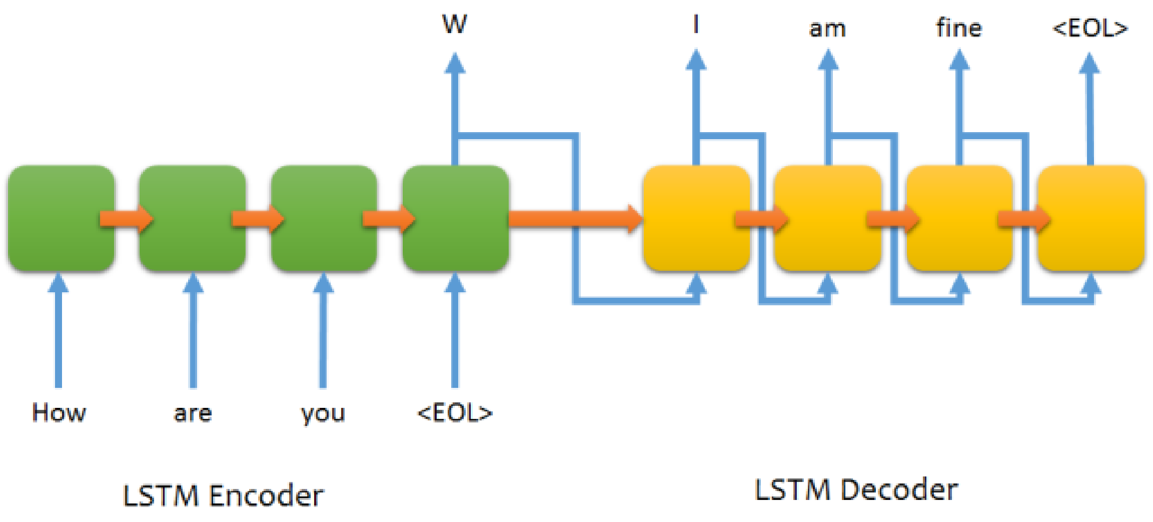


Figure 3.1: Seq2Seq model

3.2.1 Gradient Clipping

Gradient clipping limits the magnitude of the gradient and can make stochastic gradient descent (SGD) behave better in the vicinity of steep cliffs. The steep cliffs commonly occur in recurrent networks in the area where the recurrent network behaves approximately linearly. SGD without gradient clipping overshoots the landscape minimum, while SGD with gradient clipping descends into the minimum. Here we clip the gradient value between **-5 and +5**.

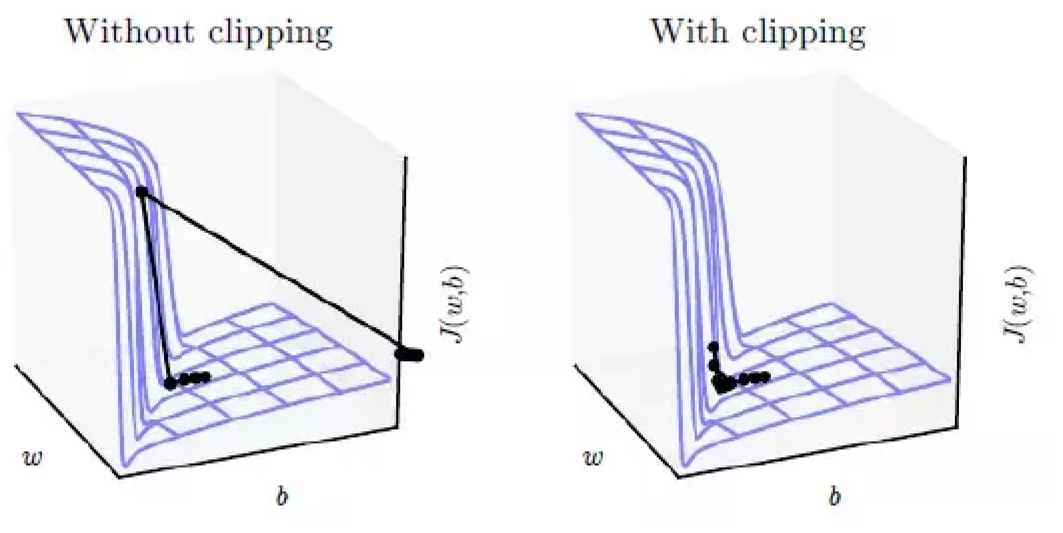


Figure 3.2: Gradient Clipping

3.2.2 Early Stopping

In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Early stopping is a practice of stopping the training of a neural network early before it has overfit the training dataset. In our model, we keep checking if the current validation loss(calculated twice per epoch) is least among all epochs. If not, we increment the early stopping counter and stop the training once this counter reaches 100. The counter is reset to 0 whenever the validation loss is found to be minimum.

Chapter 4

Background

4.1 Artificial Neural Networks

4.1.1 Artificial Neurons

Artificial neural network(3) consists of artificial neurons. Artificial neuron is a mathematical function modeling a biological neuron. The neuron receives one or more weighted inputs and fires an output. The inputs represent dendrites and output represents an axon within neuroscience perspective.

4.1.2 Feed-forward Pass

The neurons can be interconnected to form a graph. The output of a neuron is used as an input for other neurons. Such a network is called Feed-forward neural network-information flows only in one direction.

The neurons are divided into the disjoint sets, called layers l_1,..., l_k. Layers l_1,..., l_{k1} are hidden layers, l_k is an output layer. Formally layer l_0 is also considered, denoting the input data also called an input layer. The nodes in layer l_i receive as an input only the outputs of one or more connected neurons in layer l_{i1}. Neurons in a fully connected layer have connections to all activations of neurons in the previous layer. The outputs of an input layer l_0 are the input data.

The activations of neurons in output layer l_k represent the output of the network. It may represent probabilities of belonging to classes. The whole computation on a feed–forward neural network is designed with Forward propagation algorithm.

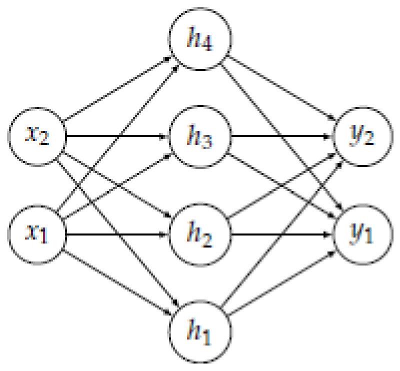


Figure 4.1: Model of a feed-forward neural network with one fully connected hidden layer and fully connected output layer

4.2 Recurrent Neural Networks

4.2.1 Introduction

Recurrent Neural Networks (RNN)(4) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory.

RNN's are relatively old, like many other deep learning algorithms. They were initially created in the 1980's, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990's.

Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next. This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

4.2.2 Basic Structure

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. A Recurrent Neural Network is able to remember exactly that, because of it's internal memory. It produces output, copies that output and loops it back into the network.*Recurrent Neural Networks add the immediate past to the present.* Therefore a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

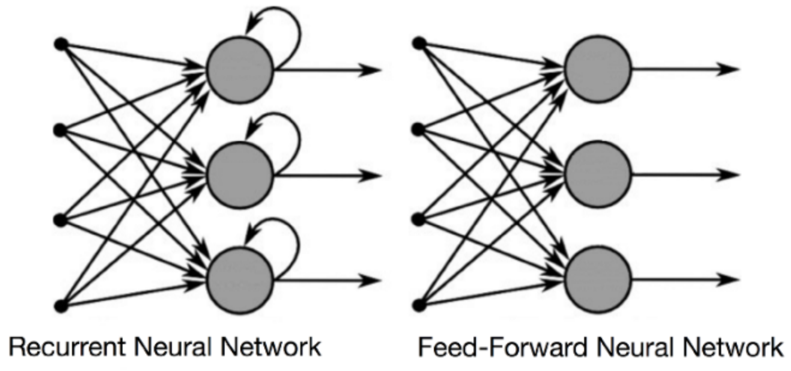


Figure 4.2: Difference in the information flow between a RNN and a Feed-Forward Neural Network

A Feed-Forward Neural Network assigns, like all other Deep Learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNN's apply weights to the current and also to the previous input. Furthermore they also tweak their weights for both through gradient descent and backpropagation through time.

4.2.3 Unfolding of a RNN

By unrolling(or unfolding) we simply mean that we write out the network for the complete sequence. On the left, you can see the RNN, which is unrolled after the equal sign. Note that there is no cycle after the equal sign since the different timesteps are visualized and information gets passed from one timestep to the next.

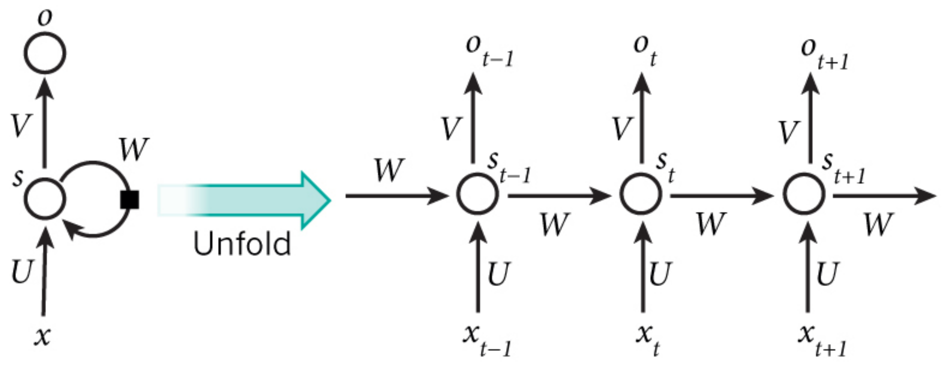


Figure 4.3: A recurrent neural network and the unfolding in time of the computation involved in its forward computation

The formulas that govern the computation happening in a RNN are as follows:

- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It is the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step:
$$s_t = f(Ux_t + Ws_{t-1}) \tag{4.2.1}$$
The function f is usually a nonlinearity such as tanh or ReLU(Rectified Linear Unit). s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary.
$$o_t = \text{softmax}(Vs_t). \tag{4.2.2}$$

4.2.4 Backpropagation in a Recurrent Neural Network

In case of a backward propagation in RNN, we figuratively going back in time to change the weights, hence we call it the **Back propagation through time(BPTT)**. In case of an RNN, if $y_t$is the predicted value $\bar{y}_t$ is the actual value, the error is calculated as a cross entropy loss-

$$E_t(\bar{y}_t, y_t) = –\bar{y}_t \; log(y_t) \tag{4.2.3}$$

$$E(\bar{y}, y) = –\sum \bar{y}_t \; log(y_t) \tag{4.2.4}$$

The steps for backpropagation are as follows-

1. The cross entropy error is first computed using the current output and the actual output.
2. Remember that the network is unrolled for all the time steps.
3. For the unrolled network, the gradient is calculated for each time step with respect to the weight parameter.

4. Now that the weight is the same for all the time steps the gradients can be combined together for all time steps.
5. The weights are then updated for both recurrent neuron and the dense layers.

4.3 Long Short Term Memory (LSTM)

4.3.1 Introduction

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

The units of an LSTM are used as building units for the layers of a RNN, which is then often called an LSTM network.

LSTMs enable RNNs to remember their inputs over a long period of time. This is because LSTMs contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory.

This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

4.3.2 Architecture of LSTM

A typical LSTM network is comprised of different memory blocks called cells.There are two states that are being transferred to the next cell - the **cell state** and the **hidden state**. The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called gates. Each of them is being discussed below-

- **Forget Gate:** A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter.

This gate takes in two inputs- h_{t-1} and x_t. h_{t-1} is the hidden state from the previous cell or the output of the previous cell and x_t is the input at that particular time step. The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state. Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a '0' is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely. Similarly, a '1' means that the forget gate wants to remember that entire piece of information. This vector output from the sigmoid function is multiplied to the cell state.

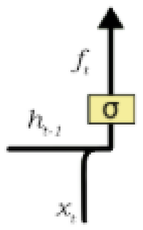


Figure 4.4: Forget Gate

- **Input Gate:** The input gate is responsible for the addition of information to the cell state. This addition of information is basically a three-step process as follows-

  1. Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from h_{t-1} and x_t.
  2. Creating a vector containing all possible values that can be added (as perceived from h_{t-1} and x_t) to the cell state. This is done using the *tanh* function, which outputs values from -1 to +1.
  3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the *tanh* function) and then adding this useful information to the cell state via addition operation.

  Once this three-step process is done with, we ensure that only that information is added to the cell state that is important and is not redundant.

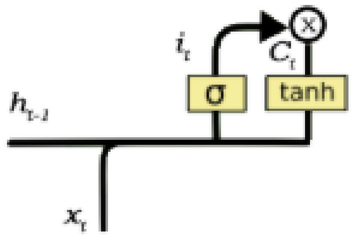


Figure 4.5: Input Gate

- **Output Gate:** The job of selecting useful information from the current cell state and showing it out as an output is done via the output gate. The functioning of an output gate can again be broken down to three steps-
  1. Creating a vector after applying **tanh** function to the cell state, thereby scaling the values to the range -1 to +1.
  2. Making a filter using the values of h_{t-1} and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
  3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.

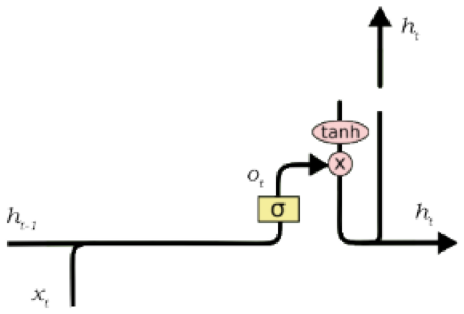


Figure 4.6: Output Gate

4.4 Sequence to Sequence Model

Sequence To Sequence(2) model introduced in Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation has since then, become the Go-To model for Dialogue Systems and Machine Translation. It consists of two RNNs : An Encoder and a Decoder. The encoder takes a sequence(sentence) as input and processes one symbol(word) at each timestep. Its objective is to convert a sequence of symbols into a fixed size feature vector that encodes only the important information in the sequence while losing the unnecessary information. You can visualize data flow in the encoder along the time axis, as the flow of local information from one end of the sequence to another.

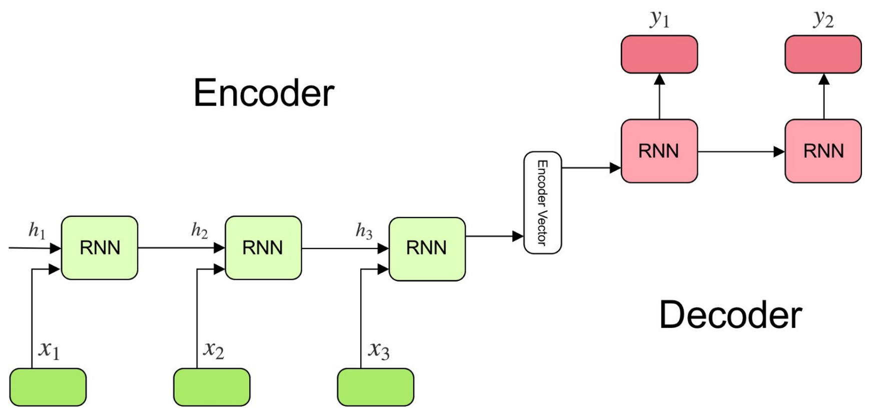


Figure 4.7: Seq2Seq model

Each hidden state influences the next hidden state and the final hidden state can be seen as the summary of the sequence. This state is called the context or thought vector, as it represents the intention of the sequence. From the context, the decoder generates another sequence, one symbol(word) at a time. Here, at each time step, the decoder is influenced by the context and the previously generated symbols.

4.4.1 Attention Mechanism

One of the limitations of seq2seq(1) framework is that the entire information in the input sentence should be encoded into a fixed length vector, context. As the length of the sequence gets larger, we start losing considerable amount of information. This is why the basic seq2seq model doesn't work well in decoding large sequences. The attention mechanism, allows the decoder to selectively look at the input sequence while decoding. This takes the pressure off the encoder to encode every useful information from the input.

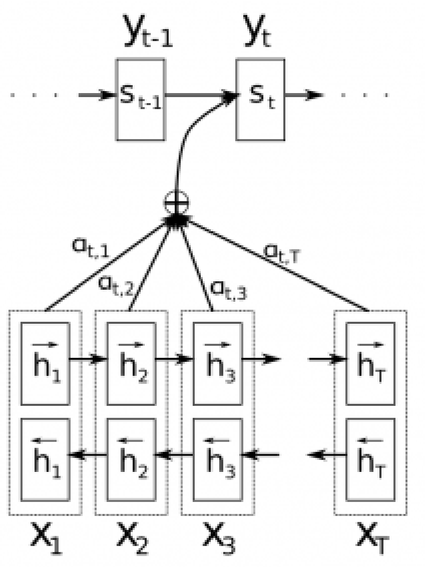


Figure 4.8: Attention Mechanism

During each timestep in the decoder, instead of using a fixed context (last hidden state of encoder), a distinct context vector c_i is used for generating word y_i. This context vector c_i is basically the weighted sum of hidden states of the encoder.

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j \tag{4.4.1}$$

where n is the length of input sequence, h_j is the hidden state at timestep j.

$$\alpha_{ij} = \exp(e_{ij}) / \sum_{k=1}^{n} \exp(e_{ik}) \tag{4.4.2}$$

e_{ij} is the alignment model which is function of decoder's previous hidden state s_{i-1} and the jth hidden state of the encoder. This alignment model is parameterized as a feedforward neural network which is jointly trained with the rest of model.

Each hidden state in the encoder encodes information about the local context in that part of the sentence. As data flows from word 0 to word n, this local context information gets diluted. This makes it necessary for the decoder to peak through the encoder, to know the local contexts. Different parts of input sequence contain information necessary for generating different parts of the output sequence. In other words, each word in the output sequence is aligned to different parts of the input sequence. The alignment model gives us a measure of how well the output at position i match with inputs at around position j. Based on which, we take a weighted sum of the input contexts (hidden states) to generate each word in the output sequence.

4.5 Natural Language Processing (NLP)

4.5.1 Introduction

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human languages. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.

Natural language processing (NLP) is not a new science, the technology is rapidly advancing thanks to an increased interest in human-to-machine communications, plus an availability of big data, powerful computing and enhanced algorithms. As a human, one may speak and write in English, Spanish or Chinese. But a computer's native language – known as machine code or machine language – is largely incomprehensible to most people. At any device's lowest levels, communication occurs not with words but through millions of zeros and ones that produce logical actions.

Natural language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment and determine which parts are important. Today's machines can analyze more language-based data than humans, without fatigue and in a consistent, unbiased way. Considering the staggering amount of unstructured data that's generated every day, from medical records to social media, automation will be critical to fully analyze text and speech data efficiently.

4.5.2 NLP working

Natural language processing includes many different techniques for interpreting human language, ranging from statistical and machine learning methods to rules-based and algorithmic approaches. Basic NLP tasks include tokenization and parsing, lemmatization/stemming, part-of-speech tagging, language detection and identification of semantic relationships. In general terms, NLP tasks break down language into shorter, elemental pieces, try to understand relationships between the pieces and explore how the pieces work together to create meaning. Tasks involved in NLP are:

- Content categorization
- Topic discovery and modeling
- Contextual extraction
- Sentiment analysis
- Speech-to-text and text-to-speech conversion
- Document summarization
- Machine translation

In all these cases, the overarching goal is to take raw language input and use linguistics and algorithms to transform or enrich the text in such a way that it delivers greater value.

4.5.3 Bag of Words in NLP

A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms. A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

It is called a "bag" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

In BoW, vocabulary is designed, which contains all the words from the dataset. After that the next step is to score the words in each document. The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.

4.5.4 Word Embedding

Word Embedding is a technique for learning dense representation of words in a low dimensional vector space. Each word can be seen as a point in this space, represented by a fixed length vector. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. The **word vectors** have some interesting properties.

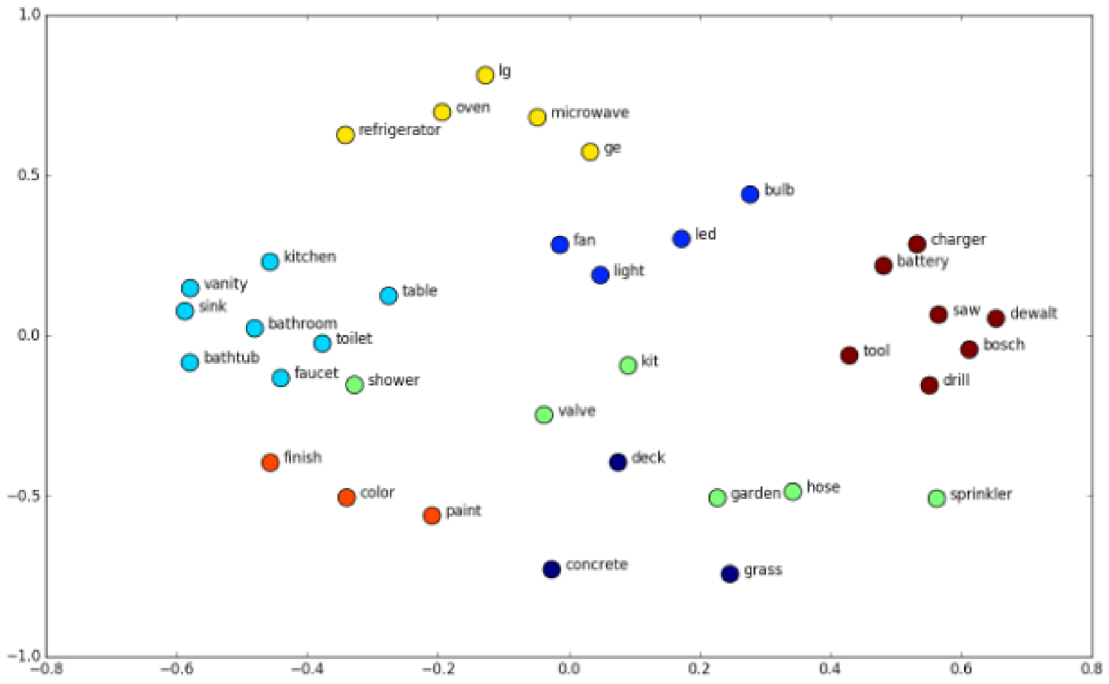


Figure 4.9: Word Embedding

Word Embedding is typically done in the first layer of the network : Embedding layer, that maps a word (index to word in vocabulary) from vocabulary to a dense vector of given size. In the seq2seq model, the weights of the embedding layer are jointly trained with the other parameters of the model.

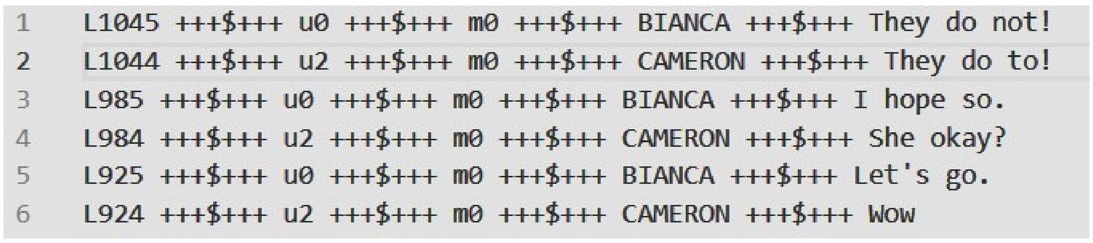
Chapter 5

Experimental Setup and Results Analysis

5.1 Dataset

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

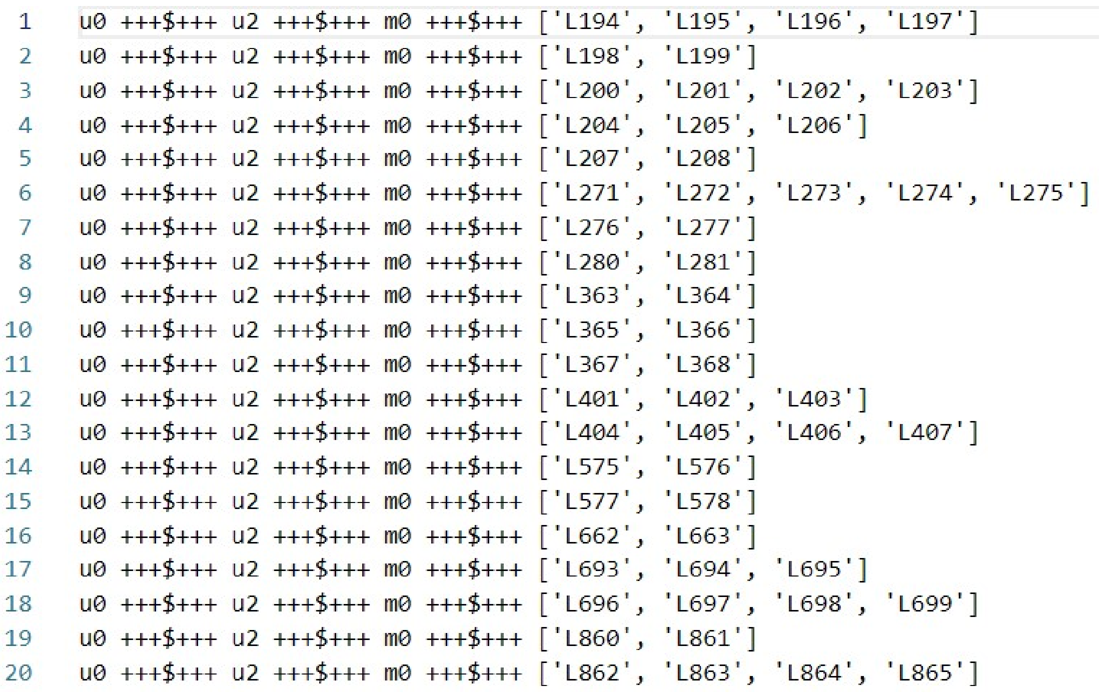
- 220,579 conversational exchanges between 10,292 pairs of movie characters.
- in total 304,713 dialogues.



```
L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!
L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!
L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.
L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?
L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.
L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow
```

Figure 5.1: Dataset Sample(movie_lines.txt)

This above dataset shows the dialogueID, followed by the speaker, the movieID, the name of the speaker and finally the dialogue.



```
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L204', 'L205', 'L206']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L207', 'L208']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L271', 'L272', 'L273', 'L274', 'L275']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L276', 'L277']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L280', 'L281']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L363', 'L364']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L365', 'L366']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L367', 'L368']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L401', 'L402', 'L403']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L404', 'L405', 'L406', 'L407']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L575', 'L576']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L577', 'L578']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L662', 'L663']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L693', 'L694', 'L695']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L696', 'L697', 'L698', 'L699']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L860', 'L861']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L862', 'L863', 'L864', 'L865']
```

Figure 5.2: Dataset Sample(movie_conversations.txt)

This above dataset shows by the speaker, the listener, the movie and the dialogues being spoken in a single conversation. The string "+++$+++" is used as a separator.

5.2 Softwares and Hardwares Used

- Software:

  Python 3

  Tensor-flow

  Keras

  Cuda (NVIDIA Computing Toolkit)

  Matplotlib

  SciPy

- Recommended Hardware:

  NVIDIA GT - 960 and higher

  Intel i5 - 6th gen(Dual Core) and higher

  GPU - VRAM : 4GB and higher

  SSDs

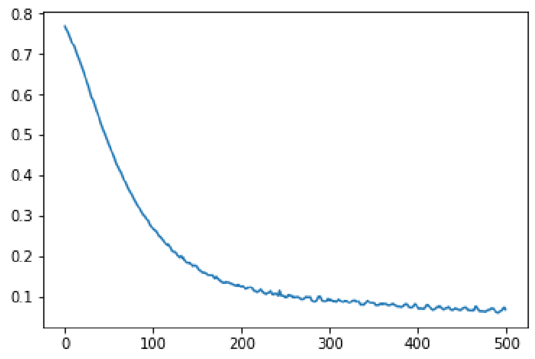
  RAM : 8GB and higher.

5.3 Result Analysis

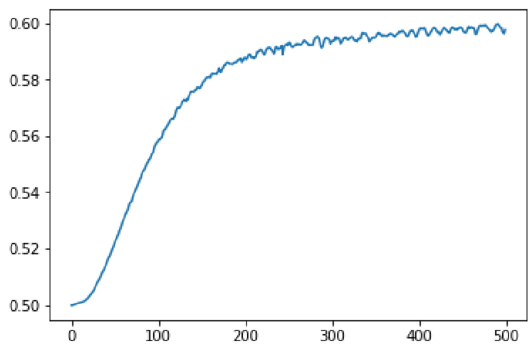
5.3.1 First trained model

For our first model, we used a relatively simple model. We used only the first 80000 pair of question & answers. The dataset was converted into word embedding of size 128. It was then fed into a seq2seq model consisting of 2 LSTM layers each for encoder & decoder layers of size 128. This model was trained for 500 epochs using Adam Optimizer for loss reduction.

Following is the validation loss & accuracy for each epoch:

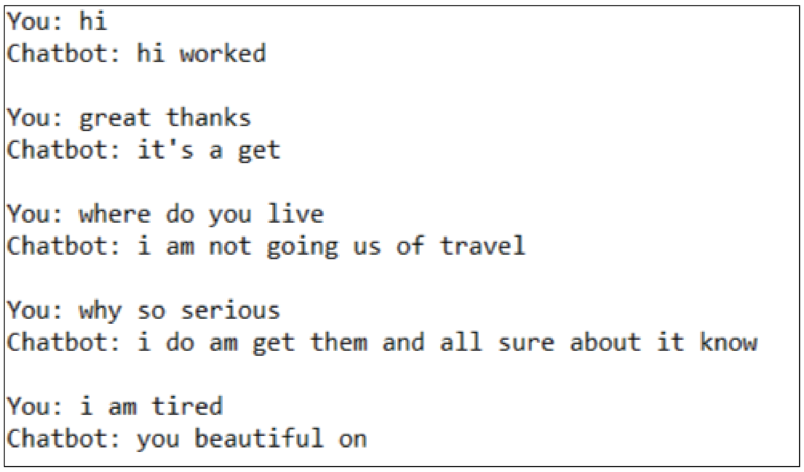


(a) Validation Loss per Epoch



(b) Accuracy per Epoch

The accuracy increases initially but soon the model stops learning indicating that the model is too simple for the given dataset. Furthermore, the output of the model were sentences which were both grammatically and contextually incorrect.



```
You: hi
Chatbot: hi worked

You: great thanks
Chatbot: it's a get

You: where do you live
Chatbot: i am not going us of travel

You: why so serious
Chatbot: i do am get them and all sure about it know

You: i am tired
Chatbot: you beautiful on
```

Figure 5.4: Chatbot responses

5.3.2 Second trained model

In order to tackle underfitting, we increased the model complexity. This time we used word embeddings of size 1024. The seq2seq model consisted of 2 LSTM layers for encoder & decoder layers of size 1024. We also used techniques like early stopping and gradient clipping for better result. The model was trained for 60 epochs this time using Adam Optimizer. The dataset used for training above model consisted of 1,00,000 pair of questions and answers.

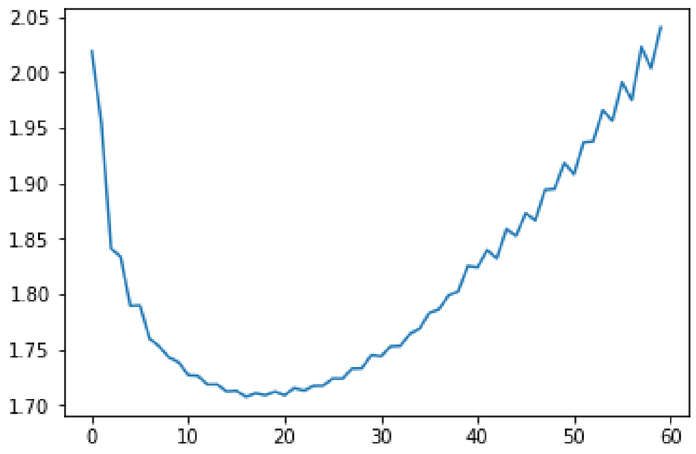
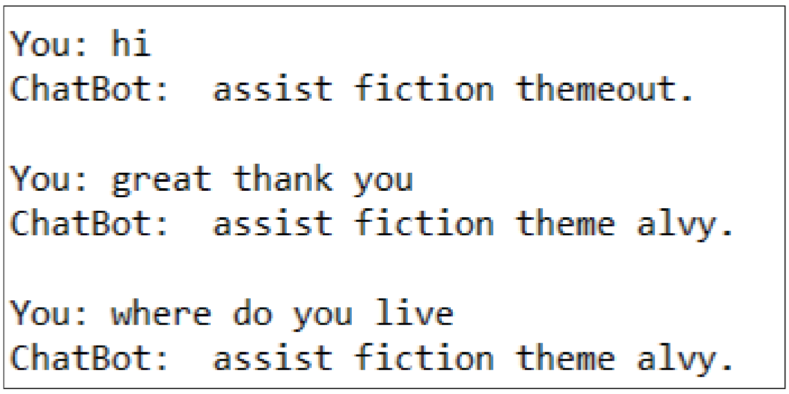


Figure 5.5: Validation Loss per Epoch

Here we can see that the model overfits after 20 epochs. When we tested the outputs for different inputs, we got repeated phrases as responses for all the inputs. Due to overfitting our model tends to prioritize high-priority, high-probability response content.



```
You: hi
ChatBot:  assist fiction themeout.

You: great thank you
ChatBot:  assist fiction theme alvy.

You: where do you live
ChatBot:  assist fiction theme alvy.
```

Figure 5.6: Chatbot responses

5.3.3 Third trained model

The previous model suffered due to lack of dataset thus resulting in overfitting. Hence, this time we used the complete dataset(around 220000 pair of question and responses). We increased the number of LSTM layers from 2 to 3. The rest hyperparameters were kept same as the second trained model. The model was trained for 70 epochs. This produced satisfactory results as you can see from the graph below.

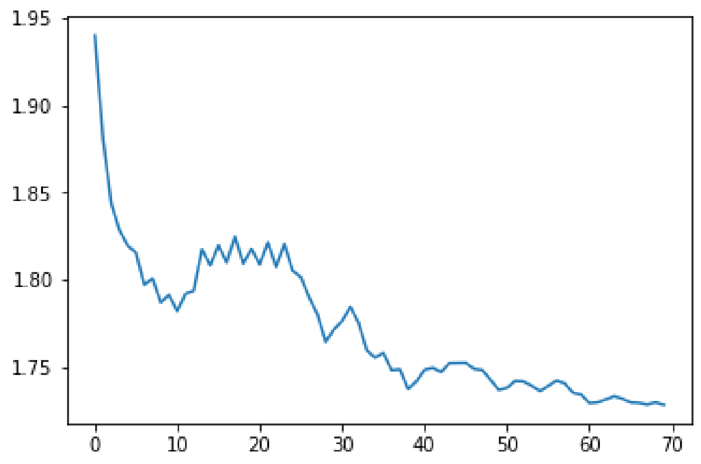


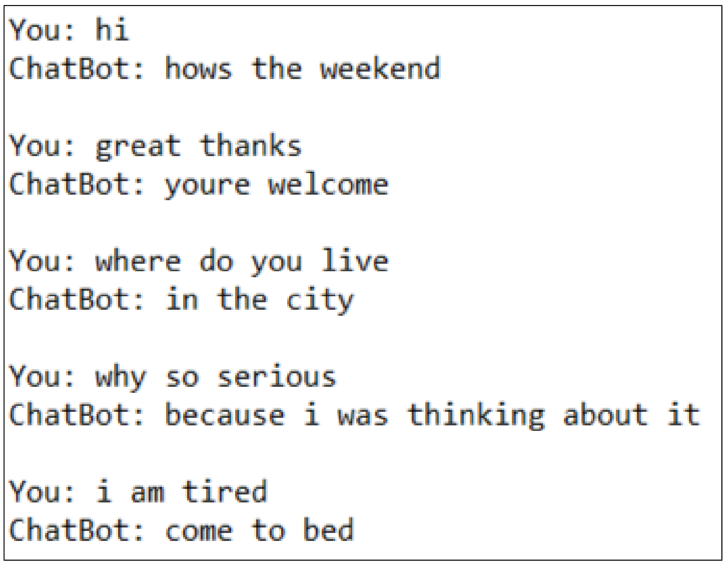
Figure 5.7: Validation Loss per Epoch

The model did not encounter overfitting and was able to reduce validation loss to appreciable value.

Chapter 6

Conclusion and Final Results

In our project, we targeted to make a model that is able to predict conversations that are both grammatically as well contextually correct. The chatbot should also be capable of replicating the conversation style of the speakers based on the dataset. Here are few of the results while testing the chatbot at the final stage of the project.



```
You: hi
ChatBot: hows the weekend

You: great thanks
ChatBot: youre welcome

You: where do you live
ChatBot: in the city

You: why so serious
ChatBot: because i was thinking about it

You: i am tired
ChatBot: come to bed
```

Figure 6.1: Chatbot responses

We figured out that when we increased the number of layers in LSTM encoder and decoder, we got appreciable results. However due to lack of computation power, we could not use a deeper LSTM thus resulting in unsatisfactory responses some times.

It can be also noted that the responses given by the chatbot are very domain specific depending on the dataset that it is trained on. The movies dataset that we used for this project is pretty diverse in terms of the human conversations. To use this chatbot for a specialized purpose, we should train it on a specialized dataset.

Chapter 7

Future Works

7.1 Adding in more data

The usual tendency of a Deep Learning model is that it performs better when it is provided with more data. Therefore, with time, as the data will increase the model will be able to be trained better. This will produce better accuracy and results.

7.2 Keep Feeding chatbot With New Information

Chatbot need to be fed continuously with real-time relevant information to stay up to date otherwise it will become outdated. To act more real and human-like, every chatbot must be maintained regularly and brought up to date so as to give the best results.

7.3 Specialized dataset

To make a chatbot specific to a topic like movies, games, markets etc, we have to use dataset related to that topic only. It will give better results for that particular domain.

7.4 Integration of voice

To make chatbot feasible to blind or illiterate people, voice recognition and text-to-speech can be added as extra feature.

Bibliography

[1] Chatbots with seq2seq. http://complx.me/2016-06-28-easy-seq2seq/.

[2] Implementing a sequence-to-sequence model. https://hackernoon.com/implementing-a-sequence-to-sequence-model-45a6133958ca.

[3] Cho, K. Learning phrase representations using rnn encoder–decoder for statistical machine translation. Master's thesis, University de Montreal, September 2014.

[4] Kostadinov, S. *Recurrent Neural Networks with Python Quick Start Guide*. Packt.com, 2018.

[5] Vinyals, O. Sequence to sequence learning with neural networks. Master's thesis, Google, December 2014.