**” CHATBOT using Deep Neural Network”**

**Submitted in the partial fulfillment of the**

**requirement for**

**Bachelor of Engineering in Computer Science Engineering**

**DECLARATION**

**I, hereby declare that this project entitled ” CHATBOT using Deep Neural Network”, is an original bonafide work carried out in fulfillment of the requirement for the award of Bachelor of Engineering in Computer Science Engineering affiliated to Chandigarh University. This Project work is done under the guidance of Professor Gursimran Kaur.**

**I also declare that no part of this representation has been previously published or submitted as a project report for any degree or diploma of any University or Institution.**

**DATE: Signature**

**\_\_\_\_\_\_\_\_\_\_\_**

**ACKNOWLEDGEMENT**

We are highly grateful to the Director of Chandigarh University, Gharuan for providing us this opportunity to carry out the project work.

The constant guidance and encouragement received from Prof. Gursimran Kaur has been of great help in carrying out the project work and it is acknowledged with the reverential thanks.

We would like to express a deep sense of gratitude and thanks proudly to our project coordinator. Without their wise counsel and able guidance, it would have been impossible to complete the report in this manner.

We express other guidance gratitude to our faculty members of CSE department of CU GHARUAN for their intellectual support throughout the course of their work.

We are also grateful gratitude to our teachers for their constant support and guidance. At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during the project work.

Finally, we are indebted to all whosever have contributed in this report work.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Topic | Page no. |
| 1. | ACKNOWLEDGEMENT | i |
| 2. | ABSTRACT | ii |
| 3. | LIST OF FIGURES | iii |
| 4. | INTRODUCTION | 1 |
| 5. | SYSTEM REQUIREMENT SPECIFICATION | 11 |
| 6. | ARCHITECTURE DIAGRAM | 14 |
| 7, | PROJECT METODOLOGY | 17 |
| 8. | SCREENSHOTS | 18 |
| 9. | CONCLUSION AND FUTURE SCOPES | 20 |
| 10. | REFERENCES | 23 |

**ABSTRACT**

A Chatbot is artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, and mobile apps or through the telephone. These chatbots use deep learning models. They use natural language processing models which use deep learning and then find at the bottom here we've got the sequence to sequence models which we'll be interested in at the end.

It’s true that, the lines between applications and chatbots can become a little bit blurred if chatbots interact via a user interface. A Chatbot however can be differentiated from an app in the way that the interactions with the bot take place, more or less sequentially (as a conversation), and the bot is used inside a chat app.

Another obvious way in which a Chatbot is different from an app is a little more reminiscent of the science fiction example, and that is the Chatbot as metaphor for an automated agent. A Chatbot unlike an app has a “identity” that is actually separate from its interaction with the user. This is in the same way that the human agent exists independently of their interaction with customers.

In short a Chatbot is another way of humans interacting with software. While there are overlaps with functionality offered by websites and apps, interacting with a chatbot is different to interacting with a website or with an app.

It is true that in some sense messaging platforms are becoming universal mobile apps or app portals. Businesses want to find ways to deliver their messages and services in the place that the consumers are which is on chat platforms. Chatbots give them a way to do this.

This definition however often leads to two potential misconceptions.

The biggest misconception that arises is that a chatbot is a bot that converses with a human in the way that another human would converse with a human. Software or even a robot (the digital part of the robot is of course software) that communicates with a human in natural language is not difficult to imagine. Science fiction is full of examples. While this may the end goal, this is simply not possible using the current technology. Not only is it not possible, it often leads to unrealistic expectations regarding the chatbots capabilities and inevitable frustrations when those expectations are not met.

The second misconception is that a chatbot communicates using only text or voice. Actually chatbots allow users to interact with them via graphical interfaces or graphical widgets, and the trend is in this direction. Many chat platforms including WeChat, Facebook Messenger and Kik allow web views on which developers can create completely customized graphical interfaces.

1. **INTRODUCTION**

The basic definition of chatbot is, it is a computer software program designed to simulate human conversation via text or audio messages. Today’s AI systems can interact with users, understand their needs, map their preferences and recommend an appropriate line of action with minimal or no human intervention. There are lot of popular conversational agents are available today like Apple’s Siri, Microsoft’s Cortana, Google Assistant, and Amazon’s Alexa.

The basic foundation of chatbots is providing the best response of any query that it receives. The best response like answering the sender questions, providing sender relevant information, ask follow-up questions and do the conversation in realistic way.

The chatbot needs to be able to understand the intentions of the sender’s message, determine what type of response message (a follow-up question, direct response, etc.) is required, and follow correct grammatical and lexical rules while forming the response. Some models may use additional meta information from data, such as speaker id, gender, emotion. Sometimes, sentiment analysis is used toallows the chatbot to ‘understand’ the mood of the user by analysing verbal and sentence structuring clues.

The NLU unit is responsible for transforming the user utterance to a predefined semantic frame according to the system’s conventions, i.e. to a format understandable for the system. This includes a task of slot filling and intent detection. For example, the intent, could be a greeting, like Hello, Hi, Hey, or it could have an inform nature, for example I like Indian food, where the user is giving some additional information. Depending on the interests, the slots could be very diverse, like the actor name, price, start time, destination city etc. As we can see, the intents and the slots are defining the closed-domain nature of the Chatbot. The task of slot filling and intent detection is seen as a sequence tagging problem. For this reason, the NLU component is usually implemented as an LSTM-based recurrent neural network with a Conditional Random Field (CRF) layer on top of it. The model presented is a sequence-to-sequence model using bidirectional LSTM network, which fills the slots and predicts the intent in the same time. On the other hand, the model is doing the same using an attention-based RNN. To achieve such a task, the dataset labels consist of: concatenated B–I–O (Begin, Inside, Outside) slot tags, the intent tag and an additional end-of-string (EOS) tag. As an example, in a restaurant reservation scenario, given the sentence Are there any French restaurants in Toronto downtown?, the task is to correctly output, or fill, the following slots: {cuisine: French} and {location: Toronto downtown}.

Natural Language Generation (NLG) is the process of generating text from a meaning representation. It can be taken as the reverse of the natural language understanding. NLG systems provide a critical role for text summarization, machine translation, and dialog systems. In the NLG, The system response as a semantic frame, it maps back to a natural language sentence, understandable for the end user. The NLG component can be rule-based or model-based. In some scenarios it can be a hybrid model, i.e. combination of both. The rule-based NLG outputs some predefined template sentences for a given semantic frame, thus they are very limited without any generalisation power. While several general-purpose rule-based generation systems have been developed, they are often quite difficult to adapt to small, task-oriented applications because of their generality. Machine learning based (trainable) NLG systems are more common in today’s dialog systems. Such NLG systems use several sources as input such as: content plan, representing meaning representation of what to communicate with the user, knowledge base, structured database to return domain-specific entities, user model, a model that imposes constraints on output utterance, dialog history, the information from previous turns to avoid repetitions, referring expressions, etc.

Trainable NLG systems can produce various candidate utterances (e.g., scholastically or rule base) and use a statistical model to rank them. The statistical model assigns scores to each utterance and is learnt based on textual data. Most of these systems use bigram and trigram language models to generate utterances.

On the other hand, In NLG based on a semantically controlled Long Short-term Memory (LSTM) recurrent network, It can learn from unaligned data by jointly optimising its sentence planning and surface realisation components using a simple cross entropy training criterion without any heuristics, and good quality language variation is obtained simply by randomly sampling the network outputs.

The DM could be connected to some external Knowledge Base (KB) or Data Base (DB), such that it can produce more meaningful answers. The Dialogue Manager consists the following two components: the Dialogue State Tracker (DST) and the Policy Learning which is the Reinforcement Learning (RL) agent. The Dialogue State Tracker (DST) is a complex and essential component that should correctly infer the belief about the state of the dialogue, given all the history up to that turn. The Policy Learning is responsible for selecting the best action, i.e. the system response to the user utterance, that should lead the user towards achieving the goal in a minimal number of dialogue turns.

Types of conversational A.I

Rule Based Chatbot

In a rule-based approach, a bot answers questions based on some rules on which it is trained on. The rules defined can be very simple to very complex. The creation of these bots are relatively straightforward using some rule-based approach, but the bot is not efficient in answering questions, whose pattern does not match with the rules on which the bot is trained. However, these systems aren’t able to respond to input patterns or keywords that don’t match existing rules. One of such languages is AIML (Artificial Intelligence Markup Language): The AIML language´s purpose is to make the task of dialog modeling easy, according to the stimulus-response approach. Moreover, it is a XML-based markup language and it is a tag based. Tags are identifiers that are responsible to make code snippets and insert commands in the chatterbot. AIML defines a data object class called AIML objects, which is responsible for modelling patterns of conversation.

Example of AIML Code,

Basic Tags:

1. **<**aiml**>:**Defines the beginning and end of an AIML document
2. **<**category**>:**Defines the knowledge in a knowledge base.
3. **<**pattern**>:**Defines the pattern to match what a user may input.
4. **<**template**>:**Defines the response of an Alicebot to user’s input.

**Retrieval Based Conversational AI**

When given user input, the system uses heuristics to locate the best response from its database of pre-defined 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 or deep learning. Regardless of the heuristic used, these systems only regurgitate pre-defined responses and do not generate new output.

With massive data available, it is intuitive to build a retrieval based conversational system as information retrieval techniques are developing fast. Given a user input utterance as the query, the system searches for candidate responses by matching metrics. The core of retrieval based conversational systems is formulated as a matching problem between the query utterance and the candidate responses. A typical way for matching is to measure the inner-product of two representing feature vectors for queries and candidate responses in a transformed Hilbert space. The modelling effort boils down to finding the mapping from the original inputs to the feature vectors , which is known as representation learning.There is two-step retrieval technique to find appropriate responses from the massive data repository. The retrieval process consists of a fast ranking by standard**TF-IDF** measurement and the re-ranking process using conversation-oriented features designed with human expertise. The systems to select the most suitable response to the query from the question-answer pairs using a statistical language model as cross-lingual information retrieval. These methods are based on shallow representations, which basically utilises one-hot representation of words. Most strong retrieval systems learn representations with deep neural networks (DNNs). DNNs are highly automated learning machines; they can extract underlying abstract features of data automatically by exploring multiple layers of non-linear transformation. Prevailing DNNs for sentence level modelling include convolution neural networks (C-NNs) and recurrent neural networks (RNNs). A series of matching methods can be applied to short-text conversations for retrieval-based systems. Basically, these methods model sentences using convolutional or recurrent networks to construct abstractive representations. Although not all of these methods are originally designed for conversation, they are effective for short-text matching tasks and are included as strong baselines for retrieval-based conversational studies.

**Generative Based**

A generative model chatbot doesn’t use any predefined repository. This kind of chatbot is more advanced, because it learns from scratch using a process called “Deep Learning.” Generative models are typically based on Machine Translation techniques, but instead of translating from one language to another, we “translate” from an input to an output (response).

Another way to build a conversational system is to use language generation techniques.We can combine language template generation with the search-based methods. With deep learning techniques applied, generation-based systems are greatly advanced.

We have a sequence-to-sequence (seq2seq) framework that emerged in the neural machine translation field and was successfully adapted to dialogue problems. The architecture consists of two RNNs with different sets of parameters.The approach involves two recurrent neural networks, one to encode the source sequence, called the encoder, and a second to decode the encoded source sequence into the target sequence, called the decoder.It was originally developed for machine translation problems, although it has proven successful at related sequence-to-sequence prediction problems such as text summarization and question answering.

So, in this context there are many possible definitions and some confusion about what a bot is. This is partly because there are so many varied use cases for bots and these influence what people perceive a Chatbot to be. The most intuitive definition is that a bot is software that can have a conversation with a human. For example a user could ask the bot a question or give it an instruction and the bot could respond or perform an action as appropriate.

1. **SRS**

2.1 Study of the current system

The motive of object detection is to recognize that an artificial intelligence service can be used to answer simple questions, help users book services, get more information about a specific topic, buy a product, etc. Having a chatbot help expedite this types of tasks, allows for human agents to focus on more relevant problems.

2.2 Intended Audience and Reading Suggestions

All IT and technology enthusiasts.

2.3 Product Scope

Future of Chatbots is very bright. With so much advancement in Artificial Intelligence sector, chatbots are the future with zero doubt.

The future chatbot will not be just a Customer Support agent, it will be an advance assistant for both the business and consumer.

We as human are not fond of doing repetitive boring tasks. So in the future companies will hire AI Chatbot for the tasks which are repetitive and doesn’t require creativity.

Also, Human doesn’t like storing up contents (mugging up) in their mind. And today with the Internet they can leverage that part. So tasks which require storing the information (data) can be transferred to AI Chatbot.

And with AI Chatbot taking over repetitive boring tasks, Companies will utilize their human resources for more creative tasks. With this, we can expect more amazing things coming up to us in the future.

At present, there is a mixed review in users (consumer) of Chatbot. Majority of users are appreciating chatbots and its feature. Today with a chatbot, users get a lot of amazing features like Instant Replies, Shopping Bot, Shipment Tracking, etc. all that at one place, a single messenger platform.

So summing all this up, yes I see a lot potential of the AI Chatbot in the future. To make a product or technology successful, it needs public support and chatbot is getting that. With Advance modern technology coming up we can expect more astonishing features from a chatbot.

2.4 Operating Environment

* Desktop
* Laptop

2.5 User Documentation

Dataset Used:

The dataset used for this project is CORNELL Movie Dataset which consists of 5 corpuses:

1. Movie Character Metadata
2. Movie Conversations
3. Movie Lines
4. Movie titles
5. Raw Scripts

Amongst these 5 files, we use “Movie Lines” to get the lines and responses for training of our chatbot. We also use “Movie Conversations” to distinguish the responses from the conversation list and make a encoder with each line is labelled with suitable integer.

File Descriptions:

DataPrep.py:

This file will contain all the pre-processing functions needed to build the corpus dataset into an encoded format for training the dataset. This will distribute the dataset into questions and their responses in suitable list.

Seq2Seq.py

This file contains the creation of Seq2Seq model. In this module, complete model is created and implemented from scratch using RNN and LSTMs as the backend function for the Seq2Seq model.

Training.py:

This file contains final training of our deep learning neural network. This file is the final product of all the modules above. In this file, we will give all the necessary functions for our chatbot as if when it is not able to process what is typed and many other exceptions.

Testing.py:

This file is used to test our trained model and look for backdoor or shortcomings of our model. If any shortcomings are found then the model is improvised and tuned properly.

2.6 Software Requirements:

* Python**:** Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.
* Nltk**:** The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.
* Tensorflow:TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
* Tkinter:Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped .
* Pickle: The pickle module implements binary protocols for serializing and de-serializing a Python object structure.“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream.

2.7 Hardware Requirements:

* RAM: 8 Gigabytes
* Operating System: Windows 10
* Graphic Card: Not necessary but recommended.(4 Gigabytes NVIDIA)
* Storage: 1 TB
* Processor: Intel i5

1. **Architecture Diagram**

3.1 Introduction

The Architecture Diagram is a graphical representation of the prototype system. The system displayed here in the following diagrams represents actual components of the built prototype. All the diagrams have been thoughtfully carefully to replicate the working of the system. The main objective of the architecture diagram chapter is to provide better understanding to the user about the prototype.

* 1. Data Flow Diagram

A dataflow diagram (DFD) is the graphical representation of the way in which data flows through a system and models the process aspects of the prototype. It provides a valuable overview of the system; which ca be further elaborated into more intricate details about the system. The following DFD displays the kind of data to be used as input and the produced output. It also throws light on where the data will be stored. The time taken for reach process is not displayed or the pattern of processing. The data on whether this is an operation working in parallel or series is not known from the diagrams.

The DFD can be split up into more levels which provide more detail about the system. This is achieved by going deep into a particular method or functioning of the system. Following the Yourdon-Coad notation of symbols, the circles are used to represent processes while the rectangles are used to represent the entities. The arrows indicate the dataflows through the system and control shifts.

3.2.1 DFD Level 0 or the Context Level Diagram

This a basic overview of the prototype proposed in this master’s thesis. This diagram of the system is meant to show the prototype as a single high-level process, connected to its various external entities. In this figure user will send his personal information to the Chat Bot to know about career opportunities in their field. The DFD level 0 shows the interaction between User and Chat Bot where the User initializes the chat and sends the message to the Chat Bot. Using Natural Language Processing Then Chat Bot sends the data and response to the User. The messages are taken to the bot via text i.e. written on the chat field or via voice, where the user speaks into the microphone and the responses are recorded and sent to the bot for processing the request. This is the main overview of the system.

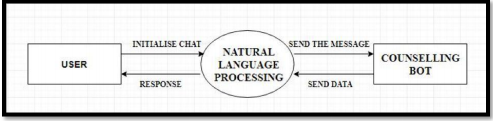


Figure: DFD level 0

3.2.2 DFD level 1

The level one dataflow diagram provides a much closer look at the process of the system. It gives furthermore intricate details, which were not displayed in level zero. This diagram shows the actual flow of working modules. Such as the User send the messages to the Chat Bot. The messages can be sent either by written texts or by speaking into the microphone, which every 36 user is more comfortable with. The message is then sent to the conversational bot. The bot takes this data and via pattern matching it finds the best possible result to reply with. This is achieved with the help of the AIML libraries. The knowledge base of company details is searched to provide the best results for the user request.

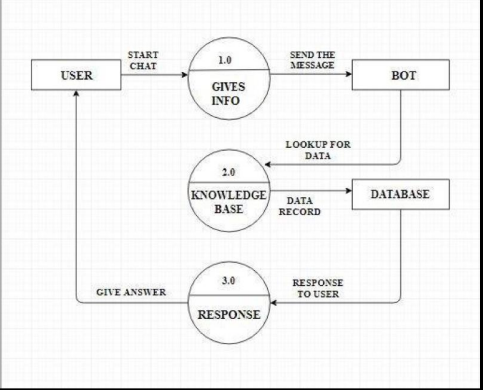


Figure: DFD level 1

* 1. Use Case Diagram

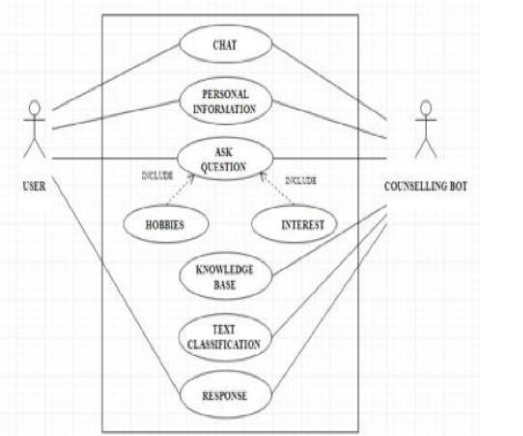
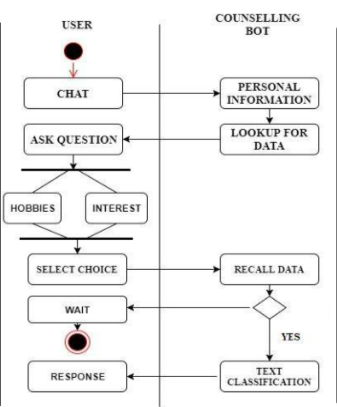


Figure: UML Diagram

The importance of the use case model is during the design analysis phase of software development. The use case diagram separates the system into its known actors. In the figure above, there are two known actors of this system, one is the chatbot and the other is the user. The use case of a system delivers the functional requirements of the system in a way which is easy to read and interpret.

* 1. Activity Diagram



The dynamic behavior of the chatbot architecture is captured by the activity diagram. This is the basic flowchart of the working of the chatbot.

1. **Methodology and Results**

* Seq2Seq

We will use an architecture called (seq2seq) or ( Encoder Decoder), It is appropriate in our case where the length of the input sequence does not has the same length as the output data .

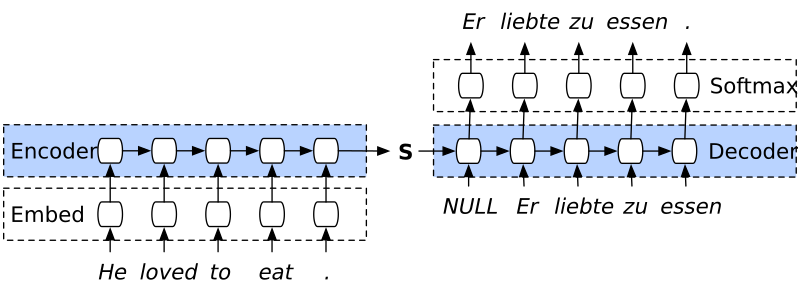
Encoder decoder architecture consists of two main parts :

1. Encoder:

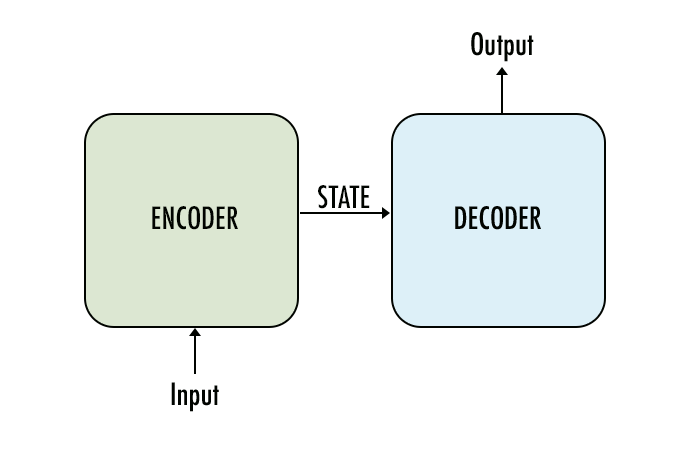
The encoder simply takes the input data, and train on it then it passes the last state of its recurrent layer as an initial state to the first recurrent layer of the decoder part.

**2.** Decoder **:**

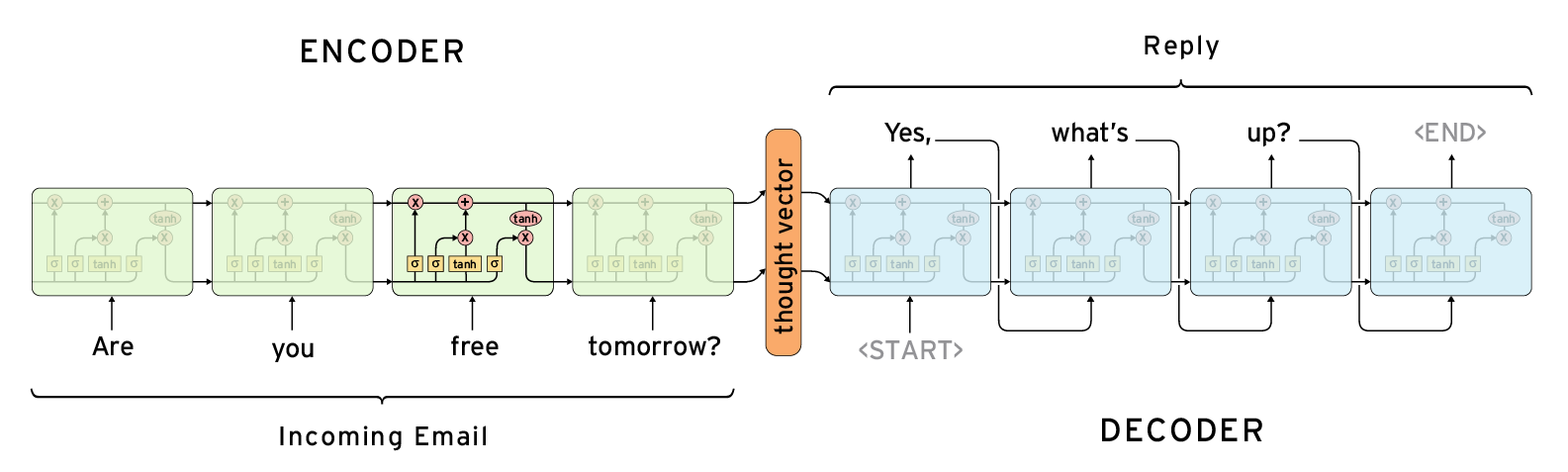
The decoder takes the last state of encoder’s last recurrent layer and uses it as an initial state to its first recurrent layer, the input of the decoder is the sequences that we want to get .



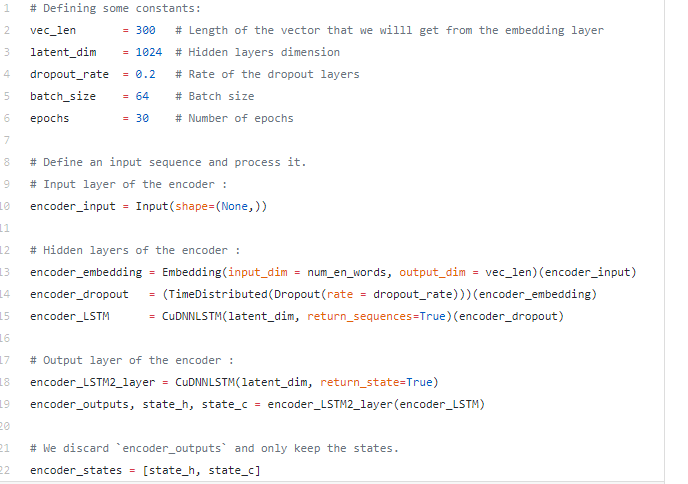
1. Simple representation :



1. Generative model Chatbots

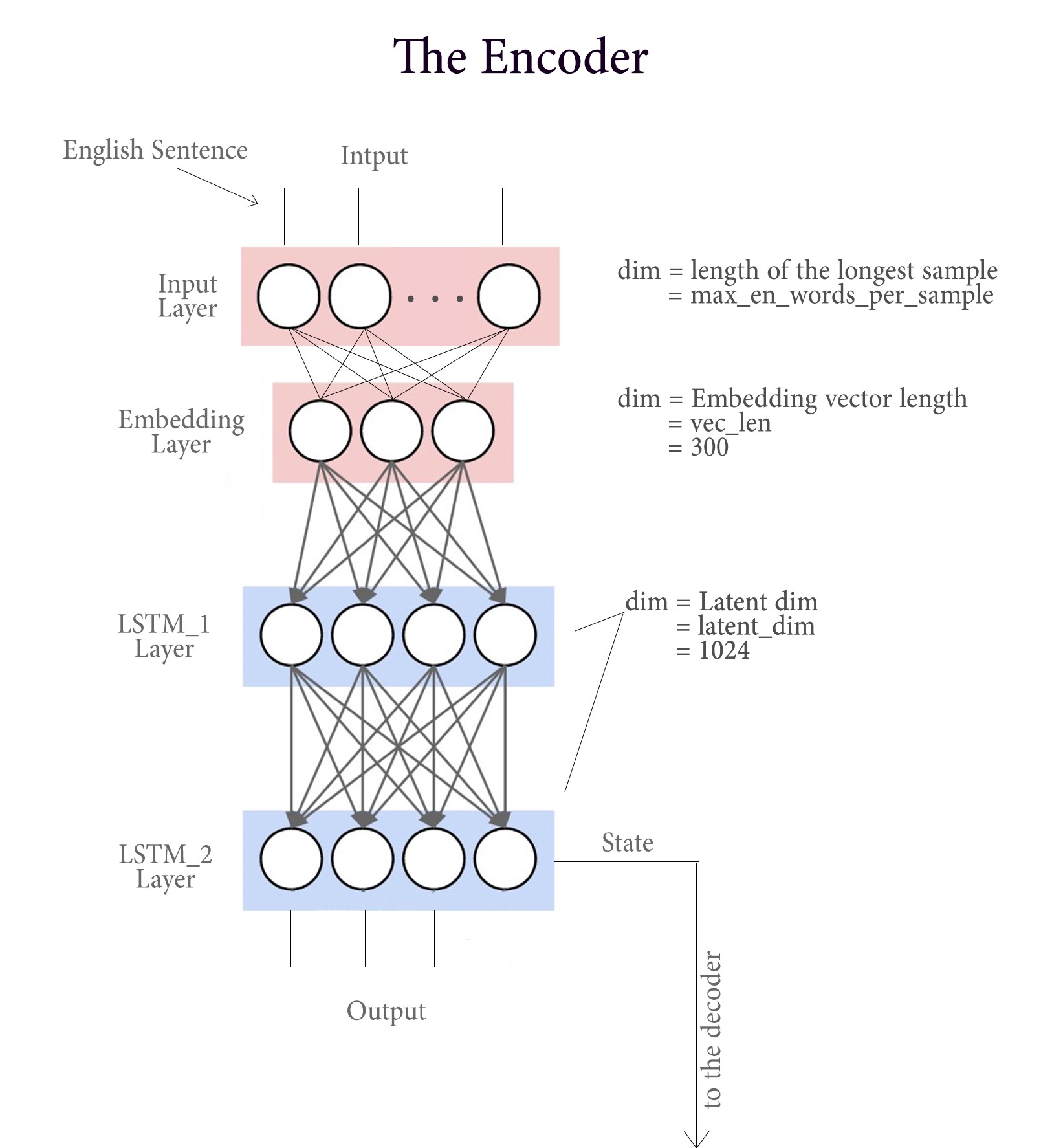


Building the encoder:

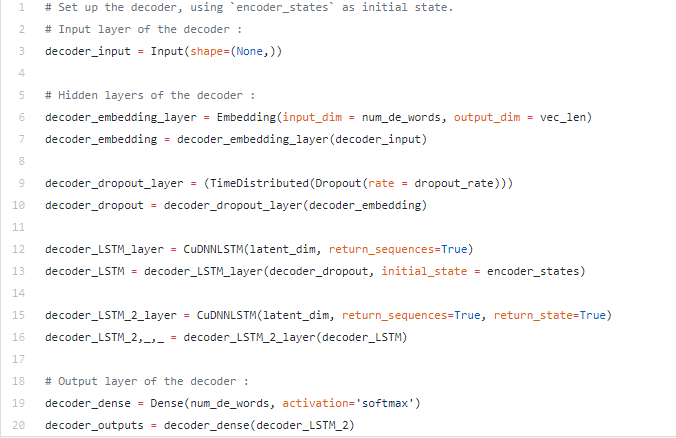


The encoder is made up of :

1. Input Layer : Takes the English sentence and pass it to the embedding layer.
2. Embedding Layer : Takes the English sentence and convert each word to fixed size vector
3. First LSTM Layer : Every time step, it takes a vector that represents a word and pass its output to the next layer, We used layer instead of LSTM because it’s much much faster.
4. Second LSTM Layer : It does the same thing as the previous layer, but instead of passing its output, it passes its states to the first LSTM layer of the decoder .

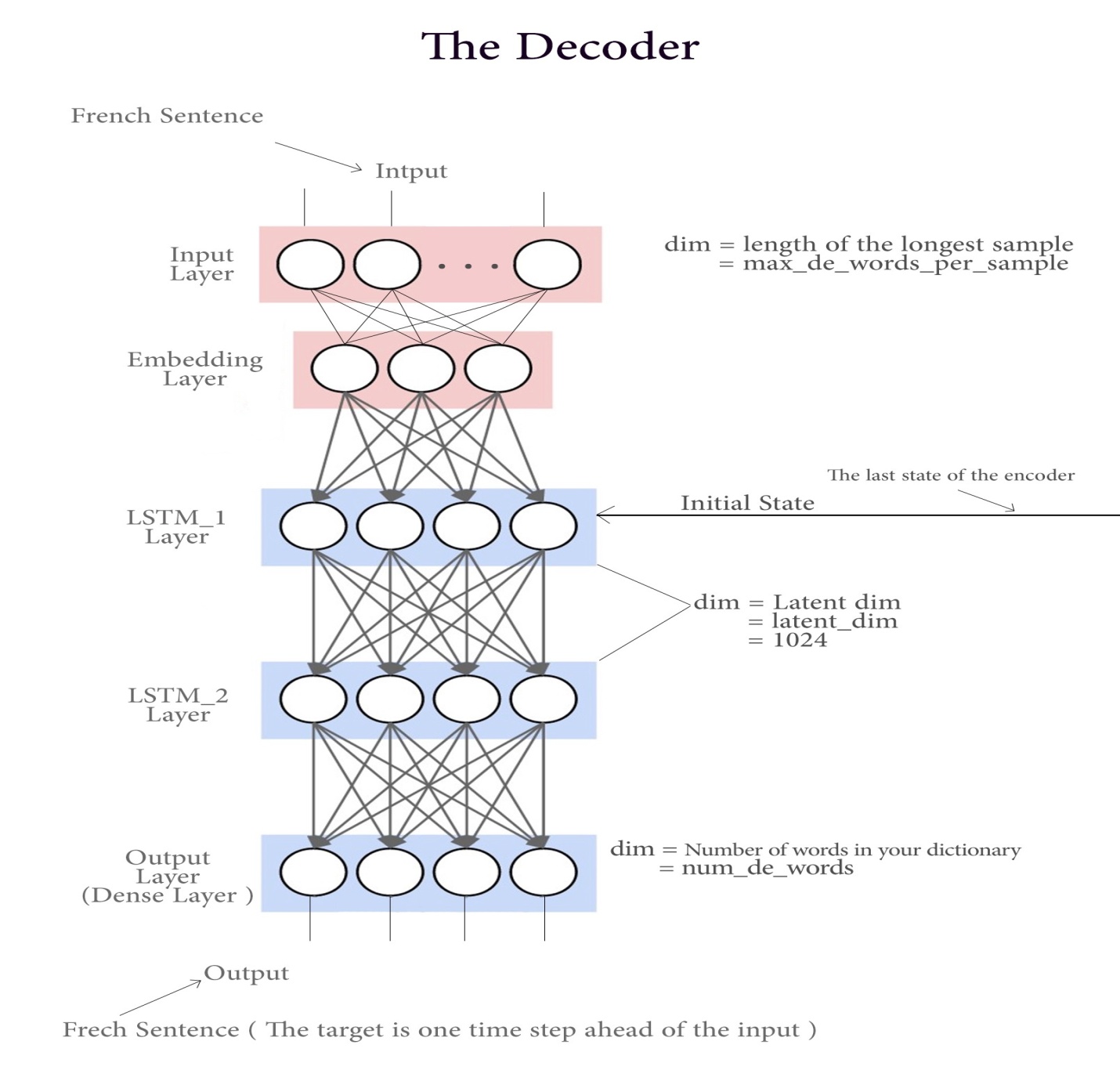


Building the decoder:



The decoder is made up of :

1. Input Layer : Takes the French sentence and pass it to the embedding layer.
2. Embedding Layer : Takes the French sentence and convert each word to fixed size vector
3. First LSTM Layer : Every time step, it takes a vector that represents a word and pass its output to the next layer, but here in the decoder, we initialize the state of this layer to be the last state of the last LSTM layer from the decoder .
4. Second LSTM Layer : Processing the output from the previous layer and passes its output to a dense layer .
5. Dense Layer (Output Layer) : Takes the output from the previous layer and outputs a one hot vector representing the target French word.



RNN (Recurrent neural networks)

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A vanilla neural network takes in a fixed size vector as input which limits its usage in situations that involve a ‘series’ type input with no predetermined size.

Recurrent Neural Network remembers the past and it’s decisions are influenced by what it has learnt from the past. Note: Basic feed forward networks “remember” things too, but they remember things they learnt during training. For example, an image classifier learns what a “1” looks like during training and then uses that knowledge to classify things in production.

While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It’s part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a “hidden” state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

In summary, in a vanilla neural network, a fixed size input vector is transformed into a fixed size output vector. Such a network becomes “recurrent” when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. There is no pre-set limitation to the size of the vector. And, in addition to generating the output which is a function of the input and hidden state, we update the hidden sate itself based on the input and use it in processing the next input.

* Deep RNNs

While it’s good that the introduction of hidden state enabled us to effectively identify the relationship between the inputs, is there a way we can make a RNN “deep” and gain the multi level abstractions and representations we gain through “depth” in a typical neural network?

Here are four possible ways to add depth.

(1) Perhaps the most obvious of all, is to add hidden states, one on top of another, feeding the output of one to the next.

(2) We can also add additional nonlinear hidden layers between input to hidden state

(3) We can increase depth in the hidden to hidden transition

(4) We can increase depth in the hidden to output transition. This paper by Pascanu et al., explores this in detail and in general established that deep RNNs perform better than shallow RNNs.

* Bidirectional RNNs

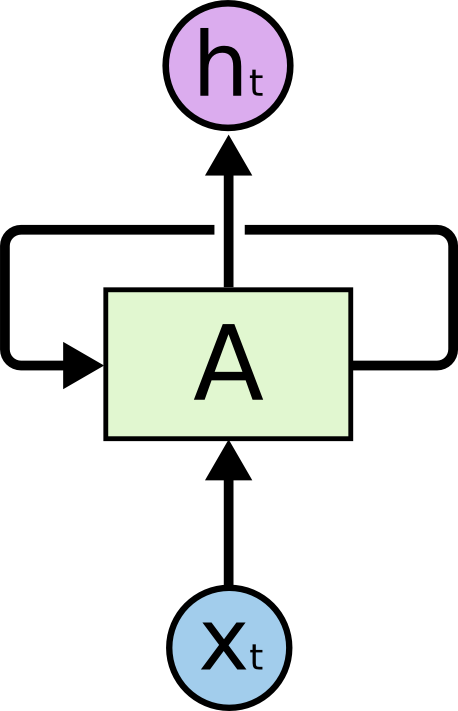
Sometimes it’s not just about learning from the past to predict the future, but we also need to look into the future to fix the past. In speech recognition and handwriting recognition tasks, where there could be considerable ambiguity given just one part of the input, we often need to know what’s coming next to better understand the context and detect the present.

This does introduce the obvious challenge of how much into the future we need to look into, because if we have to wait to see all inputs then the entire operation will become costly. And in cases like speech recognition, waiting till an entire sentence is spoken might make for a less compelling use case. Whereas for NLP tasks, where the inputs tend to be available, we can likely consider entire sentences all at once. Also, depending on the application, if the sensitivity to immediate and closer neighbors is higher than inputs that come further away, a variant that looks only into a limited future/past can be modeled.

Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

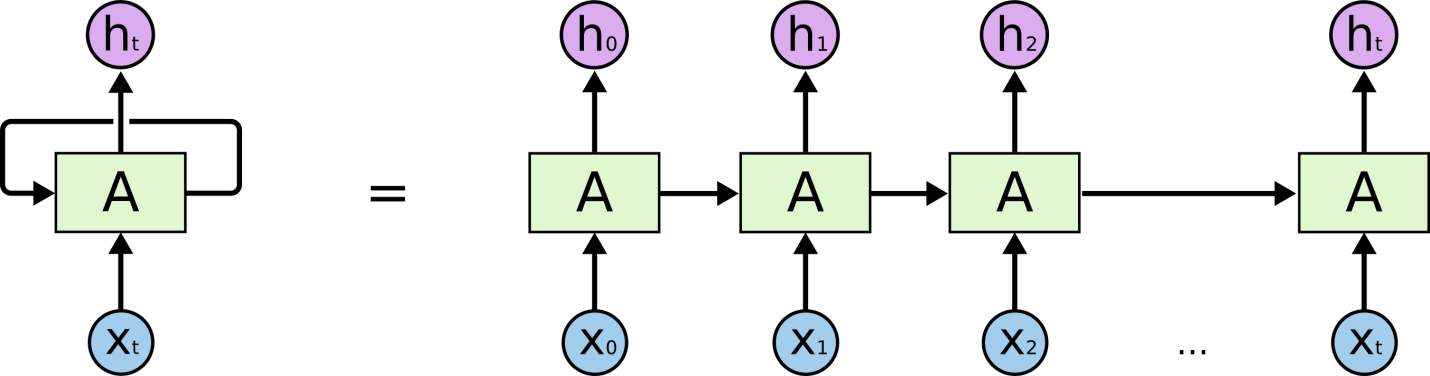
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

****

**Recurrent Neural Networks have loops.**

In the above diagram, a chunk of neural network, AA, looks at some input xtxt and outputs a value htht. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren’t all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

****

**An unrolled recurrent neural network.**

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They’re the natural architecture of neural network to use for such data.

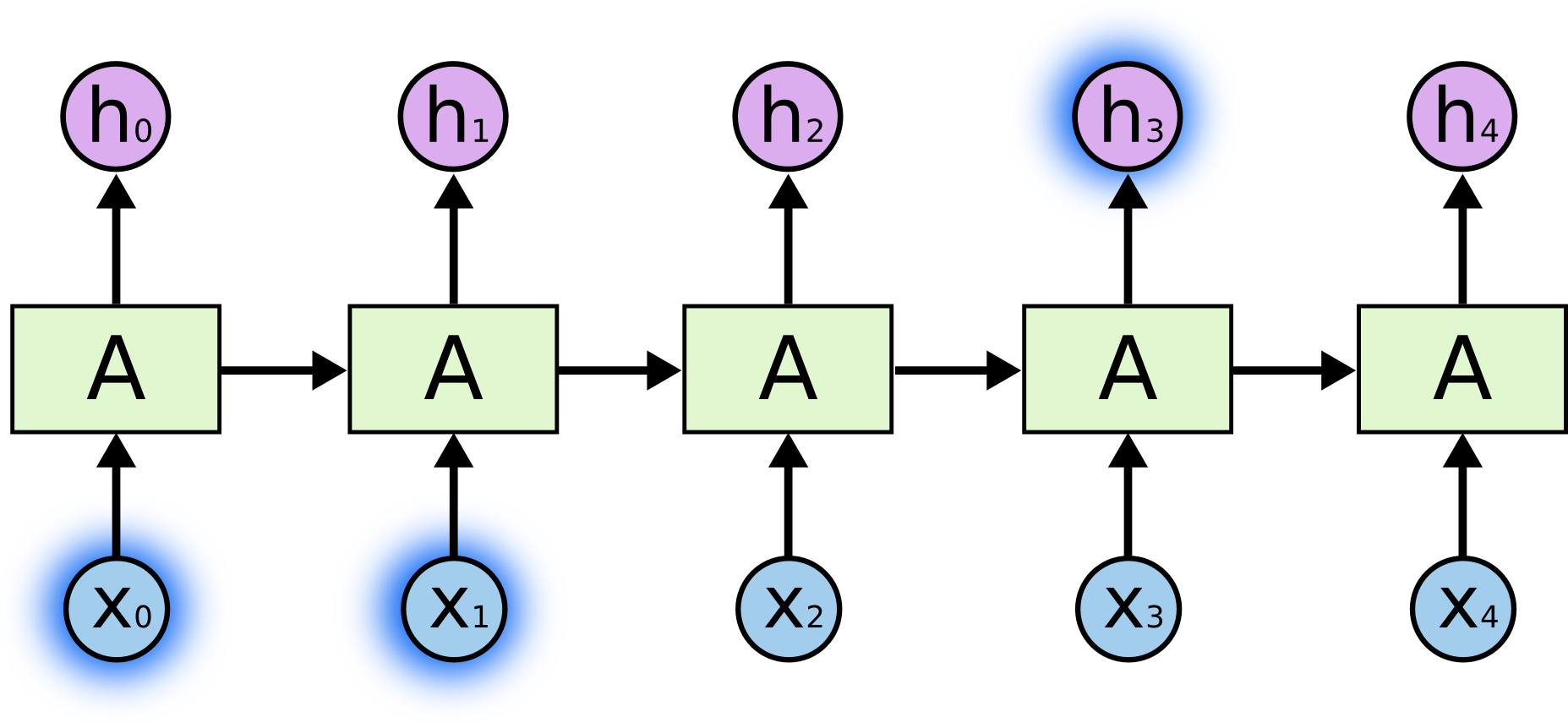
And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I’ll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy’s excellent blog post, [The Unreasonable Effectiveness of Recurrent Neural Networks](http://karpathy.github.io/2015/05/21/rnn-effectiveness/). But they really are pretty amazing.

Essential to these successes is the use of “LSTMs,” a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It’s these LSTMs that this essay will explore.

**The Problem of Long-Term Dependencies**

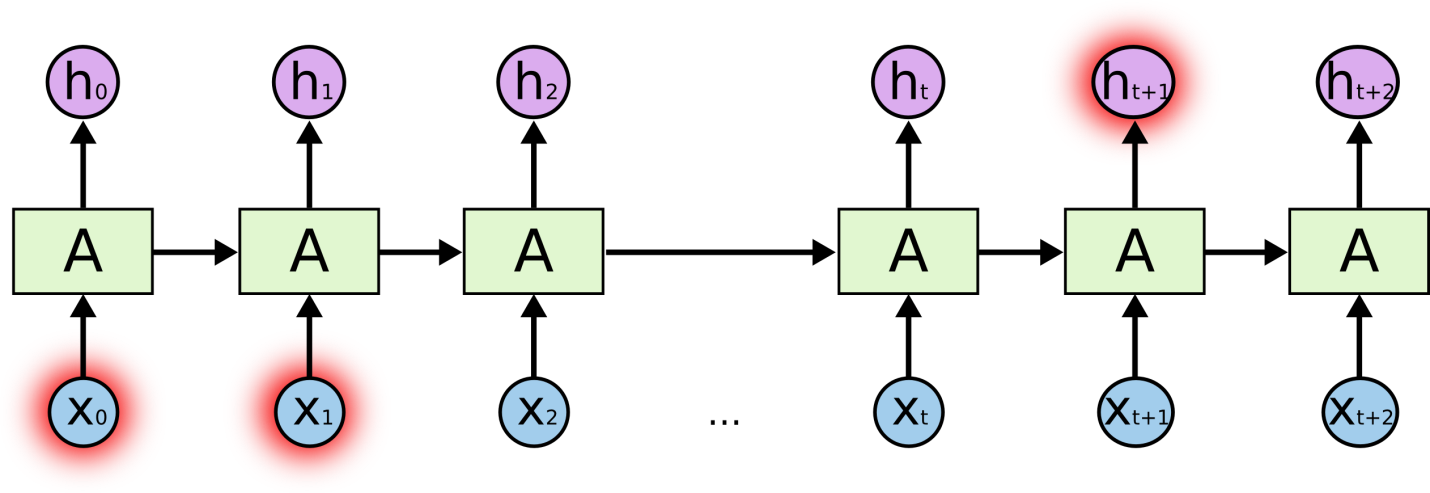
One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they’d be extremely useful. But can they? It depends.

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



In theory, RNNs are absolutely capable of handling such “long-term dependencies.”

Thankfully, LSTMs don’t have this problem!

LSTMs( Long Short term memory networks )

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by [Hochreiter & Schmidhuber (1997)](http://www.bioinf.jku.at/publications/older/2604.pdf), and were refined and popularized by many people in following work.[1](https://colah.github.io/posts/2015-08-Understanding-LSTMs/#fn1) They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.

An LSTM has three of these gates, to protect and control the cell state.

## Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at ht−1ht−1 and xtxt, and outputs a number between 00 and 11 for each number in the cell state Ct−1Ct−1. A 11 represents “completely keep this” while a 00 represents “completely get rid of this.”

Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.

It’s now time to update the old cell state, into the new cell state . The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by ftft, forgetting the things we decided to forget earlier. Then we add . This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.

Finally, we need to decide what we’re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that’s what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that’s what follows next.

**5.Related Work**

A chatbot implemented in TensorFlow based on the new sequence to sequence (NMT) model, with certain rules seamlessly integrated.The core of ChatLearner (Papaya) was built on the NMT model, which has been adapted to fit the needs of a chatbot.

Highlights and Specialties:

1. The Papaya Data Set for training the chatbot. We can easily find tons of training data online, but we cannot find any with such high quality.
2. The concise code style and clear implementation of the new seq2seq model based on dynamic RNN (a.k.a. the new NMT model). It is customized for chatbots and much easier to understand compared with the official tutorial.
3. The idea of using seamlessly integrated ChatSession to handle basic conversational context.

**Papaya Conversational Data Set**

Papaya Data Set is the best (cleanest and well-organized) free English conversational data we can find on the web for training a chatbot. Here are some details:

1. The data are composed of two sets: the first set was handcrafted, and we created the samples in order to maintain a consistent role of the chatbot, who can therefore be trained to be polite, patient, humorous, philosophical, and aware that he is a robot, but pretend to be a 9-year old boy named Papaya; the second set was cleaned from some online resources, including the scenario conversations designed for training robots, the Cornell movie dialogs.
2. The training data set is split into three categories: two subsets will be augmented/repeated during the training, with different levels or times, while the third will not. The augmented subsets are to train the model with rules to follow, and some knowledge and common senses, while the third subset is just to help to train the language model.
3. **Conclusion and Future Scope**

In conclusion, the real magic behind LSTM networks is that they are achieving almost human-level of sequence generation quality, without any magic at all.LSTMs were a big step in what we can accomplish with RNNs. It’s natural to wonder: is there another big step? A common opinion among researchers is: “Yes! There is a next step and it’s attention!” The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, do exactly this – it might be a fun starting point if you want to explore attention! There’s been a number of really exciting results using attention, and it seems like a lot more are around the corner…

**References**

1. <https://www.expertsystem.com/chatbot/>
2. <https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/>
3. <https://medium.com/@BhashkarKunal/conversational-ai-chatbot-using-deep-learning-how-bi-directional-lstm-machine-reading-38dc5cf5a5a3>
4. <https://medium.com/datathings/the-magic-of-lstm-neural-networks-6775e8b540cd>
5. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Plagiarism Checker X Originality Report

**Similarity Found: 5%**

Date: Saturday, November 6, 2019

Statistics: 150 words Plagiarized / 3074 Total words

Remarks: Less Plagiarism Detected - Your Document is Healthy.

-------------------------------------------------------------------------------------------

…………………………………………0( CHATBOT using Deep Neural Network Ankit Kumar, Shivam kumar,Utkarsh Srivastva, Abstract—A Chatbot is artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, and mobile apps or through the telephone. These Chatbots use deep learning models.

They use natural language processing models which use deep learning and then find at the bottom here we've got the sequence to sequence models which we'll be interested in at the end. INTRODUCTION A Chatbot is a service, which is mostly powered by artificial intelligence in which we interact with each other via a chat interface. The service has many applications ranging from functional to fun, and it is already present in any major chat products like Facebook, Messenger, Telegram.

**Generally speaking a Chatbot is any software that performs an automated task; however we are interested in the class of bots that live online in chat platforms or on social media called Chatbots. There are many possible definition of what a Chatbot is and it is perfectly fine because of so many use cases. One of the recent definitions is that it is a Chatbot which can have a conversation with human.**

For example a user could ask the Chatbot a question or give it an instruction and it could respond or perform an action as appropriate. Misconception This definition however often leads to two potential misconceptions. The first misconception is it has the ability to converse with a human like any other humans but this is wrong because it is simply not possible to achieve that using the current technology. It has been showcased in a lot of sci-fi films but it is still far from reality.

This also leads to sky high expectations and it leads to disappointment when these expectations are not met The second misconception is that a Chatbot communicates using only text or voice. This is not true because there are many instances of various Chatbots that not only use voice and text but also uses graphics to communicate with the user.

A large technological company like Facebook, Tencent and Alibaba uses graphical interfaces to interact with their customers. Difference with Applications There are a lot of differences between chatbot as an user interface or used as an app. A Chatbot however can be differentiated from an app in the way that the interactions with the bot take place, more or less sequentially (as a conversation), and the bot is used inside a chat app. A good metaphor for a big difference is human agent and its customers.

A Chatbot is different from an app as it has an “identity” that is actually separate from its interaction with the user. Likewise the agents exist independently when there is no engagement with the customer. This metaphor can be explained with the fact that a single Chatbot could interact with the customer over several different communication channels.

In short a Chatbot is another way of humans interacting with software. Although there are a lot of similarities between apps, websites and Chatbot and they do share most of the functionality But the interaction is very much different when done through a Chatbot. Nowadays messaging platforms are becoming universally accessible through mobile apps or app portals.

Businesses want to find ways to deliver their messages and services in the place that the consumers are which is on chat platforms. Chatbots are the Perfect platform to do this. Seq2Seq RNN Encoder-Decoder is the main go to model for Machine translation and dialogue systems which has introduced Sequence to Sequence model in learning phase representations. It consists of two RNNs: An Encoder and a Decoder.

The encoder takes a sequence say, sentence as input and processes one symbol say, word at each time step. It encodes only the important information in sequence by converting a sequence of symbols into a fixed size vector while losing the unnecessary information. 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. When we are using this model, there are a lot of challenges.

One of the most challenging part is it cannot handle different variable length sequences. Another challenging part is the vocabulary size. The decoder has to run softmax over a large vocabulary of say 20,000 words, for each word in the output. Even though the hardware is capable, but it usually slows down the training process. Representation of words is of great importance.

But how do we represent the words in the sequence? Use of one-hot vectors means we need to deal with large sparse vectors due to large vocabulary and there is no semantic meaning to words encoded into one-hot vectors. Let’s look into how we can face these challenges, one by one. / / RNN We want to take an example of any sequential data, which can be the stock market’s data for a particular stock.

A number of features are used for a simple machine learning model of an Artificial Neural Network may learn to predict the stock prices based on a number of features: the volume of the stock, the opening value etc. While the price of the stock depends on these features, it is also largely dependent on the stock values in the previous days.

In fact for a trader, these values in the previous days (or the trend) is one major deciding factor for predictions. That is when fitting the model for a particular day, there is no consideration for the stock prices on the previous days. In the conventional feed-forward neural networks, all test cases are considered to be independent. This dependency on time is achieved via Recurrent Neural Networks.

A typical RNN looks like: / This may be intimidating at first sight, but once unfolded, it looks a lot simpler: / Now it is easier for us to visualize how these networks are considering the trend of stock prices, before predicting the stock prices for today. Here every prediction at time t (h\_t) is dependent on all previous predictions and the information learned from them.

Sequence handling can be done by RNNs to a large extent but not entirely. To be able to build a story and remember it, we need our models to be able to understand and remember the context behind the sequences, just like a human brain. This cannot be done by RNN because RNNs are great when it comes to short contexts.

Limitation of RNN As we have seen Recurrent Neural Networks work just fine when we are dealing with short-term dependencies. Most of the times RNNs fail to understand the context behind an input. When we are making prediction in the present Something that was said long before, cannot be recalled .Let’s understand this as an example: Here, we can understand that since the writer has worked in India for 2 years, it is very likely that he may possess a good command over Hindi. But, to make a proper prediction, the RNN needs to remember this context.

What happens is the main information may get diluted because of the irrelevant data. This is where a Recurrent Neural Network does not work Vanishing Gradient is the reason behind this kind of problem. An existing knowledge of how a feed forward neural network learns needs to be looked into in order to understand this. According to our knowledge for a conventional feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer.

So it results in a product of all previous layer errors which gives the error term for a particular layer. There are many activation functions but in this case we took example of Sigmoid function the small values of its derivatives which occurs in the error function gets multiplied multiple times as we move towards the starting layers.

Due to this result, it becomes very difficult to train these layers as the gradient almost disappears as we move towards the starting layers. A similar case is observed in Recurrent Neural Networks. RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be again produced, but once a lot of words are feed into it, this information gets lost somewhere.

This issue can be resolved by applying a slightly tweaked version of RNNs – the Long Short-Term Memory Networks. LSTM One of the things which we do is that most often than not we arrange our calendar for the day, and also we try to prioritize our appointments. Suppose urgency comes we need to make some time and space for it so we mostly know which meetings we can cancel to make time for that possible meeting.

But we find out that a RNN doesn’t do so. If we have to order new information or add onto the existing ones, we apply a function that transforms the existing information completely. Due to this, the entire information gets modified i. e. there is no consideration for what important information is and what not so important information is.

LSTMs on the other hand, make small modifications to the information by multiplications and additions. In LSTMs, there is a mechanism through which all the information flows which is known as cell states. This way, LSTMs can selectively remember or forget things. There are three different dependencies for information at a particular cell state. We’ll visualize this with an example.

Let’s take the example of predicting stock prices for a particular stock. The stock price of today will depend upon: The trend in which the stock has been trading in the previous days, maybe a downtrend or an uptrend. The price of the stock also depends on the previous day, because there are many traders who compare the stock’s previous day price before thinking of buying it.

There can be a number of factors that can affect the price of the stock in the present day. This can be a new company policy that is being criticized widely, or a drop in the company’s profit, or maybe an unexpected change in the senior leadership of the company.

These dependencies can be generalized to any problem as: The previous cell state  The previous hidden state  The input at the current time step  Industries use these dependencies to move products around for different processes and LSTMs use this mechanism to move information. When it moves through different layers we can do some addition, modification or removal of information, just like the same way product may be molded, painted or packed . The following diagram explains the close relationship of LSTMs and conveyor belts.

/ Although this diagram is not even close to the actual architecture of an LSTM, it solves our purpose for now. We do not manipulate the entire information but rather modify them slowly and slightly, they can forget and remember things selectively due to this property of Lstms Architecture of LSTMs By understanding the functioning of a news channel’s team covering a murder story we can visualize the functioning of LSTM.

Whenever a new event occurs we take any of the three steps. Let’s say, we are trying to assume that the murder was done by giving poison to the victim, but the autopsy report that just came in said that the cause of death was due to an impact on the head. We immediately try to forget the previous cause of death and all stories that were revolving around this fact.

What can happen if an entirely new suspect is introduced into the picture. A person who had problems with the victim could be the potential murderer. We try to input this information. After a certain time interval, we need to summarize this information which cannot be served by mainstream media these broken pieces of information.

So, to output the relevant things. Now let’s get into the details of the architecture of LSTM network: / . A typical LSTM network is comprised of different memory blocks called cells the rectangles that which we are trying to see in the image.  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.  Applications First one is speech recognition. So I've got to put them in order of importance in order how many and how often and impactful it is for a lot of lives. These are world changing applications. So speech transcription is a huge one.

Whenever we talk into wire phone and try it out as text that is specious script ion so taking in audio form and putting it into text. But not only does it stop there. Remember previously on the Get excited story we talked about some of the chat boards that are or that are like Siri or town or Alexa that are voice based. Well that same technology goes in there to understand the wave forms and some of the sound we can apply a deep natural language processing and then Reiji put in that second part which is the chat board in order to understand what to do with that text what to do with those instructions.

**So speech transcription as it can that is it's already very important is going to become more and more and more important in the years to come. Proposed work We will build a Chatbot by implementing a state of the art deep NLP model which will be the Seq2Seq and we will implement that with one of the best API which is tensorflow. So this implementation will be done in five steps which are the common five steps when implementing a deeper application or an AI.**

We follow a step by step approach. Step 1: Getting the Dataset Step 2: Data Preprocessing Step 3: Building the Seq2Seq model Step 4: Training the Seq2Seq model Step 5: Testing the Seq2Seq model Getting the Dataset . We will get the data set which is the Cornell movie corpus.

It is a data set of more than 600 movies containing thousands of conversations between lots of characters. And we want to train our model. The model can also be trained on other datasets for some other purposes like for example we will be able to train the same chatbot on a more specific dataset like a calendar assistant or a navigation assistant. There are some more specific applications but this is not what we want do.

We will try a general chatbot to talk about everyday conversations and that’s why movies are perfect because in movies we have a lot of random conversations general conversations between friends. Data Preprocessing Data processing is inevitable whenever we build an AI or whenever we build a machinery model we have to make the data set compatible with the model we're going to build. We're going to build a neural network based model and therefore the data will have to have a special format especially for the inputs.

Besides we'll have to clean the text because the less we clean it and simplify it the more difficult it will be for a model to train itself to talk like a human. We want it do it the most efficiently so that we can get to step 3, 4 and 5. Building the Seq2Seq model We will be building the Seq2Seq model which is a state of the art deep NLP model. So we will build it.

We will actually build a brain composed of an encoder and then a decoder and we will assemble all of them to build the final brain which has not been trained yet. Training the Seq2Seq model We will train the model. We will set up a last function to get the optimizer and then apply some to get a grade in the center to update the weight of the neurons of the brain so that it improves its ability to talk with us.

Testing the Seq2Seq model We finally test the model to know once we executed have an interface where we can ask some questions and then the chatbot will answer and we just test the Chatbot by observing its answers and see how is capable of conversing with us. RELATED work A chatbot implemented in TensorFlow based on the new sequence to sequence (NMT) model, with certain rules seamlessly integrated.The core of ChatLearner (Papaya) was built on the NMT model, which has been adapted to fit the needs of a chatbot.

Highlights and Specialties: The Papaya Data Set for training the chatbot. We can easily find tons of training data online, but we cannot find any with such high quality. The concise code style and clear implementation of the new seq2seq model based on dynamic RNN (a.k.a. the new NMT model).

It is customized for chatbots and much easier to understand compared with the official tutorial. The idea of using seamlessly integrated ChatSession to handle basic conversational context. Papaya Conversational Data Set Papaya Data Set is the best (cleanest and well-organized) free English conversational data we can find on the web for training a chatbot.

Here are some details: The data are composed of two sets: the first set was handcrafted, and we created the samples in order to maintain a consistent role of the chatbot, who can therefore be trained to be polite, patient, humorous, philosophical, and aware that he is a robot, but pretend to be a 9-year old boy named Papaya; the second set was cleaned from some online resources, including the scenario conversations designed for training robots, the Cornell movie dialogs.

INTERNET SOURCES:

-------------------------------------------------------------------------------------------

5% - <https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/>