ARCHITECTURAL INSIGHTS: A JOURNEY IN CHATBOT DEVELOPMENT

DISSERTATION

submitted in partial fulfillment of the requirements

for the award of the degree of

Master of Science

in

MATHEMATICS AND COMPUTING

by

SEJAL

(302203010)

Under the guidance of

DR. KAVITA

DEPARTMENT OF MATHEMATICS
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA - 147004

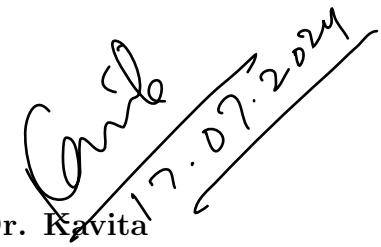
July 2024

CERTIFICATE

This is to certify that the thesis entitled "ARCHITECTURAL INSIGHTS: A JOURNEY IN CHATBOT DEVELOPMENT", being presented in partial fulfillment of the requirements for the award of the degree of Masters of Science in Mathematics and Computing and submitted to the School of Mathematics(SOM), Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Kavita. The matter presented in this thesis has not been submitted for the award of any other degree from this or any other institute.

(Sejal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.



Dr. Kavita
Supervisor
SOM, Thapar Institute of Engineering and Technology, Patiala.

ACKNOWLEDGEMENT

The completion of this thesis signifies the culmination of my rewarding pursuit of a Masters degree. I am deeply grateful for the support and guidance I have received from numerous individuals throughout this meaningful journey. This is my opportunity to express my sincere appreciation to all those who have contributed to my success. I would like to express my sincere gratitude to **Dr. Kavita**, my supervisor, for her invaluable guidance and unwavering support throughout the course of this thesis. Her profound insights and clarity of thought have significantly influenced this work. I am truly thankful for her encouragement in refining my interests and ideas. I would also like to extend my heartfelt appreciation to **Dr. Mahesh Kumar Sharma**, Head of SOM at Thapar Institute of Engineering and Technology, Patiala, for his consistent support. I am grateful to the entire faculty and staff of the School of Mathematics at Thapar Institute of Engineering and Technology for their unwavering assistance. I want to give a big shoutout to my amazing parents for their love, support, and unwavering belief in me. They've been my rock and I wouldn't be where I am today without them. I also want to thank my awesome sister for always having my back and giving me the encouragement I needed. I sincerely express my gratitude to the divine for the love and blessings bestowed upon me.

July 2024 **Sejal**

Place: Patiala **Roll No.: 302203010**

ABSTRACT

This thesis delves into a detailed examination of the effectiveness of various neural architectures for the development of chatbots. The primary focus is on exploring the capabilities of Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRUs), and Transformers. Through an in-depth review and comparative analysis, the research endeavors to shed light on the specific strengths, weaknesses, and practical implications of each architecture within the domain of chatbot development. Furthermore, the study delves into the process of implementing chatbots using these architectural models. In addition, the thesis provides a comprehensive case study that leverages Dialogflow, a widely utilized conversational AI platform, to showcase the practical application of these diverse neural architectures in real-world chatbot development scenarios.

LIST OF FIGURES

CONTENTS

1. BACKGROUND OF THE STUDY

1.1 Introduction to Chatbots

Chatbots serve as highly proficient digital assistants, adept at providing customer support comparable to human interaction. Through advanced technologies like AI, machine learning (ML), and natural language processing (NLP), contemporary chatbots engage customers with an almost human-like finesse. Their expertise lies in effectively addressing routine inquiries, thereby allowing human agents to allocate their focus towards more complex issues. Moreover, should a situation become particularly challenging, chatbots can seamlessly transfer the interaction to a human agent, equipped with all the necessary information. This streamlined approach ensures swift and efficient resolution of customer concerns, harmonizing the interaction between automated and human support.

Notably, chatbots continuously learn from their customer interactions, thereby enhancing their ability to provide precise and appropriate responses. By analyzing customer data, they can discern patterns that contribute to overall experience im-

provement and inform strategic business decisions.

In essence, chatbots offer tremendous support for businesses seeking to elevate their customer service, streamline human workload, and ensure customer satisfaction. Leveraging their advanced capabilities in AI, ML, and NLP, chatbots excel in resolving routine customer queries while upholding a seamless and pleasant experience.

1.1.1 Brief History Of Chatbots

Joseph Weizenbaum developed ELIZA in 1966, which was the world's first chatbot, introduced before the development of the first personalized computer. ELIZA, developed at the MIT Artificial Intelligence Laboratory, processed keywords as input and triggered output responses according to a defined set of rules. This methodology of generating output still holds strong, as several chatbots use it today.

Kenneth Colby developed PARRY at Stanford University soon after ELIZA, which was designed to simulate an individual with distrustful paranoid or paranoid schizophrenia and was known as "ELIZA with attitude". Richard Wallace developed ALICE or Alicebot, inspired by ELIZA, in 1995. Despite its failure to pass the Turing test, ALICE maintained its position as one of the most established chatbots of its category and received multiple recognitions through the Loebner Prize, an annual competition for artificial intelligence, on numerous occasions.

In 2001, SmarterChild, an innovative conversational agent de-

veloped by ActiveBuddy, Inc.(now Colloquis), was introduced as a feature on AOL Instant Messenger and MSN Messenger. Its purpose was to provide users with quick access to various information such as news, weather forecasts, and sports scores, taking inspiration from the growing popularity of instant messaging platforms like SMS.

The next significant milestone in the development of conversational agents came from a dedicated team at IBM working on the Watson AI project, which commenced its development in 2006. The ultimate goal for this agent was to compete and win on the American TV show Jeopardy!, a feat it successfully accomplished in 2011 by outperforming two of the show's former champions.

The early 2010s marked the emergence of virtual assistants, including Apple's Siri, Google Assistant, Amazon's Alexa, Microsoft's Cortana, etc. These virtual assistants introduced the notion of natural and goal-oriented conversations, representing a major shift in conversational agent technology. A significant development in the chatbot field occurred with the launch of the Messenger Platform for Facebook Messenger in 2016, enabling non-AI-related companies to create their own conversational agents.

1.1.2 Classification Of Chatbots

If you're considering implementing a chatbot, it's important to know that there are four different types available to suit your

specific needs. By choosing the right type of chatbot, you can provide a better customer experience and streamline your business operations. So, take the time to understand your users and their requirements, and choose the chatbot that will best meet those needs.

1. Text-To-Text Bot (TTT): It is a straightforward architecture that enables users to input text as words and sentences. The Chatbot responds with rule-based methods, allowing for a quick and efficient interaction. ELIZA was one of the pioneers of Chatbots and demonstrated the potential of TCTT. Currently, popular messaging apps such as Whatsapp, Facebook Messenger, and Telegram are utilizing TTT to facilitate two-user interaction services.

2. Text-To-Speech Bot (TTS): TTS is an advanced chatbot that is equipped with a unique feature of speech synthesis, enabling it to provide an interactive and purposeful experience to users. Its capabilities make it particularly beneficial for individuals who experience visual impairment, as they can listen to the bot's responses instead of reading them.

3. Speech-To-Text Bot (STT): STT technology generates text responses when users interact with a bot using their voice and view the answers. This technology is highly versatile and suitable for a variety of purposes. Siri, a state-of-the-art speech-to-text (STT) chatbot, is a prime example of innovative technology developed by Apple.

4. Speech-To-Speech Bot (STS): STS is a cutting-edge chatbot that has gained immense popularity among users. It is an excellent voice assistant that can be effectively utilized in the education sector for teaching purposes.

1.1.3 What if there were no chatbots?

The absence of chatbots would lead to many problems, and many tasks would have to be performed manually by humans, resulting in inefficiencies, higher costs, and longer wait times for customers to get their queries resolved. Customers would be required to download numerous apps from different vendors or wait longer to get answers to their questions. Businesses would inevitably have to hire more staff to handle customer inquiries; otherwise, they may lose potential sales and leads. Individuals would have to search for information themselves, which would make their daily lives less convenient and enjoyable. The importance of chatbots in today's world cannot be overstated, and their absence would significantly impact our lives.

Chatbots have several limitations and challenges, which we must acknowledge. Some of these include accuracy, domain knowledge, language support, privacy, and ethics. Although chatbots utilize AI and NLP to comprehend and address user inquiries, these technologies are continually advancing and enhancing. However, chatbots may not always provide the correct or relevant information, or they may misunderstand the user's intent or sentiment. Moreover, chatbots may not be able to han-

dle complex or specific queries that require expert knowledge or context. Additionally, chatbots may not support all the languages or dialects that users speak, or they may struggle with slang, idioms, or humor. It is important to note that chatbots may pose risks to the user's privacy or security, especially if they collect or store sensitive data without proper consent or protection. Ultimately, chatbots raise ethical questions about their impact on human relationships, society, and culture, which we must consider carefully.

Chatbots are not a replacement for human interaction, but a valuable complement and enhancement. They offer a range of advantages and benefits, but it is equally important to be aware of their limitations and drawbacks. While they can save us time, money, and effort, we must exercise caution, responsibility, and respect when engaging with them. Chatbots can certainly simplify our lives, but they also challenge us to think more deeply. They can be a source of amusement but inspire us to be creative. Ultimately, chatbots are not just tools, but vital partners in our digital world, and we should treat them as such.

1.2 Research Objective

The objective of this research is to present a comprehensive analysis of Sequence-to-Sequence models and the evolutionary path from Artificial Neural Networks (ANNs) to transformative models such as transformers. The study encompasses the evolution of chatbots in tandem with advancements in Natural Language

Processing (NLP). The analysis is supported by an extensive literature review, laying the groundwork for understanding the landscape of chatbot research. This thesis contributes novel insights into the performance of GRU and transformer models in chatbot development and underscores the value of integrating these models with platforms like Dialogflow for practical applications in real-world conversational AI systems.

Initially, attempts to develop a bespoke Sequence-to-Sequence model proved to be less productive. Consequently, the thesis relied on code authored by Inkawhich (2018) and created a chatbot using a transformer model. Subsequently, a domain-specific Question-Answering chatbot was developed using Dialogflow, exploring the underlying transformer model, 'DialoGPT'. This chatbot was tailored to provide support to students and parents by addressing a variety of admission-related inquiries specific to the Thapar Institute of Engineering and Technology.

1.3 Thesis Framework

In the first chapter, we thoroughly explored the fundamentals of chatbots and contemplated a world without their existence. Chapter 2 delves into the literature review, offering a comprehensive explanation of the development of models over time. Moving on to Chapter 3, we'll gain insight into the working of Sequence-to-Sequence models like Recurrent Neural Networks (RNNs), and delve into the advanced variations of RNN, including Long Short Term Memory (LSTM) and Gated Recurrent

Unit (GRU). Additionally, we'll delve into the encoder-decoder architecture and discuss the pivotal concept of the 'attention mechanism'. Chapter 4 provided an in-depth exploration of the revolutionary architecture known as Transformers, along with a discussion on potential future enhancements. Then, Chapter 5 begins with an introduction to the problem statement. The results of the chatbot, which was created using Gated Recurrent Unit by the author of the GitHub repository Inkawhich (2018) in an online tutorial, are presented. Subsequently, the process of creating a chatbot using DialogFlow is outlined, accompanied by a discussion of the results. Finally, the last chapter will give a comprehensive conclusion of the entire project. The chapter concludes with final remarks and recommendations for further reading for those interested in deeper exploration.

2. LITERATURE REVIEW

2.1 Introduction

The rapid progression of deep learning methodologies has significantly transformed the field of natural language processing (NLP), particularly in the realm of developing conversational agents, commonly referred to as chatbots. In recent years, chatbots have become increasingly prominent due to their applications in customer service, virtual assistants, and other domains. Researchers have explored various approaches to enhance chatbot capabilities, including the use of Recurrent Neural Networks (RNNs) and Transformers. Existing chatbot platforms such as Alexa, Siri, Cortana, and Google Assistant encounter challenges in accurately interpreting user intent, making it difficult to provide effective assistance. These chatbots often struggle to maintain context during lengthy conversations. Additionally, their capabilities are constrained to specific problem domains, limiting their ability to engage in coherent and compelling discussions on diverse topics such as current events, politics, and sports. Our research project delves into various artificial neural network (ANN) architectures for natural language processing (NLP) and examines the prevailing models used in chatbot development.

This literature review provides a foundational understanding of the key neural architectures in chatbot development and sets the stage for the comparative analysis presented in the thesis.

2.2 Evolution of Models

In recent years, researchers have explored various neural architectures to enhance the capabilities of chatbots in understanding and generating human-like responses. The evolution of these models can be categorized into distinct stages:

- Stage 1 (Recurrent Neural Networks (RNNs)): RNNs have been widely used in sequential data processing tasks due to their ability to maintain a memory state while processing input sequences. Introduced by Elman (1990), RNNs have shown promise in capturing time relationships in sequential data, making them suitable for tasks such as machine translation, language modeling, and sentiment analysis. Traditional recurrent neural networks (RNNs) are known to face the vanishing gradient problem, which hinders their capacity to effectively capture long-term dependencies within sequential data.

- Stage 2 (Long Short-Term Memory Networks): To address the limitations of traditional RNNs, Hochreiter and Schmidhuber (1997) proposed LSTM networks. Long Short-Term Memory (LSTM) networks incorporate a gating mechanism that enables the network to selectively update and discard

information over time, enabling better long-term memory retention. LSTMs have demonstrated superior performance in various NLP tasks, including text generation, question answering, and dialogue generation. Their ability to capture long-range dependencies makes them well-suited for chatbot development.

- Stage 3 (Gated Recurrent Units (GRUs)): GRUs, introduced by Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares, Schwenk, and Bengio (2014), are another variant of RNNs that aim to address the vanishing gradient problem. The architecture of GRUs simplifies that of LSTMs by consolidating the forget and input gates into a single update gate, leading to faster training and reduced computational complexity. While GRUs have shown comparable performance to LSTMs in several NLP tasks, they are more efficient in terms of training time and memory usage.

- Stage 4 (Encoder-Decoder Architecture):

  The first solution to solve the seq2seq tasks was encoder-decoder architecture. This solution was proposed in 2014 and was the first attempt to solve seq2seq tasks like Machine Translation. A seminal paper by Sutskever, Vinyals, and Le (2014) was published in which Encoder-Decoder architecture was introduced. In this architecture, the input sequence is passed to the Encoder which generates a summary and sends it to the state as a context vector which is then sent to Decoder for translation. The encoder-decoder architecture

is discussed in detail in section 2.14.

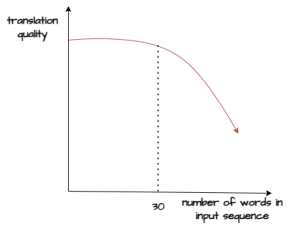


Fig. 2.1: Performance of Encoder-Decoder

- Stage 5 (Attention Mechanism):

  There was a major problem in Encoder-Decoder architecture that it was not able to successfully translate the long sequences in input (generally sentences more than 30 words) or it just changed the sentiment behind the long sentences. So to overcome this problem, a paper by Bahdanau, Cho, and Bengio (2014) was written. Attention Mechanism was first introduced in this paper. This model is very similar to the encoder-decoder architecture. The only difference this model has is in the decoder. Also, there is no single context vector. At any timestamp, the decoder has information regarding all the internal states of the encoder. Attention is a mechanism that finds the useful word from the encoder for any word being translated in the decoder.

- Stage 6 (Transformers):

  The next solution to sequence to sequence tasks was the transformers. The biggest problem in the attention mechanism was the quadratic computation complexity and the

higher training time. Different attention mechanisms were introduced from 2015 to 2017 to solve this problem. Later, researchers realized that the main problem was not in the attention mechanism but in the LSTMs used in the attention mechanism. Since LSTMs work essentially in sequential order that means one word at one time. Now the main goal was to remove this sequential nature of encoder-decoder architecture and introduce parallel processing because, through parallel processing, the training time can be reduced drastically. Thus, this was the time when transformers came into the picture and changed the entire landscape of NLP. In their paper, Vaswani et al. (2017) discussed a groundbreaking architecture called Transformer. This architecture completely ditched the use of LSTM or any other RNN cell and attention is all that was needed.

- Stage 7 (Transfer Learning):

  Transformers have represented a significant technological advancement. However, one of the primary challenges has been the complexity associated with training a transformer from the ground up, stemming from the following constraints:

  - Hardware: To train a transformer model from scratch, it is essential to have high-quality GPUs. However, acquiring such GPUs can be quite expensive.
  - Time: Time taken for training is a crucial factor. However, transformers are trained faster than other archi-

tectures like encoder-decoder or attention mechanism. However, still, it takes a significant amount of time for training and also the time depends upon the dataset.

– Data: The biggest problem for training a transformer from scratch is data. We need a lot of data. Suppose, we are doing a project on sentiment analysis and we have trained a transformer from scratch on the dataset of 1000 rows then our transformer model may not give good results due to less data.

Thus, these factors became restrictions for such a groundbreaking architecture, and not everybody was able to use this model. Hence we reach our next stage which is 'Transfer Learning' which was introduced to tackle the above-discussed restrictions. In 2018, the famous research paper by Howard and Ruder (2018) was published. This paper proposed that the concept of Transfer Learning can be used in NLP. Transfer Learning (TL) entails the utilization of previously acquired knowledge from one task to enhance performance on a related task. In the context of image classification, the knowledge attained during the process of learning to identify cars can be leveraged to improve the recognition of trucks.

Here, a question arises: Why Transfer Learning was not being used in the domain of NLP before 2018? There were several reasons why Transfer Learning was not being used in the domain of NLP before 2018. Let us explore:

- Data Availability and Size: NLP tasks require substantial amounts of labeled data for training. Before 2018, large-scale labeled NLP datasets were scarce, limiting the effectiveness of transfer learning.
- Task-Specific Models: Researchers often designed task-specific models from scratch. These models were tailored to specific NLP tasks, which made them less adaptable to other domains.
- Lack of Pre-trained Representations: Pre-trained word embeddings (Word2Vec and GloVe) existed but were limited to capturing word-level semantics. Comprehensive pre-trained representations for entire sentences or documents were not widely available.
- Computational Constraints: Training deep neural networks demands considerable computational resources to achieve optimal performance and groundbreaking results. Before 2018, training large-scale models on massive amounts of data was computationally expensive.

  However, the landscape changed dramatically after 2018. Here's why:

- Emergence of Transfer Learning Methods: Over the last few years, transfer learning methods specifically designed for NLP tasks emerged. These methods allowed models to leverage data from additional domains or tasks, leading to better generalization properties.
- Pretrained Language Models: The introduction of pre-

trained language models (such as BERT, GPT, and Roberta) revolutionized NLP. These models were pre-trained on large corpora and learned rich contextual representations. Fine-tuning these pre-trained models on downstream tasks significantly improved performance.

- Availability and Integration: Pretrained models became widely available, making it easier for researchers and practitioners to adopt them.

- Stage 8 (Large Language Models(LLMs)): In October 2018, two famous transformer-based language models were released:

  1. The first one by Devlin, Chang, Lee, and Toutanova (2018) called 'Bidirectional Encoder Representations from Transformers(BERT)' which is a deep learning model for natural language processing that improves the understanding of the meaning of queries related to Google Search.
  2. The other was 'Generative Pre-Trained Transformer(GPT)' by Radford, Narasimhan, Salimans, Sutskever et al. (2018) which refers to a family of machine learning models that utilize the transformer architecture to generate human-like text. GPT models are widely used in various applications, including language translation, content creation, and chatbots like ChatGPT.

  The dataset used to train these models was huge and these models were good at the task of transfer learning. These

were allowed for fine-tuning and could be used for any kind of task like sentiment analysis, text summarizing, parts of speech tagging, etc. The only difference between these two models is that the BERT is Encoder-only architecture and GPT is Decoder-only architecture.

2.3 Comparative Analysis

Several studies have compared the performance of RNNs, LSTMs, GRUs, and Transformers in the context of chatbot development. While traditional RNNs struggle with capturing long-term dependencies, LSTMs, and GRUs offer better memory retention capabilities. However, Transformers have emerged as the preferred choice for chatbot development due to their ability to model global dependencies and facilitate parallel processing. Comparative studies have shown that Transformers outperform traditional RNN-based architectures in terms of both performance and computational efficiency.

In conclusion, the choice of neural architecture plays a crucial role in the development of chatbots. While RNNs, LSTMs, and GRUs have been instrumental in advancing the field of NLP, Transformers have emerged as the state-of-the-art architecture for chatbot development. By leveraging self-attention mechanisms, Transformers offer superior performance in capturing long-range dependencies and modeling context across input sequences.

3. RECURRENT NEURAL NETWORKS

3.1 Introduction to RNNs

Recurrent Neural Networks (RNNs) are artificial neural networks that are particularly suitable for dealing with sequential data. They are widely used in natural language processing (NLP) tasks such as machine translation, language modeling, and sentiment analysis. The basic idea behind recurrent neural networks (RNNs) is to use feedback connections, enabling the network to maintain a "memory" of past inputs. This allows the network to process sequences of inputs of arbitrary length and to learn patterns in the data that depend on the order in which the inputs are presented.

Understanding RNNs is crucial for developing more advanced language models such as the GPT-2 (Generative Pre-trained Transformer 2) and transformer models. These models have revolutionized the field of NLP by achieving state-of-the-art results on a wide range of language tasks. To build effective models using these techniques, it is important to have a solid foundation in the underlying principles of RNNs.

Now, a fundamental question arises: 'Why do we need RNNs?' Let us delve into this matter. Artificial neural networks (ANNs) are a type of deep learning algorithm that is designed to work with fixed input sizes. However, when it comes to processing sequences of data, such as text or time series data, ANNs are not an ideal choice. This is because the length of sequences can vary greatly, whereas ANNs require fixed input sizes.

To overcome this limitation, zero padding can be used with ANNs, but this can be computationally expensive and not very efficient. Additionally, ANNs are not capable of storing the meaning of a sentence as they process all the data at once as input.

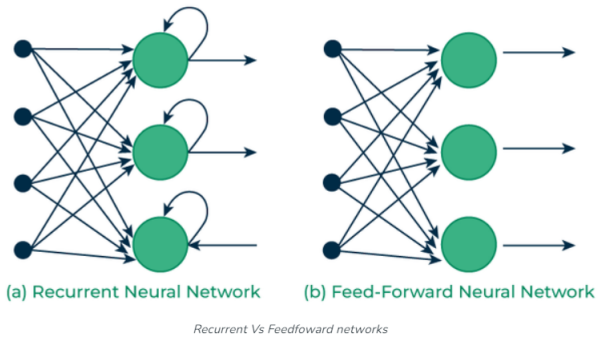


Fig. 3.1: RNN Vs ANN

On the other hand, RNNs are a type of deep learning algorithm that can process sequential data and are designed for tasks such as natural language processing and speech recognition. One of the key features of RNNs is their ability to remember past inputs, which makes them well-suited for learning patterns in sequential data. This feature is achieved through the use of memory cells, which store information about previ-

ous inputs and update themselves as new inputs are processed. Therefore, RNNs are a better choice for processing sequential data than ANNs.

The following types of Recurrent Neural Networks (RNN) are worth noting for comprehensive understanding:

1. Many to One: This is an architecture that accepts sequences like sentence, character, and time-series data in input and provides a non-sequential output like integers or numbers. Sentiment Analysis is one of the examples in which input is a sequence of words and the output we get is 1 or 0. Rating Prediction is another example of this architecture.

2. One to Many: In this architecture, the input is normal non-sequential data like numbers or images and the output is sequential like sentences, and words. This architecture is opposite to the many-to-one architecture. The most common application of this architecture is image captioning. In this, we give non-sequential data like images to the neural network and the model has to give a textual representation of what is inside that image. Music Generation is another example.

3. Many to Many: In this architecture, the sequential data is sent as an input, and the output is also sequential. Because of this, the architecture is also known as the Seq2Seq model. This architecture is further divided into two types:

    - Same length many to many: In this, the size of the In-

put sequence is the same as the size of the Output sequence. One example of this architecture can be Parts of speech tagging in which we are given a sentence and have to identify the parts of speech of that sentence. Another example could be Name Entity Recognition which is used in making chatbots.

- Variable length many to many: In this, the size of the Input sequence is not the same as the size of the Output sequence. Example: Machine Translation. In this application, a sentence of one language is to be converted to a sentence of another language.

4. One to One: This type of RNN behaves the same as any simple Neural network like ANN or CNN. It is also known as the Vanilla Neural Network. In this Neural network, there is only one input and one output. Image Classification is one of the examples in which there is an image as input and output is a scalar quantity (1 or 0).

3.1.1 Working of RNNs

RNN can be imagined as a box that takes two inputs and provides two outputs (see **Figure 3.2**). The first input is x_{it}, where i is the row number and t represents timestep. The second input is O_{t-1} which is the output of the previous timestep. These two inputs are added inside the RNN that is,

$$x_{it}w_i + O_{t-1}w_h$$

where w_i and w_h are weight matrices.

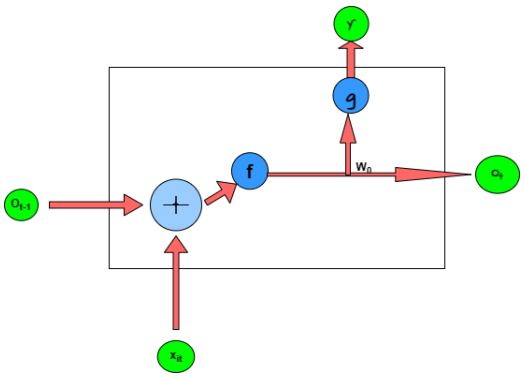


Fig. 3.2: RNN

The output of this operation is a vector which is then sent to an activation function that is,

$$f(x_{it}w_i + O_{t-1}w_h)$$

This is tanh activation function that limits the output to values from -1 through 1. The tanh function keeps the output within reasonable values. This operation gives O_t which is the output at the current timestep. If this is the last timestep then O_t with the weight function w_0 is sent to an activation function which is generally a sigmoid function in case of binary classification problem. In the case of multi-class classification problems, we use the softmax function and in regression problems, we can use the linear function. After this operation, we get our final output, $\hat{y}$.

In RNNs, data is sent in the form stated below:
(number of timesteps, number of input features)
Let us understand this with a small example. Suppose we have

three reviews for a movie and we have to do sentiment analysis. The three reviews are like: 'Movie was good', 'Movie was bad', 'Movie was not good' After applying one-hot encoding for converting words to numbers, the three reviews will be sent in the form of (3,5), (3,5), and (4,5) to the RNN. In Keras, Simple RNN, the data would be inserted as: (batch size, number of timesteps, number of input features). In our example, the three reviews together will be inserted as (3,4,5).

3.1.2 Problems with RNNs

RNNs suffer from two major problems due to which they are not generally used on sequential data but we use other architectures like LSTMs.

1. Problem of Long-Term Dependency:

   Since RNNs are unable to retain long-term memory, they fail when a word at a particular timestamp depends on a word at an earlier timestamp in a long sequence or paragraph. This problem arises due to the vanishing gradient problem.

   In the context of Recurrent Neural Networks (RNNs), the multiplication of a floating-point number below 1.0 with RNN weights that are also below 1.0 results in a vanishing gradient, which approaches zero. The vanishing gradient is a significant challenge in training RNNs as it can lead to the saturation of activation functions and a subsequent reduction in the network's learning capacity. Understanding the vanishing gradient phenomenon is crucial for optimizing

the training of RNNs and improving their performance in various applications.

There are some solutions to reduce the above-discussed problem:

- We can use different activation functions like relu or leaky relu in place of the "tanh" function because the derivative of these functions is not in the range of 0 to 1.
- We can have better weight initialization techniques.
- We can use slightly different architectures like Skip RNNs but generally an advanced architecture known as LSTM is used.

2. Problem of Stagnated Training:

   Sometimes the RNN is not trained properly, that is, the RNN model does not give good results or the loss function may not get improve. The reason behind this problem is the Exploding gradient problem.

   The phenomenon of exploding gradient manifests itself when data that commences with a floating point value greater than 1.0 is multiplied with weights greater than 1.0, resulting in values that tend towards infinity. This can potentially impede the optimization process during deep learning. It is therefore imperative to mitigate this issue through careful selection of appropriate hyper-parameters to ensure that the model trains efficiently and effectively.

3.2 Long Short Term Memory

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) architecture that has been specifically designed to address the limitations of traditional RNNs when it comes to processing long sequences of data, such as those typically found in natural language processing tasks. The problem with traditional RNNs is that they can suffer from the vanishing/exploding gradient problem, which occurs when the gradients used to update the network's parameters become too small or too large, respectively, as they are propagated through the network over many time steps. This can lead to the network being unable to effectively learn long-term dependencies in the data, which is a critical requirement for many NLP tasks. LSTM networks, on the other hand, use a more sophisticated architecture that includes memory cells and gating mechanisms to selectively retain or discard information over multiple time steps, allowing them to effectively handle long sequences of data and learn complex patterns in textual data.

To understand the core idea behind LSTMs, consider the following situation: **'In ancient times, a powerful king had a formidable adversary named King Kaali. The two monarchs engaged in war, which resulted in the death of the former king, Vikram. Vikram had a son, Vikram Junior, who, like his father, became a powerful ruler. In his quest for revenge, he attacked Kaali, but was, unfortunately, killed in the process. Vikram Junior also had**

a son named Vikram Super Junior. When he came of age, he too sought to avenge his father and grandfather by waging war against Kaali. In a fierce battle, Vikram Super Junior emerged victorious, killing Kaali and thus fulfilling his family's desire for revenge.'

Our model has to predict whether the above-told story is good or bad. So, to predict the same, the model has to remember the entire paragraph where RNN fails as it has only one path which has both the responsibilities of long-term memory and short-term memory. The idea behind LSTM is that in this we introduce one path, especially for long-term memory, which will help to remember the entire story, no matter how long it is.

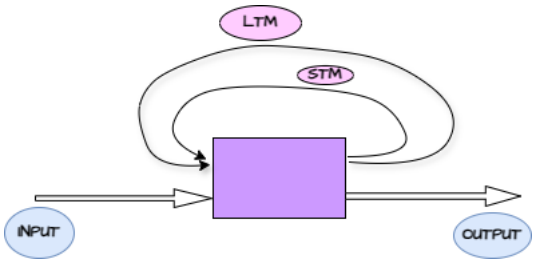


Fig. 3.3: LSTM

The main two differences in RNN and LSTM are:

- In LSTM, there are two states LTM(cell state) and STM(hidden state), whereas in RNN, we have only one state.
- In RNN, we use a simple architecture but in LSTM, a complex architecture is used which helps to establish communication between LTM and STM.

3.2.1 LSTM Architecture

In LSTM, three gates are introduced to limit the information that is passed through the cell state. They determine which part of the information is required for the next cell and which has to be discarded.

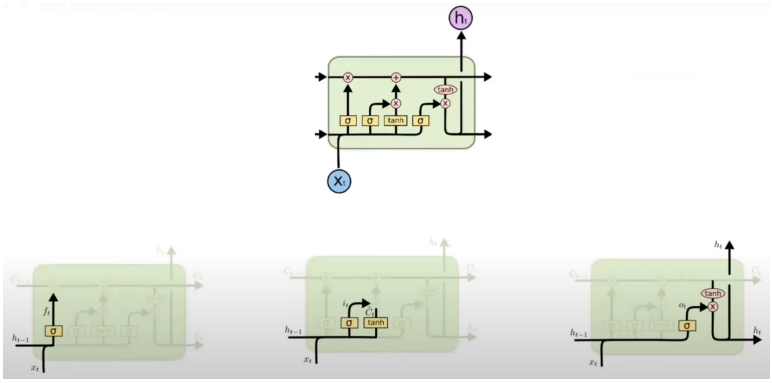


Fig. 3.4: LSTM Architecture- The three blocks are forget, input, output gate respectively: CampusX (2024)

The three gates are discussed below (see **Figure 3.4**):

- Forget Gate:

  It is used to remove the information that is no longer needed. Two inputs x_t and h_{t-1} called input at a particular time and previous hidden state respectively are given to the gate and pointwise multiplied with weight matrix followed by the addition of bias which then is passed through a sigmoid function.

- Input Gate:

  It helps to add some new important information to the cell state(LTM). Firstly, two inputs h_{t-1} and x_t are passed

through the sigmoid function, which filters the values to be remembered. Then a vector is created using the tanh function called candidate cell state, $\tilde{c}_t$, which contains all the possible values from h_{t-1} and x_t. At last, i_t and $\tilde{c}_t$ (both vectors) are pointwise multiplied to obtain the useful information.

- Output Gate:

  Through the output gate, the information is extracted from the current cell state to be presented as output in the form of the current hidden state, h_t. Firstly, the tanh activation function is applied to the cell state which gives a vector. Also, two inputs h_{t-1} and x_t are passed through the sigmoid function which again gives a vector, θ_t. At last, θ_t and $\tanh(c_t)$ are pointwise multiplied to be sent as an output (called current hidden state,h_t), which is input to the next cell.

3.3 Gated Recurrent Unit

Gated Recurrent Unit (GRU) introduced by Cho et al. (2014) is the RNN architecture used to process sequential data and is a simpler alternative to LSTMs. Like LSTM, GRU can process sequential data such as speech, text, and time-series data. Since LSTM architecture was complex and there were a large number of parameters involved, hence the time taken for training was more especially on large datasets. This was a big flaw of

LSTM architecture. In comparison to LSTM architecture, GRU consists of a simpler architecture and there are less number of parameters involved. Also in LSTM, we studied about three gates but in GRU we have only two gates namely the reset gate and the update gate. GRUs are effective in various NLP tasks, such as machine translation, and speech recognition.

The big idea behind GRU is that it does not require two separate states to carry long-term context and short-term context. That is why in GRU there is no cell state, there is only a hidden state that carries both long-term and short-term context. The LSTM recurrent unit's internal cell state information is integrated into the hidden state of the Gated Recurrent Unit, and this combined information is then propagated to the subsequent Gated Recurrent Unit.

3.3.1 GRU Setup

For any timestamp, the two inputs are given: the previous hidden state and the input at the current timestep. The goal of GRU is to find the hidden state at the current timestep. The

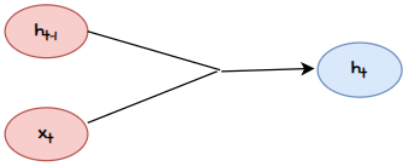


Fig. 3.5: h_{t-1} and x_t are inputs and h_t is output

formulas utilized to compute the reset gate, update gate, and hidden state of a Gated Recurrent Unit (GRU) are as follows:

Reset gate: $r_t = \text{sigmoid}(W_r * [h_{t-1}, x_t])$

Update gate: $z_t = \text{sigmoid}(W_z * [h_{t-1}, x_t])$

Candidate hidden state: $\tilde{h_t} = tanh(W_h * [r_t * h_{t-1}, x_t])$

Hidden state: $h_t = (1-z_t) * h_{t-1} + z_t * \tilde{h_t}$

where, W_r, W_z, and W_h are learnable weight matrices, x_t is the input at time step t, h_{t-1} is the previous hidden state, h_t is the current hidden state, r_t is reset gate, z_t is update gate and tilde(h_t) is candidate hidden state.

All of these are vectors and can be of any dimension. It is to be noted that all these vectors except x_t are always of the same dimension. The various gates of a Gated Recurrent Unit (GRU) are delineated as follows::-

Update Gate(z): It determines the amount of past knowledge that should be carried forward into the future. This can be likened to the Output Gate in an LSTM recurrent unit.

Reset Gate(r): This dictates the extent to which prior knowledge is to be disregarded. It can be likened to the amalgamation of the Input Gate and the Forget Gate in an LSTM recurrent unit.

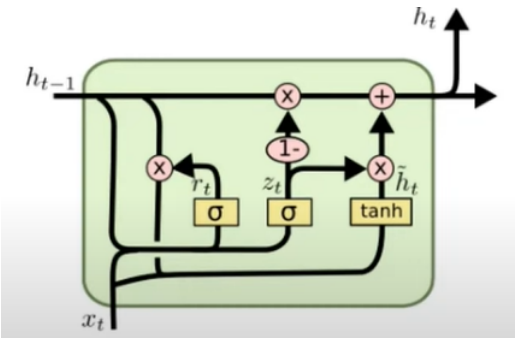


Fig. 3.6: GRU Setup: CampusX (2024)

In **Figure 3.6**, the yellow-colored boxes are neural network layers (fully connected layers) with some activation function. Note that the number of nodes in all these three boxes will be the same. Also, the number of nodes in these neural networks will be equal to the dimension of the vectors $h_{t-1}, h_t, r_t, z_t, \tilde{h}_t$. The circles with * and + represent pointwise multiplication and pointwise addition respectively.

3.3.2 Working of GRU

- Store the current input and the preceding hidden state as vectors.
- Calculate the values of the gates by following the steps given below:-
    1. To get things moving for each gate, we need to multiply the concerned vector with the respective weights for each gate. We obtain the parameterized current input and the hidden state vectors from the preceding time step. We do this by performing an element-wise multiplication which is also known as Hadamard Product.
    2. Apply the corresponding activation function element-wise on the parameterized vectors. Below are the gates with their respective activation function to be applied.

       Update Gate: Sigmoid Function

       Reset Gate: Sigmoid Function

- The calculation of the Candidate hidden state involves a distinct approach. Initially, we compute the Hadamard product of the Reset Gate and the previously hidden state vector. Then, we parameterize this vector and add it to the parameterized current input vector.

$$\tilde{h_t} = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$$

- In order to calculate the current hidden state, it is necessary to establish a vector of ones with identical dimensions as the input, referred to as "ones" and symbolically represented as 1. Subsequently, the Hadamard product of the update gate and the previously hidden state vector should be computed. Then, subtract the update gate from ones to create a new vector and calculate the Hadamard product of the new vector with the candidate hidden state. Finally, add the two resulting vectors to obtain the current hidden state vector.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h_t}$$

In **Figure 3.7**, the above-stated working of GRU is shown in the form of a mathematical flowchart:

3.4 Encoder Decoder Architecture

The Encoder-Decoder architecture for RNNs is the undisputed standard in neural machine translation, which has proven to

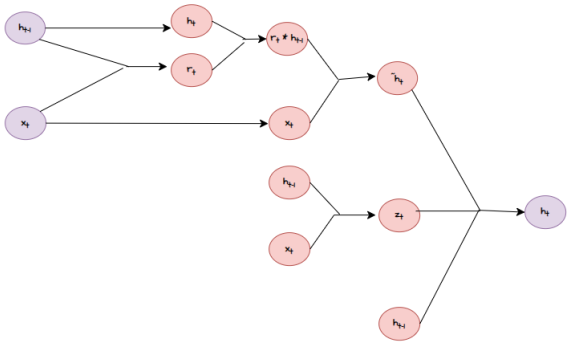


Fig. 3.7: Flowchart: Working of GRU

rival and often surpass classical statistical machine translation methods. This cutting-edge architecture was first introduced in 2014 and has since been adopted as the core technology inside Google's translation service, solidifying its position as the most effective and efficient method for machine translation available today.

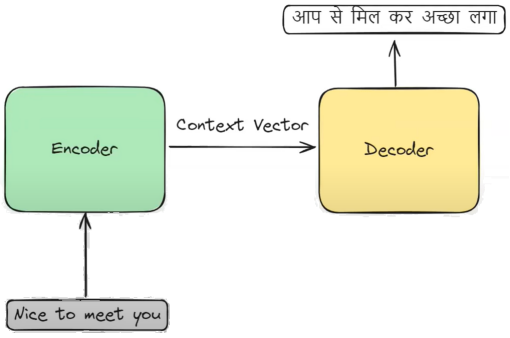


Fig. 3.8: Encoder and Decoder Flow: CampusX (2024)

There are three main blocks in the encoder-decoder model,

1. Encoder: The encoder receives an input sequence, for instance, a natural language sentence, and proceeds to generate a contextual representation. It compresses the input

sequence into a fixed-length vector (often called the hidden state or hidden vector). The encoder's job is to capture relevant information from the input sequence and summarize it in this hidden vector.

2. Hidden Vector: The hidden vector contains the essential information from the input sequence. It serves as the bridge between the encoder and the decoder. This vector captures the context of the input and is crucial for generating the output sequence.

3. Decoder: The decoder takes the hidden vector and generates the output sequence (e.g., a translation, a summary, or any other sequence). It unfolds step by step, producing one element at a time. The decoder uses the hidden vector to guide its predictions, ensuring coherence and relevance.

The input sequence is transformed into a single-dimensional vector, also known as the hidden vector, by the Encoder. Subsequently, the Decoder converts this hidden vector into the output sequence. To gain a comprehensive understanding of the model's underlying functionality, we will examine the accompanying illustration:
The encoder is that section of architecture that works on the input sequence. The input is given on a token-by-token or word-by-word basis. The encoder will take the words of a sentence one by one and capture the essence/summarize the sentence. Then the encoder will produce an output in the form of a vector(set of numbers) which is called a context vector (it is a summary of

the input sequence). Now in the third step, the decoder will receive the context vector as input. After understanding the input the decoder will print the output sequence again on a token-by-token basis. It is to be noted that LSTMs or GRUs can be used in an encoder. Generally, RNNs are not used due to vanishing gradient problems.

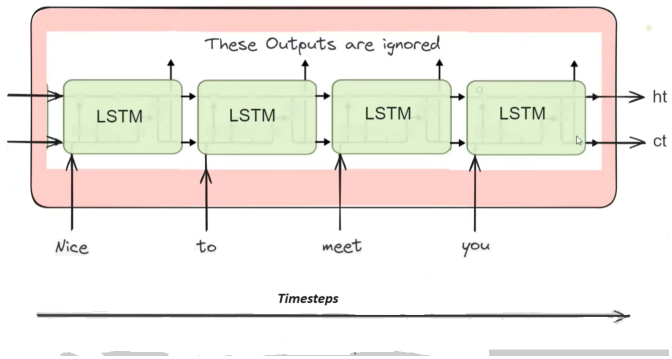


Fig. 3.9: Encoder-Decoder Architecture: CampusX (2024)

Two different LSTMs are used in the encoder and decoder. The work of LSTM in the decoder is to produce output on each timestamp. Also, the initial state of the decoder is the same as the final state of the encoder (which is the context of the input sequence). In **Figure 3.10**, on timestamp 1, a special symbol called <START> is also provided with a context vector to the decoder. This symbol tells the decoder to start producing output from this timestamp.

The output of the first timestamp acts as input to the next timestamp. The process continues until the decoder sees another symbol called <END>, which stops the decoder from using more timestamps.

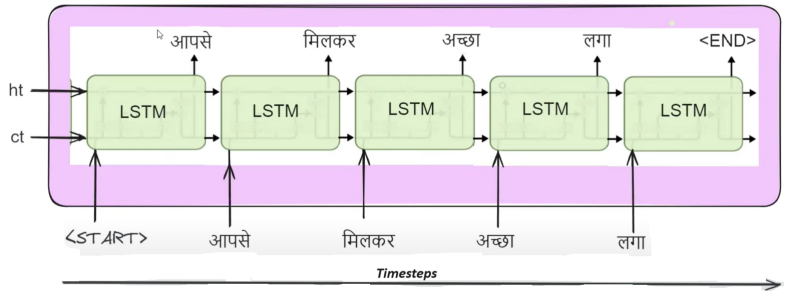


Fig. 3.10: CampusX (2024)

Output Layer: Softmax activation function is used at the output layer which is used to create a probability vector that helps in determining the final output.

The architecture of the encoder and decoder can be significantly improved in two ways.

- The first one is the use of embedding, which involves representing words as dense vectors to capture their semantic relationships.
- The second improvement is the incorporation of deep LSTMs (Long Short-Term Memory) to improve the model's ability to learn long-term dependencies in the input sequence and generate more accurate output.

These improvements have significantly enhanced the performance of sequence-to-sequence models in various natural language processing tasks.

3.5 Attention Mechanism

An Attention Mechanism is a type of neural network architecture that enables the model to selectively focus on crucial parts of the input as it processes the information. This mechanism allows the network to dynamically adjust the importance of different input features based on their relevance to the task at hand, resulting in improved performance and better accuracy. Attention mechanisms are effective in a wide range of machine learning tasks, from image captioning and speech recognition to machine translation and question answering. They are particularly useful in scenarios where the inputs are long and complex, as they enable the model to selectively attend to the most relevant parts of the input while minimizing the impact of noise or irrelevant information. The concept of attention mechanism comes from the paper by Bahdanau et al. (2014). Later, a paper by Luong, Pham, and Manning (2015) proposed several effective variations and improvements to the attention mechanism, which contributed significantly to its development and application in neural machine translation systems. Their work helped to refine and optimize how attention mechanisms are implemented and utilized in practice. Let us explore the necessity for the attention mechanism.

In Encoder-Decoder architecture, the encoder is unnecessarily pressurized to remember the entire input and pass it to the decoder on every timestamp. Also, at the time of translation, the decoder faces the same problem of static representation. The

attention mechanism introduces a dynamic approach to assigning weights to various elements within the input, reflecting their respective significance. This concept enables the model to effectively manage long-range dependencies and ultimately enhance its overall performance.

There are three elements of Attention mechanism architecture:

1. Encoder: The encoder is the initial stage of the attention mechanism. It processes the input sequence (such as a sentence or an image) and generates hidden states. These hidden states capture relevant information from the input data.

2. Attention: The attention mechanism dynamically allocates weights to various elements within the input. It calculates the correlation between the current target's hidden state (from the decoder) and the hidden states of the encoder. By doing so, it determines which parts of the input are most relevant for the current decoding step. The concept of attention enables the model to selectively concentrate on particular segments of the input data when performing a task.

3. Decoder: The decoder takes the weighted combination of the encoder's hidden states (based on attention) and generates the output sequence. It unfolds step by step, producing one element at a time. The decoder uses the context information from the attention mechanism to guide its predictions.

To start with, the process of generating an output sequence

from an input sequence involves an encoder-decoder framework with an attention mechanism. The input sequence is first passed through the encoder, which generates a set of hidden states that capture the relevant information from the input sequence. The subsequent phase involves the attention component, which evaluates the correlation between the current target's hidden state and the encoder's hidden states, thereby facilitating the generation of a series of attention weights. These weights govern how much attention should be given to each of the encoder's hidden states when creating a context vector. The context vector stores the relevant information from the encoder's hidden states and is then passed to the decoder. The decoder uses this context vector, in conjunction with the previously generated output sequence, to produce the next output element. This process is repeated until the entire output sequence is generated.

4. TRANSFORMERS

Transformers are a neural network architecture. Like Artificial Neural Networks(used for tabular data), Convolution Neural Networks(used for image-based data), and Recurrent Neural Networks(used on sequential data like text data), we have transformers that are used to handle sequence-to-sequence tasks. Seq2Seq also known as many to many is a type of RNN in which multiple inputs and multiple outputs correspond to a problem. Machine Translation is one of the examples. Just like other neural network architecture, the Transformer also has an encoder and decoder in its architecture but unlike other architectures that generally use LSTMs, the transformers use a form of attention called self-attention that allows parallel processing of input in the encoder. That is why, the transformer architecture is much scalable, which means it can be trained on a large dataset. In the following sections, an in-depth explanation of the transformer architecture is provided. This will encompass the fundamentals of the architecture, its key features, and how it has revolutionized natural language processing. The aim is to provide a comprehensive understanding of the transformer architecture to enable its effective use in the Chatbots.

4.1 History of Transformers

Transformer architecture was first introduced in the groundbreaking research paper by Vaswani et al. (2017). This paper was from Google Brain and it brought a kind of revolution in the space of deep learning. The most popular chatbot in today's world that is ChatGPT is a Transformer. The Transformers have a major impact on the revolution in NLP because in earlier times when one had to build a NLP application it had to be started from scratch and the model was built using LSTMs in deep learning. Also, a lot of data was required to build the model from scratch and then reach the stage where the model can give good results. But generally, after providing huge data and spending a lot of money the results were not as good as expected.

The invention of transformers changed the entire story by democratizing Artificial Intelligence. As it is a scalable architecture, it was continuously trained on big datasets. As a result, we got some architectures like BERT and GPT. These models were trained on a huge dataset and now are available in the public domain that is anybody can use these pre-trained models. The beauty of these transformers is that they are trained on large datasets but can be fine-tuned which then can be used for specific data.

4.1.1 Why Transformers were created?

There are three research papers, the combination of which constitutes the origin story. The first one is Sutskever et al. (2014) published in 2014-15. In this paper, an architecture was proposed to solve Sequence-to-Sequence tasks like Machine Translation using Encoder and Decoder. LSTMs were used in both the encoder and decoder. But, this architecture works well only when the small input is passed through the encoder. The biggest flaw in this model was that the context vector was not able to retain the long inputs due to which the quality of translated output was not so good. Hence, this model failed.
To overcome this problem, the second paper by Bahdanau et al. (2014) was published. This was the first paper in which the concept of attention was introduced. As a result of the attention mechanism, the architecture of the encoder and decoder has undergone improvements, leading to enhanced results in translated output, particularly for sentences exceeding 30 words. But still, there was a big problem in this architecture called Sequential Training. Since this architecture is based on LSTMs, the input is sent on a token-by-token basis. Due to this, the training is slow so this architecture cannot be applied to huge datasets.
Then finally, the landmark paper by Vaswani et al. (2017) was published in 2017 in which a transformer architecture was introduced that overcame the problem of sequential training completely.

To gain a comprehensive understanding of the Transformer

2000-2014	RNNs/LSTMs
2014	Encoder Decoder/ Attention Mechanism
2017	Attention is All You Need
2018	BERT/ GPT (Transfer Learning in NLP)
2018-2020	Vision Transfer (Alphafold-2)
2021	GenAI
2022	ChatGPT/ Stable Diffusion

Tab. 4.1: The Timeline

model, it is crucial to begin by delving into the concept of self-attention. This type of attention mechanism plays a significant role in the model's architecture, enabling it to capture important information and patterns within the input sequence. Therefore, mastering the concept of self-attention is an essential step in comprehending the inner workings of the Transformer model.

4.2 Self Attention

To make any NLP application, the first requirement is to convert the words into numbers as the computer understands only numbers. Earlier, some techniques like One Hot Encoding, Bag of Words, and TF-IDF were used, and then an advancement of these techniques came called Word Embedding. This technique has the power to capture semantic meaning. Although this is a powerful technique, it has a problem called "Average Meaning".

Let us take an example of the word "apple". Apple can be used in a sentence both as a fruit and as technology. Since word embedding gives static embedding to a word, suppose an apple is embedded as a fruit in the first sentence. In the next sen-

tence apple represents technology, but is considered as a fruit due to static representation. Thus, static embedding is a major drawback of this technique. So, here comes the self-attention mechanism to overcome the above-discussed problem by generating contextual embedding.

Self-attention is a powerful mechanism that can enhance the performance of natural language processing (NLP) applications. It works by taking static embedding as input and generating contextual or dynamic embedding, which can be more effective in capturing the nuances and context of language. As a result, self-attention can improve the accuracy and efficiency of various NLP tasks.

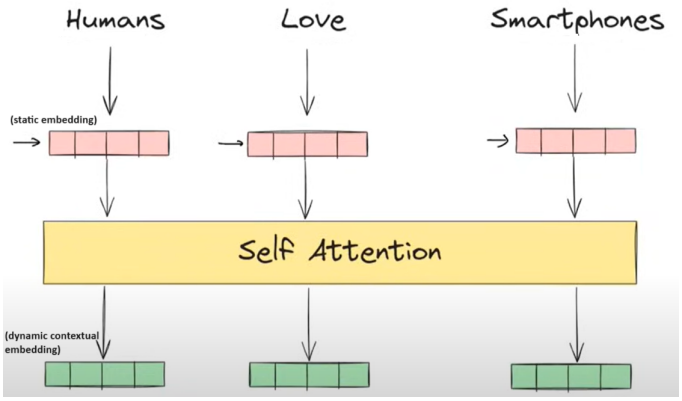


Fig. 4.1: Self-Attention: CampusX (2024)

Let us understand internal calculations in self-attention by the 'First Principle Approach'. Consider the sentence "Money Bank Grows". Let e_{money}, e_{bank}, e_{grows} denote the static word embeddings of money, bank, and grows respectively. Then, the

contextual embedding of 'bank' is calculated as:

$$e_{bank}^{(new)} = [e_{bank}.e_{money}^{T}]e_{money} + [e_{bank}.e_{bank}^{T}]e_{bank} + [e_{bank}.e_{grows}^{T}]e_{grows}$$

Note that the dot product tells the similarity between the two things.
Similarly, we can find new contextual embedding for all other words. The above equation can be visually seen in the **Figure 4.2** :

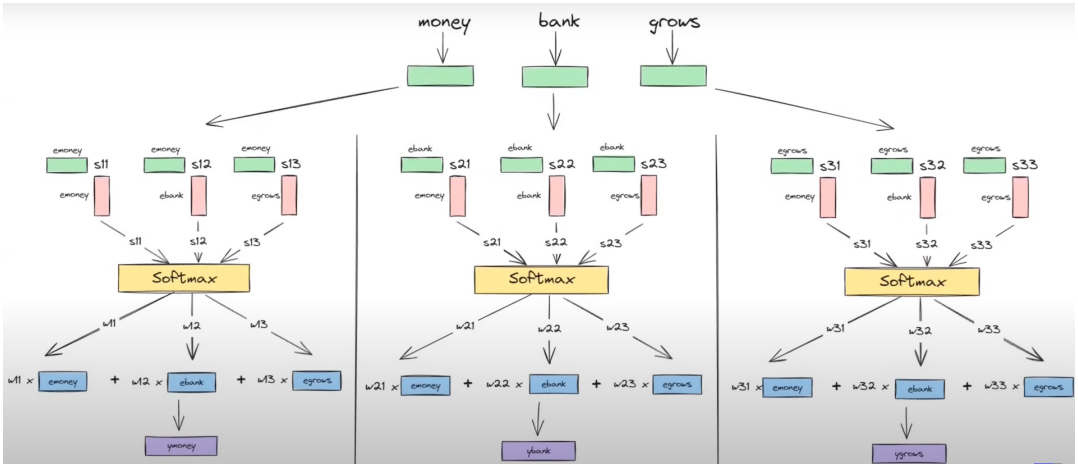


Fig. 4.2: First Principle Approach: CampusX (2024)

4.2.1 Points to Consider

- A good thing about the First Principle Approach is that this is a parallel operation, that is, the new contextual embeddings for all the words are generated parallelly. This means that multiple operations can be performed simultaneously without any interference or dependency on each other, re-

sulting in increased efficiency and faster execution times.

- The current model suffers from a notable drawback as it does not involve any parameters, which essentially means it cannot learn from data. Therefore, the model's performance is limited as it cannot adapt to changing situations or new data.

  In addition, although the model can generate contextual embedding, it is not tailored to specific tasks, which ultimately affects its effectiveness. Task-specific contextual embedding is more appropriate as it considers the unique requirements and complexities of a given task. By doing so, the model can generate more relevant and accurate results.

  To provide an example, consider the phrases "piece of cake" and "break a leg." While the current model may interpret these phrases as literal meanings, a task-specific contextual embedding model would understand their figurative meanings in different contexts.

Now, our goal is to improve this First Principle Approach by introducing the learnable parameters such as weights and biases so that our self-attention model can generate better task-specific contextual embedding after each training sample. Now we will split all the embedding vectors into three vectors: Query, Key, and Value (see **Figure 4.3**).

1. Query: The query represents the information we want to retrieve or focus on. It is typically associated with a specific

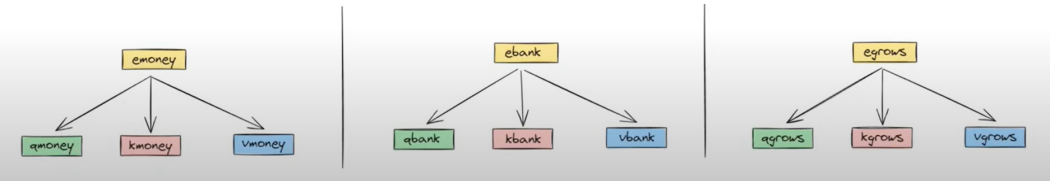


Fig. 4.3: Query, Key, Value: CampusX (2024)

token or position in the input sequence. Mathematically, we calculate the query vector by multiplying the input embedding matrix X with a learned weight matrix W_q:

$$Q = X.W_q$$

Here, Q has shape (N, h), where N is the number of tokens in the input sequence, and h is the number of attention heads.

2. Key: The key provides context or additional information related to each token. Similar to the query, we compute the key vector by multiplying the input embedding matrix X with another learned weight matrix W_k:

$$K = X.W_k$$

3. Value: The value represents the actual content associated with each token. Once again, we obtain the value vector by multiplying the input embedding matrix X with a weight matrix W_v:

$$V = X.W_v$$

Self-attention is a widely used technique in natural language processing (NLP) models that has facilitated significant advance-

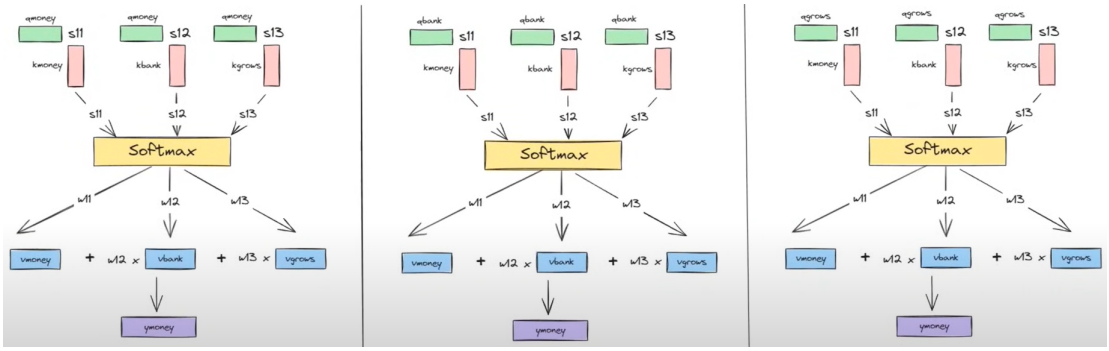


Fig. 4.4: Self attention mechanism: CampusX (2024)

ments in the field. However, it is not free from limitations. One such limitation is the inability of self-attention to capture multiple perspectives in a sentence or document. To illustrate this, let us consider the example of the ambiguous sentence 'The man saw the astronomer with a telescope'. This sentence can be interpreted in two ways: one being 'The man saw the astronomer with the help of a telescope', and the other being 'The man saw the astronomer who had a telescope'. Unfortunately, self-attention can only capture one of these meanings, which restricts its applicability in scenarios where multiple perspectives need to be taken into account, such as in text summarization.

To address this limitation, a technique known as Multi-Head Attention has been introduced. This technique enables NLP models to consider multiple perspectives of a sentence or document by allowing them to attend to different parts of the input simultaneously. The subsequent section elaborates on this technique in detail.

4.3 Transformer model architecture

The Transformer neural network utilizes an Encoder-Decoder architecture. However, the Transformer differs in that the input sequence can be processed in parallel. In the transformer encoder, the input does not have a concept of time steps. Instead, all words in the sentence are processed concurrently, and their word embeddings are determined simultaneously.

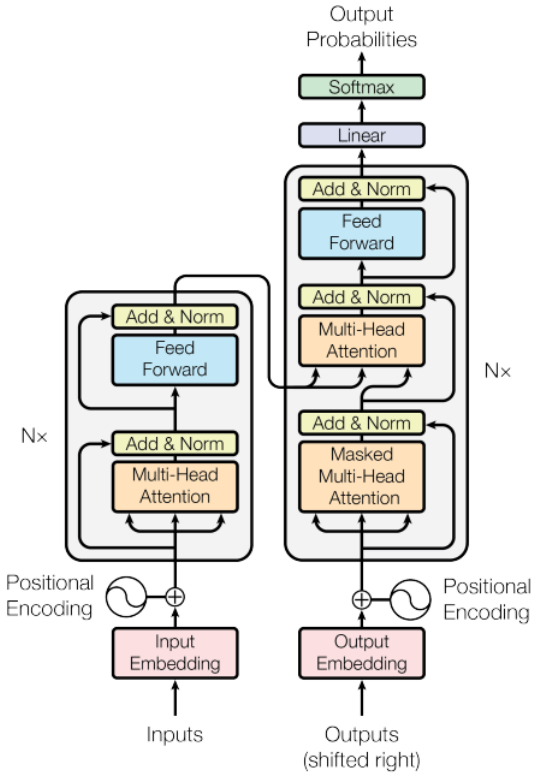


Fig. 4.5: Transformer-model architecture:Vaswani et al. (2017)

- Input Embeddings: In the realm of computing, it is widely known that computers do not have the natural capacity to understand language as humans do. Instead, they possess

a more nuanced understanding of numbers, vectors, and matrices. One way to bridge this gap is to create a map that associates every word with a specific point in space. This mapping helps to spatially represent similar words in a manner that places them in close proximity to one another. This space, referred to as an Embedding Space, can be pre-trained to optimize efficiency, or alternatively, an existing pre-trained space may be utilized. The embedding space facilitates the mapping of a given word to a vector. However, a single word can take on different meanings depending on its context within a sentence. This is where the positional encoder becomes relevant.

- Positional Encoding: The positional encoder is a vector that encapsulates information regarding the relative distances between words and sentences within a given context. In the original paper,Vaswani et al. (2017) a sine and cosine function are utilized to generate this vector, although any reasonable function could be used. After the sentence undergoes input embedding and positional encoding, it yields word vectors that encompass positional information. These vectors are then passed through the encoder block where they are subjected to the Multi-Head attention layer and Feed-Forward layer.

- Multi-Head Attention: Multi-head attention is a robust and effective attention mechanism that significantly improves the processing of a sequence of inputs. This mechanism oper-

ates by dividing the keys, queries, and values into multiple attention heads, each of which is then sent through a distinct set of attention weights. This approach enables the model to concurrently focus on different aspects of the input sequence. The outputs of the attention heads are then concatenated and linearly transformed to achieve the expected dimension, resulting in a refined representation of the input sequence. Multi-head attention is an important technique for natural language processing and machine learning tasks that require the modeling of complex relationships between inputs. It has been successfully applied to a range of applications, including language translation, image captioning, and speech recognition. Its efficiency and effectiveness make it a valuable tool for researchers and practitioners alike, and it is likely to be widely used in future developments in the field of Machine Learning.

- Feed Forward Layer: The Feed Forward Layer is a fundamental component of the neural network architecture, wherein a simple feed-forward neural network is employed to process each attention vector. The primary purpose of this layer is to transform the attention vectors into a more comprehensible form for the subsequent encoder or decoder block. In practical applications, feed-forward networks are often utilized to facilitate this transformation process.
  Now, Let us delve deeper into the decoder part of the transformer model.

- It is crucial to understand that during the training phase, we must feed the output sentence to the decoder in a form that computers can understand, i.e., numbers, vectors, and matrices. To process the output sentence, we use the input embedding to obtain the vector form of the word and then add a positional vector to provide the context of the word in the sentence. Finally, we pass this vector into a decoder block that consists of three main components, two of which are similar to the encoder block.

- The self-attention block is responsible for generating attention vectors for each word within a sentence, capturing the interdependencies among all words in the given sentence. These attention vectors, along with the vectors from the encoder, are then directed to another attention block, known as the encoder-decoder attention block. This component assesses the level of correlation between each word vector in the input and output sentences, effectively establishing a word-to-word mapping from the main input sentence to the output sentence. The output of this process yields attention vectors for every word in both the input and output sentences, depicting their relationships across both languages. The attention vectors are subsequently subjected to a feed-forward unit to facilitate their assimilation by the subsequent decoder block or linear layer.

- Using the linear layer, the dimensions are expanded into the number of words in the input before the softmax layer

transforms it into a probability distribution. The final word is the one corresponding to the highest probability making it a human-interpretable sentence.

In summary, this decoder predicts the next word and it is executed over multiple time steps until the end of the sentence token is generated.

4.4 Unvieling the Power and Pitfalls Of Transformers

- Scalability: The Transformer architecture is not LSTM-based and instead relies on attention mechanisms that enable parallel training. As a result, this architecture offers faster training times and is highly scalable to larger datasets.
- Transfer Learning: Transformers models are a type of neural network architecture that can be trained on massive amounts of data using unsupervised learning techniques. This pre-training process enables the model to learn general patterns and features in the data, which can be applied to a wide range of downstream tasks. Once the model is pre-trained, it can be fine-tuned with relatively little effort to adapt to new tasks or domains. This approach to natural language processing has revolutionized the field and led to significant improvements in language understanding and generation tasks.
- Multimodel Input/ Output: The best thing about trans-

formers is their flexible architecture as they are not only used for textual data but also can be used with other modalities like image, speech, etc. Also, we have different types of transformer architectures such as encoder-only architecture like BERT and decoder-only architecture like GPT.
Every advantage is accompanied by its corresponding set of disadvantages. The following are the associated drawbacks:

- It requires high computational resources because in transformers, the information is processed parallelly in an encoder but for training, we need highly expensive GPUs.
- In some specific domains, it requires a lot of data to give good performance. But the good part is that the applications made till now like Large Language Models(LLMs) are easily trained on text data without labels(that is, unsupervised learning).
- With so many parameters to consider, overfitting can be a serious problem when there is not enough data variety. However, some steps can be taken to mitigate this risk and ensure that the resulting models are both accurate and reliable.
- Training these huge models requires powerful hardware, which results in increased energy consumption.
- The Transformer model operates like a mysterious black box, making it difficult to understand its inner workings. Therefore, interpreting the results can be challenging. Although

the model yields good outcomes, it struggles to function efficiently in critical sectors such as banking and healthcare, where it is often crucial to understand how the results are generated.

- When the data used to train any model is biased, it is likely to create problems in the model results and raise ethical concerns.

4.5 Utilizations of Transformers

- ChatGPT: ChatGPT is an AI-powered chatbot that utilizes the advanced Generative Pre-Training technology of GPT3 to provide users with a more personalized and natural conversational experience. The advanced technology enables the system to comprehend and address user inquiries in a manner that closely resembles human interaction, thus establishing it as a dependable and valuable aide for a wide range of tasks.

- DALL-E2: An AI system has been developed with the capability to generate lifelike images and art based on natural language descriptions. Using cutting-edge technology and advanced algorithms, it can bring to life even the most complex and intricate visual concepts with stunning accuracy and attention to detail.

- Alpha Fold: The project Alpha Fold, created by Google Deep Mind, uses a transformer architecture to aid in the

exploration of structural biology. Its main focus is to accurately predict the three-dimensional structure of proteins, which is crucial for understanding their function and developing new drugs.

- Open AI Codex: This is an AI system that translates natural language into code. It is designed to make the process of programming more efficient and accessible by allowing users to simply express their ideas and intentions in everyday language, rather than having to memorize complex programming syntax and structures.

4.6 The Future Horizon

- Improvements in Efficiency: Many techniques like Proning, Quantization, and Knowledge Distillation are in use to improve the model's efficiency in terms of training and model size.

- Multimodal Capabilities: The written text has traditionally served as the primary mode of communication. However, there is a growing trend in the development of transformers designed to process diverse data types including images, speech, sensory data, biometric feedback, and time series-based data.

- Responsible Development of Transformers: By employing thorough and objective evaluation methods, it is possible to

eliminate any form of bias and address other ethical concerns that may arise in various situations. This requires a commitment to transparency, fairness, and the use of reliable data to inform decision-making processes. By doing so, we can ensure that our actions are just equitable, and free from any form of discrimination, thereby fostering a more inclusive and respectful society.

- Domain Specific: In the future, we can also get domain-specific Transformers like Doctor GPT, Teacher GPT, and Legal GPT, which will be fully trained in their specific domains.

- Integrated to AI Techniques: The transformer architecture can be improved by integrating it with other AI techniques. One potential enhancement is to combine GANs with transformers to create image generators similar to DALL-E. This approach could lead to the development of more advanced and diverse image-generation capabilities. Additionally, reinforcement learning can be used with transformers to create game-playing agents that can learn how to play games by themselves. This technique has been used to train agents that excel at complex games such as Go and Chess. Also, by integrating CNN with transformers, we can create an image caption generator that can analyze an image and generate a natural language description of it. This technology could be useful in a variety of applications, such as helping visually impaired people understand visual content.

- Multi-lingual: At present, transformers have exclusively undergone training in the English language. However, it is conceivable that future developments will involve the creation of transformers from the ground up in languages such as Hindi.

- Interpretability: Transformers are often likened to a black box, as their inner workings are opaque despite the high quality of their output. There is a need for future research to delve into the inner workings of transformers to understand the "how" behind their results. This understanding could pave the way for the development of white box transformers, which could be applied in critical fields such as banking and medicine. In these domains, the rationale behind decisions, or the "why," carries significant weight.

5. METHOD

5.1 Problem Definition

In this thesis, firstly we developed a GRU sequence-to-sequence model for chatbot utilization. Next, a chatbot is implemented to assist students by answering all admission-related queries at Thapar Institute of Engineering and Technology. In order to accomplish its objective, the chatbot must possess the capability to process and comprehend natural language input. Upon receiving this input, the bot must undertake actions such as soliciting additional information to clarify a user request, accessing a knowledge base to retrieve information, interpreting the outcomes of queries, and presenting the results in a user-friendly manner. Moreover, if the user changes their input, the bot should be equipped to discern this change and adapt to pursue the newly specified objective. This chatbot is generated by implementing the Transformer model. Lastly, we created this chatbot using Dialogflow which is then integrated on telegram.

5.2 Chatbot - Gated Recurrent Unit Model

There exists a model that utilizes a sequence-to-sequence architecture proposed by Sutskever et al. (2014). An attempt to code our own Sequence-to-Sequence model was not very fruitful so instead the thesis uses code from an online tutorial by Inkawhich (2018), which was trained on a substantial English corpus. It responded correctly to a variety of subjective questions. Below is the list of sample questions along with the responses provided by the GRU model.

```
User: hello .
Bot: you are a good person .
User: what is your name ?
Bot: sejal .
User: what time is it ?
Bot: i don t know .
User: what do you do ?
Bot: i m a comedian .
User: do you like green ?
Bot: no .
User: what is your favorite candy ?
Bot: it s a big one .
User: do you like ice cream ?
Bot: no .
User: good bye .
Bot: good bye .
```

5.3 Chatbot - Transformer Model

A transformer model is being utilized to develop a chatbot that addresses admission-related inquiries at the Thapar Institute of Engineering and Technology. The data is sourced from the official website of TIET and is fed to the model in a passage format. Here are excerpts from our code to provide clarity on the process.

```
from transformers import pipeline, AutoTokenizer,
AutoModelForQuestionAnswering
```

Source code 5.1: Importing necessary libraries

The "pipeline" module from transformers allows us to use pre-trained models for tasks like question-answering. "AutoTokenizer" and "AutoModelForQuestionAnswering" are helpful classes from the Hugging Face Transformers library. They have the capability to automatically detect and load the tokenizer that matches a specific pre-trained model. This simplifies the task of tokenizing text for a wide range of transformer models by offering a consistent and unified interface.

```
model_name = "deepset/roberta-base-squad2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForQuestionAnswering.from_pretrained(
model_name)
```

Source code 5.2: Loading the Pre-trained Model and Tokenizer

The model_name specifies the model to be used. The "deepset/roberta-base-squad2" is a pre-trained language model that is built on RoBERTa architecture and has been specifically fine-tuned on the SQuAD2.0 dataset for tasks related to extracting answers

from questions. It is adept at comprehending natural language queries and adeptly identifying the most precise answers from a provided text, even when faced with unanswerable questions. “AutoTokenizer.from_pretrained” loads the tokenizer corresponding to the specified model. “AutoModelForQuestionAnswering.from_pretrained” is utilized to instantiate the pre-trained model for the specific task of question-answering.

```
    def admission_chatbot(question, context):
    inputs = tokenizer(question, context, return_tensors="pt")
    outputs = model(**inputs)
    answer_start_scores = outputs.start_logits
    answer_end_scores = outputs.end_logits
    answer_start = torch.argmax(answer_start_scores)
    answer_end = torch.argmax(answer_end_scores) + 1
    answer = tokenizer.convert_tokens_to_string(tokenizer.convert_ids_to_tokens(inputs["input_ids"][0][answer_start:answer_end]))
    return answer
```

Source code 5.3: Defining the Question-Answering Function

We define a function that takes a question and context (admission-related information in our case) as input. “tokenizer()” tokenizes the question and context into a format suitable for the model (‘pt’ denotes PyTorch tensors). “model(**inputs)” Passes the tokenized inputs to the model. “answer_start” and “answer_end” computes the start and end positions of the answer span based on the model’s predictions. “answer” converts the predicted token IDs back into a readable answer string.

```
while True:
    user_question = input("Your question: ")
    if user_question.lower() == "exit":
```

```
4         print("Chatbot: Goodbye!")
5         break
6     answer = admission_chatbot(user_question, admission_info)
7     print("Chatbot:", answer)
```

Source code 5.4: Interaction Loop

Runs a loop where the chatbot waits for user input. If the user inputs "exit", the loop terminates. Otherwise, the chatbot uses the admission_chatbot function to find and print the answer to the user's question.

The following is an exemplary dialogue exchange between a user and the transformer chatbot:

```
Your question: What are the courses offered at TIET?
Chatbot:  undergraduate and postgraduate
Your question: Is there any library in Thapar?
Chatbot:  Nava Nalanda Central Library
Your question: What is the admission criteria?
Chatbot: <s>
Your question: What documents should I bring?
Chatbot: <s>
Your question: Is there any canteen in Thapar?
Chatbot: Our university has various canteens with a
variety of food available
Your question: What is the official website?
Chatbot: www.thapar.edu
Your question: Is Thapar a good university?
Chatbot:  Excellent Infrastructure
```

5.4 Designing a chatbot utilizing Google Dialogflow.

To create a chatbot that truly delivers, you must focus on three key phases: Design, development, and deployment. A chatbot that excels at a single function is more impactful than one that attempts to perform multiple functions inadequately. So, make sure to keep your focus narrow and your execution strong. We propose to develop a domain-specific chatbot that can cater to all the admission-related queries pertaining to the Thapar Institute of Engineering and Technology.

Dialogflow is a highly advanced platform that was originally known as Api.ai, which has now been acquired by Google. It is a revolutionary platform that is specifically designed for building conversational agents, and it streamlines the entire process, making it more efficient and easy for developers to build powerful chatbots. One of the most significant advantages of using Dialogflow is that it combines various components, including transformer-based models like DialoGPT, to create an efficient and effective conversational experience for users. DialoGPT is a dialogue-response-generation model created by Microsoft Research that is pre-trained on a large scale. It was established in November 2019 as part of the ongoing research in conversational AI. Formulated as an autoregressive (AR) language model, it employs a multi-layer transformer architecture, similar to GPT-2. The training data consists of 147 million multi-turn dialogues from Reddit, making it more conversationally interactive. This

means that developers can leverage the power of machine learning and natural language processing to create chatbots that are highly responsive, intelligent, and capable of handling complex queries and requests with ease.

To build a chatbot, the first and foremost step in Dialogflow is to create an agent. The agent should be trained to handle anticipated conversation scenarios, similar to a human call center agent, without being overly explicit. Once the agent is created, the intents must be defined. It is important to remember that intents must be mutually exclusive to avoid any ambiguity. This ensures that there is no overlap between two different intents and guarantees that user input is always mapped to an intent. While it is impossible to make intents collectively exhaustive, Dialogflow has an automatic fallback intent that maps to user input if it does not match any other intent.

The subsequent step involves the inclusion of training phrases to each intent. Dialogflow, an AI-powered technology, learns from these training phrases and associates comparable inputs to the same intent. In the context of a chatbot dealing with admission queries, training phrases such as "What is the placement rate," "How many students received placement last year," and "Which companies participate in placement drives every year" will all align with the intent of "Placement". To ensure the training phrases' effectiveness, they should vary in grammatical structure and tone, including active/passive voice, statement/question, generic/specific, and formal/informal expressions.

The chatbot's ability to maintain context enables it to keep track of the user's position within the conversation, enabling it to transition seamlessly across intents without becoming confused. Training phrases and responses play a crucial role in defining intents.

Once you have created all the intents along with their respective training phrases, the next step involves creating Entities. Entities play a crucial role in extracting relevant and useful information related to intent, thereby providing more context to the conversation. With Dialogflow, you have the flexibility to either let the values of entities be expanded automatically throughout the conversation or keep them fixed. Additionally, Dialogflow offers a unique feature called Small Talk, which can significantly enhance the end-user experience by allowing your chatbot to handle common queries and engage in casual conversation, thereby making it more conversational and intuitive for the user. Until this juncture, the process of creating intents, training phrases, context, entities, and slot filling has been facilitated through the use of the drag-and-click functionality within Google Dialogflow. However, the storage of this data necessitates manual coding via Dialogflow's inline editor. The inline editor is equipped with prewritten code in node.js and a JSON file.

5.4.1 Data

The data obtained from authentic conversations often incorporates a range of inaccuracies, such as typographical errors, erroneous information, and misinterpretations. These complexities introduce considerable challenges to the training process. Furthermore, broader or more open subject areas pose additional hurdles due to the increased diversity of dialogues and the expansion of potential responses. To address these obstacles, the chatbot was trained and evaluated within a tightly constrained domain using artificial data. Utilizing synthetic data offers greater control over the dialogue format, enabling a more direct assessment of the system's strengths and limitations. Furthermore, for a chatbot to function effectively, its training data must exhibit a conversational flow, with each interaction comprising a statement or question followed by a response. To further evaluate the system in a more authentic context, a dataset with tag, pattern, and corresponding response is created for the question-answering chatbot using dialogflow. Due to the enormity of the dataset we generated, here's a sample of the original dataset.

```
{
 "tag": "greeting",
 "patterns": "Hi",
 "responses": "Hello!"
},
{
 "tag": "creator",
 "patterns": "What is the name of your developers",
 "responses": "Sejal"
```

```
},
{
 "tag": "hours",
 "patterns": "timing of college",
 "responses": "College is open 8 am-5:10 pm Monday-Friday!"
},
{
 "tag": "number",
 "patterns": "more info",
 "responses": "You can mail at admissions@thapar.edu"
},
{
 "tag": "location",
 "patterns": "where is the college located",
 "responses": "P.O. Box 32, Bhadson Road, Patiala, Punjab,
  Pin -147004, India"
},
{
 "tag": "document",
 "patterns": "document to bring",
 "responses": "To know more about document required visit
  www.thapar.edu"
},
{
 "tag": "floors",
 "patterns": "size of campus",
 "responses": "Thapar University boasts a 250-acre campus in
   Patiala"
},
{
 "tag": "infrastructure",
 "patterns": "How is college infrastructure",
 "responses": "Our University has Excellent Infrastructure."
},
{
 "tag": "menu",
 "patterns": "food menu",
 "responses": "We serve Franky, Alu-puri, Thaali, and many
  more on the  menu"
```

```
},
{
 "tag": "Hod",
 "patterns": "Who is the  HOD of the  maths department",
 "responses": "Dr. Mahesh Kumar Sharma."
},
{
 "tag": "ragging",
 "patterns": "ragging",
 "responses": "We are proud to tell you that our college
  provides a ragging-free environment, and we have strict
  rules against ragging"
},

{
 "tag": "goodbye",
 "patterns": "Cya",
 "responses": "Sad to see you go"
}
```

Source code 5.5: Importing necessary libraries

The construction of our chatbot has been completed and is operational on Telegram. It can be accessed using the provided link: https://web.telegram.org/a/#6953281021

Here is an example dialogue exchange between a user and the conversational agent:

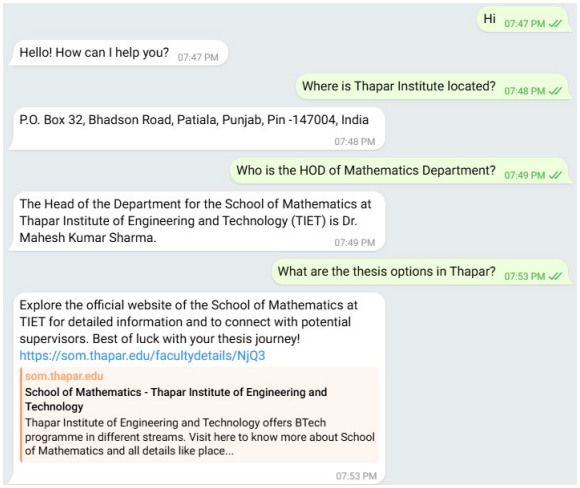


Fig. 5.1: Chatbot- Dialogflow

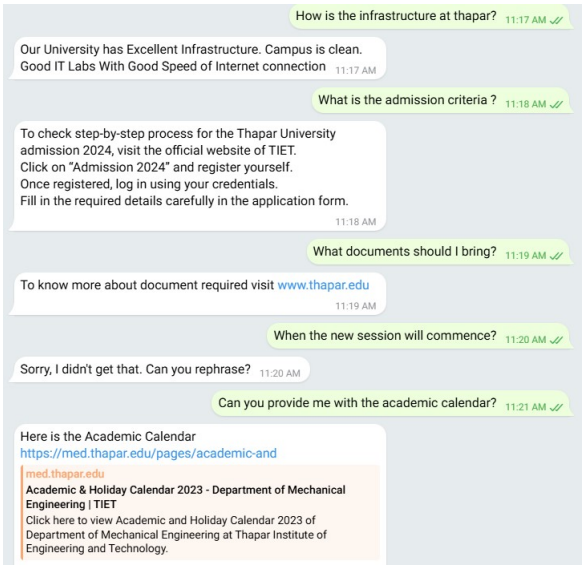


Fig. 5.2: Chatbot- Dialogflow

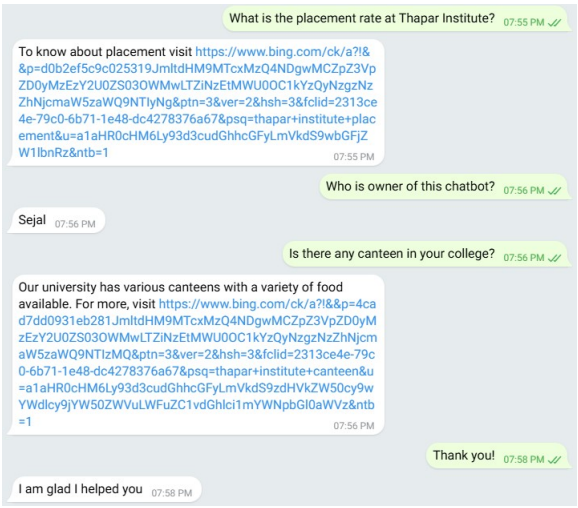


Fig. 5.3: Chatbot- Dialogflow

6. CONCLUSION AND FUTURE WORK

Throughout the thesis, we explore that the transition from attention mechanisms to Transformers has significantly advanced chatbot development, enabling more natural and contextually aware interactions.

6.1 Summary of Architectural Insights

We investigated the creation of chatbots using two distinct neural network architectures: GRU and transformer models. Subsequently, we proceeded to build a chatbot by integrating it with dialogflow. The GRU model demonstrated efficient sequential processing, making it suitable for tasks requiring context preservation over short to medium-length conversations. On the other hand, transformer models, with their attention mechanisms, excelled in capturing global dependencies and context understanding, which proved beneficial in handling longer and more complex dialogues. Integrating our neural network models with Dialogflow provided several advantages, including streamlined deployment, scalability, and enhanced natural language processing

capabilities through pre-trained language models. Dialogflow's robust platform facilitated rapid prototyping and deployment iterations, allowing us to focus more on refining dialogue design and user experience rather than underlying model implementation details. The conclusions drawn in this thesis align with previous research that suggests:

> a simple chatbot can be done without needing a lot of coding knowledge.A chatbot that excels at one or two tasks is superior to a chatbot that performs numerous tasks inadequately.

The key takeaways encompass the criticality of dataset quality in the effective training of conversational agents, the influence of model architecture on dialogue coherency and scalability, and the importance of iterative user testing in refining conversational flows and enhancing user satisfaction.

6.2 Future Directions

The scope of this thesis was confined to a specific domain. To gauge the scalability of the results, it would be necessary to conduct tests on other, potentially broader, domains. Additionally, exploring alternative learning models and evaluating their performance in comparison to the models addressed in this thesis would be of great interest. While Transformers have demonstrated remarkable performance, ongoing research is needed to address scalability, interpretability, and ethical considerations to

realize the full potential of chatbot technology. Variety of neural network-based architectures exist, each characterized by unique strengths and weaknesses. Exploring potential adaptations to the sequence-to-sequence model presents an intriguing opportunity to assess whether such modifications can yield improved outcomes. Additionally, investigating hybrid models combining GRU for sequential processing and transformer for global context understanding could potentially leverage the strengths of both architectures.

Further advancements in integrating chatbots with Dialogflow could focus on optimizing dialogue management systems for scalability across multiple channels and languages. Enhancements in handling user intents and context switching dynamically would contribute to a more robust and versatile conversational AI framework. Exploring emerging technologies such as generative pre-trained transformers (GPT) and transfer learning approaches could revolutionize chatbot development by leveraging large-scale language models for improved dialogue generation and understanding. Future work should prioritize enhancing user experience through advanced dialogue management techniques, incorporating sentiment analysis for personalized responses, and integrating multimodal interactions to support voice and text-based interactions seamlessly. In the above section, we have outlined several concepts that stem from the discoveries outlined in this thesis. However, it is important to note that there are numerous additional avenues for prospective

research within this field.

BIBLIOGRAPHY

Bahdanau, D., K. Cho, and Y. Bengio (2014): "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*.

CampusX (2024): "100 days of deep learning: YouTube Channel," .

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014): "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018): "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*.

Elman, J. L. (1990): "Finding structure in time," *Cognitive science*, 14, 179–211.

Hochreiter, S. and J. Schmidhuber (1997): "Long short-term memory," *Neural computation*, 9, 1735–1780.

Howard, J. and S. Ruder (2018): "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*.

Inkawhich, M. (2018): "pytorch-chatbot," GitHub repository.

Luong, M.-T., H. Pham, and C. D. Manning (2015): "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*.

Radford, A., K. Narasimhan, T. Salimans, I. Sutskever, et al. (2018): "Improving language understanding by generative pre-training," .

Sutskever, I., O. Vinyals, and Q. V. Le (2014): "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, 27.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017): "Attention is all you need," *Advances in neural information processing systems*, 30.