



**Academy of
Engineering**
(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

A Minor Project Report on Text Generator

Submitted by,

Sachin Chavan	(Exam Seat No. 202101040217)
Siddhesh Paithankar	(Exam Seat No. 202101040110)
Vaishnavi Bhonde	(Exam Seat No. 202101040099)
Viraj Chudasama	(Exam Seat No. 202101040241)

Guided by,

Mrs. Mayura A. Kulkarni

**A Report submitted to MIT Academy of Engineering, Alandi(D), Pune,
An Autonomous Institute Affiliated to Savitribai Phule Pune University
in partial fulfillment of the requirements of**

**SY BTECH. in
Computer Engineering**

School of Computer Engineering

MIT Academy of Engineering

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

Alandi (D), Pune – 412105

(2022–2023)

CERTIFICATE

It is hereby certified that the work which is being presented in the SY BTECH Minor Project Report entitled “**Text Generator**”, in partial fulfillment of the requirements for the SY BTECH Minor project. in Computer Engineering and submitted to the **School of Computer Engineering of MIT Academy of Engineering, Alandi(D), Pune, Affiliated to Savitribai Phule Pune University (SPPU), Pune**, is an authentic record of work carried out during Academic Year **2022–2023**, under the supervision of **Mrs. Mayura A. Kulkarni, School of Computer Engineering**

Sachin Chavan (Exam Seat No. 202101040217)

Siddhesh Paithankar (Exam Seat No. 202101040110)

Vaishnavi Bhonde (Exam Seat No. 202101040099)

Viraj Chudasama (Exam Seat No. 202101040241)

**Mrs. Mayura A.
Kulkarni
Project Advisor**

**Mrs. Padma Nimbhore
Project Coordinator**

**Dr. Rajeshwari Goudar
Dean SCE**

External Examiner

DECLARATION

We the undersigned solemnly declare that the Minor project report is based on our own work carried out during the course of our study under the supervision of **Mrs. Mayura A. Kulkarni**.

We assert the statements made and conclusions drawn are an outcome of our project work. We further certify that

1. The work contained in the report is original and has been done by us under the general supervision of our supervisor.
2. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this Institute/University or any other Institute/University of India or abroad.
3. We have followed the guidelines provided by the Institute in writing the report.
4. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Sachin Chavan (Exam Seat No. 202101040217)

Siddhesh Paithankar (Exam Seat No. 202101040110)

Vaishnavi Bhonde (Exam Seat No. 202101040099)

Viraj Chudasama (Exam Seat No. 202101040241)

Abstract

We all visit doctors in our day-to-day life and know the importance of proper diagnostics and proper prescription of medications. But the fact is, we stay in India and the population here is enormous, and the doctors are very limited in our country. Doctors follow manuals which has information about how to diagnose a patient accordingly, which they have to follow but as we know doctors have a time crunch and cannot spare much time on a single patient so therefor by using the old method of writing it on paper this is not going to work, we need a solution for that if we add a word or little bit about the disease the program can generate the history regarding the disease and it can learn the format that the doctor has to follow and further it can be helpful for the doctor for upcoming patients. Our system aims to deliver that time efficiency and effective prescription needed by the Doctors as well as patient with the help of NLP and NLG.

Acknowledgment

It is a great privilege for us to express our profound gratitude to our respected teacher Mrs. Mayura A. Kulkarni, School of Computer engineering Technology, his constant guidance, valuable suggestion, supervision and insertion throughout the course work without which it would have been difficult to complete the work within scheduled time. We are also indebted to the Head of department, school of Computer engineering Technology, MIT academy of engineering for permitting us to pursue the project. We would like to take this opportunity to thank all the respected teachers of this department for being a perennial source of inspiration and showing the right path at the time of necessity

Sachin Chavhan 202101040217

Siddhesh Paithankar 202101040110

Vaishnavi Bhonde 202101040099

Viraj Chudasama 202101040241

Contents

Abstract	iv
Acknowledgement	v
1 Introduction	1
1.1 Background	1
1.2 Project Idea	1
1.3 Motivation	2
1.4 Project Challenges	3
1.5 Proposed Solution	3
2 Literature Review	4
2.1 Related work And State of the Art (Latest work)	4
2.2 Limitation of State of the Art techniques	7
2.3 Discussion and future direction	8
2.4 Conclusion	9
3 Problem Definition and Scope	10
3.1 Problem statement	10

3.2	Goals and Objectives	10
3.3	Scope and Major Constraints	11
3.4	Expected Outcomes	11
4	System Requirement Specification	12
4.1	Overall Description	12
4.1.1	Product Perspective	12
4.2	Specific Requirements	12
4.2.1	User Requirements	12
4.2.2	External Interface Requirements	12
4.2.3	Functional Requirements	12
4.2.4	Performance Requirement	13
5	Methodology	15
5.1	System Architecture	15
5.2	Mathematical Modeling	16
5.3	Algorithms	19
6	Implementation	22
6.1	Description	22
6.2	Output	23
7	Result Analysis/Performance Evaluation	25
7.1	Result Analysis of Objective 1	25
7.2	Result Analysis of Objective 2	27
7.3	Result Analysis of Objective 3	27

7.4	Result Analysis of Objective 4	27
8	Conclusion	28
8.1	Conclusion	28
8.2	Future Scope	29
	Appendices	31
A	TensorFlow	32
	References	33

List of Figures

2.1	Survey of Knowledge-Enhanced Text generation	6
2.2	Knowledge enhanced text generation	6
4.1	Block Diagram	14
5.1	The Transformer - Model Architecture	15
5.2	Factorization of generation Probability	16
5.3	Bart Score	17
5.4	Bert	17
5.5	LSTM	18
5.6	GRU	18
5.7	LSTM - GRU	18
5.8	SGD	19
5.9	RMSProp Algorithm	19
5.10	Gradient used for Optimisation	20
5.11	Operation in Adagrad	20
5.12	Adaptive Motion	20
5.13	Vectors Bias-correction	21

6.1	Output 1	23
6.2	Output 2	23
6.3	Output 3	24

List of Tables

4.1	Functional Requirements of Topic	13
-----	--	----

Chapter 1

Introduction

1.1 Background

We all visit doctors in our day-to-day life and know the importance of proper diagnostics and proper prescription of medications. But the fact is, we stay in India and the population here is enormous (139.34 crores), and the doctors are very limited in our country. According to Indian Medical Registry, there were around 13.01 lakh registered doctors in 2021. Each physician has between 1200 and 1900 patients on the current panel, according to research. The panel of a primary care physician is averaged at 2500. That is a very huge number! We cannot control the population in our country, but we can bring about a change in the prescription methods which currently comprise of pen and paper approach, with only a few places supporting digital prescriptions – each of which has their software developed. We are trying to develop a platform that can address this issue and promote faster and better prescription practices which can in turn benefit the patient as well as the doctor.

1.2 Project Idea

We want to help the doctors by giving them a tool that is more efficient in diagnosing the patients, as we know the diagnosis of a patient has many steps to be followed in a sequence order first to ask the patient about the disease, then to ask them

about their past health issues then the symptoms and for how many days they had the thing. Doctors follow manuals which has information about how to diagnose a patient accordingly, which they have to follow but as we know doctors have a time crunch and cannot spare much time on a single patient so therefor by using the old method of writing it on paper this is not going to work, we need a solution for that if we add a word or little bit about the disease the program can generate the history regarding the disease and it can learn the format that the doctor has to follow and further it can be helpful for the doctor for upcoming patients. The text generator can generate paragraphs and history about the patient so that it is easy for the doctors to diagnose the patient and the history-taking process of the doctor is also followed in this. This is going to be helpful for the doctor because the old method to write constantly on paper is neglected and the program will be very user friendly it just needs prior references regarding the disease and it can automatically generate the history of a patient.

1.3 Motivation

A regular experience with the physician brings us some questions regarding health issues e.g.: What are your symptoms? For how many days the issue is there? etc. It is followed by a handwritten prescription consisting of the name of the clinic and the doctor written on it. It is a time-consuming practice for the doctor, especially when a hundred patients are waiting outside the chamber. A verbal conversation with multiple doctors confirmed the same Further exploration of the problem identified the problem of writing about the symptoms and health issues using a pen and paper will not cover all the areas of prescription and is also very tedious work to do. So we decided to develop a history generator software that can take the history of patients about the disease in a quicker, more precise, and more efficient manner.

1.4 Project Challenges

The biggest challenge in our software is collecting the dataset of all the diseases which is existing and putting it in a format because our tool is to generate history for a patient so it requires a dataset it the tool can refer to and then generate the history with the help of data, it will also require to store the dataset in a database of different patients because there will be a similarity in the symptoms of some patients so it can create a contradiction between datasets so we need to store both the patient's data in the database so that it does create complexity. There is a particular language model which the doctors use the program do not know about that so we have to import the language model manually, this is a major challenge Use of machine learning to make the system learn the next generation for the first few users we have to manually type it onto software then the system will learn by using different machine learning concepts.

1.5 Proposed Solution

The proposed solution is that the text given to it will generate the history of the patient and disease accordingly using the database provided to it.

3

Firstly, at the initial stage, we have to use machine learning and teach the system to generate the history accordingly because that is the most important part of the product. After a few users, it will compare the situations of the next user with the previous data stored, and then it will be effective after that to generate the history because the system has already learned those things and can generate them according to the situation. Our solution is never fully fulfilled because it every time learns new things if some new disease is discovered then it will again learn the history and details related to that disease.

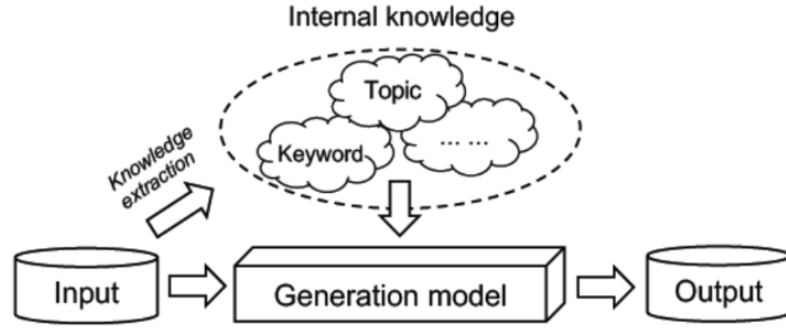
Chapter 2

Literature Review

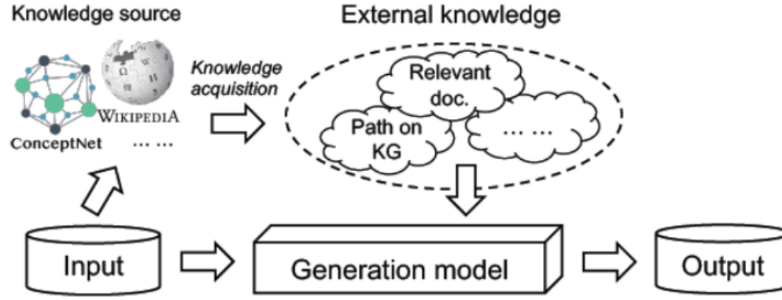
2.1 Related work And State of the Art (Latest work)

Big data has an unstructured form (mostly text data) called big text. Social media contributes a lot to good text. Additionally, other online sources such as online news portals, blogs, health records, and government data provide rich textual data for research. Deep learning has great potential in the NLP space. Many studies have contributed to text classification tasks using deep neural networks. Successful architectures include convolutional neural network (CNN)-based models such as VD-CNN and DP-CNN, recurrent neural network (RNN)-based models such as SANN, and perception-based models such as HAN and DiSAN. increase. These models use pre-trained word embeddings to improve performance on downstream tasks. Compared to previous models, BERT does not require a specific architecture for each downstream task, so this model has been very successful in many NLP downstream tasks. Data augmentation is a machine learning technique that artificially increases the amount of training data through label-preserving transformations. A related concept to data augmentation is label preservation. It describes a transformation of the training data that preserves class information. This is very important in data augmentation research. Otherwise, incorrectly classified data will be generated. Data augmentation is seen as a difficult task in NLP because it is difficult to define text transformations that preserve labels. So far, many methods have

been tested in research. These include methods for swapping, deleting, misspelling sources, paraphrasing and synonym replacement, strict embeddings, and words predicted by word-level language models. The study of natural language generation first became known as the process of converting non-verbal machine representations of information into structured natural language. Recurrent networks have been around for a long time, but they have been very difficult to train on sequential data with longterm time dependencies due to the problem of vanishing or exploding gradients. The authors named it model multiplicative RNN (MRNN) due to its multiplicative nature of transforming the hidden state weight matrix into a function of the current input. The effectiveness of the proposed model is proved by constructing a large character-level LM trained on three different datasets to predict the next character given a string as input. A very notable study on Chinese poetry generation using RNNs was done by Zhang et al. This implementation takes as input some keywords of a poem and creates a set of all possible phrases containing the keywords. Among these, n sentences are selected using trigram RNN-LM with a character-level batch decoder to generate the first line of the poem. Then all the next lines of the poem are iteratively generated from the previous lines using three different models: the convoluted set model, the recursive context model, and the recursive generative model. 5 2019, Egonmwan et al. used the seq2seq architecture for the paraphrase generation work. The novelty of this work lies in combining the advantages of the Transformer model with the seq2seq architecture. The encoder he consisted of two layers. The first is a transformation model for rich audio properties and the second is a unidirectional repeating unit. GloVe embedding was used to render the input sequence. This framework has proven to be very good in practice, improving the state of the art on two different paraphrasing datasets. In general, knowledge is familiarity, awareness, or understanding related to a particular subject. In NLG systems, knowledge is awareness and understanding of the input text and its surrounding context. These knowledge sources can be classified as internal knowledge and external knowledge. Internal knowledge generation takes place within the input text, including but not limited to keywords, topics, linguistic features, and internal diagram structures. External knowledge acquisition occurs when knowledge is provided by external sources, including but not limited to knowledge bases, external knowledge



Source: A Survey of Knowledge-Enhanced Text Generation



Source: A Survey of Knowledge-Enhanced Text Generation

Figure 2.1: Survey of Knowledge-Enhanced Text generation

graphs, and authored text. These sources of information are used as knowledge by various neural representation learning methods and provide information that can be applied to improve the process of text generation (e.g. 6 common sense triplets, topic words, reviews, background documents). Furthermore, knowledge introduces interpretability of models with explicit semantics. This line of research that incorporates knowledge into text generation is known as knowledge-based text generation.

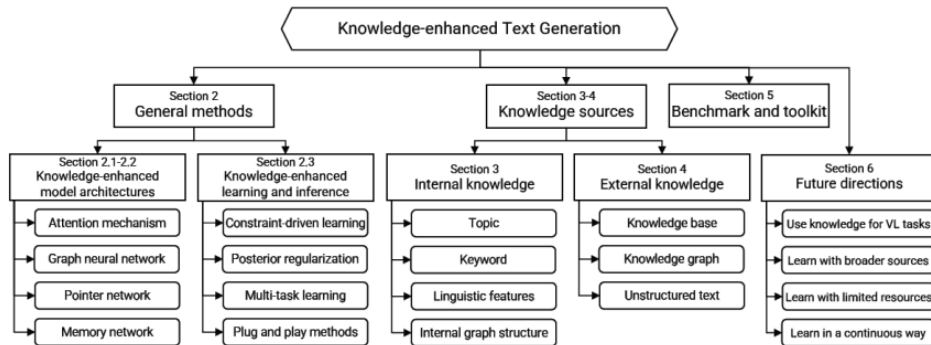


Figure 2.2: Knowledge enhanced text generation

In recent years, there has been growing interest in developing ways to integrate knowledge into NLGs beyond the input text. However, there is no comprehensive coverage of this research topic. The corresponding research laid the foundation for discussion of this topic. Galbacea et al. and Gat et al. reviewed the model architecture of the NLG core task, but did not discuss the knowledge-based method. gave an overview of knowledge graph techniques that can be used to improve NLG. kings. summarizes how structural knowledge, such as knowledge bases and knowledge graphs, can be represented for reading and retrieval. It is intended to provide NLG researchers with integration and guidance to related research. Our research includes an in-depth discussion of how NLG can benefit from recent advances in deep learning and artificial intelligence, including technologies such as graph neural networks, reinforcement learning, and neural topic modelling is included.

2.2 Limitation of State of the Art techniques

During the evaluation, restrictions were placed on the language model used. It's not clear if the performance gains remain the same even when applicable to other language models. The GPT-3 model seems like the next smart choice for boosting results. However, due to the model's high resource utilization, fine-tuning steps that are likely to be necessary cannot currently be performed. Nevertheless, its size and linguistic expressiveness make it particularly useful for addressing the challenge that pre-trained models may not see improvement with data augmentation. Regarding the study of, future studies could test the proposed method on a transformer model. It will also be interesting to see how the much faster smaller language models perform. In addition, there is also the possibility to fully automate the filtering procedure. This makes it even more universally applicable, even when human effort is already very low. Despite many efforts to augment data, the big data barrier problem mentioned at the beginning remains important. However, in the future, according to various assumptions, if very large models such as GPT-3 by 7 can solve these problems better, we will face the resource barrier problem and only large companies will train these models and can be used. The first challenge in knowledge-based NLG is obtaining

useful and relevant knowledge from various sources. There is an increasing focus on discovering knowledge from topics, keywords, knowledge bases, knowledge graphs, and knowledge base text. The second challenge is to effectively comprehend and use the acquired knowledge to facilitate the production of texts. Several methods have been explored to improve the encoder/decoder architecture. More specifically, since knowledge can come from a variety of sources, we first divide the existing knowledge base text generation work into two categories: internal knowledge base text generation and external knowledge base text generation. The separation of internal and external knowledge is pervasive in business management which can be analogous with knowledge enhanced text generation. We classify newer knowledge-based text generation methods (designated as M1, M2, etc.) that have evolved from the way knowledge is extracted and integrated into each section’s text generation process. In addition, each section reviews methods for different natural language generation applications that help practitioners select, learn, and use methods.

2.3 Discussion and future direction

The network is tuned by context vectors drawn from the same GloVe language model used to embed the words of each sequence into the vector space. Two separate networks are used for training and inference. Ray search optimization was used to generate the sequence with the largest log-likelihood value. In addition, we have proposed an evaluation technique involving human evaluation according to different criteria of outcome. We then compared the performance of various recurrent architectures. The base model proved highly effective in achieving meaningful results characterized by the target context. The success of the research presented in this paper lays the foundation for many other natural language generation-based applications that require the inclusion of preferred contextual information in the target sequence, even with short training times and limited data.

2.4 Conclusion

In this study, we present a comprehensive overview of current representative research efforts and trends in knowledge-based text generation, which we hope will facilitate future research. In summary, this study aims to answer two of his questions that frequently arise in knowledge-based text generation. How to acquire knowledge and how to integrate knowledge to facilitate text generation. Based on knowledge acquisition, the main content of our investigation is divided into three sections according to different sources of knowledge expansion. Based on knowledge incorporation, we first present general methods for incorporating knowledge into text generation, and then in each section a set of specific ideas and techniques for incorporating knowledge to improve text generation systems. I'll explain a workaround. Also, in each section, we review different text generation applications that help practitioners learn how to choose and use methods.

Chapter 3

Problem Definition and Scope

3.1 Problem statement

To Develop a system with text generator using Natural Language Processing for Doctors

3.2 Goals and Objectives

A regular experience with the physician brings us some questions regarding health issues e.g.: What are your symptoms? For how many days the issue is there? etc. It is followed by a handwritten prescription consisting of the name of the clinic and the doctor written on it. It is a time-consuming practice for the doctor, especially when a hundred patients are waiting outside the chamber. A verbal conversation with multiple doctors confirmed the same Further exploration of the problem identified the problem of writing about the symptoms and health issues using a pen and paper will not cover all the areas of prescription and is also very tedious work to do. So we decided to develop a history generator software that can take the history of patients about the disease in a quicker, more precise, and more efficient manner.

3.3 Scope and Major Constraints

Our project is based on the generation of text by using Natural language processing and Natural language generation, so it generates the history according to the dataset which we have stored in the database. We can use similar concepts in many other fields like report writing, summary writing, and scripting. Report writing generates the text simply by the previous sentences and format which is been given in first, summary writing and scripting also follow the same thing.

3.4 Expected Outcomes

The doctor is giving input in the form of the disease, and symptoms and from when it is occurring using these inputs the program understands and frame the history. Firstly, at the initial stage, we have to use machine learning and teach the system to generate the history accordingly because that is the most important part of the product. After a few users, it will compare the situations of the next user with the previous data stored, and then it will be effective after that to generate the history because the system has already learned those things and can generate them according to the situation. 9 Our solution is never fully fulfilled because it every time learns new things if some new disease is discovered then it will again learn the history and details related to that disease.

Chapter 4

System Requirement Specification

4.1 Overall Description

4.1.1 Product Perspective

Python libraries, Word cloud, python compiler

4.2 Specific Requirements

4.2.1 User Requirements

Internet connection and Mobile / PC

4.2.2 External Interface Requirements

Keyboard, Mouse and Monitor

4.2.3 Functional Requirements

Functional Requirements are as follows

Table 4.1: Functional Requirements of Topic

Functional requirement ID	Function Name	Function Requirement description
Fr ₁	Login	The user should enter their credentials to access their data and give system information about the user
Fr ₂	Import Data	Users should enter Book, and Database, as prerequisites to help the system learn
Fr ₃	Generate text	The system will analyze the user's word entered in the file and give suggestions to user about the next word or sentence that the user wants through the database entered and NLP
Fr ₄	File History	All the data and files created by the user will be saved in the cloud and can be viewed later
Fr ₅	Upload files	Users can upload important documents to the cloud

4.2.4 Performance Requirement

Pc with minimum intel i3 6th gen and 4 Gb ram and SSD.

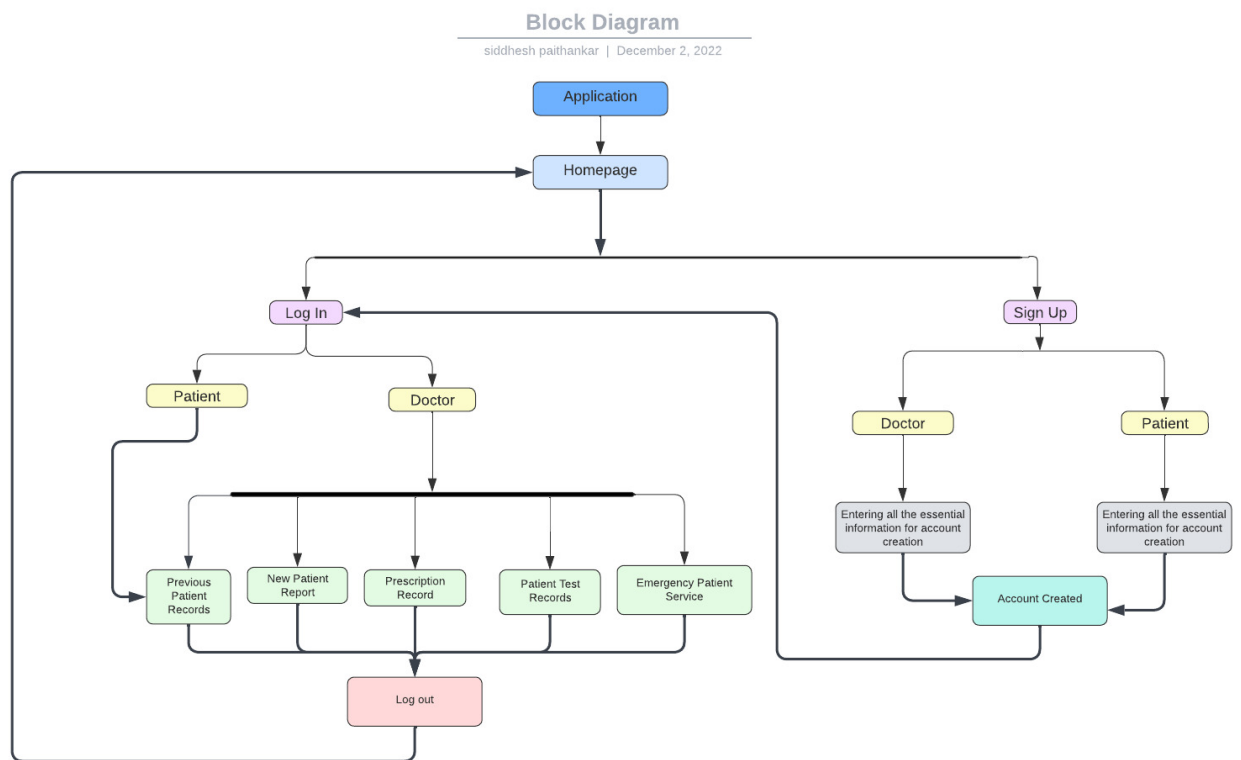


Figure 4.1: Block Diagram

Chapter 5

Methodology

5.1 System Architecture

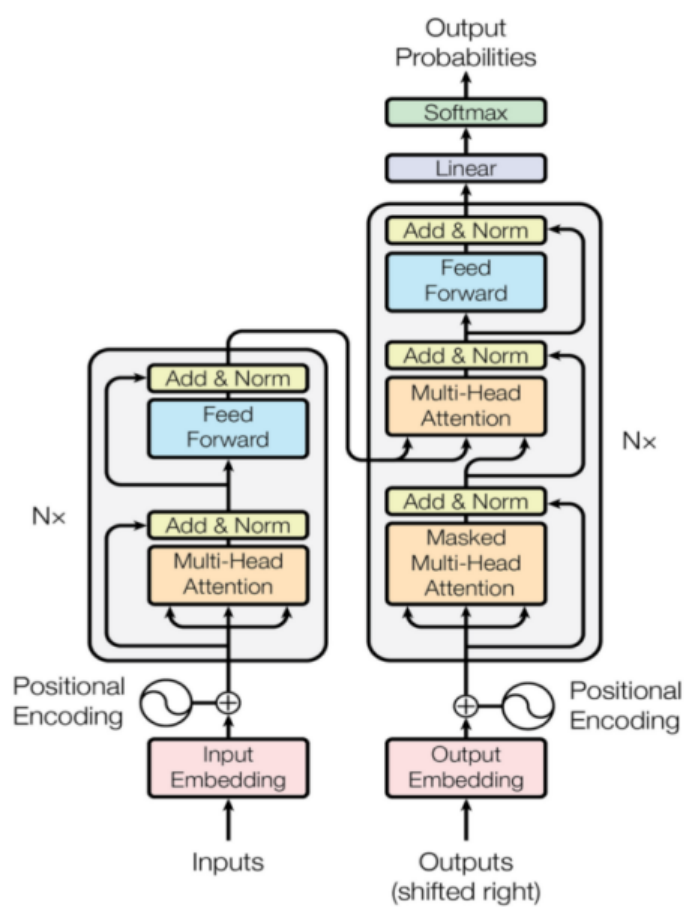


Figure 5.1: The Transformer - Model Architecture

The encoder maps an input sequence of symbolic representation (x, \dots, x) to a se-

quence of representation $z = (z, \dots, z)$. Think of them as the result of self-awareness with some post-processing. If you specify z , the decoder produces an output sequence (y_1, \dots, y_7) of symbols element by element. By computing the self-perception multiple times with different sets of q , k , and v vectors, and averaging all these outputs to get the final z , the model can be focused on different positions. Instead of dealing with these humongous vectors and averaging multiple outputs, we reduce the size of our k , q and v vectors to some smaller dimension — reduce size of W_q , W_k , and W_v matrices as well. We keep the multiple sets (h) of k , q and v and refer to each set as an “attention head”, hence the name multi-headed attention. And lastly, instead of averaging to get final z , we concatenate them. The word embedding itself lacks the location information achieved in RNNs due to the sequential nature of RNNs. On the other hand, SoftMax’s self-awareness loses such location information. To preserve location information, the transformer inserts a vector into each input embedding (word embeddings can be used to match input words). These vectors follow a certain periodic function (e.g. different sine/cosine combinations at different frequencies, i.e. they are not synchronized with each other). A model can learn this and use the values to determine the position of individual words that are related to each other.

5.2 Mathematical Modeling

Given a seq2seq model parameterized by θ , a source sequence containing n tokens $x = x_1, \dots, x_n$ and a target sequence containing m tokens $y = y_1, \dots, y_m$. We can factorize the generation probability of y conditioned on x as follows:

$$p(y|x, \theta) = \prod_{t=1}^m p(y_t | y_{<t}, x, \theta)$$

Figure 5.2: Factorization of generation Probability

The most general form of our proposed BARTSCORE is shown below, where we use the weighted log probability of one text y given another text x . The weights are used to put different emphasis on different tokens, which can be instantiated using

different methods like Inverse Document Frequency.

$$\text{BARTSCORE} = \sum_{t=1}^m \omega_t \log p(\mathbf{y}_t | \mathbf{y}_{<t}, \mathbf{x}, \theta)$$

Figure 5.3: Bart Score

Due to its generation task-based formulation and ability to utilize the entirety of BART's pre-trained parameters, BARTSCORE can be flexibly used in different evaluation scenarios. BERT is a deep learning architecture that can be used for downstream NLP tasks. The architecture consists of stacked encoder layers from transformers. BERT has two main steps: pre-training and fine-tuning. During pre-training, BERT uses two unsupervised tasks, Masked Language Model (MLM) and Next Sentence Prediction (NSP), to create a pre-trained model using large-scale unlabeled data. Trained on a corpus. For fine-tuning, the model is initialized with pre-trained parameters and all parameters are fine tuned using labeled data for a specific task such as classification. A simple SoftMax classifier is added on top of the model to predict the probability of the label c given in the formula where W is the task-specific parameter matrix. Jointly optimize all parameters of BERT and W by maximizing the log-likelihood of the right label.

$$p(c|h) = \text{softmax}(Wh)$$

Figure 5.4: Bert

Long-Short-Term Memory (LSTM) LSTM networks were first identified by Hochreiter et al. Introduced in 1997 as a cure for the gradient problem in standard RNNs, with many other improvements, it has recently become most popular for sequence modeling. LSTMs maintain and propagate high-level cell states at each time step in addition to RNN-like activation signals. LSTMs use three different gated units: update gates, output gates, and forget gates to determine, store, and regulate the flow of information related by long-term relationships.

Gated recurrent unit (GRU) GRU, proposed by Cho et al., is a much simpler version of LSTM. In contrast to LSTM, the cell state information at time step t is the same

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c[c_{t-1}, a_{t-1}, x_t] + b_c) \\
u_t &= \sigma(W_u[c_{t-1}, a_{t-1}, x_t] + b_u) \\
f_t &= \sigma(W_f[c_{t-1}, a_{t-1}, x_t] + b_f) \\
o_t &= \sigma(W_o[c_{t-1}, a_{t-1}, x_t] + b_o) \\
c_t &= u_t * \tilde{c}_t + f_t * c_{t-1} \\
a_t &= o_t * \tanh(c_t)
\end{aligned}$$

Figure 5.5: LSTM

as the activation of that time step. Moreover, GRUs employ only two gates-the update gate and the relevance gate. Due to fewer gates, GRUs have less control over maintaining more important long-term information. But the lower number of gates also means that GRUs have a lower number of parameters to train, thus, making them easier and faster to train compared to LSTMs.

$$\begin{aligned}
\tilde{c}_t &= \tanh(W_c[r_t, a_{t-1}, x_t] + b_c) \\
u_t &= \sigma(W_u[a_{t-1}, x_t] + b_u) \\
r_t &= \sigma(W_r[a_{t-1}, x_t] + b_r) \\
c_t &= u_t * \tilde{c}_t + (1 - u_t) * c_{t-1}
\end{aligned}$$

Figure 5.6: GRU

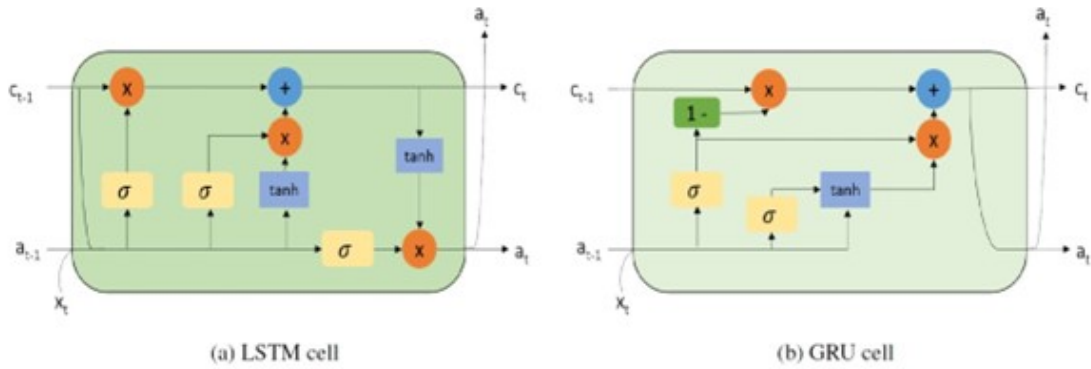


Figure 5.7: LSTM - GRU

5.3 Algorithms

Unlike batch gradient descent, the stochastic gradient descent (SGD) algorithm iteratively computes gradient-based optimization and applies it to each training sample. SGD increases the number of operations, but convergence to a local minimum is almost certain if the learning rate is small enough. Additionally, progress can be made on large training stacks without calculating the cumulative cost of the entire training stack. For a function $f()$ with parameter θ , each iteration of SGD gives an approximation of the gradient.

$$\theta = \theta - \alpha \times \nabla_{\theta} f_t(\theta)$$

Figure 5.8: SGD

Here, $f_t()$ refers to the value of objective function for t th training example, where $t = 1, 2, \dots, n$. $\nabla_{\theta} f_t(\theta)$ refers to the first-order gradient of the function and α is the learning rate. RMSprop algorithm proposed by Hinton et al. aims to integrate adaptive learning rates with mini-batch gradient descent algorithms. The algorithm adaptively estimates the parameters by keeping an exponentially weighted moving average of the squared slopes of each parameter and dividing by the square root of these values. As a result, RMSprop performs very well for nonstationary objective functions, accelerating the learning process and improving convergence. It also significantly reduces memory requirements. However, this method makes no attempt to correct for bias terms, so the algorithm's initial approximation deviates from the actual data points. The moment vector $V(\theta, T)$ and parameter θ for the T th mini-batch are computed.

$$V(\theta, T) = \gamma \times V(\theta, T - 1) + (1 - \gamma) \times (\nabla_{\theta} f(\theta, T))^2$$

Figure 5.9: RMSProp Algorithm

Here, γ is the decay rate and $\nabla_{\theta} f(\theta, T)$ denotes the first-order gradient of the objective function with the parameter θ at T th iteration. The gradient is used for optimization as shown below:

$$\theta_{t+1} = \theta_t - \alpha \times \frac{\nabla_{\theta} f(\theta_t)}{\sqrt{\sum_1^t (\nabla_{\theta} f(\theta_t))^2}}$$

Figure 5.10: Gradient used for Optimisation

Adagrad or Adaptive Gradient Algorithm by Duchi et al. is an adaptive version of the SGD algorithm with an adaptive learning rate for each parameter. In an online environment the algorithm works quite well, especially on sparse gradients. The algorithm works by scaling the learning rate by a factor that reflects the sparsity of the data. The update operation in Adagrad for parameter θ is given below:

$$\theta_{t+1} = \theta_t - \alpha \times \frac{\nabla_{\theta} f(\theta_t)}{\sqrt{\sum_1^t (\nabla_{\theta} f(\theta_t))^2}}$$

Figure 5.11: Operation in Adagrad

Adaptive motion estimation presented by Adam or Kingma et al. Another very effective adaptive optimization technique for stochastic cost functions. It is suitable for both stationary and non-stationary loss functions and is also very useful for noisy losses. In fact, this algorithm combines the benefits of impulse-based optimization with the RMSprop algorithm. As a result, the algorithm is memory efficient, converges quickly, and the approximation is also very accurate due to the built-in bias correction. As such, it is very well suited for many machine learning problems such as NLP and computer vision. The algorithm estimates two moments:

$$\begin{aligned} V_1(\theta, t) &= \beta_1 \times V(\theta, t-1) + (1 - \beta_1) \times \nabla_{\theta} f(\theta, t) \\ V_2(\theta, t) &= \beta_2 \times V(\theta, t-1) + (1 - \beta_2) \times (\nabla_{\theta} f(\theta, t))^2 \end{aligned}$$

Figure 5.12: Adaptive Motion

Here, 1 and 2 denote the two decay rates for exponential moment estimation for the moment vectors, V_1 and V_2 respectively. These moment vectors are then bias-corrected as in and respectively, before being used to optimize the parameter as

$$\begin{aligned}\bar{V}_1(\theta, t) &= \frac{V_1(\theta, t)}{(1 - \beta_1^t)} \\ \bar{V}_2(\theta, t) &= \frac{V_2(\theta, t)}{(1 - \beta_2^t)} \\ \theta_t &= \theta_{t-1} - \alpha \times \frac{\bar{V}_1(\theta, t)}{\sqrt{\bar{V}_2(\theta, t) + \epsilon}}\end{aligned}$$

Figure 5.13: Vectors Bias-correction

Here, ϵ is an infinitesimal number used to prevent a possible division by zero when the denominator is very small and close to zero.

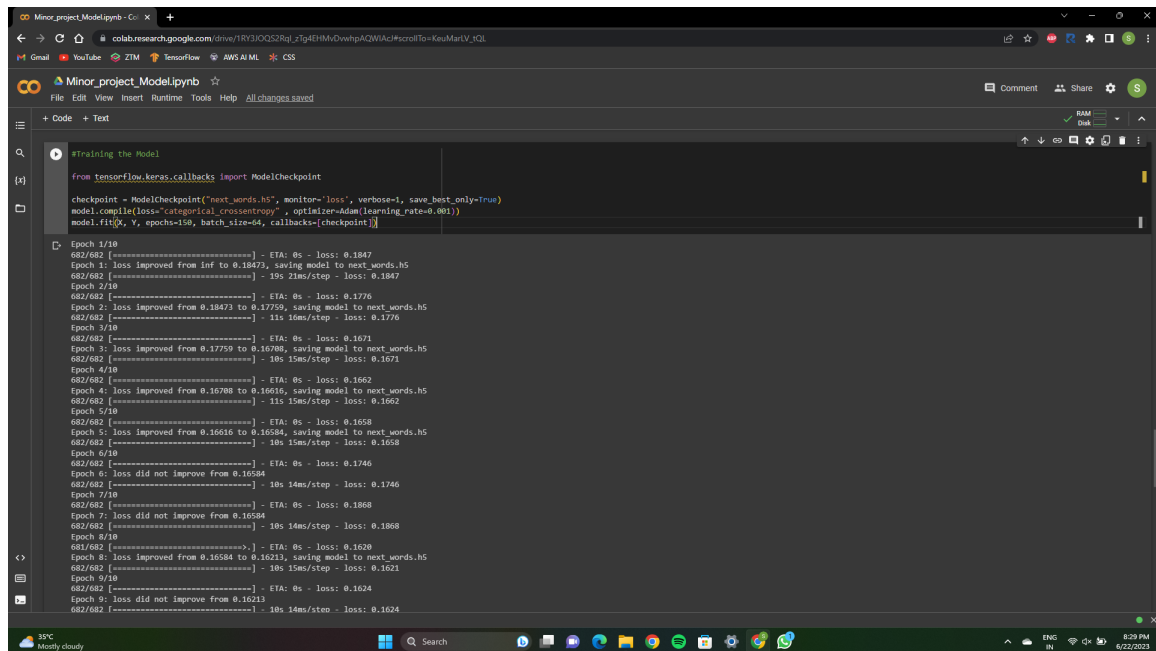
Chapter 6

Implementation

6.1 Description

This LSTM model has been meticulously crafted to facilitate the generation of text, leveraging its ability to learn from a supplied text file during the training process. To elicit accurate output, the user is expected to provide a minimum of three words as input, upon which the model generates coherent text based on the input prompt. Originally conceived to cater specifically to medical professionals, this advanced model aids doctors in meticulously documenting patient histories and maintaining comprehensive records. However, the versatility of this model extends far beyond the realm of healthcare, as it can prove invaluable to various other professions, including but not limited to novelists, screenplay writers, and report authors, empowering them to produce high-quality content with ease and efficiency.

6.2 Output

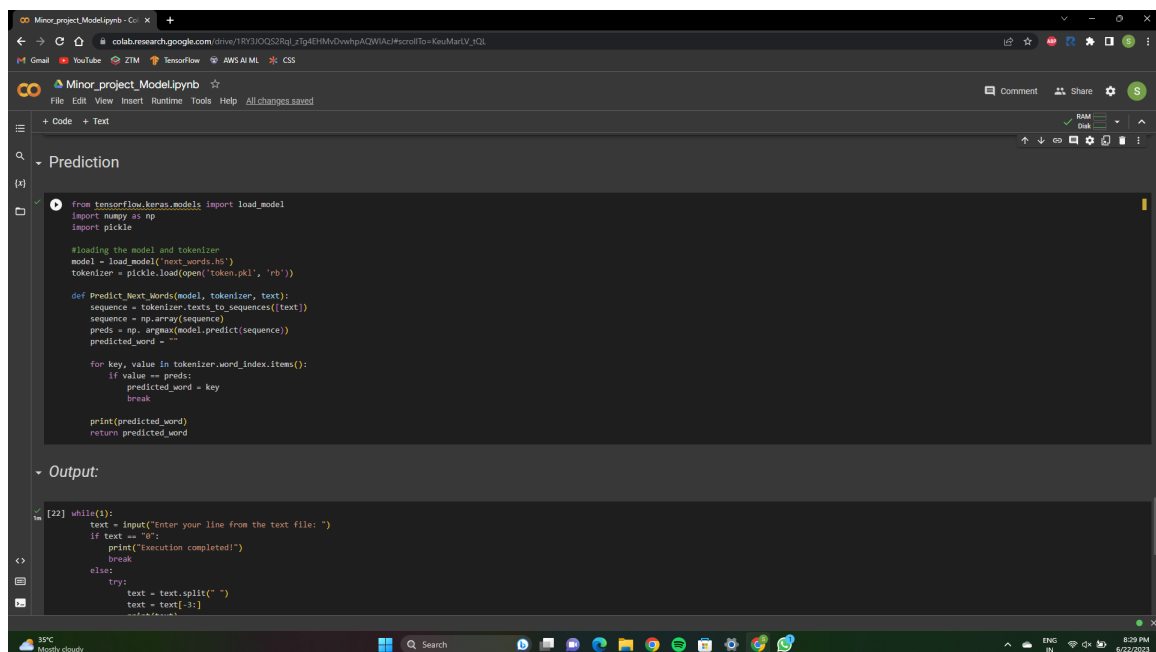


```
#Training the Model
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("next_words.h5", monitor='loss', verbose=1, save_best_only=True)
model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001))
model.fit(X, y, epochs=10, batch_size=64, callbacks=[checkpoint])

Epoch 1/10
682/682 [=====] - ETA: 0s - loss: 0.1847
Epoch 1: loss improved from inf to 0.18473, saving model to next_words.h5
682/682 [=====] - 19s 21ms/step - loss: 0.1847
Epoch 2/10
682/682 [=====] - ETA: 0s - loss: 0.1776
Epoch 2: loss improved from 0.18473 to 0.17759, saving model to next_words.h5
682/682 [=====] - 11s 16ms/step - loss: 0.1776
Epoch 3/10
682/682 [=====] - ETA: 0s - loss: 0.1671
Epoch 3: loss improved from 0.17759 to 0.16708, saving model to next_words.h5
682/682 [=====] - 10s 15ms/step - loss: 0.1671
Epoch 4/10
682/682 [=====] - ETA: 0s - loss: 0.1662
Epoch 4: loss improved from 0.16708 to 0.16616, saving model to next_words.h5
682/682 [=====] - 11s 15ms/step - loss: 0.1662
Epoch 5/10
682/682 [=====] - ETA: 0s - loss: 0.1658
Epoch 5: loss improved from 0.16616 to 0.16584, saving model to next_words.h5
682/682 [=====] - 10s 15ms/step - loss: 0.1658
Epoch 6/10
682/682 [=====] - ETA: 0s - loss: 0.1746
Epoch 6: loss did not improve from 0.16584
682/682 [=====] - 10s 14ms/step - loss: 0.1746
Epoch 7/10
682/682 [=====] - ETA: 0s - loss: 0.1868
Epoch 7: loss did not improve from 0.16584
682/682 [=====] - 10s 14ms/step - loss: 0.1868
Epoch 8/10
681/682 [=====] - ETA: 0s - loss: 0.1628
Epoch 8: loss improved from 0.16584 to 0.16213, saving model to next_words.h5
682/682 [=====] - 10s 15ms/step - loss: 0.1621
Epoch 9/10
682/682 [=====] - ETA: 0s - loss: 0.1624
Epoch 9: loss did not improve from 0.16213
682/682 [=====] - 10s 14ms/step - loss: 0.1624
```

Figure 6.1: Output 1



```
from tensorflow.keras.models import load_model
import numpy as np
import pickle

#Loading the model and tokenizer
model = load_model("next_words.h5")
tokenizer = pickle.load(open("token.pkl", 'rb'))

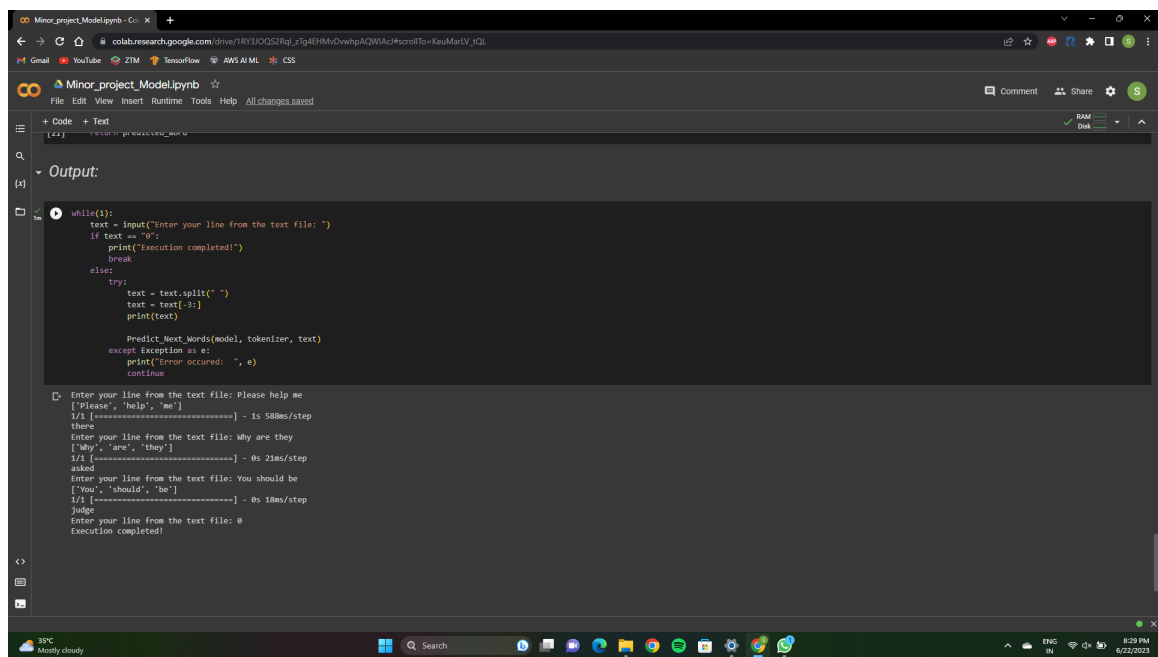
def Predict_Next_Words(model, tokenizer, text):
    sequence = tokenizer.texts_to_sequences([text])
    sequence = np.array(sequence)
    preds = np.argmax(model.predict(sequence))
    predicted_word = ""

    for key, value in tokenizer.word_index.items():
        if value == preds:
            predicted_word = key
            break

    print(predicted_word)
    return predicted_word

[22] while(1):
    text = input("Enter your line from the text file: ")
    if text == "0":
        print("Execution completed!")
        break
    else:
        try:
            text = text.split(" ")
            text = text[:3]
        except:
```

Figure 6.2: Output 2



The screenshot shows a Google Colab notebook interface. The browser address bar displays a URL from colabresearch.google.com. The notebook title is "Minor_project_Model.ipynb". The code editor contains a Python script with a `while()` loop for text processing and a `Predict_Next_Words` function call. The output area shows the execution of this script, including user prompts like "Enter your line from the text file: Please help me" and the resulting word predictions such as `['Please', 'help', 'me']` and `['Why', 'are', 'they']`. The interface includes standard Colab controls like "File", "Edit", "View", "Runtime", "Tools", and "Help" menus, as well as a "RAM Disk" indicator.

```
while():
    text = input("Enter your line from the text file: ")
    if text == '0':
        print("Execution completed!")
        break
    else:
        try:
            text = text.split(" ")
            text = text[:-1]
            print(text)

            Predict_Next_Words(model, tokenizer, text)
        except Exception as e:
            print("Error occurred: ", e)
            continue
```

Enter your line from the text file: Please help me
['Please', 'help', 'me']
1/1 [=====] - 1s 588ms/step
there
Enter your line from the text file: Why are they
['Why', 'are', 'they']
1/1 [=====] - 0s 218ms/step
asked
Enter your line from the text file: You should be
['You', 'should', 'be']
1/1 [=====] - 0s 188ms/step
Judge
Enter your line from the text file: 0
Execution completed!

Figure 6.3: Output 3

Chapter 7

Result Analysis/Performance Evaluation

7.1 Result Analysis of Objective 1

To perform a result analysis and performance evaluation of a text generator using NLP (Natural Language Processing) and LSTM (Long Short-Term Memory) for doctors to write patient's history, several factors should be considered. Here's a framework to assess its effectiveness:

1. Data quality and availability: The quality and quantity of the dataset used to train the text generator significantly impact its performance. Evaluating the diversity, size, and relevance of the data used is crucial to understand the generator's limitations and biases.
2. Language understanding: Assess how well the text generator understands medical terminology, context, and nuances. Evaluate whether it accurately represents the patient's history, including symptoms, diagnosis, treatment, and any other relevant information. Manually review generated texts to check for errors, missing details, or incorrect interpretations.
3. Coherence and readability: Measure the overall coherence and readability of the generated text. Evaluate if the generated patient history exhibits a logical flow,

maintains proper grammar, and adheres to medical writing conventions. Incoherent or confusing texts may hinder the usefulness and acceptance of the generated history.

4. Medical accuracy and completeness: Verify the accuracy and completeness of the generated patient history in terms of medical facts and information. Compare the generated history with established medical guidelines, expert opinions, or gold-standard patient records. Identify any inaccuracies, omissions, or potential biases introduced by the generator.

5. Customizability and flexibility: Assess the generator's ability to adapt to different patient cases, medical specialties, or specific requirements of doctors. Evaluate if it can incorporate different templates, formats, or guidelines to produce patient histories that align with various medical practices or specialties.

6. Human feedback integration: Incorporate feedback from doctors and medical professionals who review and use the generated patient histories. Their subjective assessment and insights are valuable for improving the generator's performance, addressing its limitations, and identifying areas of improvement.

7. Performance metrics: Quantitative metrics can be used to evaluate the performance of the text generator, such as perplexity, BLEU score, or ROUGE score. These metrics provide an objective measure of the generator's language generation quality, fluency, and similarity to human-generated texts.

8. User satisfaction: Collect feedback from doctors who use the generated patient histories in their practice. Assess their satisfaction, acceptance, and perceived usefulness of the generated texts. Analyze the feedback to identify areas for improvement and to understand how well the text generator aligns with their needs and expectations.

It's important to note that the evaluation process may involve iterative improvements, fine-tuning, and continuous monitoring to enhance the text generator's performance over time.

7.2 Result Analysis of Objective 2

Accuracy: Evaluate the accuracy of the generated patient history compared to the ground truth. Measure how well the text generator captures the relevant information from the input and produces accurate medical descriptions. You can calculate metrics such as token-level accuracy or exact match accuracy.

7.3 Result Analysis of Objective 3

Medical Terminology and Language: Check if the text generator correctly uses medical terminology and appropriate language. Evaluate whether the generated text contains relevant medical concepts, terminology, and expressions that doctors typically use when writing patient histories.

7.4 Result Analysis of Objective 4

Adequacy of Information: Evaluate the adequacy of the generated patient history in terms of the required information for medical analysis and decision-making. Determine if the generated text includes relevant details, such as past medical history, present symptoms, diagnostic test results, and treatment plans.

By considering these objectives, you can perform a comprehensive result analysis and performance evaluation of the text generator, providing insights into its strengths, limitations, and areas for improvement.

Chapter 8

Conclusion

8.1 Conclusion

In conclusion, our system aims to revolutionize the way patient histories are generated and prescriptions are provided in India's healthcare system. With the enormous population and limited availability of doctors, it is crucial to find innovative solutions to optimize their time and improve patient care. By leveraging NLP and NLG technologies, we propose an intelligent system that can generate patient histories based on provided disease information, while adhering to the required format and incorporating medical guidelines.

This system has the potential to significantly enhance time efficiency for doctors by automating the process of generating patient histories, allowing them to focus more on diagnosing and treating patients. By streamlining the documentation process, doctors can save valuable time, reduce the risk of errors, and improve overall efficiency in their clinical practice.

Moreover, the system's ability to learn and adapt to the format and guidelines followed by doctors ensures that the generated histories are accurate and aligned with the professional standards of medical practice. This not only benefits doctors by providing a reliable and standardized documentation system, but also enhances patient care through accurate diagnoses and appropriate prescriptions.

By harnessing the power of NLP and NLG, our system aims to bridge the gap between the limited availability of doctors and the increasing healthcare needs of the population. It strives to deliver time efficiency and effective prescription capabilities, ultimately improving healthcare outcomes for both doctors and patients in India.

8.2 Future Scope

The future scope for a text generator using NLP and LSTM for doctors to write patient's history is promising. Some potential areas of improvement and expansion are as follows:

1. Domain-specific training: Train the text generator using a larger and more diverse dataset of patient histories specific to different medical domains. This will improve the system's ability to generate accurate and contextually relevant patient histories for a wider range of medical conditions.
2. Integration of medical knowledge bases: Incorporate medical knowledge bases, such as electronic health records (EHRs) or medical literature databases, to enhance the system's understanding of medical concepts, terminology, and treatment guidelines. This integration can help generate more accurate and up-to-date patient histories.
3. Customization and personalization: Develop methods to allow doctors to customize the text generator according to their preferences and specific practice requirements. This could involve adapting the language style, formatting, or templates to match individual doctor's preferences.
4. Real-time assistance: Explore the possibility of integrating the text generator into a real-time clinical decision support system. This would enable doctors to receive instant suggestions or prompts based on patient data, facilitating faster and more accurate documentation.
5. Contextual generation: Enhance the system's ability to consider the broader patient context, including previous medical history, medications, allergies, and co-

existing conditions. By taking into account these factors, the text generator can generate patient histories that are more comprehensive and tailored to each individual patient.

6. Multimodal input and output: Investigate the incorporation of multimodal input, such as integrating speech recognition or image analysis, to capture information beyond text-based patient histories. Additionally, explore generating output in multiple formats, including structured data, graphical representations, or voice-based summaries.

7. Feedback and reinforcement learning: Implement mechanisms to collect feedback from doctors and patients to continuously improve the performance of the text generator. Utilize reinforcement learning techniques to fine-tune the model based on this feedback, ensuring continuous enhancement and adaptation to evolving medical practices.

8. Ethical considerations and data privacy: Address ethical concerns and ensure compliance with data privacy regulations when collecting, storing, and utilizing patient data for training and generating patient histories. Develop robust mechanisms for data anonymization and secure data handling to maintain patient confidentiality.

By exploring these future directions, a text generator using NLP and LSTM for doctors to write patient's history can evolve into a valuable tool that significantly improves efficiency, accuracy, and the overall quality of healthcare documentation, ultimately benefiting both doctors and patients.

Appendices

Appendix A

TensorFlow

TensorFlow is a popular framework of machine learning and deep learning. It is a free and open-source library which is released on 9 November 2015 and developed by Google Brain Team. It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.

TensorFlow can train and run the deep neural networks for image recognition, handwritten digit classification, recurrent neural network, word embedding, natural language processing, video detection, and many more. TensorFlow is run on multiple CPUs or GPUs and also mobile operating systems.

The word TensorFlow is made by two words, i.e., Tensor and Flow

Tensor is a multidimensional array Flow is used to define the flow of data in operation.

References

- [1] Weizhe Yuan, Graham Neubig, Pengfei Liu, Carnegie Mellon University, “Evaluating Generated Text as Text Generation”, June 22, 2021 Available:
<https://drive.google.com/file/d/14MwObRfI3s7ObnTHNbpi5h6vnVTvNUyt/view?usp=sharing>
- [2] Prashant Kaushik, Sunil Prasad, “Context Aware GAN for sequence text generation in tensor-flow lite for android AI chat application”, September 9, 2021 Available:
<https://drive.google.com/file/d/1dgdVbSpslOULaiW0OPEcSxCPoypb3uH/view?usp=sharing>
- [3] WENHAO YU, University of Notre Dame, USA, “A Survey of Knowledge-Enhanced Text Generation”, January 24, 2022 Available:
<https://drive.google.com/file/d/1icjgfQkvGQ67MlIAizNhMOHqg3Jxycv5/view?usp=sharing>
- [4] Markus Bayer¹, Marc-André Kaufhold¹, “Data augmentation in natural language processing: a novel text generation approach for long and short text classifiers”, March 22, 2022 Available:
<https://drive.google.com/file/d/1L2-2q2IMVEDjyuLK50CiRnagZqvZ4qbp/view?usp=sharing>
- [5] Kuncahyo Setyo Nugroho, Anantha Yullian Sukmadewa, Novanto Yudistira, Department of Informatics Engineering, Faculty of Computer Science, Brawijaya University, Indonesia, “Large-Scale News Classification using BERT Language Model: Spark NLP approach” Available:
<https://drive.google.com/file/d/1KfKItdEHBU3wOWOx1AxxVYNczCawZ/view?usp=sharing>
- [6] Md. Raisul Kibria, Mohammad Abu Yousuf, Jahangirnagar University, “Contextdriven Bengali Text Generation using Conditional Language Model”, July 18,

2021 Available:

<https://drive.google.com/file/d/1Vh1jIQOlf7fj-0s7oZfio8-oOaOwXvE/view?usp=sharing>