**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS**

**BACHELOR OF ENGINEERING IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**MEDBOT: CHATBOT ABOUT**

**HEALTHCARE**

**Student: Nguyen Trung Tam**

**Student ID: B1910697**

**Class: 2019-2023 (K45)**

**Advisor: Dr. Lam Nhut Khang**

**Can Tho, 12/2023**

**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS**

**BACHELOR OF ENGINEERING IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**MEDBOT: CHATBOT ABOUT**

**HEALTHCARE**

**Student: Nguyen Trung Tam**

**Student ID: B1910697**

**Class: 2019-2023 (K45)**

**Advisor: Dr. Lam Nhut Khang**

**Can Tho, 12/2023**

EVALUATION OF ADVISOR

....................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

Advisor

Lam Nhut Khang

ACKNOWLEDGMENTS

I wish to express my deep gratitude and sincere thanks to my professor Lam Nhut Khang– A lecturer at the College of Information and Communication Technology who gave me the golden opportunity to do this wonderful thesis on the topic “MedBot: Chatbot about healthcare”, which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to them. Then I would like to thank the lecturers of Can Tho University, specifically, the lecturers of the College of Information and Communication Technology who taught me invaluable knowledge when I studied.

I am extremely grateful to my family for their love, prayers, and care for the completion of this thesis. I am very much thankful to my friends for their support when I was doing research at Can Tho University.

Sincerely,

Can Tho, 01/12/2023

Nguyen Trung Tam

TABLE OF CONTENTS

[CHAPTER 1: INTRODUCTION 1](#_Toc144840280)

[1.1. Problem 1](#_Toc144840281)

[1.2. History of problem-solving 2](#_Toc144840282)

[1.3. Purpose 3](#_Toc144840283)

[1.4. The objectives and scope 3](#_Toc144840284)

[1.5. Research Methods: 3](#_Toc144840285)

[1.6. Result 4](#_Toc144840286)

[CHAPTER 2: PROBLEM DESCRIPTION 5](#_Toc144840287)

[2.1. Detailed description of the problem 5](#_Toc144840288)

[2.2. Background 5](#_Toc144840289)

[2.2.1. Python 5](#_Toc144840290)

[2.2.2. Tensorflow 7](#_Toc144840291)

[2.2.3. Deep learning 8](#_Toc144840292)

[2.2.4. BART model 8](#_Toc144840293)

[2.2.5. Dataset 12](#_Toc144840294)

[CHAPTER 3: DESIGN AND IMPLEMENT OF THE CHATBOT 14](#_Toc144840295)

[3.1. Overview 14](#_Toc144840296)

[CHAPTER 4: CONCLUSION AND FUTURE WORK 15](#_Toc144840297)

[4.1. Conclusion 15](#_Toc144840298)

[4.2. Future work 15](#_Toc144840299)

[REFERENCES 16](#_Toc144840300)

[APPENDICES 17](#_Toc144840301)

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

**ABSTRACT**

In recent years, the integration of artificial intelligence (AI) and natural language processing (NLP) technologies has ushered in a new era in healthcare with the emergence of medical chatbots. These intelligent conversational agents hold immense promise in transforming the way healthcare services are delivered, improving patient experiences, and optimizing healthcare workflows.

This abstract provides an overview of the evolving landscape of medical chatbots, their key functionalities, and their impact on the healthcare ecosystem. Medical chatbots are designed to engage in dynamic conversations with users, offering personalized health information, symptom assessment, medication reminders, and appointment scheduling. They empower patients with on-demand access to medical guidance, reducing the burden on healthcare professionals and enhancing overall patient engagement.

Furthermore, medical chatbots demonstrate significant potential in improving healthcare outcomes through early symptom detection and continuous monitoring. By analyzing user-provided information and historical health data, these chatbots can identify potential health risks and provide timely recommendations, ultimately contributing to preventive care.

However, the widespread adoption of medical chatbots does not come without challenges. Ensuring data security, privacy compliance, and maintaining a high standard of accuracy and reliability in medical advice are paramount concerns. The need for seamless integration with electronic health records (EHRs) and healthcare information systems is another technical hurdle that must be addressed.

In conclusion, medical chatbots represent a transformative technology in the healthcare domain, offering a myriad of benefits such as improved accessibility, enhanced patient engagement, and streamlined administrative processes. While challenges persist, their potential to revolutionize healthcare delivery and empower individuals to take control of their health cannot be overlooked. As research and development in AI and NLP continue to advance, the future of medical chatbots holds great promise in reshaping the landscape of healthcare for the better.

**OVERVIEW**

Medical Chatbot – a chatbot application for answering questions about medical health using Python and ML

Nowadays, the necessity for health care is becoming more and more important, especially in a technology world, because of the rapid development of IT and the need for fast and accurate information about healthcare. However, there are a lot of questions and problems that traditional healthcare is limited in providing fast and accurate information. The old manual approach needs humans to search for the info and then prepare the answer for the customers.

Medical chatbot was created to provide a useful tool to assist those who want to find and answer questions about medical health care in the most effective, fast, and convenient way. In addition, the chatbot also supports collecting and updating practical healthcare information from doctors to update the chatbot accordingly. Besides, the chatbot also allows users to search for medical information according to their desire to help them find answers and advice quickly.

# INTRODUCTION

## Problem

Nowadays, healthcare is becoming more and more important, especially in a fast-paced technology world where everyone is so busy with their phone and digital devices. Therefore, the necessity to have a medical chatbot to help people have fast and reliable information about healthcare is considerably important. However, the development of medical chatbots has faced several challenges and problems in the healthcare industry, which modern chatbots aim to address. Here are some key problems that have spurred the development of medical chatbots:

1. Healthcare Accessibility: Many individuals face challenges in accessing healthcare services, particularly in remote or underserved areas. Medical chatbots can provide instant, round-the-clock access to basic medical information and advice, bridging the gap in healthcare accessibility.

2. Appointment Scheduling: Booking appointments with healthcare providers can be cumbersome and time-consuming. Chatbots can streamline the appointment scheduling process, making it more convenient for patients and reducing the administrative burden on healthcare facilities.

3. Information Overload: The internet is filled with vast amounts of health information, making it difficult for individuals to find reliable sources. Medical chatbots offer a trusted source of medical information, helping users sift through the noise to find accurate answers to their health-related questions.

4. Health Awareness and Education: Many people lack access to comprehensive health education. Medical chatbots can serve as educational tools, providing users with valuable information about various health topics and promoting health awareness.

While medical chatbots offer innovative solutions to these problems, it's important to recognize that they are not a replacement for professional medical care. They should complement, not substitute, the expertise of healthcare providers. Additionally, ensuring the accuracy of medical information, maintaining data privacy, and addressing ethical concerns remain ongoing challenges in the development and deployment of medical chatbots.

## History of problem-solving

Related chatbots:

Woebot: The conversational agent was built using Decision Tree and appropriate NLP algorithms and needs to be installed as software in a stand-alone computer. All conversational responses are stored in a database. The specific area of application is Cognitive Behavioural Therapy (CBT) for anxiety and mood disorders targeting young adults who are college students. The data for the training chatbot are collected using surveys. The data processing and analysis are carried out by qualitative analysis of the responses from the surveyors. There were several categories on which Woebot could provide therapy. Results showed that there was a significant improvement in the mean PHQ-9 score compared to controls in the mental condition of participants with 2 weeks of therapy

iHelpr: iHelpr is a text-based interactive Chatbot intended to provide mental health support in the workplace. The iHelpr is a web-based self-assessment tool and is reported to be available for six well-being indicators viz. stress, anxiety, depression, sleep, and self-esteem**.** The Chatbot is developed using the bot development framework by Microsoft’s Cognitive Services, “an Application Programming Interface (API) that can process natural language, enable a Chatbot to recognize speech, and image-processing technology”

Tess: Tess is a web-based chatbot devised by X2AI Inc. with an access interface via SMS (on mobile) and Facebook Messenger application.The chatbot is developed based on machine learning algorithms integrated with psycho-educational concepts and is said to be developed in conjunction and collaboration with trained mental health professionals.

SISU: an I ntervention for Self-help to U plift psychological wellbeing chatbot, providing support for therapeutic writing.

## Purpose

The application "MedicalBot: Chatbot about healthcare" is used for users around the world who have healthcare problems in particular and users across the world who want to search for information about healthcare and advice for avoiding health issues. Practical healthcare news, advice, and answers are provided to users who need fast and reliable information about health issues.

## The objectives and scope

The application provides main functions to help users find, search, and explore information related to healthcare problems and advice. This thesis will be focused on the problem of searching and providing reliable answers to the medical industry. The scope of the study is: researching the problem of searching and finding health-related information online and solving the problem by implementing a chatbot with friendly UI/UX and handy features.

## Research Methods:

- Requirements analysis: study the problem related to chatbots, especially in the medical field on the network, research papers, then analyze the function, and describe the requirements to build and train the chatbot.

- Data collection: Collect questions, answers, and relevant information about the medical problem to train the chatbot.

- Design: UI/UX design; Model analysis and design model architecture.

- Implementation: Using Python, and Google Colab for training the chatbot and then deploying the chatbot to Facebook messager with Python, and Flask using REST API.

## Result

- Using acquired knowledge of analysis, research, and information gathering, build an application with medical chatbot Python, and Machine Learning.

Thesis outline

**Chapter 1**: Introduction

**Chapter 2**: Background

General information about the study and main functions of the system.

**Chapter 3**: Design and implementation

Introduction of UI/UX designs, models, and implementation, describing

technologies that will be used in the study.

**Chapter 4**: Conclusion and future work

# PROBLEM DESCRIPTION

## Detailed description of the problem

Medical chatbot provides users with functions such as answering questions related to healthcare and searching for advice about health issues in an easy, fast, and reliable way. In addition, the chatbot feature is integrated into the website to help students get answers related to actual practice accurately and quickly.

Medical chatbot application consists of 2 main parts: the chatbot and the UI application.

The application part helps users interact with the chatbot and supports collecting questions and answers to improve the effectiveness of the training chatbot. In addition, the application section also supports users to find questions, answers, and news related to practical healthcare problems. The application section will also support an administrator function to maintain and update the chatbot to improve its accuracy of the model over time.

## Background

### Python

Python is a high-level, versatile programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, Python has gained widespread popularity in various domains, including web development, data science, artificial intelligence, scientific computing, and more.

Python is one of the most popular programming languages for machine learning and artificial intelligence. Its versatility, extensive libraries, and active community support make it an ideal choice for developing machine-learning models and conducting data analysis. Here's an overview of how Python is used in the field of machine learning:

1. Libraries and Frameworks: Python has a rich ecosystem of libraries and frameworks that simplify machine learning tasks. Some of the most commonly used ones include:

- numpy: numpy is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays and mathematical functions, making it essential for data manipulation.

- pandas: pandas is a data manipulation library that provides data structures like DataFrames and Series. It is used for data cleaning, transformation, and analysis.

- scikit-learn: scikit-learn is a popular machine learning library that offers a wide range of machine learning algorithms for classification, regression, clustering, dimensionality reduction, and more. It also provides tools for model selection and evaluation.

- TensorFlow: Developed by Google, TensorFlow is an open-source machine learning framework that's widely used for deep learning tasks. It allows you to build neural networks for various applications, including image recognition, natural language processing, and reinforcement learning.

- PyTorch: PyTorch is another deep learning framework that has gained popularity for its flexibility and dynamic computation graph. It is known for its ease of use and is commonly used in research and development of neural network models.

- Keras: Keras is a high-level neural networks API that runs on top of other deep learning frameworks like TensorFlow and Theano. It simplifies the process of building and training neural networks.

2. Data Preprocessing: Python libraries like NumPy and pandas are instrumental for data preprocessing. You can clean and prepare your data, handle missing values, perform feature engineering, and create datasets suitable for machine learning tasks.

3. Visualization: Libraries like Matplotlib and Seaborn allow you to visualize your data, which is crucial for understanding patterns and relationships in your datasets. Visualization aids in data exploration and model evaluation.

4. Model Building and Training: Python's machine learning libraries provide a straightforward way to build, train, and evaluate machine learning models. You can experiment with various algorithms and techniques to find the best model for your specific problem.

5. Model Evaluation: Scikit-learn offers tools for evaluating machine learning models using metrics like accuracy, precision, recall, F1-score, and ROC curves. Cross-validation techniques help assess a model's generalization performance.

6. Deployment: After training a machine learning model, you can deploy it in a production environment. Python allows you to create web services, and RESTful APIs, or integrate models into applications using frameworks like Flask or Django.

7. Community and Resources: Python's machine learning community is vast, with a wealth of tutorials, documentation, and online courses available. Platforms like Kaggle provide datasets and competitions for practicing and honing your machine-learning skills.

8. Research: Python is commonly used for machine learning research, thanks to the availability of powerful libraries like TensorFlow and PyTorch. Researchers can experiment with cutting-edge techniques and algorithms.

Python's role in machine learning continues to evolve, and it remains at the forefront of AI and data science. Whether you're a beginner looking to get started or an experienced practitioner, Python's ecosystem and community support make it a valuable tool for tackling a wide range of machine-learning challenges.

### Tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

### Deep learning

Deep learning is the branch of machine learning which is based on artificial neural network architecture. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network. The layers of the neural network transform the input data through a series of nonlinear transformations, allowing the network to learn complex representations of the input data.

Deep learning can be used for supervised, unsupervised as well as reinforcement machine learning. It uses a variety of ways to process these.

### Chatbot

A chatbot (conversational agent (CA), dialogue system) is a computer software that acts as an interface between human users and a software application, using spoken or written natural language as the primary means of communication.

Chatbot components

The functioning of a chatbot involves several key components:

Natural Language Processing (NLP): Natural language understanding (NLU) is the first core component of the conversational agents which is responding by providing a semantic representation for user utterance [6] such as in the form of logic or class’s intent, extracting the “meaning” of an utterance. NLP is the technology that enables chatbots to understand and interpret human language.

Dialog Management: Dialogue Manager is The second core component in any chatbot and we can differentiate the chatbots through this component which has many parts that can be improved or adding some parts in the future if will be discovered that it will serve the DM.

Knowledge Base or Backend Integration: Many chatbots are connected to databases, APIs, or other systems to access information or perform actions. For example, a customer support chatbot might access a database of FAQs or connect to a CRM system to look up customer information.

User Interface: This is the medium through which users interact with the chatbot. It could be a chat window on a website, a messaging app like WhatsApp, or a voice-activated device like Amazon Echo.

Machine Learning (Optional): Some advanced chatbots incorporate machine learning algorithms. These bots can learn from user interactions to improve their responses over time. They become more effective as they gather more data and refine their understanding of user queries.

Types of chatbot

There are different types of chatbots based on their capabilities and functionalities:

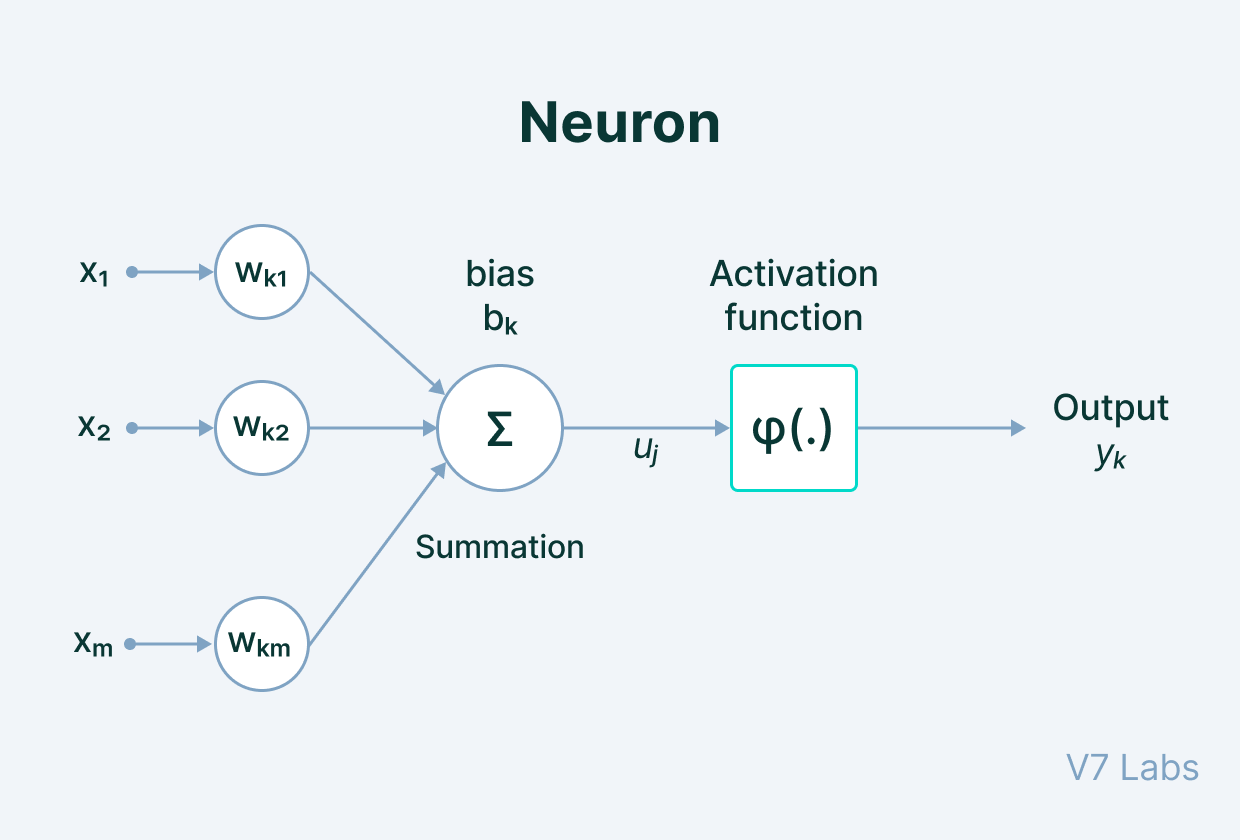
Rule-Based Chatbots: These chatbots follow predefined rules and patterns. They provide responses based on keywords and phrases. Rule-based chatbots are typically used for straightforward and specific tasks.

AI-Powered Chatbots: These chatbots leverage artificial intelligence and machine learning to provide more dynamic and context-aware responses. They can handle more complex and open-ended conversations.

Voice Assistants: Voice-activated chatbots, like Siri, Google Assistant, or Alexa, are designed to respond to spoken language inputs. They use speech recognition technology to understand and fulfill user requests.

### Neuron network

The inspiration for a Neural Network (NN) originates from the human brain, where biological neurons (nerve cells) respond to the activation of other neurons they are connected to. At a very simple level, neurons in the brain take electrical inputs that are then channeled to outputs. The sensitivity of this relation also depends on the strength of the connection, i.e. a neuron may be more responsive to one neuron, than to another.



Input - It is the set of features that are fed into the model for the learning process. For example, the input in object detection can be an array of pixel values pertaining to an image.

Weight - Its main function is to give importance to those features that contribute more towards the learning. It does so by introducing scalar multiplication between the input value and the weight matrix. For example, a negative word would impact the decision of the sentiment analysis model more than a pair of neutral words.

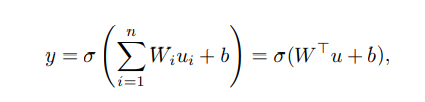
Transfer function - The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. It is done by a simple summation of all the inputs to the transfer function.

Activation Function—It introduces non-linearity in the working of perceptrons to consider varying linearity with the inputs. Without this, the output would just be a linear combination of input values and would not be able to introduce non-linearity in the network.

Bias - The role of bias is to shift the value produced by the activation function. Its role is similar to the role of a constant in a linear function.

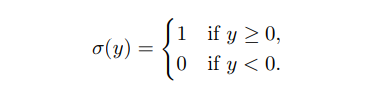
When multiple neurons are stacked together in a row, they constitute a layer, and multiple layers piled next to each other are called a multi-layer neural network.

For a single neuron/node with input u ∈ R n, a mathematical model, named the perceptron [27], can be described as

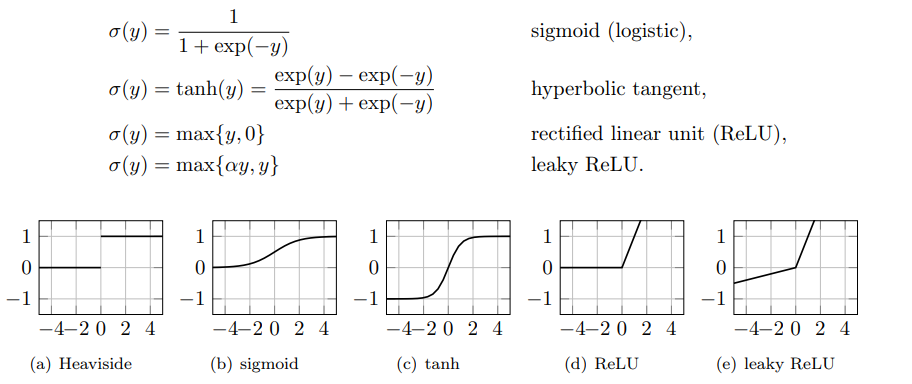


where y is the activation of the neuron/node, Wi are the weights and b is the bias.

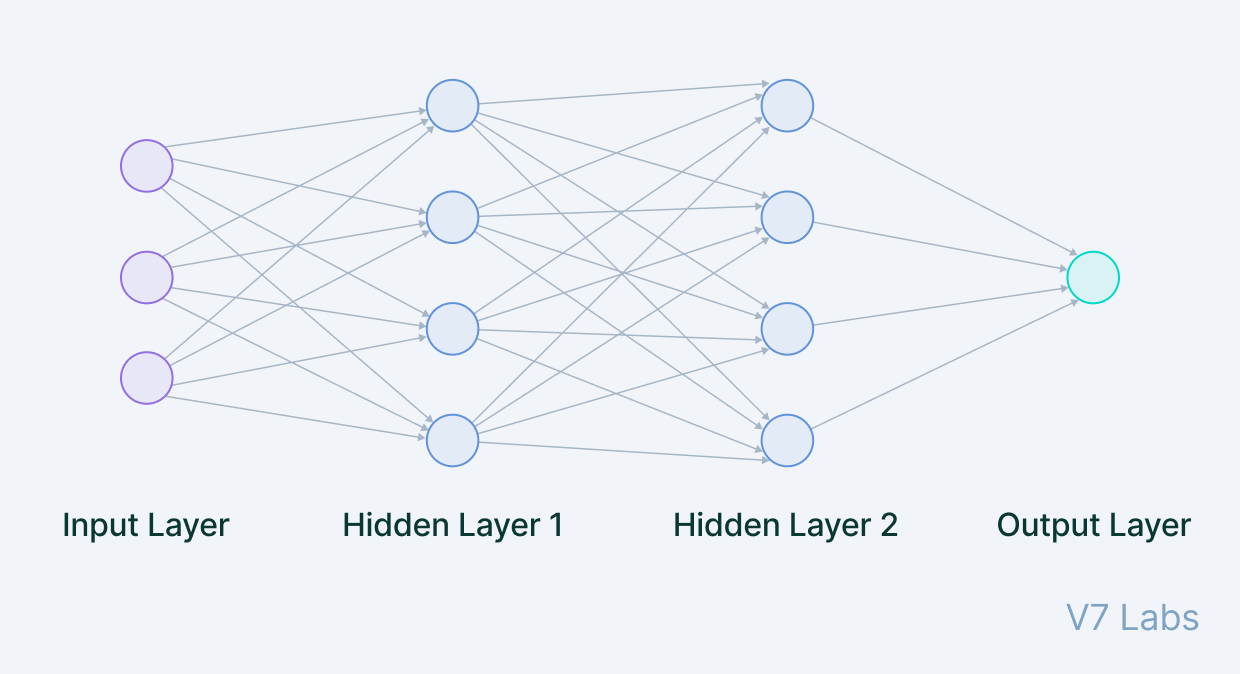
The function σ : R → R is called activation function. Originally, in [27], it was proposed to choose the Heaviside function as activation function to model whether a neuron fires or not



Popular activation functions are



ANNs consist of interconnected artificial neurons or nodes organized in layers, each layer serving a specific purpose in the network. Here's an overview of the architecture of artificial neural networks:



1. Input Layer:

The input layer is the first layer of the neural network.

Neurons in this layer receive input data, which could be features from a dataset or raw sensor data, depending on the application.

The number of neurons in the input layer is determined by the dimensionality of the input data.

2. Hidden Layers:

Between the input and output layers, there can be one or more hidden layers.

Each hidden layer consists of multiple neurons.

Neurons in the hidden layers process the input data through weighted connections and activation functions.

The number of hidden layers and neurons in each layer is a hyperparameter that can be adjusted to optimize network performance.

3. Output Layer

The output layer takes input from preceding hidden layers and comes to a final prediction based on the model’s learnings. It is the most important layer where we get the final result.

In the case of classification/regression models, the output layer generally has a single node. However, it is completely problem-specific and dependent on the way the model was built.

3. Weights and Connections:

Each connection between neurons has an associated weight.

Weights are learned during the training process and represent the strength of the connection between neurons.

Adjusting these weights is the primary mechanism by which the network learns and makes predictions.

4. Activation Functions:

Neurons in the network apply activation functions to the weighted sum of their inputs.

Activation functions introduce non-linearity into the model, enabling the network to learn complex patterns.

Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.

5. Output Layer:

The output layer produces the final prediction or output of the neural network.

The number of neurons in the output layer depends on the type of task the network is designed for. For example:

In binary classification, there might be one output neuron with a sigmoid activation function.

In multi-class classification, there would be as many output neurons as there are classes, often with softmax activation.

In regression tasks, there might be a single output neuron without an activation function.

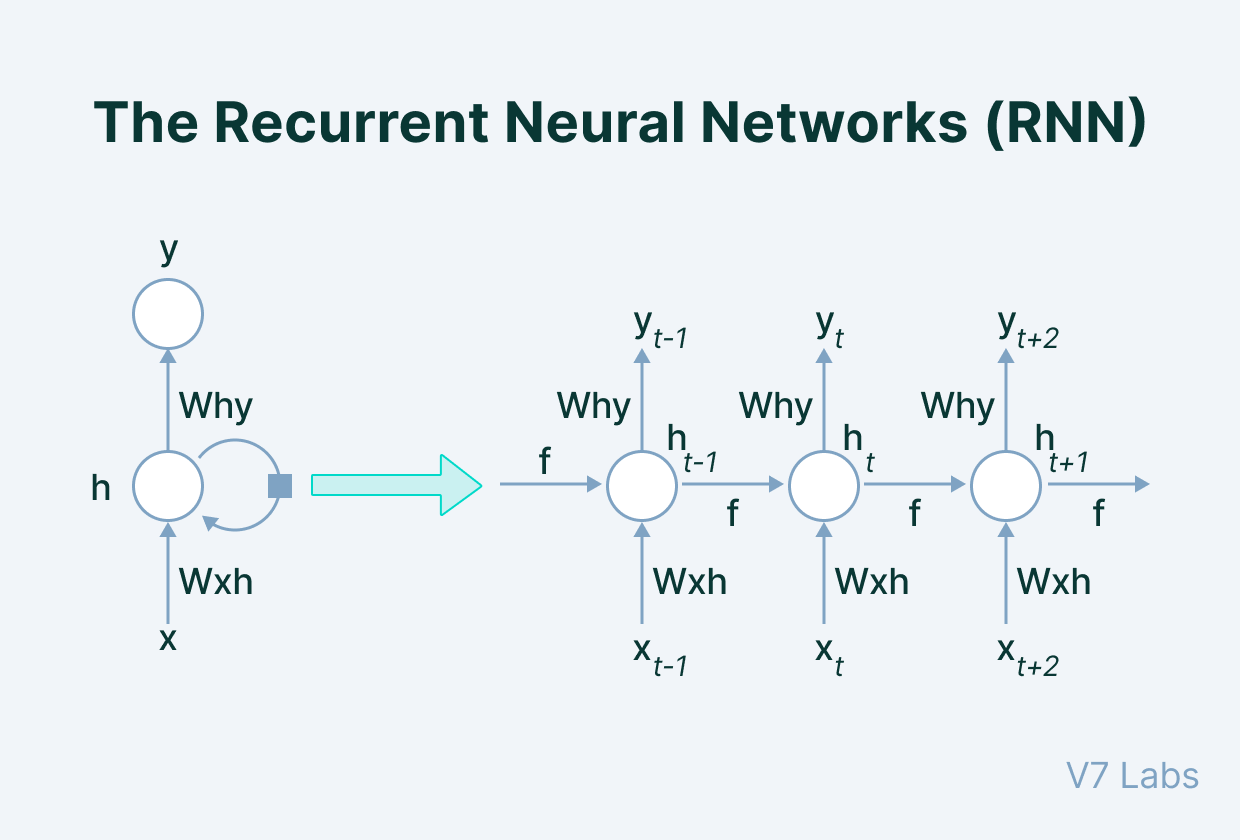
Recurrent Neural Networks (RNNs)

The basic deep learning architecture has a fixed input size, and this acts as a blocker in scenarios where the input size is not fixed. Also, the decisions made by the model were based on the current input with no memory of the past.

Recurrent Neural Networks work very well with sequences of data as input. Its functionality can be seen in solving NLP problems like sentiment analysis, spam filters, time series problems like sales forecasting, stock market prediction, etc.

The Recurrent Neural Networks (RNN)

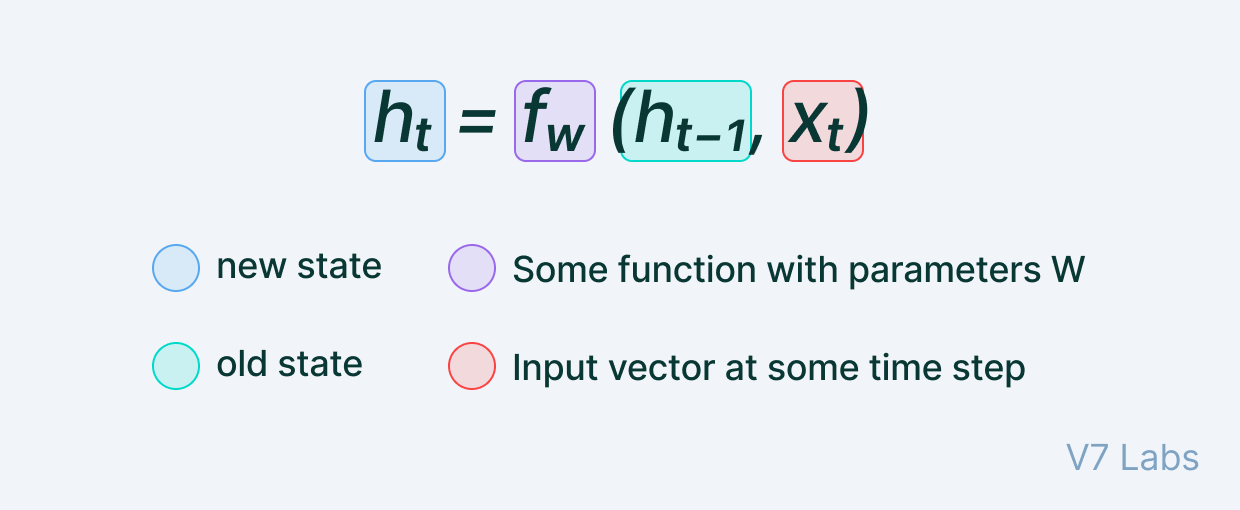
Recurrent Neural Networks have the power to remember what it has learned in the past and apply it in future predictions.



The input is in the form of sequential data that is fed into the RNN, which has a hidden internal state that gets updated every time it reads the following sequence of data in the input.

The internal hidden state will be fed back to the model. The RNN produces some output at every timestamp.

The mathematical representation is given below:

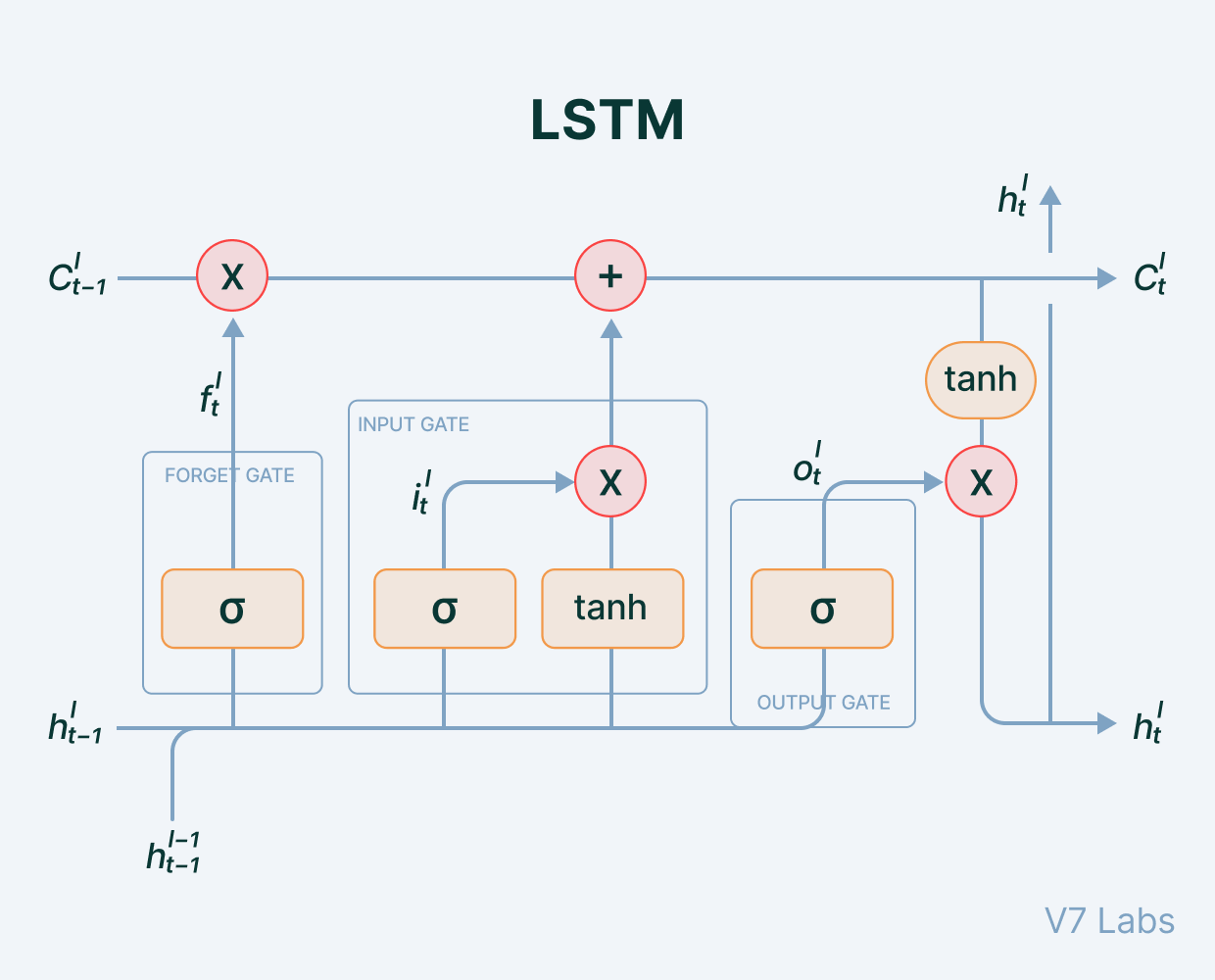


The Long Short Term Memory Network (LSTM)

In RNN each of our predictions looked only one timestamp back, and it has a very short-term memory. It doesn't use any information from further back.

To rectify this, we can take our Recurrent Neural Networks structure and expand it by adding some more pieces to it.

The critical part that we add to this Recurrent Neural Networks is memory. We want it to be able to remember what happened many timestamps ago. To achieve this, we need to add extra structures called gates to the artificial neural network structure.



Cell state (c\_t): It corresponds to the long-term memory content of the network.

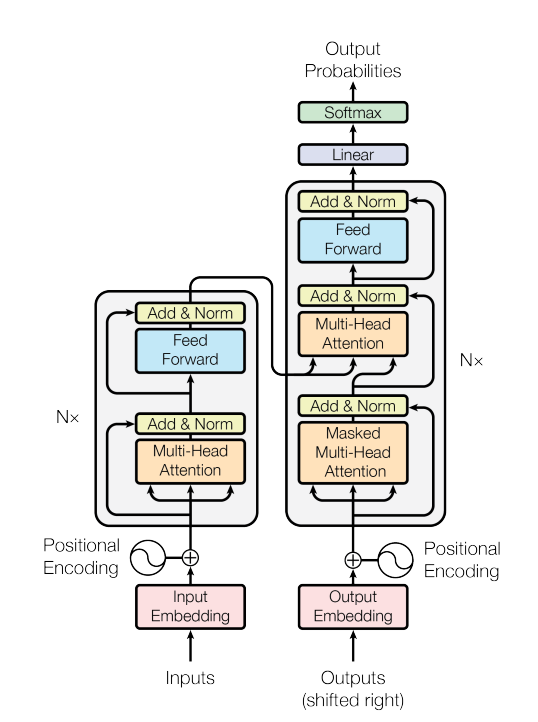
Forget Gate: Some information in the cell state is no longer needed and is erased. The gate receives two inputs, x\_t (current timestamp input) and h\_t-1 (previous cell state), multiplied with the relevant weight matrices before bias is added. The result is sent into an activation function, which outputs a binary value that decides whether the information is retained or forgotten.

Input gate: It decides what piece of new information is to be added to the cell state. It is similar to the forget gate using the current timestamp input and previous cell state with the only difference of multiplying with a different set of weights.

Output gate: The output gate's job is to extract meaningful information from the current cell state and provide it as an output.

### Transformer

The encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next



Encoder: The encoder is composed of a stack of N identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension dmodel = 512.

Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension dmodel as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

P E(pos,2i) = sin(pos/100002i/dmodel)

P E(pos,2i+1) = cos(pos/100002i/dmodel)

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000 · 2π. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, P Epos+k can be represented as a linear function of P Epos.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output,

where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key

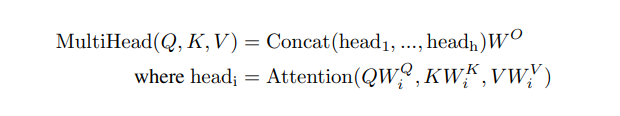
Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension dk, and values of dimension dv. We compute the dot products of the query with all keys, divide each by √ dk, and apply a softmax function to obtain the weights on the values. In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as: Attention(Q, K, V ) = softmax(QKT √ dk )V (1) The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of √ 1 dk . Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. While for small values of dk the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of dk [3]. We suspect that for large values of dk, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients 4 . To counteract this effect, we scale the dot products by √ 1 dk .

Multi-Head Attention

Instead of performing a single attention function with dmodel-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding dv-dimensional

output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this



Where the projections are parameter matrices W Q i ∈ R dmodel×dk , W K i ∈ R dmodel×dk , WV i ∈ R dmodel×dv and WO ∈ R hdv×dmodel . In this work we employ h = 8 parallel attention layers, or heads. For each of these we use dk = dv = dmodel/h = 64. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality

### Tokenizer

Tokenizing a text is splitting it into words or subwords, which then are converted to ids through a look-up table. Converting words or subwords to ids is straightforward, so we will focus on splitting a text into words or subwords (i.e. tokenizing a text)

Subword-based tokenization

Subword-based tokenization is a solution between word and character-based tokenization. 😎 The main idea is to solve the issues faced by word-based tokenization (very large vocabulary size, large number of OOV tokens, and different meanings of very similar words) and character-based tokenization (very long sequences and less meaningful individual tokens).

The subword-based tokenization algorithms do not split the frequently used words into smaller subwords. It rather splits the rare words into smaller meaningful subwords. For example, “boy” is not split but “boys” is split into “boy” and “s”. This helps the model learn that the word “boys” is formed using the word “boy” with slightly different meanings but the same root word.

Some of the popular subword tokenization algorithms are WordPiece, Byte-Pair Encoding (BPE), Unigram, and SentencePiece. We will go through Byte-Pair Encoding (BPE) in this article. BPE is used in language models like GPT-2, RoBERTa, XLM, FlauBERT, etc. A few of these models use space tokenization as the pre-tokenization method while a few use more advanced pre-tokenization methods provided by Moses, spaCY, ftfy. So, let’s get started. 🏃

Byte-Pair Encoding (BPE)

BPE is a simple form of data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data. BPE relies on a pre-tokenizer that splits the training data into words. Pretokenization can be as simple as space tokenization.

After pre-tokenization, a set of unique words has been created and the frequency with which each word occurred in the training data has been determined. Next, BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns to merge rules to form a new symbol from two symbols of the base vocabulary. It does so until the vocabulary has attained the desired vocabulary size. Note that the desired vocabulary size is a hyperparameter to define before training the tokenizer.

Encoding and Decoding

Let us now see how we will decode our example. To decode, we have to simply concatenate all the tokens together to get the whole word. For example, the encoded sequence [“the</w>”, “high”, “est</w>”, “range</w>”, “in</w>”, “Seattle</w>”], we will be decoded as [“the”, “highest”, “range”, “in”, “Seattle”] and not as [“the”, “high”, “estrange”, “in”, “Seattle”]. Notice the presence of the “</w>” token in “est”.

For encoding the new data, the process is again simple. However, encoding in itself is computationally expensive. Suppose the sequence of words is [“the</w>”, “highest</w>”, “range</w>”, “in</w>”, “Seattle</w>”]. We will iterate through all the tokens we found in our corpus — longest to the shortest and try to replace substrings in our given sequence of words using these tokens. Eventually, we will iterate through all the tokens and our substrings will be replaced with tokens already present in our token list. If a few substrings are left (for words our model did not see in training), we will replace them with unknown tokens.

In general, the vocabulary size is big but still, there is a possibility of an unknown word. In practice, we save the pre-tokenized words in a dictionary. For unknown (new) words, we apply the above-stated encoding method to tokenize the new word and add the tokenization of the new word to our dictionary for future reference. This helps us build our vocabulary even stronger for the future.

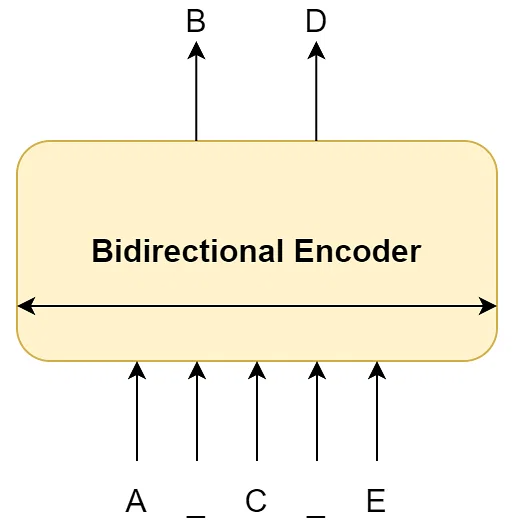
### BART model

BART is a denoising autoencoder that maps a corrupted document to the original document it was derived from. It is implemented as a sequence-to-sequence model with a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder. For pre-training, we optimize the negative log-likelihood of the original document.

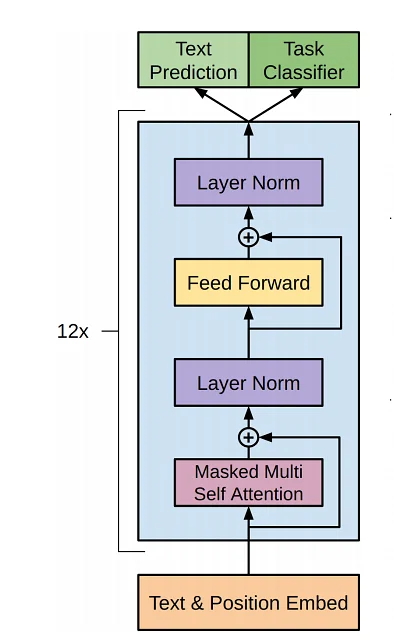
BART uses the standard sequence-to-sequence Transformer architecture from the original Transformer of Google, except, following GPT, BART modifies ReLU activation functions to GeLUs and initializes parameters from N (0, 0.02). For the base model, BART uses 6 layers in the encoder and decoder, and for the large model, BART uses 12 layers in each. The architecture is closely related to that used in BERT, with the following differences:

* Each layer of the decoder additionally performs cross-attention over the final hidden layer of the encoder (as in the transformer sequence-to-sequence model)
* BERT uses an additional feed-forward network before word prediction, which BART does not. In total, BART contains roughly 10% more parameters than the equivalently sized BERT model.

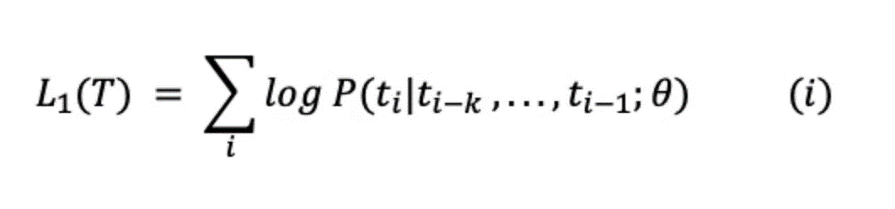
For every text sequence in its input, the BERT encoder outputs an embedding vector for each token in the sequence as well as an additional vector containing sentence-level information. In this way, the decoder can learn for both token and sentence-level tasks making it a robust starting point for any future fine-tuning tasks.



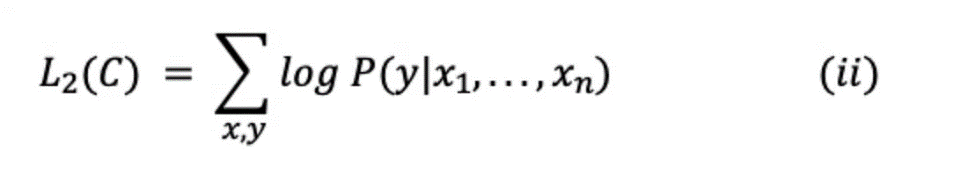
Once we get the token and sentence-level representation of an input text sequence, a decoder needs to interpret these to map with the output target. However, by using a similarly designed decoder, tasks such as next-sentence prediction or token prediction might perform poorly since the model relies on a more comprehensive input prompt. In these cases, we need model architectures that can be trained to generate the next word by only looking at the previous words in the sequence. Hence, a causal or autoregressive model that looks only at the past data to predict the future comes in handy.



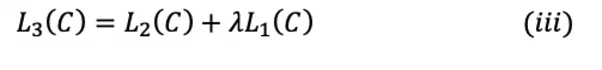
First, the model is pre-trained on tokens “t” looking back to “k” tokens in the past to compute the current token. This is done unsupervised on a vast text corpus to allow the model to “learn the language.”



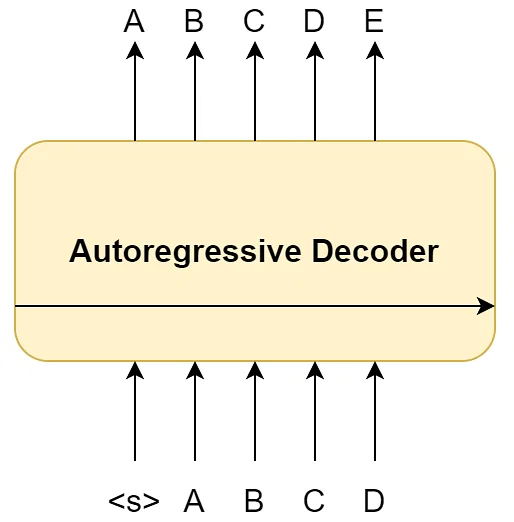
Next, to make the model robust on a specific task, it is fine-tuned in a supervised manner to maximize the likelihood of label “y” given feature vectors x1…xn.



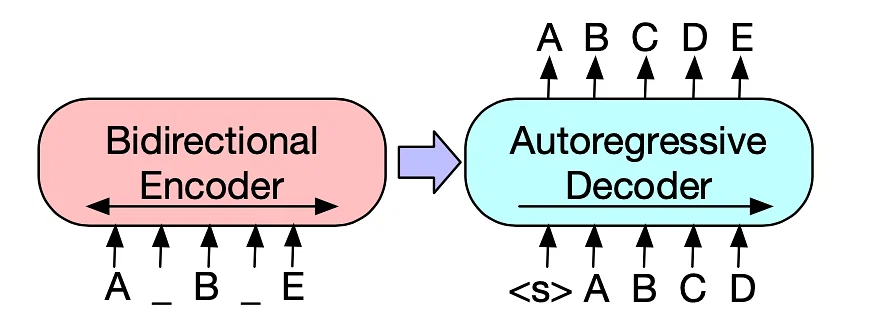
Combining 1 and 2, we get the objective in 3. Lambda represents a learned weight parameter to control the influence of language modeling.



The below image shows how the autoregressive decoder processes its input.



Although we separate the decoder from an encoder, the input to the decoder would still be a learned representation (or embedding) of the original text sequence. Thus, BART attaches the bi-directional encoder to the autoregressive decoder to create a denoising auto-encoder architecture. Based on these two components, the final BART model would look something like this:



In the above figure, the input sequence is a masked (or noisy) version of [ABCDE] transformed into [A(MASK)B(MASK)E]. The encoder looks at the entire sequence and learns high-dimensional representations with bi-directional information. The decoder takes these thought vectors and regressively predicts the next token. Learning occurs by computing and optimizing the negative log-likelihood as mapped with the target [ABCDE].

### Rough metric

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics and a software package specifically designed for evaluating summary, but that can be also used for machine translation as well as text generation. The metrics compare an automatically produced summary or translation against reference (high-quality and human-produced) summaries or translations.

ROUGE-N

ROUGE-N measures the number of matching between the model-generated text and a human-produced reference.

Consider the reference R and the candidate summary C:

- R: The cat is on the mat.

- C: The cat and the dog.

ROUGE-1

Using R and C, we are going to compute the precision, recall, and F1 scores of the matching n-grams. Let’s start computing ROUGE-1 by considering 1-grams only.

ROUGE-1 precision can be computed as the ratio of the number of unigrams in C that appear also in R (that are the words “the”, “cat”, and “the”), over the number of unigrams in C.

ROUGE-1 precision = 3/5 = 0.6

ROUGE-1 recall can be computed as the ratio of the number of unigrams in \_R\_ that appear also in C (that are the words “the”, “cat”, and “the”), over the number of unigrams in R.

ROUGE-1 recall = 3/6 = 0.5

Then, the ROUGE-1 F1-score can be directly obtained from the ROUGE-1 precision and recall using the standard F1-score formula.

ROUGE-1 F1-score = 2 \* (precision \* recall) / (precision + recall) = 0.54

ROUGE-2

Let’s try computing the ROUGE-2 considering 2-grams.

Remember our reference R and candidate summary C:

- R: The cat is on the mat.

- C: The cat and the dog.

ROUGE-2 precision is the ratio of the number of 2-grams in C that appear also in R (only the 2-gram “the cat”), over the number of 2-grams in C.

ROUGE-2 precision = 1/4 = 0.25

ROUGE-2 recall is the ratio of the number of 2-grams in R that appear also in C (only the 2-gram “the cat”), over the number of 2-grams in R.

ROUGE-2 recall = 1/5 = 0.20

Therefore, the F1-score is:

ROUGE-2 F1-score = 2 \* (precision \* recall) / (precision + recall) = 0.22

ROUGE-L

ROUGE-L is based on the longest common subsequence between our model output and reference, i.e. the longest sequence of words (not necessarily consecutive, but still in order) that is shared between both. A longer shared sequence should indicate more similarity between the two sequences.

We can compute ROUGE-L recall, precision, and F1-score just like we did with ROUGE-N, but this time we replace each n-gram match with the LCS.

Remember our reference R and candidate summary C:

- R: The cat is on the mat.

- C: The cat and the dog.

The LCS is the 3-gram “the cat the” (remember that the words are not necessarily consecutive), which appears in both R and C.

ROUGE-L precision is the ratio of the length of the LCS, over the number of unigrams in C.

ROUGE-L precision = 3/5 = 0.6

ROUGE-L precision is the ratio of the length of the LCS, over the number of unigrams in R.

ROUGE-L recall = 3/6 = 0.5

Therefore, the F1-score is:

ROUGE-L F1-score = 2 \* (precision \* recall) / (precision + recall) = 0.55

ROUGE-S

ROUGE-S allows us to add a degree of leniency to the n-gram matching performed with ROUGE-N and ROUGE-L. ROUGE-S is a skip-gram concurrence metric: this allows to search for consecutive words from the reference text that appear in the model output but are separated by one-or-more other words.

Consider the new reference R and candidate summary C:

- R: The cat is on the mat.

- C: The gray cat and the dog.

If we consider the 2-gram “the cat”, the ROUGE-2 metric would match it only if it appears in C exactly, but this is not the case since C contains “the gray cat”. However, using ROUGE-S with unigram skipping, “the cat” would match “the gray cat” too.

We can compute ROUGE-S precision, recall, and F1-score in the same way as the other ROUGE metrics.

### BLEU

BLEU, or the Bilingual Evaluation Understudy, is a metric for comparing a candidate translation to one or more reference translations.

Although developed for translation, it can be used to evaluate text generated for different natural language processing tasks, such as paraphrasing and text summarization.

The BLEU score is not perfect, but it’s quick and inexpensive to calculate, language-independent, and, above all, correlates highly with human evaluation.

How to compute the BLEU score

Consider the two reference answer R1 and R2 produced by human experts, and the candidate answer C1 produced by our chatbot system.

- R1: The cat is on the mat.

- R2: There is a cat on the mat.

- C1: The cat and the dog.

Computing unigrams precision

To express the quality of our translation with a metric, we may count how many words in the candidate translation C1 are present in the reference translations R1 and R2, and divide the result by the number of words in C1 to get a percentage. Therefore, a perfect score is 1.0, whereas the worst score is 0.0. Let’s call this metric BLEU.

In C1 there are three words (“the”, “cat”, “the”) that appear on the reference translations, thus:

BLEU\*(C1) = 3/5 = 0.6

The candidate translation is far from perfect, indeed it receives a score of 0.6. Everything looks fine.

The problem with repeating unigrams

Let’s compute the BLEU\* score of the new candidate translation C2:

- R1: The cat is on the mat.

- R2: There is a cat on the mat.

- C2: The The The The The.

This time our chatbot system is not very good, unfortunately.

Every word in C2 is present in at least one between R1 and R2, thus:

BLEU\*(C2) = 5/5 = 1

We achieved a perfect score with a non-sense translation, there’s something we need to correct on our metric.

It doesn’t make sense to consider the word “The” five times in the numerator, as it appears at most twice on each reference translation. We can try counting the word “The” only for the times it appears at most on each reference translation, that is two. Let’s call this new metric BLEU.

BLEU\*\*(C2) = 2/5 = 0.4

Now the score makes more sense, as we are accounting for the fact that a good translated word appears too many times on our candidate translation.

Considering n-grams

Let’s try computing the BLEU score on two other candidate translations C3 and C4 to check if everything looks fine.

- R1: The cat is on the mat.

- R2: There is a cat on the mat.

- C3: There is a cat on the mat.

- C4: Mat the cat is on a there.

The BLEU scores are the following:

BLEU\*\*(C3) = 7/7 = 1.0

BLEU\*\*(C4) = 7/7 = 1.0

Both candidate translations contain words that are present in the reference translations, therefore they both achieve the maximum score. However, C4 is not a well-formed English sentence.

A quick way to get higher scores for well-formed sentences is to consider matching 2-grams or 3-grams instead of 1-grams only. Let’s call BLEU\*\*₁ the score that considers only 1-grams and BLEU\*\*₂ the score that considers only 2-grams.

C3 has six 2-grams and they all appear on the reference translation R2, thus:

BLEU\*\*₁(C3) = 7/7 = 1.0

BLEU\*\*₂(C3) = 6/6 = 1.0

Instead, in C4 all the 2-grams don’t appear in any reference translation, thus:

BLEU\*\*₁(C4) = 7/7 = 1.0

BLEU\*\*₂(C4) = 0/6 = 0.0

It is generally said that the BLEU\*\*ₙ score for n-grams focuses on the sentence meaning for low n, and focuses on well-formed sentences for high n.

It has been found that the geometric mean of the BLEU\*\*ₙ scores with n between one and four has the best correlation with human evaluation, therefore it’s the score more commonly adopted. Let’s call it MEAN\_BLEU.

Penalizing short candidate translations

Let’s try now computing the BLEU\*\*₁ and BLEU\*\*₂ scores of the candidate translation C5:

- R1: The cat is on the mat.

- R2: There is a cat on the mat.

- C5: There is a cat.

The scores are:

BLEU\*\*₁(C5) = 4/4 = 1.0

BLEU\*\*₂(C5) = 3/3= 1.0

Looks like C5 achieves a perfect BLEU\*\*ₙ score for each n, even though the candidate translation is missing a piece of text with respect to the reference translations.

This can be avoided by adding a penalty for candidate translations whose length is less than the ones of the reference translations. We call it Brevity Penalty (BP).

The final BLEU score is:

BLEU = BP \* MEAN\_BLEU

That is, BLEU is the product of the Brevity Penalty BP (which penalizes short translations that don’t contain relevant text from the reference translations) and the geometric mean of the BLEU\*\*ₙ scores for n between one and four (which takes into account small n-grams, to capture the sentence meaning, and large n-grams, to get well formed sentences).

What is the value of BP?

If the length of the candidate solution is bigger than the length of the reference translation with the most similar length, then we shouldn’t penalize and therefore BP equals one. Otherwise, BP is a decaying exponential which is lower when the length difference between the candidate and the reference translations is greater. The BLEU suggests computing the brevity penalty over the entire corpus rather than over single translations to smoothen the penalties for short translations.

# DESIGN AND IMPLEMENT OF THE CHATBOT

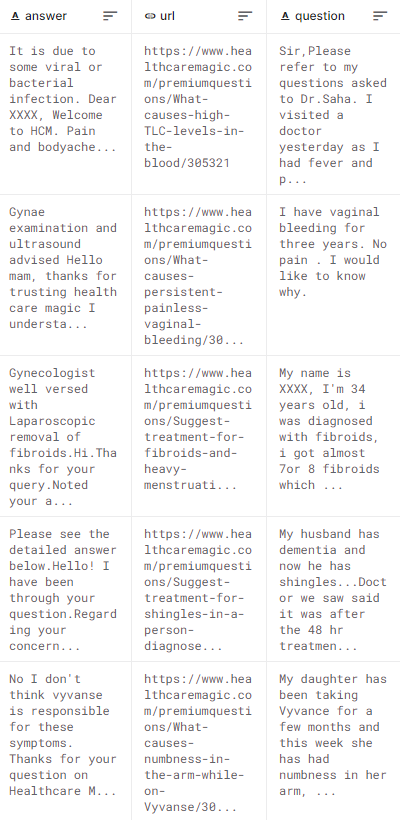
## Overview

## Implementation

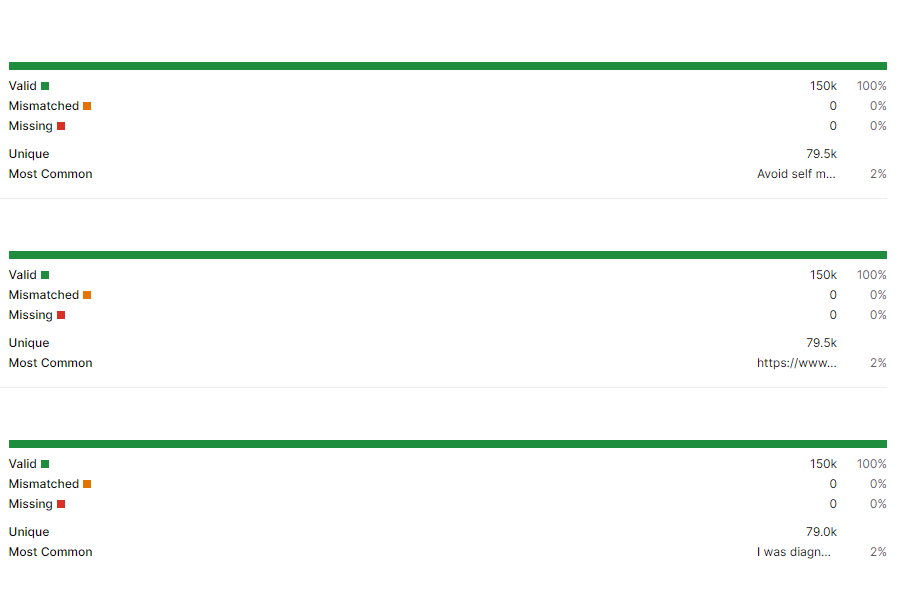
### Collect dataset

To train the bot to answer questions about healthcare, a list of collected questions and answers needs to be analyzed and processed. Based on the analysis, the final data to train the chatbot is organized and defined in train, test, and validation dataset files. Once the data is finalized, the chatbot needs to be fed all the data with the corresponding responses. Then the chatbot created needs to be tested and trained to fine-tune the results.

Below are some examples from the training dataset:



Also, details information about the dataset is shown below:



### Preprocessing

### Training

### Integrate to application

# EVALUATION

## Result

The model is trained on Google Colab (with 12GB RAM and GPU T4) as well as on Kaggle (with 12GB RAM GPU P100) for 100 epochs, each epoch takes about 3 hours to complete.

Firstly, the chatbot was trained on the Medical Chat Dataset with more than 200,000 question-answer pairs about healthcare.

Next, the chatbot continues to train on the collected datasets crawled from healthcare websites with more than 100,000 question-answer pairs.

The chatbot was initially trained with different max sequence lengths (64, 128, 256, 512) on a relatively small dataset (about 10,000 question-answer pairs) to find the optimized hyper-parameters but still have good accuracy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sequence length | 64 | 128 | 256 | 512 |
| Epochs | 100 | 100 | 100 | 100 |
| Dataset | 222,380 pairs | 222,380 pairs | 222,380 pairs | 222,380 pairs |
| Word dictionary |  |  |  |  |
| Time for 1 epoch | 45 minutes | 1 hour | 1 hour 15 mins | 2 hours |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sequence length | 64 | 128 | 256 | 512 |
| Epochs | 100 | 100 | 100 | 100 |
| Dataset | 299,757 pairs | 299,757 pairs | 299,757 pairs | 299,757 pairs |
| Word dictionary |  |  |  |  |
| Time for 1 epoch | 45 minutes | 1 hour | 1 hour 15 mins | 2 hours |

# CONCLUSION AND FUTURE WORK

## Conclusion

## Future work

# REFERENCES

# APPENDICES