St. Xavier's College

Autonomous

Kolkata, West Bengal

CHATBOT
- AN APPLICATION OF NATURAL LANGUAGE PROCESSING AND DEEP LEARNING

BY

PRASHANT GOMES, 511

SANCHITA PAUL, 549

UNDER THE GUIDANCE OF
DR. ASOKE NATH

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled .... submitted to the Department of Computer Science , ST. XAVIER'S COLLEGE [AUTONOMOUS], KOLKATA, in partial fulfilment of the requirement for the award of the degree of Master of Science (M. Sc.)/ Bachelor of Science (B.Sc.) is an original work carried out by:

NAME	**ROLL NUMBER**	**REGISTRATION NUMBER**

under my guidance. The matter embodied in this project is authentic and is genuine work done by the student and has not been submitted whether to this College or to any other Institute for the fulfilment of the requirement of any course of study.

....

Signature of the Supervisor Signature of the Head of the Department

....

Name of the Supervisor

Name and Signature of the **Project Team members:**

Name Signature

1 ....

2 ....

ABSTRACT

Retrieval-based chatbots are fundamental tools in computer-mediated conversations. They use stored responses to act as virtual assistants, facilitating smooth interactions between users and computers. This examination delves into how these chatbots work, emphasizing their crucial role in modern communication. They not only enable human-like interactions but also establish strong question-answer systems. Additionally, their ability to adapt to different contexts and user queries makes them invaluable in today's digital landscape. In a world where communication patterns are constantly evolving, retrieval-based chatbots are essential for bridging gaps in understanding and improving human-computer interaction. They excel at interpreting and responding to queries across various fields, from customer service to education. As technology advances, retrieval-based chatbots will continue to shape the future of conversational interfaces, providing dynamic platforms for interacting with machines. Their significance in digital communication is bound to increase, solidifying their status as indispensable assets in the technological landscape.

ACKNOWLEDGEMENT

We express our sincere gratitude to Dr. Asoke Nath, our esteemed project mentor, for providing invaluable guidance and direction throughout the documentation process of our project work. We extend our appreciation to the college authorities, Prof. Romit Beed, our Head of Department, the entire Computer Science department faculty of St. Xavier's College, (Autonomous), Kolkata, for their unwavering support in facilitating project work.
Furthermore, we acknowledge with deep gratitude the contributions of past and present researchers in the field of Artificial Intelligence, particularly in Natural Language Processing. Their groundbreaking work has laid the foundation upon which our own research endeavors have been built, and without their insights and advancements, our achievements would not have been attainable.

CONTENTS

1. INTRODUCTION

Cybersecurity, a critical aspect of the modern landscape of technology, encompasses a large collection of components ranging from encryption protocols to network security measures. Despite the abundance of resources and guidelines, individuals often encounter challenges in understanding fundamental concepts and terminologies within the cybersecurity domain. In this context, the development of a conversational agent capable of addressing simple knowledge queries could serve as a valuable resource for both beginners and professionals alike.The primary objective of this dissertation is to design and implement a retrieval-based chatbot equipped with NLP capabilities and feed-forward neural networks to efficiently respond to basic questions about various components of cybersecurity. The scope of this study encompasses the development of a knowledge repository, the design and training of the chatbot model, and the evaluation of its performance in accurately retrieving and presenting relevant information to users. Traditional methods of question-answer processes often involve static documents or webpages, which may not be apt to users' specific queries in a timely or accessible manner. By making use of conversational agents, users can engage in natural language interactions, thereby facilitating the retrieval of relevant information in a conversational format. This approach not only enhances user engagement but also promotes a deeper understanding of cybersecurity concepts through interactive exploration.The methodology employed in this study comprises several stages, including data gathering, preprocessing, model design, training, and evaluation. Initially, a dataset consisting of cybersecurity-related queries is collected and processed. Subsequently, a retrieval-based chatbot model is constructed, incorporating NLP techniques for text understanding and feed-forward neural networks for response generation. The model is trained on the prepared dataset and evaluated based on its ability to accurately retrieve and present relevant information in response to user queries.

In this project, a retrieval-based chatbot designed to address cybersecurity queries is showcased. Utilising PyTorch and deep learning techniques, it operates through pattern matching rather than contextual analysis. This approach enables the chatbot to provide accurate responses tailored to specific queries, enhancing user interaction and understanding.

By focusing on pattern recognition, the chatbot ensures efficiency and reliability in addressing cybersecurity concerns. This introduction highlights the project's methodology, emphasising its reliance on PyTorch and deep learning for developing a robust chatbot which tailors to the intricacies of cybersecurity-related queries.

1.1 PROBLEM STATEMENT

Within the domain of Natural Language Processing, this project works to construct a pattern-based chatbot tailored for cybersecurity queries. Operating through retrieval-based mechanisms, it prioritizes resource efficiency while delivering accurate responses to cybersecurity inquiries by making use of pattern recognition and semantic understanding algorithms.

1.2 PURPOSE OF THE PROJECT

- Leveraging NLP to deal with text-based problems
- Deploy a Retrieval-Based chatbot to construct a question-answer framework
- Help answer knowledge-based queries regarding cybersecurity
- Delve into the concepts of Deep Learning and integrate them into the application of this project

1.3 SCOPE OF THE PROJECT

- Implements the usage of deep learning and feedforward neural network to build the retrieval-based chatbot
- Includes NLP techniques for text manipulation
- The dataset used consists of carefully tailored tags, patterns and responses natural language sentences and words for the training of the model
- The model is a simple feed-forward neural network
- The model takes in cybersecurity knowledge level queries and returns a respective output
- The chatbot interface is used to provide a better conversational environment

1.4 APPLICABILITY OF THE PROJECT

- **Education**: Students can use it to learn cybersecurity basics effectively.
- **Corporate Use:** Employees can access cybersecurity guidelines quickly.
- **Professional Development**: Cybersecurity experts can stay updated on emerging threats.
- **Online Forums:** It can assist users in cybersecurity discussions and problem-solving.
- **Events/Workshops**: Attendees can use it to navigate sessions and get instant answers.

However, the project can be extended to domains other than cybersecurity, in general

- **Customer Support:** Can provide instant assistance and information to customers in different industries.
- **Healthcare:** Offers patients access to medical information and assistance with health queries.
- **E-commerce:** Helps shoppers find products, track orders, and get assistance with purchases.
- **Travel:** Assists travellers with trip planning, bookings, and destination information.

- **Finance:** Provides users with banking services, financial advice, and investment information.
- **Human Resources:** Assists employees with HR-related queries, policies, and procedures.

1.5 LIMITATIONS

1. Pattern recognition and contextual understanding: Chatbots lack deep understanding, respond based on superficial assumptions, are risk-averse and lack a nuanced understanding of user queries beyond page-level patterns

2. Possible incorrect results: Relying on pattern recognition can lead to incorrect answers when inputs vary or patterns are interpreted incorrectly, especially on computers securely, and encourages users to verify information.

3. Domain restriction of cybersecurity: Professional in cybersecurity, unable to address questions outside this domain, requiring users to search for additional sources of non-cybersecurity information.

4. Limited creativity in responses: Responses based on pre-existing models that do not obtain original information or adapt to changing circumstances, which may yield information that is predictable or outdated for the users.

5. Lack of emotional intelligence: Unable to recognize or respond to emotional cues, risks sounding deaf and insensitive, especially in sensitive cybersecurity situations, which can cause frustration or misunderstanding in the machine.

2. LITERATURE SURVEY

Retrieval-based chatbots have gained popularity because they can chat by finding and using answers from a database. This survey aims to explore how these chatbots work, the challenges they face, and how they can improve. Understanding retrieval-based chatbots is essential as we increasingly interact with technology through conversation.

Initially, chatbots followed rules to decide what to say based on patterns. Then, they started using fancy math to match questions with better answers. With new methods like deep learning, chatbots can now understand conversations better by looking at the whole chat history (Zhang et al., 2020). These advancements have made chatbots more accurate and helpful.

Despite improvements, chatbots still struggle to understand the context of conversations and may give wrong answers. The quality of the information they use is also important. If the data is old or wrong, the answers won't be helpful (Xu et al., 2022). Moreover, as more information becomes available, chatbots need better ways to find what's relevant quickly (Chen et al., 2021). Additionally, ensuring privacy and security in handling sensitive user data is a big concern for chatbot developers (Wang et al., 2020).

Researchers are finding ways to teach chatbots using lots of examples so they can understand different topics better (Wu et al., 2020). They're also working on making chatbots better at having longer conversations. Some techniques, like using memory and attention, help chatbots remember what was said earlier and keep the conversation going (Zhang et al., 2020). Hybrid approaches that mix different methods are also being explored to make chatbots give more accurate and helpful responses (Shen et al., 2021). Continued advancements and innovative ideas will further enhance the capabilities of retrieval-based chatbots, enabling more natural and effective human-computer interactions.

Retrieval-based chatbots are crucial for conversing with computers more naturally. Despite improvements, challenges like understanding context and ensuring data quality persist. With ongoing research and new ideas, retrieval-based chatbots can become even better at helping us have meaningful conversations with them. Understanding the landscape of retrieval-based chatbots is vital as they continue to evolve and play an increasingly significant role in our interaction with technology.

3. SURVEY OF TECHNOLOGIES

3.1 DEEP LEARNING

Deep learning falls under the umbrella of intelligence. Replicates the neural networks of the human brain to handle large volumes of data. By utilizing layers of nodes deep learning algorithms can identify patterns, make predictions and categorize data with precision. By employing methods such as backpropagation and gradient descent it autonomously learns from data to enhance its performance. Deep learning drives a range of applications, including image recognition, speech processing and natural language understanding. This technology is transforming sectors, like healthcare, finance and technology by excelling in deciphering data structures.

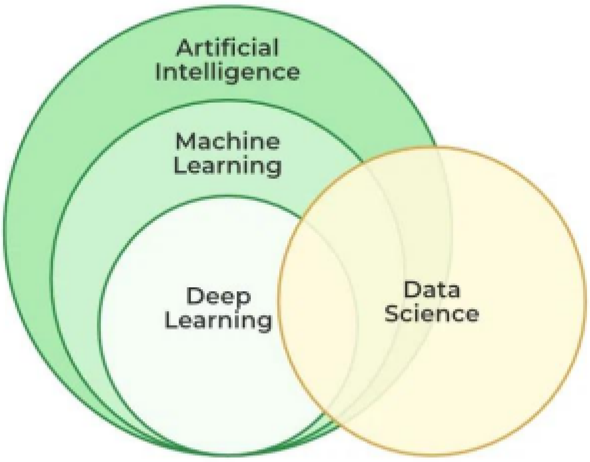


FIG 1 :- THE AI VENN DIAGRAM

3.2 NEURAL NETWORKS

A neural network is a system that takes cues from the way the human brain works. It consists of linked nodes, also known as "neurons ", arranged in layers. Each neuron handles signals, performs calculations and sends the output to the following layer. By undergoing training, the network fine tunes its settings to grasp patterns and connections, in data. Neural networks shine in activities like sorting, prediction and identifying patterns playing a role in realms such as AI, machine learning and data examination.

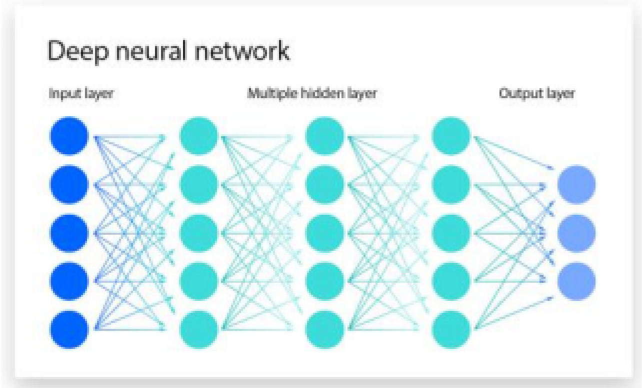


FIG 2 :- Deep Neural Network

3.3 FEED-FORWARD NEURAL NETWORKS

A feed forward neural network serves as a framework, in the realm of intelligence. It is composed of layers of interconnected nodes through which data flows unidirectionally—from input nodes traversing layers to output nodes. Each node analyzes data by utilizing weighted connections and activation functions converting input signals into outputs. This network design facilitates pattern identification, categorization and prediction assignments showcasing its adaptability across fields such as image analysis, language processing and financial projection. Its simplicity and effectiveness contribute to its widespread adoption of machine learning applications.

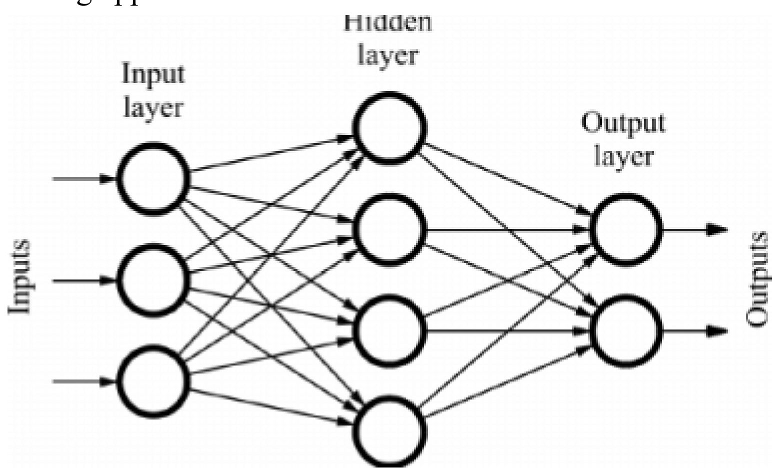


FIG 3 :- FEED-FORWARD NEURAL NETWORK

3.4 RETRIEVAL-BASED CHATBOTS

A retrieval-based chatbot uses a pre-set database or knowledge base from which it can draw answers. They don't generate responses themselves like generative models; rather, they choose them from a pool of set replies. This ensures that the responses are accurate and coherent since the bot retrieves relevant information based on the input query. Retrieval-based chatbots use this method to give exact answers, speed up communication and improve user experience in areas such as customer service, information retrieval, and virtual assistance.

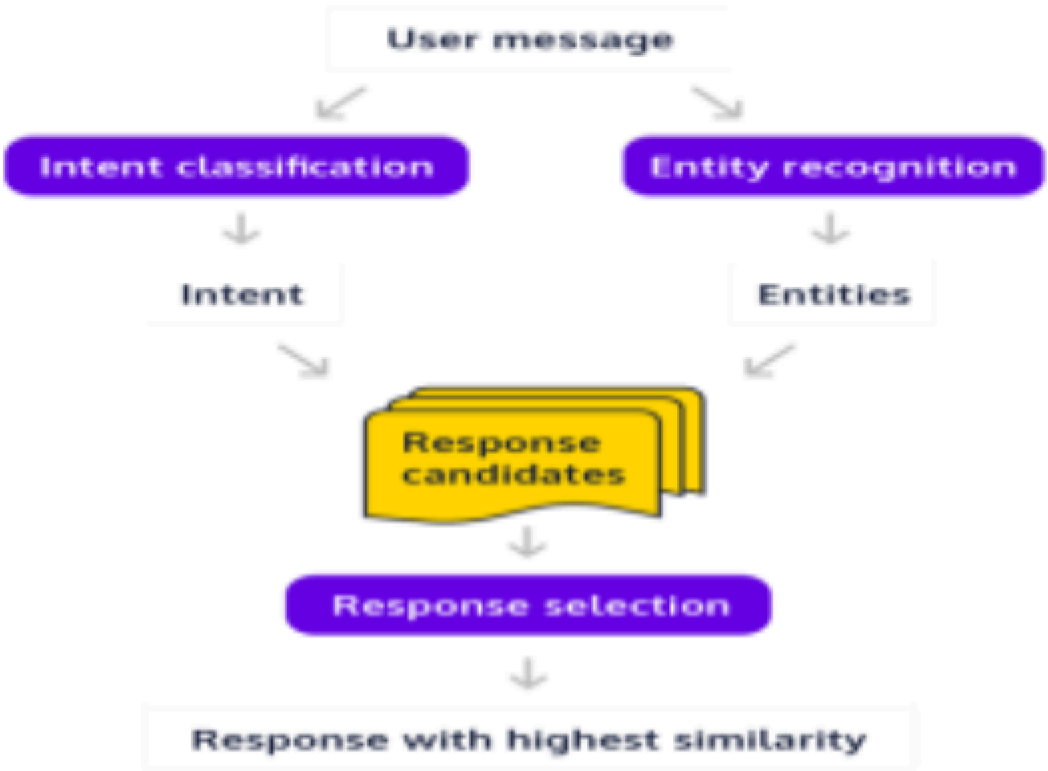


FIG 4 :- RETRIEVAL-BASED CHATBOTS< MECHANISM

3.5 BAG OF WORDS

Bag of Words (BoW) is the text analysis technique that holds a central place in natural language processing (NLP). It just counts words without considering grammar or order and hence some people refer to it as a "bag". The number of times each word appears in the document thus becomes its feature. The simplicity of this method makes it possible to treat text as numerical vectors, thus enabling computational operations like clustering or classification. Despite its basic nature and loss of context, BoW is still an underpinning concept in NLP applications such as sentiment analysis, document categorization and information retrieval.

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

FIG 5 :- BAG OF WORDS, MECHANISM

3.6 CROSS ENTROPY LOSS

Cross-entropy, also referred to as logarithmic loss or log loss, is a widely used loss function in machine learning for evaluating the performance of classification models. It quantifies the average number of bits required to distinguish events from one probability distribution using the optimal code for another probability distribution. In essence, it assesses the disparity between the predicted probability distribution generated by a classification model and the actual values. During training, the cross-entropy loss function guides the adjustment of model weights to minimize the error between predicted and actual outcomes, aiming for a measure closer to 0, indicative of a well-performing model, while a measure closer to 1 suggests poor performance.

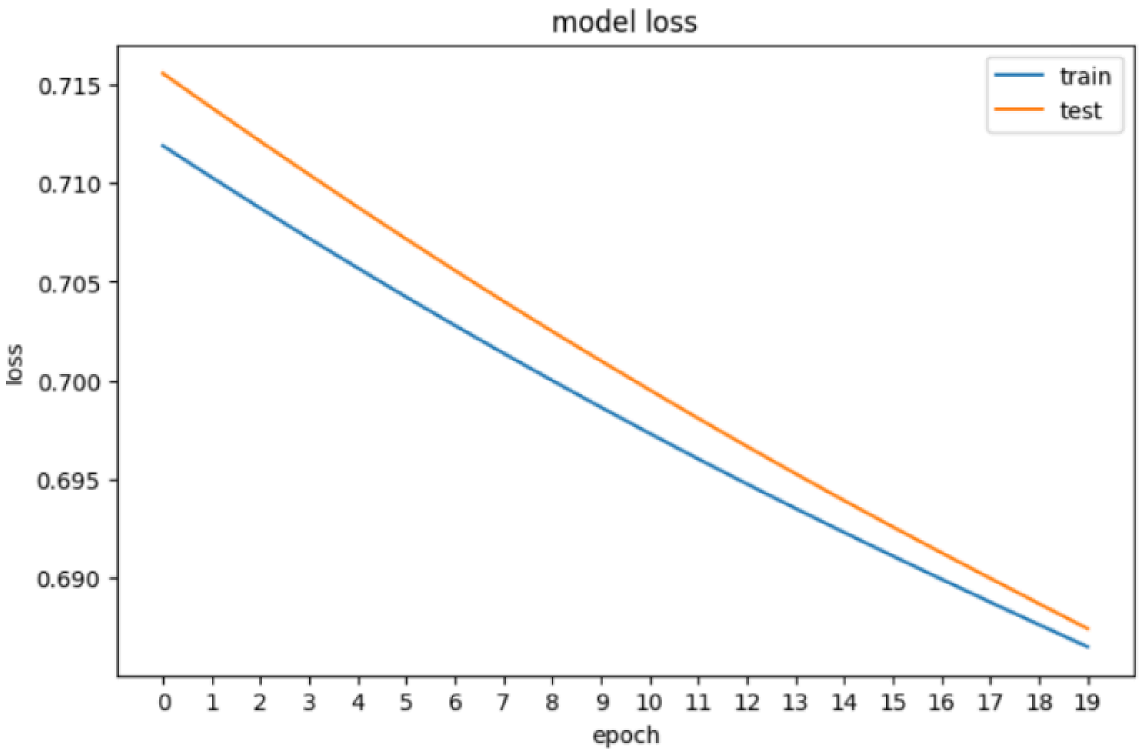


FIG 6 :- BINARY CROSS ENTROPY, MATHEMATICAL GRAPH

3.7 GRADIENT DESCENT

Gradient descent is a machine learning and optimization algorithm that seeks to minimize the loss function. The function of this algorithm is to iteratively adjust parameters in the direction of greatest drop off in the loss function as it tries to identify optimum values. To achieve this, the algorithm computes gradients of the Loss Function with respect to parameters and then updates them. By repeatedly adjusting parameters based on gradients, Gradient Descent progressively converges towards the minimum of the loss function, facilitating efficient model training and parameter optimization.

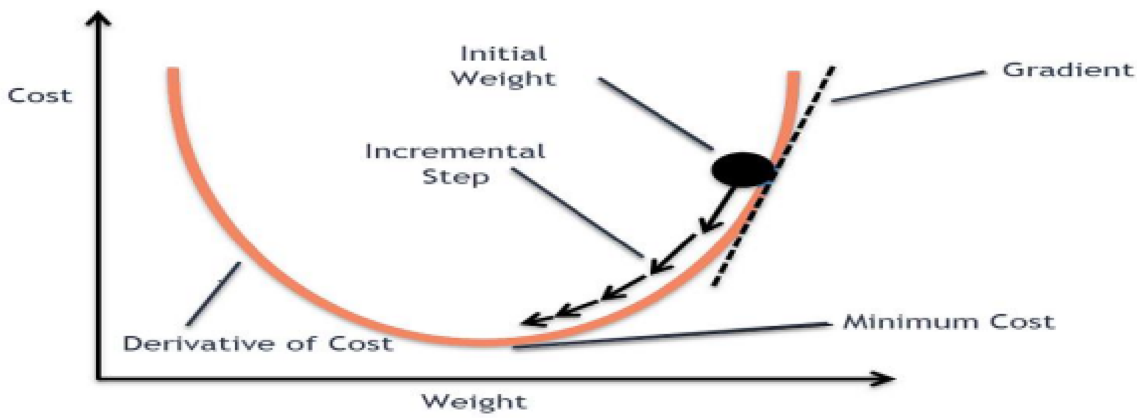


FIG 7 :- GRADIENT DESCENT, COST VS WEIGHT

3.8 RECTIFIED LINEAR UNIT (ReLU)

The Rectified Linear Unit (ReLU) is a popular activation function in neural networks. Emitting zero for negative input values and going unchanged for positive values results in nonlinearity. This simplicity speeds up the computation and contributes to gradient-based optimization during training. The effectiveness of the ReLU lies in reducing the problem of the vanishing gradient, in training deeper correlations. Despite its simplicity, ReLU products improve network resolution, minimalism, and computational performance, becoming the cornerstone of modern deep learning systems and contributing to significant advances in fields such as computer vision, natural language processing, and in reinforcement learning.

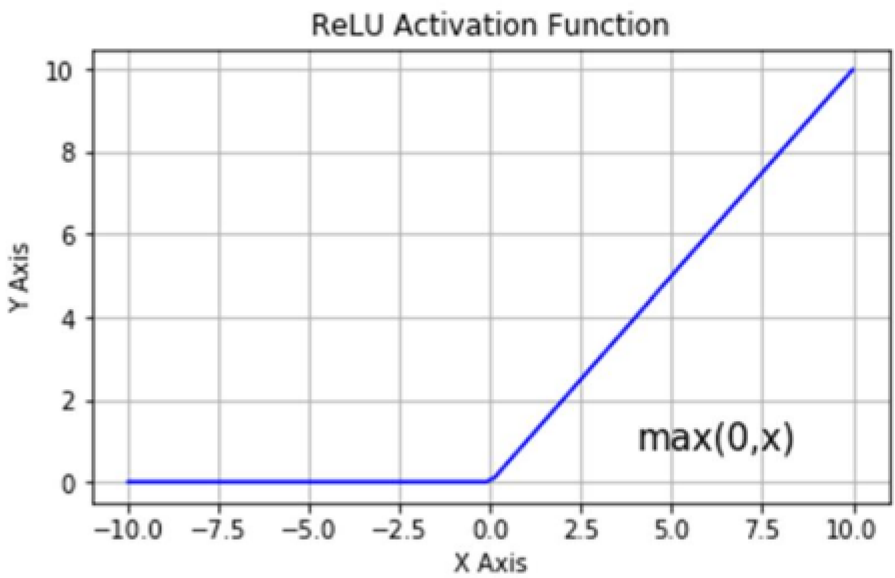


FIG 8:- ReLU ACTIVATION FUNCTION, GRAPH

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

- Python 3.9.5 or higher
- pip 24.0
- OS : Windows 11 or higher / MAC OS / Linux
- Nvidia CUDA libraries for Nvidia GPU utilization OR
- OpenCL library for AMD GPU utilization

4.2 HARDWARE REQUIREMENTS

These are the minimum hardware requirements for executing this project:

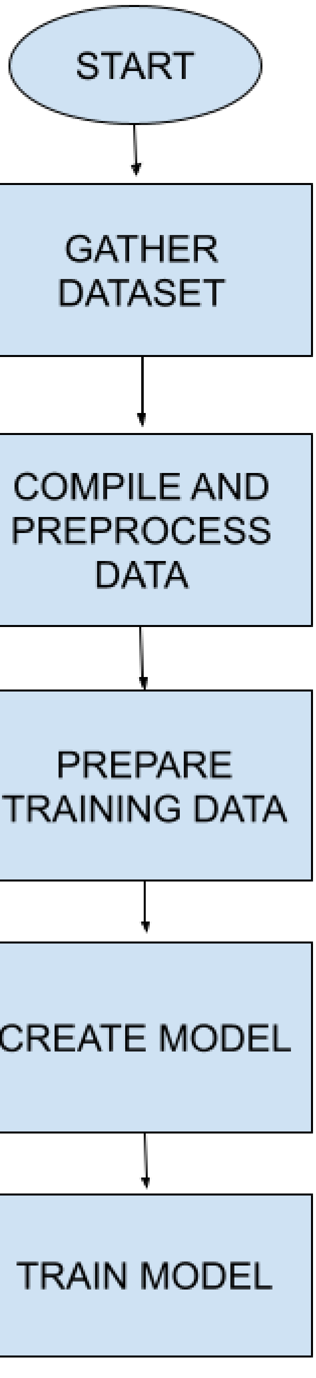
- CPU – Intel Core I3, For Mac: M1, M2
- RAM – At Least 4 GB of Memory
- Nvidia CUDA libraries for Nvidia GPU utilization
- 5GB of HDD space
- A valid internet connection

4.3 LIBRARY REQUIREMENTS

- json5==0.9.17
- jupyter==1.0.0
- matplotlib==3.8.3
- numpy==1.26.3
- pandas==2.2.1
- nltk==3.8.1
- scikit-learn==1.4.1.post1
- streamlit==1.31.1
- torch==2.2.0

5. SYSTEM DESIGN

5.1 PROJECT WORKFLOW



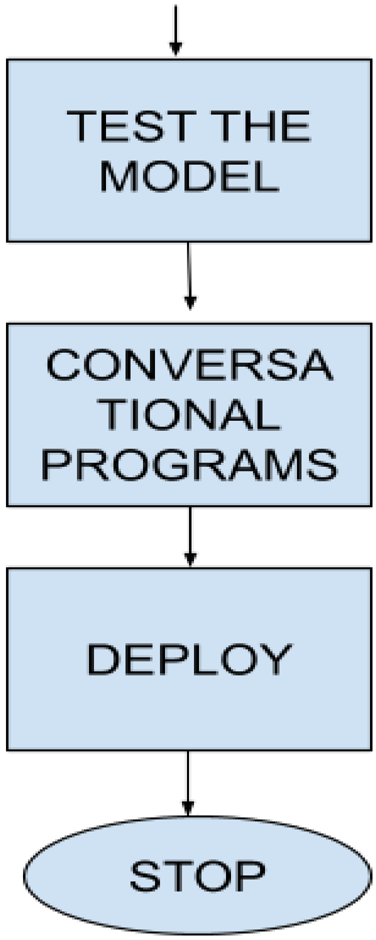


FIG 9 :-PROJECT WORKFLOW

1. START: Start the project by defining the required goals, scope, and resources.

2. GATHER DATASET: Collect conversational data and cybersecurity-related data from sources

3. COMPILE AND PREPROCESS DATASET: Clean up the compiled data set by removing noise, redundancy, and formatting inconsistencies. Tokenize text data into individuals or phrases and use preprocessing techniques such as stemming or lemmatization to normalize the text. Punctuation (common words like "the," "is," etc.) and punctuation marks are often omitted to focus on meaningful information.

4. CREATE TRAINING DATASET: Configure the pre-generated data into a suitable input-output pair for the training chatbot model. It organizes conversational exchange into context-response pairs, where context is input, and response is corresponding output.

5. CREATE MODEL: Create a chatbot model based on retrieval using a feed-forward neural network. Use methods such as bag-of-words or TF-IDF (Term Frequency-Inverse Document Frequency) to represent input. Implement the ReLU (Rectified Linear Unit) activation function in a neural network architecture that can introduce nonlinearities and improve model performance to identify complex patterns in the data.

6. TRAIN MODEL: Optimize the parameters of the chatbot model using the gradient descent
optimization algorithm. The ancient model lets it minimize the binary cross-entropy loss function, which measures the difference between the predicted response and the actual response. This strategy requires repeatedly feeding a set of training data into the model, adjusting the weights and biases of the model to improve its performance in generating correct responses.

7. TEST THE MODEL ALONG VARIOUS PARAMETERS: Use different metrics such as accuracy, precision, recall, and F1 score to evaluate the performance of the trained model. Experiment with various hyperparameters such as learning rate, batch size, and network architecture to improve the performance of the model.

8. CONVERSATIONAL PROGRAMMING: Implement logic in conversation flow and response generation using Natural Language Processing (NLP) techniques. Use tools and libraries such as NLTK for tasks such as named entity detection, part-of-speech tagging, and syntax parsing. Develop algorithms for contextual understanding, mood recognition, and response generation that enable the chatbot to engage in meaningful conversations with users.

9. DEPLOY AND STOP: Close the project by documenting the work plan, conclusions, and insights gained during the development and testing phase. Deploy the trained chatbot prototype to enterprises for real-world use, and continuously review and update the prototype as needed to maintain its effectiveness and usefulness over time.

5.2 DATASET

5.2.1 DATASET DESCRIPTION

The foundation of any successful natural language processing (NLP) project lies in the quality and relevance of the data types it uses. In this project, a dataset named ‘intents.json’ is the cornerstone of a project that focuses on cybersecurity-related questions. Let’s delve into the various features and components to understand the importance and functionality of this dataset.

Data Set Structure and Source:
The data set in JSON format is appropriately named 'intents.json'. JSON (JavaScript Object Notation) is a small data exchange format commonly used to store and transfer data between a server and a web application. This chart is human-readable and easy to navigate, making it ideal for this project.

The data in 'intents.json' has been carefully compiled and compiled from a variety of cybersecurity related sources. These resources can include cybersecurity databases, websites, forums, research papers, and informational channels. The comprehensive nature of the data collected ensures that a wide range of cybersecurity issues and questions are covered in the data set. The components of a data structure:

Tags: Tags act as categorical labels or identifiers for disciplines or topics in the cybersecurity field. Each tag represents a specific set of questions or concepts. For example, tags can include topics such as "encryption ," "firewall," and "ARPs." These tags provide an organizational structure for the data set, allowing for better classification and retrieval of relevant information.

Patterns: Patterns refer to ideas or concepts related to questions or questions commonly encountered in each identified category. These patterns represent natural language prompts or queries that allow users to search for information or guidance on cybersecurity. For example, under the "Encryption Techniques"
tag, examples might include questions such as "What is symmetric encryption?" or "How does RSA encryption work?" This process is essential for training on NLP models for understanding and answering user questions correctly.

Response: Response are the corresponding answers or explanations given for each instance in each tag. These responses clarify concepts, provide guidance, or offer solutions related to the specific cybersecurity issue at hand. Continuing with the previous example, for examples like "What is symmetric encryption?" The corresponding answer may provide a brief explanation of symmetric encryption principles, algorithms, and information processing.

```
{
    "tag": "ARP",
    "patterns": [ "What is an ARP and how does it work?", "what is ARP?", "arp in cyber security?", "What 
    "responses": [ "Address Resolution Protocol (ARP)is a protocol for mapping an Internet Protocol addres
},
```

FIG 10 :- Small Snippet from intents.json

Importance and Benefits:
The structured structure of the data set provides many advantages in terms of tags, patterns, and responses:

Pattern matching: The data set facilitates pattern matching, allowing NLP models to accurately recognize and interpret user information. By matching user input and predefined patterns, the model can identify the most appropriate tags and retrieve the appropriate response.
Knowledge level: Each tag is a repository of knowledge with cybersecurity components. By linking patterns to informative responses, the dataset effectively encapsulates domain-specific knowledge, making it accessible and actionable.
Scalability and adaptability: The modular design of data collection enables scalability and adaptability to evolve cybersecurity trends and challenges. New tags, samples, and comments can be added easily to increase the coverage and relevance of the dataset over time.
Enhanced user experience: By leveraging data types, NLP-based cybersecurity applications can provide timely, accurate and informative answers to handling queries of the role, to enhance overall user experience and satisfaction.

In summary, the 'intents.json' data set is a valuable resource for developing customized solutions using the power of NLP to solve cybersecurity questions (specifically for this project paper) and challenges Its structured organization its broad inclusion of tags, patterns, and response lays the foundation for building robust and effective cybersecurity support systems.

5.2.2 DATA PREPROCESSING

This process is aimed at refining textual data for natural language understanding tasks, particularly in the context of intent classification. It involves parsing a structured data file, typically named 'intents.json', containing various intents and associated patterns, often utilized in applications like chatbots.

Before proceeding towards the technicality of the data preprocessing mechanism incorporated in this project let us first understand what is:

STEMMING : Stemming is like simplifying words to their root form. Imagine you have words like "running," "ran," and "runs." When you stem them, you strip off the endings so that they all become "run." It's like finding the base or simplest version of a word, which can help computers understand different variations of the same word as being similar. So, stemming helps in grouping together words that have the same meaning but might look a little different.

According to '*Manika*', as documented in '*Stemming in NLP- A Beginner's Guide to NLP Mastery*', '*The process of removing affixes from a word so that we are left with the stem of that word is called stemming. For example, consider the words 'run', 'running', and 'runs', all convert into the root word 'run' after stemming is implemented on them.* '

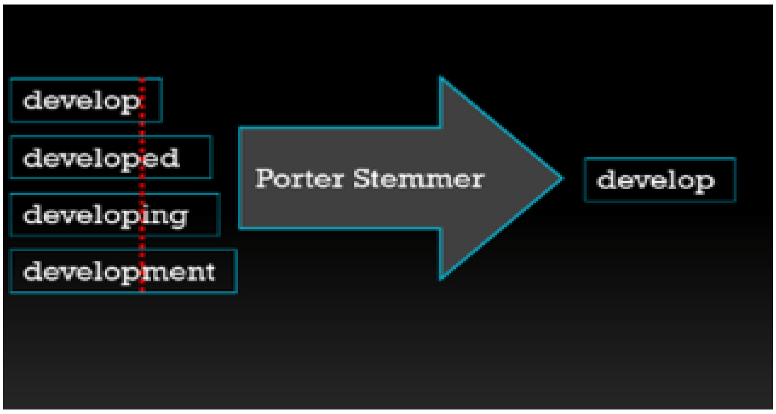


FIG 11 :- Stemming using Porter Stemmer

The stemming algorithm used in this project is the "Porter Stemmer", derived from the nltk library.

Porter stemming is a basic method to simplify words to their root form, mainly used for English. Here's how it works:

- If a word ends with "sses," Porter stemming chops off the ending to just "ss." For example, "princess" becomes "princ."
- When a word ends with "ies," Porter stemming replaces it with "i." For instance, "parties" becomes "parti."

- If a word ends with "ss," Porter stemming doesn't change it. For example, "grass" remains "grass."
- When a word ends with "s," Porter stemming removes this suffix. For example, "cats" turns into "cat."

Porter stemming is popular for its simplicity, especially for English words, and it's known to provide accurate results with minimal errors compared to other methods. Its straightforward rules make it efficient for various text processing tasks.

TOKENIZATION : Tokenization in NLP is like breaking down a piece of text into smaller chunks, such as words or sentences. Imagine you have a sentence like "Hello, how are you?" Tokenization would split it into individual words: "Hello", ",", "how", "are", "you", and possibly remove punctuation marks like ",". Essentially, tokenization helps a computer understand and work with words or sentences as separate units, making it easier to analyze and process text data. *"Tokenization is a simple process that takes raw data and converts it into a useful data string. While tokenization is well known for its use in cybersecurity and in the creation of NFTs, tokenization is also an important part of the NLP process. Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning."* as documented by '*Anni Burchfeil*' in '*Tokenex - what is NLP tokenisation*'

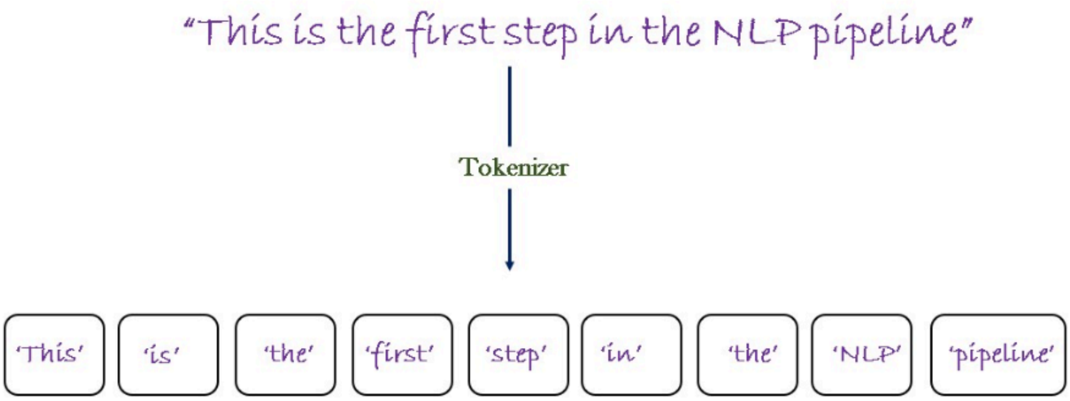


FIG 12 :- TOKENISATION IN NLP

5.2.2.1 USAGE OF STEMMING AND TOKENISATION FOR DATA PREPROCESSING

```python
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)


def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())
```

FIG 12 :- Snippet of usage of porter stemming and tokenisation in the project

The purpose of this process is to organize and refine raw text data for use in natural language processing (NLP) tasks, which includes text classification with respect to this project.It involves extracting intents and associated patterns from a JSON file and formatting the data for analysis and model training.

Key Steps:

Data Loading:

- The process starts by reading structured data from a file named 'intents.json'. This file contains information about different intents and their corresponding patterns and suitable responses.

Data Organization:

- Lists are created to store important data:
    - 'all_words': Holds individual words from patterns.
    - 'tags': Keeps track of unique intent tags.
    - 'xy': Stores pairs of tokenized words and intent tags.

Data Parsing:

- Each intent in the data is examined, and its tag is extracted. The associated patterns are then analyzed.

Tokenization:

- Patterns are broken down into words or punctuation marks using tokenization. This helps in analyzing the text effectively.

Data Aggregation:

- Tokenized words and their intent tags are combined and added to the 'xy' list. This links words to their intents for model training.

Word Processing:

- Further processing is applied to words:
    - **Stemming:** Words are simplified to their root forms for consistency.
    - **Lowercasing**: All words are converted to lowercase for uniformity.
    - **Removal of Specific Words:** Punctuation marks and similar elements are excluded from analysis.

Data Refinement:

- Duplicate words and intent tags are removed, ensuring data integrity.

Data Insights:

- The process concludes by providing insights into the processed data:
    - Total patterns processed.
    - Number of unique intent tags.
    - Count of unique stemmed words.

This process sets the groundwork for creating and functioning of the nlp based chatbot, making raw text data suitable for analysis, classification, or text generation.

5.2.3 CREATING TRAINING SET

This section focuses on preparing the training data for the model in the context of natural language processing (NLP), which will serve as the core of the retrieval-based chatbot. It involves converting textual data into numerical representations that can be understood by the algorithms and structures which will be used for the building of the model.

Bag of Words : "*The bag of words model is one particularly simple way to represent a document in numerical form before we can feed it into a machine learning algorithm. For any task based on NLP, we need a way to accomplish this before any further processing. Machine learning algorithms can't operate on raw text; we need to convert the text to some sort of numerical representation. This process is also known as embedding the text." - Noah Topper, BuiltIn*

Bag of Words (BoW) is a technique used to represent text data in a simple and straightforward manner for computer processing. Imagine you have a big bag, and you throw in all the words from a bunch of sentences. Now, when you want to know what's in the bag, you just count how many times each word appears, regardless of the order they were thrown in.

So, for example, if you have the sentences "I love cats" and "Cats are cute," your bag would have words like "I," "love," "cats," "are," and "cute." The count for each word tells you how many times it appears in all the sentences. This way, the Bag of Words represents the text data as a collection of words and their frequencies, ignoring the sequence or context in which they appear. It's a simple and effective way for computers to understand and work with text data.

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

FIG 13 :- Bag of words representation of some NL sentences

5.2.3.1 INCORPORATING BOW IN THE PROJECT

```
def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bag   = [  0 ,    1 ,    0 ,   1 ,    0 ,    0 ,      0]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag
```

FIG 14 :- Snippet of BOW usage in the project

The purpose of this data preparation process is to transform textual data into a format suitable for training machine learning models. By converting words into numerical representations and labels into numerical values, the data becomes compatible with classification algorithms.

Key Steps:

Data Structuring:

- Two lists, namely X_train and y_train, are initialized to hold the training data. X_train will store the features (bag of words), and y_train will store the corresponding labels.

Feature Extraction:

- For each pair of pattern sentence and its associated tag, the bag of words representation is computed. This representation captures the frequency of occurrence of each word in the pattern sentence compared to a predefined set of all words.

Label Encoding:

- Each tag is encoded into a numerical label. This step is necessary for machine learning models, as they require numerical labels to perform classification tasks.

Data Compilation:

- The feature vectors (bag of words) and their corresponding labels are added to the X_train and y_train lists, respectively.

Data Conversion:

- Finally, the lists X_train and y_train are converted into numpy arrays to facilitate efficient processing by machine learning libraries.

5.3 NEURAL NETWORKS IN CHATBOTS

Retrieval-based chatbots can make use of **neural networks** to retrieve pre-defined responses or information from a database or knowledge base. Here's a simplified explanation of how they work:

Input Understanding: When a user sends a message, the neural network first understands the meaning behind it. This involves breaking down the message to identify keywords, intents, and context.
Matching with Responses: The neural network then compares the understood input with a database of predefined responses or information. It looks for the closest match based on similarity or relevance.
Selecting the Best Response: Once potential responses are identified, the neural network selects the best one to send back to the user. This could be based on factors like the accuracy of the match, the context of the conversation, or user preferences.
Learning and Improvement: Neural networks can continuously learn and improve over time. They analyze user interactions and feedback to refine their matching algorithms and provide better responses in the future.

In essence, neural networks enable retrieval-based chatbots to understand user queries, match them with appropriate responses or information from a database, and continuously improve their performance through learning from user interactions.

5.3.1 FEED FOWARD NEURAL NETWORK

Feedforward neural networks are a type of artificial neural network where information moves in one direction: forward. Here's a brief overview:

Input Layer: This is where the data or input is fed into the network. Each input neuron represents a feature of the input data.
Imagine this as the part where you feed information into the network, like giving it a picture to recognize or text to understand.

Hidden Layers: Between the input and output layers, there can be one or more hidden layers. Neurons in these layers perform computations on the input data. Each neuron receives inputs from the previous layer, processes them through an activation function, and passes the result to the next layer.

These are like the brain of the network. It's where the network processes the information you give it. Each layer does some math based on the input it receives.

Weights and Biases: Each connection between neurons in adjacent layers has a weight associated with it. These weights determine the strength of the connection. Additionally, each neuron has a bias term, which helps the network adjust and fit the data better.

Think of these as knobs that the network turns to make its calculations. They help the network adjust its calculations to get closer to the right answer.

Activation Functions: Neurons in each layer typically apply an activation function to the weighted sum of their inputs. This introduces non-linearity to the network, enabling it to learn complex patterns in the data. Common activation functions include sigmoid, tanh, ReLU, and softmax.

These are like filters that help the network decide whether to 'fire' or not. They add some complexity to the calculations, allowing the network to learn more complex patterns.

Output Layer: The final layer of the network produces the output. The number of neurons in this layer depends on the task—e.g., one neuron for binary classification, multiple neurons for multi-class classification or regression tasks. The activation function in the output layer depends on the nature of the problem.

This is where the network gives you its answer. For example, if you're showing it pictures of cats and dogs, it might say whether the picture is of a cat or a dog.

Forward Propagation: During forward propagation, input data is fed into the network, and computations are performed layer by layer until the output is produced. The output is then compared to the expected output to calculate the loss.

This is just a fancy term for the process of the network making its calculations from input to output, like a relay race passing a baton from one runner to the next.

Loss Function: A loss function measures how well the network's output matches the expected output. The goal of training is to minimize this loss function by adjusting the weights and biases using optimization algorithms like gradient descent.

This is like a report card that tells the network how well it did. If it got the answer wrong, the loss function tells it how much it missed by. The network then adjusts its knobs (weights and biases) to try and get a better grade next time.

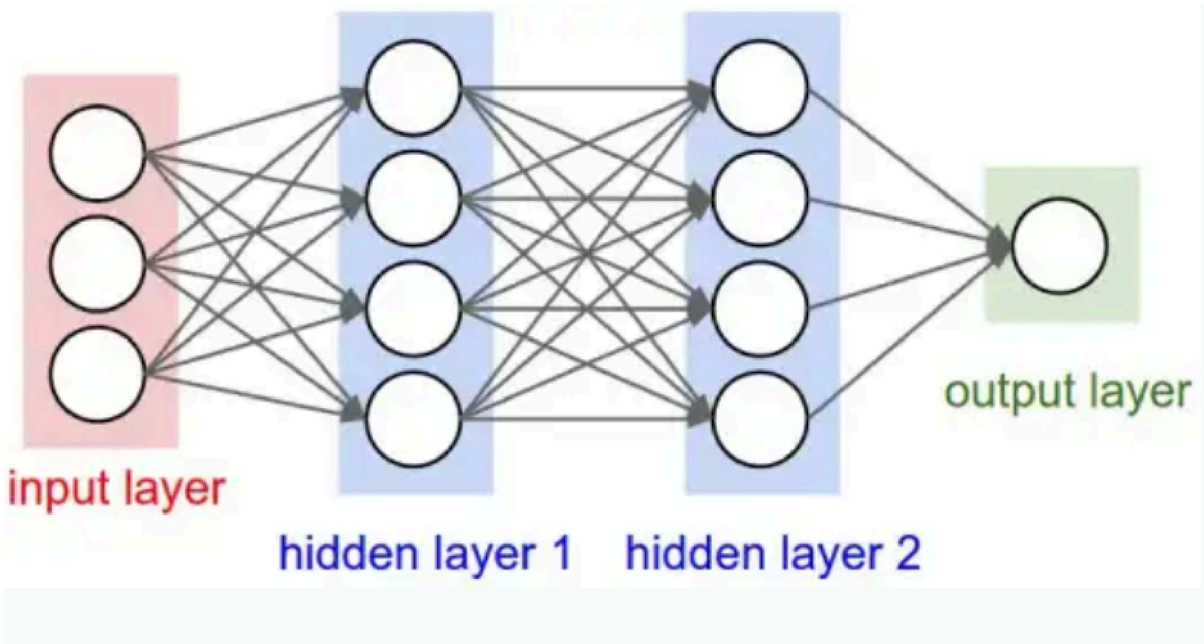


FIG 15 :- Feed Forward Neural Network

5.3.2 FEED FORWARD NEURAL NETWORK IN THE PROJECT

A feedforward neural network operates like a simplified model of the human brain. It takes inputs, processes them through layers of interconnected nodes, and produces outputs. Each node, or neuron, performs mathematical operations on the input data and passes the result to the next layer.

Network Architecture:

Our neural network consists of three main components: input layer, hidden layers, and output layer. The input layer receives data, such as text inputs from users in a chatbot scenario. The hidden layers perform calculations on this data, gradually transforming it into meaningful representations. Finally, the output layer generates responses or predictions based on the processed information.

Layer Functions:

Within each layer, we employ linear transformations and activation functions. Linear transformations adjust the input data using learnable parameters, while activation functions introduce non-linear properties to the network, enabling it to capture complex patterns in the data. The activation function used in the project is the ReLU, i,e Rectified Linear Unit.

Model Operations:

During the forward pass, data flows sequentially through the layers of the network. Each layer applies its transformation and passes the result to the next layer until the final output is generated. Our model does not apply an activation function at the output layer, leaving the decision-making process to downstream applications.

6. IMPLEMENTATION AND MODEL EVALUATION

6.1 IMPLEMENTATION

6.1.1 THE MODEL

THE NETWORK ACHITECTURE

THE INPUT LAYER : Picture yourself engaging in conversation with the chatbot. Your typed messages serve as the initial layer of the neural network, aptly named the input layer. This layer acts as a gateway, gathering all the words you convey. Imagine chatting with the bot. Your typed messages become the first layer of the neural network, called the input layer. It collects all the words you send. The gateway through which user inputs are received and processed. Mathematically, it's represented as:

Input Layer=x

```
self.l1 = nn.Linear(input_size, hidden_size)
```

THE HIDDEN LAYERS : This project uses two hidden layers. The messages travel through hidden layers. These layers act like the chatbot's brain, processing your words. Each layer carefully analyzes the information, sorts it out, and passes it to the next layer. Think of these layers as the chatbot's way of making sense of what you're saying.

In a neural network's hidden layer, neurons serve as small decision-makers, responsible for understanding and processing incoming information. Similar to mailboxes receiving letters, each neuron in the hidden layer receives numerical data representing different features of the input. Once the information is received, neurons perform mathematical operations by multiplying each piece of data by a special number called a weight and then adding them all up, akin to mini calculators. To add flexibility to their decision-making process, neurons apply an activation function like ReLU (Rectified Linear Unit), which adjusts the information based on certain rules, helping them capture more complex patterns in the data. Finally, neurons pass their processed information to the next layer or directly to the output layer, contributing their unique perspectives to the network's decision-making process. Overall, neurons in the hidden layer play a crucial role in transforming input data into meaningful insights, aiding the neural network in tasks like recognizing patterns in images or understanding language in chatbots.

```
self.l2 = nn.Linear(hidden_size, hidden_size)
self.l3 = nn.Linear(hidden_size, num_classes)
```

LINEAR TRANSFORMATION : Inside each hidden layer, there's an important step called **linear transformation**. It's like fine-tuning the information to make it clearer. This step helps the chatbot understand which words are more important in your message.
in a neural network, a linear transformation tweaks the input data by multiplying each piece of information by a special number called a weight and then adding them all up. This process helps the network understand which parts of the input are more significant for making predictions or decisions. So, linear transformations are important because they help the network learn from data and make sense of complex patterns, ultimately improving its ability to solve tasks like recognizing images or understanding language.

Mathematically, the operations within each hidden layer can be represented as:

$$y = xA^T + b$$

- 'x' is the incoming data. It must be a tensor of dtype float32 and shape (*, in_features). Here * is any number of dimensions. in_features is number of features in the input data.
- 'y' is the output data after the transformation with same dtype as 'x' and with shape (*, out_features). Note that all dimensions except last are of the same shape as input data.
- A is the learnable weights of shape (out_features, in_features). out_features is the last dimension of the output data.
- b is the additional bias learned during the training

ACTIVATION FUNCTION (ReLU) : Accompanying linear transformations are activation functions, among which the Rectified Linear Unit (ReLU) stands out. ReLU injects a layer of complexity into the chatbot's cognitive processes, enabling it to comprehend more intricate aspects of your message. ReLU, helps the chatbot understand your messages better. It's like putting on glasses to see things more clearly. activation functions are crucial because they add flexibility and non-linearity to the network's decision-making process, allowing it to solve more challenging tasks like recognizing objects in images or understanding language in chatbots. If the input to a neuron is positive, ReLU lets it pass through unchanged, like turning on the lamp. But if the input is negative, ReLU blocks it, setting the output to zero, like turning off the lamp. This simple function helps neural networks learn faster and more effectively by introducing non-linearity, making them better at recognizing patterns and solving complex problems like image recognition or language understanding.

```
self.relu = nn.ReLU()
```

$$a^{(l)} = \mathrm{ReLU}(z^{(l)})$$

‘a’is the activation output from the layer.

ReLU : f(x)=max(x,0)

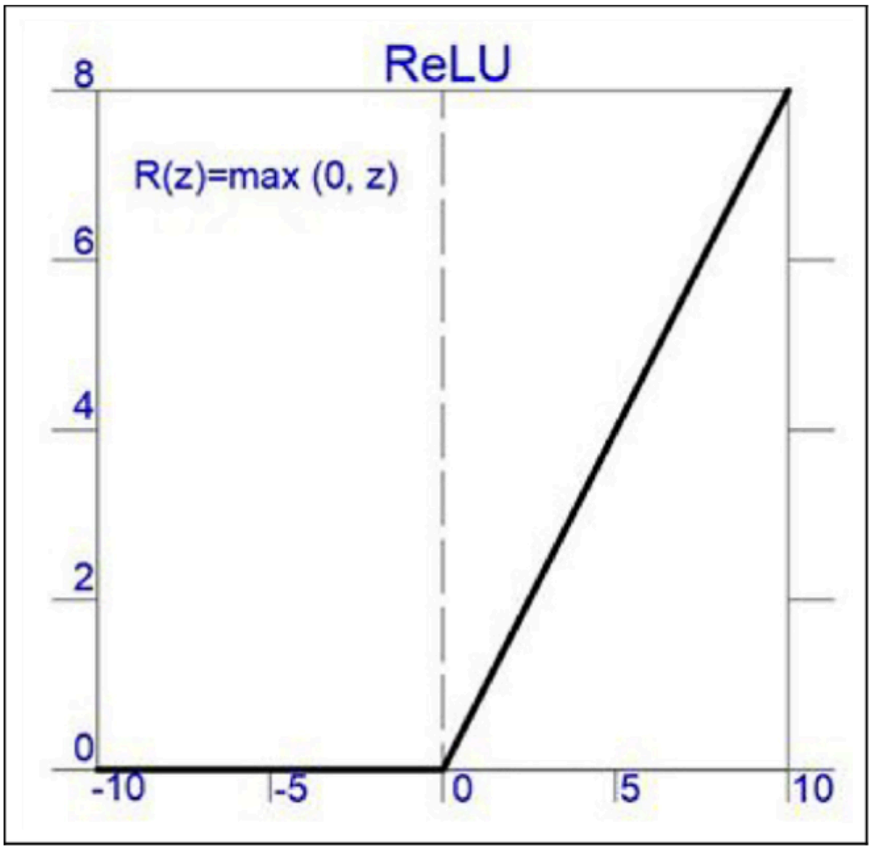


FIG 14 :- Rectified Linear Unit

THE OUTPUT LAYER :

As the final stage of processing, the output layer synthesizes the insights gleaned from the hidden layers and generates the chatbot's responses. Here, the network provides its interpretation of the user's input, ready to engage in dialogue or perform tasks.

```
def forward(self, x):
    out = self.l1(x)
    out = self.relu(out)
    out = self.l2(out)
    out = self.relu(out)
    out = self.l3(out)
```

FORWARD PROPAGATION

Each layer is applied sequentially to the input data x. The input data first goes through the first linear layer , where it gets transformed using learned weights and biases. Then, the ReLU activation function is applied, which adds non-linearity to the transformation, helping the network capture complex patterns in the data. The processed data then goes through the second linear layer and ReLU activation function again. Finally, the data passes through the output layer), where it gets transformed into the final output. In this specific implementation, there's no additional activation function or softmax applied at the end, which means the output values are directly obtained from the output layer without further processing.

WEIGHTS , BIASES AND DEFINED PARAMETERS

WEIGHTS : Parameters associated with the connections between neurons in the neural network's layers. These weights are represented as matrices

BIASES : Parameters added to each neuron in the network, allowing for additional flexibility in modeling the data. These biases are represented as vectors.

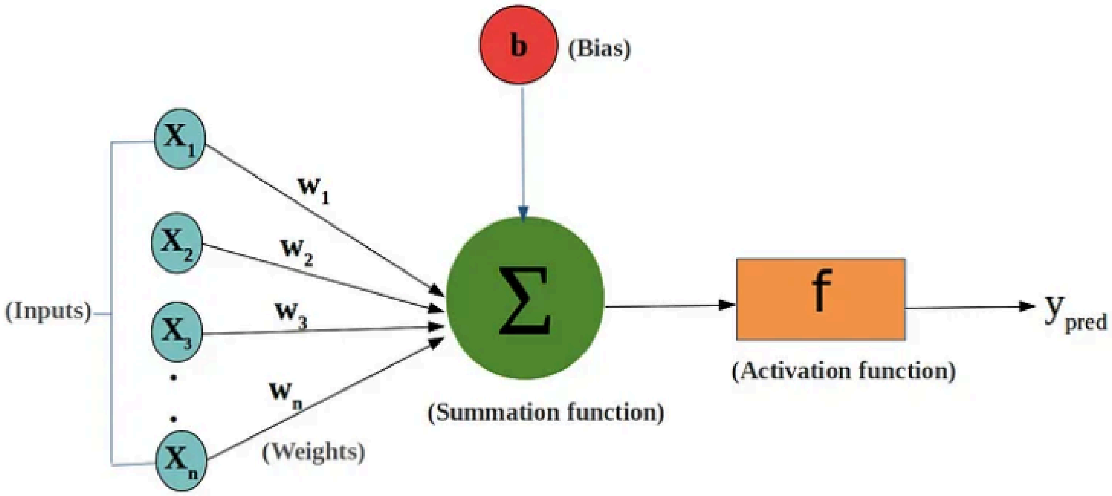


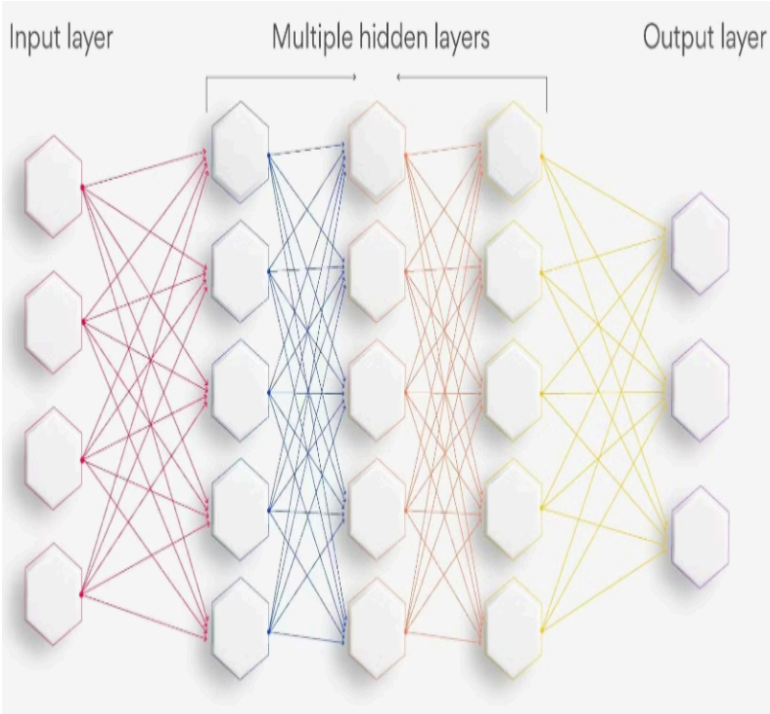
FIG 15 :- Weights and Biases in a neural Network

weights and biases are internal parameters learned by the neural network during the training process to minimize the loss function and optimize performance on the given task.

DEFINED PARAMETERS :

```
def __init__(self, input_size, hidden_size, num_classes):
```

- Input size : In the context of neural networks, it corresponds to the number of neurons in the input layer.
- Hidden Size : This parameter defines the number of neurons in the hidden layer(s) of the neural network. It determines the capacity or complexity of the model to learn and represent patterns in the data. Increasing the hidden size can potentially lead to a more expressive model but may also increase the risk of overfitting.
- Output size : This parameter indicates the number of neurons in the output layer of the neural network. It corresponds to the **number of classes** or categories in a classification task. For example, in a binary classification task, output_size would typically be set to 2, representing two output neurons corresponding to the two classes.



--->

Flow of Information

FIG 16(a) :- Feed Foward Neural Network, layers and forward propagation

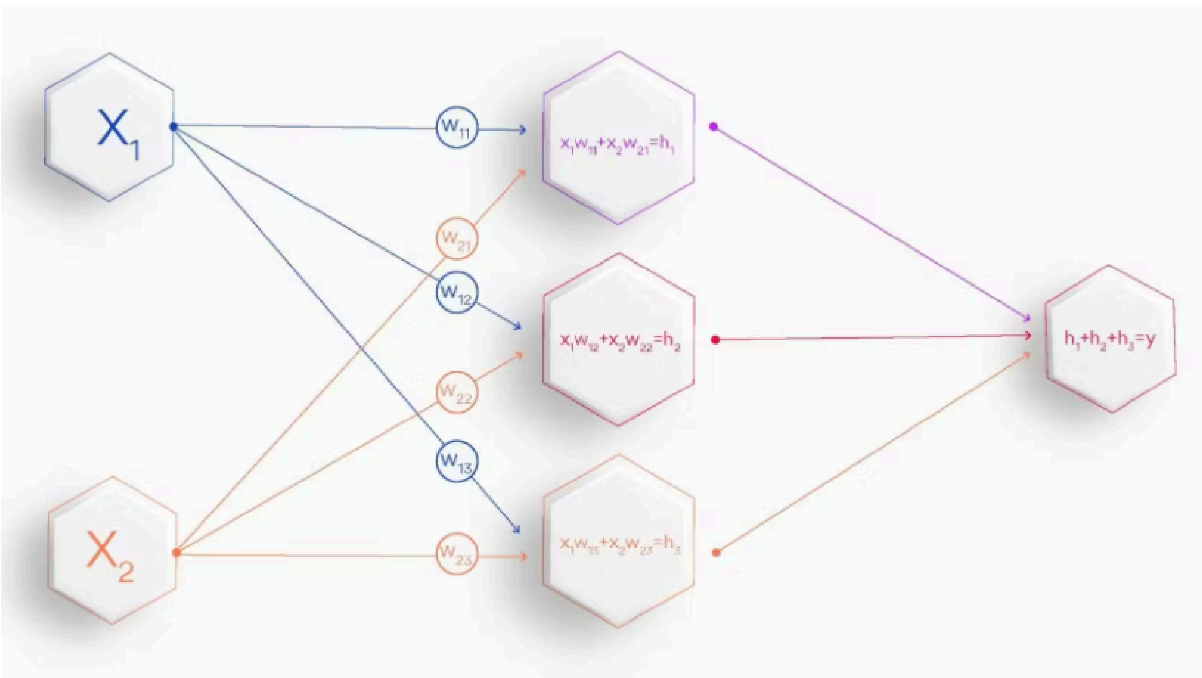


FIG 16(b) :- FORWARD PROPAGATION WITH INCLUSION OF WEIGHTS

6.1.2 TRAINING THE MODEL

In the training process of a neural network for a natural language processing task, several key steps are involved, which can be summarized as follows:

1) Loss Function Selection: The chosen loss function, in this case, **Cross Entropy Loss,** quantifies how well the model's predictions match the actual labels. It calculates the disparity between the predicted class probabilities and the true class labels.

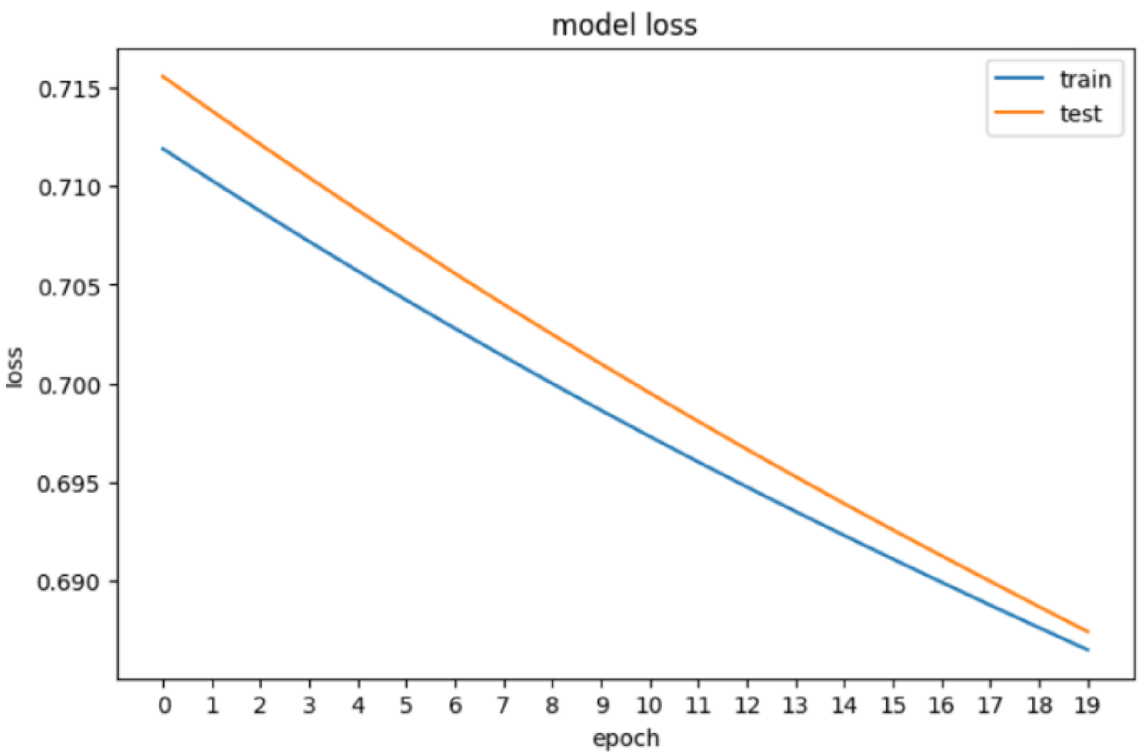


FIG 17 :- PLOT OF A SAMPLE LOSS OF A MODEL

cross-entropy loss measures the difference between two probability distributions: the predicted probabilities outputted by the model and the actual distribution of the data. It quantifies how well the predicted probabilities match the true distribution of the data.

Probability Distribution: In classification tasks, the model outputs a probability distribution over the classes for each input. For example, if you have three classes (e.g., cat, dog, bird), the model might output probabilities like [0.8, 0.1, 0.1], indicating an 80% chance of the input belonging to the "cat" class, and so on.

True Distribution: The true distribution represents the actual classes of the data. In supervised learning, you have the ground truth labels, which are often one-hot encoded. For example, if an input belongs to the "cat" class, the true distribution would be [1, 0, 0].

Loss Calculation: Cross-entropy loss compares these two distributions. It penalizes the model more heavily when it predicts low probabilities for the true class and less heavily when it predicts high probabilities for the true class. The formula for cross-entropy loss is typically:

$$-\sum_i y_i \cdot \log(p_i)$$

Where:

- *'yi'* is the true probability distribution (often one-hot encoded).
- *'pi'* is the predicted probability distribution outputted by the model.

Minimization Objective: During training, the goal is to minimize this loss. By adjusting the model's parameters (weights and biases) through techniques like gradient descent, the model learns to make better predictions and minimize the cross-entropy loss.

```
criterion = nn.CrossEntropyLoss()
```

```
loss = criterion(outputs, labels)
```

2) Optimizer Configuration: The optimizer, here Adam optimizer, determines how the model parameters are updated based on the gradients of the loss function. Adam optimizer adapts the learning rate for each parameter individually, aiming to converge faster and more effectively.

The Adam optimizer is like a smart assistant for training a machine learning model, especially in deep learning. It's a type of optimization algorithm that helps adjust the model's parameters (like weights and biases) to make it better at its task.

Here's how it works in simpler terms:

Combining Momentum and RMSProp: Adam combines the concepts of momentum and RMSProp. Momentum helps the optimizer to keep moving in the right direction by accumulating gradients from past steps, while RMSProp adjusts the learning rate for each parameter based on the magnitude of its gradients

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

Expression for first moment estimate, $\beta 1$ is the exponential decay rate for the first moment estimates.

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (g_t)^2$$

$\beta 2$ is the exponential decay rate for the second moment estimates.

Adaptive Learning Rates: Adam dynamically adjusts the learning rate for each parameter individually. It scales the learning rates based on the past gradients, allowing it to learn quickly for parameters with sparse gradients and more cautiously for parameters with frequent updates.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$$

Here,

θ_t represents the parameters being optimized,

η is the learning rate, and

ϵ is a small constant

Bias Correction: Adam includes a bias correction mechanism to compensate for the fact that it initializes the moving averages with zeros. This helps to counteract the effect of initialization biases, especially at the beginning of training.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

These corrected moments help in alleviating the bias introduced by the initialization at timestep t=0.

Efficient Convergence: By leveraging these techniques, Adam tends to converge faster and more efficiently compared to traditional gradient descent methods. It navigates the optimization landscape effectively, adjusting the learning rates based on the gradients' history and magnitudes.

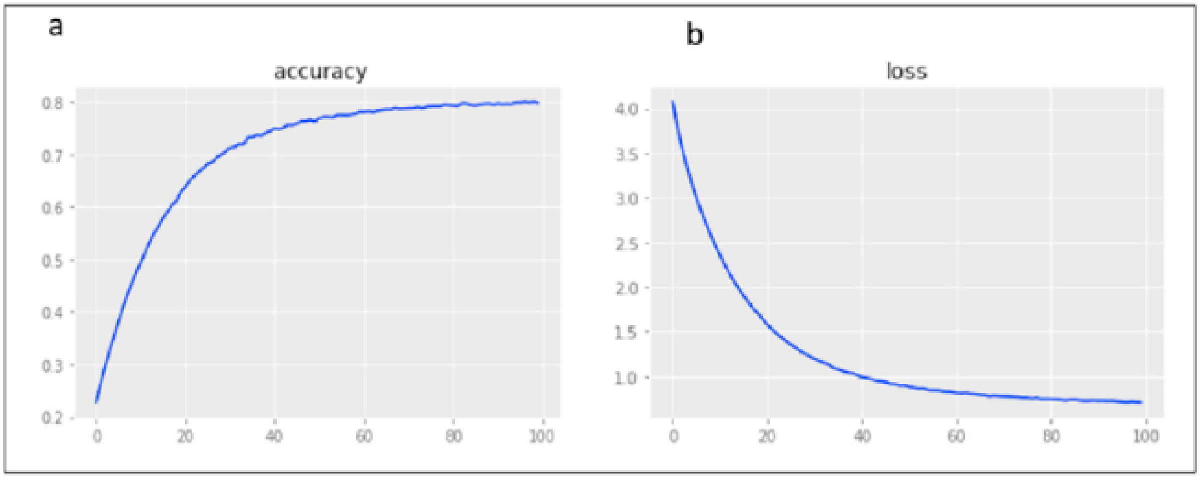


FIG 18 :- a) ACCURACY GRAPH FOR ADAM OPTIMIZER , b) LOSS GRAPH FOR ADAM OPTIMIZER

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

3) Training Loop: This loop iterates over the dataset multiple times (epochs), with each iteration adjusting the model's parameters to improve its performance.

4) Data Preparation: Data is loaded in batches (words and their corresponding labels), which enhances efficiency and allows for parallel processing.

5) Forward Pass: During each iteration, the input data (words) is fed into the model, producing predictions (outputs) for each instance.

```
# Forward pass
outputs = model(words)
# if y would be one-hot, we must apply
# labels = torch.max(labels, 1)[1]
loss = criterion(outputs, labels)
```

6) Backpropagation: The gradients of the loss function with respect to the model parameters are computed. This step determines the direction and magnitude of parameter updates necessary to minimize the loss

Backpropagation is the core algorithm used to train neural networks. It works by iteratively adjusting the weights and biases of the network to minimize the difference between the predicted outputs and the actual targets. During the forward pass, input data is fed through the network, producing predictions. These predictions are compared to the true targets to compute a loss function, which quantifies the network's performance. In the backward pass, gradients of the loss function with respect to each parameter in the network are computed using the chain rule of calculus. These gradients indicate how much each parameter contributed to the overall error, guiding adjustments to minimize the error. Through this process of propagating gradients backward through the network, the model learns to update its parameters in a way that improves its predictive accuracy. By iteratively repeating this process over multiple training examples, the network gradually learns to make better predictions, ultimately achieving higher performance on the given task.

```
# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

The backpropagation algorithm used in the project is a simplified version of the standard backpropagation algorithm.

Forward Pass: In each training iteration (looping through the dataset for a specified number of epochs), the input data is passed forward through the neural network model to obtain predictions. These predictions are compared with the actual labels to compute the loss using the specified loss function.

Backward Pass: After calculating the loss, the method, backward() is called on the loss tensor. This triggers the computation of gradients of the loss with respect to the model's parameters (weights and biases) using automatic differentiation. These gradients represent the direction and magnitude of adjustments needed to minimize the loss.

Parameter Update: The optimizer is then used to update the model's parameters based on the computed gradients.

Monitoring Training Progress: Optionally, the code prints out the loss at certain intervals to monitor the training progress.

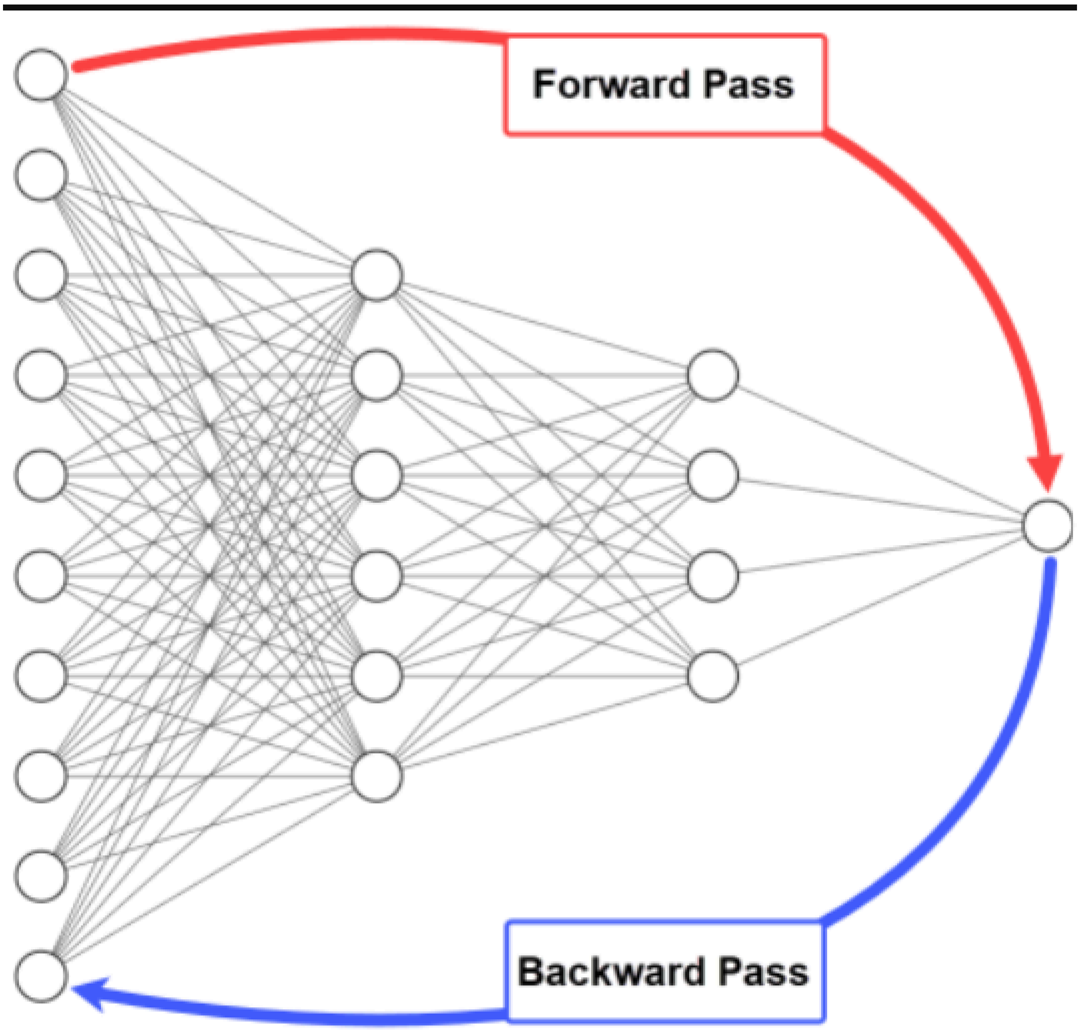


FIG 19 :- FOWARD PASS VS BACKWARD PASS IN NEURAL NETWORKS

7) Epoch Reporting: Optionally, the training process may periodically report the loss to monitor the model's progress. This can help in identifying whether the model is learning and if adjustments to the hyperparameters or architecture are necessary.

```
if (epoch+1) % 100 == 0:
    print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

6.2 MODEL EVALUATION

While training the model, which was built from scratch in Jupyter notebook, with the self-made dataset, intents.json, the model performed really well, where it took 500 epochs to produce 0 training loss and a perfect accuracy score of 0.9921, indicating good performance, however it does leave a scope for overfitting.

```
Epoch [50/500], Loss: 1.6029, Accuracy: 0.6951
Epoch [100/500], Loss: 1.3410, Accuracy: 0.8784
Epoch [150/500], Loss: 0.0027, Accuracy: 0.9321
Epoch [200/500], Loss: 0.4474, Accuracy: 0.9731
Epoch [250/500], Loss: 0.0613, Accuracy: 0.9905
Epoch [300/500], Loss: 0.0841, Accuracy: 0.9921
Epoch [350/500], Loss: 0.0005, Accuracy: 0.9921
Epoch [400/500], Loss: 0.1606, Accuracy: 0.9921
Epoch [450/500], Loss: 0.0016, Accuracy: 0.9921
Epoch [500/500], Loss: 0.0000, Accuracy: 0.9921
```

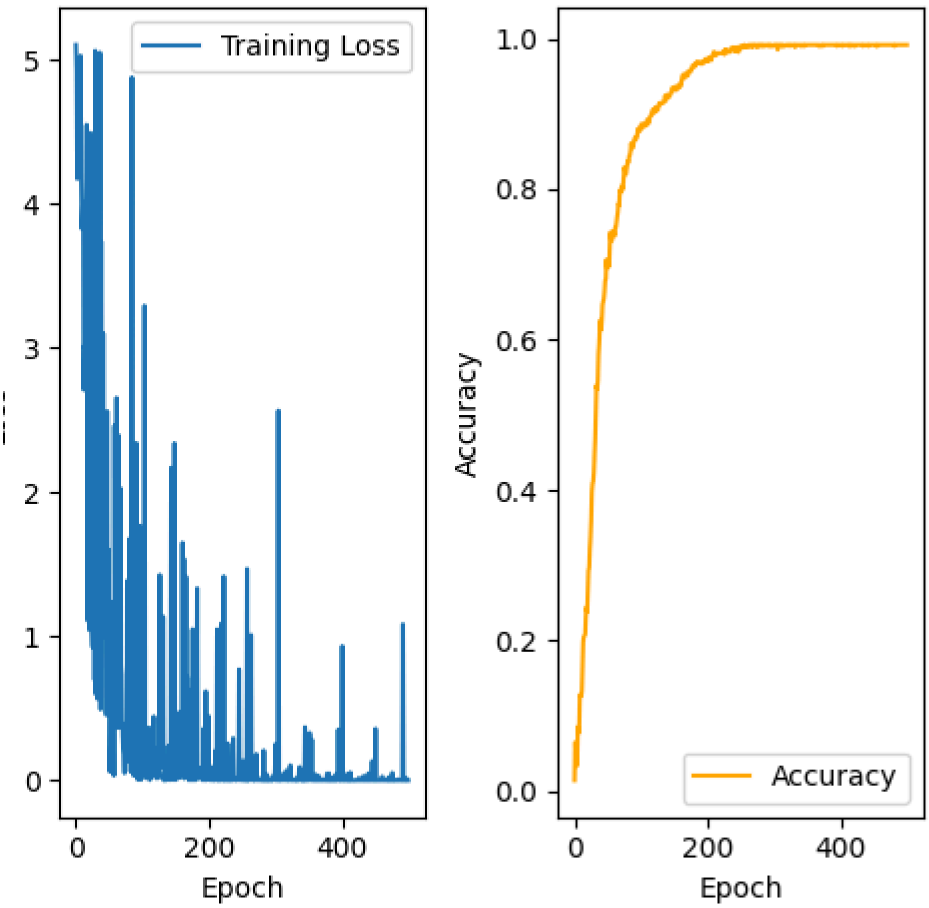


FIG 20 :- a) TRAINING LOSS GRAPH ON 'intents.json' , b) ACCURACY GRAPH ON 'intents.json'

Therefore, to ensure consistency of the model, a couple of sample test datasets were input into the model for evaluation.

This is how the model performed on unseen data sample 1:

```
Epoch [50/500], Loss: 2.8839, Accuracy: 0.1932
Epoch [100/500], Loss: 2.1246, Accuracy: 0.6364
Epoch [150/500], Loss: 0.3977, Accuracy: 0.9545
Epoch [200/500], Loss: 0.6278, Accuracy: 0.9886
Epoch [250/500], Loss: 0.1773, Accuracy: 0.9886
Epoch [300/500], Loss: 0.0436, Accuracy: 0.9886
Epoch [350/500], Loss: 0.0296, Accuracy: 0.9886
Epoch [400/500], Loss: 0.0362, Accuracy: 0.9886
Epoch [450/500], Loss: 0.1194, Accuracy: 0.9886
Epoch [500/500], Loss: 0.0072, Accuracy: 0.9886
```

Under the same parameters and hyper parameters, the loss wasn't zero, but very close to negligible, however the accuracy was fairly consistent, with a score of 0. 9886

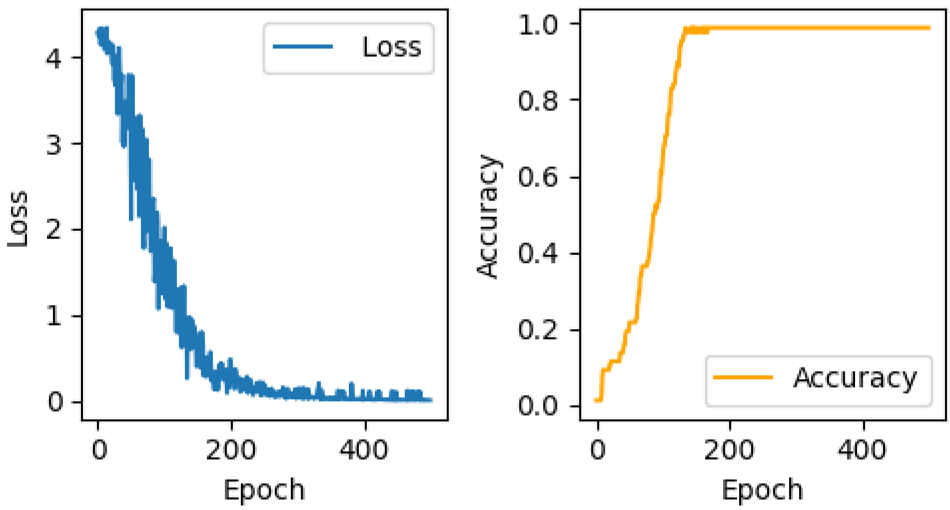


FIG 21 :- a) TRAINING LOSS GRAPH ON unseen dataset 1 , b) ACCURACY GRAPH ON unseen dataset 1

However, in terms of unknown data sample 2, the model yielded these results

```
Epoch [50/500], Loss: 3.2174, Accuracy: 0.5366
Epoch [100/500], Loss: 0.9563, Accuracy: 0.8298
Epoch [150/500], Loss: 1.1080, Accuracy: 0.9188
Epoch [200/500], Loss: 0.4904, Accuracy: 0.9476
Epoch [250/500], Loss: 0.5804, Accuracy: 0.9607
Epoch [300/500], Loss: 0.0011, Accuracy: 0.9843
Epoch [350/500], Loss: 0.0003, Accuracy: 0.9895
Epoch [400/500], Loss: 0.0022, Accuracy: 0.9895
Epoch [450/500], Loss: 0.0004, Accuracy: 0.9895
Epoch [500/500], Loss: 0.0010, Accuracy: 0.9895
```

That is a loss of 0.0010, which is not zero but indicates a very good performance by the model, backed by the consistent accuracy score of 0.9895. This test was conducted under the same parameters and hyperparameters as the original training.

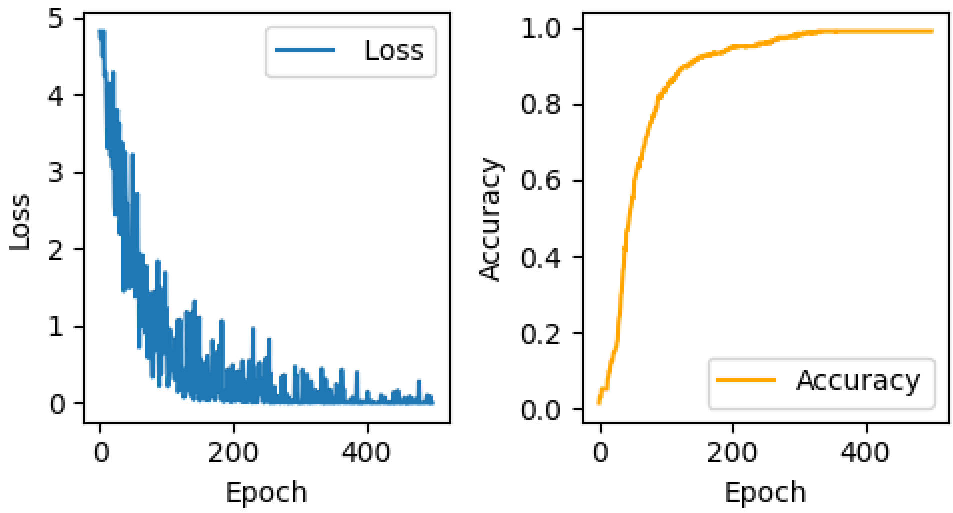


FIG 22 :- a) TRAINING LOSS GRAPH ON unseen dataset 2 , b) ACCURACY GRAPH ON unseen dataset 2

7. INTERFACE

7.1 COMMAND PROMPT BASED

The command-line-based interface provides users with a simple yet effective way to interact with the chatbot directly from their terminal or command-line environment. This interface provides a lightweight and simple interface, making it easier for users to interact with chatbots without the need for complex graphical user interfaces (GUIs) or specialized applications

In our implementation, the execution of commands follows a direct link:

Initialization: Once the chatbot script is started, the user is greeted with a welcome message indicating the beginning of the conversation. They are sent a message asking if they can type 'Stop' to exit the chatbot session.

User Input: The chatbot waits for user's input. Users can mimic natural conversations and type their messages directly into the command prompt.

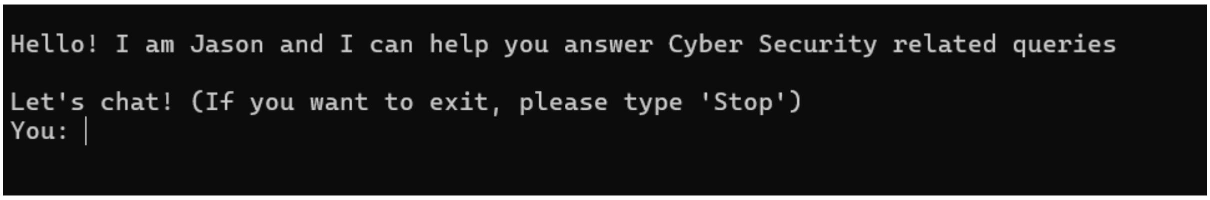


FIG 24(a) :- Chatbot Greeting Message

Response generation: Based on the predicted opinion and the confidence score associated with the prediction, the chatbot selects the appropriate response from the dataset and if confidence score exceeds a certain threshold which in our case is 75%, the chatbot sends a message it answers about it. If not, it means it doesn't understand the user's input.

Output Display: The chatbot displays its responses directly at the command prompt, allowing users to view generated messages without switching to another interface or Application

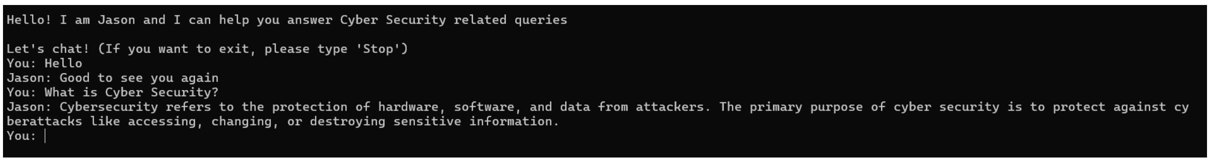


FIG 24(b) :- Chatbot Interaction on Command Prompt

Continuous communication: The conversation continues iteratively, and the chatbot waits for further information from the user. This communication cycle repeats until the user decides to exit the session by typing 'quit'.

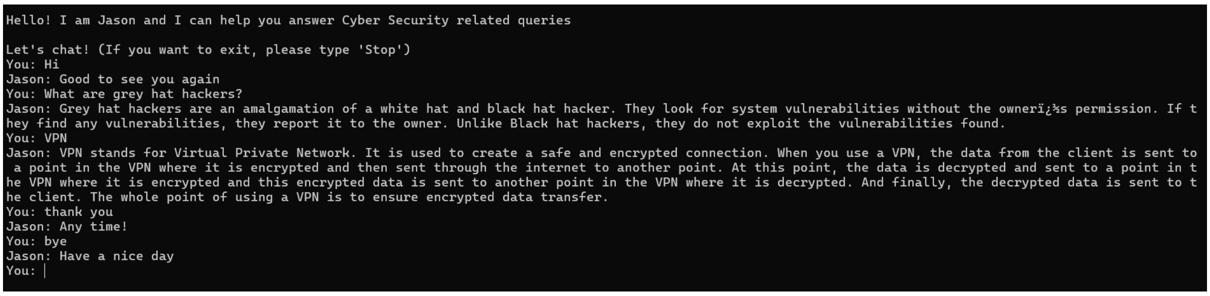


FIG 24(c) :- Overall Command Prompt based Interface

7.2 STREAMLIT BASED INTERFACE

The implementation begins with the **importation of essential libraries** required for the development of the chatbot system. These libraries include Streamlit, utilized for creating the user interface, as well as random, json, and torch, which are essential for various functionalities within the system.

Following the library imports, **the system proceeds to load crucial data** and a **pre-trained neural network model.** Specifically, the intents of the chatbot are loaded from a JSON file named **'intents.json'**. Additionally, a pre-trained neural network model, along with associated data such as input size, hidden size, output size, and vocabulary, are loaded from a file named **'data.pth'**.

Once the necessary data and model have been loaded, the system **initializes the user interface using Streamlit**, presenting users with a text input field to interact with the chatbot.

FIG 25(a) :- Streamlit based chatbot interface

Upon receiving user input, the system processes the input message. If the input message is **"exit"**, indicating the user's intention to terminate the chat session, the system promptly terminates. Otherwise, **the user's input message is displayed within the chat interface.**

Subsequently, the system tokenizes the user's input sentence and converts it into a bag-of-words representation. This representation is then fed into the pre-trained neural network model to generate a prediction.

Based on the model's prediction, the system generates a response. If the model's confidence level exceeds a predefined threshold of 0.75, a response is selected from the predefined intents associated with the predicted tag. Otherwise, a default response indicating a lack of understanding is provided.

Finally, the generated bot response is displayed within the chat interface, allowing users to engage in conversational interactions with the chatbot system.

FIG 25(b) :- Streamlit based chatbot interface, response generation

FIG 25(c) :- Streamlit based chatbot interface, response to out of scope queries

FIG 25(d) :- Streamlit based chatbot interface, exit condition

8. TESTING

8.1 UNIT TESTING

THE PREPROCESSING UNIT

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)

This code snippet was used to check if the preprocessing module correctly identifies the patterns and tags from ‘intents.json’

The result :

```
696 patterns
138 tags: ['AES', 'ARP', 'Antivirus Software', 'Biometric Authentication', 'Blockchain Technology', 'Botnet', 'Cognitive Cybersecurity', 'Common Threat
s', 'Cryptography', 'Cyber_insurance', 'Cyber_security_frameworks', 'Cybersecurity Measures', 'DDoS Attack', 'DES', 'DNS Security', 'Data Breach', 'Data
Exfiltration', 'Data Loss Prevention', 'Data Masking', 'Data_encryption', 'Decryption', 'Detecting Computer Hacks', 'Diffie-Hellman', 'Encryption', 'Enc
ryption Key Management', 'Encryption different from Hashing', 'Endpoint Security', 'Endpoint_security', 'Firewall', 'Forward Secrecy', 'General Informat
ion', 'Grey hat hackers', 'Hashing', 'Home Network Security', 'Identity', 'Identity_theft_protection', 'Incident_response', 'Insider Threats', 'Intrusio
n_detection_system', 'IoT Device Security', 'Malware', 'Mitigating Insider Threats', 'Mobile Device Security', 'Multi-Factor Authentication', 'Multipart
ite Virus', 'Network Segmentation', 'Network_security', 'Penetration Testing', 'Phishing', 'RSA', 'Ransomware', 'Rootkit', 'SIEM', 'Secure Coding Practi
ces', 'Secure Sockets Layers', 'Security Architecture', 'Security Audit', 'Security Baseline', 'Security Boundary', 'Security Certificate', 'Security Cl
earance', 'Security Compliance', 'Security Control', 'Security Directive', 'Security Exception', 'Security Framework', 'Security Guideline', 'Security I
ncident', 'Security Incident Definition', 'Security Incident Reporting', 'Security Incident Response Plan', 'Security Measure', 'Security Policy', 'Secu
rity Posture', 'Security Procedure', 'Security Risk Assessment', 'Security Standard', 'Security Through Obscurity', 'Security Token', 'Security Token Se
rvice', 'Security Update', 'Security_awareness_training', 'Social Engineering', 'Transport Layer Security', 'Trojan Horse', 'Two-Factor Authentication',
'Understanding Security Breaches', 'Understanding Security Risks', 'VPN', 'Vulnerability Assessment', 'Vulnerability_management', 'White hat hackers',
'Wireless Network Security', 'Zero-day Exploits', 'activity', 'age', 'appreciate', 'black hat hackers', 'boot sector virus', 'contact', 'covid19', 'cric
ket', 'cybersecurity', 'datetime', 'direct action virus', 'encryption', 'exclaim', 'goodbye', 'google', 'greeting', 'greetreply', 'haha', 'hours', 'insp
ire', 'insult', 'jokes', 'location', 'metamorphic virus', 'news', 'nicetty', 'no', 'noanswer', 'options', 'payments', 'phishing', 'polymorphic virus',
'programmer', 'resident virus', 'riddle', 'services', 'song', 'suggest', 'thanks', 'timer', 'traceroute', 'training', 'weather', 'whatsup']
319 unique stemmed words: ["'m", "'s", '(', ')', ',', '10', '19', '2amdl', 'a', 'accept', 'action', 'address', 'ae', 'against', 'age', 'algorithm', 'a
m', 'an', 'analysi', 'and', 'antiviru', 'anyon', 'architectur', 'are', 'arp', 'ask', 'assess', 'at', 'attack', 'audit', 'authent', 'awar', 'awesom', 'ba
d', 'baselin', 'bbye', 'be', 'been', 'benefit', 'best', 'biometr', 'black', 'blockchain', 'bonet', 'boot', 'botnet', 'boundari', 'breach', 'by', 'bye',
'can', 'card', 'cash', 'certif', 'clearanc', 'code', 'cognit', 'common', 'compani', 'complianc', 'compon', 'comput', 'conduct', 'consider', 'contact',
'control', 'could', 'cover', 'covid', 'creat', 'creator', 'credit', 'cricket', 'cryptographi', 'current', 'cyber', 'cybersecur', 'data', 'date', 'day',
'ddo', 'de', 'decrypt', 'deliv', 'design', 'detect', 'develop', 'devic', 'differ', 'diffie-hellman', 'direct', 'dn', 'do', 'doe', 'dumb', 'effect', 'ema
il', 'encrypt', 'endpoint', 'engin', 'enhanc', 'ethic', 'event', 'except', 'exchang', 'exfiltr', 'explain', 'exploit', 'fine', 'firewal', 'for', 'forwar
d', 'framework', 'from', 'funni', 'get', 'good', 'goodby', 'googl', 'great', 'grey', 'guidelin', 'ha', 'hack', 'hacker', 'haha', 'handl', 'hash', 'hat',
'hello', 'help', 'hey', 'hi', 'hola', 'home', 'hors', 'hot', 'hour', 'how', 'i', 'id', 'ident', 'identifi', 'idiot', 'if', 'implement', 'import', 'impro
v', 'in', 'incid', 'india', 'industri', 'inform', 'insid', 'inspir', 'insur', 'internet', 'intrus', 'iot', 'is', 'it', 'joke', 'key', 'know', 'later',
'latest', 'laugh', 'lmao', 'locat', 'lol', 'loss', 'lost', 'made', 'make', 'malwar', 'manag', 'mask', 'mastercard', 'match', 'me', 'mean', 'measur', 'me
tamorph', 'mitig', 'mobil', 'motiv', 'multi-factor', 'multipartit', 'my', 'namast', 'network', 'news', 'next', 'nice', 'no', 'nope', 'obscur', 'of', 'of
fer', 'ok', 'old', 'on', 'onli', 'open', 'password', 'penetr', 'phish', 'plan', 'polici', 'polymorph', 'popular', 'postur', 'practic', 'prevent', 'proce
dur', 'program', 'programm', 'protect', 'provid', 'purchas', 'question', 'ransomwar', 'report', 'resid', 'respons', 'riddl', 'risk', 'rofl', 'role', 'ro
otkit', 'rsa', 'scan', 'score', 'search', 'secreci', 'sector', 'secur', 'see', 'segment', 'servic', 'set', 'shut', 'siem', 'situat', 'social', 'softwa
r', 'song', 'ssl', 'standard', 'step', 'stolen', 'strong', 'suggest', 'summer', 'sup', 'support', 'system', 'take', 'talk', 'technolog', 'tell', 'temper
atur', 'ten', 'test', 'thank', 'that', 'the', 'theft', 'there', 'thing', 'threat', 'through', 'till', 'time', 'timer', 'tl', 'to', 'today', 'token', 'to
ol', 'top', 'topic', 'tracerout', 'train', 'trojan', 'two-factor', 'type', 'up', 'updat', 'upto', 'us', 'use', 'useless', 'va', 'viru', 'vpn', 'vulner',
'wa', 'wazzup', 'we', 'weather', 'webinar', 'were', 'what', 'when', 'where', 'whi', 'which', 'white', 'who', 'wireless', 'work', 'workshop', 'yeah', 'y
o', 'you', 'your', 'yoyooyo', 'zero-day']
```

We can conclude that the preprocessing module successfully performs the intended tasks.

THE TRAINING SET PREPARATION UNIT

X_train = np.array(X_train)

y_train = np.array(y_train)

Here, the 'X_train' numpy array consists of the BoW representation of the sentences

for i in X_train:
   print(i)

And here's a small snippet of the output :

```
[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0.]
```

print(y_train)

class labels or tags for the corresponding documents represented by the bag of words in X_train are contained in 'y_train'

Small snippet of the output :

```
[109 109 109 109 109 109 109 109 109 109 107 107 107 107 107 107 107 132
 132 132 132 132 132 112 112 112 112 112 112 112 112 112 112 116 116 116
 116 116 116 116 116 116 116 116 116 123 123 123 123 123 123 123 123 129
 129 129 129 129 129 129 129 135 135 135 135 135 135 135 135   8   8   8
   8 124 124 124 124 124 124  40  40  40  40  40  40  40  40  19  19  19
  19  19  19  46  46  46  46  46  46  36  36  36  36  36  36  36  35  35
  35  35  35  35  38  38  38  38  38  38  90  90  90  90  90  90  27  27
  27  27  27  27  10  10  10  10  10  10  81  81  81  81  81  81   9   9
   9   9   9   9  28  28  28  28  28  28  47  47  47  47  47  47  89  89
  89  89  89  89 134 134 134 134  88  88  88  88  88  88  97  97  97  97
  91  91  91  91  31  31  31  31   1   1   1   1   1   1   5   5   5   5
  54  54  54  54  54  54  83  83  83  83  83  83   6   6   6   6   6   6
```

Hence, we can conclude that the training data preparation unit functions successfully.

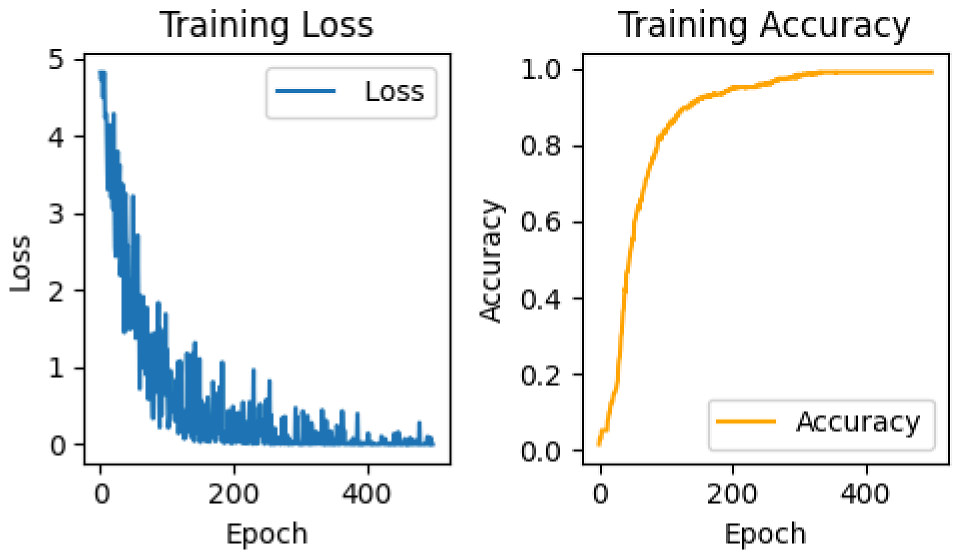
THE MODEL UNIT

From section, 6.2 of this documentation, i.e, ‘Model Evaluation ‘ we can conclude that the units responsible for the construction and training of the model have functioned successfully.

```
    if (epoch+1) % 50 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}, Accuracy:
{train_accuracy:.4f}')

# Plot training loss and accuracy
plt.figure(figsize=(5, 3))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label=' Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Accuracy', color='orange')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

The Output:

```
Epoch [50/500], Loss: 3.2174, Accuracy: 0.5366
Epoch [100/500], Loss: 0.9563, Accuracy: 0.8298
Epoch [150/500], Loss: 1.1080, Accuracy: 0.9188
Epoch [200/500], Loss: 0.4904, Accuracy: 0.9476
Epoch [250/500], Loss: 0.5804, Accuracy: 0.9607
Epoch [300/500], Loss: 0.0011, Accuracy: 0.9843
Epoch [350/500], Loss: 0.0003, Accuracy: 0.9895
Epoch [400/500], Loss: 0.0022, Accuracy: 0.9895
Epoch [450/500], Loss: 0.0004, Accuracy: 0.9895
Epoch [500/500], Loss: 0.0010, Accuracy: 0.9895
```



THE INTERFACE UNIT;

We need the interface to provide a user friendly, interactive and visually appealing environment for the question-answer framework. To achieve this we used the ***STREAMLIT*** library framework, provided by python.

The interface needs to provide a input area for the user query, should incorporate the trained neural network for the back-end processing, should provide the user the proper response based on a threshold value and should provide the user an appropriate message incase the model does not understand the query.

```
FILE = "data.pth"
data = torch.load(FILE)
model_state = data["model_state"]
model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
st.title("Chat with Jason")


user_input = st.text_input("You:", "")

if st.button("Send"):
```

```
    if user_input.strip().lower() == "exit":
        st.write("Exiting chat...")
        st.stop()

    st.write(f"You: {user_input}")


output = model(X)
  _, predicted = torch.max(output, dim=1)

if prob.item() > 0.75:
    for intent in intents['intents']:
        if tag == intent["tag"]:
            bot_response = random.choice(intent['responses'])
            st.write(f"{bot_name}: {bot_response}")
else:
    st.write(f"{bot_name}: I do not understand...")
```

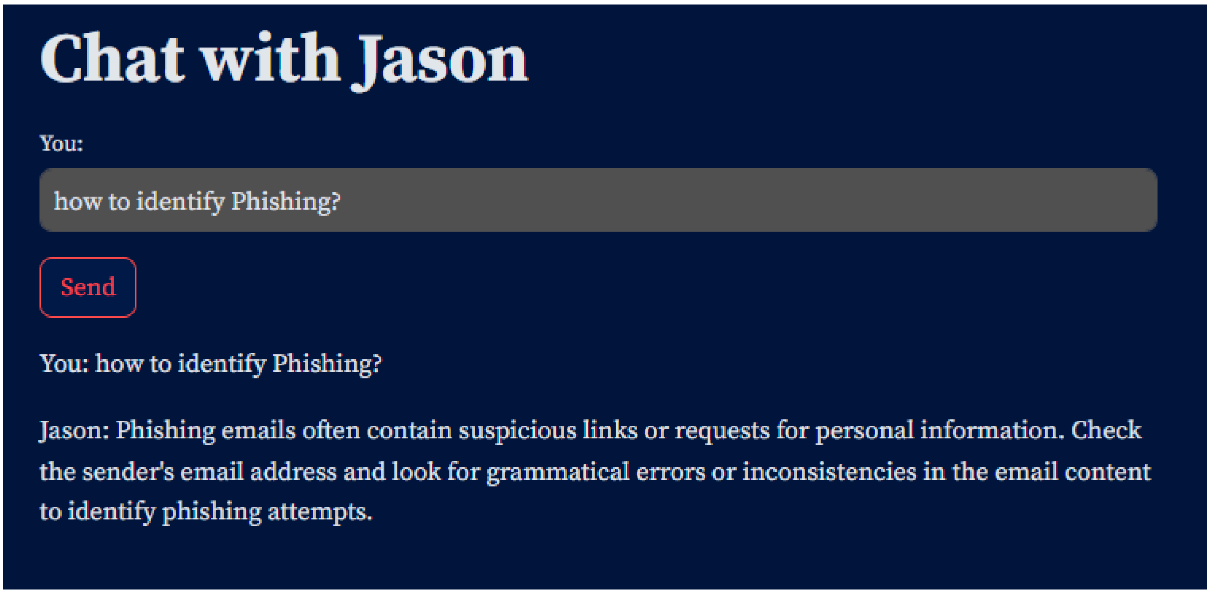
From section 7.2 of this documentation, i.e, 'Streamlit based interface' we can conclude that the interface unit functions successfully.

8.2 SYSTEM TESTING

The overall system was tested using the Streamlit based GUI, to ensure if the system produced satisfactory results, in terms of processing speed and accuracy.

The system was able to respond to various database stored queries. Also when the queries were in different formats.For example,

Moreover the system was also able to answer a different question regarding the same topic. For example,



Moreover, the responses were generated very quickly, 45 milliseconds at max. Hence, we can conclude that the system functions smoothly and successfully

9. RESULTS AND DISCUSSION

The bot is perfectly capable of providing basic natural language responses to conversational queries

```
Hello! I am Jason and I can help you answer Cyber Security r
elated queries

Let's chat! (If you want to exit, please type 'Stop')
You: Hi
Jason: Good to see you again
You: Thanks
Jason: Any time!
You: Goodbye!
Jason: Bye! Come back again soon.
You:
```

In areas where needed, the bot can provide with a different response each time (per cycle of predefined responses)

```
You: tell me a joke
Jason: I went to see the doctor about my short-term memory p
roblems â€“ the first thing he did was make me pay in advanc
e
You: tell me a joke
Jason: A perfectionist walked into a bar...apparently, the b
ar wasn't set high enough
You: tell me a joke
Jason: Never criticize someone until you've walked a mile in
 their shoes. That way, when you criticize them, they won't
be able to hear you from that far away. Plus, you'll have th
eir shoes.
You:
```

The bot can provide quick responses related to cyber security from the predefined database.

```
Hello! I am Jason and I can help you answer Cyber Security related queries

Let's chat! (If you want to exit, please type 'Stop')
You: What is Hashing?
Jason: Hashing is an algorithm performed on data such as a file or message to produce a number called a hash. The hash is used to verify that data is not mo
dified, tampered with, or corrupted. In other words, you can verify the data has maintained integrity.
You: What is VA?
Jason: Vulnerability Assessment is the process of finding flaws on the target. Here, the organization knows that their system/network has flaws or weaknesse
s and want to find these flaws and prioritize the flaws for fixing.
You:
```

The bot can respond to queries, even in different formats of the questions, as predefined at the database

```
You: botnet
Jason: A Botnet is a number of devices connected to the internet where each device has one or more bots running on it. The bots on the devices and malicious
 scripts used to hack a victim. Botnets can be used to steal data, send spams and execute a DDOS attack.
You: what is a botnet?
Jason: A Botnet is a number of devices connected to the internet where each device has one or more bots running on it. The bots on the devices and malicious
 scripts used to hack a victim. Botnets can be used to steal data, send spams and execute a DDOS attack.
You: define botnet
Jason: A Botnet is a number of devices connected to the internet where each device has one or more bots running on it. The bots on the devices and malicious
 scripts used to hack a victim. Botnets can be used to steal data, send spams and execute a DDOS attack.
You: explain botnet
Jason: A Botnet is a number of devices connected to the internet where each device has one or more bots running on it. The bots on the devices and malicious
 scripts used to hack a victim. Botnets can be used to steal data, send spams and execute a DDOS attack.
You: |
```

The bot is not case sensitive

```
You: forward secrecy
Jason: Forward Secrecy is a system that uses ephemeral session keys to do the actual encryption of TLS data so that even if the serverï¿½s private key were
to be compromised, an attacker could not use it to decrypt captured data that had been sent to that server in the past.
You: FORWARD SECRECY
Jason: Forward Secrecy is a system that uses ephemeral session keys to do the actual encryption of TLS data so that even if the serverï¿½s private key were
to be compromised, an attacker could not use it to decrypt captured data that had been sent to that server in the past.
You: foRWaRD sECrecY
Jason: Forward Secrecy is a system that uses ephemeral session keys to do the actual encryption of TLS data so that even if the serverï¿½s private key were
to be compromised, an attacker could not use it to decrypt captured data that had been sent to that server in the past.
You: |
```

The bot can respond to different types of questions around the same or various topics

```
You: MFA
Jason: Multi-factor Authentication (MFA) is an authentication method that requires the user to provide two or more verification factors to gain access to a
resource such as an application, online account, or a VPN. MFA is a core component of a strong identity and access management (IAM) policy. Rather than just
 asking for a username and password, MFA requires one or more additional verification factors, which decreases the likelihood of a successful cyber attack.
You: 2FA
Jason: Two-factor authentication adds an additional layer of security to the authentication process by making it harder for attackers to gain access to a pe
rson's devices or online accounts because, even if the victim's password is hacked, a password alone is not enough to pass the authentication check.
You: MFA vs 2FA
Jason: 2FA is a subset of MFA, All instances of 2FA are instances of MFA, Not all instances of MFA are 2FA, MFA requires more pieces of evidence than 2FA to
 grant users access, MFA is usually considered safer than 2FA as it provides the most layers of security against cybercriminals. However, the strength of an
 MFA solution depends on how secure its additional authentication methods are.
You: Phishing
Jason: Phishing is a type of cyber attack where attackers impersonate legitimate entities to trick individuals into providing sensitive information such as
passwords or financial details.
You: identify phishing
Jason: Phishing emails often contain suspicious links or requests for personal information. Check the sender's email address and look for grammatical errors
 or inconsistencies in the email content to identify phishing attempts.
```

The bot provides an appropriate response to out of scope questions

```
Hello! I am Jason and I can help you answer Cyber Security related queries

Let's chat! (If you want to exit, please type 'Stop')
You: Who was Mahatma Gandhi?
Jason: I'm sorry, but I'm having difficulty understanding as the question seems to be beyond my expertise.
You: |
```

10. CONCLUSION AND FUTURE SCOPE

In conclusion, this dissertation has focused on using Natural Language Processing (NLP) and Deep Learning techniques to develop a retrieval-based chatbot built for addressing knowledge-based queries within the domain of cybersecurity. Through the implementation of deep learning methodologies, particularly a feedforward neural network, the project aimed to enhance the chatbot's ability to understand and respond to user inquiries effectively. However, certain limitations such as the chatbot's reliance on pattern recognition, domain restriction to cybersecurity, and its lack of emotional intelligence were identified. Despite these constraints, the project showcased promising applicability across various sectors including education, corporate use, professional development, online forums, and events/workshops within the cybersecurity realm. Furthermore, the potential for extending the project's utility beyond cybersecurity into domains like customer support, healthcare, e-commerce, travel, finance, and human resources suggests avenues for future research and development, highlighting the versatility and scalability of the proposed solution. Through addressing these limitations and exploring broader applications, this project contributes to the ongoing advancement of chatbot technology, promising enhanced accessibility to information and assistance across diverse domains.

From undertaking this project we can further conclude that in the future further improvementsthis, including :

- Refine pattern recognition and contextual understanding to minimize inaccuracies and improve responses.
- Integrate emotional intelligence to enable the chatbot to recognize and respond appropriately to users' emotional cues.
- Expand domain coverage beyond cybersecurity to include other fields of knowledge.
- Train the chatbot on diverse datasets and refine its knowledge base to cater to a wider range of user inquiries.
- Leverage advancements in machine learning and natural language processing to enhance creativity in responses and adaptability to changing circumstances.
- Collaborate with industry experts and end-users to guide development and ensure alignment with real-world needs.
- Solicit continuous feedback for ongoing improvements and optimizations.
- Aim to make the chatbot an indispensable tool across various domains, including education, customer service, healthcare, e-commerce, travel, finance, and human resources.

are possible.

11. REFERENCES

[1] *Cambria E, White B, Jumping NLP curves: a review of natural language processing research. IEEE Comput Intell Mag*

[2] *Chowdhary KR, Natural Language Processing for Word Sense Disambiguation and Information Extraction. PhD Thesis, JNV University, Jodhpur (India)*

[3] *Rich Caruana. Multitask learning. Machine Learning*

[4] *Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I,Attention is all you need*

.

[5] *Yu Wu , Zhoujun Li , Wei Wu , Ming Zhou , Response selection with topic clues for retrieval-based chatbots*

[6] *Kristen Moore, Shenzun zong, A comprehensive solution to retrieval-based chatbot Construction*

[7] *Zhou X., Li L., Dong D., Liu Y., Chen Y., Zhao W.X., Yu D., Wu H. Multi-turn response selection for chatbots with deep attention matching network*

[8] *Ronen Eldan, Ohad Shamir, The Power of Depth for Feedforward Neural Networks*

[9] *Marc Moreno Lopez, Jugal Kalita, "Deep Learning applied to NLP", arXiv:1703.03091.*

[10] *Design and Development of Retrieval-Based Chatbot Using Sentence Similarity, Harita Akkineni*

[11] *A comprehensive solution to retrieval-based chatbot construction, Kristen Moore, Shenzung Jhong, Neha Jindal*

[12] *Wu, W. Wu, Z. Li, M. Zhou, Learning matching models with weak supervision for response selection in retrieval-based chatbots,*

[13] *Whang, D. Lee, C. Lee, K. Yang, D. Oh, H. Lim, An effective domain adaptive post-training method for bert in response selection*

[14] *Wang, H. Fang, An adaptive response matching network for ranking multiturn chatbot responses*

[15] *A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks [29.03.24]*

[16] *The Math behind Adam Optimizer | Towards Data Science,towardsdatascience.com [28.03.24]*

[17] *Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy | DataCamp www.datacamp.com [28.03.24]*

[18] *Understanding Feed Forward Neural Networks in Deep Learning www.turing.com [24.03.24]*

[19] *Weights and Bias in a Neural Network | Towards Data Science towardsdatascience.com [24.03.24]*

[20] *Bag of Words in NLP & Machine Learning: Examples vitalflux.com [25.03.24]*

[21] *Rectified Linear Units (ReLU) in Deep Learning www.kaggle.com [26.03.24]*

[22] *Linear Transformation to incoming data in Pytorch - GeeksforGeeks www.geeksforgeeks.org [26.03.24]*

[23] *https://www.analyticsvidhya.com/blog/2022/06/an-end-to-end-guide-on-nlp-pipeline/ [23.03.24]*

[24] *Introduction to Cyber Security, Dr. Jeetendra Pande, Assistant Professor, School of CS & IT, Uttarakhand Open University, Haldwani*