THE HISTORY OF CHATBOT:

In October of 1950, Alan Turing proposed an approach for evaluating a computer's intelligence and famously named his method, The Imitation Game. The premise is that an interrogator talks to two people through a "typewritten" machine (today we would refer to this as instant messaging). The catch is that only one of the conversations is with a real person – the other is with a computer.

Turing posited that, by the turn of the century (the year 2000), in a well-controlled experiment, a computer should be able to fool the interrogator 70% of the time. While we've made a lot of progress since 1950, no algorithm has consistently reached this bar. However, there has still been substantial progress in the field of chatbot development, which has led to a multi-billion dollar industry and dozens of profitable products.

ChatterBot

If you wish to upgrade to ChatterBot's latest development version, execute the command:

```
Pip install - -upgrade chatterbot_corpus
Pip install - -upgrade chatterbot
```

Once these steps are complete your setup will be ready, and we can start to create the Python chatbot.

1. Import classes

- The next step is to import classes. ChatterBot is a Python library designed to respond to user inputs with automated responses. It uses various machine learning (ML) algorithms to generate a variety of responses, allowing developers to build chatbots that can deliver appropriate responses in a variety of scenarios.

- Because it's built using ML algorithms, the chatbot will also be able to improve its performance as it learns over time. Chatterbot also features language independence, meaning that it can be used to train chatbots in multiple programming languages.

- A chatbot built using ChatterBot works by saving the inputs and responses it deals with, using this data to generate relevant automated responses when it receives a new input. By comparing the new input to historic data, the chatbot can select a response that is linked to the closest possible known input. Before starting, it's important to consider the storage and scalability of your chatbot's data.

- Using cloud storage solutions can provide flexibility and ensure that your chatbot can handle increasing amounts of data as it learns and interacts with users. It's also essential to plan for future growth and anticipate the storage requirements of your chatbot's conversations and training data. By leveraging cloud storage, you can easily scale your chatbot's data storage and ensure reliable access to the information it needs.

Moreover, the more interactions the chatbot engages in over time, the more historic data it has to work from, and the more accurate its responses will be.

Developing Your Own Chatbot From Scratch

Now that we're armed with some background knowledge, it's time to build our own chatbot.

1. Prepare dependencies

The first step is to install the ChatterBot library in your system. It's recommended that you use a new Python virtual environment in order to do this.

To achieve this, write and execute this command in your Python terminal:

```
pip install chatterbot
pip install chatterbot_corpus
```

Classes are code templates used for creating objects, and we're going to use them to build our chatbot.

There are two classes we'll need to download to do this: ChatBot from chatterbot, and ListTrainer from chatterbot.trainers.

To download these, execute these commands:

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
```

Code language: JavaScript (javascript)

1. Train the chatbot

The chatbot you're building will be an instance belonging to the class 'ChatBot'.

Create a new ChatterBot instance, and then you can begin training the chatbot.

Training the chatbot will help to improve its performance, giving it the ability to respond with a wider range of more relevant phrases.

You can begin by executing the following command:

```
my_bot = ChatBot(name='Chatty', read_only=True,
logic_adapters=
['chatterbot.logic.MathematicalEvaluation','chatterbot.logic.BestMatch'])
```

Code language: PHP (php)

Let's break down that command so we understand how we're training the chatbot.

'name=' refers to the name of your chatbot. **We've called this one Chatty**, but feel free to choose something more original!

If you’re planning to set up a website to give your chatbot a home, don’t forget to make sure your desired domain is available with a check domain service.

For example, if you’re developing an AI-driven chatbot for an ecommerce website, you can train it to provide product recommendations, answer customer inquiries about orders and shipping, and assist with the checkout process.

The command ‘read_only=True’ means that the chatbot’s ability to learn after this training has been disabled. This is optional, depending on the purpose of the chatbot.

The command ‘logic_adapters’ provides the list of resources that will be used to train the chatbot.

This chatbot is going to **solve mathematical problems**, so ‘chatterbot.logic.MathematicalEvaluation’ is included. This logic adapter checks statements for mathematical equations. If one is present, a response is returned containing the result.

The logic adapter ‘chatterbot.logic.BestMatch’ is used so that that chatbot is able to select a response based on the best known match to any given statement.

In order for this to work, you’ll need to provide your chatbot with a list of responses.

Here are just a few of the strings you can train your chatbot to learn:

```
animals = [“aardvark”, “badger”, “cat”, “dog”, “elephant”]
```

For clarity, you can also place each string on a separate line:

```
small_talk = [“hi!”
“how\’re you?”
“what\’s up?”
“glad to hear that”
“i\’m chatty the chatbot. Do you have a maths question?”]
```

Code language: PHP (php)

We’ll need to train it to be able to recognise and solve maths problems if it’s going to be asked them, for example:

```
math_talk_1 = [‘pythagorean theorem’,

‘a squared plus b squared equals c squared.’]
```

Your chatbot is now ready to engage in basic communication, and solve some maths problems.

1. Communicate with the chatbot

Now you can start to play around with your chatbot, communicating with it in order to see how it responds to various queries.

It’s important to remember that, at this stage, your chatbot’s training is still relatively limited, so its responses may be somewhat lacklustre.

You should take note of any particular queries that your chatbot struggles with, so that you know which areas to prioritise when it comes to training your chatbot further.

2. Train the chatbot further

Now that you’ve got an idea about which areas of conversation your chatbot needs improving in, you can train it further using an existing corpus of data.

A corpus is a collection of authentic text or audio that has been organised into datasets. There are numerous sources of data that can be used to create a corpus, including novels, newspapers, television shows, radio broadcasts, and even tweets.

ChatterBot provides some corpora that can be used for this purpose:

```
from chatterbot.trainers import ChatterBotCorpusTrainer
corpus_trainer = ChatterBotCorpusTrainer(my_bot)
corpus_trainer.train(‘chatterbot.corpus.english’)
```

Code language: JavaScript (javascript)

ChatterBot offers corpora in a variety of different languages, meaning that you'll have easy access to training materials, regardless of the purpose or intended location of your chatbot.

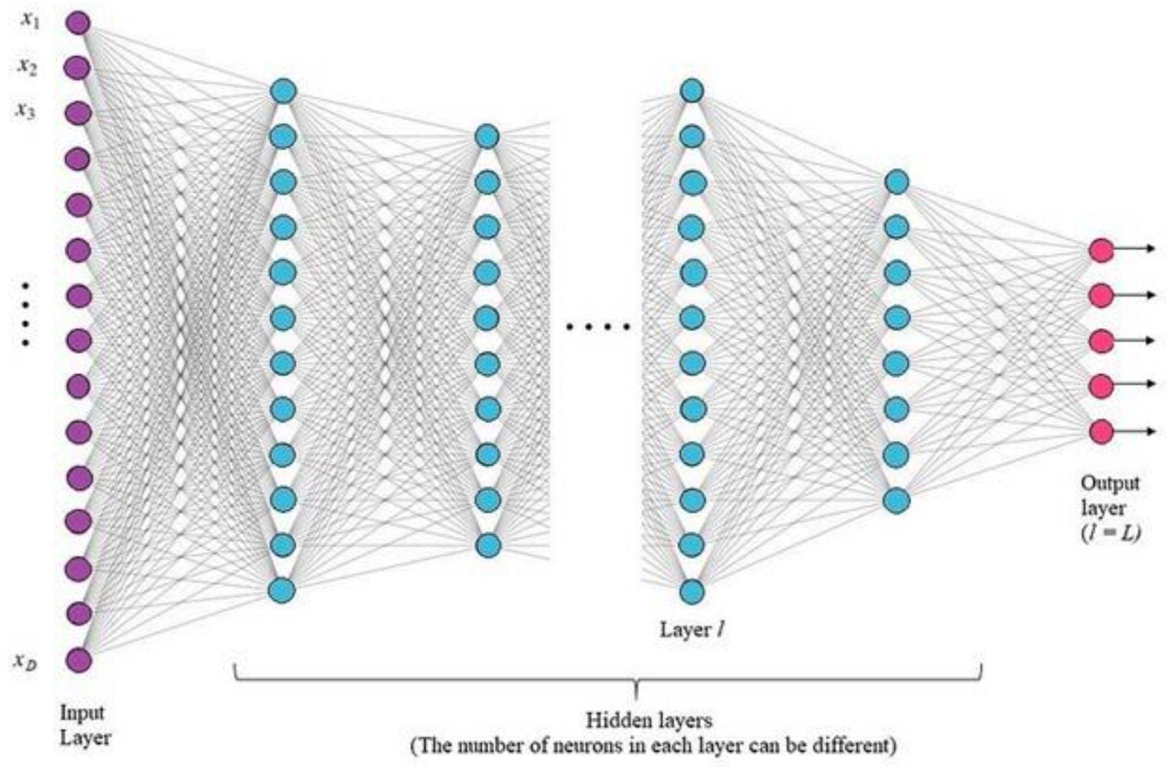
If you wish**,** you can even export a chat from a messaging platform such as WhatsApp to train your chatbot. Not only does this mean that you can train your chatbot on curated topics, but you have access to prime examples of natural language for your chatbot to learn from.

Once your chatbot is trained to your satisfaction, it should be ready to start chatting.

Don't forget to test your chatbot further if you want to be assured of its functionality, (consider using software test automation to speed the process up)

NEURAL NETWORK

It is a deep learning algorithm that resembles the way neurons in our brain process information (hence the name). It is widely used to realize the pattern between the input features and the corresponding output in a dataset. Here is the basic neural network architecture -.

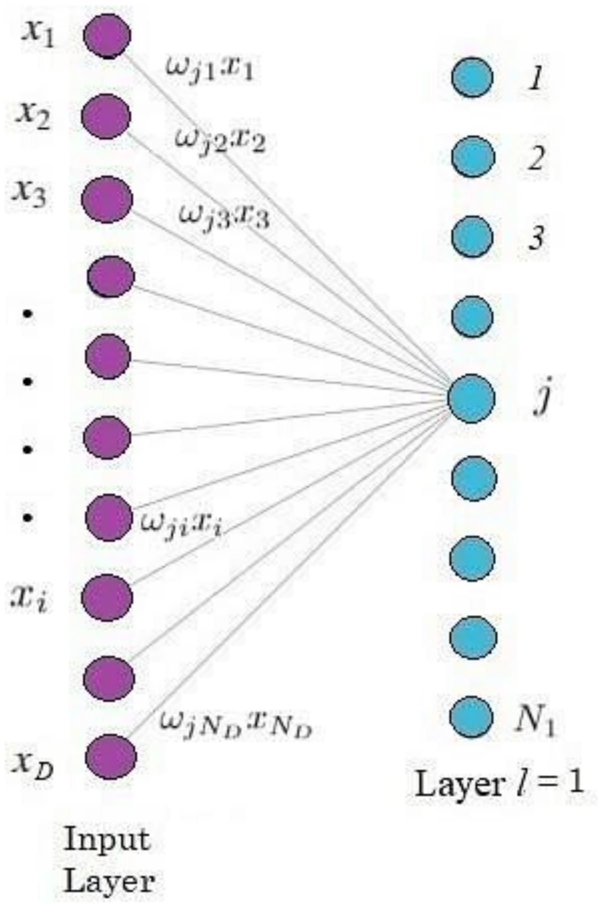


Neural Network algorithm involves two steps:

1. Forward Pass through a Feed-Forward Neural Network
2. Backpropagation of Error to train Neural Network

1. Forward Pass through a Feed-Forward Neural Network:

This step involves connecting the input layer to the output layer through a number of hidden layers. The neurons of the first layer (l=1) receive a weighted sum of elements of the input vector (xáµ¢) along with a bias term b, as shown in Fig. 2. After that, each neuron transforms this weighted sum received at the input, a, using a differentiable, nonlinear activation function h(•) to give output z.



$$z_j^{(1)} = h(a_j^{(1)}) = h(\sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + b_j^{(1)})$$

Hidden layers of neural network architecture.

The two activation functions that we will use for our ChatBot, which are also most commonly used are Rectified Linear Unit (ReLu) function and Softmax function. The former will be used for hidden layers while the latter is used for the output layer. The softmax function is usually used at the output for it gives probabilistic output. The ReLU function is defined as:

$$f(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

And the softmax function is defined as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$E = \sum_{n=1}^{N} E_n$$

Now, it’s time to move on to the second step of the algorithm that is used in building this chatbot application project.

2. Backpropagation of Error to train Neural Network

This step is the most important one because the original task of the Neural Network algorithm is to find the correct set of weights for all the layers that lead to the correct output and this step is all about finding those correct weights and biases. Consider an input vector that has been passed to the network and say, we know that it belongs to class A. Assume the output layer gives the highest value for class B. There is therefore an error in our prediction. Now, since we can only compute errors at the output, we have to propagate this error backward to learn the correct set of weights and biases.

Let us define the error function for the network:
Here Eâ‚™ is the error for a single pattern vector: xâ‚™ and is defined as,

$$E_n = \frac{1}{2}\sum_{k}(z_k^{(L)} - r_k)^2$$