UNIVERSITY OF HERTFORDSHIRE

School of Computer Science

MSc. Artificial Intelligence with Robotics.

7COM1077 and Advance Computer Science Masters Project.

Date: 11-01-2021.

EMOTION ANALYSIS USING CHATBOT

RAJESH KONANKI

STUDENT ID: 18024966

SUPERVISOR: DEEPAK PANDEY

DECLARATION

This report is submitted in partial fulfillment of the requirement for the degree of **Master of Science in Artificial Intelligence with Robotics** at **the University of Hertfordshire (UH).**

It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

RAJESH KONANKI

ID: 18024966

ACKNOWLEDGEMENT

I need to thank my boss, Prof. Deepak Panday, for their steady help and direction throughout my project completion. The meetings s and discussions were indispensable in moving me to consider think fresh possibilities, from various points of view to frame an exhaustive and objective study.

ABSRACT

Chatbots becoming more used in every organisation, they are beneficial in automating so many aspects. In this project, we hope to achieve the goal of performing the emotion analysis using a chatbot. The idea is to use two trained models to complete the task of emotion analysis. One model is to identify the sentence's emotion, and other is provided to respond to the sentence. And both these models will be used in the web application, that web application provides the interface to chat and graphically present the outcomes. For sentiment analysis, pertained BERT model provides contextual tokenisation and adds a new layer of classification to provide the sentiment analysis capability. Chatterbot API provides inbuilt database and inbuilt functionality to handle the training and deploying the model.

List of content: **Page. No**

List of figures: Page. No

List of Tables:

1. INTRODUCTION

1.1 OVERVIEW:

Emotion analysis using the Chatbot project's target is to understand the person's emotional status using the text generated in chatbot conversation and person. Sentiment analysis is one of the use cases of Natural language processing (NLP).

In NLP using different techniques like Vectorization, Lemmatization, word embedding, we can convert the text to vectors. Then after using this numerical notation of text, we can train the neural network and get the desired results. Models like BERT can be used for sentiment analysis by training with a dataset that has sentences with labelled emotions.

A chatbot is a computer program designed to simulate conversations with a human user [Anadea, 2018]. Retrieval chatbot is works based on predefined patterns/responses. By collecting the conversation history of the chatbot and person over some specific period and analyzing the text for various parameters like numbers of negative words/sentences, positive words/ sentences and other relevant parameters and representing them in the form of graphs may help in understanding the person emotional state and its variations quantitatively.

1.2 AIM OF THE PROJECT:

To create a chatbot and analyzing the emotions of a person using a chatbot.

1.3 LITERATURE REVIEW:

Generally, the Chatbots are mainly used to carry out the specific tasks that include the customer service with respect to product introduction and solve the problems, thereby saving Human Resources. This emotional analysis carried out in this project can be applied in Medical care as the user’s emotional state and the best response can be reviewed using the chatbot.The Sequence of steps incorporated in the project is the Training and Testing of the BERT model, creating the chatbot, Integrating the created bot with the BERT model, deciding the question for analysis, and Store and plot the analysis per person and per day.

1.4 RESEARCH QUESTION

Can emotion analysis done on the conversation of person and chatbot will be helpful in understanding a person's emotional state better?

1.5 OBJECTIVE

- Train and test BERT model for sentiment analysis.
- Create Chatbot using Chatterbot API.
- Create a web application using a Django framework.
- Integrate the bot with BERT model.
- Graph the analysis of the emotions.
- Create and deploy the application in AWS environment

2. BACKGROUND RESEARCH

2.1 ARTIFICIAL INTELLIGENCE(AI)

The roots of Ai were planted in late 1950 by English Scientist Alan Turing with the proposal of **'Turing Test'** which tests that can a computer think intelligently like humans. In 1951 the first AI chess game was created. John McCarthy used the term '**Artificial Intelligence'** at the Dartmouth Conference held in 1956. MIT set up the first laboratory in 1959. Followed by that first Chatbot *'ELIZA'* was introduced in 1961. IBM's *DEEPBLUE* beat the world's chess champion in 1997. Stanley, an autonomous robotic car wins the 2005 DARPA Grand Challenge. In 2011 *IBM Watson* beat the Jeopardy champions, and in the same year *Apple Siri, Google Now Cortana* created a trend. IN 2016 *Google DeepMind* beats Korean Alpha Go champion. At every stage, the advancements of AI making tremendous changes in society.

Artificial intelligence is a branch of computer science in which machines are made to mimic the human intelligence. AI machines are artificially incorporated with human-like intelligence to perform tasks as we do [Anadea, 2018]. This intelligence is built using complex algorithms and mathematical functions. AI is used in smartphones, cars, social media feeds, video games, banking surveillance and many other aspects of our daily life. Ai provides machines with the capabilities of 'Adapt', 'Reason' and provides solutions. Machines are called artificially intelligent only when it satisfies generalised learning capabilities, Reasoning, and Problem-solving. Artifical Intelligence technologies are limited and Lack of common sense is one of its Limitation. The Basic human emotions like pain, Joy, fear, sad and neutrality can easily be detected by AI

2.2 NLP

Natural language processing common called as NLP is a branch of AI which deals with a way to communication between a machine and human using natural language.

Processing of takes place in 3steps they are as follows:

1. Understanding Natural Language: Capability of a computer to understand what human says.
2. Generation of Natural Language: Generation of human understandable natural language
3. Speech Recognition: Converting oral language to text.

2.3 DEEP LEARNING

To overcome the limitations in traditional machine learning and to solve more complex problems, Deep learning is introduced. Deep learning is a new type of data-driven AI. It stacks multiple layers of ML models, one on top of each other. That allows us to learn more complex inputs and produce more complex outputs. We typically create deep-learning models using a neural network.

A neural network is a graph of nodes and edges based roughly on neurons' organization in an organic brain.

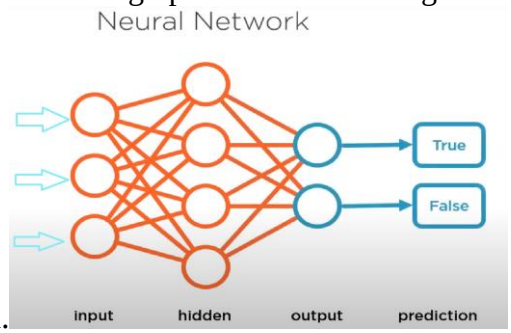


fig 2.3.1 Structure of Neural Network

The nodes represent neurons in a brain. The edges represent the connections between the neurons. First, we feed data into the neural network via its input neurons. Next, mathematical operations are performed on the data in each of the neurons. Then, each neuron forwards its resulting value to all of the other neurons that it's connected to. We repeat this process for all of the nodes in the hidden layer of the network as well as the edges in the hidden layer. Finally, the network produces a prediction from its output neurons as shown in fig 2.3.1

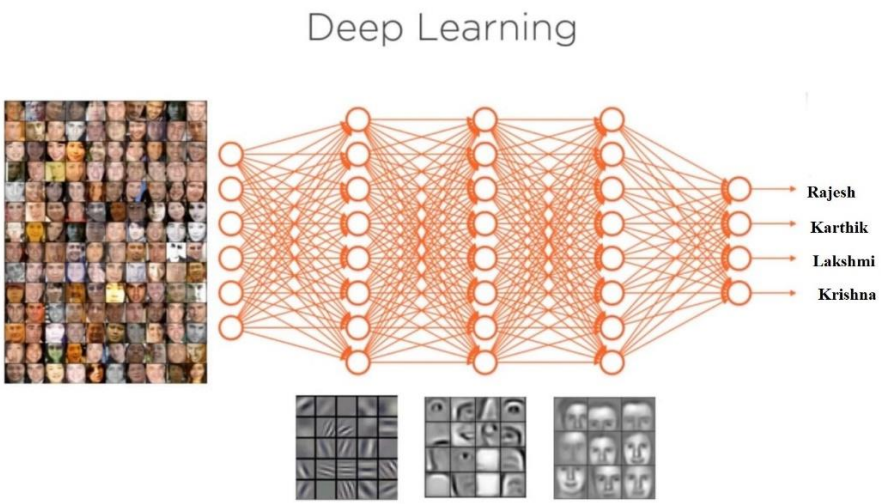


Fig 2.3.2 Deep Learning classification

A deep neural network is a neural network with more than one hidden layer. Adding more hidden layers allows the network to model progressively more complex functions. For example, imagine we want to teach a deep neural network on how to detect human faces. First, we would feed a set of labelled images into the input layer of this network. We do this to teach the network the faces of each person and their corresponding name. The neural network's first layer would learn to detect geometric primitives—for example, horizontal, vertical, and diagonal lines. The second hidden layer would learn to detect more complex facial features. or example, Eyes, noses, and mouths. The third hidden layer would learn to detect the general pattern for entire faces. The output layer would detect the most abstract representation of a person. In this case, the name of the person being recognized. Each layer learns to extract more complex features from the

preceding layer. As a result, the data becomes more abstract with each additional layer in the network.

Limitations:

Data: Training a deep learning model requires huge chunks of data sets to make it decently accurate.
computational power: Training a deep learning system requires a high amount of computation that's why we generally employ using GPU or graphical processing unit which have more core than CPU and also carries a higher cost number.
Training time: Training an average deep learning system can take weeks or even months to process and perfect the training time is usually dependent on the amount of data and the number of layers in the hidden network.

2.4 Git

Git is the most popular version control system in the world. git is almost everywhere more than 90% of software projects in the world A version control system records the changes made to our code over time in a special database called 'Repository'. we can look at our project history and see who has made what changes when and why. In nutshell with a version control system we can track our project history and work together.

Version control systems fall into two categories they are centralized and distributed. In a centralized system all team members connect to a central server to get the latest copy of the code and to share their changes with others. subversion and Microsoft team foundation server are examples of centralized version control systems. The problem with the centralized architecture is the single point of failure if the server goes offline we cannot collaborate or save snapshots of our project so we have to wait until the server comes back online.

In distributed systems we don't have these problems every team member has a copy of the project with its history on their machine so we can save snapshots of our project locally on our machine if the central server is offline we can synchronize our work directly with others git and mercurial are examples of distributed version control systems. out of all these, git is the most popular version control system in the world because it's free open source, superfast and scalable. Operations like branching and merging are very fast and git.

2.5 SQLite:

It is a database base motor which allows the customer to talk with social or RDBMS. Additionally, actualises an independent, serverless database motor. In this, it stores the database in a single report. It contains different documents, files and tables in a solitary circle record. So that, user can get to the record with no issue. User can copy or share the database or can copy the documents from the database.

Additionally, it doesn't have any different server process. Nowadays it comes as an inbuilt application in the cell phones, PC with no. of applications. The code of the SQLite is in the public domain. So anybody can utilise that for private and business reason.

Advantages:

1. It is an embedded software with devices like cell phone, computer. It is a light weighted database.
2. It performs operations quickly, like reading, writing, etc.
3. SQLite is reliable as it updates continuously. So, in case of a power failure there no work is lost.
4. It is portable with any 32 or 64 operating system without any compatibility problem.
5. SQLite can be easily extended in future.

Disadvantages:

1. It has a restriction of size up to 2GB.
2. It has limitation for rows.

2.6 AWS

AWS, also known as Amazon Web Services, is a global cloud platform that provides almost 165 fully featured services; it is one of the most comprehensive cloud computing platforms. A broad range of organisations adopts its services where users can access on-demand. Digital infrastructure capabilities like computational power storage and digital services as a plug-and-play system to build as per their needs it stands to deliver four tenets to the end-user. IAS infrastructure as a service example caching databases servers computation ETL pipes. PaaS platform as a service example application development ml and AI capabilities web streaming hosting SaaS software as a service example ERM CRM DaaS development and data as a service.

2.6.1 How popular AWS is and why most organisations are using it:

AWS drives most companies from Netflix Adobe Expedia, and so many other big or small companies are highly dependent on the AWS cloud service. AWS is the leader in the cloud market space in the world. It is also the biggest revenue generator for Amazon across amazon.com and several other businesses, including e-commerce. AWS is the leading cloud provider in the world, is a natural fit for an established company or a start-up to transition or build its business in a cloud-first world today.

There are eight reasons why organisations are using AWS:

1. AWS is the largest most comprehensive services used in a significant number of regions in the world
2. It provides users with free signup and one year of free usage of almost 95% of their services. Users can try them out and work on our MVP or cloud strategy. This also helps in easy integration and adoption for their services number 3 API's / SDK for all their services.
3. AWS provides API's for almost all their services, and hints help automate all of them if you have to spin up hundreds of instances of EC2. It's a single API call developer find this very intuitive to use the number for cost-effective.
4. High scalability and global reach one of the tenets of all AWS services is that they scale to support billions of users.
5. AWS provides a hardening of security across networks applications, firewalls, and pen testing hardware and always seeks to work with the hacker community to secure its system's security.

6. AWS works around a service-oriented architecture with individual components.
7. AWS is the clear market leader in cloud computing if you want to have a career in AWS.
8. AWS certification commands high-paying jobs; the certification is a goal that is achievable in a short period without spending too much money.

3. MODELS

Long Short-Term Memory:

It is a kind of artificial RNN (Recurrent neural net) designed to overcome the limitations like vanishing gradient and explosion gradient in RNN. LSTM networks are delayed to train. Words are passed in successively and are generated successively. Takes certain time to learn, and it's not actually the best of catching the genuine meaning of words. Even in the bi-directional LSTM's learning process takes place from right to left and left to right separately and afterwards concatenating them, so the genuine context is marginally lost as shown in the figure below.

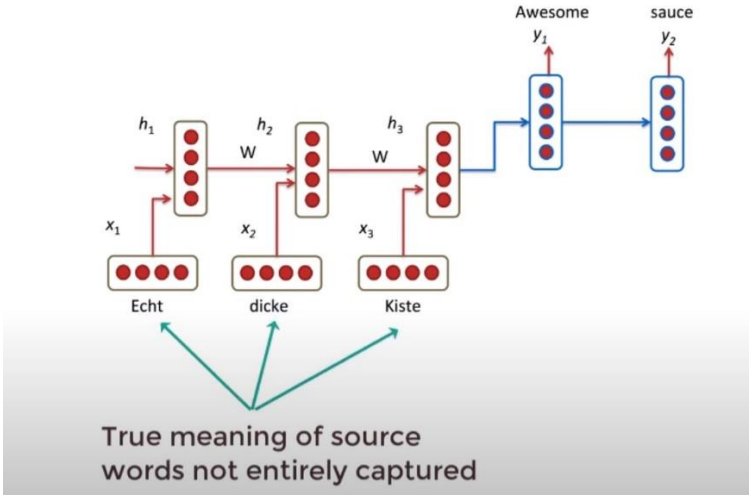


Fig 3.0.1 Drawback of LSTM

3.1 TRANSFORMERS:

2018 was an advancement year in NLP. Move learning, especially models like ELMO(Embedding for Learning models), GPT(Generative Pre-Trained Transformer), and BERT by google permitted scientists to crush various standards with minimal fine-tuning in particular task and gave the remainder to the NLP society with pre-prepared models that could fine-tuned in a simple way (with less information and less processing time) and executed to create a best in class results.

The transformer architecture addresses some of the problems in LSTM. The transformers are quick, as words are processed simultaneously and the word context is learned better as they can attain context from both directions at the same time. The transformer includes two key parts an encoder part and a decoder part. The encoder takes the English words all the while and creates embeddings for each word at the same time. These embeddings are vectors that encapsulate the significance of the word. Comparable words have nearer numbers in their vectors. The decoder takes these embeddings from the encoder and the recently generated words of the translated sentence, and afterwards, it utilizes them to create the following word, and we continue creating next word at a time until the finish of the sentence is reached.

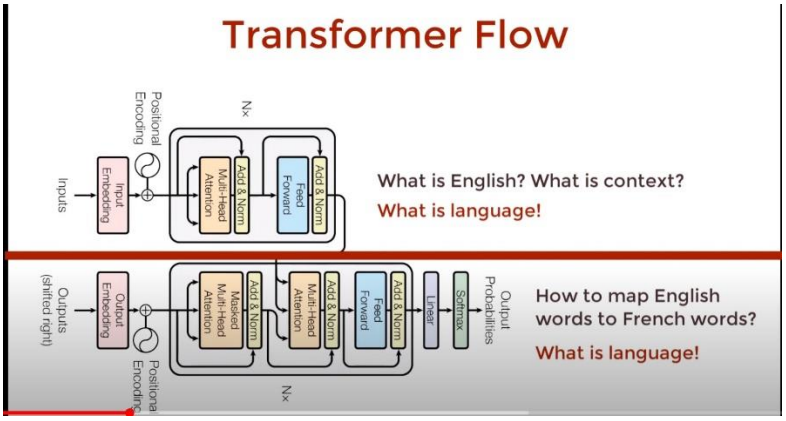


Fig 3.1.1 Transformer flow

The encoder learns what is English? what is grammar and more importantly what is context the decoder learns how to English words relate to French words both of these even separately have some underlying understanding of language and it's because of this understanding that we can pick apart this architecture and build systems that understand language. If decoders are stacked GPT (Generative pre-trained Transformer) transformer architecture is obtained. Likewise, if we stack the encoders BERT a bi-directional encoder representation can be seen.

3.2 BERT MODEL:

BERT stands for Bidirectional Encoders Representation from Transformers. These are developed in 2018 by Google. The large BERT model comprises of 24 layers achieves more accuracy with 340M parameters. The BERT base model comprises of 12 layers and has less accuracy than the lager model with 110M parameters. We can utilize BERT to learn language interpretation, question replying, Emotion Analysis, text synopsis and a lot more tasks. All of these problems require an understanding of the language. So BERT is trained to learn a language and fine-tune BERT upon the difficult we need to solve.

Architecture:

BERT training involves two stages the main stage is pre-training in which the model comprehends what is language and context and the subsequent stage is fine-tuning where the model learns ‘I know the language yet how would i settle this from’.

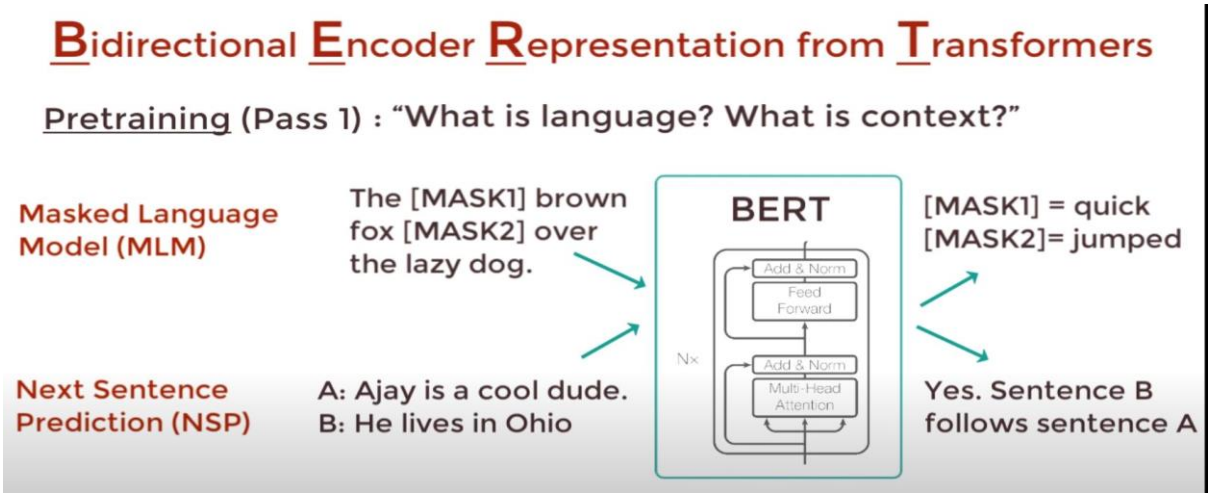


Fig 3.2.1 processing of pretraining

The goal of pre training is to make BERT learn of what is language and context. BERT learns language via training on two unsupervised undertakings at the same time. They are mass language demonstrating and next sentence prediction. For mass language modelling BERT takes in a sentence with arbitrary words loaded up with masks. The objective is to output these mask tokens, and this is somewhat similar to fill in the blanks. It helps BERT with understanding a bidirectional context inside a sentence on account of next sentence prediction. BERT takes in two sentences and it decides whether the subsequent sentence really follows the first in kind of what is like a binary classification problem. This helps BERT understand context across different sentences themselves and using both of these together BERT gets a good understanding of language great.

In the fine-tuning stage BERT further trained on very specific NLP (Natural Language Processing) tasks. For example, let's take question answering all we need to do is replace the completely associated output layers of the network with a new arrangement of output layers that can basically output the answer to the question we want. Then we can perform supervised training using a question answering data set it won't take long since it's only the output parameters that are learned from scratch as shown in fig (3.2.2). The remaining of the model parameters are just slightly fine-tuned and as a result training time is fast and we can do this for any NLP problem that is replace the Output layers and then train with a specific data set.

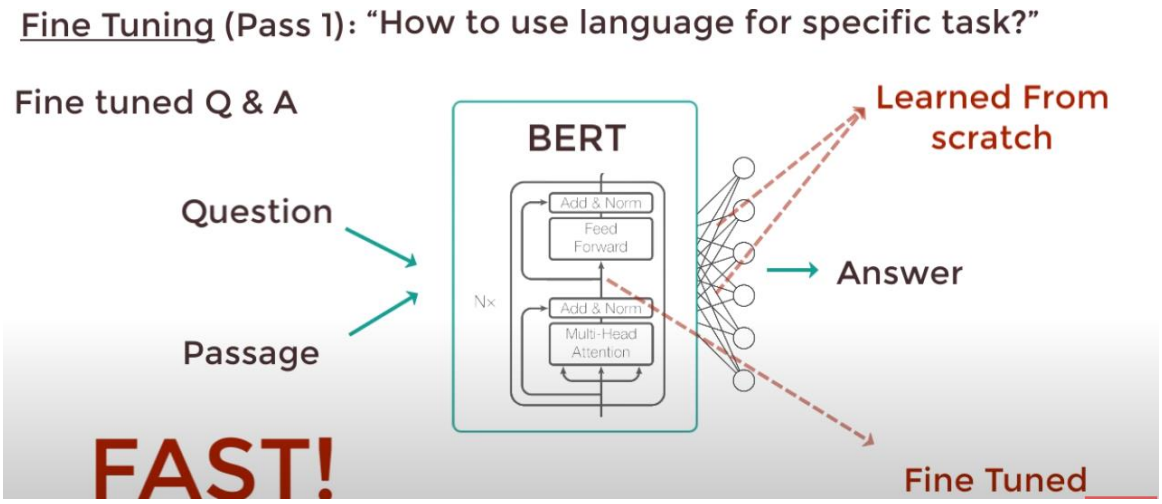


Fig (3.2.2) Learning of fine tuning

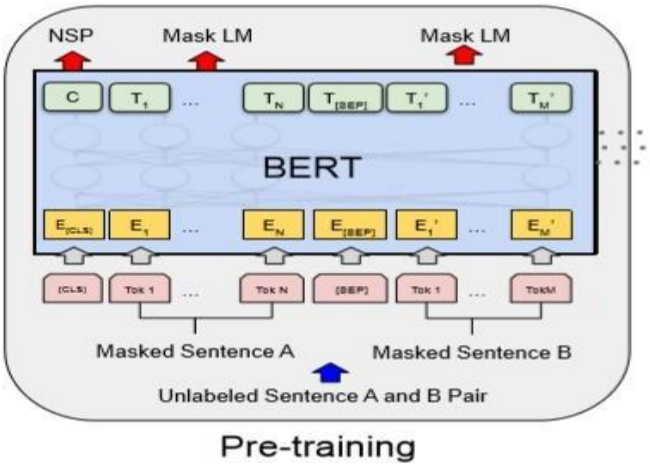


Fig 3.2.3 BERT training with unlabelled pair

As shown in the above fig (3.2.3) During BERT pre-training we train the model on modelling of mass language and prediction of next sentence. In practice both of these problems are trained simultaneously. The input is a set of two sentences with some of the words being masked. Each token is a word and we convert each of these words into embeddings using pre trained embeddings. this provides a good starting point for BERT to work with. now on the output side C is the binary output for the next sentence prediction. So, it would output 1 if sentence B follows sentence A in context and 0 if sentence B doesn't follow sentence A. Each of the T's here are word vectors that correspond to the outputs for the mass language model problem. So, the number of word vectors that we input is the same as the number of word vectors that we output.

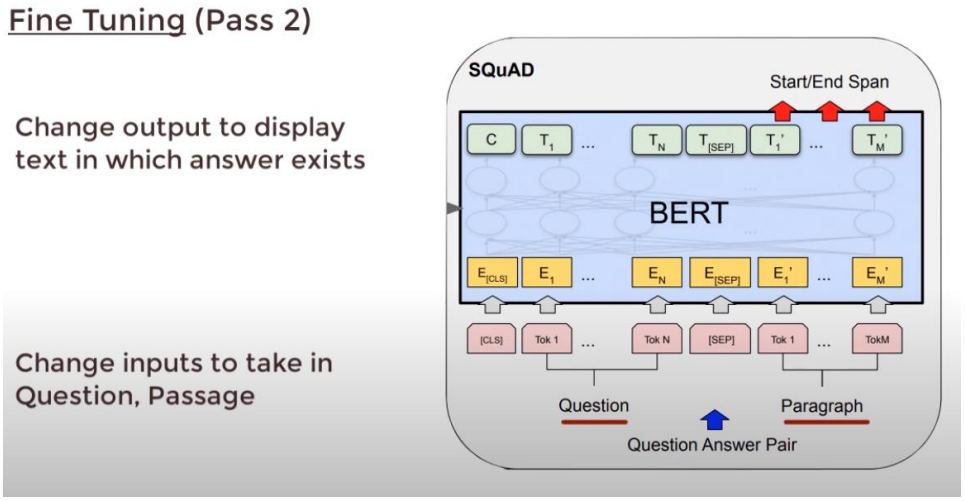


Fig 3.2.4 performing Question answer pair

As figured in fig (3.2.4) if we wanted to perform question-answering we would train the model by modifying the inputs and the output layer. we pass in the question followed by a passage containing the answer as inputs and in the output layer we would output these Start/End Span words that encapsulate the answer. Assuming that the answer is within the same span of text.

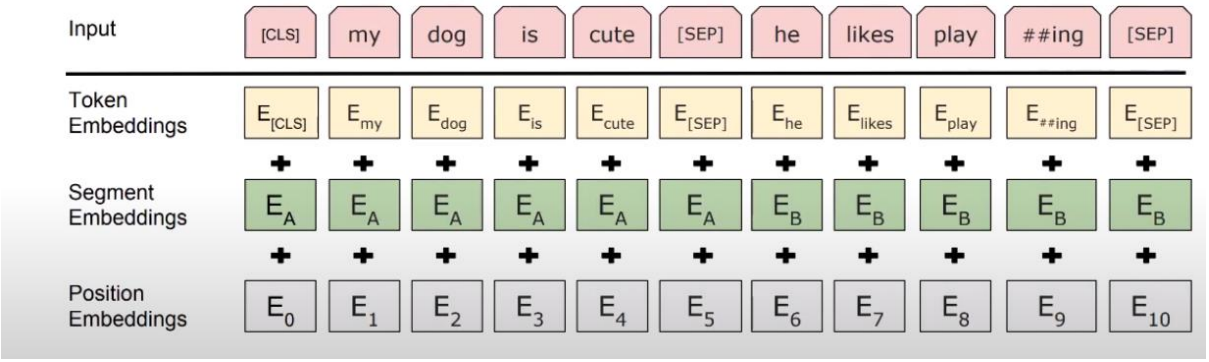
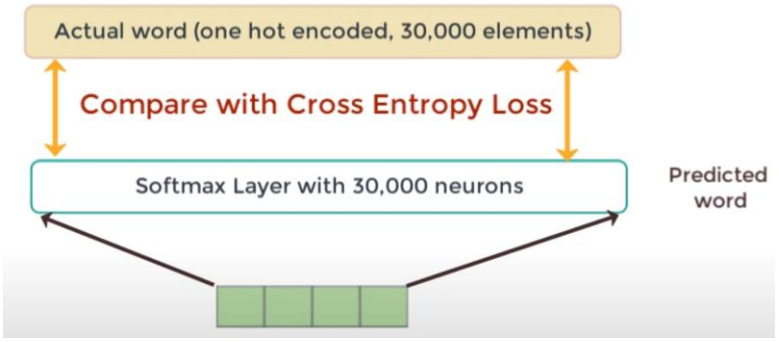


Fig 3.2.5 Generation of Embeddings

The initial embedding is constructed from three vectors which are, the token embeddings are the pre-trained embeddings the main paper uses word piece embeddings that have a vocabulary of 30,000 tokens. The segment embeddings are essentially the sentence number that is encoded into a vector. The position embeddings are the situation of a word inside that sentence that is encoded into a vector. Adding these three vectors together we get an embedding vector that we use as input to BERT. The segment and position embeddings are required for temporal ordering. since all these vectors are fed in simultaneously into BERT. Language models need this ordering preserved.



ssFig 3.2.6 comparing entropy loss

The output side now has a binary value C and a bunch of word vectors but with training we need to minimize a loss. All of these word vectors are generated simultaneously. We need to take each word vector pass it into a fully connected layered output with the same number of neurons equal to the number of tokens in the vocabulary.so that would be an output layer corresponding to 30,000 neurons in this case as shown in fig 3.2.6. We would apply a soft max activation. this way we would convert a word vector to a distribution. The actual label of this distribution would be a one hot encoded vector for the actual word and so we compare these two distributions and then train the network using the crossentropy loss. But the output has all the words even though those inputs weren't masked at all. The loss though only considers the prediction of the masked words and it ignores all the other words that are output by the network. This is done to ensure that more focus is given to predicting these mass values so that it gets them correct and it increases context awareness.

In short we pretrained BERT with mass language modelling and next sentence prediction. For every word we get the token embedding from the pre trained word piece embeddings. Add the position and segment embeddings to account for the ordering of the inputs. These are then passed into BERT which under the hood is a stack of transformer encoders and it outputs a bunch of word vectors for mass language modelling and a binary value for an next sentence prediction. The word vectors are then converted into a distribution to Train using cross entropy loss. Once training is complete BERT has some notion of language, it's a language model. Further step is the fine-tuning stage where supervised training is performed depending on the task we want to solve. The Bert SQuAD(Stanford Question & Answer Dataset) only takes about 30 minutes to fine-tune from a language model for a 91% performance. Performance depends on size of the BERT model. The large BERT model comprises of 24 layers achieves more accuracy with 340M parameters. The BERT base model comprises 12 layers and less accuracy than the lager model with 110M parameters.

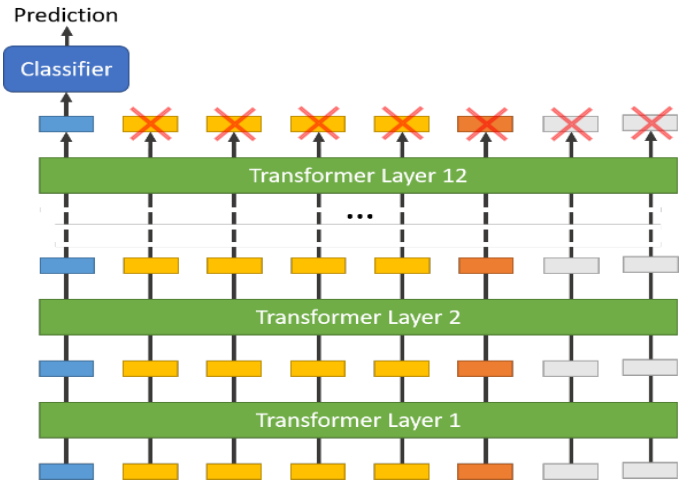


Fig (3.2.7) Adding classifier layer to BERT model

An additional layer of classifier layer is added to BERT for classifying the emotions. So that BERT model gives the emotions classified in this layer depends up on the input responses given to the chatbot. Usually BERT comes up with 12layer but one more layer is added to classify the emotion.

Word embedding Vs Contextual embedding:

- Word embedding like Word2Vec is a representation of words in vector format.
- Word will have the same vector representation even in different contexts.
- But BERT will identify this difference in context-based surrounding word and give different vectors for the same word when it is used in a different context.
- `tokenized` and `raw` versions

```
print(' Original: ', sentences[0])
print('Tokenized: ', tokenizer.tokenize(sentences[0]))
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))
```

```
 Original:  How the fu*k! Who the heck! moved my fridge!... should I knock the landlord door. #angry #mad ##
Tokenized:  ['how', 'the', 'fu', '*', 'k', '!', 'who', 'the', 'heck', '!', 'moved', 'my', 'fridge', '!', '.', '.', '.', 'shoul
d', 'i', 'knock', 'the', 'landlord', 'door', '.', '#', 'angry', '#', 'mad', '#', '#']
Token IDs:  [2129, 1996, 11865, 1008, 1047, 999, 2040, 1996, 17752, 999, 2333, 2026, 16716, 999, 1012, 1012, 1012, 2323, 1045,
7324, 1996, 18196, 2341, 1012, 1001, 4854, 1001, 5506, 1001, 1001]
```

Figure 3.2.8. *`tokenized` and `raw` versions*

3.3 ADVANTAGES OF FINE-TUNING:

We will utilize BERT to prepare a book classifier. In particular, we will take the pre-prepared BERT model, include an undeveloped layer of neurons at long last, and train the new model for our order task [Choudhury, A., 2020].

Quicker Development:

First, the pre-trained BERT model loads as of now encodes a ton of data about our language. Subsequently, it takes considerably less effort to prepare our adjusted model. Maybe we have just trained the base layers of our network widely and need to fine-tune them while utilizing their output as highlights for our arrangement task. Truth be told, the creators suggest just 2–4 epochs training for calibrating BERT on a particular NLP task (contrasted with the many GPU hours expected to prepare the first BERT model or an LSTM without any preparation!)

Less Data:

As BERT is pre-trained no need to train from scratch which is main drawback in NLP. By fine-tuning BERT performance will be good even though it is trained with less training data.

Better Results:

At last, this basic fine-tuning method (normally including one completely associated layer head of BERT and preparing for a couple ages) was appeared to accomplish a cutting edge results with negligible assignment explicit alterations for a wide assortment of undertakings: arrangement, language deduction, semantic likeness, question replying, and so on.

3.4 ADAM OPTTIMIZATION ALGORITHM:

The **Adam optimization algorithm** is a sophisticated extension to stochastic gradient descent that has newly seen more widespread adoption for deep learning applications in computer vision and natural language processing.

Adam **optimization algorithm** can be used preferably of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

Advantages:

- Straightforward to implement.
- Relatively less memory requirements.
- Computationally more efficient.
- Well suited for problems that are large in terms of data and/or parameters.
- Invariant to diagonal rescale of the gradients.
- Suitable for problems with sparse or very noisy gradients.
- Hyper parameters have natural interpretation and typically require little tuning.

Adam is different from traditional ***SGD***. ***SGD*** maintains a single learning rate termed alpha for all weight updates, and the learning rate does not change during training. Learning rate is maintained for each network weight/parameter and separately adjusted as learning unfolds. The method computes individual adaptive learning rates for different parameters from estimates of the gradients' first and second moments.

Adam **algorithm** as combining the advantages of other 2 SGD extensions as follows:

- **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients

e.g. natural language and computer vision problems.

- **Root Mean Square Propagation** (RMSProp) also maintains per-parameter learning rates adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). Means the algorithm does well on online and non-stationary problems

e.g. noisy.

Adam Configuration Parameters are as follows:

- **Alpha**: Also referred to as the learning rate or step size. The proportion that weights are updated. Larger values result in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training.
- **Epsilon**: Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).
- **beta1:** The exponential decay rate for the first moment estimates.
- **beta2:** The exponential decay rate for the second-moment estimates. This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).

3.5 CHATBOT:

According to a dictionary of Oxford chatbot is defines as "A computer program designed to simulate conversation with human users, especially over the internet." Chatbots are significant tools as they incredibly increment the client experience while connecting with apps and sites and can be immediately sent on existing sites and apps. Thus, there has been a mind-blowing ascend in the use of chatbots for different use cases in various fields, for example, client care for various associations, in-application backing to upgrade the client experience, lead age in online deals. Therefore, Chatbots can likewise be utilised to comprehend user reactions and react to get more helpful data during communication with the user.

A portion of the improvements made in chatbot innovation incorporates Chatbot advancement stages. These stages helped to create and deploy updated variants of chatbots rapidly. These stages can be utilised to build up a flow of dialogues for a chatbot. The chatbot is given a fixed arrangement of inquiries and reactions to collaborate with the client. The chatbot chooses from its predefined set of reactions to communicate with clients.

3.5.1 Types of chatbots:

Chatbots are broadly classified into five types they are as follows:

Scripted Chatbots: These are for specific instruction based interaction. When users make queries in a specific language, the bot provides answers based on the data from a pre-defined knowledge base.

Social Messaging Chatbot: These are Integrated with a social platform like WhatsApp, Messenger, Telegram. These are instant and ideal for a personalised conversation. Interactions are conversational and friendly and are known for quick response time.

Transactional Bots: These are perfect for making a quick transaction (e.g. purchase, money transfer). These are not unusual Q&A bots and can only perform limited functions. Simple yet sophisticated, these bots require structured data and NLP tools and human-centric UI/IX for conversation flow.

AI-Powered Chatbot: These are suitable for businesses that receive hundreds of queries every day and have to offer customer support services 24/7. An AI bot can communicate with several users at a time carrying out complex tasks, powered and facilitated by NLP and ML tools.

Voice Assisted Chatbot: These are assumed to be the future bot technology, AI voice assistants(Alexa, Siri, etc) provide advance support services by taking queries through voice. Powered by text-to-speech and voice recognition software. This is the best of personalisation that AI bots offer.

3.6 WEB APPLICATION:

A web application is nothing but a program which generally uses internet and technology as well to execute various tasks over the internet. Web applications mostly coded or designed with language which is understand by browsers like HTML, JavaScript. As this are the languages supported and executed by web browsers. To manage request from client, the web application uses web server, for storing data it uses database, also for perform task it uses application server.Python, JSP, ASP.NET are application server technology. Application server innovation goes from ASP.NET, ASP and ColdFusion, to python and JSP. Popular examples of web application are Gmail, Yahoo, and Google Apps.

There are following steps or flow for web application:

1. In the web application process, first client sends a request to web server over a web through internet browser or application UI.

2. After accepting request from client, web worker advances request to web application server. At that point web application server perform task like questioning or processing of information and show result.

3. Then web application server sends the outcome with expected data to web server and web server at that point reacts back that outcome to the client.

4. In this way, the correspondence occurs among client and web application.

Benefits of web application:

1. It can run on multiple platforms but not on operating system.
2. It doesn’t require any installations on hard drive. In that way, it saves memory.
3. It also allows user to update their own details.
4. It doesn’t require any maintenance. It reduces business cost
5. Each user can access the same version.

3.6.1 Model-View-Controller:

Model-View-Controller which is popularly called as MVC. It is an architectural pattern which governs the web application. MVC divides a software into three parts which are Model, View and Controller. Each of these have specific responsibilities

Model:

The Model is responsible for handling the data logics to interact with the database. Data base sends back the data to the Model. If required model creates a business logic to interact with database to fetch the data. It communicates with the controller before interacting with database

.

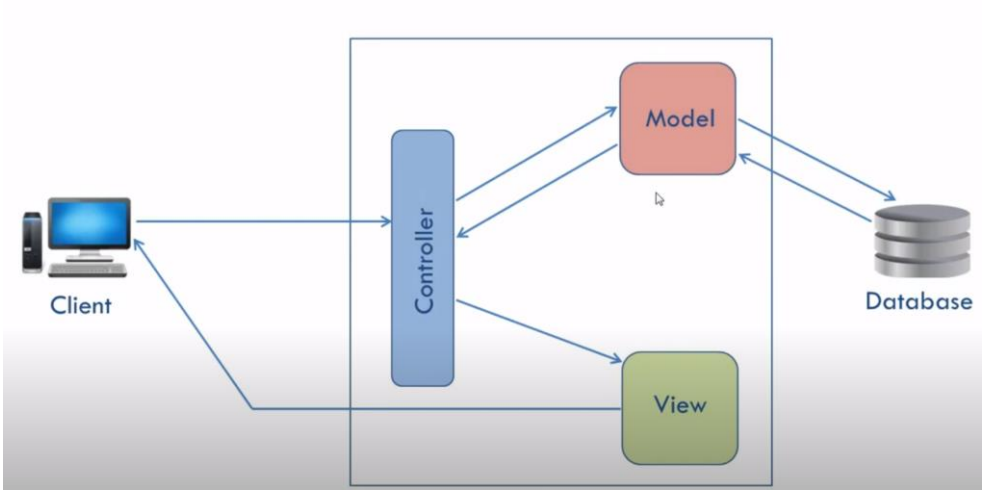
View:

View is responsible for handling data presentation. This is the actual view of user when they interact with he application. View consists of HTML/CSS along with dynamic values from the controller.

Controller:

Controllers is a mediator between Model and View parts. It receives the input from the user and gets interact with model and and view to process the output.

3.6.1(A) Working of MVC:



Fig(3.6.1) working of MVC

1. Initially user sends a request to the controller. Controller send this request to the Model component.
2. Model handles the data logic to communicate with the database and fetches the data from the database. This is sent back to the controller.
3. After getting the data from the Model, Controller send the data to View component to get the presentation or UI , View will send this response to the client.
4. In the entire process Model and View never interact with each other.

3.6.2 Django :

Django is a web development framework for python which offer standard methods for fast and effective web development. Tt helps user to assist in building and maintaining quality web applications. It removes lot of unnecessary work for web developers.it it is a high-level wave

framework which allow the performance rapid development the primary goal of the web framework is creating complex database drive and website. It is named after a famous guitarist 'Django Reinhardt.

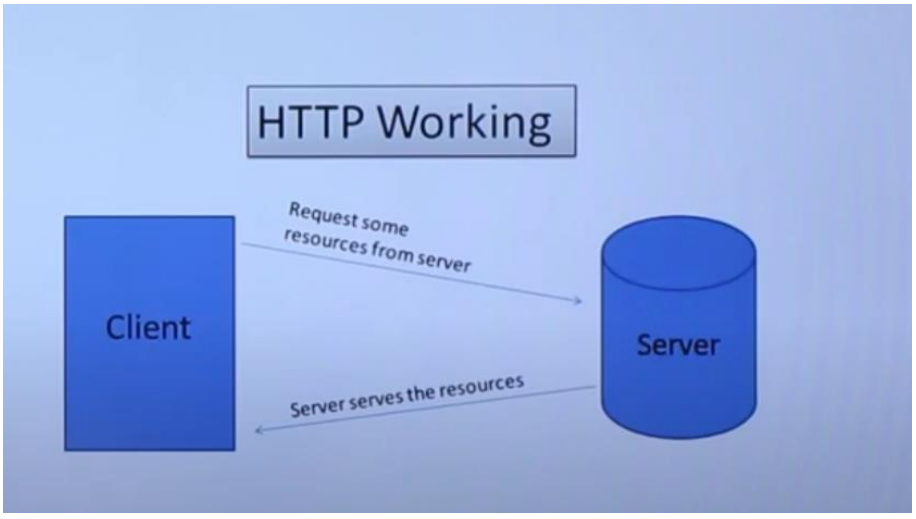
It is a open-source framework that follows principle of "don't repeat yourself" (which makes code simple and non-repeating). It follows MVC -MVT (Model View Controller-Model View-Templates) architecture. It supports multi column keying, ORM (Object Relational Mapping). It is featured with security authentication system and it provides powerful and production ready interface that can be used to manage content on user site.

Feautures of Django:

- **Fast** : Helps developers to to build web applications quickly.
- **Tons of Packages** **:** Provided with loads of packages to build web applications.
- **Secure:** **:** Provide with high security than any other framework do.
- **Scalable:** **:** Django scale to meet the heaviest traffic demands quickly.
- **Versatile** : Django's versatile feature is to allow web developers to build any type of content management system. For example Instagram, Facebook, Bitbucket, News Articles and so on.

3.6.3 HTTP:

HTTP stands for Hypertext Transfer Protocol. so it's a client-server model in which a client requests a particular resource from the server whereas the server serves that resource to the client. so HTTP protocol is used for accessing a resource on the internet and it is used to request a resource from the server. and here you can see the HTTP working so here is our client here is our server so client requests a particular resource and the server will serve that resource to the client which is illustrated in the figure.



Fig(3.6.3) working of HTTP

HTTP methods are types of requests which we want to send. For suppose we want to send an HTTP request and what we have to do in HTTP requests can be defined as HTTP methods.

Different HTTP methods are as follows:

- **Get** : Get method is used for accessing any resource or fetching out any resource.
- **Head** : Head method is used to see the server HTTP headers but not the HTML data.
- **Post** : For updating the data at a resource, POST method is used.
- **PUT** : For creating data at a resource PUT method is used.
- **Patch** : To partially update the data at a resource.
- **Options**: if the user wants to know that what kind of HTTP methods are enabled on a remote server then a user can send HTTP requests by using options method. and in the response the user will see that what kind of HTTP methods are actually enabled on a remote server.
- **Delete** : If the delete method is enabled on any server then a user can delete any particular resource on the server by sending the HTTP request by using delete method.
-

HTTP Status Codes:

Errors in the web pages occurs due to the 'http status code'. Reason behind the occurrence of http codes are tabulated in table(1.0) [Gutool.com. n.d].

"HTTPStatus Code : 100	HTTP Status Message: Continue
HTTP Status Code : 101	HTTP Status Message : Switching Protocols
HTTP Status Code : 103	HTTP Status Message : Early Hints
HTTP Status Code : 200	HTTP Status Message : OK
HTTP Status Code : 201	HTTP Status Message : Created
HTTP Status Code : 202	HTTP Status Message : Accepted
HTTP Status Code : 203	HTTP Status Message : Non-Authoritative Information
HTTP Status Code : 204	HTTP Status Message : No Content
HTTP Status Code : 205	HTTP Status Message : Reset Content
HTTP Status Code : 206	HTTP Status Message : Partial Content
HTTP Status Code : 300	HTTP Status Message : Multiple Choices
HTTP Status Code : 301	HTTP Status Message : Moved Permanently
HTTP Status Code : 302	HTTP Status Message : Found
HTTP Status Code : 303	HTTP Status Message : See Other
HTTP Status Code : 304	HTTP Status Message : Not Modified
HTTP Status Code : 307	HTTP Status Message : Temporary Redirect
HTTP Status Code : 308	HTTP Status Message : Permanent Redirect

HTTP Status Code : 400	HTTP Status Message : Bad Request
HTTP Status Code : 401	HTTP Status Message : Unauthorized
HTTP Status Code : 402	HTTP Status Message : Payment Required
HTTP Status Code : 403	HTTP Status Message : Forbidden
HTTP Status Code : 404	HTTP Status Message : Not Found
HTTP Status Code : 405	HTTP Status Message : Method Not Allowed
HTTP Status Code : 406	HTTP Status Message : Not Acceptable
HTTP Status Code : 407	HTTP Status Message : Proxy Authentication Required
HTTP Status Code : 408	HTTP Status Message : Request Timeout
HTTP Status Code : 409	HTTP Status Message : Conflict
HTTP Status Code : 410	HTTP Status Message : Gone
HTTP Status Code : 411	HTTP Status Message : Length Required
HTTP Status Code : 412	HTTP Status Message : Precondition Failed
HTTP Status Code : 413	HTTP Status Message : Payload Too Large
HTTP Status Code : 414	HTTP Status Message : URI Too Long
HTTP Status Code : 415	HTTP Status Message : Unsupported Media Type
HTTP Status Code : 416	HTTP Status Message : Range Not Satisfiable
HTTP Status Code : 417	HTTP Status Message : Expectation Failed
HTTP Status Code : 418	HTTP Status Message : I'm a teapot
HTTP Status Code : 422	HTTP Status Message : Unprocessable Entity
HTTP Status Code : 425	HTTP Status Message : Too Early
HTTP Status Code : 426	HTTP Status Message : Upgrade Required
HTTP Status Code : 428	HTTP Status Message : Precondition Required
HTTP Status Code : 429	HTTP Status Message : Too Many Requests
HTTP Status Code : 431	HTTP Status Message : Request Header Fields Too Large
HTTP Status Code : 451	HTTP Status Message : Unavailable For Legal Reasons
HTTP Status Code : 500	HTTP Status Message : Internal Server Error
HTTP Status Code : 501	HTTP Status Message : Not Implemented
HTTP Status Code : 502	HTTP Status Message : Bad Gateway
HTTP Status Code : 503	HTTP Status Message : Service Unavailable

HTTP Status Code : 504	HTTP Status Message : Gateway Timeout
HTTP Status Code : 505	HTTP Status Message : HTTP Version Not Supported
HTTP Status Code : 506	HTTP Status Message : Variant Also Negotiates
HTTP Status Code : 507	HTTP Status Message : Insufficient Storage
HTTP Status Code : 508	HTTP Status Message : Loop Detected
HTTP Status Code : 510	HTTP Status Message : Not Extended
HTTP Status Code : 511	HTTP Status Message : Network Authentication”

Table 2.0 HTTP codes and reason for the occurrence

4.PROJECT PLAN

The plan behind the implementation and execution of 'Emotion Analayis Using Chatbot' is tabulated as shown in the table(4.0).

Project Plan				
Sno	Task	Subtasks	Status	Hours Spent
1	Dataset Identification	Exploratory data analysis	Implemented and Completed Successfully	16 hours
		Drawing different graphs	Implemented and Completed Successfully	08 hours
2	BERT Model	understanding basic theory behind BERT	Implemented and Completed Successfully	20 hours
		Explore dataset distribution and some basic pre-processing	Implemented and Completed Successfully	17 hours
		Loading pre-trained tokenizer to encode our text data into numerical values (tensors)	Implemented and Completed Successfully	13 hours
		Load in pre-trained BERT with custom final layer	Implemented and Completed Successfully	14 hours
		Create data loaders to facilitate batch processing	Implemented and Completed Successfully	23 hours
		Choose and optimizer and scheduler to control training of model	Implemented and Completed Successfully	11 hours
		Design performance metrics for our problem	Implemented and Completed Successfully	19hours
		Create a training loop to control PyTorch fine-tuning of BERT using GPU acceleration	Implemented and Completed Successfully	31 hours
		Load in a pre-saved fine-tuned BERT model and evaluate its performance	Implemented and Completed Successfully	8 hours
3	Chat bot creation	Downloading Corpus	Implemented and Completed Successfully	2 hours

		Creating Chatbot	Implemented and Completed Successfully	20 hours
		Training Chatbot with Corpus	Implemented and Completed Successfully	12 hours
4	Creating Web application	Creating AWS account	Implemented and Completed Successfully	1 hour
		Creating EC2 Windows Machine	Implemented and Completed Successfully	6 hours
		Configuring EC2 Network connections, public IP etc	Implemented and Completed Successfully	4 hours
		Installing tools pycharm	Implemented and Completed Successfully	2 hours
		Installing python	Implemented and Completed Successfully	2 hours
		installing dependences libraries	Implemented and Completed Successfully	6 hours
		Creating web files HTML,Javascript files	Implemented and Completed Successfully	34 hours
		Creating python files	Implemented and Completed Successfully	42 hours
5	Integration of Chabot and BERT Model into web application	Download trained Model and placed Model in directory	Implemented and Completed Successfully	4 hours
		Loaded Model in python file	Implemented and Completed Successfully	4 hours
		Getting Data from Web pages	Implemented and Completed Successfully	4 hours
		Passing same data to ChatterBot	Implemented and Completed Successfully	2 hours
		Passing same data to Model	Implemented and Completed Successfully	2 hours
		Getting response data from ChatterBot	Implemented and Completed Successfully	2 hours
		Sending response to Web page	Implemented and Completed Successfully	2 hours

		Based on classification from Model updating Variable	Implemented and Completed Successfully	4 hours
		Based on variables updating the graphs	Implemented and Completed Successfully	6 hours
6	Documentation	Complete end to end documentation	Implemented and Completed Successfully	200 hours

Table 2.0 plan for implementation and execution

5. IMPLEMENTATION

5.1Implementation of BERT:

BERT is set up to just utilize this [CLS] token for social occasion, we comprehend that the model has been invigorated to encode all that it necessities for the depiction step into that particular 768-respect presenting vector. On the yield of the last (twelfth) transformer, basically the first installing (diverging from the [CLS] token) is utilized by the classifier. This token has exceptional imperativeness. BERT incorporates 12 Transformer layers. Every transformer takes in a rundown of token embedding and produces a near number of embeddings on the output.

- BERT has two constraints:
  - All sentences must be padded or abbreviated to a single, fixed length.
  - The most extraordinary sentence length is 512 tokens.

- BERT has two limitations:
  - All sentences must be cushioned or shortened to a solitary, fixed length.
  - The most extreme sentence length is 512 tokens.

- The Attention Mask is basically an assortment of 1s and 0s indicating which tokens are padding and which aren't.
- Bert For Sequence Classification layers
  - The inserting layer.
  - The first of the twelve transformers.
  - The ouput layer.
- Optimizer and Learning Rate Scheduler (hyperparameters)
  - Batch size: 16, 32 (We picked 32 while making our DataLoaders).
  - Learning rate (Adam): 5e-5, 3e-5, 2e-5 (We'll use 2e-5).
  - Number of ages: 2, 3, 4 (We'll utilize 4).
  - The epsilon boundary eps: 1e-8 is a modest number to forestall any division by zero in the execution.
- For each go in our circle we have a preparation stage and an approval stage.
- Training Loop:
  - Unpack our data inputs and labels
  - Load data onto the GPU for acceleration

- Clear out the gradients calculated in the previous pass.
- In pytorch the gradients accumulate by default unless you explicitly clear them out.
- Forward pass (feed input data through the network)
- Backward pass (back propagation)
- Tell the network to update parameters with optimizer.step()
- Track variables for monitoring progress

➢ Evolution loop:

Unpack our data inputs and labels

- Load data onto the GPU for acceleration
- Forward pass (feed input data through the network)
- Compute loss on our validation data and track variables for monitoring progress [CodeEmporium, 2019]

5.2 Implementation of EC2 Window:

1. Sign up and get an access to Amazon Web Services(AWS).
2. Select 'Launch a virtual machine' with EC2(Elastic cloud) and pick an AMI. As displayed in fig (5.2.1) Microsoft sever 2019 Base is chosen as it is free tier.

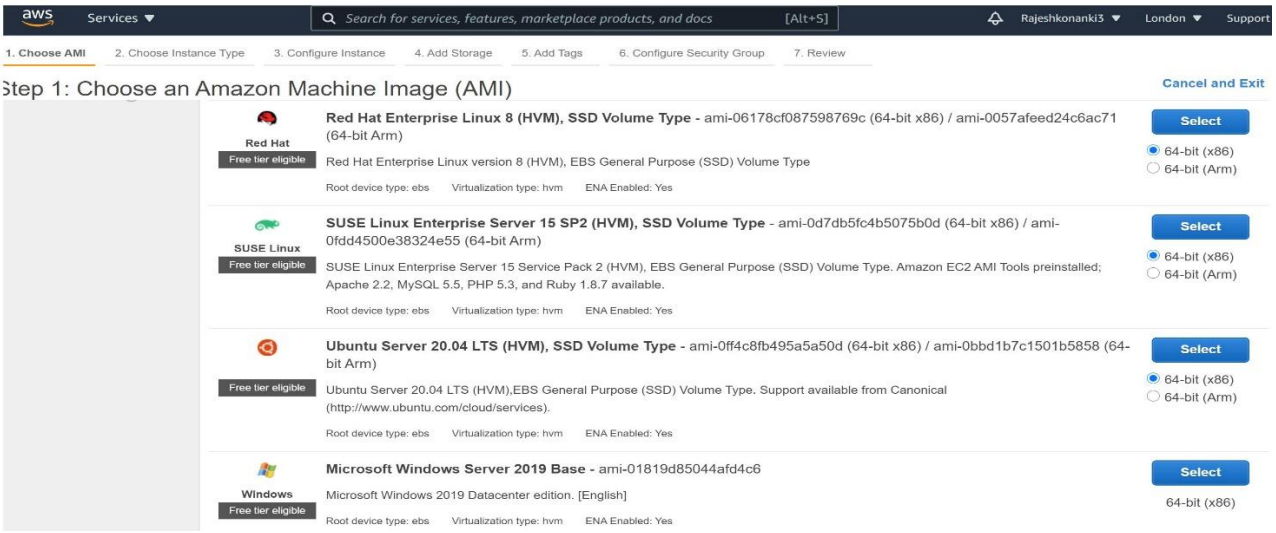


Fig 5.2.1 AMI(Amazon Machine Image)

3. Select an instance type. In this model t2.micro is chosen as it if free tier for a year and comes up with 1GB RAM and rest of them are paid.

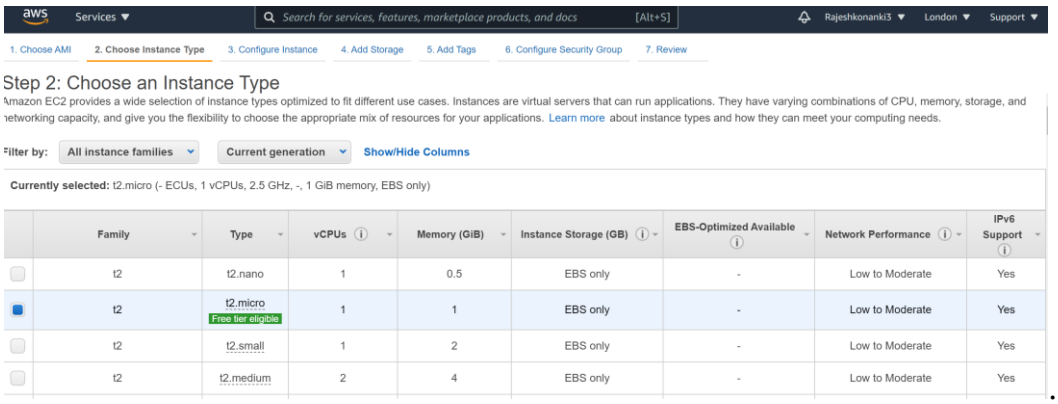


Fig 4.2.2 Instance type

4. Configure the t2.micro instance as required and add storage depends on the requirement of your application. Volume type can be changed anytime as required for the application.

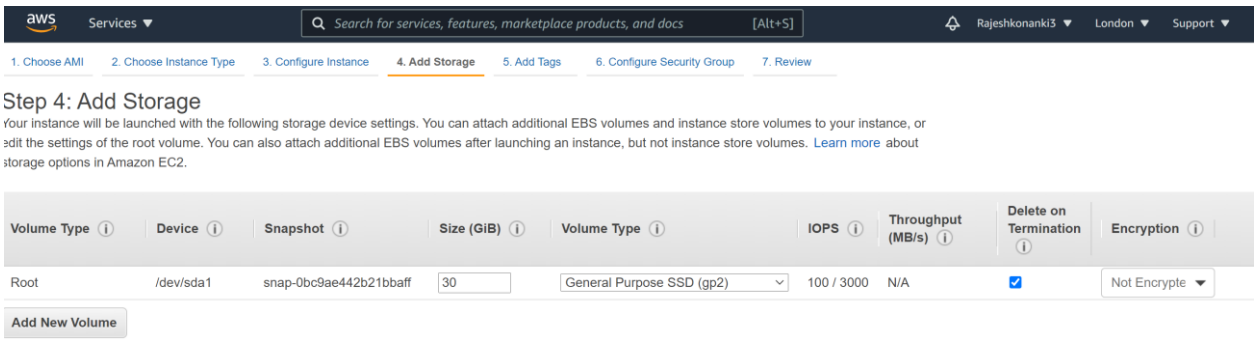
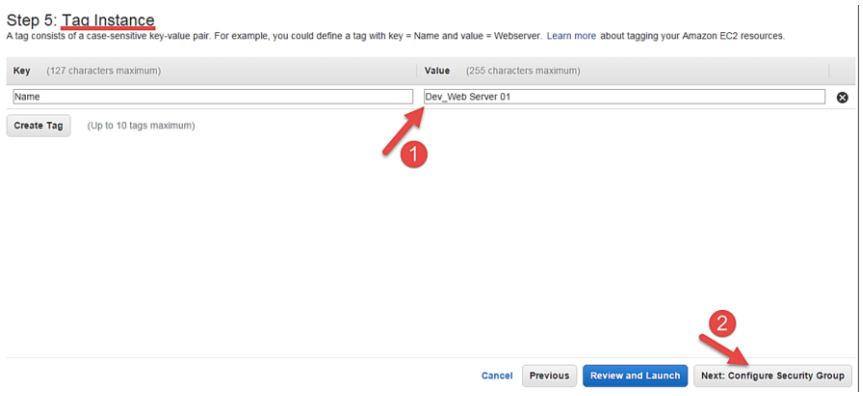


Fig (5.2.3) Storage selection

In fig (5.2.3) because of free tier we are allocated with 30gb general purpose SSD(gp2).

5.Tagging the instance and configuring security group.



Fig(5.2.4) Instance tagging

- By tagging the instance with a key pair,administrator will be having a clear view on it and can be identified if more instances are existed.
- Configure the security group to have limted access to your ip.

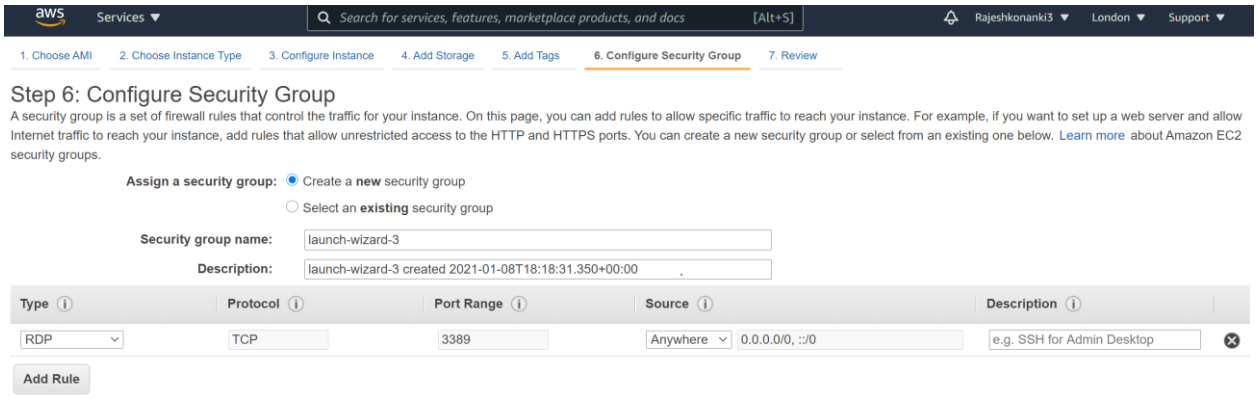


Fig (5.2.5) configuration of security group

- Select ‘create a new security group' as we are creating for the first time.
- RDP is a remote desktop protocol which is used to access desktop.

- In the source section we have two options, they are anywhere which means you can access from any ip to use the service. Whereas my ip is restricted and can be accessed from your pc ip only.
- Rules can be added to the firewall as required and launch it.
- 6. After launching it fig(5.2.6) is displayed with the figurations we have mentioned to create EC2 window.

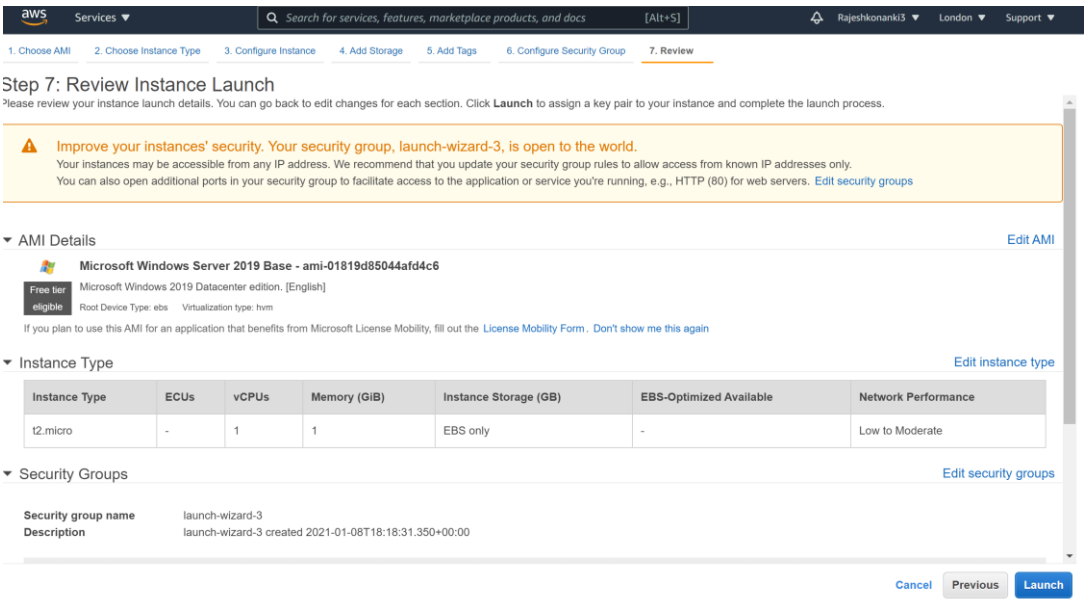


Fig 5.2.6 Review of instance

6. After launching it fig (5.2.6) it displays the configurations we have mentioned to create EC2 window. Then after fig (5.2.7) is displayed to create a key pair and name for it.

7. Download a key pair which can only be downloaded once. By using this password of the administrator window is generated.

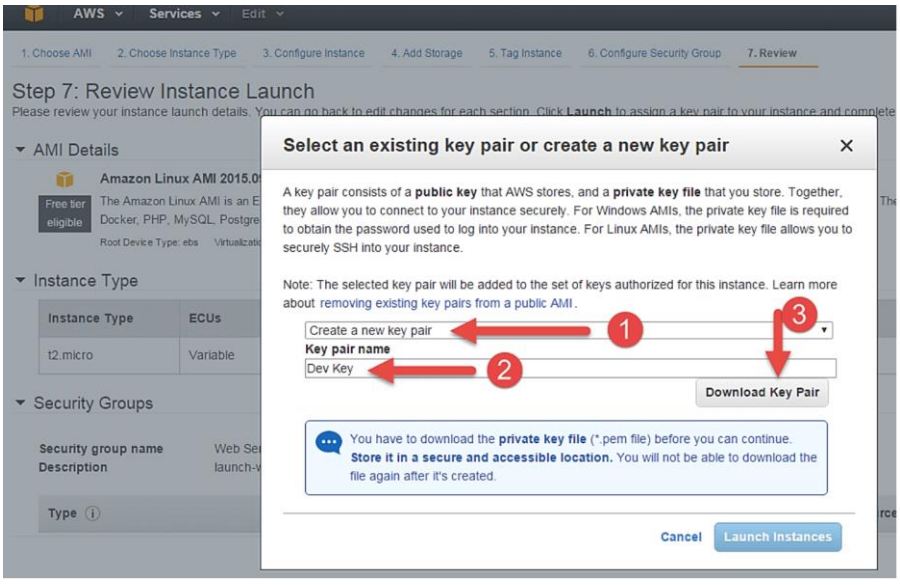
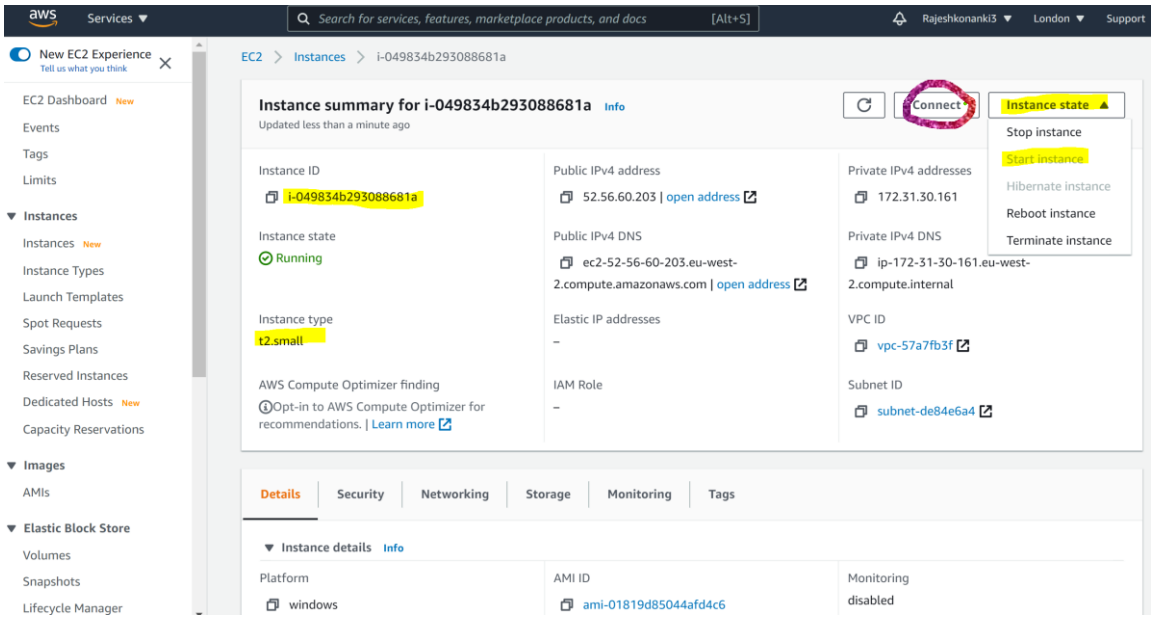


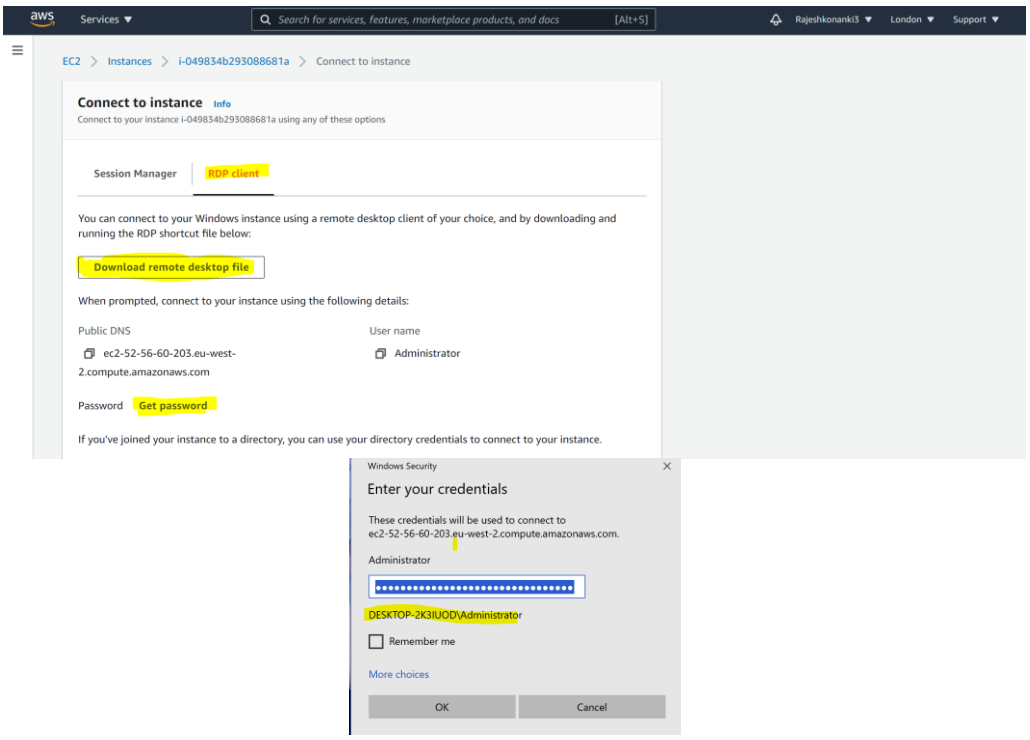
Fig 5.2.7 creating key pair

8. Now the instance is created with some ID. By selecting start instance in the instance state state highlighted. Instance starts running with ip provided as shown in the figure(5.2.8).



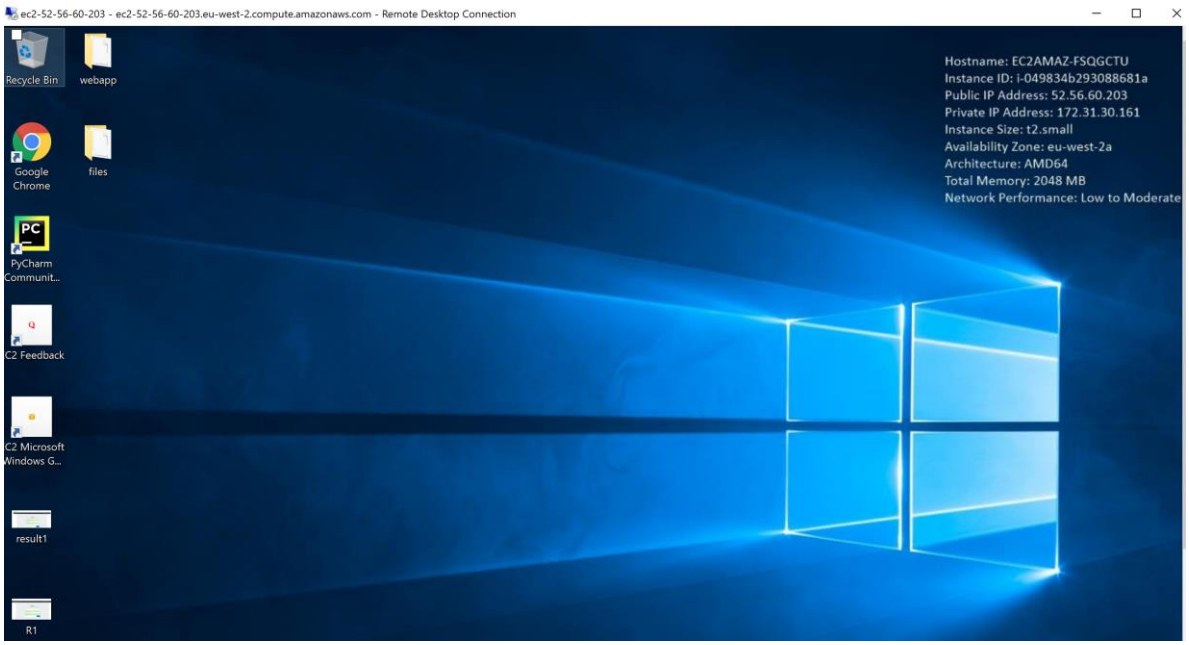
Fig(4.2.8) instance summary

9. Now connect the Instace through RDP by selecting connect button then fig(5.2.9) is displayed. By clicking download remote desktop file, desktop file will be created. To access the desktop , password has to be generated by uploading ‘key pair file’ downloaded in fig(5.2.8). Then the decrypted password is generated for secuity concern which can be further encrypted to access EC2 window.



Fig(5. 2.9) connecting to instance

10. Finally Remote desktop is accesed as displayed below.



Fig(5.2.10) AWS Ec2 window

11.Download Pycharm from ‘https://www.jetbrains.com’ and install it.
12.Download Python and install it.
13.By using pip (‘preferred installer program’) install the chatterbot with the command . Version 1.0.4 is used in this model as shown below.

```
C:\Program Files\JetBrains\PyCharm Community Edition 2020.3.1\bin>python -m chatterbot --version
1.0.4
```

Fig (5.2.11) checking version of chatterbot

12. In order to `pip install chatterbot_corpus` create chatbot we also need to install chatterbot corpus by using the command . Corpus means collection of words. Chatterbot contains data which also present in the chatbot module. Chatterbot-corpus used in this model is 0.4.2

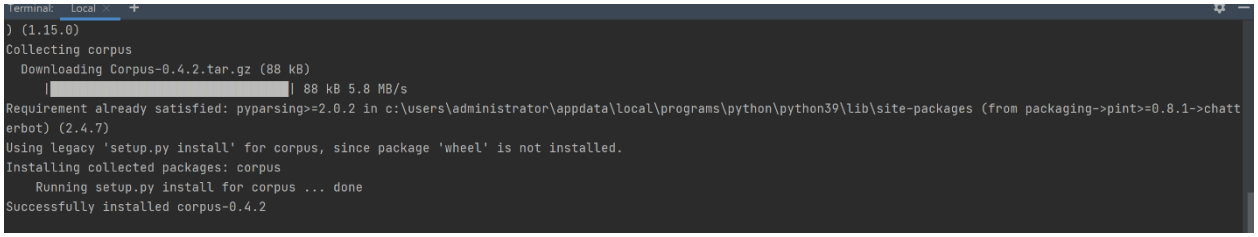


Fig (.2.12) Chatterbot_corpus

13. Install pytorch by using `pip install torch` which is a free open-source vastly used in computer vision and NLP.

14.Install transformers by using `pip install transformer`

15.Install Django by using `pip install django` to run the web application. Django version used in this model is2.1.7. The reason behind using Django in creating a web application is , it is a full stack framework.

16.After creating an app load that model from ''C:\\Users\\Administrator\\Desktop\\webapp\\Model'' and train the model.

17. By using the command `python manage.py makemigrations` we can create new migrations which are required to model.

17. `python manage.py migrate` using this we can apply and un apply migrations.

18. `python manage.py runserver` using thing we can run the web application which redirect to http link as shown below.

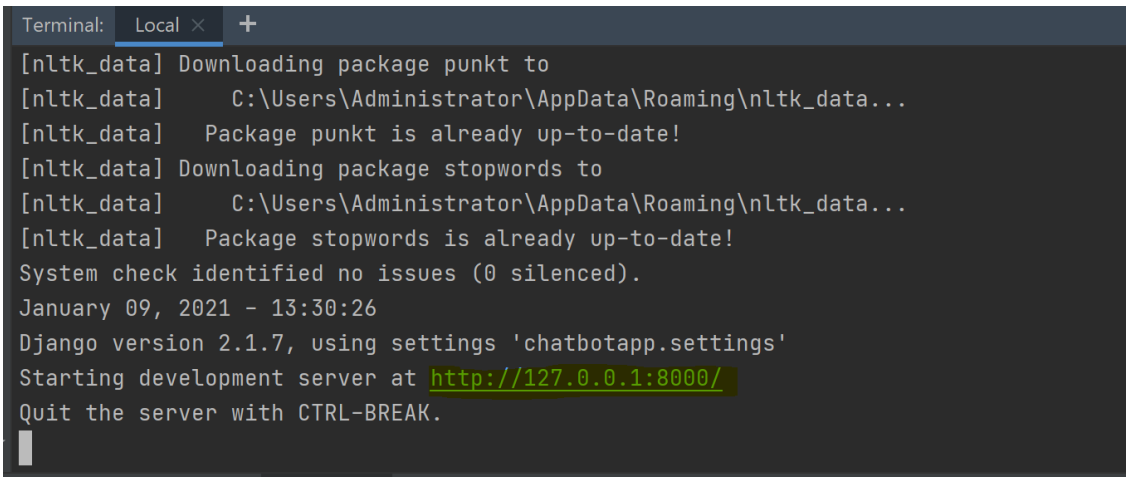


Fig (5.2.13) output executed with http link

5.3 Architecture and Working of chatbot web application:

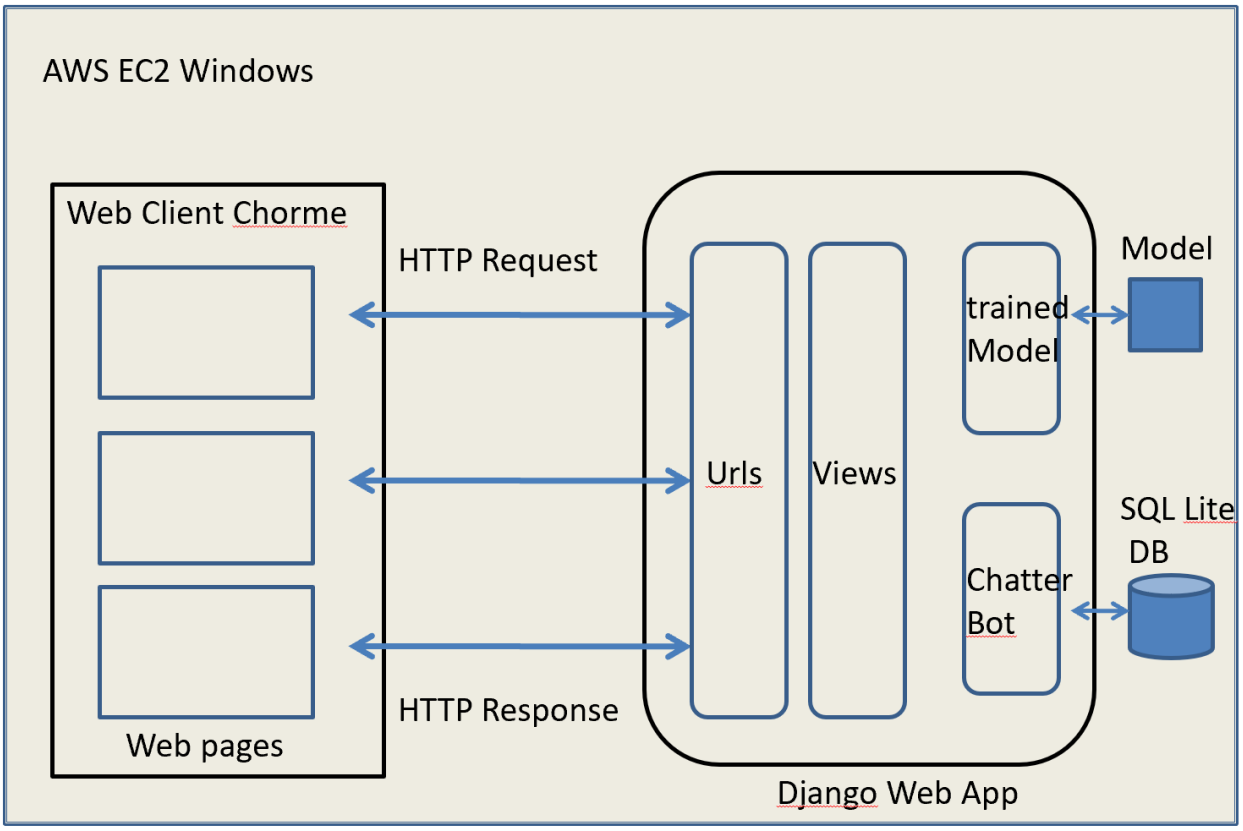


Fig 5.3.1 Architecture of chatbot web application

- Once conversation engaged with the chatbot, the 'HTTP Request' from the 'web client chrome' is sent to the 'Django Web App'.
- In 'Urls.py mapping will be present between the URI(Uniform Resource Identifier) and function to be invoked'.
- Based on mapping Urls.py will send this request to 'Views.py' where it looks for the input response.
- Furtherly response from the Views.py is sent to the trainedmodel.py and chatterbot. In trained model latest input is trained. chatterbot picks the nearest value to input from the chatterbot_corpus and sends the response to the Views.py.
- In the trained model.py we are accessing trained BERT model from folder where the model is located.
- From the Views.py, this response is sent as output to the web page where chatbot is located.
- User will be responding to the output provided by the chatbot and the above process takes place again.
- Even though the output is not appropriate for the input, chatbot is learning and getting better than before to give the nearest output.
- Likewise this model works behind the scenes.

5.4 Important steps & Model Details:

- We can't utilize the pre-tokenized variant on the grounds that, so as to apply the pre-prepared BERT, we should utilize the tokenizer gave by the model.
- This is on the grounds that
  - The model has a particular, fixed jargon.
  - The BERT tokenizer has a specific method of taking care of out-of-jargon words.
- Required Formatting
  - Add exceptional tokens to the beginning and end of each sentence.
  - Pad and shorten all sentences to a solitary consistent length.
  - Explicitly separate genuine tokens from cushioning tokens with the consideration cover.
- Special Tokens

Table 3: Table of Special Tokens

[CLS]	[SEP]	[PAD]
The primary identification of every course of action is reliably an exceptional request token [CLS]. The last hidden state identifying with this token is used as the all out gathering depiction for portrayal tasks.	Near the finish of each sentence, we have to join the unprecedented [SEP] token. This token is an old remarkableness of two-sentence attempts, where BERT is given two separate sentences and referenced to pick something.	Cushioning is done with an unprecedented [PAD] token, which is at record 0 in the BERT language.

6. Libraries and versions

absl-py == 0.9.0	keyring == 17.0.0	s3transfer == 0.3.3	django-chatbot == 0.0.2.1	preshed == 2.0.1
alabaster == 0.7.12	kiwisolver == 1.0.1	sacremoses == 0.0.43	docutils == 0.14	prometheus-client == 0.5.0
anaconda-client == 1.7.2	lazy-object-proxy == 1.3.1	scikit-image == 0.14.1	en-core-web-md == 2.1.0	prompt-toolkit == 2.0.7
anaconda-navigator == 1.9.6	libarchive-c == 2.8	scikit-learn == 0.20.1	en-core-web-sm == 2.1.0	protobuf == 3.13.0
anaconda-project == 0.8.2	llvmlite == 0.26.0	scipy == 1.1.0	entrypoints == 0.2.3	psutil == 5.4.8
asn1crypto == 0.24.0	locket == 0.2.0	seaborn == 0.9.0	et-xmlfile == 1.0.1	py == 1.7.0
astroid == 2.1.0	lxml == 4.2.5	Send2Trash == 1.5.0	fastcache == 1.0.2	pyasn1 == 0.4.8
astropy == 3.1	Markdown == 3.2.2	simplegeneric == 0.8.1	filelock == 3.0.10	pyasn1-modules == 0.2.8
astunparse == 1.6.3	MarkupSafe == 1.1.0	singledispatch == 3.4.0.3	Flask == 1.0.2	pycodestyle == 2.4.0
atomicwrites == 1.2.1	mathparse == 0.1.2	six == 1.12.0	Flask-Cors == 3.0.7	pycosat == 0.6.3
attrs == 18.2.0	matplotlib == 3.0.2	snowballstemmer == 1.2.1	future == 0.18.2	pycparser == 2.19
Babel == 2.6.0	mccabe == 0.6.1	sortedcollections == 1.0.1	gast == 0.3.3	pycrypto == 2.6.1
backcall == 0.1.0	menuinst == 1.4.14	sortedcontainers == 2.1.0	gevent == 1.3.7	pycurl == 7.43.0.2
backports.os == 0.1.1	mistune == 0.8.4	spacy == 2.1.9	glob2 == 0.6	pyflakes == 2.0.0
backports.shutil-get-terminal-size == 1.0.0	mkl-fft == 1.0.6	Sphinx == 1.8.2	google-auth == 1.20.1	pygame == 1.9.6
beautifulsoup4 == 4.6.3	mkl-random == 1.0.2	sphinxcontrib-websupport == 1.1.0	google-auth-oauthlib == 0.4.1	Pygments == 2.3.1
bitarray == 0.8.3	more-itertools == 4.3.0	spyder == 3.3.2	greenlet == 0.4.15	pylint == 2.2.2
bkcharts == 0.2	mpmath == 1.1.0	spyder-kernels == 0.3.0	grpcio == 1.31.0	pyodbc == 4.0.25
blaze == 0.11.3	msgpack == 0.5.6	SQLAlchemy == 1.3.19	h5py == 2.10.0	pyOpenSSL == 18.0.0
bleach == 3.0.2	msgpack-numpy == 0.4.3.2	sqlparse == 0.3.0	heapdict == 1.0.0	pyparsing == 2.3.0
blis == 0.2.4	multipledispatch == 0.6.0	srsly == 1.0.2	html5lib == 1.0.1	pyreadline == 2.1
bokeh == 1.0.2	murmurhash == 1.0.2	statsmodels == 0.9.0	idna == 2.8	PySocks == 1.6.8
boto == 2.49.0	mysql-connector-python == 8.0.20	sympy == 1.3	imageio == 2.4.1	pytest == 4.0.2
boto3 == 1.14.45	navigator-updater == 0.2.1	tables == 3.4.4	imagesize == 1.1.0	pytest-arraydiff == 0.3
botocore == 1.17.45	nbconvert == 5.4.0	tblib == 1.3.2	importlib-metadata == 0.6	pytest-astropy == 0.5.0
Bottleneck == 1.2.1	nbformat == 4.4.0	tensorboard == 2.3.0	ipykernel == 5.1.0	pytest-doctestplus == 0.2.0
cachetools == 4.1.1	networkx == 2.2	tensorboard-plugin-wit == 1.7.0	ipython == 7.2.0	pytest-openfiles == 0.3.1
certifi == 2018.11.29	nltk == 3.4	tensorflow-estimator == 2.3.0	ipython-genutils == 0.2.0	pytest-remotedata == 0.3.1
cffi == 1.11.5	nose == 1.3.7	termcolor == 1.1.0	ipywidgets == 7.4.2	python-dateutil == 2.8.1
chardet == 3.0.4	notebook == 5.7.4	terminado == 0.8.1	isort == 4.3.4	pytz == 2018.7
chatbotAI == 0.2.2.1	numba == 0.41.0	testpath == 0.4.2	itsdangerous == 1.1.0	PyWavelets == 1.0.1
ChatterBot == 1.1.0	numexpr == 2.6.8	thinc == 7.0.8	jdcal == 1.4	pywin32 == 223
chatterbot-corpus == 1.2.0	numpy == 1.18.5	toolz == 0.9.0	jedi == 0.13.2	pyWinhook == 1.6.1
Click == 7.0	numpydoc == 0.8.0	torch == 1.6.0	Jinja2 == 2.10	pywinpty == 0.5.5
cloudpickle == 0.6.1	oauthlib == 3.1.0	tornado == 5.1.1	jmespath == 0.10.0	PyYAML == 3.13
clyent == 1.2.2	odo == 0.5.1	tqdm == 4.28.1	joblib == 0.16.0	pyzmq == 17.1.2
colorama == 0.4.1	olefile == 0.46	traitlets == 4.3.2	jsonschema == 2.6.0	QtAwesome == 0.5.3

comtypes == 1.1.7	openpyxl == 2.5.12	transformers == 2.1.1	jupyter == 1.0.0	qtconsole == 4.4.3
conda == 4.8.4	opt-einsum == 3.3.0	ujson == 1.35	jupyter-client == 5.2.4	QtPy == 1.5.2
conda-build == 3.17.6	packaging == 18.0	unicodecsv == 0.14.1	jupyter-console == 6.0.0	regex == 2019.6.5
conda-package-handling == 1.3.11	pandas == 0.23.4	urllib3 == 1.24.1	jupyter-core == 4.4.0	requests == 2.21.0
conda-verify == 3.1.1	pandocfilters == 1.4.2	wasabi == 0.8.0	jupyterlab == 0.35.3	requests-oauthlib == 1.3.0
contextlib2 == 0.5.5	parso == 0.3.1	wcwidth == 0.1.7	jupyterlab-server == 0.2.0	rope == 0.11.0
cryptography == 2.4.2	partd == 0.3.9	webencodings == 0.5.1	Keras == 2.4.3	rsa == 4.6
cycler == 0.10.0	path.py == 11.5.0	Werkzeug == 0.14.1	Keras-Preprocessing == 1.1.2	ruamel-yaml == 0.15.46
cymem == 2.0.2	pathlib2 == 2.3.3	widgetsnbextension == 3.4.2	pluggy == 0.8.0	xlwt == 1.3.0
Cython == 0.29.2	patsy == 0.5.1	win-inet-pton == 1.0.1	ply == 3.11	zict == 0.1.3
cytoolz == 0.9.0.1	pep8 == 1.7.1	win-unicode-console == 0.5	defusedxml == 0.5.0	pkginfo == 1.4.2
dask == 1.0.0	pickleshare == 0.7.5	wincertstore == 0.2	dill == 0.2.9	plac == 0.9.6
datashape == 0.5.4	Pillow == 5.3.0	wrapt == 1.10.11	distributed == 1.25.1	XlsxWriter == 1.1.2
decorator == 4.3.0	Pint == 0.16.1	xlrd == 1.2.0	Django == 2.2.4	xlwings == 0.15.1

Table 4.0 Libraries used.

7. Analysis of DATASET for BERT Model

The dataset used in to train the BERT model is descripted in the Url below [Mohammed, S., 2017].

Url: http://saifmohammad.com/WebPages/EmotionIntensity-SharedTask.html

Column Names:

· Id
· Sentence
· Emotion –{ anger,fear,joy,sadness}
· Intensity

```
[9]: df.sample(20)
```

[9]:

	id	sentence	emotion	intensity	label
424	20424	Americans as a whole are, for the most part, f...	fear	0.562	1
1043	21043	Round 2 #pcola	fear	0.229	1
366	10366	@Nigerianscamsss @TrillSmith you think I came ...	anger	0.521	0
67	30067	@ChrisBooker @ThePerezHilton On PHP54 and tho...	joy	0.808	2
714	30714	@EngrNabeeel then rejoicing now is a lil' bit ...	joy	0.250	2
530	11471	Goddamn headache.	anger	0.646	0
126	11067	Ok but how are the fast & furious movies N...	anger	0.292	0
67	11008	After what just happened. In need to smoke.	anger	0.479	0
300	41160	@damiandmusic hubby Noah shares this somber th...	sadness	0.312	3
114	20114	@SusannahSpot I could pop round #nightmare	fear	0.771	1
912	22169	#internationaldayofpeace : When white supremac...	fear	0.750	1
310	40310	Angry shouting match between a #pessimist &amp...	sadness	0.542	3
693	10693	I can't even right now #bb18	anger	0.354	0
655	30655	Well look at the bright side. You found a use ...	joy	0.312	2
712	30712	Halfway to work and I realize I forgot to put ...	joy	0.250	2
1069	21069	If you think you're good to go already, don't ...	fear	0.208	1
680	11621	Stk is expensive but i'f rather take a bigger ...	anger	0.375	0
799	30799	@LaurenBrierley2 sparkling water = death	joy	0.125	2

Figure 7.1. Sample Rows of Dataset

❖ Samples are almost equally distributed so that, the trained model will not be biased.

```
[68]:  1  df['emotion'].value_counts().plot(kind='pie')

[68]: <matplotlib.axes._subplots.AxesSubplot at 0x211d44be7f0>
```

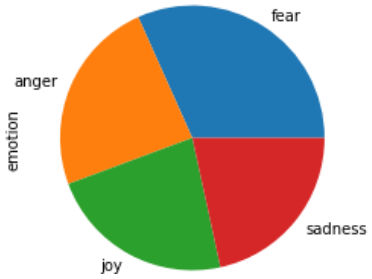


Fig 7.2 Distribution of rows based on Emotion

8 RESULTS

8.1 BERT Model Results:

Epoch 1:

When the BERT model is trained for the first time with dataset, the average loss is 1.33 and epoch training took 0:02:22 time to train the data. The accuracy achieved is 57% time took for running the validation is 6seconds as shown in fig(7.4).

```
======== Epoch 1 / 4 ========
Training...
  Batch    40  of    112.    Elapsed: 0:00:51.
  Batch    80  of    112.    Elapsed: 0:01:42.

  Average training loss: 1.33
  Training epcoh took: 0:02:22

Running Validation...
  Accuracy: 0.57
  Validation took: 0:00:06
```

Fig 7.3 Epoch 1

Epoch 2:

When the BERT model is trained further with same dataset, the average loss is 0.73 and epoch training took 0:02:22 time to train the data. The accuracy achieved is 81% time took for running the validation is 6seconds. Loss is reduced from 1.33 to 0.73 as shown in fig(7.4).

```
======== Epoch 2 / 4 ========
Training...
  Batch    40  of    112.    Elapsed: 0:00:51.
  Batch    80  of    112.    Elapsed: 0:01:42.

  Average training loss: 0.73
  Training epcoh took: 0:02:22

Running Validation...
  Accuracy: 0.81
  Validation took: 0:00:06
```

7

Fig 7.4 Epoch 2

Epoch 3:

When the BERT model is trained further with same dataset, the average loss is 0.73 and epoch training took 0:02:22 time to train the data. The accuracy achieved is 83% time took for running the validation is 6seconds. Loss is reduced from 1.33 to 0.73 as shown in fig (7.5).

```
======== Epoch 3 / 4 ========
Training...
  Batch    40  of    112.    Elapsed: 0:00:51.
  Batch    80  of    112.    Elapsed: 0:01:42.

  Average training loss: 0.38
  Training epcoh took: 0:02:22

Running Validation...
  Accuracy: 0.83
  Validation took: 0:00:06
```

Fig 7.5 Epoch 3

Epoch 4:

When the BERT model is trained further with same dataset, the average loss is 0.73 and epoch training took 0:02:22 time to train the data. The accuracy achieved is 83% time took for running the validation is 6seconds. Loss is reduced from 0.38 to 0.28 as shown in fig (7.6).

```
======== Epoch 4 / 4 ========
Training...
  Batch    40  of    112.    Elapsed: 0:00:51.
  Batch    80  of    112.    Elapsed: 0:01:42.

  Average training loss: 0.28
  Training epcoh took: 0:02:22

Running Validation...
  Accuracy: 0.83
  Validation took: 0:00:06

Training complete!
```

Fig 7.6 Epoch 4

On observing all the results, it is noticed that at every stage the loss is getting minimised and accuracy is increasing. Time consumed to train is almost same at all the stages. Train the model over fitting and outputs will not be very real. As shown in the fig(7.7) training the model at different loss may reduce, But model leads to overfitiing.

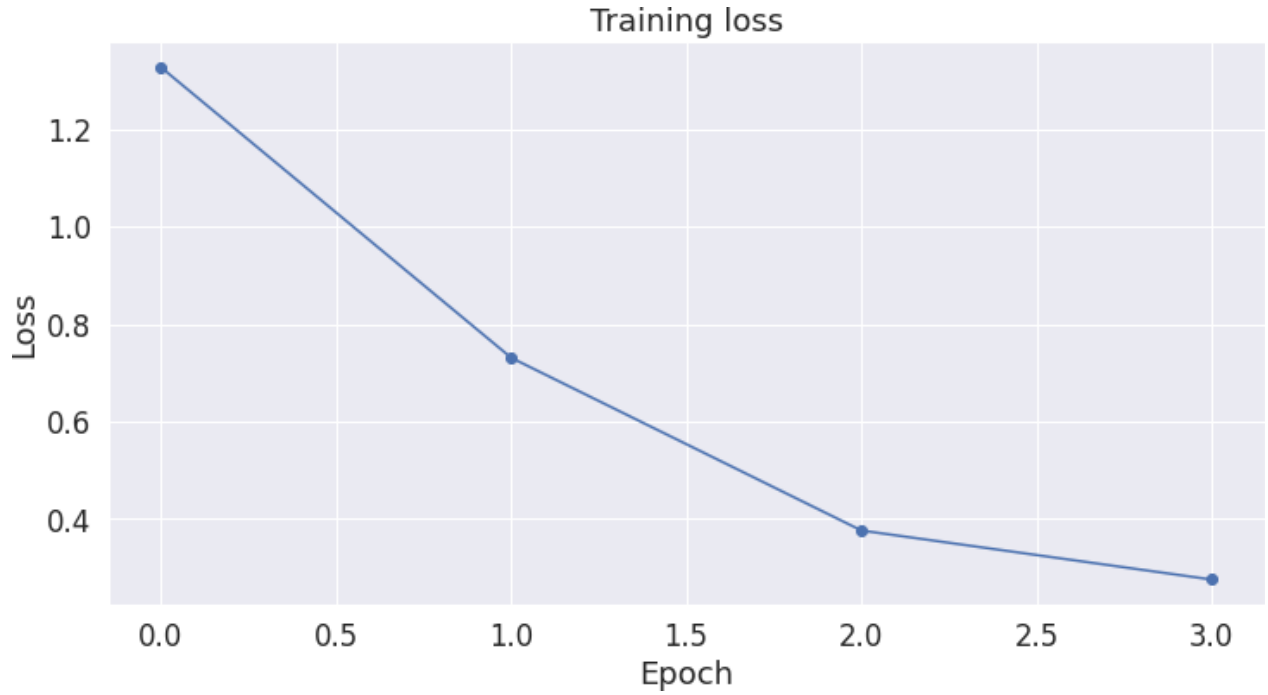


Fig 7.7. Training Loss vs Epoch

Different layers of the BERT and their parameters are illustrated in the fig (7.8).

```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                  (30522, 768)
bert.embeddings.position_embeddings.weight                (512, 768)
bert.embeddings.token_type_embeddings.weight                (2, 768)
bert.embeddings.LayerNorm.weight                              (768,)
bert.embeddings.LayerNorm.bias                                (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight          (768, 768)
bert.encoder.layer.0.attention.self.query.bias                (768,)
bert.encoder.layer.0.attention.self.key.weight            (768, 768)
bert.encoder.layer.0.attention.self.key.bias                  (768,)
bert.encoder.layer.0.attention.self.value.weight          (768, 768)
bert.encoder.layer.0.attention.self.value.bias                (768,)
bert.encoder.layer.0.attention.output.dense.weight        (768, 768)
bert.encoder.layer.0.attention.output.dense.bias              (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight        (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias          (768,)
bert.encoder.layer.0.intermediate.dense.weight           (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                (3072,)
bert.encoder.layer.0.output.dense.weight                 (768, 3072)
bert.encoder.layer.0.output.dense.bias                        (768,)
bert.encoder.layer.0.output.LayerNorm.weight                  (768,)
bert.encoder.layer.0.output.LayerNorm.bias                    (768,)

==== Output Layer ====

bert.pooler.dense.weight                                  (768, 768)
bert.pooler.dense.bias                                        (768,)
classifier.weight                                           (4, 768)
classifier.bias                                                 (4,)
```

Fig 7.8 Layers and its shape details of BERT Model

8.2 Web Application Results:

Page 1: On clicking the http link in fig(5.2.13) we are redirected to chatbot web page to have a interaction with chatbot as show in fig(8 2.1) .

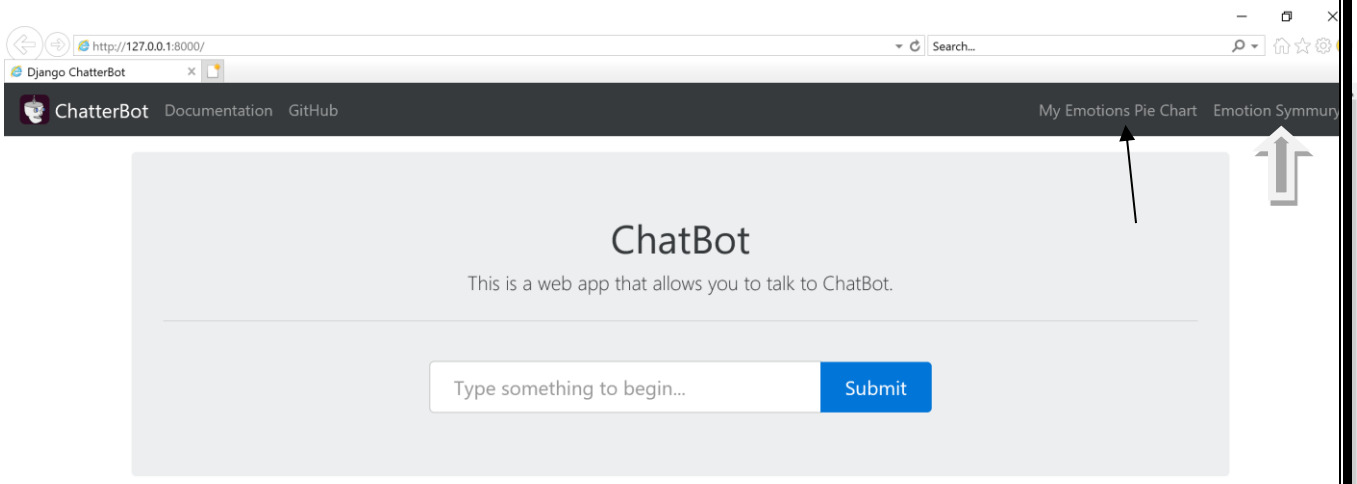


fig 8. 2.1 page to communicate with chatbot

After having enough conversation with chatbot we can get a pie chart of emotions carried throughout the conversation with chatbot just by clicking 'My Emotions Pie Chart' in fig(8.2.1).

Page 2:

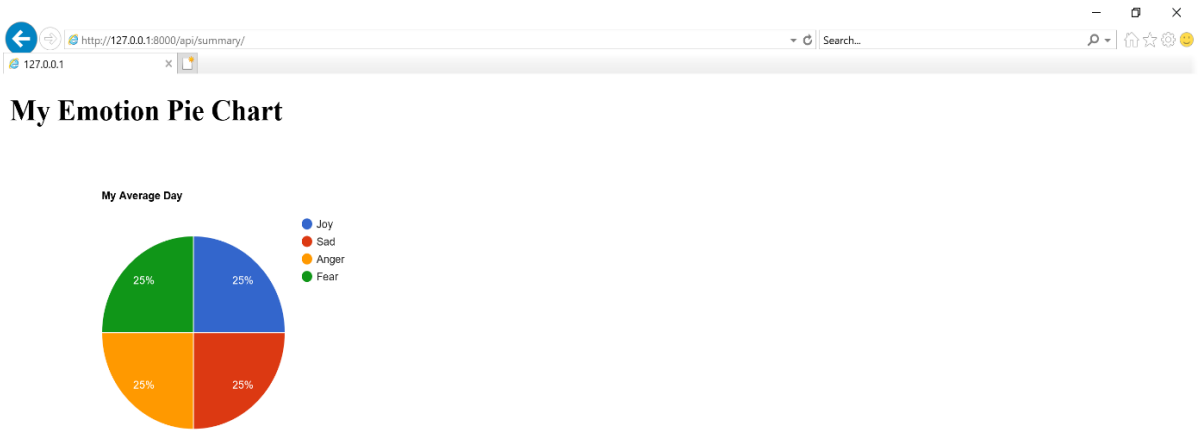


Fig 8.2.2 pie chart showing emotions after conversation.

- Fig (8.2.2) illustrates the emotions carried out by the user in a day on an average. Blue color represents joy %, Red color shows sad%, whereas green denotes Fear and orange indicates percentage of anger.
- By clicking 'Emotion Summary' as marked in fig (8.2.2) we can get summary of

emotions, for example how many times user is sad? How many times user is happy, angry and fear depends on the context of conversation with chatbot. Results of the emotion summary is figured in the fig (8.2.3).

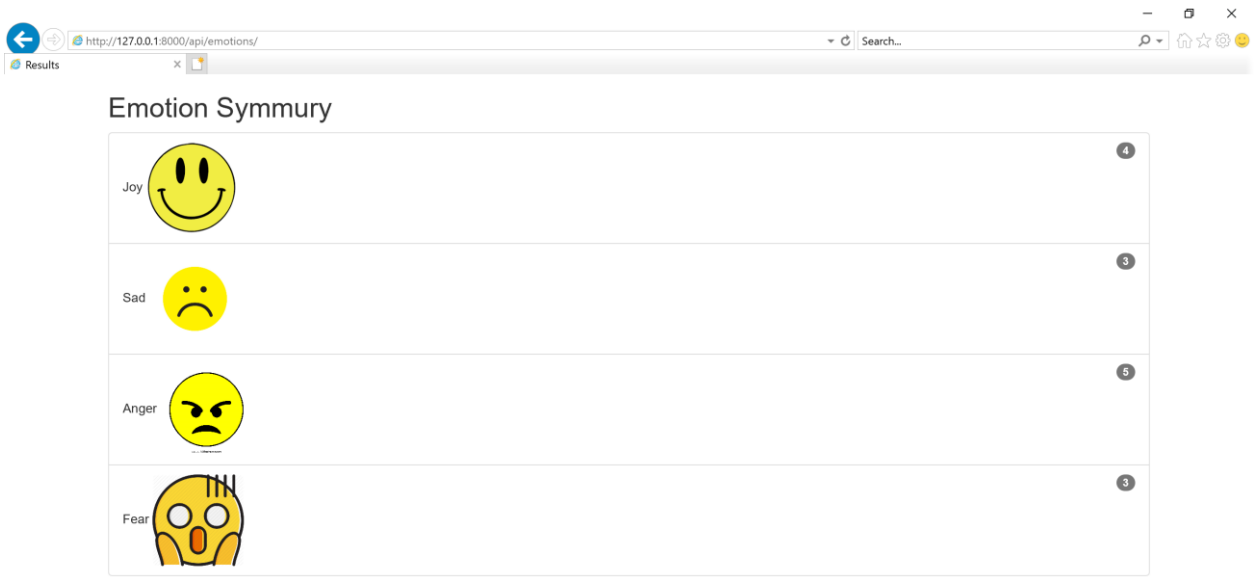


Fig 8.23 Emotion Summary

Limitations Noticed:

- Usage limited to one person.
- Taking long time to train the model with limited resources.
- Accuracy can be improved further.
- Response time can be improved.

9. CONCLUSION AND EVOLUTION

Chatbots becoming more used in every organizations, they are being very useful in automating so many aspects. In this project we have achieved hoping achieved performing the emotion analysis using chatbot. Idea is to use two trained models to achieve task of emotion analysis. One model is used to identify the emotion of sentence and other is used to response to the sentence. And both these of these models will are used in web application. The web application is provided to interface with chat and present the outcomes in graphical manner. For sentiment analysis, pre-trained BERT model provide contextual tokenization and adding new layer of classification sentiment analysis capability. The Chatterbot API provided with inbuilt database and inbuilt functionality to handle the training and deploying the model.

During Covid pandemic and due to lockdown everywhere, it makes distress to the humans where our chatbot helps in knowing that person's emotion is carrying. It doesn't harm a person while using, which is an ethical-moral of this project. In contrast, this will be communicating in a friendly way which helps in socialising with the people. By considering all the elements analysing person emotions are analysed, a decision can be taken in the right way to get back to a normal position.

Future Enhancements:

As of now this model this model is limited to one person at a time. This can be developed for the multiple users with different data sets by using more GPU's and more layers of BERT model. Till now these types of chatbots are used for the general enquiries in the organization with the structured data and fixed question and answer analyses. But our chatbot can be used in the real time in the web or mobile applications and giving suggestions to user to make his emotions better. For example when chatbot triggers that he is sad send him motivational quotes and if he is happy sending jokes as a notification.

10.APPENDICES

As the code is huge, I reposited thecode in the github and made it public and shared the link below.

https://github.com/rajeshkonanki3/chatbot.git

Code used in this model is as follows [Huggingface.2020]:

CODING:

Trained_model.py:

```
from transformers import BertTokenizer
from transformers import BertForSequenceClassification
import torch


class predictEmotion(object):
   def __init__(self):
      super()
      self.model =
BertForSequenceClassification.from_pretrained('C:\\Users\\Administrator\\Desktop\\webapp\\Model',
num_labels=4)
      self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

   def perdt(self,sent):
      # Tokenize all of the sentences and map the tokens to thier word IDs.
      input_ids = []
      #sent='The race advances only by the extra achievements of the individual. You are the individual. ~Charles
Towne\n  #inspire'
      # For every sentence...
      # `encode` will:
      #   (1) Tokenize the sentence.
      #   (2) Prepend the `[CLS]` token to the start.
      #   (3) Append the `[SEP]` token to the end.
      #   (4) Map tokens to their IDs.
      encoded_sent = self.tokenizer.encode(sent,add_special_tokens = True,)
      input_ids.append(encoded_sent)
      attention_masks = []

      # Create a mask of 1s for each token followed by 0s for padding
      for seq in input_ids:
         seq_mask = [float(i > 0) for i in seq]
         attention_masks.append(seq_mask)

      prediction_inputs = torch.tensor(input_ids)
      prediction_masks = torch.tensor(attention_masks)

      outputs = self.model(prediction_inputs, token_type_ids=None,
            attention_mask=prediction_masks)
      emotion = ['Anger', 'Fear', 'Joy', 'Sad']
      logits = outputs[0]
      #print(torch.abs(logits))
```

```
    x=torch.argmax(torch.abs(logits))
    #print(torch.argmax(torch.abs(logits)))
    #print(emotion[int(x.item())])
    return emotion[int(x.item())]
```

Setting.py:

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.8/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'fsch+6!=q+@ol&%0x!nwdl@48^ixbd4clx5f1i!5n^66y+pmn*'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = (
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'chatterbot.ext.django_chatterbot',
  'chatbotapp',
)

# ChatterBot settings

CHATTERBOT = {
  'name': 'Django ChatterBot Example',
  'django_app_name': 'django_chatterbot',

  'trainer': 'chatterbot.trainers.ChatterBotCorpusTrainer',
  'training_data': [
      'chatterbot.corpus.english.greetings',
      'chatterbot.corpus.english.conversations'
  ]
}


MIDDLEWARE = (
  'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'django.middleware.common.CommonMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    #'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
    #'django.contrib.sessions.middleware.SessionMiddleware',
    #'django.contrib.messages.middleware.MessageMiddleware',
    #'django.contrib.auth.middleware.AuthenticationMiddleware',

)

ROOT_URLCONF = 'chatbotapp.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'chatbotapp.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.8/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Internationalization
# https://docs.djangoproject.com/en/1.8/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(
        os.path.dirname(__file__),
        'static',
    ),
)
```

Views.py:

```
import json
from django.views.generic.base import TemplateView
from django.views.generic import View
from django.http import JsonResponse
from chatterbot import ChatBot
from chatterbot.ext.django_chatterbot import settings
from django.shortcuts import render


from chatbotapp.trainedmodel import predictEmotion

Sad=1
Joy=1
Anger=1
Fear=1




class ChatterBotAppView(TemplateView):
    template_name = 'app.html'

https://chatterbot.readthedocs.io/
class ChatterBotApiView(View):
    """
    Provide an API endpoint to interact with ChatterBot.
    """
    pe = predictEmotion()

    #def __init__(self):
    #    self.pe = predictEmotion()
    #    print('created.....obj')
    #    super()

    chatterbot = ChatBot(**settings.CHATTERBOT)


    def post(self, request, *args, **kwargs):
```

```
        """
        Return a response to the statement in the posted data.
        * The JSON data should contain a 'text' attribute.
        """
        input_data = json.loads(request.body.decode('utf-8'))

        if 'text' not in input_data:
            return JsonResponse({
                'text': [
                    'The attribute "text" is required.'
                ]
            }, status=400)

        response = self.chatterbot.get_response(input_data)

        response_data = response.serialize()


        emotion = ChatterBotApiView.pe.perdt(input_data['text'])

        if emotion == 'Fear':
            global Fear
            Fear+=1
        elif emotion == 'Sad':
            global Sad
            Sad += 1
        elif emotion == 'Joy':
            global Joy
            Joy += 1
        elif emotion == 'Anger':
            global Anger
            Anger += 1
        else:
            pass

        #print(emotion)

        #print(id(self.pe))

        return JsonResponse(response_data, status=200)

    def get(self, request, *args, **kwargs):
        """
        Return data corresponding to the current conversation.
        """
        return JsonResponse({
            'name': self.chatterbot.name
        })

class EmotionAppView(TemplateView):

    def get(self, request, *args, **kwargs):
        """
        Return data corresponding to the current conversation.
        """
        context = {"e1":Joy,
```

```
            "e2": Sad,
            "e3": Anger,
            "e4": Fear}
        template_name = 'Emotion.html'
        return render(request, template_name, context)




class SummaryAppView(TemplateView):

    def get(self, request, *args, **kwargs):
        """
        Return data corresponding to the current conversation.
        """
        context = {"e1":Joy,
            "e2": Sad,
            "e3": Anger,
            "e4": Fear}
        template_name = 'Summary.html'
        return render(request, template_name, context)
```

Urls.py:

```
from django.conf.urls import url
from django.contrib import admin
from chatbotapp.views import ChatterBotAppView, ChatterBotApiView,EmotionAppView,SummaryAppView

urlpatterns = [
    url(r'^$', ChatterBotAppView.as_view(), name='main'),
    url(r'^admin/', admin.site.urls, name='admin'),
    url(r'^api/chatterbot/', ChatterBotApiView.as_view(), name='chatterbot'),
    url(r'^api/emotions/', EmotionAppView.as_view(), name='emotions'),
    url(r'^api/summary/', SummaryAppView.as_view(), name='summary'),
]
```

train.py:

```
from django.core.management.base import BaseCommand


class Command(BaseCommand):
    """
    A Django management command for calling a
    chat bot's training method.
    """

    help = 'Trains the database used by the chat bot'
    can_import_settings = True

    def handle(self, *args, **options):
        from chatterbot import ChatBot
        from chatterbot.ext.django_chatterbot import settings
        from chatterbot.trainers import ChatterBotCorpusTrainer
```

```
    chatterbot = ChatBot(**settings.CHATTERBOT)

    trainer = ChatterBotCorpusTrainer(chatterbot)

    trainer.train(*settings.CHATTERBOT['training_data'])

    # Django 1.8 does not define SUCCESS
    if hasattr(self.style, 'SUCCESS'):
        style = self.style.SUCCESS
    else:
        style = self.style.NOTICE

    self.stdout.write(style('Starting training...'))
    training_class = trainer.__class__.__name__
    self.stdout.write(style('ChatterBot trained using "%s"' % training_class))
```

manage.py:

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "chatbotapp.settings")

    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)
```

app.html :

```
{% load staticfiles %}
<!doctype html>
<html>
  <head>
    <title>Django ChatterBot </title>
    <link rel="stylesheet" href="{% static 'css/bootstrap.css' %}"/>
    <link rel="stylesheet" href="{% static 'css/custom.css' %}"/>
  </head>
  <body>

    {% include 'nav.html' %}

    <div class="container">

      <div class="jumbotron mt-1">
        <h1 class="jumbotron-heading text-xs-center">ChatBot </h1>
        <p class="lead text-xs-center">
          This is a web app that allows you to talk to ChatBot.
        </p>

        <hr class="my-2">
```

```
    <div class="row">
      <div class="col-xs-6 offset-xs-3">
        <ul class="list-group chat-log js-chat-log">
        </ul>

        <div class="input-group input-group-lg mt-1">
          <input type="text" class="form-control js-text" placeholder="Type something to begin..."/>
          <span class="input-group-btn">
            <button class="btn btn-primary js-say">Submit</button>
          </span>
        </div>

      </div>
    </div>

  </div>

</div>

<script src="{% static 'js/jquery.js' %}"></script>
<script src="{% static 'js/js.cookie.js' %}"></script>
<script src="{% static 'js/bootstrap.js' %}"></script>
<script>
  var chatterbotUrl = '{% url "chatterbot" %}';
  var csrftoken = Cookies.get('csrftoken');

  function csrfSafeMethod(method) {
    // these HTTP methods do not require CSRF protection
    return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
  }

  $.ajaxSetup({
    beforeSend: function(xhr, settings) {
      if (!csrfSafeMethod(settings.type) && !this.crossDomain) {
        xhr.setRequestHeader("X-CSRFToken", csrftoken);
      }
    }
  });

  var $chatlog = $('.js-chat-log');
  var $input = $('.js-text');
  var $sayButton = $('.js-say');

  function createRow(text) {
    var $row = $('<li class="list-group-item"></li>');

    $row.text(text);
    $chatlog.append($row);
  }

  function submitInput() {
    var inputData = {
      'text': $input.val()
    }
```

```
      // Display the user's input on the web page
      createRow(inputData.text);

      var $submit = $.ajax({
        type: 'POST',
        url: chatterbotUrl,
        data: JSON.stringify(inputData),
        contentType: 'application/json'
      });

      $submit.done(function(statement) {
        createRow(statement.text);

        // Clear the input field
        $input.val('');

        // Scroll to the bottom of the chat interface
        $chatlog[0].scrollTop = $chatlog[0].scrollHeight;
      });

      $submit.fail(function() {
        // TODO: Handle errors
      });
    }

    $sayButton.click(function() {
      submitInput();
    });

    $input.keydown(function(event) {
      // Submit the input when the enter button is pressed
      if (event.keyCode == 13) {
        submitInput();
      }
    });
  </script>
 </body>
</html>
```

nav.html:

```
{% load staticfiles %}

<nav class="navbar navbar-dark bg-inverse navbar-full">

 <a class="navbar-brand" href="{% url 'main' %}">
  <img src="{% static 'img/chatterbot.png' %}" width="30" height="30" class="d-inline-block align-top"
alt="ChatterBot">
  ChatterBot
 </a>
```

```
 <ul class="nav navbar-nav">
  <li class="nav-item">
   <a class="nav-link" href="https://chatterbot.readthedocs.io">Documentation</a>
  </li>
  <li class="nav-item">
   <a class="nav-link" href="https://github.com/gunthercox/ChatterBot">GitHub</a>
  </li>
 </ul>

 <ul class="nav navbar-nav float-xs-right">
  <li class="nav-item">
   <a class="nav-link" href="/api/summary/">My Emotions Pie Chart</a>
  </li>
  <li class="nav-item">
   <a class="nav-link" href="/api/emotions/">Emotion Symmury</a>
  </li>
 </ul>
</nav>
```

Emotion.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Results</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
 <h2>Emotion Symmury</h2>
 <ul class="list-group">
  <li class="list-group-item">Joy
  <span class="badge">{{e1}}</span>
  <img src="/static/img/Happy.png" class="img-rounded" alt="Cinque Terre" width="100" height="100">
  </li>
  <li class="list-group-item">Sad
  <span class="badge">{{e2}}</span>
  <img src="/static/img/sad.jpg" class="img-rounded" alt="Cinque Terre" width="100" height="100">
  </li>
  <li class="list-group-item">Anger
  <span class="badge">{{e3}}</span>
  <img src="/static/img/anger.jpg" class="img-rounded" alt="Cinque Terre" width="100" height="100">
  </li>
  <li class="list-group-item">Fear
  <span class="badge">{{e4}}</span>
  <img src="/static/img/fear.png" class="img-rounded" alt="Cinque Terre" width="100" height="100">
  </li>
 </ul>
</div>
```

```
</body>
</html>
```

summary.html :

```
<!DOCTYPE html>
<html lang="en-US">
<body>

<h1>My Emotion Pie Chart</h1>

<div id="piechart"></div>

<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>

<script type="text/javascript">
// Load google charts
google.charts.load('current', {'packages':['corechart']});
google.charts.setOnLoadCallback(drawChart);

// Draw the chart and set the chart values

function drawChart() {
  var data = google.visualization.arrayToDataTable([
  ['Task', 'Hours per Day'],
  ['Joy', {{e1}}],
  ['Sad', {{e2}}],
  ['Anger', {{e3}}],
  ['Fear', {{e4}}],
  ]);

  // Optional; add a title and set the width and height of the chart
  var options = {'title':'My Average Day', 'width':550, 'height':400};

  // Display the chart inside the <div> element with id="piechart"
  var chart = new google.visualization.PieChart(document.getElementById('piechart'));
  chart.draw(data, options);
}
</script>

</body>
</html>
```

11. REFERENCES

- GreyCampus, 2019. *Introduction To Amazon Web Services| What Is AWS| Greycampus*. [video] Available at: <https://www.youtube.com/watch?v=uRM4258HXbI> [Accessed 16 December 2020].

- CodeEmporium, 2019. *BERT Neural Network - EXPLAINED!*. [video] Available at: <https://www.youtube.com/watch?v=xI0HHN5XKDobert> [Accessed 9 December 2020].

- Choudhury, A., 2020. *Part 2: BERT Fine-Tuning Tutorial With Pytorch For Text Classification On The Corpus Of Linguistic…*. [online] Medium. Available at: <https://medium.com/@aniruddha.choudhury94/part-2-bert-fine-tuning-tutorial-with-pytorch-for-text-classification-on-the-corpus-of-linguistic-18057ce330e1> [Accessed 13 December 2020].

- Codecademy. 2020. *What Is Sqlite? | Codecademy*. Available at: <https://www.codecademy.com/articles/what-is-sqlite> [Accessed 20 December 2020].

- Sqlite.org. n.d. *About Sqlite*. Available at: <https://www.sqlite.org/about.html> [Accessed 20 December 2020].

- www.javatpoint.com. 2021. *Sqlite Advantages And Disadvantages - Javatpoint*. Available at: <https://www.javatpoint.com/sqlite-advantages-and-disadvantages> [Accessed 20 December 2020].

- Shorif Uddin, M., Chand Bansal, J., Shorif Uddin, M., Chand Bansal, J., University, J. and University, S., 2018. *Proceedings Of International Joint Conference On Computational Intelligence | Springerlink*. Link.springer.com. Available at: <https://link.springer.com/book/10.1007%2F978-981-13-7564-4> [Accessed 8 January 2021].

- En.wikipedia.org. 2021. *Transformer (Machine Learning Model)*. Available at: <https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)> [Accessed 24 October 2020].

- Giuseppe, M., 2020. *Detect Eye Disease With Pytorch*. Medium. Available at: <https://medium.com/swlh/detect-eye-disease-with-pytorch-fc0b53ff49f9> [Accessed 29 september 2020].

- GreyCampus, 2019. *Introduction To Amazon Web Services| What Is AWS| Greycampus*. [video] Available at: <https://www.youtube.com/watch?v=uRM4258HXbI> [Accessed 16 December 2020].

- CodeEmporium, 2019. *BERT Neural Network - EXPLAINED!*. [video] Available at: <https://www.youtube.com/watch?v=xI0HHN5XKDobert> [Accessed 9 December 2020].

- Choudhury, A., 2020. *Part 2: BERT Fine-Tuning Tutorial With Pytorch For Text Classification On The Corpus Of Linguistic…*. [online] Medium. Available at: <https://medium.com/@aniruddha.choudhury94/part-2-bert-fine-tuning-tutorial-with-pytorch-for-text-classification-on-the-corpus-of-linguistic-18057ce330e1> [Accessed 13 December 2020].

- Codecademy. 2020. *What Is Sqlite? | Codecademy*. Available at: <https://www.codecademy.com/articles/what-is-sqlite> [Accessed 20 December 2020].

- Sqlite.org. n.d. *About Sqlite*. Available at: <https://www.sqlite.org/about.html> [Accessed 20 December 2020].

- www.javatpoint.com. 2021. *Sqlite Advantages And Disadvantages - Javatpoint*. Available at: <https://www.javatpoint.com/sqlite-advantages-and-disadvantages> [Accessed 20 December 2020].

- Wolf, T. and Debut, L., 2020. *Transformers: State-Of-The-Art Natural Language Processing*. Aclweb.org. Available at: <https://www.aclweb.org/anthology/2020.emnlp-demos.6.pdf> [Accessed 9 December 2020].

- jackson, m., 2020. *Learn Web Application*. Reddit.com. Available at: <https://www.reddit.com/r/LearnWebApplication/comments/k030ti/what_is_a_web_app/> [Accessed 13 November 2020].

- Deryugina, O., 2010. Chatterbots. *Scientific and Technical Information Processing*, 37(2), pp.143-147.

- De Gasperis, G. (2010). Building an AIML Chatter Bot Knowledge-Base Starting from a FAQ and a Glossary. *Journal of e-Learning and Knowledge Society, 6*(2), 75-83. Italian e-Learning Association. Retrieved January 8, 2021 from https://www.learntechlib.org/p/43452/.

- Huggingface.co. 2020. *BERT — Transformers 4.1.1 Documentation*. [online] Available at: <https://huggingface.co/transformers/model_doc/bert.html#bertforpretraining> [Accessed 8 October 2020].

- Anadea, 2018. *What Is A Chatbot And How To Use It For Your Business*. Medium Available at: <https://medium.com/swlh/what-is-a-chatbot-and-how-to-use-it-for-your-business-976ec2e0a99f> [Accessed 4 November 2020].

- Gutool.com. n.d. *Http Status Codes And Header Checker*. [online] Available at: <https://gutool.com/http-status-codes/> [Accessed 14 November 2020].

- Mohammed, S., 2017. *Shared Task On Emotion Intensity*. [online] Saifmohammad.com. Available at: <http://saifmohammad.com/WebPages/EmotionIntensity-

SharedTask.html> [Accessed 17 December 2020].