Contextual Chatbots with Pytorch

Arpit Patrange: 19th Nov 2021

INTRODUCTION

At the most basic level, a chatbot is a computer program that simulates and processes human conversation either written or spoken, allowing humans to interact with digital devices as if they were communicating with a real person. Chatbots can be as simple as rudimentary programs that answer a simple query with a single-line response, or as sophisticated as digital assistants that learn and evolve to deliver increasing levels of personalization as they gather and process information.

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

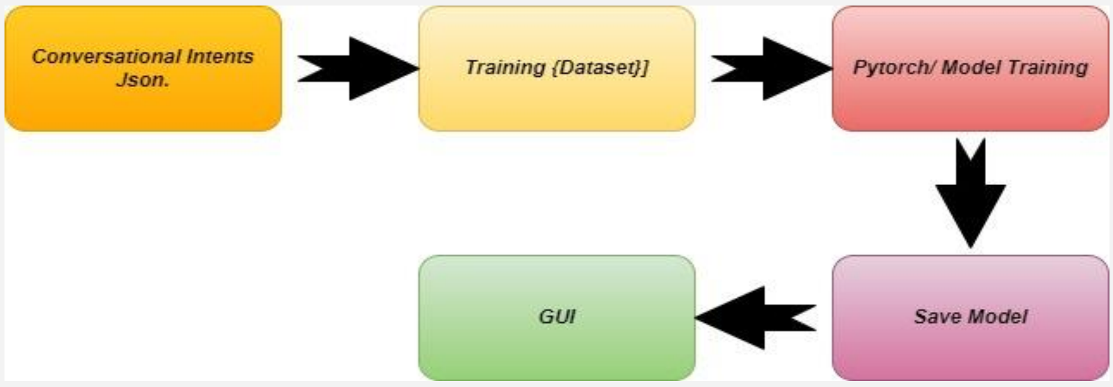
Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services, such as digital assistants, voice-enabled TV remotes, and credit card fraud detection as well as emerging technologies such as self-driving cars.

WORK FLOW

We're going to create a chatbot framework and build a conversational model for a Supermarket. The chatbot for this business needs to handle simple questions about hours of operation, reservation options and so on. We also want it to handle contextual responses such as inquiries about same-day rentals.

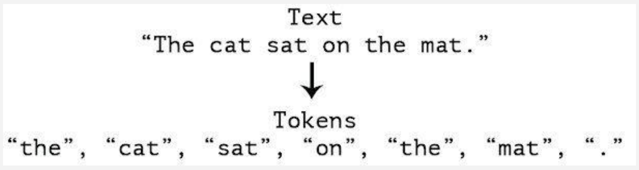
We will working with 4 steps

- We'll transform conversational intent definitions to a statistical model.
- We'll build a chatbot framework to process responses.
- We'll show how basic context can be incorporated into our response processor.
- Building user-friendly GUI.



Tokenization

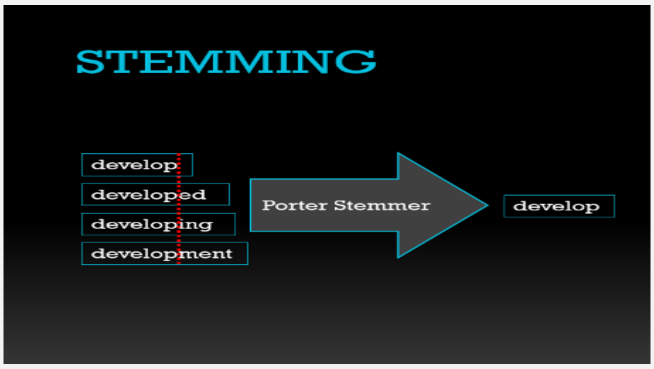
Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.


Text
"The cat sat on the mat."
Tokens
"the", "cat", "sat", "on", "the", "mat", "."

Before processing a natural language, we need to identify the words that constitute a string of characters. That's why tokenization is the most basic step to proceed with NLP (text data). This is important because the meaning of the text could easily be interpreted by analyzing the words present in the text.

Stemming

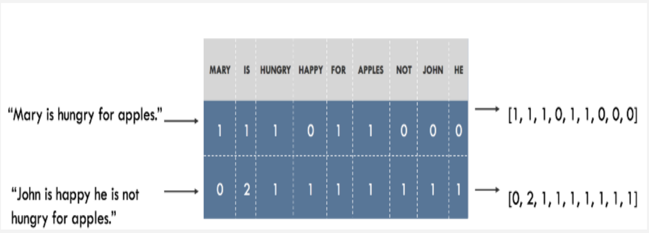
Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolaty", "Choco" to the root word, "chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve". Stemming is an important part of the pipelining process in Natural language processing. The input to the stemmer is tokenized words.



Bag-Of-Words

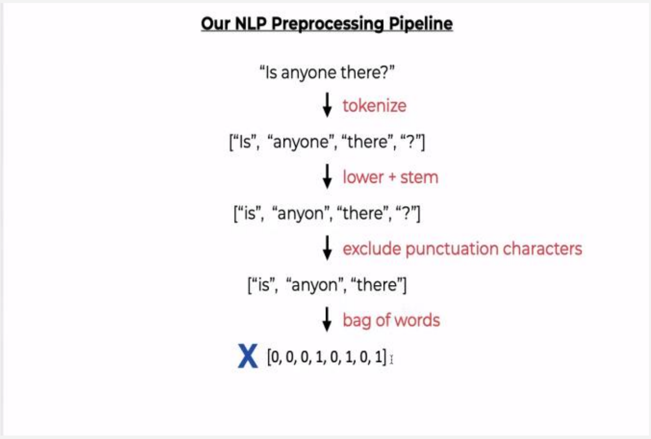
Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a "bag" of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.



	MARY	IS	HUNGRY	HAPPY	FOR	APPLES	NOT	JOHN	HE	
"Mary is hungry for apples."	1	1	1	0	1	1	0	0	0	[1, 1, 1, 0, 1, 1, 0, 0, 0]
"John is happy he is not hungry for apples."	0	2	1	1	1	1	1	1	1	[0, 2, 1, 1, 1, 1, 1, 1, 1]

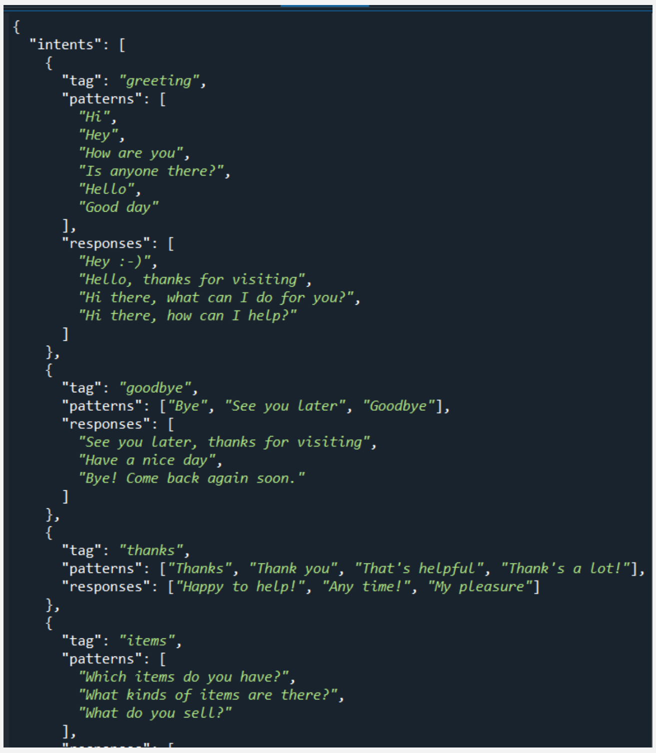
NLP Preprocessing Pipeline


Our NLP Preprocessing Pipeline
"Is anyone there?"
tokenize
["Is", "anyone", "there", "?"]
lower + stem
["is", "anyon", "there", "?"]
exclude punctuation characters
["is", "anyon", "there"]
bag of words
X [0, 0, 0, 1, 0, 1, 0, 1]

ARCHITECTURE

Transform conversational intent definitions to a statistical model

A chatbot framework needs a structure in which conversational intents are defined. One clean way to do this is with a JSON file,

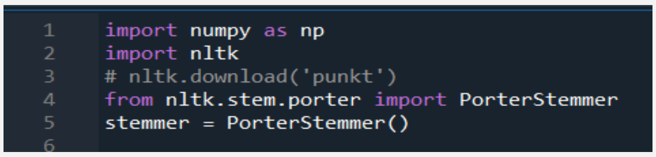


```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": [
        "Hi",
        "Hey",
        "How are you",
        "Is anyone there?",
        "Hello",
        "Good day"
      ],
      "responses": [
        "Hey :-)",
        "Hello, thanks for visiting",
        "Hi there, what can I do for you?",
        "Hi there, how can I help?"
      ]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye"],
      "responses": [
        "See you later, thanks for visiting",
        "Have a nice day",
        "Bye! Come back again soon."
      ]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Thank's a lot!"],
      "responses": ["Happy to help!", "Any time!", "My pleasure"]
    },
    {
      "tag": "items",
      "patterns": [
        "Which items do you have?",
        "What kinds of items are there?",
        "What do you sell?"
      ],
      "responses": [
```

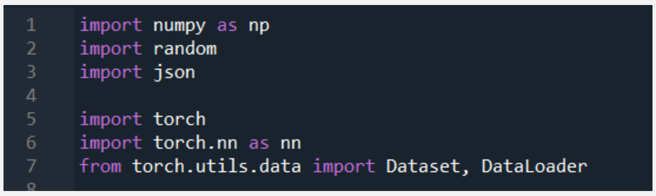
Each conversational intent contains:

- a tag (a unique name)
- patterns (sentence patterns for our neural network text classifier)
- responses (one will be used as a response)

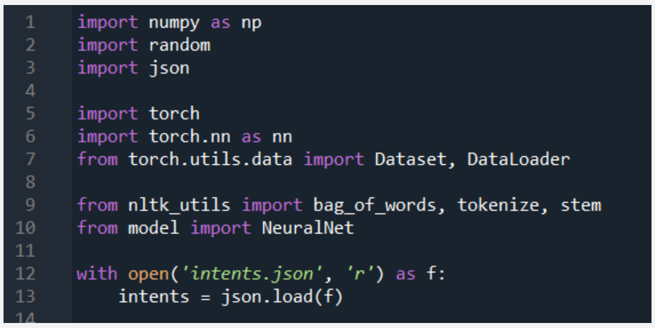
And later on we'll add some basic contextual elements.



```
import numpy as np
import nltk
# nltk.download('punkt')
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

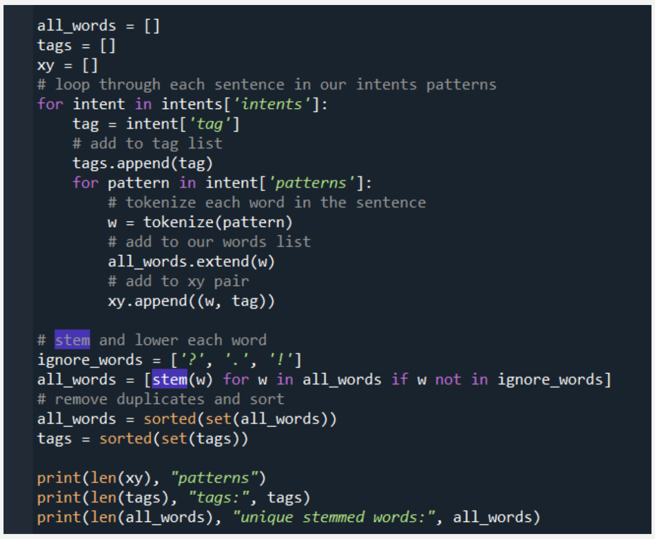


```
import numpy as np
import random
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```



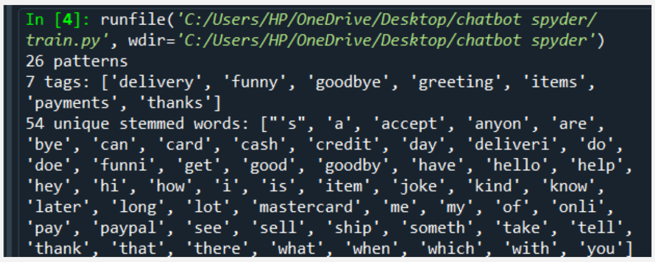
```
import numpy as np
import random
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet

with open('intents.json', 'r') as f:
    intents = json.load(f)
```

With our intents JSON file loaded, we can now begin to organize our documents, words and classification classes.



```
all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))

# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))

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

We create a list of documents (sentences), each sentence is a list of stemmed words and each document is associated with an intent (a class).

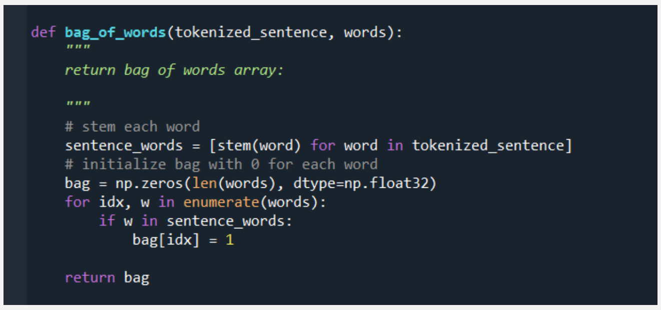


```
In [4]: runfile('C:/Users/HP/OneDrive/Desktop/chatbot spyder/
train.py', wdir='C:/Users/HP/OneDrive/Desktop/chatbot spyder')
26 patterns
7 tags: ['delivery', 'funny', 'goodbye', 'greeting', 'items',
'payments', 'thanks']
54 unique stemmed words: ["'s", 'a', 'accept', 'anyon', 'are',
'bye', 'can', 'card', 'cash', 'credit', 'day', 'deliveri', 'do',
'doe', 'funni', 'get', 'good', 'goodby', 'have', 'hello', 'help',
'hey', 'hi', 'how', 'i', 'is', 'item', 'joke', 'kind', 'know',
'later', 'long', 'lot', 'mastercard', 'me', 'my', 'of', 'onli',
'pay', 'paypal', 'see', 'sell', 'ship', 'someth', 'take', 'tell',
'thank', 'that', 'there', 'what', 'when', 'which', 'with', 'you']
```

The stem 'talk' will match 'take', 'taking', 'takers', etc. We could clean the words list and remove useless entries but this will suffice for now.

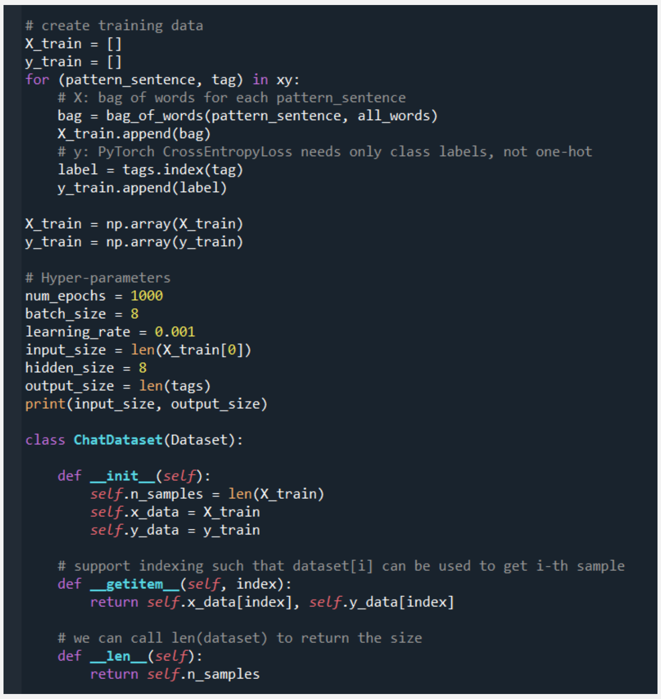
Then the next step is to get that stemmed data and move towards the numerical representation of a data by designing the dataset using the most popular algorithm Bag-Of-Words

Before we can begin processing intents, we need a way to produce a bag-of-words from user input

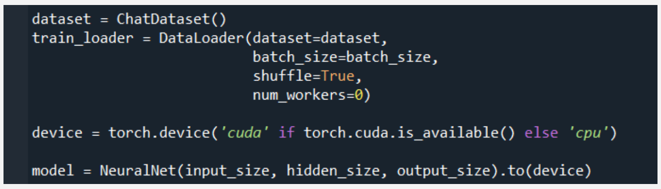


```
def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:

    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag
```

CREATING TRAINING DATASET



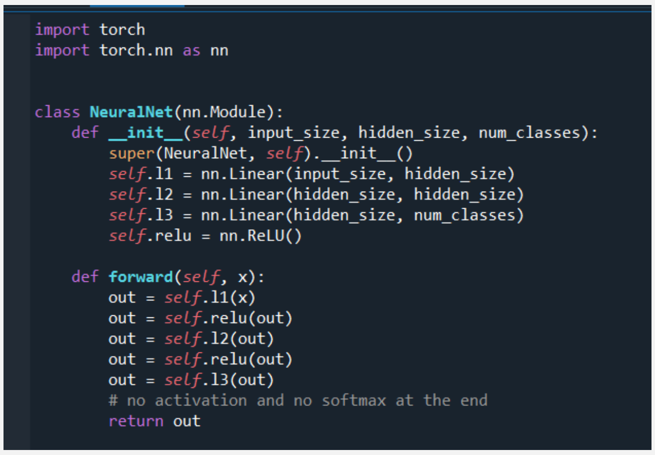
```
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)

# Hyper-parameters
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)

class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples
```



```
dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

Build a chatbot framework to process responses

We are now ready to build our response processor.



```
import torch
import torch.nn as nn


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

Each sentence passed to response is classified. Our classifier uses and is lighting fast. The probabilities returned by the model are lined-up with our intents definitions to produce a list of potential responses.

If one or more classifications are above a threshold, we see if a tag matches an intent and then process that. We'll treat our classification list as a stack and pop off the stack looking for a suitable match until we find one, or it's empty.

In this way the model building would get performed.

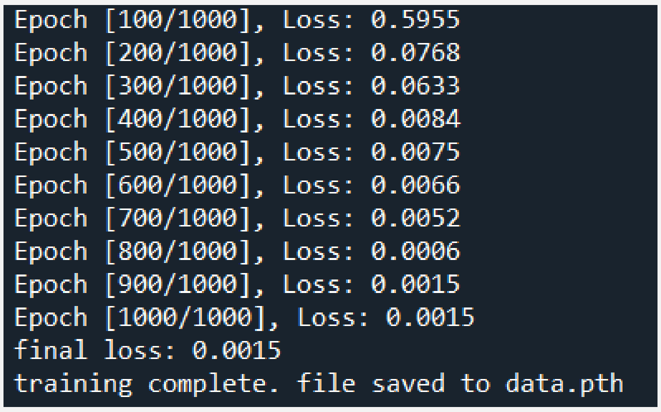
Analysis and Results

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

The Loss is used to calculate the gradients. And gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained.

Keras and Tensorflow have various inbuilt loss functions for different objectives. In this guide, I will be covering the following essential loss functions, which could be used for most of the objectives.

As we train the model of 2 layers neural network as the number of epoch increases simultaneously the accuracy of a model is also increasing.



```
Epoch [100/1000], Loss: 0.5955
Epoch [200/1000], Loss: 0.0768
Epoch [300/1000], Loss: 0.0633
Epoch [400/1000], Loss: 0.0084
Epoch [500/1000], Loss: 0.0075
Epoch [600/1000], Loss: 0.0066
Epoch [700/1000], Loss: 0.0052
Epoch [800/1000], Loss: 0.0006
Epoch [900/1000], Loss: 0.0015
Epoch [1000/1000], Loss: 0.0015
final loss: 0.0015
training complete. file saved to data.pth
```

Building a userfriendly GUI

Graphical user interfaces would become the standard of user-centered design in software application programming, providing users the capability to intuitively operate computers and other electronic devices through the direct manipulation of graphical icons such as buttons, scroll bars, windows, tabs, menus, cursors, and the mouse pointing device. Many modern graphical user interfaces feature touchscreen and voice-command interaction capabilities.

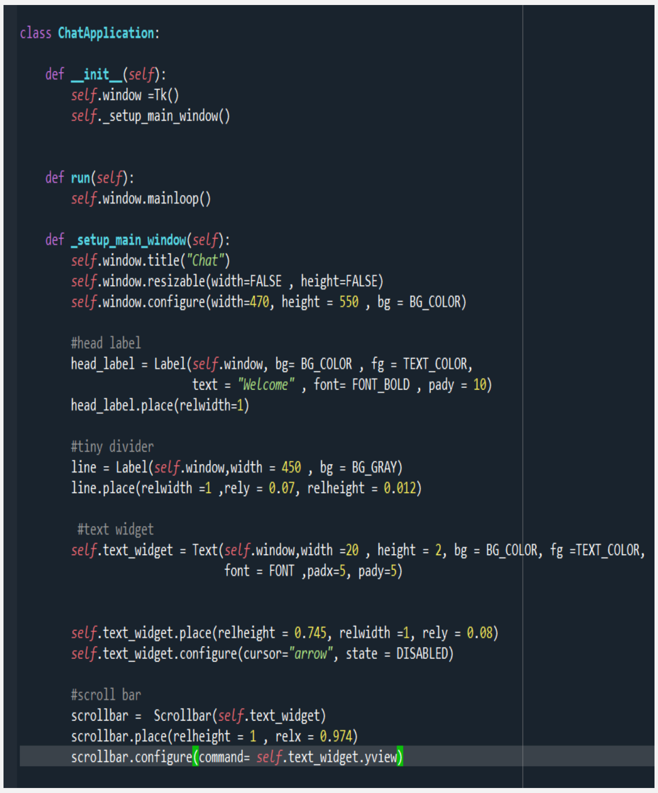
Graphical user interface design principles conform to the model–view–controller software pattern, which separates internal representations of information from the manner in which information is presented to the user, resulting in a platform where users are shown which functions are possible rather than requiring the input of command codes. Users interact with information by manipulating visual widgets, which are designed to respond in accordance with the type of data they hold and support the actions necessary to complete the user's task.

The appearance, or "skin," of an operating system or application software may be redesigned at will due to the nature of graphical user interfaces being independent from application functions.

Applications typically implement their own unique graphical user interface display elements in addition to graphical user interface elements already present on the existing operating system. A typical graphical user interface also includes standard formats for representing graphics and text, making it possible to share data between applications running under common graphical user interface design software.

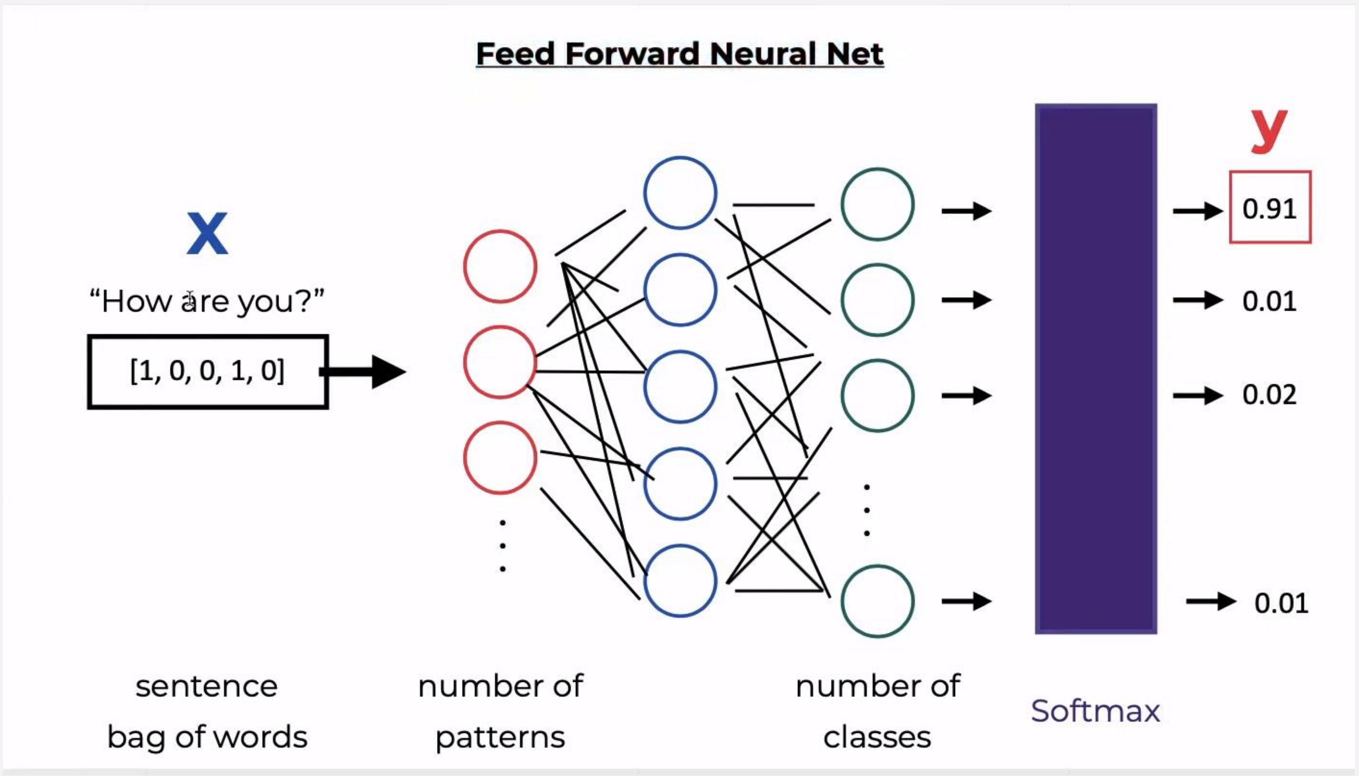
Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

We created the easy interactive GUI using the Tkinter

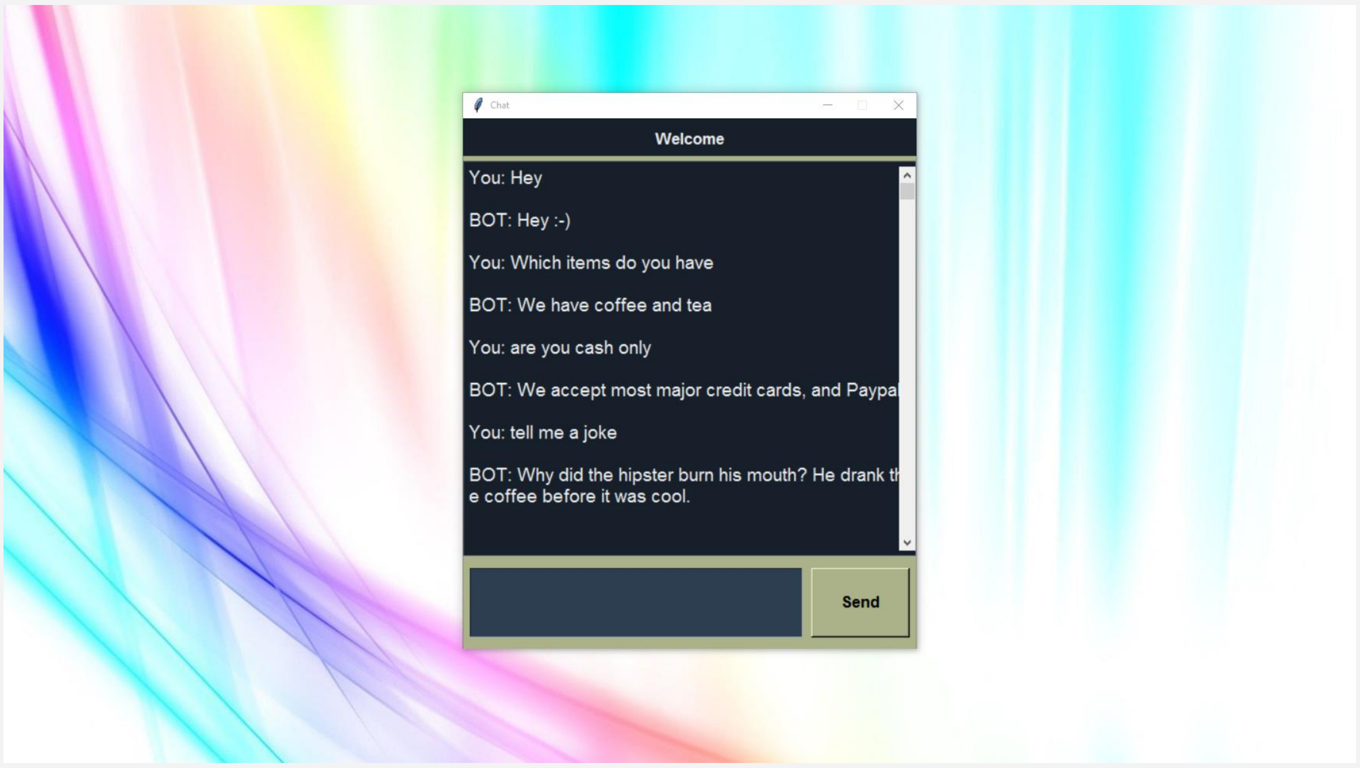


```python
class ChatApplication:

    def _init_(self):
        self.window =Tk()
        self._setup_main_window()


    def run(self):
        self.window.mainloop()

    def _setup_main_window(self):
        self.window.title("Chat")
        self.window.resizable(width=FALSE , height=FALSE)
        self.window.configure(width=470, height = 550 , bg = BG_COLOR)

        #head label
        head_label = Label(self.window, bg= BG_COLOR , fg = TEXT_COLOR,
                           text = "Welcome" , font= FONT_BOLD , pady = 10)
        head_label.place(relwidth=1)

        #tiny divider
        line = Label(self.window,width = 450 , bg = BG_GRAY)
        line.place(relwidth =1 ,rely = 0.07, relheight = 0.012)

         #text widget
        self.text_widget = Text(self.window,width =20 , height = 2, bg = BG_COLOR, fg =TEXT_COLOR,
                                font = FONT ,padx=5, pady=5)


        self.text_widget.place(relheight = 0.745, relwidth =1, rely = 0.08)
        self.text_widget.configure(cursor="arrow", state = DISABLED)

        #scroll bar
        scrollbar =  Scrollbar(self.text_widget)
        scrollbar.place(relheight = 1 , relx = 0.974)
        scrollbar.configure(command= self.text_widget.yview)
```

Short visualization of Feed Forward neural network used in constructing the model
To get more accurate results



APPLICATION



You: Hey

BOT: Hey :-)

You: Which items do you have

BOT: We have coffee and tea

You: are you cash only

BOT: We accept most major credit cards, and Paypal

You: tell me a joke

BOT: Why did the hipster burn his mouth? He drank th
e coffee before it was cool.

Conclusion and Future Scope of ChatBot

The Golden age of conversational artificial intelligence (AI) is here. Conversational bots or simply put chatbots have applications that range from customer-facing AI assistants, support chatbots, skill chatbots, assistant bots, and transactional bots. The interest of business in this segment is rampant with the huge investments in this emerging technology by governments, healthcare institutions, manufacturing enterprises and so on.

Conversational Chatbots are leveraging the power of conversational AI to improve their customer experience, and thereby increase the shareholder returns.

Technology grows with time, and it just gets better and better day by day changing on an almost daily basis and often it becomes difficult for enterprises to keep up with this pace of change!

Integrated with Artificial Intelligence and Conversational Intelligence the modern chatbots are classified into

• Voice conversational interface.

• Text conversational interface.

Customer Interaction Powered by Conversational AI:

The role of conversational AI is changing and so are the bots that power it. From answering simple queries to comprehending complex requests expect chatbots to be seen in every domain of modern enterprises. For instance, Generali, the third-largest insurance company in the world, saved US$1 million in its first year of deployment of a customer-facing cognitive assistant. This chatbot leverages NLP (natural language processing) to convert customer voice queries to text queries, handling and answering the initial queries about home and auto insurance policies and claims.

Agent-supporting AI assistants:

In the coming times, AI assistants backed by Conversational AI platforms will help human customer service agents with internal support while they interact with customers. A good example would be seen in the BFSI industry where chatbots would help bank professionals to open a customer's account by asking the agents to manually fulfil all the AOP requirements like KYC checks, and collecting proof of address and income details.

Conversational Bot for Critical Functions:

In the future, expect AI technologies to be deployed in critical lifesaving functions that include elderly care, assisting in life-saving operations and helping with disaster management. Expect a long-term friendship between humans and Conversational AI Platforms to work together for the betterment of business and mankind.

PROJECT LINK *:* ***https://github.com/Arpit-Patrange/CHATBOT---KM073A15.git***

References

1. *2016. jabong. (2016). Retrieved Dec 1, 2017 from https://www.jabong.com/*

2. *2017. Chatbots. (2017). Retrieved January 11, 2017 from https://chatbottle.co/*

3. *2017. IBM Watson Conversation. (2017). Retrieved March 1, 2017 from https://www.ibm.com/watson/services/conversation/*

4. *Kathleen Chaykowski. 2016. More Than 11,000 Bots Are Now On Facebook Messenger. (2016). Retrieved December 28, 2016 from http: //www.forbes.com/sites/kathleenchaykowski/2016/07/01/ more-than-11000-bots-are-now-on-facebook-messenger/*

5. *O' Brien Chris. 2016. Facebook Messenger chief says platform's 34,000 chatbots are finally improving user experience. (2016). Retrieved February 7, 2017 from http://venturebeat.com/2016/11/11/ facebook-messenger-chief-says-platforms-34000/ -chatbots-are-finally-improving-user-experience/*

6. *Alan Dix, Tiziana Catarci, Benjamin Habegger, Yannis loannidis, Azrina Kamaruddin, Akrivi Katifori, Giorgos Lepouras, Antonella Poggi, and Devina Ramduny-Ellis. 2006. Intelligent Context-sensitive Interactions on Desktop and the Web. In Proceedings of the International Workshop in Conjunction with AVI 2006*

7. *Craig Elimeliah. 2016. Why chatbots are replacing apps. (2016). Retrieved January 20, 2017 from*

8. *Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. Advances in psychology 52 (1988), 139–183.*

9. *Mariam Hassib, Daniel Buschek, Pawel W. Wozniak, and Florian Alt. 2017. HeartChat: Heart Rate Augmented Mobile Chat to Support Empathy and Awareness. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). 2239–2251.*

10. *BI Intelligence. 2016. Messaging apps are now bigger than social networks. (2016). Retrieved February 7, 2017*

11. *Joonhwan Lee, Soojin Jun, Jodi Forlizzi, and Scott E. Hudson. 2006. Using Kinetic Typography to Convey Emotion in Text-based Interpersonal Communication. In Proceedings of the 6th Conference on Designing Interactive Systems (DIS '06). 41–49.*

12. *Vera Q. Liao, Matthew Davis, Werner Geyer, Michael Muller, and N. Sadat Shami. 2016. What Can You Do?: Studying Social-Agent Orientation and Agent Proactive Interactions with an Agent for Employees. In Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS '16). 264–275.*

13. *Susan Robinson, David Traum, Midhun Ittycheriah, and Joe Henderer. 2008. What would you ask a conversational agent? Observations of Human-Agent dialogues in a museum setting. In Language Resources and Evaluation Conference (LREC). Marrakech (Morocco).*