## horizontal line



Esoft Chat Bot

12.05.2022

**─**

# 

# Group Members

M.N.A. Ashfaq

London Met ID : MNX0003

Kisara Wenuk

London Met ID :

# About Esoft ChatBot

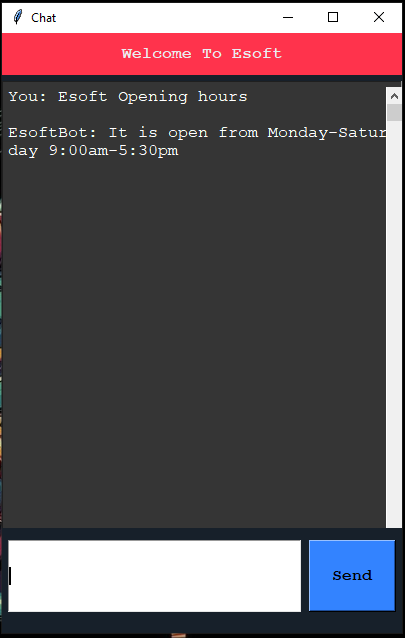
This is a simple chat bot built with python, pytorch, tkinter in order to assist students and staff in esoft metro campus with simple queries. Using this chatbot anyone can ask their basic questions like Esoft opening hours, password reset etc… using this chatbot.

We have used intents priorly and with those interns we train and store the conversation replies inorder to reply for any conversations.

The below image is an example of how the opening hour is shown using Esof Bot.

**Project Link**

https://drive.google.com/drive/folders/10jq0anOEKbYTNSU3PXOEwD0bUU6FEy44?usp=sharing

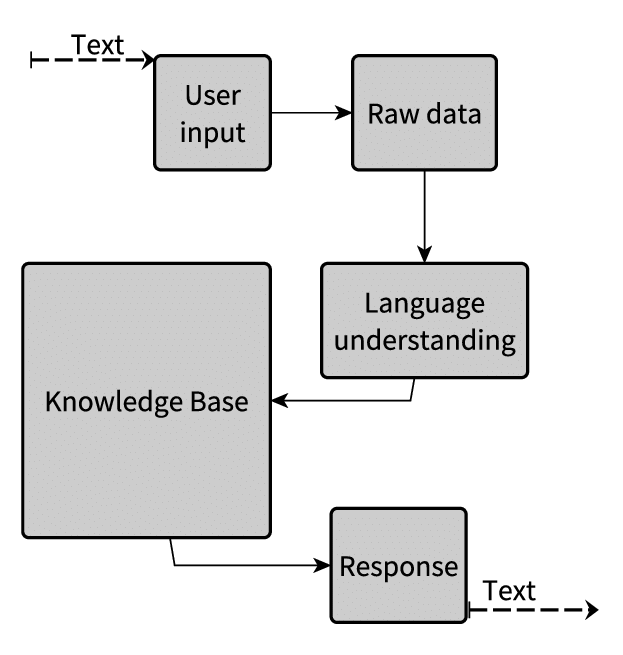


## 

## 

## State transition diagram

Since our bot is a simple textual replier, this is how the conversation flows



# P.E.A.S

## Performance

Since this is a windows application built with python, the replies are very fast and provides replies accurately

## Environment

This chatbot can be used in a campus environment and will provide answers for basic questions.

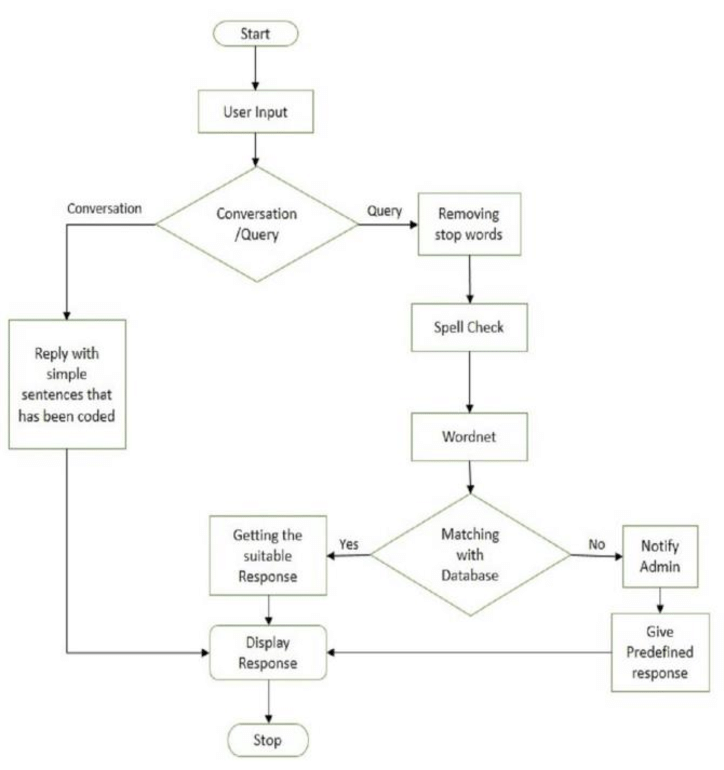
## Actuators

We don't have any actuators since this is an application

## Sensors

We don't have any actuators since this is an application

# Flow Chart



# Source Code

Application Code

from tkinter import \*

from numpy import size

from chat import get\_response, bot\_name

BG\_GRAY = "#ABB2B9"

BG\_BLUE = "#3383FF"

BG\_COLOR = "#17202A"

TEXT\_COLOR = "#EAECEE"

FONT = "Courier 12"

FONT\_BOLD = "Courier 12 bold"

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=True, height=True)

self.window.configure(width=400, height=600, bg=BG\_COLOR)

# head label

head\_label = Label(self.window, bg="#FF334C", fg=TEXT\_COLOR,

text="Welcome To Esoft", font=FONT\_BOLD, pady=10)

head\_label.place(relwidth=1)

# text widget

self.text\_widget = Text(self.window, width=20, height=2, bg="#353535", fg="#fff",

font=FONT, padx=5, pady=5)

self.text\_widget.place(relheight=1, 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)

# bottom label

bottom\_label = Label(self.window, bg=BG\_COLOR, height=80)

bottom\_label.place(relwidth=1, rely=0.825)

# message entry box

self.msg\_entry = Entry(bottom\_label, bg="#fff", fg="#202020", font=FONT)

self.msg\_entry.place(relwidth=0.74, relheight=0.06, rely=0.008, relx=0.011)

self.msg\_entry.focus()

self.msg\_entry.bind("<Return>", self.\_on\_enter\_pressed)

# send button

send\_button = Button(bottom\_label, text="Send", font=FONT\_BOLD, width=20, bg=BG\_BLUE,

command=lambda: self.\_on\_enter\_pressed(None))

send\_button.place(relx=0.77, rely=0.008, relheight=0.06, relwidth=0.22)

def \_on\_enter\_pressed(self, event):

msg = self.msg\_entry.get()

self.\_insert\_message(msg, "You")

def \_insert\_message(self, msg, sender):

if not msg:

return

self.msg\_entry.delete(0, END)

msg1 = f"{sender}: {msg}\n\n"

self.text\_widget.configure(state=NORMAL)

self.text\_widget.insert(END, msg1)

self.text\_widget.configure(state=DISABLED)

msg2 = f"{bot\_name}: {get\_response(msg)}\n\n"

self.text\_widget.configure(state=NORMAL)

self.text\_widget.insert(END, msg2)

self.text\_widget.configure(state=DISABLED)

self.text\_widget.see(END)

if \_\_name\_\_ == "\_\_main\_\_":

app = ChatApplication()

app.run()

Chat Code

import random

import json

import torch

from model import NeuralNet

from nltk\_utils import bag\_of\_words, tokenize

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

with open('intents.json', 'r') as json\_data:

intents = json.load(json\_data)

FILE = "data.pth"

data = torch.load(FILE)

input\_size = data["input\_size"]

hidden\_size = data["hidden\_size"]

output\_size = data["output\_size"]

all\_words = data['all\_words']

tags = data['tags']

model\_state = data["model\_state"]

model = NeuralNet(input\_size, hidden\_size, output\_size).to(device)

model.load\_state\_dict(model\_state)

model.eval()

bot\_name = "EsoftBot"

def get\_response(message):

sentence = tokenize(message)

X = bag\_of\_words(sentence, all\_words)

X = X.reshape(1, X.shape[0])

X = torch.from\_numpy(X).to(device)

output = model(X)

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

tag = tags[predicted.item()]

probs = torch.softmax(output, dim=1)

prob = probs[0][predicted.item()]

if prob.item() > 0.75:

for intent in intents['intents']:

if tag == intent["tag"]:

return random.choice(intent['responses'])

return "I do not understand..."

Logic Code

import numpy as np

import nltk

#nltk.download('punkt')

from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def tokenize(sentence):

return nltk.word\_tokenize(sentence)

def stem(word):

return stemmer.stem(word.lower())

def bag\_of\_words(tokenized\_sentence, words):

# 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

Training Code

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)

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)

# 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)

# Loss and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

# Train the model

for epoch in range(num\_epochs):

for (words, labels) in train\_loader:

words = words.to(device)

labels = labels.to(dtype=torch.long).to(device)

# Forward pass

outputs = model(words)

# if y would be one-hot, we must apply

# labels = torch.max(labels, 1)[1]

loss = criterion(outputs, labels)

# Backward and optimize

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if (epoch+1) % 100 == 0:

print (f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')

print(f'final loss: {loss.item():.4f}')

data = {

"model\_state": model.state\_dict(),

"input\_size": input\_size,

"hidden\_size": hidden\_size,

"output\_size": output\_size,

"all\_words": all\_words,

"tags": tags

}

FILE = "data.pth"

torch.save(data, FILE)

print(f'training complete. file saved to {FILE}')

Model Code

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)

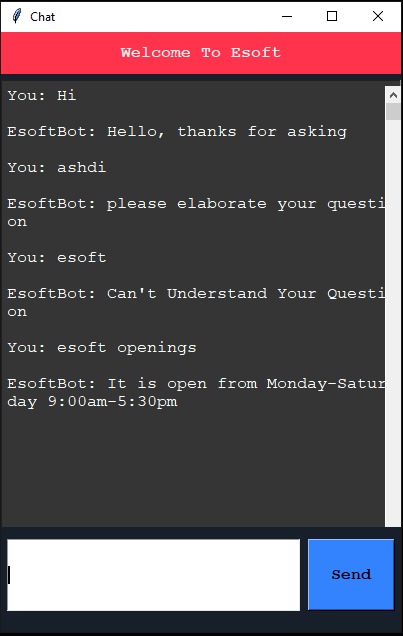
out = self.l2(out)

out = self.relu(out)

out = self.l3(out)

return out

# Tested Data



# Conclusion

We did basic research and completed this chatbot as a windows python application, and we have trained the app so that it can answer questions.

# References

<https://chatbotsmagazine.com/contextual-chat-bots-with-tensorflow-4391749d0077>

# 