-

CREATE A CHATBOT USING PYTHON

PHASE 4

SUBMITTED BY : VIKRAM S (422521104040)

INTRODUCTION

In this phase, we will focus on training the model and integrating it in the web using flask. The chatbot is designed to offer information and assistance related to coffee and tea products. Users can interact with the chatbot via a web interface.

CODE

We use intends.json file which contains tags, patterns and responses in which the chatbot model trains.

intends.json

```
{
  "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": [
              "We sell coffee and tea",
              "We have coffee and tea"
          ]
      },
      {
          "tag": "payments",
          "patterns": [
              "Do you take credit cards?",
              "Do you accept Mastercard?",
              "Can I pay with Paypal?",
              "Are you cash only?"
          ],
          "responses": [
              "We accept VISA, Mastercard, and Paypal",
              "We accept most major credit cards and Paypal"
          ]
      },
      {
          "tag": "delivery",
          "patterns": [
              "How long does delivery take?",
              "How long does shipping take?",
              "When do I get my delivery?"
          ],
          "responses": [
              "Delivery takes 2-4 days",
              "Shipping takes 2-4 days"
          ]
      },
      {
          "tag": "product_info",
          "patterns": [
              "Tell me about your tea products",
              "What types of tea do you offer?",
              "Give me details about your coffee offerings"
          ],
          "responses": [
              "We offer a wide range of tea products, including
black, green, herbal, and specialty teas. Each tea is carefully
selected for its unique flavor and aroma.",
              "Our coffee selection includes various blends,
from medium roast to dark roast, as well as single-origin
options. We have something for every coffee enthusiast."
          ]
      },
      {
          "tag": "best_sellers",
          "patterns": [
              "What are your best-selling teas?",
              "Tell me your popular coffee choices"
          ],
          "responses": [
              "Our best-selling teas include our classic black
tea and soothing chamomile herbal tea. Customers also love our
signature coffee blend for its rich taste.",
              "Customers often choose our house blend coffee
and the aromatic French roast. They're our top picks!"
          ]
      },
      {
          "tag": "discounts",
          "patterns": [
              "Do you offer any discounts?",
              "Are there any ongoing promotions?"
          ],
          "responses": [
              "Yes, we have special discounts on selected tea
and coffee products. You can check our website for the latest
promotions and offers.",
              "Absolutely! We frequently run promotions on our
premium blends. Don't miss out on our current discounts."
          ]
      },
```

-

```
        {
            "tag": "customer_support",
            "patterns": [
                "How can I contact customer support?",
                "I need assistance with my order"
            ],
            "responses": [
                "Our customer support team is here to assist you. You can reach out to us through our website's live chat, email, or phone. We're ready to help!",
                "If you have any questions or need assistance with your order, please don't hesitate to contact our dedicated customer support team."
            ]
        },
        {
            "tag": "reviews",
            "patterns": [
                "Can you share customer reviews?",
                "What do customers say about your products?"
            ],
            "responses": [
                "Our customers have shared fantastic reviews about our tea and coffee. You can find authentic reviews on our website to learn about their experiences.",
                "We're proud to have received positive feedback from our satisfied customers. Check out the reviews section on our website for firsthand testimonials."
            ]
        },
        {
            "tag": "recommendation",
            "patterns": [
                "What tea would you recommend?",
                "Suggest a coffee blend for me"
            ],
            "responses": [
                "For a soothing experience, we recommend trying
  our Chamomile Herbal Tea. If you prefer coffee, our House Blend
  is a great choice for its balanced flavor.",
                "If you're new to coffee, you might enjoy our
  Medium Roast Blend. For a bolder taste, go for our French
  Roast. Enjoy your cup of goodness!"
            ]
        },
        {
            "tag": "brewing_tips",
            "patterns": [
                "How should I brew the perfect cup of tea?",
                "Give me some coffee brewing tips"
            ],
            "responses": [
                "To brew the perfect cup of tea, use freshly
  boiled water and steep the tea bag or leaves for the
  recommended time. Follow our brewing guide on the product label
  for best results.",
                "For a delightful cup of coffee, use freshly
  ground beans, maintain the right water temperature, and
  consider your preferred brewing method, such as French press or
  drip. Check our website for detailed brewing tips."
            ]
        },
        {
            "tag": "organic_products",
            "patterns": [
                "Do you offer organic tea or coffee?",
                "Tell me about your organic options"
            ],
            "responses": [
                "Yes, we have a selection of organic teas and
  coffees. Our organic products are sourced sustainably and meet
  the highest quality standards.",
```

-

```
              "Explore our range of organic options, including organic green tea and single-origin organic coffee beans. We're committed to offering you the finest organic choices."
          ]
      }
  ]
}
```

The nltk_utils.py file has functions for tokenization, stemming, and creating a “bag of words” representation of a sentence. These functions can be helpful when working with chatbot development.

nltk_utils.py

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

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


def stem(word):
    """
    stemming = find the root form of the word
    examples:
```

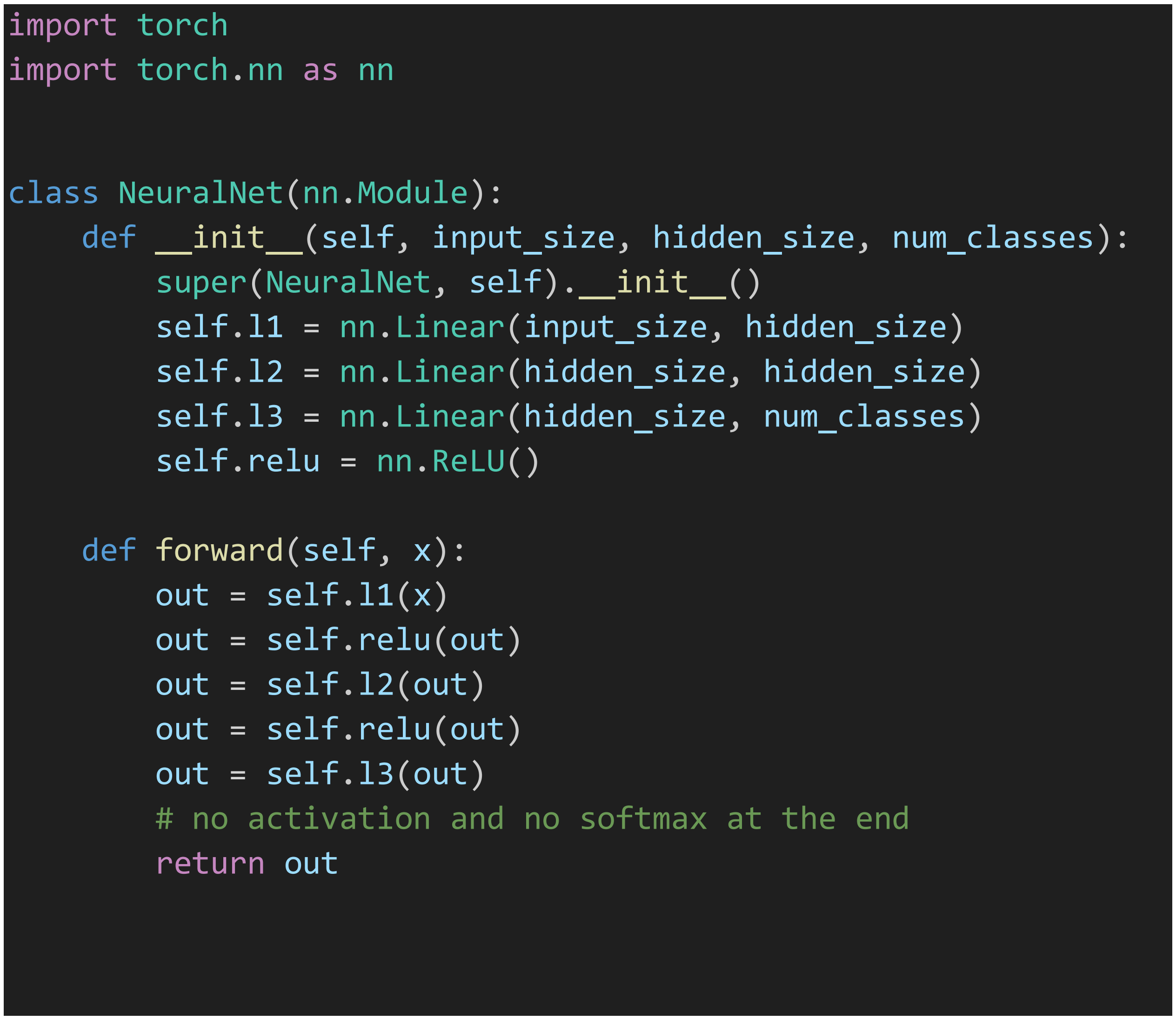
```
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())


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

    return bag
```

The model.py file defines a neural network using PyTorch. This neural network consists of three linear layers with ReLU activation function.

model.py



```python
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
```

The train.py file contains code that is responsible for training chatbot model based on a dataset of intents and associated patterns. The model is implemented using PyTorch. Here's a summary of the code:

1. Data Preparation: The code reads intent patterns and associated tags from a JSON file (intents.json). It

-

tokenizes the patterns, stems words, and organizes the data for training. It also creates a bag of words representation for the sentences and prepares the training dataset.

2. Model Architecture: It defines the architecture of the neural network model. The model consists of an input layer, a hidden layer, and an output layer. The input size, hidden size, and output size are determined based on the data and tags.
3. Training Data Setup: The code creates a custom dataset class ('ChatDataset') using PyTorch's 'Dataset' and sets up a data loader for training. It specifies hyperparameters such as the number of epochs, batch size, and learning rate.
4. Training Loop: The model is trained using a loop that iterates over the training data. It calculates the loss using cross-entropy loss, performs backpropagation, and updates the model's weights using the Adam optimizer. The training loop also prints the loss for every 100 epochs.

train.py

```
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
```

```python
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 = 2000
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
}

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

OUTPUT

```
40 patterns
14 tags: ['best_sellers', 'brewing_tips', 'customer_support', 'delivery',
'discounts', 'goodbye', 'greeting', 'items', 'organic_products', 'payments
', 'product_info', 'recommendation', 'reviews', 'thanks']
90 unique stemmed words: ["'s", 'a', 'about', 'accept', 'ani', 'anyon', 'a
re', 'assist', 'best-sel', 'blend', 'brew', 'bye', 'can', 'card', 'cash',
'choic', 'coffe', 'contact', 'credit', 'cup', 'custom', 'day', 'deliveri',
'detail', 'discount', 'do', 'doe', 'for', 'get', 'give', 'good', 'goodby',
'have', 'hello', 'help', 'hey', 'hi', 'how', 'i', 'is', 'item', 'kind', 'l
ater', 'long', 'lot', 'mastercard', 'me', 'my', 'need', 'of', 'offer', 'on
go', 'onli', 'option', 'or', 'order', 'organ', 'pay', 'paypal', 'perfect',
'popular', 'product', 'promot', 'recommend', 'review', 'say', 'see', 'sell
', 'share', 'ship', 'should', 'some', 'suggest', 'support', 'take', 'tea',
'tell', 'thank', 'that', 'the', 'there', 'tip', 'type', 'what', 'when', 'w
hich', 'with', 'would', 'you', 'your']
90 14
Epoch [100/2000], Loss: 0.9530
Epoch [200/2000], Loss: 0.1069
Epoch [300/2000], Loss: 0.0340
Epoch [400/2000], Loss: 0.0092
Epoch [500/2000], Loss: 0.0036
Epoch [600/2000], Loss: 0.0031
Epoch [700/2000], Loss: 0.0021
```

-

```
Epoch [800/2000], Loss: 0.0015
Epoch [900/2000], Loss: 0.0004
Epoch [1000/2000], Loss: 0.0007
Epoch [1100/2000], Loss: 0.0004
Epoch [1200/2000], Loss: 0.0002
Epoch [1300/2000], Loss: 0.0002
Epoch [1400/2000], Loss: 0.0002
Epoch [1500/2000], Loss: 0.0001
Epoch [1600/2000], Loss: 0.0001
Epoch [1700/2000], Loss: 0.0000
Epoch [1800/2000], Loss: 0.0000
Epoch [1900/2000], Loss: 0.0000
Epoch [2000/2000], Loss: 0.0000
final loss: 0.0000
training complete. file saved to data.pth
```

The chat.py file contains code which uses trained chatbot model to interact with users

1. It imports the necessary libraries and modules, including random, json, torch, and custom modules like model and nltk_utils.
2. The script loads the trained chatbot model and its associated data from a file named "data.pth." This data includes the model's state, input size, hidden size, output size, a list of all unique words, and tags.
3. The get_response function takes user input in the form of a message (msg) and processes it to generate a response from the chatbot.

   - It tokenizes the input message into words.
   - Converts the words into a bag-of-words representation using the same preprocessing functions used during training (from nltk_utils).
   - Feeds the bag-of-words representation into the trained model to get a prediction.
   - Determines the chatbot's response based on the predicted tag and a predefined confidence threshold

-

(0.75). If the model's confidence in its prediction is above this threshold, it randomly selects a response from the intents defined in the "intents.json" file. Otherwise, it responds with "I do not understand..."

4. The script enters a loop that continuously takes user input, processes it using the get_response function, and prints the chatbot's response. The loop continues until the user enters "quit."

chat.py

```
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 = "Sam"

def get_response(msg):
    sentence = tokenize(msg)
    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..."


if __name__ == "__main__":
    print("Let's chat! (type 'quit' to exit)")
    while True:
        # sentence = "do you use credit cards?"
        sentence = input("You: ")
        if sentence == "quit":
```

-

```
            break

        resp = get_response(sentence)
        print(resp)
```

OUTPUT

```
Let's chat! (type 'quit' to exit)
You: Hi
Hi there, how can I help?
You: What do you sell
We have coffee and tea
You: Tell me about your tea products
Our coffee selection includes various blends, from medium roast to dark ro
ast, as well as single-origin options. We have something for every coffee
enthusiast.
You: Do you take credit cards
We accept most major credit cards and Paypal
You: When do I get my delivery
Delivery takes 2-4 days
You: Thanks
Any time!
You: Bye
Have a nice day
You: quit
```

The index.html and app.py file contains basic html structure and flask code which enable us to interact with the chatbot model.

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Chatbot</title>
</head>
```

-

```
<body>
    <div id="chat-container">
        <div id="chat-box">
            <div id="chat-output"></div>
            <div id="chat-input">
                <input type="text" id="user-input" 
placeholder="Type a message...">
                <button id="send-button">Send</button>
            </div>
        </div>
    </div>
    <script src="https://code.jquery.com/jquery-
3.6.0.min.js"></script>
    <script>
        $(document).ready(function(){
            $("#send-button").on('click', function(){
                var user_msg = $("#user-input").val();
                $("#chat-output").append("<p>You: " + user_msg 
+ "</p>");
                $("#user-input").val('');

                $.ajax({
                    url: "/get_response",
                    type: "POST",
                    data: {user_msg: user_msg},
                    success: function(response){
                        var bot_response = response.response;
                        $("#chat-output").append("<p>{{ 
bot_name }}: " + bot_response + "</p>");
                    }
                });
            });
        });
    </script>
</body>
</html>
```

-

app.py

```
from flask import Flask, render_template, request, jsonify
import random
import json
import torch
from model import NeuralNet
from nltk_utils import bag_of_words, tokenize

app = Flask(__name__)

# Load intents and the chatbot model
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)
model.load_state_dict(model_state)
model.eval()

# Define the chatbot's name
bot_name = "Sam"

# Function to get chatbot responses
def get_response(msg):
    sentence = tokenize(msg)
```

```python
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X)

    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..."

# Define the home page route
@app.route('/')
def home():
    return render_template('index.html')

# Define a route for handling chat interactions
@app.route('/get_response', methods=['POST'])
def chat():
    user_msg = request.form['user_msg']
    bot_response = get_response(user_msg)
    return jsonify({'response': bot_response})

if __name__ == "__main__":
    app.run(debug=True)
```

CONCLUSION

In summary, in this project phase we created a custom dataset and trained the chatbot model with it and integrated it in web using flask. We would further improve the responses and interface in the following phase.