AI ASSISTANT

Project submitted in the

fulfilment

Of

Elective IV - Artificial Intelligence

By

Jay Menon H041
Aditi Nair H044

Under the supervision of

Saurav Verma

(Assistant Professor, MPSTME)

SVKM's NMIMS University

(Deemed-to-be University)

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT & ENGINEERING
Vile Parle (W), Mumbai- 56

2020-21

LAB 10

Aim: To study and implement AI based Project

Theory:

a. **Introduction of Project Topic**

As we all know that nowadays for businesses, it has become necessary to solve the queries and problems of the customers to ensure consumer loyalty along with the brand establishment. And just like the earlier times, man has looked to take help of machines to remove the constraints of human limitations ie. a customer representative won't be available 24/7 to solve the queries of the customers and if the queries are not resolved then the company might lose its customers and its brand value goes down. In such scenarios chatbots come in handy. Chatbots are considered the future of customer service and management. This project revolves around the domain of Deep Learning and the creation of a small artificial mind to handle basic queries in the form of a chatbot. A chatbot has varied applications throughout different domains and plays a vital role in cut shorting the human workforce where it is minimal. At times, along with much better features, a chatbot acts as a personal assistant who almost has everything we need to know. In this project we developed an AI assistant using pytorch and natural language processing toolkit so that the chatbot can understand the human language and give the response accordingly. Thus we developed a customer service chatbot by implementing neural network with backpropagation so that the bot learns from its inputs and improves the mistakes done in the past. This makes the bot reliable and accurate and it behaves exactly like a customer representative. Also we have connected our AI Assistant to the internet via APIs which substantially increases its knowledge base.

Domain Selection:-

The domain of our AI Assistant is Deep Learning. Deep learning is an AI function that mimics the workings of the human brain in processing data for use in detecting objects, recognizing speech, translating languages, and making decisions. A deep learning chatbot learns everything right from scratch, from its data and human-to-human dialogue. Nowadays, every industry engages in promoting their business online for better reachability and handling huge amounts of queries/requests merely by humans would lead to wastage in time and energy. Also, the majority of queries are fundamental which would not require human interaction, leaving just a small amount to be handled by humans.

b. Background Knowledge

Chatbots are self-help tools for improving communication. Brands use these AI powered chatbots to improve their customer's experience, to generate more sales and build a deeper rapport with customers. They allow the customers to easily interact with the respective brands through stimulated conversations. We know that a customer representative may not be present all the time that's when chatbots come into picture as they offer 24/7 customer service support .The customers need not wait several days before their queries are attended. Interacting with the earlier versions of chatbots was frustrating and time consuming as they responded to very specific input and couldn't process any information outside these parameters .Therefore it was less appealing for consumers than speaking to customer representative .Nowadays AI powered chatbots have come into action which enables the bot to mimic human conversation.

These bots learn from past conversations and improve their ability to provide appropriate solutions. Thus they can afford to keep customers happy, give them the best services and generate more sales.

c. Block diagram

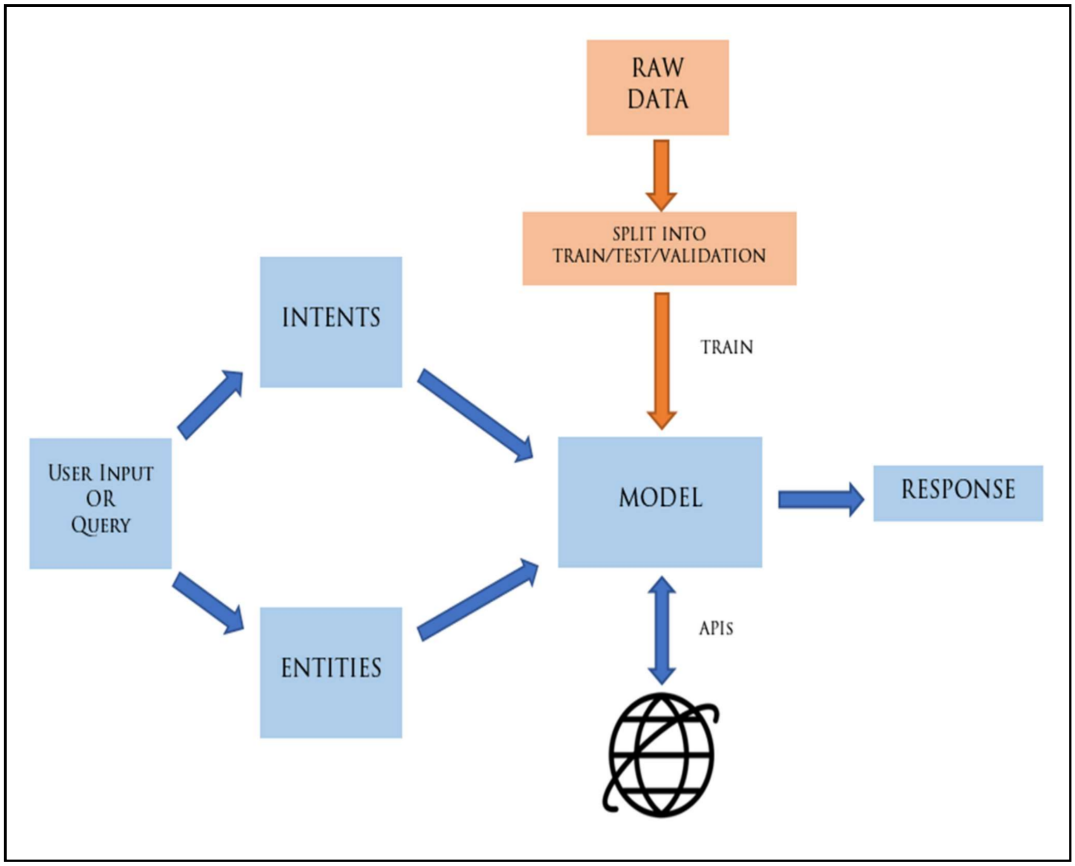


Fig 1: Schematic Diagram of AI Assistant

d. Flowchart

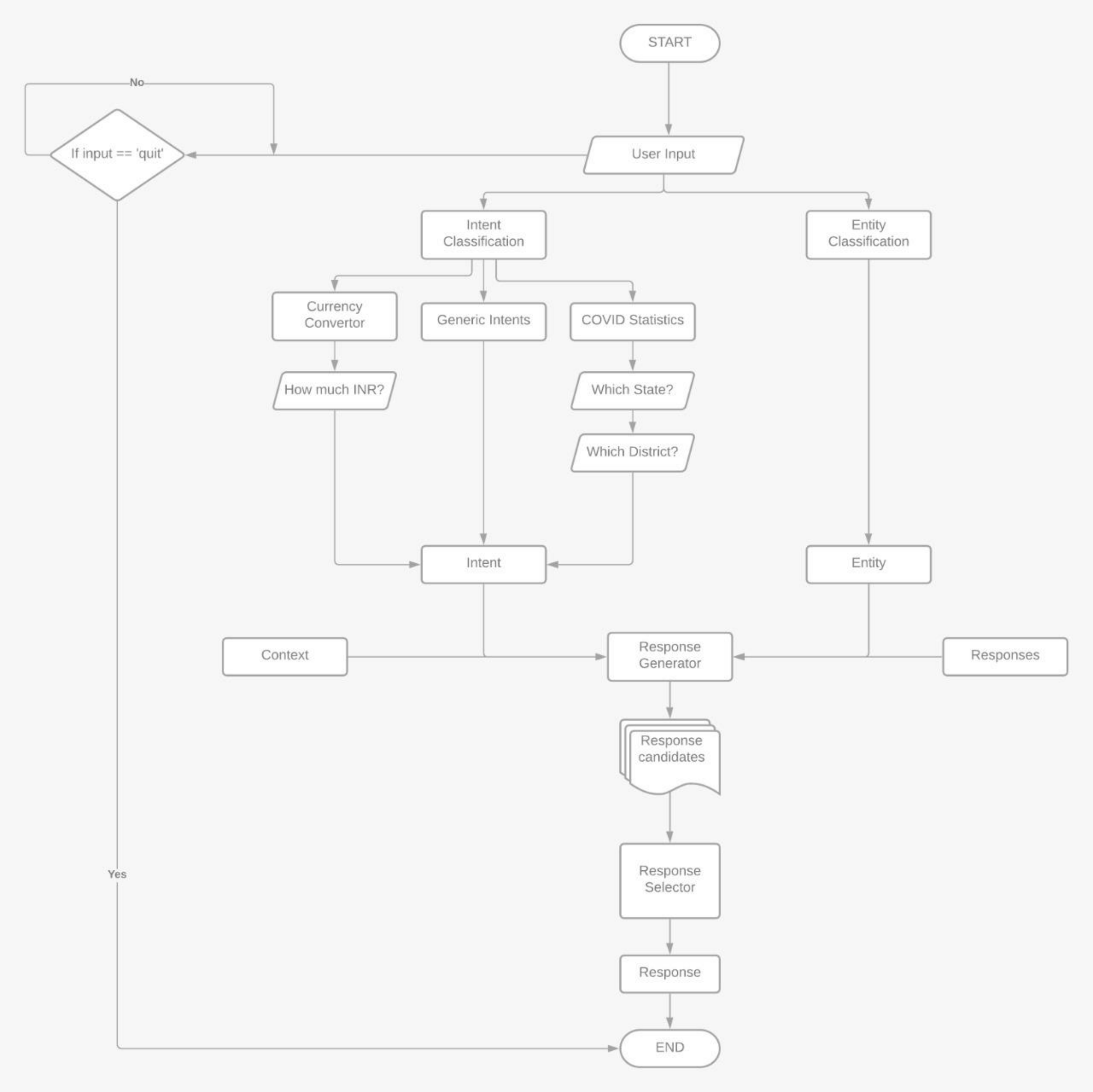


Fig 2: Flowchart of the project

e. Algorithm

Step 1: Start

Step 2: Setup environment

Step 3: Create training data

Step 4: Understanding NLP techniques and NLP Preprocessing pipeline

Step 5: Implement nlp utils

Step 6: Implement the neural network

Step 7: Implement the training pipeline

Step 8: Implement Chat

Step 9: Stop

f. Code

```
import numpy as np
import random
import json
import nltk
nltk.download('punkt')
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import requests


def tokenize(sentence):
return nltk.word_tokenize(sentence)

def stem(word):
return stemmer.stem(word.lower())


from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()


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


#base_url = "http://data.fixer.io/api/latest?access_key=8f78ecd20199e7942ef6ac4674c53c26&format=1"
#response = requests.get(base_url)
base_url = "https://free.currconv.com/api/v7/convert?apiKey=8b144724cd7d3e3cf9db"
response = requests.get(base_url)

base_url1 = "https://api.covid19india.org"
response1 = requests.get(base_url1)


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

#print(intents)


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))


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)


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)


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


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


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)


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}')


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

sentence = tokenize(sentence)
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"]:
if tag == "currency":
print(f"{bot_name}: How much INR? : ")
user_val = input("You: ")
param_url = base_url + ''.join(intent['responses'])
#print(param_url)
response = requests.get(param_url)
data = response.json()
```

```
sub_data = data['results']
sub_data = sub_data['USD_INR']
sub_data = sub_data['val']
print(f'{bot_name}: {float(user_val)/float(sub_data):.4f} USD')
elif tag == "state-covid":
param_url1 = base_url1 + ''.join(intent['responses'])
#print(param_url1)
response1 = requests.get(param_url1)
#print(response1.json())
#response1.status_code
j = response1.json()
#print(j)
print(f"{bot_name}: Which state? ")
state = input("You: ")
j1 = j[state]
print(f"{bot_name}: Which District? ")
district = input("You: ")
j2 = j1['districtData']
j3 = j2[district]
j4 = j3['active']
j5 = j3['confirmed']
j6 = j3['deceased']
j7 = j3['recovered']
print(f"{bot_name}: Number of active cases in " + district + " : " + str(j4) + "\n     Number
of confirmed cases in "+ district + " : " + str(j5) + "\n     Number of deceased in "+ district
+ " : " + str(j6) + "\n     Number of recovered in "+ district + " : " + str(j7))
else:
print(f"{bot_name}: {random.choice(intent['responses'])}")
else:
print(f"{bot_name}: I do not understand...")
```

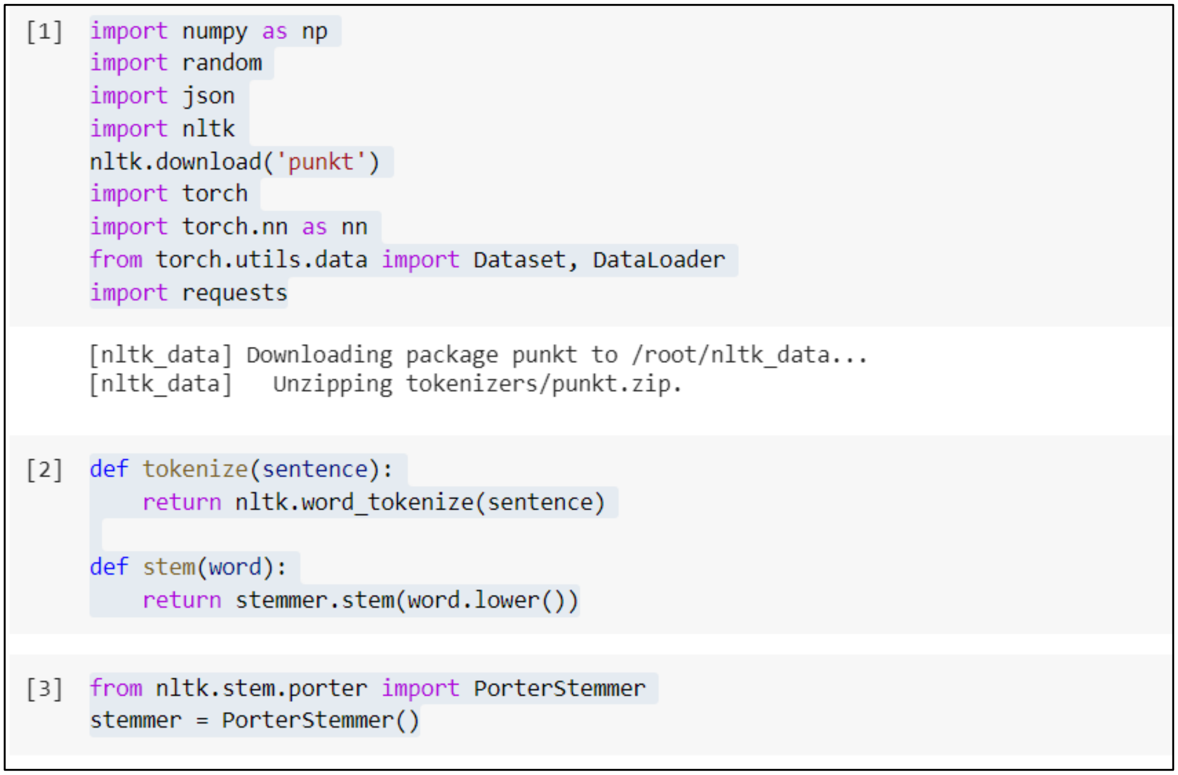
Dataset- Intents.json:

```
{
  "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": "currency",
      "patterns": ["Convert INR to USD", "INR to USD CC", "How much in USD Conversion?"
],
      "responses": ["&q=USD_INR"]
```

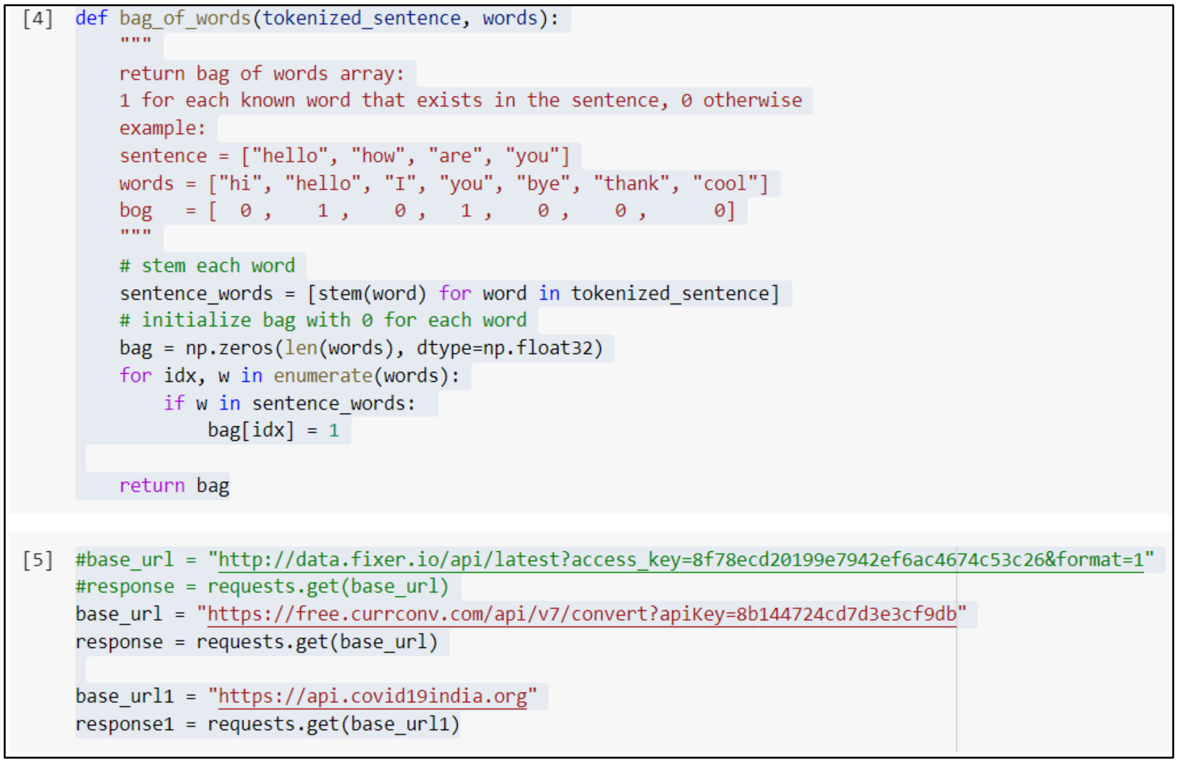
```
},
{
  "tag": "state-covid",
  "patterns": [
    "How many cases are there around me?",
    "Cases near me?"
  ],
  "responses": ["/state_district_wise.json"]
},
{
  "tag": "items",
  "patterns": [
    "Which items do you have?",
    "What kinds of items are there?",
    "What do you sell?"
  ],
  "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": "funny",
      "patterns": [
        "Tell me a joke!",
        "Tell me something funny!",
        "Do you know a joke?"
      ],
      "responses": [
        "Why did the hipster burn his mouth? He drank the coffee before it was cool.",
        "What did the buffalo say when his son left for college? Bison."
      ]
    }
  ]
}
```

g. Outputs

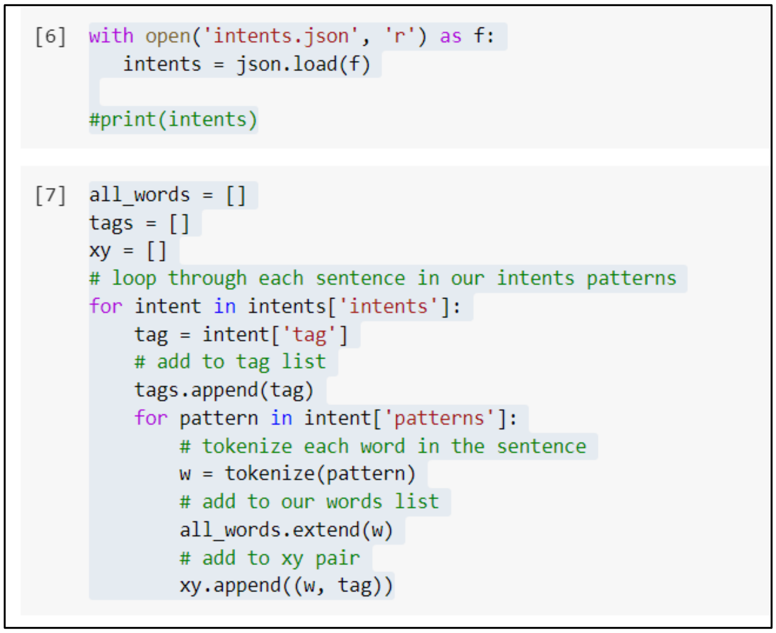


```
[1]  import numpy as np
     import random
     import json
     import nltk
     nltk.download('punkt')
     import torch
     import torch.nn as nn
     from torch.utils.data import Dataset, DataLoader
     import requests

     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Unzipping tokenizers/punkt.zip.

[2]  def tokenize(sentence):
         return nltk.word_tokenize(sentence)

     def stem(word):
         return stemmer.stem(word.lower())

[3]  from nltk.stem.porter import PorterStemmer
     stemmer = PorterStemmer()
```



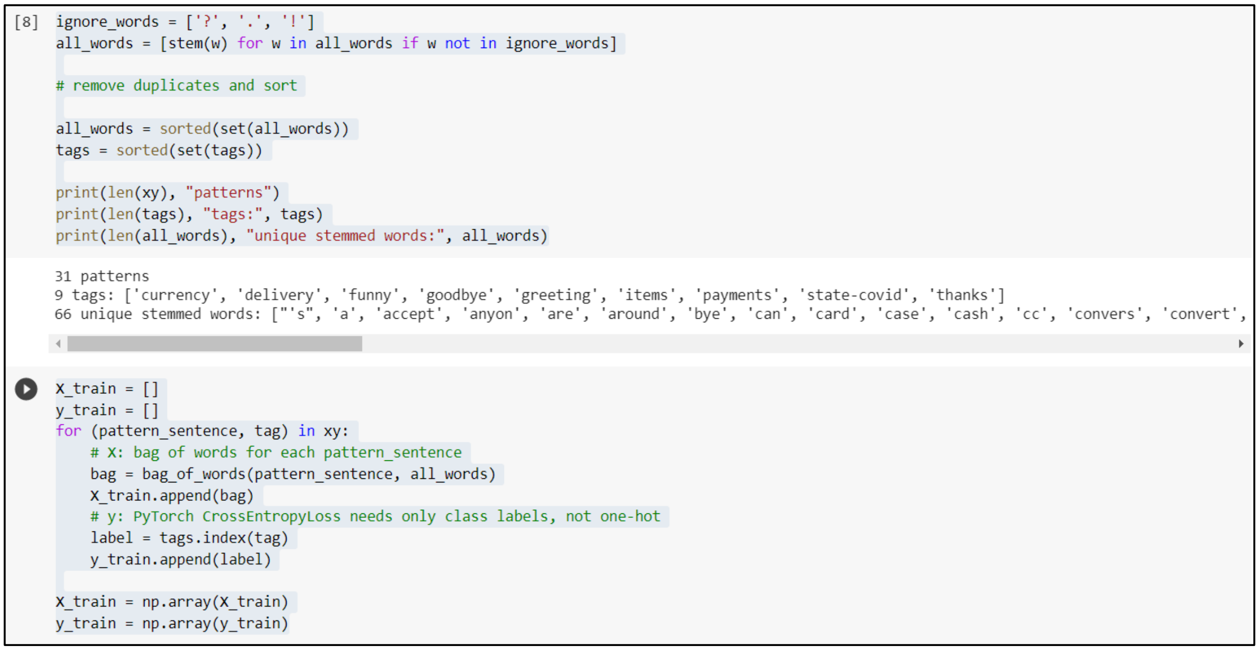
```
[4]  def bag_of_words(tokenized_sentence, words):
         """
         return bag of words array:
         1 for each known word that exists in the sentence, 0 otherwise
         example:
         sentence = ["hello", "how", "are", "you"]
         words = ["hi", "hello", "I", "you", "bye", "thank", "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
```

```
[5]  #base_url = "http://data.fixer.io/api/latest?access_key=8f78ecd20199e7942ef6ac4674c53c26&format=1"
     #response = requests.get(base_url)
     base_url = "https://free.currconv.com/api/v7/convert?apiKey=8b144724cd7d3e3cf9db"
     response = requests.get(base_url)

     base_url1 = "https://api.covid19india.org"
     response1 = requests.get(base_url1)
```



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

     #print(intents)
```

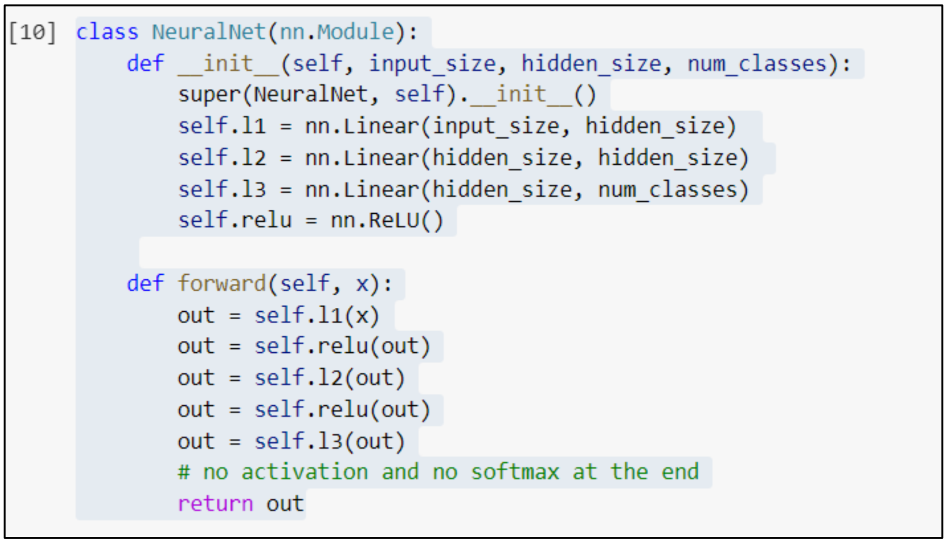
```
[7]  all_words = []
     tags = []
     xy = []
     # loop through each sentence in our intents patterns
     for intent in intents['intents']:
         tag = intent['tag']
         # add to tag list
         tags.append(tag)
         for pattern in intent['patterns']:
             # tokenize each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



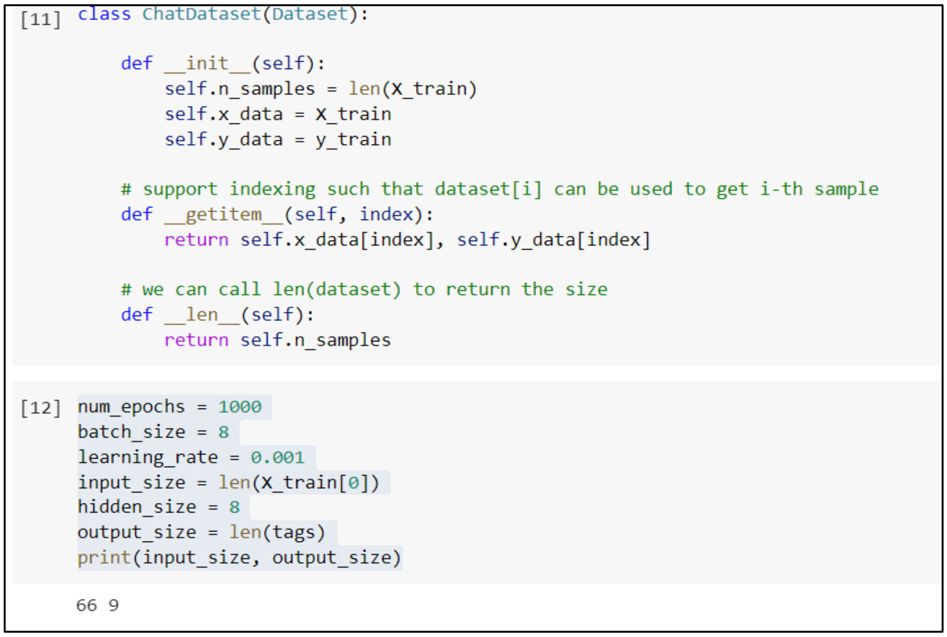
```
[8] ignore_words = ['?', '.', '!']
    all_words = [stem(w) for w in all_words if w not in ignore_words]

    # 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)
```

```
31 patterns
9 tags: ['currency', 'delivery', 'funny', 'goodbye', 'greeting', 'items', 'payments', 'state-covid', 'thanks']
66 unique stemmed words: ["'s", 'a', 'accept', 'anyon', 'are', 'around', 'bye', 'can', 'card', 'case', 'cash', 'cc', 'convers', 'convert',
```

```
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)
```



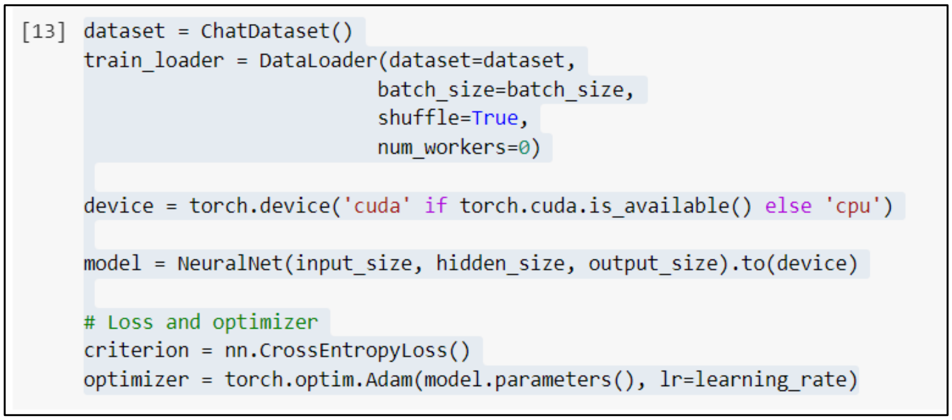
```
[10] class NeuralNet(nn.Module):
         def __init__(self, input_size, hidden_size, num_classes):
             super(NeuralNet, self).__init__()
             self.l1 = nn.Linear(input_size, hidden_size)
             self.l2 = nn.Linear(hidden_size, hidden_size)
             self.l3 = nn.Linear(hidden_size, 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
```



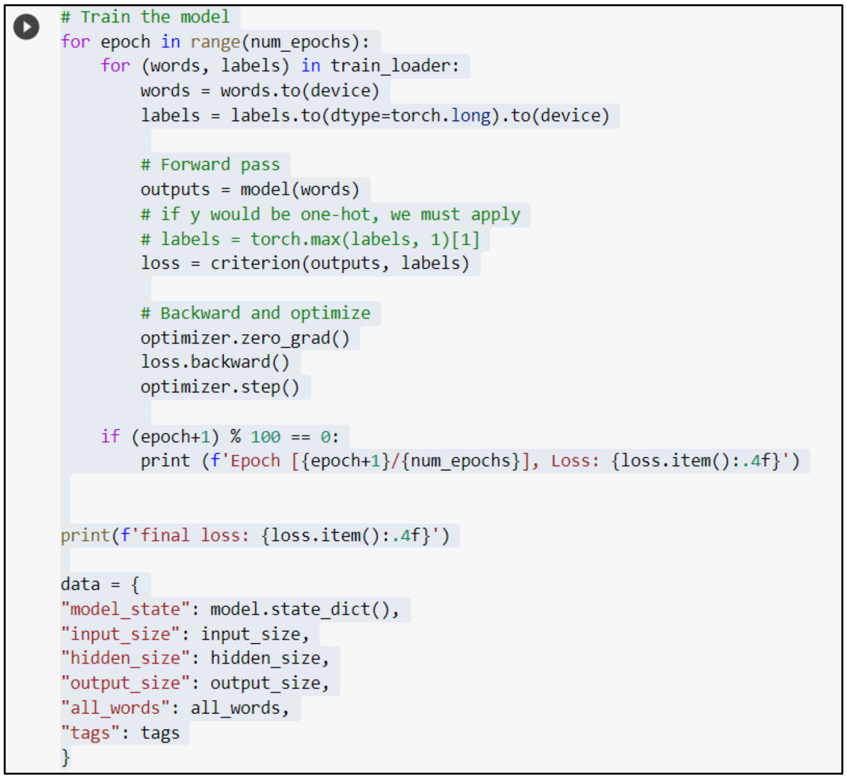
```
[11] class ChatDataset(Dataset):

         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
```

```
[12] 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)
```

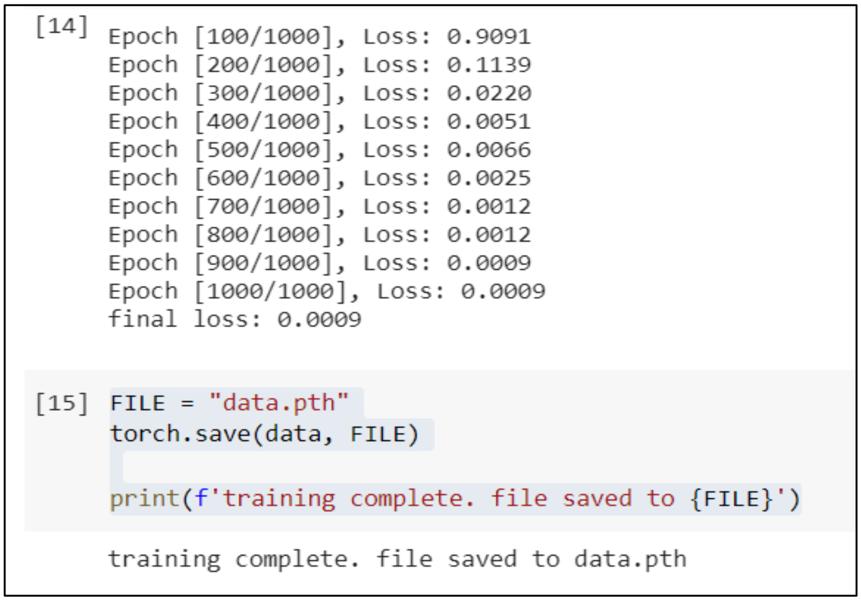
```
     66 9
```



```
[13] 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)
```



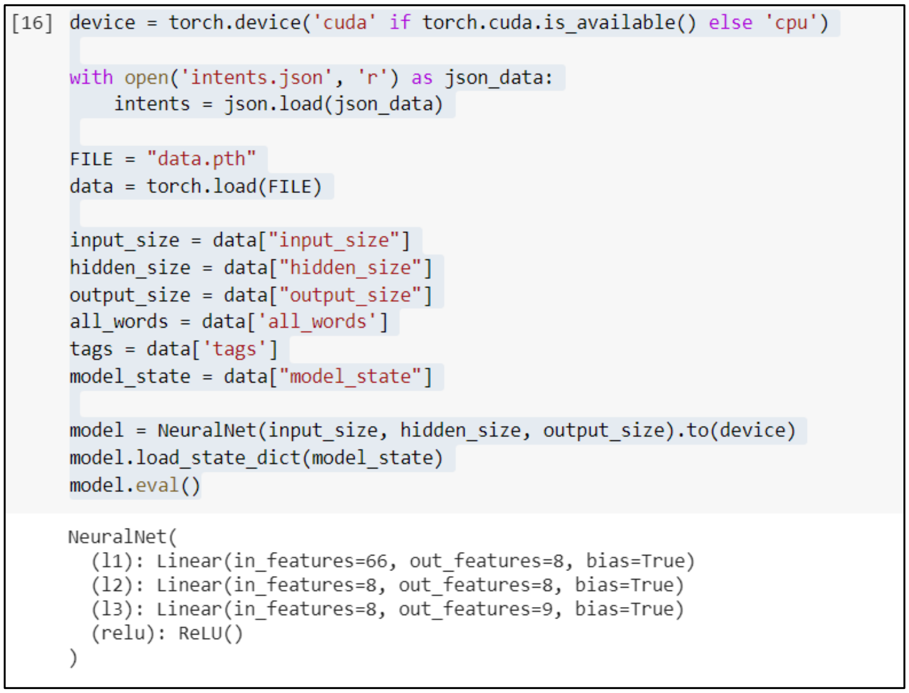
```
# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # 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
}
```



```
[14] Epoch [100/1000], Loss: 0.9091
     Epoch [200/1000], Loss: 0.1139
     Epoch [300/1000], Loss: 0.0220
     Epoch [400/1000], Loss: 0.0051
     Epoch [500/1000], Loss: 0.0066
     Epoch [600/1000], Loss: 0.0025
     Epoch [700/1000], Loss: 0.0012
     Epoch [800/1000], Loss: 0.0012
     Epoch [900/1000], Loss: 0.0009
     Epoch [1000/1000], Loss: 0.0009
     final loss: 0.0009
```

```
[15] FILE = "data.pth"
     torch.save(data, FILE)

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

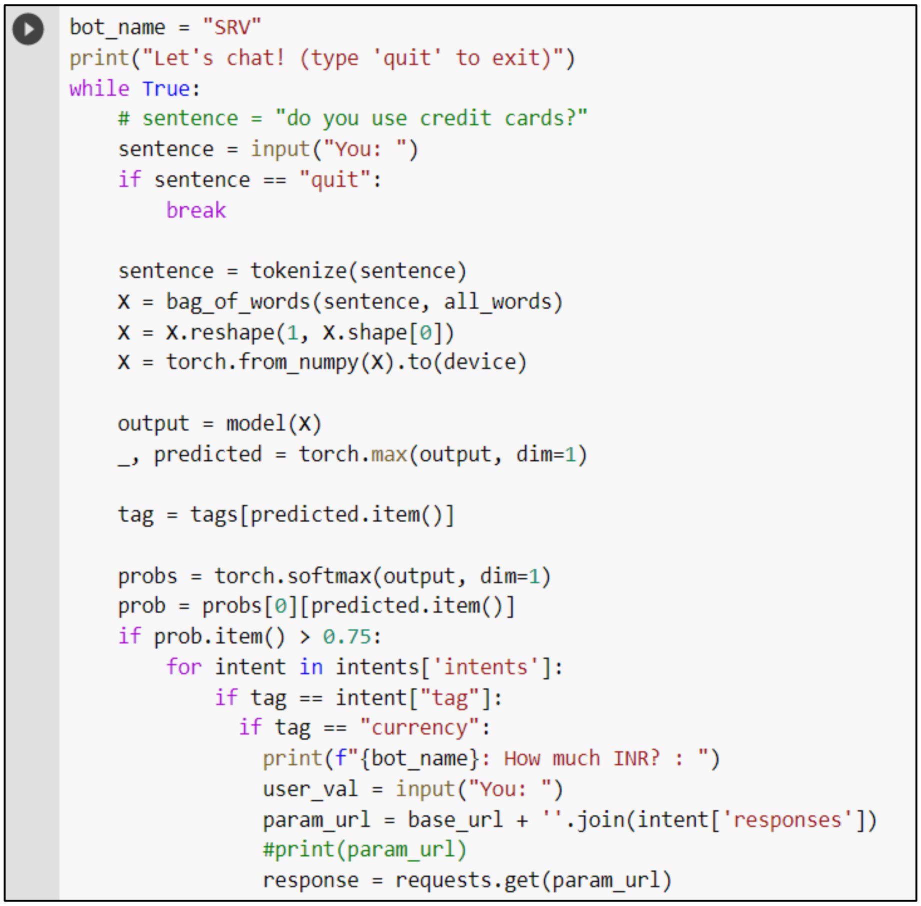
```
training complete. file saved to data.pth
```



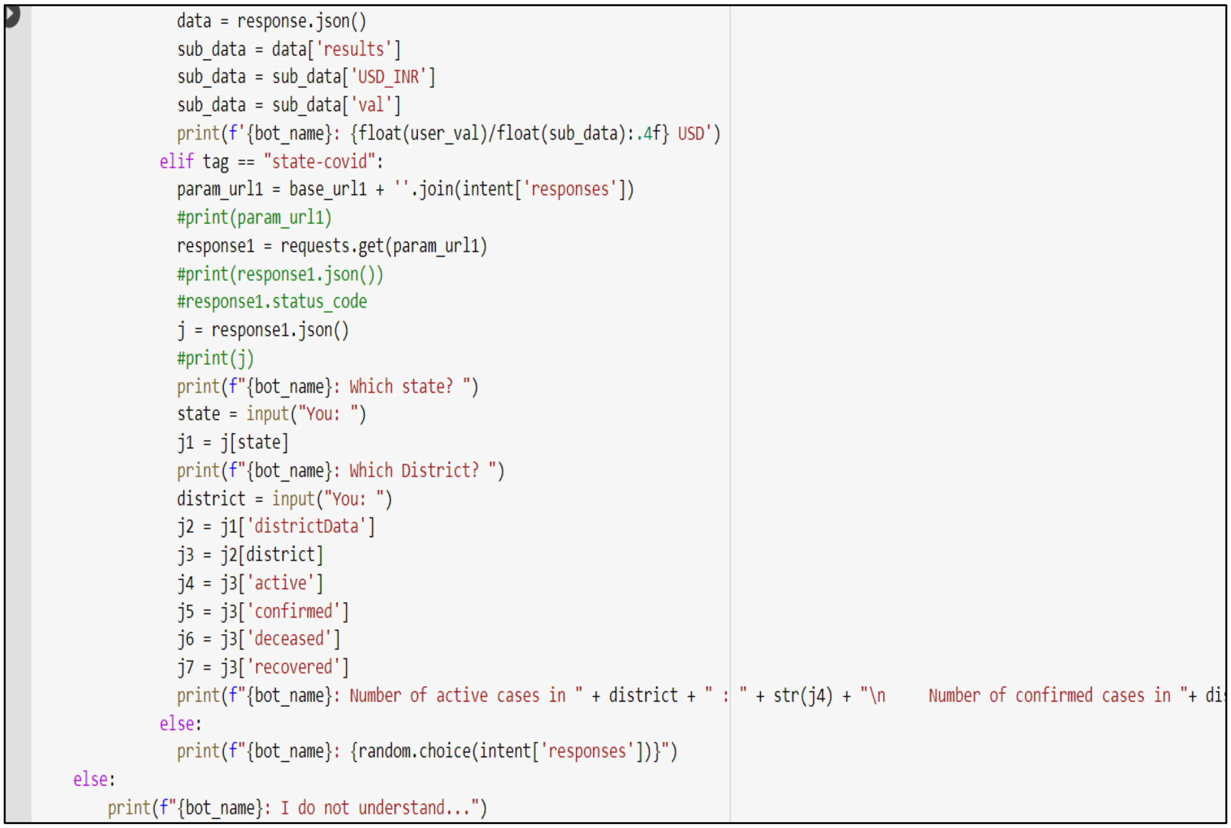
[16]

```
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()
```

```
NeuralNet(
  (l1): Linear(in_features=66, out_features=8, bias=True)
  (l2): Linear(in_features=8, out_features=8, bias=True)
  (l3): Linear(in_features=8, out_features=9, bias=True)
  (relu): ReLU()
)
```

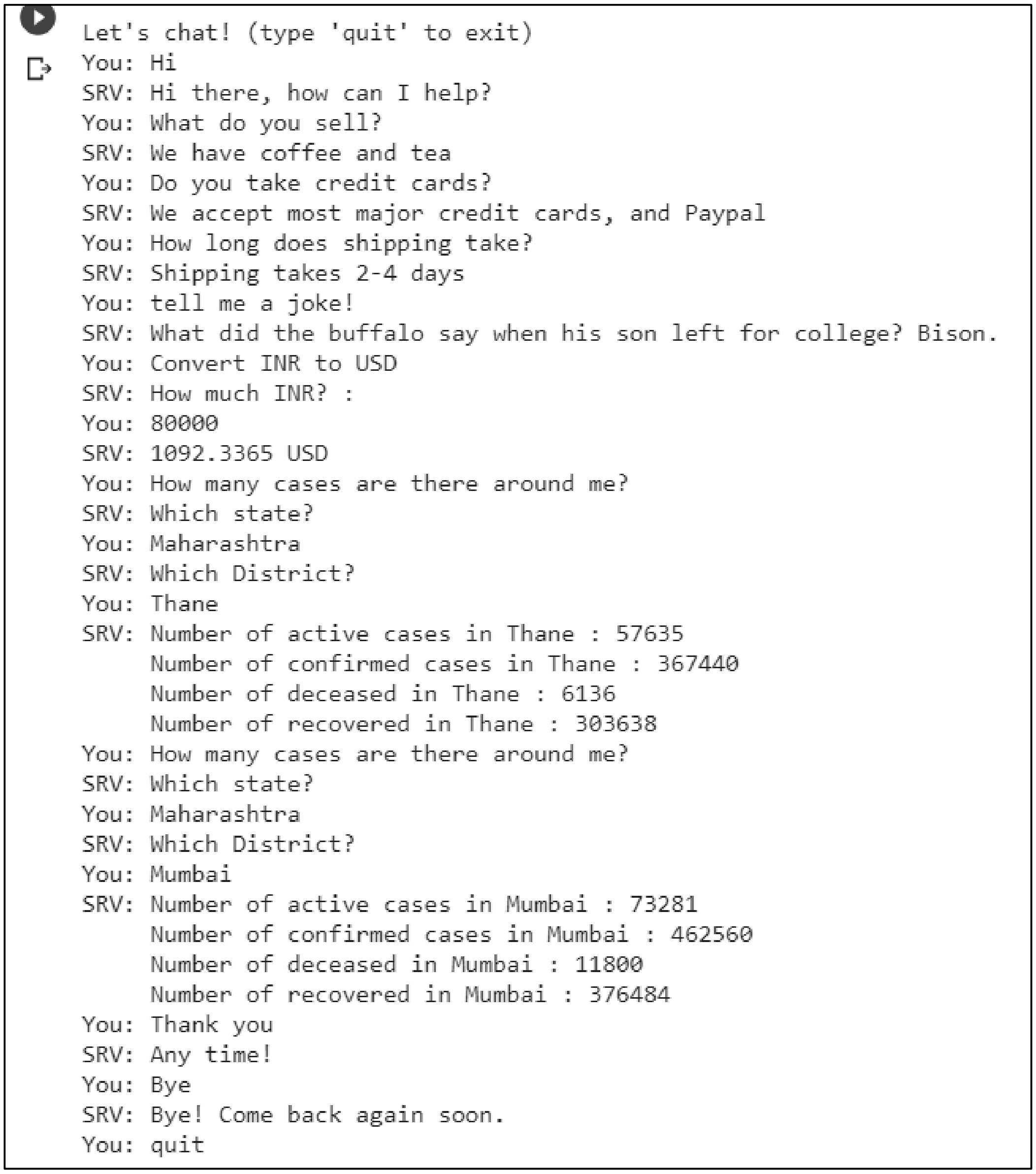


```
bot_name = "SRV"
print("Let's chat! (type 'quit' to exit)")
while True:
    # sentence = "do you use credit cards?"
    sentence = input("You: ")
    if sentence == "quit":
        break

    sentence = tokenize(sentence)
    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"]:
              if tag == "currency":
                print(f"{bot_name}: How much INR? : ")
                user_val = input("You: ")
                param_url = base_url + ''.join(intent['responses'])
                #print(param_url)
                response = requests.get(param_url)
```



```
                data = response.json()
                sub_data = data['results']
                sub_data = sub_data['USD_INR']
                sub_data = sub_data['val']
                print(f'{bot_name}: {float(user_val)/float(sub_data):.4f} USD')
              elif tag == "state-covid":
                param_url1 = base_url1 + ''.join(intent['responses'])
                #print(param_url1)
                response1 = requests.get(param_url1)
                #print(response1.json())
                #response1.status_code
                j = response1.json()
                #print(j)
                print(f"{bot_name}: Which state? ")
                state = input("You: ")
                j1 = j[state]
                print(f"{bot_name}: Which District? ")
                district = input("You: ")
                j2 = j1['districtData']
                j3 = j2[district]
                j4 = j3['active']
                j5 = j3['confirmed']
                j6 = j3['deceased']
                j7 = j3['recovered']
                print(f"{bot_name}: Number of active cases in " + district + " : " + str(j4) + "\n     Number of confirmed cases in "+ di
              else:
                print(f"{bot_name}: {random.choice(intent['responses'])}")
    else:
        print(f"{bot_name}: I do not understand...")
```

CUSTOMER SERVICE CHATBOT OUTPUT-



```
Let's chat! (type 'quit' to exit)
You: Hi
SRV: Hi there, how can I help?
You: What do you sell?
SRV: We have coffee and tea
You: Do you take credit cards?
SRV: We accept most major credit cards, and Paypal
You: How long does shipping take?
SRV: Shipping takes 2-4 days
You: tell me a joke!
SRV: What did the buffalo say when his son left for college? Bison.
You: Convert INR to USD
SRV: How much INR? :
You: 80000
SRV: 1092.3365 USD
You: How many cases are there around me?
SRV: Which state?
You: Maharashtra
SRV: Which District?
You: Thane
SRV: Number of active cases in Thane : 57635
     Number of confirmed cases in Thane : 367440
     Number of deceased in Thane : 6136
     Number of recovered in Thane : 303638
You: How many cases are there around me?
SRV: Which state?
You: Maharashtra
SRV: Which District?
You: Mumbai
SRV: Number of active cases in Mumbai : 73281
     Number of confirmed cases in Mumbai : 462560
     Number of deceased in Mumbai : 11800
     Number of recovered in Mumbai : 376484
You: Thank you
SRV: Any time!
You: Bye
SRV: Bye! Come back again soon.
You: quit
```

Advantages & Limitation of project

Advantages:-

- ✓ Systematically scale their chat support during peak hours to deliver quality support and enhance customer satisfaction
- ✓ The decision making process is fast
- ✓ It is flexible for every domain. We can change it as per the requirements.
- ✓ Chatbots can help you gather precious data from your customers by interacting with them. This includes getting insights about their activities, preferences, problems, and more
- ✓ It is available 24/7
- ✓ Customer engagement is the critical requirement to boost your sales and keep your customers engaged, and chatbots are an excellent tool for this.

Limitations:-

- ➢ Due to small dataset chance of overfitting arises in the learning model
- ➢ Cannot handle complex queries that requires human intelligence
- ➢ Chatbots require ongoing review, maintenance, and optimization in terms of their knowledge base and the way they are supposed to communicate with your customers.
- ➢ It is very challenging to create a chatbot from scratch.

Conclusion:

In this project based experiment we learnt about the basics of Artificial Intelligence, role of AI in various domains, its applications and benefits. AI makes it possible for machines to learn from experience, adjust to new inputs and perform human like tasks. We then decided to utilize our theoretical knowledge and apply the same to real life applications by developing a customer service chatbot which is useful to handle basic queries of the customers. We used pytorch and natural language processing toolkit for building the AI assistant. We also connected our AI Assistant to the internet via APIs which substantially increases its knowledge base. This project revolves around the domain of deep learning. Here we developed a customer service chatbot by implementing neural network with backpropagation so as to make the bot reliable and accurate. We first created a neural net module. After that we imported the neural net module in the training file where we implemented backpropagation so that the error reduces after each iteration making the bot more accurate. This is a very important aspect as the chatbot has to mimic the customer representative so in such cases the output should be accurate with minimum losses and that can only be achieved through backpropagation. After successful implementation of the code, we observed that the losses were minimum which means that our bot is reliable. Thus we can conclude that our AI Assistant is able to resolve the customer related queries accurately.

Future Scope:

- We can increase the existing dataset through surveys, google forms, etc. which will make the bot more accurate
- If questions become more complex an escalation path should be provided to escalate the interaction to customer representative for better results
- We can also add empathy to chatbot responses with sentiment analysis. It helps the bot respond empathetically to frustrated users and prioritize sensitive situations.
- We can integrate speech recognition within the chatbot to give voice commands