Creating Chatbot Using Natural Language Processing

Lecturer :
D4017_Jude Joseph Lamug Martinez, MCS

Arranged by :

2702368122_Vammy Johannis Jiang

Algorithm and Progamming

Faculty of Computer Science

Binus International University 2022/2023

Table of Contents

Chapter 1 – Project Specification

1.1 Introduction

This project was made using python programming language. The main purpose of this project is to make a fully functional chatbot which can answer question about Binus university. The chatbot is made by using natural language processing as its model.

1.2 Idea Inspiration

The idea came from my brother when I did not know what I wanted to make for the final project. He proposed that I should ask Chatgpt for project ideas. I proceeded to ask Chatgpt for project ideas and natural language processing caught my eye immediately so I decided to make a chatbot based on natural language processing.

1.3 How it Works

The project uses Natural Language Toolkit (NLTK) for text tokenization and stemming. The chatbot project utilizes a JSON file to define user intents with associated pattern and responses. The project also use PyTorch library for constructing and training a basic feedforward neural network. In which the json file is used as training data for the model after the pre-processing process.

Chapter 2 – Solution Design

2.1 Solution Overview

The program consists of 5 components :

- Chat.py
- Model.py
- Nltk_utils.py
- Train.py
- Intents.json

2.2 Solution Description

Nltk_utils.py

The utility tools for tokenization, stemming and bag of words part of pre-processing of a natural language processing language.

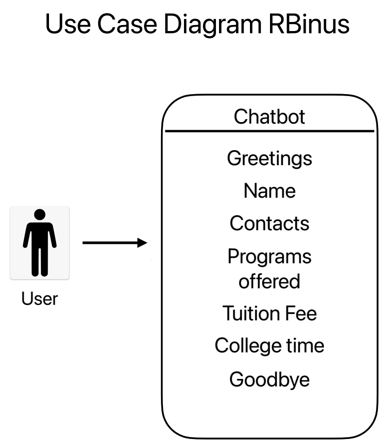
1. Model.py
   The model of the neural network module of the chatbot.

2. Intents.json
   The training data for our chatbot with consist of tags, patterns, and responses

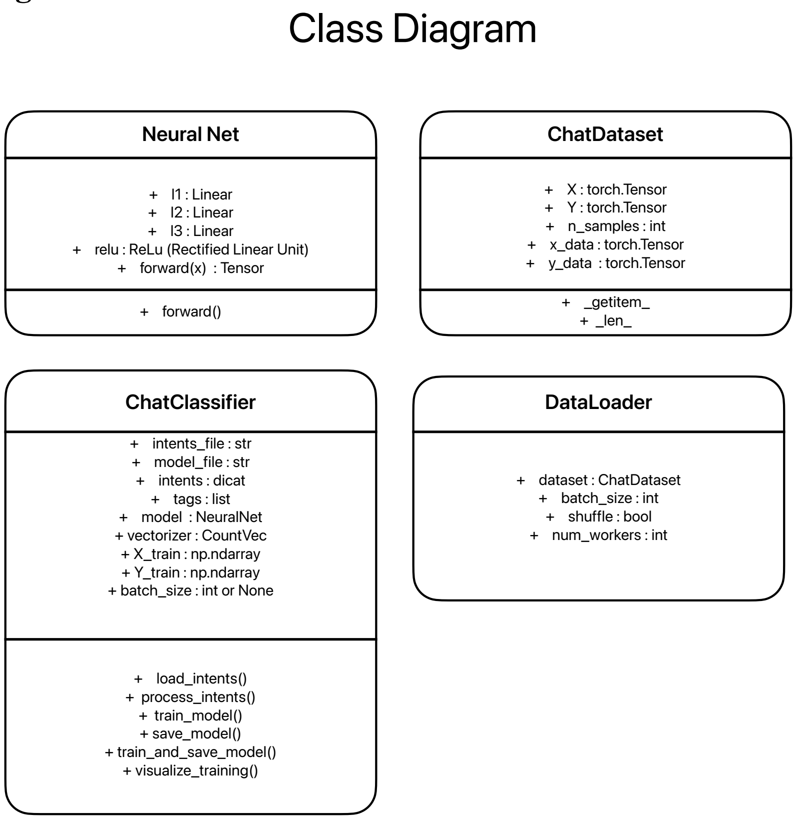
3. Train.py
   The training module for the chatbot. It consists of ChatClassifier which is used for training and saving the chatbot model. And ChatDataSet where it is a PyTorch Dataset for handling and training data.

4. Chat.py
   The main program used to activate the chatbot consists of the main body of the chatbot itself.

2.3 Use Case Diagram



2.4 Class Diagram



Chapter 3 - Implementation and Explanation

3.1 Modules Used

1. NumPy (import numpy as np):

- NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

2. Random (import random):

- The random module provides functions for generating pseudo-random numbers. It is used here to introduce randomness, such as selecting responses randomly in the chatbot.

3. JSON (import json):

- The json module is used for encoding and decoding JSON (JavaScript Object Notation) data. In this code, it is used to load JSON data from a file (intents.json).

4. PyTorch (import torch, import torch.nn as nn, from torch.utils.data import Dataset, DataLoader):

- PyTorch is an open-source machine learning library for Python. torch is the main PyTorch module, providing multi-dimensional tensors and mathematical operations. torch.nn contains neural network modules and functionalities.
- Dataset and DataLoader are classes in PyTorch's torch.utils.data module for handling datasets and creating data loaders for efficient batching.

5. Scikit-Learn (from sklearn.feature_extraction.text import CountVectorizer, from sklearn.model_selection import train_test_split):

- Scikit-Learn is a machine learning library for Python.
- CountVectorizer is used for converting a collection of text documents to a matrix of token counts. train_test_split is used for splitting datasets into training and testing sets.

6. NLTK (import nltk):

- NLTK (Natural Language Toolkit) is a powerful library for working with human language data. nltk.download('punkt') downloads the Punkt tokenizer models required for tokenization.
- PorterStemmer is a stemming algorithm used to reduce words to their root or base form.

3.2 How it works

First before we know how the chatbot works, we will need to know what natural language processing is. Natural Language Processing (NLP) is an interdisciplinary subfield of computer science and linguistics that focuses on giving computers the ability to support and manipulate human language. NLP combines computational linguistics, machine learning, and deep learning models to process human language in the form of text or voice data and understand the intent and sentiment behind the language. NLP is used for a wide variety of language-related tasks, including answering questions, classifying text in various ways, and conversing with users .Some key applications of NLP include:

- Sentiment analysis: Classifying the emotional intent of text
- Text classification: Categorizing text data based on various criteria
- Entity recognition: Identifying and extracting specific entities from the text
- Text generation: Creating human-like text using machine learning models
- Speech recognition: Parsing spoken language into words and turning sound into text

NLP is essential for processing and analyzing text and speech data, and it can work through differences in dialects, slang, and grammatical irregularities typical in day-to-day conversations. Companies use NLP for various automated tasks, such as processing, analyzing, and archiving large documents, analyzing customer feedback or call center recordings, running chatbots for automated customer service, and answering who-what-when-where question.

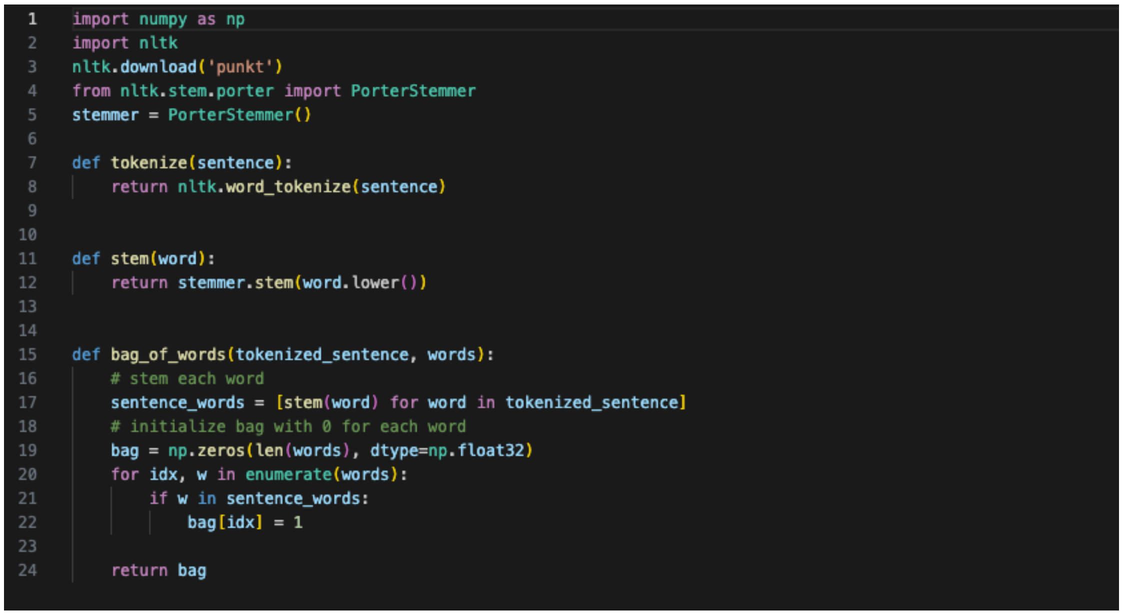
In the context of chatbots, Natural Language Processing (NLP) plays a crucial role in enabling machines to understand and respond to text or voice data, capturing the intent and sentiment behind the language. NLP works in chatbots by using a combination of techniques and algorithms to analyse and process human language.

Here are some key components of how NLP works in chatbots:

- Text processing: NLP involves techniques such as tokenization, stemming, lemmatization, and sentiment analysis to process and understand the input text. Tokenization is the process of breaking down text into individual units, such as words or phrases, while stemming and lemmatization are methods of reducing words down to their roots to group related words together.

- Intent recognition: NLP algorithms are used to identify the underlying intent of the user's input, such as making a purchase, asking a question, or seeking assistance.
- Contextual understanding: NLP models can capture contextual information from the user's input, such as the tone, style, and context of the conversation.This allows the chatbot to generate more human-like and relevant responses based on the user's context and previous interactions.
- Machine learning: NLP often involves the use of machine learning algorithms to improve the chatbot's performance over time.

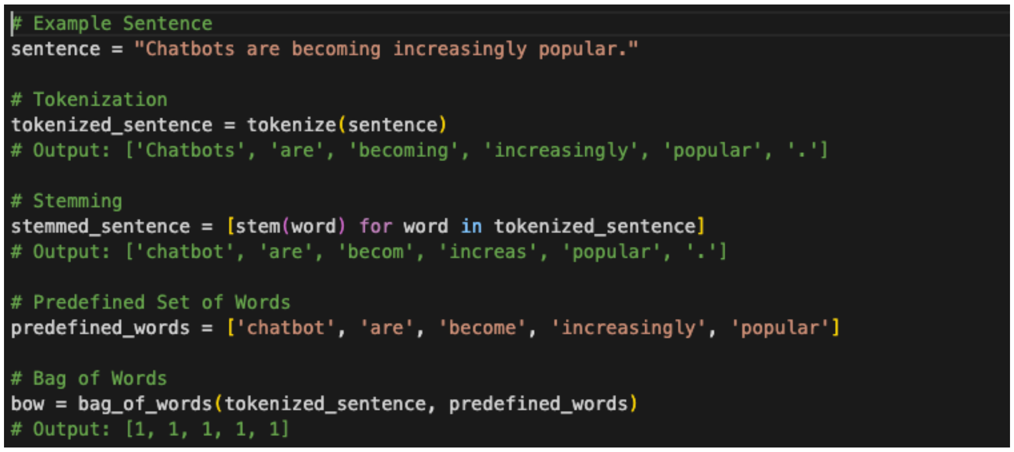
We will start with the first process of natural language processing which is text processing with functions defined in nltkutils.py



```
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
```

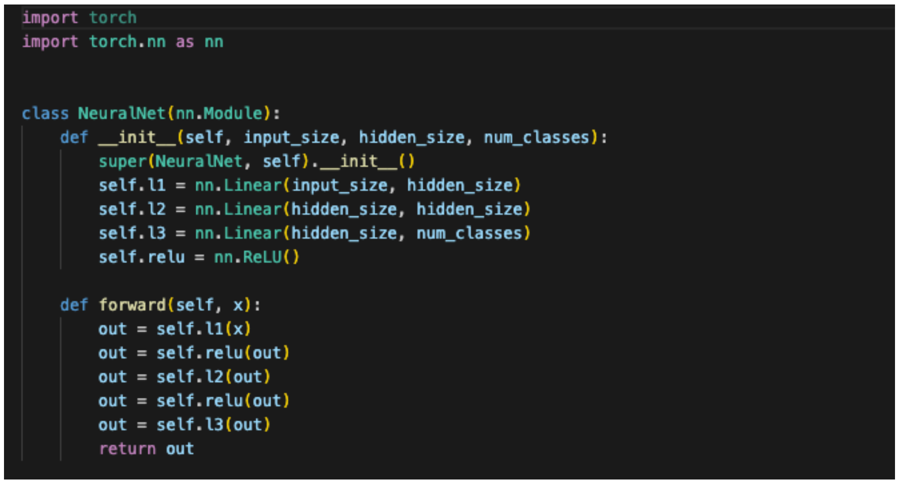
As we can see from the code above, the code first import libraries such as numpy and Natural Language Toolkit (NLTK). After that, 3 main functions which consist of tokenize, stem, bag of words. Tokenize function utilizes the NLTK library to tokenize a given sentence into a list of words or tokens. Stem from NLTK library is used to perform stemming on a given word. Stemming involves reducing a words to its base or root form removing prefixes or suffixes. Bag of words function creates a bag of words representation for a tokenized sentence based on 0 and 1s.

Example of each function is shown below



```
# Example Sentence
sentence = "Chatbots are becoming increasingly popular."

# Tokenization
tokenized_sentence = tokenize(sentence)
# Output: ['Chatbots', 'are', 'becoming', 'increasingly', 'popular', '.']

# Stemming
stemmed_sentence = [stem(word) for word in tokenized_sentence]
# Output: ['chatbot', 'are', 'becom', 'increas', 'popular', '.']

# Predefined Set of Words
predefined_words = ['chatbot', 'are', 'become', 'increasingly', 'popular']

# Bag of Words
bow = bag_of_words(tokenized_sentence, predefined_words)
# Output: [1, 1, 1, 1, 1]
```

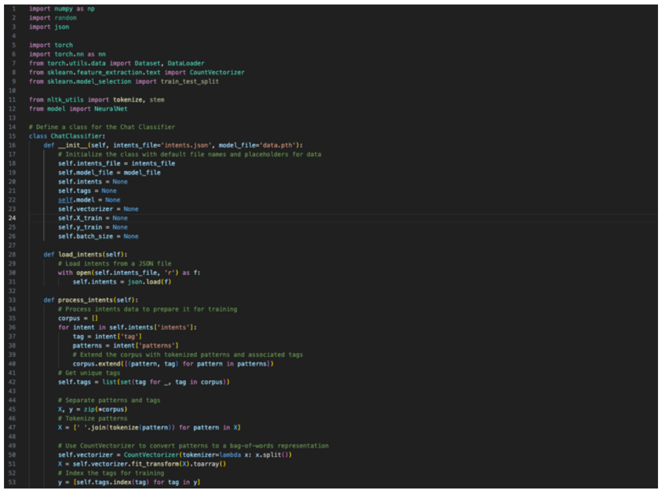
Next is the model.py component :



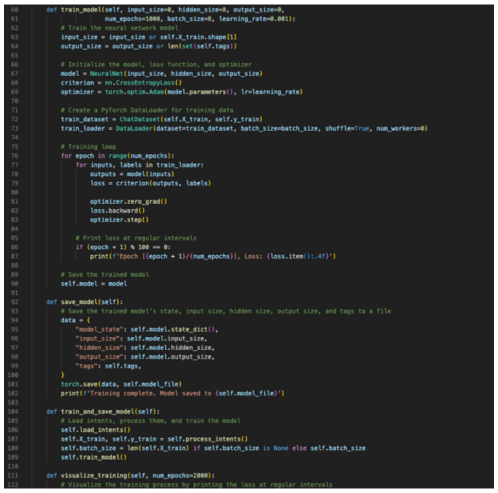
```
import torch
import torch.nn as nn


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
```

The code above showcases a neural network class using PyTorch for a classification task. The network is defined by 3 layer of linear unit (ReLU) activation functions. This neural network is a feed forward neural network used for the training on the chatbot later on. The role of this neural network is in it ability to learn and comprehend complex relationships within input data such as patterns in user queries or sentences.

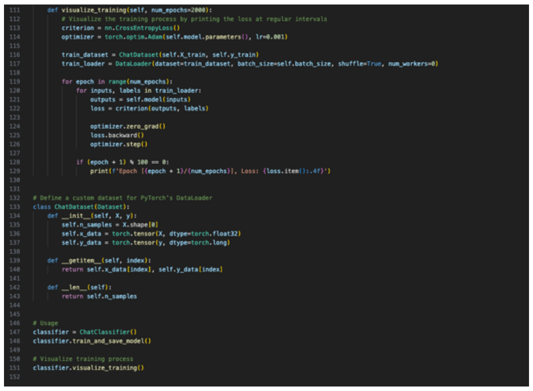
Next is train.py part :



```python
import numpy as np
import random
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from nltk_utils import tokenize, stem
from model import NeuralNet

# Define a class for the Chat Classifier
class ChatClassifier:
    def __init__(self, intents_file='intents.json', model_file='data.pth'):
        # Initialize the class with default file names and placeholders for data
        self.intents_file = intents_file
        self.model_file = model_file
        self.intents = None
        self.tags = None
        self.model = None
        self.vectorizer = None
        self.X_train = None
        self.y_train = None
        self.batch_size = None

    def load_intents(self):
        # Load intents from a JSON file
        with open(self.intents_file, 'r') as f:
            self.intents = json.load(f)

    def process_intents(self):
        # Process intents data to prepare it for training
        corpus = []
        for intent in self.intents['intents']:
            tag = intent['tag']
            patterns = intent['patterns']
            # Extend the corpus with tokenized patterns and associated tags
            corpus.extend([(pattern, tag) for pattern in patterns])
        # Get unique tags
        self.tags = list(set(tag for _, tag in corpus))

        # Separate patterns and tags
        X, y = zip(*corpus)
        # Tokenize patterns
        X = [' '.join(tokenize(pattern)) for pattern in X]

        # Use CountVectorizer to convert patterns to a bag-of-words representation
        self.vectorizer = CountVectorizer(tokenizer=lambda x: x.split())
        X = self.vectorizer.fit_transform(X).toarray()
        # Index the tags for training
        y = [self.tags.index(tag) for tag in y]
```



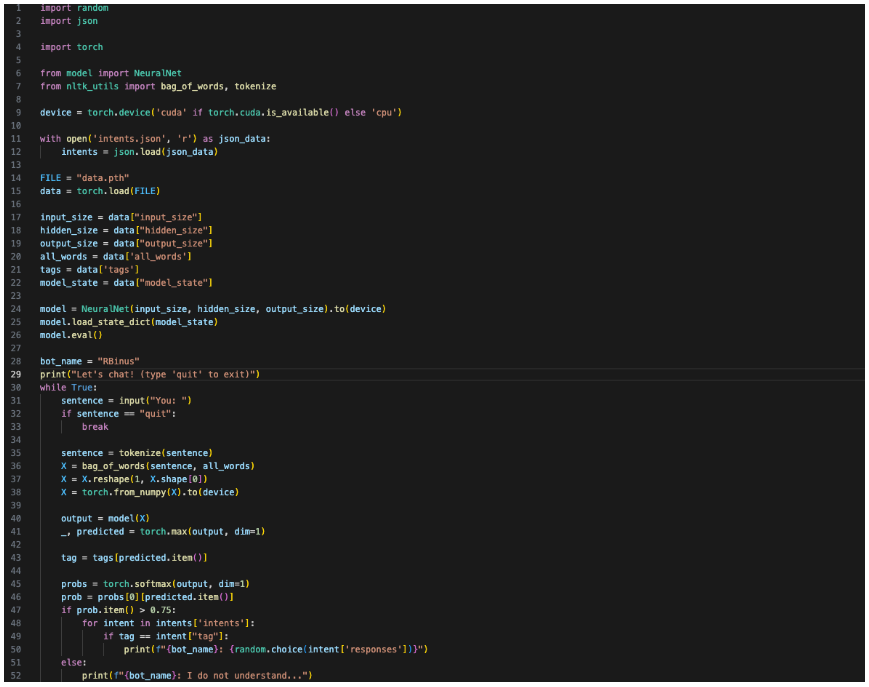
```python
    def train_model(self, input_size=0, hidden_size=8, output_size=0,
                    num_epochs=1000, batch_size=8, learning_rate=0.001):
        # Train the neural network model
        input_size = input_size or self.X_train.shape[1]
        output_size = output_size or len(set(self.tags))

        # Initialize the model, loss function, and optimizer
        model = NeuralNet(input_size, hidden_size, output_size)
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

        # Create a PyTorch DataLoader for training data
        train_dataset = ChatDataset(self.X_train, self.y_train)
        train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True, num_workers=0)

        # Training loop
        for epoch in range(num_epochs):
            for inputs, labels in train_loader:
                outputs = model(inputs)
                loss = criterion(outputs, labels)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            # Print loss at regular intervals
            if (epoch + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

        # Save the trained model
        self.model = model

    def save_model(self):
        # Save the trained model's state, input size, hidden size, output size, and tags to a file
        data = {
            "model_state": self.model.state_dict(),
            "input_size": self.model.input_size,
            "hidden_size": self.model.hidden_size,
            "output_size": self.model.output_size,
            "tags": self.tags,
        }
        torch.save(data, self.model_file)
        print(f'Training complete. Model saved to {self.model_file}')

    def train_and_save_model(self):
        # Load intents, process them, and train the model
        self.load_intents()
        self.X_train, self.y_train = self.process_intents()
        self.batch_size = len(self.X_train) if self.batch_size is None else self.batch_size
        self.train_model()

    def visualize_training(self, num_epochs=2000):
        # Visualize the training process by printing the loss at regular intervals
```



```python
    def visualize_training(self, num_epochs=2000):
        # Visualize the training process by printing the loss at regular intervals
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(self.model.parameters(), lr=0.001)

        train_dataset = ChatDataset(self.X_train, self.y_train)
        train_loader = DataLoader(dataset=train_dataset, batch_size=self.batch_size, shuffle=True, num_workers=0)

        for epoch in range(num_epochs):
            for inputs, labels in train_loader:
                outputs = self.model(inputs)
                loss = criterion(outputs, labels)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            if (epoch + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')


# Define a custom dataset for PyTorch's DataLoader
class ChatDataset(Dataset):
    def __init__(self, X, y):
        self.n_samples = X.shape[0]
        self.x_data = torch.tensor(X, dtype=torch.float32)
        self.y_data = torch.tensor(y, dtype=torch.long)

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.n_samples


# Usage
classifier = ChatClassifier()
classifier.train_and_save_model()

# Visualize training process
classifier.visualize_training()
```

The train.py code contains the ChatClassifier class that facilitates the training and usage of a neural network-based chatbot model. The class reads intent data from a JSON file, pre-processes it, and trains a feedforward neural network using PyTorch. The training involves converting text patterns into a bag-of-words representation using CountVectorizer and then training a neural network to predict the associated intent tags. The training progress can be visualized by printing the loss at regular intervals. The trained model is saved, allowing for subsequent chatbot interactions. The code also includes a custom dataset class for PyTorch's DataLoader.

Finally it is chat.py :



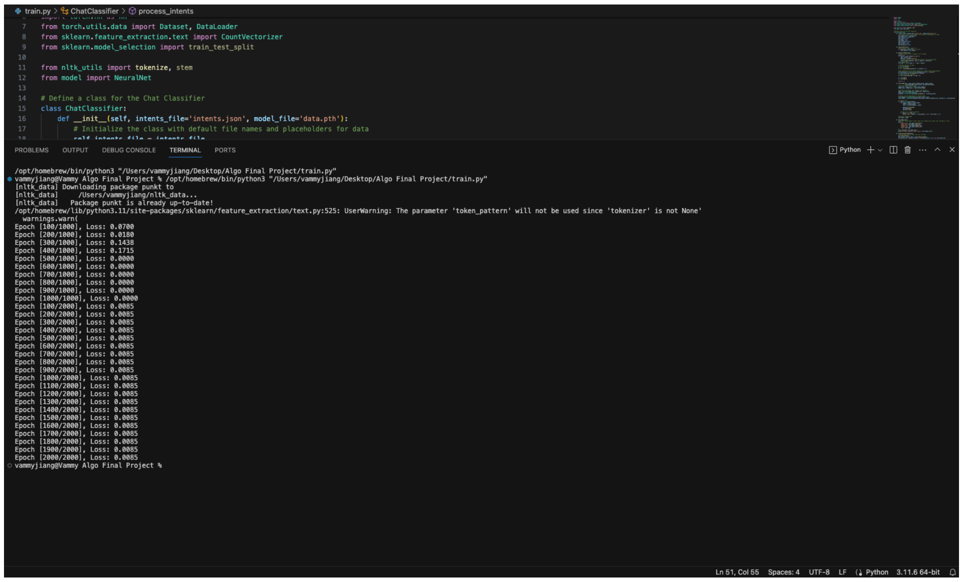
```python
import random
import json

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 = "RBinus"
print("Let's chat! (type 'quit' to exit)")
while True:
    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"]:
                print(f"{bot_name}: {random.choice(intent['responses'])}")
    else:
        print(f"{bot_name}: I do not understand...")
```

Chat.py employs a trained neural network from train.py for a chatbot application. The model is loaded from a saved state, and user input is tokenized and converted into a bag-of-words representation. The neural network predicts the intent of the input, and if the prediction surpasses a confidence threshold, the chatbot provides a relevant response randomly selected from predefined responses associated with the predicted intent. If the confidence falls below the threshold, the chatbot print out I do not understand.

The overall process to activating the chatbot is first, users will need to run the train.py until the neural network finishes training. Then, users can run chat.py to run the chatbot and ask it questions about Binus International.

Chapter 4 - Evidence of Working Program

First, when we run train.py

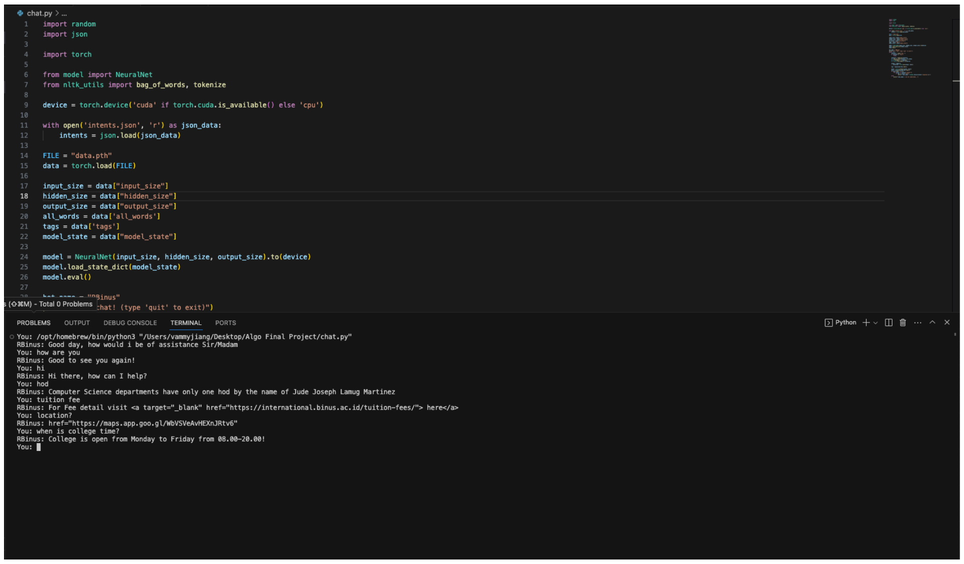


train.py > ChatClassifier > process_intents
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Ln 51, Col 55   Spaces: 4   UTF-8   LF   Python   3.11.6 64-bit

```python
from torch.utils.data import Dataset, DataLoader
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from nltk_utils import tokenize, stem
from model import NeuralNet

# Define a class for the Chat Classifier
class ChatClassifier:
    def __init__(self, intents_file='intents.json', model_file='data.pth'):
        # Initialize the class with default file names and placeholders for data
```

```
/opt/homebrew/bin/python3 "/Users/vammyjiang/Desktop/Algo Final Project/train.py"
vammyjiang@Vammy Algo Final Project % /opt/homebrew/bin/python3 "/Users/vammyjiang/Desktop/Algo Final Project/train.py"
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/vammyjiang/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
/opt/homebrew/lib/python3.11/site-packages/sklearn/feature_extraction/text.py:525: UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
Epoch [100/1000], Loss: 0.0700
Epoch [200/1000], Loss: 0.0180
Epoch [300/1000], Loss: 0.1438
Epoch [400/1000], Loss: 0.1715
Epoch [500/1000], Loss: 0.0000
Epoch [600/1000], Loss: 0.0000
Epoch [700/1000], Loss: 0.0000
Epoch [800/1000], Loss: 0.0000
Epoch [900/1000], Loss: 0.0000
Epoch [1000/1000], Loss: 0.0000
Epoch [100/2000], Loss: 0.0085
Epoch [200/2000], Loss: 0.0085
Epoch [300/2000], Loss: 0.0085
Epoch [400/2000], Loss: 0.0085
Epoch [500/2000], Loss: 0.0085
Epoch [600/2000], Loss: 0.0085
Epoch [700/2000], Loss: 0.0085
Epoch [800/2000], Loss: 0.0085
Epoch [900/2000], Loss: 0.0085
Epoch [1000/2000], Loss: 0.0085
Epoch [1100/2000], Loss: 0.0085
Epoch [1200/2000], Loss: 0.0085
Epoch [1300/2000], Loss: 0.0085
Epoch [1400/2000], Loss: 0.0085
Epoch [1500/2000], Loss: 0.0085
Epoch [1600/2000], Loss: 0.0085
Epoch [1700/2000], Loss: 0.0085
Epoch [1800/2000], Loss: 0.0085
Epoch [1900/2000], Loss: 0.0085
Epoch [2000/2000], Loss: 0.0085
vammyjiang@Vammy Algo Final Project %
```

Then, we run chat.py and asks it questions :



```python
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()
```

```
You: /opt/homebrew/bin/python3 "/Users/vammyjiang/Desktop/Algo Final Project/chat.py"
RBinus: Good day, how would i be of assistance Sir/Madam
You: how are you
RBinus: Good to see you again!
You: hi
RBinus: Hi there, how can I help?
You: hod
RBinus: Computer Science departments have only one hod by the name of Jude Joseph Lamug Martinez
You: tuition fee
RBinus: For Fee detail visit <a target="_blank" href="https://international.binus.ac.id/tuition-fees/"> here</a>
You: location?
RBinus: href="https://maps.app.goo.gl/WbVSVeAvHEXnJRtv6"
You: when is college time?
RBinus: College is open from Monday to Friday from 08.00-20.00!
You: 
```

Chapter 5 - Lesson Learned

In working on this project, I have learned a lot of new things in python :

- Several new libraries (numpy, nltk, PyTorch) as well as how to utilize them to make a very interesting programming.
- Natural language processing was something previously unknown to myself, so in this project I have learned what it is, how does it work, and the processes behind NLP in which in the current world has useful tools in the form of chatgpt and other AI chatbot which utilizes NLP to produce a fluid more human like response.

Overall, I have feel that I have grown as a programmer once again. Moreover, after this project I personally feel more interested to explore new area of coding in which I previously have not been interested to explore. I find the urge to keep improving what I made and hope to make more other projects and interesting tools to help me boost my own programming skill on my programming journey that has just begun a few months ago.

Chapter 6 - Source Code and Video Demo

GitHub Link : https://github.com/TanaRuin/Algo-Prog-Final-Project/tree/main

Video Demo Link :
https://drive.google.com/drive/folders/1JEFBTFTJYUD24jLqv7CkMdZZ1XMEJg0m?usp=sharing

References

patrickloeber/pytorch-chatbot: Simple chatbot implementation with PyTorch. (n.d.). GitHub.

https://github.com/patrickloeber/pytorch-chatbot

codebasics. End-to-End NLP Project | Build a Chatbot in Dialogflow | NLP Tutorial | S3 E2. *YouTube*. Published online June 23, 2023. Accessed January 7, 2024. https://www.youtube.com/watch?v=2e5pQqBvGco&t=9916s

sentdex. Creating a Chatbot with Deep Learning, Python, and TensorFlow p.1. *YouTube*. Published online November 24, 2017. Accessed January 7, 2024. https://www.youtube.com/watch?v=dvOnYLDg8_Y&list=PLQVvvaa0QuDdc2k5dwtDTyT9aCja0on8j

Landbot. Natural Language Processing Chatbot | Quick Overview. *YouTube*. Published online July 13, 2022. Accessed January 7, 2024. https://www.youtube.com/watch?v=XT923_4GY9k