ARTIFICIAL INTELLIGENCE FINAL PROJECT

SoulSphere - Deep Learning Based Therapist Chatbot

Avariq Fazlur Rahman - 2502043002

Bayu Hartho - 2502013731

Bernard Choa - 2502022414

Sandrian Wardana - 2502016411

L5BC

Problem Description

Sharing about our wellbeing and emotions is crucial for personal and social reasons. On a personal level, expressing emotions contributes to positive mental health by providing a therapeutic outlet for processing feelings and developing coping strategies. It fosters self-awareness and reflection, promoting personal growth. Socially, sharing emotions strengthens relationships by creating empathy and understanding, building trust, and forming a supportive network. It also helps reduce the stigma surrounding mental health issues, contributing to a more compassionate and open society. Additionally, discussing emotions enables collaborative problem-solving, offering diverse perspectives and insights, and enhances emotional intelligence, a valuable skill for navigating social interactions.

In essence, open communication about wellbeing and emotions is a cornerstone of mental health, social connection, and personal development. It not only benefits individuals by providing emotional release and self-reflection but also contributes to a more empathetic and supportive community that values and understands the importance of mental well-being.

Our project will be called SoulSphere. It is centered around giving people that are less likely to talk about their feelings openly with someone else. With our project, we are providing those people with a platform that they can express their feelings, be it frustrations, happiness, sadness, and other feelings. We are going to be using deep learning technologies from Tensorflow's Keras and natural language processing to achieve a chatbot that is as genuine and as natural as it could possibly be with Artificial Intelligence. We will be utilizing packages such as NLTK, Tensorflow, and Scikit learn.

Solution Features

Intent Recognition

The chatbot utilizes a neural network for intent recognition, comprising the following components:

- **Embedding Layer** : Converts words into dense vectors, capturing semantic relationships.

- **Global Average Pooling Layer** : Aggregates information from embeddings to create a fixed-size representation.
- **Dense Layers** : Multiple dense layers for classification, employing the softmax activation function to predict the most likely intent.

Response Generation

The chatbot generates responses based on recognized intents. Key features include:

- **Response Variation** : Responses are randomly selected from a set of predefined responses associated with each intent, ensuring diversity in conversations.
- **Dynamic Response Handling** : If the intent is not recognized, a default response is provided, enhancing user experience.

Model Persistence

The trained model, tokenizer, and label encoder are saved for seamless integration and reusability:

- **Model Saving** : The entire model is saved for deployment in other environments.
- **Tokenizer and Label Encoder** : Serialized versions are stored, allowing consistent preprocessing of user input during deployment.

Solution Design Architecture

Model Architecture

The neural network architecture is designed for effective intent recognition:

- **Embedding Layer** : Input layer to capture word semantics.
- **Global Average Pooling Layer** : Captures the global context from embeddings.
- **Dense Layers (ReLU Activation)** : Neural network layers for classification.
- **Softmax Activation** : Final layer for probability distribution over intents.

Tokenization and Padding

Text preprocessing is crucial for model input:

- **Tokenizer** : Converts text into sequences of integers, establishing a relationship between words and numerical values.

- **Padding** : Ensures uniform sequence length for training and inference.

Experiments / Tests

Training

The model training phase involved the following hyperparameters and configurations:

- **Dataset** : The training data was sourced from the intents.json file, consisting of user patterns, associated intents, and responses.
- **Tokenization and Padding** : The Tokenizer class was used with a vocabulary size of 35,000 words and an out-of-vocabulary token ("<OOV>"). Sequences were padded to a maximum length of 20.
- **Embedding Dimension** : Word embeddings were set to 300 dimensions to capture rich semantic information.
- **Model Architecture** : The Sequential model comprised an embedding layer, a global average pooling layer, and two dense layers (300 units each) with ReLU activation functions. The final dense layer had a softmax activation function with the number of units equal to the total number of unique intents.
- **Optimization** : The Adam optimizer was employed, and the model was trained for 200 epochs.

Evaluation

The model's performance was evaluated on a separate validation set using the following metrics:

- **Accuracy** : The accuracy metric was used to assess the model's overall performance in correctly classifying user intents.
- **Loss Function** : The sparse categorical cross entropy loss function was monitored during training and evaluation to understand the convergence of the model.

The training and evaluation process aimed to ensure the model's ability to generalize unseen data and handle various user inputs effectively. Adjustments to hyperparameters were made iteratively based on validation results to enhance overall performance.

This detailed evaluation process helps to understand the model's strengths, weaknesses, and areas for improvement, providing a comprehensive analysis of the chatbot's performance.

Program Manual

The program is made out of multiple cells

```
!pip install tensorflow
!pip install nltk
!pip install colorama
!pip install numpy
!pip install scikit_learn
!pip install Flask
!pip install matplotlib
```

```
import json
import tensorflow as tf
from tensorflow import keras
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

The first and second cell are the required modules imported inorder to use the code. As seen here, we will be using tensorflow to model the chatbot's language model, specifically using Keras to train the model. Aside from tensorflow, there are also sklearn for label encoders, numpy for mathematical operations, and matplotlib to plot one of our key findings later on in the project.

```
with open('C:/Users/avari/OneDrive/Documents/AI Final Project/intents.json') as file:
    data = json.load(file)

training_sentences = []
training_labels = []
labels = []
responses = []


for intent in data['intents']:
    for pattern in intent['patterns']:
        training_sentences.append(pattern)
        training_labels.append(intent['tag'])
    responses.append(intent['responses'])

    if intent['tag'] not in labels:
        labels.append(intent['tag'])

num_classes = len(labels)
```

This cell will iterate through 'intent' in the loaded JSON data and then through each 'pattern' within that intent, it then appends the response associated with each intent to the

‘responses’ list. It then checks if the intent tag is not already in the ‘labels’ list, and adds it if it doesn’t. This is done to collect unique intent labels.

```
lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)
```

This cell encodes categorical labels into numerical format, it then analyzes the unique labels and assigns a unique numerical value to each unique label.

```
vocab_size = 45000
embedding_dim = 350
max_len = 20
oov_token = "<OOV>"

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_len)
```

These hyperparameters determines the maximum number of words to keep, based on word frequency with *‘vocab_size’*, the dimension of the word embeddings with *‘embedding_dim’*, The maximum length of sequences after padding with *‘max_ len’*, and the special token used to represent out-of-vocabulary words with *‘oov_token’*.

It then prepares text data for use in a neural network model by tokenizing the text, converting words to numerical indices, and padding sequences to a fixed length.

```
| 💡 Click here to ask Blackbox to help you code faster
# Split the data into training and testing sets (70/30 split)
X_train, X_test, y_train, y_test = train_test_split(
    padded_sequences, np.array(training_labels), test_size=0.3, random_state=42
)
```

This cell is for data splitting. This is essential for creating the model’s training and test sets that will be used to calculate the metrics of the model.

```
# Model Testing
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])

model.summary()
```

This cell defines a neural network model for text classification. It uses an embedding layer to convert words into dense vectors, a global average pooling layer to reduce dimensionality, two dense layers for learning patterns, and a softmax output layer for multiclass classification.

```
# Define EarlyStopping callback to monitor both training loss and accuracy
early_stopping = keras.callbacks.EarlyStopping(
    monitor='loss',  # Monitor training loss
    patience=10,  # Wait for 10 epochs with no improvement
    restore_best_weights=True
)
```

This cell is to avoid overfitting and to stop training when there is no significant improvement in the monitored quantity (training loss in this case) over a specified number of epochs.

```
# Epochs
epochs = 200
history = model.fit(
    padded_sequences, np.array(training_labels),
    epochs=epochs,
    callbacks=[early_stopping]
)
```

This cell is a training process that involves adjusting the model's weights based on the provided input data and labels, using the specified optimizer and loss function with 1 epoch representing 1 model training session. The result will then look like the image below:

```
Epoch 8/200
5/5 [==============================] - 0s 80ms/step - loss: 3.2775 - accuracy: 0.0667
Epoch 9/200
5/5 [==============================] - 0s 52ms/step - loss: 3.2654 - accuracy: 0.0667
Epoch 10/200
5/5 [==============================] - 0s 53ms/step - loss: 3.2484 - accuracy: 0.0667
Epoch 11/200
5/5 [==============================] - 0s 61ms/step - loss: 3.2263 - accuracy: 0.0667
Epoch 12/200
5/5 [==============================] - 0s 81ms/step - loss: 3.1989 - accuracy: 0.0667
Epoch 13/200
...
Epoch 138/200
5/5 [==============================] - 0s 46ms/step - loss: 0.0380 - accuracy: 0.9800
Epoch 139/200
5/5 [==============================] - 0s 53ms/step - loss: 0.0398 - accuracy: 0.9800
```

With the help of the code below we'll be able to determine the average loss and average accuracy:

```python
# Print average model loss and accuracy
average_loss = np.mean(history.history['loss'])
average_accuracy = np.mean(history.history['accuracy'])
print(f'Average Loss: {average_loss:.4f}')
print(f'Average Accuracy: {average_accuracy:.4f}')
```

```
Average Loss: 0.6826
Average Accuracy: 0.8166
```

```
| 💡 Click here to ask Blackbox to help you code faster
# Evaluate on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
2/2 [==============================] - 0s 36ms/step - loss: 0.0551 - accuracy: 0.9565
```

This cell is specifically for model testing. This cell will tell the user on the metrics of the model testing as it will be intrinsic to test how the model deals with unknown data.

```
# Plot model loss and accuracy in one graph
plt.figure(figsize=(12, 6))

# Plot Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy', color='orange')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Combine both subplots into a single graph
plt.tight_layout()

plt.show()
```

This cell plots model loss and model accuracy. This plot will better help visualize to the users of the improvements of the model after each training.

```
# to save the trained model
model.save("C:/Users/avari/OneDrive/Documents/AI Final Project/Test/chat_model")

import pickle

# to save the fitted tokenizer
with open('C:/Users/avari/OneDrive/Documents/AI Final Project/Test/tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

# to save the fitted label encoder
with open('C:/Users/avari/OneDrive/Documents/AI Final Project/Test/label_encoder.pickle', 'wb') as ecn_file:
    pickle.dump(lbl_encoder, ecn_file, protocol=pickle.HIGHEST_PROTOCOL)
```

This cell is responsible for saving the trained neural network model, the fitted tokenizer, and the fitted label encoder to disk. These will be used as the model that powers the chatbot and will be the model that the user will chat with.

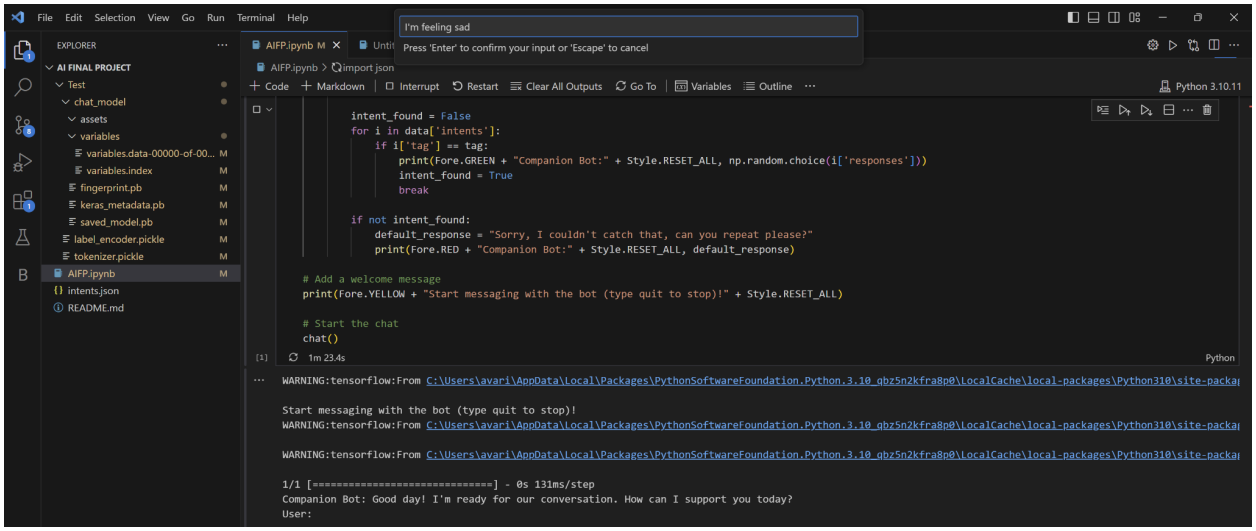
```
import json
import numpy as np
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder

import colorama
colorama.init()
from colorama import Fore, Style, Back

import random
import pickle

with open("C:/Users/avari/OneDrive/Documents/AI Final Project/intents.json") as file:
    data = json.load(file)


def chat():
    # load trained model
    model = keras.models.load_model('C:/Users/avari/OneDrive/Documents/AI Final Project/Test/chat_model')

    # load tokenizer object
    with open('C:/Users/avari/OneDrive/Documents/AI Final Project/Test/tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

    # load label encoder object
    with open('C:/Users/avari/OneDrive/Documents/AI Final Project/Test/label_encoder.pickle', 'rb') as enc:
        lbl_encoder = pickle.load(enc)

    # parameters
    max_len = 20
```

```
    while True:
        print(Fore.LIGHTBLUE_EX + "User: " + Style.RESET_ALL, end="")
        inp = input()
        if inp.lower() == "quit":
            break

        result = model.predict(keras.preprocessing.sequence.pad_sequences(tokenizer.texts_to_sequences([inp]),
                                                                          truncating='post', maxlen=max_len))
        tag = lbl_encoder.inverse_transform([np.argmax(result)])[0]

        intent_found = False
        for i in data['intents']:
            if i['tag'] == tag:
                print(Fore.GREEN + "Companion Bot:" + Style.RESET_ALL, np.random.choice(i['responses']))
                intent_found = True
                break

        if not intent_found:
            default_response = "Sorry, I couldn't catch that, can you repeat please?"
            print(Fore.RED + "Companion Bot:" + Style.RESET_ALL, default_response)

# Add a welcome message
print(Fore.YELLOW + "Start messaging with the bot (type quit to stop)!" + Style.RESET_ALL)

# Start the chat
chat()
```

This cell lets the user interact and chat with the bot. Depending on the environment used, the way users chat with the chatbot may vary. In some environments like Jupyter Notebook and

it's other IDEs such as Google Colab, users will be able to chat directly in the output of the cell. In IDEs such as Visual Studio Code, however, the user will get input bar, but the chatbot's answer will be in the output of the cell.

```
Start messaging with the bot (type quit to stop)!
User: Hi
1/1 [==============================] - 0s 65ms/step
Companion Bot: Good day! I'm ready for our conversation. How can I support you today?
User: I'm feeling down
1/1 [==============================] - 0s 14ms/step
Companion Bot: I'm here for you. If you're comfortable, share more about what you're going through.
User: I need someone to understand my feelings
1/1 [==============================] - 0s 14ms/step
Companion Bot: While I may not fully understand your experience, I'm here to listen and support you as best as I can.
User: I'm having issues in my relationship
1/1 [==============================] - 0s 14ms/step
Companion Bot: I understand that relationships can be challenging. Let's talk about what you're going through, and we can explore ways to improve the situation.
User: My partner and I are struggling lately
1/1 [==============================] - 0s 16ms/step
Companion Bot: I understand that relationships can be challenging. Let's talk about what you're going through, and we can explore ways to improve the situation.
User: Thank you for listening to me
1/1 [==============================] - 0s 14ms/step
Companion Bot: Your appreciation means a lot. It's my pleasure to be here for you.
User: Bye
1/1 [==============================] - 0s 14ms/step
Companion Bot: Until next time! If you ever need support, don't hesitate to return. Take care!
User:
```



As seen here, the chatbot accepts user input, being the questions or formulation that the user gives. It will give an appropriate answer depending on the model that we have trained beforehand. Users can keep going as much as they want so long as their input is something that is within the intents file so that the model can backtrack to it. After the user feels like they have talked with the chatbot enough, they can input 'quit' to stop the bot from running.

Demo Video

The demo video can be found in the link below :

https://drive.google.com/drive/folders/1LifYFFt5ZYGJrqXSgLzMVjZGYWmQyp-A?usp=sharing

Documentation

The documentation to this code can be found in this GitHub link below :

https://github.com/avariqfr30/SoulSphere-FinalProject.git