**File train\_chatbot.py**

Khởi tạo và nạp các thư viện cần thiết

*import nltk*

*from nltk.stem import WordNetLemmatizer*

*lemmatizer = WordNetLemmatizer()*

*import json*

*import pickle*

*import numpy as np*

*import random*

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Dense, Dropout*

*from tensorflow.keras.optimizers import SGD*

- **nltk:** Thư viện này được sử dụng để xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP). Nó cung cấp các công cụ như tokenize và lemmatize để tiền xử lý văn bản.

- **WordNetLemmatizer:** Một công cụ của nltk dùng để lemmatize, tức là chuyển các từ về dạng gốc của chúng.

- **json**: Để xử lý các tệp JSON, thư viện này cho phép đọc và ghi dữ liệu dạng JSON.

- **pickle**: Dùng để tuần tự hóa và giải tuần tự hóa các đối tượng Python, giúp lưu trữ và tái sử dụng các đối tượng sau khi chương trình đã kết thúc.

- **numpy:** Thư viện này cung cấp các công cụ tính toán số học với mảng đa chiều (arrays).

- **random:** Thư viện này cung cấp các công cụ để tạo ra các số ngẫu nhiên hoặc thực hiện các thao tác ngẫu nhiên.

- **tensorflow.keras.models.Sequential**: Mô hình Sequential của Keras, dùng để xây dựng mô hình mạng nơron theo dạng tuần tự (mỗi lớp nối tiếp nhau).

- **tensorflow.keras.layers.Dense:** Lớp mạng nơron đầy đủ kết nối, mỗi nơron trong lớp này kết nối với tất cả các nơron trong lớp trước đó.

- **tensorflow.keras.layers.Dropout:** Lớp Dropout để giảm overfitting bằng cách ngẫu nhiên bỏ qua một số nơron trong quá trình huấn luyện.

- **tensorflow.keras.optimizers.SGD**: Hàm tối ưu Stochastic Gradient Descent, sử dụng trong quá trình huấn luyện mô hình để tối ưu hóa trọng số.

**Tiền xử lý dữ liệu**

*words = []*

*classes = []*

*documents = []*

*ignore\_words = ['?', '!']*

*data\_file = open('intents.json').read()*

*intents = json.loads(data\_file)*

- **words, classes, documents**: Các danh sách này sẽ lưu trữ các từ, các lớp (tags), và các cặp câu - tag tương ứng.

- **ignore\_words**: Danh sách các từ hoặc ký tự không cần thiết sẽ bị loại bỏ trong quá trình tiền xử lý.

- **data\_file**: Mở tệp intents.json và đọc dữ liệu từ tệp này.

- **intents:** Chuyển dữ liệu từ định dạng JSON thành định dạng Python để dễ dàng xử lý.

**Lấy mẫu dữ liệu từ JSON và tiền xử lý**

*for intent in intents['intents']:*

*for pattern in intent['patterns']:*

*w = nltk.word\_tokenize(pattern)*

*words.extend(w)*

*documents.append((w, intent['tag']))*

*if intent['tag'] not in classes:*

*classes.append(intent['tag'])*

*words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]*

*words = sorted(list(set(words)))*

*classes = sorted(list(set(classes)))*

- **Vòng lặp for**: Duyệt qua tất cả các patterns (câu mẫu) trong intents.json.

- **word\_tokenize**: Chia nhỏ các câu mẫu thành các từ riêng lẻ.

- **documents:** Lưu trữ mỗi câu mẫu cùng với nhãn (tag) tương ứng.

- **classes**: Thêm các nhãn vào danh sách classes nếu nó chưa có.

- **lemmatize**: Chuẩn hóa các từ về dạng gốc, đồng thời chuyển tất cả các từ về chữ thường và loại bỏ các từ trong danh sách ignore\_words.

- **set(words)**: Loại bỏ các từ trùng lặp.

- **sorted**: Sắp xếp các từ và các nhãn theo thứ tự chữ cái.

**Tạo tập huấn luyện**

*training = []*

*output\_empty = [0] len(classes)*

*for doc in documents:*

*bag = []*

*pattern\_words = doc[0]*

*pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]*

*for w in words:*

*bag.append(1) if w in pattern\_words else bag.append(0)*

*output\_row = list(output\_empty)*

*output\_row[classes.index(doc[1])] = 1*

*training.append([bag, output\_row])*

*random.shuffle(training)*

*training = np.array(training)*

*train\_x = list(training[:, 0])*

*train\_y = list(training[:, 1])*

- **training:** Danh sách để lưu trữ tập huấn luyện.

- **output\_empty:** Vectơ nhị phân ban đầu, với các giá trị bằng 0, có kích thước bằng số lượng nhãn (tags).

- **Bag of Words**: Tạo vectơ BoW, mỗi phần tử của vectơ tương ứng với một từ trong từ điển, giá trị của nó là 1 nếu từ đó có trong câu, và 0 nếu không có.

- **output\_row:** Vectơ đầu ra, là một vectơ nhị phân với giá trị 1 tại vị trí của lớp tương ứng và 0 tại các vị trí khác.

- **random.shuffle:** Trộn ngẫu nhiên các cặp đầu vào - đầu ra trong tập huấn luyện.

**- training = np.array(training):** Chuyển đổi danh sách thành mảng numpy để dễ dàng xử lý sau này.

**Xây dựng mô hình và huấn luyện**

*model = Sequential()*

*model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(64, activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(len(train\_y[0]), activation='softmax'))*

*model.compile(loss='categorical\_crossentropy', optimizer=SGD(learning\_rate=0.01, momentum=0.9, nesterov=True), metrics=['accuracy'])*

*model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)*

- **Sequential:** Mô hình Sequential cho phép xây dựng mạng nơron theo thứ tự từng lớp.

- **Dense(128, input\_shape=...):** Thêm một lớp mạng nơron đầy đủ kết nối với 128 nơron, đầu vào có kích thước bằng chiều dài của vectơ BoW.

- **Dropout(0.5):** Thêm lớp Dropout để ngẫu nhiên bỏ qua 50% nơron trong quá trình huấn luyện, nhằm giảm overfitting.

- **Dense(64):** Thêm một lớp ẩn với 64 nơron.

- **Dense(len(train\_y[0])):** Lớp đầu ra với số nơron bằng số lượng nhãn, sử dụng hàm kích hoạt softmax.

- **categorical\_crossentropy:** Hàm mất mát dùng cho bài toán phân loại đa lớp.

- **SGD(learning\_rate=0.01, momentum=0.9, nesterov=True):** Hàm tối ưu SGD với gia tốc Nesterov, học với tốc độ 0.01 và động lượng 0.9.

- **model.fit:** Huấn luyện mô hình với tập huấn luyện train\_x và train\_y, với 200 epochs và batch size là 5.

**File chatgui.py**

**Khởi tạo và nạp mô hình**

*import tkinter as tk*

*from tkinter import*

*from keras.models import load\_model*

*import json*

*import numpy as np*

*import pickle*

*import random*

*model = load\_model('chatbot\_model.h5')*

*intents = json.loads(open('intents.json').read())*

*words = pickle.load(open('words.pkl', 'rb'))*

*classes = pickle.load(open('classes.pkl', 'rb'))*

**- tkinter:** Thư viện để tạo giao diện người dùng đồ họa (GUI) trong Python.

**- load\_model:** Hàm này dùng để nạp mô hình đã được huấn luyện và lưu trữ trước đó (chatbot\_model.h5).

**- pickle.load:** Đọc và nạp lại các đối tượng Python đã được lưu trữ dưới dạng nhị phân, ở đây là words.pkl và classes.pkl.

**Tiền xử lý câu hỏi của người dùng**

*def clean\_up\_sentence(sentence):*

*sentence\_words = nltk.word\_tokenize(sentence)*

*sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]*

*return sentence\_words*

*def bow(sentence, words):*

*sentence\_words = clean\_up\_sentence(sentence)*

*bag = [0]len(words)*

*for s in sentence\_words:*

*for i, w in enumerate(words):*

*if w == s:*

*bag[i] = 1*

*return(np.array(bag))*

**- clean\_up\_sentence:** Hàm này xử lý câu của người dùng, bao gồm chia nhỏ thành từ và lemmatize các từ.

**- bow:** Hàm này chuyển câu của người dùng thành một vectơ BoW để đưa vào mô hình.

**Dự đoán và xử lý kết quả**

*def predict\_class(sentence, model):*

*p = bow(sentence, words)*

*res = model.predict(np.array([p]))[0]*

*ERROR\_THRESHOLD = 0.25*

*results = [[i, r] for i, r in enumerate(res) if r > ERROR\_THRESHOLD]*

*results.sort(key=lambda x: x[1], reverse=True)*

*return\_list = []*

*for r in results:*

*return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})*

*return return\_list*

*def get\_response(ints, intents\_json):*

*tag = ints[0]['intent']*

*list\_of\_intents = intents\_json['intents']*

*for i in list\_of\_intents:*

*if i['tag'] == tag:*

*result = i['responses']*

*break*

*return random.choice(result)*

**- predict\_class:** Hàm này dự đoán lớp (tag) của câu người dùng, chỉ giữ lại những dự đoán có xác suất cao hơn ERROR\_THRESHOLD (0.25).

**- get\_response:** Hàm này lấy phản hồi tương ứng với lớp đã được dự đoán từ intents.json.

**Giao diện người dùng**

*def send():*

*msg = EntryBox.get("1.0", 'end-1c').strip()*

*EntryBox.delete("0.0", END)*

*if msg != '':*

*ChatLog.config(state=NORMAL)*

*ChatLog.insert(END, "You: " + msg + '\n\n')*

*ints = predict\_class(msg, model)*

*res = get\_response(ints, intents)*

*ChatLog.insert(END, "Bot: " + res + '\n\n')*

*ChatLog.config(state=DISABLED)*

*ChatLog.yview(END)*

*base = tk.Tk()*

*base.title("Chatbot")*

*base.geometry("400x500")*

*base.resizable(width=FALSE, height=FALSE)*

*ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)*

*ChatLog.config(state=DISABLED)*

*scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")*

*ChatLog['yscrollcommand'] = scrollbar.set*

*SendButton = Button(base, font=("Verdana", 12, 'bold'), text="Send", width="12", height=5, bd=0, bg="32de97", activebackground="3c9d9b", fg='ffffff', command=send)*

*EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")*

*scrollbar.place(x=376,y=6, height=386)*

*ChatLog.place(x=6,y=6, height=386, width=370)*

*EntryBox.place(x=128, y=401, height=90, width=265)*

*SendButton.place(x=6, y=401, height=90)*

*base.mainloop()*

**- send:** Hàm này xử lý sự kiện gửi tin nhắn, dự đoán câu trả lời từ chatbot và hiển thị nó trong giao diện người dùng.

**- tk.Tk():** Khởi tạo cửa sổ giao diện người dùng.

**- ChatLog, EntryBox, SendButton:** Tạo và sắp xếp các thành phần của giao diện.

***Kết luận***

***- Tiền xử lý: Cả hai tệp train\_chatbot.py và chatgui.py đều sử dụng kỹ thuật lemmatization và tokenization để chuẩn bị dữ liệu văn bản.***

***- Huấn luyện mô hình: Trong train\_chatbot.py, mô hình mạng nơron với 2 lớp ẩn được huấn luyện sử dụng hàm mất mát categorical\_crossentropy và hàm tối ưu SGD với 200 epochs.***

***- Dự đoán: Tệp chatgui.py nạp mô hình đã huấn luyện, chuyển đổi câu của người dùng thành BoW và sử dụng mô hình để dự đoán phản hồi.***

Dựa trên mã nguồn trong tệp train\_chatbot.py các bước chính trong quá trình xây dựng và huấn luyện mô hình chatbot.

**1. Mô hình mạng**

- Mô hình mạng sử dụng là một Mạng Nơron Nhân tạo **(Artificial Neural Network - ANN)** với cấu trúc tuần tự **(Sequential)** được xây dựng bằng **Keras và TensorFlow.**

- Mô hình gồm 3 lớp chính:

- Lớp đầu vào (Input Layer): Sử dụng hàm kích hoạt **ReLU với 128 nơron**.

- Lớp ẩn (Hidden Layer): Sử dụng hàm kích hoạt **ReLU với 64 nơron.**

- Lớp đầu ra (Output Layer): Sử dụng hàm kích hoạt Softmax với số nơron bằng với số lượng thẻ (tags) cần phân loại.

**2. Đầu vào mô hình**

- Đầu vào của mô hình là các câu hỏi mẫu (patterns) được token hóa thành các từ (words) từ tệp intents.json. Các từ này được chuyển đổi thành các vectơ **Bag of Words (BoW)**, trong đó mỗi vị trí trong vectơ tương ứng với một từ trong từ điển (vocabulary).

- Ví dụ: Nếu câu hỏi là "Hello", vectơ BoW sẽ chứa giá trị 1 tại vị trí của từ "hello" và giá trị 0 ở các vị trí khác.

**3. Cách tiền xử lý dữ liệu**

- **Tokenization**: Tách câu thành các từ riêng biệt.

- **Lemmatization**: Chuẩn hóa các từ về dạng gốc của chúng (ví dụ: từ "running" sẽ được chuẩn hóa thành "run").

- Loại bỏ từ không cần thiết: Các ký tự đặc biệt như dấu chấm hỏi, chấm than bị loại bỏ.

- **Bag of Words (BoW)**: Mỗi câu hỏi được chuyển thành một vectơ số bằng cách so sánh các từ trong câu với từ điển từ đã được xây dựng trước đó.

**4. Huấn luyện mô hình**

- Mô hình được huấn luyện với các tham số:

- Số lượng lớp ẩn: **2 lớp ẩn (128 nơron và 64 nơron).**

- Hàm mất mát: **categorical\_crossentropy** (phù hợp với bài toán phân loại).

- Hàm tối ưu: **Stochastic Gradient Descent (SGD)** với gia tốc **Nesterov**.

- Dropout: **Sử dụng kỹ thuật Dropout với tỷ lệ 0.5 để tránh hiện tượng overfitting.**

**5. Tham số epoch, batch**

- **Epochs**: Số lần duyệt qua toàn bộ dữ liệu huấn luyện là **200** (tức là mô hình sẽ được huấn luyện qua toàn bộ dữ liệu 200 lần).

- **Batch size**: Kích thước batch là **5** (tức là mô hình sẽ cập nhật trọng số sau mỗi 5 mẫu dữ liệu).

**6. Độ chính xác mô hình**

- Độ chính xác chấp nhận được sẽ phụ thuộc vào mục tiêu của bạn, nhưng thông thường một độ chính xác từ 90% trở lên là tốt cho một mô hình chatbot cơ bản.

**7. Số lớp ẩn**

- **Mô hình có 2 lớp ẩn: lớp thứ nhất với 128 nơron và lớp thứ hai với 64 nơron.**

**8. Chạy bao nhiêu lần để có được tham số tối ưu**

- Mô hình được huấn luyện qua 200 epochs. Sau mỗi epoch, mô hình sẽ điều chỉnh các trọng số để dần dần tối ưu hóa.

Dưới đây là một phần mã nguồn liên quan đến việc tạo và huấn luyện mô hình:

*model = Sequential()*

*model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(64, activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(len(train\_y[0]), activation='softmax'))*

*model.compile(loss='categorical\_crossentropy', optimizer='sgd', metrics=['accuracy'])*

*model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)*

Cách thức hoạt động của mã nguồn trong các tệp ‘chatgui.py’ và ‘train\_chatbot.py’, tiền xử lý dữ liệu, cấu trúc mạng, và quá trình huấn luyện mô hình.

**1. Mã nguồn trong tệp ‘train\_chatbot.py’**

**a) Tiền xử lý dữ liệu**

Tiền xử lý dữ liệu là một bước quan trọng để chuẩn bị đầu vào cho mô hình. Trong ‘train\_chatbot.py’, các bước tiền xử lý dữ liệu được thực hiện qua các đoạn mã sau:

*import nltk*

*from nltk.stem import WordNetLemmatizer*

*lemmatizer = WordNetLemmatizer()*

*import json*

*import pickle*

*import numpy as np*

*import random*

*words = []*

*classes = []*

*documents = []*

*ignore\_words = ['?', '!']*

*data\_file = open('intents.json').read()*

*intents = json.loads(data\_file)*

*for intent in intents['intents']:*

*for pattern in intent['patterns']:*

*w = nltk.word\_tokenize(pattern)*

*words.extend(w)*

*documents.append((w, intent['tag']))*

*if intent['tag'] not in classes:*

*classes.append(intent['tag'])*

*words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore\_words]*

*words = sorted(list(set(words)))*

*classes = sorted(list(set(classes)))*

***Giải thích chi tiết:***

- Tokenization: Đoạn mã ‘**w = nltk.word\_tokenize(pattern)**’ dùng để chia nhỏ các câu mẫu (‘patterns’) thành các từ riêng lẻ (tokens).

- Lemmatization: Sử dụng **WordNetLemmatizer** để chuẩn hóa các từ về dạng gốc của chúng. Điều này giúp mô hình nhận biết được các từ có chung gốc nhưng khác hình thức (như "running" và "run").

- Loại bỏ từ không cần thiết: Các từ hoặc ký tự không có ý nghĩa (ví dụ: dấu chấm hỏi, chấm than) được loại bỏ khỏi danh sách các từ.

Sau đó, các từ và các nhãn (tags) được lưu trữ trong các danh sách ‘words’ và ‘classes’ để tạo ra tập từ vựng và các lớp đầu ra cho mô hình.

**b) Xây dựng tập huấn luyện**

Sau khi tiền xử lý dữ liệu, tập huấn luyện được tạo ra từ các từ và nhãn đã chuẩn hóa.

*training = []*

*output\_empty = [0] len(classes)*

*for doc in documents:*

*bag = []*

*pattern\_words = doc[0]*

*pattern\_words = [lemmatizer.lemmatize(word.lower()) for word in pattern\_words]*

*for w in words:*

*bag.append(1) if w in pattern\_words else bag.append(0)*

*output\_row = list(output\_empty)*

*output\_row[classes.index(doc[1])] = 1*

*training.append([bag, output\_row])*

*random.shuffle(training)*

*training = np.array(training)*

*train\_x = list(training[:, 0])*

*train\_y = list(training[:, 1])*

**Giải thích chi tiết:**

- Bag of Words: Mỗi câu hỏi mẫu được biểu diễn bằng một vectơ số (Bag of Words - BoW), trong đó mỗi phần tử của vectơ đại diện cho sự xuất hiện của một từ trong từ vựng.

- Đầu ra: Đầu ra được biểu diễn bằng một vectơ nhị phân (one-hot encoded) với độ dài bằng số lượng lớp. Ví dụ, nếu có 3 lớp (classes), đầu ra cho một mẫu có thể là ‘[1, 0, 0]’, nếu nó thuộc về lớp đầu tiên.

- Shuffle: Dữ liệu huấn luyện được trộn ngẫu nhiên để đảm bảo mô hình không bị học thiên vị theo thứ tự của dữ liệu.

**c) Xây dựng và huấn luyện mô hình**

Mô hình mạng nơron được xây dựng và huấn luyện thông qua các lớp nơron khác nhau.

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Dense, Dropout*

*from tensorflow.keras.optimizers import SGD*

*model = Sequential()*

*model.add(Dense(128, input\_shape=(len(train\_x[0]),), activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(64, activation='relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(len(train\_y[0]), activation='softmax'))*

*model.compile(loss='categorical\_crossentropy', optimizer=SGD(learning\_rate=0.01, momentum=0.9, nesterov=True), metrics=['accuracy'])*

*model.fit(np.array(train\_x), np.array(train\_y), epochs=200, batch\_size=5, verbose=1)*

**Giải thích chi tiết:**

- Số lượng lớp ẩn: Mô hình có 2 lớp ẩn với số nơron lần lượt là 128 và 64.

- Hàm kích hoạt: ‘ReLU’ (Rectified Linear Unit) được sử dụng trong các lớp ẩn để giới hạn đầu ra của nơron trong khoảng [0, +∞).

- Dropout: Kỹ thuật Dropout với tỷ lệ 0.5 được áp dụng để giảm overfitting, giúp mô hình học tốt hơn và tránh phụ thuộc quá nhiều vào một số nơron nhất định.

- Hàm mất mát: ‘**categorical\_crossentropy**’ được sử dụng cho bài toán phân loại đa lớp.

- Optimizer: Sử dụng **Stochastic Gradient Descent (SGD)** với gia tốc Nesterov, đây là một phương pháp giúp tăng tốc độ hội tụ của mô hình.

- Epochs: Mô hình được huấn luyện qua 200 epochs, tức là toàn bộ dữ liệu huấn luyện được duyệt qua 200 lần.

- Batch Size: Kích thước batch là 5, tức là mô hình sẽ cập nhật trọng số sau mỗi 5 mẫu dữ liệu.

**2. Mã nguồn trong tệp ‘chatgui.py’**

Tệp ‘chatgui.py’ thường là tệp giao diện người dùng, nơi người dùng có thể tương tác với chatbot sau khi nó đã được huấn luyện.

**a) Khởi tạo giao diện và nạp mô hình**

*import tkinter as tk*

*from tkinter import*

*from keras.models import load\_model*

*import json*

*import numpy as np*

*import pickle*

*import random*

*model = load\_model('chatbot\_model.h5')*

*intents = json.loads(open('intents.json').read())*

*words = pickle.load(open('words.pkl', 'rb'))*

*classes = pickle.load(open('classes.pkl', 'rb'))*

***Giải thích chi tiết:***

- Nạp mô hình: Mô hình đã được huấn luyện và lưu trữ trước đó (‘chatbot\_model.h5’) được nạp lại để sử dụng.

- Tải dữ liệu từ file: Các dữ liệu như từ vựng (‘words.pkl’) và các lớp (‘classes.pkl’) cũng được nạp từ các tệp đã lưu.

**b) Dự đoán đầu ra của chatbot**

*def clean\_up\_sentence(sentence):*

*sentence\_words = nltk.word\_tokenize(sentence)*

*sentence\_words = [lemmatizer.lemmatize(word.lower()) for word in sentence\_words]*

*return sentence\_words*

*def bow(sentence, words):*

*sentence\_words = clean\_up\_sentence(sentence)*

*bag = [0] len(words)*

*for s in sentence\_words:*

*for i, w in enumerate(words):*

*if w == s:*

*bag[i] = 1*

*return(np.array(bag))*

*def predict\_class(sentence, model):*

*p = bow(sentence, words)*

*res = model.predict(np.array([p]))[0]*

*ERROR\_THRESHOLD = 0.25*

*results = [[i, r] for i, r in enumerate(res) if r > ERROR\_THRESHOLD]*

*results.sort(key=lambda x: x[1], reverse=True)*

*return\_list = []*

*for r in results:*

*return\_list.append({"intent": classes[r[0]], "probability": str(r[1])})*

*return return\_list*

***Giải thích chi tiết:***

- clean\_up\_sentence: Làm sạch câu đầu vào bằng cách token hóa và lemmatize các từ.

- bow: Hàm này tạo một vectơ BoW từ câu đầu vào, biểu diễn câu dưới dạng một vectơ số dựa trên từ vựng của mô hình.

- predict\_class: Dự đoán lớp (intent) của câu đầu vào dựa trên vectơ BoW. Chỉ các lớp có xác suất lớn hơn ‘ERROR\_THRESHOLD’ (0.25) mới được giữ lại.

**c) Giao diện người dùng**

*def send():*

*msg = EntryBox.get("1.0", 'end-1c').strip()*

*EntryBox.delete("0.0", END)*

*if msg != '':*

*ChatBox.config(state=NORMAL)*

*ChatBox.insert(END, "You: " + msg + '\n\n')*

*ChatBox.config(foreground=" 446665", font=("Verdana", 12 ))*

*ints = predict\_class(msg, model)*

*res = getResponse(ints, intents)*

*ChatBox.insert(END, "Bot: " + res + '\n\n')*

*ChatBox.config(state=DISABLED)*

*ChatBox.yview(END)*

***Giải thích chi tiết:***

- send: Hàm này xử lý sự kiện gửi tin nhắn từ người dùng. Nó nhận đầu vào từ người dùng, gọi các hàm dự đoán để lấy kết quả và hiển thị phản hồi của chatbot trong cửa sổ chat.

Kết luận

**Từ phân tích trên:**

**- Mô hình mạng có 2 lớp ẩn với 128 và 64 nơron, được huấn luyện qua 200 epochs với batch size là 5.**

**- Hàm mất mát là ‘categorical\_crossentropy’, và hàm tối ưu là SGD với gia tốc Nesterov.**

**- Độ chính xác mô hình sẽ phụ thuộc vào quá trình huấn luyện và chất lượng dữ liệu, nhưng một mô hình với độ chính xác từ 90% trở lên là tốt cho các chatbot cơ bản.**

**- Số lần chạy (epochs) để đạt được tham số tối ưu đã được xác định là 200 lần trong mã nguồn.**