CODEBASE

intents.json

This file contains the different intents that our chatbot can recognize. Each intent is a specific type of user query or statement that the chatbot is designed to respond to.

- **Structure**:
  - **Tag**: A label that identifies the intent.
  - **Patterns**: Sample user inputs that represent the intent.
  - **Responses**: Possible responses the chatbot can give when it recognizes this intent.
  - **Context Set** : Used to manage conversation context.
- **Example**:

```json
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": ["Hi", "Hey", "How are you", "Is anyone there?", "Hello
      "responses": ["Hello! How can I assist you today?", "Hi! How can I
      "context_set": ["greeting"]
    },
    ...
  ]
}
```

In this example, the "greeting" intent includes various ways a user might greet the chatbot and the corresponding responses the chatbot can give.

Purpose

- **Training Data for Model**: The patterns in **intents.json** are used to train the neural network model, teaching it to recognize different types of user input.
- **Response Generation**: The responses provide a pool of replies that the chatbot can choose from when it identifies an intent.

Next, lets provide an overview of the **model.py** file, explaining its code structure and purpose in the context of our project.

The **model.py** file in our project defines the architecture of the neural network used by the chatbot. This network is responsible for understanding user inputs and determining the appropriate intent.

```
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
```

Key Components:

- **NeuralNet Class**: Inherits from **nn.Module**, PyTorch's base class for all neural network modules.
- **Layers**:
  - **l1**, **l2**, **l3**: Linear layers with varying input and output sizes.
  - **relu**: A ReLU activation function to introduce non-linearity.
- **Forward Method**: Defines the forward pass of the network, applying linear layers and activation functions in sequence.

- **Purpose**:
  - This neural network takes tokenized user inputs, processes them through its layers, and outputs a prediction of the user's intent.

- The network's architecture (two hidden layers) is simple yet effective for classification tasks like intent recognition.

Next, I'll go over the **train.py** file, explaining how our chatbot model is trained.

The **train.py** file is responsible for training the neural network model using the data defined in **intents.json**. It prepares the data, trains the model, and saves the trained model for later use.

- **Training Process Overview**:

```
import numpy as np
import random
import json
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet
```

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

# Data preparation
all_words = []
tags = []
xy = []
for intent in intents['intents']:
    tag = intent['tag']
    tags.append(tag)
    for pattern in intent['patterns']:
        w = tokenize(pattern)
        all_words.extend(w)
        xy.append((w, tag))

# Model training
# (Code for preparing training data and training the model)

# Save the trained model
torch.save(data, "data.pth")
```

- **Key Steps**:
    - **Data Preparation**: Tokenizes and stems the patterns from **intents.json**, creating a list of all words and a mapping of patterns to their corresponding tags.
    - **Model Training**:
        - Prepares training data by converting it into a format suitable for the neural network (not fully shown in the snippet).
        - Instantiates the **NeuralNet** model and trains it using this data.
    - **Model Saving**: The trained model is saved to a file (**data.pth**), which can be loaded later to make predictions.
- **Purpose**:
    - This script is central to the functioning of the chatbot, as it trains the model to understand and classify different intents based on user input.

Next, I'll describe the **nltk_utils.py** file, which contains utility functions for text processing.

The **nltk_utils.py** file contains utility functions for text processing, essential for preparing data for training the neural network model. These functions handle tokenization, stemming, and converting sentences into a bag-of-words format.

- **Utility Functions Overview**:

```python
import numpy as np
import nltk
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):
    # Function to create a bag of words
```

- **Key Functions**:
    - **tokenize**: Splits a sentence into an array of words or tokens.
    - **stem**: Reduces words to their root form using the Porter Stemming Algorithm.
    - **bag_of_words**: Creates a bag-of-words array from a tokenized sentence, comparing it with a given list of all words.
- **Purpose**:
    - These functions are crucial for preprocessing the text data before it's fed into the neural network. They help in standardizing the input and simplifying the language processing.

Lastly, I'll provide an overview of the **chat.py** file, which handles the interaction between the user and

The **chat.py** file is central to the functioning of our Bangla Chat Bot, as it manages the interaction between the user and the chatbot. It loads the trained model and uses it to generate responses based on user inputs.

- **Chat Functionality Overview**:

```
import random
import json
import torch
from model import NeuralNet
from nltk_utils import bag_of_words, tokenize
from langdetect import detect
from translate import Translator
```

```
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)

model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(data["model_state"])
model.eval()

bot_name = "ChatBot"

# Function to get the response
def get_response(msg):
    # Code to process the message and get a response
```

- **Key Components**:
  - **Model Loading**: Loads the trained neural network model.
  - **Response Generation**:
    - The function **get_response** takes user input, processes it (tokenization, bag of words), and feeds it to the model to determine the most appropriate response.
    - Utilizes language detection and translation for handling non-English inputs.
- **Purpose**:
  - This script is the bridge between the user and the chatbot, handling real-time processing of user input and generating conversational responses.

The **app.py** file creates a user-friendly GUI for the chatbot using Tkinter, a standard Python library for GUI development. This interface allows users to type in their queries and view the chatbot's responses.

- **GUI Setup and Functionality**:

```
from tkinter import *
from chat import get_response, bot_name

BG_COLOR = "#5e72e4"  # Blue
TEXT_COLOR = "#ffffff"  # White
FONT = "Helvetica 12"
FONT_BOLD = "Helvetica 12 bold"
```

```
class ChatApplication:

    def __init__(self):
        self.window = Tk()
        self._setup_main_window()

    def run(self):
        self.window.mainloop()

    def _setup_main_window(self):
        # Code to set up main window components like labels, text box,

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

- **Key Components**:
    - **Window Setup**: Initializes the main window and its properties (title, size, background color, etc.).
    - **Widgets**: Sets up various widgets like labels, text entry box, and text widget for displaying the chat history.
    - **Main Loop**: Starts the Tkinter event loop, which waits for user events (like button clicks or key presses).
- **Purpose**:
    - Provides a graphical platform for users to easily interact with the chatbot, enhancing user experience.
    - Integrates the chat functionality from **chat.py**, displaying conversations in a more accessible and interactive manner.

How Context is Understood and Managed

1. Tracking the Context: The global variable current_context holds the current context of the conversation.
2. Contextual Input Processing: When a new message (msg) is received, the chatbot checks if there's an existing context. If so, it modifies the input message to include this context. For example, if current_context is "weather," the message "And tomorrow?" is modified to "weather And tomorrow?" This helps the neural network understand the context of the follow-up question.
3. Generating Context-Aware Responses: The modified input is then processed (tokenized and converted to a bag of words), and the neural network model uses this processed input to generate a response. Since the model was trained on data that included contexts, it can generate more accurate and relevant responses.

```
# Context check and update
if current_context and not any(context in msg for context in curren
    msg = f"{current_context[0]} {msg}"
```