**Model**

**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)**

**# no activation and no softmax at the end**

**return out**

**NLTK Utils**

**import numpy as np**

**import random**

**import json**

**import torch**

**import torch.nn as nn**

**from torch.utils.data import Dataset, DataLoader**

**from nltk.stem.porter import PorterStemmer**

**import nltk**

**nltk.download('punkt')**

**# Define the tokenizer and stemmer**

**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):**

**sentence\_words = [stem(word) for word in tokenized\_sentence]**

**bag = np.zeros(len(words), dtype=np.float32)**

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

**if w in sentence\_words:**

**bag[idx] = 1**

**return bag**

**# Define the neural network model**

**class NeuralNet(nn.Module):**

**def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):**

**super(NeuralNet, self).\_\_init\_\_()**

**self.fc1 = nn.Linear(input\_size, hidden\_size)**

**self.fc2 = nn.Linear(hidden\_size, hidden\_size)**

**self.fc3 = nn.Linear(hidden\_size, output\_size)**

**self.relu = nn.ReLU()**

**def forward(self, x):**

**out = self.fc1(x)**

**out = self.relu(out)**

**out = self.fc2(out)**

**out = self.relu(out)**

**out = self.fc3(out)**

**return out**

**# Load intents data from a JSON file**

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

**intents = json.load(f)**

**all\_words = []**

**tags = []**

**xy = []**

**# Loop through each sentence in intents patterns**

**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))**

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

**all\_words = [stem(w) for w in all\_words if w not in ignore\_words]**

**all\_words = sorted(set(all\_words))**

**tags = sorted(set(tags))**

**X\_train = []**

**y\_train = []**

**for (pattern\_sentence, tag) in xy:**

**bag = bag\_of\_words(pattern\_sentence, all\_words)**

**X\_train.append(bag)**

**label = tags.index(tag)**

**y\_train.append(label)**

**X\_train = np.array(X\_train)**

**y\_train = np.array(y\_train)**

**num\_epochs = 1000**

**batch\_size = 8**

**learning\_rate = 0.001**

**input\_size = len(X\_train[0])**

**hidden\_size = 8**

**output\_size = len(tags)**

**class ChatDataset(Dataset):**

**def \_\_init\_\_(self):**

**self.n\_samples = len(X\_train)**

**self.x\_data = X\_train**

**self.y\_data = y\_train**

**def \_\_getitem\_\_(self, index):**

**return self.x\_data[index], self.y\_data[index]**

**def \_\_len\_\_(self):**

**return self.n\_samples**

**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)**

**criterion = nn.CrossEntropyLoss()**

**optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)**

**for epoch in range(num\_epochs):**

**for (words, labels) in train\_loader:**

**words = words.to(device)**

**labels = labels.to(dtype=torch.long).to(device)**

**outputs = model(words)**

**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}')**

**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}')**

**Chatting Code:**

**import random**

**import json**

**import to**

**rch**

**device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')**

**# Load intents data from a JSON file**

**with open('intents.json', 'r') as json\_data:**

**intents = json.load(json\_data)**

**# Load the pre-trained model and 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 = "Sam"**

**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"]:**

**print(f"{bot\_name}: {random.choice(intent['responses'])}")**

**else:**

**print(f"{bot\_name}: I do not understand...")**

* **Defines a neural network class NeuralNet using PyTorch's nn.Module.**
* **This network consists of three linear layers with ReLU activation function.**
* **It's designed for classification tasks with an input size, hidden size, and number of classes specified during initialization.**
* **The forward method defines the forward pass of the network.**
* **Imports necessary libraries such as NumPy, random, json, torch, and nltk.**
* **Defines a tokenizer function (tokenize), a stemming function (stem), and a function to create a bag of words representation (bag\_of\_words) from a given sentence.**
* **Defines a neural network class NeuralNet similar to the one in Model.py.**
* **Loads intents data from a JSON file named intents.json, which contains patterns and responses for various intents.**
* **Preprocesses the data by tokenizing sentences, stemming words, creating a bag of words representation, and preparing training data (X\_train and y\_train).**
* **Defines a PyTorch dataset (ChatDataset) and a data loader for training.**
* **Sets up device (CPU or GPU), initializes the neural network, defines loss function (CrossEntropyLoss), and optimizer (Adam).**
* **Trains the model for a specified number of epochs, printing the loss every 100 epochs.**
* **Saves the trained model's state dictionary along with necessary data to a file named data.pth.**
* **Imports necessary libraries such as random, json, and torch.**
* **Loads intents data from intents.json.**
* **Loads the pre-trained model and data from data.pth.**
* **Sets up the device (CPU or GPU) and initializes the neural network with the loaded parameters.**
* **Starts a loop where the user can input sentences to chat with the bot.**
* **Tokenizes the input sentence, converts it to a bag of words representation, and feeds it to the model to get predictions.**
* **If the predicted probability for the detected tag is above a certain threshold (0.75 in this case), it selects a random response from the corresponding intent in the intents data.**
* **Otherwise, it replies with a default message indicating that it doesn't understand.**

Bag of Words (BoW) Example:

Let's say we have a simple dataset with three sentences:

"I love machine learning"

"Machine learning is fascinating"

"I enjoy learning"

To represent these sentences using the bag-of-words approach, we first need to create a vocabulary, which consists of all unique words in the dataset. In this case, the vocabulary would be:

cssCopy code

['I', 'love', 'machine', 'learning', 'is', 'fascinating', 'enjoy']

Now, we represent each sentence as a vector indicating the presence or absence of each word in the vocabulary. For example:

Sentence 1: [1, 1, 1, 1, 0, 0, 0] (indicating presence of "I", "love", "machine", "learning")

Sentence 2: [0, 0, 1, 1, 1, 1, 0] (indicating presence of "machine", "learning", "is", "fascinating")

Sentence 3: [1, 0, 0, 1, 0, 0, 1] (indicating presence of "I", "learning", "enjoy")

NeuralNet Class Explanation:

The NeuralNet class defined in the code is designed to work with this bag-of-words representation to classify the intents of input sentences.

Initialization: During initialization, the class defines three linear layers (l1, l2, l3) and a ReLU activation function (relu). These layers are used to transform the input (bag-of-words vector) into logits for each class.

Forward Pass: The forward method takes a bag-of-words vector x as input and passes it through the defined linear layers with ReLU activation functions in between. Let's see how this works with an example using the bag-of-words vector for Sentence 1: [1, 1, 1, 1, 0, 0, 0].

plaintextCopy code

1. Input (x): [1, 1, 1, 1, 0, 0, 0]

2. l1 (Linear Layer):

- Weight Matrix (l1.weight): (hidden\_size, input\_size)

- Bias Vector (l1.bias): (hidden\_size,)

- Output of l1: relu(l1(x))

3. ReLU Activation Function:

- Output of ReLU: relu(l1(x))

4. l2 (Linear Layer):

- Weight Matrix (l2.weight): (hidden\_size, hidden\_size)

- Bias Vector (l2.bias): (hidden\_size,)

- Output of l2: relu(l2(relu(l1(x))))

5. ReLU Activation Function:

- Output of ReLU: relu(l2(relu(l1(x))))

6. l3 (Linear Layer):

- Weight Matrix (l3.weight): (num\_classes, hidden\_size)

- Bias Vector (l3.bias): (num\_classes,)

- Output of l3: l3(relu(l2(relu(l1(x)))))

7. Output:

- Final output (logits): l3(relu(l2(relu(l1(x)))))

Output: The final output (logits) is a vector of scores for each class/intent. These scores are then used to determine the most likely intent of the input sentence using techniques like softmax.

In summary, the NeuralNet class takes a bag-of-words vector as input, processes it through multiple linear layers with ReLU activation functions, and produces logits as output, which can be further processed to determine the most likely intent of the input sentence in the context of a chatbot.