train.py

This is a Python script for training a chatbot using PyTorch. The chatbot is trained to understand natural language input and provide appropriate responses.

The script starts by importing necessary libraries such as numpy, random, json, torch, and nltk_utils. Then, it reads a JSON file named 'intents.json' containing a list of intents, where each intent is associated with a set of patterns and responses.

Next, it preprocesses the text data by tokenizing, stemming, and creating a bag of words representation for each pattern sentence. It also creates a list of tags and a list of (pattern, tag) pairs. It then removes duplicates and sorts the all_words and tags lists.

After preprocessing, it creates training data by converting each pattern sentence into a bag of words and associating it with the corresponding tag. It also defines hyperparameters such as the number of epochs, batch size, learning rate, input size, hidden size, and output size.

Then, it creates a custom ChatDataset class that inherits from the PyTorch Dataset class. The ChatDataset class takes the training data and returns a tuple of input features and their corresponding labels when accessed using an index.

The script then checks if a GPU device with CUDA support is available and assigns it to the device variable. It then creates an instance of the NeuralNet class defined in the model.py file, passing the input size, hidden size, and output size as arguments. It also moves the model to the specified device.

After creating the model, it defines the loss function and optimizer used during training. The loss function used in this

script is the Cross-Entropy Loss. The optimizer used is the Adam optimizer.

The script then trains the model for the specified number of epochs. During training, it loads a batch of data from the DataLoader and moves the input and label tensors to the specified device. It then performs a forward pass, calculates the loss, performs backpropagation, and updates the model parameters using the optimizer.

Finally, it saves the trained model parameters, input size, hidden size, output size, all_words, and tags to a file named 'data.pth'.

At the end of the script, it prints a message indicating that the training is complete, and the model has been saved to the file.