Creating Assistant & Integrate With Flask Web Page

Date	03 October 2022
Team ID	PNT2022TMID08666
Project Name	Project – AI Based discourse for Banking Industry

```
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
with open('intents.json', 'r') as f:
    intents = json.load(f)
all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
all_words.extend(w)
        # add to xy pair
xy.append((w, tag))
# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

```
print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
X_train.append(bag)
    # y: PyTorchCrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
y_train.append(label)
X_train = np.array(X_train)
y_train = np.array(y_train)
# Hyper-parameters
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)
class ChatDataset(Dataset):
    def __init__(self):
self.n_samples = len(X_train)
self.x_data = X_train
self.y_data = y_train
    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]
    # we can call len(dataset) to return the size
    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)
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)
        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)
        # Backward and optimize
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}')
```