Contextual Conversation Application

Made by : Rekha kumari

Machine Learning Project -Chat Bot

This is a contextual conversation bot. This means that the model will be effective within a given context, for example, a ticket booking helper that interacts naturally with a customer and directs them to the appropriate links.

The dataset has a number of different `tags` which each correspond to a class of `patterns` and `responses`. When the model detects that an input is provided that resembles the `patterns` within a `tag`, it provides a randomized response from within the `responses` of the same `tag`.

Program Structure

1. Dataset

The dataset here is the `intents.json` file. It contains input patterns and corresponding output patterns, grouped under different tags. For example:

```
{
      "tag": "greeting",
      "patterns": [
        "Hi", "Hey", "How are you",
        "Is anyone there?", "Hello", "Good day"
      ],
      "responses": [
        "Hey :-)", "Hello, thanks for visiting",
        "Hi there, what can I do for you?", "Hi there, how can I help?"
      ]
}
```

2. Sentence Transformation

Each sentence goes through a series of transformations, which is shown in the following example:

- Original Sentence: `Hello, thanks for visiting!`
- **Tokenization** is the step where all separate words and punctuations in the sentence get separated into a list of words, punctuations. The previous sentence after tokenization becomes `['Hello', ',', 'thanks', 'for', 'visiting', '!']`
- **Stemming** is a process where the suffixes are removed from words, and the words are transformed to their root form. For example, `playing, played, player, plays` all get converted to `play`. The previous list after converting to lowercase and stemming becomes `['hello', ',', 'thank', 'for', 'visit', '!']`
- Punctuations are also removed `['hello', 'thank', 'for', 'visit']`
- Finally, the list of words is converted into a **bag of words**. This is a sort of one-hot encoding based on *all* the words in the dataset. This conversion is explained in more detail further down. This sentence converted to bag of words would be `[0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0]`

Bag of Words Algorithm

Consider the dictionary of words for the whole project:

```
all_words = ['hi', 'hello', 'welcome', 'for', 'good', 'i', 'you', 'bye', 'thank', 'cool', 'visit']
```

```
sentence = ['hello', 'thank', 'for', 'visit']
```

The previous stemmed sentence converted to bag of words will be:

```
[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1]
```

The words present in the sentence are marked as `1`, all other words are marked as `0`.

3. Create Training Data

We use the dataset, and transform the dataset as mentioned above to create our training data.

- **Tokenize** each sentence in the dataset, **Stem** each word and add all such words to the `all_words` list. The list should be in alphabetical order. `["'s", 'a', 'accept', 'anyon', 'are', 'bye',..., 'which', 'with', 'you']`
- For each input `patterns` in the dataset, **tokenize** and store them along with their corresponding `tags`. Each pattern converted to bag of words becomes the `x_train`. The corresponding index for each `tag` becomes the `y_train`.

```
x_train:
[[0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
```

```
y_train:
[3 3 3 3 3 3 2 2 2 6 6 6 6 4 4 4 5 5 5 5 0 0 0 1 1 1 1 1 1 1]
```

4. Create Model with Training Data

Here we use a Feed-Forward Neural Network, which is a type of Artificial Neural Network (ANN). For the algorithm, we use the `pytorch` library.

The Neural Network here uses an input layer, two hidden layers and an output layer. The input layer takes any sentence as an input, and the output layer returns the `tag` which has the highest probability of being a match with the input. It actually returns the probability of the input belonging to each `tag`.

For example, an input of `Hello there` would return the tag `greeting`.

After training the model, we store the trained model into a file `data.pth`, which is a `Pytorch` trained model. We use this `data.pth` to generate responses.

5. Using the Model

To use the model we use the following steps:

- Get an input sentence from the user, and convert it into bag of words.
- Feed the transformed sentence into the model, which returns a `tag`.
- Find the corresponding `response` to the `tag` from the dataset, and randomly select and return a `response`.

Dataset

```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": [
        "Hi",
        "Hey",
        "How are you",
        "Is anyone there?",
        "Hello",
        "Good day"
      ],
      "responses": [
        "Hey :-)",
        "Hello, thanks for visiting",
        "Hi there, what can I do for you?",
        "Hi there, how can I help?"
      ]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye"],
      "responses": [
        "See you later, thanks for visiting",
        "Have a nice day",
        "Bye! Come back again soon."
      ]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Thank's a lot!"],
      "responses": ["Happy to help!", "Any time!", "My pleasure"]
    },
    {
      "tag": "items",
      "patterns": [
        "Which items do you have?",
        "What kinds of items are there?",
        "What do you sell?"
      ],
      "responses": [
        "We sell coffee and tea",
        "We have coffee and tea"
      ]
    },
    {
      "tag": "payments",
      "patterns": [
        "Do you take credit cards?",
        "Do you accept Mastercard?",
        "Can I pay with Paypal?",
```

```
      "Are you cash only?",
      "How much is it?",
      "What payment methods do you have?",
      "How do I pay?"
    ],
    "responses": [
      "We accept VISA, Mastercard and Paypal",
      "We accept most major credit cards, and Paypal"
    ]
  },
  {
    "tag": "delivery",
    "patterns": [
      "How long does delivery take?",
      "How long does shipping take?",
      "When do I get my delivery?"
    ],
    "responses": [
      "Delivery takes 2-4 days",
      "Shipping takes 2-4 days"
    ]
  },
  {
    "tag": "funny",
    "patterns": [
      "Tell me a joke!",
      "Tell me something funny!",
      "Do you know a joke?"
    ],
    "responses": [
      "I once beat a human at chess. But I was no match for him at kickboxing."
    ]
  }
 ]
}
```

Files

nltk_util.py

```
import nltk
from nltk.stem.porter import PorterStemmer
import numpy as np

stemmer = PorterStemmer()

def tokenize(sentence):
    return np.array(nltk.word_tokenize(sentence))
def stem(word):
    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, all_words):
    for i in range(tokenized_sentence.shape[0]):
        tokenized_sentence[i] = stem(tokenized_sentence[i])

    bag = np.zeros(len(all_words), dtype = np.float32)

    for i in range(len(all_words)):
        if all_words[i] in tokenized_sentence:
            bag[i] = 1.0

    return bag
```

model.py

train.py

```
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):
        # Layer 1. Input into First hidden layer
        out = self.l1(x)
        out = self.relu(out)

        # Layer 2. Output from first hidden layer into second hidden layer.
        out = self.l2(out)
        out = self.relu(out)

        # Layer 3. Output from second hidden layer to output layer
        out = self.l3(out)
        # Outputs are further converted to cross-entropy loss in optimization loop

        return out
import json
from nltk_util import tokenize, stem, bag_of_words
import numpy as np

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from model import NeuralNet
```

```
                                    # Tags is a list. This line finds the index of `tag` in that

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

all_words = []
tags = []
xy = []


# Step 1: Create List of all Words
# Also stores each sentence in patterns along with corresponding tag in '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))

# List of punctuations to be ignored
ignore_words = ['?', '!', ',', '.', ';']


# Removes punctuations from all_words
temp = []
for word in all_words:
    if word not in ignore_words:
        temp.append(stem(word))
all_words = np.array(temp)

# Sorts tags and all_words in ascending order
all_words = sorted(np.unique(all_words))
tags = sorted(np.unique(tags))


# Create training data
x_train = []    # Stores the sentence vector (1-hot encoded, sort of)
y_train = []    # Stores the index of the corresponding tag.
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)
```

```
# Hyper Parameters
batch_size = 8
hidden_size = 8     # Number of nodes in hidden layer
input_size = len(x_train[0])    # Each input is an encoded sentence.
output_size = len(tags)

learning_rate = 0.001
num_epochs = 1000   # Maximum number of iterations for optimization


# This class is used to transform the training data into data that can be used as Pytorch NN
class ChatDataset(Dataset):
    def __init__(self, x_data, y_data):
        self.n_samples = len(x_train)
        self.x_data = x_data
        self.y_data = y_data

    # Allows us to access a dataset with an index
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.n_samples

    def __str__(self):
        string = f'{self.x_data}\n{self.y_data}'
        return string


# Takes training data and transforms it into NN input
dataset = ChatDataset(x_train, y_train)
train_loader = DataLoader(dataset = dataset, batch_size = batch_size,
                          shuffle = True, num_workers = 0)

# Checks if GPU available, else uses CPU
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()   # Loss Function
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)    # Optimization Functi
```

```
# Optimization Loop
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
        outputs = model(words)
        loss = criterion(outputs, labels)

        # Backward and Optimizer Step
        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}')
```

```
# Here we will store the trained model into a file,
# so that we won't have to train the model everytime we use the bot.
# Data that will be stored in this dictionary format
data = {
    'model_state': model.state_dict(),
    'input_size': input_size,
    'output_size': output_size,
    'hidden_size': hidden_size,
    'all_words': all_words,
    'tags': tags
}

FILE = 'data.pth'
torch.save(data, FILE)

print(f'Training complete. File saved to {FILE}')
```

chat.py

```
import random
import json
import torch

from model import NeuralNet
from nltk_util import bag_of_words, tokenize

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Import dataset
# The dataset is used here to match output tag with corresponding responses.
with open('intents.json', 'r') as f:
    intents = json.load(f)
```

```
# Import pre trained model
FILE = 'data.pth'
data = torch.load(FILE)

# Retrieve trained data from 'data.pth' 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 = 'Bot'
print("Let's chat! Type 'quit' to Quit")
while True:
    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)

    output = model(x)
    _, predicted = torch.max(output, dim = 1)
    tag = tags[predicted.item()]

    # Cross entropy loss is converted to probability using softmax function
    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']:
                response_choice = random.choice(intent['responses'])
                print(f'{bot_name}: {response_choice}')
```

```
        else:
            print(f'{bot_name}: I do not understand...')
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.