In [1]:
```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```
import numpy as np
import random
import json

import torch
import torch.nn as nn
import nltk
nltk.download('punkt')

from torch.utils.data import Dataset, DataLoader

import numpy as np
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Jananisha\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [3]:
```
# ANN Model

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
```

In [4]:
```
with open('intents.json', 'r') as f:
    intents = json.load(f)
```

In [5]:
```
def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)


def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())


def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bog   = [  0 ,    1 ,    0 ,    1 ,    0 ,    0 ,      0]
    """
    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
```

In [6]:
```
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))

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))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)
```

```
232 patterns
80 tags: ['about', 'afternoon', 'anxious', 'ask', 'casual', 'creation', 'death', 'default', 'depressed', 'done', 'evening', 'fact-1', 'fact-10', 'fact-11', 'fact-12', 'fact-13', 'fact-14', 'fact-15', 'fact-16', 'fact-17', 'fact-18', 'fact-19', 'fact-2', 'fact-20', 'fact-21', 'fact-22', 'fact-23', 'fact-24', 'fact-25', 'fact-26', 'fact-27', 'fact-28', 'fact-29', 'fact-3', 'fact-30', 'fact-31', 'fact-32', 'fact-5', 'fact-6', 'fact-7', 'fact-8', 'fact-9', 'friends', 'goodbye', 'greeting', 'happy', 'hate-me', 'hate-you', 'help', 'jokes', 'learn-mental-health', 'learn-more', 'location', 'meditation', 'mental-health-fact', 'morning', 'name', 'neutral-response', 'night', 'no-approach', 'no-response', 'not-talking', 'pandora-useful', 'problem', 'repeat', 'sad', 'scared', 'skill', 'sleep', 'something-else', 'stressed', 'stupid', 'suicide', 'thanks', 'understand', 'user-advice', 'user-agree', 'user-meditation', 'worthless', 'wrong']
280 unique stemmed words: ["'ll", "'m", "'re", "'s", "'ve", ',', 'a', 'about', 'absolut', 'advic', 'affect', 'afternoon', 'again', 'all', 'alot', 'already1', 'am', 'and', 'anxi', 'anxieti', 'answer', 'anxieti', 'anxiou', 'anymor', 'anyon', 'anyth', 'appear', 'approach', 'are', 'ask', 'au', 'avai1', 'aw', 'away', 'be', 'becaus', 'becom', 'befor', 'better', 'between', 'bonjour', 'boyfriend', 'break', 'bring', 'brother', 'burn', 'by', 'bye', 'ca', 'call', 'can', 'caus', 'cheer', 'child', 'commit', 'connect', 'continu', 'control', 'could', 'crazi', 'creat', 'cure', 'dad', 'day', 'defin', 'depress', 'deserv', 'did', 'die', 'differ', 'disord', 'do', 'doe', 'down', 'dumb', 'els', 'empti', 'enough', 'even', 'exam', 'fact', 'famili', 'fare', 'feel', 'few', 'financi', 'find', 'fine', 'focu', 'for', 'friend', 'from', 'get', 'girlfriend', 'give', 'go', 'good', 'goodby', 'great', 'group', 'guess', 'guten', 'had', 'hand', 'happi', 'hate', 'have', 'health', 'hello', 'help', 'hey', 'hi', 'hmmm', 'hola', 'how', 'howdi', 'i', 'if', 'ill', 'im', 'insomnia', 'insomnia', 'interest', 'involv', 'is', 'it', 'joke', 'just', 'k', 'kill', 'know', 'konnichiwa', 'last', 'later', 'learn', 'let', 'like', 'live', 'local', 'lone', 'made', 'maintain', 'make', 'me', 'mean', 'medic', 'medit', 'mental', 'mention', 'mom', 'money', 'more', 'morn', 'much', 'my', 'myself', "n't", 'name', 'need', 'new', 'nice', 'night', 'no', 'nobodi', 'not', 'noth', 'now', 'of', 'oh', 'ok', 'okay', 'ola', 'on', 'one', 'open', 'option', 'or', 'out', 'pass', 'past', 'peopl', 'pleas', 'possibl', 'practic', 'prepar', 'prevent', 'probabl', 'problem', 'profession', 'proper', 'realli', 'recov', 'relationship', 'repeat', 'respons', 'revoir', 'right', 'robot', 'sad', 'said', 'say', 'sayonara', 'scare', 'see', 'seek', 'sens', 'should', 'shut', 'sign', 'sister', 'sleep', 'slept', 'so', 'social', 'some', 'someon', 'someth', 'sound', 'start', 'stay', 'still', 'stress', 'stuck', 'stupid', 'suffer', 'suicid', 'support', 'sure', 'symptom', 'tag', 'take', 'talk', 'tell', 'than', 'thank', 'that', 'the', 'thee', 'then', 'therapi', 'therapist', 'there', 'thi', 'think', 'thought', 'through', 'to', 'today', 'told', 'treatment', 'trust', 'tu', 'understand', 'unwell', 'up', 'use', 'useless', 'veri', 'want', 'warm', 'way', 'we', 'well', 'were', 'what', 'whatev', 'where', 'whi', 'who', 'with', 'worri', 'worthless', 'would', 'wrong', 'ye', 'yeah', 'you', 'your', 'yourself']
```

In [7]:
```
# create training data
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)
```

In [8]:
```
# 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)
```

```
280 80
```

In [9]:
```
class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train
    # getitem_(self, index):
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]
    def __len__(self):
        return self.n_samples
```

In [10]:
```
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)
```

In [11]:
```
# 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)
        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}')
```

```
Epoch [100/1000], Loss: 0.2974
Epoch [200/1000], Loss: 0.0179
Epoch [300/1000], Loss: 0.0039
Epoch [400/1000], Loss: 0.0007
Epoch [500/1000], Loss: 0.0001
Epoch [600/1000], Loss: 0.0000
Epoch [700/1000], Loss: 0.0000
Epoch [800/1000], Loss: 0.0000
Epoch [900/1000], Loss: 0.0000
Epoch [1000/1000], Loss: 0.0000
final loss: 0.0000
```

In [12]:
```
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}')
```

```
training complete. file saved to data.pth
```

In [13]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')



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 = "Personal Therapeutic AI Assistant"
print("Let's chat! (type 'quit' to exit)")
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).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...")
```

```
Let's chat! (type 'quit' to exit)
You: Hi
Personal Therapeutic AI Assistant: Hi there. What brings you here today?
You: How do I feel better when I'm sad?
Personal Therapeutic AI Assistant: I do not understand...
You: I'm sad
Personal Therapeutic AI Assistant: Why do you think you feel this way?
You: I'm not sure, maybe I'm just feeling a bit down.
Personal Therapeutic AI Assistant: If your beliefs , thoughts , feelings or behaviours have a significant impact on your ability to function in what might be considered a normal or ordinary way, it would be important to seek help.
You: I'm feeling like my thoughts are affect my daily life.
Personal Therapeutic AI Assistant: Of course. Feel free to ask me anything. I'll do my best to answer you
You: Thank you, I appreciate it. I'll keep that in mind.
Personal Therapeutic AI Assistant: Oh sorry I didn't realise that. I'll try not to repeat myself again.
You: No problem at all. Thanks for letting me know.
Personal Therapeutic AI Assistant: You're most welcome!
You: I think I've got all the information I need for now. thanks again for your help.
Personal Therapeutic AI Assistant: You can call me Pandora.
You: Alright, Pandora it is! Thanks for sharing your name.
Personal Therapeutic AI Assistant: Oh I see. Do you want to talk about something?
You: No
Personal Therapeutic AI Assistant: I'm listening. Please go on.
You: quit
```

In [ ]: