```
import numpy as np
import nltk
import random


nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import json


from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()


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]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1
```

```
    return bag


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




with open('2-1.json', 'r') as f:
    intents = json.load(f)


print(intents)
```

```
 {'intents': [{'Season': 'KHARIF', 'Sector': 'HORTICULTURE', 'Category': 'Fruits', 'Cr
```

```
ignore_words=['?','!','.',',','(',')','&','@']


all_words = []
tags = []
xy = []
patternize=[]
processed_patternize=[]
answer=[]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['QueryType']              # tag=intent eg-Fertilizer,market price,cultiva
    ans=intent['KccAns']
    bname=intent['BlockName']                  #answers for the query text
    answer.append(ans)
    # add to tag list
    tags.append(tag)
    pattern=intent['QueryText']           #querytext
    patternize.append(pattern)
        # tokenize each word in the sentence
    w = pattern.split(" ")
    w.append(bname)

        # add to our words list
    all_words.extend(w)
    i=w
    i = [stem(k) for k in i if k not in ignore_words]  # i) removing punctuation words fro
    i=" ".join(i)
    processed_patternize.append(i)
        # add to xy pair
    xy.append((w, tag))

y_train_1 = tags


print(xy)
```

```
[(['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL'], 'Fertilizer Use and Availabi
```

```
print(processed_patternize)
```

```
['top dress for sapota palayankott', 'ask about weather report for tirupur avanashi',
```

```
print(all_words)
print(tags)
```

```
['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL', 'Asking', 'about', 'Weather',
['Fertilizer Use and Availability', 'Weather', 'Weather', 'Weather', 'Market Informat
```

```
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(tags)
```

```
['Agriculture Mechanization', 'Animal Breeding', 'Animal Nutrition', 'Animal Producti
```

```
print(all_words)
```

```
['&brown', '(adt', '(ae),tiruvannamali', '(bio)', '(bpt)', '(chithiraipattam)', '(cov
```

```
remove_words=['(',')','&']          #  removing the symbols in (,),&
all_wordsn=[]
for i in all_words:
  if i[0] in remove_words or i[-1] in remove_words:
    if i[0] in remove_words:
      i=i[1:]
    if i[-1] in remove_words:
      i=i[:-1]
      all_wordsn.append(i)
  else:
    all_wordsn.append(i)
print(all_wordsn)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'c
```

```
print(len(tags))
```

```
37
```

```
all_words=all_wordsn
print(all_words)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'o
```

```
X_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)     #all_words is a dictionary now.
    X_train.append(bag)



from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y_train_1)
list(le.classes_)
y_train = le.transform(y_train_1)             #  y_train_1 = tags


y_train
```

```
array([10, 35, 35, ..., 10,  7,  8])
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
print(len(X_train[1]))
```

```
789
```

```
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()


print(len(dataset))
```

```
3225
```

```
batch_size=8


train_loader = DataLoader(dataset=dataset,batch_size=batch_size,shuffle=True,num_workers=0


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



# Hyper-parameters
num_epochs = 20
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)
```

```
 789 37
```

```
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)


# 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()
    metrics="accuracy"


    print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Streaming output truncated to the last 5000 lines.
Epoch [8/20], Loss: 1.0534
Epoch [8/20], Loss: 0.7379
Epoch [8/20], Loss: 1.0929
Epoch [8/20], Loss: 0.3908
Epoch [8/20], Loss: 0.4830
Epoch [8/20], Loss: 0.6359
Epoch [8/20], Loss: 0.1993
Epoch [8/20], Loss: 1.4959
Epoch [8/20], Loss: 0.5917
Epoch [8/20], Loss: 0.6900
Epoch [8/20], Loss: 0.7697
Epoch [8/20], Loss: 1.1763
Epoch [8/20], Loss: 1.0267
Epoch [8/20], Loss: 0.9975
Epoch [8/20], Loss: 0.8363
Epoch [8/20], Loss: 0.2948
Epoch [8/20], Loss: 1.5503
Epoch [8/20], Loss: 1.6276
Epoch [8/20], Loss: 0.6154
Epoch [8/20], Loss: 0.5289
Epoch [8/20], Loss: 0.1426
Epoch [8/20], Loss: 0.6226
Epoch [8/20], Loss: 0.3220
Epoch [8/20], Loss: 1.2771
Epoch [8/20], Loss: 1.0755
Epoch [8/20], Loss: 1.3304
Epoch [8/20], Loss: 0.7011
Epoch [8/20], Loss: 0.9317
Epoch [8/20], Loss: 0.6149
Epoch [8/20], Loss: 0.6420
Epoch [8/20], Loss: 0.6209
Epoch [8/20], Loss: 1.7253
Epoch [8/20], Loss: 1.7271
Epoch [8/20], Loss: 1.1732
Epoch [8/20], Loss: 0.8336
Epoch [8/20], Loss: 0.5979
Epoch [8/20], Loss: 0.5216
Epoch [8/20], Loss: 1.2058
Epoch [8/20], Loss: 1.0533
Epoch [8/20], Loss: 0.4801
Epoch [8/20], Loss: 1.0122
Epoch [8/20], Loss: 0.8568
Epoch [8/20], Loss: 0.4010
Epoch [8/20], Loss: 1.3182
Epoch [8/20], Loss: 1.3910
Epoch [8/20], Loss: 0.7845
Epoch [8/20], Loss: 0.9322
Epoch [8/20], Loss: 0.3996
Epoch [8/20], Loss: 0.5476
Epoch [8/20], Loss: 0.6281
```

```
Epoch [8/20], Loss: 0.8757
Epoch [8/20], Loss: 0.8089
Epoch [8/20], Loss: 0.7847
Epoch [8/20], Loss: 0.4022
Epoch [8/20], Loss: 1.2580
Epoch [8/20], Loss: 0.3416
Epoch [8/20], Loss: 0.3526
Epoch [8/20], Loss: 0.6115
```

```
print(f'final loss: {loss.item():.4f}')
```

```
final loss: 1.1475
```

```
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)
```

```
import pandas
```

```
print(f'training complete. file saved to {FILE}')
```

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

```
import torch
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
data = torch.load(FILE)
```

```
res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
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)
```

```
<All keys matched successfully>
```

```
model.eval()
```

```
NeuralNet(
  (l1): Linear(in_features=789, out_features=8, bias=True)
  (l2): Linear(in_features=8, out_features=8, bias=True)
  (l3): Linear(in_features=8, out_features=37, bias=True)
  (relu): ReLU()
)
```

```
from difflib import get_close_matches


res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
bot_name = "Sinegalatha"
print("Let's chat! (type 'quit' to exit)")
test=[]
while True:
    # sentence = "do you use credit cards?"
    sentencei = input("You: ")
    if sentencei == "quit":
        break
    sentence = sentencei.split(" ")
    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()]
    print(tag)


    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
```

```
    for intent in intents['intents']:
        if tag == intent["QueryType"]:
          test.append(intent["QueryText"])

    p=[]
    p=(get_close_matches(sentencei, test))
    if len(p)==0:
      print("Make a call to Kisan Call Centre ")
    else:
      u=res[p[0]]
      print(u)
```

```
Let's chat! (type 'quit' to exit)
You: paddie varieties
Varieties
Recommended for ADT 36, ADT 39, ASD 16, ASD 18, MDU 5, CO 47,CORH 3, ADT 43, ADT (R)
You: quit
```

```
with open('test.json', 'r') as t:
    test = json.load(t)


print(test)


print(len(test['intents']))
```

```
72
```

```
predict_tag=[]
for i in test['intents']:
    sentencei = i['QueryText']
    sentence = sentencei.split(" ")
    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()]
    predict_tag.append(tag)



print(predict_tag)
```

```
['Market Information', 'Market Information', 'Market Information', 'Nutrient Manageme
```

```
predict_train = np.array(predict_tag)
```

```
test_train=[]
for i in test['intents']:
    sentencei = i['QueryType']
    test_train.append(sentencei)


test_train = np.array(test_train)


 from sklearn.metrics import accuracy_score


accuracy_score(predict_train, test_train)
```

```
0.6805555555555556
```