A **step-by-step guide** to create a **simple chatbot in PyTorch** trained on your own text data, specifically for your computer courses info (fees, duration, syllabus). We'll keep it simple so you can easily expand later.

We'll use a **bag-of-words + simple feedforward neural network** approach, which works well for FAQs. Here's a structured guide:

For this chatbot, here's the **list of Python packages you actually need to install**, along with the commands:-

---

1. PyTorch

PyTorch is required for building and training the neural network.

```
pip install torch
```

Depending on your system, you might want the CPU-only version (simpler) or GPU version. For most beginners, CPU-only works fine.

---

2. NLTK

NLTK is used for tokenization and stemming.

```
pip install nltk
```

After installing, you also need to download the tokenizer data once:

```
import nltk
nltk.download('punkt')
```

---

3. NumPy

NumPy is used for array and vector operations.

```
pip install numpy
```

Summary

These are the only external packages you need:

```
pip install torch nltk numpy
```

Everything else, like `json`, `os`, etc., is built-in Python, so no installation is required.

Prepare your data

Your chatbot will need data in a **question-answer format**. For example:

Save this in a file, e.g., `courses.json`.

courses.json:-

```
[
    {
        "question": "What is the fee for MS Office course?",
        "answer": "The MS Office course fee is 5,000 Rs."
    },
    {
        "question": "How long is the MS Office course?",
        "answer": "The MS Office course duration is 1 month."
    },
    {
        "question": "What is the fee for Tally with GST course?",
        "answer": "The Tally with GST course fee is 10,000 Rs."
    },
    {
        "question": "How long is the Tally with GST course?",
        "answer": "The Tally with GST course duration is 1 month."
    },
    {
        "question": "What is the fee for Full Stack Development course?",
        "answer": "The Full Stack Development course fee is 40,000 Rs."
    },
    {
        "question": "How long is the Full Stack Development course?",
        "answer": "The Full Stack Development course duration is 3 months."
    },
    {
```

```
        "question": "What is the fee for Data Science course?",
        "answer": "The Data Science course fee is 60,000 Rs."
    },
    {
        "question": "How long is the Data Science course?",
        "answer": "The Data Science course duration is 3 months."
    },
    {
        "question": "What is the fee for Digital Marketing course?",
        "answer": "The Digital Marketing course fee is 30,000 Rs."
    },
    {
        "question": "How long is the Digital Marketing course?",
        "answer": "The Digital Marketing course duration is 3 months."
    },
    {
        "question": "How can I book a free demo?",
        "answer": "You can book a free demo by visiting ."
    }
]
```

Train_chatbot.py file code :-

```
import json
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
import nltk

# Ensure required NLTK resources are downloaded
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

try:
    nltk.data.find('tokenizers/punkt_tab')
except LookupError:
    nltk.download('punkt_tab')

stemmer = PorterStemmer()
```

```
# ------------------------------
# Load your FAQ data
# ------------------------------
with open('courses.json', 'r') as f:
    data = json.load(f)

# ------------------------------
# Preprocessing
# ------------------------------
def tokenize(sentence):
    return word_tokenize(sentence)

def stem(word):
    return stemmer.stem(word.lower())

all_words = []
xy = []

for item in data:
    question = item['question']
    answer = item['answer']
    w = tokenize(question)
    all_words.extend(w)
    xy.append((w, answer))

all_words = [stem(w) for w in all_words if w.isalnum()]
all_words = sorted(set(all_words))
answers = [item['answer'] for item in data]

def bag_of_words(tokenized_sentence, all_words):
    sentence_words = [stem(w) for w in tokenized_sentence]
    bag = np.zeros(len(all_words), dtype=np.float32)
    for idx, w in enumerate(all_words):
        if w in sentence_words:
            bag[idx] = 1
    return bag

X_train = []
y_train = []

for (pattern_sentence, answer) in xy:
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    y_train.append(answers.index(answer))

X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
# ------------------------------
# Define Model
# ------------------------------
class ChatDataset(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ChatDataset, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

input_size = len(all_words)
hidden_size = 8
output_size = len(answers)

model = ChatDataset(input_size, hidden_size, output_size)

# ------------------------------
# Training
# ------------------------------
X_train_tensor = torch.from_numpy(X_train).float()
y_train_tensor = torch.from_numpy(y_train).long()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(1000):
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/1000], Loss: {loss.item():.4f}')

# ------------------------------
# Save model and metadata
# ------------------------------
FILE = "course_chatbot.pth"
torch.save({
```

```
    "model_state": model.state_dict(),
    "input_size": input_size,
    "hidden_size": hidden_size,
    "output_size": output_size,
    "all_words": all_words,
    "answers": answers
}, FILE)

print(f"Training complete. Model saved to {FILE}")
```

```
(env) F:\python demo\aimodel>python train_chatbot.py
Epoch [100/1000], Loss: 2.0937
Epoch [200/1000], Loss: 1.5494
Epoch [300/1000], Loss: 1.0328
Epoch [400/1000], Loss: 0.6376
Epoch [500/1000], Loss: 0.3777
Epoch [600/1000], Loss: 0.2296
Epoch [700/1000], Loss: 0.1483
Epoch [800/1000], Loss: 0.1017
Epoch [900/1000], Loss: 0.0734
Epoch [1000/1000], Loss: 0.0551
Training complete. Model saved to course_chatbot.pth
```

chat_with_bot.py file code (for chat with trained model):-

```
import torch
import torch.nn as nn
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
import numpy as np

stemmer = PorterStemmer()

# ------------------------------
# Preprocessing functions
# ------------------------------
def tokenize(sentence):
    return word_tokenize(sentence)

def stem(word):
    return stemmer.stem(word.lower())
```

```
def bag_of_words(tokenized_sentence, all_words):
    sentence_words = [stem(w) for w in tokenized_sentence]
    bag = np.zeros(len(all_words), dtype=np.float32)
    for idx, w in enumerate(all_words):
        if w in sentence_words:
            bag[idx] = 1
    return bag

# ------------------------------
# Load model and metadata
# ------------------------------
FILE = "course_chatbot.pth"
data = torch.load(FILE)

all_words = data["all_words"]
answers = data["answers"]
input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]

class ChatDataset(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ChatDataset, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = ChatDataset(input_size, hidden_size, output_size)
model.load_state_dict(data["model_state"])
model.eval()

# ------------------------------
# Chat loop
# ------------------------------
print("Bot is ready! Type 'quit' to exit.")
while True:
    sentence = input("You: ")
    if sentence.lower() == "quit":
        break

    X = bag_of_words(tokenize(sentence), all_words)
```

```
    X_tensor = torch.from_numpy(X).float()
    output = model(X_tensor)
    _, predicted = torch.max(output, dim=0)
    answer = answers[predicted.item()]
    print("Bot:", answer)
```

to run :-

python chat_with_bot.py

```
(env) F:\python demo\aimodel>python chat_with_bot.py
Bot is ready! Type 'quit' to exit.
You: book free demo
Bot: You can book a free demo by visiting
You: ms office course
Bot: The MS Office course duration is 1 month.
You: fees
Bot: The Tally with GST course fee is 10,000 Rs.
You: full stack course fees
Bot: The Full Stack Development course fee is 40,000 Rs.
You: data science course duration
Bot: The Data Science course fee is 60,000 Rs.
You:
```

Perfect! We can improve your chatbot so it **only answers when it's confident** about a response. Otherwise, it will say something like:

"I'm not sure about that. Please visit call 8149996597 for help."

This is done by **checking the softmax probability of the model's output**.

Here's a **modified version of your chat loop** with confidence checking chat_with_bot.py file code :-

```
import torch
import torch.nn as nn
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
import numpy as np

stemmer = PorterStemmer()

# ------------------------------
# Preprocessing functions
# ------------------------------
def tokenize(sentence):
    return word_tokenize(sentence)

def stem(word):
    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, all_words):
    sentence_words = [stem(w) for w in tokenized_sentence]
    bag = np.zeros(len(all_words), dtype=np.float32)
    for idx, w in enumerate(all_words):
        if w in sentence_words:
            bag[idx] = 1
    return bag

# ------------------------------
# Load model and metadata
# ------------------------------
FILE = "course_chatbot.pth"
data = torch.load(FILE)

all_words = data["all_words"]
answers = data["answers"]
input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]

class ChatDataset(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ChatDataset, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
```

```
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = ChatDataset(input_size, hidden_size, output_size)
model.load_state_dict(data["model_state"])
model.eval()

# ------------------------------
# Chat loop with confidence
# ------------------------------
print("Bot is ready! Type 'quit' to exit.")

while True:
    sentence = input("You: ")
    if sentence.lower() == "quit":
        break

    X = bag_of_words(tokenize(sentence), all_words)
    X_tensor = torch.from_numpy(X).float()
    output = model(X_tensor)

    # Apply softmax to get probabilities
    probabilities = torch.softmax(output, dim=0)
    confidence, predicted_index = torch.max(probabilities, dim=0)

    # Set a confidence threshold (e.g., 0.7)
    if confidence.item() > 0.7:
        answer = answers[predicted_index.item()]
    else:
        answer = "I’m not sure about that. Please  or call 8149996597 for
help."

    print("Bot:", answer)
```

□ What this does

1. Converts the model output into **probabilities** with `softmax`.
2. Checks if the **highest probability** is above `0.7` (70% confidence).
3. If it’s confident → returns the corresponding answer.
4. If not → returns a **default fallback message**.