**** 

**JSPM’s**

**Bhivarabai Sawant Institute of Technology & Research, Pune.**

***Department of Computer Engineering***

***Academic Year 2020-21***

**Mini Project Report on**

**Chatbot development.**

Submitted by

***A. Shiva Surya Saran***

Under the guidance of

***Prof. G. R. Virkar***

**Laboratory Practice – I**

***Department of Computer Engineering***

**JSPM’s Bhivarabai Sawant Institute of Technology & Research, Pune.**

***CERTIFICATE***

This is to certify that **A. Shiva Surya Saran** with Roll No. **BE B – 20** submitted his Project report under my guidance and supervision. The work has been done to my satisfaction during the academic year 2020-2021 under Savitribai Phule Pune University guidelines.

**Date:** *9th November, 2020.*

**Place:** *Pune.*

Prof. Gauri Virkar Dr. G. M. Bhandari

**Mini-project Guide HOD**

***Acknowledgement***

This is a great pleasure & immense satisfaction to express my deepest sense of gratitude & thanks to everyone who has directly or indirectly helped me in completing my Project work successfully.

I express my gratitude towards guide Prof. and Dr.G.M. Bhandari Head of Department of Computer Engineering, Bhivarabai Sawant Institute Of Technology and Research, Wagholi, Pune who guided & encouraged me in completing the Project work in scheduled time. I would like to thank our Principal, for allowing us to pursue my Project in this institute.

  **A. Shiva Surya Saran**

**(BE B - 20)**

***Index***

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Chapter** | **Page No** |
|  | Cover Page | 1 |
|  | Certificate | 2 |
|  | Acknowledgement | 3 |
|  | Index | 4 |
|  | Title | 5 |
| 1 | Introduction | 6 |
| 2 | Motivation | 7 |
| 3 | Objectives and Scope | 8 |
| 4 | System Requirements | 9 |
| 5 | Methodology | 10 |
| 6 | Applications | 18 |
| 7 | Conclusion | 19 |
| 8 | References | 20 |

***Title***

Design an Intelligent Virtual Assistant for Mobile billing i.e. Chatbot, designed to deliver an intuitive, automated experience by engaging the user in natural conversations using text.

***Problem Definition***

Create a chatbot framework and build a conversational model for a mobile network operator. The chatbot for this business needs to handle simple questions about Creating Bills, bill dues, and payment modes.

***Prerequisite***

* Knowledge of Python
* Robotics
* Basic concepts of PyTorch and Tensorflow.

***1. Introduction***

A chatbot is a service, powered by rules and sometimes artificial intelligence, that you interact with via a chat interface. The service could be any number of things, ranging from functional to fun, and it could live in any major chat product

Aim of this mini -project is to create chatbot designed to deliver an intuitive, automated experience by engaging users in natural conversations using text with virtual bot which would give immediate answers.

User will initiate the chatbot with an opening and ask for the issue for contacting the business.

Initially, this project will be designed to handle issues related to Bills and payments.

***2. Motivation***

Research shows that consumers expect immediate answers from the brand in every digital channel. The inability to answer users’ questions can quickly drive them away. Delivering prompt live assistance instead makes your customers feel that their needs are being met.

Increased mobile usage and a mobile mind shift led to the rise messaging service. Automated chatbot enables to react to your users’ inquiry coming from messaging channels immediately and quickly resolve and help them with answers.

Conversational models are a hot topic in artificial intelligence research. Chatbots can be found in a variety of settings, including customer service applications and online helpdesks.

These bots are often powered by retrieval-based models, which output predefined responses to questions of certain forms.

In a highly restricted domain like a company’s IT helpdesk, these models may be sufficient, however, they are not robust enough for more general use-cases. Teaching a machine to carry out a meaningful conversation with a human in multiple domains is a research question that is far from solved. Recently, the deep learning boom has allowed for powerful generative models like Google’s Neural Conversational Model, which marks a large step towards multi-domain generative conversational models.

***3. Objectives and Scope***

***Learning Objectives***

We will learn how to design and develop a chat using Pytorch which is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing.

***Outcome***

By the end of this project, we should have a working Chatbot prototype ready to be executed with the concepts that we have learnt.

***4. System Requirements***

**Hardware**

* Quad-core 64-bit i7 Intel Processor
* Nvidia Graphics Processor with Cuda enabled.
* 8 GB RAM
* 16 GB storage

**Software**

* Ubuntu 20.04 Operating System
* PyCharm IDE
* Python package
* PyTorch Package

***5. Methodology***

***Importing Libraries***

We will first import the relevant libraries. Some of the most important libraries that have been used here are as below.

*import numpy as np*

*import random*

*import json*

*import nltk*

*import torch*

*import torch.nn as nn*

*from torch.utils.data import Dataset, DataLoader*

***Creating Custom Functions***

We will create custom Functions so that it is easy for us to implement afterwards.

*def tokenize(sentence):*

*return nltk.word\_tokenize(sentence)*

*def stem(word):*

*return stemmer.stem(word.lower())*

Nltk or natural language took kit is a really useful library that contains important classes that will be useful in any of your NLP task.

***Stemming***

If we have 3 words like “walk”, “walked”, “walking”, these might seem different words but they generally have the same meaning and also have the same base form; “walk”. So, in order for our model to understand all different form of the same words we need to train our model with that form. This is called Stemming. There are different methods that we can use for stemming. Here we will use Porter Stemmer model form our NLTK Library.

*from nltk.stem.porter import PorterStemmer*

*stemmer = PorterStemmer()*

***Bag of Words***

We will be splitting each word in the sentences and adding it to an array.

We will be using bag of words. Which will initially be a list of zeros with the size equal to the length of the all words array.

If we have a array of *` sentences = ["hello", "how", "are", "you"]`*

and an array of total *` words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]`* then its bag of words array will be `*bag = [ 0 , 1 , 0 , 1 , 0 , 0 , 0]`.*

We will loop over the each word in the all words array and the bog array corresponding to each word. If a word from the sentence is found in the all words array, 1 will be replaced at that index/position in bog array.

*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*

***Loading the Data & Cleaning it***

We will be using a data set called intents.json which has the following structure.

*```json*

*{*

*"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"]*

*},*

*```*

Now we will simply load the json file using *\*\*\*json.load()\*\*\** function.

*```python*

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

*intents = json.load(f)*

*```*

In order to get the right information, we will be unpacking it with the following code.

*```python*

*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))*

*```*

This will separate all the tags & words into their separate lists

***Implementing custom Functions***

*```python*

*# 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)*

*```*

***Creating Training Data***

We will transform the data into a format that our PyTorch Model can Easily Understand

*```python*

*# 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: PyTorch CrossEntropyLoss 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)*

*```*

***PyTorch Model***

Here we will be making a class to implement our custom neural network. It will be a feed forward neural Network which will have 3 Linear Layers and we will be using activation function “ReLU” ..

***Feed Forward Neural Network***

The feedforward neural network was the first and simplest type of artificial neural network devised In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any) and to the output nodes.

***Activation Function***

An activation function is a function used in artificial neural networks which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold. If the inputs are large enough, the activation function "fires", otherwise it does nothing. In other words, an activation function is like a gate that checks that an incoming value is greater than a critical number.

***ReLU Function***

There are a number of widely used activation functions in deep learning today. One of the simplest is the rectified linear unit, or ReLU function, which is a piece wise linear function that outputs zero if its input is negative, and directly outputs the input otherwise:



***Creating our Model***

*```python*

*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*

*```*

Here we have inherited a class from NN.Module because we will be customizing the model & its layers

***Assigning the Dataset to the Model***

We will use some Magic functions, write our class.

*```python*

*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*

*```*

***Hyper Parameters***

Every Neural network has a set of hyper parameters that need to be set before use.

Before Instantiating our Neural Net Class or Model that we wrote earlier, we will first define some hyper parameters which can be changed accordingly.

*```python*

*# 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)*

*```*

***Loss and optimizer***

We will now Instantiate the model, loss and optimizer functions.

*Loss Function: Cross Entropy*

*Optimizer: Adam Optimizer*

*```python*

*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)*

*```*

***Training the Model***

*```python*

*# 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*

*}*

*```*

***Saving the Trained Model***

*```python*

*FILE = "data.pth"*

*torch.save(data, FILE)*

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

*```*

***Loading our Saved Model***

*```python*

*device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')*

*with open('intents.json', 'r') as json\_data:*

*intents = json.load(json\_data)*

*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()*

*```*

***Using the Chatbot***

Our Model is Ready, we just need to execute chat.py to initiate the chat bot in Terminal.

***6. Applications***

With this project, it could be applied wide area of businesses across various industries.

Customer service departments benefit from them a lot as bots help companies automate support tasks and let them save a lot of money.

It would work well as entertainment chatbots as they engage users and allow to carry out long conversations. For example, Disney offers a chatbot that engages children into some crime puzzles with Lieutenant Judy Hopps, the movie character.

Businesses can also use more button-based chatbots that can help complete users various actions. Pizza bot allows customers to order pizza without leaving the chat window. Travel chatbot enables you to book a flight. Other chatbots that can help you check the weather, inform you about the latest news, schedule a meeting with your doctor or even complete the money transfer.

Chatbots can help not only your customers but also your workmates. Chatbots that integrate with major messaging applications can train new employees and help them complete their onboarding.

***8. Conclusion***

This project took me through the various phases of Chatbot development using PyTorch and also gave a brief outlook of Contextual Natural Language Processing in Chat bot development.

***9. References***

* <https://pytorch.org/tutorials/beginner/chatbot_tutorial.html>
* <https://www.nuance.com/omni-channel-customer-engagement/digital/virtual-assistant/nina.html>
* <https://www.nuance.com/omni-channel-customer-engagement/nuance-mix/tools.html>
* <https://www.jetbrains.com/pycharm/>