Development of a Pizza Ordering Chatbot

Hardik Gohil

15/2/2024

Abstract

This report outlines the development of a chatbot using Dialogflow and TensorFlow. The project involves two distinct phases: Dialogflow implementation for making a pizza ordering chatbot and TensorFlow implementation for making a chatbot which can talk and understand natural language and is trained on bAbl dataset

1 Objective

The objective of this project is to make a

- User friendly Pizza ordering chatbot using Dialogflow
- Chatbot that understands natural language using Tensorflow and train it on the bAbl dataset

.

2 Introduction

Chatbots have become increasingly popular in the food service industry, providing a convenient way for customers to place orders and inquire about menu items. In this project, we leverage Dialogflow for conversational interface setup. We also learn how to code our own chatbot using Tensorflow and train it to understand Natural language

3 Methodology: Dialogflow Implementation

Dialogflow is a powerful platform for building conversational interfaces using natural language understanding (NLU). It offers various features such as intents, entities, and contexts to design interactive chatbots. Intents represent specific user intentions or actions, while entities extract relevant information from user inputs. Contexts help maintain conversational state and facilitate dialogue flow.

3.1 Key Features of Dialogflow

- Intents: Define user actions or intentions.
- Entities: Extract relevant information from user inputs.
- Contexts: Maintain conversational state and facilitate dialogue flow.
- Fulfillment: Integrate with backend systems to fulfill user requests.

4 Methodology: Natural Language Processing using TensorFlow

TensorFlow is a popular open-source machine learning framework developed by Google. It provides tools and libraries for building and deploying machine learning models, including natural language processing (NLP) tasks. In this project, TensorFlow is used to train a chatbot using the bAbl dataset so that it can understand user language

4.1 NLP with TensorFlow

- Tokenization: Splitting text into individual tokens or words.
- Word Embeddings: Representing words as dense vectors in a high-dimensional space.
- Sequence Modeling: Modeling sequential data such as text or speech.
- Neural Network Architectures: Building deep learning models for NLP tasks like classification, sequence labeling, and language generation.

5 Code: Dialogflow Implementation

In the dialogflow implementation, there is no code. Everything was done on the Dialogflow user interface
Steps for implementation:

- Created an agent named pizzaBot
- Added features for it to perform better small talk
- Small talk includes features like About the agent, Courtesy, Emotions, Greetings, etc.
- Created an intent named "Order a pizza"
- Added training phrases for that intent like
  - Order me a large pizza

  - Order me a pizza
  - Order me a pizza now
  - Order me a pizza at address XYZ
  - and many more ...

- Added actions and parameters which will be extracted from the input by comparing them to the training phrases
- These parameters include phone number, time, address, email, toppings and size
- An action asking the user to enter the missing parameters is performed if the user has not yet given a certain parameter
- The parameters toppings and size are custom made entities which can take only certain values which we have defined while creating the entity
- Finally, the chatbot gives a response which says order is successful and gives the final list of all the parameters and their values

6 Code: Natural Language Processing with TensorFlow

```python
import pickle
import numpy as np

with open("train_qa.txt", "rb") as fp:
    train_data = pickle.load(fp)

train_data

with open("test_qa.txt", "rb") as fp:
    test_data = pickle.load(fp)

test_data

print(type(train_data), type(test_data))
print(len(train_data), len(test_data))

story = " ".join(train_data[0][0])
print(story)

question = " ".join(train_data[0][1])
print(question)

answer = train_data[0][2]
print(answer)
```

```
vocab = set()

all_data = test_data+train_data

len(all_data)

for story, question, answer in all_data:
    vocab = vocab.union(set(story))
    vocab = vocab.union(set(question))


vocab.add('yes')
vocab.add('no')

vocab

len(vocab)

vocab_len = len(vocab) + 1

max_story_len = max([len(data[0]) for data in all_data])
max_story_len

max_question_len = max([len(data[1]) for data in all_data])
max_question_len

#Vectorize the data

import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import
    pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(filters = [])

tokenizer.fit_on_texts(vocab)

tokenizer.word_index

train_story_text = []
train_question_text = []
train_answers = []
for story, question, answer in train_data:
    train_story_text.append(story)
    train_question_text.append(question)

train_story_seq = tokenizer.texts_to_sequences(
    train_story_text)
```

```
len(train_story_text)

len(train_story_seq)

def vectorize_stories(data, word_index = tokenizer.
    word_index, max_story_len = max_story_len,
    max_question_len = max_question_len):
    X = [] #stories
    Xq = [] #questions
    Y = [] #answer

    for story, question, answer in data:
        x = [word_index[word.lower()] for word in story]
        xq = [word_index[word.lower()] for word in question]
        y = np.zeros(len(word_index) + 1)
        y[word_index[answer]] = 1

        X.append(x)
        Xq.append(xq)
        Y.append(y)

    return(pad_sequences(X, maxlen = max_story_len),
           pad_sequences(Xq, maxlen = max_question_len),
           np.array(Y))

inputs_train, queries_train, answers_train =
    vectorize_stories(train_data)

inputs_test, queries_test, answers_test = vectorize_stories(
    test_data)

inputs_train

tokenizer.word_index["yes"]

tokenizer.word_index["no"]

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Input, Activation, Dense
    , Permute, Dropout, add, dot, concatenate, LSTM,
    Embedding

input_sequence = Input((max_story_len,))
question = Input((max_question_len,))

#Input Encoder m
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim = vocab_len,
    output_dim = 64))
```

```
input_encoder_m.add(Dropout(0.3))

#Input Encoder c
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim = vocab_len,
    output_dim = max_question_len))
input_encoder_c.add(Dropout(0.3))

#Question Encoder
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim = vocab_len,
    output_dim = 64, input_length = max_question_len))
question_encoder.add(Dropout(0.3))

#Encode the sequences
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)

match = dot([input_encoded_m, question_encoded], axes =
    (2,2))
match = Activation('softmax')(match)

response = add([match, input_encoded_c])
response = Permute((2,1))(response)

answer = concatenate([response, question_encoded])

answer = LSTM(32)(answer)

answer = Dropout(0.5)(answer)

answer = Dense(vocab_len, activation = 'softmax')(answer)

# answer = Activation('softmax')(answer)

model = tf.keras.Model([input_sequence, question], answer)
model.compile(optimizer = 'adam', loss = '
    categorical_crossentropy', metrics = ["accuracy"])

model.summary()

history = model.fit([inputs_train, queries_train],
    answers_train,
                    batch_size = 32, epochs = 20,
                    validation_data = ([inputs_test,
                        queries_test], answers_test))

import matplotlib.pyplot as plt
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])

plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("epochs")


#save the model
model.save("chatbot_model")

#Evaluation on the Test set
model.load_weights("chatbot_model")

pred_results = model.predict(([inputs_test, queries_test]))

test_data[0][0]

story = " ".join(word for word in test_data[0][0])
story

query = " ".join(word for word in test_data[0][1])
query

test_data[0][2]

val_max = np.argmax(pred_results[13])
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted answer:", k)
print("Probability of certainity:", pred_results[13][val_max
    ])

story = "Mary dropped the football . Sandra discarded apple
    in kitchen . Daniel went to office"
question = "Is Daniel in the kitchen ? "

my_data = [(story.split(), question.split(), 'yes')]

my_story, my_question, my_answer = vectorize_stories(my_data
    )

pred_results = model.predict(([my_story, my_question]))

val_max = np.argmax(pred_results[0])
for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key
```

```
print("Predicted answer:", k)
print("Probability of certainity:", pred_results[0][val_max
    ])
```

6.1 Import Libraries

We start by bringing in some essential tools for our project:

- `pickle` for bringing our data into play,
- `numpy` for handling all our number crunching needs,
- `tensorflow` and its powerful companion `keras` to build and train our neural network,
- `matplotlib` for visually tracking our model's learning progress.

6.2 Data Loading

We use the `pickle` library to load our datasets filled with stories, questions, and answers. This content is crucial for teaching our model how to understand and respond.

6.3 Preprocessing

To get our data ready for the model, we go through a few steps:

1. Creating a comprehensive list of all the unique words found in our stories and questions, plus adding 'yes' and 'no' to cover all answer bases.
2. Figuring out the length of the longest story and question to ensure our model receives data in a consistent format.

6.4 Tokenizer

We employ a Keras tokenizer to process our text by:

- Mapping each unique word to a specific number to easily understand our vocabulary.
- Converting our textual data into sequences of these numbers, prepping it for model training.

6.5 Vectorize Stories

With a special function called `vectorize_stories`, we:

1. Transform stories, questions, and answers into uniform sequences of numbers.
2. Prepare our answers in a binary format, making them ready for the model to classify.

6.6 Model Architecture

Our model is uniquely structured to tackle question-answering challenges by:

- Utilizing separate layers to process the input story and question through embeddings.
- Applying a combination technique to align the story and question data, facilitating an appropriate response.
- Incorporating LSTM and dense layers to digest this combined information and generate predictions.

It's all set up with the Adam optimizer and focused on achieving the highest accuracy possible.

6.7 Training

The training phase involves teaching our model using the prepared data and monitoring its performance through accuracy metrics over both training and validation phases.

6.8 Model Evaluation and Prediction

In the final stage, we:

1. Introduce new stories and questions to the model.
2. Ensure they're properly formatted for the model's consumption.
3. Observe how the model predicts answers and compare those to the actual answers to assess its performance.

7 Conclusion

The first part of the project successfully integrates Dialogflow for initial conversational interface setup. By leveraging the technology, we have developed a pizza ordering chatbot that offers a seamless user experience and demonstrates the potential of AI in customer service automation.
The second part of the project demonstrates the application of neural networks in natural language processing, specifically in building a question-answering system. Through careful preprocessing, model design, and training, the system learns to accurately answer questions based on provided stories.