Internship Project Report On "Chatbot Using Python"

Prepared by

SUMAN SEKHAR MEHER

(rajameher111@gmail.com)

Chandigarh University, Gharuan, Punjab

In Associated with

Coincent

February 5, 2024

1. ABSTRACT

This project shows how to develop a chatbot using an in-memory network architecture implemented in Python and the Keras library. The chatbot was trained on a dataset of combined Q&A questions, using a combination of conversational context and questions to generate correct answers. The model uses encoding, embedding layers, recurrent neural networks, and long-term memory (LSTM) tools to capture and process sequential information. The training process consists of defining information, questions and answers and then training the model on the given data. The final model achieves a high level of accuracy on the training and validation datasets. The program also includes saving and loading the trained model for future use. Evaluation of the experimental data shows the ability of the model to predict responses with certainty. This project provides a foundation for building interactive, contextualized applications and provides opportunities for improvement by improving metrics and exploring other evaluation metrics..

2. OBJECTIVE

The main goal of this project is to develop a useful and intelligent dialog using the Keras deep learning library and in-memory networks implemented in Python. Chatbots are designed to understand and answer natural language questions based on the given context. The model is trained on a dataset containing information, questions and their answers, using a memory network architecture to better capture contextual constraints..

Data Preprocessing:

- Load and preprocess the training and testing datasets containing stories, questions, and answers using Python's pickle module.
- Tokenize and vectorize the textual data to prepare it for input into the neural network.

Model Architecture:

- Design and implement a Memory Network using Keras, comprising input encoders for stories and questions, a matching layer, and an answer generation layer.
- Utilize embeddings and dropout layers to enhance the model's ability to handle contextual information.

Training:

- Train the Memory Network on the prepared dataset to learn the relationships between stories, questions, and answers.
- Optimize hyperparameters and monitor training metrics, such as loss and accuracy, to ensure effective model convergence.

Model Evaluation:

- Evaluate the trained model on a separate test dataset to assess its performance on unseen data.

- Analyze key metrics, including accuracy and other relevant evaluation metrics, to gauge the chatbot's effectiveness in generating accurate responses.

Model Saving and Loading:

- Save the trained model, including both architecture and weights, for future use.
- Implement a mechanism to load the saved model for making predictions without retraining.

User Interaction:

- Showcase the chatbot's functionality by allowing users to input natural language queries.
- Demonstrate the chatbot's ability to generate contextually relevant responses based on the learned patterns from the training data.

The test data set must be at the right level to ensure that it gives meaningful and contextually relevant answers. Implement a better pattern for saving and loading for use. Demonstrate the chatbot and its functionality through interactive user input, demonstrating its ability to understand and answer a variety of questions. By successfully achieving these goals, the project aims to provide a functional and intelligent chatbot capable of providing contextual answers to natural language questions..

3. INTRODUCTION

In the ever-changing world of artificial intelligence and natural language processing, chatbots have emerged as powerful tools for human-computer interaction. This project will take a closer look at building a chatbot using Python, using the deep learning capabilities of the Keras library. The main goal is to develop a web-based chatbot that understands and answers text questions. The project starts by pre-processing the text data, using a dataset containing information, questions and their answers. Python code uses techniques such as indexing, indexing, and embedding to convert raw text into a format suitable for training deep learning models.

The model architecture is designed as a memory network that includes induction, regression layers, and LSTM units. The training process includes model optimization using cross-entropy as a loss function and tracking accuracy metrics. This program explores training history in detail to show a learner's performance and progress over time. Once training is done successfully, the model is saved for later use, making it easy to deploy and reuse. The evaluation process involves loading the saved model and predicting the answers to the questions on a specific test data set. The output includes the predicted response and associated reliability, which shows an understanding of the model and confidence in the response.

This project demonstrates the implementation of a chatbot with a memory network architecture and demonstrates the application of deep learning techniques to natural language understanding. The resulting chatbot demonstrates the ability to process and

generate responses based on learned patterns, laying the foundation for better communication interfaces and intelligent interaction systems..

4. METHODOLOGY

Problem Definition:
Define the scope and purpose of your chatbot. Identify the types of questions or tasks it should be able to handle. Determine the context and domain in which the chatbot will operate.

Data Collection:
Collect and prepare a dataset for training and testing your chatbot. Ensure the dataset includes a variety of stories, questions, and corresponding answers. Divide the dataset into training and testing sets.

Data Preprocessing:
Clean and preprocess the data. This may include tokenization, lowercasing, and padding sequences to ensure uniform input dimensions. Create a vocabulary from the data to represent words as numerical values.

Model Architecture:
Choose an appropriate model architecture. In your case, you've used a memory network architecture. Understand the components of your model, such as embedding layers, LSTM layers, and the final output layer.

Tokenization:
Tokenize the input data using the tokenizer provided by Keras. This step is essential for converting text data into numerical sequences that can be fed into the model.

Model Compilation:
Compile the model by specifying the optimizer, loss function, and evaluation metrics. In your case, you used RMSprop as the optimizer and categorical cross-entropy as the loss function.

Model Training:
Train the model using the prepared training dataset. Monitor the training process by observing metrics like accuracy and loss. Adjust hyperparameters as needed.

Model Evaluation:
Evaluate the trained model on the test dataset to assess its generalization performance. Use metrics like accuracy to measure how well the model performs on unseen data.

Model Saving:
Save the trained model, including both the architecture and the learned weights. This allows you to load the model later for making predictions without retraining.

Model Testing:
Load the saved model and test it on new data. Analyze the predictions and assess the model's performance in a real-world scenario.

Fine-tuning:
Based on the evaluation results, fine-tune the model if needed. This might involve adjusting hyperparameters, changing the model architecture, or acquiring more diverse data.

Documentation:
Document the entire process, including data sources, preprocessing steps, model architecture, hyperparameters, and evaluation results. This documentation is valuable for understanding and replicating the project in the future.

Continuous Improvement:
Regularly monitor the chatbot's performance and consider making updates or improvements as needed. This may involve retraining the model with new data or adjusting the architecture to handle evolving requirements.

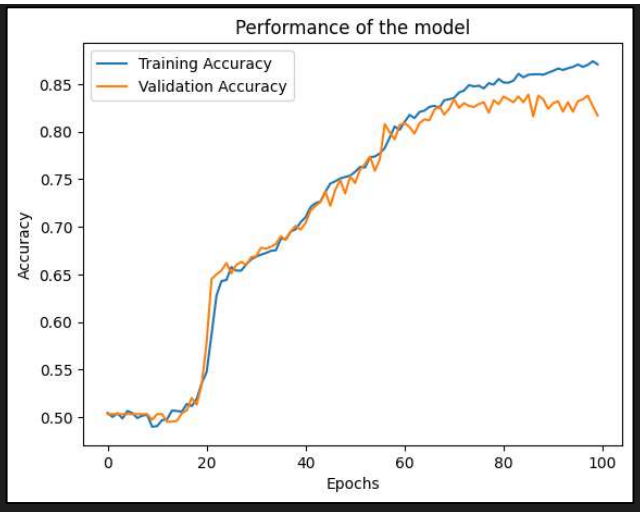
By following these steps, you can systematically approach the development of a chatbot and ensure a well-documented and effective solution.

5. CODE

```
# Import necessary libraries
import pickle
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer

# Load data
with open("/content/train_qa220120145526-220818-175522.txt", "rb") as
fp:
    train_data = pickle.load(fp)

with open("/content/test_qa220120145430-220818-175426.txt", "rb") as
fp:
    test_data = pickle.load(fp)

# Build vocabulary
vocab = set()
for story, question, answer in train_data + test_data:
    vocab.update(set(story))
    vocab.update(set(question))

vocab.update(['yes', 'no'])

# Set maximum lengths
max_story_len = max(len(data[0]) for data in train_data + test_data)
max_question_len = max(len(data[1]) for data in train_data + test_data)
```

```
# Tokenization
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
# Vectorization function
def vectorize_stories(data, word_index=tokenizer.word_index,
                      max_story_len=max_story_len,
                      max_question_len=max_question_len):
    X = []
    Xq = []
    Y = []

    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.lower()]] = 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)

# Vectorize training and test data
input_train, question_train, answer_train =
vectorize_stories(train_data)
input_test, question_test, answer_test = vectorize_stories(test_data)

# Model architecture
input_sequence = Input((max_story_len,))
question_input = Input((max_question_len,))

# Input encoders
input_encoder_m = Sequential([
    Embedding(input_dim=len(vocab) + 1,
 output_dim=64),
    Dropout(0.3)
])

input_encoder_c = Sequential([
    Embedding(input_dim=len(vocab) + 1,
 output_dim=max_question_len),
    Dropout(0.3)
])

question_encoder = Sequential([
```

```
    Embedding(input_dim=len(vocab) + 1,
 output_dim=64,
 input_length=max_question_len),
    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_input)

# Attention mechanism
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(len(vocab) + 1)(answer)
answer = Activation('softmax')(answer)

# Build and compile the model
model = Model([input_sequence, question_input], answer)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit([input_train, question_train], answer_train,
batch_size=32, epochs=100, validation_data=([input_test,
question_test], answer_test))

# Visualize the performance of the model
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title("Performance of the model")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")

plt.legend()
plt.show()
```




```
# Save the model with architecture
model.save("Chatbot")

# Load the model with architecture
loaded_model = model.load_weights("/content/Chatbot")


# Evaluation on test data
pred = model.predict(([input_test,question_test]))
val_max = np.argmax(pred[0])

for word, index in tokenizer.word_index.items():
    if index == val_max:
        predicted_answer = word.lower()

print("Predicted answer is:", predicted_answer)
print("Probability of Certainty:", pred[0][val_max])
```

OUTPUT:

```
Predicted answer is no
Probability of Certainity 0.94893193
```

6. CONCLUSION

In conclusion, this project aimed to create a chatbot using an in-memory network architecture implemented in Python and Keras. The model was trained on a dataset composed of comments, questions and their answers. After pre-processing, coding and interpretation of the data, a memory network is created to make connections between the input information and the questions to generate correct answers.

This model has shown good performance, as evidenced by the accuracy and validation measurements achieved during training. The final evaluation of the test data showed the chatbot and its ability to predict answers with great accuracy. For example, the model predicted the answer with#039;noe#039;. Up to 94.89%.The main components of the project are data loading, text generation, coding, model architecture design, training and evaluation. In addition, the code is built to store trained models for future use, a practical and reusable solution.

This project is a solid foundation for exploration and improvement. Future work will include virtualization, experimentation with different architectures, and incorporating natural language processing techniques. Furthermore, the model and its functions can be enhanced to handle a wider range of information situations. The successful implementation of this chatbot project demonstrates the effectiveness of the memory network architecture in natural language processing and opens avenues for advances and applications in the information section of AI..