
The Architecture & Design of the Chatbot

Author:

Mehrad Haghshenas

Date:

October 5, 2023

Introduction

This report provides an overview of the architecture and design of the Chatbot, an expert system addressing queries on the paper on *An automated approach to estimating code coverage measures via execution logs* by Chen et al. [1]

Technical Details

The Chatbot is built using Python version 3.9. It is advised to use this version to ensure compatibility given that the libraries employed in this module may not be guaranteed to work seamlessly on other Python versions. The design of the Chatbot is based on deep learning and neural networks. Key libraries used are *NLTK*, *TFLearn*, *TensorFlow*, *NumPy*, *Random*, *JSON*, and *Pickle*. Before running the code, please check the directory of the *intent.json* file. The codes were written in Kaggle's interface and thus, some minor changes might be necessary to run the codes on a local computer.

Design and Architecture

The Chatbot's design and architecture are characterized by a data-driven approach and a neural network-based decision-making system. Essentially we *classify* the query from the user into one of the predefined categories and generate a response according to the available responses under the same tag. The project contains two main files: *Intent File Structure* and *Python Codes*.

Knowledge Base

The intent file is a JSON file serving as the foundational structure for the Chatbot's knowledge base. It comprises a single dictionary of *tags*, *patterns*, and *responses*. Each tag represents a category or topic, and under each tag, there are associated patterns (related queries) and corresponding responses. When the user asks a question, the question is tokenized, the tokens are compared against the tokens of the questions in the Intent file. If the query from the user has a high similarity with the questions under one tag, then a random response from that category is generated by the Chatbot. Note that the patterns and responses under the same tag ideally have the same *semantics*; however, are different in structure and vocabulary. This structure allows for categorization of user input and retrieval of relevant responses. The intent file was generated by the Author. Nevertheless, ChatGPT was used to help creating a more complete and exhaustive knowledge base.

Code Base

Data preprocessing plays a key role in preparing the intent file for training the neural network. The process involves:

-
- Tokenization: Patterns within the intent file are tokenized using NLTK. This step breaks down sentences into individual tokens. This approach is very similar to what a *lexer* does when parsing a program.
 - Stemming: Stemming is employed to reduce words to their root form. As an example *Jumping* is reduced to *Jump*.
 - Bag-of-Words: For each pattern, a bag-of-words representation is generated. This representation counts the occurrences of each word in the pattern. This is quite different from one-hot encoding given that the frequency of the token is important as well not just the existence of a token.

The data processed in this section will be used as the input data of the neural network. The network has a feed forward architecture with three key layers:

- Input Layer: The input layer receives the bag-of-words representation of user queries.
- Hidden Layers: Three hidden layers are presented in the model each comprising eight neurons. The number of hidden layers and the number of neurons are hyperparameters which can be tweaked for further improvement in the accuracy.
- Output Layer: The output layer contains neurons equal to the number of unique tags in the intent file. It utilizes the softmax activation function to provide probabilities for the tags.

The training of the neural network is performed as follows:

- Data Format: The preprocessed data, including the bag-of-words representations and corresponding tags, serve as training input and output, respectively.
- Loss Function: The model employs categorical cross-entropy as the loss function.
- Optimization: Back propagation is utilized for optimizing the neural network's weights and biases.
- Hyperparameters: Several hyperparameters such as epochs, batch size, and learning rate are taken into account.

Conclusion

In conclusion, the chatbot's design emphasizes data preprocessing and neural network architecture to create an efficient and responsive conversational agent. The intent file structure provides a structured knowledge base, while the neural network is trained to make *intelligent* responses.

In the following is an execution log of the Chatbot. The responses highlighted with yellow are deemed incorrect by the author. Note that *I don't understand your question* is not an incorrect response. The main reason for the wrong responses are deemed to be rooted in the knowledge base. Specifically, the intent file is not structured properly; i.e., the tags need to be completely different and the patterns and responses under one tag are required to be very similar. However, the log shows that some different questions were bundled under the same tag because we receive a related response but not the one we are looking for. In some cases we received *I don't understand your question*. This is because we put a threshold of 70% meaning that if the query falls into a category with a probability of higher than 70%, then the response is accordingly generated. Otherwise, the maximum tag corresponding to the query will not be taken into account. Should the Chatbot generate the answer *I don't understand your question* to a large proportion of the queries, it may be due to the fact that the tags are not completely different. This results into a question having similarities with different categories and thus, cannot be classified with a high probability to only one category. In my opinion, the training of the model and adding more hidden layers might have limited improvement to the performance of the Chatbot. Putting more emphasis on providing a comprehensive knowledge base will lead to a better performance from the tool. To re-iterate the knowledge base is not complete and was the work of the author along with some cases of using Chatgpt. Finally, due to the lack of time and focusing on my thesis, I did not have sufficient time to improve the performance of the Chatbot to achieve the desired performance.

```
Lets start chatting...(type quit to stop)
ask your question: what is the title of the paper?
An Automated Approach to Estimating Code Coverage Measures via Execution Logs
ask your question: what is the paper about?
Automating Code Coverage using Execution Logs
ask your question: what is the overall message of the paper?
It introduces LogCoCo, an automated code coverage estimation approach.
ask your question: give me a brief abstract of the paper
HBase was selected for its widespread usage, serving millions of users in companies like
Facebook and Twitter.
ask your question: give me one strength of LogCoCo
The excessive instrumentation guarantees accurate measurements of code coverage.
However, problems like deployment challenges and performance overhead are imposed.
LogCoCo on the other hand, is easy to setup and imposes little performance overhead
by analyzing the execution logs.
ask your question: give me one weakness of LogCoCo
The excessive instrumentation guarantees accurate measurements of code coverage.
However, problems like deployment challenges and performance overhead are imposed.
LogCoCo on the other hand, is easy to setup and imposes little performance overhead
```

by analyzing the execution logs.

ask your question: What are the challenges of LogCoCO?

Generalizability: LogCoCo primarily focuses on server-side systems with extensive logging. To enhance its applicability, research studies can explore ways to adapt LogCoCo to mobile applications and client/desktop-based systems with limited or no logging. This expansion would make LogCoCo a more versatile tool. One way to do this expansion can be through focusing on developing automated tools or techniques that strategically insert logging statements into source code where there are limited logging available. This technique can also be used to reduce the amount of May labels.

ask your question: can LogCoCo be used in mobile applications?

LogCoCo's approach of inferring system execution contexts is significant because it allows indirect detection of covered methods.

ask your question: What are the issues associated with code coverage tools like JaCoCo in practice?

JaCoCo did not report code coverage measures for some modules, particularly those not directly invoked by the YCSB benchmark suite.

ask your question: how many experiments were conducted?

Measuring code coverage in a DevOps-like environment is challenging due to rapid deployment processes.

ask your question: what is the Hbase Experiment?

The experiment used the YCSB benchmark suite, originally developed by Yahoo!.

ask your question: what are the engineering challenges of using JaCoCO?

The paper discusses software testing and code coverage measures.

ask your question: what are the engineering challenges?

I don't understand your question

ask your question: in the paper a breadth for search algorithm was used explain

The HBase experiment used a three-machine-cluster with specific hardware specs.

ask your question: quit

References

- [1] Boyuan Chen, Jian Song, Peng Xu, Xing Hu, and Zhen Ming Jiang. An automated approach to estimating code coverage measures via execution logs. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 305–316, 2018. pages 2