**Learning to Play Rock Paper Scissors**

By

**Naga Sai Ganesh Srikar Bulusu**

A DISSERTATION

Submitted to  
**The University of Liverpool**

in partial fulfillment of the requirements  
for the degree of  
**MASTER OF SCIENCE**

20-09-2024

Dedication

STUDENT DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of  
others is set forth, quotation marks so indicate, and that appropriate credit is given where I  
have used the language, ideas, expressions or writings of another.

I confirm that I have not copied material from another source or committed plagiarism nor  
commissioned all or part of the work (including unacceptable proof-reading) nor fabricated, falsified or embellished data when completing the attached piece of work.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Naga Sai Ganesh Srikar Bulusu

Learning to play Rock Paper Scissors

Abstract:

This project aims at building a bot which would be able to play the game, Rock-Paper-Scissors with other bots by predicting their next moves and making a move accordingly. The project was in the form of a code written in python, by the application of a neural network model. With the use of “numpy”, “tensorflow” and “keras”, this was made possible.

The neural network was to be given a dataset of different bots playing each other. From this dataset, it was trained to learn how the game is played, how different bots play with each other and which move to play in order to get the maximum wins.

Statement of Ethical Compliance

I hereby state that the data has not been taken from any human or animal resource nor does it involve any human participation. The ethical compliance for this project falls under the category of A0.

**Table of Content**

**1. List of Figures iv**

**2. Chapter 1:** Introduction **1**

**3. Chapter 2:**

**Introduction**:

**Design and Implementation**:

The bot that was to be created, should be able to play other bots through a certain contest which is available on the website, [www.rpscontest.com](http://www.rpscontest.com). The contest is a python code under the name “rpsrunner.py”. This contest code is copywrite free and can be used and modified under the condition that, the copywrite notice is to be included in all copies of the software.

Originally, the contest code was written in Python-2 and was converted to the latest Python-3 programming language. The design of this contest code is such that, it takes the variable “output” from both bots and uses it to run the R-P-S contest. Any deviation from this would result in the disqualification of the bot. So, this was to be taken into consideration while building the bot.

**Creation of the First Bot:**

To create a bot which was to learn from the moves made by other bots, the first step was to conduct the R-P-S game with multiple bots and create a datafile which would store the outcomes. The bot which was to created incorporates a Long-Short Term Memory neural network model, which would learn from the outcomes of this dataset and can be trained to predict the next move of the bot that it plays the game with.

Initially, the data that was gathered displayed the names of the bots playing, the score calculated after playing and the time taken for the bots to play and make a move. If the score is positive, then bot1(the first that we entered) wins. If it is negative, that means bot2(the second bot that we entered) wins. When writing the code initially, this data was gathered and used to create a neural network model. This data was then converted to a “comma separated values” (CSV), so that it would be easier for the model to be trained.

After the data was collected, the model was created to train on it.

Another file was created which could interact with the contest runner code and would be able to play the game. This file was linked to the neural network model that was written previously. The model was then trained with the datafile provided with 1000 iterations. With the model trained and linked with the actual bot file, it was ready to play the game. Here is where the first problem arose. As the data that was derived from the running the contest displayed only the names of the bots, the score and the time taken to run each round, there was no scope to learn how to play other bots and predict their next moves. This led to the disqualification of the bot that was created. After consulting the project guides, who pointed out the flaws in the logic, a total rework was necessary in order to successfully play the game.

**Creation of the Second Bot**:

The general idea of creating two files, one for the actual model and the other to interact with the game was intact. The changes made was in the data being used, the logic behind writing the bot and the interaction between the bot and the runner code for the game.

For the neural network, four dictionaries were created. Two of them represent the moves that can be made and the value they were attached to. The first one dictionary was to encode the moves that can be made, for which the keys were the moves that can be made, that is, R(Rock), P(Paper) and S(Scissors) and the values were 0, 1, and 2 respectively. This was done because machine learning models need numerical data to be trained. The second dictionary was to decode the moves, for which the keys and values were switched, that is, the keys were 0, 1 and 2 while the values were R, P and S. As the model outputs a numerical value, we need to decode it so that it can be used to play the game.

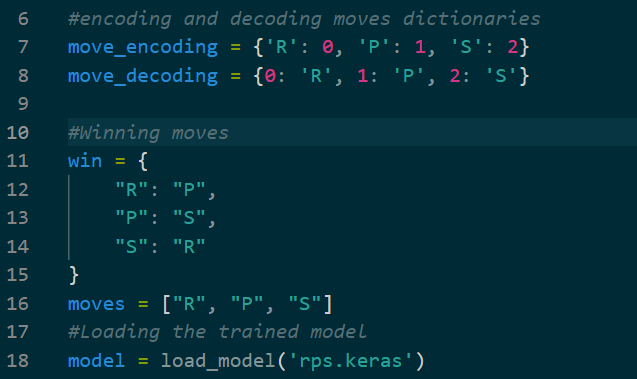
The third dictionary that was created represents the moves that need to be played against the opponent so the bot would be able to win. The keys represent a move and the values represent the move that beats it. So, with the key as R, the value would be P; with the key as P, the value would be S; with the key as S, the value would be R. The fourth and final dictionary was created so that, if there is a chance that the bot loses, instead of playing a losing move, it can play the same move as the opponent. This was done so that the game could be tied instead of the bot losing.

Afterward, the runner code was tweaked so that the output of the game, was a collection of tuples that represented the moves played by the opponents and their next move. This was possible due to the runner code being open source. Two lists were initialized to represent the histories of both the bot and the opponent. The data compiled was prepared for training.

The model was then created with many factors which played their part. First, the Keras API from TensorFlow was used to build the model. From this Keras API, the “Sequential” model was adopted as it is a layer-based model. This will be useful in identifying patterns and predicting the opponent’s moves. Then a Long Short-Term Memory (LSTM) layer with 50 neurons was added. The input shape of (5,1) was added to this layer. Each input sequence is of length 5 and each item of the sequence contains only one feature. This means that each sequence has 5 moves that have been made by the opponent and each move has only one feature i.e. the encoded move (0 for R, 1 for P, or 2 for S).

The next step was to add a fully connected layer (Dense) with 50 neurons. Rectified Linear Unit(ReLU) was used as the activation function. This helps with learning complex patterns. The output layer was added next. This is also a Dense layer but with 3 units as its output. These 3 units represent the three different classes(moves) that the bot can make. Here Softmax is used as the activation function. This will convert raw outputs into probabilities helping in predicting the move with the highest probabilities.

The training consisted of 150 epochs, a batch size of 32, a validation split of 0.1 and a verbose of 2 . And then the model was saved.

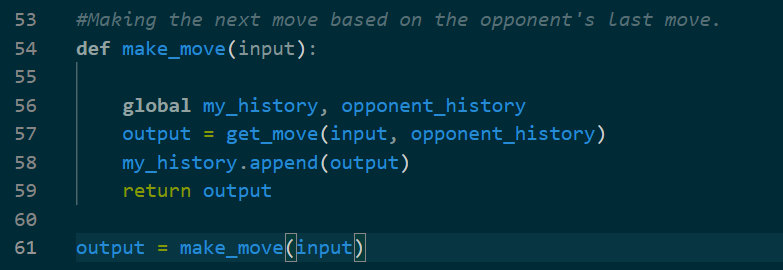
Next, the bot code file was created to interact with the runner code. This code consisted of the move encoding and decoding dictionaries from before and the winning moves dictionary. A list named moves was initialized which contained all three possible moves that the bot could make. The model that we saved before was then loaded to the file. Again, the histories of both the bot and the opponent were initialized.

Figure

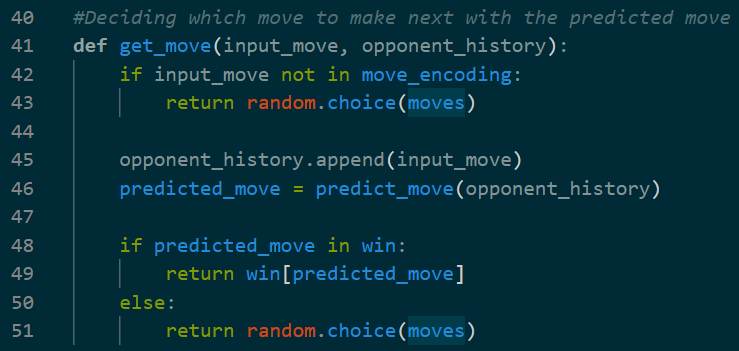
The code snippet above shows the encoded and decoded moves dictionaries along with the game strategy and moves. It also describes how the model was loaded to this file and the initialization of the bot and opponent’s histories.

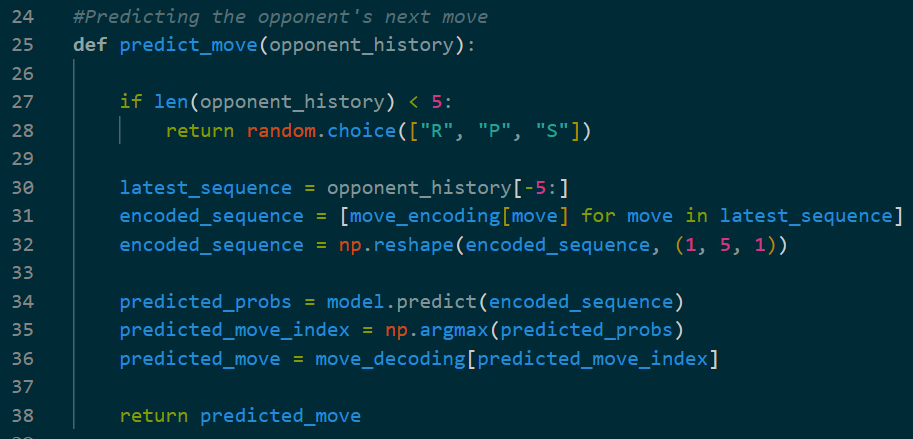
This file contains three critical functions. Their functioning depends on certain factors provided by the runner code. The runner code provides an input variable to the bot, which is the opponent’s most recent move, and receives the variable output with which it can run it’s game. A significant issue arose during the initial stages of development, where incorrect definition and usage of the bot’s input and output led to disqualification. This problem was eventually resolved by reviewing the code of other participants, ensuring the bot functioned correctly within the game's framework. The snippets to each function follows the description.

The functions in this file serve three primary purposes. The first function, make\_move, is responsible for generating the bot’s move. It is invoked by the output variable, which receives the input (the opponent’s last move). Within this function, both the bot's and the opponent’s histories are declared as global variables to maintain consistency across different rounds. The make\_move function then passes the opponent’s latest move and their move history to the next function, get\_move. The code snippet is shown below.



The second function, get\_move, involves several steps. It begins by checking whether the input is a valid move by verifying it against the keys in the encoding dictionary. If the input is not valid, the function returns a random move to the make\_move function. Once a valid input is detected, the opponent’s move is added to their history. The function then calls the third function, predict\_move, passing the updated opponent history as an argument, which is stored in the variable predicted\_move.



Finally, the predict\_move function uses the opponent’s history to forecast their next move. If the history is too short (fewer than five moves), it returns a random move. Otherwise, it encodes the most recent sequence of moves into a format suitable for the neural network model. The model predicts the opponent’s next move based on this encoded sequence, and the predicted move is returned to the get\_move function for further processing.

If this predicted value exists within the dictionary, which contains the winning and tie moves, it will choose the either the value which either allows the bot to win or at least tie the round, at random. It will return this value to the make\_moves function. If the predicted value doesn’t exist in the dictionary, it will return a random value from the moves list. In either scenario, the returned value would be stored in an output variable. The bot history list is then appended with this value and the output variable is passed back to the runner code.

**Testing and Evaluation**:

After the creation of the bot, there needed to be a way to test it separately without directly using it in the game so that, any errors in the code could be identified and corrected. A small piece of code was implemented that would pass a set of inputs, used in place of an actual opponent bot. Here it was observed that 70% of the test cases were seen to have run successfully. To increase the accuracy, the size of the dataset was increased and on which the model was trained. This led to a significant increase in the accuracy. Along with this, the number of epochs was also increased to train the model longer.

After seeing the rise in accuracy, the actual game was run to evaluate the running of the game. After playing multiple bots it was observed that the bot had a 60% chance of winning, regardless of the complexity of the opponent bot. When played with the highest rated bot on the rpscontest website, it managed to win…

Thus, ensured that the bot would be able to learn and play against other bots.

**Project Ethics**:

This project deals with bot programs playing Rock Paper Scissors and the data is only derived from these bots playing with each other and also does not involve any human interaction, the ethical guidance does not apply.

**Conclusion**:

**BCS Project Criteria and Self-Reflection**:

This project displays all six outcomes which are expected by the Chartered Intstitute of IT.

1. “The project displays an ability to apply practical and analytical skills gained during the degree programme as discussed in the Design and Implementation part of this paper.”

2. “The project displays Innovation and/or creativity which can be seen from the Design and Implementation part of this paper.”

3. “The project synthesises information, ideas and practices to provide a quality solution together with an evaluation of that solution.”

4. “This project meets a real need in a wider context.”

The first three outcomes are described in the Design and Implementation section of this paper. The fourth outcome is described in the introduction section of this paper.

Coming to the fifth outcome which is the “Ability to self-manage a significant piece of work”, I was able to complete each part of the project within time and managed to meet each deadline of the each assessment, but this did come with a few difficulties.

This leads to the sixth outcome which is the “Critical self-evaluation of the process”.