*Name: Trung Kien Nguyen*

*Student ID: 104053642*



COS30002

Artificial Intelligence for Games

REPORT

Custom Program – Waning Crescent Chess



***TABLE OF CONTENTS***

*Table of contents…………………………………………………………………………………………………….……….…2*

*Introduction…………………………………………………………………………………………………………………….…3*

*Overview of the program…………………………………………………………………………………………………….4*

*Program’s features Explanation………………………………………………………………………………………….8*

1. *Chess game implementing…………………………………………………………………………………….8*
2. *Data generating……………………………………………………………………………………………………8*
3. *Bots………………………………………………………………………………………………………………………9*

*Deep Neural Network Explanation………………………………………………………………………………………9*

1. *Overview of the Neural Network…………………………………………………………………………..9*
2. *Mechanism…………………………………………………………………………………………………………..9*

*Reflection…………………………………………………………………………………………………………………………11*

*Conclusion……………………………………………………………………………………………………………….……….11*

*References……………………………………………………………………………………………………………….….……11*

*Appendices……………………………………………………………………………………………………………………….12*

1. *FEN…………………………………………………………………………………………………………………….12*
2. *Activation Functions…………………………………………………………………………………………..12*
   1. *ReLU………………………………………………………………………………………………………….12*
   2. *Gaussian……………………………………………………………………………………………………13*
   3. *Sigmoid……………………………………………………………………………………………………..13*
3. *Derivatives in Backpropagation………………………………………………………………………….14*

**INTRODUCTION**

**Deep Learning Chess Bots using Neural Network built from scratch**

TRUNG KIEN NGUYEN – 104053642

Bachelor of Computer Science – AI major

Swinburne Unirversity of Technology – Hawthorn Campus – Semester 1 – Year 2024

*Artificial intelligence applications have been revolutionised by the advent of deep learning, especially in strategy-based games like chess. Innovative projects like Google's AlphaZero and IBM's Deep Blue have shown that AI is capable of surpassing human skill by using sophisticated algorithms to create novel strategies.*

*My project builds a neural network from the ground up to learn and play chess strategy in an effort to advance this subject. This project investigates how deep learning can be used to improve decision-making processes, by focusing on Deep Learning bots that acquires knowledge by analysing past game data, much to how humans learn. Furthermore, this aims to further AI's potential in strategic game theory and other areas while also advancing my understanding of neural networks in complex problem-solving contexts.*

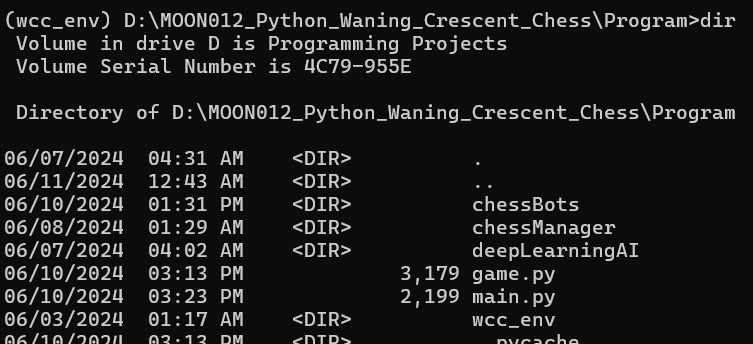
**OVERVIEW OF THE PROGRAM**

The project is built in Python, with the help of the GUI package of pyglet. Here is the instruction of how to execute it:

* Initially, we will need to set up the Python environment by importing the required libraries or packages for the program. It is recommended to use a separate environment (virtual environment) for this project, as it would maintain consistency across development environment, manage project-specific dependencies (libraries/packages) without disrupting other projects on the device.
  + From the root directory of the project in the Command Prompt, we can create a Python virtual environment using “venv” package:

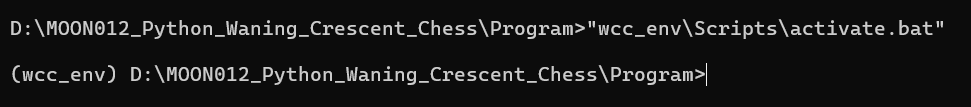


*Fig 1. Command to initialize a python virtual environment named “wcc\_env”*

**

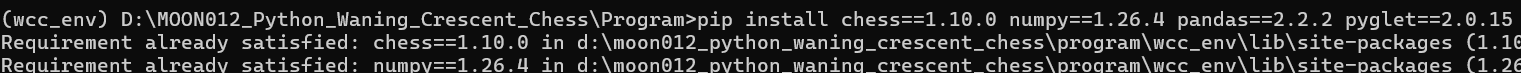
*Fig 2. A directory named “wcc\_env”, storing the dependencies of the virtual environment has been created*

* + We need to activate the virtual environment by running the file “activate.bat” in that directory.

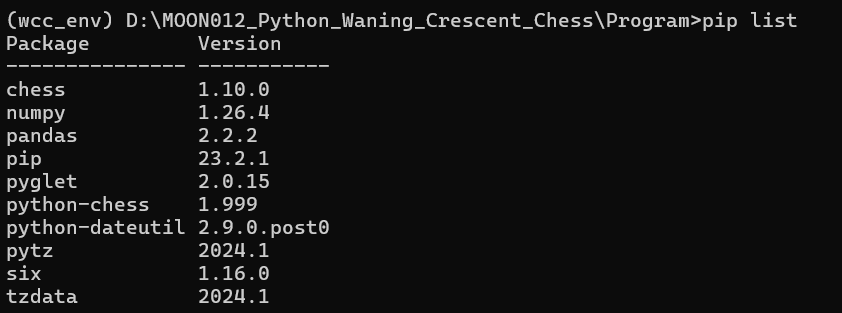


*Fig 3. Command to activate the virtual environment*

* Then, we will import all the dependencies of the program with “pip”, including: pyglet, python-chess, numpy, and pandas



*Fig 4. Command to import necessary packages*



*Fig 5. All the necessary packages has been imported*

* With the program, user can perform three different tasks:
  + Generating Data by simulating one or more matches between different kinds of bots (e.g. Random, DeepLearning). Every state of the board in all matches will be stored in a new CSV file in the folder “Program\deepLearningAI\data”

Command: “python main.py generateData <white\_bot\_name> <black\_bot\_name> <number\_of\_matches>”

with:

* “white\_bot\_name”, “black\_bot\_name”: This is the type of bots used for generating the data:
  + “Random” for RandomBot
  + “DeepLearning\_<model\_file\_name>” for DeepLearningBot. The neural network used for this bot must be trained previously.
* “number\_of\_matches”: This is the number of matches (simulations) that the two bots will play.

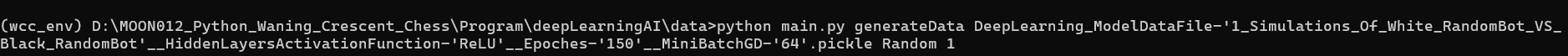
E.g.

1. Generate Data by simulating 2 matches between Random bots

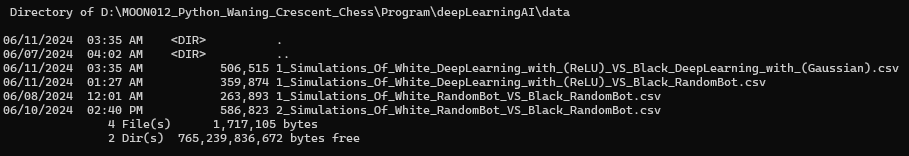


*Fig 6. Command to generate Data for example (a)*

1. Generate Data by simulating 1 match between a Deep Learning bot (with the model in the file “DeepLearning\_ModelDataFile-'1\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot'\_\_HiddenLayersActivationFunction-'ReLU'\_\_Epoches-'150'\_\_MiniBatchGD-'64'.pickle”) and a Random bot



*Fig 7. Command to generate data for example (b)*



*Fig 8. Two CSV files storing the generated data in both examples have been created (“2\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot.csv” and “1\_Simulations\_Of\_White\_DeepLearning\_with\_(ReLU)\_VS\_Black\_RandomBot.csv”)*

* + Training the Deep Neural Network to create a DeepLearning bot: Create a instance of NeuralNetwork, then save it to a new file in the directory “Program\deepLearning\AI\data”

Command: “python main.py train <data\_file\_name> <activation\_function> <epochs\_number> <batch\_size>”

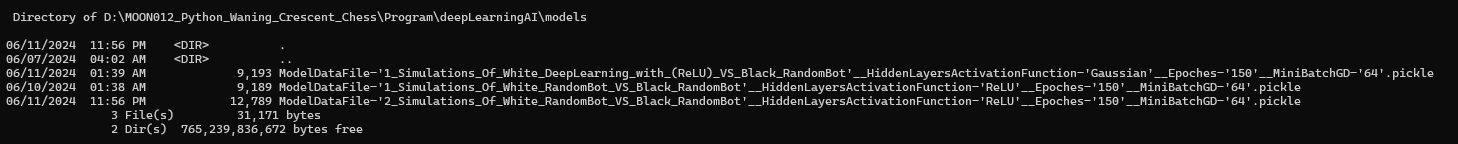
with:

* “data\_file\_name”: The CSV data file contains the board states’ information. There is no need to call for the file extension “.csv”, or the path of the file.
* “activation\_function”: The activation use in the hidden layer(s) of the neural network. It can be “ReLU”, “Sigmoid” or “Gaussian”.
* “epochs\_number”: This determine the number of epochs used in the training process.
* “batch\_size”: The number of training examples used to calculate the gradient during a single iteration of the training process. This should not be greater than the total number of samples in the training data file.

E.g.



*Fig 9. Train a Neural Network using the data in the file “2\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot.csv”*

**

*Fig 10. A model file has been created (“ModelDataFile-'2\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot'\_\_HiddenLayersActivationFunction-'ReLU'\_\_Epoches-'150'\_\_MiniBatchGD-'64'.pickle”)*

* + Play the game with DeepLearning bot:

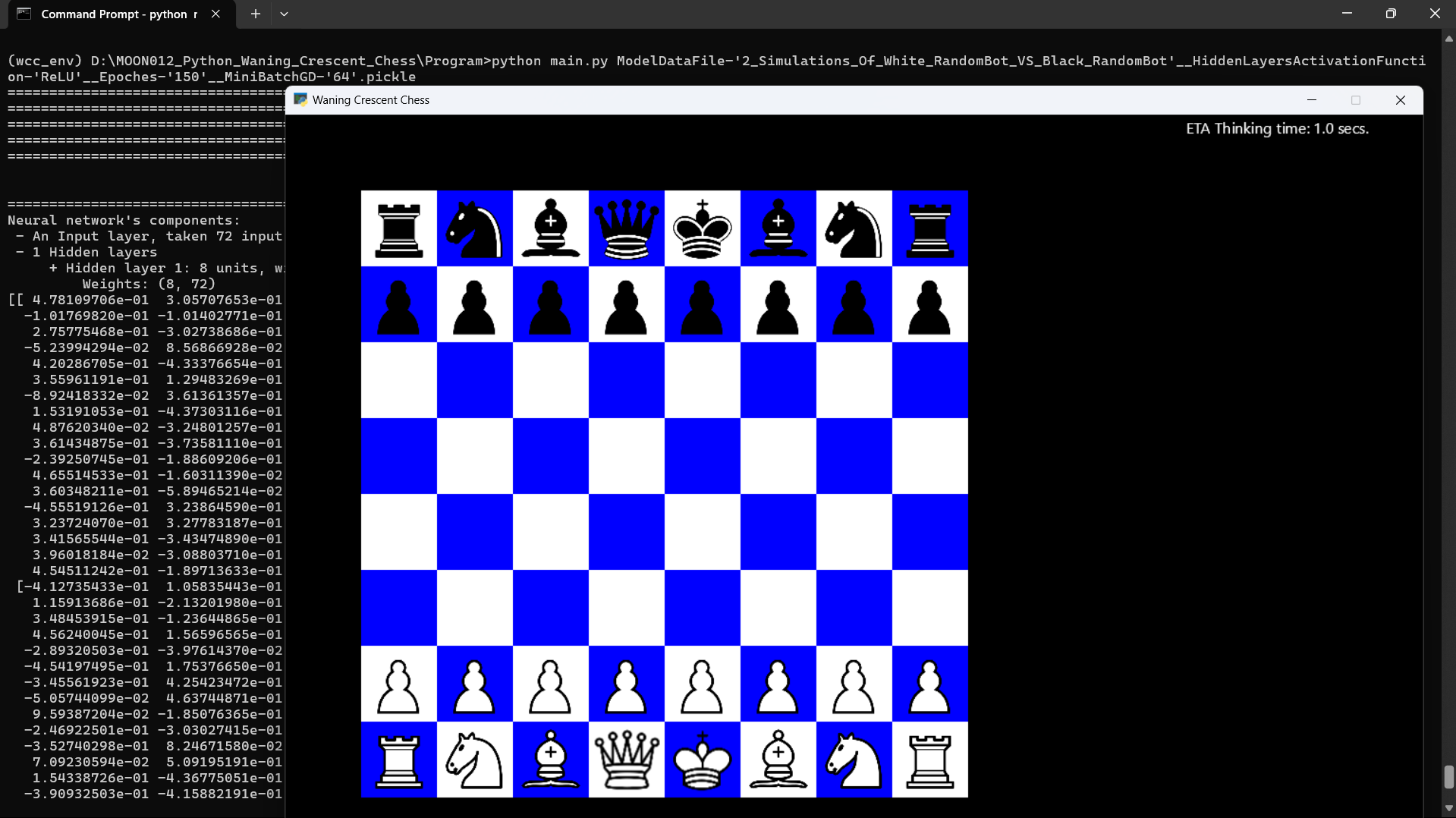
Command:

“python main.py” : Start the game to play with the default Deep Learning bot (trained by the model “ModelDataFile-'1\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot'\_\_HiddenLayersActivationFunction-'ReLU'\_\_Epoches-'150'\_\_MiniBatchGD-'64'.pickle”) or

“python main.py <model\_file\_name>”: Start the game playing with the Deep Learning bot using the specific model

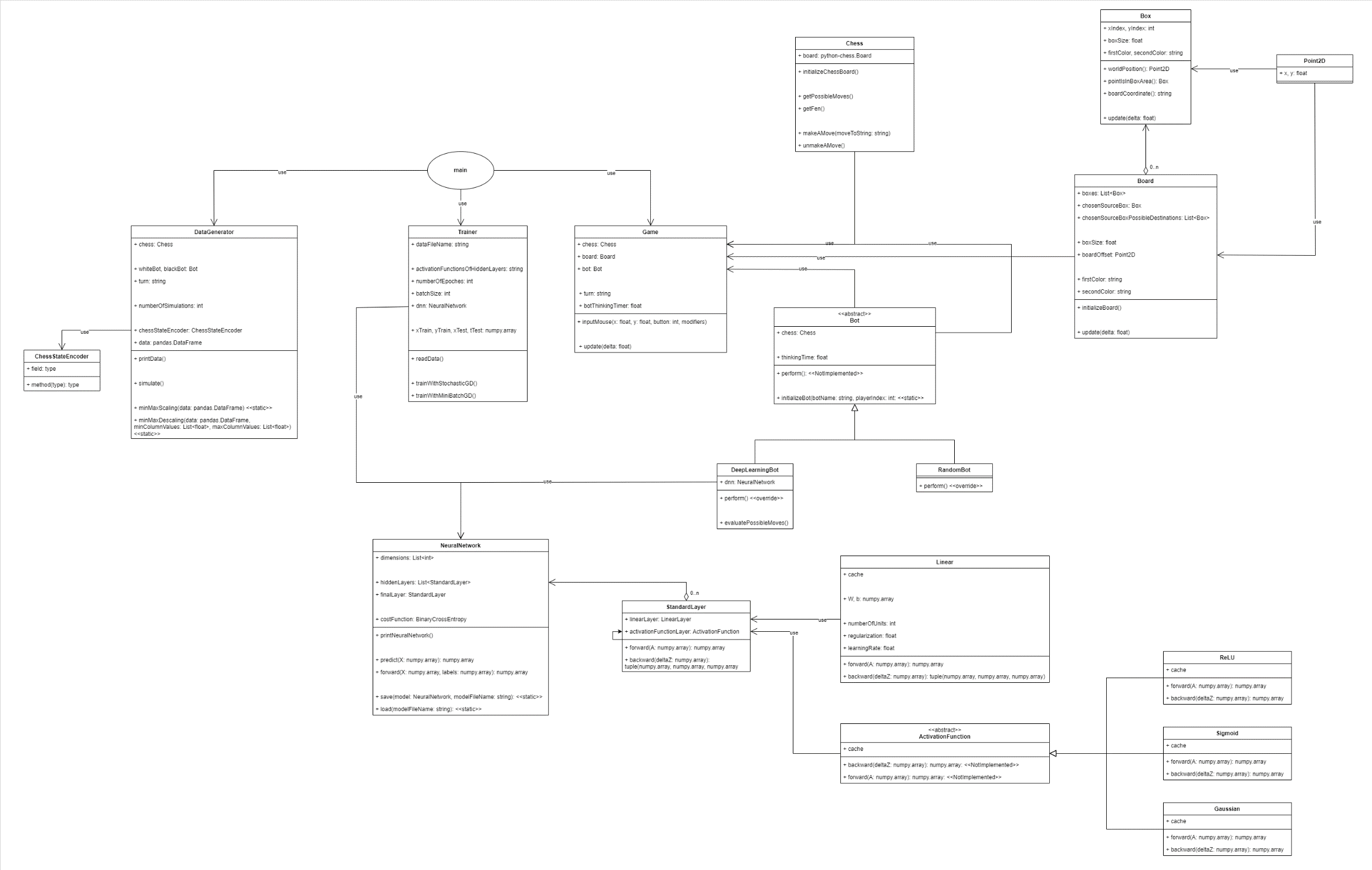
E.g. Play with the DeepLearning bot trained by the model

“ModelDataFile-'2\_Simulations\_Of\_White\_RandomBot\_VS\_Black\_RandomBot'\_\_HiddenLayersActivationFunction-'ReLU'\_\_Epoches-'150'\_\_MiniBatchGD-'64'.pickle”



*Fig 11. Game simulation*

This is the UML Class Diagram for the project:

**

*Fig 12: Detailed UML Class Diagram for the Custom Program*

**PROGRAM’S FEATURES EXPLANATION**

1. **Chess game implementing**

This project focuses on building a Deep Learning Chess bot, so implementing the complex mechanics of chess is simplified using the built-in [python-chess](https://python-chess.readthedocs.io/en/latest/) library.

This provides the tools for “move generation, move validation, and support for common formats”. I have created the Chess class (file “chess.py”) containing an attribute of python-chess.Board. With the library’s features, I can easily implement the functionalities such as initialize the chessboard, get and extract the FEN (see [Appendix 1](#Appendix_1)), make and unmake a move.

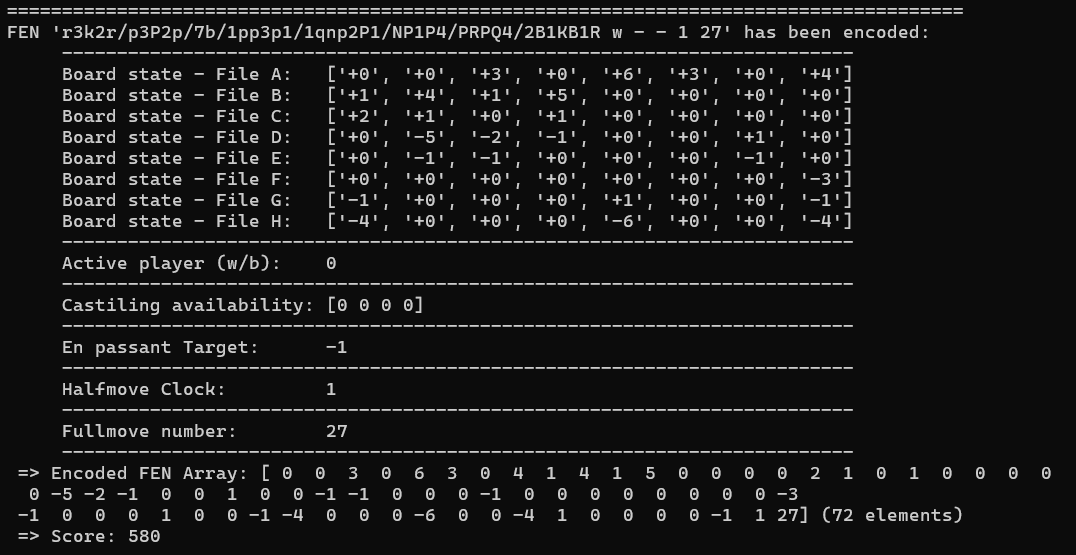
In addition, the Box and Board classes in the file “board.py”, Game class in the file “game.py”, along with the graphics module borrowed and updated from the one in Task 13 - Tactical Steering (Hiding), are responsible for illustrating the Chess Game (Board, Pieces, Move Indications, ...) in a separate Window (e.g. [Figure 11](#Figure_11))

1. **Generating Data**

To generate the data used in training the Deep Learning bot, I have created two classes in the file “dataGenerating.py”:

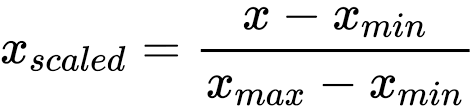
* ChessStateEncoder: This class is responsible for convert the FEN extraction to an array of 73 numeric elements:
  + The first 64 elements are the integers representing the squares’ state in the chessboard:
    - Its absolute value is the piece in a square (0 means none; 1, 2, 3, 4, 5, 6 mean Pawn, Knight, Bishop, Rook, Queen, King
    - Its sign means the piece’s color (positive for White and negative for Black)
  + The 65th element is the current active player (1 for White and -1 for Black)
  + The 66th – 69th elements are the castling availability for White in king’s, queen’s side, Black in king’s, queen’s side respectively.
  + The 70th, 71st, 72nd elements are the index of En passant Target Square, Halfmove Clock and Fullmove number respectively (see [Appendix 1](#Appendix_1))
  + The 73rd element is the Stockfish’s evaluation (Stockfish, downloaded from <https://stockfishchess.org/>, can evaluate which player is “winning”):
    - The sign of the index indicates which player is “winning” (again, positive for White and negative for Black)
    - The higher its absolute value is, the more likely the coresponding player is to win.

E.g.

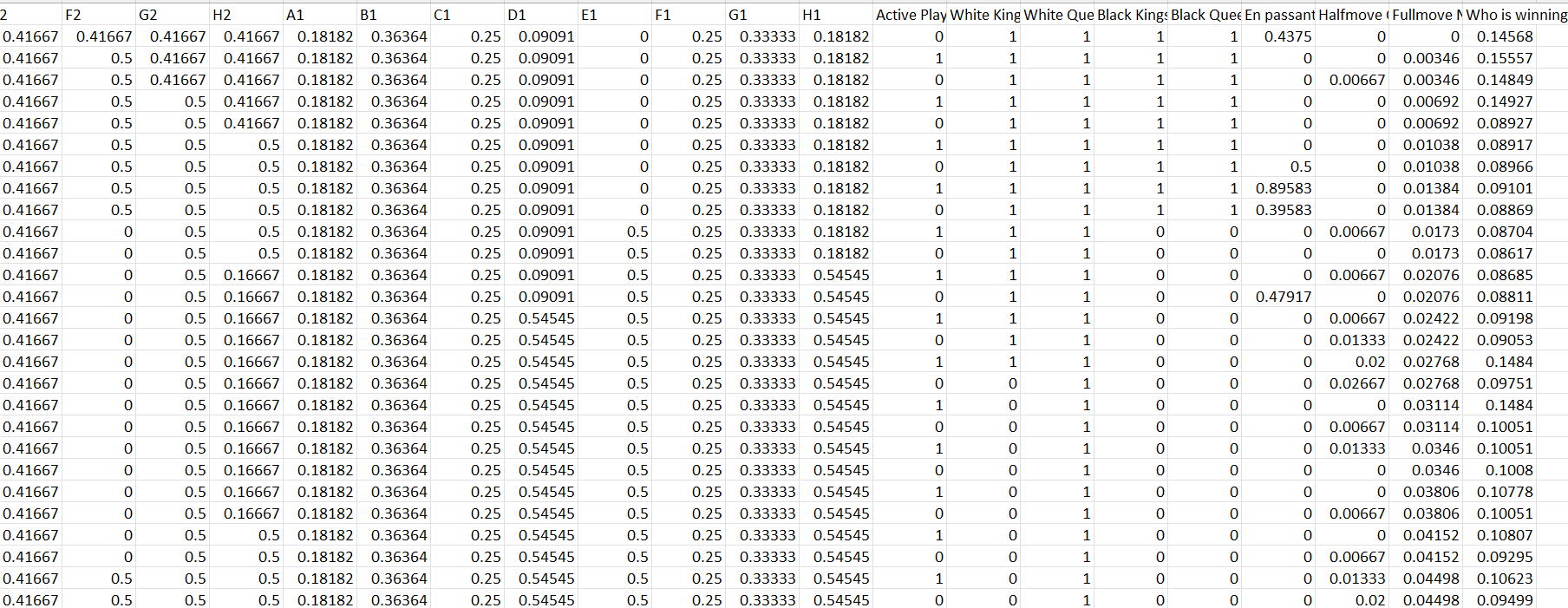


*Fig 13. Example of an chessboard state encoding*

* DataGenerator:
  + This class is in charge of generating the data using the ChessStateEncoder, by simulating matches. The number of matches, types of Bots used as White and Black players, are the parammeters of this class.
  + This class also support the scaling functionality by the Min-Max formula:



*Fig 14. Min-Max Scaler formula*

**

*Fig 15. Part of the scaled data has been saved*

1. **Bots**

Each type of bots is represented by a Class. In the version for Custom Program, there is a total of two bots:

* RandomBot (in the module “randomBot.py”): This bot simply uses methods in Chess class, to take a list of its possible moves, then choose a random one and perform it.
* DeepLearningBot (in the model “deepLearningBot.py”):
  + This bot import a Neural Network model previously saved in the “Program\deepLearningAI\models” directory. The name of that model is taken as a parameter of the class.
  + When it is its turn, the bot try to simulate all of its possible moves, using the Chess class’s functionalities of making and unmaking moves. It then uses the imported model to evaluate each possible move, and chooses the most beneficial move for it.

**DEEP NEURAL NETWORK FOR CHESS BOT**

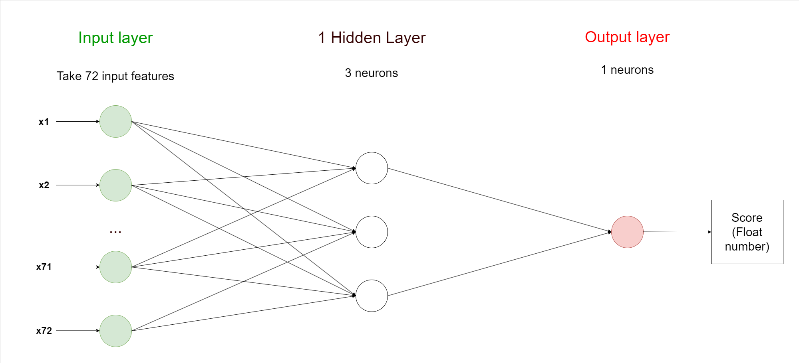
1. **Overview of a Neural Network**

A fundamental Deep neural network is made up of:

* **Layers**:
  + An **input layer**: Receives input data. For a simple neural network, data must often be preprocessed in the form of numeric matrix(s).
  + **Hidden layer(s)**: One or more intermediate layers that handle data. These layers use mathematical operations to extract features from the input data.
  + An **output layer**: Final outputs (predictions/scores) are generated using the numeric features that the hidden layers have extracted.
* **Neurons**: Neurons are the basic units of each layer of the Neural Network. After receiving the numeric input, each neuron processes it by mathematics function(s), then **adds** the output to the input value of the neuron on the next layer.

The neural network which will be used for my chess bot, contains:

* Input layer receiving 72 input features, and
* A 3-neuron hidden layer, and
* An output layer with 1 neuron to predict whether “white” or “black” has been winning. After scaling, it is a positive float between 0 and 1. If the index is “near” 1, white is winning and vice versa.



*Fig 16. Diagram of Neural Network that will be used*

1. **Mechanism**
   1. **Input and output layers**

The dataset is generates (refer to section [Generating Data](#Generating_Data)) with 72 feature columns and 1 labels column. The neural network is fed with those first **72 input features** data, through the input layer with the size of 72; and generate a float between 0 and 1, pointing out which player is winning, through the 1-neuron output layer.

* 1. **Hidden Layer**

To solve a problem with quite simple input as above, we only need to use one hidden layer in the network. The size (neurons number) of a layer is usually determined by **the square root of the multiplication** of the sizes of the previous layer and the next layer. That means, the size of the hidden layer can be compute approximately by: 

* 1. **Forward Propagation**

A standard layer (hidden layers or output layer) is a combination of a linear layer and a activation function layer, providing the mathmatics functions to compute the output.

In the forward propagation phase, the output from a hidden layer becomes the input to the next layer in the neural network.

The prediction output of the network is generated by the output layer, which is a float number in the scope of chess problem.

* + 1. **Weighted Sum by Linear Layer**

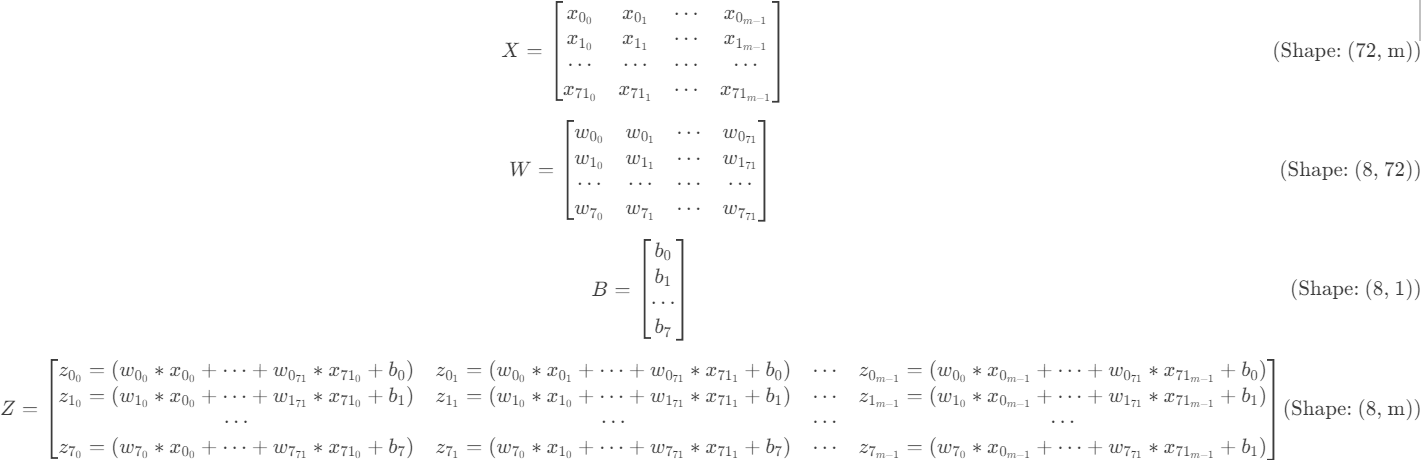
A linear layer is responsible to compute the output through a linear equation z = f(x) = w \* x + b, with:

* w is an element in the **weights** matrix set W
* b is an element in the **biases** array set B
* x is an element in the input set X
* z is an element in the output set Z

E.g. The 8-neuron Hidden layer, which taking the input with 72 features, outputing the data with 8 features:

* X\_hidden has the shape (72, m)
* W\_hidden has the shape (8, 72), to perform a matrix multiplication with X\_hidden:
* B\_hidden has the shape (8, 1), adding to the result of that matrix multiplication between W\_hidden and X\_hidden
* Z\_hidden has the shape (8, m)

(Given that m is the number of training samples of the dataset)



*Fig 17. Computations in the given example of Linear layer*

* + 1. **Activation Function Layer**

The weighted sum (Z) is then passed through an activation function, which help introduce non-linearity into the output of a neuron.

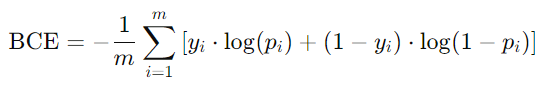
The activation functions used in the Neural Network includes ReLU (details in [Appendix 2a](#Appendix_2_ReLU)), Gaussian (details in [Appendix 2b](#Appendix_2_Gaussian)), and Softmax (details in [Appendix 2c](#Appendix_2_Sigmoid)).

Briefly, the Softmax activation generate a number between 0 and 1, which is perfectly match the requirement of the prediction output mentioned.

* + 1. **Cost**

After computing the predicted output (scores), we then compute the cost between the scores and the labels (actual values). The **primary purpose of the Neural Network is to minimize this cost**, so that the model becomes more “precise”. We can gain this by a process called backward propagation.

The output layer use Sigmoid activation function to compute a score in the extended binary form (a float between 0 and 1). There is the cost function that perfectly matches with this output, which is Binary Cross-Entropy (BCE)

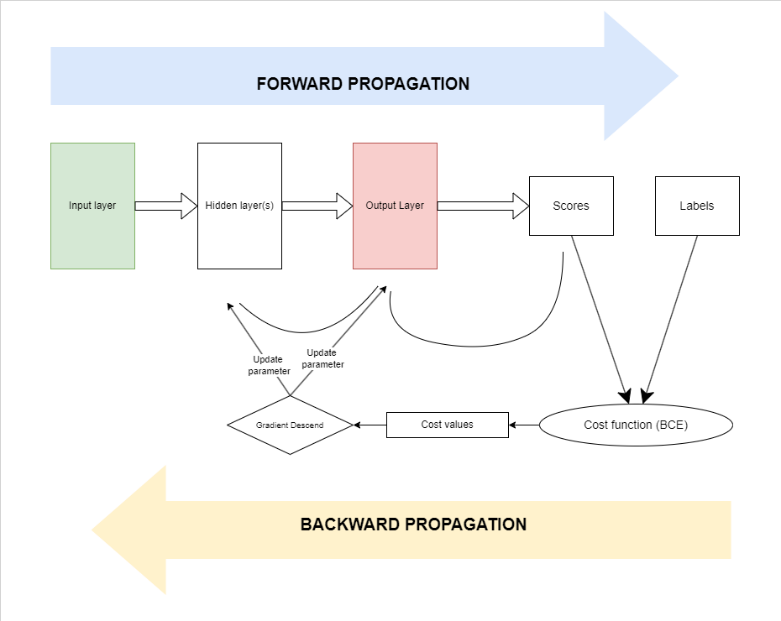


*Fig 18. Formula of Binary Cross Entropy*

where:

* m represents the number of samples.
* yi ​ is the actual label of the ith sample, which can be 0 or 1.
* 𝑝i is the predicted probability of the ith sample being class 1.
  1. **Backward propagation**

This is the other fundamental process in training neural networks, where the error is propagated back through the network by the **derivative functions** of the functions used, including linear function, activation functions (details derivatives computing in [Appendix 3](#Appendix_3_Derivatives)), orderly from the output layer to the input layer, allowing the cost to be minimized by **updating** **weights and biases** via optimization techniques of **Gradient Descent.**



*Fig 19. Backpropagation Summary*

**REFLECTION**

The following is this Project's Reflection for COS30002's Learning Outcomes Units:

1. **Discuss and implement software development techniques to support the creation of AI behaviour in games**

* The object-oriented principles are deeply applied when implementing the program:
  + Each class represents a single responsibility, and the total architecture has the suitable cohesion level. (Details in [Figure 12](#UML_Class_Diagram))
  + The extendability is also considered in the program. It is easier to add a new type of bots in the program other than the two provided (Add new sub-class of Bot)

1. **Understand and utilise a variety of graph and path planning techniques**

* The chess game is implemented as a grid-based environement (Board class), in which different agents may calculate to perform its move
* The Deep Learning bot of the program does not try to imitate the way Stockfish make prediction, but its predict values, based on the generated data, instead

1. **Create realistic movement for agents using steering force models**

* The chess game is basically a stationary game. The bots use the AI algorithm to predict the the next state of the board, instead of effective kinds of mobility. In other words, no steering force models is used in the program.

1. **Create agents that are capable of planning actions in order to achieve goals**

* The Deep Learning bot (agent) can used different kinds of Neural Network (in term of hidden layers’ activation function) to predict efficiently and effectively the next move.
* The prediction is not made simply based on one or more current chessboard’s state, but look at the whole chess game from the beginning, thanks to the functionalities provided by python-chess library

1. **Combine AI techniques to create more advanced game AI**

* A lot of Mathematics and Deep Learning techniques and algorithms are applied in training and evaluating the chess bot, including:
  + The use of a Neural Network
  + Forward Propagation, with matrices multiplication in linear equations, activation functions.
  + Backward Propagation, with derivatives computing, the Chain rule, Gradient Descend
  + Model’s regularization has been set up and ready to use (regularization of Linear class), but have not been used in the program.

Thank you for reading these words.

If you have any questions or comments about this program, including errors or misunderstandings, please inform me by emailing [104053642@student.swin.edu.au](mailto:104053642@student.swin.edu.au).

**REFERENCES**

*[1] “python-chess: a chess library for Python — python-chess 1.5.0 documentation,” python-chess.readthedocs.io.* [*https://python-chess.readthedocs.io/en/latest/*](https://python-chess.readthedocs.io/en/latest/)

*[2] “Home - Stockfish - Open Source Chess Engine,” stockfishchess.org.* [*https://stockfishchess.org/*](https://stockfishchess.org/)

*[3] “Building your Deep Neural Network: Step by Step” colab.research.google.com.* [*https://colab.research.google.com/drive/1Ok6VfH9\_y5zPFq5P5pmfLqf0dxirDaku#scrollTo=TGJuOi0pbUDS*](https://colab.research.google.com/drive/1Ok6VfH9_y5zPFq5P5pmfLqf0dxirDaku#scrollTo=TGJuOi0pbUDS)

*[4] NumPy, “NumPy Documentation,” numpy.org.* [*https://numpy.org/doc/*](https://numpy.org/doc/)

*[5] Pandas, “pandas documentation — pandas 1.0.1 documentation,” pandas.pydata.org, 2024.* [*https://pandas.pydata.org/docs/*](https://pandas.pydata.org/docs/)

*[6] OpenAI, “ChatGPT ,” chatgpt.com, Nov. 30, 2022.* [*https://chatgpt.com/*](https://chatgpt.com/)

**APPENDICES**

1. **Forsyth-Edwards Notation (FEN)**

General form of a FEN:

<Piece\_placement> <Active\_Player> <Castling\_Availability> <En\_Passant\_target\_square> <Halfmove\_clock> <Fullmove\_number>

E.g. “r1bqkb1r/pppp1Qpp/2n2n2/4p3/2B1P3/8/PPPP1PPP/RNB1K1NR b KQkq - 0 4”:

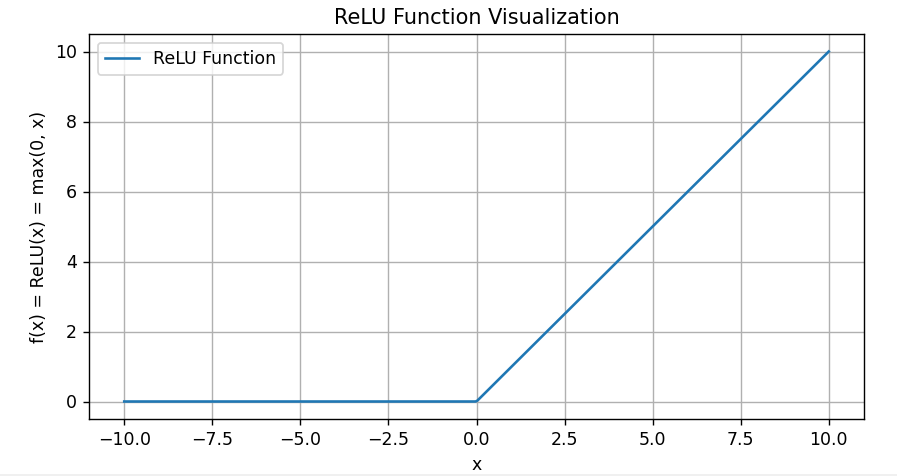
* <Piece\_placement> represents the **position of the pieces** in the chessboard:
  + “r”, “n”, “b”, “q”, “k”, and “p”: Black rook, knight, bishop, queen, king and pawn
  + “R”, “N”, “B”, “Q”, “K”, and “P”: They are similar but represents the white pieces
  + 1, 2, 3, 4, …: the number of consecutive squares horizontally.
* <Active\_player> is the **current active player**: “b” for black and “w” for white.
* <Castling\_availability>
  + “K”, “Q” indicate that White can castle on king’s, queen’s side respectively
  + “k”, “q”: They are similar, but for the Black player.
  + “-“ means no castling availability for both players
* <En\_Passant\_target\_square> represents the current possible **En passant** target square position in the chessboard:
  + If en passant is possible, this shows an integer from 0 to 63, indicating the order of a square in the chessboard (in order rank 1 to 8, file A to H)
  + “-“ means no en passant capture is possible in the next move
* <Halfmove\_clock> is an integer indicating the number of halfmoves since the last capture or pawn advance, which is relevant for the fifty-move rule.
* <Fullmove\_number> is the total number of full moves in the game. It starts at 1, and is incremented after each Black's move.

1. **Activation functions**
   1. **ReLU**

Rectified Linear Unit (ReLU) is one of the most popular activation functions employed in Neural Networks, especially in the hidden layers. Its formula is:



*Fig 20: Formula of ReLU*

**

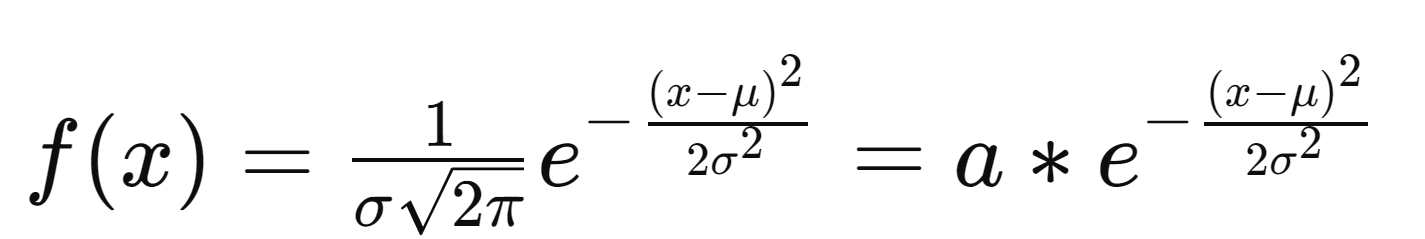
*Fig 21. Illustration of ReLU function graph. Generated by matplotlib packages.*

* Benefits of usage
  + ReLU's computing efficiency stems from its **simple formula** quickly computed mathematical processes.
  + By producing zero for every negative input. ReLU can **produce sparse output dataset**, which are useful in many deep learning models
  + Since the gradient (derivative function) for positive inputs is always 1, ReLU is very useful in mitigating the **vanishing gradient issue**, which is common in deep networks with saturating activation functions like sigmoid.
* Drawbacks of usage:
  + **Dying ReLU**: When a large number of neurons in a network **only output zeros for all inputs**. It does not contribute to the learning process in the network, which mean the model will always be inaccurate regardless of increasing the number of epochs or adjusting the batch size.

In the context of chess evaluation program, it is better **to use ReLU in the hidden layer(s)** as it is simple like the generated dataset, rather than a complecated one. However, the training process would have to be observed carefully to prevent any Dying issues happen.

* 1. **Gaussian**

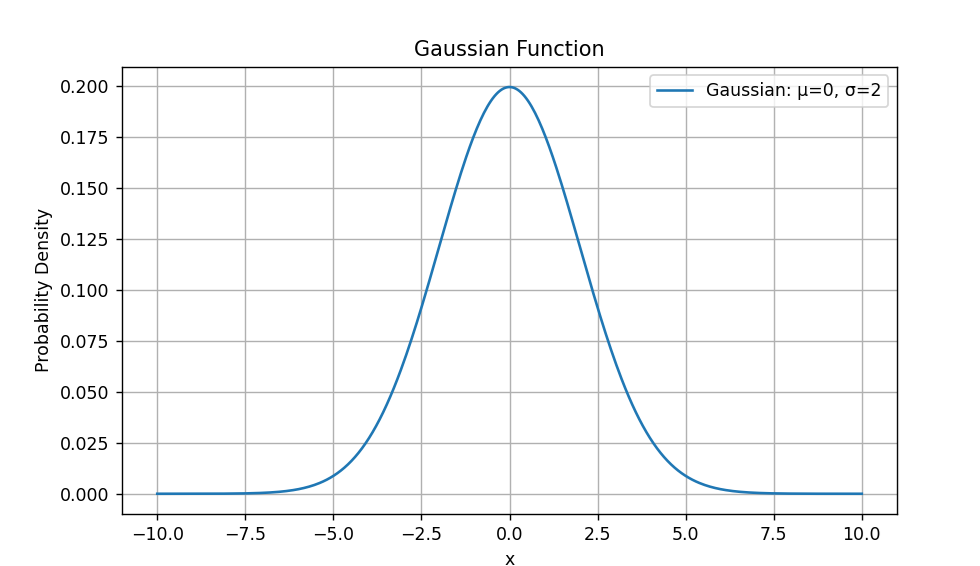
We will come to a more complex activation function – Gaussian. It depicts the symmetrical bell-shaped curve that peaks in the middle and falls off towards the tails. The Gaussian function's formula is given as:



*Fig 22: Formula of Gaussian*

where:

* μ is the mean
* σ is the standard deviation
* e is the base of the natural logarithm

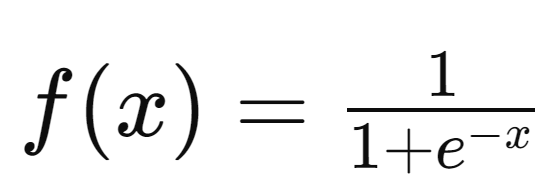


*Fig 23: Illustration of Gaussian function graph, with μ = 0 and σ = 2. Generated by matplotlib packages.*

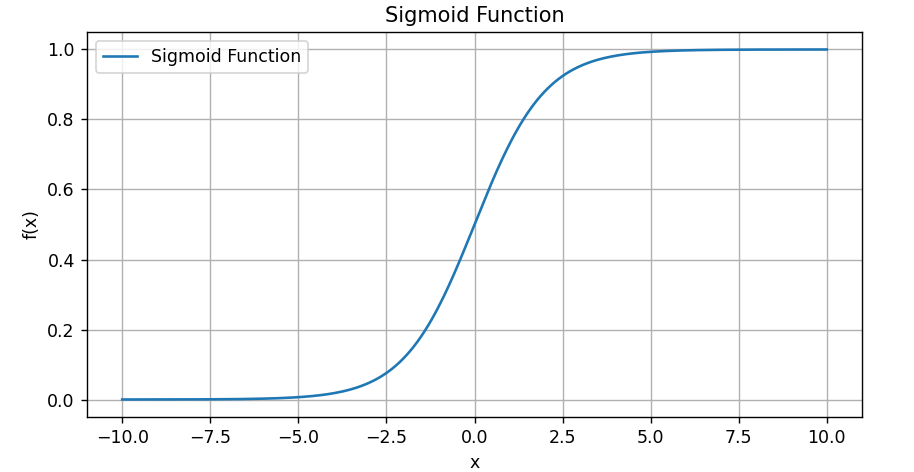
* Benefits of usage
  + **Central Limit Theorem**: The central limit theorem, which asserts that the sum of numerous independent, random variables with finite mean and variance will **roughly follow a normal distribution** regardless of the underlying distribution, is based on the Gaussian distribution. It is therefore very helpful in theory and statistical analysis.
  + Mathematical manipulation and analysis are made easier by the Gaussian function's smooth, **continuous shape and symmetry around the mean**. The Gaussian has certain qualities that facilitate its use in theoretical and computational contexts.
* Drawbacks of usage:
  + The Gaussian may not appropriately depict data distributions **if the mean and standard deviation are not the most pertinent or useful measures** (e.g., multimodal distributions) because it concentrates on these two metrics.
  + In situations when variables are interrelated in real life, the normal distribution may not be the most suitable model.
  1. **Sigmoid**

Finally, the activation function which is suitable for the output layer in the Chess Evaluation problem is Sigmoid.

Sigmoid, also known as the logistic function, is a mathematical function that has a typical sigmoid or "S"-shaped curve (see Figure 25).

****

*Fig 24. Formula of Sigmoid*

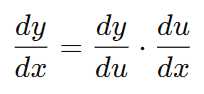
**

*Fig 25. Illustration of Sigmoid function graph. (Generated by matplotlib packages.)*

* Benefits of usage
  + The sigmoid function in logistic regression is used to simulate the likelihood that an input fits into a class with a label of "1". In this case, the sigmoid function's output can be understood as **the probability of a positive class** (white is winning, in the program), and the model is trained to reduce the discrepancy between the actual class labels and the anticipated probabilities.
* Drawbacks of usage:
  + **Vanishing gradient problems**, where the gradients (derivative function) becomes very small, can happen sometimes.

1. **Derivatives in Backward Propagation**
   1. **Computing Gradients**

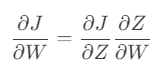
* The Chain rule:
  + Definition: In calculus, the chain rule is use to differentiate compositions of functions. Given two functions f and g, where y = g(u) = g(f(x)). The derivative of y with respect to x is given by:



*Fig 26. The chain rule formula*

* + Application in computing gradient of the cost with respect to different quantities in a Neural Network:
    - Compute gradient of the cost with respect to linear layer’s parameters – weights (∂J/ ∂W) and biases (∂J/ ∂b):

Using the Chain rule, let say that g is the cost function (J), f is the linear function, we have:



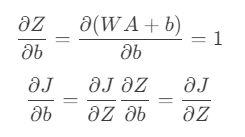
And, the linear equation state that Z = W \* A + b, we can calculate the gradient of the output Z with respect to the Weights (∂Z/∂W):



From those two, we can compute the gradient of the cost with respect to the Weights:

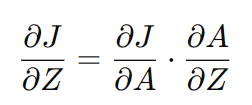


Similarly, we can compute the gradient of the cost with respect to the Biases:



* + - Now, let compute gradient of the cost with respect to output of linear layer: ∂J/∂Z (a.k.a input of corresponding activation function layer)

Given that A = σ(Z) is the output of the activation function layer. Using the chain rule, we have:



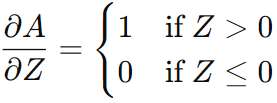
where ∂J / ∂A is the gradient of the cost J with respect to A. We will first compute the ∂A / ∂Z, which is the gradient of the output of activation function layer with respect to its input:

* + - * ReLU:

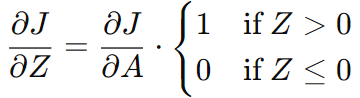
The ReLU is defined as:



So, it’s easy to define its derivative:



Thus, the gradient ∂A / ∂Z for ReLU becomes:



* + - * Gaussian:

The Gaussian’s formula is:



From that, its derivative can be calculated:



Thus, the gradient ∂A / ∂Z for Gaussian becomes:



* + - * Sigmoid:

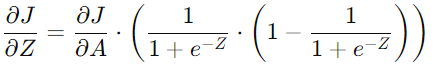
From the Sigmoid formula:



Its derivative can be calculated:



Thus, the gradient ∂A / ∂Z for Sigmoid becomes:

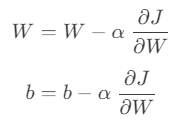


* 1. **Gradient Descend**

With the Chain Rule’s applications above, it is possible to compute the gradient of the cost with respect to every layer’s parameter (weights and biases).

The Gradient Descend state that, in order for a function y = f(x) to achieve **a minimum** through **adjusting** x, we can reduce the value of x by **an amount proportional** to the corresponding gradient value.

That means, we can update weights and biases in each layer of the network so that the cost is minimized:



where α, a very small float, is defined as the **learning rate** of the corresponding layer