GIOVANI PIGNATON DA COSTA

MACHINE LEARNING METHODS APPLIED TO THE DOTS AND BOXES BOARD GAME

UNIVERSITY OF ALGARVE
Faculty of Sciences and Technology
2022

GIOVANI PIGNATON DA COSTA

MACHINE LEARNING METHODS APPLIED TO THE DOTS AND BOXES BOARD GAME

Master in Informatics Engineering
Faculty of Sciences and Technology

Advisor:
Prof. Dr. José Valente de Oliveira

UNIVERSITY OF ALGARVE
Faculty of Sciences and Technology
2022

MACHINE LEARNING METHODS APPLIED TO THE DOTS AND BOXES BOARD GAME

Declaration of authorship of work

I hereby declare to be the author of this work, which is original and unpublished. Authors and works consulted are properly cited in the text and included in the reference list.

(Giovani Pignaton da Costa)

Resumo

Pontos e Quadrados (Dots and Boxes na versão anglo-saxónica) é um jogo clássico de tabuleiro no qual os jogadores unem quatro pontos próximos numa grelha para criar o maior número possível de quadrados. Este trabalho irá inverstigar técnicas de aprendizagem profunda e aprendizagem por reforço, que torna possível um programa de computador aprender como jogar o jogo, sem nenhuma interação humana, e aplicar o mesmo ao jogo Dots and Boxes; a abordagem usada no DeepMind AlphaZero será analisada. O AlphaZero combina uma rede neural convolucional e o algoritmo Monte Carlo Tree Search para alcançar um desempenho super humano, sem conhecimento prévio, em jogos como o Xadrez, Go, e Shogi.

Os resultados obtidos permitem aferir sobre a adequação da abordagem ao jogo Pontos e Quadrados.

Palavras Chave: adversarial search, machine learning, deep learning, reinforcement learning, Dots and Boxes, rede neural artificial, rede neural convolucional, jogos, AlphaZero, DeepMind, jogos de tabuleiro, auto aprendizado.

Abstract

Dots and Boxes is a classical board game in which players connect four nearest dots in a grid to create the maximum possible number of boxes. This work will investigate deep learning techniques with reinforcement learning to make possible a computer program to learn how to play the game, without human interaction, and apply it to the Dots and Boxes board game; the approach beyond DeepMind AlphaZero being taken as the approach to follow. AlphaZero makes a connection between a Convolutional Neural Network and the Monte Carlo Tree Search algorithm to achieve superhuman performance, starting from no a priori knowledge in games such as Chess, Go, and Shogi.

The results obtained allow to measure the approach adequacy to the game Dots and Boxes.

Keywords: adversarial search, machine learning, deep learning, reinforcement learning, Dots and Boxes, artificial neural network, convolutional neural network, game playing, AlphaZero, DeepMind, board game, self-playing.

To my family

always by my side!

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Networks.
AWS	Amazon Web Services.
Colab	Google Colaboratory.
CNN	Convolutional Neural Network.
CNTK	Microsoft Cognitive Toolkit.
CPU	Central Processing Unit.
CPUCT	Constant PUCT.
GPU	Graphics Processing Unit.
GUID	Global Unique Identifier.
IDE	Integrated Development Environment.
JSON	JavaScript Object Notation.
LR	Learning Rate.
MCTS	Monte Carlo Tree Search.
ML	Machine Learning.
ML.Net	Machine Learning for .Net.
MSE	Mean Squared Error.
MS-DOS	Microsoft Disk Operating System.
PC	Personal Computer.
PUCT	Polynomial Upper Confidence Trees.
RAD	Rapid Application Development.

ReLU	Rectified Linear Unit Function.
UCB	Upper Confidence Bound.
UCT	Upper Confidence bound applied to Trees.
UI	User Interface.
VM	Virtual Machine.

1

Introduction

Board games are amazing in many ways. Great challenges are disputed all over the world to promote the best-recognized players with their best strategies in many kinds of board games like chess, go, backgammon, and many others. Not only in the world arena those games are played, but also in small groups, family homes, and now, most of the time, on a computer or on a cell phone.

The game migration from elegant boards to different computational platforms creates a variety of possibilities to train and develop new abilities in real players. Through all kinds of competitive ways, and the improvement of the capabilities and difficulty in games, it is possible to create the best real challenge for humans. Learning the moves from the best players in the world and combining them with machine learning techniques or through only from machine learning techniques, we can create the best ma-

chine opponent, even for humans. However, in the new machine world, a machine "fights" against a machine to become the new best player, without previous data. This is what happens with AlphaZero which is able to achieve a superhuman performance, winning all possible played games and smashing all of the opponents in games such as Chess, GO, or Shogi (Silver D., 2017).

1.1 Motivation

When I was fourteen, my dad brought me a new machine, a recent 80286 PC (personal computer)! I was only familiarized with video games, and now, that computer will become my new best friend forever! My dad enrolled me in a programming course and I started to learn how to program in the Clipper Summer '86 version, an old language for MS-DOS. I did some good things in that language and years later, I learned a new language, an object programming oriented language, Object Pascal.

Delphi[1] opened to me a new world for windows programming and I created my first program in Delphi: The Dots and Boxes game. Dots and Boxes was my classical game in school, for many times my friends and I were playing that game with a pencil and a blank sheet, good times!

The game was created but with only one issue: my new PC could not play it! I could not teach it to play the game at that moment. Now, in my Master, after some machine learning classes, the curiosity of putting some piece of "intelligence" in that software comes out and my old Delphi Dots and Boxes becomes a completely rewritten new console mode game, written in C# Core. The new game was created with the intent to be a smart game.

Monte Carlo Tree Search (MCTS) was the first algorithm implemented and attached to the new game and the result was pretty good, the computer could play for himself without no instructions, only following the game rules. However, some mistakes quickly appeared in a lot of games, in other words, it "learned" great moves but it

[1]"Delphi is a cross-plataform integrated development environment (IDE) [for object pascal] that supports rapid application development (RAD) for the most operating systems."(Gabrijelčič, 2019)

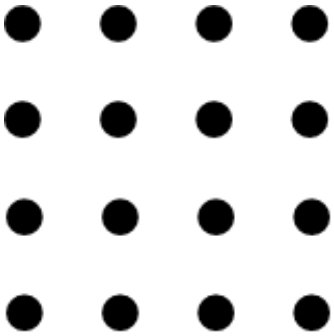


Figure 1.1: Dots and Boxes 3x3 - empty board

"forgot" all when the was ended.

The project could be smarter, other techniques could be explored, and that was the main driving force of this current work.

My childhood's personal motivation was not the only reason for starting this project. Others reasons would be learning, in a practical way, how to use reinforcement learning through artificial neural networks, how could it be done in the learning process for that game? This project is of great value to me in what concerns to know how things work behind the scenes in a game learning process.

1.2 Dots and Boxes

Dots and Boxes is a board game that can be played at least by two players. The board contains equidistant dots, horizontally and vertically positioned. The number of dots on horizontal and vertical does not matter as along as the number of dots in all columns have to be equals to the number of dots in all rows. However, in this implementation of the game, it will be used only Dots and Boxes boards with the same number of dots for the columns and rows, shaping a big square. The number of maximum boxes vertically and horizontally would be represented by 'n' (figures 1.1, 1.2).

By convention, in this work, lines, row, and column words are used to define the line created by the user for a gameplay move and columns and rows are used to define the position of that line in the board.

For example, the board 3 x 3 (n = 3) is formed by 4 small dots per row and 4 small

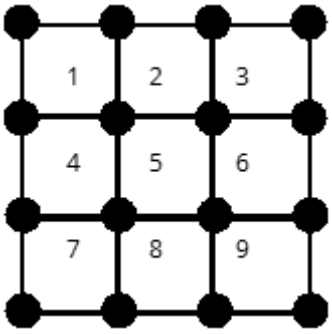


Figure 1.2: Dots and Boxes 3x3 - full board

dots per column and the connection of those dots through horizontal and vertical lines builds 9 squares (boxes). The number of lines connecting dots (or total moves of the game) is 24 for the same configuration. There are 3 small vertical lines per column, 4 small vertical lines per row, 4 small horizontal lines per column and 3 small horizontal lines per row. It could be represented by the following formulas 1.1:

Board n x n

$$n^2 = \text{total number of boxes}$$

$$m = n + 1$$

$$\text{total of horizontal moves} = mn$$

$$\text{total of vertical moves} = nm$$

$$\text{total of possible moves} = 2mn \text{ (horizontal + vertical)} \tag{1.1}$$

The game playing sequence consists of connecting two nearest dots, vertically or horizontally, by a line. Diagonal lines are not permitted. It is not important or necessary to keep any information about the player who drew the line. Each player can draw only one line in any empty space between two dots in each turn. The exception here is when a player creates a box after drawing the line, in this case, he or she writes the name or a mark in the middle of the box, to identify the box owner (figure 1.3), receives one point for box built, and plays again (mandatory). In his(her)(its) turn a

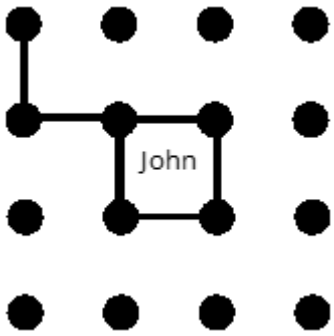


Figure 1.3: John's game point

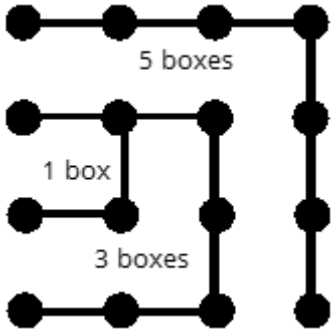


Figure 1.4: Paths

line must be drawn, it is mandatory, so the player cannot pass or skip the turn and continues while boxes are created.

The goal is to create the most number of boxes as possible connecting dots. It does not matter the number of lines drawn for each player, but wins the game, the player that could create the most number of boxes when no more lines can be drawn. The game is draw when both players finished the game with the same number of points. Some board configurations do not permit a draw game, like boards with odds N's (3, 5, 7, 9,...); for even N's (4, 6, 8,...), the game can be a draw game.

In this game, strategies can be used, and the creation of many boxes at the beginning does not give "a won game" to the player at the end. Some strategies are to create a path of boxes (figure 1.4), that so many points can be made in sequence. Many times, giving a point or two to an adversary became a strategy to earn a path with many points at the end. This game strategy is particularly important to defeat the opponent. Others games like Checkers game have similar strategies, using the mandatory moves to gain advantage (figure 1.5).

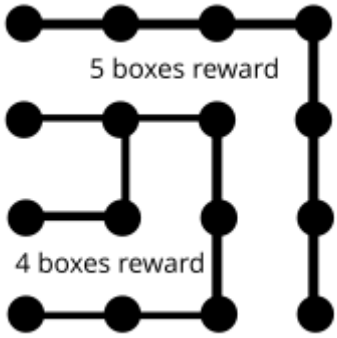


Figure 1.5: The next player to play has no possibility to win is forced to get less points than it will give

1.3 Objectives and contributions

This work will contribute to demonstrate that the AlphaZero algorithm can handle this game how efficient it can be that through deep learning techniques, whether it is possible to computer learn how to play a board game with no previous human game data and no human interaction, except following the general rules of the game.

This work will provide its own implementation of both the game and the algorithm, using as reference the AlphaZero implementation, comparing and analysing some others (described in related work section), and it will make them publicly available in the code repository GitHub[2]. It will also provide a comparison of the available implementations and results, including the results of some disputes between different agents.

1.4 Thesis organization

The thesis is divided into 6 chapters.

This chapter contains my personal motivation and a brief explanation about the game and its basic rules. It includes the objectives and contributions.

The second chapter contains the game playing and how the game works in a more technical way, an explanation of the MCTS algorithm, artificial neural network, and some other algorithms used.

[2]The source code was published in a Github repository `https://github.com/gionnani/dotsandboxes` and all documentations can be found in the repository home page.

The third chapter contains the related works on Dots and Boxes game, with their own implementations, the user interface (UI) of each one, the use of an ANN, other approaches like chains recognition, and some comparisons between them.

The fourth and fifth chapter describe the work done, the ANN structure used, the difficulties and restrictions found, and the results achieved at this moment.

The sixtieth chapter is the conclusion of the work and the future works proposed.

2

Game playing

2.1 The game

From the descriptions provided until now, it is clear that Dots and Boxes is a multi-agent, strategic-deterministic, turn-playing, zero-sum game[1], of perfect information, where we can define by the following components:

1. States

   (a) A matrix M used to define the board state[2] is composed of 0's and 1's and only lines or connections are represented. The drawn line is represented by the value 1, and the empty place by the value 0. The dimensions of the

[1] "These are games where the benefit of one player equals the loss of the other players" (Elkind E., 2011)

[2] Board state is the graphical or textual representation of the the board.

matrix M, $[M]_{r,c}$ are defined by equation 2.1, where r is number of rows, c is the number of columns and n is the number of possible boxes in a row or a column:

$$r = (2n + 1)$$
$$c = (n + 1) \tag{2.1}$$

This is an example for n = 3 boxes: r x c = (2 . 3 + 1) x (3 + 1) => 7 rows x 4 columns

All those elements will be flatten to be an input array for the neural network.

(b) Initial State

i. The initial state is composed by the matrix of zeros representing the empty board.

(c) Expansion operator

i. An operator that receives the current state and returns the immediate states which as achievable from the current one. This will be presented in chapter 4. Each state has a representation of the board and the information if the player won the game or not.

2. Number of agents, identifiers, and who's playing first

(a) The game has only two agents, identified by the version of the Neural Network in the configuration file.

(b) The neural network is defined by dotsandboxes(n)_xx, where "n" is the number of boxes and "xx" is the sequential number for the network version.

(c) The winner version of the neural network is the first player of the next game.

3. Board line positions

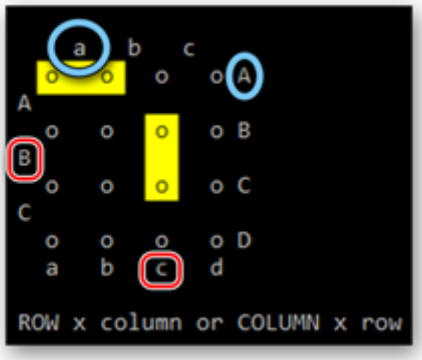


Figure 2.1: “aA” horizontal and “Bc” vertical

(a) Each board position is composed of two letters, one capital letter and one small letter as follow:

i. The letters represent the lines, not the dots.

ii. First letter starts in the left for capital letter or in the top for small letters.

iii. Second letter starts in the right for capital letter or in the bottom for small letters.

iv. The first letter defines if the line should be written in vertical or horizontal

A. If the first letter is capital (left), it means the line would be in vertical, written in the space between the two vertical dots. The following small letter in the bottom defines which column the line is going to be written.

B. If the first letter is small (top), it means the line would be in horizontal, written in the space between the two horizontal dots. The following capital letter in the right define which row the line is going to be written.

v. For example, the figure 2.1 has two marks in yellow, the line “Bc” defined by the row “B” and the column “c” (left to right then up to down). The line “aA” is defined by the column “a” and the row “A” (up to down then left to right).

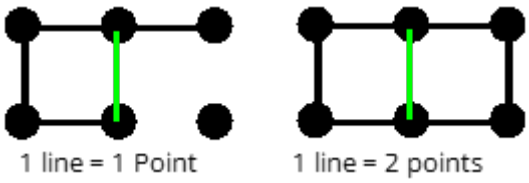


Figure 2.2: 1 or 2 points payoff

4. Agents actions

   (a) Each agent can choose any empty place (space with non-line) to write the line between the two nearest dots.

   (b) Each agent can only play once by turn and cannot skip without writing one line.

      i. If a box is created in the turn, the agent has necessarily to play again. It's mandatory! (see figure 2.2)

5. Payoff

   (a) One point for one box created with one line drawn or two points for two boxes created with the same line drawn (figure 2.2).

6. Termination condition

   (a) The game ends when the board has no empty place left. The number of the current playing is equal to the total of possible moves (equation 1.1). Wins the game the agent with more points. A draw occurs when the point of both players are the same in the end of the game.

The state diagram (figure 2.3) represents the complete game sequence. This game has an especial characteristic, it does not matter whom did the line at the moment, the only important information is who could complete the box/square, and consequently who makes the game point or points.

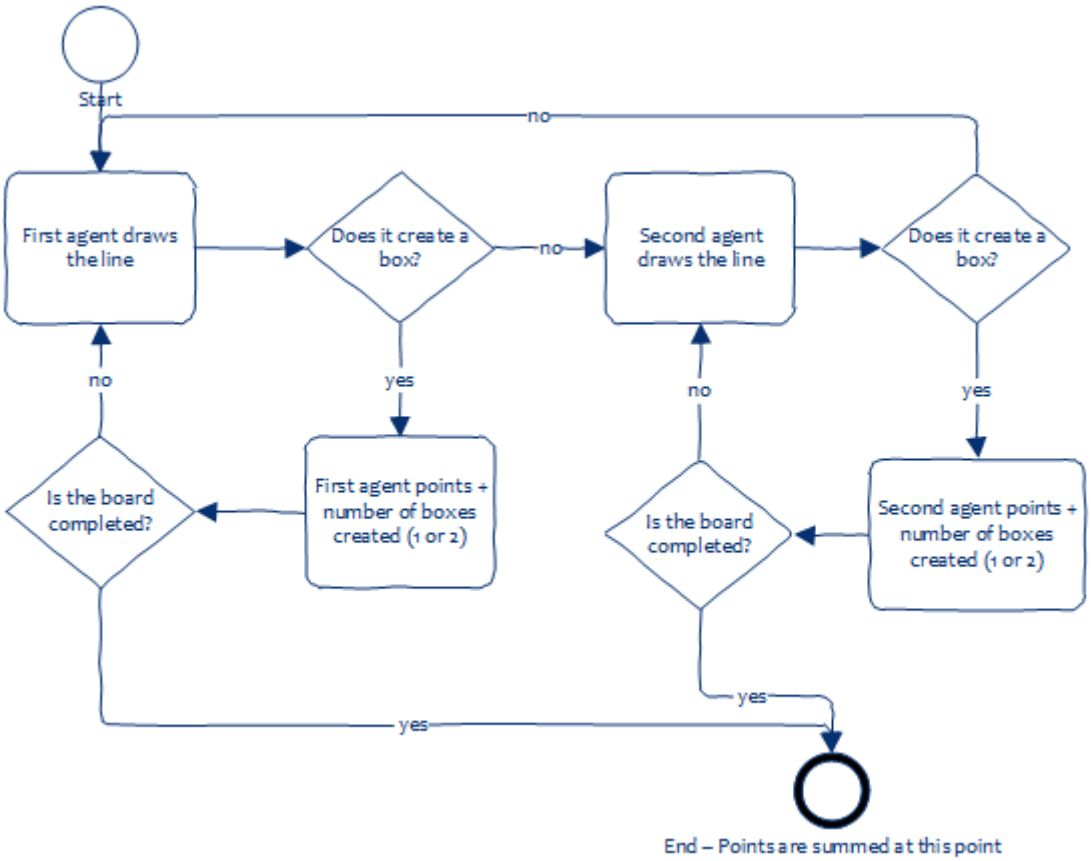


Figure 2.3: Gameplay State Diagram

2.2 Game characterization

The game is classified by the kind of information provided and how the agents actions affect the game state.

2.2.1 Information Provided

Games can be classified as games of perfect or imperfect information. Games of imperfect information have some core elements invisible to the player(Santos, 2017).

Chess, Checkers, Backgammon, and Boxes and Dots are examples of games of perfect information, while poker game is not.

2.2.2 Deterministic vs Stochastic

Deterministic games have the next state determined only by the current state and the action taken in that current state. Differently from the stochastic games where random elements affect the states. Examples of stochastic games are card games with shuffled elements (Santos, 2017).

Boxes and dots is a deterministic game.

2.2.3 Comparing games

The comparison of possible distinct games from Chess to Go, after only two start moves, is 1,225 (35^2) versus 62,500 (250^2). Dots and Boxes 3 x 3 (n = 3) has a factor of 24 (total of moves) that decreases after gameplay. In the middle game, with branching factor 15, there are 2,730 (15*14*13) different boards after only 3 moves (Aguayo, 2020).

That characteristic of the Dots and Boxes game is a game very hard to calculate all possible moves in real time without the use of an heuristic function, so the use of the AlphaZero algorithm for this game is really a very good choice.

2.3 Background

2.3.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a search algorithm based on randomness that uses a tree structure to simulate different gameplays (Santos, 2017).

No heuristic is needed in the original MCTS. The main role in the MCTS is to play the most number of games and choose the best move based on some configurations. MCTS is also part of the AlphaZero and can be used alone or associated with other strategies.

The MCTS is composed of four distinct steps to find the best option to play: Those steps are in order:

1. *Selection:* where the algorithm selects the state to be expanded.

   The selection step uses an UCB – Upper Confidence Bound. This bound implements a score to games played, using pieces of information from won games starting from node *i* (w_i), the number of simulations performed in that node, (n_i), the total of visits in the parent of *i* (*t*), and a constant c used to balance the trade-off between the exploitation and exploration components of UCB.

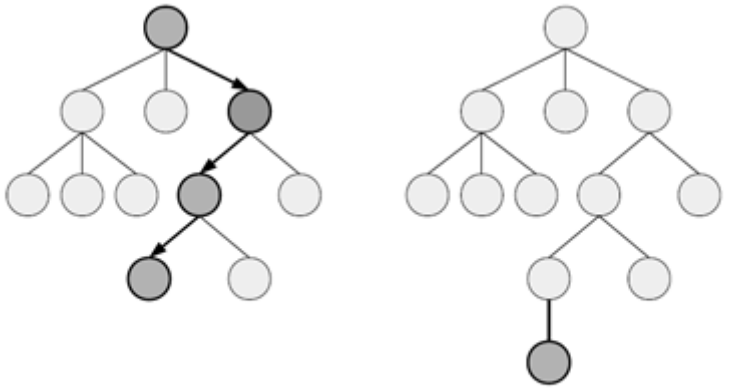
   $$UCB_i = \frac{w_i}{n_i} + c\sqrt{\frac{ln(t)}{n_i}} \tag{2.2}$$

   The selection process gets the node which will be used in the following process steps.

2. *Expansion:* the process that adds a new leaf into the tree after the selection and adds a new children to the node (state).

3. *Simulation:* after expansion, one or more game simulation are generated until the end of the game.

   Many times, this step is referenced by the rollout step. All the tracks from this step are kept for the next step.



(a) The selection phase. The selection policy is applied recursively until an unsampled action is reached.

(b) The expansion phase. The newly sampled action is executed and the resulting state is added to the tree.

Figure 2.4: Selection and expansion phases (Baier, 2015)

4. *Back-Propagation:* propagation of results

   The simulation results is backed propagated to the first node - the root node - and the MCTS process starts again from the selection stage until the exhaustion of a given computational resource; usually time. (Santos, 2017)

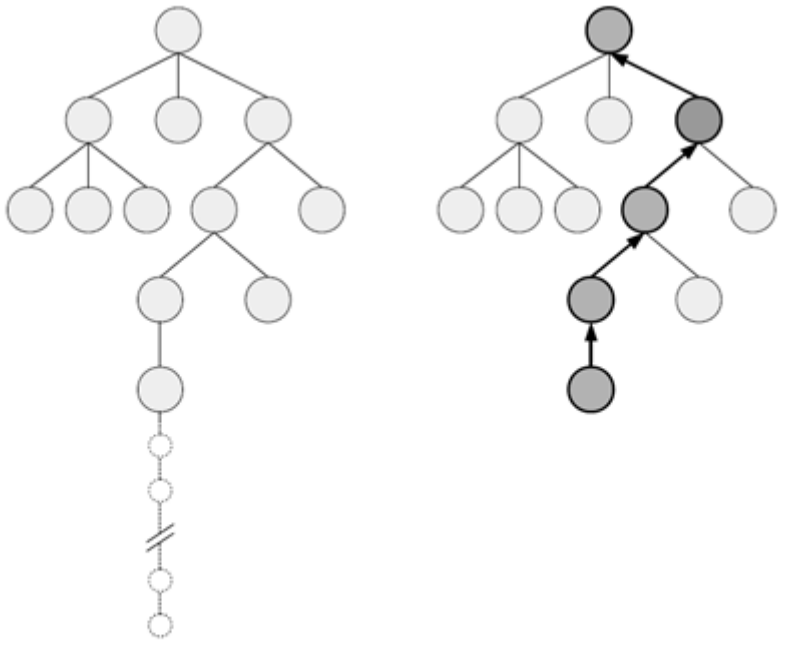
The algorithm complexity is proportional to the number of leaves. Each selected leaf has a whole game process, generating more and more leaves.

The more leaves in a horizontal level, the more time is necessary to cover different possibilities for a game. In a real-time game, it's impossible to wait until all possibilities are reached, so the use of learning techniques in this step could improve the quality of choice.

2.3.2 MCTS Alphazero Adaptions

The original MCTS was adapted in some steps to fit the alphazero format. Those modifications were in the selection, rollout, and the choice of the final element steps.

MCTS for AlphaZero is based on probabilities instead of using the simulation process (rollout) of complete gameplays to find what is the best move for the game. All probabilities are generated based on previous MCTS common game simulations and that data is used to train the neural network used in the algorithm.



(c) The rollout phase. One simulated game is played by the rollout policy.

(d) The backpropagation phase. The result of the rollout is used to update value estimates in the tree.

Figure 2.5: Simulation (rollout) and back-propagation phases (Baier, 2015)

Algorithm 1 *MCTS Implementation (Baier, 2015)*

Require: Start State

```
1:  MCTS(startState)
2:  for i = 1, 2, ..., numberOfIterations do
3:      currentState <- startState
4:      simulation <- ()                                  # selection
5:      while currentState ∈ Tree do
6:          currentState <- takeSelectionPolicyAction(currentState)
7:          simulation <- simulation + currentState
8:      end while                                         # expansion
9:      addToTree(currentState)                           # rollout
10:     while currentState.notTerminalPosition do
11:         currentState <- takeRolloutPolicyAction(currentState)
12:         simulation <- simulation + currentState
13:     end while                                         # backpropagation
14:     Score <- cumulativeReward(simulation)
15:     for all state ∈ (simulation ∩ Tree) do
16:         state.value <- backPropagate(state.value, score)
17:     end for
18: end for
19: return finalMoveChoice(Tree)
```

A most detailed explanation will be presented later in this section.

The pretrained neural network will provide the necessary prediction for each board state provided in the game simulation process. The network's output contains two sets of values. The values for the policy network for the current board and the action value for the prior value network contribute to find the best move.

The main point here is to use the MCTS to maximize the potential gains and to minimize potential losses. Classically other algorithms are used like minmax algorithm and alphabeta pruning to reduce the number of calculations to find a good result in the available time. The AlphaZero does not use any of them, what AlphaZero does is a sophisticated solution for those games (Aguayo, 2020).

Selection

Selection uses an modified version of the upper confidence bound applied to trees which is similar to upper confidence bound. It is composed of a constant C multiplied by the division of the square root of the total visit counts by the visit count of the board. The result is multiplied by the probability of an action a in the state s from the neural policy network. That's given more importance to nodes with high probability to be the best move without discarding the count of visited nodes (2.3).

$$U(s,a) = C_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{2.3}$$

The second part is to sum the result from CPUCT to the mean of Q and choose the best board (2.4).

$$a_t = argmax(Q(s_t,a) + U(s_t,a)) \tag{2.4}$$

At the beginning, the preference for selection the action a with high probability and low visit count but with the time it is more desirable to select boards with highest action value. (Silver D., 2017)

Probabilities and values

Four metrics are necessary to calculate the PUCT, those metrics values are initialized to zero at the beginning of the search (Silver D., 2017):

1. N(s,a) - The accumulated visit count for the position s and move (action) a.
2. W(s,a) - The accumulated total of the action values (backup values).
3. Q(s,a) - The average of the action value, that is the total of action value over the visit count for that position.
4. P(s,a) - The prior probability for select a edge.

Temperature

According to AlphaZero paper (Silver D., 2017), temperature value τ controls the deep search in the tree on MCTS.

In that context, temperature value will be one of two values (2.5): 2.6)

$$\tau = 1 \tag{2.5}$$

or

$$\tau \to 0 \tag{2.6}$$

When $\tau = 1$, the best move will be chosen proportionally to their visit count of MCTS and when $\tau \to 0$, the move will be chosen deterministically with the maximum visit count of MCTS.(Silver D., 2017)

The equation 2.7 represents the expression used to chose the board and how the temperature affects that. It's the division of the number of visits of the board raised to 1 over temperature value by the total visits of all boards raised to 1 over temperature. So, when the temperature is 1, it will be chosen the exact value of visits over total visits.

$$\pi(a|s_0) = \frac{N(s_0,a)^{\frac{1}{\tau}}}{\sum_b N(s_0,b)^{\frac{1}{\tau}}} \tag{2.7}$$

Expansion

The metrics are initialized to zeros every time the MCTS search is expanded for each leaf state s_L (2.8) and the values are not accumulated between the search's methods. In the game method, while N, W, and Q are calculated and increased by the MCTS, the probability value comes from the policy network for that board (Silver D., 2017). The probability is calculated by the final N, W and Q. It is the same calculus used to find the best move (see 2.7)

Those values are backpropagated to the priors nodes on the tree.

$$N(s_L, a) = 0, W(s_L, a) = 0, Q(s_L, a) = 0, P(s_L, a) = p_a \tag{2.8}$$

Rollout step

Rollout step is composed of the prediction for the board using a neural network. The results from the network fill the W and P values for that specific board.

Back propagation

The backprogation has two steps (Aguayo, 2020):

1. Adds 1 to the visit count of the board and the parents boards.
2. Calculate the Q value for the board and the parents boards.

Selecting the playing position

The final action value of the board is defined by the division of the mean of the sum of all action values of one board by the visit number v of the same board (2.9). The W value is the total sum of the priors action values and the Q value is the mean.

$$W(s_t, a_t) = W(s_t, a_t) + v \tag{2.9}$$

$$Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)} \tag{2.10}$$

That defines what playing position is the best option according to the neural network and MCTS (Silver D., 2017).

Continuous play

The game is played in two modes:

1. Competitive Mode

   The game is played in competitive mode when two different neural networks are playing against themselves. The champion neural network is going to be the candidate to the best model. The neural network which wins most of the games in the challenge is the best model and the neural network which goes to the training mode (Silver D., 2017).

   Competitive mode uses temperature value set to 1 that means the best move ever is chosen from the neural network (Silver D., 2017).

2. Training Mode

   The game is played in training mode with two of the same versions of the neural network. After the challenge in the competitive mode, the winner neural network competes with itself in training mode to generate all necessary data that will be used to train the new version of the neural network (Silver D., 2017).

The model is trained continually in both competitive and training modes and the new versions are evaluated to become the new version of the neural network to be used (Silver D., 2017).

2.3.3 Artificial Neural Networks (ANN)

Artificial Neural Networks are inspired by biological neuron structures, which use a number of neurons, interconnected by synapses, which transfer and process the signal on the path. An artificial neuron is composed of functions that interconnect inputs to output. That structure 2.6, represents a neuron in which all the *inputs*($i = [i_1, i_2, \ldots, i_p]$) multiplied by the weights ($W = [w_{i,1}, w_{i,2}, \ldots, w_{i,p}]$), together with the bias value,

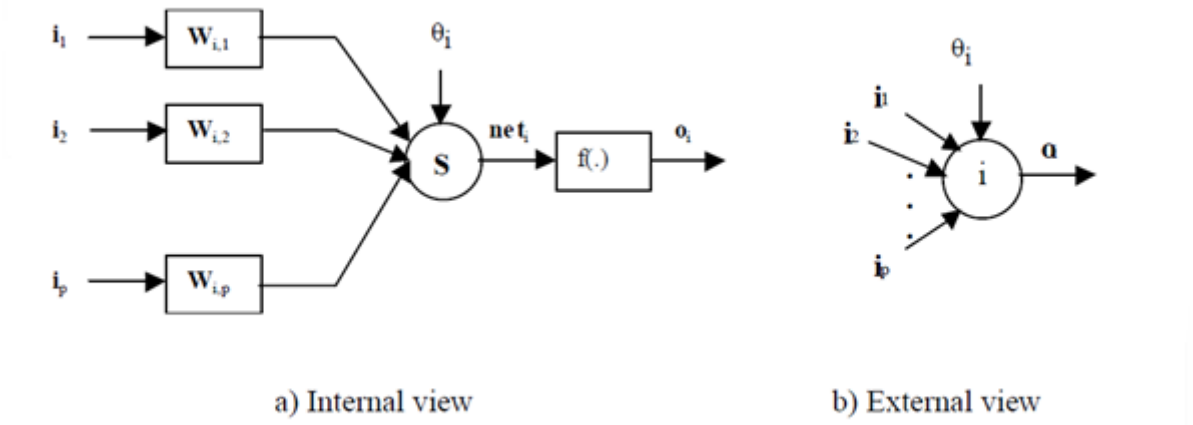


Figure 2.6: Model of a typical neuron (Ruano, 2016)

θi, are connected to the summation block *(S)*, and, through the activation function *f(.)* produces the output (Ruano, 2016).

The structure of an Artificial Neural Network is formed by many neurons, spread in many layers. According to figure 2.7, the network with inputs X1 to X7, uses the hidden layers to process those inputs and generate the output (Krogh, 2008).

The goal is to minimize the errors in all classification process, as evaluated by a loss, error or cost function. Many times the cost function used is the mean squared error (MSE).

2.3.4 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a special kind of Artificial Neural Network that utilizes convolutional operations in at least one of its layers. The CNN has a vast application in the search of patterns (Goodfellow et al., 2016).

The convolution utilizes weighted averages of the inputs and kernel as described in the figure 2.8.

Therefore, as the convolutional operation uses the real-time value, each measure has its own value, and the weighted average of several measures is used to approximate to the best value (Goodfellow et al., 2016).

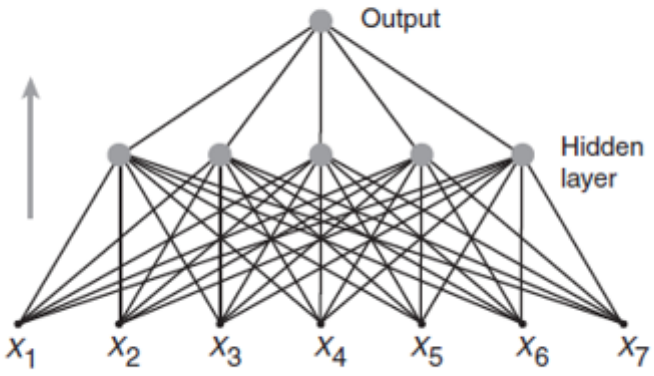


Figure 2.7: ANN Structure (Krogh, 2008)

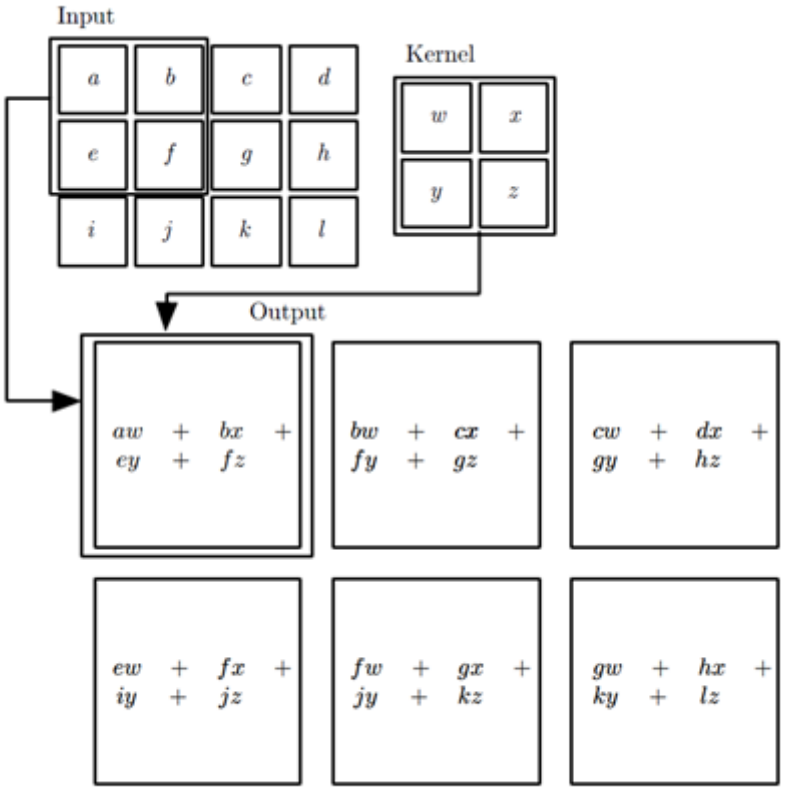


Figure 2.8: Example of convolution operation for two dimensions (Goodfellow et al., 2016)

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a) \tag{2.11}$$

The convolutional operation is realized most of the time over tensors, which means multidimensional arrays. The two-dimensional Kernel over a two-dimensional image could be represented by the formulae:

$$a)\ S(i,\ j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m,\ j-n) \tag{2.12}$$

OR

$$b)\ S(i,\ j) = (K * I)(i,j) = \sum_m \sum_n I(i-m,\ j-n)\, K(m,n) \tag{2.13}$$

OR

$$c)\ S(i,\ j) = (K * I)(i,j) = \sum_m \sum_n I(i+m,\ j+n)\, K(m,n) \tag{2.14}$$

The formulas 2.12 and 2.13 are equivalent to the cumulative characteristic of convolution and 2.14 is a cross-correlation function, which works like a convolution but flips the kernel and is used in some frameworks (Goodfellow et al., 2016).

2.3.5 Batch Normalization

Batch normalization was proposed by Sergey Ioffe and Christian Szegedy, both from Google, in 2015, to resolve what they called *"internal covariate shift"* - the parameter changing between layers on training, what required lower learning rates, and caution on choosing the initialization parameters. The proposal was to make *"normalization a part of the model architecture and performing the normalization for each training mini-batch"* (Ioffe and Szegedy, 2015).

For convolutional layers, the normalization follows the convolutional property – *"so that different elements of the same feature map, at different locations, are normalized in the same way"* (**?**). All activations are normalized in mini-batchs in all locations.

2.3.6 Rectifier

The rectified linear unit function (ReLU) performs an operation where the output is always greater than 0. It rectifies negative values to 0 but preserves the positive ones. *"ReLU is a faster learning activation function"* (Nwankpa C., 2018) and when it is *"used in hidden layers can improve the learning speed of various deep neural networks"* (Ide H., 2017).

$$f(x) = max(0, x) \tag{2.15}$$

2.3.7 Softmax

Softmax is an activation function that limits the output to a range of 0 and 1 like sigmoid functions, but with more classes simultaneously involved. This function is commonly used in the final layer of the neural network (Mercioni and Holban, 2020).

$$softmax(x)_i = \frac{e^{(x_i)}}{\sum_j e^{(x_j)}} \tag{2.16}$$

2.3.8 Argmax

Argmax is a general statistic method to estimate which parameter maximizes some function

$$\Gamma_n(\theta) = \Gamma_n(\theta; X_1, ..., X_n)$$

of some unknown parameter; sometimes, it is known as M-Estimator.(Ferger, 2004)

$$\theta_n = argmax\Gamma_n(\theta) \Rightarrow \theta \in \Theta \tag{2.17}$$

2.3.9 Dropout

Dropout is a technique to prevent overfitting during the training of the data. The dropout consists in dropping out units of hidden layers during the training of the neural network to improve generalization (Sammut and Webb, 2017; Srivastava et al., 2014) as seen in figure 2.9.

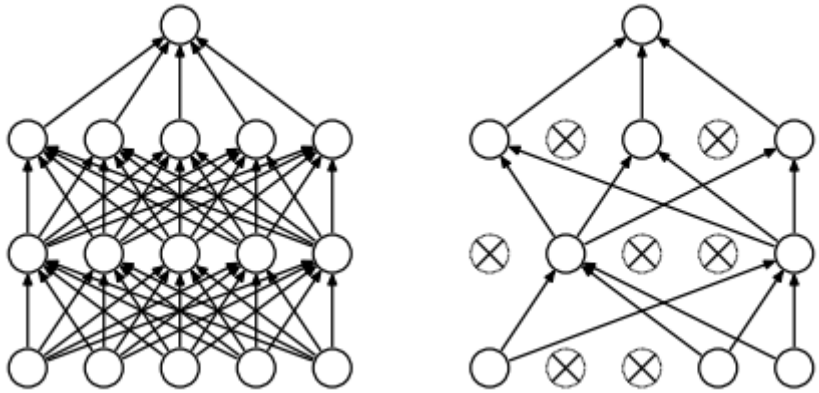


Figure 2.9: Example of neural network of 2 hidden layers with dropout at right (Srivastava et al., 2014)

2.3.10 Loss-Functions

Loss functions are functions that determine the loss or cost of a prediction y', when y is the correct value, in a training process (Sammut and Webb, 2017).

Two losses functions are used by training process of AlphaZero: Categorical Cross-Entropy and Mean Squared Error.

Categorical Cross-Entropy

Categorical (qualitative) variables are that can distinguish categories by some characteristics, link gender, colors, evaluation, etc. (Sammut and Webb, 2017).

Categorical cross-entropy is a multiclass loss function (Gulli and Pal, 2017) and it can be defined by the difference of two probability distribution of a event to occur, the summation of discrete states, and the log of the probability *log p* of a random event *t* (2.18):

$$L_i = -\sum_j t_{i,j}.\log\left(p_{i,j}\right) \tag{2.18}$$

Mean squared error

Mean squared error (MSE) is a metric evaluation function that calculates the squared mean of the difference between actual value y and the prediction *lambda(x)* over all instances of a set n (Sammut and Webb, 2017).

The mean squared error can be represented by the equation 2.19.

$$mse = \frac{\sum_{i=1}^{n}(y_i - \lambda(x_i))^2}{n} \tag{2.19}$$

2.3.11 Learning

Supervised Learning

Old versions of AlphaGoZero (version for the Go game for example) used supervised learning for training and getting game data from expert players best plays.

Supervised learning is the learning with the support of a supervisor. The data is provided in sets of inputs and expected outputs. The agent corrects the parameters to minimize the loss improving the accuracy of the model (prediction - expected value approximates to zero)(Bonaccorso, 2017).

The main point here is using new data to train - the data that never was seen before; the generalization of the model is the key to avoid the overfitting (overlearning) (Bonaccorso, 2017).

Reinforcement Learning

Reinforcement learning is based on the evaluation of unsupervised learning and the analyse of the feedback provided by the environment (see figure 2.10 (Bonaccorso, 2017)).

The feedback provided is called reward, that can be positive or negative. AlphaZero has as reward the prediction from the value network. The policy is what the agent uses to learn. The policy combined with the accumulated value/reward gives the highest immediate and cumulative rewards - a highest total reward.

The other calculus involving that highest total rewards provided by the neural network is used to chose the best move for a determined board.

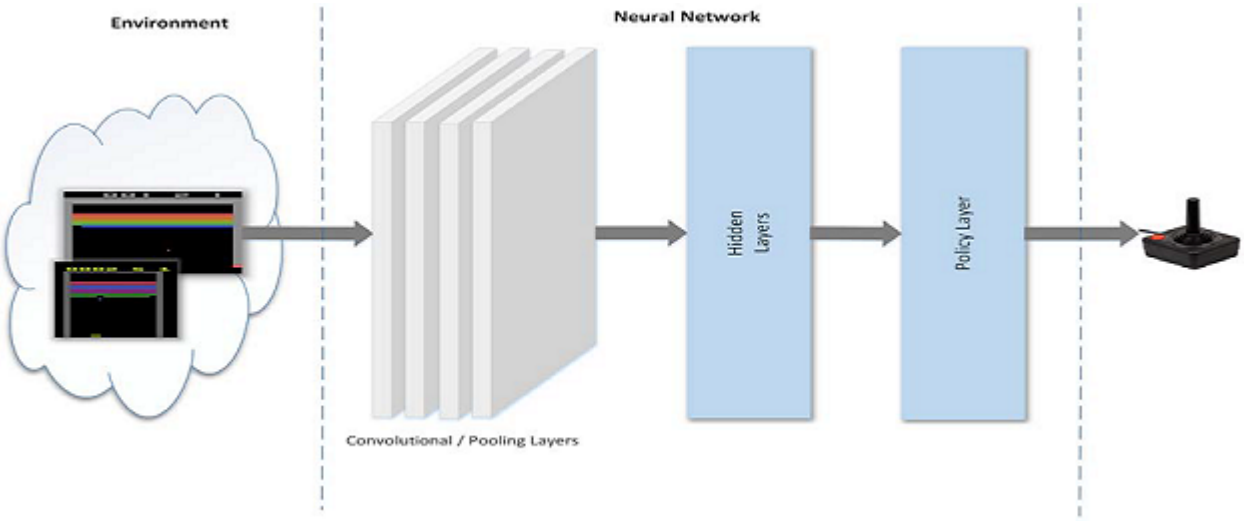


Figure 2.10: Reinforcement example for an Atari game (Bonaccorso, 2017)

3

Related Work

3.1 AlphaGo to AlphaZero

AlphaGo is a program created to defeat a human player in a Go board game (Silver D., 2017). The performance is obtained through the use of supervised learning, with techniques of experts in Go games, predicting the moves from them and *"refined by policy-gradient reinforcement learning"*(Silver D., 2017). After that, the network is retrained playing against itself and, combined with MCTS, providing moves of high probability for the game. That version was called **AlphaGo Fan**. The name was given in October 2015 because of Fan Hui, the opponent, an European champion. This version taked 176 GPUs distributed over many machines. (Silver D., 2017)

The next version was **AlphaGo Lee**, because this version defeated Lee Sedol, in-

ternational winner, in March 2016. The Lee version defeated the opponent in 4 to 1. This version taked two neural networks and was trained based in fast self-play games from Alpha Go. The policy network was trained by supervised learning and refined by policy gradient reinforcement learning. The objective is to predict human expert moves. That was distributed using 48 TPU for faster results. (Silver D., 2017)

AlphaGo Master was the last version created before the AlphaGo Zero version, defeating players in 60 to 0, in 2017. This version uses supervised learning from human data in the beginning of the training combined with other techniques coming from other versions. (Silver D., 2017)

The AlphaGo Zero version was the first version that did not use supervised learning anymore. That means that all process of training started with random weights and learned from self-play games without human supervision or any kind of human data. This was the first time that only one neural network was used in that process (previous versions used more than one neural network). The MCTS is in the process too, like its previous versions but, no rollout is made, and the network is capable to provide the result quickly and precisely. (Silver D., 2017)

Considering this work is based on AlphaZero version, evoluation of AlphaGo Zero, this version will be more detailed than the older ones.

AlphaGo Zero process is composed of three stages. The first stage is self-playing, a stage where the **AlphaGo Zero** creates data to train from games against itself. The number of games played defined was 25,000, always keeping the information of who is winning (1) or losing (0) the game. The second stage is the training, a loop from a small sample from the last 500,000 games randomly generated from the last step and, at last, it plays more 400 games with the best CNN against the new trained CNN. The winner from this dispute is defined as the new best CNN position if it is the one which wins 55% of those 400 games and does the same operation again over and over.

The game state is the representation of the board data on time and it is composed of the shadows of boards from the actual and 7 previous boards for the black, more actual and 7 previous boards for the white stones plus the sheet that identifies the

current player. This state turns a stack of 17 matrices of 19 x 19.

The model is composed of 19 or 39 residual blocks with that structure repeating a convolution of 256 filters of kernel size 3 × 3 with stride 1, batch normalization, and a rectifier nonlinearity, a fully connected linear layer to output a 362 size vector and at the end, and a fully connected linear layer in the range of -1 and 1 (Silver D., 2017).

In that process, MCTS simulates 1,600 times beginning from the actual state (root node) for the choice of each movement. The simulation phase is not a rollout but a modified phase with predictions from the CNN.

The **AlphaGo Zero** could turn itself the best player, defeating humans after 40 days of self-training. The computional power to train was only one machine with 4TPUs. Alpha Go has the capability to be distributed too, but according to (Silver D., 2017), they made the *choice to use the simplest possible search algorithm*.

3.1.1 AlphaZero

AlphaZero is the evolution of the AlphaGo Zero with improvements to work not only for Go games but also to work with others board games and it is the chosen version for this work.

Like the AlphaZero, this work will not have game data previously generated from experts, starting only from random games. The MCTS will not perform simulations, consulting always the neural network model to achieve the best movement for the actual state.

3.2 Game implementations

3.2.1 Python and Javascript implementation

The first case studied was the implementation of a web app in Javascript of AlphaZero for Dots and Boxes made by (Aguayo, 2020). The author, in his work, raised the question about the classification of the board like we do with images of dogs and cats for example.

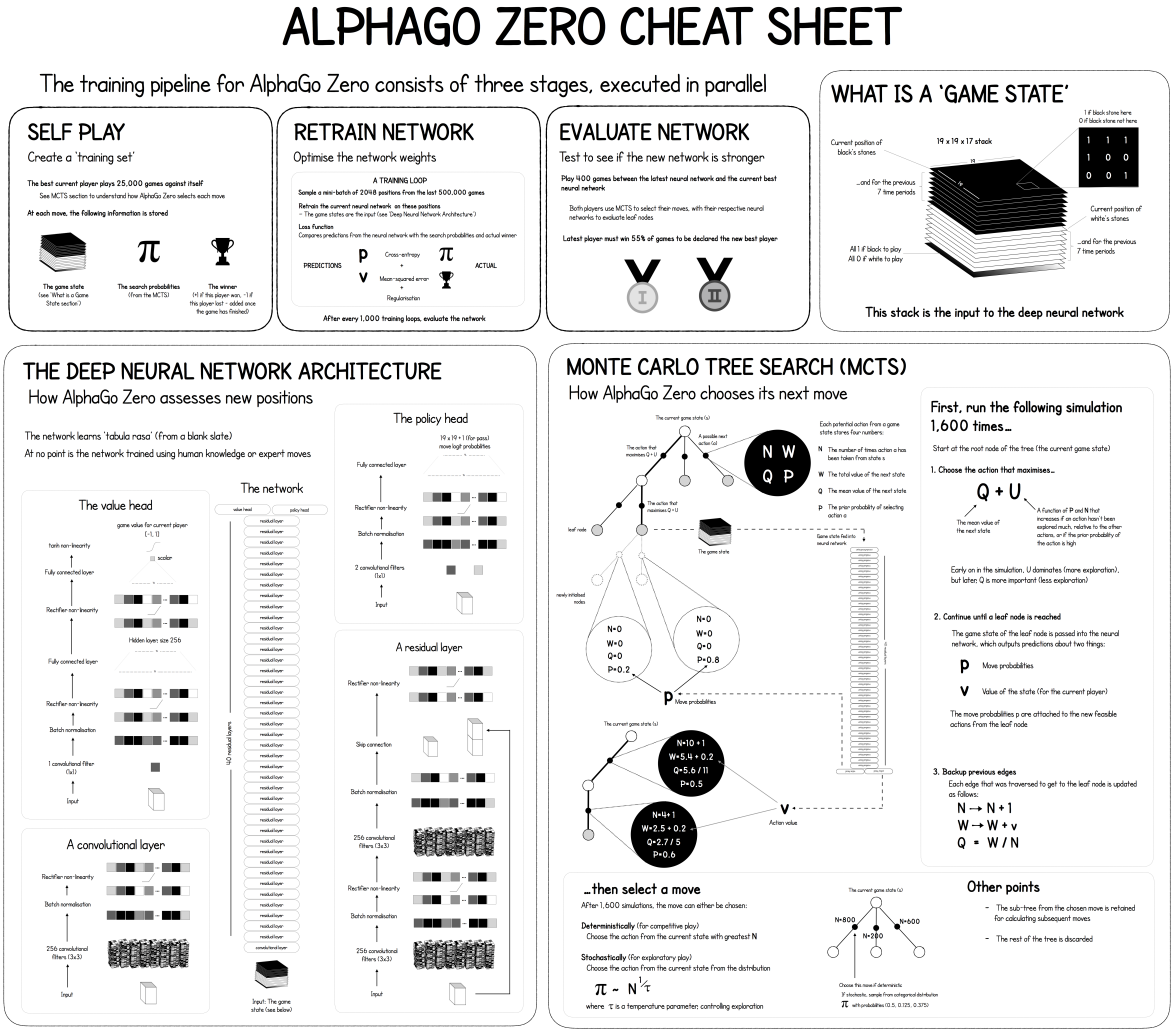


Figure 3.1: AlphaGo Zero process, state data, CNN and MCTS (Silver D., 2017)

Like dogs and cats, is it possible to classify a board in a winning or losing board? (Aguayo, 2020) And he has show that it is possible to create a simple game, a dots and boxes game 3 x 3 to demonstrate it.

The result generated was pretty good, so, and in all my tests, it was hard to defeat this game by myself (as human) but it was possible, both by myself and by my implementation of the standard MCTS.

This work was created as an extended work from Surag Nair[1], a PhD student in Stanford University. Surag implemented an AlphaZero general project with possibility of extending games to add to his project and Aguayo added his own implementation of Dots and Boxes to there.

The Surag Nair implementation was coded in Python, so Aguayo make his implementation for training the neural network in Python and ported the game to Javascript. Tensorflow.js library was used to make a connection with the pre-trained neural network and to predict the moves from that network.

Aguayo also put some Jupiter Notebooks to help anyone understand how he did the training and playing games, but at this time, the python notebooks raised some errors based on old libraries so it is not possible to simulate all his work. It was a very hard work to extract his version of the board state and understand it, but at the end this project contributed a lot to mine.

However, the code he used to train did not work without some modification but, the final version of the neural network present in his repository is already trained and works very well.

State

Aguayo defines the state a little different from mine in positions but not in size of the array. His state also has 28 positions to represent all 24 possible moves (n = 3). The others 4 positions are divided by agent points and the game end flag (figure 3.3).

An example of a board representation state with positions 13 and 4 selected would

[1]https://github.com/suragnair

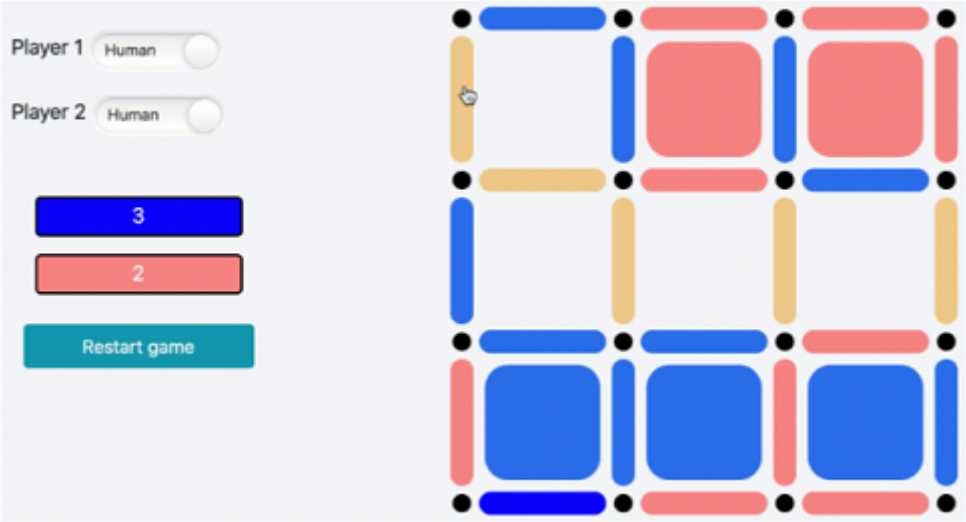


Figure 3.2: Aguayo's Javascript implementation (Aguayo, 2020)

Figure 3.3: Aguayo's board numbering positions (Aguayo, 2020)

be: "0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0", and the board representation for figure 3.2 would be: "1,1,1,3,0,1,1,2,1,1,1,0,1,1,1,0,0,1,1,1,1,0,0,0,1,1,1,1". This state can be confirmed by the conversion of the board to string, injecting the JavaScript command *board.toString()* - in the browser console (F12).

The numbers 3 and 2 represent the points for the first and second agent, respectively. This values are normalized in the training process.

Implementation Problems

The port of dots and boxes was based on the extension of Surag Nair, mentioned above, but it does not consider games which players can repeat the move, the case of dots and boxes.

The MCTS simulation step changes the value of the value network from positive to negative and vice versa for each player turn, continuously multiplying the value by -1. It works pretty good in other games. Dots and boxes game has to consider who is playing in that moment, and only invert if the player changes.

The performance of the network trained could be affected by this mistake, however the final network generated was pretty good in the performance. The most of the games finish with many player's moves in sequence what can be the cause of success in the neural network.

3.2.2 Peters' Thesis

Another related work was did by Tom Peters in his master's. He implemented the AlphaZero and the game, training the neural network but in a different way (Peters, 2018). The author uses images instead of text board configurations as inputs to the network in order to find the best boards through an CNN.

Peters' objective in that work was to check if it is possible to create a good network for this game with low computational power, not what was used to train the AlphaZero and if it is possible to use the same trained CNN for different sizes of boards, and to find new strategies for self-playing. Finally, he tested his game challenging a QDab Dots and Boxes game and Dabble, both tagged with the best AI for Dots and Boxes, and he could not defeat them in one hundred games played (Peters, 2018).

As the results, he found to be possible to train the network with small resources and in his experiments it was possible to transfer the acquired learning to big boards and could change some hiper-parameters to find better results in his training process.

The game implementation was in C++ language, divided in two softwares. The first one is an image generator to create data and the second one is the game called Ksquares. This game has a graphic interface and runs on Linux KDE.

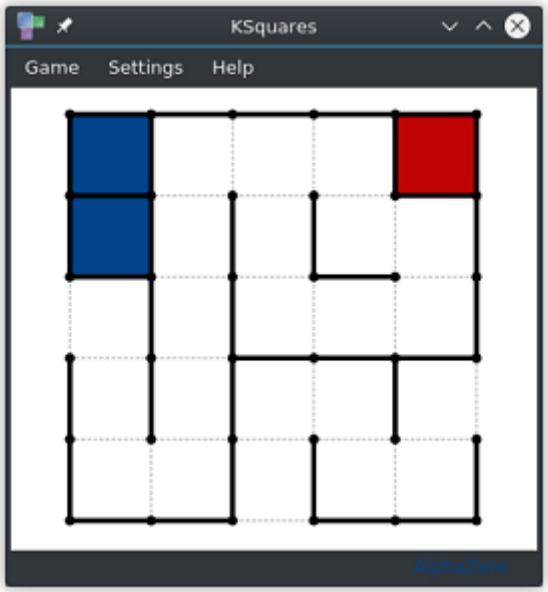


Figure 3.4: KSquares (Peters, 2018)

3.2.3 Zhang, Li, and Xiong's work

More recently, Zhang, Li and Xiong, (Zhang et al., 2019) created a Dots and Boxes AlphaZero game and compared it only to MTCS. The results was presented in a conference paper and although the training and results were small, they defeat MCTS all of 10 times played.

They used a CNN model with 5 layer, the first one with 32 filters and kernel 3x3, ReLU, the second one with 64 filter (3x3 + ReLU), and the third and forth ones with 128 filters, 3x3 and ReLU. After the fourth layer, the network splits into the policy network and value network for the fifth layer as the figure 3.6.

The MCTS simulates 400 steps for each movement chosen and the use of random probabilities with Dirichlet Distribution (0.25). The training was made every time the sample reaches 2000. In 500 games played the winner rate was 100 % with the pure MCTS game (Zhang et al., 2019).

Details about implementation and the state format was not provide and work replication becomes impossible.

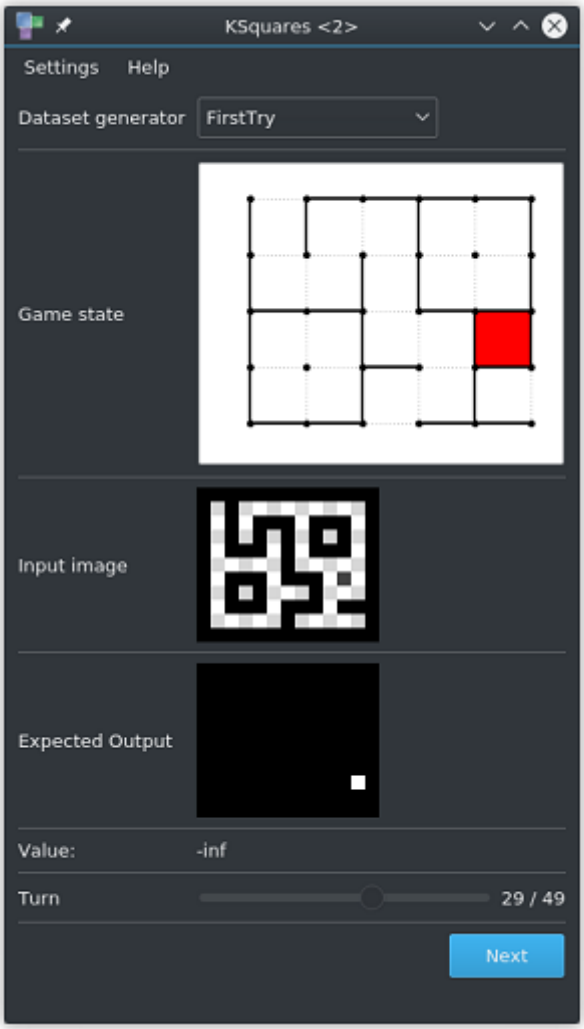


Figure 3.5: Data Gerenerator (Peters, 2018)

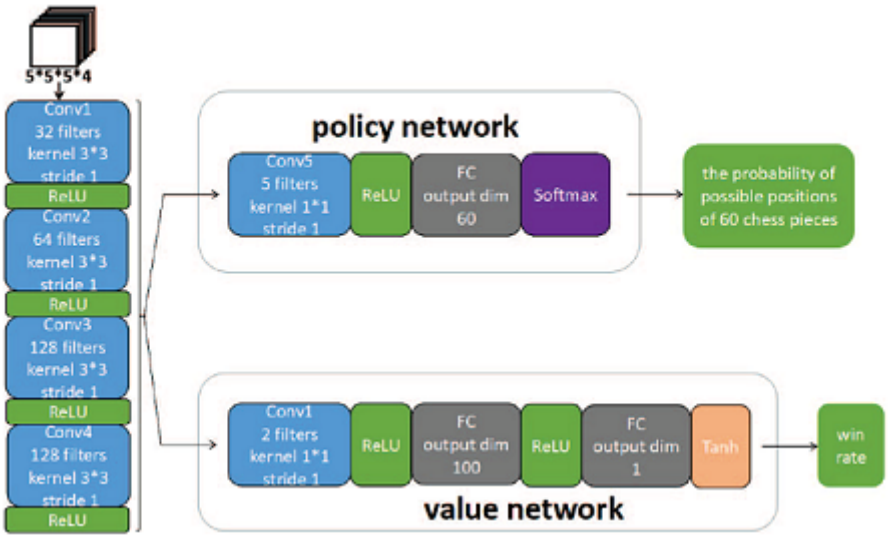


Figure 3.6: Convolutional Neural Network Model (Zhang et al., 2019)

3.2.4 Neural networks and Convolutional Neural Networks

All those games were written with different kinds of models to make the same work based on AlphaZero paper. Some of them implemented more features and others less, but all works have significant improvement from the pure/original MCTS. For example, the Aguayo's (Aguayo, 2020) implementation does not use the convolutional neural network in his model but the others implement it like (Peters, 2018) and (Zhang et al., 2019). Peters input images for training and Zhang are not clear about the state format.

3.2.5 QDab Game

Qdab is cited by Peters in his master's thesis (Peters, 2018) and this game was created by Yimmon[2] using goland and Python 2.7. and was compiled to linux.

The github contains the full source code, but without details about the implementation and resources used in this game(Zhuang, 2015).

3.2.6 Dabble Game

Dabble also was cited by Peters in his master's thesis (Peters, 2018) and this game was written in Microsoft C++ environment by J.P. Grossman. This game uses chains and double-crosses internally to find patterns and follow the best possible way. The author said sometimes it uses brute force to find the best move.

Dabble was introduced in the second Combinatorial Game Theory Research Workshop, held at the Mathematical Sciences Research Institute, July/2000 (Grossman., 2000).

[2]https://github.com/yimmon

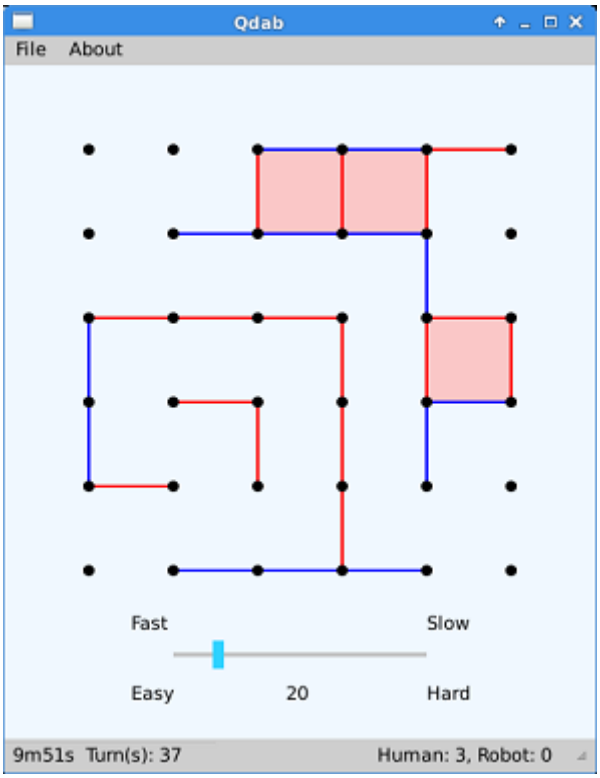


Figure 3.7: Qdab interface (Zhuang, 2015)

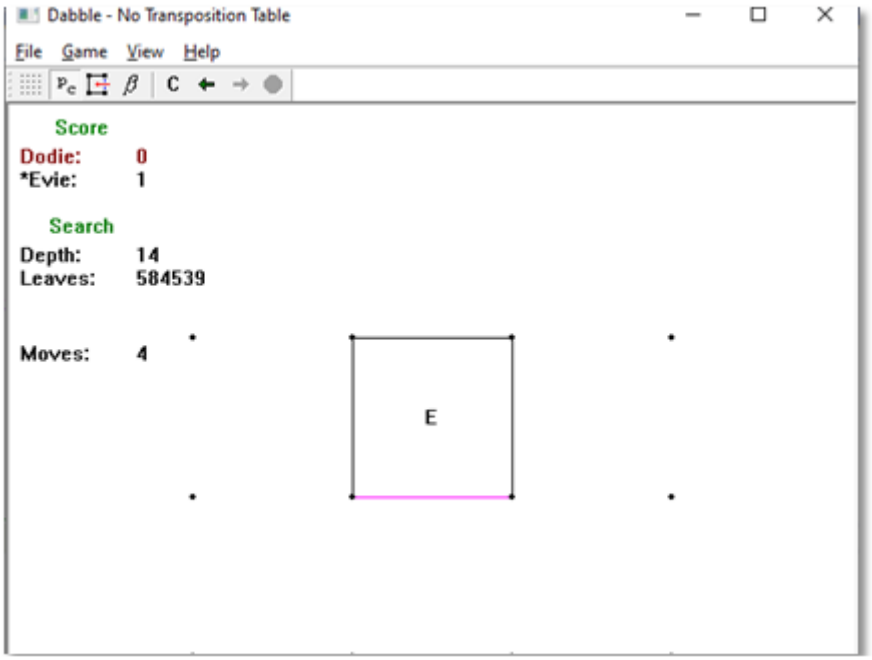


Figure 3.8: Dabble interface (Grossman., 2000)

4

Proposed approach

4.1 Language

The language of development used is C# and Python. The choice of C# was made based on my background in .net. Python 3 was used for the neural network training.

4.2 CNTK e ML.Net Frameworks

Microsoft Cognitive Toolkit (CNTK) and Machine Learning for .net (ML.net) are two libraries that work with convolutional neural networks. Both can use a CNN, but only CNTK can train the network. Microsoft did not implement the training capabilities to ML.net for tensorflow models and the GPU card is mandatory for the training process of the own model. Both provide powerful tools to work with machine learning

projects.

ML.net was preferred over CNTK because the CNTK requisite of GPU card and CNTK incompatibility with Ubuntu Colab version. ML.net does not need the graphics card to use the network pre-trained.

4.3 Google Colab

Google Colaboratory, in short Colab, is a tool that offers the use of a GPU card free of charge. Colab has a Python notebook interface and runs over Ubuntu Linux, which makes possible the .net core integration to this project. Unfortunately, the CNTK did not work for incompatibility with the Ubuntu Colab's version, so only the ML.Net could be used, but need some integration by the fact the ML.Net does not train the model.

The integration for the CNN was created and the training process was made in Colab, with Tensorflow[1] and Keras[2], so the pre-trained neural network could be used by the game for simulations.

Google Colab was used to train only in the first version of the neural network. All the process takes a lot of time and Google Colab has a reduced time until restarting the environment, many training cycles and much time was lost in preliminary tests for that limitation; cloud virtual machines were used instead of Colab in the training process.

The training process in a virtual machine with tensorflow, GPU card does the training process quickly but CPU can emulate that, but takes more time to conclude.

4.4 Cloud Virtual Machines

Virtual machines were created in three cloud platforms: Microsoft Azure, Amazon Web Services and Google Cloud. No differences between the implementation and performance of those virtual machines were noticed in all of those platforms.

[1] "The core open source library to help you develop and train ML models." from Tensorflow website.

[2] Keras is an API for make things easy in a deep learning environment. Keras and Tensorflow work together.

The only reason to use more than one platform is to balance the costs of in all of them. The resources used were the same for all of them. All machines were created with Linux, Ubuntu OS version 18.4.

4.4.1 Configuration

Two kinds of hardware were used in the process of generating initial data and training:

1. Generating initial data

   This task didn't require strong hardware to finish the work. The machine used here was from the free tier in those platforms and represents a small machine with only 1 vCpu and 1GByte of RAM.

2. Training process

   Training process needs more from the hardware to execute. The minimal requirements for this tasks was machines with 2 vCPu's and 8 GByte of RAM. That represents the D-tyer in Azure and e2-standard-2 in Google cloud.

   The configuration for AWS was a machine CPU Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz, for Azure was a machine CPUs Intel(R) Xeon(R) Platinum 8171M CPU @ 2.60GHz (50654) and the Google machine was an Intel(R) Xeon(R) CPU @ 2.30GHz.

   Google cloud suggested a e2-custom (4 vCPUs, 8 GB memory) type instead of e2-standard-2, reporting it was over utilized. Modification was done and new neural networks were trained using that configuration.

   The local machine used in some competitions was a Intel64(R) I7 Family 6 Model 58 3401 Mhz, 12 GBytes of RAM.

4.4.2 Packages

Google Colab has the packages configured by default, however virtual machines need more configuration and more packages for installation. The packages list can be viewed below:

1. .Net packages for the game, aspnetcore-runtime-3.1 and dotnet-sdk-3.1[3]
2. python3-distutils
3. python3-apt
4. pip
5. pip tensorflow
6. pip IPython
7. pip scikit-learn
8. python3-pandas

The AWS machine needs a special configuration for allocating a swapfile in the free tier. See *Appendices B - Configuration* for more details.

4.5 State

The state is the representation of the board in the written form, it contains the player's movements for each board in that moment. For this game, the identification of which player did that the movement is not represented, once that it is not relevant; other games, like Tic Tac Toe, have the player information as mandatory.

The state is composed of ones and zeros; Each possible line move turns to a matrix element, so if the line is not drawn, its value in the state is 0 else the value is 1. However, this matrix does not have the same number of zeros and ones in lines and columns. Positions that can never be played were represented by zeros. That occurs because the height and width of each of those boxes has a line with x segments, but the column has x + 1 segments, see the example in the figure (4.1).

The figure **??** is the representation of the 3 x 3 board, with the chance of 9 possible boxes. "A" represents a matrix 7 x 4, corresponding to board 3 x 3, and the colored

[3]found in https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb

numbers are represented by the same colored lines at the right side. “B” represents the completed board after the finished game. The matrix has 0’s at the last position of each row representation, but in the column representation, it fits correctly in size to the end.

At the end, this matrix is flatted to an array with all positions, even the positions that can never be reached, see the figure 4.2. Note that B state has 4 zeros and its represents the completed final board.

In this game, the state can be rotate and flipped. There are 8 possible equivalent boards when it are rotated and flipped. Those boards represents the same board, but seen in another perspective.

In that version of the alphazero, the symmetrical states are included in the training for maximizing the number of generated data by the game.

The figure 4.3 contains all possible similar states created by the rotation and the flip of the original state.

4.6 Neural Network

The neural network used in this work has the same structure as the one created by Aguayo in his work (Aguayo, 2020), with one array of 28 elements as input, each of 28 items representing a position on the board with the value 1 if this position are marked and otherwise with 0 value.

The outputs of the neural network are two arrays: the policy network array (28 items) and the value network array (1 item) (Silver D., 2017), as documented in AlphaZero. The value network returns only one value, between 1 and -1, which is the prediction of the game to be a winner or loser board. The bigger the number, the higher is the probability of being a winner board. The policy network predicts which

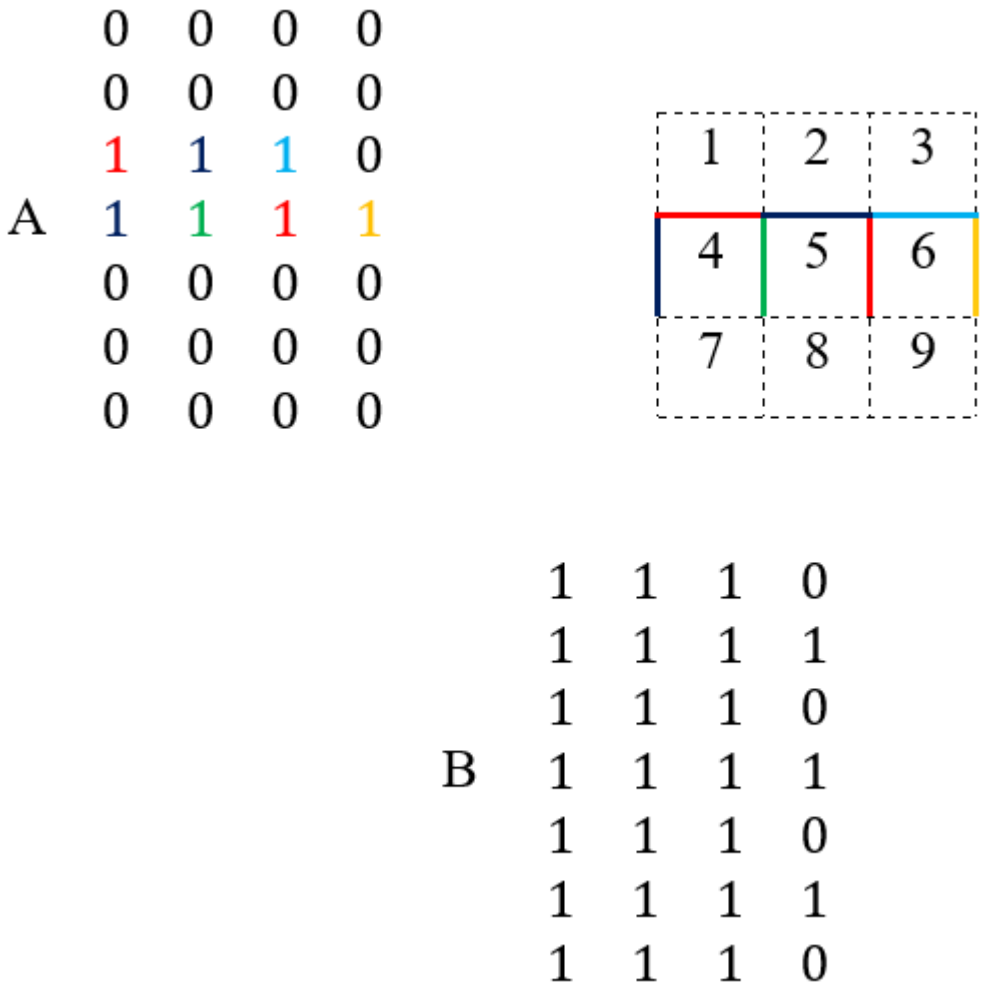


Figure 4.1: Data representation for dots and boxes state

A = 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
B = 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0

Figure 4.2: Array representation of the state A B from the figure 4.1

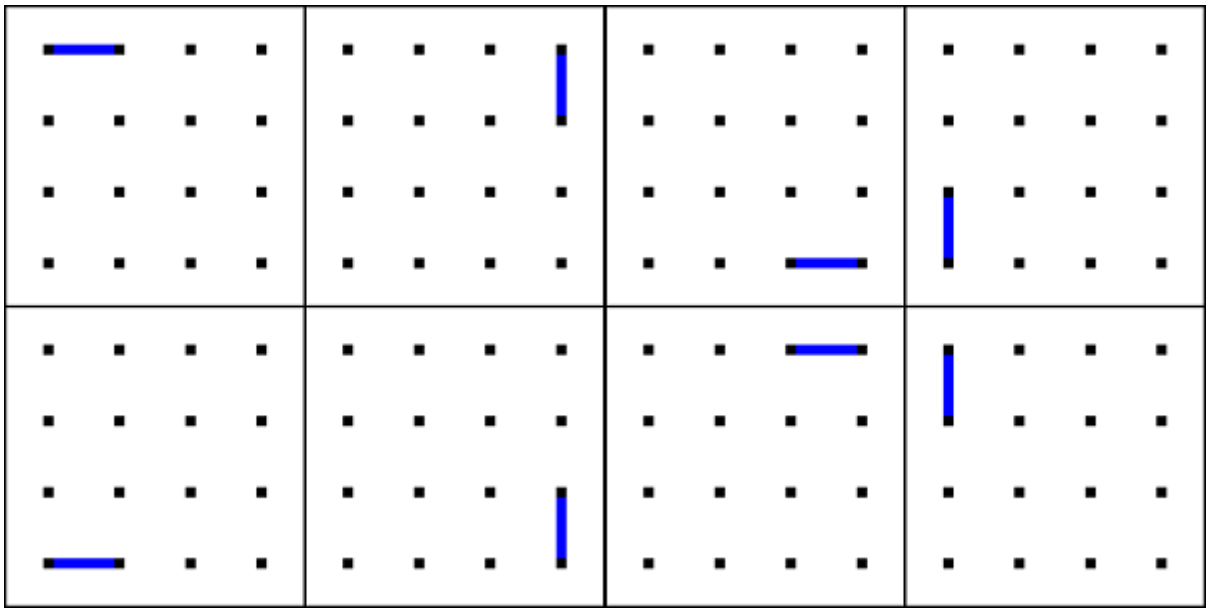


Figure 4.3: Rotation and reflection for the 8 symmetrical states (Prince, 2017)

of the positions will be the best next option to play.

$$v := \begin{cases} -1 & \text{if lose the game} \\ 0 & \text{if draw the game} \\ 1 & \text{if win the game} \end{cases} \tag{4.1}$$

$$s_x := \begin{cases} 0 & \text{if occupied position} \\ 1 & \text{if free position} \end{cases} \tag{4.2}$$

The equation 4.1 represents the v values for the output file. This file is used to train the neural network and the equation 4.2 represents each position (x) of the matrix for the policy network.

4.6.1 Neural Network Structure

The structure of the neural network was defined with 4 hidden layers. Each layer with the following functions:

1. Dropout
2. Activate functions: ReLu and Batch Normalization

```
dropout = 0.3
state = layers.Input(shape=(7, 4), name="state_7_4")
size = 7 * 4
flat = layers.Flatten(name="flatten")(state)
s_fc1 = layers.Dropout(dropout, name="drop1")(layers.Activation('relu',
    name="act1")(layers.BatchNormalization(axis=1, name="batch1")(layers.
    Dense(1024, name="dense1")(flat))))
s_fc2 = layers.Dropout(dropout, name="drop2")(layers.Activation('relu',
    name="act2")(layers.BatchNormalization(axis=1, name="batch2")(layers.
    Dense(1024, name="dense2")(s_fc1))))
s_fc3 = layers.Dropout(dropout, name="drop3")(layers.Activation('relu',
    name="act3")(layers.BatchNormalization(axis=1, name="batch3")(layers.
    Dense(1024, name="dense3")(s_fc2))))
s_fc4 = layers.Dropout(dropout, name="drop4")(layers.Activation('relu',
    name="act4")(layers.BatchNormalization(axis=1, name="batch4")(layers.
    Dense(512, name="dense4")(s_fc3))))
pi = layers.Dense(size, activation='softmax', name='policy_net')(s_fc4)
v = layers.Dense(1, activation='tanh', name='value\_net')(s_fc4)
model = models.Model(inputs=state, outputs=[pi, v], name="noConv7x4")
model.summary()
```

Figure 4.4: Neural Network Model (Aguayo, 2020)

The network was compiled with loss functions as described on AlphaZero documentation (Silver D., 2017): categorical entropy for the policy network and mean squared error for the policy network 3.1.

All configuration for the gameplay was defined by the JSON file (figure 4.6 example of JSON file).

4.7 Related work integration

The integration between this game and Aguayo's game needed a converter for different state configurations to provide a dispute between both games (see B in figure 4.1).

4.7.1 API

The interface has two components: an API for communication and Javascript code with some new functions.

The Javascript code was injected in the Aguayo's web page to overwrite the human player who used the API to create a new challenge to this game. It was possible to

```
Model: "noConv7x4"
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
state_7_4 (InputLayer)       [(None, 7, 4)]            0
_________________________________________________________________________________________
flatten (Flatten)            (None, 28)                0           state_7_4[0][0]
_________________________________________________________________________________________
dense1 (Dense)               (None, 1024)              29696       flatten[0][0]
_________________________________________________________________________________________
batch1 (BatchNormalization)  (None, 1024)              4096        dense1[0][0]
_________________________________________________________________________________________
act1 (Activation)            (None, 1024)              0           batch1[0][0]
_________________________________________________________________________________________
drop1 (Dropout)              (None, 1024)              0           act1[0][0]
_________________________________________________________________________________________
dense2 (Dense)               (None, 1024)              1049600     drop1[0][0]
_________________________________________________________________________________________
batch2 (BatchNormalization)  (None, 1024)              4096        dense2[0][0]
_________________________________________________________________________________________
act2 (Activation)            (None, 1024)              0           batch2[0][0]
_________________________________________________________________________________________
drop2 (Dropout)              (None, 1024)              0           act2[0][0]
_________________________________________________________________________________________
dense3 (Dense)               (None, 1024)              1049600     drop2[0][0]
_________________________________________________________________________________________
batch3 (BatchNormalization)  (None, 1024)              4096        dense3[0][0]
_________________________________________________________________________________________
act3 (Activation)            (None, 1024)              0           batch3[0][0]
_________________________________________________________________________________________
drop3 (Dropout)              (None, 1024)              0           act3[0][0]
_________________________________________________________________________________________
dense4 (Dense)               (None, 512)               524800      drop3[0][0]
_________________________________________________________________________________________
batch4 (BatchNormalization)  (None, 512)               2048        dense4[0][0]
_________________________________________________________________________________________
act4 (Activation)            (None, 512)               0           batch4[0][0]
_________________________________________________________________________________________
drop4 (Dropout)              (None, 512)               0           act4[0][0]
_________________________________________________________________________________________
policy_net (Dense)           (None, 28)                14364       drop4[0][0]
_________________________________________________________________________________________
value_net (Dense)            (None, 1)                 513         drop4[0][0]
=========================================================================================
Total params: 2,682,909
Trainable params: 2,675,741
Non-trainable params: 7,168
_________________________________________________________________________________________
```

Figure 4.5: Neural Network Summary

```
settings = {
  "Uct": 1.41,                    -> Standard MCTS UCT (UCB for trees)
  "Ucb": 1,                       -> Puct for NN
  "SimulationsP1": 100,           -> MCTS simulation agent 1
  "SimulationsP2": 100,           -> MCTS simulation agent 2
  "MillisecondsP1": 5000,         -> Thinking time agent 1
  "MillisecondsP2": 5000,         -> Thinking time agent 2
  "Bonus": 3,                     -> Bonus for standard MCTS
  "Boxes": 3,                     -> Board configuration
  "NumberOfGames": 50,            -> Games to play
  "Agent1Type": 4,                -> Agent type: 3 = MCTS, 4 = Neural Network
  "Agent1Cnn": 0,                 -> Network version
  "Agent2Type": 4,
  "Agent2Cnn": 1,
  "OutputFolder": "Results",
  "OutputFile": "settings-0",
  "OutputId" : "false",           -> Adds an Id to file
  "Mode": 0                       -> Game Mode: 0 Competitive, 1 Training
}
```

Figure 4.6: Configuration File

integrate both responses and requests, using the independent implementation of each game.

That integration was done only to make possible the dispute proposed; it does not change any original game component related to the game expertise.

API endpoints

The GET verb returns a GUID (Global Unique Identifier) which identifies the current player.

The "Getplayerturn()" method of an specific "gameid" returns 1 for player 1 and 2 for player 2.

POST for the game turn has the following structure (figure 4.7).

The API will convert the format from Aguayo's board to my board and plays that in the board. The result, or next move of the game plays in the Javascript implementation through JavaScript function **play(<position>)**. The position will be converted again to Aguayo's format and it plays in the JavaScript game.

The game looping continue until all those free positions on the board have finished.

```
{
    "position":"7",
    "board":"000000000000000001000000000",
    "fullBoard":"\r\n
          a   b   c  \r\n
        o   o   o   o A\r\n
      A                \r\n
        o   o   o   o B\r\n
      B                \r\n
        o   o---o   o C\r\n
      C                \r\n
        o   o   o   o D\r\n
        a   b   c   d  \r\n"
}
```

Figure 4.7: Json turn object to play in position 7 (Aguayo's board version)

Internal

The gameplay through the API is slow so the new internal solution was developed. This solution is simplest than API because uses the Aguayo's pretrained network inside of my implementation in my own code, making possible the dispute between the networks without the Javascript integration.

5

Results and discussions

5.1 Execution pipeline

5.1.1 Data generation for training

The data was generated through gameplays between different versions of the model and the standard MCTS. The first group of data was generated for the 3 x 3 board, playing itself using only the standard Monte Carlo Tree Search algorithm, which does a rollout of many games until the end to choose the best move. One set of 500 games was generated by that algorithm in 16 hours of work in a cloud free-tier virtual machine; a total of 192,000 boards were generated as the result of those 500 games as below:

The 192,000 initial boards were used as the first input to train the first version of the network. The neural network was trained at the proportion of 80% of data for the

```
500 Games\newline
 24 Boards\newline
  8 Symmetries\newline
  2 Sets\newline
000 Total Boards
```

Figure 5.1: Total Boards

training set and 20% of data for testing set.

This first CNN version was generated in Colab and the saved network was transported to the second step.

5.1.2 Competition and evolution

The second step began with the competition of the same version of the neural network versions called 0 and 1. After 50 games played, the best network, or the network that wins most of the time, become the new candidate for training.

The data generated for the new version was created in the training mode option of the game with the winner from the previous step playing on itself. More 100 games were generated, creating more 38,400 boards (100 x 24 x 8 x 2 = 38,400) for each version with 100 MCTS simulations.

The new neural network was trained with the data coming from the last step, with the configuration:

1. LR = 0.001
2. Batch size = 64
3. Epochs = 200

The training history is saved in pandas dataframe and the new neural network is ready for the challenge with the neural network winner version from the last competition.

Again, both neural networks play more 50 games, the winner neural network generates more 100 games in training mode and trains it for more 200 epochs, and repeats the same steps several times.

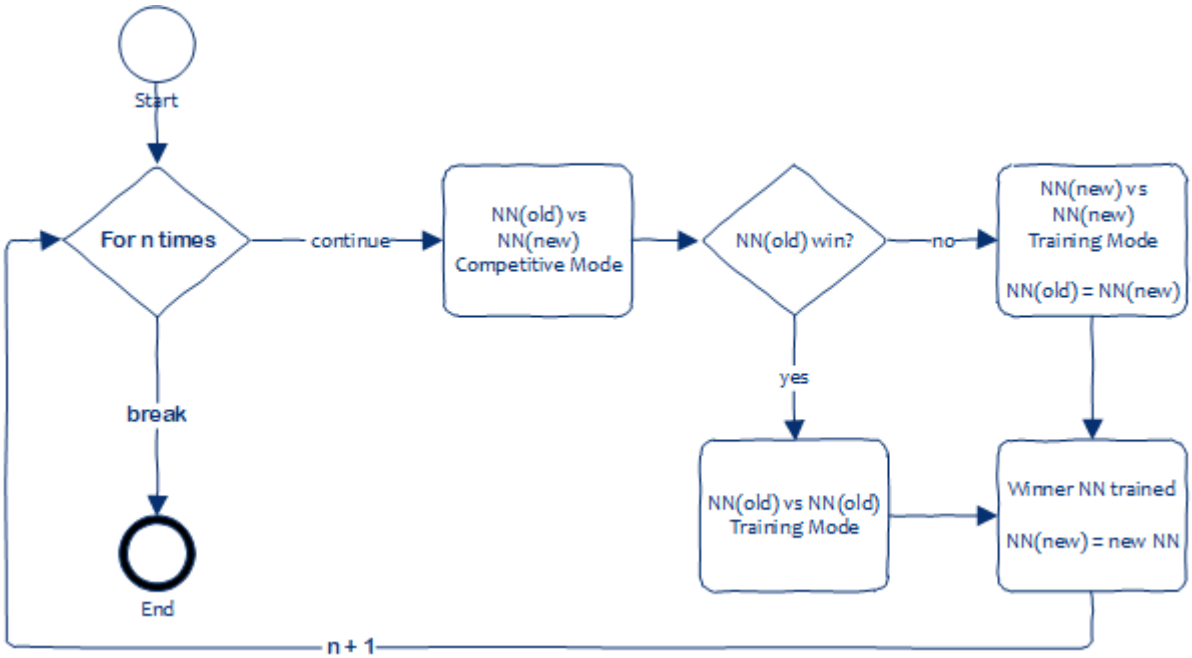


Figure 5.2: Training diagram

5.2 Results and discussions

The results were divided in two subsets as follows:

5.2.1 Subset A

This set is related to the first execution of this game. Sixty versions of neural networks were created in the training processing in this subset; each training step with 38,400 boards in the proportion of 80% for training and 20% for testing. The network evolution occurred as expected. Tables 5.1 and 5.2 have the details of all training processing, including the network evolution and times elapsed.

The first column of the table contains the version of the each neural network used in the dispute, followed by the points of the first neural network and the points of the opponent neural network. The neural network winner version column contains which version of the neural networks won the dispute of 50 games.

The last three columns contain the elapsed time to play 50 games (dispute), the elapsed time to play more 100 games with the winner (data generation for training), and the elapsed time to train the new neural network in 200 epochs.

The total of training was 12000 epochs of training divided in 60 new neural networks. Figures 5.3 for policy network and 5.4 for the value networks shows the evolution of the neural networks. The x axis represents the neural network version.

5.2.2 Related work results

The tables 5.3 and 5.4 contain the results of 2 game competitions of 4 rounds each (20 games per round) between this neural network and Aguayo's neural network.

The first game competition was between the 60^{th} neural network version and Aguayo's neural network and achieved the percentage of 35% and 45% of the total of victories over Aguayo's neural network. Aguayo's neural network demonstrates to be more efficient in those games.

The second game competition was between the 60^{th} neural network version and the original MCTS. The improvement achieved was around of 68% of the total of victories over MCTS, a great learning evolution from the original game.

However, the evolution of the network has occurred, the last version of the network generated was not sufficiently good to defeat the Aguayo's opponent 100% of time, but it demonstrated a great progress faced the MCTS original (tables 5.1 and 5.2).

5.2.3 Subset B

The subset B was created with more games to find some relation between time, number of games, and more training and a more efficient network.

One hundred versions of the neural networks were created in the training processing in this subset, almost the double of the first subset; each training step with 76,800 boards (double of the first one) in the proportion of 80% for training and 20% for testing.

The data generated for the new version was created in the training mode option of the game as the first subset but with the double of the boards generated: 76,800 boards generated for each version (200 x 24 x 8 x 2 = 76,800) with 160 MCTS simulations.

NN version	First NN victories (0 - 50)	Second NN victories (0 -50)	Winner NN version	Time		
				50 games competitive mode	100 games training mode	NN Training 200 epochs
0 vs 1	36	14	1	00:35:48	01:15:32	00:44:08
0 vs 2	17	33	2	00:30:44	00:52:30	00:42:09
2 vs 3	32	18	3	00:21:53	00:53:51	00:41:28
3 vs 4	8	42	4	00:21:39	00:51:25	00:41:01
4 vs 5	13	37	5	00:19:40	00:48:38	00:41:10
5 vs 6	17	33	6	00:18:54	00:45:52	00:42:10
6 vs 7	19	31	7	00:18:54	00:46:28	00:41:19
7 vs 8	20	30	8	00:17:37	00:44:12	00:41:25
8 vs 9	20	30	9	00:17:25	00:44:24	00:41:09
9 vs 10	28	22	9	00:17:47	00:43:23	00:41:11
9 vs 11	19	31	11	00:17:33	00:42:35	00:41:05
11 vs 12	18	32	12	00:16:40	00:47:03	00:42:28
12 vs 13	25	25	13	00:16:20	00:42:09	00:41:27
12 vs 14	11	39	14	00:16:44	00:42:35	00:42:01
14 vs 15	26	24	14	00:17:48	00:44:23	00:41:11
14 vs 16	9	41	16	00:17:00	00:42:03	00:41:28
16 vs 17	25	25	17	00:17:25	00:42:57	00:41:11
16 vs 18	12	38	18	00:16:47	00:42:21	00:41:50
18 vs 19	16	34	19	00:17:34	00:42:50	00:41:20
19 vs 20	19	31	20	00:16:58	00:42:27	00:40:43
20 vs 21	13	37	21	00:15:40	00:39:15	00:37:48
21 vs 22	16	34	22	00:15:50	00:37:59	00:36:48
22 vs 23	0	50	23	00:16:30	00:38:25	00:36:15
23 vs 24	24	26	24	00:14:08	00:37:57	00:36:21
24 vs 25	15	35	25	00:15:20	00:39:15	00:36:41
25 vs 26	12	38	26	00:14:44	00:38:26	00:35:56
26 vs 27	28	22	26	00:14:33	00:38:47	00:36:32
26 vs 28	17	33	28	00:14:55	00:38:41	00:36:08
28 vs 29	0	50	29	00:14:12	00:39:19	00:36:02
29 vs 30	22	28	30	00:14:17	00:38:03	00:36:26

Table 5.1: Network evolution results - 0 to 30 NN versions

NN version	First NN victories (0 - 50)	Second NN victories (0 -50)	Winner NN version	Time		
				50 games competitive mode	100 games training mode	NN Training 200 epochs
30 vs 31	24	26	31	00:13:59	00:39:21	00:36:18
31 vs 32	19	31	32	00:14:17	00:37:40	00:36:12
32 vs 33	6	44	33	00:14:07	00:38:15	00:36:26
33 vs 34	1	49	34	00:14:31	00:38:09	00:36:31
34 vs 35	15	35	35	00:14:59	00:38:33	00:36:29
35 vs 36	23	27	36	00:14:30	00:38:30	00:37:40
36 vs 37	20	30	37	00:15:13	00:37:27	00:36:12
37 vs 38	15	35	38	00:14:04	00:38:05	00:36:05
38 vs 39	13	27	39	00:14:03	00:38:59	00:37:04
39 vs 40	17	33	40	00:14:31	00:37:26	00:36:07
40 vs 41	20	30	41	00:14:36	00:40:14	00:39:39
41 vs 42	21	29	42	00:13:13	00:34:55	00:36:29
42 vs 43	21	29	43	00:13:52	00:35:37	00:34:39
43 vs 44	18	32	44	00:12:49	00:35:13	00:34:02
44 vs 45	12	38	45	00:13:14	00:35:10	00:33:57
45 vs 46	20	30	46	00:13:01	00:34:56	00:34:13
46 vs 47	18	32	47	00:13:08	00:34:10	00:33:35
47 vs 48	20	30	48	00:12:47	00:33:52	00:34:10
48 vs 49	12	38	49	00:12:48	00:33:38	00:33:53
49 vs 50	12	38	50	00:12:42	00:34:03	00:34:37
50 vs 51	6	44	51	00:13:15	00:33:42	00:34:20
51 vs 52	8	42	52	00:12:43	00:34:26	00:34:20
52 vs 53	23	27	53	00:13:35	00:34:26	00:34:30
53 vs 54	16	34	54	00:13:09	00:34:29	00:34:31
54 vs 55	20	30	55	00:12:45	00:34:30	00:35:00
55 vs 56	26	24	55	00:12:46	00:34:43	00:35:13
55 vs 57	12	38	57	00:12:52	00:36:35	00:34:57
57 vs 58	15	35	58	00:14:36	00:38:51	00:38:43
58 vs 59	10	40	59	00:13:04	00:37:23	00:37:45
59 vs 60	15	35	60	00:13:10	00:34:02	00:50:03

Table 5.2: Network evolution results - 31 to 60 NN versions

Neural Networks subsetA v60	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes NN	9	7	7	9
P2 - Aguayo's NN	11	13	13	11

Table 5.3: Victories in 4 rounds of 20 games each - NN subsetA v60 vs Aguayo's NN

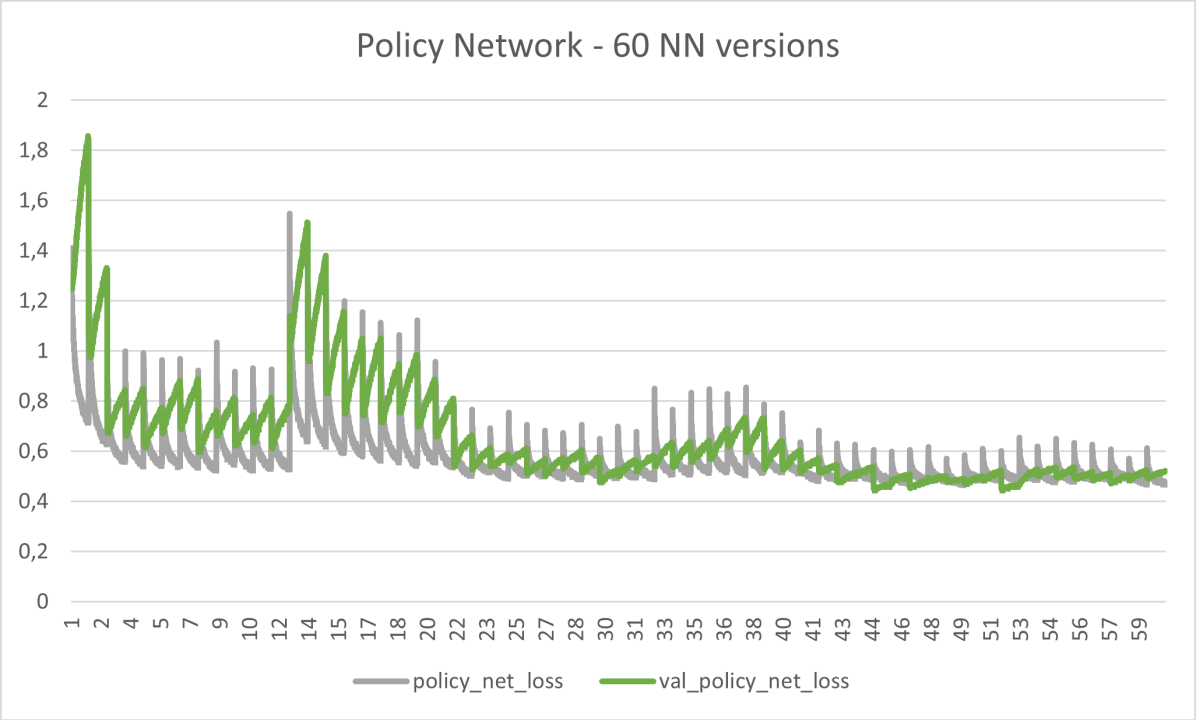


Figure 5.3: Policy network evolutions (60 NN versions)

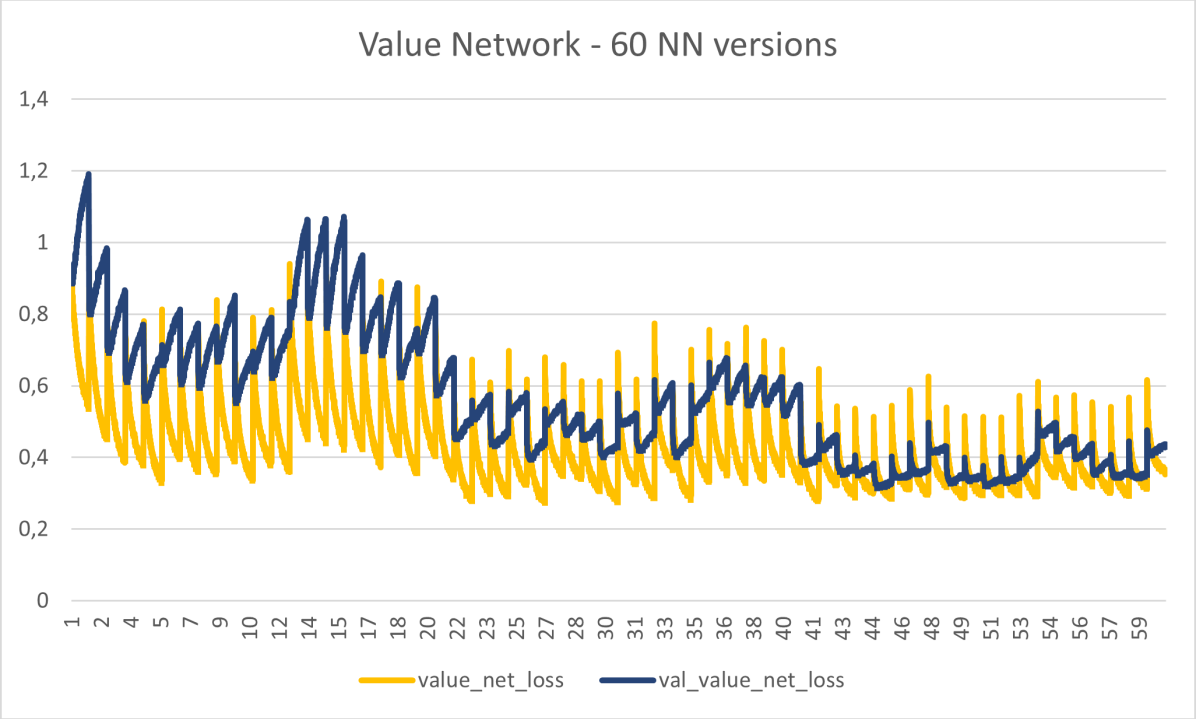


Figure 5.4: Value network evolutions (60 NN versions)

Neural Networks subsetA v60	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes MCTS	5	6	5	9
P2 - DotsAndBoxes NN	15	14	15	11

Table 5.4: Victories in 4 rounds of 20 games each - MCTS vs NN subsetA v60

Neural Networks subsetB v100	20 Games	20 Games	20 Games	20 Games
P1 - Aguayo's NN	1	0	0	1
P2 - DotsAndBoxes NN	19	20	20	19

Table 5.5: Victories in 4 rounds of 20 games each - Aguayo's NN vs NN subsetB v100

Neural Networks subsetB v100	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes NN	15	11	12	14
P2 - Aguayo's NN	5	9	8	6

Table 5.6: Victories in 4 rounds of 20 games each - NN subsetB v100 vs Aguayo's NN

The new neural network was trained with the same parameters used before, only changing the epoch number to 350, as below:

1. LR = 0.001

2. Batch size = 64

3. Epochs = 350

The results of training occurred as expected, and the network evolution from 1 to 100 versions took 42 days (figures 5.8, 5.6).

Results are presented sometimes in two ways. For example, tables 5.5 and 5.6 contain results from the same agents, but, in the table 5.5, Aguayo's NN was the first player, and in the table 5.6, it was the second one.

Subset B demonstrates a little competitive advantage of player two over player one in the same player's game. Those can be confirmed with the results in the table 5.5 and 5.6, and 5.7 and 5.8. This behavior could not be observed in some other cases, so this was not conclusive.

Neural Networks subsetB v100	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v100	16	17	16	15
P2 - DotsAndBoxes v60	4	3	4	5

Table 5.7: Victories in 4 rounds of 20 games each - NN subsetB v100 vs NN subsetB v60

Number of games	Agent1 Type	Agent1 Cnn	Agent2 Type	Agent2 Cnn	Mode	Agent1 Victories	Agent2 Victories	Winner	Net Version
200	4	0	4	1	0	132	68	1	0
200	4	0	4	2	0	46	154	2	2
200	4	2	4	3	0	106	94	1	2
200	4	2	4	4	0	77	123	2	4
200	4	4	4	5	0	95	105	2	5
200	4	5	4	6	0	34	166	2	6
200	4	6	4	7	0	61	139	2	7
200	4	7	4	8	0	94	106	2	8
200	4	8	4	9	0	67	133	2	9
200	4	9	4	10	0	95	105	2	10
200	4	10	4	11	0	81	119	2	11
200	4	11	4	12	0	97	103	2	12
200	4	12	4	13	0	81	119	2	13
200	4	13	4	14	0	98	102	2	14
200	4	14	4	15	0	81	119	2	15
200	4	15	4	16	0	81	119	2	16
200	4	16	4	17	0	78	122	2	17
200	4	17	4	18	0	81	119	2	18
200	4	18	4	19	0	110	90	1	18
200	4	18	4	20	0	83	117	2	20
200	4	20	4	21	0	81	119	2	21
200	4	21	4	22	0	37	163	2	22
200	4	22	4	23	0	86	114	2	23
200	4	23	4	24	0	81	119	2	24
200	4	24	4	25	0	45	155	2	25
200	4	25	4	26	0	54	146	2	26
200	4	26	4	27	0	52	148	2	27
200	4	27	4	28	0	84	116	2	28
200	4	28	4	29	0	55	145	2	29
200	4	29	4	30	0	84	116	2	30

Figure 5.5: Network evolution - first 30 versions

Neural Networks subsetB v100	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v60	1	2	1	1
P2 - DotsAndBoxes v100	19	18	19	19

Table 5.8: Victories in 4 rounds of 20 games each - NN subsetB v60 vs NN subsetB v100

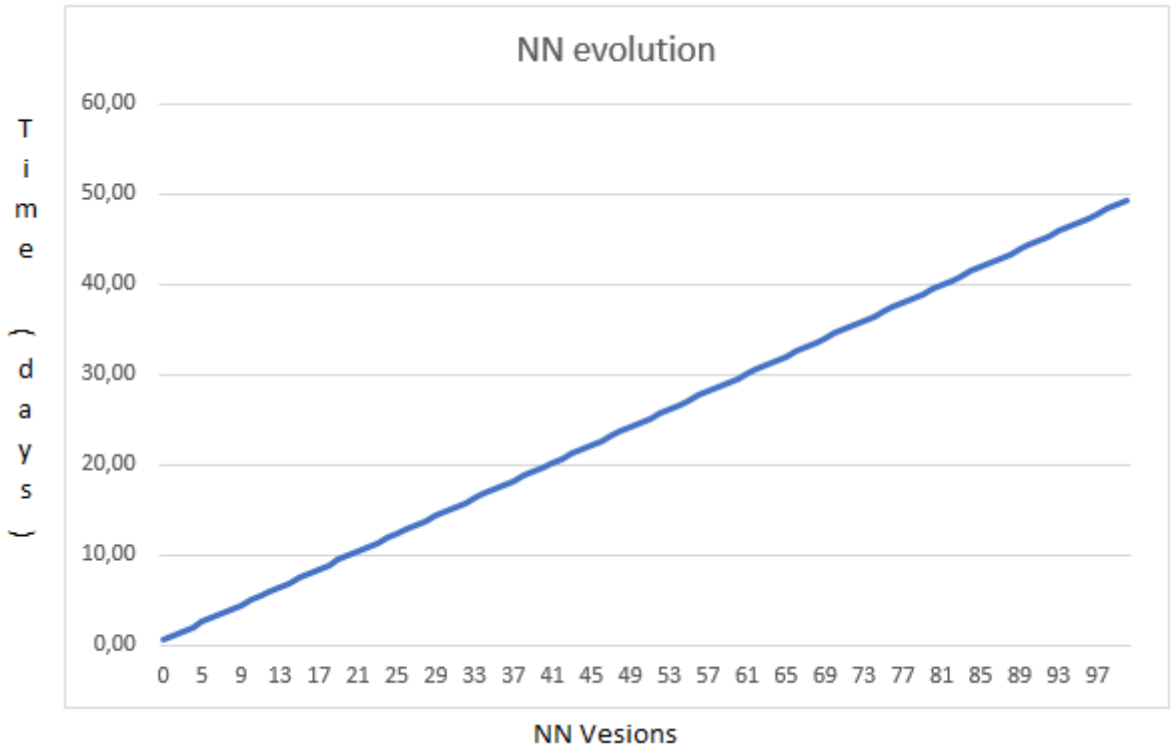


Figure 5.6: Network evolution - 100 versions / 49,31 Days

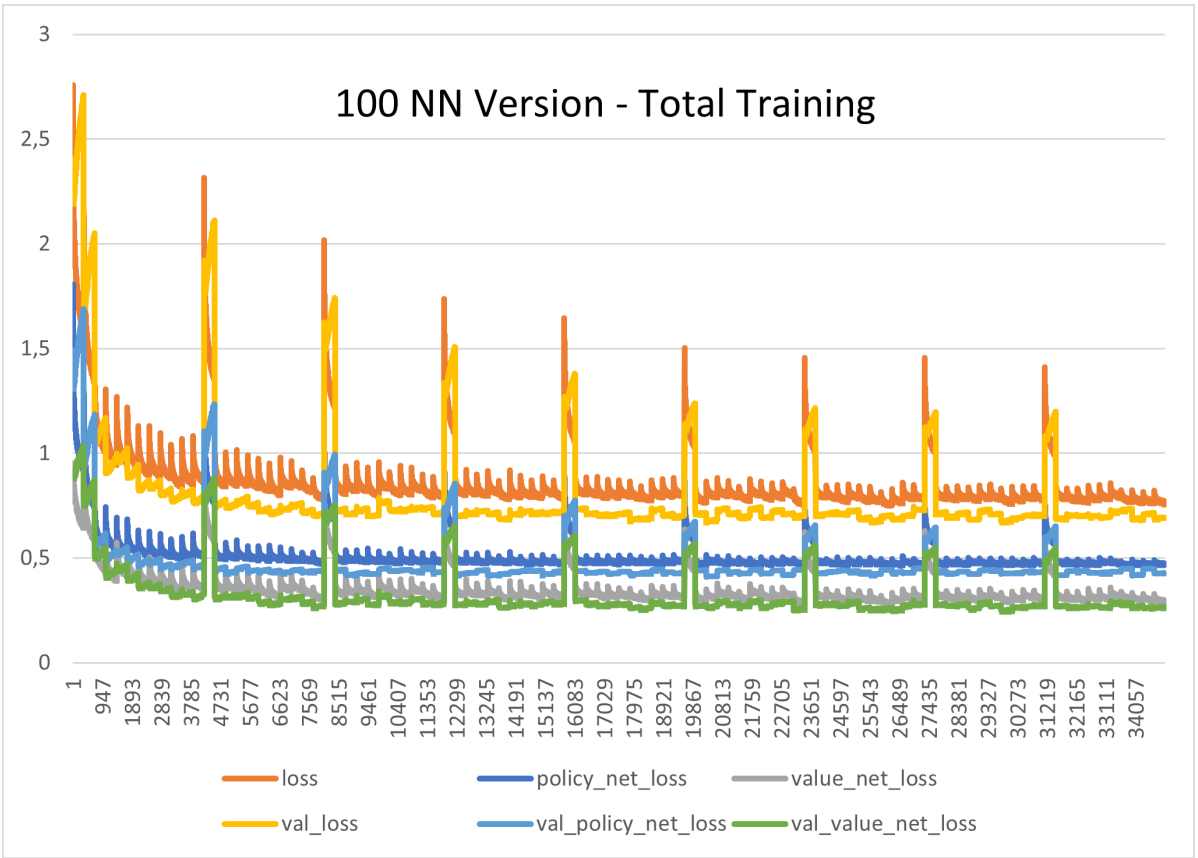


Figure 5.7: Network 100v - Total Training

Neural Networks subsetC v40/24	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v24	15	15	16	18
P2 - DotsAndBoxes v60	5	5	4	2

Table 5.9: Victories in 4 rounds of 20 games each - NN subsetC v40/24 vs NN subsetB v60

Neural Networks subsetC v40/24	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v60	3	0	3	1
P2 - DotsAndBoxes v24	17	20	17	19

Table 5.10: Victories in 4 rounds of 20 games each - NN subsetB v60 vs NN subsetC v40/24

5.2.4 Other subsets

Others subsets were generated during this work, but no relevant changes were presented in any of them. The purpose of those subsets was to confirm if the random initial weight makes some differences in training process or if only the time of training is relevant; what was confirmed.

All subsets have demonstrated similar performance for the same training effort.

However, one special subset was trained for 5 days, generating 40 neural network versions, with one modification that shows to be more powerful than subset B.

The original MCTS for Alphazero has one problem related to this game. This problem is the fact of the player can play again if the point is made in the current move. Therefore, the last modification in MCTS implemented with this modification, the training process increases quality in the versions of this subset.

Subset C

The subset C was trained in 2 moments, one of 17 versions and another of 24. The 24^{th} version has as initial weights the weights from 17^{th} version.

This subset demonstrates high performance comparatively to subset B as presented bellow:

However, with the improvement, v40/24 could not defeat the stronger v100 in most of the games. The total training time was 72 hours and that confirms one more time

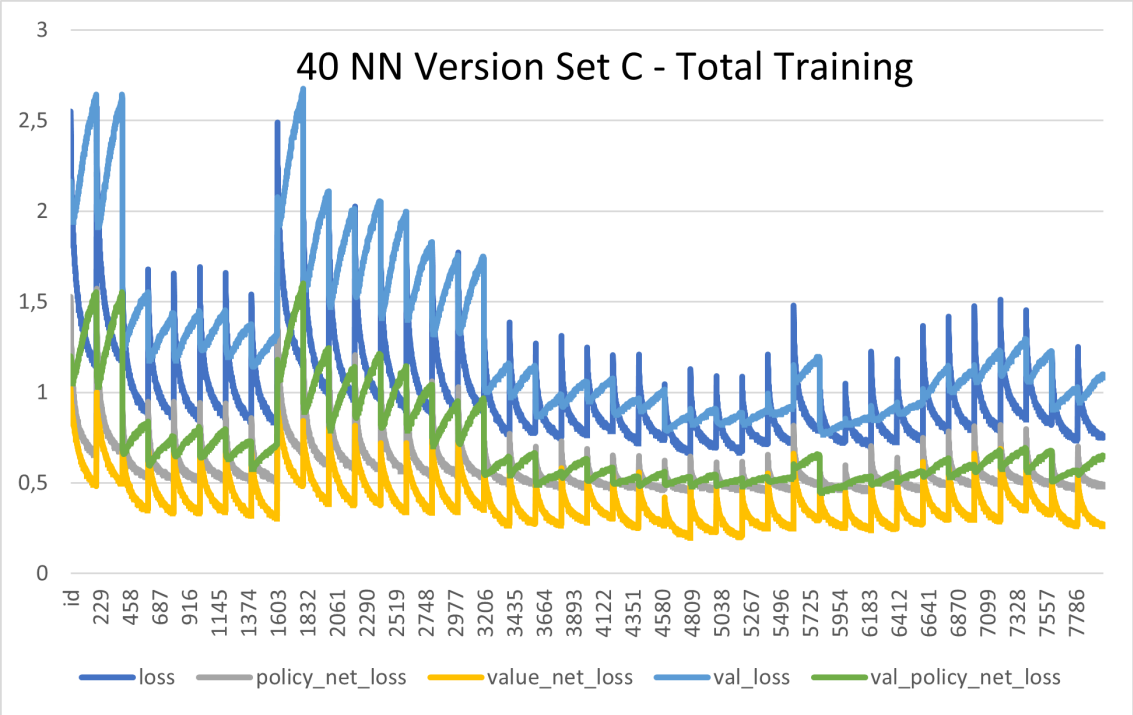


Figure 5.8: Network v40 - Total Training

Neural Networks subsetC v40/24	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v24	4	5	4	7
P2 - DotsAndBoxes v100	16	15	16	13

Table 5.11: Victories in 4 rounds of 20 games each - NN subsetC v40/24 vs NN subsetB v100

that the training time has the most relevant influence over the learning process in the Alphazero algorithm.

Neural Networks subsetC v40/24	20 Games	20 Games	20 Games	20 Games
P1 - DotsAndBoxes v100	8	5	11	3
P2- DotsAndBoxes v24	12	15	9	17

Table 5.12: Victories in 4 rounds of 20 games each - NN subsetC v40/24 vs NN subsetB v100

6

Conclusion and Future Work

That is an exciting work to me because I could improve my old program and it got smarter through machine learning techniques.

Although the performance of the network did not achieve unbreakable results yet, it demonstrates the process worked very well with time training the network. The process of obtaining, training, generating and exporting data and using a neural network is pretty good, but it has a strong dependency of finding the best hyper-parameters which takes computational and time resources which were very limited for the thesis.

However, the two subsets exposed in the last chapter were generated with random initial weights, demonstrating the only difference of those two subsets were the time of training. Only more training time was able to improve the network quality.

The tests demonstrated that it is possible to work with the neural network and

ML.net framework, VM's and Colab Python notebook for training, generating data and make some predictions as the AlphaZero demonstrates to be possible.

Using the same network, a simple network proposed by Aguayo ((Aguayo, 2020)), was demonstrated the only training time was sufficient to improve the neural network to a level above the original one.

6.1 Problems faced

Some problems were faced in all the thesis process:

1. AlphaZero was created with more computational power than I have available. It is possible to achieve good results, but this requires a large amount of training time. For example the GPU computation is the fastest way to train the network and despite being possible to train it using only the CPU card, it takes many more hours to finish, and takes much more time to finish the tests. If something is wrong with the software, or the VM, the time waste to verify, to correct and to retrain is simply too much.

2. The Colab solution (free GPU card available) was a good option but only to tasks that require few time and supervised execution. If the browser is closed for more than 15 minutes or PC is idle or after 8 hours of continuous use (few days in the paid version), the task stops and all progress is lost. Often, the training process takes around a week or more, so Colab was not the option.

3. The VM cost was another great problem faced. The basic free VM from the cloud is very slow to do this training process. The machine crashed after the first hours. The costs for the most sophisticate machines is very high, forcing the use of medium configuration machines, and the time was increased too.

4. As my neural network was not able to defeat the opponents (100% of time) only more changes and retrain is necessary to better understand the problem cause, if

other implementations were more efficient than mine, or if it was a matter of the training time, computational power and network hiper-parameters.

5. No versions of the a convolutional network worked properly. More studies and time will be necessary to implement this network type.

6.2 Future work

Three important steps could be used in the future to improve the quality of the network and possibly achieve better results:

1. In order to defeat some others opponents, it will be necessary more time of training and more changes in the hyperparameters to train the network properly. Network changes can be pleased too. The original AlphaGo Zero takes 40 days of disputing and training until it become invincible, with a 4TPU machine, in the Go game. So, more training and adjustments could improve the game quality.

2. The creation of the convolutional neural network model could help to achieve better results instead the use of a neural network without convolutional layers. Although ((Peters, 2018)) used convolutional neural network in his thesis, his games was not able to defeat some opponents in some games. However, adjusting the hiper-parameters and increasing training time could be a good attempt to achieve better results with a CNN.

3. At least, the use of a GPU to make the whole process of dispute and training could be a attempt to save time and get better results. For the dispute, the parallel methods should be implemented in the game, so not only the training will be improved, but the game dispute too. Using the paralleling techniques presented in AlphaZero paper (Silver D., 2017), it is possible to distribute tasks between the training and data generation, saving time in this process. It should be confirmed if the performance of MCTS will be increase with use of one single PC instead of a cluster of computers used in some version of AlphaGo.

4. Create an arena to dispute with QDab and Dabble games. The interface to make a competition with those are a little more complex to implement because some changes must be done in their own code to access a rest API or through sockets for example. Those implementations are very old what makes this task more difficult.

Bibliography

C. Aguayo. Alphazero, a novel reinforcement learning algorithm, in javascript., 2020. https://towardsdatascience.com/alphazero-a-novel-reinforcement-learning-algorithm-deployed-in-javascript-56018503ad18, Last accessed on 2021-02-04.

H. Baier. Monte-carlo tree search enhancements for one-player and two-player domains. Technical report, Netherlands, 2015.

G. Bonaccorso. *Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning*. Packt Publishing, 2017. ISBN 1785889621.

M. E. Elkind E. *17 - Game-theoretic Foundations of Multiagent Systems from Multiagent Systems*. MIT Press, 2011.

D. Ferger. A continuous mapping theorem for the argmax-functional in the non-unique case. *Statistica Neerlandica*, 58(1):83 – 96, 2004. ISSN 00390402. URL `https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=12010914&lang=pt-pt&site=ehost-live&scope=site`.

P. Gabrijelčič. *Mastering Delphi Programming: A Complete Reference Guide: Learn all about building fast, scalable, and high performing applications with Delphi*. Packt Publishing, 2019. ISBN 9781838983918. URL `https://books.google.pt/books?id=r4LADwAAQBAJ`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

J. P. Grossman. Dabble - ai for dots and boxes, 2000. presented on the second Combinatorial Game Theory Research Workshop, at the Mathematical Sciences Research Institute, http://www.mathstat.dal.ca/ jpg/dabble/, Last accessed on 2021-02-04.

A. Gulli and S. Pal. *Deep Learning with Keras*. Packt Publishing, 2017.

K. T. Ide H. *Improvement of learning for CNN with ReLU activation by sparse regularization.*, pages 2684–2691. 2017.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. Technical report, 2015.

A. Krogh. What are artificial neural networks? *Nat Biotechnol*, 26:195–197, 2008 2008.

M. A. Mercioni and S. Holban. The most used activation functions: Classic versus current. In *2020 International Conference on Development and Application Systems (DAS)*, pages 141–145, 2020. doi: 10.1109/DAS49615.2020.9108942.

G. A. e. a. Nwankpa C., Ijomah W. Activation functions: Comparison of trends in practice and research for deep learning. Technical report, 2018.

T. V. Peters. Mastering the game of dots and boxes with deep neural networks and tree search. Master's thesis, University of Bremen, Germany, 2018. Advisors: Drechsler, Prof. Dr. Rolf; Meine, Dr. Hans.

J. Prince. Game specific approaches to monte carlo tree search for dots and boxes. 2017.

A. Ruano. *Artificial Neural Network*. Centre for Intelligent Systems, University of Algarve, Faro, Portugal, 2016.

C. Sammut and G. Webb. *Encyclopedia of Machine Learning*. 2017. ISBN 978-0-387-30768-8. doi: 10.1007/978-1-4899-7687-1.

A. Santos. Monte carlo tree search experiments in hearthstone. Technical report, 2017.

S. K. e. a. Silver D., Schrittwieser J. Mastering the game of go without human knowledge. *Nature*, pages 354–359, 2017 2017.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Y. Zhang, S. Li, and X. Xiong. A study on the game system of dots and boxes based on reinforcement learning. pages 6319–6322, Nanchang, China, 2019.

Y. Zhuang. Dots-and-boxes. a simple dots and boxes ai., 2015. `http://dotsandboxes.tar.xyz/`, Last accessed on 2021-02-04.

A

Samples

A.1 Gameplay Sample

This is a sample of the gameplay dump[1] used in the training process with all metrics:

1-1 PC/17 | Next: 2 PC/17 | Pts: 0 | Root(v2313/q-136.58878/w-287.718/p0)

Winner: cA(v183/q0/w0/p0.040288422) »

aA(v115/q0/w0/p0.040031295) bA(v47/q0/w0/p0.042764116)

cA(v183/q0/w0/p0.040288422) Aa(v92/q0/w0/p0.04184781)

Ab(v137/q0/w0/p0.04358654) Ac(v70/q0/w0/p0.042818278)

Ad(v93/q0/w0/p0.041306034) aB(v93/q0/w0/p0.0431126)

[1] All dump files can be found in a Github project repository `https://github.com/gionnani/dotsandboxes`

bB(v93/q0/w0/p0.037944812) cB(v116/q0/w0/p0.04437638)
Ba(v24/q0/w0/p0.04160703) Bb(v70/q0/w0/p0.03836427)
Bc(v24/q0/w0/p0.038355287) Bd(v162/q0/w0/p0.04381453)
aC(v47/q0/w0/p0.043455698) bC(v93/q0/w0/p0.038355198)
cC(v161/q0/w0/p0.04249355) Ca(v115/q0/w0/p0.04167688)
Cb(v115/q0/w0/p0.043465793) Cc(v93/q0/w0/p0.04325168)
Cd(v91/q0/w0/p0.04131624) aD(v93/q0/w0/p0.041861136)
bD(v92/q0/w0/p0.04393054) cD(v93/q0/w0/p0.0399758)

```
     a   b   c
   o   o   o---o A
 A
   o   o   o   o B
 B
   o   o   o   o C
 C
   o   o   o   o D
   a   b   c   d
```

2-2 PC/17 | Next: 1 PC/17 | Pts: 0 | Root(v2194/q-110.494255/w-242.65576/p0)
Winner: aD(v261/q0/w0/p0.042993095) »
aA(v45/q0/w0/p0.046434067) bA(v45/q0/w0/p0.046149526)
Aa(v67/q0/w0/p0.04498107) Ab(v67/q0/w0/p0.04072039)
Ac(v111/q0/w0/p0.046871047) Ad(v111/q0/w0/p0.048296433)
aB(v89/q0/w0/p0.039480664) bB(v130/q0/w0/p0.042238094)
cB(v67/q0/w0/p0.043463137) Ba(v111/q0/w0/p0.046082843)
Bb(v67/q0/w0/p0.043011703) Bc(v67/q0/w0/p0.04036699)
Bd(v89/q0/w0/p0.04032224) aC(v23/q0/w0/p0.039025698)
bC(v89/q0/w0/p0.041102294) cC(v153/q0/w0/p0.038830735)

Ca(v121/q0/w0/p0.04546082) Cb(v89/q0/w0/p0.041936103)
Cc(v65/q0/w0/p0.045346517) Cd(v196/q0/w0/p0.047058802)
aD(v261/q0/w0/p0.042993095) bD(v107/q0/w0/p0.045183625)
cD(v23/q0/w0/p0.044644102)

```
     a   b   c
   o   o   o---o A
 A
   o   o   o   o B
 B
   o   o   o   o C
 C
   o---o   o   o D
   a   b   c   d
```

3-1 PC/17 | Next: 2 PC/17 | Pts: 0 | aD(v1916/q-120.39701/w-250.14388/p0.042993095)
Winner: Ad(v189/q0/w0/p0.050594784) »
aA(v88/q0/w0/p0.03533924) bA(v126/q0/w0/p0.04490071)
Aa(v124/q0/w0/p0.048420236) Ab(v0/q0/w0/p0.050569266)
Ac(v151/q0/w0/p0.048575755) Ad(v189/q0/w0/p0.050594784)
aB(v22/q0/w0/p0.041192673) bB(v158/q0/w0/p0.043849535)
cB(v141/q0/w0/p0.047411405) Ba(v62/q0/w0/p0.0458468)
Bb(v64/q0/w0/p0.045353036) Bc(v102/q0/w0/p0.046580978)
Bd(v57/q0/w0/p0.047312286) aC(v59/q0/w0/p0.040053766)
bC(v64/q0/w0/p0.045510218) cC(v77/q0/w0/p0.046656672)
Ca(v37/q0/w0/p0.048815463) Cb(v102/q0/w0/p0.05110715)
Cc(v80/q0/w0/p0.03856383) Cd(v43/q0/w0/p0.04445541)
bD(v59/q0/w0/p0.04863409) cD(v110/q0/w0/p0.04025667)

```
    a   b   c
  o   o   o---o A
A             ¦
  o   o   o   o B
B
  o   o   o   o C
C
  o---o   o   o D
  a   b   c   d
```

4-2 PC/17 | Next: 1 PC/17 | Pts: 0 | Ad(v1780/q-82.8274/w-184.66017/p0.050594784)
Winner: Cb(v693/q0/w0/p0.52010405) »
aA(v21/q0/w0/p0.00047816857) bA(v578/q0/w0/p0.43556684)
Aa(v41/q0/w0/p0.009145977) Ab(v0/q0/w0/p0.00034903048)
Ac(v21/q0/w0/p0.00028601722) aB(v21/q0/w0/p0.00019030644)
bB(v21/q0/w0/p0.0019782844) cB(v21/q0/w0/p0.0005608395)
Ba(v41/q0/w0/p0.0123757105) Bb(v21/q0/w0/p0.0011595082)
Bc(v0/q0/w0/p0.00014016504) Bd(v21/q0/w0/p0.0009295938)
aC(v38/q0/w0/p0.0009972106) bC(v39/q0/w0/p0.0013121681)
cC(v39/q0/w0/p0.00025122944) Ca(v40/q0/w0/p0.0045590126)
Cb(v693/q0/w0/p0.52010405) Cc(v21/q0/w0/p0.0018852957)
Cd(v21/q0/w0/p0.0021438962) bD(v21/q0/w0/p0.0055246977)
cD(v60/q0/w0/p6.2033054E-05)

```
    a   b   c
  o   o   o---o A
A             ¦
  o   o   o   o B
B
```

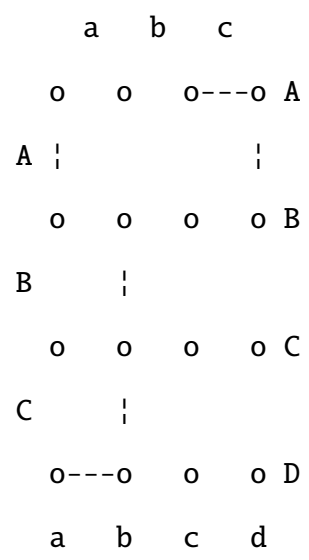
```
   o   o   o   o C
 C     ¦
   o---o   o   o D
   a   b   c   d
```

5-1 PC/17 | Next: 2 PC/17 | Pts: 0 | Cb(v261/q-5.079575/w-21.408968/p0.52010405)
Winner: Bb(v180/q0/w0/p0.818083) »
aA(v0/q0/w0/p5.029458E-06) bA(v20/q0/w0/p9.73782E-07)
Aa(v0/q0/w0/p1.3419015E-07) Ab(v0/q0/w0/p4.10394E-05)
Ac(v0/q0/w0/p2.6158674E-05) aB(v0/q0/w0/p1.3179354E-08)
bB(v0/q0/w0/p6.94787E-08) cB(v0/q0/w0/p2.0305494E-07)
Ba(v40/q0/w0/p0.18163635) Bb(v180/q0/w0/p0.818083)
Bc(v0/q0/w0/p3.5405967E-06) Bd(v0/q0/w0/p0.00017574911)
aC(v0/q0/w0/p1.32263445E-08) bC(v0/q0/w0/p4.6476904E-07)
cC(v0/q0/w0/p5.160206E-08) Ca(v0/q0/w0/p3.532192E-06)
Cc(v0/q0/w0/p1.5519784E-07) Cd(v0/q0/w0/p2.6613277E-08)
bD(v20/q0/w0/p2.3484827E-05) cD(v0/q0/w0/p6.414831E-08)

```
     a   b   c
   o   o   o---o A
 A             ¦
   o   o   o   o B
 B     ¦
   o   o   o   o C
 C     ¦
   o---o   o   o D
  a   b   c   d
```

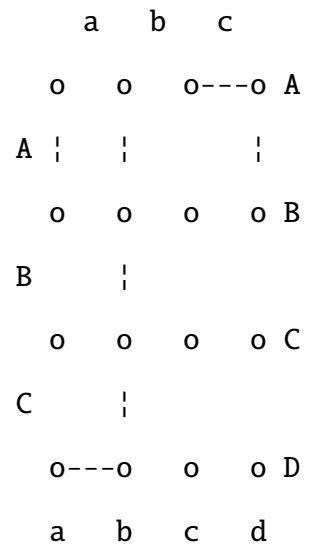
6-2 PC/17 | Next: 1 PC/17 | Pts: 0 | Bb(v450/q-2.3078866/w-20.565342/p0.818083)

Winner: Aa(v411/q0/w0/p0.99999976) »
aA(v19/q0/w0/p7.240633E-09) bA(v0/q0/w0/p9.105264E-11)
Aa(v411/q0/w0/p0.99999976) Ab(v0/q0/w0/p4.2234902E-08)
Ac(v0/q0/w0/p7.6211853E-10) aB(v0/q0/w0/p2.4437006E-13)
bB(v0/q0/w0/p3.801159E-11) cB(v0/q0/w0/p3.740199E-12)
Ba(v0/q0/w0/p4.7529376E-12) Bc(v0/q0/w0/p2.7795996E-10)
Bd(v0/q0/w0/p2.3645731E-11) aC(v0/q0/w0/p2.8120197E-11)
bC(v0/q0/w0/p5.1175088E-14) cC(v0/q0/w0/p9.506224E-12)
Ca(v0/q0/w0/p2.1980591E-07) Cc(v0/q0/w0/p9.46709E-15)
Cd(v0/q0/w0/p6.126411E-10) bD(v19/q0/w0/p1.1322993E-11)
cD(v0/q0/w0/p8.0464975E-12)

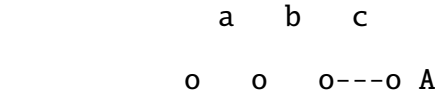


7-1 PC/17 | Next: 2 PC/17 | Pts: 0 | Aa(v260/q-3.9944196/w-20.565342/p0.99999976)
Winner: Ab(v241/q0/w0/p0.99999756) »
aA(v0/q0/w0/p7.365032E-08) bA(v0/q0/w0/p3.7739812E-07)
Ab(v241/q0/w0/p0.99999756) Ac(v0/q0/w0/p2.6850663E-10)
aB(v0/q0/w0/p3.9529393E-08) bB(v0/q0/w0/p8.7665717E-07)
cB(v0/q0/w0/p3.8890576E-08) Ba(v0/q0/w0/p9.951108E-10)
Bc(v18/q0/w0/p7.639772E-09) Bd(v0/q0/w0/p1.6229833E-07)

aC(v0/q0/w0/p7.301555E-09) bC(v0/q0/w0/p3.1380093E-11)
cC(v0/q0/w0/p4.8960754E-07) Ca(v0/q0/w0/p3.0500388E-10)
Cc(v0/q0/w0/p2.016523E-09) Cd(v0/q0/w0/p2.8982626E-07)
bD(v0/q0/w0/p2.6941478E-09) cD(v0/q0/w0/p1.1615708E-09)



8-2 PC/17 | Next: 1 PC/17 | Pts: 0 | Ab(v408/q-2.5454633/w-20.565342/p0.99999756)
Winner: Cd(v356/q0/w0/p0.999836) »
aA(v0/q0/w0/p1.2396176E-09) bA(v17/q0/w0/p5.642813E-08)
Ac(v0/q0/w0/p2.789358E-09) aB(v0/q0/w0/p7.3158024E-10)
bB(v17/q0/w0/p4.5015895E-08) cB(v17/q0/w0/p6.2172545E-10)
Ba(v0/q0/w0/p2.777997E-08) Bc(v0/q0/w0/p1.9575712E-09)
Bd(v0/q0/w0/p7.831977E-06) aC(v0/q0/w0/p3.341968E-08)
bC(v0/q0/w0/p4.0613104E-09) cC(v0/q0/w0/p0.00012336015)
Ca(v0/q0/w0/p5.3801625E-07) Cc(v0/q0/w0/p1.4502029E-05)
Cd(v356/q0/w0/p0.999836) bD(v0/q0/w0/p1.5899319E-05)
cD(v0/q0/w0/p1.6599654E-06)

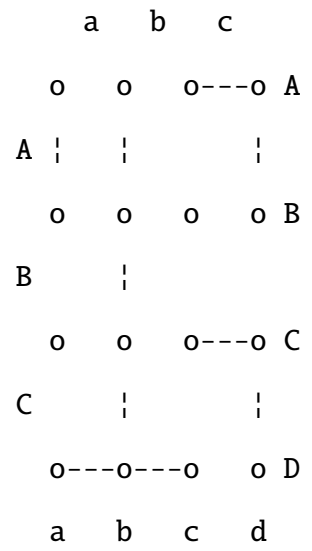


```
A ¦   ¦       ¦
  o   o   o   o B
B     ¦
  o   o   o   o C
C     ¦       ¦
  o---o   o   o D
  a   b   c   d
```

9-1 PC/17 | Next: 2 PC/17 | Pts: 0 | Cd(v340/q-3.0545557/w-20.565342/p0.999836)
Winner: cC(v323/q0/w0/p0.9999772) »
aA(v0/q0/w0/p1.6624644E-07) bA(v0/q0/w0/p5.8734565E-07)
Ac(v0/q0/w0/p8.2856066E-10) aB(v0/q0/w0/p7.343553E-10)
bB(v0/q0/w0/p3.1597033E-08) cB(v0/q0/w0/p2.621904E-09)
Ba(v16/q0/w0/p6.790776E-08) Bc(v0/q0/w0/p2.6440348E-06)
Bd(v0/q0/w0/p3.4889464E-07) aC(v0/q0/w0/p2.0485045E-07)
bC(v0/q0/w0/p5.5284072E-08) cC(v323/q0/w0/p0.9999772)
Ca(v0/q0/w0/p2.0730043E-10) Cc(v0/q0/w0/p1.7794679E-05)
bD(v0/q0/w0/p8.888527E-07) cD(v0/q0/w0/p2.2732263E-10)

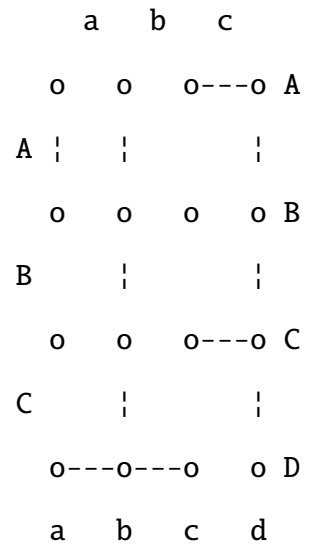
```
    a   b   c
  o   o   o---o A
A ¦   ¦       ¦
  o   o   o   o B
B     ¦
  o   o   o---o C
C     ¦       ¦
  o---o   o   o D
 a   b   c   d
```

10-2 PC/17 | Next: 1 PC/17 | Pts: 0 | cC(v392/q-2.6493592/w-20.565342/p0.9999772)
Winner: bD(v361/q0/w0/p0.9999086) »
aA(v0/q0/w0/p2.4602095E-10) bA(v0/q0/w0/p3.0610525E-05)
Ac(v0/q0/w0/p3.1596947E-10) aB(v0/q0/w0/p1.7262652E-10)
bB(v0/q0/w0/p1.5624471E-07) cB(v0/q0/w0/p2.1702247E-10)
Ba(v0/q0/w0/p4.579013E-06) Bc(v0/q0/w0/p1.8931365E-08)
Bd(v0/q0/w0/p3.270403E-05) aC(v0/q0/w0/p6.148302E-07)
bC(v0/q0/w0/p2.2702729E-05) Ca(v0/q0/w0/p5.0000355E-09)
Cc(v15/q0/w0/p9.358344E-09) bD(v361/q0/w0/p0.9999086)
cD(v15/q0/w0/p6.3429066E-09)

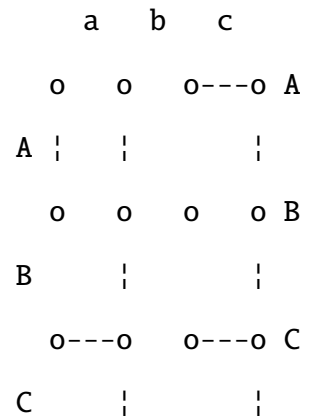


11-1 PC/17 | Next: 2 PC/17 | Pts: 0 | bD(v422/q-2.461016/w-20.565342/p0.9999086)
Winner: Bd(v421/q0/w0/p0.9997443) »
aA(v0/q0/w0/p8.52246E-09) bA(v0/q0/w0/p2.0802614E-05)
Ac(v0/q0/w0/p1.2336277E-07) aB(v0/q0/w0/p3.173262E-08)
bB(v0/q0/w0/p1.0294211E-09) cB(v0/q0/w0/p2.3156488E-06)
Ba(v0/q0/w0/p0.00016246713) Bc(v0/q0/w0/p1.1153524E-08)
Bd(v421/q0/w0/p0.9997443) aC(v0/q0/w0/p5.5277866E-07)
bC(v0/q0/w0/p3.587992E-09) Ca(v0/q0/w0/p3.100275E-07)

Cc(v0/q0/w0/p2.0252822E-08) cD(v0/q0/w0/p6.907612E-05)



12-2 PC/17 | Next: 1 PC/17 | Pts: 0 | Bd(v432/q-2.4040484/w-20.565342/p0.9997443)
Winner: aC(v431/q0/w0/p0.99999917) »
aA(v0/q0/w0/p4.0017394E-08) bA(v0/q0/w0/p5.4234714E-11)
Ac(v0/q0/w0/p1.9722721E-09) aB(v0/q0/w0/p1.7248205E-08)
bB(v0/q0/w0/p4.213653E-09) cB(v0/q0/w0/p2.282512E-11)
Ba(v0/q0/w0/p7.673732E-07) Bc(v0/q0/w0/p6.063707E-08)
aC(v431/q0/w0/p0.99999917) bC(v0/q0/w0/p3.9569847E-10)
Ca(v0/q0/w0/p9.493798E-10) Cc(v0/q0/w0/p2.409216E-09)
cD(v0/q0/w0/p4.1113657E-10)



```
   o---o---o   o D
   a   b   c   d
```

13-1 PC/17 | Next: 1 PC/17 | Pts: 1 | aC(v520/q-1.9972095/w-20.565342/p0.99999917)
Winner: Ca(v507/q0/w0/p0.99999493) »
aA(v0/q0/w0/p5.073615E-06) bA(v12/q0/w0/p7.505881E-11)
Ac(v0/q0/w0/p1.8471106E-09) aB(v0/q0/w0/p1.4011581E-13)
bB(v0/q0/w0/p6.900005E-13) cB(v0/q0/w0/p3.3461076E-13)
Ba(v0/q0/w0/p1.6084748E-09) Bc(v0/q0/w0/p4.0513767E-10)
bC(v0/q0/w0/p3.017271E-09) Ca(v507/q0/w0/p0.99999493)
Cc(v0/q0/w0/p6.977939E-11) cD(v0/q0/w0/p5.40872E-10)

```
     a   b   c
   o   o   o---o A
 A ¦   ¦       ¦
   o   o   o   o B
 B     ¦       ¦
   o---o   o---o C
 C ¦ 1 ¦       ¦
   o---o---o   o D
   a   b   c   d
```

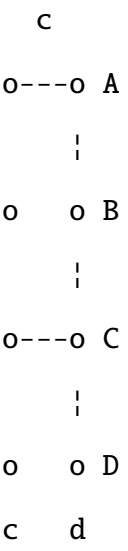
14-1 PC/17 | Next: 2 PC/17 | Pts: 1 | Ca(v607/q-1.7109538/w-20.565342/p0.99999493)
Winner: aA(v606/q0/w0/p0.9999816) »
aA(v606/q0/w0/p0.9999816) bA(v0/q0/w0/p1.7691503E-09)
Ac(v0/q0/w0/p1.1431795E-06) aB(v0/q0/w0/p4.1319708E-09)
bB(v0/q0/w0/p4.6624393E-10) cB(v0/q0/w0/p5.0991095E-10)
Ba(v0/q0/w0/p1.3218675E-08) Bc(v0/q0/w0/p2.571412E-07)
bC(v0/q0/w0/p5.7619634E-08) Cc(v0/q0/w0/p8.880994E-10)
cD(v0/q0/w0/p1.6989336E-05)

```
    a   b   c
  o---o   o---o A
A ¦   ¦       ¦
  o   o   o   o B
B     ¦       ¦
  o---o   o---o C
C ¦ 1 ¦       ¦
  o---o---o   o D
  a   b   c   d
```

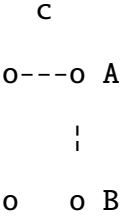
15-2 PC/17 | Next: 2 PC/17 | Pts: 1 | aA(v518/q-2.0049207/w-20.565342/p0.9999816)
Winner: aB(v517/q0/w0/p1) » bA(v0/q0/w0/p5.135806E-10)
Ac(v0/q0/w0/p1.2679883E-13) aB(v517/q0/w0/p1)
bB(v0/q0/w0/p2.30424E-08) cB(v0/q0/w0/p3.049008E-12)
Ba(v0/q0/w0/p1.3169442E-08) Bc(v0/q0/w0/p4.970721E-08)
bC(v0/q0/w0/p8.4103724E-10) Cc(v0/q0/w0/p3.399076E-14)
cD(v0/q0/w0/p7.0642823E-15)

```
    a   b   c
  o---o   o---o A
A ¦ 2 ¦       ¦
  o---o   o   o B
B     ¦       ¦
  o---o   o---o C
C ¦ 1 ¦       ¦
  o---o---o   o D
  a   b   c   d
```

16-2 PC/17 | Next: 2 PC/17 | Pts: 1 | aB(v617/q-1.6832237/w-20.565342/p1)
Winner: Ba(v616/q0/w0/p1) »
bA(v0/q0/w0/p3.1568398E-13) Ac(v0/q0/w0/p1.724065E-16)
bB(v0/q0/w0/p1.1245699E-16) cB(v0/q0/w0/p9.375877E-15)
Ba(v616/q0/w0/p1) Bc(v0/q0/w0/p1.2977947E-14)
bC(v0/q0/w0/p2.1557784E-12) Cc(v0/q0/w0/p4.4112913E-18)
cD(v0/q0/w0/p2.6848992E-17)



17-2 PC/17 | Next: 1 PC/17 | Pts: 1 | Ba(v716/q-1.4504875/w-20.565342/p1)
Winner: bC(v715/q0/w0/p0.9999896) »
bA(v0/q0/w0/p9.025953E-09) Ac(v0/q0/w0/p8.104956E-10)
bB(v0/q0/w0/p1.7033696E-09) cB(v0/q0/w0/p9.208377E-06)
Bc(v0/q0/w0/p1.1454061E-06) bC(v715/q0/w0/p0.9999896)
Cc(v0/q0/w0/p9.6154835E-09) cD(v0/q0/w0/p2.496784E-12)



```
B ¦ 2 ¦       ¦
  o---o---o---o C
C ¦ 1 ¦       ¦
  o---o---o   o D
  a   b   c   d
```

18-1 PC/17|Next: 1 PC/17|Pts: 1|bC(v703/q-1.4773095/w-20.565342/p0.9999896)
Winner: Cc(v702/q0/w0/p1) »
bA(v0/q0/w0/p5.1736443E-08) Ac(v0/q0/w0/p5.753286E-15)
bB(v0/q0/w0/p2.1771488E-13) cB(v0/q0/w0/p1.0163723E-14)
Bc(v0/q0/w0/p7.170426E-11) Cc(v702/q0/w0/p1)
cD(v0/q0/w0/p4.4959061E-13)

```
    a   b   c
  o---o   o---o A
A ¦ 2 ¦       ¦
  o---o   o   o B
B ¦ 2 ¦       ¦
  o---o---o---o C
C ¦ 1 ¦ 1 ¦   ¦
  o---o---o   o D
  a   b   c   d
```

19-1 PC/17|Next: 1 PC/17|Pts: 1|Cc(v802/q-1.2949489/w-20.565342/p1)
Winner: cD(v801/q0/w0/p0.99999523) »
bA(v0/q0/w0/p1.31454385E-08) Ac(v0/q0/w0/p4.4900297E-10)
bB(v0/q0/w0/p2.611951E-07) cB(v0/q0/w0/p1.2191957E-08)
Bc(v0/q0/w0/p4.4894737E-06) cD(v801/q0/w0/p0.99999523)

```
    a   b   c
  o---o   o---o A
A ¦ 2 ¦       ¦
  o---o   o   o B
B ¦ 2 ¦       ¦
  o---o---o---o C
C ¦ 1 ¦ 1 ¦ 1 ¦
  o---o---o---o D
  a   b   c   d
```

20-1 PC/17 | Next: 2 PC/17 | Pts: 1 | cD(v901/q-1.1526624/w-20.565342/p0.99999523)
Winner: Bc(v900/q0/w0/p0.999853) »
bA(v0/q0/w0/p3.1919097E-08) Ac(v0/q0/w0/p1.2025893E-07)
bB(v0/q0/w0/p0.00014633572) cB(v0/q0/w0/p4.9289844E-07)
Bc(v900/q0/w0/p0.999853)

```
    a   b   c
  o---o   o---o A
A ¦ 2 ¦       ¦
  o---o   o   o B
B ¦ 2 ¦   ¦   ¦
  o---o---o---o C
C ¦ 1 ¦ 1 ¦ 1 ¦
  o---o---o---o D
  a   b   c   d
```

21-2 PC/17 | Next: 2 PC/17 | Pts: 1 | Bc(v806/q-1.2885224/w-20.565342/p0.999853)
Winner: bB(v805/q0/w0/p0.99999994) »
bA(v0/q0/w0/p8.1876685E-14) Ac(v0/q0/w0/p4.166678E-17)

bB(v805/q0/w0/p0.99999994) cB(v0/q0/w0/p5.990542E-08)

```
     a   b   c
   o---o   o---o A
 A ¦ 2 ¦       ¦
   o---o---o   o B
 B ¦ 2 ¦ 2 ¦   ¦
   o---o---o---o C
 C ¦ 1 ¦ 1 ¦ 1 ¦
   o---o---o---o D
   a   b   c   d
```

22-2 PC/17 | Next: 2 PC/17 | Pts: 1 | bB(v905/q-1.147568/w-20.565342/p0.99999994)
Winner: cB(v904/q0/w0/p0.99999714) »
bA(v0/q0/w0/p3.3591622E-08) Ac(v0/q0/w0/p2.8563513E-06)
cB(v904/q0/w0/p0.99999714)

```
     a   b   c
   o---o   o---o A
 A ¦ 2 ¦       ¦
   o---o---o---o B
 B ¦ 2 ¦ 2 ¦ 2 ¦
   o---o---o---o C
 C ¦ 1 ¦ 1 ¦ 1 ¦
   o---o---o---o D
   a   b   c   d
```

23-2 PC/17 | Next: 2 PC/17 | Pts: 1 | cB(v1004/q-1.0344112/w-20.565342/p0.99999714)
Winner: Ac(v1003/q0/w0/p0.999997) »

bA(v0/q0/w0/p2.9815042E-06) Ac(v1003/q0/w0/p0.999997)

```
    a   b   c
  o---o   o---o A
A ¦ 2 ¦   ¦ 2 ¦
  o---o---o---o B
B ¦ 2 ¦ 2 ¦ 2 ¦
  o---o---o---o C
C ¦ 1 ¦ 1 ¦ 1 ¦
  o---o---o---o D
  a   b   c   d
```

24-2 PC/17 | Next: 2 PC/17 | Pts: 1 | Ac(v1103/q-0.94156736/w-20.565342/p0.999997)
Winner: bA(v1102/q0/w0/p1) » bA(v1102/q0/w0/p1)

```
    a   b   c
  o---o---o---o A
A ¦ 2 ¦ 2 ¦ 2 ¦
  o---o---o---o B
B ¦ 2 ¦ 2 ¦ 2 ¦
  o---o---o---o C
C ¦ 1 ¦ 1 ¦ 1 ¦
  o---o---o---o D
  a   b   c   d
```

A.2 Training mode visualization sample

INPUT-NET : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,6360114]

OUTPUT-POLICY-NET : [0,039366383 0,04268062 0,040236995 1,565611E-12 0,040688317 0,04277322 0,04351223 0,039834104 0,043835912 0,043203708 0,042056892 1,7453664E-12 0,04149754 0,042861585 0,044954855 0,043145053 0,040814146 0,04109886 0,044247154 1,7760877E-12 0,03862778 0,042480122 0,043054357 0,03961953 0,03911055 0,040412232 0,039887827 1,5102035E-12]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A
   o   o   o   o B
 B
   o   o   o   o C
 C
   o   o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,5370412]

OUTPUT-POLICY-NET : [0,043005027 0,041829202 0,04188238 2,1807044E-10 0,045227394 0,039883986 0,044426523 0,044966318 0,042671796 0,043566685 0,043483388 1,9633647E-10 0,04299262 0,044501323 0,0124434335 0,043732267 0,042574566 0,043026946 0,042477198 2,036505E-10 0,04327468 0,041617237 0,044327784 0,045350987 0,043824535 0,03878909

0,040124632 2,363185E-10]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A
   o   o   o   o B
 B         ¦
   o   o   o   o C
 C
   o   o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,8084888]

OUTPUT-POLICY-NET : [0,042266477 0,04355092 0,04064793 1,0114992E-11 0,04793293 0,04059138 0,04445904 0,046527136 0,050207943 0,04617587 0,042867977 9,721724E-12 0,046019044 0,041657887 0,015198369 0,04686839 0,04631198 0,041191097 0,047224633 9,18528E-12 0,043765113 0,044670116 0,048742328 0,0074030184 0,0421364 0,04212585 0,04145807 1,0089635E-11]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A
   o   o   o   o B
 B         ¦
   o   o   o   o C
 C             ¦
```

```
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,9957478]

OUTPUT-POLICY-NET : [1,0476485E-08 1,3749133E-07 0,00017224098 1,581842E-13 5,3201524E-07 5,551553E-07 2,4128866E-07 3,240599E-05 1,9800034E-07 3,3383035E-06 0,00018384053 1,3643564E-13 3,5835267E-06 1,0273343E-06 2,435277E-10 6,838459E-07 1,0876386E-07 1,7414951E-06 5,684226E-06 1,4593753E-13 5,7221573E-06 2,6823545E-07 0,9995857 9,014911E-09 6,1349866E-08 6,103544E-08 1,7416811E-06 1,5264046E-13]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A
   o---o   o   o B
 B         ¦
   o   o   o   o C
 C             ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,9985361]

OUTPUT-POLICY-NET : [8,0823006E-13 8,298956E-09 4,4426386E-08 1,19268685E-14 1,7150503E-10 5,4317557E-11 2,4914993E-07 0,99999964 3,0165213E-12 1,6484466E-12 1,4528259E-

10 9,75156E-15 2,4191885E-11 8,742927E-11 2,715629E-14 1,2924723E-08 1,3941778E-09 8,392766E-12 4,6760246E-10 1,0595521E-14 7,57391E-11 3,4444777E-11 8,539928E-14 3,1611907E-12 5,725883E-11 4,435348E-12 4,7908934E-11 8,526805E-15]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A
   o---o   o   o B
 B         ¦
   o   o   o   o C
 C         ¦   ¦
   o   o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,99997854]

OUTPUT-POLICY-NET : [1,053584E-12 2,7716263E-10 4,767173E-12 6,6667753E-15 6,747747E-10 1,4339029E-09 0,99999964 6,0451406E-08 3,3044429E-12 1,1667864E-10 2,2414972E-07 4,99894E-15 4,41985E-09 1,5596362E-10 3,109877E-13 5,054501E-11 2,3764369E-11 1,02447835E-10 2,1171824E-10 6,2243437E-15 1,40659645E-11 8,149709E-11 6,1869503E-12 1,0688343E-10 1,725962E-13 7,860587E-10 2,9246078E-10 7,655314E-15]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A             ¦
   o---o   o   o B
 B         ¦
```

```
   o   o   o   o C
 C         ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,9999824]

OUTPUT-POLICY-NET : [5,6116377E-14 1,1090775E-12 2,6733688E-11 9,020266E-16 2,735451E-11 1,5126646E-13 5,9155283E-14 1,4993193E-13 3,0122287E-15 2,6599182E-13 1,3907022E-12 5,9842253E-16 1 1,2069219E-13 1,2160882E-10 5,207634E-11 1,7756709E-12 1,2835615E-13 2,0549865E-13 6,6938123E-16 8,607675E-12 5,837618E-10 2,3837532E-13 4,3336376E-16 4,9010004E-15 5,518372E-10 8,2405575E-11 5,3331497E-16]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A         ¦   ¦
   o---o   o   o B
 B         ¦
   o   o   o   o C
 C         ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,99938464]

OUTPUT-POLICY-NET : [5,6329198E-11 1,8308711E-10 1,2433406E-11 1,1767294E-14 5,9652755E-11 6,611386E-08 2,1023967E-11 3,7552387E-12 3,5162544E-15 2,5926608E-10 2,303628E-12 6,925425E-15 1,0062352E-11 3,5778156E-09 8,9141076E-11 1,9352514E-07 2,0717445E-12 8,0191714E-10 2,7462085E-11 8,0796415E-15 1,5062383E-07 0,99999964 7,925076E-11 9,339762E-12 2,039836E-11 3,9917337E-11 2,948867E-14 8,428848E-15]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A         ¦   ¦
   o---o   o   o B
 B ¦       ¦
   o   o   o   o C
 C         ¦   ¦
   o   o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,99999005]

OUTPUT-POLICY-NET : [6,527558E-13 9,053948E-12 3,445535E-12 1,3503934E-15 1,6793983E-13 5,523215E-14 5,623311E-16 8,920004E-13 1,4101131E-15 9,828609E-15 1,972771E-12 7,0783325E-16 1,0185382E-10 9,484408E-11 6,1192466E-16 1 1,1006676E-11 4,0677056E-13 5,438981E-12 1,0905741E-15 1,320461E-11 1,5795937E-13 3,2154988E-18 6,013952E-16 1,7033108E-11 8,48993E-13 9,691368E-13 9,669252E-16]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A         ¦   ¦
```

```
   o---o   o   o B
 B ¦       ¦
   o   o   o   o C
 C     ¦   ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,99997705]

OUTPUT-POLICY-NET : [4,412688E-11 3,989465E-12 1,7649548E-11 1,1092425E-16 0,0002191119 2,8933138E-12 3,0511695E-14 1,9545715E-13 2,1132185E-13 5,83087E-12 8,419067E-15 8,3462147E-17 4,80522E-14 7,410497E-11 6,657691E-12 1,24403396E-11 9,494598E-15 6,5457023E-16 1,6832444E-12 8,235082E-17 0,99978083 4,074052E-12 2,6511952E-14 4,4096017E-14 1,7171617E-07 1,1302158E-14 1,8276752E-11 7,201862E-17]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A         ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o   o   o   o C
 C     ¦   ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,99999464]

OUTPUT-POLICY-NET : [2,4381472E-09 2,0180986E-08 1,7817386E-10 7,047621E-15 1 6,3102643E-09 1,5078844E-11 2,0427382E-14 1,1712495E-12 1,6761735E-08 4,7550014E-14 9,551321E-15 7,474873E-15 5,769533E-13 1,7366883E-13 5,8160845E-16 1,400015E-15 1,4465759E-12 9,250756E-14 9,586803E-15 6,022252E-12 9,12541E-16 5,1385786E-14 5,887029E-14 4,1603583E-09 2,1741693E-13 2,7620317E-11 6,748684E-15]

BOARD : [

```
     a   b   c
   o   o   o   o A
 A         ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o   o   o   o C
 C ¦   ¦   ¦   ¦
   o   o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [0,9989917]

OUTPUT-POLICY-NET : [9,232138E-08 0,99994314 2,2172846E-09 7,996063E-15 9,92237E-09 1,709295E-07 2,5073519E-09 4,4869983E-13 3,5358333E-12 6,9390735E-06 7,334317E-10 6,906408E-15 2,4167827E-11 7,2288106E-13 1,5081957E-11 2,3693148E-11 3,5124818E-12 6,6696976E-10 5,346105E-09 7,031243E-15 2,9126634E-10 8,213316E-12 4,6935573E-11 4,5739614E-07 1,1441275E-05 3,7832244E-05 2,8757585E-08 7,075045E-15]

BOARD : [

```
     a   b   c
```

```
   o   o   o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o   o   o   o C
 C ¦   ¦   ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,99998426]

OUTPUT-POLICY-NET : [6,904619E-10 8,504095E-13 1,2392576E-10 1,0649046E-15 7,0746853E-13 4,3410024E-15 7,148583E-17 4,764568E-15 1,6299833E-14 1,587539E-11 1,7022038E-14 1,3798039E-15 5,5927E-15 1,0190755E-08 6,780802E-14 1,9807093E-14 9,189724E-12 1,6358331E-13 3,144441E-14 1,5083358E-15 4,505308E-12 1,6290791E-10 1,0106228E-11 3,7566348E-13 1 6,038323E-11 1,6870859E-11 1,1828681E-15]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o   o   o   o C
 C ¦   ¦   ¦   ¦
   o   o   o   o D
   a   b   c   d
]
```

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [6,998049E-18 1,1769834E-18 8,432513E-16 3,5312347E-20 3,0327786E-20 1,360284E-15 6,3907982E-21 3,4557176E-19 1,1317525E-16 2,2855367E-19 1,410739E-20 2,0941675E-20 1,0753575E-18 9,977724E-13 1,06529825E-16 1,2479643E-19 1 1,36841E-20 7,088509E-19 2,2798668E-20 7,2781706E-28 8,5972573E-23 9,270303E-19 1,4616775E-16 1,0182547E-21 8,986265E-16 5,7659193E-15 1,883713E-20]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o   o   o   o C
 C ¦   ¦   ¦   ¦
   o---o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [1,3451935E-14 2,2835148E-19 8,898136E-20 3,8806406E-20 7,59372E-21 2,0964327E-17 1,1380765E-18 1,4320212E-19 9,591258E-24 2,4831802E-19 5,423014E-21 1,923535E-20 4,4177256E-25 1 4,993176E-19 2,2061514E-20 7,6217797E-22 4,311725E-15 2,4937576E-17 2,1292728E-20 1,3425525E-19 2,2004574E-16 7,1856564E-17 5,0496564E-22 3,9596035E-16 1,8829615E-17 2,400666E-18 1,8565898E-20]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦       ¦   ¦
   o---o   o   o C
 C ¦ 2 ¦   ¦   ¦
   o---o   o   o D
   a   b   c   d
```

]

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [8,635348E-12 6,7278508E-18 3,5820594E-07 6,017432E-18 1,9116294E-16 4,7172312E-14 2,7664115E-15 2,3652334E-13 1,8017598E-15 6,8937135E-12 8,8135575E-13 3,5971227E-18 2,4996658E-10 6,425822E-15 5,7830524E-12 1,08988695E-16 6,75975E-18 3,3825258E-17 4,5817347E-14 4,3265947E-18 6,3467655E-16 2,4222737E-14 2,643437E-15 1,0112852E-16 5,3534086E-16 2,952431E-13 0,99999964 4,377996E-18]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦ 2 ¦   ¦   ¦
   o---o   o   o C
 C ¦ 2 ¦   ¦   ¦
   o---o   o   o D
```

```
   a   b   c   d
]
```

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 0]

POINTS : [0]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [-0,9999997]

OUTPUT-POLICY-NET : [9,799462E-19 7,948355E-20 1,659042E-16 8,681776E-19 8,36042E-22 7,317852E-18 3,9784515E-17 7,4991094E-18 9,5126234E-17 9,491663E-19 8,97321E-13 1,0495737E-18 7,874098E-17 6,7635188E-18 2,5365397E-16 5,888628E-20 7,4482406E-20 1,0374294E-17 1 8,483381E-19 3,2635625E-15 1,4617619E-15 7,425318E-20 9,138664E-20 7,494501E-19 1,897413E-15 1,2995954E-18 7,51007E-19]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦ 2 ¦   ¦   ¦
   o---o   o   o C
 C ¦ 2 ¦   ¦   ¦
   o---o   o---o D
   a   b   c   d
]
```

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0]

POINTS : [1]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [-0,99999964]

OUTPUT-POLICY-NET : [8,839558E-16 6,616707E-17 1,586338E-14 2,0852345E-17 1,5413374E-17 4,0058064E-15 1,3503606E-14 3,960376E-15 5,5297712E-14 4,847074E-18 1 1,6455465E-17 1,1370883E-16 2,8127228E-15 1,5928992E-14 7,377321E-20 1,0881331E-18 9,7643315E-

15 1,8183718E-13 1,5610657E-17 1,1406117E-12 7,1781496E-17 6,364221E-13 5,9880185E-15 4,01668E-18 3,3871E-12 7,576438E-12 1,2125171E-17]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o   o B
 B ¦ 2 ¦   ¦   ¦
   o---o   o---o C
 C ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o D
   a   b   c   d
```

]

INPUT-NET : [0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0]

POINTS : [1]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [-0,9999932]

OUTPUT-POLICY-NET : [1,2287994E-11 5,067011E-19 1 7,445152E-18 9,814544E-18 3,2670336E-10 7,1378773E-16 8,555385E-22 1,5769898E-13 1,3189224E-12 1,506031E-14 4,500716E-18 2,4632506E-22 5,5598756E-16 3,6272547E-13 7,0084485E-16 2,7408266E-21 4,4224814E-17 2,0469347E-14 3,7907785E-18 8,968646E-12 8,487732E-16 2,0888899E-16 2,3082436E-15 1,1041957E-15 9,773844E-13 3,3364376E-13 3,551583E-18]

BOARD : [

```
     a   b   c
   o   o---o   o A
 A ¦       ¦   ¦
   o---o   o---o B
 B ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o C
```

```
 C ¦ 2 ¦    ¦ 1 ¦
   o---o    o---o D
   a    b    c    d
]
```

INPUT-NET : [0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0]

POINTS : [1]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [-0,9999978]

OUTPUT-POLICY-NET : [1,9228615E-11 1,3833832E-14 2,9064928E-09 3,3207133E-16 1,0918609E-14 1 4,670647E-08 3,149195E-10 3,1629577E-09 8,7609835E-13 1,3082505E-10 3,0015597E-16 1,5495386E-16 2,803425E-15 4,5584066E-12 2,5011124E-17 3,5349078E-16 1,4145686E-15 5,175672E-10 2,2847033E-16 1,9635177E-12 4,3183486E-12 2,643587E-09 1,3826678E-11 7,723662E-15 8,976724E-12 6,5187467E-10 2,626239E-16]

BOARD : [

```
      a    b    c
   o    o---o---o A
 A ¦         ¦ 1 ¦
   o---o    o---o B
 B ¦ 2 ¦    ¦ 1 ¦
   o---o    o---o C
 C ¦ 2 ¦    ¦ 1 ¦
   o---o    o---o D
   a    b    c    d
]
```

INPUT-NET : [0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0]

POINTS : [0]

VALUE-NET MULTI : [1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [2,2899207E-09 8,981994E-28 1,5131671E-20 3,1812675E-23 4,054043E-

24 7,805546E-26 8,9655336E-27 7,452049E-20 1,0685463E-23 1 3,0849463E-21 1,9563665E-23 2,51261E-22 1,03280607E-16 3,463595E-20 4,4979634E-21 2,8760478E-27 8,062446E-12 5,5050763E-20 2,2152475E-23 1,173699E-22 4,1605802E-24 3,1369175E-25 1,2907784E-22 2,1231783E-23 9,759183E-17 7,2262936E-22 2,7382217E-23]

BOARD : [

```
     a   b   c
   o   o---o---o A
 A ¦   ¦   ¦ 1 ¦
   o---o   o---o B
 B ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o C
 C ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o D
   a   b   c   d
```

]

INPUT-NET : [0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [9,248268E-14 6,341494E-23 1,180289E-24 4,525975E-25 5,0272468E-29 2,148185E-27 1,2108389E-25 1,173671E-24 5,6348845E-31 2,9012622E-18 1,2681408E-24 1,890886E-25 1,2904577E-23 5,907153E-17 2,5587705E-20 1,6483976E-26 1,6883347E-30 1 4,6178882E-23 2,520584E-25 1,6923282E-20 6,75429E-25 2,975128E-21 2,638663E-22 3,0143022E-27 2,3027798E-13 1,2951243E-21 3,6693872E-25]

BOARD : [

```
     a   b   c
   o   o---o---o A
 A ¦   ¦ 2 ¦ 1 ¦
   o---o---o---o B
```

```
 B ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o C
 C ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o D
   a   b   c   d
]
```

INPUT-NET : [0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [1]

OUTPUT-POLICY-NET : [1,0075943E-13 2,0346759E-19 2,6086267E-23 2,5560592E-27 4,3410754E-27 2,8346484E-26 1,0037681E-23 4,5422185E-24 9,086841E-26 1,4359846E-15 6,630988E-23 1,5956405E-27 3,2732316E-26 7,466577E-23 4,7736556E-26 2,3846755E-25 9,849998E-31 1,5512676E-21 6,46232E-25 3,1210155E-27 3,6715556E-26 9,565004E-28 5,5438964E-27 1,9165626E-27 2,9487018E-20 1 1,8655795E-18 3,538571E-27]

BOARD : [

```
     a   b   c
   o   o---o---o A
 A ¦   ¦ 2 ¦ 1 ¦
   o---o---o---o B
 B ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o C
 C ¦ 2 ¦   ¦ 1 ¦
   o---o   o---o D
   a   b   c   d
]
```

INPUT-NET : [0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [0,6784105]

OUTPUT-POLICY-NET : [1 3,0447478E-16 1,499222E-13 2,2792853E-19 3,1460427E-14 2,9742827E-17 1,8234609E-12 1,18519205E-11 4,0105617E-15 5,9342424E-14 3,3321622E-14 1,3427831E-19 3,9181223E-14 6,3060885E-13 1,3320749E-18 5,9581184E-17 1,1832291E-15 1,7729599E-16 1,8529163E-14 1,7121602E-19 6,4707616E-16 5,138554E-16 2,6463913E-17 1,0880731E-14 9,099141E-14 9,714E-13 5,0948346E-15 1,2247138E-19]

BOARD : [

```
     a   b   c
   o   o---o---o A
 A ¦   ¦ 2 ¦ 1 ¦
   o---o---o---o B
 B ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o C
 C ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o D
   a   b   c   d
```

]

INPUT-NET : [1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0]

POINTS : [1]

VALUE-NET MULTI : [-1]

OUTPUT-VALUE-NET : [0,48259243]

OUTPUT-POLICY-NET : [0,10809224 0,040538922 0,08000807 2,5188915E-09 0,05965983 0,042984888 0,0039687986 0,061079517 0,0027836615 2,147384E-05 0,022070961 1,9131166E-09 0,24017887 0,001494113 9,801371E-06 0,041257426 0,005247985 0,009847539 0,0013811173 1,8350108E-09 0,0039976 0,001567072 0,020048786 0,0014219857 0,05115176 0,09612585 0,105061755 1,8307107E-09]

BOARD : [

```
     a   b   c
   o---o---o---o A
```

```
 A ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o B
 B ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o C
 C ¦ 2 ¦ 2 ¦ 1 ¦
   o---o---o---o D
   a   b   c   d
]
```

A.3 History of training

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
0	1.2962737083435059	0.7126913666725159	0.5835815668106079	0.9627030491828918	0.493597149848938	0.4691062867641449
1	1.2720657587051392	0.6964056491851807	0.5756598114967346	0.9549358487129211	0.48871690034866333	0.46621939539909363
2	1.2538642883300781	0.6856260299682617	0.5682377219200134	0.9458531141281128	0.4857501983642578	0.46010321378707886
3	1.2406600713729858	0.6729955077171326	0.5676650404930115	0.9418566823005676	0.48502492904663086	0.45683160424232483
4	1.2351489067077637	0.6729058623313904	0.5622426867485046	0.9387000203132629	0.48236629366874695	0.4563336968421936
5	1.2159011363983154	0.6647782921791077	0.5511242151260376	0.9366114735603333	0.48317083716392517	0.45344093441963196
6	1.2148867845535278	0.6622486710548401	0.5526383519172668	0.9313719868659973	0.4807576537132263	0.4506140351295471
7	1.2098978757858276	0.6594310402870178	0.550466001033783	0.9329649209976196	0.4814899265766144	0.4514748752117157
8	1.2030268907546997	0.6541215181350708	0.5489054322242737	0.9282054305076599	0.4808324873447418	0.44737300276756287
9	1.1955543756484985	0.6479690074920654	0.5475850701332092	0.9258015751838684	0.48057782649993896	0.4452240765094757
10	1.1964987516403198	0.6505661606788635	0.5459321141242981	0.9240594506263733	0.47972494363784479	0.44433480501174927
11	1.18351149559021	0.6440975666046143	0.5394133925437927	0.9218652844429016	0.4793599545955658	0.4425053298473358
12	1.1795109510421753	0.6398640275001526	0.539647102355957	0.9200870394706726	0.4787817895412445	0.4413054287433624
13	1.166465401649475	0.6351719498634338	0.531293511390686	0.9203698039054871	0.4782571494579315	0.44211283326148987
14	1.1685081720352173	0.6359207034111023	0.5325880646705627	0.9180367588996887	0.4776262044906616	0.4404104948043823
15	1.1659979820251465	0.636169970035553	0.5298274755477905	0.9182522296905518	0.47940877079963684	0.43884286284446716
16	1.166198492050171	0.6332157850265503	0.5329834818840027	0.9177935719490051	0.47998616099357605	0.43780747056007385
17	1.1529303789138794	0.6282815933227539	0.5246487259864807	0.9174250960350037	0.47856253385543823	0.43886250257492065
18	1.1555564403533936	0.6301212310791016	0.5254356265068054	0.9175968170166016	0.47964397072792053	0.4379526972770691
19	1.1514440774917603	0.6269518136978149	0.5244923233985901	0.9160472750663757	0.4788055121898651	0.43724191188812256
20	1.1497238874435425	0.626339852809906	0.5233846306800842	0.9160000681877136	0.47954514622688293	0.43645524978637695
21	1.1476954221725464	0.6253617405891418	0.5223345160484314	0.9150757193565369	0.4786932170391083	0.43638256192207336
22	1.1458287239074707	0.6241925954818726	0.5216355323791504	0.9146838784217834	0.4794136881828308	0.4352695345878601
23	1.1361346244812012	0.619689404964447	0.5164459347724915	0.9128802418708801	0.47855594754219055	0.43432414531707764
24	1.140162706375122	0.6228122115135193	0.5173491835594177	0.9137775301933289	0.4789814054965973	0.4347960352897644
25	1.1303694248199463	0.6176456809043884	0.5127236247062683	0.9125849008560181	0.47851327061653137	0.4340716600418091
26	1.1297801733016968	0.6146266460418701	0.5151537656784058	0.9130997061729431	0.47966593503952026	0.43343353271484375
27	1.1371376514434814	0.622349202632904	0.5147880911827087	0.9100960493087769	0.47785624861717224	0.4322396516799927
28	1.1288318634033203	0.6156390905380249	0.5131924748420715	0.9097995162010193	0.47837188839912415	0.4314279854297638
29	1.1238816976547241	0.6121435761451721	0.5117378830909729	0.9088765382766724	0.47727715969085693	0.4315999448299408
30	1.1199767589569092	0.6103042364120483	0.5096724629402161	0.9100058674812317	0.47868576645851135	0.4313201308250427
31	1.124277949333191	0.6135554909706116	0.5107229351997375	0.9086301326751709	0.477970689535141	0.43065929412841797
32	1.1253876686096191	0.614254891872406	0.5111324787139893	0.9071482419967651	0.4761597216129303	0.4309886395931244
33	1.1235826015472412	0.6142008900642395	0.5093820691108704	0.9083123803138733	0.47742143273353577	0.43089109659194946
34	1.12030827999115	0.6117584109306335	0.5085496306419373	0.9074925184249878	0.4779779314994812	0.4295142889022827

Table A.1 continued from previous page

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
35	1.1164332628250122	0.6114710569381714	0.5049618482589722	0.90794438123703	0.47901612520217896	0.4289282262325287
36	1.1208815574645996	0.6096417903900146	0.511239767074585	0.9096197485923767	0.48033225536346436	0.4292873740196228
37	1.113873839378357	0.6072109937667847	0.5066633820533752	0.9080883264541626	0.47931572794914246	0.42877230048179626
38	1.1084223985671997	0.606953501701355	0.5014693140983582	0.9075406789779663	0.4779379963874817	0.42960259318351746
39	1.10933518409729	0.6071118116378784	0.5022229552268982	0.9079885482788086	0.480150043964386	0.4278383255004883
40	1.1043263673782349	0.6029844284057617	0.5013412833213806	0.9080638289451599	0.47946640849113464	0.42859750986099243
41	1.1076878309249878	0.6086129546165466	0.499074250459671	0.9088468551635742	0.4792303442955017	0.42961645126342773
42	1.1071513891220093	0.6056560277938843	0.5014961361885071	0.9088951945304871	0.4803631603717804	0.42853212356567383
43	1.1059744358062744	0.6047134399414062	0.5012620687484741	0.906801164150238	0.4794544279575348	0.4273470342159271
44	1.1010149717330933	0.6037341952323914	0.49728038907051086	0.9069405794143677	0.4797353744506836	0.4272053837776184
45	1.097078561782837	0.6012440919876099	0.4958341717720032	0.906846821308136	0.47913315892219543	0.42771321535110474
46	1.096753478050232	0.5995243191719055	0.4972282350063324	0.9077710509300232	0.48015955090522766	0.4276118278503418
47	1.0924795866012573	0.5974329113960266	0.4950462281703949	0.9068877696990967	0.4803031086921692	0.42658451199531555
48	1.0957859754562378	0.5974012017250061	0.49838492274284363	0.9099524021148682	0.48085716366767883	0.42909544706344604
49	1.095436930656433	0.6035935282707214	0.4918438792228699	0.9090133309364319	0.47984758019447327	0.4291655719280243
50	1.0884894132614136	0.5937326550483704	0.49475622177124023	0.9078619480133057	0.48144620656967163	0.42641589045524597
51	1.0921306610107422	0.5985012054443359	0.4936293959617615	0.9093038439750671	0.4822739362716675	0.42702993750572205
52	1.094101071357727	0.5995481610298157	0.4945524334907532	0.9101099371910095	0.48193007707595825	0.42817994952201843
53	1.0903629064559937	0.5975017547607422	0.4928607940673828	0.9075958728790283	0.48089027404785156	0.42670562863349915
54	1.0892523527145386	0.5990006327629089	0.4902515113353729	0.9084603786468506	0.48149073123931885	0.4269695580005646
55	1.085436224937439	0.5978575944900513	0.4875779449939728	0.9074952006340027	0.4805116653442383	0.42698317766189575
56	1.0826964378356934	0.5955331921577454	0.4871631860733032	0.9065156579017639	0.48167064785957336	0.4248445928096771
57	1.0900357961654663	0.5989152193069458	0.49112004041671753	0.9072303175926208	0.48204052448272705	0.425189346075058
58	1.0907628536224365	0.601088285446167	0.48967498540878296	0.9085162878036499	0.4828592538833618	0.42565682530403137
59	1.0880484580993652	0.5967121720314026	0.49133557081222534	0.906074583530426	0.48191410303115845	0.42416003346443176
60	1.0830940008163452	0.5941823124885559	0.48891201615333557	0.9052157998085022	0.4816218316555023	0.42359378933906555
61	1.0833497047424316	0.5955154299736023	0.4878346025943756	0.908991813659668	0.4841054379940033	0.4248865246772766
62	1.0768369436264038	0.5906621217727661	0.48617616295814514	0.9092456698417664	0.4834618866443634	0.42578402161598206
63	1.080619215965271	0.593582034111023	0.48703768849372864	0.9088274836540222	0.48364952206611633	0.425178200006485
64	1.0730377435684204	0.5899666547775269	0.483070969581604	0.9108629822731018	0.4831332266330719	0.4277297854423523
65	1.0712168216705322	0.5893895030021667	0.4818272292613983	0.9085461497306824	0.4827289879322052	0.42581671476364136
66	1.071825385093689	0.5904756784439087	0.4813513159751892	0.9089197516441345	0.48336559534072876	0.42555445432662964
67	1.0778446197509766	0.5937974452972412	0.48404693603515625	0.9116252064704895	0.48360732197761536	0.42801815271377563
68	1.073495864868164	0.5922418236732483	0.481253057718277	0.9103575944900513	0.4839676320552826	0.426390141248703
69	1.0685983896255493	0.5879715085029602	0.4806269407272339	0.9089112877845764	0.48291778564453125	0.4259937107563019
70	1.0679421424865723	0.5871715545654297	0.48076969385147095	0.9117333889007568	0.4852493107318878	0.4264843761920929
71	1.0681648254394531	0.5883936285972595	0.4797707498073578	0.9105769395828247	0.4852448105812073	0.42533236742019653
72	1.0673224925994873	0.5868881344795227	0.4804341197013855	0.9106056094169617	0.4852191209793091	0.4253864288330078
73	1.0709108114242554	0.588248074054718	0.482662558555603	0.9105302095413208	0.48491746187210083	0.42561256885528564
74	1.06876540184021	0.5893753170967102	0.4793897569179535	0.911074697971344	0.48462504148483276	0.42644959688186646
75	1.0646203756332397	0.5874272584915161	0.4771926701068878	0.9097803831100464	0.4846091568470001	0.4251709282398224
76	1.0650080442428589	0.5846458673477173	0.48036250472068787	0.9104455709457397	0.4847621023654938	0.4256831407546997
77	1.0706976652145386	0.5940353870391846	0.4766612648963928	0.9113103151321411	0.48427537083625793	0.4270351827144623
78	1.0706918239593506	0.5911555886268616	0.4795357584953308	0.9116398096084595	0.4854260981082916	0.4262135922908783
79	1.068267822265625	0.5896217823028564	0.47864583134651184	0.9129855632781982	0.48662349581718445	0.4263613820075989
80	1.0596383810043335	0.584480881690979	0.4751567542552948	0.9126961827278137	0.48651358485221863	0.4261822998523712
81	1.0611766576766968	0.5855056643486023	0.4756709337234497	0.9141902923583984	0.4880580008029938	0.4261325001716614
82	1.064165711402893	0.5860679745674133	0.47809723019599915	0.9117763638496399	0.48669517040252686	0.4250815510749817
83	1.0579874515533447	0.5842195749282837	0.4737682044506073	0.9121800065040588	0.4860581159591675	0.4261220097541809
84	1.0693812370300293	0.5886915922164917	0.4806903302669525	0.911811888217926	0.4866374731063843	0.42517387866973877
85	1.0616036653518677	0.5857316255569458	0.47587281465530396	0.9114907383918762	0.486906498670578	0.42458391189575195
86	1.059807538986206	0.581697404384613	0.47810932993888855	0.9115593433380127	0.4868186116218567	0.42474040389060974
87	1.053341031074524	0.5818563103675842	0.4714851975440979	0.9156926274299622	0.48692068457603455	0.4287720322608948
88	1.0494009256362915	0.5779415965080261	0.47145891189575195	0.9147000908851624	0.4871930778026581	0.4275069534778595
89	1.0547620058059692	0.5821491479873657	0.47261369228363037	0.9133653044700623	0.48639553785324097	0.4269697964191437
90	1.0551321506500244	0.5817152261734009	0.4734167158603668	0.9126080274581909	0.4868760406970978	0.42573240399360657
91	1.0507588386535645	0.5806647539138794	0.47009342908859253	0.9118577837944031	0.4858156740665436	0.4260425567626953
92	1.0506001710891724	0.5802425146102905	0.47035714983940125	0.9156368374824524	0.4875187873840332	0.4281177818775177

Table A.1 continued from previous page

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
93	1.0550062656402588	0.5816002488136292	0.4734061360359192	0.9136117696762085	0.48628953099250793	0.42732223868370056
94	1.0506869554519653	0.5822476744651794	0.4684392511844635	0.9131292700767517	0.4868021011352539	0.42632731795310974
95	1.0534636974334717	0.5823945999145508	0.47106894850730896	0.9118685126304626	0.48552435636520386	0.4263438880443573
96	1.051488995552063	0.5787398815155029	0.4727492332458496	0.9140998125076294	0.48723095655441284	0.42686888575553894
97	1.0541964769363403	0.5837090611457825	0.47048673033714294	0.9136505722999573	0.48726728558540344	0.4263833165168762
98	1.0489587783813477	0.5793518424034119	0.4696071445941925	0.915493369102478	0.48775455355644226	0.4277389645576477
99	1.0485482215881348	0.5801522731781006	0.4683953821659088	0.9148460030555725	0.4879043400287628	0.42694130539894104
100	1.0484449863433838	0.5809397101402283	0.4675055742263794	0.914793074131012	0.486907422542572	0.4278859794139862
101	1.0415087938308716	0.5736963748931885	0.4678126573562622	0.9141136407852173	0.48766419291496277	0.42644935846328735
102	1.0446159839630127	0.5766307711601257	0.4679860770702362	0.914655864238739	0.48699063062667847	0.4276654124259949
103	1.0438315868377686	0.5758964419364929	0.4679355323314667	0.9147354960441589	0.4875369966030121	0.4271984398365021
104	1.0468535423278809	0.5771142244338989	0.4697398841381073	0.9135618805885315	0.48709842562675476	0.4264633357524872
105	1.0472129583358765	0.5790421962738037	0.4681713879108429	0.9157275557518005	0.48751771450042725	0.42820975184440613
106	1.043917179107666	0.5765781998634338	0.46733957529067993	0.9156902432441711	0.48851484060287476	0.4271756410598755
107	1.0467501878738403	0.5786494016647339	0.46810171008110046	0.9152087569236755	0.48723962903022766	0.42796939611434937
108	1.0414888858795166	0.5737324953079224	0.4677572250366211	0.9147631525993347	0.48800477385520935	0.4267584979534149
109	1.0424185991287231	0.5763567686080933	0.4660622179508209	0.9157590866088867	0.488765150308609	0.426994264125824
110	1.0396802425384521	0.5788730382919312	0.4608072340488434	0.9149221777915955	0.48819711804389954	0.4267253577709198
111	1.0416349172592163	0.5768617391586304	0.46477317810058594	0.9167123436927795	0.4896557927131653	0.42705610394477844
112	1.0452015399932861	0.579807460308075	0.4653940796852112	0.9152839779853821	0.48890209197998047	0.4263821542263031
113	1.0454344749450684	0.5761081576347351	0.46932709217071533	0.9157934188842773	0.4891023635864258	0.42669111490249634
114	1.0453330278396606	0.5802265405654907	0.46510741114616394	0.9146770238876343	0.48844000697135925	0.426236629486084
115	1.0397019386291504	0.5738046765327454	0.4658971130847931	0.9152485132217407	0.48779013752937317	0.42745813727378845
116	1.036521553993225	0.5733945965766907	0.46312713623046875	0.9161445498466492	0.48887115716934204	0.4272737205028534
117	1.0385615825653076	0.5746519565582275	0.4639095664024353	0.9145588278770447	0.4876113533973694	0.42694753408432007
118	1.0379680395126343	0.5737359523773193	0.46423178911209106	0.9185482263565063	0.48836690187454224	0.4301815330982208
119	1.037366271018982	0.5756452679634094	0.46172043681144714	0.9180372357368469	0.48901861906051636	0.4290185272693634
120	1.0375959873199463	0.5756751894950867	0.4619206190109253	0.9166902899742126	0.48936301469802856	0.42732757329940796
121	1.037338376045227	0.5735152363777161	0.46382278203964233	0.9194186329841614	0.4904361069202423	0.428982138633728
122	1.035306692123413	0.5735089778900146	0.4617975652217865	0.9179779887199402	0.4890742599964142	0.4289037883281708
123	1.0371850728988647	0.5746508836746216	0.4625345766544342	0.9184318780899048	0.4893072247505188	0.42912501096725464
124	1.038512110710144	0.5782087445259094	0.46030357480049133	0.9188887476921082	0.4897102117538452	0.4291781485080719
125	1.0393937826156616	0.5764773488044739	0.4629170000553131	0.9184818267822266	0.4900767207145691	0.4284049868583679
126	1.0321053266525269	0.5736926198005676	0.4584123492240906	0.9234462976455688	0.4912366271018982	0.4322097599506378
127	1.0365201234817505	0.5752193927764893	0.4612996280193329	0.9181003570556641	0.4896663427352905	0.42843419313430786
128	1.0349746942520142	0.5748924016952515	0.4600813090801239	0.9176708459854126	0.4886893630027771	0.42898133397102356
129	1.039414644241333	0.5782459378242493	0.46116819977760315	0.9183905124664307	0.4896702170372009	0.42872050404548645
130	1.0358821153640747	0.5743698477745056	0.46151164174079895	0.9188072681427002	0.4900958240032196	0.4287116825580597
131	1.0328925848007202	0.5743647813796997	0.45852795243263245	0.9196730256080627	0.490904301404953	0.4287688136100769
132	1.03117036819458	0.572933554649353	0.4582371413707733	0.9208678603172302	0.4911212921142578	0.42974653840065
133	1.0241672992706299	0.5646216869354248	0.4595455825328827	0.9186878800392151	0.4897940754890442	0.42889365553855896
134	1.0339568853378296	0.5753937363624573	0.4585636258125305	0.91709303855896	0.48971083760261536	0.42738184332847595
135	1.0341204404830933	0.5771579146385193	0.45696231722831726	0.9225329160690308	0.49103009700775146	0.4315027594566345
136	1.0389569997787476	0.5764995813369751	0.4624571204185486	0.9217149615287781	0.4922162592411041	0.4294987916946411
137	1.0301624536514282	0.5722798705101013	0.45788225531578064	0.9213699698448181	0.4916285276412964	0.42974135279655457
138	1.0297857522964478	0.5729182958602905	0.4568670392036438	0.918720543384552	0.49095186591148376	0.42776840925216675
139	1.0304644107818604	0.5707088112831116	0.45975562930107117	0.9186868071556091	0.4908902943134308	0.4277960956096649
140	1.0296945571899414	0.5708745121955872	0.458819717168808	0.9205701947212219	0.4922104775905609	0.42835918068885803
141	1.028691053390503	0.5699555277824402	0.4587370455265045	0.9189446568489075	0.4906699061393738	0.42827481031417847
142	1.0288430452346802	0.5697249174118042	0.45911866426467896	0.9200631976127625	0.4908031225204468	0.4292598366737366
143	1.0295672416687012	0.5725052356719971	0.45706090331077576	0.9199578166007996	0.49033716320991516	0.42962121963500977
144	1.0281566381454468	0.5695267915725708	0.458630234003067	0.9233339428901672	0.49204710125923157	0.4312871992588043
145	1.0202418565750122	0.5705768465995789	0.44966527819633484	0.9228817820549011	0.4919230043888092	0.43095865845680237
146	1.0213282108306885	0.5671077370643616	0.45422160625457764	0.925342857837677	0.49183425307273865	0.43350866436958313
147	1.0279440879821777	0.5710896849632263	0.45685529708862305	0.921523928642273	0.491362065076828	0.4301616847515106
148	1.02877938747406	0.5712938904762268	0.45748385787010193	0.9239640831947327	0.49388375878334045	0.43007993698120117
149	1.018946647644043	0.5656929612159729	0.4532545208930969	0.9235441088676453	0.492768794298172	0.4307754933834076
150	1.029026746749878	0.5730605721473694	0.45596614480018616	0.9225621819496155	0.4931652843952179	0.42939695715904236

Table A.1 continued from previous page

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
151	1.0279549360275269	0.5697857737541199	0.45816951990127563	0.9230880737304688	0.49301299455278168	0.4300754964351654
152	1.0190073251724243	0.5678589940071106	0.4511486887931824	0.9244948029518127	0.49318999905204773	0.43130505084991455
153	1.0208439826965332	0.5674123764038086	0.4534316062927246	0.9236162304878235	0.49330231547355565	0.43031418323516846
154	1.0267013311386108	0.57041996717453	0.45628082752227783	0.9203795194625854	0.4914169907569885	0.428962379693985
155	1.0329920053482056	0.5732476115226746	0.4597453773021698	0.9213874936103821	0.49045360088343839	0.43093442916870117
156	1.0239657163619995	0.569423496723175	0.45454198122024536	0.9252133965492249	0.494259238243103	0.43095412850379944
157	1.0183627605438232	0.5692555904388428	0.4491077661514282	0.9256883859634399	0.49396401643753305	0.4317243695259094
158	1.0252586603164673	0.5686951279640198	0.4565640389919281	0.9249007105827332	0.49372190237045291	0.4311787188053131
159	1.0216888189315796	0.568889856338501	0.4527983069419861	0.9280418157577515	0.49447938799858093	0.4335622787475586
160	1.0248825550079346	0.5688242316246033	0.4560582935810089	0.9264236688613892	0.4950620234012604	0.43136173486709595
161	1.0221575498580933	0.5701010823249817	0.4520561397075653	0.9275827407836914	0.49370157718658445	0.43388137221336365
162	1.0206382274627686	0.5676759481430054	0.4529634118080139	0.9286792278289795	0.4941726326942444	0.4345066249370575
163	1.0130064487457275	0.5630710124969482	0.44993463158607483	0.9254297614097595	0.49322378635406494	0.4322056174278259
164	1.0194872617721558	0.5669772624969482	0.45251068472862244	0.9265595078468323	0.49401527643203735	0.4325442314147949
165	1.0177569389343262	0.5686390399932861	0.44911810755729675	0.9311113357543945	0.49449530243873596	0.4366162419319153
166	1.016827940940857	0.5683452486991882	0.44848334789276123	0.9282078146934509	0.49451935291290283	0.4336884617805481
167	1.0197747945785522	0.567004382610321	0.4527697265148163	0.9269810318946838	0.4931081235408783	0.4338729679584503
168	1.0193345546722412	0.5669732689857483	0.45236173272132874	0.9253555536270142	0.49359825253486633	0.43175753951072693
169	1.0195950269699097	0.5676289796829224	0.45196661353111267	0.9282181859016418	0.494236558675766	0.43398138880729675
170	1.0135962963104248	0.5660510659217834	0.4475460648536682	0.930051863193512	0.4949844181537628	0.4350672960281372
171	1.017537236213684	0.5675684809684753	0.44996875524520874	0.9253159761428833	0.4941534399986267	0.43116289377212524
172	1.0178053379058838	0.568099319934845	0.44970574975013733	0.9296265244483948	0.4958883821964264	0.4337385594844818
173	1.0152945518493652	0.5653136372566223	0.44998130202293396	0.931841254234314	0.49647727608680725	0.4353639781475067
174	1.016649842262268	0.5647755265235901	0.45187506079673767	0.928092896938324	0.49405020475387573	0.4340427815914154
175	1.0121790170669556	0.5655498504638672	0.4466291666030884	0.9291257262229919	0.4964318871498108	0.4326940178871155
176	1.0140070915222168	0.5647351741790771	0.44927263259887695	0.9292359948158264	0.4959457516670227	0.43329012393951416
177	1.0132330656051636	0.5624778866767883	0.45075491070747375	0.9316127896308899	0.49671265482902527	0.4349004626274109
178	1.015887975692749	0.5650520324707031	0.4508366584777832	0.9276146292686462	0.49556222558021545	0.4320524036884308
179	1.0149245262145996	0.5651141405105591	0.44981008768081665	0.9271402359008789	0.4948078691959381	0.43233194947242737
180	1.0152376890182495	0.5647070407867432	0.4505302608013153	0.9288027882575989	0.4958980977535248	0.43290454149246216
181	1.0115689039230347	0.5658024549484253	0.4457660913467407	0.9291466474533081	0.49635183811187744	0.4327942728996277
182	1.018610954284668	0.5666569471359253	0.45195475220680237	0.9311339855194092	0.4963396489620209	0.43479397892951965
183	1.0149869918823242	0.56406569480896	0.45092126727104187	0.9285990595817566	0.4953297972679138	0.43326887488365173
184	1.0141502618789673	0.5650578141212463	0.4490930438041687	0.9311828017234802	0.4968419075012207	0.4343404471874237
185	1.0113434791564941	0.5644071102142334	0.4469369649887085	0.9302657842636108	0.49569007754325867	0.43457549810409546
186	1.0084716081619263	0.5646283626556396	0.4438430368900299	0.9308502674102783	0.4956999719142914	0.435150146484375
187	1.0022629499435425	0.5589621067047119	0.443300724029541	0.9337139129638672	0.4974867105484009	0.4362274706363678
188	1.010462760925293	0.563588559627533	0.44687432050704956	0.93216872215271	0.4964750111103058	0.4356934130191803
189	1.0098745822906494	0.5669596791267395	0.442915141582489	0.9316402077674866	0.49689289927482605	0.43474724888801575
190	1.0072517395019531	0.5629821419715881	0.44427043199539185	0.9326554536819458	0.49698570370674133	0.4356696903705597
191	1.0109952688217163	0.5646440386772156	0.4463520646095276	0.9331636428833008	0.49680081009864807	0.4363626539707184
192	1.0179054737091064	0.5668085813522339	0.45109742879867554	0.929749608039856	0.4956516623497009	0.4340978264808655
193	1.0050342082977295	0.5620152354240417	0.44301944971084595	0.9339377880096436	0.4967551827430725	0.43718260526657104
194	1.011902093887329	0.5628689527511597	0.4490337669849396	0.9311550259590149	0.49691882729530334	0.4342365562915802
195	1.0149061679840088	0.5669757723808289	0.44793006777763367	0.9342151880264282	0.4976752996444702	0.4365396499633789
196	1.0118377208709717	0.5642898082733154	0.44754764437675476	0.9292335510253906	0.49610641598701477	0.43312767148017883
197	1.0059762001037598	0.5607737302780151	0.4452028274536133	0.933688223361969	0.4975859522819519	0.4361023008823395
198	1.0087275505065918	0.5613117814064026	0.4474164843559265	0.9331406950950623	0.49780598282814026	0.43533486127853394
199	1.0114206075668335	0.5626768469810486	0.4487435221672058	0.9366050958633423	0.4984317123889923	0.4381736218929291
200	1.0048301219940186	0.5593607425689697	0.4454685151576996	0.9334269762039185	0.49876877665519714	0.43465766310691833
201	1.0095527172088623	0.56533282995224	0.44422057271003723	0.9352532625198364	0.4979263246059418	0.43732738494873047
202	1.012204885482788	0.5634075403213501	0.44879743456840515	0.9331136345863342	0.4975895583629608	0.4355241656303406
203	1.0067756175994873	0.5617693066596985	0.44500675797462463	0.9322689175605774	0.49661508202552795	0.43565377593040466
204	1.0120153427124023	0.5632447004318237	0.448770135641098	0.9336621761322021	0.49776312708854675	0.4358990490436554
205	1.0040136575698853	0.5589050650596619	0.4451097548007965	0.9352455139160156	0.49871519207954407	0.436530739068985
206	1.009358525276184	0.5645768046379089	0.44478222727775574	0.9332889914512634	0.49693068861961365	0.43635818362236023
207	1.0095804929733276	0.5639387965202332	0.44564247131347656	0.9362379312515259	0.4985688030719757	0.43766945600509644
208	1.0096646547317505	0.5686319470405579	0.4410324692726135	0.9386622905731201	0.49899426102638245	0.4396677315235138

Table A.1 continued from previous page

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
209	1.0045340061187744	0.5613900423049927	0.4431437849998474	0.9365870356559753	0.49852699041366577	0.43806037306785583
210	1.005076289176941	0.5619632601737976	0.44311344623565674	0.9351778030395508	0.498038649559021	0.43713924288749695
211	1.0003092288970947	0.5593581795692444	0.4409516453742981	0.9379473924636841	0.49831607937812805	0.43963125348091125
212	1.002041220664978	0.5601845383644104	0.441855788230896	0.9333047270774841	0.49761953949928284	0.43568500876426697
213	1.0003490447998047	0.5617542862892151	0.43859419226646423	0.9348194003105164	0.4970645606517792	0.4377545416355133
214	1.0013240575790405	0.5609144568443298	0.44040998816490173	0.9351730346679688	0.4975537657737732	0.43761947751045227
215	1.0044969320297241	0.560567319393158	0.4439300298690796	0.9366010427474976	0.4986927807331085	0.43790820240974426
216	1.0043432712554932	0.5613779425621033	0.4429660439491272	0.9360083937644958	0.49693530797958374	0.43907299637794495
217	0.9994690418243408	0.5608779191970825	0.43859168887138367	0.9363541007041931	0.4991658329963684	0.43718820810317993
218	1.0060851573944092	0.5594073534011841	0.44667795300483704	0.9363306760787964	0.4996338486671448	0.43669694662094116
219	1.0051872730255127	0.5618908405303955	0.4432963728904724	0.9337584376335144	0.49741077423095703	0.43634727597236633
220	1.0067857503890991	0.5654515027999878	0.44133421778678894	0.9356008768081665	0.4987417459487915	0.43685922026634216
221	1.0048868656158447	0.5623752474784851	0.4425121247768402	0.9365498423576355	0.4983486533164978	0.4382012188434601
222	1.0034369230270386	0.559360921382904	0.44407540559768677	0.938368558883667	0.49979981780052185	0.4385690987110138
223	0.9978277683258057	0.5585823655128479	0.43924465775489807	0.9371516704559326	0.49985018372535706	0.4373011887073517
224	1.002355933189392	0.5602257251739502	0.4421303868293762	0.9348543286323547	0.4982748329639435	0.43657949566841125
225	1.0046554803848267	0.5613739490509033	0.4432810842990875	0.9363396763801575	0.4992222487926483	0.4371170699596405
226	1.0057168006896973	0.5607449412345886	0.4449719488620758	0.9399279952049255	0.5013396143913269	0.4385882616043091
227	1.0035626888275146	0.5614485144615173	0.44211456179618835	0.9386551380157471	0.5011641383171082	0.4374905228614807
228	1.0029493570327759	0.5620381832122803	0.44091179966926575	0.9386416077613831	0.4997248351573944	0.4389168322086334
229	1.0036870241165161	0.5619088411331177	0.44177791476249695	0.9383774399757385	0.4984394907951355	0.43993791937828064
230	1.0054925680160522	0.5634271502494812	0.44206610321998596	0.9382073879241943	0.49879080057144165	0.4394156336784363
231	1.0028622150421143	0.5626165270805359	0.4402454197406769	0.9373486638069153	0.49806302785873413	0.4392857253551483
232	0.9952123165130615	0.5562523603439331	0.4389595091342926	0.9386739134788513	0.5002731084823608	0.4384010136127472
233	1.0032204389572144	0.5599488615989685	0.4432721436023712	0.9412303566932678	0.5015033483505249	0.4397270977497101
234	0.9960607290267944	0.5583130717277527	0.43774718046188354	0.9416810870170593	0.5013591051101685	0.4403219223022461
235	1.0008268356323242	0.5592172145843506	0.4416103959083557	0.9401289820671082	0.5006250143051147	0.439503937959671
236	0.9968042969703674	0.5572798252105713	0.43952444195747375	0.9397736191749573	0.5009164810180664	0.4388572871685028
237	1.0033305883407593	0.5619306564331055	0.441398561000824	0.9384805560112	0.4989146590232849	0.43956610560417175
238	1.002616047859192	0.5609740614891052	0.44164136052131653	0.9385055303573608	0.5010517239570618	0.4374536871910095
239	0.9994375705718994	0.5602509379386902	0.439186692237854	0.9383321404457092	0.4995696246623993	0.4387625455856323
240	1.0034962892532349	0.5614157915115356	0.44208019971847534	0.9390134811401367	0.5000278949737549	0.43898528814315796
241	1.0002132654190063	0.5593069791793823	0.4409070611000061	0.9372215867042542	0.4978722035884857	0.4393496811389923
242	0.9975730776786804	0.561967670917511	0.4356050193309784	0.9390024542808533	0.5003935694694519	0.43860870599746704
243	1.0002202987670898	0.5581273436546326	0.4420938491821289	0.9397771954536438	0.5010464191436768	0.4387308359146118
244	0.9981772899627686	0.5611728429794312	0.4370042681694031	0.9433889389038086	0.5018019676208496	0.441586971282959
245	0.9986715912818909	0.5597946643829346	0.43887630105018616	0.9443122148513794	0.5004619359970093	0.44385018944740295
246	0.9953332543373108	0.5554688572883606	0.4398641586303711	0.9409132599830627	0.5015206933021545	0.43939265608787537
247	0.9931190013885498	0.5546157956123352	0.4385037124156952	0.9431121945381165	0.5025656819343567	0.44054603576660156
248	0.9946774244308472	0.5553562045097351	0.4393211305141449	0.9433215856552124	0.5017327070236206	0.44158926606178284
249	0.9933452010154724	0.5570454001426697	0.4362994432449341	0.9429627060890198	0.5016821026802063	0.44128090143203735
250	0.9914389848709106	0.5552685260772705	0.43617019057273865	0.9441039562225342	0.5022728443145752	0.4418310821056366
251	0.9937124252319336	0.5562232732772827	0.43748942017555237	0.9425495266914368	0.5017534494400024	0.4407960772514343
252	0.9990050196647644	0.5588817596435547	0.4401226341724396	0.943905234336853	0.503187358379364	0.4407179653644562
253	1.0003849267959595	0.5611705780029297	0.43921419978141785	0.9460873007774353	0.5046283602714539	0.441458523273468
254	0.9980008006095886	0.5578826069831848	0.4401186406612396	0.9433848857879639	0.5028365254402161	0.44054827094078064
255	0.9947952032089233	0.5560591816902161	0.43873608112335205	0.9449889063835144	0.5038674473762512	0.4411212205886841
256	0.9957295060157776	0.5601661801338196	0.4355625510215759	0.9440086483955383	0.50202876329422	0.441980242729187
257	0.9901061654090881	0.5566972494125366	0.43340909481048584	0.9456049203872681	0.5021912455558777	0.44341355562210083
258	0.989719569683075	0.5528366565704346	0.43688279390335083	0.9439164996147156	0.5023820400238037	0.4415345788002014
259	0.9905751943588257	0.5550554990768433	0.43551960587501526	0.9432510733604431	0.5029900670051575	0.4402613043785095
260	0.9939877390861511	0.5580857396125793	0.43590205907821655	0.9453449249267578	0.502878725528717	0.4424663484096527
261	0.9942054152488708	0.5577381253242493	0.4364667236804962	0.9430965185165405	0.5021150708198547	0.44098132848739624
262	0.9941205978393555	0.5574031472206116	0.4367174506187439	0.9484463930130005	0.5044525861740112	0.4439937472343445
263	0.9895194172859192	0.5547767281532288	0.434742271900177	0.947275698184967	0.5034597516059875	0.4438158869743347
264	0.9988983273506165	0.559112012386322	0.4397861063480377	0.9475221633911133	0.5032411217689514	0.44428110122680664
265	0.9952409863471985	0.5594831109046936	0.43575742840766907	0.9478034973144531	0.5043551921844482	0.4434480667114258
266	0.9945647120475769	0.5540851354598999	0.44047942757606506	0.950228750705719	0.5050081610679626	0.445220410823822

Table A.1 continued from previous page

	loss	policy_net_loss	value_net_loss	val_loss	val_policy_net_loss	val_value_net_loss
267	0.9880883097648621	0.5543086528778076	0.4337801933288574	0.9475696682929993	0.5038408041000366	0.4437287151813507
268	0.9983953833580017	0.5597705245018005	0.4386249780654907	0.9469287395477295	0.5037143230438232	0.4432142972946167
269	0.9888975024223328	0.5537710785865784	0.4351261556148529	0.94930100440979	0.5032076835632324	0.4460936188697815
270	0.9918916821479797	0.5542848110198975	0.43760716915130615	0.9471441507339478	0.5038893222808838	0.44325506687164307
271	0.9969497323036194	0.5602486729621887	0.4367004632949829	0.9473037123680115	0.5040683150291443	0.44323498010635376
272	0.9898043870925903	0.5557802319526672	0.4340246915817261	0.9466174244880676	0.5038204789161682	0.44279715418815613
273	0.992627739906311	0.5544851422309875	0.43814197182655334	0.9485705494880676	0.5037086606025696	0.4448617994785309
274	0.9944506883621216	0.5587907433509827	0.4356597363948822	0.9494113326072693	0.5041213035583496	0.4452902674674988
275	0.9942619204521179	0.5582864284515381	0.4359748661518097	0.9536365270614624	0.5069758296012878	0.44666069746017456
276	0.9936226606369019	0.5582395792007446	0.4353833496570587	0.9503892064094543	0.504633367061615	0.4457562267780304
277	0.9886900186538696	0.5551004409790039	0.4335889220237732	0.9470466375350952	0.5041482448577881	0.44289812445640564
278	0.9889993667602539	0.5544964075088501	0.43450289964675903	0.9536550641059875	0.5055358409881592	0.44811880588531494
279	0.9922619462013245	0.5547406077384949	0.4375205636024475	0.9465362429618835	0.5039126873016357	0.4426231384277344
280	0.9857951998710632	0.5531835556030273	0.4326123595237732	0.9478769302368164	0.5050527453422546	0.4428243041038513
281	0.9911256432533264	0.5564438104629517	0.43468156456947327	0.9472343921661377	0.5055626034736633	0.441671758890152
282	0.9894036054611206	0.5568118691444397	0.4325915575027466	0.9470367431640625	0.504158079624176	0.4428790807723999
283	0.9908217787742615	0.5571851134300232	0.4336368441581726	0.949933648109436	0.5054536461830139	0.4444800913333893
284	0.9879868030548096	0.5549989342689514	0.43298789858818054	0.9531517028808594	0.5070666670799255	0.44608446955680847
285	0.9852899312973022	0.5551581978797913	0.4301310181617737	0.951856255531311	0.5077223181724548	0.4441337287425995
286	0.9922258257865906	0.5578976273536682	0.43432822823524475	0.9500274658203125	0.506120502948761	0.44390666484832764
287	0.9949134588241577	0.5588733553886414	0.4360400140285492	0.9485616683959961	0.5053067207336426	0.4432543218135834
288	0.9936727285385132	0.557328462600708	0.4363454282283783	0.9508365988731384	0.5058395862579346	0.4449976086616516
289	0.9837911128997803	0.5543424487113953	0.4294489622116089	0.9517180323600769	0.5062809586524963	0.44543713331222534
290	0.9844657182693481	0.5541254281997681	0.43033990263938904	0.9494717121124268	0.5061826109886169	0.4432887136936188
291	0.9838549494743347	0.5508860945701599	0.43296903371810913	0.9506162405014038	0.5063633918762207	0.4442530572414398
292	0.9863754510879517	0.5550476908683777	0.43132808804512024	0.9489202499389648	0.5051538944244385	0.443766713142395
293	0.9881338477134705	0.5544527173042297	0.43368083238601685	0.9479323029518127	0.5060405135154724	0.44189199805259705
294	0.9829780459403992	0.553572952747345	0.4294055104255676	0.9506534934043884	0.5057621598243713	0.44489097595214844
295	0.9879100918769836	0.5551950931549072	0.4327157437801361	0.9507713913917542	0.5065964460372925	0.444175124168396
296	0.9899653792381287	0.5554571151733398	0.4345085918903351	0.9512231945991516	0.5065385699272156	0.44468414783477783
297	0.9838765859603882	0.5536510944366455	0.4302251935005188	0.949711799621582	0.5061488151550293	0.44356271624565125
298	0.9877735376358032	0.5560689568519592	0.43170350790023804	0.950348973274231	0.5060612559318542	0.44428750872612
299	0.9902716875076294	0.5560818910598755	0.4341893196105957	0.948144793510437	0.5055281519889832	0.4426170885562897

B

Configurations

B.1 Commands for VM configuration

```
wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb;
sudo dpkg -i packages-microsoft-prod.deb;
sudo apt-get -y install apt-transport-https;
sudo add-apt-repository universe;
sudo apt-get update;
sudo apt-get -y install p7zip-full p7zip-rar;
sudo apt-get -y install aspnetcore-runtime-3.1;
sudo apt-get -y install dotnet-sdk-3.1;
```

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
sudo apt-get -y install python3-distutils
sudo apt-get -y install python3-apt
python3 get-pip.py
python3 -m pip install tensorflow
python3 -m pip install IPython
python3 -m pip install scikit-learn
sudo apt-get -y install python3-pandas
```

B.2 Special configuration AWS free tier

```
sudo fallocate -l 1G /swapfile
sudo dd if=/dev/zero of=/swapfile bs=1024 count=1048576
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```