ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DISI

Corso di Laurea in INGEGNERIA INFORMATICA MAGISTRALE

TESI DI LAUREA

in

FONDAMENTI DI INTELLIGENZA ARTIFICIALE M

Symbolic versus sub-symbolic approaches: a case study on training Deep Networks to play Nine Men's Morris game

CANDIDATO
Galassi Andrea

RELATORE:
Chiar.ma Prof.ssa Mello Paola

CORRELATORI
Ing. Chesani Federico
Dott. Lippi Marco

Anno Accademico 2015/2016

Sessione III

“The answer that came to me again and again was play.

Every human society in recorded history has games.

We don't just solve problems out of necessity.

We do it for fun.

Even as adults.

Leave a human being alone with a knotted rope

and they will unravel it.

Leave a human being alone with blocks

and they will build something.

Games are part of what makes us human.

We see the world as a mystery,

a puzzle,

because we've always been a species of problem-solvers.”

-

“The Talos Principle”

Table of Contents

List of Figures and Tables

Abstract

In the last few years, due to the new Deep Learning techniques, artificial neural networks have completely revolutionized the technologic landscape, demonstrating themselves effective in many tasks of Artificial Intelligence and similar research fields. Therefore it could be interesting to analyse how and by what measure deep networks can replace symbolic AI systems. After the impressive results obtained in the game of Go, the game of Nine Men's Morris has been chosen as case of study in this work, because it is a widely spread and deeply studied board game. Therefore, the Neural Nine Men's Morris system has been created, a completely sub-symbolic program which uses three deep networks to choose the best move for the game. Networks have been trained over a dataset of more than 1,500,000 pairs (game state, best move), created according to the choices of a symbolic AI system. The tests have demonstrated that the system has learnt the rules of the game, predicting a legal move in more than 99% of the cases. Moreover, it has reached an accuracy on the dataset of 39% and has developed its own game strategy, which results to be different from its trainer one, proving itself to be a better or a worse player according to its adversary. Results achieved in this case study show that the key issue in designing state-of-the-art AI systems in this context seems to be a good balance between symbolic and sub-symbolic techniques, giving more relevance to the latter, with the aim to reach a perfect integration of these technologies.

Le reti neurali artificiali, grazie alle nuove tecniche di Deep Learning, hanno completamente rivoluzionato il panorama tecnologico degli ultimi anni, dimostrandosi efficaci in svariati compiti di Intelligenza Artificiale e ambiti affini. Sarebbe quindi interessante analizzare in che modo e in quale misura le deep network possano sostituire le IA simboliche. Dopo gli impressionanti risultati ottenuti nel gioco del Go, come caso di studio è stato scelto il gioco del Mulino, un gioco da tavolo largamente diffuso e ampiamente studiato. È stato quindi creato il sistema completamente sub-simbolico Neural Nine Men's Morris, che sfrutta tre reti neurali per scegliere la mossa migliore. Le reti sono state addestrate su un dataset di più di 1.500.000 coppie (stato del gioco, mossa migliore), creato in base alle scelte di una IA simbolica. Il sistema ha dimostrato di aver imparato le regole del gioco proponendo una mossa valida in più del 99% dei casi di test. Inoltre ha raggiunto un'accuratezza del 39% rispetto al dataset e ha sviluppato una propria strategia di gioco diversa da quella della IA addestratrice, dimostrandosi un giocatore peggiore o migliore a seconda dell'avversario. I risultati ottenuti in questo caso di studio mostrano che, in questo contesto, la chiave del successo nella progettazione di sistemi AI allo stato dell'arte sembra essere un buon bilanciamento tra tecniche simboliche e sub-simboliche, dando più rilevanza a queste ultime, con lo scopo di raggiungere la perfetta integrazione di queste tecnologie.

Chapter 1

Introduction

Artificial neural networks first appeared more than 60 years ago and in the last decade have once again improved artificial intelligence researches and applications, achieving impressive results in tasks such as image and speech recognition and classification, natural language processing, sentiment analysis, and game playing.

The new deep learning techniques take advantage from a huge amount of unstructured data and knowledge and the impressive computational power of modern computer architectures in order to define very fast and effective machine learning algorithms, considered very interesting and promising by all the major ICT companies that are currently investing on them [1].

As evolution of neural networks, deep learning technologies are sub-symbolic techniques that does not require an explicit representation and modelling of the problem to be solved. The solution is "hidden" inside the configuration of the networks and in the weights of its connections. On the other hand, symbolic systems, in which knowledge and reasoning are expressed through rules and symbols manipulation, are still used in several Artificial Intelligence applications since they are generally more reliable, transparent and seem to perform better when solving problems where knowledge is explicit and specialized.

Nowadays, a very important research issue in the Artificial Intelligence area is trying to exploit the advantages of both symbolic and sub-symbolic approaches, combining or integrating them in a single hybrid system, therefore solving their apparent dichotomy.

If we consider the context of board games, traditionally, Artificial Intelligence researchers have used symbolic techniques to approach the challenge to play, obtaining the impressive result of defeating human champions in many classical games like Chess, Draughts or Othello. Combining symbolic systems and neural networks, relying on the latter for tasks such as the evaluation of game states, the new hybrid systems have demonstrated themselves better player than their "ancestors", being able to triumph even in the game of Go. Recognized as one of the most difficult

board game existing, until the last year the game of Go was considered too difficult for an AI system, but the hybrid program *AlphaGo* proved this belief wrong, earning itself the title of first computer program to defeat a Go international champion.

A spontaneous question which rises is if a sub-symbolic system could substitute entirely a symbolic one. In particular, in the context of board games, the system should learn the game rules and constraints and optimizing some objective, providing a solution which is both good and acceptable without any a priori explicit knowledge about the problem and without any human intervention. If this is the case, what are the strengths and weaknesses of this approach with respect to a symbolic one and how could them be merged together in a synergic way?

The purpose of this thesis is to investigate this issue and try to give a first answer to this question by using a case study and therefore by designing, implementing testing a pure sub-symbolic system able to play a popular board game, Nine Men's Morris, which is wide-spread, deeply studied and solved.

The product of this work is Neural Nine Men's Morris (NNMM), a program able to play the game following its rules and taking smart decisions using only sub-symbolic machine learning techniques based on neural networks.

To train it, a dataset of 1,628,673 "good moves" has been created, which contains a set of states and the corresponding best moves according to a symbolic AI, that has the role of "teacher" of NNMM.

What is aimed to achieve is not a symbolic AI system that chooses the best move according to explicit rules and heuristics by considering future, possible moves (states) that it and its adversary will make and therefore trying to identify the more promising move that would lead to win the game (goal state). The NNMM system is sub-symbolic instead and will learn to play simply following its "instinct", developed after considering a huge number of examples, rather than following, a priori, a complex strategy which explicitly involves knowledge elicited from human expert player about the game.

Using supervised learning and neural networks, the desired system will independently learn to recognize some patterns and features and to associate them to a particular move, so it will be able to predict the best move simply by watching the board. Furthermore, fed with thousands possible moves, this system will understand when a move is considered legal, even though nobody has explained it the rules of the game.

To make a comparison, the system will not be like a person who learns to play reading the rule book or strategy guides, but simply watching an expert playing.

To verify if these goals have been accomplished, Neural Nine Men's Morris will be tested firstly confronting its choices with the dataset ones and verifying if it respects the game rules, than its skill as player will be tested playing against other AI, among which there will be its "teacher".

Chapter 2 presents the problems of representing knowledge and how a system can increase its knowledge, describing the main approaches and the different types of learning.

Chapter 3 examines the game of Nine Men's Morris, analysing its rules, the way it can be represented in a computer system and comparing it to other popular board games.

Chapter 4 explains neural network models, showing some architectures invented through the decades and illustrating the principles behind them, finally focusing on approaches that have been proved successful playing board games.

Chapter 5 describes what has been the approach to the problem, the dataset that has been created, the system which has been designed, its neural networks and how they have been realized and trained.

Chapter 6 concerns the tests that have been done to tune the networks and evaluate the system in terms of accuracy with respect to the dataset, respect of game rules and skill as player against other AI.

Chapter 7 sums up what tests have proven and proposes possible future works on this subject, both as improvements to NNMM and further studies on the sub-symbolic systems.

Chapter 2

Knowledge and learning

Artificial intelligence is probably one of the computer science research fields that deals the most with philosophy or cognitive science. Apparently vague and abstract questions like “what is knowledge?” and “what means to learn” became fundamental, because they are strictly linked with very practical questions that researches deal with:

- How can knowledge be represented?
- How can a system be taught with new knowledge?

Knowledge representation and machine learning are considered two fundamental aspects for an AI that has the ambition to pass the Touring Test: it must store what it knows, it must adapt to new circumstances and be able to extrapolate patterns [2].

2.1 Symbolic approaches

The theory that human thinking is a kind of symbols manipulation and that can be expressed as formal rules is deeply-rooted in western philosophy and expressed partially by Hobbes, Leibniz, Hume and Kant [3]. Therefore, for long time the dominant theory has been that many aspects of intelligence could be achieved manipulating symbols, so the best way to represent knowledge could be only to use symbols.

This position is well embodied in Simon and Newell physical symbol system hypothesis:

> *“A physical symbol system has the necessary and sufficient means for general intelligent action” [4]*

Following this idea, what are called physical symbol systems (or formal systems) have been defined and realized: systems which use physical patterns called symbols, combine them into structures called expressions and manipulate them with processes to obtain new expressions.

Examples of symbolic systems are formal logic, algebra or the rules of a board game such as chess: the pieces are the symbols, the positions of the pieces on the boards are the expressions and the legal moves that modify those positions are the processes.

An important idea linked to this approach is the *search space* [5]. It is supposed that in any problem there is a space of states, defined by an initial state, a set of actions that can be done and a transition model that defines the consequences of the actions. Therefore the state space forms a direct graph in which the nodes are the states and the links are the actions. The solution to a problem is the sequence of actions that determine the path from the initial state to a goal state. Considering that a solution must begin from the initial state, the possible actions sequences form a search tree which has the initial state as root. Having a test that allows to determine if a given state is a goal state, an intelligent system can navigate the tree with the aim of founding a goal state and therefore the solution. To speed up the process, knowledge can be provided to the search algorithm, allowing it to evaluate the states and to infer faster a path to the goal [2].

Symbolic approaches are still researched today and have produced one of the first truly successful forms of artificial intelligence: *expert systems*. Built mainly by if-then rules, they are designed to solve complex problems and imitate the decision-making ability of human expert. They are *knowledge-based system*, which means that are made by two distinct part: a knowledge base that represent facts about the world and an inference engine that permit them to discover new knowledge basing on the one that is already possessed.

2.2 Sub-symbolic approaches

Physical symbol system hypothesis has been criticized under many aspects because, according to some researchers, does not resemble a human intelligence.

According to Dreyfus, part of the human intelligence derive from unconscious instincts that allow people to take quick decision using intuition; this aspect is unlikely to be captured by a formal rule [6].

The theory of *embodied cognition* affirms that symbol manipulation is just a little part of human intelligence, most of it depends by unconscious skills that derives from the body rather than the mind.

Brooks has written that sometimes symbols are not necessary: in the case of human basic skills like motion or perception they are a complicated representation of something that is far more simple [7].

The poor results obtained with symbol manipulation models, especially in their inability to handle flexible and robust processing in an efficient manner, lead in the 1980 to the *connectionist paradigm* [5]. It does not deny that at some levels human beings manipulate symbols, but suggest that this manipulation is not implied in cognition; it tries to model the source of all this unconscious skills and instinct as an interconnected network of simple and almost uniform units [2]. After an initial period of enthusiasm, the interest in connectionist models and systems had lowered for several years, but has rose again recently, stimulated by the great results achieved by the combination of new models and modern hardware.

Even though connectionist and symbolic approaches are viewed as complementary, not competing, it is very interesting to investigate their limits comparing their results on the same task.

2.3 Learning

> *"An agent is learning if it improves his performance on future tasks after making observations about the world. Learning can range from the trivial, as exhibited by jotting down a phone number, to the profound, as exhibited by Albert Einstein, who inferred a new theory of the universe" [2]*

The purpose of machine learning is to give to a system the ability of make predictions about unknown data, predictions that will be helpful for the users or for the system itself, letting it able to improve its performance on a task.

Taking advantage of a base knowledge acquired during training, the system infers new knowledge in the form of a model of the data and behaves according to it, letting the system able to perform tasks for which it has not explicitly programmed for.

Indeed, there are many scenarios in which the system cannot be programmed for all the possible situations in which it will have to act, so it is fundamental that it could take decisions by its own experience. For example, the number of possible configurations it could be too vast for being anticipated by the designer, or in a dynamic environment could be impossible to predict which

changes my occur during time or simply human programmers could have no idea how to program a solution [2].

Training can be divided into three categories according to the feedback that is provided to the system: *unsupervised learning*, *reinforcement learning* and *supervised learning*,

In unsupervised learning, the system analyse a set of known inputs and learns patterns that characterize the domain; a typical task is detecting potentially useful clusters of input example, which is called *clustering*.

Reinforcement learning aim to teach a system to take action in a variable environment; depending on how much the system output is appropriate to the environment state, the system is fed with a series reinforcements which can be rewards or punishments.

Supervised learning consists into giving to the system a set of couple of inputs and desired output, making it learn the relation between input and output.

In this work, to train the neural networks will be used supervised learning techniques, therefore it is necessary to describe this category in more detail.

2.3.1 Supervised Learning

Formally, the task of supervised learning is [2]:

> Given a *training set* of N example input-output pairs
>
> $$(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)$$
>
> where each y_j has been generated by an unknown function
>
> $$y = f(x)$$
>
> discover a function h that approximates the true function f

Function h is called hypothesis and learning means to search through the space of all the possible hypothesis for one that is consistent and generalize well.

A consistent hypothesis is an hypothesis which agrees with training data and it is said that generalizes well if it is able to predict the correct output for inputs which has not be trained on; this can be verified using only a part of the available data for the training set and using the remaining part as a *test set*.

According to the number of times that an hypothesis makes a correct or an incorrect prediction, there are several ways to measure its goodness. Given an hypothesis h, a couple (x, y) and calling a prediction $h(x)$ wrong if $y \neq h(x)$, correct otherwise, it is possible to define the *error rate* of h as the proportion between the number of wrong predictions and the number of total predictions, in the same way is possible to define the *accuracy* of h as the proportion between the number of correct predictions and the number of total predictions.

These measures could be not informative enough, because not all the wrong prediction could be equally negative: for example, in a mail system, could be better to label a spam mail as important rather than label an important mail as spam. Therefore is typical to use a *loss function* $L(x, y, \hat{y})$ that represents the cost of making a wrong prediction $h(x) = \hat{y}$ rather than a correct one; often it is used a simplified version $L(y, \hat{y})$ independent from x.

According to this, the best hypothesis is the one that minimizes the expected loss over all the pairs that the system will encounter. For finding it, the probability distribution of the couples should be known, but because it is generally not known must be estimated on the set of training examples E. It is so defined the *empirical loss*:

$$EmpLoss_{L,E}(h) = \frac{1}{N} \sum_{(x,y) \in E} L(y, h(x))$$

Therefore, the best hypothesis is the one that minimizes the empirical loss.

The two aims of finding an hypothesis that is so accurate to fit the data but also simple enough to generalise well are in conflict most of the times: optimizing the loss function probably will lead to *overfitting*, which is an excessive specialization of the hypothesis over the given example, leading to very low loss on the training set but an elevated loss on the test set.

The trade-off between the two objective can be expressed through *regularization*, which is the process of penalizing complex (less regular) hypothesis; during the search of the best hypothesis are considered both the loss function and a complexity measure, aiming to minimize their weighted sum.

$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h)$$

The best hypothesis becomes the one that minimizes the whole cost.

Chapter 3

Nine Men's Morris

Nine Men's Morris, also called with other names like Mill Game, Merrils, or Cowboy Checkers, is a strategy board game for two players.

It is a very ancient game [8], the oldest trace of it is in fact dated about 1400 BC, and has been played through the centuries by many different civilizations; nowadays is played in many country of the world, such as United States, United Kingdom, Italy, India, Algeria [9] and Somalia [10].

This game has been deeply analysed by Ralph Gasser: he programmed *Bushy*, an AI that has been able to defeat the British champion in 1990; later, in 1993, he proved that the solution to this game is a draw using a brute force approach, which is an exhaustive exploration of the possible game states [11].

More recently, even the *extended strong solutions* have been found [12], which are the theoretical results for all possible game states reachable with slightly different initial configuration of the game. In the same study, the game has been *ultrastrongly solved* too, which means that has been proposed a strategy that increases the chance of the player to achieve a result than is better than the theoretical one, maximizing the number of moves that leads to a loss and making distinction between draws.

This game has been chosen as subject of this study because of these characteristics:

- State space searches, so symbolic approaches, have been proved successful to solve and to play it.
- The complexity of the state space is not very big, so the process of training could not require excessive resources in terms of time and hardware.
- The choice of the best move implies several decisions and a legal move must satisfy constraints both on single decisions and their totality. So it will be interesting not only verifying if a sub-symbolic system is able to learn to do the best move, but also if is able to learn to do a fully legal move.

3.1 Rules

There are some variants of the grid and of the rules, the most common ones are presented here.

The game board, illustrated in Figure 3.1, consists in 3 concentric squares and 4 segments which link the midpoint of the sides of the squares. The intersections of two or more line create a grid of 24 points where checkers can be placed.

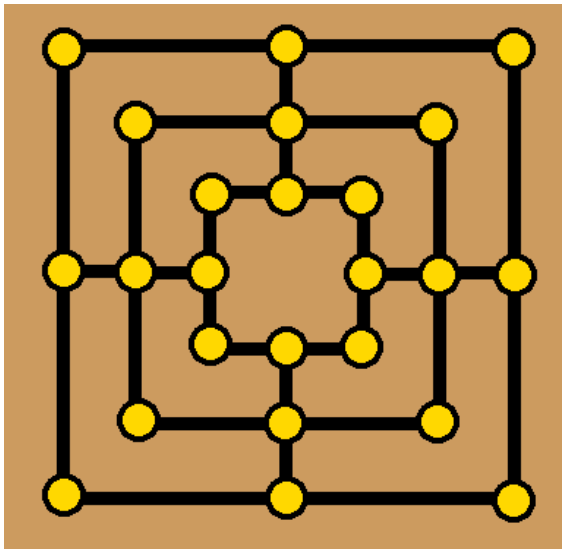

Figure 3.1 Representation of Nine Men's Morris game board

Each player has nine checkers (also called stones or men), usually coloured white for a player and black for the other one; the two player are called *white player* and *black player*, depending on the colour of their checkers.

When a player is able to align 3 checker along a line it is said he has "*closed a mill*" and he is allowed to remove from the game an opponent's checker which is on the board but is not aligned in a mill. The removed checker is sometime called "*eaten*" or "*captured*".

Initially the board is empty and each player has its own checker in hand, than player alternately make a move, starting with the white player.

The game proceed through 3 different phases that defines the moves that players are allowed to do:

1) Initially players alternately place a stone on an empty position of the board.
2) When both player have placed all their stones, they must slide a checker along a line to a nearby vacant point.

3) When a player is left with only 3 stones is able to "*fly*" or "*jump*": he can move a checker from a point of the board to any other empty point of the board

The game ends when one of this conditions occurs:

- A player wins removing 7 adversary stone, leaving the opponent with less than 3 stones
- A player is not able to make a legal move, so he loses
- During phase 2 or 3, a configuration of the board is repeated, so it's a draw.

In case during phase 1 a player is able to close two mills at the same time, he can remove only one checker.

In case a player close a mill but all opponent's checker are aligned in a mill, he is allowed to remove one aligned checker

3.2 Model of the problem

Nine Men's Morris is a *perfect-information game*, which means that all player, at any time during the game, access to all the information defining the game state and its possible continuation [13]. This means that the game state is shared by all players and that there are no stochastic elements to consider.

It is important to underline that two of the ending conditions can be detected simply by the state of the game, but for the third condition became necessary to maintain a history of the states presented during the game.

3.2.1 Symmetries and state space

Each configuration of the board can be transformed to obtain a symmetric configuration [11]. There are 5 axes of symmetry, as shown in Figure 3.2, but one is redundant because can be obtained as a combination of the other, so there are 4 axis that can lead up to 15 symmetric configurations.

In the particular case in which both player have only 3 checkers left, there is one more relevant symmetry because all the three squares are interchangeable (not only the inner and the outer one).

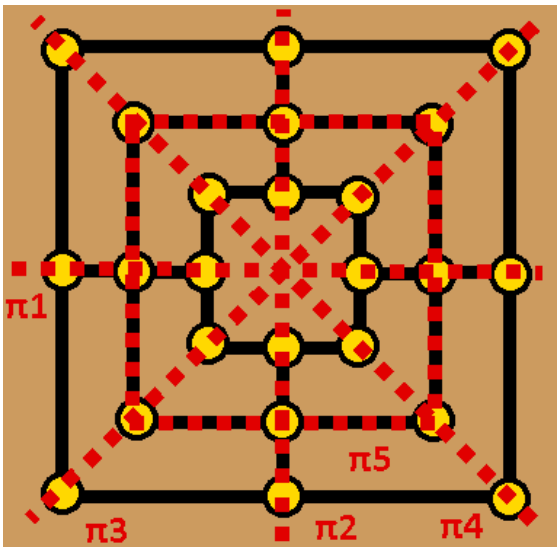

Figure 3.2 Symmetries in Nine Men's Morris game board

So how many configuration can the board present? Taking into account only the configurations of phase 2 and 3 we can make the following considerations.

Each of the 24 points of the board can be occupied by a white checker, a black checker or can be empty, so an upper bound for the number of possible states is 3^{24}, which is approximately 2.8×10^{11}.

However, it must be considered that every player has from 3 to 9 checker on the board and that some configuration are impossible: for example if a player has a closed mill, the other cannot have 9 checkers on the board.

If symmetries are considered too, it is found that the game has 7,673,759,269 possible states in phase 2 and phase 3 [11].

3.2.2 Representation of the state

The state of the game is made by 3 data: the state of the board, the phase of the game and the number of checkers each player has in his hand. The phase of the game can be deducted by the other two information, so only these two must be represented.

The number of checkers in players' hands can obviously be represented with two numbers.

Due to its own peculiarity, it is easy to represent the board as an object of 1, 2 or 3 dimensions.

1) It's possible to enumerate the grid points and represent the board as an array, where the value of index i represents the state of the i-th point. A possible enumeration is presented in Figure 3.3.

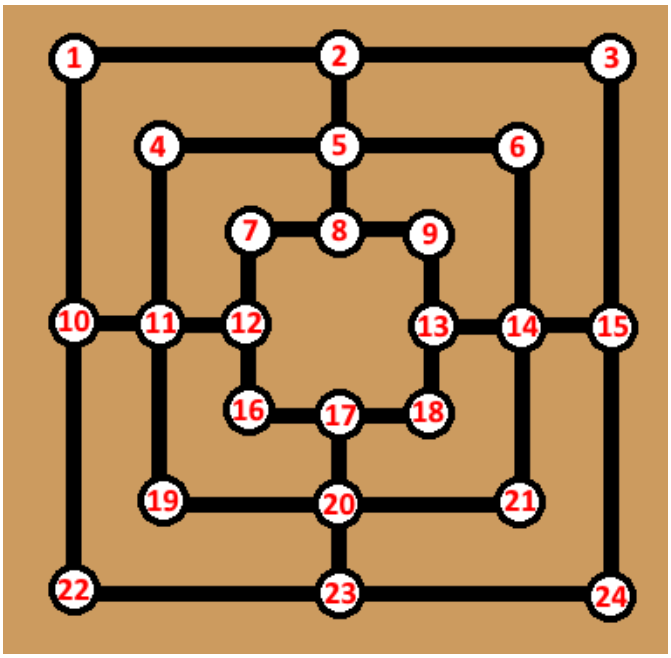

Figure 3.3 Enumeration of Nine Men's Morris board positions

2) Another simple representation of the board is as a 7x7 matrix, where only some elements of the matrix are relevant. Such representation is illustrated in Figure 3.4.

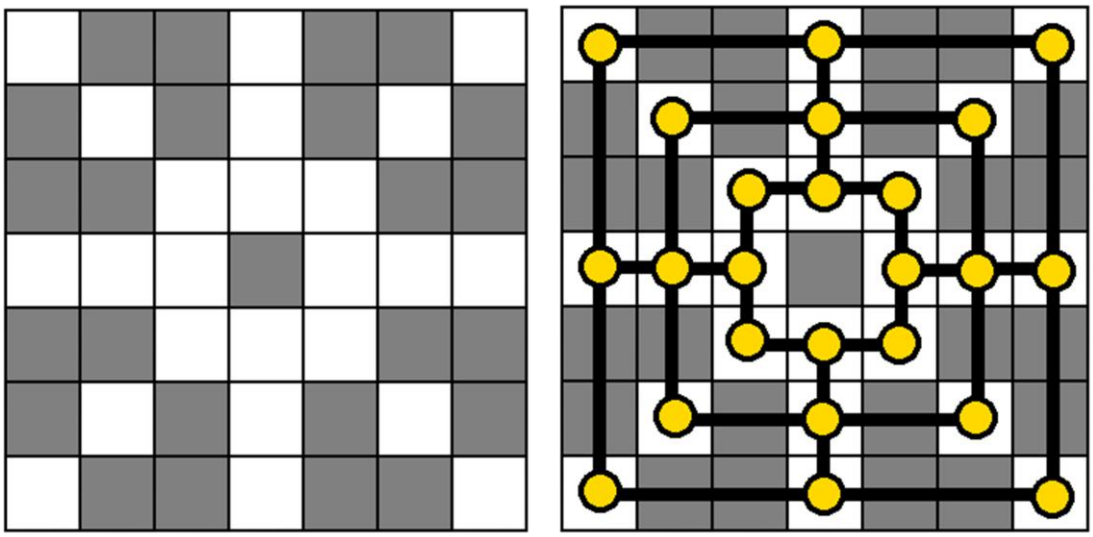

Figure 3.4 2D representation of Nine Men's Morris board (left) and relationship with the board (right). The white cells contains relevant elements.

3) The last representation is a three-dimensional object made by 3 matrix of size 3x3 which represents the three concentric squares of the grid. The center of each matrix is a not relevant element. Figure 3.5 presents such representation.

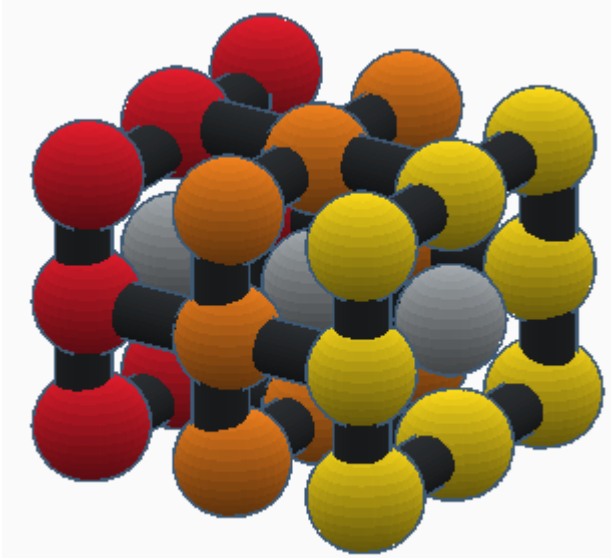

Figure 3.5 3D representation of the board state. The three colors, yellow, orange and red, differentiate the elements belonging respectively to the inner, the middle and the outer square of the board. The grey elements are not relevant.

Each point of the grid can assume three states: occupied by a white checker, occupied by a black checker or empty. So it's easy to represent the possible states of a point of the board with 3 different values, for example (1, -1, 0) or (W, B, E).

An important aspect to underline is that, as illustrated in Figure 3.6, none of the suggested representations of the board are capable to capture the concept of logical distance between two board points. For logical distance between two points is meant how much two points are distant according to the game rules and in this case can be defined by the minimum number of moves that are necessary for a checker to move from one point to another during phase 2.

1) Obviously the 1-dimensional representation cannot preserve the logical distance because imposes a linearization of the positions.
2) The 2-dimensional representation fails because weight differently the distance of two points of the same square according to which is the square: the distance between two logically adjacent points of the inner square will be 1, while the distance between two adjacent points in the middle one will be 2 and in the outer one will be 3: there is an overestimation of some distances.

3) The 3-dimensional representation fails because the points in the same angles of two different squares are represented at the same distance as the points in the midpoint of the same side of two different squares, but while the midpoint are linked so their distance is 1 or 2, the angles are not linked so their distance is 3 or 4: there is an underestimation of some distances.

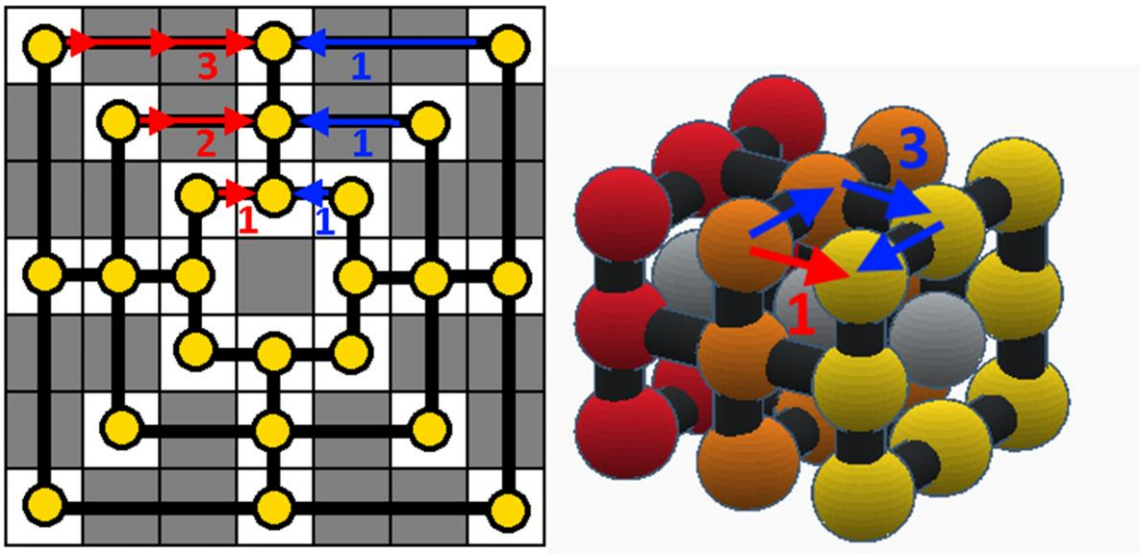

Figure 3.6 Differences between logical distance (in blue) and "physical" distance (in red) in the 2D representation (left) and the 3D representation (right) of the board

3.2.3 Representation of a move

With the term "*move*" is meant the whole set of choices that a player takes during his turn.

A very peculiar characteristic that make this problem different from many other board games is that a move is defined by a variable amount of information.

For the entire game, an information that is always needed to define a move is where a player want to place his checker, this can be represented with the coordinates of that point. The coordinates of a point of the grid can be a single number, a couple or a triplet, according on the chosen board representation. This information will be referred to as the "*TO*" move part and will always be present in the move.

In case that placing the checker causes the closing of a mill, another information that must be represented is the checker that the player wants to remove. This can be represented using the coordinates of the point where the checker is. This information will be referred to as the "*REMOVE*" move part and is the least frequent part in the move: during a single match, the

maximum number of moves with a REMOVE part is 13, 7 by the winning player and 6 by the losing one.

Finally, during phase 2 and 3, the checker is moved from a position to another, so another information is where the checker is moved from, one more time this information is a point of the grid so can be represented with the coordinates. This information will be referred to as the "*FROM*" move part.

So a move is defined by 1 to 3 coordinates among which only the first one is always present. For this reason a special point must be defined, a point that will be addressed by the *EAT* and *FROM* part in case the move do not require them.

It is important to underline that for any game state there are constraints on each part itself, but there are also constraints on the whole move: the legality of the parts does not guarantee the legality of the full move.

3.3 Comparison with other board games

It is useful to analyse briefly other popular strategy board games, with the aim of finding similarities and differences with Nine Men's Morris, in order to make considerations about which of the techniques that will be used in this case of study could be used for other games.

All the games examined are perfect-information games played by two players.

3.3.1 Draughts

Draughts or Checkers is another very wide-spread game, but unlike Nine Men's Morris, which rules are almost the same everywhere, rules of Draughts change very much depending on the place where is played: Polish draughts, Canadian checkers, Russian draughts, English draughts and Italian draughts are only a few examples of the many different traditional versions [14].

For this reason is difficult to compare precisely those two games: between different versions change the size of the board, the number of checkers and, more important, what are the possible legal moves.

What can be said for all versions is that the board is a squared checkerboard, so it can be easily represented as a 2-dimensional object, preserving the logical distance between two points and without the introduction of non-

relevant objects. There is not the possibility to have checkers on hand, so the board representation is sufficient to represent the game state; an example of game state is presented in Figure 3.7.

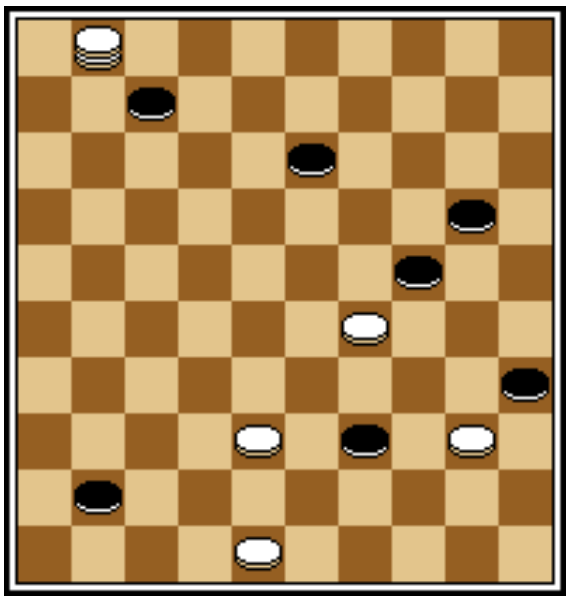

Figure 3.7 Example of game state in international draughts

Another difference is that the definition of legal moves is the same for the whole game, but there are two types of checkers, Men and Kings, which can do different moves; so a point on the board can assume 5 possible states.

The last important difference is that a move is defined by at least two points: the checkers that a player wants to move and the position where that checker will be moved. In the case of a move that captures more than one adversary checker, the move is defined also by the points in which the checker passes. So a move is formed by a variable number of parts, but if in Nine Men's Morris the parts that form a move are semantically a syntactically different, each part of a Draughts move has the same meaning as the others.

The game can end when a player loses either because he cannot make legal moves or because all his checkers are captured; each version of Draughts has its own conditions to determine the end in a draw, but substantially the game is declared a draw if none of the player has the possibility to win or if a certain state is reached many times. So while it is possible to determine if a player has won simply looking at the state, for detecting a draw it could be necessary to maintain history of the states.

The state-space complexity changes between different versions: for example for the English draughts is estimated an upper bound of 10^{20} [15], while for the international International draughts the upper bound is 10^{30} [16]. The English version has been weakly solved [17], which means that not only the

game-theoretic values of the starting position is known, but a strategy to achieve it is known as well.

In 1994, the computer program Chinook won the Checkers World Championship, achieving the title of first program to win a human world championship [18]. Chinook knowledge consisted of a library of opening moves, an incomplete end-game database and a move evaluation function which were used by a deep search algorithm to choose the best move. All this knowledge was programmed by Chinook creator, rather than obtained through machine learning.

3.3.2 Chess

Chess is different from Nine Men's Morris under almost any aspect.

The chess board is an 8x8 checkerboard so, as has been said for draughts, can be easily represented as a 2-dimensional object, with all the positive aspects previously mentioned, and example of representation is shown in Figure 3.8.

One more aspect in common with draughts is the presence of 6 different types of pieces and each one can do a different types of moves.

Figure 3.8 Example of game state in chess

A move is always defined by only two points, the one in which the piece is and the one in which the piece move, with a notable exception: the player who moves a pawn to the end of the board must decide the piece in which he wants the pawn get promoted to, but this decision occurs very few times during a game.

For almost any aspects, the chess board embodies the game state and the definition of legal moves remain the same for the whole game, the exception to this statement is the castling move. The castling move it's a special move that the king can do with an allied tower, only under certain conditions of the board and only if none of the two has been moved since the beginning of the game, so it's important to maintain these information in the game state. The castling move can still be represented with two points: for example the position of the king and of the chosen tower.

The game ends with a winner when a player is under checkmate, which means that his king is threatened with capture and there is no way to remove that threat, so that player have lost. The game ends with a draw if a player has no legal move possible (stalemate) or if none of the player can put the other in checkmate, this is possible if only some pieces remains on the board. These conditions are detectable simply looking at the state.

It is possible to end a game with a draw, under a player request, if one of two conditions occur: for 50 consecutive moves no pawn have been moved and no pieces has been captured or if the same state is reached for the third time (even not consecutively). The second condition is moreover relevant in an AI vs AI game, because it is possible for the players to remain stuck in a loop, so it is important to maintain a history of the game state.

The state-space complexity for chess has not been calculated precisely yet, the upper bound estimated by different authors vary between 10^{43} and 10^{50} [16]; due to its high complexity, chess is still an unsolved game.

IBM's computer Deep Blue was able to won a game against human world champion Kasparov in 1996 and to won a 6 game match the following year. Deep Blue relied on a vast resource of knowledge, such as a database of opening games played by past grandmasters, and applied a brute force approach, exploring that knowledge to figure out the best move. During the state search, the program considered the pieces on the board, their position, the safety of the King and the progress toward vantage states; the evaluation of all these components was made following the behaviour defined by the programmers, without the possibility to change it or adapt it to the opponent strategy [19].

"Kasparov isn't playing a computer, he's playing the ghosts of grandmasters past." [19]

3.3.3 Othello

Othello is the easiest game to model between the ones which has been analysed so far.

In Othello the board is a squared plane of size 8x8, so once again it can be represented as a 2-dimensional object, but like in Nine Men's Morris, only one type of checker exists.

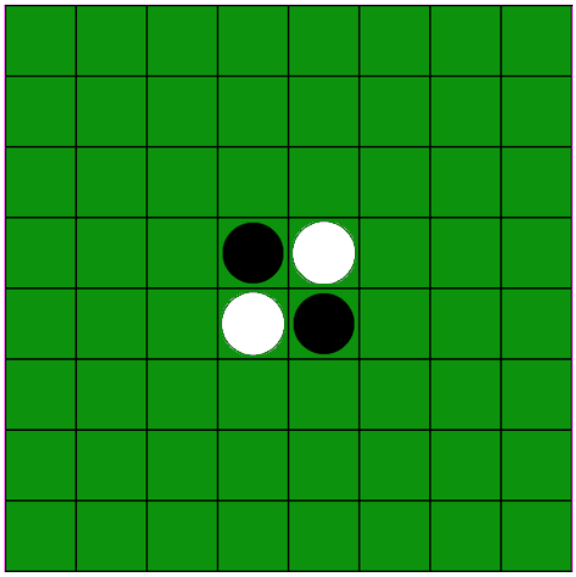

Figure 3.9 Initial game state in Othello

The board is sufficient to represent the game state, as illustrated in Figure 3.9, the definition of legal moves remain the same for the whole game and a move can be represented simply by a point on the board.

An important characteristic is that if a player cannot make a legal move, he must pass the turn, while in the games analysed so long the impossibility for a player to make a legal move meant the end of the game.

When none of the players can make a legal move, the game ends, and the player who has more checkers wins. So the end of the game can be detected simply by analysing the game state.

The state-space complexity has been estimated to be 10^{28} [16] and it is still mathematically unsolved.

The first Othello program computer able to won a single match against a human world champion was "The Moor" in 1980 [16], while in 1997 the program "Logistello" won a six games match against the human world champion.

Logistello relied upon a table-based pattern evaluation: to each occurrence of each pattern in each game state corresponded a value that would be added to the state value if those conditions were met. The search through the state space considered not only the best move but also some possible deviations, allowing the system to learn from his games [20].

3.3.4 Go

The game of Go is a very ancient oriental strategy board game. As for Draught, the wide diffusion of this game among centuries has led to the born of many different versions of the rules. Only the basic rules are considered here.

The game board is a grid made by 19x19 lines, even if it is possible to play with smaller boards, therefore it is easy to represent it with a 2-dimensional object, as shown in Figure 3.10.

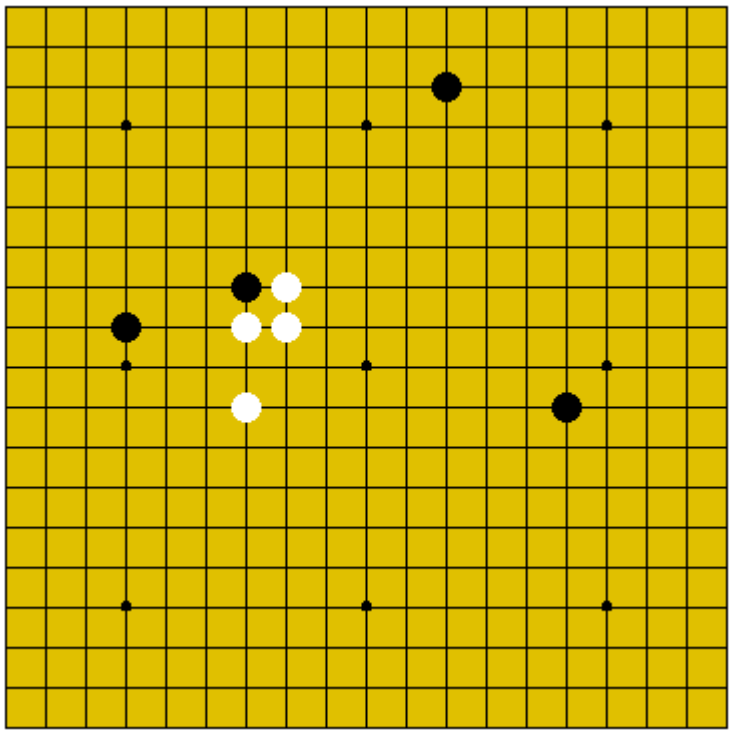

Figure 3.10 Example game state in go

During his turn, a player may pass or place a “stone” on the board, so a move can be represented simply by the position where the stone will be placed, using an illegal coordinate to indicate the choice to pass.

For the whole game the definition of legal move is the same, but it is illegal to recreate a board state which has occurred previously, so in order to define legal moves, it is important to maintain an history of the game states.

The game ends when both player passed and the outcome is determined by the last board configuration.

For this reason, the state of the game could be defined by the board state, the set of previously reached board configuration and the last move made by a player.

Go is probably the most complex traditional board game, because its complexity has been estimated to be 10^{172} [16] and is still unsolved.

The computer program FUEGO [21] is considered one of the best open source go computer player [22], it uses symbolic techniques like Alpha-Beta Search and Monte Carlo Tree Search to explore the possibile moves and to choose the better one.

Chapter 4

Artificial Neural Networks

Artificial Neural Networks (so forth called neural networks or simply networks) have a long history, which goes back to the first studies on computational models of biological neural networks by McCulloch and Pitts in 1943 [23]. Through time, many models have been proposed, different either in the architecture of the network or in the elements that constitute it or in the training techniques.

Partially inspired by the biological neuron model, as illustrated in Figure 4.1, Neural Networks are a sub-symbolic approach to the problem of learning. Made by many simple units linked together, neural networks store the learned knowledge into the connections between these units as a numerical weight and into other parameters.

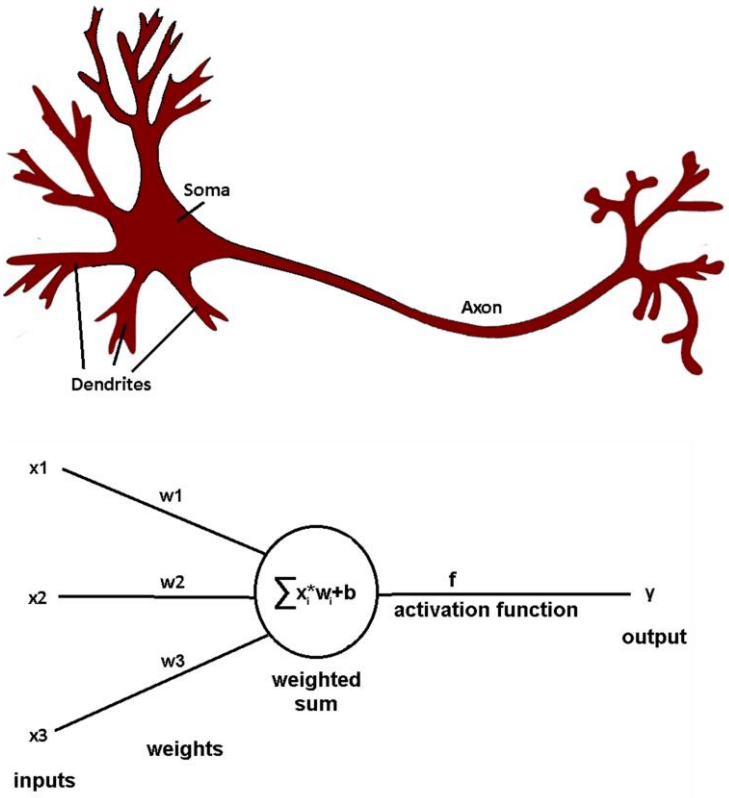

Figure 4.1 Simplified models of biological neuron and artificial neuron. The biological neuron gather impulses/inputs through dendrites, aggregates them in the soma and propagate the impulse/output with the axon

In a neural network, a *neuron* is a computational *unit* that receives a set of inputs x_i, elaborates a weighted sum with weights w_i and adds a bias b, than applies an activation function f to compute the output y. Each neuron has therefore as many parameters as the number of inputs plus one.

The output of each neuron can therefore be written as

$$Y(X) = f\left(\sum_i x_i w_i + b\right)$$

where x_i are the neuron inputs, while w_i and b are the parameters to be learned.

A typical activation function is the *sigmoid function*, but other possible activation functions will be discussed later. The plotting of this function is presented in Figure 4.2, while its definition is:

$$S(t) = \frac{1}{1 + e^{-t}}$$

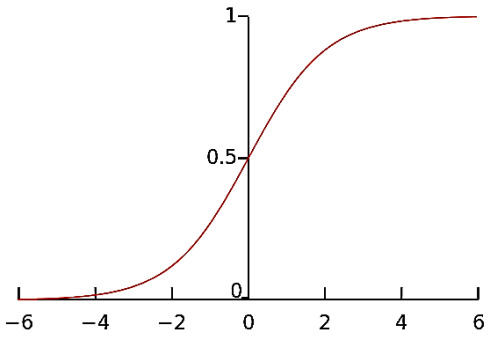

Figure 4.2 Sigmoid function plotting

4.1 Feed-forward model and network supervised training

In the *feed-forward neural network* model, neurons are grouped in layers, stacked one on top of the other. Therefore, the first layer of neurons (*input units*), receive the data as input, while the others (*hidden units*) receive the output of the previous layer as input. The output of the network is made by the outputs of each neuron that belongs to the last layer (*output units*). A scheme of a possible artificial neural network is presented in Figure 4.3.

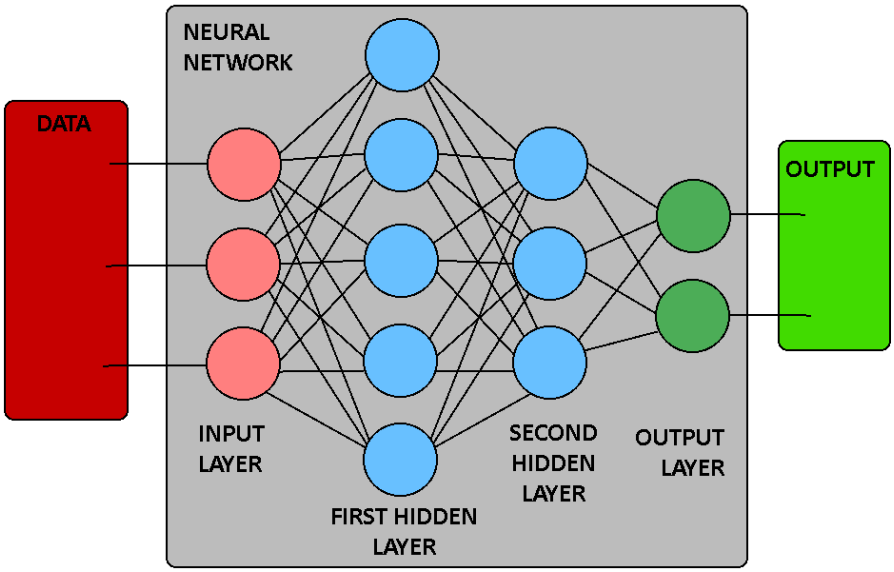

Figure 4.3 Feed-forward neural network architecture scheme

For any given data, the network calculate an output based on his parameters and hyper-parameters. Training a network means to alter these parameters with the aim to make the outputs of the network similar to the desired outputs or, more formally, to optimize a loss function L that has to measure the discrepancy between the output of the network and the target.

4.1.1 Gradient descent and backpropagation

If the activation function is differentiable, the *gradient descent* can be applied to optimize the loss function: the gradient of the loss function with respect to each parameter is computed, than the parameters are modified by the gradient multiplied by an hyper-parameter α, called *learning rate*. An iteration on the whole training set, and the related gradient computation, is called *training epoch*.

By applying this technique many times over the same data, it is possible to reach a local minimum of the loss function. The choice of the learning rate can dramatically influence the training: a small learning slows down the training and could make impossible to escape from a local minimum once it is reached, on the other hand an high learning rate could alter the weights too much on each gradient computation, making the global minimum impossible to reach.

If the activation function is not linear and is differentiable, it is possible to train the network using the *backpropagation* technique [24]. The idea behind

backpropagation is to do a two-steps process: a propagation phase and an updating phase.

During the forward propagation the data are given as input to the network and the output o_j of each neuron is kept in memory; during the backward propagation the error on the output is calculated and propagated backwards, computing the error δ_j associated to each neuron.

It is possible to compute the gradient of the loss function of any parameter of network following the *chain rule*: calling par_j a parameter of the j-th neuron and net_j the input to its activation function so that $o_j = f(net_j)$, it results

$$\frac{\partial L}{\partial par_j} = \frac{\partial L}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial par_j} = \delta_j \frac{\partial net_j}{\partial par_j}$$

Where δ_j is the error associated to the neuron and $\frac{\partial net_j}{\partial par_j}$ can be defined for the weights and the bias of the neuron as

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial\left(\sum_k o_k w_{kj} + b_j\right)}{\partial w_{ij}} = o_i$$

$$\frac{\partial net_j}{\partial b_j} = \frac{\partial\left(\sum_k o_k w_{kj} + b_j\right)}{\partial b_j} = 1$$

During the updating step it is thus possible to compute the gradient of the loss function with respect to each network parameter and finally apply the gradient descent.

4.1.2 Training process

Testing the trained network on the same data used for training could provide a misleading result: the networks performance could be good even in presence of overfitting. To verify the ability of the network to generalize, it is a common practice to split the dataset into two subsets: the training set and the test set.

During training it could be necessary to adjust some hyper-parameters, so it could be useful to test the network after each epoch of training. It is also important to underline that almost every network training will lead to an high overfitting, so it is important to stop the training at the right moment, a technique which is called *early stopping*.

The training set therefore could be divided into two subsets: the actual training set that will be used for training and a validation set, that will be used

for testing after each epoch. This division can be defined for the whole training or can change within an epoch; for example the *k-fold cross-validation* technique divides the training set into k partitions of the same size, during an epoch it does *k* training steps using one partition for validation and the others for training, computing the error and the accuracy scores as an average over the scores obtained in each step.

Monitoring the performance of the network over the validation set is possible to determine if the network is improving or has already reached its maximum performance and therefore the training can be stopped.

The amount of data used to calculate the gradient, and therefore to update the parameters, have repercussion on the accuracy and the speed of the training.

Computing the gradient over the whole training set (*batch optimization*) could require a great amount of time, therefore it is common to use *mini-batch optimization*, that means to compute the gradient only on a small subset of the data. The smaller the subset, the more inaccurate is the gradient estimation, but the faster is the gradient computation, so it will probably require more estimation but they will be faster. If the size of the batch is one, the gradient is computed independently for each training example; the approach is called *stochastic optimization*.

4.1.3 Softmax classifier and negative log likelihood loss

In classification tasks or, more generally, if the output of the network must be one class out of many, it is a very common practice to put a softmax classifier as last layer of the network [25]. The *softmax function* takes a set of real values as input, which are considered as un-normalized log probabilities, and map them into normalized probabilities.

More formally, calling $f_j(x)$ the score computed by the network for the class j with the provided input x, the probability P that the input x should be labelled as k is:

$$P(Y = k, x) = \frac{e^{f_k(x)}}{\sum_j e^{f_j(x)}}$$

From this probability score is possible to obtain a loss score computing the negative log likelihood.

In information theory the difference between a true distribution $t(x)$ and a distribution $p(x)$ can be evaluated with the *cross-entropy* between them, which is defined as:

$$H(t,p) = -\sum_{x} t(x) \log p(x)$$

If as $t(x)$ is considered a distribution where all probability mass is on the correct class, so it is formed by only zeros except a 1 on the correct class, and as $p(x)$ is considered the probability distribution resulting from the softmax function, the result is the *negative log likelihood* of the correct class k, which will be used as loss function:

$$L_k = -\log P(Y = k,\ x) = -\log\left(\frac{e^{f_k(x)}}{\sum_j e^{f_j(x)}}\right)$$

4.1.4 Regularizations

Two common regularization penalties used in the loss function are the L1 and L2 norms, which discourage the use of large weights in the network to obtain a more general model. The penalty is added to the loss function with a weight λ which is an hyper-parameter of the training. Calling $W_{i,j}$ the weight of the connection between neurons i and j, the two penalties are defined as:

$$R_{L2}(W) = \sum_i \sum_j {W_{i,j}}^2$$

$$R_{L1}(W) = \sum_i \sum_j \left|W_{i,j}\right|$$

4.1.5 Learning rate annealing

On one hand a low learning rate requires many epochs of training to reduce the error significantly, but on the other hand a high learning rate slows down or prevents the system from achieving highest accuracy.

The solution to this problem is the *learning rate annealing*, which means to reduce the learning rate after some epochs of training, obtaining an heavy and fast error reduction at first and a slow but constant one later.

Many learning rate annealing techniques have been designed:

- Step decay: every few epochs the learning rate is reduced by some factor, for example it can be halved every 10 epochs, or divided by 10 every 20 epochs.
- Exponential decay: an initial learning rate α_0 is established, than the learning rate for each epoch is defined as $\alpha = \alpha_0 e^{-kt}$, where t is the epoch and k is an hyper-parameter
- Proportional decay: similar to the previous one, but the learning rate is defined as $\alpha = \frac{\alpha_0}{(1+kt)}$

4.1.6 Ensemble learning

To achieve better results, it is a common machine learning technique to train more than one model and use them together to achieve better results. In *bagging ensemble* [26] all the models are trained independently and contribute equally to the final prediction, while in *boosting ensemble* [27] each model is trained with the aim of fixing the errors of the previous ones.

In the case of neural networks, it is possible to train different models on the same data changing the architecture or an hyper-parameter, or train the same model on different data, or even use the same model trained on the same data but for a different amount of epochs.

4.2 Historical background

To better understand which are the problems which concerns modern networks and how they have been addressed, it useful to present the evolution of neural networks technologies since their invention.

4.2.1 Neuron and perceptron model

In 1943, McCulloch and Pitts presented the first mathematical model based on human neurons [23]. The *artificial neuron* took a weighted sum of inputs with weight equals to +1 or -1 and applied a threshold; the output was binary, with value 1 if the sum is greater than 0, with value 0 otherwise.

That model was than improved by Frank Rosenblatt, who proposed the *Perceptron Algorithm* in 1957 [28]:

$$f(x) = \begin{cases} 1 \; if \; w * x + b > 0 \\ 0 \; otherwise \end{cases}$$

The activation function was a binary step function, so it was not-differentiable and therefore backpropagation was impossible; the update rule did not take into account any loss function or similar, indeed it was simply defined as:

$$w_i(t+1) = w_i(t) + \alpha\big(d_j - y_j(t)\big)x_{ji}$$

The perceptron was physically implemented into a machine designed for image recognition called *Mark 1 Perceptron*. Few years later, in 1960, Widrow and Hoff stacked together many single units creating the first multilayer perceptrons: *Adeline* and *Madaline*.

In 1969, Minsky and Papert wrote a book [29] in which they analysed the limits of the perceptron: they are able to solve only linearly separable problems, for example they underlined the impossibility to learn the XOR function, even though multilayer perceptron have not these limitations.

4.2.2 Backpropagation, CNNs and DBNs

Even if some progress was made during the sequent years, that book caused a general lack of interest in the research on perceptron, which last until 1986, when *backpropagation* was defined clearly [24] and became popular, greatly improving neural networks performances.

The problem concerning neural networks still was that they were not scalable: an high number of layers or an high number of neurons in a layer brought to worse performance.

LeNet-5, a *Convolutional Neural Network* for handwritten and machine-printed character recognition, was realized in 1989 [30] and improved during the following decades.

Designed specifically for computer vision tasks, convolutional neural networks have a different architecture: each neuron represented the convolution between the image of the previous layer and a filter, therefore the neurons of a layer were distributed along three dimensions and were connected to only few of neurons of the previous and of the successive layer. The learnable parameters of the network were the weights used in the convolution, which were shared between many neurons of the same layer. In this way the network resulted more independent from the size of the input image and therefore scaled better.

In 2006 *Deep Belief Networks* were presented, combining supervised and unsupervised training to achieve better results in a 2 step process [31]: firstly couples of adjacent layers were trained to reconstruct the input through Restricted Boltzman Machines, than the whole network was fine-tuned with backpropagation.

The key of the success of this process resided in the first step: the unsupervised training initialized the parameters of the network, partially preventing the sigmoid function from saturating.

4.3 Deep Networks and recent approaches

In the last years, enthusiasm for neural networks has risen once again, probably due to a great results obtained in 2010 in speech recognition [32] and another one obtained in 2012 in image classification, when the network *AlexNet* [33] won the ImageNet Large Scale Visual Recognition Challenge classification task [34].

The success of neural networks still continue, indeed since 2012 the ILSVRC classification task has always been won by neural networks, which have also been successfully used in many other computer vision task [34].

These impressive achievements have been made possible thanks to new models and techniques which have been studied in the last 5 years.

4.3.1 Rectified Linear Unit

The use of the sigmoid function has many downsides: it has two saturation zone in which the gradient is annealed, his mean is not zero and the computation of the exponential operator could be very expensive.

Further studies on biological neurons and the advantages of sparsity have led to the definition of an alternative model of the artificial neuron [35], which has been successfully tested in convolutional networks performing supervised training tasks [36].

The rectifier neuron, also called *ReLU* (*REctified Linear Unit*), uses the rectifier activation function, which is plotted in Figure 4.4 and defined as:

$$rectifier(x) = max(x, 0)$$

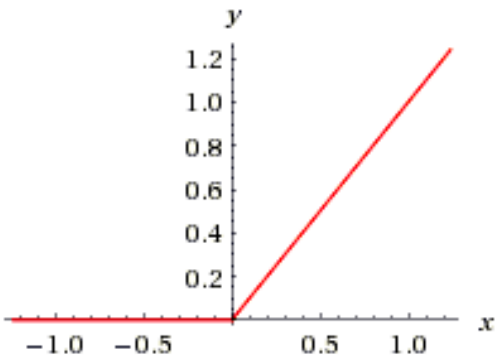

Figure 4.4 Rectifier function plotting

As shown in Figure 4.5, for any given input, only a subset of the neurons are active, which implies that a paths selection is made on the whole network and this is the only non-linearity present in the network. The computation on the subset of active neurons is linear, so gradient flow well on the active paths making mathematical investigation easier and computations cheaper [36].

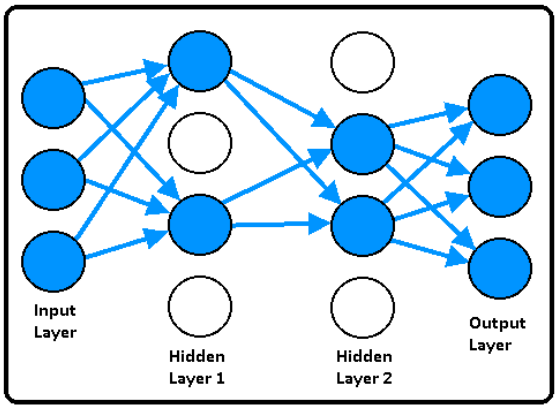

Figure 4.5 In a ReLU network the input operates a paths selection

4.3.2 Dropout

As previously said, combining different models in ensemble learning often provides better results because they generalise better, but that comes at cost: training different architectures or the same architecture on different data is very expensive, using many large network at test time could be infeasible and the dataset could be not enough big to be divided in more than one training set.

Dropout [37] is a technique presented as a solution to these problems, allowing to prevent overfitting in an efficient way: it consists in temporarily removing (dropping out) randomly chosen units of the neural network during training, along with their connections, as illustrated in Figure 4.6.

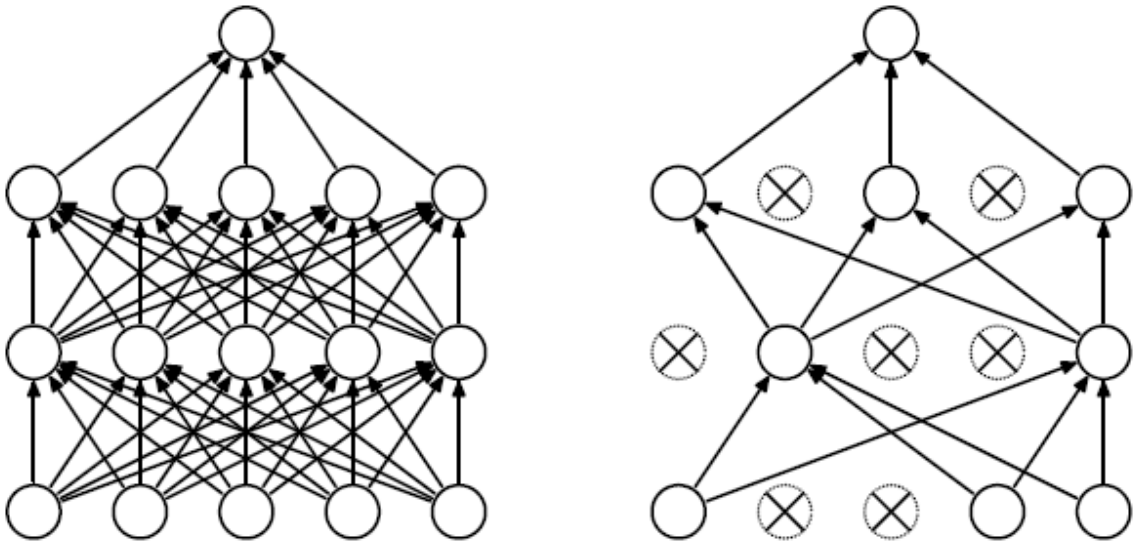

Figure 4.6 Example of a network before (left) and after (right) dropout application [37]

During training time, for each case in a mini-batch, each unit has a certain probability to be retained. During test time all the units are active, so the trained network resembles an ensemble of networks, each one with a slightly different architecture but all sharing the same parameters.

Dropout can be applied both on visible and hidden units and can be interpreted as a way to introduce noise inside the network, thus it acts as a regularizer.

4.3.3 He parameters initialization

The initialization of the parameters is a key element of the training process because it could prevent the neuron saturation and therefore improve its learning ability, so the ReLU has been investigated to define the best initialization for its parameters.

The result is that, to achieve better performance, the weights of each layer of the network should be sampled from a zero-mean Gaussian distribution whose standard deviation is $\sqrt{2/n_l}$, where n_l is the number of inputs of the l-th layer [38]; this initialization is commonly called *He initialization*, from the name of one the authors of this study.

4.3.4 Adam update function

To improve training performance, a new update method has been recently proposed: the *Adam update* consist in estimating the first and the second moments of the gradients, which are the mean and the uncentered variance,

and then computing an individual adaptive learning rate for each parameter [39].

Calling g_t the gradients calculated at timestep t, the updated parameters θ_t, are defined as follows:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \qquad \widehat{m_t} = m_t / {\beta_1}^t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * {g_t}^2 \qquad \widehat{v_t} = v_t / {\beta_2}^t$$

$$\theta_t = \theta_{t-1} - \alpha * \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

Where m_t and v_t are, respectively, the first and the second moment estimate, β_1 and β_2 are their decay rates, α is the learning rate and ϵ is a very small number.

4.3.5 Batch Normalization

During the training, the parameters of each network's layer change, resulting in the change of the distribution of the input of the following layers, producing what is called *internal covariance shift* [40]. This slows down the training because the parameters of the lower layers have to re-adjust in order to compensate for the change in the input distribution; moreover, if a saturating nonlinearity is used, this parameters change probably will move many dimensions of the input in the saturated regime of the nonlinearity, slowing down the convergence.

The use of ReLU, appropriate parameter initialization and small learning rates can partially compensate this problem, but assuring a stable distribution of the nonlinearity inputs will accelerate the training because the optimizer will be less likely to get stuck in the saturated regime. The reduction of the internal covariance shift can be achieved by *Batch Normalization* [41], a technique that introduces a normalization steps before each nonlinearity, fixing means and variances of their inputs. This also reduce the dependence of the gradient on the initial values of the parameters, allowing the use of higher learning rates.

Taking on exam a single layer, inputs are normalized computing mean and variance over each dimension, than they are scaled by two multi-dimensional parameters γ and β, that will be learned along with the other parameters of the network.

Given a *d*-dimensional input $x = \left(x^{(1)}, x^{(2)}, \dots, x^{(d)}\right)$, the output of the normalization $y = \left(y^{(1)}, y^{(2)}, \dots, y^{(d)}\right)$ is therefore defined as:

$$y^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} * \gamma^{(k)} + \beta^{(k)}$$

To conjugate this approach with stochastic optimization and therefore allowing the statistics used for normalization to participate in gradient backpropagation, a simplification is made: mean and variance are computed separately over each mini-batch.

As already said, this technique allows to use of higher learning rates, therefore a faster training, but has another convenient consequence: since each training example is seen in conjunction with other examples in the mini-batch, batch normalization regularize the model making it more general.

4.4 Going deeper: most recent architectures

If deep networks are better than shallow ones, are also deeper networks better than deep ones?

As already said, the obstacle of the exploding or vanishing gradient can be resolved with normalized parameter initialization and normalization layers, but even with these expedients networks with an higher depth are not better: when deeper networks start converging, their accuracy gets saturated and then degrades. This could appear to be caused by overfitting, but further studies have noticed that deeper networks lead also to greater training errors [42].

In the last years different structures of networks have been investigated, leading to the construction of new architectures which have been proved to be have better performance than previous architectures.

4.4.1 Residual Networks

The degradation problem has been solved with deep residual networks called *ResNets* [42]. The idea beyond them is that if some stacked layers can approximate a complicate function $H(x)$ which is the desired mapping, than they can also approximate the residual function $F(x) = H(x) - x$, the original function thus becomes $H(x) = F(x) + x$. In this way, the stacked layers approximates $F(x)$ rather than $H(x)$; these stacked layers are called residual units or residual blocks, and they are schemed in Figure 4.7.

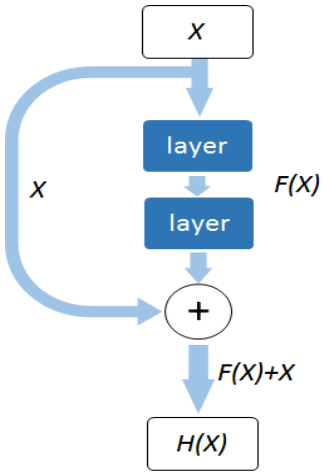

Figure 4.7 A building block of a residual network

These shortcut connections have thus been inserted every two layers of the networks, allowing the construction of good networks with more than 1000 layers. Applying this architecture, a 152 layers network has been able to won the ILSVRC 2015 classification task [34].

Further researches done very recently have led to many design experiments illustrated in Figure 4.8: a different order of the layers [43] and the introduction of dropout in the residual block [44] achieved better results on typical computer vision tasks. These architectures can be used both with convolutional and fully connected layers.

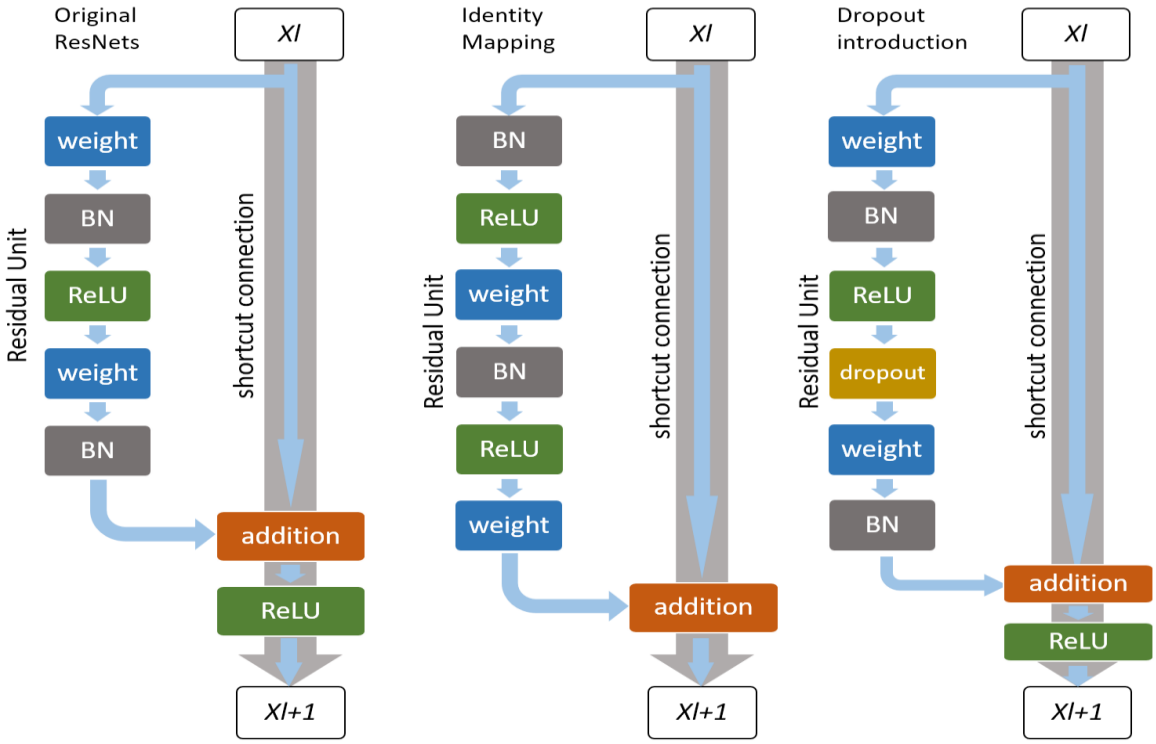

Figure 4.8 Three different versions of Residual Networks building blocks

4.4.2 Dense Networks

Following the idea of shorter paths from early layers to later ones, *DenseNets* [45] connects all layers directly to each other, concatenating rather than summing them.

In feed-forward networks the output x_l of the l^{th} layer is the result of a function H_l applied to the output of the previous layer, so $x_l = H_l(x_{l-1})$; in dense networks, the output of a layer depends from the output of all previous layers concatenated together, so $x_l = H_l([x_0, x_1, \ldots, x_{l-1}])$. The set of layers linked together in this way compose a unique *Dense Block*.

This architecture has been proposed for convolutional networks, which usually makes extended use of pooling layers, which change the size of feature-maps, but it is possible to use the concatenation operation only when the size of the feature-maps does not change, so there is need for a *Transition Layer* which do convolution and pooling between different Dense Blocks.

An example of dense network, with a focus on its different components, is illustrated in Figure 4.9.

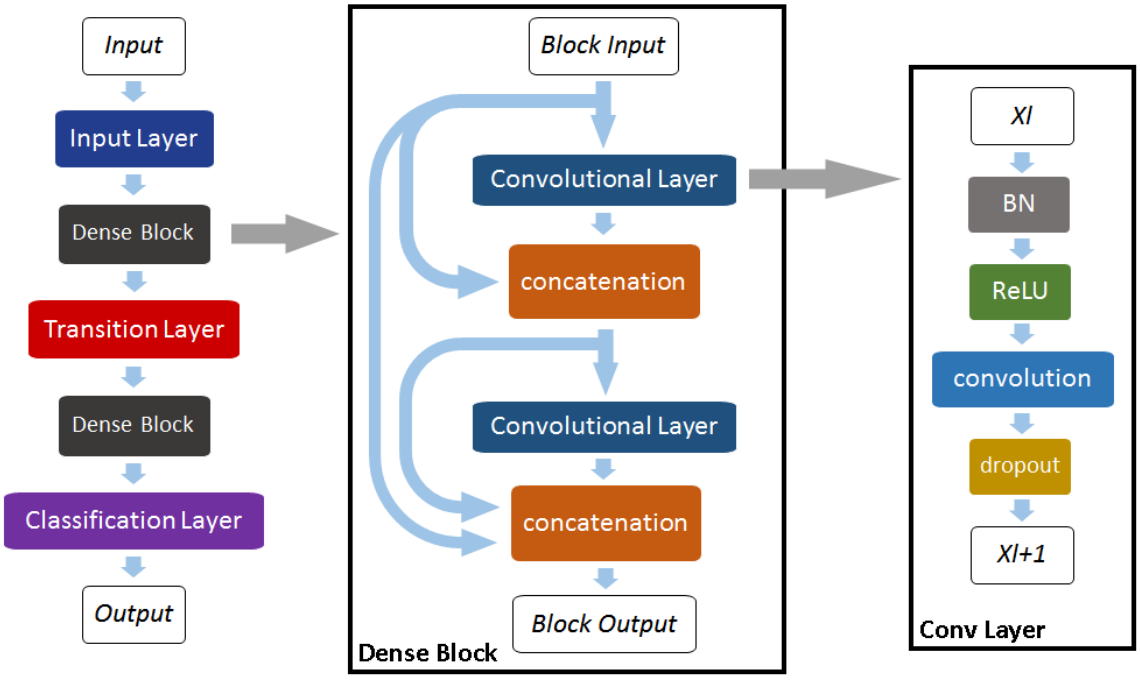

Figure 4.9 An example of dense network made by two dense blocks (on the left), an example of dense block made by two layers (in the middle) and the detailed composition of a single convolutional layer (on the right)

4.5 Neural Networks for playing board games

Neural networks have been widely applied to the solution of board games, especially to the most difficult ones, typically combined with other symbolic techniques in an hybrid system; for example using the network to numerically evaluate the game states and using that value to guide a search algorithm.

4.5.1 Evolutionary Neural Networks

Neuroevolution is a machine learning technique which has been applied frequently in works concerning games and robotics, essentially it consists into training neural networks using evolutionary algorithms [46].

Many different networks (*individuals*) are created with randomly generated parameters, then they are used to perform a task. According on the result of the task and on their properties, each network is evaluated, then it is applied what is called *selection*: the worst networks are deleted while the best ones survive. Applying little changes to the parameters or the structures of the surviving networks, new ones are created and then the process restarts, using this new set of networks (*population*) for performing the task.

It is possible to combine evolutionary neural networks with traditional space-state exploration techniques (such as min-max search) to create an AI able to play a game: a population of different AI players is created, each one use the same algorithm to explore the space state and a different neural network to perform the evaluation of the game state, than players compete against each other or against a fixed adversary and they are selected depending on the result of the game.

4.5.2 Previous works in other games

Evolutionary neural networks have been used successfully for draught in 1999 [47].

In 2015 *Giraffe* has been created, an engine which has learned to play chess throught reinforcement learning and neural networks. Starting with a knowledge base of the rules and of features to extract from the board, Giraffe explores the game state space and evaluates the states using a neural network [48].

DeepChess is a neural network created in 2016; without any previous knowledge, the network has been taught to recognize features with unsupervised pretraining, than thanks to supervised training it has learnt to

choose the better between two states. This network can be used by a state-space searching algorithm to decide which is the best possible move. It is the first end-to-end machine learning-based method that results in a grandmaster-level chess playing performance [49].

Neural networks have been used in many works concerning Othello, either using evolutionary neural networks [50] [51] [52] [53] or reinforcement learning [54] [55].

4.6 Neural Networks for the game of Go

Even though neural networks have been used to develop AIs able to play games, it is not common to train them to directly choose the best move in a game. Recently, a series of studies in which deep convolutional neural networks were trained to evaluate the best move in the game of Go have produced great results, which have culminated in a computer program defeating an international Go champion for the first time.

4.6.1 Clark and Storkey's Network

The first deep network [56] able to play the game of Go has been built in 2014 and has scored an accuracy on the test set of more than 40%, achieving good results against other popular computer Go programs.

The network takes a matrix representing the board as an input and provides the best move to make among the possible ones, excluded the choice to pass.

The input matrix contains 3 information: the position of the player's stones on the board, the position of its opponent's ones and the illegal positions. The first two information are pre-processed into three channels each, dividing the stones on the base of the liberties they have, which are the number of stones that surround them, leading to a 7 channels matrix.

The network is made by 7 convolutional layer followed by one fully connected layer and as activation function is used the rectifier. No regularization nor dropout has been used during training because overfitting was not considered a major problem.

The dataset consisted into 16.5 million of couples board-move, the 4% was used for validation and 8% for testing, dividing the training set into mini-batch of size 128. The training has lasted 9 epochs and the learning rate has been annealed only in the last 2.

4.6.2 AlphaGo

AlphaGo [22] is a program developed by Google DeepMind which has been able to achieve a winning rate higher than 99% against other Go computer programs, to defeat the European Go champion by 5 games to 0 and to defeat an international champion by 4 to 1.

The program combines both symbolic and sub-symbolic techniques, using 3 different convolutional neural networks trained in a supervised or unsupervised manner; in particular the *policy network* classifies the possible moves, predicting the best one. The network is made by 13 convolutional layers and uses the rectifier activation function.

During supervised training, the policy network has taken as input a matrix with 48 channels, containing the board state and many pre-processed features extracted from it.

The training set consisted of almost 30 million of moves, using 1 million as validation set and using one randomly sampled mini-batch for each step. The whole training has last 340 million of steps, beginning with a learning rate of 0.003 and halving it each 80 million of steps.

Capitolo 5

Neural Nine Men's Morris

Neural Nine Men's Morris, (so forth abbreviated in NNMM), is the system which has been realized as product of this thesis: it is a Python system which is able to play Nine Men's Morris using neural networks, therefore relying on sub-symbolic knowledge.

It is possible to use it in *train mode*, to train existing or new neural networks on a given dataset of "best moves"; this gathered knowledge is then exploited in *play mode*, when NNMM is capable of predict, for any given game state, which is the best move according to what it has learned.

A single network able to predict the best move in its entirety would have to choose between 1500 possible solutions: 24 possible choices for the first part of the move, 25 for the second, 25 for the third lead to $24 * 25 * 25 = 1500$ total choices. To correctly train a network to classify among so many classes would be necessary a very large training set having desired outputs well distributed among these classes, but the construction of a dataset with these characteristics is very difficult.

Therefore, NNMM architecture is made by three networks, each one dedicated to the choice of a single part of the move, so they have to classify the inputs between 24 or 25 possible classes. Since these three decisions are strictly linked, the networks operate sequentially and the decision taken by a network is given as input to the sequent ones.

The order in which the networks are stacked has been established on the basis of the frequencies of the move parts: the first decision is the TO one, which is always present, followed by the FROM one, which is present in phase 2 and 3, while the REMOVE one is the last.

So the working principle of the designed system is the following:

1) The board state is read and pre-processed
2) The first network predicts the best place where to put a stone.

3) The second network predicts which is the stone that should be moved (if it is necessary)
4) Finally, the third network predicts if an adversary stone must be removed and chooses which one

The final output of the system will therefore be the complete move, made by the aggregation of the three parts.

Even though it is possible to represent the game state as a bi-dimensional image, the Fully Connected Network model has been chosen instead of the Convolutional Network one; this decision has been taken considering the available hardware resources and the problem characteristics, in particular for the differences between logical distance and "physical" distance of two board positions.

5.1 Networks I/O model

The output of each network is a probability distribution between 24 or 25 numerical classes, respectively for the first network and the other two. These classes represent all the 24 positions on the board and, for the FROM and REMOVE networks, the possibility to do not indicate a position (class 0). The class with the highest probability is considered the position chosen by the network as the best one.

The array representation of the board, presented in 3.2.2, has been chosen as input for the networks, so the complete input to each network will be an array representing the game state and the already made choices.

5.1.1 Binary raw representation

The first board representation is a binary array of size 118, following the representation indicated in Table 5.1

This representation can be addressed as the "*raw"* representation of the game, because there are simply the information that a human player is aware of, without any additionally computed feature.

The choices made by previous networks are represented as array of size 24 or 25, with value 1 if the bit indicates the position with the highest probability, 0 otherwise.

Bits	Meaning
24	For each position, the presence (1) or the absence (0) of a checker of the player
24	Same purpose as the previous ones, but for the adversary checkers
24	Indicates if a position is empty (1) or occupied (0)
9	Number of checkers in player's hand, as the number of consecutive bits with value 1 Example: 6 becomes 111111000, 3 becomes 111000000
9	Same as previous ones, but for adversary checkers
9	Number of players' checkers on board, as the number of consecutive bits with value 1
9	Same as previous ones, but for adversary checkers
3	Phase of game of the player, as the number of consecutive bits with value 1
3	Same as previous ones, but for adversary

Table 5.1 Binary raw representation of the input

5.1.2 Integer raw representation

This representation is similar to the previous one but use integer numbers instead of binary ones, resulting in an array of size 30, as indicated in Table 5.2.

Numbers	Meaning
24	For each position, the presence of a checker of the player (+1), of his adversary (-1) or none of it (0)
2	Number of checkers in the hands of the player and its adversary
2	Number of checker on the board: the player's ones and its adversary ones.
3	Phase of game of the player and of its adversary

Table 5.2 Integer raw representation of the input

The choices made by previous networks are represented as two integers between 1 and 24 or between 0 and 24, that indicate the chosen position.

5.2 Networks architecture

Taking into account the most recent studies, described in 4.3 and 4.4, three different models of neural network have been realized, each one parametric in almost any of its aspects, to allow a better investigation of it.

The networks are made by fully connected layer of neurons and the rectifier function has been chosen as the activation function, therefore has been decided to initialize the weights with the He method.

In each model it is possible to apply a dropout that turns-off input neurons with probability pi; furthermore, before and after the application of the non-linearity, it is possible to apply, respectively, batch normalization and a dropout with probability p.

The last layer of the networks is a fully connected layer with 24 or 25 neurons (the number depends by the network purpose) to which the softmax function is applied.

5.2.1 Residual network

The first architecture is a residual network: the network has a fully connected layer with a parametric number of units, followed by a parametric amount of residual units.

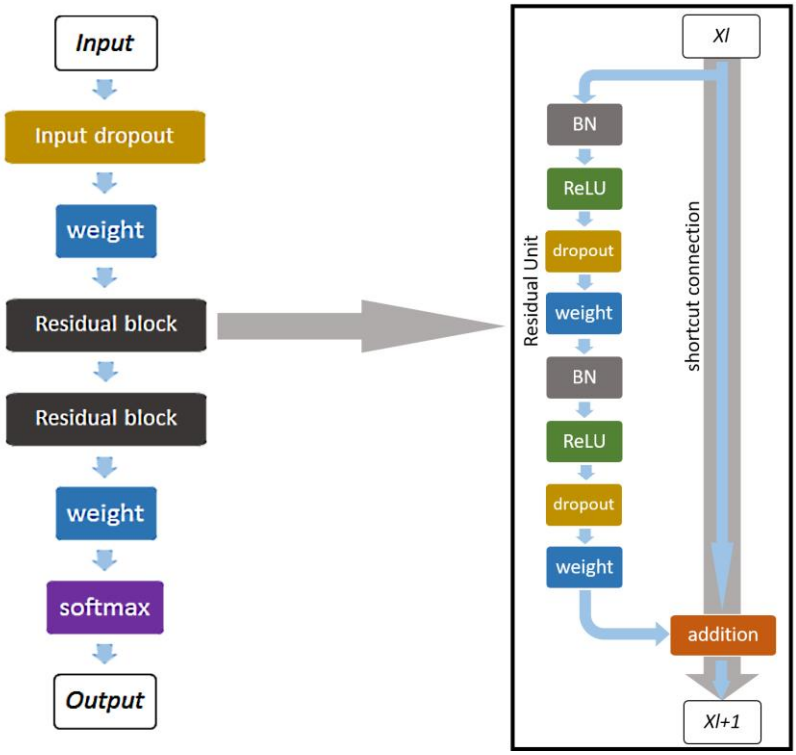

Figure 5.1 An example of a Residual Network with two blocks as implemented in the system, with a focus on the single building block (on the right)

Considering all the improvements proposed for the residual network, the residual units have been realized as pictured in Figure 5.1. Each unit is composed by two layers with a parametric number of neurons and each layer is made by 4 operations: batch normalization, application of the rectifier

function, dropout and weighted sum, so an entire residual unit is made by 8 operations.

5.2.2 Dense network

The second architecture is a dense network with a single dense block, illustrated in Figure 5.2. Using fully connected layers instead of convolutional ones has made possible to construct the network without transition layers, because there is no more the need of pooling operations.

The architecture of a single layer has been slightly modified, as represented in Figure 5.2: the operation applied to the layer input are, in order, batch normalization, rectity function, dropout and weighted sum.

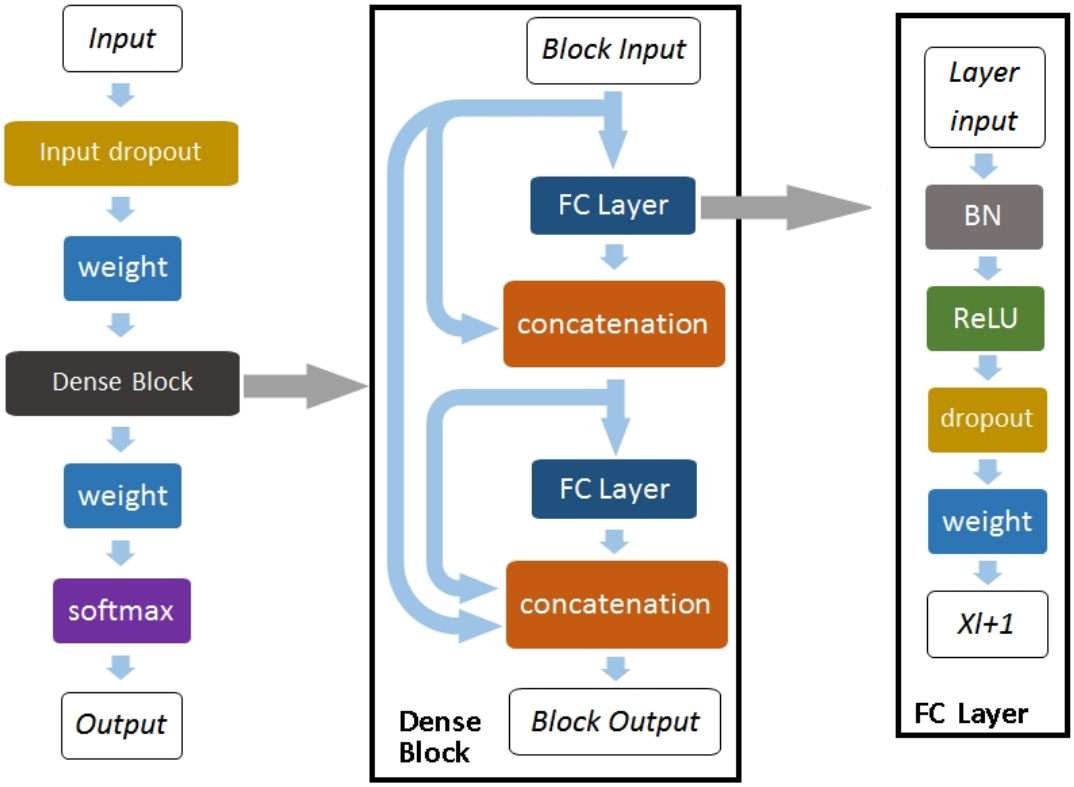

Figure 5.2 An example of dense network as implemented (left), with a focus on an example dense block made by two layers (middle) and a focus on the composition of a single fully-connected layer (right)

5.2.1 Feed-forward network

The last and simplest model is a feed-forward neural network, in which every layer is connected to the previous and the subsequent ones.

The network is made by a parametric number of fully connected layers, each one composed by a parametric number of neurons, in which the following

sequence of operators is applied: weighted sum, batch normalization, rectify function and finally dropout.

5.3 Dataset

The dataset [57] used to train the networks is a text file made by 100154 strings which represents game states and corresponding suggested moves. The application of all the possible symmetries to all the data greatly increase the size of the dataset, which reach the size of 1628673 unique pairs.

5.3.1 Dataset creation

As part of the course of Foundations of Artificial Intelligence M, the instructors Prof. Paola Mello and Ing. Federico Chesani have proposed to the students the challenge to develop a software capable of playing Nine Men's Morris using AI techniques [58] [59]. Working in small teams, students have developed 17 different programs which have played against each other both as white player and black player.

The AI which has obtained the best results is DeepMill [60] (so forth abbreviated in DM), a Java application that exploits a NegaScout search, increasing the depth of the search over time until a timeout occurs. Therefore, it can be said that keeping fixed the available hardware and time resources, its choices are deterministic. On 32 matches played against 16 adversaries, it has won 30 times and the remaining 2 times the game has ended in a draw.

Since this program follows a typical symbolic approach and has proved itself to be the best AI between the ones developed, it has been chosen as teacher for the neural network.

Therefore the dataset is composed by the pairs of game states occurred during matches of Nine Men's Morris and the choices made by DM with a time limit of about 60 seconds. To generalize better, the pairs do not consider the players as black or white but consider them as "me" and "adversary".

Between these pairs, all the ones where the state can be obtained from another dataset state (applying one of the symmetries previously described in 3.2.1) have been excluded.

Data have been gathered from 7244 matches of three different classes:

- Firstly DM has played a full game against all the other students' AI (including itself), both as white and as black player; this has brought to a collection of 491 data.

- Then, to compensate the fact that in previous games DM has never reached disadvantaged positions, games with custom start have been designed: these matches started with reachable configurations where one player has an advantage over the other one. Once again, DM has played against all the other AI, both as white and black player, allowing the gather of 1197 data.
- To greatly increase the number of training examples, DM has disputed a series of game against itself and other players where the starting configuration has been randomly generated. It is important to point out that even if some arrangements have been made to generate realistic starting configurations, these initial states could be not reachable in a normal game. However, this does not weaken the usefulness of the dataset as a mean to train a neural network to recognize useful features, learn to play following the rules of the game and to choose the best move according to DM.

5.3.2 Dataset composition

An entry of the dataset consist of a string of 31 to 35 characters:

- The first 24 characters describe the board state with a letter representing the state of each position: O if the position is empty, M (Mine) if there is a checker of the player and E (Enemy) if there is an opponent one.
- A sequence of 4 numbers completes the state representation, where the first two numbers represent, respectively, the number of checkers that the player has in its hands and the ones that his adversary has; the last two represent, in the same order, the number of checker that the players have on the board.
- An hyphen divides the game state from the move description, which is written as pairs of coordinates letter-number; the meaning of each coordinate depends on the game phase: the parts of the move are written (if present) in the order FROM, TO and REMOVE.

Example of dataset entries and corresponding states and moves are illustrated in Figure 5.3.

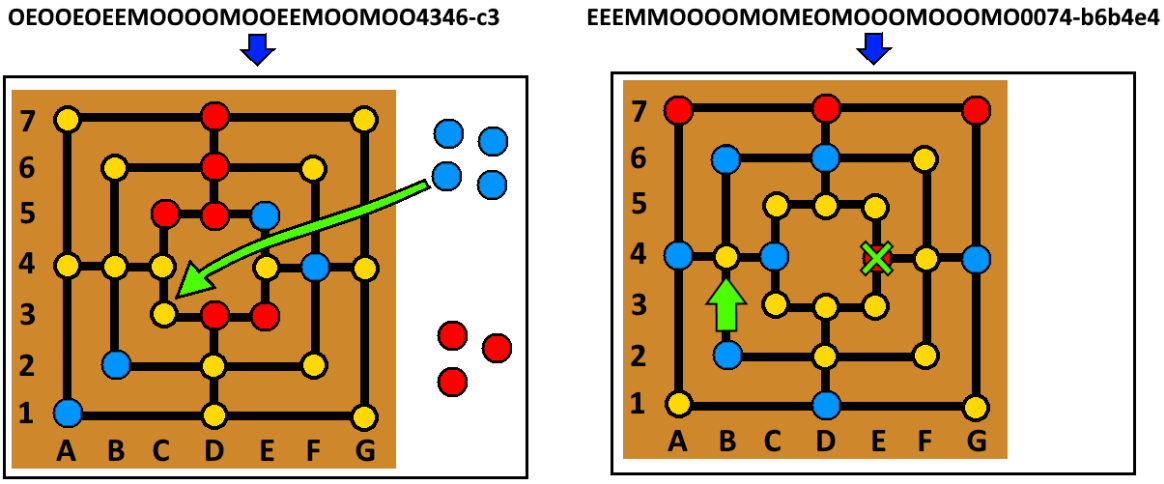

Figure 5.3 Two example of dataset entries and their corresponding meaning as a game state and a move (in green). The player's stones are coloured in blue, while its adversary's stones are coloured in red.

As can be seen in Figure 5.4, the single decisions of the expanded dataset are almost equally distributed among symmetric positions; it is important to underline that the FROM part of the move is absent in nearly 20% of the cases, while the REMOVE part is absent in more than the 70%, making the 0 class by far the most present.

Calculating the entropy of the three choices, the TO one results to be the most uniformly distributed with an entropy of 4.53, followed tightly by the FROM one with 4.41, while the REMOVE decision has an entropy of barely 2.00.

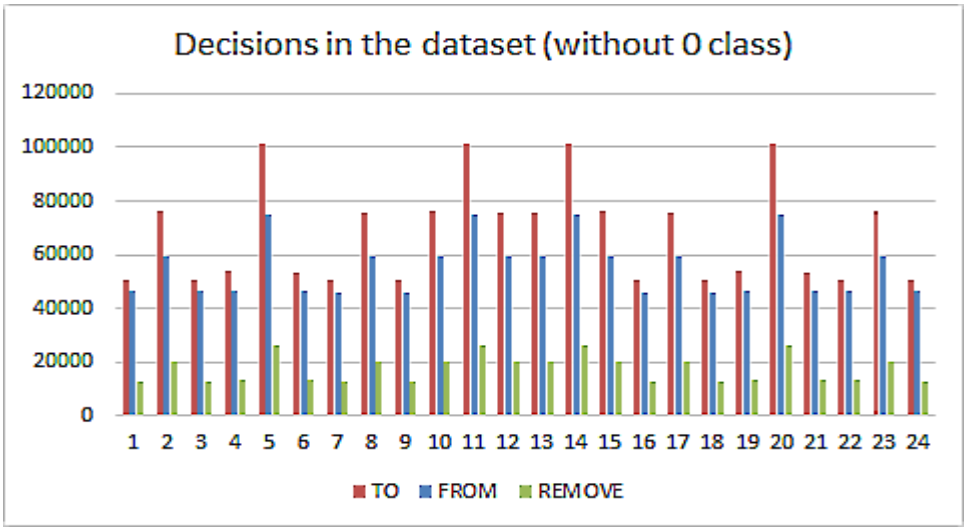

Figure 5.4 Distribution of the decisions in the expanded dataset among the different classes, without considering the 0 class

Another important semantic consideration about the dataset is that is made mostly by examples extracted from phase 2 of game, as can be seen in Figure

5.5, while phase 1 and 3 are almost equally present. This is perfectly reasonable because usually the second phase of the game is the longest one, therefore the one which provides an higher amount of pairs. The different distribution of examples among the three phases could influence the training, making the system better at playing in phase 2 than in the other phases.

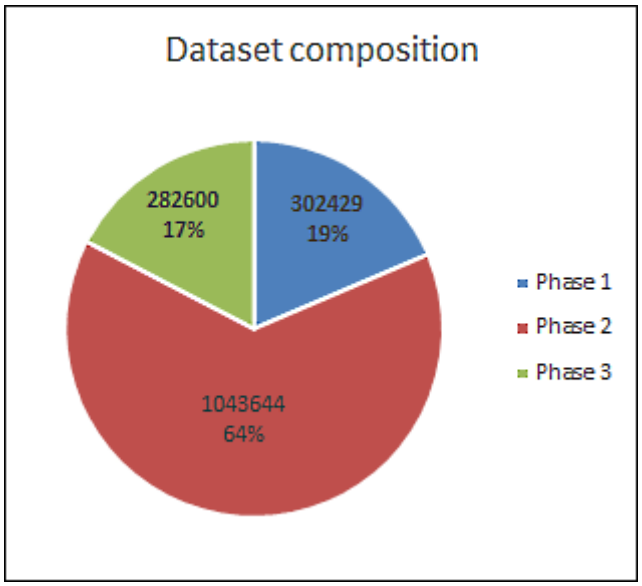

Figure 5.5 Composition of the expanded dataset among the different phases of the game

5.4 Networks Training

The training of each network is independent from the training of the other ones, but all of them are based on the same dataset of pairs $[game\ state, best\ move]$ and exploit the symmetries of the problem to increase the number of examples.

The training process of a network follows the sequent workflow:

1) The dataset is read
2) Each entry of the dataset is expanded creating all the different symmetric entries
3) The entries are pre-processed into the chosen I/O model
4) The network is trained

The use of the data is different among the networks:

- The first network is trained using the board state as input and the choice "TO" of the entry as desired output.

- The second network is trained using both the board state and the choice "TO" of the entry as input, while the choice "FROM" of the entry is used as target.
- Similarly to the second network, the training of the third one uses the board state and the choices "TO" and "FROM" of the entry as input, whereas the target is the "REMOVE" choice.

The loss function chosen for the training is the negative log-likelihood of the target class, to which a L1 or a L2 regularization can be optionally added; the presence of the regularization, its type and its weight are training parameters.

As update function has been chosen the Adam update, so three more training parameters are the learning rate α_0 and the decay rates $b1$ and $b2$. The initial learning rate is progressively annealed through epochs with proportional decay, guided by the parameter k.

The training follows the mini-batch optimization, but since the size of the batch is a parameter, batch optimization or stochastic optimization could be done too.

Even if it has been mentioned as an architecture element, is important to underline that the dropout is a training technique, so the probability *pi* and *p* are other two training parameters.

The early stopping technique is adopted: the network is saved each time it improves its accuracy on the validation set; if the accuracy does not improve for a parametric number *patience* of training epochs, the training is stopped.

5.5 Play mode

Once that three neural networks have been created and trained, Neural Nine Men's Morris can use them to play.

Using socket connection, any program can send a game state to NNMM (in the same format described previously as a dataset entry) and receive the best move according to the prediction done by the networks.

As presented in Figure 5.6, once received, the game state is fed to the first network, which elaborates the best TO decision; the following networks elaborates, in order, the FROM decision and the REMOVE decision, receiving as input both the game state and the previously made decisions.

The legality of each decision and therefore of the entire move is guaranteed by the *legality check*.

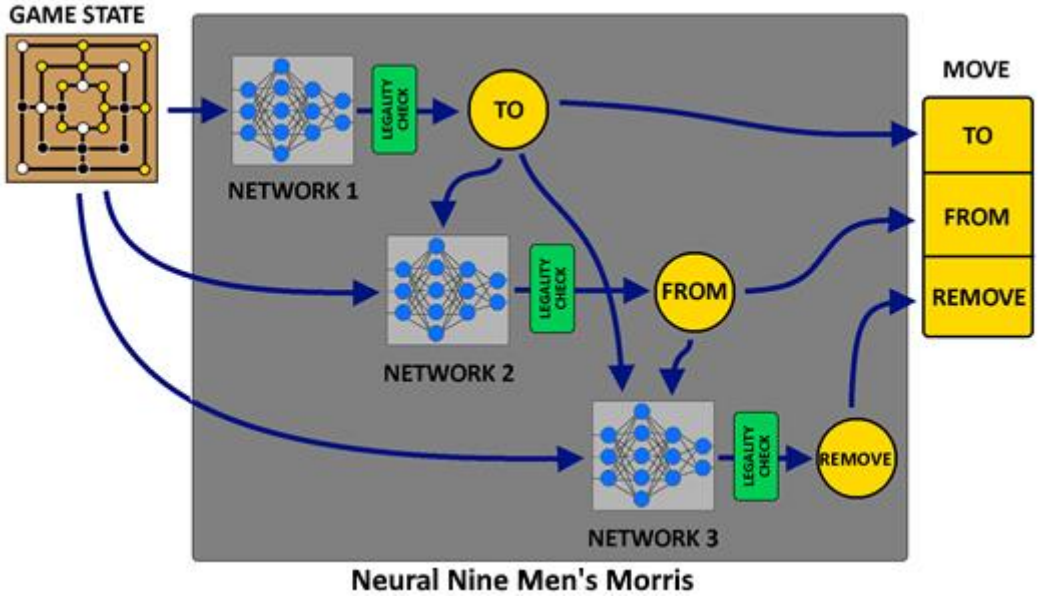

Figure 5.6 System gaming mode workflow

5.5.1 Legality check

The legality check is the only symbolic element present in the system and verifies that the suggested decisions satisfy the game rules.

Since the output of each network is the ranked list of alternatives, if the best one is not legal, the system considers and tests the second one and so on until it finds a legal decision, which will be considered the best one.

The presence of this symbolic check is necessary because is not possible to guarantee that a system which relies only on sub-symbolic knowledge will be able to follow the game rules.

Nonetheless, with a proper training, the legality check could became unnecessary: a very important part of the testing of the system will be to verify how many times it is able to make a completely legal move; in other words, verify if NNMM has learned to play following the rules without the help of any symbolic knowledge.

5.6 Implementation

Neural Nine Men's Morris has been implemented in Python programming language [61], in particular the networks have been realized using the

Lasagne [62] and Theano [63] libraries. For testing purpose, a Java player has been also created, which operates as an interface between a Java game engine and the system.

The file *networks.py* contains the functions for the creation of the networks described previously and also the persistence functions for saving and loading both networks structure and parameters.

The functions for the training of a neural network are contained on the *training.py* file, allowing both to create a new one or load an existing one. The functions allow to specify all the parameters of the network and of the training.

All the functions for the communication with other programs are written in the file *connection.py*

5.6.1 Data Processing

The file *dataprocessing.py* contains the functions for the loading of the data and their pre-processing.

The dataset loading function process each data into a state object and a numeric tuple that represents the corresponding move; each couple is then expanded in all the possible symmetries.

The other functions of the module allow to convert the pairs state-move into an array, following the representations described in 5.1.1; it is possible to extend the module to operate on other representation.

5.6.2 Legality

The file *legality.py* contains the functions for the validation of the network choices. The specific implementation of the functions is made to operate on the binary raw representation of the states, but is possible to realize equivalent functions for the other representations.

The self-legality tests verify if a single choice is legal according to the game state:

- The chosen TO position must be empty and, in phase 2, in one of the adjacent positions there must be a stone of the player

- The chosen FROM position must be a position different from 0 only in phase 2 and 3, in this case it must be occupied by a player's stone.
- The chosen REMOVE position, if it is different from 0, must be occupied by an adversarial stone; in addition, if the chosen stone is aligned in a mill, all the other adversary's stones must be aligned.

The other legality tests verify if a choice is legal according to the previous taken decisions:

- In phase 2, the chosen FROM position must be adjacent to the chosen TO position.
- The REMOVE position must be different from 0 if and only if the other two choices realize the closing of a mill.

Capitolo 6

Experimental Results

To test the playing ability of NNMM at its best, the first problem has been to find a proper configuration to train different network architectures.

A hyper-parameter space with high dimensionality, such as the one that defines the possibilities of architectures and training in NNMM, makes the problem to find the optimal configuration extremely difficult.

Firstly, a wide tuning on a restricted dataset has been done, in order to find possible good configurations in a smaller amount of time. Then few configurations have been tested on the whole dataset.

Once found a good configuration, to verify if the proposed system could be as good as its teacher, it has been tested in terms of accuracy of the prediction, legality of the prediction and goodness at playing.

6.1 Tuning on restricted dataset

To initially calibrate the hyper-parameters of the network, tests about the TO decision have been done using a partial dataset of 30,469 pairs expanded into 49,1970 taking symmetries into account. The 5% of this dataset, therefore 24,598 pairs, have been used as validation set, while the remaining has been used as training set.

The decision to investigate the performance of the TO network, instead of the FROM or the REMOVE one, comes from the distribution of data among the classes which is more uniform in the TO decision.

Of more than 100 tests, only the results of some interesting comparisons have been reported here.

6.1.1 FFNets testing

Figure 6.1 and Figure 6.2 report the outcomes of 4 brief trainings of 100 epochs on a 4 layers feed-forward neural network with different input dropout probability. Applying dropout over the input slows the training both

as time needed for the computation and as number of training epochs necessary for achieving a certain accuracy, but on the other hand using a low dropout can lead to a better final result in an acceptable amount of time.

The parameters that have been kept fixed are presented in Table 6.1.

Network Structure	4 layers: 200, 200, 100, 50 neurons
Batch normalization	Yes
Dropout	$p = 0\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001$, $b1 = 0.99$, $b2 = 0.999$, $k = 0.2$
Batch size	1000

Table 6.1 Configuration of FFNets trainings on partial dataset for dropout testing

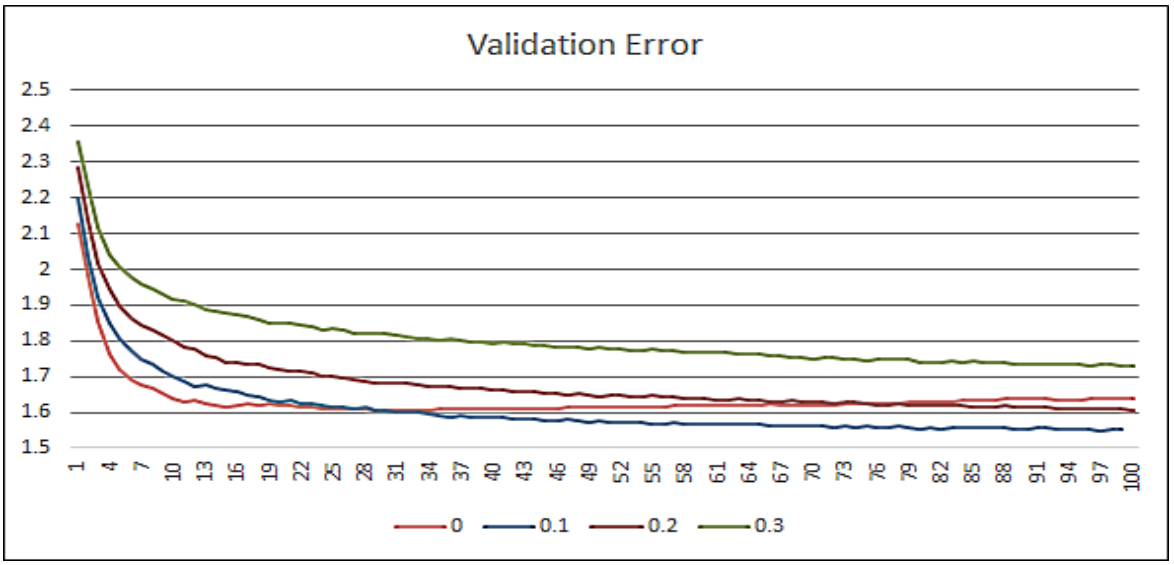

Figure 6.1 Validation error for different percentages of input dropout in FFnets

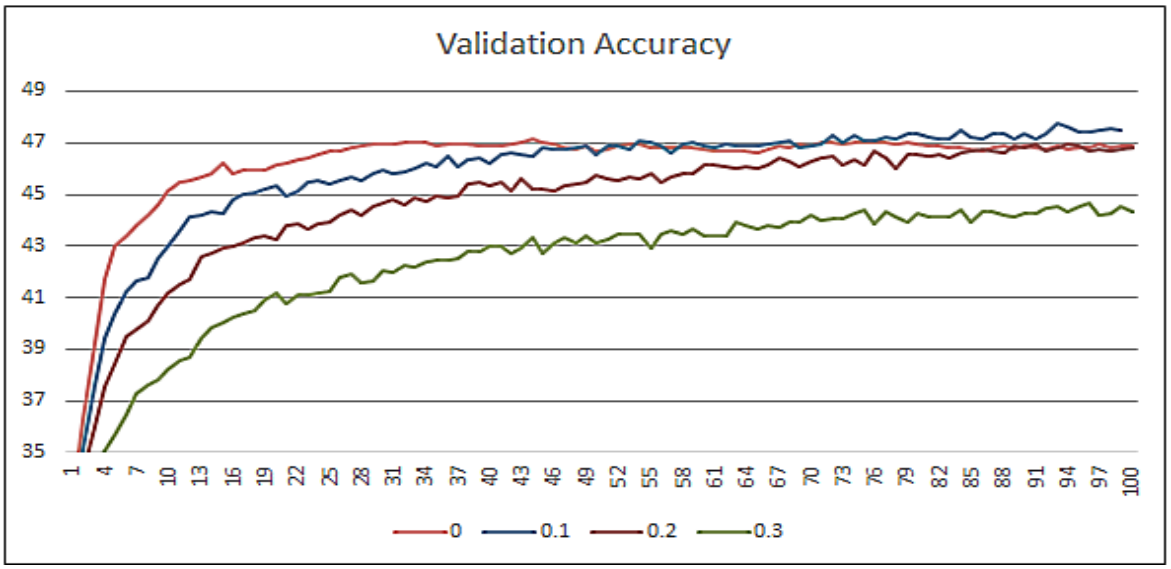

Figure 6.2 Validation accuracy for different percentages of input dropout in FFnets

Figure 6.3 and Figure 6.4 show how the network structure influences the performance: after 9 brief trainings of 100 epochs, it has been observed that feed-forward networks with too many neurons lead to worse performances, probably due to overfitting.

The parameters that have been kept fixed are presented in Table 6.2.

Network Structure	4 layers
Batch normalization	Yes
Dropout	$p = 0\%, pi = 0\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.2$
Batch size	1000

Table 6.2 Configuration of FFNets trainings on partial dataset for layers width testing

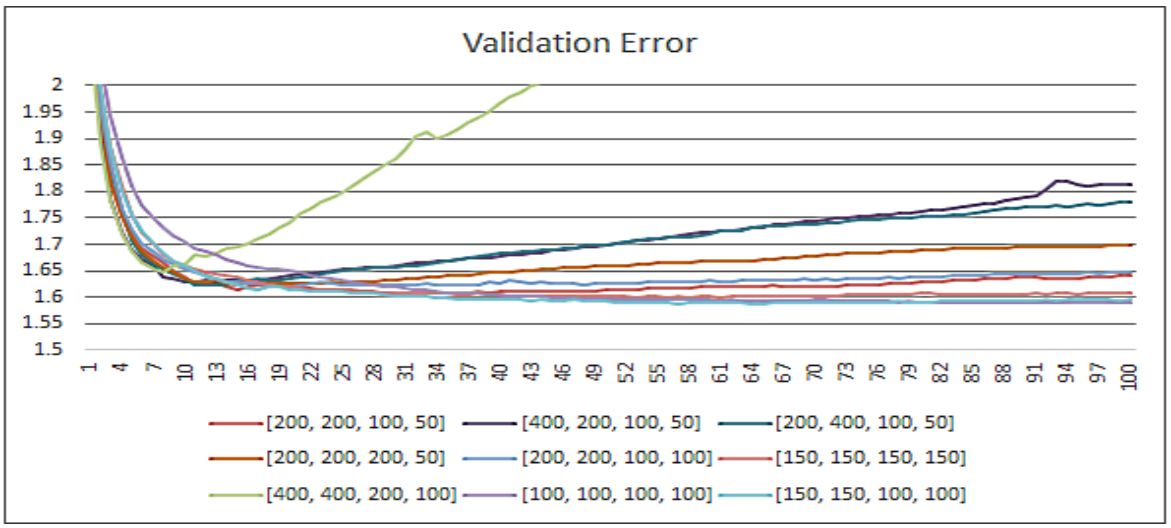

Figure 6.3 Validation error for different number of neurons in FFnets

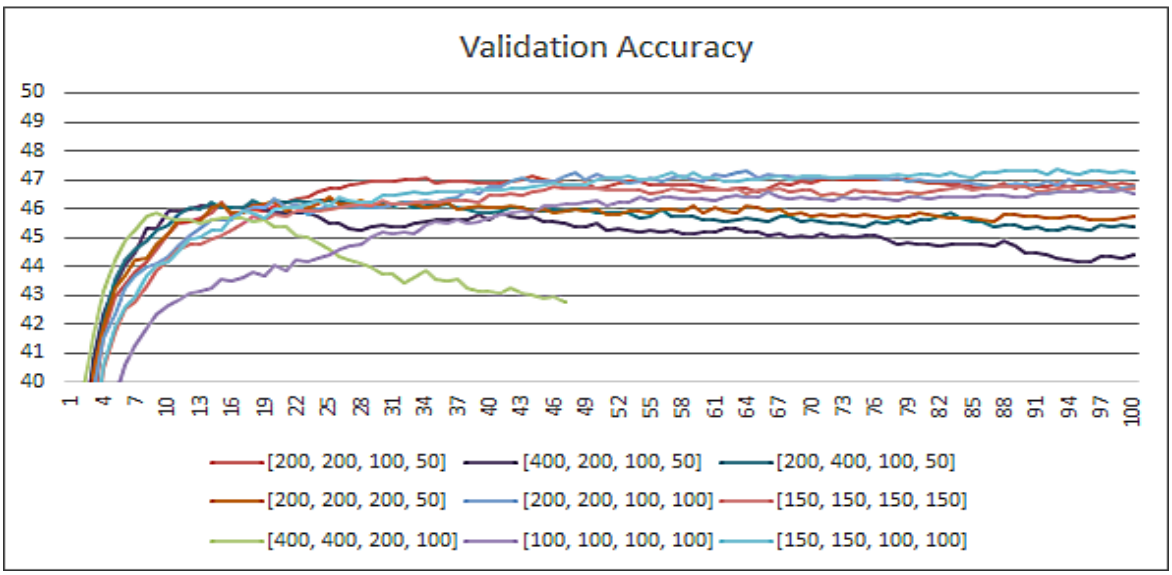

Figure 6.4 Validation accuracy for different number of neurons in FFnets

6.1.2 ResNets testing

Figure 6.5 and Figure 6.6 report the impact of dropout application in residual networks. After 10 trainings of 500 epochs, the application of high percentages of dropout has demonstrated itself worse than low percentages, both in the first layer and in the following ones.

The parameters that have been kept fixed are presented in Table 6.3.

Network Structure	1° layer 100 neurons, 3 blocks of 200 neurons layers
Batch normalization	Yes
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.2$
Batch size	2000

Table 6.3 Configuration of ResNets trainings on partial dataset for dropout testing

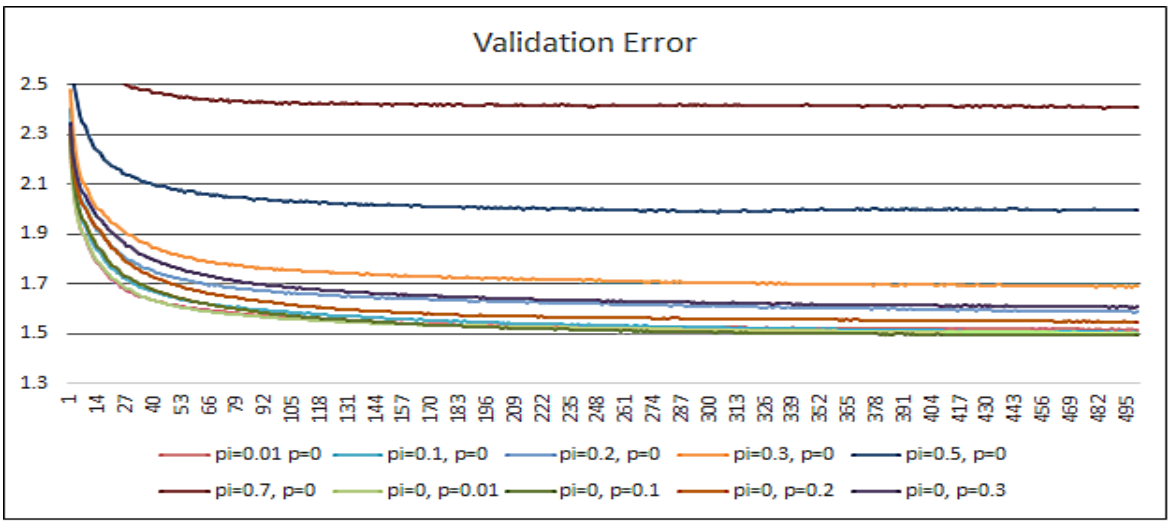

Figure 6.5 Validation error for different percentages of dropout in ResNets

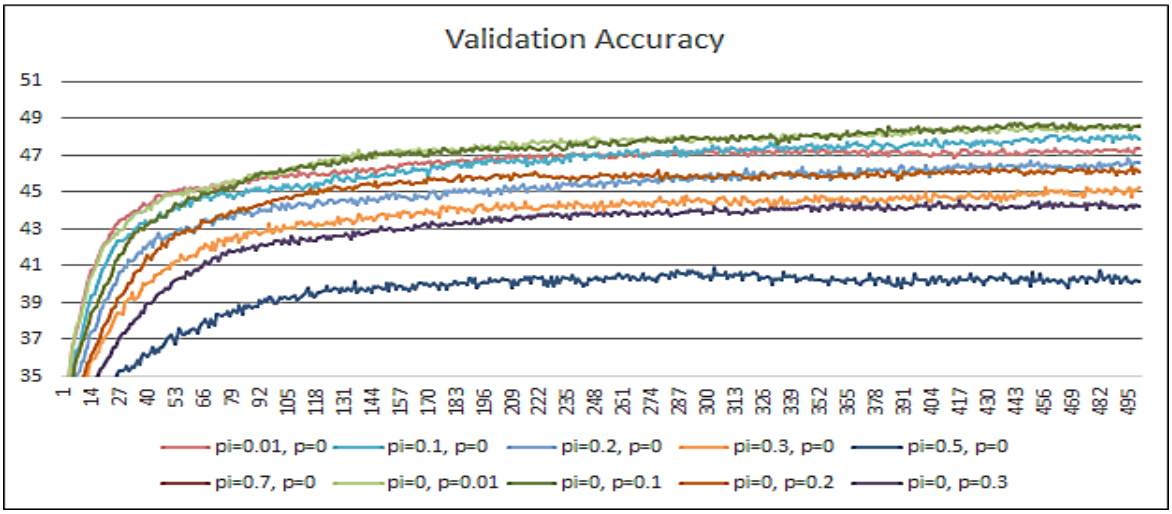

Figure 6.6 Validation accuracy for different percentages of dropout in ResNets

Figure 6.7 and Figure 6.8 show how the depth of the network influences the performance. All the trainings have been done over 500 epochs and despite the great difference between the number of residual blocks of the networks, the highest accuracy reached is very similar.

The parameters that have been kept fixed are presented in Table 6.4.

Network Structure	1° layer 200 neurons, blocks of 300 neurons layers
Batch normalization	No
Dropout	$p = 10\%, pi = 0\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$
Batch size	2000

Table 6.4 Configuration of FFNets trainings on partial dataset for depth testing

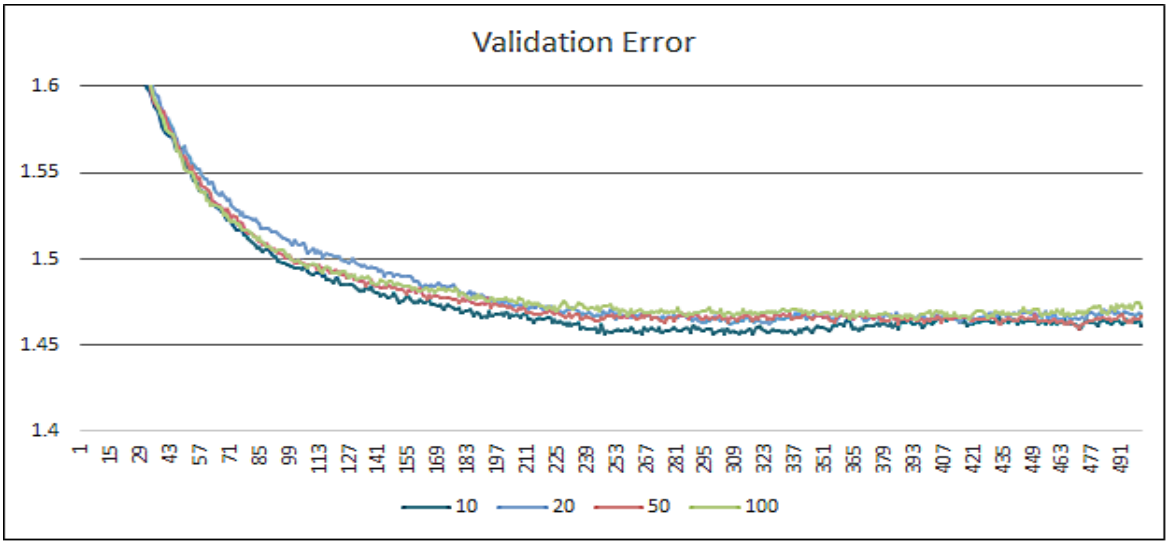

Figure 6.7 Validation error for different number of residual blocks in ResNets

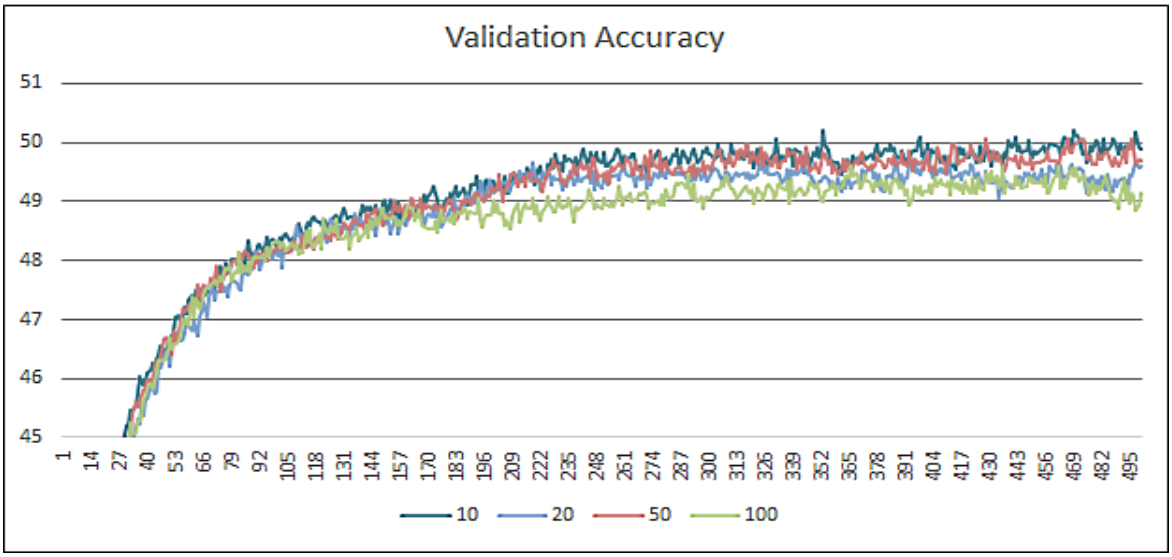

Figure 6.8 Validation accuracy for different number of residual blocks in ResNets

6.1.3 Comparison between the architectures

To decide which architecture could be the best for this case of study, the best results obtained for each one of the three architecture have been compared, as it can be seen in Figure 6.9 and Figure 6.10.

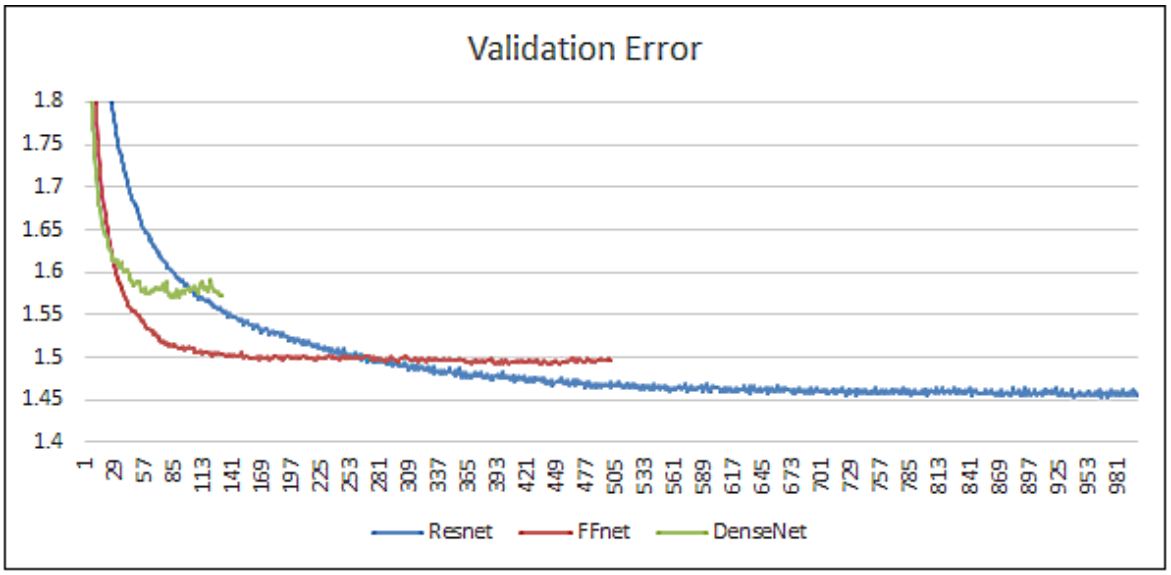

Figure 6.9 Validation error for the best TO network for of each architecture

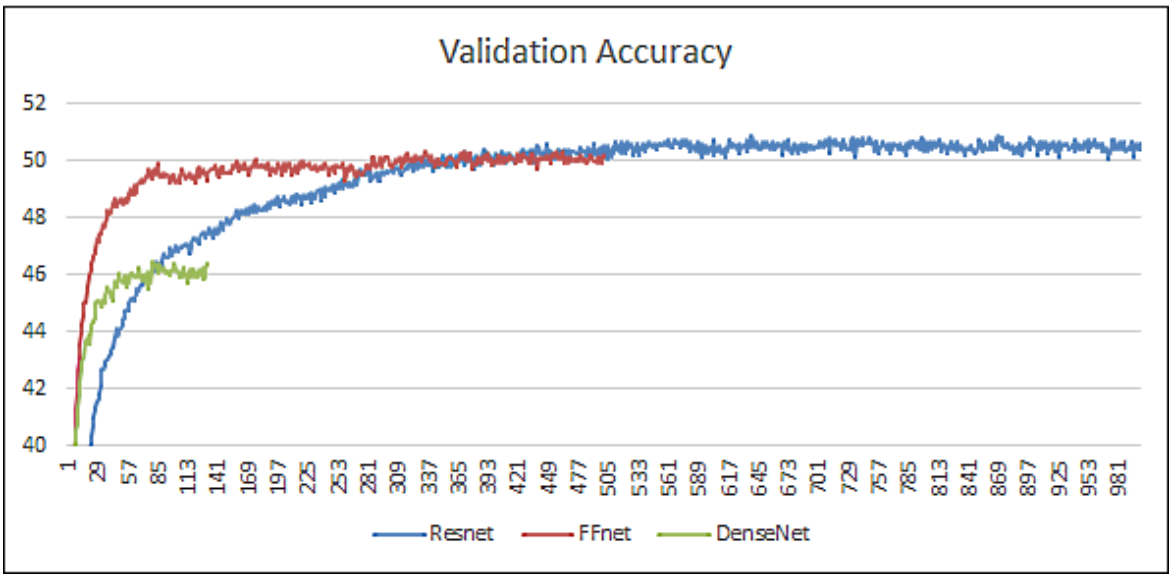

Figure 6.10 Validation accuracy for the best TO network for each architecture

Residual networks and Feed-Forward networks provide similar results (respectively 50.91% and 50.37% of top validation accuracy), which are better than the ones achieved by Dense networks (46.71%). To choose the architecture to be considered for the tuning on the whole dataset, the time needed to reach the best accuracy has been taken into account: the FFNet has

reached the best accuracy in about 48 hours, while the ResNet has reached it in less than 20 hours.

The parameters for the three network trainings are presented in Table 6.5, Table 6.6 and Table 6.7.

Architecture	Residual Network
Network Structure	1° layer: 200 neurons, 10 blocks of 300 neurons layers
Batch normalization	No
Dropout	$p = 10\%, pi = 10\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$
Batch size	5000

Table 6.5 Configuration of the best ResNet training on the partial dataset

Architecture	Feed Forward Network
Network Structure	4 layers of 500 neurons
Batch normalization	Yes
Dropout	$p = 10\%, pi = 10\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$
Batch size	5000

Table 6.6 Configuration of the best FFNet training on the partial dataset

Architecture	Dense Network
Network Structure	1° layer: 200 neurons, 3 layers of 100 neurons
Batch normalization	No
Dropout	$p = 0\%, pi = 0\%$
Regularization	L1 with weight 0.001
Update parameters	$\alpha_0 = 0.002, b1 = 0.99, b2 = 0.999, k = 0.1$
Batch size	2000

Table 6.7 Configuration of the best DenseNet training on the partial dataset

6.2 Tuning on whole dataset

For the tuning on the whole dataset, few configurations have been experimented using the 5% of the expanded dataset, therefore 81,433 pairs, as validation set.

The experiments have focused on different depths and different regularizers and have been done with a patience of 50 epochs.

6.2.1 TO network tuning

Figure 6.11 and Figure 6.12 shows the results of the TO network tuning. The highest validation accuracy reached in the 4 trainings differs for less than 1% and the best one (53.98%) is slightly better than the best validation accuracy obtained on the partial dataset.

The experiment which has provided the best outcome is the first, with a top validation accuracy of 53.98%, which results to be the longest experiment too, with about 18 hours needed to reach it as presented in Figure 6.13. The parameters used in the trainings are presented in Table 6.8.

Architecture	Residual Network
Network Structure	
1, 2, 3	1° layer: 200 neurons, 10 blocks of 300 neurons layers
4	1° layer: 200 neurons, 30 blocks of 300 neurons layers
Batch normalization	
1, 2, 4	No
3	Yes
Dropout	
1, 3	$p = 10\%, pi = 10\%$
2, 4	$p = 0\%, pi = 0\%$
Regularization	
1, 2, 4	L1 with weight 0.001
3	L1 with weight 0.0001
Update parameters	
1, 2, 3	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$
4	$\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.2$
Batch size	20000

Table 6.8 Configurations of TO trainings of the networks on whole dataset

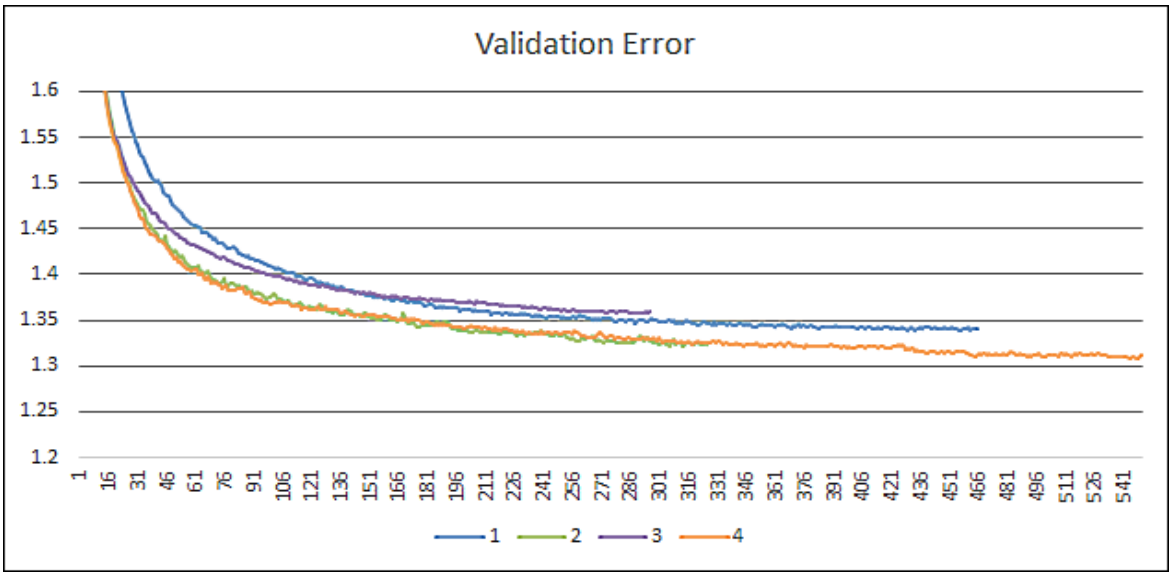

Figure 6.11 Validation error for the TO networks

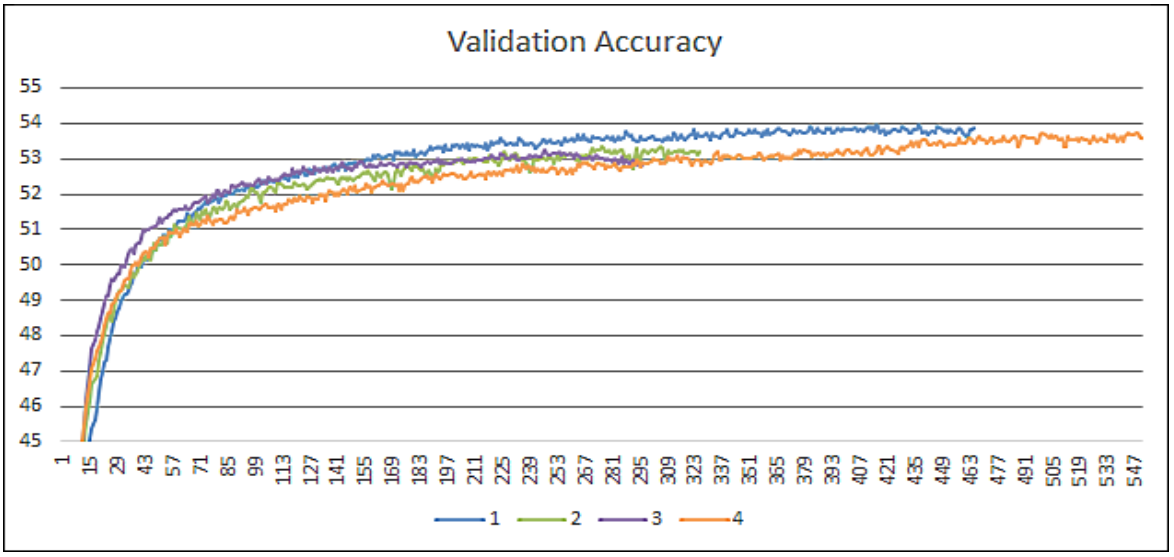

Figure 6.12 Validation accuracy for the TO networks

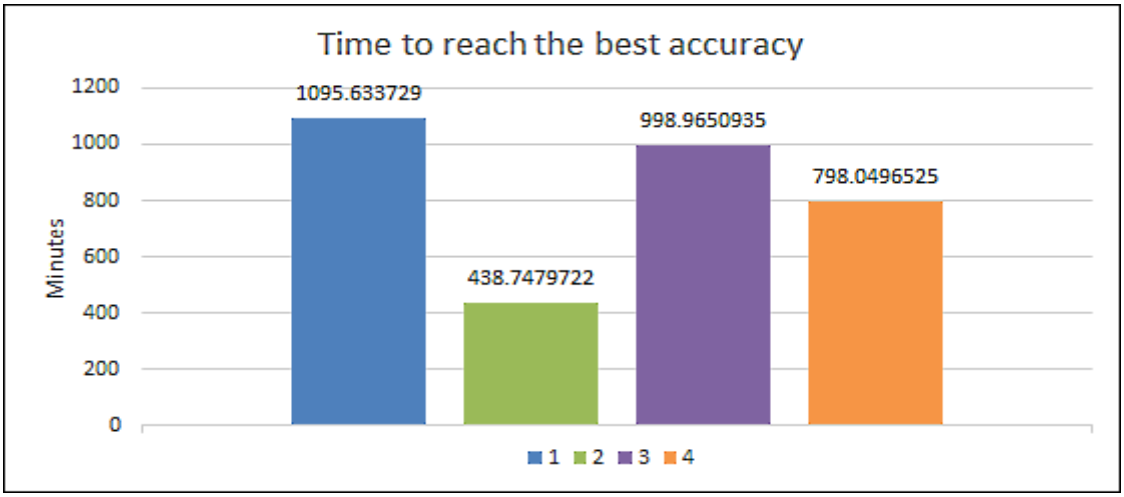

Figure 6.13 Time needed to reach the best accuracy for the TO networks

6.2.2 FROM network tuning

In Figure 6.14 and Figure 6.15 the trainings of the FROM networks are presented, which have all reached the similar result of about 89% of top validation accuracy. Once again the first configuration has proven to be the best with an outcome of 89.53% and has been the fastest, requiring about 3 hours, as showed in Figure 6.16. The parameters used are presented in Table 6.9.

Architecture	Residual Network
Network Structure 1, 2 3	 1° layer: 200 neurons, 10 blocks of 300 neurons layers 1° layer: 200 neurons, 30 blocks of 300 neurons layers
Batch normalization 1, 3 2	 No Yes
Dropout 1, 2 3	 $p = 10\%, pi = 10\%$ $p = 0\%, pi = 0\%$
Regularization	L1 with weight 0.001
Update parameters 1, 2 3	 $\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$ $\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.2$
Batch size	20000

Table 6.9 Configurations of FROM trainings of the networks on whole dataset

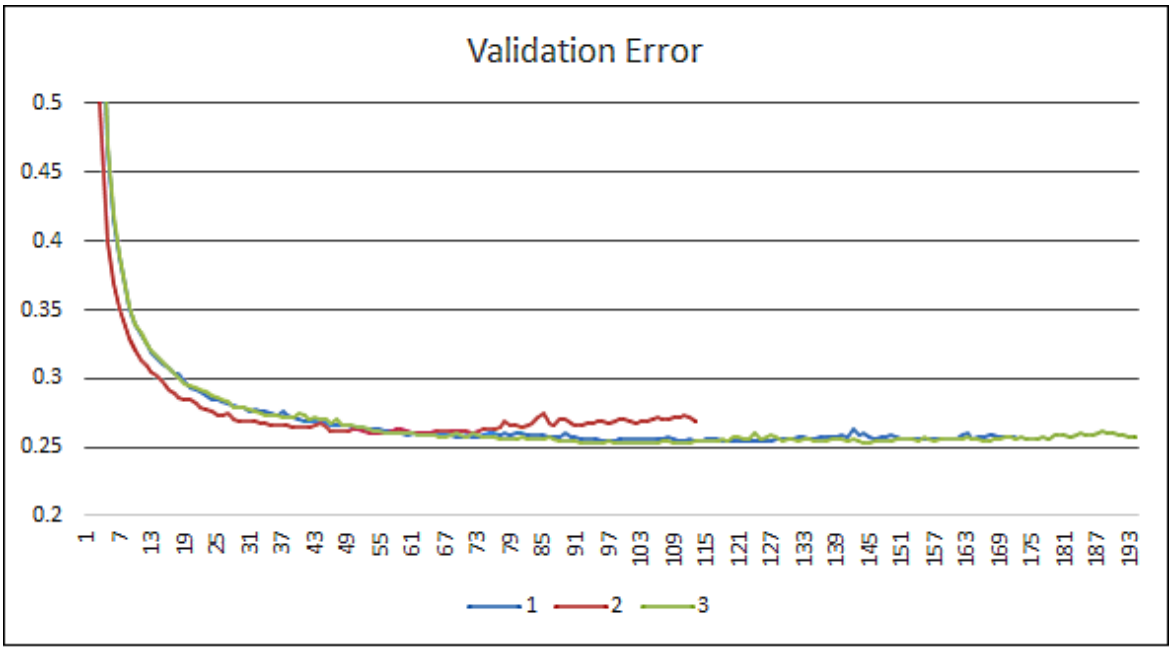

Figure 6.14 Validation error for the FROM networks

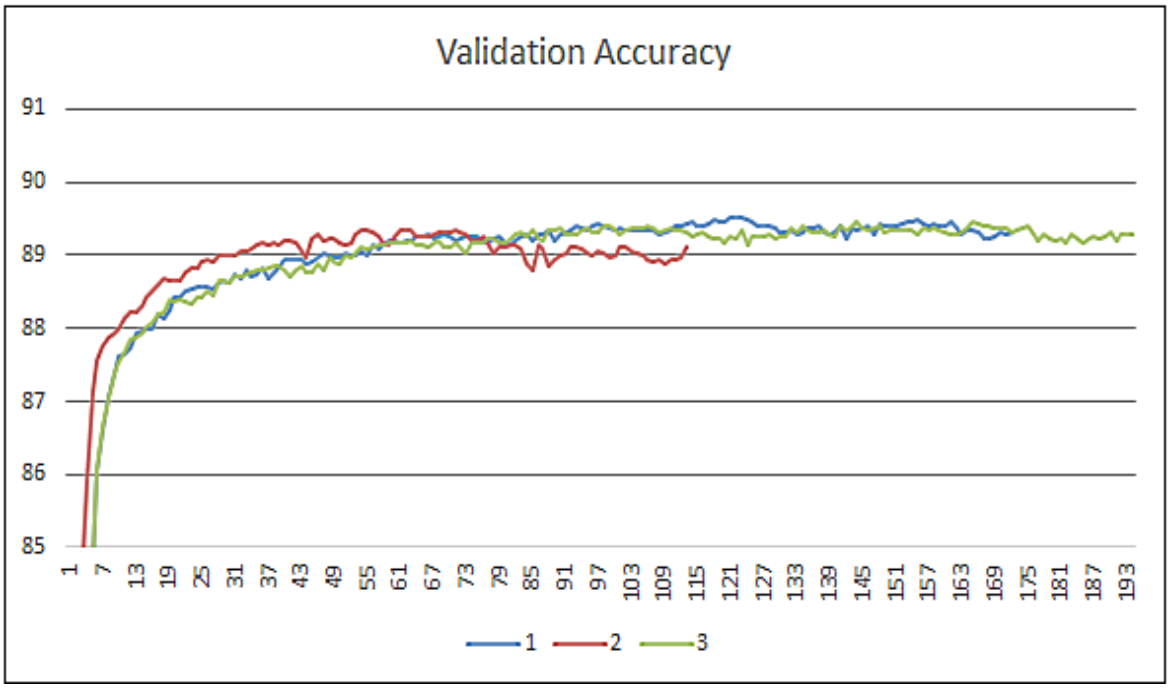

Figure 6.15 Validation accuracy for the FROM networks

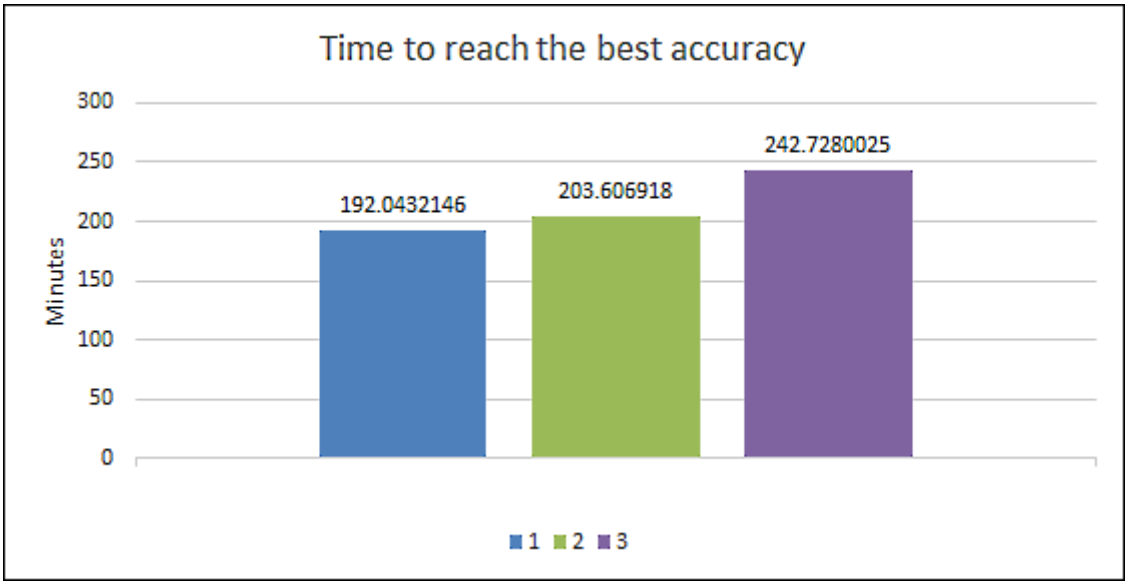

Figure 6.16 Time needed to reach the best accuracy for the FROM networks

6.2.3 REMOVE network tuning

The considerations done for the different FROM trainings of the networks holds for the REMOVE ones as shown in Figure 6.17, Figure 6.18 and Figure 6.19: the highest top validation accuracy is almost the same for each training, but this time the use of a deeper networks with no dropout nor batch normalization has provided better results both as accuracy reached (86.47% against 86.26%) and time spent (about 30 minutes less) as shown in Figure 6.19.

The networks trainings parameter are presented in Table 6.10.

Architecture	Residual Network
Network Structure 1 2	 1° layer: 200 neurons, 10 blocks of 300 neurons layers 1° layer: 200 neurons, 30 blocks of 300 neurons layers
Batch normalization	No
Dropout 1 2	 $p = 10\%, pi = 10\%$ $p = 0\%, pi = 0\%$
Regularization	L1 with weight 0.001
Update parameters 1 2	 $\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.1$ $\alpha_0 = 0.001, b1 = 0.99, b2 = 0.999, k = 0.2$
Batch size	20000

Table 6.10 Configurations of REMOVE trainings of the networks on whole dataset

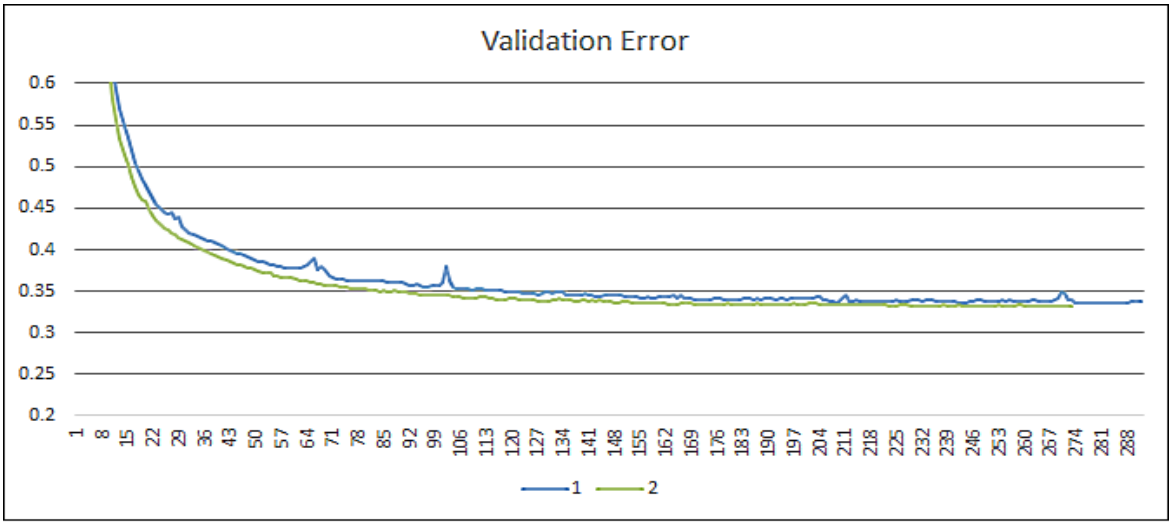

Figure 6.17 Validation accuracy for the REMOVE networks

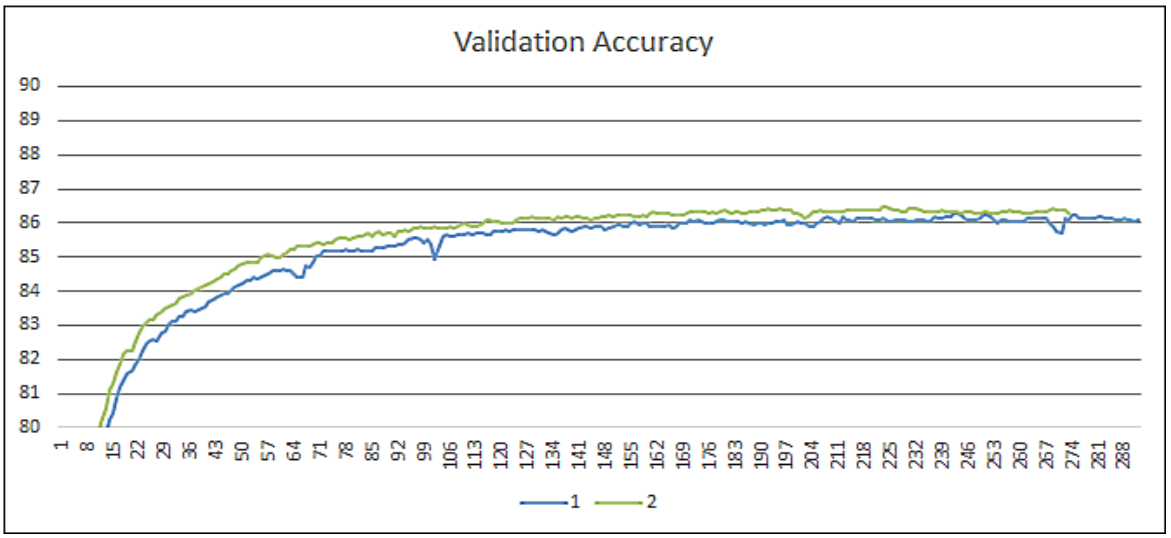

Figure 6.18 Validation accuracy for the REMOVE networks

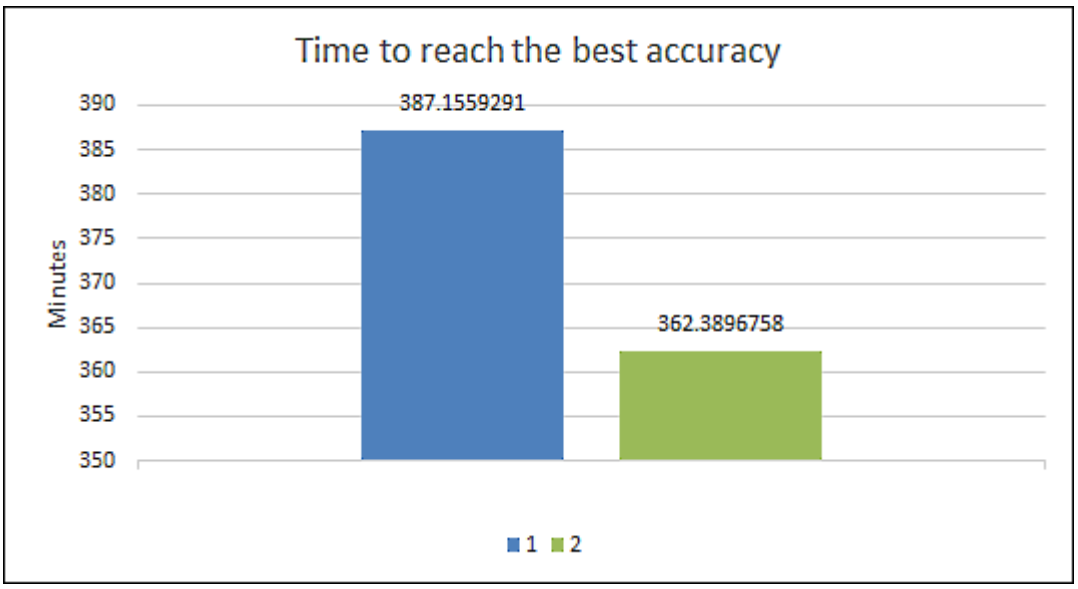

Figure 6.19 Time needed to reach the best accuracy for the REMOVE networks

6.2.4 Best Networks performance

Using the configurations which have proved to provide the best results, the three networks have been trained using the 85% of the dataset as training set, 5% as validation set and the remaining 10% as test set.

As shown in Figure 6.20, the FROM and REMOVE networks have reached a high test accuracy (respectively about 90% and 85%), while the TO network has not surpassed the 55%.

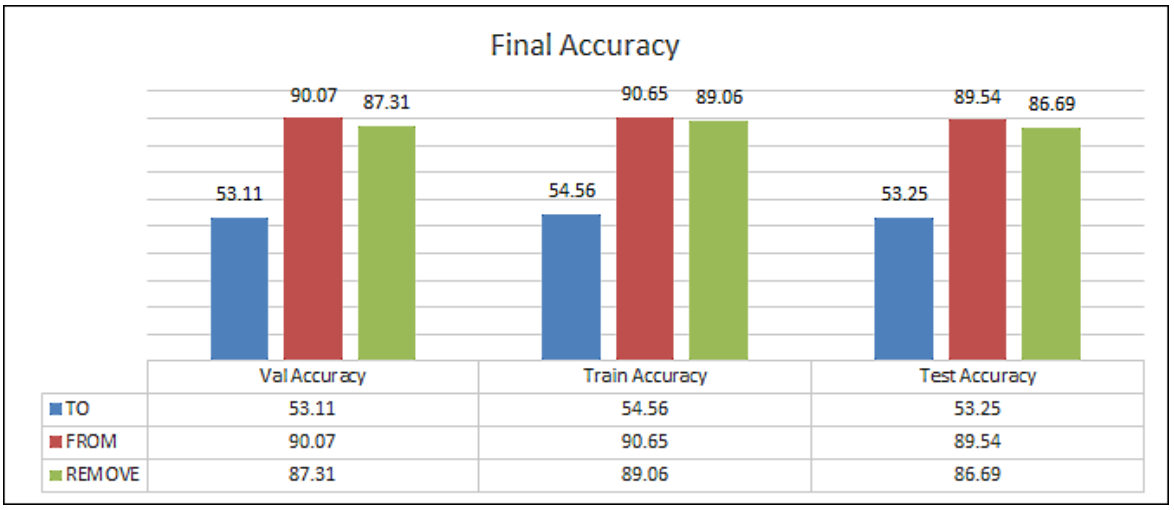

Figure 6.20 Accuracy of the best networks at the end of the test training

It is important to underline that the accuracy over the different sets of data are extremely similar, which means that the networks have maintained a good generalization and that there is not the problem of overfitting.

The difference between the results of the networks could be explained by the composition of the dataset: as explained in 5.3.2, the TO decision is the most uniformly distributed among the 24 classes, therefore probably its classification it is the most difficult to learn.

Even if two networks out of three have reached a good accuracy, their performance alone does not give a full insight of how accurate the entire system will be, because the three networks work in pipeline. Probably, being the TO network the first and most inaccurate one, it will penalize heavily the whole system.

6.3 Accuracy test

To verify if NNMM is able to approximate the symbolic reasoning of DM using the three best networks, its accuracy has been tested: each entry of the expanded dataset has been given as input to the system asking it to predict the best move without considering the legality check. Then, the accuracy of each decision and of the whole move have been measured. The accuracy on the whole move has been calculated separately for each phase of the game in which the player can be, to better understand which are the game phases in which the system plays best and worst.

Given that the 0 class is frequently present for the FROM and the REMOVE decision of the dataset, the accuracy of these decisions has been computed

separately for the cases where the target class in the dataset is 0 and where it is different.

The outcome of this test is illustrated in Figure 6.21 and it results to be:

- TO accuracy: 54.39%
- FROM accuracy: 64.40%
- FROM not 0 accuracy: 56.28%
- REMOVE accuracy: 78.19%
- REMOVE not 0 accuracy: 51.27%
- Whole MOVE accuracy: 39.54%
- Whole MOVE phase 1 accuracy: 47.91%
- Whole MOVE phase 2 accuracy: 38.71%
- Whole MOVE phase 3 accuracy: 33.67%

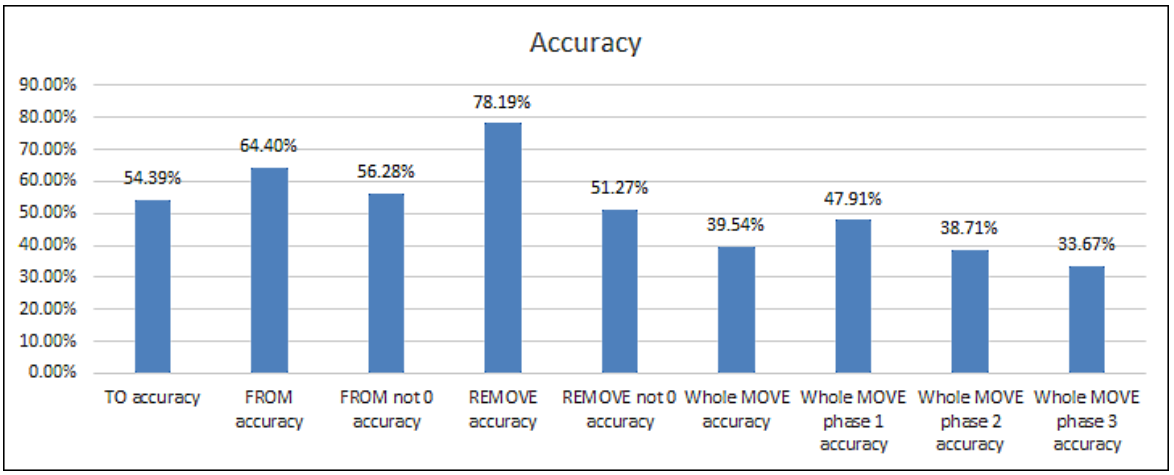

Figure 6.21 Accuracy of NNMM in the different decisions

It is evident that working in pipeline makes the networks more inaccurate and this is probably caused by the TO network which has provided the worst results and thus influences the choices of the following ones. This is more evident in the "not 0" accuracy tests, in which the decisions of each network are strongly correlated.

Nonetheless, the system is able to reach an accuracy of almost 40%, which is a satisfying achievement. The accuracy decreases with the progression of the game along the three phases, confuting the supposition expressed in 5.3.2, that a dataset with a high number of phase 2 examples would have led to a better specialization in that phase.

6.4 Legality test

As previously said, an interesting issue is whether the system is capable of understanding the game rules or not. To answer this question, the decisions taken by the system have been analysed, in each of the three parts, from the point of view of a legality check.

To better understand which rules the system has learned and which it has not, the legality of two particular cases has been investigated too:

- The legality of the whole FROM decision when the game state is in the second phase, during which the FROM decision has more constraints
- The legality of the whole REMOVE decision when it is different from 0, therefore when the chosen move consists of the removal of an opposite stone.

The outcome of this test results to be:

- TO self legality: 99.99%
- FROM self legality: 99.99%
- REMOVE self legality: 99.59%
- FROM-TO relation legality: 100%
- REMOVE-FROM-TO relation legality: 99.97%
- Whole FROM legality: 99.99%
- Whole phase 2 FROM legality: 99.99%
- Whole REMOVE legality: 99.55%
- Whole chosen REMOVE legality: 98.68%
- Whole MOVE legality: 99.55%

It is evident that the system has almost completely learnt to correctly play to Nine Men's Morris, violating the rules in less than 1% of the cases.

In particular, the rules about moving or placing a player stone have been learnt almost perfectly (less than 0.01% of error), while the rules about removing a player stone, when is possible to remove a stone and which stone should be removed, are the ones that NNMM violates the most (about 0.45% of error).

6.5 Gaming test

The accuracy test has given information about the tendency of the system to do the same move which DM has done, but it has not provided any clue about the true "skills" of NNMM as a player: even if the choice is different, the move predicted by the system could be equivalent or even better than the one elaborated by DM.

To figure out how good can be NNMM at playing a real Nine Men's Morris match, the system has played against each one of the other AI developed for the course of Foundations of Artificial Intelligence M.

To have a better measurement of the achieved results, a new training of each network has been done for this test, using the same parameters of the previous one, but changing the composition of the training set. Since the dataset is composed by the matches done by DM against the other AI, even an highly overfitted system could be able to win almost any match against those same AI, because no learning principle nor random element have been implemented in them. Therefore all the game state obtained from the regular matches between DM and the other AI have been removed from the dataset; the 5% of remaining entries have been used as validation set, obtaining the outcomes presented in Figure 6.22.

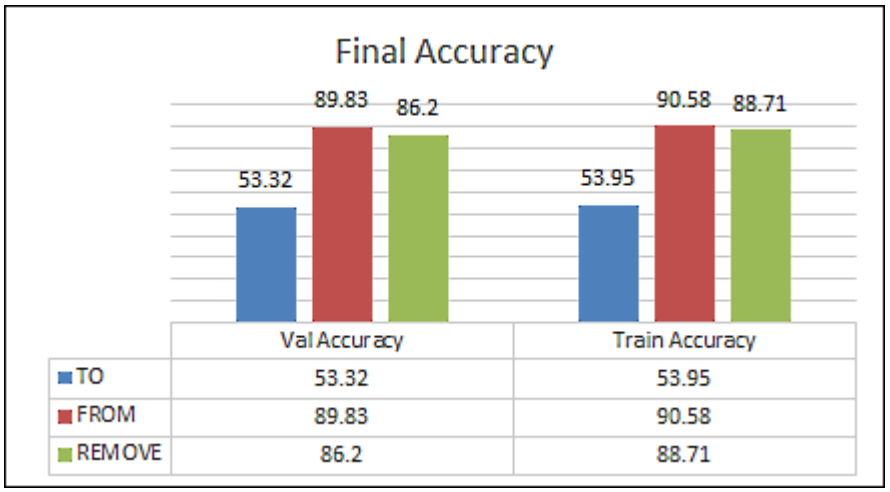

Figure 6.22 Accuracy of the best networks at the end of the game training

As presented in Table 6.11, NNMM has been able to win 18 matches out of 34, with 7 draws and 9 defeats, performing worse than DM. In particular the two matches against its "teacher" have ended both with a victory of DM eliminating 7 stones of NNMM.

An interesting observation that can be done is that in many cases, even if both the AI have won the match, the victory has been achieved in different ways: for example, against AI n° 12 as white player, DM has won suffocating the adversary (leaving him with no possible moves), while NNMM has removed 7 of its stones. Moreover NNMM has outperformed DM in a case, defeating an opponent that DM has not been able to beat: the match 4vsNNMM has ended with the latter suffocating the former, while the match 4vsDM ended in a draw.

This outcomes suggests that NNMM has learnt its own evaluation strategy, which is different from DM's one and could be better, worse or equivalent to it according to the specific case.

The system has won most of the games as White player, this could be caused by an advantage of the white player in the game or by a better ability of the system with white checkers. To verify this, the system, playing as white, has challenged itself, playing as black. The game has ended in a draw, suggesting that the differences in the results could be caused by characteristics of the game itself.

AI	DM as Black	DM as White	NNMM as Black	NNMM as White
1	WIN (removing)	WIN (timeout)	WIN (removing)	WIN (removing)
2	WIN (removing)	WIN (timeout)	LOSS (suffocating)	LOSS (suffocating)
3	WIN (timeout)	WIN (timeout)	DRAW	LOSS (removing)
4	DRAW	WIN (suffocating)	WIN (suffocating)	WIN (suffocating)
5	WIN (suffocating)	WIN (suffocating)	WIN (suffocating)	WIN (suffocating)
6	WIN (suffocating)	WIN (removing)	LOSS (removing)	WIN (removing)
7	WIN (suffocating)	WIN (suffocating)	DRAW	WIN (removing)
8	WIN (suffocating)	WIN (suffocating)	DRAW	WIN (removing)
9	WIN (suffocating)	WIN (removing)	WIN (suffocating)	DRAW
10	DRAW	WIN (removing)	LOSS (suffocating)	LOSS (removing)
11	WIN (removing)	WIN (timeout)	DRAW	WIN (removing)
12	WIN (suffocating)	WIN (suffocating)	WIN (removing)	DRAW
13	WIN (suffocating)	WIN (suffocating)	WIN (suffocating)	WIN (removing)
14	WIN (removing)	WIN (suffocating)	LOSS (suffocating)	WIN (removing)
16	WIN (suffocating)	WIN (removing)	DRAW	WIN (timeout)
17	WIN (suffocating)	WIN (suffocating)	WIN (suffocating)	WIN (timeout)
DM	DRAW		LOSS (removing)	LOSS (removing)
NNMM			DRAW	

Table 6.11 Outcomes of the matches played by DM and NNMM against other AI

One last consideration is that the legality check of NNMM has not modified the choices of the network in any of the matches; this confirms what has been said previously, that the system could play most of the matches following the rules independently from a symbolic validation.

Capitolo 7

Conclusions and future developments

The resulting product of this thesis is Neural Nine Men's Morris (NNMM), a software program that is able to play the game of Nine Men's Morris relying only upon neural networks knowledge, and a dataset of good moves for the same game.

The dataset is available online [57] and contains 1,628,673 states both reachable and unreachable during a normal match, decreasing the probability of reaching a training state during a testing match. The moves contained in it could be different from the optimal ones, however it constitutes a good knowledge base, from which other AI system can learn to play the game.

The NNMM system has been proven to be able to play following the rules of the game, while its ability as a player is generally worse than the symbolic AI used to generate the dataset. Having trained the system upon the choices of another AI system, it was unlikely for it to defeat its "teacher" at Nine Men's Morris, indeed it has lost the match both as white and black player. The outcomes of the matches disputed against other AI have been very different from the results of the "teacher" matches, worse in some cases but better in one; this suggests that the system has learnt its own evaluation strategy, which can be better or worse than the training one according to the specific case.

7.1 NNMM and its neural networks

Even if NNMM has demonstrated to not be able to win a whole match against DM, it could be interesting to study how it performs in certain game phases or situations starting from a fixed state. For example, if it is able to win a match starting from a vantage state or if it is able to reach a draw starting from a disadvantaged one.

As previously said, due to the limited time and resources available for this study, many training configurations have not been tested, so it is probable that a further tuning of the system hyper-parameters could lead to a better configuration for the networks and therefore to better results.

Furthermore, the training could be enhanced with more advanced techniques, such as k fold cross validation, regularization weight decay or introducing more randomness, for example using residual networks with stochastic depth [64].

The application of unsupervised learning techniques could enhance the networks performances, making them learning from games against other AI systems or even against themselves.

Another influent element that could be studied is how the performance of the networks changes with a different input representation: besides the proposed integer representation, an input which embodies pre-calculated features could help the networks to understand which are the good moves. For example, which position must the player occupy to close a mill or to prevent a closing of the adversary could be codified in the input. It is important to underline that these features could be extracted with symbolic techniques but also applying sub-symbolic extraction methods, without compromising the aim of the study.

Finally, a different relation between the networks, therefore a different architecture of the whole system, could improve the final outcome. The simpler possibility is a different order of the networks, for example postponing the TO network, which has resulted to be the most inaccurate. A more interesting proposal could be to use 5 networks instead of 3, using two networks for the FROM and REMOVE parts: a network for the decision about if the part should be present in the move or not and a network which, in the first case, predicts the best position.

7.2 Symbolic and sub-symbolic systems

In this case of study, sub-symbolic techniques have proven themselves capable to respect rules and compliance with a success rate of almost 100%. On the contrary, to emulate a symbolic strategy using neural networks seems to be more tough.

Besides their performance, another issue about full sub-symbolic systems is that they are not transparent: the knowledge stored in the parameters is not

easily interpretable, so the logic behind their decisions cannot be understood and this can be an obstacle in matter of safety and reliability of the programs.

Therefore it is unlikely that pure sub-symbolic AI could totally replace symbolic ones, but maybe they could have a higher relevance in hybrid systems: the sub-symbolic part could select a set of alternatives among which the symbolic part of the system will choose or, in other words, it could be used to do a fast initial pruning of the search space or as tool to order the states that should be explored.

Indeed, if a properly trained network is able to respect the constraints of a problem in almost any cases and to provide good (even if not optimal) outputs, the symbolic part of the system could simply analyse the more probable choices of the network, instead of the most probable only, verifying their legality and evaluating them with a symbolic strategy.

Applying this consideration to this case of study it is possible to suggest the following workflow: for each input, a sub-symbolic system such as NNMM could suggest a set of moves (for example, if each network provides 3 decisions, for each input state the output could consist of 27 moves ranked by their probability) and a symbolic system could check the legality of each move and evaluate them with a score function or even do a very small search starting with those moves.

In case of necessity, a way to improve the respect of constraints could be to add a legality contribute in the loss function: heavily penalizing the illegal choices, the networks could learn to observe the rules as first thing, then learn the optimization strategy.

The dichotomy between symbolic and sub-symbolic can be overcome not only with hybrid systems made by cooperating but separate parts, but even with hybrids where elements of both the typologies are fully integrated together. In these systems often there is a symbolic interpretation of the sub-symbolic connections between the elements and symbolic relations can be realised through sub-symbolic links. Even if studies in this field have been conducted since the 80s, the possible applications of the new deep network technologies still have to be fully investigated, so it remains an open research field [1].

Bibliography

[1] M. Lippi, “Reasoning with Deep Learning: an Open Challenge,” in *Deep Understanding and Reasoning: A Challenge for Next-generation Intelligent Agents 2016*, Genova, Italy, 2016.

[2] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Upper Saddle River, New Jersey: Prentice Hall, 2009.

[3] J. Haugeland, Artifical Intelligence: the Very Idea, Cambridge, Massachussets: MIT Press, 1985.

[4] A. Newell and H. A. Simon, "Computer science as empirical inquiry: symbols and search," *Communications of the ACM,* vol. 19, no. 3, pp. 113-126, 1976.

[5] R. Sun, "Artificial Intelligence: Connectionist and Symbolic Approaches," 1999.

[6] H. Dreyfus, What Computers Still Can't Do, New York: MIT Press, 1979.

[7] R. A. Brooks, "Elephants Don't Play Chess," *Robotics and Autonomous Systems,* vol. 6, no. 1-2, pp. 3-15, 1990.

[8] R. C. Bell, Board and Table Games from Many Civilisations, Oxford: Oxford Univeristy Press, 1969.

[9] A. Robert, "Jeux et divertissements des indigènes d'Algérie," *Revue Africaine,* vol. 62, pp. 62-84, 1921.

[10] J. M. Jama, Shax: the preferred game of our camel-herders and other traditional African entertainments, Rome: Sun Moon Lake, 2000.

[11] R. Gasser, “Solving Nine Men's Morris,” *Computational Intelligence,* vol. 12, no. 1, pp. 24-41, 1996.

[12] G. E. Gévay and G. Danner, “Calculating Ultra-Strong and Extended Solutions for Nine Men's Morris, Morabaraba, and Lasker,” *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 8, no.

3, pp. 256-267, 2016.

[13] M. J. Osborne and A. Rubinstein, "Chapter 6: Extensive Games with Perfect Information," in *A Course in Game Theory*, Cambridge M.A., The MIT Press, 1994.

[14] "Draughts - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Draughts.

[15] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu and S. Sutphen, "Checkers Is Solved," *Science*, vol. 317, no. 5844, p. 1518–1522, 2007.

[16] V. Allis, "Searching fo Solutions in Games and Artificial Intelligence," Maastricht, The Netherlands, 1994.

[17] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu and S. Sutphen, "Checkers Is Solved," *Science*, vol. 317, no. 5844, pp. 1518-1522, 2007.

[18] "Play Chinook," [Online]. Available: http://webdocs.cs.ualberta.ca/~chinook/play/.

[19] IBM, "IBM Research | Deep Blue | Overview," [Online]. Available: https://www.research.ibm.com/deepblue/meet/html/d.3.3a.html. [Accessed 13 January 2017].

[20] M. Buro, "Logistello: A Strong Learning Othello Program," in *19th Annual Conference Gesellschaft für Klassifikation e.V.*, Basel, 1995.

[21] M. Enzenberger and M. Müller, "Fuego – An Open-source Framework for Board Games and Go Engine Based on Monte-Carlo Tree Search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 259 - 270, 2010.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, p. 484–489, 2016.

[23] W. S. McCulloch and P. Walter, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*,

vol. 5, no. 4, pp. 115-133, 1943.

[24] D. E. Rumelhart, G. E. Hinton and R. J. Williams., “Learning representations by back-propagating errors,” *Nature,* no. 323, pp. 533-536, 1986.

[25] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/linear-classify/.

[26] L. Breiman, “Bagging Predictors,” *Machine Learning,* vol. 24, no. 2, p. 123–140, 1996.

[27] Y. Freund and R. E. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,” *Journal of Computer and System Sciences,* vol. 55, no. 1, pp. 119-139, 1997.

[28] F. Rosenblatt, "The Perceptron--a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, New York, 1957.

[29] M. Minsky and S. Papert, Perceptrons, Cambridge MA: The MIT Press, 1969.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation,* vol. 1, no. 4, pp. 541-551, 1989.

[31] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science,* no. 313, pp. 504-507, 2006.

[32] G. E. Dahl, D. Yu, L. Deng and A. Acero, “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition,” *IEEE Transactions on Audio, Speech, and Language Processing,* vol. 20, no. 1, pp. 30 - 42, 2012.

[33] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25 (NIPS 2012),* Lake Tahoe, Nevada, 2012.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV),* vol. 115, no. 3, pp.

211-252, 2015.

[35] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas and H. S. Seung, “Digital selection and analogue amplification coesist in a cortex-inspired silicon circuit,” *Nature,* no. 405, p. 947–951, 2000.

[36] X. Glorot, A. Bordes and Y. Bengio, “Deep sparse rectifier neural networks,” *Aistats*, vol. 15, no. 106, p. 275, 2011.

[37] 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,* vol. 15, no. 1, pp. 1929-1958, 2014.

[38] K. He, X. Zhang, S. Ren and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *arXiv preprint,* vol. arXiv:1502.01852, 2015.

[39] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv e-prints,* vol. arXiv:1412.6980, 2014.

[40] H. Shimodaira, “Improving predictive inference under covariate shift by weighting the log-likelihood function,” *Journal of Statistical Planning and Inference,* vol. 90, no. 2, pp. 227-244, 2000.

[41] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint,* vol. arXiv:1502.03167, 2015.

[42] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[43] K. He, X. Zhang, S. Ren and J. Sun, “Identity Mappings in Deep Residual Networks,” *ArXiv e-prints,* vol. arXiv:1603.05027, March 2016.

[44] M. S. Ebrahimi and H. K. Abadi, "Study of Residual Networks for Image Recognition".

[45] G. Huang, Z. Liu and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *arXiv e-prints,* vol. arXiv: 1608.06993, 2016.

[46] J. Lehman and R. Miikkulainen, "Neuroevolution," *Scholarpedia,* vol. 8,

no. 6, p. 30977, 2013.

[47] K. Chellapilla and D. B. Fogel, "Evolving Neural Networks to Play Checkers," *IEEE TRANSACTIONS ON NEURAL NETWORKS*, vol. 10, no. 6, pp. 1382-1391, 1999.

[48] M. Lai, "Giraffe: Using Deep Reinforcement Learning to Play Chess," London, 2015.

[49] O. E. David, N. S. Netanyahu and L. Wolf, "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess," in *Artificial Neural Networks and Machine Learning -- ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II*, Barcelona, 2016.

[50] Gunawan, H. Armanto, J. Santoso, D. Giovanni, F. Kurniawan, R. Yudianto and Steven, "Evolutionary Neural Network for Othello Game," *Procedia - Social and Behavioral Sciences*, vol. 57, pp. 419-425, 2012.

[51] S. Y. Chong, D. C. Ku, H. S. Lim, M. K. Tan and J. D. White, "Evolved neural networks learning Othello strategies," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 2003.

[52] A. J. Pinkney, "Development of an Artificial Neural Network to Play Othello," 2009.

[53] D. E. Moriarty and R. Miikkulainen, "Discovering Complex Othello Strategies Through Evolutionary Neural Networks," *Connection Science*, vol. 7, pp. 195-209, 1995.

[54] A. Leouski, "Learning of Position Evaluation in the Game of Othello," 1995.

[55] M. van der Ree and M. Wiering, "Reinforcement Learning in the Game of Othello: Learning Against a Fixed Opponent and Learning from Self-Play," in *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, Singapore, 2013.

[56] C. Clark and A. J. Storkey, "Teaching Deep Convolutional Neural Networks to Play Go," *arXiv preprint*, vol. arXiv:1412.3409, 2014.

[57] A. Galassi, “Nine Men's Morris Good Moves Dataset,” 3 3 2017. [Online]. Available: http://ai.unibo.it/DatasetNineMenMorris. [Accessed 3 3 2017].

[58] F. Chesani and P. Mello, “Fondamenti di Intelligenza Artificiale M, Gioco del Mulino, A.A. 2015--2016,” [Online]. Available: http://lia.deis.unibo.it/Courses/AI/fundamentalsAI2015-16/Mulino.html.

[59] F. Chesani and P. Mello, “Fondamenti di Intelligenza Artificiale M, Gioco del Mulino,” [Online]. Available: http://lia.disi.unibo.it/Courses/AI/fundamentalsAI2014-15/Mulino.html.

[60] T. Madonia, A. Di Cesare and A. Franzoni, “DeepMill,” 2015. [Online]. Available: https://github.com/Frugghi/DeepMill.

[61] Python Software Foundation, “Python Language Reference, version 2.7,” 2010. [Online]. Available: http://www.python.org.

[62] Lasagne Development Team, “Lasagne: First release,” August 2015. [Online]. Available: https://doi.org/10.5281/zenodo.27878.

[63] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. arXiv:1605.02688, 2016.

[64] G. Huang, Y. Sun, Z. Liu, D. Sedra and K. Weinberger, "Deep Networks with Stochastic Depth," *arXiv e-prints*, vol. arXiv:1603.09382, 2016.

Acknowledgements

Se sono arrivato a questo traguardo, è merito di un infinito numero di persone che mi hanno reso quello che sono. Potevate fare un lavoro migliore, ma questo è quello che è saltato fuori. Tutto sommato ne sono soddisfatto, quindi Grazie.

Ancora una volta, prima di tutto Grazie alla mia famiglia. Il nostro rapporto non sarà perfetto, ma sinceramente faccio fatica a immaginarmi qualcuno di migliore da avere attorno.

Grazie allo scoutame, persone strane, tutte diverse, che ho dovuto sopportare nei momenti più estenuanti, con cui ho condiviso le esperienze più appaganti, con cui ho fatto le cose più qualcosanti; compagni di viaggio con cui ho condiviso una parte enorme e bellissima della mia Strada e che ancora oggi mi tormentano in certi tratti, alcuni più lontani, altri più vicini (tipo la S. Inquisizione).

Grazie agli Artisti Senza Tetto e a Migliucci, che mi hanno permesso di affinare l'arte della supercazzola agli esami e di apparire figo durante le presentazioni PowerPoint. Ci sono altre cose per cui ringraziarli ma queste erano indubbiamente le più importanti.

Grazie a tutti i compagni di bevute, al NWO-CB e agli sbattitori, perché (volenti o nolenti) hanno ascoltato le mie lamentele sull'università, sulla tesi e sulla vita; dispensatori di consigli e minuziosi ricercatori del numero ottimale di birre per superare ogni problema e soprattutto Grazie a Beppe, indiscusso capo del dipartimento.

Grazie a Ceci, Ila e Mati, che miliardi di volte mi hanno ospitato, sfamato e sopportato e giuro che prima o poi vi ripagherò di tutte le mele, le arance, l'acqua e…ok, forse non proprio di tutto ecco..

Grazie a Marta, per circa un milione di cose, dalla A di abaco alla Z di zuzzurellone, servirebbe una tesi solo per elencarle tutte, magari da vecchio se nella noia vorrò prendermi un'altra laurea.

Grazie a MUgo, che crede in me e pensa che io sia "cervellone", "intelligentone" e "bravone" e che soprattutto ha saputo vedere in me cose belle che io faccio molta fatica a vedere.

E poi Grazie alle strane e bizzarre creature di Ingegneria Informatica.

Grazie ai professori che ho incontrato in questa magistrale, che hanno saputo raccontare, condividere e trasmettere la passione per quello che fanno e che studiano. Grazie a Paola e Federico che hanno mostrato interesse per ciò che ho fatto e mi hanno intrigato con il loro bizzarro mondo, al punto da convincermi a volerne farne parte (e se ci sono delle fregature, sono stati bravi a nasconderle). Grazie ovviamente anche a Marco, che mi ha proposto questo studio e mi ha aiutato a capire come far funzionare questi cervelloni artificiali.

Un Grazie è doveroso anche verso tutti i miei compagni di corso che hanno sviluppato le diverse IA, inconsapevoli del lavoro estenuante a cui le poverine sarebbero state crudelmente sottoposte.

Grazie a Elisa, grande capo dei lavori di gruppo, coordinatrice di gruppi di studio e confidente con cui lamentarsi di Pier.

Grazie a Tommy, che come minimo mi rinfaccerà a vita di avermi permesso di laurearmi grazie alla sua IA e che ha schifato circa tutti i corsi che ho scelto, suggerendomi che della vita non ho capito nulla.

Grazie a Pier, che penso di non aver mai visto a lezione in magistrale, ma che passandomi 9.99 GB di video sulle reti neurali ha pareggiato i debiti di appunti che gli ho prestato. O almeno così crede lui. Non ha ancora capito quanti favori esigerò nel caso entrambi facessimo il dottorato.

Grazie a Nic, che mi ha impunemente copiato il piano di studi cambiando giusto un esame (roba che neanche alle superiori oh), ma che così facendo mi ha salvato dal dover conoscere altre persone durante le lezioni.

Grazie a Zano, che mi ha sempre offerto una boiata, un pasto e un tetto ogni volta che ne avevo bisogno. Anche se probabilmente le ultime due qualche volta se le sarebbe risparmiate volentieri. E di boiate ce n’è sempre stata più di una sola.

Grazie a Piwa, Pug e Leo, per l’ospitalità, i deliri e i consigli, un po’ dei fratelli maggiori in questo delirante universo informatico. E tanto che ci siamo, Grazie ad Artigan e la sua Gang per questi due anni e mezzo di scorrerie!