Deep Reinforcement Learning: An Investigation into the AlphaZero Algorithm

Adriana Álvaro & Madison Chester

Abstract

This paper explores the theoretical concepts of deep reinforcement learning, specifically focusing on Monte Carlo Tree Search, the Alphazero algorithm, and the integration of deep neural networks. Finally, a breakdown of the process as a whole is demonstrated using an example. These theoretical frameworks, rooted in game theory, form the basis of advanced decision making amidst uncertain environments.

Introduction

Deep reinforcement learning is an expansive field, so we chose to focus only on the introductory algorithms that form the basis of decision making in uncertain environments. Monte Carlo Tree Search and the Alphazero algorithm are the intersection of data science and game theory. The theories mentioned can be used to play much more complicated games such as the traditional Go but also checkers, chess, and more. This paper introduces the concept of deep reinforcement learning, discusses the trade off between exploitation and exploration, explains the Markov Decision Process, overviews the Alphazero algorithm using Monte Carlo Search, and finally, details how these theoretical concepts are accomplished in practice with the utilization of deep neural networks. A simple walk-through exploring the algorithms at hand is presented at the end. While this is a continuously developing field and algorithms are being iterated upon and improved constantly, as mentioned previously, Monte Carlo Tree Search and the Alphazero algorithm form the theoretical basis for many of the more complex approaches. The intent of this paper is for the reader to gain a basis of understanding of deep reinforcement learning and the theories and algorithms crucial to making decisions in complex, uncertain environments.

1 Background

1.1 Basic Concepts and definitions

In this section, the foundational concepts of reinforcement learning (RL) are introduced. This will consist of providing definitions and explanations of fundamental ideas. These theories serve as the basis for better understanding the algorithms developed for reinforcement learning in game playing.

At the core of reinforcement learning are two essential components: the agent and the environment. The environment is a dynamic entity with which the agent interacts. This interaction occurs through a predetermined set of actions, denoted as A = A1, A2, ... This action set encompasses all feasible actions the agent has at their disposal. The primary objective of reinforcement learning algorithms is to guide the agent to interact effectively with the environment, enabling them to achieve a favorable score (a reward, denoted as r) based on a predefined evaluation metric.

This process of receiving rewards or penalties based on the agent's actions is integral to guiding the learning process in RL scenarios. In Fig 1.1.1 a simple schema of this interaction between agent and environment can be seen. At each time step t, the agent in reinforcement learning first observes the current state of the environment, S_t, and the corresponding reward, R_t. Based on this information, the agent decides its next action, denoted as A_t, which is then executed in the environment. This action leads to a transition to a new state, S_{t+1}, and the reception of a reward, R_{t+1}. The observed state, s, may not always include all information about the environment from the agent's perspective; if it only contains partial information, the environment is considered **partially observable**. Conversely, if the observation includes complete information about the environment's state, it is termed **fully observable**.

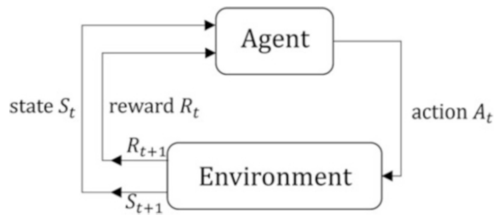

Figure 1.1.1: Basic schema: agent and environment. Source: [1]

Following the interaction between the agent and the environment, feedback is conveyed through a reward function denoted as R. This function generates an immediate reward, R_t, based on the current state of the environment and returns it to the agent at each time step. Occasionally, the reward function solely relies on the current state, expressed as $R_t = R(S_t)$. A trajectory in this context represents a sequence of states, actions, and rewards, denoted as $\tau = (S0, A0, R0, S1, A1, R1, ...)$. This trajectory records how the agent interacts with the environment over time.

The transition from one state to the next can follow either a deterministic or stochastic process. A **policy** is a strategy that defines the actions recommended to the agent based on its current state with the goal of optimizing their long-term cumulative reward. Policies that are deterministic specify a single action for each state whereas stochastic policies return a probability distribution over possible actions. The objective of RL is often to learn an optimal policy that maximizes the expected cumulative reward over time.

Two other important concepts in RL are exploitation and exploration. **Exploitation** involves maximizing the agent's performance based on its current knowledge, typically assessed by the immediate expected reward. On the other hand, **exploration** entails the process of enhancing the agent's knowledge by choosing actions with lower immediate reward but greater potential reward over time. The exploration-exploitation trade-off encapsulates the equilibrium between the agent's endeavors.

1.2 Introduction to Markov Decision Process

In this subsection, we briefly introduce the concept of a Markov Process (MP), Markov Reward Process (MRP) and Markov Decision Process (MDP). For MDP specifically we will state the optimization problem that RL aims to solve.

A **Markov Process (MP)** is a stochastic model that undergoes transitions between a set of states with the Markov property. The **Markov property** implies that the future state of the system depends solely on the current state (memory-less property) and is independent of the sequence of events leading up to that state. Let us introduce the state transition matrix, $P = [P_ij]$, where each P_ij represents the probability of transferring from the current state, S_i, to the next state, S_j.

A **Markov Reward Process (MRP)** extends the concept of a Markov Process by associating a reward with each state transition. In addition to the Markov property, an MRP includes a reward function that assigns a numerical value to each state transition, representing the immediate reward associated with moving from one state to another. A return is the cumulative reward of a trajectory, τ.

A **Markov Decision Process (MDP)** introduces the element of decision-making into the framework. In an MDP, an agent interacts with an environment by taking actions in a given state. Each action leads to a new state with associated rewards. In contrast to MRP, MDP rewards do not depend just on the state transition but also on the action taken. In Fig 1.2.1 an example schema of an MDP can be seen which represents a person working on two tasks and going to bed. Each circle represents a state, and each edge represents a state transition.

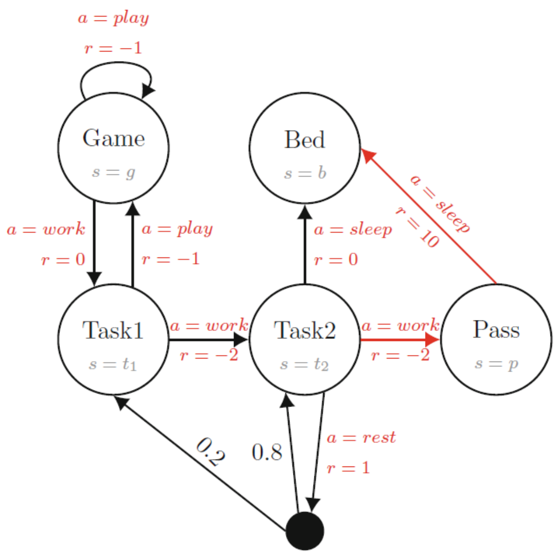

Figure 1.2.1: Markov Decision Process schema: The immediate rewards, denoted by r, are associated with the current state and the action taken. On the edges, probabilities associated to the state transition are marked. The black solid node is the initial state. Source: [1]

MDP is defined as a tuple of $< S, A, P, R, \gamma >$, and the state transition matrix becomes $p(s|s, a) = p(S_{t+1} = s|S_t = s, A_t = a)$. As a consequence, the next state now depends not only on the current state, but also on the action.

A represents the finite action set, $a1, a2, ...$, and the immediate reward becomes $R_t = R(S_t, A_t)$. A **policy** π represents the way in which the agent behaves, sampling a and s from the probability distribution function:

$$\pi(a|s) = p(A_t = a|S_t = s), \exists t \tag{1.2.1}$$

As mentioned previously, a return is the cumulative reward of a trajectory τ. The **value function** $V(s)$ represents the expected return from the state s.

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s] = \mathbb{E}_{A_t\sim\pi(\cdot|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t, A_t)|S_0 = s\right] \tag{1.2.2}$$

The Monte Carlo method helps estimate $V(s)$ by sampling a large number of trajectories starting from state s. After estimating the expected return under all states, a simple policy for the agent is the next state with the highest expected return.

The **action-value function** $Q^\pi(s, a)$, which depends on both the state and the action just taken, provides the expected return under said conditions. If the agent acts according to a policy π, it is defined as:

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|S_0 = s, A_0 = a] = \mathbb{E}_{A_t\sim\pi(\cdot|S_t)}\left[\sum_{t=0}^{\infty}\gamma^t R(S_t, A_t)|S_0 = s, A_0 = a\right] \tag{1.2.3}$$

The relation between state-value function $v^\pi(s)$ and action-value function $q^\pi(s, a)$ is given by:

$$v^\pi(s) = \mathbb{E}_{a\sim\pi}[q^\pi(s, a)] \tag{1.2.4}$$

and while in the optimal:

$$v^*(s) = \max_a q^*(s, a) \tag{1.2.5}$$

The expected return is the expectation of cumulative returns over all possible trajectories under a policy. Therefore, the goal of reinforcement learning is to find the highest expected return by optimizing the underlying policy. Mathematically, given the start-state distribution, ρ_0, and the policy, π, the probability of a T-step trajectory for an MDP is:

$$p(\tau|\pi) = \rho_0(S_0)\prod_{t=0}^{T-1} p(S_{t+1}|S_t, A_t)\pi(A_t|S_t) \tag{1.2.6}$$

Given the reward function, R, and all possible trajectories, τ, the expected return, $J(\pi)$, is defined as follows:

$$J(\pi) = \sum_\tau p(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{1.2.7}$$

The resulting reinforcement learning optimization problem therefore is:

$$\pi^* = \arg\max_\pi J(\pi) \tag{1.2.8}$$

2 AlphaZero algorithm

2.1 Overview

The Key Components of the AlphaZero algorithm are:

- Self-play process using Monte Carlo Tree Search (MCTS) for data collection: AlphaZero utilizes Monte Carlo Tree Search (MCTS), a heuristic search algorithm used in decision processes. MCTS combines tree exploration and exploitation strategies to estimate the state value, $v(s)$, of each node.
- Deep Neural Networks: A convolutional network is trained with the data generated by the MCTS algorithm to predict the state value, $v(s)$, at each node and the probability of every possible action, a, from the given state.

This process is illustrated in the following figure:

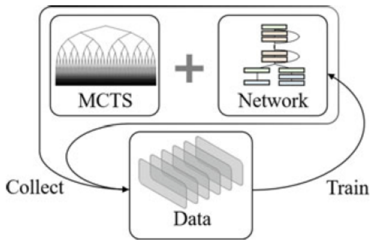

Figure 2.1.1: AlphaZero algorithm schematic overview. Source: [1]

The idea is that the network assists the MCTS by predicting the value of the state value, $v(s)$, and the probability distribution of the possible actions from a given state in order to build the decision tree. Once it has been built and explored, the generated data is fed to the neural network to train and improve its performance. This will simultaenously improve the estimations of the MCTS algorithm under its next iteration.

This resembles the behavior of a game player: one plays a game based on their current knowledge/intuition, and with the results obtained from that game, one updates/grows their knowledge of the game.

2.2 Monte Carlo Tree Search MCTS

Monte Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree (see Fig 2.2.1 according to the results.

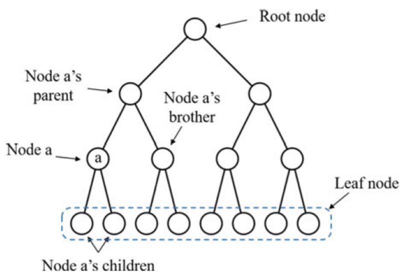

Figure 2.2.1: Monte Carlo Tree Search scheme. Source: [1]

Each node in the tree has associated properties:

- Action (A): the action taken to reach that node
- Visit Count (N): the number of times the node has been visited during simulations
- Node's total reward (W): initialized to zero
- Node average reward (Q): indicates the state-action value of the node, $Q(s, a) = \frac{W}{N}$
- Prior Probability (P): the probability of selecting action a, determined by the policy defined by the neural network

In Algorithm 1, the Monte Carlo Tree Search updates the Q values for all visited state-action pairs (s, a) along the sampled trajectory from the current state, S_t. This update is performed by considering the average reward obtained from all trajectories originating from the same state-action pair (s, a). In the section dedicated to the break-down of the AlphaZero 2.4, this algorithm will be further explained.

Algorithm 1: Monte Carlo Tree Search

```
Input: Model $M$, Action set $A$, Simulation policy $\pi$
Output: Action $A_t$
1: for each action $a \in A$ do
2:     for each episode $k \in \{1, 2, \dots, K\}$ do
3:         Following the model $M$ and simulation policy $\pi$, roll out in the environment starting from current state $S_t$
4:         Record the trajectory as $\{S_t, a, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \dots, S_T\}$
5:         Update the $Q$ value of every $(S_i, A_i)$, $i = t, \dots, T$ by mean return starting from $(S_i, A_i)$ with $A_t = a$
6:         Update the simulation policy $\pi$ according to the current $Q$ values
7:     end for
8: end for
9: Output the action with maximum $Q$ value at the current state, $A_t = \arg\max_{a \in A} Q(S_t, a)$
```

2.3 Deep Neural Network

A deep convolutional network, $f_\theta(s)$, is used to estimate the probability distribution of actions and state values, (p, v), at each node. In the case of study, *ResNet* structure is used (see Fig 2.3.1). The network inputs are the game states.

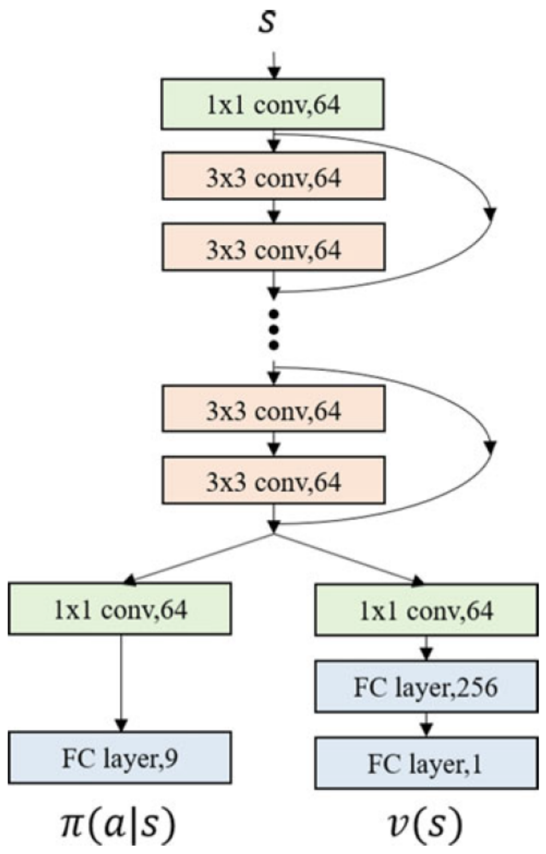

Figure 2.3.1: ResNet Structure. Its outputs are the probability distribution $\pi(a|s)$ and the state value $v(s)$ separately. Source: [1]

The loss function, l, is given by:

$$l = (r - v)^2 - \pi^T \log(p) + c\|\theta\|^2 \tag{2.3.1}$$

Eq. 2.3.1 combines mean squared error over the state value, v, cross entropy over the action's probability distribution, and L2 regularization. Parameter c controls the strength of the regularization over the parameters.

The network is trained with the data generated by the MCTS. Given a representation of a certain state, the network outputs an estimation of the probability distribution for all possible actions from that state, and the estimated state value for that node (i.e. the expected return for that state). The "true" probabilities for the actions are a function of the visit count, N (see Eq.2.4.3), of each node after the MCTS is completed. The "true" state values are found heuristically by MCTS.

2.4 Algorithm breakdown

At each iteration of the tree search there are three steps:

- **Selection:** Actions are chosen from the root node according to a specific policy.
- **Expand and evaluate:** Child nodes (understood as the possible next states) are added to the current leaf node. The probability of each action and the state value of the child nodes are then outputted by the DNN explained in subsection 2.3.
- **Backup:** The outcome of the simulation is propagated back, updating the information at each selected node during the current iteration. If the leaf node is the termination of the game, the value is given by the game directly, otherwise it is given by the DNN.

For all steps mentioned, no real move on the game is done, the possibilities are instead explored.

At the *selection* step, the child node with the highest Upper Confidence bound in Tree (UCT) is selected:

$$a = \arg\max_a \left(Q(s,a) + U(s,a)\right) \tag{2.4.1}$$

$$U(s,a) = c_{puct} \cdot \frac{P(s,a)}{\sqrt{1 + N(s,b)}} \tag{2.4.2}$$

The first term encourages exploitation of nodes with higher rewards, and the second term, $U(s,a)$, represents the exploration of less visited nodes. The c_{puct} parameter allows us to regulate the amount of exploration. In the *backup* step, we update the information of each selected node within the trajectory.

$$N(s,a) = N(s,a) + 1 \; W(s,a) = W(s,a) + v \; Q(s,a) = \frac{W(s,a)}{N}$$

The following example will break this down further:

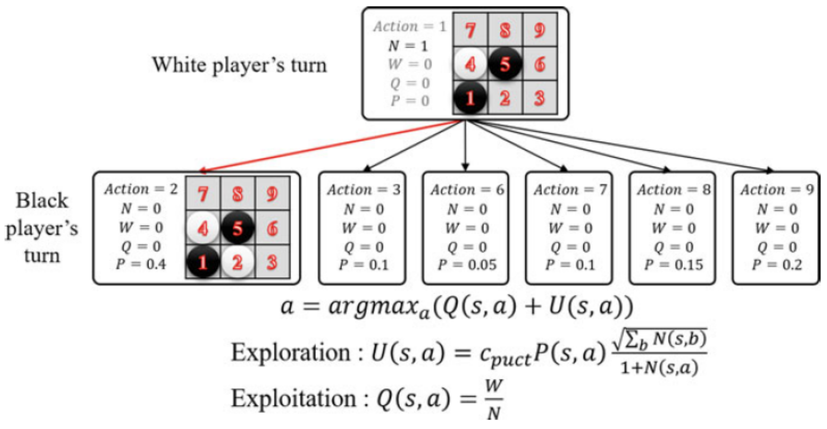

Figure 2.4.1: **Selection**: The node with with highest UCT value has been selected from the child nodes that hang from root one. (Recall that the child nodes are all the possible actions that can be taken from the given state.)

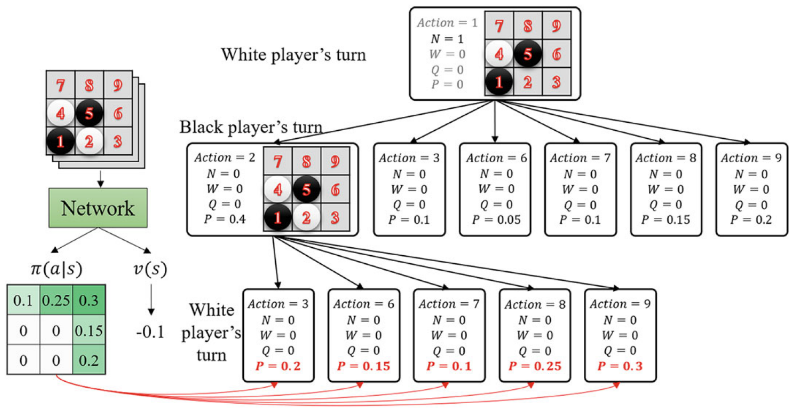

Figure 2.4.2: **Expand and Evaluate**: Once the child node to visit has been selected ($a = 2$ in this case), we expand it. This means adding their child nodes and evaluating them. (The probability for each action and the expected return for each node is given by the network.)

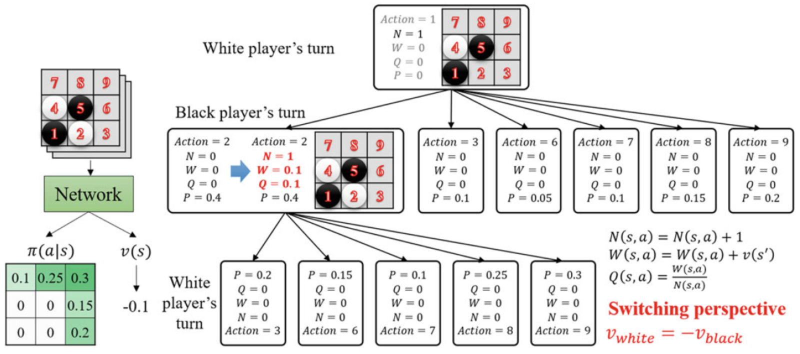

Figure 2.4.3: **Backup**: The information of this node and the previously selected ones are updated. In this case, just the current and root node are updated because we are in the first "layer" of nodes.

An important remark is that the tree is built from the perspective of both players at the same time considering the algorithm is self-playing. This means that the values of the states and their representation are according to the perspective of the player whose turn is upcoming. This implies that in every "layer" of nodes, we toggle the state values: what is a reward of 1 for one player, is a reward of -1 for the opponent, $v_{white} = -v_{black}$.

From Fig 2.4.3, the next node to be selected would be the one for $action = 9$. Once again, it would be expanded and evaluated. Next, we would backup the results by updating the information for both the current node, the previously selected node, and the root node. This process continues until a terminatal state is reached in which the reward is given by the game and not the DNN (1 if the game is won by the player with the upcoming turn, -1 if lost). See Fig 2.4.4

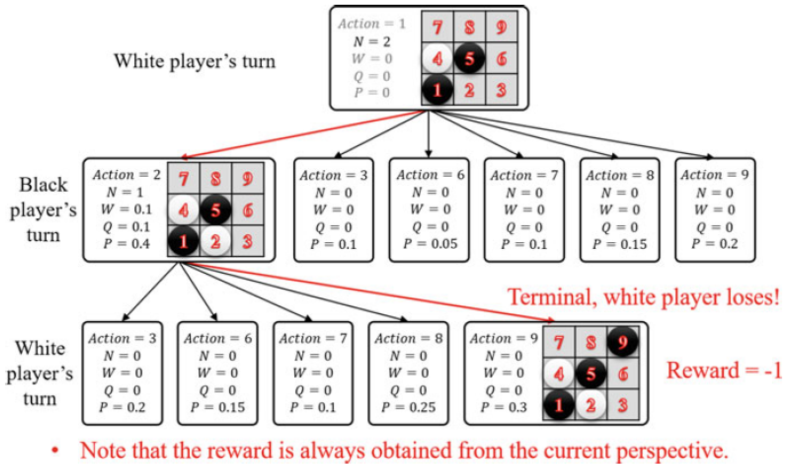
White player's turn
Action = 1
N = 2
W = 0
Q = 0
P = 0
Black player's turn
White player's turn
Terminal, white player loses!
Reward = -1
Note that the reward is always obtained from the current perspective.

Figure 2.4.4: End of game: The last selected node was action = 9 which makes the white player lose, and therefore, the reward is -1.

Here, we have conducted the MCTS three times while no move was performed. After an arbitrarily large number of searches, the information in the nodes will be more accurate, and a move can be chosen according to the visits of each node:

$$\pi(a|s) = \frac{N(s,a)^{\frac{1}{\tau}}}{\sum_b N(s,b)^{\frac{1}{\tau}}} \tag{2.4.3}$$

where τ is a parameter to measure the amount of exploration.

3 Conclusion

In summary, this theoretical exploration of deep reinforcement learning delves into three pivotal concepts: Monte Carlo Tree Search, the AlphaZero algorithm, and the integration of deep neural networks in RL. Rooted in game theory, these frameworks collectively advance the field's understanding of intelligent decision-making in complex and uncertain environments.

Monte Carlo Tree Search provides a systematic and efficient approach for navigating decision spaces. Its adaptability positions and the good trade-off it offers between exploration and exploitation make it a key element of RL. The AlphaZero algorithm leverages self-play and deep neural networks to master diverse games while deep neural networks offer a scalable foundation for learning intricate representations. When utilized together, these concepts empower agents to traverse vast decision spaces, learn optimal strategies, and adapt to changing environments.

The underlying algorithms and techniques employed by AlphaZero, have actually been adapted to various domains beyond board games: financial trading (for strategic decision making), optimizing supply chain management, decision making in autonomous vehicles, energy grid optimization to manage supply and demand, among others. This adaptability makes evident its remarkable potential in the framework of intelligent decision-making across multiple fields.

References

[1] H. Dong, Z. Ding, S. Zhang, E. Fundamentals, and A. Research, “Deep reinforcement learning,” 2020.