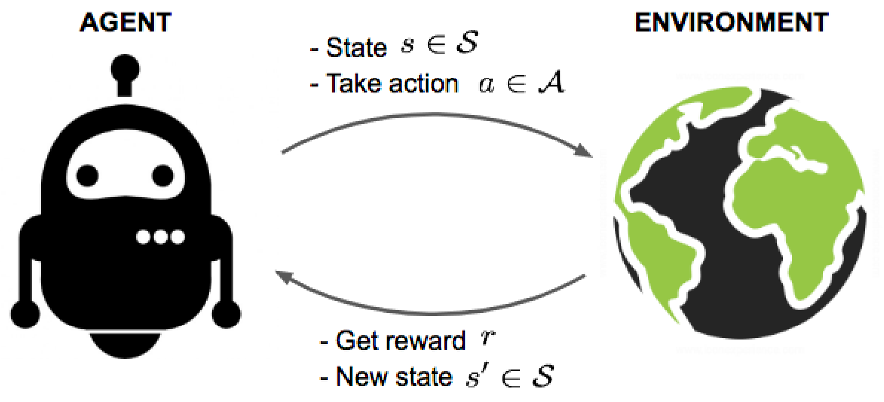
Reinforcement Learning

Vishal Raj (17EE35027)
Ritam Talukder (17EE33001)
Sanket Kumar Singh (17EE35010)

27th March 2021

1 Introduction

Figure 1: An agent interacts with the environment, trying to take smart actions to maximize cumulative rewards.


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Deep Learning has most of the attention when it comes to AI. But there are other promising areas in AI as well, one such being Reinforcement Learning. Companies like DeepMind and OpenAI have made breakthroughs using this approach to AI.

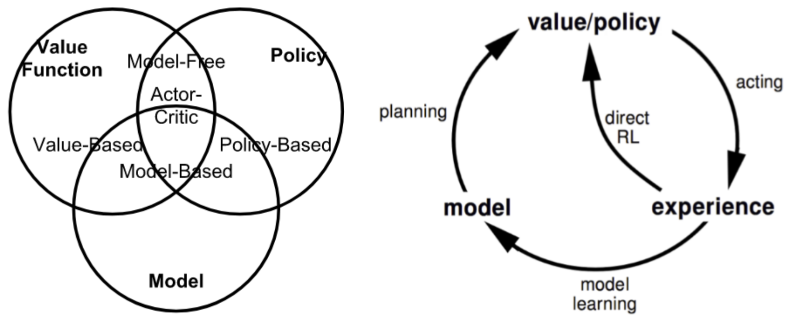
Reinforcement Learning can be traced back to psychology literature in the 1950s. In its simplest form, it states that the frequency of a behavior will go up or down depending on the direct consequences of that behavior. This is true of animal behavior as well as human behavior.

Recently, Reinforcement Learning has been in the news for many reasons. AlphaGo defeated Lee Sedol, the best professional human player in the game of Go. Very soon the extended algorithm AlphaGo Zero beat AlphaGo by 100-0 without any supervised learning on human knowledge. Top professional game players lost to the bot developed by OpenAI on DOTA2 1v1 competition.

The agent, in the context of Reinforcement Learning, interacts with its surrounding environment and observes a specific activity and responds to the environment to produce a desired response. The agent engages in a series of trials to obtain the highest score or reward. After several attempts, it learns to maximize its cumulative reward

2 So, what is Reinforcement Learning?

Figure 2: Summary of approaches in RL based on whether we want to model the value, policy, or the environment. (Image source: reproduced from David Silver's RL course lecture 1.)



Consider an agent in an unknown environment. The agent gets a reward on interacting with the environment. Now, to maximize the cumulative reward, the agent will take actions. The goal in Reinforcement Learning is to learn strategy for the agent from the rewards obtained by interacting with the environment

Some formal definitions:-

There is an environment on which the agent acts. The reaction of the environment is defined by via a model. The agent can be in multiple states and in each state has a given set of actions to perform. Transition probabilities determine the state in which the agents would arrive. After an action is taken the environment sends a reward. The transition probabilities are defined by the model. There are two scenarios in here depending on knowledge of agent about the model - Model is known - Dynamic Programming solution can be used in this case as we know the model completely. Model is not known - We try to learn the model itself (one possible way to define the model can be a neural network).

Using a policy the agents decide on what is the optimal action to take in a particular state so that the total rewards is maximized. Every state has a value function associated with it. The value function predicts the expected amount of future rewards we will be able to receive by enacting a corresponding policy.

Thus, the value function quantifies how good a state is. We try to learn both the policy and the value function in reinforcement learning.

The interaction between the agent and the environment involves a sequence of actions and observed rewards in time, $t = 1, 2, \ldots, T$. During the process, the agent accumulates the knowledge about the environment, learns the optimal policy, and makes decisions on which action to take next so as to efficiently learn the best policy. Let us call the state, action, and reward at time step t as S_t, A_t, and R_t, respectively. The full description of an interaction sequence is called an **episode** (also known as "trial" or "trajectory") and the sequence ends at the terminal state S_T:

$$S_1, A_1, R_2, S_2, A_2, ..., S_T \tag{1}$$

Some important terms in the context of RL are:-

- **Model-based**: Rely on the model of the environment; either the model is known or the algorithm learns it explicitly.
- **Model-free**: No dependency on the model during learning.
- **On-policy**: Use the deterministic outcomes or samples from the target policy to train the algorithm.
- **Off-policy**: Training on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy.

2.1 Model: Transition and Reward

The model describes the environment. With the model, we can learn or infer how the environment would interact with and provide feedback to the agent. Transition probability function P and reward function R are the two major parts of the model.

Let's say when we are in state s, we decide to take action a to arrive in the next state s' and obtain reward r. This is known as one *transition* step, represented by a tuple (s, a, s', r).

The transition function P records the probability of transitioning from state s to s' after taking action a while obtaining reward r. We use P as a symbol of "probability".

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a] \tag{2}$$

Thus the state-transition function can be defined as a function of $P(s', r|s, a)$:

$$P^a_{ss'} = P(s'|s, a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] = \sum_{r \in R} P(s', r|s, a) \tag{3}$$

The reward function R predicts the next reward triggered by one action:

$$R(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} P(s', r|s, a) \tag{4}$$

2.2 Policy

Policy, as the agent's behavior function , tells us which action to take in state s. It is a mapping from state s to action a and can be either deterministic or stochastic:

- Deterministic: $\pi(s) = a$
- Stochastic: $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$

2.3 Value Function

Value function measures how rewarding a state or an action is by a prediction of future reward. The future reward (called **return**), is a total sum of discounted rewards going forward. Let's compute the return Gt starting from time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0} \gamma^k R_{t+k+1} \tag{5}$$

The discounting factor $\gamma \in [0, 1]$ penalize the rewards in the future, because:

- The future rewards may have higher uncertainty
- The future rewards do not provide immediate benefits
- Discounting provides mathematical convenience; i.e., we don't need to track future steps forever to compute return.
- We don't need to worry about the infinite loops in the state transition graph.

The **state-value** of a state s is the expected return if we are in this state at time t, $S_t = s$:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{6}$$

Similarly, we define the **action-value** ("Q-value") of a state-action pair as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{7}$$

Additionally, since we follow the target policy π, we can make use of the probability distribution over possible actions and the Q-values to recover the state-value:

$$V_\pi(s) = \sum_{a \in A} Q_\pi(s, a)\pi(a|s) \tag{8}$$

The difference between action-value and state-value is the action *advantage* function ("A-value"):

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{9}$$

2.4 Optimal Value and Policy

The optimal value function produces the maximum return:

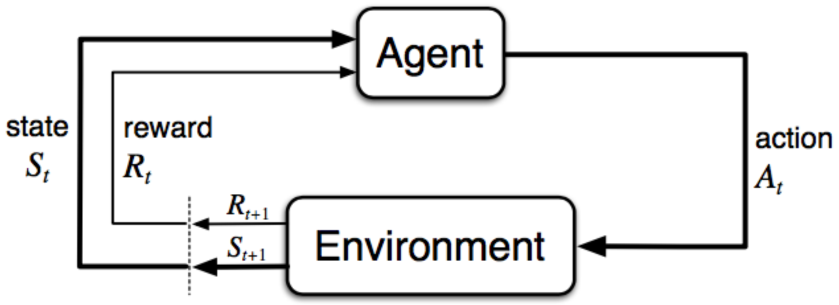
$$V_*(s) = \max_\pi V_\pi(s), Q_*(s, a) = \max_\pi Q_\pi(s, a) \tag{10}$$

The optimal policy achieves optimal value functions:

$$\pi_* = \operatorname*{argmax}_\pi V_\pi(s), \pi_* = \operatorname*{argmax}_\pi Q_\pi(s, a) \tag{11}$$

And of course, we have $V_{\pi_*}(s) = V_*(s)$ and $Q_{\pi_*}(s, a) = Q_*(s, a)$.

Figure 3: The agent-environment interaction in a Markov decision process. (Image source: Sec. 3.1 Sutton Barto (2017).)



3 Markov Decision Processes

In more formal terms, almost all the RL problems can be framed as Markov Decision Processes (MDPs). All states in MDP has "Markov" property, referring to the fact that the future only depends on the current state, not the history:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{12}$$

Or in other words, the future and the past are conditionally independent given the present, as the current state encapsulates all the statistics we need to decide the future. A Markov decision process consists of five elements $M = (S, A, P, R, \gamma)$,where the symbols carry the same meanings as key concepts in the previous section, well aligned with RL problem settings:

- S - a set of states;
- A - a set of actions;
- P - transition probability function;
- R - reward function;
- γ - discounting factor for future rewards. In an unknown environment, we don't have perfect knowledge about P and R

4 Bellman Equations

Bellman equations refer to a set of equations that decompose the value function into the immediate reward plus the discounted future values.

$$V(s) = \mathbb{E}[G_t|S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}|S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3})|S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$$

Similarly for Q-value,

$$Q(s,a) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}|S_t = s, A_t = a]$$

$$\mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a\sim\pi} Q(S_{t+1}, a)|S_t = s, A_t = a]$$

4.1 Bellman Expectation Equations

The recursive update process can be further decomposed to be equations built on both state-value and action-value functions. We extend V and Q alternatively by following the policy π.

$$V_\pi(s) = \sum_{a\in A} \pi(a|s) Q_\pi(s,a)$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} P^a_{ss'} V_\pi(s')$$

$$V_\pi(s) = \sum_{a\in A} \pi(a|s)(R(S,a) + \gamma \sum_{s'\in S} P^a_{ss'} V_\pi(s'))$$

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'\in S} P^a_{ss'} \sum_{a'\in A} \pi(a'|s') Q_\pi(s',a')$$

4.2 Bellman Optimality Equations

If, instead of computing the expectation following a policy, we are interested only in the optimal value, we can go ahead with the maximum returns during the alternative updates and use a policy.

$$V_*(s) = \max_{a\in A} Q_*(s,a)$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s'\in S} P^a_{ss'} V_*(s')$$

$$V_*(s) = \max_{a\in A} (R(s,a) + \gamma \sum_{s'\in S} P^a_{ss'} V_*(s'))$$

$$Q_*(s,a) = R(s,a) + \gamma \sum_{s'\in S} P^a_{ss'} \max_{a'\in A} Q_*(s',a')$$

These equations look very similar to Bellman expectation equations.

If we have complete information of the environment, this turns into a planning problem, solvable by DP. Unfortunately, in most scenarios, we do not know $P^a_{ss'}$ or $R(s,a)$, so we cannot solve MDPs by directly applying Bellmen equations. But, Bellman Equations lay the theoretical foundation for many RL algorithms.

5 Known Problems

5.1 Exploration-Exploitation Dilemma

When the RL faces an unknown issue, we are faced with the following dilemma: without enough exploration, we cannot learn the environment well enough; without enough exploitation, we cannot complete our reward optimization task. This is known as the exploration-exploitation dilemma.

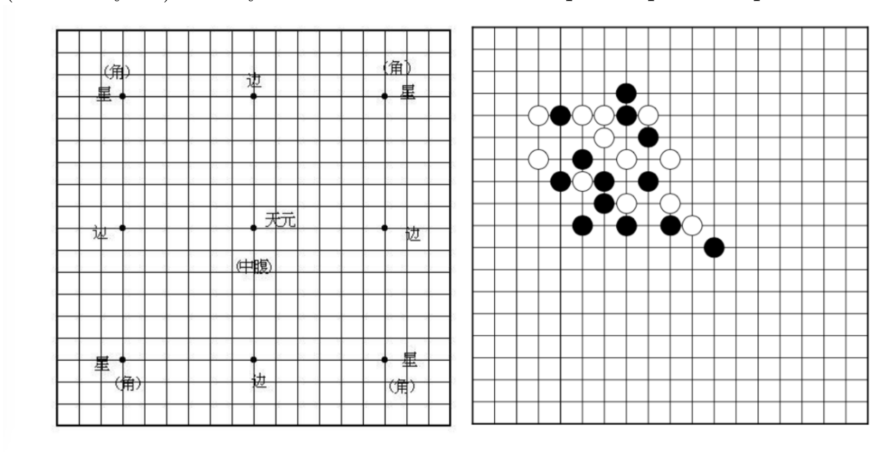
Different RL algorithms balance between exploration and exploitation in different ways. In MC methods, Q-learning or many on-policy algorithms, the exploration is commonly implemented by -greedy; In ES, the exploration is captured by the policy parameter perturbation.

5.2 Deadly Triad Issue

When off-policy, nonlinear function approximation, and bootstrapping are combined in one RL algorithm, the training could be unstable and hard to converge. This issue is known as the deadly triad (Sutton Barto, 2017). Many architectures using deep learning models were proposed to resolve the problem, including DQN to stabilize the training with experience replay and occasionally frozen target network.

6 Case Study: AlphaGo Zero

Figure 4: The board of Go. Two players play black and white stones alternatively on the vacant intersections of a board with 19 x 19 lines. A group of stones must have at least one open point (an intersection, called a "liberty") to remain on the board and must have at least two or more enclosed liberties (called "eyes") to stay "alive". No stone shall repeat a previous position.



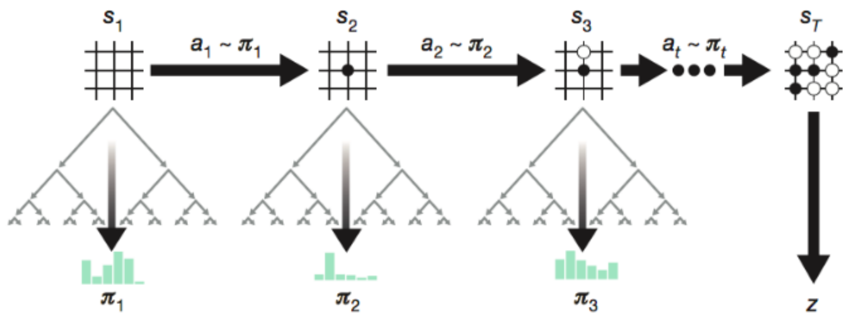
The game of Go has been an extremely hard problem in the field of Artificial Intelligence for decades until recent years. DeepMind developed two programs, AlphaGo and AlphaGo Zero. Both involve deep Convolutional Neural Networks (CNN) and Monte Carlo Tree Search (MCTS) and both have been approved to achieve the level of professional human Go players. Different from AlphaGo that relied on supervised learning from expert human moves, AlphaGo Zero used only reinforcement learning and self-play without human knowledge beyond the basic rules.

With all the knowledge of RL above, let's take a look at how AlphaGo Zero works. The main component is a deep CNN over the game board configuration (precisely, a ResNet with batch normalization and ReLU). This network outputs two values:

$$(p, v) = f_\theta(s) \tag{13}$$

- s: the game board configuration, 19 x 19 x 17 stacked feature planes; 17 features for each position, 8 past configurations (including current) for the current player + 8 past configurations for the opponent + 1 feature indicating the color (1=black, 0=white). We need to code the color specifically because the network is playing with itself and the colors of current player and opponents are switching between steps.
- p: the probability of selecting a move over $19^2 + 1$ candidates (19^2 positions on the board, in addition to passing).
- v: the winning probability given the current setting.

Figure 5: AlphaGo Zero is trained by self-play while MCTS improves the output policy further in every step. (Image source: Figure 1a in Silver et al., 2017).



During self-play, MCTS (Monte Carlo Tree Search) further improves the action probability distribution $\pi \sim p(.)$ and then the action a_t is sampled from this improved policy. The reward z_t is a binary value indicating whether the current player eventually wins the game. Each move generates an episode tuple (s_t, π_t, z_t) and it is saved into the replay memory.

The network is trained with the samples in the replay memory to minimize the loss:

$$L = (z - v)^2 - \pi^T log(p) + c||\theta||^2 \tag{14}$$

where c is a hyperparameter controlling the intensity of L2 penalty to avoid overfitting.

AlphaGo Zero simplified AlphaGo by removing supervised learning and merging separated policy and value networks into one. AlphaGo Zero outperformed AlphaGo with much shorter training time.