VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE

Course Work

Game playing using reinforcement learning

(Žaidimų žaidimas naudojantis skatinamuoju mokymu)

Done by: 3rd course, 4 group student
Arnoldas Čiplys (signature)

Supervisor:
Andrius Grevys (signature)

Vilnius
2020

Table of contents

Introduction

Reinforcement learning (abbreviated as RL) is an area of machine learning that learns what to do to maximize a numerical reward signal by not being told what actions to take, but instead by trial and error to discover which actions yield the most reward.

The advantage of reinforcement learning versus other machine learning areas is that it does not need any training data beforehand and learns as it goes along, similar to how humans or animals would learn to handle given tasks.

My focus in this topic will be analyzing, discussing and comparing different reinforcement learning algorithms, their types and strengths, including how reinforcement learning is used for AI learning in complex modern games.

1. Introduction to reinforcement learning

Reinforcement learning as we know today came around in the 1980s by combining 2 different threads. First was learning by trial and error, which originated studying psychology of animal learning, where animals would choose different actions depending on previous experiences, which had different rewards or punishments for doing certain actions. Edward Thorndike called this the "Law of Effect" because it describes the effect of reinforcing events on the tendency to select actions. Second was the problem of optimal control and its solution using value functions and dynamic programming. This thread mostly did not involve any learning. First approaches to this problem was by Richard Bellman in the 1950s using optimal return function to define a functional equation, now often called the Bellman equation. The class of methods solving optimal control by using mostly this equation came to be called dynamic programming and is essential in reinforcement learning theory. [SB18]

Reinforcement learning represents an agent's attempt to approximate the environment's function, such that we can send actions into the black-box environment that maximize the rewards it spits out. [Nic20]

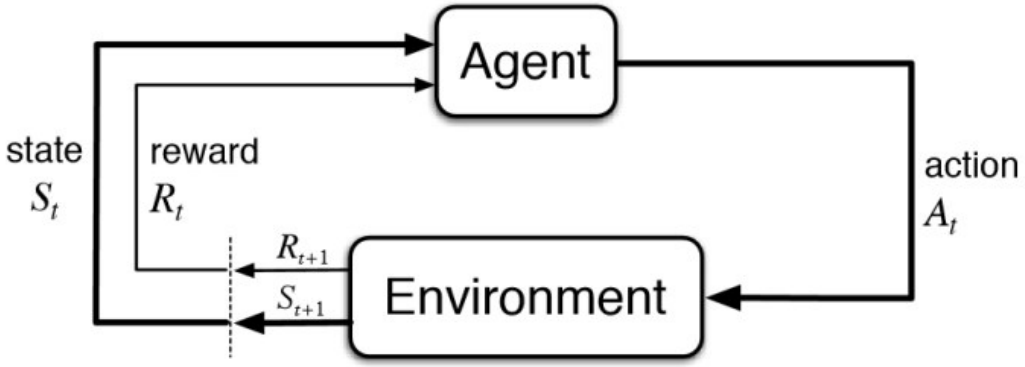

Figure 1. Basic reinforcement learning model. [Bha18]

Different algorithms use different techniques to calculate reward function and from that depends how long the agent will need to train in order to achieve success.

2. Algorithm characterizations

Reinforcement learning algorithms usually follow some characteristics that are important to know to compare and analyze specific algorithms.

2.1. Model-Based vs Model-Free

Model-Based reinforcement learning uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Actions are chosen by searching or planning in this world model.

Model-Free reinforcement learning uses experience to learn directly one or both of state/action values or policies which can achieve the optimal behavior without estimation or use of a world model.

Model-free methods are statistically less efficient than model-based methods, because information from the environment is combined with previous, and possibly erroneous, estimates or beliefs about state values, rather than being used directly. [Mon19]

2.2. On-policy vs off-policy

On-policy methods attempt to evaluate or improve the policy that is used to make decisions. That means that the agent plays and improves only from himself and tries to improve his actions right now.

Off-policy methods attempt to evaluate or improve a policy different from that used to generate the data. Agent does not assume that the sessions he's obtained or trained are from the same place as one he's playing right now. That means training can be done in a simpler or cheaper to train environment, where the agent can explore and learn more before going to the real one. [SP20]

2.3. Discrete vs continuous vs parameterized action space

In discrete action space, the agent decides which distinct action to perform from a finite action set.

In continuous action space, actions are expressed as a single real-valued vector since the action set is infinite.

Parameterized action state combines advantages of discrete and continuous by treating different kinds of continuous actions as distinct. At each step the agent chooses both which action to use and what parameters to execute it with. [MRK16]

3. Basic algorithms

Simple environments like old games do not have many variables at play, usually just a few basic controls, like move, jump or use ability. These environments can be explored and perfect using basic reinforcement learning algorithms that have non complicated steps and help to understand basic important reinforcement learning concepts.

We are going to analyze few of them:

- Monte carlo methods
- Q-learning
- TD learning

3.1. Monte Carlo methods

Monte Carlo methods are a group of one of the most simple algorithms used for reinforcement learning, where an agent does not have complete knowledge of the environment, but only requires experience from actual or simulated interaction with the environment.

Monte Carlo methods solve reinforcement learning problems based on averaging sample returns. The term “Monte Carlo” is often used for any estimation method where operation involves a lot of randomness. [SB18]

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$

Figure 2. First-visit Monte Carlo prediction pseudocode. [SB18]

3.2. Q-learning

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.

The ‘q’ in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. [Vio19]

Q-learning uses matrix called q-table of size states * actions:

$$Q : S \times A \to \mathbb{R}$$

This matrix is used as a reference table to select the best action based on found value.

Algorithm:

1. Initialize q-table to all 0.
2. Agent selects action from:
   2.1. Exploit. Reference q-table and select best possible action from your current state.
   2.2. Explore. Move randomly. This helps to discover new states that could not be discovered otherwise.
3. Update q-table using formula:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}\Big)}^{\text{temporal difference}}$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

Figure 3. Q-table value calculation formula. [Bha18]

where:

- Learning rate - determines to what extent newly acquired information overwrites old information. A factor of 0 makes the agent learn nothing, while a factor of 1 makes the agent consider only the most recently acquired information.
- Discount factor - determines importance of future rewards. A factor of 0 will make the agent consider only current rewards, while a factor approaching 1 will consider long term rewards.

3.2.1. Deep Q-learning

Q-learning is a simple yet quite powerful algorithm to create a cheat sheet for our agent to help figure out exactly which action to perform. However in an environment with many states and actions per state, q-table could be enormous and training would be insanely long. In deep Q-learning, neural networks are used to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. [Cho19]

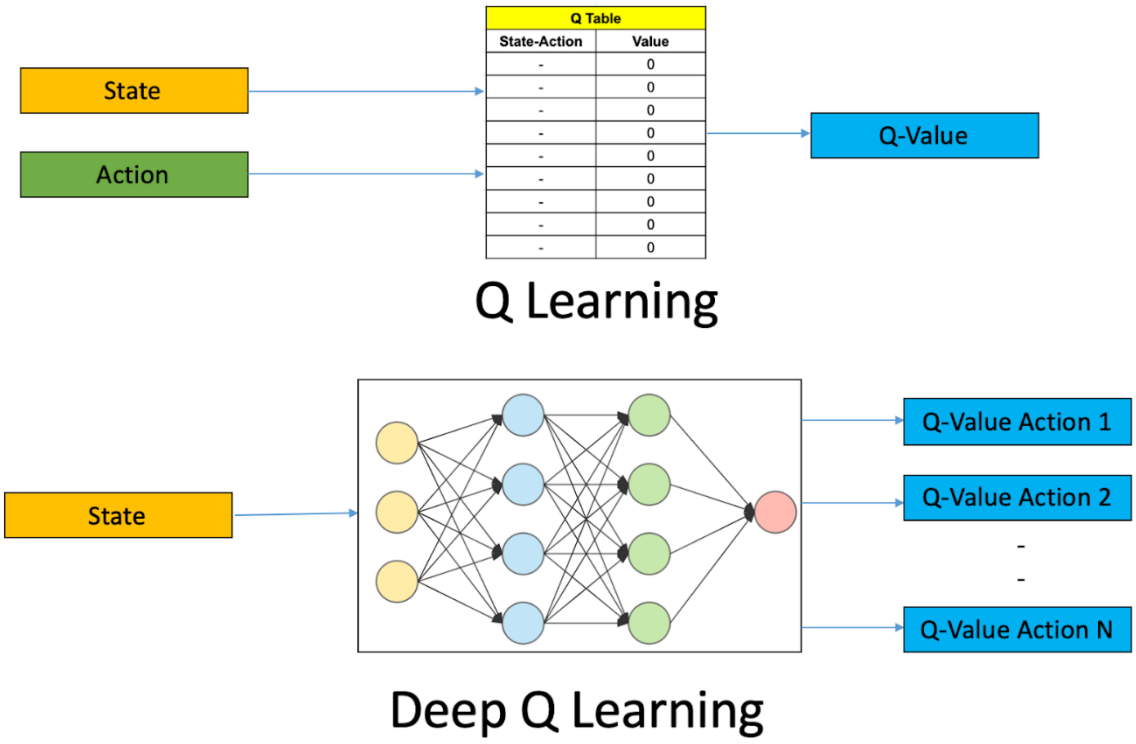

Figure 4. The comparison between Q-learning & Deep Q-learning. [Cho19]

3.2.2. Double Q-learning

In some stochastic environments Q-learning performs very poorly. This poor performance is caused by large overestimations of action values. These overestimations result from a positive bias that is introduced because Q-learning uses the maximum action value as an

approximation for the maximum expected action value. Double Q-learning introduce an alternative way to approximate the maximum expected value for any set of random

variables. The obtained double estimator method is shown to sometimes underestimate rather than overestimate the maximum expected value. [Has10]

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right)$$

Figure 5. Double Q-learning q-table calculation formula. [Has10]

3.3. Temporal-Difference learning

Temporal-difference (TD) learning is a combination of Monte Carlo and dynamic programming ideas. It is a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function. Temporal-difference methods sample from the environment, similar to Monte Carlo methods and perform updates based on current estimates, similar to dynamic programming methods. [SB18]

A simple every-visit Monte Carlo method suitable for nonstationary environments is

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

where Gt is the actual return following time t , and α is a constant step-size parameter. Monte Carlo methods such as this must wait until the end of the episode to determine the increment to V(St), meanwhile TD methods only need to wait until the next time step. At time t+1 they immediately form a target and make a useful update using the observed reward Rt+1 and the estimate V(St+1). Simplest TD method, called TD(0) or one-step TD makes the update immediately on transition to St+1 and receiving Rt+1:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

[SB18]

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 6. TD(0) pseudocode. [SB18]

Conclusions

Whole concept of reinforcement learning has many advantages over different problem solving techniques, such as the ability to achieve long-term results, error correction occurred in the training process, power to learn without a dataset and only with its own experience. However reinforcement learning shouldn't always be used, since it has some cons. Using it for simple problems is overkill and might lead to much longer processes and worse results than using simple AI. Reinforcement learning assumes that the world is Markovian, which it is not. And using it in the real world might be too expensive, like using it with real robots, since this technique has to fail to learn.

Analyzed reinforcement learning algorithms also have their own advantages and disadvantages against one another. To summarize them, they can be grouped by previously explained characterizations:

Algorithm	Model	Policy	Action space
Monte Carlo	Model-Free	Either	Discrete
Q-learning	Model-Free	Off-policy	Discrete
TD learning	Model-Free	Either	Discrete

All of these basic algorithms are very universal and could work in almost any case. One flaw is that if your environment has continuous action space, different, more complex algorithms or variations of these algorithms would be needed.

Between algorithms themselves, TD learning methods is basically an upgrade to Monte Carlo methods by having an ability to learn every step instead of a whole episode. Q-learning is a more complex off-policy TD learning method than TD(0) only used to learn q-function. If reinforcement learning alone is not enough, Q-learning is great for combining it with different types of machine learning, like neural networks (in deep Q-learning) to help the agent learn quicker.

4. Complex algorithms

For more complex problems/games, simple previously analyzed algorithms might not be enough. Usually there are too many choices, no clear reward system and other limitations that makes learning too hard. To solve this, developers often use combinations of different machine learning techniques to allow reinforcement learning strengths come faster. We are going to be looking at few examples of how few different teams used reinforcement learning to train AI for modern complex games:

- Alphastar for Starcraft 2 (by DeepMind)
- OpenAi for Dota 2
- AlphaZero for chess, go (by DeepMind)

4.1. AlphaStar for Starcraft 2

Starcraft 2 is a real-time strategy (RTS) video game that focuses on balancing base expansions, army training, attacking, defending to win against your opponent by destroying all their structures. DeepMind created AI called AlphaStar in January 2019 and continued to train it and by August 2019 after it was allowed to play with real human players, it managed to reach Grandmaster rank on 1v1 European ladder, meaning it was amongst 0.2% of human players.

Starcraft 2 is incredibly hard game for an AI to play for a few reasons:

- There is no single winning strategy, the agent has to balance exploration, building the army, expanding the economy and countless other things.
- Game has imperfect information and the agent has to scout enemy player to see what actions they are doing to prepare counters for them.
- There is no clear reward and the only important thing is winning the game, so the agent has to decide what leads to that victory.
- Starcraft 2 maps are incredibly large compared to other games like chess or go where boards are 8x8 or 19x19 respectively. Here maps span over several screens and each map will behave differently.

[Alp19b]

To handle such difficulties, AlphaStar training process was as follows:

1. Use traditional supervised learning leveraging a dataset of anonymized human games released by Blizzard (developers of Starcraft 2).
2. Create a multi-agent reinforcement learning environment in which multiple variations of agents will play against themselves.

[Alp19b]

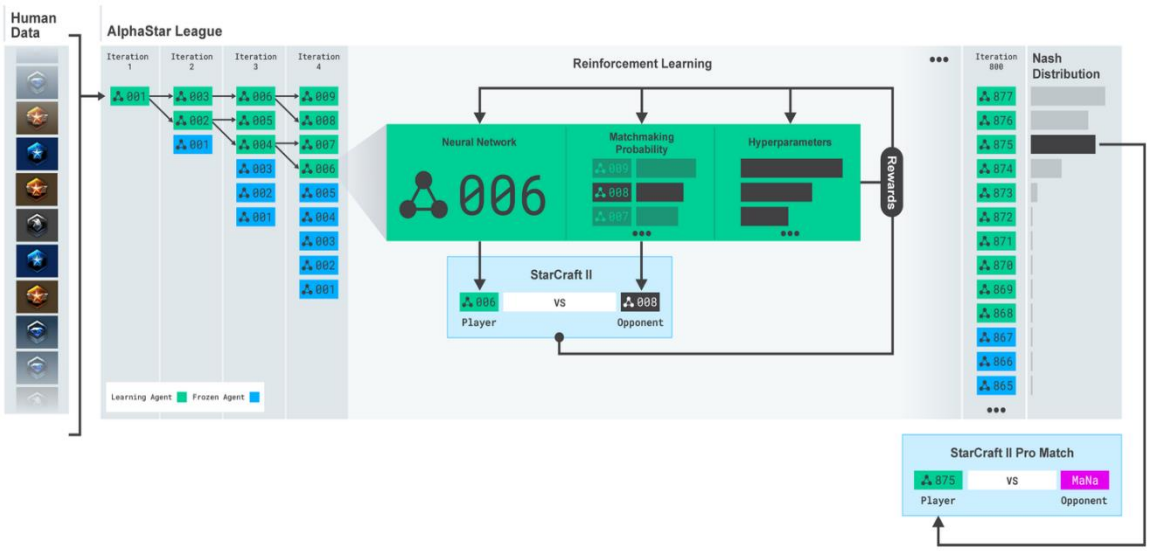

Figure 7. AlphaStar learning process. [Alp19b]

However, after a long time of training like that, the team realised that training like that was insufficient and AlphaStar stopped improving. Starcraft 2 often has strategies that are direct counters to specific strategies (think of rock-paper-scissors) and to force AlphaStar to explore more different strategies, exploiters, different versions of bot were added to execute common specific strategies for main bot to train on. [Alp19a]

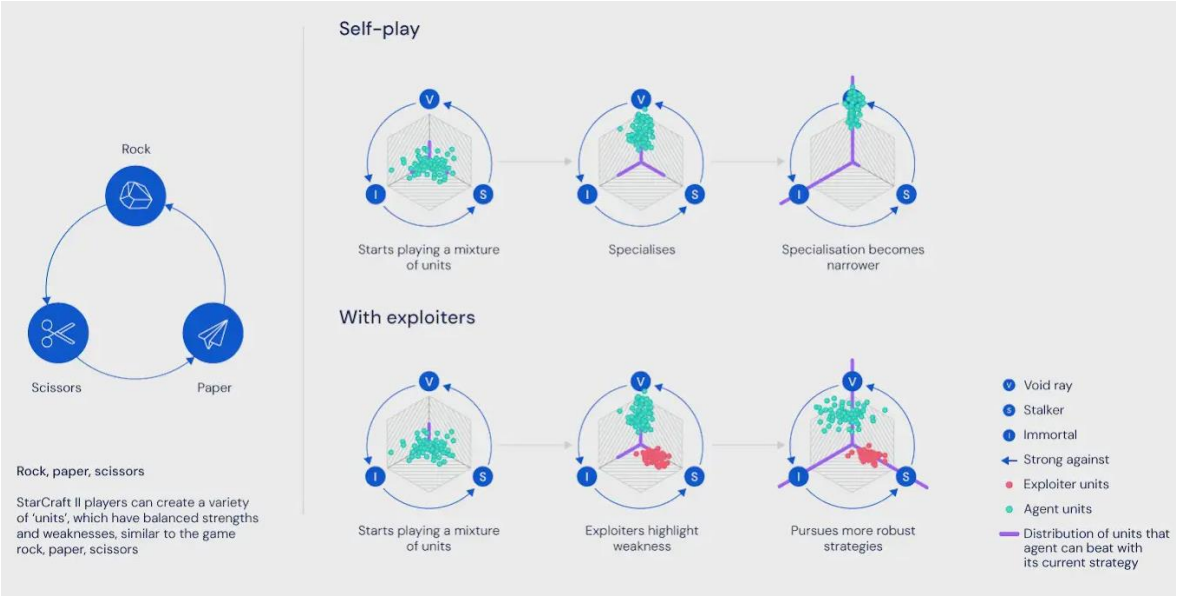

Figure 8. Introducing exploiters. [Alp19a]

After that, further using reinforcement learning against itself and copies of itself that uses only specific strategy, AlphaStar was able to reach the highest level.

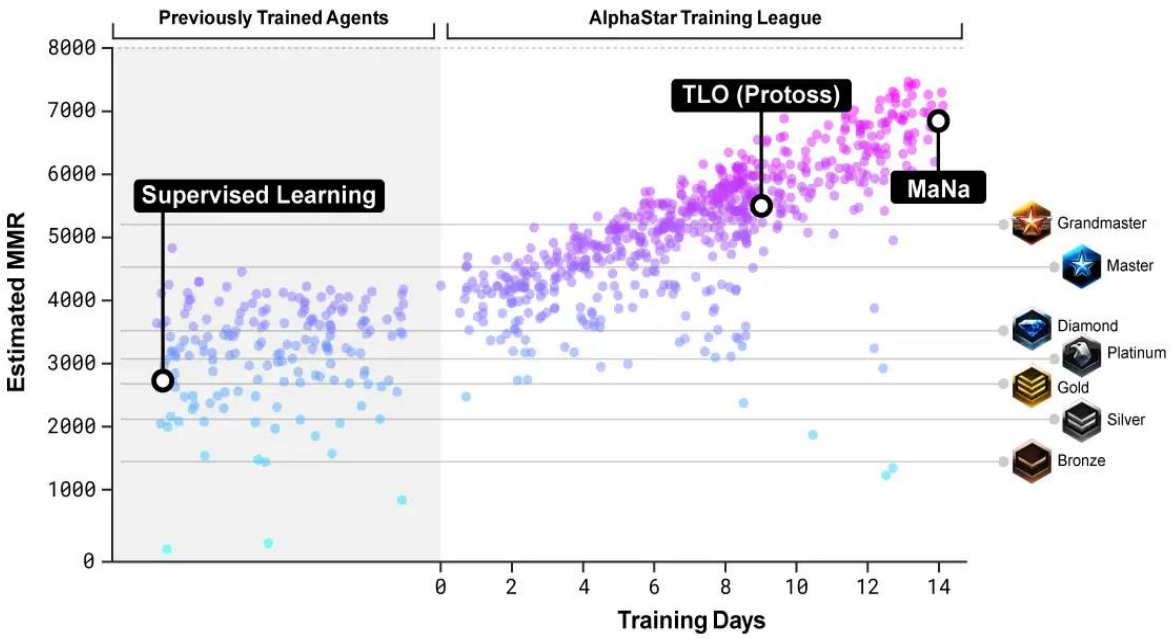

Figure 9. AlphaStar rank climb. [Alp19b]

4.2. OpenAI for Dota 2

Dota 2 is a multiplayer online battle arena (MOBA) video game, where teams of 5 players fight against other teams to destroy each other's bases. Individual skill of the players is a significant factor, however, teamwork is usually the most important aspect of the outcome of the match and that is incredibly challenging for AI to understand and learn. OpenAI created Dota 2 playing agent called OpenAI Five and in april 2019, agent defeated the Dota 2 world champions (Team OG). Afterwards OpenAI Five were opened to play against Dota 2 community and they won 99.4% of over 7000 games.

Dota 2, similar to previously mentioned Starcraft 2 is incredibly complex and has many challenges for AI to face:

- Amount of moves the agent have to do is approximately over 20000 per match, whereas chess usually lasts only around 80.
- Dota 2 has a very large map, dozens of buildings, 10 heroes, dozens of non-player units that are constantly respawning and interacting with each other, a lot of game features such as trees, wards, runes
- There is always imperfect information, as each team can only see a portion of the map near their units and buildings

However, there are some limitations for OpenAI Five:

- Subset of 17 heroes - in the normal game players are able to choose from over 100 unique heroes but the agent only supports 17 of them
- No control of multiple units - in the game there are several items that allow player to control multiple units at the same time (Illusion Rune, Helm of the Dominator, Manta Style, and Necronomicon) that are removed to avoid more technical complexity

To train OpenAI policy, an algorithm called Proximal Policy Optimization (PPO) is used. The optimization algorithm uses Generalized Advantage Estimation (GAE), a standart advantage-based variance reduction technique to stabilize and accelerate training. [Ope19]

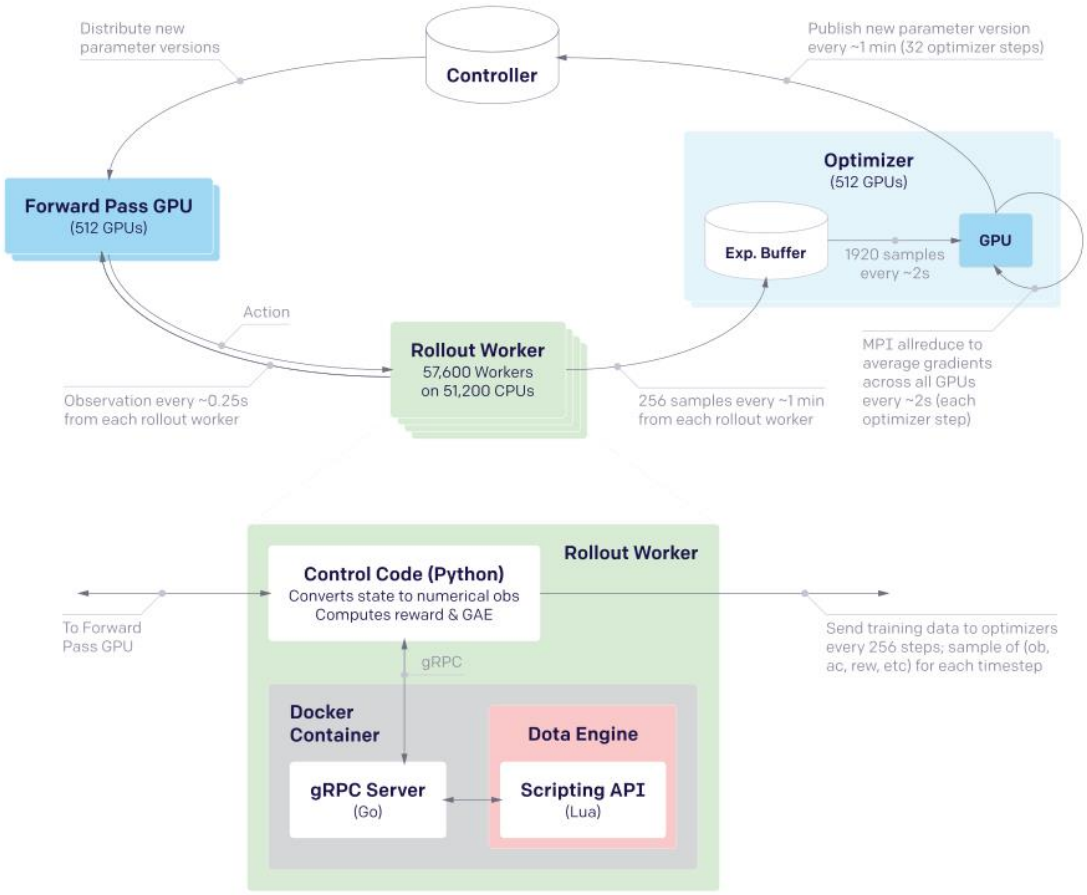

Figure 10. PPO training system. [Ope19]

Each of OpenAI Five's networks contain a single-layer, 1024 unit Long Short Term Memory network that sees the current game state and emits actions through several possible action heads. [Ope19]

Figure 11. Observation space. [Ope18]

OpenAI Five learns from self-play (starting from random weights), which provides a natural curriculum for exploring the environment. To avoid “strategy collapse”, the agent trains 80% of its games against itself and the other 20% against its past selves. To train efficiently, exploration is incredibly important. Due to the game's structure, it's not very hard to define a good reward for the agent. Reward consists mostly of metrics humans track to decide how they’re doing in the game: net worth, kills, deaths, assists, last hits. Each agent’s reward post processed by subtracting the other team’s average reward to prevent the agents from finding positive-sum situations. At the start, items and skill builds were hardcoded and chosen at random to not overwhelm agents. [Ope18]

4.3. AlphaZero for Chess

Chess is a very deeply explored game and already a lot of world class chess bots have been created, such as Stockfish or IBM's Deep Blue that rely on thousands of rules and heuristics handcrafted by strong human players that try to account for every eventuality in a game. However, AlphaZero after only 4 hours of training could already outperform them. [SHS+18a]

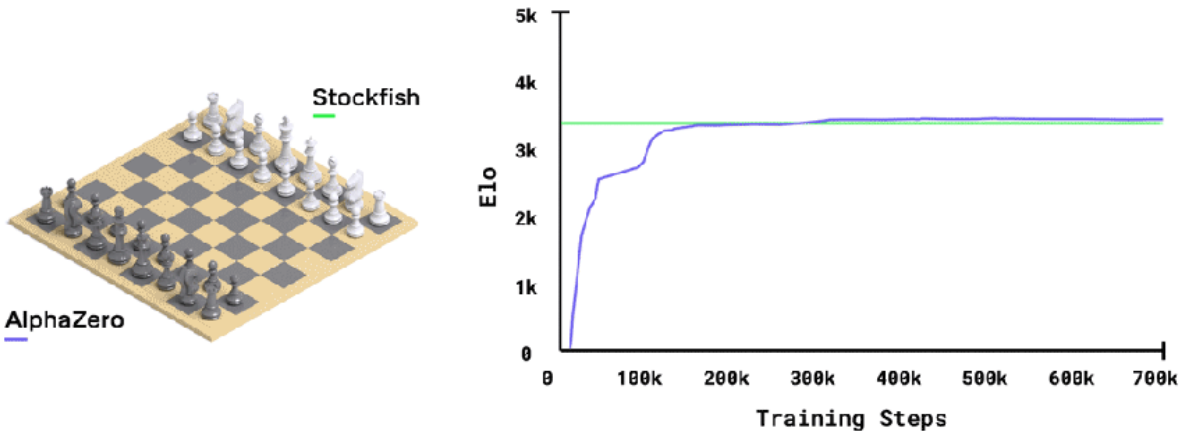

Figure 12. AlphaZero training. [SHS+18a]

AlphaZero uses a general-purpose Monte Carlo tree search (MCTS) algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root state until a leaf state is reached. Each simulation proceeds by selecting in each state a move with low visit count (not previously frequently explored), high move probability, and high value according to the current neural network. The search returns a vector representing probability distribution over moves. The parameters of the deep neural network are trained by reinforcement learning from self-play games, starting from random parameters. That leads to much faster speeds than best chess engines because the amount of moves searched are much smaller in size. [SHS+18b]

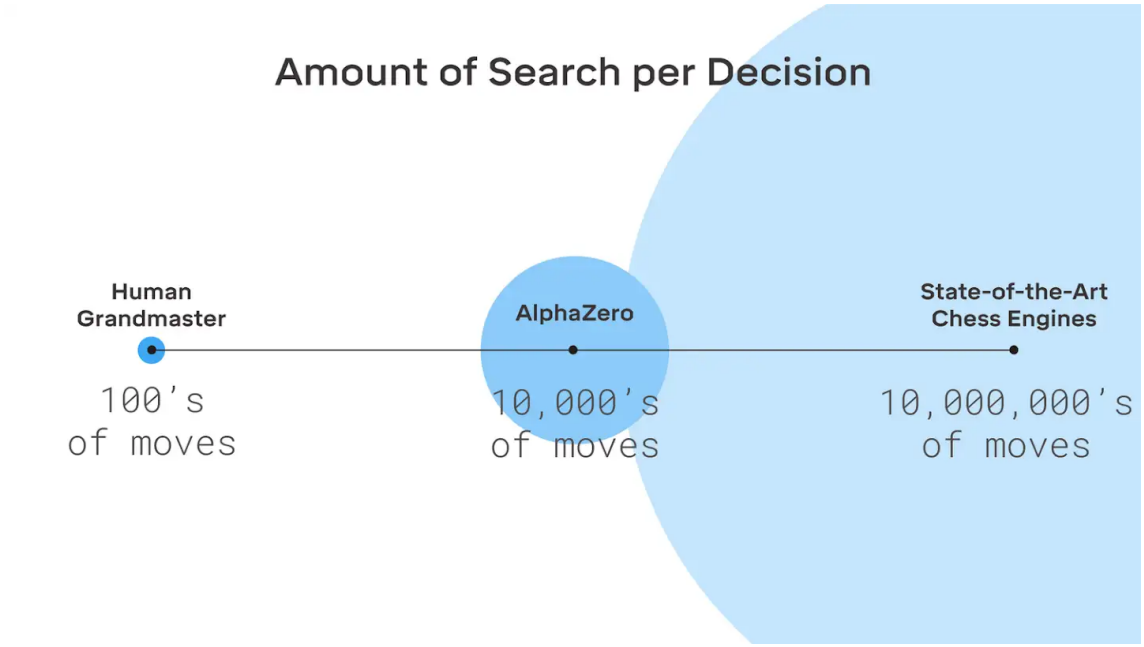

Figure 13. Move consideration. [SHS+18a]

Conclusions

While reinforcement learning is incredibly good at training, when there are too many choices for the agent to make, like in these scenarios, where the environment and action count is very large, training would take too much time to be practical. In these scenarios combining reinforcement learning with other machine learning techniques might speed up the process. In these examples different kinds of neural networks are used for making moves and reinforcement learning is used for training these neural networks, where only several layers with nodes exist and training grounds for reinforcement learning becomes much smaller.

Additional problem we could see is agents stuck in plateau and not learning anymore and teams solving it using different exploration methods, like playing with previous and not current copy of agent, introducing specific enemies for the agent to learn to counter specific strategies etc.

References

[Alp19a] The AlphaStar team, AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning, https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning. 2019.

[Alp19b] The AlphaStar team, AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii. 2019.

[Bha18] S. Bhatt, 5 Things You Need to Know about Reinforcement Learning, https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html. 2018.

[Cho19] A. Choudhary, A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python, https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python. 2019.

[Has10] H. van Hasselt, Double Q-learning, https://papers.nips.cc/paper/3964-double-q-learning. 191KB, 2010.

[Mon19] R. Moni, Reinforcement Learning algorithms — an intuitive overview, https://medium.com/@SmartLabAI/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc. 2019.

[MRK16] W. Masson, P. Ranchod and G. Konidaris, Reinforcement Learning with Parameterized Actions, https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/11981/11825. 1112KB, 2016.

[Nic20] C. Nicholson, A Beginner's Guide to Deep Reinforcement Learning, https://pathmind.com/wiki/deep-reinforcement-learning. 2020.

[Ope18] OpenAI team, OpenAI Five, https://openai.com/blog/openai-five. 2018.

[Ope19] OpenAI team, Dota 2 with Large Scale Deep Reinforcement Learning, https://cdn.openai.com/dota-2.pdf. 8345KB, 2019.

[SB18] R. S. Sutton and A .G. Barto. Reinforcement Learning, An Introduction (second edition), https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf. 4059KB, 2018.

[SHS+18a] D. Silver, T. Hubert, J. Schrittwieser, D. Hassabis, AlphaZero: Shedding new light on chess, shogi, and Go, https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go. 2018.

[SHS+18b] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, https://science.sciencemag.org/content/362/6419/1140/tab-pdf. 3822KB, 2018.

[SP20] P. Shvechikov and A. Panin, On-policy vs off-policy; Experience replay, https://www.coursera.org/lecture/practical-rl/on-policy-vs-off-policy-experience-replay-YdFw3. 2020.

[Vio19] A. Violante, Simple Reinforcement Learning: Q-learning, https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56. 2019.