Overview of selected reinforcement learning solutions to several game theory problems

R. JAROSZ

robert.jarosz@wat.edu.pl

Military University of Technology, Faculty of Cybernetics
Kaliskiego Str. 2, 00-908 Warsaw, Poland

This paper collects several applications of reinforcement learning in solving some problems related to game theory. The methods were selected to possibly show variety of problems and approaches. Selections includes Thompson Sampling, Q-learning, DQN and AlphaGo Zero using Monte Carlo Tree Search algorithm. Paper attempts to show intuition behind proposed algorithms with shallow explaining of technical details. This approach aims at presenting overview of the topic without assuming deep knowledge about statistics and artificial intelligence.

Keywords: artificial intelligence, game theory, Thompson sampling, Q-learning, DQN, Monte Carlo tree search, AlphaZero.

DOI: 10.5604/01.3001.0053.9698

1. Introduction

Reinforcement learning is one of three major areas in machine learning – alongside supervised learning and unsupervised learning. A factor distinguishing these three fields is how the process of learning is conducted, including how data is provided to algorithm and how the solution is assessed.

Supervised learning utilises maps input data to output data e.g. images of animals to labels representing species. The algorithm assigns labels to data, comparing it's solution to provided by supervisor (human). Unsupervised learning does not depend on labels and categories data by extracting traits in data samples. During learning algorithm adjust considered traits, trying to select these which provide the most accurate categorization. While these two focus usually on analysing (or sometimes synthesizing) data samples, reinforcement learning focuses on optimised choosing action steps leading to solution with maximalised cumulative reward. A model learns by approaching dynamic problem in a series of experiments and receiving assessment for each experiment. Typically each such experiment is composed with a number of actions, which usually are also assessed.

This approach has found application in creating artificial intelligence aimed at solving challenging problems like playing games or driverless steering machines. Algorithm learns solution by playing a particular game a number of times, each time assessing it's result and applying gained experience for following games. In this paper a view on some selected problems paired with reinforcement learning solution approach will be presented.

2. Scope dictionary

This paper focuses on appliance of reinforcement learning in solving game theory problems [1], [2].

To avoid confusion in the understanding certain words their description is provided in the list below. Whenever the word appears in cursive font it should be understood in this particular way, otherwise the meaning is context specific:

- **game** – a single independent competition of one or more players. The subsequent games are independent from previous, meaning that score and final state of past game does not influence current – no assets are moved between games. For example in chess the *game* starts with the first move of white figures and ends with mate, surrender or draw.
- **player** – an entity taking part in the *game*, that has an ability to influence the state of it by making decisions defined by rules of the game.

- **world** – a special entity that is not controlled by human or computer players. The entity is used to model randomness in game. For example if a *player* makes a dice roll, then score of the roll is not in his might to decide, instead the *world* states the value rolled.
- **move** – a single decision made by player in scope of the *game* – like moving a single figure or playing single card.
- **turn –** a moment in the *game* when *player* makes one or a series of *moves*. For example a chess *game* consists of alternating *turns* of white and black player (in chess player makes exactly one *move* per *turn*).

3. General learning model

In scope of reinforcement learning the following entities are defined:

- **agent** – the entity is intended to solve problems it is initially trained for this purpose, in problems of game theory *agent* is commonly a *player*;
- **environment** – physical or simulated surrounding reacting to *actions* performed by *agent* providing the *agent* observable information about state of experiment and assessment of it's performance;
- **action** – an activity performed by *agent* impacting the *environment*;
- **reward** – value assessing performed *action*, passed from *environment* to *agent*, *reward* is expected to be comparable and summable, therefore usually it is a numeric value;
- **observation** – data that is presented to the *agent* from *environment* about it's current state, this information may be partial or full.

The process of reinforcement learning consists of *agent* taking consecutive *actions*, influencing the *environment*. After the *action* is performed the *environment* reacts to it by changing the current state and providing *reward* and *observation* (information about *state*) for the *agent*.

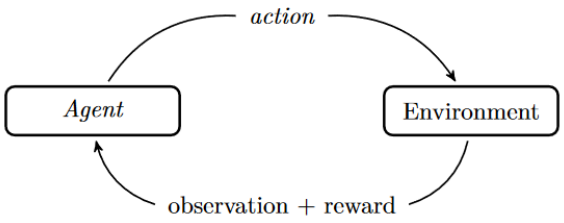

Fig. 1. Model scheme of reinforcement learning

Learning algorithm's objective is to score best possible outcome at the end of the *game*. In some *games* it will be accumulated rewards for all the actions, in some the final result can be disconnected from partial *rewards* – like in chess where no matter the course of *game* it ends with win, lose or draw. Reinforcement learning finds it's usage when the number of policies in game is too big for exhaustive search or designing a procedural strategy is significantly complex process. Program can play many more *games* than human in time unit to collect experience. The core problem of designing self-learning algorithm is optimizing *exploration-exploitation* trade-off. *Exploitation* is following already known policy and *exploration* is searching for a better policy. The balance between exploring and exploiting policies is important for upgrading performance over time.

4. Thompson Sampling – problem

Multi-armed bandit problem (MAB) – a single player game in which *player* is presented a slot machine with N-arms. The game lasts for T *turns* and each *turn player* must choose one of the arms to pull. Each pulled arm immediately yields a random rational reward from set $\langle 0,1 \rangle$. The distribution of rewards is not known for any of arms, however it does not change during *game*. After T *turns game* ends. MAP problem is widely discussed in [3].

Player does not know which arm has the best reward distribution therefore it needs to learn which arm is the best one to be played.

To evaluate solution a function of *regret* is introduced. Denote expected reward of playing i-th arm as μ_i and expected reward from best arm: $\mu^* = \max_i \mu_i$. Let $k_i(t)$ be the number of times the i-th arm was played until (excluding) step t. Then the total expected *regret* is given:

$$\mathbb{E}\big(\mathcal{R}(T)\big) = \mathbb{E}\left[\sum_{t=1}^{T}\big(\mu^* - \mu_{i(t)}\big)\right] = \\ = \sum_i \big[(\mu^* - \mu_i) \cdot \mathbb{E}\big(k_i(T+1)\big)\big] \tag{1}$$

With random exploring picking arms with equal probabilities $\mathbb{E}\big(k_i(T+1)\big) = T/N$ and expected *regret* is equal:

$$\mathbb{E}\big(\mathcal{R}_{\text{rnd}}(T)\big) = \sum_{i=1}^{N}\left[(\mu^* - \mu_i) \cdot \frac{T}{N}\right] \tag{2}$$

Another naive approach would be to test each arm m-times and then picking the one appearing to be the best (assuming that mN is significantly smaller than T). With this approach the estimated *regret* is given:

$$\mathbb{E}\big(\mathcal{R}(T)\big) = m\sum_{i=1}^{N}[(\mu^* - \mu_i)] + \\ + (T - mN)\big(\mu^* - \mu_p\big) \tag{3}$$

where μ_p is selected arm for the rest of the game. This *regret* has it's minimum in case the best arm was selected after exploring phase: $\mu_p = \mu^*$.

5. Thompson Sampling – solution

Apart from these naive approaches exist some more sophisticated methods, like Thompson sampling described below. Thompson sampling is heuristic method that takes it's name after William R. Thompson whose work related to exploitation-exploration dilemma [4], [5]. Over time various researchers have been working on developing and formally proving regarding theorems. An example of such work is estimation of expected regret done by Agraval and Goyal [6].

The idea behind the method is to randomly choose arm to pull. The probability of arm selection depends on approximation of it's distribution. This section is strongly based on work presented in [6] and [7], notation and models are taken from [6], where method is elaborated in more detailed way. The spoken notation is defined:

1. Parameters $\tilde{\mu}$;
2. An assumed prior distribution with given density $P(\tilde{\mu})$ on parameters;
3. Past observations $\mathcal{D}$;
4. An assumed *likelihood* function $P(r|\tilde{\mu})$ – probability of gaining *reward* r, given the parameter $\tilde{\mu}$;
5. A posterior distribution $P(\tilde{\mu}|\mathcal{D}) \propto P(\mathcal{D}|\tilde{\mu})P(\tilde{\mu})$, where $P(\mathcal{D}|\tilde{\mu})$ is *likelihood* function.

The prior distribution is the assumption made by *agent* on how the *environment* works without backing it with evidence. Previous observations $\mathcal{D}$ is list including selected arms and reward they returned. Then $P(r|\mathcal{D})$ is *posterior distribution* (backed by evidence $\mathcal{D}$) and by Bayes theorem [8], [3] it is proportional to $P(\mathcal{D}|\tilde{\mu})P(\tilde{\mu})$. $P(\mathcal{D}|\tilde{\mu})$ is *likelihood* of making $\mathcal{D}$ observations, when parameters are $\tilde{\mu}$.

Although *agent* does not know the real chances of winning on each arm, it can estimate it's distribution parameters. Posterior distribution is the distribution of expected reward assumed by *agent*. *Agent* needs to select best arm to play, to do so it uses posterior distribution to sample μ for the arm, after comparing samples from each arm's posterior distribution, it selects the one which has returned largest sample.

Note 1: As in this section terminology from [6] had been adapted, the inconsistence with model described in section 3 of this article has occurred. Observation set $\mathcal{D}$ is list of tuples linking index of played arm and reward $\{(\mathrm{t}, \mathrm{i_t}, \mathrm{r_t})\}$. In terms of model from section 3 *reward* is separated entity to *observation*. In terms described in section 3 this set would be actually $\{(t, a, r)\}$ where a is action taken and r is *reward* granted. Beside returned reward there are no significant *observations*.

Note 2: The actual formula for $P(\tilde{\mu}|\mathcal{D})$ is given:

$$P(\tilde{\mu}|\mathcal{D}) = \frac{P(\mathcal{D}|\tilde{\mu})\,P(\tilde{\mu})}{P(\mathcal{D})} \tag{4}$$

however $P(\mathcal{D})$ is expensive to calculate (the law of total probability) and it is not needed here because during *game* in the moment of picking arm observations $\mathcal{D}$ does not change.

Bernoulli Bandit Problem is special case of MAB problem. The *rewards* are discrete – either 0 or 1. The probability of i-th arm to give reward of 1 (a chance of success) is μ_i.

Beta distribution makes good prior for Bernoulli Bandit Problem for if the parameters α, β denote accordingly observed successes and failures the prior distribution is $P(\tilde{\mu}) = Beta(\alpha, \beta)$. If $\alpha = \beta = 1$ then it is uniform distribution on (0,1), and it's expected *reward* is $1/2$ which is good assumption when no information has been gained. Let S_i be the number of previous successes while pulling i-th arm, and F_i be the number of failures caused by pulling i-th arm. The posterior distribution for i-th arm will be $Beta(S_i + 1, F_i + 1)$ [9], [6], [7].

If $X \propto Beta(\alpha, \beta)$ then:

$$\mathbb{E}(X) = \frac{\alpha}{\alpha + \beta} \tag{5}$$

so it follows the intuition that $\tilde{\mu_i} \propto \frac{S_i}{S_i + F_i}$ (in asymptotic term, as actually $\alpha = S_i + 1$ and $\beta = F_i + 1$. The variance is given:

$$V[X] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \tag{6}$$

and as S_i and F_i gets larger the variance is getting smaller (distribution gets narrow), therefore the more samples were drawn the more reliable is posterior distribution.

The procedure for playing *game* using Thompson Sampling is following:

```
S_i := 0; F_i := 0;
for t in 1..= T{
    for i in 1..= N {
    sample θ_{i,t} from Beta(S_i + 1, F_i + 1);
    }
    pick i := index of(max_i θ_i(t));
    play i-th arm;
    observe reward r_t
    if r_t == 1 {
        S_i := S_i + 1
    }
    else {
        S_i := S_i + 1
    }
}
```

Algorithm 1: Thompson Sampling using beta priori

In more general multi armed bandit problem rewards are distributed from real numbers in certain interval. As the *rewards* are now real one cannot consider them in terms of failure and success, therefore different prior distribution is needed.

Let $i(t)$ denote index of arm played at *turn t*; $r_i(t)$ denote reward of *i*-th arm at *turn t*, and $k_i(t)$ number of plays arm i until time t.

Define:

$$\hat{\mu}_{i(t)} = \sum_{w=1:\ i(w)=i}^{t-1} \frac{r_i(t)}{k_i(t)+1} \tag{7}$$

Then it is assumed that *likelihood* of reward $r_i(t)$ at *turn t* with parameter μ_i is given by probability density function of Gaussian distribution $\mathcal{N}(\mu_i, 1)$. Next the prior distribution is given $\mathcal{N}\left(\hat{\mu}(t), \frac{1}{k_i(t)+1}\right)$ and posterior distribution $P(\widetilde{\mu_i}|r_i(t)) \propto P(r_i(t)|\widetilde{\mu_i})P(\widetilde{\mu_i})$ given as $\mathcal{N}\left(\hat{\mu}_{i(t+1)}, \frac{1}{k_i(t+1)+1}\right)$. The algorithm of Thompson Sampling problem is given:

```
k_i := 0; μ̂_i := 0; I = [0; T]
for t in 1..=T {
    for i in 1..=N {
        Sample θ_i(t) ← N(μ̂_i(t), 1/(k_i(t+1)+1))
    }
    pick i(t):= index of (max_i θ_i(t));
    play i(t)-th arm;
    observe reward r_t
  set μ̂_{i(t)} := (μ̂_{i(t)} k_{i(t)} + r_t)/(k_{I(t)}+2);
  set k_{i(t)} := k_{i(t)} + 1;
}
```

Algorithm 2: Thompson Sampling using Gaussian priors

In [10] Kaufman et al. have proved bound for regret of Thompson Sampling for Bernoulli bandits.

Theorem 1: In Bernoulli bandit problem for every $\epsilon > 0$ there exists a problem dependent constant $C(\epsilon, \mu_1, \dots \mu_N)$ such that the regret of Thompson Sampling satisfies:

$$\mathcal{R}(\mathrm{T}) \leq$$

$$\leq (1+\epsilon) \sum_{a \in A: \mu_a \neq \mu^*} \frac{\Delta_a \left(\ln(T) + \ln(\ln(T))\right)}{K(\mu_a, \mu^*)} + \tag{8}$$

$$+C(\epsilon, \mu_i, \dots, \mu_N)$$

where A is set of arm indexes and $K(\mu_a, \mu^*)$ is Kullback-Leiber divergence between Bernoulli distributions $\mathcal{B}(\mu_a)$ and $\mathcal{B}(\mu^*)$:

$$K(\mu_a, \mu^*) = \mu_a \ln \frac{\mu_a}{\mu^*} + (1-\mu_a) \ln \frac{1-\mu_a}{1-\mu^*} \tag{9}$$

In [6] Agraval and Goyal proved bound of regret's expected value for Thompson Sampling on Bernoulli bandit problem:

Theorem 2: For *N*-armed stochastic bandit problem, Thompson Sampling using Beta priors has expected regret:

$$\mathbb{E}[\mathcal{R}(T)] \leq$$
$$\leq (1+\epsilon) \sum_{a \in A: \mu_a \neq \mu^*} \frac{\ln(T)}{K(\mu_a, \mu^*)} \Delta_a + O\left(\frac{N}{\epsilon^2}\right) \tag{10}$$

where $A, \epsilon, K(\mu_a, \mu^*)$ has the same meaning as in Theorem [10], and *O* means Big *O* notation.

Further in [6] Agraval and Goyal showed both for Thompson Sampling using Beta priors and Gaussian priors has common expected regret bound.

Theorem 3: For *N*-armed stochastic bandit problem Thompson Sampling using Gaussian priors and Beta priors has expected regret:

$$\mathbb{E}[\mathcal{R}(T)] \leq O\left(\sqrt{NT \ln N}\right) \tag{11}$$

in time $T \geq N$.

They also shown that for Thompson Sampling using Gaussian priors exists problem instance with lower bound of expected regret.

Theorem 4: There exists an instance of N – armed stochastic bandit problem, for which Thompson Sampling, using Gaussian priors, has expected regret:

$$\mathbb{E}[\mathcal{R}(T)] \geq \Omega(\sqrt{NT \ln N} \tag{12}$$

in time $T \geq N$.

The problem of multi-armed bandit has several other solutions worth to mention. Algorithm UCB1 was introduced in [11] with regret bound of $O\left(\sqrt{NT\ln T}\right)$ proved in [12]. Another algorithm MOSS[12] having upper bound of regret $O\left(\sqrt{NT}\right)$ matching problem independent lower bound $\Omega\left(\sqrt{NT}\right)$, the algorithm works when there is known horizon for *T*. These algorithms are suited for stochastic multi-armed bandit problem.

There is also a generalisation of the problem called *adversarial multi-armed bandit problem*, in which rewards are dependent on previous choices of *agent* allowing in particular penalising frequently chosen arms. These algorithms can be applied for adversarial problem however regret bounds proven for stochastic problem do not hold in that case.

6. Q-learning – introduction

Q-learning is an reinforcement learning algorithm designed to find optimal policy for problems modelled by Finite Markov Decision Process (FMDP). The term was introduced in Watkin's PhD thesis [13].

Definition 1: Markov Decision Process is 4-element tuple $\mathcal{S}$, $\mathcal{A}$, P_a, P_r where:

- $\mathcal{S}$ is set of possible states (*state space*);
- $\mathcal{A}$ is set of possible *actions* (*action space*);
- $P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is probability that taking action a in state s will make transition to state s';
- $R_a(s,s')$ is immediate reward after transitioning from state s to s' after making action *a.*

It is worth noting that Markov process is memoryless process, as transitions and rewards are generally random however with probabilities depending only on current state and taken action.

Definition 2: Finite Markov Decision Process is Markov Decision Process where both state space and action space is finite.

Q-learning is based on concept of *Q*-function [14] $Q(s,a)$. *Q*-learning aims at finding the best policy. Algorithm looks forward to find optimal *Q*-function:

$$Q^*(s,a) = E\left[r + \gamma \max_{a'} Q^*(s',a')\right] \tag{13}$$

which resembles Bellman equation [15] used in solving various dynamic programming problems.

7. Q-learning – algorithm

Algorithm require storing $\overline{\mathcal{A}\times\mathcal{S}}$ values of *Q*-function over it's domain. Learning *agent* plays *M* number of *games*, the more *games* it plays the closer to knowing optimal policy it gets, provided that parameters of learning are set correctly. In every *game*, at every time-step t *agent* selects *action* a acording to exploitation--exploration strategy. Commonly used is ε-strategy, where exists probability ε of exploring (and $1-\varepsilon$ probability of exploiting experience), however other solutions to dilemma are possible. After *action* a is taken, *agent* collects *reward* r_t, and makes an *observation* of new state s_{t+1}. Before it takes another *action* it updates value in *Q*-function table setting new value:

$$Q_{new}(s_t,a_t) = (1-\alpha)Q_{old}(s_t,a_t) + \\ +\alpha\left(r_t + \gamma\cdot\max_a Q_{old}(s_{t+1},a)\right) \tag{14}$$

where α is learning rate and γ is discount factor.

Learning rate α indicates how fast algorithm learns, with $\alpha = 0$ algorithm never learns and *Q*-function does not change. On the other hand, with $\alpha = 1$ only new learnt experience is valid, and previous is discarded – this is also not expected behaviour, due to possible randomness in the process. Optimal value is problem specific and must be found between 0 and 1.

Discount factor γ rates importance of future rewards. In case $\gamma = 1$ future rewards have the same meaning as immediate, $\gamma < 1$ means inflation of reward and $\gamma > 1$ means deflation. In case $\gamma = 0$ future rewards are not considered at all (only the next one). Note that if *game* has no limit in steps (time) and γ is high enough *agent* may find that optimal policy is looping over some states without taking actions leading to finishing *game*. It is important that if game model does not provide step limit or quick solution is preferred, future rewards should inflate enforcing *agent* to aim at solution ending *game*. Overall the algorithm with step limit N in follows as presented in algorithm:

```
Require parameters: ℰ,𝒮,𝒜,M,N, ε, α;
initialise table Q[|𝒮̄|][|𝒜̄|];
for _ in 1..=M {
    initialise new game with ℰ;
    observe s from ℰ;
    for t in 1..=N{
        e ≔ random from (0, 1⟩;
        if e > ε{
            â ≔ argmax_{a∈𝒜} Q(s,a)
        }
```

```
        else{
            â := random from 𝒜
        }
        play action â;
        claim reward r;
        observe new state s';
        update Q[s][â]; //equation (14)
        set s := s'
    }
}
```

Algorithm 3: Q-learning

In [16] Melo showed that given the state space and action space is finite the algorithm is convergent to optimal Q-function Q^*(13) if:

$$\forall_{s\in\mathcal{S}}\forall_{a\in\mathcal{A}}: \sum_t \alpha_t(s,a) = \infty \wedge \wedge \sum_t {\alpha_t}^2(s,a) < \infty \quad (15)$$

where α_t is learning rate at step *t*.

However space demand of algorithm rises quickly with cartesian product of state space and action space. As an example in battleship board game with board of $n \times n$ fields and total length of ships m. With $n = 8$ number of states exceeds 2^{256}. Using this algorithm straightforward in larger problems is strongly limited and optimalisations need to be used, as for example grouping certain states in generalised classes and execute algorithms with them. One can use some heuristic to approximate behaviour and adjusting Q-function. In the following sections one such solution is presented.

8. DQN – problem (Console Atari)

In this section a condensed work of DeepMind team over applying reinforcement learning to create algorithm able to learn how to play effectively various console games [17], [18].

Task given to learning *agent* is to play console game and maximize it's score. The *environment* in the model is console emulator (the research was done on Atari 2600).

The full internal state is not known to *agent*, instead every time their *observation* is limited to screen frame (for this particular problem the resolution was 210 x 160 pixels in RGB palette, and frame frequency was 60 Hz). The *observation* is represented with vector of raw pixels representation.

Along the *observation* in each time step *agent* is provided with immediate *reward* which is a difference between current score and the one from previous time step. The target of algorithm to maximize is final score, therefore consequences of taking particular *action* may be observed many steps later and probably that action is not a sole purpose of the consequences, which causes computational difficulty with providing immediate rating for *actions*.

Possible *actions* for *agent* are combinations on the controller (used console had 18 possible *actions* including *no-action*).

The model for the problem is following:

- environment $\mathcal{E}$;
- time steps $t \in \{1, \dots, T\}$;
- observation space $X = \mathbb{R}^d$, with observation made at time t denoted as $x_t \in X$;
- reward space $\mathcal{R}$ such that every time step reward $r_t \in \mathcal{R}$;
- action space: $\mathcal{A}$ with action made at time t denoted as $a_t \in \mathcal{A}$;
- *agent*'s state space for every step t: $(X \times \mathcal{A})^t \times X$, the state at time step t is $s_t = (x_0, a_0, \dots, x_{t-1}, a_{t-1}, x_t)$.

Agent makes use of information about current state s, it does not have insight to internal state of *environment* and cannot fully understand current situation only basing on latest observation. It learns strategies considering sequence of previous observations and actions taken, therefore $s_t = (x_0, a_0, \dots, x_{t-1}, a_{t-1}, x_t)$.

Note 3: In fact observation space was reduced with greying pixels, cropping image to focus on playing screen and downscaling using convolutional network.

9. DQN – solution (deep Q-learning)

The state space in problem is too large to possibly use Q-learning based on Q-table. Researchers in DeepMind proposed implementing deep artificial network to be Q-function approximation. The neural network function approximator is given:

$$Q(s,a;\theta) \approx Q^*(s,a) \quad (16)$$

where θ is set of weights of used neural network. The approximator is called Q-network and is trained by minimizing loss function $L_i(\theta_i)$ each step i:

$$L_i(\theta_i) = E_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a,\theta_i)\right)^2\right] \quad (17)$$

with $\rho(s,a)$ is probability distribution over states s and actions a and

$$y_i = E_{s'\sim\mathbb{E}(s,a)}\left[r + \max_{a'} Q(s',a',\theta_{i-1})\right] \quad (18)$$

if s' being next state after taking *action* a in state s, it is achieved with tranistion function

taken from emulator (*environment*) $s' = \mathcal{E}(s, a)$.

During learning target of optimalisation depends on approximation given by remembered previous network θ_{i-1} and is not fixed like in supervised learning, where algorithm is provided with reference solution. In this case neural network takes action and compares result with result approximated Q-function with previous network. Algorithm does not have target score in horizon instead it measures performance change and tries to be better than itself previously.

Differential of loss function with respect of weights leads to gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \\ = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\varepsilon}\Big[\Big(r + \gamma \max_{a'} Q(s', a', \theta_{i-1}) \\ - Q(s, a, \theta_i)\Big)\nabla_{\theta_i} Q(s, a, \theta_i)\Big] \quad (19)$$

Instead of calculating full expectation of gradient authors [17] optimised loss function using stochastic gradient descent as it gives better computational performance in most cases.

Authors of algorithm used technique called *experience replay*. It utilizes a set $\mathcal{D} = e_1, \dots, e_H$ called *experience memory* or *experience buffer*. Theoretically $e_t(s_t, a_t, r_t, s_{t+1})$ maps state and action to reward and transitioned state. However as previously defined *state* s_t is list of subsequent *observations* and *actions* $s_t = (x_1, a_1, \dots, x_t)$ the entry e would be dynamically sized and that would be difficult to maintain in neural network. Therefore introduced is function $\phi(s)$ producing fixed size history representation. Experience elements are then:

$$e_t = \big(\phi(s_t), a_t, r_t, \phi(s_{t+1})\big) \quad (20)$$

The idea behind *experience replay* is to use random previous experiences from buffer when gradient descent step is calculated, instead of current values. As algorithm authors wrote updating network weights on consecutive samples was inefficient due to strong correlation between them.

The DQN algorithm is presented as algorithm:

```
Require parameters: ℰ, H, M, T, ε, x_1;
initialise empty 𝒟 of size H;
initialise Q-network with random weights θ
for _ in 1.. =M {
    initialise s_1 := x_1; ϕ_1 := ϕ(s_1);
    for t in 1 ..= T do
        e := random from (0, 1⟩;
        if e > ε{
            a := argmax_{ã\in 𝒜} Q(s_t, ã)
        }
        else {
            a := random from 𝒜;
        }
        execute a in ℰ;
        observe x_t and collect r_t;
        set s_{t+1} := s_t + [a_t, x_{t+1}]
        set ϕ_{t+1} := ϕ(s_t)
        store (ϕ_t, a_t, r_t, ϕ_{t+1}) in 𝒟;
        sample (ϕ_j, a_j, r_j, ϕ_{j+1}) from 𝒟;
        if ϕ_{j+1} is terminal {
            y_j := r_j
        }
        else {
            y_j := r_j + γ max_ã Q(ϕ_{j+1}, ã; θ)
        }
        set l := (y_i − Q(ϕ_{j+1}, ã; θ))^2
        perform gradient descent step on l;
    }
}
```

Algorithm 4: DQN

Note 4: Experiences can be (and it is usually done so) sampled in minibatch (a subset of $\mathcal{D}$). Several experiences e_j are sampled and gradient descent step is calculated on them combined which provides more data and thus better stability. Minibatching is common optimalisation technique for neural networks. Efficiency was discussed by Li et al. in [19].

Team developing DQN comprised it's results to other reinforcement learning techniques which addressed Atari games before (Sarsa [20] and Contingency [21]). Compared to these algorithm DQN scored higher results. Comparing to expert human players relative performance of the algorithm differs depending on game. It showed to be better at games: Breakout, Enduro and Pong, however scored lower in games: Space Invaders, Q*bert, Seaquest and Beamrider. Detailed results of algorithm can be read in papers: [18], [17].

Algorithm is model free which mean that it is not tied to playing Atari games and can perform decision optimalisation for problems in more general class.

10. AlphaGo Zero – problem (Go)

Go is 2-player strategic board game invented in ancient China over 2500 years ago. Both players alternately place their stones joints of 19x19 lines (or smaller used for education). Game is simple in it's principles with few simple rules however the estimated number of legal board

states is calculated as $2.1 \cdot 10^{170}$ [22]. Mastering game was big achievement for artificial intelligence development.

Team DeepMind developed algorithm AlphaGo Zero which uses reinforcement learning playing games against itself, the following section is based on their research paper [23]. AlphaGo Zero suppresses previous work of the team: algorithms AlphaGo Fan (codenamed after defeating European champion Fan Hui in October 2015) and AlphaGo Lee (codenamed after defeating Lee Sedol, winner of 18 international titles in March 2016). AlphaGo Zero differs from previous version primarily in approach of learning from it's own plays instead of implementing supervised learning of world's best players. AlphaGo Zero uses Monte Carlo Tree Search (MCTS) algorithm aided with deep neural network. The first to be described will be concept behind MCTS.

11. AlphaGo Zero – solution

One of the key concept of algorithm is usage of Monte Carlo Tree Search (MTLS) algorithm. MCTS is general heuristic search algorithm for decision process mostly using in solving game trees. Algorithm uses tree graph with nodes being *states* of game (or observed state in games with partial information) and arcs (directed edges) being *actions* linking parent *state* with child *state*. To nodes and edges there is attached data helping in selection optimalisation – usually it consists of information how many times this node was explored before and some averaged result of the *games* in which the node was explored. At the beginning tree contains only root node and a number of *games* is played. In each game the following steps are made:

1. *Selection:* starting from root a legal *action* is chosen and nodes they lead to explored. *Action* selection is randomized with respect to expected results. The selection continues until selected node L has potential child which has not been explored yet.
2. *Expansion:* from node L select random *action* leading to unexplored node C.
3. *Simulation:* Continue {*game* from node C using default policy (e.g. random actions) leading to end of *game*. Playing the game from this point to the end is often called *playout* or *roll-out*.
4. *Backpropagation:* update nodes selected in game (from C, through L to root) with information gained from this *game.*

More detailed explanation of method is available in [24].

Solution utilizes deep neural network f_θ with parameters θ which output is given $(p, v) = f_\theta(s)$, where p represents probabilities of selecting each move a ($p_a(s) = Pr(a|s)$ and v estimates probability of current player winning the *game*.

Note 5: Parameter p here is actually set of probabilities and if seen in equation operation refers to members separately. This notion was copied from reference paper [23] in order to remain compatible with original.

In AlphaGo Zero nodes are the state of board and every arc (s, a) stores prior probability $P(s, a)$, visit count $N(s, a)$, and action value $Q(s, a)$.

Policy π is selected with MCTS algorithm with help of neural network f_θ. The adapted MCTS algorithm follows:

1. *Selection:* Select moves maximizing upper confidence bound $Q(s, a) + U(s, a)$, where $U(s, a) \propto \frac{P(s,a)}{1+N(s,a)}$ [25], [26] until leaf node s' is encountered.
2. *Expansion and Simulation:* until the end of the game nodes are expanded and followed using just neural network:
$$\big(P(s',\cdot), V(s')\big) = f_\theta(s') \qquad (21)$$
At the end of game the winner is proclaimed and granted reward $z \in \{-1,1\}$ and that step is marked T network:
3. *Backpropagation*: for each traversed edge (s, a) parameters are updated: incremented $N(s, a)$, evaluated:
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{s'|s,a\to s'} V(s')) \qquad (22)$$
Also for every traversed edge neural network is updated so it better fits equation $(p, v) = f_\theta(s)$ minimizing loss function:
$$l = (z - v^2) - \pi^T \log(p) + c\|\theta\|^2 \qquad (23)$$
where $-\pi^T \log(p)$ is cross entropy between move probabilities sampled from neural network and search probabilities π. π is vector of search probabilities (recommending moves to play) sampled from MCTS algorithm.
Therefore network training aims at minimalising difference between result prediction v and actual result z and maximalisation of similarity between neural network move suggestion and search probabilities. Finally c is parameter preventing overfitting.

AlphaGo Zero learnt for 40 days, played 29 million games. After learning it proved superior to previous algorithms (detailed report of competition is provided in project's paper [23]. Although *agent* used powerful computing machine with tensor processing unit accelerators and used large number of resources it marked another big step for artificial intelligence. Research proved that algorithms learning only by playing with themselves can reach superhuman possibilities.

12. Summary

In this paper a selection of reinforcement learning algorithms was presented in order to give possibly wide overview of reinforcement learning application in game theory. The focus was placed on games, however reinforcement learning finds application in industry for example in developing car parking [27].

Game theory delivers many problems with different characteristics, thus methods should be adjusted to specific problem classes. Existence of perfect algorithm which is optimal for learning solution of every problem is questionable (assuming algorithm limited with computing power). Monte Carlo Tree Search combined with artificial neural network proved to be very successful in games Go and Chess, yet these games include perfect situational information and no world randomness. Application of this algorithm to games with randomization and imperfect information (like most card games and board games with dices) may be suboptimal and need special adjustment or developing algorithm basing on different concepts. Development of new powerful hardware including tensor processing units, graphical processing units and standard central processing units gives researchers more possibilities and opens new problems for practical analysis.

Recent years brought significant development of artificial intelligence, with artificial neural networks with the greatest share of the stage. In fact artificial intelligence is already used in the industry.

Reinforcement learning is not limited to academic experiments on the field of game theory. Firstly, some cars are currently constructed with driving assistance, automatic parking or even with fully autonomic driving possibilities. Still errors occur causing controversies with allowing them on public row, however with little doubt machines will prove better and safer on the roads than human driven cars. Another example is rising bot activity on stock and cryptocurrency exchange. Most investors are aware of patterns occurring on the market, these patterns are also viable to artificial intelligence instances. Further increase in AI presence on trading market may lead to sharpening of existing patterns, their evolving or even introducing new patterns. Last but not least reinforcement learning could be utilized in business, political and military games strengthening results already calculated in simulators. In the last case, there is nearly no doubt, that the most significant players develop this branch of technology in some level of secrecy. Real world games have more complicated fields, rules and objectives. Mathematic workshop is being developed, computational possibilities increase leading possibly to state when most crucial decisions could be made by artificial intelligence. In the end, if artificial intelligence has already won with world champions in board games it could supposedly beat strategists on real life fields.

13. Bibliography

[1] Binmore K., *Game theory: a very short introduction*, OUP Oxford, 2007.

[2] Ameljańczyk A., "Teoria gier", Vol. 690, p. 78, WAT, 1978.

[3] Lattimore T., Szepesvári C., *Bandit algorithms*, Cambridge University Press, 2020.

[4] Thompson W.R., "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples", *Biometrika*, Vol. 25, No. 3–4, 285–294 (1933).

[5] Thompson W.R., "On the theory of apportionment", *American Journal of Mathematics*, Vol. 57, No. 2, 450–456 (1935).

[6] Agrawal S., Goyal N., "Further optimal regret bounds for thompson sampling", *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics* (*AISTATS*), Vol. 31, pp. 99–107, USA, 2013.

[7] Chapelle O., Li L., "An empirical evaluation of thompson sampling", in: *Advance in Neural Information Processing Systems 24*(*NIPS 2011*), Vol. 24, 1–9, 2011.

[8] Bolstad W.M., Curran J.M., *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.

[9] Agrawal S., Goyal N., "Analysis of thompson sampling for the multi-armed bandit problem", *Proceedings of the Conference on Learning Theory*, Edinburgh, UK, 25–27 June 2012, pp. 31–39.
[10] Kaufmann E., Korda N., Munos R., "Thompson sampling: An asymptotically optimal finite-time analysis", in: *International Conference on Algorithmic Learning Theory*, LNCS 7568, pp. 199–213, Springer 2012.
[11] Auer P., Cesa-Bianchi N., Fischer P., "Finite-time analysis of the multiarmed bandit problem", *Mach. Learn.*, Vol. 47, No. 2, 235–256 (2002).
[12] Audibert J.-Y., Bubeck S., "Minimax policies for adversarial and stochastic bandits", in: *COLT – The 22nd Conference on Learning Theory*, Montreal, Quebec, Canada, June 18–21, 2009.
[13] Watkins C.J.C.H., *Learning from delayed rewards*, University of London 1989.
[14] Sutton R.S., Barto A.G., *Reinforcement learning: An introduction*, MIT Press, 2018.
[15] Bellman R., Kalaba R.E., *Dynamic programming and modern control theory*, Academic Press, NY 1965.
[16] Melo F.S., "Convergence of Q-learning: A simple proof", *Institute of Systems and Robootics Tech. Rep.*, pp. 1–4, 2001.
[17] Mnih V. and others, "Playing atari with deep reinforcement learning", *arXiv Prepr. arXiv1312.5602*, 2013.
[18] Mnih V. and others, "Human-level control through deep reinforcement learning", *Nature*, Vol. 518, No. 7540, 529–533 (2015).
[19] Li M., Zhang T., Chen Y., Smola A.J., "Efficient mini-batch training for stochastic optimization", in: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, NY, 2014, pp. 661–670.
[20] Bellemare M.G., Naddaf Y., Veness J., Bowling M., "The arcade learning environment: An evaluation platform for general agents", *Journal of Artiicial . Intelligence Research*, Vol. 47, 253–279 (2013).
[21] Bellemare M.G., Veness J., Bowling M., "Investigating contingency awareness using Atari 2600 games", *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, Vol. 26, No. 1, pp. 864–871, 2012.
[22] Tromp J., Farnebäck G., "Combinatorics of go", in: *Computers and Games*, LNCS 4630, pp. 84–99, Springer 2006.
[23] Silver D. and others, "Mastering the game of go without human knowledge", *Nature*, Vol. 550, No. 7676, 354–359 (2017).
[24] Browne C.B. and others, "A survey of monte carlo tree search methods", *IEEE Transaction on Computational Intelligenceand AI in Games*, Vol. 4, No. 1, 1–43 (2012).
[25] Silver D. and others, "Mastering the game of Go with deep neural networks and tree search", *Nature*, Vol. 529, No. 7587, 484–489 (2016).
[26] Rosin C.D., "Multi-armed bandits with episode context", *Annals of Mathematics and Artificial Intelligence*, Vol. 61, No. 3, 203–230 (2011).
[27] Zhang P. and others, "Reinforcement learning-based end-to-end parking for automatic parking system", *Sensors*, Vol. 19(18), 3996 (2019).

Przegląd wybranych rozwiązań opartych na uczeniu ze wzmocnieniem dla problemów z teorii gier

R. JAROSZ

Artykuł gromadzi wybrane podejścia do rozwiązania problemów z teorii gier wykorzystując uczenie ze wzmocnieniem. Zastosowania zostały dobrane tak, aby przedstawić możliwie przekrojowo klasy problemów i podejścia do ich rozwiązania. W zbiorze wybranych algorytmów znalazły się: próbkowanie Thompsona, Q-learning (Q-uczenie), DQN, AlphaGo Zero. W artykule nacisk położono na przedstawienie intuicji sposobu działania algorytmów, koncentrując się na przeglądzie technologii zamiast na technicznych szczegółach.

Słowa kluczowe: sztuczna inteligencja, teoria gier,, próbkowanie Thompsona, przeszukiwanie drzew Monte Carlo.