FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Přemysl Bašta

Thesis title

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	: Mgr. Martin Pilát, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date ....

Author's signature

Dedication.

Title: Thesis title

Author: Bc. Přemysl Bašta

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: : Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstract.

Keywords: key words

Contents

Introduction

In recent years people are witnessing fast progress in artificial inteligence(AI) in all sort of domains. Beating world champions in chess or Go is no longer any issue for AI models. Same pattern is showing in more recent popular games such as Dota (OpenAI [2019]) or Starcraft, where even great Human-AI team cooperation behavior has been achieved. However, all of these examples share common property of being competetive. Ultimate goal of our society is to create AI that will be cooperating with humans, not competing.

Recent work has shown us that AI cooperative models trained together for purely cooperative tasks tend to rely mostly on expect near optimal behavior from their partners and fail to cooperate with partners who don't satisfy this condition.

Which is bad news for us humans as our behavior is rarely optimal.

Great example of human-AI cooperation domain where humans do not always perform perfectly are self driving cars. In a situation where an accident is imminent humans have to react quickly without having enough time to consider all possible reactions or even analyze entire current road situation. However car accidents maybe even too extreme example of human unoptimal behavior. People often fail at even simplier task of obeying the standard traffic rules when having enough time to react. We can imagine how predicting human behavior is not an easy task for self driving car.

In this work we will firstly revisit definition of Markov decision process, building block of reinforcement learning, then mention basic aproaches of Q-learning. We will focus on policy branch of reinforcement algorithms, mention their variants and conclude with policy learning algorithm Proximal policiy optimization which is considered as state of art algorithm masively deployed in many succesful projects.

We will utilize simplified cooperative cooking environment based on popular video game Overcooked, where two partners are forced to coordinate shared task of cooking and delivering soup to customer. Here we are going to summarize what aproaches have been tested in related work in terms of ad-hoc agent cooperation. Some of which will we reimplement for our evaluating purposes. We mention problem of definition of robustness of agent cooperation.

And finally we contribute with our ideas of diversificated partners population.

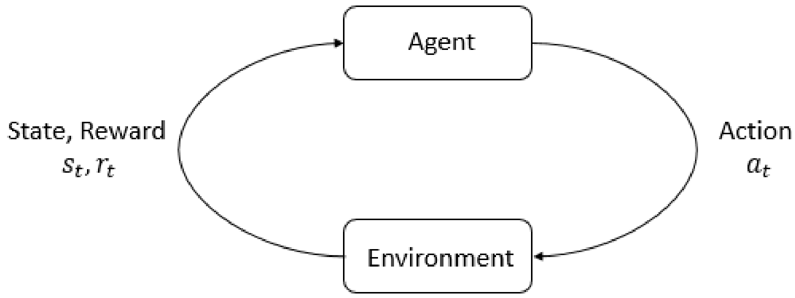
1. Introduction to reinforcement learning

1.1 Markov decision process

Environment cycle

TODO: Cite properly, more sources

This entire chapter is inspired by introduction presented by `https://spinningup.openai.com/en/latest/spinningup/rl_intro.html` Whole problem of reinforcement learning (RL) can be best described by following visualisation.



Environment represents some kind of world with its inner rules and properties. Agent is then an entity that exists inside this world, observes **state s** of the world, decides on the basis of this state to react with **action a** and as consequence of this action recieves **reward r**. Entire mechanism of this environment can be then broken into these cycles of states, actions and rewards. The goal of an agent is to interact with environemt in such a way to maximize its cumulative reward.

Definition

Briefly described environment can be transformed into mathematical model.

Markov Decision Process is 5-tuple $\langle S, A, R, P, \rho_0 \rangle$, where

- S is the set of all valid states,
- A is the set of all valid actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$ being reward obtained when transitioning from state s_t to s_{t+1} using action a_t.
- $P : S \times A \to \mathcal{P}(S)$ is the transition probabilty function, where $P(s_{t+1}|s_t, a_t)$ is probability of transition from state s_t to s_{t+1} after taking action a_t.
- ρ_0 is starting state distribution.

The name Markov comes from the fact, that the system satisfies Markov property, which states that history of previous states have no effect on next state and always only current state is considered for state transition.

TODO: Do I need also Partial observable MDP?

After having defined mathematical model of environment, let's review over the related concepts.

Observability

State s contains all information about environment at given time. However, agent in some environment can only percieve **observation o** where some information about environment can be missing. In that case we say environment is **partially observable** as opposed to **fully observable** environment where agent has available entire information at it's observation.

Actions and policies

Environments can also differ from point of what actions are possible inside of given world. Set of possible actions is called **action space** which again can be divided into two types. **Discrete** action space contains finite number of possible actions. And **continuous** action space which allows for action to be any real-valued number or vector.

Agent's action selection can then be described by a rule called **policy**. Common notation is established that if the action selection is deterministic, we say policy is **deterministic** and denote

$$a_t = \mu(s_t).$$

If policy is **stochastic** it is usually noted as

$$a_t \sim \pi(\cdot|s_t).$$

Policies are main object of interest of reinforcement learning as this action selection mechanism of an agent is what we are trying to learn. Policy, for optimization purposes, is function often **parametrized** by a neural network whose parameters are usually denoted by symbol θ , therefore parametrized deterministic and stochastic policy are represented by symbols $\mu_\theta(s_t), \pi_\theta(\cdot|s_t)$ respectively.

Trajectory

Next important definition is notion of trajectory also known as episode or rollout. Trajectory is a sequence of states and actions in an environment.

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

Initial state s_0 of an environment is sampled from **start-state distribution** denoted as ρ_0. Subsequent states follow transition laws of environment. These can be again deterministic

$$s_{t+1} = f(s_t, a_t)$$

or stochastic,

$$s_{t+1} \sim P(\cdot|s_t, a_t)$$

Return

We already mentioned agent aspiration of maximization of cumulative rewards. Now we combine it with trajectories and derive formulation of **return**.

$$R(\tau) = \sum_{t=0}^{T} r_t$$

for finite-horizon. And infinite-horizon discounted return:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

Discounting infinite-horizon is both intuitive and convinient for mathematical purposes.

Optimal policy

In general, the goal of the RL is to find such policy that maximizes expected return when acted upon it. Let us suppose both the environment state transitions and policy are stochastic. Then we can define probability of T-step trajectory as

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t).$$

Expected return $J(\pi)$ can be then expressed as

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)].$$

And finally we can conclude with definition of optimal policy

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

which is also expression that describes central optimization RL problem.

Value functions

Once we have some policy π it would be useful to define value of observed state. For that matter we define two functions.

On-Policy Value Function $V^{\pi}(s)$, which yields value of expected return when starting from state s and following policy π:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

Similarly we define **On-Policy Action-Value Function** $Q^{\pi}(s, a)$ which adds possibility to say that in state s we take an arbitrary action a that does not necesarily have to come from policy π:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

For the optimal policy we further define **optimal value function** $V_{\pi^*(s)}$ and **optimal action-value function** $Q_{\pi^*(s,a)}$:

$$V^*(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s],$$
$$Q^*(s, a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

Bellman equations

There exist formulations called Bellman equations that provide us a way how to express value function using action-value function and vice versa. They are built on idea that the value of a state is equal to reward obtained in given state, plus the value of a state where you get in next transition. This idea provides also recursive relation.

$$\begin{aligned} V^\pi(s) &= \mathop{\mathbb{E}}_{a\sim\pi(s)}[Q_\pi(s,a)] \\ &= \mathop{\mathbb{E}}_{a\sim\pi(s),s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

$$\begin{aligned} Q^\pi(s,a) &= \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma V_\pi(s')] \\ &= \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)}[R(s,a,s') + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(s')}[Q_\pi(s',a')]] \end{aligned}$$

For us, the most important theorem is reformulation of Bellman equations for optimal policies:

$$\begin{aligned} V^*(s) &= \max_a \mathop{\mathbb{E}}_{s'\sim P}[R(s,a,s') + \gamma V^*(s')] \\ Q^*(s,a) &= \mathop{\mathbb{E}}_{s'\sim P}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] \end{aligned}$$

As we will see in next section. Firts RL algorithm will be straightforward application of Bellman equation for optimal policy.

Advantage function

After we've devoted few sections defining functions for absolute value of actions or state-action pairs, it is worthy to consider also relative value. Often, when dealing with RL problems, it is not so important for us to know the exact value of the action-state pair, but rather whether and by how much a given action is better, on average, relative to others. In other words, we want to now relative advantage of given action over others. For a given policy π, advantage function $A^\pi(s,a)$ describes how much it is better to take action a over randomly sampled actions following policy π under assumption of following policy π in all consequent steps. From the mathematical point of view advantage function is defined as follows:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

The concept of an advantage function will be an integral part of policy gradient based methods, as we will see in a later section.

2. RL alrotighms

2.1 Q-Learning

Idea

We will start with family of algorithms that focuses on learning action-value approximator $Q_\theta(s, a)$ as described in previous chapter. For this reason, the group of such algorithms can be also refered to as Q-learning. Our primary goal in RL problem is to find policy that agent can follow. In the case of Q-learning once we have have learned approximator $Q_\theta(s, a)$ we can derive policy by always taking the best possible action in given state according to the learned action-value function

$$a(s) = \arg\max_a Q_\theta(s, a).$$

By incorporating Bellman equations for optimal policy we can directly train Q-network by minimizing the loss

$$L(\Theta) = (r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a))^2.$$

And by computing loss gradient we arrive at update rule

$$Q_\theta(s, a) = Q_\theta(s, a) + \alpha(r + \gamma \max_{a'} Q_\theta(s', a') - Q(s, a)),$$

which is the backbone of the algorithm 2.1 bearing the same name.

Instability

Algorithm has been rarely used in his pure form. This aproach has been primarily described for tabular methods where action-value function is represented by table instead of network approximator. In its simplest form, the training is unstable and suffers from a number of significant shortcomings. Most worthy of mention is the theoretical deadly triad counter example Sutton and Barto [2018] which consists of a combination of value approximation, bootstrapping, and off-policy training that can lead to instability and divergence. Condition of value approximation is met since we use Q-network to approximate action-value. Bootstrapping means that estimate is used for computation of targets, this is also true in Q-learning for the same reason. Lastly term off-policy stands for approach where training data are collected using different distribution than that of a target policy.

DQN

One of the most outstanding paper based on Q-learning was the algorithm Deep Q-learning 2.2 which demonstrated super-human results on multiple Atari games Mnih et al. [2015]. We give the pseudocode of the algorithm in it's original form. Notation might seem a bit different from ours, nevertheless it represents the same mechanisms that we expect. To address the problem of correlated transition sequences of data they introduce experience replay where previously sampled transitions are stored. During training data are sampled from this buffer

Algorithm 2.1: Q-learning

Input: initial action-value aproximator Q parameters θ .

repeat

  Observe state s and select action a according to ϵ-greedy w.r.t. Q e.g.

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \arg\max_{a} Q(s,a), & \text{otherwise.} \end{cases}$$

  Execute a in the environment.

  Observe next state s', reward r and done signal d to indicate whether s' is terminal.

  if *d is true* **then**

    Reset environment state.

  end if

  Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \max_{a'} Q_\theta(s', a')$$

  Update Q-network taking one step of gradient decent on

$$(y(r, s', d) - Q_\theta(s', a))^2$$

until *convegence*;

thus smoothing the training distribution over different past behaviors. However, probably the most important idea was the usage of target Q-network, which broke value approximation condition of deadly triad, thus making the algorithm more stable. Target network is a copy of original Q-network that has its parameters frozen and updated only once in while based on parameters of the main network. It's sole purpose is to compute target estimates that are not directly dependent on Q-network function.

Rainbow

Deep Q Learning was a significant contribution that led to the study of further Q Learning capabilities. Project Rainbow Hessel et al. [2017] could be probably marked as peak of such research. In this paper they examine further several isolated ideas of possible improvements and try to combine them together. To name a few, they use Double Q-network to address the problem of maximization bias and improves sampling from experience buffer by considering priority of stored individual data samples. Together with all other improvements that achieved at given time state-of-the-art performance on Atari 2600 benchmark, both in terms of data efficiency and final performance.

Algorithm 2.2: Algorithm 1: deep Q-learning with experience replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode = 1,M* **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for *t=1, T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observer reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set

$$y_j = \begin{cases} r_j & \text{if terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

2.2 Policy gradient algorithms

Idea and mathematical theory

In the previous section, we got acquainted with the first group of RL algorithms, where we derived final policy by taking action according to *argmax* of our Q function approximator. Since our objective is to find optimal policy, using this approach of considering Q values may seem a bit indirect. There is a whole other family of algorithms out there that deal with this very issue. As the name suggests Policy gradient algorithms focuses on directly optimizing policy $\pi_\theta(a|s)$. This is achieved by directly taking steps along gradient of the performance objective of expected return $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$. Optimization step then has the form of:

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|\theta_k$$

and the gradient $\nabla_\theta J(\pi_\theta)$ is called **policy gradient**.

Before we can transform this into an algorithm we have to be able to compute policy gradient numerically. Such expression can be obtained as result of Policy Gradient Theorem (PGT). Policy Gradient Theorem. It holds:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &\propto \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\sum_{t=0}^{|\tau|} R(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)] \\ &\propto \mathop{\mathbb{E}}_{s_t\sim\eta_{\pi_\theta}} \mathop{\mathbb{E}}_{a_t\sim\pi_\theta}[R(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)]\end{aligned}$$

TRPO

limiting how much can changed policy differ from previous, making save steps Second order method

PPO

easy clipped version of TRPO

2.3 Comparison with off policy Q learning methods

2.4 Idea, motivation and brief technical description of algorithm

2.5 Variants of policy theorem

Vanilla

PPO

3. Multi agent environments for RL ???

3.1 Definitions

3.2 Possible mention of MAPPO success

3.3 Cooperation harder than competition

4. Overcooked environment

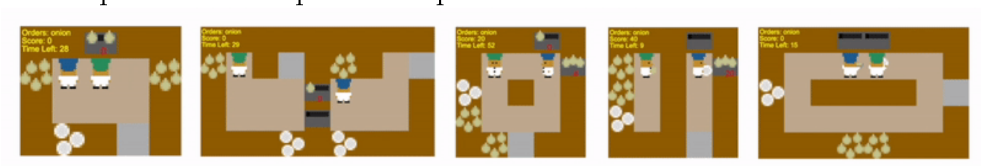
4.1 Overcooked game

Before we get into our problems with cooperation let us first examine the environment. We will be working with environment based on popular cooking video game `https://ghosttowngames.com/overcooked/`. Overcooked is multiplayer cooperative game where the goal is to work in a kitchen as a team with partner cooks and prepare together various dishes within limited time. However, the game is dynamic to a great extent. In many maps the kitchen itself is not static and may be changing on a run. Moreover, random events such as pots catching fire add to the chaos. The challenge lies in coordination with rest of the team and dividing subtasks efficiently.

The aforementioned game was simplified and reimplemented to simpler environment `https://github.com/HumanCompatibleAI/overcooked_ai` to serve a purpose of scientific common ground for studying multi agent cooperation in somehwat complex settings. Lot of additional features of original game were removed and remained only essential coordination aspects. In its simplest form, environment is taking place in small static kitchen layout where only available recipe is onion soup which can be prepared by putting three onions in a pot and waiting for given time period. Somewhere in the kitchen there is unlimited source of onions and dish dispenser, where player can grab a dish to carry cooked onion soup in to the counter. Team of cooks is rewarded as team by abstract reward of value 20 every time cooked soup is delivered to the counter. It may seem that the task is quite straightforward. However, players face problems on multiple levels.

4.2 Basic layouts

Although the Overcooked implementation has its own generator that can be used to generate new random kitchen layouts, the majority of the related scientific work has so far experimented with a fixed set of predefined layouts, where each of them capture some important aspect of coordination.



(From left to right: Cramped room, Assymetric advantages, Coordination ring, Forced coordination, Counter circuit)

Cramped room as a name suggests represents cramped kitchen layout where all important places are relatively easy to reach. Challenge lies in low level coordination of movement with the other partner as there is no spare room.

In Assymetric advantages both players are located in separated regions where each region is fully self-sustaining. However, each region has better potential for specific subtask. And it is only when both players make the most of their own region's potential that the maximal shared efficacy is reached.

The Coordination ring is another example of a layout where clever coordination is required as the only possible movement around the kitchen is along a narrow circular path that can be used in a given direction. For example, if one player decides to move in clockwise direction, the other player would automatically get stuck if persuing counter-clockwise movement.

Forced coordination kitchen layout is significantly different from others. In this layout, each player is located in a separate region where neither player has all the resources necessary to prepare a complete onion soup. Thus, players are forced to cooperate with each other with the resources they have.

In the last layout, the situation may look similar to the coordination ring. However, in this case, carrying onions around the entire kitchen is highly suboptimal no matter which direction the players choose. To deliver onions efficiently, players must pass them over the counter to shorten the distance. However, the cooks still need to decide who will be responsible for bringing the plates.

4.3 Environment description

4.3.1 Actions space, episode horizont, shaped rewards, state representation MLP vs CNN

4.3.2 Reset state static position, index switching, Randomization function

5. Related work

5.1 Human-ai cooperation results

5.1.1 Human cooperation

Most of the previous work in this area has focused on one of two types of coordination. The first being coordination between a human and an AI partner. And second, focusing solely on the fully AI-driven pair.

While perfect AI-human coordination is generally a more desirable goal to achieve in all sorts of domains, it will not be our main focus. Several previous scientific papers have addressed this issue. A particularly noteworthy contribution is the article On the Utility of Learning about Humans for Human-AI Coordination `https://arxiv.org/abs/1910.05789`, whose authors are also responsible for creating the overcooked environment implementation. They collected many human-human episodes and incorporated these experiences in the form of human behavior clone models into the training. Their main conclusions were that AI models often rely heavily on the optimal behavior of their partner. However, when such a model is paired with generally suboptimal human behavior, it often fails to cooperate at all, regardless of the approach used for training of the AI model.

One of the other important conclusions they came to was that even AI-AI coordination often fails when models are paired with another AI model trained using a different approach. We will discuss some of these popular approaches in the next section.

TODO: How much further focus on human-ai coordination if not our interest?

5.2 Problem of robustness

5.2.1 Problem of robustness definition

Ad hoc agent playing? Trivial states failure (unit-test based aproach)?

5.3 AI-AI coordination

5.3.1 Aproaches

Self-play, Population

5.3.2 Results

It fails

6. Our work - Preparation

6.1 Utilized framework

Comparision of rllib and StableBaselines3

Rllib framework was used in original paper, however for our usages stable baselines seemed sufficient and reasonably easy to extend. Stable baselines has no explicit support of multi-agent environments.

6.1.1 Modifications of stable baselines

CNN policy wrapper, Partner embedded into environment

6.1.2 NN structure modification

6.1.3 Hyperparameters random search

6.1.4 Randomization function correction

6.2 Self-play

6.2.1 Training

6.2.2 Results

7. Our work - Contribution

7.1 Our definition(s?) of robustness

Probably just average of pair results (non diagonal in case of same sets). Maybe percentage of pairs who surpassed some threshold reward?

7.2 Population construction

7.2.1 SP agents initialization

One agent is not enough?

7.2.2 population partner sampling during training

See if playing with whole population at once differs from one random partner for episode

7.2.3 Final agent training

7.3 Diverzification

maximize kl divergence among population partners policies

7.3.1 Population policies difference rewards augmentation

7.3.2 Population policies difference loss

Conclusion

Bibliography

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017. URL `https://arxiv.org/abs/1710.02298`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518: 529–33, 02 2015. doi: 10.1038/nature14236.

OpenAI, 2019. URL `https://openai.com/blog/openai-five-finals/`.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment