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

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 have witnessed rapid progress in artificial intelligence (AI) in all kinds of fields. Beating world champions at chess or Go is no longer a problem for AI models. The same pattern can be seen 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 the common trait of being competitive. The ultimate goal of our society is to create AI that cooperates with humans, not competes with them.

Recent work has shown that cooperative AI models trained together on purely cooperative tasks tend to rely on near-optimal behavior from their partners, and fail to cooperate with partners who don't meet this condition. This is bad news for us humans, because our behavior is rarely optimal.

A great example of human-AI cooperation 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 the entire current road situation. However, car accidents are perhaps even too extreme an example of human suboptimal behavior. Nevertheless, people often fail at the even simpler task of following standard traffic rules when they have enough time to react. We can imagine that predicting human behavior is not an easy task for a self-driving car.

In this work, we will first operate in a single-agent environment, revisiting the definition of Markov decision, the building block of reinforcement learning. Based on this theory, we will intuitively introduce popular reinforcement learning algorithms divided into two categories of Q-learning and policy optimization. We primarily focus on policy branch of reinforcement algorithms, especially policy learning algorithm proximal policy optimization, which is considered as state of the art algorithm masively deployed in many successful projects.

Subsequently, we extend the theory developed for single-agent environments to more complex scenarios where more agents are involved. We highlight problems related to multi-agent settings, where the observability of the world is often an issue compared to single-agent environments. We illustrate problems with multi-agent training, where changes in one agent affect the behavior of the environment from the point of view of other agents, introducing the problem of non-stationarity. Finally, some single-agent variants of reinforcement learning algorithms are extended to multi-agent settings, where we mention the important aspects of these extensions.

We will use a simplified cooperative cooking environment based on the popular video game Overcooked, where two partners are forced to coordinate a shared task of cooking and delivering soup to a customer. We will familiarize ourselves with several different kitchen layouts, as each layout may offer different cooperative obstacles. Here we summarize what approaches have been tested in related work with respect to ad hoc agent cooperation. We mention the problem of defining the robustness of agent cooperation and different possible definitions of robustness.

And in the last part of this work, we prepare our working environment by modifying the stable-baselines3 library designed for reinforcement learning algorithms. We try to reimplement some methods of previously related work regarding the problem of ad hoc coordination in AI-AI settings, both for verification purposes and also for building us an evaluation tool for our experiments. And finally, we propose a diversification method for building a population of agents that are designed to try to differentiate their behavior from those they have encountered in the training population. After tuning and optimizing our experiments on a selected kitchen layout, we then evaluate our approach on some of the remaining layouts.

1. Introduction to reinforcement learning

In this introductory chapter, we slowly build up the intuition and motivation behind reinforcement learning. We start by defining the mathematical model of the Markov decision process and then proceed with other related properties and relationships. The following pages are inspired by the introduction to reinforcement learning as presented by the authors of the spinningup library (Achiam [2018]) and also by the introductory book to reinforcement learning (Sutton and Barto [2018]).

1.1 Markov decision process

Definition

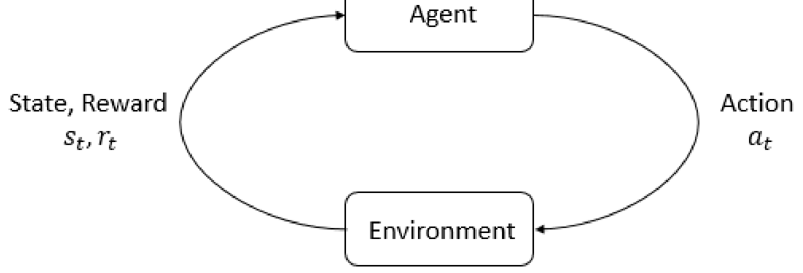
Markov Decision Process is tuple $\langle S, A, R, P, \rho_0 \rangle$, where

- 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 the Markov property, which states that the history of previous states has no effect on the next state and that only the current state is considered for state transitions.

1.2 Single-agent environment

Having defined the mathematical model of the environment, let's review the related concepts. The whole problem of reinforcement learning (RL) can be best described by the following visualization.



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

Observability

State s contains all information about environment at given time. However, in some environment agent can perceive only **observation o** where some information about environment can be missing. In this case we say that environment is **partially observable** as opposed to **fully observable** environment where agent has all information available at it's observation. Nevertheless, this problem is more related to multi-agent environments discussed in the next chapter and not so much to single-agent settings.

Actions and policies

Environments can also differ in terms of what actions are possible within a given world. The 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 the action to be any real-valued number or vector.

The agent's choice of action can then be described by a rule called **policy**. A common notation is that if the action selection is deterministic, we say the policy is **deterministic** and denote by

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

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

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

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

Trajectory

The next important definition is the notion of trajectory, also known as episode or rollout. A trajectory is a sequence of states and actions in an environment.

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

The initial state s_0 of an environment is sampled from **start-state distribution**, denoted as ρ_0. Subsequent states follow the transition function of the environment. These may again be deterministic

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

or stochastic,

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

Return

We have already mentioned the agent's desire to maximize cumulative rewards. Now we combine it with trajectories and derive the formulation of **return**.

$$R(\tau) = \sum_{t=0}^{|\tau|} r_t \quad \text{(finite-horizon)}$$

$$R(\tau) = \sum_{t=0}^{|\tau|} \gamma^t r_t \quad \text{(infinite-horizon discounted return)}$$

Infinite-horizon discounting is both intuitive and mathematically convenient.

Optimal policy

In general, the goal of RL is to find such a policy that maximizes the expected return when acted upon. Suppose both the environment state transitions and the policy are stochastic. Then we can define the probability of the trajectory as

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

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

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

And finally, we can finish by defining the optimal policy

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

which is also an expression describing the central RL optimization 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 the possibility to say that in state s we take an arbitrary action a that does not necessarily have to come from policy π:

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

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

$$\begin{aligned} V^*(s) &= \max_{\pi} \underset{\tau\sim\pi}{\mathbb{E}}[R(\tau)|s_0 = s], \\ Q^*(s,a) &= \max_{\pi} \underset{\tau\sim\pi}{\mathbb{E}}[R(\tau)|s_0 = s, a_0 = a] \end{aligned}$$

Bellman equations

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

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

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

The most important theorem for us is the reformulation of Bellman's equations for optimal policies:

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

As we will see in the next section. The first RL algorithm will be a straightforward application of the Bellman equation for optimal policy.

Advantage function

Now that we've spent a few sections defining functions for the absolute value of actions or state-action pairs, it's worth considering relative value as well. 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, on average, relatively better than others. In other words, we want to know the relative advantage of a given action over others. For a given policy π, the advantage function $A^{\pi}(s,a)$ describes how much better it is to take action a over randomly sampled actions following policy π, assuming that policy π is followed in all subsequent steps. Mathematically, the 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. Reinforcement learning algorithms

In the previous chapter, we built the theoretical foundation of reinforcement learning. In this chapter, we translate that theory into practical algorithms. There are no precise classification boundaries between RL algorithms, as many of the techniques described are shared and crossed by all kinds of algorithms. Therefore, it is difficult to come up with an absolutely definitive taxonomy. However, for our purposes and the scope required, the following division into Q-learning and policy optimization is sufficient.

2.1 Q-Learning

Idea

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

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

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

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

And by computing the loss gradient, we arrive at the 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 2.1 algorithm bearing the same name.

Instability

The algorithm has rarely been used in this pure form. It has primarily been described for tabular methods, where the action-value function is represented by a table instead of a network approximator. In its simplest form, training is unstable and suffers from a number of significant shortcomings. Most notable 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. The value approximation condition is met because we use a Q-network to approximate the action value. Bootstrapping means that the estimate is used to compute the targets, this is also true in Q-learning for the same reason. Finally, the term off-policy stands for an approach where training data is collected using a different distribution than that of a target policy.

Algorithm 2.1: Q-learning

```
    Input: initial action-value aproximator Q parameters $\theta$ .
1   repeat
2       Observe state $s$ and select action $a$ according to $\epsilon$-greedy w.r.t. $Q$ e.g.
            $a = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \arg\max_a Q(s,a), & \text{otherwise.} \end{cases}$
3       Execute $a$ in the environment.
4       Observe next state $s'$, reward $r$ and done signal $d$ to indicate whether
         $s'$ is terminal.
5       if $d$ is true then
6           Reset environment state.
7       end if
8       Compute targets
9           $y(r, s', d) = r + \gamma(1 - d) \max_{a'} Q_\theta(s', a')$
10      Update Q-network taking one step of gradient decent on
            $(y(r, s', d) - Q_\theta(s', a))^2$
11  until convegence;
```

DQN

One of the most outstanding papers based on Q-learning was the algorithm Deep Q-learning 2.2, which demonstrated super-human results on several Atari games Mnih et al. [2015]. We give the pseudocode of the algorithm in it's original form. The notation may seem a bit different from ours, but 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 is sampled from this buffer, thus smoothing the training distribution over different past behaviors. However, probably the most important idea was the usage of a target Q-network, which broke the value approximation condition of the deadly triad, thus making the algorithm more robust. The target network is a copy of the original Q-network, with its parameters frozen and updated only once in a while based on the parameters of the main network. Its sole purpose is to compute target estimates that are not directly dependent on the Q-network function.

Rainbow

Deep Q Learning was a significant contribution that led to the study of further Q Learning capabilities. The Rainbow project Hessel et al. [2017] could probably be called the pinnacle of such research. In this paper, they further investigate several isolated ideas of possible improvements and try to combine them. To name a few, they use double Q-network to address the problem of maximization

Algorithm 2.2: Deep Q-learning with experience replay

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

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

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

bias and enhance sampling from experience buffer by considering the priority of stored individual data samples. Together with all the other improvements, they achieved state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance.

2.2 Policy gradient methods

2.2.1 Idea

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

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

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

Before converting this into an algorithm, we have to figure out how to compute the policy gradient numerically. Such an expression can be obtained as a result of the Policy Gradient Theorem (PGT).

TODO: Nasledujici veta a dukaz je inspirovany odtud `https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html` kde se odkazuji na puvodni literaturu, kde je skutecne PGT dokazany. Ale je tam pouze v Sumarni podobe s hodne predpoklady a celkove dost jinou reprezentaci. Nikde se tam veta nebo dukaz pro integral nevyskytuje. Dava mi intuitivne dost smysl dukaz prepsat z diskretniho stavu do spojiteho a tvarit se ze vse plati jako predtim, ale nevim, nevim, nepodarilo se mi najit nikde jinde odkaz na vysledek v teto podobe. Zajimave bylo kdyz jsem se koukal do diplomky od Honzy Uhlika, ze on to tam uvadi take v integralni podobe, zavadi si tam jeste distribuci pro pocatecni stav a do zneni vety rovnou zahrne i fakt za expected return lze nahradit za jinou nahodnou promenou nezavislou na akci - take tam ma pouze odkaz na tu samou puvodni literaturu. Tak nevim, jak se k tomu postavit. Jen tak prohlasit ze kdyz to plati pro diskretni podminky, tak ze to plati i pro spojite, mi pripada zvlastni.

2.2.2 Policy Gradient Theorem

Policy Gradient Theorem(Sutton and Barto [2018]). It holds:

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

This can be proven by rewriting the formula in the following way:

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

It is useful to have a look at the expression and what it represents. There are two important components of the expression: $\pi_\theta(a_t|s_t)$ and $R(\tau)$. Taking the gradient step of this objective, we are actually stating that we want to make the change in log probability $\pi_\theta(a_t|s_t)$ weighted by how good the expected return was. However, this may feel slightly counterintuitive, since the expected return takes into account all the rewards of the episode. We may want to restrict ourselves to the future consequences of a given action. Fortunately, it can be shown that $R(\tau)$ can be replaced by many other useful functions(Schulman et al. [2015b]):

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathbb{E}}\left[\sum_{t=0}^{|\tau|} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right],$$

where Ψ_t, can be one of the following:

$\sum_{t=0}^{|\tau|} r_t$: total reward of the trajectory

$\sum_{t=t'}^{|\tau|} r_t$: reward following action a_t

$\sum_{t=t'}^{|\tau|} r_t - b(s_t)$: baselined version of previous formula

$Q^\pi(s_t, a_t)$: state-action value function

$A^\pi(s_t, a_t)$: advantage function

2.2.3 Vanilla Policy Gradient

Common practice for policy gradient algorithms is to use some form of advantage function, where the baseline function is the value approximator $b(s_t) = V^\pi(s_t)$. The value approximator is typically represented by another neural network and is learned in parallel with the policy. There is a naming convention for the two types of networks. The policy network is usually called an actor since it's job is to provide a policy for the agents to act on. The value network, on the other hand, is often referred to as a critic, because it produces, in a sense, a critique of the value of the state. Incorporating the value approximator and the idea of the utility function reduces the variance in the sample estimation and leads to more stable and faster learning.

With all of this said, we present the first algorithm of this class.

Algorithm 2.3: Vanilla Policy Gradient Algorithm

```
1 Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2 for $k$ = 0,1,2,... do
3     Collect set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in
      the environment.
4     Compute rewards-to-go $\hat{R}$.
5     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage
      estimation) based on the current value function $V_{\phi_k}$.
6     Estimate policy gradient as
          $$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$
7     Compute policy update, either using standard gradient ascent,
          $$\theta_{l+1} = \theta_k + \alpha_k \hat{g}_k,$$
      or via another gradient ascent algorithm like Adam. Fit value
      function by regression on mean-squared error:
8         $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2,$$
      typically via some gradient descent algorithm.
9 end for
```

Policy gradient methods differ from Q-learning in several ways.

First, the absolute value of the objective function we are optimizing cannot be interpreted in terms of performance outcome. In fact, taking a step in the gradient descent does not even guarantee an improvement in the expected return in general. On a given set of samples, a value of even $-\infty$ can be achieved. However, the expected return of a changed policy would most likely be abysmal.

And second, policy gradient algorithms are **on-policy**, meaning that only data samples collected using the most recent policy are used for training. This is in contrast to the off-policy approach of Q-learning, where training data samples collected throughout the training process are used for learning steps. For this reason, policy gradient algorithms, especially vanilla policy gradient algorithms, are often considered sample-inefficient compared to off-policy algorithms.

2.2.4 Trust Region Policy Optimization

Although the standard policy gradient step makes a small policy change within the parameter space, it turns out that it may have a significant change on the performance difference. Therefore, vanilla policy gradient algorithm must be careful not to take large steps, making it even more sample-inefficient.

The next algorithm in the policy gradient family attempts to address this problem. As its name suggests, Trust Region Policy Optimization (Schulman et al. [2015a]) tries to take steps within the trusted region in which it is constrained so as not to degrade performance. TRPO proposes theoretical update of parameterized policy π_θ as

$$\begin{aligned}\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \\ \text{s.t.} \bar{D}_{KL}(\theta || \theta_k) \leq \delta\end{aligned}$$

where

$$\mathcal{L}(\theta_k, \theta) = \underset{s,a\sim\pi_{\theta_k}}{\mathbb{E}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \right]$$

is a surrogate advantage measuring the performance of the policy π_θ relative to the old policy π_{θ_k}.

And

$$\bar{D}_{KL}(\theta||\theta_k) = \underset{s\sim\pi_{\theta_k}}{\mathbb{E}} \left[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s)) \right]$$

is the average KL-divergence(Kullback [1959]) between policies evaluated on states visited by the old policy. However, working with theoretical updates of TRPO in this form is not an easy task. Therefore, approximations obtained by applying Taylor expansion around θ_k are being used:

$$\begin{aligned}\mathcal{L}(\theta_k, \theta) &\approx g^T(\theta - \theta_k) \\ \bar{D}_{KL}(\theta||\theta_k) &\approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)\end{aligned}$$

And the original problem can then be reformulated as an approximate optimization problem:

$$\begin{aligned}\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k) \\ \text{s.t.} \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leq \delta\end{aligned}$$

Such an approximate reformulation can be solved analytically using methods of Lagrangian duality (Rockafellar [1970]), yielding the solution:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

However, since we employed Taylor expansion, the approximation error could violate the KL divergence constraint. For this reason, TRPO incorporates the idea of backtracking line search (Armijo [1966]):

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

where $\alpha \in (0, 1)$ is the backtracking coefficient and j is the smallest non-negative integer such that the KL divergence constraint is fulfilled and the surrogate advantage is positive.

Algorithm 2.4: Trust Region Policy Optimization

```
1 Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2 Hyperparameters: KL-divergence limit $\delta$, backtracking coefficient $\alpha$,
    maximum number of backtracking steps $K$
3 for $k$ = 0,1,2,... do
4     Collect set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in
       the environment.
5     Compute rewards-to-go $\hat{R}$.
6     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage
       estimation) based on the current value function $V_{\phi_k}$.
7     Estimate policy gradient as
          $$\hat{g_k} = \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$
8     Use the conjugate gradient algorithm to compute
          $$\hat{x_k} \approx \hat{H}_k^{-1} \hat{g_k},$$
       where $\hat{H}_k^{-1}$ is the Hessian of the sample average KL-divergence.
9     Compute the policy by backtracking line search with
          $$\theta_{l+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$
       where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the
       sample loss and satisfies the sample KL-divergence constraint.
10    Fit value function by regression on mean-squared error:
          $$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2,$$
       typically via some gradient descent algorithm.
11 end for
```

2.2.5 Proximal Policy Optimization

Finally, we will conclude this chapter with last algorithm from family of policy gradient algorithms, which is considered in many aspects to be the state of the art algorithm. As the authors of this next algorithm state. With the leading contenders Q-learning, "vanilla" policy gradient methods, and trust region policy gradient methods, there is still room for the development of a method that is

- scalable - large models and parallel implementations
- data efficient
- robust - successful on a variety of problems without hyperparameters tuning

Q-learning is poorly understood and fails on many trivial problems. Vanilla policy gradient methods suffer from poor data efficiency and robustness. And trust region policy optimization is relatively complicated and not well suited to architectures including noise or parameter sharing.

Proximal policy optimization (Schulman et al. [2017]) aims at data efficiency and reliability of TRPO performance while using only first-order optimization. The authors propose a novel objective with clipped probability ratios, which provides a pessimistic lower bound on the performance of the policy.

Let $r_t(\theta)$ denote the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}.$$

Then the "surrogate" objective of TRPO can be expressed again in the form:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

CPI refers to conservative policy iteration (Kakade and Langford [2002]), where this objective was originally proposed. Maximizing L^{CPI} without any constraint would, as discussed in the previous section, lead to an excessively large policy update. Thus, the authors propose a new modified objective, called the clipped surrogate objective, which penalizes policy adjustments that move $r_t(\theta)$ far from 1.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right],$$

where the value $\epsilon = 0.2$ is empirically suggested. The objective can be intuitively explained as following. The first term in min is the previous L^{CPI}. And the second term modifies the surrogate objective by a clipping probability ratio that prevents r_t from escaping the interval $[1-\epsilon, 1+\epsilon]$. Finally, taking the minimum of the clipped and unclipped objectives makes the final objective pessimistically bounded on the unclipped objective.

The training process of PPO then proposes to alternately sample data from the policy and then perform several epochs of optimization steps on the sampled data. Note here that for each first training epoch it holds that $r_t(\theta) = 1$, so the objective in the first epoch always equals L^{CPI}.

Alternatively, the authors propose a second version of PPO that includes the KL penalty as part of the objective, making it even more similar to the idea proposed in TRPO. However, we will not cover more details here, as the authors themselves prefer the version including clipping of the surrogate objective.

When using a neural network architecture that shares parameters between the policy and the value function, a combined objective function that includes both the policy surrogate and the value function error term must be applied.

$$L_t^{CLIP+VF}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) \right],$$

where c_1 is the coefficient and L_t^{VF} is a squared error loss $(V_\theta(s_t) - V_t^{targ})^2$.

Algorithm 2.5: Proximal Policy Optimization

```
1 Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2 for $k$ = 0,1,2,... do
3     Collect set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in
        the environment.
4     Compute rewards-to-go $\hat{R}_t$.
5     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage
        estimation) based on the current value function $V_{\phi_k}$.
6     Update the policy by maximizing the PPO-Clip objective:
          $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$
        typically via stochastic gradient ascent with Adam.
7     Fit value function by regression on mean-squared error:
          $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2,$$
        typically via some gradient descent algorithm.
8 end for
```

Lastly, one problem we haven't explicitly addressed is that of exploitation and exploration. In general, during training we always try to strike a balance between exploiting learned experience and exploring new possibilities. In the case of Q-learning, this is most often done artificially by using the ϵ-greedy approach, where with probability $1 - \epsilon$ we exploit our knowledge by taking $\arg\max_a$ action according to the Q value. And with probability ϵ we explore by taking a random action.

Sampling according to the policy π_θ in policy optimization methods solves this problem more naturally. Typically, at the beginning of the training process, when the parameters are initialized, the parametrized probability distribution is close to uniform, which implicitly makes the action selection mechanism exploratory. And as the policy is updated, it becomes more specialized towards the optimal policy, effectively forcing the action selection to be more exploitative. Unfortunately, the exploration described in policy optimization methods is often insufficient, and policies tend to get stuck at bad local optima despite an initial uniform distribution. On this account, an exploration mechanism in the form of bonus entropy(Williams [1992]) is often employed and also recommended by the PPO authors. By adding a small bonus for policy entropy, the policy is slightly forced in the direction of uniform distribution, shifting towards exploratory behavior. Thus, the final PPO objective may have a form as follows:

$$L_t^{CLIP+VF}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right],$$

where c_2 is another coefficient and S denotes an entropy bonus.

Q-Learning and Policy optimization

So far we have covered the simplest possible division rule between types of reinforcement learning algorithms. However, there are actually several algorithms that combine both Q-learning and policy optimization. In these algorithms, both the Q approximator and the policy approximator are learned simultaneously. To name just a handful of the more popular ones: Deep Deterministic Policy Gradient (DDPG, Lillicrap et al. [2015]), Twin Delayed DDPG (TD3, Fujimoto et al. [2018]), Soft Actor-Critic (SAC, Haarnoja et al. [2018]). We won't cover them in detail, as they are beyond the scope of our needs.

3. Multi-agent environment

In this chapter, we revisit concepts from the previous section and extend them to multi-agent settings. We introduce theoretical definitions for various types of settings and provide possible schemes for concurrent learning of multiple agents. And at the end, we mention two popular RL approaches in the field of MARL.

3.1 Multi-agent Markov Decision Process

So far, all the theory and algorithms that have been built have revolved around environments where there is a single agent. However, this is rarely the case in real-world problems. Much more often we encounter environments with multiple agents operating within them. In the general case, the agents are heterogeneous, which means that the agents may have different goals. Nevertheless, we will mostly focus on environments that are fully cooperative, meaning that the utility of any given state of the system is equivalent for all agents.

With this setting we can extend the definition from the MDP section 1.1:

Definition

Multiagent Markov Decision Process(Boutilier [1996])is a tuple $\langle n, S, \mathcal{A}, T, R \rangle$

- n is the number of agents
- S is the set of all valid states,
- $\mathcal{A}$ is the set of joint actions
- $T : S \times \mathcal{A} \times S \to [0, 1]$ is the transition function
- $R : S \to \mathbb{R}$ is a real-valued reward function, where reward determined by the reward function $R(s)$ is received by the the entire collection of agents, or alternatively, all agents receive the same reward.

3.2 Decentralized Partially Observable MDP

Unfortunately, we cannot be satisfied with this definition, since MMDP provides a description that is far too idyllic for real-world problems. The MMDP model presumes that the state of the environment can be globally accessed by all agents. However, this is rarely the case in multi-agent environments. In general, the world can be of high complexity, and combined with limited sensory capabilities, the agent may perceive only limited observations describing only a part of the entire environment state. Therefore, we extend our model definition to include the concept of partial observability.

Definition

Decentralized partially observable Markov decision process (Dec-POMDP, Oliehoek et al. [2008]) is tuple $\langle n, S, \mathcal{A}, P, R, \mathcal{O}, O, h, b^0 \rangle$ where,

- n is the number of agents
- S is the set of all valid states,
- $\mathcal{A}$ is the set of joint actions
- $P : S \times \mathcal{A} \times S \to [0, 1]$ is the transition function
- $R : S \to \mathbb{R}$ is the immediate reward function
- $\mathcal{O}$ is the set of joint observations
- $O : \mathcal{A} \times S \to \mathcal{P}(\mathcal{O})$ is the observation function
- h is the horizon of the problem
- $b^0 \in \mathcal{P}(S)$, is the initial state distribution at time $t = 0$

We define $\mathcal{A} = \times_i \mathcal{A}^i$, where $\mathcal{A}^i$ is the set of actions available to agent i. Similarly, $\mathcal{O} = \times_i \mathcal{O}^i$, where $\mathcal{O}^i$ is the set of observations available to agent i. Note that even this definitional extension does not provide us with a model capable of describing an environment where agents have different reward functions, which is needed for situations where agents have different goals or are even competing. This extension is possible with the definition of Stochastic game (Shapley [1953]). However, we don't need to provide the formal definition and extend our models, since the primary goal of this work will focus mainly on the cooperative setting where all agents have a common goal.

If the observation satisfies the condition that the individual observation of all agents uniquely identifies the true state of the environment, the environment is considered fully observable and such a Dec-POMDP can be reduced to MMDP.

Notation

To denote common entities, we will use bold: $\boldsymbol{a} = (a^1, ..a^n) \in \mathcal{A}$. The common policy $\boldsymbol{\pi}$ induced by the set of individual policies $\{\pi^i\}_{i \in n}$ gives the mapping from states to common actions. Now we can use the similar notation as in the first chapter by using the bold symbols:

$$\begin{aligned}
\text{Trajectory :} \quad & \boldsymbol{\tau} = (s_0, \boldsymbol{a_0}, s_1, \boldsymbol{a_1}, ...) \\
\text{Return :} \quad & R(\boldsymbol{\tau}) = \sum_{t=0}^{\boldsymbol{\tau}} r_t \\
\text{Probability of trajectory :} \quad & P(\boldsymbol{\tau}|\boldsymbol{\pi}) = \rho_0(s_0) \prod_{t=0}^{|\boldsymbol{\tau}|} P(s_{t+1}|s_t, \boldsymbol{a_t}) \boldsymbol{\pi}(\boldsymbol{a_t}|s_t) \\
\text{Expected return :} \quad & J(\boldsymbol{\pi}) = \int_{\boldsymbol{\tau}} P(\boldsymbol{\tau}|\boldsymbol{\pi}) R(\boldsymbol{\tau}) = \mathop{\mathbb{E}}_{\boldsymbol{\tau} \sim \boldsymbol{\pi}} [R(\boldsymbol{\tau})]
\end{aligned}$$

Given the joint utility function, it is useful to think of the collection of agents as a single agent whose goal is to produce an optimal joint policy. The problem with treating MMDP as a standard MDP where actions are distributed lies in coordination. In general, there are several different optimal joint policies. However, even if all agents choose their individual policies according to some optimal policy, there is no guarantee that they all pick from the same optimal joint policy. Such a final joint policy may be nowhere near the optimal one, and most likely even produce significantly worse performance. In theory, there are two simple ways to ensure optimal coordination. First, there could exist a central control mechanism 3.3 that can compute the joint policy and then communicate the result actions to all individual agents. Or second, each agent can communicate its choice of individual policy to the others. However, neither of these approaches are feasible in the real application.

3.2.1 Nash Equilibria

Alternatively, we can look at the MMDP from the perspective of an n-person game. Then the problem of determining an optimal joint policy can be viewed as a problem of optimal equilibrium selection. A Nash equilibrium (Nash [1950]) for a $\boldsymbol{\pi}^*$ can be defined as:
A set of policies π^* is a Nash equilibrium if:

$$\forall i \in n, \forall \pi^i : J(\boldsymbol{\pi}^{*-i}, \pi^{*i}) \geq J(\boldsymbol{\pi}^{*-i}, \pi^i)$$

However, not all Nash equilibria are optimal joint policies, as some may have lower utility than others. This makes multi-agent environments more sensitive to convergence to local suboptimal policies, as the only way to escape such an equilibrium is through coordinated modifications of all individual policies.

3.3 Learning schemes

Centralized scheme

From a theoretical point of view, we could consider MMDP as an instance of single-agent MDP, where the goal would be to learn a central joint policy, which would then be distributed among individual agents. With this idea, we could solve MMDP problems using the same single-agent RL algorithms from the second chapter, only instead of learning a single policy, we would learn a joint policy. However, this approach has several shortcomings in real-world cases.

First, from a practical point of view, agents in this scenario would be somewhat passive entities whose job would only be to report perceived observations to some central authority. After all the observations have been collected by the central unit, a joint policy is constructed and distributed to the agents, who then blindly act as instructed by the central control. This has several serious problems. Imagine that some failure of the central mechanism occurs. Suddenly, the whole system of agents collapses because all agents lack individual autonomy.

Second, such an intensive communication between all agents and the central control mechanism may be too demanding or not even plausible for technical reasons.

Finally, from a theoretical point of view, the representation of such a joint policy grows exponentially with the number of agents with respect to both observations ($\prod_{i=0}^{n} |\mathcal{O}^i|$) and actions ($\prod_{i=0}^{n} |\mathcal{A}^i|$), which makes it unscalable.

Concurrent scheme

At the other end of the spectrum is the concurrent scheme. Here, all kinds of global information are omitted and agents rely solely on their local observations. The training of such an agent is then completely independent.

Centralized training with decentralized execution

And finally, somewhere in between the concurrent and centralized schemes, we can identify the so-called centralized training with decentralized execution. Here we consider the two life stages of agents. First, we train our agents in safe laboratory conditions, and only after the training is complete are they deployed in the real world.

During training, we can take advantage of the fact that we presumably have more information about the state of the environment at our disposal. Whether it is the true global state of the environment, local observations of other agents, or actions taken by others. All of this additional information can be incorporated into the training process to capture the true state to act upon. In other words, we want to make the training process as simple and accurate as possible.

Once the agents are deployed, they again depend solely on their local observations.

This learning scheme is often used in RL algorithms where both actors and critics are employed. For example, this is reflected in the PPO 2.5 algorithms mentioned above. During training, both actor and critic are trained in parallel, and the value estimate obtained by the critic can be used to provide more accurate information about the real value of the given state, making the actor's policy update more stable and accurate. Once training is complete, the agents, provided with local observations, are asked to take action based solely on the actor.

3.4 Non-stationarity

In addition to all the problems of partial observability, local equilibria, and policy distribution that have been mentioned so far there is another important problem that is not present in single-agent settings. Although we are able to reproduce the value function expressions:

$$\begin{aligned} V^{\boldsymbol{\pi}}(s) &= \mathop{\mathbb{E}}_{\boldsymbol{\tau}\sim\boldsymbol{\pi}}[R(\boldsymbol{\tau})|s_0 = s] \\ Q^{\boldsymbol{\pi}}(s,\boldsymbol{a}) &= \mathop{\mathbb{E}}_{\boldsymbol{\tau}\sim\boldsymbol{\pi}}[R(\boldsymbol{\tau})|s_0 = s, \boldsymbol{a_0} = \boldsymbol{a}] \\ A^{\boldsymbol{\pi}}(s,\boldsymbol{a}) &= Q^{\boldsymbol{\pi}}(s,\boldsymbol{a}) - V^{\boldsymbol{\pi}}(s) \end{aligned}$$

due to the non-stationarity problem, we cannot obtain the optimal value using Bellman equations as it was done in the single agent setting. Non-stationarity refers to the fact that changing policies of other agents as a consequence changes, from a fixed agent's perspective, the state transition and the reward function of the environment. Using the idea of Bellman equations for optimality is still possible. However, in a multi-agent setting, we lose the theoretical basis that promises convergence to the optimal policy, and training using the Q-learning update rule must be accompanied by some mechanisms to overcome the non-stationarity problem.

3.5 RL algorithms

Having extended our mathematical model to settings that satisfy the conditions for a multi-agent environment, we can continue with a brief mention of the multi-agent variants of reinforcement learning algorithms discussed in the previous chapter.

MADDPG

The first of the algorithms mentioned is the Multi Agent Deep Deterministic Policy Gradient (MADDPG, Lowe et al. [2017]) algorithm. The authors extend DDPG (briefly mentioned in 8) by using centralized learning with decentralized execution. To address the problem of non-stationarity, the authors propose sampling from the ensemble of policies for each agent to obtain more robust multi-agent policies. This algorithm has been extensively experimented in academia with respect to multi-agent coordination environments.

MAPPO

Finally, we want to mention the recent success in the form of Multi Agent Proximal Policy Optimization (MAPPO, Yu et al. [2021]) variant of the PPO 2.5 algorithm. The authors revisit the use of PPO in multi-agent settings. In recent years, MADDPG has usually been the first choice of algorithm when dealing with multi-agent environments, leaving PPO ommitted. The authors hypothesize that this is due to two reasons: first, the belief that PPO is less sample-efficient than off-policy methods, and second, the fact that common implementation and hyperparameter tuning techniques when using PPO in single-agent settings often do not yield strong performance when transferred to multi-agent settings.

The authors propose five important changes with respect to adaptation to multi-agent settings.

- Value normalization:
  employing value normalization using Generalized Advantage Estimation (Schulman et al. [2015b])

- Input Representation to Value Function:
  enhanced global state with agent-specific features are used for critic, where such representation does not have substantially higher dimension

- Training Data Usage:
  using smaller amount of trainig epochs to avoid non-stationarity problem and not splitting data to mini-batches to make policy update more accurate

- PPO Clipping:
  maintain a clipping ratio ϵ under 0.2, within this range, tune ϵ as trade-off between training stability and fast convergence
- PPO Batch Size:
  utilize a large batch size

Using these five concepts, they were able to achieve comparable or superior results to off-policy algorithms on several benchmark frameworks without any other domain or structural modifications.

4. Overcooked environment

4.1 Overcooked game

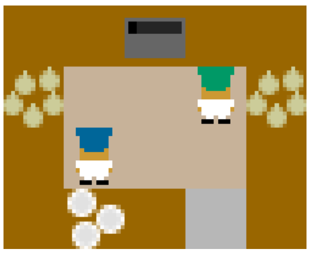
Before we get into our problems with cooperation let us first examine the environment. We will be working with environment based on popular cooking video game `https://ghosttowngames.com/overcooked/`. Overcooked is multiplayer cooperative game where the goal is to work in a kitchen as a team with partner 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

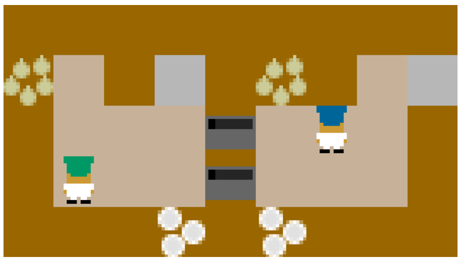
Although the Overcooked implementation has its own generator that can be used to generate new random kitchen layouts, the majority of the related scientific work has so far experimented with a fixed set of predefined layouts, where each of them capture some important aspect of coordination.

Cramped room



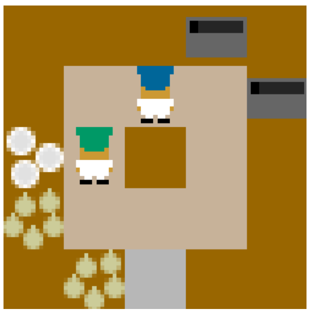
Cramped room as a name suggests represents cramped kitchen layout where all important places are relatively easy to reach. Challenge lies in low level coordination of movement with the other partner as there is no spare room.

Assymetric advantages



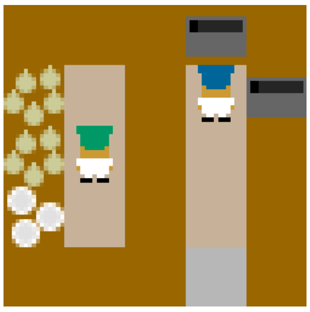
In Assymetric advantages both players are located in separated regions where each region is fully self-sustaining. However, each region has better potential for specific subtask. And it is only when both players make the most of their own region's potential that the maximal shared efficacy is reached.

Coordination ring



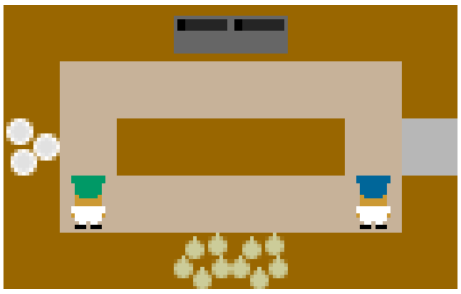
The Coordination ring is another example of a layout where clever coordination is required as the only possible movement around the kitchen is along a narrow circular path that can be used in a given direction. For example, if one player decides to move in clockwise direction, the other player would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

Forced coordination



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.

Counter circuit



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 Action space

Action space is quite trivial as it contains only essential actions needed to operate within this environment.

- Go north
- Go south
- Go east
- Go west
- Stay
- Interact

4.3.2 State representation

There are two state representation functions prefabricated by the authors, namely `featurize_state` and `lossless_state_encoding`.

First of the two extracts manually designed features into the single dimension vector of ones and zeroes. These features can be interpreted as partial observable representation of the environment state as majority of features are computed in relation to the closest point of interest. For instance only closest source locations of onions and dishes related to current player location are included, which results in loss of global information about whole layout.

Second of the mentioned state representation, as the name suggests, provides lossless global information about environment state. As in previous case, variables about environment are also represented by ones and zeros. However, contrary to previous function, resulting representation is not a single dimension vector. Result is formed by stacked masks, where each mask represents some feature of the environment in the form of two dimension vector corresponding to layout width and height. For instance, the mask representing player i location is vector of shape $(width, height)$ filled with zeroes on all positions except for the value one at coordinates where player i is located. Similarly the mask of the same shape representing onion sources is filled with zeroes and on all coordinates where there is an onion source in the environment there are ones instead. We can see that this representation goes beyond the closest locations and provide global informations.

4.3.3 Rewards

As we already said in the introduction, the environment is purely cooperative in a sense that players share the common reward of value 20 every time a soup is delivered to counter location. And this reward is player independent. The cycle of the environment is virtually infinite as there is no a priori terminal state. Hence it could be theoretically possible to easily obtain cummulative sum reward of infinity. This is prevented be setting finite time horizon that is set by a convention to the limit of 400 steps. This way it makes sense to compare the results of different runs.

Apart from the common reward for delivering soups, there can be utilized player-dependent partial rewards that are not considered in overall cummulative sum of rewards. These rewards are utilized mostly for learning purposes of the agents as especially on some particular maps it is extremly unlikely that by following initially random policy whole process of soup making including delivery will be completed. Using predefined partial rewards

- Dish pickup reward - dish is picked in usefull situation, eg. pot is ready or cooking
- Soup pickup reward
- Placement in pot reward - usefull ingredience is added to the pot, in our setting only onion soup recipe is used, so this corresponds to the action where onion was put in the pot

allows agents to learn subtasks first. It is important to eventually ignore these partial rewards during training, as agents could focus on these subtasks and ignore the main goal thus failing the main task whatsoever. This is usually implemented by linearly decreasing the weight of partial rewards over some finite time horizon.

4.3.4 Initial state

As there were two functions predefined by the authors to represent the environment state, there are also two ways how the initial state of the environment can be created.

First of the two uses fixed layout initial player locations, which always create identical initial state including both players initial locations. However, learning single agent by always setting him in same initial location will probably cause him to fail when initially placed in second location. For this reason the $player\_index$ is introduced to specify which agent is interpreted as player number one and player two. This index is being randomized every time the environment resets, this way it is ensured that agent encounters both starting locations during learning. Using this index introduces a mild chaos into the environment as multiple parts of environment control must adress the problem of the index and both actions and state observations must be switched to correct order, which makes it a bit opaque. We suppose that this kind of initial state is suitable for some sort of benchmarking where the initial conditions are always the same.

However, in our experiments we utilize the second approach to state initialization, where initial locations are always sampled randomly for both players. We claim, that this is in agreement with our intuition as robust cooperative agents should be able cooperate well no matter their starting position, which has been partially demonstrated (Knott et al. [2021]). However, this might come along with the cost of decreased average performance. Apart from random locations, the randomized initialization function offers also probabilty threshold argument that can be used to randomly initialize some random objects within the environment.

5. Related work

Common approach when dealing with two-player game is to train agent by confronting him with a set of other AI agents. This has been shown to perform impressively well against human experts in multiple complex games like Dota (OpenAI [2019]) or Starcraft (Deepmind [2019]). The authors of overcooked environment believe (Carroll et al. [2020]) that the distributional shift from AI training to Human evaluation is successful due to the fact that the nature of these environment is strongly competitive. This is ilustrated by the canonical case of a two-player zero-sum game. When humans take branch in search tree that is mistakenly suboptimal while being in the role of minimizer, this only increases the result for the maximizer.

However, the situation is not this ideal in case of common-payoff games such as overcooked environment. If a self-play trained AI agent is paired with another sub-optimal partner the result can be abysmal. In this case both agents play maximizing role in search tree. When self-play AI agent is expecting it's sub-optimal partner to be the same agent, it can choose branches that lead to maximal common payoff. However, since sub-optimal partner may behave sub-optimally, it can choose by mistake some worse branches unforeseen by self-play agent, which can lead to way worse result. However, since common payoff is shared between the two agents, this is no longer advantage for the self-play maximizer agent. It rather causes failure for both agents.

5.1 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 (Carroll et al. [2020]). They collected several human-human episodes and incorporated these experiences during training. These human data have been used to create human-based models using behavior cloning and Generative Adversarial Imitation Learning (Ho and Ermon [2016]) and consequently incorporated to the training. One of the important conclusion was that when self-play and population-based agents were paired with human models the overall results were way worse than when paired with agents designed to play with human models.

5.2 AI-AI cooperation

We already mentioned the problem when self-play agent is paired with sub-optimal (eg. human) partner. Nevertheless, similar problem can occur even when two different agents trained in self-play mode are paired together. Common approach used in competetive setting is to introduce diversity through exposing trained agent to diverse set of partners during training phase. There are several frequently used methods for partner selection during training.

Self-play

Self-play method was already mentioned, it is argubaly the simplest and most straightforward method for partner sampling. As the name suggest, the idea of this approach is that the agent that is currently being learned is paired during training with the copy of itself. This method does not introduce any kind of diversity into the learning process as it learns exclusively from it's own current policy.

Partner Sampling

Partner sampling is extension of the previous method. However, Instead of playing solely with current policy, partner is sampled from previously periodically saved policies of this given training period. The diversity in this method comes from the theoretical point of view that previous policies correspond to different behaviors.

Population-Based Training

Population-Based Training method is based on evolutionary algorithm focusing on training hyperparameters and model selection. Population consists of agents parametrized by neural network and trained via DRL algorithm. During each iteration agents are drawn from the population and are being trained using the collected trajectories. Pair-wise results are recorded and at the end of the iteration the worst agents are replaced by a copy of best ones with their hyperparameter mutated.

Pre-trained Partners

Lastly, remaining often used method consists of using previously pre-trained partners. Diversity is here expected due to the fact that different runs of reinforcement learning algorithms often yield different agent behaviors.

Result

Problem with fully cooperative games is that when agents are trained together they tend to exploit the common knowledge experienced during training, which often makes them unable to cooperate with unseen agents. It has been shown (Charakorn et al. [2020]) that this strongly holds for the first three mentioned methods, where when performing cross-play evaluation of the set of different agents obtained by the same method, only such pair of agents that have been explicitly trained together, managed to perform well, while the remaining pairs failed.

Nemam si tady od nich z toho clanku pujcit primo ty obrazky ukazujici ten SP off-diagonal failure, abych se na to pak mohl vizualne odkazat ve svych experimentech? Nebo krast obrazky se nedela? The authors conclude with results showing that incorporating different pre-trained agents for training robust agents was significantly more successful than other methods.

5.3 Problem of robustness

Another important problem regarding common-payoff is performance evaluation of the trained agents. As discussed already, we cannot rely solely on training performance, no matter what metric was chosen, as training set of agents exploit shared knowledge thus making the training performance appear excelent. As noble as it sounds, it is quite complicated to define what agent behavior is actually robust. Obvious desirable goal is to come up with such trained agent that will be capable of cooperation with ideally all possible behaviors of it's partner. However, since it is difficult to come up with diverse set of agents for partner sampling during training, exactly the same holds also for the evaluation agents.

Average performance

First obvious choice of evaluation metric could be to look at the average evaluation performance over the evaluation set of agents. Nevertheless, it is questionable whether this answers question of robustnes. In one scenerio we could theoretically achieve result where our agent failed to cooperate with half of evaluation agents completely while performing excelently with other half. In another case we could achieve sort of medium performance with all of the evaluation agent set, where the cooperation does not fail completely but the performance is nowhere near the optimal values neither. However, in both cases the average evaluation performance reaches similar values.

Threshold performance

Another possible approach for the given layout could be to define our own threshold value that could practically be equivalent to some number of delivered soups. Evaluation metric could be then simply expressed in number of evaluation agents that managed to reach such threshold when paired with our trained agent.

Edge case testing

The problem with previous metrics is that these are reasonable only in situation where the evaluation set of agent is complex enough in terms of behavior diversity, which also implies condition on sufficient size of such set. However, creating evaluation set satisfying this condition is practically impossible.

Rather than trying to create such evaluation set and evaluate whole episodes in effort to test cooperation on the run, we could break down the interesting cooperation challenging situation into seperate tests (Knott et al. [2021]). Suddenly our thinking about robustness shifts from looking at the cooperation from the point of result of entire episode to just separate small modular situations where desired correct behavior of both agents can be described. This approach is inspired by unit testing in software development where even though the program behaves correctly in vast majority of cases it can still produce undesirable reaction in edge cases.

Similarly to unit test we can define wide set of situations (eg. instances of overcooked environment states) and pair them with set of acceptable consequent behavior of agents which can be classified as robust reaction. Evaluation using such metric can then be expressed as number of passed tests.

However, this evaluation metric has also several challenges as these edge cases have to be designed manually, in many scenarios it is difficult to say with certainty what is the correct expected behavior, and lastly, similar to unit testing in software development, there is never enough defined edge cases to cover all of the possible situations.

6. Our work - Preparation

Before we get into our own experiments with training a robust agent, which will be main focus in the next chapter, we want to prepare our solution first and reimplement some of the aforementioned results regarding self-play agents.

6.1 Framework

Writing all our code on top of the overcooked environment is not necessary as there already exist several frameworks implementing deep reinforcement learning algorithms. Although there are in general many more DRL frameworks that could be suitable for our purposes, most of related projects concerning overcooked environment utilized either library RLib or StableBaselines.

As authors themself say, RLlib offers support for production-level and highly distributed RL workloads. This framework is also suitable for multi-agent learning. The downside here is that it is not so much suitable for smaller projects and developing on a local machine, as there is lot of a overhead due to it's paralel capabilities towards cluster computing. I found the framework a bit intimidating and documentation a little confusing.

Other option was the Stable baselines framework, which in my opinion offers a documentation that is clearer and easier to understand. Also, in my humble opinion, the code base structure is more transparent and basic API is very simple. To demonstrate its simplicity, once you have environment implementing standard RL OpenAI environment interface, you can perform whole training with default hyperparameters and policy represented by multi layered perceptron network using PPO algorithm as simply as follows:

```
env = make_vec_env("CartPole-v1", n_envs=4)
model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=25000)
```

The downside here is that the framework does not have native support for simultaneous multi agent learning.

6.1.1 Modifications

After considering both frameworks and extent of our project we decided build our project withing the stable baselines framework. We will not cover all the details regarding our diverse population approach yet as there will be plenty of room for this in next chapter. Here we just want to mention few of the essential framework modification that have to be done in order to bend the framework toward our purposes.

Embedding partner

First of all as it was already mentioned, stable baselines framework does not have support for multi agent learning. In our approach we will make slight simplification by theoretically transforming the environment from multi agent to single agent settings. Obviously we will not reduce the number of players in the environment. We will rather look at the situation from the point of view of the single agent that is being learned. This way we can look at the environment as if the partner cook is just embedded as part of the system.

And exactly the same modification has to be made to the framework. During one training run the partner is embeded as part of the environment and every iteration when the actions are being sampled. Action of trained agent are sampled normally as expected and actions of partner are sampled according to the policy of embedded partner.

This mechanism allows us to use any partner sampling method (5.2) including self-play where the same instance of parametrized policy as the one that is being learned is used.

Resulting two actions are concatenated together and passed into the environment as expected.

Convolution policy

By default stable baselines offers multiple types of parametrized policy representations including fully connected dense multi layered perceptron network (MLP) and convolutional neural network (CNN). However, regarding the CNN representation, when using this wrapper the framework expects the inputs to be strictly in some standard format of image (RGB, RGBD, GrayScale). Which is unfortunately accompanied by using unsufficient assumptions trhoughout the code base heuristially expecting the inputs in certain format. For instance function for detection if image space is in channels first format only checks if the first dimension is the smallest one. Which mistakenly returns true for some overcooked layouts using lossless state representation (4.3.2) where width is smaller than both height and also the number of stacked masks. However, in lossless encoding masks are equivalent to channels and are represented by the last dimension. Authors could argue that with such state representation the inputs are not technically image therefore such sanity checks expecting images should not be used in the first place. However, this is the problematic part as I was unable to find if framework allows some options how to bypass these constraints. Eventually after disabling these assertions in certain critical places, the code managed to work with our convolutional representation flawlessly without any need of further modifications.

Loss and rewards augmentation

Lastly, two important aspects of our experiments will be augmentation of rewards and extending the objective loss function. These modifications are not really related solely to stable baselines as these modification would have to be made no matter the chosen framework, we just want to mention it here to make modification list complete. Fortunately, these modifications were also quite straightforward as both of these factor are nicely separated in transparent easily extandable places.

6.2 Self-play

Before diving into our experiments we wanted to make sure our solution is ready to be used with the environment. Therefore we decided to reimplement basic agent training using self-play (5.2) and demonstrating it's supposedly poor cooperation abilities.

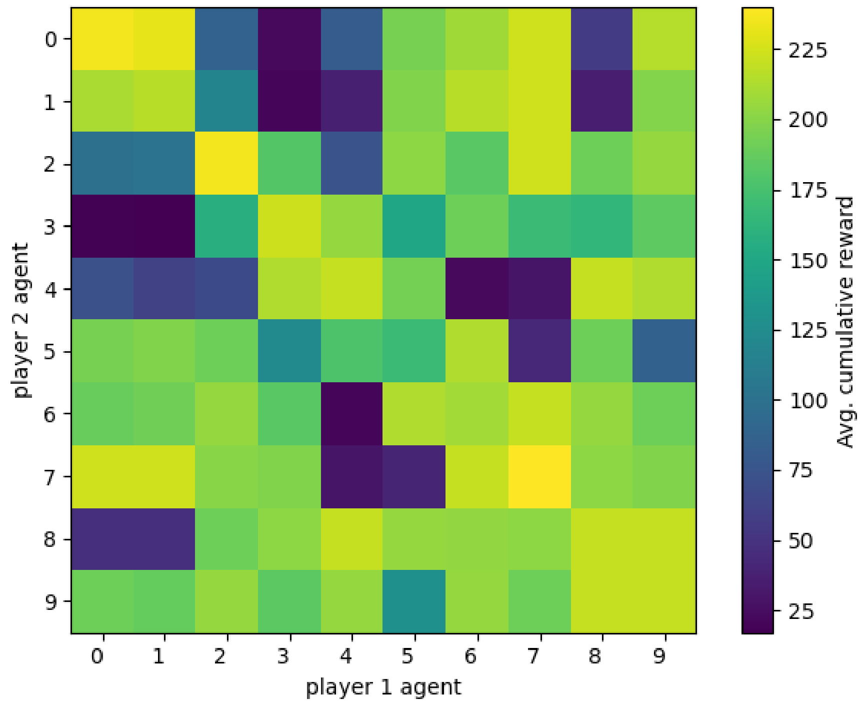


Figure 6.1: Self-play MLP cross play evaluation

Cross-play evaluation of 10 agents trained via self-play method on cramped_room layout, where policy is parametrized by MLP

6.2.1 NN structure modification

6.2.2 Hyperparameters random search

6.2.3 Randomization function correction

7. Our work - Contribution

7.1 Idea

In this chapter, we propose a novel approach to injecting explicit diversification into agent behavior. Although the traditional population-based methods discussed in the previous chapter were not effective enough to add the necessary diversification to the population, we try to use them in a slightly different way.

We will transform population learning into an incremental process, where only newly added agents are learned. This process is designed to be quite general with respect to the potentially pre-existing set of autonomous agents.

The building process will start with a population of whatever pre-trained agents are available, no matter what the source of agents is. Consequently, a new agent is learned with respect to the previous population by using some diversification techniques. After the agent is trained to cooperate with individual agents within the existing population while maintaining its diversity, it is added to the fixed population and the next agent can be trained.

By fixing the previously trained agent, we effectively solve the problem of non-stationarity, since no two agent policies are changed at the same time. By embedding sampled partners from the population into the environment, we can consider the environment as a single agent from the point of view of the agent being trained. However, despite the fact that the original overcooked environment is deterministic from the point of view of the transition function, by embedding partners whose policies will be mostly stochastic into the environment, the transition function becomes stochastic from the point of view of the trained agent. The same applies to the reward function. Since part of our proposed diversification methods involve reward augmentation based on the agent's policy, the environment's reward function will no longer be stationary throughout training due to the learning policy.

Po prostudovani a sepsani MARL kapitoly mi pripada, ze muj pristup popsany vyse je vlastne hrozne slaby, obzvlaste tim, ze trenuju ciste reaktivni agenty bez jakekoliv pameti (uvazuji jen soucasnou CNN reprezentaci stavu, bez RNN, bez predeslych akci partnera, bez historie predchozich stavu), tim vlastne ani nejsem schopny komplexne zachytit nejakou high-level strategii partnera, namisto toho se proste ucim jak se nejlepe (robustne?) zachovat v jednom konkretnim stavu vzhledem k potencialne vice moznych strategii, kterymi by se mohl partner v danem stavu ridit. Tim, ze CNN kompletne popisuje stav prostredi, tak ani nemam partial observability a vstup kritika jsem nijak nerozsiroval. Proste pripada mi, ze jsem v tom reseni nezahrnul moc zadne techniky popsane pro MAS.

Pripada mi ze i tak ty experimenty ktere zkousim a navrhuju jsou zajimave, jen mi pripadaji takove vytrzene z toho MAS kontextu - ale o tom jsme si myslim minule bavili ze nevadi.

Diverse population can be thought of as domain randomization technique "Given just this, it is unclear what the agent should do: the optimal policy for the agent depends heavily on the human's policy, which the agent has no control over"

"From the perspective of game theory, we are interested in n-person games in which the players have a shared or joint utility function. In other words, any outcome of the game has equal value for all players. Assuming the game is fully co- operative in this sense, many of the interesting problems in cooperative game theory (such as coalition formation and ne- gotiation) disappear. Rather it becomes more like a standard (one-player) decision problem, where the collection of n play- ers can be viewed as a single player trying to optimize its be- havior against nature."

"Solutions to the coordination problem can be divided into three general classes, those based on communication, those based on convention and those based on learning"

Convention probably does not make sense as we have ad hoc partner Craig Boutilie 1996

7.2 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?

7.3 Population construction

7.3.1 SP agents initialization

One agent is not enough?

7.3.2 population partner sampling during training

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

7.3.3 Final agent training

7.4 Diverzification

maximize kl divergence among population partners policies

7.4.1 Population policies difference rewards augmentation

7.4.2 Population policies difference loss

Conclusion

Bibliography

Josh Achiam, 2018. URL `https://spinningup.openai.com/en/latest/`.

Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1 – 3, 1966. doi: pjm/1102995080. URL `https://doi.org/`.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '96, page 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1558604179.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination, 2020.

Rujikorn Charakorn, Poramate Manoonpong, and Nat Dilokthanakul. Investigating partner diversification methods in cooperative multi-agent deep reinforcement learning. In Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King, editors, *Neural Information Processing*, pages 395–402, Cham, 2020. Springer International Publishing. ISBN 978-3-030-63823-8.

Deepmind, 2019. URL `https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii`.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL `https://arxiv.org/abs/1802.09477`.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018. URL `https://arxiv.org/abs/1812.05905`.

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`.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

Paul Knott, Micah Carroll, Sam Devlin, Kamil Ciosek, Katja Hofmann, A. D. Dragan, and Rohin Shah. Evaluating the robustness of collaborative agents, 2021.

Solomon Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015. URL `https://arxiv.org/abs/1509.02971`.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017. URL `https://arxiv.org/abs/1706.02275`.

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.

John F. Nash. Equilibrium points in ¡i¿n¡/i¿-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. doi: 10.1073/pnas.36.1.48. URL `https://www.pnas.org/doi/abs/10.1073/pnas.36.1.48`.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, may 2008. doi: 10.1613/jair.2447. URL `https://doi.org/10.1613%2Fjair.2447`.

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

R. Tyrrell Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, N. J., 1970.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015a. URL `https://arxiv.org/abs/1502.05477`.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2015b. URL `https://arxiv.org/abs/1506.02438`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

L. S. Shapley. Stochastic games*. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi: 10.1073/pnas.39.10.1095. URL `https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095`.

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

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2021. URL https://arxiv.org/abs/2103.01955.