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 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.

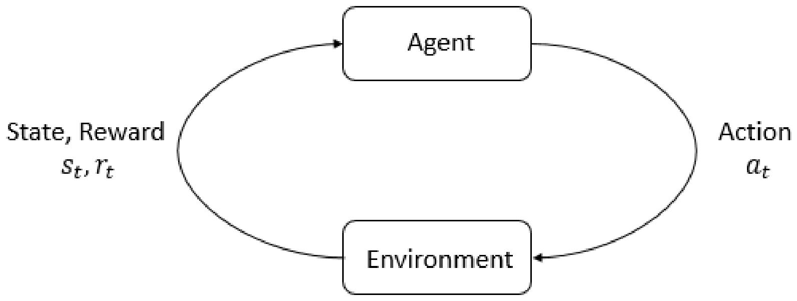
1. Introduction to reinforcement learning

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 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.

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**.

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

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

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 trajectory as

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

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 algorithms

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 $\theta$ .
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;
```

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: 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
```

2.2 Policy gradient methods

2.2.1 Idea

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 focus 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).

TODO: Following algorithm seems to be inspired by `https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html` and references original literature where no proof for integral is included. Also it already includes baseline b(s) which was not stated, nor proved as a part of original theorem

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:

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

It is interesting to look at the expression and what does it represent. There are two important part of the expression $\pi_\theta(a_t|s_t)$ and $R(\tau)$. If we take gradient step of this objective we say we want to make change of the log probability $\pi_\theta(a_t|s_t)$ weighted by how good the expected return was. However this may feel somehow counter intuitive since excepted return considers all rewards of the episode. We might want to limit ourselfs only to the future consequences of given action. Fortunatly it can be shown that $R(\tau)$ can be replaced by many other useful functions(Schulman et al. [2015b]):

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

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 practise for policy gradient algorithms is utilize some form of advantage function where baseline function is using value aproximator $b(s_t) = V^\pi(s_t)$. Value approximator is usually represented by another neural network and is being learned concurrently with the policy. There exists a naming convenction for the two types of network. The policy network is usually called an actor because it's job is to provide a policy that the agents act upon. The value network, on the other hand, is often recalled as a critic, since it in a sense produces a critique of the value of the state. Incorporating value approximator and idea of advantage function reduces variance in sample estimation and results in more stable and faster learning.

With all of this being said we present first algorithm from 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 aspects.

Firstly the absolute value of performance objective function we optimize cannot be interpreted in terms of performace result. Taking one step of gradient descent does not even guarantee improving expected return in general. On a given batch of samples value of $-\infty$ can be achieved. However expected return of changed policy would most likely be abysmal.

And secondly, policy gradient algorithms are **on-policy**, meaning that only data samples collected using most recent policy are used for training. Opposately to Q-learning off-policy approach where training data samples collected throughout entire training process are used for learning steps. For this reason policy gradient algorithms, especially Vanilla Policy Gradient algorithm is often considered as sample inefficient in comparision to off-policy algorithms.

2.2.4 Trust Region Policy Optimization

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

Next algorithm from the family of policy gradient theorem is trying to face this problem. As the name suggests, Trust Region Policy Optimization (Schulman et al. [2015a]) attempts to take steps within the region where it is constrained so as not to degrade performance.

TRPO proposes theoretical update of parametrized policy π_θ as

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

where

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

is surrogate advantage measuring policy π_θ peformance relative to the old policy π_{θ_k}.

And

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

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

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

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

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

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

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

However, since we used Taylor expansion, approximation error might break KL divergence constraint. For this reason TRPO incorporate 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 smallest non negative integer such that KL divergence constraint is satisfied and 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 that is considered in many aspects to be the state of art algorithm. As the authors of this next algorithm state. (TODO: should I emphasize more quoting the original text?) With the leading contenders Q-learning, "vanilla" policy gradient methods and trust region policy gradient methods there is still room for developing 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 simple problems. Vanilla policy gradient methods have poor data effiency and robustness. And trust region policy optimization is relatively complicated, and is not compatible with architectures that include noise or parameter sharing.

Proximal policy optimization (Schulman et al. [2017]) aims at data efficiency and realiability of TRPO performance, while using only first-order optimization. Authors propose a novel objective with clipped probability ratios, which forms a pessimistic lower bound of 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 was this objective originaly proposed. Maximization of L^{CPI} without any constraint would lead, as it was already discussed in previous section, to an excessively large policy update. Hence authors propose new modified objective called clipped surrogate objective that penalizes changes to the policy that move $r_t(\theta)$ far away 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. Objective can be intuitively explained as follows. First term in min is L^{CPI} as before. And the second term modifies surrogate objective by clipping probability ratio which prevents r_t from escaping the interval $[1-\epsilon, 1+\epsilon]$. Finally taking the minimum of the clipped and unclipped objective makes the final objective pessimistic bound on the unclipped objective.

Training process of PPO then proposes alternating between sampling data from the policy and then performing several epochs of optimization steps on the sampled data. Notice here that for every first training epoch it holds that $r_t(\theta) = 1$ making the objective in first epoch always equal to L^{CPI}

Alternatively authors propose second version of PPO incorporating KL penalty as part of objective, which makes it even more similar to idea proposed in TRPO. However, we will not cover here more details as authors themselfs uplifts more the version including clipping of surrogate objective.

If using neural network architecture that shares parameters between policy and value function, a combined objective function that contains both the policy surrogate and 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 coefficient nad 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 thing, that we explicitly haven't covered yet is the problem of exploitation and exploration. In general during training we always seek balance between exploitation of learned experience and exploring new possibilities. In the case of Q-learning, this is done quite artificially most often using 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 exploring by taking random action.

By sampling according to the policy π_θ in policy optimization methods this problem is solved more naturally. Usually, in the begining of the training process when the parameters are initialized, the parametrized probability distribution is close to uniform, which implicitly makes the action selection mechanism to be exploratory. And as the policy is updated, it becomes more specialized toward the optimal policy, effectively forcing the choice of actions 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 initial uniform distribution. On that account explorative mechanism in the form of bonus entropy(Williams [1992]) is often being utilized and also suggested by the PPO authors. By adding a small bonus for policy entropy, the policy is slightly forced toward uniform distribution, shifting toward exploratory behavior. Hence final PPO objective may have a following form:

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

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

Q-Learning and Policy optimization

So far we have covered possible the simplest division rule among types of reinforcement learning algorithms. However, there are actually several algorithms that combine both the Q-learning and Policy optimization. In these algorithms both Q approximator and policy approximator are being learned concurrently. Just to name a few 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 these in detail, as they are beyond the scope of our needs.

3. Multi agent environments for RL ???

In this chapter we revisit concepts from previous section and extend them to multi-agent settings. We introduce theoretical definitions to various settings type and provide possible schemes for concurrent learning of multiple agetns. 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 of 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 particular state of the system is equivalent for all agents.

With this setting we can extend the definition from MDP section 1.1: **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 entire collection of agents, or alternatively, all agents receive the same reward.

3.2 Decentralized Partially Observable MDP

Unfortunately, we cannot stop with this definition, because the MMDP model provides a description that is far too ideal for real-world problems. The MMDP model expects the state of the environment to be globally available to all agents. However, in multi-agent environments this is rarely the case. Usually, the world can be of high complexity and combined with limited sensory capabilities, the agent is able to perceive only limited observations that describe only a part of the entire environment state. Therefore, we extend our model definition to include the concept of partial observability.

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. Notice here, that even this definition extension did not provide us with model that is capable of describing environment where agents have different reward function, which is needed for situation where agents have different goals or are even competetive. This extension is possible with definition of Stochastic game. However, we do not need to

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

Notation

To denote joint entities we will be using bold case: $\boldsymbol{a} = (a^1, ..a^n) \in \mathcal{A}$. Joint policy $\boldsymbol{\pi}$ induced by the set of individual policies $\{\pi^i\}_{i \in n}$ gives the mapping from states into joint actions. Now the similar notation as in first chapter may be used 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 funtion, it is useful to think of the collection of agents as single agent whose goal is to produce optimal joint policy. The problem with treating MMDP as a standard MDP where actions are distributed lies in coordination. In general there exist multiple different optimal joint policies. However, even when all agents choose their individual policy according to some optimal policy, there is no guarantee that they all select from the same optimal joint policy. Such 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 how to ensure optimal coordination. Firstly, there can exist some central control mechanism 3.3, that can compute joint policy and then instruct it to all individual agents. Or secondly, each agent may comunicate its choice of individual policy to others. However, both approaches are in general case unfeasible.

3.2.1 Nash Equilibria

Alternatively we can look at the MMDP from the point of n-person game. Then the problem of determining an optimal joint policy can be viewed as a problem of optimum equilibirum selection. Nash equilibrium (Nash [1950]) for a $\boldsymbol{\pi}^*$ can be defined as:
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 correspond to optimal joint policy as some may have smaller utility value then others. This causes multi-agent environments to being more sensitive to converging to local suboptimal policies as only way to escape from such equilibrium is through coordinated change of all individual policies.

3.3 Learning schemes

Centralized scheme

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

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

Secondly, such intensive communication between all agents and the central control mechanism may be too demanding or may not be even plausible due to technical reasons.

And lastly from the theoretical point of view, representation of such joint policy grows exponentially with 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

On the other side of the spectrum lies the concurrent scheme. Here all kinds of global informations are ommitted and agents solely rely on their local observations. Such training of a agent is then completely independent.

Centralized training with decentralized execution

And lastly somewhere in between the concurrent and centralized scheme we can identify so called centralized training with decentralized execution. Here we look at the two life stages of agents. We firstly teach our agents in safe laboratory conditions, and only after training is finished they are deployed in real world conditions.

During the training, we can take advantage of the fact that we are likely to have more information about the state of the environment at our disposal. Whether that is 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 exact as possible.

Once the agents are deployed they are once again dependent solely on their local observation.

This learning scheme is quite often utilized in RL algorithms, where both actor and critic is utilized. It is demonstrated for example in aformentiond PPO 2.5 algorithms. During training both actor and critic are being trained concurrently and the value estimate obtained by the critic can be utilized to give more accurate information about true value of the given state to make the actor policy update more stable and exact. Once the training is complete, the agents that are given local observations are asked to take action based solely on the actor.

3.4 Non-stationarity

Besides all already mentioned problems connected with partial observability, local equilibiria and policy distribution, there is one more important problem that we did not have to deal with in single-agent settings. Even though we are capable of reproducing the value function expressions:

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

due to the non-stationarity problem, we cannot obtain optimal value using Bellman equations as it was done in 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, state transition and reward function of the environment. Using the idea of Bellman equations for optimality is still possible. However, in multi-agent setting we lose the theoretical foundation promising convergence for optimal policy, and training using the Q-learning update rule must be acompanied by some mechanisms to overcome 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

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

MAPPO

Lastly we want to mention the recent success in the form of Multi Agent Proximal Policy Optimization (MAPPO, Yu et al. [2021]) variant of PPO 2.5 algorithm. Authors revisit the usage of PPO in multi-agent settings. In recent years, MADDPG was usually the first choice of an algorithm when dealing with multi-agent environments, leaving PPO ommited. Authors hypothesize, this was due to the two reasons, firstly, belief that PPO is less sample-efficient than off-policy methods and secondly, the fact that common implementation and hyperparameters practices when using PPO in single-agent settings often do not yield strong performance when transfered to multi-agent settings.

Authors propose five important changes with regard to the multi-agent settings. TODO:

- Value normalization:
- Input Representation to Value Function:
- Training Data Usage:
- PPO Clipping:
- PPO Batch Size:

Utilizing these five concepts, they were able, without any other domain or structural modification, to achieve comparable or superiror results compared to off-policy algorithms on several benchmark frameworks.

4. Overcooked environment

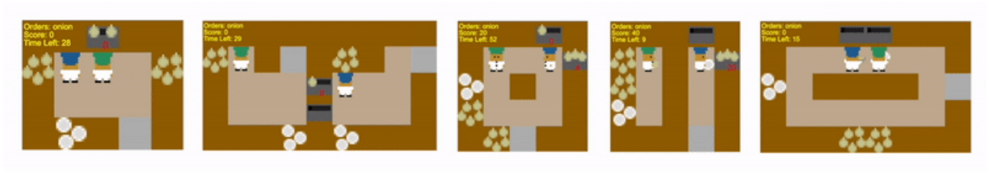
4.1 Overcooked game

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

”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

7.1 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

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.

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.

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

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`.

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`.