Multi-agent RL

All assignments (questions marked with an A that yield points) must be submitted on Brightspace before the corresponding tutorial begins (see due date). Please submit all answers in one PDF file. Scans of handwritten solutions (e.g. for math answers) are permitted, but must be readable and have a file-size below 5MB. Do not submit the voluntary exercises (marked with an E), which will not earn you any points, but can help you practice the math and prepare for the exam.

You will also be asked to implement and test something in python/pytorch. We recommend that you use a Jupyter Notebook, convert your final version (including result plots) to PDF ("Download as" → "PDF via LaTeX") and attach the PDF at the end of your submission PDF. You are welcome to use other editors, but please make sure you submit the code (or the crucial code segments) and the results in an easily readable format together with your theory answers as one PDF file.

A sample solution will be published after the tutorial. Please always demonstrate how you arrived at your solution. You will receive the points when you convince us that you have seriously attempted to answer the question, even if your answer is wrong. You qualify for the exam when you have earned 75% of the total achievable points (sum over all 4 exercise sheets, not for individual sheets).

Good luck!

A4.1: Implement exploration for MountainCar **(7 points)**

In this question you will implement *deep exploration* to solve environments like `MountainCar-v0`, which are impossible to learn with random exploration. The question can be completed in the Jupyter notebook `explore.ipynb`, that can be found on Brightspace.

(a) *[1 point]* The `Mountaincar-v0` environment is challenging because of the exploration behavior, random episodes will rarely see any reward before the maximum length is reached, but also because of its extremely unevenly scaled state space: while the car's position is between -1.2 and $+0.6$, the car's velocity is bounded between -0.07 and $+0.07$. Run a DQN experiment, which changes this state space by rescaling each dimension to be between -1 and 1 using the provided `RescaledEnv` wrapper class, for $200k$ environmental steps. To make sure you propagate rewards fast enough to see a result in that time frame (if there is one), run 10 gradient updates for every sampled episode.

(b) *[2 points]* To solve the `Mountaincar-v0` environment, you will need to use some deep exploration technique. To test this we will "cheat" a bit at first: use the provided `CountUncertainty` class to produce intrinsic reward during training. The class must `observe()` states from newly sampled episodes (make sure each transition in your replay buffer has only been observed once) and provides a scaled uncertainty estimate with a `()` method call, for example, `u(state)` for `u = CountUncertainty(...)`. This particular uncertainty class divides the state space into $\texttt{uncertainty\_resolution}^m$ bins, where m is the number of state-dimensions, counts how often observed states fall into these bins and estimates the resulting uncertainty as the standard deviation of an empirical estimator: `uncertainty_scale / sqrt(n)` for `n` observations of the same state. Complete the implementation of `ExplorationDQNExperiment` and run the `Mountaincar-v0` environment with intrinsic rewards for $200k$ steps. Make sure that the experiment only uses intrinsic rewards if the `intrinsic_reward` parameter is **`True`**.

Hint: there are multiple ways to implement intrinsic reward, but the most intuitive is to use the *uncertainty* of having seen the next state of a transition as additional reward. Make sure that you have observed those states before you compute their uncertainty, though, as states with observation-counts of 0 produce extremely large uncertainties that can destabilize learning.

(c) *[2 points]* Your above implementation of `ExplorationDQNExperiment` should be able to solve the `Mountaincar-v0` now. However, the `CountUncertainty` class does not scale to other environments. For example, the `Acrobot-v1` environment has 6-dimensional states and would induce over 15 billion bins if we would count with the same resolution as above. To use intrinsic rewards in high dimensional state spaces, you will complete the implementation of Random Network Distillation (slide 19 of Lecture 7) in the `RNDUncertainty` class. The RND uncertainty estimate shall use 3 linear layers with hidden dimension 1024, ReLU's between them and an output dimension of 256. Test your implementation on the `Mountaincar-v0` environment with intrinsic rewards for $200k$ steps.

Hint: useful `unertainty_scale` parameters depend a lot on your exact implementation of RND. It is recommended to print the average intrinsic reward in `Mountaincar-v0` achieved using `CountUncertainty` (e.g. in the previous question) and then change `unertainty_scale` for this question until the uncertainty of `RNDUncertainty` yields similar intrinsic rewards at the beginning of training. A good value for average intrinsic rewards of an initial episode is ≈ 0.1.

(d) *[2 points]* The `Acrobot-v1` environment requires the agent to learn how to swing up a chain of two connected links. The joint between the two links is (under-) actuated, and swinging the acrobot can exhibit chaotic behavior. Normally this environment allows episodes of up to 500 steps to ensure the agent sees at least some rewards using random exploration. We make this here a bit harder by restricting episodes to 200 steps (like in `Mountaincar-v0`). First run double DQN without intrinsic rewards for $200k$ steps to evaluate whether random exploration is enough to learn in this environment. Next run `Acrobot-v1` with `RNDUncertainty` intrinsic reward.

Hint: If you do not see similar learning as in `Mountcar-v0`, try to adjust `uncertainty_scale` with the same techniques as in A4.1c.

A4.2: Nash equilibria **(2 points)**

Let in the following a Nash equilibrium (NE) of a state-less general-sum game of n agents be defined as a joint action $\boldsymbol{a}_* \in \mathcal{A} := \mathcal{A}^1 \times \ldots \times \mathcal{A}^n$ for which holds $r^i(\boldsymbol{a}_*^{-i}, a_*^i) \geq r^i(\boldsymbol{a}_*^{-i}, a^i), \forall a^i \in \mathcal{A}^i, \forall i$.

(a) *[1 point]* Which inequalities of r^1 hold in a state-less two-player zero-sum game?

(b) *[1 point]* Prove analytically that all NE in such a game yield exactly the same reward.

A4.3: Solving cyclic games **(3 points)**

Let the following matrix define the rewards $r^1(a^1, a^2)$ of a state-less two-player zero-sum game:

P1\P2	a_1^2	a_2^2
a_1^1	+1	-1
a_2^1	-1	+1

(a) *[1 point]* Simulate a cycle of max-min decisions to show the game does *not* have a Nash equilibrium.

(b) *[1 point]* Compute the analytical expected reward $\mathbb{E}[r^1]$ for a pair of stochastic policies

$$\pi^i(a^i) := \begin{cases} \theta_i & \text{, if } a^i = a^i_1 \\ (1-\theta_i) & \text{, if } a^i = a^i_2 \end{cases} .$$

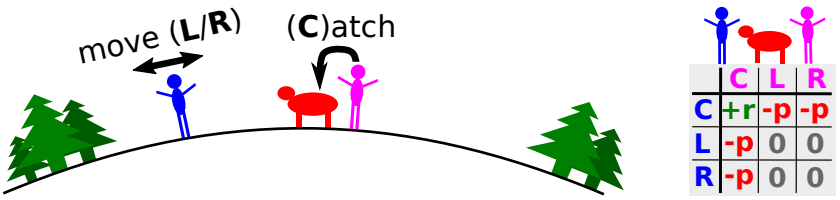
(c) *[1 point]* Compute the parameters $\boldsymbol{\theta} = [\theta_1, \theta_2]$ of a mixed Nash equilibrium for π^1 and π^2.

Hint: a mixed Nash equilibrium is a saddle point of $\mathbb{E}[r^1]$ w.r.t. $\boldsymbol{\theta}$.

Bonus-question: can you do the same analysis for the rock-paper-scissors game?

A4.4: Relative overgeneralization **(5 points)**

In this question we will analyze relative overgeneralization at the example of a predator prey game, depicted in the figure below. Here two agents (Blue and Magenta) live in a one-dimensional fully observable world, in which they can move left (L) or right (R), but get blocked by trees and the stationary prey (red). If *both* agent stand next to the prey, they have each access to a special "catch" action (C). In this special state s the collaborative reward is given by the matrix below. If both agents select the C action together in this state, the episode ends (but it continues for any other action combination). In all other states the catch action is not available, the collaborative reward is 0 and the episode continues. Assume that both agents learn with independent Q-learning (IQL) and have discount factor $\gamma \in [0, 1)$.

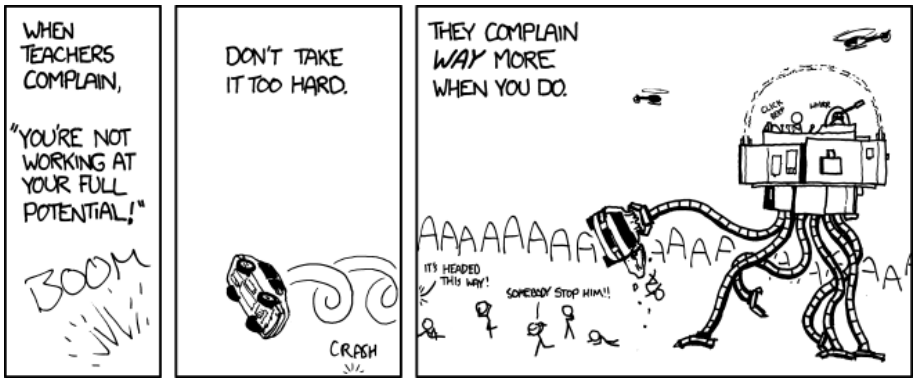


(a) *[1 point]* Determine analytically the value of the optimal policy when the first agent is 3 steps, and the second agent 5 steps away from the prey.

(b) *[2 points]* Assume that both agents will get stuck (never return to s) once they leave s. Determine analytically the independent Q-value of agent 1 (Blue) in state s (i.e. $q^1(s, a^1; \boldsymbol{\pi})$ from slide 10 of Lecture 9), assuming that both agents uniformly explore (i.e. $\pi^i(a^i|s) = \frac{1}{3}, \forall a \in \mathcal{A}^i, \forall i \in \{1, 2\}$).

Hint: don't give up if the terms become a bit unwieldy. The solution contains (among others) surprisingly large multiples of 3 in ugly fractions. Just simplify as far as possible and keep going!

(c) *[2 points]* For a given reward $r > 0$ in your solution of (b), which punishments $p \in \mathbb{R}$ induce in expectation *relative overgeneralization*, i.e., for which range of p holds $q^1(s, C; \boldsymbol{\pi}) < q^1(s, L/R; \boldsymbol{\pi})$?



https://xkcd.com/987

Total 17 points.