# Introduction to Artificial Intelligence - 236501

Homework #3: Markov Decision Processes & Learning


| Name | ID | Email |
|---|---|---|
| Matan Kutz | 205488679 | matankutz@campus.technion.ac.il |
| Yotam Granov | 211870241 | yotam.g@campus.technion.ac.il |




Faculty of Computer Science
Technion - Israel Institute of Technology

Winter 2022/23

# Question 1: MDP & RL

## Part 1 - Dry Questions

In the tutorial we've seen Bellman's equations when the reward function was a function of the current state, meaning $R : S \to \mathbb{R}$. This reward function is called "State-Reward" because it is depended on the given state and the given state only. Now, we shall expand this idea, for a reward as a function of the current state, the chosen action and the next state that was (non-deterministically) chosen, meaning $R : S \times A \times S' \to \mathbb{R}$. This reward function is called "Edge-Reward".

### Part 1.1

Change the formula for the expected utility from the tutorial for the "Edge-Reward" case, no explanation needed.

**Solution:**

$$U^{\pi}(s) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t, S_{t+1}) | S_0 = s, S_1 = s', A_0 = a]$$

### Part 1.2

Rewrite Bellman's equation for the "Edge-Reward" case, no explanation needed.

**Solution:**

$$U(s) = \max_{a \in \mathbb{A}(s)} \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma U(s')$$

### Part 1.3

Rewrite the "Value Iteration" pseudo-code for the "Edge-Reward" case.

**Solution:**

**Function** VALUE-ITERATION(*mdp*, $\epsilon$) **returns** a utility function

**Inputs**: *mdp*, an MDP with states S, actions A(s), transition model P(s'|s,a), rewards R(s,a,s'),
discount $\gamma$

$\epsilon$, the maximum error allowed in the utility of any state

**local variables**: U,U', vectors of utilities for states in S, initially zero

$\delta$, the maximum change in the utility of any state in an iteration

**Repeat**

$U \leftarrow U'; \delta \leftarrow 0$

**For each** *s* **in** *S* **do**

$U'[s] \leftarrow \max_{a \in \mathbb{A}(s)} \sum_{s'} P(s'|s,a)R(s,a,s') + \gamma U(s')$

**If** $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$

**Until** $\delta < \frac{\epsilon(1-\gamma)}{\gamma}$

**Return** U

For the $\gamma = 1$ case, we see that the function can only terminates when $\delta < 0$, and yet $\delta \geq 0$ will be true throughout any run of the algorithm - the function can never actually terminate. This result seems reasonable, since there should be no degradation in the cost of future states when $\gamma = 1$, and thus there will be no convergence when conducting the value iteration method. If we would want to find the optimal utility function when $\gamma = 1$, we would have to change the condition on $\delta$ to be that $\delta < \varepsilon \cdot \theta$, where $0 < \theta \ll 1$. The amount of accuracy we want here will determine our choice for the value of $\theta$.

## Part 1.4

Rewrite the "Policy Iteration" pseudo-code for the "Edge-Reward" case.

**Solution:**

**Function** POLICY-ITERATION(*mdp*) **returns** a policy

**Inputs**: *mdp*, an MDP with states S, actions A(s), transition model P(s'|s,a)

**Local variables**: U, a vector of utilities for states in S, initially zero

$\pi$, a policy vector indexed by state, initially random

**Repeat**

$U \leftarrow POLICY - EVALUATION(\pi, U, mdp)$

$unchanged? \leftarrow true$

**for each** s **in** *S* **do**

**if**

$$\max_{a \in A(s)} \sum_{s'} P(s'|s,a)(R(s,a,s')+U[s']) > \sum_{s'} P(s'|s,\pi[s])(R(s,\pi[s],s')+U[s'])$$

$$\pi[s'] \leftarrow \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s'|s,a)(R(s,a,s')+U[s'])$$

$unchanged? \leftarrow false$

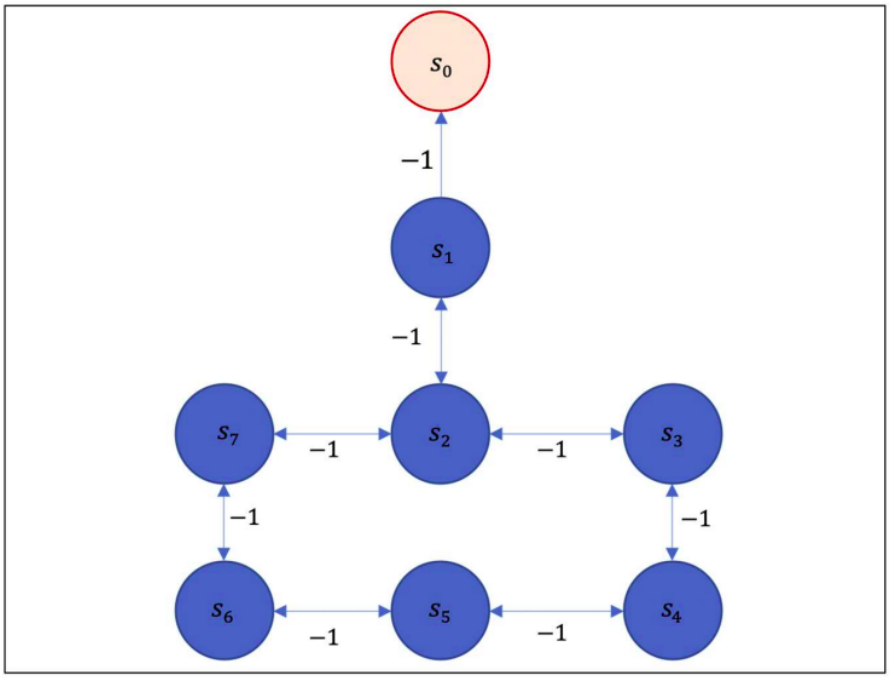
**Until** unchanged?

**Return** $\pi$

For the $\gamma = 1$ case, we can't obtain an expected utility, and thus the Policy-Evaluation function will not work. This results from the fact that the sum of the rewards will not be able to converge efficiently when the algorithm is run. If we want to obtain results for the Policy-Evaluation when $\gamma = 1$, we must instead use $0 \ll \theta < 1$. As before, the amount of accuracy we want here will determine our choice for the value of $\theta$.

## Part 1.5

The following graph is given:




and the following:

- (Discount factor) $\gamma = 1$.
- Infinite horizon.
- $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ – set of states – describing the position of the agent in the graph.
- $S_G = \{s_0\}$ – set of terminal states.
- The action set for each state in $S$, for example $A(s_2) = \{\uparrow, \rightarrow, \leftarrow\}$.
- "Edge-Reward" function:

$$\forall s \in S \backslash S_G, a \in A(s), s' \in S: \ R(s, a, s') = -1$$

- Deterministic transition function – each action succeeds with the probability of 1.

Run the Value Iteration algorithm you rewrote in subsection (c) on the given graph. Fill in the values in the following table when $\forall s \in S : U_0(s) = 0$. (You may not need to fill it all out)

**Solution:**

| | $U_0(s_i)$ | $U_1(s_i)$ | $U_2(s_i)$ | $U_3(s_i)$ | $U_4(s_i)$ | $U_5(s_i)$ | $U_6(s_i)$ |
|---|---|---|---|---|---|---|---|
| $s_1$ | 0 | -1 | -1 | -1 | -1 | -1 | -1 |
| $s_2$ | 0 | -1 | -2 | -2 | -2 | -2 | -2 |
| $s_3$ | 0 | -1 | -2 | -3 | -3 | -3 | -3 |
| $s_4$ | 0 | -1 | -2 | -3 | -4 | -4 | -4 |
| $s_5$ | 0 | -1 | -2 | -3 | -4 | -5 | -5 |
| $s_6$ | 0 | -1 | -2 | -3 | -4 | -4 | -4 |
| $s_7$ | 0 | -1 | -2 | -3 | -3 | -3 | -3 |

### Part 1.6

Run the Policy Iteration algorithm you rewrote in subsection (d) on the given graph. Fill in the values in the following table when the initial policy is already given in the first column. (You may not need to fill it all out)

**Solution:**

| | $\pi_0(s_i)$ | $\pi_1(s_i)$ | $\pi_2(s_i)$ | $\pi_3(s_i)$ |
|---|---|---|---|---|
| $s_1$ | ↑ | ↑ | ↑ | ↑ |
| $s_2$ | ↑ | ↑ | ↑ | ↑ |
| $s_3$ | ← | ← | ← | ← |
| $s_4$ | ↑ | ↑ | ↑ | ↑ |
| $s_5$ | → | → | → | → |
| $s_6$ | → | → | ↑ | ↑ |
| $s_7$ | ↓ | → | → | → |

## Part 2 - Wet Questions

John Doe ran the Qlearning algorithm on an MDP with a positive reward for each state and an initial Qtable with all zeros. At the end of the run he printed the table and saw that some of the values of certain states are still 0 for certain actions. Explain how such situation can occur.

**Solution:**
The actions in Qlearning are selected in one of two ways:

- Randomly
- Optimal action for the given state according to the knowledge so far; in this case, this method will not select an action whose corresponding slot value in the table is 0 unless all other values for the given state are 0, since any value updated at runtime will be positive

Therefore, it is possible, and perhaps also reasonable (depending on the number of the episode, the learning rate...), that not all actions will be selected for every situation and therefore there will be cells in the table that will remain with the initial value 0.

# Question 2: Intro. to Learning

## Part 1 - Dry Questions

### Part 1.3

Consider a decision tree $T$, a test example $x \in \mathbb{R}^d$ and a vector $\varepsilon \in \mathbb{R}^d$ s.t. $\forall i \in [1, d] : \varepsilon_i > 0$. An epsilon decision rule differs from the normal decision rule learnt in class in such way:

Consider arriving to a node in the decision tree that splits the tree according to the $i$-th feature with the threshold value of $v_i$ while trying to classify $x$. If $|x_i - v_i| \leq \varepsilon_i$ then the classification process continues on **both** children of the node instead of just one. In the end, $x$ is classified according to the

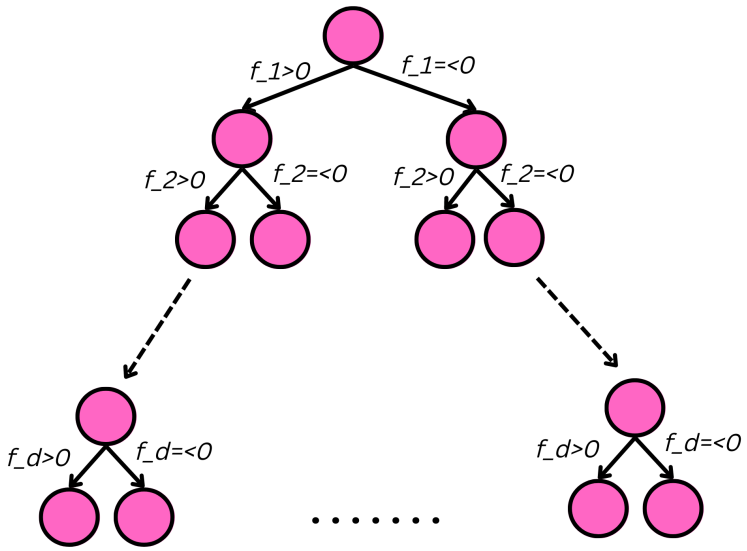
most common label of **all** of the examples in **all** of the leaves reached in the classification process. (In case of a draw the classification will be "True").

Let $T$ be a decision tree and $T'$ be the tree resulting by pruning the last level of $T$ (meaning all the examples belong to each pair of sibling leaves were passed to the parent).

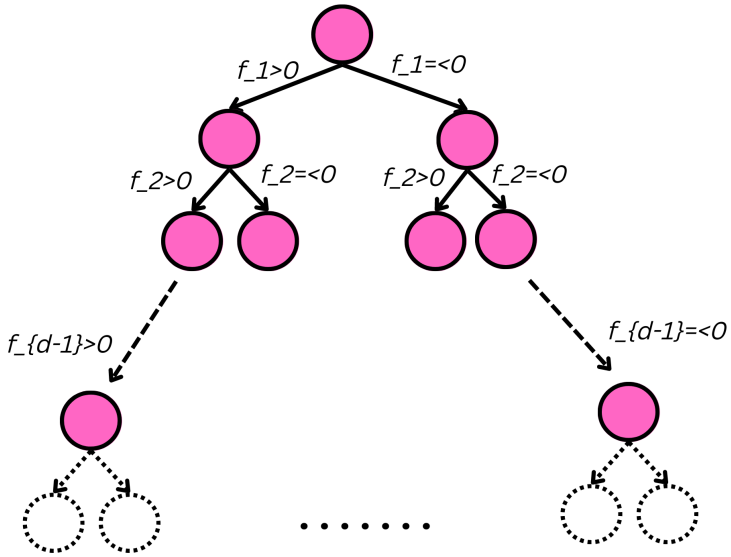
**Prove or Disprove:** There exists a vector $\varepsilon$ such that $T$ with an epsilon decision rule and $T'$ with the normal decision rule will classify every testing example in $\mathbb{R}^d$ identically.

**Solution:**

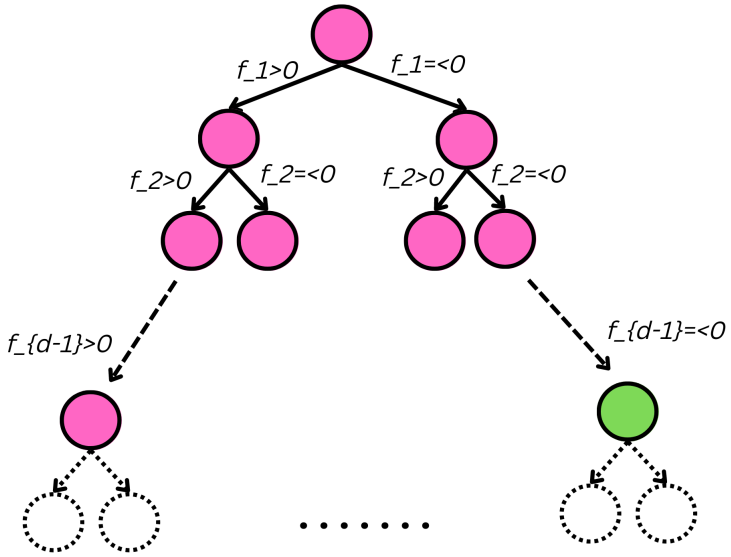
We assert that this claim is **false**. Let's assume we have a simple tree $T$ of depth $d$, where all the branches just check if the value of some feature is positive or non-positive. Such a tree might look as follows:



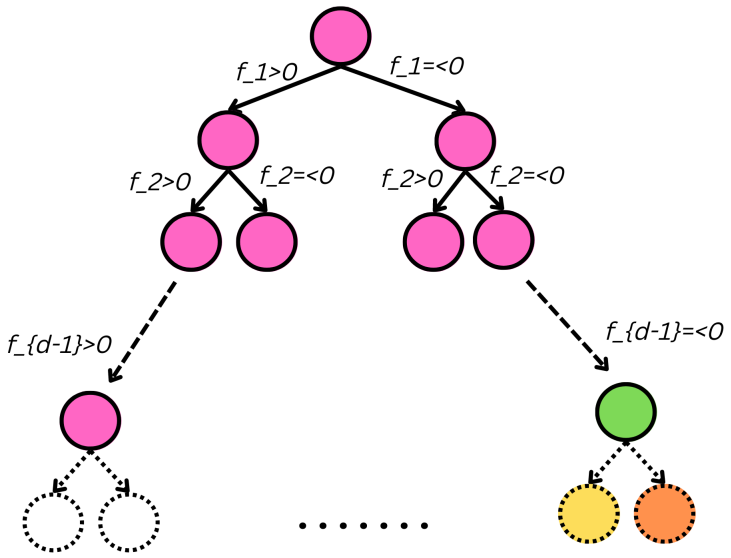

Let's construct a second tree $T'$ by pruning the last tier ($d$) of $T$, such that $T'$ has a depth of $d-1$. Such a tree might look as follows:




Let's say that now we're given a sample $x = \{x_1, x_2, ..., x_d\}$, where $x_i \leq 0, \forall i$. Let's try to classify this sample using the pruned decision tree $T'$ and the normal decision rule. Doing so, we should reach the green leaf node highlighted below:




Next, let's try to classify this sample $x$ using the original decision tree $T$ and the epsilon decision rule. In order to guarantee that we receive the same result as we just got with $T'$, we would have to require that $\varepsilon = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_{d-1}, \varepsilon_d\}$ is defined such that $|x_i| > \varepsilon_i$ for $i \in [1, d-1]$ and $|x_d| \leq \varepsilon_d$. In this case, all nodes from the root until tier $d-1$ will be treated the same way under both the normal decision rule and the epsilon decision rule. The difference will take place once tier $d$ is reached, and in this case both the yellow and orange leaf nodes highlighted in the figure below will contribute to the evaluation made by the algorithm:




This will result in exactly the same classification as we would've obtained with the normal decision rule on the pruned tree $T'$! In order to ensure that this is indeed what occurs, we would define $\varepsilon$ such that $\varepsilon_i < |x_{min}|$ for $i \in [1, d-1]$ (where $x_{min} = min\{x\}$) and $\varepsilon_d \geq |x_{max}|$ (where $x_{max} = max\{x\}$).

Unfortunately, this choice of $\varepsilon$ will only be guaranteed to work for this sample $x$ - it does not generalize to the entire space of possible samples! It will always be possible to find some other sample $z$ such that $|z_{min}| < \varepsilon_i < |x_{min}|$ (and possibly $|z_{max}| > \varepsilon_d \geq |x_{max}|$), and in such a case there will be other tiers in the tree where the epsilon decision rule will have to come into play (i.e. where $|z_i - v_i| = |z_i| \leq \varepsilon_i$ for some $i \neq d$) - therefore we are not guaranteed to obtain the same result as the normal decision rule on the pruned decision tree $T'$. Thus, it is impossible to choose $\varepsilon$ which will satisfy the requirement we specified for every possible sample $x \in \mathbb{R}^d$.

# Part 2 - Wet Questions

## Part 2.5(b)

Implement the `basic_experiment` method in `ID3_experiments.py`. Run the corresponding part in `main` and add the accuracy in your report.

**Solution:**

We get a test accuracy of 94.69% for the basic experiment.

## Part 2.6(a)

Splitting a node in the tree happens as long as the number of its examples is larger than the minimum bound $m$, causing a "pre-pruning" effect as learnt in class. Note that in that case the resulting tree might not be consistent with the training set. After the learning process (of a single tree), the classification of a new a new example is determined by the classification of the majority of examples in the corresponding leaf. Explain the importance of the pruning in general and state what phenomena it tries to prevent.

**Solution:**

Pruning attempts to prevent a phenomenon known as **overfitting**, where the model we train performs too well on the training data without being able to generalize much and perform well on unseen data (such as the test set). It is also important because it helps decrease the size of the tree (which can be preferable when we seek simpler or more easily explainable models with better statistical significance).

## Part 2.6(d)

Use the ID3 algorithm with the pre-pruning in order to build a classifier from the entire training set and make a prediction on the test set. Use the best $M$ value found in subsection (c). (Implement the `best_m_test` method in the `ID3_experiments.py` file and run the corresponding part in `main`). Add to your report the accuracy for the test set. Did the pruning improved the performance compared to the un-pruned classifier in section 5? In this subsection, if you did not implement subsection (c), use the value $M = 50$.

**Solution:**

When we use $M = 50$, we see that the best $M$ experiment results in a test accuracy of 97.35% for our model - this is 3 points higher than the performance of the model obtained without any pruning.