Mini-project 1: Deep Q-learning for Epidemic Mitigation

1 Introduction

What is the project about?

In this mini-project, you will train an artificial agent using deep-Q-learning to find a decision-making policy, regarding the mitigation of an epidemic process. You will compare performance of different methods with various **action** and **observation-spaces**.

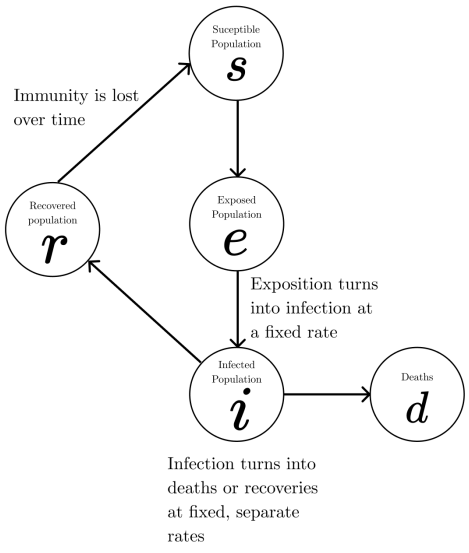
What should you submit?

1. **Report** (in `.pdf` format): Report should be at most 10 pages. It should include the results of your analyses in answers to the questions below. (See the LaTeX template in the project folder.) Please do not elaborate over 1-3 sentences per (sub-)question.

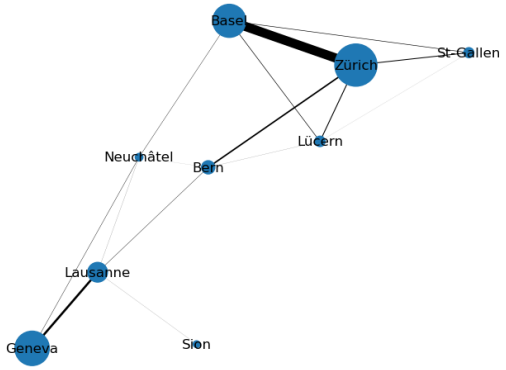
2. Jupyter notebook (in `.ipynb` format): The Jupyter notebook should contain the code for reproducing your results. The code should be well-documented and readable.

The report may be just a pdf version of the notebook, if you have answered the questions in the notebook. In this case, make sure that it is clear what question is answered where; and also submit the notebook itself!
When should you submit?
Either **A** Monday last week of teaching term, the 29th of May, or **B** Monday after end of teaching term, the 5th of June. Choose your deadline yourself (**A** or **B**), but you must be available three days after your submission (**A**: May 31st or June 1st, **B**: June 6th or 7th) in person for discussion of the miniproject (fraud detection interview).

Working with the epidemic modeling environment



(a) Schematic representation of the flow of the state variables into each other in the epidemic model. Only **infected** and **deaths** are measurable.



(b) Approximate map of Switzerland. The size of the nodes is proportional to population; the width of the edges to the rate of cross-contamination between cities.

Figure 1: The environment dynamical model.

You will act as a computer scientist trying to provide insights for optimal decision making in the context of an epidemic scenario. An epidemic of a new virus named *MARVIN23* (apologies to any Marvin amongst the students) has just started propagating in Switzerland's neighbor Listenburg. You will use a predictive model designed by epidemiologists to train a reinforcement learning agent for *epidemic mitigation*. The model takes the form of a few python classes that we call a reinforcement learning *environment*. It implements a simulation of epidemic dynamics on a simplified map of Switzerland as well as a bunch of utility functions to facilitate RL implementation.

Simulating epidemics

The epidemic model is implemented through a set of stochastic differential equations with 5 simulation variables (see figure 1a per-city on the map, drawn in figure 1b). The variables represent susceptible (s), exposed (e), infected (i), recovered (r) and dead (d) individuals (see table 1 for details), and each variable evolves in time following a set of differential equations. Time in the simulation is measured in *days*, since the onset of the epidemic (which always happens on day $d_0 = 0$). We denote time by the variable $d \in \mathbb{N}$. We use the superscript notation $\square^{(d)}$ to denote that a simulation variable is measured on day d, $s^{(17)}_{\text{Lausanne}}$ indicates the number of susceptible people in *Lausanne*, 17 days after the onset of the epidemic. We use the $\square_{\text{total}}$ convention to denote the total number of individuals in a given state across all cities, for instance,

$$s^{(d)}_{\text{total}} = \sum_{\text{city} \in \text{map}} s^{(d)}_{\text{city}},$$

gives the total number of susceptible people in Switzerland on day d.

State variable	Quantity represented by the variable
s_{city}	number of *susceptible* individuals in city "*city*".
e_{city}	number of *exposed* (infected but not yet contagious) individuals in city "*city*".
i_{city}	number of *infected* (and contagious) individuals in city "*city*".
r_{city}	number of *recovered* (cannot be infected) individuals in city "*city*".
d_{city}	number of *dead* individuals in city "*city*".

Table 1: State variables of the dynamical model.

Decision Process

In this project **we will implement agents which are only allowed to measure two of these variables**: i_{city} and d_{city} (for each city). The agent is also allowed to know the full population of each city at time $t = 0$ (before the epidemic starts), which is useful to scale the parameters. This makes the problem we will later try to solve with reinforcement learning a control problem on **Partially Observable Markov Decision Processes** (*POMDP*). The POMDP, uses a different time-step than the simulation, here, time is not measured in *days* but rather in *weeks*. In other words, the agent only makes one set of decisions per-week. At each time step of the *POMDP* the agent makes **observations** computes **actions** and a receives a **reward**. To avoid confusion between the simulation time-step measurements (in days) and the POMDP time-step measurements (in weeks) we use the following square brackets ($\square^{[w]}$) convention to denote a time-measurement in weeks and the parentheses convention ($\square^{(d)}$) to denote a measurement in days. Furthermore, we always assume that when we measure a simulation variable in weeks, that measurement is done *on the last day of that week*. For instance notation $r^{[2]}_{\text{Bern}} = r^{(14)}_{\text{Bern}}$.

Observations

At each time step the agent receives an observation consisting in a measurement of the **number of infected** ($i^{(7\cdot w+d)}_{\text{city}}$) and **dead people** ($d^{(7\cdot w+d)}_{\text{city}}$), in each city, on each day $i \in [0, ..., 6]$ of the week. This gives an observation tensor $\mathbf{o}^{[w]} \in \mathbb{R}^{(2\times 7\times \#\text{ cities})}$ (see figure 2). We call the set of all possible observations the **observation space** (denoted O).

The agent also has access to the total number of people in each city, before the epidemic started (this is meant to be used for scaling). Refer to the **tutorial jupyter notebook** (`tutorial.ipynb`) to learn how to actually gather observations using the environment library.

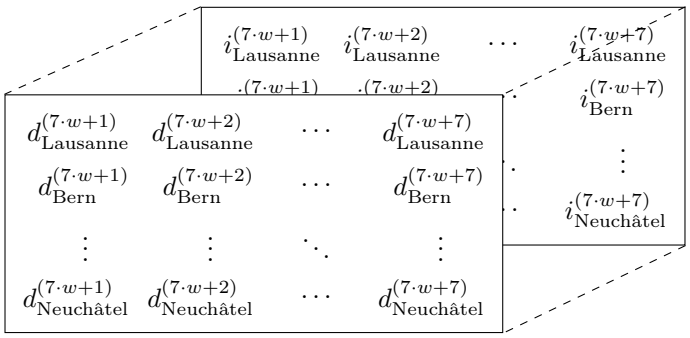


Figure 2: An example observation tensor measured on week w.

Actions

The agent action space consists of 4 binary decisions: **confinement**, **isolation**, **add additional hospital beds** and **vaccinate**. It is allowed for an agent to pick any combination of decisions, for instance: *confine*, and *vaccinate* but *do not isolate*, and *do not add additional hospital beds*. We denote the action taken by the agent on week w as $\mathbf{a}^{[w]} \in \{\texttt{True}, \texttt{False}\}^4$, we represent each decision as a boolean variable and thus the action is represented as a set of four labeled boolean variables. We use the convention that the action is *being performed* when set to `True`. Actions can be written as composition of binary decisions as follows:

$$\mathbf{a}^{[w]} = a^{[w]}_{conf} \cup a^{[w]}_{\text{isol}} \cup a^{[w]}_{\text{hosp}} \cup a^{[w]}_{\text{vacc}} = \bigcup_{\mathfrak{d} \in \text{actions}} a^{[w]}_{\mathfrak{d}}, \tag{1}$$

where $a^{[w]}_{\mathfrak{d}} \in \{\texttt{True}, \texttt{False}\}$ and $\mathfrak{d} \in [\text{conf}, \text{isol}, \text{hosp}, \text{vacc}]$. The effects of each decision is detailed in table 2. Refer to the **tutorial jupyter notebook** to learn how to actually handle actions using the environment library. We call the set of all possible actions the **action space** (denoted A). All of the actions have an associated cost (see the **reward** paragraph).

notation	decision name	Effect
a_{conf}	**confinement**	Strongly reduces the rate at which people are exposed to the virus.
a_{isol}	**isolation**	Slows down propagation from one city to the next. Does not slows down exposition within a single city.
a_{hosp}	**add hospital beds**	Reduces mortality. Has no impact on the exposition/infection rate.
a_{vacc}	**vaccinate**	Increases immunity directly. Makes people recovered without them being infected.

Table 2: Actions and their exact effects.

The environment

The decision process takes place during 30 week **episodes** (an episode therefore lasts 310 days). For each episode, the model is initialized with every person in the *susceptible* state except for a very small subset of randomly contaminated individuals. The process advances one week at a time. At the end of each week the agent gets an observation $\mathbf{o}^{[w]}$ and takes an action $\mathbf{a}^{[w]}$. As is most commonly the case, we call **policy** the function $\pi : O \to A$ that maps observations to actions. See figure 3.

The reward

In order to better quantify the performance of a policy we decide to compute a metric of the performance of a given policy. Because we are going to train reinforcement learning agents to maximize this metric we call it a reward. We define the reward as:

$$R^{[w]} = R_{\text{c}} - \mathcal{C}(\mathbf{a}^{[w]}) - D \cdot \Delta d^{[w]}_{\text{total}} \tag{2}$$

which is split into three terms: a *constant reward* term R_{c} to which we subtract two costs, a death cost $D \cdot \Delta d^{[w]}_{\text{total}}$ which is directly proportional to the number of people who died since the last POMDP step, and an action cost

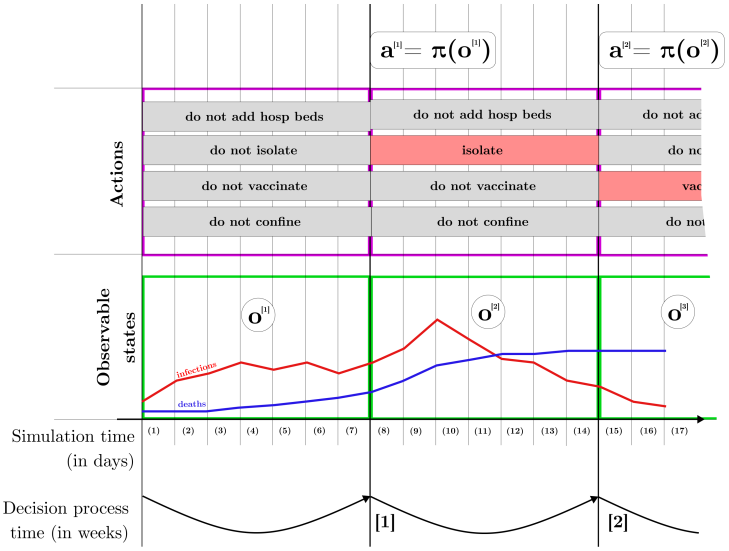


Figure 3: Illustration of part of a decision process episode. Note the difference between simulation time (in days) and decision process step (in weeks), as well as the fact that actions are always taken based on an observation from the previous week. For instance, on week 1, the agent makes observation $\mathbf{o}^{[1]}$, and uses it to compute action $\mathbf{a}^{[1]}$, which is applied until the end of week 2.

Term	Value Represented
R_{c}	constant positive reward term.
D	cost of deaths.
V	cost of vaccination.
H	cost of extra hospitalization.
C	cost of confinement.
I	cost of isolation.
$\mathbf{1}_{vac}, \mathbf{1}_{hosp}, \mathbf{1}_{conf}, \mathbf{1}_{isol}$	indicator functions for vaccination, hospitalization, confinement and isolation. I.e. $\mathbf{1}_{action} = 1$ if the action is take, 0 otherwise.
$\mathbf{1}^{+}_{vac}, \mathbf{1}^{+}_{isol}, \mathbf{1}^{+}_{conf}$	indicator functions for transition to vaccination, isolation or confinement. I.e. $\mathbf{1}^{+}_{action} = 1$ if the action was just taken, 0 otherwise.
$\Delta d^{(d)}_{\text{city}} = d^{(d)}_{\text{city}} - d^{(t-1)}_{\text{city}}$	new deaths from one timestep to the next.

Table 3: Terms of the reward function.

$\mathcal{C}(\mathbf{a}^{[w]})$. The action cost itself consists in fixed cost associated with taking each action for one time step (the price of vaccines in the case of vaccination) to which we add a cost for announcing that an action moves from `False` to `True` at a given time step. This is meant to model the additional cost that comes with the transition to a new policy (for instance the extra cost induced when setting up new vaccination center), and to encourage convergence to a less noisy policy.

$$\begin{aligned} \textbf{Action cost} \quad & \mathcal{C}(\mathbf{a}^{[w]}) = \mathcal{A}(\mathbf{a}^{[w]}) + \mathbf{1}_{vac} \cdot V + \mathbf{1}_{hosp} \cdot H + \mathbf{1}_{conf} \cdot C + \mathbf{1}_{isol} \cdot I \\ \textbf{Annoucement costs} \quad & \mathcal{A}(\mathbf{a}^{[w]}) = A \cdot (\mathbf{1}^{+}_{\text{vac}} + \mathbf{1}^{+}_{\text{isol}} + \mathbf{1}^{+}_{\text{conf}}) \end{aligned} \tag{3}$$

all the terms of the rewards are detailled in the table 3.

Question 1.a) study the behavior of the model when epidemics are unmitigated

Run the epidemic simulation for *one episode* (30 weeks), *without epidemic mitigation* (meaning no action is taken, i.e. all values in the action dictionary are set to `False`) and **produce three plots**:

1. A plot of variables $s^{[w]}_{\text{total}}, e^{[w]}_{\text{total}}, i^{[w]}_{\text{total}}, r^{[w]}_{\text{total}}, d^{[w]}_{\text{total}}$ over time, where time is measured in weeks and all the variables share the y axis scaling.

2. A plot of variables $i^{[w]}_{\text{total}}, d^{[w]}_{\text{total}}$ over time, where time is measured in weeks and all the variables share the y axis scaling.

3. A set of plots of variables $i^{[w]}_{\text{city}}, d^{[w]}_{\text{city}}$ over time, where time is measured in weeks (one subplot per-city, variables share the y-scaling per-city).

Discuss the evolution of the variables over time.

2 Professor Russo's Policy

Since the epidemic hit Listenburg before Switzerland, the listenburgish medical community has had time to study the epidemic behavior of the disease and one of the listenburgish experts, professor Russo, suggests the following mitigation policy:

Algorithm 1: Pr. Russo's Policy (π_{Russo})

Input : $x \leftarrow i^{[w]}_{\text{total}}$ number of infected people at the end of week w
if $x > 20'000$ **then**
| *Confine the entire country for 4 weeks*
end

The number of infected cases is not evaluated during a confinement period, *i.e.* if the policy declares a 4-week confinement starting at week w and the number of infected people is still $> 20'000$ at week $w+2$, a new 4-week confinement does not start at $w+2$.

Question 2.a) Implement Pr. Russo's Policy

Implement Pr. Russo's Policy **as a python class** (*we recommend that you subclass the* `Agent` *abstract class provided with the project files, and as is demonstrated in the tutorial notebook*). Run the epidemic simulation for one episode using Pr. Russo's Policy to pick actions and **produce four plots**:

1. A plot of variables $s^{[w]}_{\text{total}}, e^{[w]}_{\text{total}}, i^{[w]}_{\text{total}}, r^{[w]}_{\text{total}}, d^{[w]}_{\text{total}}$ over time, where time is measured in weeks and all the variables share the y axis scaling.

2. A plot of variables $i^{[w]}_{\text{total}}, d^{[w]}_{\text{total}}$ over time, where time is measured in weeks and all the variables share the y axis scaling.

3. A set of plots of variables $i^{[w]}_{\text{city}}, d^{[w]}_{\text{city}}$ over time, where time is measured in weeks (one subplot per-city, variables share the y-scaling per-city).

4. A plot of the action taken by the policy over time (whether the policy chooses to confine or not).

Discuss how the epidemic simulation responds to Pr. Russo's Policy (focus on how it differs from the unmitigated scenario).

Question 2.b) Evaluate Pr. Russo's Policy

In order to be able to make meaningful conclusions about the behavior of the policy, you will need properly evaluate its behavior. To do so, running a single episode is not enough. **Implement the following evaluation procedure**: run 50 simulation episodes where actions are chosen from policy π (in this case π will be π_{Russo}, but you should make the effort to write evaluation code in which you can easily change the policy being evaluated as you will be asked to evaluate each policy you train later). **For each episode, save the following values**:

1. the **number of total confined days** $N_{\text{confinement}} = 7 \cdot$ number of confined weeks ,
2. the **cumulative reward** (the sum of all rewards collected during the episode) $R_{\text{cumulative}} = \sum_{i \in [0,\dots,30]} R^{[i]}$.
3. the **number of total deaths** $N_{\text{deaths}} = d^{[30]}_{\text{total}}$,

Make sure to always be using the same sequence of random seeds (see the **tutorial notebook** for details on the environment seed) when evaluating different policies, in order to ensure that results are repoducible. (To check that they are, run the eval twice, and check that you get the same average values). Once those values are logged for each episode, **plot a histogram**. (Use the histogram plotting function from the tutorial notebook.)

3 A Deep Q-learning approach

Now you will try to improve on Pr. Russo's Policy using Deep-Q-Learning. In the following section you will implement a first version of Deep-Q-learning with a simple action space.

The method that you will implement is the Deep Q-Network (DQN) algorithm, as first presented in Mnih et. al 2013. We recommend that you implement the algorithm using the *pytorch* library and that you make use of the *pytorch* example for DQN which is a really good reference implementation.

Note unless special measures are taken, the training of neural networks is non-deterministic in most deep-learning libraries. To ensure that your results are reproducible you will thus need to seed not only the environment but also your deep-learning library. Refer to the **jupyter tutorial notebook** for instructions.

For each deep-learning policy that you train in this project we recommend that you use the following hyperparameters. If you wish so, you are free to test other values, but as searching through the hyperparameter-space can be quite a tedious experience we are giving you values that we know will allow the algorithm to converge to a good policy.
Remark on pre-processing: note that if you "naively" scale the observations, you will end up with really small values in the observation tensor (since the number of deaths is much smaller than the total population). To avoid this, we recommend that you pass the (scaled) observations through a $(\cdot)^{\frac{1}{4}}$ function.

hyperparameter	Value
neural network architecture	A 3 hidden layer fully connected neural net with layers of size **input size**, 64, 32, 16, **output size** (see figure 4)
activations	ReLU activations after each layers (except the output which is purely linear)
target update rate	(fully) update the target network every 5 episodes
training length	train for 500 episodes
learning rate	$5 \cdot 10^{-3}$ (when the action-space is binary)
discount factor γ	0.9
buffer size	20000
batch size	2048

Table 4: Suggested training hyperparameters.

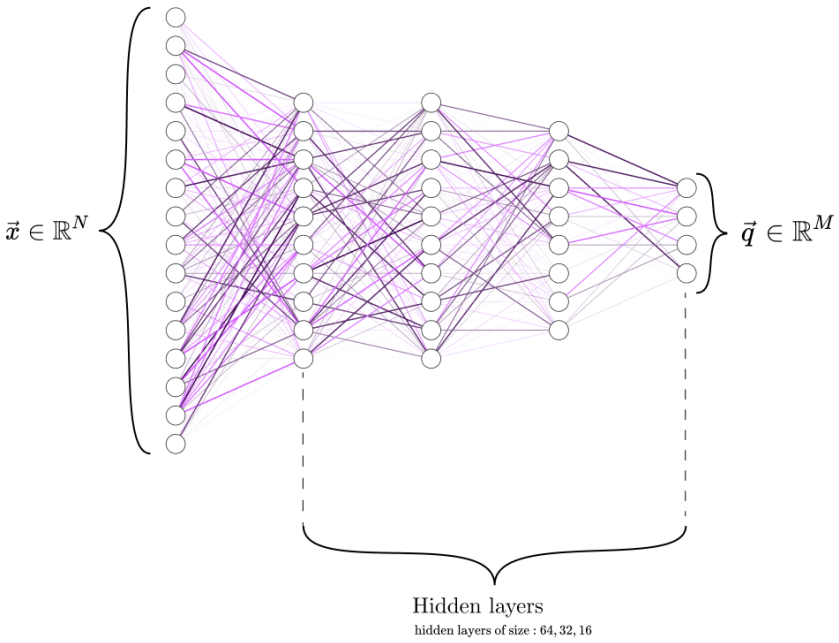


Figure 4: Network architecture.

3.1 Deep Q-Learning with a binary action space

Your friend suggests using Deep Q-learning as an improvement to Pr. Russo's policy, using a very simple action/observation-space. *The action space is binary* (meaning the neural network has an output size of 2: where one neuron estimates the Q value of confinement, and the other the Q-value of non confinement). The *observation space* contains the measurement of the proportion of dead and infected people in each city, each day of a given week (it is a matrix of shape $2 \times 9 \times 7$, exactly as in figure 2).

Question 3.a) implementing Deep Q-Learning

Implement and train the Deep Q-Learning agent π_{DQN} for 500 training episodes, with $\epsilon = 0.7$. **For each episode, log the cumulative reward.** You will see that Deep Q-Learning training curves are quite noisy, so to better evaluate the learning process we ask you to **implement the following evaluation procedure**.

Algorithm 2: Training/Logging procedure for DQN

```
for episode in 500 do
    Run an episode.
    Log the cumulative reward R_cumulative to training trace.
    Run a training step.
    if episode%50 = 0 or episode = 500 then
        Run a 20 episode evaluation procedure, log their cumulative rewards.
        Compute the average cumulative reward over the 20 evaluation episodes and log it to eval trace.
    end
end
```

The total training time is 500 episodes. Every 50 episodes, run a 20 episode evaluation process *with* $\epsilon = 0$ *(no exploration) and no learning occurring* and compute an average cumulative reward across the 20 episodes. We call the plot of the training reward, the *training trace* and the plot of the evaluation reward the *evaluation trace*. For the evaluation procedure before, be careful to always use the same seed, to ensure repoducible results. Once you have successfully done so, **plot the training trace** and the **eval trace**. (We expect a plot of the reward in y and the training episode in x both for the training trace and the eval trace). It might be more readable to plot the training trace as a scatter plot. Since training results can be hard to reproduce, **we ask you to average the eval trace across three full training processes (for the training trace, you can just scatter-polt the three trainings together).** Do not forget to **save your trained weights** (the ones which gave the best reward on evaluation) for later evaluation. **Does your agent learn a meaningful policy? Record three example episodes** where actions are picked by the best policy π^*_{DQN} you obtain. **Plot one of those episodes**, using the same plotting procedure as in question *2.a)* and **interpret the policy**.

Question 3.b) decreasing exploration

Implement and train Deep Q-Learning agent for 500 training episodes with decreasing ϵ:

$$\epsilon(t) = \max\left(\frac{\epsilon_0(T_{\max} - t)}{T_{\max}}, \epsilon_{\min}\right) \tag{4}$$

where t is the episode number, $T_{\max} = 500$, $\epsilon_0 = 0.7$ and $\epsilon_{\min} = 0.2$. **Plot the evaluation and training traces** on a shared plot with the traces from question *3.b)*, (we ask you to average across 3 training runs). **Compare and discuss the results** between questions 3.a) and 3.b). **Which policy gets the best results and why?**

Question 3.c) evaluate the best performing policy against Pr. Russo's policy

Run the best performing policy π^*_{DQN} through the evaluation code that you wrote to evaluate Pr. Russo's policy in question 2.b), **generate the same histogram plots and compare the results**. **Did the reinforcement learning policy outperform Pr. Russo's**, if so in what sense?

4 Dealing with a more complex action Space

From here on, we expect that all agents are trained with decreasing exploration (following the same $\epsilon(t)$ as described in question 3.b).

In order to achieve stable training, **the multi-action policies trained in question *4.1* and *4.2* require that learning rate to be much smaller than in the binary action situation. We recommend that you use a learning rate lr $= 10^{-5}$.**

4.1 Toggle-action-space multi-action agent

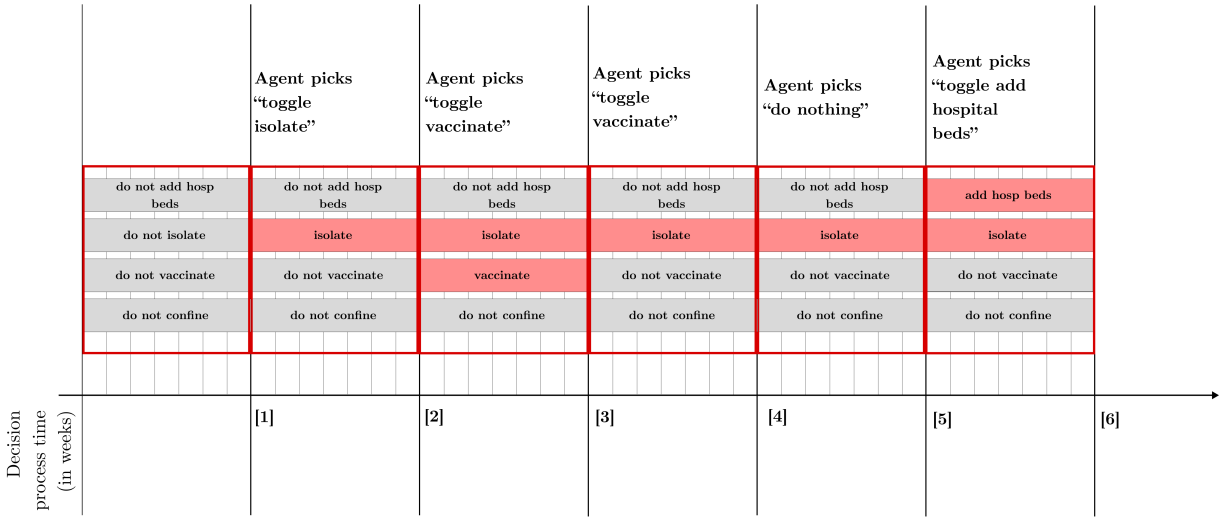


Figure 5: Example decision process with the toggled action space.

In order to to allow your agent to be use every action in the environment, your Friend proposes the following method. Let the action space have 5 actions: [`do nothing`,`toggle confinement`,`toggle isolation`, `toggle additional hospital beds`, `toggle vaccination`]. When the agent takes a `toggle` action, the state of that specific action changes state from what is previously was to the opposite. If the agent takes the `does nothing` action, no action is changed. For instance if the current state of the *isolation* action is `True`, picking the action `toggle isolation` makes the state of the action `False` in the next *POMDP step* (see figure 5 for an example). For each of the 5 actions, a Q-value is estimated. Because the agent doesn't have direct control anymore, and because the Q-value of the *toggle* actions changes depending on whether their state is set to `True` or `False`, we also need to change the observation space to include the current state of each action as an observation. This should be implemented using action and observation preprocessors, as is displayed in the tutorial notebook.

Question 4.1.a) (Theory) Action space design

Why would one want to use such an action-observation space as the one above, rather than directly compute $Q(s, a)$ for each action? **Discuss, the impact on network architecture and on training.**

Question 4.1.b) Toggle-action-space multi-action policy training

Implement the toggled-action and observation spaces and train your Deep Q-Learning agent on it. We call the best trained policy on this action-space π^*_{toggle}. **Plot the training and evaluation traces** (we ask you to average across 3 training runs). **Is the agent properly learning?** Run a 3 episodes of the best policy π^*_{Toggle}, to better understand the behavior of the learned policy. **Plot one of those episodes and interpret the policy**.

Question 4.1.c) Toggle-action-space multi-action policy evaluation

Evaluate the π^*_{Toggle} policy trained in question *4.1.b)*, using the evaluation procedure that you previously defined. **Plot the histograms** and discuss the results. **How does the policy perform compared to the binary action policy** evaluated in question 3.c?

Question 4.1.d) (Theory) question about toggled-action-space policy, what assumption does it make?

In question *4.1.b)*, you implemented a toggle-action-space policy. **What assumptions does the use of such a technique make on the action space**? Could you **think of an action space for which toggling the**

actions would not be suitable? Discuss.

4.2 Factorized Q-values, multi-action agent

You decide to try an alternative approach, *factorized Q-values*. Instead of implementing a toggled behavior as above you let the neural network decide whether to make each decision `True` or `False` independently. To do so you create a 4×2 output layer for your Deep Q Network. Each decision ($a_{\text{conf}} =$ `True` or $a_{\text{conf}} =$ `False`, $a_{\text{isol}} =$ `True` or $a_{\text{isol}} =$ `False`, etc.) is associated with a neuron pair. The Q-value of the complete action is given by the sum of the Q-values of each decision picked:

$$Q(\mathbf{a}^{[w]}, s) = Q(a_{conf}^{[w]} \cup a_{\text{isol}}^{[w]} \cup a_{\text{hosp}}^{[w]} \cup a_{\text{vacc}}^{[w]}, s) = \sum_{\mathfrak{d} \in \text{decisions}} Q(a_{\mathfrak{d}}, s). \tag{5}$$

In other words it is a sum of half the neurons of the last layer, where we choose each neuron according to which decision was made. Once Q-values are picked in that fashion we get that:

$$\begin{aligned} \text{argmax}_{\mathbf{a}^{[w]}} Q(\mathbf{a}^{[w]}, s) &= \text{argmax}_{\mathbf{a}^{[w]}} \sum_{\mathfrak{d} \in \text{decisions}} Q(a_{\mathfrak{d}}, s) \\ &= \bigcup_{\mathfrak{d} \in \text{actions}} \text{argmax}_{a_{\mathfrak{d}}^{[w]}} Q(a_{\mathfrak{d}}^{[w]}, s) \end{aligned} \tag{6}$$

I.e. we can pick pick the decision that maximizes the Q-value term associated with a given action for each element. We then train our neural network by backpropagating on the sum of whatever actions we independently picked. Since there is no toggle behavior anymore, you can use the same observation space as in question *3.a)* (no need to expand the observation space in any specific way). You can use the MultiBinary space type to create the action space.

Question 4.2.a) multi-action factorized Q-values policy training

Implement the multi-action factorized Q-values agent and observation spaces and **train your Deep Q-Learning agent on it**. We call the best-performing trained agent π^*_{factor}. **Plot the evaluation and training traces** on a graph together with the traces from the toggle-action-space training (averaged across 3 training runs). **Does it successfully learn?** Run a few episodes of the best policy π^*_{factor}, to better understand the behavior of the learned policy. **Plot one of those episodes and interpret the policy**. Is the policy realistic?

Question 4.2.b) multi-action factorized Q-values policy evaluation

Evaluate the best policy (π^*_{Factor}) trained in question *4.2.a)*, using the evaluation procedure that you previously defined. **Plot the histograms**, discuss the results, and compare them to the policy implemented in *4.1.b)*. **How does it compare to the toggled policy**?

Question 4.2.c) (Theory) Factorized-Q-values, what assumption does it make?

In question *4.2.a)*, you implemented a factorized-Q-value policy. **What assumptions does the use of such a technique make on the action space?** Could you **think of an action space for which factorizing Q-values would not be suitable**? Discuss.

5 Wrapping Up

Question 5.a) (Result analysis) Comparing the training behaviors

Compare the evaluations and training curves of **Pr. Russo's Policy**, **single-action DQN**, **factorized Q-values** and **toggled-action-space** policies. Discuss the performance differences, what do you observe? How do the two approaches compare ? What approach performs best? Why?

Question 5.b) (Result analysis) Comparing policies

Run the evaluation procedure with each (π_{DQN}, π_{toggle}, π_{factor} as well as the original π_{russo}) trained policy (always pick the best-performing policy) for 50 episode and compute the following metrics (all of the averages are empirical means computed over the 50 episodes):

1. the average number of **total confined days** avg$[N_{\text{confinement}}]$ (*lower is better*),

2. the average number of **total isolation days** avg$[N_{\text{isolation}}]$ (*lower is better*),
3. the average number of **total vaccination days** avg$[N_{\text{vaccination}}]$ (*lower is better*),
4. the average number of **total additional hospital bed days** avg$[N_{\text{confinement}}]$ (*lower is better*),
5. the average **number of total deaths** avg$[N_{\text{deaths}}]$ (*lower is better*),
6. the average **cumulative reward** avg$[R_{\text{cumulative}}]$ (*higer is better*).

Make a table with each of those values for each policy. Clearly mark the best performing policy with respect to each metric. Exclude the policies that do not have access to an action from the relevant metric (for instance, as π_{Russo} does not have access to vaccination, exclude it from the total vaccination days comparaison). Discuss.

Question 5.c) (Interpretability) Q-values

For both π_{DQN} and π_{factor}, produce a plot for visualizing the estimated Q-values for one episode. **Run a simulation episode, and plot a heat-map of the evolution of all Q-values with time**, with action selections in the y-axis and time (in weeks) in the x axis. Clearly label which action is associated with which Q-value on the y axis as well as the evolution of time in weeks, provide a color-bar. **Discuss your results.** How interpretable is your policy?

Question 5.d) (Theory), Is cumulative reward an increasing function of the number of actions?

In the following project you have implemented different policies acting on the exact same environment with a different number of actions. **Is cumulative reward an increasing function of the number of actions?** (*In other words, does adding an action always yield a better reward?*)