

Computational Intelligence 2024/25

(Solution) Mock Exam – January 30th, 2025

Please mind the following:

- Fill in your **personal information** in the fields below.
- You may use an unmarked dictionary during the exam. **No** additional tools (like calculators, e.g.) might be used.
- Fill in all your answers **directly into this exam sheet**. You may use the backside of the individual sheets or ask for additional paper. In any case, make sure to mark **clearly** which question you are answering. Do not use pens with green or red color or pencils for writing your answers. You may give your answers in both **German and English**.
- Points annotated for the individual tasks only serve as a preliminary guideline.
- At the end of the exam hand in your **whole exam sheet**. Please make sure you do not undo the binder clip!
- To forfeit your exam (“entwerten”), please cross out clearly this **cover page** as well as **all pages** of the exam sheet. This way the exam will not be evaluated and will not be counted as an exam attempt.
- Please place your LMU-Card or student ID as well as photo identification (“Lichtbildausweis”) on the table next to you. Note that we need to check your data with the data you enter on this cover sheet.

Time Limit: 90 minutes

First Name:																						
Last Name:																						
Matriculation Number:																						
<table border="1"><tr><td>Topic 1</td><td>max. 18 pts.</td><td>pts.</td></tr><tr><td>Topic 2</td><td>max. 10 pts.</td><td>pts.</td></tr><tr><td>Topic 3</td><td>max. 22 pts.</td><td>pts.</td></tr><tr><td>Topic 4</td><td>max. 10 pts.</td><td>pts.</td></tr><tr><td>Topic 5</td><td>max. 10 pts.</td><td>pts.</td></tr><tr><td>Topic 6</td><td>max. 20 pts.</td><td>pts.</td></tr><tr><td colspan="2">Total max. 90 pts.</td><td>pts.</td></tr></table>		Topic 1	max. 18 pts.	pts.	Topic 2	max. 10 pts.	pts.	Topic 3	max. 22 pts.	pts.	Topic 4	max. 10 pts.	pts.	Topic 5	max. 10 pts.	pts.	Topic 6	max. 20 pts.	pts.	Total max. 90 pts.		pts.
Topic 1	max. 18 pts.	pts.																				
Topic 2	max. 10 pts.	pts.																				
Topic 3	max. 22 pts.	pts.																				
Topic 4	max. 10 pts.	pts.																				
Topic 5	max. 10 pts.	pts.																				
Topic 6	max. 20 pts.	pts.																				
Total max. 90 pts.		pts.																				

1 General Knowledge / Blanks

10pts

For each of the following gaps in the text, fill in the answer in the form of 1-3 words. Every correctly filled gap is awarded one point. Incorrect or empty gaps will be marked with zero points.

The field of evolutionary computing is a subset of computational intelligence that draws inspiration from the principles of nature-inspired processes. Central to this paradigm is the concept of a population of potential solutions, each referred to as an individual. These solutions are evaluated using a fitness function that quantifies their effectiveness at solving the given problem. Through iterative cycles, the solutions undergo selection, where more optimal candidates are more likely to be chosen for further generations.

Optimization involves two primary variational operators:

Recombination combines parts of two chosen individuals — also called the parent candidates — to produce offspring candidate(s), introducing diversity and enabling broader search in the solution space.

The mutation operator on the other hand introduces random alterations to a candidate's genome, helping to maintain genetic diversity and prevent premature convergence. In lower dimensions, the optimization process can be visualized using the optimization landscape metaphor, where each solution represents a point in the search space. The evaluations then define the 'topology', with peaks and valleys representing higher and lower target scores. However, the presence of multiple local optima can pose challenges, highlighting the exploration vs. exploitation tradeoff inherent to any optimization algorithm.

2 Agents and Goals

18pts

Scenario 1. We develop a *CrowBot* that should be able to closely resemble a crow and mimic its behavior. We first test the *CrowBot* in a designated space of dimensions $10 \times 10 \times 10$ (in some unit of measurement, like meters, e.g.), in which the *CrowBot* occupies a single point at any given time step. We encode its current position c_t at time t as a value from $[0; 10]^3 \subset \mathbb{R}^3$. At a given time step t , the *CrowBots*'s action $a_t \in \mathcal{A} \subset \mathbb{R}^3 \times \mathbb{B}$ consists of a real-valued three-dimensional vector of geometric length of 0.1 in 3d space describing the *CrowBot*'s current movement and a Boolean flag that is *true* iff at the current time step t the *CrowBot* uses its claws. By using its claws, the *CrowBot* can attempt to pick up an object within 0.1 range of its current position, if it is not currently carrying any object, or it can drop an object, if it is currently carrying one.

Within our designated space there also exists exactly one nut, which can reside at a position $p \in [0; 10]^3 \subset \mathbb{R}^3$. However, $p = (x, y, z)$ can only have a non-zero height, i.e., $z > 0$, if the nut is carried by the *CrowBot*. As soon as the nut is not carried by the *CrowBot*, its position p is restricted to $p \in [0; 10]^2 \times \{0\} \subset \mathbb{R}^2 \times \{0\}$ on the ground (i.e., $z = 0$), since we assume that the nut falls down immediately if dropped from a higher altitude, for example.

There can also exist exactly one truck, which can reside at a position $l \in [0; 10]^2 \times \{0\} \subset \mathbb{R}^2 \times \{0\}$ on the ground, since we assume that the truck can never be picked up and neither otherwise move in the air.

There exist no other objects (or other important information) within the *CrowBot*'s environment. The system is fully observable.

You can use a 3d geometric distance function between arbitrary 3d positions d_1, d_2

$$\text{dist}(d_1, d_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

where $d_1 = (x_1, y_1, z_1) \in \mathbb{R}^3$ and $d_2 = (x_2, y_2, z_2) \in \mathbb{R}^3$.

(i) Give a suitable observation space \mathcal{O} for the *CrowBot* and a suitable state space \mathcal{S} for the whole system. Briefly explain your reasoning. (4pts)

Define $\mathcal{O} = \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{B} \times \mathbb{R}^2$, where these components represent the *CrowBot*'s position, the nut's position, a Boolean indicating whether the nut is being carried, and the truck's position on the ground, respectively. Since the environment is fully observable, $\mathcal{S} = \mathcal{O}$.

(ii) To mimic real crow behavior, the *CrowBot* should feel good if it manages to crack a nut by having the truck drive over it, i.e., it should be carrying the nut at some point in time and at a later point in time the nut should be on the ground and the truck's position should be within a distance of 1 from the nut's position. Give a goal predicate $\gamma : \langle \mathcal{S} \rangle \times \langle \mathcal{A} \rangle$ so that γ holds iff the behavior described above has been performed within the given sequences of states and actions. (6pts)

$$\begin{aligned}
\gamma(\langle s_t \rangle_t, \langle a_t \rangle_t) \iff & \exists t_1, t_2 : t_1 < t_2 \\
& \wedge q_{t_1} = 1 \\
& \wedge q_{t_2} = 0 & \text{(only one of these lines...)} \\
& \wedge p_{t_2} \in \{(x, y, 0) \mid x, y \in \mathbb{N}\} & \text{(... is strictly necessary.)} \\
& \wedge \text{dist}(p_{t_2}, l_{t_2}) < 1
\end{aligned}$$

where $s_t = (c_t, p_t, q_t, l_t)$ for all t

(iii) Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process (MDP, cf. Definition 4 in the appendix) with state space \mathcal{S} as you defined in task (i), action space \mathcal{A} from Scenario 1, target space $\mathcal{T} = \mathbb{R}$, and a given transition probability function P . Note that P fulfills the following properties:

- The *CrowBot*'s actions work deterministically on the *CrowBot*'s and the nut's respective positions as intended in Scenario 1. If the *CrowBot* is ever within a distance of 1 to the truck's position, the MDP ends.
- The nut resides at a random position on the ground in the starting state. The position of the nut only changes when carried by the *CrowBot*. If the nut is ever within a distance of 1 to the truck's position (as described in task (ii)), the MDP ends.
- In the starting state, the truck resides at a position $l = (x, 0, 0)$ for random $x \in [0; 10] \subset \mathbb{R}$. At each time step, the truck drives a distance of 0.2 directly in y -direction until it arrives at the point $l = (x, 10, 0)$. Then, the truck appears at a point $l = (x', 0, 0)$ for a new random $x' \in [0; 10] \subset \mathbb{R}$.
- Aside from the cases described above, the MDP never ends.

We successfully train the *CrowBot* to maximize the total sum of rewards generated by the reward function $R(-, -, -) = 1$. What behavior should we expect the *CrowBot* to have learned? Briefly explain your reasoning. (4pts)

Since the reward is additively constant over time and destroying the nut ends the MDP, the *CrowBot* should have learned to both move the nut out of the truck's path and stay itself further away than distance 1 to the truck.

(iv) Now compare the goal predicate γ from task (ii) and the reward function R from task (iii). Does optimizing for total reward based on R help to fulfill γ or are the two goals in conflict? Briefly explain why. (4pts)

The goal predicate γ and the reward function stand in direct conflict, since quick fulfillment of γ would involve moving the nut in the truck's path, i.e., ending the MDP with a minimal reward. On the other hand, a never-ending MDP with maximizing reward means the nut never gets driven over, hence the goal predicate will never be fulfilled.

3 Fuzzy Logic

10pts

We now want to integrate the *CrowBot* with a murder¹ of real crows. To this end, the *CrowBot* needs to match the appearance of the other crows. Since crows are very intelligent animals, they were able to communicate to us that they judge other crows based on the color of their beak k (on a scale of $[0; 255] \subset \mathbb{R}$ with $k = 0$ being the darkest color and $k = 255$ the lightest) and the volume of their caws v (on a scale of $[0; 1] \subset \mathbb{R}$ with $v = 0$ being absolutely silent and $v = 1$ being extremely loud).

The crows then told us about the following formula in fuzzy logic:

$$acceptable(k, v) = \text{AND}(dark(k), kinda(loud(v)))$$

$$\text{where } dark(k) = 1 - \frac{k}{255},$$

$$kinda(x) = \begin{cases} 2x & \text{if } x \leq 0.5, \\ 2 - 2x & \text{otherwise,} \end{cases}$$

$$\text{and } loud(v) = \min\{1, 2v\},$$

(i) For our first attempt at integrating the *CrowBot*, compute $acceptable(0, 0.45)$ and show all computation steps. (4pts)

For $k = 0$:

$$dark(0) = 1 - \frac{0}{255} = 1$$

For $v = 0.45$:

$$loud(0.45) = \min\{1, 2 \cdot 0.45\} = 0.9$$

Applying *kinda* to $loud(0.45)$:

$$kinda(0.9) = 2 - 2 \cdot 0.9 = 0.2$$

Therefore, $acceptable(0, 0.45)$:

$$acceptable(0, 0.45) = \text{AND}(1, 0.2) = \min\{1, 0.2\} = 0.2$$

¹A group of crows is called a *murder* in English. Yes, really.

(ii) We aim to build our next *CrowBot* so that it is at least 0.5 acceptable. However, black paint and precise speakers are expensive. What are the highest values for k and v so that $acceptable(k, v) \geq 0.5$ can be achieved? Show all computation steps. State briefly if the solutions for k and v are dependent on each other, i.e., if there is a trade-off. (6pts)

To ensure $acceptable(k, v) \geq 0.5$, we need both $dark(k)$ and $kinda(loud(v))$ to be at least 0.5.

For $dark(k) = 1 - \frac{k}{255}$, solving for k :

$$\begin{aligned} 1 - \frac{k}{255} &\geq 0.5 \\ \implies k &\leq 127.5 \end{aligned}$$

For $kinda(loud(v))$, we only consider this case first since we are looking for a large number:

$$\begin{aligned} 2 - 2 \cdot \min\{1, 2v\} &\geq 0.5 \\ \implies 2 \cdot \min\{1, 2v\} &\leq 1.5 \\ \implies v &\leq 0.375 \end{aligned}$$

Thus, the highest values for k and v are $k = 127.5$ and $v = 0.375$. These conditions are independent, meaning no direct trade-off between k and v exists as long as each individually satisfies its constraint.

4 Optimization

22pts

Scenario 2. We now want to implement a policy for a *CrowBot* using an artificial neural network (cf. Definition 5 in the appendix). For this, we choose a fully connected deep neural network with

- 8 input neurons,
- two hidden layers with 10 neurons each, and
- 4 output neurons

and without any biases. We encode the neural network via a weight vector $w \in \mathbb{R}^N$ where N is the number of weights.

We employ an evolutionary algorithm (cf. Algorithm 1 in the appendix) to optimize the weight vector w according to a given fitness function τ to be minimized.

- (i) Give the number of dimensions of the weight vector $|w| = N$ according to the network architecture described in Scenario 2. Show all your computation steps. (4pts)

$$N = 8 \times 10 + 10 \times 10 + 10 \times 4 = 80 + 100 + 40 = 220$$

(ii) In order to optimize the weight vector via an evolutionary algorithm, we consider a *one-point crossover* function for recombination. *One-point crossover* for two individuals $w, w' \in \mathbb{R}^N$ is defined by the following function:

$$\text{one-point_crossover}(\langle w_0, \dots, w_{N-1} \rangle, \langle w'_0, \dots, w'_{N-1} \rangle) = \langle w_0, \dots, w_{V-1}, w'_V, \dots, w'_{N-1} \rangle$$

where $V \sim [1; N-1] \subset \mathbb{N}$

Note that $V \sim \mathcal{V}$ describes that the value of V is sampled randomly from the elements of the set \mathcal{V} according to a uniform distribution.

Is this kind of recombination a good fit for optimizing weight vectors of neural networks? State potential benefits or problems of *one-point crossover* when applied to neural networks' weight vectors. Give your argument. (4pts)

One-point crossover like this applied to a weight vector might just randomly split one layer in the middle, combining parts of two neural networks that are not trained to work together. Since weights within a layer have no training-independent order, this split is especially meaningless. Thus, this should not be a good fit.

(iii) Alternatively, we want to find a policy for a single *CrowBot* directly via an evolutionary algorithm, consisting *only* of mutations! To this end, we consider only policies which run for exactly 1000 time steps and are fully defined by a parameter $\theta \in \mathcal{A}^{1000}$, i.e., a list of actions with length 1000.

We define that a smart mutation function $\text{mutate} : \mathcal{A}^{1000} \rightarrow \mathcal{A}^{1000}$ for such an evolutionary algorithm has the following properties:

- It changes exactly one action within the policy. That action can be any other action afterwards.
- The mutated policy is not fully determined by the parent policy but depends on random effects in some way.
- Through iterative application we can produce any arbitrary policy.

Give a smart mutation function **mutate**. (8pts)

Hint: You may use a probabilistic function **random** : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ by calling **random**(i, j), which returns a random integer within $[i; j) \subset \mathbb{N}$ according to a uniform distribution.

$$\begin{aligned} \text{mutate}(\theta) &= \theta' \\ \text{where for all } i \in [0, 1000) : \theta'_i &= \begin{cases} a & \text{if } i = r, \\ \theta_i & \text{otherwise,} \end{cases} \\ \text{where } a &\sim \mathcal{A} \setminus \{\theta_r\} \\ \text{and } r &= \text{random}(0, 1000) \end{aligned}$$

(iv) A selection function in an evolutionary algorithm receives a (multi-)set of $n + m$ solution candidates and reduces that to a (multi-)set of n solution candidates for any given $m, n \in \mathbb{N}$. To this end, for each solution candidate x a selection function considers its fitness $\tau(x) \in [1; 10] \subset \mathbb{R}$ where $\tau(x)$ should be minimized.

A nice selection function has the following properties:

- The best individual is always selected (elitism).
- Every individual has a non-zero chance to be selected.
- The second best individual has a better chance to be selected than the second worst individual.

Give a nice selection function and its type signature. (6pts)

Hint: You may call a probabilistic function `random()` that returns a random real number $r \in [0; 1) \subset \mathbb{R}$. You may also use the \sim notation from task (ii).

$$nice_selection : \wp^*(\mathbb{R}^N) \rightarrow \wp^*(\mathbb{R}^N)$$

$$nice_selection(X) = \{\mathbf{best}(X)\} \cup helper(X \setminus \{\mathbf{best}(X)\}, n - 1)$$

where $helper(X, n) = \begin{cases} X & \text{if } |X| = n, \\ \{\mathbf{best}(X)\} \cup helper(X \setminus \{\mathbf{best}(X)\}, n - 1) & \text{if } |X| > n \wedge \mathbf{random}() \leq 0.5, \\ helper(X, n) & \text{otherwise.} \end{cases}$

where `best` returns the single individual with the best fitness value in the given (multi-)set (with some arbitrary tie breaking).

5 Game Theory

10pts

We examine how a *CrowBot* can interact with real crows in order to fit right in. The *CrowBot* can mimic most of the crows' actions but usually does a little bit too much. Luckily, a leading crow expert advised us to write down the outcomes of crow-to-*CrowBot* interactions as the following two-player normal form game called the *crowding protocol*:

	blast off	pierce	torment	shut down	post online
flee	5, 3	4, 5	7, 3	3, 4	5, 7
pick	5, 7	3, 7	7, 2	2, 6	6, 2
provoke	3, 5	6, 3	2, 3	6, 6	7, 5
ignore	7, 3	2, 6	6, 2	7, 5	6, 7

- (i) Compute all pure Nash equilibria (cf. Definitions 8 and 10 in the appendix) for the *crowding protocol* by explicitly denoting all best responses in the matrix above. State all pure Nash equilibria separately below. (6pts)

The best responses are....

	blast off	pierce	torment	shut down	post online
flee	5, 3	4, 5	7, 3	3, 4	5, <u>7</u>
pick	5, <u>7</u>	3, <u>7</u>	7, 2	2, 6	6, 2
provoke	3, 5	6, 3	2, 3	6, <u>6</u>	7, 5
ignore	7, 3	2, 6	6, 2	7, 5	6, <u>7</u>

	blast off	pierce	torment	shut down	post online
flee	5, 3	4, 5	<u>7</u> , 3	3, 4	5, 7
pick	5, 7	3, 7	<u>7</u> , 2	2, 6	6, 2
provoke	3, 5	<u>6</u> , 3	2, 3	6, 6	<u>7</u> , 5
ignore	<u>7</u> , 3	2, 6	6, 2	<u>7</u> , 5	6, 7

	blast off	pierce	torment	shut down	post online
flee	5, 3	4, 5	<u>7</u> , 3	3, 4	5, <u>7</u>
pick	5, <u>7</u>	3, <u>7</u>	<u>7</u> , 2	2, 6	6, 2
provoke	3, 5	<u>6</u> , 3	2, 3	6, <u>6</u>	<u>7</u> , 5
ignore	<u>7</u> , 3	2, 6	6, 2	<u>7</u> , 5	6, <u>7</u>

In this case there are no joint pure best responses, and thus there are no pure NEs.

- (ii) Give all strategies on the Pareto front (cf. Definition 11 in the appendix) for the *crowding protocol*. Give your reasoning. (4pts)

There are three solutions on the Pareto-front (none of which are pure NEs since there are none). They are: (ignore, shut down) and (provoke, post online) with payoffs (7, 5) as well as (ignore, post online) with payoffs (6, 7).

6 Scientific Reading: Empowerment

20pts

Read the following (correct) explanation of the novel idea of *empowerment*, which we did not discuss in the lecture, and answer the questions afterwards.

The CoIn Team:

Generate a one-page explanation of empowerment in reinforcement learning. Include and explain relevant formulas in LaTeX format. Make the text suitable for computer science graduate students.

The ChatGPT Assistant:

Empowerment in reinforcement learning refers to the ability of an agent to maximize its influence over the environment. It is an intrinsic motivation that encourages agents to explore and take actions that can lead to long-term benefits, even if they do not result in immediate rewards.

One way to measure empowerment is through information-theoretic concepts, such as the mutual information between the agent's actions and the resulting state of the environment. Mutual information is a measure of the degree of dependence between two variables, and in this case, it can be used to capture how much the agent's actions influence the environment. The formula for mutual information is:

$$I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

Where X and Y represent two random variables, and $p(x, y)$, $p(x)$, and $p(y)$ represent their joint and marginal probabilities, respectively.

To apply empowerment in reinforcement learning, the agent maximizes the mutual information between its actions and the resulting state of the environment. This can be achieved through various algorithms, such as empowerment-based exploration or empowerment-driven reinforcement learn-

ing.

Empowerment has several benefits in reinforcement learning. First, it encourages agents to explore their environment more broadly, which can lead to the discovery of new strategies and solutions. Second, it can help agents to learn more robust and adaptable policies, as they focus on the long-term effects of their actions rather than just maximizing immediate rewards.

In conclusion, empowerment is a valuable concept in reinforcement learning that can help agents to become more effective at achieving their goals. By maximizing the mutual information between its actions and the environment, an agent can become more empowered and navigate complex and changing environments more efficiently.

The CoIn Team:

Give a very short overview of common problems when using empowerment.

The ChatGPT Assistant:

One common problem when using empowerment in reinforcement learning is that it can be difficult to estimate the mutual information between the agent's actions and the environment, which is required to compute empowerment. Another challenge is that maximizing empowerment can sometimes lead to suboptimal behavior if the agent becomes overly focused on exploring at the expense of exploiting known high-reward actions. Finally, the computational complexity of empowerment-based algorithms can be high, which may limit their practical applicability in some settings.

(i) Answer the following question partly generated by ChatGPT: Explain the concept of empowerment in reinforcement learning, and give an example of how it can be used to encourage more efficient exploration. (12pts)

The ChatGPT Assistant:

Empowerment in reinforcement learning refers to the agent's ability to maximize its influence over the environment, by taking actions that lead to long-term benefits even if they do not result in immediate rewards. One way to measure empowerment is through information-theoretic concepts, such as the mutual information between the agent's actions and the resulting state of the environment. Maximizing empowerment can encourage more efficient exploration, as the agent focuses on actions that have a high potential to influence the environment, rather than just maximizing immediate rewards.

(ii) Discuss one potential challenge in using empowerment as a learning signal for reinforcement learning agents. (8pts)

The ChatGPT Assistant:

One potential challenge in using empowerment as a learning signal for reinforcement learning agents is that it can be difficult to estimate the mutual information between the agent's actions and the environment. This can result in high computational complexity, which may limit the practical applicability of empowerment-based algorithms. One possible solution to address this issue is to use approximations or simplified models to estimate mutual information, or to combine empowerment with other learning signals such as reward-based reinforcement learning to balance exploration and exploitation more effectively.

Appendix: Definitions

Notation. \mathbb{B} is the set of truth values or Booleans, i.e., $\mathbb{B} = \{0, 1\}$. \mathbb{N} is the set of natural numbers starting from zero, i.e., $\mathbb{N} = \{0, 1, 2, \dots\}$ so that it holds that $\mathbb{B} \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{R} \subset \mathbb{C}$. \mathbb{P} denotes the space of probabilities and \mathbb{F} denotes the space of fuzzy values with $\mathbb{P} = \mathbb{F} = [0; 1] \subset \mathbb{R}$ being only discerned for semantic but not mathematical reasons. $\wp(X)$ denotes the power set of X . \mathbb{E} denotes the expected value. $\#$ denotes vector or sequence concatenation, i.e., given two vectors $\mathbf{x} = \langle x_1, \dots, x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, \dots, y_{|\mathbf{y}|} \rangle$, $\mathbf{x} \# \mathbf{y} = \langle x_1, \dots, x_{|\mathbf{x}|}, y_1, \dots, y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, \dots, x_{n-1} \rangle$ with length $n \in \mathbb{N}$ can also be written as $\langle x_i \rangle_{0 \leq i \leq n-1}$ for a new iteration variable i . $_-$ denotes unspecified function arguments ($f(-) = 0$ is the constant function that always returns zero, e.g.). For any finite set $X = \{x_0, \dots, x_n\}$, $|X| = n$ denotes the number of elements in X . For infinite sets, $|X| = \infty$.

Definition 1 (agent). Let \mathcal{A} be a set of actions. Let \mathcal{O} be a set of observations. An agent A can be given via a policy function $\pi : \mathcal{O} \rightarrow \mathcal{A}$. Given a time series of observations $\langle o_t \rangle_{t \in \mathcal{Z}}$ for some time space \mathcal{Z} the agent can thus generate a time series of actions $\langle a_t \rangle_{t \in \mathcal{Z}}$ by applying $a_t = \pi(o_t)$.

Definition 2 (optimization). Let \mathcal{X} be an arbitrary state space. Let \mathcal{T} be an arbitrary set called target space and let \leq be a total order on \mathcal{T} . A total function $\tau : \mathcal{X} \rightarrow \mathcal{T}$ is called target function. Optimization (minimization/-maximization) is the procedure of searching for an $x \in \mathcal{X}$ so that $\tau(x)$ is optimal (minimal/maximal). Unless stated otherwise, we assume minimization. An optimization run of length $g + 1$ is a sequence of states $\langle x_t \rangle_{0 \leq t \leq g}$ with $x_t \in \mathcal{X}$ for all t .

Let $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \rightarrow \mathcal{T}) \rightarrow \mathcal{X}$ be a possibly randomized or non-deterministic function so that the optimization run $\langle x_t \rangle_{0 \leq t \leq g}$ is produced by calling e repeatedly, i.e., $x_{t+1} = e(\langle x_u \rangle_{0 \leq u \leq t}, \tau)$ for all t , $1 \leq t \leq g$, where x_0 is given externally (e.g., $x_0 =_{\text{def}} 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

Algorithm 1 (basic evolutionary algorithm). Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be a population-based optimization process. The process \mathcal{E} continues via an evolutionary algorithm if E has the form

$$E(\langle X_u \rangle_{0 \leq u \leq t}, \tau) = X_{t+1} = \text{selection}(X_t \uplus \text{variation}(X_t))$$

where *selection* and *variation* are possibly randomized or non-deterministic functions so that for any $X \in \wp^*(\mathcal{X})$ it holds that $|\text{selection}(X)| \leq |X|$ and $|\text{selection}(X \uplus \text{variation}(X))| = |X|$.

Definition 3 (optimization (policy)). Let $\mathcal{X} = \Pi$ be a policy space. Let $\mathcal{D} = (\Pi, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$ be an optimization process according to Definition 2. \mathcal{D} is called a policy optimization process.

Definition 4 (Markov decision process (MDP)). Let \mathcal{A} be a set of actions. Let \mathcal{S} be a set of states. Let A be an agent given via a policy function $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{P}$ is the *transition probability function* often written as $P(s'|s, a) = \Pr(s_{t+1} = s' \mid s_t = s \wedge a_t = a)$.

A Markov decision process run for policy π is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \tau, \pi, \langle s_t \rangle_{t \in \mathbb{Z}}, \langle a_t \rangle_{t \in \mathbb{Z}})$ where

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

and where τ is to be minimized (maximized) and usually has a form similar to

$$\begin{aligned} \text{accumulated cost (reward)} \tau(\pi) &=_{\text{def}} \sum_{t \in \mathbb{Z}} R(s_t, a_t, s_{t+1}) \\ \text{or discounted expected cost (reward)} \tau(\pi) &=_{\text{def}} \mathbb{E} \left[\sum_{t \in \mathbb{Z}} \gamma^t \cdot R(s_t, a_t, s_{t+1}) \right] \end{aligned}$$

where $\gamma \in [0; 1] \subset \mathbb{R}$ is called a discount factor.

A decision process generates a (possibly infinite) series of rewards $\langle r_t \rangle_{t \in \mathbb{Z}}$ with $r_t = R(s_t, a_t, s_{t+1})$.

Algorithm 2 (optimal policy). Let $V^* : \mathcal{S} \rightarrow \mathcal{T}$ be the *true value function* of a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$. The optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ is given via

$$\pi^*(s_t) = \arg \max_{a \in \mathcal{A}} V^*(s')$$

where $s' \sim P(s' | s_t, a)$ is the follow-up state when executing action a in state s_t .

Definition 5 (neural network). A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers L_1, \dots, L_r where each layer L_l consists of $|L_l|$ cells $C_{l,1}, \dots, C_{l,|L_l|}$, which make up the graph's vertices, and each cell $C_{l,c}$ of the layer L_l is connected to all cells of the previous layer, i.e., $C_{l-1,d}$ for $d = 1, \dots, |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}, e = 1, \dots, |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \rightarrow \mathbb{R}$, the vector of all edge weights

$$\mathbf{w} = \langle E_{l,c,e} \rangle_{l=1, \dots, r, \ c=1, \dots, |L_l|, \ e=1, \dots, |L_{l-1}|}$$

and the vector of all cell biases

$$\mathbf{b} = \langle B_{l,c} \rangle_{l=1, \dots, r, \ c=1, \dots, |L_l|}$$

with $B_{l,c} \in \mathbb{R}$ define the network's functionality. The combined vector $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b}$ is called the network \mathcal{N} 's parameters.

A network's output given an input $\mathbf{x} \in \mathbb{R}^p$ is given via

$$\mathbf{y} = \mathcal{N}(\mathbf{x}) = \langle O(l, c) \rangle_{c=1, \dots, |L_r|} \in \mathbb{R}^q$$

$$\text{where } O(l, c) = \begin{cases} x_c & \text{if } l = 0, \\ f(B_{l,c} + \sum_{i=1}^{|L_{l-1}|} E_{l,c,i} \cdot O(l-1, i)) & \text{otherwise.} \end{cases}$$

Definition 6 (training of a neural network (shortened)). Let $\mathcal{N} : \mathbb{R}^p \rightarrow \mathbb{R}^q$ be a neural network with n weights $\overline{\mathcal{N}} = \mathbf{w} \# \mathbf{b} \in \mathbb{R}^n$.

- Let $(\mathcal{O}, \mathcal{A}, \mathcal{T}, e, R)$ be a decision process (cf. Definition ??) for which policy $\pi_{\overline{\mathcal{N}}} : \mathcal{O} \rightarrow \mathcal{A}$ yields (possibly randomized or non-deterministic) rewards $\langle r_t \rangle_{t \in \mathcal{Z}}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls \mathcal{N} to produce its output, for example

$$\pi_{\overline{\mathcal{N}}}(o) = \mathcal{N}(o)$$

for $\mathcal{O} \subseteq \mathbb{R}^p, \mathcal{A} \subseteq \mathbb{R}^q$ or if suitable translations exist.

If τ is of the form

$$\tau(\overline{\mathcal{N}}) = -\mathbb{E} \left[\sum_{t \in \mathcal{Z}} \gamma^t \cdot r_t \right]$$

or a similar form, the process of training \mathcal{N} is called policy-based reinforcement learning.

- Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process (cf. Definition 4) for which we run policy $\pi_{\overline{\mathcal{N}}} : \mathcal{S} \rightarrow \mathcal{A}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls \mathcal{N} to produce its output, for example

$$\pi_{\overline{\mathcal{N}}}(s) = \arg \max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim P(s'|s,a)} [\mathcal{N}(s')]$$

for $\mathcal{S} \times \mathcal{A} \subseteq \mathbb{R}^p$ with $q = 1$ or if suitable translations exist.

Let $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{T}$ be the expected reward of executing an action in a given state, i.e., $R(s, a) = \mathbb{E}[R(s, a, s')]$ where $s' \sim P(s'|s, a)$. Let $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ be a (possibly randomized or non-deterministic) transition function, i.e., $T(s, a) = s'$ where $s' \sim P(s'|s, a)$. Let $\gamma \in [0; 1]$ be a discount factor. Let $V_{\pi_{\overline{\mathcal{N}}}} : \mathcal{S} \rightarrow \mathbb{R}$ be the total discounted reward that policy $\pi_{\overline{\mathcal{N}}}$ generates when starting in state s , i.e.,

$$V_{\pi_{\overline{\mathcal{N}}}}(s) = R(s, \pi_{\overline{\mathcal{N}}}(s)) + \gamma \cdot V_{\pi_{\overline{\mathcal{N}}}}(T(s, \pi_{\overline{\mathcal{N}}}(s))).$$

Note that for $\gamma < 1$ we can abort this recursive computation once the effect of the further recursive part is sufficiently small. Note that we may also have a fixed recursion depth or that $T(s^\dagger, \cdot)$ might not be defined for all $s^\dagger \in \mathcal{S}$, which are then called terminal states and also cause the recursion to end.

Let $\mathbb{S} = \{\mathbf{s}_i : i = 1, \dots, N\} \subseteq \mathcal{S}$ be a set of training states. If τ is of the form

$$\tau(\overline{\mathcal{N}}) = -\frac{1}{N} \cdot \sum_{i=1}^N V_{\pi_{\overline{\mathcal{N}}}}(\mathbf{s}_i)$$

or a similar form, the process of training \mathcal{N} is called value-based reinforcement learning.

Definition 7 (normal-form game). Let $G = \{G^{[1]}, \dots, G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times \dots \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i . Let $\chi : \mathcal{A} \rightarrow \mathcal{T}$ be a utility function for the joint action space \mathcal{A} and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times \dots \times \mathcal{T}^{[N]}$ where $\mathcal{T}^{[i]}$ is the target space of agent $G^{[i]}$ for all i . Unless stated otherwise, the utility χ is to be maximized. From χ we can derive a set of single-agent utility functions $\chi^{[i]} : \mathcal{A} \rightarrow \mathcal{T}^{[i]}$ for all i . A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Definition 8 (strategy). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. In a single iteration of the game, an agent $G^{[i]}$'s behavior is given by a (possibly randomized) policy $\pi^{[i]} : () \rightarrow \mathcal{A}^{[i]}$. Then, $\pi^{[i]}$ is also called $G^{[i]}$'s *strategy*. If multiple iterations of the game are played, an agent $G^{[i]}$'s behavior can be given by a (possibly randomized) policy $\pi^{[i]} : \mathcal{A}^n \rightarrow \mathcal{A}^{[i]}$ where n is a number of previous iterations. The agent's strategy is then conditioned on a list of previous joint actions.

An agent $G^{[i]}$ whose actions are given via a policy $\pi^{[i]}$ of the form $\pi^{[i]}(-) = a^{[i]}$ for some action $a^{[i]} \in \mathcal{A}^{[i]}$ is playing a pure strategy.

An agent $G^{[i]}$ whose actions are given via a policy $\pi^{[i]}$ of the form $\pi^{[i]}(-) \sim A^{[i]}$ according to some distribution over $A^{[i]} \subseteq \mathcal{A}^{[i]}$ is playing a mixed strategy.

If a mixed strategy is based on a uniform distribution over actions $A^{[i]} \subseteq \mathcal{A}^{[i]}$, we write $\pi^{[i]} = A^{[i]}$ as a shorthand.

Definition 9 (best response). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, \dots, G^{[i-1]}, G^{[i+1]}, \dots, G^{[N]}$, i.e., all agents except $G^{[i]}$. Let $\pi^{[i]} \oplus \pi^{[-i]}$ be a joint strategy of all agents then. Given a strategy $\pi^{[-i]}$ for all agents except $G^{[i]}$, $G^{[i]}$'s *best response* is the strategy $\pi^{*[i]}$ so that for all strategies $\pi'^{[i]}$ it holds that

$$\chi^{[i]}((\pi^{*[i]} \oplus \pi^{[-i]})(-)) \geq \chi^{[i]}((\pi'^{[i]} \oplus \pi^{[-i]})(-)).$$

Definition 10 (Nash equilibrium). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played for a single iteration. A joint strategy π is a *Nash equilibrium* iff for all agents $G^{[i]}$ it holds that $\pi^{[i]}$ is the best response to $\pi^{[-i]}$.

Definition 11 (Pareto front for strategies). Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(-) = (\pi^{[1]}(-), \dots, \pi^{[|G|]}(-))$ Pareto-dominates another joint strategy $\pi'(-) = (\pi'^{[1]}(-), \dots, \pi'^{[|G|]}(-))$ iff for all agents $G^{[i]}$ it holds that

$$\chi^{[i]}(\pi(-)) \geq \chi^{[i]}(\pi'(-))$$

and there exists some agent $G^{[j]}$ so that

$$\chi^{[j]}(\pi(-)) > \chi^{[j]}(\pi'(-)).$$

A joint strategy π is Pareto-optimal iff there is no other strategy π' so that π' Pareto-dominates π .

The set of all Pareto-optimal strategies is called the Pareto front.