Computational Intelligence
Definition Sheet

Thomas Gabor
LMU Munich

Maximilian Zorn
LMU Munich

Claudia Linnhoff-Popien
LMU Munich

winter term 2023/2024

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., $\mathbb{N} = \{0, 1, 2, ...\}$ 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 $\mathbf{x} = \langle x_1, ..., x_{|\mathbf{x}|} \rangle$ and $\mathbf{y} = \langle y_1, ..., y_{|\mathbf{y}|} \rangle$, $\mathbf{x} +\!\!+ \mathbf{y} = \langle x_1, ..., x_{|\mathbf{x}|}, y_1, ..., y_{|\mathbf{y}|} \rangle$. A vector $\langle x_0, ..., x_{n-1} \rangle$ 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 $X = \{x_0, ..., x_n\}$, $|X| = n$ 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 $\pi : \mathcal{O} \to \mathcal{A}$. 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 $a_t = \pi(o_t)$.

Algorithm 1 (brute force (policy))**.** Let $\mathcal{A}$ be a set of actions. Let $\mathcal{O}$ be a set of observations. Let $\Gamma \subseteq (\mathcal{O} \to \mathcal{A}) \to \mathbb{B}$ be a space of goal predicates on policy functions. Let $\gamma \in \Gamma$ be a goal predicate. We assume that the policy space $\Pi \subseteq \mathcal{O} \to \mathcal{A}$ is enumerable, i.e., $\Pi = \langle \pi_i \rangle_{i \in \mathbb{N}}$. Brute force starting from i is given via the function

$$b(i) = \begin{cases} \pi_i & \text{if } \gamma(\pi_i), \\ b(i+1) & \text{otherwise.} \end{cases}$$

If not further specified, the call to $b(0)$ is called brute force search for an agent policy. Usually, an additional termination condition is specified.

Algorithm 2 (random search (policy))**.** Let $\mathcal{A}$ be a set of actions. Let $\mathcal{O}$ be a set of observations. Let $\Gamma \subseteq (\mathcal{O} \to \mathcal{A}) \to \mathbb{B}$ be a space of goal predicates on policy functions. Let $\gamma \in \Gamma$ be a goal predicate. We assume that the policy space $\Pi \subseteq \mathcal{O} \to \mathcal{A}$ can be sampled from, i.e., $\pi \sim \Pi$ returns a random element from Π. Random search for n samples is given via the function

$$\rho(n) = \begin{cases} \emptyset & \text{if } n = 0, \\ \pi & \text{if } n > 0 \text{ and } \gamma(\pi) \text{ where } \pi \sim \Pi, \\ \rho(n-1) & \text{otherwise.} \end{cases}$$

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 $\tau : \mathcal{X} \to \mathcal{T}$ 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 $e : \langle \mathcal{X} \rangle \times (\mathcal{X} \to \mathcal{T}) \to \mathcal{X}$ 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., $x_0 =_{def} 42$) or chosen randomly (e.g., $x_0 \sim \mathcal{X}$). An optimization process is a tuple $(\mathcal{X}, \mathcal{T}, \tau, e, \langle x_t \rangle_{0 \leq t \leq g})$.

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.

Algorithm 3 (simulated annealing)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let *neighbors* : $\mathcal{X} \to \wp(\mathcal{X})$ be a function that returns a set of neighbors of a given state $x \in \mathcal{X}$. Let $\kappa : \mathbb{N} \to \mathbb{R}$ be a temperature schedule, i.e., a function that returns a temperature value for each time step. Let $A : \mathcal{T} \times \mathcal{T} \times \mathbb{R} \to \mathbb{P}$ with $\mathbb{P} = [0; 1] \subset \mathbb{R}$ be a function that returns an acceptance probability given two target values and a temperature. Commonly, we use

$$A(Q, Q', K) = e^{\frac{-(Q'-Q)}{K}}$$

for $\mathcal{T} \subseteq \mathbb{R}$. The process $\mathcal{D}$ continues via simulated annealing if e is of the form

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = \begin{cases} x'_t & \text{if } \tau(x'_t) \leq \tau(x_t) \text{ or } r \leq A\big(\tau(x_t), \tau(x'_t), \kappa(t)\big), \\ x_t & \text{otherwise,} \end{cases}$$

where $x'_t \sim \textit{neighbors}(x_t)$ and $r \sim \mathbb{P}$ are drawn at random for each call to e.

Definition 4 (population-based optimization)**.** Let $\mathcal{X}$ be a state space. Let $\mathcal{T}$ be a target space with total order $\leq$. Let $\tau : \mathcal{X} \to \mathcal{T}$ be a target function. A tuple $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_t \rangle_{0 \leq t \leq g})$ is a population-based optimization process iff $X_t \in \wp^*(\mathcal{X})$ for all t and $E : \langle \wp^*(\mathcal{X}) \rangle \times (\mathcal{X} \to \mathcal{T}) \to \wp^*(\mathcal{X})$ is a possibly randomized, non-deterministic, or further parametrized function so that the population-based optimization run is produced by calling E repeatedly, i.e., $X_{t+1} = E(\langle X_u \rangle_{0 \leq u \leq t}, \tau)$ where X_0 is given externally or chosen randomly.

Definition 5 (population-based optimization (alternate))**.** An optimization process $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_t \rangle_{0 \leq t \leq g})$ is called population-based iff $\mathcal{X}$ has the form $\mathcal{X} = \wp^*(\mathcal{Y})$ for some other state space $\mathcal{Y}$.

Algorithm 4 (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

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

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

Theorem 1 (no free lunch [1, 5])**.** As measured by sample efficiency, i.e., the achieved minimal value of τ per evaluations of $\tau(x)$ for some new $x \in \mathcal{X}$ for finite $\mathcal{X}$, all optimization algorithms perform the same when averaged over all possible target functions τ. So, for any search/optimization algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.

Definition 6 (multi-objective optimization)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be an optimization process. $\mathcal{E}$ is a multi-objective optimization process iff the target space $\mathcal{T}$ has the form $\mathcal{T} = \mathcal{T}_0 \times \cdots \times \mathcal{T}_{N-1}$ for some $N \in \mathbb{N}$ with $<_i$ being a total order on $\mathcal{T}_i$ for any $i \in [0; N-1] \subset \mathbb{N}$. Unless stated otherwise, we assume that no single total order on $\mathcal{T}$ is available. However, we can construct a partial order $\prec$ between target values $g, g' \in \mathcal{T}$ so that

$$(g_0, ..., g_{N-1}) \prec (g'_0, ..., g'_{N-1}) \iff \forall i \in [0; N-1] \subset \mathbb{N} \ : \ g_i < g'_i,$$

which is sufficient to adapt many standard optimization algorithms.

Definition 7 (Pareto front for optimization)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be a multi-objective optimization process with $\prec$ being a partial order on the multi-objective target space $\mathcal{T}$.

- A solution candidate x Pareto-dominates a solution candidate x', written $x \prec x'$ (assuming minimization), iff $\tau(x) \prec \tau(x')$.
- A solution candidate x is Pareto-optimal if there exists no other solution candidate $x' \in \mathcal{X}$ so that $x' \prec x$.
- The set of all Pareto-optimal solution candidates in $\mathcal{X}$ is called the Pareto front of $\mathcal{X}$ (w.r.t. $\prec$).

Algorithm 5 (gradient descent)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \mathcal{X} \to \mathcal{T}$ be the first derivative of τ. The process $\mathcal{E}$ continues via gradient descent (with learning rate $\alpha \in \mathbb{R}^+$) if e is of the form

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha \cdot \tau'(x_t).$$

The learning rate α can also be given as a function, usually $\alpha : \mathbb{N} \to \mathbb{R}$, so that $e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha(t) \cdot \tau'(x_t)$.
If the computation of τ' is stochastic, usually because it is derived from a stochastic sampling of data points, this process is called stochastic gradient descent (SGD).

Algorithm 6 (gradient descent (policy))**.** Let π_θ be a policy π that depends on a vector of continuous parameters $\theta \in \Theta$, usually with $\Theta = \mathbb{R}^N$ for some N. Let $\tau : \Theta \to \mathcal{T}$ be a target function on the parameters θ of a policy π_θ. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \Theta \to \mathcal{T}$ be the first derivative of τ, i.e., $\tau'(\theta) = \frac{\partial \tau(\theta)}{\partial \theta}$. If $\mathcal{E} = (\Theta, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ is an optimization process that continues via gradient descent, $\mathcal{E}$ is a process of policy optimization via gradient descent.

Definition 8 (neural network)**.** A neural network (NN) is a function $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ with p inputs and q outputs. This function is defined via a graph made up of r layers $L_1, ..., L_r$ where each layer L_l consists of $|L_l|$ cells $C_{l,1}, ..., C_{l,|L_l|}$, 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 $d = 1, ..., |L_{l-1}|$, via the graph's edges. Each edge of a cell $C_{l,c}$ is assigned an edge weight $E_{l,c,e} \in \mathbb{R}, e = 1, ..., |L_{l-1}|$. Given a fixed graph structure and activation function $f : \mathbb{R} \to \mathbb{R}$, the vector of all edge weights

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

and the vector of all cell biases

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

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

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

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

Theorem 2 (Kolmogorov-Arnold representation [4])**.** Any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ for some $n \in \mathbb{N}$ can be written as a finite composition of continuous functions of a single variable ($f_i : \mathbb{R} \to \mathbb{R}$ for $i \in \mathbb{R}$, $1 \leq i \leq n$ for some $n \in \mathbb{N}$) and addition ($\_ + \_ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$).

Definition 9 (decision process)**.** Let $\mathcal{A}$ be a set of actions. Let $\mathcal{O}$ be a set of observations. Let A be an agent given via a policy function $\pi : \mathcal{O} \to \mathcal{A}$. Let $R : \mathcal{A} \to \mathcal{T}$ be a possibly randomized, non-deterministic, or hidden-state *cost* (*reward*) function. A decision process is given by a tuple $(\mathcal{O}, \mathcal{A}, \mathcal{T}, e, R)$ where e generates new observations given the agent's previous actions.
A decision process run for policy π is a tuple $(\mathcal{O}, \mathcal{A}, \mathcal{T}, \tau, \pi, \langle o_t \rangle_{t \in \mathcal{Z}}, \langle a_t \rangle_{t \in \mathcal{Z}})$ where τ is to be minimized (maximized) and usually has a form similar to

$$\textit{accumulated cost (reward)}\ \tau(\pi) =_{def} \sum_{t \in \mathcal{Z}} R(a_t)$$

$$\text{or } \textit{discounted expected cost (reward)}\ \tau(\pi) =_{def} \mathbb{E}\Big[\sum_{t \in \mathcal{Z}} \gamma^t \cdot R(a_t)\Big]$$

where $\gamma \in [0; 1] \subset \mathbb{R}$ is called a discount factor.
A decision process run generates a (possibly infinite) series of rewards $\langle r_t \rangle_{t \in \mathcal{Z}}$ with $r_t = R(a_t)$.

Definition 10 (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 $\pi : \mathcal{S} \to \mathcal{A}$. Let $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{T}$ be a possibly randomized *cost* (*reward*) function. A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ where $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{P}$ is the *transition probability function* often written as $P(s'|s,a) = \Pr(s_{t+1} = s' \,|\, s_t = s \wedge a_t = a)$.
A Markov decision process run for policy π is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \tau, \pi, \langle s_t \rangle_{t \in \mathcal{Z}}, \langle a_t \rangle_{t \in \mathcal{Z}})$ 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

$$\textit{accumulated cost (reward)}\ \tau(\pi) =_{def} \sum_{t \in \mathcal{Z}} R(s_t, a_t, s_{t+1})$$

$$\text{or } \textit{discounted expected cost (reward)}\ \tau(\pi) =_{def} \mathbb{E}\Big[\sum_{t \in \mathcal{Z}} \gamma^t \cdot R(s_t, a_t, s_{t+1})\Big]$$

where $\gamma \in [0;1] \subset \mathbb{R}$ is called a discount factor.
A decision process generates a (possibly infinite) series of rewards $\langle r_t \rangle_{t \in \mathcal{Z}}$ with $r_t = R(s_t, a_t, s_{t+1})$.

Algorithm 7 (optimal policy)**.** Let $V^* : \mathcal{S} \to \mathcal{T}$ be the *true value function* of a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$. The optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ 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.

Theorem 3 (Bellman equation)**.** Let $(\mathcal{S}, \mathcal{A}, \mathcal{T}, P, R)$ be a Markov decision process. Let $R : \mathcal{S} \times \mathcal{A} \to \mathcal{T}$ 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 $\gamma \in [0;1) \subseteq \mathbb{R}$ be a temporal discount factor.
The expected reward of a policy π being executed starting from state s is given via π's value function

$$V^\pi(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \cdot V^\pi(s').$$

The value function of the optimal policy π^* is given via

$$V^{\pi^*}(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot V^{\pi^*}(s') \right).$$

Definition 11 (training of a neural network)**.** Let $\mathcal{N} : \mathbb{R}^p \to \mathbb{R}^q$ be a neural network with n weights $\overline{\mathcal{N}} = \mathbf{w} +\!\!+ \mathbf{b} \in \mathbb{R}^n$ as in Definition 8. Note that thus $|\overline{\mathcal{N}}| = n$. Let $\tau : \mathbb{R}^n \to \mathbb{R}$ be a target function as in Definition 2. Note that thus $\mathcal{T} = \mathbb{R}$. The process of optimizing the network weights $\overline{\mathcal{N}}$ so that $\tau(\overline{\mathcal{N}})$ becomes minimal is called training.

- Let $\mathbb{T} = \{(\mathbf{x}_i, \mathbf{y}_i) \,:\, i = 1, ..., N\}$ be a set of N points of training data, where $\mathbf{x}_i \in \mathbb{R}^p, \mathbf{y}_i \in \mathbb{R}^q$ for all i.
  If τ is of the form
  $$\tau(\overline{\mathcal{N}}) = \sum_{i=1}^{N} (\mathcal{N}(\mathbf{x}_i) - \mathbf{y}_i)^2$$
  or a similar form, the process of training $\mathcal{N}$ is called supervised learning.

- Let $(\mathcal{O}, \mathcal{A}, \mathcal{T}, e, R)$ be a decision process (cf. Definition 9) for which policy $\pi_{\overline{\mathcal{N}}} : \mathcal{O} \to \mathcal{A}$ 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
  $$\tau(\overline{\mathcal{N}}) = -\mathbb{E}\Big[\sum_{t \in \mathcal{Z}} \gamma^t \cdot r_t\Big]$$
  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 10) for which we run policy $\pi_{\overline{\mathcal{N}}} : \mathcal{S} \to \mathcal{A}$. Note that $\pi_{\overline{\mathcal{N}}}$ in some way calls $\mathcal{N}$ to produce its output, for example
  $$\pi_{\overline{\mathcal{N}}}(s) = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim P(s'|s,a)} \left[\, \mathcal{N}(s') \,\right]$$
  for $\mathcal{S} \times \mathcal{A} \subseteq \mathbb{R}^p$ with $q = 1$ or if suitable translations exist.
  Let $R : \mathcal{S} \times \mathcal{A} \to \mathcal{T}$ 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 $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ 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 $V_{\pi_{\overline{\mathcal{N}}}} : \mathcal{S} \to \mathbb{R}$ be the total discounted reward that policy $\pi_{\overline{\mathcal{N}}}$ generates when starting in state s, i.e.,
  $$V_{\pi_{\overline{\mathcal{N}}}}(s) = R\big(s, \pi_{\overline{\mathcal{N}}}(s)\big) + \gamma \cdot V_{\pi_{\overline{\mathcal{N}}}}\big(T(s, \pi_{\overline{\mathcal{N}}}(s))\big).$$
  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 $T(s^\dagger, \_)$ 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 $\mathbb{S} = \{\mathbf{s}_i \,:\, i = 1, ..., N\} \subseteq S$ 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.

Example 1. Various methods for reinforcement learning in an MDP setting with neural networks.

method name	network type	policy
policy-based	$\mathcal{N}: \mathcal{S} \to \mathcal{A}$	$\pi_{\overline{\mathcal{N}}}(s) = \mathcal{N}(s)$
value-based (V)	$\mathcal{N}: \mathcal{S} \to \mathbb{R}$	$\pi_{\overline{\mathcal{N}}}(s) = \arg\max_{a \in \mathcal{A}} \mathbb{E}_{s' \sim P(s'\|s,a)} \left[R(s,a) + \mathcal{N}(s') \right]$
value-based (Q)	$\mathcal{N}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	$\pi_{\overline{\mathcal{N}}}(s) = \arg\max_{a \in \mathcal{A}} \mathcal{N}(s,a)$

Definition 12 (multi-agent system)**.** Let $G = \{G^{[1]}, ..., G^{[N]}\}$ be a set of $|G| = N$ agents with observation spaces $\mathcal{O}^{[i]}$ and action spaces $\mathcal{A}^{[i]}$ controlled by policies $\pi^{[i]}$ for all $i = 1, ..., N$, respectively. The multi-agent system G then takes a joint action $a \in \mathcal{A}$ with $\mathcal{A} = \mathcal{A}^{[1]} \times ... \times \mathcal{A}^{[N]}$ after making a joint observation $o \in \mathcal{O}$ with $\mathcal{O} = \mathcal{O}^{[1]} \times ... \times \mathcal{O}^{[N]}$ based on the joint policy $\pi(o^{[1]}, ..., o^{[N]}) = (a^{[1]}, ..., a^{[N]})$ where $a^{[i]} = \pi^{[i]}(o^{[i]})$ for all i.

Definition 13 (normal-form game)**.** Let $G = \{G^{[1]}, ..., G^{[N]}\}$ be a set of $|G| = N$ agents. Let $\mathcal{A} = \mathcal{A}^{[1]} \times ... \times \mathcal{A}^{[N]}$ be the space of joint actions where $\mathcal{A}^{[i]}$ is the set of actions available to agent $G^{[i]}$ for all i. Let $\chi : \mathcal{A} \to \mathcal{T}$ be a utility function for the joint action space $\mathcal{A}$ and the joint target space $\mathcal{T} = \mathcal{T}^{[1]} \times ... \times \mathcal{T}^{[N]}$ 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 $\chi^{[i]} : \mathcal{A} \to \mathcal{T}^{[i]}$ for all i. A tuple $(G, \mathcal{A}, \mathcal{T}, \chi)$ is called a *normal-form game*.

Definition 14 (common-payoff game)**.** A normal-form game $(G, \mathcal{A}, \mathcal{T}, \chi)$ is a *common-payoff game* iff for any two agents $G^{[i]}, G^{[j]}$ and for any joint action $a = (a^{[1]}, ..., a^{[N]}) \in \mathcal{A}$ it holds that $\chi^{[i]}(a) = \chi^{[j]}(a)$.

Definition 15 (zero-sum game)**.** A normal-form game $(G, \mathcal{A}, \mathcal{T}, \chi)$ is a *zero-sum game* iff for any joint action $a = (a^{[1]}, ..., a^{[N]}) \in \mathcal{A}$ it holds that

$$\sum_{i=1}^{|G|} \chi^{[i]}(a) = 0.$$

Definition 16 (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 $\pi^{[i]} : () \to \mathcal{A}^{[i]}$. 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 $\pi^{[i]} : \mathcal{A}^n \to \mathcal{A}^{[i]}$ 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 17 (Pareto front for strategies)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. A joint strategy $\pi(\_) = (\pi^{[1]}(\_), ..., \pi^{[|G|]}(\_))$ Pareto-dominates another joint strategy $\pi'(\_) = (\pi'^{[1]}(\_), ..., \pi'^{[|G|]}(\_))$ 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.

Definition 18 (best response)**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game. Let $\pi^{[-i]}$ be a joint strategy of agents $G^{[1]}, ..., G^{[i-1]}, G^{[i+1]}, ..., G^{[N]}$, 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 19 (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 $\pi^{[-i]}$.

Theorem 4 (Nash's theorem)**.** Every game $(G, \mathcal{A}, \mathcal{T}, \chi)$ with $|G|$ and $|\mathcal{A}|$ finite has a Nash equilibrium.

Definition 20 (evolutionary stable strategy [3])**.** Let $(G, \mathcal{A}, \mathcal{T}, \chi)$ be a normal-form game played by two players $i, -i$ with the same action space $\mathcal{A}^{[i]} = \mathcal{A}^{[-i]}$. A strategy $\pi^{[i]}$ for agent $G^{[i]}$ is an *evolutionary stable strategy* iff

- $\pi = \pi^{[i]} \oplus \pi^{[-i]}$ with $\pi^{[-i]} = \pi^{[i]}$ is a Nash equilibrium and
- for every other strategy $\pi'^{[i]} = \pi'^{[-i]} \neq \pi^{[i]}$ it holds that

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

Example 2 (Hawk–Dove game in π-calculus)**.** Let `Game` be an external function that returns the outcome for the first player given two strategies.

$$\begin{aligned}
\text{Let } \mathit{Agent}^{[i]}(\mathit{strategy}, \mathit{total_payoff}) = {} & \overline{\mathit{play}}\langle \mathit{strategy}, \mathit{total_payoff} \rangle. \\
& 0 \\
& \mid \mathit{play}(\mathit{opponent_strategy}, \mathit{opponent_total_payoff}). \\
& \quad \mathit{Agent}^{[i]}(\mathit{new_strategy}, \mathit{new_total_payoff}) \\
\text{where } \mathit{new_total_payoff} = {} & \mathit{total_payoff} + \texttt{Game}(\mathit{strategy}, \mathit{opponent_strategy}) \\
\text{and } \mathit{new_strategy} = {} & \begin{cases} \mathit{opponent_strategy} & \text{if } \mathit{opponent_total_payoff} > \mathit{total_payoff}, \\ \mathit{strategy} & \text{otherwise.} \end{cases}
\end{aligned}$$

A Hawk–Dove game with a population of 20 can then be constructed as the process

$$\begin{aligned}
\mathit{Population} = {} & \mathit{Agent}^{[1]}(\text{“}\mathit{Hawk}\text{”}, 0) \mid ... \mid \mathit{Agent}^{[10]}(\text{“}\mathit{Hawk}\text{”}, 0) \\
& \mid \mathit{Agent}^{[11]}(\text{“}\mathit{Dove}\text{”}, 0) \mid ... \mid \mathit{Agent}^{[20]}(\text{“}\mathit{Dove}\text{”}, 0).
\end{aligned}$$

Definition 21 (π-process [2])**.** Let N be a set of names ($N = \{$"a", "b", "c", ...$\}$, e.g.). $\mathbb{L}_\pi$ is the set of valid processes in the π-calculus given inductively via:

(null process) $0 \in \mathbb{L}_\pi$,

(τ prefix) if $P \in \mathbb{L}_\pi$, then $\tau.P \in \mathbb{L}_\pi$,

(receiving prefix) if $a, x \in N$ and $P \in \mathbb{L}_\pi$, then $a(x).P \in \mathbb{L}_\pi$,

(sending prefix) if $a, x \in N$ and $P \in \mathbb{L}_\pi$, then $\overline{a}\langle x\rangle.P \in \mathbb{L}_\pi$,

(choice) if $P, Q \in \mathbb{L}_\pi$, then $(P + Q) \in \mathbb{L}_\pi$,

(concurrency) if $P, Q \in \mathbb{L}_\pi$, then $(P \mid Q) \in \mathbb{L}_\pi$,

(scoping) if $x \in N$, $P \in \mathbb{L}_\pi$, then $(\nu x)\ P \in \mathbb{L}_\pi$,

(replication) if $P \in \mathbb{L}_\pi$, then $!P \in \mathbb{L}_\pi$.

Any element $P \in \mathbb{L}_\pi$ is called a π-process. If the binding order is clear, we leave out parentheses.
The free names of a π-process $P \in \mathbb{L}_\pi$, written $\mathfrak{F}(P)$ with $\mathfrak{F} : \mathbb{L}_\pi \to \wp(N)$ are given inductively via:

- $\mathfrak{F}(0) = \emptyset$,
- $\mathfrak{F}(\tau.P) = \mathfrak{F}(P)$,
- $\mathfrak{F}(a(x).P) = \{a\} \cup (\mathfrak{F}(P) \setminus \{x\})$,
- $\mathfrak{F}(\overline{a}\langle x\rangle.P) = \{a, x\} \cup \mathfrak{F}(P)$,
- $\mathfrak{F}(P + Q) = \mathfrak{F}(P) \cup \mathfrak{F}(Q)$,
- $\mathfrak{F}(P \mid Q) = \mathfrak{F}(P) \cup \mathfrak{F}(Q)$,
- $\mathfrak{F}((\nu x)\ P) = \mathfrak{F}(P) \setminus \{x\}$,
- $\mathfrak{F}(!P) = \mathfrak{F}(P)$,

for any $a, x \in N$ and any $P, Q \in \mathbb{L}_\pi$.

Definition 22 (π-congruence [2])**.** Two π-processes $P, Q \in \mathbb{L}_\pi$ are structurally congruent, written $P \equiv Q$, if they fulfill the predicate $\equiv\colon \mathbb{L}_\pi \times \mathbb{L}_\pi \to \mathbb{B}$. We define inductively:

(α-conversion) $P \equiv Q$ if both only differ by the choice of bound names,

(choice rules) $P+Q \equiv Q+P$, and $(P+Q)+R \equiv P+(Q+R)$, and $P \equiv P+P$

(concurrency rules) $P \mid Q \equiv Q \mid P$, and $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$, and $P \equiv P \mid 0$,

(scoping rules) $(\nu x)\ (\nu y)\ P \equiv (\nu y)\ (\nu x)\ P$, and $(\nu x)\ (P \mid Q) \equiv P \mid ((\nu x)\ Q)$ if $x \notin \mathfrak{F}(P)$, and $(\nu x)\ 0 \equiv 0$,

(replication rules) $!P \equiv P \mid !P$,

for any names $x, y \in N$ and processes $P, Q, R \in \mathbb{L}_\pi$.

Definition 23 (π-substitution)**.** For a π-process P we write $P[y := z]$ for the π-process where every free occurrence of name y is replaced by name z. Formally, we define:

- $0[y := z] = 0$,
- $(\tau.P)[y := z] = \tau.(P[y := z])$,
- $(a(x).P)[y := z] = z(x).P$ for $a = y$ and $x = y$,
- $(a(x).P)[y := z] = a(x).P$ for $a \neq y$ and $x = y$,
- $(a(x).P)[y := z] = z(x).(P[y := z])$ for $a = y$ and $x \neq y$,
- $(a(x).P)[y := z] = a(x).(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{z}\langle z\rangle.(P[y := z])$ for $a = y$ and $x = y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{a}\langle z\rangle.(P[y := z])$ for $a \neq y$ and $x = y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{z}\langle x\rangle.(P[y := z])$ for $a = y$ and $x \neq y$,
- $(\overline{a}\langle x\rangle.P)[y := z] = \overline{a}\langle x\rangle.(P[y := z])$ for $a \neq y$ and $x \neq y$,
- $(P + Q)[y := z] = (P[y := z]) + (Q[y := z])$,
- $(P \mid Q)[y := z] = (P[y := z]) \mid (Q[y := z])$,
- $((\nu x)\ P)[y := z] = ((\nu x)\ P)$ for $x = y$,
- $((\nu x)\ P)[y := z] = (\nu x)\ (P[y := z])$ for $x \neq y$,
- $(!P)[y := z] = !(P[y := z])$,

for any names $a, x, y, z \in N$ and processes $P, Q \in \mathbb{L}_\pi$.

Definition 24 (π-evaluation)**.** An evaluation of a π-process P is a sequence of π-processes $P_0 \rightarrowtail ... \rightarrowtail P_n$ where $P_0 = P$ and P_{i+1} is generated from P_i via the application of an evaluation rule $\rightarrowtail: \mathbb{L}_\pi \rightarrow \mathbb{L}_\pi$ to any sub-term of P_i. We define the following evaluation rules:

(reaction) $(a(x).P + P') \mid (\overline{a}\langle y\rangle.Q + Q') \rightarrowtail_{\text{REACT}} (P[x := y]) \mid Q$,

(τ transition) $\tau.P + P' \rightarrowtail_{\text{TAU}} P$,

(parallel execution) $P \mid R \rightarrowtail_{\text{PAR}} Q \mid R$ if it holds that $P \rightarrowtail Q$,

(restricted execution) $(\nu x)\, P \rightarrowtail_{\text{RES}} (\nu x)\, Q \mid R$ if it holds that $P \rightarrowtail Q$,

(structural congruence) $P' \rightarrowtail_{\text{STRUCT}} Q'$ if it holds that $P \rightarrowtail Q$ and $P \equiv P'$ and $Q \equiv Q'$,

for any names $a, x, y \in N$ and processes $P, P', Q, Q' \in \mathbb{L}_\pi$ where $\equiv: \mathbb{L}_\pi \times \mathbb{L}_\pi \rightarrow \mathbb{B}$ is the predicate for structural congruence.

Example 3 (vacuum world in π-calculus)**.** We define:

$$\begin{aligned} \mathit{Robot} = \mathit{is\_clean}(\mathit{True}).(&\mathit{position}(\mathtt{A}).\overline{\mathit{move\_to\_right\_room}}.\mathit{Robot} \\ &+ \mathit{position}(\mathtt{B}).\overline{\mathit{move\_to\_left\_room}}.\mathit{Robot}) \\ &+ \mathit{is\_clean}(\mathit{False}).\mathit{position}(\_).\overline{\mathit{clean}}.\mathit{Robot} \end{aligned}$$

$$\begin{aligned} &\mathit{Environment}(\mathit{robot\_position}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean}) \ = \\ &\quad \overline{\mathit{position}}\langle \mathit{robot\_position}\rangle. \\ &\qquad \mathit{Environment}(\mathit{robot\_position}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean}) \\ &\quad + \overline{\mathit{is\_clean}}\langle \mathtt{check}(\mathit{robot\_position}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean})\rangle. \\ &\qquad \mathit{Environment}(\mathit{robot\_position}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean}) \\ &\quad + \mathit{move\_to\_left\_room}. \\ &\qquad \mathit{Environment}(\mathtt{A}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean}) \\ &\quad + \mathit{move\_to\_right\_room}. \\ &\qquad \mathit{Environment}(\mathtt{B}, \mathit{left\_room\_clean}, \mathit{right\_room\_clean}) \\ &\quad + \mathit{clean}. \\ &\qquad \mathit{Environment}(\\ &\qquad\quad \mathit{robot\_position}, \\ &\qquad\quad \mathtt{check}(\mathit{robot\_position}, \mathit{True}, \mathit{left\_room\_clean}), \\ &\qquad\quad \mathtt{check}(\mathit{robot\_position}, \mathit{right\_room\_clean}, \mathit{True}) \\ &\qquad) \\ &\quad + \tau.\mathit{Environment}(\mathit{robot\_position}, \mathit{False}, \mathit{right\_room\_clean}) \\ &\quad + \tau.\mathit{Environment}(\mathit{robot\_position}, \mathit{left\_room\_clean}, \mathit{False}) \end{aligned}$$

where $\mathtt{check}(\mathit{robot\_position}, \mathit{robot\_left\_value}, \mathit{robot\_right\_value}) =$

$$\begin{cases} \mathit{robot\_left\_value} & \text{if } \mathit{robot\_position} = \mathtt{A}, \\ \mathit{robot\_right\_value} & \text{otherwise.} \end{cases}$$

The vacuum world example can then be simulated via the π-process $\mathit{VacuumWorld} = \mathit{Robot} \mid \mathit{Environment}(\mathtt{A}, \mathit{False}, \mathit{False})$.

References

[1] No free lunch theorems. `http://www.no-free-lunch.org`. Accessed 2022-05-10.

[2] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[3] Hans Peters. *Game Theory: A Multi-Leveled Approach*. Springer, 2015.

[4] Wikipedia. Kolmogorov-Arnold representation theorem. `https://en.wikipedia.org/wiki/KolmogorovArnold_representation_theorem`. Accessed 2022-05-10.

[5] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.