L2 - The basic concepts and vocabulary of reinforcement learning

Andrzej Szwabe

Laboratory of Intelligent Decision Support Systems
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznań, Poland
email: Andrzej.Szwabe@put.poznan.pl

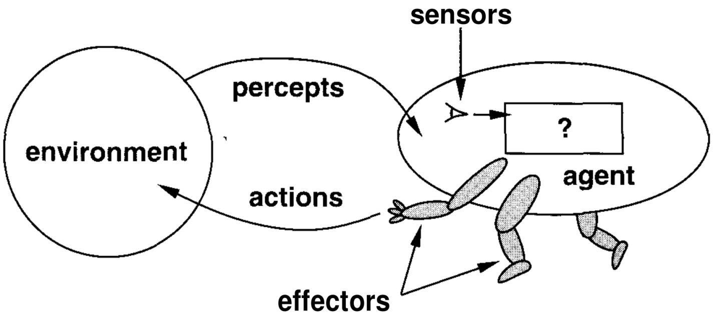
October 2021

Outline

- The concept of a rational agent, performance measure, percept sequence, mapping from percept sequences to actions
- The conventional hierarchy of intelligent agents considered in research on AI
- The elements of the general model of sequential decision making problems
  - time step (discrete time representation)
  - agent's action in the environment based on its policy and the current observation
  - environment's reaction: a reward and the next observation provided to the agent
  - reinforcement learning agent actions in an environment and the cumulative (time-discounted) reward maximization objective
- Technical-level reinforcement learning environment concepts: state, step, episode, reward, observation, evaluation metrics
- Technical-level reinforcement learning agent concepts: policy, collect policy, replay buffer, learner

The concept of a rational agent

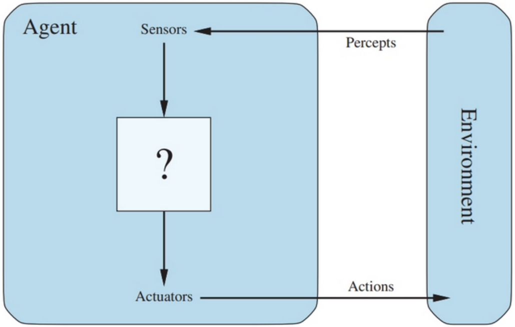
- 'An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.' [AIMA][1]]
- Examples: animal (including human), robot (with some sensors used for control), thermostat, software system (interacting with its environment)
- A view on a generic agent [AIMA1995]:



[1] [AIMA] Russell, Stuart, and Peter Norvig. Artificial Intelligence: A Modern Approach, eBook, Global Edition. Available from: VitalSource Bookshelf, (4th Edition). Pearson International Content, 2021 (available via https://bookshelf.vitalsource.com//explore via library.put.poznan.pl)

Intelligent agent is about interaction with her/his/its environment.

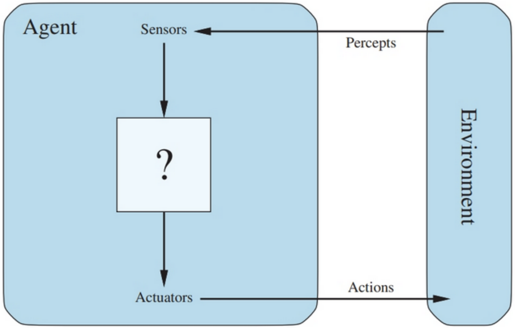
- Another view on a generic agent [AIMA2021]:



- The agent function (aka the agent's policy) maps from percept 'histories' (i.e. sequences) to actions:

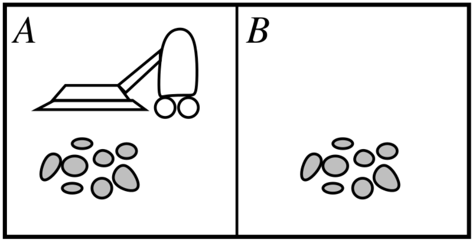
$$f : \mathcal{P}^* \rightarrow \mathcal{A} \qquad (1)$$

Intelligent agent sensors, percept and percept sequences



- Percept is the content an agent's sensors.
- An agent's percept sequence is the complete history of everything the agent has ever perceived.
- In general, an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn't perceived.

A toy example: a vacuum-cleaner world [AIMA] (1 of 2)

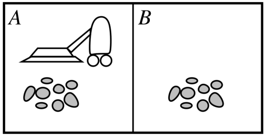


Percepts: location and contents, e.g., $[A, Dirty]$

Actions: $Left$, $Right$, $Suck$, $NoOp$

- There are just two locations in the vacuum-cleaner world.
- Each location can be clean or dirty.
- The agent can move left or right and can clean the square that it occupies.

A toy example: a vacuum-cleaner world [AIMA] (1 of 2)



Percepts: location and contents, e.g., $[A, Dirty]$

Actions: $Left$, $Right$, $Suck$, $NoOp$

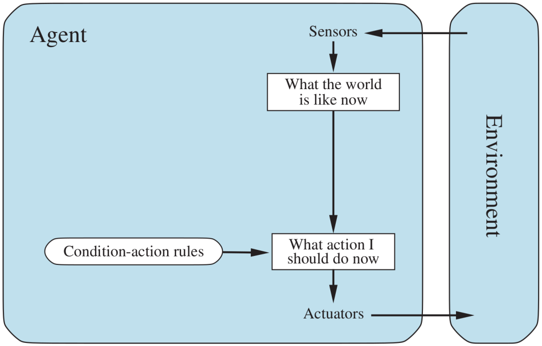
Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean]$, $[A, Clean]$	$Right$
$[A, Clean]$, $[A, Dirty]$	$Suck$
⋮	⋮

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
    if $status = Dirty$ then return $Suck$
    else if $location = A$ then return $Right$
    else if $location = B$ then return $Left$
```

Rationality of an intelligent agent

- What is rational at any given time depends on four things ('PEAS', (Performance measure, Environment, Actuators, Sensors)):
  - The performance measure that defines the criterion of success.
  - The agent's prior knowledge of the environment
  - The actions that the agent can perform
  - The agent's percept sequence to date
- Definition of a rational agent:
  - *For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

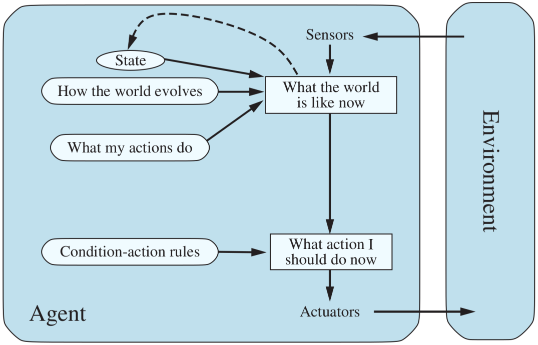
Simple reflex agent



- Simple reflex agent works by finding a rule whose condition matches the current situation (as defined by the percept) and then doing the action associated with that rule. [AIMA]

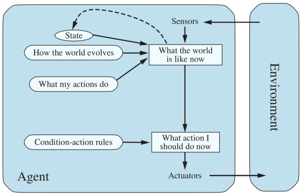
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Model-based reflex agent (1 of 2)



- Model-based reflex agent keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent. [AIMA]

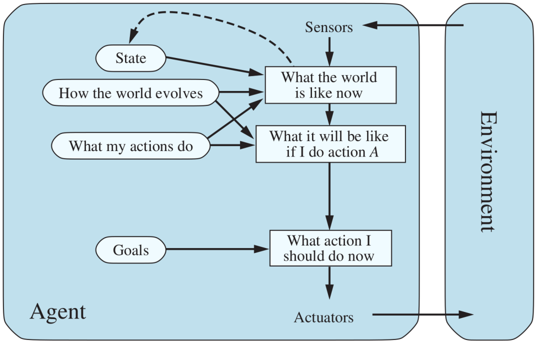
Model-based reflex agent (2 of 2)



- Model-based reflex agent keeps track of the state of the world using its model. [AIMA]

```
function MODEL-BASED-REFLEX-AGENT($percept$) returns an action
  persistent: $state$, the agent's current conception of the world state
              $transition\_model$, a description of how the next state depends on
                  the current state and action
              $sensor\_model$, a description of how the current world state is reflected
                  in the agent's percepts
              $rules$, a set of condition–action rules
              $action$, the most recent action, initially none

  $state \leftarrow$ UPDATE-STATE($state, action, percept, transition\_model, sensor\_model$)
  $rule \leftarrow$ RULE-MATCH($state, rules$)
  $action \leftarrow rule$.ACTION
  return $action$
```

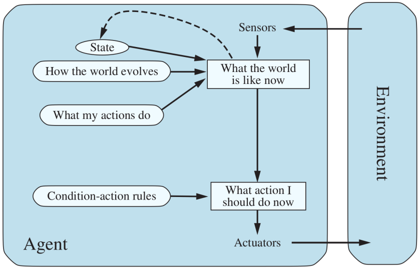
Model- and goal-based agent (1 of 2)



Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

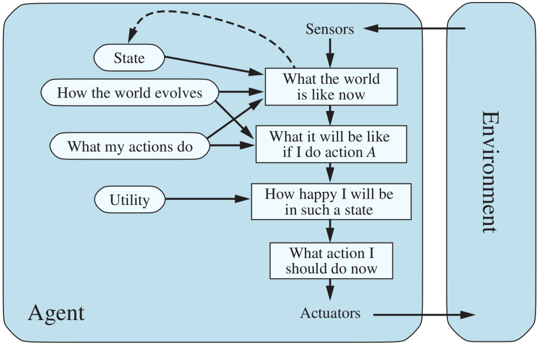
- Model- and goal-based agent keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals. [AIMA2021]

Model- and goal-based agent (2 of 2)



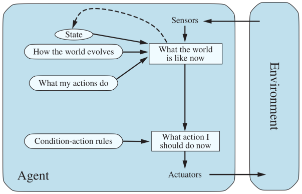
- Model- and goal-based agent keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals. [AIMA]
  - Decision making of this kind is fundamentally different from the condition–action rules in that it involves consideration of the future.
  - A goal-based agent is more flexible/adaptable because the knowledge that supports its decisions is represented explicitly and can be modified.
  - Search and planning are some of the subfields of AI devoted to finding action sequences that achieve the agent's goals.

Model- and utility-based agent (1 of 2)


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action *A*
How happy I will be in such a state
What action I should do now
Environment
Agent
Actuators

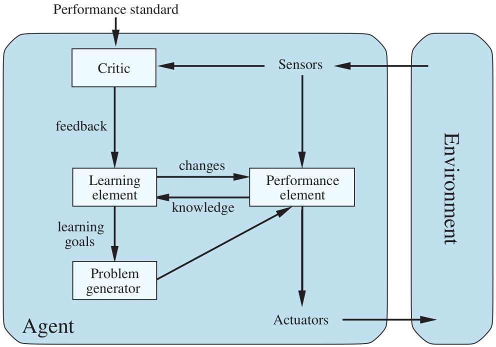
- Model- and utility-based agent uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome. [AIMA2021]

Model- and utility-based agent (2 of 2)



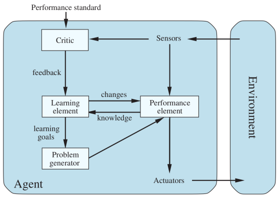
- Model- and utility-based agent uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome. [AIMA2021]
  - Utility function is in general at least partly unknown.
  - A special type of utility-based agent - model-free agent - can learn what action is best in a particular situation without learning how that action changes the environment.
  - Search and planning are the subfields of AI devoted to finding action sequences that achieve the agent's goals.
    - Active learning and Bayesian optimization can be seen as subfields of AI in which 'search is planned' and the utility function is to be effectively represented, predicted and 'discovered' by the agent.

General learning agent (1 of 2)



- In a general learning agent the “performance element” box represents what is the whole agent program of a non-learning agent. The “learning element” modifies the “performance element” to improve its performance. [AIMA2021]
- Any type of agent (model-based, goal-based, utility-based, etc.) can be built as a learning agent (or not).

General learning agent (2 of 2)



- A learning agent can be divided into four conceptual components: Critic, Learning element, Performance element and Problem generator.
- The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.
- The performance standard distinguishes part of the incoming percept as a reward Reward (or penalty) that provides direct feedback on the quality of the agent's behavior.
- The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.
- The problem generator is responsible for suggesting actions that will lead to new and informative experiences.

The Hitchhiker's Guide to Reinforcement Learning ;-)

- Reinforcement learning agent aims at improving its polity through performing actions in an environment in a way that maximizes the cumulative reward. In slightly other words: the reward maximization is the objective of improving the policy in a way maximizing the sum of rewards.
- Policy is a mapping from every possible state observation from the environment to the best move (i.e. agent's action in the environment) in that state: formally, it is a mapping from states to probabilities of selecting each possible action.
- The value function of a state s under a policy is the expected return when starting in s and following the policy thereafter.
- In some scenarios reward is assumed to be observable at each observation, but in other may only be observable at the end of s sequence of time steps called an episode (e.g. at the end of some game).
- The value of the terminal state, if any, is always zero.

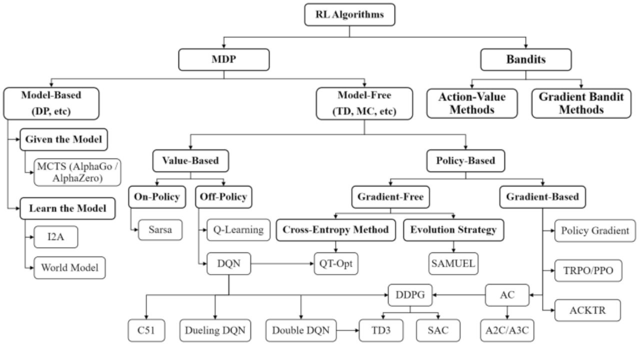
Episodic environment vs episodic decision process

- Episodic environment
    - *In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic. (...) In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.* [AIMA]
- Episodic decision process
    - *Playing blackjack is naturally formulated as an episodic finite MDP. Each game of blackjack is an episode. Rewards of +1, 1, and 0 are given for winning, losing, and drawing, respectively. All rewards within a game are zero, and we do not discount (= 1); therefore these terminal rewards are also the returns.* [SuttonAndBarto[2]]

[2] [SuttonAndBarto] Richard S. Sutton and Andrew G. Barto, „Reinforcement Learning: An Introduction", 2018 (online: http://incompleteideas.net/book/the-book.html)

Model-based and model-free reinforcement learning [AIMA]

- The distinction between model-based and model-free reinforcement learning and the distinction between value-based model-free reinforcement learning and policy-based model-free reinforcement learning are the key elements of the taxonomy of RL algorithms.



Model-based reinforcement learning [AIMA]

- Model-based reinforcement learning
  - The agent uses a transition model of the environment to help interpret the reward signals and to make decisions about how to act.
  - The model may be initially unknown, in which case the agent learns the model from observing the effects of its actions, or it may already be known — for example, a chess program may know the rules of chess even if it does not know how to choose good moves.
  - In partially observable environments, the transition model is also useful for state estimation.
  - Model-based reinforcement learning systems often learn a utility function $U(s)$, defined in terms of the sum of rewards from state s onward.
    - Terminological remark: In the RL literature, which draws more on operations research than economics, utility functions are often called value functions and denoted $V(s)$.

Model-free reinforcement learning [AIMA]

- Model-free reinforcement learning
  - Model-free reinforcement learning: In these approaches the agent neither knows nor learns a transition model for the environment. Instead, it learns a more direct representation of how to behave.
  - There are two general types of model-free reinforcement learning:
    - Action-utility learning: Q-learning: the most common form of action-utility learning in case of which the agent learns Q-learning a Q-function, or quality-function, $Q(s, a)$, denoting the sum of rewards from state Q-function s onward if action a is taken. Given a Q-function, the agent can choose what to do in s by finding the action with the highest Q-value.
    - Policy search: The agent learns a policy $\pi(s)$ that maps directly from states to Policy search actions, i.e. it is a reflex agent.

Very briefly about active and passive RL [AIMA]

- Active reinforcement learning case
  - Apart from learning the utilities of states (or of state–action pairs) and possibly also learning a model of the environment, the agent must also figure out what to do.
- Passive reinforcement learning case/mode
  - The agent's policy is fixed (it is sometimes called 'collect policy') and the only task is to learn the utilities of states (or of state–action pairs); this could also involve learning a model of the environment.

Briefly about active reinforcement learning [AIMA]

- In practice, no agent can act effectively without learning, so one of the main tasks of active reinforcement learning agent is to learn the utilities of states (or of state–action pairs).
    - Optionally it can also involve learning a model of the environment (in case of model based RL).
- However, in case of active reinforcement learning, apart from learning the utilities of states (or of state–action pairs) and possibly also learning a model of the environment, the agent must also figure out what to do.
- The principal issue is exploration: an agent must experience of its environment in order to learn how to behave in it.
    - In general, exploration has no sense in case of passive learning - whether it is ’traditional supervised learning or passive RL.

Briefly about passive reinforcement learning [AIMA]

- In case/mode of passive reinforcement learning, the agent's policy is fixed (it is sometimes called 'collect policy') and the task is, again, to learn the utilities of states (or of state–action pairs); this could also involve learning a model of the environment.
    - Speaking more precisely, a passive RL agent is trying to learn the utility function $U_\pi(s)$ — the expected total discounted reward if policy π is executed beginning in state s.
    - Each transition is annotated with both the action taken and the reward received at the next state. The objective is to use the information about rewards to learn the expected utility $U_\pi(s)$ associated with each nonterminal state s.
    - The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed.
- A passive learning agent does not know the transition model $P(s'|s, a)$, which specifies the probability of reaching state s' from state s after doing action a; nor does it know the reward function $R(s, a, s')$, which specifies the reward for each transition.

Reward discounting and myopic RL agent [AIMA]

- Discount factor γ is a number between 0 and 1 describing the preference of an agent for current rewards over future rewards.
  - When γ is close to 0, rewards in the distant future are viewed as insignificant and we can all the agent as being myopic.
  - When γ is close to 1, an agent is more willing to wait for long-term rewards.
  - When γ is exactly 1, discounted rewards reduce to the special case of purely additive rewards.
- To learn the expected utility $U^{\pi}(s)$ associated with each nonterminal state s, reward time-discounting is used.
  - $R(S_t, \pi(S_t), S_{t+1})$ is the reward received when action $\pi(S_t)$ is taken in state S_t and reaches state S_{t+1}.
  - S_t is a random variable denoting the state reached at time t when executing policy π, starting from state $S_0 = s$.
  - The utility is defined to be the expected sum of discounted rewards obtained if policy π is followed:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\right]$$

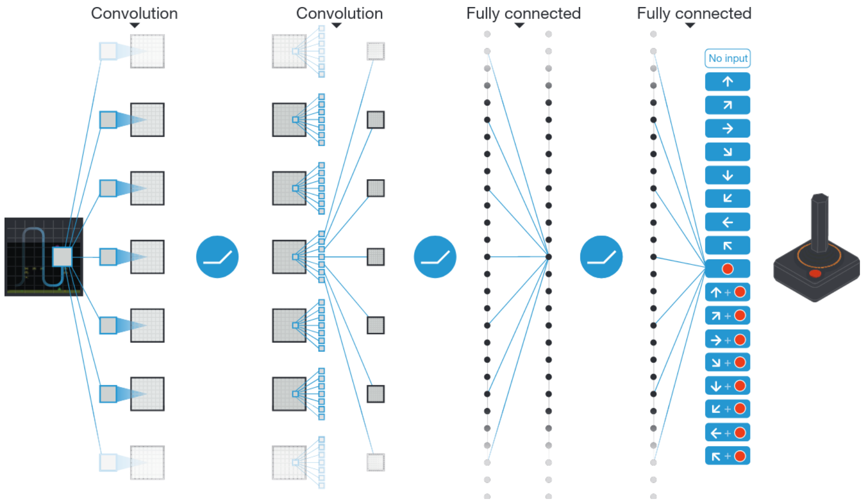
Exploration-exploitation dilemma from the policy learning perspective

- Policy is the RL agent's state-action mapping.
- All learning control methods (RL, AL, BO) face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions).
- In case of RL, an agent learns about the optimal policy while behaving according to an exploratory, i.e. sub-optimal policy.

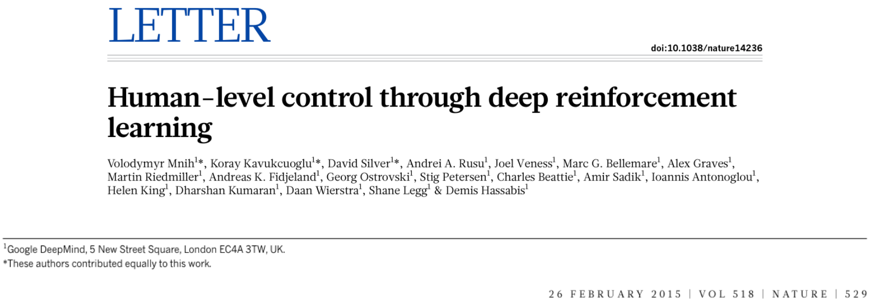
On-policy and off-policy reinforcement learning

- Policy is the RL agent's state-action mapping.
- On-policy and off-policy learning as two alternative ways of handling the conflict between exploitation and exploration in RL.
- The on-policy approach is an evident 'learning compromise': it learns action values not for the optimal policy, but for a near-optimal policy that still explores.
- In case of off-policy approach two policies are used:
  - the target policy (e.g. represented as target network) - the one that is learned about and that becomes the optimal policy
  - the behavior policy (aka collect policy) (e.g. represented as Q-network) - the one that is more exploratory and is used to generate behavior.
- In case of off-policy approach to RL learning is from data “off” the target policy.

Too early and too quick view on Deep Q Learning ;-)



The famous 'Deep Q Learning paper' [Mnih2015][3]



LETTER

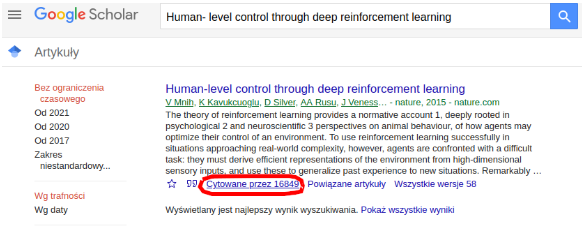
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK.
*These authors contributed equally to this work.

26 FEBRUARY 2015 | VOL 518 | NATURE | 529



Human-level control through deep reinforcement learning
V Mnih, K Kavukcuoglu, D Silver, AA Rusu, J Veness... - nature, 2015 - nature.com
The theory of reinforcement learning provides a normative account 1, deeply rooted in psychological 2 and neuroscientific 3 perspectives on animal behaviour, of how agents may optimize their control of an environment. To use reinforcement learning successfully in situations approaching real-world complexity, however, agents are confronted with a difficult task: they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Remarkably ...
☆ 99 Cytowane przez 16849 Powiązane artykuły Wszystkie wersje 58

Wyświetlany jest najlepszy wynik wyszukiwania. Pokaż wszystkie wyniki

[3][Mnih2015] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533

Too early and too quick view on Deep Q Network ;-)

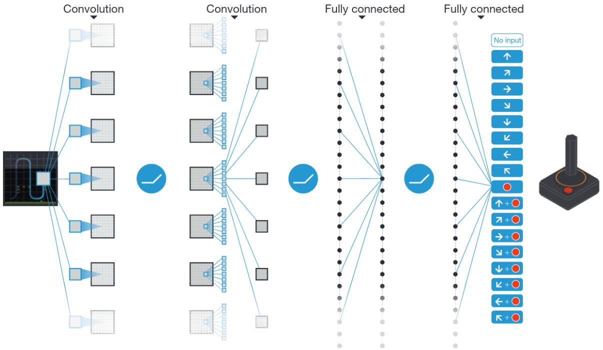


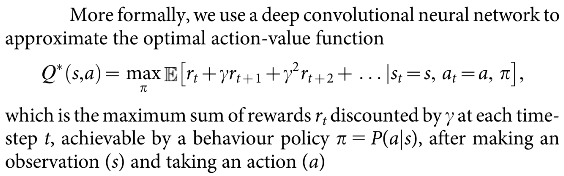
Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an 84 × 84 × 4 image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a)

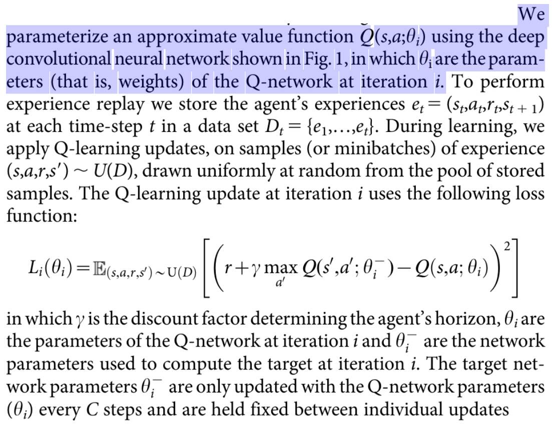
Quick view on DQN's action-value function



More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \middle| s_t = s,\, a_t = a,\, \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a)

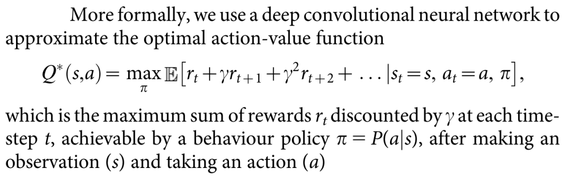


We parameterize an approximate value function $Q(s,a;\theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences $e_t = (s_t,a_t,r_t,s_{t+1})$ at each time-step t in a data set $D_t = \{e_1,\ldots,e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) every C steps and are held fixed between individual updates

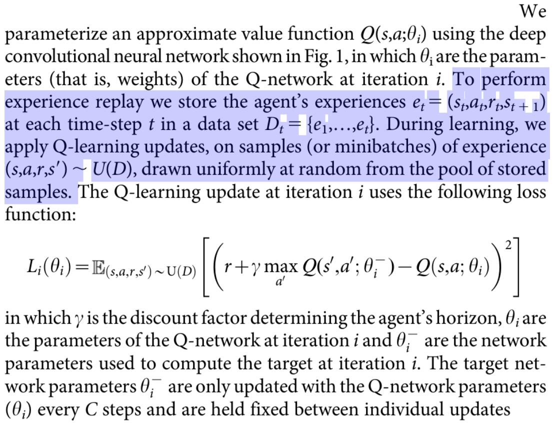
Quick view on DQN's experience replay



More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \mid s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a)



We parameterize an approximate value function $Q(s,a;\theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t in a data set $D_t = \{e_1, \ldots, e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) every C steps and are held fixed between individual updates

Quick view on DQN's target network

More formally, we use a deep convolutional neural network to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\ a_t = a,\ \pi\right],$$

which is the maximum sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a)

We parameterize an approximate value function $Q(s,a;\theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step t in a data set $D_t = \{e_1, \ldots, e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

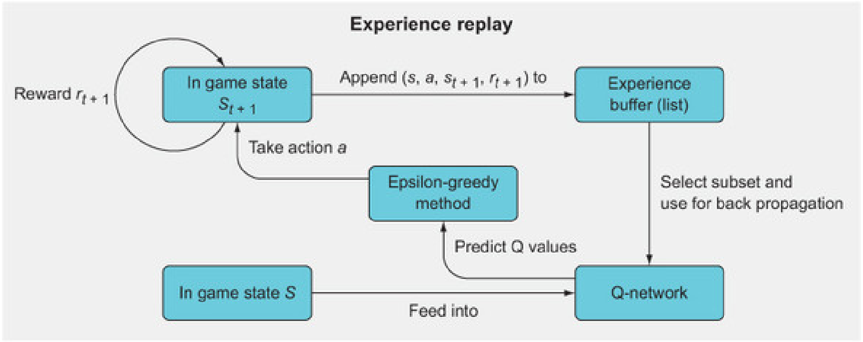
in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) every C steps and are held fixed between individual updates

Experience storage in practice: experience replay method and Deep Q-learning

- Experience replay method stores the agent's experience at each time step in a replay memory that is accessed to perform the NN weight updates.
- Experience replay method requires an off-policy algorithm. (Why?)
- It has been popularized by the DQN (Deep Q-Network) algorithm.
- In DQN experience replay works as follows:
  - The game emulator executes action A_t in a state (represented by the image stack) S_t, and returns reward R_{t+1} and image stack S_{t+1}.
  - The tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ is added to the replay memory accumulating experiences over many plays of the same Atari game.
  - At each time step multiple Q-learning updates (a mini-batch) are performed based on experiences sampled uniformly at random from the replay memory. Instead of S_{t+1} becoming the new S_t for the next update as it would in the usual form of Q-learning, a new unconnected experience is drawn from the replay memory to supply data for the next update.
- Because Q-learning is an off-policy algorithm, it does not need to be applied along connected trajectories.

Another view on experience replay [Zai2020][4]

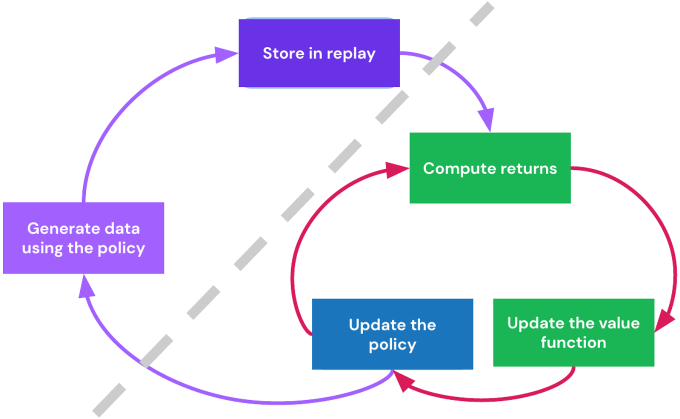
- Experience replay as a method for mitigating a major problem with online training algorithms: catastrophic forgetting.
- The idea: to employ mini-batching by storing past experiences and then using a random subset of these experiences to update the Q-network, rather than using just the single most recent experience.



[4][Mnih2015] Zai, Alexander, and Brandon Brown. Deep reinforcement learning in action. Manning Publications, 2020

Yet another view on experience replay [Behbahani2020][5]

- In some sense there are two loops: data generation and (network's) parameter optimization.



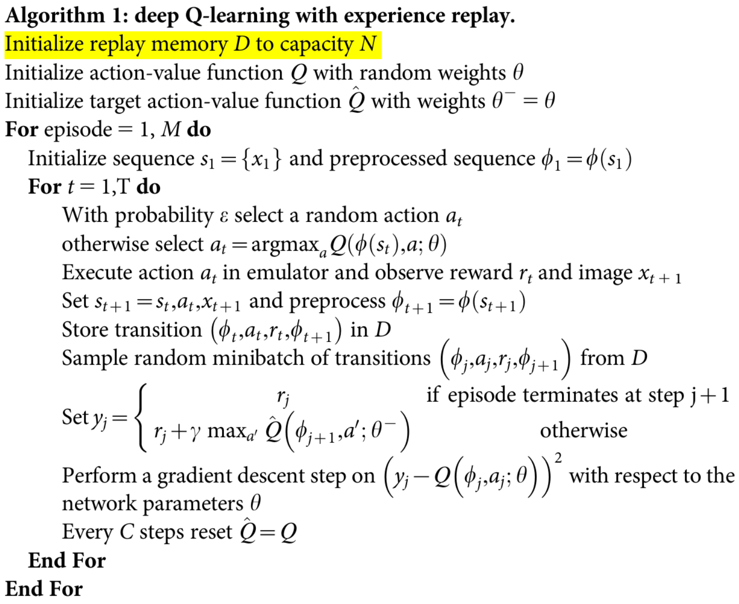
[5] [Behbahani2020] Feryal Behbahani, Matt Hoffman and Bobak Shahriari, Slides and Colab for the Reinforcement Learning lecture and tutorial at MLSS2020, https://github.com/Feryal/rl_mlss_2020

Deep Q-learning algorithm overview

Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  **For** $t = 1$,T **do**
     With probability $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
     Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
     Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
     Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
     Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from $D$
     Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
     Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

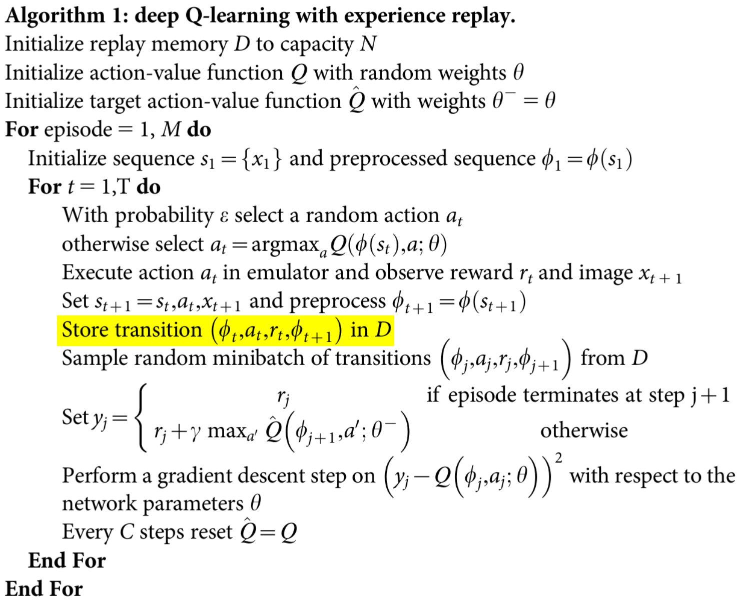
Experience replay in Deep Q-learning algorithm (1/3)



Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T do
     With probability $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
     Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
     Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
     Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
     Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
     Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
     Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

Experience replay in Deep Q-learning algorithm (2/3)



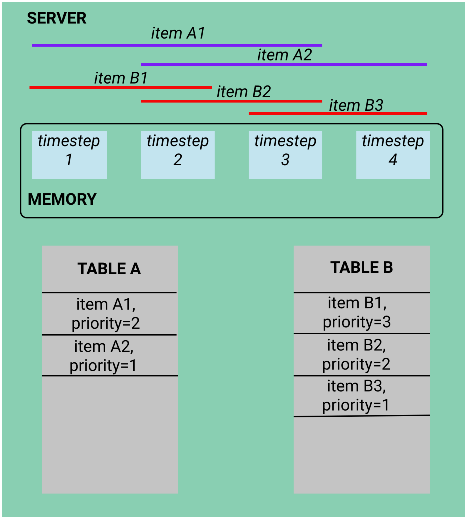
```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
**For** episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  **For** $t = 1$,T **do**
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

Experience replay in Deep Q-learning algorithm (3/3)

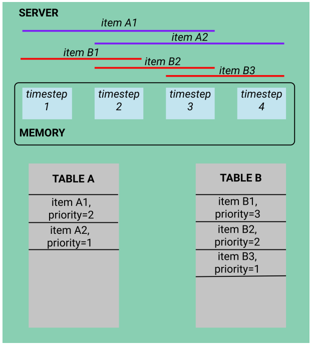
Algorithm 1: deep Q-learning with experience replay.

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
     With probability $\varepsilon$ select a random action $a_t$
     otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
     Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
     Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
     Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
     Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
     Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
     Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
     network parameters $\theta$
     Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

DeepMind Reverb - a popular implementation of experience replay buffer with prioritization capabilities



DeepMind Reverb items [ReverbDemo][6]



- Items are potentially overlapping sequences of varying length (aka trajectories).
- Items may be inserted into multiple tables.
- Items may have different (i.e. differing) and dynamically-calculated priority values.

[6] [ReverbDemo] Reverb demo, Inserting Sequences of Varying Length into Multiple Priority Tables, https://github.com/deepmind/reverb/blob/master/examples/demo.ipynb

Summary

- Intelligent agent concept
  - Intelligent agent sensors, percept and percept sequences
  - A toy example: a vacuum-cleaner world
  - Rationality of an intelligent agent
- Classes of intelligent agents: simple reflex, model-based reflex, model- and goal-based, model- and utility-based, general learning agent
- Quick introduction to ('vocablary' of) reinforcement learning
  - Episodic environment vs episodic decision process
  - Model-based and model-free reinforcement learning
  - Taxonomy of RL algorithms
  - Active and passive RL
  - Reward discounting and myopic RL agent
  - Exploration-exploitation dilemma from the policy learning perspective
  - On-policy and off-policy reinforcement learning
- Quick view on Deep Q Learning: DQN's action-value function, DQN's experience replay, DQN's target network
- Experience replay method
  - Experience replay in Deep Q-learning algorithm
  - DeepMind Reverb - a popular implementation of experience replay buffer