Reinforcement Learning

A Very Brief Introduction and With a Focus on Robotics

Alex Mitrevski
PhD Student in the Autonomous Systems Group
Computer Science Department

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Resources

This slide set is primarily compiled from the following sources:

L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237-285, 1996.

M. A. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1-142, 2011.

J. Kober and J. Peters, "Reinforcement Learning in Robotics: A Survey," *Reinforcement Learning: State-of-the-Art*, pp. 579-610, 2013.

A. F. Abdelrahman, "Incorporating Contextual Knowledge Into Human-Robot Collaborative Task Execution," Hochschule Bonn-Rhein-Sieg, 2020, http://dx.doi.org/10.18418/978-3-96043-080-3.

S. Levine, "Introduction to Reinforcement Learning," CS 285 (Deep Reinforcement Learning), http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-4.pdf

OpenAI, Spinning Up in Deep RL, https://spinningup.openai.com/en/latest/index.html

Reminder: Important Definitions

Expectation of a random variable

$$E[X] = \int xp(x)dx$$

First-order Markov assumption

$$P(s_{t+1}|s_{1:t-1}, s_t) = P(s_{t+1}|s_t)$$

Gradient of a log function

$$\nabla \log (f(x)) = \frac{\nabla f(x)}{f(x)}$$

→ Preliminaries

Classical model-based learning algorithms

Model-free learning

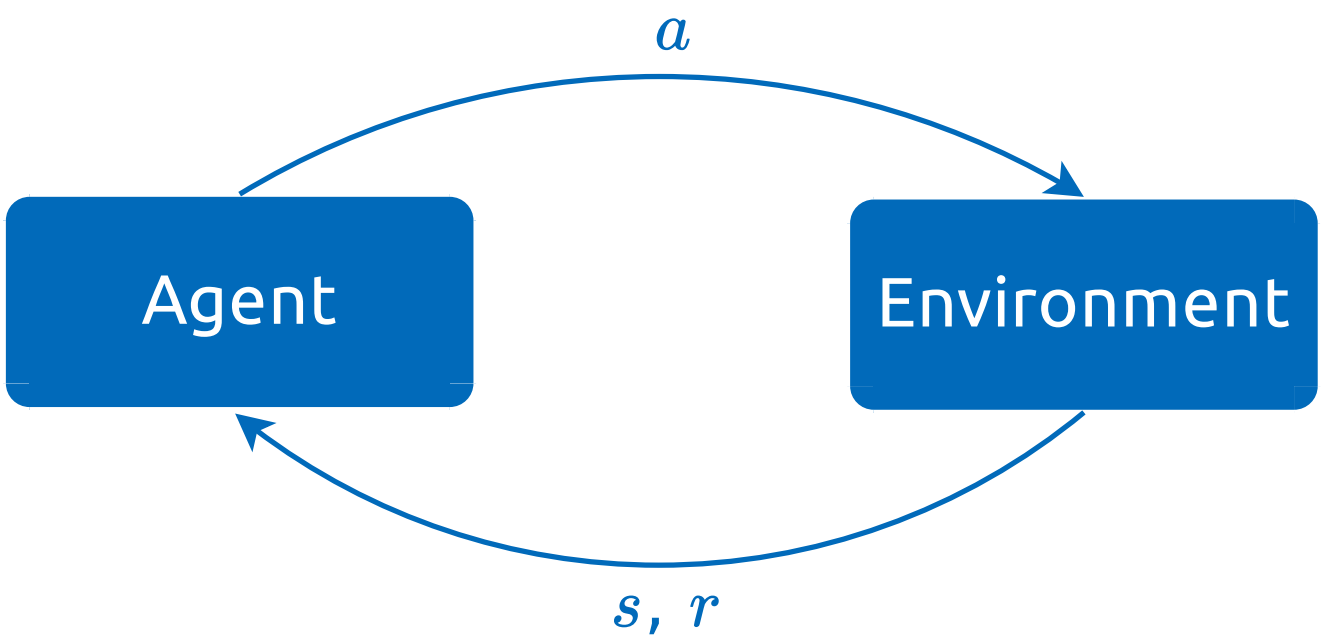
Deep reinforcement learning

Reinforcement learning outlook

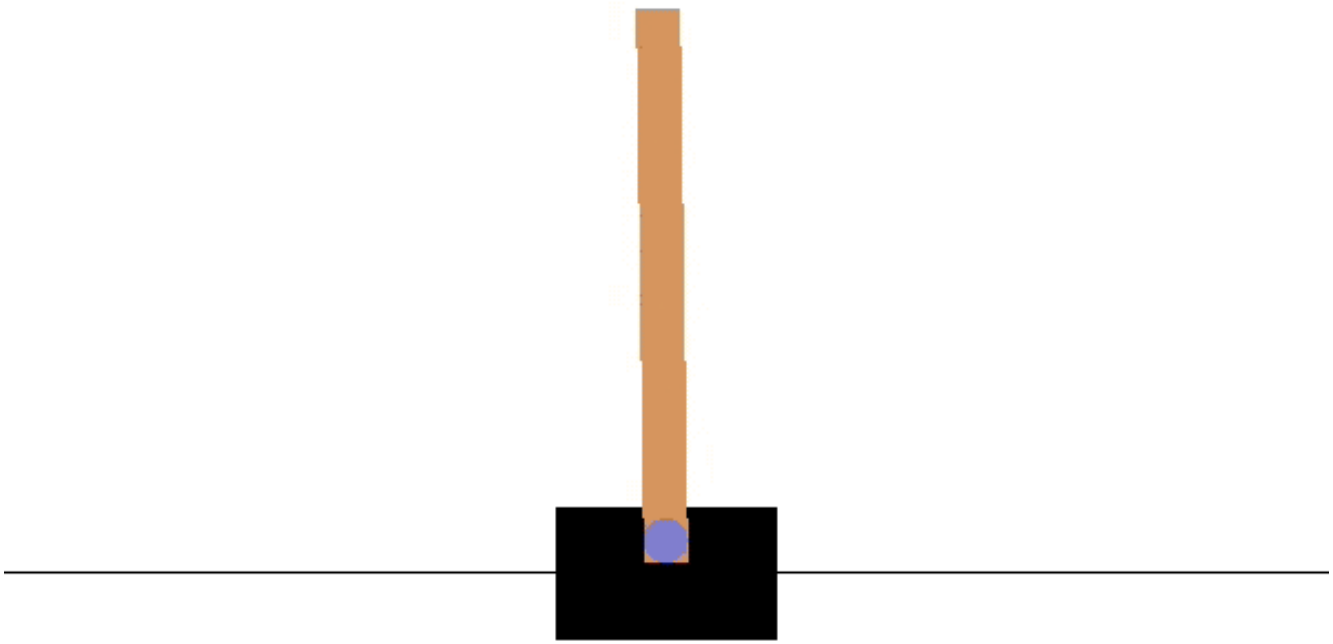
What is Reinforcement Learning (RL)?

Reinforcement learning (RL) is one of the three major learning paradigms, along with supervised and unsupervised learning

In reinforcement learning, an agent learns by **interacting with the environment** and **obtaining a reward signal for its actions**



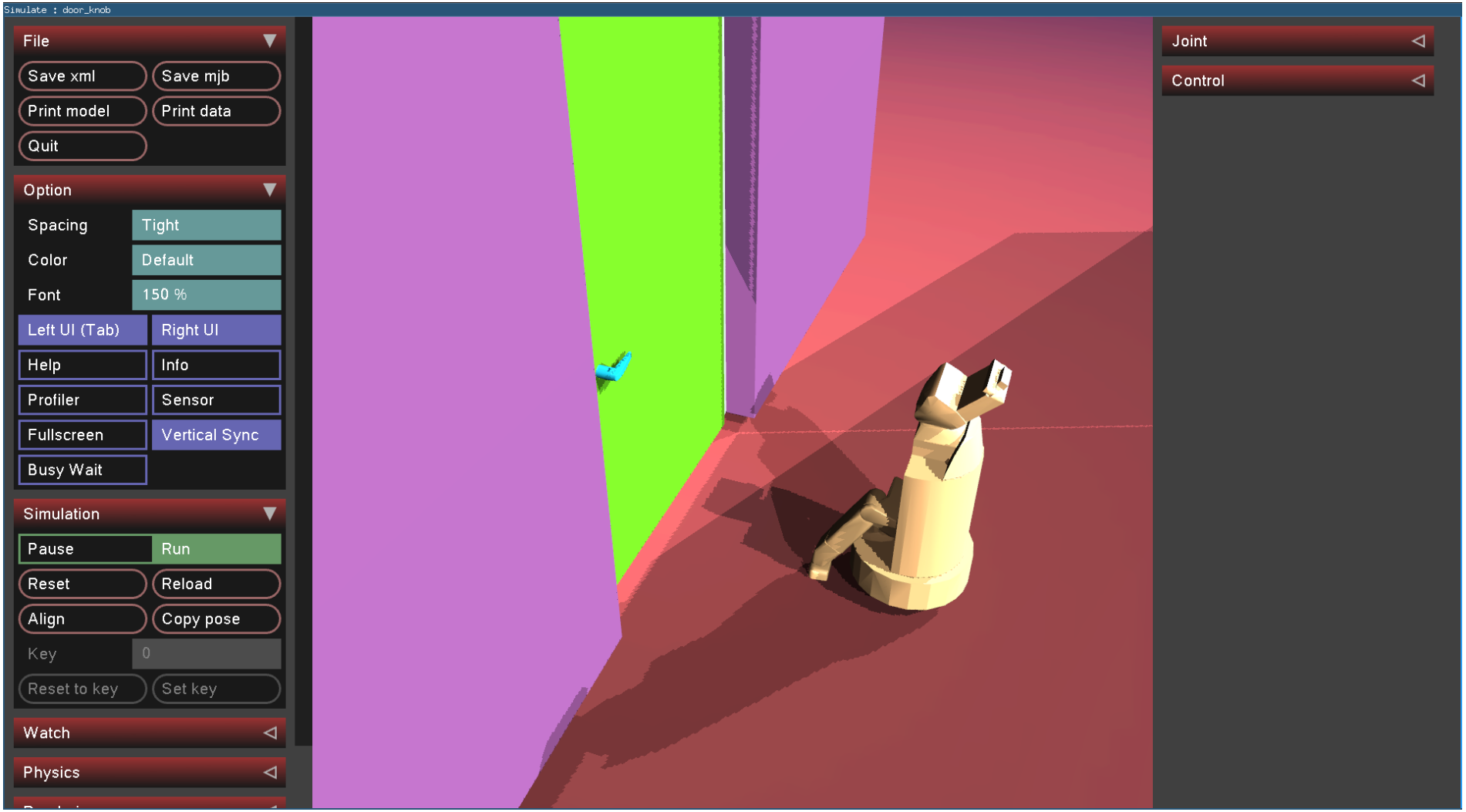
Teaser Problem: Cart Pole Balancing



Robotics Example: Context-Aware Object Hand-Over[1]

[1] A. F. Abdelrahman, "Incorporating Contextual Knowledge Into Human-Robot Collaborative Task Execution," Hochschule Bonn-Rhein-Sieg, 2019.

Robotics Example 2: Opening Doors[2]



[2] A. Narasimamurthy, "Manipulating Handles in Domestic Environments," Hochschule Bonn-Rhein-Sieg, 2019.

Reinforcement Learning vs. Other Learning Paradigms

Supervised learning

Given: inputs $\mathbf{x}$ and labels/outputs $\mathbf{y}$
Learn: a mapping $f(\mathbf{x}) = \mathbf{y}$

Unsupervised learning

Given: inputs $\mathbf{x}$
Learn: some structure about the data

Reinforcement learning

Given: a set of experiences (observed states, applied actions, and obtained rewards along the way)
Learn: how to behave so that the reward is maximised (i.e. an optimal mapping from observed states to actions)

The RL learning objective is the most difficult of the three because it is rather underspecified/more open-ended than the other two

Basic RL Formalisation

A reinforcement learning problem is typically modelled as a Markov Decision Process (MDP)

An MDP $\mathcal{M}$ is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where

- $\mathcal{S}$ is a set of **states**
- $\mathcal{A}$ is a set of **actions**
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a **state transition function**
  - in other words, T models the probability $P(s_{t+1}|s_t, a_t)$
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a **reward function**
  - the reward function evaluates the behaviour of the agent

The accumulation of rewards over multiple time steps is called the **return** R

In principle, the objective of an RL agent is rather simple to state: **maximise the expected return** $J = E\left[\sum_t r(a_t, s_t)\right]$

Robotics: Continuous States (and Often Actions)

The previous formulation assumes discrete state space $\mathcal{S}$ and action space $\mathcal{A}$

In robotics, we usually have a more complicated problem - a continuous state space (and sometimes also continuous action space)

Examples: force sensor measurements, joint angle/velocity measurements and commands, distance measurements...

We will occasionally come back to this problem during the lecture

Policies and Trajectories

A **policy** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is the model of the agent's behaviour

Very often (particularly in robotics), a policy is represented by parameters θ, in which case it is written as π_θ

A policy is used to define a **trajectory** (also called **episode** or **rollout**)

$$\tau = (s_0, a_0, s_1, ..., a_n, s_{n+1})$$

Given a policy π, the probability of a trajectory can be found to be

$$P_\pi(s_0, a_0, s_1, ..., a_n, s_{n+1}) = P(s_0) \prod_{i=0}^{n} P_\pi(a_i|s_i) P(s_{i+1}|s_i, a_i)$$

Deterministic vs. Stochastic Policies

Regarding how actions are selected from a policy, we can distinguish between deterministic and stochastic policies

In a **stochastic** policy, actions are selected by sampling from the distribution of actions given a state

$$a_t \sim \pi(a|s_t)$$

Example: choosing a tennis stroke given the observed motion of the incoming ball

A **deterministic** policy is a deterministic function of the state

$$a_t = \pi(s_t)$$

Example: grasping a door handle for opening

Finite and Infinite Horizon Tasks

In RL, we can distinguish between two types of tasks:

- a **finite horizon/episodic task** ends after a certain number of time steps or when a goal is reached (e.g. robot navigation from one place to another)
- an **infinite horizon task** potentially goes on indefinitely (e.g. cleaning a room)

In the finite horizon case, the return is given as

$$R(\tau) = \sum_{t=0}^{T} r(s_t, a_t)$$

The return in the infinite horizon case is discounted by a factor $\gamma \in (0, 1]$

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

RL Learning Objective

In general, we want to find a policy π^* that maximises the agent's expected return

General learning objective

$$\pi^* = \arg\max_{\pi} \underset{\tau \sim \pi}{E} \left[\sum_t r(s_t, a_t) \right] = \arg\max_{\pi} \int P(\tau|\pi) R(\tau) d\tau$$

If we are given a parameterised policy π_θ, the problem becomes that of finding a set of parameters θ^* that maximise the expected return

Learning objective given a parameterised policy

$$\theta^* = \arg\max_{\theta} \underset{\tau \sim \pi_\theta}{E} \left[\sum_t r(s_t, a_t) \right] = \arg\max_{\theta} \int P(\tau|\pi_\theta) R(\tau) d\tau$$

Model-Based vs. Model-Free RL

The transition function $\mathcal{T}$ and the reward function r are usually referred to as the **model** of the environment

- If $\mathcal{T}$ and r are known in advance, we have **model-based** RL
  - In this case, the agent can exploit the knowledge of the model to find an optimal policy
- The more likely case in which $\mathcal{T}$ and r are unknown is that of **model-free** RL
  - Here, the best the agent can do is learn using trial-and-error

A combination of the two paradigms is possible, where an agent tries to learn both the model and a policy

Exploration vs. Exploitation

During learning, an agent has to balance:

- **Exploitation**: Acting according to the best policy (so far)
- **Exploration**: Acting by trying out actions that may not be the most optimal under the current best policy

There is always a trade-off between exploitation and exploration:

- if the agent exploits too much too early, it risks converging to a sub-optimal policy
- the agent's policy should eventually converge however; too much exploration can prevent that from happening

On-Policy vs. Off-Policy RL

With respect to how actions are selected during learning, we distinguish between:

- **off-policy** learning: an agent samples actions from a policy different from the policy that is being learned
  - Example: the agent acts randomly during learning
- **on-policy** learning: an agent samples actions from the policy that it is trying to learn
  - Example: the agent greedily samples actions from its current best policy during learning

Off-policy learning encourages more exploration and is thus slower, but is likely to find better policies in general

Value Functions

Value functions are at the core of several RL algorithms

The **state value function** $V^{\pi}(s)$ gives the expected return given that the agent starts in state s and then acts according to its policy π

State value function

$$V^{\pi}(s) = \underset{\tau \sim \pi}{E}[R(\tau)|s_0 = s]$$

The **state-action value function** $Q^{\pi}(s, a)$ (more commonly referred to as the Q function) gives the expected return given that the agent starts in state s, applies action a there, and then acts according to its policy π

State-action value function

$$Q^{\pi}(s, a) = \underset{\tau \sim \pi}{E}[R(\tau)|s_0 = s, a_0 = a]$$

Optimal Value Functions

Optimal state value function

$$V^*(s) = \arg\max_{\pi} V^{\pi}(s) \; \forall s \in \mathcal{S}$$

Optimal state-action value function

$$Q^*(s,a) = \arg\max_{\pi} Q^{\pi}(s,a) \; \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Important consequences:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} Q^*(s,a)$$

$$a^* = \arg\max_{a} Q^*(s,a)$$

Advantage Function

Various learning algorithms also make use of the so-called **advantage function**

Advantage function

$$A(s, a) = Q(s, a) - V(s)$$

This function quantifies the benefit of taking a specific action a in state s over taking an action that is sampled from $\pi(a|s)$

Preliminaries

→ Classical model-based learning algorithms

Model-free learning

Deep reinforcement learning

Reinforcement learning outlook

Model-Based Learning

If we assume that the model is given and we use discounted rewards, the optimal value function is given by

$$V^*(s) = \max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

$V^*(s)$ can be found by solving the system of equations

$$V^*(s) = \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

These are famously known as the **Bellman equations**

Given $V^*(s)$, the optimal policy is then

$$\pi^*(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right)$$

Value Iteration

A learning algorithm that makes direct use of the Bellman equations is **value iteration**

Value iteration starts by random initialisation of $V(s), \forall s \in \mathcal{S}$ and iterates over the equations

$$V_i(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_i(s') \right)$$

until a convergence criterion is reached (e.g. $\forall s \in \mathcal{S},\ V^i(s) - V^{i-1}(s) \leq \epsilon$ for some small ϵ)

Value iteration does not require any interaction with environment and is guaranteed to find an optimal policy after convergence (or even before)

Policy Iteration

A related algorithm that directly modifies the policy rather than calculating it indirectly from the value function is **policy iteration**

Policy iteration randomly initialises π and then solves the system of equations

$$V^{\pi_i}(s) = r(s, \pi_i(s)) + \gamma \sum_{s'} P(s'|s, a) V^{\pi_i}(s')$$

An updated policy is then found as

$$\pi_{i+1}(s) = \max_a \left(r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi_i}(s') \right)$$

Classical Learning Algorithm Summary

Classical model-based RL algorithms are dynamic programming-based; they thus have a strong mathematical foundation and are guaranteed to find an **optimal** policy

An additional benefit of these is that they do not require direct interaction with the environment

The assumption about the existence of a model is however too limiting for practical purposes

Preliminaries

Classical model-based learning algorithms

→ Model-free learning

Deep reinforcement learning

Reinforcement learning outlook

Model-Free Learning Preliminaries

In model-free RL, the transition probability distribution $\mathcal{T}$ and the reward function r are unknown, so we cannot use the algorithms above; an agent has to explore the environment on its own instead

The model-free learning setup is that we have trajectories τ and the accompanying rewards along the trajectories

An optimal policy has to be found from these experiences

There are two major types of model-free algorithms:

- **Temporal difference learning**: perform value/policy updates at every step
- **Monte Carlo learning**: estimate the return and then perform value/policy updates

Temporal Difference - TD(λ) - Learning

The **TD(λ)** learning algorithm attempts to bring the value function closer to the reward function, while preventing myopic updates

The parameter λ controls the amount of prediction during learning

For TD(0), only a single-step prediction is done:

$$V(s_t) = V(s_t) + \alpha \left(r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t) \right)$$

where α is a learning rate

In the more general case of $\lambda > 0$, older states are taken into account as well

Q-Learning

A very popular temporal difference RL algorithm is Q-learning

The algorithm is very similar to TD(λ), but instead of the state-value function, Q-learning estimates the state-action value function

The Q-learning update rule is given by

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

Using a function approximator (e.g. a neural network), Q-learning can be extended to continuous state spaces (as we will see a bit later)

Policy Search (PS)

Value-based algorithms, such as TD(λ) and Q-learning, derive the policy from the value function

Policy search algorithms, as the name implies, circumvent the need for the value function by optimising in the policy space directly

Particularly for robotics, PS algorithms are very useful since they allow incorporating prior knowledge about the policy

There are three major families of PS algorithms:

Policy gradient methods

The policy is parameterised as a differentiable function and optimisation is performed using gradient-based methods

Information-theoretic algorithms

The distance between the old policy and the updated policy is bounded to prevent large changes between subsequent policies

Expectation-maximisation-based methods

Policy search is formulated as an expectation-maximisation inference problem

Policy Gradients

Given a parameterised policy π_θ, a **policy gradient** estimates the gradient of the expected return and modifies the parameters θ using the update rule

$$\theta \leftarrow \theta + \nabla J(\theta)$$

where we recall that

$$J(\theta) = \int R(\theta) P(\tau|\theta) d\tau$$

Policy gradient algorithms often make use of the so-called likelihood ratio trick

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)$$

while estimating the gradient of J

REINFORCE Algorithm[3]

REINFORCE is a quite old policy gradient algorithm that forms the backbone of many modern PG algorithms

The algorithm was originally formulated for neural network-based policies, but its general formulation is applicable to any differentiable policy

A high-level overview of the algorithm is given below:

1: Initialise θ randomly
2: **for** $i \leftarrow 1$ **to** N **do**
3: $\quad \mathcal{T} \leftarrow \{\}$
4: $\quad$ **for** $j \leftarrow 1$ **to** M **do**
5: $\quad\quad \tau \leftarrow \texttt{sample}(\pi_\theta)$
6: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \tau$
7: $\quad J_\theta \leftarrow \frac{1}{M} \sum_{j=1}^{M} \sum_t r_t^j$
8: $\quad \theta \leftarrow \theta + \nabla J_\theta$

[3] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229-256, 1992.

Actor-Critic Learning

Value-based algorithms can be referred to as **critic-based**

Policy search algorithm are also called **actor-based**

As would be expected, a combination of the two also exists - this forms the so-called **actor-critic** family of RL algorithms, which estimate the value function and maintain a policy at the same time

Actor-critic algorithms make use of a **baseline** b when estimating the gradient of J

$$\nabla_\theta J(\theta) = E\left[\sum_{i=0}^{T} \nabla_\theta \log P_\theta(a_t|s_t)(R_t - b_t)\right]$$

The advantage function that we saw before is a commonly used baseline (giving rise to the A3C algorithm[4] and its variants)

The benefit of these is that the variance of policy updates is reduced

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," CoRR, vol. abs/1602.01783, 2017. Available: https://arxiv.org/abs/1602.01783

Preliminaries

Classical model-based learning algorithms

Model-free learning

→ Deep reinforcement learning

Reinforcement learning outlook

Deep Reinforcement Learning

In deep reinforcement learning, the policy is generally represented by a (deep) neural network

These algorithms can make use of the capabilities of deep neural networks for processing high-dimensional visual input, thereby representing more expressive state spaces

Various recent interesting applications in robotics make direct use of deep RL[5,6,7]

We will only *briefly* look at a few popular deep RL algorithms here

[5] M. Andrychowicz et al., "Hindsight Experience Replay," CoRR, vol. abs/1707.01495, 2017. Available: https://arxiv.org/abs/1707.01495

[6] M. A. Lee et al., "Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8943-8950, 2019.

[7] Y. Urakami, A. Hodgkinson, C. Carlin, R. Leu, L. Rigazio, and P. Abbeel, "DoorGym: A Scalable Door Opening Environment And Baseline Agent," CoRR, vol. abs/1908.01887, 2017. Available: https://arxiv.org/abs/1908.01887

Deep Q-Learning[8]

As the name suggests, deep Q-learning represents the Q-function using a deep neural network

The first success story of deep Q-learning was that of playing Atari games

The objective function that is being minimised here is often of the form

$$\mathcal{L}(\theta) = E\left[Q(s_t, a_t) - \left(r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)\right)\right]$$

Q-learning can be very unstable when a neural network is used to represent the value function (and may not even converge) - but there are practical tricks to improve the convergence

[8] V. Mnih et al. "Human-level Control Through Deep Reinforcement Learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

Deep Deterministic Policy Gradient (DDPG)[9]

DDPG is a popular deep RL algorithm for deterministic policies in continuous action spaces

In DDPG, an estimate of the Q-function is maintained in the form of a deep neural network, but the function is parameterised so that continuous actions can be selected; we denote this as Q_ϕ

A deterministic policy $\mu_\theta(s)$ is calculated from the value function as

$$\max_\theta E\left[Q_\phi(s, \mu_\theta(s))\right]$$

DDPG is thus an actor-critic algorithm (and learns off-policy since it reuses experiences from old policies in the gradient estimation)

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, "Continuous Control With Deep Reinforcement Learning," CoRR, vol. abs/1509.02971, 2015. Available: https://arxiv.org/abs/1509.02971

Proximal Policy Optimisation (PPO)[10]

PPO is an algorithm similar to information-theoretic policy search methods

The optimisation objective of PPO is maximising

$$\mathcal{L}(\theta) = E\left[\min\left(q_t(\theta)A_\theta(s,a), \text{clip}\left(q_t(\theta), 1-\epsilon, 1+\epsilon\right)A_\theta(s,a)\right)\right]$$

for a small ϵ where

$$q_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$$

$$\text{clip}(x,y,z) = \begin{cases} y & \text{if } x < y \\ z & \text{if } x > z \\ x & \text{otherwise} \end{cases}$$

In principle, PPO maintains a (deep) policy network (thus it is considered a deep RL algorithm) and tries to limit the amount by which the policy is updated (hence the parallel with information-theoretic algorithms)

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," CoRR, vol. abs/1707.06347, 2017. Available: https://arxiv.org/abs/1707.06347

Preliminaries

Classical model-based learning algorithms

Model-free learning

Deep reinforcement learning

→ Reinforcement learning outlook

Fundamental Questions in RL

When or how well does a learning algorithm work?

One of the reasons for the vast number of RL algorithms is that not all algorithms seem to be suitable for all problems

How long will it take for an algorithm to converge?

Due to the experience-based nature of RL, it is difficult to predict when (or sometimes even whether) an algorithm will converge to an optimal policy; for instance, policy search algorithms cannot guarantee a globally optimal policy

What is a good reward function for a given problem?

The reward function induces prior knowledge into an RL problem, but poorly specified reward functions can be detrimental to learning (or may even cause the agent to learn an unintended policy)

and various others...

Interesting Open Problems in Reinforcement Learning

Transfer learning

Given a learned policy for one problem, how can we reuse the policy to learn a related problem? When is such transfer even possible?

Curriculum learning

How to split a learning problem into a sequence of subproblems in which simpler problems are learned before more difficult ones?

Learning predictive models

Models can speed up learning, but how can useful models be learned so that they can be used effectively?

Safe reinforcement learning

How to prevent the learner from violating safety constraints during learning?

Simulation to real system policy transfer

Policies learned in simulations may overfit to some inherent inaccuracies of the simulator; how can such policies be transferred to real systems?

The eternal problem - sample efficiency

How to speed up learning so that the required number of learning experiences is reasonably small?

Reinforcement Learning and Control

The problem that reinforcement learning is trying to solve (particularly in robotics) is often the same as that solved by classical control theory: get the system to act so that it satisfies a certain objective

The approaches are however different:

- control theory models systems and controllers explicitly
- RL lets agents derive controllers through direct experience

Both approaches have practical advantages and disadvantages on their own

A combination of the two is most likely ideal in practice[11,12,13]

[11] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian Processes for Data-Efficient Learning in Robotics and Control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408-423, 2015.

[12] Daniel Görges, "Relations between Model Predictive Control and Reinforcement Learning," in *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920-4928, 2017.

[13] A. Marco et al., "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization," in *2017 IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2017, pp. 1557-1563.

A Zoo of Reinforcement Learning Libraries

- *OpenAI gym*: http://gym.openai.com - simulated environments for training RL agents
- *OpenAI safety gym*: https://github.com/openai/safety-gym - environments for safe RL
- *RLkit*: https://github.com/vitchyr/rlkit - PyTorch implementations of deep RL algorithms
- *Softlearning*: https://github.com/rail-berkeley/softlearning - Tensorflow implementations of entropy maximisation algorithms
- *rlpyt*: https://github.com/astooke/rlpyt - another PyTorch-based deep RL library

Summary

- RL is a learning paradigm in which an agent learns by exploring the environment and receiving rewards from it
- The main objective in RL is finding a policy that maximises the expected agent performance
- The developments in the field started by exploring discrete state and action spaces
  - Under a given model, the Bellman optimality equation is used to find a (guaranteed) optimal policy
- For continuous state and action spaces, there are (in general) no such theoretical guarantees
- Modern applications of RL use policy search algorithms (particularly in robotics)
- Deep reinforcement learning combines RL with deep learning, allowing RL agents to process high-dimensional inputs

Additional Useful Resources

R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2nd ed., The MIT Press, 2018.

D. Silver, UCL Course on Reinforcement Learning, http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html