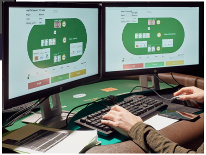
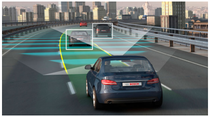
Intro to deep RL,
the concept of generalization and
the importance of representation learning

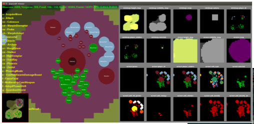
Vincent François-Lavet

27 octobre 2023

Motivation : Overview







Motivation

FIGURE – Video of a trained agent for an ATARI game : Seaquest

Motivation : Robotics

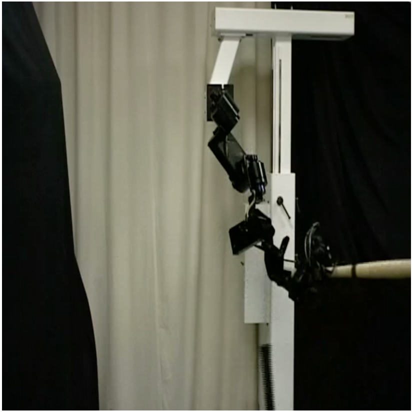


FIGURE – Application in robotics (credits : Jan Peters'team, Darmstadt)

Outline

Markov Decision Processes

Outline

Markov Decision Processes

Different families of techniques in Reinforcement learning

- Model-based methods (planning-based techniques)
- Model-free techniques

Outline

Markov Decision Processes

Different families of techniques in Reinforcement learning

- Model-based methods (planning-based techniques)
- Model-free techniques

How can we scale with deep learning as a function approximator ?

Outline

- Markov Decision Processes
- Different families of techniques in Reinforcement learning
  - Model-based methods (planning-based techniques)
  - Model-free techniques
- How can we scale with deep learning as a function approximator ?
- Real-world examples using deep RL

Outline

- Markov Decision Processes
- Different families of techniques in Reinforcement learning
  - Model-based methods (planning-based techniques)
  - Model-free techniques
- How can we scale with deep learning as a function approximator ?
- Real-world examples using deep RL
- Generalization in deep RL

Outline

- Markov Decision Processes
- Different families of techniques in Reinforcement learning
  - Model-based methods (planning-based techniques)
  - Model-free techniques
- How can we scale with deep learning as a function approximator ?
- Real-world examples using deep RL
- Generalization in deep RL
- Improving generalization
  - Combining model-free and model-based
  - Using self-supervised learning and abstract representations

Outline

Markov Decision Processes

Different families of techniques in Reinforcement learning

- Model-based methods (planning-based techniques)
- Model-free techniques

How can we scale with deep learning as a function approximator ?

Real-world examples using deep RL

Generalization in deep RL

Improving generalization

- Combining model-free and model-based
- Using self-supervised learning and abstract representations

Abstract representations for reasoning, exploration and transfer learning

Outline

Markov Decision Processes

Different families of techniques in Reinforcement learning
- Model-based methods (planning-based techniques)
- Model-free techniques

How can we scale with deep learning as a function approximator ?

Real-world examples using deep RL

Generalization in deep RL

Improving generalization
- Combining model-free and model-based
- Using self-supervised learning and abstract representations

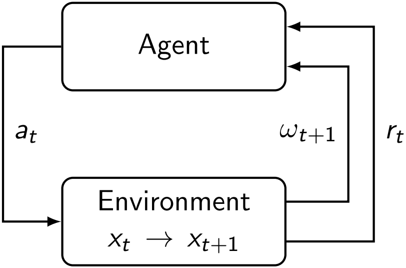
Abstract representations for reasoning, exploration and transfer learning

A few other challenges for RL (disentanglement of controllable and uncontrollable feature + causal representations)

Markov Decision Processes

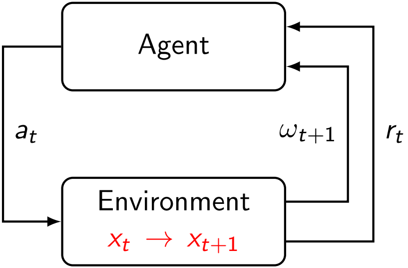
Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Objective

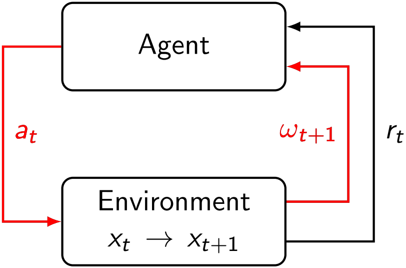
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



transitions
are usually
stochastic

Objective

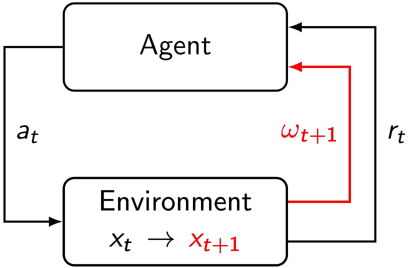
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations and
actions may be
high dimensional

Objective

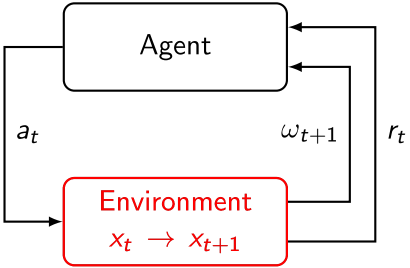
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations may not
provide full knowledge
of the underlying
state : $\omega_t \neq x_t$

Objective

From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Experience may be constrained (e.g., not access to an accurate simulator or limited data)

Example of a Markov Decision Process (MDP)

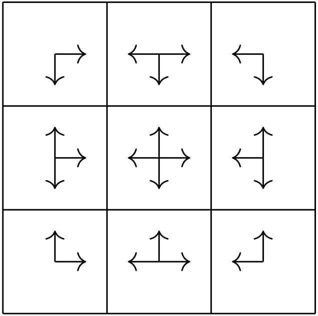
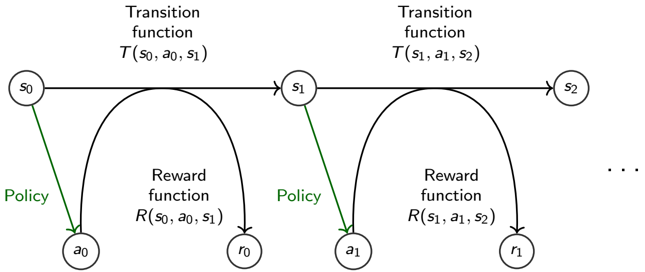


FIGURE – Representation of a (deterministic) mini grid-world with 9 discrete states and 4 discrete actions. The agent is able to move in the four directions, except when the agent is trying to get "out of the grid-world".

Definition of an MDP

An MDP can be defined as a 5-tuple $(\mathcal{X}, \mathcal{A}, T, R, \gamma)$ where :

- $\mathcal{S}$ is a finite set of states $\{1, \ldots, N_{\mathcal{S}}\}$,
- $\mathcal{A}$ is a finite set of actions $\{1, \ldots, N_{\mathcal{A}}\}$,
- $T : \mathcal{X} \times \mathcal{A} \to \mathbb{P}(\mathcal{X})$ is the transition function (set of conditional transition probabilities between states),
- $R : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathcal{R}$ is the reward function, where $\mathcal{R}$ is a continuous set of possible rewards in a range $R_{max} \in \mathbb{R}^+$ (e.g., $[0, R_{max}]$),
- $\gamma \in [0, 1)$ is the discount factor.



Performance evaluation

In an MDP $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, the discounted expected return $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ ($\pi \in \Pi$, e.g., $\mathcal{X} \to \mathcal{A}$) is defined such that

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right], \tag{1}$$

with $\gamma \in [0, 1)$.

From the definition of the (discounted) expected return, the optimal expected return can be defined as

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s). \tag{2}$$

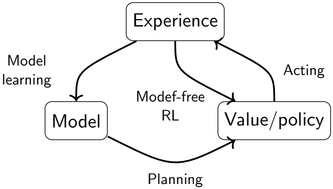
and the optimal policy can be defined as :

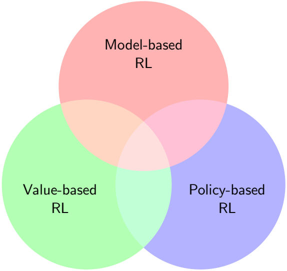
$$\pi^*(s) = \operatorname*{argmax}_{\pi \in \Pi} V^\pi(s). \tag{3}$$

Overview of the techniques used for finding the optimal policy π^*

In general, an RL agent may include one or more of the following components :

- a model of the environment in conjunction with a planning algorithm.
- a representation of a value function that provides a prediction of how good is each state or each couple state/action,
- a direct representation of the policy $\pi(s)$ or $\pi(s, a)$, or





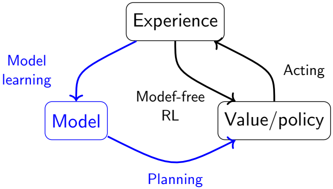
Different families of techniques in Reinforcement learning

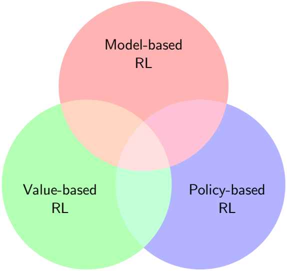
Overview of the techniques used for finding the optimal policy π^*

In general, an RL agent may include one or more of the following components :

→ a model of the environment in conjunction with a planning algorithm.

- a representation of a value function that provides a prediction of how good is each state or each couple state/action,
- a direct representation of the policy $\pi(s)$ or $\pi(s, a)$, or





Model-based methods (planning-based techniques)

Motivation for planning with tree search

Given that you're playing the crosses, what would be your next move ?

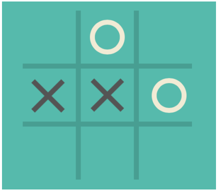


FIGURE – Illustration of a state in the tic-tac-toe game.

→ how did you come up with that choice ?

Monte-Carlo Tree Search methods

The overall idea is to estimate the action with the highest expected return.

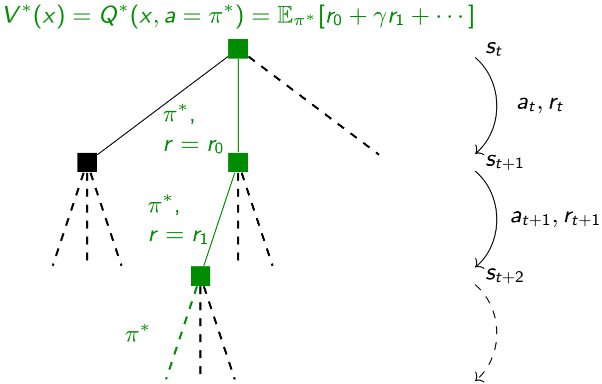


FIGURE – Illustration of model-based planning with tree search.

Motivation

MCTS algorithms need (only) a generative model of the environment (i.e. model-based) :

$$s_{t+1}, r_t \sim G(s_t, a_t)$$

Advantages :

- it is possible to obtain samples without having the whole transition function for the model in an explicit form.
- it can learn a strong policy only where needed (from the current state s).
- it is useful for a sequence of decisions.

MCTS

MCTS can converge to the optimal policy (finite action space, finite horizon) from any state s as long as the generative model is accurate.
However,

- The breath of search grows with the actions space.
- The depth of search grows with the horizon considered.

Applications

- Tree search algorithms can be used along with different heuristics as well as model-free deep RL techniques.

→ MCTS has been a key part of alpha Go for instance.

Strengths and weaknesses of model-based methods

The respective strengths of the model-free versus model-based approaches depend on different factors.

- ✓ For some tasks, the model of the environment is available or can be learned efficiently due to the particular structure of the task.
- ✗ If the agent does not have access to a generative model of the environment, the learned model will have some inaccuracies.
- ✗ A model-based approach requires working in conjunction with a planning algorithm, which is computationally demanding.

Model-free techniques

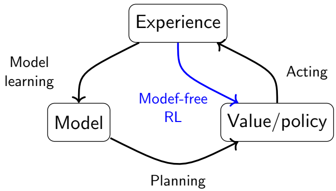
Overview of deep RL

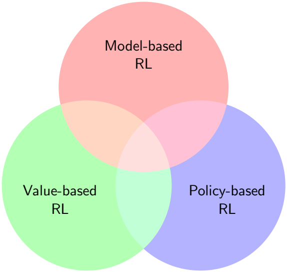
In general, the learning algorithm in RL may include one or more of the following components :

- a model of the environment in conjunction with a planning algorithm.

→ a value function that provides a prediction of how good is each state or each couple state/action (main focus), or

→ a direct representation of the policy $\pi(s)$ or $\pi(s, a)$





Deep learning has brought its generalization capabilities to RL.

Value based methods : recall

In an MDP $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, the expected return $V^{\pi}(s) : \mathcal{S} \to \mathbb{R}$ ($\pi \in \Pi$, e.g., $\mathcal{S} \to \mathcal{A}$) is defined such that

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right],$$

with $\gamma \in [0, 1)$.

Value based methods : recall

In addition to the V-value function, the Q-value function $Q^\pi(s, a) : \mathcal{S} \times A \to \mathbb{R}$ is defined as follows :

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right].$$

The particularity of the Q-value function as compared to the V-value function is that the optimal policy can be obtained directly from $Q^*(s, a)$:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^*(s, a).$$

Value based methods : recall

The Bellman equation that is at the core of value-based learning makes use of the fact that the Q-function can be written in a recursive form :

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\left[\sum\nolimits_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] \\ &= \mathbb{E}\left[r_t + \sum\nolimits_{k=1}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right] \\ &= \mathbb{E}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a' \sim \pi) \mid s_t = s, a_t = a, \pi\right] \end{aligned}$$

In particular :

$$\begin{aligned} Q^{*}(s,a) &= \mathbb{E}\left[r_t + \gamma Q^{*}(s_{t+1}, a' \sim \pi^{*}) \mid s_t = s, a_t = a, \pi^{*}\right] \\ &= \mathbb{E}\left[r_t + \gamma \max_{a' \in \mathcal{A}} Q^{*}(s_{t+1}, a') \mid s_t = s, a_t = a, \pi^{*}\right] \end{aligned}$$

Convergence Q-learning

Theorem : Given a finite MDP, the Q-learning algorithm given by the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)],$$

converges w.p.1 to the optimal Q-function as long as

- $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, and
- the exploration policy π is such that $P_\pi[a_t = a | s_t = s] > 0, \forall (s, a)$.

Convergence Q-learning

Theorem : Given a finite MDP, the Q-learning algorithm given by the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)],$$

converges w.p.1 to the optimal Q-function as long as

- $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$, and
- the exploration policy π is such that $P_\pi[a_t = a | s_t = s] > 0, \forall (s, a)$.

Limitations of tabular approaches

A tabular approach fails for large scale problems due to the curse of dimensionality.

- Robot states with 10 features (e.g. position, speed, angle of joints) discretized into 100 bins $\rightarrow 100^{10} = 10^{20}$ states.
- Chess : $\approx 10^{120}$ states
- Go : $\approx 10^{170}$ statess

Limitations of tabular approaches

A tabular approach fails for large scale problems due to the curse of dimensionality.

- Robot states with 10 features (e.g. position, speed, angle of joints) discretized into 100 bins $\rightarrow 100^{10} = 10^{20}$ states.
- Chess : $\approx 10^{120}$ states
- Go : $\approx 10^{170}$ statess

Three problems

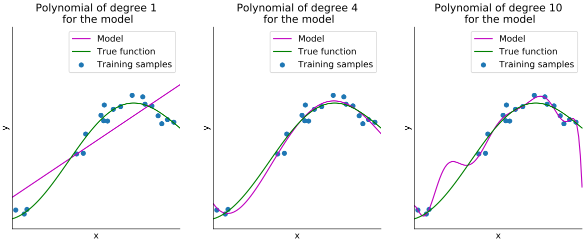
- Memory
- Compute time
- No generalization in the limited data context

How can we scale with deep learning as a function approximator ?

Function approximators

A function approximator $f : \mathcal{X} \to \mathcal{Y}$ parameterized with $\theta \in \mathbb{R}^{n_\theta}$ takes as input $x \in \mathcal{X}$ and gives as output $y \in \mathcal{Y}$ ($\mathcal{X}$ and $\mathcal{Y}$ depend on the application) :

$$y = f(x; \theta). \tag{4}$$



Different types of function approximators

	Flexible	Differentiable
Linear function approximator	✗	✓
SVMs	(✓)	✗
Tree-based approximators	(✓)	✗
⋮	⋮	⋮
Neural networks	✓	✓

Different types of function approximators

	Flexible	Differentiable
Linear function approximator	✗	✓
SVMs	(✓)	✗
Tree-based approximators	(✓)	✗
⋮	⋮	⋮
Neural networks	✓	✓

Neural networks (with backpropogation of the gradients) has brought its generalization capabilities to RL.

Gradient descent

We need an objective function to be minimized $J(\theta)$ that is a differentiable function of parameters θ.

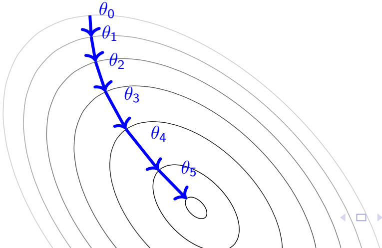
It starts at an initial point and then repeatedly takes a step opposite to the gradient direction of the function at the current point.

for $k = 0, 1, 2, \ldots$ **do**

$g_k \leftarrow \nabla J(\theta_k)$

$\theta_{k+1} \leftarrow \theta_k - \alpha_k g_k$

end for

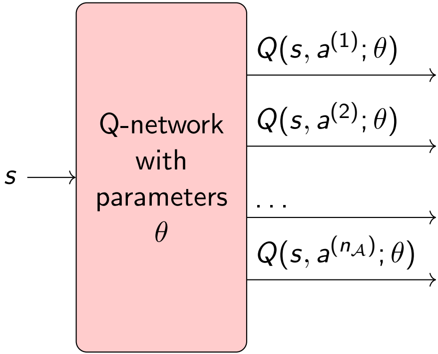


Q-learning with function approximator

We can represent value functions with function approximators and parameters θ :

$$Q(s, a; \theta) \approx Q(s, a)$$

Q-network usual structure (finite number $n_{\mathcal{A}}$ of actions) :



Q-learning with function approximator

The parameters θ are updated with gradient descent :

$$\theta := \theta - \alpha \nabla_\theta \underbrace{\left(Q(s, a; \theta) - Y_k^Q \right)^2}_{\text{Objective to be minimized}}$$

with the target

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k).$$

DQN algorithm

For Deep Q-Learning, we can represent value function by deep Q-network with weights θ (instabilities !). In the DQN algorithm :

- Replay memory
- Target network

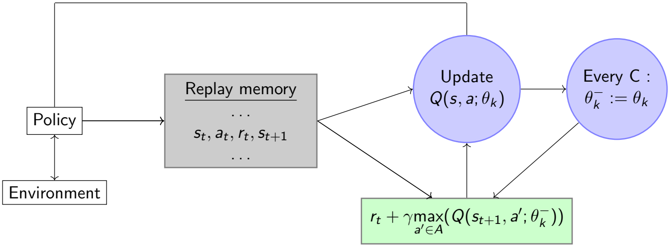


FIGURE – Sketch of the DQN algorithm. $Q(s, a; \theta_k)$ is initialized to random values (close to 0) everywhere on its domain and the replay memory is initially empty ; the target Q-network parameters θ_k^- are only updated every C iterations with the Q-network parameters θ_k and are held fixed between updates ; the update uses a mini-batch (e.g., 32 elements) of tuples $< s, a, r, s' >$ taken randomly in the replay memory.

Example : Mountain car

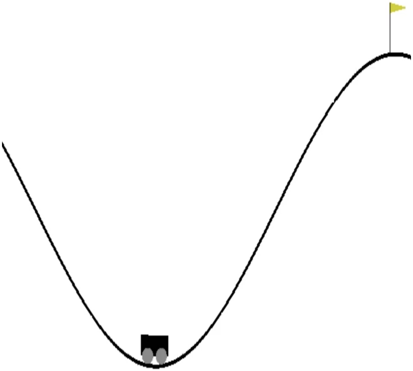


FIGURE – Mountain car optimal policy

Visualization of Q-values in mountain car

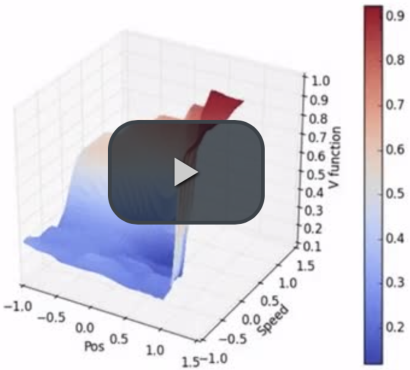


FIGURE – DQN for mountain car ($V = \max_a Q(s, a)$)

Mountain car

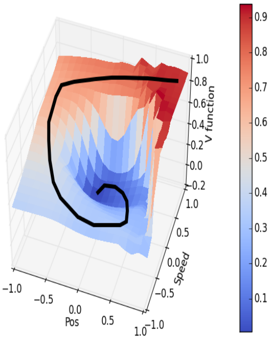


FIGURE – Application to the mountain car domain : $V = \max_a Q(s, a)$.

Deep learning architectures

Similarly to other fields of machine learning, there exists a whole zoo of deep learning modules that can be combined into one deep learning architecture.

Beyond deep Q learning

- Double DQN
- Multi-step learning
- How to discount deep RL
- Dueling network
- Fighting overfitting and lack of plasticity
- Distributional RL
- Some state-of-the-art results

State of the art

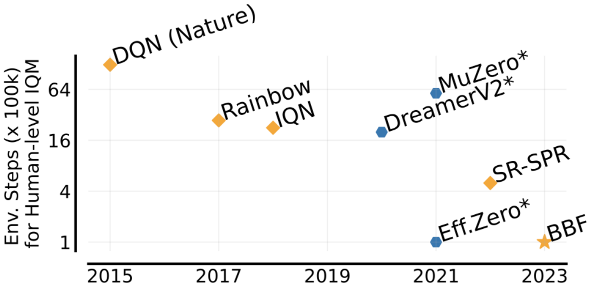


Figure – State of the art algorithms on Atari 100k benchmark.

Source : "Bigger, Better, Faster : Human-level Atari with human-level efficiency" (2023), M Schwarzer et al.

Real-world examples using deep RL

Stratospheric Balloon

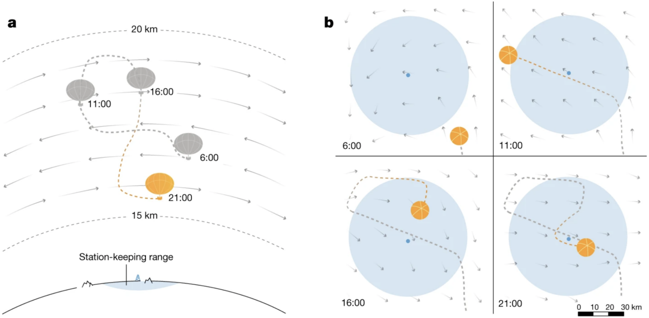
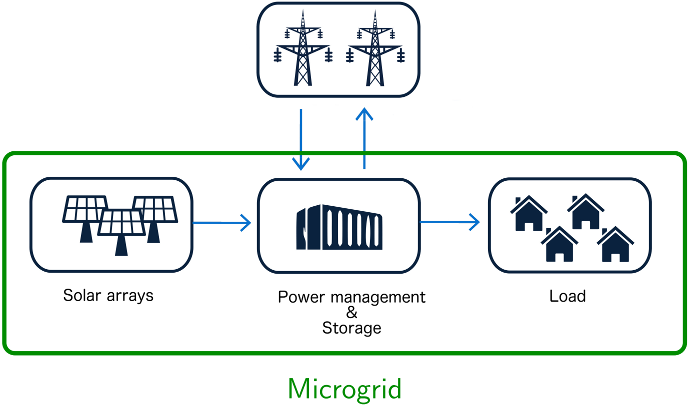


Figure – a, Schematic of a superpressure balloon navigating a wind field. The balloon remains close to its station by moving between winds at different altitudes. Its altitude range is indicated by the upper and lower dashed lines. b, The balloon's flight path, viewed from above. The station and its 50-km range are shown in light blue. Shaded arrows represent the wind field. The wind field constantly evolves, requiring the balloon to replan at regular intervals.

More details : *Autonomous navigation of stratospheric balloons using reinforcement learning*, M. G. Bellemare et al., 2020 (Nature).

Real-world application of deep RL : microgrid

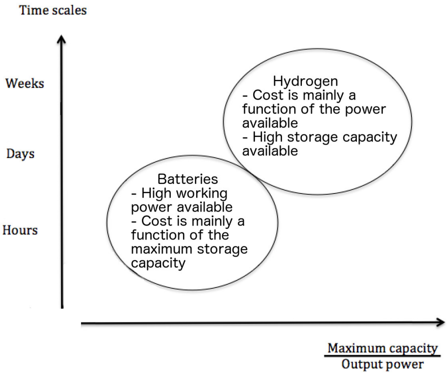
A microgrid is an electrical system that includes multiple loads and distributed energy resources that can be operated in parallel with the broader utility grid or as an electrical island.



Microgrids and storage

There exist opportunities with microgrids featuring :

- A short term storage capacity (typically batteries),
- A long term storage capacity (e.g., hydrogen).



Structure of the Q-network

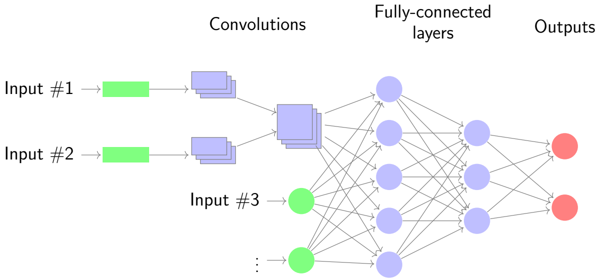


FIGURE – Sketch of the structure of the neural network architecture. The neural network processes the time series using a set of convolutional layers. The output of the convolutions and the other inputs are followed by fully-connected layers and the ouput layer. Architectures based on LSTMs instead of convolutions obtain similar results.

Results

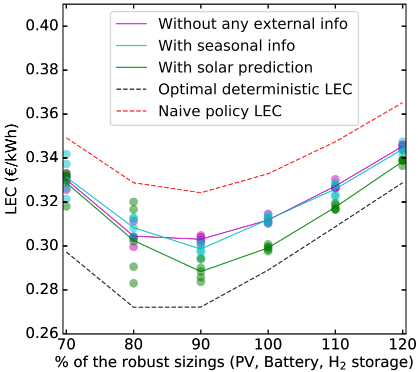


FIGURE – LEC on the test data function of the sizings of the microgrid.

More details : *Deep Reinforcement Learning Solutions for Energy Microgrids Management*, V. Francois-Lavet, D. Taralla, D. Ernst, R. Fonteneau, 2016 (EWRL).

Questions so far ?

Time for a break ?

Generalization in deep RL

Challenges of applying RL to real-world problems

In real-world scenarios, it is often not possible to let an agent interact freely and sufficiently in the actual environment :

- The agent may not be able to interact with the true environment but only with an inaccurate simulation of it. This is known as the **reality gap**.
- The agent might have access to only **limited data**. This can be due to safety constraints (robotics, medical trials, etc.), compute constraints or due to limited exogenous data (e.g., weather conditions, trading markets).

Challenges of applying RL to real-world problems

In order to deal with the reality gap and limited data, different elements are important :

- One can aim to develop **a simulator that is as accurate as possible**.
- One can design the learning algorithm so as to **improve generalization** (and/or use specific transfer learning methods).

Generalization

In an RL algorithm, *generalization* refers to either

- the capacity to achieve good performance in an environment where limited data has been gathered, or
- the capacity to obtain good performance in a related environment. This latter case can be tackled with specific *transfer learning* techniques.

Overview

To understand generalization in RL from limited data, we will

- recall the concept in supervised learning, and
- introduce the formulation in RL.

We'll then discuss how an agent can have a good generalization in RL (disclaimer : we'll see where deep RL comes in !)

Bias and overfitting in supervised learning

For one given $x \sim X$, the predictive model $f(x \mid D_{LS})$ can be illustrated as follows for unseen data $y \sim (Y \mid X = x)$:

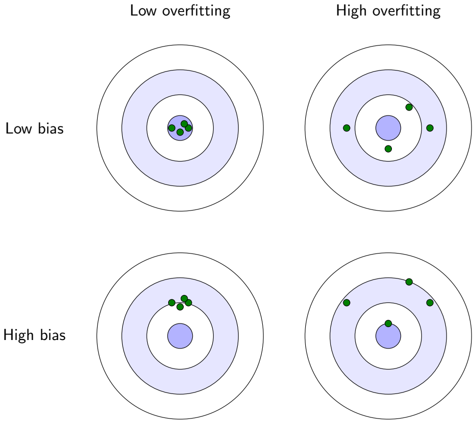
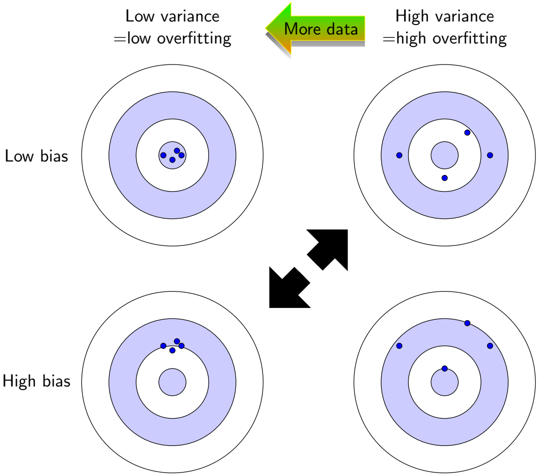


FIGURE – Illustration of bias and overfitting for unseen tuples, where Y is a 2D continuous RV for visualisation purposes.

Bias and overfitting in supervised learning

There are many choices to optimize the learning algorithm and there is usually a tradeoff between the bias and the overfitting terms to reach to best solution.



Bias and overfitting in supervised learning

Assuming a random sampling scheme $D_{LS} \sim \mathcal{D}_{LS}$, $f(x \mid D_{LS})$ is a random variable, and so is its average error over the input space. The expected value of this quantity is given by :

$$I[f] = \mathop{\mathbb{E}}_{X} \mathop{\mathbb{E}}_{D_{LS}} \mathop{\mathbb{E}}_{Y|X} L(Y, f(X \mid D_{LS})), \tag{5}$$

where $L(\cdot, \cdot)$ is the loss function.

Bias and overfitting in supervised learning

Assuming a random sampling scheme $D_{LS} \sim \mathcal{D}_{LS}$, $f(x \mid D_{LS})$ is a random variable, and so is its average error over the input space. The expected value of this quantity is given by :

$$I[f] = \mathop{\mathbb{E}}_{X} \mathop{\mathbb{E}}_{D_{LS}} \mathop{\mathbb{E}}_{Y|X} L(Y, f(X \mid D_{LS})), \tag{5}$$

where $L(\cdot,\cdot)$ is the loss function. If $L(y,\hat{y}) = (y-\hat{y})^2$, the error naturally gives the **bias-variance decomposition** :

$$\mathop{\mathbb{E}}_{D_{LS}} \mathop{\mathbb{E}}_{Y|X} (Y - f(X \mid D_{LS}))^2 = \sigma^2(x) + \text{bias}^2(x), \tag{6}$$

where

$$\text{bias}^2(x) \triangleq \left(\mathbb{E}_{Y|x}(Y) - \mathbb{E}_{D_{LS}} f(x \mid D_{LS})\right)^2,$$

$$\sigma^2(x) \triangleq \underbrace{\mathbb{E}_{Y|x}\left(Y - \mathbb{E}_{Y|x}(Y)\right)^2}_{\text{Internal variance}} + \underbrace{\mathbb{E}_{D_{LS}}\Big(f(x \mid D_{LS}) - \mathbb{E}_{D_{LS}} f(x \mid D_{LS})\Big)^2}_{\text{Parametric variance = overfitting}}.$$

Bias and overfitting in reinforcement learning

This bias-variance decomposition highlights a tradeoff between

- an error directly introduced by the learning algorithm (the bias) and
- an error due to the limited amount of data available (the parametric variance).

Bias and overfitting in reinforcement learning

This bias-variance decomposition highlights a tradeoff between

- an error directly introduced by the learning algorithm (the bias) and
- an error due to the limited amount of data available (the parametric variance).

Note that there is no such direct bias-variance decomposition for loss functions other than the L_2 loss! It is however always possible to decompose the prediction error with a term related to the lack of expressivity of the model (the bias) and a term due to the limited amount of data (overfitting comes from the variance of $f(x \mid D_{LS})$ on the loss when $D_{LS} \sim \mathcal{D}_{LS}$ but $\neq$ statistical variance if loss function is not L_2).

Bias and overfitting in supervised learning

Since there is no direct bias-variance decomposition for loss functions other than L_2 loss in supervised learning, there is not an actual "bias-variance" tradeoff in RL.

However, there is still a tradeoff between a sufficiently rich learning algorithm (to reduce the model bias, which is present even when the amount of data would be unlimited) and a learning algorithm not too complex (so as to avoid overfitting to the limited amount of data).

Bias and overfitting in RL

The *batch* or *offline* algorithm in RL can be seen as mapping a dataset $D \sim \mathcal{D}$ into a policy π_D (independently of whether the policy comes from a model-based or a model-free approach) :

$$D \to \pi_D.$$

In an MDP, the suboptimality of the expected return can be decomposed as follows :

$$\underset{D\sim\mathcal{D}}{\mathbb{E}}[V^{\pi^*}(x) - V^{\pi_D}(x)] = \underbrace{(V^{\pi^*}(x) - V^{\pi_{D\infty}}(x))}_{\text{asymptotic bias}} + \underbrace{\underset{D\sim\mathcal{D}}{\mathbb{E}}[(V^{\pi_{D\infty}}(x) - V^{\pi_D}(x))}_{\substack{\text{error due to finite size of the dataset } D_s \\ \text{referred to as } \textit{overfitting}}}].$$

How to obtain the best policy ?

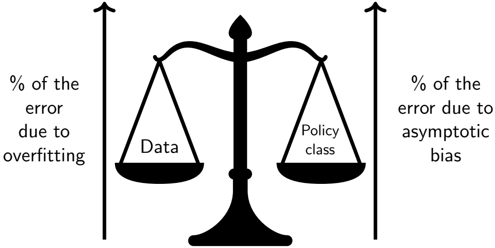


FIGURE – Schematic representation of the bias-overfitting tradeoff.

How to improve generalization ?

We can improve generalization of RL thanks to the following elements :

- an **abstract representation** that discards non-essential features,
- the **objective function** (e.g., reward shaping, tuning the training discount factor) and
- the **learning algorithm** (type of function approximator and model-free vs model-based).

And of course, if possible :

- improve the dataset (exploration/exploitation dilemma in an online setting)

Improving generalization

Combining model-free and model-based

Choice of the learning algorithm : a parallel with neurosciences

In cognitive science, there is a dichotomy between two modes of thoughts (*D. Kahneman. (2011). Thinking, Fast and Slow*) :

- a "System 1" that is fast and instinctive and
- a "System 2" that is slower and more logical.

FIGURE – System 1

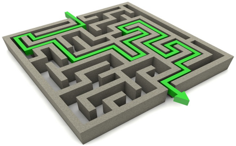


FIGURE – System 2

In deep reinforcement, a similar dichotomy can be observed when we consider the model-free and the model-based approaches.

Choice of the learning algorithm and function approximator selection

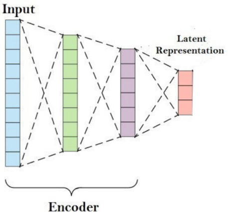
- The function approximator in deep learning characterizes how the features will be treated into higher levels of abstraction. A fortiori, it is related to feature selections (e.g., an attention mechanism), etc.
- Depending on the task, finding a performant function approximator is easier in either a model-free or a model-based approach. The choice of relying more on one or the other approach is thus also a crucial element to improve generalization.

Using self-supervised learning and abstract representations

Foreword

Vocabulary

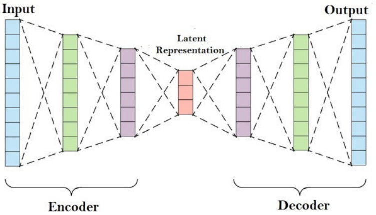
- An **encoder** is a specific deep learning component that transforms the input (to reduce the dimensionality).
- An **abstract representation** or **latent representation** is the representation obtained after the input goes through the encoder.



Foreword

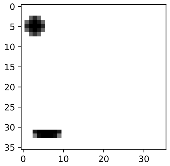
Vocabulary

- An **encoder** is a specific deep learning component that transforms the input (to reduce the dimensionality).
- An **abstract representation** or **latent representation** is the representation obtained after the input goes through the encoder.



Catcher

This environment has only a few important features
(i) the position of the paddle and
(ii) the position of the blocks.



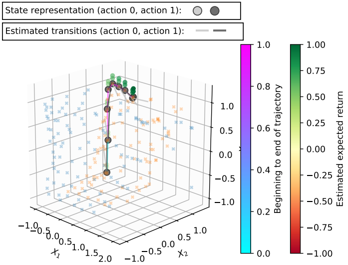


FIGURE –
Without interpretability loss.

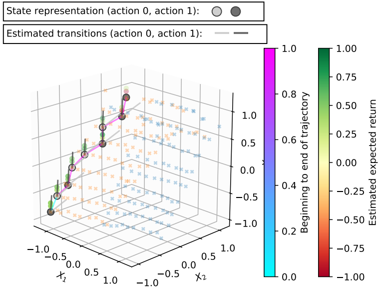


FIGURE – With interpretability loss : $v(a^{(1)}) = (1, 1)$ and $v(a^{(2)}) = (-1, 1)$.

Abstract representations for reasoning, exploration and transfer learning

Combining model-based and model-free via abstract representations

We are interested to learn both the model and the value function through one abstract representation :

- it can enforce a good generalization (information bottleneck),
- planning is computationally efficient,
- it facilitates interpretation of the decisions taken by the agent,

Information bottleneck

As opposed to auto encoders, we seek to preserve only *relevant* information and we apply the *Information Bottleneck* (IB) principle to representation learning of state. This gives us the functional that we minimize

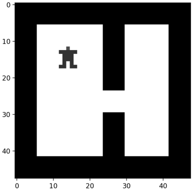
$$\mathcal{L} = I[S; X] - \beta I[(X); \{X^*, A^*\}]$$

This corresponds to a trade-off between

- minimizing the encoding rate $I[S; X]$ and
- maximizing the mutual information between the abstract state X (and reward) and the tuple previous abstract state, previous action (X^*, A^*).

Simple labyrinth

Abstract representation of states for a labyrinth task (without any reward).



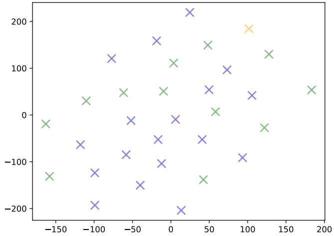


FIGURE – 2D representation using t-SNE (blue represents states where the agent is on the left part, green on the right part and orange in the junction).

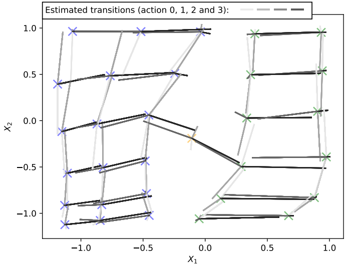


FIGURE – The CRAR agent is able to reconstruct a sensible representation of its environment in 2 dimensions.

Combined Reinforcement via Abstract Representations (CRAR)

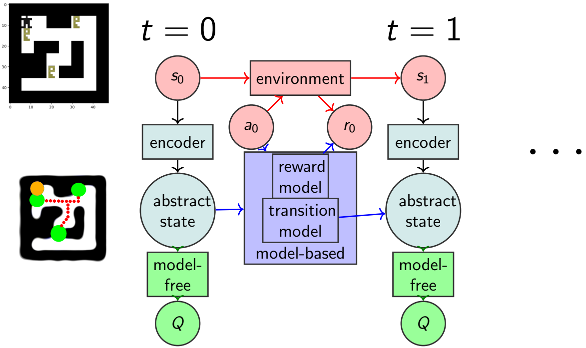


Figure – Illustration of the integration of model-based and model-free RL in the CRAR architecture.

The value function and the model are trained via the abstract representation.

Learning the model

Here is how we learn the internal model :

$$\mathcal{L}_\tau(\theta_e, \theta_\tau) = \mid (e(s;\theta_e) + \tau(e(s;\theta_e), a; \theta_\tau) - e(s';\theta_e)) \mid^2,$$

Learning the model

Here is how we learn the internal model :

$$\mathcal{L}_\tau(\theta_e, \theta_\tau) = \mid (e(s;\theta_e) + \tau(e(s;\theta_e), a; \theta_\tau) - e(s';\theta_e)) \mid^2,$$

$$\mathcal{L}_\rho(\theta_e, \theta_\rho) = \mid r - \rho(e(s;\theta_e), a; \theta_\rho) \mid^2,$$
$$\mathcal{L}_g(\theta_e, \theta_g) = \mid \gamma - g(e(s;\theta_e), a; \theta_g) \mid^2 .$$

These losses train the weights of both the encoder and the model-based components.

Learning the model

Here is how we learn the internal model :

$$\mathcal{L}_\tau(\theta_e, \theta_\tau) = \mid (e(s;\theta_e) + \tau(e(s;\theta_e), a;\theta_\tau) - e(s';\theta_e)) \mid^2,$$

$$\mathcal{L}_\rho(\theta_e, \theta_\rho) = \mid r - \rho(e(s;\theta_e), a;\theta_\rho) \mid^2,$$
$$\mathcal{L}_g(\theta_e, \theta_g) = \mid \gamma - g(e(s;\theta_e), a;\theta_g) \mid^2 .$$

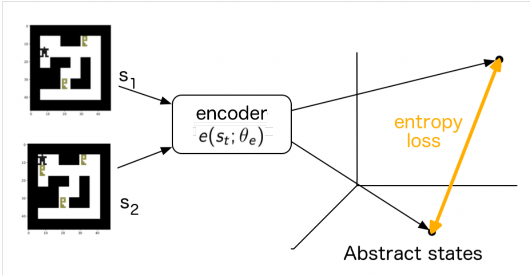
These losses train the weights of both the encoder and the model-based components.

Training of the value function is done with DDQN

When learning the transition function there is a pressure to decrease the amount of information being represented.
In our model, we introduce an **entropy loss** :

$$\mathcal{L}_{d1}(\theta_e) = \exp(-C_d \| e(s_1; \theta_e) - e(s_2; \theta_e) \|_2),$$

where s_1 and s_2 are random past states of the agent and C_d is a constant.



Interpretability

Interpretability can mean that some features of the state representation are distinctly affected by some actions. The following optional loss makes the predicted abstract state change aligned with the chosen embedding vector $v(a)$:

$$\mathcal{L}_{interpr}(\theta_e, \theta_\tau) = -\cos\Big(\tau(e(s;\theta_e), a; \theta_\tau)_{0:n}, v(a)\Big),$$

where cos stands for the cosine similarity.

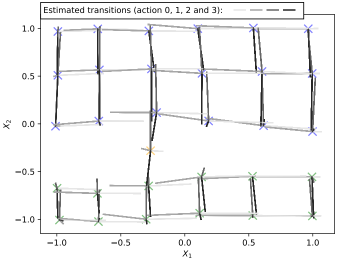


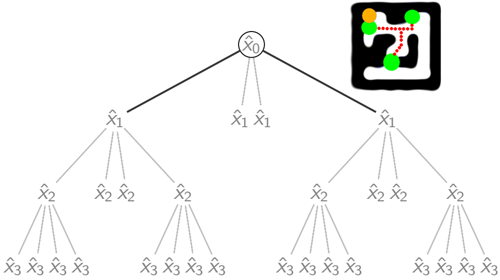
FIGURE – With enforcing $\mathcal{L}_{interpr}$ and $v(a_0) = [1, 0]$

Planning

The trajectories for some sequence of actions are estimated recursively as follows for any t' :

$$\hat{x}_{t'} = \begin{cases} e(s_t; \theta_e), \text{ if } t' = t \\ \hat{x}_{t'-1} + \tau(\hat{x}_{t'-1}, a_{t'-1}; \theta_\tau), \text{ if } t' > t \end{cases}$$

A set $\mathcal{A}^*$ of best potential actions is considered based on $Q(\hat{x}_t, a; \theta_Q)$ ($\mathcal{A}^* \subseteq \mathcal{A}$).



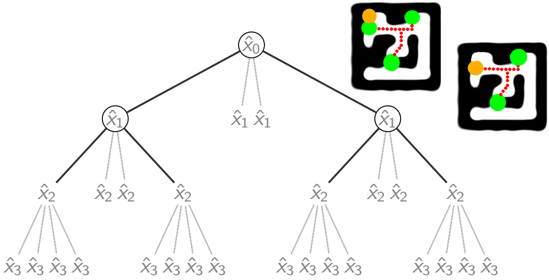
Expansion from $\hat{x}_0$ until a certain depth.

Planning

The trajectories for some sequence of actions are estimated recursively as follows for any t' :

$$\hat{x}_{t'} = \begin{cases} e(s_t; \theta_e), \text{ if } t' = t \\ \hat{x}_{t'-1} + \tau(\hat{x}_{t'-1}, a_{t'-1}; \theta_\tau), \text{ if } t' > t \end{cases}$$

A set $\mathcal{A}^*$ of best potential actions is considered based on $Q(\hat{x}_t, a; \theta_Q)$ ($\mathcal{A}^* \subseteq \mathcal{A}$).



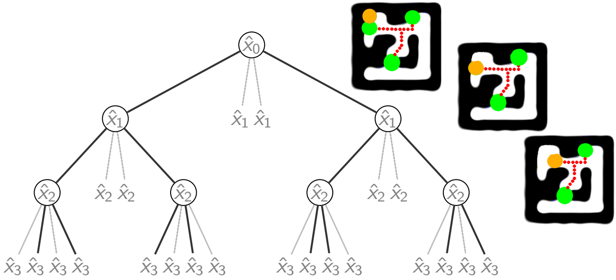
Expansion from $\hat{x}_0$ until a certain depth.

Planning

The trajectories for some sequence of actions are estimated recursively as follows for any t' :

$$\hat{x}_{t'} = \begin{cases} e(s_t;\theta_e), \text{ if } t' = t \\ \hat{x}_{t'-1} + \tau(\hat{x}_{t'-1}, a_{t'-1};\theta_\tau), \text{ if } t' > t \end{cases}$$

A set $\mathcal{A}^*$ of best potential actions is considered based on $Q(\hat{x}_t, a;\theta_Q)$ ($\mathcal{A}^* \subseteq \mathcal{A}$).



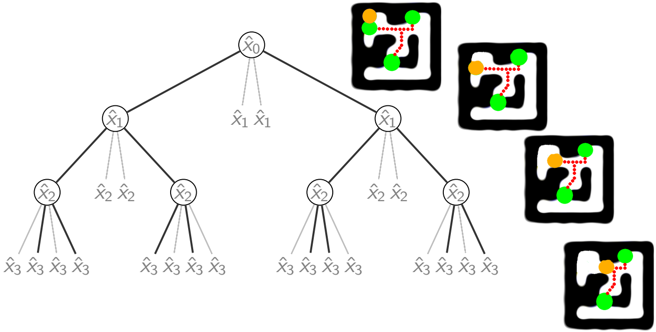
Expansion from $\hat{x}_0$ until a certain depth.

Planning

The trajectories for some sequence of actions are estimated recursively as follows for any t' :

$$\hat{x}_{t'} = \begin{cases} e(s_t;\theta_e), \text{ if } t' = t \\ \hat{x}_{t'-1} + \tau(\hat{x}_{t'-1}, a_{t'-1};\theta_\tau), \text{ if } t' > t \end{cases}$$

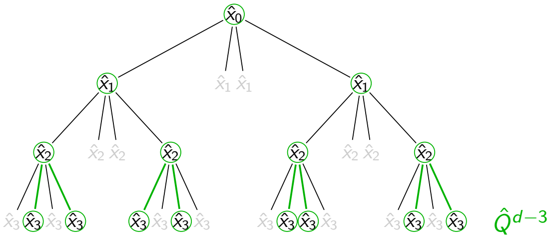
A set $\mathcal{A}^*$ of best potential actions is considered based on $Q(\hat{x}_t, a;\theta_Q)$ $(\mathcal{A}^* \subseteq \mathcal{A})$.



Expansion from $\hat{x}_0$ until a certain depth.

We define recursively the depth-d estimated expected return as

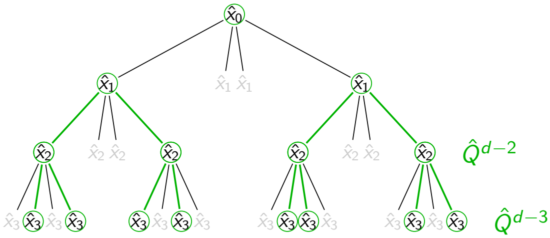
$$\hat{Q}^d(\hat{x}_t, a) = \begin{cases} \rho(\hat{x}_t, a; \theta_\rho) + g(\hat{x}_t, a; \theta_g) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}(\hat{x}_{t+1}, a'), & \text{if } d > 0 \\ Q(\hat{x}_t, a; \theta_k), & \text{if } d = 0 \end{cases}$$



Backup

We define recursively the depth-d estimated expected return as

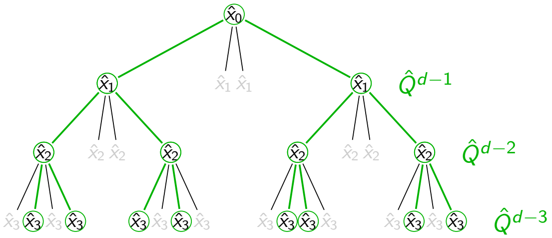
$$\hat{Q}^d(\hat{x}_t, a) = \begin{cases} \rho(\hat{x}_t, a; \theta_\rho) + g(\hat{x}_t, a; \theta_g) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}(\hat{x}_{t+1}, a'), \\ \qquad \text{if } d > 0 \\ Q(\hat{x}_t, a; \theta_k), \text{ if } d = 0 \end{cases}$$



Backup

We define recursively the depth-d estimated expected return as

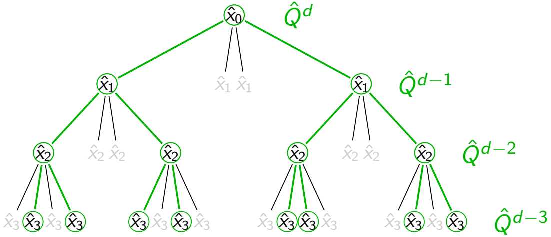
$$\hat{Q}^d(\hat{x}_t, a) = \begin{cases} \rho(\hat{x}_t, a; \theta_\rho) + g(\hat{x}_t, a; \theta_g) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}(\hat{x}_{t+1}, a'), \\ \qquad \text{if } d > 0 \\ Q(\hat{x}_t, a; \theta_k), \text{ if } d = 0 \end{cases}$$



Backup

We define recursively the depth-d estimated expected return as

$$\hat{Q}^d(\hat{x}_t, a) = \begin{cases} \rho(\hat{x}_t, a; \theta_\rho) + g(\hat{x}_t, a; \theta_g) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}(\hat{x}_{t+1}, a'), & \text{if } d > 0 \\ Q(\hat{x}_t, a; \theta_k), \text{ if } d = 0 \end{cases}$$



Backup

Planning - summary

$$\hat{x}_{t'} = \begin{cases} e(s_t; \theta_e), \text{ if } t' = t \\ \hat{x}_{t'-1} + \tau(\hat{x}_{t'-1}, a_{t'-1}; \theta_\tau), \text{ if } t' > t \end{cases}$$

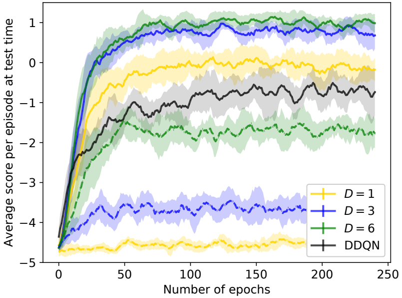
$$\hat{Q}^d(\hat{x}_t, a) = \begin{cases} \rho(\hat{x}_t, a; \theta_\rho) + g(\hat{x}_t, a; \theta_g) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}(\hat{x}_{t+1}, a'), \\ \qquad \text{if } d > 0 \\ Q(\hat{x}_t, a; \theta_k), \text{ if } d = 0 \end{cases}$$

To obtain the action selected at time t, we use a hyper-parameter $D \in \mathbb{N}$ and use a simple sum of the Q-values obtained with planning up to a depth D :

$$Q^D_{plan}(\hat{x}_t, a) = \sum_{d=0}^{D} \hat{Q}^d(\hat{x}_t, a).$$

The optimal action is given by $\underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q^D_{plan}(\hat{x}_t, a)$.

Generalization



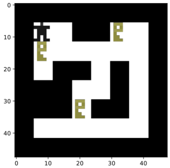


FIGURE – Meta-learning score on a distribution of labyrinths where the training is done with a limited number of transitions obtained by a random policy. 2×10^5 tuples, ~ 500 labyrinths.

More details : *Combined Reinforcement Learning via Abstract Representations, V. Francois-Lavet, Y. Bengio, D. Precup, J. Pineau (AAAI 2019).*

Another important challenge : exploration

- Undirected exploration (e.g. ϵ-greedy)
- Directed exploration
  - When rewards are not sparse, a measure of the uncertainty on the value function can be used ;
  - If sparse rewards or no rewards, some exploration rewards have to be used.

Exploration

Given a point x in representation space, we define a **reward function for novelty** that considers the *sparsity* of states around x - with the average distance between x and its k-nearest-neighbors in its visitation history buffer $\mathcal{B}$:

$$\hat{\rho}_{\mathcal{X}}(x) = \frac{1}{k} \sum_{i=1}^{k} d(x, x_i), \tag{7}$$

where x is a given encoded state, $k \in \mathbb{Z}^+$, $d(\cdot, \cdot)$ is some distance metric in $\mathbb{R}^{n_\mathcal{X}}$ and x_i are the k nearest neighbors (by encoding states in $\mathcal{B}$ to representational space).

More details : *Novelty Search in representational space for sample efficient exploration, D. Tao, V. Francois-Lavet, J. Pineau (NeurIPS 2020).*

Exploration

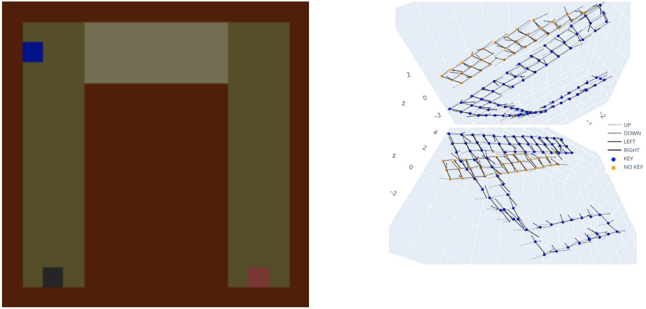


Figure : Multi step environment (left) and the abstract representations of states (right)

Exploration

This technique can also be used for control tasks such as the double pendulum (acrobot), where only intrinsic rewards allows the agent to solve the task.

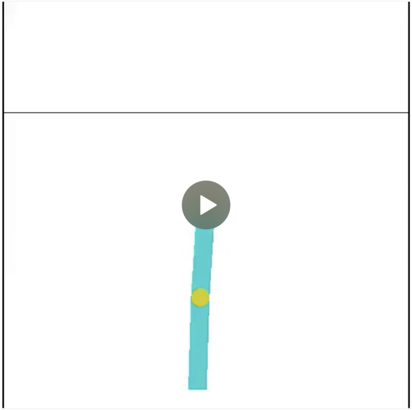
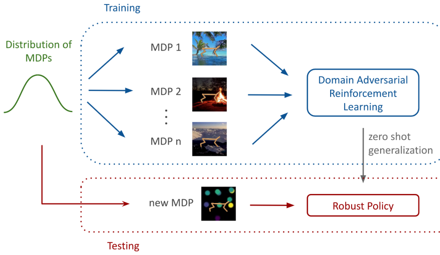


Figure : acrobot.

Transfer learning



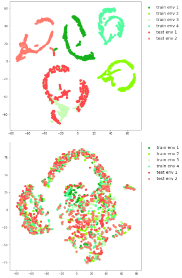


FIGURE – Set up : the agent is trained in a distribution of MDPs and evaluation is done in new domains with unknown backgrounds.

More details : *Domain adversarial reinforcement learning, B. Li, V. Francois-Lavet, T. Doan, J. Pineau (2020).*

Component transfer learning

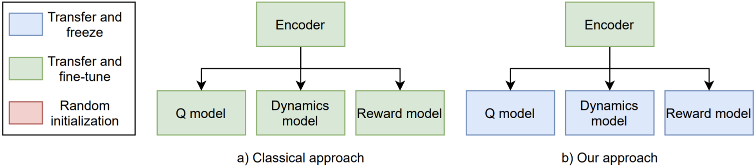


FIGURE – Set up : the agent is trained in a distribution of MDPs and evaluation is done in new domains with unknown backgrounds.

More details : *Component Transfer Learning for Deep RL Based on Abstract Representations, Geoffrey van Driessel, V. Francois-Lavet (2021)*.

A few other challenges for RL (disentanglement of controllable and uncontrollable feature + causal representations)

Disentangled (un-)controllable features

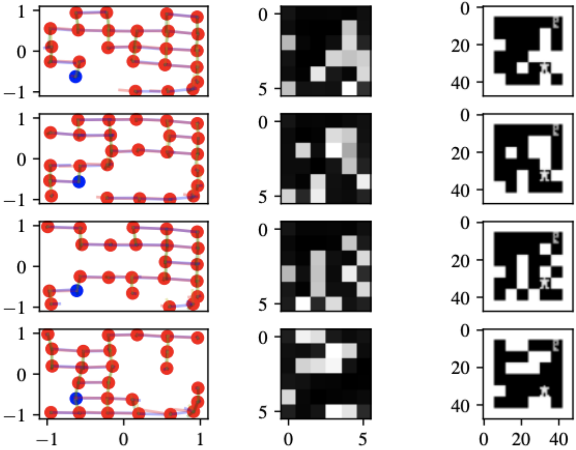


FIGURE – The disentangled latent state representations of four different random maze observations. The left column represents the controllable latent representation. The middle column represents the uncontrollable latent representation and the right column is the original state.

Disentangled (un-)controllable features

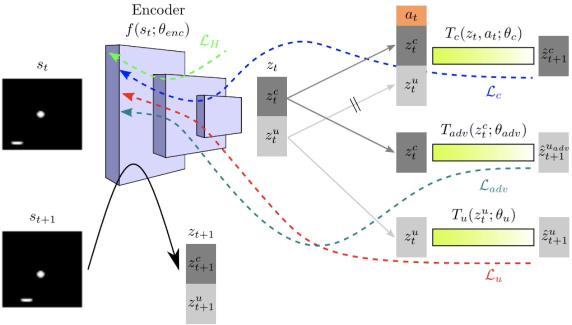


Figure – The disentangled latent state representations of four different random maze observations. The left column represents the controllable latent representation. The middle column represents the uncontrollable latent representation and the right column is the original state.

More details : *Disentangled (Un) Controllable Features. JE Kooi, M Hoogendoorn, V François-Lavet (2022).*

Causality

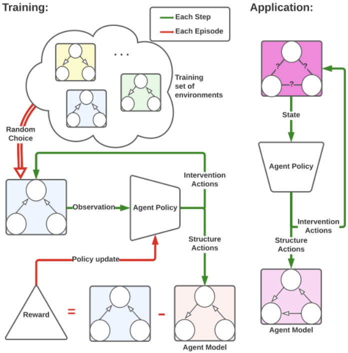


FIGURE – Learning to learn causal graphs.

More details : *A Meta-Reinforcement Learning Algorithm for Causal Discovery. Andreas Sauter, Erman Acar, Vincent François-Lavet (2022).*

Summary of the lecture

- Markov Decision Processes
- Different families of techniques in Reinforcement learning
  - Model-based methods (planning-based techniques)
  - Model-free techniques
- How can we scale with deep learning as a function approximator ?
- Real-world examples using deep RL
- Generalization in deep RL
- Improving generalization
  - Combining model-free and model-based
  - Using self-supervised learning and abstract representations
- Abstract representations for reasoning, exploration and transfer learning
- A few other challenges for RL (disentanglement of controllable and uncontrollable feature + causal representations)

Questions ?