Policy gradient methods in deep RL

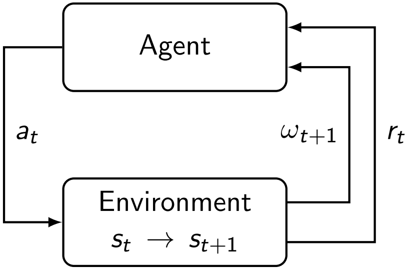
Vincent François-Lavet

Outline

Sequential decision-making and the MDP setting

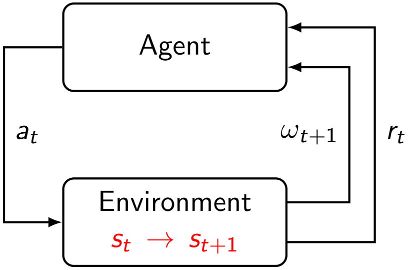
Objective

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



Objective

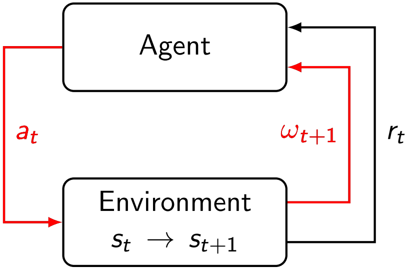
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



transitions
are usually
stochastic

Objective

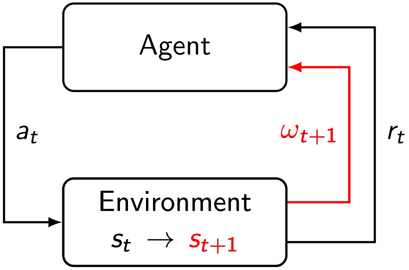
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



Observations and
actions may be
high dimensional

Objective

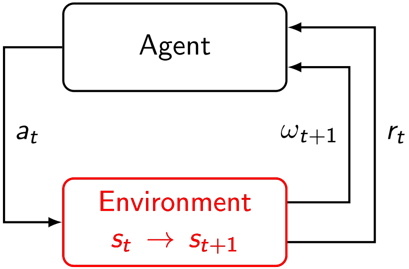
From experience in an environment,
an artificial agent
should be able to **learn** a sequential decision making task
in order **to achieve goals**.



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

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)

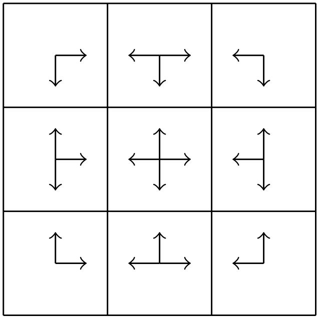
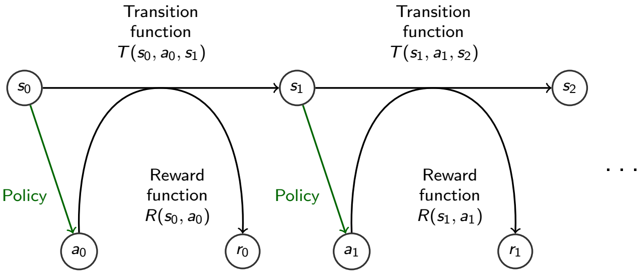


FIGURE – Representation of a (deterministic) mini grid-world with 9 discrete states and 4 discrete actions. The agent is able to move in the four directions, except when the agent is trying to get "out of the grid-world".

Definition of an MDP

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

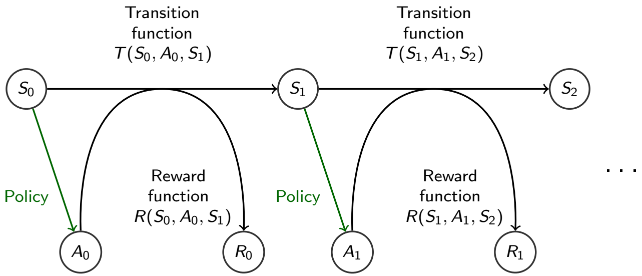
- $\mathcal{S}$ is a finite set of states $\{1, \dots, N_{\mathcal{S}}\}$,
- $\mathcal{A}$ is a finite set of actions $\{1, \dots, N_{\mathcal{A}}\}$,
- $T : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$ is the transition function (set of conditional transition probabilities between states),
- $R : \mathcal{S} \times \mathcal{A} \times \to \mathcal{R}$ 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.



Transition function $T(s_0, a_0, s_1)$
Transition function $T(s_1, a_1, s_2)$
s_0
s_1
s_2
Policy
Reward function $R(s_0, a_0)$
Policy
Reward function $R(s_1, a_1)$
$\dots$
a_0
r_0
a_1
r_1

Definition of an MDP

You can also see the MDP as a computation graph where the nodes are random variables (denoted as upper case letters) that depend on other random variables.


Transition function $T(S_0, A_0, S_1)$
Transition function $T(S_1, A_1, S_2)$
S_0
S_1
S_2
Policy
Reward function $R(S_0, A_0, S_1)$
Policy
Reward function $R(S_1, A_1, S_2)$
$\dots$
A_0
R_0
A_1
R_1

- S_0 represent the distribution of initial states
- The action A_0 depends on S_0 and the policy.
- The reward R_0 depends on S_0 and A_0.
- The next state S_1 depends on S_0 and A_0.
- $\dots$

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., $\mathcal{S} \to \mathcal{A}$) is defined such that

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

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

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

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

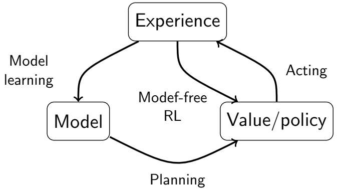
and the optimal policy can be defined as :

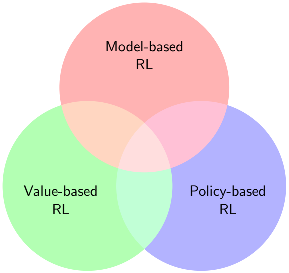
$$\pi^*(s) = \operatorname*{argmax}_{\pi \in \Pi} V^{\pi}(s). \tag{3}$$

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





Why deep RL ?

Motivation

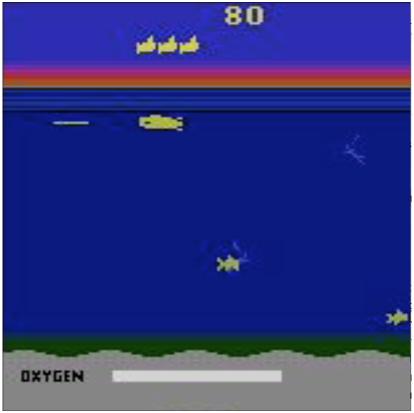
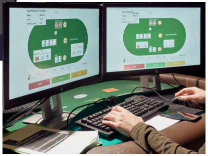
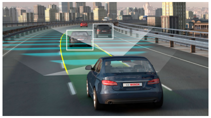
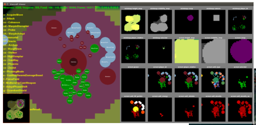


FIGURE – Example of an ATARI game : Seaquest

Motivation : Overview



Motivation

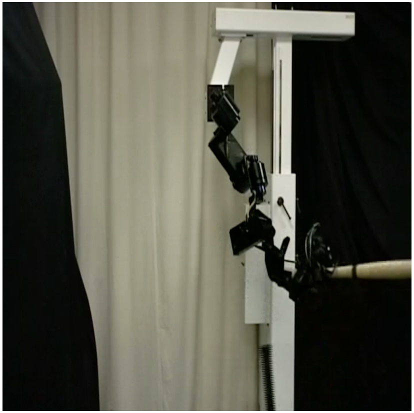


FIGURE – Application in robotics (credits : Jan Peters'team, Darmstadt)

Policy-based methods

Motivation

Motivation for policy based methods

They can represent stochastic policies : $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$.
When is it useful ?

Motivation for policy based methods

They can represent stochastic policies : $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$.
When is it useful ?

- It is useful for building policies that can explicitly explore, and
- this is also useful in multi-agent systems where the Nash equilibrium is a stochastic policy.

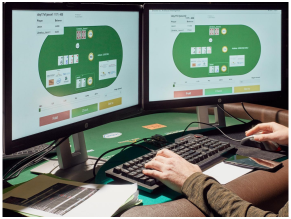


Figure – Super-human performance in heads-up no-limit poker (Libratus and Deepstack)

Different categories of policies

Policies can also be categorized under a second criterion of being either deterministic or stochastic :

- The **deterministic policy** is described by $\pi(s) : \mathcal{S} \to \mathcal{A}$.
- The **stochastic policy** is described by $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(s, a)$ denotes the probability that action a may be chosen in state s.

NB : This concept is very different than the determinist/stochastic nature of the *environment*.

Policy gradient methods

Policy gradient methods optimize a performance objective (typically the expected cumulative reward)

- by finding a good policy (e.g a neural network parameterized policy)
- thanks to variants of stochastic gradient ascent with respect to the policy parameters.

Policy gradient methods belong to a broader class of policy-based methods

- We do not cover evolution strategies. These methods use a learning signal derived from sampling instantiations of policy parameters and the set of policies is developed towards policies that achieve better returns.

Policy-based methods

- Parametrized policies $\Pi = \{\pi_w : w \in \mathbb{R}^n\}$.
- Policy search or gradient ascent on V^{π_w} to improve the policy.

We will see the **stochastic gradient theorems** that provide gradients on the policy parameters in order to optimize the performance objective.

Stochastic Policy Gradient

Policy gradient methods

Find the policy $\pi_w(s, a)$ in the set of parametrized policies $\Pi = \{\pi_w : w \in \mathbb{R}^n\}$ such that :

$$w^* = \text{argmax}_w V^{\pi_w}(s_0)$$

In general, this can thus be seen as an optimization problem and it can be solved with gradient ascent :

$$w_{t+1} = w_t + \eta \nabla_w V^{\pi_w}(s_0)$$

How can we estimate $\nabla_w V^{\pi_w}(s_0)$?

Policy-based methods

The expected return of a stochastic policy π starting from a given state s_0 can be written as

$$V^{\pi}(s_0) = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(s,a) R(s,a) da ds,$$

where $\rho^{\pi}(s)$ is the discounted state distribution defined as

$$\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi\}.$$

Policy-based methods

For a differentiable policy π_w, the fundamental result underlying these algorithms is the policy gradient theorem (see Sutton et al. 2000 for the demonstration) :

$$\nabla_w V^{\pi_w}(s_0) = \int_{\mathcal{S}} \rho^{\pi_w}(s) \int_{\mathcal{A}} \nabla_w \pi_w(s,a) Q^{\pi_w}(s,a) da dx. \qquad (4)$$

- Intuitively, the policy improvement step increases the probability of the actions proportionally to their expected return.
- This result allows us to adapt the policy parameters w : $\Delta w \propto \nabla_w V^{\pi_w}(s_0)$ from experience. NB : the policy gradient does not depend on the gradient of the state distribution !

Policy-based methods

The likelihood ratio trick can be exploited as follows to derive a general method of estimating gradients from expectations :

$$\begin{aligned}\nabla_w \pi_w(s,a) &= \pi_w(s,a)\frac{\nabla_w \pi_w(s,a)}{\pi_w(s,a)} \\ &= \pi_w(s,a)\nabla_w \log(\pi_w(s,a)).\end{aligned} \tag{5}$$

Policy-based methods

Considering Equation 4 and 5, it follows that

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s\sim\rho^{\pi_w}, a\sim\pi_w}\left[\nabla_w \left(\log\ \pi_w(s,a)\right) Q^{\pi_w}(s,a)\right]. \quad (6)$$

Note that, in practice, most policy gradient methods effectively use undiscounted state distributions, without hurting their performance.

Policy-based methods

Two things remain to be done :

- the policy evaluation estimates Q^{π_w}.
- the policy improvement takes a gradient step to optimize the policy $\pi_w(s, a)$ with respect to the value function estimation.

Policy evaluation

How can the agent perform the policy evaluation step, i.e., how to obtain an estimate of $Q^{\pi_w}(s, a)$?

One "simple" option is to estimate $Q^{\pi_w}(s, a)$ from (on-policy) rollouts on the environment while following policy π_w, which is the REINFORCE algorithm (Williams, 1992).

Policy evaluation

How can the agent perform the policy evaluation step, i.e., how to obtain an estimate of $Q^{\pi_w}(s, a)$?

One “simple” option is to estimate $Q^{\pi_w}(s, a)$ from (on-policy) rollouts on the environment while following policy π_w, which is the REINFORCE algorithm (Williams, 1992).

- ✓ The Monte-Carlo estimator is an unbiased well-behaved estimate.
- × However, the main drawback is that the estimate requires on-policy rollouts and can exhibit high variance. Many rollouts are needed.

In practice, a more efficient approach is to instead use an estimate of the return given by a value-based approach $\rightarrow$ actor-critic methods.

Remark 1 : entropy regularizer

To prevent the policy from becoming deterministic, it is common to add an entropy regularizer to the gradient. With this regularizer, the learnt policy can remain stochastic (better behaved and this ensures that the policy keeps exploring).

$$H(\pi_w) = -\sum \pi_w(s) log \pi_w(s)$$

Other options are :

- $KL(\pi_w \parallel \pi_{w_t})$
- $KL(\pi_w \parallel \pi_{w_t})$

Remark 2 : baseline

Using $A^{\pi_w}(s,a) = Q^{\pi_w}(s,a) - V^{\pi_w}(s)$ has usually lower magnitudes than $Q^{\pi_w}(s,a)$. This helps reduce the variance of the gradient estimator $\nabla_w V^{\pi_w}(s_0)$ in the policy improvement step, while not modifying the expectation[1]. In other words, the value function $V^{\pi_w}(s)$ can be seen as a *baseline* or *control variate* for the gradient estimator.

- ✓ Using such a baseline allows for improved numerical efficiency – i.e. reaching a given performance with fewer updates – because the learning rate can be bigger.

1. Indeed, subtracting a baseline that only depends on s to $Q^{\pi_w}(s,a)$ in Eq. 4 does not change the gradient estimator because $\forall s$, $\int_{\mathcal{A}} \nabla_w \pi_w(s,a) da = 0$.

Policy optimization with softmax

In the finite number of actions case, one way to ensure that $\pi_w(s,a)$ stays consistent with a probability distribution over actions is to use at the last layer of a neural network the softmax function that transforms the vector $o(s,a;w)$ into a probability distribution :

$$\pi_w(s,a) = \frac{e^{o(s,a;w)}}{\sum_{a'} e^{o(s,a';w)}}$$

Further topics not covered here

- We have only covered one family of policy-based methods (other possibilities : Deep Deterministic Policy gradient, ...).
- In some specific settings, depending on the loss function and the entropy regularization used, value-based methods and policy-based methods are equivalent. All model-free methods can be seen as different facets of the same family of approaches.

Further resources on policy gradient

- Sutton, R. S., McAllester, D., Singh, S., Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12, 1057-1063.
- Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992) : 229-256.

A few words on model-based approaches

Motivation

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

$$x', r \sim G(x, a)$$

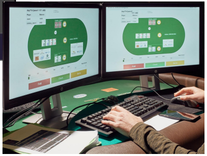
Advantages :

- 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 x).
- it is useful for a sequence of decisions.

Motivation

- 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.



Discussion on model-based methods

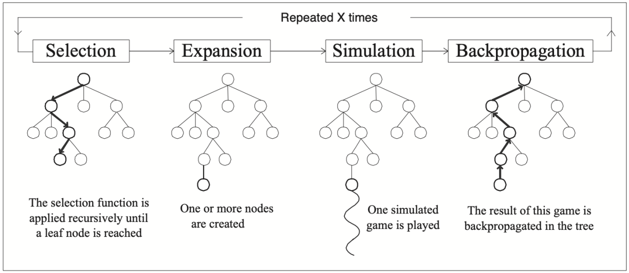
The respective strengths of the model-free versus model-based approaches depend on different factors.

- If the agent does not have access to a generative model of the environment, the learned model will have some inaccuracies.
- Second, a model-based approach requires working in conjunction with a planning algorithm, which is often computationally demanding.
- Third, for some tasks, the model of the environment may be learned more efficiently due to the particular structure of the task.

MCTS with UCT

MCTS with UCT (upper confidence trees) provides a solution for

- exploration/exploitation by selecting the branches to expand in the selection step
- random rollout from the leafs to get an estimate of the win/lose.



Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

FIGURE – Illustration of MCTS (Chaslot et al. 2008).

Conclusions

Summary of the lecture

- Introduction to deep RL
- Introduction to policy-based methods
- Stochastic policy gradient

Questions ?