# Beyond Model-Free Reinforcement Learning

## Solving Optimal Decision Making Problems from a Different Perspective

Matthia Sabatelli

December 17, 2021

# Recap

Last week we have seen ...

## Model-Free Reinforcement Learning

- How to construct algorithms when the state $|\mathcal{S}|$ and action spaces $|\mathcal{A}|$ are large
- Function Approximators
- Linear and Non-Linear Functions
- Deep Reinforcement Learning
- Experience Replay $D = \langle s_t, a_t, r_t, s_{t+1} \rangle$

# Today's Agenda

1. Policy Gradient Methods
2. Beyond Model-Free Learning
3. Beyond Online Learning
4. Beyond Markov Decision Processes

# Policy Gradient Methods

Throughout this course we have constantly seen how important value functions are:

- They correspond to the knowledge of the agent
- They help us understanding the environment
- But most importantly they define the policy $\pi$ that the agent follows

$\Rightarrow$ The most powerful value function is the state-action value function $Q(s, a)$ since:

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

# Policy Gradient Methods

Our goal so far has always been that of learning $Q(s, a)$:

1. We can do this in a tabular fashion
2. Or we can generalize this process with a function approximator
3. We can do this in an *on-policy* or *off-policy* fashion

$\Rightarrow$ First we learn a value function, and then we derive the policy $\pi$

$\Rightarrow$ We have never seen how to learn $\pi$ directly!

# Policy Gradient Methods

Action value based methods such as Q-Learning, SARSA and QV($\lambda$) are very powerful algorithms but have some important limitations:

1. Are restricted to discrete action spaces
2. Work alongside carefully designed exploration-exploitation strategies
3. Are unable to learn an optimal policy which is stochastic
4. Sometimes learning $\pi(a|s;\theta)$ is easier than learning $Q(s,a;\theta)$

# Policy Gradient Methods

There is a family of techniques which tries to learn $\pi(a|s;\theta)$ directly:

**Policy Gradient Methods**

$\Rightarrow$ They learn a parametrized policy that learns how to select actions without having to consult a value function:

$$\pi(a|s;\theta) = \Pr\{a_t = a, s_t = s\,; \theta_t = \theta\}$$

The parameters $\theta$ usually correspond to the weights of a neural network

# Policy Gradient Methods

Training policy gradient methods significantly differs from training a Deep-Q Network:

⇒ The idea of last week's methods was that of minimizing a certain quantity called the TD-error:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle \cdot \rangle \sim U(D)} \left[ \left( r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \right)^2 \right]$$

⇒ Policy Gradient methods instead seek to maximize some scalar performance measure $J(\theta)$ and update the parameters via gradient ascent!

$$\theta_{t+1} = \theta + \alpha \widehat{\nabla J(\theta_t)}$$

# Policy Gradient Methods

It is also possible to learn $\pi(a|s;\theta)$ in combination with a value function!

$\Rightarrow$ These algorithms come with the name of **Actor-Critic** methods:

- It can be hard to learn a policy $\pi$ directly
- We would like to tell our agent learning who is learning $\pi$ how good its policy is
- To do so we can use the state-value function $V^{\pi}(s)$

# Policy Gradient Methods

How do Actor-Critic methods intuitively work?

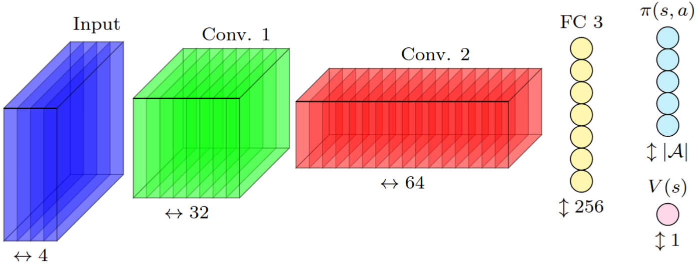


Figure: Image courtesy of Van de Wolfshaar (2017)

# Policy Gradient Methods

How do we train this network?

$$\theta_{t+1} = \theta + \alpha\Big(r_t + \gamma V(s_{t+1};\phi) - V(s_t;\phi)\Big)\frac{\nabla\pi(a_t|s_t;\theta_t)}{\pi(a_t|s_t;\theta_t)}$$
$$= \theta_t + \alpha\delta_t\frac{\nabla\pi(a_t|s_t;\theta_t)}{\pi(a_t|s_t;\theta_t)}$$

⇒ Note that a value function ($\phi$) helps learning the policy parameters $\theta$ but is not required for action selection purposes.

# Policy Gradient Methods

⇒ Nowadays actor-critic algorithms have become very popular in Deep Reinforcement Learning:

1. They are theoretically motivated thanks to the *policy gradient theorem* (see page 325 of the book)
2. The critic can technically learn any value function
3. Can be massively parallelized (see A3C algorithm)

⇒ However, compared to action-value based methods, actor-critic algorithms are less well understood (maybe because learning a policy $\pi$ is still more complex than learning a value function?)

# Beyond Model-Free Learning

$\Rightarrow$ Actor-Critic algorithms, just like action value based methods are also model-free Reinforcement Learning algorithms. As a result we have never even attempted learning the transition function $\mathcal{P}$ of the Markov Decision Process $\mathcal{M}$.

- Recall that the $\mathcal{P}$ and $\Re$ components of $\mathcal{M}$ are usually called the model of the environment
- If they are known we can use Dynamic Programming algorithms like value iteration :)
- However, in the typical Reinforcement Learning scenario this is never the case :(

# Beyond Model-Free Learning

What to do?

- In Model-Based Reinforcement Learning the goal is to learn the model of the environment through experience!
- The idea is to learn a function that comes in the following form: $f(s_t, a_t) = s_{t+1}$
- If learned $f$ would give us $p(s_{t+1}|s_t, a_t)$
- We can do somethinf similar for learning $p(r_t|a_t, s_t)$

$\Rightarrow$ The task of learning a model of the environment corresponds to a supervised learning problem!

# Beyond Model-Free Learning

$\Rightarrow$ The overall learning strategy is very simple:

1. We start with a random policy $\pi(a_t|s_t)$
2. This policy results in a dataset of trajectories $\mathcal{D} = \{(s, a, s')_i\}$
3. We learn the dynamics of the model by minimizing $\sum_i ||f(s_i, a_i) - s_i'||^2$
4. Plan through the model and go back to step 2.

# Beyond Model-Free Learning

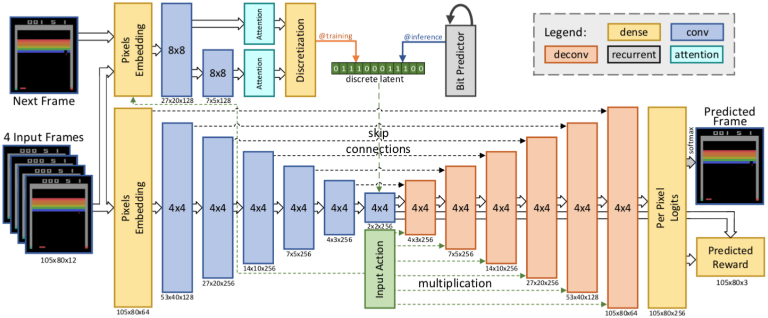


Figure: Image courtesy of Kaiser et al. (2020)

# Beyond Model-Free Learning

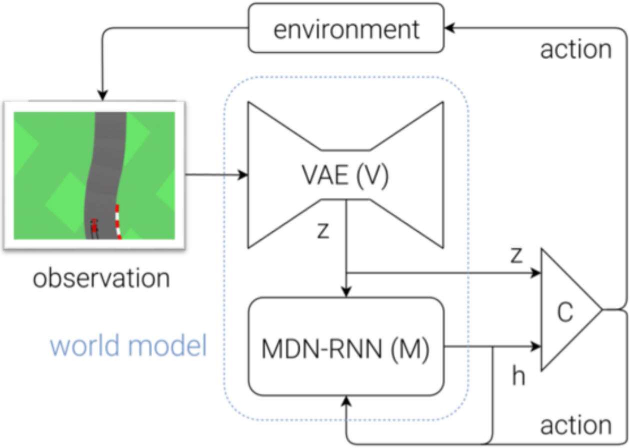


Figure: Image courtesy of Ha et al. (2019)

# Beyond Model-Free Learning

Pros & Cons of Model-Based Reinforcement Learning:

- Some argue that learning $p(s_{t=1}|s_t, a_t)$ is easier than learning $Q^\pi(s, a)$ or $\pi(a|s)$ ✓
- If some dynamics of the environment are known it is easy to include them in the learning process ✓
- As it is essentially supervised learning it scales to a wide range of other applications (GANs and VAEs) ✓
- Is computationally very expensive ✗
- What if the model is wrong? ✗

# Beyond Online Learning

Throughout our lectures we have always assumed that the agent-environmnet interaction:

1. Can last as long as we want (possibly infinite)
2. Is inexpensive
3. Moving from $s_t$ to $s_{t+1}$ is fast
4. We have as many trajectories $\tau\langle s_t, a_t, r_t, s_{t+1}\rangle$ as we want at our disposal for learning

$\Rightarrow$ However, for many practical applications this is not the case!

# Beyond Online Learning

Imagine the following situations:

- Clinical trials
- Self-driving cars
- Drones
- Finance

$\Rightarrow$ These are all examples for which it is very costly to interact with the environment!
We do have access to some data of the environment but not to the environment itself.

# Beyond Online Learning

Sometimes we have to train Reinforcement Learning algorithms with limited data samples

**Batch Reinforcement Learning**

$\Rightarrow$ We have a small dataset of trajectories $\tau\langle s_t, a_t, r_t, s_{t+1}\rangle$, collected by an unknown policy $\pi$ and wish to learn an optimal value function.

The learning agent is not allowed to gather any new experience!

# Beyond Online Learning

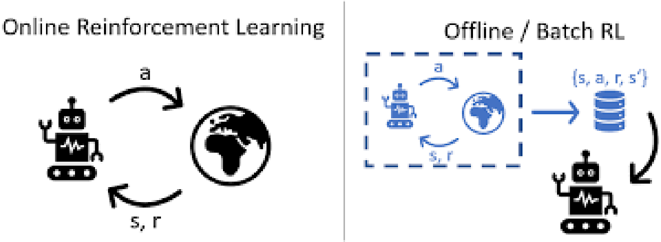


Figure: Image courtesy of Swazinna et al. (2021)

# Beyond Online Learning

⇒ How do we train an agent in the Batch Reinforcement Learning context?

1. We start from a dataset $D$ of trajectories $\tau$ collected by policy $\pi'$
2. We train our favourite model-free algorithm on $D$ until convergence: $Q^{\pi'}(s, a)$
3. We can deploy the learned state-action value function to the environment and hope for the best

Where is the tricky part?

- Our dataset of trajectories is very small ✗
- Do you trust the $Q$ function that you have learned on $D$? ✗
- We can only collect $D$ a few times ✗

# Beyond Online Learning

$\Rightarrow$ Batch Reinforcement Learning poses some very interesting challenges to the community:

1. It forces us to think in practical terms: we do not always have access to a simulation of the environment
2. It questions the reliability of the mathematical tools we are using
3. We need to reason about uncertainty even more: what if $D$ is only representative of a very small portion of $\mathcal{M}$
4. Opens the door to transfer-learning training strategies

# Beyond Markov Decision Processes

We have always modeled the environment as a Markov Decision Process:

$$\mathcal{M}\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r \rangle$$

and assumed that the entire state space $\mathcal{S}$ is visible to the agent.

# Beyond Markov Decision Processes

⇒ However there are many practical applications where it is impossible for an agent to observe some states of the environment

How do we formalize such an environment?

# Beyond Markov Decision Processes

$\Rightarrow$ We generalize the concept of a MDP to a setting where once we have taken action $a_t$ we either can, or cannot observe $s_{t+1}$: Partially Observable Markov Decision Process (POMDP)

$$\mathcal{M}\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \Omega, \mathcal{O}, \gamma \rangle$$

We add two new components to the MDP:

1. $\Omega$ is a set of observations
2. $\mathcal{O}$ are the conditional observation probabilities

$\Omega$ gives us a hint about the state the agent is visiting, but not the true state itself!

# Beyond Markov Decision Processes

When dealing with a POMDP an agent deals with the uncertainty of the state space $\mathcal{S}$ by mantaining a belief $b$ over the visited states:

- $b$ represents a probabilty distribution $\mathcal{P}(\mathcal{S})$
- $b(s)$ denotes the probability $P(\mathcal{S} = s)$ under the current belief state
- Throughout interaction with the environment $b$ is updated

$\Rightarrow$ The goal becomes to learn $\pi^*(b)$

**The End !**

# In Memoriam: Marco A. Wiering (1971-2021)