Deep Reinforcement Learning

Introduction to Approximate Solutions

Jordi Casas Roma

jordi.casas.roma@uab.cat

October 10, 2025

Table of Contents

Table of Contents

Introduction
Preliminaries

In this lesson, we will go one step further in **reinforcement learning** to solve some important problems:

1. Situations in which the state space is too large to be represented in a table.
2. Environments where you work with continuous spaces.

Introduction
Preliminaries

Some examples:

Environment	**Number of States**	**Action-State Pairs**
FrozenLake (8×8)	64	$64 \times 4 = 256$
Chess	10^{120}	!!
Go	10^{170}	!!

Introduction
Preliminaries

Thus, in most **real situations**, we encounter several problems:

- **Storage limitation**: the number of possible states in which the agent can be found is huge. There are too many states and/or action-state pairs to be stored in memory.
- **Computational limitation**: since computational resources are limited, the process of finding the value of each state individually is very slow.
- **Impossibility of knowing all possible states**: it is impossible to know all possible states in advance, since individual states are often not fully observable.
  - Let's imagine, for example, a robot that has a front camera. The agent (robot) will see what the camera focuses on, what is in front of it, but it cannot see through the walls or what is behind it: there are many states that cannot be observed and many that will not even have existed before.

Introduction
Preliminaries

Conclusions:

- It is obvious that in all these cases, the state space can no longer be represented by a table.
- Additionally, we cannot expect to find an optimal policy or value function.
- Tabular methods are no longer useful in these environments, but we can try to find an approximate solution.

Table of Contents

Table of Contents

Introduction
Preliminaries

How do we proceed? (1/3)

1. We need to approximate the value function or policy.
2. The main idea is to combine reinforcement learning with approximation functions (typical of supervised learning) in a process called **generalization**.
   - The goal is to construct a global approximation of the desired function (value function or policy function) by generalizing from some examples of this function.
3. This combination is known as **Reinforcement Learning with approximation functions**.
   - When the approximation function is a neural network, then we talk about **Deep Reinforcement Learning** (DRL).
4. However, the objective remains the same as in the tabular methods.
   - That is, from a given state and environment, we want to find the most optimal action (i.e. the one which gets the maximum reward).

Approximate Solutions
Approximation Functions

How do we proceed? (2/3)

- To solve the problem of very large state spaces, we will adopt an approximate solution based on the attributes (characteristics) of each state.
- Since we will constantly encounter new states (not seen before), we will use the set of attributes of previous states that are similar to the current state to generalize the agent's decisions in unknown states.
- Actually, this is an **estimate of value** in states that have similar characteristics.
- Instead of representing the approximate value function as a table, it will now be a form of parameterized function with a vector of weights $\mathbf{w} \in \mathbb{R}$.

Approximate Solutions
Approximation Functions

How do we proceed? (3/3)

The approximate state value (s) given a vector of weights $\mathbf{w}$ will be described by:

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s) \tag{1}$$

And, the approximate action-state pair value (s, a) given a vector of weights $\mathbf{w}$ will be described by:

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a) \tag{2}$$

- where $\hat{v}(s, \mathbf{w})$ and $\hat{q}(s, a, \mathbf{w})$ are the approximation function.

Table of Contents

Approximate Solutions

Objective Function

Objective Function:

- The objective is to learn those sequences of actions that will allow the agent to achieve its purpose, minimizing or maximizing its objective function:
  - The objective function is $J(\mathbf{w})$ in the case of the approximation of the value function or action-value function,
  - or $J(\theta)$ for the case of the policy approximation.
- We define the objective function as a differential function, with $\mathbf{w}$ (or θ in policy) representing the parameter vector of the problem at hand.
- To update and optimize the value function or policy, there are many optimizers in deep learning, although one of the most used methods is gradient descent or ascent.

Approximate Solutions
Objective Function

Objective Function:

If we apply a gradient to $J(\mathbf{w})$ we obtain:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix} \tag{3}$$

The objective will be to find the local minimum or maximum of $J(\mathbf{w})$. To find the descent or ascent of the gradient, we will move the parameters $\mathbf{w}$ in the direction of negative or positive gradient, respectively:

$$\Delta \mathbf{w} = \pm \frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{4}$$

- where α is the *step-size*, i.e. the number of updates per epoch.
- The process is equivalent for $J(\theta)$, the policy objective function.

Approximate Solutions
Objective Function

Objective Function:

Objective Function

Basically, in RL any optimizer that is capable of receiving a gradient and performing an update can be used in this process.

Even so, we will always use the stochastic version of gradient descent or ascent, since using a single example in each update allows us, in general, to converge faster and reach an optimal value sooner.

Table of Contents

Approximate Solutions

Types of function estimators

Differences with supervised learning:

1. **Correlations**: Successive time steps in an episode are correlated
   - Experiences are not i.i.d. variables (independent and identically distributed variables)
2. **Varying features**: The agent's policy influences the data it receives, altering the nature of the features being learned.
3. **Non-stationarity**: Value functions $v_\pi(s)$ may not be stationary depending on the learning algorithm.
   - For example, in *bootstraping*, the value in the new state is part of the update function, which implies that the update itself is not stationary and can cause problems with (or invalidate) some approximation functions.

Approximate Solutions
Types of function estimators

Types of function estimators:

1. To update the function, in general, we will use the gradient descent/ascent method, so we are interested in the estimator being differentiable.
2. Given these characteristics, the most commonly used estimators that best adapt to most RL problems are **linear functions** and **neural networks** (nonlinear functions).
3. Of course, the choice will depend on the objective, although sometimes it can be difficult to find clear reasoning.
4. In this course, we will use neural networks, following the state of the art in this field.

Table of Contents

Approximate Solutions
Nonlinear function estimator: Neural Networks

Nonlinear function estimator:

1. In **supervised learning**, each instance has a label associated with it (the class to which it belongs) and we obtain as an output a classification probability for each of the possible classes.
2. In **DRL**, on the other hand, the output is the reward that we will obtain by performing a specific action in a specific state.
3. Consequently, in RL the type of neural networks that interest us will be those that allow us to do incremental learning.
   - Every time the agent interacts with the environment, we obtain more data.
   - We are interested in taking into account this data because they are part of the learning process.
4. Both Convolutional (**CNN**) and Recurrent Neural Networks (**RNN**), using backpropagation, allow this type of process.
   - Therefore, both networks can be used in DRL.

Approximate Solutions

Nonlinear function estimator: Neural Networks

We can highlight the **three DRL methods**:

- **Deep Q-Networks**: These methods try to approximate the Q value function (reward function), and return the Q value of all possible actions as output, while minimizing TD errors.
- **Policy Gradients**: Policy gradient methods directly determine the policy by directly maximizing the expected reward using gradient methods (hence the name policy gradient). They are generally *on-policy* methods (i.e. the method learns from the trajectories generated by the current policy).
- **Actor-Critic**: Actor-Critic methods combine policy gradients with value functions, taking advantage of both methods.

Table of Contents

Value Function Approximation

Value function update and optimization

Value function update and optimization

To apply the gradient to value functions, we define the Value function update and optimization $J(\mathbf{w})$ as the expectation $\mathbb{E}$ in a given policy π:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}_\mathbf{w}(s))^2 \right] \tag{5}$$

- where $v_\pi(s)$ is the true value of state s and $\hat{v}_\mathbf{w}(s)$ the estimated value.
- It is essentially an *on-policy* loss function.

Value Function Approximation

Value function update and optimization

Combining the previous equation 5 with the equation 4 we obtain the gradient descent:

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}_{\mathbf{w}}(s)) \nabla_{\mathbf{w}} \hat{v}_{\mathbf{w}}(s) \right]\end{aligned} \tag{6}$$

Value Function Approximation

Value function update and optimization

Combining the previous equation 5 with the equation 4 we obtain the gradient descent:

$$\begin{aligned}\Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_{\pi}\left[(v_{\pi}(s) - \hat{v}_{\mathbf{w}}(s))\nabla_{\mathbf{w}}\hat{v}_{\mathbf{w}}(s)\right]\end{aligned} \tag{6}$$

And the stochastic case:

$$\Delta \mathbf{w} = \alpha \left[v_{\pi}(s) - \hat{v}_{\mathbf{w}}(s)\right] \nabla_{\mathbf{w}}\hat{v}_{\mathbf{w}}(s) \tag{7}$$

Table of Contents

Bibliography

References

Some relevant references:

1. **R. S. Sutton, A. G. Barto**. (2018). *Reinforcement Learning: An Introduction (Second edition)*. MIT Press, Cambridge, MA.
2. **M. Lapan**. (2024). *Deep Reinforcement Learning Hands-On (Third edition)*. Packt Publishing.
3. **A. Bosch, J. Casas, T. Lozano**. (2019). *Deep Learning. Principios y fundamentos*. Universitat Oberta de Catalunya.
4. **M. J. Hausknecht, P. Stone**. (2017). *Deep Recurrent Q-Learning For Partially Observable MDPs*. Austin: University of Texas.