Value Function Approximation

Prof. Alfio Ferrara

Reinforcement Learning

Introduction: Approximate solution methods

RL Tabular methods are based on the idea to have a **limited number** of **well-defined states** in the MDP.

In such cases, we can:

- Find the **optimal value function** $V(s)$ for each of the states
- Find the **optimal policy** π

Approximate solution methods are the methods we can use in RL when it's impossible to find the optimal value function and the optimal policy, but we can only **find a good approximate solution using limited computational resources**.

This happens for two reasons:

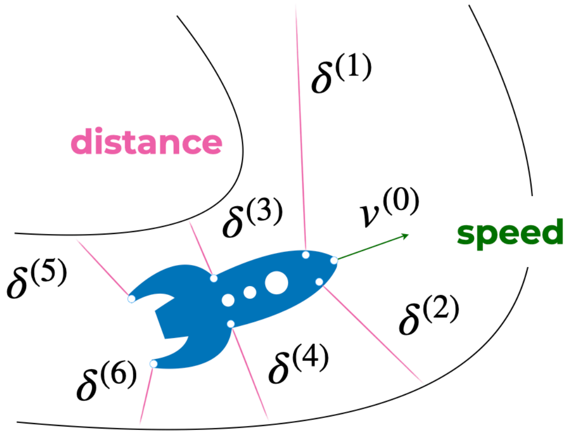
- The **number of states is huge**
- Many states are **similar** but **not identical due to minimal variations in the data featuring them**

The last issue implies that along time **almost every state** encountered **will never have been seen before**.

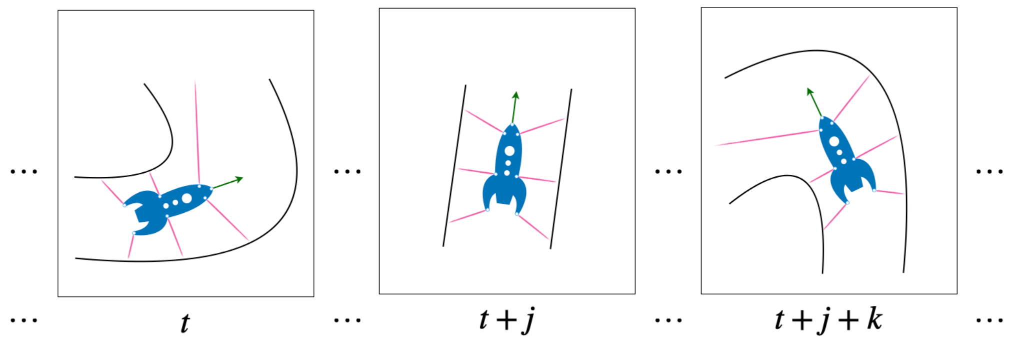
Example

Let's take the example of an **autonomous driving vehicle** moving along a narrow path. The vehicle is equipped with **7 sensors**:

- one for **speed** measured in a **range 0-85 m/s** with precision **0.1**
- 6 for **distance from obstacles**, measured in **meters from 0 to 100** with precision **0.1**



It's easy to see that we may have 850 possible values for seed plus 1000^7 possible combination of distances from obstacles. This means 850×1000^7 states in total. Moreover, we may end up in a high number of states that are **different** but **very similar**.



s_t	s_{t+j}	s_{t+j+k}
22.4	40.1	23.2
100.0	48.2	100.0
60.8	48.4	60.6
26.0	42.1	24.9
34.6	42.8	36.7
28.2	48.2	28.3
26.5	48.4	25.7

State s_t is **very similar** to state s_{t+j+k}, but they are not identical. This implies that most likely when we are at time $t+j+k$ we **have never yet observed** the state s_{t+j+k}. However we could exploit what we have learned from state s_t to evaluate s_{t+j+k}, because they represent almost the same situation.

Feature vectors

The first step towards the generalization of the notion of state is to represent a state as a feature vector $\mathbf{x}(s)$ such that

$$\mathbf{x}(s) = \begin{bmatrix} x_1(s) \\ x_2(s) \\ \dots \\ x_n(s) \end{bmatrix} \tag{1}$$

Featurs may include three different types of knowledge:

1. **Algorithmic state**: data cached by the agent not to represent the current agent situation nor the environment, but just information useful to take algorithmic decisions
2. **Situational state**: the summary of the current agent situation (e.g., its position in space, the internal state of agent's devices such as batteries)

3. **Epistemic state**: the agent current knowledge about the environment (e.g., sensors, environment features)

Linear value function approximation

Now we can set up a set of tools to formalize the goal of approximating the value function $V(s)$ as a linear combination of state features.

Value function for a policy as a weighted combination of features

$$\hat{V}(s; \mathbf{w}) = \sum_{i=1}^{n} x_i(s) w_i = \mathbf{x}(s)^T \mathbf{w} \tag{2}$$

Objective function (we want to learn $\mathbf{w}$)

This is the mean squared error between the true value of state s under policy π and our approximation $\hat{V}(s; \mathbf{w})$.

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[\left(V^\pi(s) - \hat{V}(s; \mathbf{w}) \right)^2 \right] \tag{3}$$

Weight update

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

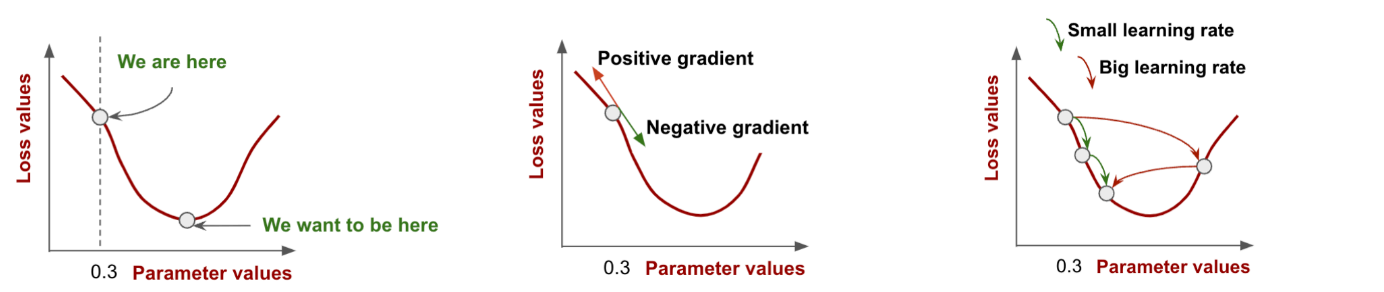
Review of gradient descent as updating method

Since our goal is to minimize $J(\mathbf{w})$ and $J(\mathbf{w})$ is differentiable we can learn how to minimize it by updating the parameters using **gradient descent**. In general, given a differentiable function $f(\mathbf{w})$ we do

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla f(\mathbf{w}_t) \tag{5}$$

where α is the learning rate that specifies how large the update step should be and $\nabla f((w))$ is

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)^T \tag{6}$$



Use of gradients to determine the update direction (i.e., sign)

Use of learning rate to determine amount of the update

In particular, we are interested in **Stochastic Gradient Descent (SGD)** because it does not compute the gradient from the whole dataset but **estimates the gradient from a randomly selected subset of data**, making is feasible to perform the update on-line as RL requires.

With SGD we do not compute an average of several points in the function but just the update for a single point repeated multiple times:

$$\Delta \mathbf{w} = \alpha \left(V^{\pi}(s) - \hat{V}(s; \mathbf{w}) \right) \nabla_{\mathbf{w}} V(s) \tag{7}$$

Model Free VFA for Policy Evaluation

The reason why we cannot simply solve this by a supervised approach is that we do not have the true value $V_\pi(s)$ for any state s. Thus, we want to perform policy evaluation without a model.

Review of Model Free Policy Evaluation

- We assume to have a policy π
- We want to estimate V_π and/or Q_π

The general approach we used was to maintain a look up table of the estimates V_π and/or Q_π by updating them:

- **Monte Carlo**: update estimates after each episode
- **TD methods**: update estimates after each step

VFA variants

What we need to change is the estimate update step including fitting the parameters of VFA.