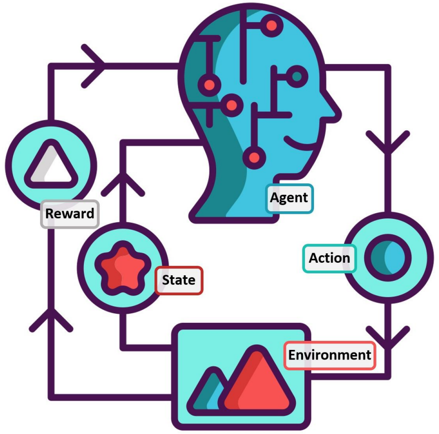
Reinforcement Learning

2025/2026

Lecture #17 Value Function Approximation

Riccardo De Monte
Gian Antonio Susto



The beautiful world of Tabular RL

The beautiful world of Tabular RL

1. Finite number of possible states (or state/action pairs).

The beautiful world of Tabular RL

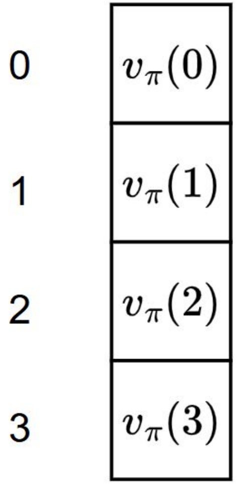
1. Finite number of possible states (or state/action pairs).

2. Reasonable amount of states (or state/action pairs).

The beautiful world of Tabular RL

1. Finite number of possible states (or state/action pairs).

2. Reasonable amount of states (or state/action pairs).

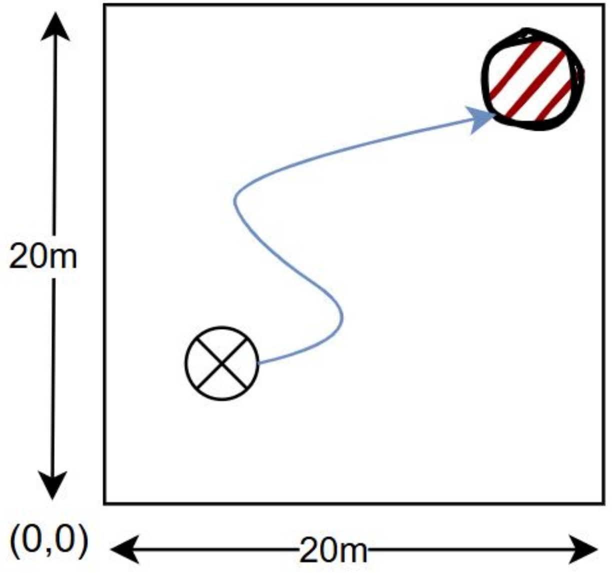


0	$v_\pi(0)$
1	$v_\pi(1)$
2	$v_\pi(2)$
3	$v_\pi(3)$

1. In this case, we can assign to any state an integer index, the row of the table.

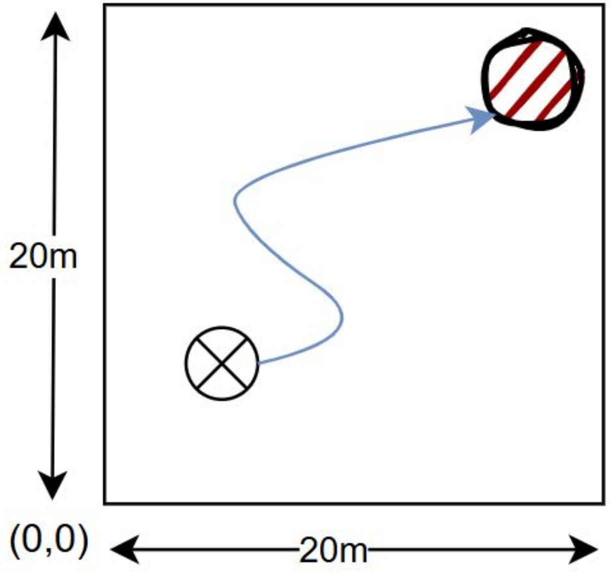
2. The semantics or structure of the state do not matter. Once a state is mapped to an index, the algorithm treats it as an atomic symbol.

Simple example: continuous state space



1. A robot should reach the goal in red.
2. The environment is a room 20 mˆ2.
3. The state is: (x,y) coordinates where x,y in [0,20] .

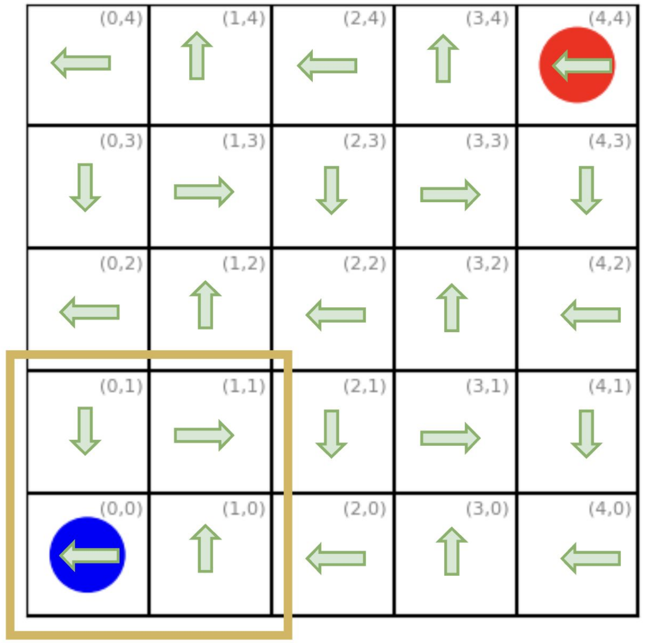
Simple example: continuous state space



1. A robot should reach the goal in red.
2. The environment is a room 20 m^2.
3. The state is: (x,y) coordinates where x,y in [0,20] .

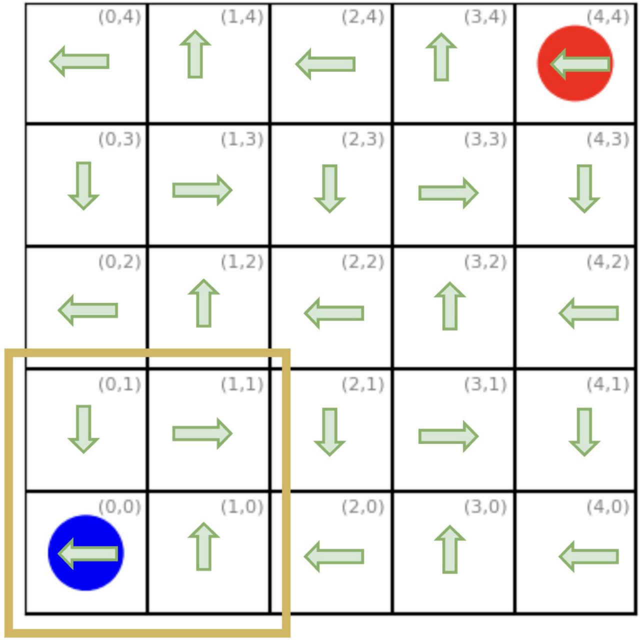
   Too many states -> we cannot assign one integer index to each (x,y)

What about gridworld?



1. We have a "Discrete" set of possible states: finite number of cells.

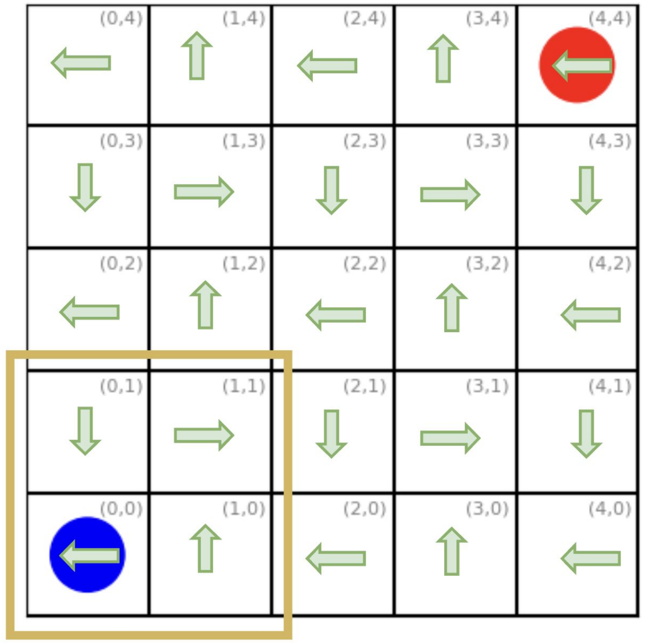
What about gridworld?



1. We have a “Discrete” set of possible states: finite number of cells.

2. We can ideally have one index per cell: Nˆ2 indices.

What about gridworld?

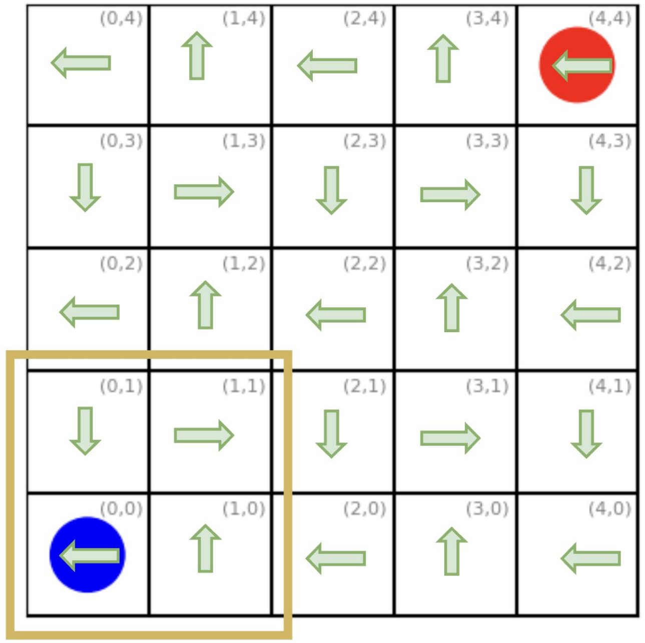


1. We have a “Discrete” set of possible states: finite number of cells.

2. We can ideally have one index per cell: N^2 indices.

3. However, mapping each cell to the corresponding (x,y) (x and y integers), is more reasonable and we have the notion of closeness.

What about gridworld?



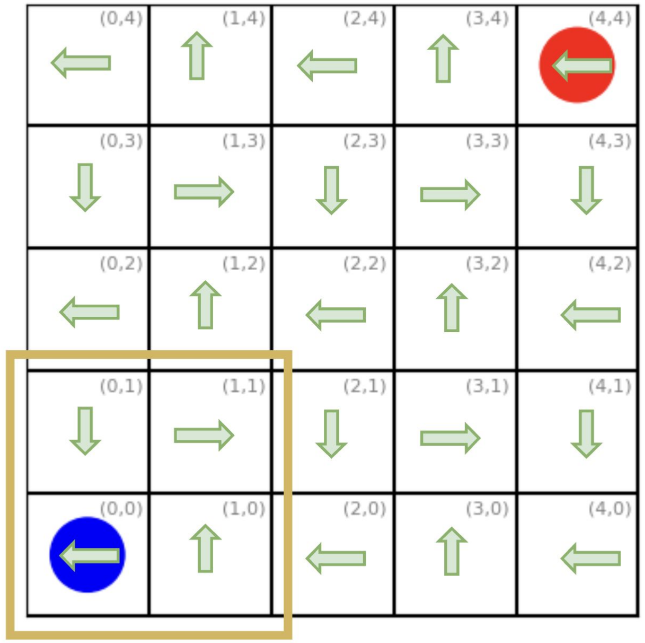
1. We have a “Discrete” set of possible states: finite number of cells.

2. We can ideally have one index per cell: Nˆ2 indices.

3. However, mapping each cell to the corresponding (x,y) (x and y integers), is more reasonable and we have the notion of closeness.

4. Do we need this kind of state for the tabular case?

What about gridworld?



1. We have a “Discrete” set of possible states: finite number of cells.
2. We can ideally have one index per cell: N^2 indices.
3. However, mapping each cell to the corresponding (x,y) (x and y integers), is more reasonable and we have the notion of closeness.
4. Do we need this kind of state for the tabular case? **NO!**

What about chess?

Chess

Number of states ~ 10^{123}

41 zeros

1. We have a “Discrete” set of possible states: finite number states.

What about chess?

Chess

Number of states ~ 10^{123}

41 zeros

1. We have a "Discrete" set of possible states: finite number states.

2. We can still assign an integer index to each state.

What about chess?

Chess

Number of states ~ 10^{123}

41 zeros

1. We have a “Discrete” set of possible states: finite number states.

2. We can still assign an integer index to each state.

3. If we do so, does any mapping have any kind of semantics or structure?

   Maybe it exists a kind of mapping for which close indices maps to “similar” chess configurations. It is quite hard to find one (at least for me).

Two kinds of environment

1. The environment provides “naturally” a numerical representation:
    a. Example 1. Robotics

2. The environment provides an abstract state (no numerical).
    a. Chess: each configuration is a state.
    b. Gridworld: each cell is a state.

The book flow is quite confusing (in my modest opinion)

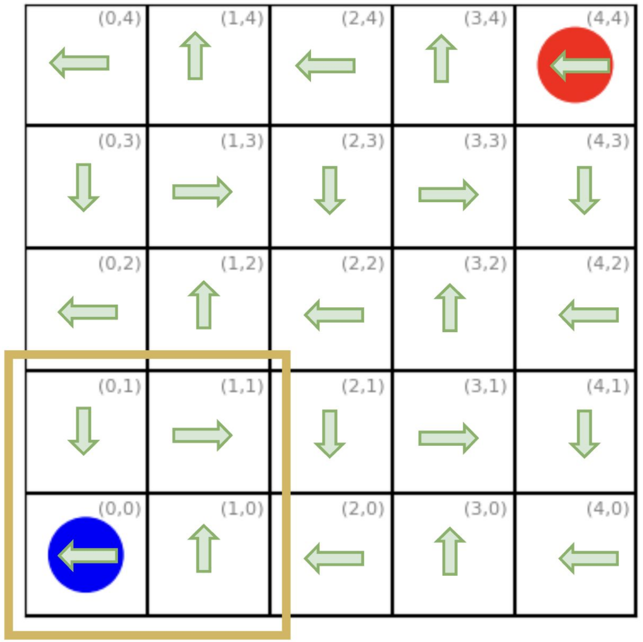
For me, any state s is abstract -> we need a numerical one for function approximators

Feature functions

In general, given any environment we need numerical representation. We aim to either receive it for free or to derive a feature function:

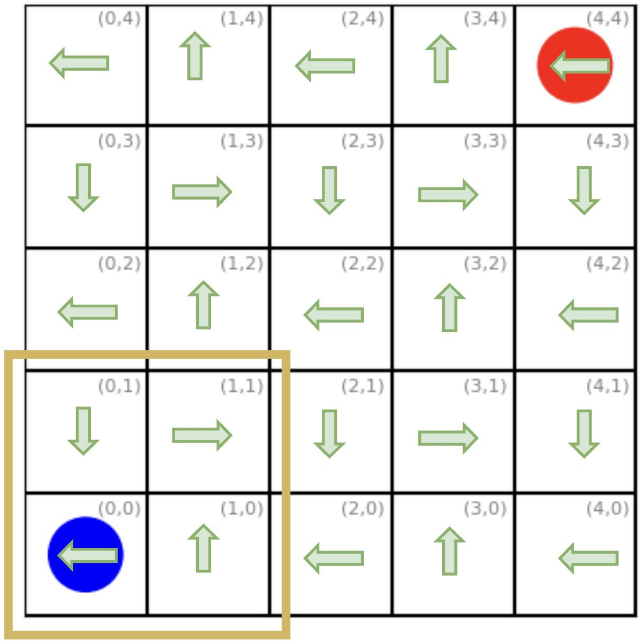
$$X : \mathcal{S} \to \mathbb{R}^d, \quad X(s) = \begin{bmatrix} x_1(s) \\ x_2(s) \\ \vdots \\ v_d(s) \end{bmatrix}$$

Gridworld



We have to derive a suitable X. Two possible ways:

Gridworld: NxN

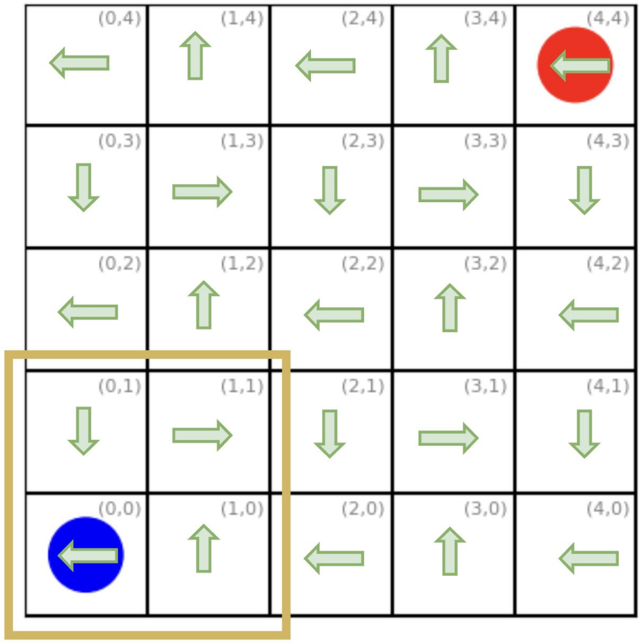


We have to derive a suitable X. Two possible ways:

1. One-hot encoding:

$$\forall i \in \{0, 1, \ldots, N^2 - 1\}$$

Gridworld: NxN



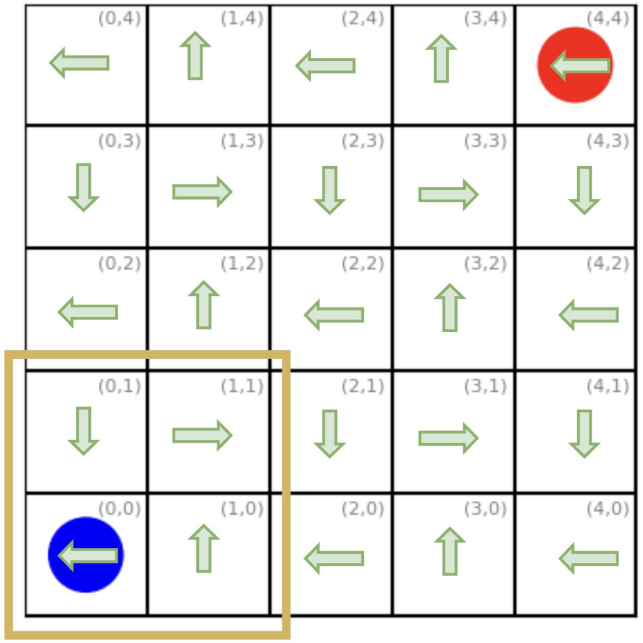
We have to derive a suitable X. Two possible ways:

1. One-hot encoding:

$$\forall i \in \{0, 1, \dots, N^2 - 1\}$$

$$X(i) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{N^2}, \quad x_j(i) = \begin{cases} 1 & i = j \\ 0 & \end{cases}$$

Gridworld: NxN



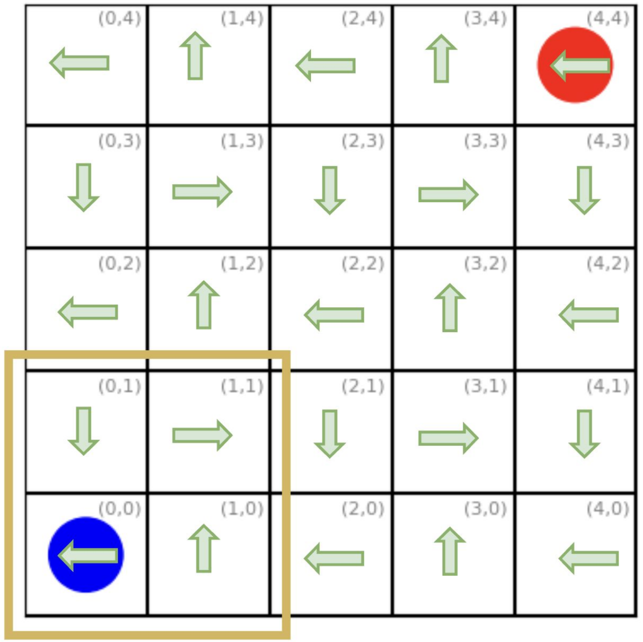
We have to derive a suitable X. Two possible ways:

1. One-hot encoding.

2. (x,y) version:

$$\boldsymbol{X}(\boldsymbol{s}) = \begin{bmatrix} \boldsymbol{x}(\boldsymbol{s}) \\ \boldsymbol{y}(\boldsymbol{s}) \end{bmatrix} \in \mathbb{R}^2$$

Gridworld: NxN



We have to derive a suitable X. Two possible ways:

1. One-hot encoding.

2. (x,y) version:

$$X(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix} \in \mathbb{R}^2$$

This version introduces "closeness"

We have to design X. It is not given, just in some cases it is given for free e.g. robotics.

Tabular RL – Features

Tabular RL and RL with one-hot encoding and linear value function approximation are the same:

$$|\mathcal{S}| = d, \quad X(i) = \begin{bmatrix} x_1(i) \\ x_2(i) \\ \vdots \\ x_d(i) \end{bmatrix} \in \mathbb{R}^d, \quad x_j(i) = \begin{cases} 1 & i = j \\ 0 \end{cases}$$

Tabular RL – Features

Tabular RL and RL with one-hot encoding and linear value function approximation are the same:

$$|\mathcal{S}| = d, \quad X(i) = \begin{bmatrix} x_1(i) \\ x_2(i) \\ \vdots \\ x_d(i) \end{bmatrix} \in \mathbb{R}^d, \quad x_j(i) = \begin{cases} 1 & i = j \\ 0 & \end{cases}$$

$$\boldsymbol{w} \in \mathbb{R}^d, \quad \hat{v}(s, \boldsymbol{w}) = \boldsymbol{w}^\top X(s) = \sum_{j=1}^{d} w_j x_j(s) = w_i$$

Tabular case = assigning one weight to each individual state

1. However, in some cases we cannot assign a different weights to each state:
   a. Robotics example: we cannot assign a weight to each possible (x,y).
   b. Chess: one weight per chessboard configuration.

Tabular case = assigning one weight to each individual state

1. However, in some cases we cannot assign a different weights to each state:
    a. Robotics example: we cannot assign a weight to each possible (x,y).
    b. Chess: one weight per chessboard configuration.

2. We need function approximation.

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s), \qquad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

Last notes about features

For the next slides, $\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s)$, $\hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$

I either assume that the state is already numerical (robotics case) or I consider the feature function as part of $\hat{v}(s, \boldsymbol{w})$

Last notes about features

Important concepts:

1. Since we have less parameters that states, the function approximation should generalize to any state.

Last notes about features

Important concepts:

1. Since we have less parameters that states, the function approximation should generalize to any state.

2. If we design X(s) and we use simple models for $\hat{v}(\boldsymbol{s}, \boldsymbol{w})$ (e.g. linear models), we should ensure some kind of “closeness” structure.

Last notes about features

Important concepts:

1. Since we have less parameters that states, the function approximation should generalize to any state.

2. If we design X(s) and we use simple models for $\hat{v}(\boldsymbol{s}, \boldsymbol{w})$ (e.g. linear models), we should ensure some kind of "closeness" structure.

3. This is why NNs are useful: independently by what we provide as input, NN might learn its own way to represent different states.

Last notes about features

Important concepts:

1. Since we have less parameters that states, the function approximation should generalize to any state.

2. If we design X(s) and we use simple models for $\hat{v}(\boldsymbol{s}, \boldsymbol{w})$ (e.g. linear models), we should ensure some kind of “closeness” structure.

3. This is why NNs are useful: independently by what we provide as input, NN might learn its own way to represent different states.

4. Feature Construction can be done also by coarse coding (other examples in the book).

We need a metric to measure how good is our approximation

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s), \qquad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

Mean Squared Value Error

1. Recall that we have less parameters than states:
    a. We can perfectly match the true value function just in case our function approximation class includes the true one.
    b. In practice this is not the case.

Mean Squared Value Error

1. Recall that we have less parameters than states:
   a. We can perfectly match the true value function just in case our function approximation class includes the true one.
   b. In practice this is not the case.

2. We need to focus on relevant states: states that are likely to be observed under current policy. Here I present a generic notation:

Mean Squared Value Error

1. Recall that we have less parameters than states:
    a. We can perfectly match the true value function just in case our function approximation class includes the true one.
    b. In practice this is not the case.

2. We need to focus on relevant states: states that are likely to be observed under current policy. Here I present a generic notation:

$$\overline{\mathrm{VE}}(w) = \sum_{s \in \mathcal{S}} \mu_\pi(s) \left[v_\pi(s) - \hat{v}(s, w) \right]^2$$

Mean Squared Value Error

1. Recall that we have less parameters than states:
    a. We can perfectly match the true value function just in case our function approximation class includes the true one.
    b. In practice this is not the case.

2. We need to focus on relevant states: states that are likely to be observed under current policy. Here I present a generic notation:

$$\overline{\mathrm{VE}}(w) = \sum_{s \in \mathcal{S}} \mu_\pi(s) \left[v_\pi(s) - \hat{v}(s, w) \right]^2$$

This for prediction!

Episodic case

In episodic tasks, we have that:

$$\mu(s) = \text{fraction of time spent in } s$$

Continuing tasks (no terminal state)

1. We have no terminal states: time step “t” goes to infinity.

Continuing tasks (no terminal state)

1. We have no terminal states: time step “t” goes to infinity.

2. Up to now we have seen the discounted formulation of RL. However, for continuing tasks, discounting might be not reasonable: we should pay more attention to what happen in the long horizon!

Continuing tasks (no terminal state)

1. We have no terminal states:time step “t” goes to infinity.

2. Up to now we have seen the discounted formulation of RL. However, for continuing tasks, discounting might be not reasonable: we should pay more attention to what happen in the long horizon!

3. In continuing tasks with no discount, value function and action-value function are different. For our purposes, we don’t consider this case, but sutton’s book provide details about this formulation (10.3, 10.4 and 10.5).

Continuing tasks (no terminal state)

1. We have no terminal states:time step “t” goes to infinity.

2. Up to now we have seen the discounted formulation of RL. However, for continuing tasks, discounting might be not reasonable: we should pay more attention to what happen in the long horizon!

3. In continuing tasks with no discount, value function and action-value function are different. For our purposes, we don’t consider this case, but sutton’s book provide details about this formulation (10.3, 10.4 and 10.5).

We will consider always the discounted setting

Stationary State distribution for Continuing tasks

1. Once the policy is fixed, any MDP is just a Markov Chain.

Stationary State distribution for Continuing tasks

1. Once the policy is fixed, any MDP is just a Markov Chain.

2. Formally: $$p_\pi(s_{t+1}|s_t) = \sum_{a \in \mathcal{A}} p(s_{t+1}|s_t, a) \cdot \pi(a|s_t)$$

Stationary State distribution for Continuing tasks

1. Once the policy is fixed, any MDP is just a Markov Chain.

2. Formally: $$p_\pi(s_{t+1}|s_t) = \sum_{a\in\mathcal{A}} p(s_{t+1}|s_t, a) \cdot \pi(a|s_t)$$

3. Even if we such a transition distribution, we can define a state distribution at time t:

$$p_{\pi,t}(s') = \sum_{s\in\mathcal{S}} p_\pi(s'|s)p_{\pi,t-1}(s)$$

Stationary State distribution for Continuing tasks

1. Once the policy is fixed, any MDP is just a Markov Chain.
2. Formally:
$$p_\pi(s_{t+1}|s_t) = \sum_{a \in \mathcal{A}} p(s_{t+1}|s_t, a) \cdot \pi(a|s_t)$$
3. Even if we such a transition distribution, we can define a state distribution at time t:
$$p_{\pi,t}(s') = \sum_{s \in \mathcal{S}} p_\pi(s'|s) p_{\pi,t-1}(s)$$
4. What happens when t goes to infinity?

Stationary State distribution for Continuing tasks

Under some assumptions (ergodicity), we have that such distribution converges to the so called **Stationary Distribution**

$$\mu_\pi(s) = \lim_{t \to +\infty} p_{\pi,t}(s)$$

Take home message: both in the episodic and continuing scenarios we have a state distribution induced by the current policy

100 notations for the same thing …

$$\overline{\mathrm{VE}}(w) = \sum_{s \in \mathcal{S}} \mu_\pi(s) \left[v_\pi(s) - \hat{v}(s, w) \right]^2 = \mathbb{E}_{s \sim \mu_\pi} \left[(v_\pi(s) - \hat{v}(s, w))^2 \right]$$

In the ideal world of supervised learning …

Given a dataset with many samples (x,y), we optimize a loss function using Gradient Descent:

$$J(w) = \frac{1}{2}\overline{\mathrm{VE}}(w) = \frac{1}{2}\mathbb{E}_{s\sim\mu_\pi}\left[\left(v_\pi(s) - \hat{v}(s,w)\right)^2\right], \qquad w_{t+1} = w_t - \alpha\nabla_w J(w_t)$$

In RL we have no dataset

1. In RL we don’t have access to any dataset. We can just sample!

In RL we have no dataset

1. In RL we don't have access to any dataset. We can just sample!

2. Stochastic Gradient Descent might help: we approximate the expectation of a gradient with one sample:

$$w_{t+1} = w_t + \alpha(v_\pi(s_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

In RL we have no dataset

1. In RL we don’t have access to any dataset. We can just sample!

2. Stochastic Gradient Descent might help: we approximate the expectation of a gradient with one sample:

$$w_{t+1} = w_t + \alpha(v_\pi(s_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

3. NOTE: we don’t need to know the state distribution!

In RL we have no dataset

1. In RL we don’t have access to any dataset. We can just sample!

2. Stochastic Gradient Descent might help: we approximate the expectation of a gradient with one sample:

$$w_{t+1} = w_t + \alpha(v_\pi(s_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

3. NOTE: we don’t need to know the state distribution!

4. In Machine Learning/Deep Learning we usually assume i.i.d. samples. here instead we have correlated samples. However, SGD still works!

Without oracle

1. Monte Carlo return as target:

$$w_{t+1} = w_t + \alpha(G_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

Without oracle

1. Monte Carlo return as target:

$$w_{t+1} = w_t + \alpha(G_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

2. TD(0) target (biased estimate):

$$w_{t+1} = w_t + \alpha(r_t + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

Without oracle

1. Monte Carlo return as target:

$$w_{t+1} = w_t + \alpha(G_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

2. TD(0) target (biased estimate):

$$w_{t+1} = w_t + \alpha(r_t + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

3. In general we might use any reasonable target:

$$w_{t+1} = w_t + \alpha(U_t - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

TD learning and VFA

1. With TD(0) we have

$$w_{t+1} = w_t + \alpha(r_t + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w\hat{v}(s_t, w_t)$$

TD learning and VFA

1. With TD(0) we have

$$w_{t+1} = w_t + \alpha(r_t + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

2. This is called semi-gradient method since it is not a true gradient.

TD learning and VFA

1. With TD(0) we have

$$w_{t+1} = w_t + \alpha(r_t + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t))\nabla_w \hat{v}(s_t, w_t)$$

2. This is called semi-gradient method since it is not a true gradient.

3. However, we can derive a true one:

$$\delta_t \;=\; r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)$$

$$w_{t+1} \;=\; w_t + \alpha\delta_t(\nabla_w \hat{v}(s_t, w_t) - \gamma\nabla_w \hat{v}(s_{t+1}, w_t))$$

Section 11.5 of the book shows that exact gradient is not our main goal.

Interesting fact

1. We have seen that one-hot encoding and linear value function approximation with many weights as many states is equal to Tabular RL.

$$\hat{v}(s, w) = w^\top X(s), \quad X(s) = [0,\, 0, \cdots, 1, \cdots, 0]^\top$$

Interesting fact

1. We have seen that one-hot encoding and linear value function approximation with many weights as many states is equal to Tabular RL.

$$\hat{v}(s, w) = w^\top X(s), \quad X(s) = [0,\, 0, \cdots, 1, \cdots, 0]^\top$$

2. The update rules we have seen for the Tabular case can be derived by applying SGD in this special case:

$$\nabla \hat{v}(s_t = i, w) = X(i) \implies w_{t+1}[j] = \begin{cases} w_t[j] + \alpha(U_t - w_t[j]) & j == i \\ w_t[j] & j \neq i \end{cases}$$

What about TD(λ) backward view?

TD(λ) backward view with VFA

1. Recall that in the tabular case we have a trace per state.

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) + 1 & s = s_t \\ \lambda\gamma e_{t-1}(s) & s \neq s_t \end{cases}$$

TD(λ) backward view with VFA

1. Recall that in the tabular case we have a trace per state.

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) + 1 & s = s_t \\ \\ \lambda\gamma e_{t-1}(s) & s \neq s_t \end{cases}$$

2. With VFAs instead we have parameters! (less parameters than states)

TD(λ) backward view with VFA

1. Recall that in the tabular case we have a trace per state.

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) + 1 & s = s_t \\ \lambda\gamma e_{t-1}(s) & s \neq s_t \end{cases}$$

2. With VFAs instead we have parameters! (less parameters than states)

3. A reasonable trace definition for VFA: we associate a trace per parameter and we update the trace as follows.

$$z_t = \lambda\gamma z_{t-1} + \nabla_w \hat{v}(s_t, w_t)$$

vettore con un componente per ogni parametro

TD(λ) backward view with VFA

1. Trace update:

$$z_t = \lambda\gamma z_{t-1} + \nabla_w \hat{v}(s_t, w_t)$$

TD(λ) backward view with VFA

1. Trace update:

$$z_t = \lambda\gamma z_{t-1} + \nabla_w \hat{v}(s_t, w_t)$$

2. Value function update:

$$\delta_t \;=\; r_{t+1} + \gamma\hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)$$

$$w_{t+1} \;=\; w_t + \alpha\,\delta_t\, z_t$$

TD(λ) backward view with VFA == Tabular case

1. Recall that in the tabular case we have a trace per state.

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) + 1 & s = s_t \\ \lambda\gamma e_{t-1}(s) & s \neq s_t \end{cases}$$

TD(λ) backward view with VFA == Tabular case

1. Recall that in the tabular case we have a trace per state.

$$e_t(s) = \begin{cases} \lambda\gamma e_{t-1}(s) + 1 & s = s_t \\ \\ \lambda\gamma e_{t-1}(s) & s \neq s_t \end{cases}$$

2. If we have many parameters as many states, we rely on one-hot encoding and we have a linear model:

$$\nabla_w \hat{v}(i, w_t) = X(i), \quad z_{t+1}[j] = \begin{cases} \lambda\gamma z_t[j] + 1 & j == i \\ \lambda\gamma z_t[j] & j \neq i \end{cases}$$

On-Policy Control

We learn q instead of v

1. We need function approximation for q:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s), \qquad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

We learn q instead of v

1. We need function approximation for q:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s), \qquad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

2. Same objective and same SGD update:

$$w_{t+1} = w_t + \alpha(U_t - \hat{q}(s_t, a_t, w_t))\nabla_w \hat{q}(s_t, a_t, w_t), \quad \text{e.g. } U_t = r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t)$$

We learn q instead of v

1. We need function approximation for q:

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s), \qquad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

2. Same objective and same SGD update:

$$w_{t+1} = w_t + \alpha(U_t - \hat{q}(s_t, a_t, w_t))\nabla_w \hat{q}(s_t, a_t, w_t), \quad \text{e.g. } U_t = r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t)$$

3. Policy improvement: epsilon-greedy policy from current q.

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

```
Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R, S'$
        If $S'$ is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., $\varepsilon$-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$
```

Off-Policy Control

Off-Policy control with Action-Value function approximation

1. Use of importance sampling. Example with TD(0):

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \\ w_{t+1} &= w_t + \alpha \rho \delta_t \nabla_w \hat{q}(s_t, a_t, w_t) \end{aligned} \qquad \rho = \frac{\pi(a_{t+1} | s_{t+1})}{b(a_{t+1} | s_{t+1})}$$

Off-Policy control with Action-Value function approximation

1. Use of importance sampling. Example with TD(0):

$$\delta_t = r_{t+1} + \gamma \hat{q}(s_{t+1}, a_{t+1}, w_t) - \hat{q}(s_t, a_t, w_t) \qquad \rho = \frac{\pi(a_{t+1} | s_{t+1})}{b(a_{t+1} | s_{t+1})}$$

$$w_{t+1} = w_t + \alpha \rho \delta_t \nabla_w \hat{q}(s_t, a_t, w_t)$$

2. Q-learning:

$$\delta_t = r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t)$$

$$w_{t+1} = w_t + \alpha \delta_t \nabla_w \hat{q}(s_t, a_t, w_t)$$

The Deadly Triad

The Deadly Triad

The so called **Deadly Triad:**

1. Function-approximation
2. Bootstrapping
3. Off-policy learning

The Deadly Triad

The so called **Deadly Triad:**

1. Function-approximation
2. Bootstrapping
3. Off-policy learning

- Sutton's book provide some example for which we have instability or even divergence (e.g. Q grows a lot or the loss is increasing)

The Deadly Triad

The so called **Deadly Triad:**

1. Function-approximation
2. Bootstrapping
3. Off-policy learning

- Sutton’s book provide some example for which we have instability or even divergence (e.g. Q grows a lot or the loss is increasing).
- When we have all three, learning might be problematic.

The Deadly Triad

The so called **Deadly Triad:**

1. Function-approximation
2. Bootstrapping
3. Off-policy learning

- Sutton's book provide some example for which we have instability or even divergence (e.g. Q grows a lot or the loss is increasing).
- When we have all three, learning might be problematic.
- We can give-up off-policy learning, but it might be necessary for some applications. Moreover many DRL approaches are off-policy: DDPG, SAC, TD3, DQN (many versions), etc.