Data-driven and Learning-based Control

Value-function Reinforcement Learning

Erica Salvato

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Table of Contents

What we know so far?

1 A brief recap

We are now able to solve optimal control problems in the case of dynamics that are:

- Known deterministic (linear or non-linear) or stochastic
  - Dynamic Programming
  - Policy iteration

What we know so far?

1 A brief recap

We are now able to solve optimal control problems in the case of dynamics that are:

- Known deterministic (linear or non-linear) or stochastic
  - Dynamic Programming
  - Policy iteration

Limitation: Update equations (i.e., Bellman equations) require access to the dynamical model

What we know so far?

We are now able to solve optimal control problems in the case of dynamics that are:

- Known deterministic (linear or non-linear) or stochastic
  - Dynamic Programming
  - Policy iteration

  Limitation: Update equations (i.e., Bellman equations) require access to the dynamical model

- Unknown deterministic (linear or non-linear) or stochastic
  - Monte Carlo (MC)
  - Temporal difference (TD)

What we know so far?

We are now able to solve optimal control problems in the case of dynamics that are:

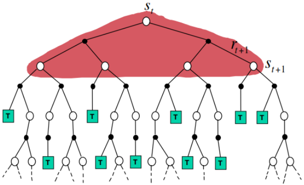
- Known deterministic (linear or non-linear) or stochastic
  - Dynamic Programming
  - Policy iteration

  Limitation: Update equations (i.e., Bellman equations) require access to the dynamical model

- Unknown deterministic (linear or non-linear) or stochastic
  - Monte Carlo (MC)$\rightarrow$ **sampling**
  - Temporal difference (TD) $\rightarrow$ **sampling & bootstrapping**

What we know so far?

Dynamic Programming updates

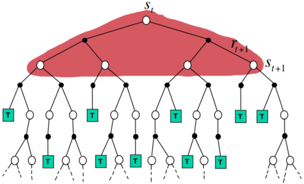


$$V\left(x^{(k)}\right) = \mathbb{E}\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right)\right]$$

What we know so far?

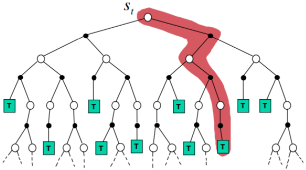
1 A brief recap

Dynamic Programming updates



$$V\left(x^{(k)}\right) = \mathbb{E}\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right)\right]$$

Monte Carlo updates

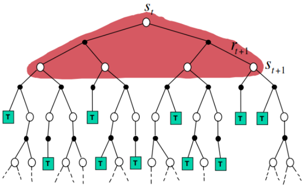


$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[G_k - V\left(x^{(k)}\right)\right]$$

What we know so far?

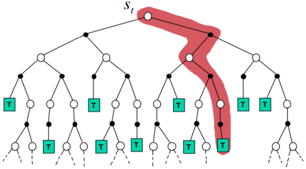
1 A brief recap

Dynamic Programming updates



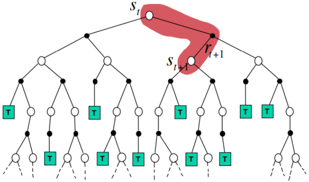
$$V\left(x^{(k)}\right) = \mathbb{E}\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right)\right]$$

Monte Carlo updates



$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[G_k - V\left(x^{(k)}\right)\right]$$

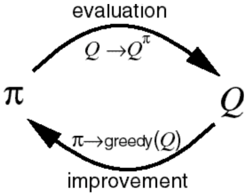
Temporal-difference updates



$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]$$

What we know so far?

The way in which all these approaches solve optimal control problems is through policy iteration.

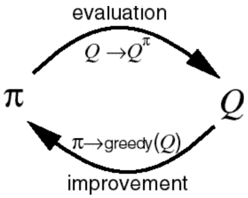


- policy evaluation
- policy improvement

What we know so far?

1 A brief recap

The way in which all these approaches solve optimal control problems is through policy iteration.



- policy evaluation
- policy improvement

They differ:

1. as already mentioned, in their policy evaluation updates,
2. but also in their policy improvement procedure.

What we know so far?

- **Dynamic Programming improvement**

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i} \rightarrow \textbf{Greedy policy}$$

What we know so far?

1 A brief recap

- **Dynamic Programming improvement**

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i} \rightarrow \textbf{Greedy policy}$$

- **MC and TD improvement**

$$\pi\left(u^{(k)} | x^{(k)}\right) = \begin{cases} \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} \neq \arg\max_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \\ 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} = \arg\max_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \end{cases} \rightarrow \epsilon\textbf{-greedy policy}$$

What we know so far?

- **Dynamic Programming improvement**

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i} \rightarrow \textbf{Greedy policy}$$

- **MC and TD improvement**

$$\pi\left(u^{(k)} | x^{(k)}\right) = \begin{cases} \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} \neq \arg\max_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \\ 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} = \arg\max_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \end{cases} \rightarrow \epsilon\textbf{-greedy policy}$$

It guarantees an **exploration-exploitation trade off**

Table of Contents

2 Reinforcement Learning taxonomy

Reinforcement Learning taxonomy

We can classify Reinforcement Learning approaches in:

- **Value-function methods:**
  The policy is implicitly defined via $V(x^{(k)})$ or $Q(x^{(k)}, u^{(k)})$

Reinforcement Learning taxonomy

We can classify Reinforcement Learning approaches in:

- **Value-function methods:**
  The policy is implicitly defined via $V(x^{(k)})$ or $Q(x^{(k)}, u^{(k)})$
- **Policy optimization methods:**
  The policy is a parameterized function whose weights are learned in order to maximize the expected cumulative discounted reward

We can classify Reinforcement Learning approaches in:

- **Value-function methods:**
  The policy is implicitly defined via $V(x^{(k)})$ or $Q(x^{(k)}, u^{(k)}) \rightarrow$ **Critic**

- **Policy optimization methods:**
  The policy is a parameterized function whose weights are learned in order to maximize the expected cumulative discounted reward $\rightarrow$ **Actor**

Reinforcement Learning taxonomy

We can classify Reinforcement Learning approaches in:

- **Value-function methods:**
  The policy is implicitly defined via $V(x^{(k)})$ or $Q(x^{(k)}, u^{(k)}) \rightarrow$ **Critic**
- **Policy optimization methods:**
  The policy is a parameterized function whose weights are learned in order to maximize the expected cumulative discounted reward $\rightarrow$ **Actor**
- **Actor-critic methods:**
  Merging the two ideas by guiding the actor's learning on the basis of the critic's estimated return

Table of Contents

Value-function methods taxonomy

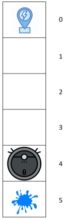
Value function methods assume to work with a discrete compact action set.

Discrete compact set

A set $\mathcal{S}$ is a discrete compact space if it includes h finite number of possible values:

$$\mathcal{S} = \{s_1, s_2, \ldots, s_h\}$$

For example in the robot vacuum cleaner example:



Control input: $u \in \mathcal{U} = \{-1, 1\}$

Value-function methods taxonomy

We can classify Value-function methods depending on the **state set representation**.

In this case, we can distinguish value function methods in:

- Tabular
- Function approximation

Value-function methods taxonomy

We can classify Value-function methods depending on the **state set representation**.

In this case, we can distinguish value function methods in:

- Tabular
- Function approximation

We can classify Value-function methods depending on **how the policy is used in the evaluation/improvement steps**.

In this case, we can distinguish value function methods in:

- On-policy
- Off-policy

Value-function methods taxonomy

We can classify Value-function methods depending on the **state set representation**.

In this case, we can distinguish value function methods in:

- Tabular
- Function approximation

We can classify Value-function methods depending on **how the policy is used in the evaluation/improvement steps**.

In this case, we can distinguish value function methods in:

- On-policy
- Off-policy

On-policy Value-function methods

On-policy means that the evaluation is based on the **current policy**:

$$Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma Q\left(x^{(k+1)}, u^{(k+1)}\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$$

At each iteration:

- we apply the **current policy** producing $u^{(k)}$
- we evaluate the **current policy** by producing $u^{(k+1)}$ applying it
- we update the **current policy** according to its evaluation

On policy Value-function methods

An on-policy algorithm is **SARSA:**

Initialization. $Q\,\forall x \in \mathcal{X}\; u \in \mathcal{U}$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until the terminal condition is met
  - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
  - Select an action $u^{(k+1)}$ with ϵ-greedy policy
  - $Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma Q\left(x^{(k+1)}, u^{(k+1)}\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$
  - Update $x^{(k)} = x^{(k+1)}$, $u^{(k)} = u^{(k+1)}$

Off-policy means that the evaluation is based on the **greedy policy**:

$$Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma \max Q\left(x^{(k+1)}, u\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$$

At each iteration:

- we apply the **current policy** producing $u^{(k)}$
- we evaluate by applying the **greedy policy** $\max_u Q\left(x^{(k+1)}, u\right)$
- we update the **current policy** according to the evaluation thus obtained

Off-policy Value-function methods

An off-policy algorithm is the **Q-Learning:**

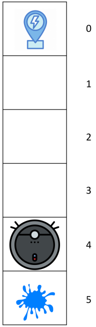
Initialization. $Q \, \forall x \in \mathcal{X} \; u \in \mathcal{U}$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until the terminal condition is met
  - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
  - $Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma \max Q\left(x^{(k+1)}, u\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$
  - Update $x^{(k)} = x^{(k+1)}$,

Example

Consider a robot vacuum cleaner that needs to clean a patch on the floor and also needs to recharge the batteries. Set $\gamma = 0.5$.



- **State:** $x \in \mathcal{X} = \{0, 1, 2, 3, 4, 5\}$
- **Control input:** $u \in \mathcal{U} = \{-1, 1\}$
- **State transition function:** unknown
- **Reward function:**

$$\begin{cases} 5 & \text{if } x^{(k)} \neq 5 \text{ and } x^{(k+1)} = 5 \\ 1 & \text{if } x^{(k)} \neq 0 \text{ and } x^{(k+1)} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Example

1. Apply the **SARSA** algorithm for one episode:
   - starting from $x^{(0)} = 2$
   - with $Q_0 =$

0	**1**	**2**	**3**	**4**	**5**
$0; 0$	$0; 0$	$0; 0$	$0; 0$	$0; 0$	$0; 0$

   - and an ϵ-greedy policy with $\epsilon = 0.5$

1. Apply the **SARSA** algorithm for one episode:
   - starting from $x^{(0)} = 2$
   - with $Q_0 =$

0	**1**	**2**	**3**	**4**	**5**
$0;0$	$0;0$	$0;0$	$0;0$	$0;0$	$0;0$

   - and an ϵ-greedy policy with $\epsilon = 0.5$
2. Apply the **Q-Learning** algorithm for one episode:
   - starting from $x^{(0)} = 2$
   - with $Q_0 =$

0	**1**	**2**	**3**	**4**	**5**
$0;0$	$0;0$	$0;0$	$0;0$	$0;0$	$0;0$

   - and an ϵ-greedy policy with $\epsilon = 0.5$

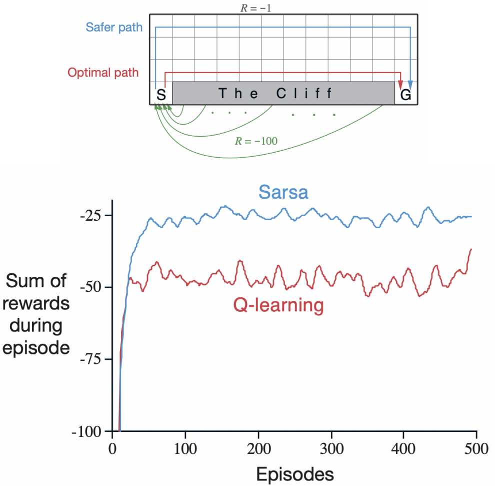
Home exercises

1. Repeat both considering $\epsilon = 0$
2. Repeat both considering $\epsilon = 1$
3. Repeat both considering $\epsilon = \frac{1}{\#\text{of episodes}}$

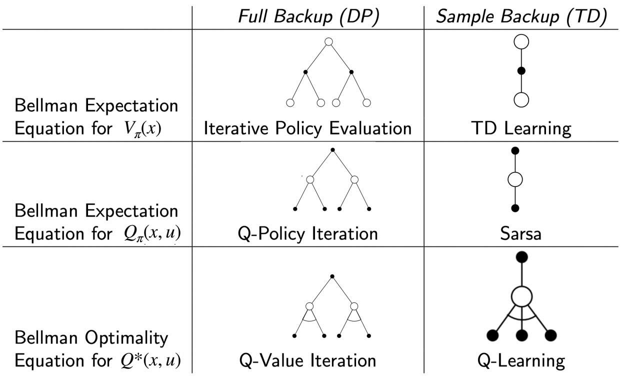
Differences between Sarsa and Q-learning

- SARSA converges to the optimal ϵ-greedy policy, but being on-policy has a better on-line performance

- Q-learning converges to the optimal policy π^* and action-value function Q^* but occasionally fails due to its off-policy nature



Relation between DP and TD



	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $V_\pi(x)$	Iterative Policy Evaluation	TD Learning
Bellman Expectation Equation for $Q_\pi(x, u)$	Q-Policy Iteration	Sarsa
Bellman Optimality Equation for $Q^*(x, u)$	Q-Value Iteration	Q-Learning

Value-function methods taxonomy

We can classify Value-function methods depending on the **state set representation**.

In this case, we can distinguish value function methods in:

- Tabular
- Function approximation

We can classify Value-function methods depending on **how the policy is used in the evaluation/improvement steps**.

In this case, we can distinguish value function methods in:

- On-policy
- Off-policy

Tabular means that the $Q\left(x^{(k)}, u^{(k)}\right)$ representation is a lookup table, i.e., one entry for every state/action pair.

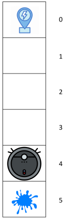
Consequently, it assumes to work with a discrete compact state set.

	u_1	u_2	$\dots$	u_h
x_1	$Q\left(x_1, u_1\right)$	$Q\left(x_1, u_2\right)$	$\dots$	$Q\left(x_1, u_h\right)$
x_2	$Q\left(x_2, u_1\right)$	$Q\left(x_2, u_2\right)$	$\dots$	$Q\left(x_2, u_h\right)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
x_l	$Q\left(x_l, u_1\right)$	$Q\left(x_l, u_2\right)$	$\dots$	$Q\left(x_l, u_h\right)$

Tabular means that the $Q\left(x^{(k)}, u^{(k)}\right)$ representation is a lookup table, i.e., one entry for every state/action pair.

Consequently, it assumes to work with a discrete compact state set.

For example in the robot vacuum cleaner example:



State: $x \in \mathcal{X} = \{0, 1, 2, 3, 4, 5\}$

Control input: $u \in \mathcal{U} = \{-1, 1\}$

Tabular Value-function methods

In large MDPs, a lookup table might be prohibitive:

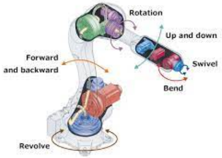
- **Memory demand:** too many actions/states to store
- **Sparsity/Curse of dimensionality:** learning the value of each state/action pair individually might take too long

In large MDPs, a lookup table might be prohibitive:

- **Memory demand:** too many actions/states to store
- **Sparsity/Curse of dimensionality:** learning the value of each state/action pair individually might take too long

Backgammon: 10^{20} states

Go: 10^{170} states



continuous state space

Function Approximation methods

Function approximation means to define an approximate representation of the value function $\hat{V}(x)$ or of the action-value function $\hat{Q}(x, u)$.

We can recognize two main types of approximators:

- **parametric approximators:** given a vector of parameters $\theta \in \mathbb{R}^t$ a parametric approximator maps from the parameters space $\mathbb{R}^t$ to the value-function $\mathcal{V}$ or action-value function space $\mathcal{Q}$

$$\hat{V} : \mathbb{R}^t \to \mathcal{V} \quad \to \quad V(x) \approx \hat{V}_\theta(x)$$

$$\hat{Q} : \mathbb{R}^t \to \mathcal{Q} \quad \to \quad Q(x, u) \approx \hat{Q}_\theta(x, u)$$

- **non-parametric approximators:** typically still have parameters, but the number of parameters, as well as their values, is determined from data

Function Approximation methods

Function approximation allows to represent the V or Q compactly

Function Approximation methods

Function approximation allows to represent the V or Q compactly $\rightarrow$ **less memory demanding**

Function Approximation methods

Function approximation allows to represent the V or Q compactly $\rightarrow$ **less memory demanding**

When function approximation is used, the Q-values of each state influence the Q-values of other nearby states.

Function Approximation methods

Function approximation allows to represent the V or Q compactly $\rightarrow$ **less memory demanding**

When function approximation is used, the Q-values of each state influence the Q-values of other nearby states. $\rightarrow$ **less curse of dimensionality**

Function Approximation methods

Function approximation allows to represent the V or Q compactly $\rightarrow$ **less memory demanding**

When function approximation is used, the Q-values of each state influence the Q-values of other nearby states. $\rightarrow$ **less curse of dimensionality**

If good estimates of Q-values of certain states are available, the algorithm can make reasonable control decisions in the nearby states.

Function Approximation methods

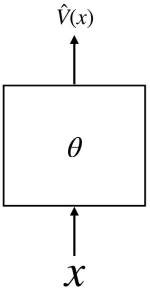
Function approximation allows to represent the V or Q compactly $\rightarrow$ **less memory demanding**

When function approximation is used, the Q-values of each state influence the Q-values of other nearby states. $\rightarrow$ **less curse of dimensionality**

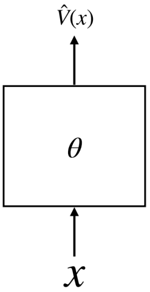
Generalization

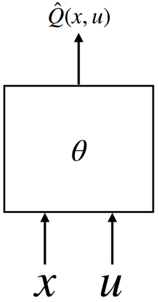
If good estimates of Q-values of certain states are available, the algorithm can make reasonable control decisions in the nearby states.

Parametric functions



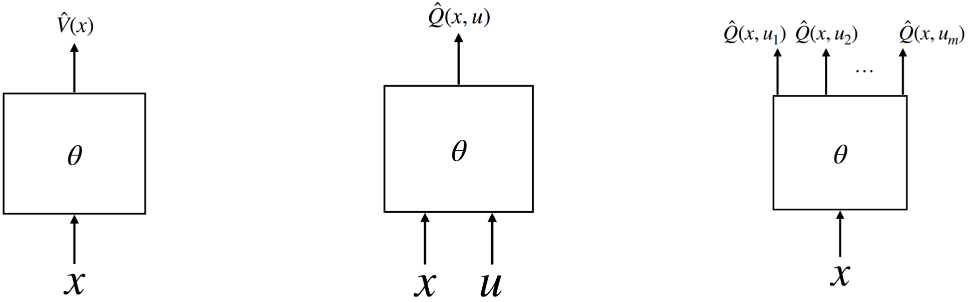
Parametric functions



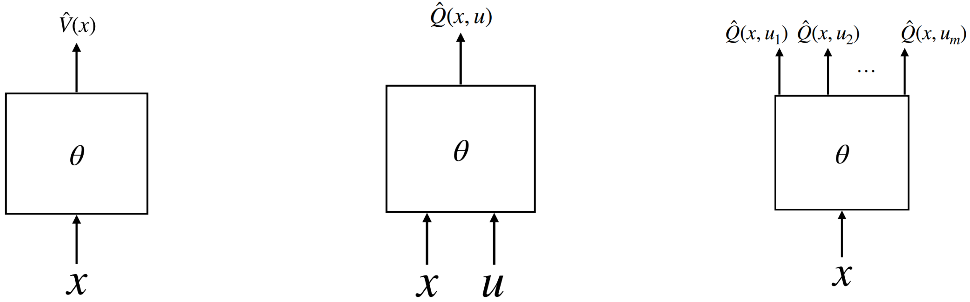


Parametric functions

3 Value-function Reinforcement Learning



Parametric functions



There are many possible function approximators:

- linear function approximation
- neural networks
- ...

Linear function approximation

Basis functions (BF) ϕ are building blocks for creating more complex functions.

In other words, they are a set of h standard functions, combined to estimate another function that is difficult to exactly model.

Examples of BFs are:

- Polinomials
- Fourier Basis
- Radial Basis function

Polynomial Basis Functions

A polynomial basis function is a function defined as a polynomial expression.

The general form of a polynomial basis function given an n-dimensional state vector is given by:

$$\phi_i(x) = \prod_{j=1}^{n} x_j^{c_{i,j}}$$

where:

- x_j is the j-th component of the state vector,
- $c_{i,j}$ is an integer in the set $\{0,\ 1,\ \ldots,\ N\}$ where N is the order of polynomial.

We obtain $h = (1 + N)^d$ basis functions

Fourier Basis Functions

A Fourier basis function is a function defined in terms of cosines.

The general form of a Fourier basis function given an n-dimensional state vector is given by:

$$\phi_i(\boldsymbol{x}) = \cos\left(\pi \boldsymbol{c}_i \cdot \boldsymbol{x}\right)$$

where:

- c_i is a vector of n components whose integer-values belong to $\{0,\ 1,\ \dots,\ N\}$.

We obtain $h = (1 + N)^d$ basis functions

Fourier Basis Functions

A Fourier basis function is a function defined in terms of cosines.

The general form of a Fourier basis function given an n-dimensional state vector is given by:

$$\phi_i(x) = \cos\left(\pi c_i \cdot x\right)$$

where:

- c_i is a vector of n components whose integer-values belong to $\{0,\ 1,\ \ldots,\ N\}$.

We obtain $h = (1 + N)^d$ basis functions

Notice that $c_i \cdot x$ is the element-wise product.

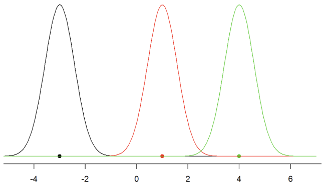
A Radial basis function is a bell-shaped (Gaussian) function.

The general form of a Radial basis function given an n-dimensional state vector is given by:

$$\phi_i(x) = \exp\left(-\frac{||x - c_i||^2}{2\sigma_i^2}\right)$$

where:

- c_i is a vector of n components whose values are the centers of the i-th BF.
- σ_i is a positive real value corresponding to the width of the i-th BF.



Linear function approximation

A linearly parametrized V or Q-function approximator employs h basis function $\phi_1 \ \ldots \ \phi_h : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ and a h-dimensional parameters vector θ.

Approximated V-values are computed as follows:

$$\hat{V}_\theta(x) = \sum_{l=1}^{n} \theta_l^\top \phi_l(x) = \theta^\top \phi(x)$$

where $\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_h(x) \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_h \end{bmatrix}$

While approximated Q-values are computed as follows:

$$\hat{Q}_\theta(x,u) = \sum_{l=1}^{n} \theta_l^\top \phi_l(x,u) = \theta^\top \phi(x,u)$$

where $\phi(x,u) = \begin{bmatrix} \phi_1(x,u) \\ \phi_2(x,u) \\ \vdots \\ \phi_h(x,u) \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_h \end{bmatrix}$

Linear function approximation

Since in Value-function methods we deal with discrete compact action set, assuming that it includes c values $\{u_1, \ldots, u_c\}$, for approximating Q-values we have the following:

$$\phi(x, u_i) = [0, \ldots, 0, \ldots, \phi_1(x), \ldots, \phi_n(x), 0, \ldots, 0]$$

$$u_1 \quad \ldots \qquad\qquad u_i \qquad\qquad u_c$$

and

$$\theta = \begin{bmatrix} \theta_1 & \theta_2 & \ldots & \theta_{h*c} \end{bmatrix}$$

Function Approximation updates

The goal is to find the parameter vector $\theta^* \in \mathbb{R}^t$ that minimizes the **mean-squared error** (MSE) between the estimated value $\hat{V}_\theta(x)$ and the true value $V_\pi(x)$:

$$\mathbb{E}\left[V_\pi(x) - \hat{V}_\theta(x)\right]$$

The MSE measures how far is the approximate value from the exact one.

Function Approximation updates

The goal is to find the parameter vector $\theta^* \in \mathbb{R}^t$ that minimizes the **mean-squared error** (MSE) between the estimated value $\hat{V}_\theta(x)$ and the true value $V_\pi(x)$:

$$\mathbb{E}\left[V_\pi(x) - \hat{V}_\theta(x)\right]$$

The MSE measures how far is the approximate value from the exact one.

The method more frequently used to search for θ^* is the **Stochastic Gradient Descent** (SGD).

Function Approximation updates

The goal is to find the parameter vector $\theta^* \in \mathbb{R}^t$ that minimizes the **mean-squared error** (MSE) between the estimated value $\hat{V}_\theta(x)$ and the true value $V_\pi(x)$:

$$\mathbb{E}\left[V_\pi(x) - \hat{V}_\theta(x)\right]$$

The MSE measures how far is the approximate value from the exact one.

The method more frequently used to search for θ^* is the **Stochastic Gradient Descent** (SGD).

It tries to minimize the MSE, adjusting θ after each step of an episode by a small amount in the direction that would most reduce it:

Function Approximation updates

Gradient

Given a function $f(s)$ of variable s, we denote by $\nabla_s f$ the gradient of the function with respect of s which is defined as follows:

$$\nabla_s f = \frac{\partial f}{\partial s}\vec{s}$$

Gradient

Given a function $f(s)$ of variable s, we denote by $\nabla_s f$ the gradient of the function with respect of s which is defined as follows:

$$\nabla_s f = \frac{\partial f}{\partial s}\vec{s}$$

Therefore, the SGD performs as follows:

$$\begin{aligned}\theta &= \theta - \alpha \nabla_\theta \mathbb{E}\left[V_\pi(x) - \hat{V}_\theta(x)\right] \\ &= \theta + \alpha\left[V_\pi(x) - \hat{V}_\theta(x)\right]\nabla_\theta \hat{V}_\theta(x)\end{aligned}$$

Function Approximation updates

Do we have all the elements for solving this update?

$$\theta = \theta + \alpha \left[V_\pi(x) - \hat{V}_\theta(x) \right] \nabla_\theta \hat{V}_\theta(x)$$

Function Approximation updates

3 Value-function Reinforcement Learning

Do we have all the elements for solving this update?

$$\theta = \theta + \alpha \left[V_\pi(x) - \hat{V}_\theta(x) \right] \nabla_\theta \hat{V}_\theta(x)$$

We don't know the exact value.

Function Approximation updates

3 Value-function Reinforcement Learning

Do we have all the elements for solving this update?

$$\theta = \theta + \alpha \left[V_\pi(x) - \hat{V}_\theta(x) \right] \nabla_\theta \hat{V}_\theta(x)$$

We don't know the exact value.

From Temporal difference we know that the temporal difference error is:

$$r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)$$

Function Approximation updates

Do we have all the elements for solving this update?

$$\theta = \theta + \alpha \left[V_\pi(x) - \hat{V}_\theta(x) \right] \nabla_\theta \hat{V}_\theta(x)$$

We don't know the exact value.

From Temporal difference we know that the temporal difference error is:

$$r^{(k+1)} + \gamma V \left(x^{(k+1)} \right) - V \left(x^{(k)} \right)$$

Therefore, we can substitute the TD target in the θ update thus obtaining

$$\theta = \theta + \alpha \left[r^{(k+1)} + \gamma \hat{V}_\theta \left(x^{(k+1)} \right) - \hat{V}_\theta(x^{(k)}) \right] \nabla_\theta \hat{V}_\theta(x^{(k)})$$

The same intuitions hold when we would like to approximate the action value function.

The **mean-squared error** (MSE) between the estimated $\hat{Q}_\theta(x,u)$ and the true $Q_\pi(x,u)$:

$$\mathbb{E}\left[Q_\pi(x,u) - \hat{Q}_\theta(x,u)\right]$$

The same intuitions hold when we would like to approximate the action value function.

The **mean-squared error** (MSE) between the estimated $\hat{Q}_\theta(x, u)$ and the true $Q_\pi(x, u)$:

$$\mathbb{E}\left[Q_\pi(x, u) - \hat{Q}_\theta(x, u)\right]$$

The SGD updating rule used to search for θ^* is

$$\theta = \theta + \alpha\left[r^{(k+1)} + \gamma\hat{Q}_\theta\left(x^{(k+1)}, u^{(k+1)}\right) - \hat{Q}_\theta\left(x^{(k)}, u^{(k)}\right)\right]\nabla_\theta\hat{Q}\left(x^{(k)}, u^{(k)}\right)$$

SARSA with linear function approximation

Initialization. θ choose $\phi(x, u)$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until the terminal condition is met
  - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
  - Select an action $u^{(k+1)}$ with ϵ-greedy policy
  - $Q\left(x^{(k)}, u^{(k)}\right) = \theta^\top \phi\left(x^{(k)}, u^{(k)}\right)$ and $Q\left(x^{(k+1)}, u^{(k+1)}\right) = \theta^\top \phi\left(x^{(k+1)}, u^{(k+1)}\right)$
  - $\theta = \theta + \alpha\left[r^{(k+1)} + \gamma Q\left(x^{(k+1)}, u^{(k+1)}\right) - Q\left(x^{(k)}, u^{(k)}\right)\right] \nabla_\theta Q\left(x^{(k)}, u^{(k)}\right)$
  - Update $x^{(k)} = x^{(k+1)}$, $u^{(k)} = u^{(k+1)}$

Q-Learning with linear function approximation

Initialization. θ choose $\phi(x, u)$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until the terminal condition is met
    - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
    - $Q\left(x^{(k)}, u^{(k)}\right) = \theta^\top \phi\left(x^{(k)}, u^{(k)}\right)$ and $\forall u$ $Q\left(x^{(k+1)}, u\right) = \theta^\top \phi\left(x^{(k+1)}, u\right)$
    - $\theta = \theta + \alpha\left[r^{(k+1)} + \gamma \max Q\left(x^{(k+1)}, u\right) - Q\left(x^{(k)}, u^{(k)}\right)\right] \nabla_\theta Q\left(x^{(k)}, u^{(k)}\right)$
    - Update $x^{(k)} = x^{(k+1)}$,

Questions’ time!

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE