TTK4215 - Adaptive Control
Introduction to Reinforcement Learning

Norwegian University of Science and Technology (NTNU)

20. November, 2018

Machine Learning

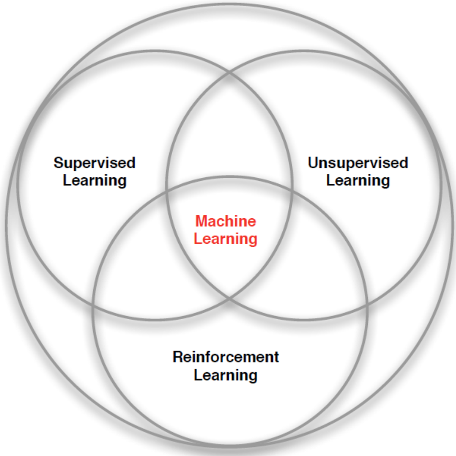
- Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data.

Machine Learning

- Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data.
- Major focus:
  - Automatically learn to recognize patterns.
  - Make intelligent decisions based on data.

Machine Learning

- Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data.
- Major focus:
  - Automatically learn to recognize patterns.
  - Make intelligent decisions based on data.



Supervised Learning

Supervised Learning

1. Learning some function from examples.

Supervised Learning

1. Learning some function from examples.
2. Labeled training data.

Supervised Learning

1. Learning some function from examples.
2. Labeled training data.
3. For every input, the learner is shown what the correct output should be.

Supervised Learning

1. Learning some function from examples.
2. Labeled training data.
3. For every input, the learner is shown what the correct output should be.

$$(x_1, y_1, x_2, y_2, x_3, y_3 ...) \Rightarrow y = f(x)$$

Unsupervised Learning

Unsupervised Learning

1. No teacher or supervisor.

Unsupervised Learning

1. No teacher or supervisor.
2. No correct answers.

Unsupervised Learning

1. No teacher or supervisor.
2. No correct answers.
3. Finding structure hidden in collections of *unlabeled* data, and *model* this structure.

Unsupervised Learning

1. No teacher or supervisor.
2. No correct answers.
3. Finding structure hidden in collections of *unlabeled* data, and *model* this structure.
4. Clustering.

Unsupervised Learning

1. No teacher or supervisor.
2. No correct answers.
3. Finding structure hidden in collections of *unlabeled* data, and *model* this structure.
4. Clustering.

$$(x_1, x_2, x_3, \ldots) \Rightarrow f(x)$$

Reinforcement Learning

Reinforcement Learning

Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

Reinforcement Learning

Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

- No supervisor, only a *reward signal*.

Reinforcement Learning

Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

- No supervisor, only a *reward signal*.
- Feedback is delayed, not instantaneous $\Rightarrow$ temporal credit assignment.

Reinforcement Learning

Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

- No supervisor, only a *reward signal*.
- Feedback is delayed, not instantaneous $\Rightarrow$ temporal credit assignment.
- Learn from *all* actions.

Reinforcement Learning

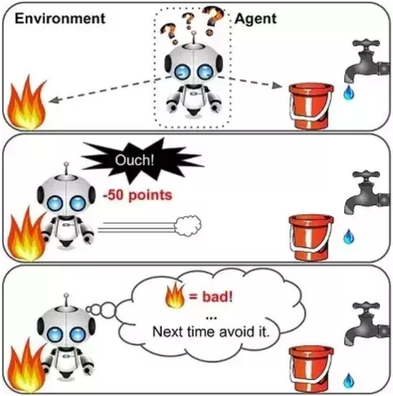
Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

- No supervisor, only a *reward signal*.
- Feedback is delayed, not instantaneous $\Rightarrow$ temporal credit assignment.
- Learn from *all* actions.
- *Exploration* vs *exploitation* tradeoff.
- Inspired by behaviourist psychology.

Reinforcement Learning

Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

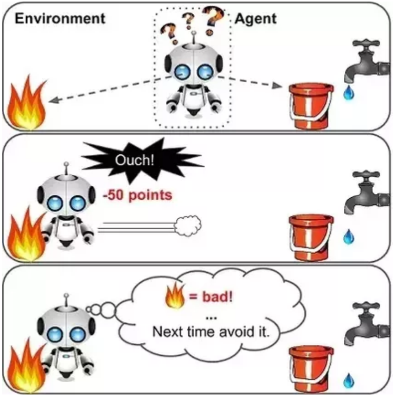
- No supervisor, only a *reward signal*.
- Feedback is delayed, not instantaneous $\Rightarrow$ temporal credit assignment.
- Learn from *all* actions.
- *Exploration* vs *exploitation* tradeoff.
- Inspired by behaviourist psychology.



Reinforcement Learning

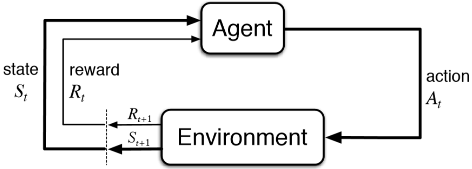
Reinforcement learning (RL): *how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.*

- No supervisor, only a *reward signal*.
- Feedback is delayed, not instantaneous $\Rightarrow$ temporal credit assignment.
- Learn from *all* actions.
- *Exploration* vs *exploitation* tradeoff.
- Inspired by behaviourist psychology.

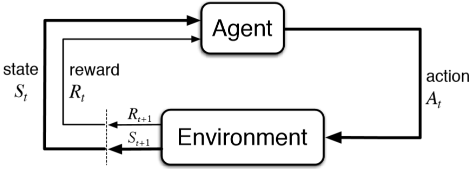


Example Video: Robot walking.

States, Actions and Rewards



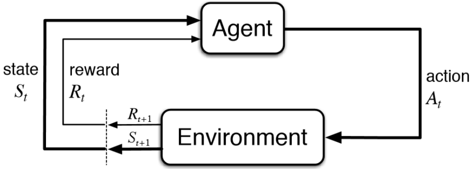
States, Actions and Rewards



Definition

The state is the information (about the environment) used to determine what happens next.

States, Actions and Rewards



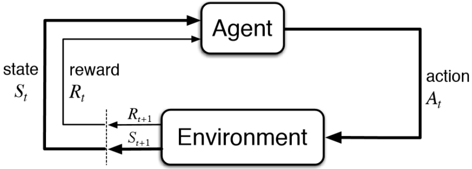
Definition

The state is the information (about the environment) used to determine what happens next.

Definition

An action is what the agent does that affects the environment.

States, Actions and Rewards



Definition

The state is the information (about the environment) used to determine what happens next.

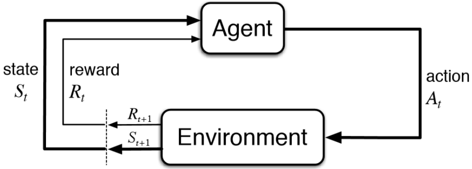
Definition

An action is what the agent does that affects the environment.

Definition

The reward is a measure of how well the agent is doing at step t.

States, Actions and Rewards



Definition

The state is the information (about the environment) used to determine what happens next.

Definition

An action is what the agent does that affects the environment.

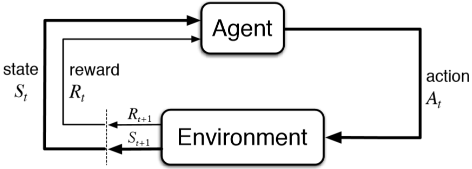
Definition

The reward is a measure of how well the agent is doing at step t.

Hypothesis

All goals can be described by the maximization of expected cumulative reward.

States, Actions and Rewards



Definition

The state is the information (about the environment) used to determine what happens next.

Definition

An action is what the agent does that affects the environment.

Definition

The reward is a measure of how well the agent is doing at step t.

Hypothesis

All goals can be described by the maximization of expected cumulative reward.

The agent's goal in Reinforcement Learning: Maximize total future reward.

States, Actions and Rewards: Examples

Example (Playing Chess)

- State: The organization of pieces on the board
- Action: The agent's (your) move
- Reward: +/- for winning or losing

States, Actions and Rewards: Examples

Example (Playing Chess)

- State: The organization of pieces on the board
- Action: The agent's (your) move
- Reward: +/- for winning or losing

Example (Managing investment portfolio)

- State: The stock market and your own portfolio
- Action: Sell / buy
- Reward: +/- for increase/decrease in portfolio's total worth

States, Actions and Rewards: Examples

Example (Playing Chess)

- State: The organization of pieces on the board
- Action: The agent's (your) move
- Reward: +/- for winning or losing

Example (Managing investment portfolio)

- State: The stock market and your own portfolio
- Action: Sell / buy
- Reward: +/- for increase/decrease in portfolio's total worth

Example (Making a robot walk)

- State: Orientation (and position) of robot and its limbs
- Action: Moving the joints / limbs
- Reward: +/- for forward motion / falling over

Markov Decision Process

How to describe the environment?

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S, A, P, R, γ) consists of:

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S,A,P,R,γ) consists of:

- A finite set of possible states S.

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S, A, P, R, γ) consists of:

- A finite set of possible states S.
- (In each state) a finite set of possible actions A.

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S,A,P,R,γ) consists of:

- A finite set of possible states S.
- (In each state) a finite set of possible actions A.
- $P(s'|s,a)$, the probability of reaching state s' when taking action a in state s.

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S,A,P,R,γ) consists of:

- A finite set of possible states S.
- (In each state) a finite set of possible actions A.
- $P(s'|s,a)$, the probability of reaching state s' when taking action a in state s.
- $R(s,a,s')$, the expected immediate reward of reaching state s' after taking action a in state s.

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S,A,P,R,γ) consists of:

- A finite set of possible states S.
- (In each state) a finite set of possible actions A.
- $P(s'|s,a)$, the probability of reaching state s' when taking action a in state s.
- $R(s,a,s')$, the expected immediate reward of reaching state s' after taking action a in state s.
- Discount factor $0 \leq \gamma \leq 1$, giving less weight on rewards into the future.

Markov Decision Process

How to describe the environment? Markov decision processes(MDPs) formally describe the environment for reinforcement learning.

Definition

A Markov decision process (S,A,P,R,γ) consists of:

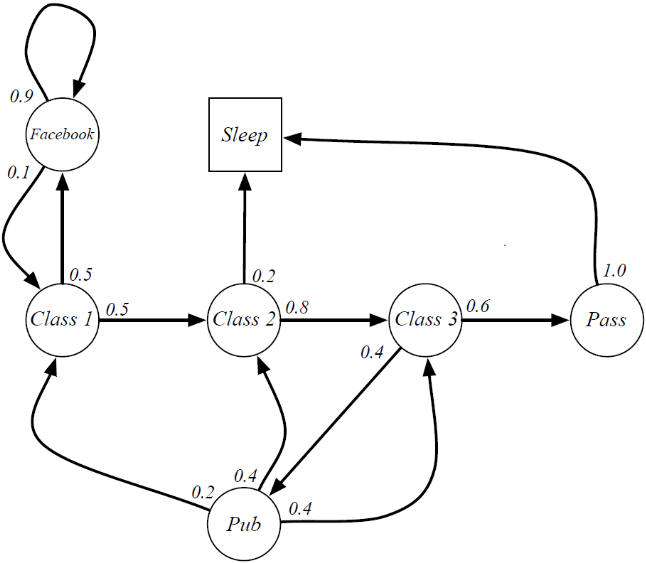
- A finite set of possible states S.
- (In each state) a finite set of possible actions A.
- $P(s'|s,a)$, the probability of reaching state s' when taking action a in state s.
- $R(s,a,s')$, the expected immediate reward of reaching state s' after taking action a in state s.
- Discount factor $0 \leq \gamma \leq 1$, giving less weight on rewards into the future.

A discrete stochastic process, a *Markov chain* with rewards and actions.

Markov Property

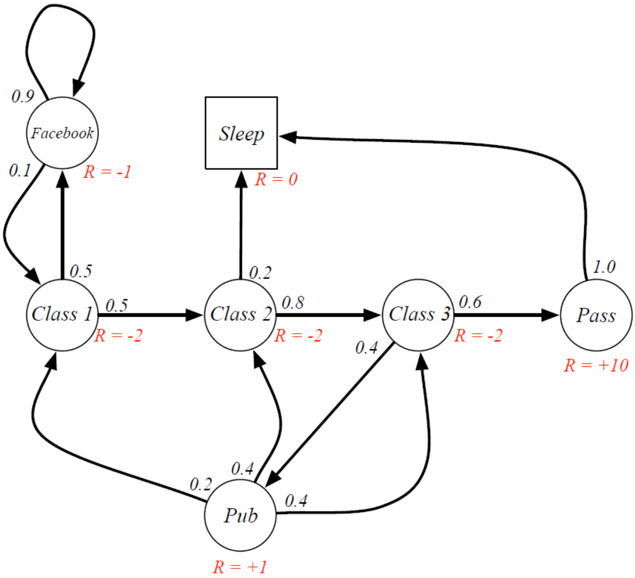
The probability of reaching s' from s dependes only on s and not on the history of earlier states.

Markov Decision Process



(Example from David Silver's "Lecture 2: Markov Decision Processes")

Markov Decision Process



Markov Decision Process

Deterministic or Stochastic

The state transition and reward functions can also be deterministic:

$$s' = \delta(s, a)$$
$$r = R(s, a, s') = R(s, a, \delta(s, a)) \triangleq r(s, a)$$

For now, think about the state transition function/probabilities as a large multi-dimensional table.

Policies

How to describe the agent's desired actions?

Policies

How to describe the agent's desired actions?

Definition

A policy π is the agent's action selection map:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]. \tag{2.1}$$

It is a *probability distribution*.

Policies

How to describe the agent's desired actions?

Definition

A policy π is the agent's action selection map:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]. \tag{2.1}$$

It is a *probability distribution*.

A policy tells you what to do in every state ("master plan").

Policies

How to describe the agent's desired actions?

Definition

A policy π is the agent's action selection map:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]. \tag{2.1}$$

It is a *probability distribution*.

A policy tells you what to do in every state ("master plan").
A RL control problem is the problem of finding an optimal policy.

Policies

How to describe the agent's desired actions?

Definition

A policy π is the agent's action selection map:

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]. \tag{2.1}$$

It is a *probability distribution*.

A policy tells you what to do in every state ("master plan").
A RL control problem is the problem of finding an optimal policy. Optimal in what sense?

Return and value function

Imagine that you are following some policy $\pi(a \mid s)$ producing a state, action, reward sequence $(s, a, r)_t$.

Definition

The **return** G_t is the total future discounted reward from time-step t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2.2}$$

Return and value function

Imagine that you are following some policy $\pi(a \mid s)$ producing a state, action, reward sequence $(s, a, r)_t$.

Definition

The **return** G_t is the total future discounted reward from time-step t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2.2}$$

where

$$\gamma \in [0, 1] \tag{2.3}$$

is the *discount factor*.

Return and value function

Imagine that you are following some policy $\pi(a \mid s)$ producing a state, action, reward sequence $(s, a, r)_t$.

Definition

The **return** G_t is the total future discounted reward from time-step t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2.2}$$

where

$$\gamma \in [0, 1] \tag{2.3}$$

is the *discount factor*.

If $\gamma > 0$: Immediate reward are valued higher than future rewards.

- $\gamma = 0$: "Short-sighted" evaluation
- $\gamma = 1$: "Far-sighted" evaluation

Return and value function

Imagine that you are following some policy $\pi(a \mid s)$ producing a state, action, reward sequence $(s, a, r)_t$.

Definition

The **return** G_t is the total future discounted reward from time-step t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2.2}$$

where

$$\gamma \in [0, 1] \tag{2.3}$$

is the *discount factor*.

If $\gamma > 0$: Immediate reward are valued higher than future rewards.

- $\gamma = 0$: "Short-sighted" evaluation
- $\gamma = 1$: "Far-sighted" evaluation

Definition

The **value function** $V^\pi(s)$ is the expected return starting from state s when following policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]. \tag{2.4}$$

Optimal Policy and Value Function

Definition

The optimal policy $\pi^*(s)$ is the policy that maximizes $V^\pi(s)$ for all states s:

$$\pi^*(s) \triangleq \operatorname{argmax}_\pi V^\pi(s), (\forall s) \tag{2.5}$$

Optimal Policy and Value Function

Definition

The optimal policy $\pi^*(s)$ is the policy that maximizes $V^\pi(s)$ for all states s:

$$\pi^*(s) \triangleq \text{argmax}_\pi V^\pi(s), (\forall s) \tag{2.5}$$

Definition

The optimal value function $V^*(s)$ is the value function of an optimal policy:

$$V^*(s) \triangleq V^{\pi^*}(s) = \max_\pi V^\pi(s), (\forall s) \tag{2.6}$$

This represents the true value of being in state s.

Optimal Policy and Value Function

Definition

The optimal policy $\pi^*(s)$ is the policy that maximizes $V^\pi(s)$ for all states s:

$$\pi^*(s) \triangleq \text{argmax}_\pi V^\pi(s), (\forall s) \tag{2.5}$$

Definition

The optimal value function $V^*(s)$ is the value function of an optimal policy:

$$V^*(s) \triangleq V^{\pi^*}(s) = \max_\pi V^\pi(s), (\forall s) \tag{2.6}$$

This represents the true value of being in state s.

Bellman Equation for $V^*(s)$

We can rewrite a little bit:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.7}$$

Optimal Policy and Value Function

Definition

The optimal policy $\pi^*(s)$ is the policy that maximizes $V^\pi(s)$ for all states s:

$$\pi^*(s) \triangleq \text{argmax}_\pi V^\pi(s), (\forall s) \tag{2.5}$$

Definition

The optimal value function $V^*(s)$ is the value function of an optimal policy:

$$V^*(s) \triangleq V^{\pi^*}(s) = \max_\pi V^\pi(s), (\forall s) \tag{2.6}$$

This represents the true value of being in state s.

Bellman Equation for $V^*(s)$

We can rewrite a little bit:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.7}$$

This is called a **Bellman equation**. Can solve using iterative dynamic programming methods if we know $R(s)$ and $P(s' \mid s, a)$!

Action-Value (Q) Function

Bellman equation for $V^*(s)$:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.8}$$

Action-Value (Q) Function

Bellman equation for $V^*(s)$:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.8}$$

Let's do a little more rewriting..

Definition

The action-value function (Q-function) $Q(s,a)$ is the expected reward of taking action a in state s *plus* the expected return of following the optimal policy $\pi^*(s)$ thereafter:

$$Q(s,a) \triangleq R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \tag{2.9}$$

Action-Value (Q) Function

Bellman equation for $V^*(s)$:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.8}$$

Let's do a little more rewriting..

Definition

The action-value function (Q-function) $Q(s,a)$ is the expected reward of taking action a in state s *plus* the expected return of following the optimal policy $\pi^*(s)$ thereafter:

$$Q(s,a) \triangleq R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \tag{2.9}$$

In the deterministic case, this reduces to:

$$Q(s,a) \triangleq r(s,a) + \gamma V^*(\delta(s,a)) \tag{2.10}$$

Action-Value (Q) Function

Bellman equation for $V^*(s)$:

$$V^*(s) = \max_{a \in A(s)} \left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \right) \tag{2.8}$$

Let's do a little more rewriting..

Definition

The action-value function (Q-function) $Q(s,a)$ is the expected reward of taking action a in state s *plus* the expected return of following the optimal policy $\pi^*(s)$ thereafter:

$$Q(s,a) \triangleq R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s') \tag{2.9}$$

In the deterministic case, this reduces to:

$$Q(s,a) \triangleq r(s,a) + \gamma V^*(\delta(s,a)) \tag{2.10}$$

The following relations hold:

Definition

$$V^*(s) = \max_a Q(s,a) \tag{2.11}$$

$$\pi^*(s) = \text{argmax}_a Q(s,a) \tag{2.12}$$

Action-Value (Q) Function

Another Bellman equation:

Bellman Equation for $Q(s, a)$

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a') \tag{2.13}$$

Action-Value (Q) Function

Another Bellman equation:

Bellman Equation for $Q(s, a)$

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a') \quad (2.13)$$

In the deterministic case, this reduces to:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a') \quad (2.14)$$

Action-Value (Q) Function

Another Bellman equation:

Bellman Equation for $Q(s, a)$

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q(s', a') \tag{2.13}$$

In the deterministic case, this reduces to:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a') \tag{2.14}$$

Recall:

$$\pi^*(s) = \text{argmax}_a Q(s, a) \tag{2.15}$$

This shows that if we learn $Q(s, a)$, we know how to select optimal actions without knowing the system transition model or the reward function! We will make use of this fact later (Q-Learning).

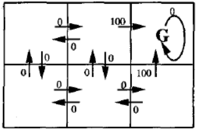
Theorem

For any Markov Decision Process:

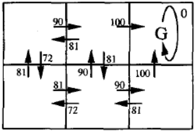
- *There exists an optimal policy $\pi^*(s)$ that is better than or equal to all other policies $\pi(s)$*
- *All optimal policies achieve the optimal value function, $V^{\pi^*}(s) = V^*(s)$*
- *All optimal policies achieve the optimal action-value function, $Q^{\pi^*}(s,a) = Q^*(s,a)$*

- The optimal policy can easily be computed if you know V^* or Q^*.

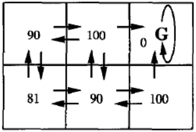
Example



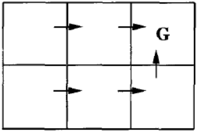
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy
2. Finds the optimal value function by iterating the *Bellman equation* until convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V_k(s') \tag{2.16}$$

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy
2. Finds the optimal value function by iterating the *Bellman equation* until convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V_k(s') \tag{2.16}$$

3. Note that we have to know the reward function and state transition probabilities.

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy
2. Finds the optimal value function by iterating the *Bellman equation* until convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V_k(s') \tag{2.16}$$

3. Note that we have to know the reward function and state transition probabilities.
4. This update step is called a Bellman update.

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy
2. Finds the optimal value function by iterating the *Bellman equation* until convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V_k(s') \tag{2.16}$$

3. Note that we have to know the reward function and state transition probabilities.
4. This update step is called a Bellman update.
5. The sequence V_i generated by value iteration converges to the optimal value function V^*.

Value iteration

Let's look at some dynamic programming based algorithms for solving the MDP.

Value iteration

1. No explicit policy
2. Finds the optimal value function by iterating the *Bellman equation* until convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V_k(s') \tag{2.16}$$

3. Note that we have to know the reward function and state transition probabilities.
4. This update step is called a Bellman update.
5. The sequence V_i generated by value iteration converges to the optimal value function V^*.
6. The optimal policy can be found from the optimal value function V^*.

Policy iteration

Policy iteration

Start with a random policy π.

Policy iteration

Policy iteration

Start with a random policy π.

1. *Evaluate* the policy π

$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.17}$$

Policy iteration

Policy iteration

Start with a random policy π.

1. *Evaluate* the policy π

$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.17}$$

2. *Improve* the policy by acting greedily with respect to V^{π}

$$\pi'(s) = \text{greedy}(V^{\pi}) \tag{2.18}$$

Policy iteration

Policy iteration

Start with a random policy π.

1. *Evaluate* the policy π

$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.17}$$

2. *Improve* the policy by acting greedily with respect to V^{π}

$$\pi'(s) = \text{greedy}(V^{\pi}) \tag{2.18}$$

3. Repeat until convergence.

Policy iteration

Policy iteration

Start with a random policy π.

1. *Evaluate* the policy π
$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.17}$$

2. *Improve* the policy by acting greedily with respect to V^{π}
$$\pi'(s) = \text{greedy}(V^{\pi}) \tag{2.18}$$

3. Repeat until convergence.

This process of *policy iteration* always converges to the optimal policy π^*.

Policy iteration

Policy iteration

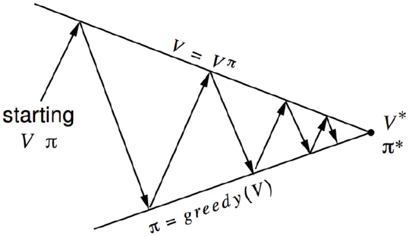
Start with a random policy π.

1. *Evaluate* the policy π
$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s] \tag{2.17}$$

2. *Improve* the policy by acting greedily with respect to V^{π}
$$\pi'(s) = \text{greedy}(V^{\pi}) \tag{2.18}$$

3. Repeat until convergence.

This process of *policy iteration* always converges to the optimal policy π^*.

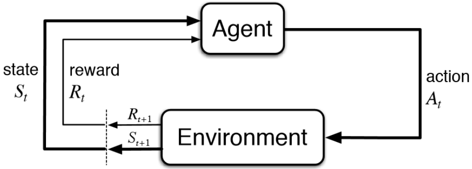


Learning to Act In an Unknown Environment

In the previous section, when doing value iteration and policy iteration to solve the MDP, the state transition function and the reward function where *known*.

Learning to Act In an Unknown Environment

In the previous section, when doing value iteration and policy iteration to solve the MDP, the state transition function and the reward function where *known*.



What if the system model and reward is unknown?

What are we going to learn?

Different Agent Designs

- Model-based methods: Learn transition and reward function. Then solve MDP.

What are we going to learn?

Different Agent Designs

- Model-based methods: Learn transition and reward function. Then solve MDP.
- Model-free (value based): Instead of learning the model, learn some value function instead.

What are we going to learn?

Different Agent Designs

- Model-based methods: Learn transition and reward function. Then solve MDP.
- Model-free (value based): Instead of learning the model, learn some value function instead.
- Policy search. Learn policy directly.

What are we going to learn?

Different Agent Designs

- Model-based methods: Learn transition and reward function. Then solve MDP.
- Model-free (value based): Instead of learning the model, learn some value function instead.
- Policy search. Learn policy directly.

We are going to focus on a *model-free* method, called *Q-Learning* that learns an action-value representation $\hat{Q}(s, a)$ of $Q(s, a)$.

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$
- Observe resulting reward r and state s'

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$
- Observe resulting reward r and state s'
- Update Q-value:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)) \tag{3.3}$$

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$
- Observe resulting reward r and state s'
- Update Q-value:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t (r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)) \tag{3.3}$$

- $s \leftarrow s'$

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$
- Observe resulting reward r and state s'
- Update Q-value:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)) \tag{3.3}$$

- $s \leftarrow s'$
- Repeat forever

What action to take?

Q-Learning

Recall the Bellman equations for Q:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.1}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a') \tag{3.2}$$

Q-learning Algorithm

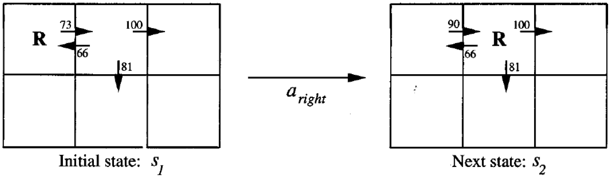
- For each s, a, initialize the table entry $\hat{Q}(s,a)$ to arbitrary values (e.g. zero).
- Take an action: $a \sim \mu$
- Observe resulting reward r and state s'
- Update Q-value:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)) \tag{3.3}$$

- $s \leftarrow s'$
- Repeat forever

What action to take? Exploration vs exploitation. E.g. act greedy, but leave some probability to explore (random action).

Illustration



$$
\begin{aligned}
\hat{Q}(s_1, a_{right}) &\leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \\
&\leftarrow 0 + 0.9 \ \max\{66, 81, 100\} \\
&\leftarrow 90
\end{aligned}
$$

Convergence of Q-Learning

Theorem (Watkins and Dayan (1992))

Q-learning control converges to the optimal action-value function Q^*.

Convergence of Q-Learning

Theorem (Watkins and Dayan (1992))

Q-learning control converges to the optimal action-value function Q^*.

Assumptions

- Bounded rewards.
- Every state-action pair is visited infinitely often.
- Conditions on α_t

Might need *a lot* of iterations to converge!

Approximate methods

So far: $V(s)$ or $Q(s,a)$ stored as a look-up table.

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

$\Rightarrow$ Too many states for a look-up table.

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

$\Rightarrow$ Too many states for a look-up table.

Value Function Approximation

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

$\Rightarrow$ Too many states for a look-up table.

Value Function Approximation

Approximate v and/or q using function approximations

$$V(s) \approx \hat{V}(s, \theta) \qquad Q(s, a) \approx \hat{Q}(s, a, \theta) \tag{3.4}$$

Approximate methods

So far: $V(s)$ or $Q(s, a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

$\Rightarrow$ Too many states for a look-up table.

Value Function Approximation

Approximate v and/or q using function approximations

$$V(s) \approx \hat{V}(s, \theta) \qquad Q(s, a) \approx \hat{Q}(s, a, \theta) \tag{3.4}$$

Different approximation methods:

- Linear combinations of features
- Neural network ("deep" methods)
- Decision tree
- Nearest neighbor
- Fourier / wavelet bases
- ...and many more

Approximate methods

So far: $V(s)$ or $Q(s,a)$ stored as a look-up table.

What if the state space is too large?

- Backgammon: 10^{20} states
- Chess: 10^{45} states
- A helicopter: continuous state space (infinitely many states)

$\Rightarrow$ Too many states for a look-up table.

Value Function Approximation

Approximate v and/or q using function approximations

$$V(s) \approx \hat{V}(s,\theta) \qquad\qquad Q(s,a) \approx \hat{Q}(s,a,\theta) \tag{3.4}$$

Different approximation methods:

- Linear combinations of features
- Neural network ("deep" methods)
- Decision tree
- Nearest neighbor
- Fourier / wavelet bases
- ...and many more

θ can be updated using e.g. gradient descent or least squares methods.

Example

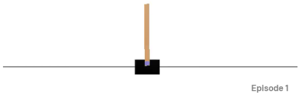
- OpenAI Gym RL toolkit

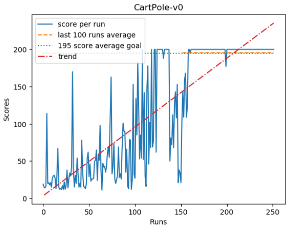
Example

- OpenAI Gym RL toolkit
- Learning to control the CartPole (Python code on Blackboard).
- Continuous state-space discretized into *boxes*.

Example

- OpenAI Gym RL toolkit
- Learning to control the CartPole (Python code on Blackboard).
- Continuous state-space discretized into *boxes*.
- Guided practical exercise: Exchange Q-table with function approximator (neural net) using TensorFlow and Keras.





Further Reading

RL:

- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, New York, NY, 1997 (Chapter on RL)
- Russel and Norvig: *Artificial Intelligence: A Modern Approach*, Third Edition, Pearson, 2010 (Chapter on RL)

Further Reading

RL:

- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, New York, NY, 1997 (Chapter on RL)
- Russel and Norvig: *Artificial Intelligence: A Modern Approach*, Third Edition, Pearson, 2010 (Chapter on RL)
- Udacity free online course on Reinforcement Learning

Further Reading

RL:

- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, New York, NY, 1997 (Chapter on RL)
- Russel and Norvig: *Artificial Intelligence: A Modern Approach*, Third Edition, Pearson, 2010 (Chapter on RL)
- Udacity free online course on Reinforcement Learning
- Sutton and Barto: *Reinforcement Learning: An Introduction*, Second Edition, MIT Press, Cambridge, MA, 2017
- David Silver, UCL Course on RL (a 15 hour course at Univ. Coll. London, available on YouTube)

Further Reading

RL:

- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, New York, NY, 1997 (Chapter on RL)
- Russel and Norvig: *Artificial Intelligence: A Modern Approach*, Third Edition, Pearson, 2010 (Chapter on RL)
- Udacity free online course on Reinforcement Learning
- Sutton and Barto: *Reinforcement Learning: An Introduction*, Second Edition, MIT Press, Cambridge, MA, 2017
- David Silver, UCL Course on RL (a 15 hour course at Univ. Coll. London, available on YouTube)
- Lewis, Vrabie and Vamvoudakis: *Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers*, IEEE Control Systems Magazine, Volume 32, Issue 6, Dec. 2012

Further Reading

RL:

- Tom M. Mitchell: *Machine Learning*, McGraw-Hill, New York, NY, 1997 (Chapter on RL)
- Russel and Norvig: *Artificial Intelligence: A Modern Approach*, Third Edition, Pearson, 2010 (Chapter on RL)
- Udacity free online course on Reinforcement Learning
- Sutton and Barto: *Reinforcement Learning: An Introduction*, Second Edition, MIT Press, Cambridge, MA, 2017
- David Silver, UCL Course on RL (a 15 hour course at Univ. Coll. London, available on YouTube)
- Lewis, Vrabie and Vamvoudakis: *Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers*, IEEE Control Systems Magazine, Volume 32, Issue 6, Dec. 2012

Deep RL:

- OpenAI Spinning Up
- YouTube videos and slides from Deep Reinforcement Learning course by Sergey Levine at UC Berkeley