Machine Learning

C. Reinforcement Learning
C.2. Markov Decision Process

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Outline

1. Introduction

2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

5. Partially Observable Markov Decision Processes

Syllabus

Tue. 21.10.	(1)	0. Introduction
		A. Supervised Learning
Wed. 22.10.	(2)	A.1 Linear Regression
Tue. 28.10.	(3)	A.2 Linear Classification
Wed. 29.10.	(4)	A.3 Regularization
Tue. 4.11.	(5)	A.4 High-dimensional Data
Wed. 5.11.	(6)	A.5 Nearest-Neighbor Models
Tue. 11.11.	(7)	A.6 Decision Trees
Wed. 12.12.	(8)	A.7 Support Vector Machines
Tue. 18.11.	(9)	A.8 A First Look at Bayesian and Markov Networks
		B. Unsupervised Learning
Wed. 19.11.	(10)	B.1 Clustering
Tue. 25.11.	(11)	B.2 Dimensionality Reduction
Wed. 26.11.	(12)	B.3 Frequent Pattern Mining
		C. Reinforcement Learning
Tue. 2.12.	(13)	C.1 State Space Models
Wed. 3.12.	(14)	C.2 Markov Decision Processes

Outline

1. Introduction

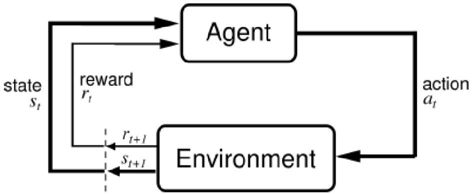
2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

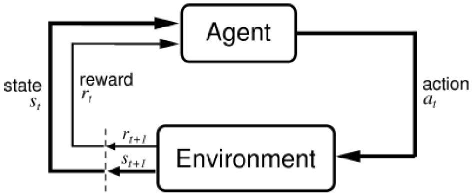
5. Partially Observable Markov Decision Processes

Agent-Environment Interaction



[SB98, fig. 3.1]

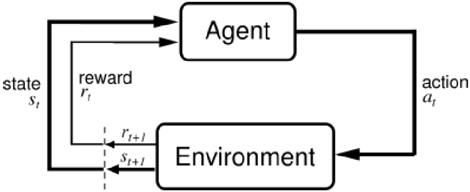
Agent-Environment Interaction



- learner and decisionmaker is called **agent**

[SB98, fig. 3.1]

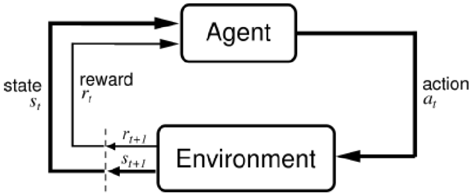
Agent-Environment Interaction



- learner and decisionmaker is called **agent**
- the agent interacts with the **environment** (everything outside the agent)

[SB98, fig. 3.1]

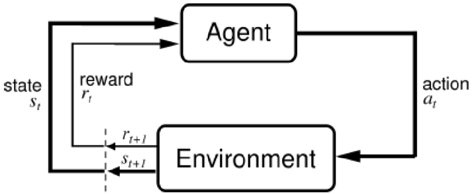
Agent-Environment Interaction



- learner and decisionmaker is called **agent**
- the agent interacts with the **environment** (everything outside the agent)
- the agents selects actions and the environment responds with new situations and rewards

[SB98, fig. 3.1]

Agent-Environment Interaction



- learner and decisionmaker is called **agent**
- the agent interacts with the **environment** (everything outside the agent)
- the agents selects actions and the environment responds with new situations and rewards
- the agents tries to maximize the rewards over time

[SB98, fig. 3.1]

Overview

- Markov Decision Process (MDP)

Overview

- Markov Decision Process (MDP)
- An MDP consists of States S (fully observed), actions A, rewards $r_a(s) = R(s, a, s')$ and transition probabilities $T_a(s, s')$

Overview

- Markov Decision Process (MDP)
- An MDP consists of States S (fully observed), actions A, rewards $r_a(s) = R(s, a, s')$ and transition probabilities $T_a(s, s')$
- Our System is Markovian, so the transition function depends just on the current state:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t) = T_{a_t}(s, s')$$

Overview

- A **policy** π describes how actions are picked at each state:

Overview

- A **policy** π describes how actions are picked at each state:

$$\pi(s, a) = p(a_t = a \mid s_t = s)$$

Overview

- A **policy** π describes how actions are picked at each state:

$$\pi(s, a) = p(a_t = a \mid s_t = s)$$

- The **value function** of a policy V^π is:

Overview

- A **policy** π describes how actions are picked at each state:

$$\pi(s, a) = p(a_t = a \mid s_t = s)$$

- The **value function** of a policy V^π is:

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

Overview

- A **policy** π describes how actions are picked at each state:

$$\pi(s, a) = p(a_t = a \mid s_t = s)$$

- The **value function** of a policy V^π is:

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- We can find V^π by solving a linear system of equations

Overview

- A **policy** π describes how actions are picked at each state:

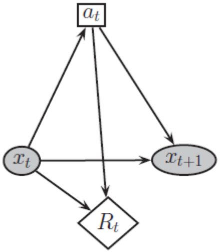
$$\pi(s, a) = p(a_t = a \mid s_t = s)$$

- The **value function** of a policy V^π is:

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- We can find V^π by solving a linear system of equations
- **Policy iteration** gives a greedy local search procedure based on the value of policies

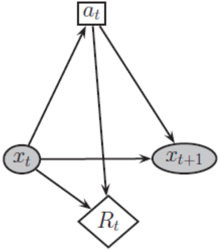
Markov Decision Process



(b)

[Mur12, fig. 10.13 (b)]

Markov Decision Process

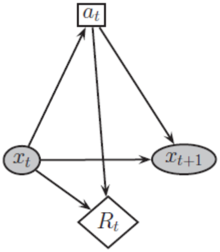


(b)

- influence diagram of an MDP

[Mur12, fig. 10.13 (b)]

Markov Decision Process

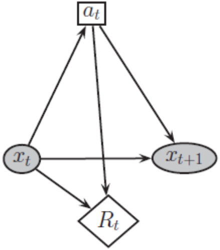


(b)

- influence diagram of an MDP
- easy to solve, since we only have to compute mapping from observed states to actions

[Mur12, fig. 10.13 (b)]

Markov Decision Process



(b)

- influence diagram of an MDP
- easy to solve, since we only have to compute mapping from observed states to actions
- assume Markov Property holds as nearly as possible

[Mur12, fig. 10.13 (b)]

Action-Value Function

Action-Value Function

Similarly to the Value function we define the value of taking action a in state s under a policy π, denoted as $Q^\pi(s, a)$

Action-Value Function

Similarly to the Value function we define the value of taking action a in state s under a policy π, denoted as $Q^\pi(s,a)$

$$Q^\pi(s,a) = E_\pi\left\{R_t \mid s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Action-Value Function

Similarly to the Value function we define the value of taking action a in state s under a policy π, denoted as $Q^\pi(s,a)$

$$Q^\pi(s,a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

expected return starting from s, taking action a and following policy π

Action-Value Function

Similarly to the Value function we define the value of taking action a in state s under a policy π, denoted as $Q^\pi(s, a)$

$$Q^\pi(s, a) = E_\pi\{R_t \mid s_t = s, a_t = a\} = E_\pi\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\Big\}$$

expected return starting from s, taking action a and following policy π

We call Q^π **action-value function** for policy π

Action-Value Function

Similarly to the Value function we define the value of taking action a in state s under a policy π, denoted as $Q^\pi(s, a)$

$$Q^\pi(s, a) = E_\pi\left\{R_t \mid s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

expected return starting from s, taking action a and following policy π

We call Q^π **action-value function** for policy π

return means a function of future rewards that the agent seeks to maximize

Outline

1. Introduction

2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

5. Partially Observable Markov Decision Processes

Dynamic Programming (DP)

Dynamic Programming (DP)

- are well developed mathematically

Dynamic Programming (DP)

- are well developed mathematically
- but require a complete and accurate model of the environment

Dynamic Programming (DP)

- are well developed mathematically
- but require a complete and accurate model of the environment
- refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment

Dynamic Programming (DP)

- are well developed mathematically
- but require a complete and accurate model of the environment
- refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment
- computational expensive

Dynamic Programming (DP)

- are well developed mathematically
- but require a complete and accurate model of the environment
- refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment
- computational expensive
- assume the environment is a finite MDP, state and action sets are finite

Optimal Policies and Optimal Value Functions

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- the best value that can be achieved at each state

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- the best value that can be achieved at each state
- In a **finite** MDP exists a unique optimal value function (shown by Bellman, 1957)

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- the best value that can be achieved at each state
- In a **finite** MDP exists a unique optimal value function (shown by Bellman, 1957)
- any policy that achieves the optimal value is called a **optimal policy**, there has to be at least one deterministic optimal policy

Optimal Policies and Optimal Value Functions

- goal: find policy that has maximum value
- The **optimal value function** V^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)$$

- the best value that can be achieved at each state
- In a **finite** MDP exists a unique optimal value function (shown by Bellman, 1957)
- any policy that achieves the optimal value is called a **optimal policy**, there has to be at least one deterministic optimal policy
- both policy iteration and value iteration can be used

Algorithm Idea

Algorithm Idea

- Update rule with recursive Bellman equations

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:
  1. Start: initial approximation V_0
  2. Each iteration: update value function estimate

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:
  1. Start: initial approximation V_0
  2. Each iteration: update value function estimate

$$V_{k+1}(s) \leftarrow \max_a \left(r_a(s) + \gamma \sum_{s'} T_a(s, s') V_k(s') \right), \forall s$$

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:
  1. Start: initial approximation V_0
  2. Each iteration: update value function estimate

$$V_{k+1}(s) \leftarrow \max_a \left(r_a(s) + \gamma \sum_{s'} T_a(s, s') V_k(s') \right), \forall s$$

  3. Stop when maximum value change between iterations is below a threshold

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:
  1. Start: initial approximation V_0
  2. Each iteration: update value function estimate

$$V_{k+1}(s) \leftarrow \max_a \left(r_a(s) + \gamma \sum_{s'} T_a(s, s') V_k(s') \right), \forall s$$

  3. Stop when maximum value change between iterations is below a threshold
- Similar update for policy evaluation

Algorithm Idea

- Update rule with recursive Bellman equations
- Value iteration:
  1. Start: initial approximation V_0
  2. Each iteration: update value function estimate

$$V_{k+1}(s) \leftarrow \max_a \left(r_a(s) + \gamma \sum_{s'} T_a(s,s') V_k(s') \right), \forall s$$

  3. Stop when maximum value change between iterations is below a threshold
- Similar update for policy evaluation
- more efficient: instead of updating every state on every iteration, focus on **important states**

Learning with Dynamic Programming

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s), \hat{T}_a(s, s')$; observe transitions in the environment

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:
  - learn MDP model, or an approximation

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:
  - learn MDP model, or an approximation
  - use it for policy evaluation or to find optimal policy

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:
  - learn MDP model, or an approximation
  - use it for policy evaluation or to find optimal policy

  Model-free:

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:
  - learn MDP model, or an approximation
  - use it for policy evaluation or to find optimal policy

  Model-free:
  - derive optimal policy without explicitly learning the model

Learning with Dynamic Programming

- learn **approximate model** $\hat{r}_a(s)$, $\hat{T}_a(s, s')$; observe transitions in the environment
- maximum likelihood for probabilities and supervised learning for the rewards
- this approach is called **model-based RL**
- **model-based vs. model-free reinforcement learning**:
  Model-based:
  - learn MDP model, or an approximation
  - use it for policy evaluation or to find optimal policy

  Model-free:
  - derive optimal policy without explicitly learning the model
  - useful when model is difficult to represent and/or learn

Asynchronous Dynamic Programming

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP
- if the state set is very large, then even a single sweep can be prohibitively expensive

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP
- if the state set is very large, then even a single sweep can be prohibitively expensive
- an asynchronous algorithm must continue to backup the values of all the states (to converge correctly), it can't ignore any state after some point in the computation

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP
- if the state set is very large, then even a single sweep can be prohibitively expensive
- an asynchronous algorithm must continue to backup the values of all the states (to converge correctly), it can't ignore any state after some point in the computation
- asynchronous algorithms allow great flexibility in selecting states

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP
- if the state set is very large, then even a single sweep can be prohibitively expensive
- an asynchronous algorithm must continue to backup the values of all the states (to converge correctly), it can't ignore any state after some point in the computation
- asynchronous algorithms allow great flexibility in selecting states
- e.g. generate trajectories through the MDP and update states whenever they appear on such a trajectory

Asynchronous Dynamic Programming

- DP methods involve operations over the entire state set of the MDP
- if the state set is very large, then even a single sweep can be prohibitively expensive
- an asynchronous algorithm must continue to backup the values of all the states (to converge correctly), it can't ignore any state after some point in the computation
- asynchronous algorithms allow great flexibility in selecting states
- e.g. generate trajectories through the MDP and update states whenever they appear on such a trajectory
  or: important states like: visited often during a game/procedure

Outline

1. Introduction

2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

5. Partially Observable Markov Decision Processes

Introduction

Introduction

- first learning method for estimating value functions and discover optimal policies

Introduction

- first learning method for estimating value functions and discover optimal policies
- no need of complete knowledge of the environment

Introduction

- first learning method for estimating value functions and discover optimal policies
- no need of complete knowledge of the environment
- Monte Carlo methods require only **experience** (sample sequences of states, actions and rewards) from online or simulated interaction with the environment

Introduction

- first learning method for estimating value functions and discover optimal policies
- no need of complete knowledge of the environment
- Monte Carlo methods require only **experience** (sample sequences of states, actions and rewards) from online or simulated interaction with the environment
- only for episodic tasks, i.e. we assume experience is divided into episodes

Introduction

- first learning method for estimating value functions and discover optimal policies
- no need of complete knowledge of the environment
- Monte Carlo methods require only **experience** (sample sequences of states, actions and rewards) from online or simulated interaction with the environment
- only for episodic tasks, i.e. we assume experience is divided into episodes
- only after completition of an episode value estimates and policies are changed

Monte Carlo Method (simple)

Monte Carlo Method (simple)

- suppose we have episodic tasks

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^{\pi}(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^\pi(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited
- **Every-Visit MC**: average returns for every time s is visited in an episode

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^\pi(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited
- **Every-Visit MC**: average returns for every time s is visited in an episode
- **First-Visit MC**: average returns only for first time s is visited in an episode

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^{\pi}(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited
- **Every-Visit MC**: average returns for every time s is visited in an episode
- **First-Visit MC**: average returns only for first time s is visited in an episode
- both converge asymptotically

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^\pi(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited
- **Every-Visit MC**: average returns for every time s is visited in an episode
- **First-Visit MC**: average returns only for first time s is visited in an episode
- both converge asymptotically
- do the computation incrementally: after received return R_t, update

Monte Carlo Method (simple)

- suppose we have episodic tasks
- the agent behaves according to some policy π for a while, generating several trajectories
- Compute $V^{\pi}(s)$ by **averaging the observed returns** after s on the trajectories in which s was visited
- **Every-Visit MC**: average returns for every time s is visited in an episode
- **First-Visit MC**: average returns only for first time s is visited in an episode
- both converge asymptotically
- do the computation incrementally: after received return R_t, update

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)), \alpha \in (0, 1) \text{ learning rate}$$

First-visit Monte Carlo policy evaluation

Initialize:

- $\pi \leftarrow$ policy to be evaluated
- $V \leftarrow$ an arbitrary state-value function
- $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

- (a) Generate an episode using π
- (b) For each state s appearing in the episode:
  - $R \leftarrow$ return following the first occurrence of s
  - Append R to $Returns(s)$
  - $V(s) \leftarrow \text{average}(Returns(s))$

[SB98, fig. 5.1]

Example: Blackjack

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:
  - curent sum $(12 - 21)$
  - dealer's showing card $(ace, 2 - 10)$
  - usable ace? (counting ace as 11 without going to bust)

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:
  - curent sum $(12 - 21)$
  - dealer's showing card $(ace, 2 - 10)$
  - usable ace? (counting ace as 11 without going to bust)
- **reward**: s+1 winning, 0 draw, -1 loss

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:
  - curent sum $(12 - 21)$
  - dealer's showing card $(ace, 2 - 10)$
  - usable ace? (counting ace as 11 without going to bust)
- **reward**: s+1 winning, 0 draw, -1 loss
- **actions**: stick (stop receiving cards), hit (receive another card)

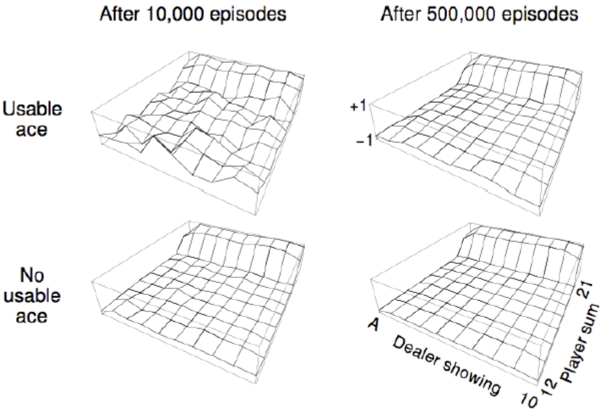
Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:
  - curent sum $(12 - 21)$
  - dealer's showing card $(ace, 2 - 10)$
  - usable ace? (counting ace as 11 without going to bust)
- **reward**: s+1 winning, 0 draw, -1 loss
- **actions**: stick (stop receiving cards), hit (receive another card)
- **policy**: stick if sum is 20 or 21, else hit

Example: Blackjack

- **object**: card sum to be greater than the dealers, not exceeding 21
- **states**:
  - curent sum $(12 - 21)$
  - dealer's showing card $(ace, 2 - 10)$
  - usable ace? (counting ace as 11 without going to bust)
- **reward**: s+1 winning, 0 draw, -1 loss
- **actions**: stick (stop receiving cards), hit (receive another card)
- **policy**: stick if sum is 20 or 21, else hit
- Dealer's fixed strategy: stick if ≤ 17 and hit if < 17

Example: Blackjack/approximate value functions



[SB98, fig. 5.2]

Monte Carlo Estimation of Action Values

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)
- without a model, this is not sufficient: one must explicitly estimate the value of each action

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)
- without a model, this is not sufficient: one must explicitly estimate the value of each action
- estimate $Q^{\pi}(s, a)$, he expected return when starting in state s, taking action a and following policy π

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)
- without a model, this is not sufficient: one must explicitly estimate the value of each action
- estimate $Q^\pi(s, a)$, he expected return when starting in state s, taking action a and following policy π
- its like for the value function, here as the average return starting from state s and action a following π

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)
- without a model, this is not sufficient: one must explicitly estimate the value of each action
- estimate $Q^{\pi}(s, a)$, he expected return when starting in state s, taking action a and following policy π
- its like for the value function, here as the average return starting from state s and action a following π
- problem if π is deterministic, then some state-action pairs (s, a) will never be visited, so we must assure continual exploration:

Monte Carlo Estimation of Action Values

- If a model is not available, then it is particularly useful to estimate action values rather than the values
- with a model, value function is sufficient to determine a policy (one looks one step ahead an chooses the action which leads to the best combination of reward and next state)
- without a model, this is not sufficient: one must explicitly estimate the value of each action
- estimate $Q^\pi(s, a)$, he expected return when starting in state s, taking action a and following policy π
- its like for the value function, here as the average return starting from state s and action a following π
- problem if π is deterministic, then some state-action pairs (s, a) will never be visited, so we must assure continual exploration: e.g. every state-action pair has a non-zero proability of being starting pair

Policy improvement

Policy improvement

is done by making the policy greedy with respect to the current value function

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

policy improvement can then be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

policy improvement can then be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

$$Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s, a))$$

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

policy improvement can then be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \end{aligned}$$

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

policy improvement can then be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \end{aligned}$$

Policy improvement

is done by making the policy greedy with respect to the current value function the greedy policy is the one that, for each $s \in S$, deterministically chooses an action with maximal Q-value

$$\pi(s) = \arg\max_a Q(s, a)$$

policy improvement can then be done by constructing each π_{k+1} as the greedy policy with respect to Q^{π_k}

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &= V^{\pi_k}(s) \end{aligned}$$

Outline

1. Introduction

2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

5. Partially Observable Markov Decision Processes

Temporal-Difference (TD)

Temporal-Difference (TD)

- Monte Carlo uses actual return R_t for estimating the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

Temporal-Difference (TD)

- Monte Carlo uses actual return R_t for estimating the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

- The simplest TD method, $TD(0)$, uses instead an **estimate** of the return:

Temporal-Difference (TD)

- Monte Carlo uses actual return R_t for estimating the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

- The simplest TD method, $TD(0)$, uses instead an **estimate** of the return:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

Temporal-Difference (TD)

- Monte Carlo uses actual return R_t for estimating the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right]$$

- The simplest TD method, $TD(0)$, uses instead an **estimate** of the return:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

- if $V(s_{t+1})$ were correct, this would be like dynamic programming

TD Learning Algorithm

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
    $s \leftarrow s'$
  until s is terminal

[SB98, fig. 6.1]

Advantages

Advantages

- No model of the environment

Advantages

- No model of the environment
- TD only needs experience with the environment

Advantages

- No model of the environment
- TD only needs experience with the environment
- Online, incremental learning:

Advantages

- No model of the environment
- TD only needs experience with the environment
- Online, incremental learning:
  - can learn before knowing the final outcome

Advantages

- No model of the environment
- TD only needs experience with the environment
- Online, incremental learning:
  - can learn before knowing the final outcome
  - less memory required

Advantages

- No model of the environment
- TD only needs experience with the environment
- Online, incremental learning:
  - can learn before knowing the final outcome
  - less memory required
- both TD and MC converge, but TD usually learns faster

Outline

1. Introduction

2. Dynamic Programming and Reinforcement Learning

3. Monte Carlo Method

4. Temporal-Difference Prediction

5. Partially Observable Markov Decision Processes

brief overview of POMDPs

- like MDP but agent does not have fully knowledge about the environment

brief overview of POMDPs

- like MDP but agent does not have fully knowledge about the environment
- what distinguishes a POMDP from a MDP is that the agent perceives an observation $o \in \Omega$, instead of observing s' directly

brief overview of POMDPs

- like MDP but agent does not have fully knowledge about the environment
- what distinguishes a POMDP from a MDP is that the agent perceives an observation $o \in \Omega$, instead of observing s' directly
- we want to find a mapping from probability distrinutions (over states) to actions

brief overview of POMDPs

- like MDP but agent does not have fully knowledge about the environment
- what distinguishes a POMDP from a MDP is that the agent perceives an observation $o \in \Omega$, instead of observing s' directly
- we want to find a mapping from probability distrinutions (over states) to actions
- a probability distributions over state are called **belief states** b and the entire probability space the **belief space**

brief overview of POMDPs

- like MDP but agent does not have fully knowledge about the environment
- what distinguishes a POMDP from a MDP is that the agent perceives an observation $o \in \Omega$, instead of observing s' directly
- we want to find a mapping from probability distrinutions (over states) to actions
- a probability distributions over state are called **belief states** b and the entire probability space the **belief space**
- so agent needs to update its belief upon taking action a and observation o, $b' = \tau(b, a, o)$, wth τ belief state transition function

Policy and Value Function

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t r(b_t, a_t) = \sum_{t=0}^{\infty} \gamma^t E\{R(s_t, a_t) \mid b_0, \pi\}$$

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t r(b_t, a_t) = \sum_{t=0}^{\infty} \gamma^t E\{R(s_t, a_t) \mid b_0, \pi\}$$

- optimal policy

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t r(b_t, a_t) = \sum_{t=0}^{\infty} \gamma^t E\{R(s_t, a_t) \mid b_0, \pi\}$$

- optimal policy

$$\pi^* = \arg\max_{pi} V^\pi(b_0)$$

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t r(b_t, a_t) = \sum_{t=0}^{\infty} \gamma^t E\{R(s_t, a_t) \mid b_0, \pi\}$$

- optimal policy

$$\pi^* = \arg\max_{pi} V^\pi(b_0)$$

- optimal value function

Policy and Value Function

- policy π and action $a = \pi(b)$ for any belief, b_0 initial belief state
- expected Reward for policy π is

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t r(b_t, a_t) = \sum_{t=0}^{\infty} \gamma^t E\{R(s_t, a_t) \mid b_0, \pi\}$$

- optimal policy

$$\pi^* = \arg\max_{pi} V^\pi(b_0)$$

- optimal value function

$$V^*(b) = \max_a \left[r(b,a) + \gamma \sum_{o \in O} \Omega(o \mid b, a) V^*(\tau(b,a,o)) \right]$$

Further Readings

- [SB98, chapter 5,6,7].
- [WvO12].

References

Kevin P. Murphy.
Machine learning: a probabilistic perspective.
The MIT Press, 2012.

R.S. Sutton and A.G. Barto.
Reinforcement Learning: An Introduction.
A Bradford book. MIT Press, 1998.

M. Wiering and M. van Otterlo.
Reinforcement Learning: State-of-the-Art.
Adaptation, Learning, and Optimization. Springer, 2012.