Planning and Optimal Control

6. Introduction to Reinforcement Learning

Lars Schmidt-Thieme

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

Syllabus

		A. Models for Sequential Data
Tue. 22.10.	(1)	1. Markov Models
Tue. 29.10.	(2)	2. Hidden Markov Models
Tue. 5.11.	(3)	3. State Space Models
Tue. 12.11.	(4)	3b. (ctd.)
		B. Models for Sequential Decisions
Tue. 19.11.	(5)	1. Markov Decision Processes
Tue. 26.11.	(6)	1b. (ctd.)
Tue. 3.12.	(7)	1c. (ctd.)
Tue. 10.12.	(8)	2. Monte Carlo and Temporal Difference Methods
Tue. 17.12.	(9)	3. Q Learning
Tue. 24.12.	—	— *Christmas Break* —
Tue. 7.1.	(10)	4. Policy Gradient Methods
Tue. 14.1.	(11)	tba
Tue. 21.1.	(12)	tba
Tue. 28.1.	(13)	8. Reinforcement Learning for Games
Tue. 4.2.	(14)	Q&A

Outline

Outline

1. Introduction

2. Monte Carlo Methods

3. Learning the Value Function: TD(0)

4. Learning the Action Value Function: SARSA

Policy Inference vs. Policy Learning

- **Markov Decision Problem**:
  - transition model p and reward model r are known.
  - compute an optimal policy (**optimal policy inference**).
- **Reinforcement Learning**:
  - transition model p and reward model r are not known.
  - learn an optimal policy from state/action/reward sequence data (**optimal policy learning**).

Problems & Sampling MDP Data

a. **state/action/reward Markov process learning**:

- learn the
  - transition model P^{π} of the state/action/reward Markov process or the
  - value function V^{π}

  of an MDP under a fixed pre-existing policy π.
  - called **generating policy**, **explorative policy** or **sampling policy**.
  - π does not depend on past samples and current estimates.
  - e.g., choose actions in each state uniformly at random

b. **MDP learning**:

- learn the transition model p and reward model r of an MDP.
- **passive sampling**: sample with a generating policy that
  - does not depend on past samples and current estimates, but should
  - guarantee to explore the whole state/action space.
- **active sampling**: use a generating policy that
  - selects actions s.t. informative samples are created.
    — e.g., uncertain transitions, uncertain rewards.

Problems & Sampling MDP Data (2/2)

c. **reinforcement learning**:
- learn an optimal policy π^*,
  without knowing the MDP (p, r).
  - i.e., a policy with maximal value function.
- the generating policy has to balance
  - **exploration** of actions with uncertain effects (on transitions and rewards) and
  - **exploitation** of actions with likely best value

  to ensure focus on valuable actions.

d. **joint process learning and control**:
- create state/action/reward sequences with maximal value.
  - i.e., learn and execute a policy that overall will lead to maximal value at the same time.
- such a policy also needs to balance exploration and exploitation
  to ensure it realizes a high value.

Reinforcement Learning Approaches

1. **model-based reinforcement learning**:
   - given state/action/reward sequences, learn the MDP model (p, r),
   - afterwards use a policy optimization algorithm on the learnt model to find an optimal policy.
2. **direct reinforcement learning**:
   - given state/action/reward sequences, learn the
     - optimal policy π^* or even only the
     - value function V^* or the
     - action value function Q^* of the optimal policy.

Generating Policies

- **uniform at random policy**:

$$\pi_{\text{uniform}}(s, a) := \frac{1}{|A|}$$

Outline

Observed and Estimated Values

data: N state/action/reward sequences sampled from an MDP with unknown transition/reward model under a policy π:

$$\mathcal{D} := \{((s_{n,t}, a_{n,t}, r_{n,t})_{t=0:T_n-1}, s_{T_n}) \mid n = 1:N\} \subseteq (S \times A \times \mathbb{R})^* \times S$$

observed values:

$$V_{n,t} := \text{value}(r_{n,t:T_n-1}), \quad n = 1:N, t = 0:T_n - 1$$

e.g., for the discounted criterion:

$$:= \sum_{t'=t}^{T_n-1} \gamma^{t'-t}\, r_{n,t'}$$

occurrences of state $s \in S$ in data $\mathcal{D}$:

$$I_s := \{(n,t) \mid n = 1:N, t = 0:T_n - 1, s_{n,t} = s\}$$

estimated value of state $s \in S$:

$$\hat{V}_s := \frac{1}{|I_s|} \sum_{(n,t)\in I_s} V_{n,t}$$

Observed Values?

- in general, values cannot be observed.
  - as they may depend on infinitely many future rewards.
- to observe values, one needs to ensure/assume that all sequences terminate in a state of true value zero:

$$V_{n,T_n} = 0 \quad n = 1:N$$

- e.g., if the state Markov chain has an absorbing state s with reward 0,

$$\forall s' \exists t : p(s_t = s \mid s_0 = s') = 1, \quad p(s \mid s) = 1, \quad \forall a : r(s,a) = 0$$

all sequences will be essentially finite (**terminal state**).

Excursion: Online Update of the Mean

- given: streaming data $x_t \in \mathbb{R}, t = 1, 2, 3, \ldots$.
- wanted: mean μ_T of values received so far (for any T).

- method 1: **store data**
  - store all data:

$$X_0 := ()$$
$$X_T := X_{T-1} \oplus (x_T)$$

  - $\mu_T := \frac{1}{|X_T|} \sum_{t=1}^{|X_T|} X_t$

Note: $\oplus$ denotes concatenation of two sequences.

Excursion: Online Update of the Mean

- given: streaming data $x_t \in \mathbb{R}, t = 1, 2, 3, \ldots$.
- wanted: mean μ_T of values received so far (for any T).

- method 1: **store data**
  - store all data:

$$X_0 := ()$$
$$X_T := X_{T-1} \oplus (x_T)$$

  - $\mu_T := \frac{1}{|X_T|} \sum_{t=1}^{|X_T|} X_t$

- method 2: **maintain sum and length** (online update)
  - maintain sum and length:

$$s_0 := 0, \qquad L_0 := 0$$
$$s_T := s_{T-1} + x_T, \qquad L_T := L_{T-1} + 1$$

  - $\mu_T := \frac{s_T}{L_T}$

Note: $\oplus$ denotes concatenation of two sequences.

Excursion: Online Update of the Mean

- given: streaming data $x_t \in \mathbb{R}, t = 1, 2, 3, \ldots$.
- wanted: mean μ_T of values received so far (for any T).

- method 1: **store data** ...

- method 2: **maintain sum and length** (online update)
  - maintain sum and length:

$$s_0 := 0, \qquad L_0 := 0$$
$$s_T := s_{T-1} + x_T, \qquad L_T := L_{T-1} + 1$$

  - $\mu_T := \frac{s_T}{L_T}$

- method 3: **maintain mean and length** (online update)
  - maintain mean and length:

$$\mu_0 := 0, \qquad L_0 := 0$$
$$\mu_T := \mu_{T-1} + \frac{1}{L_{T-1}+1}(x_T - \mu_{T-1}), \qquad L_T := L_{T-1} + 1$$

Recursive Estimation of Values

observed state values:

$$V_{s,k} := V_{n_k,t_k} \quad \text{for } I_s = \{(n_1,t_1),(n_2,t_2),\ldots,(n_K,t_K)\}$$

estimated state values from first k occurrences:

$$\begin{aligned}\hat{V}_{s,k} &:= \frac{1}{k}\sum_{j=1}^{k} V_{s,j}\\ &= \hat{V}_{s,k-1} + \frac{1}{k}(V_{s,k} - \hat{V}_{s,k-1})\\ \hat{V}_{s,0} &:= 0\end{aligned}$$

- does not have to store all values $V_{s,k}$ for all k,
- but just two values: $\hat{V}_{s,k}$ (for current k) and k itself. (**online update of the mean**).

Recursive Estimation of Values

for

$$\hat{V}_{s,k} = \hat{V}_{s,k-1} + \frac{1}{k}(V_{s,k} - \hat{V}_{s,k-1})$$

and even with

$$\hat{V}_{s,k} = \hat{V}_{s,k-1} + \alpha_k(V_{s,k} - \hat{V}_{s,k-1}), \quad \alpha_k > 0, \alpha_k \rightarrow 0$$

we will get

$$\lim_{k \rightarrow \infty} \hat{V}_{s,k} = V^{\pi}(s)$$

Monte Carlo method update rule

Monte Carlo method update rule:

$$\hat{V}_{s_t} := \hat{V}_{s_t} + \alpha_{k(s_t)}(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} - \hat{V}_{s_t}),$$
$$t = 0 : T-1$$

- update values of all visited states after completion of each sequence $s, a, r, T := s_n, a_n, r_n, T_n$.
  - as it has to compute the observed value from all future rewards
- $k(s)$ keeps track of the frequency of state s seen so far.

Monte Carlo Value Function Learning Algorithm

learn-value-discounted-mc$(S, A, \gamma, s_{\text{term}}, N, \pi, v_0, \alpha)$:
  $\hat{V} := (v_0)_{s \in S}, \quad k := (0)_{s \in S}$
  for $n := 1, \ldots, N$:
    $(s, a, r, T) :=$ generate-episode$(S, A, s_{\text{term}}, \pi)$
    for $t := 0, \ldots, T-1$:
      $\hat{V}_{s_t} := \hat{V}_{s_t} + \alpha_{k(s_t)}((\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}) - \hat{V}_{s_t})$
      $k(s_t) := k(s_t) + 1$
  return $\hat{V}$

generate-episode$(S, A, s_{\text{term}}, \pi)$:
  $s := (), a := (), r := (), T := 0$
  $s_0 :=$ new_process()
  while $s_T \neq s_{\text{term}}$:
    $a_T := \pi(s_T)$
    $(r_T, s_{T+1}) :=$ execute_action(s_T, a_T)
    $T := T + 1$
  return (s, a, r, T)

where

- s_{term} **terminal state** with zero reward.
- π **generating policy**
- $v_0 \in \mathbb{R}$ initial value of all states.
- α_k **learning rate** for update step k, e.g., $\alpha_k := 1/k$.
- new_process() sets up a new process.
- execute_action(s, a) executes action a in process in state s.

Unbiased Estimation from First Visit only

- if a state if visited twice in a sequence, two estimates of its value enter the overall mean (**all visits**).
  - which are not independent and introduce a bias.
- fix: retain only the first occurrence (**first visit**)
  - unbiased
  - experimentally said to yield somewhat smaller squared error.

Outline

Temporal Differences

Monte Carlo: update after completion of a sequence:

$$\hat{V}_{s_t} = \hat{V}_{s_t} + \alpha(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T - \hat{V}_{s_t}), \quad t = 1 : T$$

Temporal differences:

$$\begin{aligned}
\hat{V}_{s_t} &= \hat{V}_{s_t} + \alpha(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T - \hat{V}_{s_t}) \\
&= \hat{V}_{s_t} + \alpha(\quad (r_t + \gamma \hat{V}_{s_{t+1}} - \hat{V}_{s_t}) \\
&\qquad + \gamma (r_{t+1} + \gamma \hat{V}_{s_{t+2}} - \hat{V}_{s_{t+1}}) \\
&\qquad + \gamma^2 (r_{t+2} + \gamma \hat{V}_{s_{t+3}} - \hat{V}_{s_{t+2}}) \\
&\qquad \vdots \\
&\qquad + \gamma^{T-t} (r_{T-1} + \gamma \hat{V}_{s_T} - \hat{V}_{s_{T-1}})) \\
&= \hat{V}_{s_t} + \alpha(\delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \ldots \gamma^{T-t} \delta_{T-1}) \\
\delta_t &:= r_t + \gamma \hat{V}_{s_{t+1}} - \hat{V}_{s_t}
\end{aligned}$$

Note: Still assuming the sequence terminates in a value 0 state: $\hat{V}_{s_T} = V_{s_T} = 0$.

Temporal Difference update rule TD(0):

$$\hat{V}_{s_t} = \hat{V}_{s_t} + \alpha(r_t + \gamma \hat{V}_{s_{t+1}} - \hat{V}_{s_t})$$

- update values of a visited state immediately after action is taken, reward is received and next state has been determined.
- TD(0) estimates only the value function.
  - the value function of the generating policy.
  - if the transition model is known,
    one easily can estimate a policy from its value function, e.g.,

$$\pi^*(s) \in \arg\max_{a \in A}(r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a)\, V^*_\gamma(s')), \quad s \in S$$

  - but without access to the transition model,
    the value function alone does not allow to do so.

TD(0) Value Function Learning Algorithm

learn-value-discounted-td0$(S, A, \gamma, s_{\text{term}}, N, \pi, v_0, \alpha)$:
  $\hat{V} := (v_0)_{s \in S}$
  for $n := 1, \ldots, N$:
    $s := (), a := (), r := (), t := 0$
    $s_0 :=$ new_process()
    while $s_t \neq s_{\text{term}}$:
      $a_t := \pi(s_t)$
      $(r_t, s_{t+1}) :=$ execute_action(s_t, a_t)
      $\hat{V}_{s_t} := \hat{V}_{s_t} + \alpha(r_t + \gamma \hat{V}_{s_{t+1}} - \hat{V}_{s_t})$
      $t := t + 1$
  return $\hat{V}$

where

- s_{term} **terminal state** with zero reward.
- π **generating policy**
- $v_0 \in \mathbb{R}$ initial value of all state pairs.
- α_k **learning rate** for update step k, e.g., $\alpha_k := 1/k$.
- new_process() sets up a new process.
- execute_action(s, a) executes action a in process in state s.

Outline

Action Value Function

Given

- an MDP (p, r),
- a value aggregation function $\text{value}(r_1, r_2, \ldots)$
  - e.g., for discounted criterion: $\text{value}(r_1, r_2, \ldots) := r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots)$
- a policy π:

value function (review):

$$V^\pi(s) := \mathbb{E}(\text{value}(r_0, r_1, \ldots, r_t, \ldots) \mid s_0 = s, \pi), \quad s \in S$$

action value function:

$$Q^\pi(s, a) := \mathbb{E}(\text{value}(r_0, r_1, \ldots, r_t, \ldots) \mid s_0 = s, a_0 = a, \pi)$$

- value when taking action a in state s
  - regardless what policy π would do
- and afterwards following policy π.

Optimal Action Value Function

action value function:

$$Q^\pi(s, a) := \mathbb{E}(\text{value}(r_0, r_1, \ldots, r_t, \ldots) \mid s_0 = s, a_0 = a, \pi)$$

Bellman equation for optimal action value function:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) \max_{a'} Q^*(s', a'), \quad s \in S, a \in A$$

Optimal value function from optimal action value function:

$$V^*(s) = \max_a Q^*(s, a), \quad s \in S$$

Optimal policy from optimal action value function:

$$\pi^*(s) = \arg\max_a Q^*(s, a), \quad s \in S$$

Monte Carlo for the Action Value Function

Monte Carlo method Action Value update rule:

$$\hat{Q}_{s_t,a_t} := \hat{Q}_{s_t,a_t} + \alpha_{k(s_t,a_t)}(r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} - \hat{Q}_{s_t,a_t}),$$
$$t = 0 : T-1$$

- update values of all visited **state/action pairs** after completion of each sequence $s, a, r, T := s_n, a_n, r_n, T_n$.
  - as it has to compute the observed value from all future rewards
- $k(s, a)$ keeps track of the frequency of the state/action pair (s, a) seen so far.

Monte Carlo Action Value Function Learning Algorithm

learn-action-value-discounted-mc$(S, A, \gamma, s_{\text{term}}, N, \pi, q_0, \alpha)$:

  $\hat{Q} := (q_0)_{s \in S, a \in A}, \quad k := (0)_{s \in S, a \in A}$

  for $n := 1, \ldots, N$:

    $(s, a, r, T) :=$ generate-episode$(S, A, s_{\text{term}}, \pi)$

    for $t := 0, \ldots, T-1$:

      $\hat{Q}_{s_t, a_t} := \hat{Q}_{s_t, a_t} + \alpha_{k(s_t, a_t)}((\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}) - \hat{Q}_{s_t, a_t})$

      $k(s_t, a_t) := k(s_t, a_t) + 1$

  return $\hat{Q}$

where

- s_{term} **terminal state** with zero reward.
- π **generating policy**
- $q_0 \in \mathbb{R}$ initial value of all state/action pairs.
- α_k **learning rate** for update step k, e.g., $\alpha_k := 1/k$.

Generating Policies II

- **uniform at random policy**:

$$\pi_{\text{uniform}}(s, a) := \frac{1}{|A|}$$

  - maximally explores all actions.

- **greedy policy**:

$$\pi_{\text{greedy}}(s; \hat{Q}) := \arg\max_{a \in A} \hat{Q}(s, a)$$

  - maximally exploits current estimates; does not guarantee exploration.

- **ϵ-greedy policy**:

$$\pi(s, a; \hat{Q}, \epsilon) := (1 - \epsilon)\, \pi_{\text{greedy}}(s, a; \hat{Q}) + \epsilon\, \pi_{\text{uniform}}(s, a), \quad \epsilon \in [0, 1]$$

- **Boltzmann at random policy**:

$$\pi(s, a; \hat{Q}, \tau) := \frac{e^{\hat{Q}(s,a)/\tau}}{\sum_{a' \in A} e^{\hat{Q}(s,a')/\tau}}, \quad \tau \to 0$$

Note: $\pi_{\text{greedy}}(s, a; \hat{Q}) := \mathbb{I}(a = \arg\max_{a' \in A} \hat{Q}(s, a'))$

How to Learn the Optimal Policy

- let the sampling policy π approach the greedy policy.
  - e.g., ϵ-greedy with $\epsilon \to 0$
  - or Boltzmann at random with $\tau \to 0$.
- then $\hat{Q}^{\pi} \to Q^*$

Monte Carlo Optimal Policy Learning Algorithm:

learn-opt-policy-discounted-mc$(S, A, \gamma, s_{\text{term}}, N, \pi, q_0, \alpha)$:
  $\hat{Q} :=$ learn-action-value-discounted-mc$(S, A, \gamma, s_{\text{term}}, N, \pi, q_0, \alpha)$
  for $s \in S$:
    $\hat{\pi}^*_s := \arg\max_{a \in A} \hat{Q}_{s,a}$
  return $\hat{Q}, \hat{\pi}^*$

where

- s_{term} terminal state with zero reward.
- π generating policy, **depending on $\hat{Q}$, approaching the greedy policy.**
- $q_0 \in \mathbb{R}$ initial value of all state/action pairs.
- α_k learning rate for update step k, e.g., $\alpha_k := 1/k$.

Temporal difference update rule for action value function (SARSA):

$$\hat{Q}_{s_t,a_t} = \hat{Q}_{s_t,a_t} + \alpha(r_t + \gamma \hat{Q}_{s_{t+1},a_{t+1}} - \hat{Q}_{s_t,a_t})$$

- update values of a visited state after an action is taken, a reward is received and the next state and **next action has been determined**.
  - requires $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, therefore name SARSA.
- SARSA estimates the action value function,
  thus allows to infer the generating policy ("**on policy**").

SARSA Algorithm

learn-opt-policy-discounted-sarsa$(S, A, \gamma, s_{\text{term}}, N, \pi, q_0, \alpha)$:
    $\hat{Q} := (q_0)_{s \in S, a \in A}$
    for $n := 1, \ldots, N$:
        $s' :=$ new_process()
        $a' := \pi(s')$
        while $s' \neq s_{\text{term}}$:
            $s := s'$
            $a := a'$
            $(r, s') :=$ execute_action(s, a)
            $a' := \pi(s')$
            $\hat{Q}_{s,a} := \hat{Q}_{s,a} + \alpha(r + \gamma \hat{Q}_{s',a'} - \hat{Q}_{s,a})$
    for $s \in S$:
        $\hat{\pi}^*_s := \arg\max_{a \in A} Q(s, a)$
    return $\hat{Q}, \hat{\pi}^*$

where

- s_{term} terminal state with zero reward.
- π generating policy, **depending on $\hat{Q}$, approaching the greedy policy.**
- $q_0 \in \mathbb{R}$ initial value of all state/action pairs.
- α_k learning rate for update step k, e.g., $\alpha_k := 1/k$.
- new_process() sets up a new process.
- execute_action(s, a) executes action a in process in state s.

Summary

- **Reinforcement learning** aims to learn the optimal policy for an unknown MDP (that can be sampled from / operated).
- **Monte Carlo Methods** estimate the value function V^π of a policy based on single, complete episodes.
- **Temporal Difference Methods** update the value function faster already after each individual action / step.
- The optimal **action value function** Q^* allows to reconstruct the optimal policy without knowledge of the transition model.
  - while the optimal value function V^* requires the transition model, too.
- Both, Monte Carlo and Temporal Difference Methods can also be used to learn the action value function Q (**SARSA** algorithm).

Further Readings

- Reinforcement Learning:
  - Olivier Sigaud, Frederick Garica (2010): *Reinforcement Learning*, ch. 1 in Sigaud and Buffet [2010].
  - Sutton and Barto [2018]

References

Olivier Sigaud and Olivier Buffet, editors. *Markov Decision Processes in Artificial Intelligence*. Wiley, 2010.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, 2nd edition edition, 2018.