Tutorial
Probabilistic Artificial Intelligence

Reinforcement Learning

Charlotte Bunne

ETH Zurich

Winter Term 2019

Overview:

- **Part 0**: MDPs and Bellman Equations
- **Part 1**: Model-Based Reinforcement Learning
- **Part 2**: Model-Free Reinforcement Learning
- **Part 2**: (Cool) Example of Q-Learning
- **Part 3**: Discussion of Homework

MDPs and Bellman Equations

Markov Decision Process (MDPs)

A MDP is defined by . . .

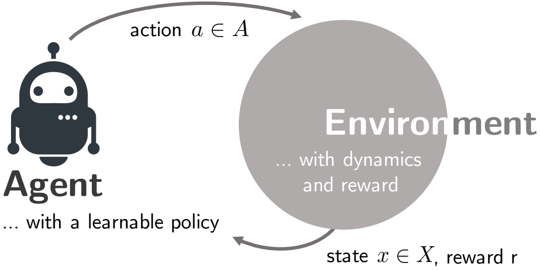
. . . a set of states $X = \{1 \dots n\}$

. . . a set of actions $A = \{1 \dots m\}$

. . . transition probabilities $p(x'|x, a)$

. . . a reward function r

Planning via a (deterministic) policy
$\pi : X \to A$



Review by Kaelbing, Littman and Moore (1996)

Several optimality criteria exist:

Finite Horizon: $\mathbb{E}\left(\sum_{t=0}^{T} r_t\right)$

Average Reward: $\lim_{h\to\infty} \mathbb{E}\left(\frac{1}{h}\sum_{t=0}^{\infty} r_t\right)$

Infinite Horizon: $\mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$

- discount factor $0 \leq \gamma \leq 1$ which quantifies importance of future rewards

Value Function

Find the optimal policy via $V^*(x) = \max_\pi \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$ (optimality criterium)

Bellman equation (1957) defines *recursively* the **value function** of policy π at state x:

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} p(x'|x, \pi(x)) V^\pi(x')$$

- Dynamic Programming

Assumption: transition probabilities and reward function are known

Bellman Theorem

Value Function V^π

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{x'} p(x'|x, \pi(x)) V^\pi(x')$$

$$\updownarrow$$

Greedy Policy w.r.t. V

$$\arg\max_a r(x, a) + \gamma \sum_{x'} p(x'|x, a) V(x')$$

Value Iteration Algorithm

Value iteration computes the optimal state value function by iteratively improving estimate of $V(x)$

Algorithm 1 Pseudocode for Value Iteration Algorithm

```
input A, X, r, p
initialize V_0 to arbitrary values
while not converged do
    for x ∈ X do
        for a ∈ A do
            Q_t(x, a) ← r(x, a) + γ Σ_{x'} p(x'|x, a) V_{t-1}(x')
        end for
        V_t(x) ← max_a Q_t(x, a)
    end for
end while
return V_T
```

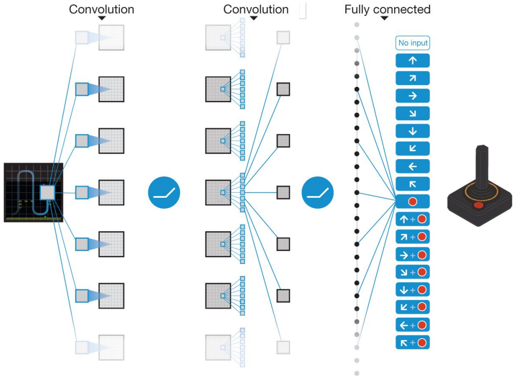
$$Q_t(x,a) \leftarrow r(x,a) + \gamma \sum_{x'} p(x'|x,a)V_{t-1}(x')$$

$$V_t(x) \leftarrow \max_a Q_t(x,a)$$

= Planning in unknown MDPs

Challenges:

- Exploration-Exploitation Dilemma
- Credit Assignment Problem
- Function Approximation



DQN [Minh et al. (2015)]

Approaches:

Model-Based Reinforcement Learning
Model-Free Reinforcement Learning

On-Policy Reinforcement Learning
Off-Policy Reinforcement Learning

Common Approaches:

1. ϵ-Greedy

$$\pi = \begin{cases} \arg\max_a r(a), & \text{with prob. } 1-\epsilon \\ \texttt{rand}(a), & \text{with prob. } \epsilon \end{cases}$$

Common Approaches:

1. ϵ-Greedy

$$\pi = \begin{cases} \arg\max_a r(a), & \text{with prob. } 1-\epsilon \\ \mathtt{rand}(a), & \text{with prob. } \epsilon \end{cases}$$

   - ▷ if ϵ satisfies **Robbins Monro (RM)** conditions ($\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$), it converges to optimal policy with probability 1

Common Approaches:

1. ϵ-Greedy

$$\pi = \begin{cases} \arg\max_a r(a), & \text{with prob. } 1-\epsilon \\ \texttt{rand}(a), & \text{with prob. } \epsilon \end{cases}$$

- if ϵ satisfies **Robbins Monro (RM)** conditions ($\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$), it converges to optimal policy with probability 1
- performs quite well **but** does not quickly eliminate clearly suboptimal actions

Exploration-Exploitation Dilemma

Common Approaches:

1. ϵ-Greedy

$$\pi = \begin{cases} \arg\max_a r(a), & \text{with prob. } 1-\epsilon \\ \mathtt{rand}(a), & \text{with prob. } \epsilon \end{cases}$$

2. Optimistic Initialization: *optimism under uncertainty*

Common Approaches:

1. ϵ-Greedy

$$\pi = \begin{cases} \arg\max_a r(a), & \text{with prob. } 1-\epsilon \\ \texttt{rand}(a), & \text{with prob. } \epsilon \end{cases}$$

2. Optimistic Initialization: *optimism under uncertainty*
3. Upper Confidence Bound (UCB)
4. Posterior Sampling

Model-Based Reinforcement Learning

Strategy: Learn MDP by estimating the transition probabilities and reward function from observed variables $< x, a, r, x' >$

R-max Algorithm

1. Start with an optimistic MDP of the environment
   - ▷ *optimism under uncertainty*: initialize $r = R_{max}$ and $p(x^*|x, a) = 1$ for $\forall x \in X, a \in A$, with x^* being a fictitious state

R-max Algorithm

1. Start with an optimistic MDP of the environment
   - *optimism under uncertainty*: initialize $r = R_{max}$ and $p(x^*|x, a) = 1$ for $\forall x \in X, a \in A$, with x^* being a fictitious state

     implicit explore or exploit approach

1. Start with an optimistic MDP of the environment
   - ▷ *optimism under uncertainty*: initialize $r = R_{max}$ and $p(x^*|x, a) = 1$ for $\forall x \in X, a \in A$, with x^* being a fictitious state
2. Solve for optimal policy π in optimistic MDP (for given r and p)

R-max Algorithm

1. Start with an optimistic MDP of the environment
   - *optimism under uncertainty*: initialize $r = R_{max}$ and $p(x^*|x, a) = 1$ for $\forall x \in X, a \in A$, with x^* being a fictitious state
2. Solve for optimal policy π in optimistic MDP (for given r and p)
3. Take greedy action according to policy π
4. Update MDP by estimating p and updating r
5. Repeat (from 2)

R-max Algorithm

1. Start with an optimistic MDP of the environment
    - *optimism under uncertainty*: initialize $r = R_{max}$ and $p(x^*|x, a) = 1$ for $\forall x \in X, a \in A$, with x^* being a fictitious state
2. Solve for optimal policy π in optimistic MDP (for given r and p)
3. Take greedy action according to policy π
4. Update MDP by estimating p and updating r
5. Repeat (from 2)

Implicit explore or exploit approach

Agent acts greedily according to a model that assumes all *underexplored* states are maximally rewarding

Analysis of algorithm $\mathcal{A}$

Definition: Probably Approximately Correct (PAC)-MDP[1]

An algorithm $\mathcal{A}$ is said to be an efficient PAC-MDP algorithm if, for any $\epsilon > 0$ ('accuracy'), $0 < \delta < 1$ ('probability of failure'), the per-timestep computational complexity, space complexity and sample complexity of $\mathcal{A}$ are less than some polynomial in the relevant quantities $(X, A, \frac{1}{\delta}, \frac{1}{(1-\gamma)})$, with probability at least $1 - \delta$.

Result: R-max is a PAC-MDP algorithm, it is called *efficient* because its sample complexity scales only polynomially in the number of states

[1] [Strehl, Li, Littman (2009)]

Definition: Probably Approximately Correct (PAC)-MDP[1]

An algorithm $\mathcal{A}$ is said to be an efficient PAC-MDP algorithm if, for any $\epsilon > 0$ ('accuracy'), $0 < \delta < 1$ ('probability of failure'), the per-timestep computational complexity, space complexity and sample complexity of $\mathcal{A}$ are less than some polynomial in the relevant quantities (X, A, $\frac{1}{\delta}$, $\frac{1}{(1-\gamma)}$), with probability at least $1 - \delta$.

Result: R-max is a PAC-MDP algorithm, it is called *efficient* because its sample complexity scales only polynomially in the number of states

- In natural environments, however, number of states is enormous: it is exponential in the number of objects in the environment

[1][Strehl, Li, Littman (2009)]

Model-Free Reinforcement Learning

How can we solve Bellman's fixed point equation without knowing transition probabilities and reward?

$$V^*(x) = \max_a r(x, a) + \gamma \sum_{x'} p\left(x'|x, a\right) V^*(x')$$

$$\pi^*(x) \in \arg\max_a \underbrace{r(x, a) + \gamma \sum_{x'} p(x'|x, a) V^*(x')}_{Q^*(x,a)}$$

Estimate V or Q directly from samples (via stochastic approximation methods).

Temporal Difference (TD) Learning

Observe samples $< x, a, r, x' >$
Assumption: if value estimates are correct, the following must hold

$$V(x) = r + \gamma V(x')$$

Otherwise there is an error: $\delta = r + \gamma V(x') - V(x)$ (TD error)

To learn a better estimate minimize δ and with learning rate α update

$$V(x) \leftarrow V(x) + \alpha(\underbrace{r + \gamma V(x')}_{\text{TD target}} - V(x))$$

[Samuel (1959), Sutton (1984, 1988)]

Q-Learning

Q-function ($Q : X \times A \rightarrow \mathbb{R}$) quantifies quality of a state-action combination

Core of the algorithm: **value-iteration update**

- weighted average of old value and new information

$$Q(x,a) = Q(x,a) + \alpha \left(r + \gamma \max_{a'} Q(x',a') - Q(x,a) \right)$$

$$= (1-\alpha) \underbrace{Q(x,a)}_{\text{old value}} + \alpha \overbrace{\left(r + \gamma \underbrace{\max_{a'} Q(x',a')}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Stochastic Approximation used in Q-learning and TD-learning to estimate the value function

- Q-Learning: (asynchronous) implementation of the Robbins Monro algorithm for finding fixed points
- To prove convergence of Q-Learning we need results from Robbins Monro

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

[Robbins & Munro (1951)]

Robbins Monro algorithm

A classical methodology, studied by Robbins and Monro is

$$\theta_n = \theta_{n-1} - \alpha_n \left(h\left(\theta_{n-1}\right) + \varepsilon_n \right)$$

where $h : \mathbb{R}^d \to \mathbb{R}^d$, $(\varepsilon_n)_{n \geq 1}$ are i.i.d. random variables[2]

- ▷ Stochastic Gradient Descent (SGD)

Robbins-Munro: general sufficient conditions for iterates to converge,

$$\sum_{t=1}^{\infty} \alpha_t = \infty : \quad \text{convergence can be towards any point in } \mathbb{R}^d$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty : \quad \text{guarantees convergence}$$

[2] [Robbins & Munro (1951)]

Q-Learning Algorithm

Algorithm 2 Pseudocode for Q-Learning Algorithm

input actions, states, rewards, learning rate α, discount factor γ
initialize $Q : X \times A \to \mathbb{R}$
for each episode **do**
  observe initial state x
  for each step t = 0, 1, 2, … **do**
    take action a and observe r and x'
    $Q(x,a) \leftarrow (1-\alpha)Q(x,a) + \alpha\,(r + \gamma \max_{a'} Q(x',a'))$
    $x = x'$
  end for
end for
return Q

[Watkins (1989), Dayan & Watkins (1992)]

Q-Learning with Function Approximation

To generalize over states and actions, parametrize Q with a function approximation, e.g. deep neural networks

$$\delta = r + \gamma \max_{a'} Q(x', a', \theta) - Q(x, a, \theta)$$

Learning is unstable due to ***deadly triade*** [Sutton & Barto (2018)]

- ▷ off-policy learning, flexible function approximation, bootstrapping

First stable Q-learning with function approximation: DQN [Minh et al. (2015)]

[Watkins (1989), Riedmiller (2000, 2005)]

Solution of Homework

Slides based on material accompanying the textbook 'AI: A Modern Approach' (3rd edition) by S. Russell and P. Norvig and the **NeuRIPS 2019 Tutorial** by Katja Hofmann.