Reinforcement Learning

Michèle Sebag ; TP : Diviyan Kalainathan
TAO, CNRS − INRIA − Université Paris-Sud

Nov. 20th, 2017

Credit for slides: R. Sutton, F. Stulp

RL Milestones

Markov Decision Process

- State space S — Terminal states $T \subset S$
- Action space A
- Transition $p(s, a, s')$: probability of arriving in s' after doing a in s
- Reward $r(s, a)$: goodies for doing a in s
- Discount factor $\gamma < 1$ — sometimes, $r(s)$: just for being in s

General settings

	Model-based	Model-free
Finite	**Dynamic Programming**	Discrete RL
Infinite	(optimal control)	Continuous RL

Overview

Dynamic Programming, followed

Discrete Model-Free RL

Temporal difference

Dyna

Value functions

Cumulative rewards (trajectory: $rs_0, a_0, r_0, s_1, a_1, \ldots$)

$$R_t = r_0 + \gamma r_1 + \ldots + \gamma^k r_k + \ldots$$

$$= \sum_{k=0}^{\infty} \gamma^k r_k$$

Value function associated to π $a_i = \pi(s_i)$

$$V_\pi(s) = \mathbb{E}[R_t | s_0 = s]$$

Bellman equation

$$V_\pi(s) = \mathbb{E}[r(s)] + \gamma \sum_{s'} p(s, \pi(s), s')\, V_\pi(s')$$

Goal

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

Dynamic Programming for RL

- Given π
- Policy evaluation: build V_π
- Policy improvement: build π'
- Iterate

Can we go faster ?
Don't wait until convergence.

Policy evaluation, 1

Truncate at k time steps

$$V_{\pi,k}(s) = \mathbb{E}\left[\sum_{\ell=1}^{k} \gamma^{\ell} r_{\ell} | s_0 = s\right]$$

$$lim_{k\to\infty} V_{\pi,k}(s) = V_{\pi}(s)$$

($V_{\pi,k}$ is an approximation of V_{π}; can we bound the approximation error ?)

Policy evaluation, 2

Given policy π

Init

$$\forall s \in S, V_\pi(s) = 0$$

Loop

$$\begin{array}{ll} \Delta = 0 & \\ \text{For each} & s \in S \\ & v = V(s) \\ & V(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s')\, V(s') \\ & \Delta = max(\Delta, |v - V(s)|) \end{array}$$

Until $\Delta < \varepsilon$

Output $V \approx V_\pi$

Policy Improvement

Intuition

- Build $V_\pi(s)$
- You are in s
- This is the model-based setting
- Can you think of better than doing $\pi(s)$?

Policy Improvement

Intuition

- Build $V_\pi(s)$
- You are in s
- This is the model-based setting
- Can you think of better than doing $\pi(s)$?

Improved π'

$$\pi'(s) = \arg\max_a \left\{ p(s, a, s')\, V_\pi(s') \right\}$$

Algorithm

1. Define π
2. Build V_π
3. π': Policy improvement(π)
4. $\pi = \pi'$; Goto 2

This converges toward optimal π^* but takes for ever

Policy iteration

Principle

- Modify π step 1
- Update V until convergence step 2

Getting faster

- Don't wait until V has converged before modifying π.

Value Iteration

Policy evaluation recall

$$V_{\pi,k+1}(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s')\, V_{\pi,k}(s')$$

Value iteration more greedy

Value Iteration

Policy evaluation recall

$$V_{\pi,k+1}(s) = r(s) + \gamma \sum_{s'} p(s,\pi(s),s')\, V_{\pi,k}(s')$$

Value iteration more greedy

$$V_{k+1} = r(s) + \gamma \arg\max_a \sum_{s'} p(s,a,s')\, V_k(s')$$

Policy evaluation vs Value iteration

	Policy evaluation	Value iteration
Init	π	V
loop	$a = \pi(s)$	$a = argmax$
Output	V_π	Greedy policy (V)

Initialization

Random ?

- Educated initialisation is better
- See Inverse Reinforcement Learning
- https://www.youtube.com/watch?v=0JL04JJjocc
- https://www.youtube.com/watch?v=VCdxqn0fcnE
- More: ICML 2004, Pieter Abbeel and Andrew Ng

Discussion: Policy and value iteration

Similarities

- Must wait until the end of the episode
- Episodes might be long

Differences

- Policy iteration: $\pi \to V^{\pi} \to \text{Greedy}(V^{\pi})$ and iterate
- Value iteration: Interleave value computation and policy improvement.
- (More efficient: prioritized sweeping; focus on states with changing values and their neighbors)

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^{\pi}$

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Dynamic programming, summary

Policy evaluation

$$V_{k+1}(s) = r(s) + \gamma \sum_{s'} p(s, a, s') V_k(s') \text{ with } a = \pi(s)$$

Policy improvement

$$\pi(s) = \arg\max_a \left\{ r(s) + \gamma \sum_{s'} p(s, a, s') V(s') \right\}$$

Value iteration

$$V_{k+1}(s) = r(s) + \gamma max_a \left\{ \sum_{s'} p(s, a, s') V_k(s') \right\}$$

Policy iteration converges toward the optimum

$$\pi^*(s) = \arg\max_a \left\{ r(s) + \gamma \sum p(s, a, s') V^*(s') \right\}$$

Why doesn't this work in Model-Free setting ?

The model-free world

- p, transition model, is unknown
- Some effort must be put on estimating p
- The exploration vs exploitation dilemma (don't be greedy; or not always...)
- The EvE dilemma: more later (the Multi-Armed Bandit course)

Overview

Dynamic Programming, followed

Discrete Model-Free RL

Temporal difference

Dyna

This course

MDP Main Building block

General settings

	Model-based	Model-free
Finite	Dynamic Programming	**Discrete RL**
Infinite	(optimal control)	Continuous RL

Discrete Model-Free RL

Fundamentals

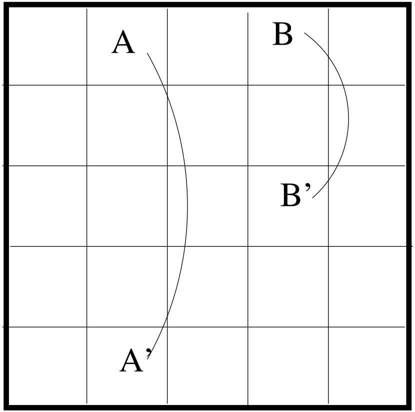
- Estimate values from interactions with environment
- Only rely on observations
- What you do (actions) influences what you see (states)

Key questions

- WHAT: a new definition of value
- HOW
  - Monte-Carlo: from episodes
  - Temporal Differences: after each action

Monte-Carlo estimations

Random π



A –> A', reward 10

B –> B', reward 5

- Start in s
- generate an episode with random π
- get the return for the episode
- average over several episodes: call it $\hat{V}(s)$

Monte-Carlo estimations, 2

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 $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Finally: After a long time, we'll have an estimate $\hat{V}(s)$ for all states (assuming well behaved MDP, i.e. we can go everywhere from anywhere).

Question: Can we apply Policy improvement ?

Monte-Carlo estimations, 2

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 $Returns(s)$
$V(s) \leftarrow$ average$(Returns(s))$

Finally: After a long time, we'll have an estimate $\hat{V}(s)$ for all states (assuming well behaved MDP, i.e. we can go everywhere from anywhere).
Question: Can we apply Policy improvement ?

No ! we do not know $p(s, a, s')$

Then change the goal: Another value function

Define

$$Q_\pi(s,a) = \mathbb{E}[R | s_0 = s; a_0 = a, a_t = \pi(s_t)]$$

Start in s, set first action to a, use π ever after.

Algorithm Monte-Carlo Q

- Initialize a list of returns for each pair (s,a)
- Add the return after each trajectory.
- Average $\rightarrow \hat{Q}(s,a)$

Greedifying Q

$$\pi_{\hat{Q}}(s) = \arg\max_a \hat{Q}(s,a)$$

The exploration vs exploitation dilemma

Exploration only

- Use $\pi = \text{random}$
- Your estimation of $\arg\max_a Q(s, a)$ will
  - be good ?
  - when ?

Exploitation only

- Build $Q(s, a)$
- Use $\pi(s) = \arg\max_a Q(s, a)$
- ... would be good if $Q(s, a)$ were good...

Finding a Trade-off

Example

Goal: go to a restaurant

Exploration: select a random one

Exploitation: select the one with best advices on *La Fourchette*

ε-greedy

- With proba $1 - \varepsilon$, exploitation
- With proba ε, exploration

Decreasing ε

- After each episode $\varepsilon \to \beta\varepsilon$, with $\beta < 1$.

Finally

1. π is ε-greedy wrt Q
2. Use π to build Q
3. Decay ε

PROS

- Learns Q-values from observed returns (doesnt require a model)
- Estimates become better over time with more experience
- Can choose the best action as $\arg\max_a Q(s, a)$
- Starts out with exploration ($\varepsilon = 1$), but slowly becomes greedy ($\varepsilon = 0$)

CONS

- requires a lot of experience to get good estimates and policy
- works for small finite MDPs only
- applicable to episodic problems only
- wastes time evaluating bad policies

Overview

Dynamic Programming, followed

Discrete Model-Free RL

Temporal difference

Dyna

Temporal differences

Principle

- Monte Carlo: updates values after an episode is done (based on returns $R = \sum_t r_t$
- Temporal-difference learning: updates values after each step (based on immediate reward r_t)

PRO

- Faster

CONS

- Possibly brittle

Intermediate possibility: Incremental Monte-Carlo

Batch

$$V_\pi(s) = \frac{1}{N}\sum_{k=1}^{N} R^{(k)}(s)$$

Where

N is the number of episodes
$R^{(k)}(s)$ is the sum of discounted rewards gathered after first visit to s in k-th episode.

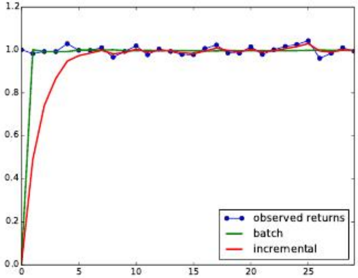
Incremental update

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha \left(R^{(k)}(s) - V_\pi(s) \right)$$

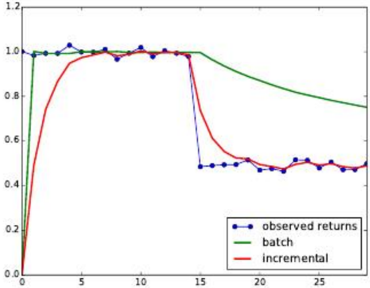
Where

α is the learning rate
(What happens for $\alpha = 0$? for $\alpha = 1$?)

Incremental Monte-Carlo, 2



stationary setting



non-stationary setting

Temporal Differences, with V

Main equation

$$
\begin{aligned}
V_\pi(s_t) &\leftarrow V_\pi(s_t) + \alpha\left(R^{(k)}(s) - V_\pi(s)\right) \\
&= (1-\alpha)V_\pi(s_t) + \alpha\left(R^{(k)}(s) - V_\pi(s)\right) \\
&= (1-\alpha)V_\pi(s_t) + \alpha\left(r(s_t) + \gamma V_\pi(s_{t+1})\right)
\end{aligned}
$$

Algorithm

1. Initialize V and π
2. Loop on episode
   2.1 Initialize s
   2.2 Repeat

$$
\begin{array}{l}
\text{Select action } a = \pi(s) \\
\text{Observe } s' \text{ and reward } r \\
V(s) \leftarrow V(s) + \alpha(\underbrace{r + \gamma V(s')}_{R} - V(s)) \\
s \leftarrow s'
\end{array}
$$

   2.3 Until s' terminal state

Unroll the algorithm: https://www.youtube.com/watch?v=DZzffdHNqtQ

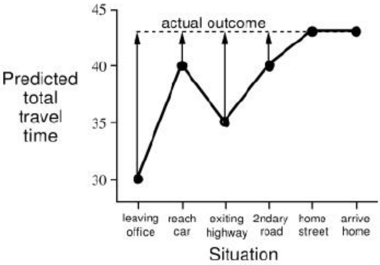
Why is this useful ?

Policy and value iteration

- Must wait until the end of the episode
- Episodes might be long

We can update V on the fly:

- I have estimates of how long it takes to go to RER, to catch the train, to arrive at Cité-U
- Something happens on the way (bump into a friend, chat, delay, miss the train,...)
- I can update my estimates of when I'll be home...



Monte-Carlo

Temporal Differences

Temporal Differences, with Q

Main equations

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

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\left(r(s_t) + \gamma Q(s_{t+1}, a_{t+1})\right)$$

Input: tons of $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ and the name of the algorithm is SARSA

- These 5-tuples are either gathered using the current policy — on-policy
- Or, are reused from other trajectories — off-policy

Algorithm

1. Initialize Q
2. Loop on episode
   2.1 Initialize s
   2.2 Repeat

$$\begin{array}{l} \text{Select action } a = \varepsilon\text{-Greedy}(s, Q) \\ \text{Observe } s' \text{ and reward } r \\ Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{r + \gamma Q(s', a')}_{R} - Q(s,a)) \\ s \leftarrow s' \end{array}$$

   2.3 Until s' terminal state

Discussion

Update on the spot ?

- Might be brittle
- Instead one can consider several steps

$$R = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$$

Find an intermediate between

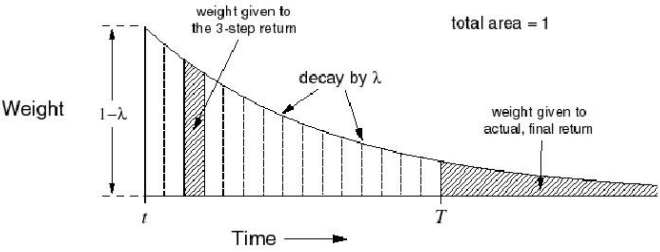
- Policy iteration

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

- TD(0)

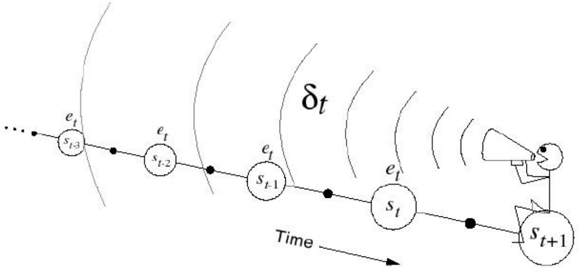
$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

TD(λ), intuition



$$R_t^\lambda = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)}}_{} + \underbrace{\lambda^{T-t-1}R_t}_{}$$

TD(λ), intuition, followed



$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

TD(λ)

1. Initialize V and π
2. Loop on episode
   2.1 Initialize s
   2.2 Repeat

$a = \pi(s)$
Observe s' and reward r
$\delta \leftarrow r + V(s') - V(s)$
$e(s) \leftarrow e(s) + 1$

For all s"
$V(s'') \leftarrow V(s") + \alpha \delta e(s'')$
$e(s'') \leftarrow \gamma \lambda e(s'')$

$s \leftarrow s'$

   2.3 Until s' terminal state

Q-learning

Principle: combine temporal difference and value iteration
Iterate

- During an episode (from initial state until reaching a final state)
- At some point explore and choose another action;
- If it improves, update $Q(s, a)$:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \left[\overbrace{\underbrace{r(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

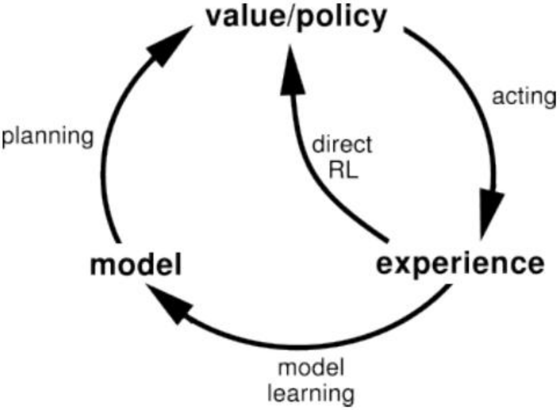
Equivalent to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha) + \alpha[r(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$

Overview

A few slides ago...

From episodes, we can estimate V....
Question: Can we apply Policy improvement ?
No: we do not know $p(s, a, s')$
Aha ! we can learn it...



Dyna Algo

Algorithm

Initialize $Q(s,a)$, $\hat{p}$, $\hat{r}$ for all s, a

Loop (while budget not exhausted)

1. $s =$ current state
2. $a = \varepsilon$-greedy(s, Q)
3. Do action a, arrive in state s' with reward r
4. Update Q:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha \left(r + max_{a'} Q(s',a')\right)$$

5. Update $\hat{p}$ and $\hat{r}$ from s' and r'
6. Repeat N times
   - 6.1 Select z previous state
   - 6.2 Select b action taken in z
   - 6.3 Estimate s', r' from $\hat{p}, \hat{r}$
   - 6.4 Update Q

$$Q(z,b) = (1-\alpha)Q(z,b) + \alpha \left(r' + max_{a'} Q(s',a')\right)$$

Discussion

1. s = current state
2. $a = \varepsilon$-greedy(s, Q)
3. Do action a, arrive in state s' with reward r
4. Update Q: **With real experience**

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r + max_{a'}Q(s',a')\right)$$

5. Update $\hat{p}$ and $\hat{r}$ from s' and r' **With real experience**
6. Repeat N times
   6.1 Select z previous state
   6.2 Select b action taken in z
   6.3 Estimate s', r' from $\hat{p}, \hat{r}$
   6.4 Update Q

$$Q(z,b) = (1-\alpha)Q(z,b) + \alpha\left(r' + max_{a'}Q(s',a')\right)$$

Discussion, 2

1. s = current state
2. $a = \varepsilon$-greedy(s, Q)
3. Do action a, arrive in state s' with reward r
4. Update Q: **using real experience**

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r + max_{a'} Q(s',a')\right)$$

5. Update $\hat{p}$ and $\hat{r}$ from s' and r' **using real experience**
6. Repeat N times
    6.1 Select z previous state
    6.2 Select b action taken in z
    6.3 Estimate s', r' from $\hat{p}, \hat{r}$
    6.4 Update Q **using simulated experience**

$$Q(z,b) = (1-\alpha)Q(z,b) + \alpha\left(r' + max_{a'} Q(s',a')\right)$$

Discussion, 3

TD-Learning **as in real life**

$$\boxed{Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(r + max_{a'}Q(s',a')\right)}$$

Dynamic programming **as when dreaming**

Repeat N times

1. Select z previous state
2. Select b action taken in z
3. Estimate s', r' from $\hat{p}, \hat{r}$
4. Update Q

$$Q(z,b) = (1-\alpha)Q(z,b) + \alpha\left(r' + max_{a'}Q(s',a')\right)$$