Reinforcement learning II

Tomáš Svoboda

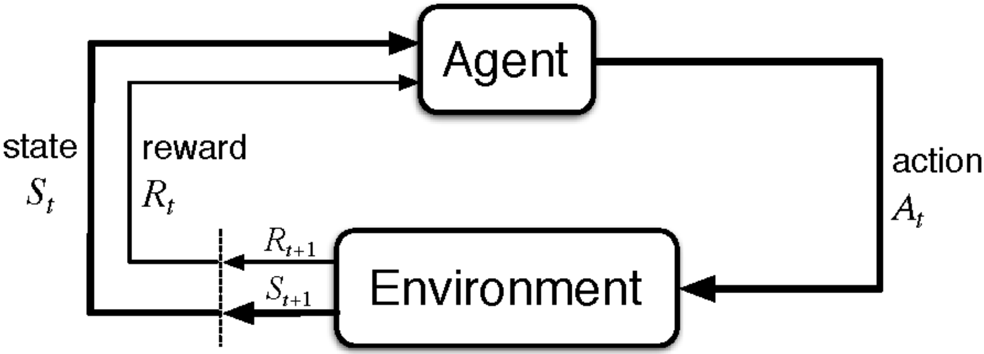
Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

April 23, 2020

Notes

Not all slides with notes. What can be noted about the title page, eh?

Recap: Reinforcement Learning



[1]

- Feedback in form of Rewards
- Learn to act so as to maximize sum of expected rewards.
- In `kuimaze` package, `env.step(action)` is the method.

[1] Scheme from [2]

Notes

Robot/agent action changes environment.

- Environment is everything ...
- battery state
- robot position
- object position (manipulation task)
- ...

Learning to control flippers

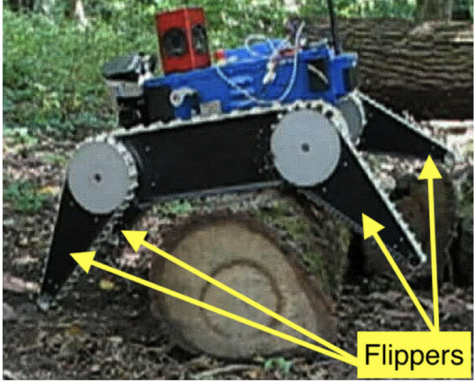
- What are the states?
- How to design rewards?
- How to perform training episodes (roll-outs)?
- Simulator to reality gap.

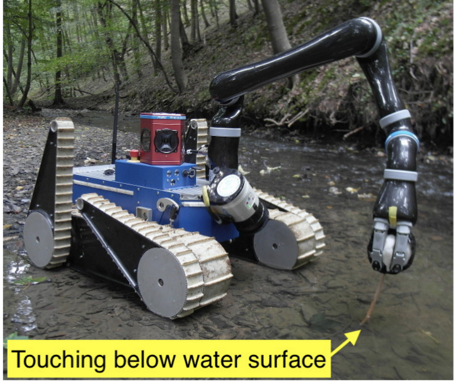
http://cyber.felk.cvut.cz/vras/

Notes

- States may contain interoceptive as well as exteroceptive sensing.
- Reward shaping.
- Train in simulator, then go for a real roll-out, back to simulator and so on.
- Physical simulator for robot terrain interactions.
- Sensor models.

Next few slides display a possible parameterization of the flipper control task.





- Construction: $2\times$ main tracks, $4\times$ subtracks (flippers), differential break
  great stability and climbing capability
- Sensor suite: SICK LMS-151 range finder, Ladybug omnicam, Xsens MTi-G IMU
  3D sensing and localization
- Control inputs: Velocity vector, $4\times$**flipper angle**, $4\times$ **flipper stiffness**, differential break (0/1)
  difficult to control all of them manually!

Notes

This and the next three slides introduce some ideas and approaches published in:

- Karel Zimmermann, Petr Zuzanek, Michal Reinstein, Vaclav Hlavac. Adaptive traversability of unknown complex terrain with obstacles for mobile robots. In 2014 IEEE international conference on robotics and automation (ICRA).

The work has been exteneded in several directions:

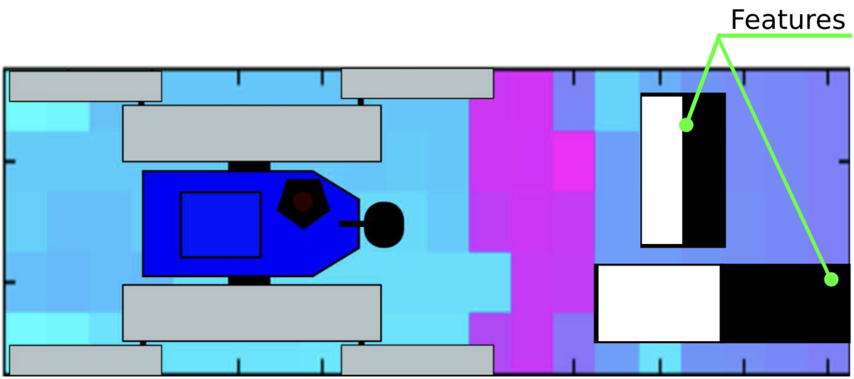
- Martin Pecka, Vojtěch Šalanský, Karel Zimmermann, Tomáš Svoboda. Autonomous flipper control with safety constraints. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
- M. Pecka, K. Zimmermann, M. Reinstein, and T. Svoboda. Controlling Robot Morphology from Incomplete Measurements. In *IEEE Transactions on Industrial Electronics*, Feb 2017, Vol 64, Issue: 2, pp. 1773-1782
- M. Pecka, K. Zimmermann, T. Svoboda. Fast Simulation of Vehicles with Non-deformable Tracks. In Intelligent Robots and Systems (IROS), 2017.
- Martin Pecka, Karel Zimmermann, Matěj Petrlík, Tomáš Svoboda. Data-driven Policy Transfer with Imprecise Perception Simulation. *IEEE Robotics and Automation Letters*, Vol. 3, Issue 4, Oct 2018

State $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$ concatenates:

- Proprioceptive measurements: roll, pitch, torques, velocity, acceleration.

- Local exteroceptive measurements.

features on digital elevation map with fixed size.

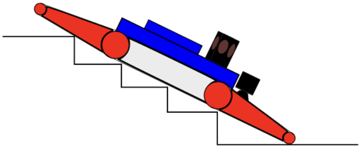


Notes

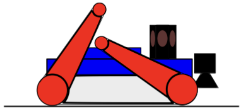
- Colors encode height of the terrain
- Haar's features: Think about sum of heights in white are - sum of heights in the black area.

Instead of $\mathbf{a} \in \mathcal{A} \subset \mathbb{R}^8$ we consider only 5 configurations[2]:

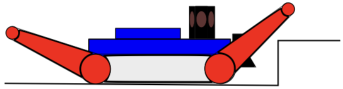
$$\mathcal{A} = \{\text{I-shape}\,,\text{V-shape}\,,\text{L-shape}\,,\text{U-shape soft}\,,\text{U-shape hard}\}$$



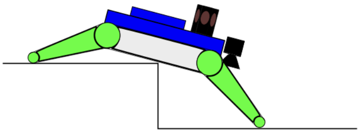
I-shape
(Maximizes traction)



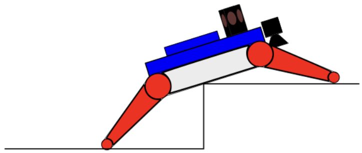
V-shape
(Provides observability)



L-shape
(Forward approach)



U-shape soft
(Smooth climbing down)



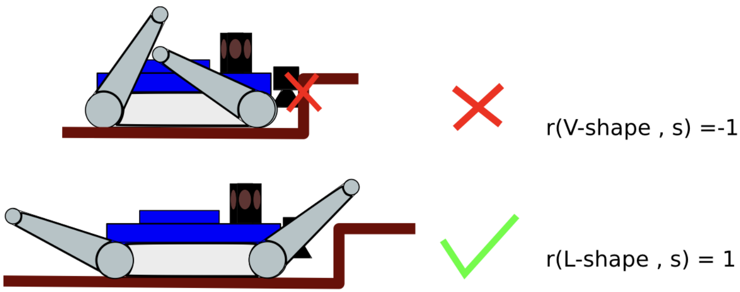
U-shape hard
(Lifts the body up)

Notes

- Discretization of the Action space.
- Colors of the flippers encode wheather they are stiff (red) or soft – terrain compliant (green).

Reward $r(a, \mathbf{s}) : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a weighted sum of following contributions:

1. Safe pitch and roll reward, avoiding tipping over
2. Smoothness reward, suppresses body hits
3. Speed reward, drives robot forward
4. User denoted reward (penalty) indicating the success (failure) of the particular maneuver indicates failure/possible damages



Notes

- Hand-crafted reward function.

From off-line (MDPs) to on-line (RL)

Markov decision process – MDPs. Off-line search, we know:

- A set of states $s \in \mathcal{S}$ (map)
- A set of actions per state. $a \in \mathcal{A}$
- A transition model $p(s'|s, a)$ (robot)
- A reward function $r(s, a, s')$ (map, robot)

Looking for the optimal policy $\pi(s)$. We can plan/search before the robot enters the environment.

On-line problem:

- Transition p and reward r functions not known.
- Agent/robot must act and learn from experience.

Notes

For MDPs, we know p, r for all possible states and actions.

(Transition) Model-based learning

The main idea: Do something and:

- Learn an approximate model from experiences.
- Solve as if the model were correct.

Learning MDP model:

- Try s, a, observe s', count s, a, s'.
- Normalize to get and estimate of $p(s'|s, a)$
- Discover each $r(s, a, s')$ when experienced.

Solve the learned MDP.

Notes

Model-free learning

- r, p not known.
- Move around, observe
- And learn on the way.
- **Goal:** learn the state value $v(s)$ or (better) q-value $q(s, a)$ functions.

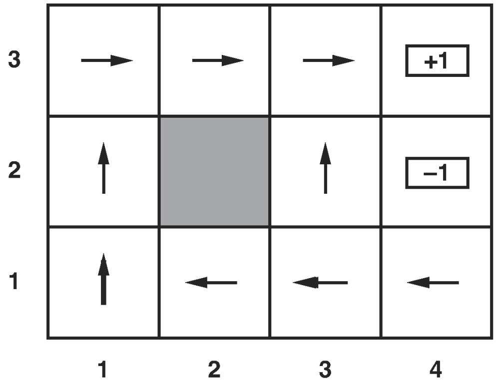


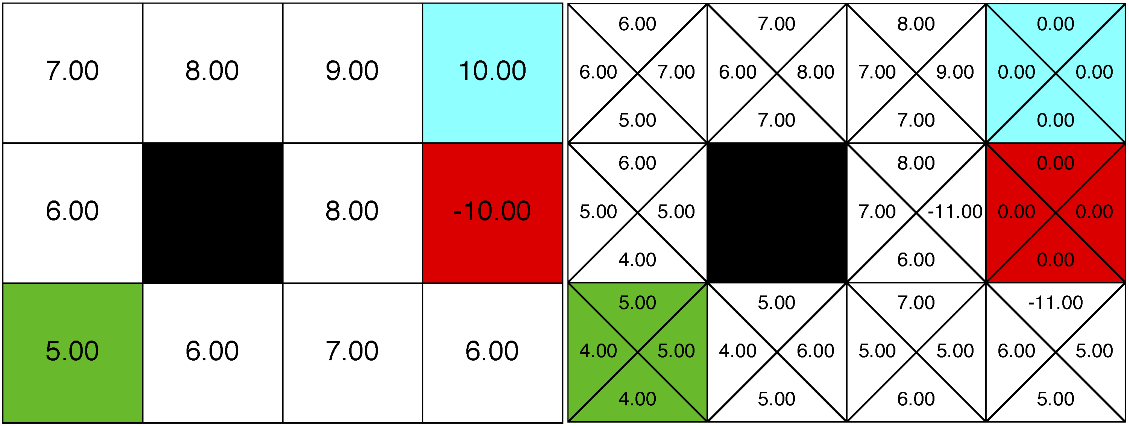
Image from [1]

Notes

Executing policies - training, then learning from the observations. We want to do the policy evaluation but the necessary model is not known.

Recap: $V-$ and $Q-$ values, converged . . .

$\gamma = 1$, rewards $-1, +10, -10$, and no confusion - deterministic robot



$$V(S_t) = R_{t+1} + V(S_{t+1})$$

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

Notes

$\gamma = 1$, Rewards $-1, +10, -10$, and no confusion - deterministic robot/agent. Rewards associated with leaving the state. Q values close next to terminal state includes the actual reward and the transition cost steping in, or better, leaving the last living state.

$Q(s, a)$ - expected sum of rewards having taken the action and acting according to the (optimal) policy.

How would the (q)values change if $\gamma = 0.9$?

Model-free TD learning, updating after each transition

- Observe, experience environment through learning episodes, collecting:

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, \ldots$$

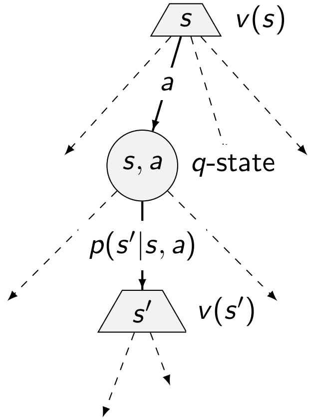
- Update by mimicking Bellman updates after each transition $(S_t, A_t, R_{t+1}, S_{t+1})$

Notes

Think about $S_t - A_t - S_{t+1} - A_{t+1} - S_{t+2}$ tree with associated rewards. Episode starts in a start state and ends in a terminal state.

Recap: Bellman optimality equations for $v(s)$ and $q(s, a)$

$$
\begin{aligned}
v(s) &= \max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma\, v(s')\right] \\
&= \max_a q(s,a)
\end{aligned}
$$



The value of a q-state (s, a):

$$
\begin{aligned}
q(s,a) &= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma\, v(s')\right] \\
&= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \max_{a'} q(s',a')\right]
\end{aligned}
$$

Notes

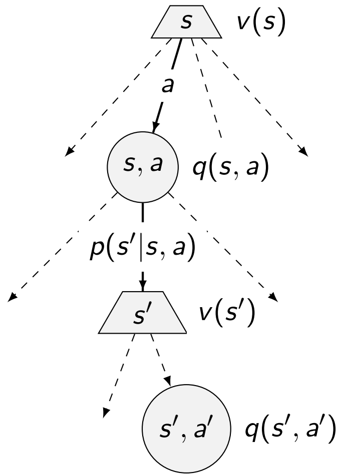
The tree continues from s' through a' and so on until it terminates

Recap: Bellman optimality equations for $v(s)$ and $q(s, a)$

$$
\begin{aligned}
v(s) &= \max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma\, v(s')\right] \\
&= \max_a q(s,a)
\end{aligned}
$$

The value of a q-state (s, a):

$$
\begin{aligned}
q(s,a) &= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma\, v(s')\right] \\
&= \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma \max_{a'} q(s',a')\right]
\end{aligned}
$$



Notes

The tree continues from s' through a' and so on until it terminates

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  $\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning

Learn Q values as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

In each step Q approximates the optimal q^* function.

Notes

There are alternatives how to compute the trial. SARSA method takes $Q(S_t, A_t)$ directly, not the max. Hence we need 5-tuples $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

Q-learning: algorithm

step size $0 < \alpha \leq 1$
initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{S}(s)$
repeat episodes:
  initialize S
  for for each step of episode: **do**
    choose A from S
    take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  end for until S is terminal
until Time is up, ...

Notes

How to select A_t in S_t?

- time t, at S_t
- take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

Notes

How to select A_t in S_t?

- time t, at S_t
- take A_t derived from Q , observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless S_t is terminal)

Notes

. . . A_t derived from Q

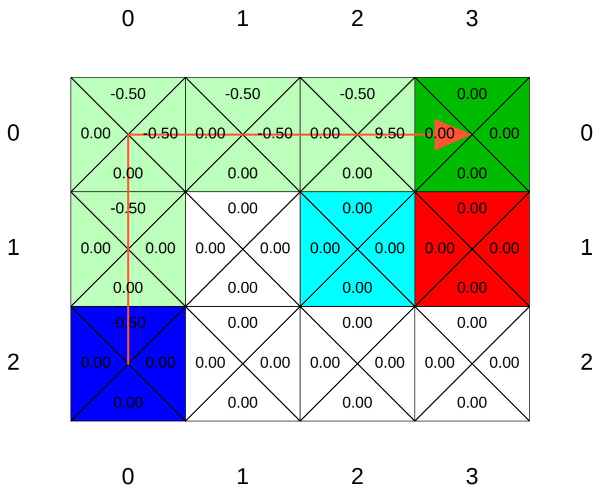
What about keeping optimality, taking max?

$$A_t = \arg\max_a Q(S_t, a)$$

see the demo run of `rl_agents.py`.

Notes

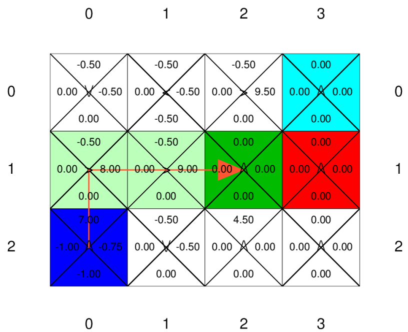
Two good goal states



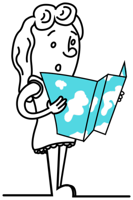
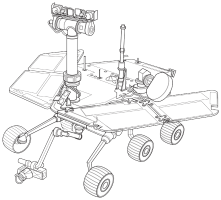
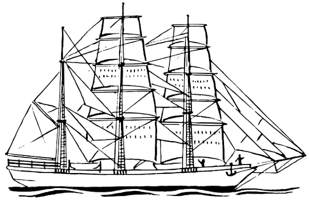
Notes

Discuss the on-line demo with two good goal states. $\gamma = 1, \alpha = 0.5$, Living reward -1, $R(1,2) = 10, R(0,3) = 20, R(1,1) = -10$. Taking the action, corresponding the max Q. If equal options, than in the $0,1,2,3$ action order. 50 training episodes. What happened?

- No exploration step: https://youtu.be/Y5yLttbkPMM
- Exploring steps involved (will be talking in a few minutes): https://youtu.be/cAr-IrawF_c



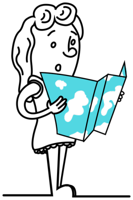
Exploration vs Exploitation

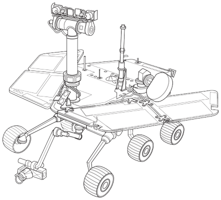




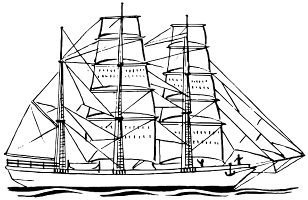
- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

Notes

Exploration vs Exploitation



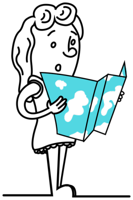


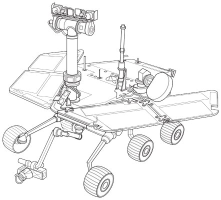


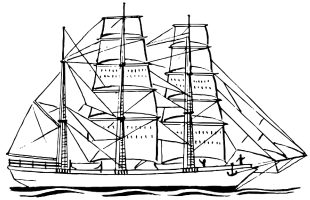
- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- . . .

Notes

Exploration vs Exploitation



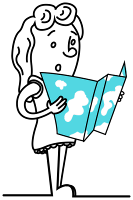
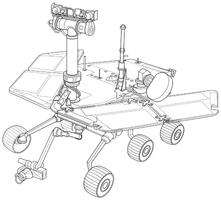
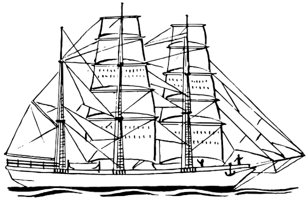




- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

Notes

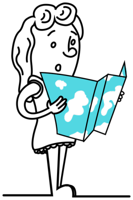
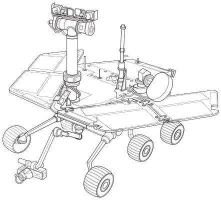
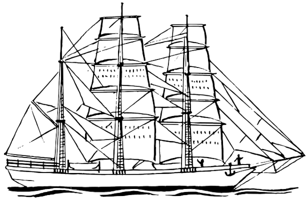
Exploration vs Exploitation

- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

Notes

Exploration vs Exploitation

- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

Notes

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1-\epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

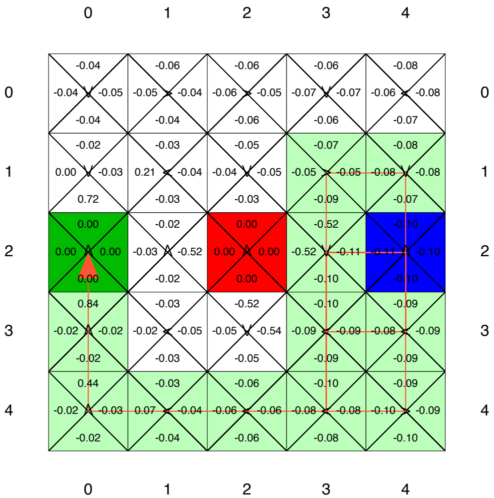
Problems with randomness?

- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

Notes

We can think about lowering ϵ as the learning progresses.

How to evaluate result, when to stop learning?

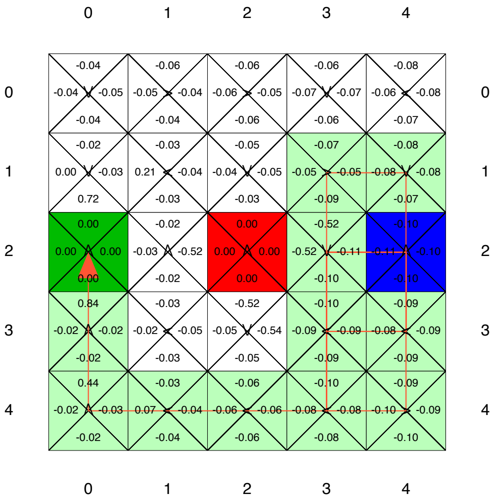


- What is the actual result of the q-learning?
- How to evaluate it?
- When to stop learning?

Notes

- Policy is the main result. Note, that policy can be poorly estimated in areas less visited.
- Evaluating policy - run the trials and compute the average Return.
- The more learning the better - think about visiting all places/states.

How to evaluate result, when to stop learning?

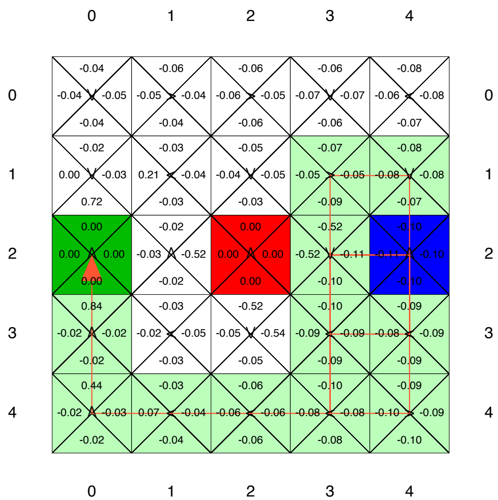


- What is the actual result of the q-learning?
- How to evaluate it?
- When to stop learning?

Notes

- Policy is the main result. Note, that policy can be poorly estimated in areas less visited.
- Evaluating policy - run the trials and compute the average Return.
- The more learning the better - think about visiting all places/states.

How to evaluate result, when to stop learning?



- What is the actual result of the q-learning?
- How to evaluate it?
- When to stop learning?

Notes

- Policy is the main result. Note, that policy can be poorly estimated in areas less visited.
- Evaluating policy - run the trials and compute the average Return.
- The more learning the better - think about visiting all places/states.

Exploration function $f(u, n)$

- Regular trial/sample estimate: trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- If (S_t, a) not yet tried, then perhaps too pesimistic.
- trial $= R_{t+1} + \gamma \max_a f\left(Q(S_{t+1}, a), N(S_{t+1}, a)\right)$

where $f(u, n)$

$$\begin{aligned} f(u, n) &= R^+ \text{ if } n < N_e \\ &= u \text{ otherwise} \end{aligned}$$

where

- R^+ is an optimistic estimate of the best possible reward obtainable in any state
- N_e fixed parameter
- The function $f(u, n)$ should be increasing in u and decreasing in n.

Notes

Will have the effect of making the agent try each action-state pair at least N_e times.

Exploration function $f(u, n)$

- Regular trial/sample estimate: trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- If (S_t, a) not yet tried, then perhaps too pesimistic.
- trial $= R_{t+1} + \gamma \max_a f\left(Q(S_{t+1}, a), N(S_{t+1}, a)\right)$

where $f(u, n)$

$$\begin{aligned} f(u,n) &= R^+ \text{ if } n < N_e \\ &= u \text{ otherwise} \end{aligned}$$

where

- R^+ is an optimistic estimate of the best possible reward obtainable in any state
- N_e fixed parameter
- The function $f(u, n)$ should be increasing in u and decreasing in n.

Notes

Will have the effect of making the agent try each action-state pair at least N_e times.

Exploration function $f(u, n)$

- Regular trial/sample estimate: trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- If (S_t, a) not yet tried, then perhaps too pesimistic.
- trial $= R_{t+1} + \gamma \max_a f\left(Q(S_{t+1}, a), N(S_{t+1}, a)\right)$

where $f(u, n)$

$$\begin{aligned} f(u, n) &= R^+ \text{ if } n < N_e \\ &= u \text{ otherwise} \end{aligned}$$

where

- R^+ is an optimistic estimate of the best possible reward obtainable in any state
- N_e fixed parameter
- The function $f(u, n)$ should be increasing in u and decreasing in n.

Notes

Will have the effect of making the agent try each action-state pair at least N_e times.

Going beyond tables - generalizing across states

	0	1	2	3	4	
0	0.84	0.88	0.92	0.96	1.00	0
	0	1	2	3	4	

Notes

We were talking about $v-$ and $q-$ *functions* but what was the representation? (look-up) Tables. Looking at $v(s)$, we need a table for each of the state!

This world is small, but think bigger!

Going beyond tables - generalizing across states

	0	1	2	3	4	
0	0.84	0.80	0.76	0.72	0.68	0
1	0.88	0.84	0.80	0.76	0.72	1
2	0.92	0.88	0.84	0.80	0.76	2
3	0.96	0.92	0.88	0.84	0.80	3
4	1.00	0.96	0.92	0.88	0.84	4
	0	1	2	3	4	

Notes

Looking a $V(s)$, we need a table for each of the state! This world is small, but think bigger!

v(s) not as table but as an approximation function

	0	1	2	3	4	
0	0.84	0.88	0.92	0.96	1.00	0
	0	1	2	3	4	

$$\hat{v}(s, \mathbf{w}) = w_0 + w_1 s$$

What are w_0, w_1 equal to?
Instead of the complete table, only 2 parameters to learn $\mathbf{w} = [w_0, w_1]^\top$

Notes

What are w_0, w_1 equal to?, we can start from left, target is the true $v(s = 0) = 0.84$, next target is $v(s = 1) = 0.88$, ...

Note about notation. Bold lower cases are used to denote vectors. Vectors are always considered oriented columnwise unless explicitly stated otherwise.

v(s) not as table but as an approximation function

	0	1	2	3	4	
0	0.84	0.88	0.92	0.96	1.00	0
	0	1	2	3	4	

$$\hat{v}(s, \mathbf{w}) = w_0 + w_1 s$$

What are w_0, w_1 equal to?

Instead of the complete table, only 2 parameters to learn $\mathbf{w} = [w_0, w_1]^\top$

Notes

What are w_0, w_1 equal to?, we can start from left, target is the true $v(s = 0) = 0.84$, next target is $v(s = 1) = 0.88$, ...

Note about notation. Bold lower cases are used to denote vectors. Vectors are always considered oriented columnwise unless explicitly stated otherwise.

v(s) not as table but as an approximation function

	0	1	2	3	4	
0	0.84	0.88	0.92	0.96	1.00	0
	0	1	2	3	4	

$$\hat{v}(s, \mathbf{w}) = w_0 + w_1 s$$

What are w_0, w_1 equal to?
Instead of the complete table, only 2 parameters to learn $\mathbf{w} = [w_0, w_1]^\top$

Notes

What are w_0, w_1 equal to?, we can start from left, target is the true $v(s = 0) = 0.84$, next target is $v(s = 1) = 0.88$, ...

Note about notation. Bold lower cases are used to denote vectors. Vectors are always considered oriented columnwise unless explicitly stated otherwise.

Linear value functions

7.00	8.00	9.00	10.00
6.00		8.00	-10.00
5.00	6.00	7.00	6.00

$$
\begin{aligned}
\hat{v}(s, \mathbf{w}) &= w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \cdots + w_n f_n(s) \\
\hat{q}(s, a, \mathbf{w}) &= w_1 f_1(s, a) + w_2 f_2(s, a) + w_3 f_3(s, a) + \cdots + w_n f_n(s, a)
\end{aligned}
$$

Notes

What could be the f functions for the grid world?

Obviously, when data are available, we can fit. How to do it on-line?

Learning $\mathbf{w}$ by Stochastic Gradient Descent (SGD)

- assume $\hat{v}(s, \mathbf{w})$ differentiable in all states
- we update $\mathbf{w}$ in discrete time steps t
- in each step S_t we observe a new example of (true) $v^\pi(S_t)$
- $\hat{v}(S_t, \mathbf{w})$ is an approximator $\rightarrow$ error $v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right]^\top$$

Notes

Gradient descent - all samples are known, Stochastic GD - update after each sample
$\hat{v}(s, \mathbf{w})$ could be quite complex, e.g. a Multi Layer Perceptron (MLP), Deep Network, and $\mathbf{w}$ represents the weights. See, e.g.

- https://skymind.ai/wiki/deep-reinforcement-learning
- Vision for robotics course you may take next term. https://cw.fel.cvut.cz/wiki/courses/b3b33vir/start

Learning **w** by Stochastic Gradient Descent (SGD)

- assume $\hat{v}(s, \mathbf{w})$ differentiable in all states
- we update $\mathbf{w}$ in discrete time steps t
- in each step S_t we observe a new example of (true) $v^\pi(S_t)$
- $\hat{v}(S_t, \mathbf{w})$ is an approximator $\to$ error $v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \Big]^2 \\ &= \mathbf{w}_t + \alpha \Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t) \end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right]^\top$$

Notes

Gradient descent - all samples are known, Stochastic GD - update after each sample
$\hat{v}(s, \mathbf{w})$ could be quite complex, e.g. a Multi Layer Perceptron (MLP), Deep Network, and **w** represents the weights. See, e.g.

- https://skymind.ai/wiki/deep-reinforcement-learning
- Vision for robotics course you may take next term. https://cw.fel.cvut.cz/wiki/courses/b3b33vir/start

Learning $\mathbf{w}$ by Stochastic Gradient Descent (SGD)

- assume $\hat{v}(s, \mathbf{w})$ differentiable in all states
- we update $\mathbf{w}$ in discrete time steps t
- in each step S_t we observe a new example of (true) $v^\pi(S_t)$
- $\hat{v}(S_t, \mathbf{w})$ is an approximator $\rightarrow$ error $v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right]^\top$$

Notes

Gradient descent - all samples are known, Stochastic GD - update after each sample
$\hat{v}(s, \mathbf{w})$ could be quite complex, e.g. a Multi Layer Perceptron (MLP), Deep Network, and $\mathbf{w}$ represents the weights. See, e.g.

- https://skymind.ai/wiki/deep-reinforcement-learning
- Vision for robotics course you may take next term. https://cw.fel.cvut.cz/wiki/courses/b3b33vir/start

Learning $\mathbf{w}$ by Stochastic Gradient Descent (SGD)

- assume $\hat{v}(s, \mathbf{w})$ differentiable in all states
- we update $\mathbf{w}$ in discrete time steps t
- in each step S_t we observe a new example of (true) $v^\pi(S_t)$
- $\hat{v}(S_t, \mathbf{w})$ is an approximator $\rightarrow$ error $v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]\nabla\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right]^\top$$

Notes

Gradient descent - all samples are known, Stochastic GD - update after each sample
$\hat{v}(s, \mathbf{w})$ could be quite complex, e.g. a Multi Layer Perceptron (MLP), Deep Network, and $\mathbf{w}$ represents the weights. See, e.g.

- https://skymind.ai/wiki/deep-reinforcement-learning
- Vision for robotics course you may take next term. https://cw.fel.cvut.cz/wiki/courses/b3b33vir/start

Learning $\mathbf{w}$ by Stochastic Gradient Descent (SGD)

- assume $\hat{v}(s, \mathbf{w})$ differentiable in all states
- we update $\mathbf{w}$ in discrete time steps t
- in each step S_t we observe a new example of (true) $v^\pi(S_t)$
- $\hat{v}(S_t, \mathbf{w})$ is an approximator $\rightarrow$ error $v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\Big]\nabla\hat{v}(S_t, \mathbf{w}_t) \end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right]^\top$$

Notes

Gradient descent - all samples are known, Stochastic GD - update after each sample
$\hat{v}(s, \mathbf{w})$ could be quite complex, e.g. a Multi Layer Perceptron (MLP), Deep Network, and $\mathbf{w}$ represents the weights. See, e.g.

- https://skymind.ai/wiki/deep-reinforcement-learning
- Vision for robotics course you may take next term. https://cw.fel.cvut.cz/wiki/courses/b3b33vir/start

Approximate Q-learning (of a linear combination)

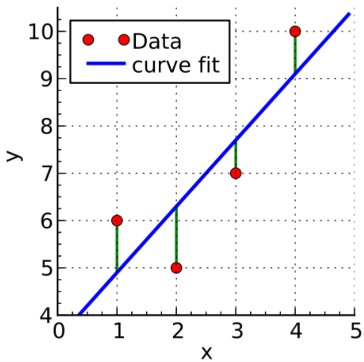
$$\hat{q}(s, a, \mathbf{w}) = w_1 f_1(s, a) + w_2 f_2(s, a) + w_3 f_3(s, a) + \cdots + w_n f_n(s, a)$$

- transition $= S_t, A_t, R_{t+1}, S_{t+1}$
- trial $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$
- diff $= \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)\right] - \hat{q}(S_t, A_t, \mathbf{w}_t)$
- Update: $\mathbf{w} = [w_1, w_2, \cdots, w_d]^\top$
  from previous slide we know that $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\left[v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right]\nabla \hat{v}(S_t, \mathbf{w}_t)$
  and $\hat{q}(s, a, \mathbf{w})$ is linear in $\mathbf{w}$
  $w_i \leftarrow w_i + \alpha\,[\text{diff}]\, f_i(S_t, A_t)$

Notes

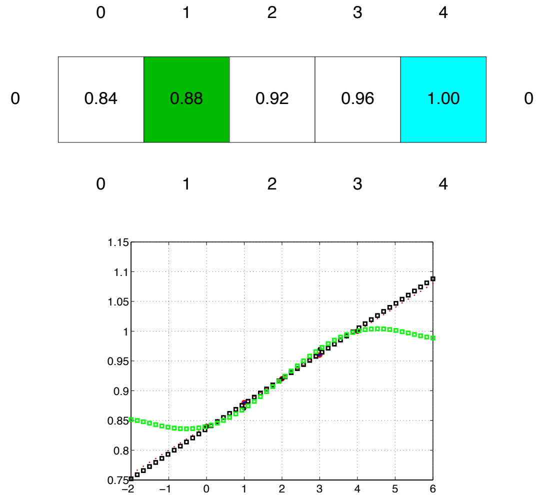
- We are minimizing error in point where we measure (sample).
- However, we know we only approximate.
- α is a kind of dumping factor, convergence of SGD requires that α decreases over time.

How is it possible at all? On-line least squares!



By Krishnavedala - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15462765

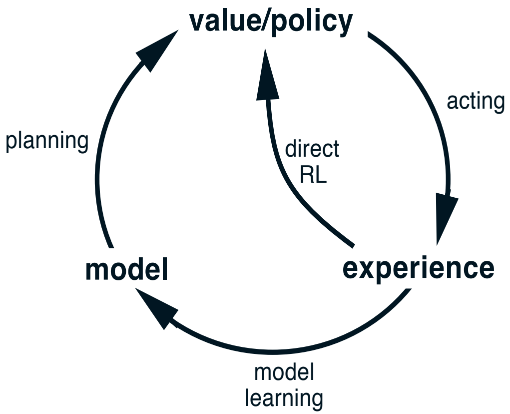
Overfitting

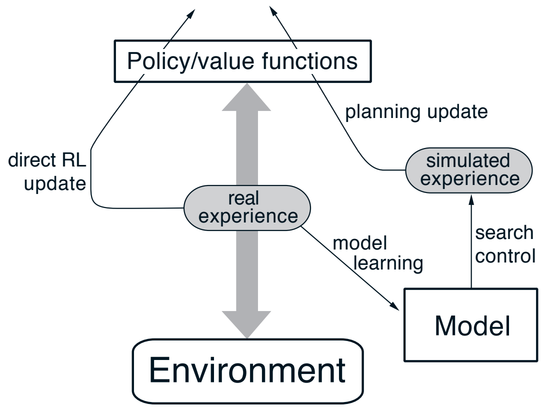


Notes

See the `fitdemo.m` run, higher degree polynomials perfectly fits, but poorly generalizes outside the range

Going beyond - Dyna-Q integration planning, acting, learning





[2]

[2] Schemes from [2]

Notes

References

Further reading: Chapter 21 of [1]. More detailed discussion in [2] Chapters 6 and 9. You can read about strategies for exploratory moves at various places, Tensor Flow related[3].

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[2] Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning; an Introduction.
MIT Press, 2nd edition, 2018.
http://www.incompleteideas.net/book/the-book-2nd.html.

[3] https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf

Notes