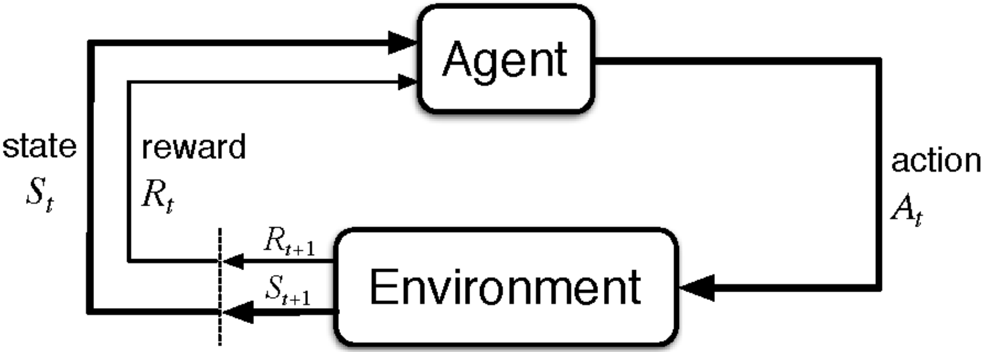
Reinforcement learning II
Active learning

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 6, 2022

Recap: Reinforcement Learning



[1]

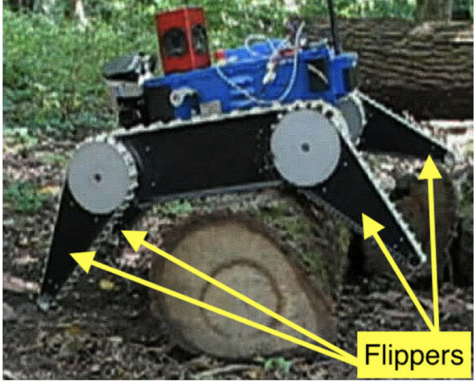
- Feedback in form of Rewards
- Learn to act so as to maximize sum of expected rewards.
- In `kuimaze` package, `env.step(action)` is the method.

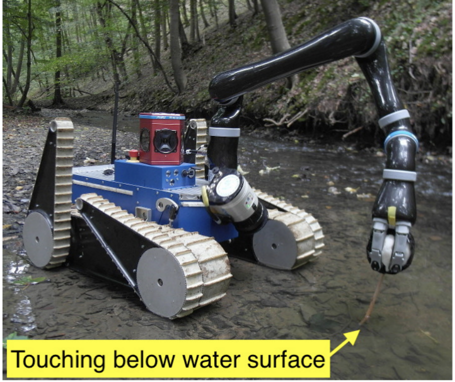
[1] Scheme from [2]

Learning to control flippers

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

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



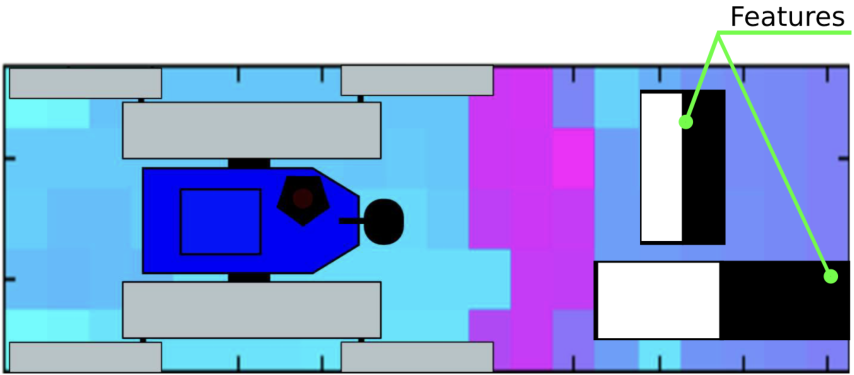


- Construction: $2\times$ main tracks, $4\times$ subtracks (flippers), differential break
  great stability and climbing capability
- Sensor suite: SICK LMS-151 range finder, Ladybug omnicam, Xsens MTi-G IMU
  3D sensing and localization
- Control inputs: Velocity vector, $4\times$**flipper angle**,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

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

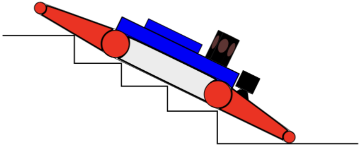
- Proprioceptive measurements: roll, pitch, torques, velocity, acceleration.

- Local exteroceptive measurements. features on digital elevation map with fixed size.

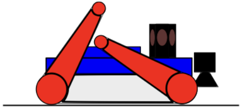


Instead of $\mathbf{a} \in \mathcal{A} \subset \mathbb{R}^8$ we consider only 5 configurations[2]:

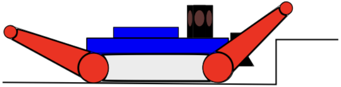
$$\mathcal{A} = \{\text{I-shape}\,,\, \text{V-shape}\,,\, \text{L-shape}\,,\, \text{U-shape soft}\,,\, \text{U-shape hard}\}$$



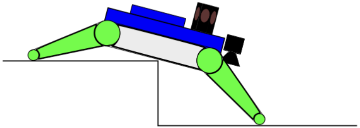
I-shape
(Maximizes traction)



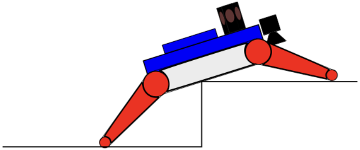
V-shape
(Provides observability)



L-shape
(Forward approach)



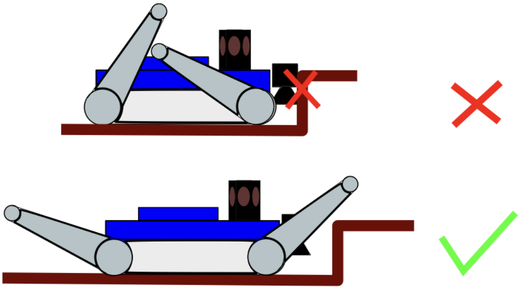
U-shape soft
(Smooth climbing down)



U-shape hard
(Lifts the body up)

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



r(V-shape , s) =-1

r(L-shape , s) = 1

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.

(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.

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.

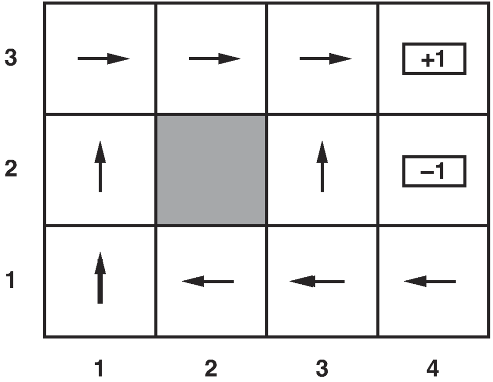
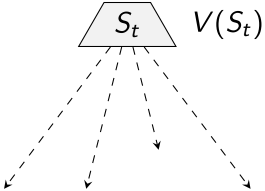


Image from [1]

Recap: $V-$ learning

Learn $V(s)$ values as the robot/agent goes (temporal difference).

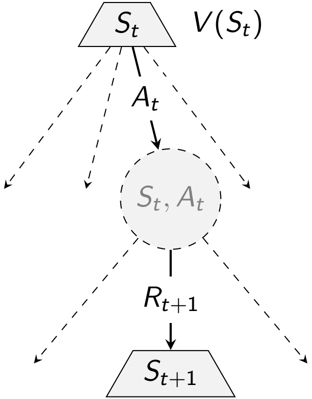


- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma V(S_{t+1})$
- α temporal difference update
  $V(S_t) \leftarrow V(S_t) + \alpha(\text{trial} - V(S_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless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

Recap: $V-$ learning

Learn $V(s)$ values as the robot/agent goes (temporal difference).

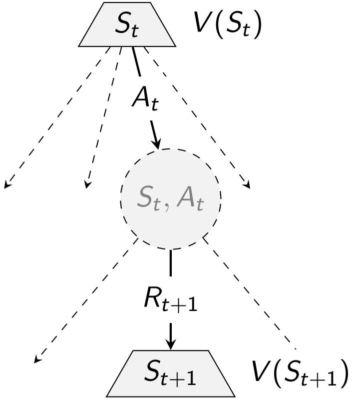
- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma V(S_{t+1})$
- α temporal difference update
  $V(S_t) \leftarrow V(S_t) + \alpha(\text{trial} - V(S_t))$
- $S_t \leftarrow S_{t+1}$ and repeat (unless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



Recap: $V-$ learning

Learn $V(s)$ values as the robot/agent goes (temporal difference).

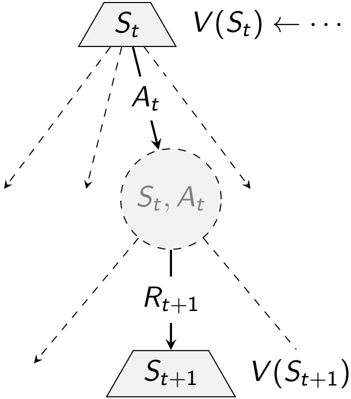
- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  $\text{trial} = R_{t+1} + \gamma V(S_{t+1})$
- α temporal difference update
  $V(S_t) \leftarrow V(S_t) + \alpha(\text{trial} - V(S_t))$
- $S_t \leftarrow S_{t+1}$ and repeat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



Recap: $V-$ learning

Learn $V(s)$ values as the robot/agent goes (temporal difference).

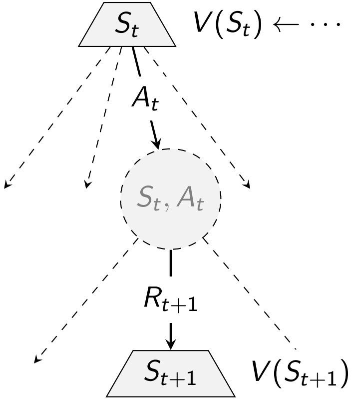
- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  $\text{trial} = R_{t+1} + \gamma V(S_{t+1})$
- α temporal difference update
  $V(S_t) \leftarrow V(S_t) + \alpha(\text{trial} - V(S_t))$
- $S_t \leftarrow S_{t+1}$ and repeat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



Recap: $V-$ learning

Learn $V(s)$ values as the robot/agent goes (temporal difference).

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



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

$\gamma = 1$, rewards $-1, +10, -10$, and deterministic robot

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

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

What is wrong with the temporal difference Value learning?

The Good: Model-free value learning by mimicking Bellman updates.

The Bad: How to turn values into a (new) policy?

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

What is wrong with the temporal difference Value learning?

The Good: Model-free value learning by mimicking Bellman updates.

The Bad: How to turn values into a (new) policy?

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

What is wrong with the temporal difference Value learning?

The Good: Model-free value learning by mimicking Bellman updates.

The Bad: How to turn values into a (new) policy?

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

Model-free TD learning, updating after each transition

- Observe, experience environment through learning episodes, collecting:

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

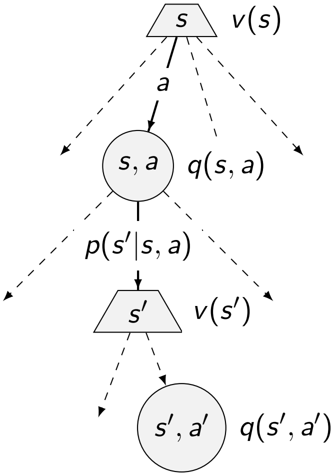
- Update by mimicking Bellman updates after each transition $(S_t, A_t, R_{t+1}, S_{t+1})$

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

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

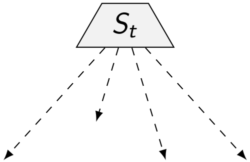
The value of a q-state (s, a):

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



Q-learning (off-policy TD control)

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



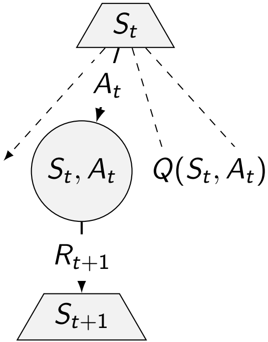
- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  trial $= R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t \leftarrow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

In each step Q approximates the optimal q^* function (learns optimal policy).

Q-learning (off-policy TD control)

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

- time t, at S_t
- select and 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 (learns optimal policy).

Q-learning (off-policy TD control)

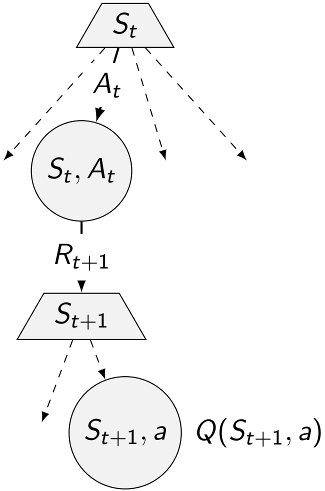
Learn policy (Q-values) as the robot/agent goes (temporal difference). If some Q quantity not known, initialize.

- time t, at S_t
- select and 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 (learns optimal policy).

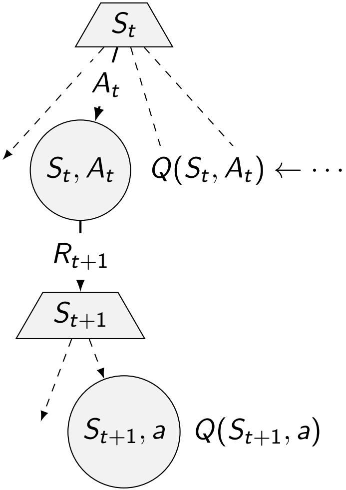


Q-learning (off-policy TD control)

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

- time t, at S_t
- select and 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)

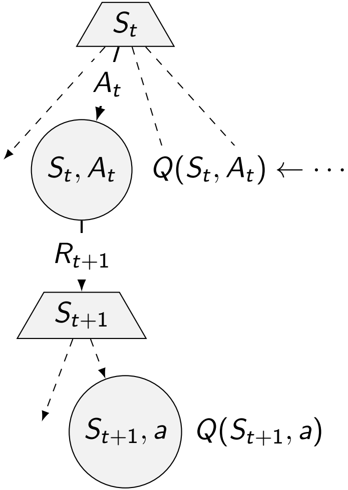
In each step Q approximates the optimal q^* function (learns optimal policy).



Q-learning (off-policy TD control)

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

- time t, at S_t
- select and 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)



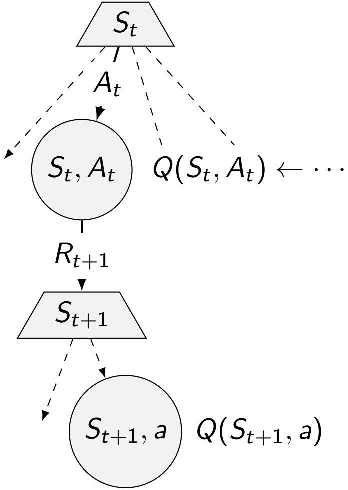
In each step Q approximates the optimal q^* function (learns optimal policy).

Q-learning (off-policy TD control)

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

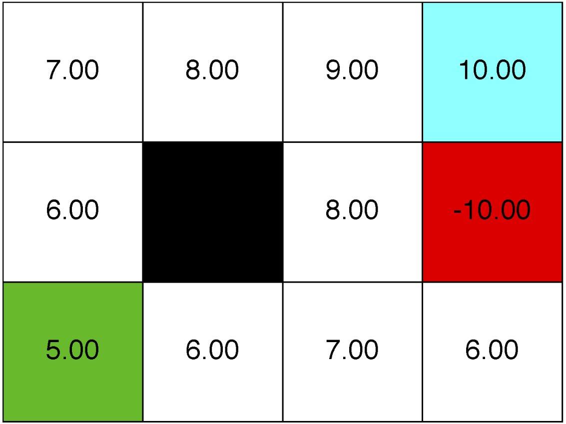
- time t, at S_t
- select and take $A_t \in \mathcal{A}(S_t)$, observe R_{t+1}, S_{t+1}
- compute trial/sample estimate at time t
  $\text{trial} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- α temporal difference update
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\text{trial} - Q(S_t, A_t))$
- $S_t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

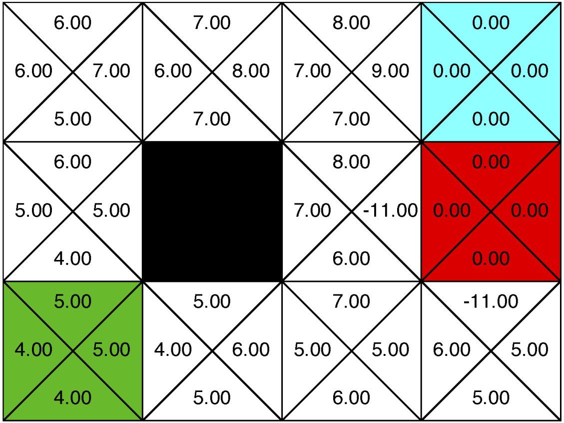
In each step Q approximates the optimal q^* function (learns optimal policy).



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

$\gamma = 1$, rewards $-1, +10, -10$, and deterministic robot





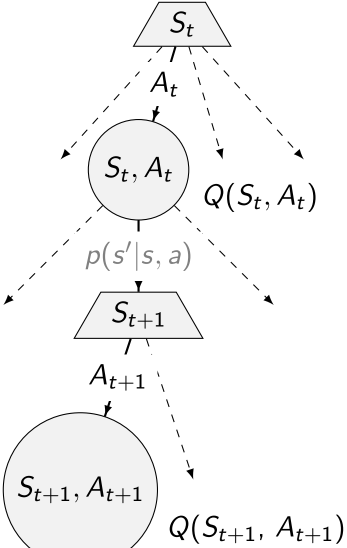
$$V(S_t) = R_{t+1} + V(S_{t+1})$$
$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a)$$

Sarsa (on-policy TD control)

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

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

In each step learns Q.



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'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
        $S \leftarrow S'$
    end for until S is terminal
until Time is up, ...

Sarsa: 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
    choose A from S
    for for each step of episode: **do**
        take action A, observe R, S'
        choose A' from S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \big[R + \gamma Q(S', A') - Q(S, A) \big]$
        $S \leftarrow S'$, $A \leftarrow A'$
    end for until S is terminal
until Time is up, ...

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)

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)

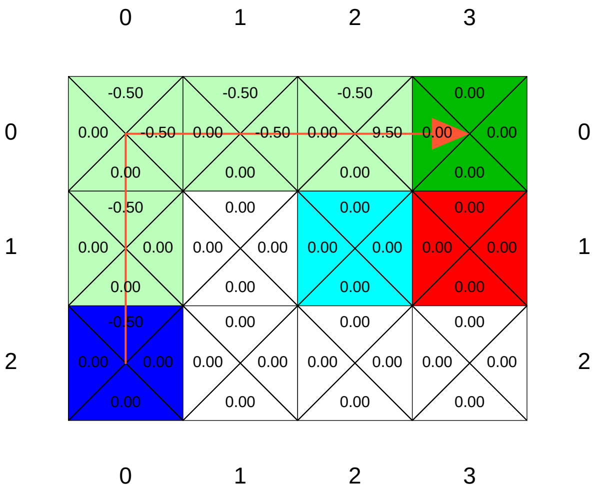
... A_t derived from Q

What about keeping optimality, taking max?

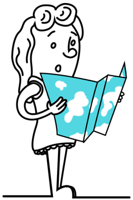
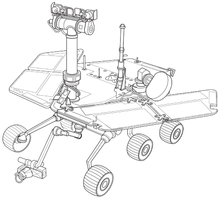
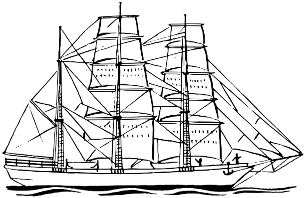
$$A_t = \arg\max_a Q(S_t, a)$$

see the demo run of `rl_agents.py`.

Two good goal states



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?
- ...

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?

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?

How to explore?

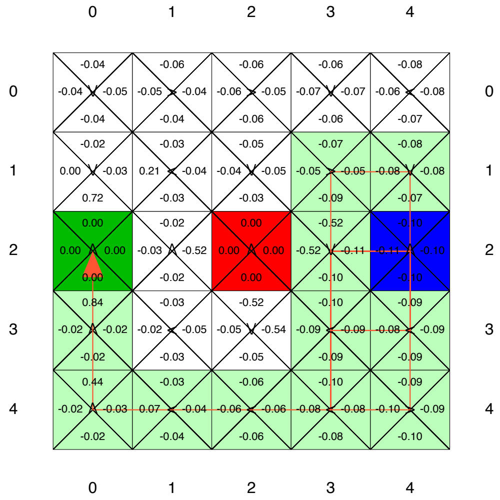
Random (ϵ-greedy):

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

Problems with randomness?

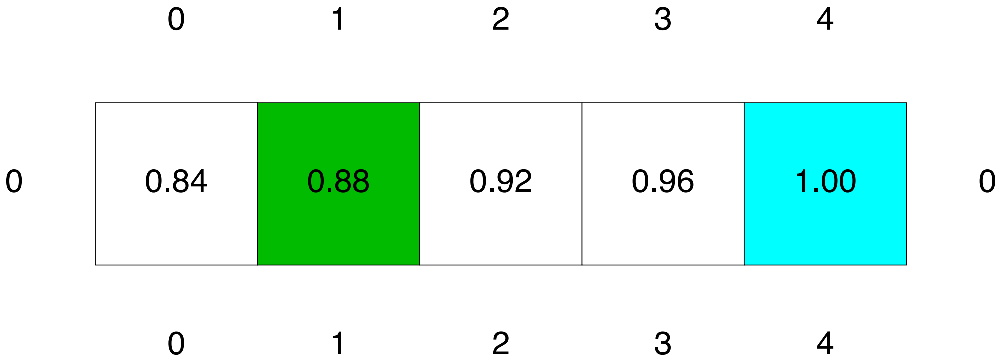
- Keeps exploring forever.
- Should we keep ϵ fixed (over learning)?
- ϵ same everywhere?

How to evaluate the result? When to stop learning?



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

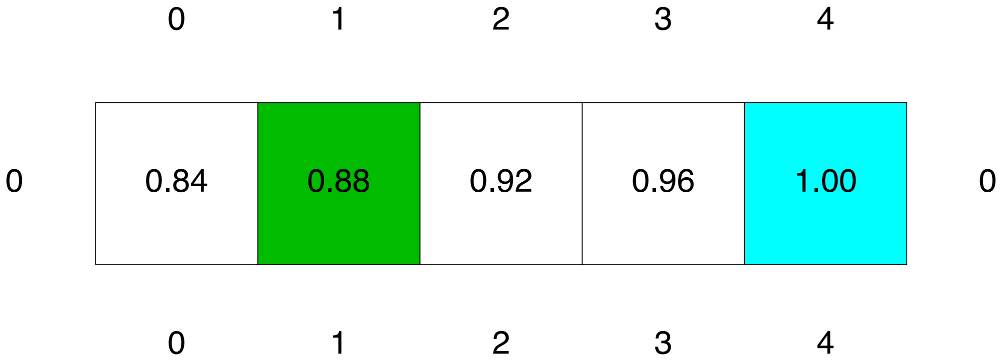
Going beyond tables – generalizing across states



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	

$v(s)$ not as a table but as an approximation function $\hat{v}(s, \mathbf{w})$



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

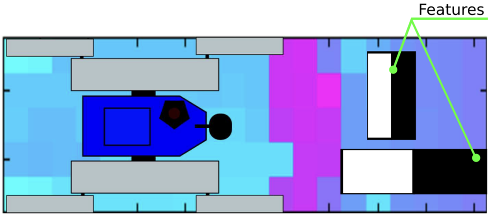
What are w_0, w_1 equal to?
Instead of the complete table, only 2 parameters to learn $\mathbf{w} = [w_0, w_1]^\top$

Linear value functions

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.

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



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

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 $= v^{\pi}(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

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

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

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 $= v^\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)$

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

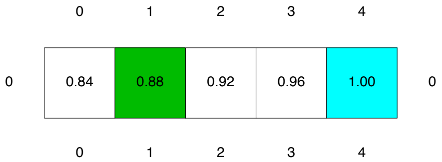
$$
\nabla f(\mathbf{w}) \doteq \left[\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right]^\top
$$

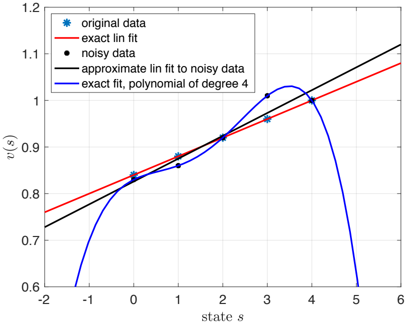
Approximate Q-learning (of a linear combination)

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

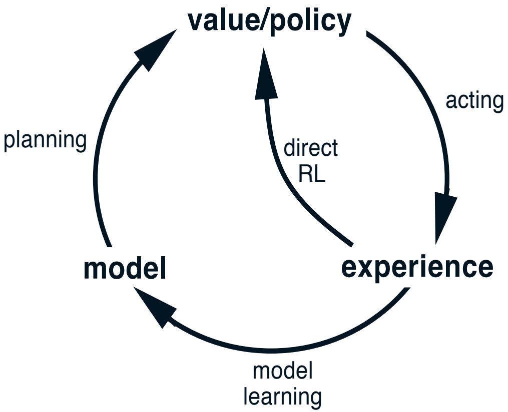
- transition $= S_t, A_t, R_{t+1}, S_{t+1}$
- trial $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$
- diff $= \Big[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)\Big] - \hat{q}(S_t, A_t, \mathbf{w}_t)$
- Update: $\mathbf{w} = [w_1, w_2, \cdots, w_d]^\top$
  from previous slide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

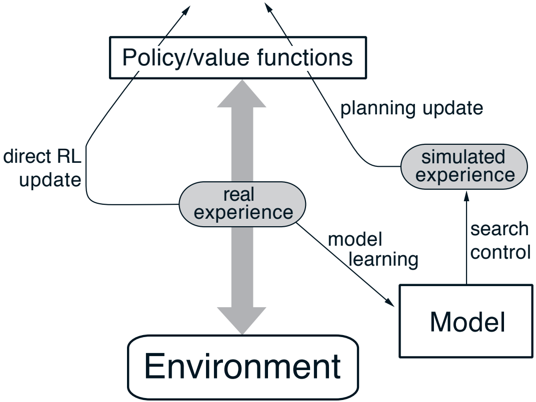
How to design the q-function? Overfitting ...





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





2

[2]Schemes from [2]

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]. More RL URLs at the course pages[4].

[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/bookdraft2018jan1.pdf.

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

[4] https://cw.fel.cvut.cz/wiki/courses/b3b33kui/cviceni/program_po_tydnech/tyden_09#reinforcement_learning_plus