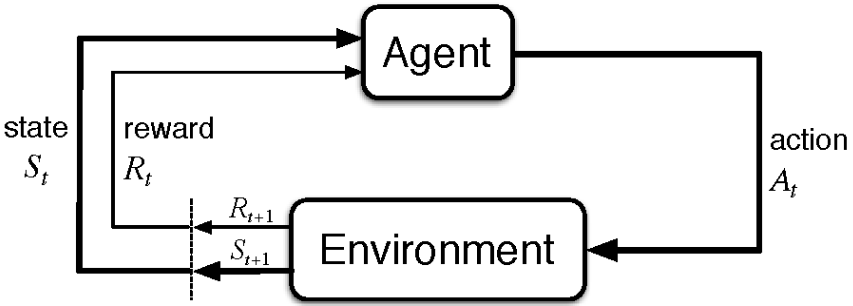
Reinforcement learning II

Tomáš Svoboda

Department of Cybernetics, Vision for Robotics and Autonomous Systems,
Center for Machine Perception (CMP)

May 28, 2018

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]

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

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

- A set of states $s \in S$ (map)
- A set of actions per state. $a \in A$
- A transition model $T(s, a, s')$ or $P(s'|s, a)$ (robot)
- A reward function $R(s)$ (map, robot)

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

On-line problem:

- T and R not known.
- Agent/robot must **act** and **learn from experience**.

For MDPs, we know T, 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 $P(s'|s, a)$[2]
- Discover each $R(s, a, s')$ when experience.

Solve the learned MDP.

[2] The same as $T(s, a, s')$. Probability gives perhaps a better insight how to normalize.

Model-free learning

Model-free learning

- R, T not known.
- Move around, observe
- And learn on the way.
- **Goal:** learn the state values $V(s)$ or (better) $Q(s, a)$.

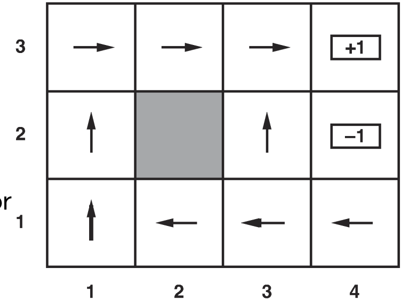
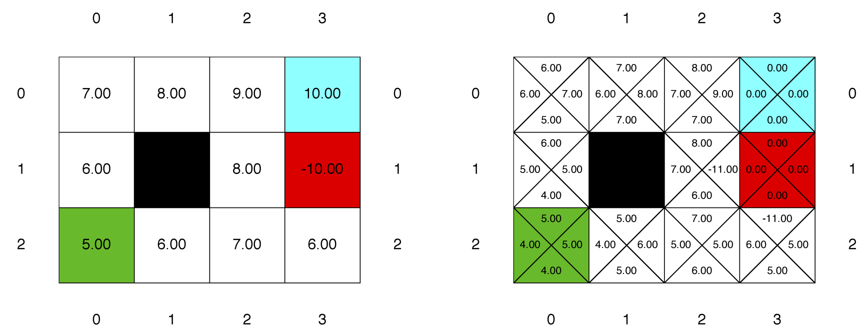


Image from [1]

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



$$V^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^{t}R(S_t)\right]$$

$$Q(s,a) = \mathrm{E}\left[R(s,a,s') + \sum_{t=1}^{\infty}\gamma^{t}R(S_t)\right]$$

$\gamma = 1$, 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.

Model-free TD learning, updating after each transition

- Observe, experience environment through learning episodes, collecting:

$$(s, a, r, s', a', r', s'', a'', r'', \dots)$$

- using t for trial/iteration:

$$(s_1, a_1, r_1, s_2, a_2, r_2, s_3 \dots) = s_t, a_t, r_t, s_{t+1}, \dots$$

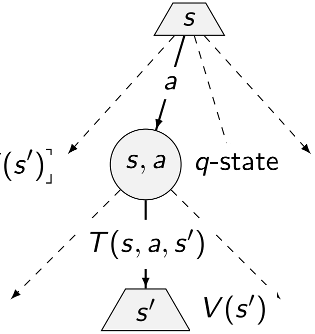
- Update by mimicking Bellman updates after each transition (s, a, r, s')

Think about s-a-s'-a'-s" tree with associated rewards. Episode starts in a start state and ends in a terminal state.

Recap: Bellman equations for $V(s)$ and $Q(s, a)$

The value of a state s:

$$\begin{aligned} V(s) &= R(s) + \gamma \max_a \sum_{s'} T(s, a, s') V(s') \\ &= \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s') \right] \\ &= \max_a Q(s, a) \end{aligned}$$



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

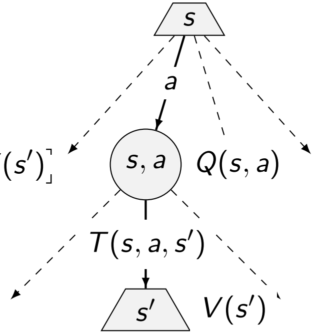
$$\begin{aligned} Q(s, a) &= \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V(s') \right] \\ &= \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') \right] \end{aligned}$$

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

Recap: Bellman equations for $V(s)$ and $Q(s,a)$

The value of a state s:

$$
\begin{aligned}
V(s) &= R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V(s') \\
&= \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right] \\
&= \max_a Q(s,a)
\end{aligned}
$$



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

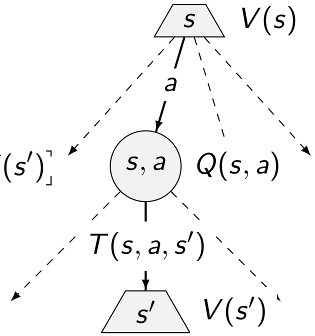
$$
\begin{aligned}
Q(s,a) &= \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right] \\
&= \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') \right]
\end{aligned}
$$

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

Recap: Bellman equations for $V(s)$ and $Q(s, a)$

The value of a state s:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')V(s')$$
$$= \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$$
$$= \max_a Q(s,a)$$



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

$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$$
$$= \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$

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

Recap: V, Q-value iteration for MDPs

Value/Utility iteration (depth limited evaluation):

- Start: $V_0(s) = 0$
- In each step update V by looking one step ahead:
  $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\,[R(s,a,s') + \gamma V_k(s')]$

Q values more useful (think about updating π)

- Start: $Q_0(s,a) = 0$
- In each step update Q by looking one step ahead:
  $Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$

Draw the (s)-(s,a)-(s')-(s',a') tree. It will be also handy when discussing exploration vs. exploitation - where to drive next.

Q-learning

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Learn Q values as the robot/agent goes (temporal difference)

- Drive the robot and fetch: $s, a, s', R(s,a,s')$
- We know old estimates $Q(s,a)$ (and $Q(s',a')$), if not, initialize.
- A new trial/sample estimate
  trial $= R(s,a,s') + \gamma \max_{a'} Q(s',a')$
- α update
  $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{trial} - Q(s,a))$
  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,\text{trial}$

A step-by-step hand-computed example on a blackboard. There will be also a related quizz during the labs.

There alternatives how to compute the trial. SARSA method takes $Q(s',a')$ directly, not the max. Hence we need 5-tuples s, a, r, s', a'

Q-learning

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Learn Q values as the robot/agent goes (temporal difference)

- Drive the robot and fetch: $s, a, s', R(s,a,s')$
- We know old estimates $Q(s,a)$ (and $Q(s',a')$), if not, initialize.
- A new trial/sample estimate
  trial $= R(s,a,s') + \gamma \max_{a'} Q(s',a')$
- α update
  $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{trial} - Q(s,a))$
  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,\text{trial}$

A step-by-step hand-computed example on a blackboard. There will be also a related quizz during the labs.

There alternatives how to compute the trial. SARSA method takes $Q(s',a')$ directly, not the max. Hence we need 5-tuples s, a, r, s', a'

Q-learning

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Learn Q values as the robot/agent goes (temporal difference)

- Drive the robot and fetch: $s, a, s', R(s,a,s')$
- We know old estimates $Q(s,a)$ (and $Q(s',a')$), if not, initialize.
- A new trial/sample estimate
  trial $= R(s,a,s') + \gamma \max_{a'} Q(s',a')$
- α update
  $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{trial} - Q(s,a))$
  $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\,\text{trial}$

A step-by-step hand-computed example on a blackboard. There will be also a related quizz during the labs.

There alternatives how to compute the trial. SARSA method takes $Q(s',a')$ directly, not the max. Hence we need 5-tuples s, a, r, s', a'

Q-learning

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Learn Q values as the robot/agent goes (temporal difference)

- Drive the robot and fetch: $s, a, s', R(s,a,s')$
- We know old estimates $Q(s,a)$ (and $Q(s',a')$), if not, initialize.
- A new trial/sample estimate
  $\text{trial} = R(s,a,s') + \gamma \max_{a'} Q(s',a')$
- α update
  $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{trial} - Q(s,a))$
  $Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha\,\text{trial}$

A step-by-step hand-computed example on a blackboard. There will be also a related quizz during the labs.

There alternatives how to compute the trial. SARSA method takes $Q(s',a')$ directly, not the max. Hence we need 5-tuples s, a, r, s', a'

From Q-learning to Q-learning agent

- Drive the robot and fetch: $s, a, s', R(s, a, s')$
- We know old estimates $Q(s, a)$ (and $Q(s', a')$), if not, initialize.
- A new trial/sample estimate: $\text{trial} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- α update: $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{trial} - Q(s, a))$

Technicalities for the Q-learning agent

- How to represent Q-function?
- What is the value for terminal? $Q(s, \text{Exit})$ or $Q(s, \text{None})$
- How to drive? Where to drive next? Does it change over the course?

Q-function for a discrete, finite problem? But what about continous space ot discrete but a very large one?

Use the (s)-(s,a)-(s')-(s',a') tree to discuss the next-action selection.

From Q-learning to Q-learning agent

- Drive the robot and fetch: $s, a, s', R(s, a, s')$
- We know old estimates $Q(s, a)$ (and $Q(s', a')$), if not, initialize.
- A new trial/sample estimate: $\text{trial} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- α update: $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{trial} - Q(s, a))$

Technicalities for the Q-learning agent

- How to represent Q-function?
- What is the value for terminal? $Q(s, \text{Exit})$ or $Q(s, \text{None})$
- How to drive? Where to drive next? Does it change over the course?

Q-function for a discrete, finite problem? But what about continous space ot discrete but a very large one?

Use the (s)-(s,a)-(s')-(s',a') tree to discuss the next-action selection.

From Q-learning to Q-learning agent

- Drive the robot and fetch: $s, a, s', R(s, a, s')$
- We know old estimates $Q(s, a)$ (and $Q(s', a')$), if not, initialize.
- A new trial/sample estimate: $\text{trial} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- α update: $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{trial} - Q(s, a))$

Technicalities for the Q-learning agent

- How to represent Q-function?
- What is the value for terminal? $Q(s, \text{Exit})$ or $Q(s, \text{None})$
- How to drive? Where to drive next? Does it change over the course?

Q-function for a discrete, finite problem? But what about continous space ot discrete but a very large one?

Use the (s)-(s,a)-(s')-(s',a') tree to discuss the next-action selection.

From Q-learning to Q-learning agent

- Drive the robot and fetch: $s, a, s', R(s, a, s')$
- We know old estimates $Q(s, a)$ (and $Q(s', a')$), if not, initialize.
- A new trial/sample estimate: $\text{trial} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- α update: $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{trial} - Q(s, a))$

Technicalities for the Q-learning agent

- How to represent Q-function?
- What is the value for terminal? $Q(s, \text{Exit})$ or $Q(s, \text{None})$
- How to drive? Where to drive next? Does it change over the course?

Q-function for a discrete, finite problem? But what about continous space ot discrete but a very large one?
Use the (s)-(s,a)-(s')-(s',a') tree to discuss the next-action selection.

From Q-learning to Q-learning agent

- Drive the robot and fetch: $s, a, s', R(s, a, s')$
- We know old estimates $Q(s, a)$ (and $Q(s', a')$), if not, initialize.
- A new trial/sample estimate: $\text{trial} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- α update: $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{trial} - Q(s, a))$

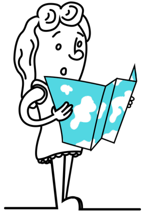
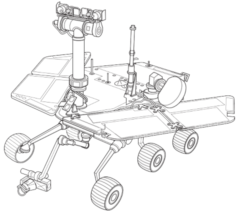
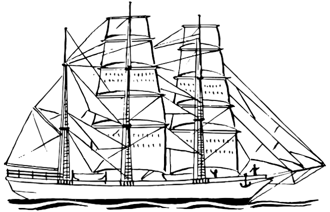
Technicalities for the Q-learning agent

- How to represent Q-function?
- What is the value for terminal? $Q(s, \text{Exit})$ or $Q(s, \text{None})$
- How to drive? Where to drive next? Does it change over the course?

Q-function for a discrete, finite problem? But what about continous space ot discrete but a very large one?

Use the (s)-(s,a)-(s')-(s',a') tree to discuss the next-action selection.

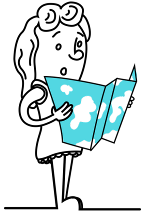
Exploration vs Exploitation

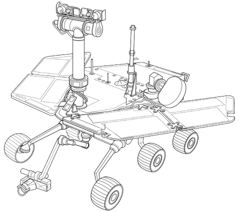




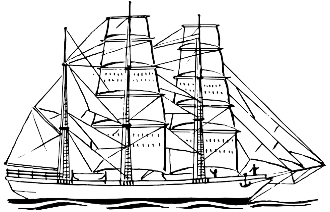
- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

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

Exploration vs Exploitation



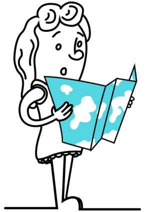


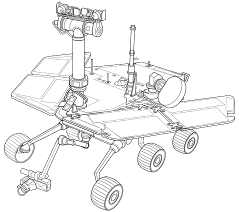


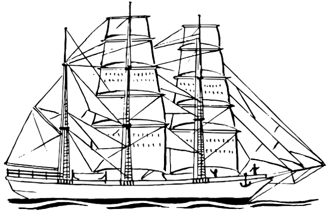
- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

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

Exploration vs Exploitation



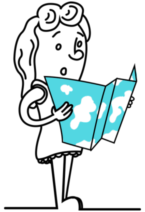
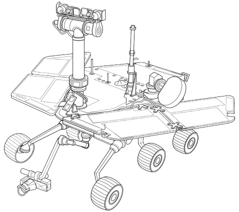
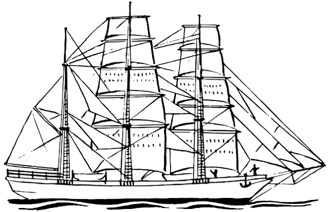




- Drive the known road or try a new one?
- Go to the university menza or try a nearby restaurant?
- Use the SW (operating system) I know or try new one?
- Go to bussiness or study a demanding program?
- ...

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

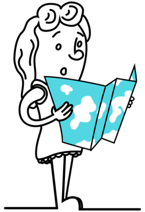
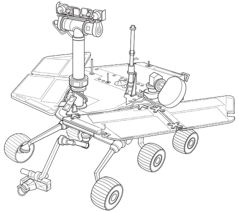
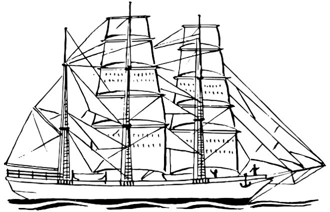
Exploration vs Exploitation

- 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?
- ...

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

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?
- ...

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

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?

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?

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?

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?

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?

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?

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?

We can think about lowering ϵ as the learning progresses.

How to explore?

Random (ϵ-greedy):

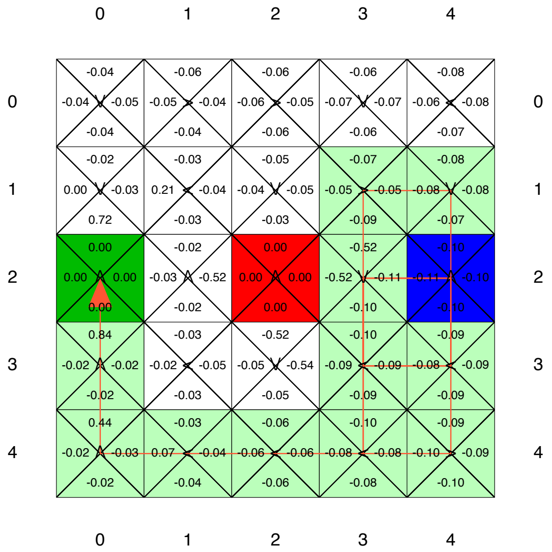
- Flip a coin every step.
- With probability ϵ, act randomly.
- With probability $1 - \epsilon$, use the policy.

Problems with randomness?

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

We can think about lowering ϵ as the learning progresses.

How to evaluate result, when to stop learning?



Run the found policy, discuss some traps, . . .

Exploration function $f(u, n)$

- Regular trial/sample estimate: trial $= R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- If (s', a') not yet tried, than perhaps too pesimistic.
- trial $= R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

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. N_e fixed.

Exploration function $f(u, n)$

- Regular trial/sample estimate: trial $= R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- If (s', a') not yet tried, than perhaps too pesimistic.
- trial $= R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

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. N_e fixed.

Exploration function $f(u, n)$

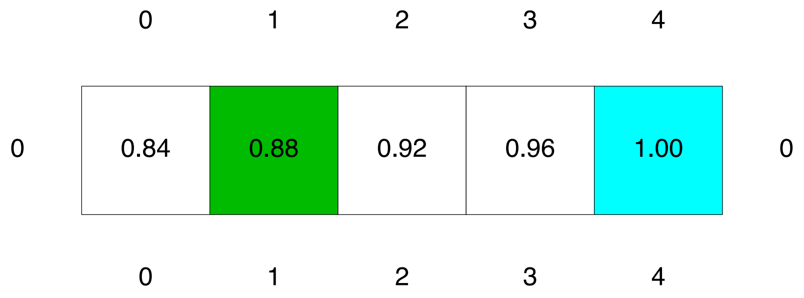
- Regular trial/sample estimate: trial $= R(s, a, s') + \gamma \max_{a'} Q(s', a')$
- If (s', a') not yet tried, than perhaps too pesimistic.
- trial $= R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

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. N_e fixed.

Going beyond tables - generalizing across states



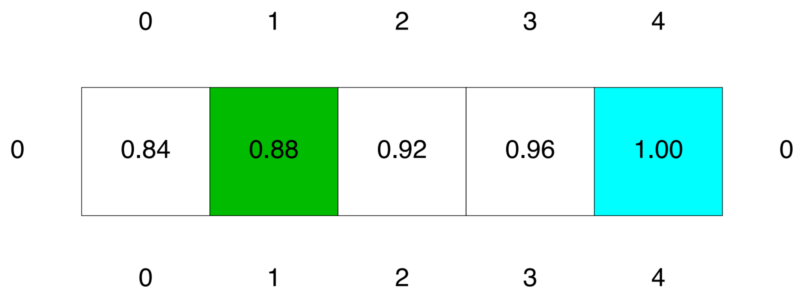
Looking a $V(s)$, we need a table for each of the state! This guy is small, but think bigger!

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	

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

V(s) not as table but as a function



$$V(s) = w_0 + w_1 s$$

Instead of the complete table, only 2 parameters to learn w_0, w_1

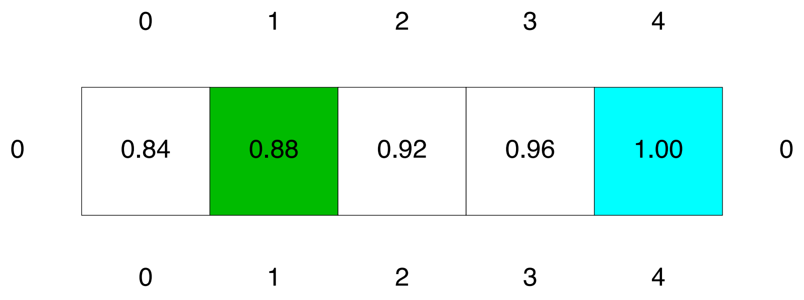
V(s) not as table but as a function

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

$$V(s) = w_0 + w_1 s$$

Instead of the complete table, only 2 parameters to learn w_0, w_1

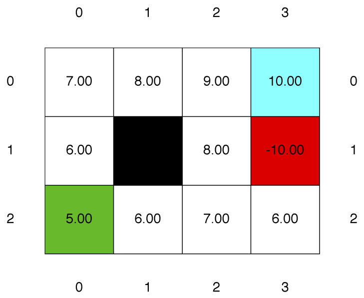
V(s) not as table but as a function



$$V(s) = w_0 + w_1 s$$

Instead of the complete table, only 2 parameters to learn w_0, w_1

Linear value functions



$$V(s) = w_1 f_1(s) + w_2 f_2(s) + w_3 f_3(s) + \cdots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a) + \cdots + w_n f_n(s,a)$$

What could be the f functions for the grid world?

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

Approximate Q-learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a) + \cdots + w_n f_n(s,a)$$

- transition $= s, a, r, s'$
- diff $= \left[r + \gamma \max_{a'} Q(s', a') \right] - Q(s,a)$
- Update:
  $Q(s,a) \leftarrow Q(s,a) + \alpha\,\text{diff}$
  $w_i \leftarrow w_i + \alpha\,[\text{diff}]\, f_i(s,a)$

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

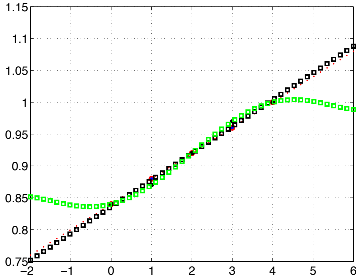
Optimization: Least Squares

$Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + w_3 f_3(s, a) + \cdots + w_n f_n(s, a)$

- Prediction: $Q(s, a)$
- Observation: $r + \gamma \max_{a'} Q(s', a')$

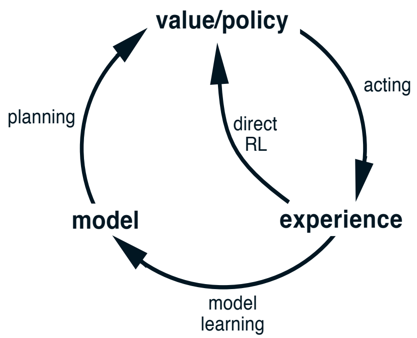
Overfitting

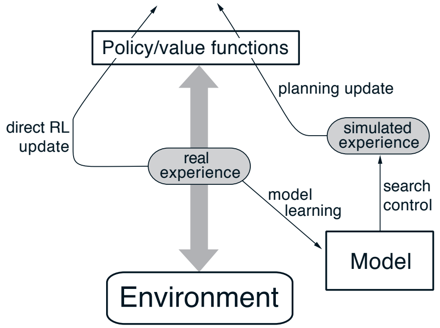
	0	1	2	3	4	
0	0.84	0.88	0.92	0.96	1.00	0
	0	1	2	3	4	



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

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





[3]

[3]Schemes from [2]

We are done with Search and Planning

- Search
- Games
- Markov Decision Problems
- Reinforecement Learning

Next: Uncertainty, Learning, (Conditional) Probabilities, Bayesian Decisions, Matlab, ...

We are done with Search and Planning

- Search
- Games
- Markov Decision Problems
- Reinforecement Learning

Next: Uncertainty, Learning, (Conditional) Probabilities, Bayesian Decisions, Matlab, ...

References

Further reading: Chapter 21 of [1]. More detailed discussion in [2] with slightly different notation, though. You can read about strategies for exploratory moves at various places, Tensor Flow related[4]. More RL URLs at the course pages[5].

[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.

[4] https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-stra

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