Unsupervised and Reinforcement Learning
In Neural Networks
Fall 2012

Reinforcement learning

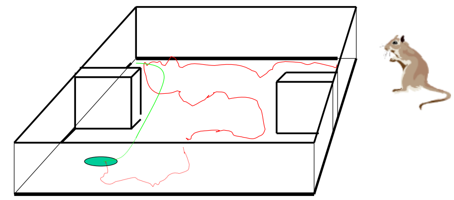
second lecture on RL:
- review from previous week
- on-policy (SARSA) vs off-policy (Q-learning
- eligibility traces

http://moodle.epfl.ch/

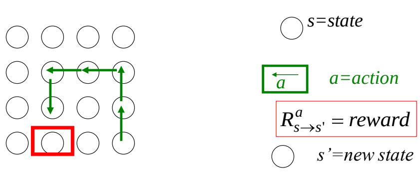
Wulfram Gerstner

http://lcn1.epfl.ch/

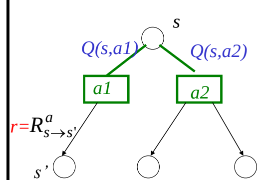
Introduction to reinforcement learning



Theory of Reinforcement learning



Theory of reinforcement learning – optimal strategy



$Q(s,a)=\sum_{s'} P^a_{s\to s'} R^a_{s\to s'}$

Optimal strategy:
- take action a with*
Q(s,a)>Q(s,aj)*
other actions

But: Q values not known
(probabilities not known)
→ *Estimate Q values*

Iterative Update
$\Delta Q(s,a)=\eta\ [r-Q(s,a)]$

Reward-based Action Learning

Exploration versus exploitation

Problem: correct Q values not known
(since reward probabilities are not known)

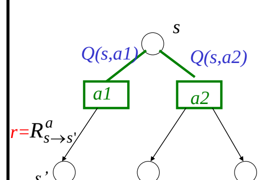
Exploration versus exploitation

Explore so as to estimate reward probababities

Take action which looks optimal, so as to maximize reward

Reward-based Action Learning

Exploration versus exploitation



$Q(s,a)=\sum_{s'} P^a_{s\to s'} R^a_{s\to s'}$

Update
$\Delta Q(s,a)=\eta\ [r-Q(s,a)]$

Problem: correct Q values not known
Greedy strategy:
- take action a which looks best*
Q(s,a)>Q(s,aj)]*

ε-greedy strategy:
- take action a which looks best*
with prob $P=1-\varepsilon$

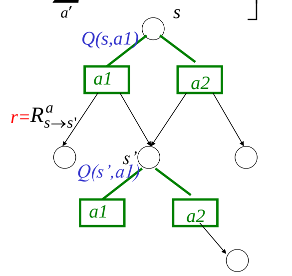
Softmax strategy:

$$P(action\ a\,|\,s)=\frac{\exp[Q(s,a)]}{\sum_{a'}\exp[Q(s,a')]}$$

Optimistic greedy:
start with Q values that are too big

$$Q(s,a) = \sum_{s'} P^a_{s \to s'} \left[R^a_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

Bellman equation



$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

SARSA algo

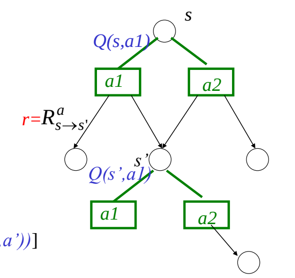
SARSA algo

Initialise Q values
Start from initial state ***s***

1) Being in state ***s*** choose action ***a*** according to policy
2) Observe reward ***r*** and next state ***s'***
3) Choose action **a'** in state s' according to policy
4) Update

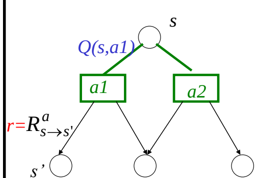
$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

5) s' → s; a' → a
6) Goto 1)



$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

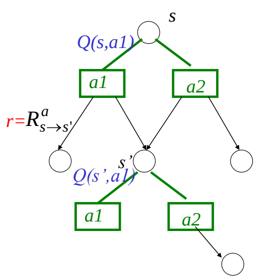
Reward-based Action Learning



$Q(s,a) = \sum_{s'} P^a_{s \to s'} R^a_{s \to s'}$

Update

$\Delta Q(s,a) = \eta\ [r - Q(s,a)]$



$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

Small learning rate, algos converge in expectation to the correct solution

Reinforcement learning

second lecture on RL:

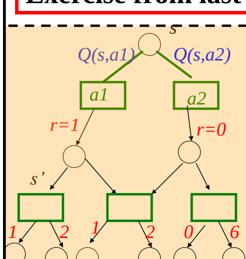
- review from previous week
- on-policy (SARSA) vs off-policy (Q-learning)
- eligibility traces

http://moodle.epfl.ch/

Wulfram Gerstner

http://lcn1.epfl.ch/

Exercise from last week: Bellman equation



a2 *a=action*

$R^a_{s \to s'}$ *= reward*

s'=new state

Calculated Q(s,a1) and Q(s,a2)

a) All actions equal probability
b) Optimal action

$$Q(s,a) = \sum_{s'} P^a_{s \to s'} \left[R^a_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

$$Q(s,a) = \sum_{s'} P^a_{s \to s'} \left[R^a_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

Bellman equations

Resulting Q values are different!

$$Q(s,a) = \sum_{s'} P^a_{s \to s'} \left[R^a_{s \to s'} + \gamma \max_{a'} Q(s',a') \right]$$

$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma Q(s',a'))]$

SARSA algo

$\Delta Q(s,a) = \eta\ [r - (Q(s,a) - \gamma \max_{a'} Q(s',a'))]$

Q-learning algo

Temporal-Difference Learning

Sarsa – on policy temporal difference algo

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Action for update as used for trajectory

Figure 6.9 Sarsa: An on-policy TD control algorithm

$\Delta Q(s,a) = \eta\,[r - (Q(s,a) - \gamma\, Q(s',a'))]$

From: Reinforcement Learning, Sutton and Barto 1998

Q-learning – off-policy temporal difference algo

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Policy used for Action selection

Greedy used for update

Figure 6.12 Q-learning: An off-policy TD control algorithm.

$\Delta Q(s,a) = \alpha \cdot \left[r(a) + \gamma \cdot \max_{s'} Q(s',a) - Q(s,a) \right]$

From: Reinforcement Learning, Sutton and Barto 1998

Reinforcement learning

second lecture on RL:
- review from previous week
- on-policy (SARSA) vs off-policy (Q-learning)
- eligibility traces

http://moodle.epfl.ch/

Wulfram Gerstner

http://lcn1.epfl.ch/

Exercise from last week

- ***Update of Q values in SARSA***

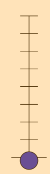
  $\Delta Q(s,a) = \eta\,[r - (Q(s,a) - Q(s',a'))]$

- ***policy for action choice:***

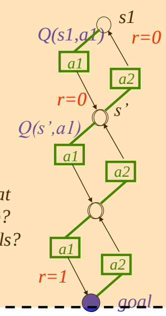
  Pick most often action

  $a_t^* = \arg\max_a Q_a(s,a)$

  Consider a linear sequence of states. Reward only at goal. Actions are up or down.
  a) Initialise Q values at 0. Start at top. How do Q values develop?
  b) Q values after 3 complete trials?



goal

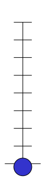


Problem: learning is slow

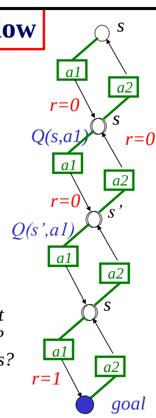
- Slow diffusion of information across several states

$a_t^* = \arg\max_a Q_a(s,a)$

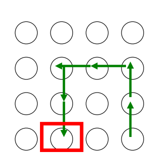
Consider a linear sequence of states. Reward only at goal. Actions are up or down.
a) Initialise Q values at 0. Start at top. How do Q values develop?
b) Q values after 3 complete trials?



goal



Eligibitiy trace



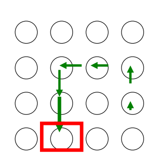
s=state

a=action

$R^a_{s \to s'}$ = *reward*

s'=new state

Sarsa $\Delta Q(s,a) = \eta\,[r - (Q(s,a) - \gamma\, Q(s',a'))]$



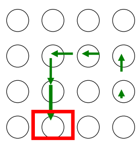
Idea: at the moment of reward update also previous action values along trajectories

Eligibitiy trace - algo

Sarsa(0) $\Delta Q(s,a) = \eta \; [r - (Q(s,a) - \gamma Q(s',a'))]$

$$\boxed{\delta_t = r_{t+1} - Q_a(s,a) - \gamma \cdot Q_a(s',a')}$$

Sarsa(λ) $\boxed{\Delta Q_{aj} = \eta \; \delta_t \; e_{aj}}$



$if \; a = action \;\; taken \, in \;\; state \; j$

$e_{aj}(t) = \gamma\lambda e_{aj}(t-\Delta t) + 1$

$else$

$e_{aj}(t) = \gamma\lambda e_{aj}(t-\Delta t)$

Elibitility trace – Sarsa()

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s, a *and e(s,a)=0*
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until s is terminal

Figure 7.11 Tabular Sarsa(λ).

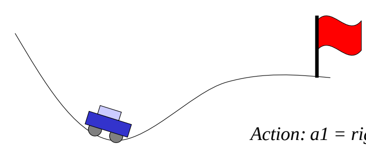
From: Reinforcement Learnin
Sutton and Barto 1998

Reinforcement learning

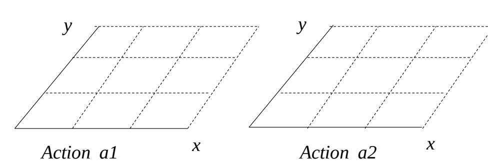
second lecture on RL:
- review from previous week
- on-policy (SARSA) vs off-policy (Q-learning)
- eligibility traces
- continuous state space

http://moodle.epfl.ch/
Wulfram Gerstner
http://lcn1.epfl.ch/

Continuous states – example

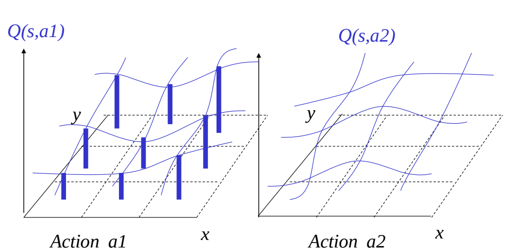


Action: a1 = right
a2 = left



Continuous states – example

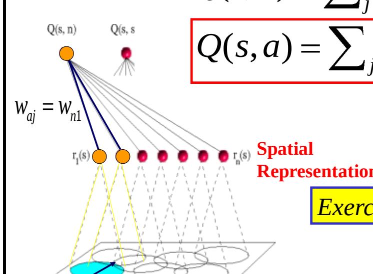
Blackboard:



Continuous states

$$Q(s,a) = \sum_j \; w_{aj} r_j(s)$$

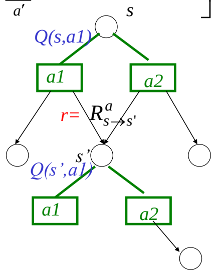
$$\boxed{Q(s,a) = \sum_j \; w_{aj} \Phi(s - s_j)}$$



Exercise: $\dfrac{dQ(s',a')}{dw_{aj}}$

$$Q(s,a) = \sum_{s'} P^a_{s\to s'} \left[R^a_{s\to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a') \right]$$

Bellman equation



Blackboard: idea

$Q(s,a) = r + \gamma Q(s',a')$

Eligibility and continous: TD( )

- ***TD error in SARSA***

$$\delta_t = R_{t+1} - Q_a(s,a) - \gamma \cdot Q_{a'}(s',a')$$

- ***policy for action choice:***
  Pick most often action

$$a^*_t = \arg\max_a Q_a(s,a)$$

- ***Eligibility trace***

$$e_{aj}(t) = \gamma\lambda\, e_{aj}(t-\Delta t) + \begin{cases} r_j & \text{if } a = \text{action taken} \\ 0 & \end{cases}$$

memory

- **Function approximation**

$$Q_w(s,a) = \sum_{i=1}^{n} w_i^a \cdot r_i(s)$$

- **Weight update**

$$\Delta w_{aj} = \eta\, \delta_t\, e_{aj}$$

Eligibility and continous: TD( )

- ***TD error in SARSA***

$$\delta_t = R_{t+1} - Q_a(s,a) - \gamma \cdot Q_{a'}(s',a')$$

- **Function approximation**

$$Q_w(s,a) = \sum_{i=1}^{n} w_i^a \cdot r_i(s)$$

- **Synaptic update**

$$\Delta w_{aj} = \eta\, \delta_t\, e_{aj}$$

- ***Eligibility trace (memory at synapse):***

$$e_{aj}(t) = \gamma\lambda\, e_{aj}(t-\Delta t) + \begin{cases} r_j \bullet 1 & \text{if } a = a' \ (= \text{action taken}) \\ r_j \bullet 0 & \text{else} \end{cases}$$

$$e_{aj}(t) = \gamma\lambda\, e_{aj}(t-\Delta t) + r_j\, \delta_{a,a'}$$

Biological interpretation: next week

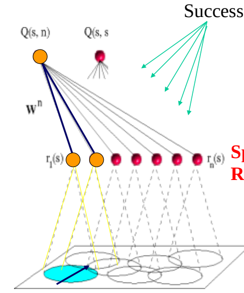
Reward-based Action Learning

Connection reinforced if
action a at state s successful

Success signal

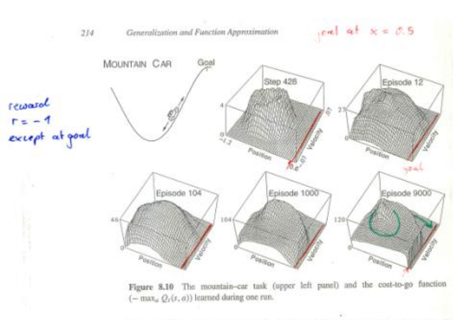
Learning Rule

$$\Delta w_{aj} = \eta\, \delta_t\, e_{aj}$$

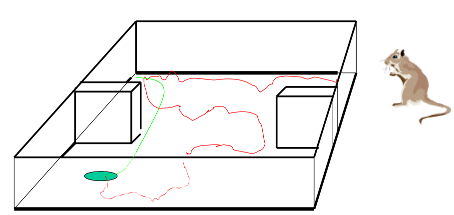


Spatial Representation

Example – mountain car



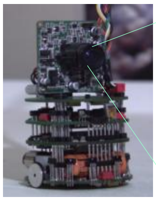
Biological Principles of Learning: spatial learning



Validating the Model

The KHEPERA mobile miniature robot

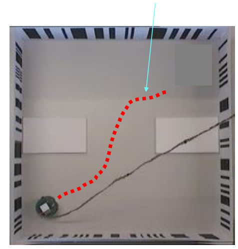
Experimental arena

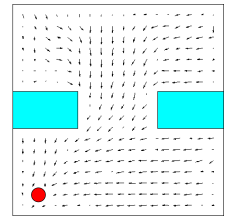


422 x 316 pixels

Open-field Navigation Experiments

(Biol. Cybern., 2000)





Navigation map after 20 training trials

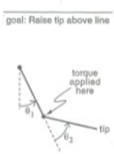


Figure 11.4 The acrobot.

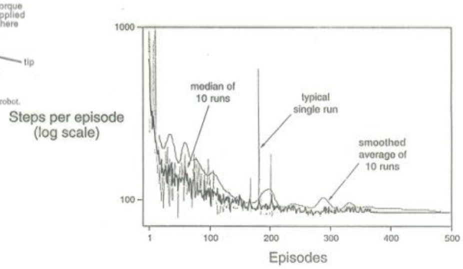


Figure 11.6 Learning curves for Sarsa(λ) on the acrobot task.

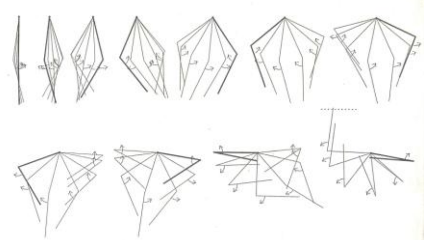


Figure 11.7 A typical learned behavior of the acrobot. Each group is a series of consecutive positions, the thicker line being the first. The arrow indicates the torque applied at the second joint.

The end