Natural Computation Methods in Machine Learning (NCML)

Lecture 8: Reinforcement Learning II

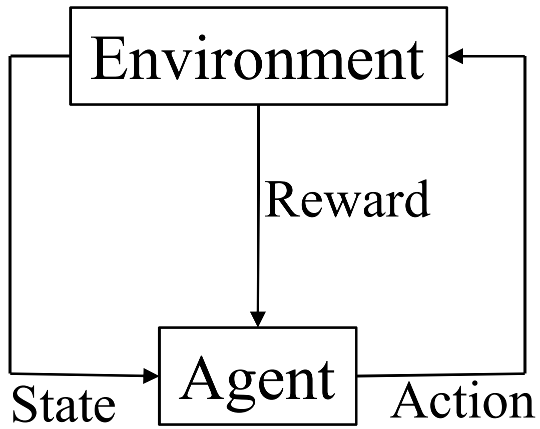
Temporal Difference Learning

UPPSALA UNIVERSITET

Reinforcement Learning

From previous lecture

- Learning by interaction with an environment, to maximize som long-term scalar value
  - (or to minimize a cost)
  - Learning by trial-and-error



UPPSALA UNIVERSITET

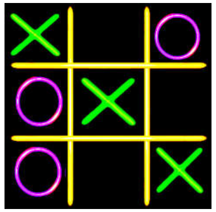
Challenges from last lecture

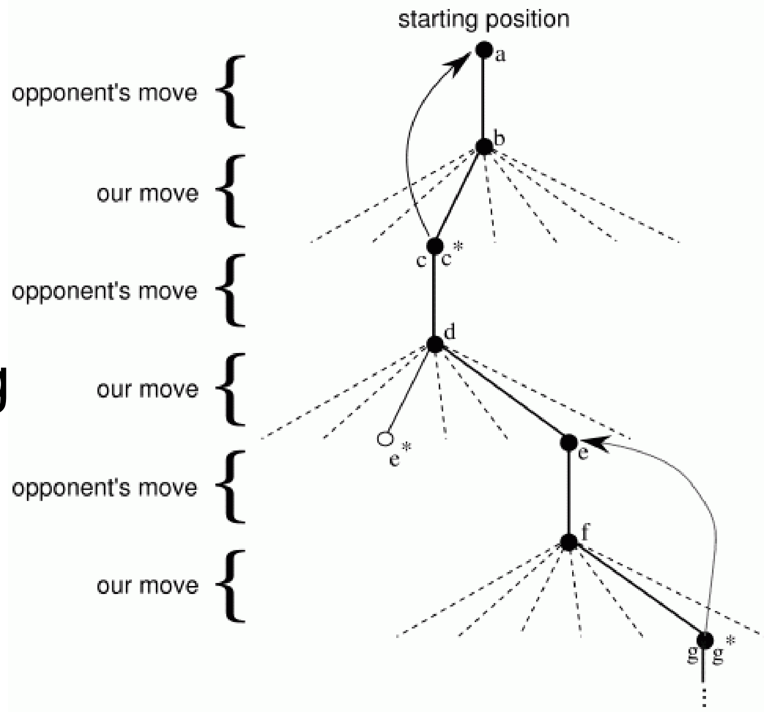
- Implement MENACE as a table of values, and try to make it learn by playing against itself!
  - Nowadays, this should work even if you don't remove illegal/symmetric states
- How many matchboxes would we need if we wanted MENACE to play Connect Four instead?
  - Upper bound $3^{42} \approx 10^{20}$
  - In practice $\approx 10^{13}$
- Does a reinforcement learning problem have to have an end state?
  - No, many RL problems don't

Tic-Tac-Toe game example

A temporal a difference learning (TD) perspective

- Assign an initial value to each game state:
  - +1 if the state is a winning terminal state
  - 0 if the state is a losing terminal state
  - +0.5 to all other states (terminal and non-terminal)
  - Possible interpretation: probability estimate of winning
- Then play many games ...





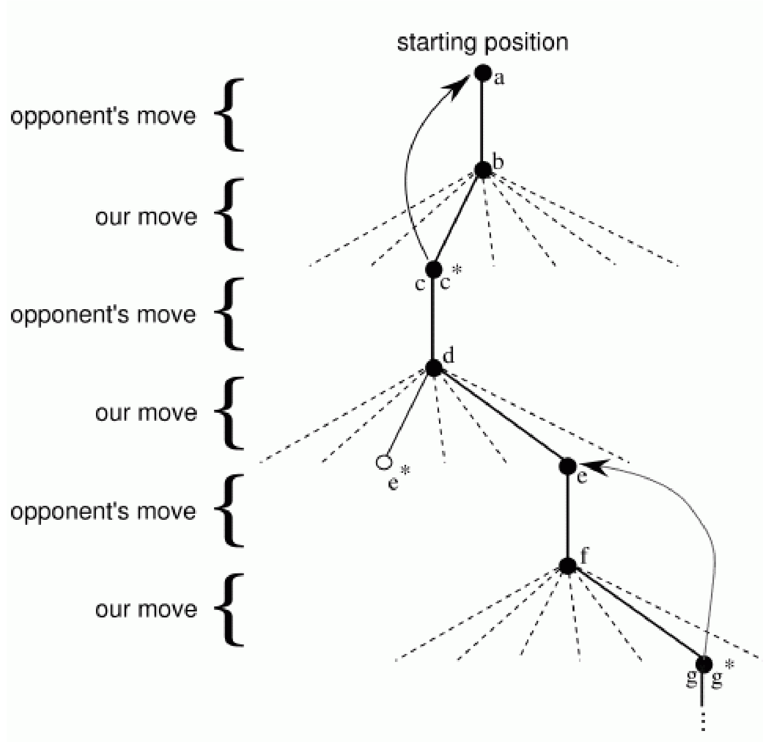
Tic-Tac-Toe game example

A temporal a difference learning (TD) perspective

- After each move, adjust the value of the previous 'state', s, towards the value of the current one, s'

  $$V(s) \coloneqq V(s) + \eta[V(s') - V(s)]$$

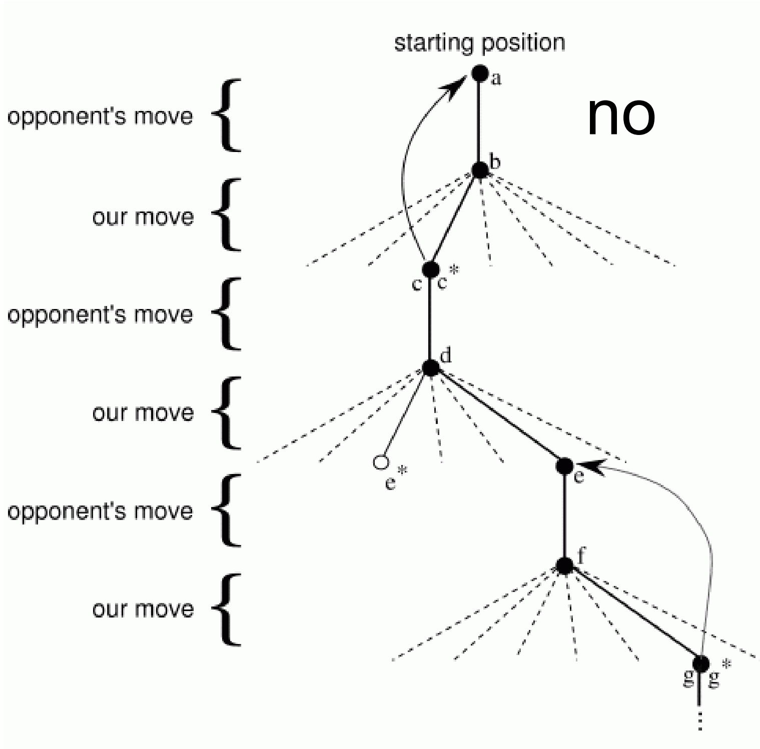
- If we play sufficiently many games, the pre-set terminal state values (0, 0.5, or 1) should move up the tree
- 'States' in this example are actually not states
  - They are 'afterstates'
  - We put values on the states we give the opponent, not our own



Tic-Tac-Toe game example

A temporal a difference learning (TD) perspective

- Note that, unlike MENACE, we don't wait until the game ends, before we update values!
  - TD is to learn from successive estimates
  - Useful, since many RL problems never ends (have no terminal states)
  - For some RL problems, terminal states exist, but the the goal is to <u>not</u> reach them!
    - Pole balancing, for example

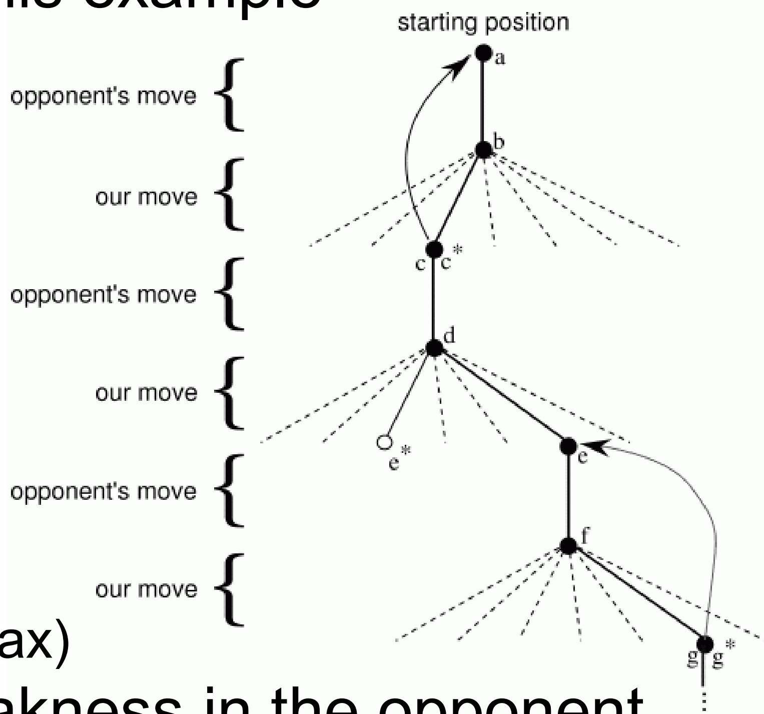


no

Tic-Tac-Toe game example

A temporal a difference learning (TD) perspective

- There are no explicit rewards in this example
  - We set values of terminal states instead, making this a prediction problem
  - TD is applicable also when we <u>only</u> want to predict (not necessarily to maximize, as in RL)
- Once converged, all necessary information about the future from state s is captured by $V(s)$
  - The agent never has to look more than one move ahead! (unlike Minimax)



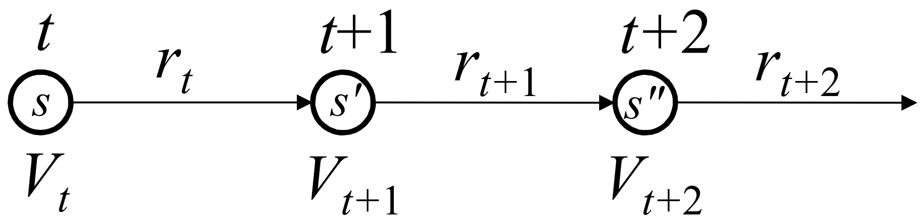
- The agent will learn to exploit weakness in the opponent
  - Unlike Minimax, which assumes that the opponent is optimal
  - On the other hand, this means that the agent may become too specialized (over-fitted) for a particular opponent

TD based Reinforcement Learning

- Last lecture, we defined the *Value* at time t as a discounted sum of expected future rewards:

$$V_t = E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right)$$

- These are values for states in a sequence:



- Note that the sum of all future rewards from $t+1$ is already captured by V_{t+1}, so we can simplify:

$$V_t = E(r_t + \gamma V_{t+1})$$

TD based Reinforcement Learning

Algebraic proof (skipping expected values for clarity)

$$V_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

move the first term out from the sum

$$V_t = \gamma^0 r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k}$$

and a γ

$$V_t = r_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k}$$

make the sum count from 0 again

$$V_t = r_t + \gamma \boxed{\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}}$$

$= V_{t+1}$, by definition

$$V_t = r_t + \gamma V_{t+1}$$

TD based Reinforcement Learning

Bellman's equation, 1957

$$V_t = E(r_t + \gamma V_{t+1})$$

- If all values are correct, this *relation* holds by definition of V_t
- If we drop the expected value-part, it should still hold on average:

$$V_t = r_t + \gamma V_{t+1}$$

- If the value estimates are bad, though, there will be a difference between the two sides
  - *the temporal difference error*

$$\text{r.h.s.} - \text{l.h.s.} = r_t + \gamma V_{t+1} - V_t$$

  - this can be used to update the value estimates

r.h.s = right hand side, l.h.s = left hand side

TD based Reinforcement Learning

TD(0) (Sutton, 1988)

- To minimize this error, we move the estimated value on the l.h.s. of

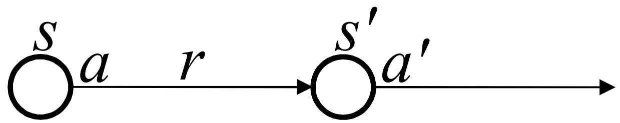
$$V_t = r_t + \gamma V_{t+1}$$ *relation*

$\Downarrow$

towards the r.h.s:

$$V_t \coloneqq V_t + \eta(r_t + \gamma V_{t+1} - V_t)$$ *update equation*

- This is the TD(0) update rule (Sutton, 1988)
  - The Tic-Tac-Toe example above was a special case where r=0 and γ=1 (and the 'states' were actually afterstates)
  - TD(0) is in turn a special case of TD(λ)
    - λ controls 'eligibility traces' (out-of-scope here)



Sarsa

Rummery & Niranjan, 1994

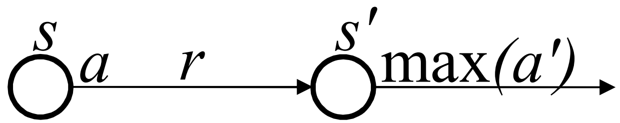
- It is often more convenient to associate values to state-action pairs, rather than just states
  - As we (kind of) did in the Tic-Tac-Toe example
- Q-values, $Q(s,a)$ estimate the value of doing action a in state s
- By the same reasoning as for TD(0), the relation

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

must hold if all Q-values are correct, leading to the update equation:

$$Q(s,a) \coloneqq Q(s,a) + \eta[r + \gamma Q(s',a') - Q(s,a)]$$

Tip: Don't confuse the relation with the update equation. The former is used to define the latter!



Q-Learning

Watkins, 1989

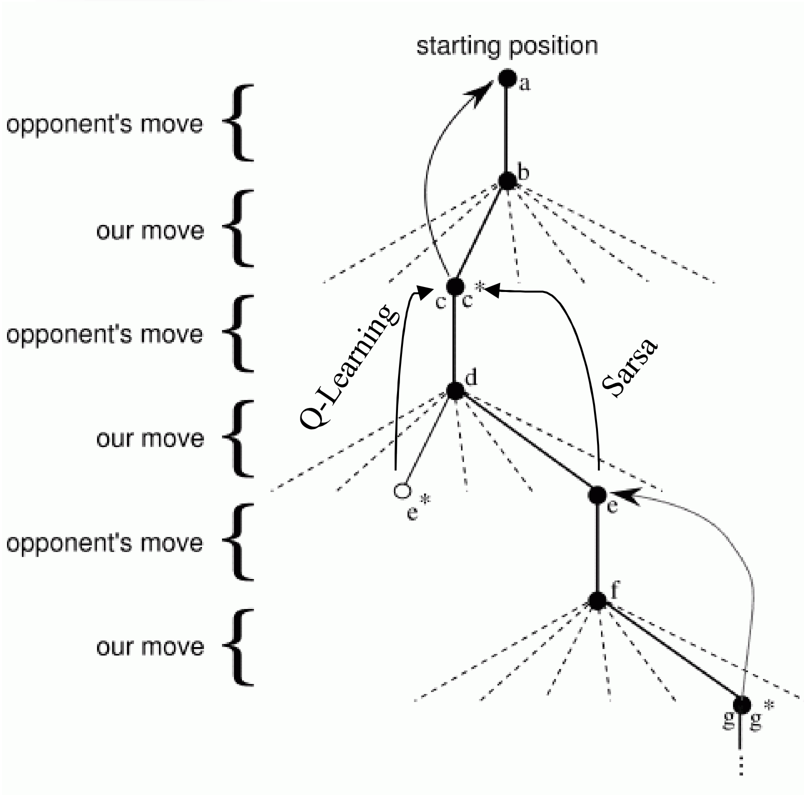
- The most commonly used RL algorithm
- As Sarsa (though Sarsa is younger), but with an assumption:
- Q(s,a) is the estimated value of doing action a in state s, <u>assuming that all future actions are greedy</u>:

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

$$\Downarrow$$

$$Q(s,a) \coloneqq Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

Comparing Sarsa to Q-Learning

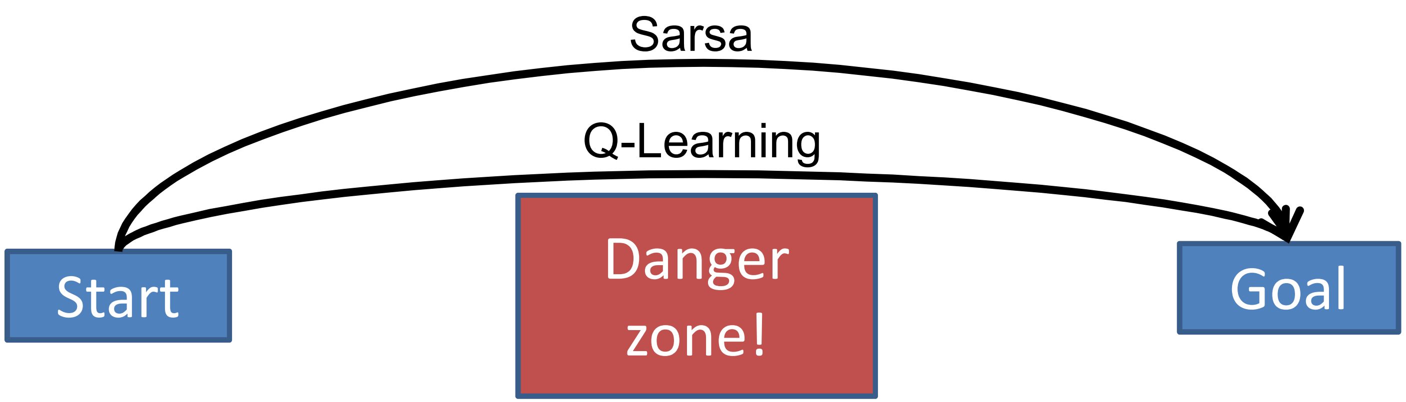


- Consider the Tic-Tac-Toe game tree again
- Arrows represent value updates
- Greedy choices marked by asterisks (*)
- Both existing arrows are valid for both Q-Learning and Sarsa
- But they would be different for the level where the arrow is missing (e to c)

Comparing Sarsa to Q-Learning

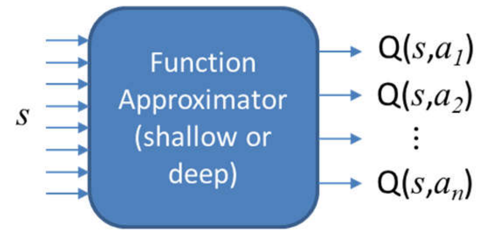
- Sarsa is more careful around dangerous parts of the state space
  - When Sarsa explores, bad experience will affect earlier values in the sequence
  - In Q-Learning, exploratory moves don't affect earlier values unless they turn out to be better than the old ones (becoming new greedy choices)

But Q-Learning still has to explore!



Dealing with large state spaces

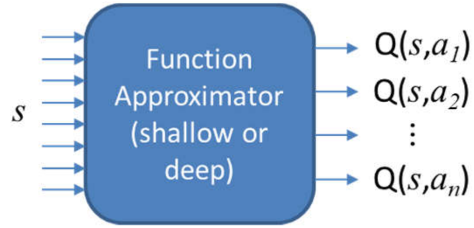
- Q-Learning and Sarsa (in this form) require huge tables with an entry for each state-action pair
- Most interesting problems have too many states!
  - Tic-Tac-Toe works (as shown), Connect 4 does not
  - Not only a storage issue, we also need time to visit all states, many times
- We need some form of state-aggregation, recognizing that two similar state-action pairs probably should have similar Q-values
  - We need generalization ability!
- Solution: Use a neural network!



Dealing with large state spaces

Using a neural network to estimate Q-values

- ANN with the state, s, on its inputs and n outputs, estimating the Q-values of the n possible actions in that state
  - Why n outputs? Why not just one, $Q(s, a)$, with a as an extra input?

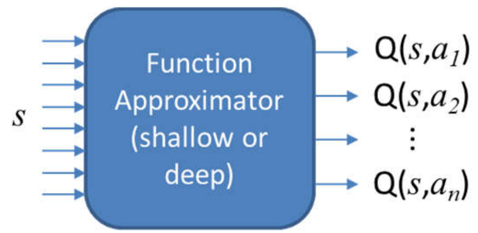


- Train by supervised learning with the r.h.s of the relation (not the update equation!) as target
  - So, for Q-Learning the target would be $r + \gamma \max_{a'} Q(s', a')$
- We can only do this for the selected action!
  - We don't know what would have happened if we had chosen the other ones, so we set the error of those outputs to 0 (i.e. we set target = output)

Dealing with large state spaces

Using a neural network to estimate Q-values

- Use linear output nodes!
  - This is function approximation, not classification
- Backprop not so good for this
  - Continuous learning v.s. a finite training set
  - For shallow networks, *Radial Basis Functions* work better (lecture 15)
  - Backprop can be made to work, though, by *Experience Replay* (also lecture 15)
  - Currently, Adam is also very popular (it was made for stochastic loss functions)
- Deep Q-Learning
  - Same thing – the network is just bigger



Lab 2

Reinforcement learning

- A robot navigating in a grid world
- Sarsa and Q-Learning
- Very small state and action spaces
  - Table based, so that we can see what's going on
- The environment is deterministic
  - Rarely the case for real applications
- Common mistakes
  - Confusing the relation with the update equation
  - Confusing exploration rate (ε) and discount factor (γ)
  - Thinking that Q-Learning is greedy (does not explore)