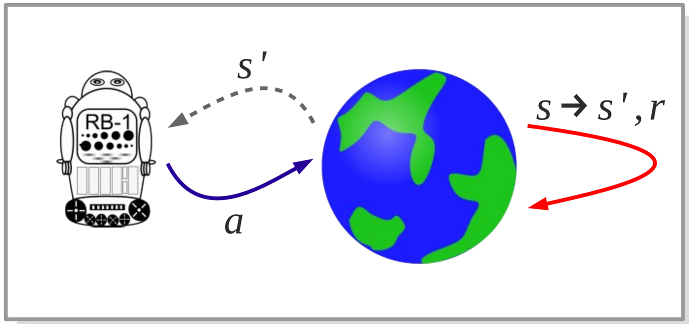
Probabilistic Artificial Intelligence

Lecture 9: (Model-Free) Reinforcement Learning

Slides, RN



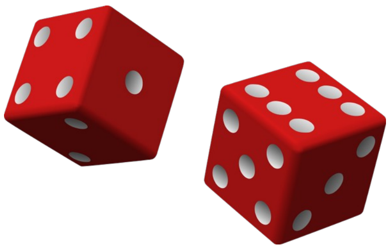
further reading: Sutton&Barto v2

http://incompleteideas.net/book/the-book-2nd.html

Dr. F. Oliehoek

Recap & Motivation

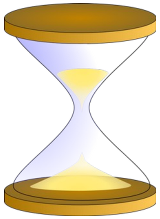
We saw that…
we need probability



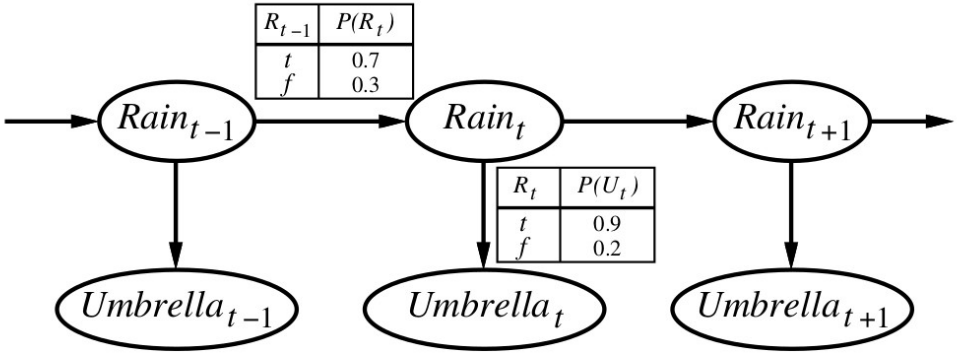
- For complex tasks: agent needs to maintain a belief
- Bruno de Finetti:
  - If agent’s beliefs violate the axioms of probability, then there exists a combination of bets against it which it is willing to accept that guarantees it will lose money, every time.

- Update a belief: Bayes’ rule

Beliefs over time

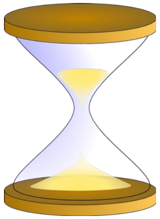


- To reason over time, we needed something more…
  - prediction!

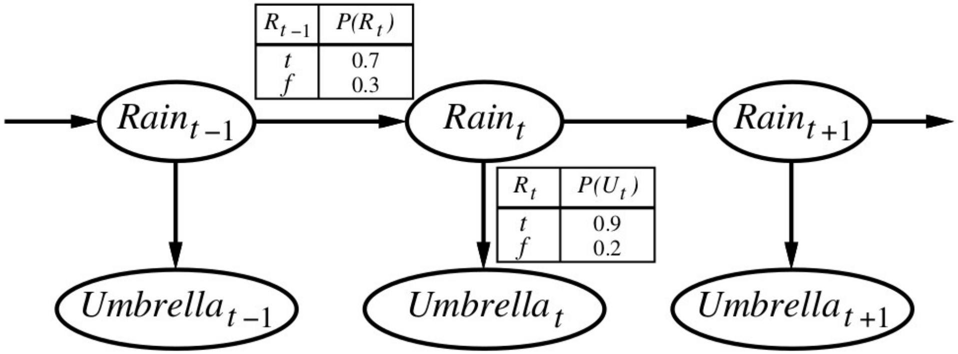


- Maintaining a belief (‘filtering’): still Bayes’ rule!
  - exploits conditional independence of … given the … ?

Beliefs over time



- To reason over time, we needed something more…
  - ▹ prediction!



- : still Bayes' rule!
  - nce of … given the … ?

And this 'inference' assumed that the CPTs were given!

When not the case…
→ try to **learn** CPTs from data
→ e.g., using the EM algorithm

TU Delft

Making ‘simple’ decisions…

- Decision theory = utility + probability
- Maximum expected utility, or maximum ‘value’
  - E.g., single shot case:

$$Q(a) = \sum_{s'} u(s') * P(s' | a)$$

TU Delft

Making 'simple' decisions…

- Decision theory = utility + probability
- Maximum expected utility, or maximum 'value'
  - E.g., single shot case:

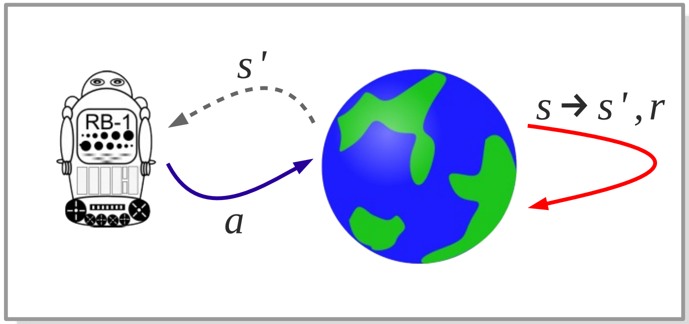
$$Q(a) = \sum_{s'} u(s') * P(s' | a)$$

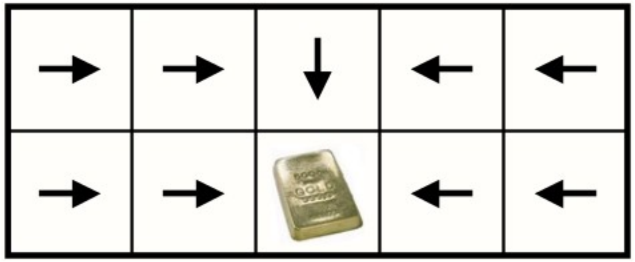
Notation…

- Russel&Norvig write 'U' for pretty much anything
- I write
  - 'u' for utility
  - 'Q' for expected utility (=value) of actions
  - 'V' for expected utility (=value) of other things

Complex decisions over time

- MDPs
  - now utility **of a trajectory**
  - e.g. discounted sum of rewards





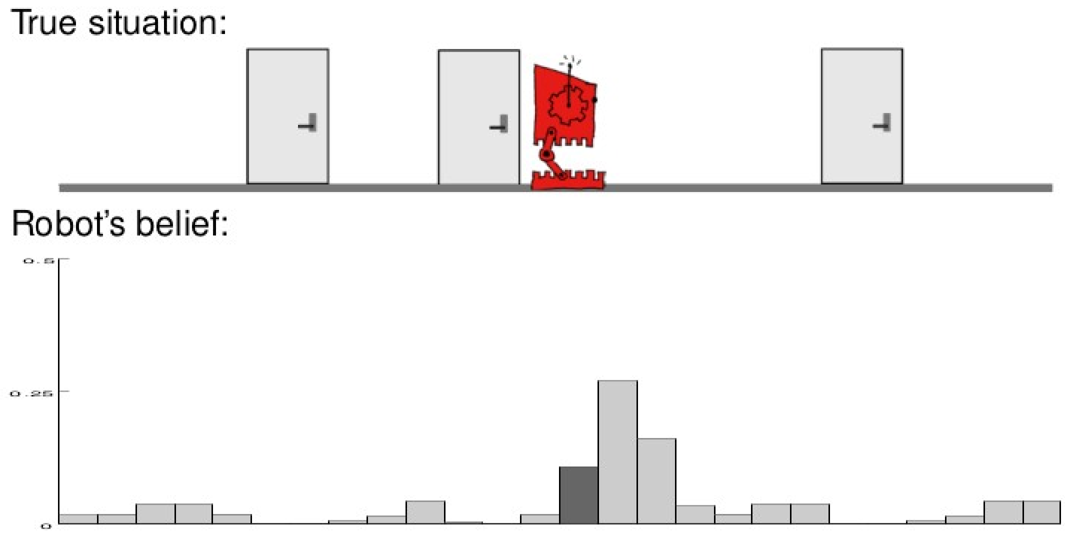
- Value = expected utility
- Bellman optimality equations
  - $V^*(s) = \max_a Q^*(s,a)$
  - $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$

TUDelft

Complex decisions over time

- MDPs
  - now util
  - e.g. disc
- Value = ex
- Bellman c
  - V*(s) = n
  - Q*(s,a) =

Even more complex when **partially observable**



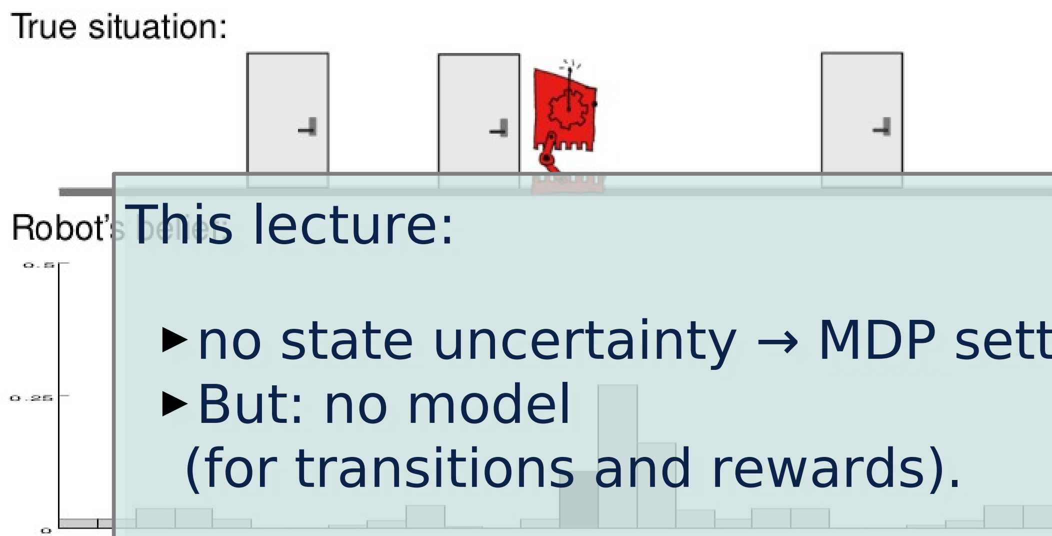
- now need to track the belief.
- i.e., belief is the state of a special MDP
  - called "belief MDP"

TUDelft

Complex decisions over time

- MDPs
  - now util
  - e.g. disc
- Value = ex
- Bellman c
  - V*(s) = n
  - Q*(s,a) =

Even more complex when **partially observable**



- now need to track the belief.
- i.e., belief is the state of a special MDP
  - called “belief MDP”

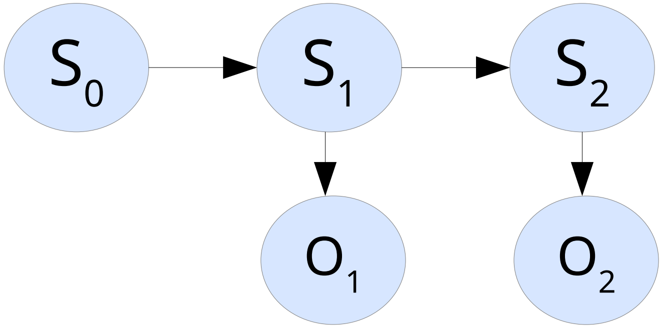
This lecture:

- no state uncertainty → MDP setting
- But: no model
  (for transitions and rewards).

→ Need to **learn**!

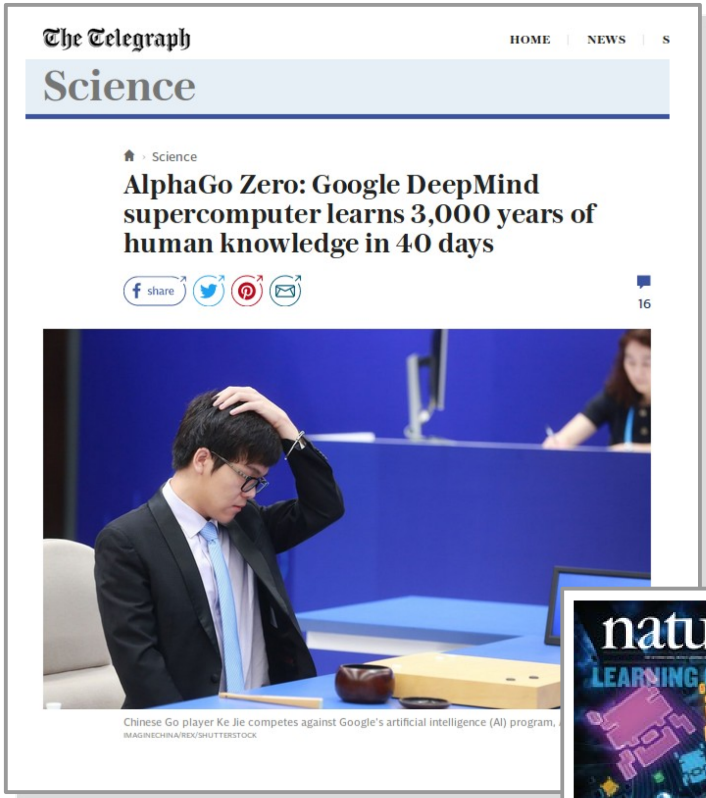
We did learning...

- learning = induction
- perspectives:
  - ‘idealistic’ (everything is Bayes’ rule)
  - ‘pragmatic’ (some type of optimization)
- learning latent variable models
  - using EM
  - e.g., learning parameters of an HMM
- many different learning settings
  - supervised learning
  - unsupervised
  - active learning
  - **reinforcement learning ← today**



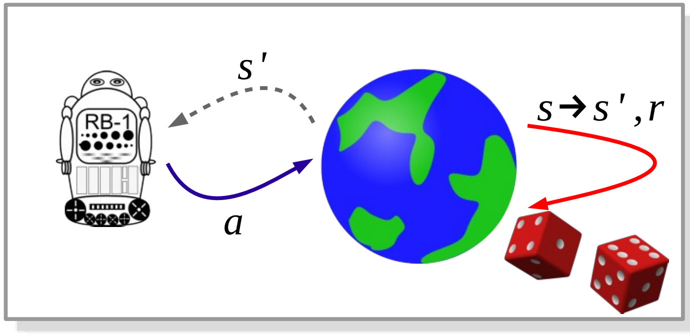
Outline

- Overview of RL
- Value-based, model-free methods:
  - “Passive learning”
    - policy evaluation
    - (without a model!)
  - “Active learning”
    - learning a good policy via Q*
    - exploration
  - Generalization
- Policy Search

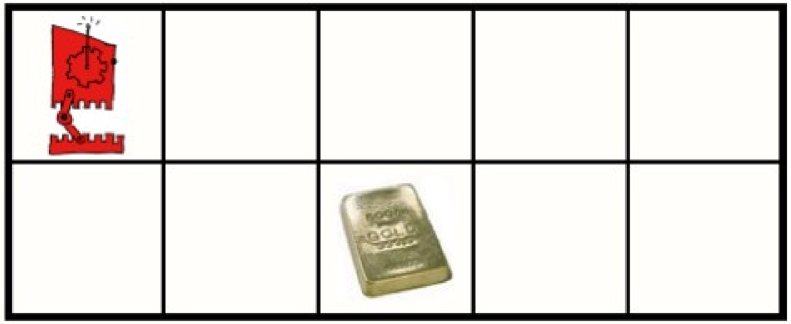

The Telegraph
HOME | NEWS | S
Science
Science
AlphaGo Zero: Google DeepMind supercomputer learns 3,000 years of human knowledge in 40 days
share
16
Chinese Go player Ke Jie competes against Google's artificial intelligence (AI) program,
IMAGINECHINA/REX/SHUTTERSTOCK

Overview of RL

MDP Planning



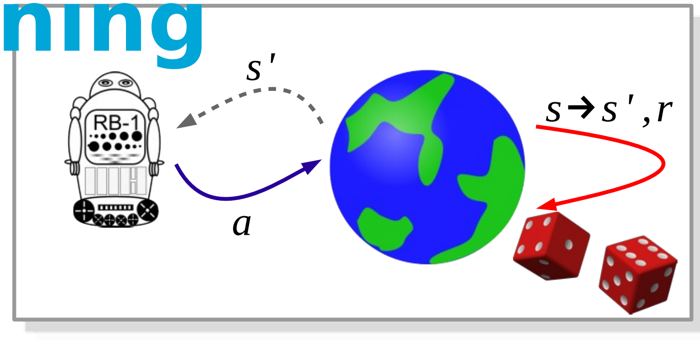
- Given an MDP:
  - *S* – set of states
  - *A* – set of actions
  - transition model: $p(s'|s,a)$
  - rewards: $R(s,a)$



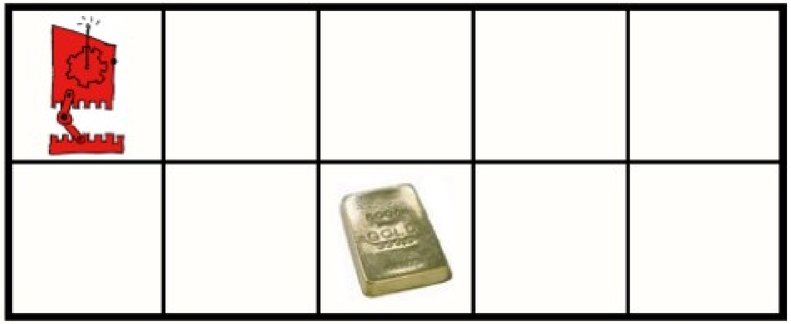
- Goal:
  - **compute** a policy π
  - that optimizes value V(π)

TUDelft

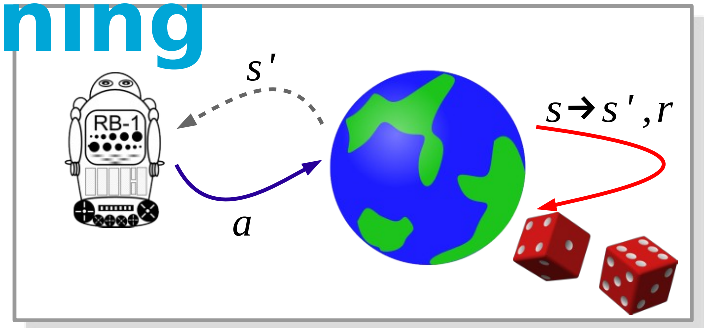
~~MDP Planning~~
Reinforcement Learning



- Given ~~an MDP~~:
  - *S* – set of states
  - *A* – set of actions
  - ~~transition model: *p(s'|s,a)*~~
  - ~~rewards: *R(s,a)*~~
- Goal:
  - ~~**compute**~~ **learn** a policy *π*
  - that optimizes value V(π)



~~MDP Planning~~
Reinforcement Learning



- Given ~~an MDP~~:
  - *S* – set of states
  - *A* – set of actions
  - ~~transition model: *p(s'|s,a)*~~
  - ~~rewards: *R(s,a)*~~
- Goal:
  - **~~compute~~ learn** a policy *π*
  - that optimizes value V(π)

We are talking about
general purpose learners

That is, we drop our agent in **any** MDP and it is able to work out how to be happy in it!

Example...

- You are in state 23
  - what do you want to do? (A or B)

Example…

- You are in state 23
  - what do you want to do? (A or B)
  - +14
- You are now in state 12
  - what do you want to do? (A or B)

Example...

- You are in state 23
  - what do you want to do? (A or B)
  - +14
- You are now in state 12
  - what do you want to do? (A or B)
  - -30
- You are in state 23 again
  - what do you want to do? (A or B)

Example…

- You are in state 23
  - what do you want to do? (A or B)
  - +14
- You are now in state 12
  - what do you want to do? (A or B)
  - -30
- You are in state 23 again
  - what do you want to do? (A or B)
  - -5

Overall Approaches...

- What we could do...

Overall Approaches...

- What we could do...
  - try and learn the model T, R
    - then can use planning (VI and PI etc.) afterwards
  - ...?

Overall Approaches...

- What we could do...
  - try and learn the model T, R (“model-based RL”)
    - then can use planning (VI and PI etc.) afterwards
  - directly try and learn the policy (“model-free RL”)
    - **value based**: learn Q*
    - **policy search**: ‘just optimize’ π directly

Overall Approaches...

- What we could do...
  - try and learn the model T, R (“model-based RL”)
    - then can use planning (VI and PI etc.) afterwards
  - directly try and learn the policy (“model-free RL”)
    - **value based**: learn Q*
    - **policy search**: ‘just optimize’ π directly

- S&B use ‘dynamic programming’ for ‘planning’
- relation:

model based RL != planning
but
model based RL uses planning

Policy Representations

- A policy π, in an MDP:
  states to actions $\pi{:}S \rightarrow A$

- How **represented**?

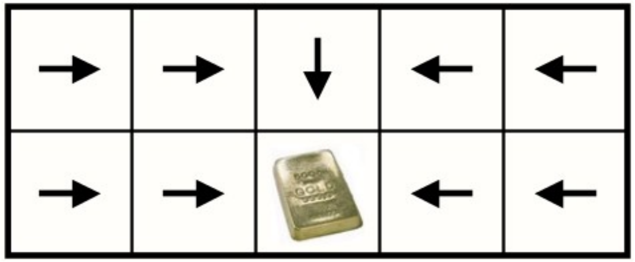
Policy Representations

- A policy π, in an MDP:
  states to actions $\pi : S \rightarrow A$

- How **represented**?
  - Lookup table

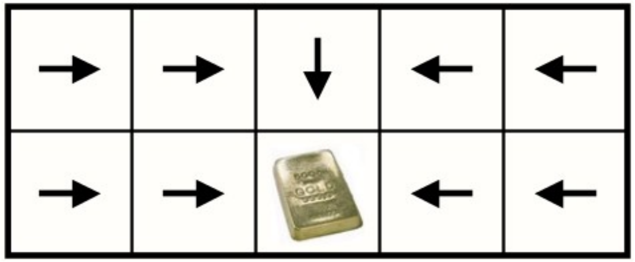
Policy Representations

- A policy π, in an MDP: states to actions $\pi{:}S \rightarrow A$
- How **represented**?
  - Lookup table

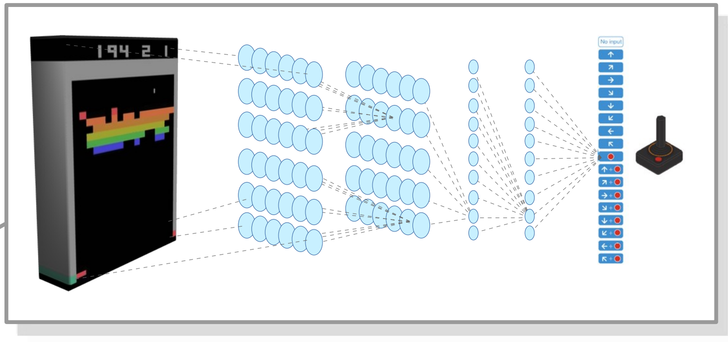


Policy Representations

- A policy π, in an MDP: states to actions $\pi{:}S \rightarrow A$

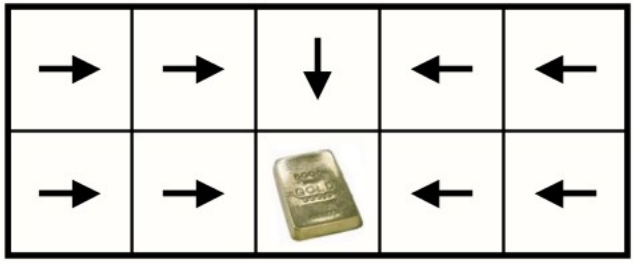


- How **represented**?
  - Lookup table
  - Computation
    - e.g., neural network

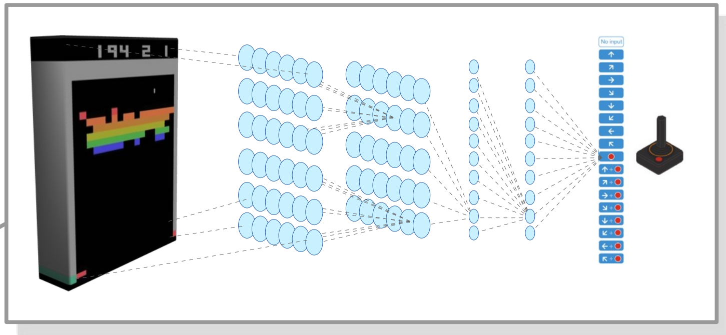


Policy Representations

- A policy π, in an MDP: states to actions $\pi : S \rightarrow A$



- How **represented**?
  - Lookup table
  - Computation
    - e.g., neural network
    - entire planning algorithm



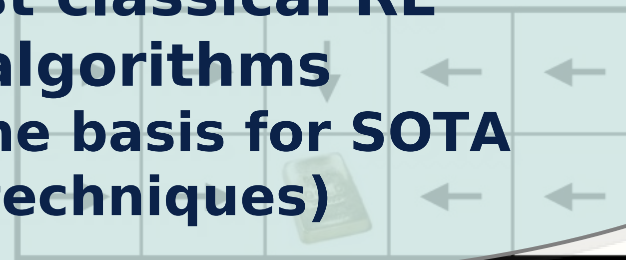
```
function MonteCarloPlanning(state)
repeat
    search(state, 0)
until Timeout
return bestAction(state,0)

function search(state, depth)
if Terminal(state) then return 0
if Leaf(state, d) then return Evaluate(state)
action := selectAction(state, depth)
(nextstate, reward) := simulateAction(state, action)
q := reward + γ search(nextstate, depth + 1)
UpdateValue(state, action, q, depth)
return q
```

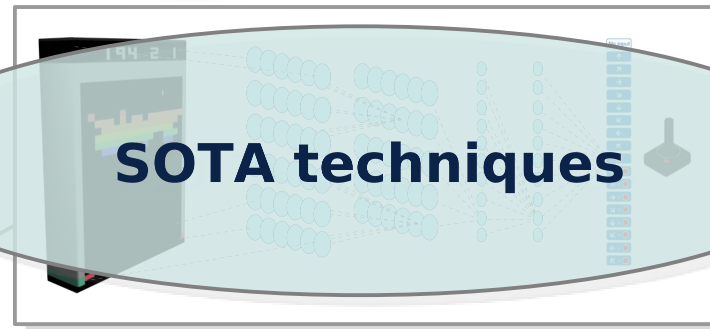
Again: Policy Representations

- A policy π, in an MDP: states to actions $\pi : S \rightarrow A$
- How **represented**?
  - Lookup table
  - Computation
    - e.g., neural network
    - entire planning algorithm

most classical RL algorithms (still the basis for SOTA techniques)



SOTA techniques

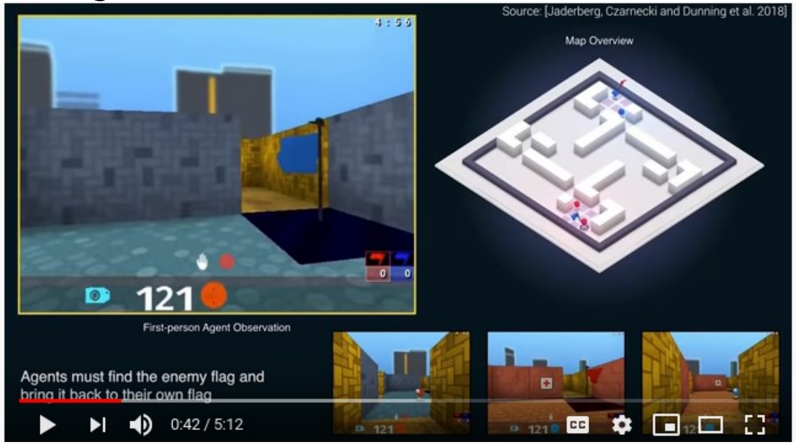


```
1: function MonteCarloPlanning(state)
2: repeat
3:     search(state, 0)
4: until Timeout
5: return bestAction(state,0)
...
8: if Leaf(state, d) then return Evaluate(state)
...
11: q := reward + γ search(nextstate, depth + 1)
12: UpdateValue(state, action, q, depth)
13: return q
```

learning to plan...! Future SOTA?

State of the Art…

- "Deep RL": Combination of RL techniques with deep neural networks
- Many recent results that demonstrate the power of these techniques. E.g.:
  - Atari Breakout:
    https://www.youtube.com/watch?v=V1eYniJ0Rnk
  - Pyramids:
    https://www.youtube.com/watch?v=yEcBMCU_VRU
  - Locomotion:
    https://www.youtube.com/watch?v=hx_bgoTF7bs
  - Dota:
    https://youtu.be/UZHTNBMAfAA?t=15
  - Capture the flag:
    https://www.youtube.com/watch?v=MvFABFWPBrw
  - Alpha Star:
    https://www.youtube.com/watch?v=cUTMhmVh1qs
  - Chip Design:
    https://ai.googleblog.com/2020/04/chip-design-with-deep-reinforcement.html
  - Summarizing books with human feedback:
    https://openai.com/blog/summarizing-books/
  - Stratego
    https://www.deepmind.com/blog/mastering-stratego-the-classic-game-of-imperfect-information
    https://www.volkskrant.nl/nieuws-achtergrond/ook-met-stratego-wint-kunstmatige-intelligentie-nu-van-de-topspelers-wat-betekent-dat~b86429be/
  - RL learns better sorting (2023)
    https://towardsdatascience.com/deep-reinforcement-learning-improved-sorting-algorithms-2f6a1969e3af



- Current trends:
  - meta-learning
  - 'off-line' reinforcement learning (from dataset)
  - many others

Interested?

→ I and others in the SDM group offer MSc projects on this topic

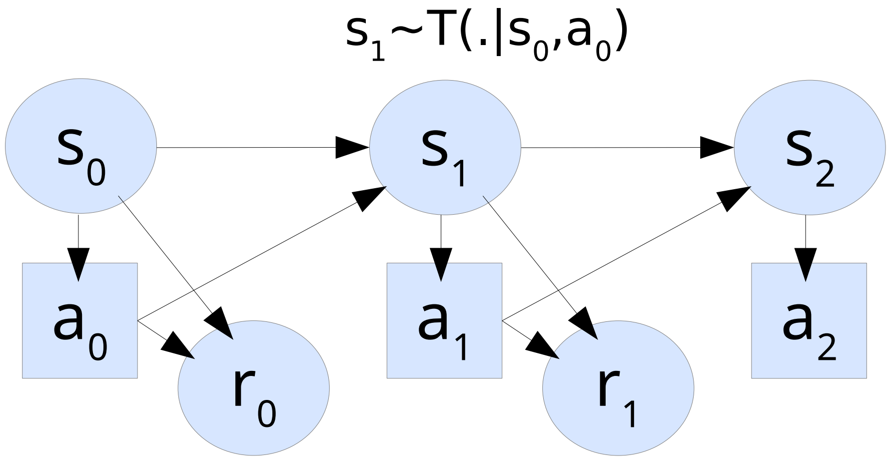
→ Take the deep RL course (CS4400)

“Passive learning”

model-free policy evaluation

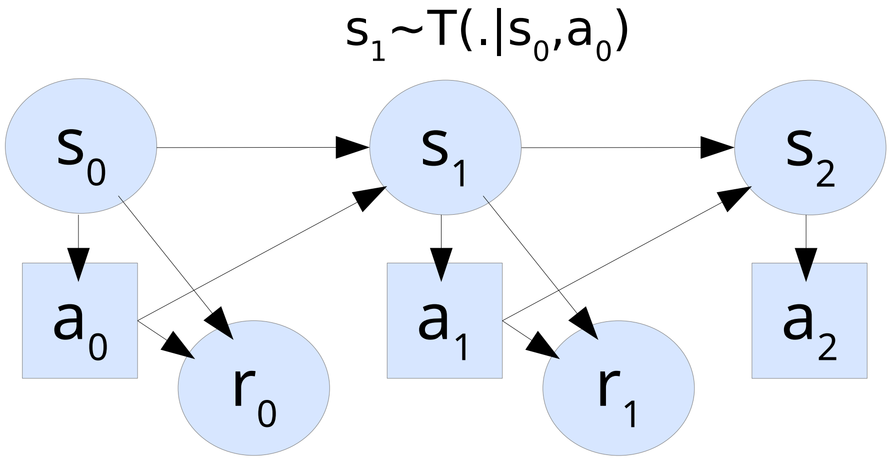
Fixing the Policy

- An MDP graphically

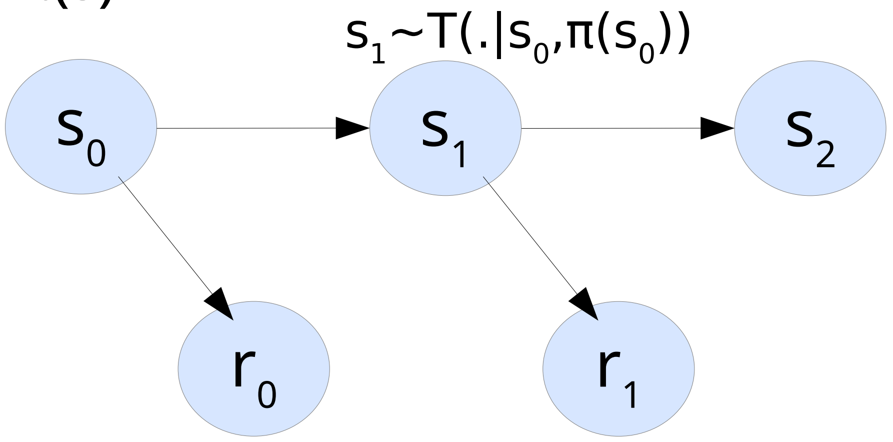


Fixing the Policy

- An MDP graphically

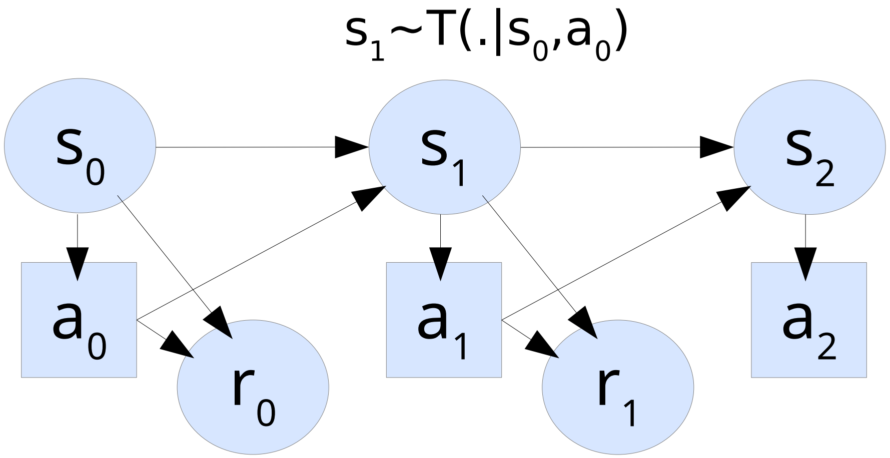


- Fixing the policy...
  - agent will always select $a=\pi(s)$
  - Induces a "Markov reward process"

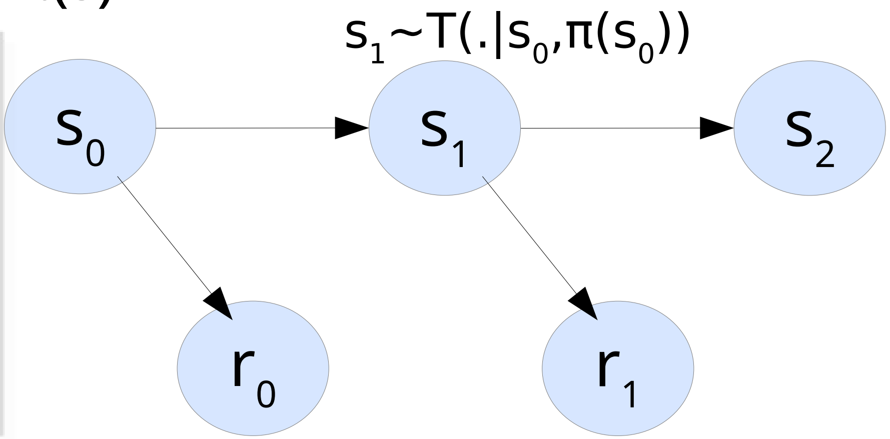


Fixing the Policy

- An MDP graphically



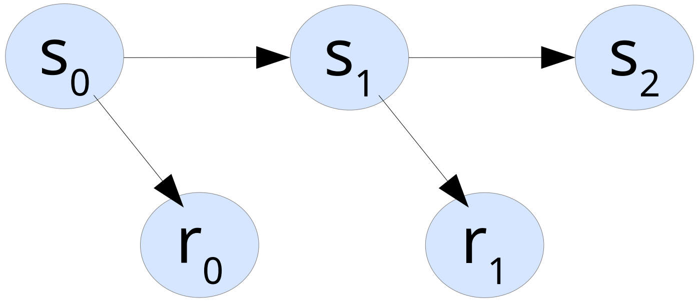
- Fixing the policy...
  - agent will always select $a=\pi(s)$



we are assuming:

- ***some*** **policy π is given to us.**
- e.g., random, or perhaps reasonable
  (think of "policy iteration" but without model)

Policy Evaluation

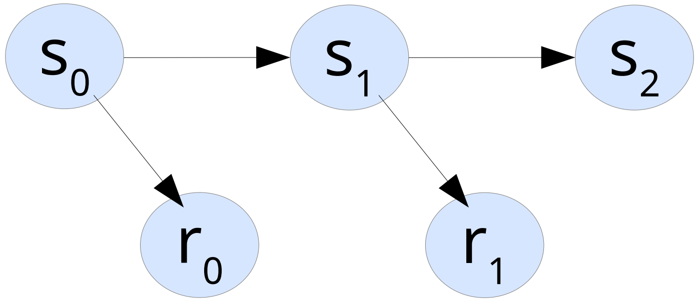


- Want to compute $V_\pi(s)$
- Given the model, this is easy...
  - E.g., successive approximation:

    $V_\pi(s) := R(s,\pi(s)) + \gamma \Sigma_{s'} T(s'|s,\pi(s)) V_\pi(s')$

  - or solve system of linear equations.
- But without model...? from interaction...?

Policy Evaluation



- Want to compute $V_\pi(s)$
- Given the model, this is easy...
  - E.g., successive approximation:

    $$V_\pi(s) := R(s,\pi(s)) + \gamma \sum_{s'} T(s'|s,\pi(s))\, V_\pi(s')$$

  - or solve system of linear equations.
- But without model...? from interaction...?

We will first consider this **policy evaluation** setting.

We will consider optimizing the policy (what actions to take) later

Idea 1: "direct utility estimation" [RN 23.2.1]
aka "Monte Carlo Prediction" [SBv2 5.1]

- Given a sequence $h=(s_0, r_0, s_1, r_1,\ldots,s_T, r_T)$
  - utility ("return"):
    - $u(h) = r_0 + r_1 + r_2 + \ldots + r_T$
    - $u(s_t) = r_t + r_{t+1} + r_{t+2} + \ldots + r_T$

Discounted:
- $u(h) = r_0 + \gamma\, r_1 + \gamma^2\, r_2 + \ldots + \gamma^T\, r_T$
- $u(s_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots + \gamma^T\, r_T$

- $V_\pi(s)$ is the ***expected*** **return**
  - $V_\pi(s) = \boldsymbol{E}\,[\, u(s) \mid \pi\,]$
  - from s, when executing π, until the end of the episode

So...?

Idea 1: "direct utility estimation" [RN 21.2.1]
aka "Monte Carlo Prediction" [SBv2 5.1]

- Given a sequence $h=(s_0, r_0, s_1, r_1,\ldots,s_T, r_T)$
  - utility ("return"):
    - $u(h) = r_0 + r_1 + r_2 + \ldots + r_T$
    - $u(s_t) = r_t + r_{t+1} + r_{t+2} + \ldots + r_T$

Discounted:
- $u(h) = r_0 + \gamma\, r_1 + \gamma^2\, r_2 + \ldots + \gamma^T\, r_T$
- $u(s_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots + \gamma^T\, r_T$

- $V_\pi(s)$ is the ***expected* return**
  - $V_\pi(s) = \mathbf{E}\,[\,u(s)\mid \pi\,]$
  - from s, when executing π, until the end of the episode

- Treat as a supervised problem!
  - execute the policy many times
  - estimate $V_\pi(s)$, for each s, based on the observed returns

Example

P(intended)=0.8
P(sideways)=0.2

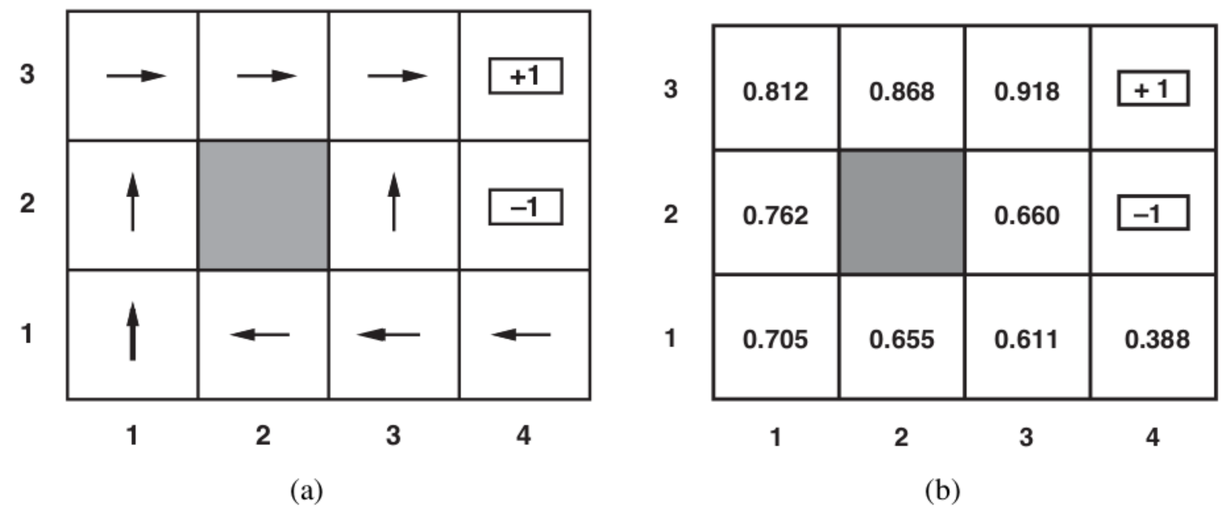


Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

- different episodes, or ‘trials’:

$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (2,3)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (4,3)_{\mathbf{+1}}$$
$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (2,3)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (3,2)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (4,3)_{\mathbf{+1}}$$
$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (2,1)_{\mathbf{-.04}} \rightsquigarrow (3,1)_{\mathbf{-.04}} \rightsquigarrow (3,2)_{\mathbf{-.04}} \rightsquigarrow (4,2)_{\mathbf{-1}}\ .$$

Example

P(intended)=0.8
P(sideways)=0.2

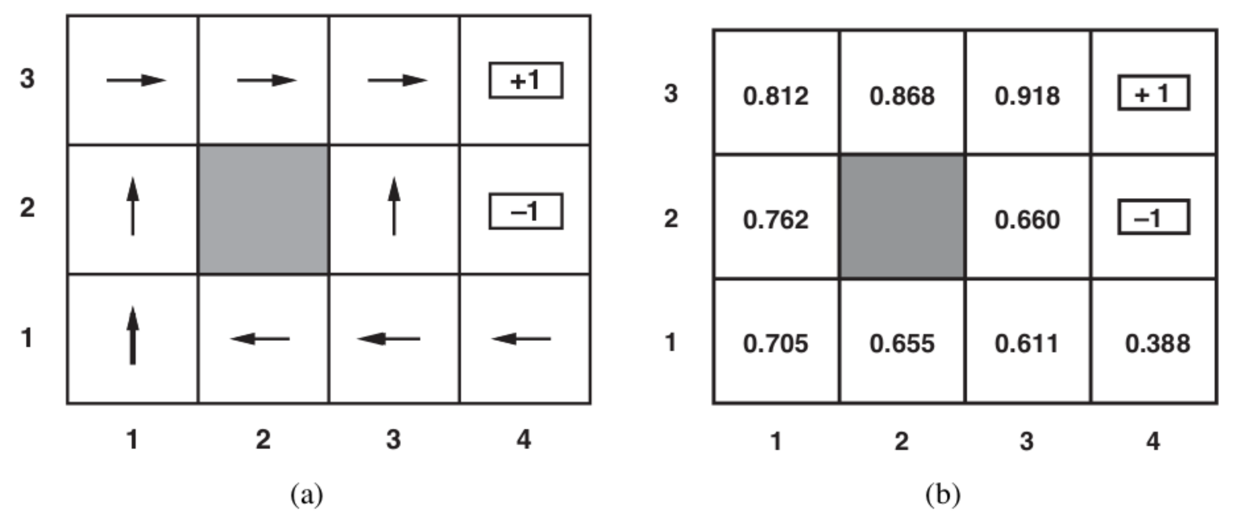


Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

- different episodes, or 'trials':

$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}} \ .$$

Value for (1,3)
without discounting?

Example

P(intended)=0.8
P(sideways)=0.2

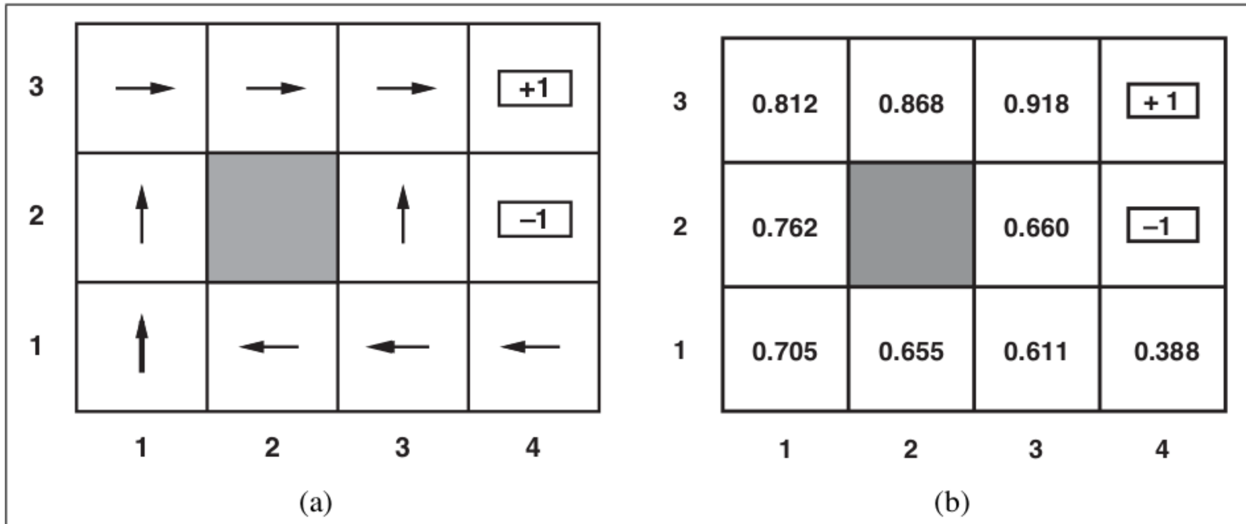


Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

- different episodes, or 'trials':

$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (2,3)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (4,3)_{\mathbf{+1}}$$
$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (1,2)_{\mathbf{-.04}} \rightsquigarrow (1,3)_{\mathbf{-.04}} \rightsquigarrow (2,3)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (3,2)_{\mathbf{-.04}} \rightsquigarrow (3,3)_{\mathbf{-.04}} \rightsquigarrow (4,3)_{\mathbf{+1}}$$
$$(1,1)_{\mathbf{-.04}} \rightsquigarrow (2,1)_{\mathbf{-.04}} \rightsquigarrow (3,1)_{\mathbf{-.04}} \rightsquigarrow (3,2)_{\mathbf{-.04}} \rightsquigarrow (4,2)_{\mathbf{-1}} \; .$$

Value for (1,3)
without discounting?

depends...
→ "first visit" or "every visit" MC?

Properties of Monte Carlo Estimation

- Works if you have enough samples
- It is **unbiased** → will converge to right solution
- But **high variance** → need many samples in practice

- It does **not** exploit any knowledge of the Bellman equation:
  - $V_\pi(s) = R(s,\pi(s)) + \gamma \Sigma_{s'} T(s'|s,\pi(s)) V_\pi(s')$
  - but we know that the values should adhere to it...

TUDelft

Temporal Difference Learning

- So perhaps we can **use Bellman equation to make things more efficient…?**

Note: still **policy evaluation** setting!

Temporal Difference Learning

- So perhaps we can **use Bellman equation to make things more efficient…?**

- First: **incremental form of MC estimation**
  - maintain an estimate V_k of V_π
  - when we get a new return u(s) from s, update using:

    $$V_{k+1}(s) := (1-\alpha_k)\, V_k(s) + \alpha_k\, u(s)$$

  - "move in direction of u(s)"
  - this is called "stochastic gradient descent" (SGD)
  - converges when picking appropriate step sizes α_k

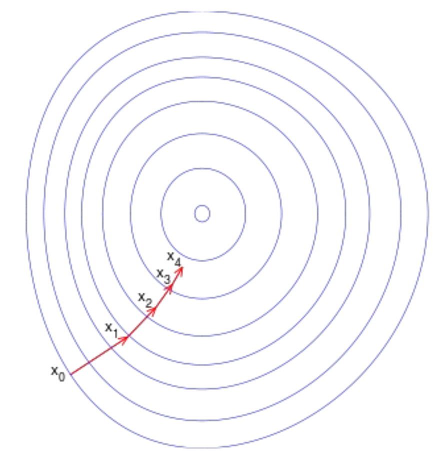
Temporal Difference Learning

How is this SGD? [S&B v2 9.3]

Our estimation $V_k(s)$ is trivially parameterized: $V_k(s) = V(s;w_k) = w_k$

→ note: 's' is fixed! its value is w_k

If we knew $V_\pi(s)$ we could minimize $[V_\pi(s) - V(s;w_k)]^2$ via GD:



$$
\begin{aligned}
w_{k+1} &:= w_k - \alpha \nabla [V_\pi(s) - V(s;w_k)]^2 / 2 \\
&= w_k + \alpha [V_\pi(s) - V(s;w_k)] \nabla V(s;w_k) \\
&= w_k + \alpha [V_\pi(s) - V(s;w_k)] \\
&\quad \{\text{since } V(s;w_k) = w_k\}
\end{aligned}
$$

We don't know $V_\pi(s)$, but turns out that SGD only needs an unbiased estimate, such as u(s), so we can do:

$$w_{k+1} := w_k + \alpha [\, u(s) - V(s;w_k) \,]$$

Translating back:

$$V_{k+1}(s) := V_k(s) + \alpha [\, u(s) - V_k(s) \,] = (1-\alpha)\, V_k(s) + \alpha\, u(s)$$

Bootstrapping the target

- SGD updates estimate towards **target *T(s)***
  - $V_{k+1}(s) := (1-\alpha_k)\, V_k(s) + \alpha_k\, \boldsymbol{T(s)}$
  - for MC updates:

    $$T(s) = u(s) = r_t + r_{t+1} + \ldots + r_T$$

  - What if we don't want to wait until end of episode?

Bootstrapping the target

- SGD updates estimate towards **target *T(s)***
  - $V_{k+1}(s) := (1\text{-}\alpha_k)\, V_k(s) + \alpha_k\, \mathbf{T(s)}$
  - for MC updates:

    $$T(s) = u(s) = r_t + r_{t+1} + \dots + r_T$$

  - What if we don't want to wait until end of episode?

- **Suppose we would know $V_\pi(s_{t+1})$...**

  ...could do: $T(s) = r_t + \gamma\, V_\pi(s_{t+1})$
  - could immediately update after seeing s_{t+1}
  - would still be unbiased: $V_\pi(s) = r_t + \gamma\, E_{s'}[\, V_\pi(s')\,]$
  - and would be an update with less variance

TU Delft

Bootstrapping the target

- SGD updates estimate towards **target *T(s)***
  - $V_{k+1}(s) := (1-\alpha_k)\, V_k(s) + \alpha_k\, \mathbf{T(s)}$
  - for MC updates:

    $$T(s) = u(s) = r_t + r_{t+1} + \ldots + r_T$$

  - What if we don't want to wait until end c

Key idea: "**bootstrapping**"
- we don't know $V_\pi(s_{t+1})$
- but can approximate it using our current estimate $V_k(s_{t+1})$

- **Suppose we would know $V_\pi(s_{t+1})$…**

  …could do: $T(s) = r_t + \gamma\, V_\pi(s_{t+1})$
  - could immediately update after seeing s_{t+1}
  - would still be unbiased: $V_\pi(s) = r_t + \gamma\, E_{s'}[\, V_\pi(s')\,]$
  - and would be an update with less variance

TU Delft

Temporal Difference Learning
[RN 23.2.3, SBv2 6.1]

- TD updates, after a transition *(s,r,s')*:
  - $V_{k+1}(s) := (1-\alpha_k)\, V_k(s) + \alpha_k\, T(s)$
  - with target $T(s) = r + \gamma\, V_k(s')$
- Usual form (just rewriting, dropping '*k*'):
  - $V(s) := V(s) + \alpha\, [\, r + \gamma\, V(s') - V(s)\,]$

TU Delft

Temporal Difference Learning

- TD updates, after a transition *(s,r,s')*:
  - $V_{k+1}(s) := (1-\alpha_k)\, V_k(s) + \alpha_k\, T(s)$
  - with target $T(s) = r + \gamma\, V_k(s')$

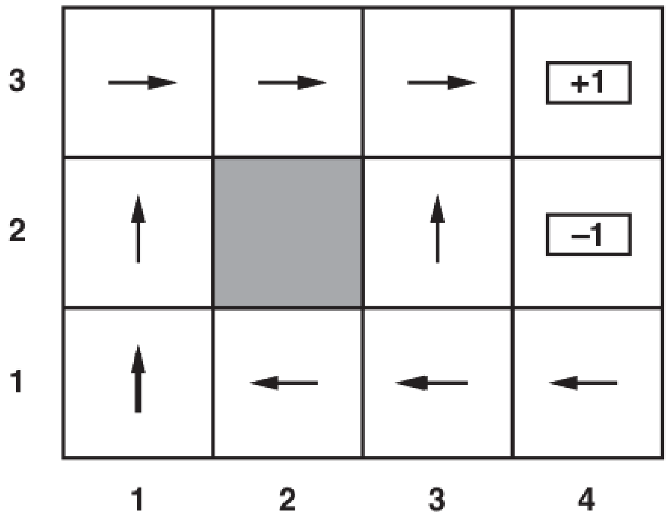
- Usual form (just rewriting, dropping '*k*'):
  - $V(s) := V(s) + \alpha\, [\, r + \gamma\, V(s') - V(s)\,]$

"TD error"
(difference between prediction after observing the next reward and state)

TUDelft

TD learning: properties
(cf. S&B v2 6.2,6.3)

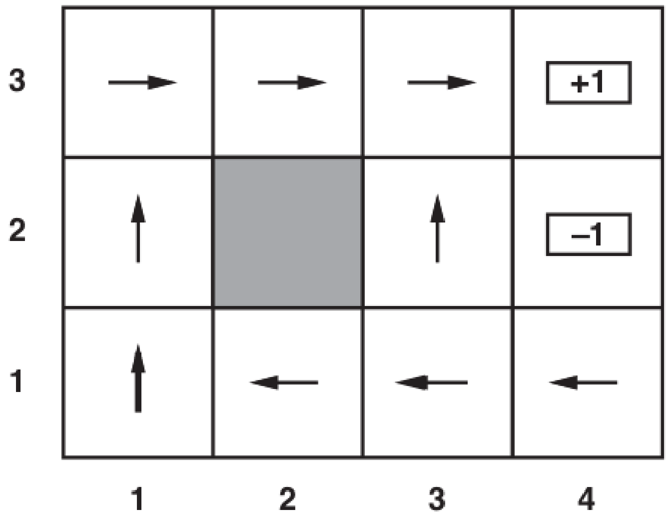
- Don't need to wait until end of episode to update values
- Updates have lower variance, but biased
  (due to bootstrapping)
- Converges to V_π
- Not (stochastic) gradient descent
  (but a "semi-gradient" method)
- how many episodes to estimate V(s_0=(1,1)) ?



TD learning: properties
(cf. S&B v2 6.2,6.3)

- Don't need to wait until end of episode to update values
- Updates have lower variance, but biased
  (due to bootstrapping)
- Converges to V_π
- Not (stochastic) gradient descent
  (but a "semi-gradient" method)

- how many episodes to
  estimate V(s_0=(1,1)) ?
  - a solution: 'eligibility traces'
    (cf. SBv12 chap.12)



“Active learning”

model-free, value-based control

Control… to *optimize* actions!
[RN 23.3.3]

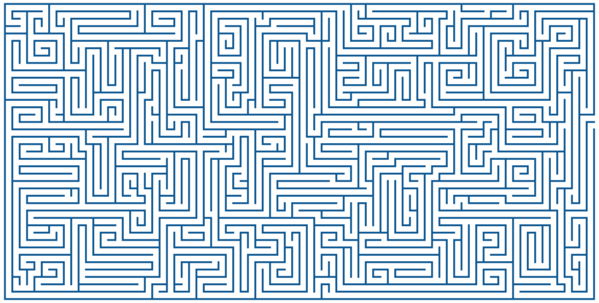
We now leave the **policy evaluation** setting and switch to the **control** setting

- OK, so now we can learn $V_\pi(s)$… great…?
  - we want V*
  - and really Q*(s,a), such that we can take actions…!
- If one would learn the model… (next week)
  - can compute Q from V by ‘backprojection’:
    $Q(s,a) = R(s,a)+\gamma\ \Sigma_{s'}\ P(s'|s,a)V(s')$
  - but then, why not apply VI / PI directly…?
- Instead, try and learn Q*(s,a) directly!

TU Delft

Q-learning

- TD-learning: after (s,r,s') we update
  - $V(s) := V(s) + \alpha [r + \gamma V(s') - V(s)]$
- Q-learning: after (s,a,r,s') we update
  - $Q(s,a) := Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- Exploration…! [RN 23.3.1]
  - Now need to try out all actions…
  - A simple rule is **epsilon greedy**:
    - random action with prob. ε,
    - greedy action otherwise
  - But active research topic!



SARSA

- Q-learning learns **off-policy:**
  value of target does not depend the followed policy
  - due to the maximization:

    $Q(s,a) := Q(s,a) + \alpha\, [\, r + \gamma \max_{a'} Q(s',a') \; - \; Q(s,a)\,]$

  - converges to Q*

- Alternatively **SARSA** learns **on-policy**:
  - about the policy π we are following
  - after (s,a,r,s',a') we update:

    $Q(s,a) := Q(s,a) + \alpha\, [\, r + \gamma\, Q(s',a') \; - \; Q(s,a)\,]$

  - converges to Q_π
  - if we want to learn Q*, need to adapt $\pi \rightarrow \pi^*$

SARSA

- Q-learning learns **off-policy:**
  value of target does not depend the foll
  - due to the maximization:

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

  - converges to Q*

- Alternatively **SARSA** learns **on-policy**:
  - about the policy π we are following
  - after (s,a,r,s',a') we update:

    $Q(s,a) := Q(s,a) + \alpha\, [\, r + \gamma\, Q(s',a') - Q(s,a)\,]$

  - converges to Q_π
  - if we want to learn Q*, need to adapt $\pi \rightarrow \pi^*$

What to use…?

- off-policy learning has great promise: learn about optimal values following **any** policy
- but when using 'function approximation' SARSA tends to be more stable
- "deadly triad" of RL:
  combining
  1) bootstrapping,
  2) off-policy learning, and
  3) function approximation

Function Approximation

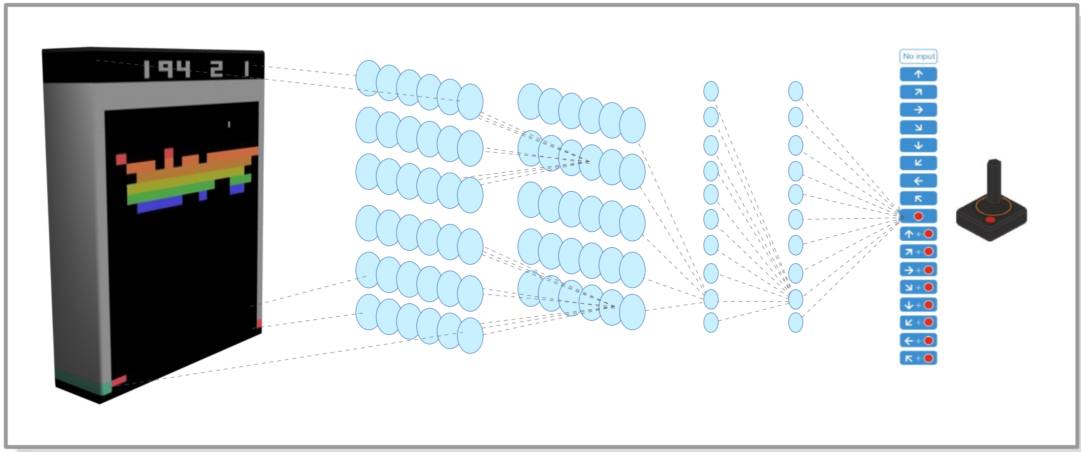
[RN 23.4]

Scaling up…?

- Methods covered so far are **tabular**
  - Q(s,a) values for each (s,a) in a table

- But MDPs are huge…!
  (e.g., number of possible screens in Atari?)

- use function approximation to scale up!
  - e.g., represent Q(s,a) with a deep neural network



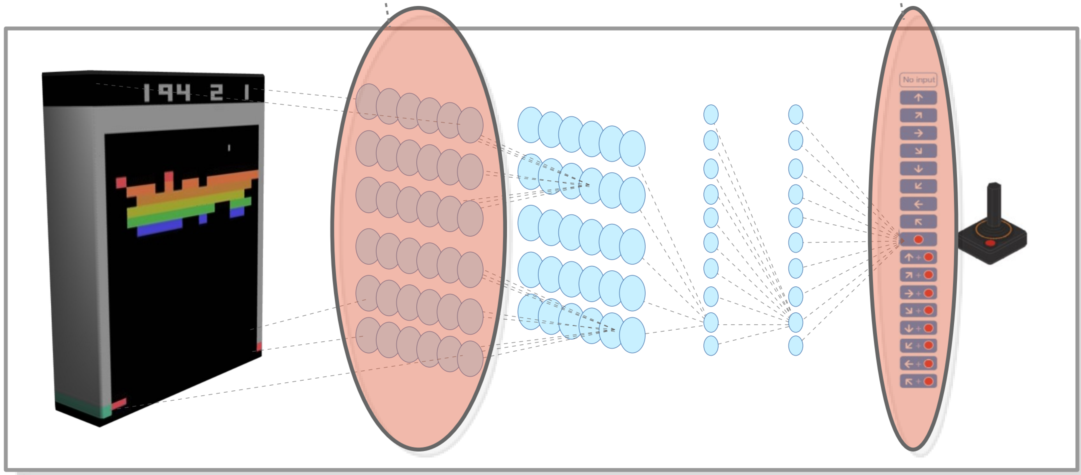
TU Delft

Scaling up...?

- Methods covered so far are **tabular**
  - Q(s,a) values for each (s,a) in a table
- But MDPs are huge...!
  (e.g., number of possible screens in Atari?)
- use function approximation to scale up!
  - e.g., represent Q(s,a) with a deep neural network

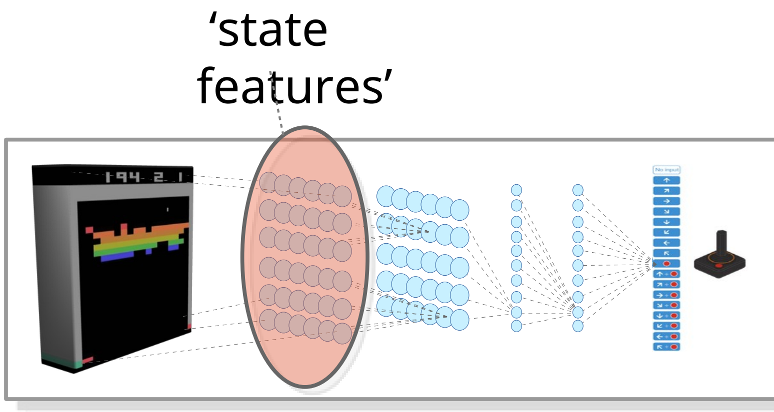
map from 'state features'

to values for each action



Function approximation for RL

- For simplicity: policy evaluation
  - we want to find $V_\pi(s;w)$
  - w is a vector of parameters
- Using features:
  - $x(s)=(x_1(s),\ldots,x_d(s))$ is a d-dimensional feature vector
  - $V_\pi(s;w) = f(x(s); w)$
- For instance…
  - f can be a neural network
  - or a linear function $V_\pi(s;w) = w^T x(s)$



Function approximation in TD

- F.A. in TD
  - target is $T(s) = r + \gamma V_k(s')$
  - we minimize the squared TD error:

$$w_{k+1} := w_k - \alpha \nabla [T(s) - V(s;w_k)]^2 / 2$$
$$= w_k + \alpha [r + \gamma V(s';w_k) - V(s;w_k)] \nabla V(s;w_k)$$

- E.g., for linear function approximation:
  - $\nabla V(s;w_k) = \nabla w^T x(s) = x(s)$
  - $w_{k+1} := w_k + \alpha [r + \gamma V(s';w_k) - V(s;w_k)] x(s)$

- Q-learning is adapted similarly

TU Delft

Properties of RL with F.A.

- What is optimal under function approximation?
  - ‘best’ value function in class
  - under some metric…

- No true gradient descent → few of the nice properties remain…
  - linear function approximation:
    - TD learning: **converges** to bounded approximation
    - SARSA: might **‘chatter’** (cycle around the optimal solution) but not diverge
    - Q-learning: **can diverge** (‘deadly triad’!)
  - **non-linear** function approximation:
    - none of these methods have guarantees

- Possible solutions:
  - more principled algorithms – some exist, but in practice not very effective.
  - practical tricks that help with convergence…

Tricks for Convergence

- Prototypical example: DQN [Mnih et al. 2015]

- Combines a number tricks:
  - experience replay
  - gradient clipping
  - ‘target network’

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.

Policy Search

[RN 23.5]

Policy Gradient Methods

- main idea: do not bother with value functions Q or V
- Instead,
  - directly parametrize policy $\pi(a|s\,;w)$
  - update these parameters based on the returns $u(s)$ observed.

- REINFORCE:
  - $w_{t+1} := w_t + \alpha\, u(s_t)\, \nabla \log \pi(a_t \mid s_t\,;w_t)$
  - See S&B v2 13.3

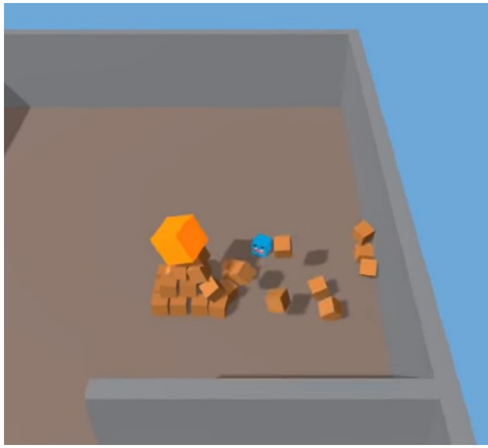
TU Delft

Actor-Critic Methods

- Policy gradient methods can be combined with estimated Q-value functions:
  - policy = actor → just tries to take good actions
  - value function = critic → gives feedback to policy

- This addresses the high variance that PG methods (working directly on returns) otherwise have.

- Versions of these methods have led to state-of-the-art performance in many domains



TU Delft

Summary

Summary

- RL: when we don't have a model
- Value-based, model-free methods
  - "Passive learning"
    - policy evaluation: Monte Carlo estimation, TD-learning
  - "Active learning"
    - use TD-learning to lean Q*
    - Q-learning, SARSA
    - exploration…!
  - Function approximation to scale up: generalization
- (also model free) Policy Search

TU Delft