Einführung ins Reinforcement Learning

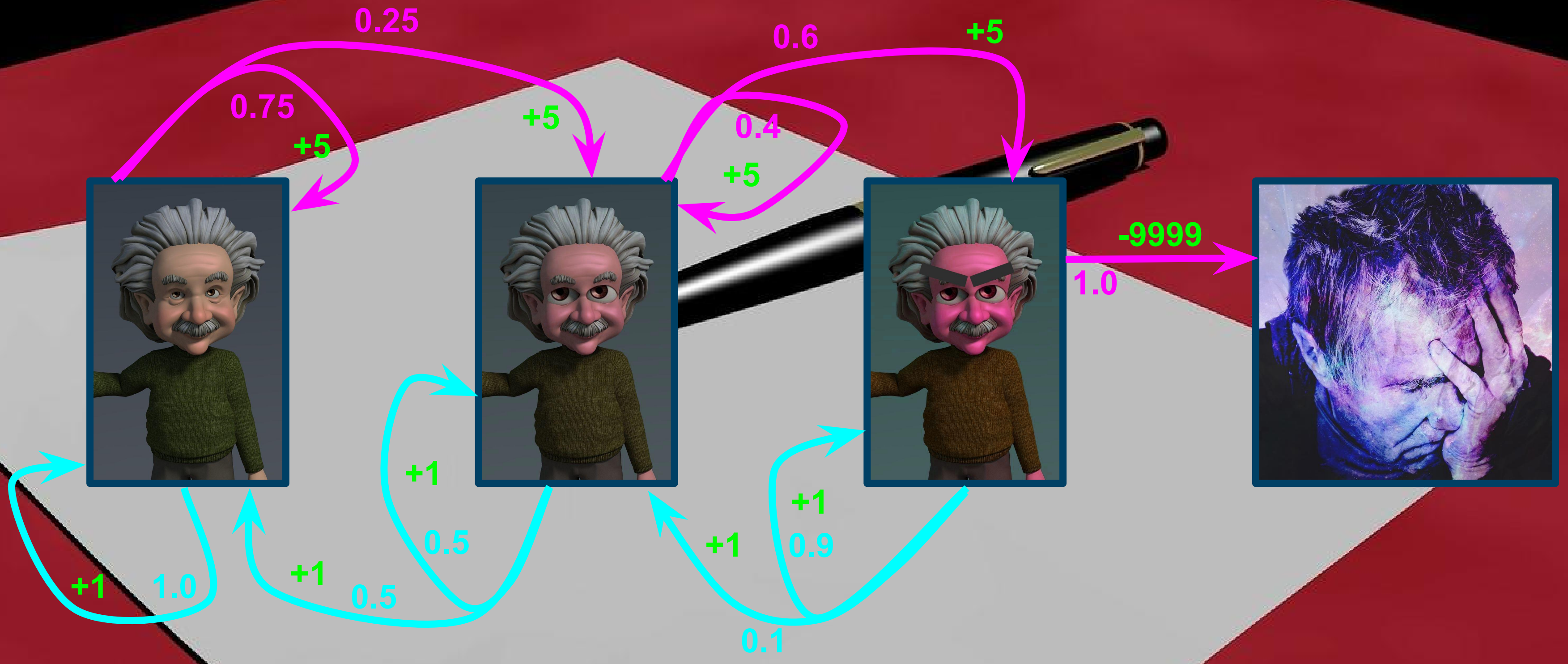
~ Patrick Dammann ~

~ Ist künstliche Intelligenz gefährlich? ~

how to cheat in an exam

cheat

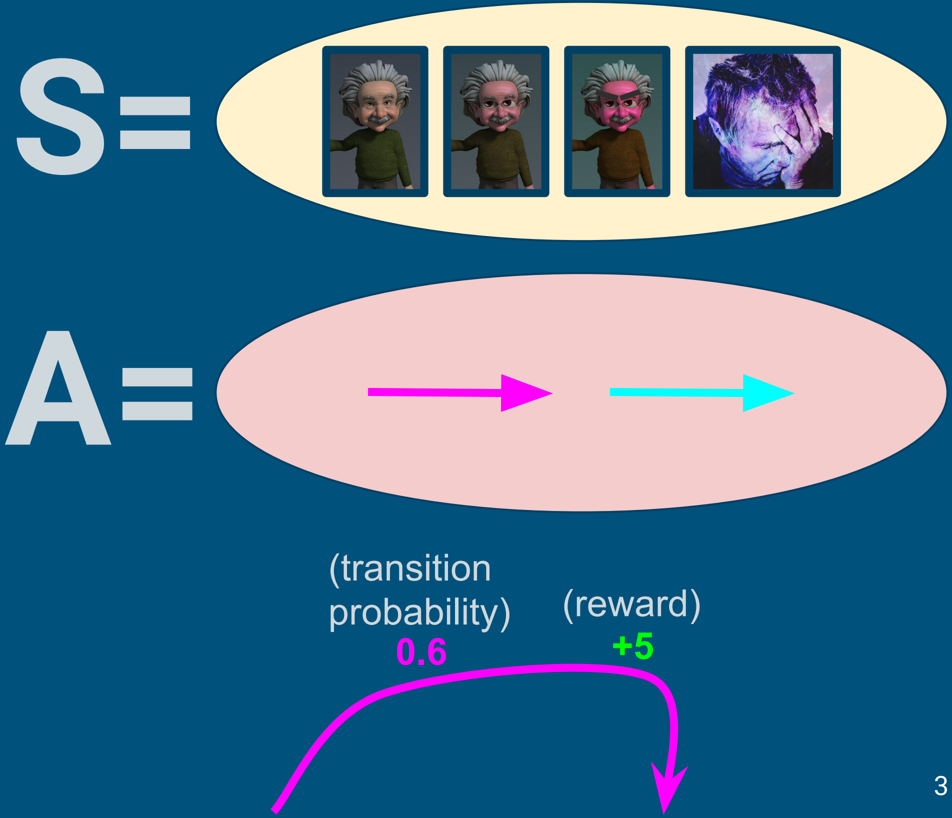
don’t cheat



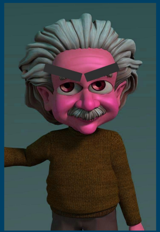
markov decision process

the given problem:

- set of **states** $S = \{s_1,\ldots,s_n\}$
- set of **actions** $A = \{a_1,\ldots,a_m\}$
- **transition** between states via actions (and randomness)
- **rewards** for transitions
- markov property is given



markov decision process ~ cont.

$\pi($  $) = \rightarrow$

<u>what we want:</u>

- maximize rewards (just a value)
- a **policy** π^* that defines the best action for every state
- $\pi: S \rightarrow A$

another example

- 4x4 **states** (visualized as 2d grid)
- 4 **actions** (north, south, east, west)
- chance to take random orthogonal direction (e.g. 10%)
- invalid movement results in transition to same state
- negative reward for moving
- terminal fields (colored) end game on any action, giving noted reward

	0	1	2	3
0				
1		X	-100	+100
2				
3	S	X	-100	-100

some notation

- T(s, a, s’) # transition
  - **probabillity** of getting into state s’ when using action a in state s
  - $T(s_{0,2}, a_E, s_{0,3}) = 0.8$ # go E, as planned
    $T(s_{0,2}, a_E, s_{0,2}) = 0.1$ # go N, bump against wall
    $T(s_{0,2}, a_E, s_{1,2}) = 0.1$ # go S
    $T(s_{0,2}, a_E, s_{0,1}) = 0.0$ # go W
- R(s, a, s’) # reward
  - **reward** for getting into state s’ when using action a in state s
  - $R(s_{1,2}, \_\_, \_\_) = -100$ # lose
    $R(s_{2,0}, a_N, s_{1,0}) = -2$ # moved
    $R(s_{1,3}, \_\_, \_\_) = +100$ # win
- $\gamma \in [0,1]$:= discount factor
  - in timestep t, rewards are worth $\gamma^t \cdot R(s,a,s')$
  - makes sooner rewards worth more
  - $\gamma = 1$: don’t care when rewards are achived
  - $\gamma = 0$: only care about immediate rewards

the V-values

$V^*(s)$ is the estimated reward when starting in s, taking the optimal action and continue to act optimally.

- $V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^*(s') \right]$
- $V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^*(s') \right]$
- $V^*(s) = \max_a \sum_{s'} T(s,a,s')$ [*est. reward when taking action a, landing in state s' and continue acting optimally*]
- $V^*(s) = \max_a \sum_{s'}$ [*est. reward when taking action a, landing in state s' and continue acting optimally, weighted by probabillity*]
- $V^*(s) = \max_a$ [*est. reward when taking action a and continue acting optimally*]
- $V^*(s) =$ [*est. reward when taking best action and continue acting optimally*]

value iteration

$V^k(s)$ is the estimated reward when starting in s, taking the optimal action and continue to act optimally with only **k timesteps left**.

- find V^* via bottom up, iterative approach
- V^1 is known (by problem definition)
- calculate V^{k+1} via information from V^k

$$V^{k+1}(s) = \max_a \Sigma_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^k(s') \right]$$

- for $k \longrightarrow \infty$: $V^k \longrightarrow V^*$

	0	1	2	3
0				
1		X	-100	+100
2				
3	S	X	-100	-100

value iteration ~ example

$V^k(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^{k-1}(s') \right]$

$\gamma = 0.9$

$r = -2$ (moving reward)

```
p(random)       = 0.2
    ⇒T(s,a_N,s_N) = 0.8
    ⇒T(s,a_N,s_W) = 0.1
    ⇒T(s,a_N,s_E) = 0.1
```

	0	1	2	3
0	-2	-2	-2	-2
1	-2	0	-100	+100
2	-2	-2	-2	-2
3	-2	0	-100	-100

value iteration ~ example steps

	0	1	2	3
0	-2	-2	-2	2
1	-2	0	-100	+100
2	-2	-2	-2	-2
3	-2	0	-100	-100

V^1 immediate reward for best option

	0	1	2	3
0	-3.8	-3.8	-3.8	69.6
1	-3.8	0	-100	+100
2	-3.8	-3.8	-21.4	69.6
3	-3.8	0	-100	-100

V^2 probability weighted average over immediate rewards and discounted reward from there for best action

	0	1	2	3
0	-5.4	-5.4	47.5	69.3
1	-5.4	0	-100	+100
2	-5.4	-5.4	30.1	66.1
3	-5.4	0	-100	-100

$$V^3 = \max_a \Sigma_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \cdot V^2(s') \right]$$

live demo

value iteration

policy extraction

what we want:

- maximize rewards
- a **policy** π^* that defines the best action for every state
- $\pi: S \rightarrow A$

- assume we have V^*, how to get π^* ?
- simulate **one timestep for every action**, take best action

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

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot V^*(s') \right]$$

policy extraction ~ example

- value iteration might give approximations for V^* converged to machine ε
- policy extraction then generates the optimal policy π^*
- we now have the **perfect action** for **every state** in a game with **unlimited time**

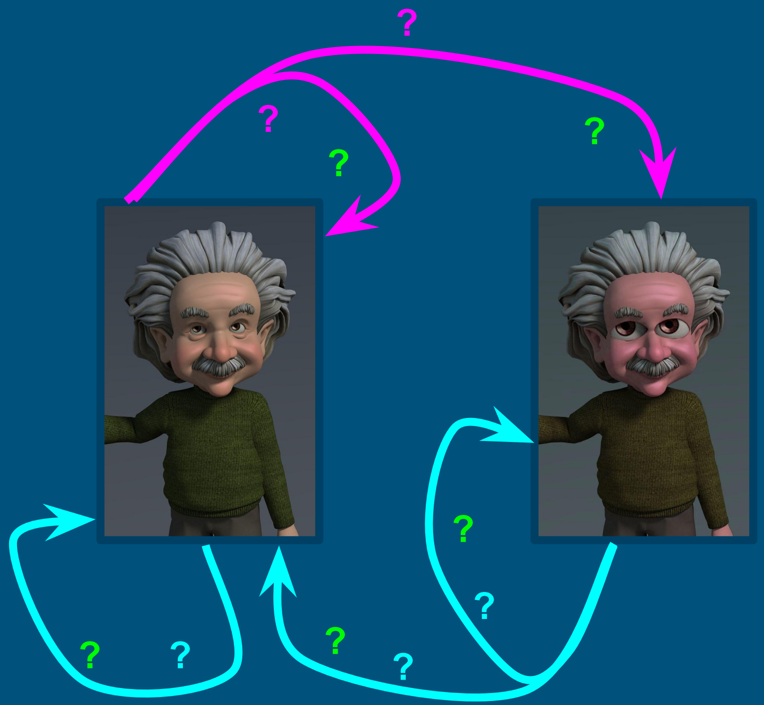
	0	1	2	3
0	49.5	59.1	70.1	82.6
1	41.0	0	-100	+100
2	33.6	28.2	34.9	76.3
3	27.0	0	-100	-100

	0	1	2	3
0	⇒	⇒	⇒	⇓
1	⇑	X	✱	✱
2	⇑	⇒	⇒	⇑
3	⇑	X	✱	✱

live demo

policy extraction

a slightly different problem

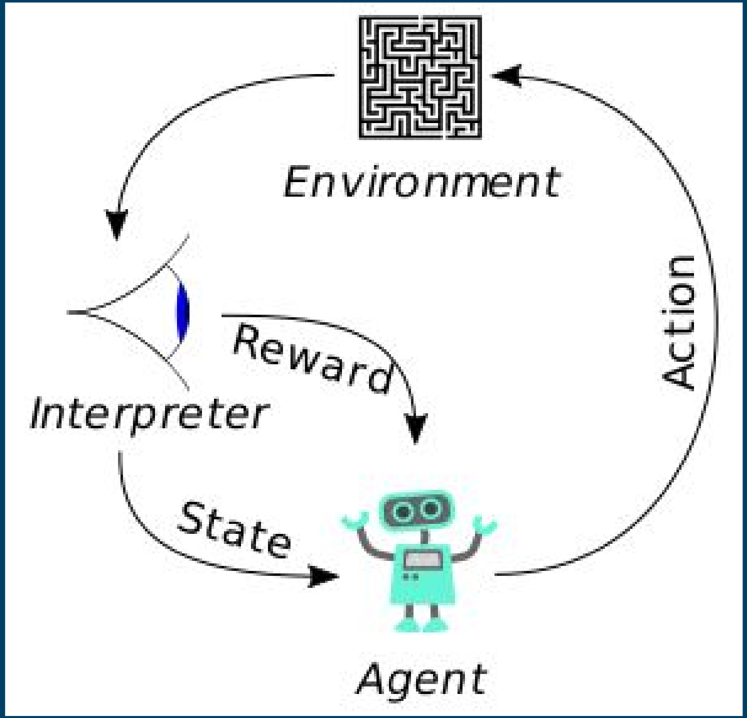


- assume having an MDP
- set of action $A = \{a_1, \ldots, a_n\}$ is known
- current state s is known

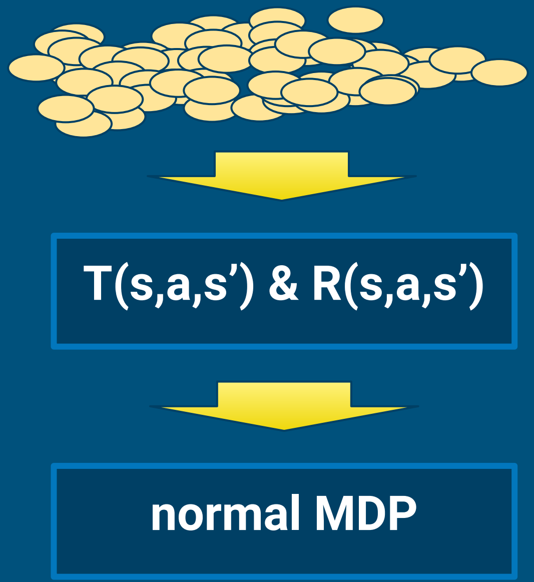
T(s,a,s') and R(s,a,s') are **<u>not</u>** known and must be **determined by trial and error**

⇉ the agent must actively explore the environment

reinforcement learning



model based approach

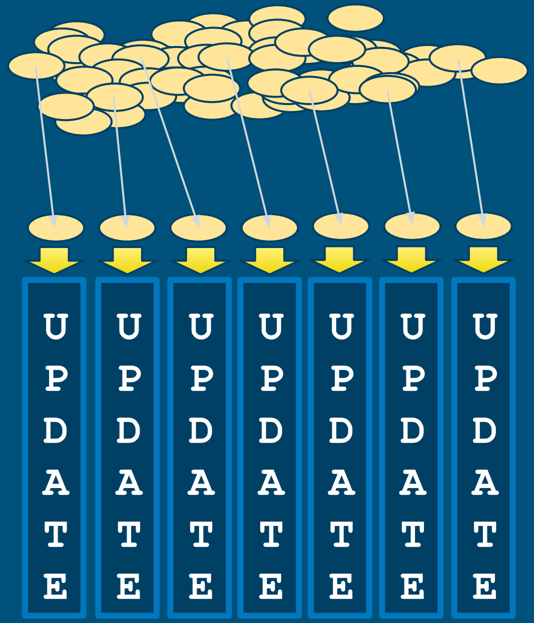


- **approximate** T(s,a,s') and R(s,a,s')
  - by collecting as many samples as possible
- **solve** MDP
  - e.g. via value iteration and policy extraction

- problematic, since the agent often can't move "freely"
- requires **huge amount** of samples

Why not learn V directly, without a model?

temporal difference learning



- initialize V randomly

1. take action based on your policy
2. **update V based on your experience**
   (only for state you came from)
3. **update policy**
4. if terminated, go to start state
5. go to 1.

(2. and 3. can be made batchwise every n actions)

td learning ~ **update V based on your experience**

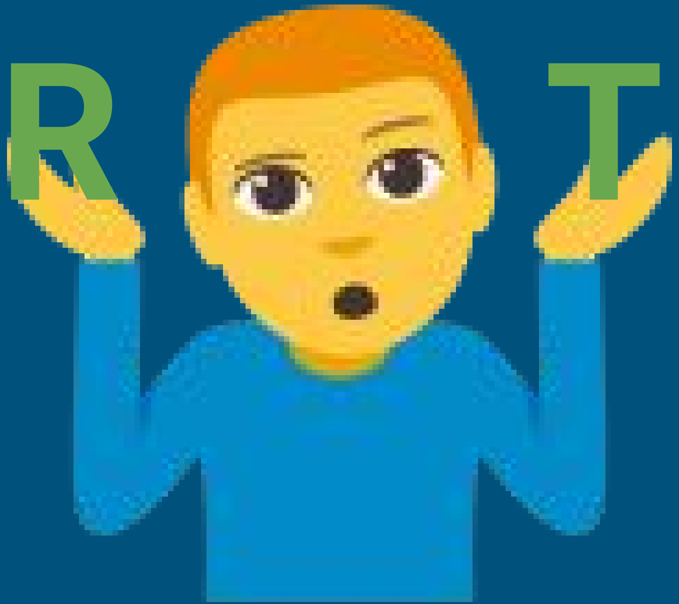
Took action a in state s.
Landed in state s' gaining reward r.

$$V(s) \leftarrow V(s) + \alpha\big(r + \gamma \cdot V(s') - V(s)\big)$$

(α : learning rate)

- adjust V(s) a little into the direction of the **sample**
- let α decay over time
- converges to V^* under certain circumstances

td learning ~ update policy



$$\pi^*(s) = \text{argmax}_a \Sigma_{s'} T(s,a,s') [\, R(s,a,s') + \gamma \cdot V^*(s') \,]$$

- can't use policy extraction, since R and T are not present
- other methods not that trivial

Can we directly learn a policy?

the Q-values

$V^*(s)$ is the estimated reward when starting in s, taking the optimal action and continue to act optimally.

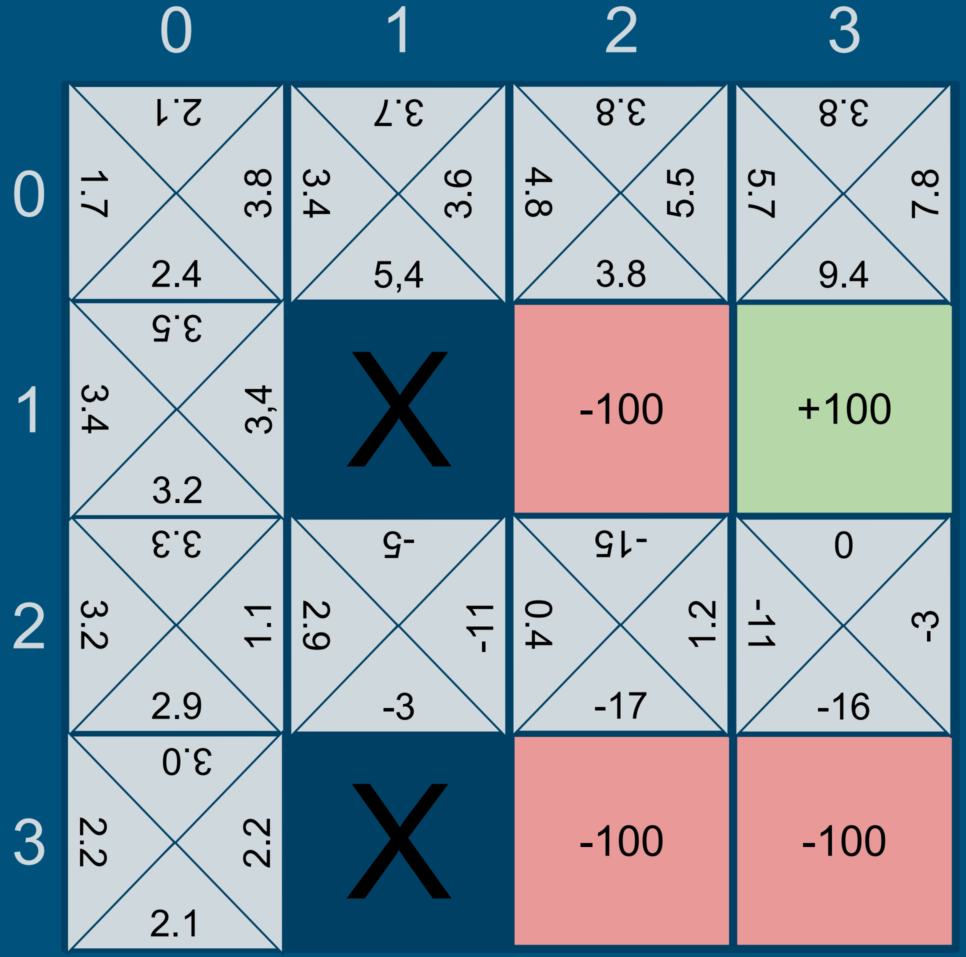
$Q^*(s,a)$ is the **estimated reward** when **starting in s**, taking the **action a** and continue to **act optimally**.

⇒ |A| values per state instead of one

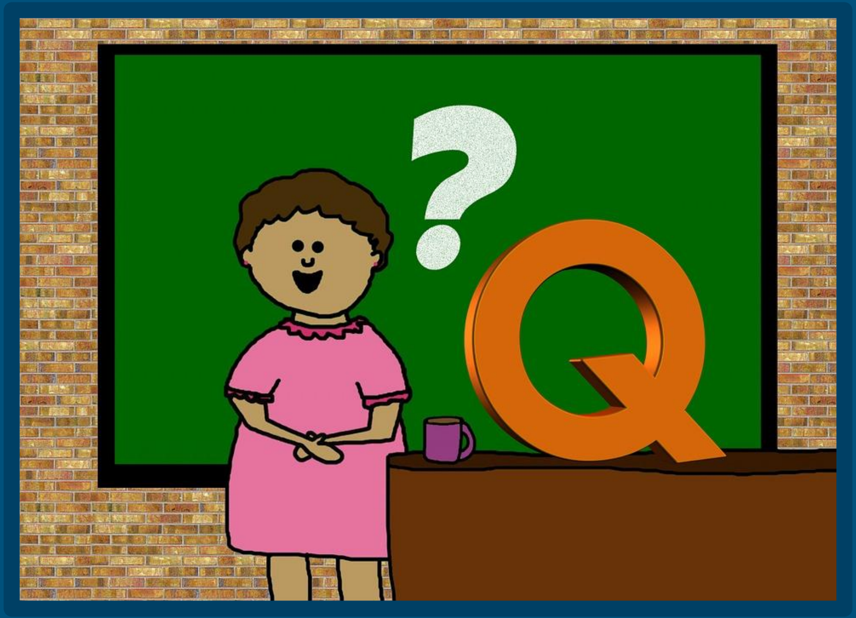
$V(s) = \quad \mathbf{max}_a\, Q(s,a)$

$\pi(s) = \mathbf{argmax}_a\, Q(s,a)$

How to learn those Q-values?



Q-learning



- temporal difference learning on Q-values
- ⊖ needs **more samples** to converge
- ⊕ **policy** is learned **implicitly**
- ⊕ no V values needed

$$V(s) \leftarrow V(s) + \alpha\big(r + \gamma \cdot V(s') - V(s)\big)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \cdot \mathbf{V(s')} - Q(s,a)\big)$$

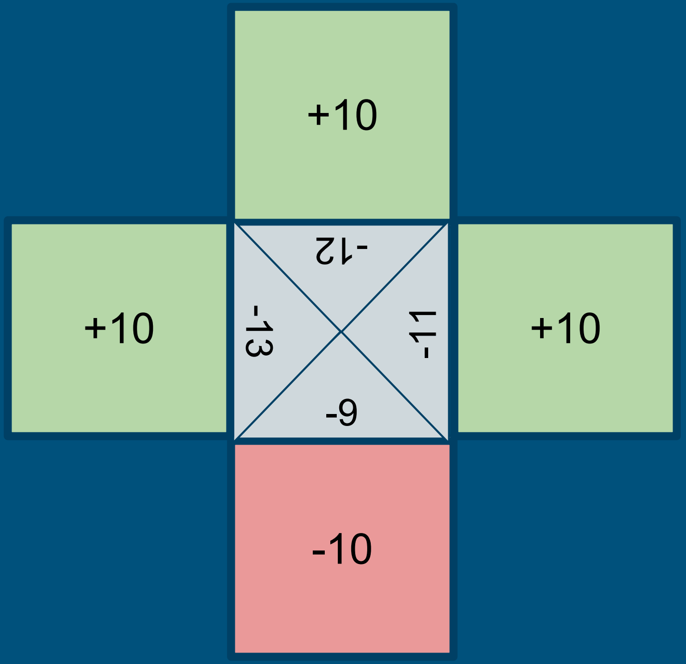
$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \cdot \mathbf{max_{a'} Q(s',a')} - Q(s,a)\big)$$

where to go?

imagine this situation:

- all Q-values are initialized with some random value
- **worst action a** has initially **highest Q-value**
- other actions have initially lower Q-values than the optimal Q-value of **a**

⇉ the agent will never try the other actions
⇉ we need to motivate him doing so



exploration

optimistic initial conditions

- estimate maximum Q-values
- **initialize all Q-values higher**
- updates will decrease Q-values
  (since Q-learning converges)
- agent will prefer other action in later iterations

ε-greedy exploration

- new hyperparameter $0 \leq \varepsilon \leq 1$
  (this can change during training)
- agent will perform a **random action** with a **chance of ε**
- at test time, ε is usually set to 0

exploration ~ cont.

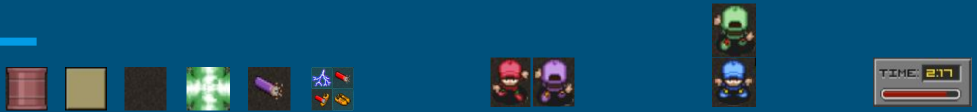
exploration function

- artificially boost Q-values of state-actions that were not used frequently
- choose an exploration function **E(s,a,n)** with $\mathbf{E \longrightarrow 0}$ for $\mathbf{n \longrightarrow \infty}$
  (n: number of state-action uses)
- add this function during Q-value updates

e.g.: $E(s,a,n) = k/n\ ,\ k \in \mathbb{Q}$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\ r + \gamma \cdot \max_{a'}(Q(s',a') + \mathbf{E(s',a',n_{s,a})}) - Q(s,a)\ ]$$

the state space



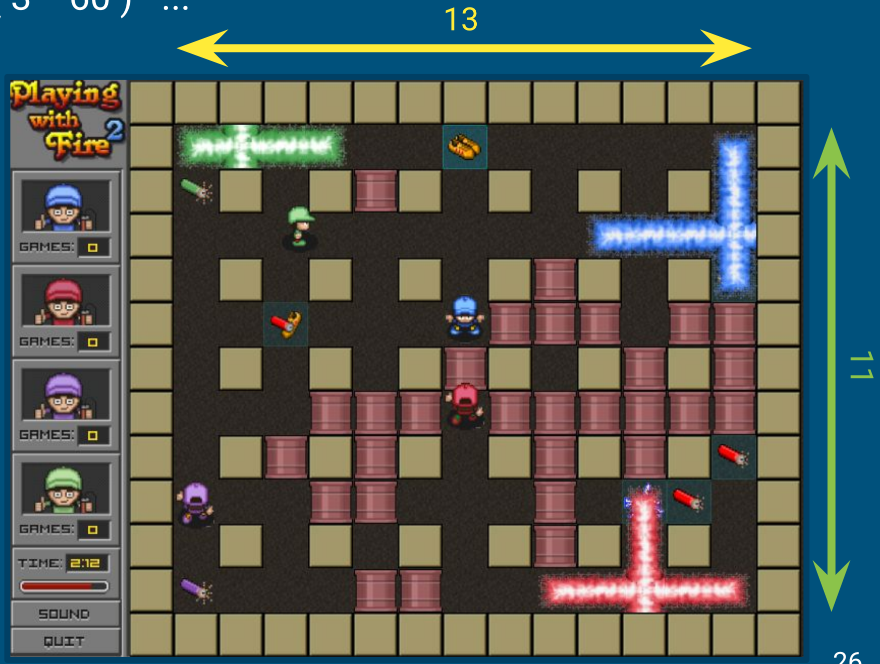
$$|S| = (1 + 1 + 1 + 4 + 4 + 4)^{13 \cdot 11} \cdot (13 \cdot 44)^4 \cdot (11 \cdot 44)^4 \cdot (3 \cdot 60) \cdot \ldots$$

$$|S| = 15^{13 \cdot 11} \cdot 276848^4 \cdot 180$$

$$|S| \approx 1.5 \cdot 10^{168} \cdot 5.9 \cdot 10^{21} \cdot 180$$

$$|S| \approx 1.593 \cdot 10^{192}$$

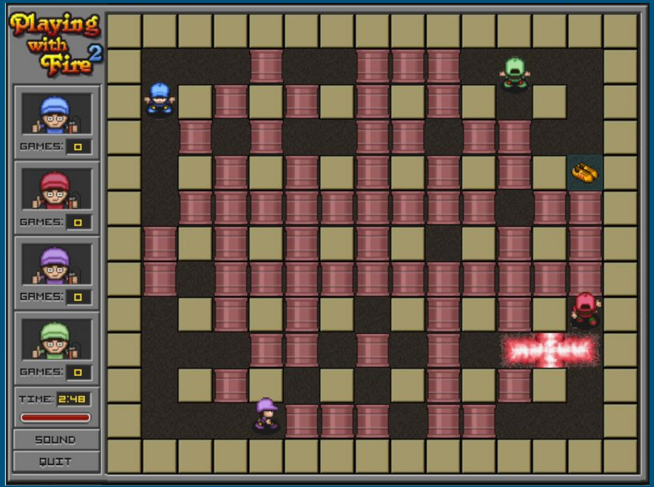
(~ $3.4 \cdot 10^{21}$ · possible constellations in Go)



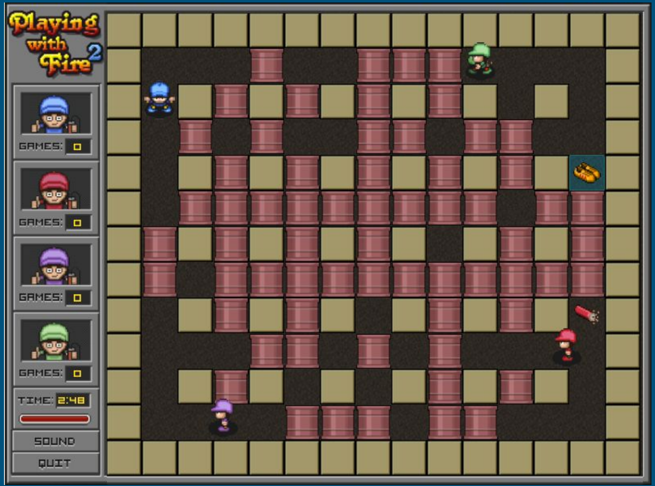
⇉ **enormous** state space in **realistic** scenarios

|S| ≈ 1593000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

the state space and its redundancies



- these states are **completely different** for our agent
- the **optimal action** here is (most likely) **the same**



state features

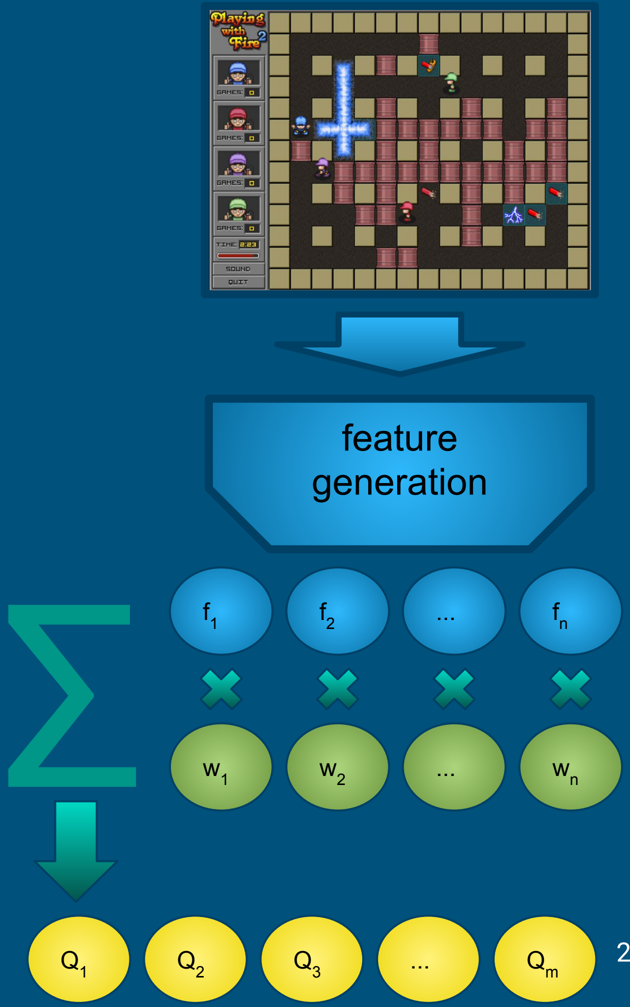
- generate **powerful features** from states
- use features to generate **continuous Q-function** instead of lookup table

$$Q(s,a) = w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + \ldots + w_n \cdot f_n(s,a)$$

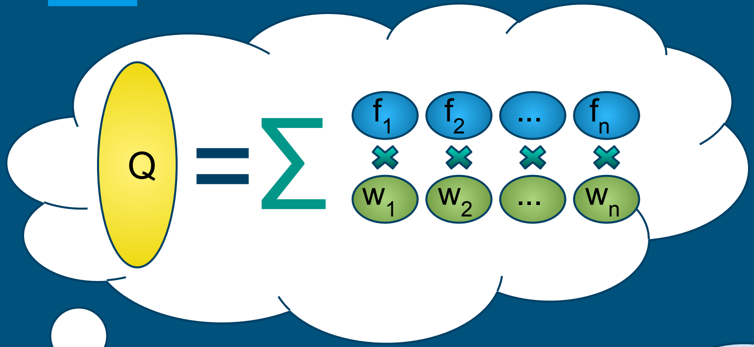
- during training: **tweak weights instead of entries**

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot < \mathit{error} >$$

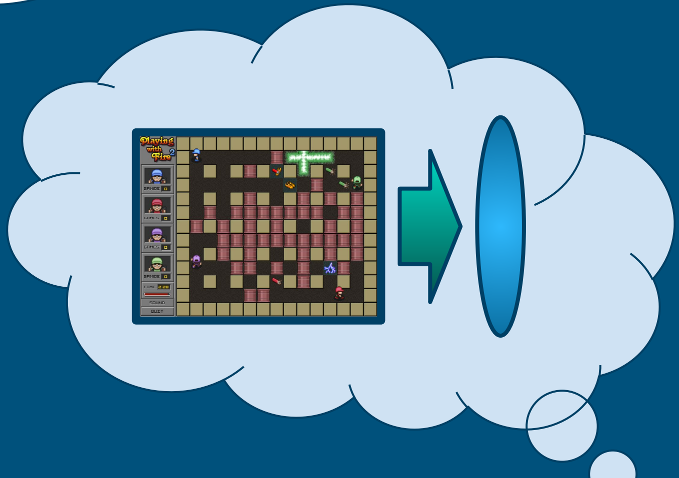
$$w_i \leftarrow w_i + \alpha \cdot < \mathit{error} > \cdot f_i(s,a)$$



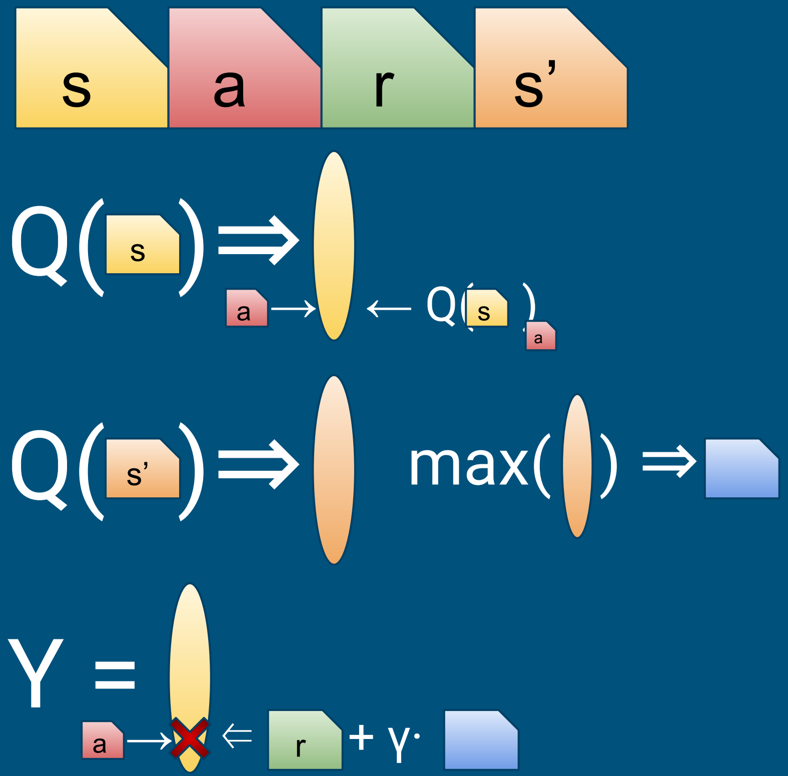
does this sound familiar..?



$$w_i \leftarrow w_i + \alpha \cdot <error> \cdot f_i(s,a)$$



deep Q-learning



- Q-function is approximated by **neural network**
  - input: state
  - output: vector w/ Q-values for all actions
- CNNs allow use of pixel data (game screens, camera) as input
- train with the same samples as in normal Q-learning (s,a,r,s’)
- output “label” for training contains:
  - $r + \gamma \cdot \max_{a'} Q(s')_{a'}$ for a action taken
  - $Q(s)_b$ for all other actions $b \in A$ ($\Rightarrow$ no error)

sources

- AI Course CS188 (University Berkley)
  - http://ai.berkeley.edu/home.html
  - https://www.youtube.com/channel/UCB4_W1V-KfwpTLxH9jG1_iA/videos
- Harmon & Harmon: Reinforcement Learning: A Tutorial
  - http://web.cs.iastate.edu/~honavar/rltut.pdf
- qlearning4k (deep Q-learning framework)
  - https://github.com/farizrahman4u/qlearning4k

The End.

Thank you for your attention!

Any Questions?