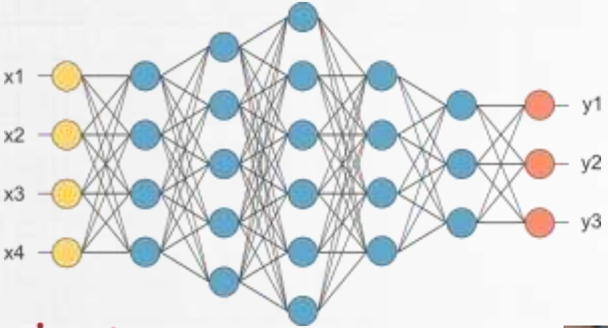
Reinforcement Learning

A short introduction (powered by *Sutton and Barto, 2012*)

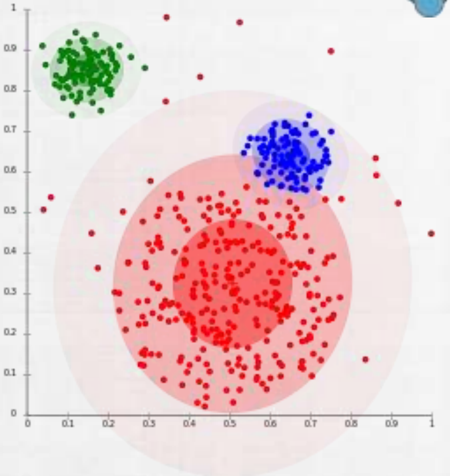
10.01.2019

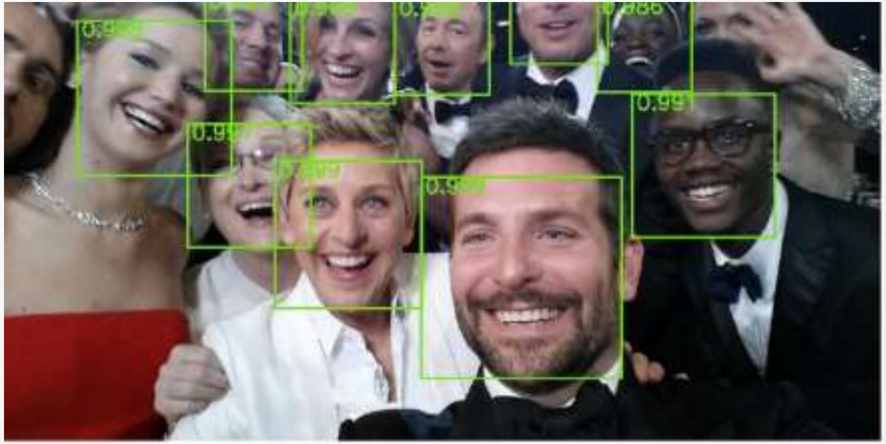
Severin Angerpointner
LMU Munich

AI in different areas



AlphaGo





How do organisms learn?

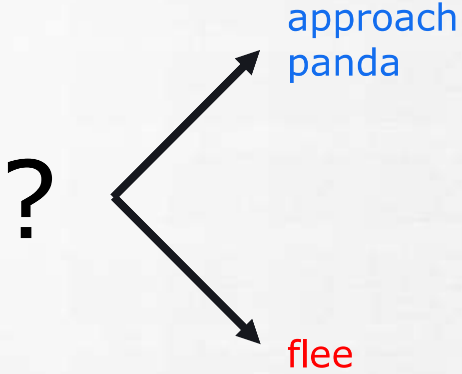
Classification: microscopic model

panda

not a panda

Neural Networks, Clustering, …

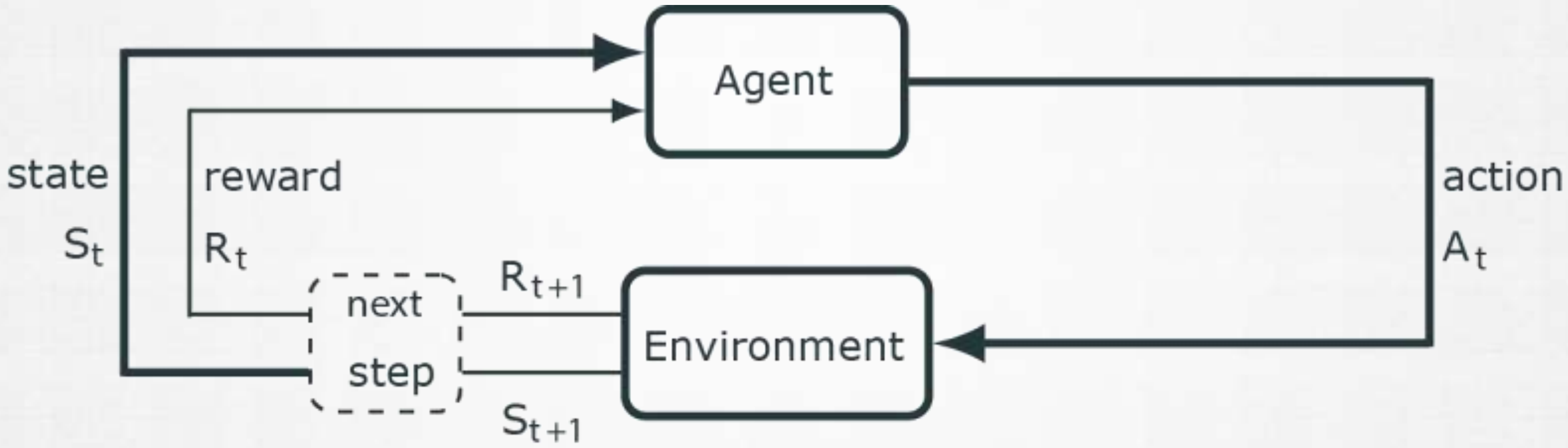
Behaviour (policy): "black box"



Reinforcement Learning, …

Dynamic-Programming
Agent Policy
Watkin's
On-Policy Return
Off-Policy
Environment Action
Bellman-Equation
Greedy Exploitation Exploration
Reinforcement
Value Peng State
Learning
Non-greedy SARSA
Q-learning
Reward
Temporal-Difference

The RL Problem



$t \in \{0, 1, \ldots, T\}$ (episodic tasks: $T < \infty$)

Basic Not(at)ions

- States: $S_t \in \mathcal{S}$
- Actions: $A_t \in \mathcal{A}(S_t)$
- Rewards: $R_t \in \mathbb{R}$ (i.g. randomly distributed)

"input"

- Policies: $\pi : \mathcal{S} \to \mathrm{PDF}[\mathcal{A}]$

  $s \mapsto \pi(a \mid s)$

  (Probability to select action *a* being in state *s*)

"output"

Examples (blackboard)

- Pole Balancing

- Gridworld

Reward vs. Return vs. Value

Learner should achieve an overall *goal* (not just immediate reward)

- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$ $\quad \gamma \in [0, 1]$

- State Value: $v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s]$

State-Action Value

Alternative quantity: $q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

(Expected return after choosing action a in state s and then following policy π)

\+ No search for best action necessary

– Higher computational cost

Bellman Equation

$$q_\pi(s,a) = \sum_{s'} p(s' \mid s,a)\left[r(s,a,s') + \gamma \sum_{a'} \pi(a' \mid s')\, q_\pi(s',a')\right]$$

$$p(s' \mid s,a) = p(S_{t+1} = s' \mid S_t = s, A_t = a)$$

$$r(s,a,s') = \mathbb{E}_\pi\left[R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'\right]$$

Strictly applies only to *Markov Decision Processes (MDP)*!

Optimal Policies

There is at least one "optimal" policy π_*, i.e.:

$$\forall s \in \mathcal{S} \;\; \forall a \in \mathcal{A}(s) : \qquad q_*(s,a) = \max_{\pi} \; q_\pi(s,a)$$

(M.L. Puterman, "Markov Decision Processes", 2016)

→ π_* = "Always choose action with highest value" (greedy)

→ But we don't know corresponding *q*

General RL Algorithms

RL problem solved by finding q_*, with:

$$\pi_*(a_* \mid s) \neq 0 \qquad \text{only for} \qquad a_* = \arg\max_a \; q_*(s,a)$$

Apply Bellman equation:

$$q_*(s,a) = \sum_{s'} p(s' \mid s,a) \left[r(s,a,s') + \gamma \max_{a'} \; q_*(s',a') \right]$$

Unique solution exists for finite MDP!

Common RL Methods

Dynamic Programming

Monte Carlo

Temporal Difference-/ Q-Learning

Q(λ)-Learning

Dynamic Programming

Example: Policy Iteration

1) Use Bellman eq. iteratively to update v for given policy
2) Find better policy by selecting argmax of q as action

\+ Guaranteed convergence (finite MDP)

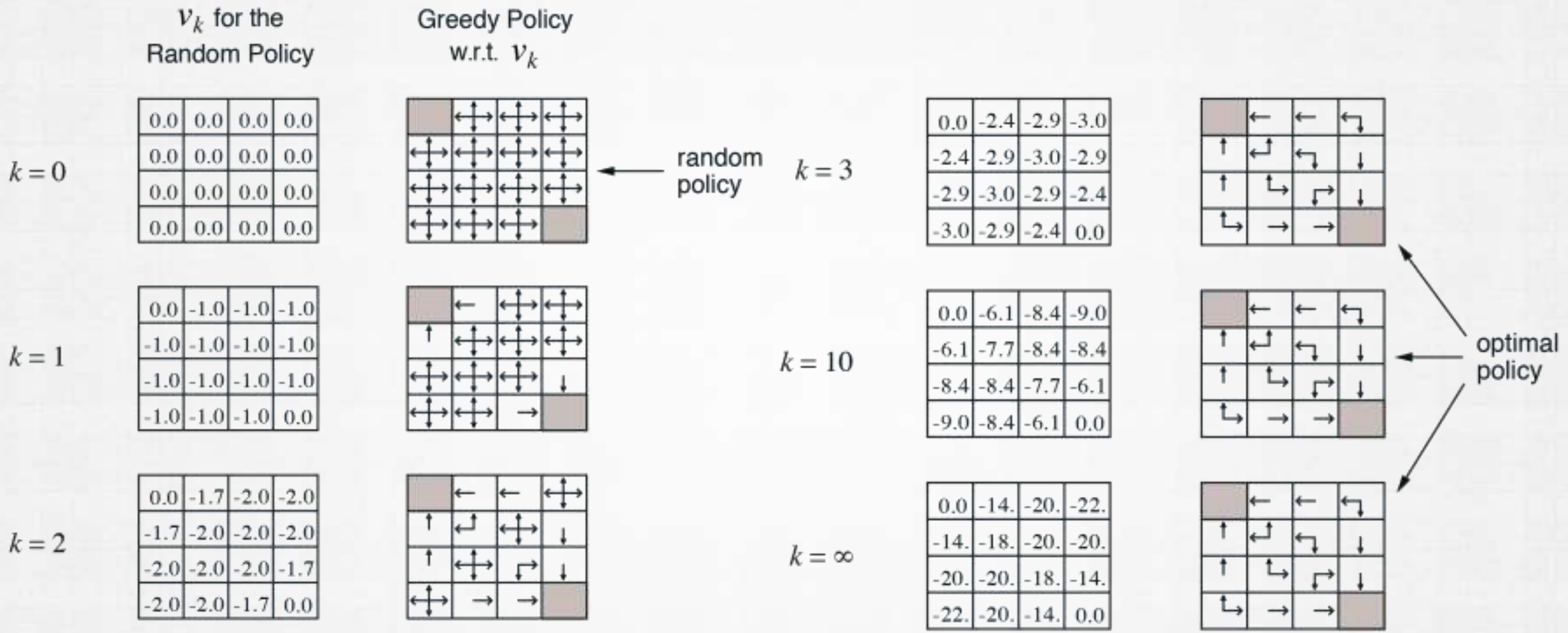
– Computationally expensive
– Complete knowledge of MDP required

Policy Iteration

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad temp \leftarrow v(s)$
   $\quad\quad v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\left[r(s,\pi(s),s') + \gamma v(s')\right]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad temp \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$
   $\quad$ If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2



v_k for the Random Policy
Greedy Policy w.r.t. v_k
$k = 0$
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
random policy
$k = 1$
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
$k = 2$
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0
$k = 3$
0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0
$k = 10$
0.0 -6.1 -8.4 -9.0
-6.1 -7.7 -8.4 -8.4
-8.4 -8.4 -7.7 -6.1
-9.0 -8.4 -6.1 0.0
$k = \infty$
0.0 -14. -20. -22.
-14. -18. -20. -20.
-20. -20. -18. -14.
-22. -20. -14. 0.0
optimal policy

Monte Carlo

Suppose only sample of MDP kown, not full process

1) Approximate value functions empirically $\quad q_\pi(s,a) \leftarrow \frac{1}{N}\sum_{m=1}^{N} G_t^{(m)}$
2) Improve policy similar to DP

+ Requires only sample returns/episodes

– Maintaining exploration
– Can only update after each episode

On-/Off-Policy

- On-Policy: follow *and* evaluate same policy π (as before)

- Off-Policy: follow *behaviour policy* π / evaluate *estimation policy* π'

→ Can choose exploring (*soft*) policy to sample whole state space

Examples: *ε-greedy*, *Softmax*, …

Temporal Difference Learning

General idea (combine DP and MC):

Gradually update q towards optimum

learning rate

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\, \delta_t$$

error/temporal difference
= "target – current value"

e.g. update towards G after full episode (MC): $\delta_t = G_t^{(m)} - Q(S_t, A_t)$

One Step TD Learning

On-Policy: SARSA $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

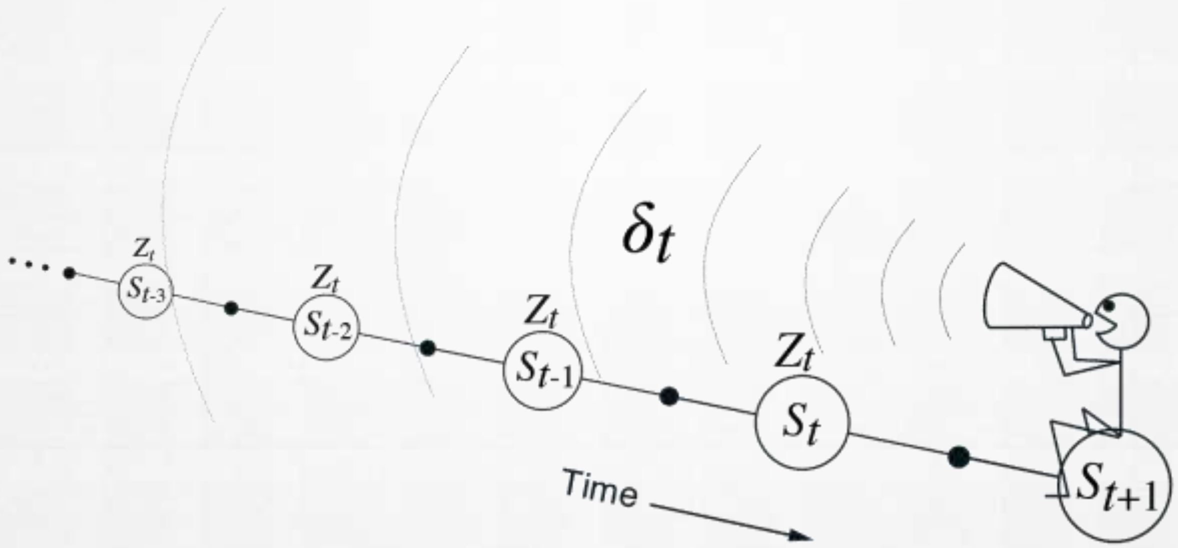
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

Off-Policy: Q-Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Eligibility Traces

Update more than one previously visited states
→ Compromise between full MC and one step TD



TD(λ) Algorithms

New update rule: $\forall s \in \mathcal{S} \quad \forall a \in \mathcal{A}(s)$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\, \delta_t\, Z_t(s,a)$$

With eligibility trace:

$$Z_t(s,a) = \gamma\lambda Z_{t-1}(s,a) + \delta_{S_t,s}\delta_{A_t,a}$$

Sarsa(λ)

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Repeat (for each episode):

    $Z(s, a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

    Initialize S, A

    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)

        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

        $Z(S, A) \leftarrow Z(S, A) + 1$

        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:

            $Q(s, a) \leftarrow Q(s, a) + \alpha \delta Z(s, a)$

            $Z(s, a) \leftarrow \gamma \lambda Z(s, a)$

        $S \leftarrow S'$; $A \leftarrow A'$

    until S is terminal

Summary

1) Model specific tasks with state and action spaces

2) Define goal via reward funcion

3) RL problem: find optimal policy/value function

4) Methods: systematic policy improvement (DP), learning from experience (MC, TD, Q)

*References

- https://qph.fs.quoracdn.net/main-qimg-330e8b2941bc0164211bbdc7d5c693f3
- https://de.wikipedia.org/wiki/Datei:AlphaGo.svg
- https://de.wikipedia.org/wiki/Clusteranalyse#/media/File:EM-Gaussian-data.svg
- https://www.dailydot.com/debug/face-detection-algorithm-image-search/
- http://m.koreatimes.co.kr/pages/article.asp?newsIdx=260722

- Any other graphics are taken from *Sutton and Barto, 2012*