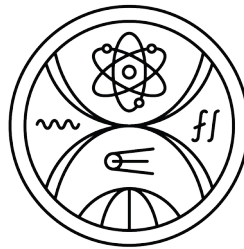


# Introduction to Computational Intelligence

## Basics of reinforcement learning



**Igor Farkaš**

Centre for Cognitive Science  
Faculty of Mathematics, Physics and Informatics  
Comenius University Bratislava

# Types of machine learning

**Supervised learning** – error correction;

Tasks: classification, regression, prediction

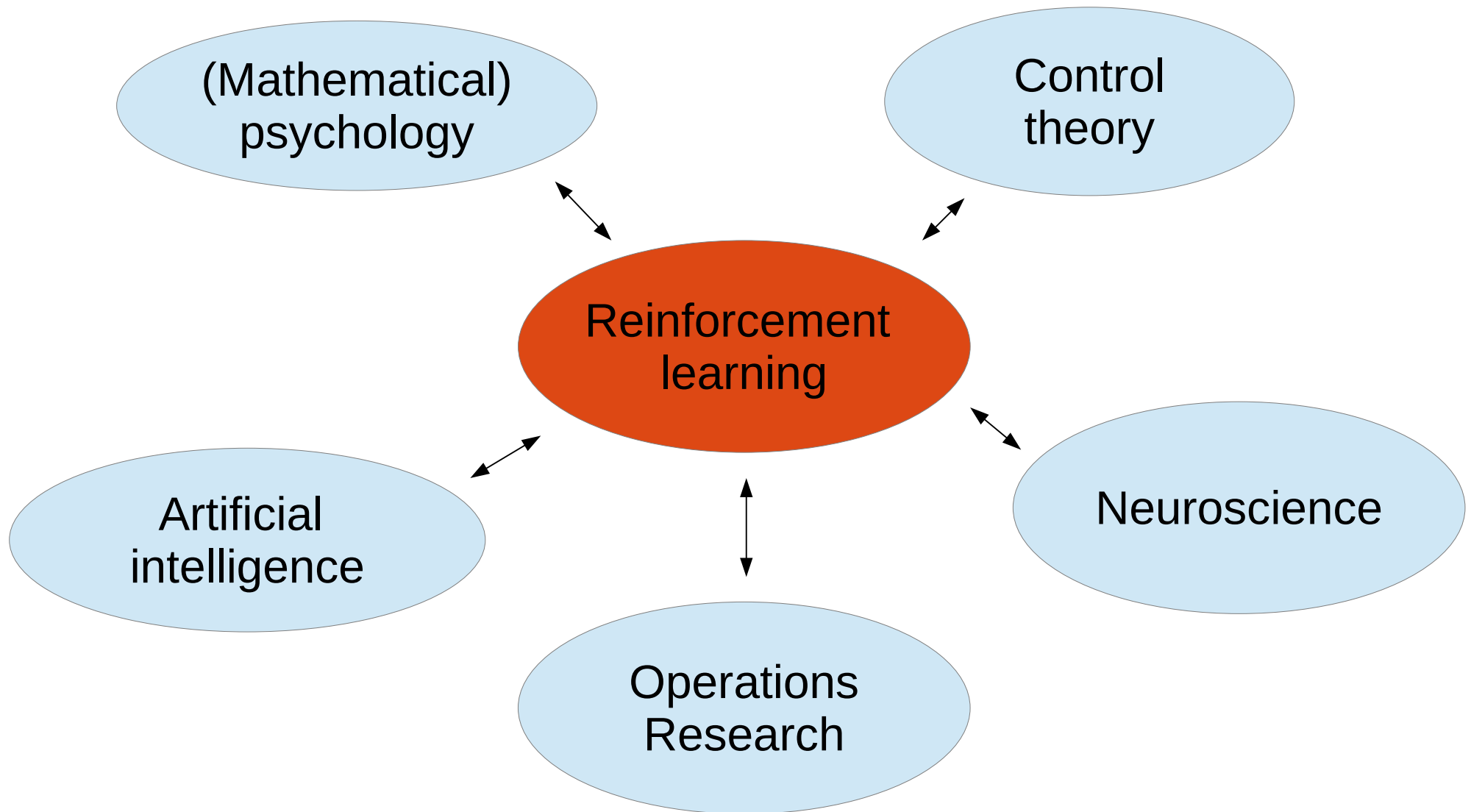
**Unsupervised learning** – statistical correlations

Tasks: dimensionality reduction, density estimation, data visualization

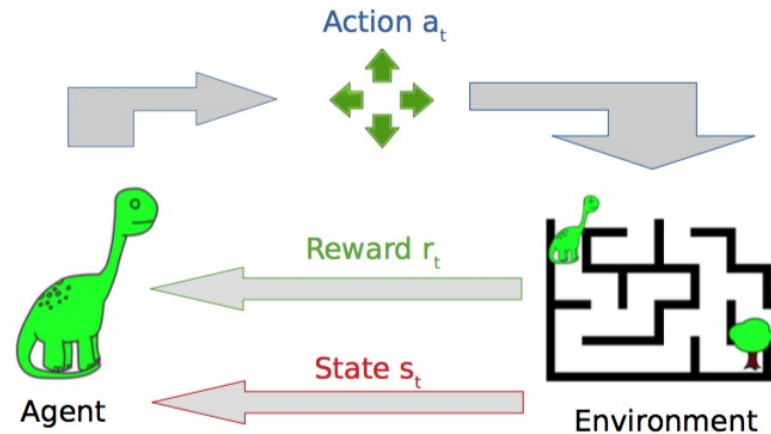
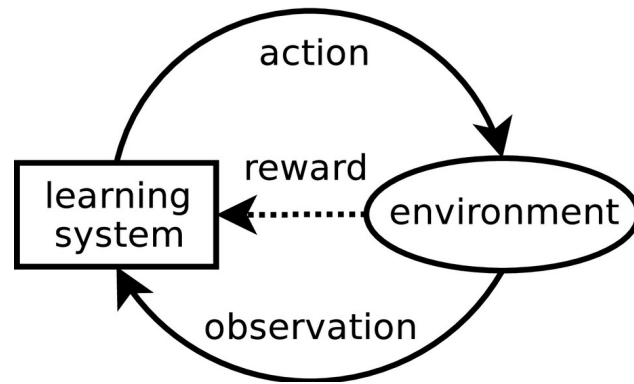
**Reinforcement learning** – maximizing long-term reward

Tasks: sequential decision making via interaction with the environment.

# Links to other research areas



# Reinforcement learning is for



S. Devlin, TWIML Online Meetup, 2018

- Sequential tasks
- with limited (potentially **sparse**) feedback
- when the input data (observations) is not given a priori...
- and the learner (agent) acts on the **environment**
- e.g. learning certain (optimal) behavior, problem solving, game playing, decision making...

# Origins of RL

- **The law of effect** in behavioral conditioning:

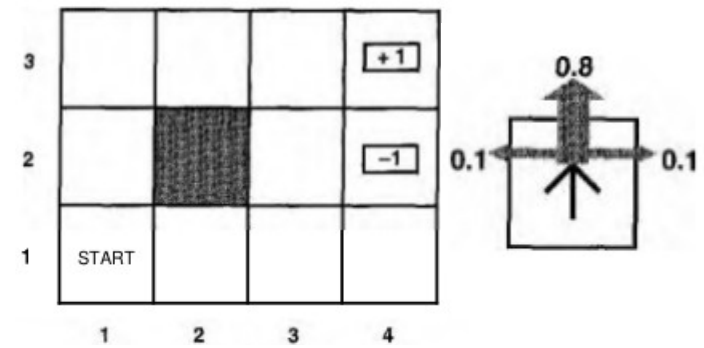
"responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation."  
(Thorndike, 1998)

- **The law of exercise**: "those things most often repeated are best remembered." It is the basis of drill and practice.
- has neuroscientific relevance (dopamine system in the brain – predicted reward)
- **operant conditioning** – learning via rewards and punishments (Skinner, 1938)



# Sequential decision problems

- Agent's utility (function) depends on a sequence of decisions
- Environment assumed to be
  - discrete and finite
  - fully observable and stochastic
- agent can execute (discrete) actions in each state
- **Transition function** – the outcome of each action in each state
- **Reward function** – for agent in each state
- then we have **Markov decision process**
- RL works with MDP assumption
- Solution = **policy**  $\pi$  (actions taken)



Maze: 11 states, 4 actions

# Markov decision process

Sequence :  $s_1, a_1, r_2, s_2, \dots, s_t, a_t, r_{t+1}, s_{t+1}, \dots$

Markov property assumed:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, \dots, s_1, a_1)$$

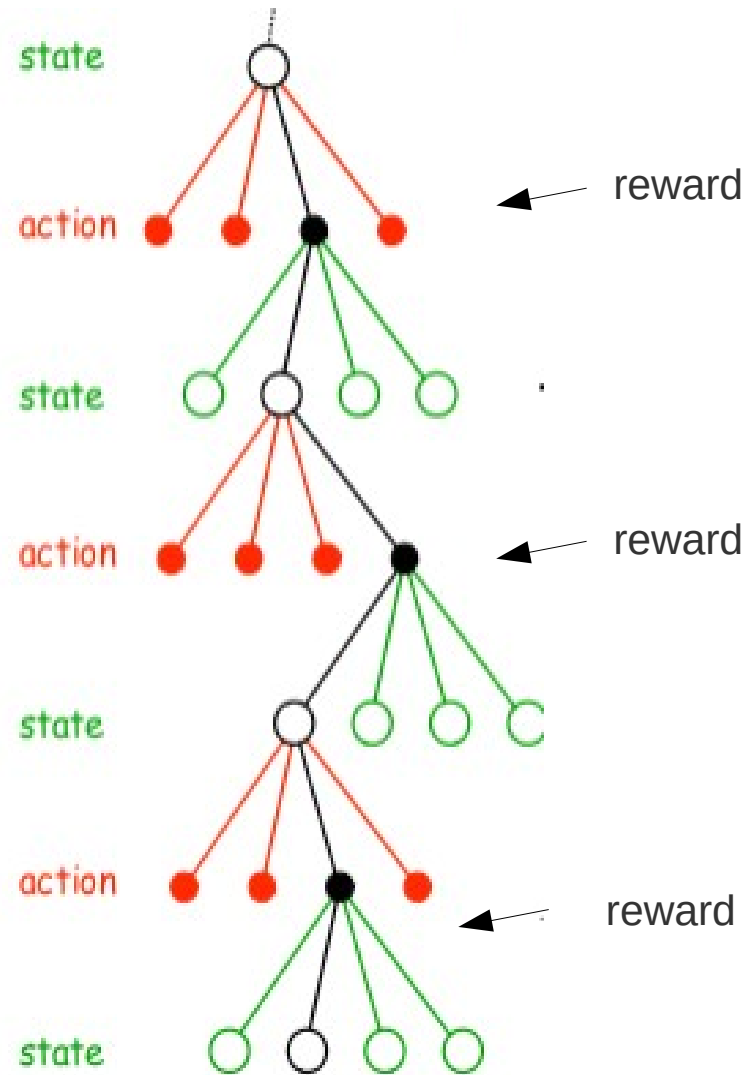
Transition function:

$$P_{ss'}^a = P(s_{t+1} = s' | s_t = s, a_t = a)$$

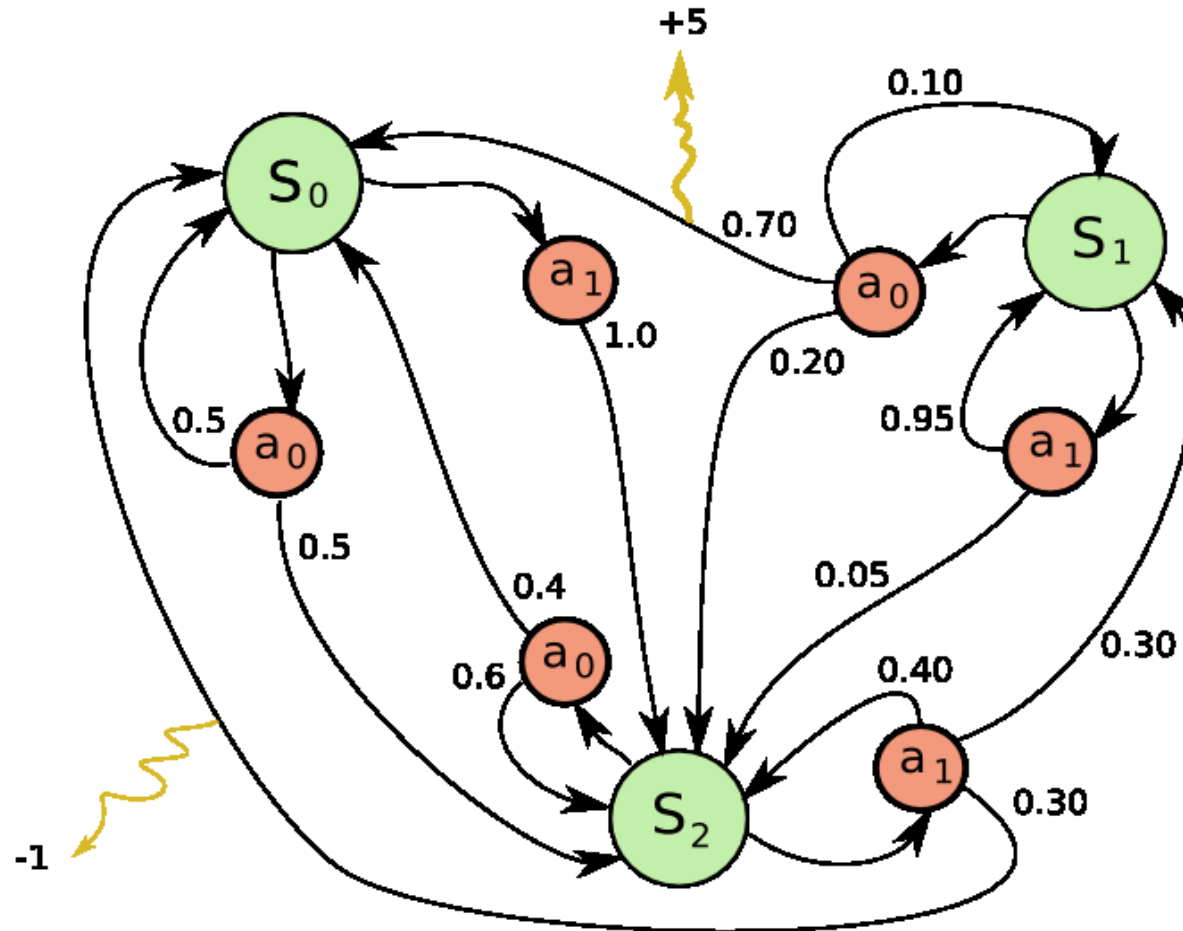
Reward function:

$$R_{ss'}^a = E(r_{t+1} | s_{t+1} = s', s_t = s, a_t = a)$$

- Hence, the world is stochastic



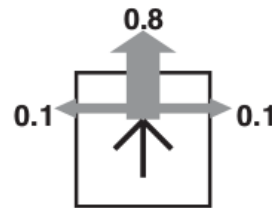
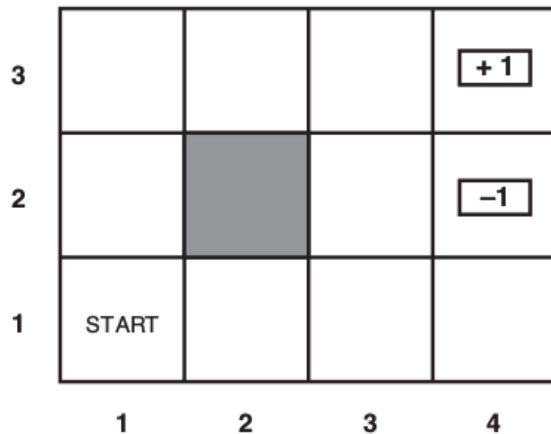
# Example of MDP transition automaton



- states ( $S_i$ ), 2 actions ( $a_j$ ), numbers at links show  $P(s'|a,s)$ , and  $R(s'|a,s)$
- What is the optimal policy?

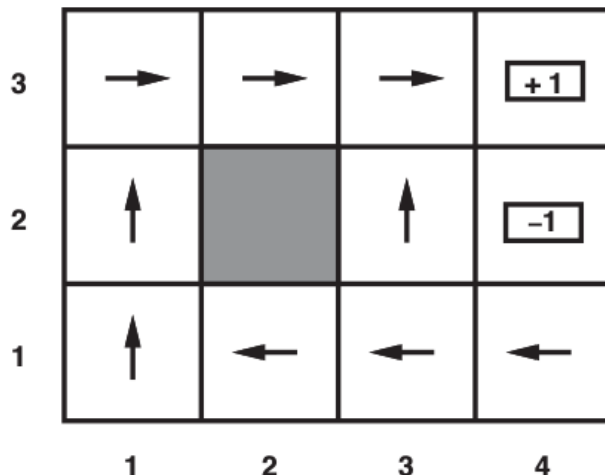


# Maze example

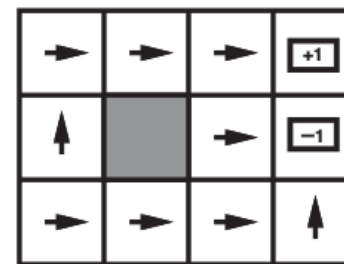


- Stochastic, fully observable environment
- State = agent's position (known)
- Probabilities of actions: 0.8, 0.1, 0.1
- What's the optimal solution (policy)?
- It depends on reward

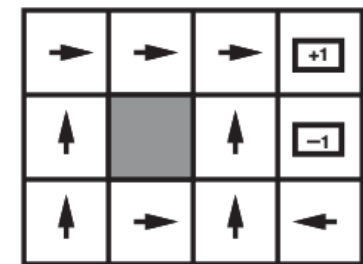
Optimal policies:



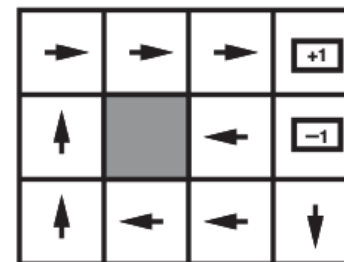
Reward at each transition:  $R(s) = -0.04$



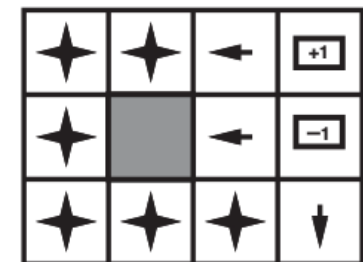
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



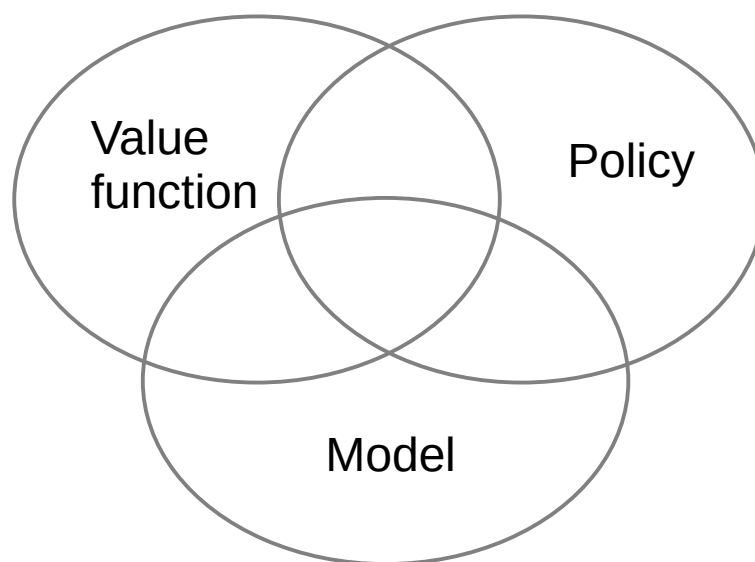
$-0.0221 < R(s) < 0$



$R(s) > 0$

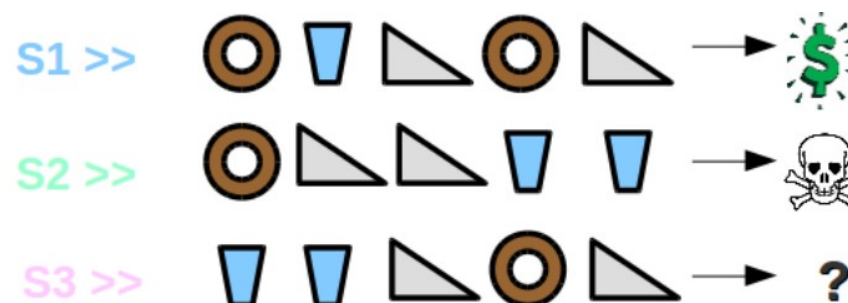
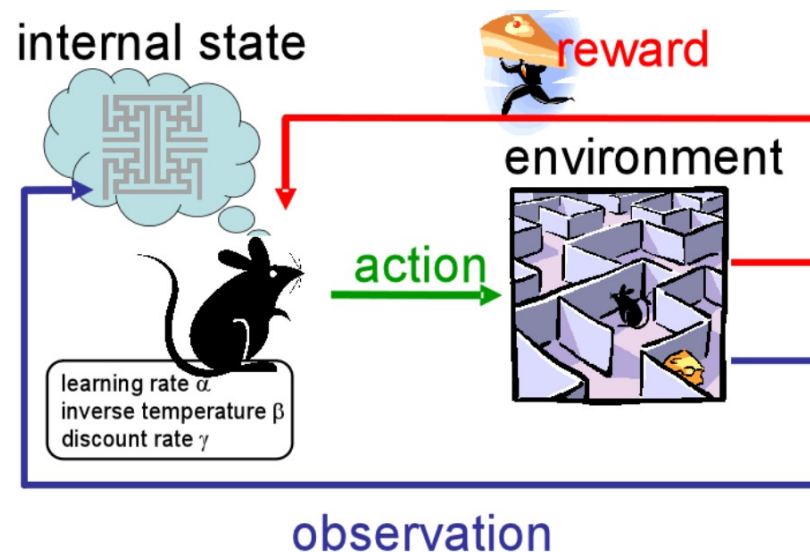
# Components of a RL agent

- An RL agent may include one or more of these components:
  - **Value (utility) function**: how good is each state or action
  - **Policy**: how to behave: state  $\rightarrow$  action(s)
  - **Model**: agent's representation of the environment
- According to its components, we get a taxonomy of RL agents:



# Definition of a state matters

- Environment can be:
  - fully observable
  - partially observable
- Example: In a restaurant, customer orders 5 items (Donuts, Drinks and Sandwiches).
- ... and then responds
- What is to be predicted after sequence S3?
- Depends on a definition of state



# Reward, utility function and policy

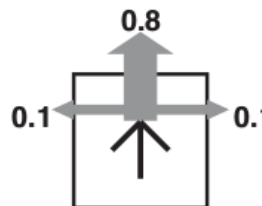
- utility (value) function – allows to choose actions
- Finite or infinite horizon for decision making?
- Additive / discounted rewards:  $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$
- $\gamma \in (0,1)$  – discount factor (future rewards are valued less)
- **Utility function:**  $U^\pi(s) = E_\pi \{ R_t | s_t = s \} = E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \}$
- **Optimal policy:** based on  $U^*(s) = \max_{\pi} U^\pi(s)$
- **Bellman equation:**  $U^\pi(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|a, s) U(s')$   
(value iteration)
- In model-based RL: *Transition & Reward* functions are known

# Value iteration – utilities of states

Calculate the optimal policy with Bellman eq. (iterative process): then use the utilities of states to select an optimal action in each state.

3	0.812	0.868	0.918	<div>+1</div>
2	0.762		0.660	<div>-1</div>
1	0.705	0.655	0.611	0.388
	1	2	3	4

Calculated for  $R(s) = -0.04$  and  $\gamma = 1$



$$U(1,1) = -0.04 + \max \left\{ \begin{array}{l} 0.8 \cdot U(1,2) + 0.1 \cdot U(2,1) + 0.1 \cdot U(1,1), \\ 0.9 \cdot U(1,1) + 0.1 \cdot U(1,2), \\ 0.9 \cdot U(1,1) + 0.1 \cdot U(2,1), \\ 0.8 \cdot U(2,1) + 0.1 \cdot U(1,2) + 0.1 \cdot U(1,1) \end{array} \right\}$$

Up  
Left  
Down  
Right

$$U^\pi(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|a, s) U(s')$$

# Passive RL

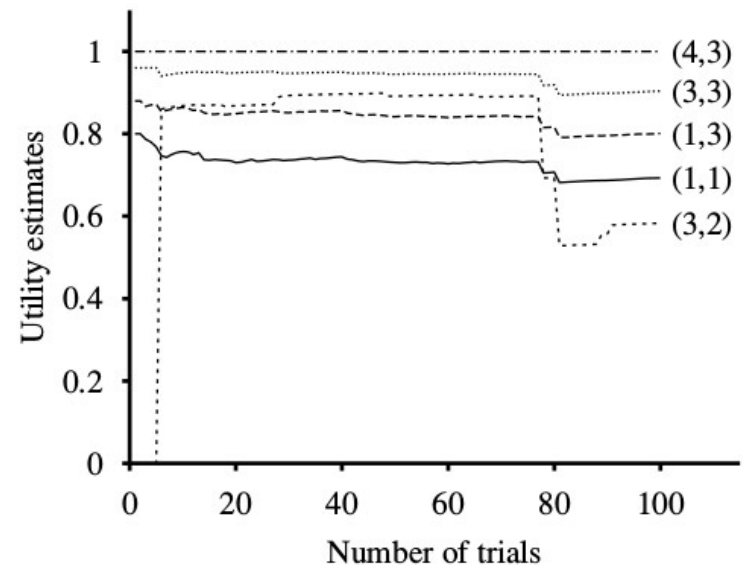
- Agent's policy is **fixed** (in state  $s$ , action  $\pi(s)$  is performed)
- Goal: to learn how good the policy is (i.e. learn  $U(s)$ )
- passive RL agent does not know  $T(s,a,s')$ , nor reward-to-go  $R(s)$  for each state (i.e. average reward accumulated from that state)
- Ways to learn the utility function:
  - **Direct utility estimation** (reward-to-go known)
  - **Adaptive dynamic programming** (learns the transition model in fully observable environment)
  - **Temporal difference learning** (model-free, approximation to ADP)

# Direct utility estimation

- Estimated reward-to-go for each state
- Idea: calculate average reward over all trials (episodes) for each state independently, use as teaching signal
- Ignores connections between successive states (used in Bellman eq.)
- Hence, it converges very slowly

# Adaptive dynamic programming

- Agent takes advantage of the constraints (resulting from links between states)
- Learns the transition model  $T(s,a,s')$  - by estimating  $P(s'|a,s)$ 
  - easy because environment is fully observable
  - a supervised learning task
- $P(.)$  are plugged (together with obtained reward) into Bellman eq.
- But: intractable for large spaces





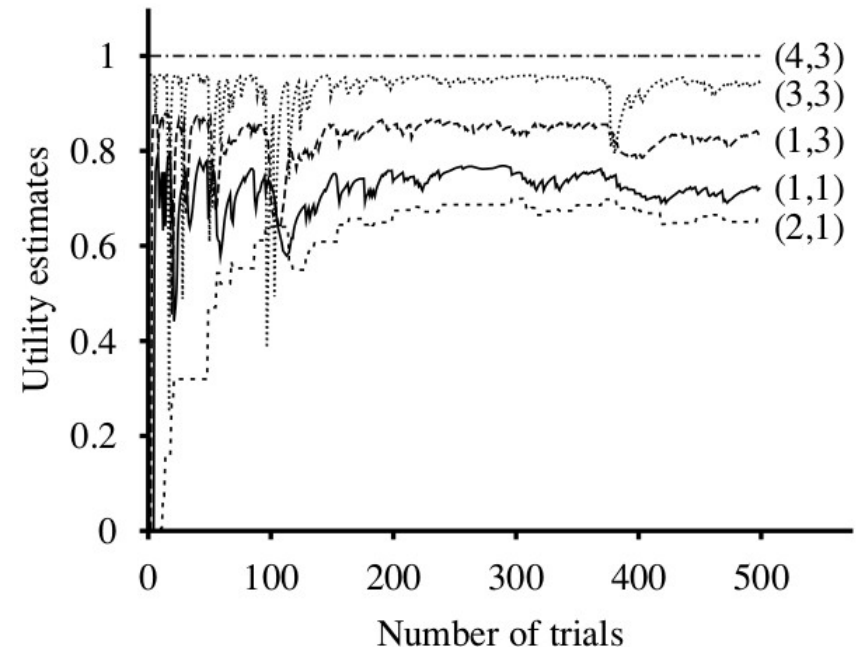
# Temporal-difference learning

- Successive state considered, world model not learned
- Temporal difference error:  $\delta_t = r_{t+1} + \gamma U_t(s_{t+1}) - U_t(s_t)$
- **TD update rule:**  $U_{t+1}(s_t) = U_t(s_t) + \alpha \delta_t$ ,  $\alpha$  - learning rate
- prediction task
- analogy to error BP in supervised learning

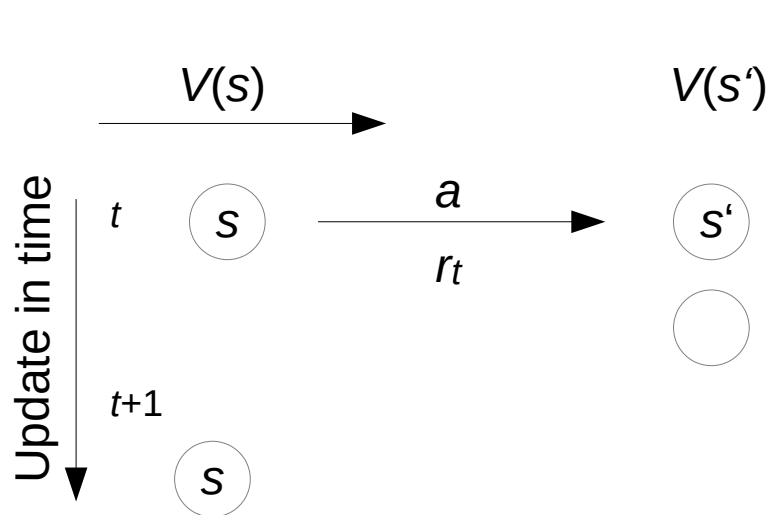
$$U_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$U_t = r_t + \gamma (r_{t+1} + \gamma^1 r_{t+2} + \dots)$$

$$U_t = r_t + \gamma U_{t+1}$$



# TD learning in more detail



$$V_{t+1}(s_t) = V_t(s_t) + \alpha \underbrace{(r_t + \gamma V_t(s_{t+1}) - V_t(s_t))}_{\text{moving target}}$$

Value estimate for current state  $V(s)$  is moved towards its discounted estimate related to the next state  $V(s')$ .

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ R_t &= r_t + \gamma (r_{t+1} + \gamma r_{t+2} + \dots) \\ R_t &= r_t + \gamma R_{t+1} \end{aligned}$$

$$U^\pi(s) = E_\pi \{ R_t | s_t = s \} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\}$$

$$U^\pi(s') = E_\pi \{ R_t | s_{t+1} = s' \} = E_\pi \left\{ \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_{t+1} = s' \right\}$$

$$U^\pi(s) = r_t + \gamma E_\pi \left\{ \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_{t+1} = s' \right\}$$

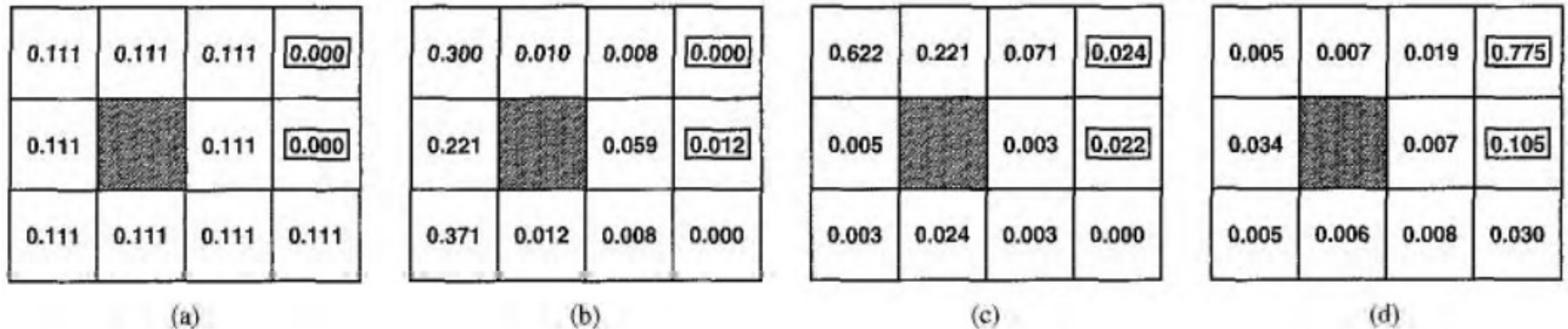
$$U^\pi(s) = r_t + \gamma U^\pi(s')$$

$U^\pi(s)$  - predicts  $R_t(s)$

$E(.)$  - expected value

# Partially observable MDPs

- Agent does not know true state (due to limited sensors)
- Optimal action depends on agent's current **belief state**
- Agent also has a sensor (evidence) model:  $P(e|s)$
- Belief state = probability distribution over the states



**Figure 17.8** (a) The initial probability distribution for the agent's location. (b) After moving *Left* five times. (c) After moving *Up* five times. (d) After moving *Right* five times.

# Review of important concepts in RL

- (numerical) reward (includes also punishment)
- RL = temporal credit assignment problem (learning from delayed reward)
- **Policy** – strategy for choosing actions in various states
- **Exploration** vs **exploitation**
- Model-based (we know transition function and reward function) vs model-free approaches
- Two related RL problems:
  - **Prediction (passive)**: learn value function for given policy
  - **Control (active)**: learn optimal policy (includes prediction)

# Active RL – control task

- **exploration** enabled (non-greedy behavior), model-free
  - random factor – no guarantee
- Control task – choosing best actions  $U(s) = \max_a Q(s, a)$
- Learning state-action  $Q$ -values:
- Optimal policy:  $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$
- **Q-learning** update (off-policy): converges faster

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

- **SARSA** update: (on-policy)  $V_{t+1}(s_t) = V_t(s_t) + \alpha (r_t + \gamma V_t(s_{t+1}) - V_t(s_t))$

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha [r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

# Actor–critic architectures

- Agent is split into two components
- **Critic** – learns the state values  $V(s)$
- **Actor** – learns the policy  $\pi(s)$
- Critic should not learn too quickly, nor too slowly
- A-C approach has advantages in high-dim. spaces and continuous actions (over Q-learning and SARSA)
- biological relevance

# Generalization in RL

- MDP in continuous state space (and action space)
- Use **function approximation**
- Learning = setting parameters of this function
- Relationship to supervised learning (gradient-based methods)
- Very useful in various continuous, high-dimensional, partially observable environments (e.g. robotics).
- E.g. using actor-critic architecture (both the policy and value function are stored)

# Function approximation

- **Linear** – combination of features
  - Simple to implement, faster to compute, convergence
  - Common method to find features: discretization of state space; fuzzy sets can be used (designed by hand)
- **Non-linear** – more general and complex
  - Less convergence guarantees
- **Updating parameters** of the function

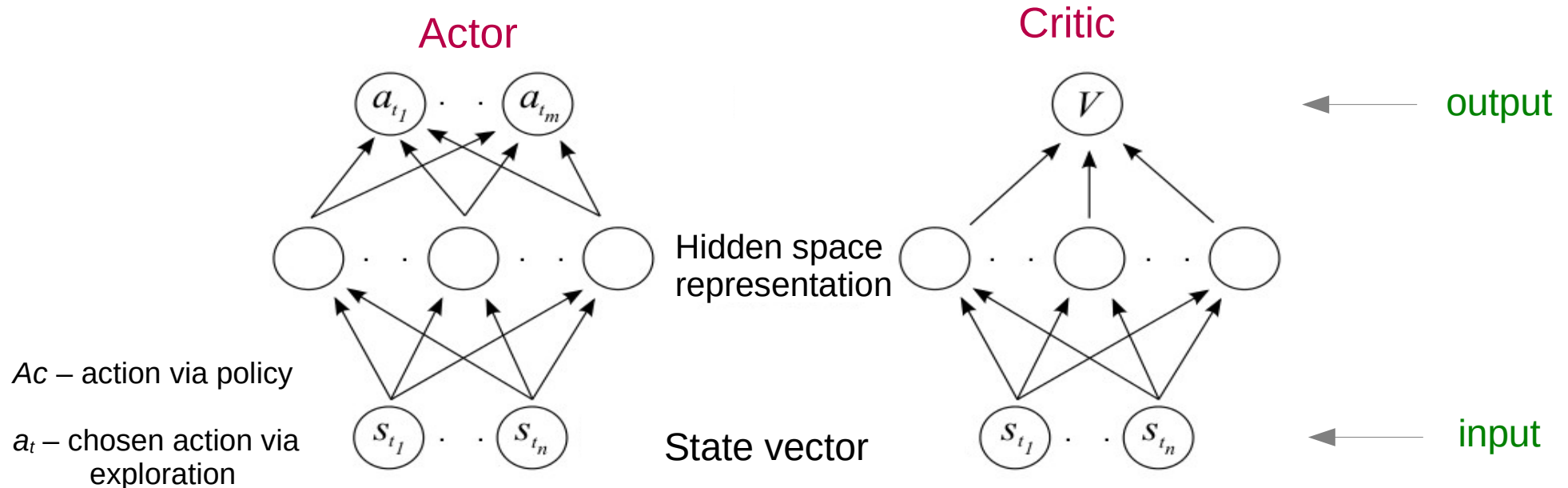


# Updating parameters

- In some cases, closed-form solution can be found
- Otherwise, iterative methods must be used (Bayesian methods possible for stationary functions)
- **Gradient descent methods**
- **Gradient-free optimization** – useful when the function to be optimized is not differentiable
  - evolutionary strategies can be used

# Continuous A-C Learning Automaton

(van Hasselt & Wiering, 2007)



```

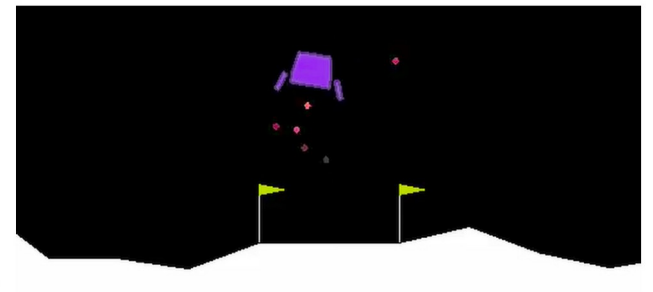
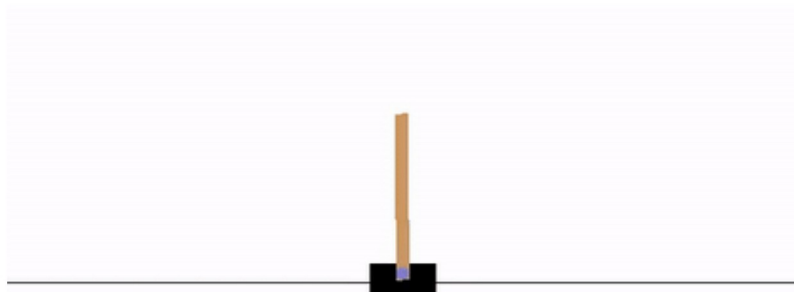
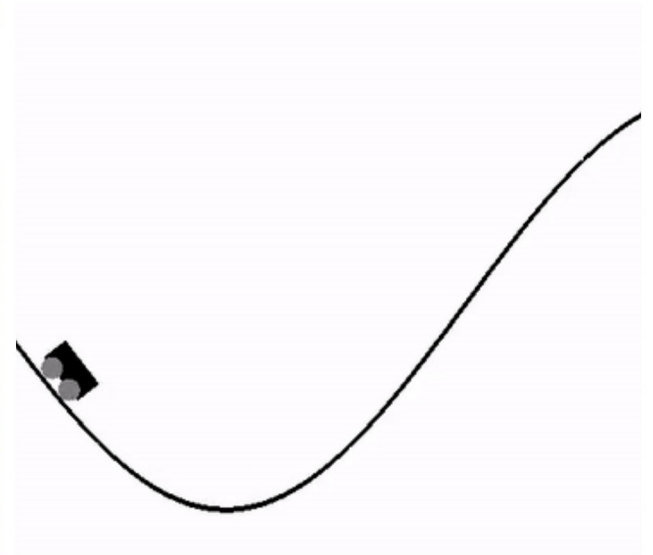
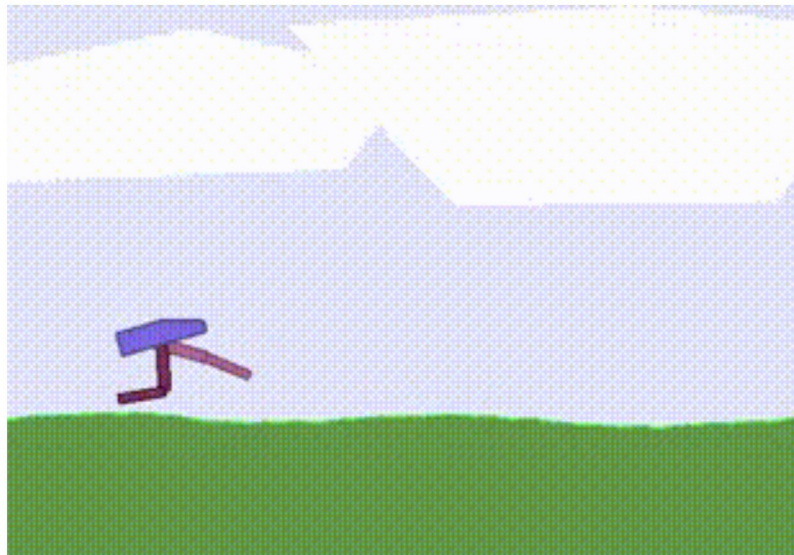
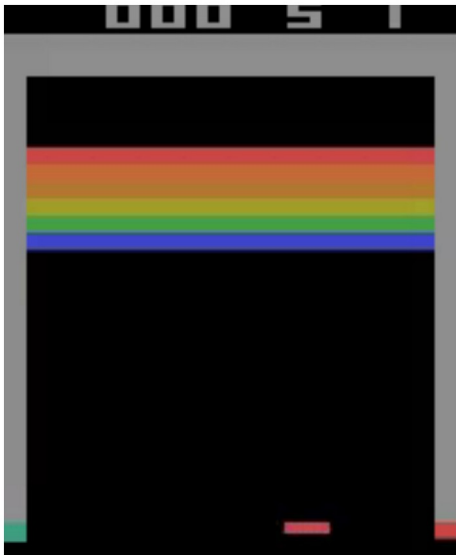
for  $t=0,1,2\dots$  do
  choose action  $a_t \leftarrow Ac_t(s_t)$ , using exploration
  make action  $a_t$ , get to next state  $s_{t+1}$ 
  update critic:  $V_{t+1} \leftarrow r_{t+1} + \gamma V_t(s_{t+1})$ 
  if  $V_{t+1}(s_t) > V_t(s_t)$ , then
    update actor's parameters such that:  $Ac_{t+1} \leftarrow a_t$ 
  end if
end for
    
```

$$\theta_{i,t+1}^V = \theta_{i,t}^V + \alpha \delta_t \frac{\partial V_t(s_t)}{\partial \theta_{i,t}^V}$$

$$\theta_{i,t+1}^{Ac} = \theta_{i,t}^{Ac} + \alpha (a_t - Ac_t(s_t)) \frac{\partial Ac_t(s_t)}{\partial \theta_{i,t}^{Ac}}$$

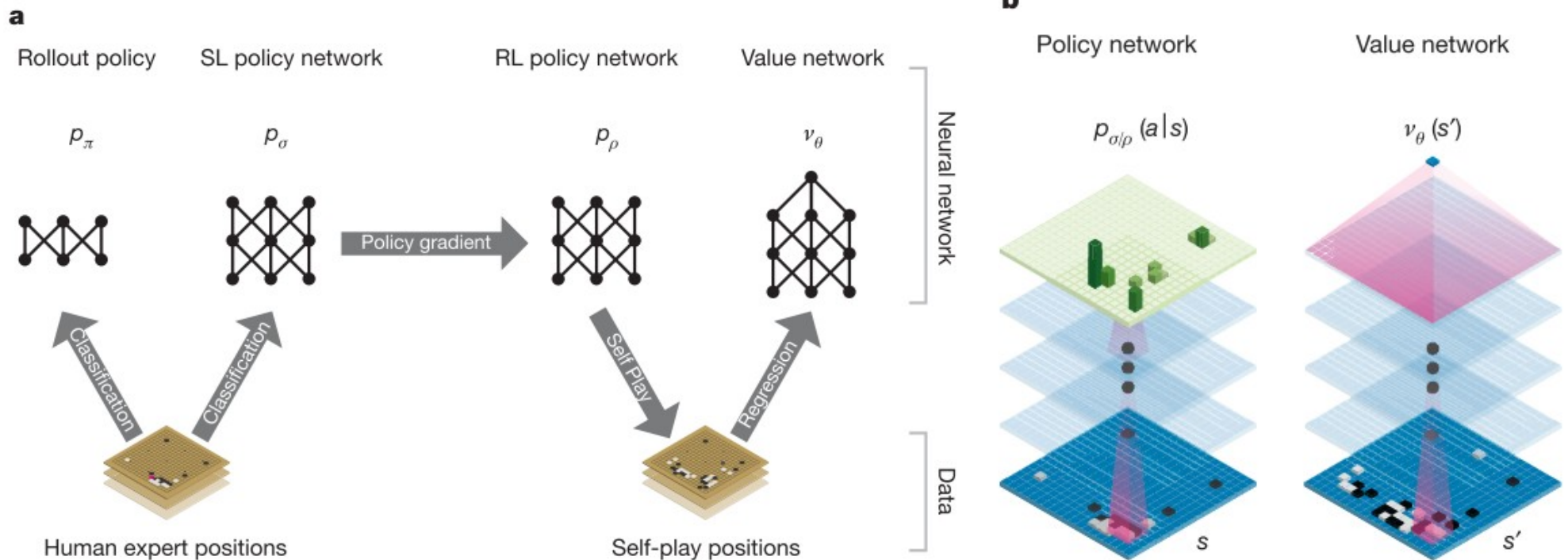
# Open AI gym

Excellent toolkit for running RL experiments in python, in plenty of environments



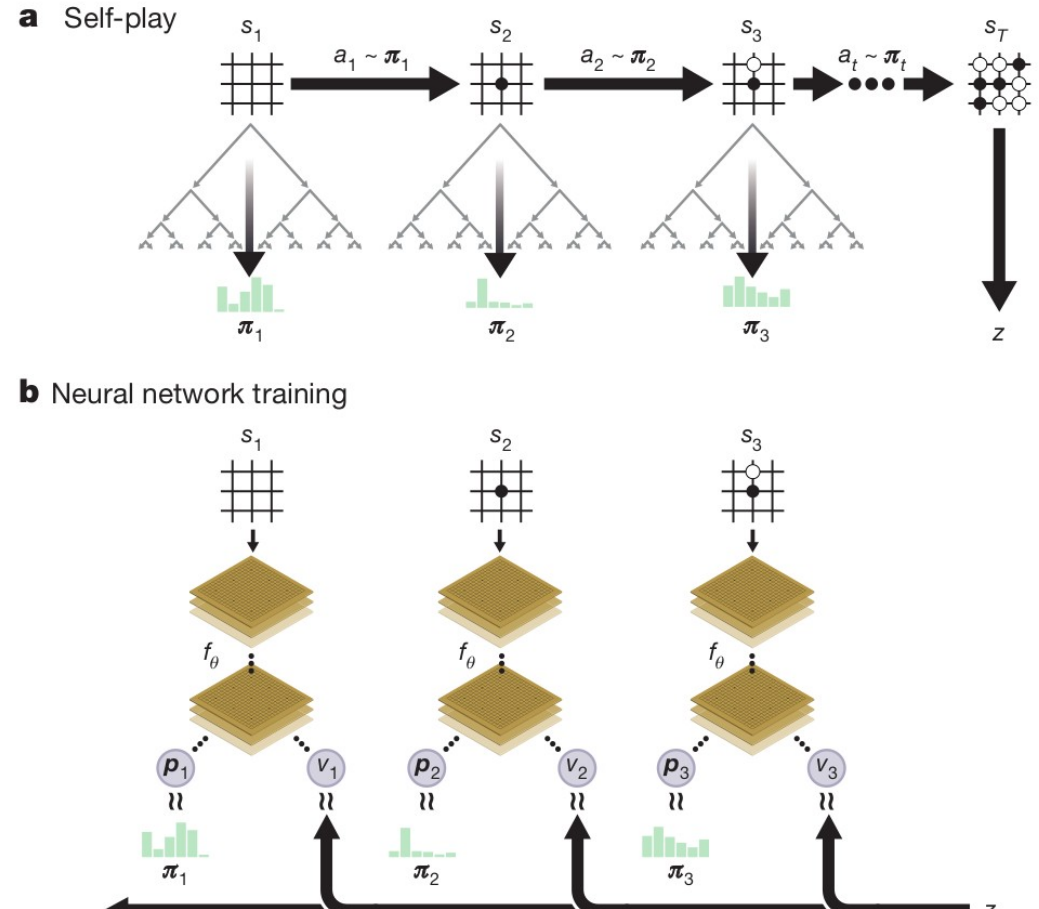
# RL achievement: AlphaGo

- Beating human champion Lee Sedol by a computer using deep RL combined with (a novel) tree search (Silver et al, 2016)
- Combo of supervised learning (from human expert games) and RL (random games of self-play)
- Thanks to fast computing (GPU enabled)



# RL biggest achievement: AlphaGo Zero

- Based on self-played games
- Does not need knowledge from human expert games (unlike its predecessor AlphaGo, beating it 100:0)
- Tabula rasa starting point
- MTCS chooses moves based on current NN (being trained)
- NN = merged value + policy net
- novel strategies invented
- economic interpretation of AlphaGo possible (Imani, 2018)



# Challenges and extensions to RL

- Curse of dimensionality
- Temporal credit assignment problem
- Partially observable problems
- Length of training
- Non-stationary environments
- Reward shaping
- Exploration-exploitation dilemma
- Multi-objective RL
- Intrinsic motivation approaches (copying w

# Summary

- Origin in psychology, biological relevance (in animals)
- RL = how an agent should behave in unknown environment, given only its percepts and occasional rewards
- The overall agent design dictates the kind of information that must be learned
- To be learned: utilities, optimal policy
- A variety of on-policy and off-policy methods
- For large spaces – function approximation inevitable
- Great promise for robotics