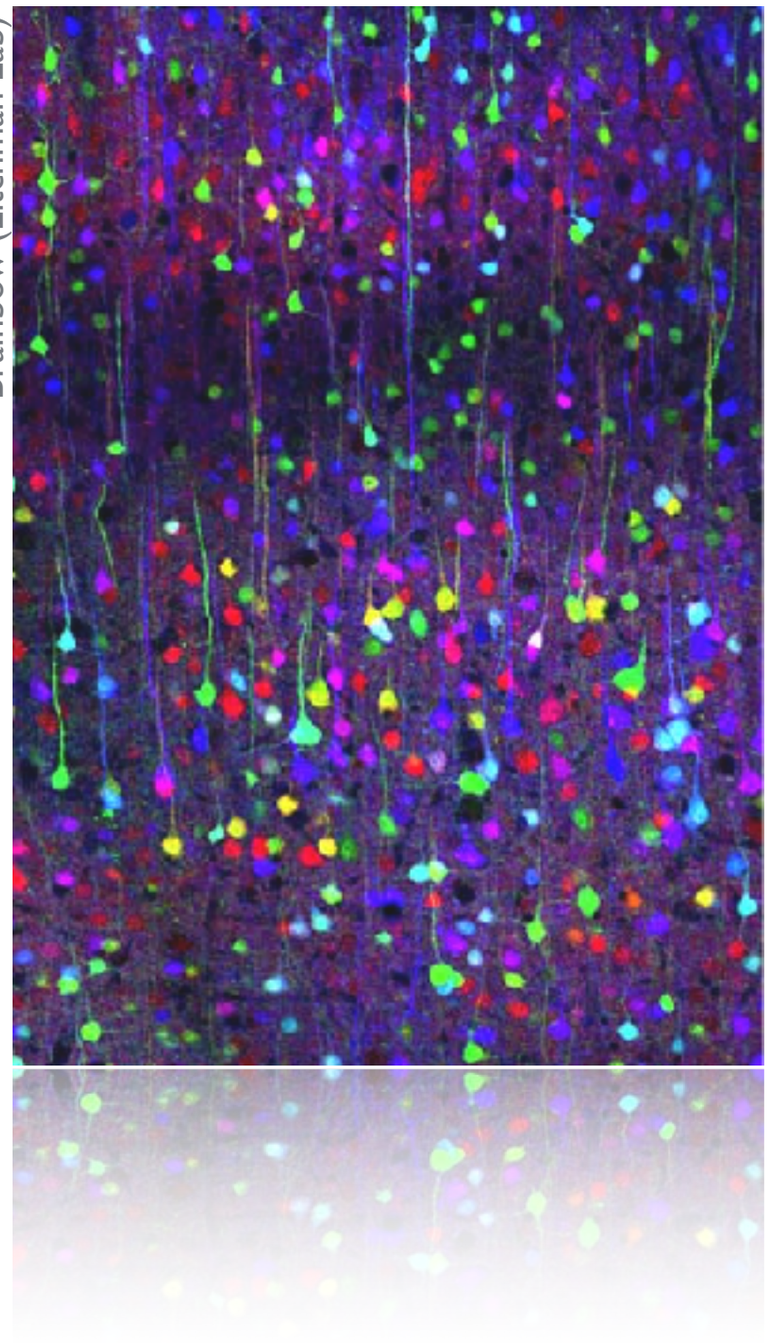
Information Processing & the Brain 2019/2020

University of BRISTOL



Brainbow (Litchman Lab)

Lecture 15

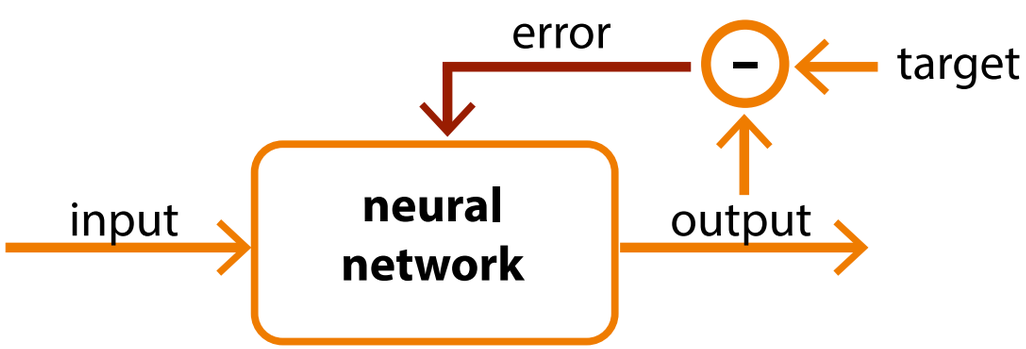
Neural circuits and learning:

Reinforcement Learning

Previously on *IPB*…

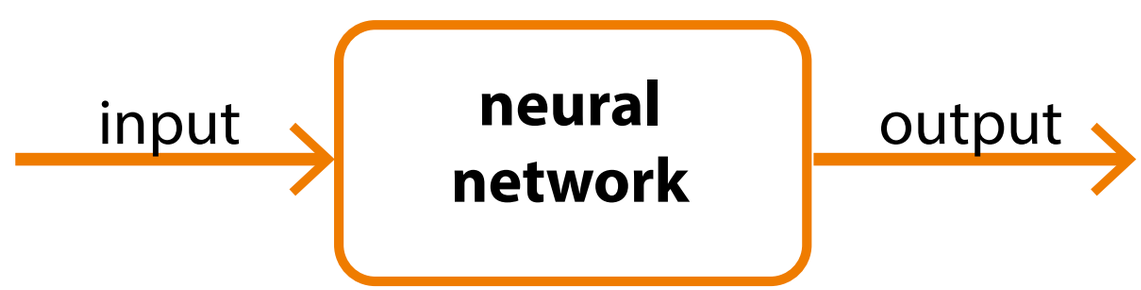
Supervised learning:

Relies on a teaching signal



Unsupervised Learning:

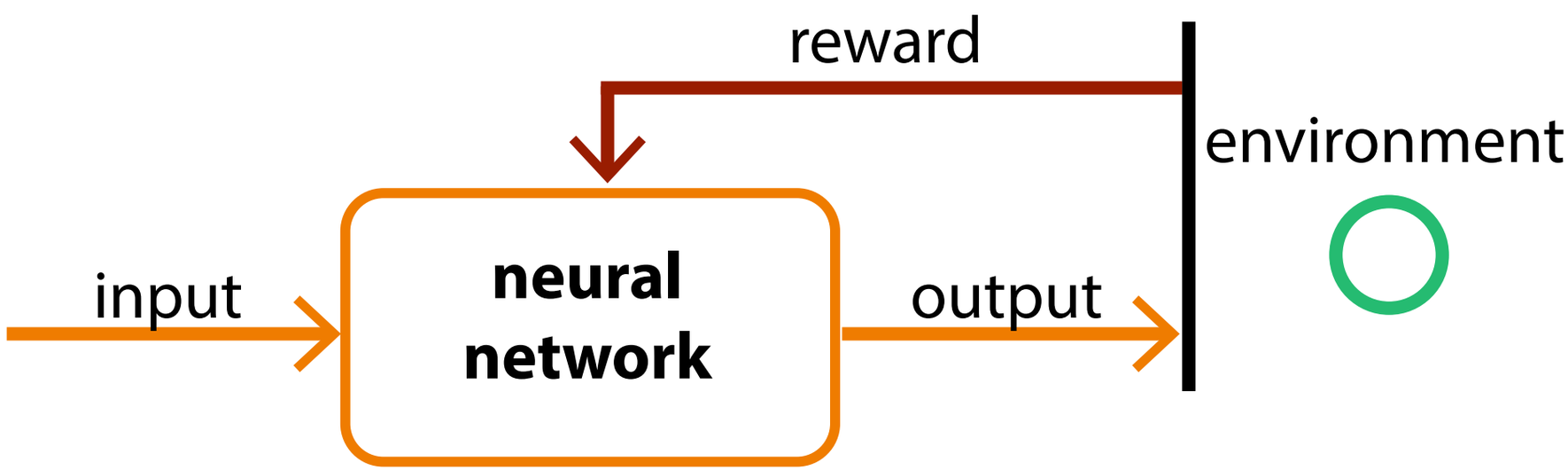
Extracts useful representations of input



But, what if an agent/animal is exploring the unknown?

Reinforcement Learning:

Learn to navigate in an environment



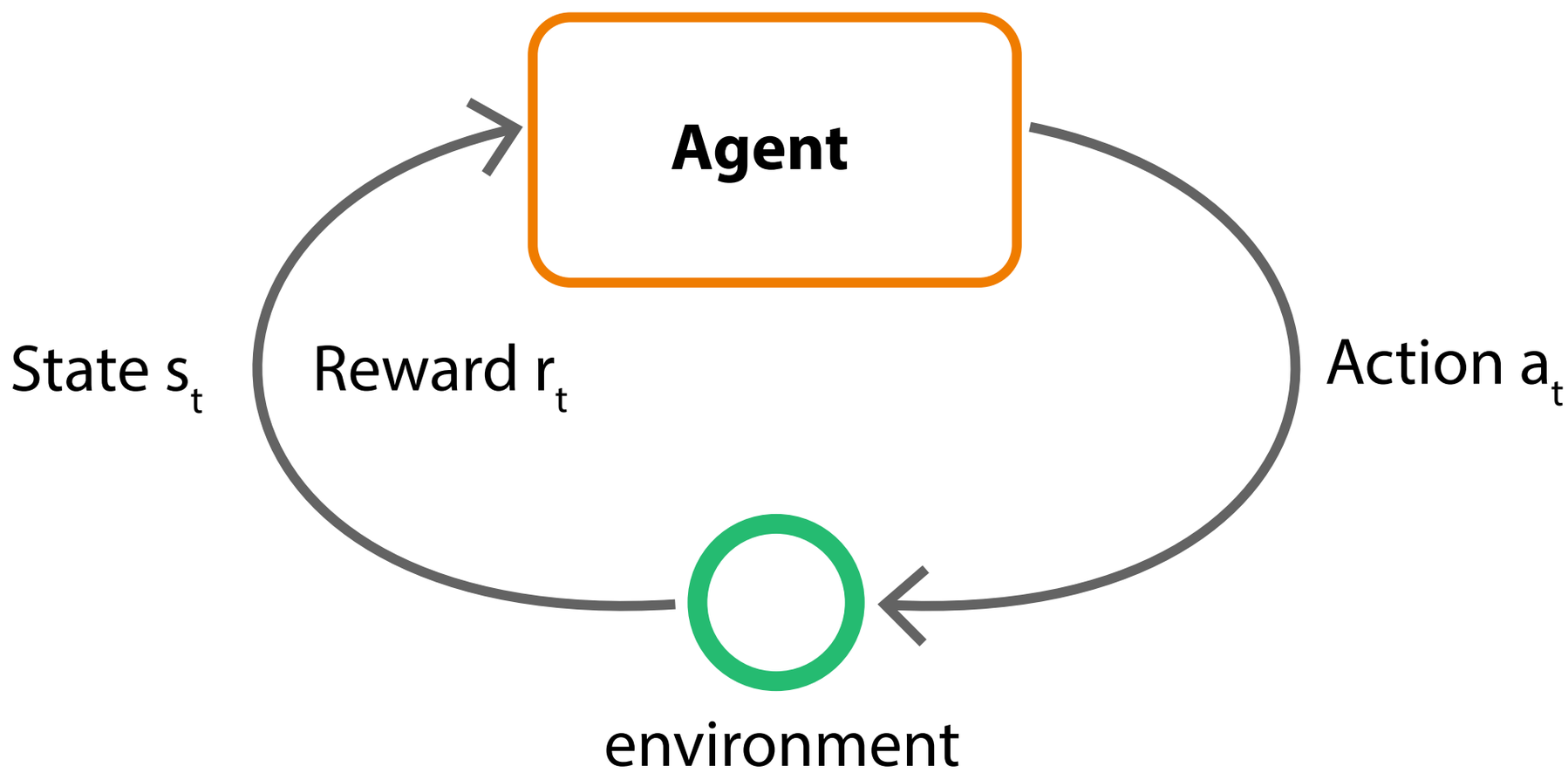
Outline

1. **Reinforcement learning: the basics**
2. **Temporal difference and Q-learning**
3. **RL in the brain**
4. **Deep RL**

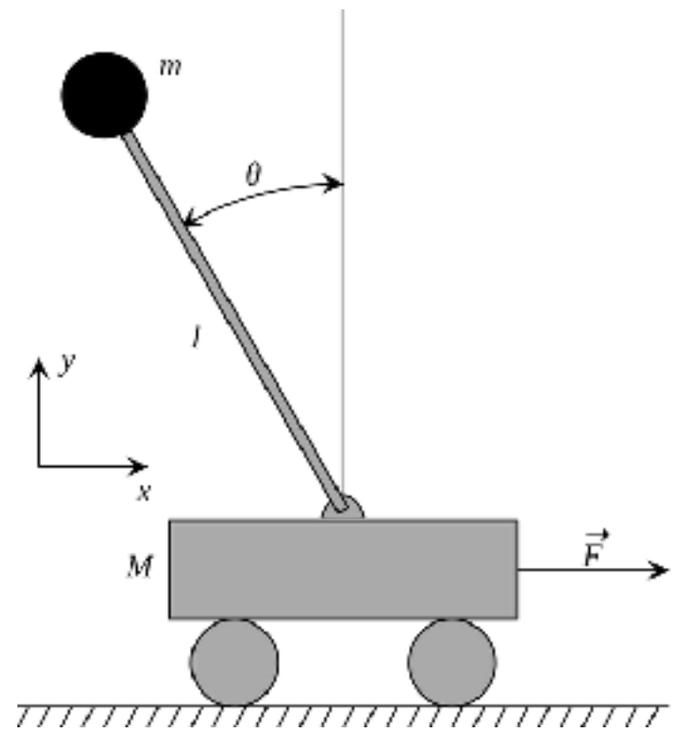
Reinforcement learning

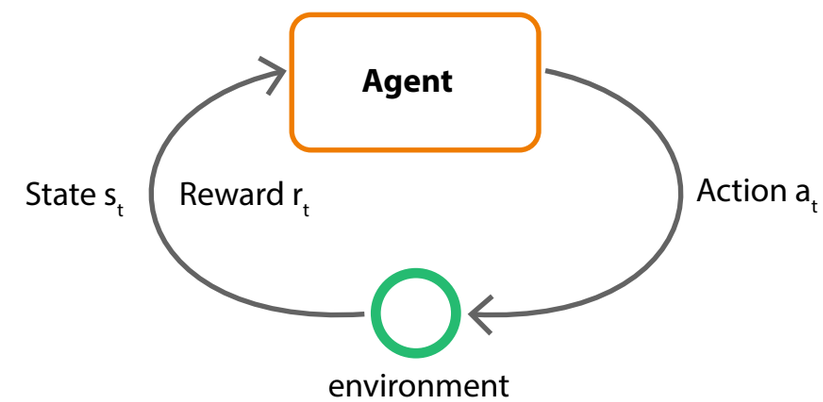
the basics

Goal: Learn which actions to take in order to maximise rewards!



Reinforcement learning examples: cart-pole problem





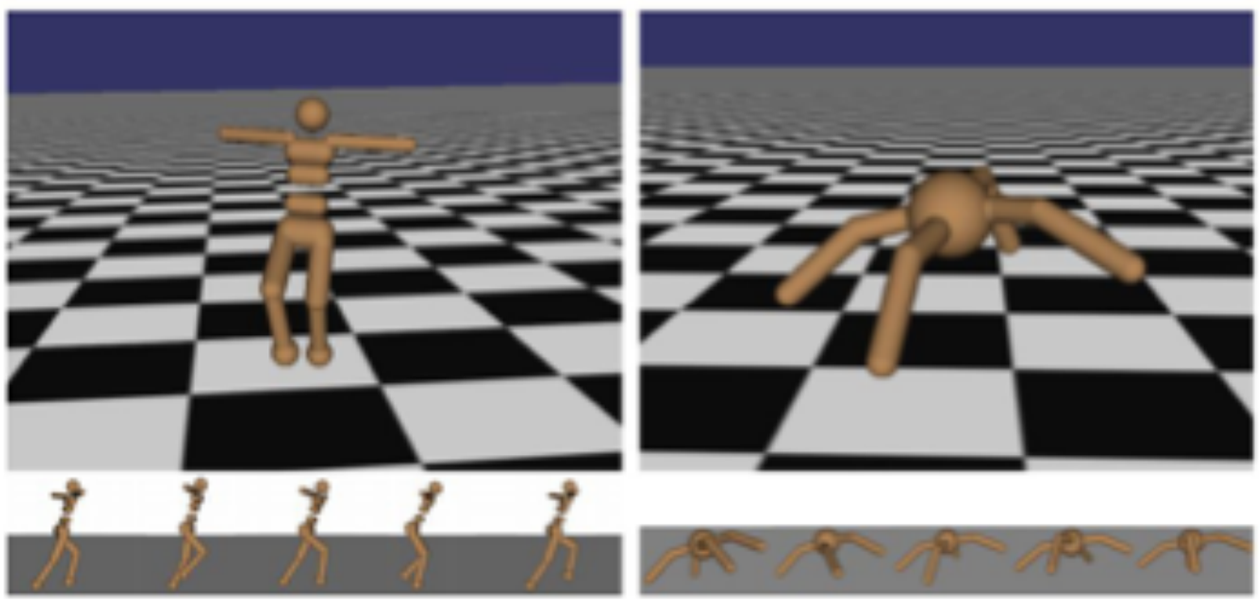
Goal: Balance a pole on top of a movable cart/car

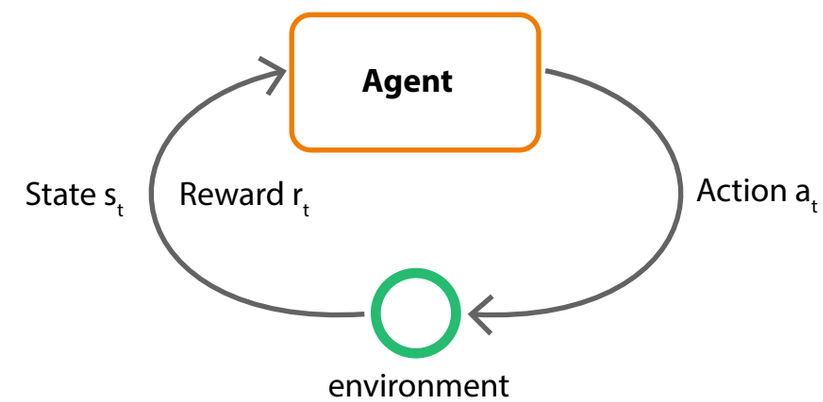
State, s_t: angle, angular speed, position, horizontal velocity

Action(s), a_t: horizontal force applied on the cart

Reward, r_t: 1 at each time step if the pole is upright, 0 (or -1) otherwise

Reinforcement learning examples: learn to move (locomotion)





Goal: Make a simulated animal move forward

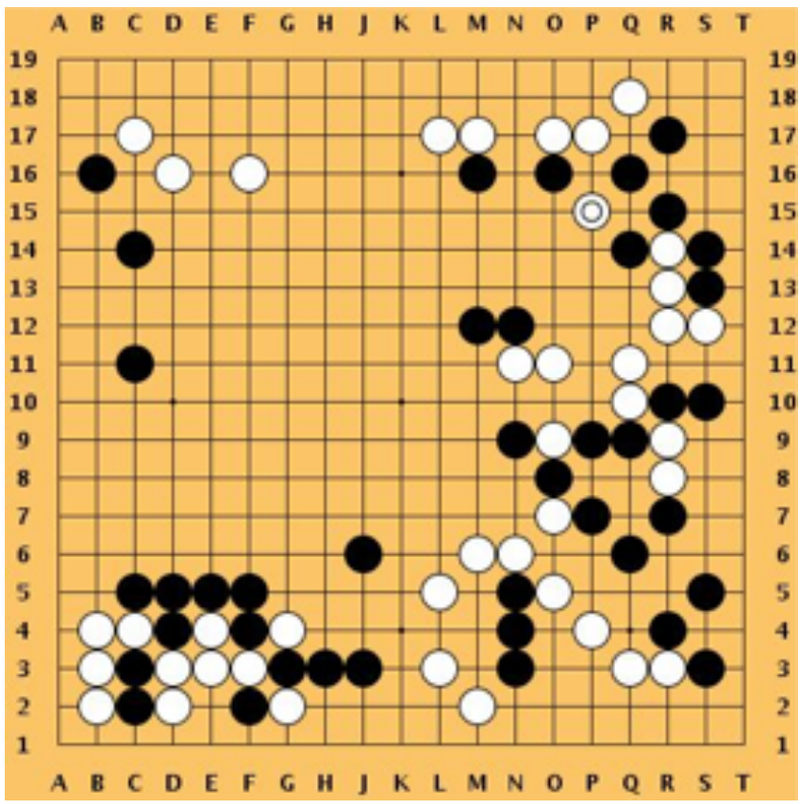
State, s_t: angle and position of joints

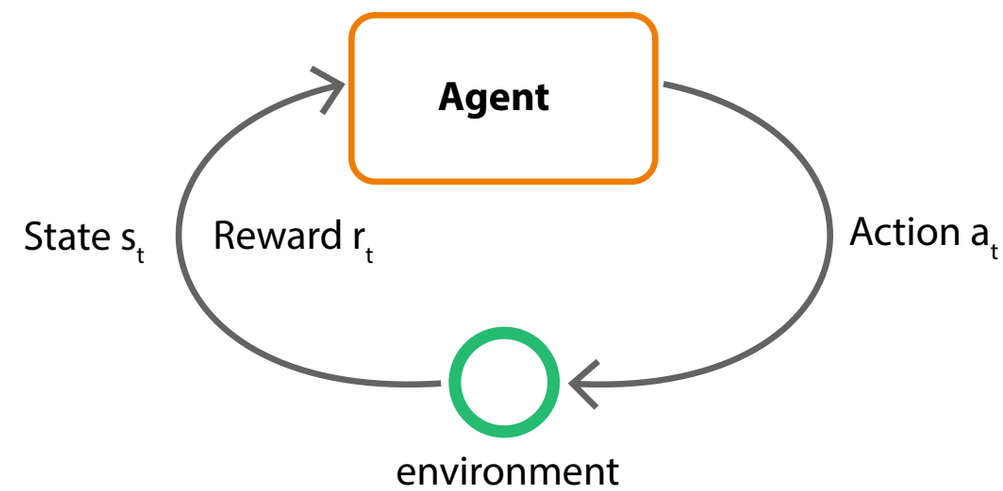
Action(s), a_t: Torque (or force) applied to joints

Reward, r_t: 1 for each time step the animal is upright, 0 (or -1) otherwise

Schulman et al. 2016

Reinforcement learning examples: Board game Go





Goal: Win the game!

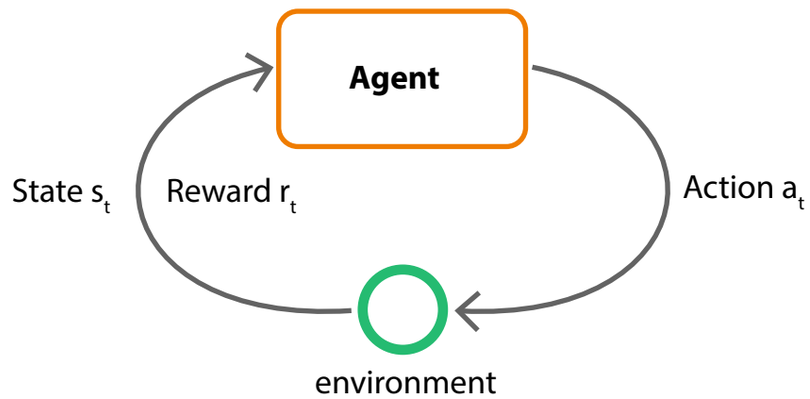
State, s_t: Position of the game pieces

Action(s), a_t: Where to place next piece

Reward, r_t: 1 if won, -1 if lost (super sparse reward!)

Silver et al. Nature (2017)

Reinforcement learning examples: playing Atari games



Goal: Maximise game score

State, s_t: raw pixels of the current game state (high dimensional state!)

Action(s), a_t: Game controls (e.g. left, right, up, down, fire, etc.)

Reward, r_t: Score changes (increase, decrease) at each time step

Mnih et al, Nature (2013)

Reinforcement learning

General framework: Markov Decision Process

Markov property: “The future is independent of the past given the present.”

In mathematical terms:

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, \ldots, S_t)$$

This means that the *current state has all the information needed to make a decision about what to do next* (note that information of the sequence that lead to a particular state is lost).

Sutton and Barto book (2018)

Reinforcement learning

General framework: Markov Decision Process

Markov decision process: Is a Markov Process defined by the following tuple:

$$(S, A, P, R, \gamma)$$

S: Set of states

A: Set of actions

P: Transition probability $P^a_{ss'} = \mathbb{P}(S_{t+1} = s' \,|\, S_t = s, A_t = a)$

R: Reward function (for state, action pairs) $R^a_s = \mathbb{E}(R_{t+1} = s' \,|\, S_t = s, A_t = a)$

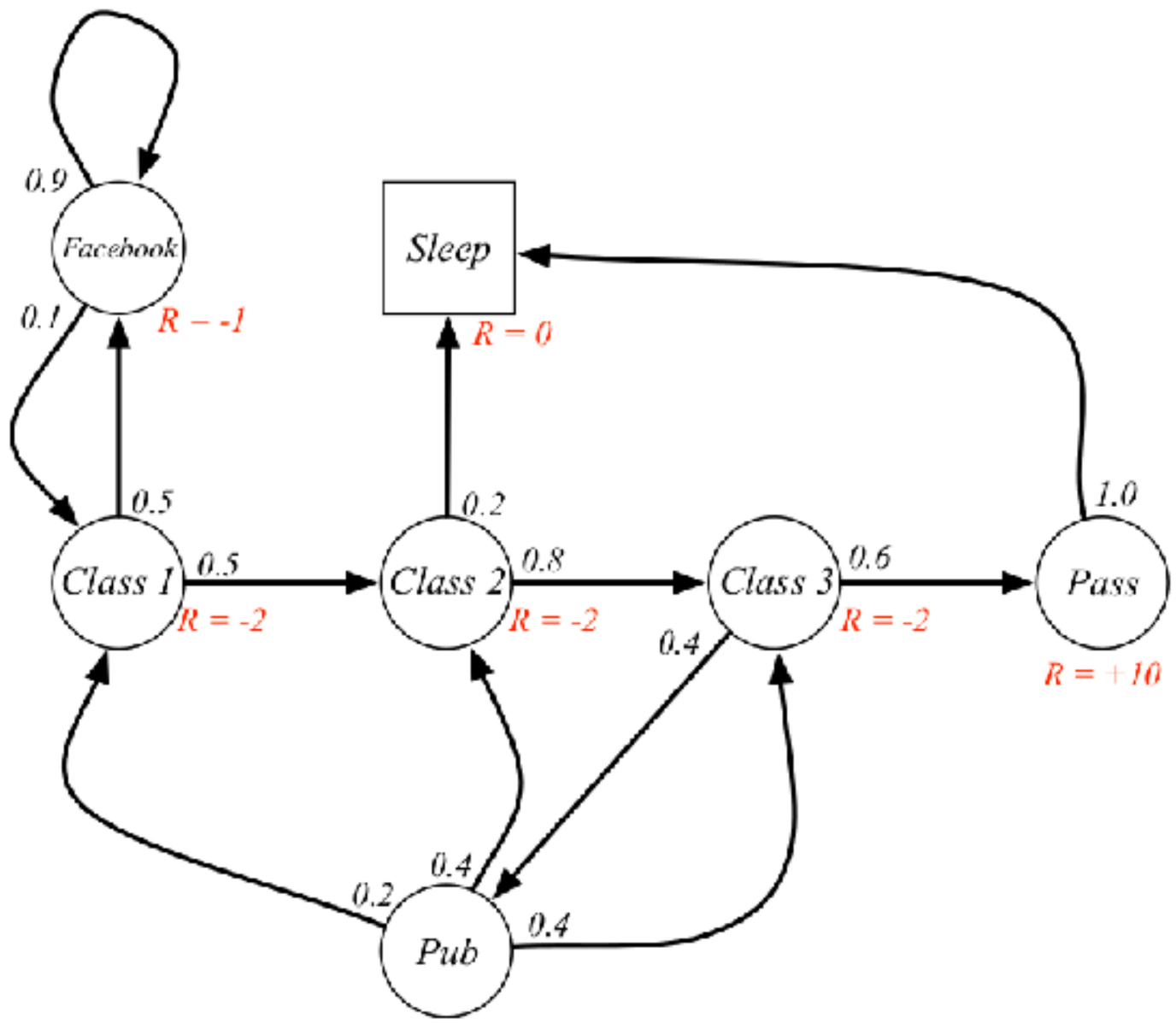
γ: discount factor [0,1]

Sutton and Barto book (2018)

Reinforcement learning

General framework: Markov Decision Process

Example of a **Markov decision process** for a student:



Silver lectures (UCL)

Reinforcement learning

Goal, policy and reward..

Goal: find **policy** that maximises **discounted reward**.

Policy: Distribution over actions given states, it defines the behaviour of an agent:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

Discounted reward: Total (discounted) reward received by the agent:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Sutton and Barto book (2018)

Reinforcement learning

Value-state function

It is useful to define the ***value*** of a state as its expected cumulative rewards:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Sutton and Barto book (2018)

Reinforcement learning

Bellman equation

But how to update the Value function when faced with a new environment?

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

<u>*Bellman equation*</u> for a value-state function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[r_0 + \gamma V^{\pi}(s_1) \,|\, s_0 = s]$$

Sutton and Barto book (2018)

Reinforcement learning

Temporal difference (TD) learning

TD learning: model-free reinforcement learning method which learns by bootstrapping based on current value function estimates.

From the Bellman equation we get an iterative method for updating the value function (TD learning):

$$\underbrace{V(S_t)}_{\text{value}} = V(S_t) + (\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \lambda \underbrace{V(S_{t+1})}_{\text{future value}} - V(S_t)}^{\delta\text{, prediction error}})$$

λ: discount factor

Sutton and Barto book (2018)

Reinforcement learning

Temporal difference (TD) learning

$$\underbrace{V(S_t)}_{\text{value}} = V(S_t) + (\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \gamma \underbrace{V(S_{t+1})}_{\text{future value}} - V(S_t)}^{\delta\text{, prediction error}})$$

γ: discount factor

Tabular TD(0) for estimating v_π

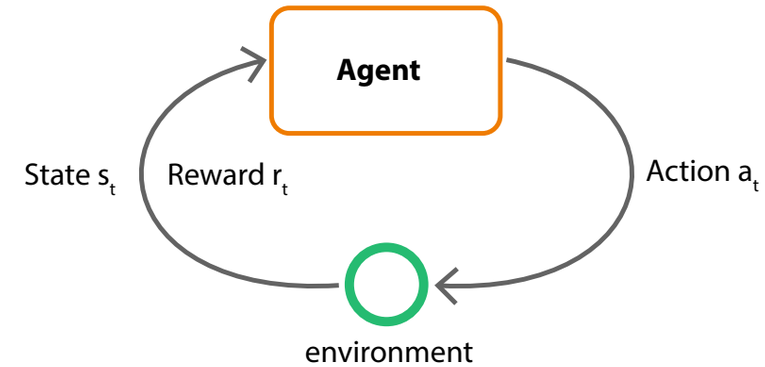
Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Sutton and Barto book (2018)

Reinforcement learning

Gridworld example



Actions = {left, right, up, down}

Values

0	0	* (R=1)
0	0	0
Start	0	0

Policy

		* (R=1)
Start		

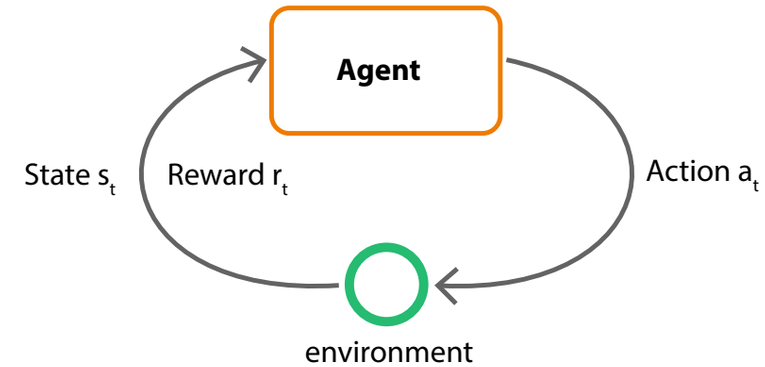
1. Define Value-state table (we assume that values are zero initially)

2. Define Policy (we use a random policy to start with)

Sutton and Barto book (2018)

Reinforcement learning

Gridworld example



Values

(example after a few iterations)

0.6	0.25	* (R=1)
0.3	0.5	0.8
Start	0.3	0.2

Policy

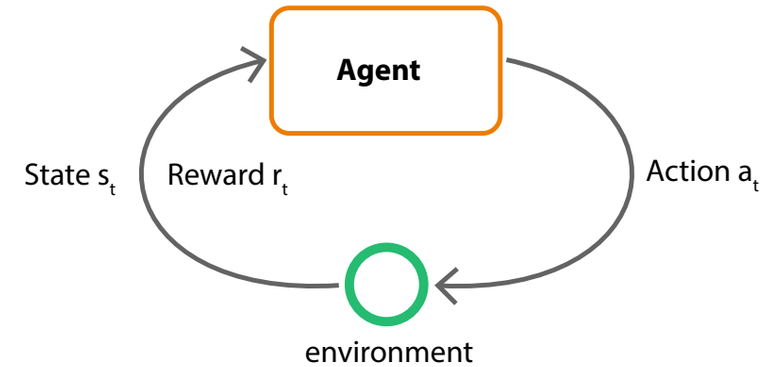
		* (R=1)
Start		

1. Define Value-state table (we assume that values are zero initially)
2. Define Policy (we use a random policy to start with)
3. Sample actions-states pairs (i.e. s_0, a_0, s_1, a_1, s_2, a_2 …)
4. Until a stop criteria is met (e.g. Terminal state is achieved or after ***n*** iterations)

Sutton and Barto book (2018)

Reinforcement learning

Gridworld example



Values

(example after a few iterations)

0.6	0.8	* (R=1)
0.3	0.6	0.8
Start	0.3	0.6

Policy *

(optimal policy)

→	→	* (R=1)
↑→	↑→	↑
↑→ **Start**	↑→	↑

1. Define Value-state table (we assume that values are zero initially)
2. Define Policy (we use a random policy to start with)
3. Sample actions-states pairs (i.e. s_0, a_0, s_1, a_1, s_2, a_2 …)
4. Until a stop criteria is met (e.g. Terminal state is achieved or after ***n*** iterations)

Sutton and Barto book (2018)

Reinforcement learning

Q-learning

In Q-learning we learn a action-value function.

$$\underbrace{Q(S_t, A_t)}_{\text{value}} = Q(S_t, A_t) + (\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \gamma \max_a \underbrace{Q(S_{t+1}, a)}_{\text{future value}} - Q(S, A)}^{\delta\text{, prediction error}})$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Sutton and Barto book (2018)

Reinforcement learning

TD vs Q-learning

TD-learning uses the value of the **state given** by the **policy** (on policy method):

$$\underbrace{V(S_t)}_{\text{value}} = V(S_t) + (\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \gamma \underbrace{V(S_{t+1})}_{\text{future value}} - V_t}^{\delta\text{, prediction error}})$$

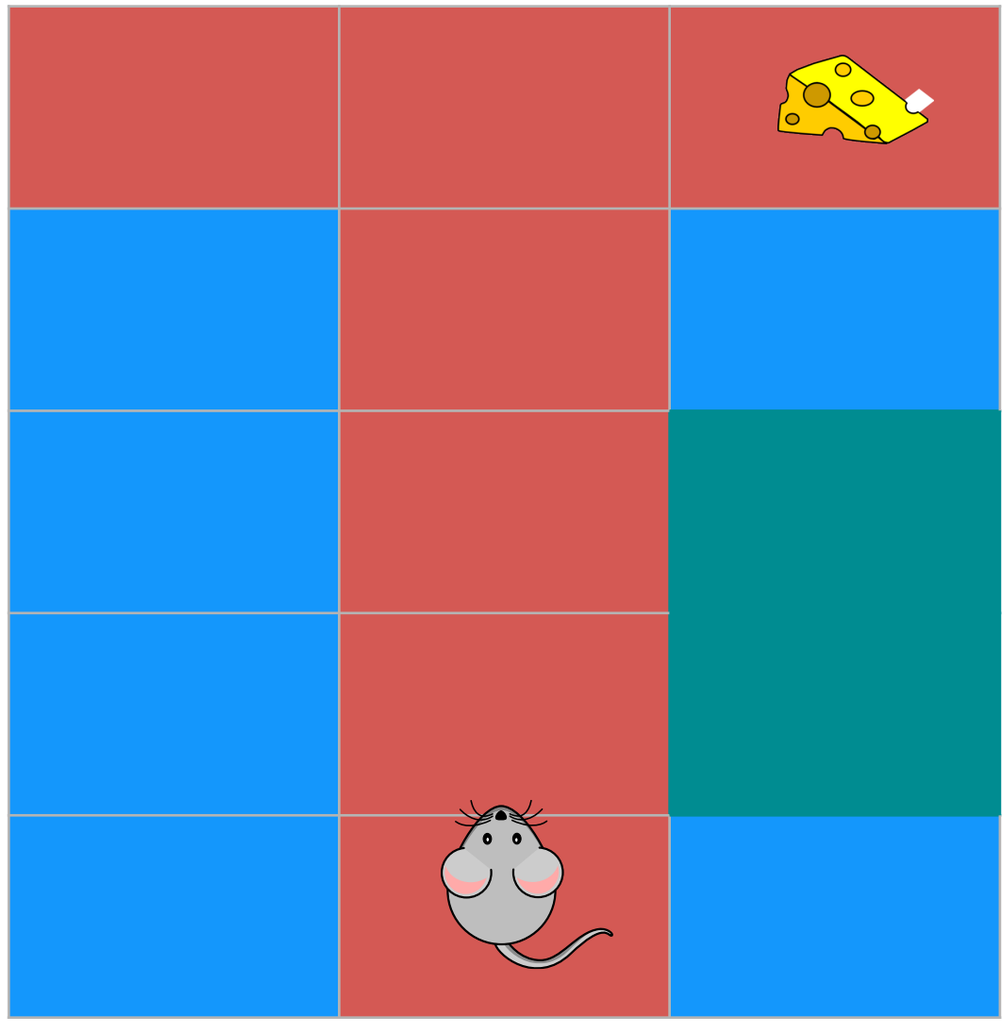
Q-learning uses the **best possible action**, not necessarily the one given by the policy (off policy method):

$$\underbrace{Q(S_t, A_t)}_{\text{value}} = Q(S_t, A_t) + (\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \gamma \max_a \underbrace{Q(S_{t+1}, a)}_{\text{future value}} - Q(S, A)}^{\delta\text{, prediction error}})$$

Sutton and Barto book (2018)

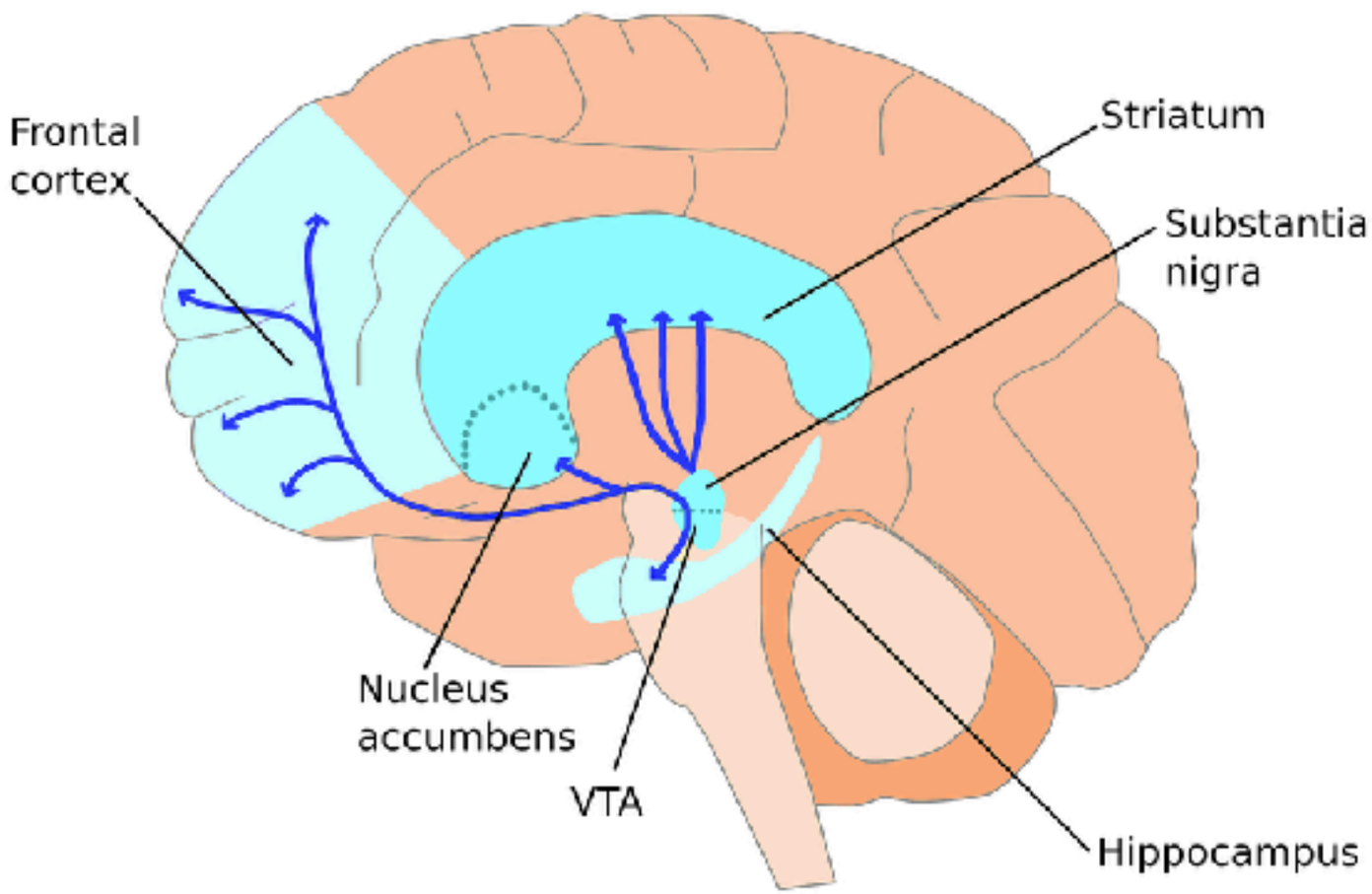
Reinforcement learning in the brain
Typical experiment

Typical T-maze experiment:



Reward pathways in the brain

Specific brain regions encode reward signals. A classical example is ***dopamine*** which is neuromodulator responsible for reward-driven behaviour, which is released from regions such as the ***Ventral tegmental area*** (VTA).



Reward prediction errors
dopamine

Recall that TD-learning suggests a specific

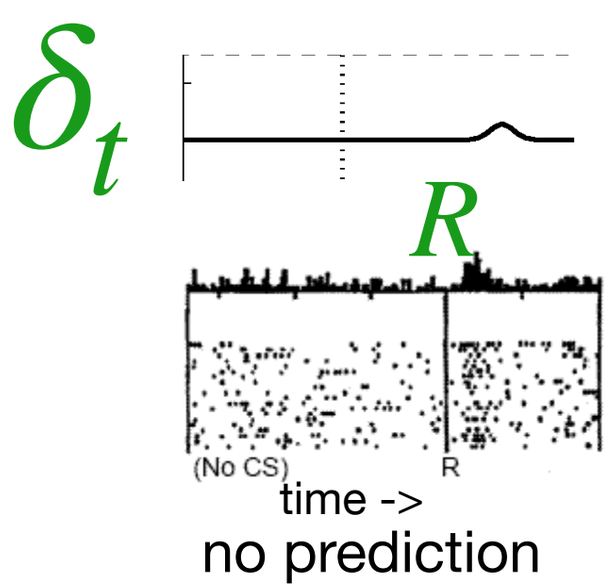
reward prediction error:

$$\underbrace{V(S_t)}_{\text{value}} = V(S_t) + \overbrace{(\underbrace{R_{t+1}}_{\text{reward}} + \gamma \underbrace{V(S_{t+1})}_{\text{future value}} - V(S_t))}^{\delta\text{, prediction error}}$$

Reward prediction errors
dopamine

Reward prediction error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$



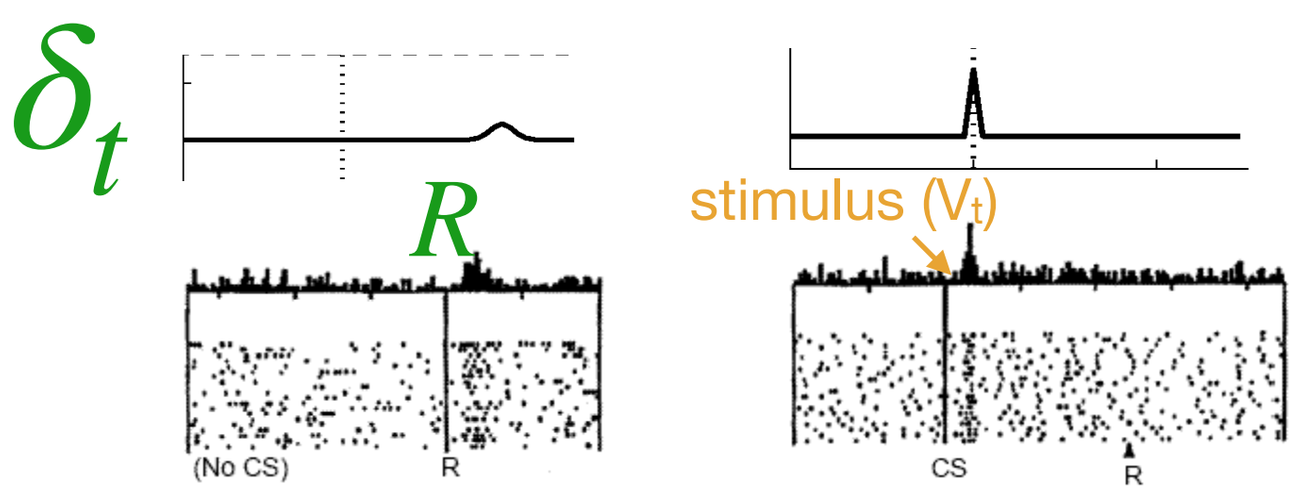
CS: conditioned stimulus

Schultz et al. (1997)

Reward prediction errors
dopamine

Reward prediction error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$



no prediction

prediction: reward

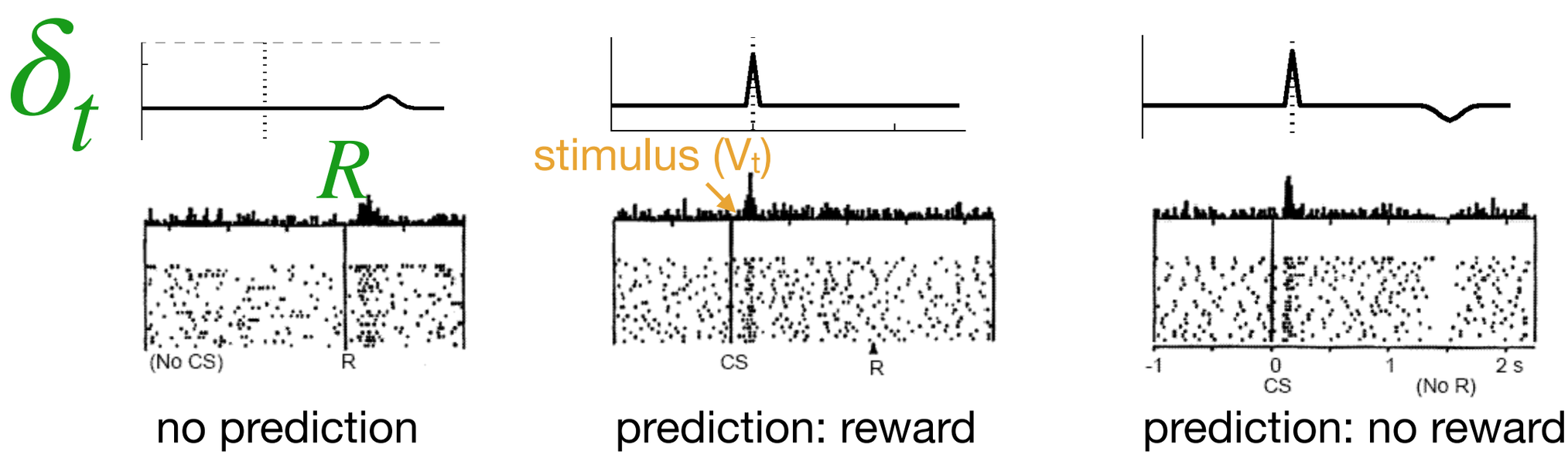
CS: conditioned stimulus

Schultz et al. (1997)

Reward prediction errors
dopamine

Reward prediction error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$



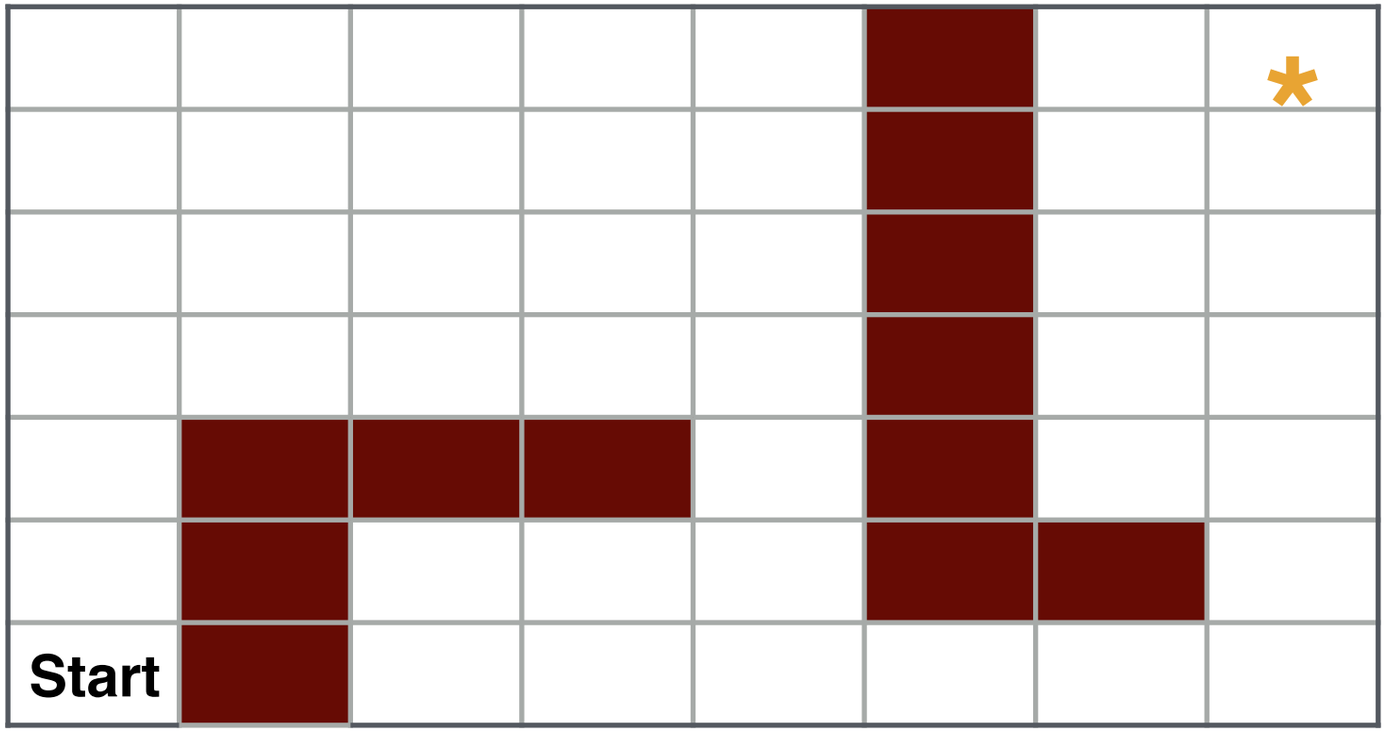
CS: conditioned stimulus

Schultz et al. (1997)

Group discussion
groups of 2-3 (5min)

What if the state space is **much larger**

(for example hundreds or millions of states)?

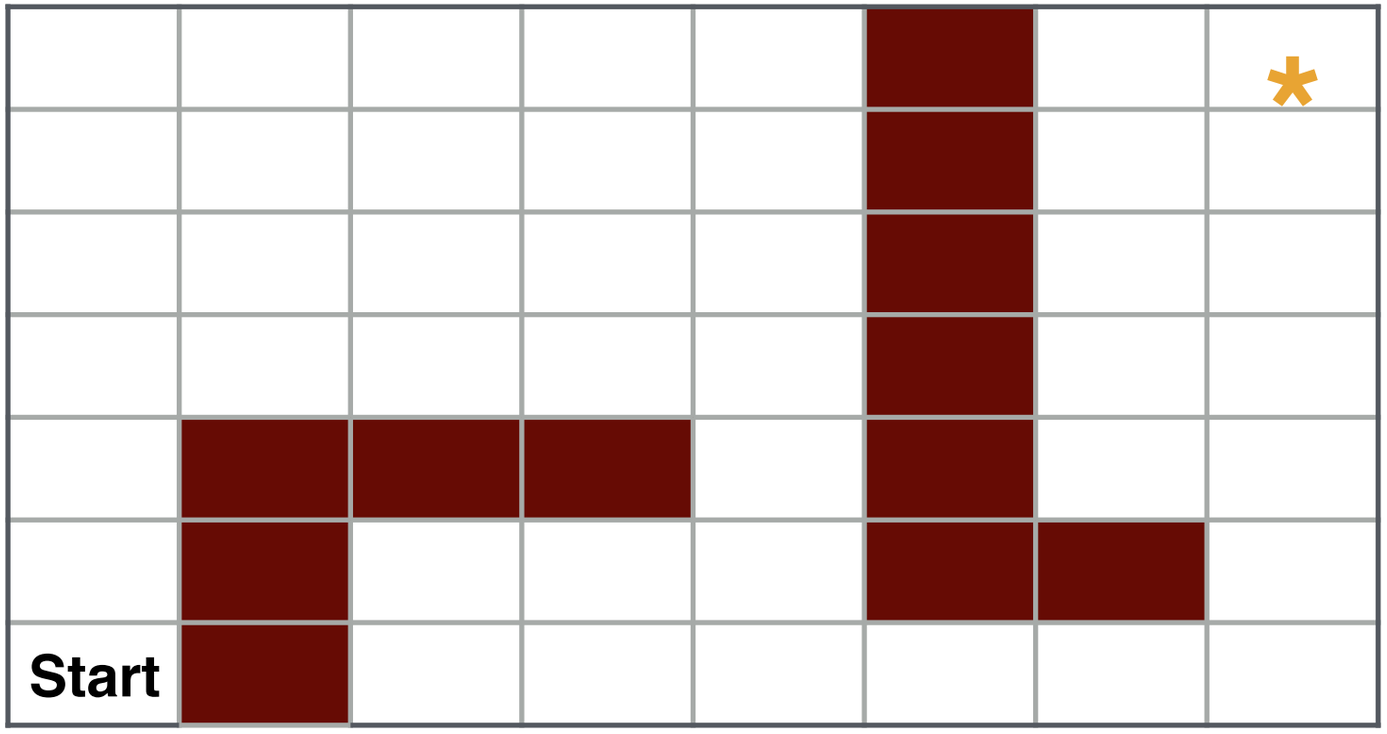


Group discussion
groups of 2-3 (5min)

What if the state space is **much bigger**

(for example hundreds or millions of states)?

Possible solutions: Use exploration-exploitation strategies, and/or function approximates (e.g. neural networks).



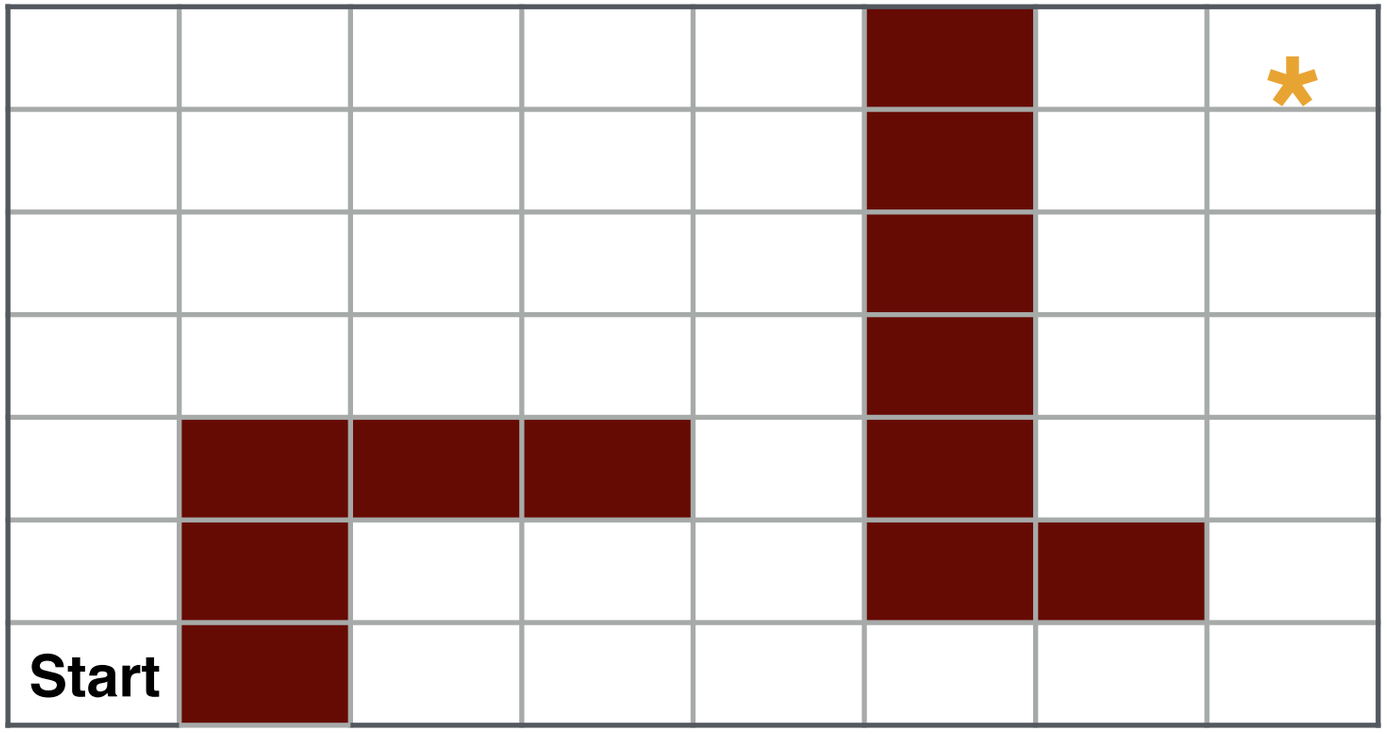
Reinforcement learning

Exploration-exploitation tradeoff

Use a good *exploration-exploitation tradeoff*.

Exploitation: make the best decision given current information

Exploration: gather more information



Reinforcement learning

Policy: ε-greedy

ε-greedy policy: a policy for when to choose the best available action ***a**** or a random action.

$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \boldsymbol{\varepsilon} \\ \text{random action} & \text{with probability } \boldsymbol{\varepsilon} \end{cases}$$

- **ε=1** gives a purely **exploratory** policy
- **ε=0** gives a purely **exploitative** policy
- **ε=]0,1[** Somewhere in-between gives a tradeoff between exploration-exploitation. It is often a good idea to explore more in the beginning and exploit more towards the end of optimization.

Sutton and Barto book (2018)

Deep Q-learning

Recent success in reinforcement learning relies on two tricks:

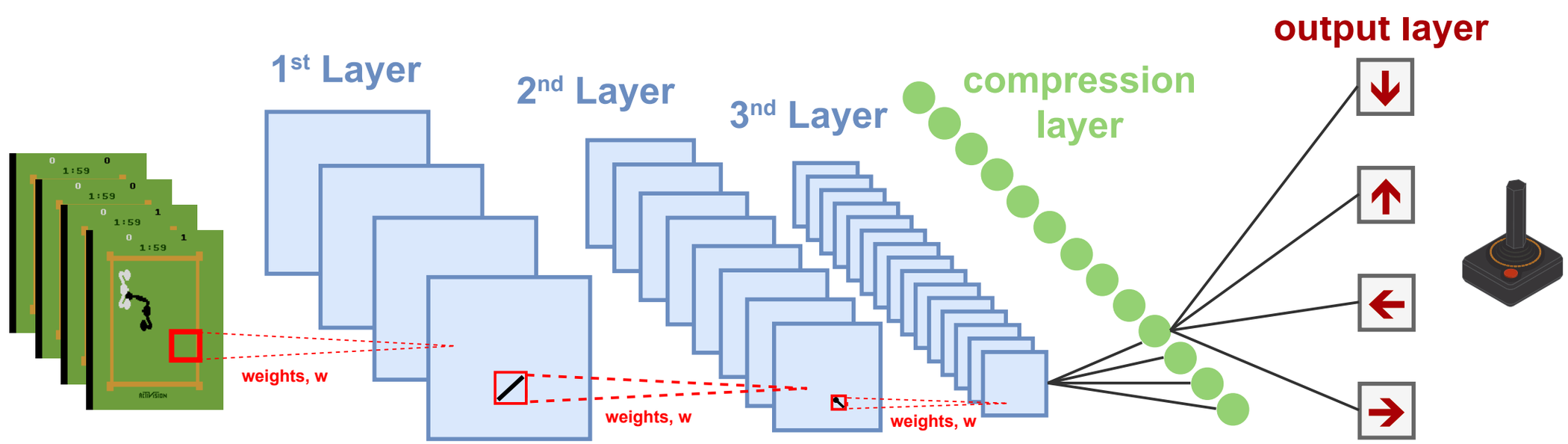
Trick 1: Use neural network as $Q(S,A;\theta)$ function approximator with parameters θ

Trick 2: Use separate memory system to *avoid local optima*

Deep Q-learning

Trick 1: deep net as function approximator

Use deep network as a Q(S,A; θ) function approximator when in the presence of high-dimensional state spaces:



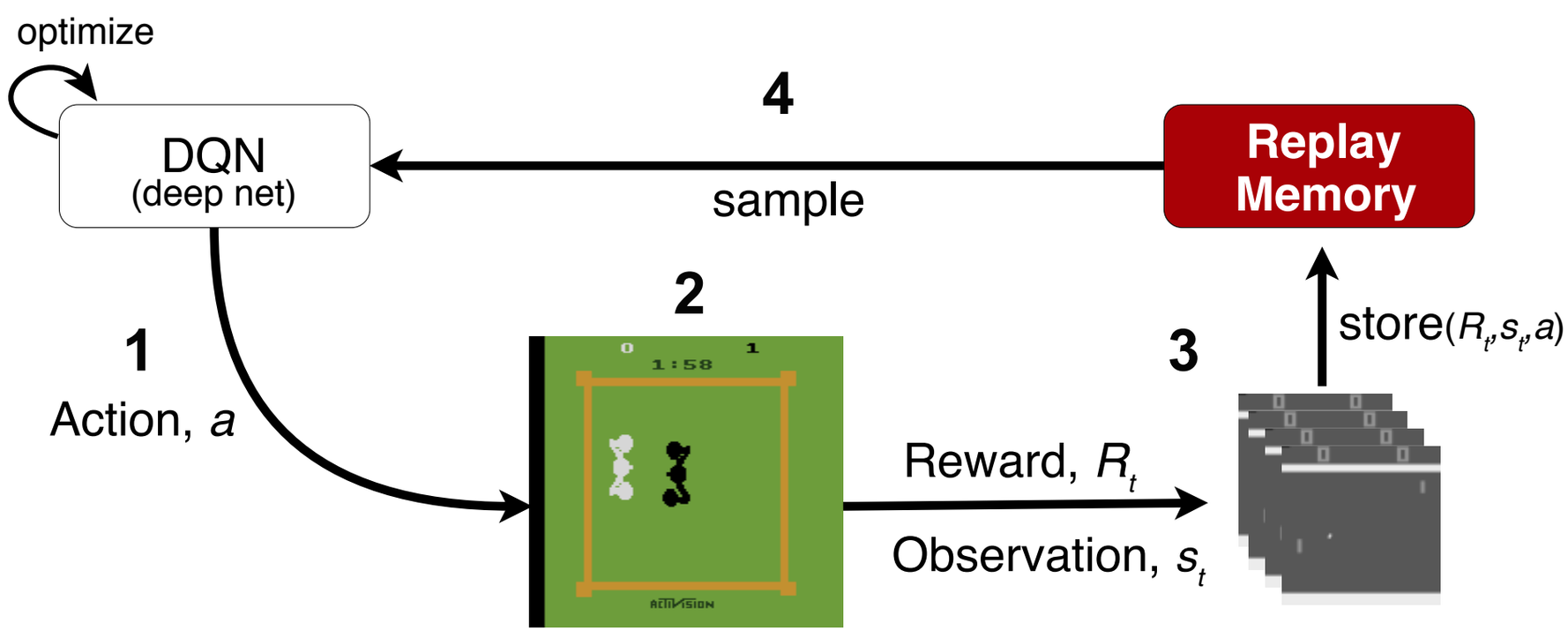
Input (pixels) → **Low level features** → **Higher level features** → **Game action**

State S_t → **$Q(S_t,A_t)$**

Mnih et al, Nature (2013)

Deep Q-learning

Trick 2: Replay previous episodes



1. Agent (player) performs an ***action a*** given by a **deep neural net**

2. This action triggers a new ***obervation*** $\boldsymbol{s_t}$ in the game and a ***reward*** $\boldsymbol{R_t}$

3. The new observation and reward are stored in the **replay memory**

4. This memory is then used to train a **deep neural network** (DQN)

Mnih et al, Nature (2013)

Deep Q-learning

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 et al, Nature (2013)

Deep Q-learning

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$$

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

Equivalent to standard Q-learning!

Cost used for backprop in the deepnet

$$\underbrace{Q(S_t, A_t)}_{\text{value}} = Q(S_t, A_t) + \overbrace{(\underbrace{R_{t+1}}_{\text{reward}} + \gamma \max_a \underbrace{Q(S_{t+1}, a)}_{\text{future value}} - Q(S, A))}^{\delta\text{, prediction error}}$$

Mnih et al, Nature (2013)

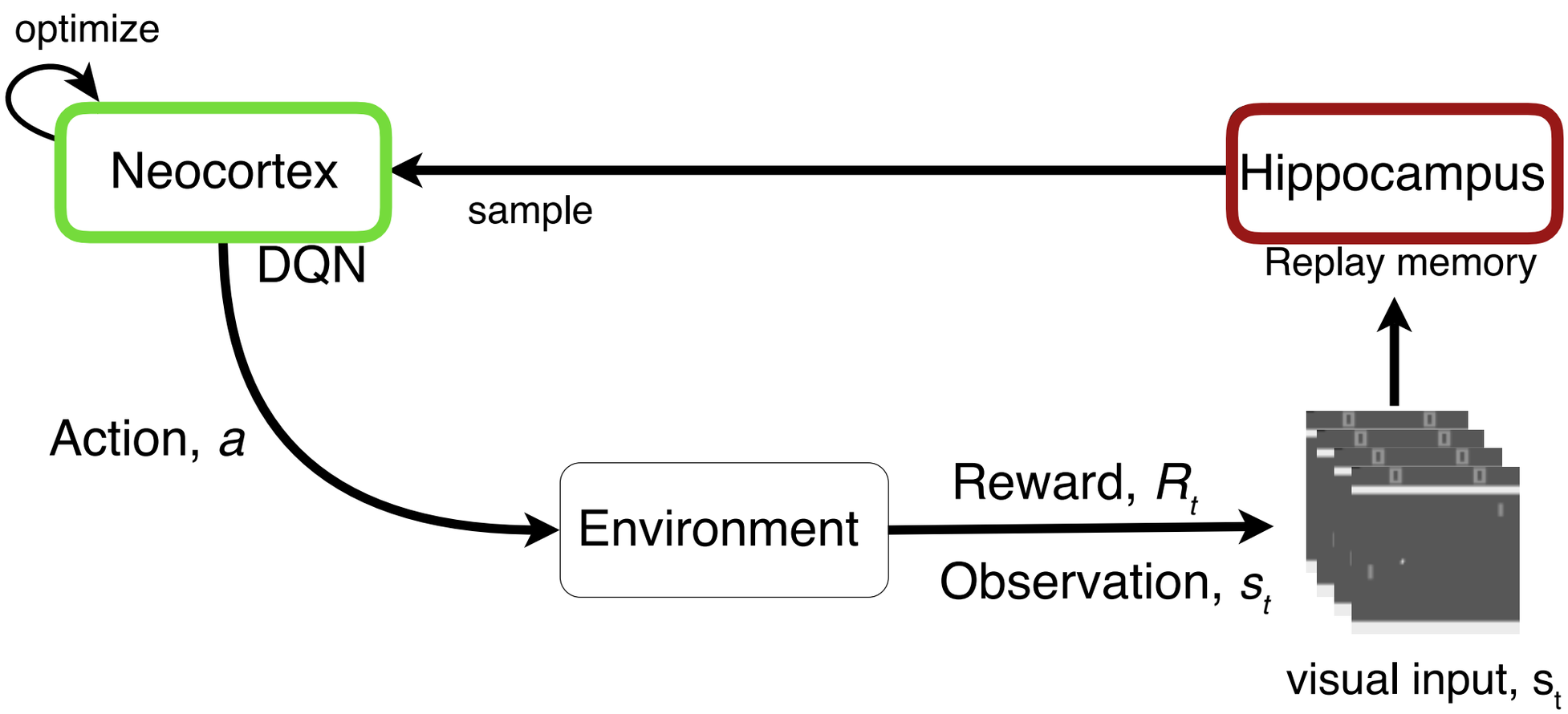
Deep Q-learning

Learning to play Atari games: Pong

Deep Q-learning

Learning to play Atari games: Seaquest

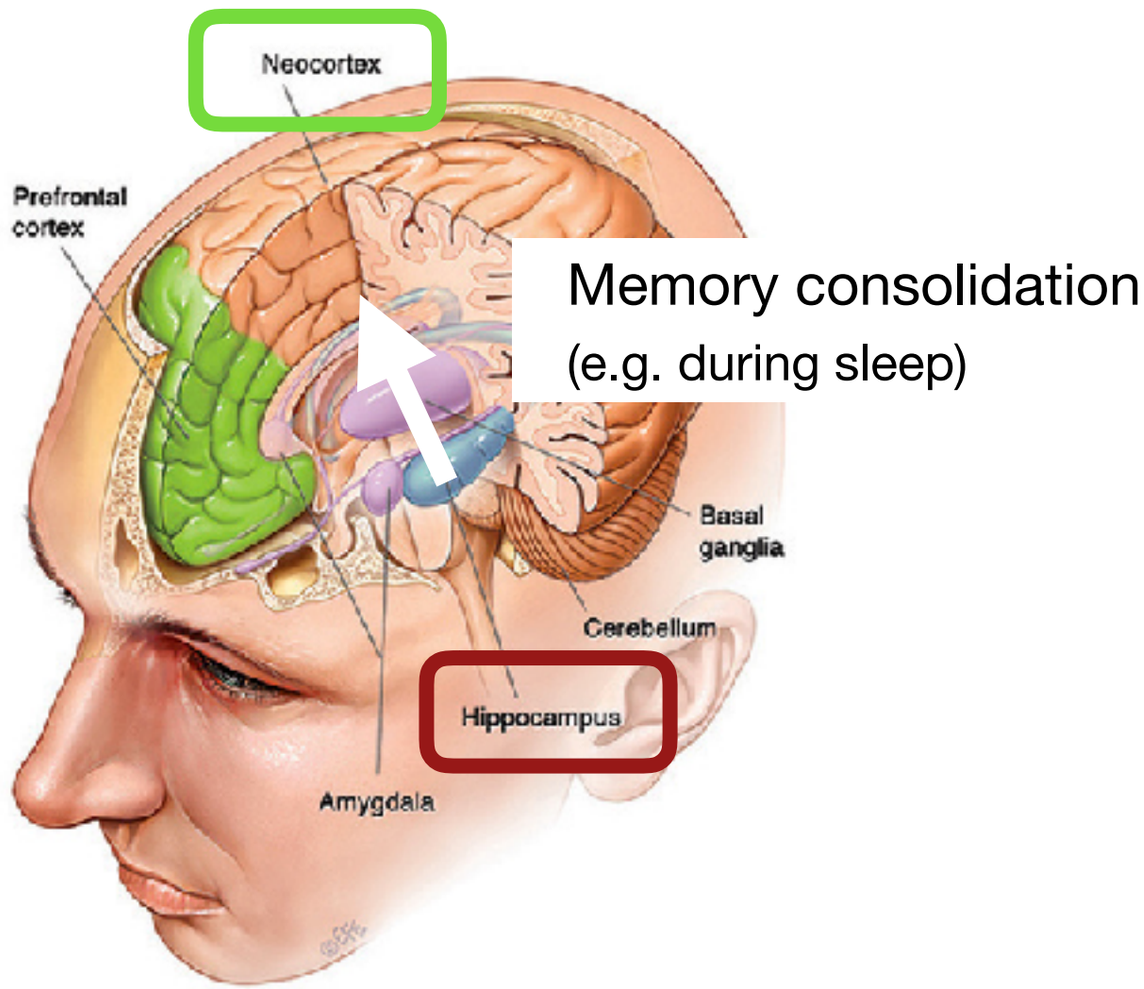
Deep Q-learning in the brain?



Mnih et al, Nature (2013)

Deep Q-learning in the brain?

Memory consolidation seems to rely on the *communication* between *hippocampus* (responsible for episodic memory) and the *neocortex*.



Summary

1. **Reinforcement learning relies on sparse rewards**

2. **TD-learning and Q-learning are two of the most common methods**

3. **The dopamine system encodes reward prediction errors**

4. **Replay in deep RL is inspired in neuroscience to improve RL methods**

References

Text books:

Reinforcement Learning, Sutton and Barto 2018 (2nd edition available online)

Theoretical neuroscience: Dayan and Abbott 2001 [Unsupervised learning: chapter 3]

Note: Some of the slides are based on RL slides by Li, Johnson and Yeung, Stanford University

Relevant papers:

- Niv, Reinforcement learning in the brain, Journal of Mathematical Psychology (2009)
- Schultz et al. Science (1997) (seminal paper looking at reward prediction errors in the dopaminergic system)

Upcoming lectures

- L11[6]: Neural circuits and learning: introduction
- L12[6]: Visual system: deep learning?
- L13[7]: Supervised learning & backprop
- L14[7]: Unsupervised learning
- L15[9]: Reinforcement learning
- **L16[9]: Temporal processing & recurrent neural networks**
- L17[10]: Recurrent neural networks
- L18[10]: Brain vs machine: meta-learning and inductive bias
- L19-20[11]: Neural data analysis (by Cian)
- L21-22[12]: Revision week