

# An introduction to reinforcement learning

## @ Harvard RL Workshop 2022

---

Lucy Lai and John Vastola

Dept. of Psychology

Dept. of Neurobiology

Center for Brain Science

Harvard University

- Slides: <https://bit.ly/RLtutorials>
- Companion notebook (courtesy of John!): [bit.ly/RLnotebook](https://bit.ly/RLnotebook)

+ Code + Text

An introduction to reinforcement learning @ Harvard RL Workshop 2022

A companion notebook by John Vastola and Lucy Lai

Welcome to the reinforcement learning tutorial! The goal of this tutorial is to present a brief introduction to

- (i) the basic concepts of reinforcement learning (RL), and
- (ii) how those concepts might relate to the brain.

This companion notebook contains demos which complement the main presentation, plus summaries of some of the essential points. It is self-contained, and only uses basic Python packages.

The structure of the notebook is as follows:

1. The Rescorla-Wagner model of classical conditioning
2. Generalizing Rescorla-Wagner using the temporal difference learning model
3. Intermission: Mathematical formulation of reinforcement learning problems
4. Temporal difference learning as a general model-free prediction algorithm

Can run online using Google Colab;  
don't need your own Python  
installation.

- Career level?
- Discipline?
- Knowledge of RL?

Let's get started!



When poll is active, respond at **PollEv.com/lucylai**  
Text **LUCYLAI** to **22333** once to join

## Are you a...

PI A Postdoc B Graduate student C Undergraduate student D Research assistant E Other F





< Activities



Visual settings



Edit

00

→

Loading...



< Activities



Visual settings



Edit

00

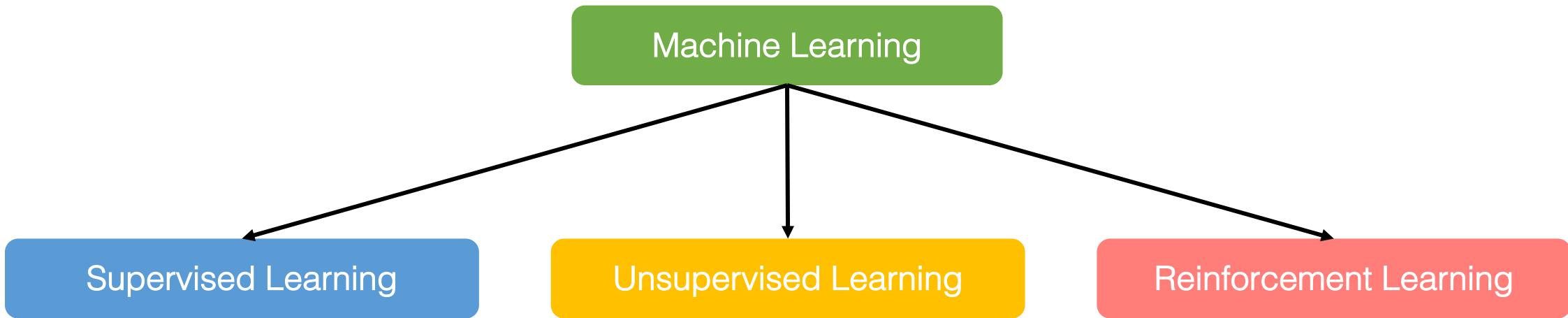
→

Loading...

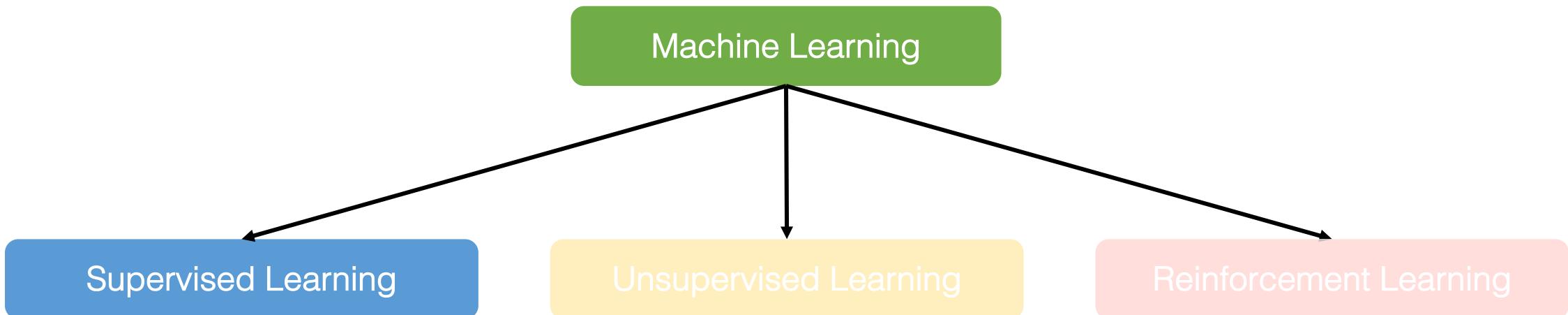
# Learning objectives & Key takeaways

- After today's session, you will be able to:
  - Define the key ingredients of RL: **agent, state, action, reward, policy, value** and how they work together in an algorithm
    - Gives you a primer for common terms used in talks
  - Implement the Rescola-Wagner rule and the TD-learning algorithm
  - Explain why dopamine encodes a **temporal difference (TD)** or **reward prediction error (RPE)**
  - Identify some open questions in RL

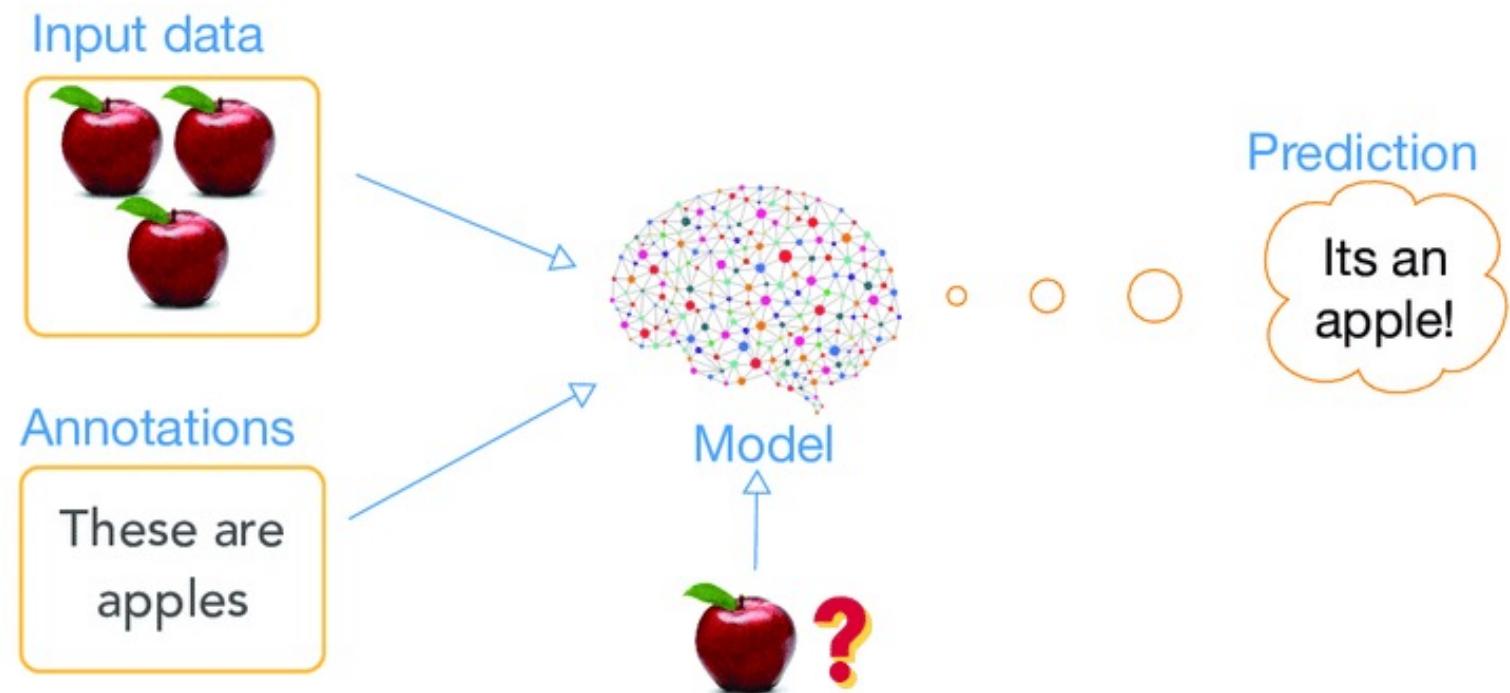
# Three types of learning



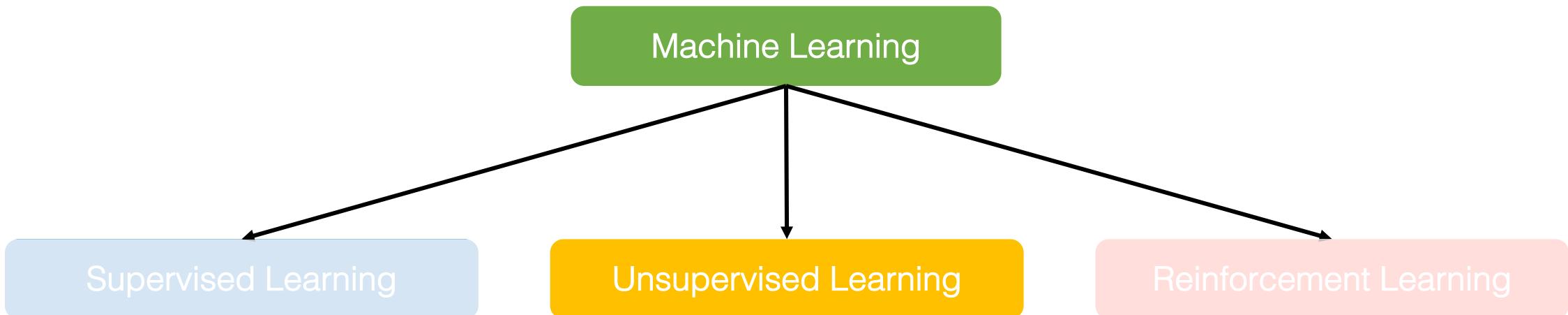
# Three types of learning



learning with a teaching signal;  
e.g. assigning pictures to labels

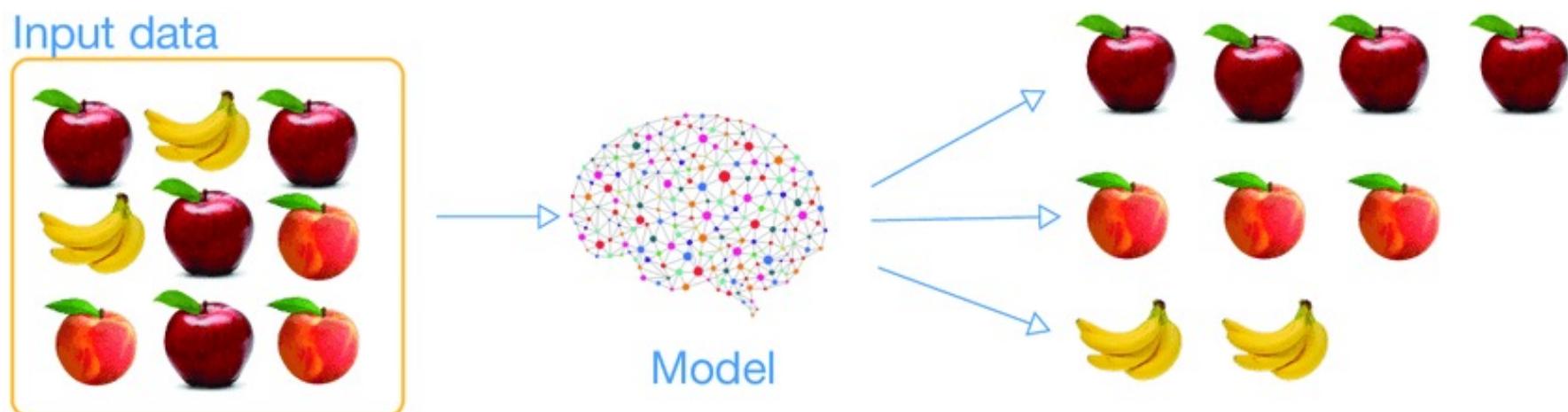


# Three types of learning

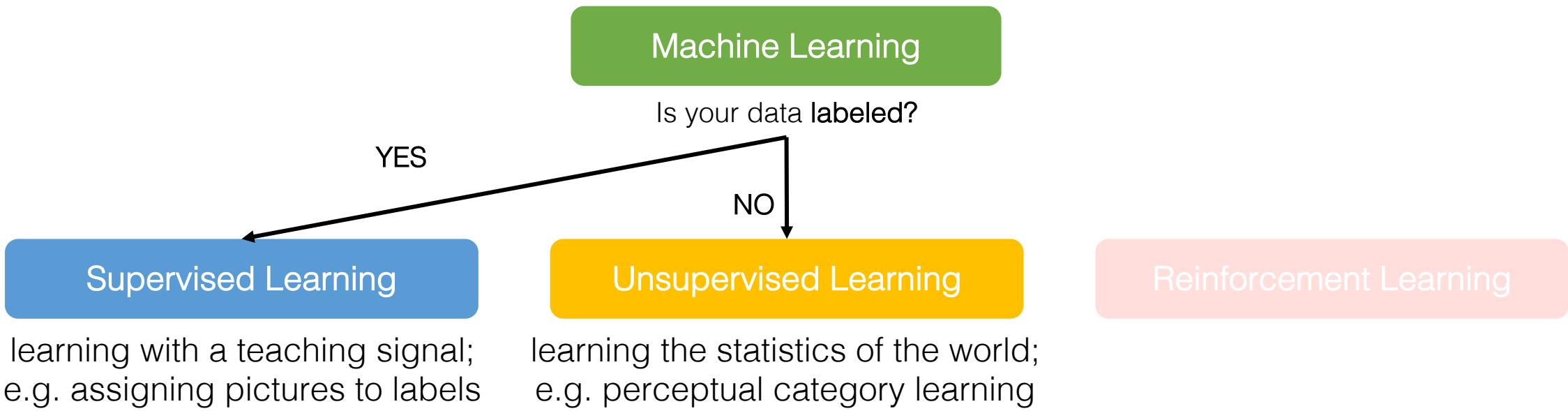


learning with a teaching signal;  
e.g. assigning pictures to labels

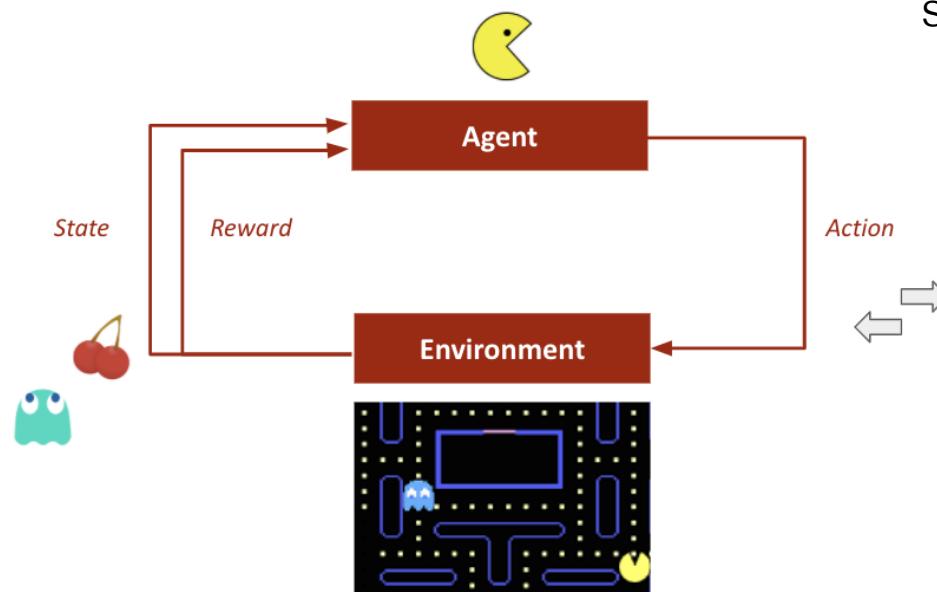
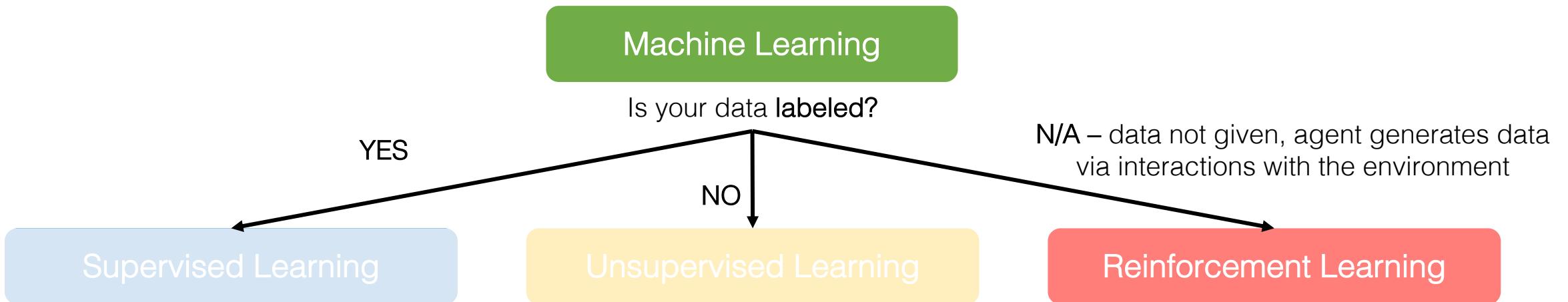
learning the statistics of the world;  
e.g. perceptual category learning



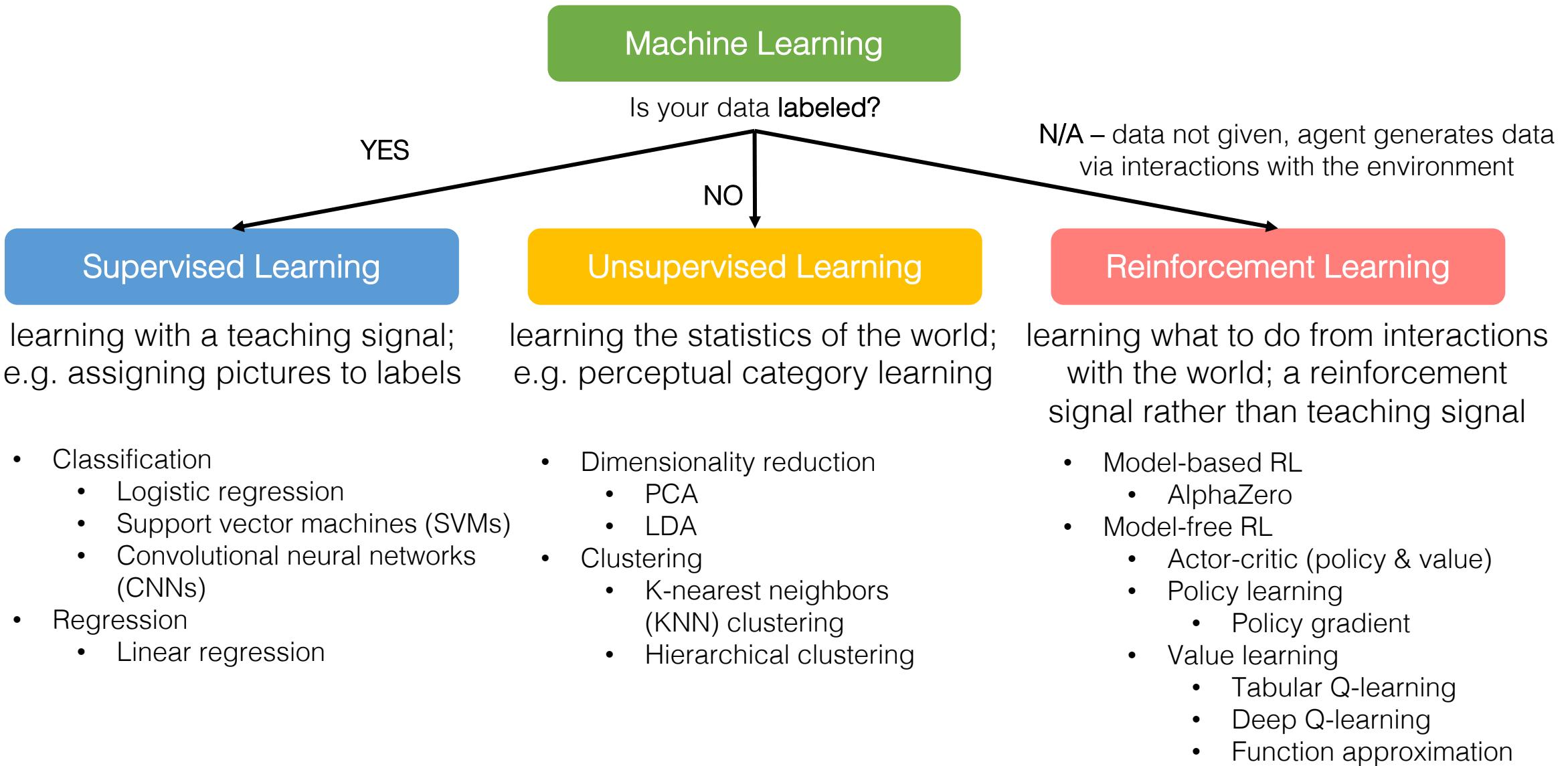
# Three types of learning

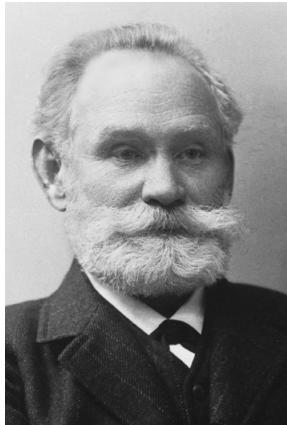


# Three types of learning



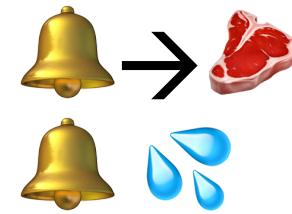
# Three types of learning





# Pavlovian conditioning

Ivan Pavlov



pair stimulus with some significant event   
measure anticipatory behavior

Terminology:

Unconditional stimulus (US)

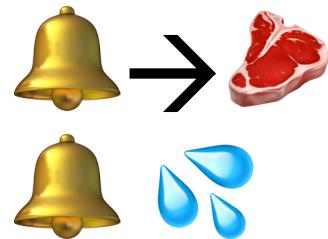
Conditional stimulus (CS)

Conditional response (CR)

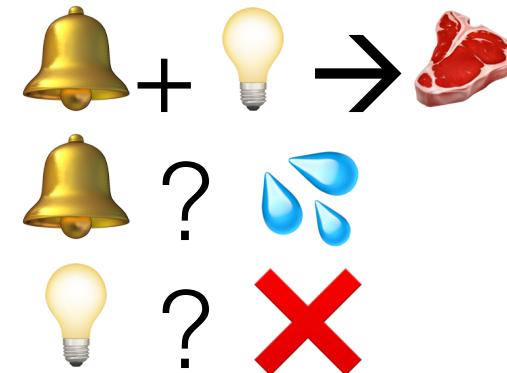
## Kamin's blocking

- What happens when you pair another predictor with the US after animal has already been conditioned to one CS?

Phase 1



Phase 2



- First CS blocks acquisition of the second CS!
- CS-US pairing is not enough, also need surprise!

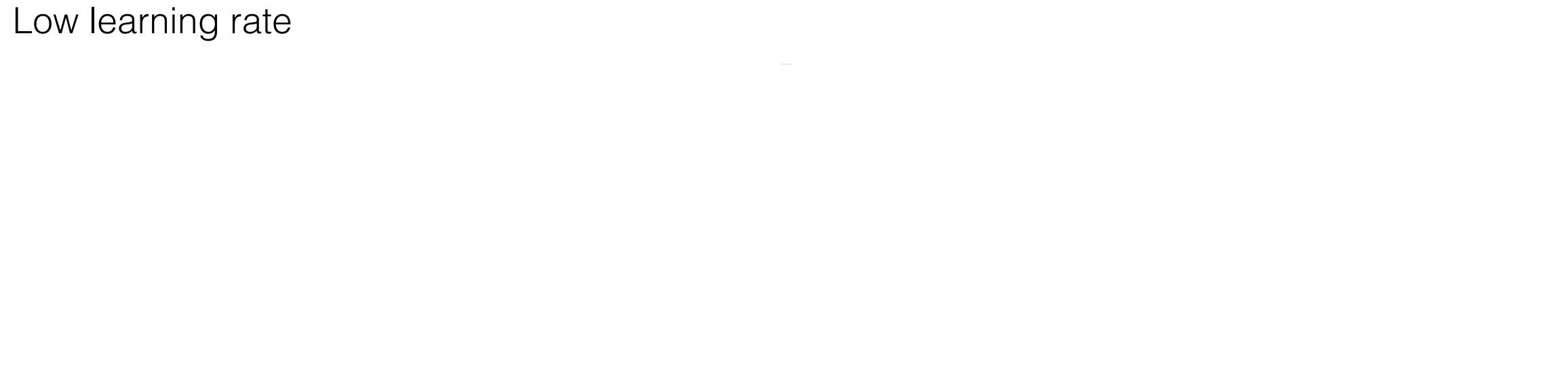
## Rescorla-wagner rule

- Core idea: an animal only learns when surprised, i.e., when events violate its expectations
- The change in the value of a CS (🔔) is proportional to the difference between the value of the US (🥩) and value predicted by the CS (🔔, can be multiple!), a.k.a. the *prediction error*.

$$\Delta V(CS_j) = \alpha(R_{US} - \sum_{i \in \text{trial}} V(CS_i))$$

- Two assumptions/hypotheses:
  - (1) learning is driven by *error* (formalizes the notion of surprise)
  - (2) prediction error driven by *summation* of predictors
- What happens to the *prediction error* after a couple iterations of this?
- What happens to the value if the learning rate  $\alpha$  is high? If  $\alpha$  is low?

Low learning rate



—

High learning rate

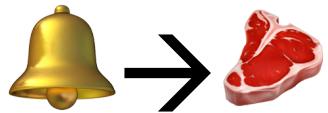
Rescorla-Wagner can model blocking experiments:

## Summary so far

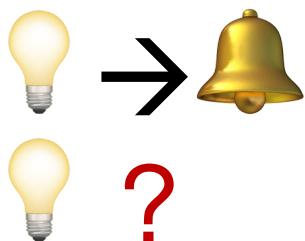
- Rescorla-Wagner learning suggests that we **learn from prediction errorss**
- In this framework, **learning = erasing previous beliefs**
- Slow learning is not necessarily bad!!!
- We can think of the “learning rate” as an important factor determining how we **balance old and new information**

## But...second-order conditioning

Phase 1



Phase 2

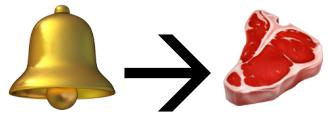


$$\Delta V(CS_j) = \alpha(R_{US} - \sum_{i \in \text{trial}} V(CS_i))$$

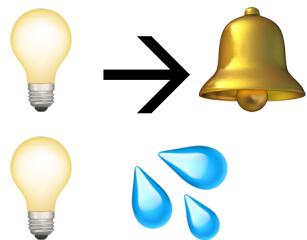
- What does the Rescorla-Wagner model predict?
  - A. animals will salivate to the light
  - B. animals will not salivate to the light

# But...second-order conditioning

Phase 1



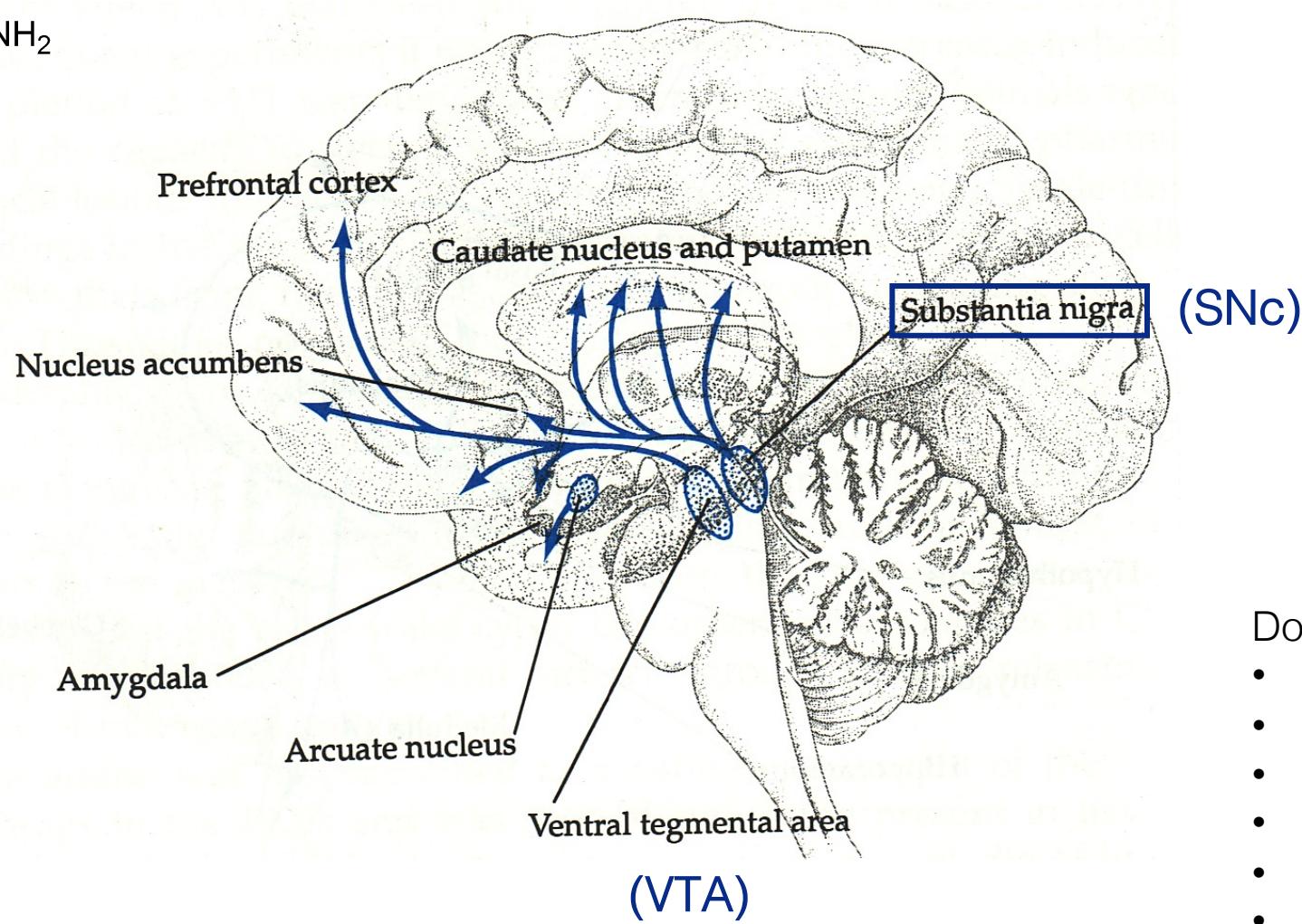
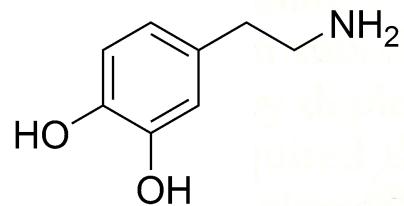
Phase 2



**Second-order conditioning:** a neutral conditional stimulus (CS) acquires the ability to elicit a conditioned response (CR) without ever being directly paired with an unconditioned stimulus (US)

- What do YOU think actually happens?
  - A. animals will salivate to the light
  - B. animals will not salivate to the light

# Another puzzle...dopamine

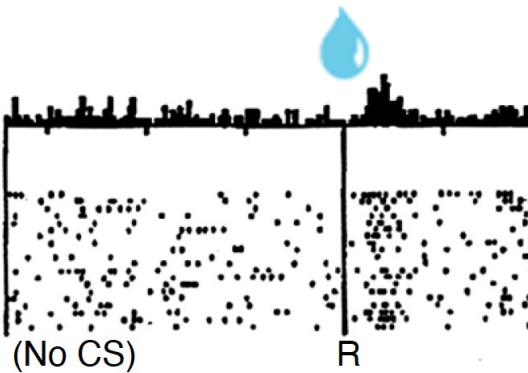


- Dopamine plays a role in:
- motivation
  - learning
  - action-selection
  - addiction
  - schizophrenia
  - Parkinson's disease
  - etc...

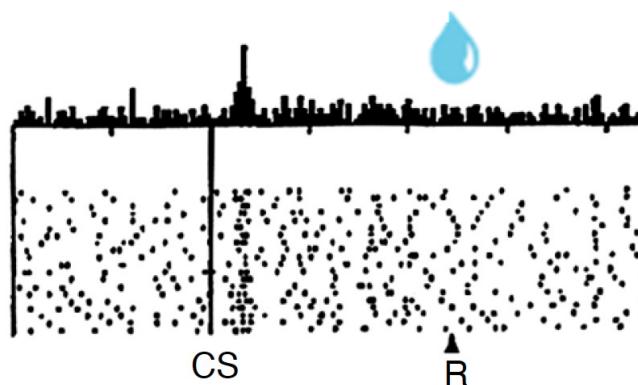
## Another puzzle...dopamine

CS              R  
cue → 

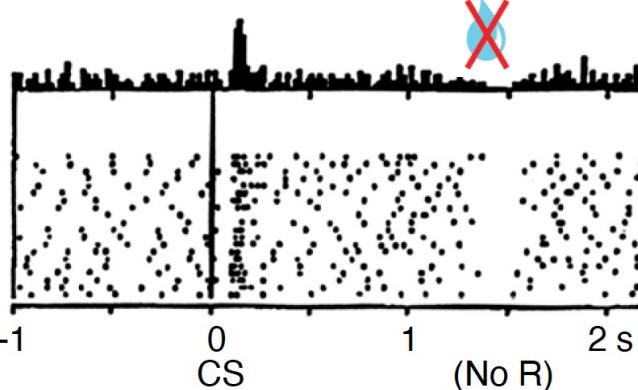
No prediction  
Reward occurs



Reward predicted  
Reward occurs



Reward predicted  
No reward occurs



Dopamine responds **to reward-predicting stimuli** instead of to rewards themselves

So...two puzzles

Behavioral puzzle: second order conditioning

Neural puzzle: dopamine responds to reward-predicting stimuli instead of to rewards

**How to solve these puzzles?**

# TD learning

- **Core idea:** Temporal difference learning extends the ideas from Rescorla-Wagner learning from immediate rewards to **all rewards in the future**.

$$\text{Rescorla-Wagner: } \Delta V(CS_j) = \alpha[R_{US} - \sum_{i \in \text{trial}} V(CS_i)]$$

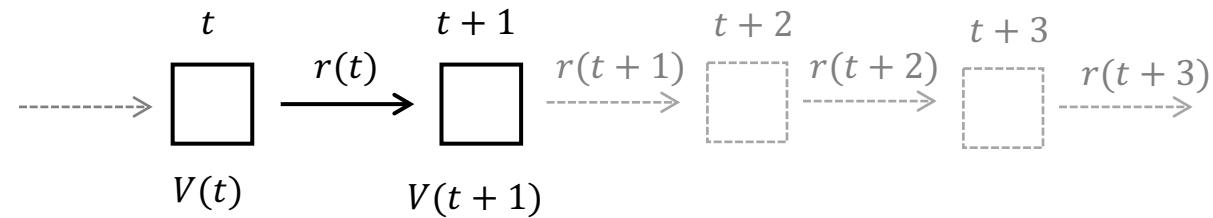
$$\text{TD learning: } \Delta V(t) = \delta = \alpha[R(t) + \gamma \cdot \hat{V}(t+1) - \hat{V}(t)]$$

Value (at the current point in time):      Predicted reward  
the *discounted* sum of all future rewards

- New idea: a **discount factor  $\gamma$**  that tells you how much to care about **rewards in the future**

# TD learning

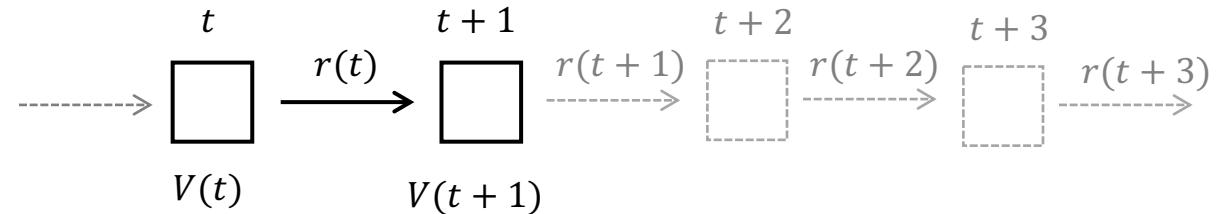
States



$t$  : time  
 $r$  : reward  
 $V$  : value

# TD learning

States



$t$  : time  
 $r$  : reward  
 $V$  : value

Value: the discounted sum of all future rewards

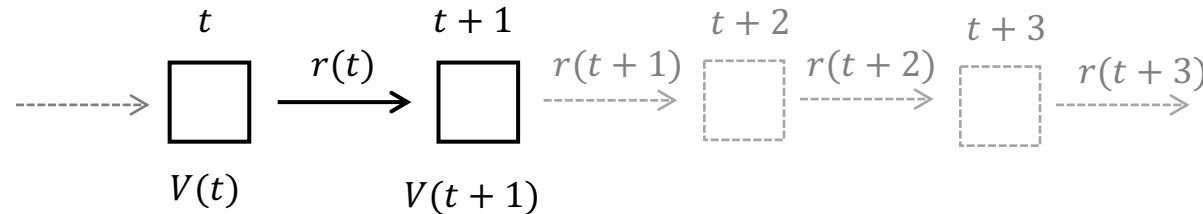
Discount factor

$$0 < \gamma < 1$$

$$V(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

# TD learning

States



$t$  : time  
 $r$  : reward  
 $V$  : value

Value: the discounted sum of all future rewards

Discount factor

$$V(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r(t) + \boxed{\gamma \cdot r(t+1) + \gamma^2 \cdot r(t+2) + \dots}$$

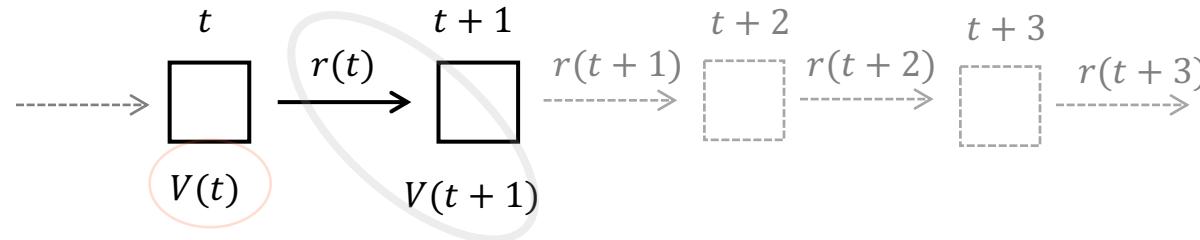
$0 < \gamma < 1$

$$\gamma \cdot V(t+1)$$

$$V(t) = r(t) + \gamma \cdot V(t+1)$$

# TD learning

States



$t$  : time  
 $r$  : reward  
 $V$  : value

Value: the discounted sum of all future rewards

Discount factor

$$V(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r(t) + \gamma \cdot r(t+1) + \gamma^2 \cdot r(t+2) + \dots$$

$0 < \gamma < 1$

$$\gamma \cdot V(t+1)$$

$$\underline{V(t)} = r(t) + \gamma \cdot \underline{V(t+1)}$$

Temporal difference (TD) prediction error

$$\delta = \underline{r(t) + \gamma \cdot \hat{V}(t+1)} - \underline{\hat{V}(t)} \quad \Leftrightarrow \quad \text{Update } V(t)$$

$$\hat{V}(t) \leftarrow \hat{V}(t) + \alpha \cdot \delta \quad (\alpha: \text{learning rate})$$

Discount factor  $\gamma$  determines how much you care about future reward

When there is no anticipating the future ( $\gamma = 0$ ), second order conditioning doesn't happen:

---

Discount factor  $\gamma$  determines how much you care about future reward

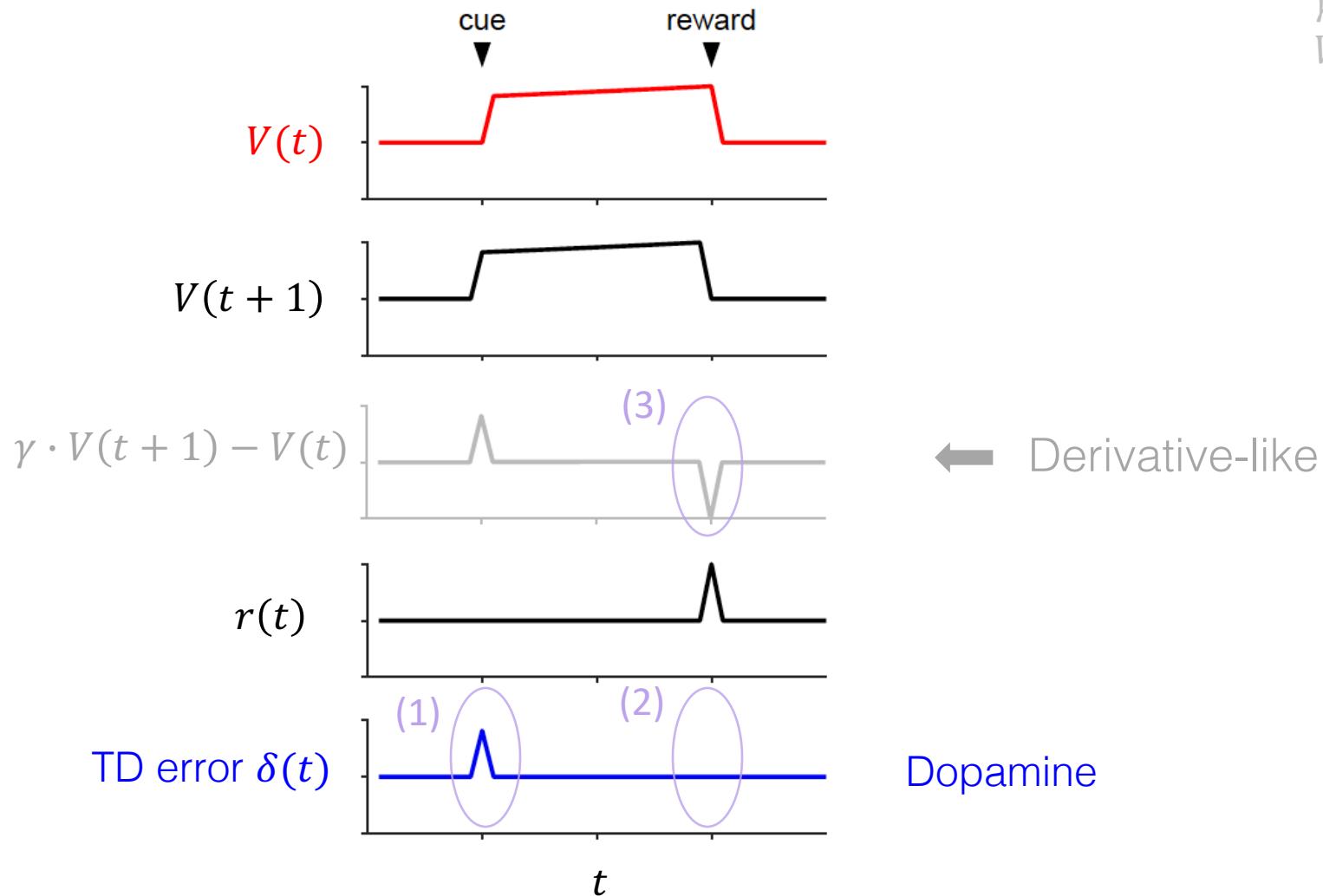
When anticipating *eventual*/reward ( $\gamma = 1$ ), second order conditioning *does* happen:

---

# Dopamine responses as TD errors

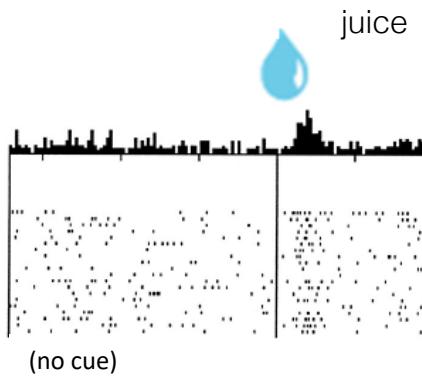
$$\delta(t) = r(t) + \gamma \cdot V(t + 1) - V(t)$$

$r(t)$ : reward  
 $\gamma$ : discount factor  
 $V(t)$ : Value

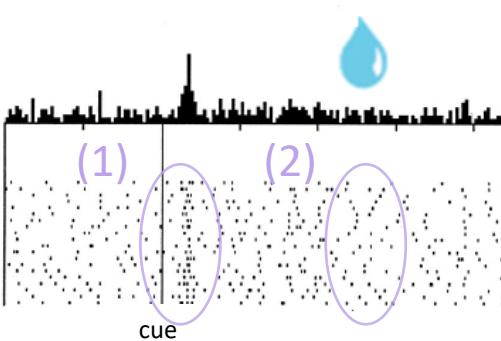


# Dopamine responses as TD errors

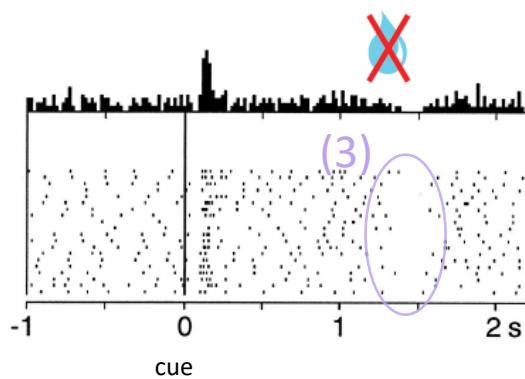
No prediction  
Reward occurs



Cue predicts reward  
Reward occurs

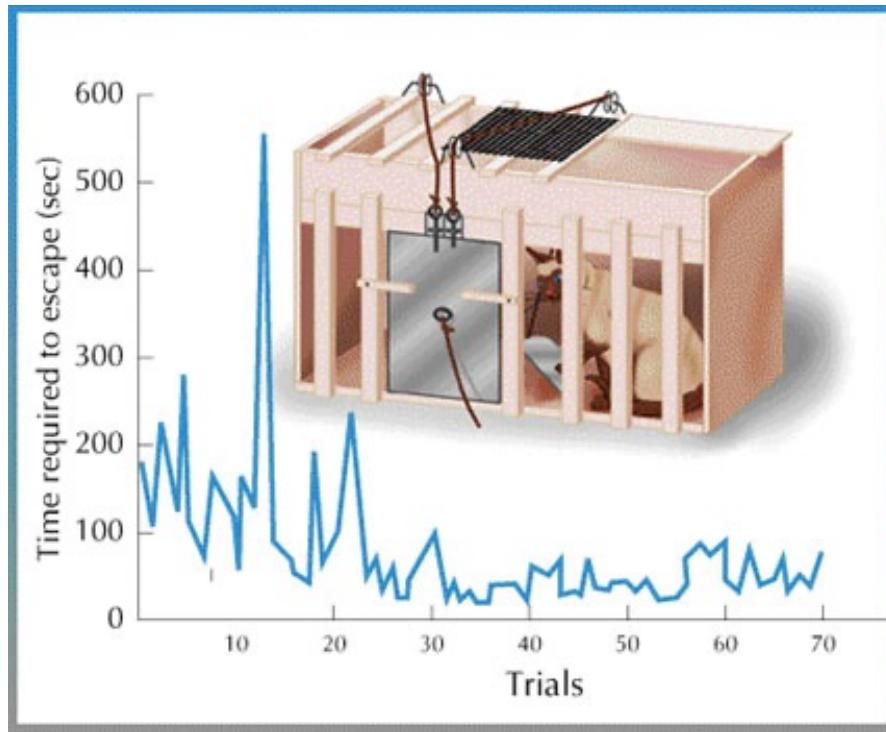


Cue predicts reward  
No reward occurs

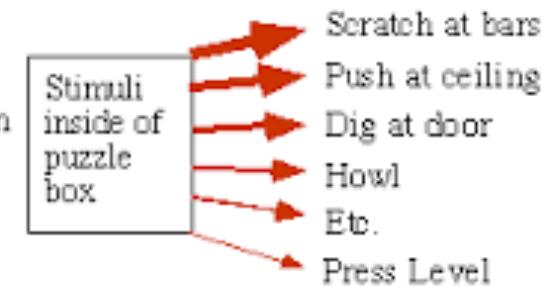


# Action selection

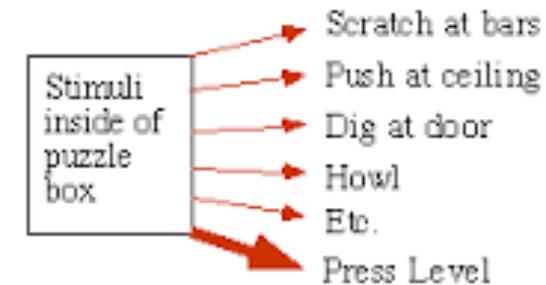
- Prediction learning is not enough by itself, need something that helps you change behavior!
- **Instrumental learning:** Thorndike's Law of Effect



First trial in  
puzzle box

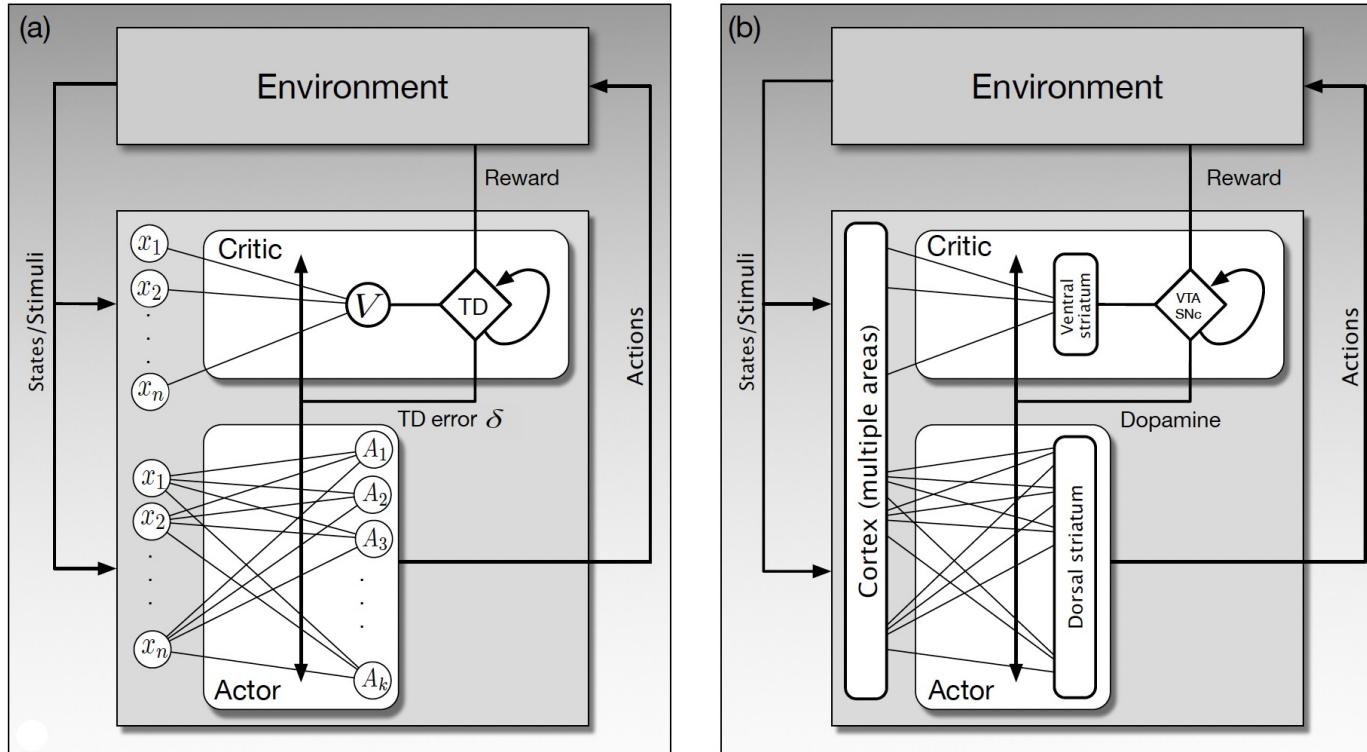


After many  
trials in  
puzzle box



- Successful actions will be reinforced!

# Actor-critic models



“Actor” learns policies (mappings from state to actions)

“Critic” evaluates whether the actions that the actor chose were high or low in value

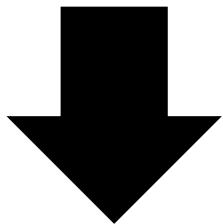
TD error updates both state and action values

Dorsal striatum: implicated in influencing action selection

Ventral striatum: reward processing and assigning affective value to sensory input

Dopamine modulates synaptic plasticity in both the dorsal and ventral striatum

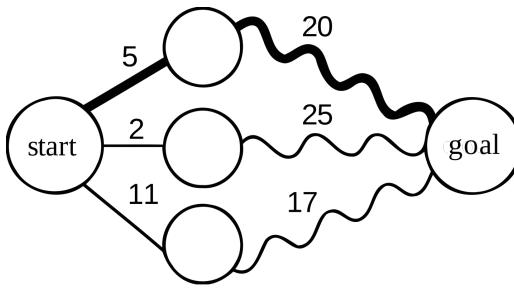
RL in neuroscience



RL in machine learning

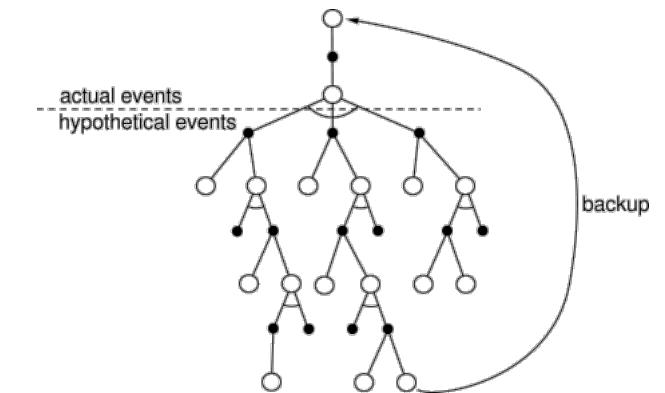
# Historical threads

Optimal control theory



Dynamic  
programming:  
Bellman (~1950s)

Trial-and-error learning



e.g. Samuel's  
checker-playing  
program  
(~1959)



Sutton ~ 1988:  
First  
systematic  
treatment of  
TD methods

Watkins 1989: Q-learning

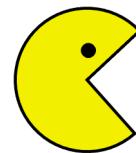
Tesauro 1992: TD-Gammon

Temporal difference  
learning, Q-learning...

Basic RL: problem formulation



observe state



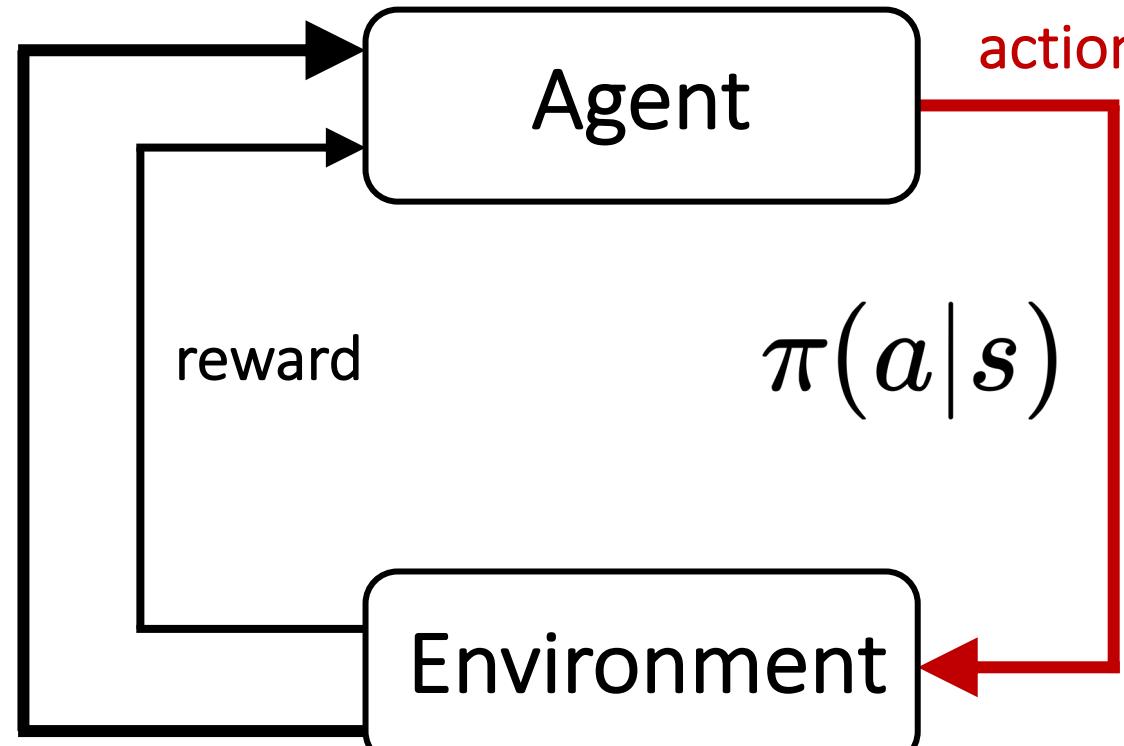
### State:

current status of environment  
(e.g., location of agent or obstacles, “internal state”)



### Reward:

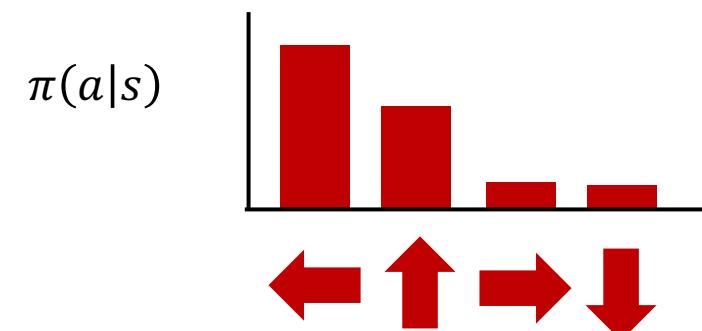
feedback from environment;  
agent tries to maximize this



how an agent **interacts** with its environment

### Policy:

the probability of taking each action in a given state



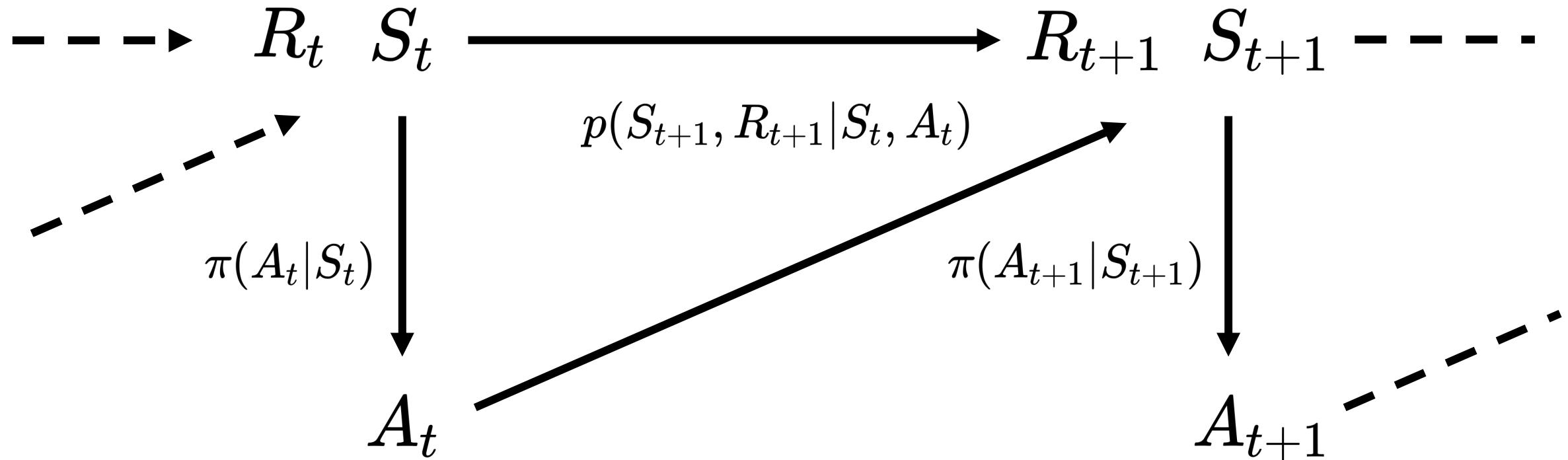
### Model of environment (optional)

:

knowledge of how your actions will bring you to rewards and other states (as opposed to just treating each state



# The math setting of RL: Markov decision processes



Key assumption: what you do, and what happens next, only depends on the **current state**, and not on previous ones.

The goal of RL: maximize expected (discounted) future reward

- **Return:** discounted future reward

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

- **State value:** expected return, given a specific starting state and policy

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

- **Action value:** expected return, given a specific starting state, action, and policy

$$q_\pi(s, a) := \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \quad \text{Q-value function}$$

The goal of RL: maximize expected (discounted) future reward

- **Return:** discounted future reward

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

- Prediction problem: How much reward will we get if we follow a given policy?

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

- Control problem: What's the 'best' policy?

Target:  $\pi_*(a|s)$   $v_{\pi_*}(s) \geq v_\pi(s)$

not unique in general

Basic RL: solving the prediction problem

# The Bellman equation and dynamic programming

- Prediction problem: How much reward will we get if we follow a given policy?
- Useful fact: the value function satisfies the **Bellman equation**:

$$v_{\pi}(s) = \sum_{s',r,a} p(s', r | s, a) \pi(a | s) [ r + \gamma v_{\pi}(s') ]$$

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

- This equation is the basis of an iterative approach to evaluating a policy.

$$V_{\pi}(s) \leftarrow \sum_{s',r,a} p(s', r | s, a) \pi(a | s) [ r + \gamma V_{\pi}(s') ]$$

# The TD algorithm for evaluating the goodness of a policy

- Dynamic programming requires knowing environment's dynamics... generally not the case.
- But it inspires an online, model-free prediction algorithm:  
**temporal difference (TD) learning!**
- The TD algorithm lets you evaluate a policy in real time, by trial and error.

# The TD algorithm for evaluating the goodness of a policy

In state  $S$ , take action  $A$  according to policy  $\pi(A | S)$ .



Transition to  $S'$  and collect reward  $R$ .



$$\delta := R + \gamma V_\pi(S') - V_\pi(S)$$

$$V_\pi(S) \leftarrow V_\pi(S) + \alpha \delta$$

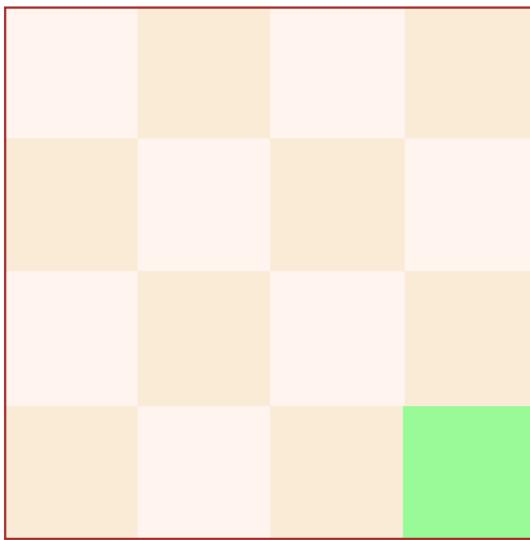
Prediction

error  
Value update

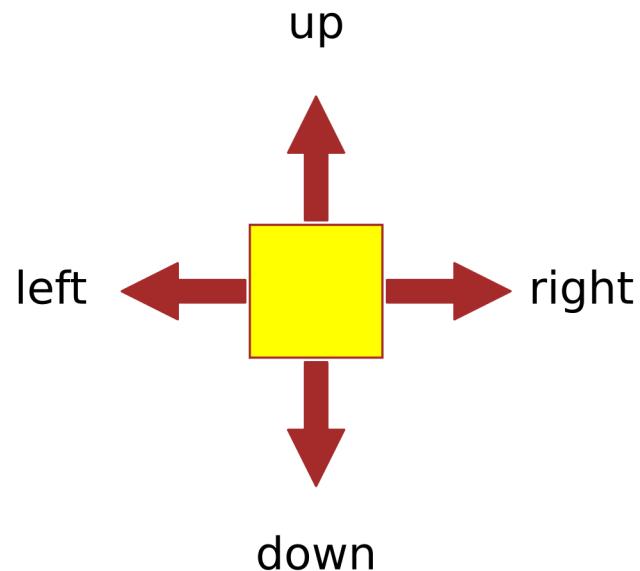
$$S \leftarrow S'$$

# A gridworld example

Environment



Possible actions



Possible rewards

Reach non-goal square:



$$R = -1$$

Reach goal square:

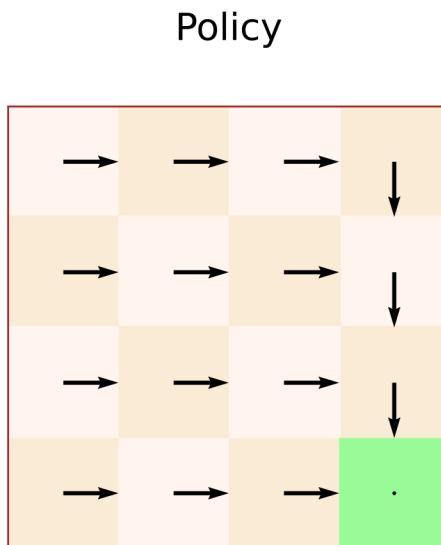


$$R = 10 - 1$$

Initial grid location is random

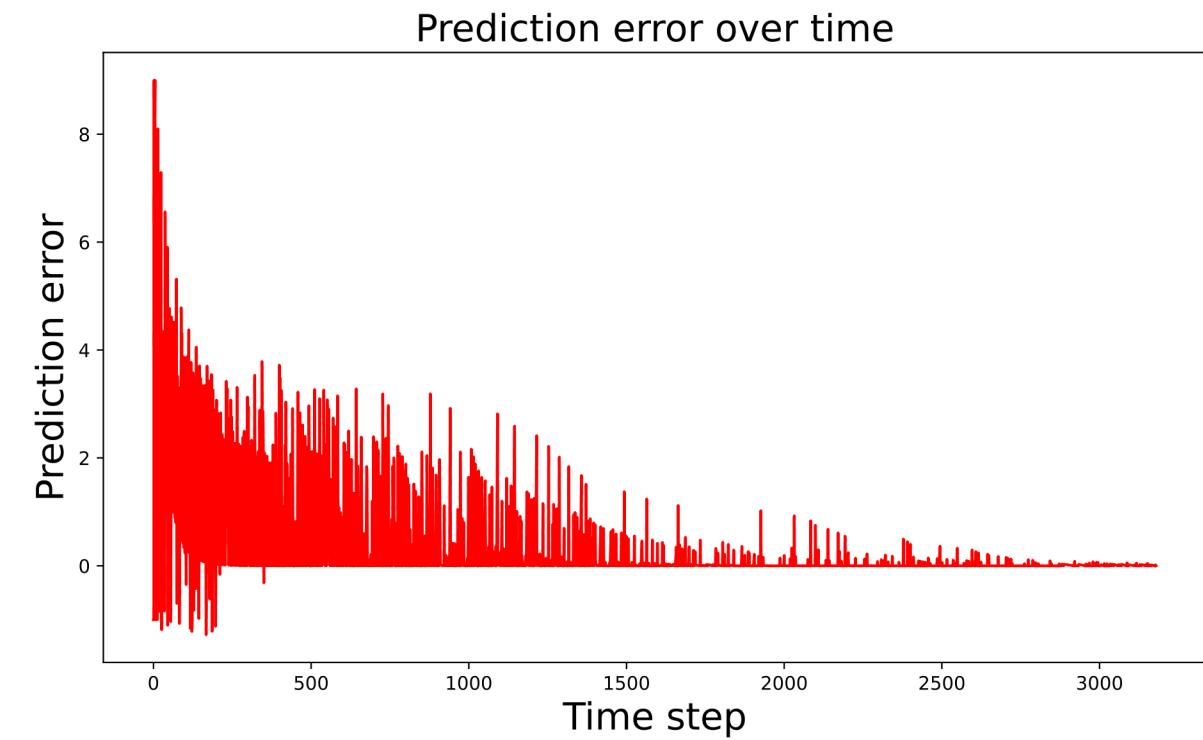
# A gridworld example: evaluating the ‘down-right’ policy

Using the TD algorithm (1000 episodes):



Value function

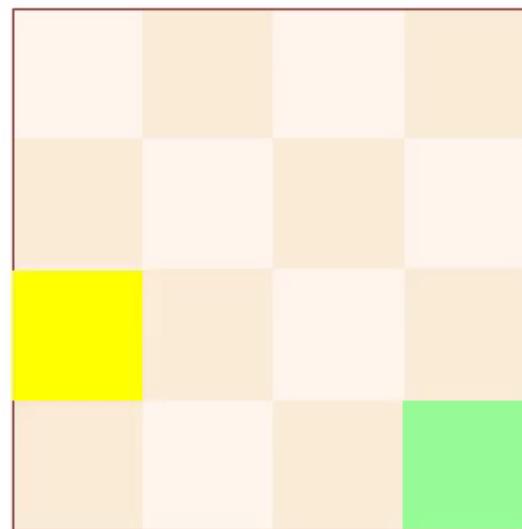
3.98	5.0	6.0	7.0
4.99	6.0	7.0	8.0
5.96	7.0	8.0	9.0
6.98	8.0	9.0	.



# A gridworld example: evaluating the ‘down-right’ policy

Using the TD algorithm (1000 episodes):

Episode 1



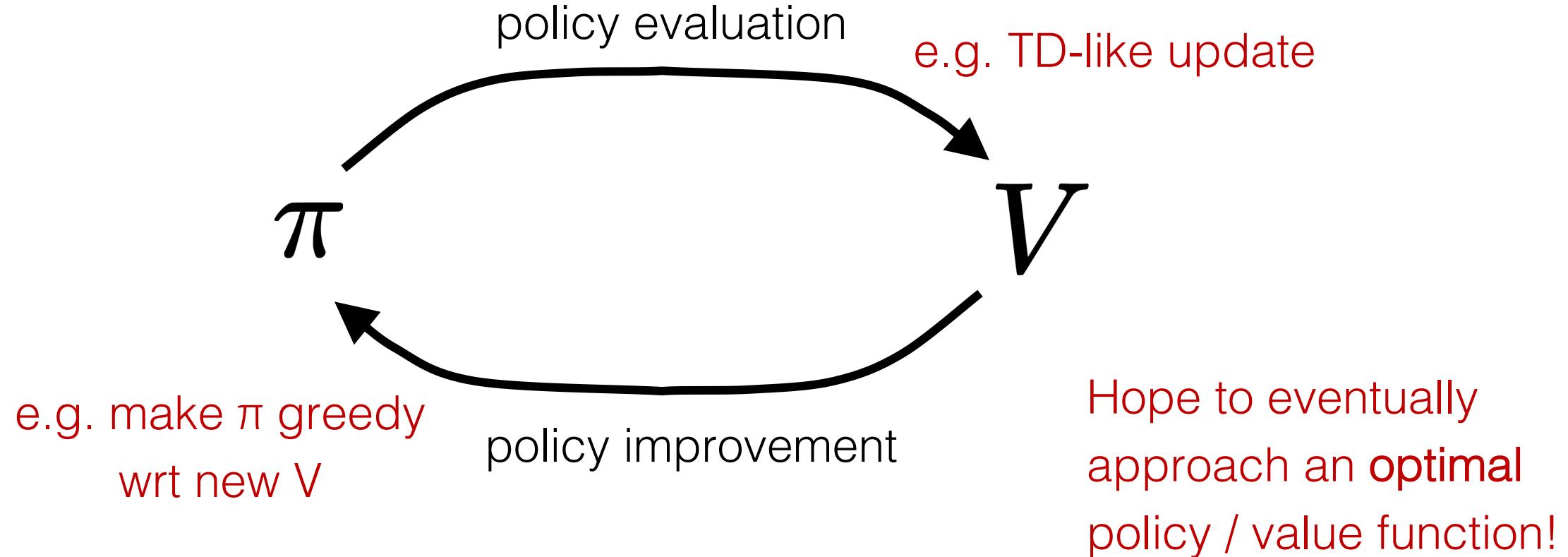
Basic RL: solving the control problem

# Generalized policy iteration and the RL control problem

- Two things you don't know in general:
  - (i) how much reward you're going to get given your current policy
  - (ii) what the best policy is.
- Idea of **generalized policy iteration**: switch between improving both by a little!

# Generalized policy iteration and the RL control problem

- Idea of generalized policy iteration: switch between improving both by a little!



Two control strategy examples:  
Q-learning and actor-critic

## Q-learning: TD policy evaluation + greedy action selection

$$Q(s, a) \approx q_*(s, a) = \max_{\pi} \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

- Suppose you know  $Q(s, a)$ , an estimate of the optimal action-value function.
- Suppose you're in a state  $s$ . Could turn this into a control strategy by always picking the action  $a$  for which  $Q(s, a)$  is largest (with ties broken arbitrarily).

# Q-learning: TD policy evaluation + greedy action selection

$$Q(s, a) \approx q_*(s, a) = \max_{\pi} \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

- But if your Q estimate isn't perfect, may not truly know which actions are best.
- One possibility: couple a strategy like this with a **method for exploration**. Sometimes take actions you wouldn't normally take, and see if they give the expected reward. If so, great. If not, update Q!
- **Epsilon-greedy**: pick 'best' action with probability  $1 - \varepsilon$ ; otherwise random.

# Q-learning: TD policy evaluation + greedy action selection

In state  $S$ , take action  $A$  according to policy  $\pi(A | S)$ .



Transition to  $S'$  and collect reward  $R$ .



$$\delta := R + \gamma \max_a Q(S', a) - Q(S, A)$$

$$Q(S, A) \leftarrow Q(S, A) + \alpha \delta$$

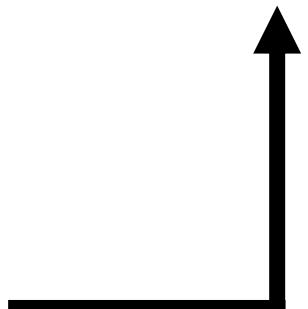
$S \leftarrow S'$

**Update policy**

e.g.  $\epsilon$ -greedy wrt updated  $Q$ .

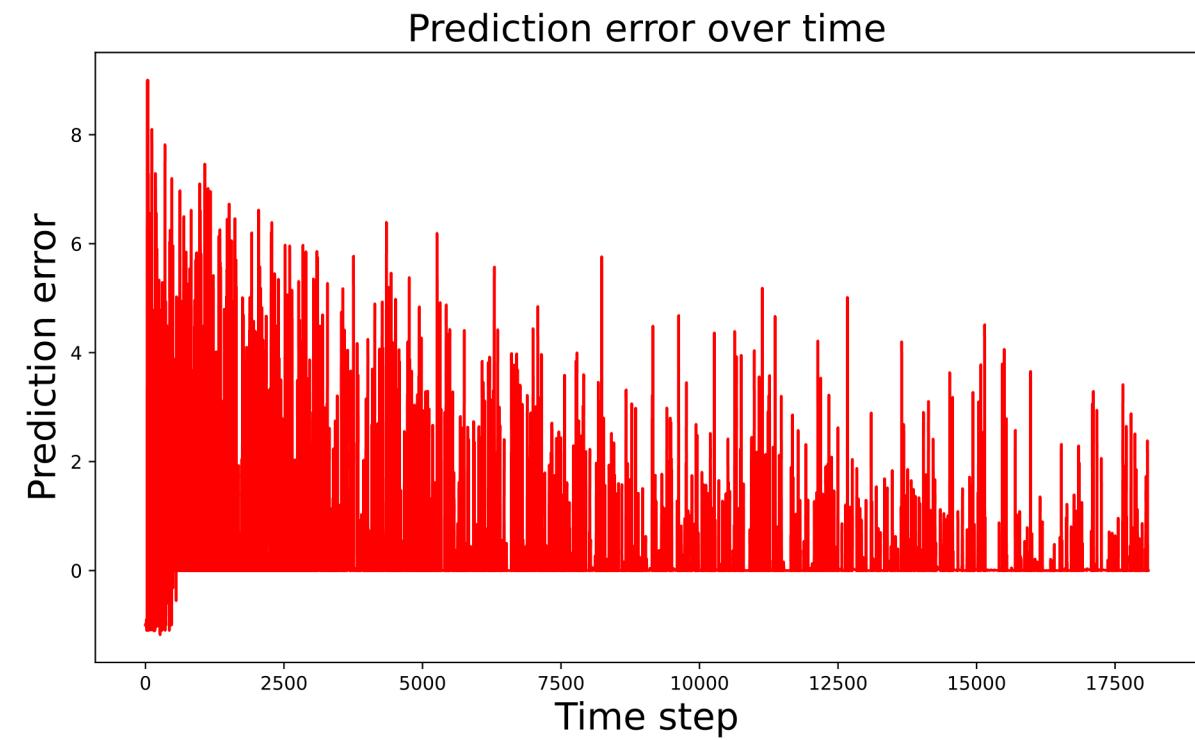
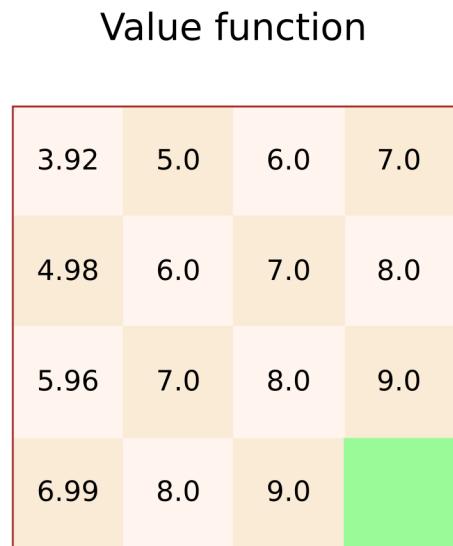
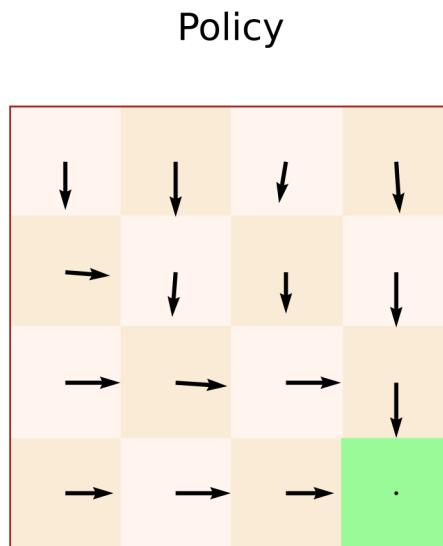
Prediction  
error

Value update



# A gridworld example: learning a good policy

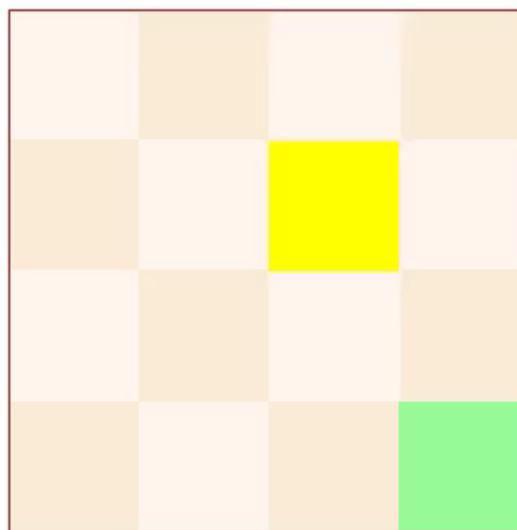
Using Q-learning (5000 episodes):



# A gridworld example: learning a good policy

Using Q-learning (5000 episodes):

Episode 1



# Actor-critic: TD policy evaluation + action preferences

- One problem with Q-learning: policy (e.g. epsilon-greedy) closely linked to Q.
- Greedy is *eventually* optimal, but not while learning...
- Either have to accept randomly taking bad actions, or determining a (possibly complicated) schedule for reducing exploration over time.
- **Idea for alternative**: separate value and policy, and learn both simultaneously!

# Actor-critic: TD policy evaluation + action preferences

- **Idea:** separate value and policy, and learn both simultaneously!

$$\pi(a|s) = \frac{e^{h(s,a)}}{\sum_b e^{h(s,b)}}$$

Want to learn “action preferences”  
 $h(s, a)$  in addition to  $V(s)$ .

- How to update? **Gradient descent** (using value as objective function).
- Intuitive idea: if you took an action and things went better than expected, take that action more often!

# Actor-critic: TD policy evaluation + action preferences

In state  $S$ , take action  $A$  according to policy  $\pi(A | S)$ .



Transition to  $S'$  and collect reward  $R$ .



$$\delta := R + \gamma V_\pi(S') - V_\pi(S)$$

Prediction  
error

$$V_\pi(S) \leftarrow V_\pi(S) + \alpha \delta$$

Value update

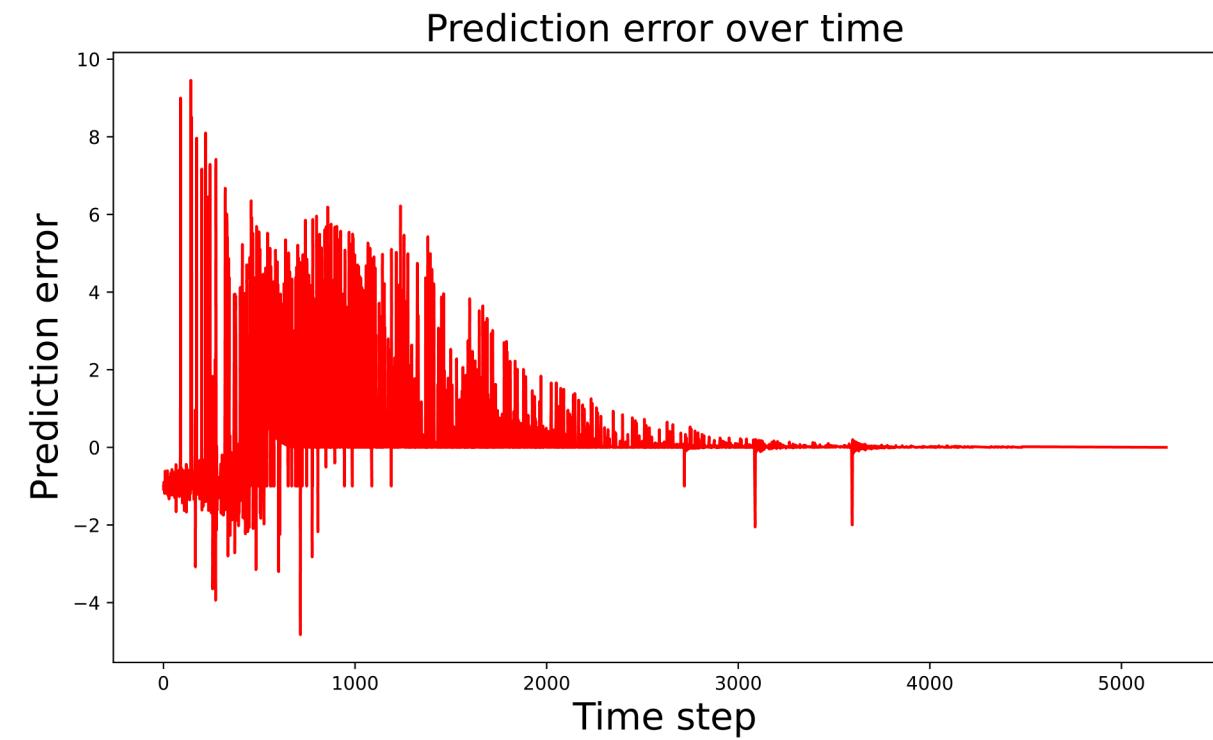
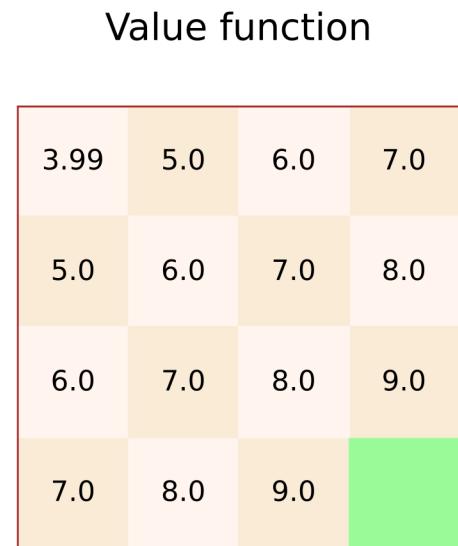
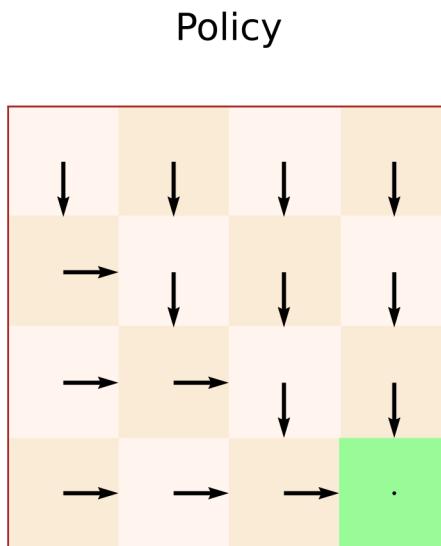
$$S \leftarrow S'$$

$$h(S, A) \leftarrow h(S, A) + \alpha \delta \frac{\nabla_{h(S)} \pi(A|S)}{\pi(A|S)}$$



# A gridworld example: learning a good policy

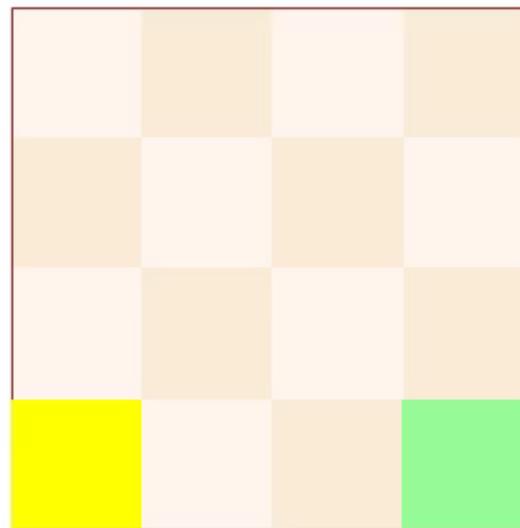
Using actor-critic (1500 episodes):



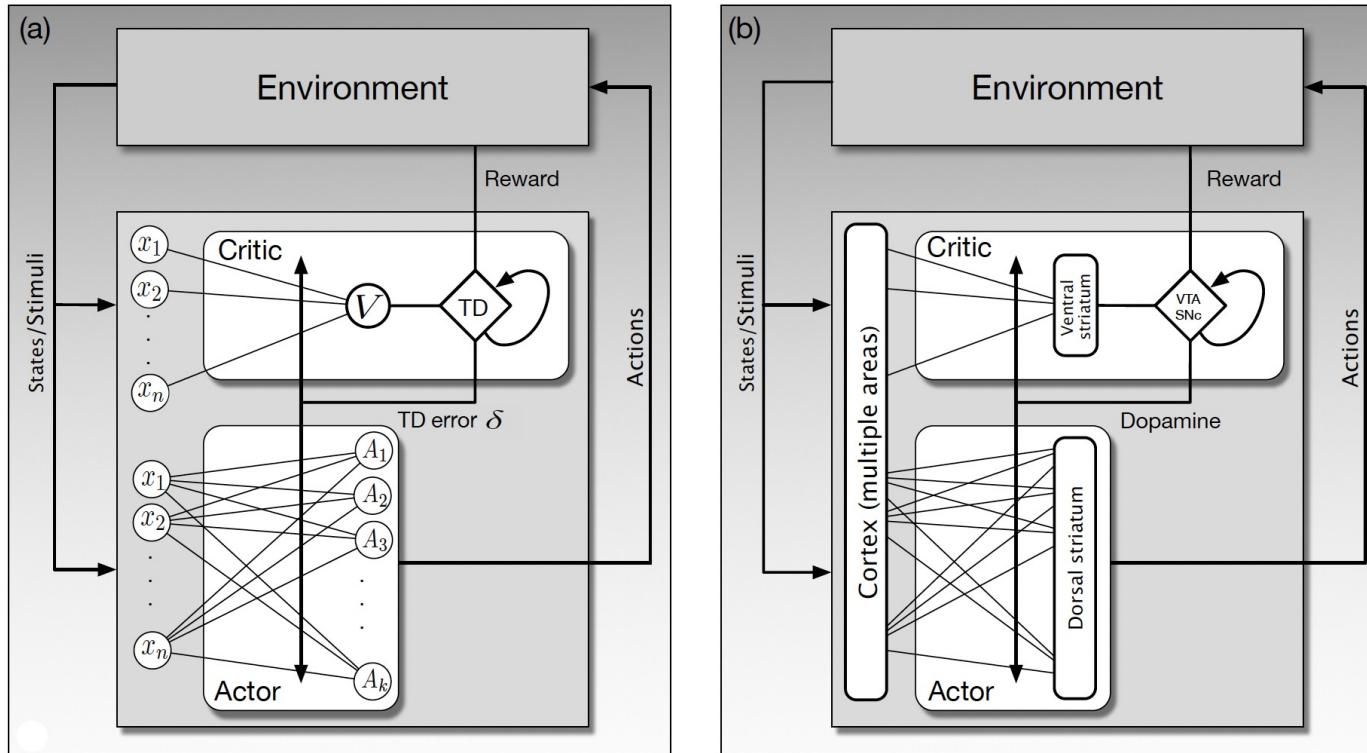
# A gridworld example: learning a good policy

Using actor-critic (1500 episodes):

Episode 1



# Actor-critic models



“Actor” learns policies (mappings from state to actions)

“Critic” evaluates whether the actions that the actor chose were high or low in value

TD error updates both state and action values

Dorsal striatum: implicated in influencing action selection

Ventral striatum: reward processing and assigning affective value to sensory input

Dopamine modulates synaptic plasticity in both the dorsal and ventral striatum

Deep RL: approximation as a way to address complexity

# The curse of dimensionality



Earth  $\sim 10^{50}$  atoms



$\sim 10^{84}$  photons  
produced in history of  
universe



Chess  $\sim 10^{46}$  states



Go  $\sim 10^{170}$  states



Action space  
effectively  
continuous

Tabular RL requires  
separately keeping track of  
values/policies for every  
state/action...

Almost always impossible!

# Function approximation as a way to accommodate complexity

- The way out is obvious: just don't use a table to track values/policies.
- This means coming up with a **parameterized approximation** to values/policies.
- E.g. A set of parameters **w** determine the value of  $V(s)$  for all states  $s$ .
- Function approximation can be used for...

$$V(s) \rightarrow \hat{v}(s, \mathbf{w}) \quad \text{Value function}$$

$$\pi(a|s) \rightarrow \hat{\pi}(a|s, \theta) \quad \text{Policy}$$

$$Q(s, a) \rightarrow \hat{q}(s, a, \mathbf{w}) \quad \text{Q-value function}$$

# Function approximation as a way to accommodate complexity

- Can come up with analogues to all the important tabular RL algorithms, e.g.

TD prediction

$$\delta_t := R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

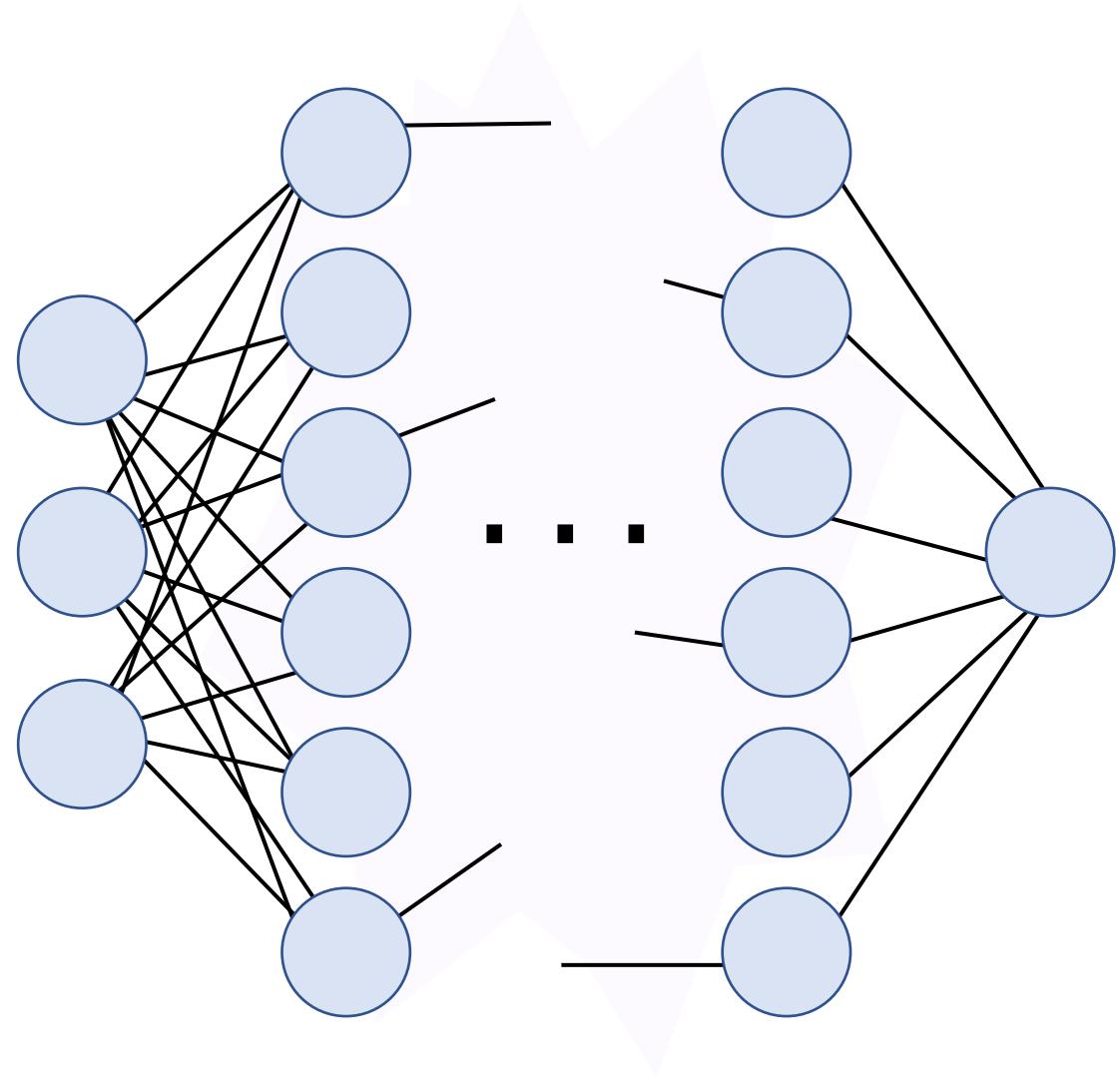
Q-learning  
control

$$\delta_t := R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

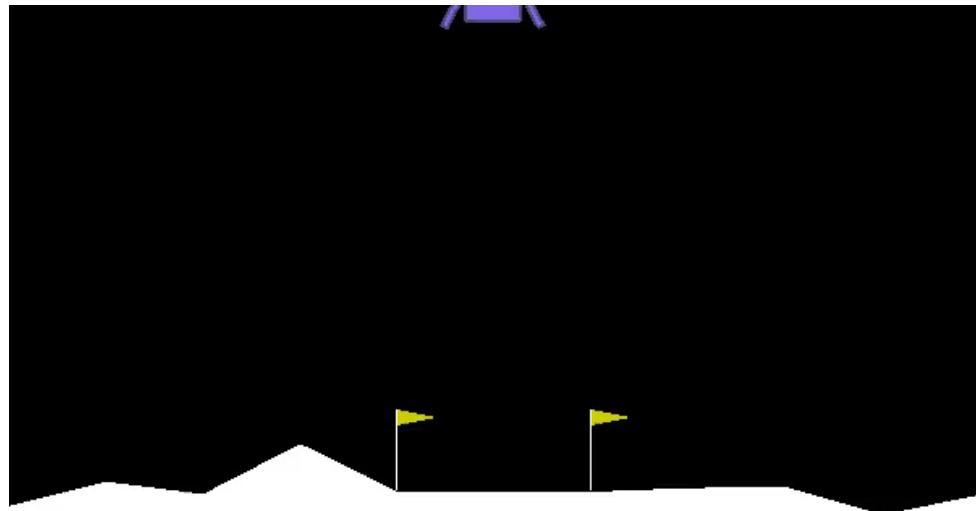
# Deep RL: function approximation via deep neural networks

- Want function approximators good enough that closely approximating any (e.g.)  $V(s)$  is plausible.
- **Neural networks** are a natural and convenient choice.
- Despite being the workhorse of almost all practical RL...  
*few theoretical guarantees so far!*

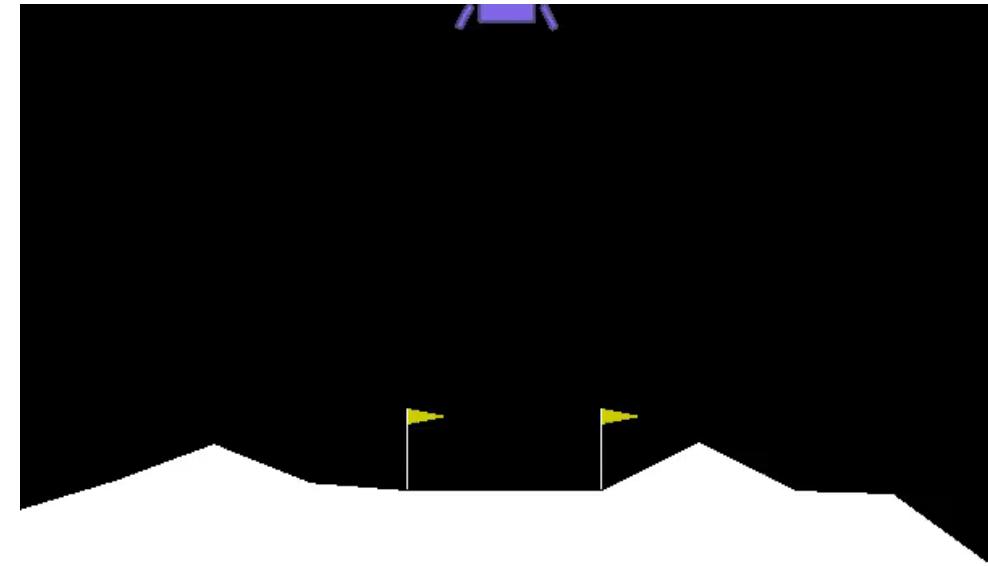


# Deep RL: function approximation via deep neural networks

Before training:



After training:

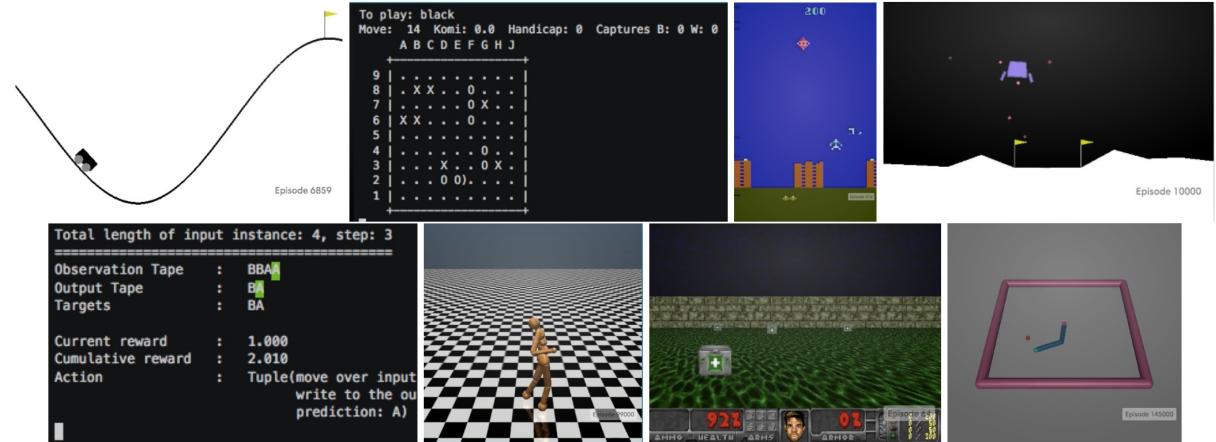


# OpenAI Gym and Stable Baselines

- Want to play around with RL yourself? Some recommended resources:

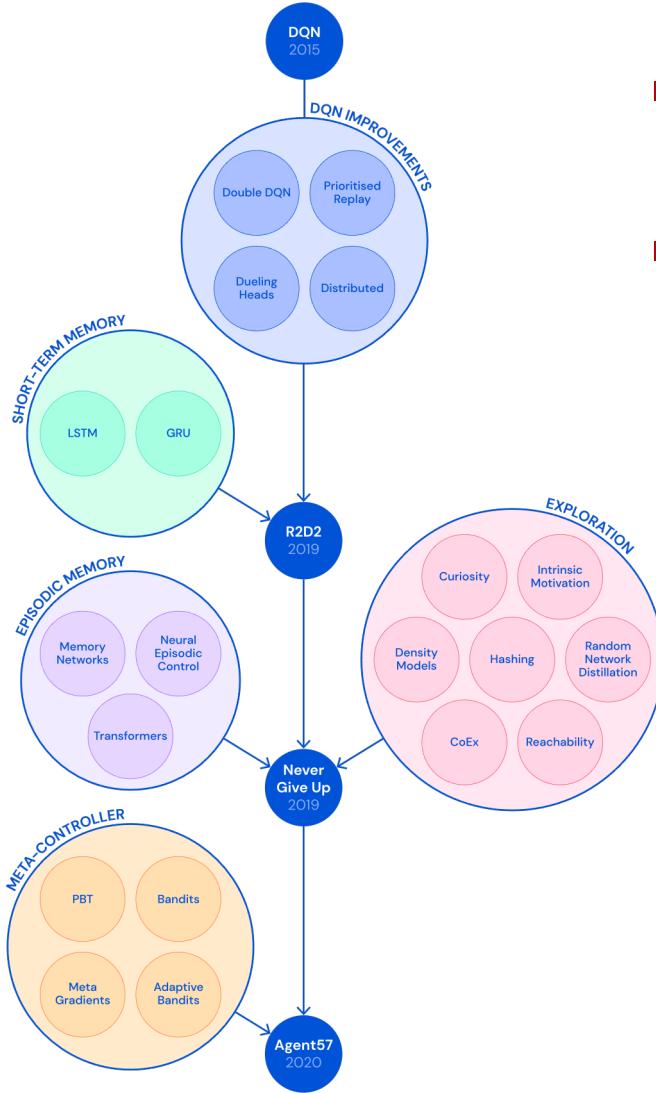
## OpenAI Gym ([gymlibrary.dev](https://gymlibrary.dev)):

Environments to play around with + convenient API for constructing your own.



Stable Baselines ([stable-baselines.readthedocs.io/en/master/](https://stable-baselines.readthedocs.io/en/master/)):  
Reference implementations of standard RL algorithms.  
(e.g. Deep Q networks, actor-critic, etc.)

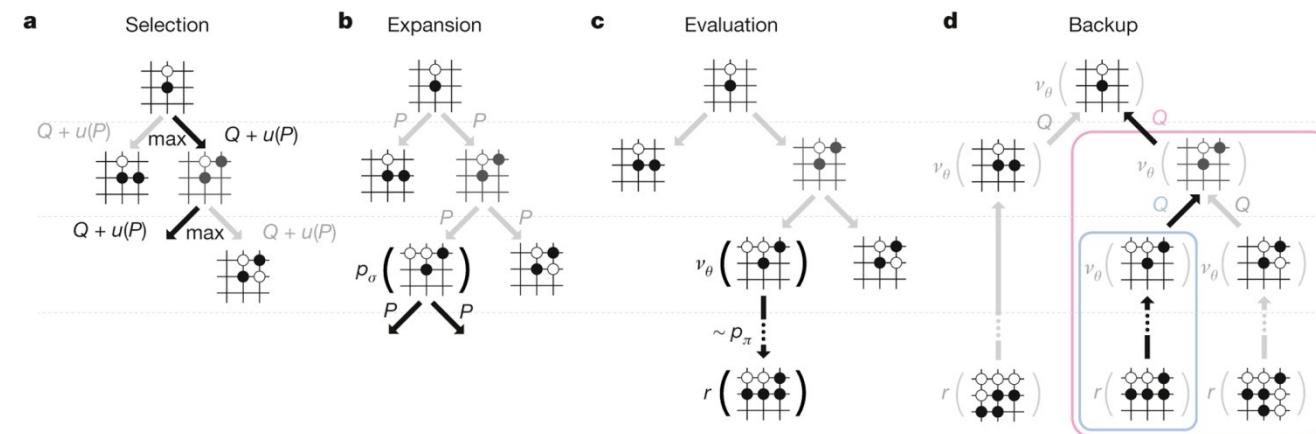
# RL in practice: hacks, choices, and approximations



- Exploration (try new things) vs exploitation (use what works)
- Model-free vs model-based
  - On-policy learning vs off-policy learning
- Memory mechanism
- Attention and prioritization mechanisms
- Hyperparameter tuning, expensive training using many GPUs...

# On-policy vs off-policy learning

- **Monte Carlo Tree Search (MCTS)**: a very successful off-policy learning technique, used to get e.g. Go AI from amateur to world-class level.



D Silver et al. *Nature* **529**, 484–489 (2016) doi:10.1038/nature16961

- **Idea:** With a model of the world, can mentally simulate how things might happen to reduce amount of trial and error necessary to learn.
- **In a slogan:** we don't need to walk into a wall to know that it's a bad idea!

# Frontiers in RL

- Reinforcement learning is a huge field with many open questions!

**Theoretical questions,**

e.g. What kind of theoretical guarantees can we provide for typical RL implementations?

**Scientific questions, e.g.**

If the brain is doing something like RL, what do its RL algorithms look like?  
Also, what is dopamine doing???

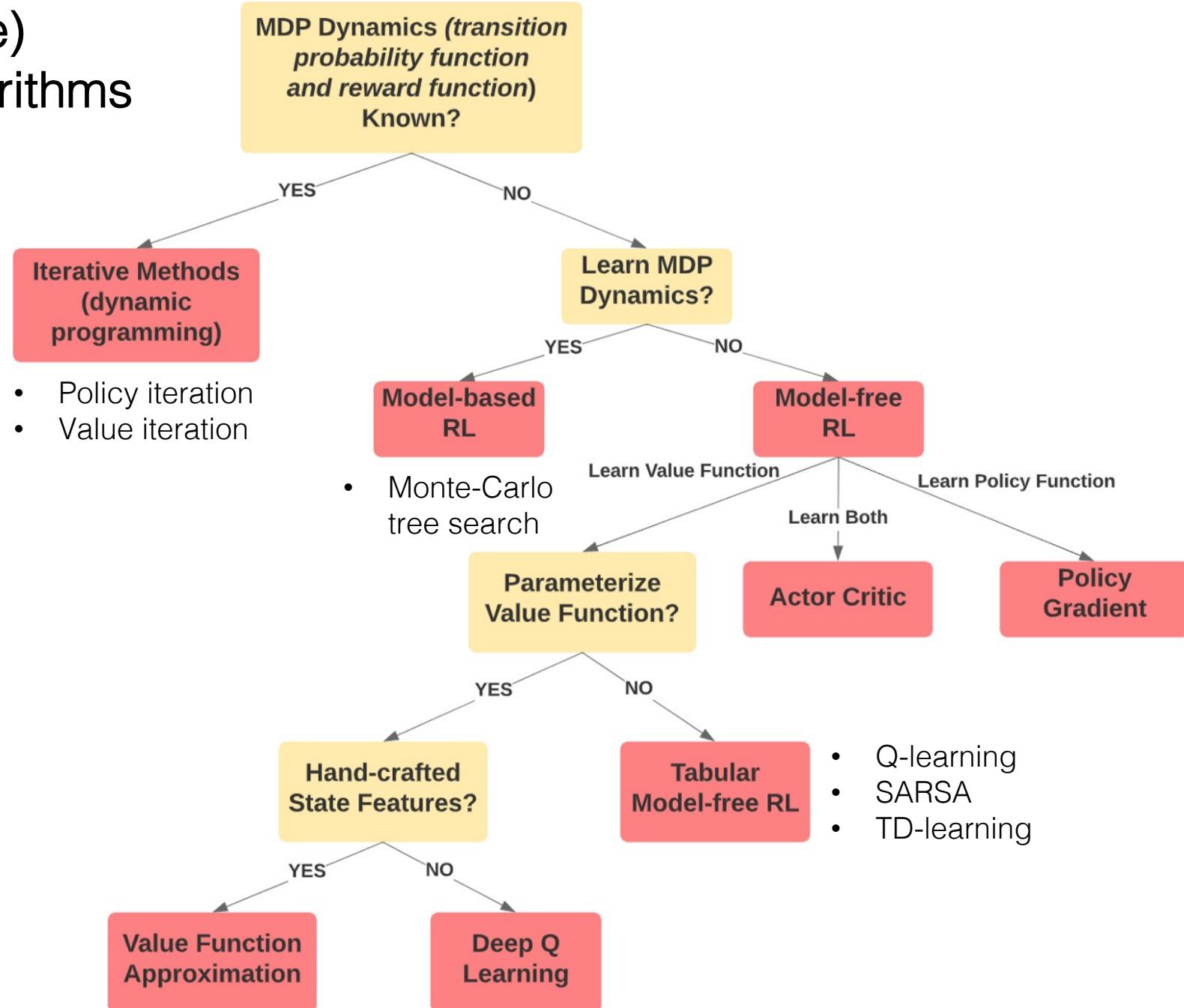
**Maybe *you* will figure it out!**

**Application questions,**

e.g. How do we use RL in robotics?  
In public health? In self-driving cars? ...

# (Non-exhaustive) taxonomy of RL algorithms

(By default,  
model-based RL)



<p><b>Policy iteration</b>  A dynamic programming algorithm (model-based RL; used when the model of the environment is fully known) that finds the optimal policy by iteratively evaluating and improving the policy until convergence.</p>	<p><b>Value iteration</b>  A dynamic programming algorithm (model-based RL; used when the model of the environment is fully known) that finds the optimal state value function by iteratively looking ahead one step to find the maximum value action until convergence.</p>
<p><b>On-policy</b>  Agent evaluates or improves the policy that is used to make decisions. e.g., SARSA</p>	<p><b>Off-policy</b>  Agent evaluates or improves a policy different than the one used to act e.g., Q-learning</p>
<p><b>Model-based RL</b>  Learns a model of the world (the dynamics of the MDP, e.g., transition function <math>T(s, a, s')</math>, reward function <math>R(s, a)</math>, etc.)</p>	<p><b>Model-free RL</b>  Does not learn a model of the world and treats states independently</p>
<p><b>Value learning</b>  Quantifying the value of every state-action pair and using that for action selection</p>	<p><b>Policy learning</b>  Directly inferring a policy that maximizes the reward in a specific environment</p>
<p><b>Actor-critic</b>  Dual architecture algorithm where actor learns policies and critic evaluates whether the actions chosen were high or low in value</p>	

# Correspondences between psychology and RL

Behavioral psychology



Computer science

Pavlovian (classical)

predicting upcoming events  
(stimuli/reward)

Algorithms for prediction

can I predict the future values of states?

vs

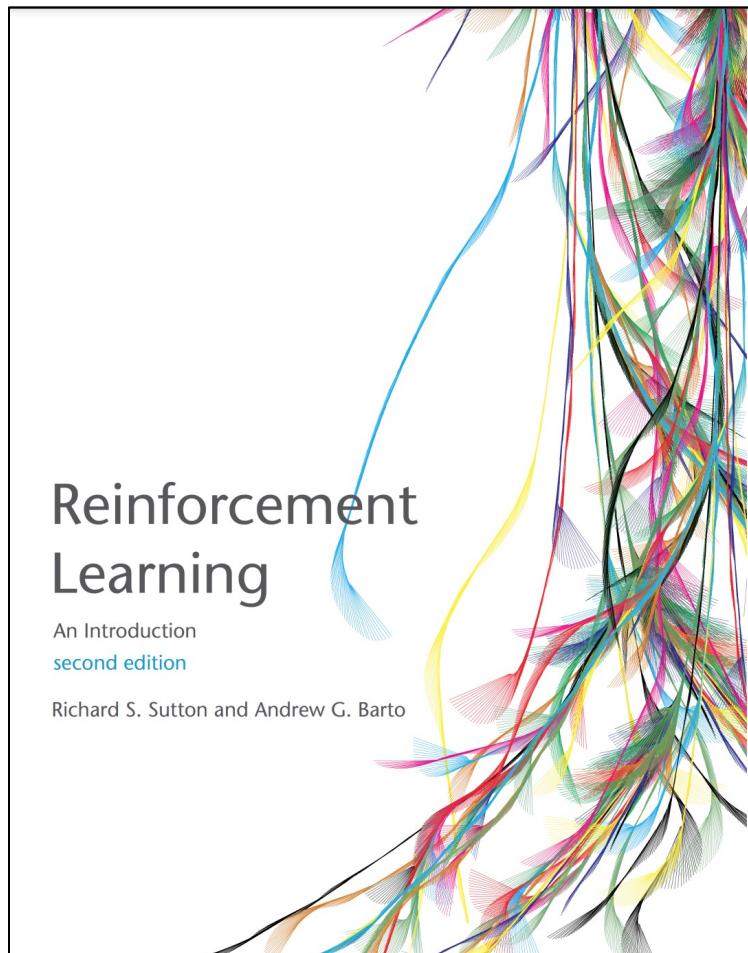
behavior contingent on  
environment

vs

operant conditioning

algorithms for control  
what should my policy be?

# To learn more...



- Read this book by Sutton and Barto!

- **Companion Python notebook:**  
[github.com/john-vastola/RL-at-Harvard-tutorial-2022](https://github.com/john-vastola/RL-at-Harvard-tutorial-2022)
- Conferences
  - Reinforcement Learning and Decision Making (RLDM) <https://rldm.org/>
  - From Neuroscience to Artificially Intelligent Systems (NAISys)
  - More ML/CS: ICLR, ICML, NeurIPS
  - More Psych/Neuro: CogSci,
- Labs at Harvard
  - See the people on the speaker list!