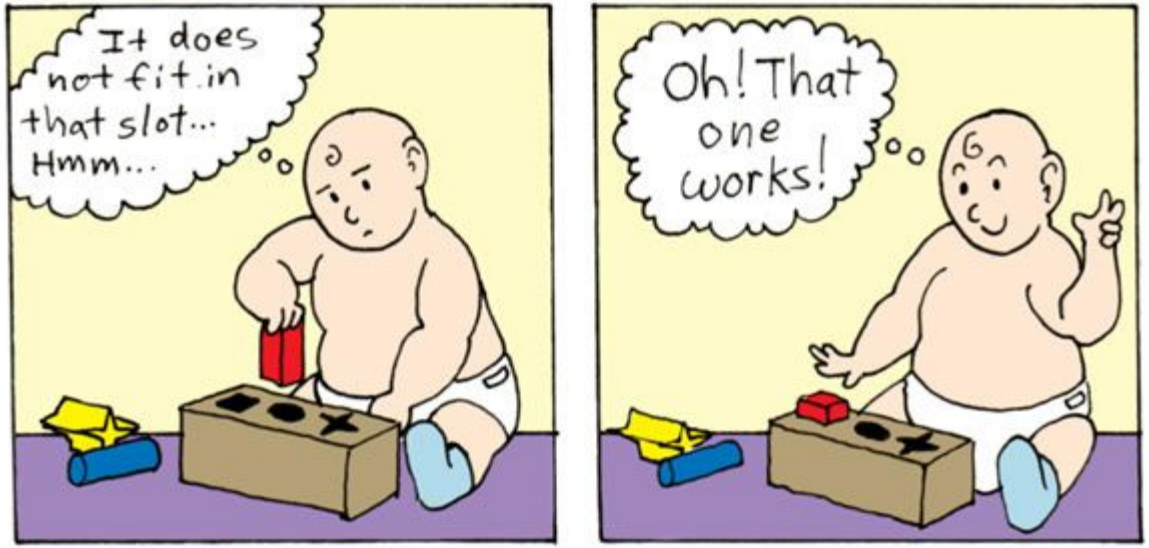
An Introduction to Reinforcement Learning

Dongyan Lin

ISICNI2023

Learning from Trial and Error



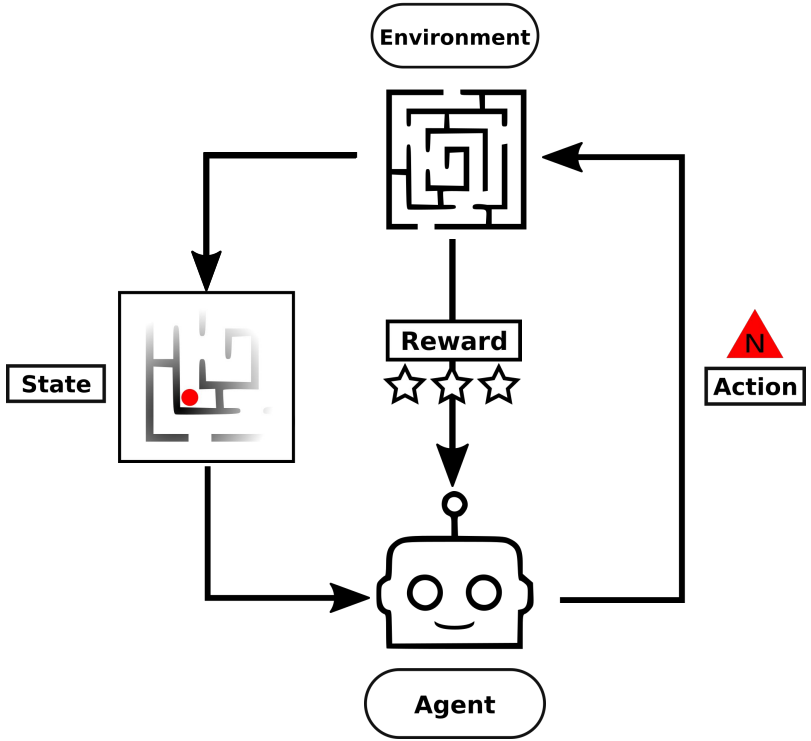
Tutorial Goals

1. Reinforcement learning **is** learning from trial-and-error

2. What Goes Into a RL problem
   - Markov Decision Processes
   - Environments
   - Agents

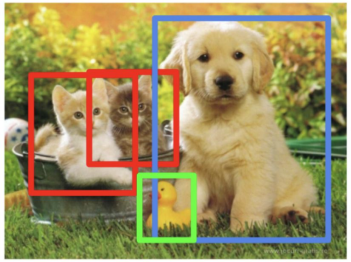
3. Solving RL Problems with RL agents
   - Tabular Solutions
   - Approximate Solutions

What is Reinforcement Learning?



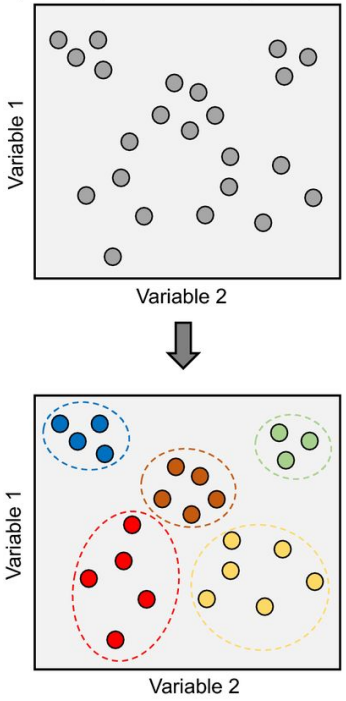
What is Reinforcement Learning?

x_1	x_2	target	output
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1



CAT, DOG, DUCK

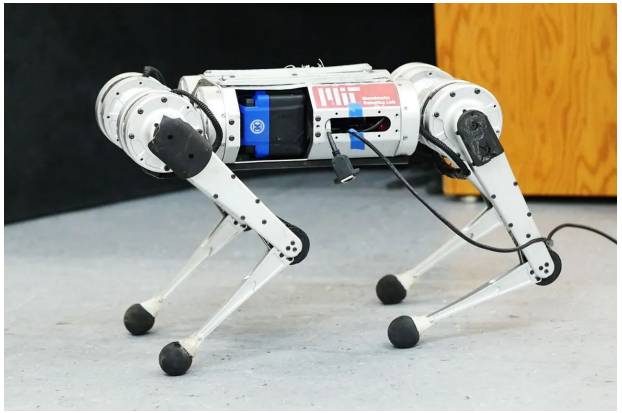
Supervised learning: Learn input-output mapping from labeled examples



Unsupervised learning: Learn structure in unlabeled data

Reinforcement Learning: Learn from a trial-and-error with a reward signal

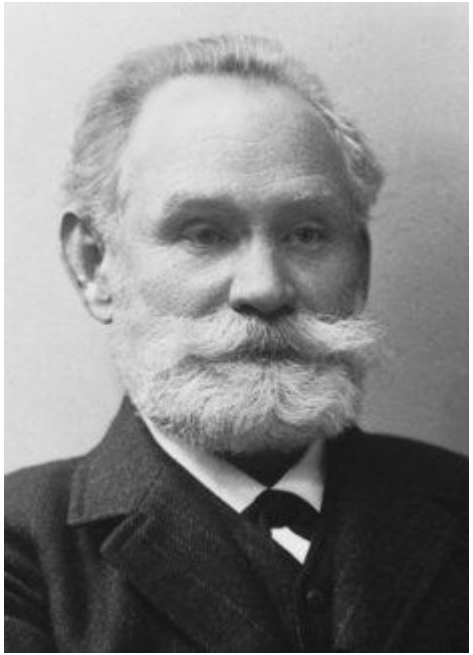
What is RL Used For?



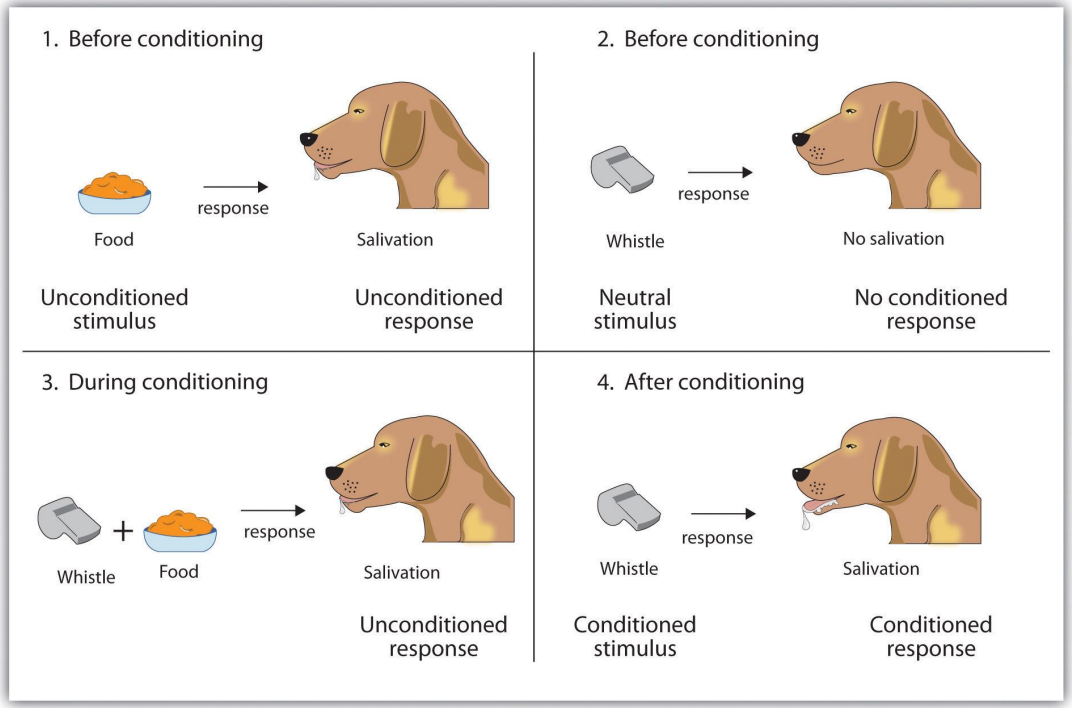
What is the meaning of life?

The meaning of life is a philosophical and existential question that has been debated by scholars and thinkers throughout history. There is no single answer that can be universally applied to all individuals, as each person's experience of life is unique and subjective.

How Does RL Relate to Neuroscience?



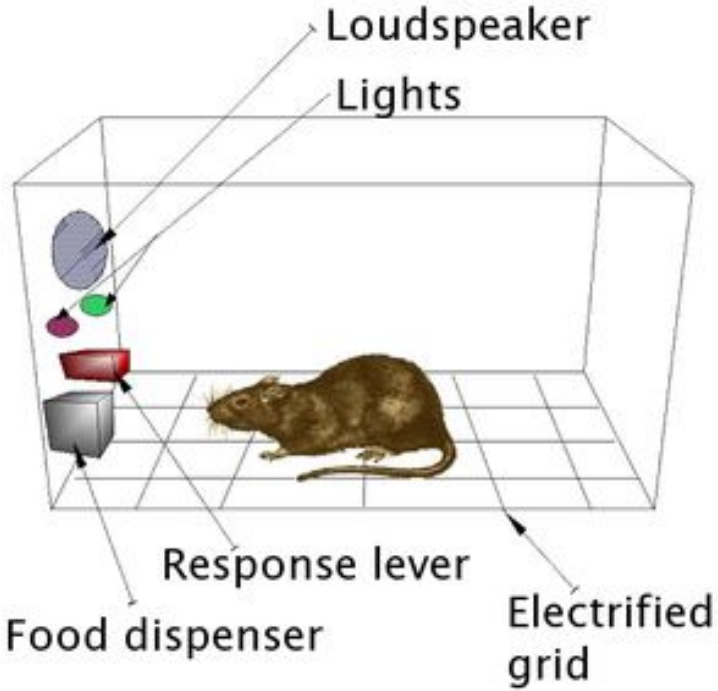
Ivan Pavlov



Classical conditioning is the process of learning to **predict** the world around you

How Does RL Relate to Neuroscience?

BF Skinner

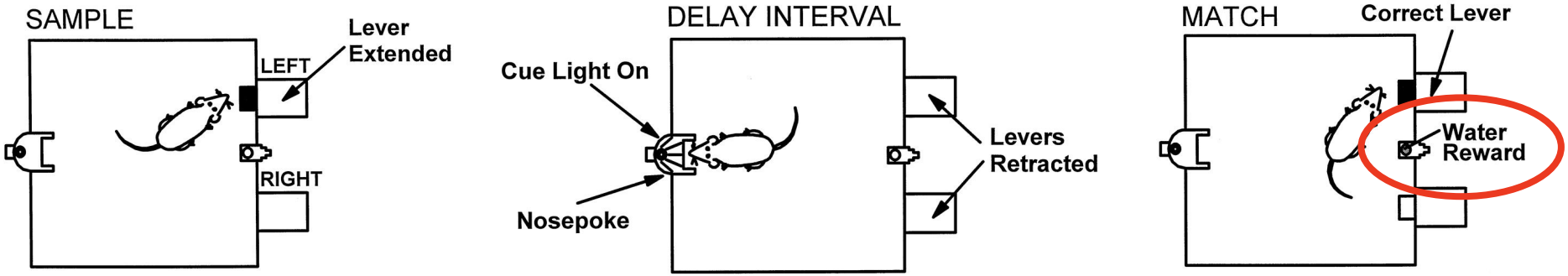


Operant conditioning is the process of learning how to **control** behaviour for the best outcome

How Does RL Relate to Neuroscience?

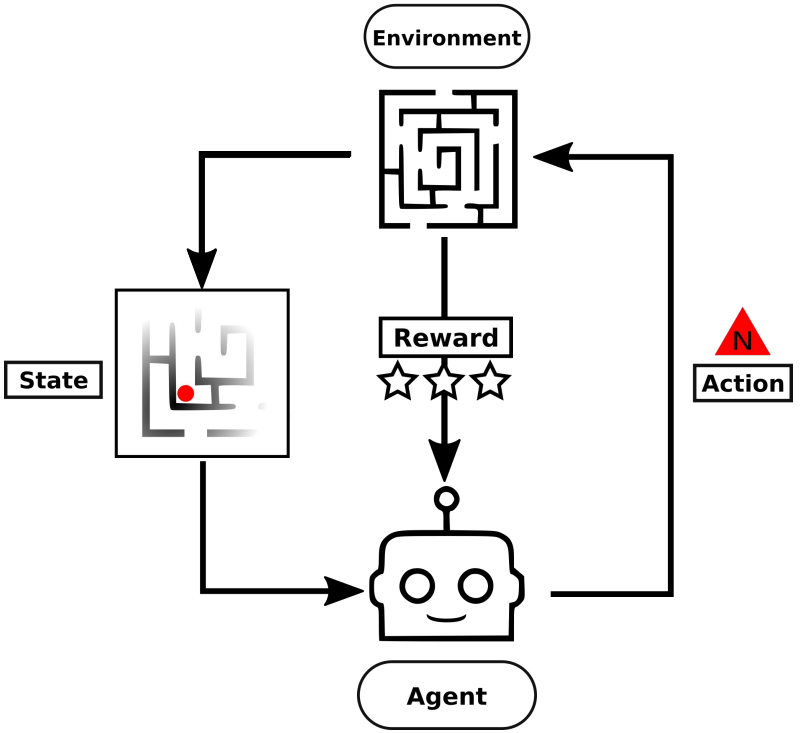
Many neuroscience experiments train animals on some task with RL... Making RL an excellent model for animal **behaviour** (and sometimes **neural activity**!)

Eg. Delayed Match-to-Sample task



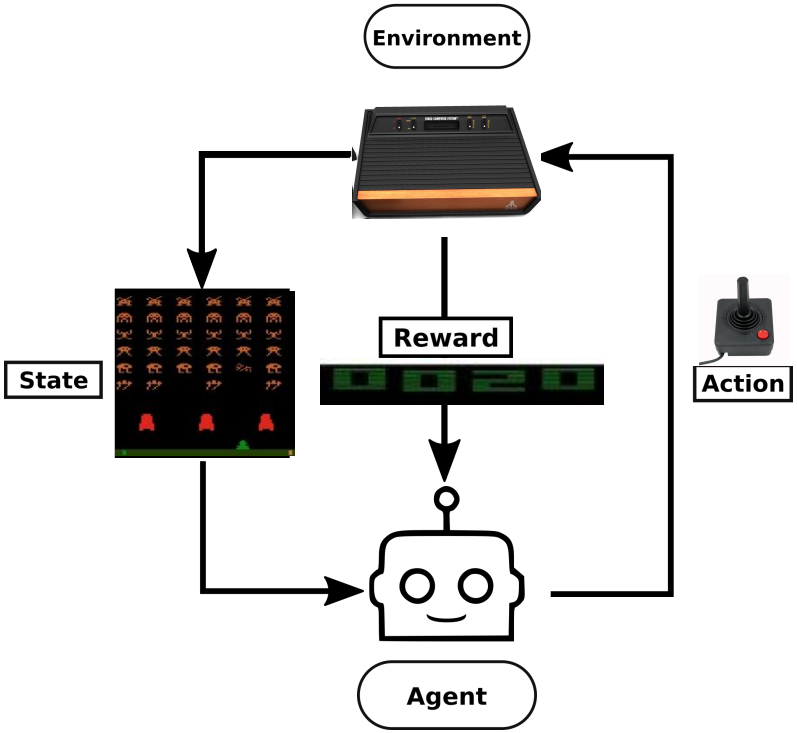
Reinforcement Learning = Optimizing Behaviour

- How should agents act **optimally**?
  - Optimality ~ maximizing cumulative reward
- Formulated as the environment-agent interaction
  - Agent provides **action** to the environment
  - Environment provides **observation (i.e., state)** and **reward** to the agent



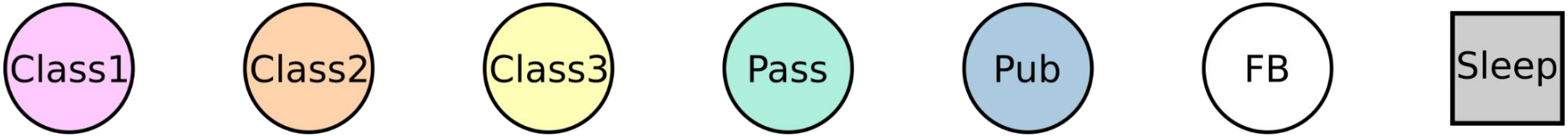
Reinforcement Learning = Optimizing Behaviour

- How should agents act **optimally**?
  - Optimality ~ maximizing cumulative reward
- Formulated as the environment-agent interaction
  - Agent provides **action** to the environment
  - Environment provides **observation (i.e., state)** and **reward** to the agent



What goes into a RL problem?

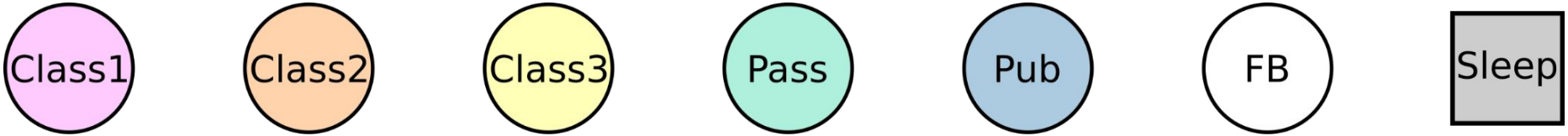
Ch.1. Markov Decision Process



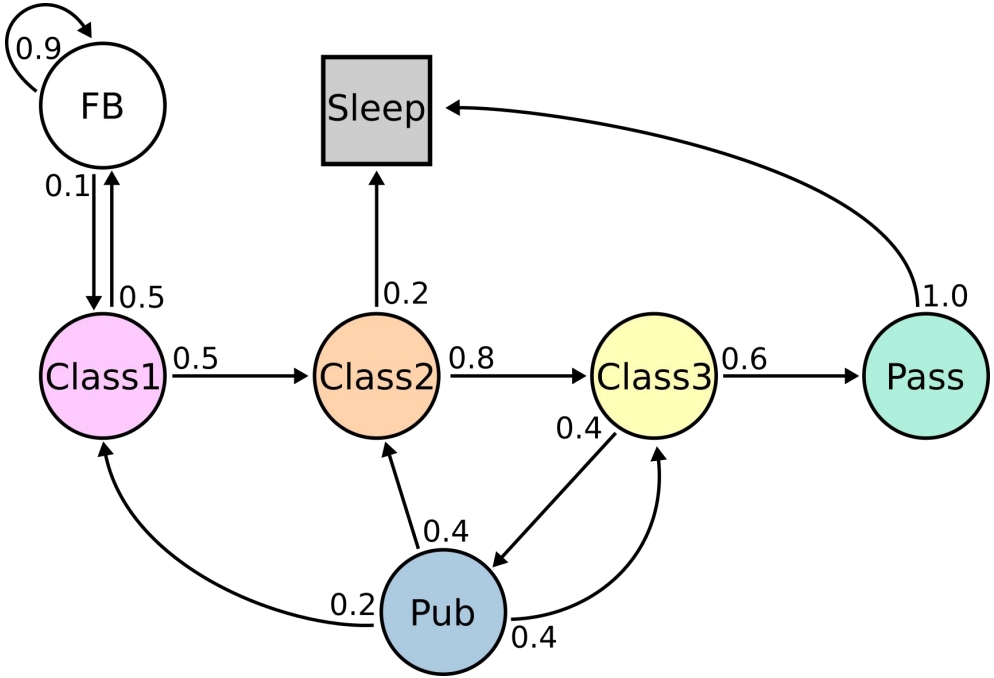
An environment is a **set of states** and

the **rules** that specify moves between them

States



State Transitions



$$\mathcal{P}^{a}_{ss'} = \Pr[S_{t+1} = s' | S_t = s, A_t = a]$$

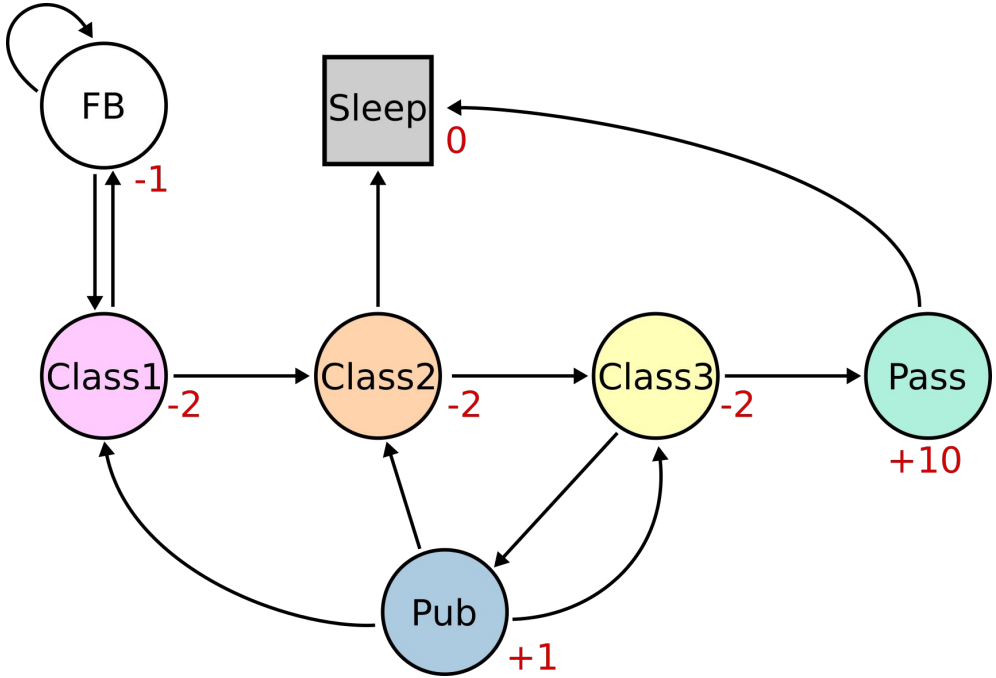
next state s'

current state s

	Class1	Class2	Class3	Pass	Pub	FB	Sleep
Class1		0.5				0.5	
Class2			0.8				0.2
Class3				0.6	0.4		
Pass							1.0
Pub	0.2	0.4	0.4				
FB	0.1					0.9	
Sleep							1.0

Markov Process

Rewards



$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_{t+1} | S_t = s, S_{t+1} = s', A_t = a\right]$$

Class1	Class2	Class3	Pass	Pub	FB	Sleep
-2	-2	-2	+10	+1	-1	0

Returns

- The **return** (aka Gain, hence G_t) is the total discounted reward from step t:

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

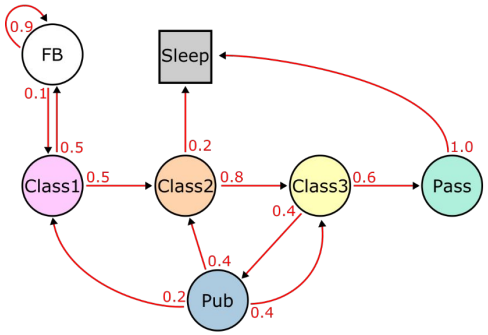
- Why discount?
  - Rewards in the future may not be as valuable as rewards now
  - $\gamma \in [0, 1]$ controls our 'temporal horizon':
    - γ = 0: the only reward we care about is NOW (most 'nearsighted')
    - γ = 1: all rewards (to infinity) have equal value (most 'farsighted')

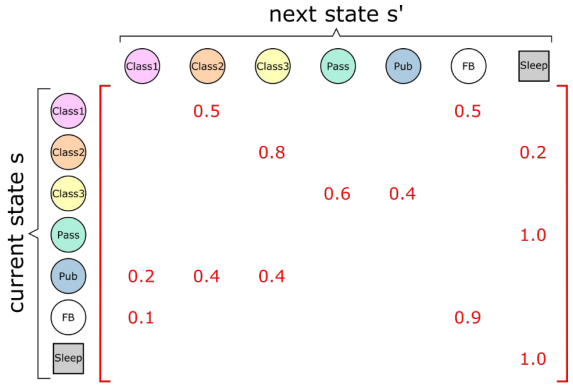
Markov Reward Process

Actions & Transitions

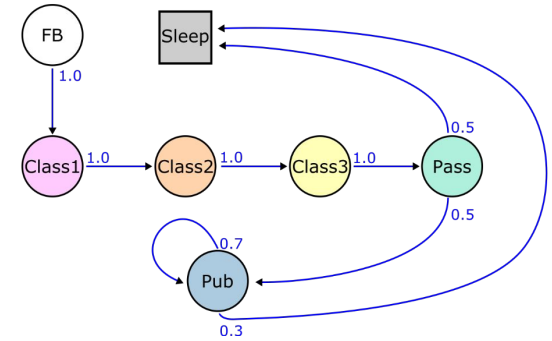
$$\mathcal{P}^{a}_{ss'} = \Pr[S_{t+1} = s' | S_t = s, A_t = a]$$

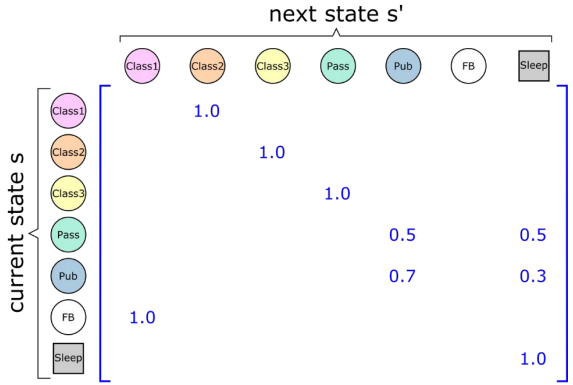
Chill





Study



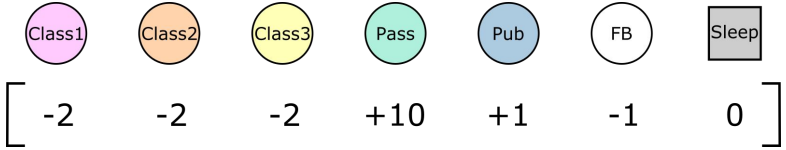


Markov Decision Processes

- $\mathcal{S}$, a finite set of states

- $\mathcal{A}$, a finite set of actions **Study, Chill**
- $\mathcal{P}$, state transition probabilities (one matrix for each action)

- $\mathcal{R}$, a reward function: (state, action, next state) $\longrightarrow$ scalar value



- $\gamma \in [0, 1]$, a discount factor (weighing importance of immediate vs future reward)

Value & Policy Functions

The value of a state s is the **expected long term reward that can be achieved**

$$v(s) = \mathbb{E}[G_t|S_t = s]$$

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

$$\mathbb{X} = \{x_1, x_2, x_3 \ldots x_n\}$$

$$\mathbb{P} = \{p_1, p_2, p_3 \ldots p_n\}$$

$$\mathbb{E} = p_1x_1 + p_2x_2 + \ldots + p_nx_n$$

Value & Policy Functions

The value of a state s is the **expected long term reward that can be achieved**

$$v(s) = \mathbb{E}[G_t|S_t = s]$$

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

A policy (π) is a **distribution over actions given states**

$$\pi[a|s] = \Pr[A_t = a|S_t = s]$$

Markov Decision Process

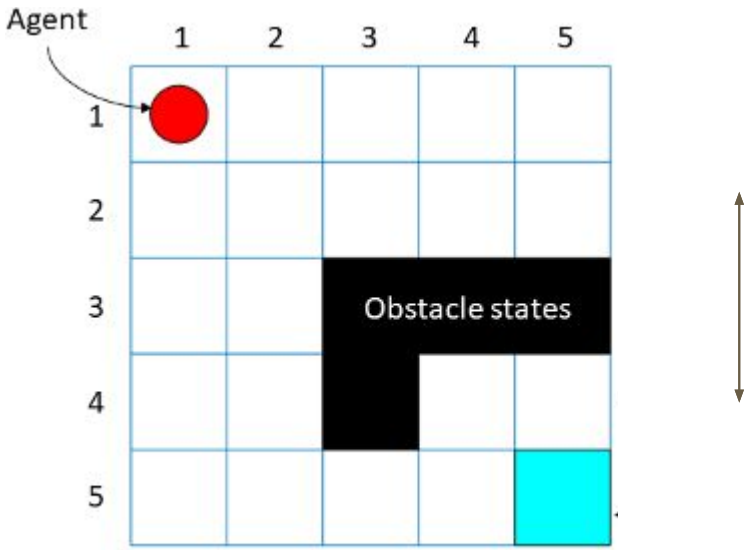
Worksheet Sec. 1&2

BREAK

Worksheet Sec. 1&2

What goes into a RL problem?

Ch.2. The environment



An environment is a **set of states** and

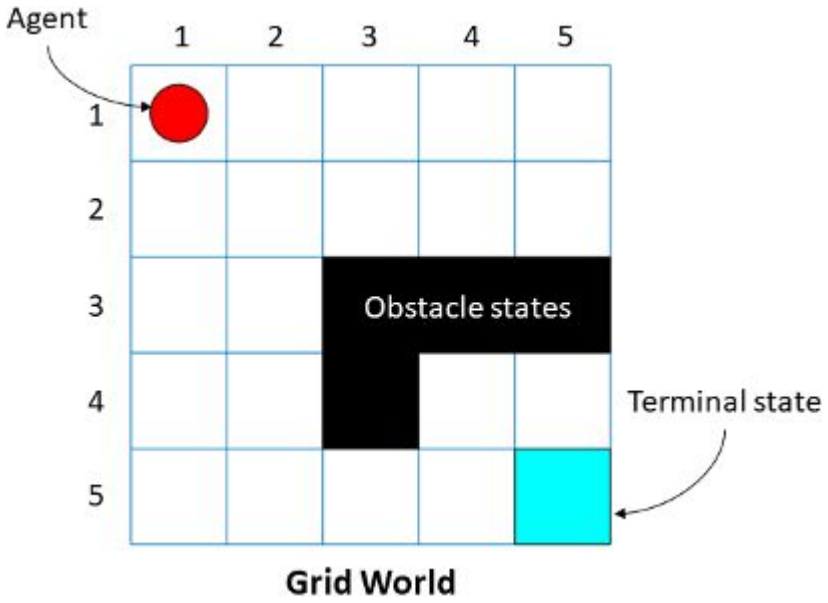
the **rules** that specify moves between them

Environment

State Ex. S1=(1,1), S25=(5,5)

Action Ex. “Down”

Reward Ex. -1 or +10



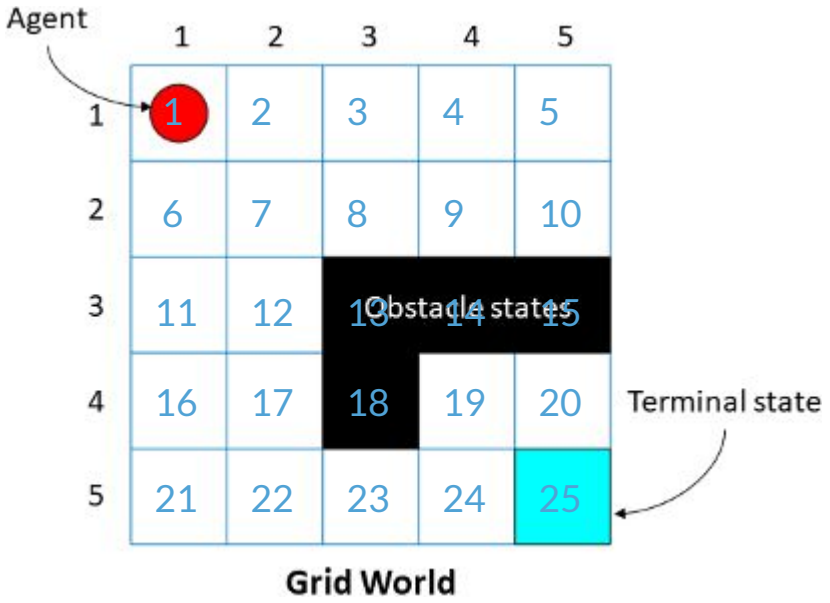
Environment: Rewards

Reward function **R(s)**

- R(S1) = -1
- R(S25) = +10

→Can be represented by a 1D vector:

-1	-1	-1	-1	…	+10

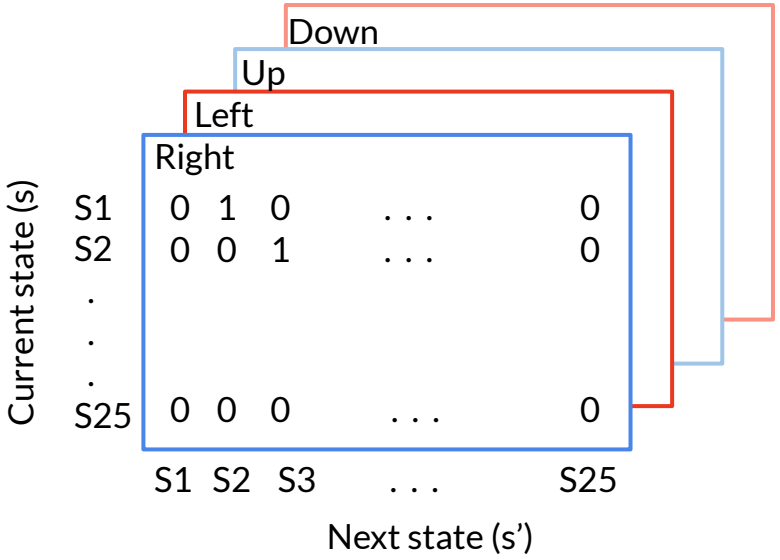


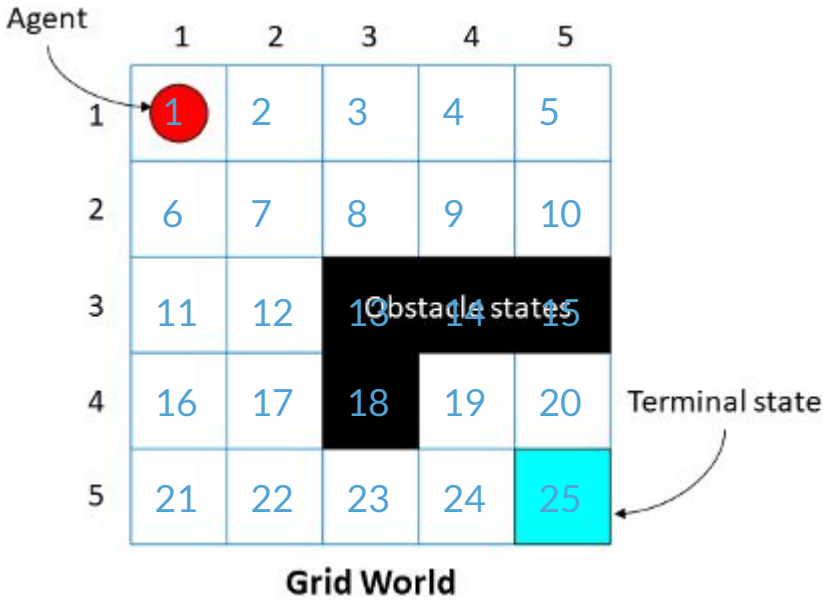
Environment: State Transition Probability

Transition Function **P(s, s', a)**:

A rule for what moves are allowed

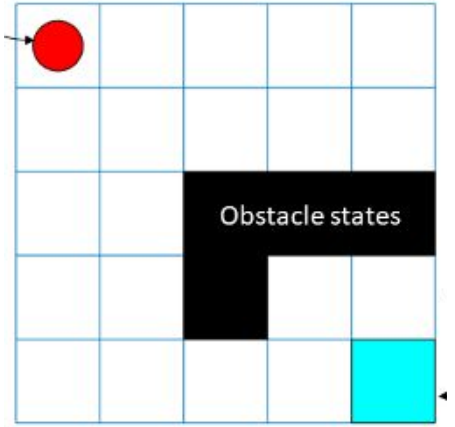
→ Can be represented by a 3D tensor





Worksheet Sec. 3

Markov Decision Processes



- $\mathcal{S}$, a finite set of states **{(0,0), (0,1), (0,2), ... (5,5)}**
- $\mathcal{A}$, a finite set of actions **{Up, Down, Left, Right}**
- $\mathcal{P}$, state transition probabilities (one matrix for each action)

- $\mathcal{R}$, a reward function: (state, action, next state) $\longrightarrow$ scalar value

- $\gamma \in [0, 1]$, a discount factor (weighing importance of immediate vs future reward)

Trackword & Gridworld

BREAK

What goes into a RL problem (and how to solve it)?

Ch.3. The agent (aka the fun part)

What goes into beating a video game?

What goes into beating a video game?

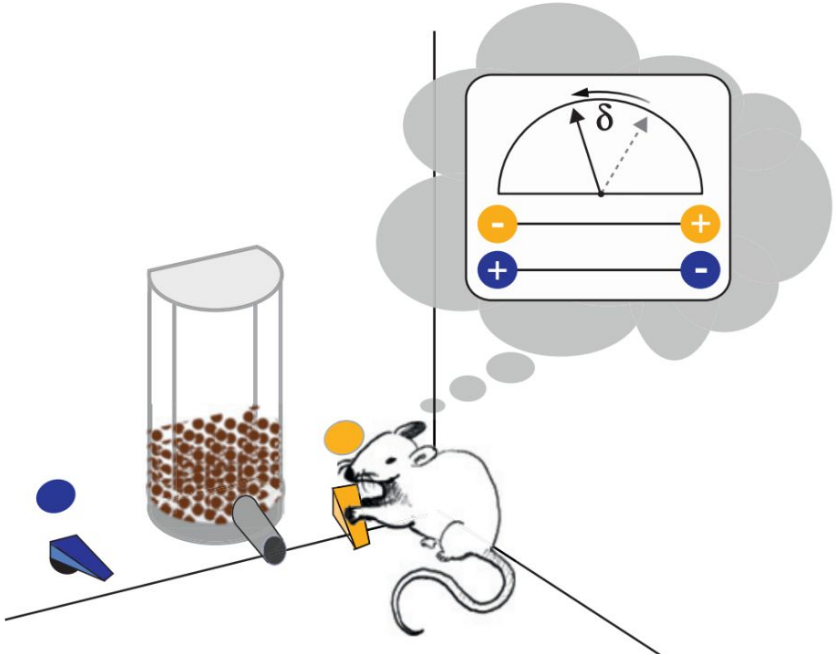
- Model **T (i.e., $<R^a_s, P^a_{ss'}>$)**
  - A representation of the **environment dynamics**
- Value Function **v(s)** or **q(s,a)**
  - An estimate how much reward is expected over time for each state and/or action
- Policy **π**
  - A function which maps states onto actions
  - A (conditional) probability distribution
    - gives the probability of selecting each action given current state

Generally, RL can be categorized as...

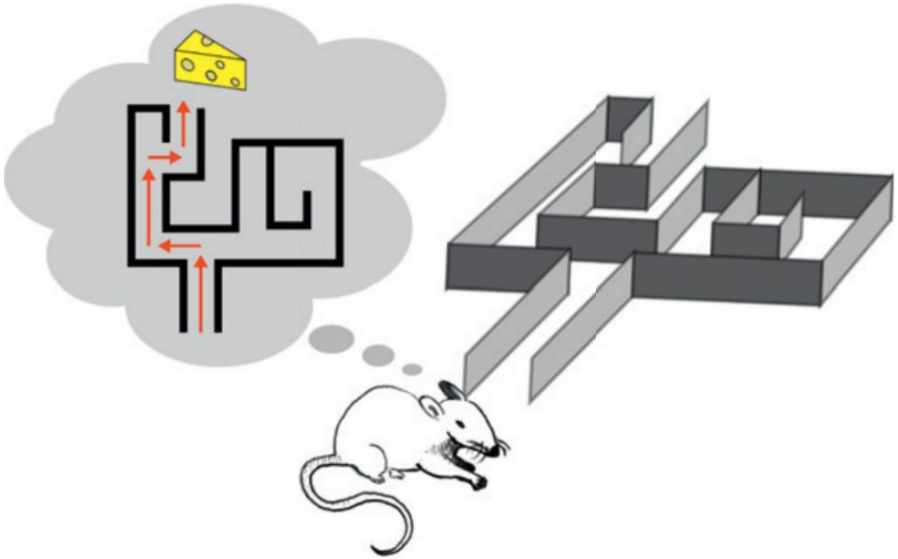
- Model-based RL
  - solves the problem using **model** (i.e., by knowing the **transition function** and **reward function**)
- Model-free RL:
  - solves the problem using **value function** and **policy** learned from experience

Examples of MF vs. MB RL

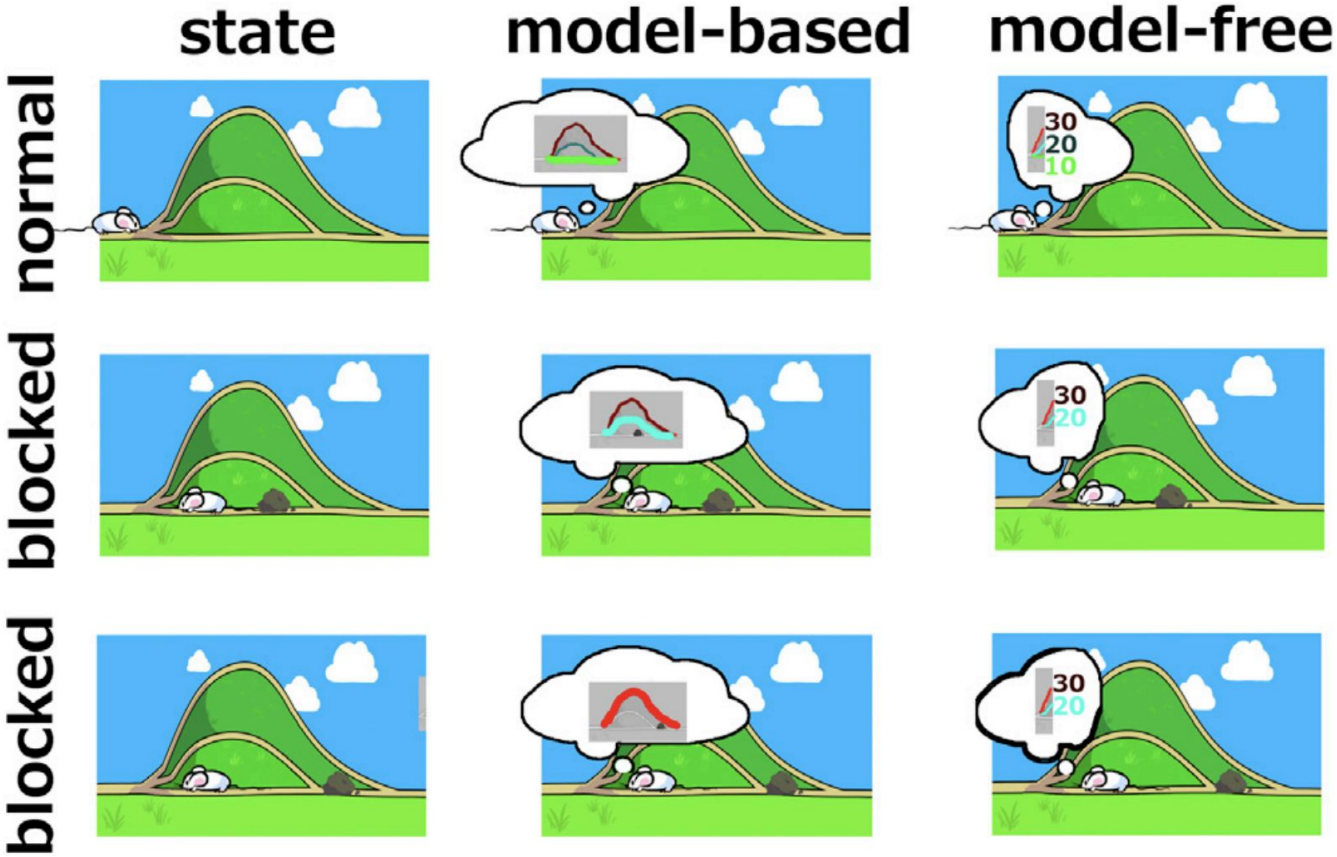
Model-free



Model-based



Examples of MF vs. MB RL



Comparing MF and MB RL

Model-Based	Model-Free
Learns through constructing a **world model**	Learns through **extensive experience** of trial and error
Flexible adaptation to changes in environment dynamics	**Slow** adaptation to changes in environment dynamics
Requires **large amount of compute** (i.e., memory) to store the transitions	Only need to store V (or q) and π
Goal-directed	Habitual

Recall: Value & Policy Functions

The **value** of a state s: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$

The **policy**: $\pi[a|s] = \Pr[A_t = a|S_t = s]$

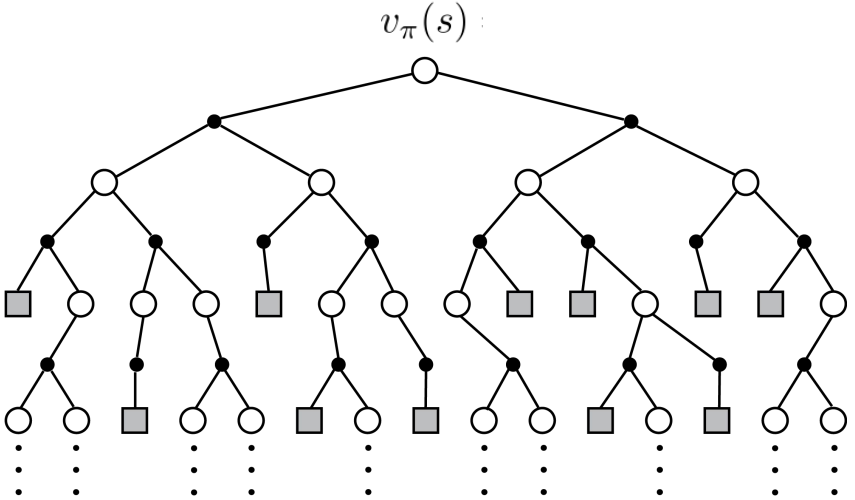
Ex. Greedy policy: always take the most optimal action

$$\pi(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s, a)$$

Ex. ε-Greedy policy: only takes the optimal action with probability (1-ε); other times random action

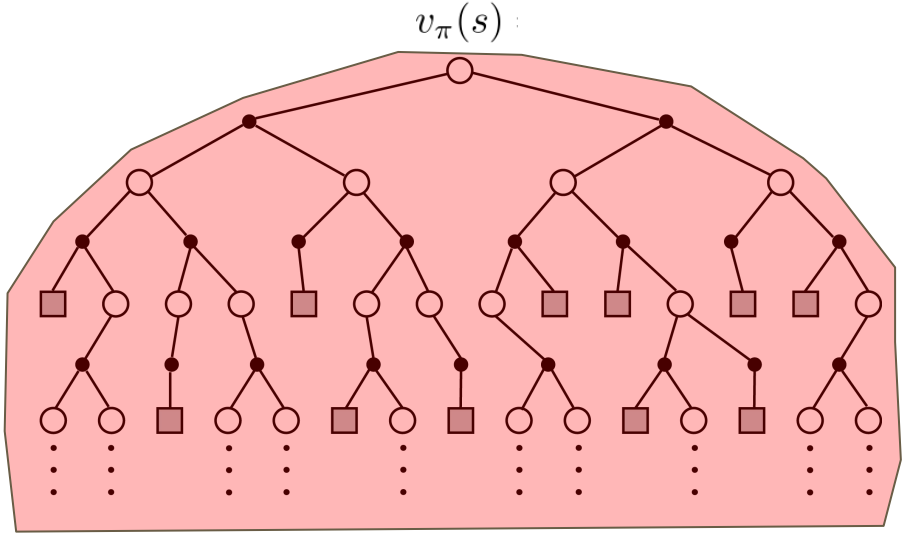
$$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

How to Compute Value of Current State?



How to Compute Value of Current State?

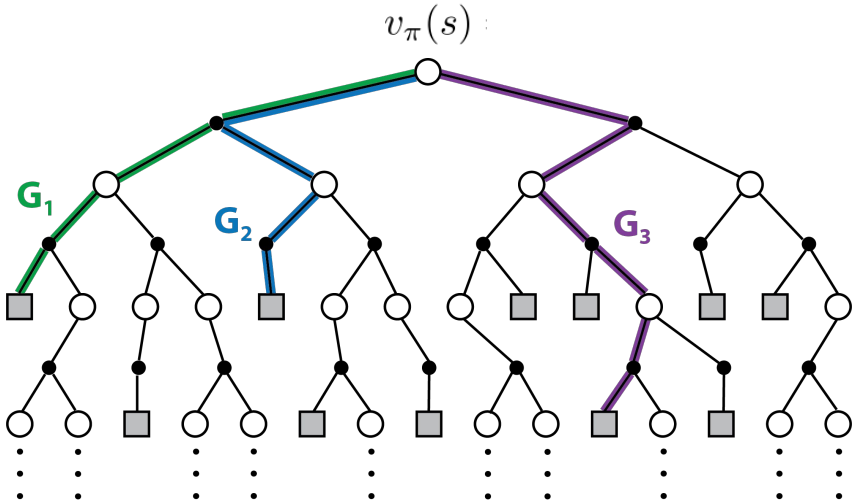
- Brute force
  - Compute value of all possible episodes and take weighted sum.
  - Need to know MDP transition and reward
  - I.e., "**Exhaustive Search**"



$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \sum_{\tau \in \text{MDP}} p(\tau|\pi, S_t = s)G(\tau)$$

How to Compute Value of Current State?

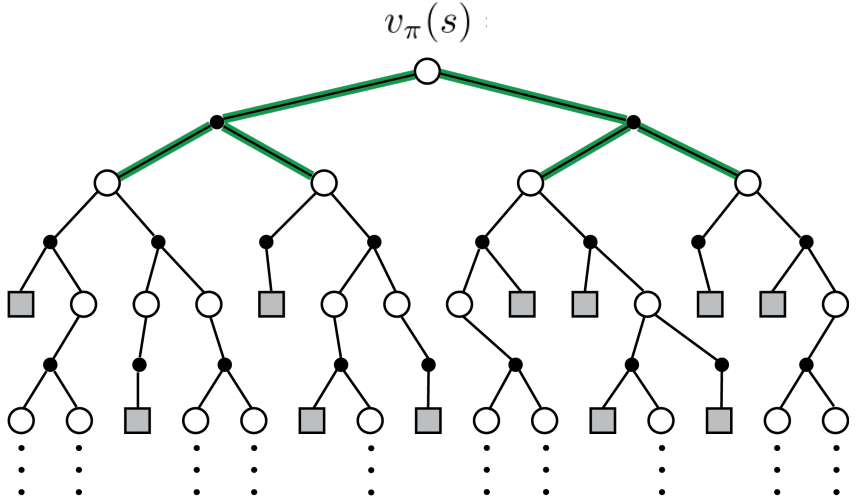
- Sampling
  - Pick episodes randomly and take empirical
    average over N sampled complete episodes
  - Do not need to know MDP transition and reward
  - Eg. **Monte Carlo**



$$v_\pi(s) = \frac{1}{N}\sum_{i=1}^{N} G(\tau_i)$$

How to Compute Value of Current State?

- Bootstrapping
  - Learn from incomplete episodes
  - Update one estimate with another estimate
  - Do not need to know MDP transition and reward
  - Eg. **Dynamic Programming, Temporal Difference**



$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + ... | S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

Learning to solve RL problems...

- MB RL: Through learning the model
- MF RL: Through learning the value function and the policy
  - Tabular Methods
    - Dynamic Programming
    - Monte Carlo
    - Temporal Difference
  - Approximate solution methods
    - Deep Q Network (DQN)
    - Policy-gradient methods
    - Actor-Critic

Worksheet Sec. 4 (first 3 bullet points)

Tabular agent

BREAK

Worksheet Sec. 4 (first 3 bullet points)

Learning Value

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

Error – how much to update by

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\left(\boxed{\mathbf{TARGET}} - V_\pi(s_t)\right)$$

New (Updated) Value

Old Value

Learning Rate – how much error contributes to new value

Learning Value with Monte Carlo

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

Error – how much to update by

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\left(G_t - V_\pi(s_t)\right)$$

New (Updated) Value

Old Value

Learning Rate – how much error contributes to new value

Empirical return G_t (i.e. sampled)

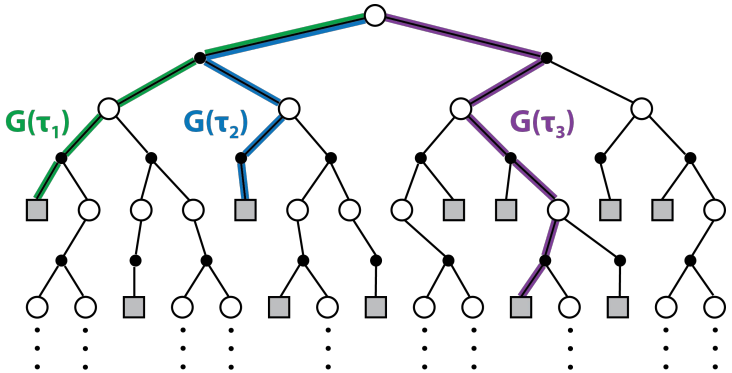
$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_{t+T}$$

Learning Value with Monte Carlo

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

Learn directly from episodes of experience (i.e., sampling) — upon episode completion, update the value of **all states (or state-actions) visited** with:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_{t+T}$$



$$V(s) = \frac{1}{N} \sum_{i=1}^{N} G_t(\tau_i), \text{ for } S_t = s$$

Worksheet
Sec. 4. Bullet 4

Incremental update:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\big(G_t - V_\pi(s_t)\big)$$

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

Learning Value and Policy with Monte Carlo

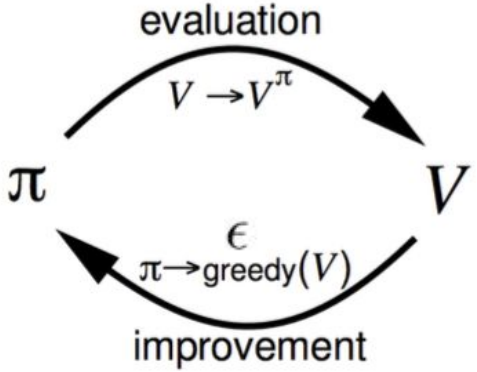
- Evaluation: for each episode, update the estimate of the value

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\big(G_t - V_\pi(s_t)\big)$$

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\big(G_t - Q_\pi(s_t, a_t)\big)$$

- Improvement: update policy to be ϵ-greedy w.r.t value function

$$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



Monte Carlo

Learning Value: Temporal Difference

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

Temporal difference (TD) learning learns value estimation by **bootstrapping using another value estimate,** plus some empirical evidence (observed reward)

$$\begin{aligned} v(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right] \end{aligned}$$

Learning Value with Temporal Difference

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

Error – how much to update by

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\left(r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\right)$$

Estimate of return G_t

Learning Rate – how much error contributes to new value

New (Updated) Value

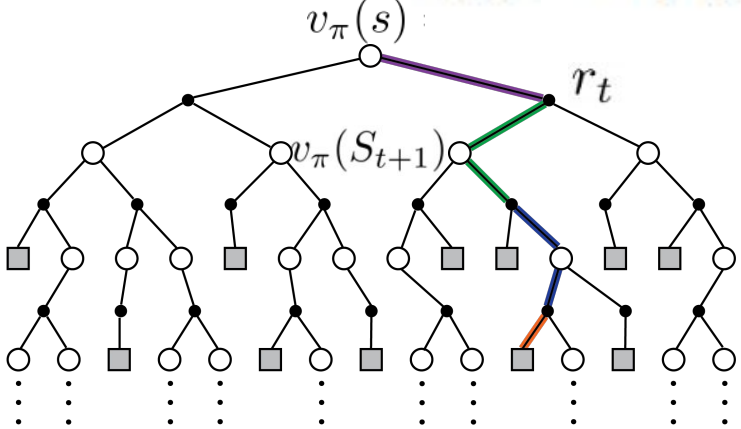
Old Value

Learning Value: Temporal Difference

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

Temporal difference (TD) learning learns value estimation by **bootstrapping using another value estimate,** plus some empirical evidence (observed reward)

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$



Incremental update:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\big(r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\big)$$

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\big(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)\big)$$

Learning Value: Temporal Difference

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

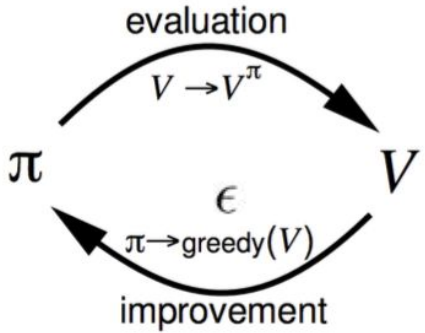
- **Evaluation:** for each episode, update the estimate of the value

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha\big(r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\big)$$

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\big(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)\big)$$

- **Improvement:** update policy to be ϵ-greedy w.r.t value function*

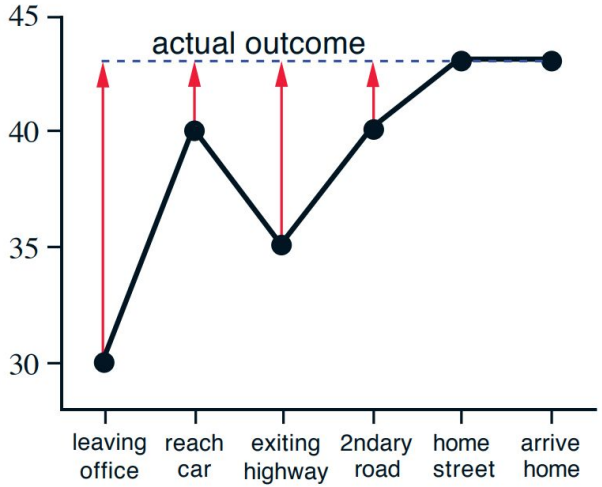
$$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



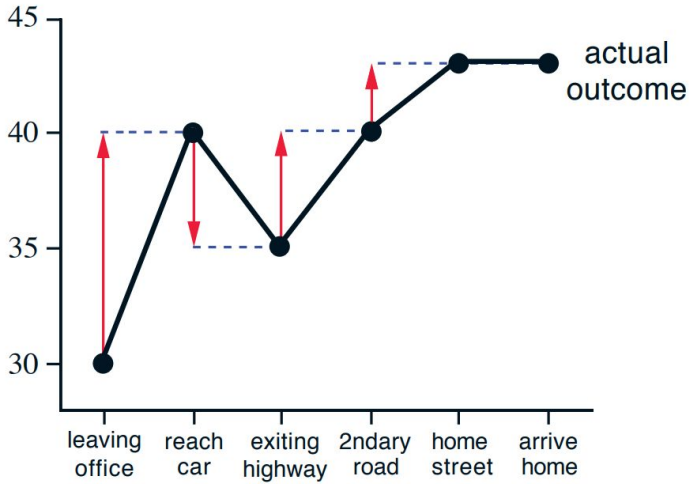
Example: Driving Home

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Changes Recommended by MC



Changes Recommended by TD



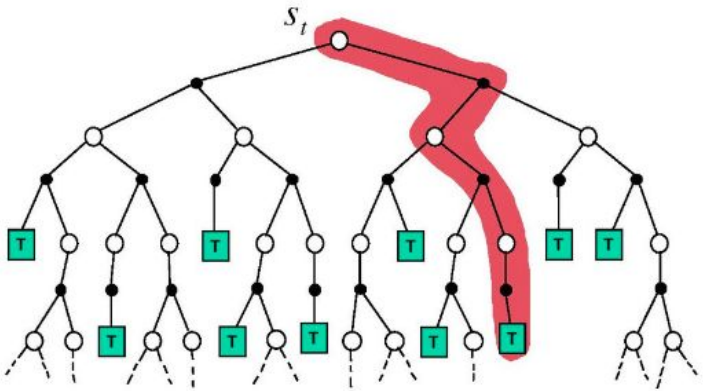
Temporal Difference vs Monte Carlo

MC	Simplest TD
Update $V(s_t)$ toward actual sampled return G_t	Update $V(s_t)$ toward estimated return $R_{t+1} + \gamma V(s_{t+1})$
$V(s_t) \leftarrow V(s_t) + \alpha (G_t - V(s_t))$	$V(s_t) \leftarrow V(s_t) + \alpha (R_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

Temporal Difference vs Monte Carlo

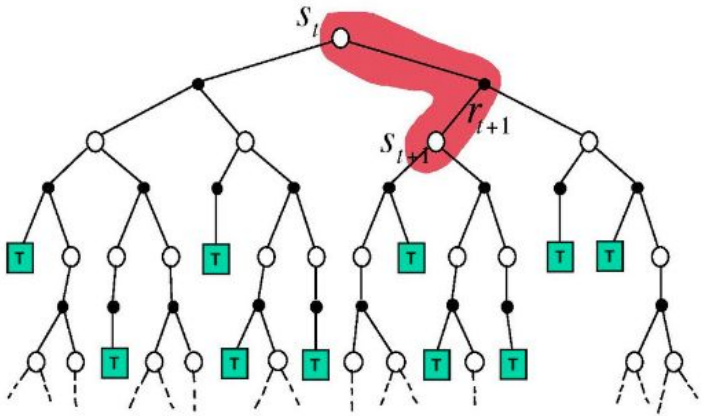
Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$



Temporal-Difference Backup

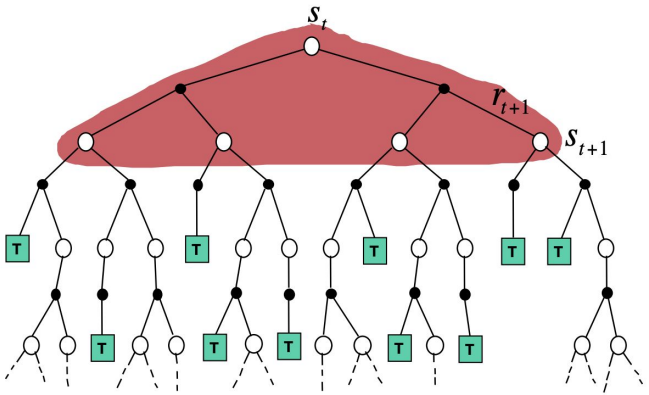
$$V(S_t) \leftarrow V(S_t) + \alpha\,(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Things we didn't go into detail about...

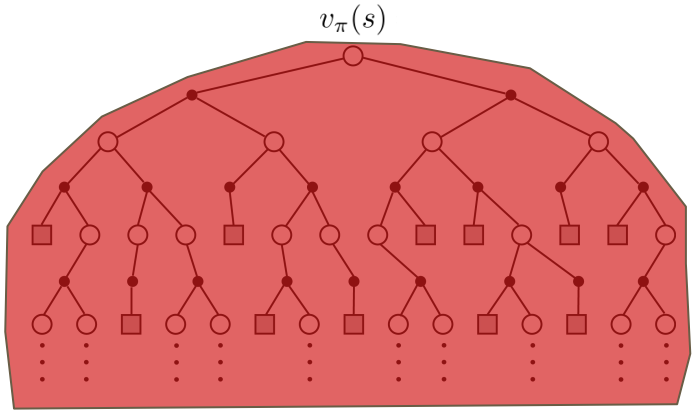
Dynamic Programming Backup

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) \right]$$



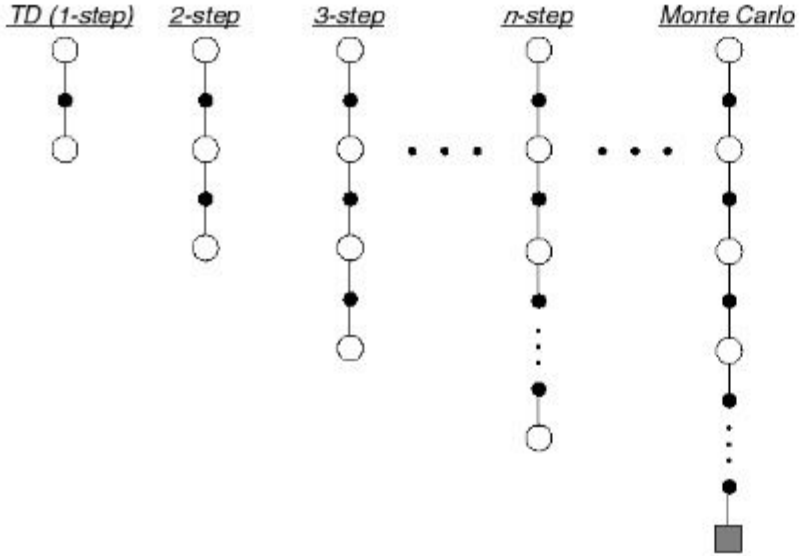
Exhaustive Search

$$v_\pi(s) = \mathbb{E}[G_t | S_t = s] = \sum_{\tau \in \text{MDP}} p(\tau | \pi, S_t = s) G(\tau)$$



In Between MC & TD: TD-lambda

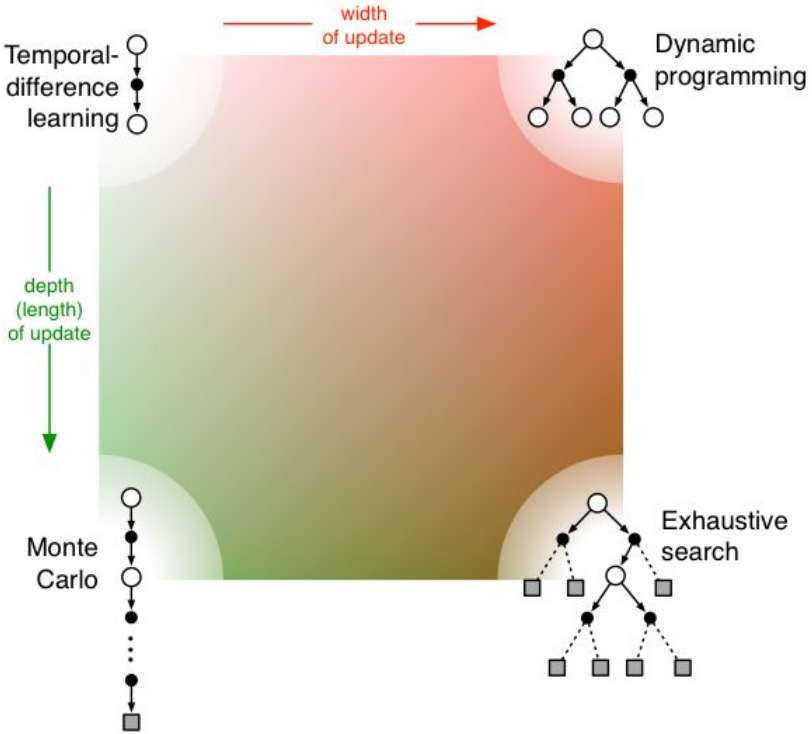
Let TD target look n steps into the future



Consider the following n-step returns for $n = 1, 2, \infty$:

$$
\begin{aligned}
n = 1 \quad (TD) \quad & G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 \quad & G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots \quad & \vdots \\
n = \infty \quad (MC) \quad & G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{aligned}
$$

How Do All These Relate?



Sutton & Barto

On-/Off-Policy Learning

- On-policy: behavioural policy = target policy
- Off-policy: behavioural policy ≠ target policy

	Action Selection Policy (Behaviour Policy)	Target Generation Policy
On Policy	ϵ- greedy	ϵ - greedy
Off Policy	ϵ- greedy	Greedy - no chance of random action selection

Learning Value: Temporal Difference

Error – how much to update by

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha\left(r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)\right)$$

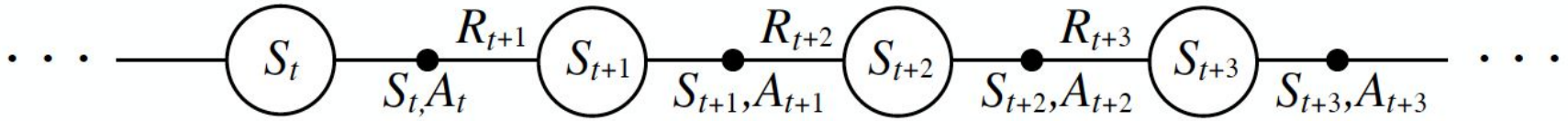
Estimate of return G_t

Learning Rate – how much error contributes to new value

New (Updated) Value

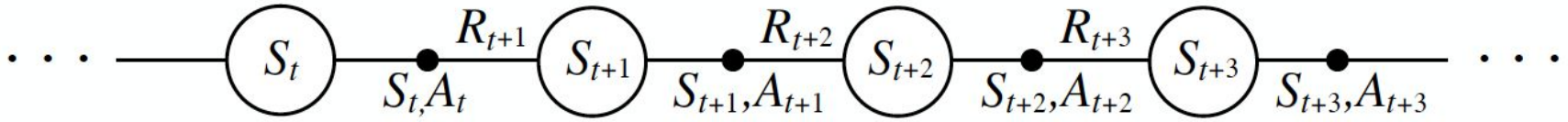
Old Value

On-Policy TD Learning: SARSA



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

On-Policy TD Learning: SARSA



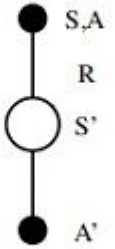
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

The policy that led to this reward...

...is the same as the policy that led to this action

On-Policy TD Learning: SARSA

"One-step look-ahead"



"SARSA"

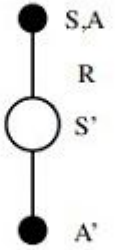
"Learning rule"

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

On-Policy TD Learning: SARSA

"One-step look-ahead"



"SARSA"

"Learning rule"

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Behavioural policy

Target policy

TD Learning: SARSA

Off-Policy TD Learning: Q Learning

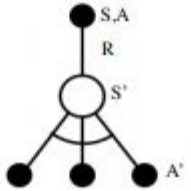
- The agent has two policies — the behavior policy $\mu(s)$ (e.g., ϵ-greedy), and the optimized policy $\pi(s)$ (e.g., greedy)

- The next action is chosen using the behaviour policy, but Q-values are updated using the optimized policy.

$$Q(s, a_\mu) \leftarrow Q(s, a_\mu) + \alpha(r + \gamma Q(s', a'_\pi) - Q(s, a_\mu))$$

ϵ

Off-Policy TD Learning: Q Learning

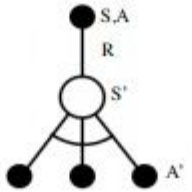


"One-step look-ahead"

"Learning rule" $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
        $s \leftarrow s'$;
    until s is terminal

Off-Policy TD Learning: Q Learning



"One-step look-ahead"

"Learning rule" $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q(S',a') - Q(S,A)\right)$

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Behavioural policy

Target policy

SARSA vs Q-Learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

	Action Selection Policy (Behaviour Policy)	Target Generation Policy
On Policy	ϵ- greedy	ϵ - greedy
Off Policy	ϵ- greedy	Greedy - no chance of random action selection

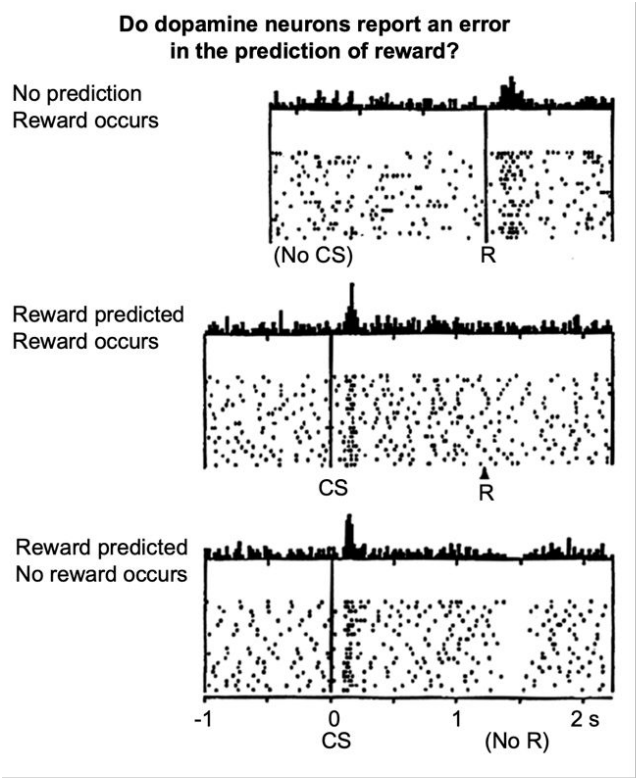
TD Learning: Q-Learning

TD-learning in the brain (RL for neuroscience!!)

Phasic dopamine encodes "reward prediction error"

$$\delta_t = V_t(s_t) - R_t + \gamma V_t(s_{t+1})$$

= Dopamine Firing Rate



Schultz, Dayan Montague (1997)

Approximate Solution Methods

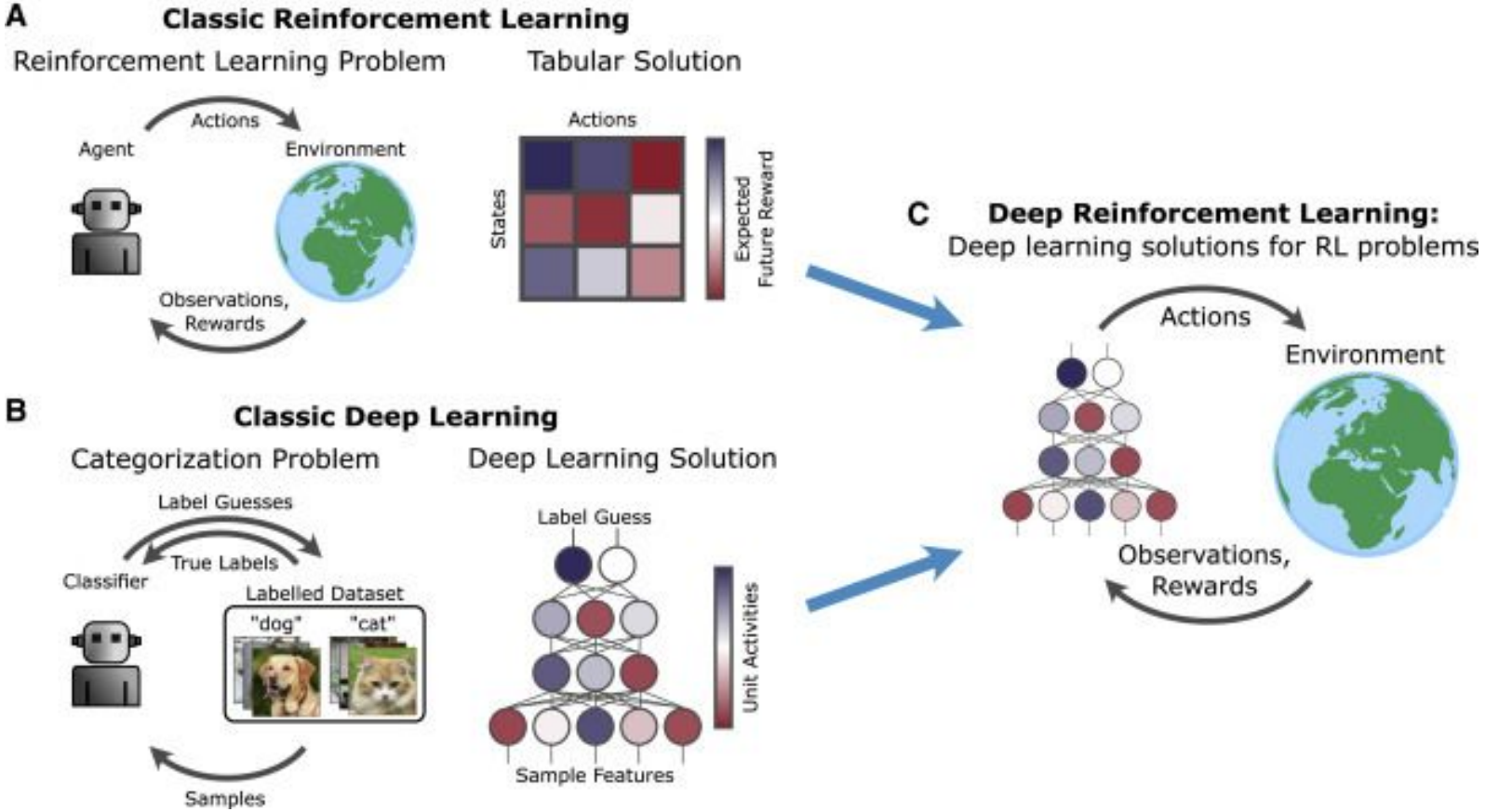
- So far we have just looked at tabular solution methods

  What if your state space is too large for tables, or continuous?

  $$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

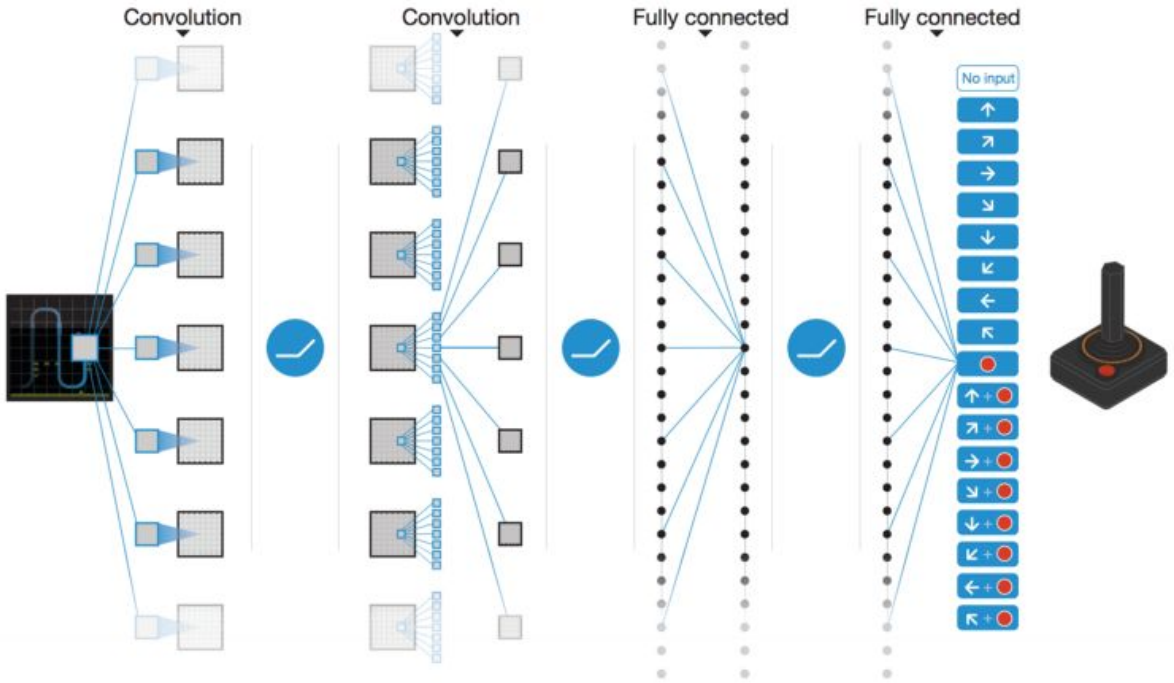
- Use function approximation
  - eg. linear combination of features, neural network, etc.

Approximate Solution Methods: Neural Networks



Approximate Solution Methods: Neural Networks

“DQN”

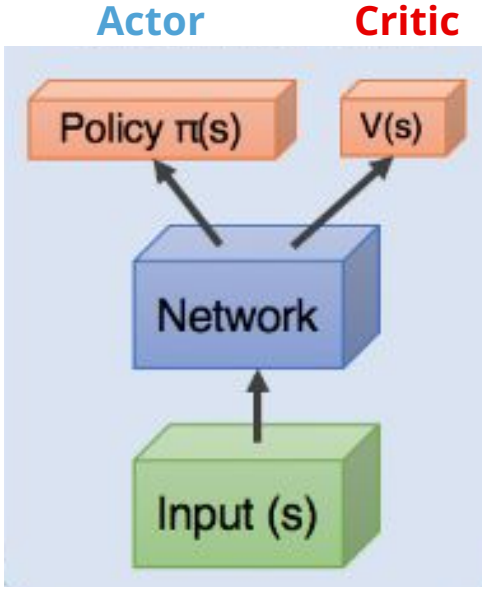


Mnih et al. (2015) *Nature*

Approximate Solution Methods: Neural Networks

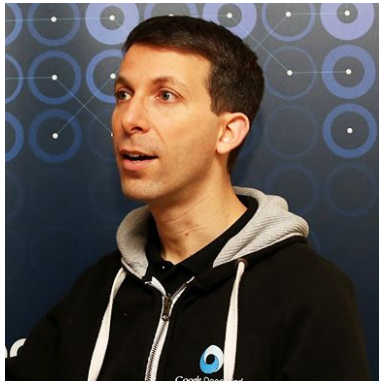
The Actor Critic Network

- Learn policy and value with same network
  - $\mathscr{L} = \underbrace{(R - V)\nabla_{W_u} log\ \pi}_{\text{Actor Loss}} + \underbrace{(R - V)^2}_{\text{Critic Loss}}$



More Materials on RL Fundamentals

Rich Sutton



David Silver

Algorithms for RL - Csaba Szepesvari

Arthur Juliani @ Medium