A Gentle* Introduction to Reinforcement Learning

Annik Carson

ISICNI2022

*ideally

Learning from Trial and Error

SURINAME	11,909km	➡️	40%
RUSSIA	3,981km	↙️	80%
EGYPT	2,991km	↗️	85%

Country, territory...

🌍 GUESS

Tutorial Goals

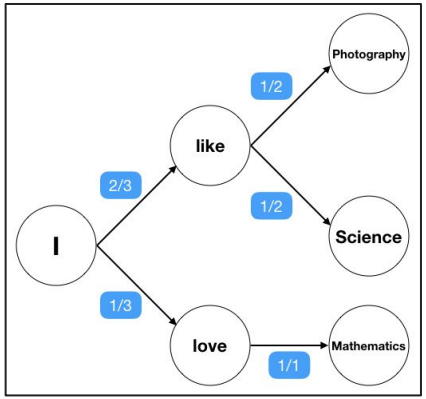
1. Reinforcement learning **is** learning from trial-and-error

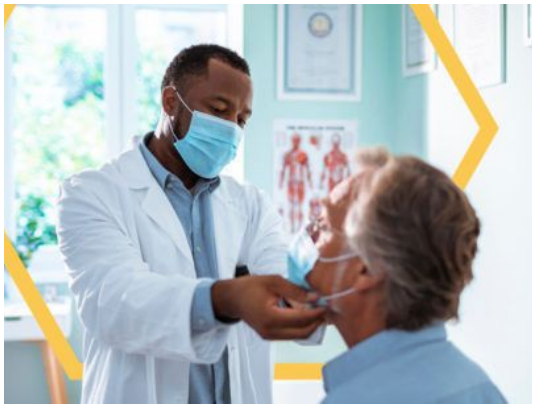
2. Game Architect or Game Player: What Goes Into a RL problem
   - Markov Decision Processes
   - Environments
   - Agents

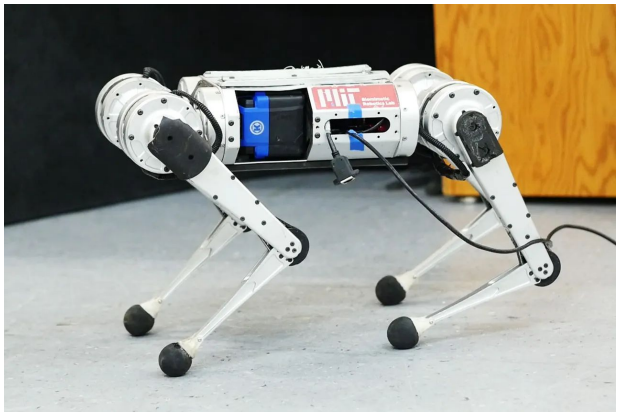
3. (Some) Approaches to Solving RL Problems
   - Types of agents
   - Tabular Solutions
   - Approximate Solutions (just some hints)

What is Reinforcement Learning?

What is RL Used For?

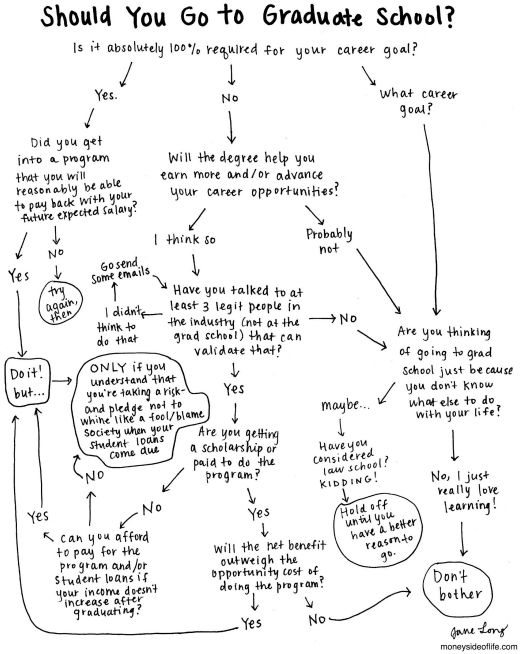




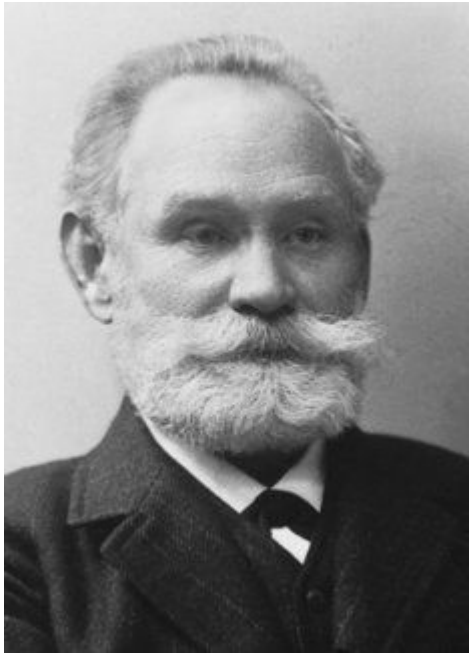


We All Have This Problem

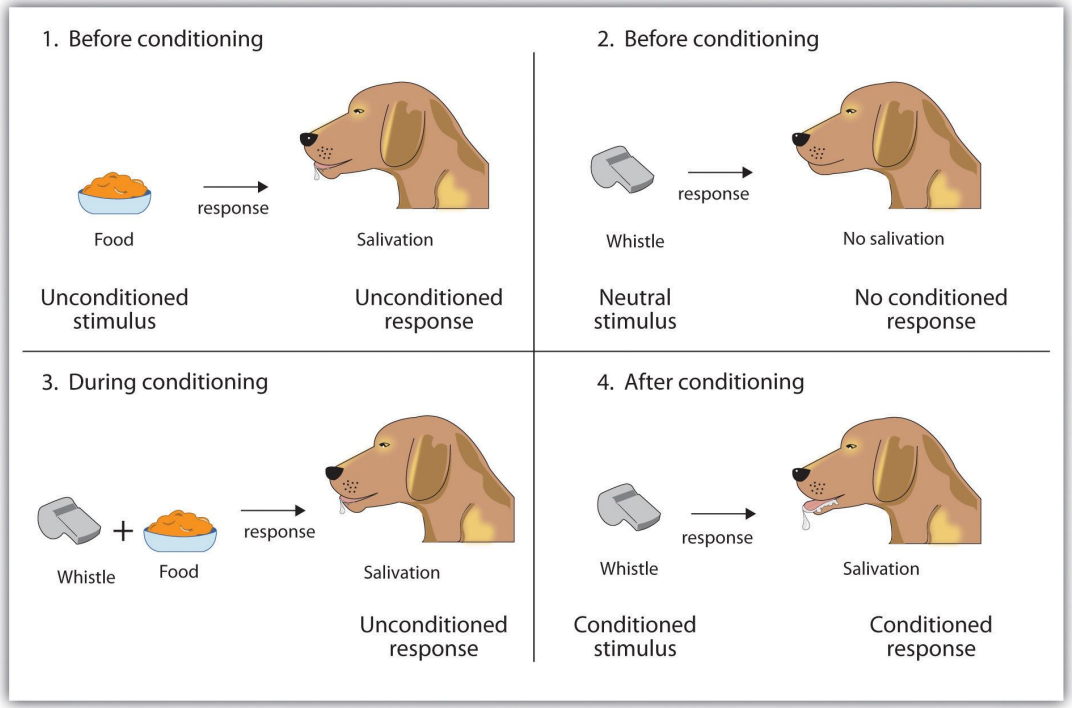
“What should I do?”



How Does RL Relate to Neuroscience?



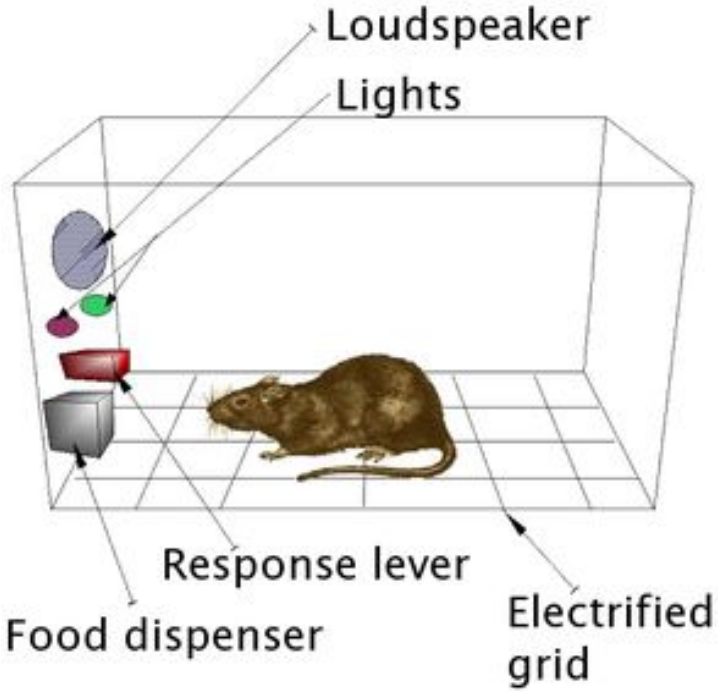
Ivan Pavlov



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

How Does RL Relate to Neuroscience?

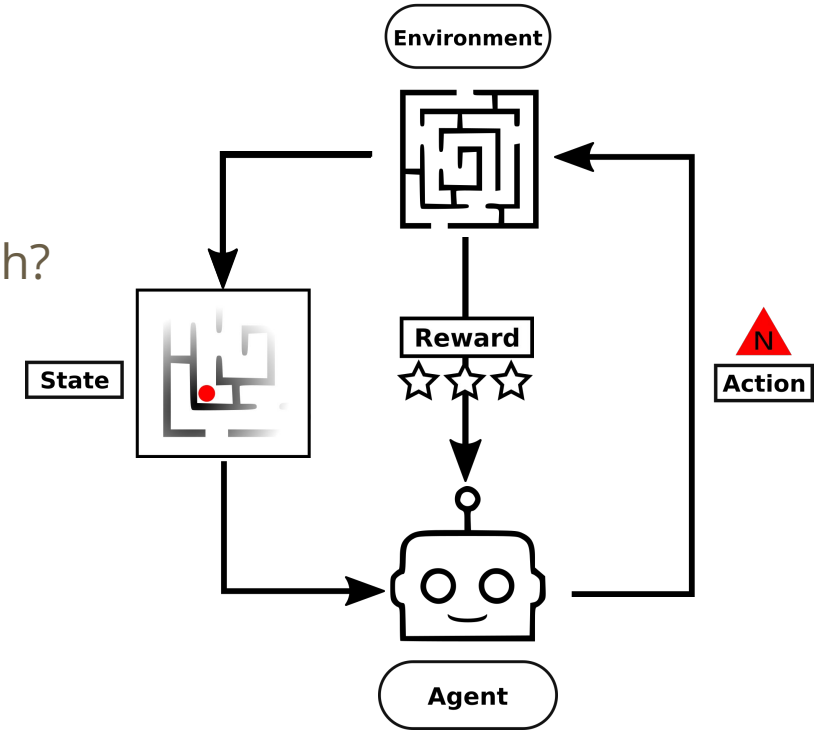
BF Skinner



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

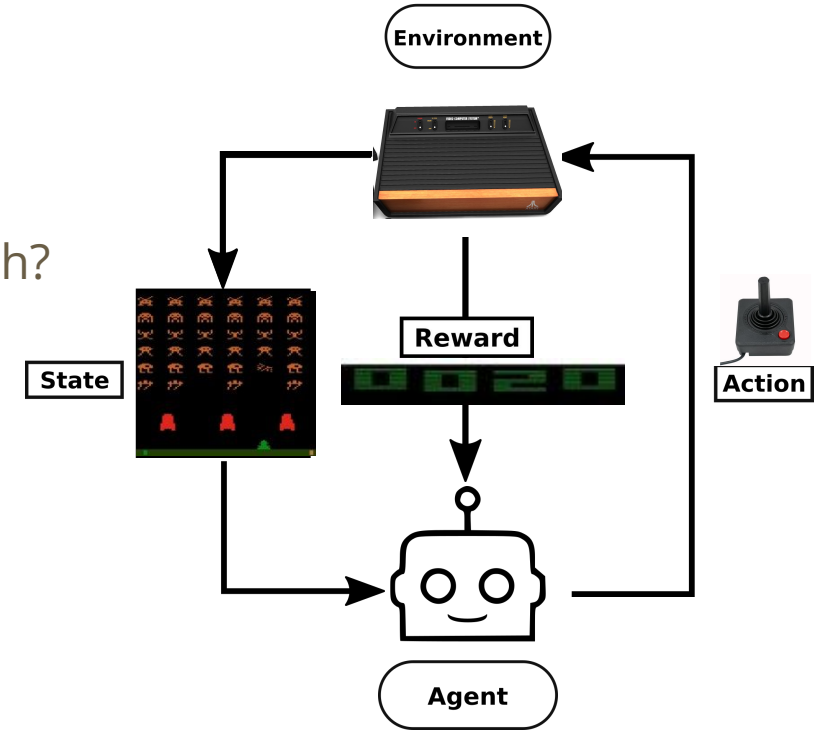
Reinforcement Learning = Optimizing Behaviour

- How should agents act **optimally**?
  - Optimality ~ maximizing cumulative reward
- What information is available to learn with?
  - Trial and error interaction with the world



Reinforcement Learning = Optimizing Behaviour

- How should agents act **optimally**?
  - Optimality ~ maximizing cumulative reward
- What information is available to learn with?
  - Trial and error interaction with the world



David Silver, 2015

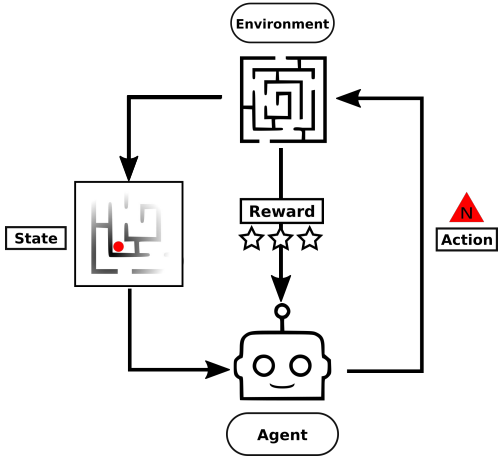
How do we represent these tasks and their solutions?

Markov Decision Processes

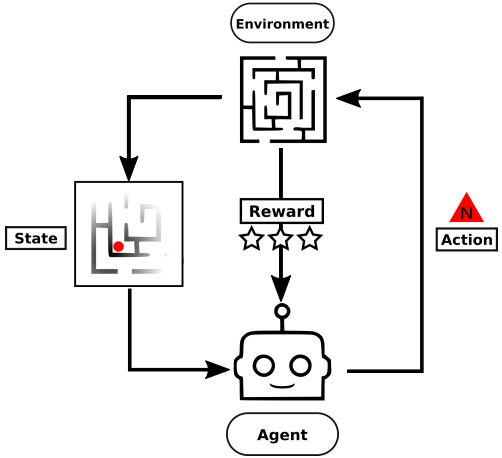
- RL problems are formalized as **Markov Decision Processes**
  - The Markov property states that

    the future is independent of the past given the present:

$$\Pr[s_{t+1}|s_1, ..., s_t] = \Pr[s_{t+1}|s_t]$$



Markov Decision Processes



- RL problems are formalized as **Markov Decision Processes**
  - The Markov property states that
    the future is independent of the past given the present:

$$\Pr[s_{t+1}|s_1, ..., s_t] = \Pr[s_{t+1}|s_t]$$

- MDPs are defined by the following variables:
  - $\mathcal{S}$, a finite set of states
  - $\mathcal{A}$, a finite set of actions
  - $\mathcal{P}$, state transition probabilities

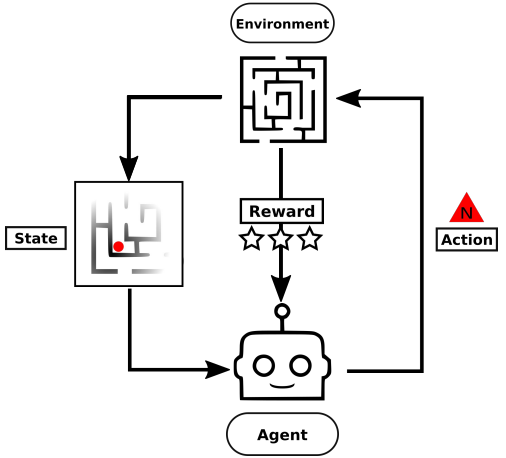
$$\mathcal{P}^a_{ss'} = \Pr[S_{t+1} = s'|S_t = s, A_t = a]$$

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

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

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

Markov Decision Processes



- RL problems are formalized as **Markov Decision Processes**
  - The Markov property states that
    the future is independent of the past given the present:

$$\Pr[s_{t+1}|s_1, ..., s_t] = \Pr[s_{t+1}|s_t]$$

- MDPs are defined by the following variables:
  - $\mathcal{S}$, a finite set of states
  - $\mathcal{A}$, a finite set of actions
  - $\mathcal{P}$, state transition probabilities

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

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

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

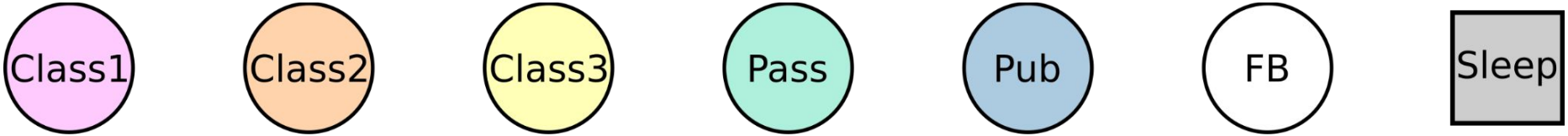
  - $\gamma \in [0, 1]$, a discount factor (weighing importance of immediate vs future reward)

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

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

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

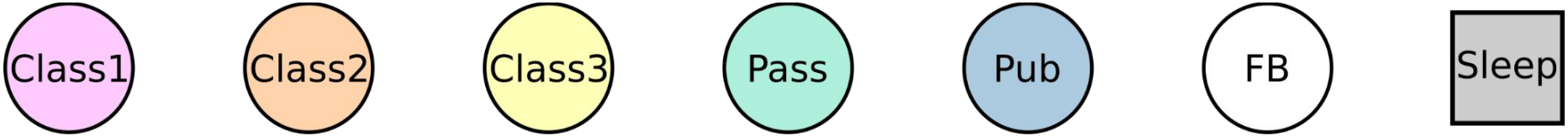
Exercise 1



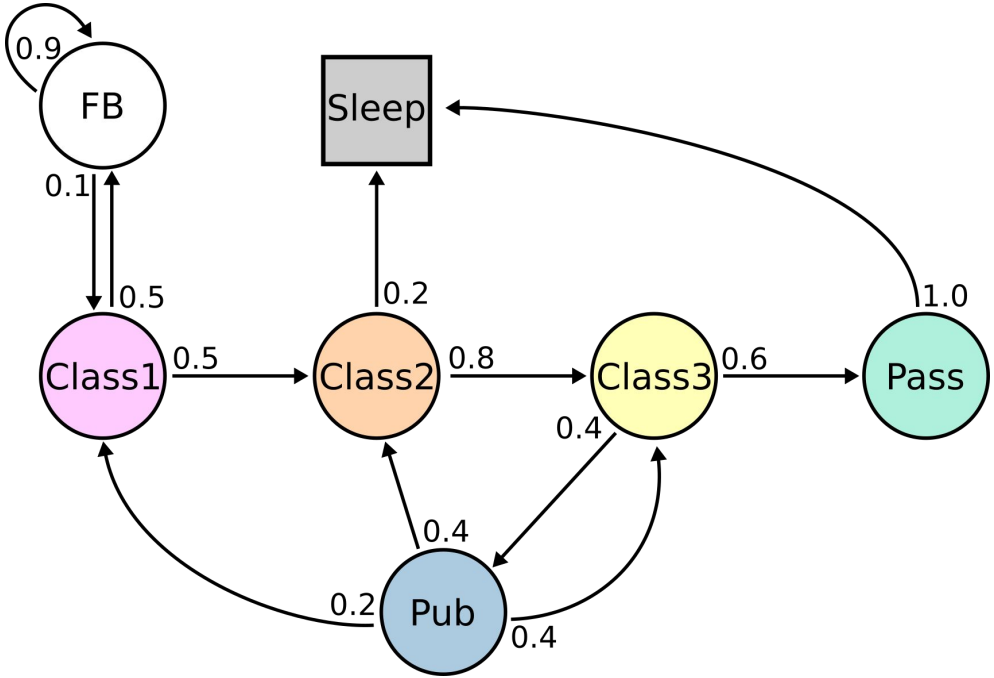
An environment is a **set of states** and

the **rules** that specify moves between them

States



State Transitions



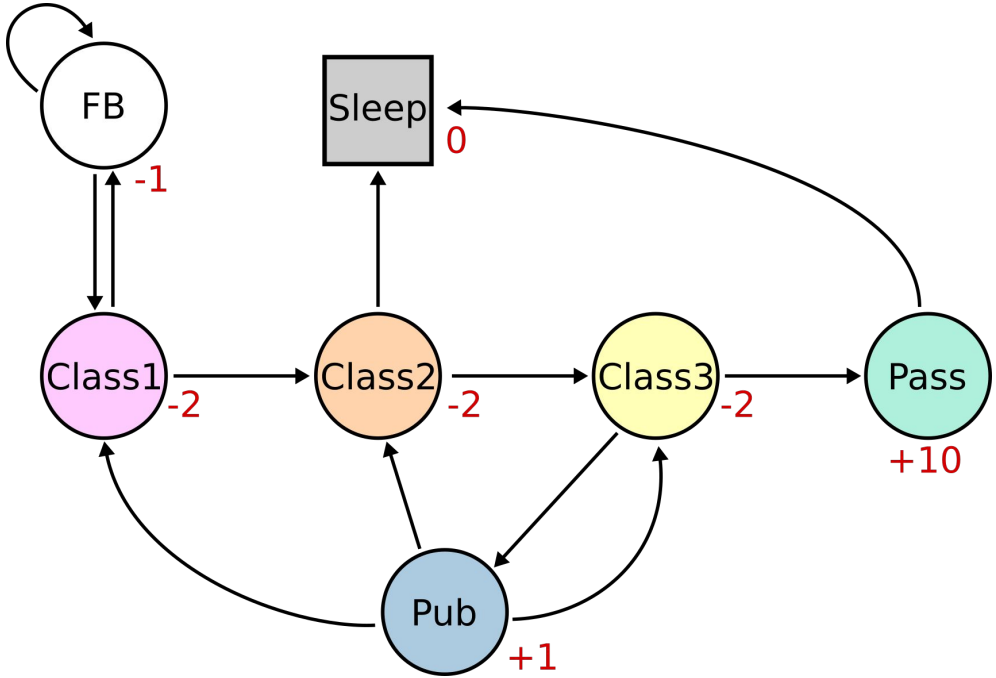
$$\mathcal{P}^{a}_{ss'} = \Pr[S_{t+1} = s' | S_t = s, A_t = a]$$

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

Rewards



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

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

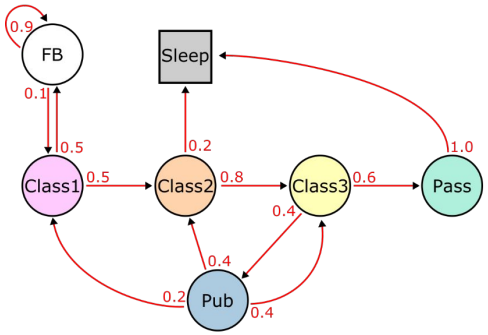
$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_{t+1} | S_t = s, S_{t+1} = s', A_t = a\right]$$

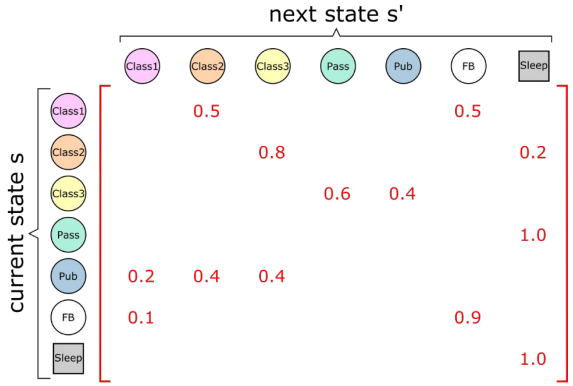
Class1	Class2	Class3	Pass	Pub	FB	Sleep
-2	-2	-2	+10	+1	-1	0

Actions & Transitions

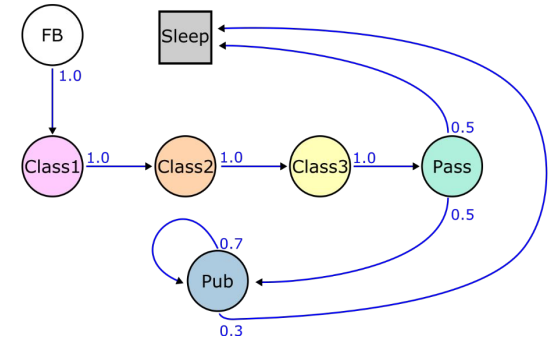
$$\mathcal{P}^{a}_{ss'} = \Pr[S_{t+1} = s' | S_t = s, A_t = a]$$

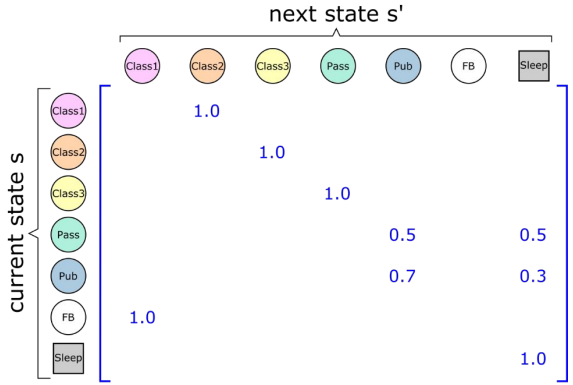
Chill





Study



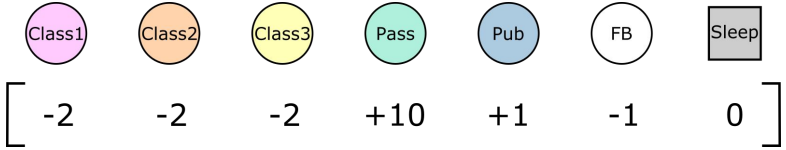


Markov Decision Processes

- $\mathcal{S}$, 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)

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')

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**

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

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

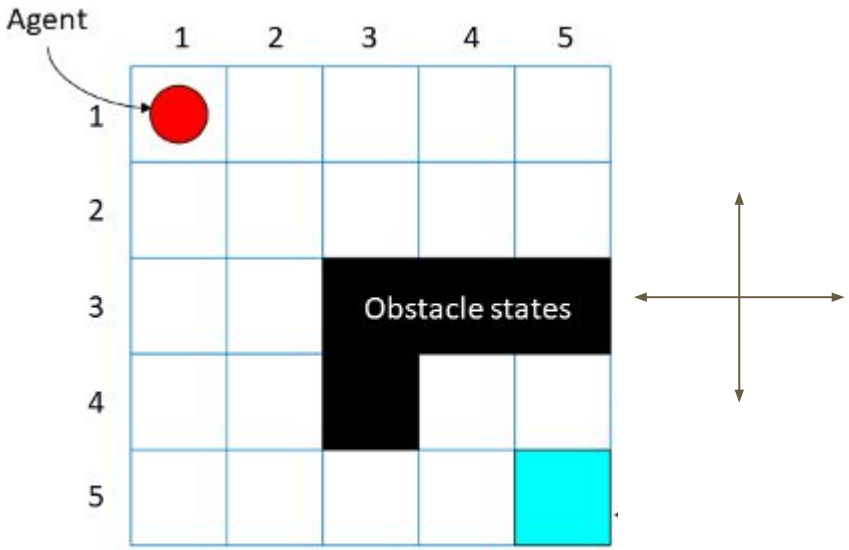
A policy (π) is a **distribution over actions given states**

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

BREAK

How do we represent these tasks and their solutions?

Specifying the Environment



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

the **rules** that specify moves between them

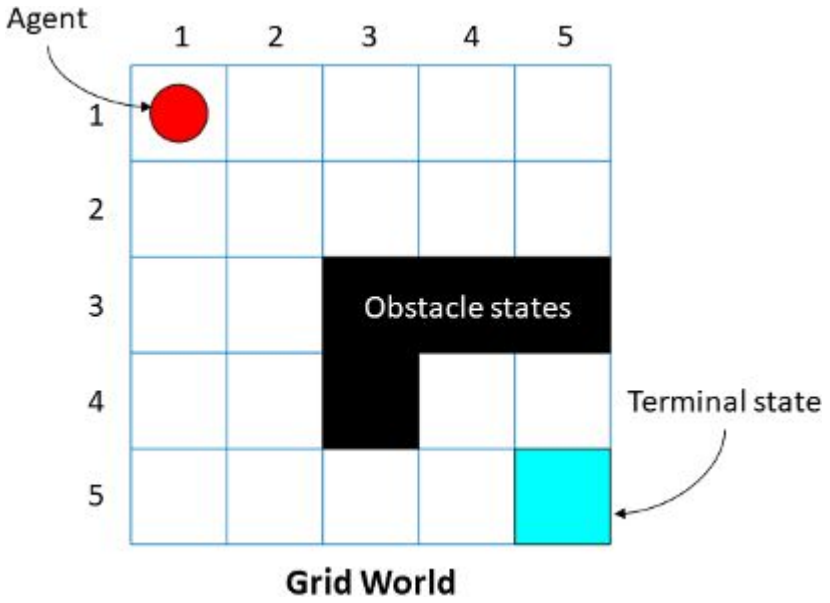
Exercise 2

Environment

State Ex. (1,1), (5,5)

Action Ex. “Down”

Reward Ex. -1 or +10



Environment: Rewards

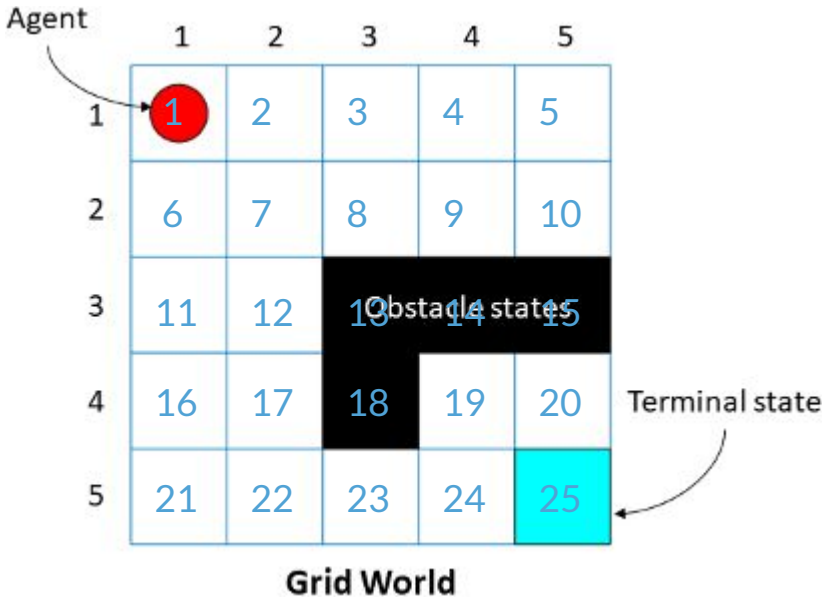
Reward function R(state)

- R(1,1) = -1
- R(5,5) = +10

R(s) for this environment can be represented by a vector:

-1	-1	-1	-1	…	+10

Q: How long is the vector R(s)?

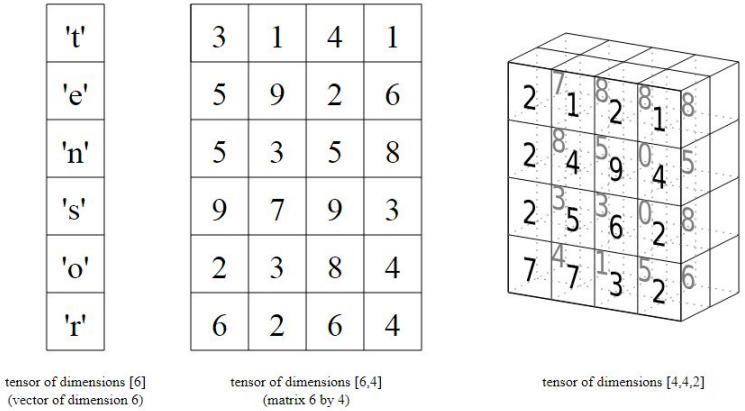


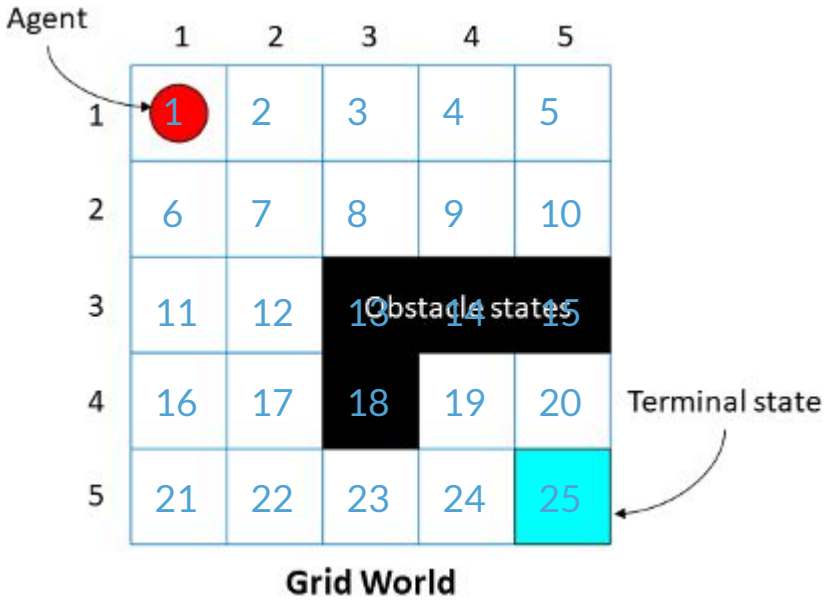
Environment: State Transition Probability

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

A rule for what moves are allowed

→ Can be represented by a 3D tensor



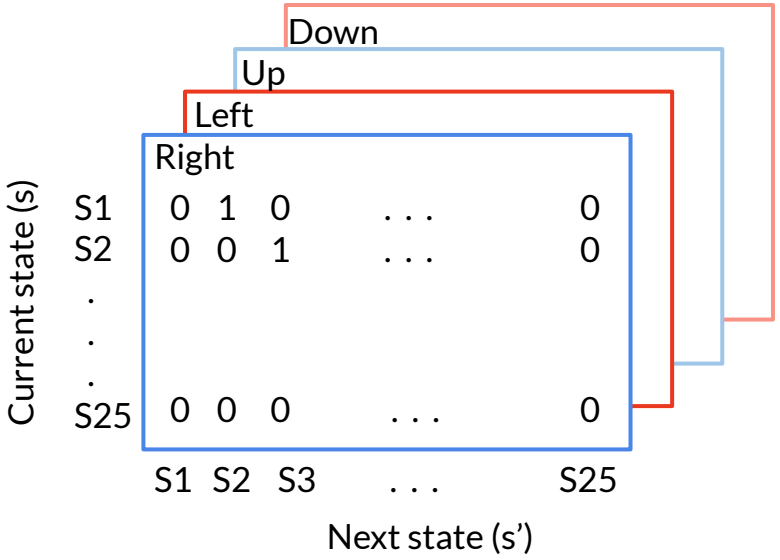


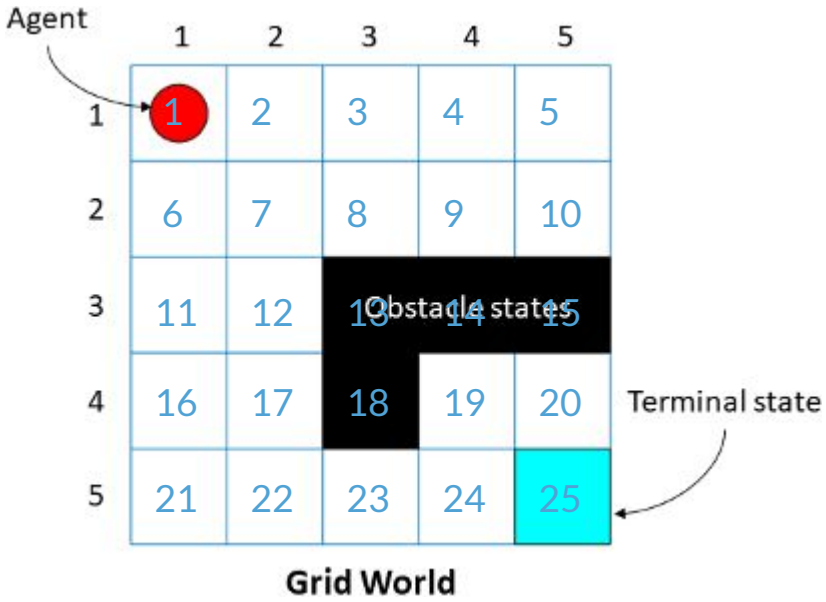
Environment: State Transition Probability

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

A rule for what moves are allowed

→ Can be represented by a 3D tensor





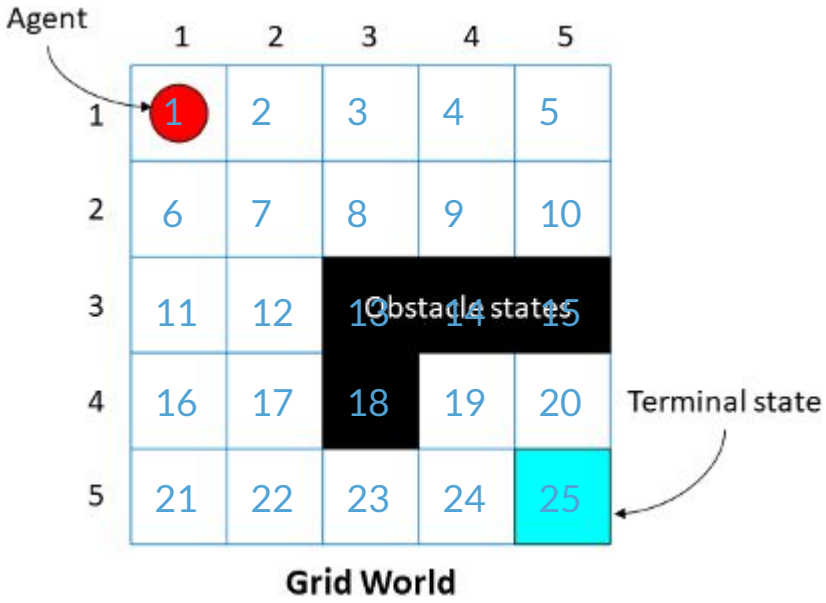
Environment: State Transition Probability

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

A rule for what moves are allowed

For state 1 (i.e. (1,1)):

	S1	S2	S3	S4	S5	S6	. . . S25
Down	[0.,	0.,	0.,	0.,	0.,	1.,	..., 0.]
Up	[1.,	0.,	0.,	0.,	0.,	0.,	..., 0.]
Left	[1.,	0.,	0.,	0.,	0.,	0.,	..., 0.]
Right	[0.,	1.,	0.,	0.,	0.,	0.,	..., 0.]



Environment: State Transition Probability

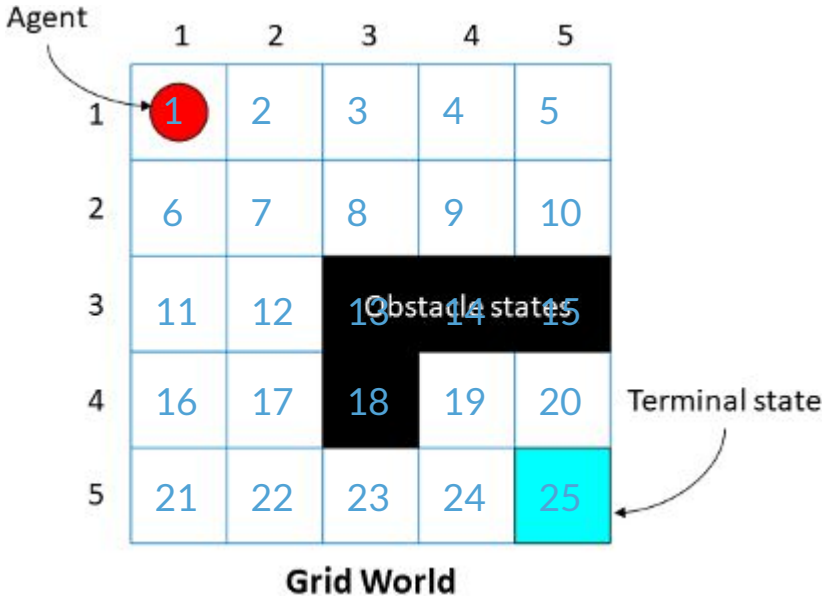
A different transition function:

Environment has wrapping edges
ex. pacman

For state 1 (i.e. (1,1)):

	S1	S2	S3	S4	S5	S6	. . .	S25
Down	[0.,	0.,	0.,	0.,	0.,	1.,	...,	0.]
Up	[0.,	0.,	0.,	0.,	0.,	0.,	...,	0.]
Left	[0.,	0.,	0.,	0.,	1.,	0.,	...,	0.]
Right	[0.,	1.,	0.,	0.,	0.,	0.,	...,	0.]

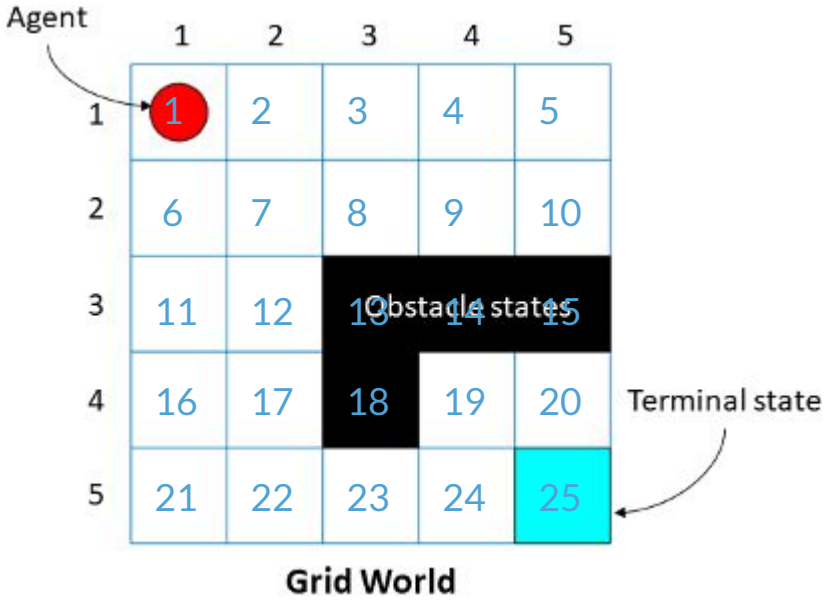
There should be a 1 in the matrix at position [1, 20]
i.e. for action up the probability of transitioning to state 21 is 1



Environment: State Transition Probability

A different transition function:

Environment has a strong wind blowing to the right

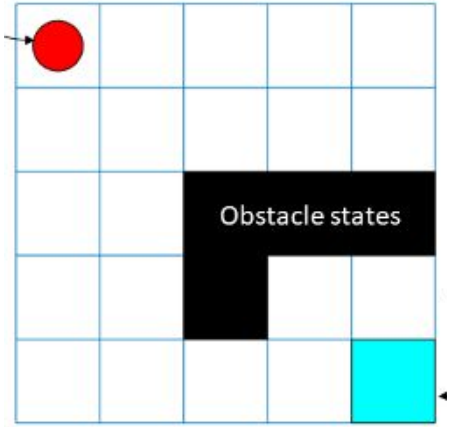


For state 1 (i.e. (1,1)):

	S1	S2	S3	S4	S5	S6	S7	S8	. . .	S25
Down	0.	0.	0.	0.	0.	0.5	0.5	0.	...	0.
Up	0.5	0.	0.	0.	0.	0.	0.	0.	...	0.
Left	1.	0.	0.	0.	0.	0.	0.	0.	...	0.
Right	0.	0.3	0.6	0.1	0.	0.	0.	0.	...	0.

The probability of leaving a state = 1
→ Sum of values along the row = 1

Markov Decision Processes



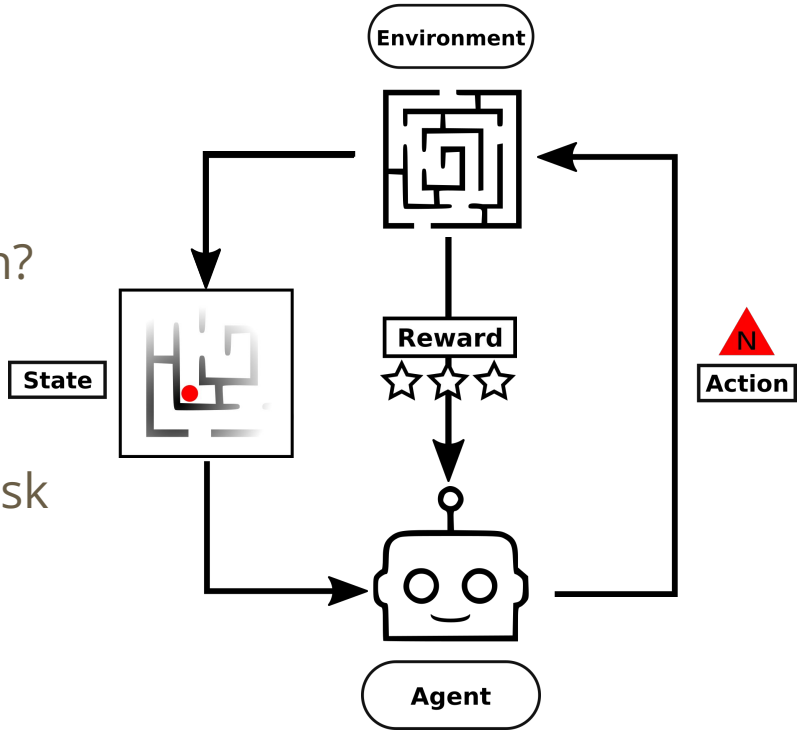
- $\mathcal{S}$, a finite set of states **{(0,0), (0,1), (0,2), ... (5,5)}**
- $\mathcal{A}$, a finite set of actions **{Up, Down, Left, Right}**
- $\mathcal{P}$, state transition probabilities (one matrix for each action)

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

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

What is Reinforcement Learning (RL)?

- How should agents act **optimally**?
  - Optimality ~ maximizing cumulative reward
- What information is available to learn with?
  - Trial and error interaction with the world
- So far, we have just set up a formalized task
  - How do we go about solving it?



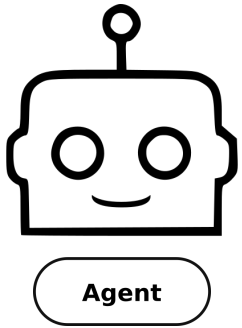
BREAK

How do we represent these tasks and their solutions?

Specifying the Agent

What's In An Agent?

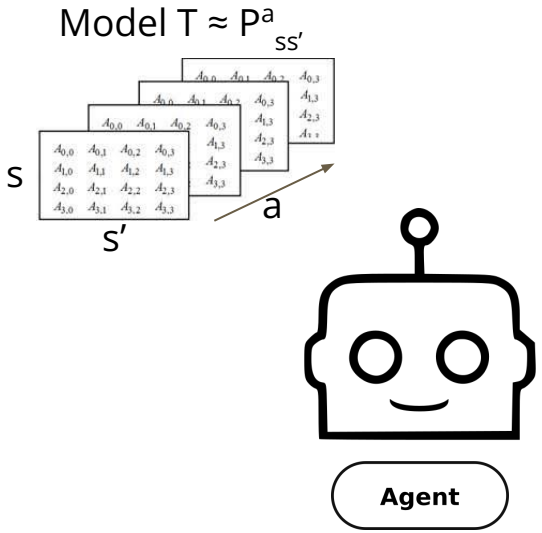
An agent is made up of one or more of these pieces:



What's In An Agent?

An agent is made up of one or more of these pieces:

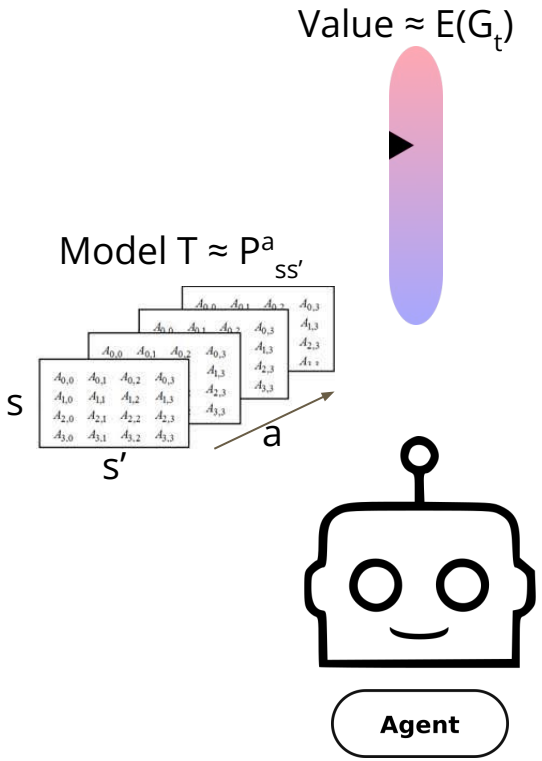
- Model
  - A representation of the environment dynamics



What's In An Agent?

An agent is made up of one or more of these pieces:

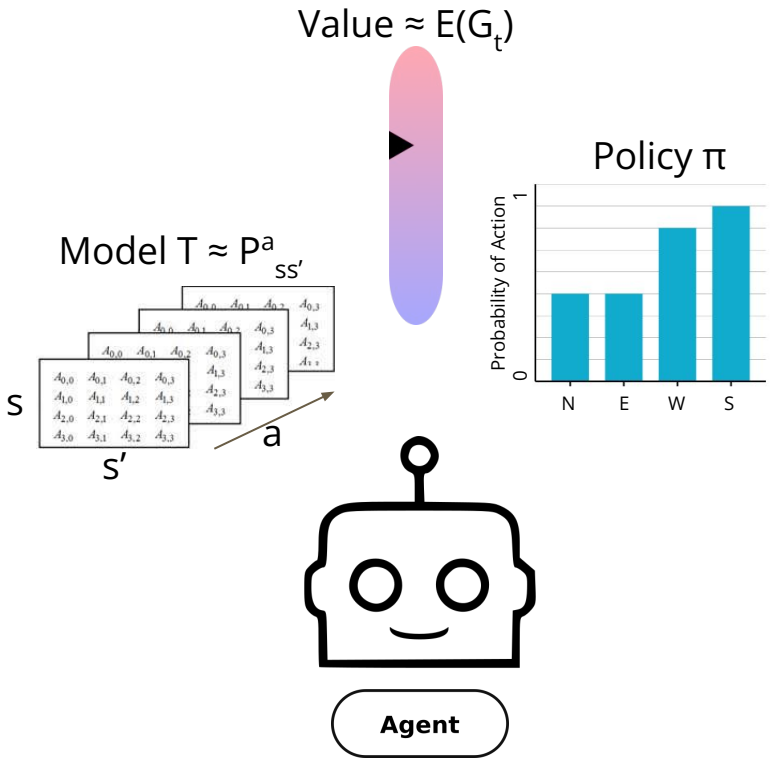
- Model
  - A representation of the environment dynamics
- Value Function
  - An estimate how much reward is expected over time for each state and/or action



What's In An Agent?

An agent is made up of one or more of these pieces:

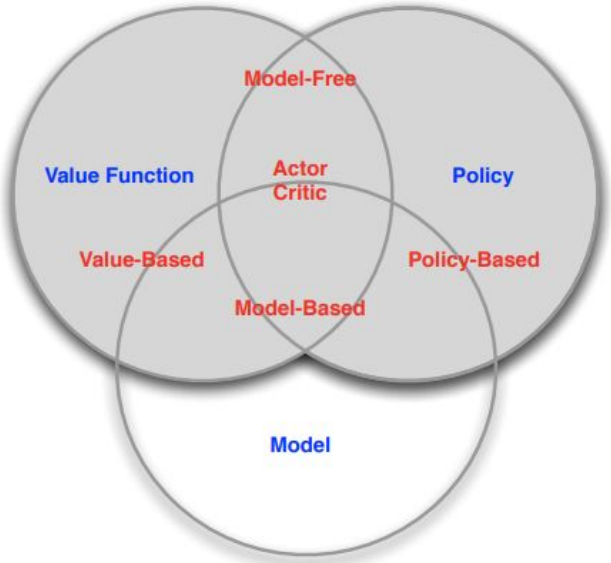
- Model
  - A representation of the environment dynamics
- Value Function
  - An estimate how much reward is expected over time for each state and/or action
- Policy
  - A function which maps states onto actions
  - A (conditional) probability distribution
    - gives the probability of selecting each action given current state



What's In An Agent?

An agent is made up of one or more of these pieces:

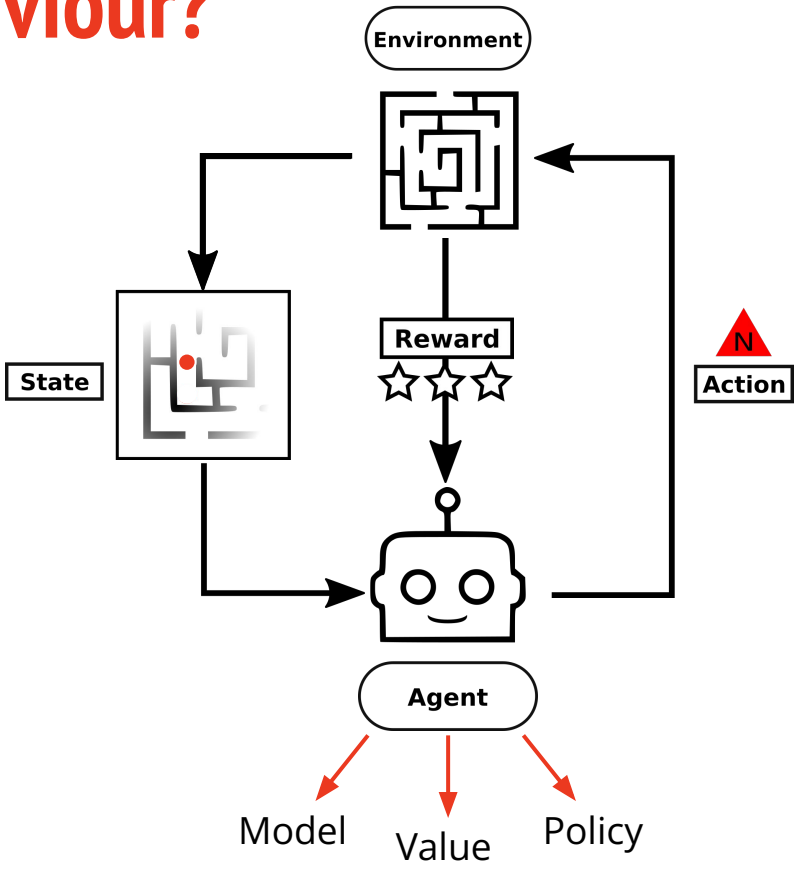
- Model
  - A representation of the environment dynamics
- Value Function
  - An estimate how much reward is expected over time for each state and/or action
- Policy
  - A function which maps states onto actions
  - A (conditional) probability distribution
    - gives the probability of selecting each action given current state



David Silver, 2015

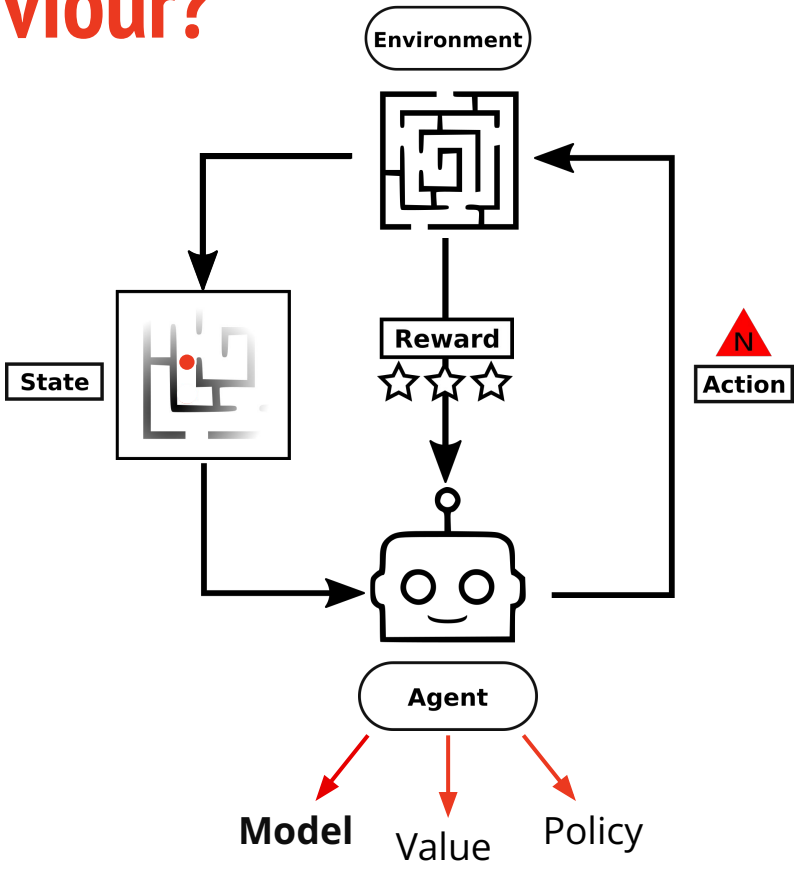
How to Learn Intelligent Behaviour?

- Trial and error learning
- Traditional RL categorized as
  - Model-based (~ goal directed)
  - Model-free (~ habitual)



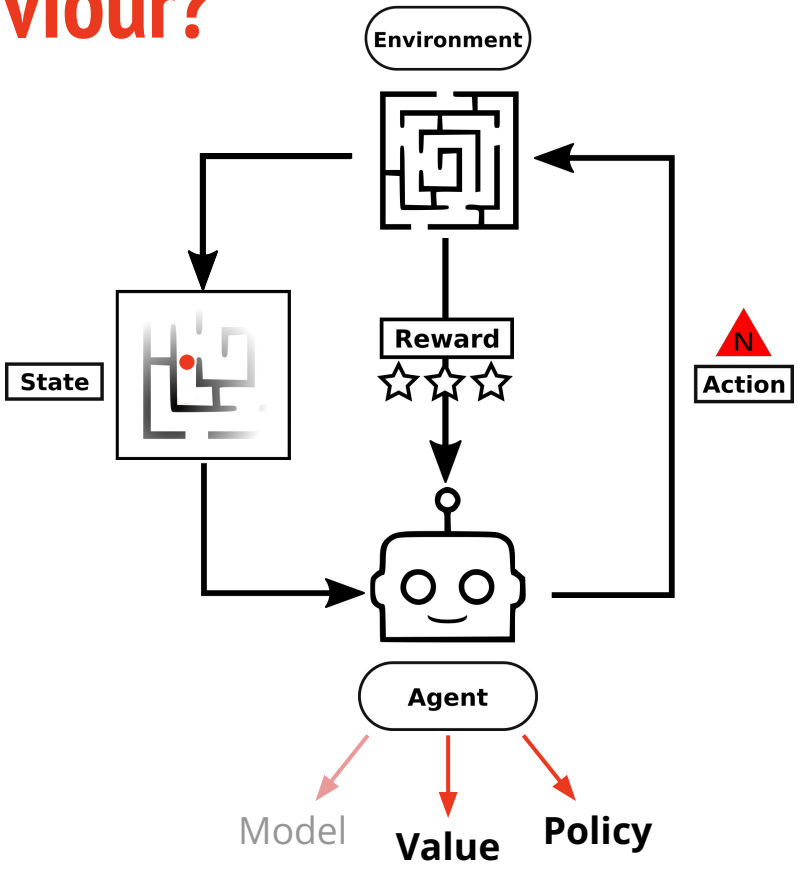
How to Learn Intelligent Behaviour?

- Trial and error learning
- Traditional RL categorized as
  - **Model-based** (~ goal directed)
  - Model-free (~ habitual)



How to Learn Intelligent Behaviour?

- Trial and error learning
- Traditional RL categorized as
  - Model-based (~ goal directed)
  - **Model-free** (~ habitual)



Value & Policy Functions

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

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

Recall: $\mathbb{E} = p_1 x_1 + p_2 x_2 + \ldots + p_n x_n$

Value & Policy Functions

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

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

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

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

Value & Policy Functions

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

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

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

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

Ex. Greedy policy:

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

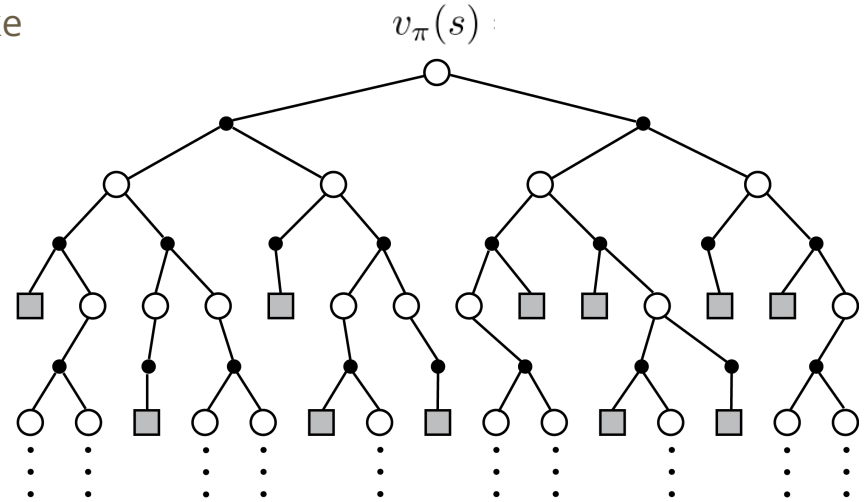
Ex. ε-Greedy policy:

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

How to Compute Value of Current State?

- Brute force
  - Compute value of all possible traces and take weighted sum

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



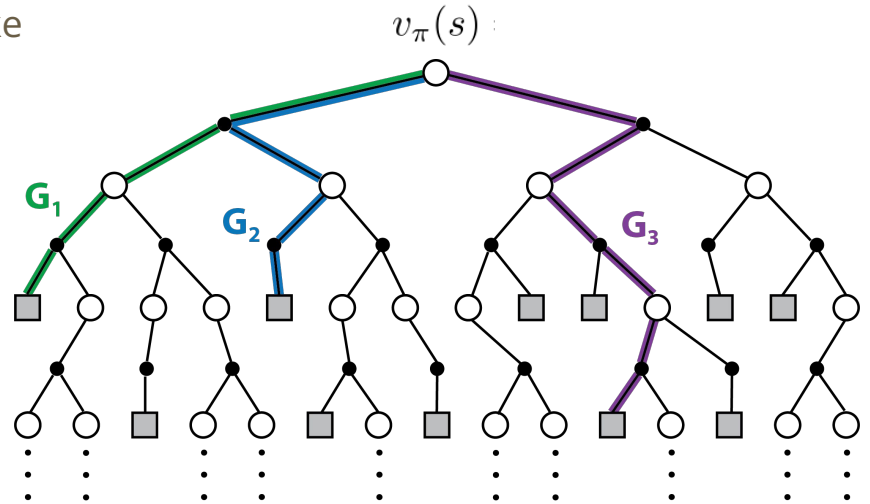
How to Compute Value of Current State?

- Brute force
  - Compute value of all possible traces and take weighted sum

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

- Sampling
  - Pick traces randomly and take empirical average over sampled traces

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



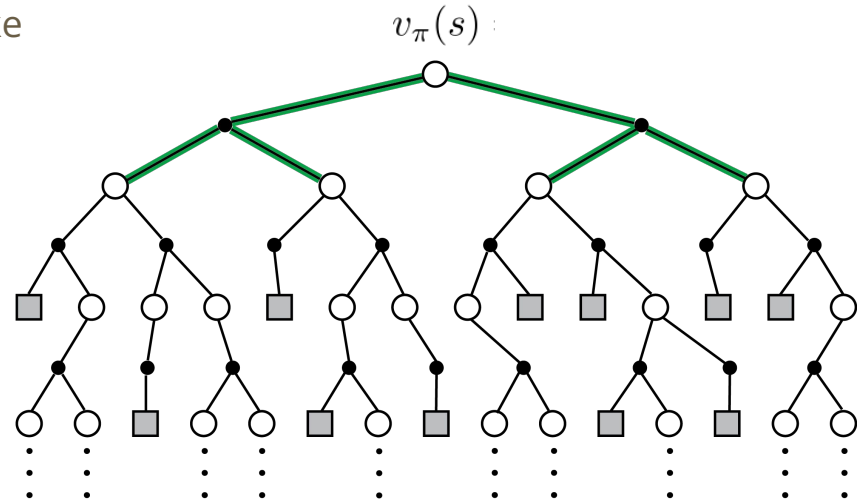
How to Compute Value of Current State?

- Brute force
  - Compute value of all possible traces and take weighted sum

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

- Sampling
  - Pick traces randomly and take empirical average over sampled traces

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



- Bootstrapping
  - Estimate from values of successive states

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

Using Value to Construct Policy

- How do we judge **value**?
  - The **state-value function $v_\pi(s)$** is the expected return starting from state s and following policy π thereafter

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

  - The **state-action-value function $q_\pi(s,a)$** is the expected return starting from state s, taking action a, then following policy π thereafter

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

- Some **policies** based on these value functions:

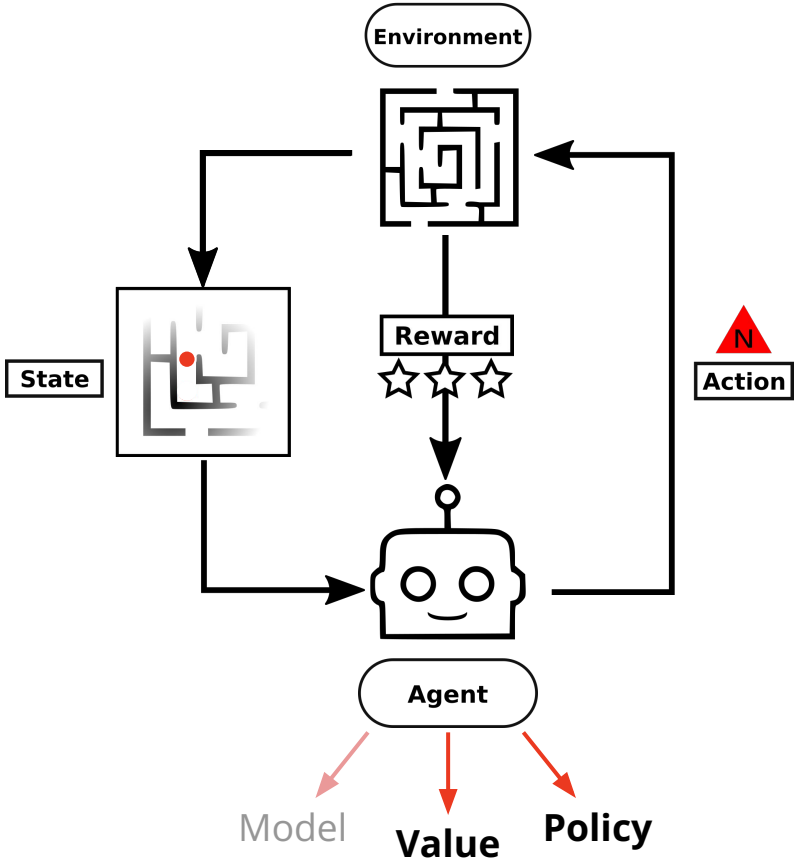
greedy: $\pi(s) = \arg\max_{a \in \mathcal{A}} q_\pi(s,a)$

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

BREAK

Solution Methods

- Tabular Methods
  - ~~Dynamic Programming~~
  - Monte Carlo
  - Temporal Difference
  - ~~N-step TD bootstrapping~~
- Approximate solution methods
  - Deep Q Network (DQN)
  - Policy-gradient methods
  - Actor-Critic



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: Monte Carlo

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

Error – how much to update by

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

Empirical return G_t (i.e. sampled)

New (Updated) Value

Old Value

Learning Rate – how much error contributes to new value

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

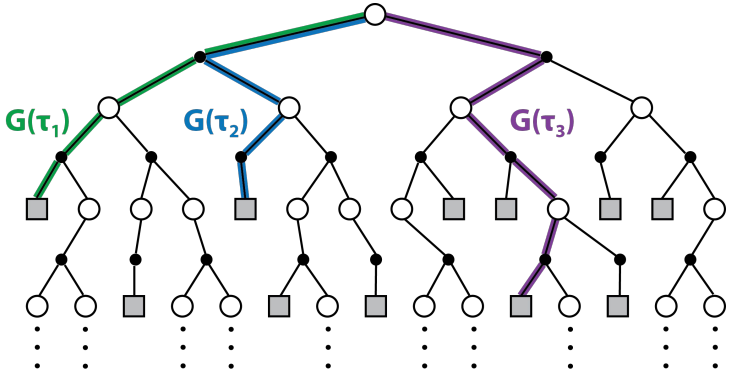
Learning Value: Monte Carlo

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

Learn directly from episodes of experience — upon episode completion, update the value of all states (or state-actions) visited [sampling]

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

- **General idea:** run some episodes and estimate value of a state as empirical mean of the return starting from that state



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

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: Monte Carlo

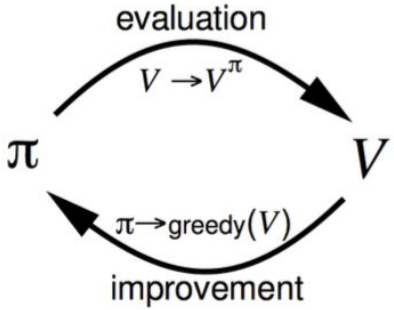
- Evaluation: for each episode, update the estimate of the value

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

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

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

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



Learning Value: Monte Carlo

- Run through an episode making ε-greedy choices, record each (s, a, r)
  - At the end, compute V(s) or Q(s,a) explicitly
- Repeat, each time average the values of the V(s)/Q(s,a) we have so far
  - This way, we know which state/action gave the highest value of anything we encountered

PROS	CONS
Does not require knowledge/model of environment (learns from interaction) Can focus on one region of interest without having to evaluate the rest of state space	Requires episodic (terminating) environments Requires complete episodes, can't bootstrap (can't estimate values of successive states) Return only known at the end of episode, so very slow for long episodes

Exercise 3

Learning Value: Temporal Difference

Error – how much to update by

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

New (Updated) Value

Old Value

Learning Rate – how much error contributes to new value

Estimate of return G_t

Learning Value: Temporal Difference

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

Temporal difference (TD) learning combines efficient value estimation by bootstrapping along with sampling from episodes.

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

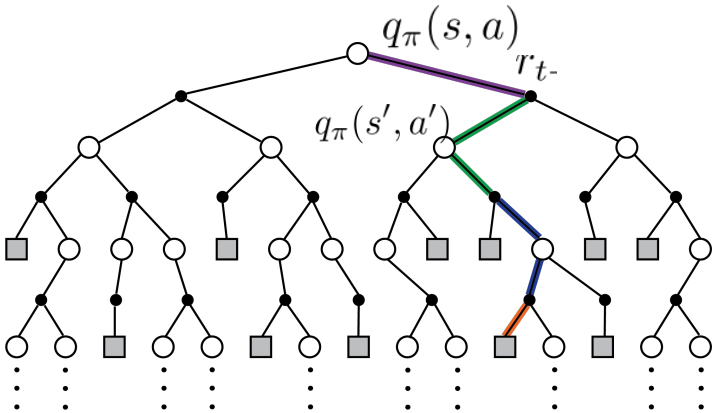
Learning Value: Temporal Difference

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

Temporal difference (TD) learning combines efficient value estimation by bootstrapping along with sampling from episodes.

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

- **General idea:** Update your estimate of currnet value using your estimate of the next value plus some empirical evidence (observed reward)



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

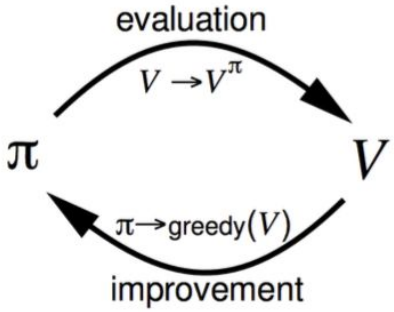
- Evaluation: for each episode, update the estimate of the value

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

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

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

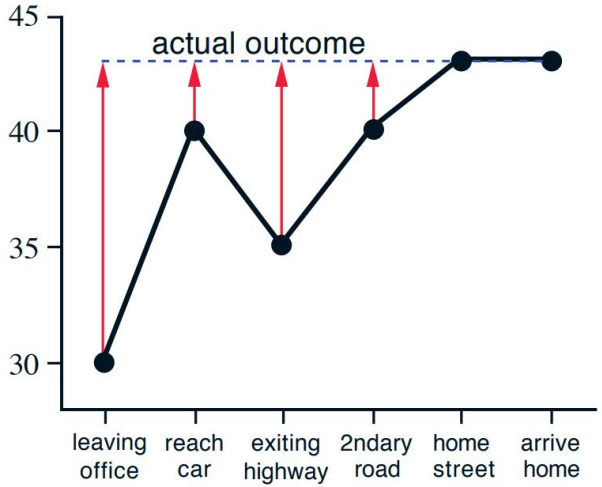
$$\pi[a|s] = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_{a \in \mathcal{A}} q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$



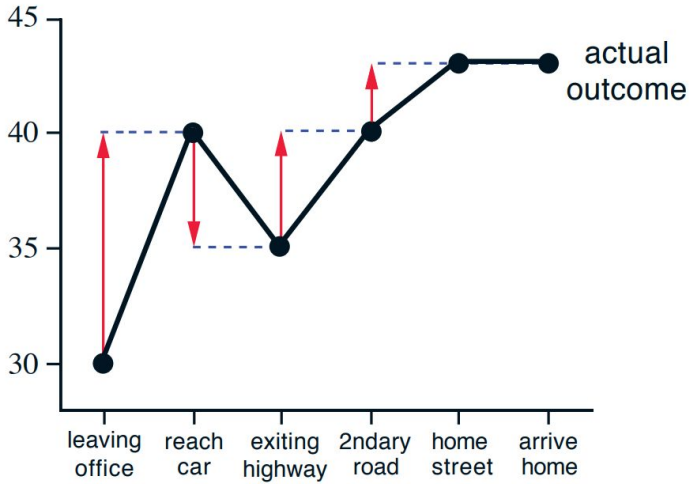
Example: Driving Home

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

Goal: learn v_π online from experiences under policy π

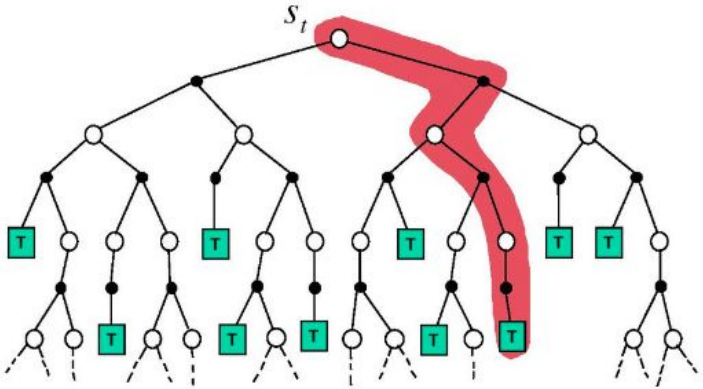
Incremental MC	**Simplest TD**
Update $V(s_t)$ toward actual return G_t	Update $V(s_t)$ toward estimated return $R_{t+1} + \gamma V(s_{t+1})$
$V(s_t) \leftarrow V(s_t) + \alpha (G_t - V(s_t))$	$V(s_t) \leftarrow V(s_t) + \alpha (R_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

Temporal Difference vs Monte Carlo

Goal: learn v_π online from experiences under policy π

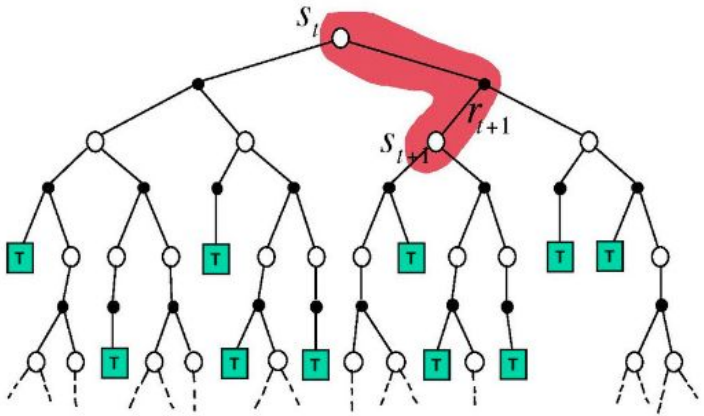
Monte-Carlo Backup

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



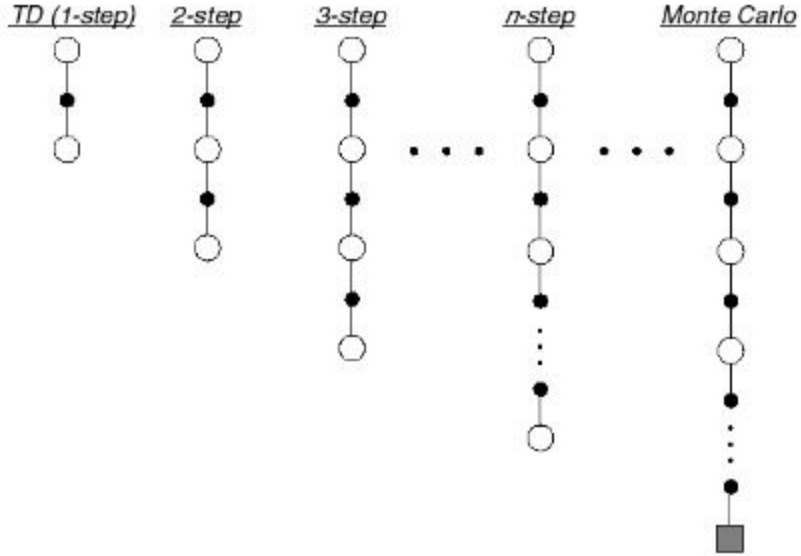
Temporal-Difference Backup

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



In Between MC & TD: TD-lambda

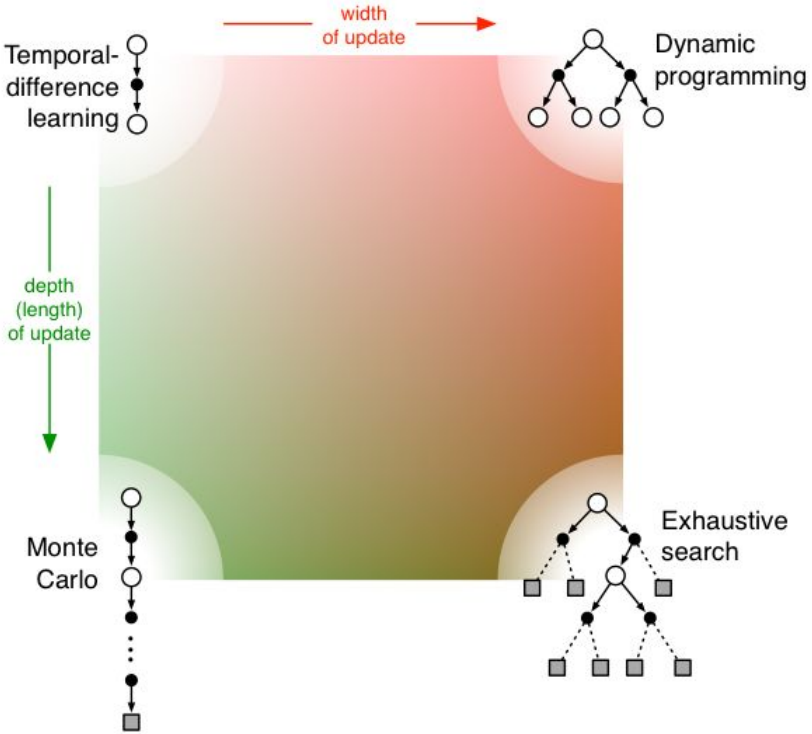
Let TD target look n steps into the future



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

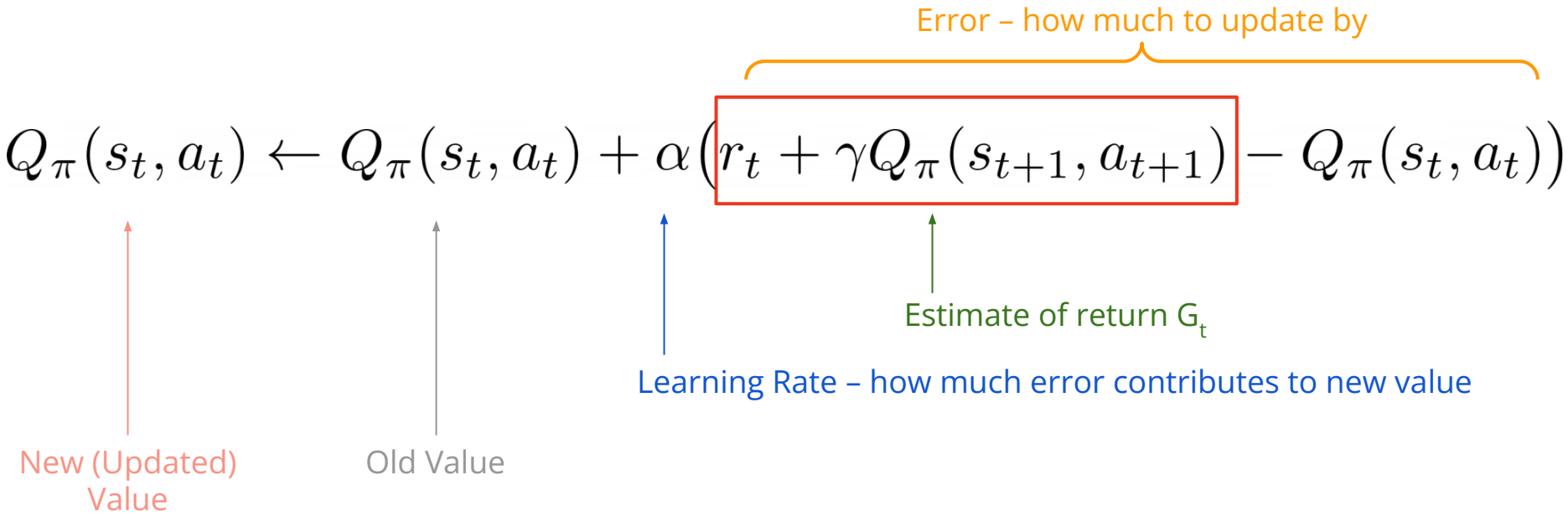
- Generate your target from the **same** policy you used to select actions
  - **ON** policy

- Generate your target from a **different** policy than the one you used to select actions
  - **Off** 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

Exercise 4

Learning Value: Temporal Difference

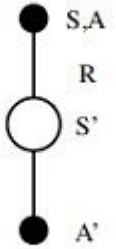


On-Policy TD Learning: SARSA

- The agent collects info over two steps: **S**tate/**A**ction/**R**eward/Next **S**tate/Next **A**ction
- Agent uses an ϵ-greedy policy for action selection in each step
- Update state (state-action) values by this temporal difference error which takes into account the states/actions sampled in step t and step t+1

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

SARSA (TD Method)



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

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

Off-Policy TD Learning: Q Learning

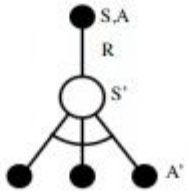
- The agent has two policies — the behavior policy $\mu(s)$ (e.g., ϵ-greedy), and the optimized policy $\pi(s)$ (e.g., greedy)

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

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

- This allows us to optimize a greedy policy (as in DP methods), but we don't have to worry about exploration (due to ϵ-greedy behavior policy)

Q-Learning (TD Method)



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

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

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

Approximate Solution Methods

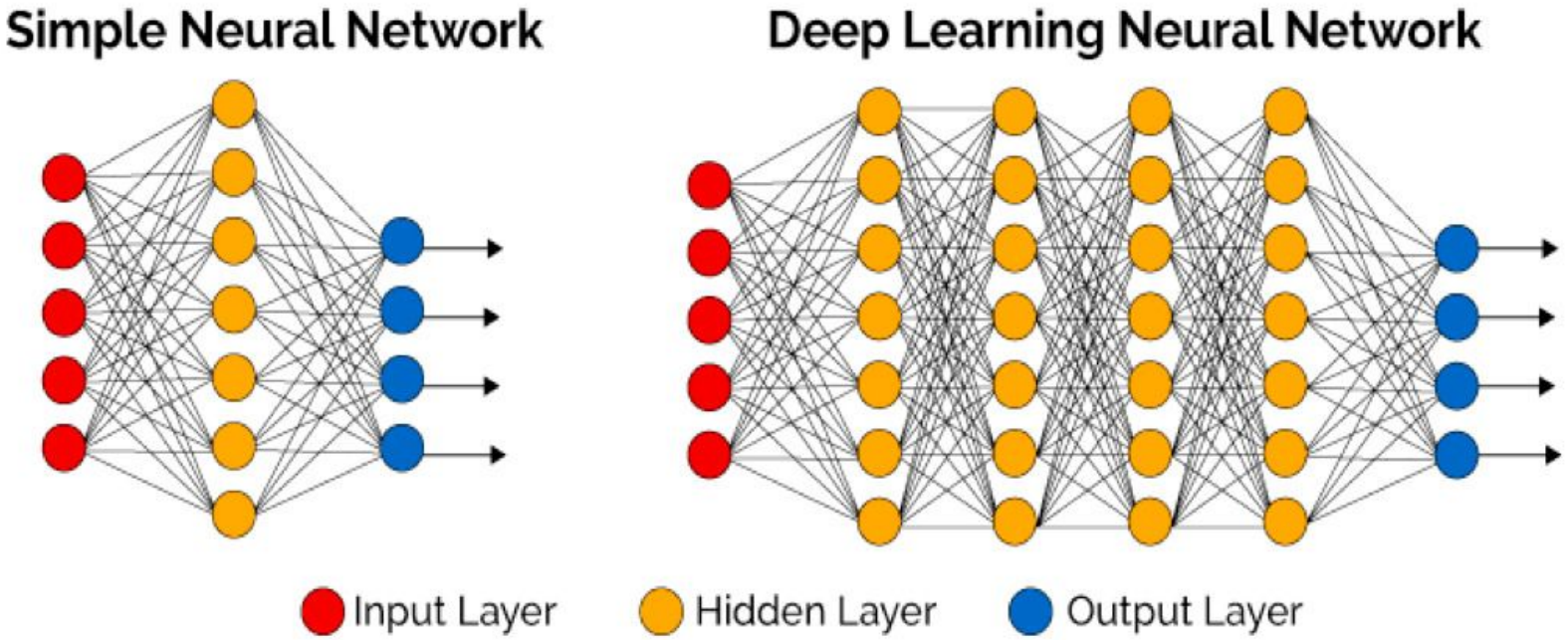
- So far we have just looked at tabular solution methods

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

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

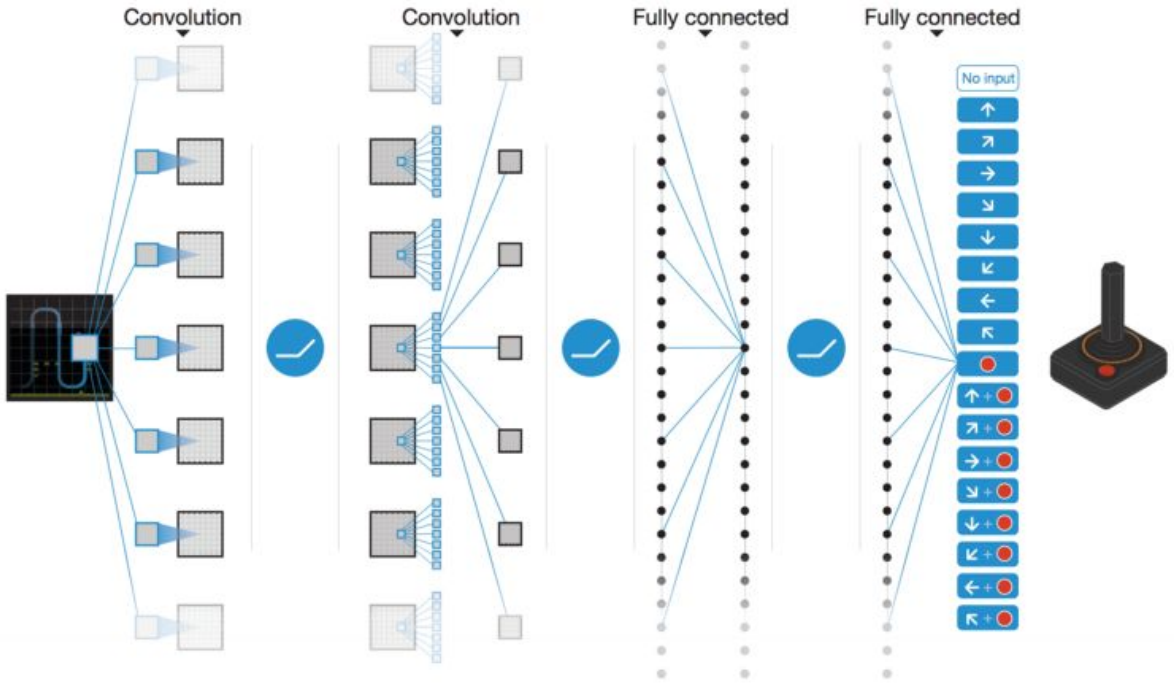
- Use function approximation
  - eg. linear combination of features, neural network, etc.

Approximate Solution Methods: Neural Networks



Approximate Solution Methods: Neural Networks

“DQN”

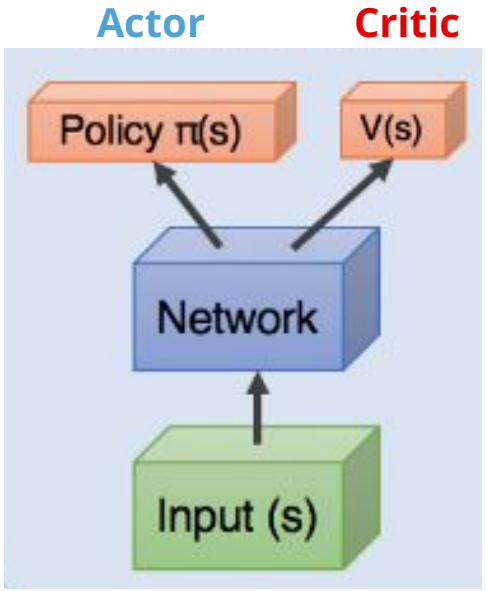


Mnih et al. (2015) *Nature*

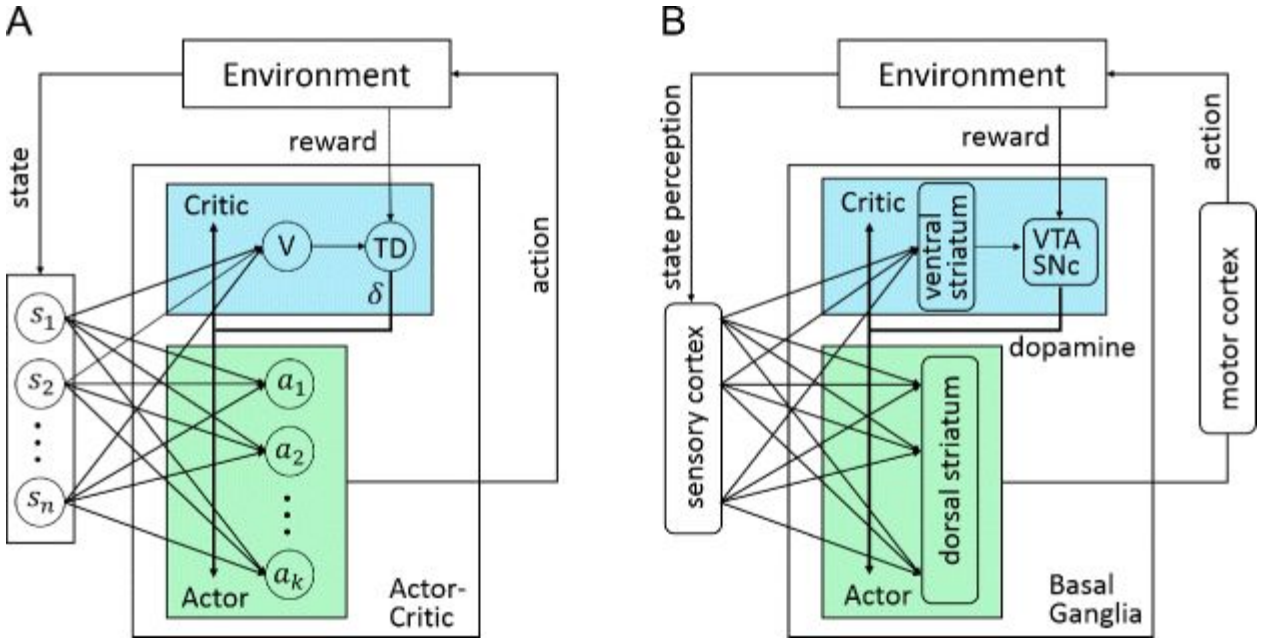
An Example We Won't Get Into

The Actor Critic Network

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



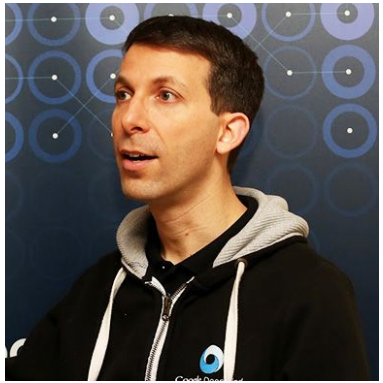
An Example We Won't Get Into



Zhao et al. (2018) *Cog. Computation*

More Materials on RL Fundamentals

Rich Sutton



David Silver

Algorithms for RL - Csaba Szepesvari

Arthur Juliani @ Medium