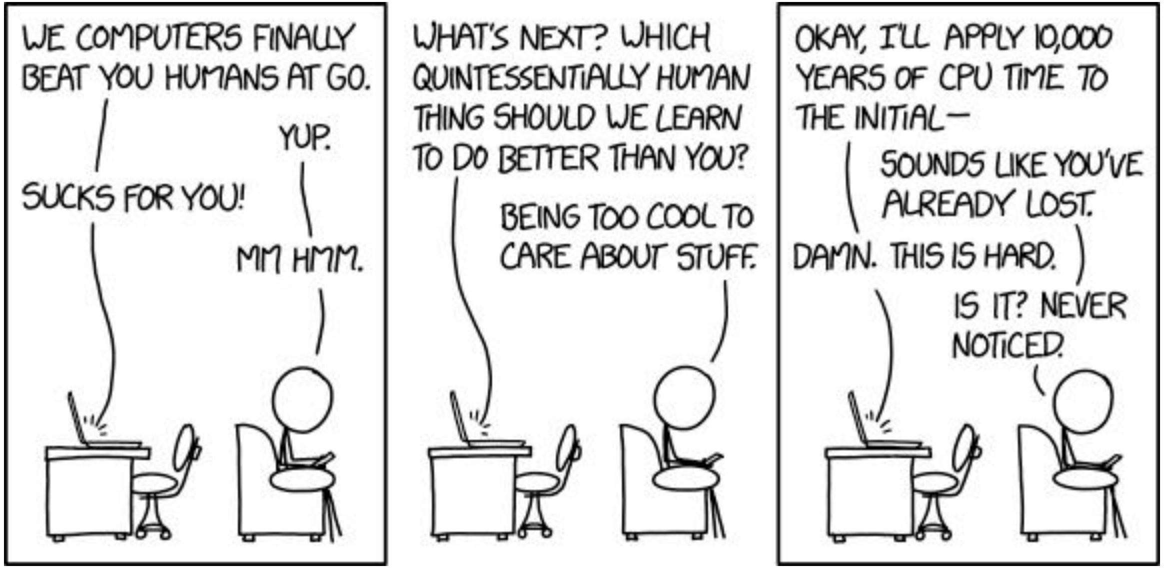

WE COMPUTERS FINALLY BEAT YOU HUMANS AT GO.
YUP.
SUCKS FOR YOU!
MM HMM.
WHAT'S NEXT? WHICH QUINTESSENTIALLY HUMAN THING SHOULD WE LEARN TO DO BETTER THAN YOU?
BEING TOO COOL TO CARE ABOUT STUFF.
OKAY, I'LL APPLY 10,000 YEARS OF CPU TIME TO THE INITIAL—
SOUNDS LIKE YOU'VE ALREADY LOST.
DAMN. THIS IS HARD.
IS IT? NEVER NOTICED.

CIS 522 Lecture 17

Introduction to Reinforcement Learning
3/26/20

Today

What is RL

Markov Decision Processes

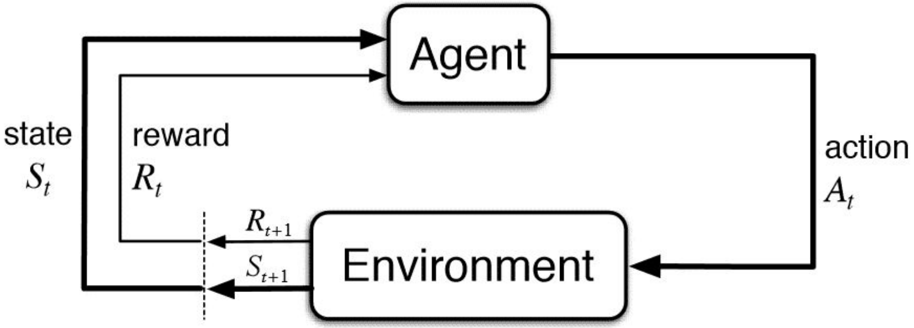
Model free RL

Model based RL

Some Neuroscience inspiration

What is reinforcement learning?

Reinforcement learning (RL)



- Get a (positive) reward when you do something right
- Negative reward if something wrong
- No explicit description of what the right/wrong thing was
- Typically a time delay

See towards data science

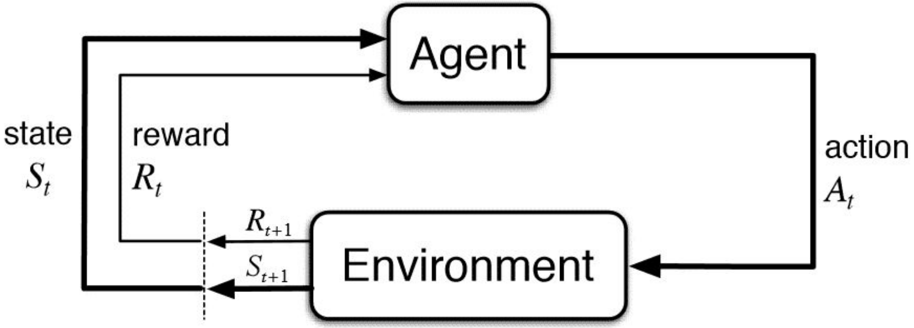

mama
Mummy

What is R, S, A?

Compared with supervised and unsupervised learning

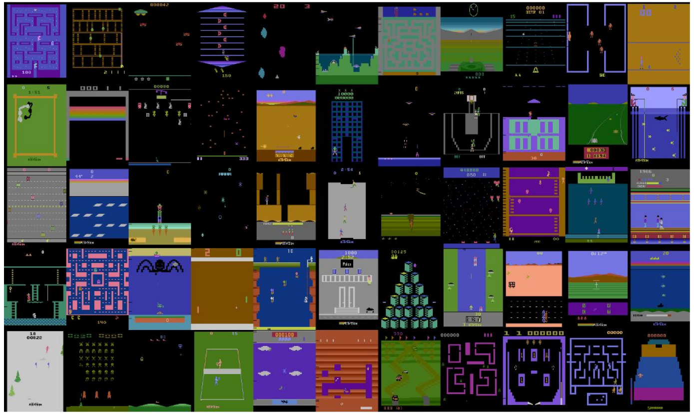
	Supervised	Unsupervised	Reinforcement
Data	(x, labels y)	x	(State s, action a, reward r)
Goal	Learn a mapping x->y	Learn structure of x	Maximize future reward
Example	Classification	Clustering	Game playing -- win the game

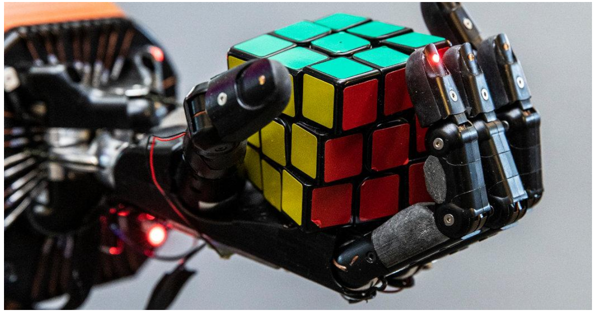
Deep reinforcement learning (next lecture)



Name one way in which deep learning can fit into this RL scheme.

Deep RL successes

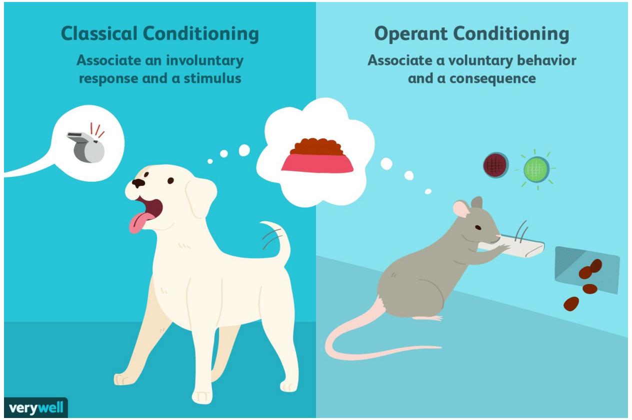




Deep RL hype – is the AI singularity near?

- RL imagines a machine as an agent (something with agency) – temptation to treat RL models as somehow more cognitive
- Touted as a path to artificial general intelligence
- It’s helpful to understand what these algorithms are to see how far we have to go

Origins of reinforcement learning:



- Behaviorism: operant conditioning (cf. classical conditioning)
- Thorndike’s law of effect (1905):

  “if an association is followed by a "satisfying state of affairs" it will be strengthened and if it is followed by an "annoying state of affairs" it will be weakened”

What are the properties of the real word?

Underlying causal relationships

- Over time
- Across objects

The world has sheer unlimited memory

We can not enumerate its possible states

etc.

The Markov decision process (MDP)

- The *environment* is in a *state*, $s_t \in \mathcal{S}$
- An *agent* takes an *action* $a_t \in \mathcal{A}$ according to a *policy* $a_t = \pi(s_t)$ (or $a_t \sim \pi(\cdot|s_t)$)
- For each (state, action), the environment gives the agent *reward* $r_t \in \mathbb{R}$, $r_{t+1} \sim \mathcal{R}(\cdot|s_t, a_t)$
- Then transitions to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$
- Can be finite time, or indefinite
- The goal is to find a policy to maximize expected discounted future reward:

$$\mathcal{G}_t = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right)$$

$0 \leq \gamma \leq 1$ is a *discount factor*.

How are the meaningful aspects of our world not an MDP?

Why discount reward?

- Mathematical convenience – makes sum finite
- Future rewards may be less valuable than immediate rewards (e.g. in finance)
- Future rewards are less certain
- Gamma
    - near 1: far-sighted decision making
    - near 0: myopic
- Matches human/animal behavior

How does the discount matter?

- Small gamma:
    - Relevant future is short
    - Future can be described relatively compactly
    - There are dragons

Approaches to solving RL problems

- Model-free
- Model-based
- Policy optimization
- Some hybrid

Model-free reinforcement learning

- Consider the *value* of states, or the value of state-action pairs

$$V_\pi(s) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right)$$

$$Q_\pi(s,a) = \mathbb{E}_\pi\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right)$$

- Value functions obey the *Bellman equations*:

$$V_\pi(s) = \mathbb{E}_\pi(r_{t+1}|S_t = s) + \gamma \mathbb{E}_\pi\left[V(s_{t+1})|s_t = s\right]$$

$$Q_\pi(s,a) = \mathbb{E}_\pi(r_{t+1}|s_t = s, a_t = a) + \gamma \mathbb{E}_\pi\left[Q(s_{t+1}, a_{t+1})|s_t = s, a_t = a\right]$$

Value functions capture the expected future reward if we are in state s now, and act according to policy pi in the future

Model-free reinforcement learning

- The optimal value functions obey the *Bellman optimality equations*:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right]$$

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{\sigma \in \mathcal{S}} P(\sigma|s,a) \max_{A \in \mathcal{A}} Q^*(\sigma, A)$$

  Optimal value functions capture the expected future reward if we are in state s now, and act optimally in the future

- If we have learnt the optimal value function, *V**, or state-action value function, *Q**, we can back out the optimal policy:

$$\pi(s) = \mathrm{argmax}_{a \in \mathcal{A}} Q^*(s,a)$$

  This is the approach model-free RL takes

Model-free reinforcement learning

- Bellman optimality equations can be used as the basis for iterative procedures to find the optimal value function
- Temporal difference (TD) learning:

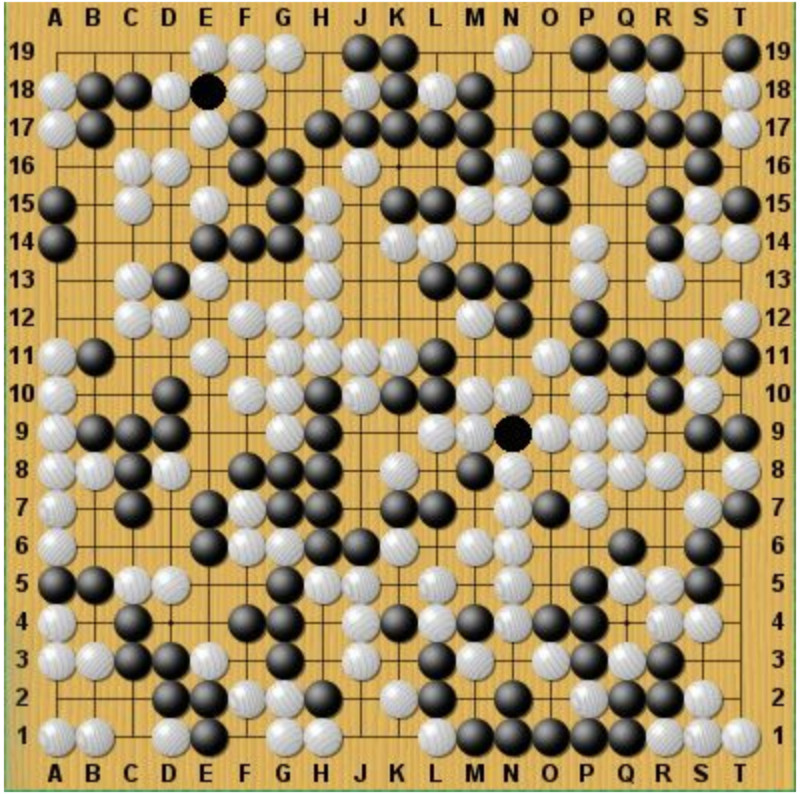
$$V_\pi(s_t) = V_\pi(s_t) + \alpha(r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t))$$

Update value function based on reward prediction error

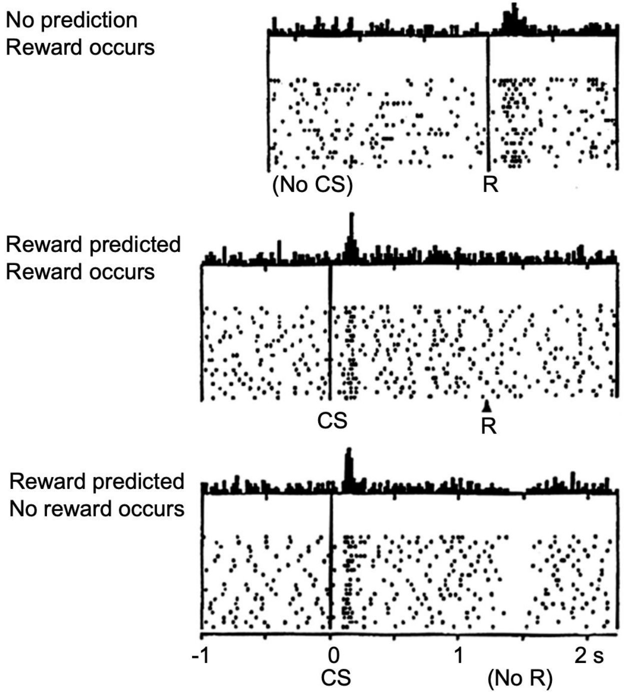
What does "model free" exactly mean?

What is Q? what is V?

What is Q, what is V?



TD learning and the brain



Schultz, Dayan, Montague, 1997

Model-free reinforcement learning (TD)

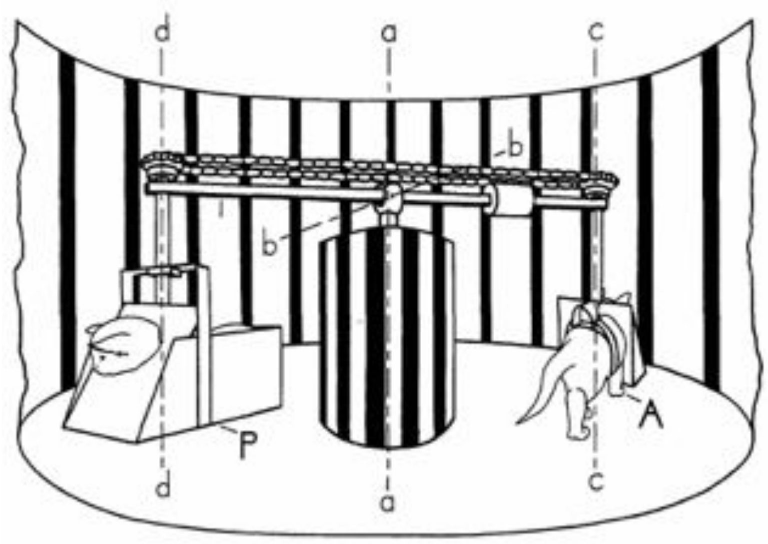
- This can be used as the basis for iterative procedures to find the optimal value function
- State-action-reward-state-action (SARSA) learning:

$$Q(s_t, a_a) = Q(s_t, a_a) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

  Update action value function based on reward prediction error
- These are *on-policy* methods: we are learning about the value of the policy we are implementing

Off-policy vs. on-policy learning

- On-policy learning means the agent can choose actions.
- Off-policy learning means actions are chosen for the agent. This is harder.
- Held & Hein (1963) experiments with kittens.



Model-free reinforcement learning

- Q learning is an *off-policy* version of SARSA:

$$Q(s_t, a_a) = Q(s_t, a_a) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

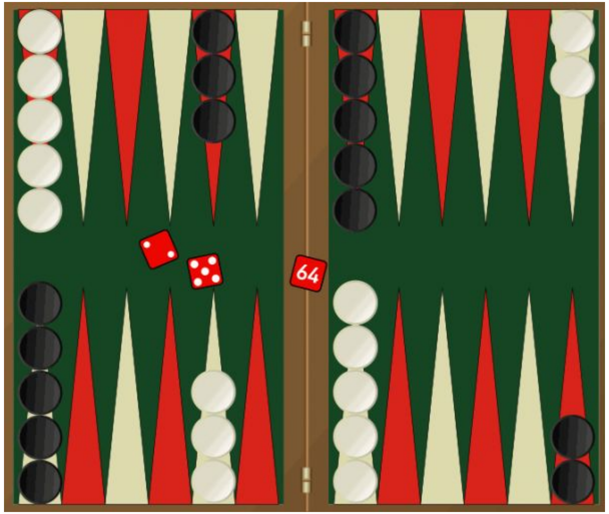
- We learn the Q function of the optimal policy, while taking actions from another policy
- This can facilitate better exploration. But we now have an exploration/exploitation trade-off
- Epsilon-greedy:
  - (1-epsilon)% of steps: act greedily
  - epsilon% of steps: select random action
- Simple but surprisingly hard to beat

Convergence of Q-learning

- Converges to opt Q* if visit each (state, action) pair some large number of times.
- The bounds are pretty terrible (like MCMC).
- In practice, no-one ever runs for that long.
- But also in practice, it works pretty well.

Example: TD Gammon

- Developed 1992
- Uses TD(lambda) algorithm
- Near champion-level backgammon -- novel gameplay style



Model-based RL

Model-based RL

- Alternatively, we can learn (or already know) the MDP dynamics:

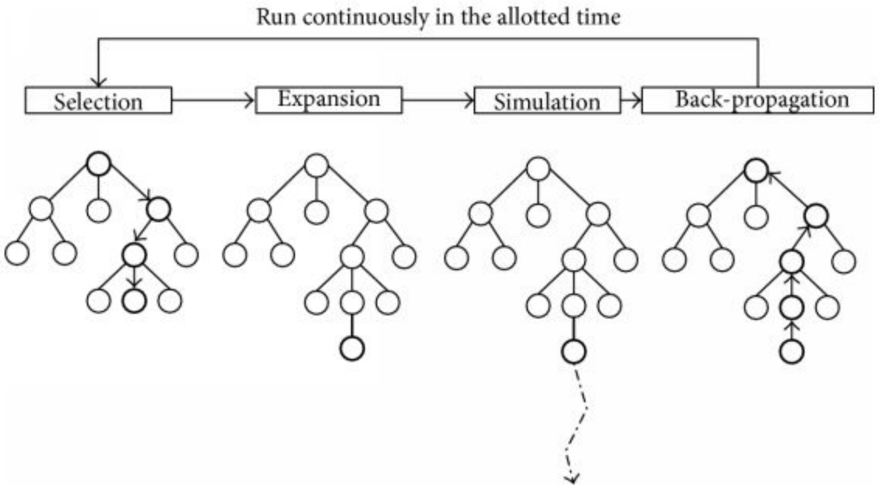
$$s_{t+1} \sim P(\cdot|s_t, a_t) \quad r_{t+1} \sim \mathcal{R}(\cdot|s_t, a_t)$$

- More sample efficient: Once the model and the reward function are known, we can plan the optimal controls without further sampling

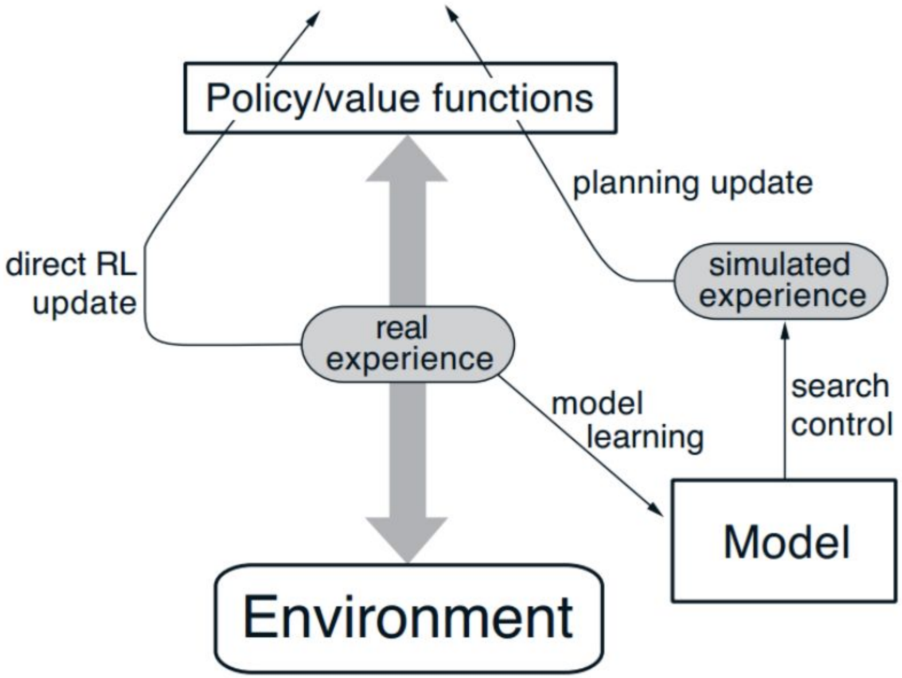
- Analytic gradient optimization (e.g. linear control theory)
- Sample-based planning
- Model-based data generation

Model-based RL

- Sample-based planning
- In discrete action spaces, can use Monte Carlo tree search (MCTS)



Example: Dyna Q



Model-based RL: example Dyna-Q

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \epsilon$-greedy(S, Q)
- (c) Execute action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Policy optimization methods

What does a change in policy do?

Policy gradient

- Directly maximize the total expected reward over the entire trajectory τ

$$\mathbb{E}_\pi(r(\tau))$$

- Gradient descent on the policy's parameters $\boldsymbol{\theta}$:

$$\begin{aligned}
\nabla_\theta \mathbb{E}((\tau)) &= \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau \\
&= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\
&= \int \pi_\theta(\tau) r(\tau) \nabla_\theta \log \pi_\theta(\tau) d\tau
\end{aligned}$$

$$\nabla_\theta \mathbb{E}_\pi(r(\tau)) = \mathbb{E}_\pi(r(\tau) \nabla_\theta \log \pi(\tau))$$

$$\nabla_\theta \mathbb{E}_\pi(r(\tau)) = \mathbb{E}_\pi(r(\tau)\nabla_\theta \log \pi(\tau))$$

Policy gradient

Expand over the trajectory τ

$$\pi_\theta(\tau) = \mathcal{P}(s_0)\Pi_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}, r_{t+1}|s_t, a_t)$$

$$\log \pi_\theta(\tau) = \log \mathcal{P}(s_0) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}, r_{t+1}|s_t, a_t)$$

$$\nabla \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)$$

$$\implies \nabla \mathbb{E}_{\pi_\theta}[r(\tau)] = \mathbb{E}_{\pi_\theta}\left[r(\tau)\left(\sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t)\right)\right]$$

Policy gradient

- That means can estimate the policy gradient by sampling trajectories and computing:

$$\text{update} = \sum_{t=1}^{T} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- We actually don't have to use the whole trajectory reward.
  Only future rewards are affected by the action.

$$\text{update} = \sum_{t=1}^{T} G(t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$G_t = r_{t+1} + \gamma r_{t+2} + \dots$$

Policy gradient

- This is called the REINFORCE algorithm.

$$\text{update} = \sum_{t=1}^{T} G(t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

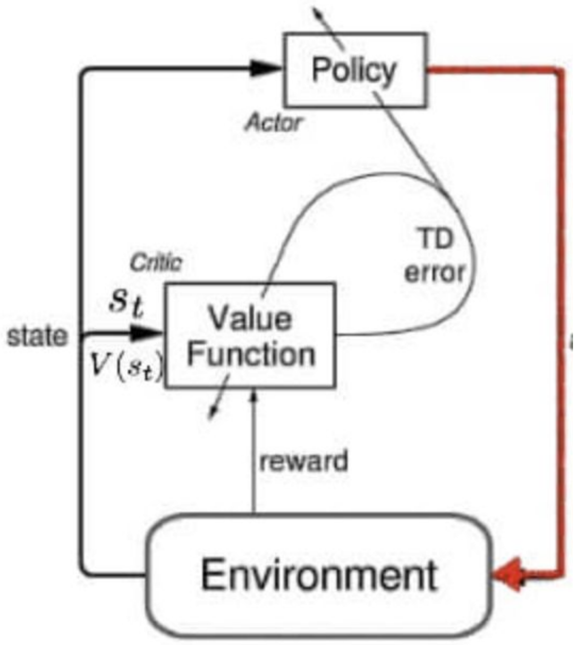
- There is a simple trick that reduces the variance of the estimator:

$$\text{update} = \sum_{t=1}^{T} (G(t) - b) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- The value b is called a baseline, and is just a constant.

Hybrid methods

The actor critique idea



Actor-critic

- We can also learn the baseline!

$$\text{update} = \sum_{t=1}^{T} (G(t) - V(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

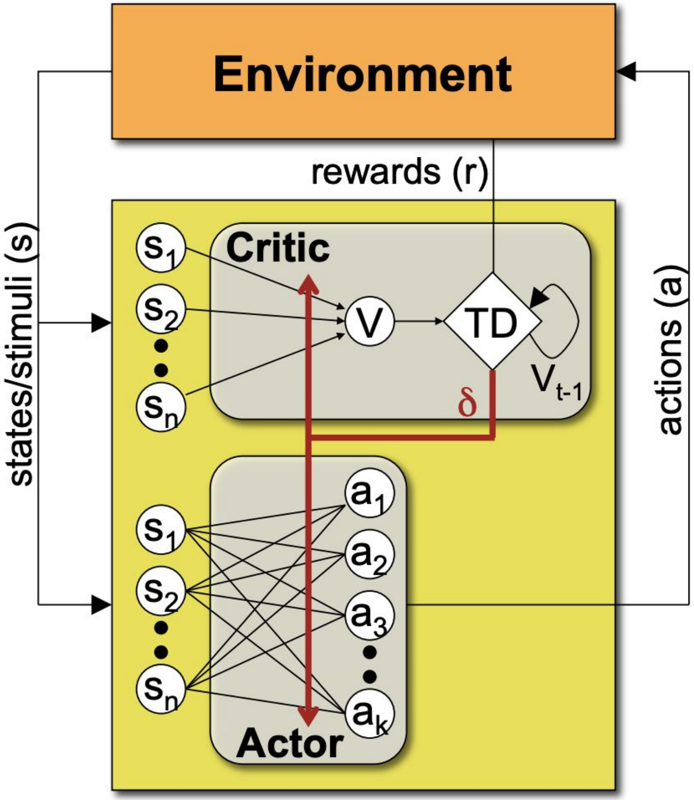
$$\approx \sum_{t=1}^{T} (r_{t+1} + (s_{t+1}) - V(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- This V function (the *value function*) is learned with L2 loss:

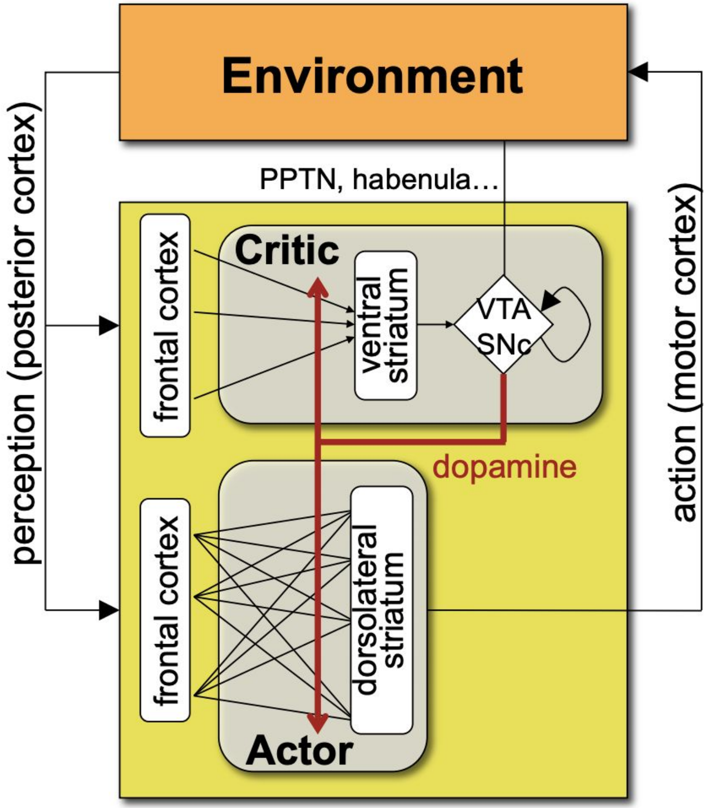
$$(r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t))^2$$

- Two functions to update: policy and value networks

Actor-critic in the brain



Actor-critic in the brain



Summary – model-free RL

- Pros
  - Computationally less complex
  - Needs no representation of environment
- Cons
  - Needs many samples to learn
  - Reward and model dynamics are intertwined in learnt value functions -- slow to adjust to changes in reward or dynamic structure

Sometimes the policy is simple, even if the world is complicated

Summary – model-based RL

- Pros
  - Plan and explore from simulated experience
  - Quickly adapt to changes in environment (e.g. cheese is placed somewhere else in a maze)
- Cons
  - Only as good as the model learnt
  - Planning can be computationally intensive

Sometimes the policy is complicated, even if the world is simple

RL vs Causal Inference

Ultimately RL is about causal inference:

- Which things can I do that will make things better
- That is why randomness matters so much

Ongoing work in lab (Ben Lansdell)

Deep reinforcement learning

Penn Engineering

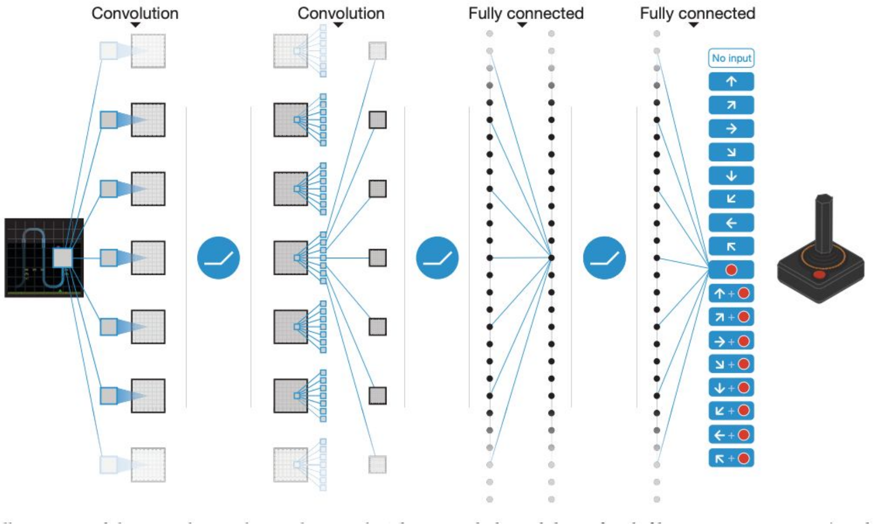
Tabular vs function approximation

- In many applications, state and action spaces are too high dimensional to estimate value of every state-action pair (e.g. states are pixel arrays)
- Approximate (e.g.) value function V_theta(s)
  - Linear functions
  - Neural networks
  - Decision trees
  - etc.

Deep Q learning (DQN)

- Use deep net to estimate Q-values
- Input: the state
- Output: Q-values for possible actions.
- Learning step: gradient descent with the loss:

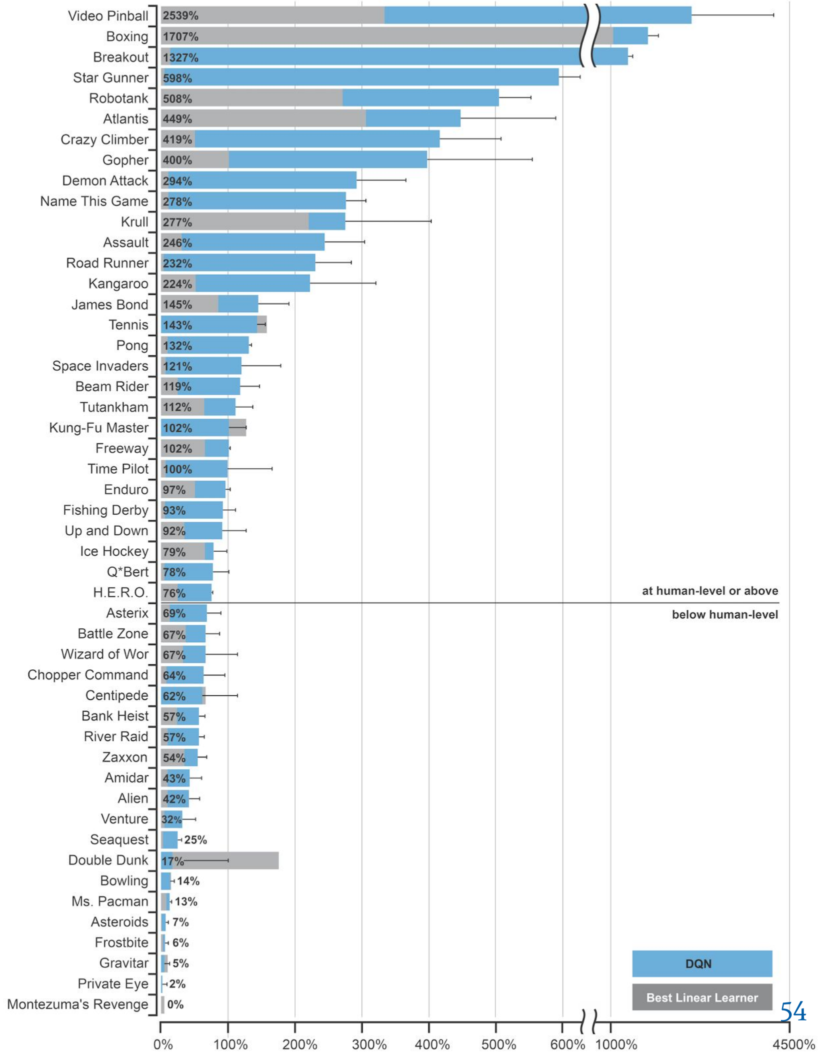
$$\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)^2$$



- Policy: choose action to maximize the Q-value

Deep Q learning (DQN)

- Distribution of actions changes over time.
- Can forget stuff it learned early on (e.g. basic mistakes).
- Experience replay: store past experiences = (state, action, reward, next state)
- Learn from random samples from the past



What we learned

What is RL

Markov Decision Processes

Model free RL

Model based RL

Some Neuroscience inspiration

How could intro to RL have been better?