Reinforcement Learning workshop

Florian Goebels Sep 2019

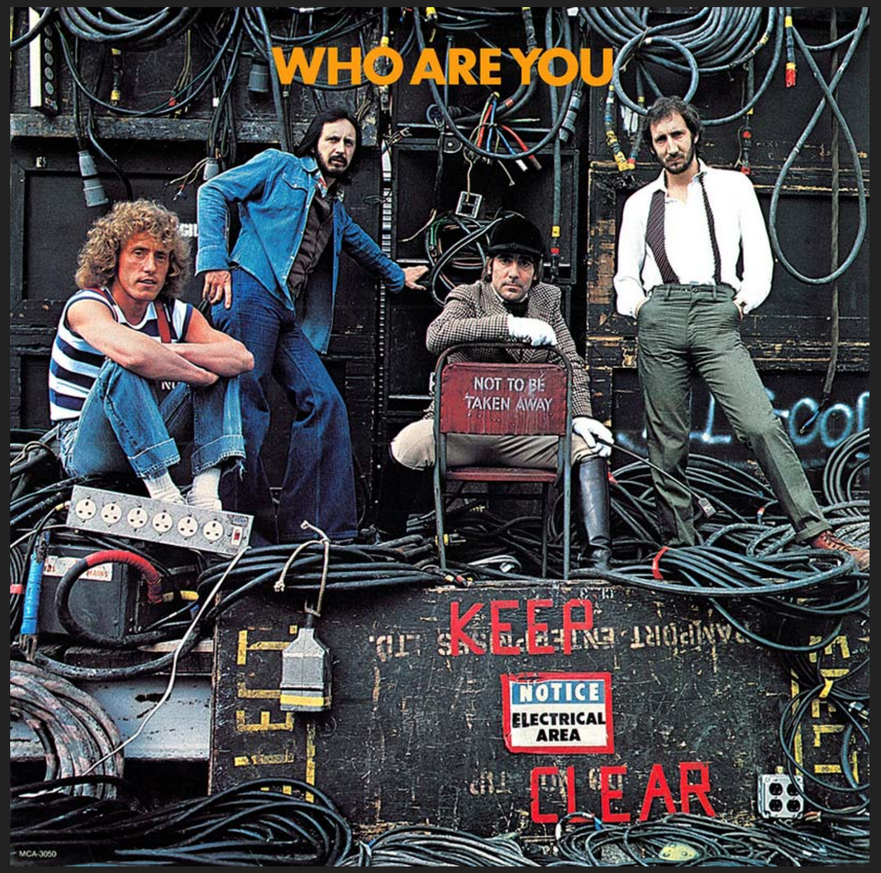
September 19th 2019

aggregate intellect

A little bit about me

- CS PhD University of Munich
- Post-doc at University of Toronto
- 10 Years of experience in ML, 6 of which in bioinformatics
- Experience in ML in life science, sales, and finance

A little bit about you



- Come up with one goal for this course
- Come up with one rule for this workshop

A little bit about the course

- During the test your team can score points
- We have some nice goody for the top team
- How to score:
  - Complete assignment +3 points per assignment
  - Score first, second and third in challenges gives you 5, 3 and 2 points respectively

Course project

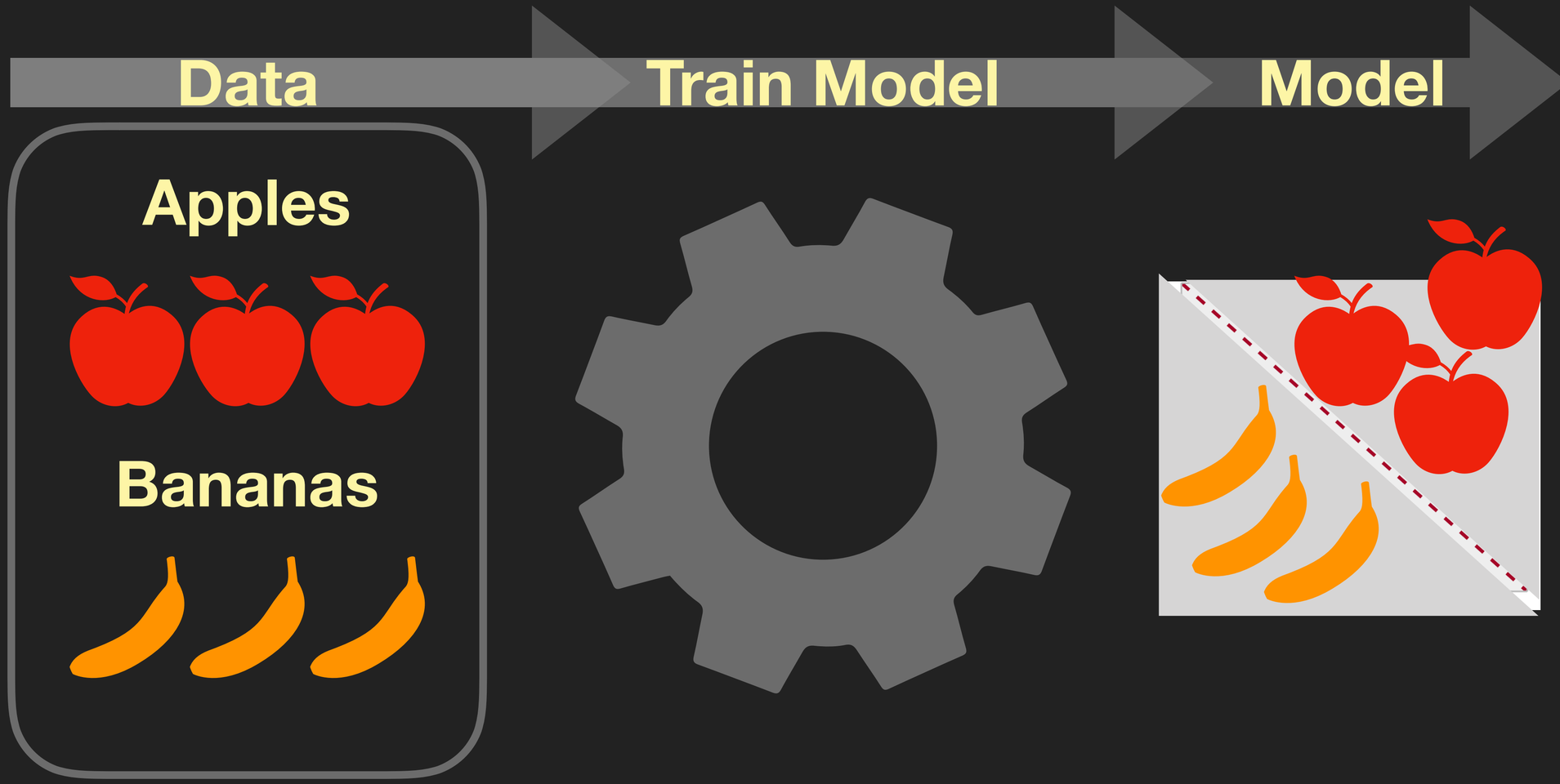
Agenda

- Day 1: RL Basics (theory)
- Day 2: Deep Q-learning (coding)
- Day 3: Policy Gradient methods (coding)

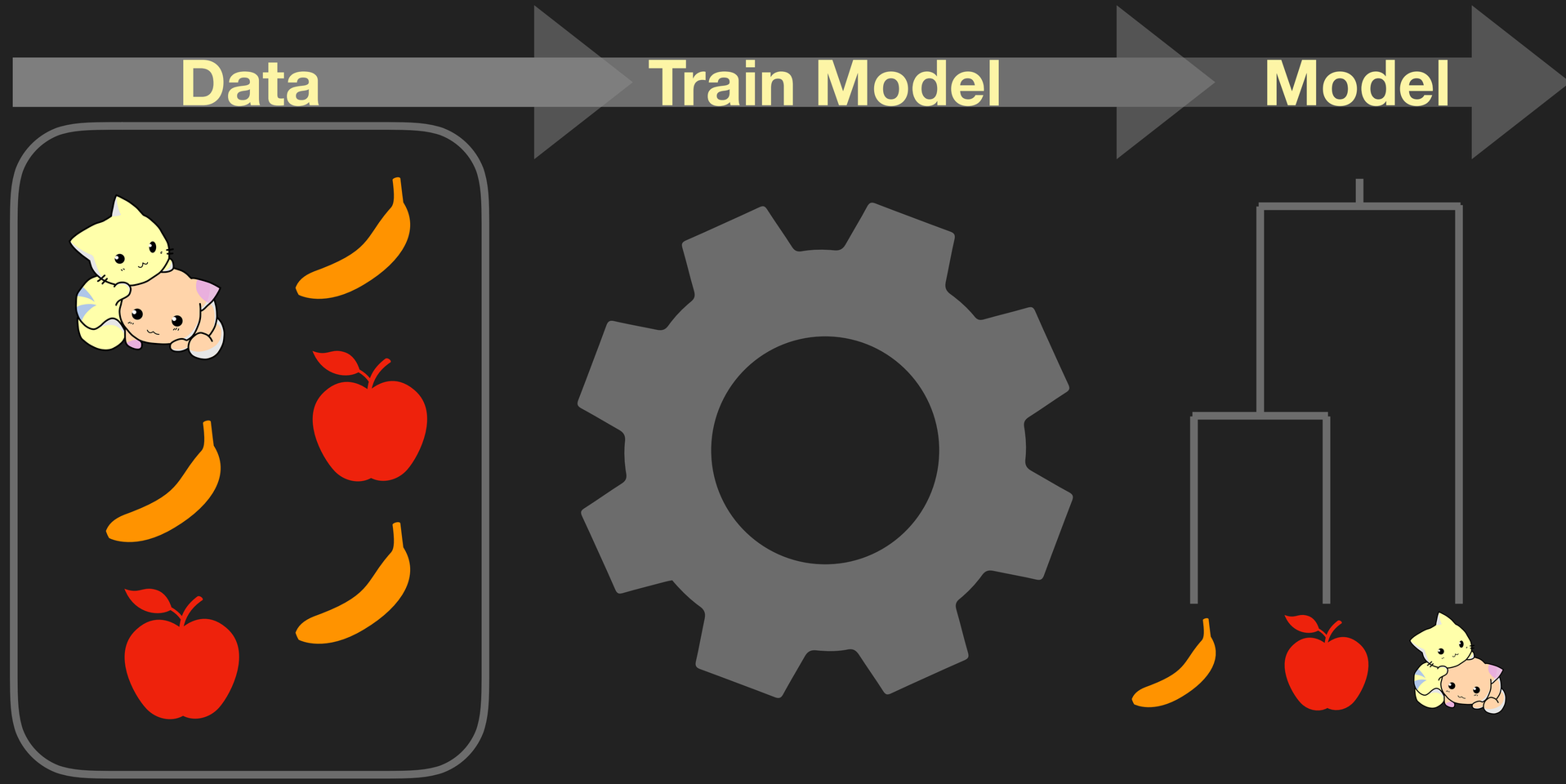
Today’s Agenda

- Day 1: Basics
  - Greedy method
  - Dynamic Programming
  - Monto Carlo based optimization
  - Q-learning/SARSA

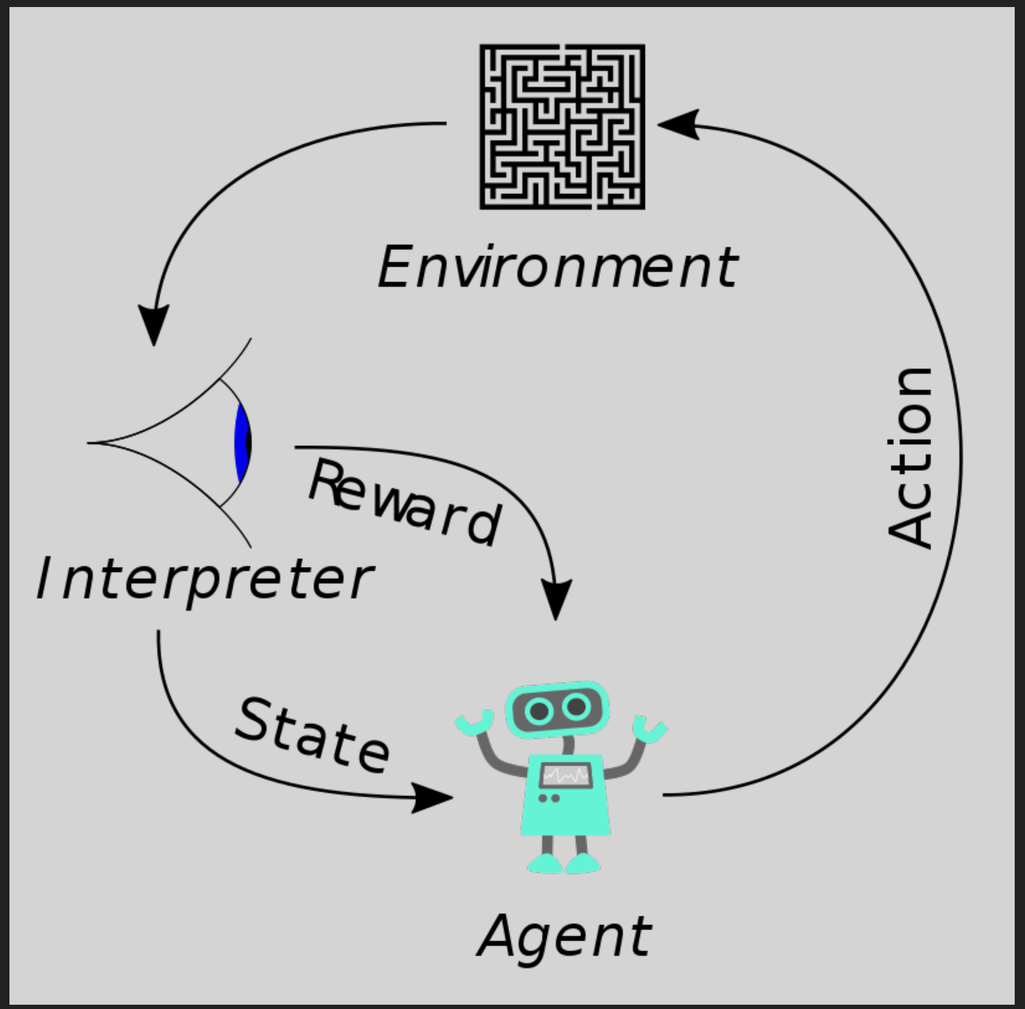
Supervised learning



Unsupervised learning



Reinforcement learning



Central RL concepts

- Policy:
  Defines how our AI choose his action for a given state.

- Reward Signal:
  The reward for a given action.

- Value function:
  Expected future rewards of a state (how good is a given state)

- Environment models:
  Simulates future stares, allows for planning

Let’s play a game

2-armed Bandit

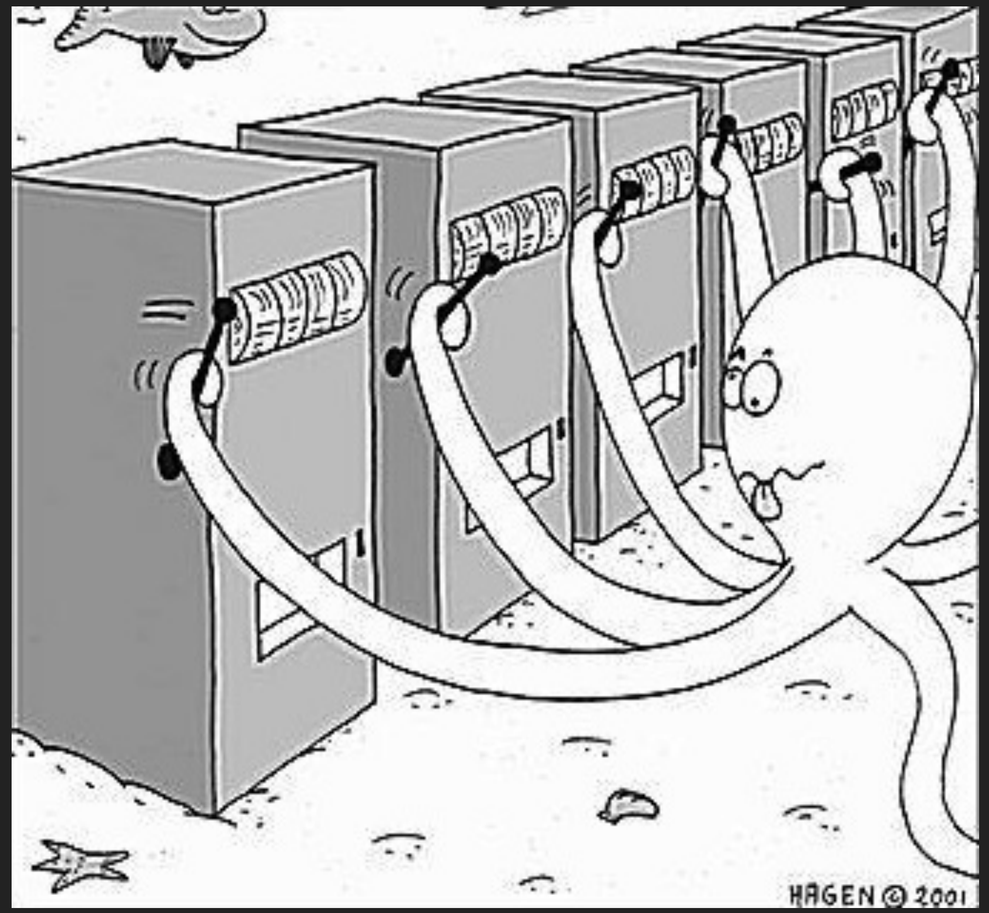
- Choose left or right rewards are unknown
- Pull lever
- Collect money
- For your self or in team write down next 10 pulls
- Vote in the slack how many times you choose left

2-armed Bandit

	Prob	Reward
Left	1	1
Right	0.1	100

k-armed Bandit

- Choose action A_i
- Collect reward R_i



Action

- Response, control
- They can be discrete or continuous
- Set of actions are known

Reward

- Gain, payoff, cost
- Immediate after each time-step
- RL maximies reward

How to come up with a strategy (policy)

- Pull every lever *n* times select lever with best reward.
- Randomly Pull lever *n* times and then select lever with best average reward.
- Randomly Pull lever *n and then select lever with best average reward and every n pulls select random lever.*

How to come up with a strategy (policy)

- at time-step ***t*** we perform action A_t and receive reward R_t
- for any possible action ***a*** we donate ***q*(a)*** the expected reward

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

- We are looking for an estimation of expected reward for an action $Q_t(a)$
- One possible estimation is the mean reward:

$$Q_{n+1} = \frac{1}{n} \sum_{i=1}^{n} R_i$$

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n} \sum_{i=1}^{n} R_i \\
&= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\
&= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\
&= \frac{1}{n} \left(R_n + n Q_n - Q_n \right) \\
&= Q_n + \frac{1}{n} \left[R_n - Q_n \right]
\end{aligned}
$$

How to come up with a strategy (policy)

step size

New estimate Old estimate Reward - old estimate

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n]$$

Epsilon-Greedy

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Goal?

- Any game is a sequence of actions and rewards know as ***trace***
- Total future reward of a trace of size ***T***

$$G_t = R_{t+1} + R_{t+2} + R_{t+2} + \ldots + R_T$$

- We discount future rewards

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

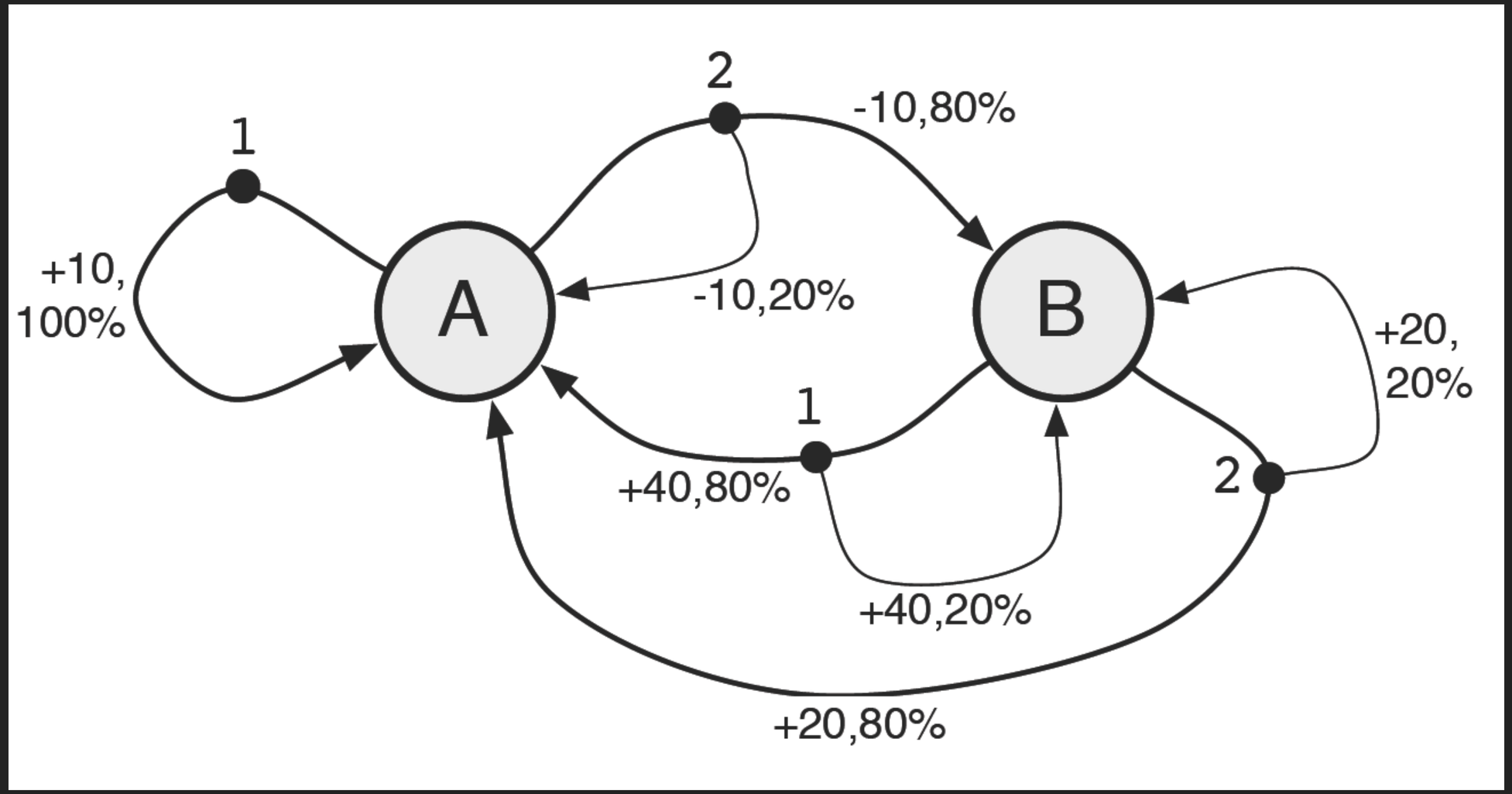
Goal?

- G_t is recursive:

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

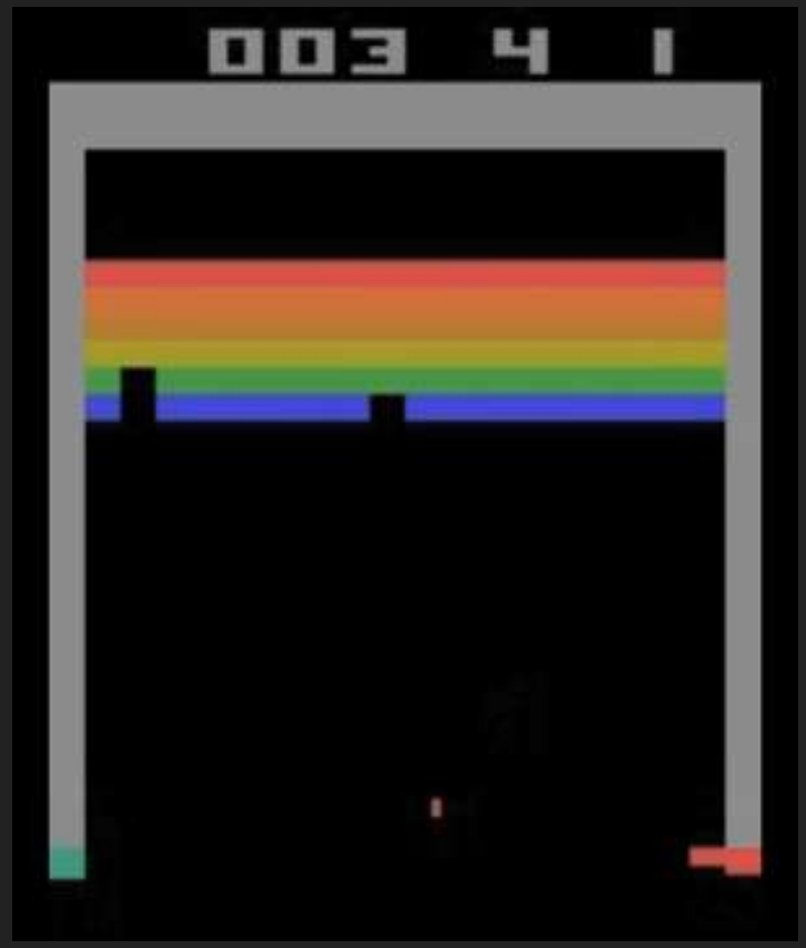
What about state?

Simple game with state



State

- How the environment communicates with the agent: current and next state.
- We aim to score and evaluate states
- The RL aims to find favourable states and avoid bad states



Markov Decision Process

- Sequence of actions ***A***, states, ***S*** and rewards ***R***

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

- Trade-off immediate and delayed rewards
- Two approaches:
  - Estimate value of state action pair ***q(s,a)***
  - Estimate value of state **v(s)** given action selection

Markov Property

- Each possible next S_t and R_t value is only dependent on the immediately preceding state and action, S_{t-1} and A_{t-1}

- Best interpreted as restriction on state:

  The state must include information about all aspects of the past agent–environment interaction that make a difference for the future

$$p(s', r \mid s, a) = Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\}$$

Value function

- Value function of state **s** under random policy **π**

$$q_\pi(s,a) = \mathbf{E}_\pi\left[G_t \,|\, S_t = s, A_t = a\right] = \mathbf{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a\right]$$

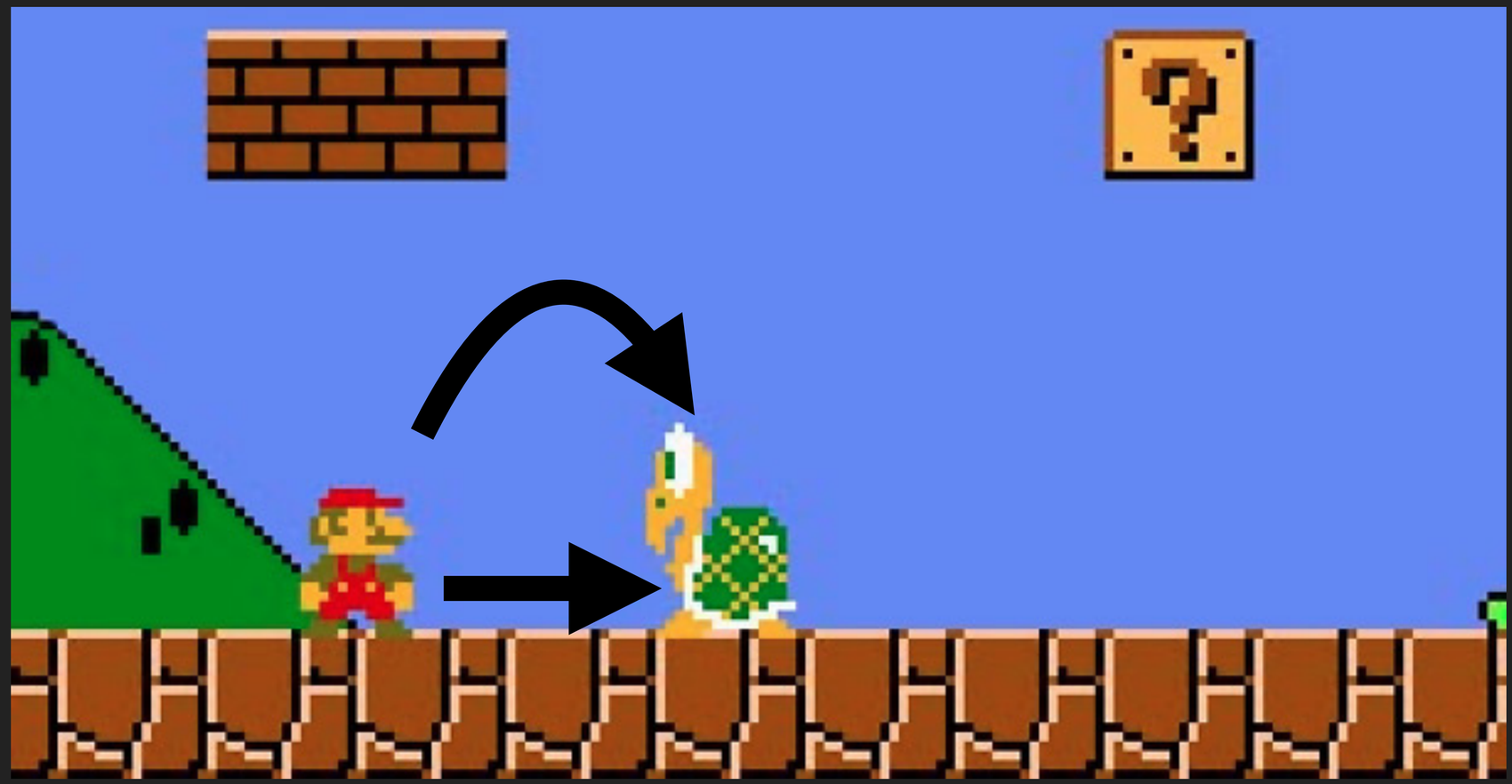
- Value function of action ***a*** in state **s** under random policy **π**

$$v_\pi(s) = \mathbf{E}_\pi\left[G_t \,|\, S_t = s\right] = \mathbf{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s\right]$$

Bellman equation

$$
\begin{aligned}
v_\pi(s) &= \mathbf{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbf{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma \mathbf{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\
&= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right], \textit{ for all } s \in S,
\end{aligned}
$$

The current state

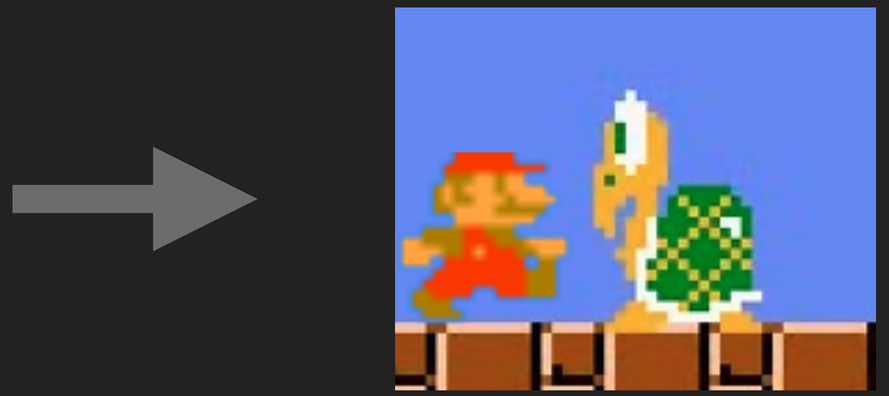


The current state

$v_\pi(s)$

State s

Action State Reward



+10

So we’re done now?

- Agent almost never learns optimal policy
- Even if we have perfect model of environment (i.e. chess), the search space if too large
- We have to approximate the optimal policy:
  - Do better in frequent states
  - Do worse in rare states

Dynamic programming

- Use value function to perform a structured search of good policies
- Iterative approximations $v_1, v_2, v_3, v_4, \ldots$ of $v_\pi(s)$ by using Bellman equation as update rule
- Replace $v(s)$ with new value calculated bu the old values of $v(s')$. This is called expected update.
- Terminates ones value functions minimal change after iteration

DP demo

- https://cs.stanford.edu/people/karpathy/reinforcejs/

Dynamic programming

Value Iteration, for estimating $\pi \approx \pi_*$

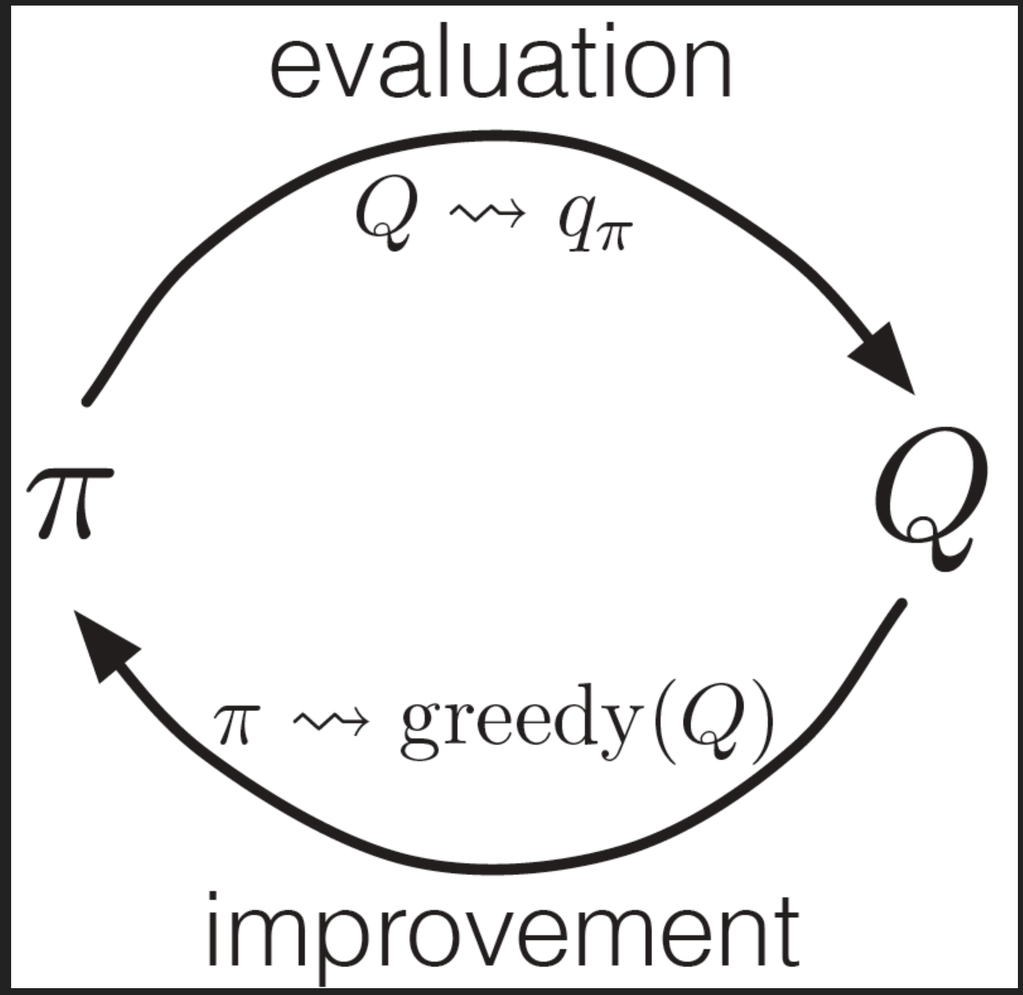
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Generalized Policy iteration

- Almost all RL algorithm are GPI
- Maintain both an approximate value function and an approximate policy
- Iteratively improve policy with respect to value function, and value function always drives to the value function of the current policy
- Overall process converges to to optimal policy and optimal value function



Monte Carlo

- DP search the entire state and needs store full ***a x s*** space table (backgammon 10^{20} states)
- DP requires distribution of the next events
- Monte Carlo based methods rely only on experience
- No prior knowledge of the environment is required
- Averages sample returns (remember k-armed bandits)

On-policy first-visit

GPI

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

On/Off-Policy

- Problem all method seek optimal behaviour, but need to behave non-optimally to explore all actions
- **On policy** evaluate and improve policy that is used to make decision
- **Off policy** evaluate and improve policy that is different from decision making

Off policy

- Deploy two policies:
  - behavioural policy ***b***: Exploration
  - Target policy ***π***: Optimized
- Assumption of coverage:
  - ***π(a|s)>0*** then ***b(a|s)>0***

Importance sampling

- Technique for estimating expected values of one distribution (***π***) given samples from another distribution (***b***)
- $\mathbf{E}[G_t \mid S_t = s] = v_b(s)$ is based on ***b*** can not averaged to estimate v_π
- Importance sampling transforms returns so that it has right expected value:

$$\mathbf{E}[P_{t:T-1} G_t \mid S_t = s] = v_\pi(s)$$

Transform weight

- Probability of a trajectory:

$$Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\}$$

$$= \prod_{k=t}^{T-1} \pi(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)$$

- Rescaling is $P_\pi(t)/P_b(t)$

$$p_{t:T-1} = \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$$

Weight importance scaling

- Rescale returns:

$$V(s) = \frac{\sum_{t \in (T)(s)} P_{t:T(t)} G_T}{\sum_{t \in (T)(s)} P_{t:T(t)}}$$

- Update rule:

$$V_{n+1} = V_n + \frac{W_n}{C_n} \left[G_n - V_n \right]$$

and

$$C_{n+1} = C_n + W_{n+1}$$

Off-policy

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

Temporal difference

- Combines ideas from DP and MC methods
  - Like MC: learns directly from raw experiences
  - Like DP: iteratively updates estimates

TD *V(S)* estimate

MC

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t) \right]$$

DP

$$v_\pi(s) = \mathbf{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s \right]$$

TD

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

TD *V(S)* estimate

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

?

Better estimate of *V(S)*

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

TD ERROR

Current estimate of *V(S)*

TD advantages

- Does not require model of environment (unlike DP)
- MC needs to wait until episode finish, TP can online update
  - MC it’s hard to estimate value of action-state pair
- Both TD and MC converge to optimal policy?

TD demo

- https://cs.stanford.edu/people/karpathy/reinforcejs/

Q-learning

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

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

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

SARSA

- it is the only learning update that uses exactly these five things from the trajectory:

$$\ldots, S_1, A_1, R_2, S_2, A_2, \ldots$$

- It is equivalent to Q-learning but applied to state-action pairs
- Unlike q-learning it is on policy

SARSA

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

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

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

Next time on AISC

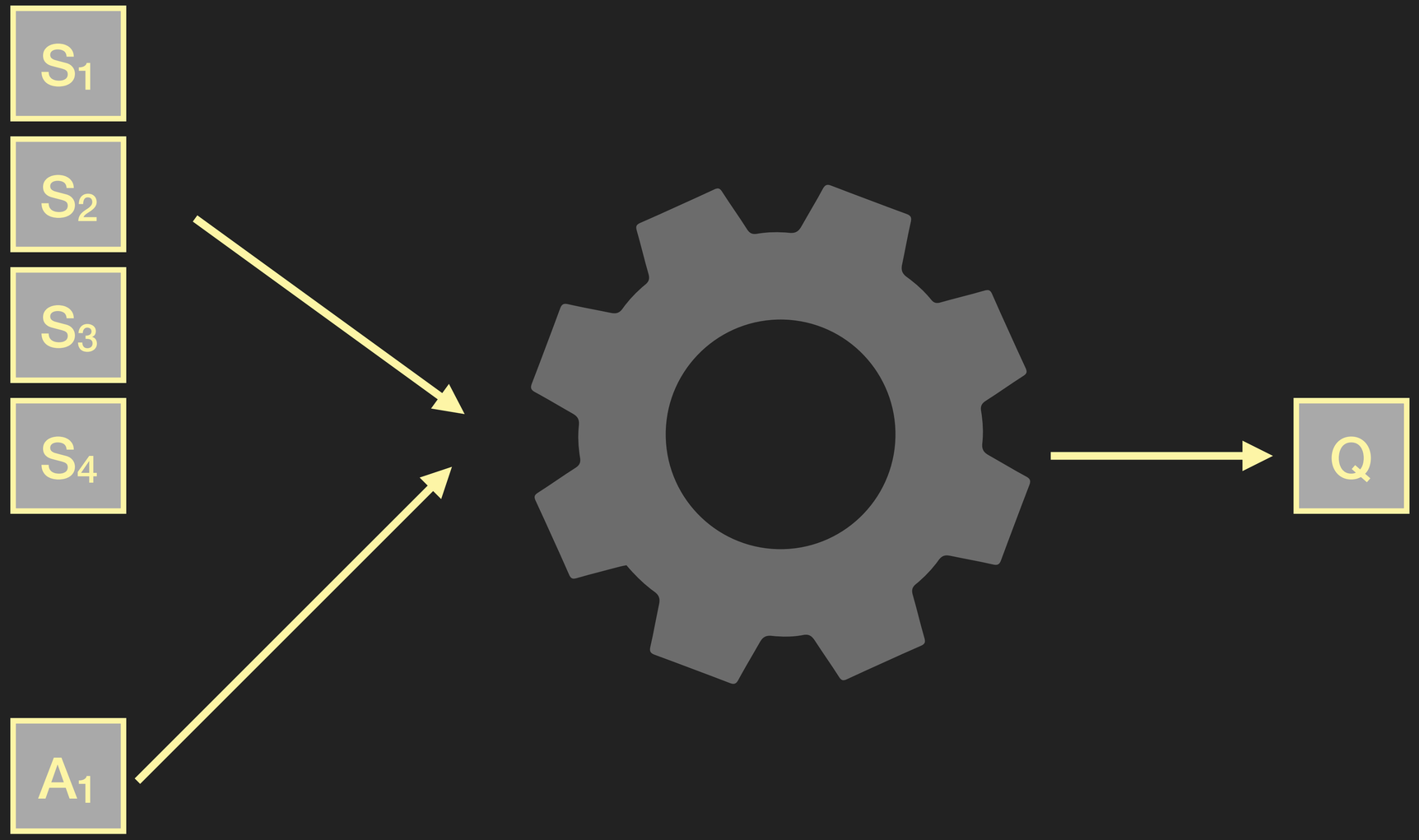
Deep Q-Learning

Deep Q-Learning

- In Q learning we iteratively update ***Q(A,S)*** via policy iteration
- Neural networks are general function approximators NN(S) -> V(A)
- Cost function:

$$\left(\mathbf{NN}(S_t, a) - r + \gamma \max_a \mathbf{NN}(S_{t+1}, a) \right)^2$$

Deep Q-Learning in a nutshell



Appendix

Optimal Bellman equation

- Each possible next S_t and R_t value is only dependent on the immediately preceding state and action, S_{t-1} and A_{t-1}

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$

$$= \max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right].$$

$$\mathbb{E}_\pi[r_{t+1}|s_t = s] = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \mathcal{R}^a_{ss'}$$

$$\mathbb{E}_\pi\left[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_t = s\right] = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_{t+1} = s'\right]$$

By distributing the expectation between these two parts, we can then manipulate our equation into the form:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_{t+1} = s'\right]\right]$$

Now, note that equation (1) is in the same form as the end of this equation. We can therefore substitute it in, giving us

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] \tag{3}$$

The Bellman equation for the action value function can be derived in a similar way. The specific steps are included at the end of this post for those interested. The end result is as follows:

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a')\right] \tag{4}$$

https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/

https://stats.stackexchange.com/questions/243384/deriving-bellmans-equation-in-reinforcement-learning

License