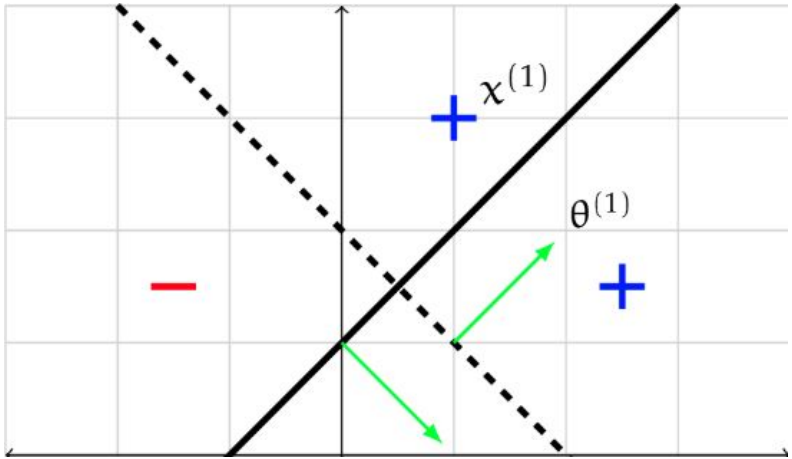
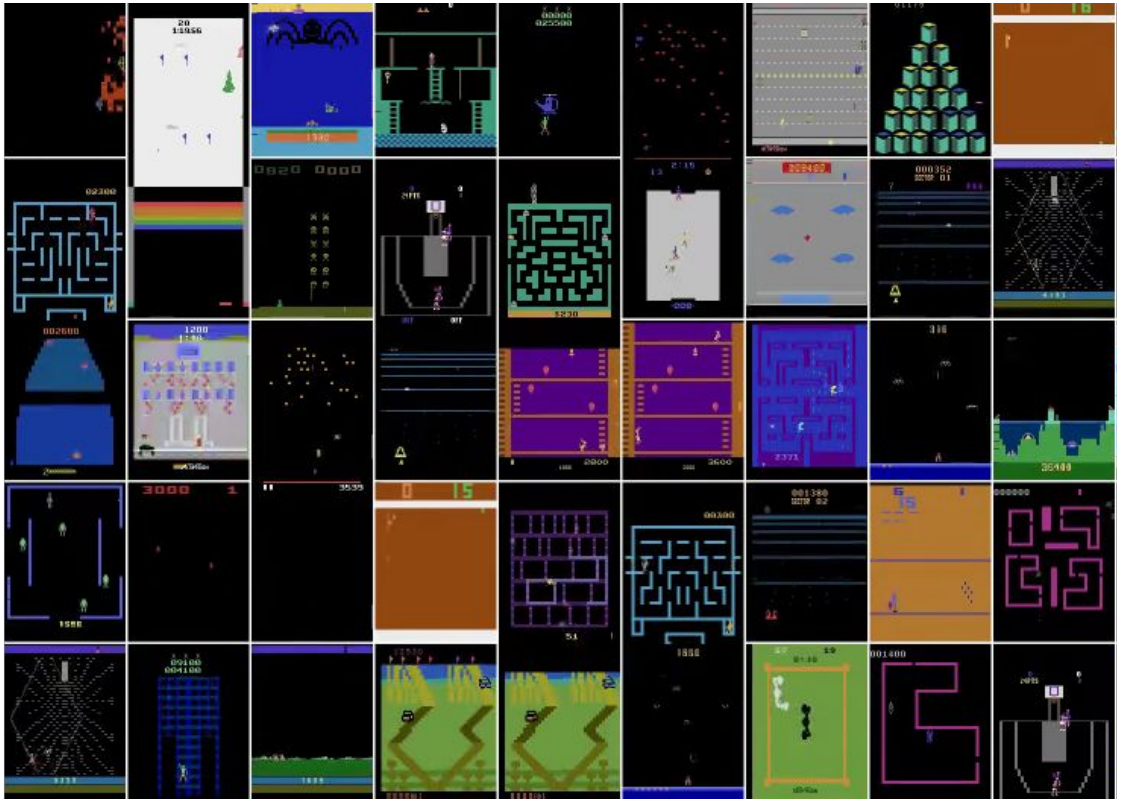
Introduction to Machine Learning



Reinforcement Learning




ALPHAGO
00:00:54
AlphaGo
Lee Sedol


Workers: 3/3
Workers: 25/18
Workers: 3/3
14:32
Catalyst LE
AlphaStar
LiquidTLO
177 /200
147 /172
945 +2015
335 +1595
758 +873
442 +1030
64
61
113
86
940
1377
SUPPLY
MINERALS
GAS
WORKERS
ARMY
APM
PRODUCTION

Reinforcement Learning

Markov Decision Process

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, s_0)$$

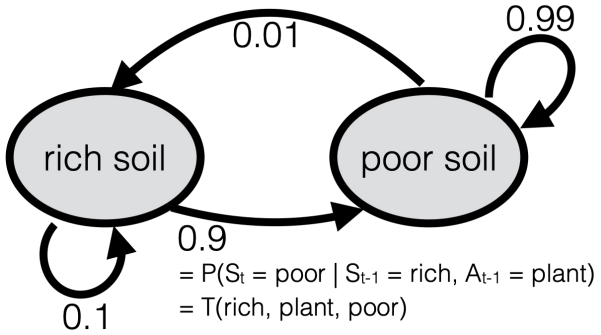
- $\mathcal{S}$ = set of possible states
- $\mathcal{A}$ = set of possible actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: transition model
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: reward function
- γ = discount factor

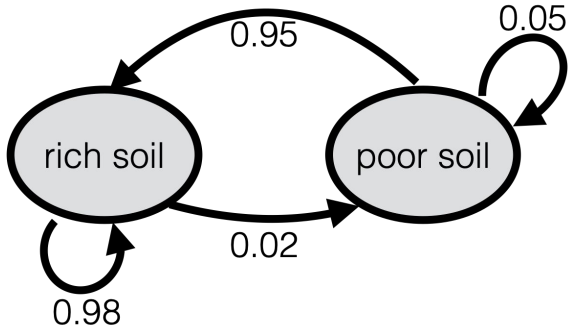
Reinforcement Learning

Markov Decision Process

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, s_0)$$

- $\mathcal{S}$ = set of possible states
- $\mathcal{A}$ = set of possible actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: transition model
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: reward function
- γ = discount factor



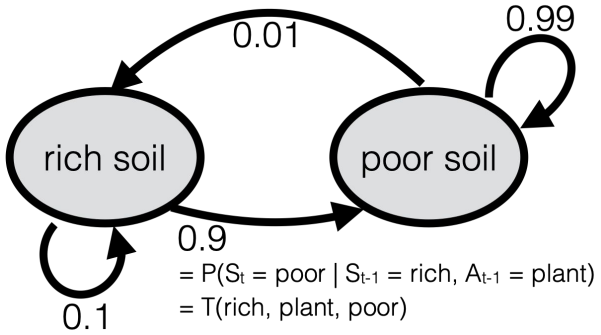


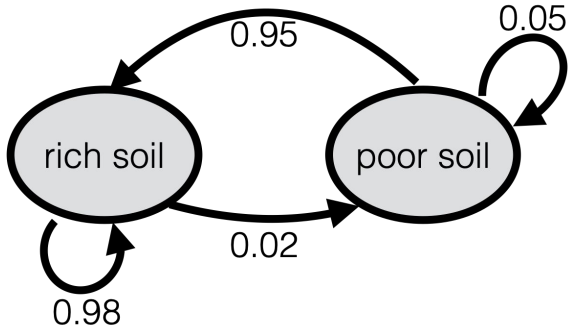
Reinforcement Learning

Markov Decision Process

$$(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, s_0)$$

- $\mathcal{S}$ = set of possible states
- $\mathcal{A}$ = set of possible actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$: transition model
- $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: reward function
- γ = discount factor





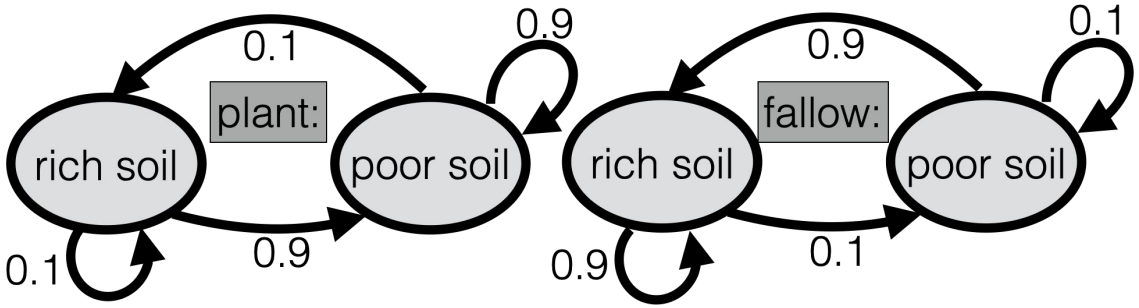
Goal in RL: find a "policy" $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes reward

Review: Value of a policy

- Given an MDP and a policy $\pi : \mathcal{S} \to \mathcal{A}$ we can find the value of a policy by solving a system of linear equations.

Review: Value of a policy

- Given an MDP and a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ we can find the value of a policy by solving a system of linear equations.



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

$$V_\pi^0(s) = 0; V_\pi^h(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') \cdot V_\pi^{h-1}(s')$$

value of the policy with *h* steps left

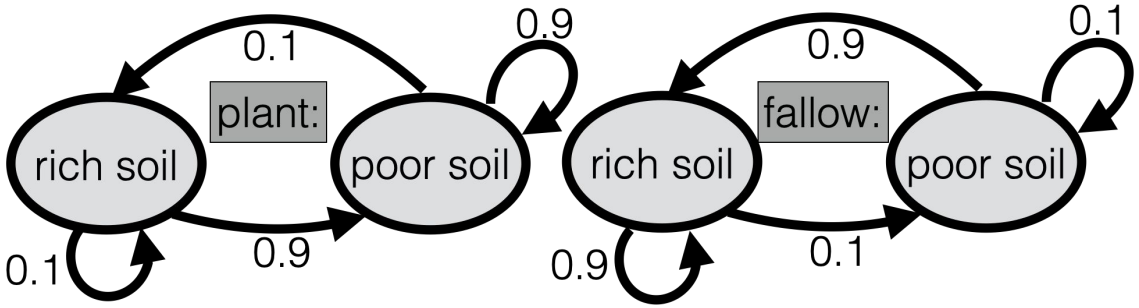
value of the policy on this time step

(expected) value of the policy across all future time steps

- *h*: horizon (e.g. how many growing seasons left)
- $V_\pi^h(s)$: value (expected reward) with policy π starting at *s*

Review: Value of a policy

- Given an MDP and a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ we can find the value of a policy by solving a system of linear equations.



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

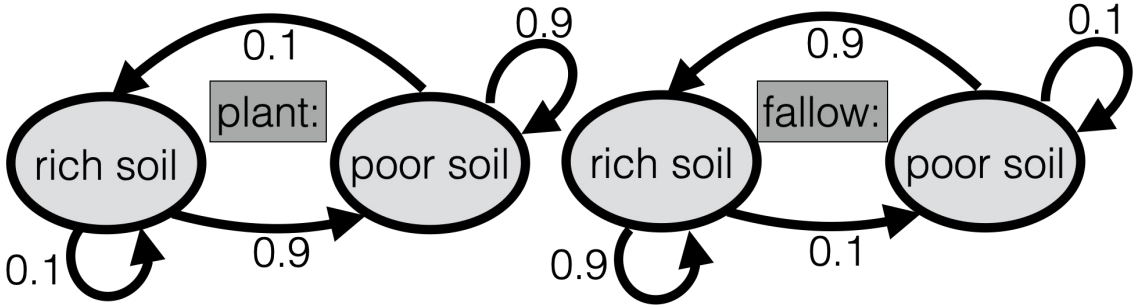
$$V_\pi^0(s) = 0; V_\pi^h(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') \cdot V_\pi^{h-1}(s')$$

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi(s')$$

- *h*: horizon (e.g. how many growing seasons left)
- $V_\pi^h(s)$: value (expected reward) with policy π starting at *s*

Can use to evaluate which policy is better.
How to compute best policy?

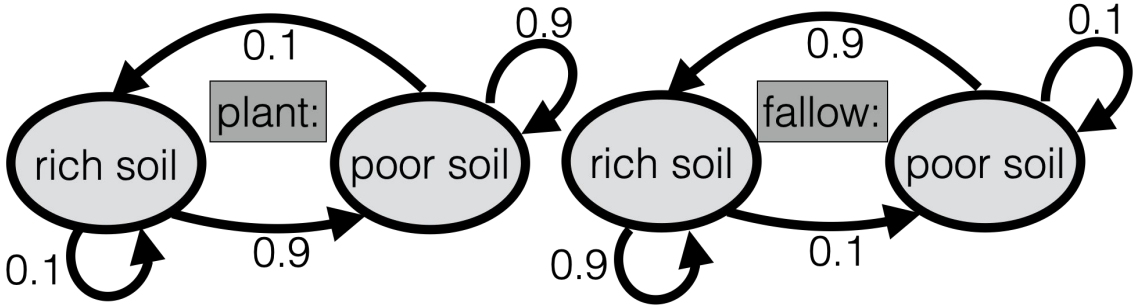
Review: Optimal policy in a known MDP



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

Review: Optimal policy in a known MDP

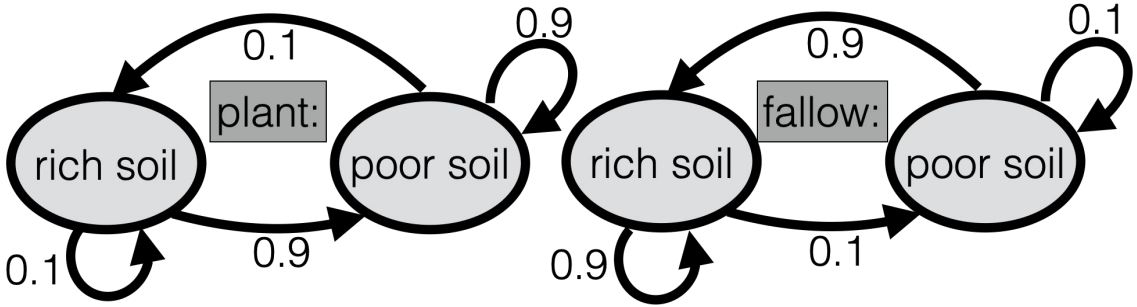


R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

Review: Optimal policy in a known MDP



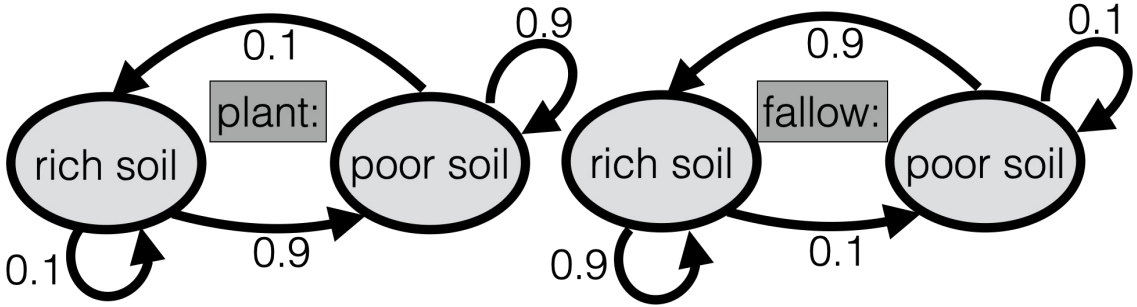
R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

$$Q^1(\text{rich}, \text{plant}) = 100; Q^1(\text{rich}, \text{fallow}) = 0;\ Q^1(\text{poor}, \text{plant}) = 10; Q^1(\text{poor}, \text{fallow}) = 0$$

Review: Optimal policy in a known MDP



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

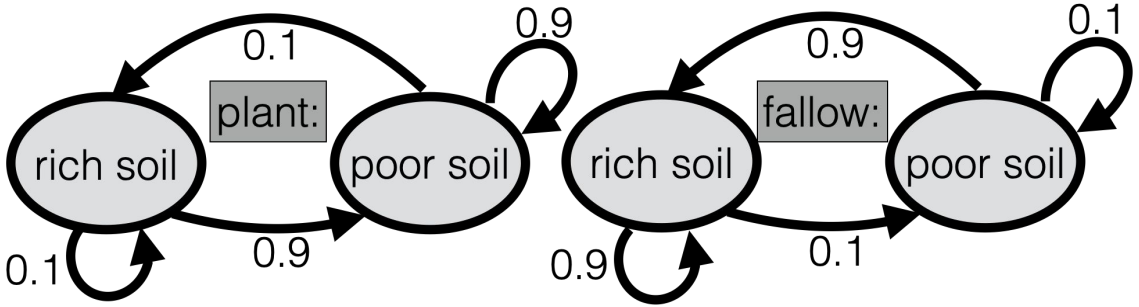
- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

$$Q^1(\text{rich}, \text{plant}) = 100; Q^1(\text{rich}, \text{fallow}) = 0;\ Q^1(\text{poor}, \text{plant}) = 10; Q^1(\text{poor}, \text{fallow}) = 0$$

$$Q^2(\text{rich}, \text{plant}) = R(\text{rich}, \text{plant}) + T(\text{rich}, \text{plant}, \text{rich}) \max_{a'} Q^1(\text{rich}, a') + T(\text{rich}, \text{plant}, \text{poor}) \max_{a'} Q^1(\text{poor}, a')$$

Review: Optimal policy in a known MDP



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

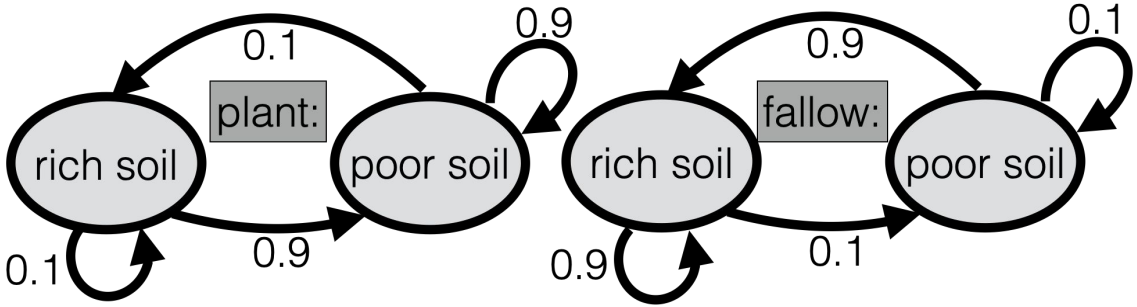
- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

$$Q^1(\text{rich}, \text{plant}) = 100; Q^1(\text{rich}, \text{fallow}) = 0;\ Q^1(\text{poor}, \text{plant}) = 10; Q^1(\text{poor}, \text{fallow}) = 0$$

$$Q^2(\text{rich}, \text{plant}) = 100 + (0.1)(100) + (0.9)(10) = 119$$

Review: Optimal policy in a known MDP



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

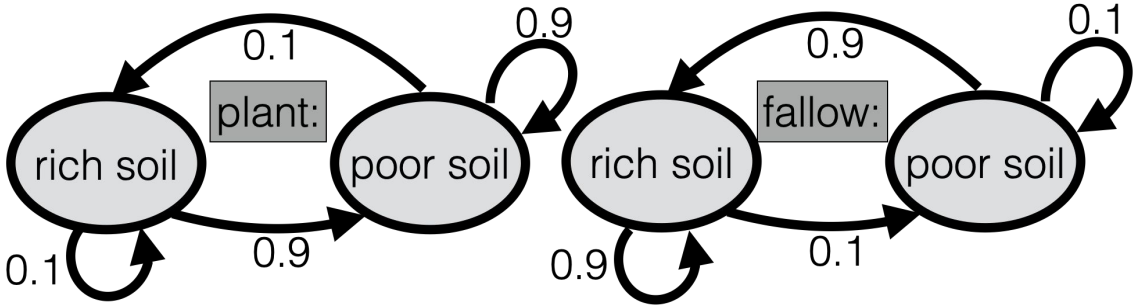
- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

$$Q^1(\text{rich}, \text{plant}) = 100; Q^1(\text{rich}, \text{fallow}) = 0;\ Q^1(\text{poor}, \text{plant}) = 10; Q^1(\text{poor}, \text{fallow}) = 0$$

$$Q^2(\text{rich}, \text{plant}) = 119; Q^2(\text{rich}, \text{fallow}) = 91; Q^2(\text{poor}, \text{plant}) = 29; Q^2(\text{poor}, \text{fallow}) = 91$$

Review: (Finite-Horizon) Value Iteration



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- *h*: horizon (e.g. how many planting seasons)
- $Q^h(s,a)$: expected reward of starting at *s*, making action *a*, and then making the "best" action for the *h*-1 steps left
- With *Q*, can find **an optimal policy**: $\pi_h^*(s) = \arg\max_a Q^h(s,a)$

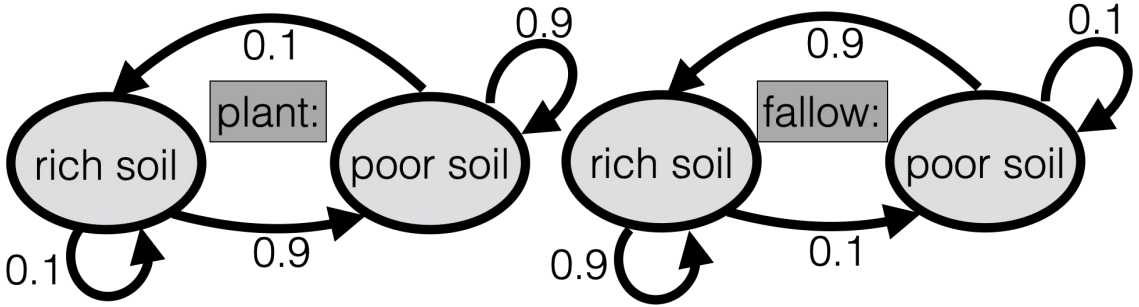
$$Q^0(s,a) = 0; Q^h(s,a) = R(s,a) + \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

$$Q^1(\text{rich}, \text{plant}) = 100; Q^1(\text{rich}, \text{fallow}) = 0;\ Q^1(\text{poor}, \text{plant}) = 10; Q^1(\text{poor}, \text{fallow}) = 0$$

$$Q^2(\text{rich}, \text{plant}) = 119; Q^2(\text{rich}, \text{fallow}) = 91; Q^2(\text{poor}, \text{plant}) = 29; Q^2(\text{poor}, \text{fallow}) = 91$$

What's best? Any *s*, $\pi_1^*(s) = \text{plant}; \pi_2^*(\text{rich}) = \text{plant}, \pi_2^*(\text{poor}) = \text{fallow}$

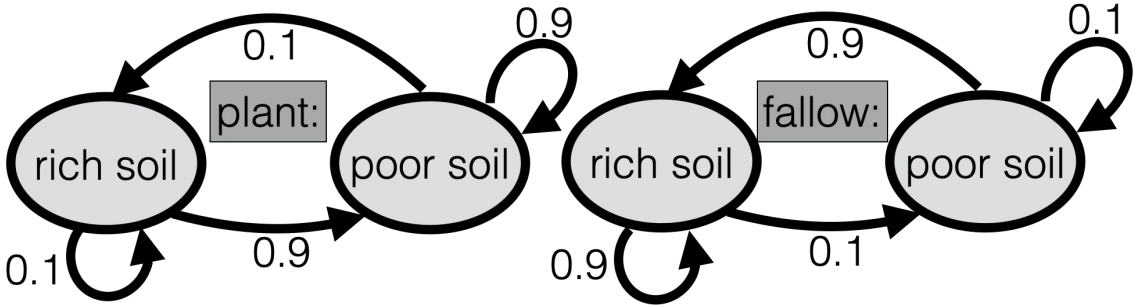
Review: (Infinite-Horizon) Value Iteration



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- What if I don't stop farming? Is there any optimal policy?
- **Theorem**. There exists a (stationary) optimal policy π^*. I.e., for every policy π and for every state $s \in \mathcal{S}$, $V_{\pi^*}(s) \geq V_{\pi}(s)$

Review: (Infinite-Horizon) Value Iteration



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0

- What if I don't stop farming? Is there any optimal policy?
- **Theorem**. There exists a (stationary) optimal policy π^*. I.e., for every policy π and for every state $s \in \mathcal{S}$, $V_{\pi^*}(s) \geq V_\pi(s)$
- $Q^*(s,a)$: expected reward if we make best actions in future
  - If we knew $Q^*(s,a)$, then: $\pi^*(s) = \arg\max_a Q^*(s,a)$
- Note: $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^*(s',a')$
  - Not linear in $Q^*(s,a)$, so not as easy to solve as $V_\pi(s)$

Review: (Infinite-Horizon) Value Iteration

Finite-horizon value iteration:

$$Q^0(s,a) = 0 \quad Q^1(s,a) = R(s,a)$$
$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

Review: (Infinite-Horizon) Value Iteration

Finite-horizon value iteration:

$$Q^0(s,a) = 0 \;\; Q^1(s,a) = R(s,a)$$
$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

`Infinite-Horizon-Value-Iteration` $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

  `for` `each state` $s \in \mathcal{S}$ `and each action` $a \in \mathcal{A}$

    `Initialize` $Q_{\text{old}}(s,a) = 0$

Review: (Infinite-Horizon) Value Iteration

Finite-horizon value iteration:

$$Q^0(s,a) = 0 \;\; Q^1(s,a) = R(s,a)$$
$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

`Infinite-Horizon-Value-Iteration` $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

```
  for each state s ∈ S and each action a ∈ A
    Initialize Q_old(s,a) = 0
  while True
    for each state s ∈ S and each action a ∈ A
      Q_new(s,a) = R(s,a) + γ Σ_s' T(s,a,s') max_a' Q_old(s',a')
```

Review: (Infinite-Horizon) Value Iteration

Finite-horizon value iteration:

$$Q^0(s,a) = 0 \;\; Q^1(s,a) = R(s,a)$$
$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

```
Infinite-Horizon-Value-Iteration(S, A, T, R, γ, ε)
  for each state s ∈ S and each action a ∈ A
    Initialize Q_old(s,a) = 0
  while True
    for each state s ∈ S and each action a ∈ A
      Q_new(s,a) = R(s,a) + γ Σ_s' T(s,a,s') max_a' Q_old(s',a')
    if max_{s,a} |Q_old(s,a) − Q_new(s,a)| < ε
      return Q_new
    Q_old = Q_new
```

Review: (Infinite-Horizon) Value Iteration

Finite-horizon value iteration:

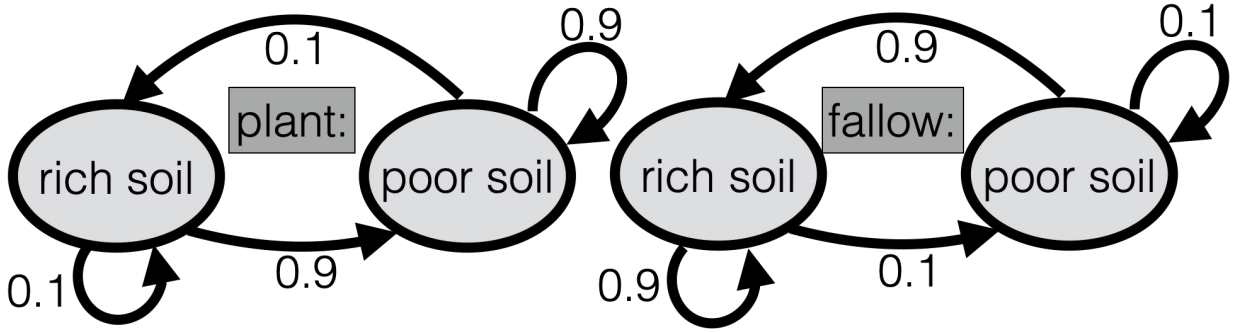
$$Q^0(s,a) = 0 \quad Q^1(s,a) = R(s,a)$$
$$Q^h(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^{h-1}(s',a')$$

`Infinite-Horizon-Value-Iteration` $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \epsilon)$

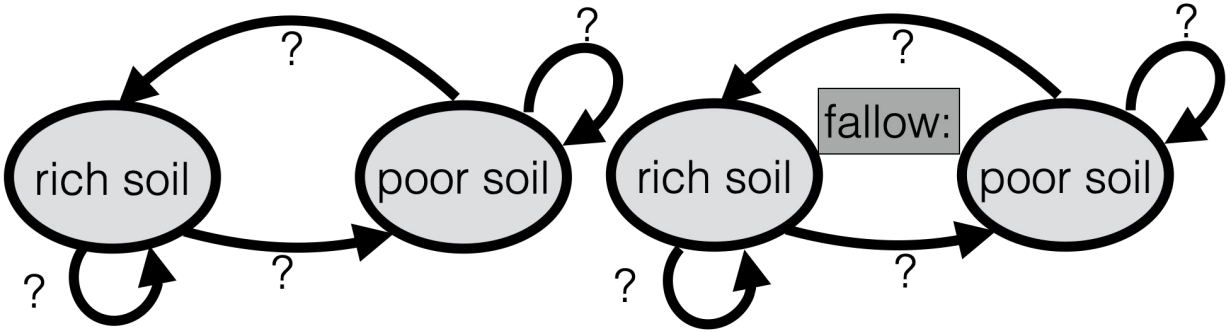
```
  for each state s ∈ S and each action a ∈ A
    Initialize Q_old(s,a) = 0
  while True
    for each state s ∈ S and each action a ∈ A
      Q_new(s,a) = R(s,a) + γ Σ_s' T(s,a,s') max_a' Q_old(s',a')
    if max_{s,a} |Q_old(s,a) − Q_new(s,a)| < ε
      return Q_new
    Q_old = Q_new
```

Issue: need to know reward and transition functions!

A more realistic scenario



R(rich, plant)=100
R(poor, plant)=10
R(rich, fallow)=0
R(poor, fallow)=0



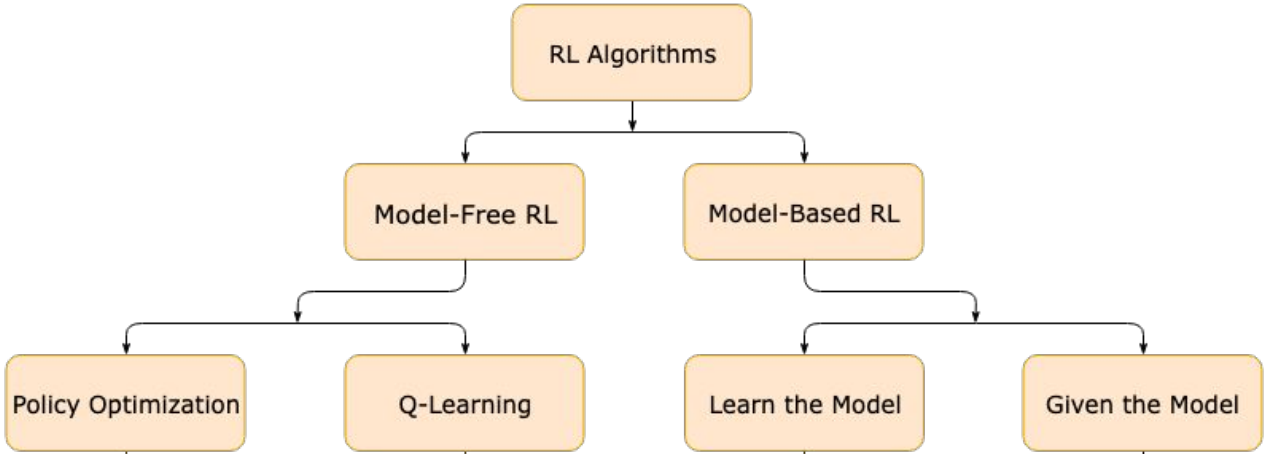
R(rich, plant)=?
R(poor, plant)=?
R(rich, fallow)=?
R(poor, fallow)=?

Reinforcement Learning Overview

Goal in RL: find a “policy” $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes reward in an **unknown** MDP.

Reinforcement Learning Overview

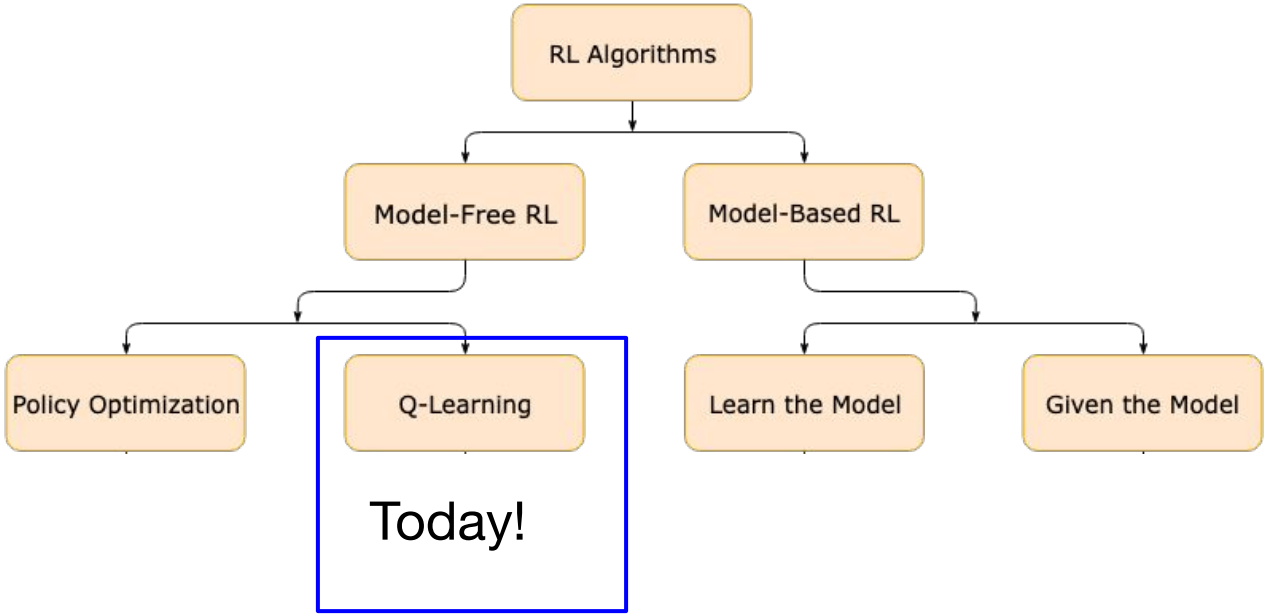
Goal in RL: find a "policy" $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes reward in an **unknown** MDP.



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Reinforcement Learning Overview

Goal in RL: find a "policy" $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes reward in an **unknown** MDP.



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$

Initialize Q function to 0

Q-Learning

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$ Initialize Q function to 0

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a = \text{select\_action}(s, Q)$ “Act” in the environment

Q-Learning

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$ Initialize Q function to 0

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a = \text{select\_action}(s, Q)$ "Act" in the environment

$r, s' = \text{execute}(a)$ Receive reward, transition to s'

Intuition: sampling to estimate the transition model

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$$

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$ — Initialize Q function to 0

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a = \text{select\_action}(s, Q)$ — "Act" in the environment

$r, s' = \text{execute}(a)$ — Receive reward, transition to s'

$Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

Update Q function for the sampled state and action

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$

Initialize Q function to 0

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a = \text{select\_action}(s, Q)$

"Act" in the environment

$r, s' = \text{execute}(a)$

Receive reward, transition to s'

$Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

$s = s'$

Move onto next state

Update Q function for the sampled state and action

What changed?

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:
    $Q[s, a] = 0$
$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
while True:
    $a =$ select_action(s, Q)
    $r, s' =$ execute(a)
    $Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
    $s = s'$

---

while True
  for each state $s \in \mathcal{S}$ and each action $a \in \mathcal{A}$
    $Q_{\text{new}}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$

What changed?

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:
    $Q[s, a] = 0$
$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
while True:
    $a = \text{select\_action}(s, Q)$
    $r, s' = \text{execute}(a)$
    $Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
    $s = s'$

---

while True
  for each state $s \in \mathcal{S}$ and each action $a \in \mathcal{A}$
    $Q_{\text{new}}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')$

Since we don't know R and T, we estimate it by "sampling" in the environment

What changed?

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a =$ select_action(s, Q)

$r, s' =$ execute(a)

$Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

$s = s'$

Moving average

What changed?

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:

$Q[s, a] = 0$

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$

while True:

$a = \text{select\_action}(s, Q)$

$r, s' = \text{execute}(a)$

$Q[s, a] = (1 - \alpha) Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$

$s = s'$

Moving average

$$Q[s, a] = Q[s, a] - \alpha \left(Q[s, a] - (r + \gamma \max_{a'} Q[s', a']) \right)$$

Looks like gradient descent!

Q-Learning

$$Q[s,a] = Q[s,a] - \alpha\left(Q[s,a] - (r + \gamma \max_{a'} Q[s',a'])\right)$$

Linear regression with just the bias term

$$y = \theta_0$$

$$L(\theta_0) = \frac{1}{2}(y - \theta_0)^2$$

$$\theta_0 = \theta_0 - \eta(\theta_0 - y)$$

Q-Learning

$$Q[s,a] = Q[s,a] - \alpha\left(Q[s,a] - (r + \gamma \max_{a'} Q[s',a'])\right)$$

Linear regression with just the bias term

$$y = \theta_0$$

$$L(\theta_0) = \frac{1}{2}(y - \theta_0)^2$$

$$\theta_0 = \theta_0 - \eta(\theta_0 - y)$$

Current guess

Target

Q-Learning

$$Q[s,a] = Q[s,a] - \alpha\left(Q[s,a] - (r + \gamma \max_{a'} Q[s',a'])\right)$$

Linear regression with just the bias term

$$y = \theta_0$$

$$L(\theta_0) = \frac{1}{2}(y - \theta_0)^2$$

$$\theta_0 = \theta_0 - \eta(\theta_0 - y)$$

Learning rate

Current guess

Target

Q-Learning

$$Q[s,a] = Q[s,a] - \alpha\left(Q[s,a] - (r + \gamma \max_{a'} Q[s',a'])\right)$$

Linear regression with just the bias term

$$y = \theta_0$$

$$L(\theta_0) = \frac{1}{2}(y - \theta_0)^2$$

$$\theta_0 = \theta_0 - \eta(\theta_0 - y)$$

Learning rate

Current guess

Target

Key difference: in Q-learning, the target itself is a function of what is being learned (and the reward)!

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:
    $Q[s, a] = 0$
$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
while True:
    $a =$ select_action(s, Q)
    $r, s' =$ execute(a)
    $Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
    $s = s'$

How to select an action?

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:
    $Q[s, a] = 0$
$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
while True:
    $a =$ select_action(s, Q)
    $r, s' =$ execute(a)
    $Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
    $s = s'$

How to select an action?

Argmax action from current policy?

$$\arg\max_{a \in \mathcal{A}} Q(s, a)$$

Initially Q function is bad ⇒ argmax not a good idea.

Q-Learning

Q-LEARNING$(\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha)$

for $s \in \mathcal{S}, a \in \mathcal{A}$:
    $Q[s, a] = 0$
$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
while True:
    $a =$ select_action(s, Q)
    $r, s' =$ execute(a)
    $Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
    $s = s'$

How to select an action?

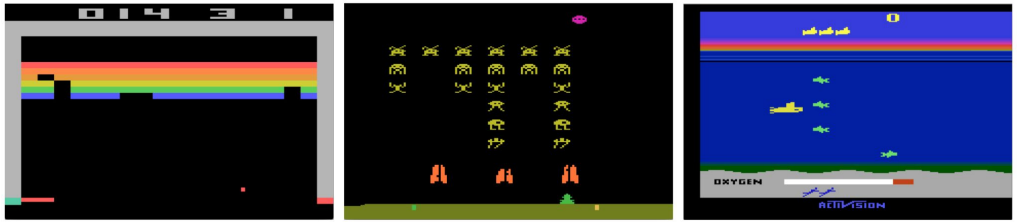
ε-greedy strategy:

- with probability $1 - \epsilon$, choose $\arg\max_{a \in \mathcal{A}} Q(s, a)$ — Exploitation
- with probability ϵ, choose the action $a \in \mathcal{A}$ uniformly at random — Exploration

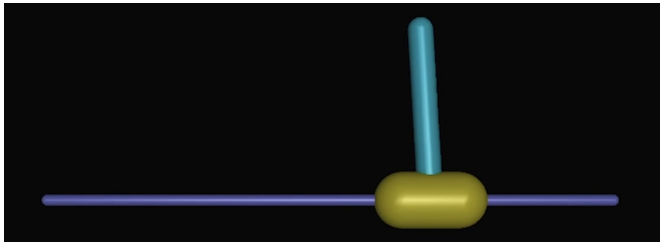
Q-Learning

- Q[s,a] is a scalar for each possible state and action. (“Tabular” Q-learning)

- What if states are high dimensional or continuous?



- What if actions are not discrete?

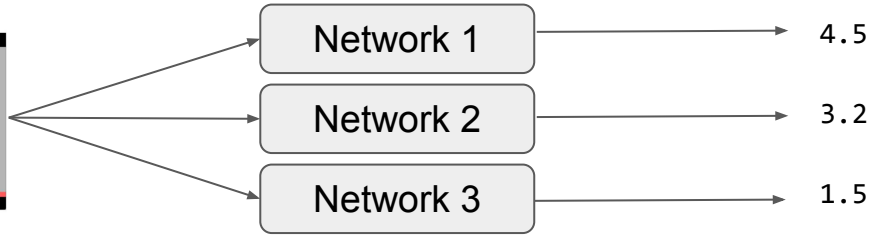


DQN: Deep Q-Networks

- Parameterize Q function to be an output from a neural network!

DQN: Deep Q-Networks

State



Separate network for each action (3 actions)

DQN: Deep Q-Networks

State

Network 1

Network 2

Network 3

4.5

3.2

1.5

Separate network for each action (3 actions)

State

Neural Network

$$\begin{bmatrix} 4.5 \\ 3.2 \\ 1.5 \end{bmatrix}$$

Score for each action

DQN: Deep Q-Networks

State

Network 1

Network 2

Network 3

4.5

3.2

1.5

Separate network for each action (3 actions)

State

Neural Network

4.5
3.2
1.5

Score for each action

State

Action (one-hot for discrete)

Neural Network

1.5

Score corresponding to input action

DQN: Deep Q-Networks

State

Network 1

Network 2

Network 3

4.5

3.2

1.5

Separate network for each action (3 actions)

State

Neural Network

4.5
3.2
1.5

Score for each action

What are the pros and cons of these approaches?

State

Action (one-hot for discrete)

Neural Network

1.5

Score corresponding to input action

DQN: Deep Q-Networks

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

1 **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2 $\quad Q[s, a] = 0$
3 $s = s_0$ // Or draw an s randomly from $\mathcal{S}$
4 **while** True:
5 $\quad a = \text{select\_action}(s, Q)$
6 $\quad r, s' = \text{execute}(a)$
7 $\quad Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
8 $\quad s = s'$

DQN: Deep Q-Networks

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

1 **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2 $\quad Q[s, a] = 0$

Randomly initialize $Q_\theta(s, a)$

3 $s = s_0$ // Or draw an s randomly from $\mathcal{S}$
4 **while** True:
5 $\quad a = \text{select\_action}(s, Q)$
6 $\quad r, s' = \text{execute}(a)$
7 $\quad Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
8 $\quad s = s'$

DQN: Deep Q-Networks

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

```
1  for s ∈ S, a ∈ A :
2      Q[s, a] = 0
3  s = s_0 // Or draw an s randomly from S
4  while True:
5      a = select_action(s, Q)
6      r, s' = execute(a)
7      Q[s, a] = (1 − α)Q[s, a] + α(r + γ max_a' Q[s', a'])
8      s = s'
```

Randomly initialize $Q_\theta(s, a)$

Minimize regression loss

$$(Q_\theta(s, a) - (r + \gamma \max_{a'} Q_\theta(s', a')))^2$$

DQN: Deep Q-Networks

Q-LEARNING($\mathcal{S}, \mathcal{A}, s_0, \gamma, \alpha$)

1 **for** $s \in \mathcal{S}, a \in \mathcal{A}$:
2 $\quad Q[s, a] = 0$
3 $s = s_0$ // Or draw an s randomly from $\mathcal{S}$
4 **while** True:
5 $\quad a = \text{select\_action}(s, Q)$
6 $\quad r, s' = \text{execute}(a)$
7 $\quad Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'])$
8 $\quad s = s'$

Randomly initialize $Q_\theta(s, a)$

Minimize regression loss

$$(Q_\theta(s, a) - (r + \gamma \max_{a'} Q_\theta(s', a')))^2$$

Issue: instability arising from both guess and target being from a learned network.

DQN: Fitted Q-Learning

FITTED-Q-LEARNING($\mathcal{A}, s_0, \gamma, \alpha, \epsilon, m$)

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
$\mathcal{D} = \{\}$
initialize neural-network representation of Q
while True:
  $\mathcal{D}_{new}$ = experience from executing ϵ-greedy policy based on Q for m steps
  $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{new}$ represented as (s, a, r, s') tuples
  $D_{sup} = \{(x^{(i)}, y^{(i)})\}$ where $x^{(i)} = (s, a)$ and $y^{(i)} = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$
    for each tuple $(s, a, r, s')^{(i)} \in \mathcal{D}$
  re-initialize neural-network representation of Q
  Q = supervised_NN_regression(D_{sup})

DQN: Fitted Q-Learning

FITTED-Q-LEARNING($\mathcal{A}, s_0, \gamma, \alpha, \epsilon, m$)

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
$\mathcal{D} = \{\}$
initialize neural-network representation of Q
while True:
  $\mathcal{D}_{new}$ = experience from executing ϵ-greedy policy based on Q for m steps
  $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{new}$ represented as (s, a, r, s') tuples
  $D_{sup} = \{(x^{(i)}, y^{(i)})\}$ where $x^{(i)} = (s, a)$ and $y^{(i)} = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$
    for each tuple $(s, a, r, s')^{(i)} \in \mathcal{D}$
  re-initialize neural-network representation of Q
  Q = supervised_NN_regression(D_{sup})

Collect data from current policy

DQN: Fitted Q-Learning

FITTED-Q-LEARNING($\mathcal{A}, s_0, \gamma, \alpha, \epsilon, m$)

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
$\mathcal{D} = \{\}$
initialize neural-network representation of Q
while True:

Collect data from current policy

$\mathcal{D}_{new}$ = experience from executing ϵ-greedy policy based on Q for m steps
$\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{new}$ represented as (s, a, r, s') tuples
$D_{sup} = \{(x^{(i)}, y^{(i)})\}$ where $x^{(i)} = (s, a)$ and $y^{(i)} = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$
for each tuple $(s, a, r, s')^{(i)} \in \mathcal{D}$

Create supervised learning dataset

re-initialize neural-network representation of Q
$Q = \text{supervised\_NN\_regression}(D_{sup})$

DQN: Fitted Q-Learning

FITTED-Q-LEARNING($\mathcal{A}, s_0, \gamma, \alpha, \epsilon, m$)

$s = s_0$ // Or draw an s randomly from $\mathcal{S}$
$\mathcal{D} = \{\}$
initialize neural-network representation of Q
while True:

Collect data from current policy

$\mathcal{D}_{new}$ = experience from executing ϵ-greedy policy based on Q for m steps
$\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{new}$ represented as (s, a, r, s') tuples
$D_{sup} = \{(x^{(i)}, y^{(i)})\}$ where $x^{(i)} = (s, a)$ and $y^{(i)} = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$
for each tuple $(s, a, r, s')^{(i)} \in \mathcal{D}$

Create supervised learning dataset

re-initialize neural-network representation of Q
Q = supervised_NN_regression(D_{sup})

Train a neural network with regression loss!