

Sign-in Form

bit.ly/2VN3vhM





Deep Reinforcement Learning

Learning from trial and error

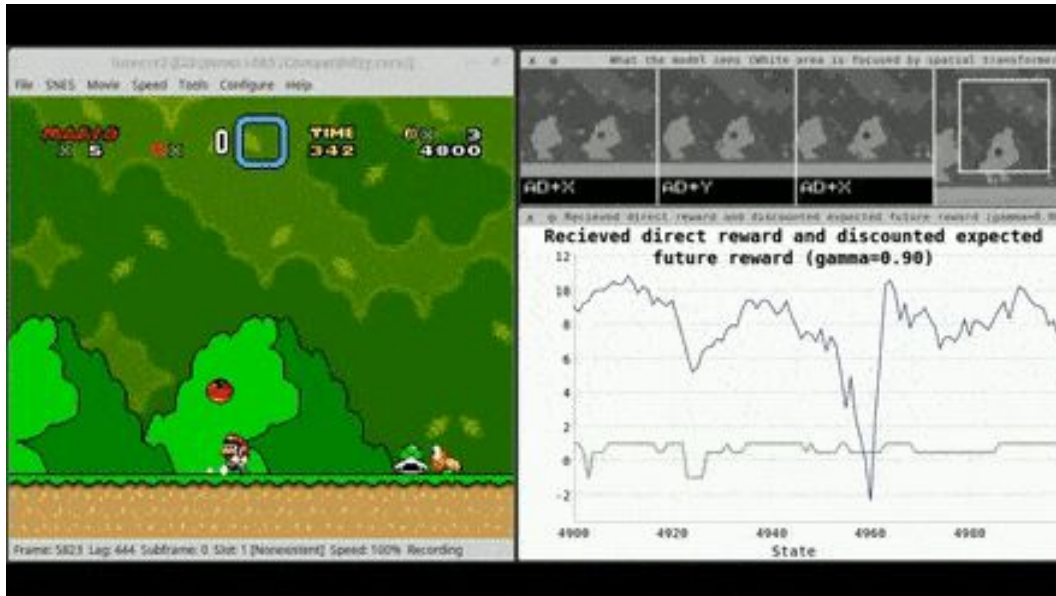


Reinforcement Learning Framework

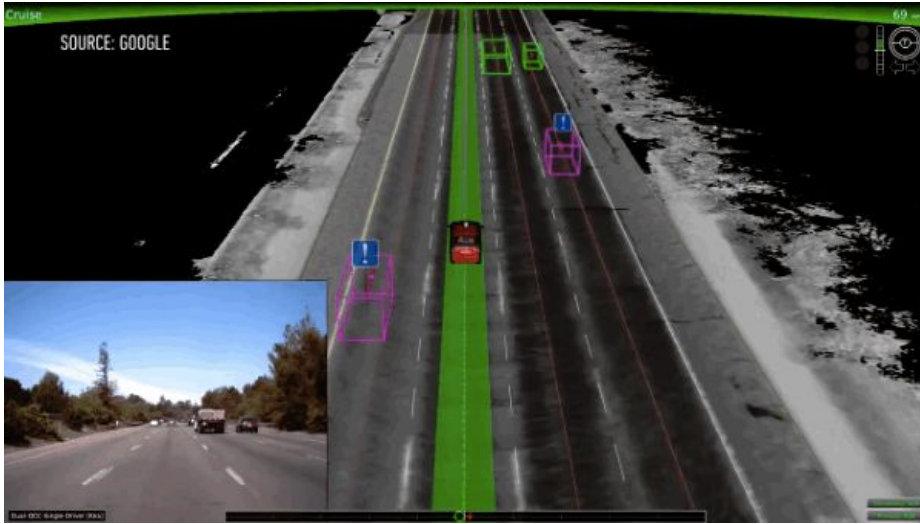
State, Action, Policy...

What is Reinforcement Learning?

- The process of learning by trial and error.
- Uses a reward system to tell how good a state is.
- The agent tries to maximize the amount of reward it can get.



Examples of Reinforcement Learning



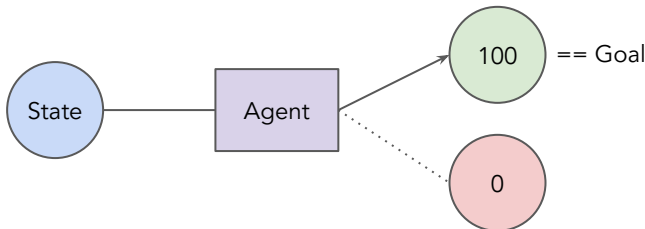
Reward

1. A **reward** R_t is a scalar feedback signal
2. Indicates how well agent is doing at step t
3. The agent's job is to maximise cumulative reward (aka. return G_t)

Reinforcement learning is based on the reward hypothesis

Reward Hypothesis

- All goals can be described by the maximisation of expected cumulative reward



Sequential Decision Making

Goal = select actions to maximise total future reward

1. Actions may have long term consequences
2. Reward may be delayed
3. It may be better to sacrifice immediate reward to gain more long-term reward

Return

1. Return (G_t) is the sum of rewards over a bunch of states (trajectory).
2. Sometimes we value more immediate reward over long term reward.
3. To achieve giving less value to rewards over time, we discount the rewards.
 - a. The rate at which we discount is called the discount factor, γ , which is between 0 and 1.

Equation for return:

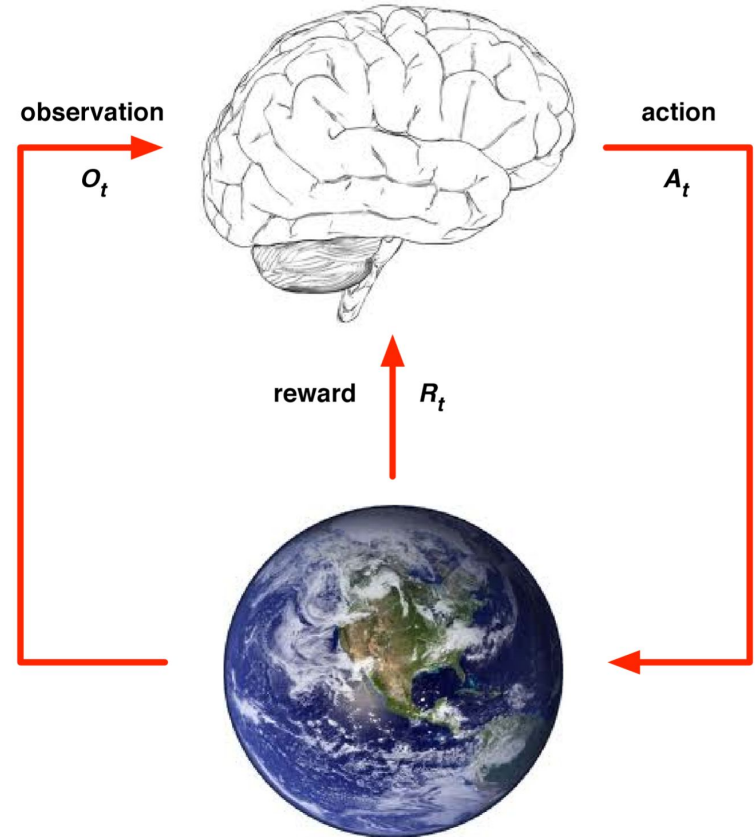
$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{T-t} R_T = \sum_{k=0}^{T-t} \gamma^k R_{t+k}$$

Common discount rates $\gamma = 0.9, 0.99$



Agent and Environment

1. At each step t , the agent:
 - a. Receives observation O_t
 - b. Receives scalar reward R_t
 - c. Executes action A_t
2. The environment:
 - a. Receives action A_t
 - b. Produces observation O_{t+1}
 - c. Produces scalar reward R_{t+1}
3. t increments at the environment step



History and State

History is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

State is the information used to determine what happens next

$$S_t = f(H_t)$$

$$S = [S_1, \dots, S_n]$$



Agent State and Environment State

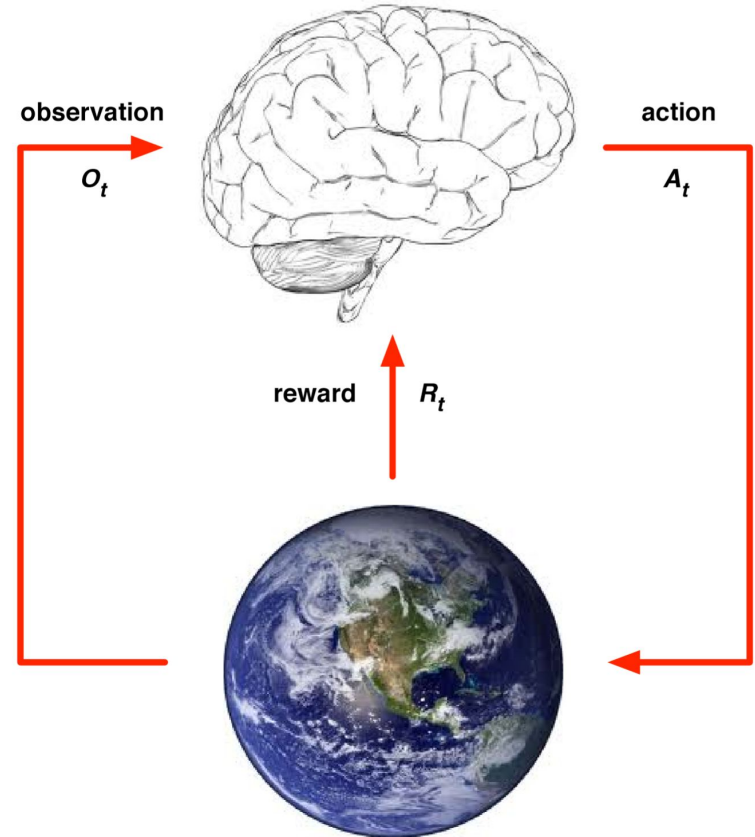
1. Agent state

- a. Information the agent uses to pick the next action
- b. An interpretation on historic sensor data
- c. $S_t^a = f(H_t)$

2. Environment state

- a. All information about the environment
- b. The environment state is not usually visible to the agent
- c. May contain irrelevant information for the agent
- d.

S_t^e



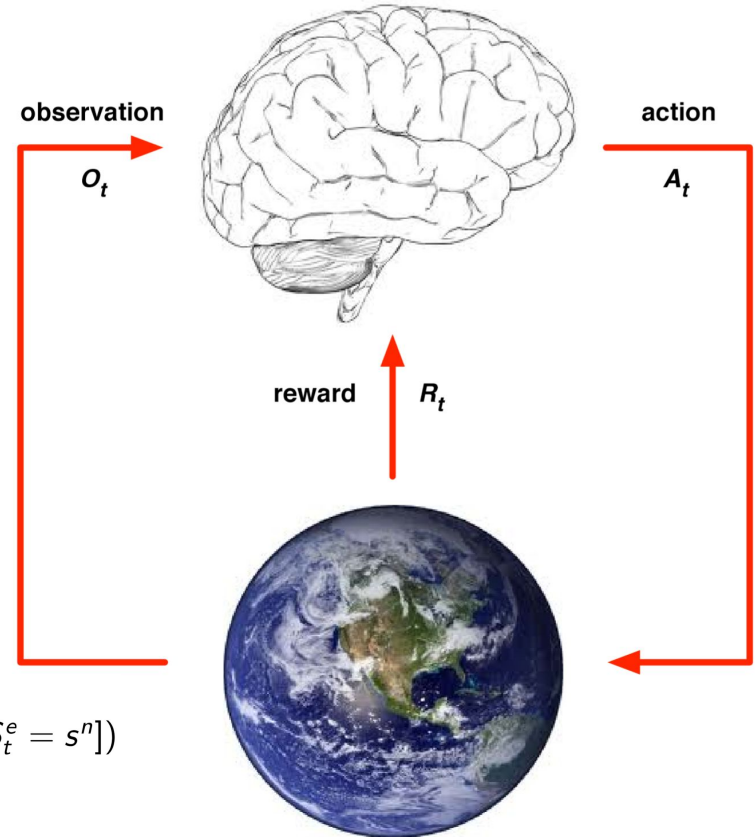
Information State

1. An information state (a.k.a. Markov state) contains all useful information from the history.
 - a. We want this to be equal to the agent state S_t^a so it learns efficiently
2. Markov State
 - a. $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t] = \mathbb{P}[S_{t+1}|H_t]$
 - b. “The future is independent of the past given the present”
 - c. If we assume the state is Markov in a RL problem, then we can throw away the history
 - i. This assumption allows us to design efficient RL algorithms



Fully and Partially Observable Environments

1. **Full observability**: agent directly observes environment state
 - a. $O_t = S_t^a = S_t^e$
 - b. = Markov Decision Process (MDP)
2. **Partial observability**: agent indirectly observes environment
 - a. $S_t^a \neq S_t^e$
 - b. = partially observable Markov decision process (POMDP)
 - c. Agent must construct its own state representation
 - Complete history: $S_t^a = H_t$
 - **Beliefs** of environment state: $S_t^a = (\mathbb{P}[S_t^e = s^1], \dots, \mathbb{P}[S_t^e = s^n])$
 - Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

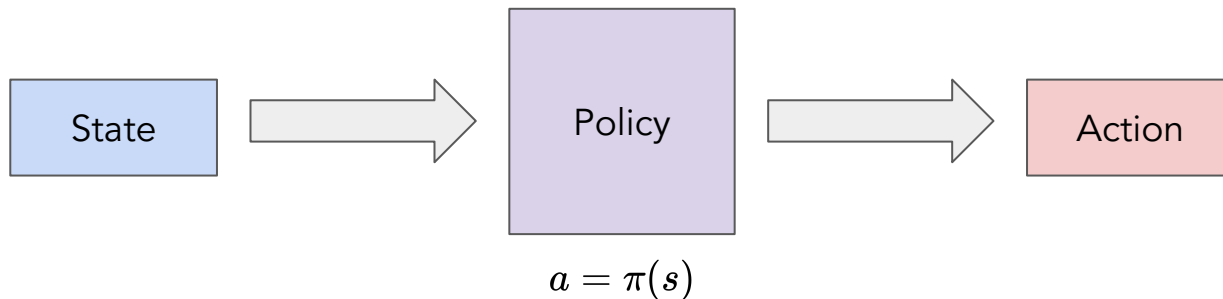


Major Reinforcement Learning Components

1. Policy: What is the strategy to choose actions?
 - a. Agent's behaviour function
2. Value Function: How good is this state and/or action?
 - a. Agent's prediction of future reward
3. Model: How does one state transition into the next?
 - a. Agent's representation of the environment

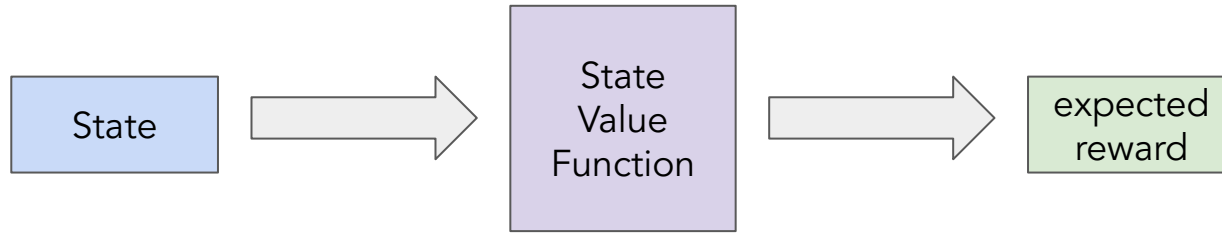
Policy

1. A **policy** is the agent's behaviour
2. It is a map from state to action
3. Deterministic policy: $a = \pi(s)$
4. Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

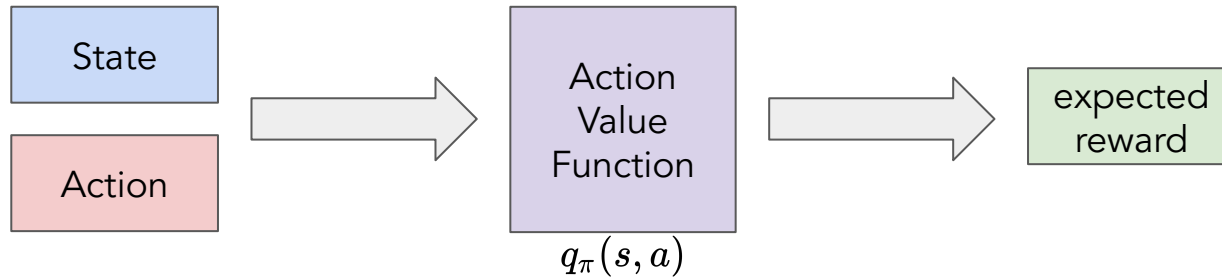


Value Function

1. A **value function** predicts future reward
2. It tells the goodness/badness of a given state and/or action

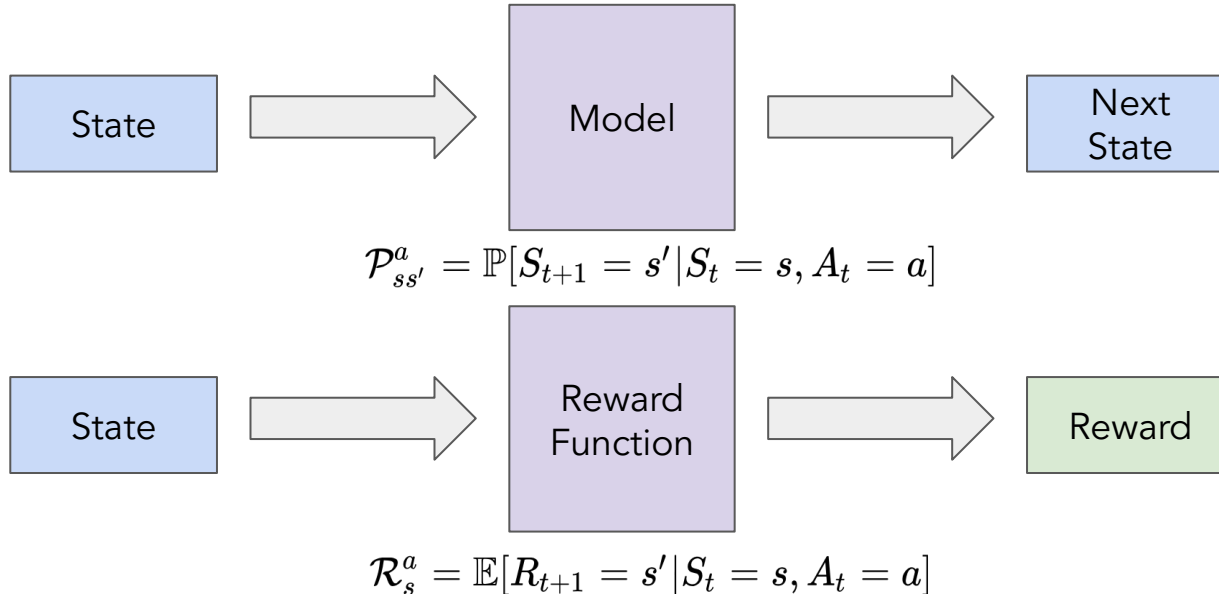


$$v_{\pi}(s) = \mathbb{E}_{\pi}[\underbrace{R_{t+1}}_{\text{expected}} + \underbrace{\gamma R_{t+2} + \gamma^2 R_{t+3} + \dots}_{\text{future (discounted) reward}} \mid \underbrace{S_t = s}_{\text{given state}}]$$

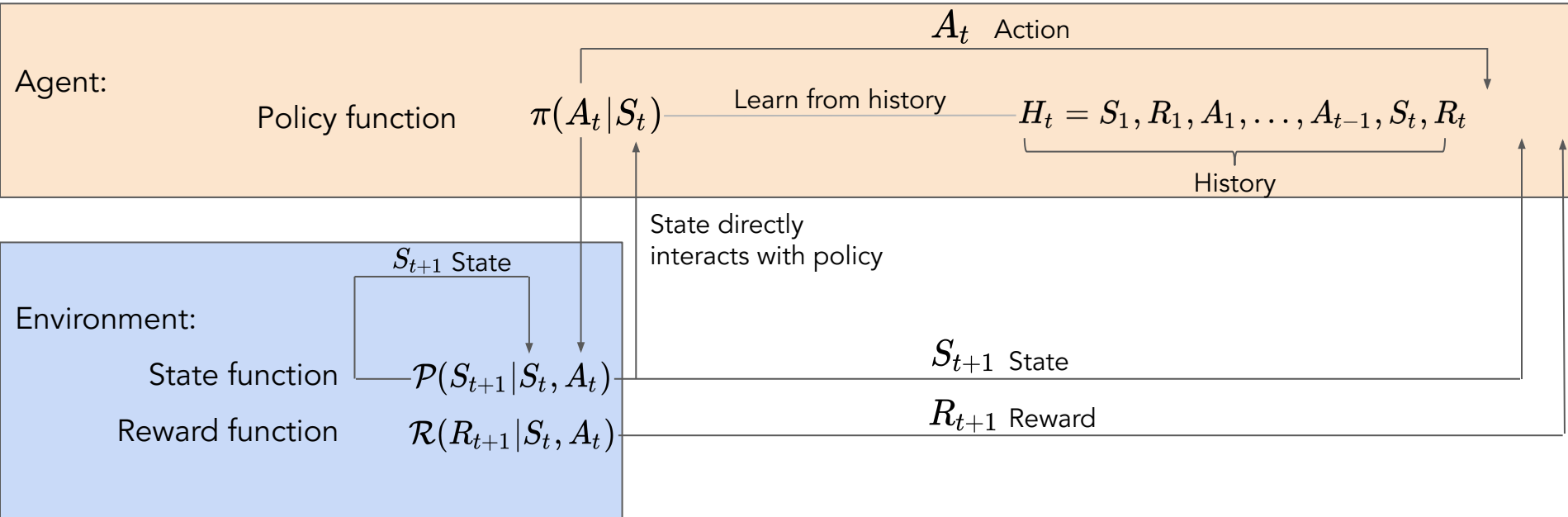


Model

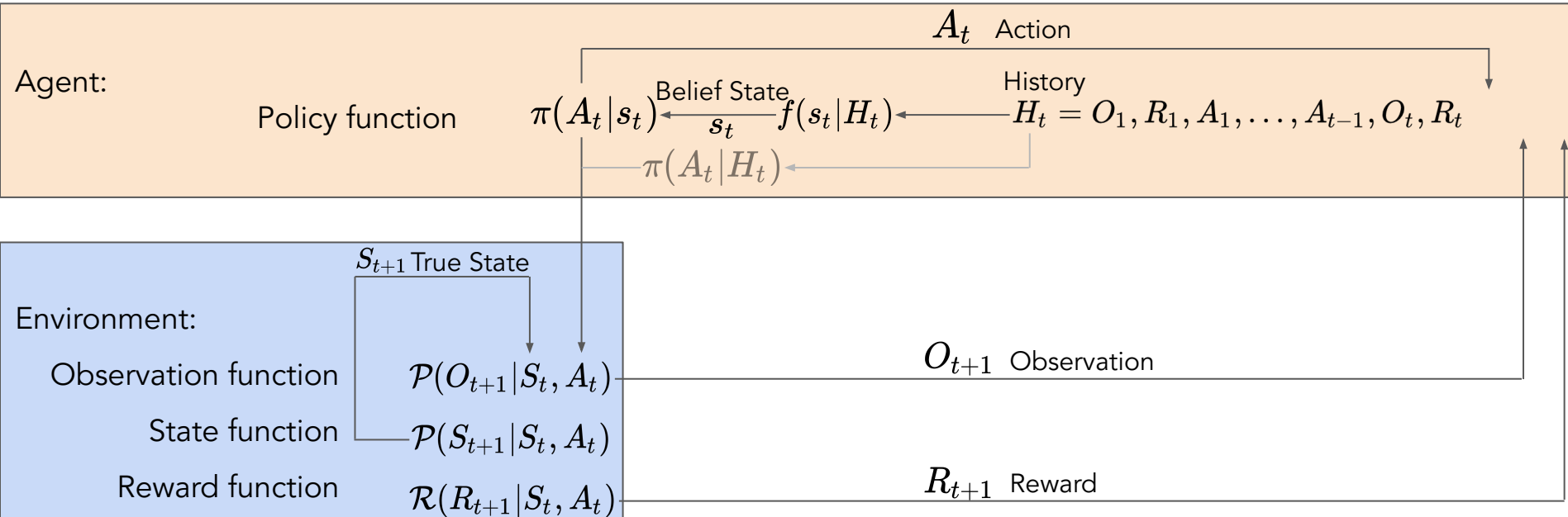
1. A **model** predicts what the environment will do next
2. P predicts the next state
3. R predicts the next (immediate) reward



The Full Picture (Full Observability)



The Full Picture (Partial Observability)





Reinforcement Learning Formulation

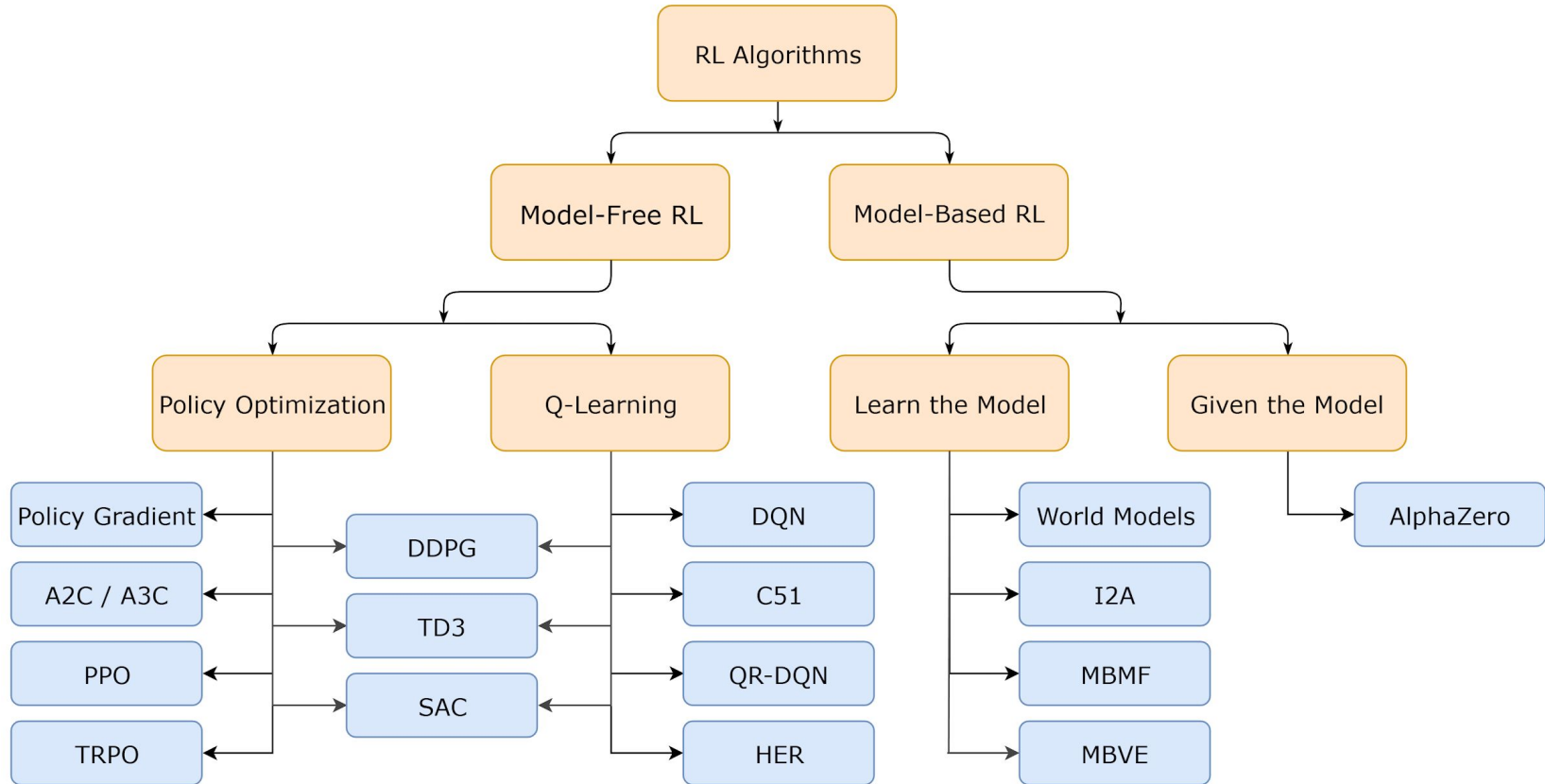
Markov Decision Process (MDP)

- Requirement: probability of transitioning to a given state is independent of any previous state transitions
 - $\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$
 - Fully observable environments are MDP
- This way, we can represent each state transition as (s, a, r, s')
 - (state, action, reward, next state)

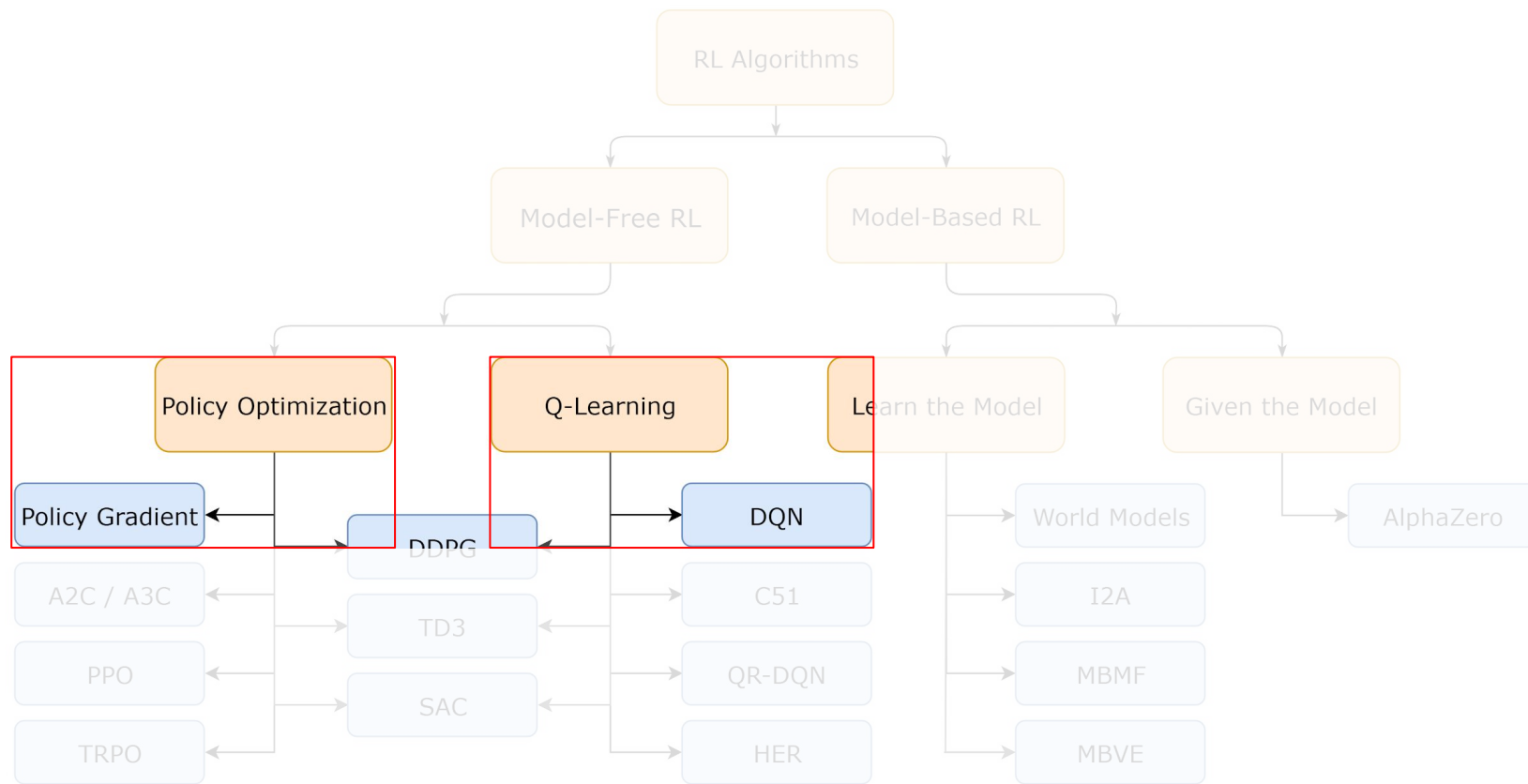


Deep RL Algorithms

Taxonomy of Deep RL algorithms

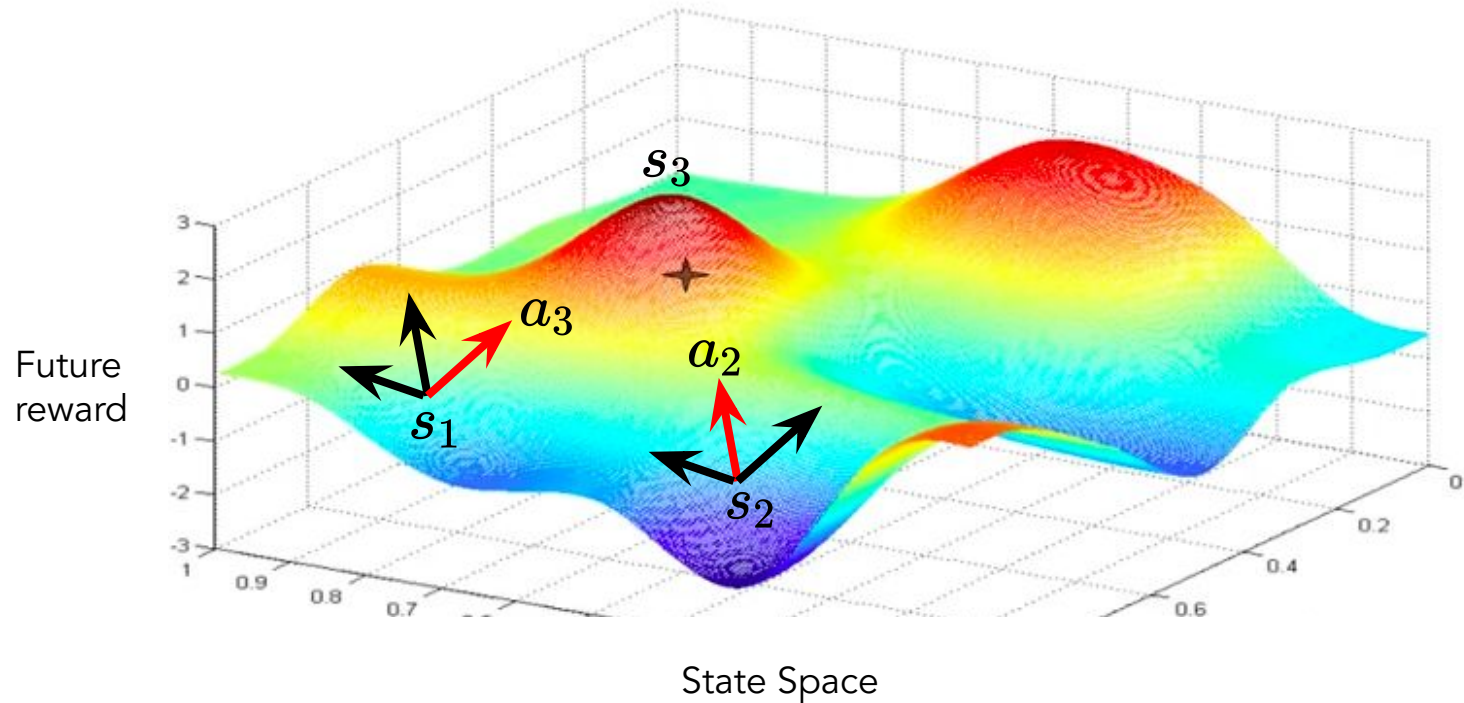


Taxonomy of Deep RL algorithms



Q-Learning in a nutshell

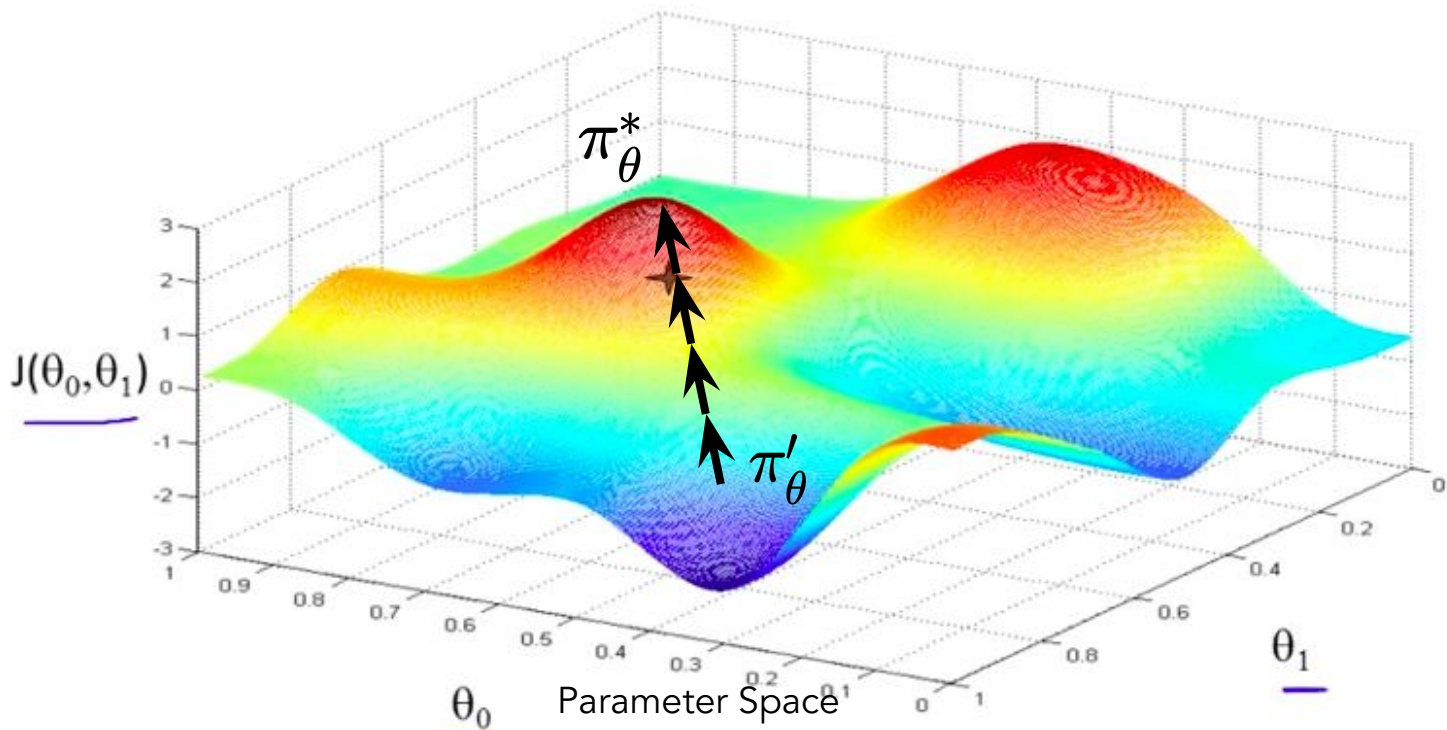
Learn which actions lead to the highest reward, and select the action with the maximum value.



Policy Optimization in a nutshell

Within the space of policy functions, π_θ , we try to pick parameters that produce the most reward.

Average
reward for
policy



Q-Learning vs. Policy optimization

1. Q-Learning

- a. Learns the value function, and indirectly the policy.
- b. is more sample efficient but hard to get working without tricks.

2. Policy optimization

- a. Learns the policy directly.
- b. is more robust but needs more samples.

3. Combining the two

- a. Actor critic is a combination of learning the policy and value function.





Policy Optimization

Policy Gradients, Actor Critic

Dataset

Our agent's experience is stored in a dataset of trajectories.

$$\mathcal{D} = \{\tau_i\}_{i=1, \dots, N}$$

A single trajectory contains states, actions, rewards, at each time step until the end of the episode. It is different from the history in that a trajectory is a slice of an episode of experience.



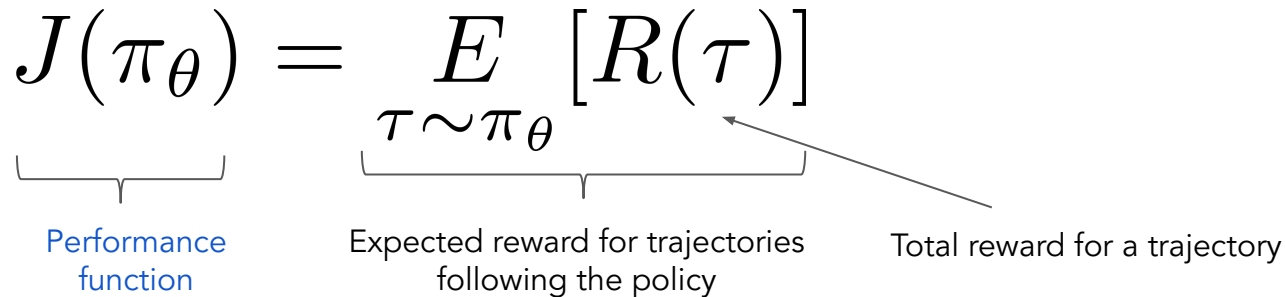
Policy Revisited

We want to pick actions given states so that we accomplish some goal.

$$\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$$

We now define how well our policy does with a performance function.

$$J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} [R(\tau)]$$



Performance function

Expected reward for trajectories following the policy

Total reward for a trajectory

This metric is similar to a cost function. By maximizing this score, we maximize reward & achieve our goal.



Learning a better policy

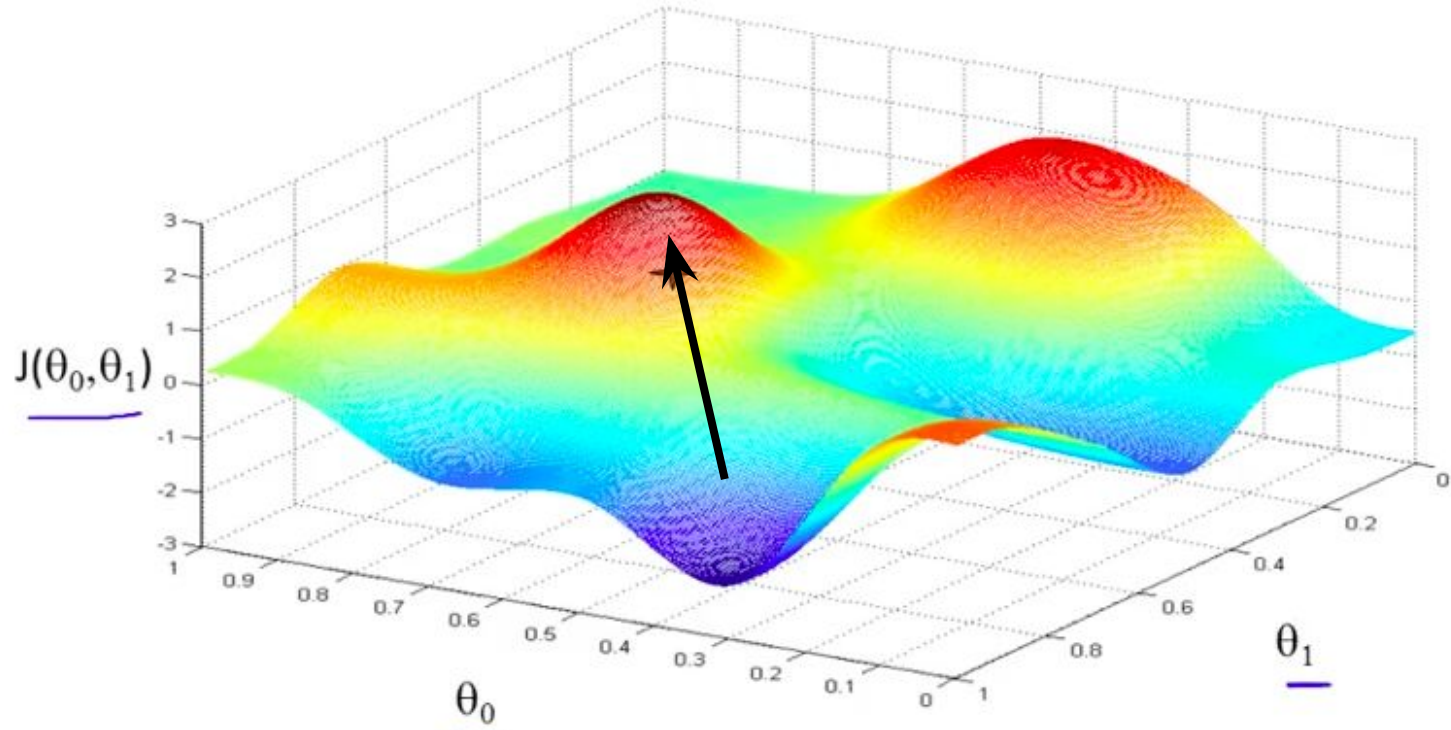
1. To make the agent learn, we want to find out **how to change the parameters in such a way to increase the performance function, $J(\pi_\theta)$** .
 - a. One answer is to use the gradients of the performance function with respect to the parameters, $\nabla J(\pi_\theta)$. This known as the **policy gradient**.
 - b. Using the gradient, we use gradient ascent to increase performance.

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$$



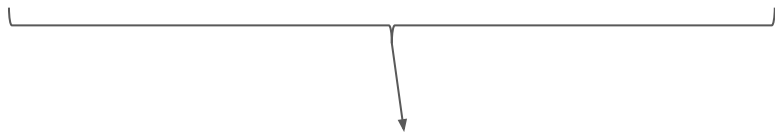
Gradient Ascent

We change the policy parameters in the direction of increasing performance.



Gradient Ascent

We change the policy parameters in the direction of increasing performance.



$$\underbrace{\theta_{k+1}}_{\text{Policy parameters for iteration } k+1} = \underbrace{\theta_k}_{\text{Policy parameters for iteration } k} + \underbrace{\alpha}_{\text{Learning rate}} \underbrace{\nabla_{\theta} J(\pi_{\theta})|_{\theta_k}}_{\text{Gradient of the performance function with respect to the policy parameters at iteration } k}$$

Policy parameters for
iteration k+1

Policy parameters for
iteration k

Learning
rate

Gradient of the performance function with
respect to the policy parameters at iteration k



Gradient Ascent

We change the policy parameters in the direction of increasing performance.

Q: How do we calculate the policy gradient?
A: We redefine it as a function of experience
(sampled trajectories)

$$\underbrace{\theta_{k+1}}_{\text{Policy parameters for iteration } k+1} = \underbrace{\theta_k}_{\text{Policy parameters for iteration } k} + \underbrace{\alpha}_{\text{Learning rate}} \underbrace{\nabla_{\theta} J(\pi_{\theta})|_{\theta_k}}_{\text{Gradient of the performance function with respect to the policy parameters at iteration } k}$$

Calculating the gradient (1)

1. Note that our current definition of the performance function does not contain the policy parameters inside the expectation.
 - a. So we cannot get the policy gradient from data sampled from the environment.
2. To solve this problem, we must derive a equation that contains the policy parameters, and have the ability to compute from sampled trajectories.

$$J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} [R(\tau)]$$

$$\nabla_{\theta} J(\pi_{\theta}) = ?$$

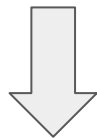


Calculating the gradient (2)

$$\nabla_{\theta} J(\pi_{\theta}) = \nabla_{\theta} \underbrace{E_{\tau \sim \pi_{\theta}} [R(\tau)]}_{\text{The expected reward of trajectories following the policy}}$$

We start by changing the expectation into probability form. However we are not done because we cannot calculate directly the probability of a trajectory from samples.

The expected reward of trajectories following the policy



$$= \nabla_{\theta} \underbrace{\int_{\tau} P(\tau|\theta) R(\tau)}_{\text{Is equivalent to the sum over the probability of a trajectory times the reward for such trajectory}}$$

Is equivalent to the sum over the probability of a trajectory times the reward for such trajectory



Calculating the gradient (3)

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \nabla_{\theta} \int_{\tau} P(\tau|\theta) R(\tau) \\ &\quad \Downarrow \\ &= \int_{\tau} \underbrace{\nabla_{\theta} P(\tau|\theta)}_{\text{Bring the gradient under the integral (calculus rules)}} R(\tau)\end{aligned}$$

Bring the gradient under the integral (calculus rules)

Calculating the gradient (4)

$$\nabla_{\theta} J(\pi_{\theta}) = \int_{\tau} \nabla_{\theta} P(\tau|\theta) R(\tau)$$

Multiply and divide by
the probability of a
trajectory

$$= \int_{\tau} P(\tau|\theta) \frac{\nabla_{\theta} P(\tau|\theta)}{P(\tau|\theta)} R(\tau)$$

The derivative of the
log is equivalent
 $d/dx \log x = (d/dx x)/x$

$$= \int_{\tau} P(\tau|\theta) \nabla_{\theta} \log P(\tau|\theta) R(\tau)$$



Calculating the gradient (4)

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \int_{\tau} P(\tau|\theta) \nabla_{\theta} \log P(\tau|\theta) R(\tau) \\ &= E_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log P(\tau|\theta) R(\tau)]\end{aligned}$$

Bring back the
integral into
expectation form

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[\underbrace{\nabla_{\theta} \log P(\tau|\theta) R(\tau)}_{\text{Let's elaborate on the trajectory}} \right]$$

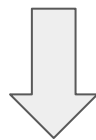
$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^T P(s_{t+1}|s_t, a_t) \pi_{\theta}(a_t|s_t)$$

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^T \left(\log P(s_{t+1}|s_t, a_t) + \log \pi_{\theta}(a_t|s_t) \right)$$

$$\begin{aligned} \nabla_{\theta} \log P(\tau|\theta) &= \cancel{\nabla_{\theta} \log \rho_0(s_0)} + \sum_{t=0}^T \left(\cancel{\nabla_{\theta} \log P(s_{t+1}|s_t, a_t)} + \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \right) \\ &= \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \end{aligned}$$

Calculating the gradient (5)

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log P(\tau | \theta) R(\tau)]$$



$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right]$$

Final form of the policy gradient.

Calculating the gradient from data

Final policy gradient:

$$\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right]$$

Generate an estimate of the policy gradient from a dataset of trajectories:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau)$$

Perform gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \hat{g}$$

Algorithm 1 Vanilla Policy Gradient Algorithm

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) |_{\theta_k} \hat{A}_t.$$

- 7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.

- 8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 9: **end for**



Policy Gradient Implementation

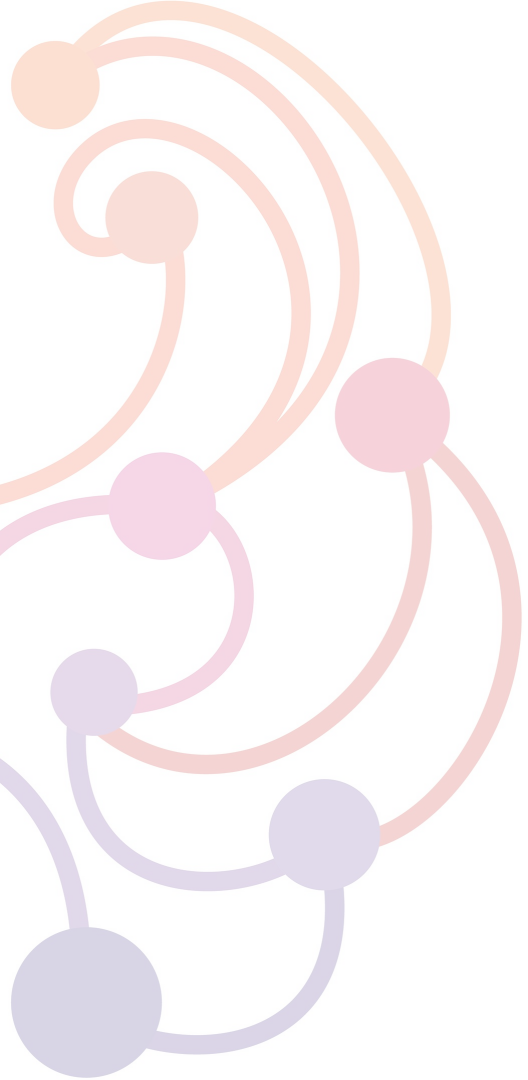
<https://colab.research.google.com/drive/1f1AsgnqRSHaBMj0-jQfW4rUsdcI4IXks>



References and Further Reading

1. Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. The MIT Press, 2018. <http://incompleteideas.net/book/the-book-2nd.html>
2. Spinning Up in Deep RL: <https://spinningup.openai.com/en/latest/index.html>





Thank you for coming!