Deep Reinforcement Learning

Learning from trial and error

BOSTON UNIVERSITY
MACHINE INTELLIGENCE
COMMUNITY

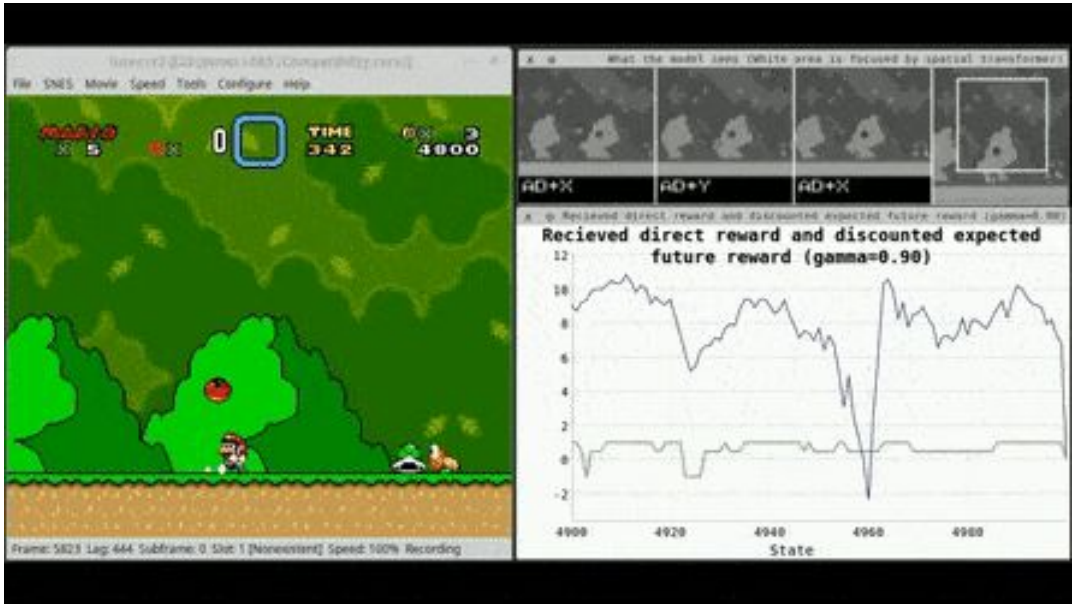
Julius, Darcy, Curtis
11/6/2019

Reinforcement Learning Framework

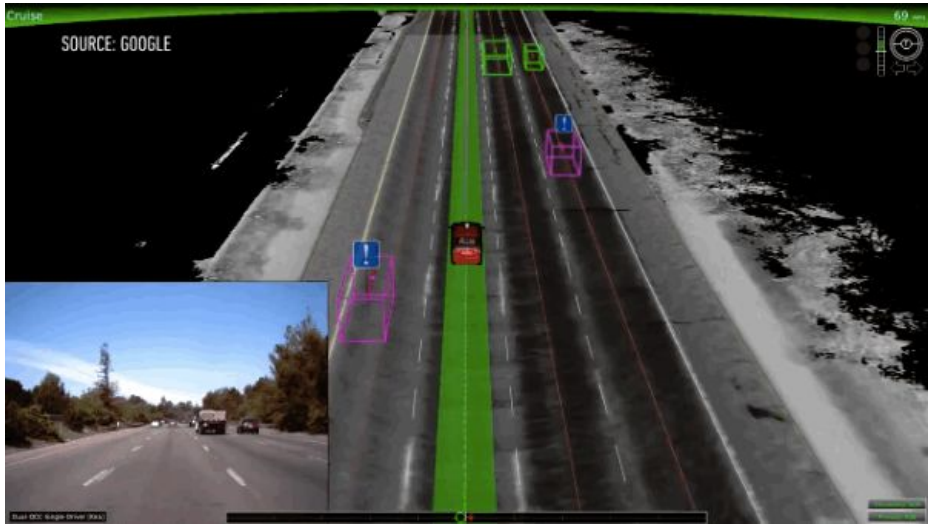
State, Action, Policy...

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



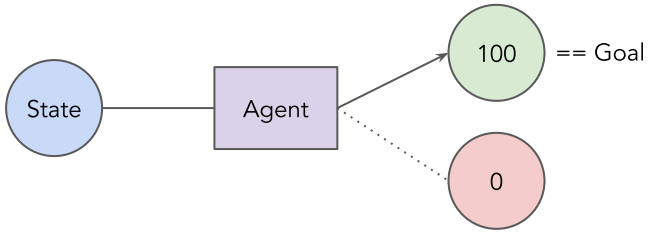
Reward

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.

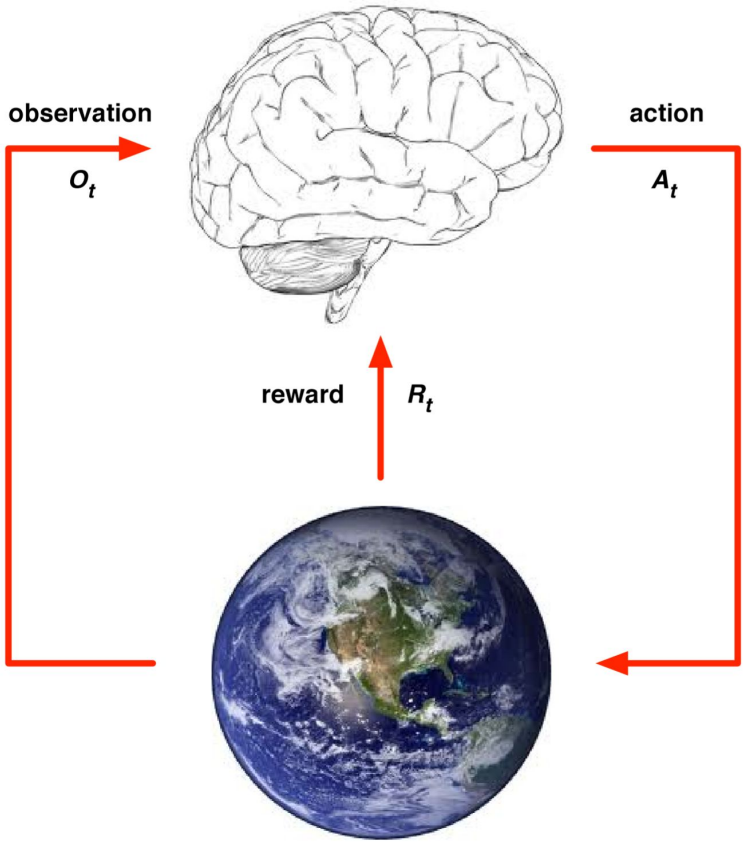
Equation for return:

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

Common discount rates γ= 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

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

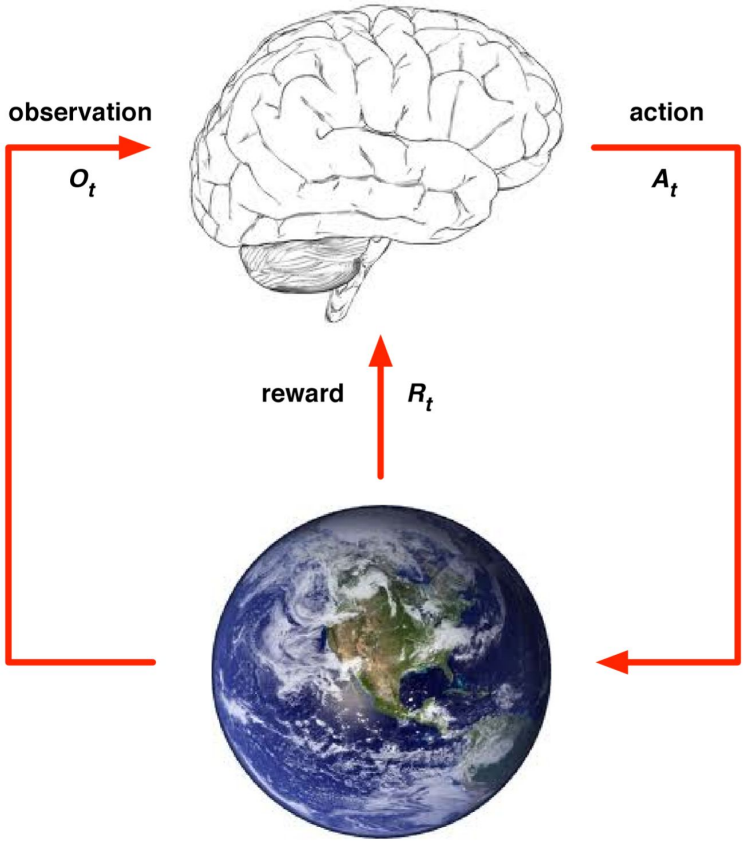
State is the information used to determine what happens next

$$S_t = f(H_t)$$

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

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

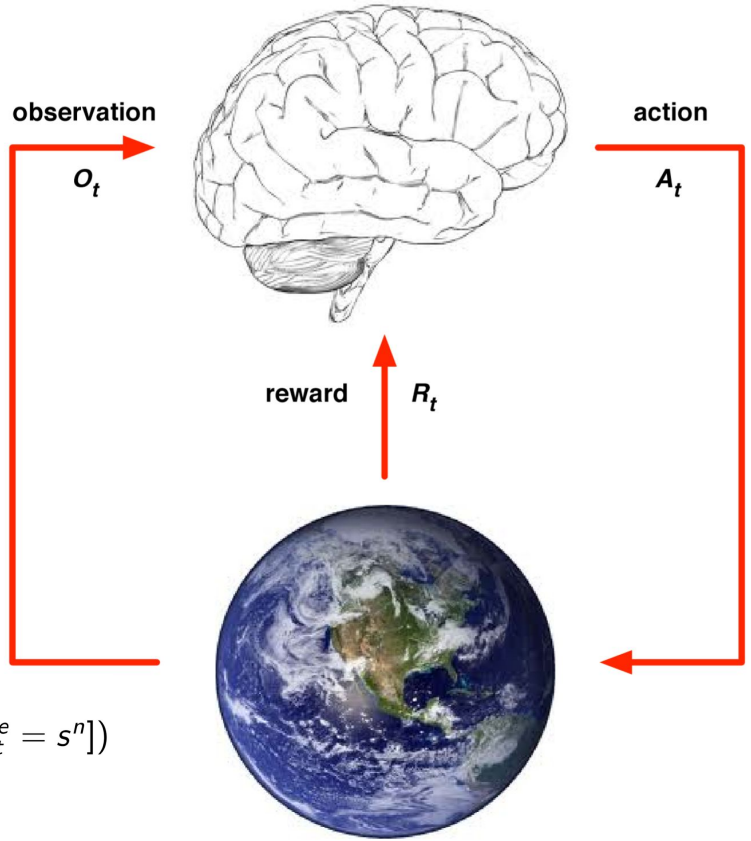


Information State

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. $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t] = \mathbb{P}[S_t + 1|H_t]$
    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: $S_t^a = (\mathbb{P}[S_t^e = s^1], \ldots, \mathbb{P}[S_t^e = s^n])$
      - Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

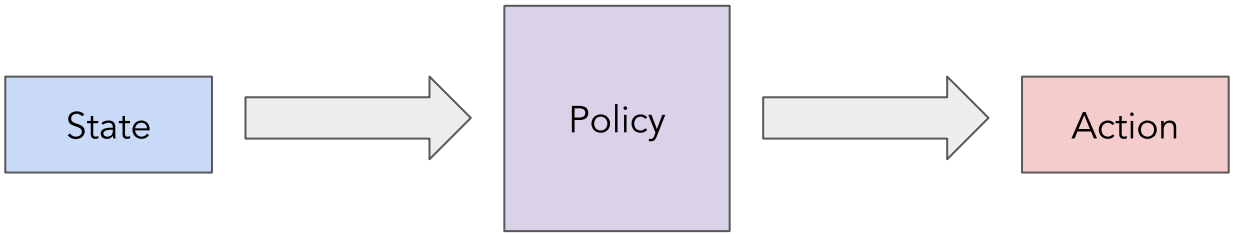


Major Reinforcement Learning Components

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

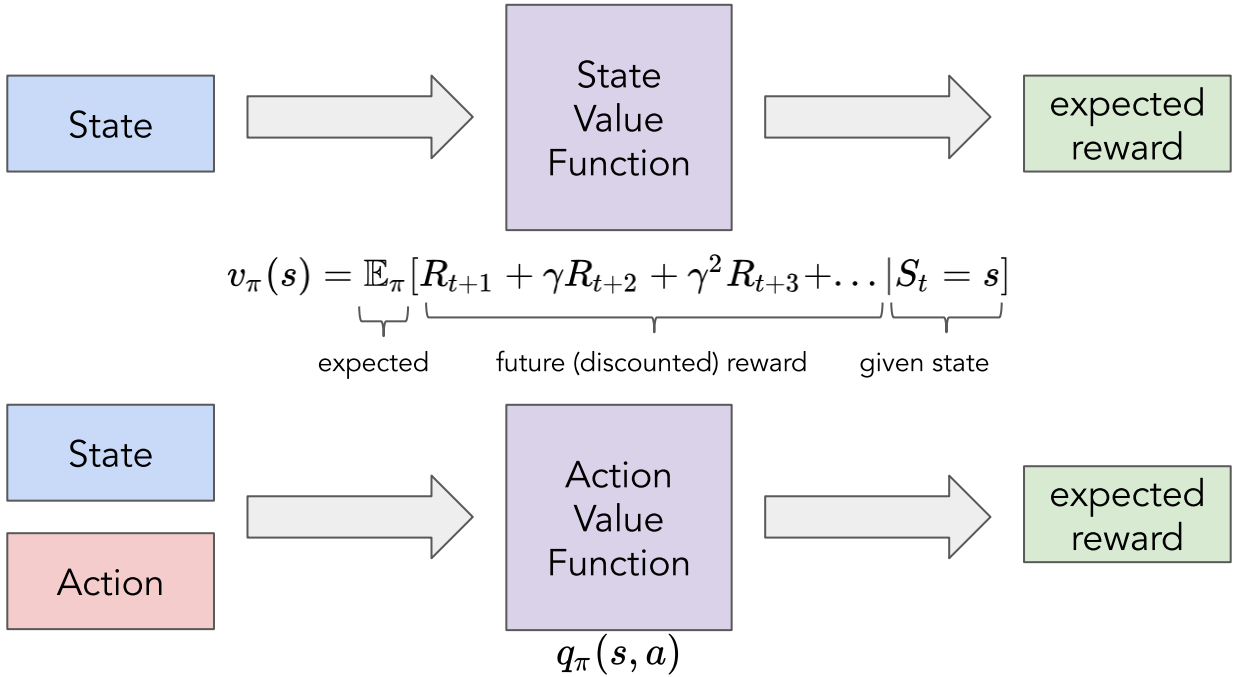
1. A policy is the agent's behaviour
2. It is a map from state to action
3. Deterministic policy: $a = \pi(s)$
4. Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$



$a = \pi(s)$

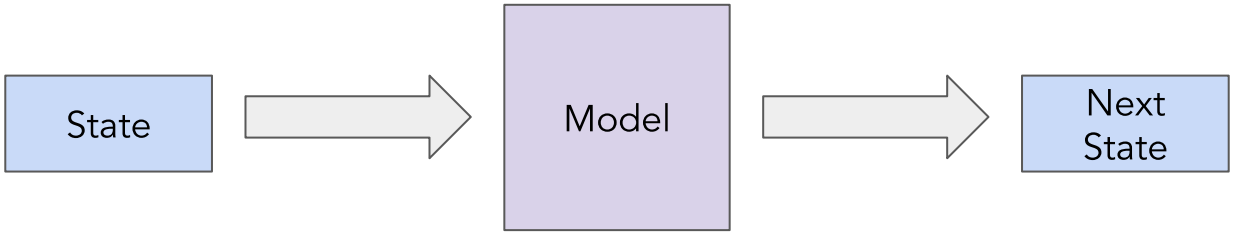
Value Function

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



Model

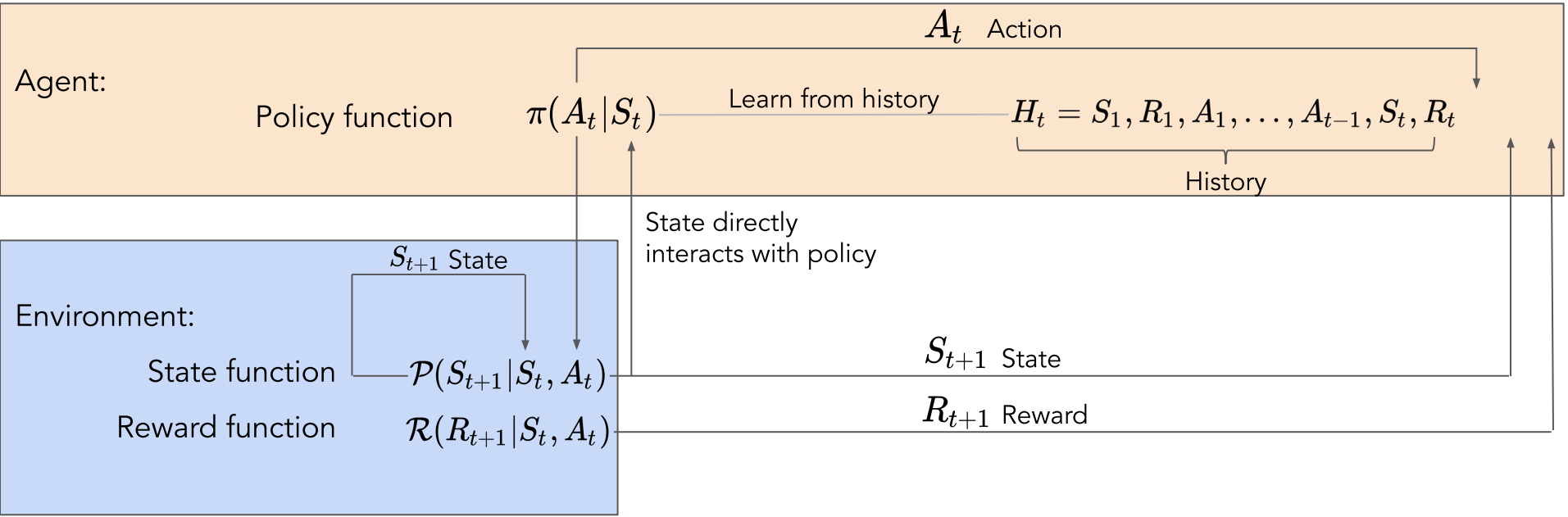
1. A model predicts what the environment will do next
2. P predicts the next state
3. R predicts the next (immediate) reward



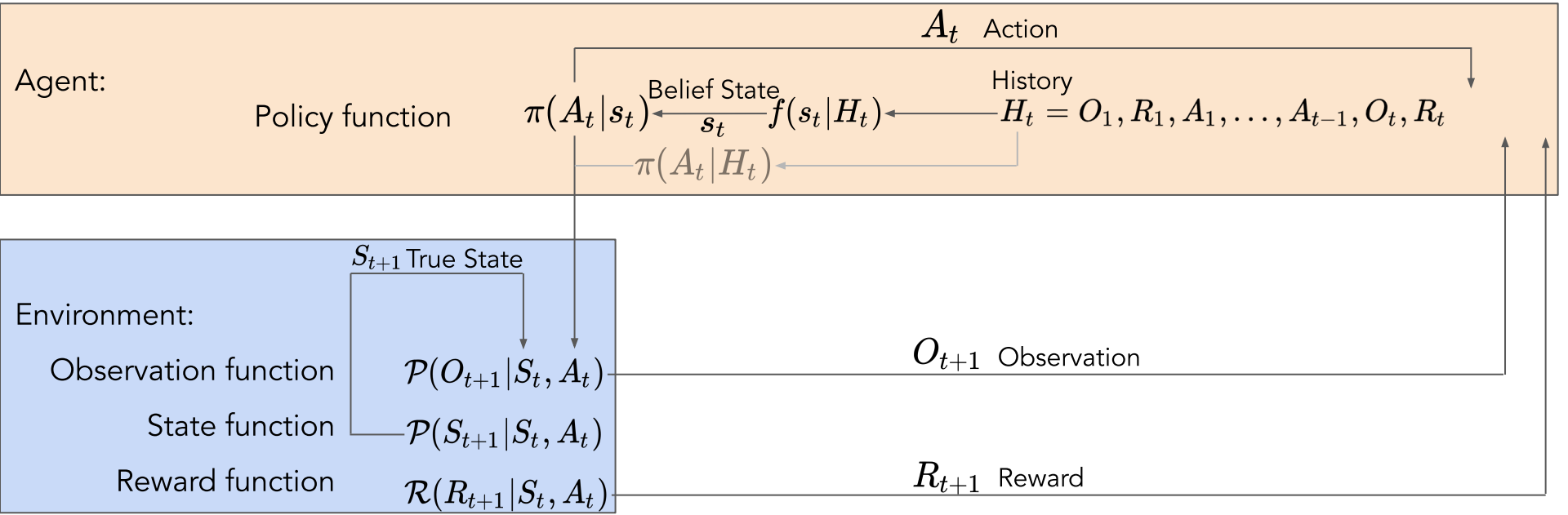
$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} = s' | S_t = s, A_t = a]$$

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
  - $\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$
  - Fully observable environments are MDP
- This way, we can represent each state transition as (s, a, r, s')
  - (state, action, reward, next state)

Value Functions

State-value function: $v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$

- "Expected return, given a current state (for the policy π)"
- Recall that G_t is return: $G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

Action-value function: $q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

- "Expected return, given a current state and action (for the policy π)"

Bellman Equations

Purpose: express value as function of future value: $G_t = f(G_{t+1})$

- An accurate value function will satisfy the Bellman equation
  - Different versions for state-value and action-value functions
- They are used to iteratively make the value function more accurate
  - $G_t := f(G_{t+1})$

State-value Bellman Equation

Derive Bellman equation for state-value function $v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s\right]$$

$$= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=t+2}^{T} \gamma^{k-t-1} R_k \mid S_t = s\right]$$

(expand expectation)

$$= \sum_a \pi(a \mid s) \sum_{s'} \mathcal{P}(s' \mid s, a) \left(\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_\pi\left[\sum_{k=t+2}^{T} \gamma^{k-t-1} R_k \mid S_{t+1} = s'\right]\right)$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \mathcal{P}(s' \mid s, a) \left(\mathcal{R}^a_{ss'} + \gamma v_\pi(s')\right)$$

Action-value Bellman Equation

Derive Bellman equation for action-value function $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \mid S_t = s, A_t = a\right]$$

$$= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=t+2}^{T} \gamma^{k-t-1} R_k \mid S_t = s, A_t = a\right]$$

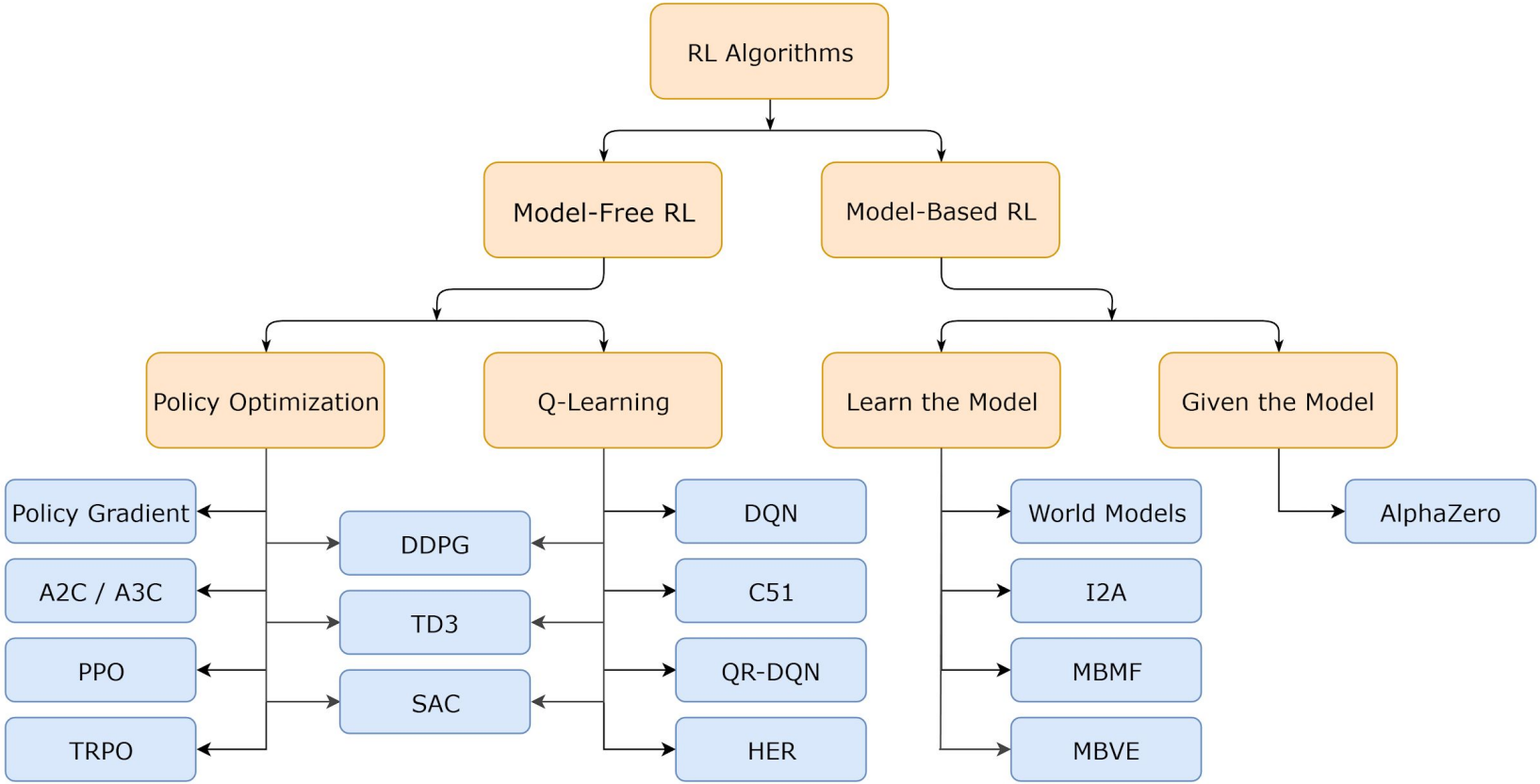
$$= \sum_{s'} \mathcal{P}(s' \mid s,a)\left(\mathcal{R}^a_{ss'} + \gamma \sum_{a'} \pi(s' \mid a') \mathbb{E}_\pi\left[\sum_{k=t+2}^{T} \gamma^{k-t-1} R_k \mid S_{t+1} = s', A_{t+1} = a'\right]\right)$$

(fully expand expectation)

$$q_\pi(s,a) = \sum_{r,s',a'} p(s', r \mid s, a)\pi(a' \mid s')[r + \gamma q_\pi(s', a')], \text{ for all } s \in \mathcal{S}$$

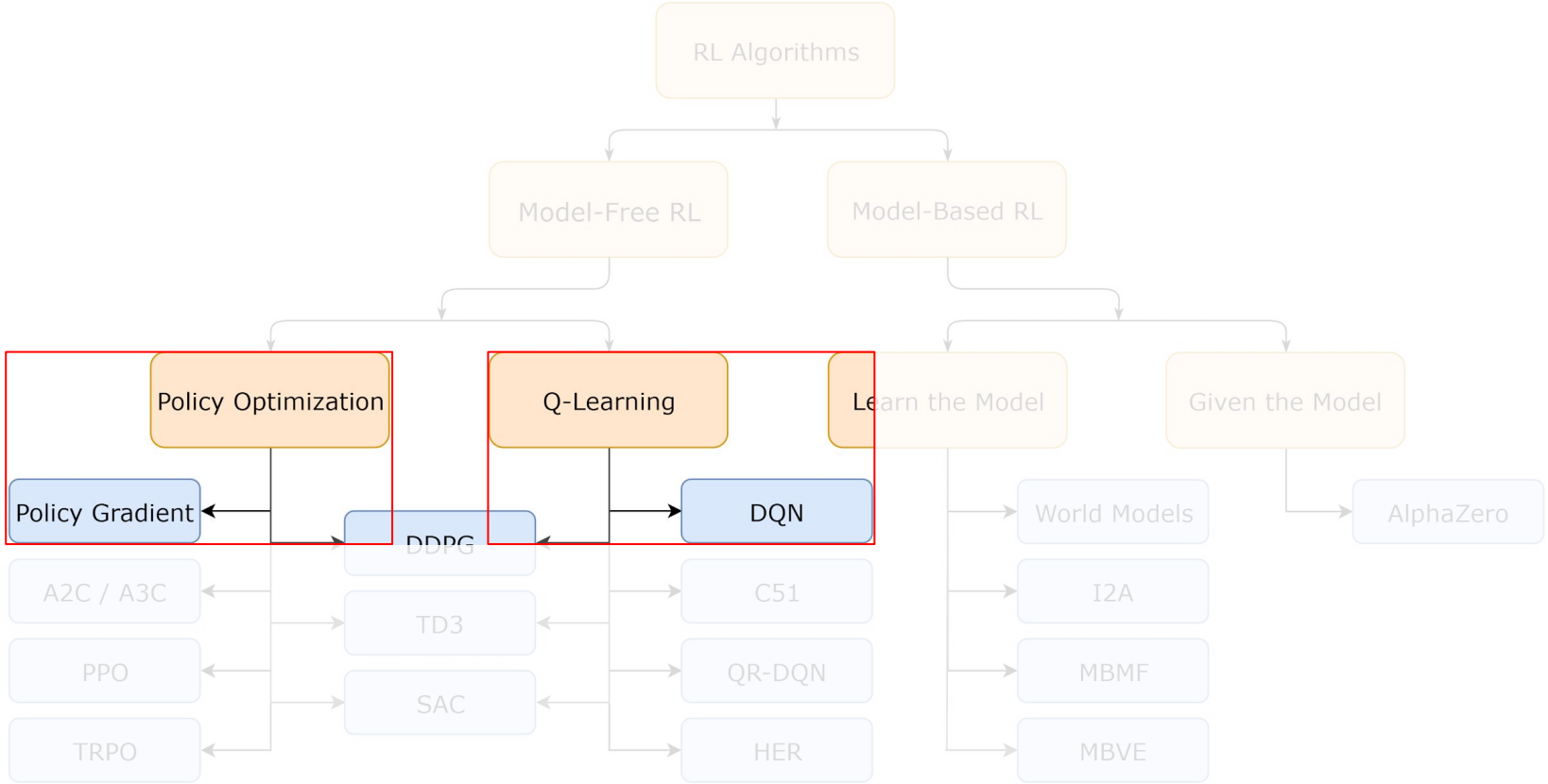
Deep RL Algorithms

Taxonomy of Deep RL algorithms



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

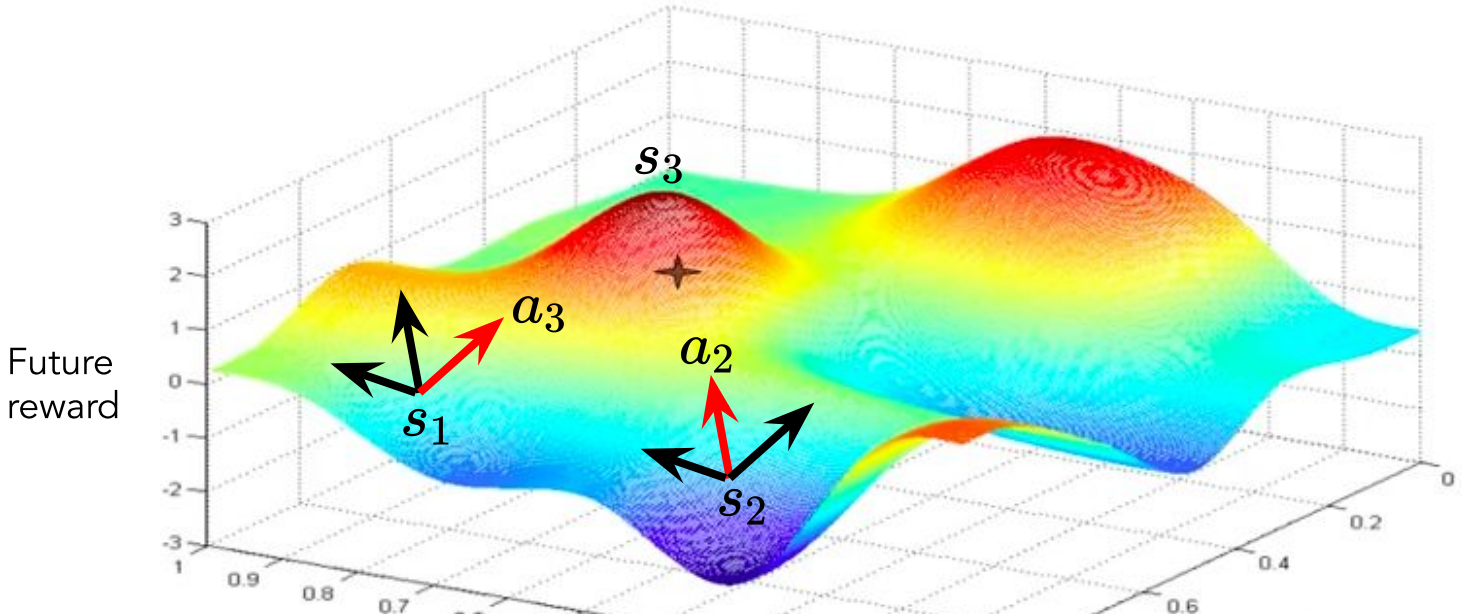
Taxonomy of Deep RL algorithms



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

Q-Learning in a nutshell

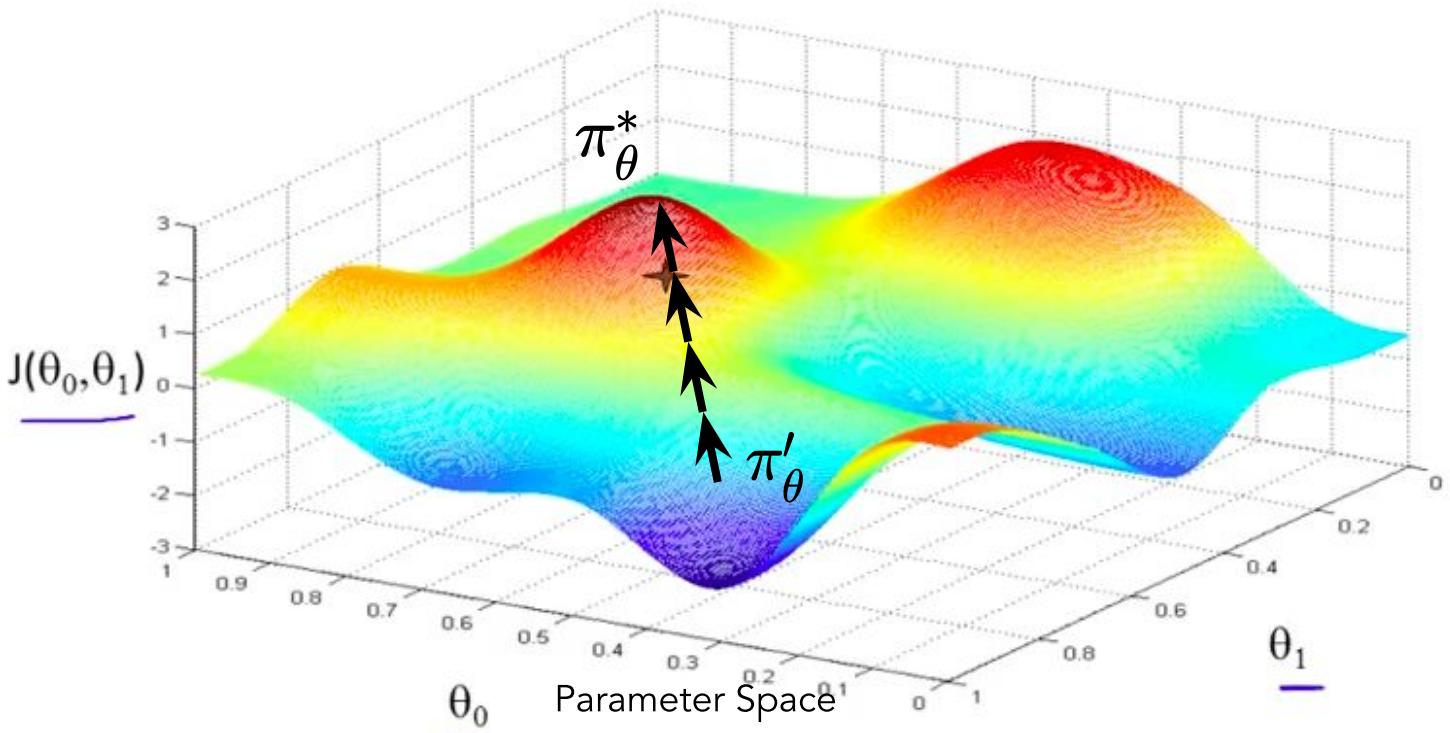
Learn which actions lead to the highest reward, and select the action with the maximum value.



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.

Q-Learning

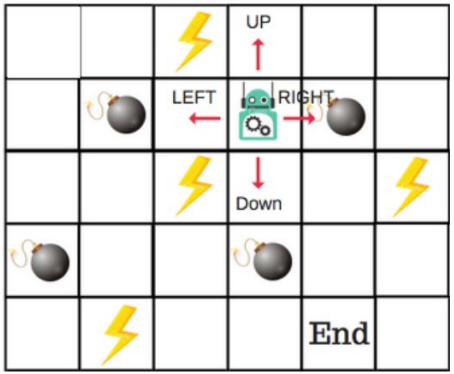
DQN

What is Q Learning?

1. Value Based and Model Free RL Method
    a. Q Learning attempts to estimate total rewards for every given state/action pair within the environment
    b. Q Learning does not model its environment (laymen terms, it doesn't guess what a future state will be given some state, action pair)
2. Find the Optimal Policy
    a. We define an optimal policy to be some parameterized policy which maximizes our expected reward across all time steps
    b. In essence we're optimizing our parameters Θ such that our policy chooses actions which will give us the highest total reward at the end of an episode $\max_{\theta}(E[\sum R_t])$
3. Iterative Process
    a. Q Learning does all of this through an iterative training process

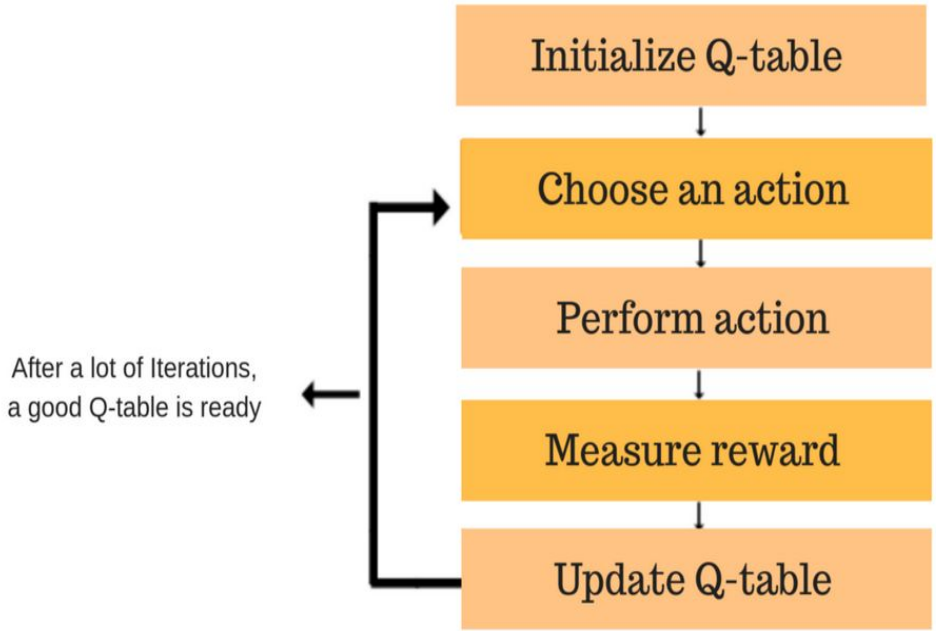
Example Environment

- The reward/scoring system:
  1. The robot loses 1 point at every time step. This is done to incentivize getting to the goal as fast as possible
  2. If the robot steps on a mine, it blows up losing 100 points and the episode is terminated
  3. The robot gets 1 point for every unique power tile it visits
  4. If the robot reaches the end it gets 100 points
  5. Robot terminates after 50 time steps



Training Cycle

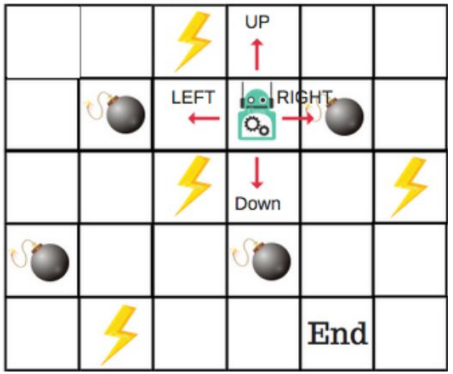
- Note that Q-table is synonymous with Q-function



Simple Q-table Example

- Our Q-table captures what we expect the future cumulative reward to be for any action at any state/tile on the board. For this simple example, the Q-table can be shown as an empty 2-D matrix which is initialized with only zeros.
- A cell in the Q-table is commonly referred to as a Q-value

Actions :	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0



Q-table functionality

For this example, rows in the Q table represent various states while columns represent possible actions that can be taken at a given state. If you input a state action pair into the Q-table you'll receive a reward expectation (aka Q-value). This Q-value is modelled after bellman's equation

$$Q(S_t, A_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t, A_t]$$

Q-table Policy

Recall that policy is a function that takes a state and outputs an action. For our 2D Q-table, a typical deterministic policy would just be:

$$\pi_\theta(S_t) = max(QTable[S_t])$$

Where in the case of tiebreakers max chooses at random

Update Rule Using Bellman's Equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

$A_{t+1} = \pi_\theta(S_{t+1})$

$\gamma = discount\_rate$

$S_{t+1} = next\_state$

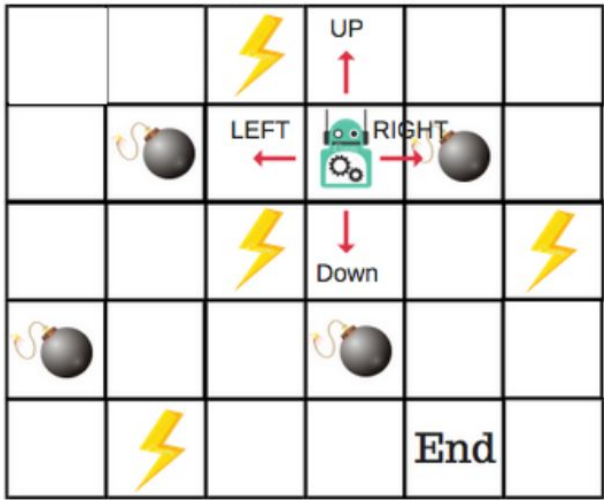
$\alpha = learning\_rate$

$Q(S_t, A_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t, A_t]$

Example Update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

Let's assume Q-table is empty with learning rate of 0.01 with discount of 1.



Assume Policy chooses to go right

- Q(Nothing, right) = 0 + 0.01 * (-100 + 1*0 - 0) = -1
- Episode ends

Instead let's assume policy goes left

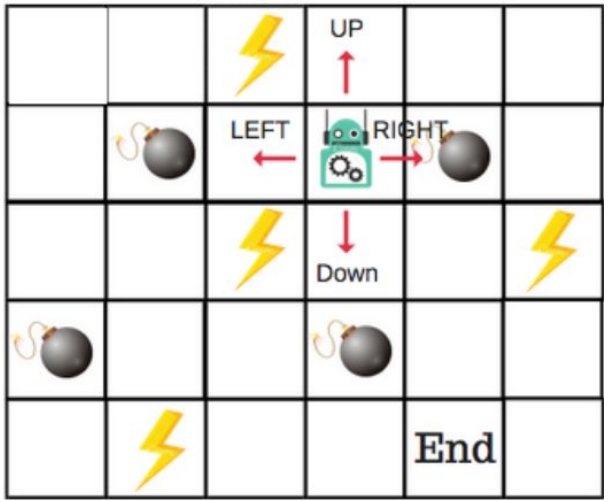
- Q(Nothing, left) = 0 + 0.01 * (-1 + 1*0 - 0) = -0.01
- Episode continues
- Note that always going left on a nothing tile is pretty bad, so a better state space might be 30 states for each tile

Problems

Thus far we have a problem

- We would repeat the same actions always given some state
- For the maze, this may lead us not picking the shortest path or we may never even find the goal

We need a stochastic policy as opposed to a deterministic one



Exploration vs Exploitation

Exploitation is the essence of our deterministic policy. We always choose the best action at our disposal. This however leads to repetitive episodes and learning may come to a complete halt sooner. This is why we need **exploration**!

We understand **exploration** in RL as the process of stochastically (with some probability or randomly) choosing actions in order to find a more optimal overall action which we wouldn't find through exploitation.

Epsilon Greedy Strategy

When training we will define a probability, epsilon which defines the probability of taking a random action

$$\epsilon = 0.1$$

Once again, you need randomness while training to discover potentially more optimal states.

Deep Q-learning

What about more complex state/action spaces? For example,

1. What if we have an image as a state space. How do we map an image (3D vector of pixels) onto a 2D vector (such as a Q-table)?
2. What if we have an actual robot moving around with continuous action spaces?

Deep Q-learning

What about more complex state/action spaces? For example,

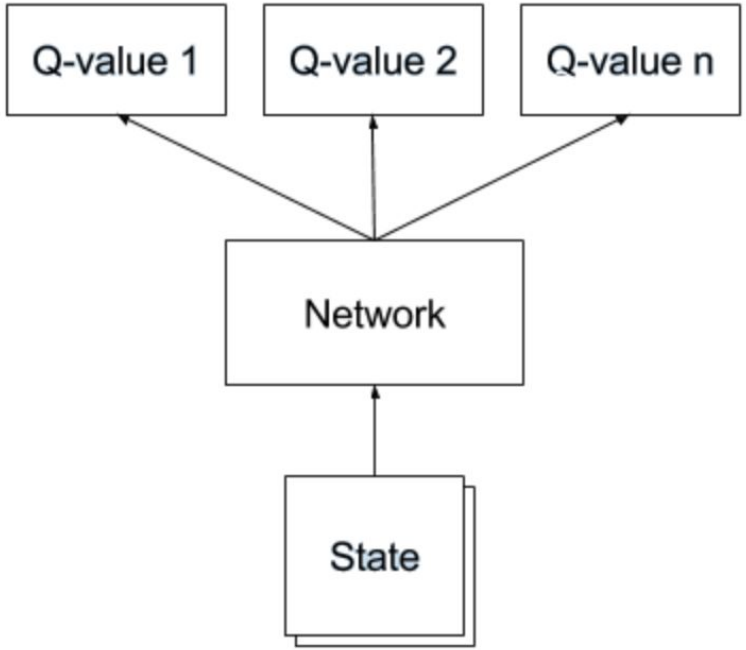
1. What if we have an image as a state space. How do we map an image (3D vector of pixels) onto a 2D vector (such as a Q-table)?
2. What if we have an actual robot moving around with continuous action spaces?

To solve these problems, we use neural networks to approximate our Q-table!

How does this work?

Neural Networks as you all know are great for making approximations!

Instead of receiving a state action pair and returning one q-value, now we receive just the state and it attempts to estimate a q-value for every action within that state



Ok… but how do we train this?

The training process would be similar to training normal neural networks.

$$l2\_loss = (predicted - actual)^2$$

Predicted would simply be the largest q-value our neural network spits out at a given time step

So what is actual?

Calculating Actual

Because this is RL, we don't have a definitive value for the actual expected cumulative reward at any given time step

To solve this, let's approximate actual

$$actual = R(S_t, A_t) + \gamma max_{A_t+1} Q(S_{t+1}, A_{t+1})$$

$$l2\_loss = (predicted - actual)^2$$

Calculating Actual

Notice that we use our neural network to help approximate actual. Even if our neural network is completely wrong, there is some validity to the expression due to a correct reward therefore the network will also slowly update its gradients to produce a valid actual q-value!

$$actual = R(S_t, A_t) + \gamma max_{A_t+1} Q(S_{t+1}, A_{t+1})$$

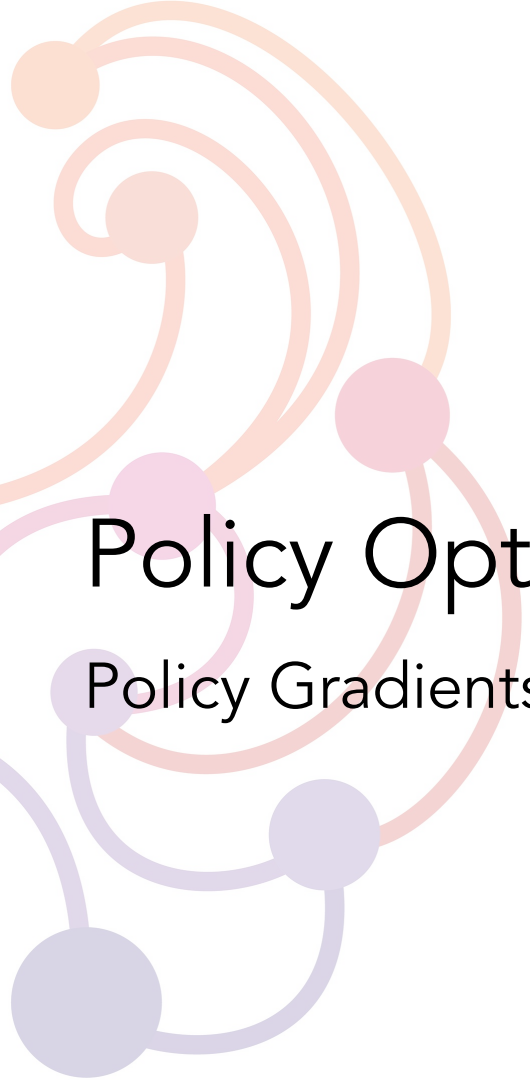
$$l2\_loss = (predicted - actual)^2$$

Wrapping up training

After substituting your approximation for actual and q-value into prediction, you simply perform gradient descent and your deep q network is ready to go!

To recap:

- Calculate Q(St, At) and Q(S(t+1), A(t+1)) indicating next optimal step which we'll use as our actual value. This requires two forward passes
- Perform gradient descent at every time step
- Takes a while to train, but requires less memory than vanilla Q Learning



Policy Optimization

Policy Gradients, Actor Critic

Dataset

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

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

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.

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

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.

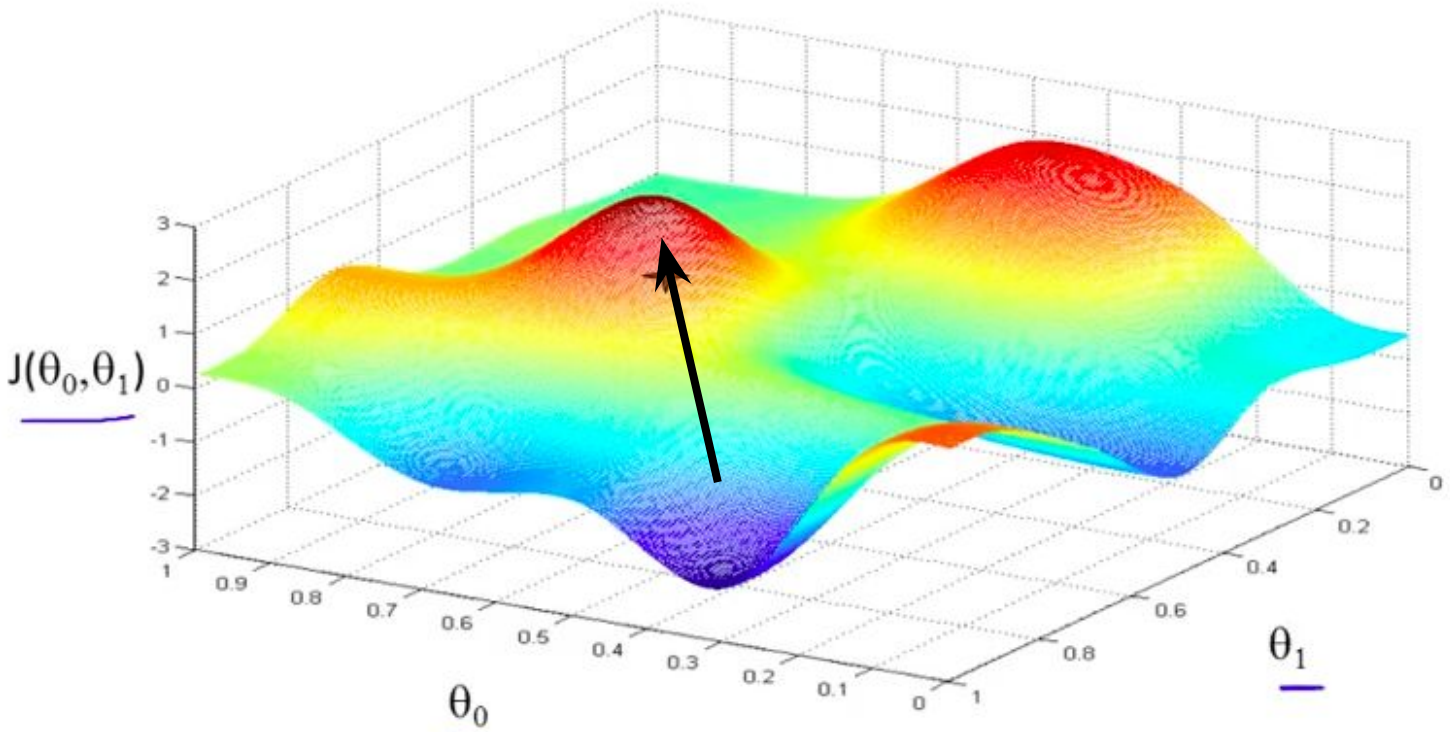
Learning a better policy

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.

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

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.

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

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)

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

- θ_{k+1}: Policy parameters for iteration k+1
- θ_k: Policy parameters for iteration k
- α: Learning rate
- $\nabla_\theta J(\pi_\theta)|_{\theta_k}$: 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.

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

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

Calculating the gradient (2)

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

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.

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

Calculating the gradient (3)

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

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

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)

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

Bring back the integral into expectation form

$$= \underset{\tau \sim \pi_\theta}{E} \left[\nabla_\theta \log P(\tau|\theta) R(\tau) \right]$$

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{E}\left[\nabla_\theta \log P(\tau|\theta) R(\tau)\right]$$

Let's elaborate on the trajectory

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

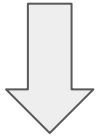
$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \left(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \right)$$

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

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Calculating the gradient (5)

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{E} \left[\nabla_\theta \log P(\tau|\theta) R(\tau) \right]$$



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

Final form of the policy gradient.

Calculating the gradient from data

Final policy gradient:

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

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:

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

Problem with rewards in gradients

1. High variance in rewards
    a. 0 to 10000 rewards
    b. causes problems in backprop
2. Solutions
    a. Advantage function

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **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-jQfW4rUsdcI4lXks

RL: Expectations vs. Reality

https://www.alexirpan.com/2018/02/14/rl-hard.html

1. Deep Reinforcement Learning Can Be Horribly Sample Inefficient
2. If You Just Care About Final Performance, Many Problems are Better Solved by Other Methods
3. Reinforcement Learning Usually Requires a Reward Function
4. Reward Function Design is Difficult
5. Even Given a Good Reward, Local Optima Can Be Hard To Escape
6. Even When Deep RL Works, It May Just Be Overfitting to Weird Patterns In the Environment
7. Even Ignoring Generalization Issues, The Final Results Can be Unstable and Hard to Reproduce
8. Not much industry usage, yet

Research Directions for better RL methods

1. Sample efficient RL
   a. Model-based RL
   b. Hierarchical RL
2. Learning the rewards
   a. Inverse RL
3. Learning from humans
   a. Imitation learning
4. Harder environments
   a. Starcraft II
   b. Real world Robotics
5. Alternatives to RL
   a. Evolutionary Algorithms
6. etc...

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!