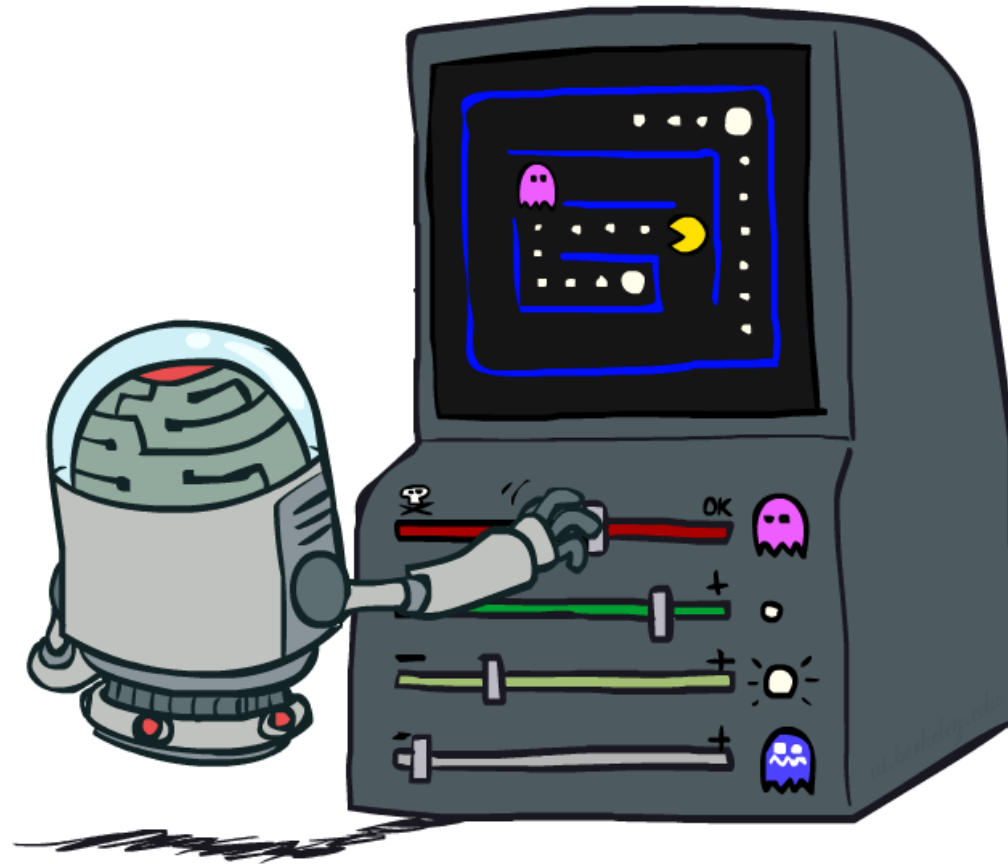


# CS 6300: Artificial Intelligence

## Reinforcement Learning III: Policy Gradients



Instructor: Daniel Brown --- University of Utah

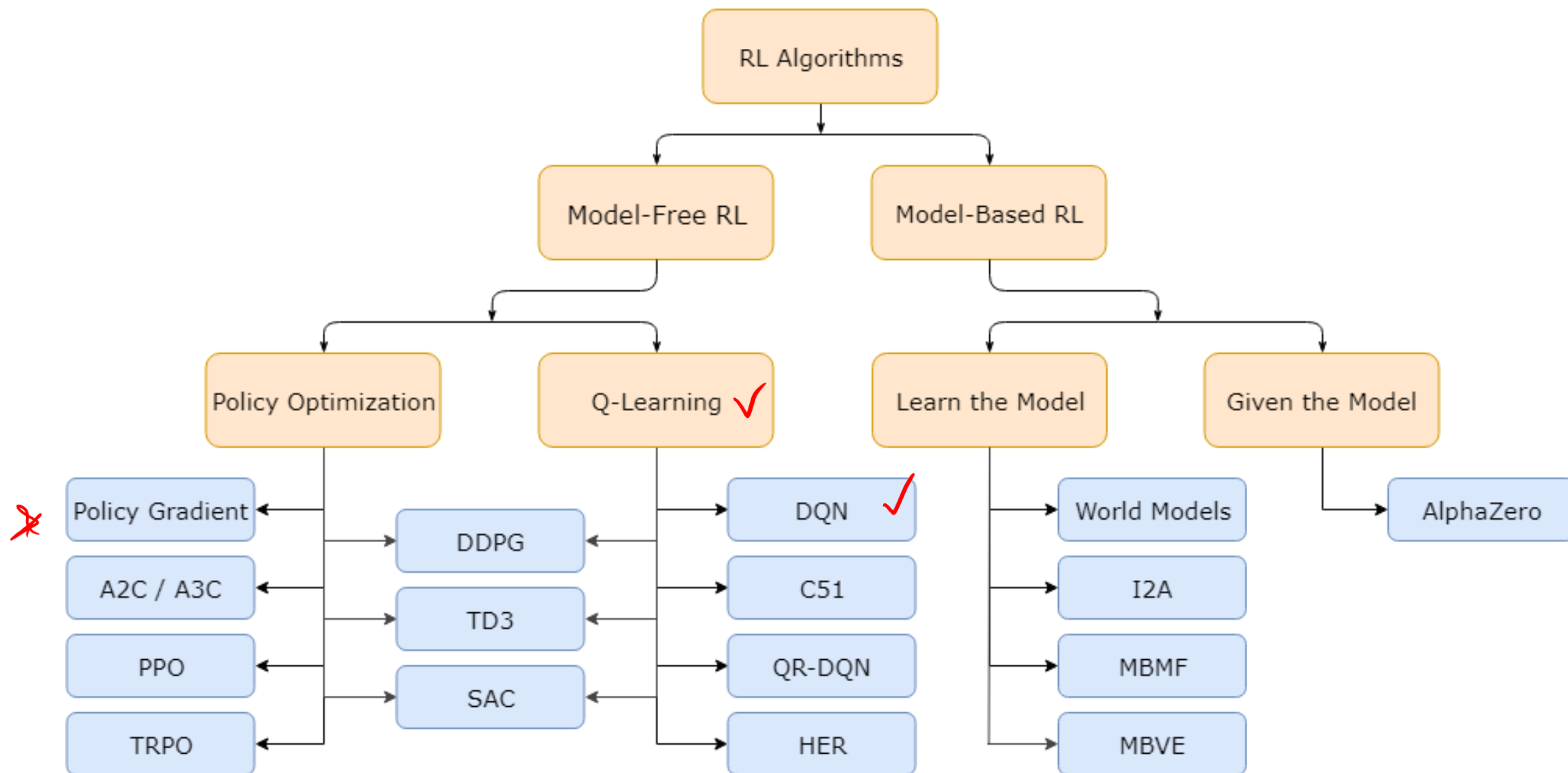
[Based on slides created by Dan Klein and Pieter Abbeel <http://ai.berkeley.edu>.]

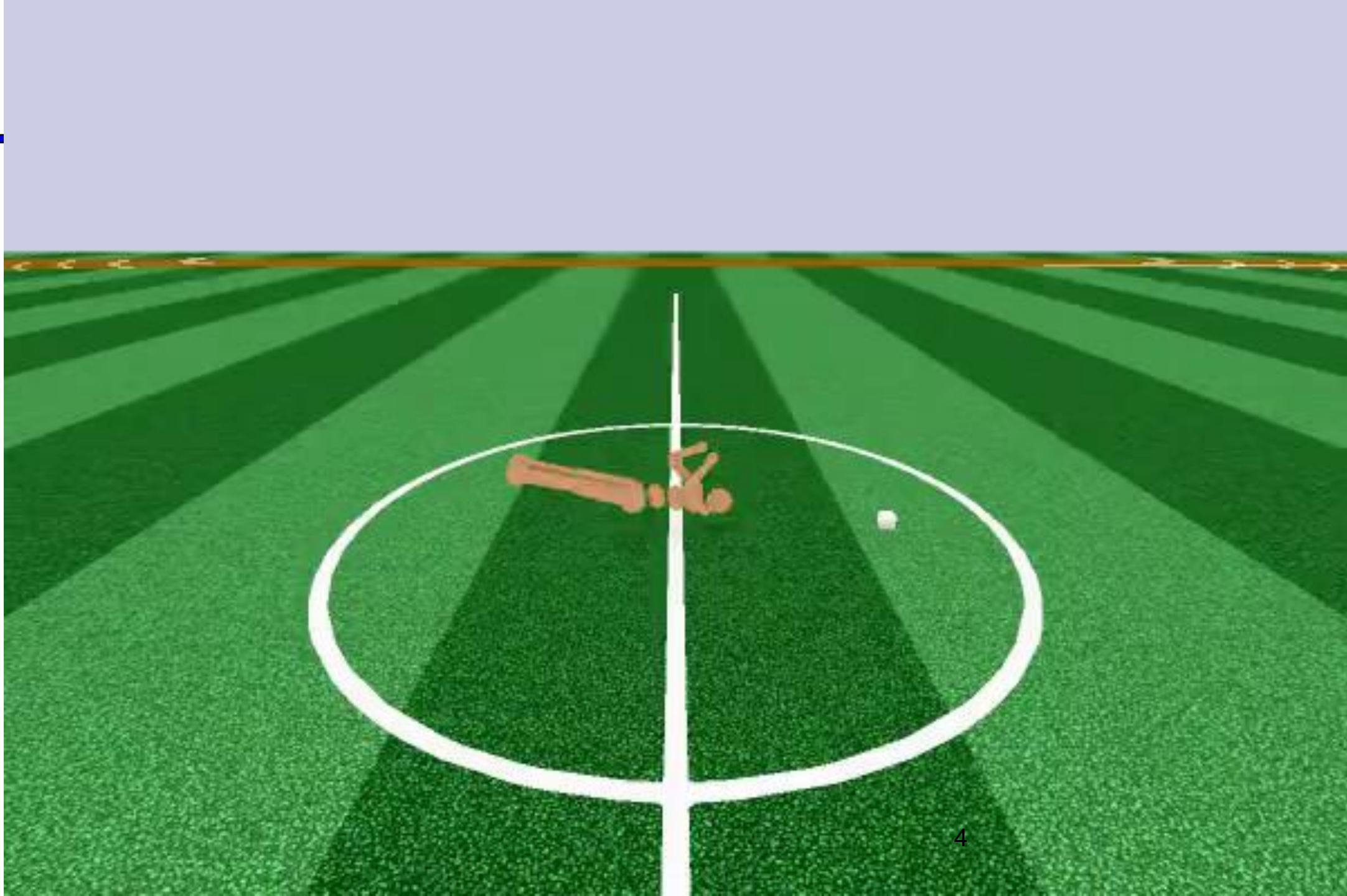
# Announcement

---

- Mid semester feedback is open!
- Help me make the class better!
- If you submit feedback you get extra credit equivalent to 2 class quizzes!
  - All feedback is anonymous and appreciated.
  - Confirm submission of feedback via canvas

# Rough Taxonomy of RL Algorithms







# What is the goal of RL?

---

- Find a policy that maximizes expected utility (discounted cumulative rewards)

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(s, \pi(s), s') \right]$$

# Two approaches to model-free RL

- Learn Q-values

- Trains Q-values to be consistent. Not directly optimizing for performance.
- Use an objective based on the Bellman Equation

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

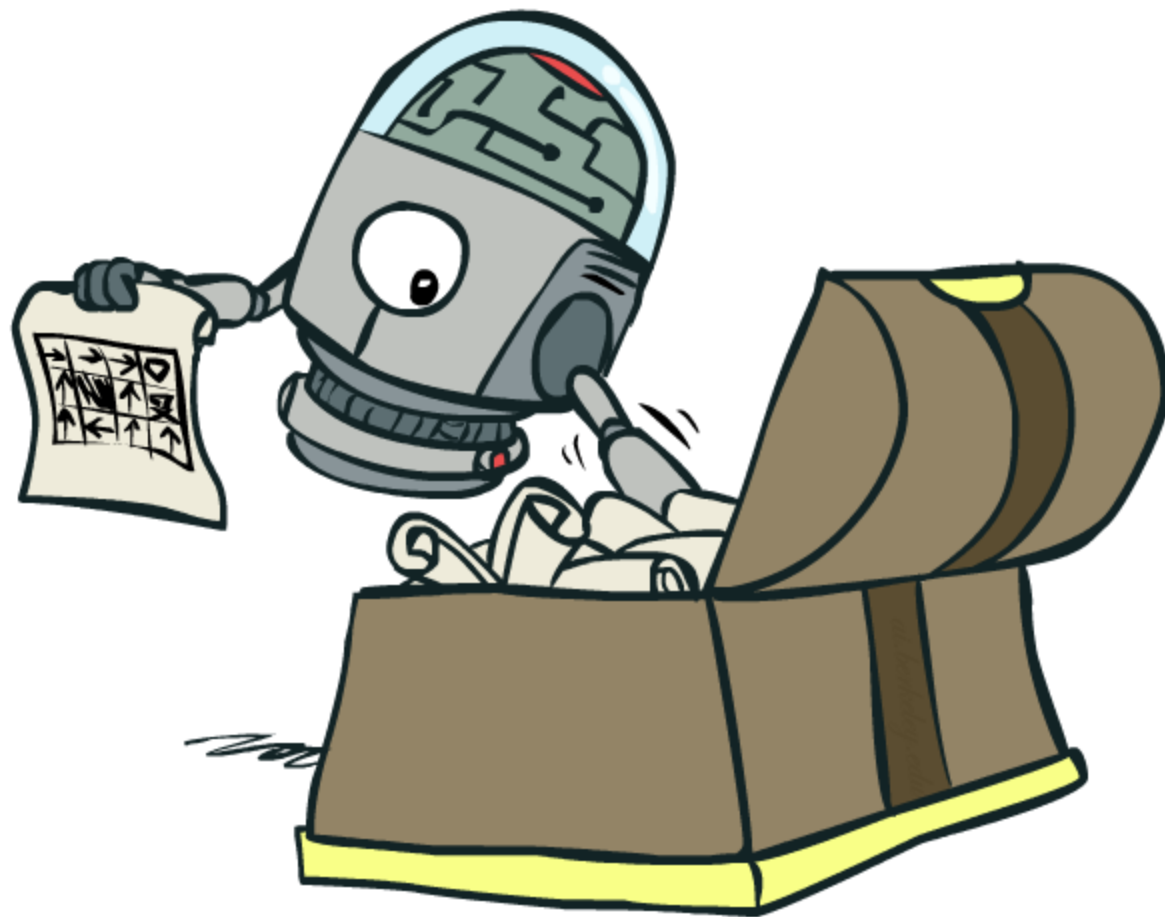
- Learn Policy Directly

- Have a parameterized policy  $\pi_{\theta}$
- Update the parameters  $\theta$  to optimize performance of policy.



# Policy Search

---





# Preliminaries

- Trajectory (rollout, episode)  $\tau = (s_0, a_0, s_1, a_1, \dots)$

- $s_0 \sim \rho_0(\cdot), s_{t+1} \sim P(\cdot | s_t, a_t)$

- Rewards  $r_t = R(s_t, a_t, s_{t+1})$

- Finite-horizon undiscounted return of a trajectory

$$R(\tau) = \sum_{t=0}^T r_t$$

*maximize goal*

- Actions are sampled from a parameterized policy  $\pi_\theta$

$$a_t \sim \pi_\theta(\cdot | s_t)$$

$P(s) P(a_0|s_0) P(s_1|a_0, s_0) P(a_1|s_1) P(s_2|a_1, s_1)$   
**Preliminaries**

Chain Rule of Prob w/ Markov

- Probability of a trajectory (rollout, episode)  $\tau = (s_0, a_0, s_1, a_1, \dots)$

$$P(\tau|\pi) = \underbrace{\rho_0(s_0)}_{\text{initial dist}} \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_{\theta}(a_t|s_t)$$

Assumption

- Expected Return of a policy  $J(\pi)$

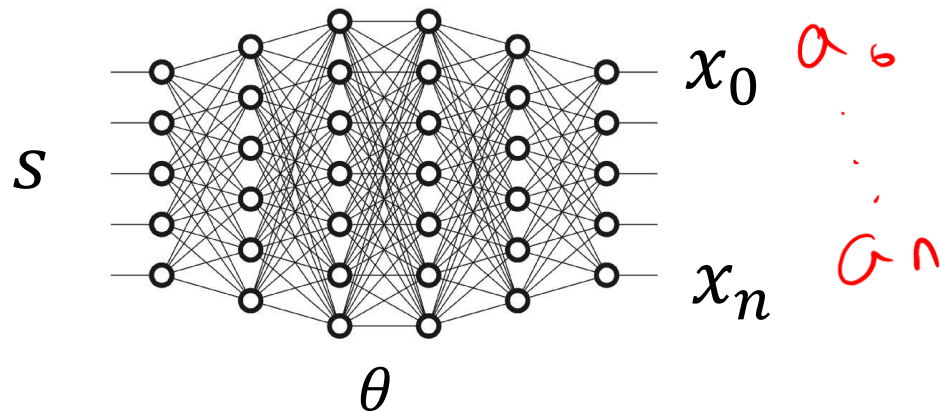
$$J(\pi) = \sum_{\tau} P(\tau|\pi) R(\tau) = E_{\tau \sim \pi} [R(\tau)]$$

- Goal of RL: Solve the following optimization problem

$$\pi^* = \operatorname{argmax}_{\pi} J(\pi)$$

# How should we parameterize our policy?

- We need to be able to do two things:
  - Sample actions  $a_t \sim \pi_\theta(\cdot | s_t)$
  - Compute log probabilities  $\log \pi_\theta(a_t | s_t)$
- Categorical (classifier over discrete actions)
  - Typically, you output a value  $x_i$  for each action (class) and then the probability is given by a softmax equation



$$\pi_\theta(a_i | s) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

# How should we parameterize our policy?

- Diagonal Gaussian (distribution over continuous actions)

$$a \sim N(\underline{\mu}, \underline{\Sigma})$$

where  $\Sigma$  has non-zero elements only on the diagonal.

Thus, an action can be sampled as

$$a = \mu_{\theta}(s) + \sigma_{\phi}(s) \odot z,$$

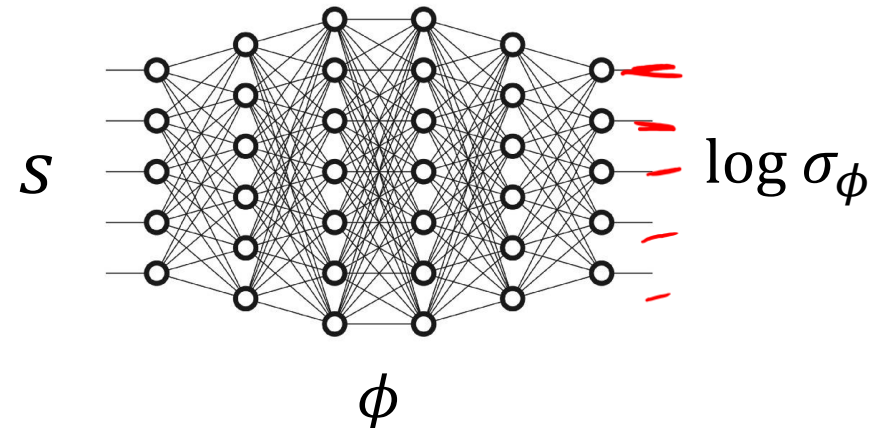
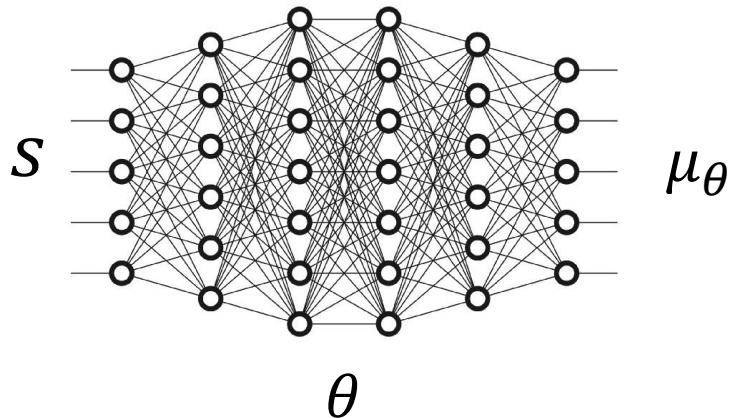
*element wise mult*

$$z \sim N(0, I)$$

$$\sigma_{\phi} = \exp(\log \sigma_{\phi})$$



$$\vec{x} \odot \vec{y} = \begin{bmatrix} x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_n y_n \end{bmatrix}$$



# Goal: Update Policy via Gradient Ascent

- We have a parameterized policy and we want to update it so that it maximizes the expected return.
- We want to find the gradient of the return with respect to the policy parameters and step in that direction.

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

Policy gradient

# Fact #1

- Probability of a trajectory:

- The probability of a trajectory  $\tau = (s_0, a_0, \dots, s_{T+1})$  given that actions come from  $\pi_\theta$  is

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

# Fact #2

$$\log(A \cdot B) = \log A + \log B$$

- Log-probability of a trajectory:

- The log-probability of a trajectory  $\tau = (s_0, a_0, \dots, s_{T+1})$  given that actions come from  $\pi_\theta$  is

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



# Fact #3

- Grad-Log-Prob of a Trajectory

- Note that gradients of everything that doesn't depend on  $\theta$  is 0.

$$\nabla_{\theta} \log P(\tau|\theta) = \cancel{\nabla_{\theta} \log p_0(s_0)} + \sum_{t=0}^T (\cancel{\nabla_{\theta} \log P(s_{t+1}|s_t, a_t)}) + \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)$$

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

# Fact #4

- Log-Derivative Trick:

- This is based on the rule from calculus that the derivative of  $\log x$  is  $1/x$

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

$$\overset{g(x)}{\frac{d}{dx}} \log g(x) = \frac{1}{\cancel{g(x)}} \frac{d}{dx} \cancel{g(x)} \Rightarrow \boxed{g(x) \frac{d}{dx} \log g(x) = \frac{d}{dx} g(x)}$$

$\pi_\theta \triangleq \theta$ 

# Derivation of Policy Gradient

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta E_{\tau \sim \pi_\theta} [R(\tau)] \\ &= \nabla_\theta \sum_\tau P(\tau|\theta) R(\tau) \\ &= \sum_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\ &= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \\ &= E_{\tau \sim \pi_\theta} [\nabla_\theta \log P(\tau|\theta) R(\tau)] \\ &= E_{\tau \sim \pi_\theta} [\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]\end{aligned}$$

$\sum_{t=0}^T r_t$   
def Expectation

Fact #4

Fact #3

# The Policy Gradient (REINFORCE)

- We can now perform gradient ascent to improve our policy!

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

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

Estimate with a sample mean over a set  $D$  of policy rollouts given current parameters

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

# How would you implement this?

1. Start with random policy parameters  $\theta_0$
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

$$a_t \sim \pi_\theta(\cdot | s_t) \Rightarrow (s_0, a_0, r_0, s_1, a_1, r_1, \dots, r_T, s_{T+1})$$

$$D = \{\tau_1, \dots, \tau_N\}, \quad R = \{R(\tau_1), \dots, R(\tau_N)\}$$

3. Compute policy gradient

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

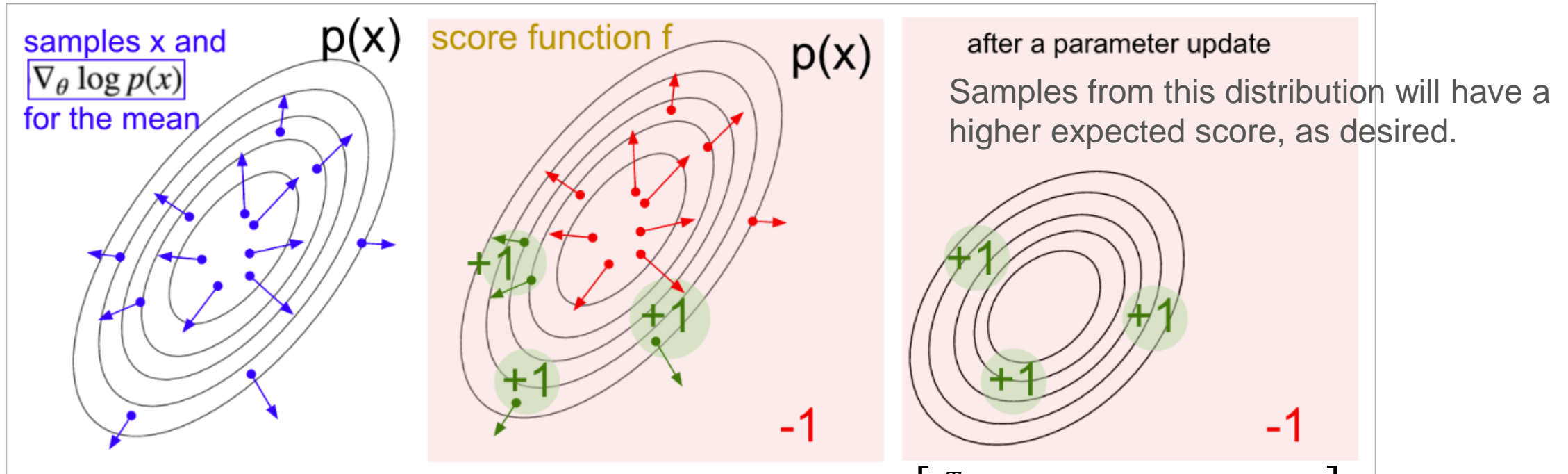
4. Update policy parameters

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

5. Repeat from step 2

# Some more intuition (thanks to Andrej Karpathy)

- Blue Dots: samples from Gaussian
- Blue arrows: gradients of the log probability with respect to the gaussian's mean parameter
- We score each sample
- Red have score -1
- Green have scores +1
- To update the Gaussian mean parameter, we average up all the green arrows, and the *negative* of the red arrows.



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

# Policy Gradient RL Algorithms

---

- We can directly update the policy to achieve high reward.
- Pros:
  - Directly optimize what we care about: Utility!
  - Naturally handles continuous action spaces!
  - Can learn specific probabilities for taking actions.
  - Often more stable than value-based methods (e.g. DQN).
- Cons:
  - On-Policy -> Sample-inefficient we need to collect a large set of new trajectories every time the policy parameters change.
  - Q-Learning methods are usually more data efficient since they can reuse data from any policy (Off-Policy) and can update per sample.



# Many forms of policy gradients

$(s_0, a_0, r_0, s_1, a_1, r_1, s_2)$

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

*weight*

What we derived:  $\Phi_t = R(\tau),$

Follows a similar derivation:

$$\Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}),$$

<https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63>

- What is better about the second *Reward to go* approach?
  - Focuses on rewards in the future!
  - Less variance -> less noisy gradients.

# Many forms of policy gradients

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

$$\Phi_t = \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}),$$

Looks familiar....

$$\Phi_t = Q^{\pi_{\theta}}(s_t, a_t)$$

- Now we have an approach that combines a parameterized policy and a parameterized value function!

I rotate  
the piece



Really bad  
action



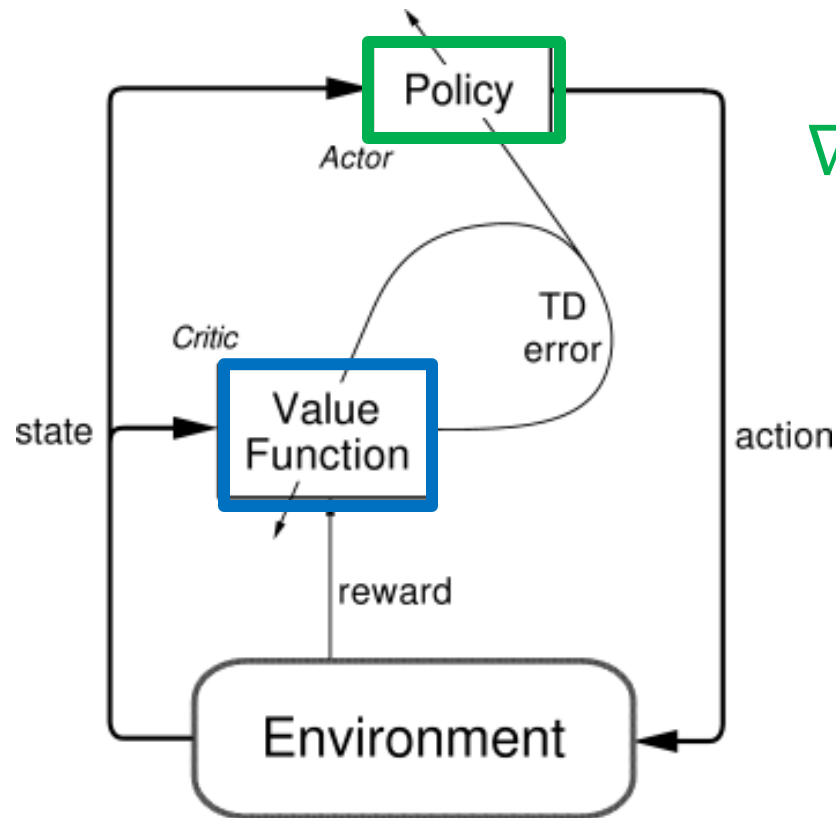
Actor



Critic

# Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



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

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

$$\delta = (r_t + \gamma Q_w^{\pi_{\theta}}(s_{t+1}, a_{t+1}) - Q_w^{\pi_{\theta}}(s_t, a_t))$$

$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_{\theta} Q_w^{\pi_{\theta}}$$

# Q Actor Critic Algorithm Pseudo Code

---

**Algorithm 1** Q Actor Critic

---

Initialize parameters  $s, \theta, w$  and learning rates  $\alpha_\theta, \alpha_w$ ; sample  $a \sim \pi_\theta(a|s)$ .

**for**  $t = 1 \dots T$ : **do**

    Sample reward  $r_t \sim R(s, a)$  and next state  $s' \sim P(s'|s, a)$

    Then sample the next action  $a' \sim \pi_\theta(a'|s')$

    Update the policy parameters:  $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$ ; Compute the correction (TD error) for action-value at time t:

$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

    and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$

    Move to  $a \leftarrow a'$  and  $s \leftarrow s'$

**end for**

---

Adapted from Lilian Weng's post "Policy Gradient algorithms"

# Many forms of policy gradients

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \Phi_t \right]$$

$$\Phi_t = R(\tau), \quad \Phi_t = \sum_{t'=t}^{T-1} R(s_{t'}, a_{t'}, s_{t'+1}), \quad \Phi_t = Q^{\pi_{\theta}}(s_t, a_t)$$

$$\Phi_t = \sum_{t'=t}^{T-1} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

$$\Phi_t = A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

Advantage Function

# The Advantage Function

$$A(s, a) = \underline{Q(s, a)} - \underline{V(s)}$$

q value for action a  
in state s

average  
value  
of that  
state

- Why good?
- Why bad?



# The Advantage Function

---

$$A(s, a) = \boxed{Q(s, a)} - V(s)$$

|

$$r + \gamma V(s')$$

|

$$A(s, a) = \underline{r + \gamma V(s')} - V(s)$$

TD Error

# Advantage Actor Critic (A2C)

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function

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

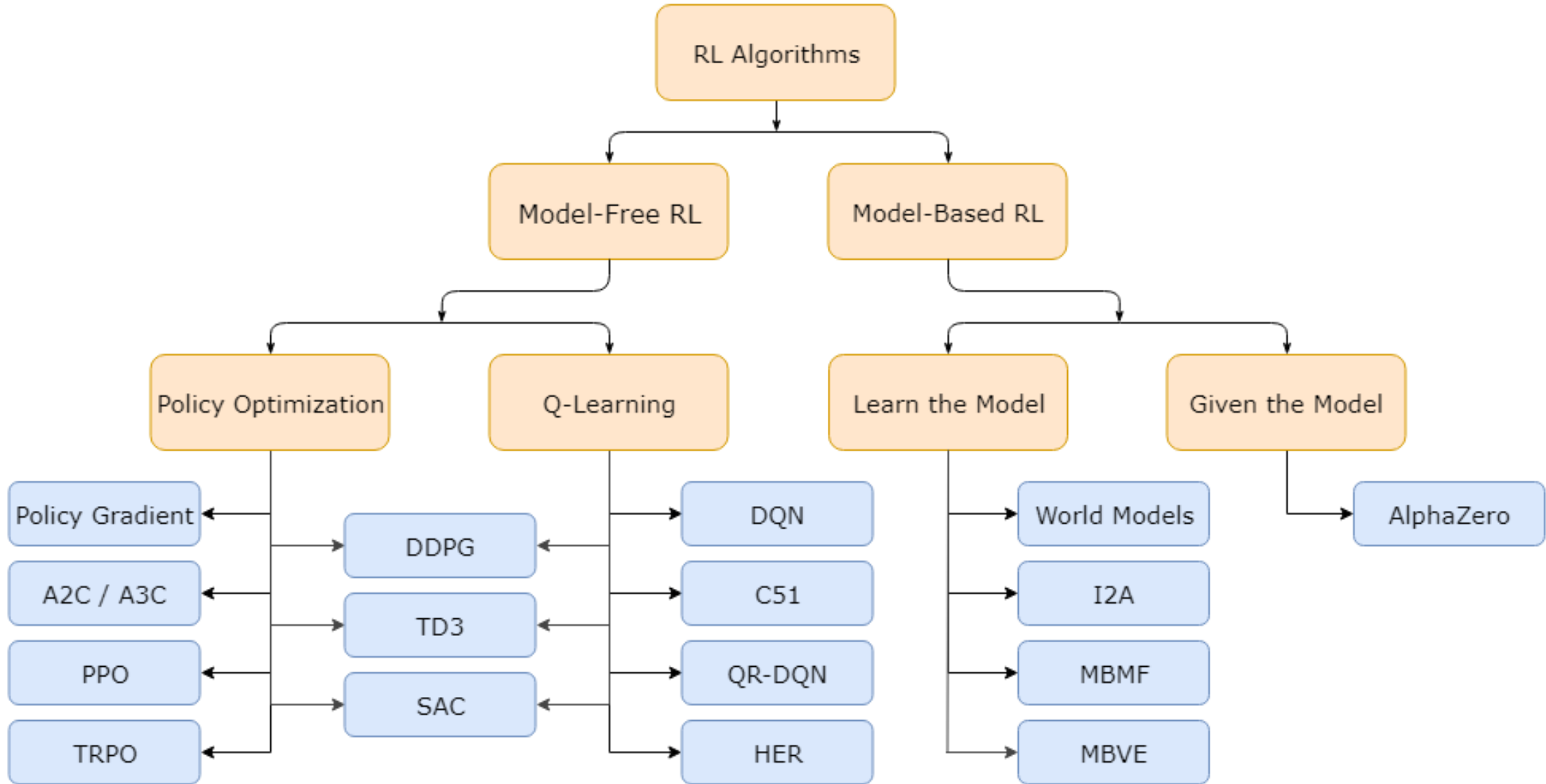
$$\Phi_t = A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

$$\text{TD error } \delta_t = r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

TD-Learning update

$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_w V(s, a; w)$$

# Rough Taxonomy of RL Algorithms



- 
- Next time: Alpha Go