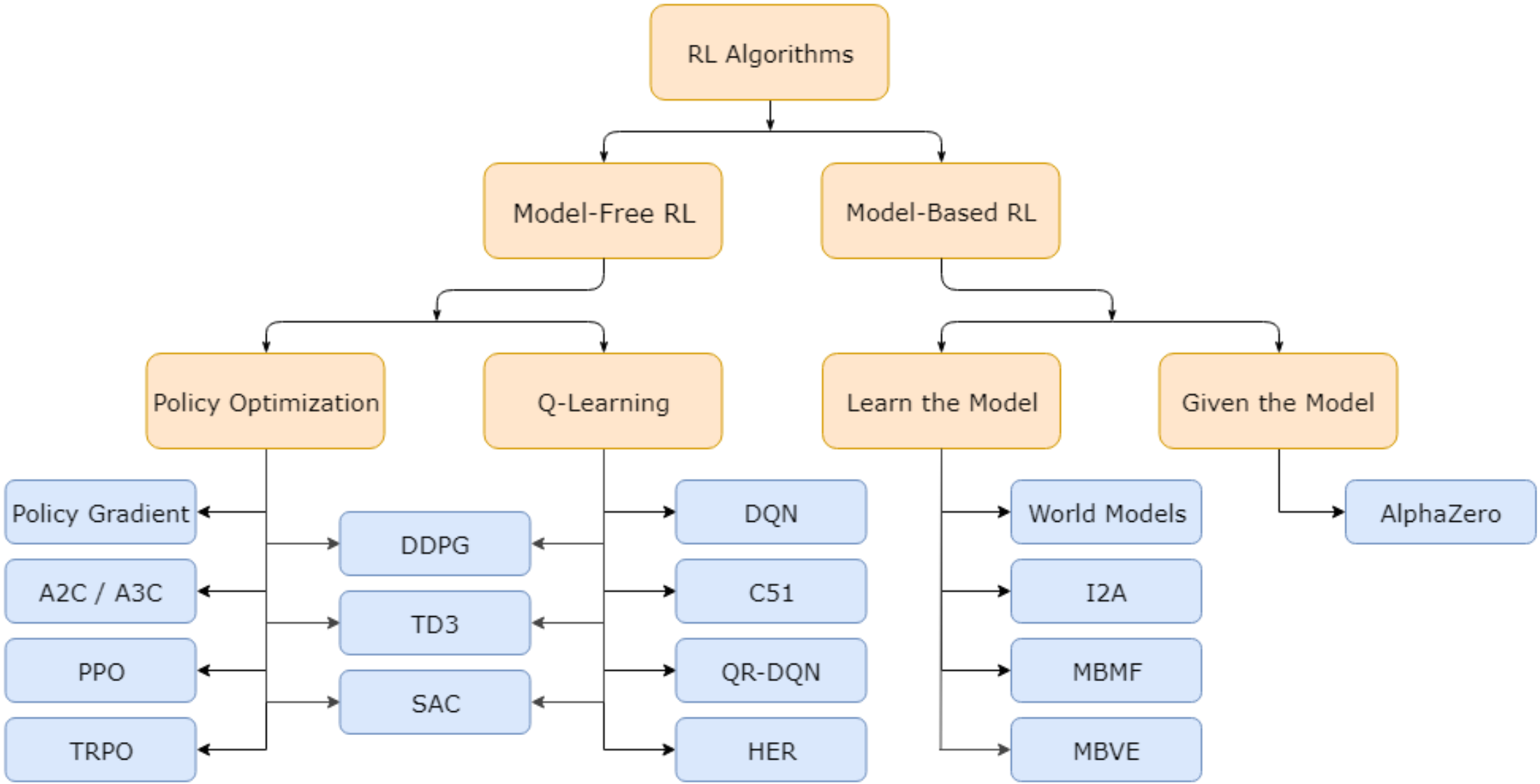
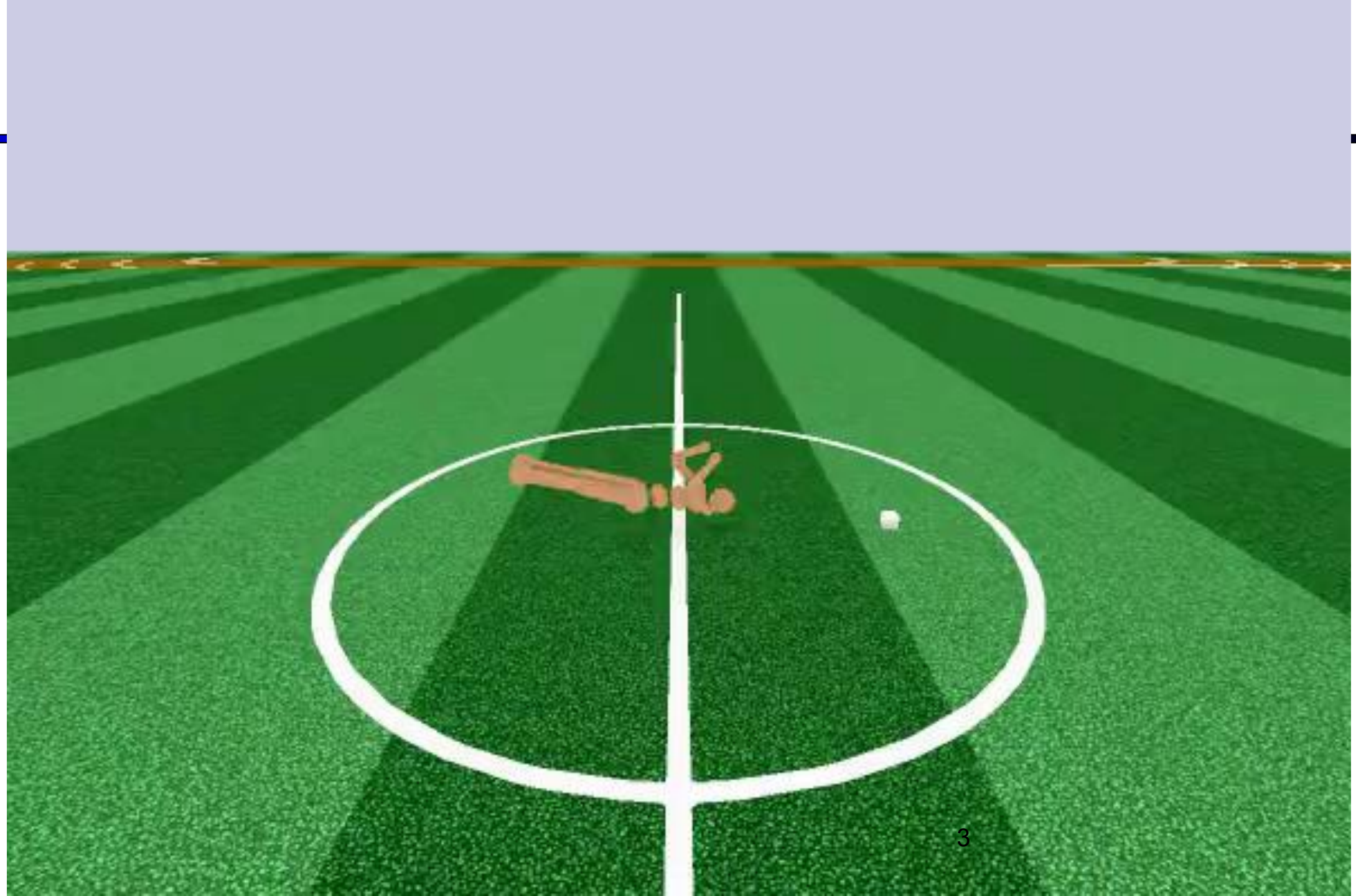
Policy Gradients

Daniel Brown --- University of Utah

Rough Taxonomy of RL Algorithms






O
A

What is the goal of RL?

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

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

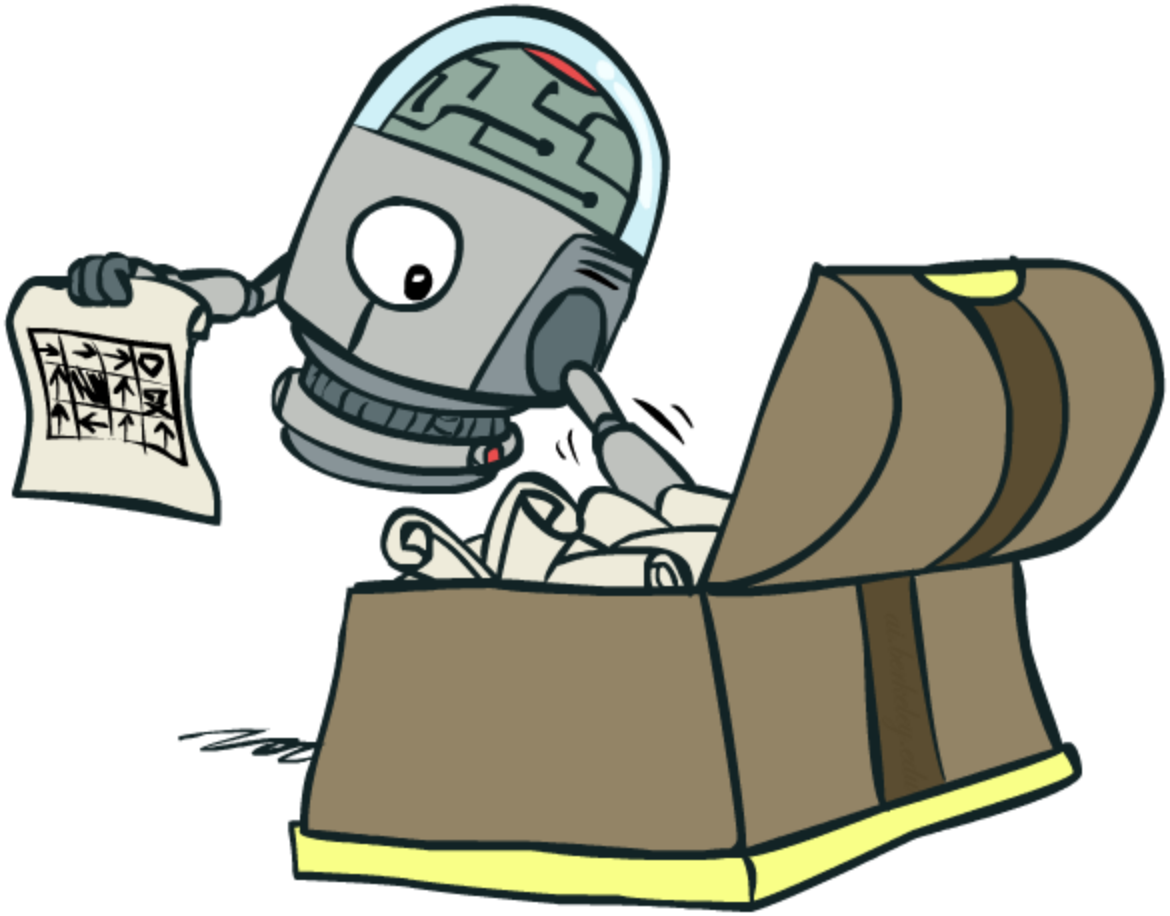
Two approaches to model-free RL

- 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 π_θ
  - Update the parameters θ to optimize performance of policy.

Policy Search



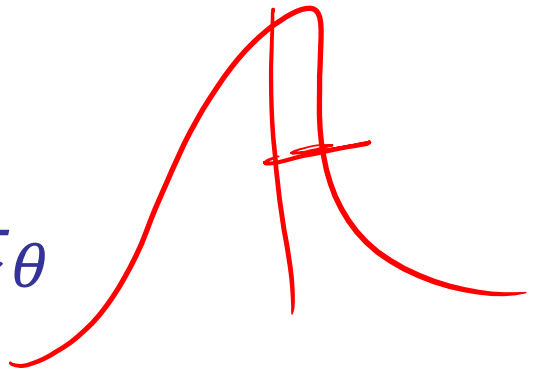
Preliminaries

- Trajectory (rollout, episode) $\tau = (s_0, a_0, s_1, a_1, \dots)$
  - $s_0 \sim \rho_0(\cdot), \quad s_{t+1} \sim P(\cdot\,|s_t, a_t)$
- Rewards $r_t = R(s_t, a_t, s_{t+1})$
- Finite-horizon undiscounted return of a trajectory

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

- Actions are sampled from a parameterized policy π_θ

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



Preliminaries

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

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

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

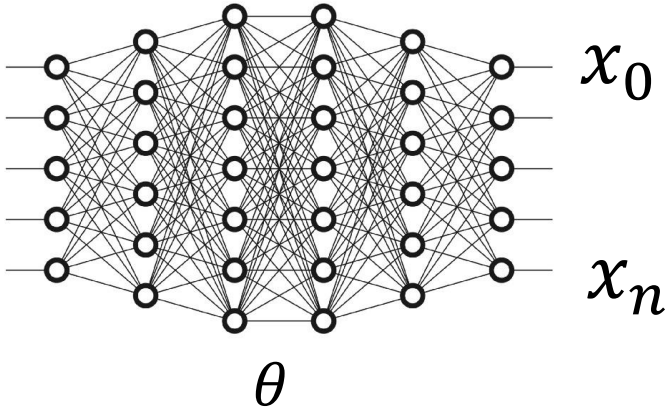
$$J(\pi) = \sum_{\tau} P(\tau|\pi)\, R(\tau) = E_{\tau\sim\pi}[R(\tau)]$$

- Goal of RL: Solve the following optimization problem

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

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?

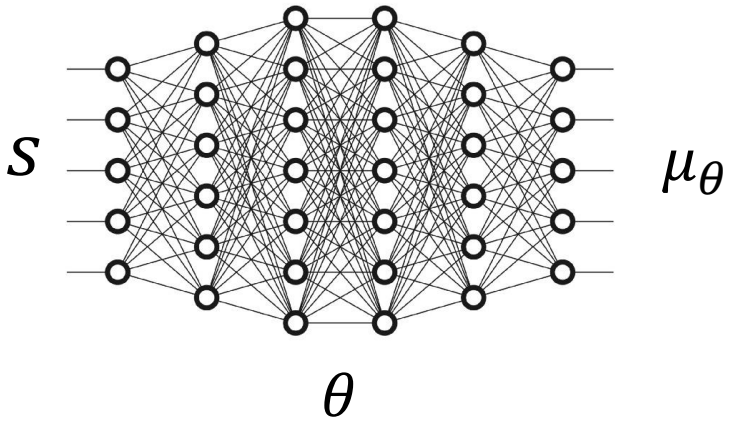
- Diagonal Gaussian (distribution over continuous actions)

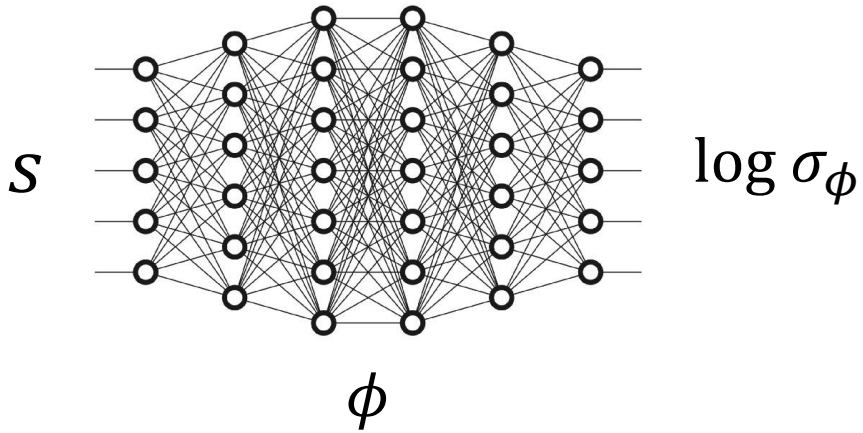
$$a \sim N(\mu, \Sigma)$$

where Σ has non-zero elements only on the diagonal.

Thus, an action can be sampled as

$$a = \mu_\theta(s) + \sigma_\phi(s) \odot z, \qquad z \sim N(0, I)$$





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 $\tau = (s_0, a_0, \ldots s_{T+1})$ given that actions come from π_θ is

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

Fact #2

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

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

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

Fact #3

- Grad-Log-Prob of a Trajectory
  - Note that gradients of everything that doesn't depend on θ is 0.

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

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

Fact #4

$$\frac{\partial}{\partial x}\log f(x) = \frac{1}{f(x)}\frac{\partial}{\partial x}f(x)$$

- Log-Derivative Trick:
  - This is based on the rule from calculus that the derivative of log x is 1/x

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

$$\frac{d}{dx}\log g(x) = \frac{1}{g(x)}\frac{d}{dx}g(x) \quad \Rightarrow \quad g(x)\frac{d}{dx}\log g(x) = \frac{d}{dx}g(x)$$

Derivation of Policy Gradient

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

Try it!

Derivation of Policy Gradient

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

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

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

$$= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)\, R(\tau) \qquad \text{Fact \#4}$$

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

$$= E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\; R(\tau)] \qquad \text{Fact \#3}$$

The Policy Gradient (REINFORCE)

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

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

$\tau = (s_0, a_0, r_0, s_1, \ldots)$

$R(\tau)$

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

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

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

The Policy Gradient (REINFORCE)

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

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

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

Wait, doesn't this remind you of something else?

What does the log probability look like?

- $\log \pi_\theta(a|s) = ?$

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right)$$

https://en.wikipedia.org/wiki/Normal_distribution

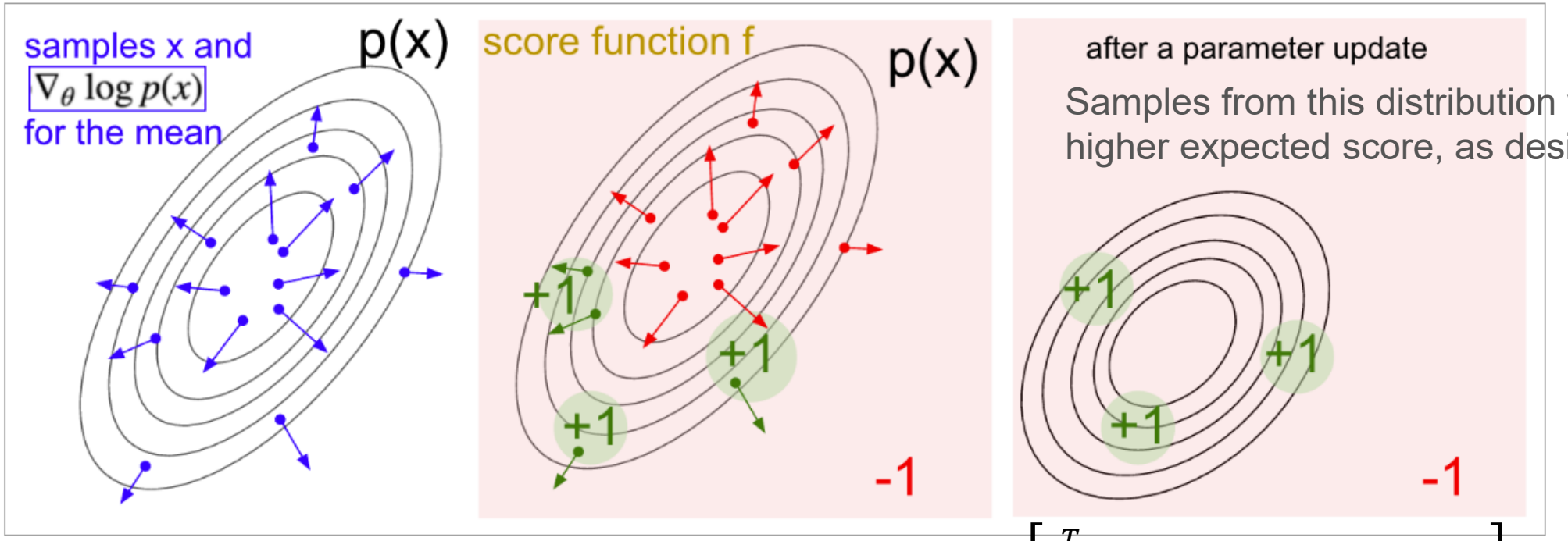
$$\log \pi_\theta(a|s) = -\frac{1}{2}\left[\log(2\pi\sigma^2) + \frac{(a-\mu)^2}{\sigma^2}\right]$$

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.



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

https://karpathy.github.io/2016/05/31/rl/

How would you implement this?

1. Start with random policy parameters θ_0
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

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

$$D = \{\tau_1, \ldots \tau_N\}, \qquad R = \{R(\tau_1), \ldots R(\tau_N)\}$$

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

Simple Pytorch Pseudocode

```
for episode in range(num_episodes):
    state = env.reset()
    trajectory = []

    while True:
        action, log_prob = select_action(policy_net, state)
        next_state, reward, done, _ = env.step(action)

        trajectory.append((log_prob, reward))
        state = next_state

        if done:
            break
```

```
# Compute returns and policy loss
 log_probs, rewards = zip(*trajectory)
 returns = compute_returns(rewards, gamma)
 policy_loss = -sum(log_prob * G
                    for log_prob, G in zip(log_probs, returns))

 # Update policy network
 optimizer.zero_grad()
 policy_loss.backward()
 optimizer.step()
```

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

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

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 approach?
  - Focuses on rewards in the future!
  - Less variance -> less noisy gradients.

Many forms of policy gradients

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

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

Looks familiar….

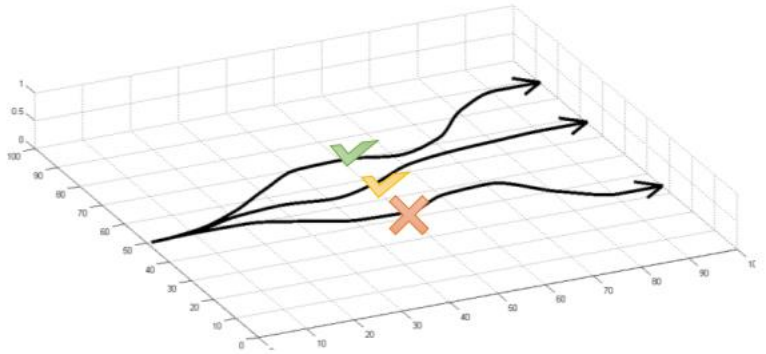
$$\Phi_t = Q^{\pi_\theta}(s_t, a_t)$$

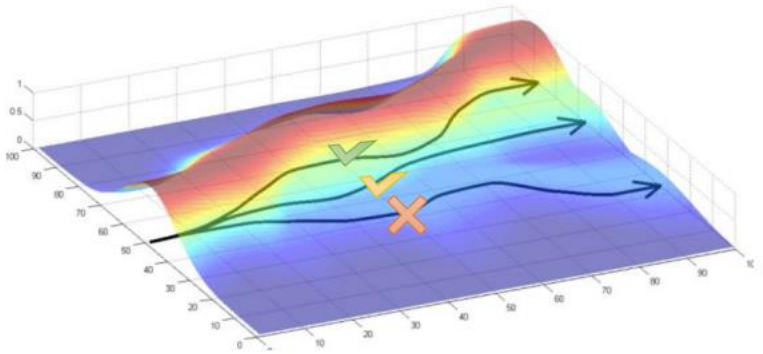
- Now we have an approach that combines a parameterized policy and a parameterized value function!

Baselines

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

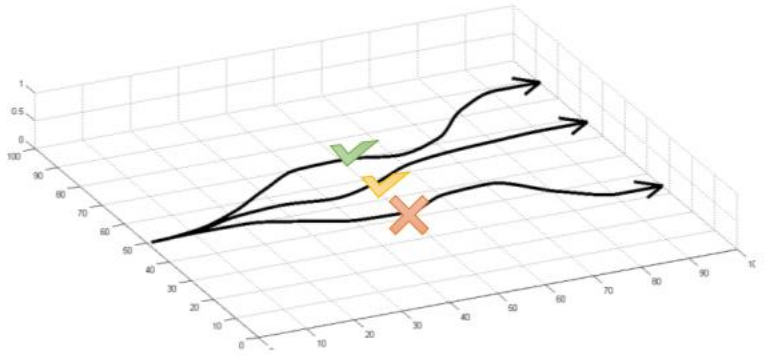
$$\approx \frac{1}{|D|}\sum_{\tau\in D}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)$$





Baselines

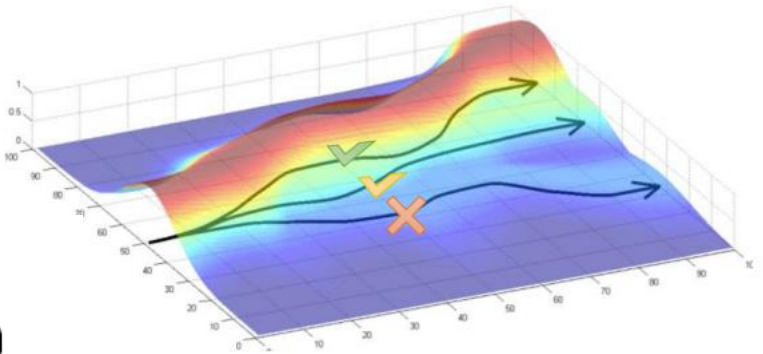
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau)[r(\tau) - b]$$



$$b = \frac{1}{N} \sum_{i=1}^{N} r(\tau)$$

But can we do this?

$$E[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b \, d\tau$$



$$= \int \nabla_\theta p_\theta(\tau) b \, d\tau = b \nabla_\theta \int p_\theta(\tau) d\tau = b \nabla_\theta 1 = 0$$

Many forms of policy gradients

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

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

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

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

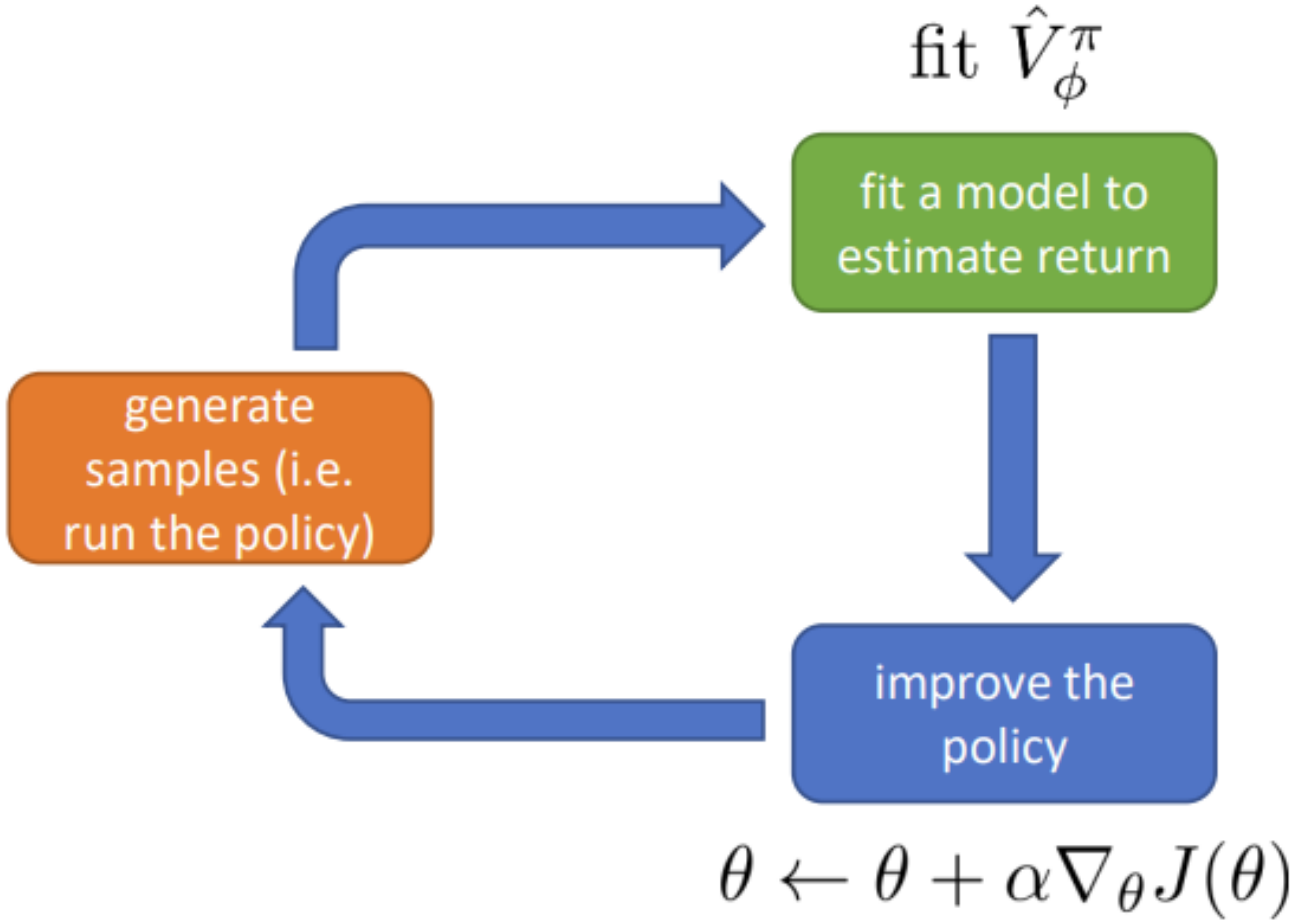
Advantage Function

I rotate the piece

Really bad action

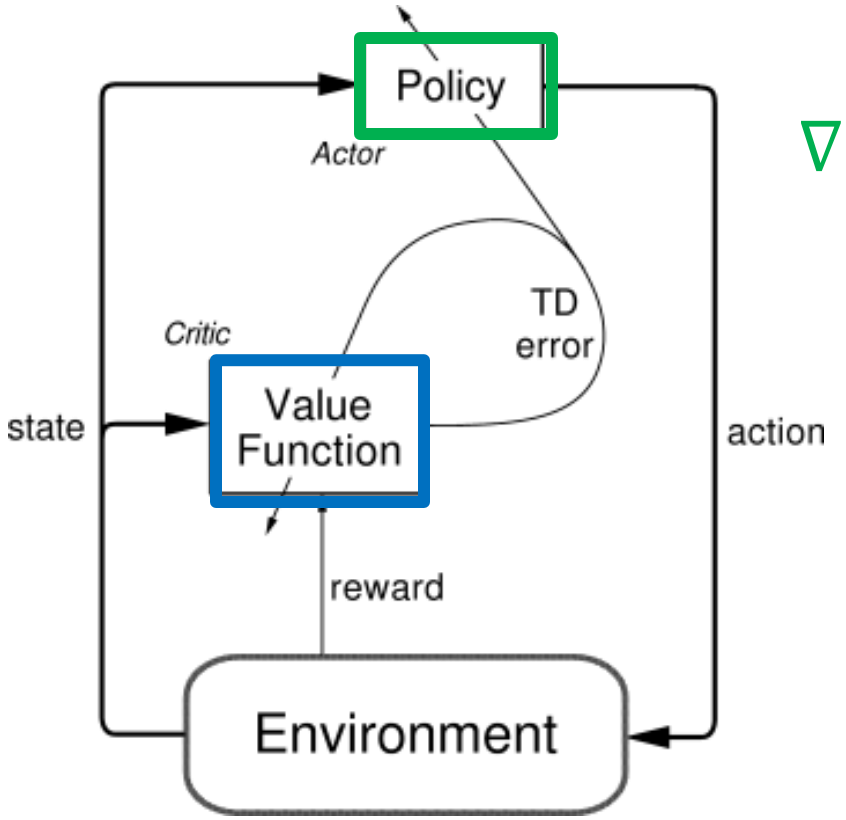
Actor

Critic


fit $\hat{V}_\phi^\pi$
fit a model to estimate return
improve the policy
generate samples (i.e. run the policy)
$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

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, θ, w and learning rates α_θ, α_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"

The Advantage Function

$$A(s,a) = Q(s,a) - V(s)$$

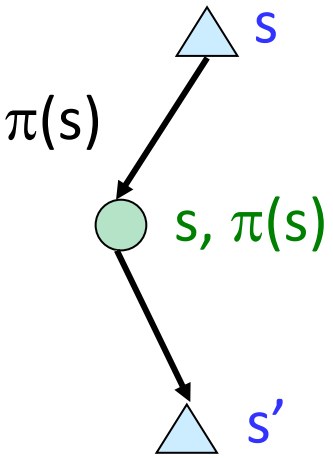
q value for action a in state s

average value of that state

- Why good?
- Why bad?

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

Update to V(s): $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

The Advantage Function

$$A(s,a) = Q(s,a) - V(s)$$

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

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

TD Error

Advantage Actor Critic (A2C)

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

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

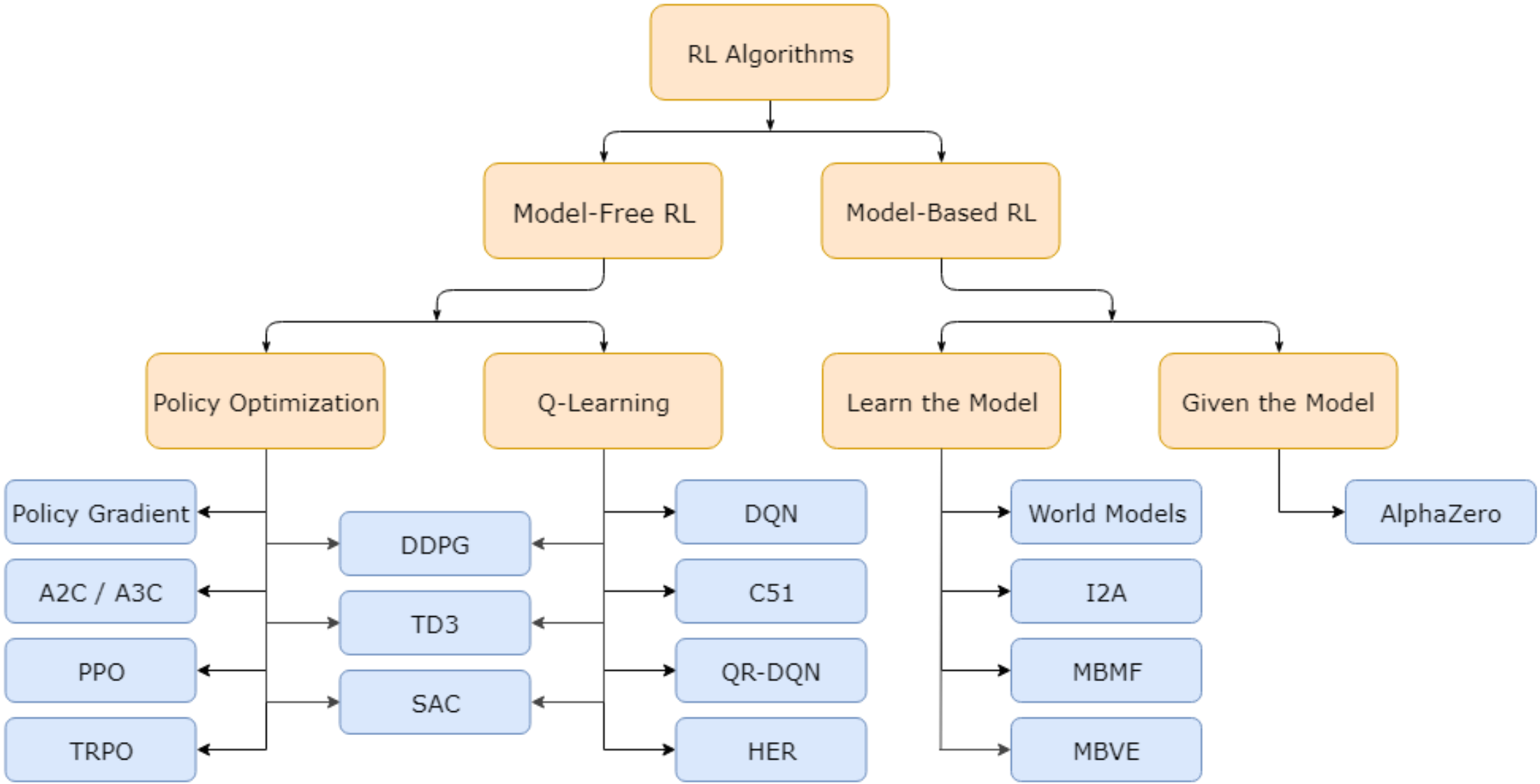
$$\Phi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

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



Asynchronous Advantage Actor Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Asynchronous Advantage Actor Critic (A3C)

- Adds a few tricks
  1. Multiple parallel workers to collect rollouts in different copies of the same env and update the global policy and value models asynchronously
  2. n-step returns
  3. Entropy regularization
  4. Share neural network weights for actor and critic

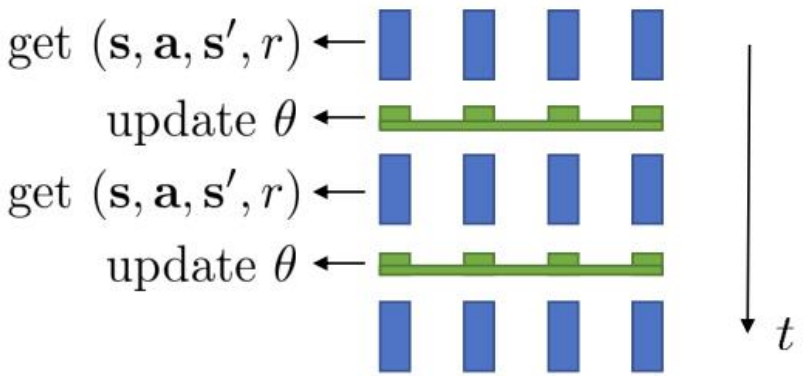
Parallel actors

online actor-critic algorithm:

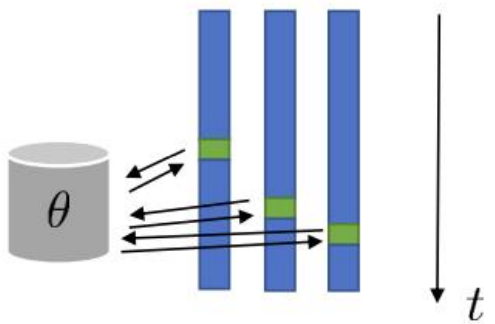
1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

works best with a batch (e.g., parallel workers)

synchronized parallel actor-critic



asynchronous parallel actor-critic



N-Step Returns

- At convergence we want $V^{\pi}(s_t) = E_{\pi}[r_t + \gamma V^{\pi}(s_{t+1})]$
- So given experience (s_t, a_t, r_t, s_{t+1}), TD methods push $V^{\pi}(s_t)$ towards $r_t + \gamma V^{\pi}(s_{t+1})$
- But why only look one step ahead? [1-step return]
- In practice we have experience that looks like this

$$(s_0, a_0, r_0, s_1, s_2, a_2, r_2, s_3, \dots, s_t, a_t, r_t, s_{t+1}, \dots)$$

What if we pushed $V^{\pi}(s_t)$ towards $r_t + \gamma r_{t+1} + \gamma^2 V^{\pi}(s_{t+2})$?

Or even pushed $V^{\pi}(s_t)$ towards $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V^{\pi}(s_{t+3})$?

We can generalize this idea to use n-step returns!

N-Step Returns for A3C updates

Given $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots, s_t, a_t, r_t, s_{t+1}, \ldots r_{T-1}, s_T)$

Compute advantage for each state. If s_T is a terminal state, then define $V_w^\pi(s_T)$=0

$$A(s_t, a_t) = \sum_{i=0}^{T-t-1} \gamma^i r_{t+i} + \gamma^{T-t} V_w^\pi(s_T) - V_w^\pi(s_t)$$

Accumulate gradients for each state and update policy using policy gradient

$$\nabla_\theta \log \pi_\theta(a_t|s_t)\, A_w(s_t, a_t)$$

Update Value function based on TD-error using MSE loss

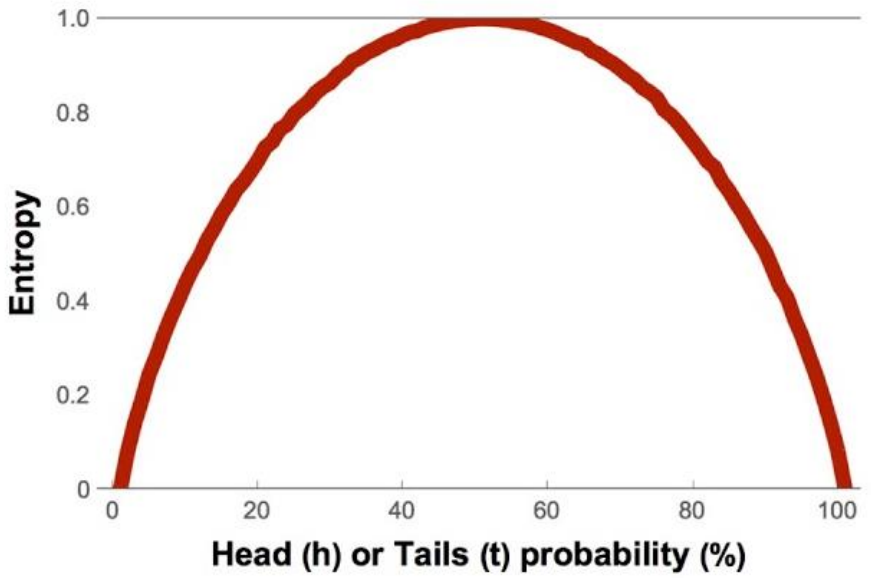
$$\nabla_w \sum_{t=0}^{T-1} \left(\sum_{i=0}^{T-t-1} \gamma^i r_{t+i} + \gamma^{T-t} V_w^\pi(s_T) - V_w^\pi(s_t) \right)^2$$

Shannon Entropy

- Average level of uncertainty associated with a random variable's possible outcomes.

$$P(X = heads) = \frac{1}{2} \qquad P(X = tails) = \frac{1}{2}$$

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$



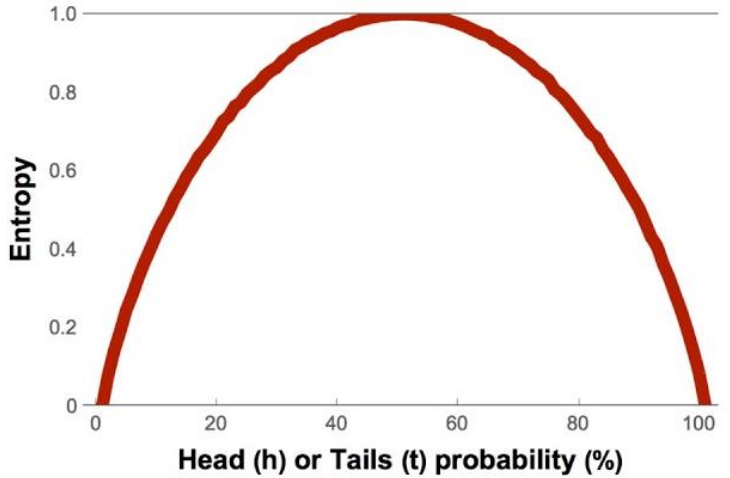
Policy Entropy Bonus

- Improves exploration by discouraging premature convergence to suboptimal deterministic policies.

$$H(\pi) = -\sum_{a} \pi(a|s) \log \pi(a|s)$$

$$H(\pi) = -\int \pi(a|s) \log \pi(a|s) da$$

$$P(X = heads) = \frac{1}{2} \qquad P(X = tails) = \frac{1}{2}$$

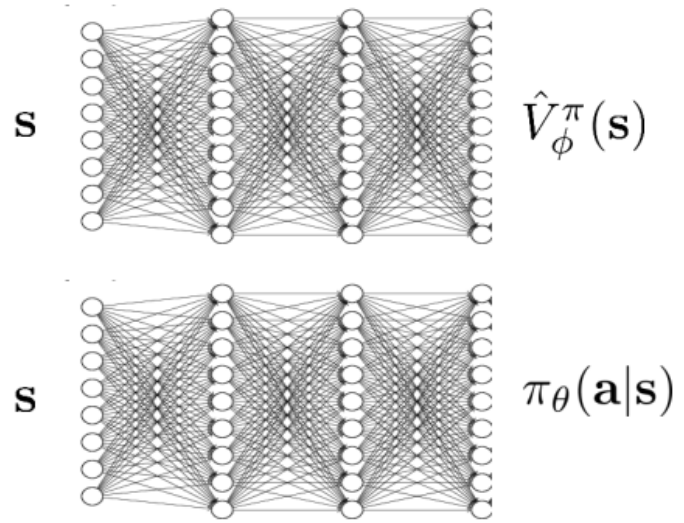


Parameter Sharing

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

two network design



+ simple & stable

- no shared features between actor & critic

shared network design