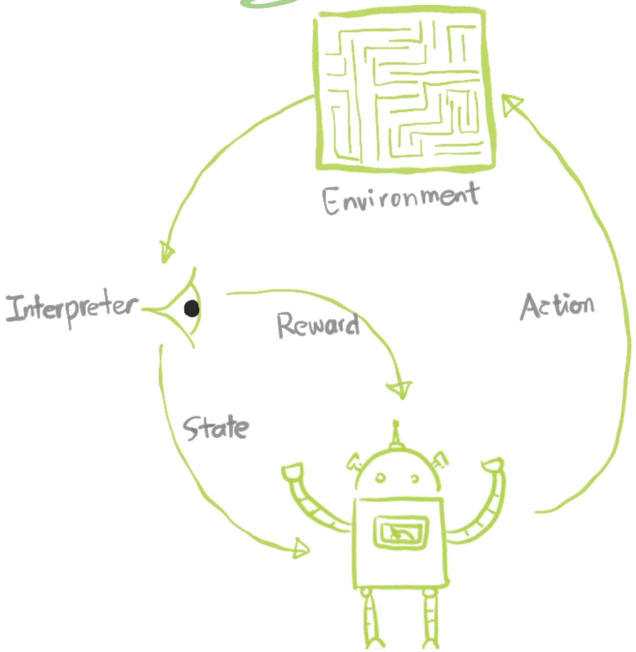
CS5491: Artificial Intelligence

Reinforcement Learning



Instructor: Kai Wang

Today

Why is Reinforcement Learning?

Markov Decision Processes

Algorithm Anatomy

Policy Gradients

Today

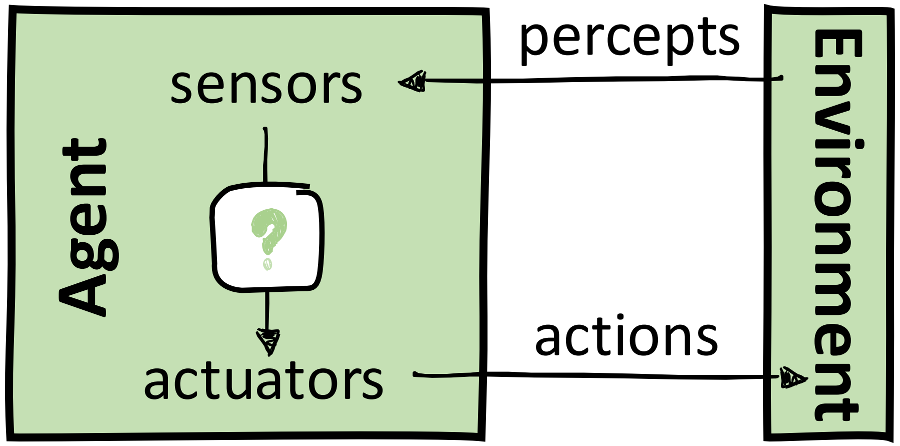
Why is Reinforcement Learning?

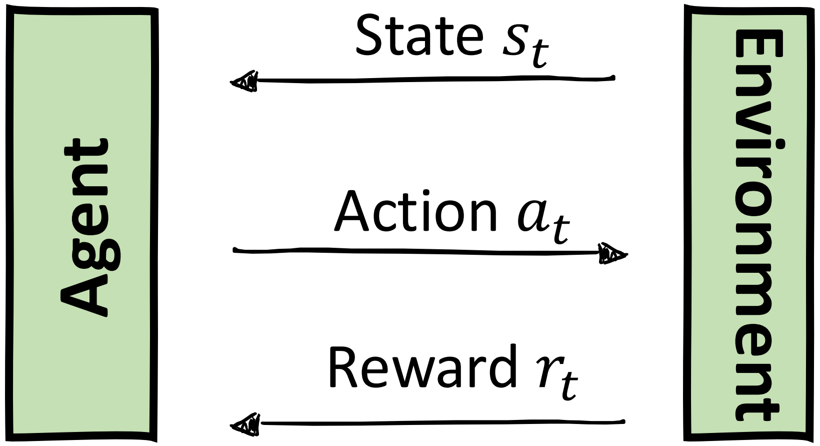
Markov Decision Processes

Algorithm Anatomy

Policy Gradients

Reinforcement Learning





Example: Driving a Car

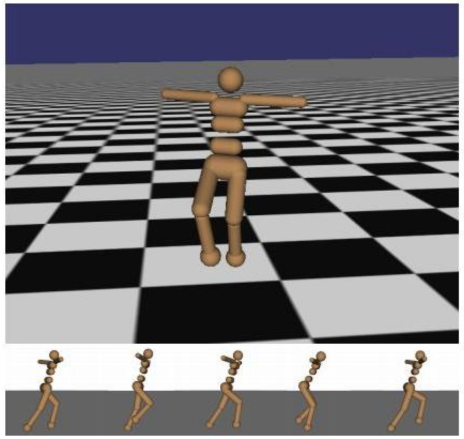
- Objective: drive a car not to hit the lion

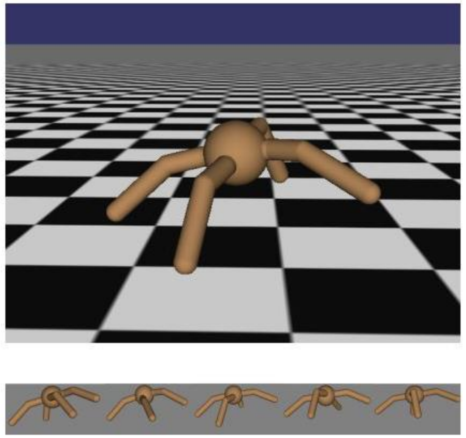
- State: road scene, status of lions (why not percepts?)
- Action: turn left or right

- Reward: 1 at each time step if the lion is not hit.

Example: Robot Locomotion

- Objective: make the robot move forward
- State: angle and position of the joints
- Action: torques applied on joints
- Reward: 1 at each time step upright + forward movement





Example: Atari Games

- Objective: complete the game with the highest score

- State: raw pixel inputs of the game state

- Action: game controls (left, right, up, down)

- Reward: score increase/decrease at each time step

Today

Why is Reinforcement Learning?

Markov Decision Processes

Algorithm Anatomy

Policy Gradients

Markov Decision Process

Markov Decision Process provides a mathematical formulation of the Reinforcement Learning problem, defined as $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$.

- $\mathcal{S}$: set of all possible states
- $\mathcal{A}$: set of all possible actions
- $\mathcal{R}$: distribution of reward given (state, action) pair
- $\mathbb{P}$: transition probability, i.e., distribution over next state given (state, action) pair
- γ : discounting factor

Markov Decision Process

- Step $t = 0$: environment samples initial state $s_0 \sim p(s_0)$
- For $t = 0$ until done:
  - Agent selects action a_t based on a policy $\pi(a_t|s_t)$
  - Environment samples reward $r_t \sim R(\cdot\,|s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot\,|s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

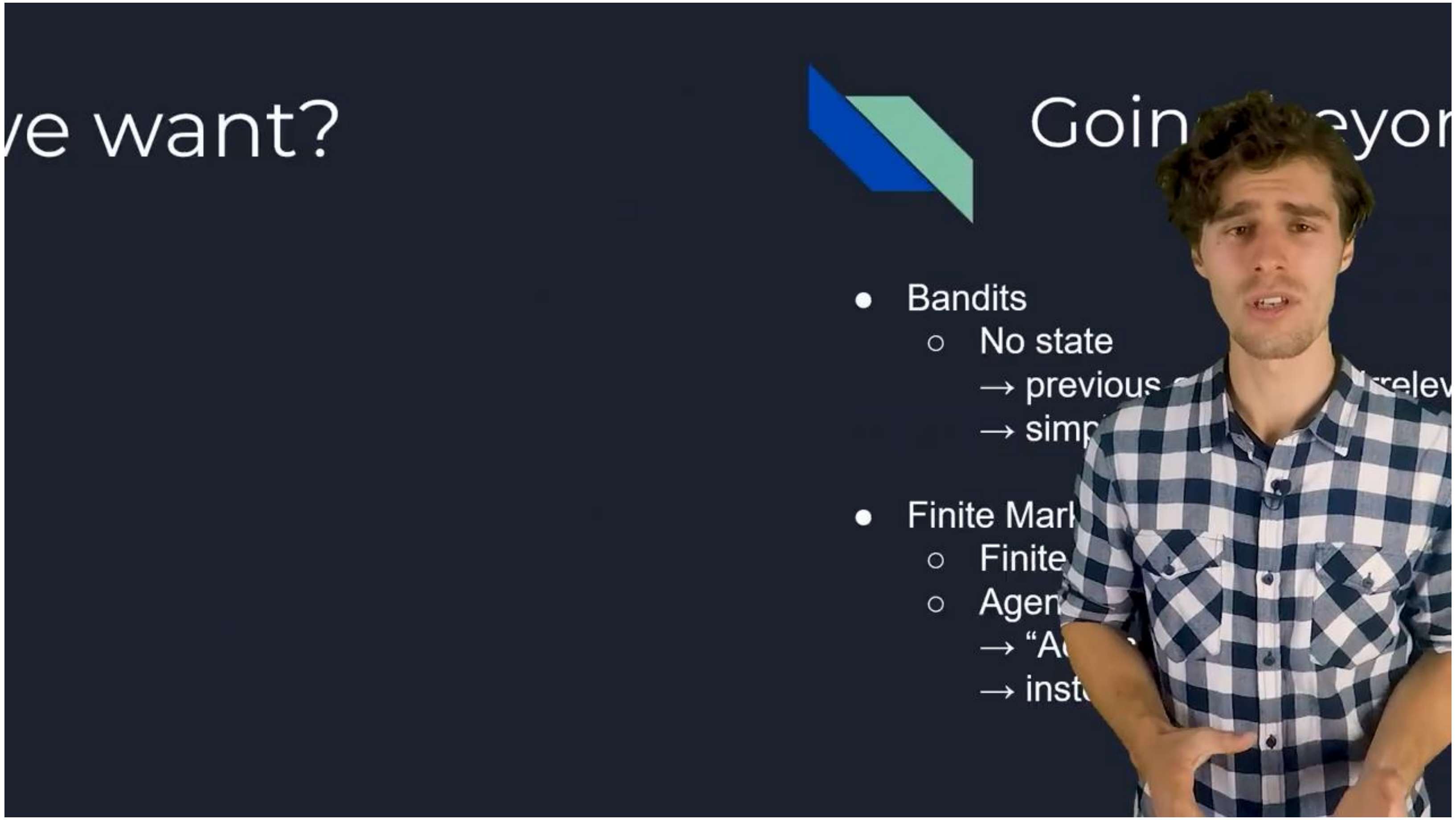
- Objective: find policy π^* that maximizes cumulative discounted reward: $\sum_{t\geq 0} \gamma^t r_t$.

The Optimal Policy

- How do we handle the randomness (initial state, transition probability …)?
  - Maximize the expected sum of rewards!
  - $\pi^* = \text{argmax}_\pi \mathbb{E}[\sum_{t\geq 0} \gamma^t r_t | \pi]$ with
    $s_0 \sim p(s_0), a_t \sim \pi(\cdot\,|s_t), s_{t+1} \sim p(\cdot\,|s_t, a_t)$

- Why discounting factor?

The Optimal Policy



Value Function

Following a policy produces sample trajectories $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots)$. The value function evaluates how good a state s is by measuring the expected cumulative reward from following the policy from state s.

$$V^{\pi}(s) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi]$$

Q-value Function

Following a policy produces sample trajectories $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots)$. The Q-value function evaluates how good a state-action pair (s, a) is by measuring the expected cumulative reward from taking action a in state s and following the policy.

$$Q^{\pi}(s, a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi]$$

Value Function and Q-value Function

- Connections

$$V^\pi(s) = E_{a\sim\pi(a|s)} Q^\pi(s,a)$$

- Using Q-functions and value functions
  - If we have policy π, and we know $Q^\pi(s,a)$, then we can improve π:

    Set $\pi'(a|s) = 1$, if $a = \text{argmax}_a Q^\pi(s,a)$

    This policy is at least as good as π (and probably better)!
  - Compute gradient to increase the probability of good actions a:

    If $Q^\pi(s,a) > V^\pi(s)$, then a is better than average!

Today

Why is Reinforcement Learning?

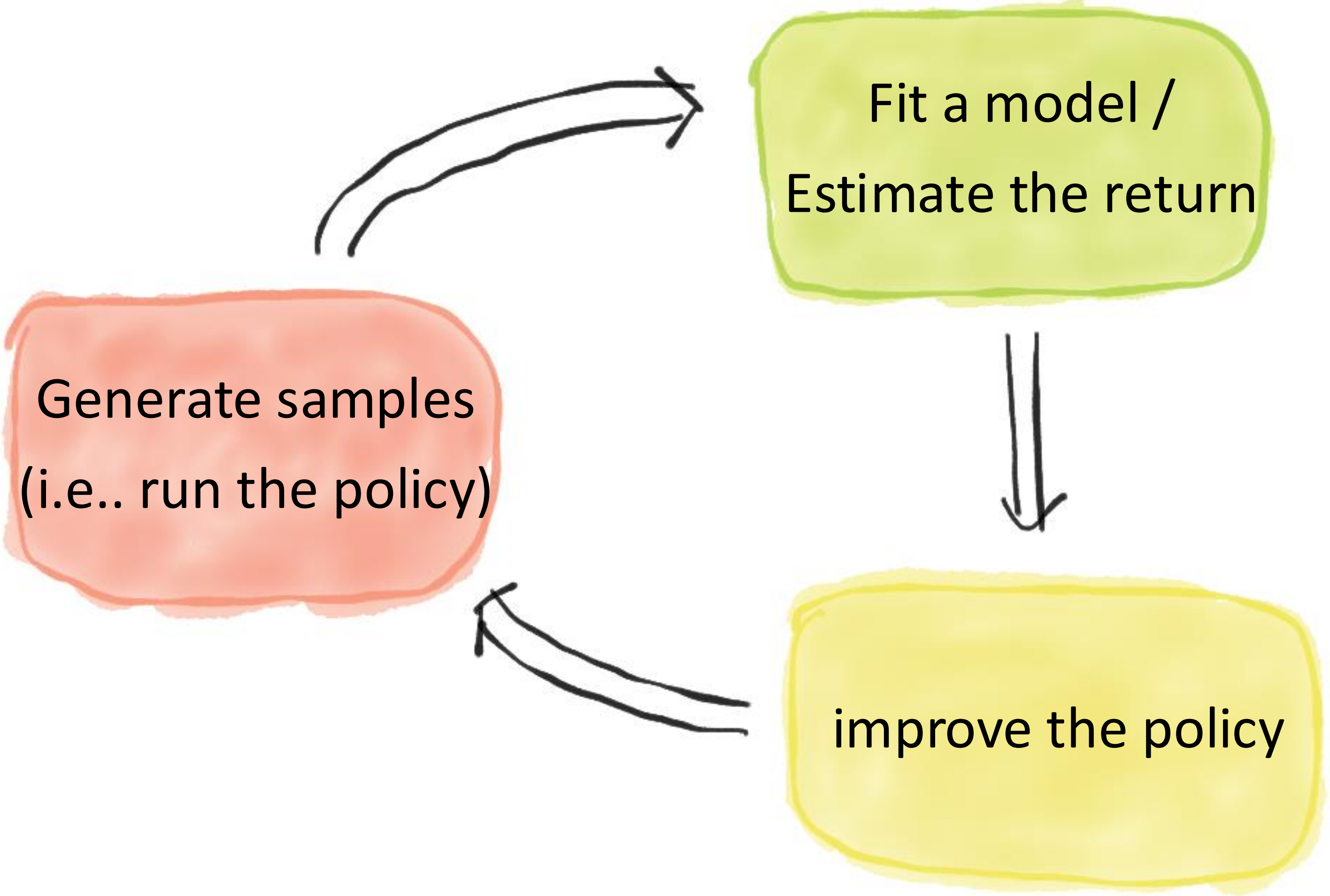
Markov Decision Processes

Algorithm Anatomy

Policy Gradients

Types of RL Algorithms

- Value function based algorithms

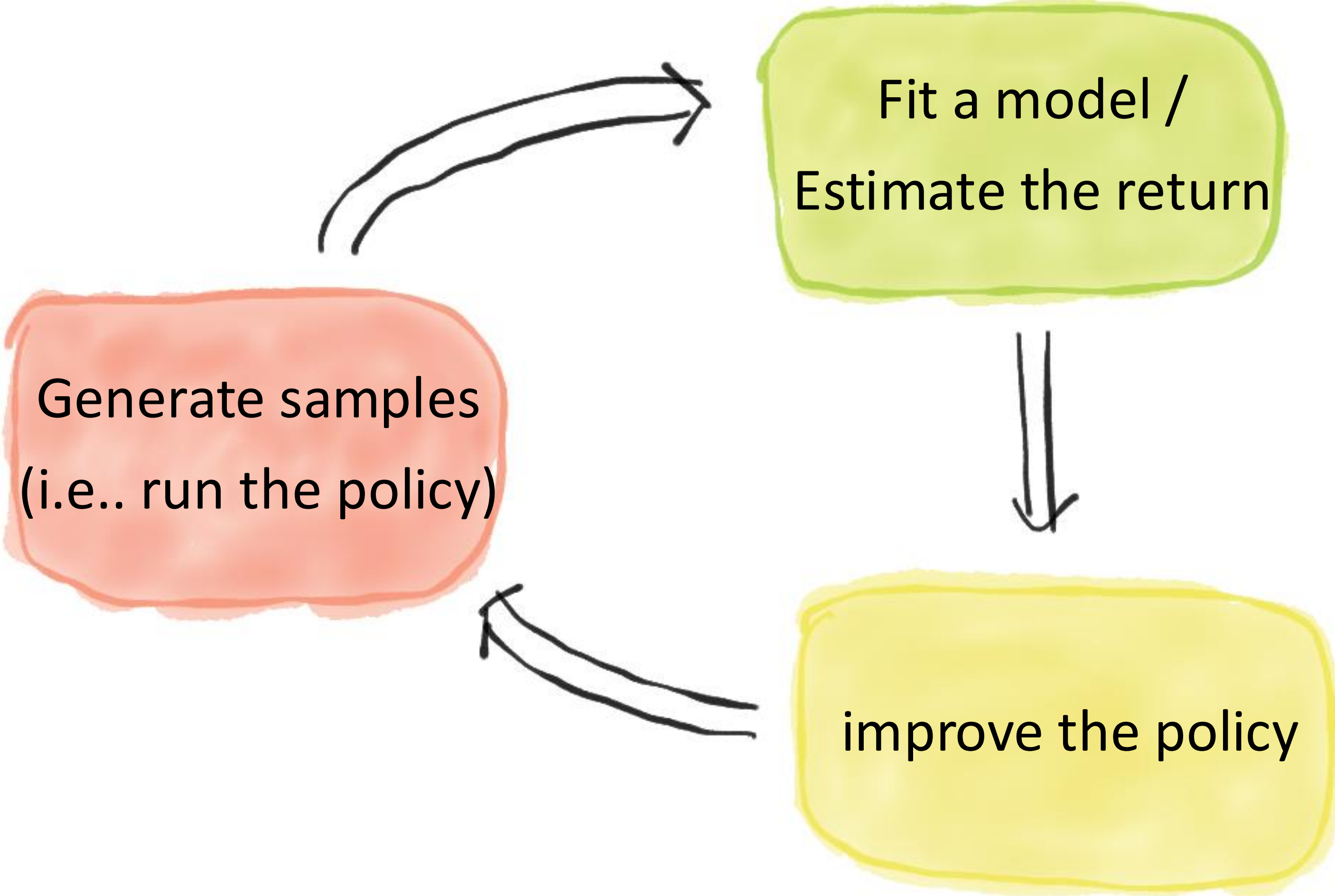


fit $V(s)$ or $Q(s,a)$

set $\pi(s) = \arg\max_{a} Q(s,a)$

Types of RL Algorithms

- Direct policy gradients algorithms



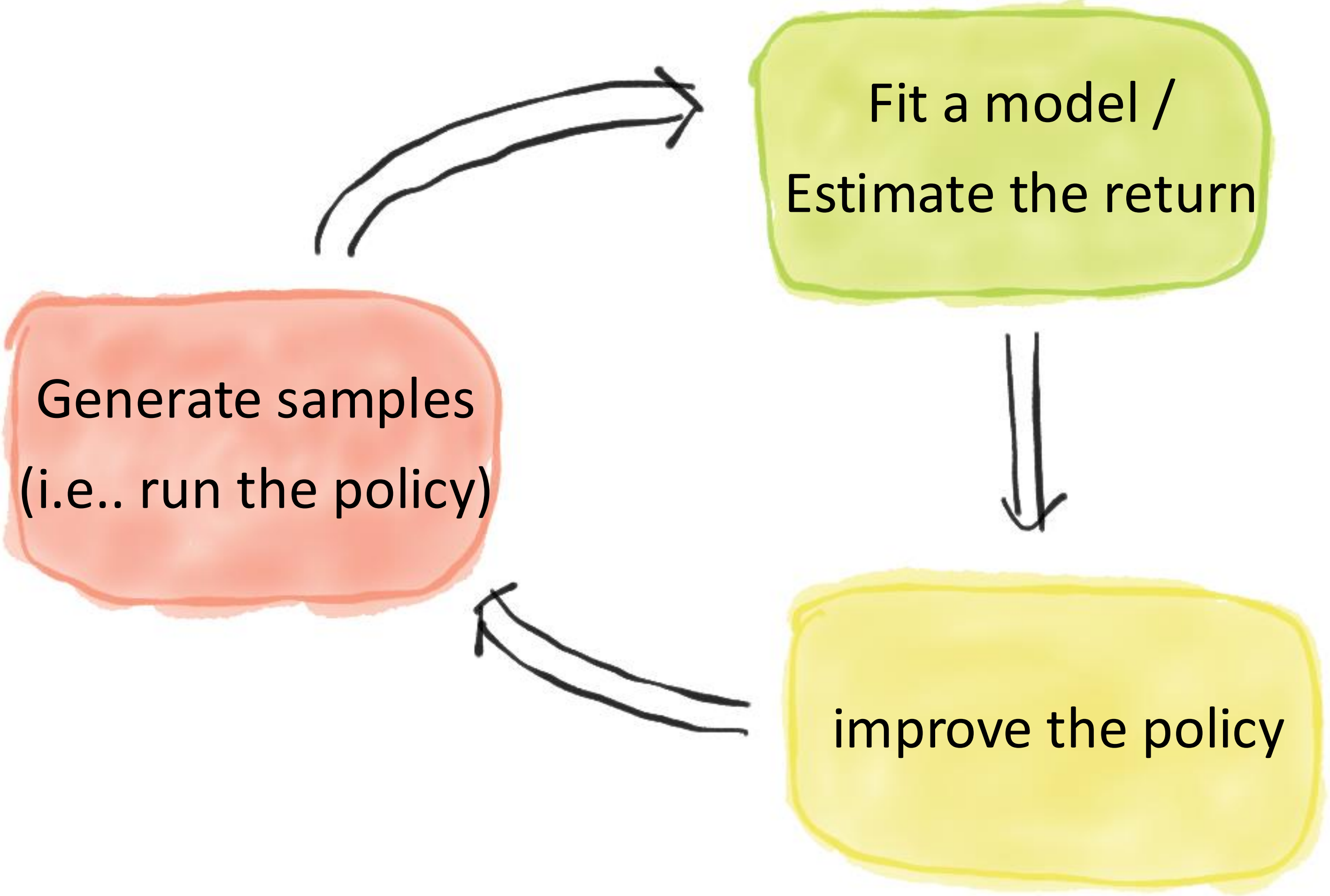
evaluate returns

$$R_\tau = \sum_t r(s_t, a_t)$$

$$\theta \leftarrow \theta + \alpha \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Types of RL Algorithms

- Actor-critic algorithms

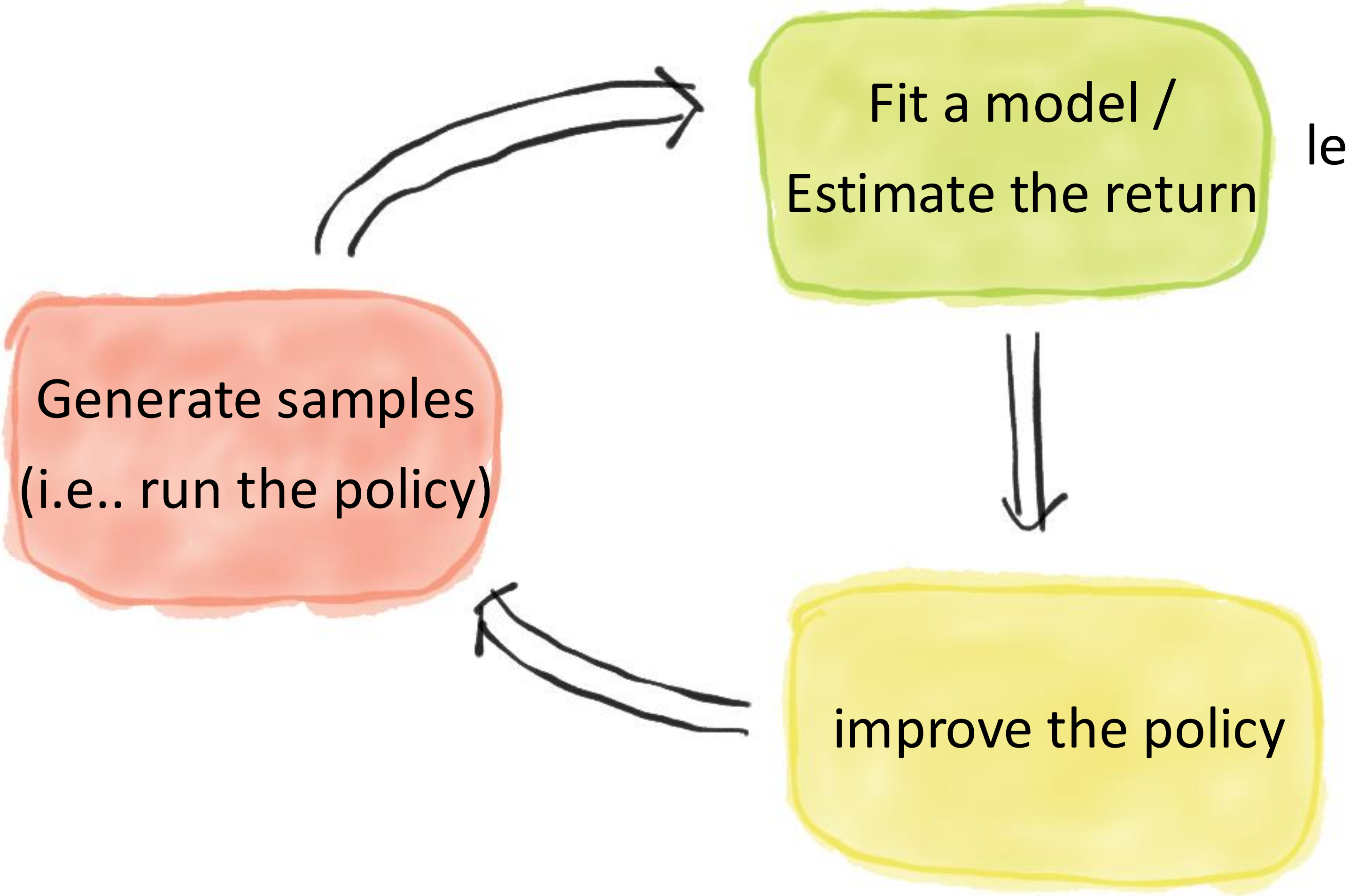


fit $V(s)$ or $Q(s, a)$

$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[Q(s_t, a_t)]$

Types of RL Algorithms

- Model-based algorithms



learn $p(s_{t+1}|s_t, a_t)$

a few options

Model-based Algorithms

- A few options
  - Just use the model to plan (no policy)
    Discrete planning in discrete action spaces, e.g., Monte Carlo tree search
  - Backpropagation gradients into the policy
    Requires some tricks to make it work
  - Use the model to learn a value function
    Dynamic programming

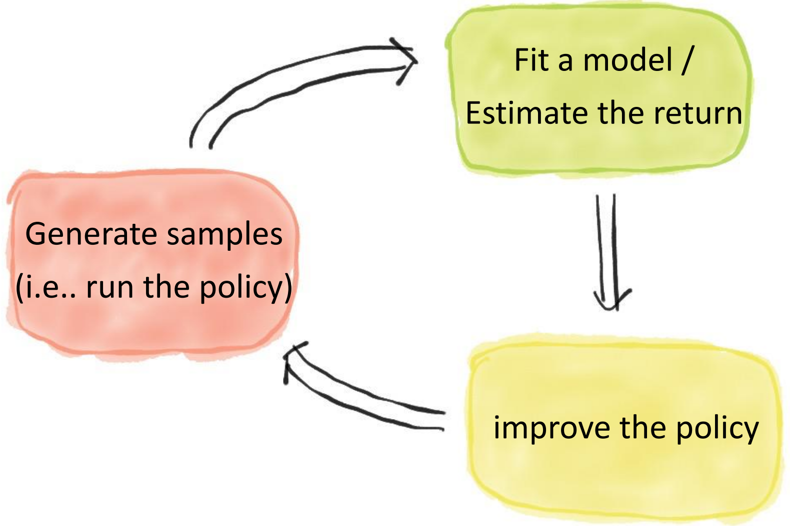
Tradeoffs Between Algorithms

- Different Tradeoffs
  - Sample efficiency
  - Stability & easy of use
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?

Comparison on Sample Efficiency

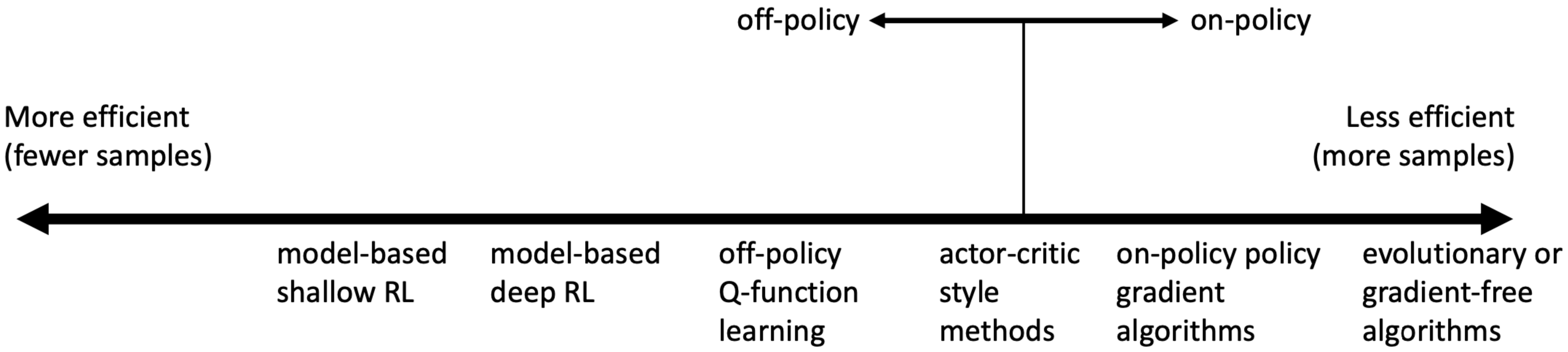
- Sample efficiency = how many samples do we need to get a good policy?

- Most important question: is the algorithm off-policy?
  - Off-policy: able to improve the policy without generating new samples from that policy
  - On policy: each time the policy is changed, even a little bit, we need to generate new samples



$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}[Q(s_t, a_t)]$$

Comparison on Sample Efficiency



Why should we use a less efficient algorithm?

Wall clock time is not the same as sample efficiency!

Comparison on Stability and Ease of Use

- Does it converge?
- And if it converges, to what?
- And does it converge every time?

Why is any of this even a question???

- Supervised learning: almost always gradient descent
- Reinforcement learning: often not gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model is not optimized for expected reward
  - Policy gradient: is gradient descent, but also often the least efficient!

Comparison on Stability and Ease of Use

- Value function fitting
  - At best, minimizes error of fit (“Bellman error”)
    Not the same as expected reward
  - At worst, doesn’t optimize anything
    Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the non-linear case

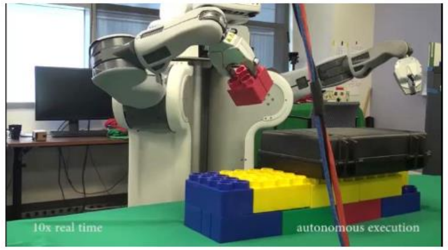
- Model-based RL
  - Model minimizes error of fit -> this will converge
  - No guarantee that better model = better policy

- Policy gradient
  - The only one that actually performs gradient descent on the true objective.

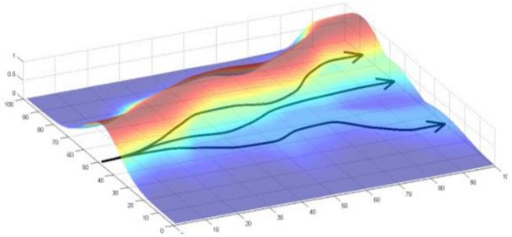
Comparison on Assumptions

- Common assumption #1: full observability
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence

- Common assumption #2: episodic learning
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods



- Common assumption #3: continuity or smoothness
  - Assumed by some continuous value function methods
  - Often assumed by some model-based RL methods



Examples of Algorithms

- Value function fitting methods
  - Q-learning, DQN
  - Fitted value iteration
- Policy gradient methods
  - REINFORCE
  - Trust region policy optimization
- Actor-critic algorithms
  - Asynchronous advantage actor-critic (A3C)
- Model-based RL algorithms
  - Dyna

Today

Why is Reinforcement Learning?

Markov Decision Processes

Algorithm Anatomy

Policy Gradients

Policy Gradients

- We define a class of parameterized policies $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

- Objective: find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r^t \,|\pi_\theta\right]$$

REINFORCE

- Mathematically, we can write

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r^t \,|\pi_\theta\right] = \mathbb{E}_{\tau\sim p(\tau;\theta)}[r(\tau)]$$

$$= \int_\tau r(\tau)p(\tau;\theta)d\tau$$

- $r(\tau)$ is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \cdots)$

REINFORCE

- Expected Reward

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

- Differentiate this

$p(\tau;\theta)\nabla_\theta \log p(\tau;\theta) = \nabla_\theta p(\tau;\theta)$

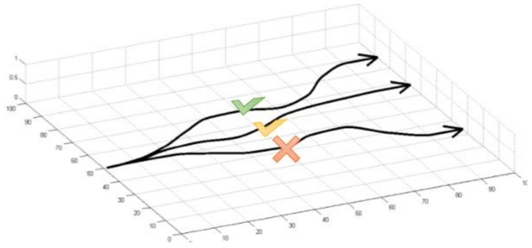
$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

$$= \int_{\tau} r(\tau)p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)\, d\tau$$

$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

REINFORCE

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$



We have $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

Thus $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i}^{N}\sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_{it}|s_{it})$$

Intuition:

If $r(\tau)$ is high, push up the probabilities of the actions seen

If $r(\tau)$ is low, push down the probabilities of the actions seen

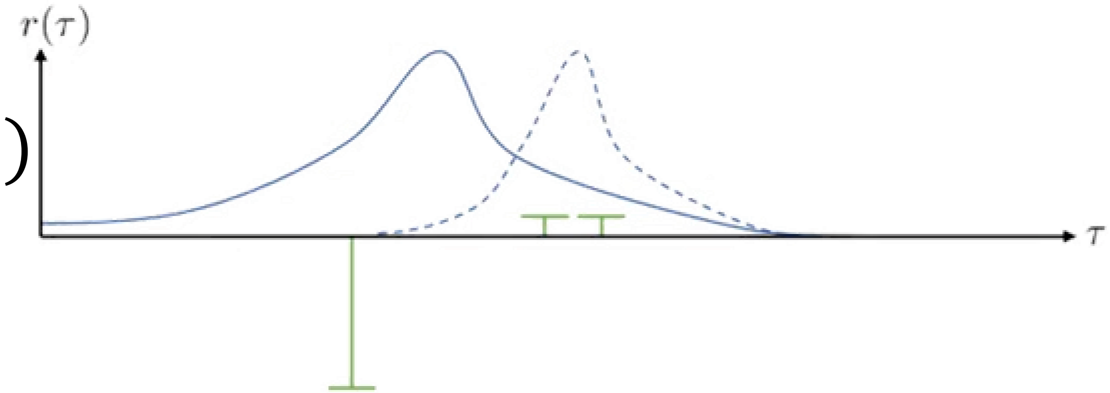
REINFORCE

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

$$v_t = \sum_t^T r_t$$

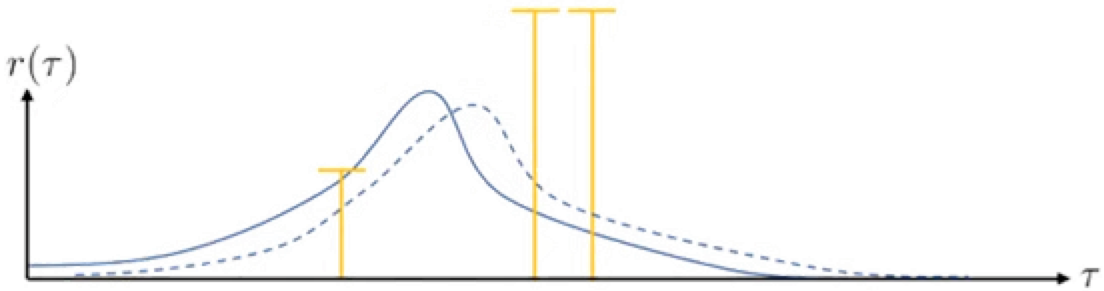
Problems with the Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i}^{N}\sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_{it}|s_{it})$$



High variance!

Even worse: what if the two good samples have $r(\tau) = 0$?



Reducing Variance

- Using causality

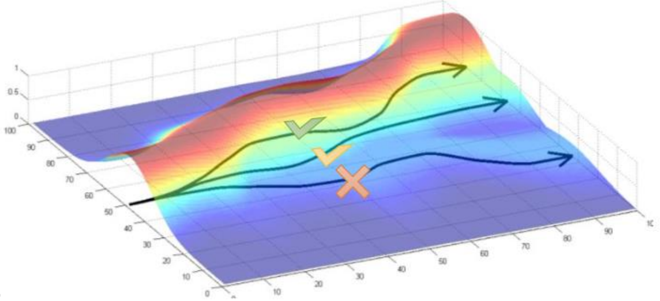
  Policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i}^{N} \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta (a_{it} | s_{it})$$

$$\Downarrow$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i}^{N} \sum_{t \geq 0} \nabla_\theta \log \pi_\theta (a_{it} | s_{it}) \sum_{t'=t}^{T} r(s_{it'}, a_{it'})$$

Baselines



- Using a baseline

  Subtracting a baseline is unbiased in expectation!

  Average reward is not the best baseline, but it's pretty good!

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i^N \sum_{t\geq 0} (r(\tau) - b)\nabla_\theta \log \pi_\theta(a_{it}|s_{it}) \qquad b = \frac{1}{N}\sum_i^N r(\tau)$$

$$E[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) b \, d\tau = \int \nabla_\theta p_\theta(\tau) b \, d\tau = b\nabla_\theta \int p_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

$$p(\tau;\theta)\nabla_\theta \log p(\tau;\theta) = \nabla_\theta p(\tau;\theta)$$

Optimal Baseline

$$\mathrm{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]$$

$$\mathrm{Var} = E_{\tau\sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2] - E_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2$$

$$\frac{d\mathrm{Var}}{db} = \frac{d}{db}E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right)$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

This is just expected reward, but weighted by gradient magnitudes!

Off-policy Policy Gradients

- On policy learning can be extremely inefficient!

```
function REINFORCE
    Initialise θ arbitrarily
    for each episode {s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T} ~ π_θ do
        for t = 1 to T − 1 do
            θ ← θ + α∇_θ log π_θ(s_t, a_t) v_t
        end for
    end for
    return θ
end function
```

Off-policy Policy Gradients

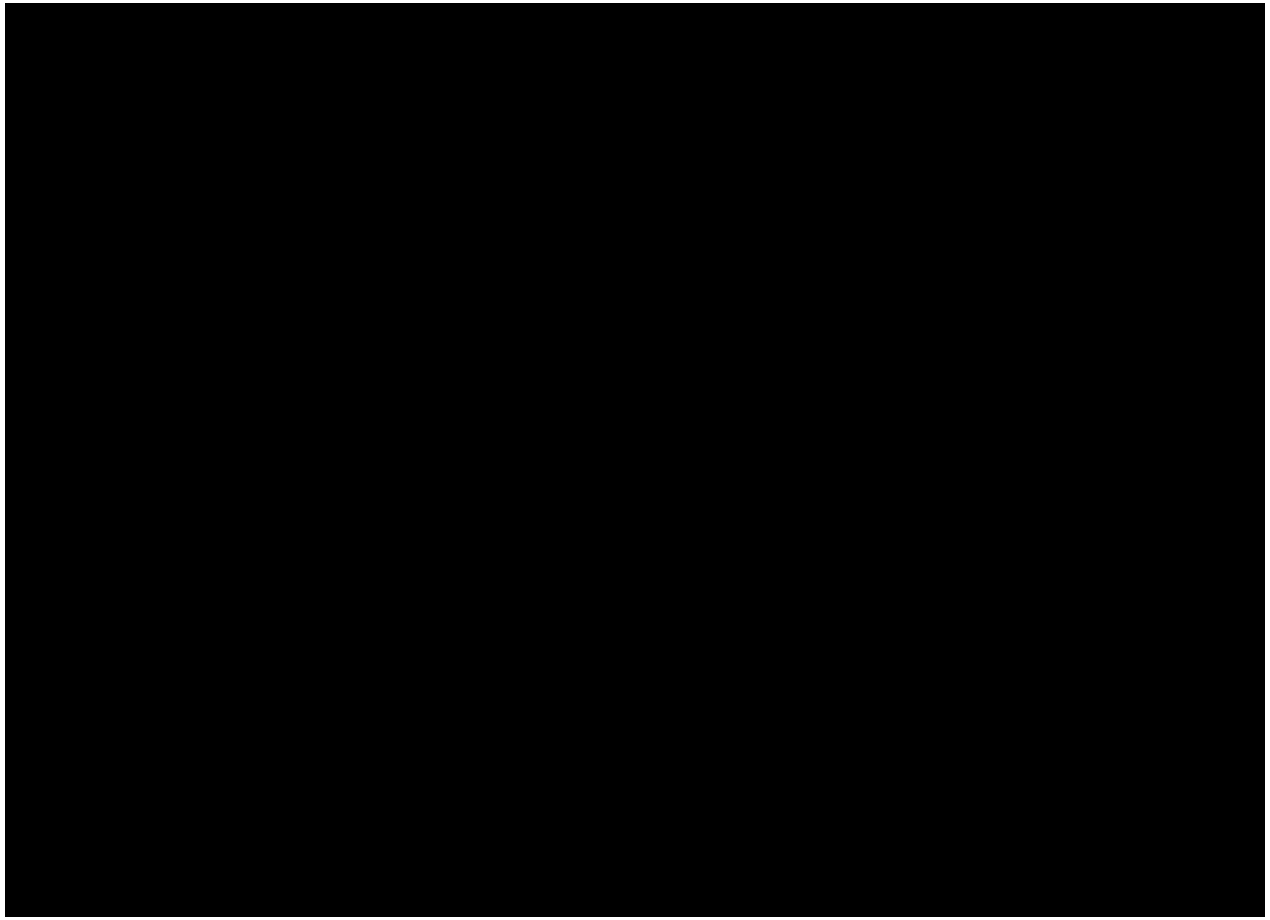
- We have off-policy samples $p(\tau;\bar{\theta})$ (i.e., $\bar{p}(\tau)$) instead

$$J(\theta) = E_{\tau\sim\bar{p}(\tau)}\left[\frac{p_\theta(\tau)}{\bar{p}(\tau)} r(\tau)\right]$$

$$p_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\frac{p_\theta(\tau)}{\bar{p}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

RL Applications



Goals

- Understand what Reinforcement Learning (RL) is.
- Understand categorization of existing Reinforcement Learning algorithms.
- Learn how to use Keras to implement REINFORCE.
- Learn how to debug REINFORCE and make it work in practice.

Important This Week

- Read Reinforcement Learning: An Introduction.
- Know more about Reinforcement Learning algorithms here. http://rail.eecs.berkeley.edu/deeprlcourse/
- Know more about implementation issues of RL here. https://spinningup.openai.com/en/latest/