Microsoft

Policy Gradient & Actor Critic

MATTHEW HAUSKNECHT
Researcher, MSR-AI

Objectives

- Understand Policy Gradient and Actor-Critic Reinforcement Learning.
- Implement REINFORCE
- Understand bias and variance of estimators
- Implement a baselined version of REINFORCE
- Understand Actor-Critic Methods
- Implement an A3C like algorithm

Acknowledgement

The content in this module is inspired by:

- Pieter Abbeel's lecture on PGAC: https://sites.google.com/view/deep-rl-bootcamp/lectures
- Andrej Karpathy's blogpost Pong from Pixels: http://karpathy.github.io/2016/05/31/rl/
- Jan Peter's article: http://www.scholarpedia.org/article/Policy_gradient_methods

Policy Optimization

Reinforcement Learning

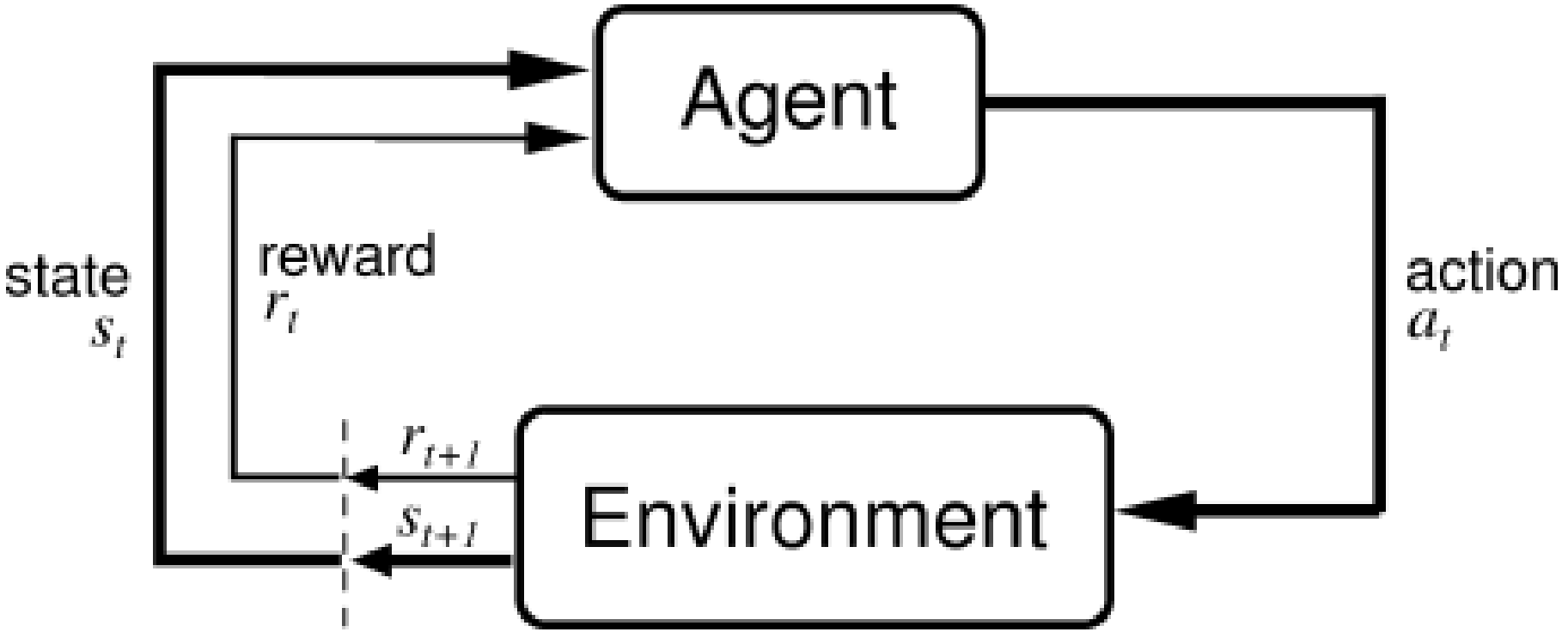
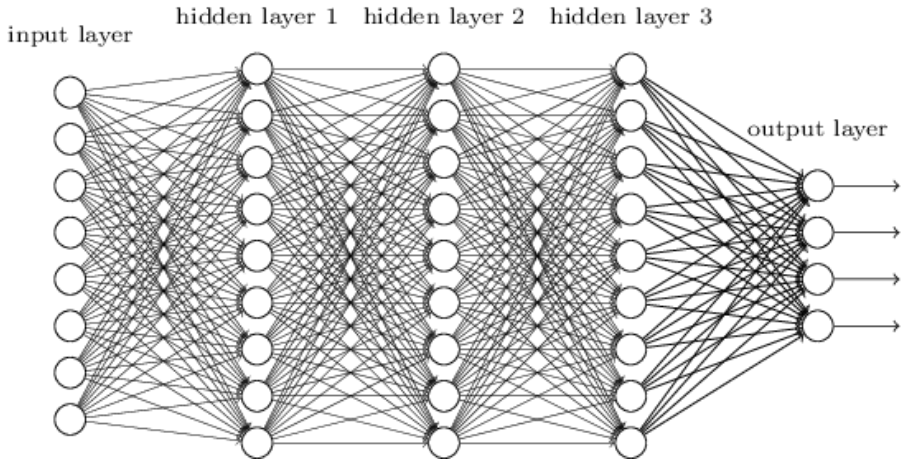


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Policy Optimization



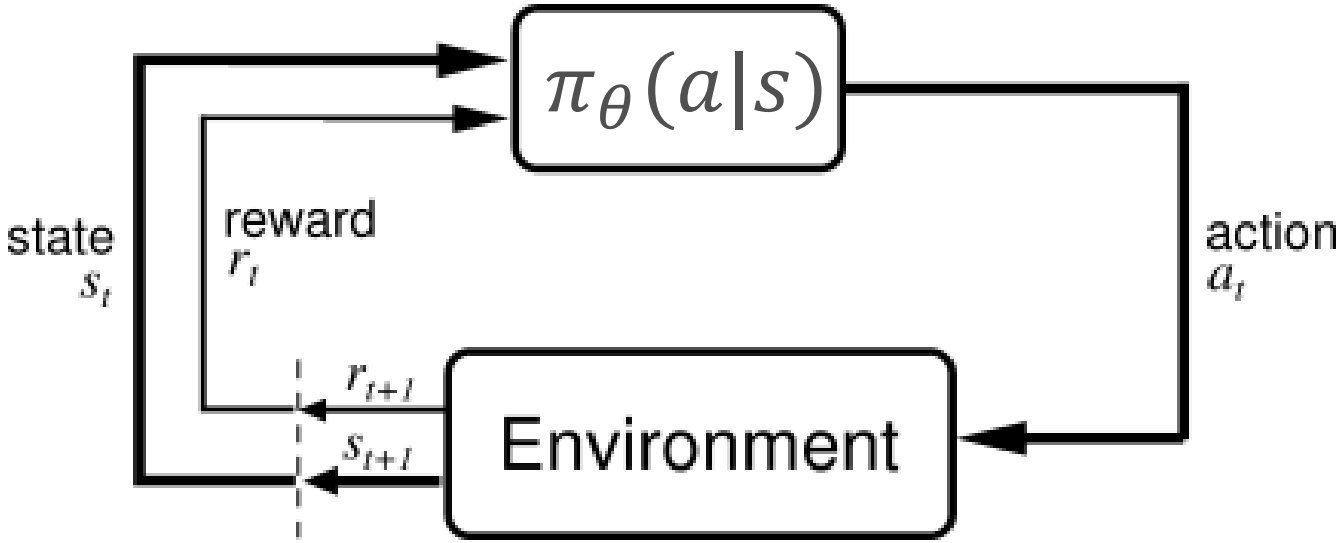
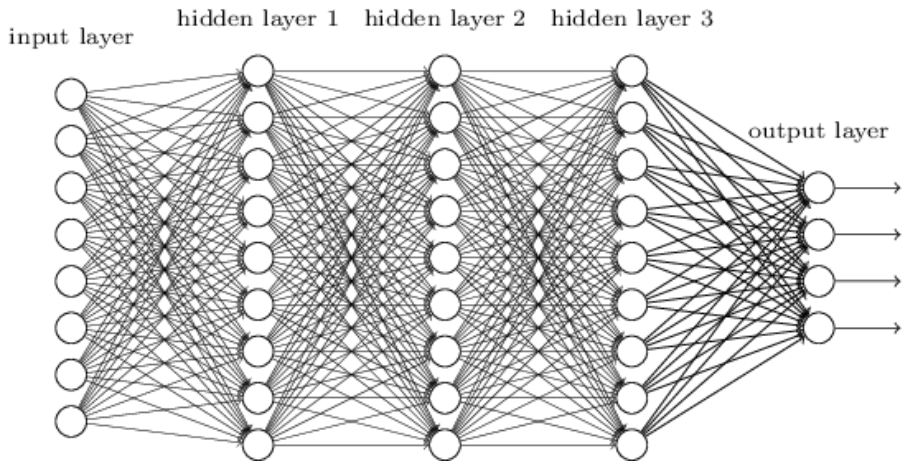
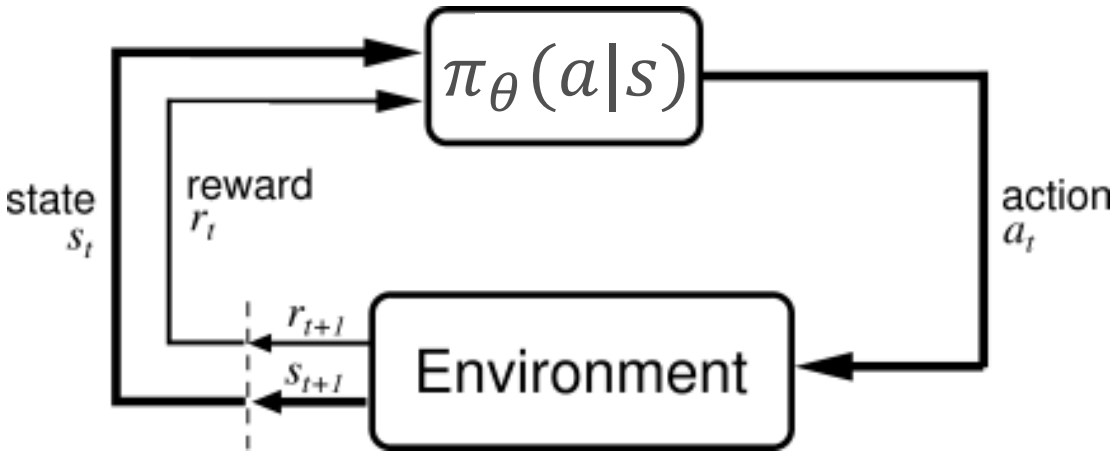


Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Policy Optimization





Given: Policy parameterized by θ

Objective:

$$\max_\theta \; E_{\pi_\theta}[\sum_{t=0}^{T} R(s_t, a_t)]$$

Doesn't care about value functions, dynamic programming, or models.

Only cares about finding θ that maximizes rewards.

Image Credit: Sutton and Barto, Reinforcement Learning, An Introduction 2017

Advantages / Disadvantages of Policy Gradient

Policy Gradient Methods

Advantages

- Local convergence properties
- Effective in high dimensional or continuous action spaces: Hard to solve $\arg\max_a Q(s,a)$
- Learns **stochastic** policy $\pi(a|s)$
- Often, learning π can be simpler than Q or V

Stochastic Policy

Certain problems are better solved by stochastic policies.

Consider repeated **Rock-Paper-Scissors**:

- Any deterministic policy is easily exploited
- Only a uniform random policy is optimal
  - Policy gradient methods can learn such a stochastic policy:
    $\pi(rock) = \pi(paper) = \pi(scissors) = 1/3$

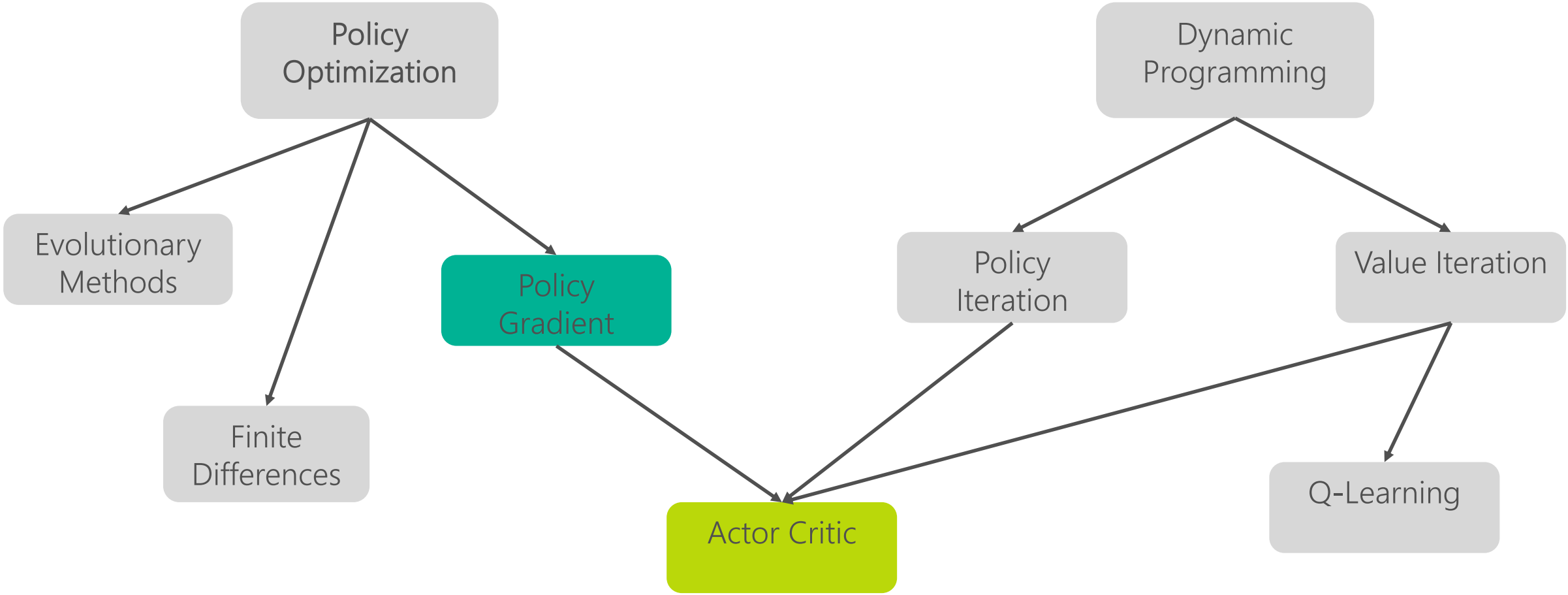
Policy Gradient Methods

Disadvantages

- Learning can be less sample efficient than competing methods
- Converges to a local optimum rather than global

Relationship to other RL methods

Relationship to other RL methods



Policy Gradient Derivation

Likelihood Ratio Methods

- Let trajectory τ be a sequence of states and actions $s_0, a_0, \ldots, s_T, a_T$ with return $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$.
- Define Expected Return:

$$J(\theta) = E[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta] = \sum_\tau P(\tau; \theta) R(\tau)$$

- **Goal**: find θ that maximizes:

$$\max_\theta J(\theta) = \max_\theta \sum_\tau P(\tau; \theta) R(\tau)$$

Likelihood Ratio Methods

$$J(\theta) = \sum_\tau P(\tau;\theta) R(\tau)$$

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau)$$

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

$$= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau)$$

$$= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau)$$

$$= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) \; R(\tau)$$

Definition:

$$\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$$

Likelihood Ratio Methods

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)\ R(\tau)$$

In general, we wish to approximate this using a set of m trajectories sampled from π_θ:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P\left(\tau^i;\theta\right) R(\tau^i)$$

Policy Gradient Intuition

Likelihood Ratio Methods: Intuition

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P\left(\tau^i; \theta\right) R(\tau^i)$$

- Gradient increases the likelihood of trajectories with high reward.
- Decreases the likelihood of trajectories with low/negative reward.

Likelihood Ratio Methods: Intuition

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P\left(\tau^i;\theta\right) R(\tau^i)$$

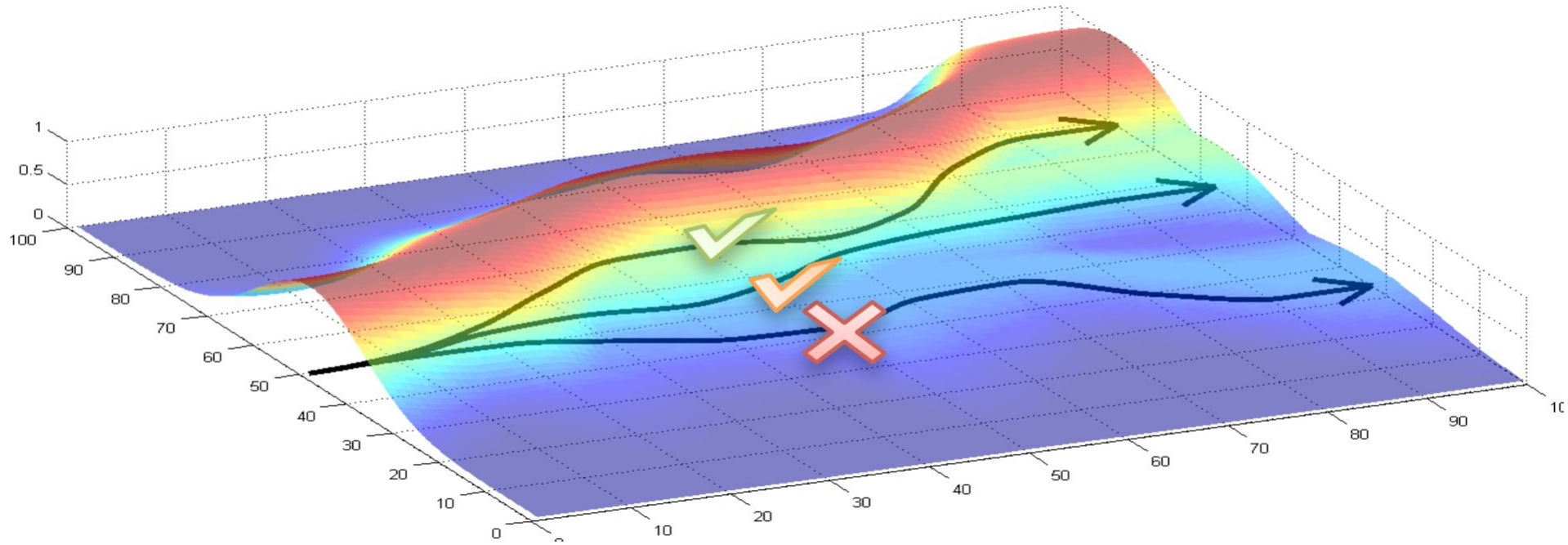


Image Credit: Pieter Abbeel, Policy Gradients and Actor Critic, Deep RL Bootcamp, 2017

From trajectories to states/actions

Likelihood Ratio Methods

Decomposing the trajectories into states & actions:

$$\nabla_\theta \log P(\tau;\theta) = \nabla_\theta \log[\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)]$$

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

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

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

Since $\frac{\partial P}{\partial \theta} = 0$

No model of the dynamics required!

Likelihood Ratio Methods

Reassembling the equation:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P\left(\tau^i; \theta\right) R(\tau^i)$$

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R\left(\tau^i\right)$$

This is an unbiased expression of the policy gradient.

Unbiased means that as $m \to \infty$, it will converge to true gradient of J

Likelihood Ratio Methods

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

This gradient is the basis for the algorithm called **REINFORCE***

***Simple statistical gradient-following algorithms for connectionist reinforcement learning, Williams, 1992

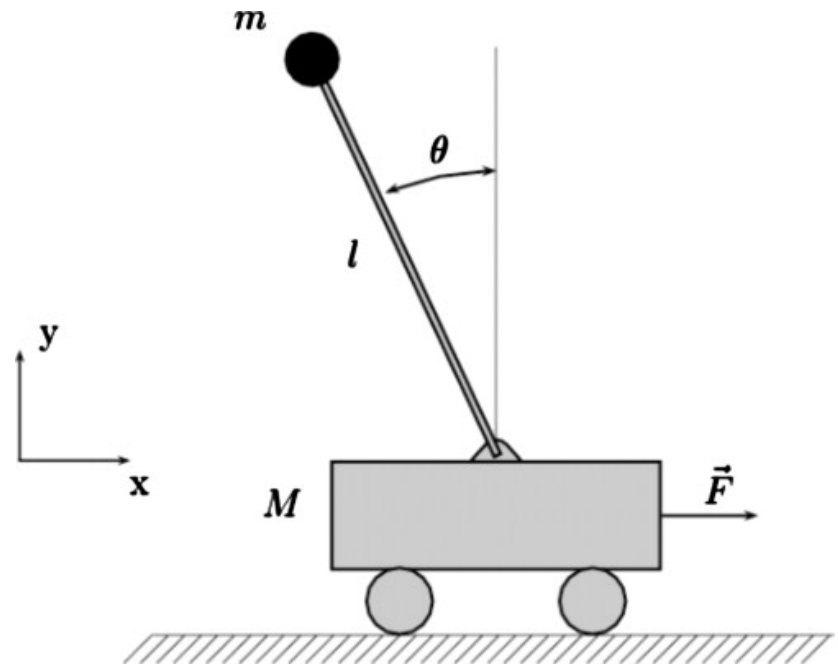
REINFORCE algorithm

REINFORCE

1. Create initial policy π_θ
2. Act in the environment $a_t \sim \pi(s_t)$ for an episode storing states, actions, rewards: $s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
3. Compute return $R = \sum_{t=0}^{T} r_t$
4. Compute policy gradient $\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R$
5. Take a step in the direction of the gradient: $\theta = \theta + \alpha \nabla_\theta J(\theta)$
6. Repeat until convergence

Cartpole Domain & Reinforce Update

Cartpole



Objective: Balance the pendulum by applying force to the cart

State: $\{x, \dot{x}, \theta, \dot{\theta}\}$

2 Discrete Actions: Apply force to cart left/right

Rewards: +1 for each step

Termination: Pole falls below certain angle (failure), cart reaches end of track (failure), or 200 steps (success).

Image Credit: Optimal Control of Underactuated Mechanical Systems: A Geometric Approach

Neural Network Update

It's necessary to compute the policy gradient:

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

Specifically consider the term $\nabla_\theta \log \pi_\theta(a_t|s_t)$:

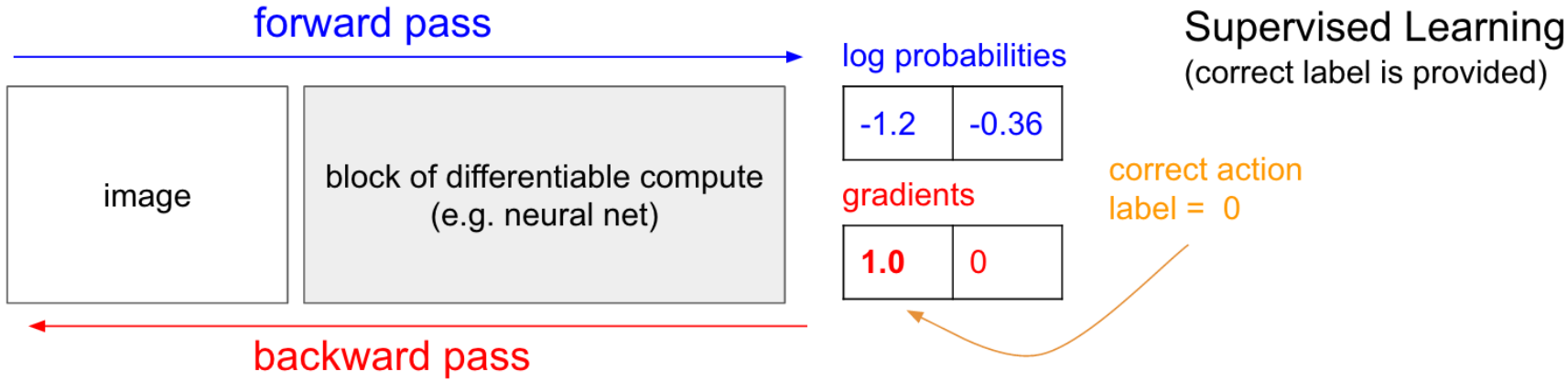
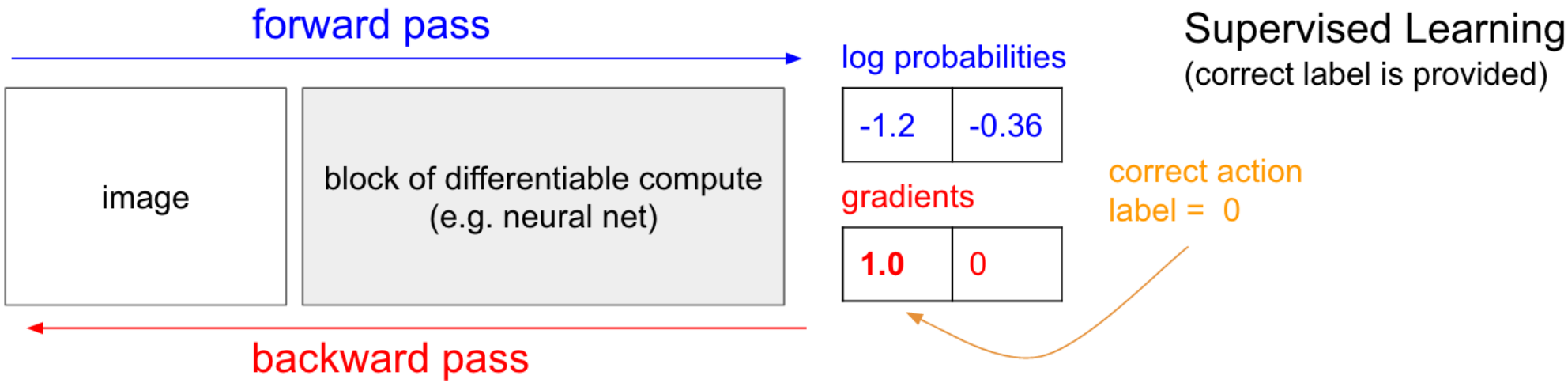


Image Credit: Andrej Karpathy, Deep Reinforcement Learning: Pong from Pixels

Neural Network Update



- Need to create a ‘pseudo label’ corresponding to the action that was taken.
- Backpropagated gradient is scaled by return R.

Overview of REINFORCE assignment

Assignment 1:

- **Objective**: Understand and implement REINFORCE update

1. Implement the REINFORCE loss function: $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R$
2. Implement the 'pseudo labels' needed to train REINFORCE in a neural network context
3. Verify that REINFORCE trains successfully on Cartpole environment.

Bias and Variance

REINFORCE

- Ideally we want a *low variance, low bias* estimator
- We know that REINFORCE has low bias (unbiased)
- However, variance in returns is often quite high
- How can we lower the variance of the gradient estimate?

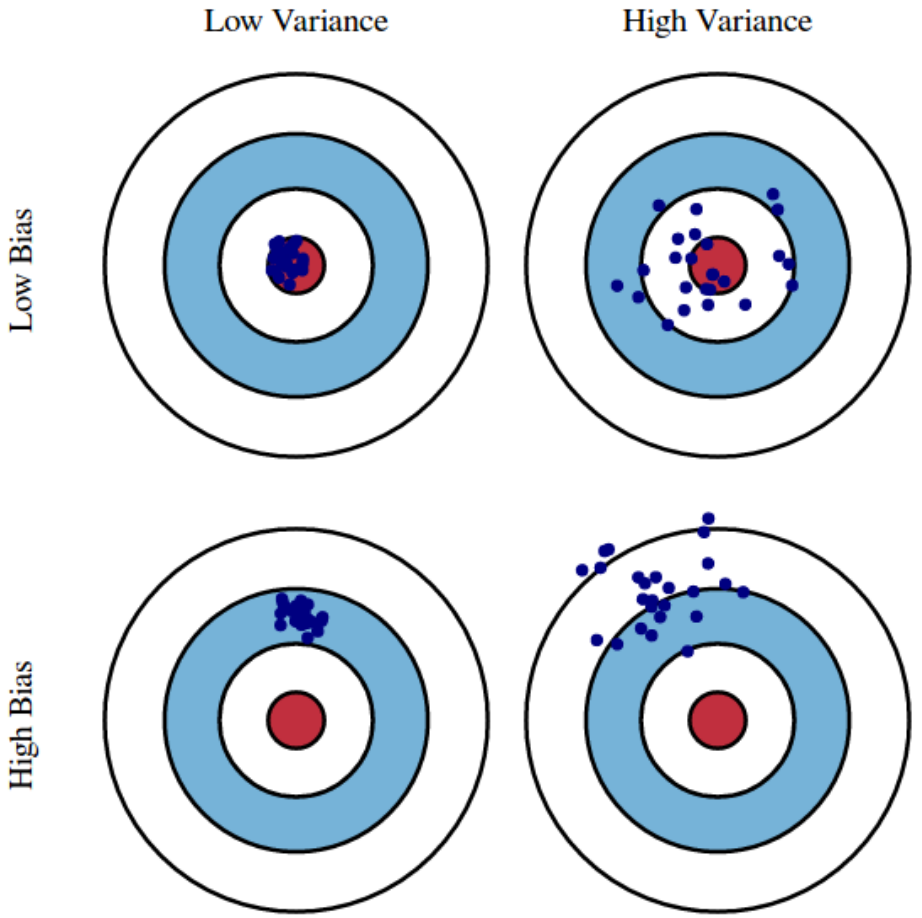


Image Credit: Scott Fortmann-Roe, Understanding the Bias-Variance Tradeoff, 2012

Variance Reduction through discounted return

Variance Reduction: Discounted Return

- The return ($R = \sum_{t=0}^{T} r_t$), which scales the gradient, sums all rewards for the episode, ignoring that fact that rewards at each step are correlated to actions taken.
- Consider a trajectory that starts with bad actions resulting in negative reward, then chooses good, highly-reward actions.
- Assuming the overall return $R = 0$, policy gradients won't learn anything from this trajectory.
- However, if we compute return at each timestep, we can recognize that later actions should be encouraged.

Variance Reduction: Discounted Return

- **Define** a per-step discounted Return $\mathrm{R}_t = \sum_{t'=t}^{T} \gamma^{t'} r_{t'}$ where γ is the standard RL discount factor. Then:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'} r_{t'}$$

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

- This trajectory will now encourage the good actions and discourage the bad ones.

Variance Reduction through baseline

Variance Reduction: Baseline

- We can further reduce variance by keeping an estimate of the expected performance of the policy. This estimate is called a baseline $\mathrm{b_t} = E[R_t; \theta]$

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b_t)$$

- Using a baseline we can discourage trajectories that are below the baseline even if their returns are positive & encourage trajectories above the baseline even if their returns are negative.

Variance Reduction: Baseline

- Possible choices of baselines:
  - Constant: $b = \frac{1}{m}\sum_{i=1}^{m} R(\tau^i)$
  - Time Dependent: $b_t = \frac{1}{m}\sum_{i=1}^{m}\sum_{t'=t}^{T} r_{t'}^{i}$
  - State Varying: $b_t(s_t) = E\left[\sum_{t'=t}^{T} r_{t'} \middle| s_t\right]$
    - This is actually just a value function:

$$V^{\pi}(s_t) = E_{\pi}[\, r_t + r_{t+1} + \ \ldots + r_T \mid s_t]$$

Value function estimation

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t))$$

How to estimate V_ϕ?

1. Collect trajectories $\tau_1, \tau_2, \ldots, \tau_m$
2. Regress V_ϕ towards empirical return:

$$\phi_{i+1} = \arg\min_\phi \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T}(V_\phi^\pi(s_t^i) - \sum_{t'=t}^{T} r_{t'}^i)^2$$

Baselined Reinforce

Policy Gradient Recap

1. Initialize policy π_θ and value function V_ϕ
2. For episode 1,2,…,n do:
    1. Collect trajectory τ by running current policy
    2. For each step in τ compute the discounted return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
    3. Re-fit the baseline minimizing $\left\| V_\phi(s_t) - R_t \right\|^2$ over all steps
    4. Update π_θ using gradient $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t))$

Baselined Reinforce Assignment

Assignment 2

- **Objective**: Implement a baselined version of REINFORCE

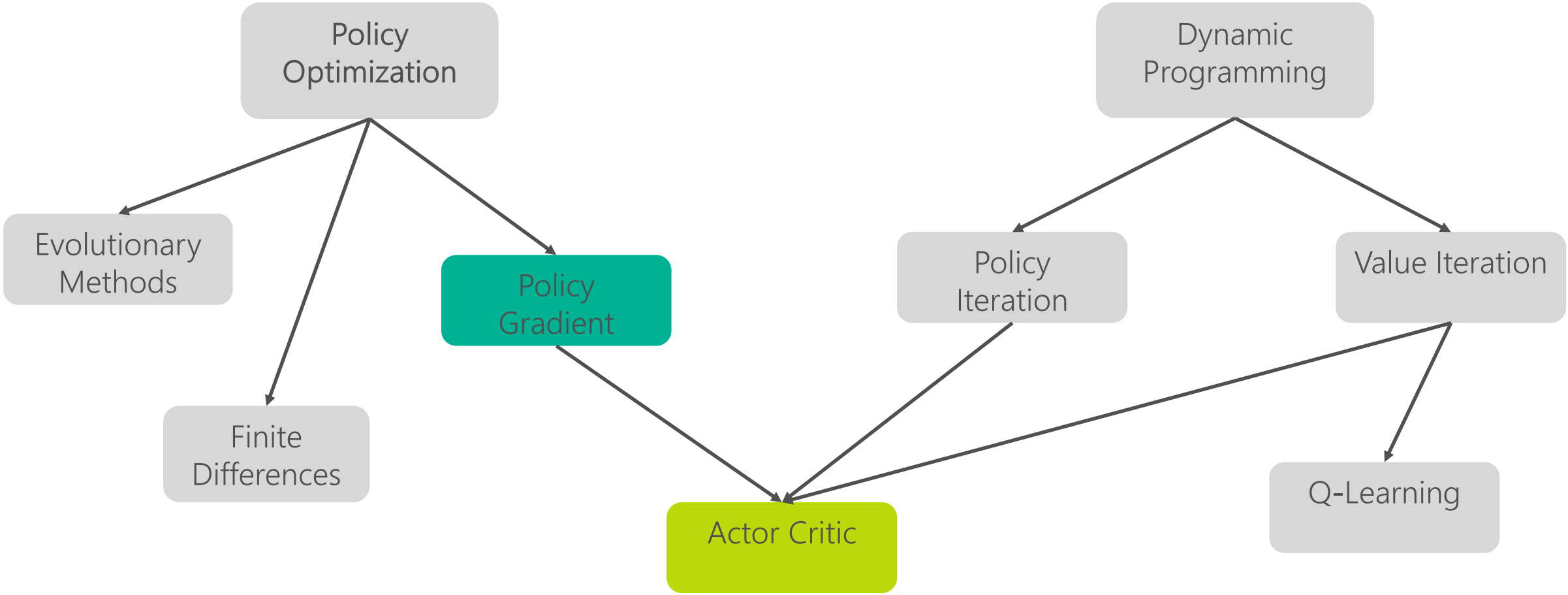
1. Define a critic network that computes $V_\phi(s_t)$
2. Build an associated Trainer and Loss function
3. Train REINFORCE using the baselined rewards:
   $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t))$
4. Verify that the variance of the estimator is less than normal REINFORCE. Variance of baselined REINFORCE should be <1000, vs ~2500 for REINFORCE.

Additional Readings

1. “Deep Reinforcement Learning: Pong from Pixels” by Andrej Karpathy; http://karpathy.github.io/2016/05/31/rl/
2. “Policy Gradient Methods” by Jan Peters; http://www.scholarpedia.org/article/Policy_gradient_methods
3. “Simple statistical gradient-following algorithms for connectionist reinforcement learning” by Williams, 1992

Introduction to Actor-Critic

Relationship to other RL methods



Actor Critic

Actor-critic algorithms consist of two components:

Actor: The policy responsible for taking actions in the environment.

Critic: Responsible for evaluating the quality of the selected actions and suggesting directions for improvement.

Actor Critic

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t))$$

Actor

Critic

- Our likelihood ratio PG estimator can be considered Actor-Critic since it uses a separate policy and value function.
- However there are still high variance parts of the Critic that aren't learned, namely R_t.
- How can we reduce the variance of R_t?

Actor Critic

- Recall the definition of a Q-Value Function:

$$Q^{\pi}(s_t, a_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T]$$

- Compare to the definition of discounted reward used previously:

$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

- We can express our PG estimator as:

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(R_t - V_\phi(s_t) \right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(Q(s_t, a_t) - V_\phi(s_t))$$

Advantage Functions

Actor Critic

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(Q(s_t, a_t) - V_\phi(s_t))$$

- The difference between a Q-Value and a state Value is called an advantage: $A = Q(s_t, a_t) - V(s_t)$
- $V^\pi(s)$ is expected return from state using policy π. $Q^\pi(s, a)$ is expected return from taking action a in state s and following π thereafter.
- The advantage tells how much more reward we got from a_t, compared to the action π would normally select.

Actor Critic

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

There are many possible ways to estimate A:

- Maintaining separate Q, V networks and compute:
$$A = Q(s_t, a_t) - V(s_t)$$
- However maintaining a separate Q-Network $Q(s_t, a_t)$ can add bias.
- Instead we will use a single network $V(s_t)$ for both.

n-step Q-Value Estimation

N-step Q-Value Estimation

Goal: Estimate $Q(s_t, a_t)$ without a separate network.

The empirical Q-Value estimate has high variance, low bias:

$$Q^{\pi}(s_t, a_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^T r_T]$$

A middle path is to use a n-step Q-Value estimate:

$$Q^{\pi}(s_t, a_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n V(s_{t+n})]$$

2-step truncated estimate (medium variance, medium bias):

$$Q^{\pi}(s_t, a_t) = E[r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})]$$

1-step truncated estimate (low variance, high bias):

$$Q^{\pi}(s_t, a_t) = E[r_t + \gamma V(s_{t+1})]$$

A3C

Asynchronous Advantage Actor-Critic (A3C)

- State-of-the-art method for Deep Reinforcement Learning
- Uses 5-step Q-Value estimation
- Shares parameters between the actor and critic networks
- A3C runs multiple copies of the environment simultaneously
- Single policy acting & collecting experience from the parallel environments.
- Eliminates the need for a replay memory (like DQN)

"Asynchronous methods for Deep Reinforcement Learning" by Vlad Mnih, 2016

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Overview of n-step update assignment

Assignment 3

- **Objective**: Implement the A3C n-step update

1. Complete the n-step-update function to compute a n-step target $Q^{\pi}(s_t, a_t) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n V(s_{t+n})]$
2. Check that the variance of the update is smaller than Baselined REINFORCE (assignment 2).
3. Experiment with different values of n. How does the variance of the update change as a function of n?

Note: this is not full A3C due to lack of parallelism.

Conclusions

Conclusions

- We have shown how to directly optimize policies using only observed rewards
- We outlined a neural network update for the REINFORCE algorithm
- Introduced the concepts of bias and variance
- Demonstrated several methods of variance reduction including baselining and n-step updates
- Creating better estimators is an active area of research