Reinforcement Learning

Lecture 7
Policy Gradient

Stergios Christodoulidis

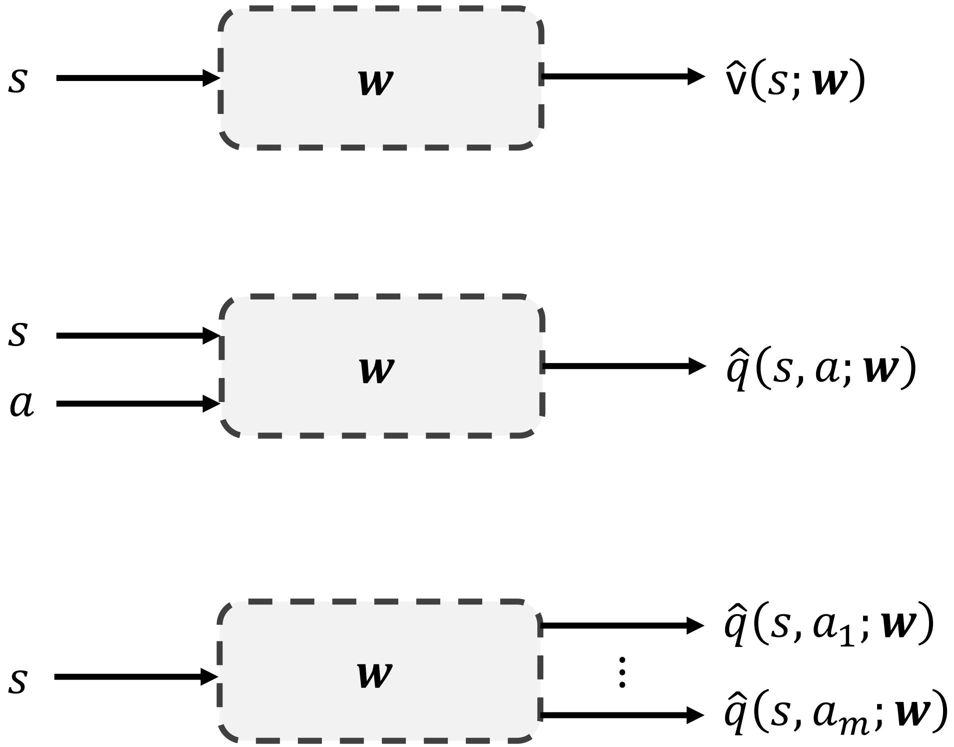
MICS Laboratory
CentraleSupélec
Université Paris-Saclay

https://stergioc.github.io/

CentraleSupélec

Last Lecture

Types of Value Function Approximation

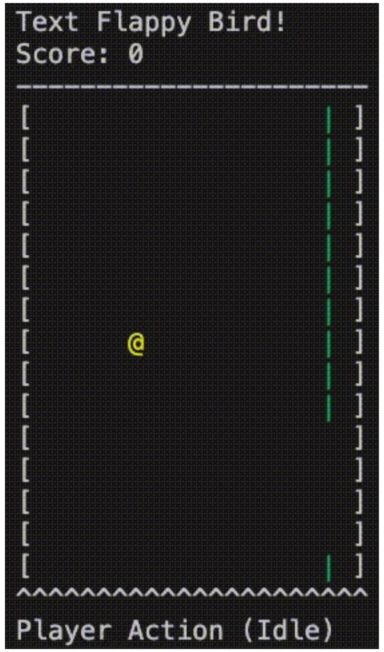


Feature Vectors

- Represent state by a feature vector

$$\varphi(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

- E.g. Polynomials, Fourier Basis
- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess



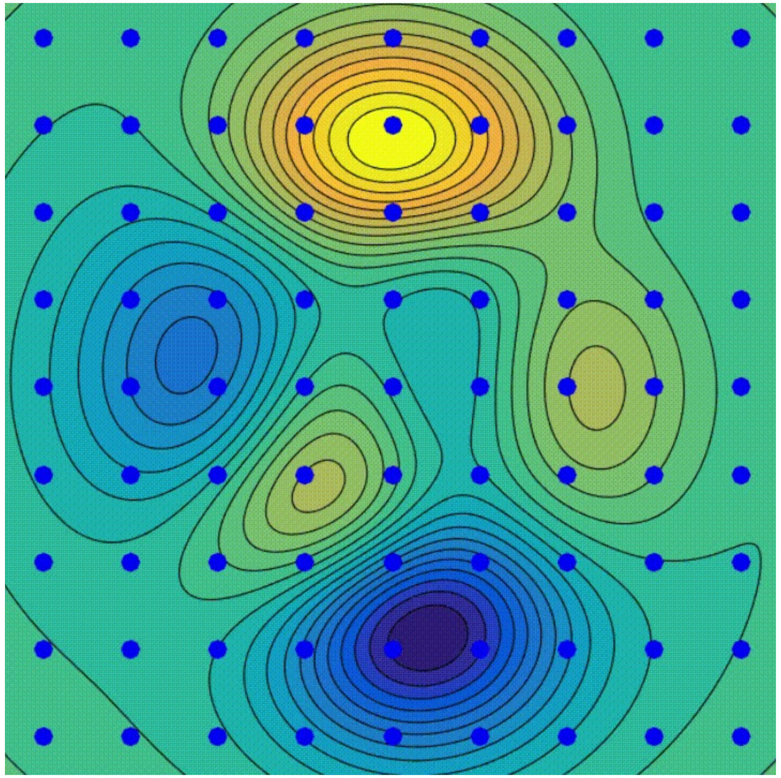
Gradient Descent

- Let $J(\boldsymbol{w})$ be a differentiable function of parameter vector $\boldsymbol{w}$
- Define the gradient of $J(\boldsymbol{w})$ to be

$$\nabla_w J(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{w})}{\partial w_n} \end{pmatrix}$$

- To find a local minimum of $J(\boldsymbol{w})$
- Adjust w in direction of negative gradient

$$\Delta\boldsymbol{w} = -\frac{1}{2} a \nabla_w J(\boldsymbol{w})$$



Incremental Prediction Algorithms

- Have assumed true value $\mathrm{v}_\pi(s)$ function given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a target for $\mathrm{v}_\pi(s)$
  - For MC, the target is the return G_t

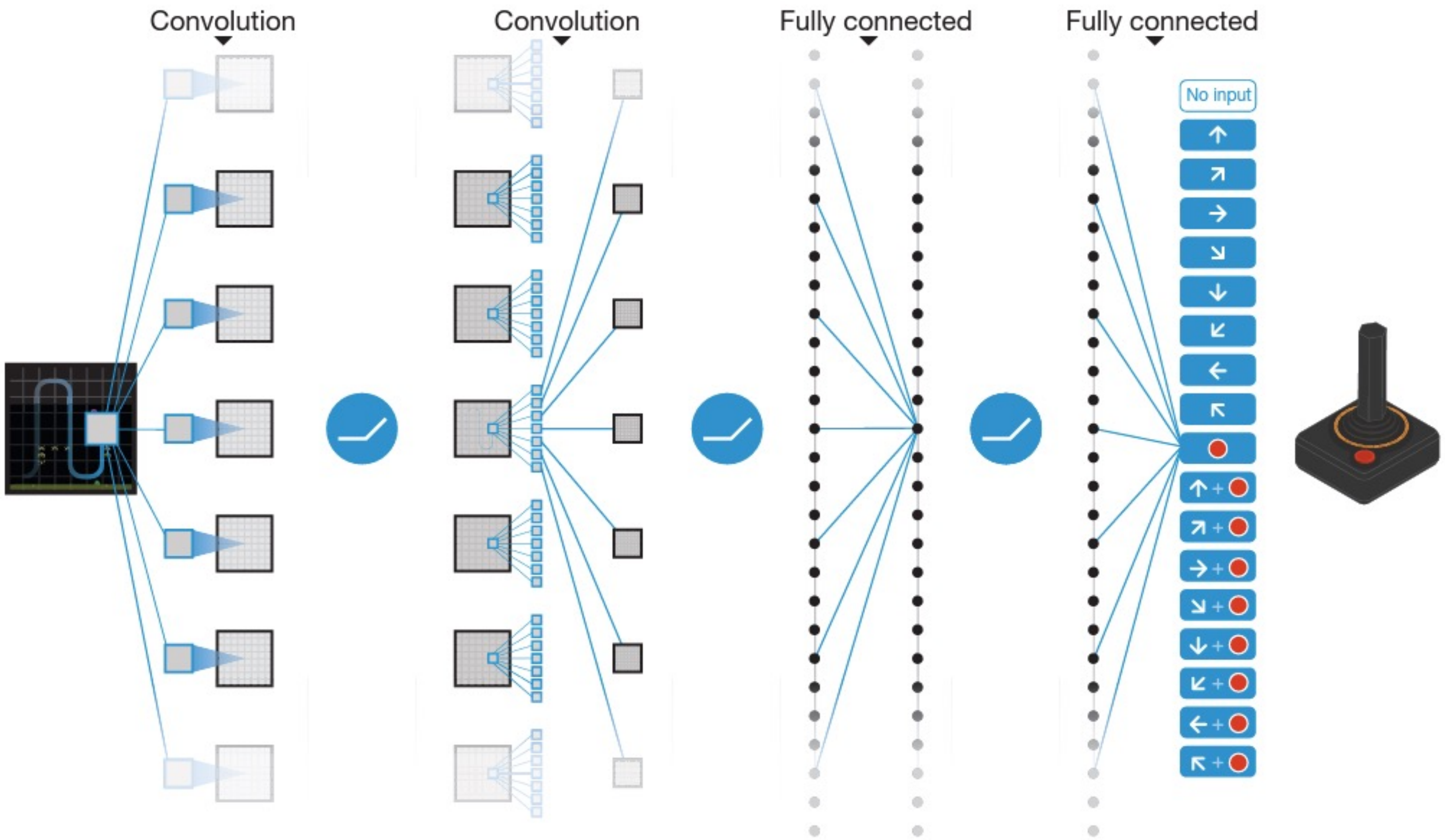
$$\Delta \boldsymbol{w} = \alpha\big(G_t - \hat{\mathrm{v}}(S_t;\boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t;\boldsymbol{w})$$

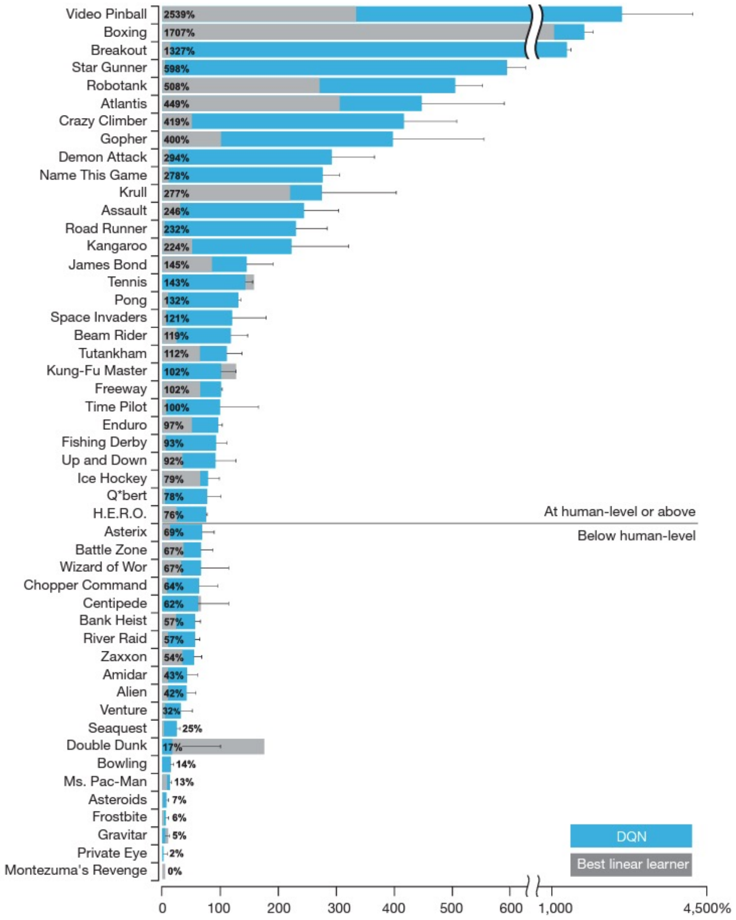
  - For TD(0), the target is the TD target $G_{t:t+1} = R_{t+1} + \gamma\, \hat{\mathrm{v}}(s_{t+1};\boldsymbol{w})$

$$\Delta \boldsymbol{w} = \alpha\big(R_{t+1} + \gamma\, \hat{\mathrm{v}}(S_{t+1};\boldsymbol{w}) - \hat{\mathrm{v}}(S_t;\boldsymbol{w})\big)\nabla_w \hat{\mathrm{v}}(S_t;\boldsymbol{w})$$

  - For TD(n), the target is the TD target $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^n \hat{\mathrm{v}}(s_{t+n};\boldsymbol{w})$

DQN in Atari





[doi:10.1038/nature14236]

Today’s Lecture

Today's Lecture

- Finite Difference Policy Gradient
- Monte-Carlo Policy Gradient
- Actor-Critic Policy Gradient

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters w,

$$\mathrm{v}_w(s) \approx \mathrm{v}_\pi(s)$$
$$\mathrm{q}_w(s,a) \approx q_\pi(s,a)$$

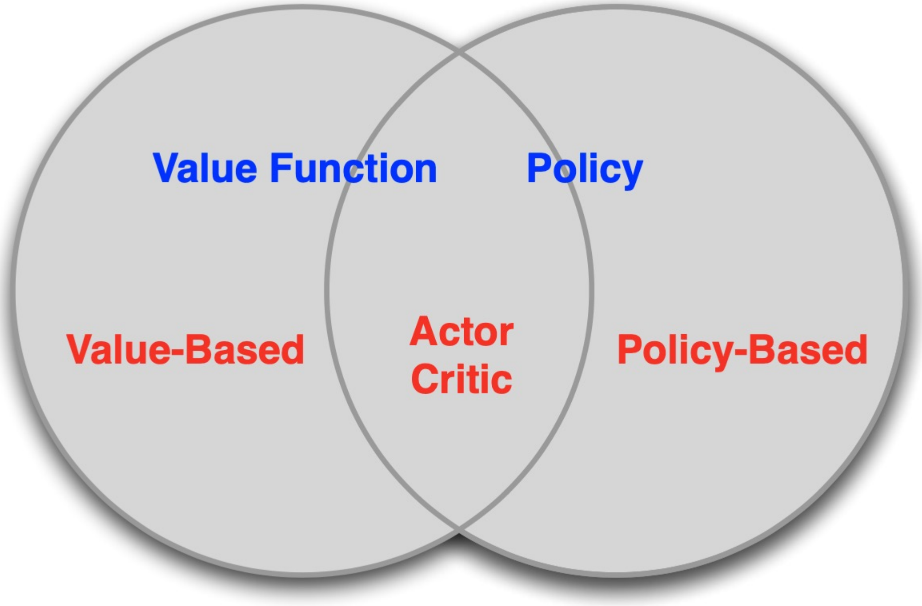
- A policy was generated directly from the value function
  - e.g., using ε-greedy
- In this lecture we will directly parametrize the policy

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s\,; \theta]$$

- We will focus again on model-free reinforcement learning

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g., ε-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



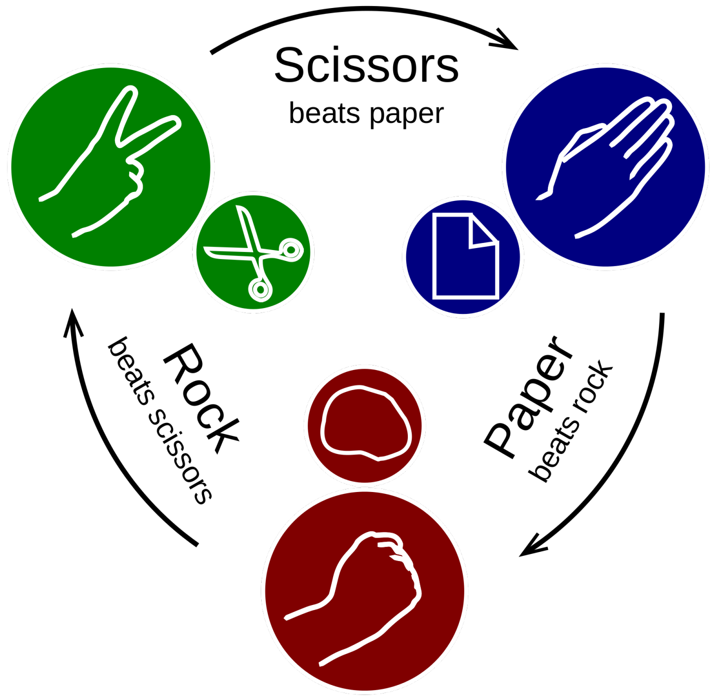
[David Silver, IRL, UCL 2015]

Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties (in contrast to value function approximation that can oscillate in some configurations)
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)



[https://en.wikipedia.org/wiki/Rock_paper_scissors]

Example: Aliased Gridworld (1/3)

- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

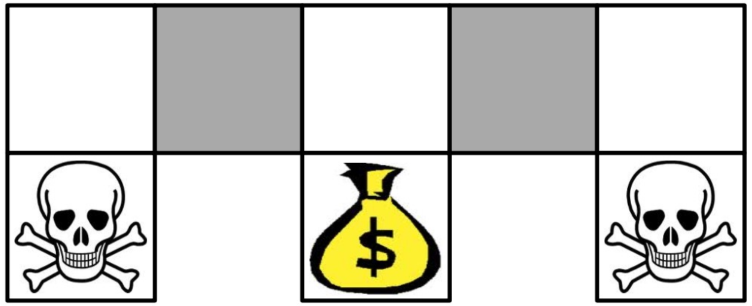
$$\varphi(s,a) = \mathbf{1}(walls\ around, a = move\ E)$$

- Compare value-based RL, using an approximate value function

$$q_w(s,a) = f(\varphi(s,a); w)$$

- To policy-based RL, using a parametrized policy

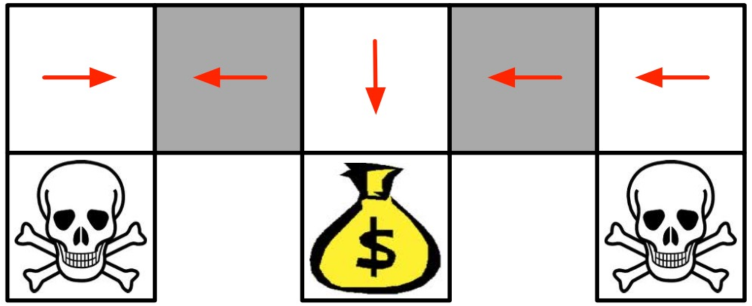
$$\pi_\theta(s,a) = g(\varphi(s,a); \theta)$$



[David Silver, IRL, UCL 2015]

Example: Aliased Gridworld (2/3)

- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and never reach the money
- Value-based RL learns a near-deterministic policy
  - e.g., greedy or ε-greedy
- So, it will traverse the corridor for a long time



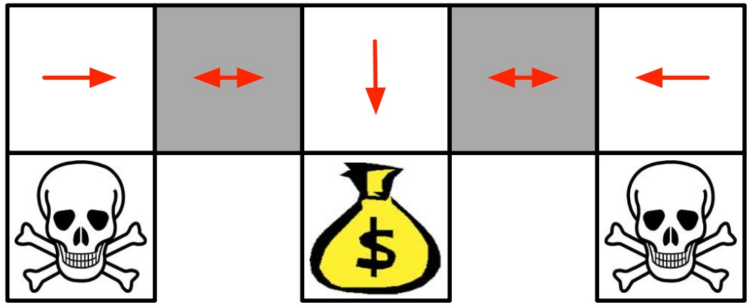
[David Silver, IRL, UCL 2015]

Example: Aliased Gridworld (2/3)

- An optimal stochastic policy will randomly move E or W in grey states

$$\pi_\theta(wall\ to\ N\ and\ S, move\ E) = 0.5$$
$$\pi_\theta(wall\ to\ N\ and\ S, move\ W) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy



[David Silver, IRL, UCL 2015]

Policy Objective Functions

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments, we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s)\, V^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a)\mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Optimization

- Policy based reinforcement learning is an optimization problem
- Find θ that maximizes $J(\theta)$
- Some approaches do not use gradient
    - Hill climbing
    - Simplex / amoeba / Nelder Mead
    - Genetic algorithms
- Greater efficiency often possible using gradient
    - Gradient descent
    - Conjugate gradient
    - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

Finite Difference Policy Gradient

Policy Gradient

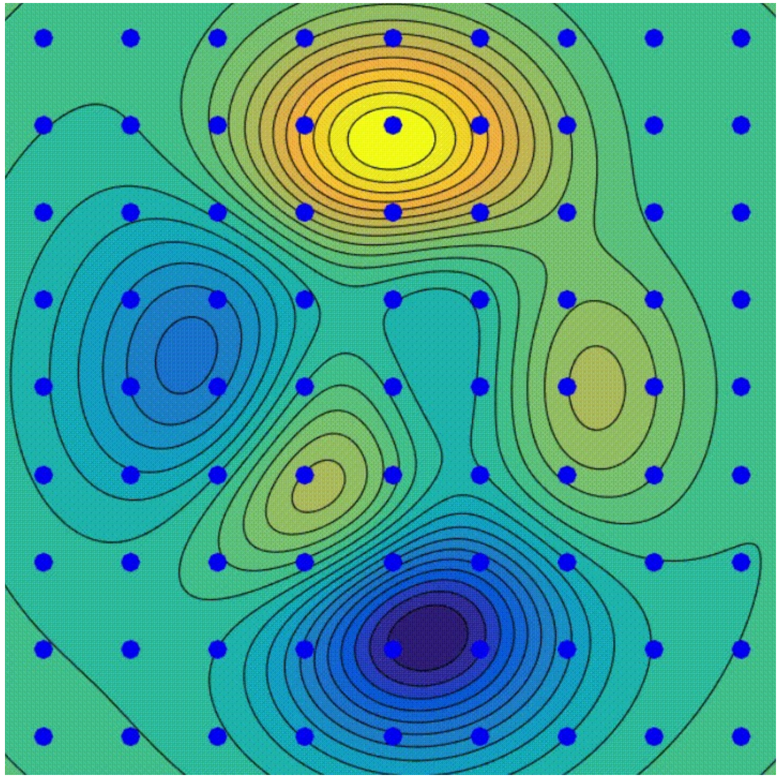
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = a\nabla_\theta J(\boldsymbol{\theta})$$

- Where $\nabla_\theta J(\boldsymbol{\theta})$ is the policy gradient

$$\nabla_\theta J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter



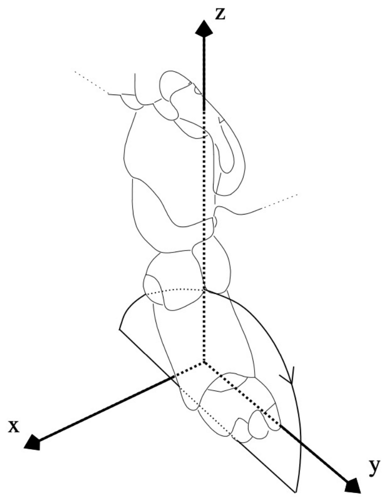
Computing Gradients By Finite Differences

- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ε in kth dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{\mathrm{J}(\theta + \varepsilon u_k) - J(\theta)}{\varepsilon}$$

  - where u_k is unit vector with 1 in kth component, 0 elsewhere
- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

Training AIBO to walk by Finite Difference Policy Gradient



- Goal: Learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controled by 12 numbers (elliptical loci)
- Adapt these parameters by finite diference policy gradient
- Evaluate performance of policy by field traversal time

[DOI: 10.1109/ROBOT.2004.1307456]

Score Function

- We now compute the policy gradient analytically
- Assume policy π_θ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following identity

$$\begin{aligned} \nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \end{aligned}$$

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Softmax Policy

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\mathrm{h}(\mathrm{s}, \mathrm{a}) = \varphi(s,a)^T\theta$, called numerical preferences.
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) = \frac{e^{\varphi(s,a)^T\theta}}{\sum_{k=1}^{N} e^{\varphi(s,a_k)^T\theta}} \propto e^{\varphi(s,a)^T\theta}$$

- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \varphi(s,a) - \mathbb{E}_{\pi_\theta}[\varphi(s,\cdot)]$$

Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \varphi(s)^T\theta$
- Variance σ^2 may be fixed , or can also parametrized
- Policy is Gaussian, $\alpha \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{\big(a - \mu(s)\big)\varphi(s)}{\sigma^2}$$

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_s^a$
- Use likelihood ratios to compute the policy gradient

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\, \mathcal{R}_s^a \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\, \nabla_\theta \log \pi_\theta(s,a)\, \mathcal{R}_s^a \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, r]
\end{aligned}
$$

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $q_\pi(s,a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem (Policy Gradient)

For any differentiable policy $\pi_\theta(s,a)$ and for any of the policy objective functions $J = J_1, J_{avR}$ or $\frac{1}{1-\gamma} J_{avV}$ the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) q_\pi(s,a)]$$

Monte-Carlo Policy Gradient

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return G_t as an unbiased sample of $q_{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)\, G_t$$

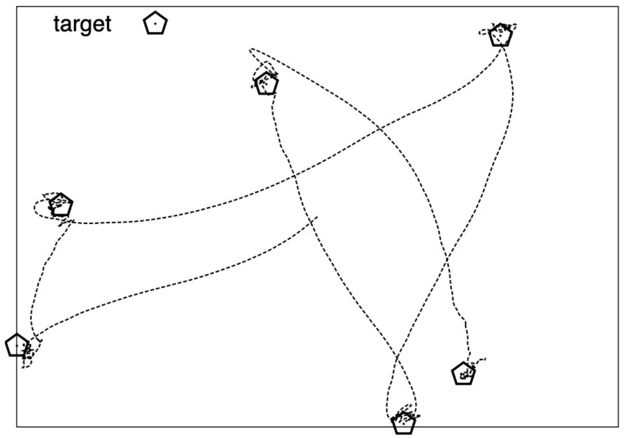
REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

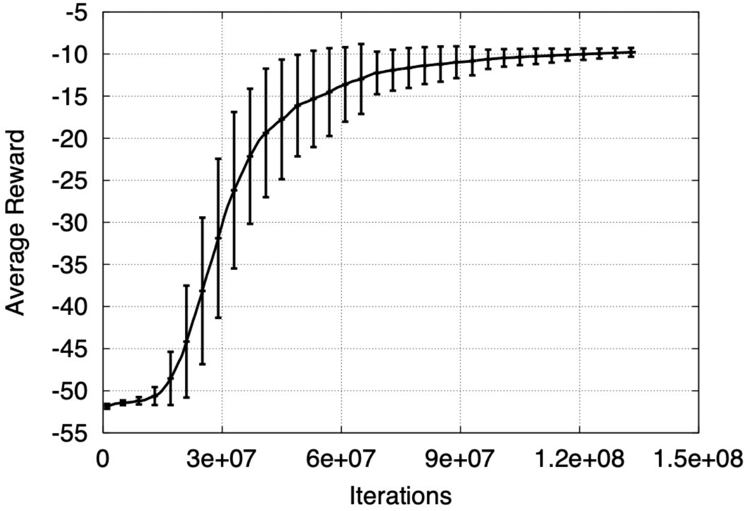
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

[An Introduction to Reinforcement Learning, Sutton and Barto]

Example: Puck World





- Continuous actions exert small force on puck
- Puck is rewarded for getting close to target
- Target location is reset every 30 seconds
- Policy is trained using a variant of Monte-Carlo policy gradient

[David Silver, IRL, UCL 2015]

Actor Critic

Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a critic to estimate the action-value function,

$$q_w(s,a) \approx q_{\pi_\theta}(s,a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) q_w(s,a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) q_w(s,a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ?
- This problem was explored in previous two lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(n)
- Could also use e.g., least-squares policy evaluation

Action-Value Actor-Critic (1/2)

- Simple actor-critic algorithm based on action-value critic
- Using linear value function approximation $q_w(s, a) = \varphi(s, a)^T w$
  - Critic Updates w by linear TD(0)
  - Actor Updates θ by policy gradient

Action-Value Actor-Critic (2/2)

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

[An Introduction to Reinforcement Learning, Sutton and Barto]

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
  - e.g., if $q_w(s, a)$ uses aliased features, can we solve gridworld example?
- Luckily, if we choose value function approximation carefully
- Then we can avoid introducing any bias
  - i.e., We can still follow the exact policy gradient

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, B(s)] = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) \sum_{a\in\mathcal{A}} \nabla_\theta\, \pi_\theta(s,a)\, \mathrm{B}(\mathrm{s})$$

$$= \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a) = 0$$

- A good baseline is the state value function $B(s) = \mathrm{v}^{\pi_\theta}(s)$
- So, we can rewrite the policy gradient using the advantage function

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\left(q^{\pi_\theta}(s,a) - \mathrm{v}^{\pi_\theta}(s)\right)\right] = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Estimating the Advantage Function (1/2)

- The advantage function can significantly reduce variance of policy gradient
- So, the critic should really estimate the advantage function
- For example, by estimating both $\mathrm{v}^{\pi_\theta}(s)$ and $q^{\pi_\theta}(s,a)$
- Using two function approximators and two parameter vectors,

$$
\begin{aligned}
\mathrm{v}_u(s) &\approx \mathrm{v}^{\pi_\theta}(s) \\
q_w(s,a) &\approx q^{\pi_\theta}(s,a) \\
A(s,a) &= q_w(s,a) - \mathrm{v}_u(s)
\end{aligned}
$$

- And updating both value functions by e.g., TD learning

Estimating the Advantage Function (2/2)

- For the true value function $\mathrm{v}^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma \mathrm{v}^{\pi_\theta}(s') - \mathrm{v}^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta} | s, a] = \mathbb{E}_{\pi_\theta}[r + \gamma \mathrm{v}^{\pi_\theta}(s') | s, a] - \mathrm{v}^{\pi_\theta}(s) = q^{\pi_\theta}(s, a) - \mathrm{v}^{\pi_\theta}(s)$$

- So, we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \delta^{\pi_\theta}]$$

- In practice we can use an approximate TD error

$$d_u = r + \gamma \mathrm{v}_u(s') - \mathrm{v}_u(s)$$

- This approach only requires one set of critic parameters u

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \mathrm{G}_t] && \text{(REINFORCE)} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \mathrm{q}_w(s,a)] && \text{(Q Actor-Critic)} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^w(s,a)] && \text{(Advantage Actor-Critic)} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta] && \text{(TD Actor-Critic)}
\end{aligned}
$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g., MC or TD learning) to estimate $q_\pi(s,a)$, $A_\pi(s,a)$ or $\mathrm{v}_\pi(s)$

Summary

- We considered methods that instead learn a parameterized policy that can select actions without consulting a value function.
- Based on the policy gradient theorem we introduced several algorithms that utilize this main idea.
- Introduced the actor-critic methods that utilize both value-function approximation as well as policy gradient methods.