Policy Gradient

I-Chen Wu

- Sutton, R.S. and Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998. (Bible for RL)
  - http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
  - Chapters 9&13
- David Silver, Online Course for Deep Reinforcement Learning.
  - http://www.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
  - Chapters 6-7

Outline

- Value Function Approximation
  - Incremental Methods
  - Batch Methods
- Policy Gradient
  - Finite Difference Policy Gradient
  - Monte-Carlo Policy Gradient
  - Actor-Critic Policy Gradient

The purpose of this chapter

- Learn policy gradient and function approximation.

Large-Scale Reinforcement Learning

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space
- How can we scale up the model-free methods for prediction and control from the last two lectures?

Value Function Approximation

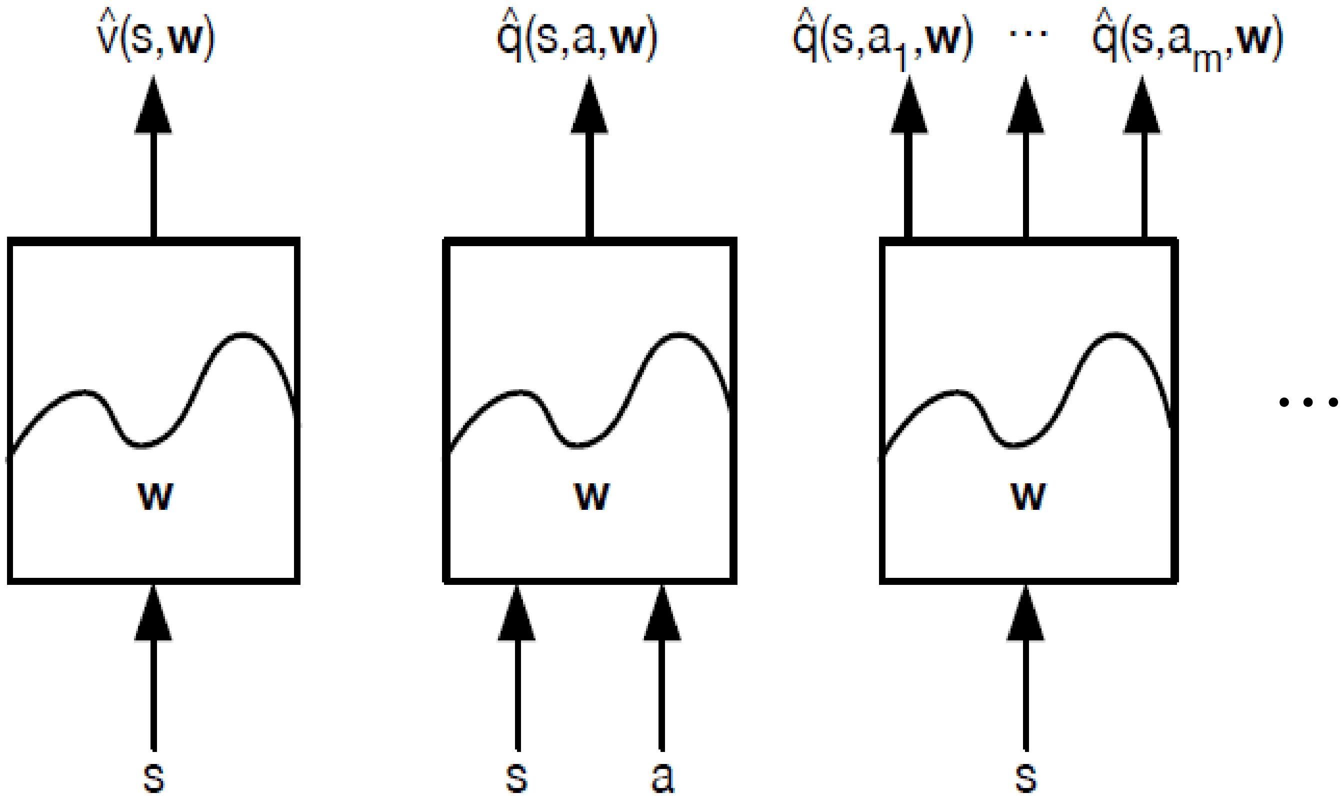
- So far we have represented value function by a lookup table
  - Every state s has an entry $V(s)$
  - Or every state-action pair s; a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, w) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, w) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states
  - Update parameter w using MC or TD learning

Types of Value Function Approximation



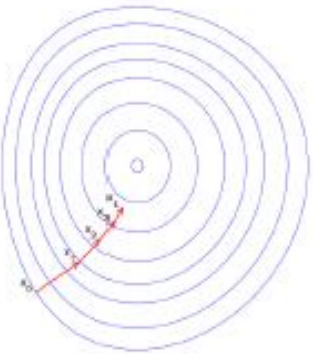
Which Function Approximator?

- There are many function approximators, e.g.
  - Linear combinations of features
    - N-Tuple Networks
  - Neural network
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - ...
- Better to consider differentiable function approximators (in red above)
- Furthermore, we require a training method that is suitable for non-stationary, non-iid data

Gradient Descent

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

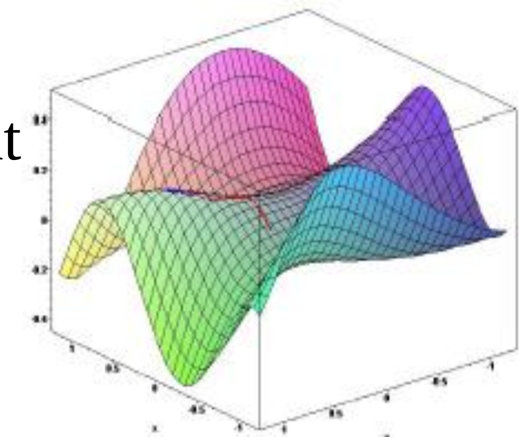
$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$



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

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

  - where α is a step-size parameter



Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector w
  - minimizing mean-squared error between approximate value function $\hat{v}(s, w)$ and true value function $v_\pi(s)$

$$J(w) = \mathbb{E}_\pi[(v_\pi(S) - \hat{v}(S, w))^2]$$

- Gradient descent finds a local minimum

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$
$$= \alpha \mathbb{E}_\pi[\big(v_\pi(S) - \hat{v}(S, w)\big)\nabla_w \hat{v}(S, w)]$$

- Stochastic gradient descent samples the gradient

$$\Delta w = \alpha\big(v_\pi(S) - \hat{v}(S, w)\big)\nabla_w \hat{v}(S, w)$$

- Expected update is equal to full gradient update

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S,w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j$$

- Objective function is quadratic in parameters *w*

$$J(w) = \mathbb{E}_\pi[(v_\pi(S) - x(S)^T w)^2]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_w \hat{v}(S,w) = x(S)$$

$$\Delta \mathrm{w} = \alpha\left(v_\pi(S) - \hat{v}(S,w)\right)x(S)$$

- Update = step-size × prediction error × feature value

Incremental Prediction Algorithms

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

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

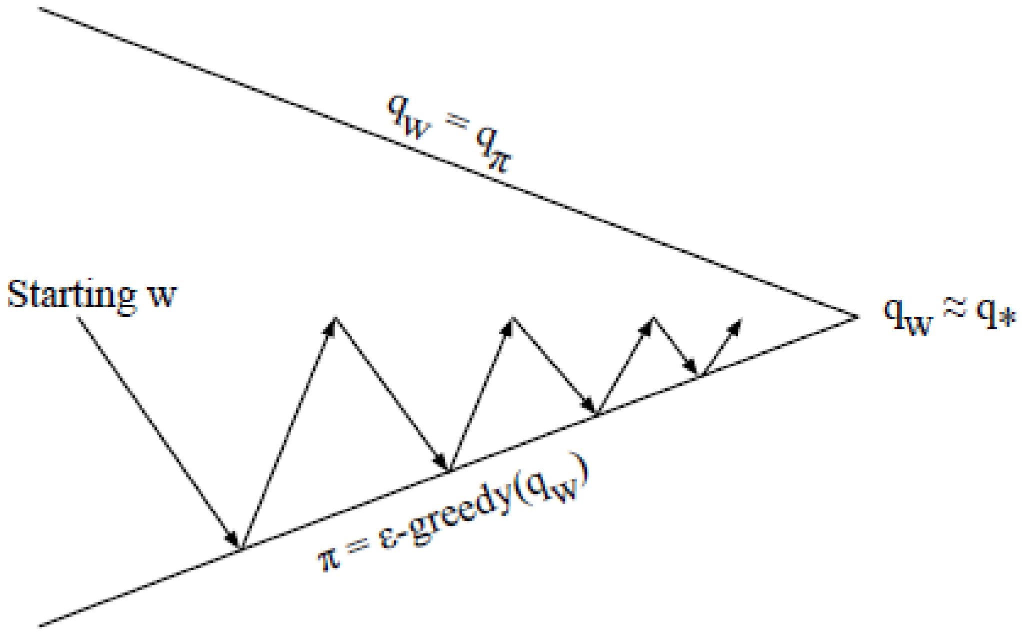
  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$

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

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathrm{w} = \alpha(G_t^\lambda - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t w)$$

Control with Value Function Approximation



- Policy evaluation
  - Approximate policy evaluation, $\hat{q}(\cdot,\cdot,w) \approx q_\pi$
- Policy improvement
  - ε-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, w) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value function $\hat{q}(S, A, w)$ and true action-value function $q_\pi(S, A)$

$$J(w) = \mathbb{E}_\pi[(q_\pi(S, A) - \hat{q}(S, A, w))^2]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_w J(w) = \left(q_\pi(S, A) - \hat{q}(S, A, w)\right)\nabla_w \hat{q}(S, A, w)$$

$$\Delta \mathrm{w} = \alpha\left(q_\pi(S, A) - \hat{q}(S, A, w)\right)\nabla_w \hat{q}(S, A, w)$$

Incremental Control Algorithms

- Like prediction, we must substitute a target for $q_\pi(S,A)$
  - For MC, the target is the return G_t

$$\Delta w = \alpha\big(G_t + \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)\big)$$
$$\nabla_w \hat{q}(S_t, A_t, w)$$

  - For forward-view TD(λ), target is the action-value λ-return

$$\Delta w = \alpha\big(q_t^\lambda - \hat{q}(S_t, A_t, w)\big)\nabla_w \hat{q}(S_t, A_t, w)$$

  - For backward-view TD(λ), equivalent update is

$$\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)$$
$$E_t = \gamma\lambda E_{t-1} + \nabla_w \hat{q}(S_t, A_t, w)$$
$$\Delta w = \alpha\delta_t E_t$$

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is not sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience (“training data”)

Least Squares Prediction

- Given value function approximation $\hat{v}(s, w) \approx v_\pi(s)$
- And experience D consisting of $\langle \text{state}, \text{value} \rangle$ pairs
$$D = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$
- Which parameters w give the best fitting value function $\hat{v}(s, w)$?
- Least squares algorithms find parameter vector w minimizing sum-squared error between $\hat{v}(s_t, w)$ and target values v_t^π,
$$LS(w) = \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, w))^2$$
$$= \mathbb{E}_D[(v^\pi - \hat{v}(s, w))^2]$$

Stochastic Gradient Descent with Experience Replay

- Given experience consisting of $\langle \text{state}, \text{value} \rangle$ pairs

$$D = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots \langle s_T, v_T^\pi \rangle\}$$

- Repeat:
  – Sample (state, value) from experience

$$\langle s, v^\pi \rangle \sim D$$

  – Apply stochastic gradient descent update

$$\Delta \mathrm{w} = \alpha(v^\pi - \hat{v}(s, w))\, \nabla_w \hat{v}(s, w)$$

- Converges to least squares solution

$$\mathrm{w}^\pi = \underset{w}{\operatorname{argmin}}\, LS(\mathrm{w})$$

  – Similar for action value function q^π

Experience Replay in Deep Q-Networks (DQN)

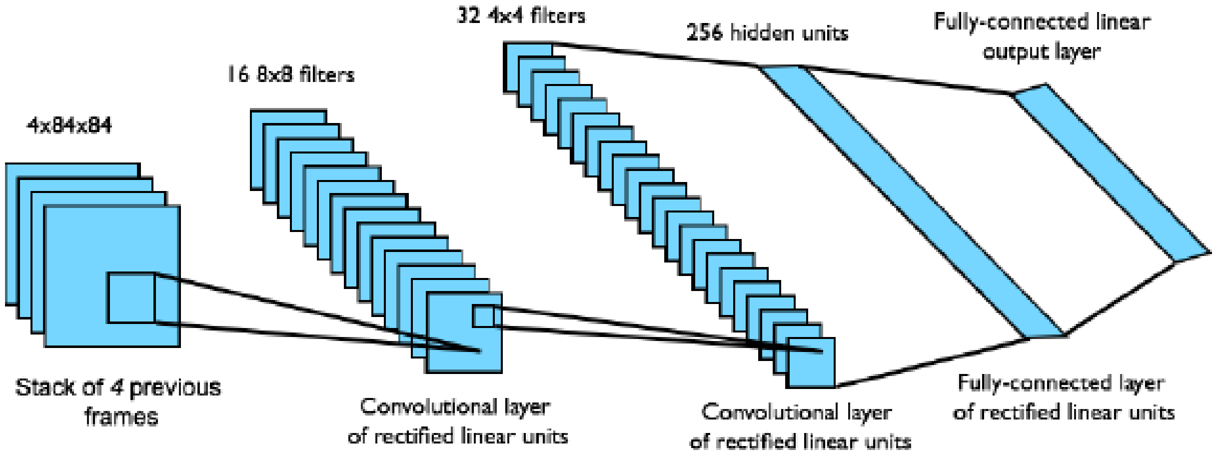
- DQN uses experience replay and fixed Q-targets
  - Take action at a_t according to ε-greedy policy
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory D
  - Sample random mini-batch of transitions (s, a, r, s') from D
  - Compute Q-learning targets w.r.t. old, fixed parameters w^-
  - Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

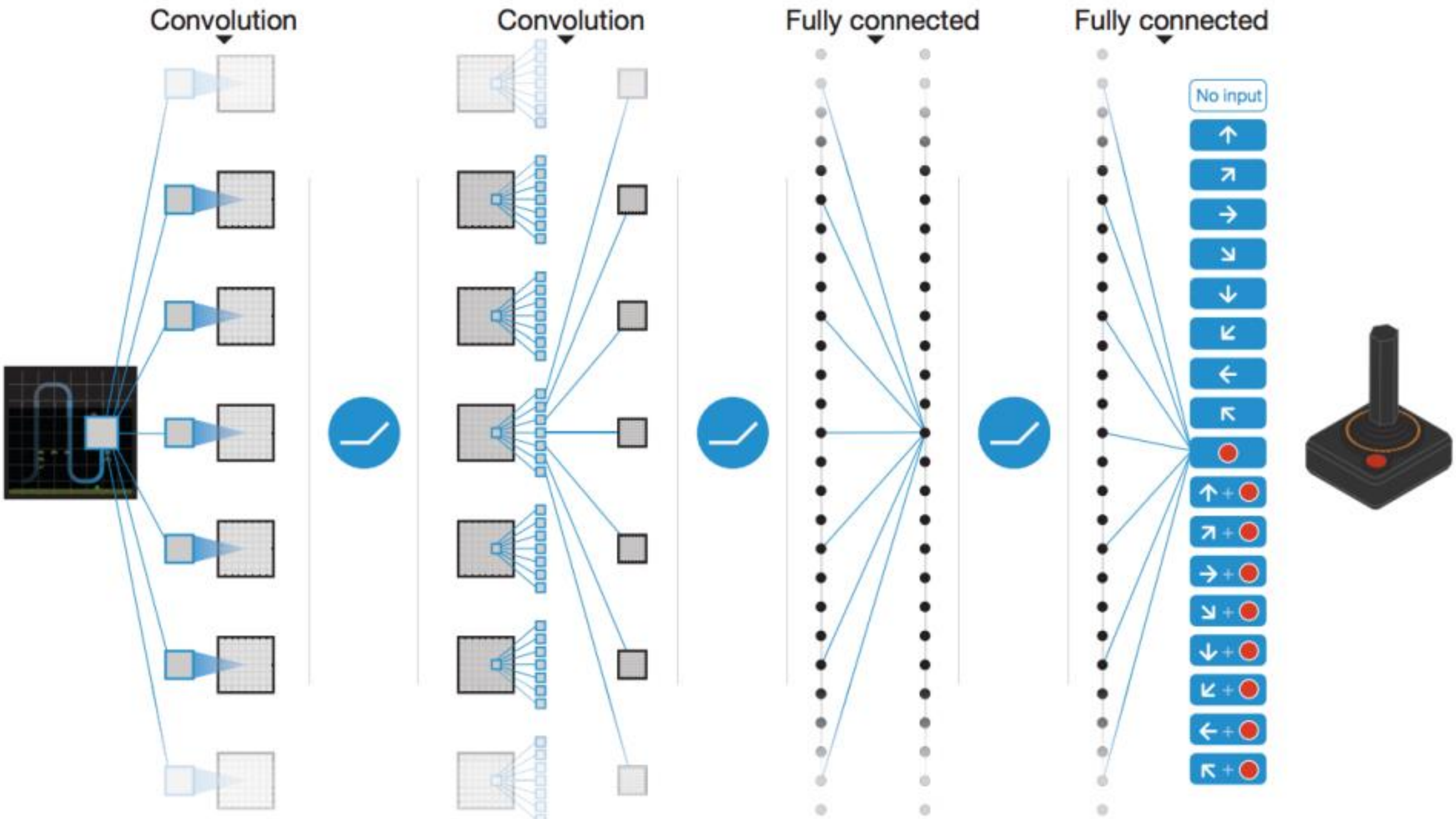
- Using variant of stochastic gradient descent

DQN in Atari

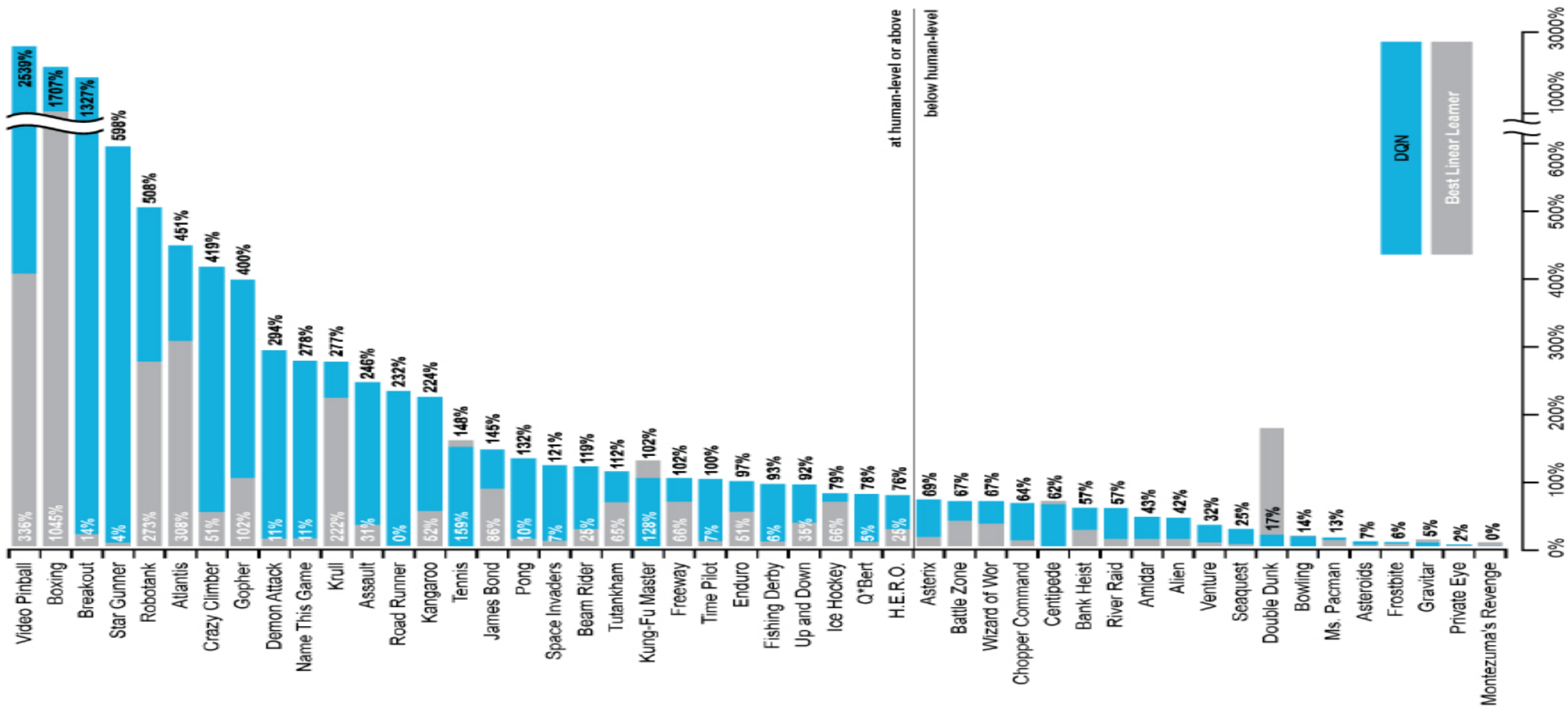
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games


Convolution
Convolution
Fully connected
Fully connected
No input

DQN Results in Atari



How much does DQN help?

	Q-learning	Q-learning +Target Q	Q-learning + Replay	Q-learning + Replay +Target Q
Breakout	3	10	241	317
Enduro	29	142	831	1006
River	1453	2868	4103	7447
Seaquest	276	1003	823	2894
Space Invaders	302	373	826	1089

Linear Least Squares Prediction

- Experience replay finds least squares solution
- But it may take many iterations
- Using linear value function approximation $\hat{v}(s, w) = \mathrm{x}(\mathrm{s})^T w$
- We can solve the least squares solution directly

Linear Least Squares Prediction (2)

- At minimum of $LS(\mathrm{w})$, the expected update must be zero

$$\mathbb{E}_{\mathcal{D}}[\Delta \mathrm{w}] = 0$$

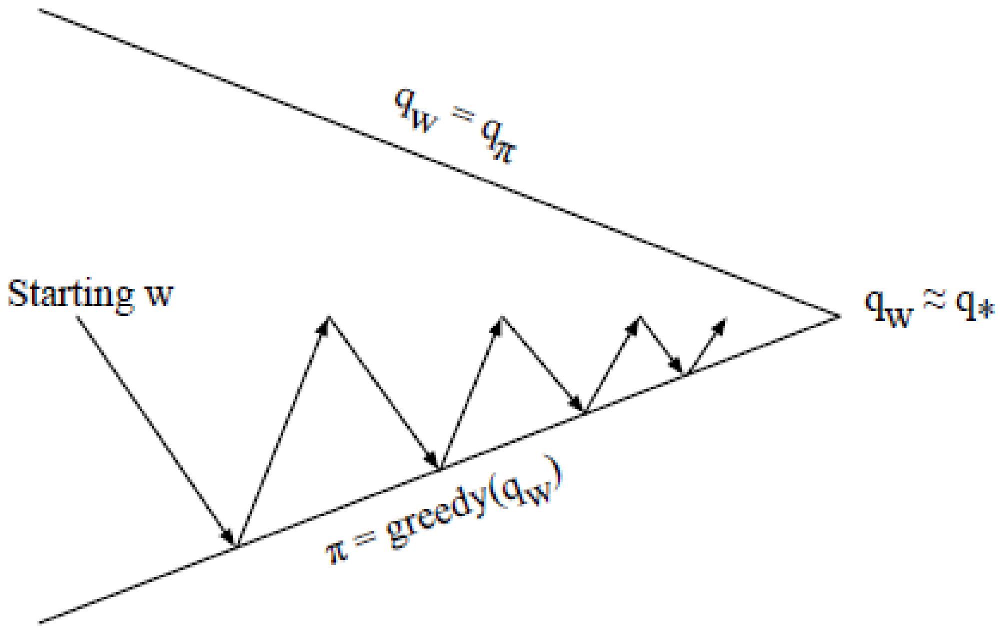
$$\alpha \sum_{t=1}^{T} x(s_t)(v_T^{\pi} - x(s_t)^T w) = 0$$

$$\sum_{t=1}^{T} x(s_t) v_T^{\pi} = \sum_{t=1}^{T} x(s_t) x(s_t)^T w$$

$$w = \left(\sum_{t=1}^{T} x(s_t) x(s_t)^T \right)^{-1} \sum_{t=1}^{T} x(s_t) v_T^{\pi}$$

- For N features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

Least Squares Policy Iteration



- Policy evaluation Policy evaluation by least squares Q-learning
- Policy improvement Greedy policy improvement

Outline

- Value Function Approximation
  - Incremental Methods
  - Batch Methods
- Policy Gradient
  - Finite Difference Policy Gradient
  - Monte-Carlo Policy Gradient
  - Actor-Critic Policy Gradient

Policy-Based Reinforcement Learning

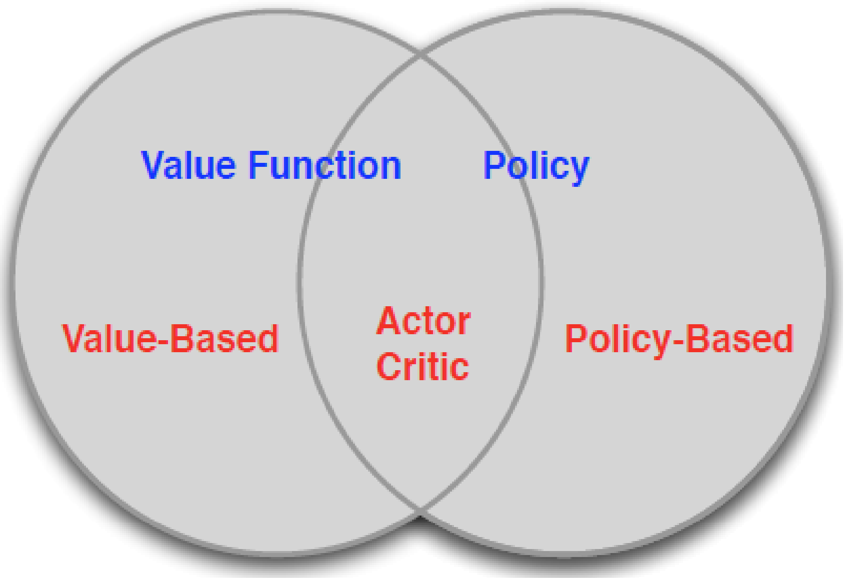
- By approximation with parameters θ, we have

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- A policy was generated directly from the value functions
  - e.g. using ε-greedy
  - This implies: the policy is also parametrized by θ.
- Here, we will directly parametrize the policy
  - Deterministic: $a = \pi_\theta(s)$, or $a = \pi(s,\theta)$
  - Stochastic: $\pi_\theta(s,a)$, $\pi_\theta(a|s)$, or $\pi(a|s,\theta)$
- 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



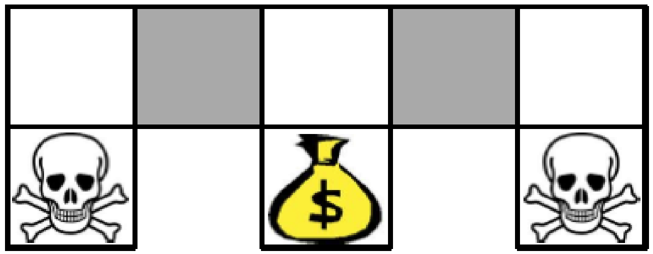
Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
  - 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)
- Hard for deterministic policy

Example: Aliased Gridworld (1)



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

$$\phi(s, a) = 1(\text{wall to N}, a = \text{move E})$$

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

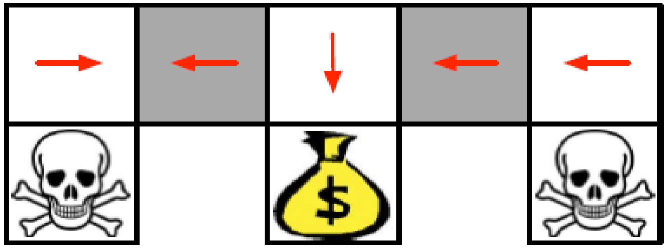
$$Q_\theta(s, a) = f(\phi(s, a), \theta)$$

- To policy-based RL, using a parametrized policy

$$\pi_\theta(s, a) = g(\phi(s, a), \theta)$$

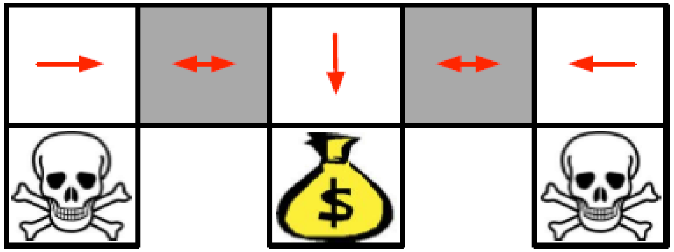
- Difficult for deterministic policy with approximator

Example: Aliased Gridworld (2)



- 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

Example: Aliased Gridworld (3)



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

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

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

Policy Objective Functions

- Goal:
  - given policy $\pi_\theta(s,a)$ with parameters θ, find best θ
    - What does the best mean?
    - 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

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

  - 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

Policy Gradient

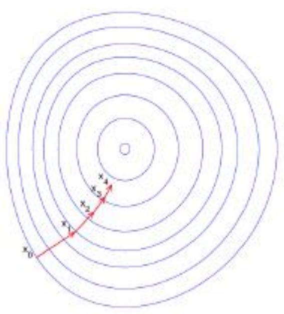
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

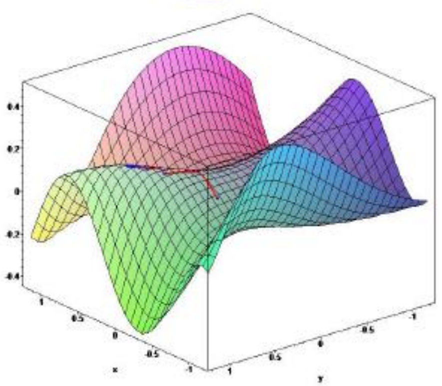
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

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

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

  - and α is a step-size parameter

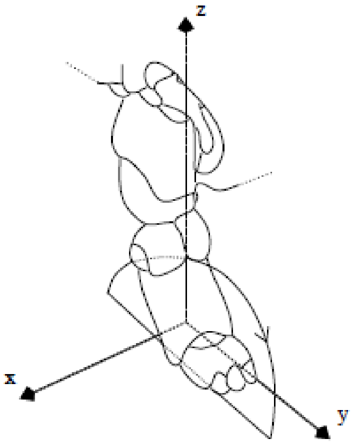




Computing Gradients By Finite Differences

- 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
$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$
    - 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 controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

Score Function

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

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

$$= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)$$

  - $\nabla_\theta \log \pi_\theta(s,a)$ is called the score function.

Softmax Policy

- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^T\theta}$$

  - Weight actions using linear combination of features $\phi(s,a)^T\theta$
- The score function is

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

- Example:
  - In Computer Go, Silver used this to solve a problem
    - Simulation Balancing

Gaussian Policy

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

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a - \mu_\theta(s))\phi(s)}{\sigma^2}$$

Score Function Gradient Estimator

- Consider an expectation $\mathbb{E}_{x\sim p(x|\theta)}[f(x)]$.
- The gradient w.r.t. θ is:

$$\nabla_\theta \mathbb{E}_x[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log p(x|\theta)]$$

  – Just sample $x_i \sim p(x|\theta)$, and compute

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$

  – Need to be able to compute and differentiate density $p(x|\theta)$ w.r.t. θ
  – This gives us an unbiased gradient estimator.
  – Note: $\pi_\theta(s,a)$ can be viewed as $p(x|\theta)$.

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 $r = R_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$

$$= \sum_{s \epsilon S} d(s) \sum_{a \epsilon A} \pi_\theta(s,a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \epsilon S} d(s) \sum_{a \epsilon A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R_{s,a}$$

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

Policy Gradient Theorem

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

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

Monte-Carlo Policy Gradient (REINFORCE)

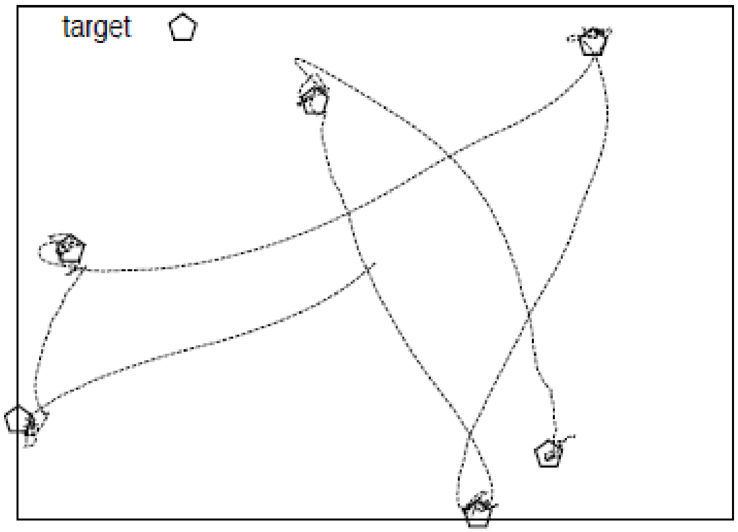
- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

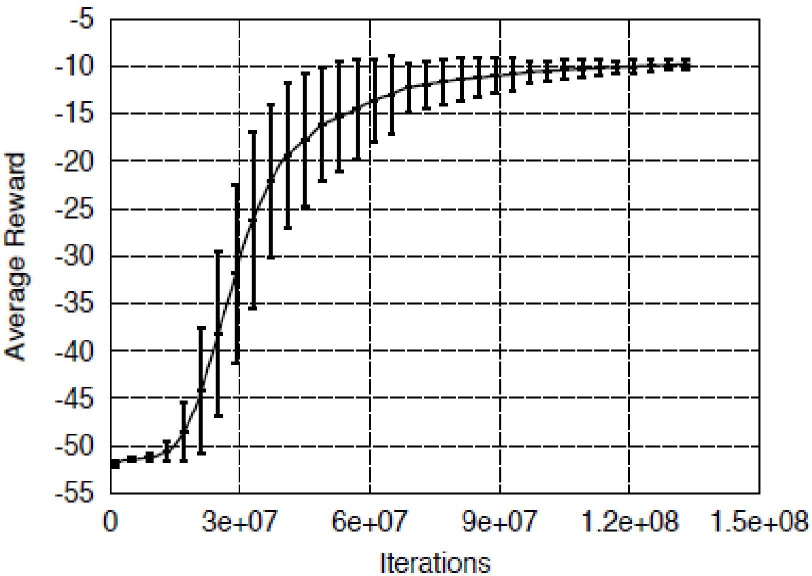
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot v_t$$

- If v_t is large, $\Delta\theta_t$ moves towards the score function more.

function REINFORCE
    Initialize θ arbitrarily
    for each episode $\{s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_t\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot v_t$
        end for
    end for
    return θ
end function

Puck World Example





- 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 variant of Monte-Carlo policy gradient

Reducing Variance Using a Critic

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

$$Q_w(s_t, a_t) \approx Q^{\pi_\theta}(s, a)$$

- 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

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)$$

Estimating the Action-Value Function

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

Action-Value Actor-Critic

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

function QAC

Initialise s, θ

Sample a $\sim \pi_\theta$

for each step **do**

Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_s^a,\cdot$

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)$

$w \leftarrow w + \beta \delta \emptyset(s, a)$

$a \leftarrow a', s \leftarrow s'$

end for

end function

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
- That is, follow the exact policy gradient (see next page)

Compatible Function Approximation

- Theorem (Compatible Function Approximation Theorem)
  - If the following two conditions are satisfied:
    - Value function approximator is compatible to the policy
      $$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$
    - Value function parameters w minimize the mean-squared error
      $$\varepsilon = \mathbb{E}_{\pi_\theta}[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2]$$
  - Then the policy gradient is exact,
    $$\nabla_\theta \ J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$
- “Compatible” means:
  - Optimizing Q_w is equal to optimizing $\log \pi_\theta$

Proof of Compatible Function Approximation Theorem

- If w is chosen to minimize mean-squared error, gradient of ε w.r.t. w must be zero,

$$\nabla_w \varepsilon = 0$$

$$\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_w Q_w(s,a)\right] = 0$$

$$\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_\theta \log \pi_\theta(s,a)\right] = 0$$

$$\mathbb{E}_{\pi_\theta}\left[Q^\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\right] = \mathbb{E}_{\pi_\theta}\left[Q_w(s,a)\nabla_\theta \log \pi_\theta(s,a)\right]$$

- So $Q_w(s,a)$ can be substituted directly into the policy gradient,

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

Reducing Variance Using a Baseline

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

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)B(s)] &= \sum_{s \epsilon S} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a)B(s) \\ &= \textstyle\sum_{s \epsilon S} d^{\pi_\theta} B(s) \cdot \nabla_\theta \sum_{a \epsilon A} \pi_\theta(s,a) \\ &= 0\end{aligned}$$

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

$$A^{\pi_\theta}(s) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

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

Estimating the Advantage Function (1)

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

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

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

Estimating the Advantage Function (2)

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}
$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$
  - is an unbiased estimate of the advantage function
$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$
- 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
$$\delta_v = r + \gamma V_v(s') - V_v(s)$$
- This approach only requires one set of critic parameters v

Critics at Different Time-Scales

- Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales From last lecture...
  - For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - V_\theta(s))\phi(s)$$

  - For TD(0), the target is the TD target $\mathrm{r} + \gamma V(s')$

$$\Delta\theta = \alpha(\mathrm{r} + \gamma V(s') - V_\theta(s))\phi(s)$$

  - For forward-view TD(λ), the target is the λ-return v_t^λ

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\phi(s)$$

  - For backward-view TD(λ), we use eligibility traces

$$\delta_v = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
$$e_t = \gamma\lambda\, e_{t-1} + \phi(s_t)$$
$$\Delta\theta = \alpha\delta_t e_t$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

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

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ v_t]$	REINFORCE
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)]$	Q Actor-Critic
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)]$	Advantage Actor-Critic
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta]$	TD Actor-Critic
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta e]$	TD(λ) Actor-Critic

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a), A^\pi(s,a)\ or\ V^\pi(s,a)$