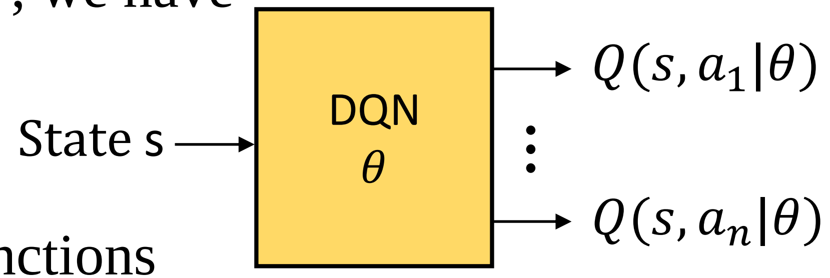
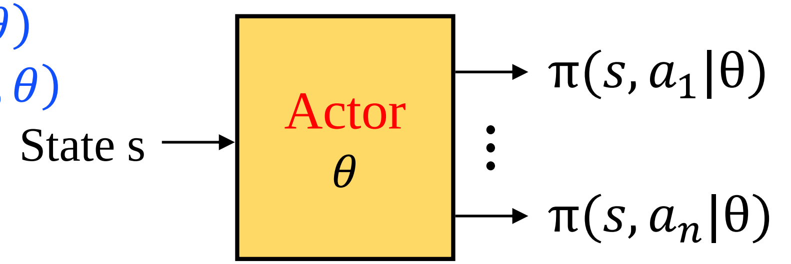
Policy Gradient

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 for value-based was generated directly from the value functions
  - e.g. using greedy or ε-greedy
  - This implies: the policy is also parametrized by θ.
- For policy-based, we directly parametrize the policy in actor
  - 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

An Example

- DQN outputs the values of actions. (Up/Down/Left/Right)
- Actor outputs the policy, probability of selecting actions.

State s → DQN θ

$$Q(s,\Uparrow|\theta) = 500$$
$$Q(s,\Downarrow|\theta) = 300$$
$$Q(s,\Leftarrow|\theta) = 600$$
$$Q(s,\Rightarrow|\theta) = 700$$

State s → Actor θ

$$\pi(s,\Uparrow|\theta) = 0.1$$
$$\pi(s,\Downarrow|\theta) = 0.1$$
$$\pi(s,\Leftarrow|\theta) = 0.3$$
$$\pi(s,\Rightarrow|\theta) = 0.5$$

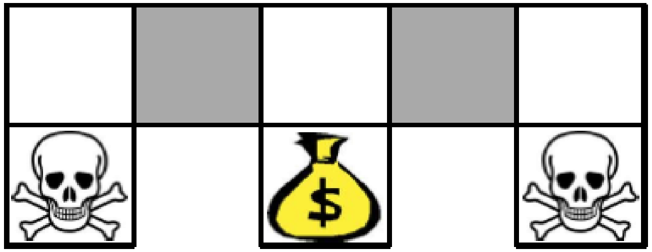
Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
    - Recall grid world with equal policy for left/up/right/down operations.
  - 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

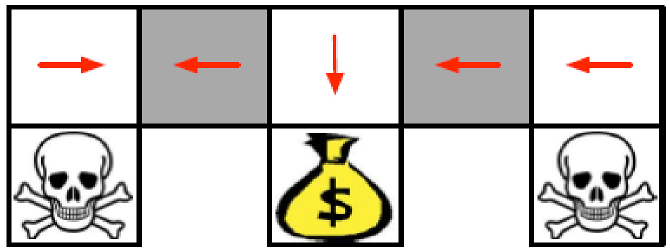
- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)
- Hard for deterministic policy

Example: Aliased Gridworld (1)



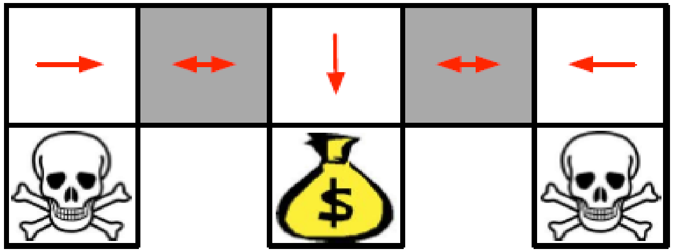
- The agent cannot differentiate the grey states, when functional approximation is used.
- Consider features of the following form (for all N, E, S, W)
$$\phi(s,a) = 1(\text{wall to N}, a = \text{move E})$$
- Compare value-based RL, using an approximate value function
$$Q_\theta(s,a) = f(\phi(s,a),\theta)$$
- 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

$$J_0(\theta) = V^{\pi\theta}(s_0) = \mathbb{E}_{\pi_\theta}[v_0]$$

- 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

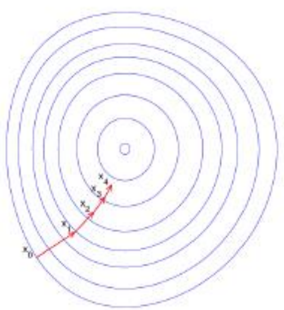
Policy Optimization

- 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

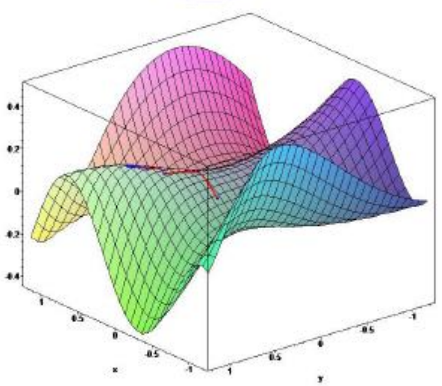
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

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



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

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



  - 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

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

    - 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

Policy Gradient (One Step)

- Consider a simple class of one-step MDPs
- Starting in state $s_0 \sim d(s)$
- Terminating after one time-step with reward $r = R_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J_0(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}[r] = \sum_{a \epsilon A} \pi_\theta(s_0, a)\, R_{s_0,a}$$

$$\nabla_\theta J_0(\theta) = \sum_{a \epsilon A} \nabla_\theta\, \pi_\theta(s_0, a)\, R_{s_0,a}$$

$$= \sum_{a \epsilon A} \pi_\theta(s_0, a) \nabla_\theta \log \pi_\theta(s_0, a) R_{s_0,a}$$

Score function

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

Let $s_0 \sim d(s)$

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

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

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

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)

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

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

  - If G_t is large, $\Delta\theta_t$ moves towards the score function more.
- Applications: Go, job-shop scheduling (hard to calculate value anyway)

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