Policy Gradients

Some Extra Complexity...

$$\tau = s_0, a_0, s_1, a_1, \ldots, s_n, a_n$$

$$R(\tau) = \sum_{i=1}^{n} \gamma^i r(s_i, a_i)$$

Policies are distributions $\pi(a_i | s_i)$

A policy π induces a distribution P_π over trajectories τ

Find π which maximizes $\mathrm{E}_{\tau \sim P_\pi}[R(\tau)]$

Policy Gradients

We can't compute gradients through the environment but we can *approximate* gradients on the *expected* return

$$\nabla_\theta P_\theta(x) = P_\theta(x) \nabla_\theta \log P_\theta(x)$$

$$\mathrm{E}_{\tau \sim P_\pi}[R(\tau)] = \sum_\tau P_\pi(\tau) R(\tau)$$

$$\nabla \mathrm{E}_{\tau \sim P_\pi}[R(\tau)] = \sum_\tau \nabla (P_\pi(\tau) R(\tau))$$

$$= \sum_\tau R(\tau) P_\pi(\tau) \nabla \log P_\pi(\tau)$$

$$= \mathrm{E}_{\tau \sim P_\pi}[R(\tau) \nabla \log P_\pi(\tau)]$$

For a more rigorous argument, see "Policy Gradient Methods for Reinforcement Learning with Function Approximation" by Sutton, McAllester, Singh, and Mansour. NeurIPS 1999.

REINFORCE

Approximate by sampling

$$E_{\tau \sim P_\pi}[R(\tau)\nabla \log P_\pi(\tau)] \approx \frac{1}{N}\sum_{\tau \sim P_\pi}[R(\tau)\nabla \log P_\pi(\tau)]$$

- Requires *a lot* of samples
  - High variance in gradient estimates
  - Rollouts cannot be reused

Baselines

- Reduce variance of gradient estimates

$$\frac{1}{N}\sum_{\tau\sim P_\pi}[R(\tau)\nabla\log P_\pi(\tau)] \rightarrow \frac{1}{N}\sum_{\tau\sim P_\pi}[(R(\tau)-b)\nabla\log P_\pi(\tau)]$$

Simplest case: Average return

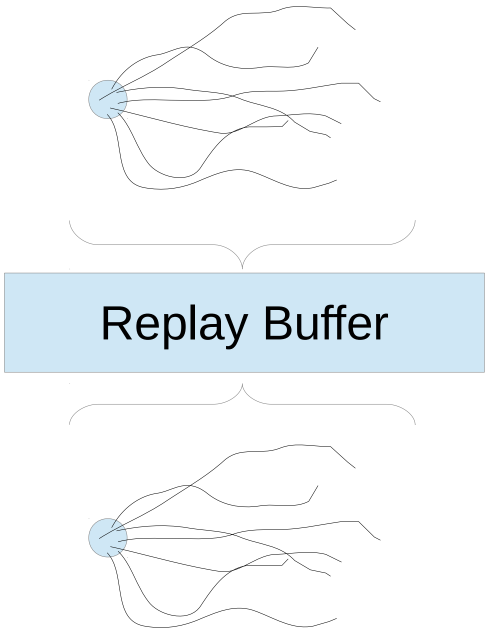
- Expected gradient estimates are the same
  - But variance is reduced

$$\mathrm{E}_{\tau\sim P_\pi}[b\nabla\log P_\pi(\tau)]=0$$

Off-Policy Algorithms

$$\frac{1}{N}\sum_{\tau \sim P_\pi}\left[R(\tau)\nabla \log P_\pi(\tau)\right] \approx \frac{1}{N}\sum_{\tau \sim Q}\left[\frac{P_\pi(\tau)}{Q(\tau)} R(\tau)\nabla \log P_\pi(\tau)\right]$$

- For some number of iterations
  - For some number of episodes
    - Collect data and store it in a replay buffer
  - Update the baseline
  - For some number of batches
    - Estimate the gradient on a sample from the replay buffer
    - Take a gradient step



Policy Gradient Algorithms

- ✔ Do not require demonstrations
- ✔ Work well in high-dimensional parameter spaces
- ✗ Are (usually) not sample efficient
- ✗ Are high-variance (though there is some work on this)

Gradient-Free Optimization

Gradient-Free Setting

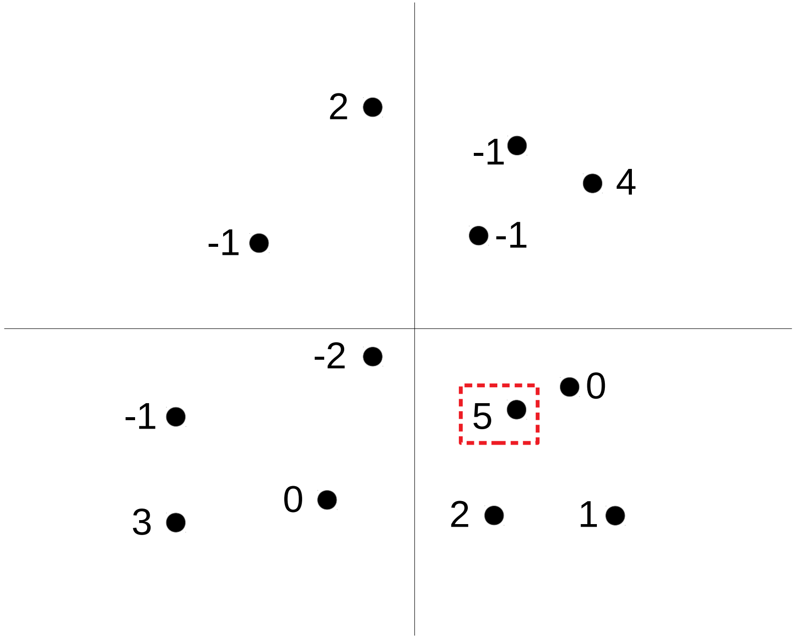
Gradients are hard, but evaluation is easy

For an evaluation function f we can compute $f(\theta)$ easily but not $\nabla_\theta f(\theta)$

Assume the evaluation function is smooth

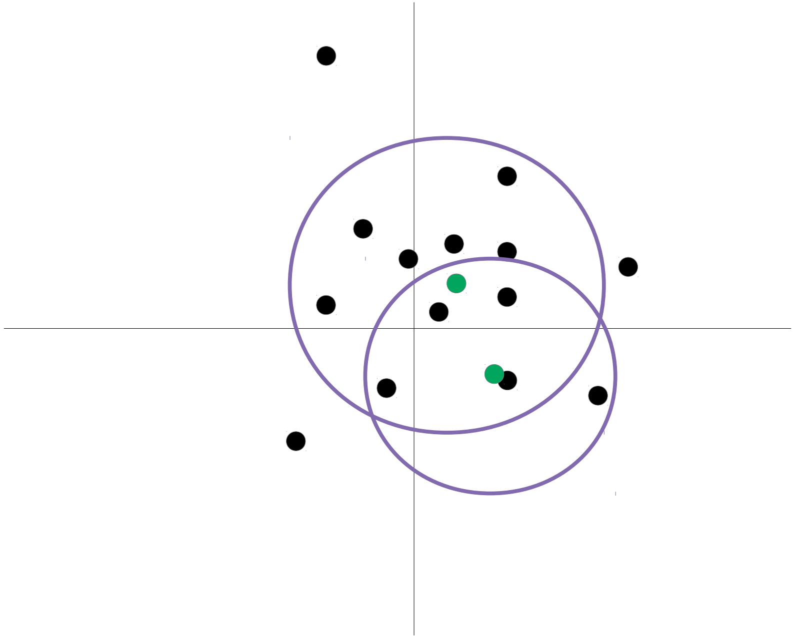
Random Search

- Randomly generate samples
- Score each one
- Choose the best



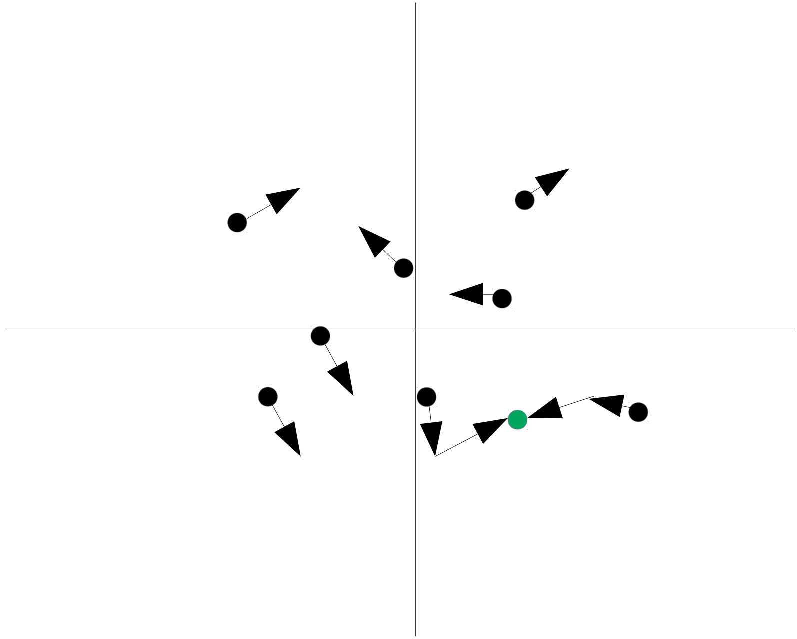
Cross Entropy Method

- Initialize μ, σ
- Loop
  - Sample $\theta_1, \ldots, \theta_n \sim N(\mu, \sigma)$
  - Compute $f(\theta_1), \ldots, f(\theta_n)$
  - Select top p% of parameters values Θ
  - Compute $\mu = \mathrm{E}[\Theta], \sigma = \sqrt{\mathrm{Var}[\Theta]}$



Evolutionary Strategies

- Initialize a population of solutions
- Loop
  - Mutate each solution
  - Evaluate the results
  - Recombine high-performing policies



Augmented Random Search

- Initialize θ
- Loop
  - Sample $\epsilon_1, \ldots, \epsilon_n \sim N(0, I)$
  - For each ϵ_i evaluate $r_i^+ = f(\theta + \nu \epsilon_i)$ and $r_i^- = f(\theta - \nu \epsilon_i)$
  - Compute $\sigma_R = \sqrt{\mathrm{Var}[\{r_1^+, \ldots, r_n^+, r_1^-, \ldots, r_n^-\}]}$
  - Update $\theta := \theta + \frac{\alpha}{n \sigma_R} \sum_{i=1}^{n} [r_i^+ - r_i^-] \epsilon_i$

Horia Mania, Aurelia Guy, Benjamin Recht. 2018. Simple Random Search Provides a Competitive Approach to Reinforcement Learning. NeurIPS 2018

Gradient-Free Optimization

- Trade sampling trajectories for sampling parameters.
- Works best in small parameter space
- Works best if there is a relatively simple correlation between parameters and returns

Open Problem: Structure vs. Data