RL for Large State Spaces: Policy Gradient

Alan Fern

RL via Policy Gradient Search

- So far all of our RL techniques have tried to learn an exact or approximate value function or Q-function
  - Learn optimal value of being in a state, or taking an action from state.

- Value functions can often be much more complex to represent than the corresponding policy
  - Do we really care about knowing Q(s,left) = 0.3554, Q(s,right) = 0.533
  - Or just that “right is better than left in state s”

- Motivates searching directly in a parameterized policy space
  - Bypass learning value function and “directly” optimize the value of a policy

Reminder: Gradient Ascent

- Given a function $f(\theta_1,\ldots,\theta_n)$ of n real values $\theta=(\theta_1,\ldots,\theta_n)$ suppose we want to maximize f with respect to θ
- A common approach to doing this is gradient ascent
- The gradient of f at point θ, denoted by $\nabla_\theta f(\theta)$, is an n-dimensional vector that points in the direction where f increases most steeply at point θ
- Vector calculus tells us that $\nabla_\theta f(\theta)$ is just a vector of partial derivatives

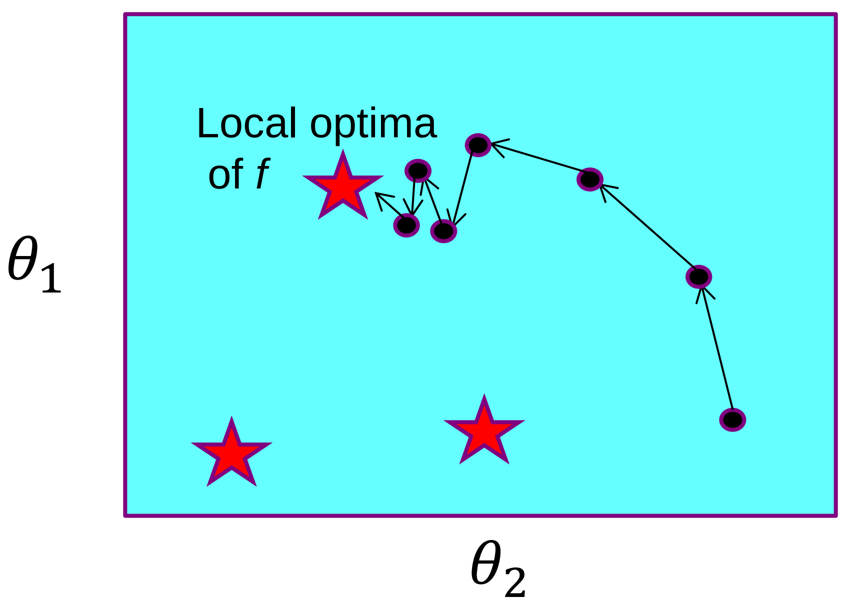
$$\nabla_\theta f(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1}, \ldots, \frac{\partial f(\theta)}{\partial \theta_n}\right]$$

where $$\frac{\partial f(\theta)}{\partial \theta_i} = \lim_{\varepsilon \to 0} \frac{f(\theta_1, \ldots \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) - f(\theta)}{\varepsilon}$$

Aside: Gradient Ascent

- Gradient ascent iteratively follows the gradient direction starting at some initial point
  - Initialize θ to a random value
  - Repeat until stopping condition

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} f(\theta)$$



With proper decay of learning rate gradient descent is guaranteed to converge to local optima.

RL via Policy Gradient Ascent

- The policy gradient approach has the following schema:
  1. Select a space of parameterized policies
  2. Compute the gradient of the value of current policy wrt parameters
  3. Move parameters in the direction of the gradient
  4. Repeat these steps until we reach a local maxima
  5. Possibly also add in tricks for dealing with bad local maxima (e.g. random restarts)

- So we must answer the following questions:
  - How should we represent parameterized policies?
  - How can we compute the gradient?

Parameterized Policies

- One example of a space of parametric policies is:

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a)$$

where $\hat{Q}_\theta(s,a)$ may be a linear function, e.g.

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- The goal is to learn parameters θ that give a good policy
- Note that it is not important that $\hat{Q}_\theta(s,a)$ be close to the actual Q-function
  - Rather we only require $\hat{Q}_\theta(s,a)$ is good at ranking actions in order of goodness

Policy Gradient Ascent

- For simplicity we will make the following assumptions:
  - Each run/trajectory of a policy starts from a fixed initial state
  - Each run/trajectory always reaches a terminal state in a finite number of steps (alternatively we fix the horizon to H)

- Let $\rho(\theta)$ be expected value of policy π_θ at initial state
  - $\rho(\theta)$ is just the expected discounted total reward of a trajectory of π_θ

- Our objective is to find a θ that maximizes $\rho(\theta)$

Policy Gradient Ascent

- Policy gradient ascent tells us to iteratively update parameters via: $$\theta \leftarrow \theta + \alpha \nabla_\theta \rho(\theta)$$

- **Problem:** $\rho(\theta)$ is generally very complex and it is rare that we can compute a closed form for the gradient of $\rho(\theta)$ even if we have an exact model of the system.

- **Key idea:** estimate the gradient based on experience

Gradient Estimation

- **<u>Concern</u>**: Computing or estimating the gradient of discontinuous functions can be problematic.

- For our example parametric policy

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a)$$

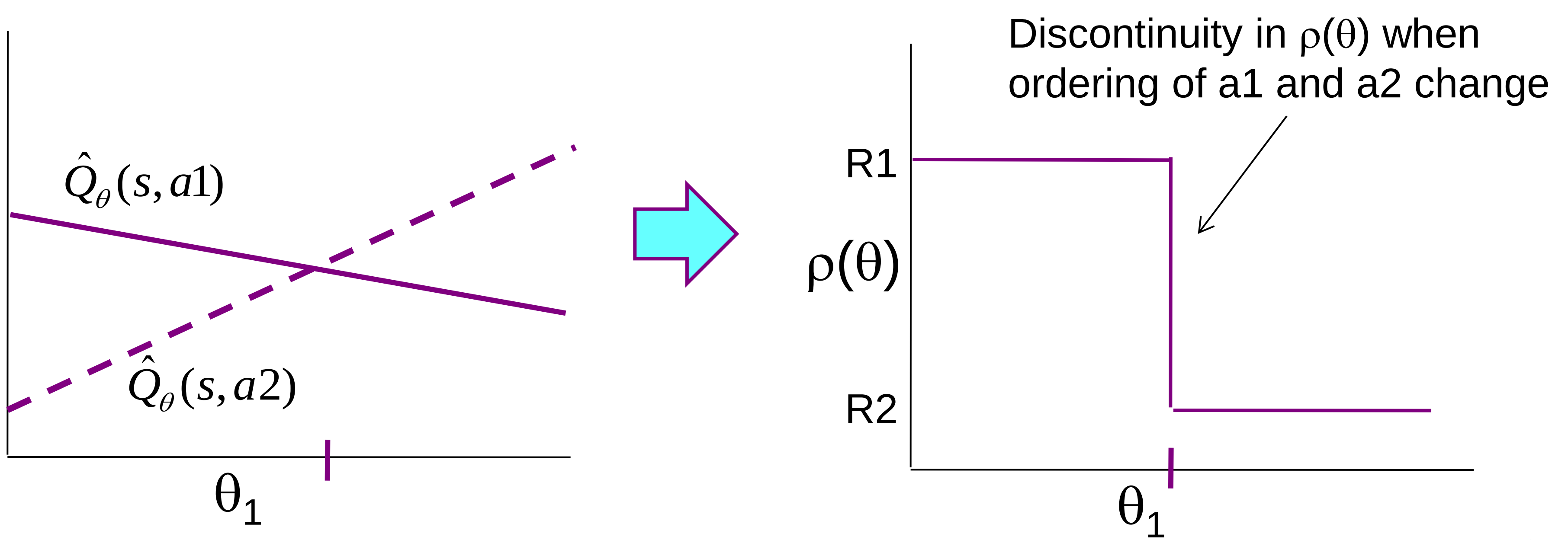
  is $\rho(\theta)$ continuous?

- No.
  - There are values of θ where arbitrarily small changes, cause the policy to change.
  - Since different policies can have different values this means that changing θ can cause discontinuous jump of $\rho(\theta)$.

Example: Discontinous $\rho(\theta)$

$$\pi_\theta(s) = \arg\max_a \hat{Q}_\theta(s,a) = \arg\max_a \theta_1 f_1(s,a) + \theta_2 f_2(s,a)$$

- Consider a problem with initial state s and two actions a1 and a2
  - a1 leads to a very large terminal reward R1
  - a2 leads to a very small terminal reward R2
- Fixing θ_2 to a constant we can plot the ranking assigned to each action by Q and the corresponding value $\rho(\theta)$



Probabilistic Policies

- We would like to avoid policies that drastically change with small parameter changes, leading to discontinuities
- A **probabilistic policy** π_θ takes a state as input and returns a distribution over actions
  - Given a state s $\pi_\theta(s,a)$ returns the probability that π_θ selects action a in s
- Note that $\rho(\theta)$ is still well defined for probabilistic policies
  - Now uncertainty of trajectories comes from environment and policy
  - Importantly if $\pi_\theta(s,a)$ is continuous relative to changing θ then $\rho(\theta)$ is also continuous relative to changing θ
- A common form for probabilistic policies is the **softmax function or Boltzmann exploration function**

$$\pi_\theta(s,a) = \Pr(a \mid s) = \frac{\exp\left(\hat{Q}_\theta(s,a)\right)}{\sum_{a' \in A} \exp\left(\hat{Q}_\theta(s,a')\right)}$$

Empirical Gradient Estimation

- Our first (naïve) approach to estimating $\nabla_\theta \rho(\theta)$ is to simply compute empirical gradient estimates
- Recall that $\theta = (\theta_1, \ldots, \theta_n)$ and $\nabla_\theta \rho(\theta) = \left[\frac{\partial \rho(\theta)}{\partial \theta_1}, \ldots, \frac{\partial \rho(\theta)}{\partial \theta_n} \right]$

  so we can compute the gradient by empirically estimating each partial derivative

$$\frac{\partial \rho(\theta)}{\partial \theta_i} = \lim_{\varepsilon \to 0} \frac{\rho(\theta_1, \ldots \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) - \rho(\theta)}{\varepsilon}$$

- So for small ε we can estimate the partial derivatives by

$$\frac{\rho(\theta_1, \ldots \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) - \rho(\theta)}{\varepsilon}$$

- This requires estimating n+1 values:

$$\rho(\theta), \quad \left\{ \rho(\theta_1, \ldots \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) \mid i = 1, \ldots, n \right\}$$

Empirical Gradient Estimation

- How do we estimate the quantities

$$\rho(\theta), \quad \{\rho(\theta_1, \ldots \theta_{i-1}, \theta_i + \varepsilon, \theta_{i+1}, \ldots, \theta_n) \mid i = 1, \ldots, n\}$$

- For each set of parameters, simply execute the policy for N trials/episodes and average the values achieved across the trials
- This requires a total of N(n+1) episodes to get gradient estimate
  - For stochastic environments and policies the value of N must be relatively large to get good estimates of the true value
  - Often we want to use a relatively large number of parameters
  - Often it is expensive to run episodes of the policy
- So while this can work well in many situations, it is often not a practical approach computationally

Likelihood Ratio Gradient Estimation

- The empirical gradient method can be applied even when the functional form of the policy is a black box (i.e. don't know mapping from θ to action distribution)

- If we know the functional form of the policy and can compute its gradient with respect to θ, we can do better.
  - Possible to estimate $\nabla_\theta \rho(\theta)$ directly from trajectories of just the current policy π_θ

- We will start with a general approach of likelihood ratio gradient estimation and then show how it applied to policy gradient.

General Likelihood Ratio Gradient Estimate

- Let F be a real-valued function over a finite domain D
  - Everything generalizes to continuous domains
- Let X be a random variable over D distributed according to $P_\theta(x)$
  - θ is the parameter vector of this distribution
- Consider the expectation of $F(X)$ conditioned on θ

$$\rho(\theta) = E\left[F(X) \mid \theta\right] = \sum_{x \in D} P_\theta(x) F(x)$$

Often no closed form for sum (|D| can be huge)

- We wish to estimate $\nabla_\theta \rho(\theta)$ given by:

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_{x \in D} P_\theta(x) F(x) = \sum_{x \in D} \left(\nabla_\theta P_\theta(x)\right) F(x)$$

General Likelihood Ratio Gradient Estimate

- Rewriting

$$\nabla_\theta \rho(\theta) = \sum_{x \in D} \left(\nabla_\theta P_\theta(x)\right) F(x)$$

$$= \sum_{x \in D} P_\theta(x) \frac{\left(\nabla_\theta P_\theta(x)\right)}{P_\theta(x)} F(x)$$

$$= \sum_{x \in D} P_\theta(x) \underbrace{\nabla_\theta \log\left(P_\theta(x)\right)}_{z_\theta(x)} F(x) = E\left[z_\theta(X) F(X)\right]$$

$X \sim P_\theta$

- So $\nabla_\theta\, \rho(\theta)$ is just the expected value of $z_\theta(X)F(X)$
  - Get unbiased estimate of $\nabla_\theta\, \rho(\theta)$ by averaging over *N* samples of *X*

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} z_\theta(x_j) F(x_j)$$

x_j is the *j*'th sample of *X*

- Only requires ability to sample *X and to compute* $z_\theta(x)$
Does not depend on how big D is!

Application to Policy Gradient

- Define $X = (s_1, a_1, s_2, a_2, \ldots, s_H)$ as sequence of *H* states and *H*-1 actions generated for single episode of π_θ
  - In general *H* will differ across episodes.
- *X* is random due to policy and environment, distributed as:

$$P_\theta(s_1, a_1, \ldots, s_H) = \prod_{t=1}^{H-1} \pi_\theta(s_t, a_t) T(s_t, a_t, s_{t+1})$$

- Define $F(X) = \sum_{t=1}^{H} R(s_t)$ as the total reward of *X*
- $\rho(\theta) = E\left[F(X) \middle| \theta\right] = E\left[\sum_{t=1}^{H} R(s_t)\right]$ is expected total reward of π_θ
  - We want to estimate the gradient of $\rho(\theta)$
  - Apply likelihood ratio method!

Application to Policy Gradient

- Recall, for random variable X we have unbiased estimate

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N}\sum_{j=1}^{N} z_\theta(x_j)F(x_j)$$

- We can generate samples of $X = (s_1, a_1, s_2, a_2, \ldots, s_H)$ by running policy π_θ from the start state until a terminal state
  - $x_i = (s_{i,1}, a_{i,1}, s_{i,2}, a_{i,2}, \ldots, s_{i,H})$ is *i*'th sampled episode
  - $F(x) = \sum_{t=1}^{H} R(s_i)$ is sum of observed rewards during i'th episode
  - $z_\theta(x) = \nabla_\theta \log\big(P_\theta(x)\big) = \nabla_\theta \log\left(\prod_{t=1}^{H-1} \pi_\theta(s_t, a_t) T(s_t, a_t, s_{t+1})\right)$

$$= \nabla_\theta \sum_{t=1}^{H-1} \left[\log\big(\pi_\theta(s_t, a_t)\big) + \log\big(T(s_t, a_t, s_{t+1})\big)\right]$$

$$= \sum_{t=1}^{H-1} \nabla_\theta \log\big(\pi_\theta(s_t, a_t)\big)$$

Does not depend on knowing model!
Allows model-free implementation.

Application to Policy Gradient

- Recall, for random variable X we have unbiased estimate

$$\nabla_\theta \rho(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} z_\theta(x_j) F(x_j)$$

- Consider a single term $z_\theta(x) F(x) = \sum_{t=1}^{H-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \sum_{k=1}^{H} R(s_k)$

- Since action at time t does not influence rewards before time t+1, we can derive the following result: (this is non-trivial to derive)

$$E\left[z_\theta(x) F(x)\right] = E\left[\sum_{t=1}^{H-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \underbrace{\sum_{k=t+1}^{H} R(s_k)}_{\text{Total reward after time t.}}\right]$$

- This justifies using a modified computation for each term:

$$\sum_{t=1}^{H-1} \nabla_\theta \log \pi_\theta(s_t, a_t) \sum_{k=t+1}^{H} R(s_k)$$

Estimate is still unbiased but generally has smaller variance.

Application to Policy Gradient

- Putting everything together we get:

$$\nabla_\theta \rho(\theta) \approx \underbrace{\frac{1}{N}}_{\text{\# of sampled trajectories of current policy}} \sum_{j=1}^{N} \sum_{t=1}^{H_j^{\text{ length of trajectory j}}} \underbrace{\left(\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t})\right)}_{\text{Direction to move parameters in order to increase the probability that policy selects } a_{jt} \text{ in state } s_{jt}} \overbrace{\sum_{k=t+1}^{H_j} R(s_{j,k})}^{\text{Observed reward after taking } a_{jt} \text{ in state } s_{jt}}$$

- **Interpretation:** each episode contributes weighted sum of gradient directions
  - Gradient direction for increasing probability of $a_{j,t}$ in $s_{j,t}$ is weighted by sum of rewards observed after taking $a_{j,t}$ in $s_{j,t}$
- Intuitively this increases/decreases probability of taking actions that are typically followed by good/bad reward sequences

Basic Policy Gradient Algorithm

- Repeat until stopping condition
  1. Execute π_θ for N episodes to get set of state, action, reward sequences
  2. $\nabla_\theta \leftarrow \frac{1}{N}\sum_{j=1}^{N}\sum_{t=1}^{H_j}\left(\nabla_\theta \log \pi_\theta(s_{j,t}, a_{j,t})\right)\sum_{k=t+1}^{H_j} R(s_{j,k})$
  3. $\theta \leftarrow \theta + \alpha \nabla_\theta$

- Unnecessary to store N episodes (use online mean estimate in step 2)
- Disadvantage: small # of updates per # episodes
  - Also is not well defined for (non-episodic) infinite horizon problems
- Online policy gradient algorithms perform updates after each step in environment (often learn faster)

Toward Online Algorithm

- Consider the computation for a single episode

$$\Delta_H = \sum_{t=1}^{H} \left(\nabla_\theta \log \pi_\theta(s_t, a_t)\right) \sum_{k=t+1}^{H} R(s_k)$$

$$= \underbrace{\sum_{t=2}^{H} R(s_t) \sum_{k=1}^{t-1} \nabla_\theta \log \pi_\theta(s_t, a_t)}_{\text{Just reorganize terms}}$$

$$= \sum_{t=2}^{H} R(s_t) z_t$$

- Notice that we can compute z_t in an online way

$$z_1 = 0; \qquad z_{t+1} \leftarrow z_t + \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- We can now incrementally compute Δ_T for each episode

$$\Delta_0 = 0; \qquad \Delta_{t+1} \leftarrow \Delta_t + R(s_{t+1}) z_{t+1}$$

Storage requirement depends only on # of policy parameters

Toward Online Algorithm

- So the overall gradient estimate can be done by incrementally computing Δ_H for each of N episodes and computing their mean
  - The mean of the Δ_H across episodes can be computed online
  - Total memory requirements still depends only on # parameters
  - Independent of length and number of episodes!

- But what if episodes go on forever?
  - We could continually maintain Δ_T (using a constant amount of memory) but we would never actually do a parameter update
  - Also Δ_T can have infinite variance in this setting (we will not show this)

- Solution:
  - Update policy parameters θ after each reward is observed (rather than simply update gradient estimate Δ_T)
  - Introduce discounting
  - Results in OLPOMDP algorithm

Online Policy Gradient (OLPOMDP)

Repeat forever

1. Observe state *s*
2. Draw action *a* according to distribution $\pi_\theta(s)$
3. Execute *a* and observe reward *r*
4. $z \leftarrow \beta z + \nabla_\theta \log \pi_\theta(s,a)$;; discounted sum of ;; gradient directions
5. $\theta \leftarrow \theta + \alpha \cdot r \cdot z$

- Performs policy update at each time step and executes indefinitely
  - This is the OLPOMDP algorithm [Baxter & Bartlett, 2000]

Interpretation

Repeat forever

1. Observe state *s*
2. Draw action *a* according to distribution $\pi_\theta(s)$
3. Execute *a* and observe reward *r*
4. $z \leftarrow \beta z + \nabla_\theta \log \pi_\theta(s,a)$;; discounted sum of ;; gradient directions
5. $\theta \leftarrow \theta + \alpha \cdot r \cdot z$

- Step 4 computes an “eligibility trace” *z*
  - Discounted sum of gradients over previous state-action pairs
  - Points in direction of parameter space that increases probability of taking more recent actions in more recent states
- For positive rewards step 5 will increase probability of recent actions and decrease for negative rewards.

Computing the Gradient of Policy

- Both algorithms require computation of

$$\nabla_\theta \log\left(\pi_\theta(s,a)\right)$$

- For the Boltzmann distribution with linear approximation we have:

$$\pi_\theta(s,a) = \frac{\exp\left(\hat{Q}_\theta(s,a)\right)}{\sum_{a' \in A} \exp\left(\hat{Q}_\theta(s,a')\right)}$$

where

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

- Here the partial derivatives composing the gradient are:

$$\frac{\partial \log\left(\pi_\theta(s,a)\right)}{\partial \theta_i} = f_i(s,a) - \sum_{a'} \pi_\theta(s,a') f_i(s,a')$$

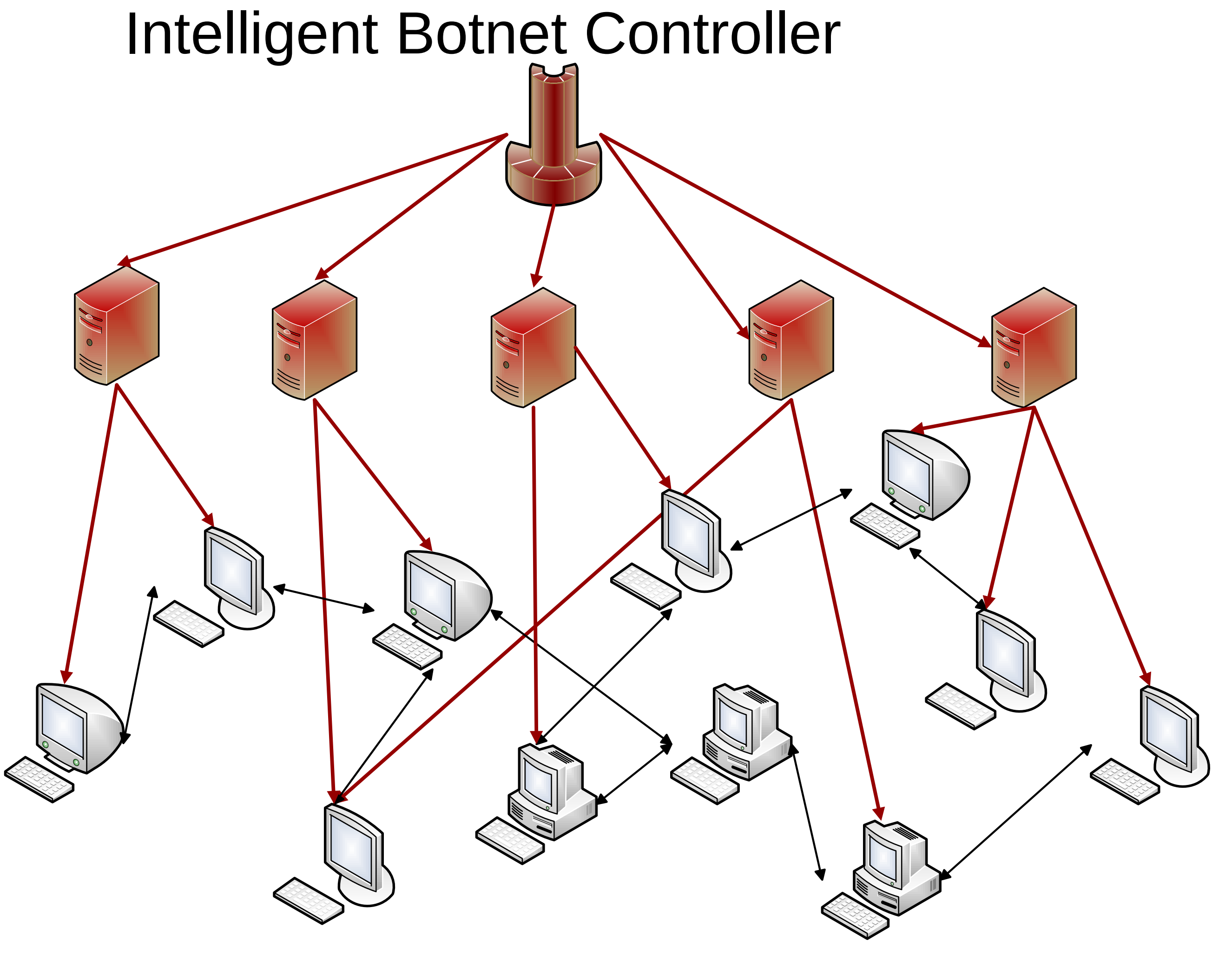
Controlling Helicopters

- Policy gradient techniques have been used to create controllers for difficult helicopter maneuvers
- For example, inverted helicopter flight.

Quadruped Locomotion

- Optimize gait of 4-legged robots over rough terrain

Proactive Security



• Used OLPOMDP to proactively discover maximally damaging botnet attacks in peer-to-peer networks

Policy Gradient Recap

- When policies have much simpler representations than the corresponding value functions, direct search in policy space can be a good idea
  - Or if we already have a complex parametric controllers, policy gradient allows us to focus on optimizing parameter settings
- For baseline algorithm the gradient estimates are unbiased (i.e. they will converge to the right value) but have high variance for large T
  - Can require a large N to get reliable estimates
- OLPOMDP can trade-off bias and variance via the discount parameter and does not require notion of episode
- Can be prone to finding local maxima
  - Many ways of dealing with this, e.g. random restarts or intelligent initialization.