Policy-based methods

Herke van Hoof

Big picture: How to learn policies

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\ldots N}$$

Policy

$\pi(\mathbf{a}|\mathbf{s})$

Thanks to Jan Peters

Big picture: How to learn policies

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1 \dots N}$$

Learn value function
$V(\mathbf{s}), Q(\mathbf{s}, \mathbf{a})$

Policy
$\pi(\mathbf{a}|\mathbf{s})$

Value-based

Optimize policy
$\pi(\mathbf{a}|\mathbf{s})$

Policy-based

Model-free RL

Learn dynamics model
$p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \; r(\mathbf{s}, \mathbf{a})$

Learn value or policy
$V(\mathbf{s}), Q(\mathbf{s}, \mathbf{a}), \pi(\mathbf{a}|\mathbf{s})$

Policy
$\pi(\mathbf{a}|\mathbf{s})$

Model-based RL

Thanks to Jan Peters

Shortcomings of action-value methods

Handle continuous actions

- How can we apply the max operator efficiently?

Ensure smoothness in policies

- Sometimes, need to ensure policy does not change too much in 1 step
- We can make the stepsize for value small. But small change in value can lead to big change in policy

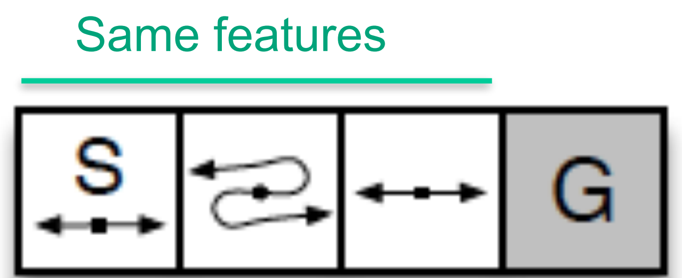
Hard to include prior knowledge about possible solutions

Can’t learn stochastic policies

Shortcomings of action-value methods

Can't learn stochastic policies

- With function approximation, states might be aliased



Sutton & Barto. Reinforcement Learning: An introduction.

- With ϵ greedy: will tend to get stuck at 'reversed' state
- Choosing either action with 50% probability would do better here
- No way to learn how much randomness is optimal
- We might want **stochastic** policies

Policy representation

Instead of optimising a value function that implicitly defines a policy; directly optimise the **parameters of a policy function**

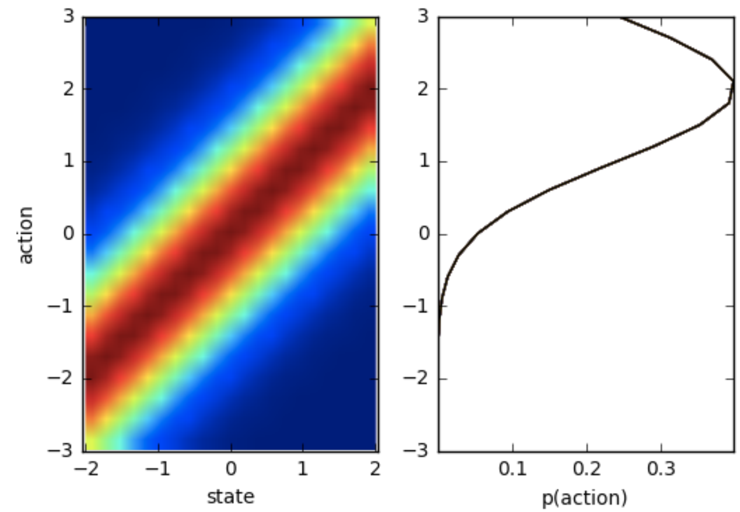
Can choose policy how we want, we often need:

- A stochastic policy, that defines a distribution over all actions
- The (log)probabilities to be differentiable wrt parameters

Policy representation

Linear gaussian policy (cont. actions)

$$\mathbf{a} \sim \mathcal{N}(\boldsymbol{\theta}^T \mathbf{s}; \sigma)$$

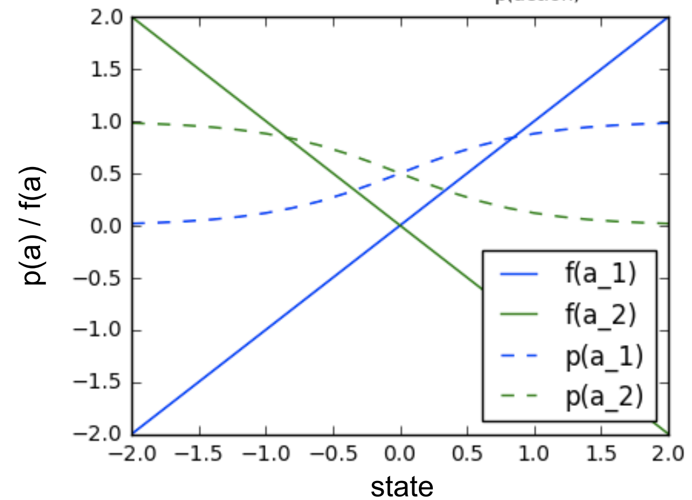


Neural network policy (cont. actions)

$$\mathbf{a} \sim \mathcal{N}(\mathrm{NN}_{\theta_\mu}(\mathbf{s}); \mathrm{NN}_{\theta_\sigma}(\mathbf{s}))$$

Softmax policy (discrete actions)

$$p(\mathbf{a}|\mathbf{s}) = \frac{\exp f_\theta(\mathbf{s}, \mathbf{a})}{\sum_{\mathbf{a}' \in \mathcal{A}} \exp f_\theta(\mathbf{s}, \mathbf{a}')}$$



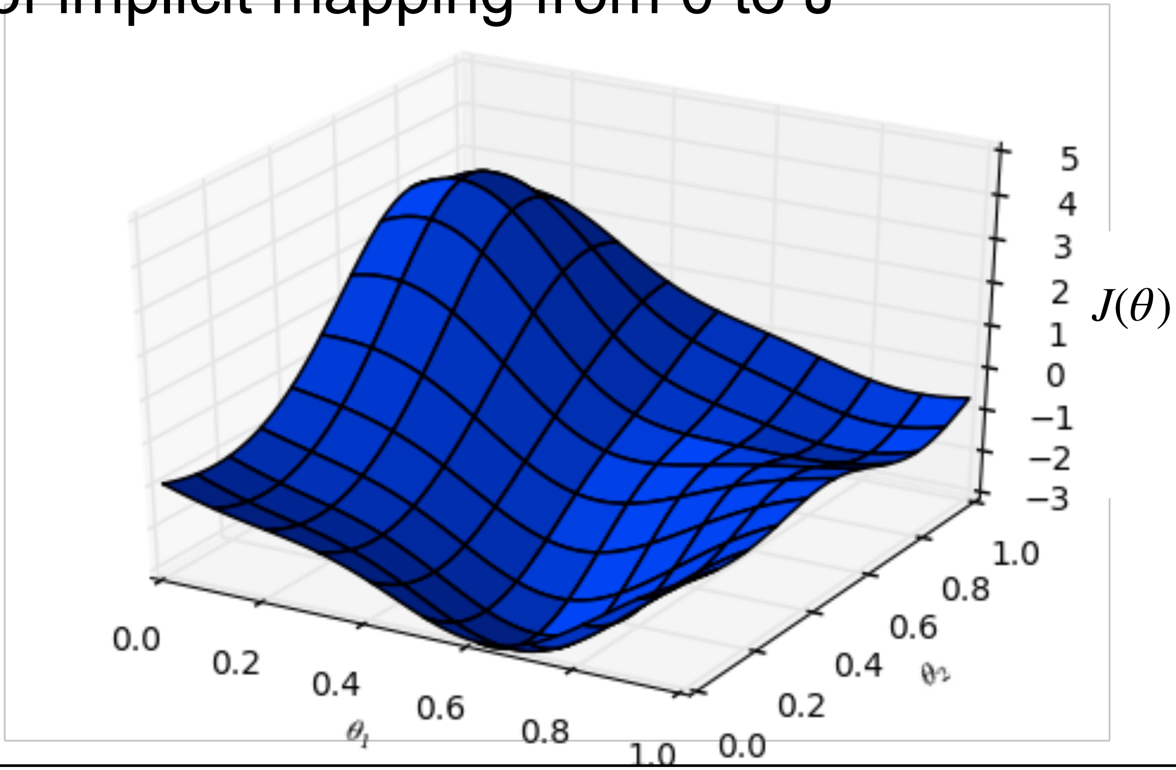
f itself can be linear, NN, …

Objective

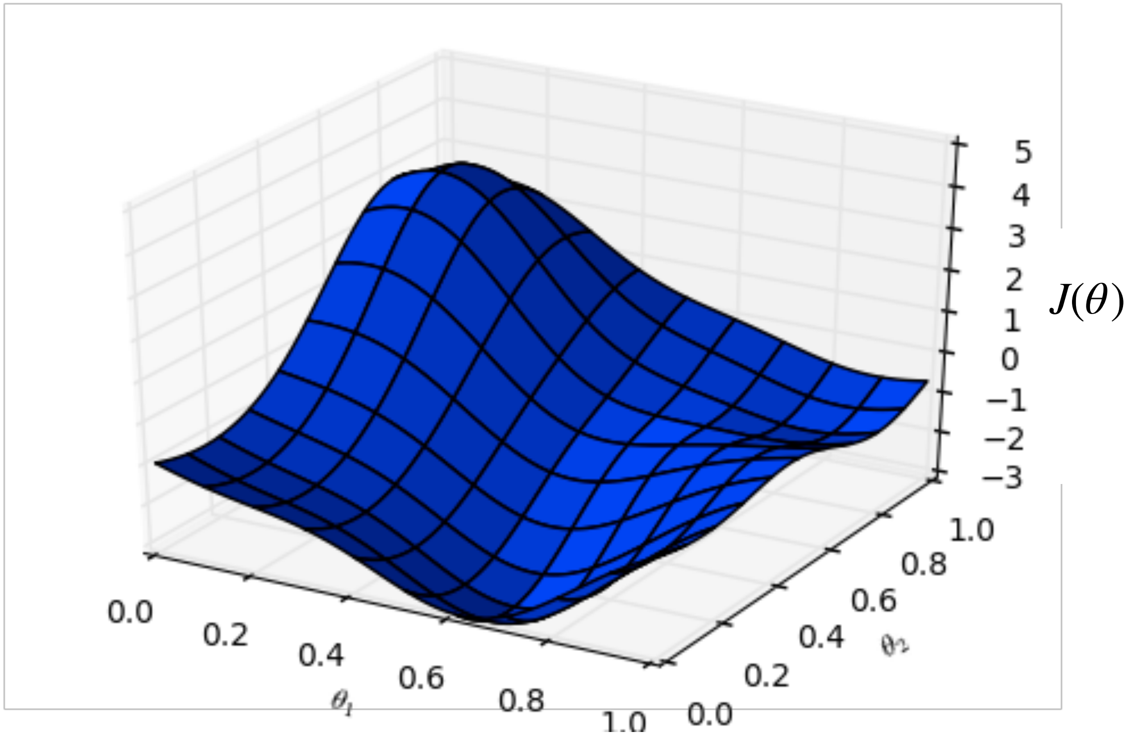
Every policy is defined by a parameter vector θ

Every policy has an expected return $J = \mathbb{E}[G]$

Can think of implicit mapping from θ to J

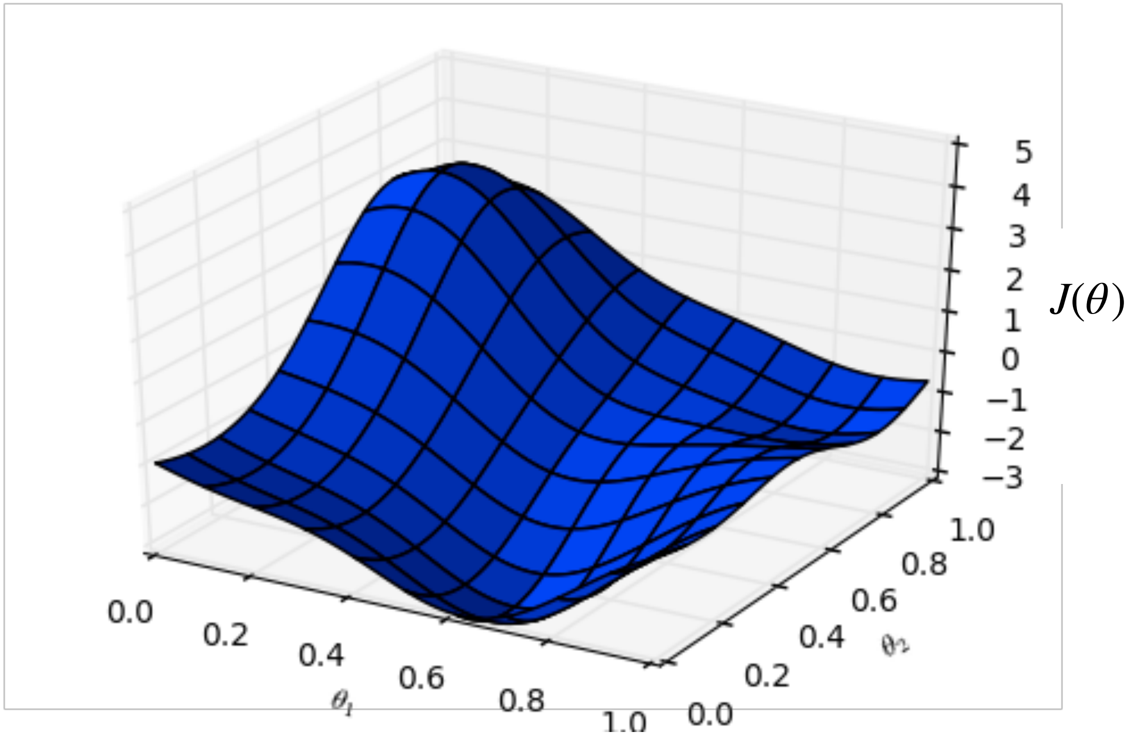


How do we find the best policy



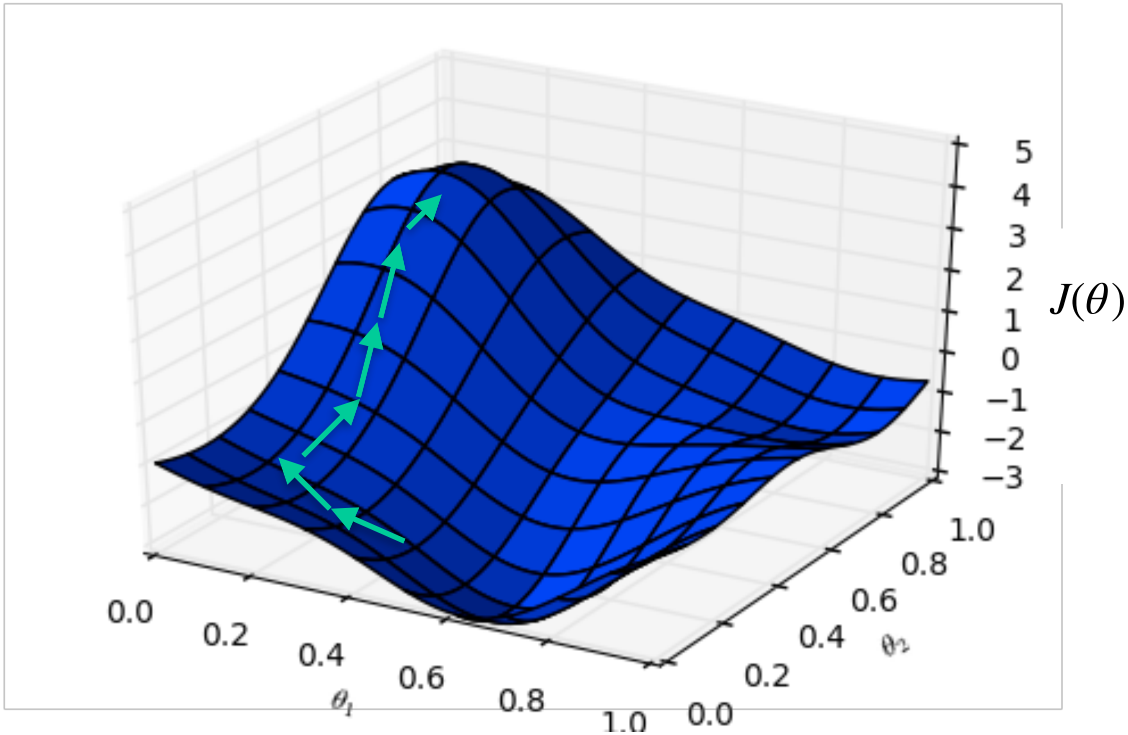
How do we find the best policy

- 0-order
  - Grid-search
  - Random search
  - Meta-heuristics



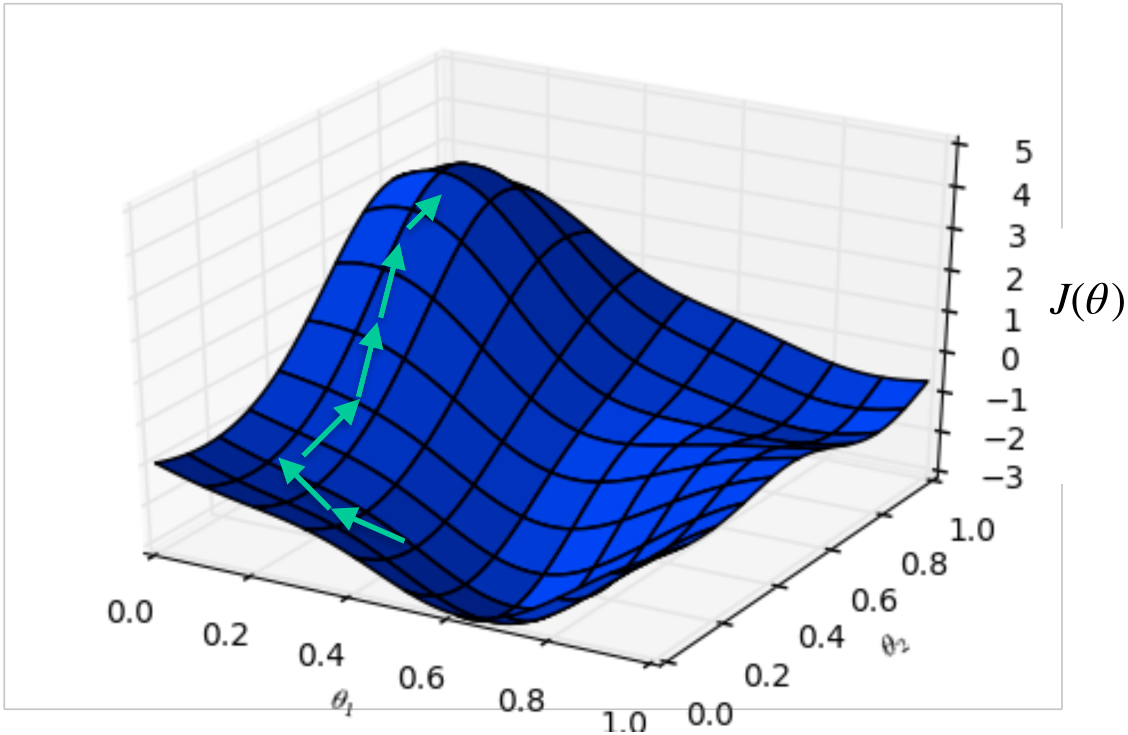
How do we find the best policy

- 0-order
  - Grid-search
  - Random search
  - Meta-heuristics
- 1st order
  - Policy Gradients
  - Today's topic



How do we find the best policy

- 0-order
  - Grid-search
  - Random search
  - Meta-heuristics
- 1st order
  - Policy Gradients
  - Today's topic
- 2nd order
  - Use information about how gradient changes
  - Next week



Finite difference gradients

Simplest update:

Consider finite difference estimates of gradient

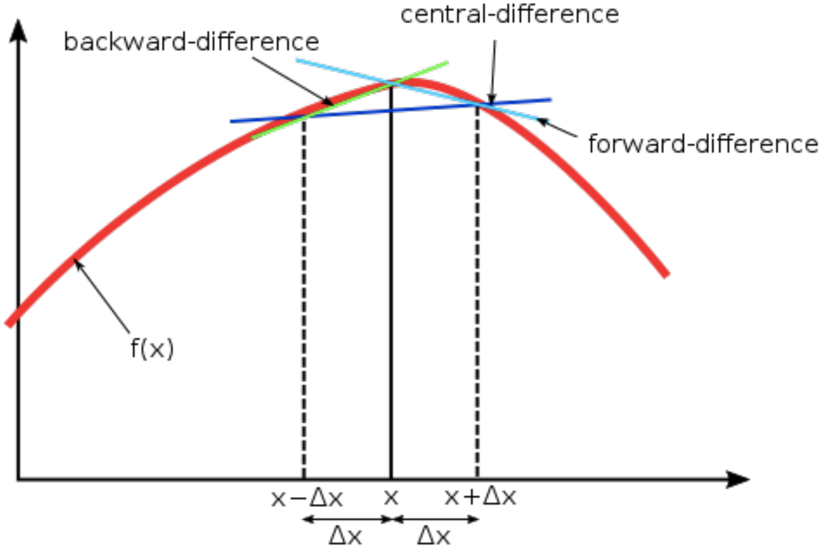
Evaluate two controllers, with parameters $\theta + \epsilon$ and $\theta - \epsilon$

Estimate gradient as

$$\nabla J \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

Update parameters

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J}$$



Finite difference gradients

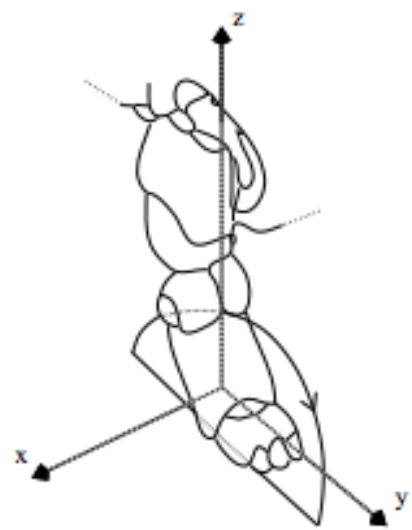
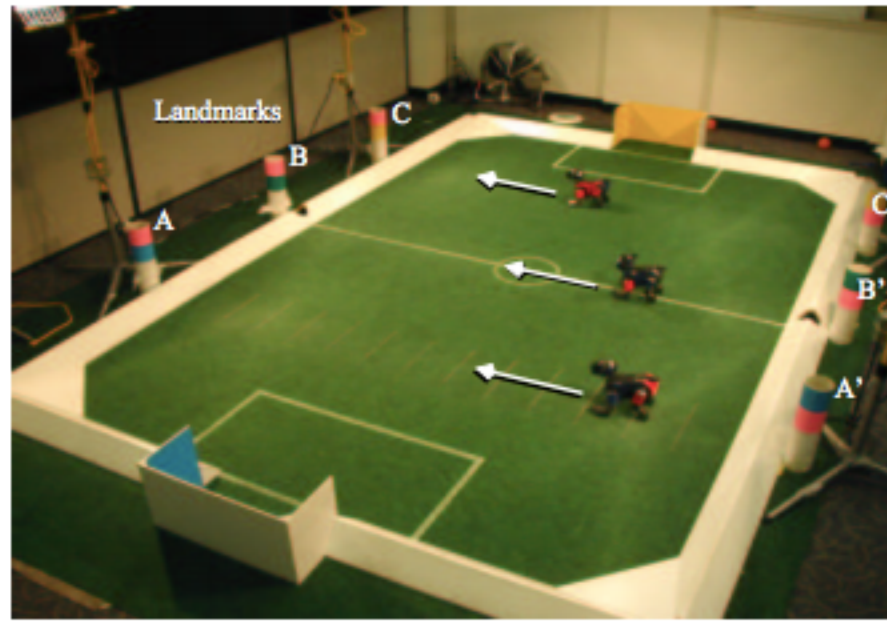


Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.



Kohl & Stone, Policy gradient reinforcement learning for fast quadrupedal locomotion, 2004

Finite difference gradients

General principle stays the same, always $\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J}$

However: for 1 parameter, need 2 full roll-outs

For *n* parameters, will need 2*n* full roll-outs

If function evaluation is stochastic, gradients will be *incredibly* noisy - thus, inefficient

What can we do with policy gradients?

- handle continuous actions

Can use policy with continuous output (linear, neural net)

- ensure smoothness in policies

Small step size ensures small change in policy

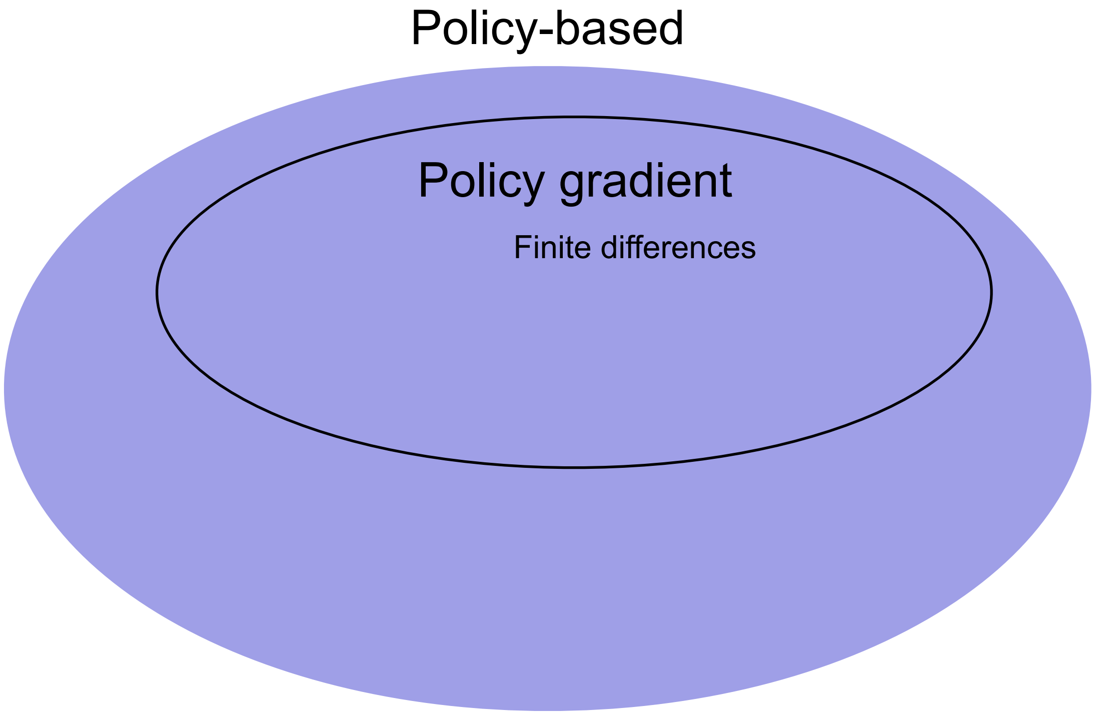
- hard to include prior knowledge about possible solutions

Include prior knowledge as policy form or initialisation

- can’t learn stochastic policies

Can easily train stochastic policies

Bigger picture



Represent policy directly
(e.g. as parametrised function)

Original REINFORCE

Can we calculate an **analytical** gradient for episodic problems

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau} G(\tau) \\ &= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) G(\tau) d\tau\end{aligned}$$

1) $$\nabla_{\boldsymbol{\theta}} p_{\theta}(\tau) = \nabla_{\boldsymbol{\theta}} \underbrace{p(\mathbf{s}_0)}_{?} \prod_{t=0}^{T} \underbrace{p(\mathbf{a}_t|\mathbf{s}_t)}_{\text{known}} \underbrace{p(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t)}_{?}$$

2) How to calculate the integral?

Original REINFORCE

Can we calculate an **analytical** gradient for episodic problems

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_\tau G(\tau) \\ &= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) G(\tau) d\tau\end{aligned}$$

Original REINFORCE

Can we calculate an **analytical** gradient for episodic problems

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_\tau G(\tau) \\
&= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) G(\tau) d\tau \\
&= \int \frac{p_{\boldsymbol{\theta}}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) G(\tau) d\tau \\
&= \mathbb{E}_\tau \left[\frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} G(\tau) \right] \\
&= \mathbb{E}_\tau \left[\nabla \log p_{\boldsymbol{\theta}}(\tau) G(\tau) \right] = \mathbb{E}_\tau \left[G(\tau) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{a}_t | \mathbf{s}_t) \right]
\end{aligned}$$

Intuitively: make action in high-return trajectories more likely

$$\nabla_{\boldsymbol{\theta}} \log p_\theta(\tau) = \underbrace{\nabla_{\boldsymbol{\theta}} \log p(\mathbf{s}_0)}_{=0} + \sum_{t=0}^{T} \underbrace{\nabla_{\boldsymbol{\theta}} \log p(\mathbf{a}_t | \mathbf{s}_t)}_{\text{known}} + \underbrace{\nabla_{\boldsymbol{\theta}} \log p(\mathbf{s}_{t+1} | \mathbf{a}_t, \mathbf{s}_t)}_{=0}$$

Original REINFORCE

As before, we can replace the expectation operator with an average to get a practical algorithm:

Sample N trajectories (N=1 is possible)

$$\widehat{\nabla_{\boldsymbol{\theta}} J} = \frac{1}{N} \sum_{i=1}^{N} \left[G(\tau_i) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathbf{a}_t^i | \mathbf{s}_t^i) \right]$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J}$$

Gradient estimates are **unbiased** and **consistent**

Easy to implement!

Original REINFORCE

The REINFORCE gradient uses knowledge of the policy.

- This makes it more efficient then finite differences

However

- Policy needs to be differentiable!
- Efficiency is still low!

Inefficient calculation of gradient

Consider a Bernoulli policy with parameter θ=0.5

Say, r(a=0) = 1 and r(a=1) = 3 & immediate termination

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} r(\mathbf{a}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} \left[\nabla \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) r(\mathbf{a}) \right]$$

observe a=0, r=1:

observe a=1, r=3:

Inefficient calculation of gradient

Consider a Bernoulli policy with parameter θ=0.5

Say, r(a=0) = 1 and r(a=1) = 3 & immediate termination

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} r(\mathbf{a}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} \left[\nabla \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) r(\mathbf{a}) \right]$$

observe a=0, r=1: $\widehat{\nabla J} = \frac{1}{1-\theta} \cdot -1 \cdot 1 = -2$

observe a=1, r=3:

$$\widehat{\nabla J} = \frac{1}{\theta} \cdot 3 = 6$$

Expected value: +2, Variance: 16

Inefficient calculation of gradient

Consider a Bernoulli policy with parameter θ=0.5

Say, r(a=0) = ~~1~~ -1 and r(a=1) = ~~3~~ +1 subtract a constant

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} r(\mathbf{a}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathbf{a})} \left[\nabla \log \pi_{\boldsymbol{\theta}}(\mathbf{a}) r(\mathbf{a}) \right]$$

observe a=0, r=1: $\widehat{\nabla J} = \frac{1}{1-\theta} \cdot -1 \cdot \cancel{1}_{-1} = -\cancel{2}_{+2}$

observe a=1, r=3:

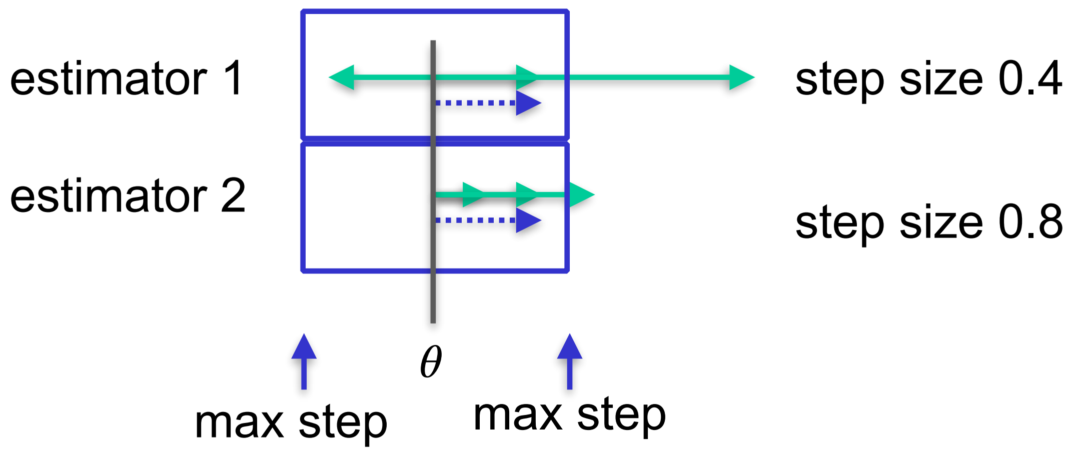
$$\widehat{\nabla J} = \frac{1}{\theta} \cdot \cancel{3}_{+1} = \cancel{6}_{+2}$$

Expected value: +2, Variance: 16

Same expected value, but now 0 variance!

Inefficient calculation of gradient

Why is variance bad?



- With lower variance, can probably use a higher learning rate

Inefficient calculation of gradient

Is average guaranteed to stay the same?

$$\mathbb{E}_\tau \left[(G(\tau) - b) \nabla \log p(\tau) \right]$$

$$= \underbrace{\mathbb{E}_\tau \left[G(\tau) \nabla \log p(\tau) \right]}_{= \nabla J} - \mathbb{E}_\tau \left[b \nabla \log p(\tau) \right]$$

$$\mathbb{E}_\tau \left[b \nabla \log p(\tau) \right] =$$

Inefficient calculation of gradient

Is average guaranteed to stay the same?

$$\mathbb{E}_\tau\left[(G(\tau)-b)\nabla\log p(\tau)\right]$$
$$=\underbrace{\mathbb{E}_\tau\left[G(\tau)\nabla\log p(\tau)\right]}_{=\nabla J}-\mathbb{E}_\tau\left[b\nabla\log p(\tau)\right]$$

$$\mathbb{E}_\tau\left[b\nabla\log p(\tau)\right]=b\int p(\tau)\nabla\log p(\tau)d\tau$$
$$=b\int p(\tau)\frac{\nabla p(\tau)}{p(\tau)}d\tau=b\,\nabla\underbrace{\underbrace{\int p(\tau)d\tau}_{=1}}_{=0}=0$$

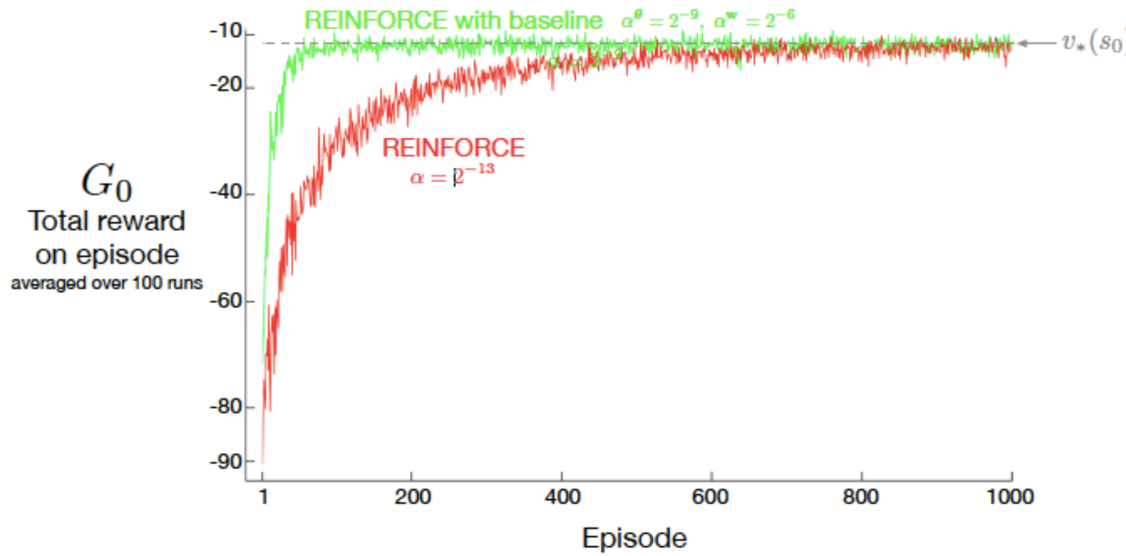
Thus, the gradients are still **unbiased**

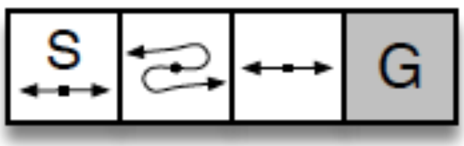
We'll look at the **variance** in the exercises

Inefficient calculation of gradient

A good baseline is the expected reward (e.g. observed average)

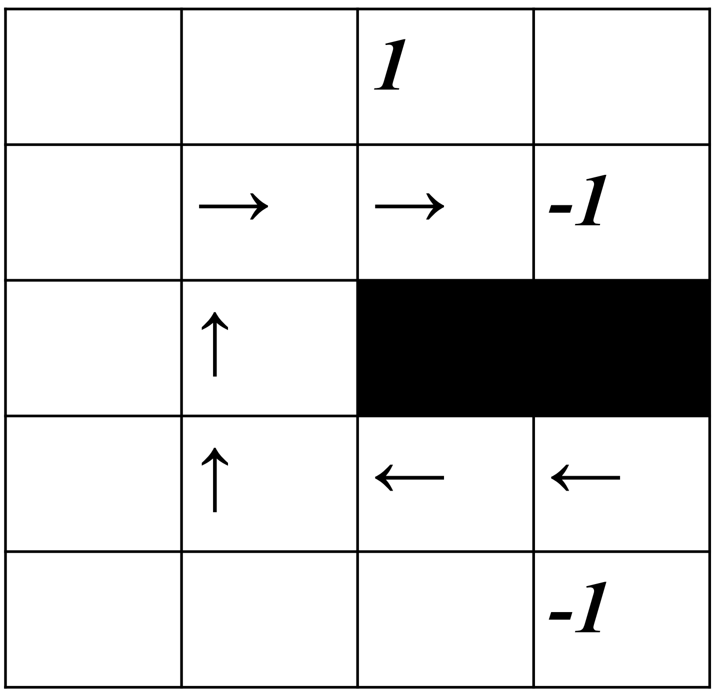
Baselines can depend on state. Then, a value function estimate is a popular choice.

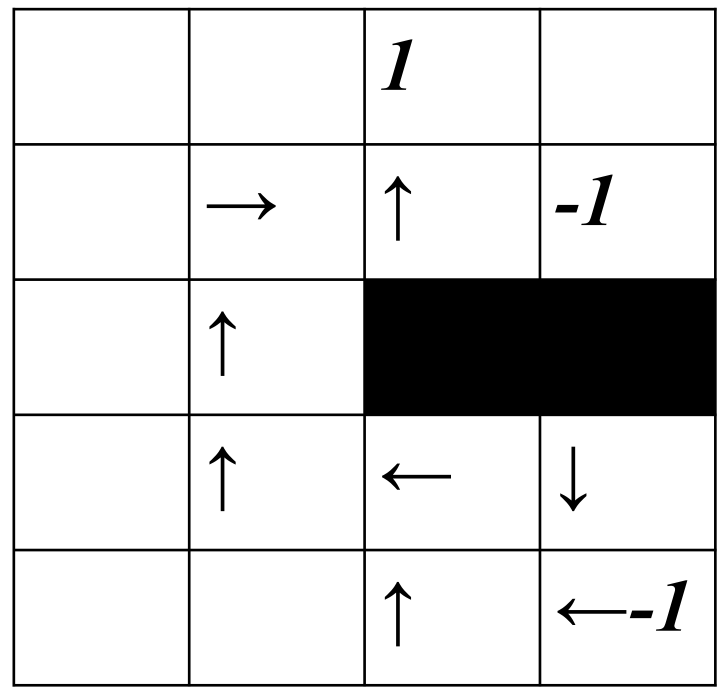




Sutton & Barto. Reinforcement Learning: An introduction.

Inefficient credit assignment





Reinforce with baseline

The REINFORCE gradient uses knowledge of the policy.

- This makes it more efficient then finite differences

However

- Policy needs to be differentiable!
- Efficiency is still low!

- Baseline improves variance a bit
- All actions get credit for all rewards in a trajectory
  - Good action in end gets punished for bad action in beginning
- Variance from stochastic transitions increases with T

Comparison

Finite differences
(Blackbox gradients)

Original REINFORCE
(Whitebox gradients)

Episode-based

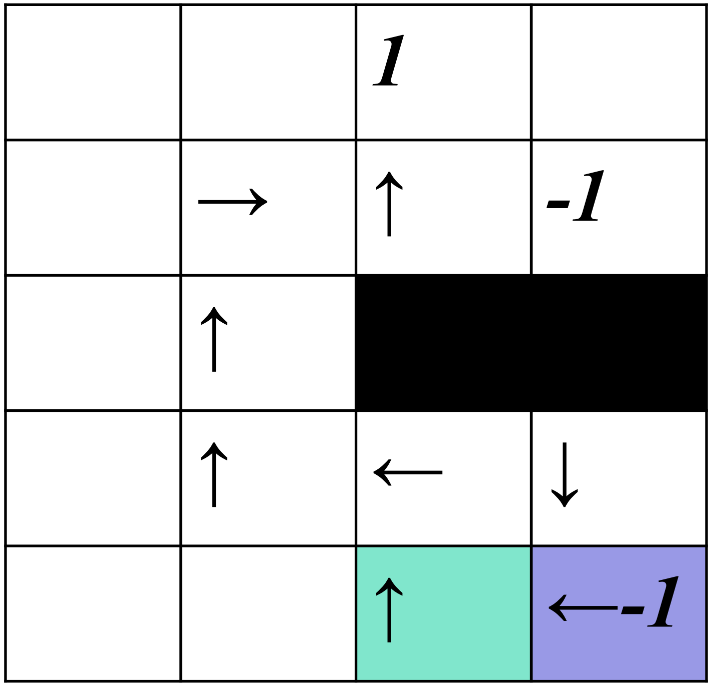
Exploration

Step-based

Reinforce / G(PO)MDP

Intuitively, we can't conclude whether the green action was good or bad based on the blue reward.

Can we use this to come up with a better update?



Reinforce / G(PO)MPD

$$\nabla J = \mathbb{E}_\tau \left[\sum_{t=1}^{T} r_t \sum_{t'=1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right]$$

$$= \sum_{t=1}^{T} \mathbb{E}_{\tau_{1:t}} \mathbb{E}_{\tau_{t+1:T}} \left[r_t \left(\sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) + \sum_{t'=t+1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right) \middle| \tau_{1:t} \right]$$

$$= \sum_{t=1}^{T} \mathbb{E}_{\tau_{1:t}} \left[r_t \left(\sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) + \mathbb{E}_{\tau_{t+1:T}} \left[\sum_{t'=t+1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \middle| \tau_{1:t} \right] \right) \right]$$

Reinforce / G(PO)MPD

$$\nabla J = \mathbb{E}_\tau \left[\sum_{t=1}^{T} r_t \sum_{t'=1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right]$$

$$= \sum_{t=1}^{T} \mathbb{E}_{\tau_{1:t}} \mathbb{E}_{\tau_{t+1:T}} \left[r_t \left(\sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) + \sum_{t'=t+1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right) \middle| \tau_{1:t} \right]$$

$$= \sum_{t=1}^{T} \mathbb{E}_{\tau_{1:t}} \left[r_t \left(\sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) + \underbrace{\mathbb{E}_{\tau_{t+1:T}} \left[\sum_{t'=t+1}^{T} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \middle| \tau_{1:t} \right]} \right) \right]$$

$$= \sum_{t'} \mathbb{E}_{s_{t'}} \mathbb{E}_{a_{t'}} \left[\frac{\nabla p(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{p(\mathbf{a}_{t'}|\mathbf{s}_{t'})} | \mathbf{s}_{t'} \right]$$

$$= \sum_{t'} \mathbb{E}_{s_{t'}} \int_{\mathcal{A}} p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \frac{\nabla p(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{p(\mathbf{a}_{t'}|\mathbf{s}_{t'})} d\mathbf{a}_{t'}$$

$$= \sum_{t'} \mathbb{E}_{s_{t'}} \nabla \int_{\mathcal{A}} p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) d\mathbf{a}_{t'} = 0$$

$$= \mathbb{E}_\tau \left[\sum_{t=1}^{T} r_t \sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \right]$$

Reinforce / G(PO)MPD

$$\mathbb{E}_\tau \left[\sum_{t=1}^{T} r_t \sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'} | \mathbf{s}_{t'}) \right] \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} r_t \sum_{t'=1}^{t} \nabla \log p(\mathbf{a}_{t'} | \mathbf{s}_{t'}) \right)$$

Replacing the expected value by an average again creates an implementable algorithm

Reinforce/GPOMDP with baseline

REINFORCE with Baseline (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 $\hat{v}(s,\mathbf{w})$
Algorithm 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 $\mathbf{0}$)

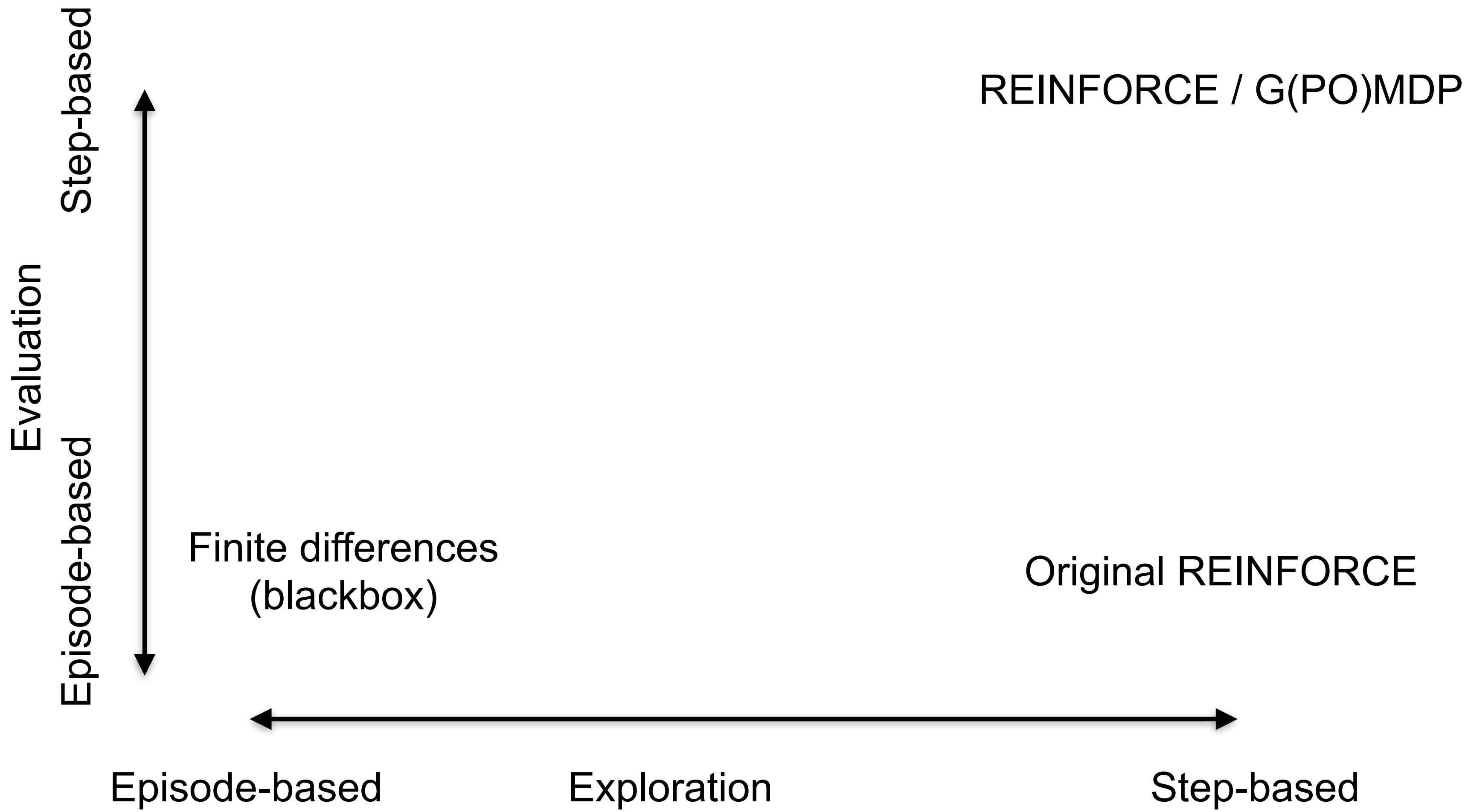
Learning together with v-function

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)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$ V-fc update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$ Policy update

Reinforce/GPOMDP with baseline

- Still easy to implement
- Cut one source of variance compared to REINFORCE
- Not changing any expected value: still **consistent**, **unbiased**
- Can be combined with baseline (e.g., learned value fc)
- Still: variance from stochastic transitions increases with T

Comparison



Step-based
Evaluation
Episode-based
REINFORCE / G(PO)MDP
Finite differences (blackbox)
Original REINFORCE
Episode-based
Exploration
Step-based

Big picture: How to learn policies

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\dots N}$$

Learn value function
$V(\mathbf{s}), Q(\mathbf{s}, \mathbf{a})$

Policy
$\pi(\mathbf{a}|\mathbf{s})$

Value-based

Optimize policy
$\pi(\mathbf{a}|\mathbf{s})$

Policy-based

Model-free RL

Learn dynamics model
$p(\mathbf{s}'|\mathbf{s}, \mathbf{a})\ r(\mathbf{s}, \mathbf{a})$

Learn value or policy
$V(\mathbf{s}), Q(\mathbf{s}, \mathbf{a}), \pi(\mathbf{a}|\mathbf{s})$

Policy
$\pi(\mathbf{a}|\mathbf{s})$

Model-based RL

Thanks to Jan Peters

Advantages of policy-based methods

- handle continuous actions

Can use policy with continuous output (linear, neural net)

- ensure smoothness in policies

  - Setting small step size will ensure change are small in each step
  - We'll see better methods next week

- hard to include prior knowledge about possible solutions

Include prior knowledge as policy form or initialisation

- can't learn stochastic policies

Can easily train stochastic policies

Weaknesses of policy-based methods

Actor-only methods have high variance from Monte-Carlo

A lot of the methods we discussed are specific to episodic setting

Requires stochastic policies, what if deterministic is optimal?

- If amount of randomness is learned, can get close to deterministic
- We will also see a policy gradient method to learn deterministic policies

Thanks for your attention!

Feedback?

h.c.vanhoof@uva.nl