Off-policy learning &Policy-based methods

Herke van Hoof

Wrap-up from last week

Off-policy learning with bootstrapping & function approximation tricky!

Semi-gradient TD(0) can diverge.

Can we use a *true gradient* method? If so, with respect to which objective function?

Mean squared projected Bellman error?

PBE last chance for bootstrapping!

- Linear approximation: PBE=0 at w_{TD}
- On-policy, semi-gradient finds w_{TD}
- LSTD find PBE

Note: for on-policy data, semi-gradient TD converges to PBE=0, although the updates are not its true gradient

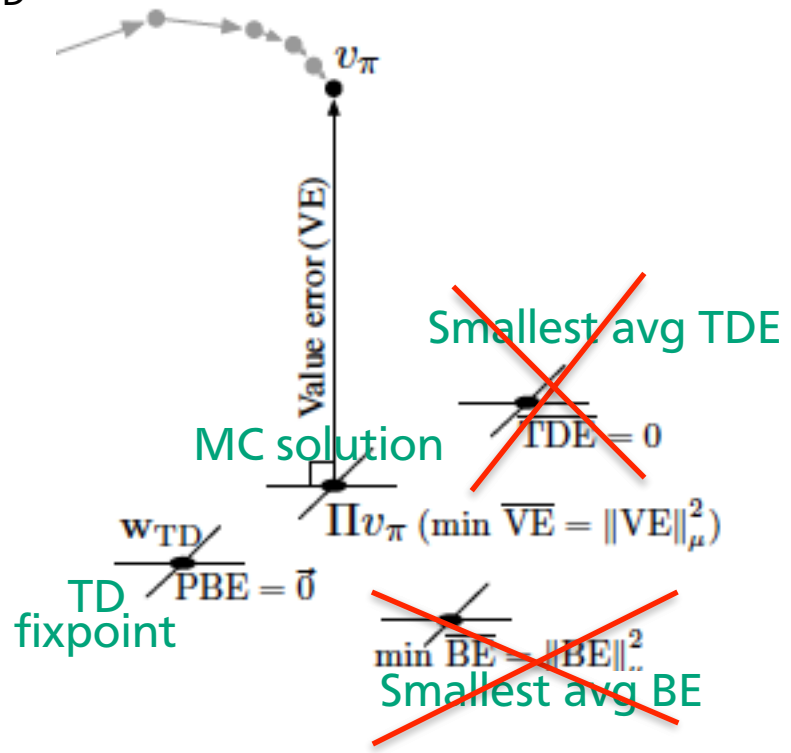


Figure: Sutton & Barto. RL:AI

Mean squared projected Bellman error?

$$\mathbf{D} = \begin{bmatrix} \mu(s_1) & 0 & \dots & 0 \\ 0 & \mu(s_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mu(s_n) \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1(s_1) & \dots & x_m(s_1) \\ x_1(s_2) & \dots & x_m(s_2) \\ \vdots & \ddots & \vdots \\ x_1(s_n) & \dots & x_m(s_n) \end{bmatrix}$$

$$\begin{aligned} \overline{\mathrm{PBE}}(\mathbf{w}) &= \left\| \Pi \bar{\delta}_{\mathbf{w}} \right\|_{\mu}^2 \\ &= \left(\Pi \bar{\delta}_{\mathbf{w}} \right)^\top \mathbf{D} \Pi \bar{\delta}_{\mathbf{w}} \\ &\vdots \\ &= \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right)^\top \left(\mathbf{X}^\top \mathbf{D} \mathbf{X} \right)^{-1} \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right) \\ \nabla \overline{\mathrm{PBE}}(\mathbf{w}) &= 2 \nabla \left[\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right]^\top \left(\mathbf{X}^\top \mathbf{D} \mathbf{X} \right)^{-1} \left(\mathbf{X}^\top \mathbf{D} \bar{\delta}_{\mathbf{w}} \right) \end{aligned}$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\nabla \left[\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}}\right]^\top \left(\mathbf{X}^\top \mathbf{D}\mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D}\bar{\delta}_{\mathbf{w}}\right)$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\nabla \left[\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right]^\top \left(\mathbf{X}^\top \mathbf{D}\mathbf{X}\right)^{-1} \left(\mathbf{X}^\top \mathbf{D}\bar{\delta}_\mathbf{w}\right)$$

$$\approx \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]$$

$$\mathbf{X}^\top \mathbf{D}\mathbf{X} = \sum_s \mu(s)\mathbf{x}_s \mathbf{x}_s^\top = \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]$$

$$= \mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right]$$

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]$$

Can we just plug in a new samples (s,a,s') tuple in each of these?

Mean squared projected Bellman error?

$$\nabla \overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \underbrace{\mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]}_{\mathbf{v}}$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, and only plug in sample for part of the data

Mean squared projected Bellman error?

$$\nabla \overline{\text{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t \left(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t\right) \mathbf{x}_t^\top\right] \underbrace{\mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\rho_t \delta_t \mathbf{x}_t\right]}_{\mathbf{v}}$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, and only plug in sample for part of the data

v is the solution to a least squares problem - for such problems there is a SGD algorithm: $\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \beta \rho_t \left(\delta_t - \mathbf{v}_t^\top \mathbf{x}_t\right) \mathbf{x}_t$

Plug in the parts: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha/2 \nabla \overline{\text{PBE}}\left(\mathbf{w}_t\right)$

Mean squared projected Bellman error?

$$\nabla\overline{\mathrm{PBE}}(\mathbf{w}) = 2\mathbb{E}\left[\rho_t\left(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t\right)\mathbf{x}_t^\top\right]\underbrace{\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right]^{-1}\mathbb{E}\left[\rho_t\delta_t\mathbf{x}_t\right]}_{\mathbf{v}}$$

The first and last factor are dependent - both depend on x_{t+1}

Learn some factors from all data, and only plug in sample for part of the data

v is the solution to a least squares problem - for such problems there is a SGD algorithm: $\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \beta\rho_t\left(\delta_t - \mathbf{v}_t^\top\mathbf{x}_t\right)\mathbf{x}_t$

Plug in the parts: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha/2\nabla\overline{\mathrm{PBE}}\left(\mathbf{w}_t\right)$

$$\approx \mathbf{w}_t + \alpha\mathbb{E}\left[\rho_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)\mathbf{x}_t^\top\right]\mathbf{v}_t$$

$$\approx \mathbf{w}_t + \alpha\rho_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)\mathbf{x}_t^\top\mathbf{v}_t$$

Mean squared projected Bellman error?

This is GTD2, a gradient-TD method

- Converges to 0 mean squared projected Bellman error (linear features)
- With non-linear features, converges to local optimum
- There are other, more sophisticated gradient-TD methods

- Cost?
- Additional step size parameter for learning $\mathbf{v}$ ('larger' than α)
- Need to store and update two parameter vectors

Let's revise last week's conclusion

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *	global c. min. VE	global c. min. VE	local c. min. VE		
Semi-gradient TD *	global c. PBE=0	global c. PBE=0	no c. guarantee		
Gradient TD *					
LSTD	global c. PBE=0	global c. PBE=0	-		

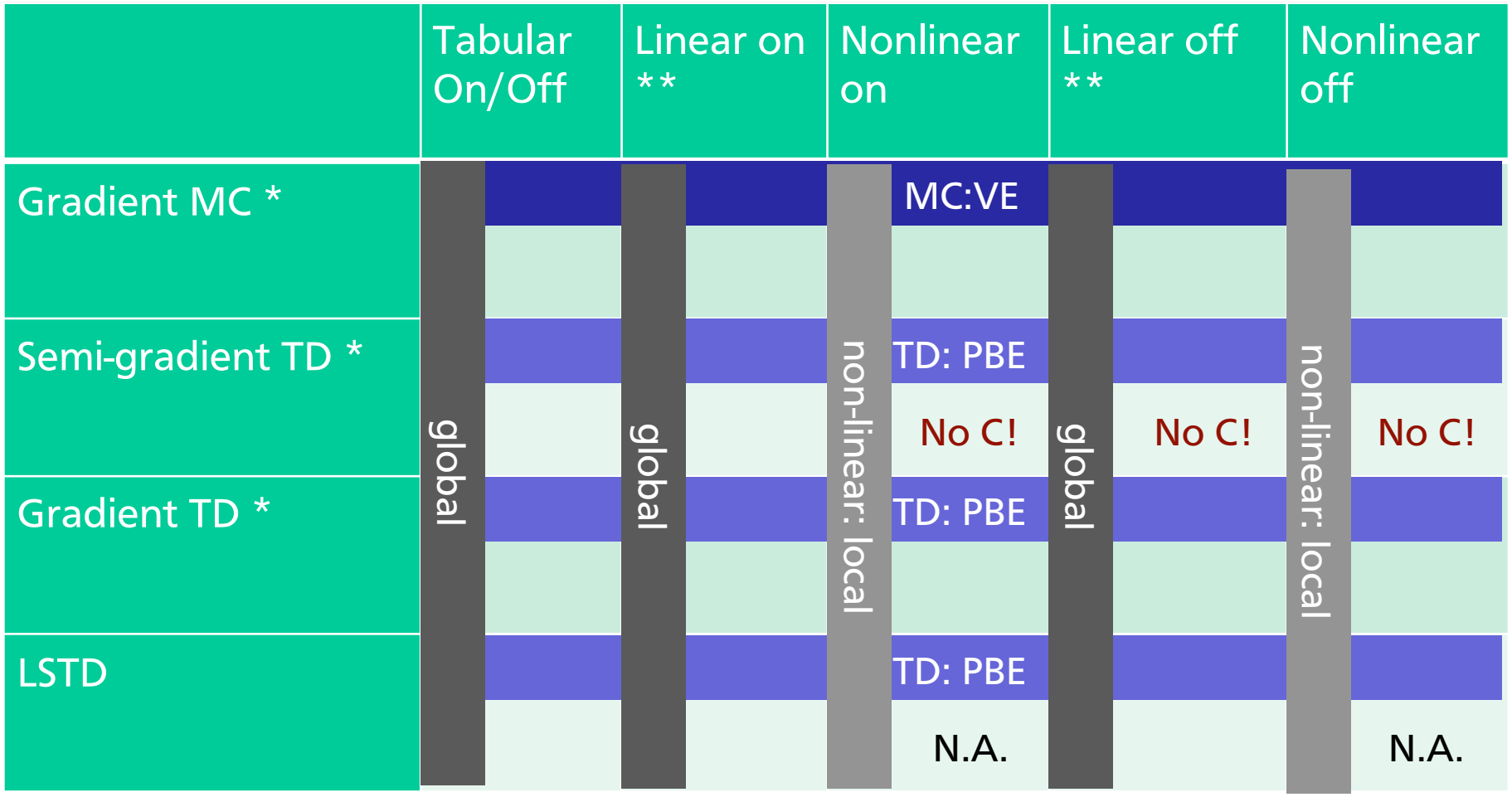
* with appropriate step-size schedule ** if features independent, single solution

Let's revise last week's conclusion

	Tabular On/Off	Linear on **	Nonlinear on	Linear off **	Nonlinear off
Gradient MC *	global c. min. VE	global c. min. VE	local c. min. VE	global c. min. VE	local c. min. VE
Semi-gradient TD *	global c. PBE=0	global c. PBE=0	no c. guarantee	no c. guarantee	no c. guarantee
Gradient TD *	global c. PBE=0	global c. PBE=0	local c. min PBE	global c. PBE=0	local c. min PBE
LSTD	global c. PBE=0	global c. PBE=0	-	global c. PBE=0	-

* with appropriate step-size schedule ** if features independent, single solution

Let's revise last week's conclusion



* with appropriate step-size schedule

** if features independent, single solution

Deep Q networks

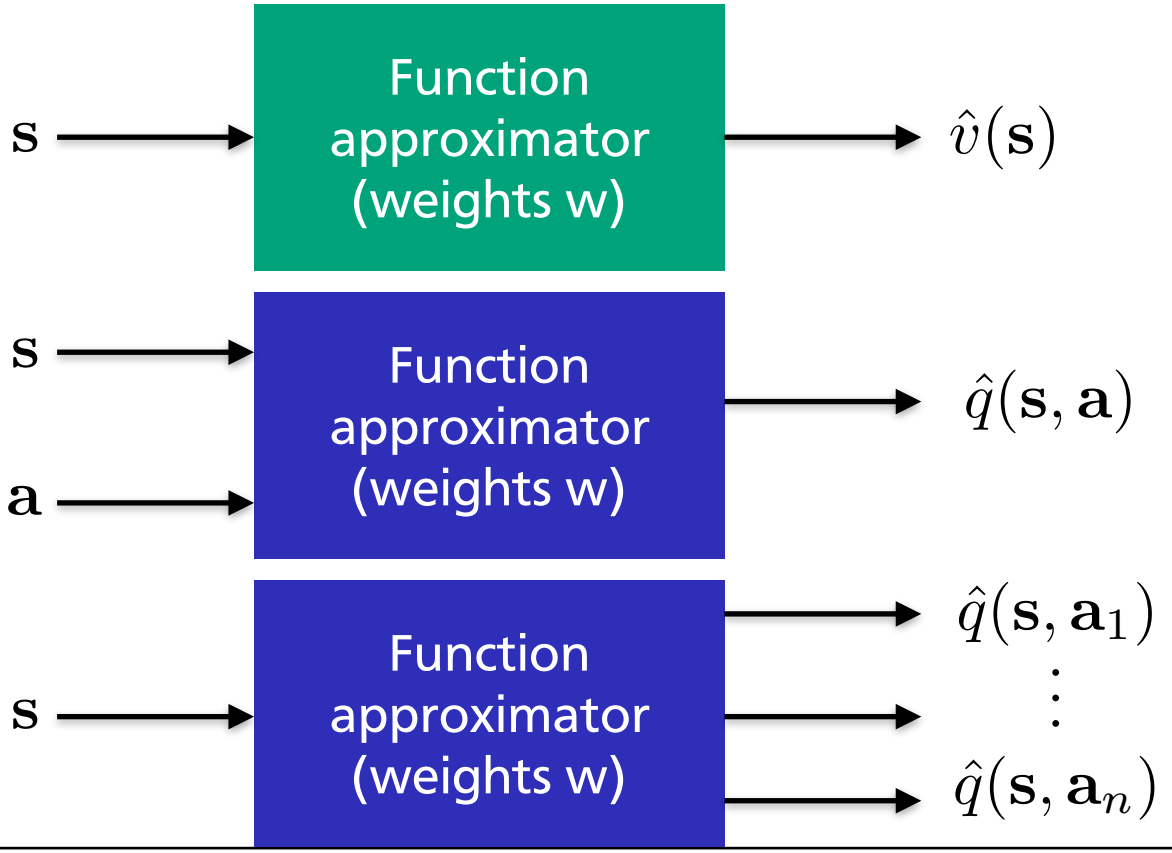
The famous algorithm for learning to play Atari games: Deep Q networks (DQN)

Illustrates some of the challenges in doing control with (non-linear) function approximation

Figure: Deepmind.com

Deep Q networks

DQN approximates the Q function. Two possibilities compared to approximating V:



(basically, a mapping is learned from states to lookup tables over actions)

Deep Q networks

Linear approximation has nice properties, but task-specific features need to be designed

DQN: Instead, use a multi-layer neural network. Idea is that early layers can learn features

Thus, the network can directly take relatively ‘raw’ input without task-specific preprocessing

Deep Q networks

Most layers in DQN are convolutional layers:

- Same filters are used at each location in the image
- Reduces #weights by only connecting neurons locally
- Further reduces #weights by sharing parameters at each location
- Imposes translation equivariance

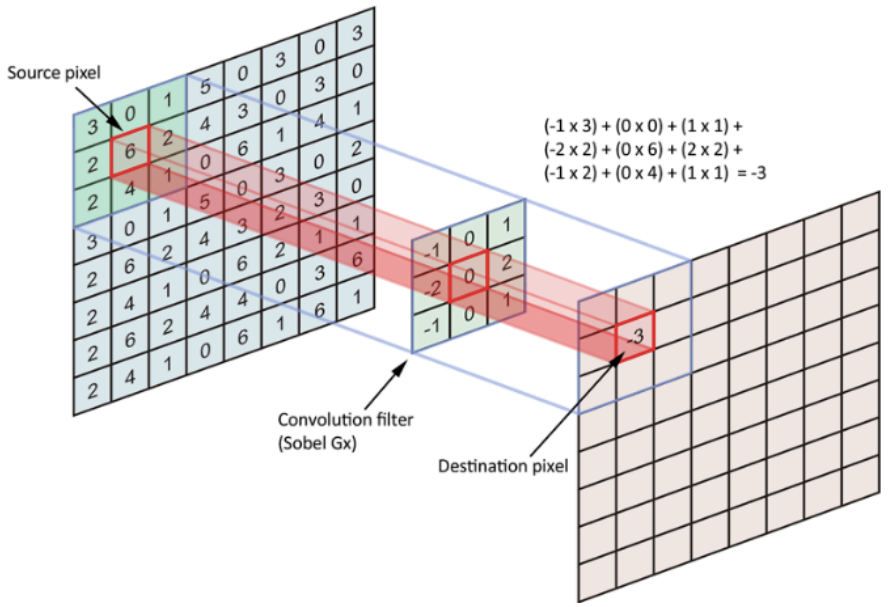


Figure: towardsdatascience.com

Deep Q networks

Convolutional layers and fully connected layers of ReLU units

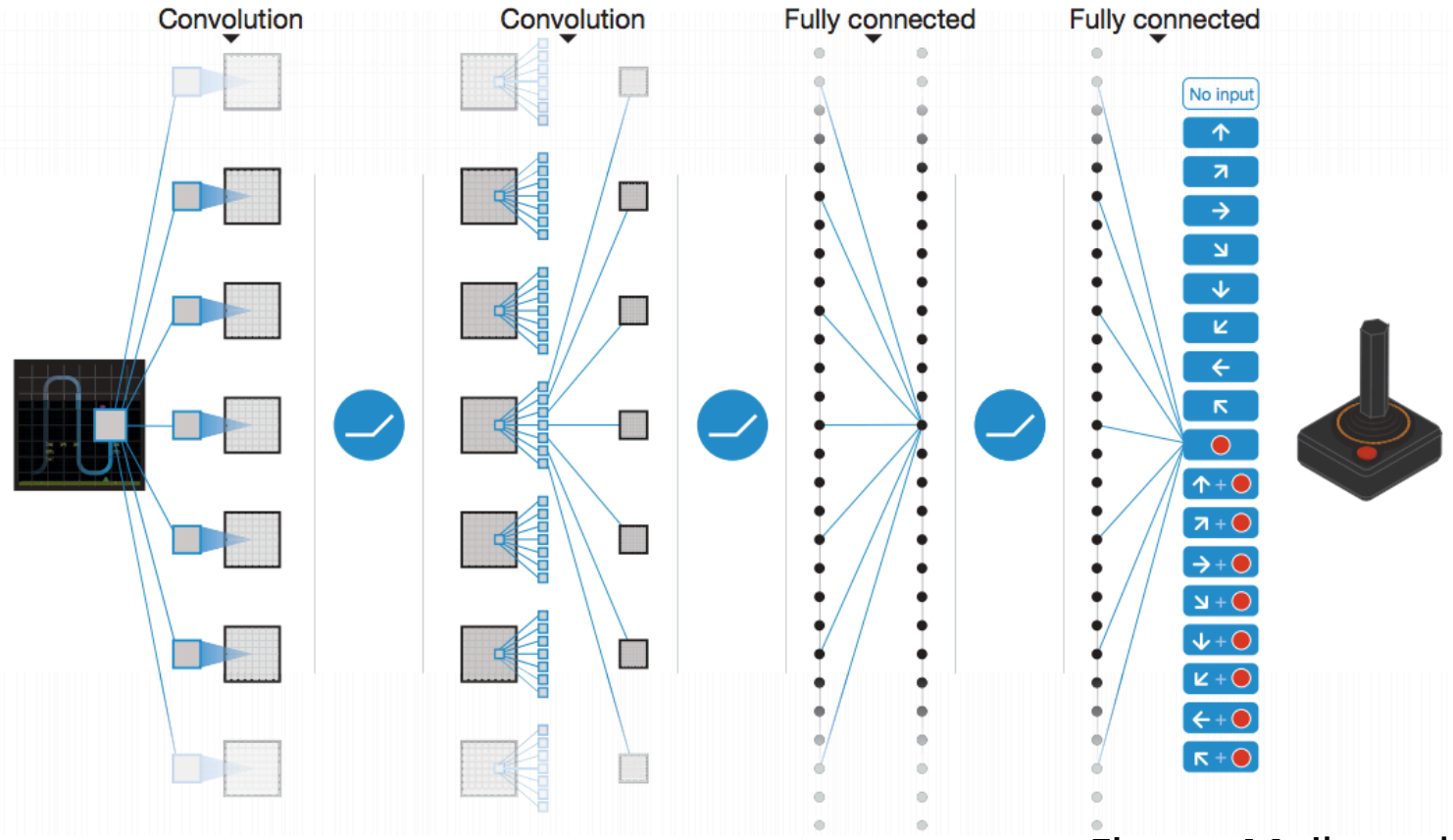
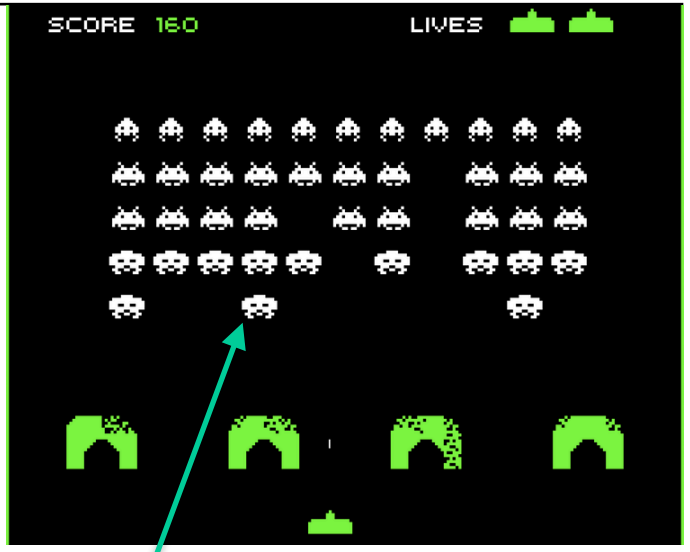


Figure: Mnih et al., 2015

Deep Q networks

Pre-processing (Task-agnostic)

- Downscale image and convert to grayscale
- Stack frames
- Reward = increment in score



Going left or right?

Figure: Smithsonianmag.com

Deep Q networks

In regular TD-learning, each transition used once. Inefficient

Also, subsequent transitions are highly **correlated**

Break correlation using experience replay

- Store experiences in a buffer
- At every time step, train on a batch of experiences from the buffer, not the most recent one
- Makes sure the network can't 'overfit' to recent experience only

Implication for on-policy / off-policy learning?

Deep Q networks

Learning with semi-gradient version of q-learning

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Target depends on w

Dependence of target on w can cause instabilities

- Copy parameters $\tilde{\mathbf{w}}$ every C steps, and use these in the target

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \tilde{\mathbf{w}}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Deep Q networks

Learning with semi-gradient version of q-learning

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Target depends on w

Dependence of target on w can cause instabilities

- Copy parameters $\tilde{\mathbf{w}}$ every C steps, and use these in the target

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_a \hat{q}(S', a, \tilde{\mathbf{w}}) - \hat{q}\left(S_t, a, \mathbf{w}\right) \right] \nabla \hat{q}\left(S_t, a, \mathbf{w}\right)$$

Error term

Extra trick: clip error to [-1,1] (more stability, easy to set α)

Deep Q networks

Results with the **same** parameters and settings on many games

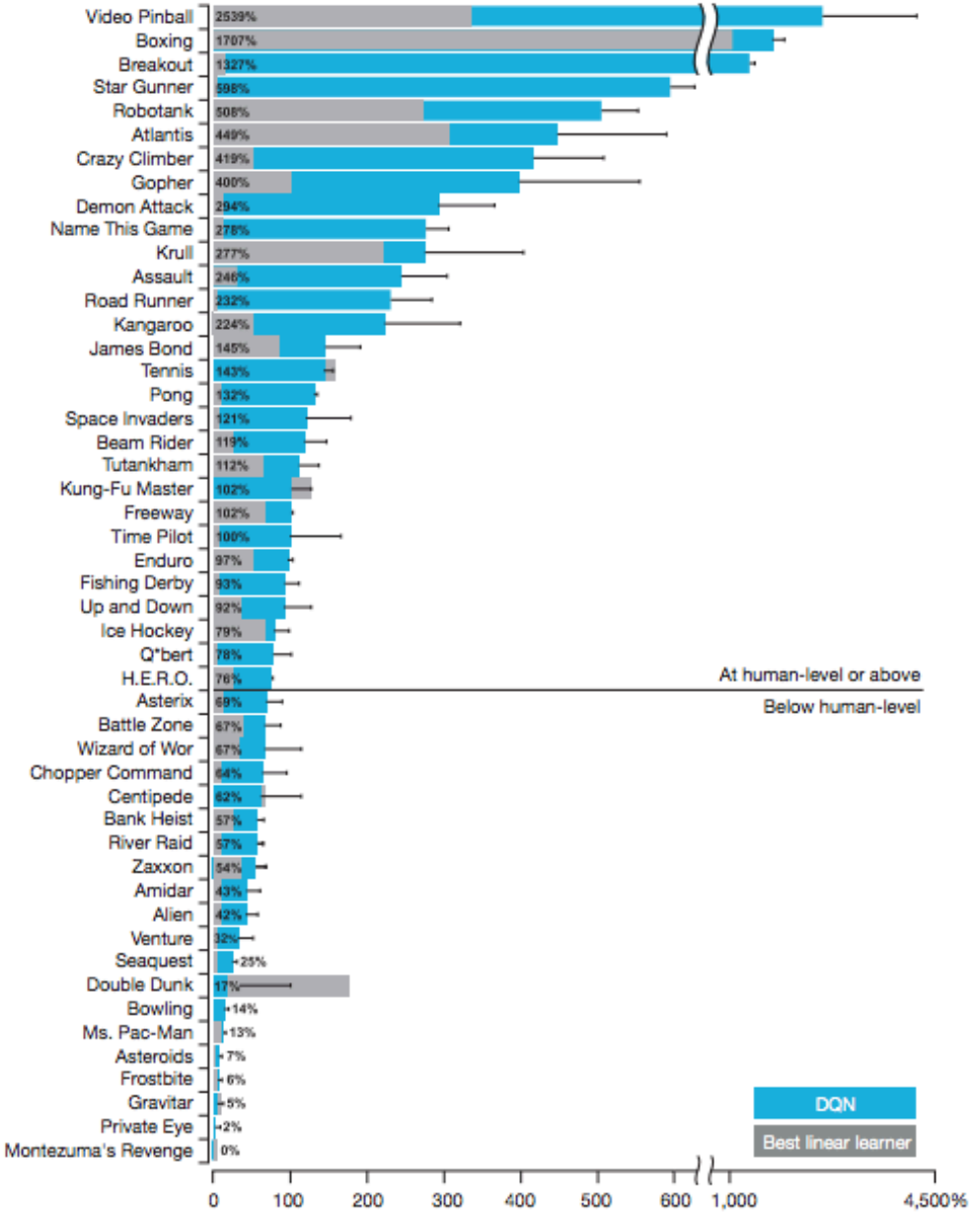


Figure: Mnih et al., 2015

Deep Q networks

Many of the tricks make the problem more ‘like supervised’

- Target network: fixed target value
- Experience replay: closer to iid

Still semi-gradient used off-policy

- Hence the need to stabilise learning
- Lack theoretical guarantees, but good empirical performance

Non-linear function approximation

- Risk of local optima seems not so bad in practice for deep nets

Extension: double DQN (similar to double Q learning)

Powerful demonstration of RL without task-specific features!

Conclusion

On-policy control straightforward

Off-policy prediction and control very tricky!

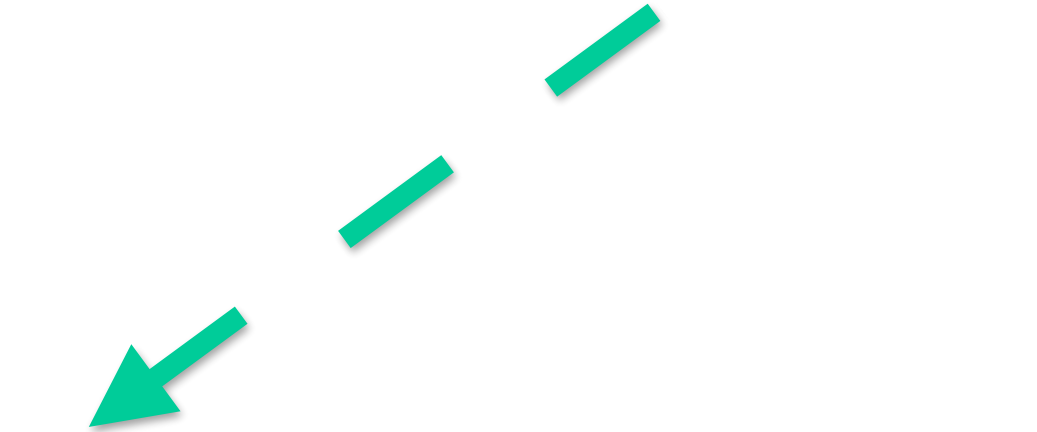
- Gradient TD, that uses gradient of PBE, is a solution
- Overhead: keep track of more values

Alternative: use additional mechanisms to keep learning stable

- Good empirical performance with DQN

Big picture: How to learn policies

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

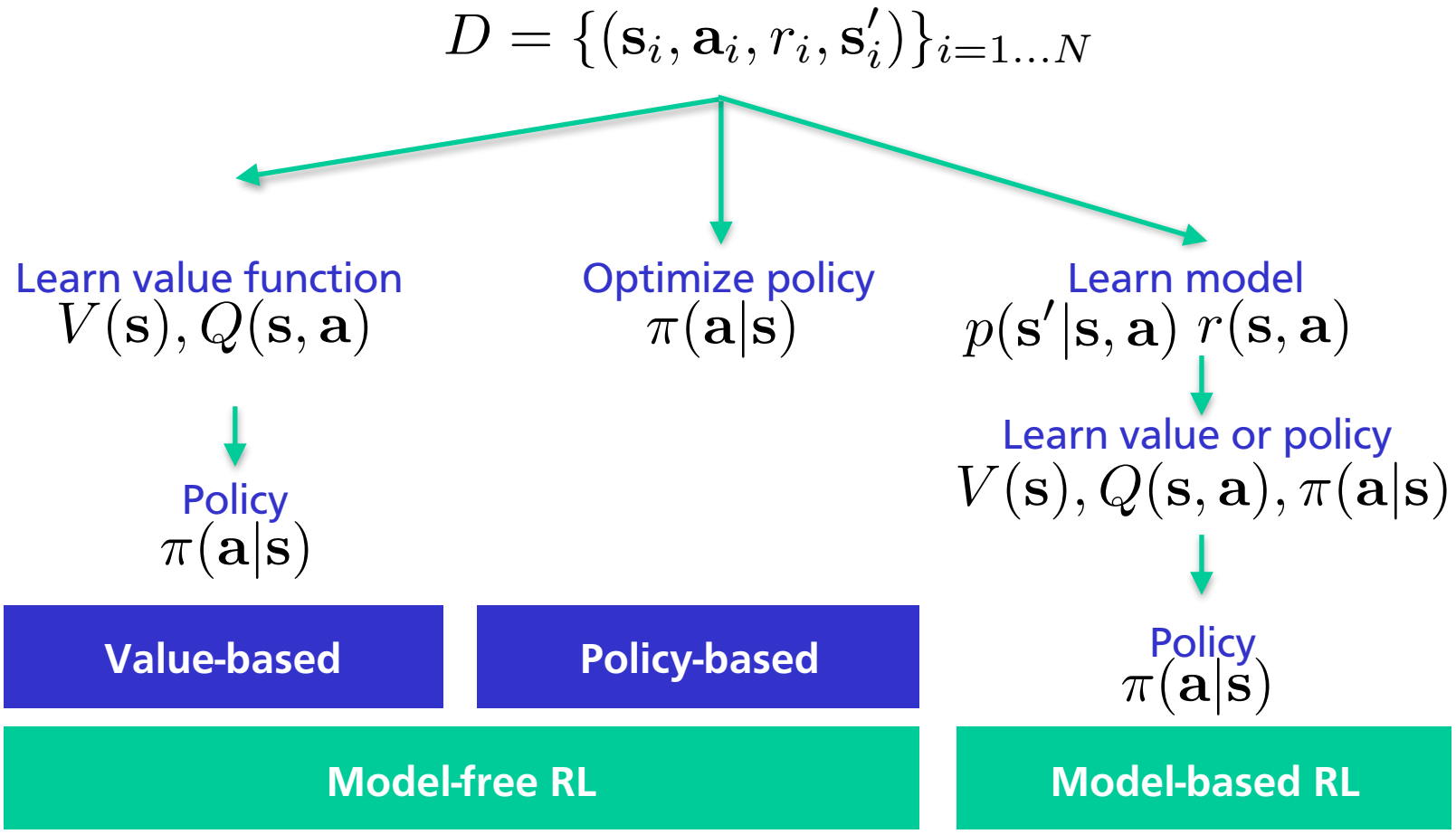


Policy

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

Thanks to Jan Peters

Big picture: How to learn policies



Thanks to Jan Peters

Shortcomings of action-value methods

Handle continuous actions

- How can we do the max operator efficiently?

Ensure smoothness in policies

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

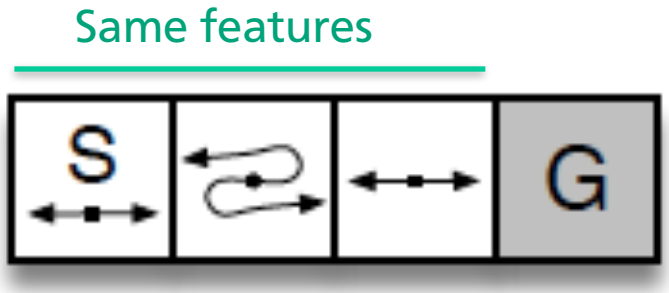
Hard to include prior knowledge about possible solutions

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

A deterministic policy can be turned in to a stochastic policy by adding random noise. The intensity of the noise can be pre-set or it can be one of the controller parameters.

Policy representation

Linear gaussian policy (cont. actions)

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

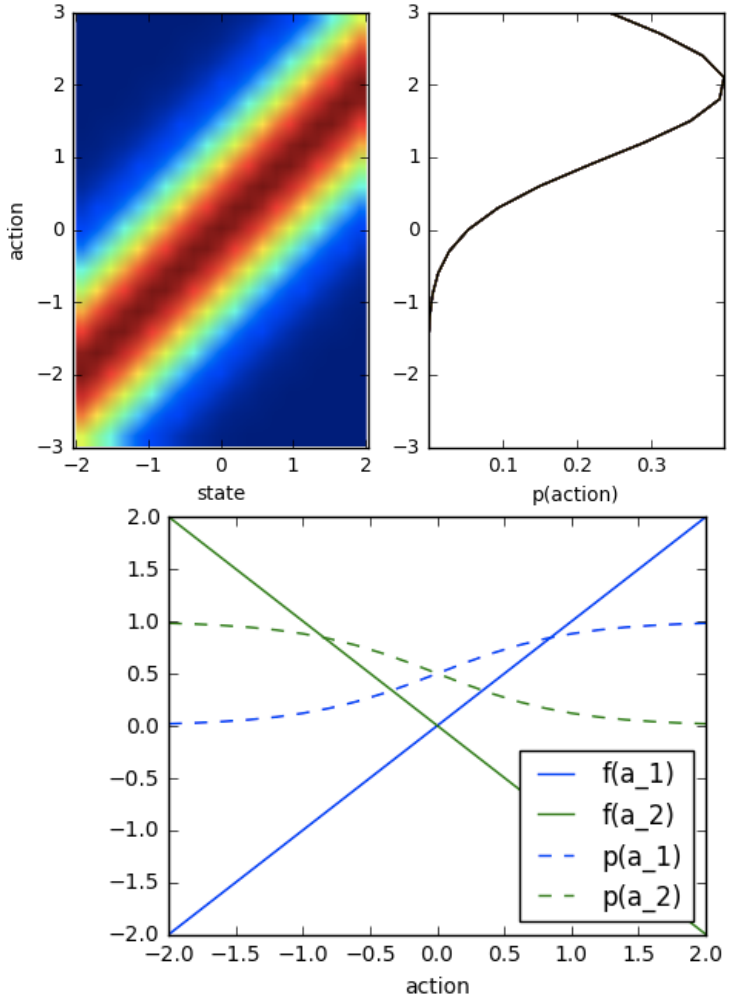
Neural network policy (cont. actions)

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

Softmax policy (discrete actions)

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

f itself can be linear, NN, ...

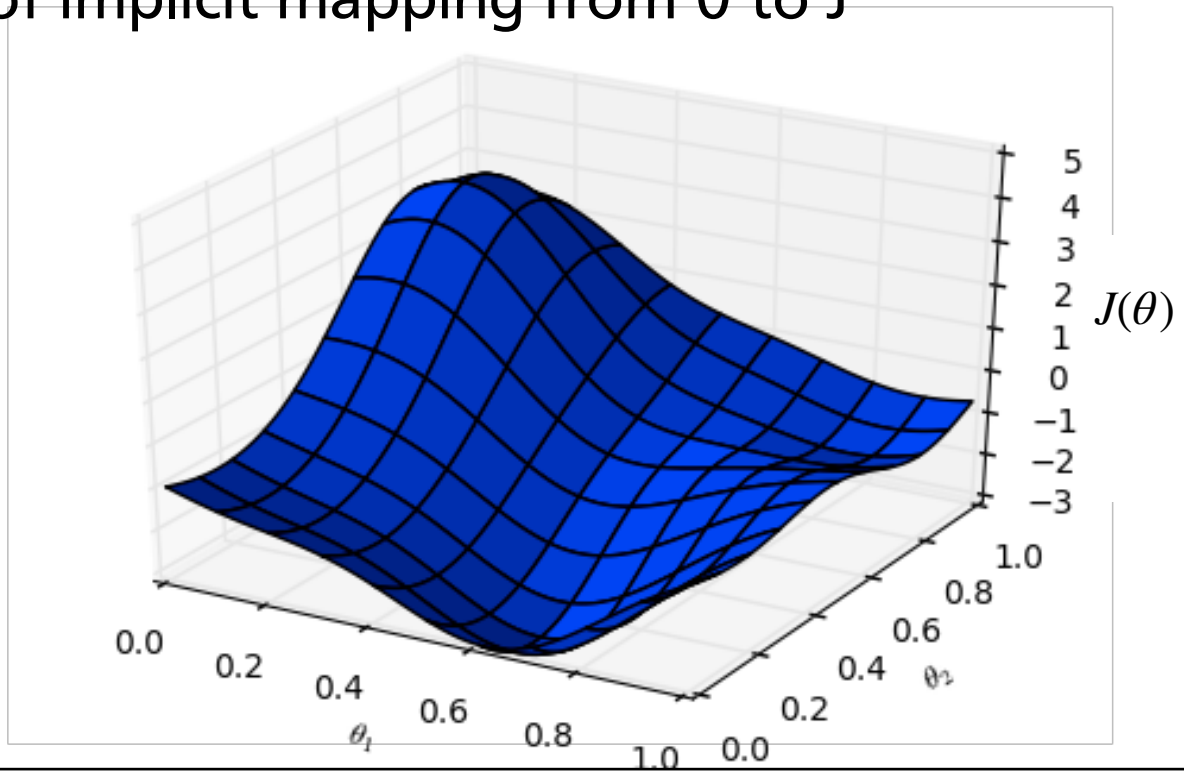


Objective

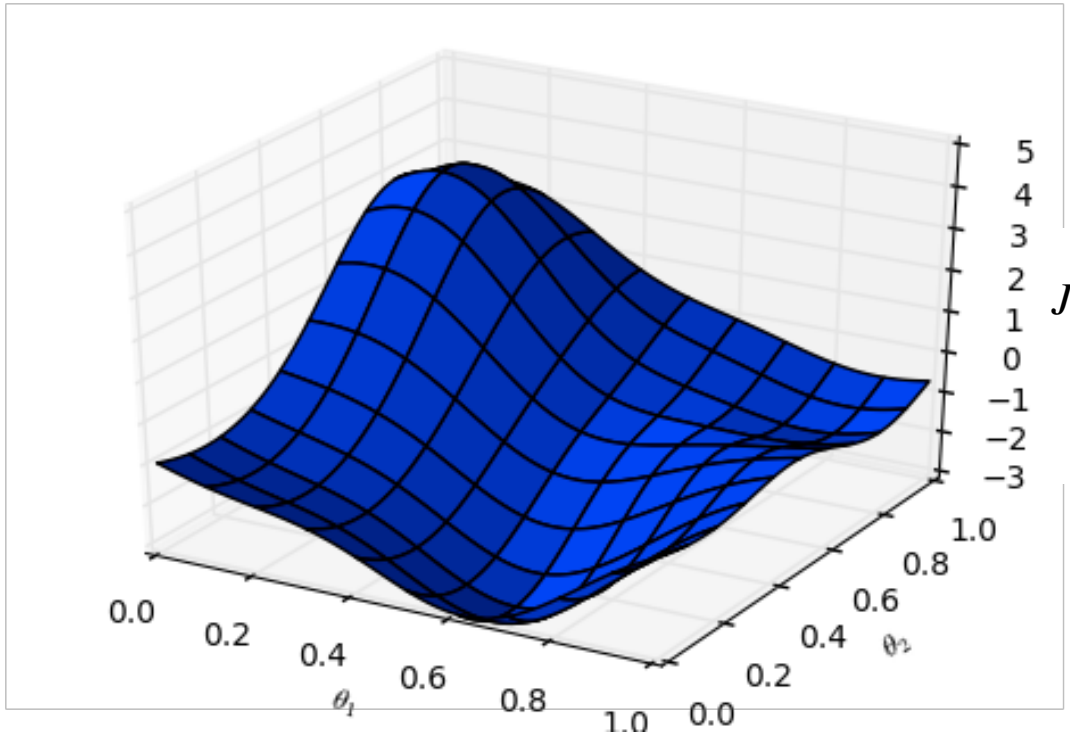
Every policy is defined by a parameter vector θ

Every policy as an expected return J

Can think of implicit mapping from θ to J

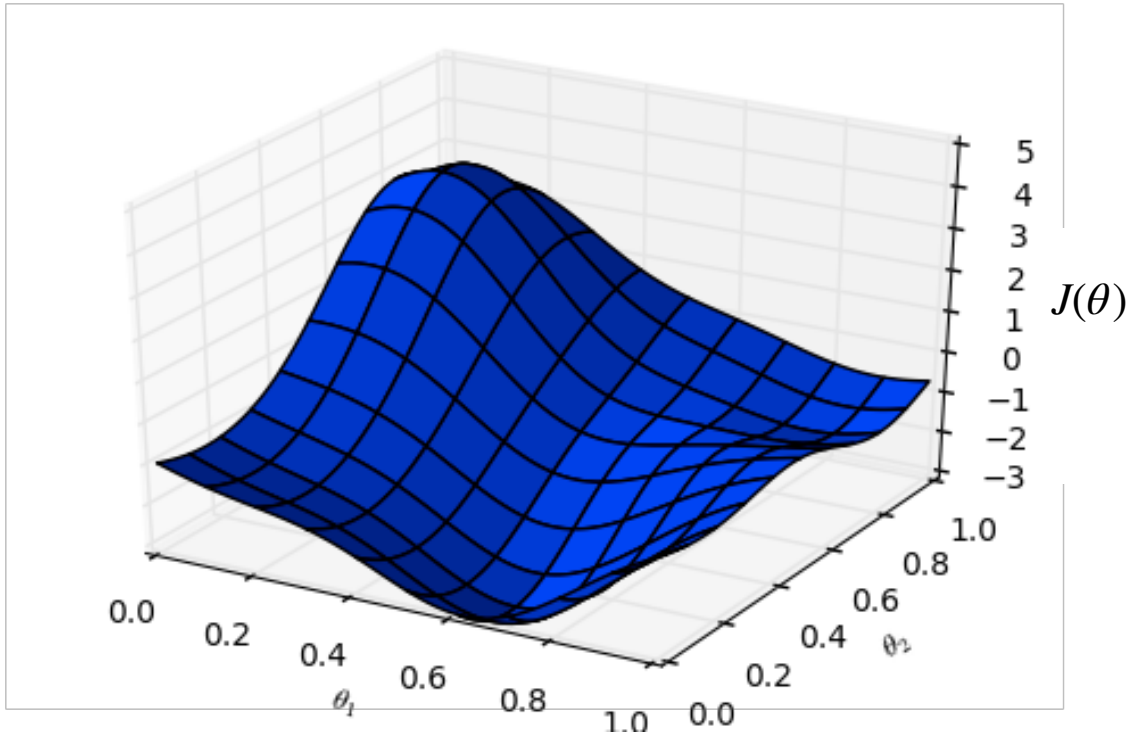


How do we find the best policy



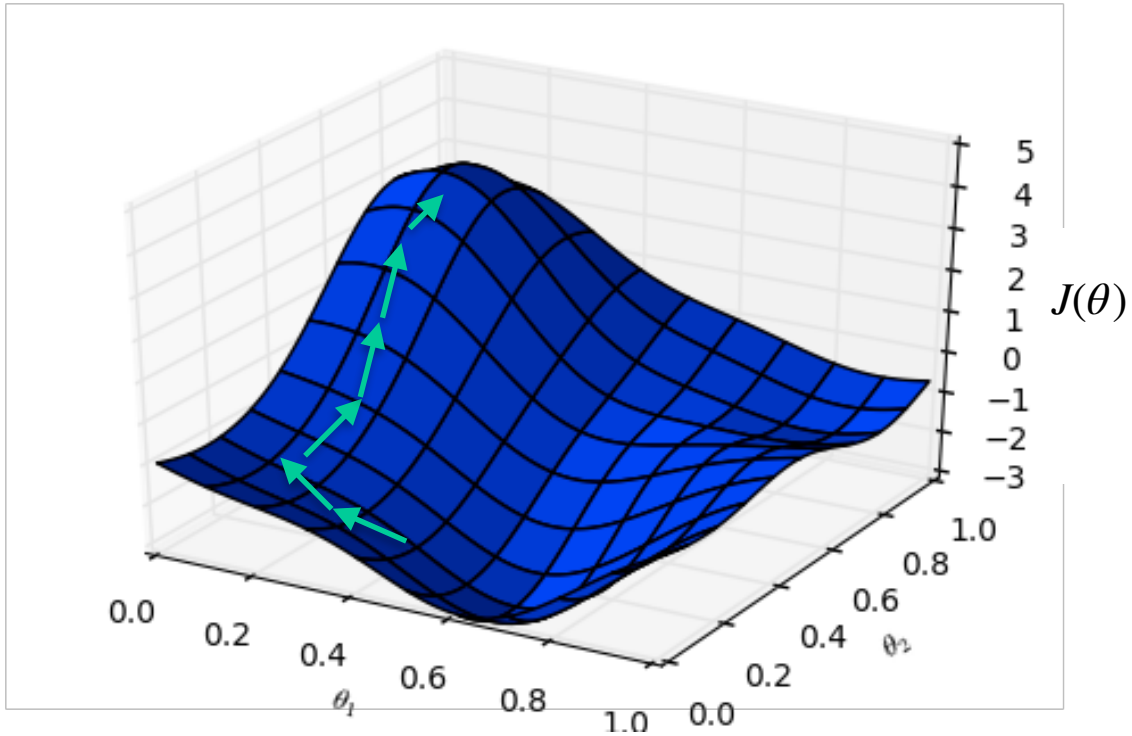
How do we find the best policy

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



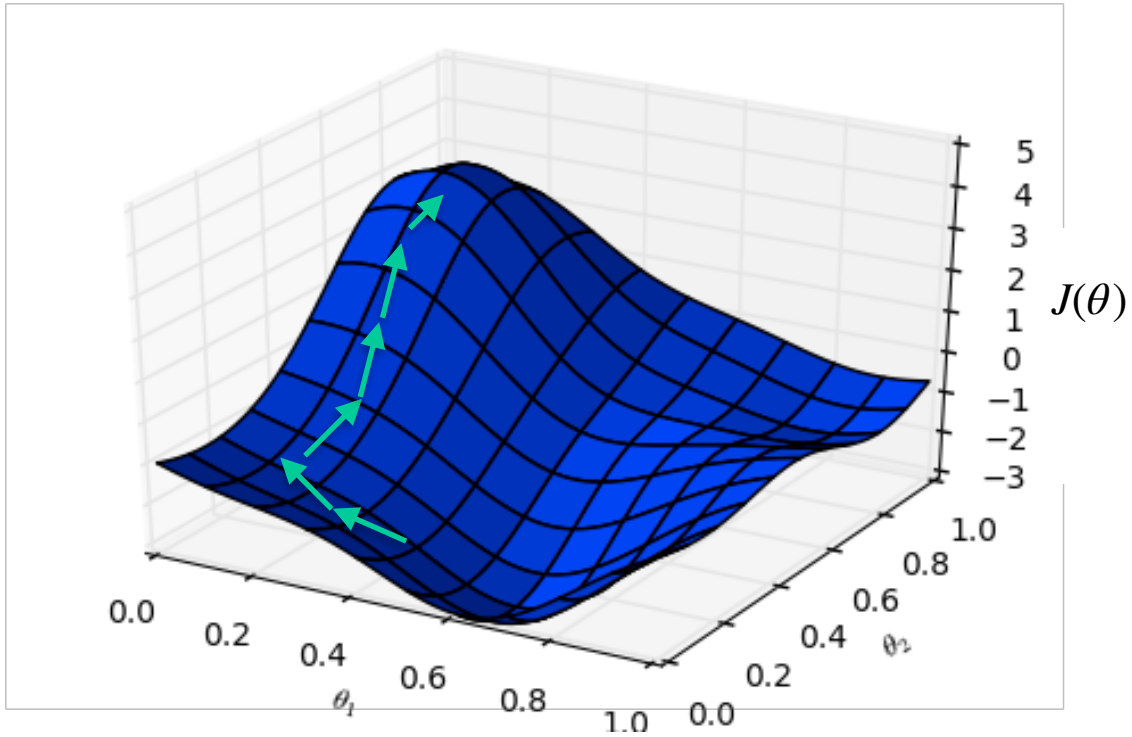
How do we find the best policy

- 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

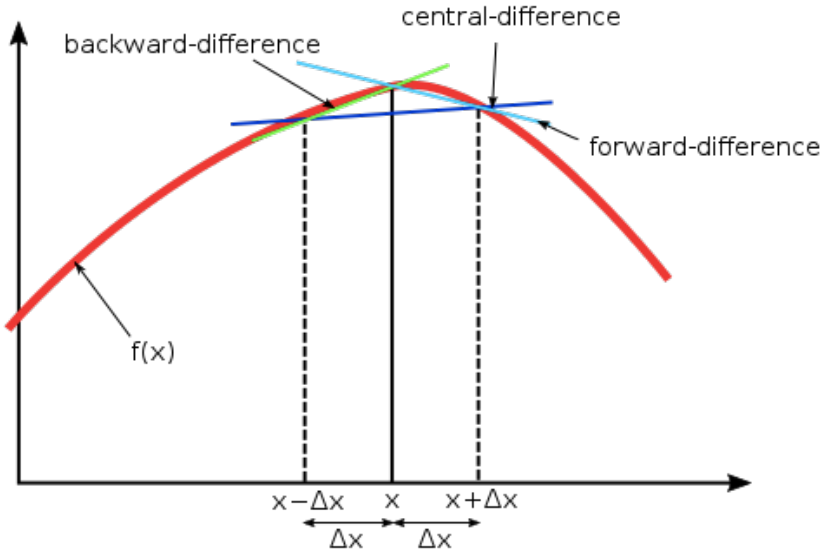
Evaluate two controllers, with parameters $\theta + \epsilon$ and $\theta - \epsilon$

Estimate gradient as

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

Update parameters

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



By Kakitc - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=63327976

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

Finite difference gradients

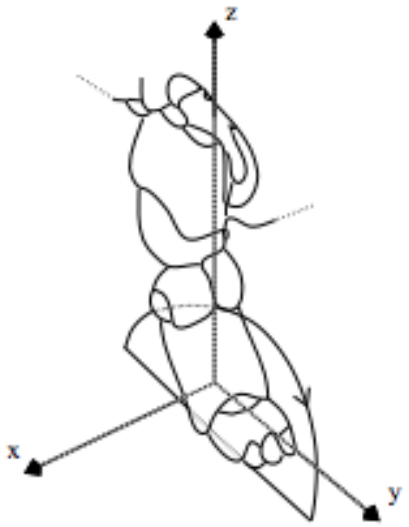


Fig. 2. The elliptical locus of the Aibo's foot. The half-ellipse is defined by length, height, and position in the x-y plane.

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

Why policy-based methods

- handle continuous actions

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

- ensure smoothness in policies

Small step size ensures small change in policy

- small errors in V not the same as small errors in policy

Directly optimise quantity of interest

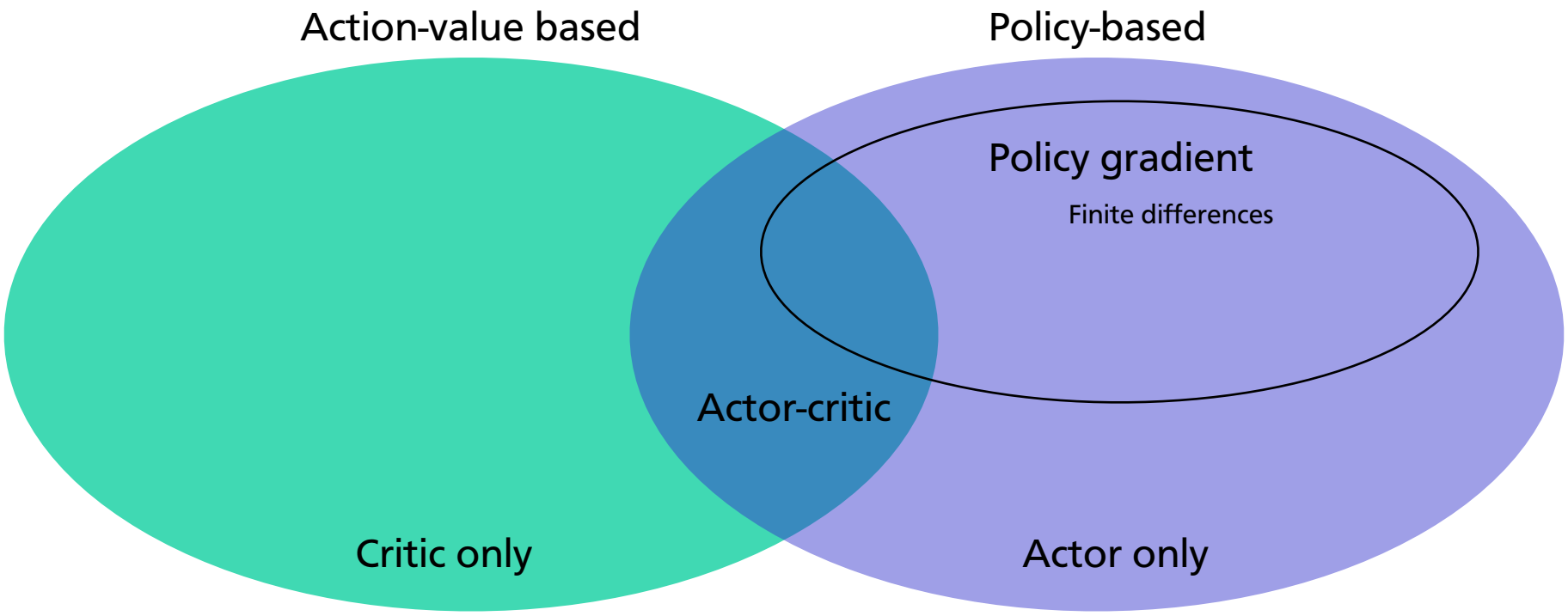
- hard to include prior knowledge about possible solutions

Include prior knowledge as policy form or initialisation

- can’t learn stochastic policies

Can easily train stochastic policies

Policies and action-values



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

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

REINFORCE

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

$$\nabla_{\boldsymbol{\theta}} J = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\tau} G(\tau)$$

$$= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) G(\tau) d\tau$$

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?

REINFORCE

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

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

REINFORCE

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

Sample N trajectories (N=1 is possible)

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

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

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

Easy to implement!

REINFORCE

The REINFORCE gradient uses knowledge of the policy.

- This makes it more efficient then finite differences

However

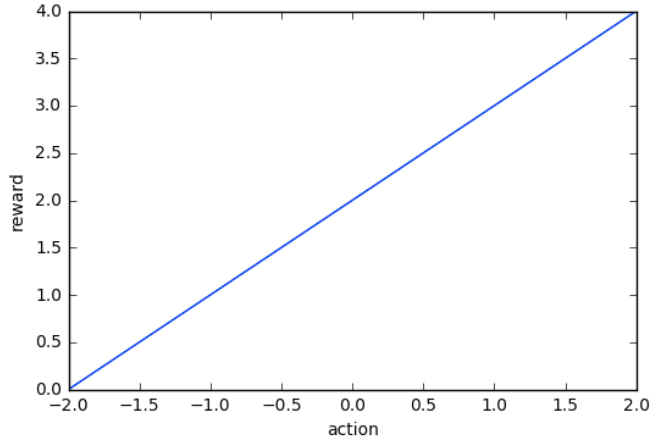
- Policy needs to be differentiable!
- Efficiency is still low!

Inefficient calculation of gradient

One time step, one state

$$r = a + 2$$

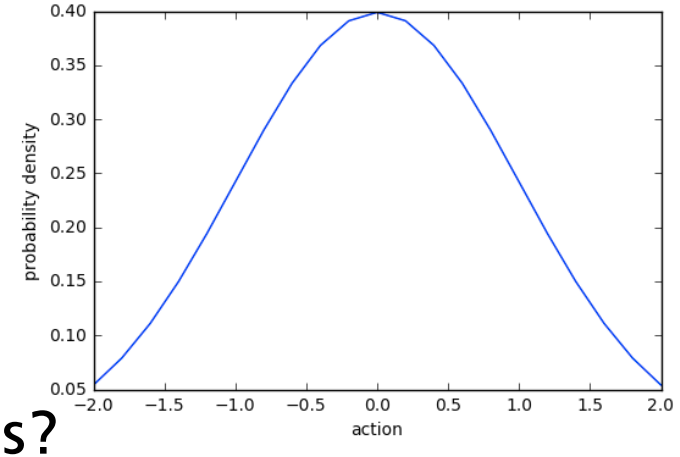
$$a \sim \mathcal{N}(\theta; 1) \;\; \nabla_\theta \log \pi(a) = a - \theta$$



Batch 1: {a=-1; r=1}

$$\nabla_\theta J = r \nabla_\theta \log \pi(a)$$

$$= 1(-1 - 0) = -1$$



Batch 2: {a=1; r=3}

$$\nabla_\theta J = 3(1 - 0) = 3$$

What if we subtracted 2 from all rewards?

Inefficient calculation of gradient

What if we subtracted 2 form all rewards?

Dataset 1: {a=-1; r=-1}

$$\nabla_\theta J = -1(-1-0) = 1$$

Dataset 2: {a=1; r=1}

$$\nabla_\theta J = 1(1-0) = 1$$

- Average gradient does not seem to change
- Variance is now lower
- Estimates less erratic, could probably use higher learning rate

Inefficient calculation of gradient

Does average indeed not change?

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

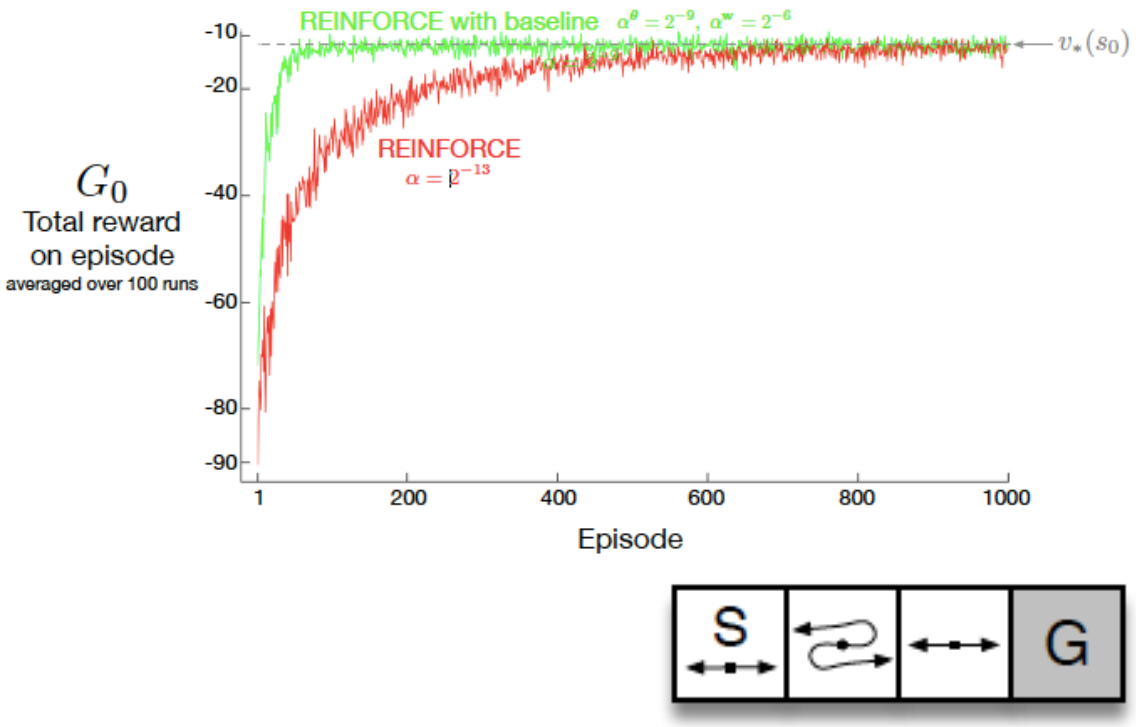
Thus, the gradients are still **unbiased**

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

Inefficient calculation of gradient

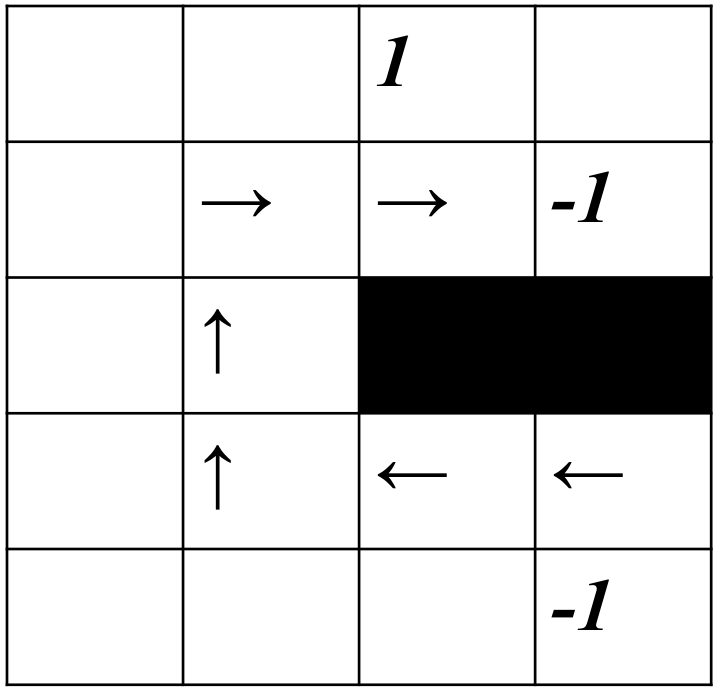
A good baseline is the expected reward (e.g. observed average)

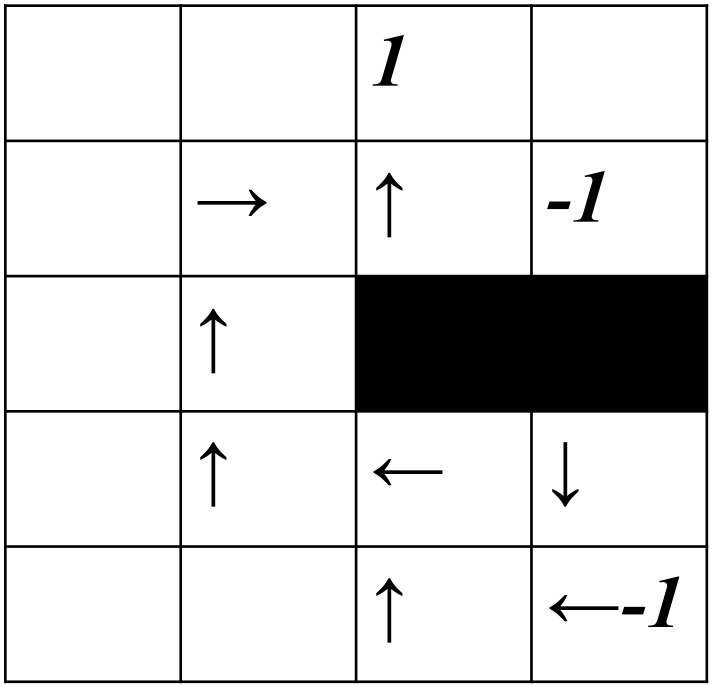
Baselines can depend on state. Then, a value function estimate is a popular choice.



Sutton & Barto. Reinforcement Learning: An introduction.

Inefficient credit assignment





REINFORCE

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

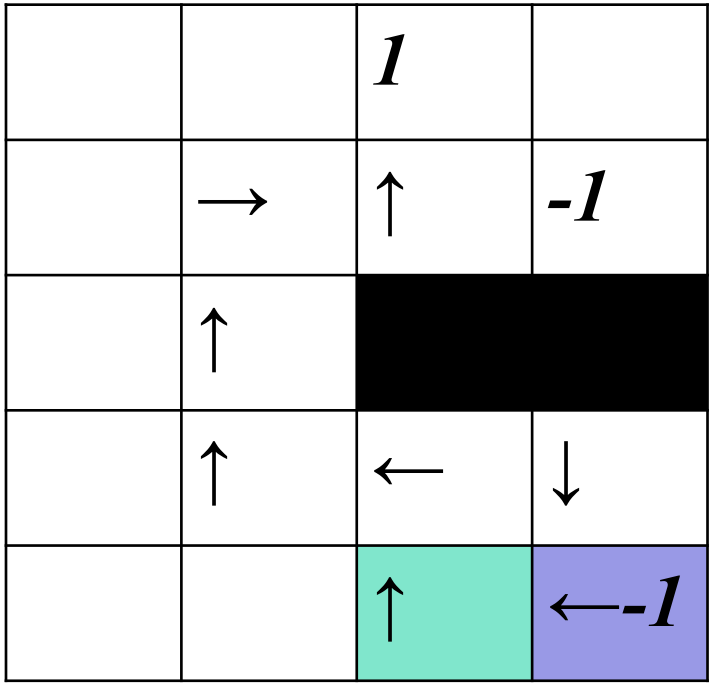
Comparison


Finite differences
(Blackbox gradients)
REINFORCE
(Whitebox gradients)
Episode-based
Exploration
Step-based

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?



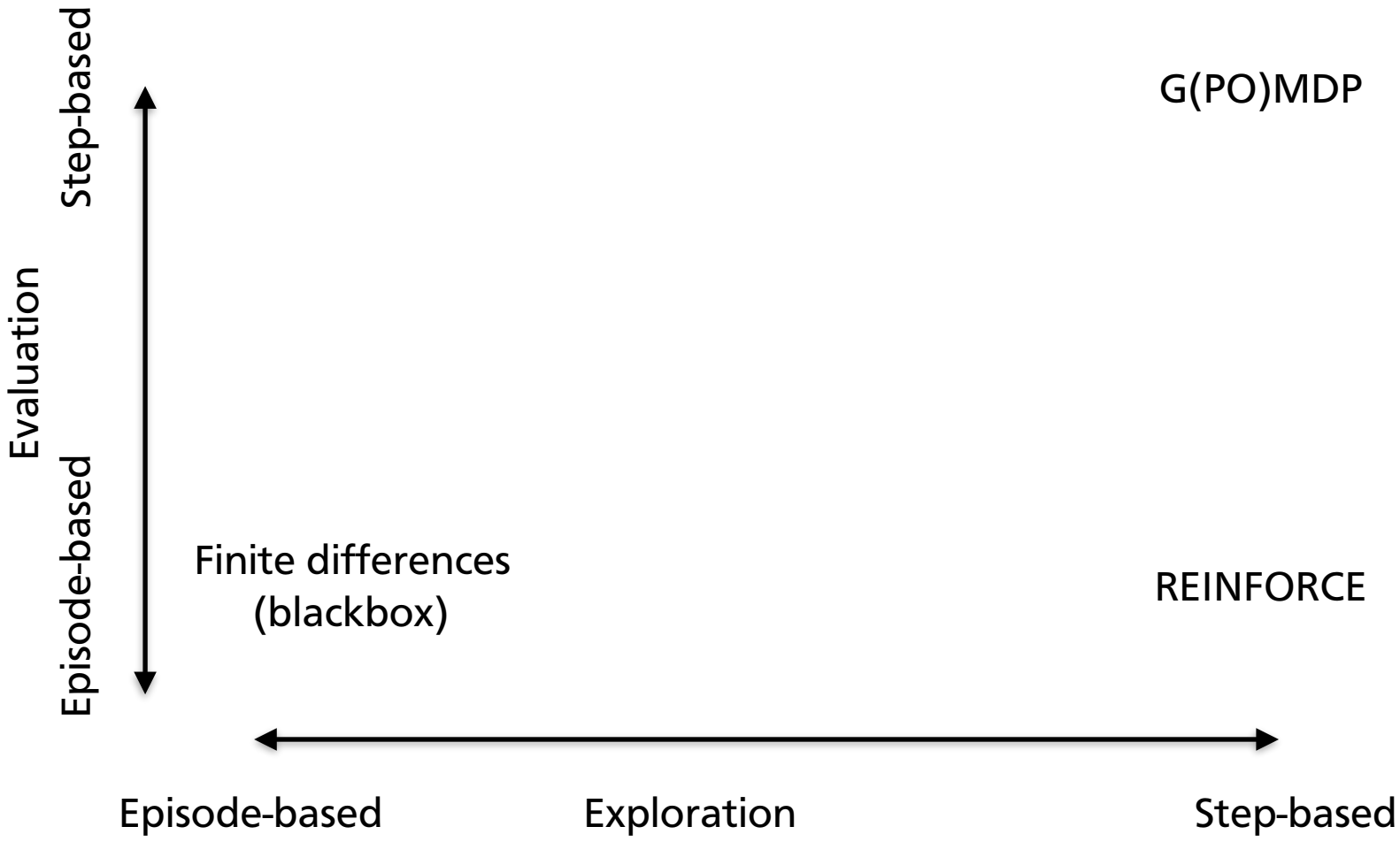
G(PO)MPD

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

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

- 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
- Still: variance from stochastic transitions increases with T

Comparison


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

Policy Gradient Theorem and Actor critic

By shuffling the terms around, we can re-formulate G(PO)MDP

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

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

Replace return by its expected value

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

	a_1	a_2	a_3
r_1	→		
r_2	→	→	
r_3	→	→	→

	a_1	a_2	a_3
r_1	↓		
r_2	↓	↓	
r_3	↓	↓	↓

PGT and Actor critic

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

State-action pairs at each time step will contribute equally to the total gradient. So this is equivalent to an expectation over the on-policy distribution over (s,a)

$$\nabla J = \int_{\mathcal{S}} \mu(\mathbf{s}) \int_{\mathcal{A}} \underbrace{\pi(\mathbf{a}|\mathbf{s}) \nabla \log \pi(\mathbf{a}|\mathbf{s})}_{\nabla \pi(\mathbf{a}|\mathbf{s}} q_\pi(\mathbf{s}, \mathbf{a}) d\mathbf{s} d\mathbf{a}$$

Formal proof, also for the continuing case:

Sutton et al., Policy Gradient Methods for Reinforcement Learning with Function Approximation

PGT and Actor critic

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

Maybe we can replace q_π by an estimate of the q_w function?

Advantage: break the high variance of monte-carlo returns

Disadvantage: possibly include bias?

PGT and Actor critic

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

Maybe we can replace q_π by an estimate of the q_w function?

Advantage: break the high variance of monte-carlo returns

Disadvantage: possibly include bias?

This algorithm has a parametrised policy *and* a parametrized action-value function, with parameters θ and *w,* respectively.

The policy is often called an actor while the value function is called critic. A method that uses both is an *actor-critic* method.

Compatible function approximation thm.

Disadvantage of actor-critic: possibly include bias?

Question: are there functions q_π that give an unbiased estimate?

$$\nabla J = \mathbb{E}_\tau \left[\sum_{t'=1}^{T} \nabla \log \pi(\mathbf{a}_{t'} | \mathbf{s}_{t'}) q_\pi(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right] \stackrel{?}{=} \mathbb{E}_\tau \left[\sum_{t'=1}^{T} \nabla \log \pi(\mathbf{a}_{t'} | \mathbf{s}_{t'}) \hat{q}_{\mathbf{w}}(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right]$$

Compatible function approximation thm.

Disadvantage of actor-critic: possibly include bias?

Question: are there functions q_π that give an unbiased estimate?

$$\nabla J = \mathbb{E}_\tau \left[\sum_{t'=1}^{T} \nabla \log \pi(\mathbf{a}_{t'}|\mathbf{s}_{t'}) q_\pi(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right] \stackrel{?}{=} \mathbb{E}_\tau \left[\sum_{t'=1}^{T} \nabla \log \pi(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \hat{q}_{\mathbf{w}}(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right]$$

Answer: Yes, if value function is 'compatible':

$$\nabla_{\mathbf{w}} \hat{q}_{\mathbf{w}} = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a}) \quad \text{e.g. } \hat{q}_{\mathbf{w}} = \mathbf{w}^T \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$$

and

$$\mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} \left(q_\pi(\mathbf{s}, \mathbf{a}) - \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \right) \frac{\partial \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a})}{\partial \mathbf{w}} = 0$$

$$\text{e.g. } \mathbf{w} = \arg\min_{\mathbf{w}} \mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} \left(q_\pi(\mathbf{s}, \mathbf{a}) - \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \right)^2$$

Compatible function approximation

Proof:

1) $\nabla_{\mathbf{w}} \hat{q}_{\mathbf{w}} = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{s}, \mathbf{a})$

2) $\mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} \left(q_\pi(\mathbf{s}, \mathbf{a}) - \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \right) \frac{\partial \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a})}{\partial \mathbf{w}} = 0$

Substitute (1) into (2)

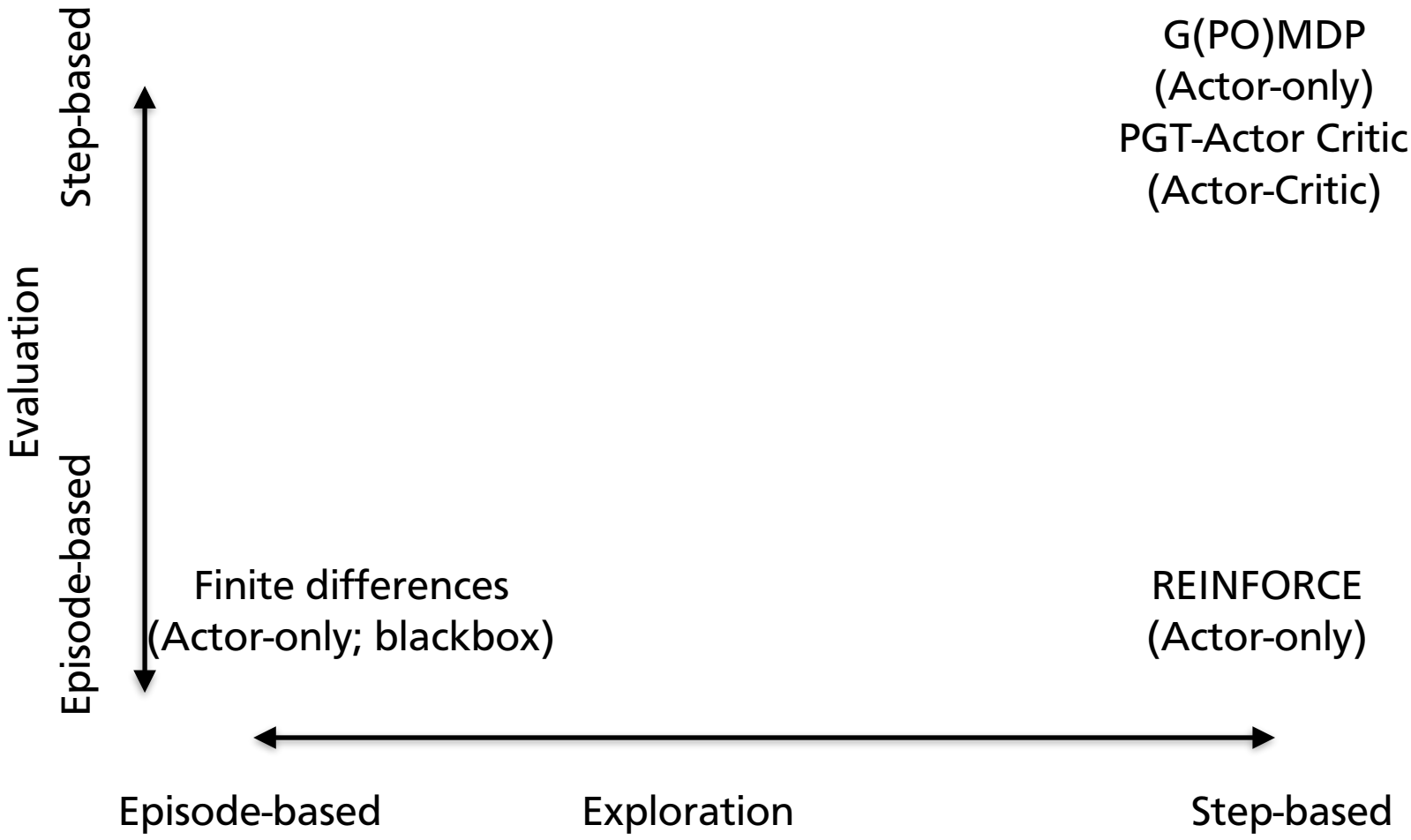
$$\mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} \left(q_\pi(\mathbf{s}, \mathbf{a}) - \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \right) \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{a}|\mathbf{s}) = 0$$

$$\mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} q_\pi(\mathbf{s}, \mathbf{a}) \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{a}|\mathbf{s}) = \mathbb{E}_{\mathbf{s}\sim\mu, \mathbf{a}\sim\pi(\cdot,\mathbf{s})} \hat{q}_{\mathbf{w}}(\mathbf{s}, \mathbf{a}) \nabla_{\boldsymbol{\theta}} \log \pi(\mathbf{a}|\mathbf{s})$$

Again yielding a **unbiased** and **consistent** estimate

This can again be combined with a **baseline**

Comparison

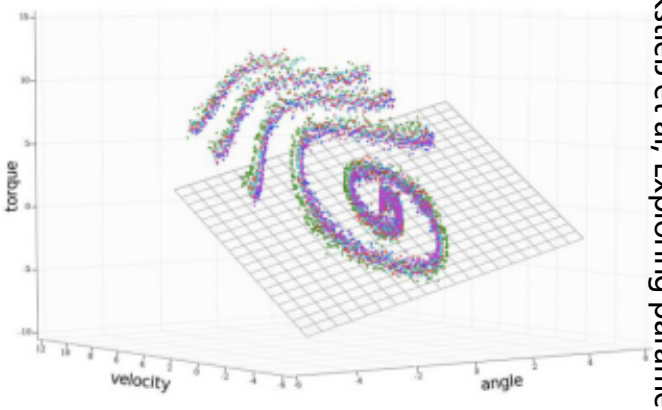

Step-based
Evaluation
Episode-based
G(PO)MDP
(Actor-only)
PGT-Actor Critic
(Actor-Critic)
Finite differences
(Actor-only; blackbox)
REINFORCE
(Actor-only)
Episode-based
Exploration
Step-based

PEPG

Using stochastic policies, typically in various roll-outs the trajectories are similar with minor perturbations.

Also: perturbations cannot be be repeated by any deterministic policy.

Perturbation in every time step does not explore much, and can damage e.g. robots.



Rückstieß et al, Exploring parameter space in reinforcement learning, 2010.

PEPG

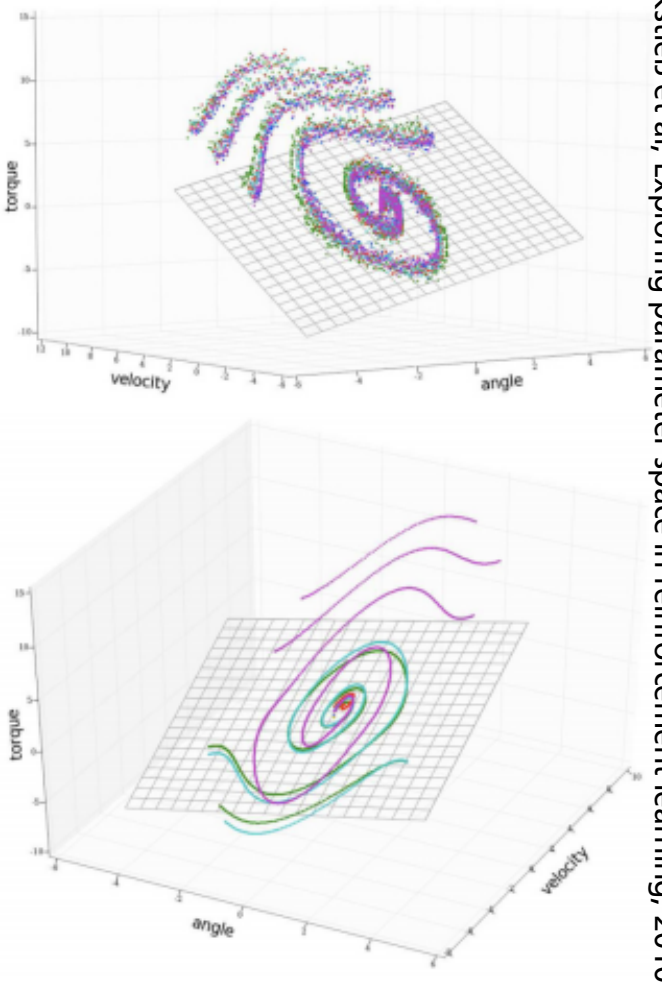
Instead, randomise the *policy parameters* at the beginning and then keep them constant

Need a search distribution

$$p(\boldsymbol{\theta}|\boldsymbol{\nu})$$

where we want to learn parameters ν

Policy is now a deterministic function $\mathbf{a} = \pi_{\boldsymbol{\theta}}(\mathbf{s})$



Rückstieß et al, Exploring parameter space in reinforcement learning, 2010.

PEPG

Can look at this set-up as a single time-step with

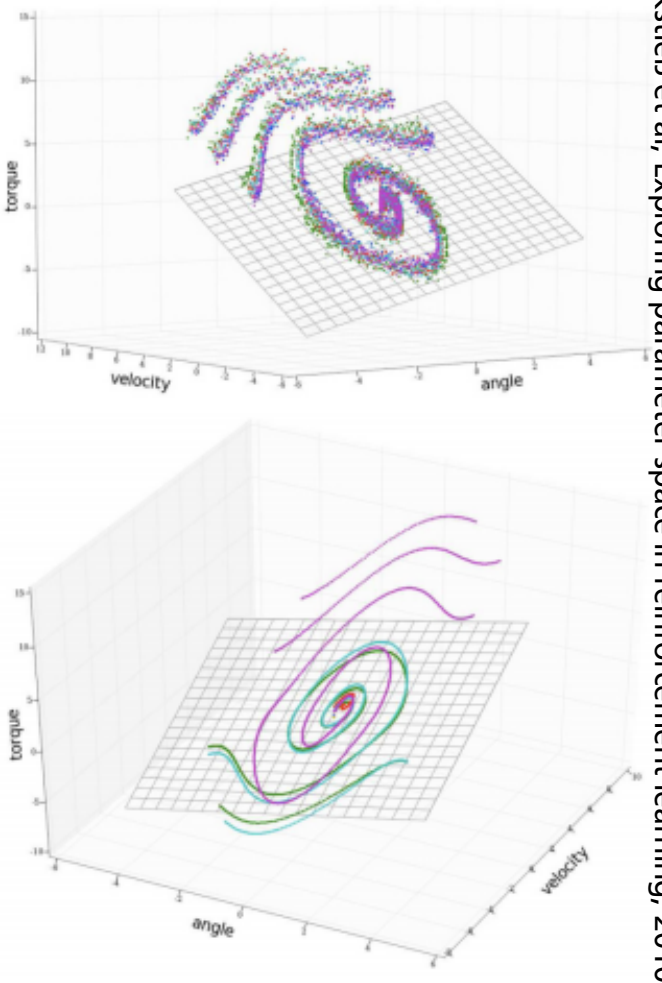
$$a_{\text{new}} = \boldsymbol{\theta}$$

And the rest of the dynamics encoded in

$$r_{\text{new}} = G(\boldsymbol{\theta})$$

Then, we are simply doing REINFORCE on this new MDP

$$\begin{aligned}\nabla_{\boldsymbol{\nu}} J &= \mathbb{E}_{a_{\text{new}}} r_{\text{new}} \nabla_{\boldsymbol{\nu}} \log p(\mathbf{a}_{\text{new}}; \boldsymbol{\nu}) \\ &= \mathbb{E}_{\boldsymbol{\theta}} G(\boldsymbol{\theta}) \nabla_{\boldsymbol{\nu}} \log p(\boldsymbol{\theta}; \boldsymbol{\nu})\end{aligned}$$



Rückstieß et al, Exploring parameter space in reinforcement learning, 2010.

PEPG

Can look at this set-up as a single time-step with

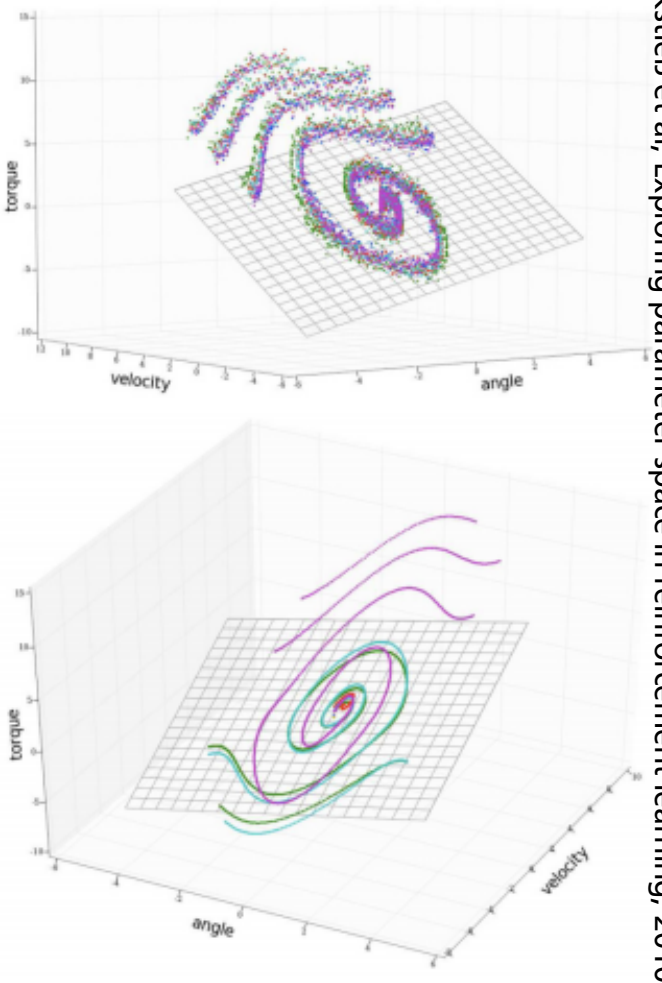
$$a_{\text{new}} = \boldsymbol{\theta}$$

And the rest of the dynamics encoded in

$$r_{\text{new}} = G(\boldsymbol{\theta})$$

Then, we are simply doing REINFORCE on this new MDP

$$\begin{aligned}\nabla_{\boldsymbol{\nu}} J &= \mathbb{E}_{a_{\text{new}}} r_{\text{new}} \nabla_{\boldsymbol{\nu}} \log p(\mathbf{a}_{\text{new}}; \boldsymbol{\nu}) \\ &= \mathbb{E}_{\boldsymbol{\theta}} G(\boldsymbol{\theta}) \nabla_{\boldsymbol{\nu}} \log p(\boldsymbol{\theta}; \boldsymbol{\nu})\end{aligned}$$



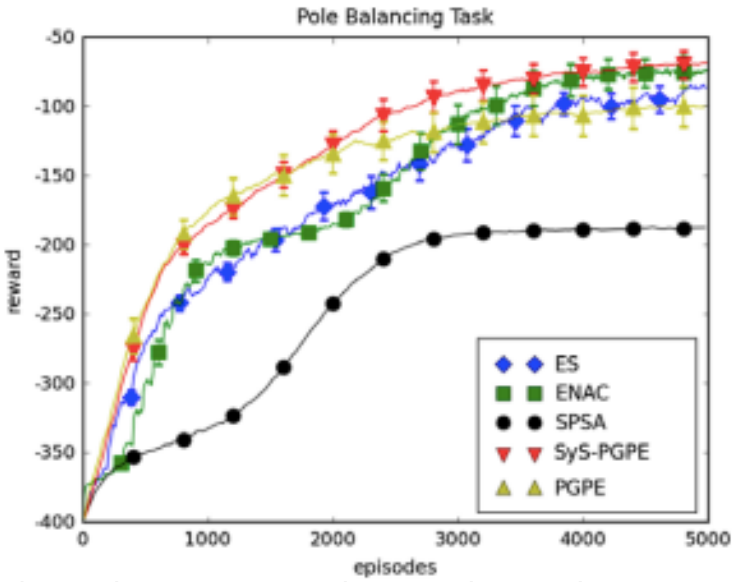
Rückstieß et al, Exploring parameter space in reinforcement learning, 2010.

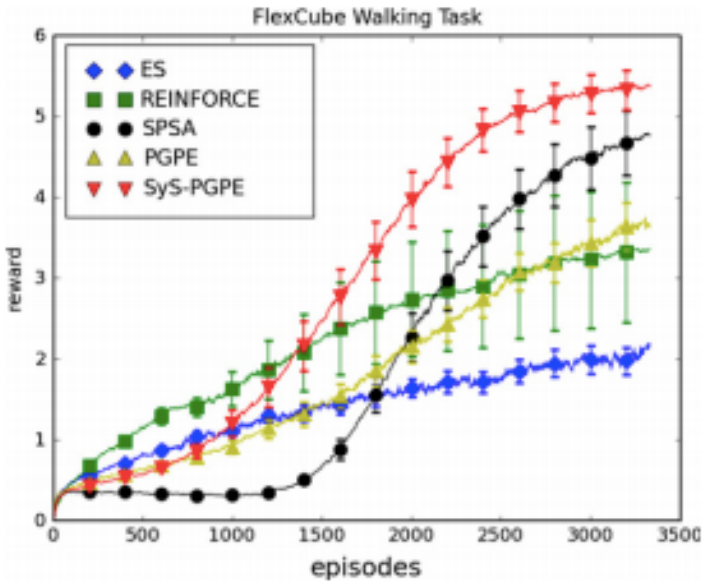
PEPG performance

PEPG can help explore more widely

Enforcing samples to be symmetric has additional benefit

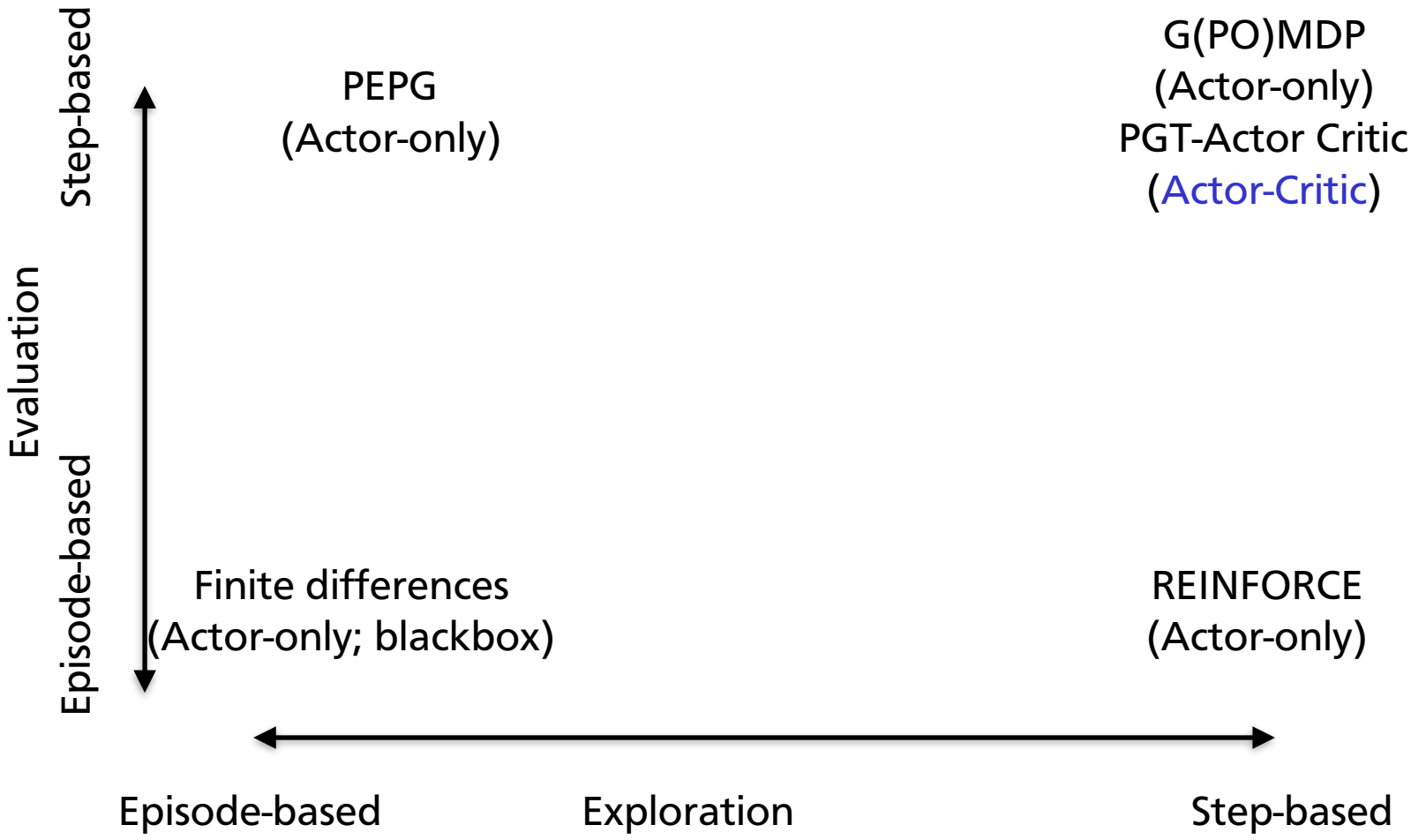
However, can only try one strategy per rollout (like finite difs)



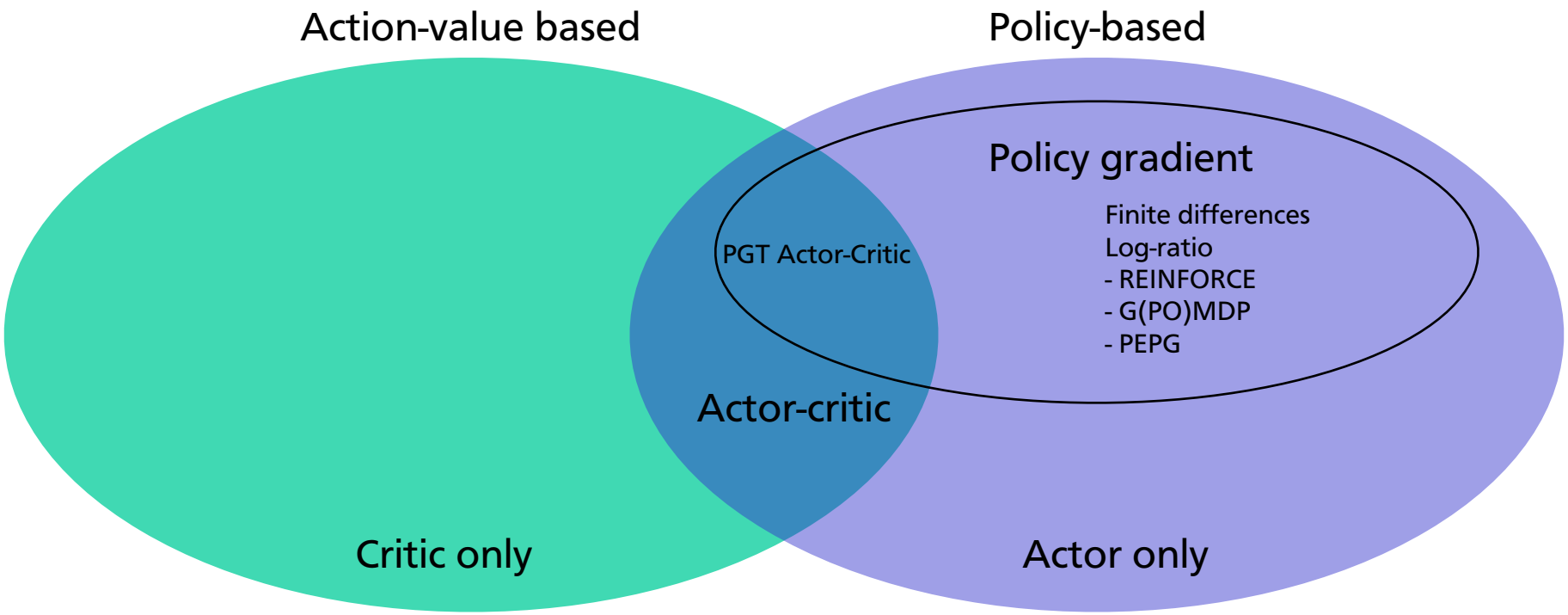


Sehnke et al, Parameter-exploring policy gradients, 2008.
Rückstieß et al, Exploring parameter space in reinforcement learning, 2010.

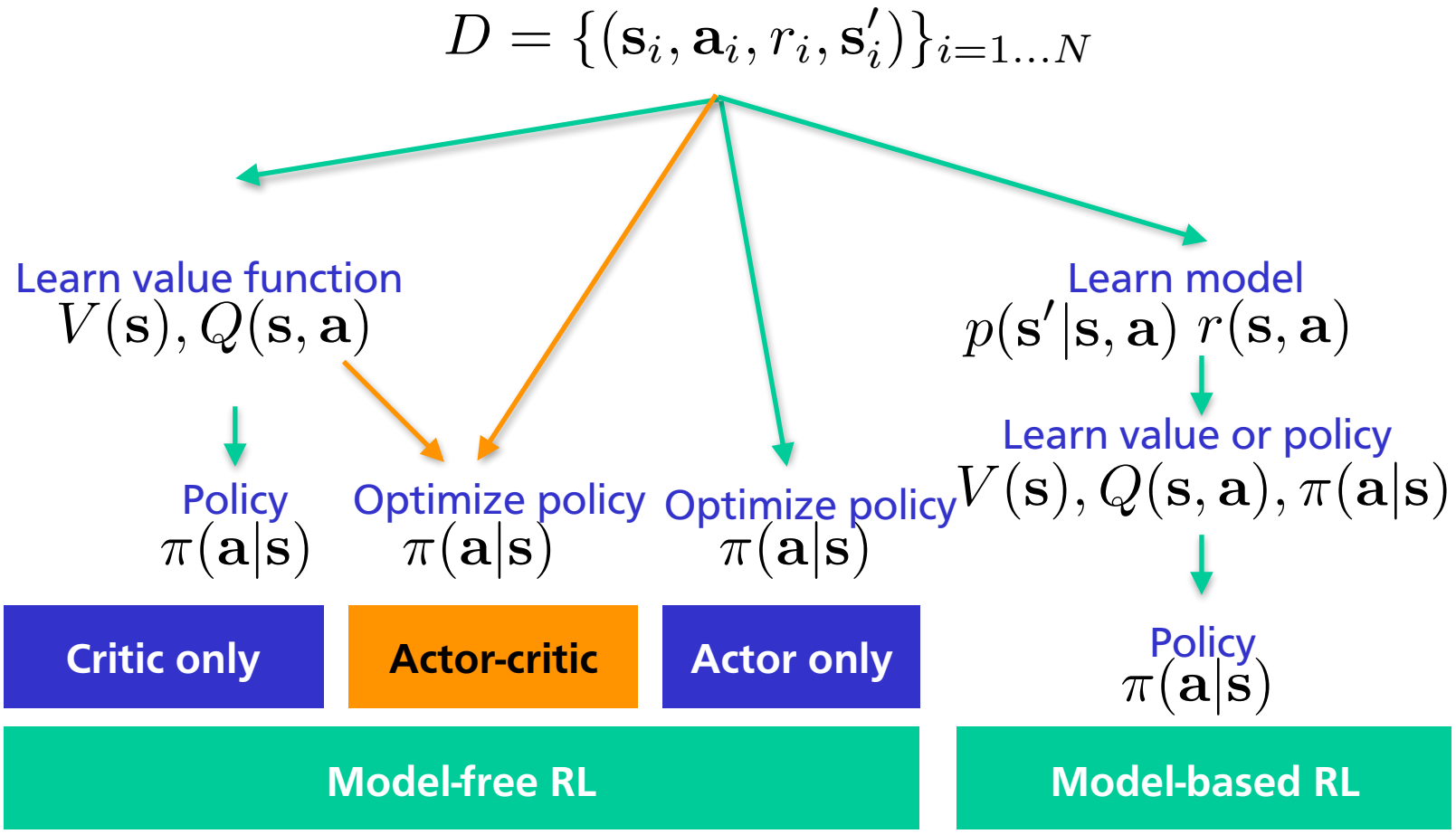
Comparison


Evaluation
Step-based
Episode-based
PEPG
(Actor-only)
G(PO)MDP
(Actor-only)
PGT-Actor Critic
(Actor-Critic)
Finite differences
(Actor-only; blackbox)
REINFORCE
(Actor-only)
Episode-based
Exploration
Step-based

Policies and action-values



Big picture: How to learn policies



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

- small errors in V not the same as small errors in policy

Directly optimise quantity of interest

- 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

- Actor-critic lowers variance using critic

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

- Actor-critic can be formulated for continuing setting

Actor-critic can be ‘fiddly’, many moving parts

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