Reinforcement Learning

Lecture 9

Policy Gradient

Finite Differences

REINFORCE

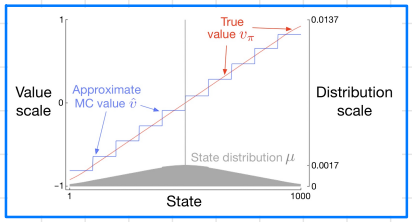
Actor-Critic Methods

Robert Peharz

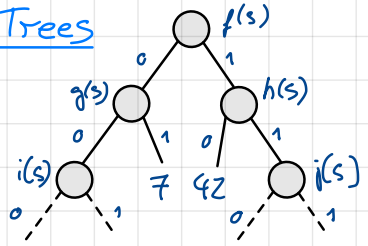
Institute of Theoretical Computer Science
Graz University of Technology
Winter Term 2023/24

Function Approximators

- aggregated states



- Decision Trees

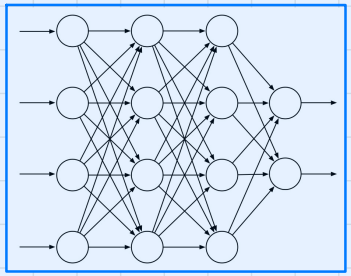


- Linear

$$\hat{v}(s) = \sum_{i=1}^{K} w_i \, x_i(s)$$

$$\hat{q}(s,a) = \sum_{i=1}^{K} w_i \, x_i(s,a)$$

- Neural Network



Differentiable Approximators

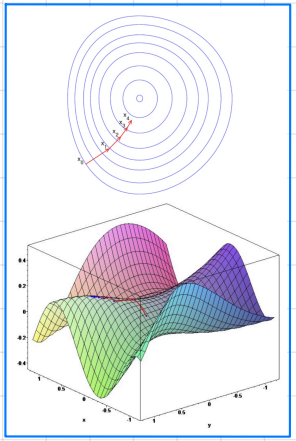
$$\nabla_{\underline{w}} \hat{v}(s,\underline{w}) := \begin{pmatrix} \frac{\partial \hat{v}(s,\underline{w})}{\partial w_1} \\ \frac{\partial \hat{v}(s,\underline{w})}{\partial w_2} \\ \vdots \\ \frac{\partial \hat{v}(s,\underline{w})}{\partial w_d} \end{pmatrix}$$

$$\nabla_{\underline{w}} \hat{q}(s,a,\underline{w}) := \begin{pmatrix} \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_1} \\ \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_2} \\ \vdots \\ \frac{\partial \hat{q}(s,a,\underline{w})}{\partial w_d} \end{pmatrix}$$

Gradient Descent for Supervised Learning

Stochastic Gradient Descent (SGD)

init $\underline{w}$
repeat
$\underline{w} \leftarrow \underline{w} - \alpha \nabla f_{\underline{w}}(\underline{w})$ (stepsize α)
until $\nabla f_{\underline{w}}(\underline{w}) \approx 0$



Value Function Approximation

$$l(\hat{v}(s,\underline{w}),\ v_\pi(s)) = \frac{1}{2}\left(v_\pi(s) - \hat{v}(s,\underline{w})\right)^2$$

$$\nabla_{\underline{w}}\, l = -\left(v_\pi(s) - \hat{v}(s,\underline{w})\right) \nabla_{\underline{w}}\, \hat{v}(s,\underline{w})$$

(P)rediction

$$l(\hat{q}(s,a,\underline{w}),\ q_\pi(s,a,\underline{w})) = \frac{1}{2}\left(q_\pi(s,a,\underline{w}) - \hat{q}(s,a,\underline{w})\right)^2$$

$$\nabla_{\underline{w}}\, l = -\left(q_\pi(s,a) - \hat{q}(s,a,\underline{w})\right) \nabla_{\underline{w}}\, \hat{q}(s,a,\underline{w})$$

(C)ontrol

$$\mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s,\underline{w})\right)^2\right] \ / \ \mathbb{E}_\pi\left[\left(q_\pi(s,a) - \hat{q}(s,a,\underline{w})\right)^2\right]$$

quadratic loss

Monte Carlo

$$\underline{w} \leftarrow \underline{w} + \alpha\left(G - \hat{v}(S_t,\underline{w})\right) \nabla_{\underline{w}}\, \hat{v}(S_t,\underline{w})$$

TD

$$\underline{w} \leftarrow \underline{w} + \alpha\left[R + \gamma\, \hat{v}(S',w) - \hat{v}(S,\underline{w})\right] \nabla_{\underline{w}}\, \hat{v}(S,\underline{w})$$

Linear Function Approximation

Easy Gradient, Convex Loss

$$\hat{v}(s,\underline{w}) = \underline{w}^T \underline{x} = \sum_{i=1}^{d} w_i\, x_i(s)$$

$$\nabla_{\underline{w}}\, \hat{v} = \underline{x}$$

$$\hat{q}(s,a,\underline{w}) = \underline{w}^T \underline{x} = \sum_{i=1}^{d} w_i\, x_i(s,a)$$

$$\nabla_{\underline{w}}\, \hat{q} = \underline{x}$$

Convergence of Algorithms

Prediction

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

Control

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning *	✓	✓	✗

(✓) = chatters around near-optimal value function

Policy Gradient Methods

(last time)

(today)

What are Policy Gradient Methods?

- so far, we focused on value functions v_π and q_π
- last lecture, we approximated value functions with differentiable approximators $\hat{v}(s, \underline{w})$, $\hat{q}(s, \underline{w})$ and learned them with stochastic gradient descent (SGD)
- Policy Gradient: parametrize policy with a vector $\underline{\theta}$

$$\pi(a|s) := \pi(a|s, \underline{\theta})$$

and maximize a suitable performance measure $J(\underline{\theta})$ with gradient ascend:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\underline{\theta})$$

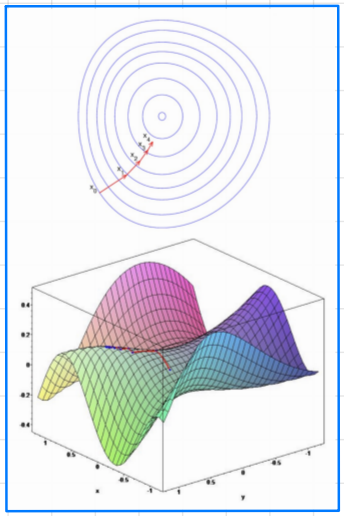


Image: D. Silver

Pros of Policy Gradient Methods

- directly learn $\hat{\pi}$ rather than "indirectly" via q (data efficiency)
- better convergence properties ——— non-linear value function approximators often unstable
- sometimes $\hat{\pi}(a|s,\underline{\theta})$ is easier to specify than $\hat{v}(s,\underline{w})$, $\hat{q}(s,a,\underline{w})$
- $\hat{\pi}(a|s,\underline{\theta})$ can encode <u>domain knowledge</u> (e.g. physical constraints), or be combined of pre-specified sub-policies (e.g. robot controlers)
- $\hat{\pi}(a|s,\underline{\theta})$ can learn <u>stochastic</u> policies, which is important in <u>partially observed</u> MDPs (POMDPs)

<u>Example</u>: aliased grid world, <u>grey states are indistinguishable</u>

- deterministic policy fails in 50% of the trials
- <u>optimal</u>: go left/right with probability 0.5 on grey states

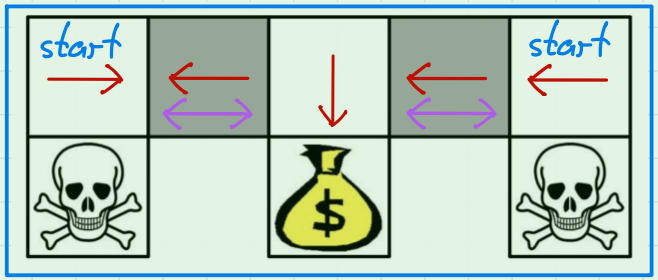


Image: D. Silver

Cons of Policy Gradient Methods

- local maxima — value-based RL (GPI) converges in principle to a globally optimal policy, while policy gradient typically gets stuck in local optima

Example [Jan Peters, scholarpedia.org]

Policy initialized with "straddle jump" won't find the "Fosbury flop"

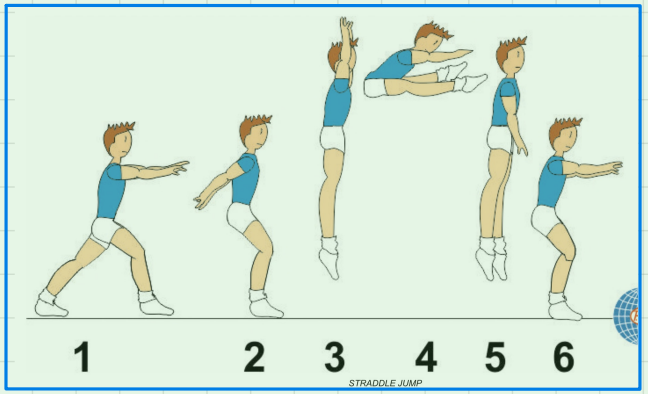


Image: fig-aerobic.com

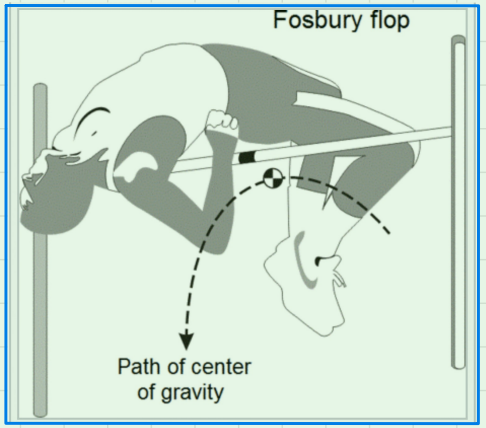


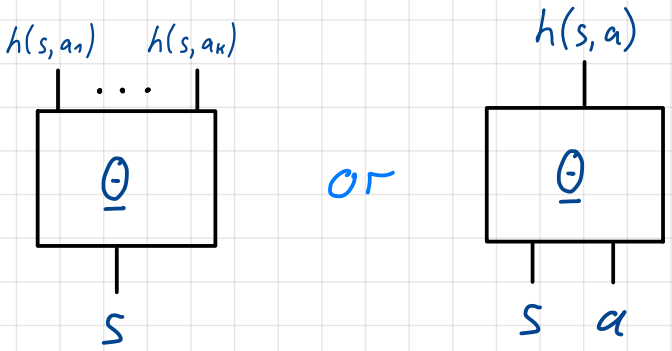
Image: wikipedia.org

- ignores information about the environment (robustness, transferability)
- thus, might generalize poorly to (slightly) changed environments

Softmax Policy

generic parametrization for finite $\mathcal{A}$

- specify "numeric preference score" $h(s, a, \underline{\theta})$, which is large if a should be preferred in state s, and low if it should be avoided.
- linear I: specify non-linear features $x_i(s, a)$ and let
$$h(s, a, \underline{\theta}) = \sum_i \theta_i \, x_i(s, a)$$
- linear II: specify non-linear features $x_i(s)$ and let
$$h(s, a, \underline{\theta}) = \sum_i \theta_{ia} \, x_i(s)$$
- neural network:



Softmax policy

$$\hat{\pi}(a|s, \underline{\theta}) := \frac{e^{h(s, a, \underline{\theta})}}{\sum_{a'} e^{h(s, a', \underline{\theta})}}$$

(= softmax layer in NNs)

(note that $\hat{\pi} > 0$ and $\sum_a \hat{\pi}(a|s) = 1$)

Behavioural Cloning - Supervised Pre-training

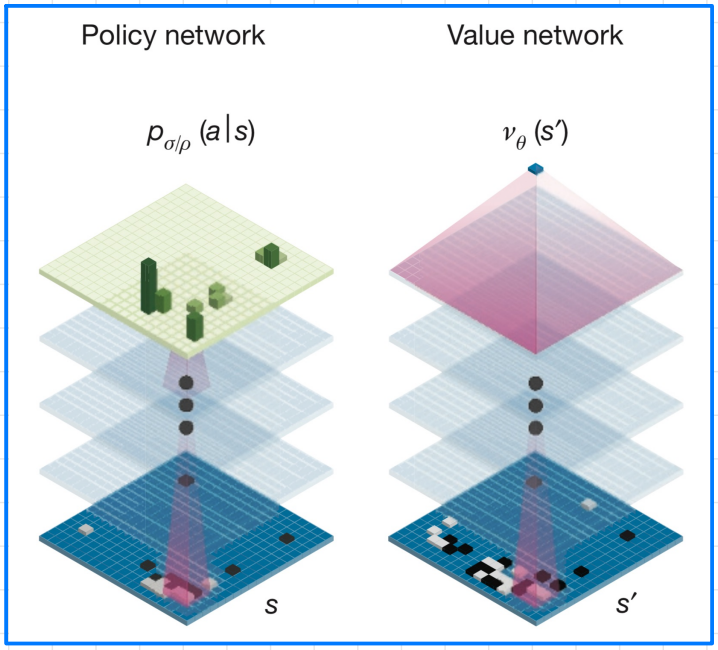
- we have now a parametrized policy $\tilde{\pi}(a|s,\underline{\theta})$
- say we have some data $D = \{(s_1, a_1), (s_2, a_2), \ldots, (s_N, a_N)\}$ collected from (human) experts with policy $\tilde{\pi}_{expert}$
- learning from this data is just vanilla classification*!
- maximize conditional log-likelihood

(*regression for continuous a)

$$J(\underline{\theta}) = \sum_{i=1}^{N} \log \tilde{\pi}(a_i | s_i, \underline{\theta})$$

- behavioral cloning / supervised pre-training is often as an effective initialization

Pre-Training in Alpha Go



Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

NATURE | VOL 529 | 28 JANUARY 2016

- Alpha Go, which defeated the European Go Champion Fan Hui and later Grand Master Lee Sedol, used a "policy network", i.e. a parametrized policy based on a convolutional neural network
- the policy network was pre-trained on 30 M positions from the KGS Go Server and was then refined using policy gradient
- later versions (Alpha Go Zero) didn't use pre-training any more

Performance Metric for Interactive Setting

- for RL (interactive setting) we need a performance metric $J(\underline{\theta})$
- for episodic environments, we can use the value of a dedicated start state s_1

$$J_1(\underline{\theta}) = v_{\pi_{\underline{\theta}}}(s_1) = \mathbb{E}_{\pi_{\underline{\theta}}}[G_1 \mid S_1 = s_1]$$

- for continuing environments, we might use
  - expected value per time step

$$J_{EV}(\underline{\theta}) = \mathbb{E}_{\pi_{\underline{\theta}}}[v_{\pi_{\underline{\theta}}}(S)] = \sum_s \mu_{\pi_{\underline{\theta}}}(s)\, v_{\pi_{\underline{\theta}}}(s)$$

  - or expected reward per time step

$$J_{ER}(\underline{\theta}) = \sum_s \mu_{\pi_{\underline{\theta}}}(s)\, r(s)$$

where $\mu_{\pi_{\underline{\theta}}}$ is the stationary state distribution under $\pi_{\underline{\theta}}$

Methods to Estimate the Policy Gradient

- Finite Differences
- REINFORCE (Williams 1992)
- Actor-Critic Methods

Finite Differences

$$J(\underline{\theta}) = J_1(\underline{\theta}) = v_\pi(s_1)$$

- definition of partial derivative

$$\frac{\partial J}{\partial \theta_i} := \lim_{h \to 0} \frac{J(\dots, \theta_i + h, \dots) - J(\dots, \theta_i, \dots)}{h}$$

- finite difference: for small $h > 0$

$$\frac{\partial J}{\partial \theta_i} \approx \frac{J(\dots, \theta_i + h, \dots) - J(\dots, \theta_i, \dots)}{h}$$

- run M roll-outs for current $\underline{\theta}$ and for each $\underline{\theta} + h\underline{e}_i$ (i^{th} unit vector)
- compute $\frac{\partial J}{\partial \theta_i} \approx \frac{\frac{1}{M}\sum_m g_m - \frac{1}{M}\sum_m g_{m,i}}{h}$ (Monte Carlo)

g_m: m^{th} return with $\underline{\theta}$

$g_{m,i}$: m^{th} return with $\underline{\theta} + h\underline{e}_i$

\+ simple to implement

\+ can be applied to non-differentiable π as well

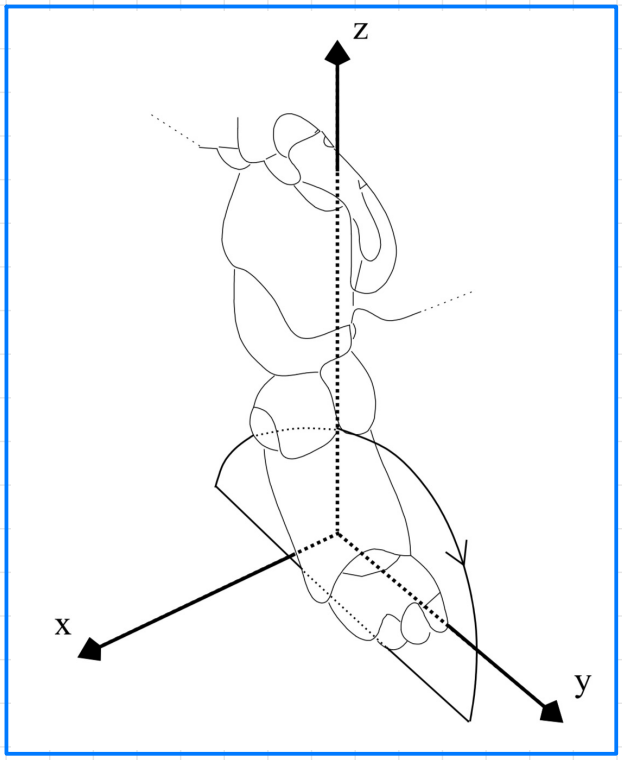
\- expensive, only feasible for low-dimensional $\underline{\theta}$

in total, we need $M(d+1)$ rollouts per gradient estimate

Teaching Aibo Fast Walk

Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion

Nate Kohl and Peter Stone
Department of Computer Sciences, The University of Texas at Austin



- 12 parameters
- J: walking speed
- policy gradient with finite differences
- parallelized over several robots

All told, the following set of 12 parameters define the Aibo's gait [10]:

- The front locus (3 parameters: height, x-pos., y-pos.)
- The rear locus (3 parameters)
- Locus length
- Locus skew multiplier in the x-y plane (for turning)
- The height of the front of the body
- The height of the rear of the body
- The time each foot takes to move through its locus
- The fraction of time each foot spends on the ground

REINFORCE

Williams, 1992

- consider episodic tasks and $J_1(\underline{\theta}) = v_{\pi}(s_1) = \mathbb{E}_{\pi}[G_1 \mid S_1 = s_1]$
- how to compute $\nabla_{\underline{\theta}} J_1(\underline{\theta})$?
- let $S_1, A_1, R_2, S_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$ be an episode
- split episode into $\tau = S_1, A_1, S_2, A_2, \ldots, S_{T-1}, A_{T-1}, S_T$ and
  $\bar{R} = R_2, R_3, \ldots, R_T$
- now, we can write

$$J_1(\underline{\theta}) = \mathbb{E}_{\tau}\left[\mathbb{E}_{\bar{R}}[G_1 \mid \tau]\right] = \mathbb{E}_{\tau}[\bar{g}(\tau)]$$

→ just use observed return $\bar{g}(\tau) \approx \sum_{k=1}^{T-1} \gamma^k R_{t+1}$ ✓

- the gradient is tricky:

$$\nabla_{\underline{\theta}} J_1(\underline{\theta}) = \nabla_{\underline{\theta}} \mathbb{E}_{\tau}[\bar{g}(\tau)] = \nabla_{\underline{\theta}} \sum_{\tau} p(\tau; \underline{\theta})\, \bar{g}(\tau)$$

does not depend on $\underline{\theta}$

1. τ is a (long) sequence
2. $p(\tau; \underline{\theta})$ is determined by both the unknown MDP and π

The Log-Derivative Trick

- Let $p(X|\underline{\theta})$ be a distribution over some random variable X
- note that $\nabla_\theta \log p(X|\underline{\theta}) = \frac{1}{p(X|\underline{\theta})} \nabla_\theta p(X|\underline{\theta})$
- thus, $\nabla_\theta p(X|\underline{\theta}) = p(X|\underline{\theta}) \nabla_\theta \log p(X|\underline{\theta})$
- now, note that

$$\nabla_\theta \mathbb{E}_x[f(x)] = \nabla_\theta \sum_x p(x,\theta) f(x)$$

$$= \sum_x \nabla_\theta p(x,\theta) f(x)$$

$$= \sum_x p(X|\underline{\theta}) \nabla_\theta \log p(X|\underline{\theta}) f(x)$$

$$= \mathbb{E}_x[\nabla_\theta \log p(X|\underline{\theta}) f(x)]$$

REINFORCE cont'd

$$\nabla_{\underline{\theta}} J_1(\underline{\theta}) = \nabla_{\underline{\theta}} \mathbb{E}_\tau[\bar{g}(\tau)] = \nabla_{\underline{\theta}} \sum_\tau p(\tau; \underline{\theta})\, \bar{g}(\tau)$$

- $p(\tau) = \hat{\pi}(a_1|s_1)\, p(s_2|s_1,a_1)\, \hat{\pi}(a_2|s_2) \ldots = \prod_{t=1}^{T-1} \hat{\pi}(a_t|s_t,\underline{\theta})\, p(s_{t+1}|s_t,a_t)$ (policy, MDP)
- $\nabla_{\underline{\theta}} J_1(\underline{\theta}) = \nabla_{\underline{\theta}} \mathbb{E}_\tau[\bar{g}(\tau)]$

$$= \mathbb{E}_\tau[\nabla_{\underline{\theta}} \log p(\tau)\, \bar{g}(\tau)] \qquad \text{log-derivative trick}$$

$$= \mathbb{E}_\tau\left[\nabla_{\underline{\theta}} \left(\sum_{t=1}^{T-1} \log \hat{\pi}(a_t|s_t,\underline{\theta}) + \log p(s_{t+1}|s_t,a_t) \right) \bar{g}(\tau)\right]$$

(does not depend on $\underline{\theta}$, $\nabla_\theta \log p(\ldots) = 0$

$$= \mathbb{E}_\tau\left[\left(\sum_{t=1}^{T-1} \overbrace{\nabla_{\underline{\theta}} \log \hat{\pi}(a_t|s_t,\underline{\theta})}^{\text{automatic differentiation}} \right) \bar{g}(\tau)\right] \longrightarrow \text{estimate with MC}$$

REINFORCE cont'd

initialize $\underline{\theta}$

repeat

generate episode $S_1, A_1, R_2, S_2, A_2, \ldots, S_{T-1}, A_{T-1}, R_T$ with $\pi(a|s, \underline{\theta})$

$$\tilde{\nabla}_\theta J_1(\underline{\theta}) \leftarrow \left(\sum_{k=1}^{T-1} \gamma^k R_{k+1} \right) \left(\sum_{k=1}^{T-1} \nabla_{\underline{\theta}} \log \pi(A_t | S_t, \underline{\theta}) \right)$$

$$\theta \leftarrow \theta + \alpha \tilde{\nabla}_\theta J_1(\underline{\theta})$$

- $\tilde{\nabla}_\theta J_1(\underline{\theta})$ an <u>unbiased</u> estimate of $\nabla_\theta J_1(\underline{\theta})$
- $\mathbb{E}\left[\tilde{\nabla}_\theta J_1(\underline{\theta})\right] = \nabla_\theta J_1(\underline{\theta})$
- thus, under usual <u>SGD conditions</u> $\left(\sum_{i=1}^{\infty} \alpha_i = \infty, \sum_{i=1}^{\infty} \alpha_i^2 < \infty \right)$

  REINFORCE converges to a local maximum of $J_1(\underline{\theta})$

REINFORCE with Baselines

- the REINFORCE estimator

$$\tilde{\nabla}_\theta J_1(\underline{\theta}) \leftarrow \left(\sum_{k=1}^{T-1} \gamma^k R_{k+1}\right)\left(\sum_{k=1}^{T-1} \nabla_{\underline{\theta}} \log \tilde{\pi}(A_t | S_t, \underline{\theta})\right)$$

  is unbiased but usually has high variance

- we can subtract a constant "baseline" b from $\bar{g} = \sum_k \gamma^k R_{k+1}$

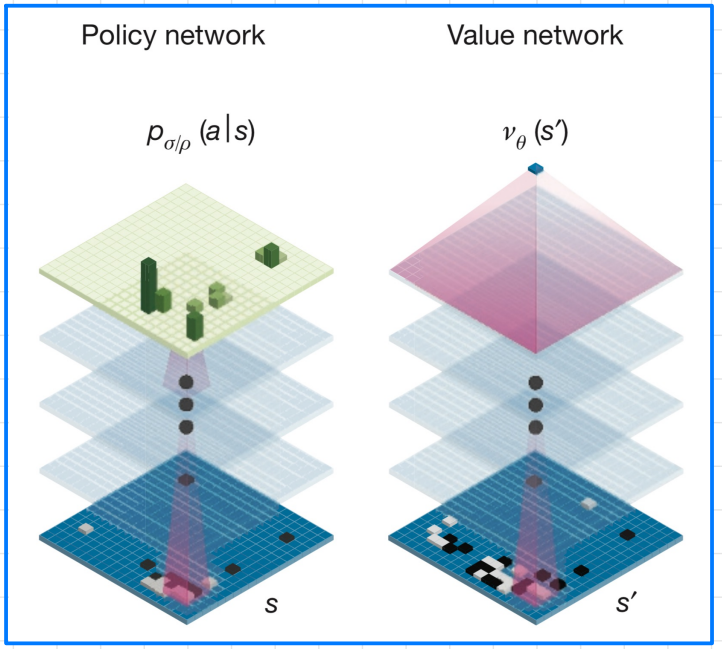
$$\nabla_\theta J_1(\underline{\theta}) = \nabla_{\underline{\theta}} \mathbb{E}_\tau[\bar{g}(\tau)] \underbrace{- \nabla_\theta b}_{=\underline{0}} = \nabla_{\underline{\theta}} \mathbb{E}_\tau[\bar{g}(\tau) - b]$$

- for example, we can estimate $b = \tilde{v} \approx v_\pi(S_1)$, yielding

$$\tilde{\nabla}_\theta J_1(\underline{\theta}) \leftarrow \left(\left(\sum_{k=1}^{T-1} \gamma^k R_{k+1}\right) - \tilde{v}\right)\left(\sum_{k=1}^{T-1} \nabla_{\underline{\theta}} \log \tilde{\pi}(A_t | S_t, \underline{\theta})\right)$$

- still unbiased, but lower variance
- hence, faster learning

REINFORCE in Alpha Go



- the pre-trained policy network (30M positions on KGS server) was then trained with REINFORCE
- self-play: one network is trained, while opponent is a previous version of the policy network
- the resulting policy network won 85% of the games against "Pachi," the strongest open source Go program back then
- (the final Alpha Go system used Monte Carlo Tree search combining the policy network with value function approximation)

Towards Online Policy Gradients

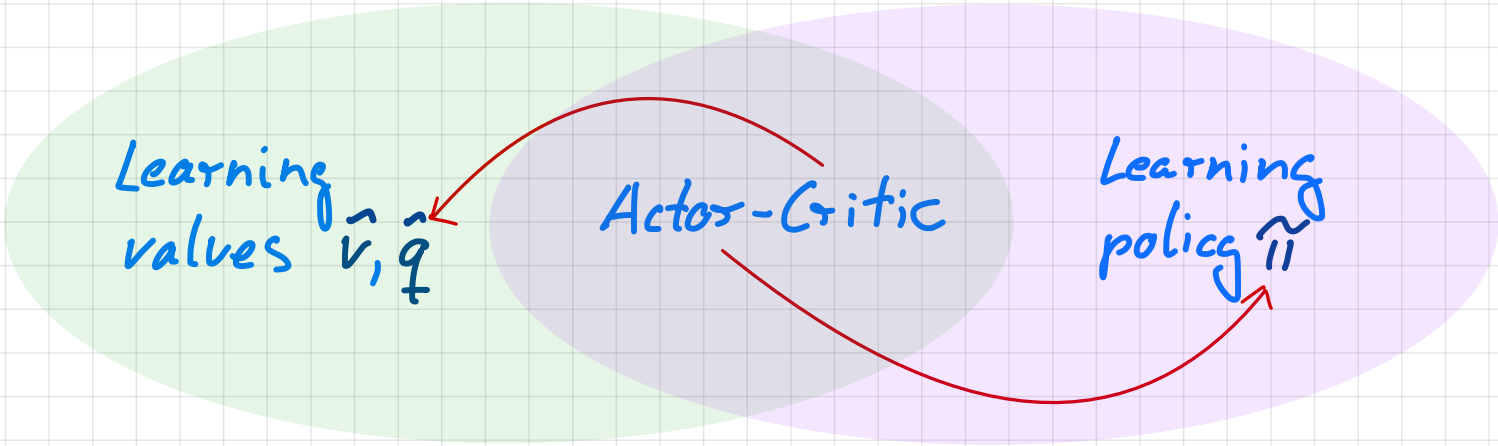
$$\tilde{\nabla}_\theta J_1(\underline{\theta}) \leftarrow G_1 \left(\sum_{k=1}^{T-1} \nabla_{\underline{\theta}} \log \hat{\pi}(A_t | S_t, \underline{\theta}) \right)$$

- REINFORCE is a typical Monte Carlo algorithm
  - need to wait until end of episode
- can we get an online algorithm, similar to TD?
- under certain assumptions (ergodicity), also the following is an unbiased estimator of the policy gradient:

$$\tilde{\nabla}_\theta J_1(\underline{\theta}) \leftarrow \sum_{t=1}^{T-1} G_t \nabla_{\underline{\theta}} \log \hat{\pi}(A_t | S_t, \underline{\theta})$$

- replace G_t with TD-style backups → actor-critic methods

Actor-Critic Methods



- learn both $\hat{q}(\underline{w})$ (last lecture) and $\tilde{\pi}(\underline{\theta})$ simultaneously
- $\tilde{\pi}(\underline{\theta})$ follows approximate policy gradient given by $\hat{q}(\underline{w})$
- inherits instability of value function approximation
- however, actor-critic methods tend to be relatively stable in practice, since each update changes the policy only slightly
- in generalized policy iteration, on the other hand, small changes in $\hat{q}$ can lead to dramatic changes in the greedy policy

TD Actor-Critic

input: stepsizes $\alpha^{\underline{\theta}}$, $\alpha^{\underline{w}}$
initialize policy parameters $\underline{\theta}$ and $\hat{q}$ parameters $\underline{w}$
repeat // for each episode (one for continuing tasks)
  initialize S
  $A \sim \pi(a|S, \underline{\theta})$
  repeat until S is terminal
    take action A, observe S', R
    $A' \sim \pi(a|S', \underline{\theta})$
    $\delta \leftarrow R + \gamma \hat{q}(S', A', \underline{w}) - \hat{q}(S, A, \underline{w})$ // TD delta
    $\underline{w} \leftarrow \underline{w} - \alpha^{\underline{w}} \delta \nabla_{\underline{w}} \hat{q}(S, A, w)$ // update critic
    $\underline{\theta} \leftarrow \underline{\theta} + \alpha^{\underline{\theta}} \hat{q}(S, A, \underline{w}) \nabla_{\underline{\theta}} \log \pi(A|S, \underline{\theta})$ // update policy
    $S, A \leftarrow S', A'$

Advantage Function - Baseline for Actor-Critic

- TD actor-critic is a one-step algorithm and has less variance than REINFORCE ✓
- still, $\hat{q}$ might have significant variance
- we can use a baseline $b(s)$ and subtract it from $\hat{q}$
- a natural baseline is v_π, leading to the advantage function

$$a_\pi(s,a) := q_\pi(s,a) - v_\pi(s)$$

- the TD error $\delta = R + \gamma v_\pi(S') - v_\pi(S)$ estimates $a_\pi(s,a)$:

$$\mathbb{E}[\delta \mid s, A] = \mathbb{E}[R + \gamma v_\pi(S') - v_\pi(S) \mid s, A] = q_\pi(s,A) - v_\pi(S)$$

- thus, instead of $\hat{q}$, learn $\hat{v}$ as critic and use update

$$\theta \leftarrow \theta + \alpha^\theta \delta \nabla_\theta \log \pi(A \mid s, \theta) \qquad \delta \approx R + \gamma \hat{v}(s') - \hat{v}(s)$$

Super Mario

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM
[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

- actor-critic method using advantage function leading to super-human play of the classic game "Super Mario Bros"
- asynchronous: multiple agents in parallel environments
- Compared best against asynchronous Sarsa and Q-learning