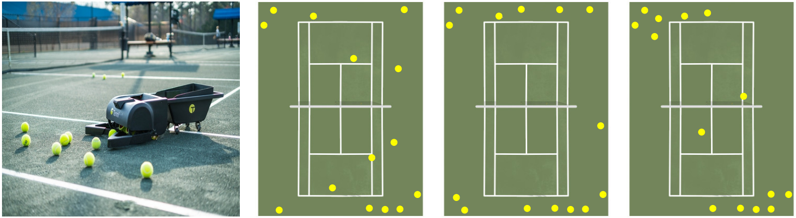
From Policy Gradient to Actor-Critic methods

Olivier Sigaud

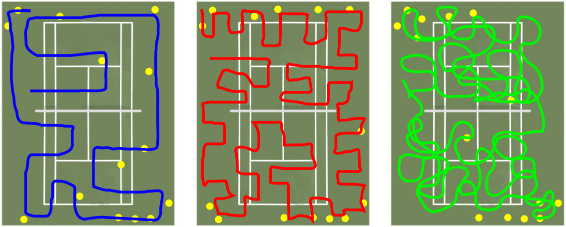
Sorbonne Université
http://people.isir.upmc.fr/sigaud

Example: a (cheap) tennis ball collector



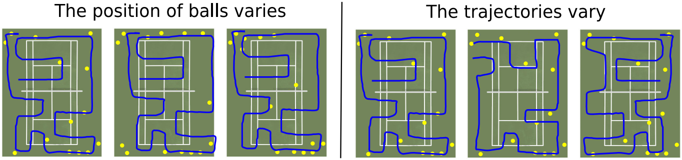
- A robot without a ball sensor
- Travels on a tennis court based on a parametrized controller
- Performance: number of balls collected in a given time
- Just depends on robot trajectories and ball positions

Influence of policy parameters



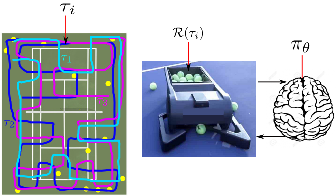
- Controller parameters: proba of turn per time step, travelling speed
- How do the parameters influence the performance?
- Policy search: find the optimal policy parameters

Two sources of stochasticity



- From the environment: position of the balls
- From the policy, if it is stochastic
- The performance can vary a lot → need to repeat
- Tuning parameters can be hard

The policy search problem: formalization



- τ_i is a robot trajectory
- $R(\tau_i)$ is the corresponding return
- $\pi_{\boldsymbol{\theta}}$ is the parametrized policy of the robot

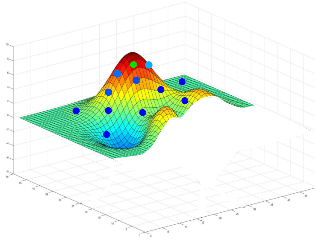
- We want to optimize $J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[R(\tau)]$, the global utility function
- We tune policy parameters $\boldsymbol{\theta}$, thus the goal is to find

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\tau} P(\tau|\boldsymbol{\theta})R(\tau) \tag{1}$$

- where $P(\tau|\boldsymbol{\theta})$ is the probability of trajectory τ under policy $\pi_{\boldsymbol{\theta}}$

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013) A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142

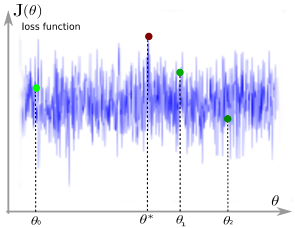
Direct Policy Search is black box optimization



- $J(\boldsymbol{\theta})$ is the performance over policy parameters
- Choose a $\boldsymbol{\theta}$
- Generate trajectories $\tau_{\boldsymbol{\theta}}$
- Get the return $J(\boldsymbol{\theta})$ of these trajectories
- Look for a better $\boldsymbol{\theta}$, repeat

- DPS uses $(\boldsymbol{\theta}, J(\boldsymbol{\theta}))$ pairs and directly looks for $\boldsymbol{\theta}$ with the highest $J(\boldsymbol{\theta})$

(Truly) Random Search



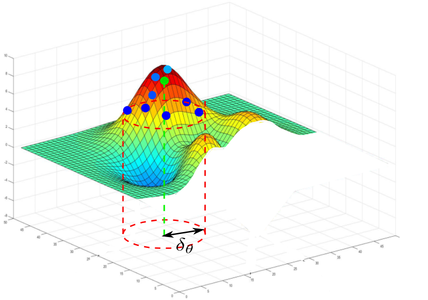
- Select $\boldsymbol{\theta}_i$ randomly
- Evaluate $J(\boldsymbol{\theta}_i)$
- If $J(\boldsymbol{\theta}_i)$ is the best so far, keep $\boldsymbol{\theta}_i$
- Loop until $J(\boldsymbol{\theta}_i) > target$

- Of course, this is not efficient if the space of $\boldsymbol{\theta}$ is large
- General "blind" algorithm, no assumption on $J(\boldsymbol{\theta})$
- We can do better if $J(\boldsymbol{\theta})$ shows some local regularity

Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Direct policy search

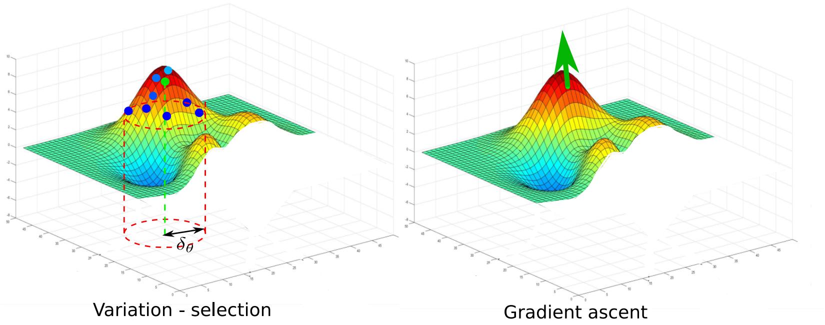
- Locality assumption: The function is locally smooth, good solutions are close to each other



Variation - selection

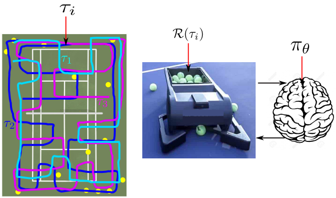
- Variation - selection: Perform well chosen variations, evaluate them
- Variations generally controlled using a multivariate Gaussian

Gradient ascent



- Gradient ascent: Following the gradient from analytical knowledge
- Issue: in general, the function $J(\boldsymbol{\theta})$ is unknown
- How can we apply gradient ascent without knowing the function?
- The answer is the Policy Gradient Theorem
- Next lessons: Policy Gradient methods

Reminder: policy search formalization



- τ_i is a robot trajectory
- $R(\tau_i)$ is the corresponding return
- $\pi_{\boldsymbol{\theta}}$ is the parametrized policy of the robot

- We want to optimize $J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}}[R(\tau)]$, the global utility function
- We tune policy parameters $\boldsymbol{\theta}$, thus the goal is to find

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\tau} P(\tau|\boldsymbol{\theta}) R(\tau) \tag{2}$$

- where $P(\tau|\boldsymbol{\theta})$ is the probability of trajectory τ under policy $\pi_{\boldsymbol{\theta}}$

Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013) A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142

Policy Gradient approach

- General idea: increase $P(\tau|\boldsymbol{\theta})$ for trajectories τ with a high return
- Gradient ascent: Following the gradient from analytical knowledge
- Issue: in general, the function $J(\boldsymbol{\theta})$ is unknown
- How can we apply gradient ascent without knowing the function?
- The answer is the Policy Gradient Theorem

Policy Gradient approach (2)

- Direct policy search works with $< \boldsymbol{\theta}, J(\boldsymbol{\theta}) >$ samples
- It ignores that the return comes from state and action trajectories generated by a controller $\pi_{\boldsymbol{\theta}}$
- We can obtain explicit gradients by taking this information into account
- Not black-box anymore: access the state, action and reward at each step
- The transition and reward functions are still unknown (gray-box approach)
- Requires some math magics
- This lesson builds on "Deep RL bootcamp" youtube video $\#4A$: https://www.youtube.com/watch?v=S_gwYj1Q-44 (Pieter Abbeel)

Plain Policy Gradient (step 1)

- We are looking for $\boldsymbol{\theta}^* = \mathrm{argmax}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathrm{argmax}_{\boldsymbol{\theta}} \sum_\tau P(\tau|\boldsymbol{\theta})R(\tau)$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \sum_\tau P(\tau|\boldsymbol{\theta})R(\tau) \\
&= \sum_\tau \nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta})R(\tau) && \text{* gradient of sum is sum of gradients} \\
&= \sum_\tau \frac{P(\tau|\boldsymbol{\theta})}{P(\tau|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta})R(\tau) && \text{* Multiply by one} \\
&= \sum_\tau P(\tau|\boldsymbol{\theta}) \frac{\nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta})}{P(\tau|\boldsymbol{\theta})} R(\tau) && \text{* Move one term} \\
&= \sum_\tau P(\tau|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})R(\tau) && \text{* by property of gradient of log} \\
&= \mathbb{E}_\tau[\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta})R(\tau)] && \text{* by definition of the expectation}
\end{aligned}
$$

Plain Policy Gradient (step 2)

- We want to compute $\mathbb{E}_\tau[\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta}) R(\tau)]$
- We do not have an analytical expression for $P(\tau|\boldsymbol{\theta})$
- Thus the gradient $\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta}) R(\tau)$ cannot be computed
- Let us reformulate $P(\tau|\boldsymbol{\theta})$ using the policy $\pi_{\boldsymbol{\theta}}$
- What is the probability of a trajectory?
- At each step, probability of taking each action (defined by the policy) times probability of reaching the next state given the action
- Then product over states for the whole horizon H

$$P(\tau|\boldsymbol{\theta}) = \prod_{t=1}^{H} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t).\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \tag{3}$$

- (Strong) Markov assumption here: holds if steps are independent

Plain Policy Gradient (step 2 continued)

- Thus, under Markov assumption,

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log \mathrm{P}(\tau|\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \log[\prod_{t=1}^{H} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t).\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)] \\
&\quad \text{* log of product is sum of logs} \\
&= \nabla_{\boldsymbol{\theta}}[\sum_{t=1}^{H} \log\ \mathrm{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) + \sum_{t=1}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)] \\
&= \nabla_{\boldsymbol{\theta}} \sum_{t=1}^{H} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \text{ * because first term independent of } \boldsymbol{\theta} \\
&= \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \text{ * no dynamics model required!}
\end{aligned}$$

- The key is here: we know $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$!

Plain Policy Gradient (step 2 continued)

- The expectation $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\tau[\nabla_{\boldsymbol{\theta}} \log P(\tau|\boldsymbol{\theta}) R(\tau)]$ can be rewritten

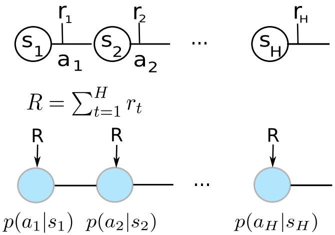
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_\tau[\sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) R(\tau)]$$

- The expectation can be approximated by sampling over m trajectories:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) R(\tau^{(i)}) \tag{4}$$

- The policy structure $\pi_{\boldsymbol{\theta}}$ is known, thus the gradient $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$ can be computed for any pair $(\mathbf{s}, \mathbf{a})$
- We moved from direct policy search on $J(\boldsymbol{\theta})$ to gradient ascent on $\pi_{\boldsymbol{\theta}}$
- Can be turned into a practical (but not so efficient) algorithm

Algorithm 1



- Sample a set of trajectories from $\pi_{\boldsymbol{\theta}}$
- Compute:

$$Loss(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H} \log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})R(\tau^{(i)}) \tag{5}$$

- Minimize the loss using the NN backprop function with your favorite pytorch or tensorflow optimizer (Adam, RMSProp, SGD...)
- Iterate: sample again, for many time steps
- Note: if $R(\tau) = 0$, does nothing

Limits of Algorithm 1

- Needs a large batch of trajectories or suffers from large variance
- The sum of rewards is not much informative
- Computing R from complete trajectories is not the best we can do

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) R(\tau^{(i)})$$

$$\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [\sum_{k=1}^{H} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})]$$

* split into two parts

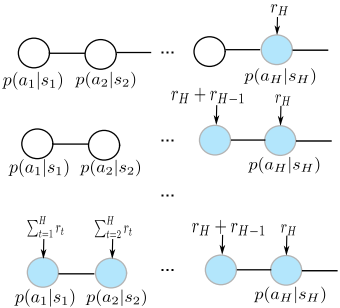
$$\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [\sum_{k=1}^{t-1} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}) + \sum_{k=t}^{H} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})]$$

* past rewards do not depend on the current action

$$\sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [\sum_{k=t}^{H} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})]$$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (28')

Algorithm 2



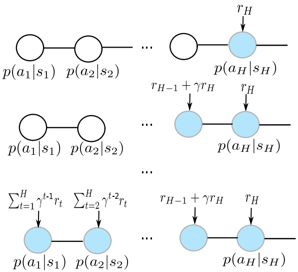
- Same as Algorithm 1
- But the sum is incomplete, and computed backwards
- Slightly less variance, because it ignores irrelevant rewards

Discounting rewards

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \sim \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})[\sum_{k=t}^{H} r(\mathbf{s}_k^{(i)},\mathbf{a}_k^{(i)})]$$

* reduce the variance by discounting the rewards along the trajectory

$$\sim \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})[\sum_{k=t}^{H} \gamma^{k-t} r(\mathbf{s}_k^{(i)},\mathbf{a}_k^{(i)})]$$



https://www.youtube.com/watch?v=S_gwYj1Q-44 (39')

Introducing the action-value function

- $\sum_{k=t}^{H} \gamma^{k-t} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})$ can be rewritten $Q_{(i)}^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$
- $$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \sim \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \text{log} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) Q_{(i)}^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$$
- It is just rewriting, not a new algorithm
- But suggests that the gradient could be just a function of the local step if we could estimate $Q_{(i)}^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t, \mathbf{a}_t)$ in one step

Estimating $Q^{\pi_{\boldsymbol{\theta}}}(s,a)$

- Instead of estimating $Q^{\pi_{\boldsymbol{\theta}}}(s,a) = \mathbb{E}_{(i)}[Q^{\pi_{\boldsymbol{\theta}}}_{(i)}(s,a)]$ from Monte Carlo,
- Build a model $\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}$ of $Q^{\pi_{\boldsymbol{\theta}}}$ through function approximation
- Two approaches:
  - Monte Carlo estimate: Regression against empirical return

$$\boldsymbol{\phi}_{j+1} \rightarrow arg \min_{\boldsymbol{\phi}_j} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} (\sum_{k=t}^{H} \gamma^{k-t} r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}) - \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}))^2$$

  - Temporal Difference estimate: init $\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_0}$ and fit using $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ data

$$\boldsymbol{\phi}_{j+1} \rightarrow \min_{\boldsymbol{\phi}_j} \sum_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')} ||r + \gamma f(\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}', .)) - \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}, \mathbf{a})||^2$$

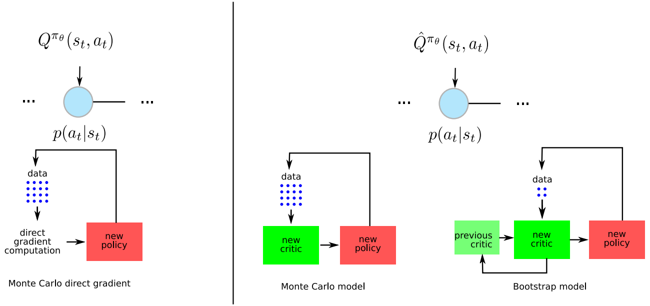
  - $f(\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}', .)) = \max_{\mathbf{a}} \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}', \mathbf{a})$ (Q-learning), $= \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}_j}(\mathbf{s}', \pi_{\boldsymbol{\theta}}(\mathbf{s}'))$ (AC)...
  - May need some regularization to prevent large steps in $\boldsymbol{\phi}$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (36')

Martin Riedmiller. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005

András Antos, Csaba Szepesvári, and Rémi Munos. Fitted Q-iteration in continuous action-space MDPs. In *Advances in neural information processing systems*, pp.9–16, 2008.

Monte Carlo versus Bootstrap approaches



- Three options:
  - MC direct gradient: Compute the true Q^{π_θ} over each trajectory
  - MC model: Compute a model $\hat{Q}^{\pi_\theta}_\phi$ over rollouts using MC regression, throw it away after each policy gradient step
  - Bootstrap: Update a model $\hat{Q}^{\pi_\theta}_\phi$ over samples using TD methods, keep it over policy gradient steps
- With bootstrap, update everything from the current state, see next parts

Policy Gradient with constant baseline

- Reminder:

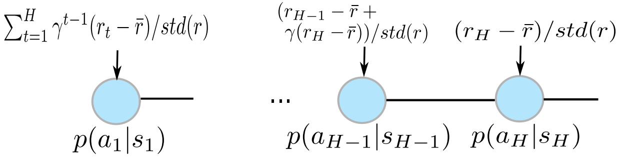
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [\sum_{k=t}^{H} \gamma^k r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})] \tag{6}$$

- If all rewards are positive, the gradient increases all probabilities
- But with renormalization, only the largest increases emerge
- We can substract a "baseline" to (6) without changing its mean:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [\sum_{k=t}^{H} \gamma^k r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)}) - b]$$

- A first baseline is the average return $\bar{r}$ over all states of the batch
- Intuition: returns greater than average get positive, smaller get negative
- Use $(r_t^{(i)} - \bar{r})$ and divide by std $\rightarrow$ get a mean $= 0$ and a std $= 1$
- This improves variance (does the job of renormalization)
- Suggested in https://www.youtube.com/watch?v=tqrcjHuNdmQ

Algorithm 4: adding a constant baseline



- Estimate $\bar{r}$ and $std(r)$ from all rollouts
- Same as Algorithm 2, using $(r_t^{(i)} - \bar{r})/std(r)$
- Suffers from even less variance
- Does not work if all rewards r are identical (e.g. CartPole)

Policy Gradient with state-dependent baseline

- No impact on the gradient as long as the baseline does not depend on action
- A better baseline is $b(\mathbf{s}_t) = V^{\pi}(\mathbf{s}_t) = \mathbb{E}_{\tau}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{H-t} r_H]$
- The expectation can be approximated from the batch of trajectories
- Thus we get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) [Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)} | \mathbf{a}_t^{(i)}) - V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)})]$$

- $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = Q^{\pi}(\mathbf{s}_t | \mathbf{a}_t) - V^{\pi}(\mathbf{s}_t)$ is the advantage function
- And we get

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$$

`https://www.youtube.com/watch?v=S_gwYj1Q-44` (27')

Williams, R. J. (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256

Estimating $V^\pi(s)$

- As for estimating $Q^\pi(s,a)$, but simpler
- Two approaches:
  - Monte Carlo estimate: Regression against empirical return

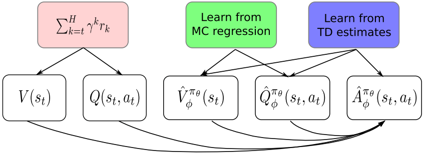
$$\boldsymbol{\phi}_{j+1} \rightarrow arg \min_{\boldsymbol{\phi}_j} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} ((\sum_{k=t}^{H} \gamma^k r(\mathbf{s}_k^{(i)}, \mathbf{a}_k^{(i)})) - \hat{V}^\pi_{\boldsymbol{\phi}_j}(\mathbf{s}_t^{(i)}))^2$$

  - Temporal Difference estimate: init $\hat{V}^\pi_{\boldsymbol{\phi}_0}$ and fit using $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ data

$$\boldsymbol{\phi}_{j+1} \rightarrow \min_{\boldsymbol{\phi}_j} \sum_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')} ||r + \gamma \hat{V}^\pi_{\boldsymbol{\phi}_j}(\mathbf{s}') - \hat{V}^\pi_{\boldsymbol{\phi}_j}(\mathbf{s})||^2$$

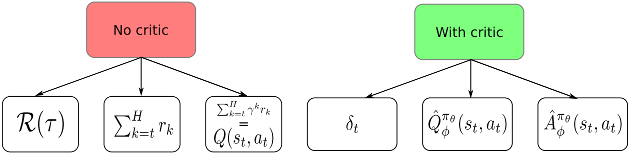
  - May need some regularization to prevent large steps in ϕ

Algorithm 5: adding a state-dependent baseline



- Learn $\hat{V}^\pi_{\boldsymbol{\phi}}$ from TD, from MC rollouts, or compute $V^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)})$ from MC
- Learn $\hat{Q}^\pi_{\boldsymbol{\phi}'}$ from TD, from MC rollouts, or compute $Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)})$ from MC
- Compute $\hat{A}^\pi(\mathbf{s}_t^{(i)}|\mathbf{a}_t^{(i)}) = \hat{Q}^\pi_{\boldsymbol{\phi}}(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) - \hat{V}^\pi_{\boldsymbol{\phi}}(\mathbf{s}_t^{(i)})$
- Or even learn $\hat{A}^\pi_{\boldsymbol{\phi}}$ directly from TD updates using $A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}[\delta_t]$
- Same as Algorithm 3 using $A^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}|\mathbf{a}_t^{(i)})$ instead of $Q^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_t^{(i)}|\mathbf{a}_t^{(i)})$
- Suffers from even less variance

Synthesis



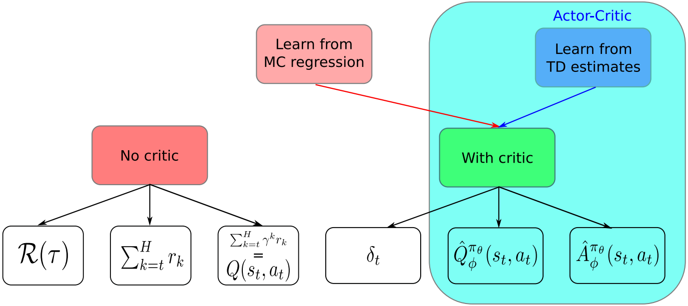
$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(.)}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)})] \psi_t$ where ψ_t can be:

1. $\sum_{t=0}^{H} \gamma^t r_t$: total (discounted) reward of trajectory
2. $\sum_{k=t}^{H} \gamma^{k-t} r_k$: sum of rewards after $\mathbf{a}_t$
3. $\sum_{k=t}^{H} \gamma^{k-t} r_k - b(\mathbf{s}_t)$: sum of rewards after $\mathbf{a}_t$ with baseline
4. $\delta_t = r_t + \gamma V^{\pi}(\mathbf{s}_{t+1}) - V^{\pi}(\mathbf{s}_t)$: TD error, with $V^{\pi}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t}[\sum_{k=0}^{H} \gamma^k r_{t+l}]$
5. $\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{a_{t+1}}[\sum_{k=0}^{H} \gamma^k r_{t+l}]$: action-value function
6. $\hat{A}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) = \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) - \hat{V}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t) = \mathbb{E}[\delta_t]$, advantage function

- Next lesson: Difference to Actor-Critic

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015

Being truly actor-critic



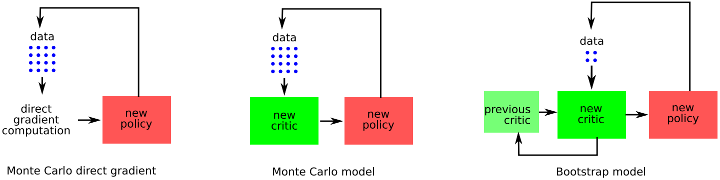
- PG methods with V, Q or A baselines contain a policy and a critic
- Are they actor-critic?
- Only if the critic is learned from bootstrap!

Being Actor-Critic

- "Although the REINFORCE-with-baseline method learns both a policy and a state-value function, we do not consider it to be an actor–critic method because its state-value function is used only as a baseline, not as a critic."
- "That is, it is not used for bootstrapping (updating the value estimate for a state from the estimated values of subsequent states), but only as a baseline for the state whose estimate is being updated."
- "This is a useful distinction, for only through bootstrapping do we introduce bias and an asymptotic dependence on the quality of the function approximation."

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Second edition)*. MIT Press, 2018, p. 331

Monte Carlo versus Bootstrap approaches



- Three options:
  - MC direct gradient: Compute the true $Q^{\pi_{\boldsymbol{\theta}}}$ over each trajectory
  - MC model: Compute a model $\hat{Q}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$ over rollouts using MC regression, throw it away after each policy gradient step
  - Bootstrap: Update a model $\hat{Q}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$ over samples using TD methods, keep it over policy gradient steps
  - Sutton&Barto: Only the latter ensures "asymptotic convergence" (when stable)

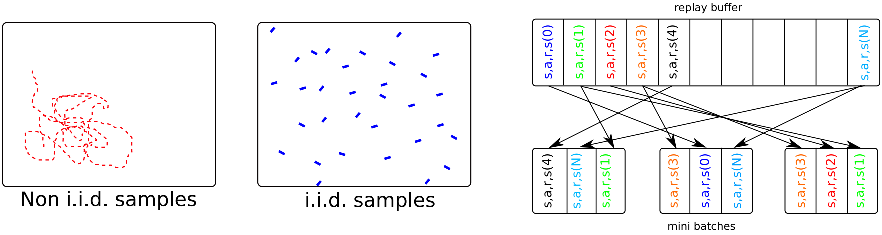
Single step updates

- With a model $\psi_t(s_t^{(i)}, a_t^{(i)})$, we can compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ over a single state using:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})\psi_t(s_t^{(i)}, a_t^{(i)})$$

- With $\psi_t = \hat{Q}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}(s_t^{(i)}, a_t^{(i)})$ or $\psi_t = \hat{A}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}(s_t^{(i)}, a_t^{(i)})$
- This is true whatever the way to obtain $\hat{Q}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$ or $\hat{A}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$
- Crucially, samples used to update $\hat{Q}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$ or $\hat{A}_{\boldsymbol{\phi}}^{\pi_{\boldsymbol{\theta}}}$ do not need to be the same as samples used to compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

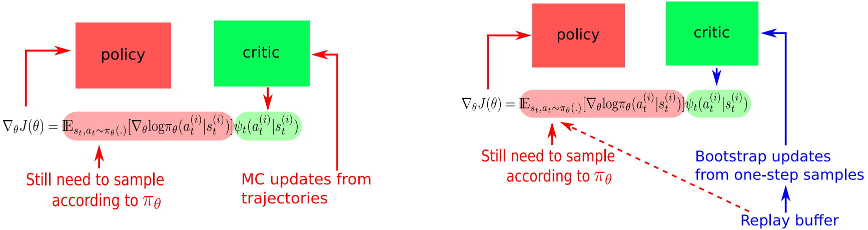
Using a replay buffer



- Agent samples are not independent and identically distributed (i.i.d.)
- Shuffling a replay buffer (RB) makes them more i.i.d.
- It improves a lot the sample efficiency
- Recent data in the RB come from policies close to the current one

Lin, L.-J. (1992) Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3/4), 293–321

Bootstrap properties



- If $\hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}$ is obtained from bootstrap, everything can be done from a single sample
- Samples to compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ still need to come from $\pi_{\boldsymbol{\theta}}$
- Samples to update the critic do not need this anymore
- This defines the shift from policy gradient to actor-critic
- This is the crucial step to become off-policy
- However, using bootstrap comes with a bias
- Next lesson: bias-variance trade-off

Bias versus variance

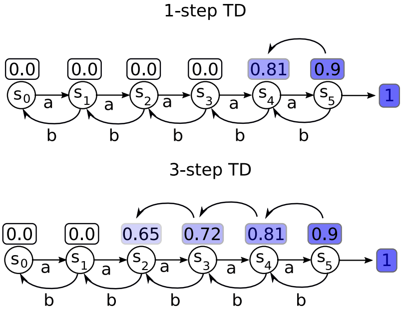
- PG methods estimate an expectation from a finite set of trajectories
- If you estimate an expectation over a finite set of samples, you get a different number each time
- This is known as variance
- Given a large variance, you need many samples to get an accurate estimate of the mean
- That's the issue with MC methods
- If you update an expectation estimate based on a previous (wrong) expectation estimate, the estimate you get even from infinitely many samples is wrong
- This is known as bias
- This is what bootstrap methods do

Geman, S., Bienenstock, E., & Doursat, R. (1992) Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58

Bias variance trade-off

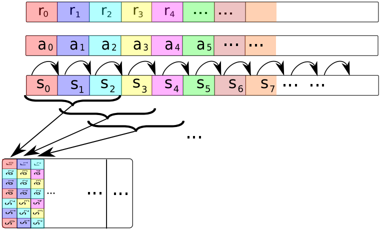
- More complex model (e.g. bigger network): more variance, less bias
- Total error $=$ bias^2 $+$ variance $+$ irreducible error
- There exists an optimum complexity to minimize total error

Using the N-step return



- 1-step TD is poor at backpropagating values along trajectories
- N-step TD is better: N steps of backprop per trajectory instead of one

N-step return and replay buffer



- N-step TD can be implemented efficiently using a replay buffer
- A sample contains several steps
- Various implementations are possible

Lin, L.-J. (1992) Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3/4), 293–321

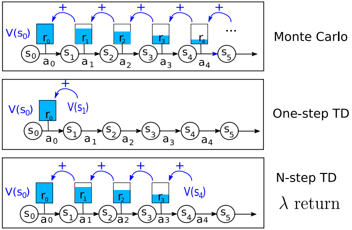
Generalized Advantage Estimation: λ return

- The N-step return can be reformulated using a continuous parameter λ
- $\hat{A}_{\boldsymbol{\phi}}^{(\gamma,\lambda)} = \sum_{l=0}^{H} (\gamma\lambda)^l \delta_{t+l}$
- $\hat{A}_{\boldsymbol{\phi}}^{(\gamma,0)} = \delta_t =$ one-step return
- $\hat{A}_{\boldsymbol{\phi}}^{(\gamma,1)} = \sum_{l=0}^{H} (\gamma)^l \delta_{t+l} =$ MC estimate
- The λ return comes from eligibility trace methods
- Provides a continuous grip on the bias-variance trade-off

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015

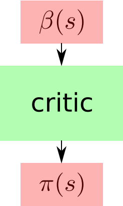
Sharma, S., Ramesh, S., Ravindran, B., et al. (2017) Learning to mix N-step returns: Generalizing λ-returns for deep reinforcement learning. *arXiv preprint arXiv:1705.07445*

Bias-variance compromize



- MC: unbiased estimate of the critic
- But MC suffers from variance due to exploration (+ stochastic trajectories)
- MC on-policy → no replay buffer → less sample efficient
- Bootstrap is sample efficient but suffers from bias and is unstable
- N-step TD or λ return: control the bias-variance compromize
- Acts on critic, indirect effect on performance
- Next lesson: on-policy vs off-policy

Basic concepts



- To understand the distinction, one must consider three objects:
  - The behavior policy $\beta(s)$ used to generate samples.
  - The critic, which is generally $V(s)$ or $Q(s, a)$
  - The target policy $\pi(s)$ used to control the system in exploitation mode.

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepesvári, C. (2000) Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308

Off-policy learning: definition

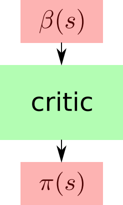
- "Off-policy learning" refers to learning about one way of behaving, called the *target policy*, from data generated by another way of selecting actions, called the *behavior policy*.
- Two notions:
  - Off-policy policy evaluation (not covered)
  - Off-policy control:
    - Whatever the behavior policy (as few assumptions as possible)
    - The target policy should be an approximation to the optimal policy
    - Ex: stochastic behavior policy, deterministic target policy

Maei, H. R., Szepesvári, C., Bhatnagar, S., & Sutton, R. S. (2010) Toward off-policy learning control with function approximation. *ICML*, pages 719–726.

Why prefering off-policy to on-policy control?

- Reusing old data, e.g. from a replay buffer (sample efficiency)
- More freedom for exploration
- Learning from human data (imitation)
- Transfer between policies in a multitask context

Approach: two steps



- Open-loop study
  - Use uniform sampling as "behavior policy" (few assumptions)
  - No exploration issue, no bias towards good samples
  - NB: in uniform sampling, samples do not correspond to an agent trajectory
  - Study critic learning from these samples
- Then close the loop:
  - Use the target policy + some exploration as behavior policy
  - If the target policy gets good, bias more towards good samples

Learning a critic from samples

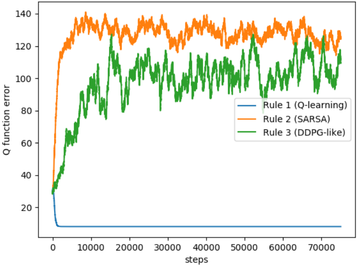
- General format of samples S: $(s_t, a_t, r_t, s_{t+1}, a')$
- Makes it possible to apply a general update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a') - Q(s_t, a_t)]$$

- There are three possible update rules:
  1. $a' = \text{argmax}\, a Q(s_{t+1}, a)$ (corresponds to Q-LEARNING)
  2. $a' = \beta(s_{t+1})$ (corresponds to SARSA)
  3. $a' = \pi(s_{t+1})$ (corresponds e.g. to DDPG, an ACTOR-CRITIC algorithm)

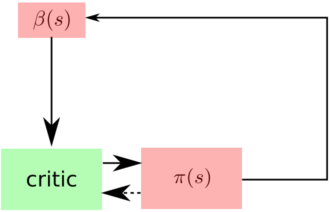
INSTITUT DES SYSTÈMES INTELLIGENTS ET DE ROBOTIQUE

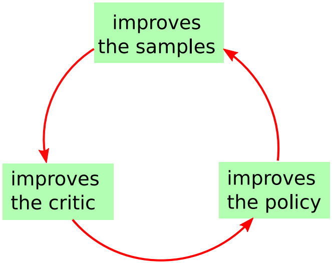
Results



- Rule 1 learns an optimal critic (thus Q-LEARNING is truly off-policy)
- Rule 2 fails (thus SARSA is not off-policy)
- Rule 3 fails too (thus an algorithm like DDPG is not truly off-policy!)
- NB: different ACTOR-CRITIC implementations behave differently
- E.g. if the critic estimates $V(s)$, then equivalent to Rule 1

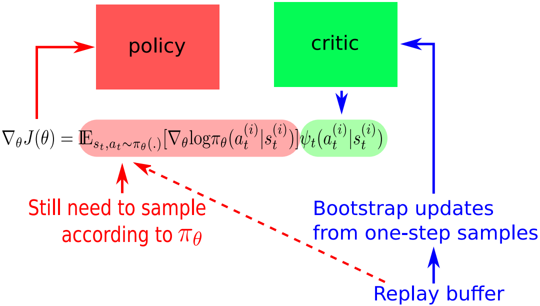
Closing the loop





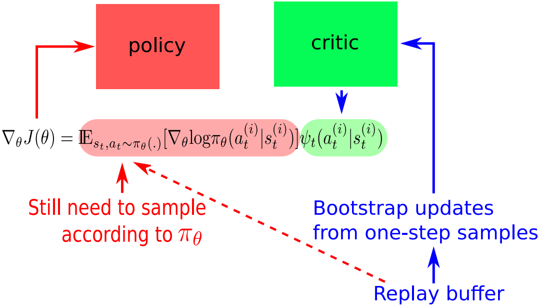
- If $\beta(s) = \pi^*(s)$, then Rules 2 and 3 are equivalent,
- Furthermore, $Q(s, a)$ will converge to $Q^*(s, a)$, and Rule 1 will be equivalent too.
- Quite obviously, Q-LEARNING still works
- SARSA and ACTOR-CRITIC work too: $\beta(s)$ becomes "Greedy in the Limit of Infinite Exploration" (GLIE)

Policy search case



- Q-LEARNING is the only truly off-policy algorithm that I know about
- With continuous action, you cannot compute $\max_a Q^\pi_\phi(\mathbf{s}_{t+1}, \mathbf{a})$
- An algorithm is more or less off-policy depending on assumptions on $\beta(\mathbf{s})$
- With a replay buffer, $\beta(\mathbf{s})$ is generally close enough to $\pi(\mathbf{s})$
- DDPG, TD3, SAC are said off-policy because they use a replay buffer

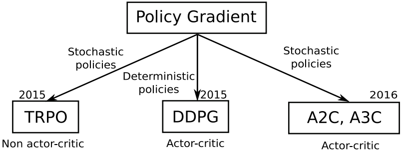
Limits to being off-policy



- DDPG, TD3, SAC use the same off-policy samples to update both the critic and the actor
- OK for the critic, not for the actor
- Does it make sense to sample differently for actor and critic?
- Yes, if several actors share one critic
- Towards offline reinforcement learning

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020

Advantage Actor Critic (A2C)



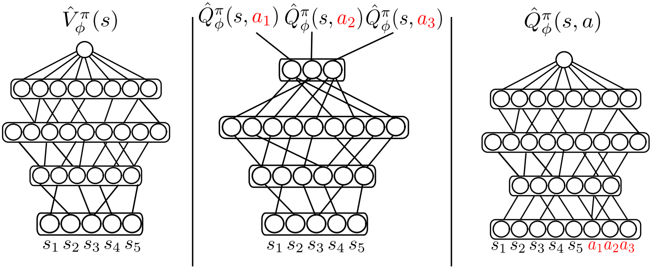
- A crucial move from Policy Gradient methods to Actor-Critic methods
- The earliest actor-critic algorithm of the deep RL era using stochastic policies
- It directly derives from the basic Policy Gradient method
- The critic is learned using bootstrap, which makes it an actor-critic algorithm
- The A2C paper focuses more on A3C, an asynchronous version where several agents generate data without using a replay buffer
- A2C can be seen as a simplified version of A3C with a single agent

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. (2016) Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*

Main distinguishing features

- To perform policy gradient, you need the advantage function
- Computes the advantage function as value function minus the return of the current N-step trajectory
- Adds entropy regularization to favor exploration in the gradient calculation step
- Uses n-step updates
- Does not use a replay buffer
- Note that A2C is Actor-Critic, but on-policy, so one cannot equate Actor-Critic and off-policy
- But adding a replay buffer and making it more off-policy would be straightforward

Choice of a V critic



- Main point: By contrast with $Q(s, a)$, $V(s)$ can be estimated in the same way irrespective of using discrete or continuous actions
- $\hat{V}^{\pi}_{\boldsymbol{\phi}}$ is smaller, but not necessarily easier to estimate (implicit max over actions)
- Temporal difference error: $\delta = [r(\mathbf{s}_t) + \gamma V^{i}_{\boldsymbol{\phi}}(\mathbf{s}_{t+1}) - V^{i}_{\boldsymbol{\phi}}(\mathbf{s}_t)]$
- Standard update rule: $V^{i+1}_{\boldsymbol{\phi}}(\mathbf{s}_t) \leftarrow V^{i}_{\boldsymbol{\phi}}(\mathbf{s}_t) + \alpha\delta$

Advantage function calculation

- To perform policy gradient updates, one needs to compute $\hat{A}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t)$
- By definition, $A(\mathbf{s}_t, \mathbf{a}_t) = Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t)$
- A2C computes the advantage with $\hat{A}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) = R_t(\mathbf{s}_t) - V_{\boldsymbol{\phi}}(\mathbf{s}_t)$
- $R_t(\mathbf{s}_t) = \sum_{i=0}^{N-1} \gamma^i r_{t+i} + \gamma^N V_{\boldsymbol{\phi}}(\mathbf{s}_{t+N})$ is the return of the current N-step trajectory from state $\mathbf{s}_t$
- $R_t(\mathbf{s}_t)$ can be seen as an approximate of $Q(\mathbf{s}_t, \mathbf{a}_t)$ computed along one trajectory

Policy Gradient updates

- The standard Policy Gradient update is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}_t,\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(.)}[\nabla_{\boldsymbol{\theta}} \text{log} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})] \hat{A}_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t)$$

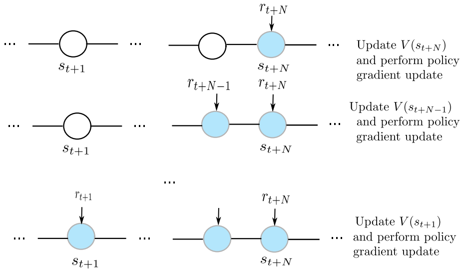
- But to favor exploration, A2C adds an entropy term to the gradient calculation
- Thus the policy update rule is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}_t,\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(.)}[\nabla_{\boldsymbol{\theta}}[\text{log} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)})(R_t - V_{\boldsymbol{\phi}}(\mathbf{s}_t)) - \beta \mathcal{H}(\pi_{\boldsymbol{\theta}}(\mathbf{s}_t))]]$$

- where $\mathcal{H}(\pi_{\boldsymbol{\theta}}(\mathbf{s}_t))$ is the entropy of policy $\pi_{\boldsymbol{\theta}}$ at state $\mathbf{s}_t$.
- Note that A2C adds entropy in the update of the actor, but outside the critic, whereas SAC adds it in the critic target, which has a deeper impact.

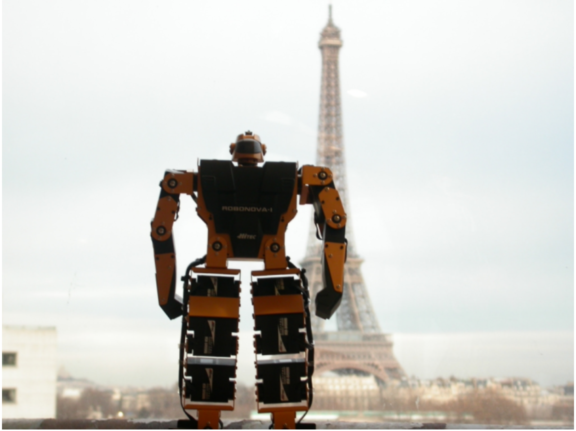
Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A. Abbeel, P. et al. (2018) Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*

N-step updates



- The agent performs N steps in the environment (or less if the episode stops earlier in the episodic case) before each update
- At each update, the agent has collected up to N states and rewards
- It can update the value of the last state using the last reward, the value of the second last step with two rewards
- And so on up to the first state of the current collection
- It updates both the critic and the policy at each update

Any question?



Send mail to: `Olivier.Sigaud@upmc.fr`