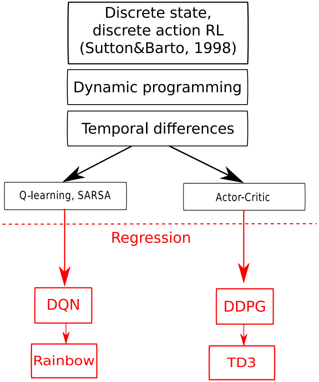
(Deep) Policy Search and Policy Gradient

Olivier Sigaud

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

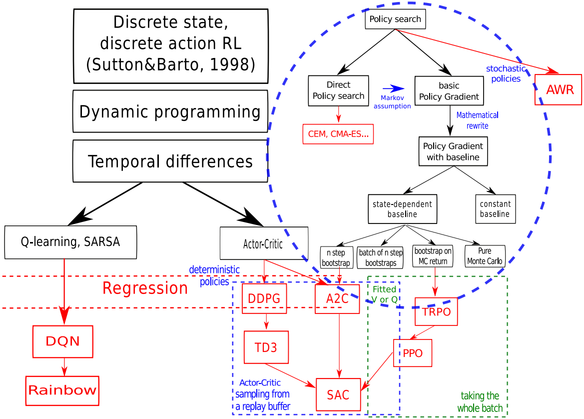
Standard RL Class overview



- From Sutton&Barto to deep RL...

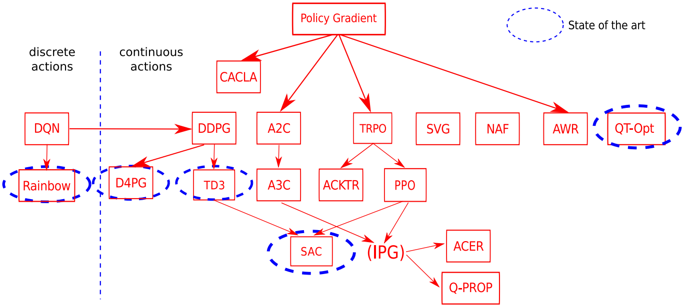
Sutton, R. S. & Barto, A. G. (1998) *Reinforcement Learning: An Introduction*. MIT Press.

Deep Policy Search overview



- Builds on "Deep RL bootcamp" youtube videos https://www.youtube.com/watch?v=S_gwYj1Q-44
- Differences between "pure" policy gradient and actor critic

Next video



- Overview of the most important state-of-the-art deep policy search algorithms
- Main concepts and properties
- Plus videos for individual algorithms

General Goal of Policy Search

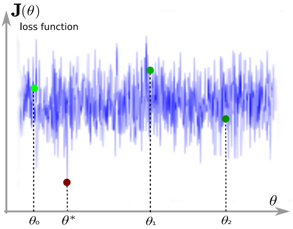
- Let:
  - π_θ be the parametrized policy of an agent
  - τ_θ is an agent trajectory
  - $R(\tau_\theta)$ is the corresponding return
  - $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$ is the global utility (or cost) function
- We have to sample the expectation, thus the goal is to find

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

- where $P(\tau, \theta)$ is the probability of τ under policy π_θ
- We are in a black-box context: we choose a θ, we generate trajectories and get the return $J(\theta)$ of these trajectories
- Then we look for a better θ

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

(Truly) Random Search



- Select θ_i randomly
- Perform a set of τ and get $\hat{J}(\theta_i)$
- If $\hat{J}(\theta_i)$ is the best so far, keep θ_i
- Loop until $\hat{J}(\theta_i) > target$

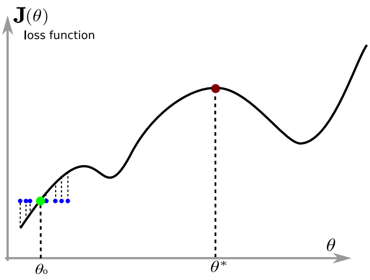
- Of course, this is not efficient if the space of θ is large
- General "blind" algorithm, no assumption on $J(\theta)$
- We can do better if $J(\theta)$ shows some local regularity

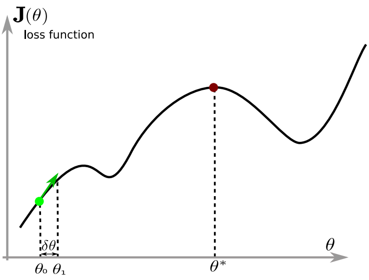
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search

Gradient descent

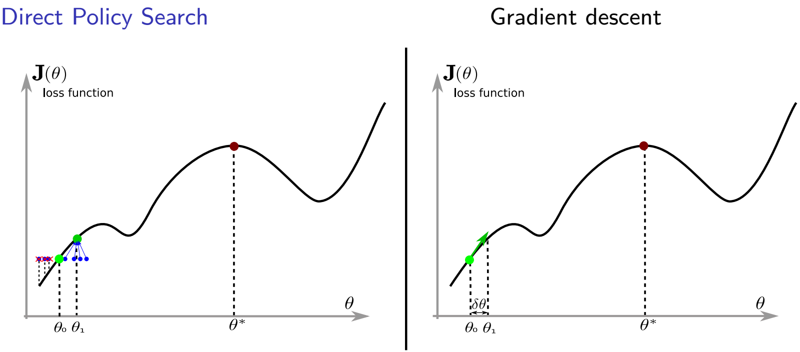




- Start with a policy π_θ with performance $J(\theta)$
- Generate random variations of π_{θ_i} and evaluate their performance $J(\theta_i)$

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

Two families of methods

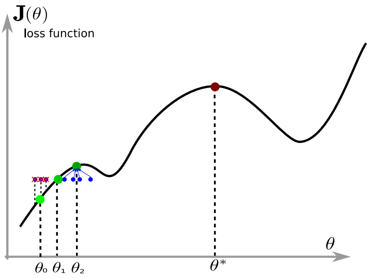


- Select the best variations, ignore the rest
- Get a new policy π_θ from selected variations

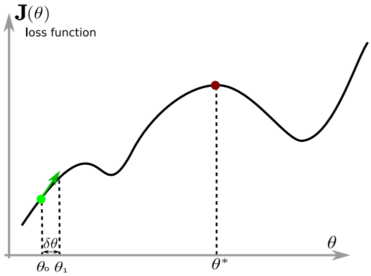
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

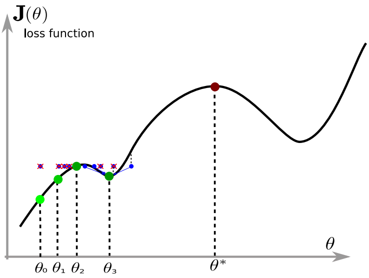


- Repeat the same process
- Approximates the gradient

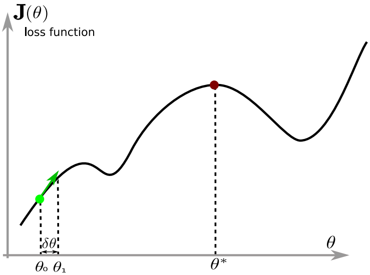
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

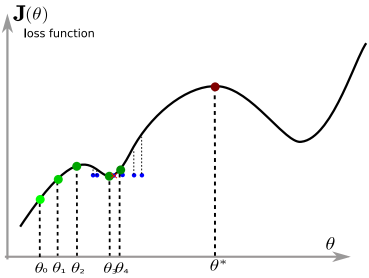


- If variations are wide enough, may escape from easy local minima
- Covariance matrices adapt the width of variations

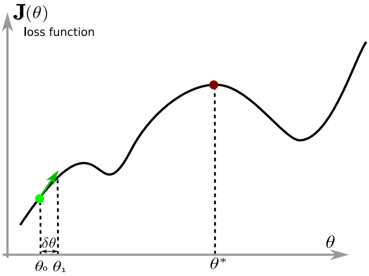
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

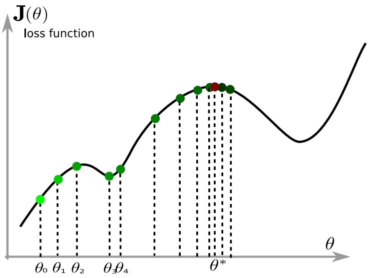


- If variations are wide enough, may escape from easy local minima
- Covariance matrices adapt the width of variations

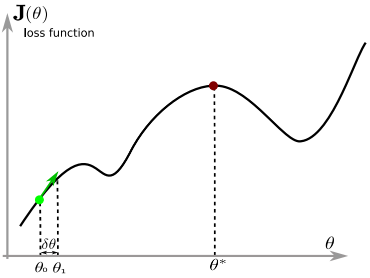
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

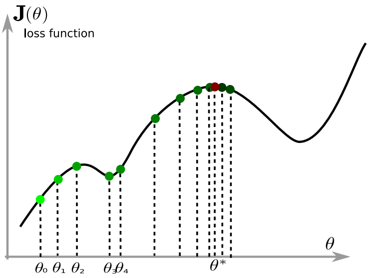


- Until stuck into a wide local minimum
- Genetic Algorithms, Evolution Strategies, Finite Differences...

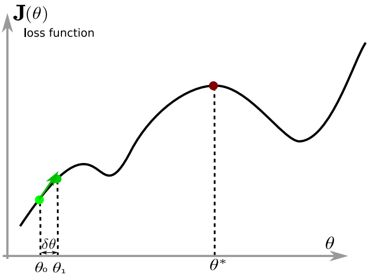
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

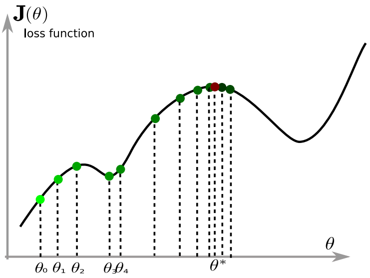


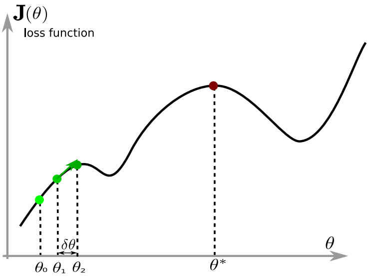
- Compute the local derivative
- Provides steepest descent

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

Two families of methods

Direct Policy Search | Gradient descent



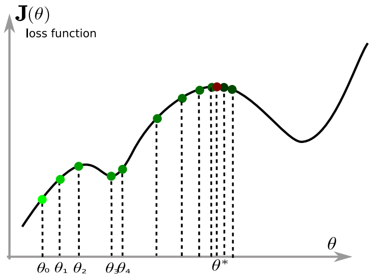


- Follow the gradient with a step
- Necessity to tune step size

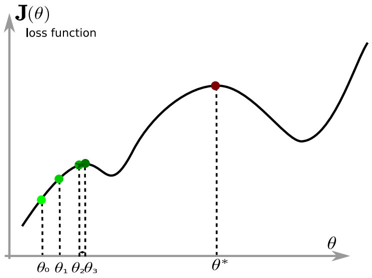
Sigaud, O. & Stulp, F. (2019) Policy search in continuous action domains: an overview. *Neural Networks*, 113:28-40

Two families of methods

Direct Policy Search



Gradient descent

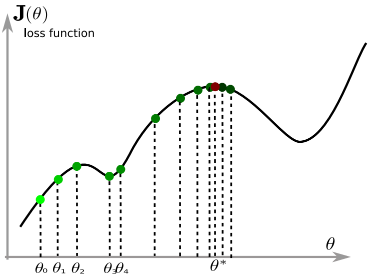


- Iterate until no more improvement
- Stochastic variant escapes too local minima

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

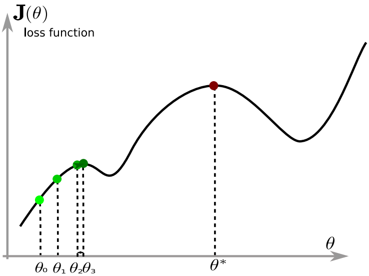
Two families of methods

Direct Policy Search



- Needs many samples
- More easily escapes local minima
- A separate class about this topic

Gradient descent



- No sample needed
- Gets stuck into local minima
- $J(\theta)$ unknown in policy search
- Solution: policy gradient methods

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

Policy gradient methods

- Direct policy search uses $< \theta, J(\theta) >$ pairs and directly looks for θ with the highest $J(\theta)$
- It ignores the fact that the return comes from state and action trajectories generated by a controller π_θ
- We can use explicit gradients if we take this information into account
  - Represent a family of stochastic policies
  - Increase the probabilities of actions producing trajectories with a high return
  - Not black-box anymore: access the state, action and immediate reward at each step
  - The transition and reward functions are still unknown (gray-box approach)
- Watch Pieter Abbeel's deep RL bootcamp video $\#4A$:
  https://www.youtube.com/watch?v=S_gwYj1Q-44

Plain Policy Gradient (step 1)

- Reminder: we look for $\theta^* = \text{argmax}_\theta J(\theta) = \text{argmax}_\theta \sum_\tau P(\tau,\theta)R(\tau)$

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

- The expectation can be approximated over m trajectories

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \text{log} P(\tau^{(i)},\theta)R(\tau^{(i)})$$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (12')

Plain Policy Gradient (step 2)

- We do not have an analytical expression for $P(\tau, \theta)$
- Thus the gradient $\nabla_\theta \log P(\tau^{(i)}, \theta) R(\tau^{(i)})$ cannot be computed
- Let us reformulate $P(\tau, \theta)$ using the policy π_θ

$$P(\tau^{(i)}, \theta) = \prod_{t=1}^{H} p(s_{t+1}^{(i)} | s_t^{(i)}, a_t^{(i)}).\pi_\theta(a_t^{(i)} | s_t^{(i)}) \tag{2}$$

- (Strong) Markov assumption here
- 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

https://www.youtube.com/watch?v=S_gwYj1Q-44 (18')

Plain Policy Gradient (step 2 continued)

- Thus

$$
\begin{aligned}
\nabla_\theta \log \mathrm{P}(\tau^{(i)},\theta) &= \nabla_\theta \log[\prod_{t=1}^{H} p(s_{t+1}^{(i)}|s_t^{(i)},a_t^{(i)}).\pi_\theta(a_t^{(i)}|s_t^{(i)})] \\
&\quad \text{* log of product is sum of logs} \qquad (3) \\
&= \nabla_\theta[\sum_{t=1}^{H} \log\ \mathrm{p}(s_{t+1}^{(i)}|s_t^{(i)},a_t^{(i)}) + \sum_{t=1}^{H} \log\pi_\theta(a_t^{(i)}|s_t^{(i)})] \\
&= \nabla_\theta \sum_{t=1}^{H} \log\pi_\theta(a_t^{(i)}|s_t^{(i)}) \text{ * because first term independent of } \theta \\
&= \sum_{t=1}^{H} \nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)}) \text{ * no dynamics model required!} \qquad (4)
\end{aligned}
$$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (18')

Plain Policy Gradient (step 2 continued)

- Reminder

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \text{log} P(\tau^{(i)}, \theta) R(\tau^{(i)})$$

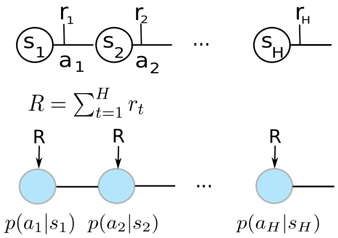
- Thus

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \text{log} \pi_\theta (a_t^{(i)} | s_t^{(i)}) R(\tau^{(i)}) \tag{5}$$

- The policy structure π_θ is known, thus the gradient $\nabla_\theta \text{log} \pi_\theta$ can be computed
- Can be turned into a practical (but inefficient) algorithm
- We moved from direct policy search on $J(\theta)$ to gradient descent on π_θ

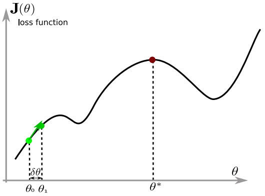
https://www.youtube.com/watch?v=S_gwYj1Q-44 (18')

Pratical algorithm 1: overview



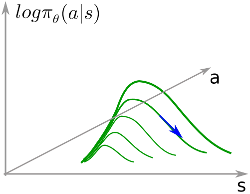
- Collect a set of m trajectories $(s_t^{(i)}, a_t^{(i)}, r_t^{(i)}), i \in \{1, H\}$
- Compute the resulting return $R(\tau^{(i)}) = \sum_{t=1}^{H} r_t^{(i)}$.
- For each visited $(s_t^{(i)}, a_t^{(i)})$ pair, apply $\nabla_\theta \text{log}\pi_\theta(a_t^{(i)}|s_t^{(i)}).R(\tau^{(i)})$
- Given (5), this ensures $J(\theta)$ will improve
- Loop until $J(\theta)$ reaches a local optimum or after some budget

Pratical algorithm 1: intuition



$\theta \leftarrow \theta + \nabla_\theta J(\theta)$

$\Longleftrightarrow$

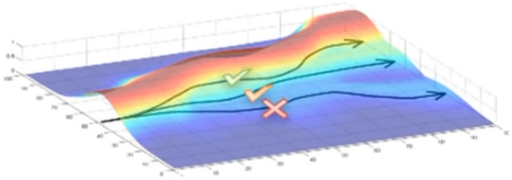


$\theta \leftarrow \theta + \sum(\mathsf{R}\nabla_\theta log\pi_\theta(a|s))$

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \text{log}\pi_\theta(a_t^{(i)}|s_t^{(i)})R(\tau^{(i)})$$

- PG is quite different from DPS: search in the state-action space versus parameter space, using some structural assumptions
- Increasing the log proba. of rewarded actions taken in states increases $J(\theta)$
- $R(\tau)$ is the step size of each gradient update
- A bigger $R(\tau)$ results in a bigger update

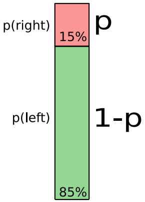
Pratical algorithm 1: further intuition



- Probabilities π_θ must sum to 1, thus increasing one decreases the others
- Moves the action probabilities π_θ in each state towards those providing the highest $R(\tau)$

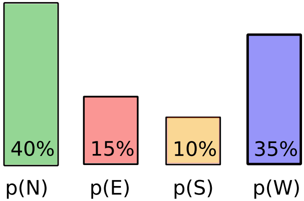
https://www.youtube.com/watch?v=S_gwYj1Q-44

Distributions over actions: Bernoulli



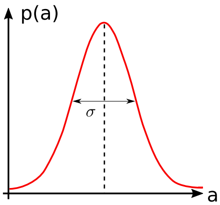
- Binary choice between two actions
- p is a probability, must keep between 0 and 1
- Use sigmoid, or tanh...
- Increasing $p(left)$ decreases $p(right)$

Distributions over actions: Categorical



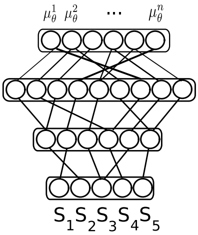
- Choice between K discrete actions
- All the probabilities must sum to 1
- When increasing one probability, how should we decrease the others? (renormalize?)

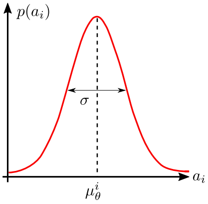
Distributions over actions: Normal



- Choice of a continuous action (extension to mutidimensional with multivariate Gaussian)
- The integral must keep to 1
- Standard approach: keep variance σ constant

Policy representation (continuous action)





The stochastic policy is represented as a multivariate Gaussian: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}) = e^{-\frac{1}{2}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)}$

$$\text{log}\pi_\theta(a_t|s_t) = -\frac{1}{2}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)$$

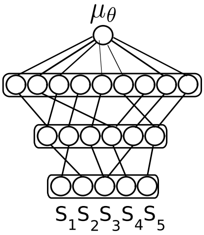
$$\nabla_\theta \text{log}\pi_\theta(a_t|s_t) = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)$$

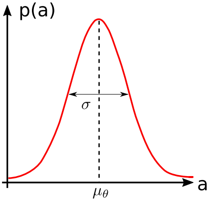
Just backpropagate $\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_\theta - \mathbf{a}_t)\sum_t (r(\mathbf{s}_t, \mathbf{a}_t))$

- NB: We considered a fixed $\boldsymbol{\Sigma}$.
- Learning $\boldsymbol{\Sigma}_\theta$ results in a more involved derivation (but provided by librairies)

https://www.youtube.com/watch?v=SQtOI9jsrJ0

Policy representation (1D continuous action case)





The stochastic policy is represented as a Gaussian: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = e^{-\frac{1}{2}\frac{(\mu_\theta - a_t)^2}{\sigma}}$

$$log\pi_\theta(a_t|s_t) = -\frac{1}{2}\frac{(\mu_\theta - a_t)^2}{\sigma}$$

$$\nabla_\theta \text{log}\pi_\theta(a_t|s_t) = -\frac{\mu_\theta - a_t}{\sigma}$$

Just backpropagate $-\frac{\mu_\theta - a_t}{\sigma}\sum_t r(s_t, a_t)$

https://www.youtube.com/watch?v=SQtOI9jsrJ0

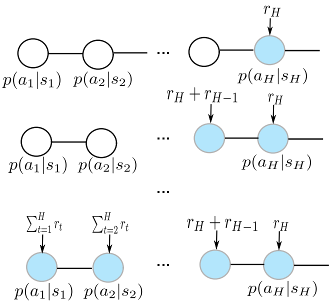
Back to Plain Policy Gradient (step 3)

- Algo. 1 takes a large batch of trajectories: suffers from large variance
- Computing from complete trajectories is not the best we can do

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)})R(\tau^{(i)}) \\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)})[\sum_{t=1}^{H} r(s_t^{(i)}, a_t^{(i)})] \\
&\quad \text{* split into two parts} \qquad (6) \\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)})[\sum_{k=0}^{t-1} r(s_k^{(i)}, a_k^{(i)}) + \sum_{k=t}^{H} r(s_k^{(i)}, a_k^{(i)})] \\
&\quad \text{* past rewards do not depend on the future} \qquad (7) \\
&= \frac{1}{m}\sum_{i=1}^{m}\sum_{t=1}^{H}\nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)})[\sum_{k=t}^{H} r(s_k^{(i)}, a_k^{(i)})]
\end{aligned}
$$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (26')

Algorithm 2: from step 2 to step 3



- Same as Algorithm 1
- But computes the sum backwards
- Slightly better algorithm

Plain Policy Gradient (step 3 continued)

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta (a_t^{(i)} | s_t^{(i)}) [\sum_{k=t}^{H} r(s_k^{(i)}, a_k^{(i)})] \tag{8}$$

- We can reduce the variance by discounting the rewards along the trajectory

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta (a_t^{(i)} | s_t^{(i)}) [\sum_{k=t}^{H} \gamma^k r(s_k^{(i)}, a_k^{(i)})]$$

-

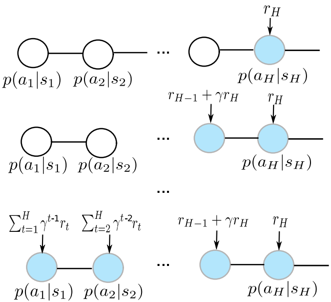
$$\sum_{k=t}^{H} \gamma^k r(s_k^{(i)}, a_k^{(i)}) \text{ can be rewritten } Q^\pi(s_t^{(i)}, a_t^{(i)})$$

- Thus we get

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta (a_t^{(i)} | s_t^{(i)}) Q^\pi(s_t^{(i)}, a_t^{(i)})$$

https://www.youtube.com/watch?v=S_gwYj1Q-44 (26')

Algorithm 3: discounting the reward



- Q^π is estimated from Monte Carlo
- Even smaller variance

Policy Gradient with constant baseline

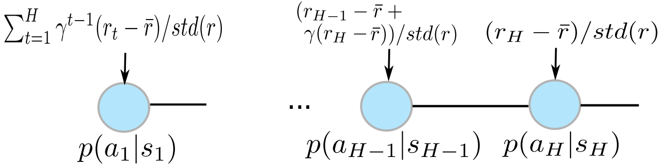
- Besides, we can substract a "baseline" to (8) without changing its mean, but improving its variance

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) [\sum_{k=t}^{H} r(s_k^{(i)}, a_k^{(i)}) - b(s_t^{(i)})]$$

- A first baseline is the average return $\bar{r}$ over all states of the batch
- We then normalize each local return with $r_t^{(i)} - \bar{r}$ and divide by the standard deviation so as to get a mean of 0 and a standard deviation of 1.
- Greater than average returns get positive, smaller get negative
- 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

Policy Gradient with state-dependent baseline

- A better baseline is $b(s_t) = \mathbb{E}_\tau[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{H-t} r_H] = V^\pi(s_t)$
- The expectation can be approximated from the batch of trajectories
- Thus we get

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})[Q^{\pi_\theta}(s_t^{(i)}|a_t^{(i)}) - V^\pi(s_t^{(i)})]$$

- $A^\pi(s_t, a_t) = Q^\pi(s_t|a_t) - V^\pi(s_t)$ is the advantage function
- And we get

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) A^\pi(s_t^{(i)}, a_t^{(i)})$$

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

Algorithm 5: adding a state-dependent baseline

- Estimate $V^\pi(s_t)$ from all rollouts
- Estimate $A^\pi(s_t^{(i)}|a_t^{(i)})$ from all rollouts
- Same as Algorithm 1 with $A^\pi(s_t^{(i)}|a_t^{(i)})$ instead of $R(\tau^{(i)})$
- Suffers from even less variance
- Still no bootstrap update of an estimate $\hat{V}_\phi$ or $\hat{Q}_\phi$

State-dependent baseline: towards bootstrap

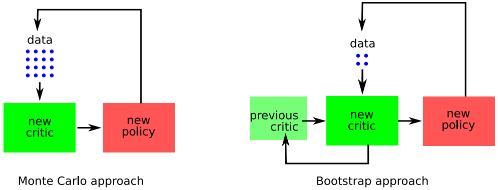
Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration$=1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

- A $\hat{V}_\phi$ or $\hat{Q}_\phi$ baseline provides a value even in unseen states
- Recompute the baseline from all trajectories
- Or update the baseline from one trajectory
- If the critic is estimated based on the previous critic, it becomes bootstrap

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

Monte Carlo versus Bootstrap approaches



- Trajectory-based approach: Monte Carlo methods
- Does not record a critic from an iteration to the next
- Gets an unbiased estimate for all visited state-action pairs using the current batch
- Bootstrap approaches: record a parametrized critic
- Bootstrap is sample efficient but suffers from bias and is unstable
- Monte Carlo is stable, but suffers from variance and is slower

Estimating $V^\pi(s)$ or $Q^\pi(s,a)$

- Let us define $\hat{V}_\phi$ or $\hat{Q}_\phi$ as estimators of $V^\pi(s)$ or $Q^\pi(s,a)$
- Two approaches to estimate them:
  - Monte Carlo estimate: Regression against empirical return

$$\phi_{j+1} \rightarrow arg \min_{\phi_j} \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H} (\hat{V}^\pi_{\phi_j}(s_t^{(i)}) - (\sum_{k=t}^{H} r(s_t^{(i)}, a_t^{(i)})))^2$$

  - Temporal difference estimate: init $\hat{V}^\pi_{\phi_0}$ and fit using (s, a, r, s') data

$$\phi_{j+1} \rightarrow \min_{\phi_j} \sum_{(s,a,r,s')} ||r + \gamma \hat{V}^\pi_{\phi_j}(s') - \hat{V}^\pi_{\phi_j}(s)||^2$$

    May need some regularization to prevent large steps in ϕ
  - Similar equations for $\hat{Q}_\phi$

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

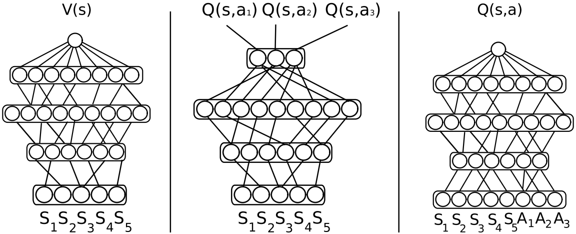
András Antos, Csaba Szepesvári, and Rémi Munos. Fitted Q-iteration in continuous action-space MDPs. In *Advances in neural information processing systems*, pp.9–16, 2008.

Being truly 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

Practical implementation of critics



- $\hat{V}_\phi$ is smaller, but not necessarily easier to estimate

Synthesis

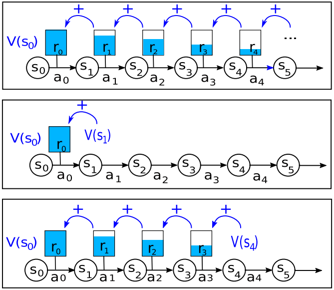
$$\nabla_\theta J(\theta) = \mathbb{E}[\psi_t \nabla_\theta \log\pi_\theta(a_t^{(i)}|s_t^{(i)})]$$

where ψ_t can be:

1. $\psi_t = \sum_{t=0}^{H} \gamma^t r_t$: total reward of trajectory
2. $\psi_t = \sum_{t'=t}^{H} \gamma^{t'-t} r_{t'}$: sum of rewards after a_t
3. $\psi_t = \sum_{t'=t}^{H} \gamma^{t'-t} r_{t'} - b(s_t)$: sum of rewards after a_t with baseline
4. $\psi_t = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) = \delta_t$: TD error, with $V^\pi(s_t) = \mathbb{E}_{a_t}[\sum_{l=0}^{H} \gamma^l r_{t+l}]$
5. $\psi_t = Q^\pi(s_t, a_t)$: state-action value function, with $Q^\pi(s_t, a_t) = \mathbb{E}_{a_{t+1}}[\sum_{l=0}^{H} \gamma^l r_{t+l}]$
6. $\psi_t = A^\pi(s_t, a_t)$: advantage function, with $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) = \mathbb{E}[\delta_t]$

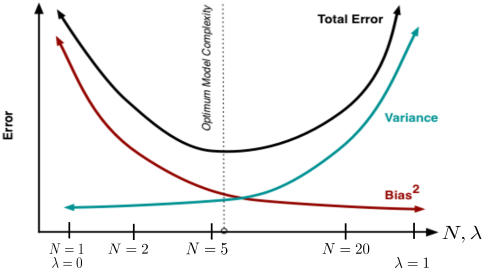
John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015

Monte Carlo, One-step TD and N-step return



- One-step TD suffers from bias
- MC suffers from variance due to exploration (+ stochastic trajectories)
- MC is on-policy → less sample efficient
- N-step TD: tuning N to control the bias-variance compromize

Bias-variance compromize



- Total error = bias^2 + variance + irreducible error
- A more complex model (e.g. bigger network) generally has more variance, but less bias
- Tuning N in the N-step return helps finding the right compromize.

Generalized Advantage Estimation: λ return

- The N-step return can be reformulated using a continuous parameter λ
- $\hat{A}_\phi^{(\gamma,\lambda)} = \sum_{l=0}^{H} (\gamma\lambda)^l \delta_{t+l}$
- $\hat{A}_\phi^{(\gamma,0)} = \delta_t$ = one-step return
- $\hat{A}_\phi^{(\gamma,1)} = \sum_{l=0}^{H} (\gamma)^l \delta_{t+l}$ = 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

Computing $\hat{V}_\phi$ or $\hat{Q}_\phi$

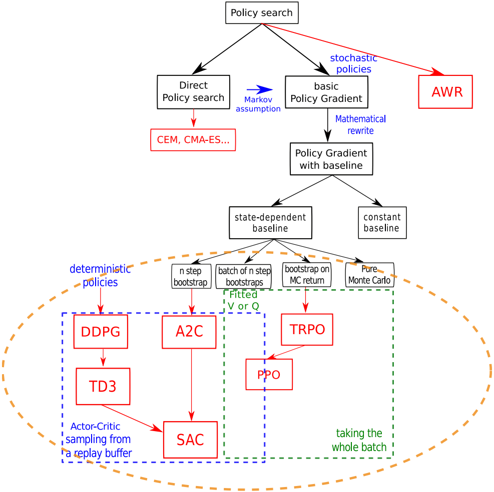
	Batch	Incremental
		One-step, N-step, or λ return
Monte Carlo: no bias, higher variance, lower sample efficiency	Batch Monte Carlo estimate TRPO	Incremental Monte Carlo estimate PPO
bootstrap: bias, lower variance, higher sample efficiency	Batch Temporal Difference estimate	Incremental Temporal Difference estimate (actor-critic) A2C, SAC

- There are many possibilities to approximate the policy gradient
  - Six proxies to advantage estimators
  - Batch, N-step return, λ return, one-step TD update...
- Results in a large variety of algorithms

Take home messages: DPS vs PG

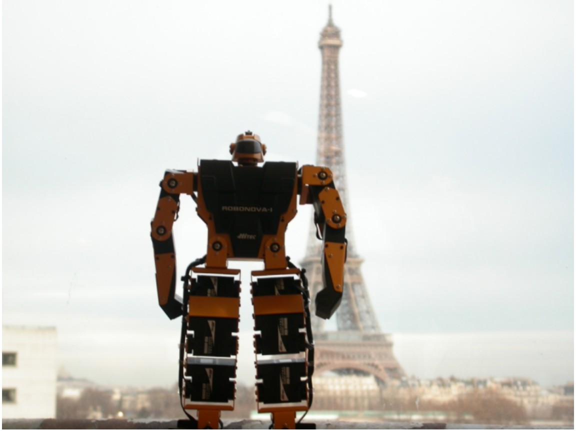
- Direct policy search:
  - optimization without a utility model
  - derivative-free
  - poor sample reuse, low sample efficiency
- Policy Gradient:
  - Uses analytical derivative of the policy function
  - Uses information from each step, not just trajectories
  - A baseline is used to reduce variance
  - When bootstrap comes into play, becomes actor-critic

Global view



- A secon video will focus on state-of-the-art algorithms

Any question?



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

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.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al.
A survey on policy search for robotics.
Foundations and Trends® in Robotics, 2(1–2):1–142, 2013.

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.

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.

Olivier Sigaud and Freek Stulp.
Policy search in continuous action domains: an overview.
Neural Networks, 113:28–40, 2019.

Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
MIT Press, 1998.

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

Ronald J. Williams.
Simple statistical gradient-following algorithms for connectionist reinforcement learning.
Machine Learning, 8(3-4):229–256, May 1992.
ISSN 0885-6125.
doi: 10.1007/BF00992696.