Policy Gradients

Christopher Mutschler

Policy Gradients

- Assume a state-action sequence in a complete trajectory with T steps (with s_T being a terminal state):

$$\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$$

- As usual,
  - $R(s_t, a_t)$ is the reward received after observing s_t and performing action a_t
  - $G(\tau) \coloneqq \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$ is the (discounted) sum of rewards (return)

- Our goal is to maximize the expected reward:

$$\max_\theta \mathbb{E}_{\pi_\theta} G(\tau)$$

(where π_θ is a parameterized policy, e.g., a neural network)

Policy Gradients

- But how do we maximize this?
  → Gradient Ascent! *Suppose* we know how to calculate the gradient w.r.t. the parameters:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)$$

- Then we can update our parameters θ with a learning rate α in the direction of the gradient:

$$\theta \leftarrow \theta + \alpha \underline{\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} G(\tau)}$$

Policy Gradient
often in literature referred to as $\nabla_\theta J(\pi_\theta)$

Policy Gradients

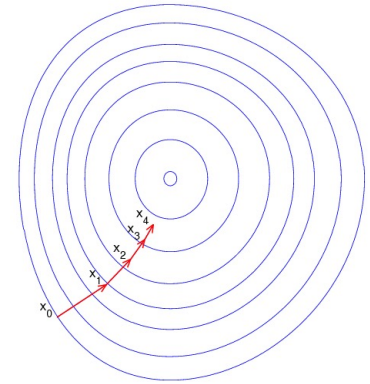
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy w.r.t. the parameters θ:

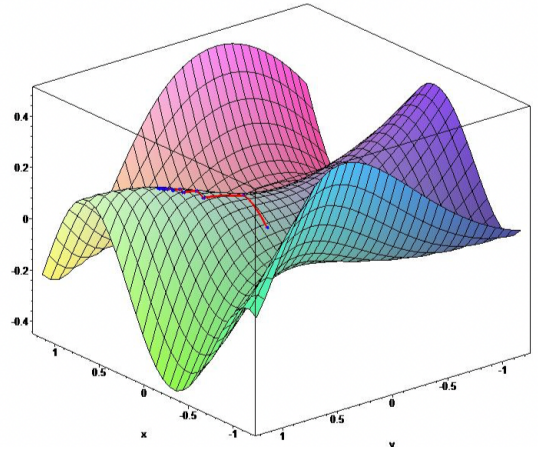
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

and α is a step-size parameter





David Silver. 2016.

Policy Gradients

Let $P(\tau|\theta)$ be the probability of a trajectory τ under policy π_θ, then

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \nabla_\theta \sum_\tau P(\tau|\theta) G(\tau)$$ definition of expectation

$$= \sum_\tau \nabla_\theta P(\tau|\theta) G(\tau)$$ swap sum/integral and gradient

$$= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta) G(\tau)$$ multiply and divide by $P(\tau|\theta)$

$$= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) G(\tau)$$ $\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$

$$= \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$ definition of expectation

https://www.janisklaise.com/post/rl-policy-gradients/

Policy Gradients

- The probability of a trajectory τ can be formulated as (note: MDP):

$$P(\tau|\theta) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

argh…we do not like this!

- But wait, let‘s take the gradient of the log-prob first:

$$\nabla_\theta \log P(\tau|\theta) = \nabla_\theta \left(\log p(s_0) + \sum_{t=0}^{T} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)) \right)$$

$$= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

dynamics model disappears!

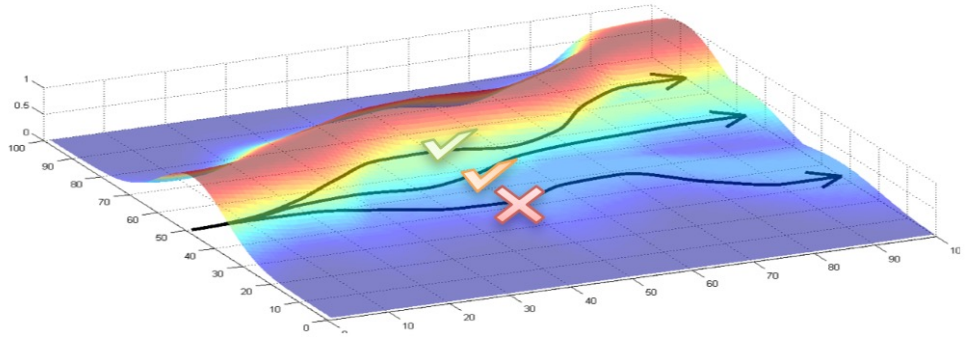
Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- But what is the intuition behind this gradient?

- The gradient tries to
  - Increase probability of paths with positive G
  - Decrease probability of paths with negative G



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

- As this is an expectation, we can estimate it with a sample mean using Monte-Carlo sampling of $|\tau| = L$ trajectories:

each action in the episode in influenced by the reward of the whole episode???
→ *reward-to-go* policy gradient

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)$$

Policy Gradients

- Plugging grad-log-prob into the gradient update gives:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}(\nabla_\theta \log P(\tau|\theta) G(\tau))$$
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right)$$

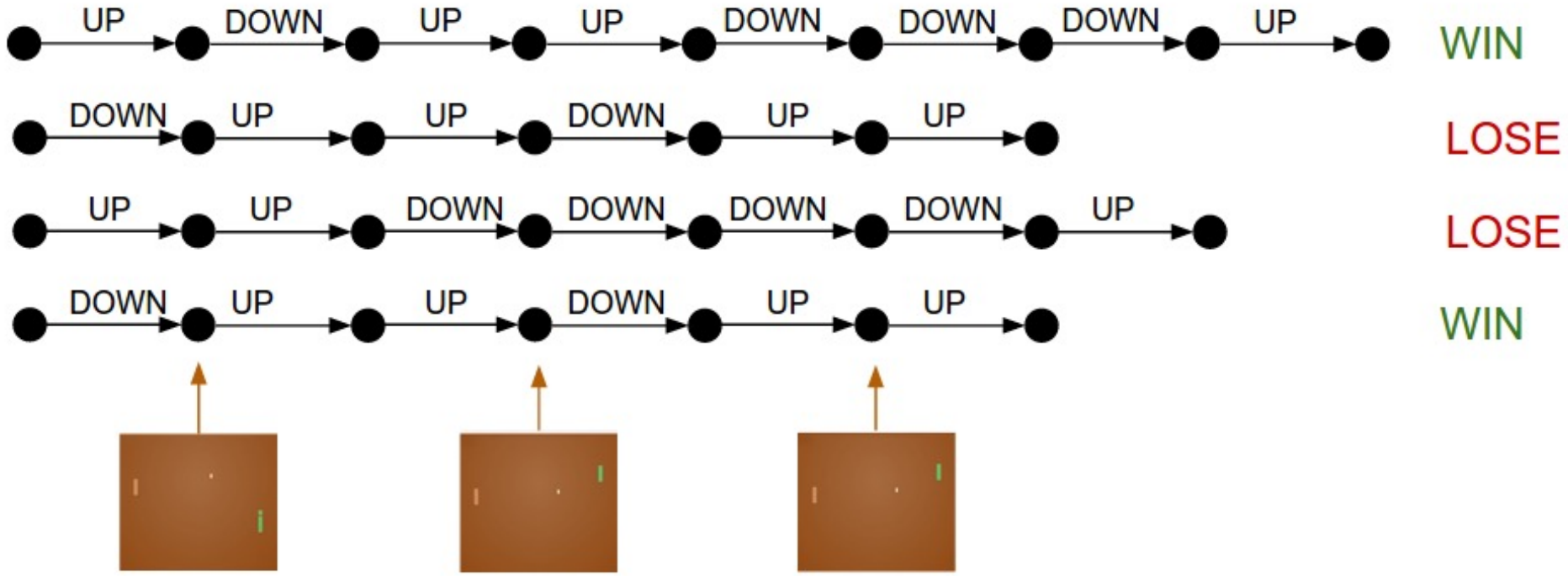
- Reduce variance: as this is an expectation, we can estimate it with a sample mean using Monte-Carlo sampling of L trajectories:

each action in the episode in influenced by the reward of the whole episode???
→ *reward-to-go* policy gradient

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) \approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)$$
$$\approx \frac{1}{L}\sum_{\tau}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'})$$

Policy Gradients

- How does the reward-to-go help?



http://karpathy.github.io/2016/05/31/rl/
Andrej Karpathy. DeepRL Bootcamp 4B

Policy Gradient: REINFORCE

- Monte-Carle Policy Gradient Control
  - Update parameters by stochastic gradient ascent
  - Using policy gradient theorem
  - Using return-to-go $Q^{\pi_\theta}(s_t, a_t)$ as an unbiased sample of G:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t)\, \gamma G_t$$

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

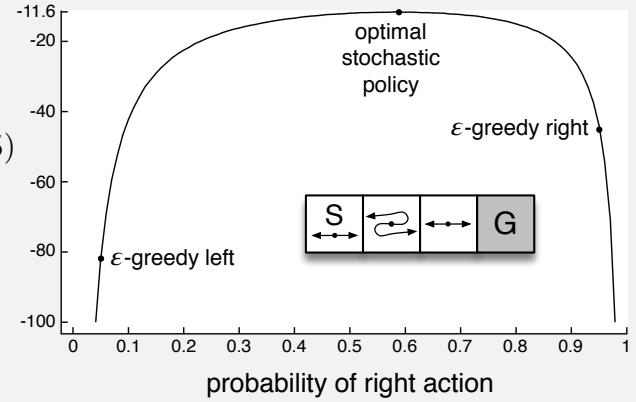
Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy Gradient: REINFORCE

Example 13.1 Short corridor with switched actions

Consider the small corridor gridworld shown inset in the graph below. The reward is -1 per step, as usual. In each of the three nonterminal states there are only two actions, right and left. These actions have their usual consequences in the first and third states (left causes no movement in the first state), but in the second state they are reversed, so that right moves to the left and left moves to the right. The problem is difficult because all the states appear identical under the function approximation. In particular, we define $\mathbf{x}(s, \text{right}) = [1, 0]^\top$ and $\mathbf{x}(s, \text{left}) = [0, 1]^\top$, for all s. An action-value method with ε-greedy action selection is forced to choose between just two policies: choosing right with high probability $1 - \varepsilon/2$ on all steps or choosing left with the same high probability on all time steps. If $\varepsilon = 0.1$, then these two policies achieve a value (at the start state) of less than -44 and -82, respectively, as shown in the graph. A method can do significantly better if it can learn a specific probability with which to select right. The best probability is about 0.59, which achieves a value of about -11.6.

$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathsf{S})$



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy Gradient: REINFORCE

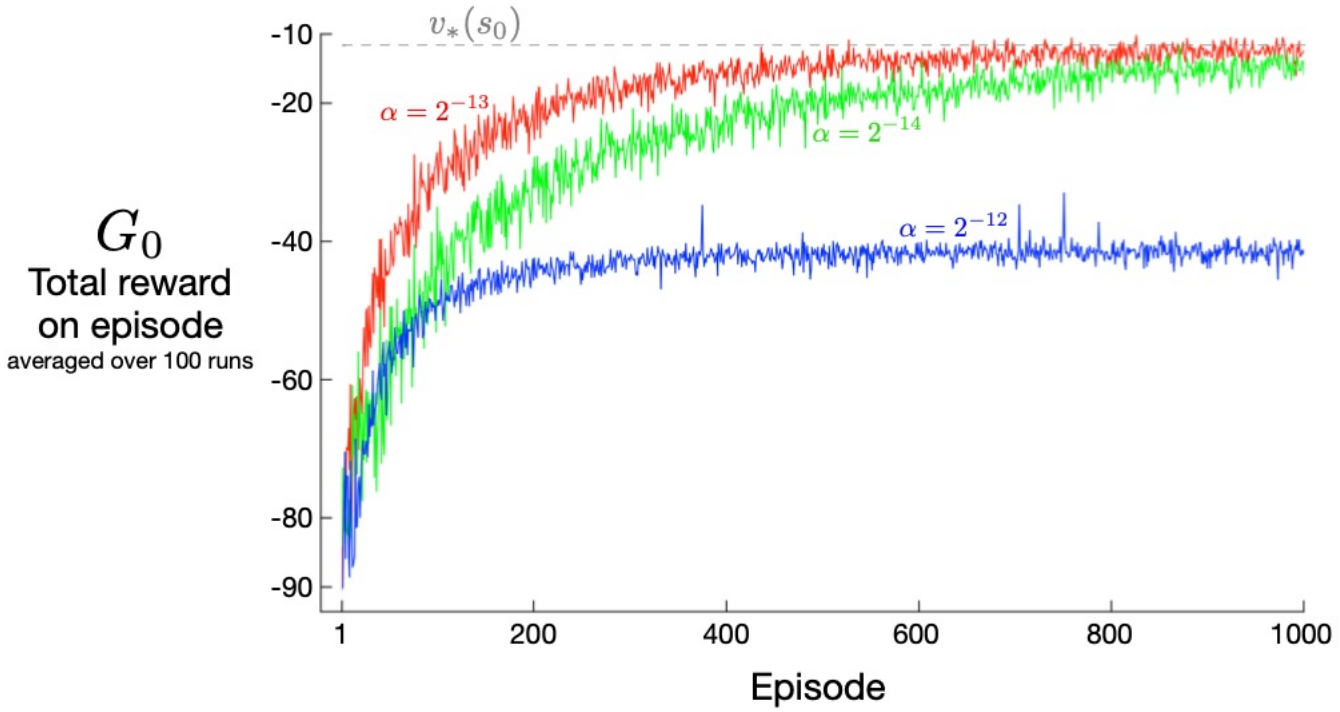


Figure 13.1: REINFORCE on the short-corridor gridworld (Example 13.1). With a good step size, the total reward per episode approaches the optimal value of the start state.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Policy Gradients

- Policy Gradient:
  - On-policy algorithm that also works w/ continuous action-spaces

```
Initialize a policy network randomly.
Repeat forever:
        collect a bunch of rollouts with the policy.
        Increase the probability of actions that worked well.
???
Profit.
```

- Summary:
  - \+ Good & easy-to-follow implementations available[1]
  - \+ Intuitive algorithm
  - \+ Easy to parallelize
  - \- High variance
  - \- Not capable of solving modern continuous action control problems

[1] *https://www.janisklaise.com/post/rl-policy-gradients*

Policy Gradients: where to hide the variance?

DIGRESSION

- Expected Grad-Log-Prob Lemma*:
  - P_θ is a parameterized probability distribution over a random variable x, then:

$$\mathbb{E}_{x\sim P_\theta}[\nabla_\theta \log P_\theta(x)] = 0$$

- From this follows that for any function b that *only depends on states*:

$$\mathbb{E}_{a_t\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)] = 0, \quad \text{because:}$$

$$= \mathbb{E}\left[\sum_a \pi(a|s_t) b \nabla_\theta \log \pi(a|s_t)\right]$$

$$= \mathbb{E}\left[b\nabla_\theta \sum_a \pi(a|s_t)\right]$$

$$= \mathbb{E}[b\nabla_\theta 1]$$

$$= 0$$

inverse score-function trick

prob-distribution and we sum over all actions

* You can find one version of the proof here: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#id1

Policy Gradients: where to hide the variance?

DIGRESSION

- This allows us to add or subtract any of such terms (i.e., **baseline functions**) to our policy gradient without changing its expectation:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta} G(\tau) = \mathbb{E}_{\tau\sim\pi_\theta}\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\,(G_t - b(s_t))\right)$$

$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} R(s_{t'}, a_{t'}) = Q^\pi(s_t, a_t)$

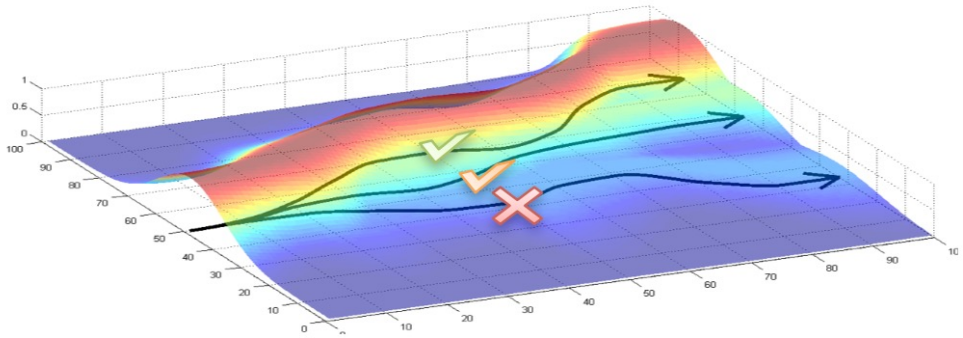
given that b does not depend on the action

** You can find one version of the proof here: https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#id1*

Advantage Functions: Intuition

- But what is the intuition behind this gradient?

- The gradient tries to
  - Increase probability of paths with positive G
  - Decrease probability of paths with negative G



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

- You mostly keep on increasing everything (but some more than others)
  - And this requires a lot of rollouts to average the effect out!
  - Ideal: in-/decrease probs of paths that are better/worse nach the average!

Advantage Functions: Introduce a Baseline

But how does this help?

- We can introduce a baseline to our targets!
- We can define *advantage functions* that reduce variance
- Intuition: if an agent sees what it expected, it feels *neutral* about it
- *Hint: we already did this when we introduced the reward-to-go variant!*

- For instance, we can use $b(s_t) = v^{\pi}(s_t)$ to reduce variance in the sample estimate of the policy gradient:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

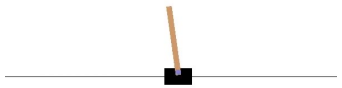
→ faster and more stable policy learning!

Advantage Functions: Intuition

Question: what helps our agent learn faster: information about

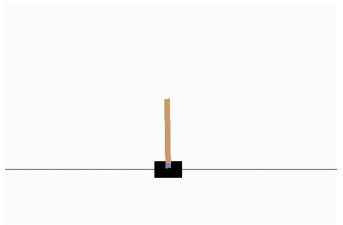
1. *invariant actions on good states,* or
2. information about *good actions in bad/challenging states*?

Iteration 100



Q(s2, right) = 50
V(s2) = 10
A(s2, right) = 40

Iteration 1000



Q(s1, do nothing) = 100
V(s1) = 100
A(s1, do nothing) = 0

REINFORCE w/ Baselines

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$ ← TD-error, or *advantage*
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$ ← one-step TD-error update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$ ← policy gradient update

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

REINFORCE w/ Baselines

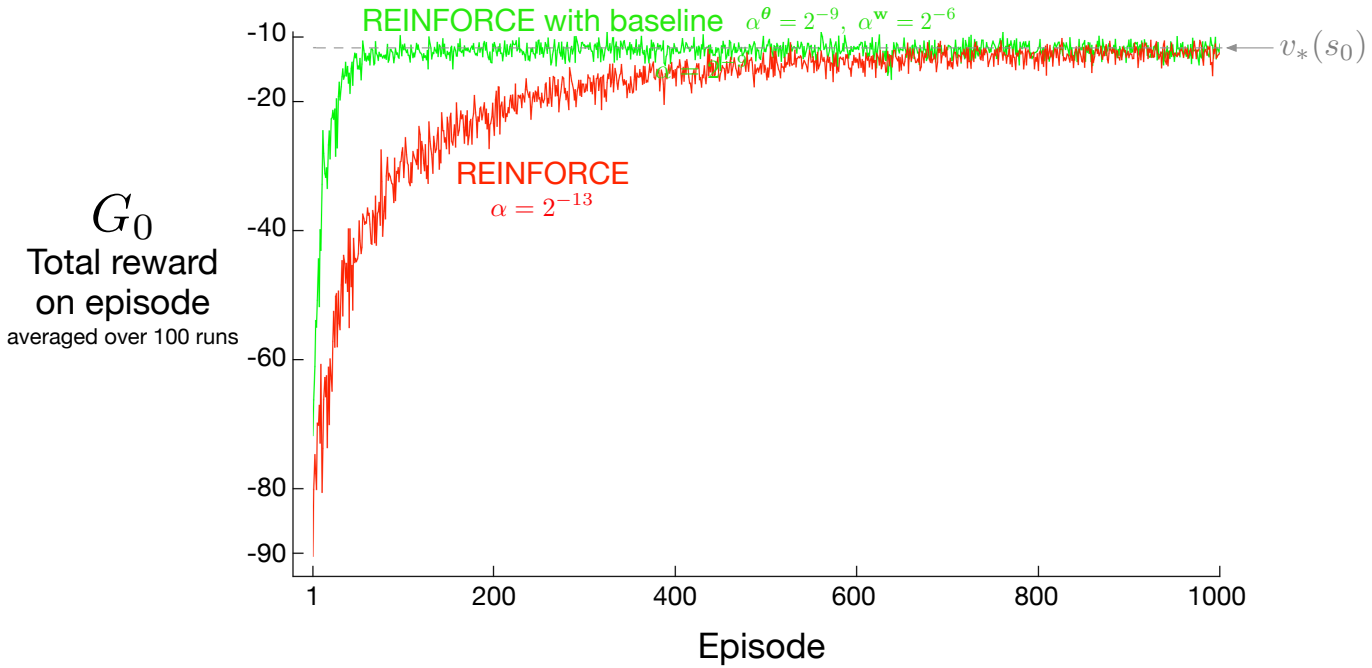


Figure 13.2: Adding a baseline to REINFORCE can make it learn much faster, as illustrated here on the short-corridor gridworld (Example 13.1). The step size used here for plain REINFORCE is that at which it performs best (to the nearest power of two; see Figure 13.1).

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Vanilla Policy Gradient Algorithm

Problem:
- Learning rate is a heuristic that cannot be easily tuned
- If we take large steps (especially in the beginning of training) we might never recover

Pseudocode

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\big|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

Estimate the Advantages

Calculate the gradient and take a gradient step

Update the Critic (used for the Advantage estimation)

Policy Gradients

- Exploration vs. Exploitation of Policy Gradient methods:
  - PG trains a stochastic policy in an on-policy way
  - Actions are sampled from the environment according to the latest version of its stochastic policy
  - Randomness in selecting actions
    - (initially) depends on the initialization
    - Becomes less over the course of training
    - …as the update rule encourages to exploit rewards that the policy already has found
  - May result in local optima

References

- Deisenroth, M. P., Neumann, G., & Peters, J. (2013). A survey on policy search for robotics. Foundations and Trends® in Robotics, 2(1–2), 1-142: https://spiral.imperial.ac.uk/bitstream/10044/1/12051/7/fnt_corrected_2014-8-22.pdf
- Sigaud, O., & Stulp, F. (2019). Policy search in continuous action domains: an overview. Neural Networks. ArXiv: https://arxiv.org/pdf/1803.04706.pdf
- Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems (pp. 1057-1063). Link: http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf
- Kakade, S. M. (2002). A natural policy gradient. In Advances in neural information processing systems (pp. 1531-1538). Link: http://papers.nips.cc/paper/2073-a-natural-policy-gradient.pdf
- Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016, June). Benchmarking deep reinforcement learning for continuous control. In International Conference on Machine Learning (pp. 1329-1338): http://proceedings.mlr.press/v48/duan16.pdf
- Riedmiller, M., Peters, J., & Schaal, S. (2007, April). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (pp. 254-261). IEEE. link: http://is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/ADPRL2007-Peters2_[0].pdf
- Kober, J., & Peters, J. R. (2009). Policy search for motor primitives in robotics. In Advances in neural information processing systems (pp. 849-856). Link: https://papers.nips.cc/paper/3545-policy-search-for-motor-primitives-in-robotics
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4), 229-256.
- Vlassis, N., Toussaint, M., Kontes, G., & Piperidis, S. (2009). Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots, 27(2), 123-130.