Policy Gradient Methods

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

Summary

- Introduction
- Policy Approximation and its Advantages
- The Policy Gradient Theorem
- REINFORCE: Monte-Carlo Policy Gradient
- REINFORCE with baseline
- Actor-Critic Methods

Introduction

Introduction

All methods seen **so far** are **action-value methods**. They:

- **estimate** action values
- **select** actions based on these values
- but **do not explicitly represent the policy function**

Policy gradient methods are different. They:

- **learn** a **parametrized policy function**
- **selects** actions using this **policy** and **without** consulting **value functions**

A **value function** can be used **only to learn policy parameters**

Actor-critic methods are **policy gradient** methods that learn **also** approximations of the **value function**

- **Actor:** learned policy
- **Critic:** learned value function

Introduction

Some **notation**:

- $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ is the **policy's parameter** vector

- $\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ is the probability that action *a* is taken at time *t* given the environment is in state *s* and policy parameters are $\boldsymbol{\theta}$

- $\hat{v}(s, \mathbf{w})$ is the learned **value function**, if required by the method, with parameters $\mathbf{w} \in \mathbb{R}^d$

- $J(\boldsymbol{\theta})$ is a measure of **policy performance** depending on policy parameters

The **goal** of policy gradient methods is to learn parameters $\boldsymbol{\theta}$ that maximize $J(\boldsymbol{\theta})$

Introduction

- The **goal** of policy gradient methods is to learn parameters $\boldsymbol{\theta}$ that maximize $J(\boldsymbol{\theta})$

- **Parameter updates** approximate **gradient ascent** in J :

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

  where $\widehat{\nabla J(\boldsymbol{\theta}_t)} \in \mathbb{R}^{d'}$ is a **stochastic estimate of the gradient** of $J(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}_t$

- **Episodic case:** performance is the **value of the start state** under the parametrized policy

- **Continuing case:** performance is the **average reward rate**

Policy Approximation and its Advantages

Policy Approximation and its Advantages

- The **policy** can be **parametrized** in any way as long as $\pi(a|s, \boldsymbol{\theta})$ is **differentiable** w.r.t. its parameters

- To ensure **exploration** we require the policy **never** become **deterministic**, i.e., $\pi(a|s, \boldsymbol{\theta}) \in (0, 1)$

- **Discrete (and not too large) action space:** a natural **parametrization** are numerical **preferences** $h(s, a, \boldsymbol{\theta}) \in \mathbb{R}$ for each state-action pair

- **Probability** is assigned to **actions proportionally to preferences**, e.g., according to **exponential soft-max distribution** (called **soft-max in action preferences**)

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

Policy Approximation and its Advantages

- **Preferences** can themselves be **parametrized** arbitrarily

- A **deep ANN** can be used to compute **preferences** (as in AlphaGo). In this case $\boldsymbol{\theta}$ is the vector of connection weights

- Or the preferences could be **linear in the features**:
  $h(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s, a)$ with $\mathbf{x}(s, a) \in \mathbb{R}^{d'}$ features of the policy

- **Action values could be used as preferences with soft-max** but this would not allow the policy to approach **deterministic** behaviours

- Instead, **general action preferences** do not have to approach specific values allowing them to approach also deterministic policies
  - E.g., preferences of **optimal** actions can be driven **infinitely higher** than all **suboptimal** actions

Policy Approximation and its Advantages

Advantages of policy parametrization vs action value parametrization:

- The **policy** may be a **simpler** function to approximate
- Policy parametrization **allows to inject prior knowledge:** this is **often the most important reason to use for using a policy-based learning method**
- **Action-value** methods have no natural way of finding **stochastic policies**, while **policy gradient** methods (e.g., with soft-max in action preferences) enables the selection of actions with **arbitrary probabilities (e.g., stochastic policies)**
- **Policy-based methods** can deal with **continuous action spaces**

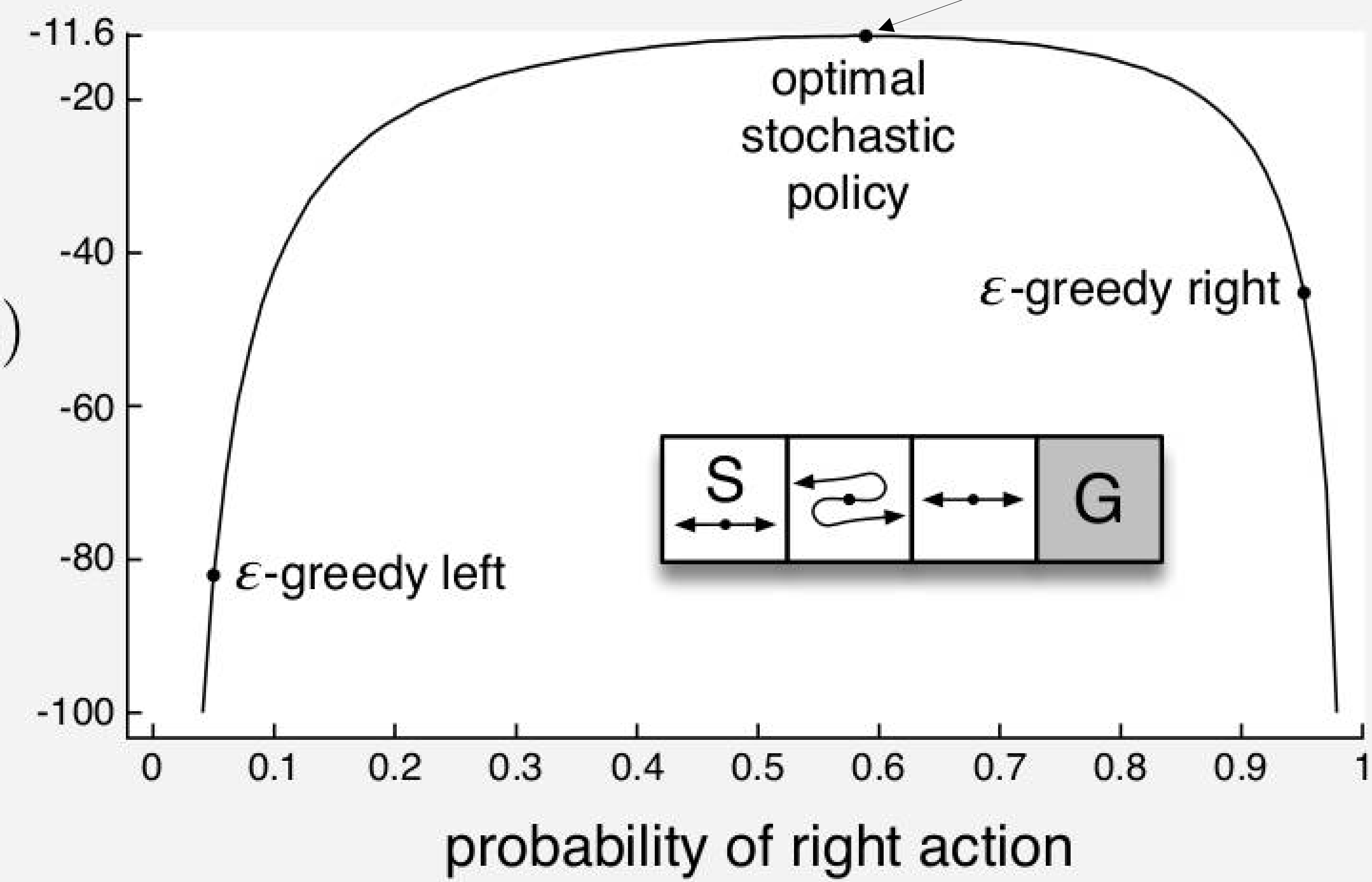
Example: Short corridor with switched actions

- **Reward**: -1 per step
- **Actions**: left, right
- **State features**:
  - $x(s,\text{right}) = [1,0]^T$
  - $x(s,\text{left}) = [0,1]^T$

  for all states *s*

- **Action value** method: 2 possible policies
  - 1. Right with probability $(1-\varepsilon)/2$
  - 2. Left with probability $(1-\varepsilon)/2$
- **Policy gradient** method:
  - Best probability to select right: **0.59**

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



The Policy Gradient Theorem

The Policy Gradient Theorem

- With **continuous policy parametrization** the action **probabilities change smoothly** as a function of parameters, instead in **action value methods** with ε-greedy selection action **probabilities may change dramatically** for small changes of action values

- Because of this, **stronger convergence guarantees ara available for policy gradient methods**

- Given the **performance** of the episodic case $J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$ and assuming **no discounting** (i.e., $\gamma = 1$)

- **How can we change the policy parameters in a way that ensure improvement?**

- Performance depends on both **action selection** and **distribution of states** and both are affected by the policy parameters

The Policy Gradient Theorem

Given a **state**:

- The effect of **policy parameters** on **actions** and therefore **reward** can be computed
- The effect of the **policy** on the **state distribution** is a function of the environment which is typically **unknown** (we are in a model-free setting) -> **Problem!**

- **Question: How** can we **estimate** the **performance gradient** w.r.t. the **policy parameters** when the gradient **depends** on the **unknown** effect of policy changes on the **state distribution**?

- The **Policy Gradient Theorem** answers to this question with an **analytic expression for the gradient of the performance w.r.t. policy parameter that does not involve the derivative of the state distribution**

The Policy Gradient Theorem

The **Policy Gradient Theorem** for the **episodic case** estabilishes that:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$

where the **gradients** are column vectors of partial derivatives w.r.t. the components of $\boldsymbol{\theta}$ and $\mu(s)$ is the **on-policy state distribution** under policy π (parametrized by $\boldsymbol{\theta}$)

- The **constant of proportionality** is
  - the **average length of an episode** in the episodic case
  - **1** in the continuing case

- Proof: page 325 of the book

The Policy Gradient Theorem (proof)

$$\nabla v_\pi(s) = \nabla\left[\sum_a \pi(a|s)q_\pi(s,a)\right], \quad \text{for all } s \in \mathcal{S} \qquad \text{(Exercise 3.18)}$$

$$= \sum_a \Big[\nabla\pi(a|s)q_\pi(s,a) + \pi(a|s)\nabla q_\pi(s,a)\Big] \quad \text{(product rule of calculus)}$$

$$= \sum_a \Big[\nabla\pi(a|s)q_\pi(s,a) + \pi(a|s)\nabla \sum_{s',r} p(s',r|s,a)\big(r + v_\pi(s')\big)\Big]$$

$$\text{(Exercise 3.19 and Equation 3.2)}$$

$$= \sum_a \Big[\nabla\pi(a|s)q_\pi(s,a) + \pi(a|s)\sum_{s'} p(s'|s,a)\nabla v_\pi(s')\Big] \qquad \text{(Eq. 3.4)}$$

$$= \sum_a \Big[\nabla\pi(a|s)q_\pi(s,a) + \pi(a|s)\sum_{s'} p(s'|s,a) \qquad \text{(unrolling)}$$

$$\sum_{a'}\big[\nabla\pi(a'|s')q_\pi(s',a') + \pi(a'|s')\sum_{s''} p(s''|s',a')\nabla v_\pi(s'')\big]\Big]$$

$$= \sum_{x\in\mathcal{S}} \sum_{k=0}^{\infty} \Pr(s\to x, k, \pi) \sum_a \nabla\pi(a|x)q_\pi(x,a),$$

Where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state *s* to state *x* in *k* steps under policy π

The Policy Gradient Theorem (proof)

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(box page 199)} \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Eq. 9.3)} \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Q.E.D.)}
\end{aligned}
$$

REINFORCE: Monte Carlo Policy Gradient

REINFORCE: Monte Carlo Policy Gradient

- Given the strategy of **stochastic gradient ascent** seen at the beginning

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

we need a way to obtain samples such that the **expectation** of the **sample gradient** $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ is **proportional** to the **actual gradient** $\nabla J(\boldsymbol{\theta}_t)$

- The **policy gradient theorem** provides an **exact expression proportional to the gradient**, hence **we use it for sampling** from that expression

REINFORCE: Monte Carlo Policy Gradient

- We have that $$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta})$$ $$= \mathbb{E}_\pi\left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta})\right]$$

  since following π the states are encountered according to distribution $\mu(s)$

- Then, we can instantiate a **first stochastic gradient-ascent algorithm** as $$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \underbrace{\sum_a \hat{q}(S_t,a,\mathbf{w}) \nabla \pi(a|S_t,\boldsymbol{\theta})}_{\widehat{\nabla J(\boldsymbol{\theta}_t)}}$$

  where $\hat{q}$ is some **learned approximation** of q_π

- We call this algorithm **all-actions** because its update involves all of the actions

REINFORCE: Monte Carlo Policy Gradient

- If we consider instead **only the action** A_t **taken at time *t*** we obtain the **REINFORCE algorithm**

- To derive it we first take the last formula of the gradient $\nabla J(\boldsymbol{\theta})$ and multiply and divide the summed terms by $\pi(a|S_t, \boldsymbol{\theta})$

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi\left[\sum_a \boxed{\pi(a|S_t, \boldsymbol{\theta})} q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\boxed{\pi(a|S_t, \boldsymbol{\theta})}}\right]$$

As done with *s* in the previous slide

$$= \mathbb{E}_\pi\left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}\right] \qquad \text{(replacing } a \text{ by the sample } A_t \sim \pi)$$

$$= \mathbb{E}_\pi\left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}\right], \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t))$$

where G_t is the **return**, as usual

- The **final expression** is what we need: **a quantity that can be sampled on each time step, whose expectation is equal to the gradient**

REINFORCE: Monte Carlo Policy Gradient

- The parameter **update rule of the REINFORCE algorithm** is therefore:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

- **Idea:** each increment is a product of a return G_t and the vector $\frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$

- Vector $\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)$ is the **direction in the parameter space** that **most increases the probability of repeating the action** A_t of **future visits** of state S_t

- The update **increases the parameter vector in this direction** **proportional** to the **return** and **inversely proportional** to the **action probability**

Causes the parameters to move most in the direction that favour highest returns

Remove advantage of actions selected most frequently, they could not bring the highest return

REINFORCE: Monte Carlo Policy Gradient

- REINFORCE uses the **complete return** G_t from time t (i.e., all rewards until the end of the episode)

- REINFORCE is a **Monte Carlo algorithm** and it is well defined for the **episodic** case

REINFORCE: Monte Carlo Policy Gradient (Williams 1992)

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)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \boxed{\ln \pi(A_t|S_t, \boldsymbol{\theta})}$

$$\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}_t) = \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

since $\nabla \ln x = \frac{\nabla x}{x}$

This holds for the general discount case with γ

REINFORCE: Monte Carlo Policy Gradient

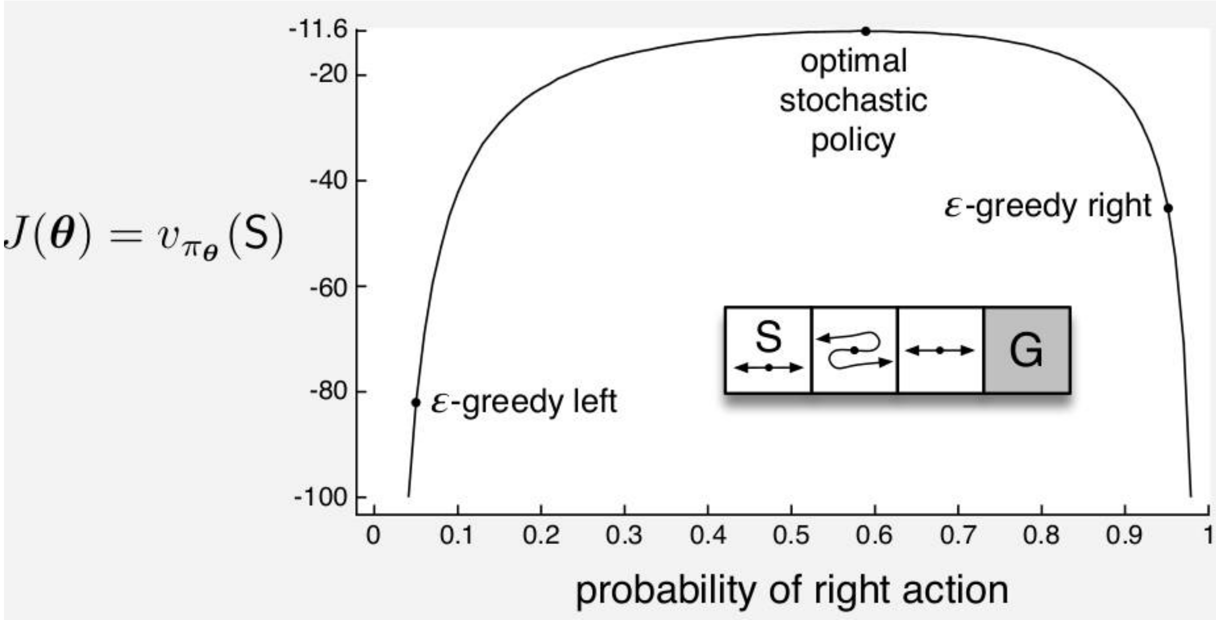
- REINFORCE has **good theoretical convergence properties**

- The expected update over an episode is in the same direction as the performance gradient

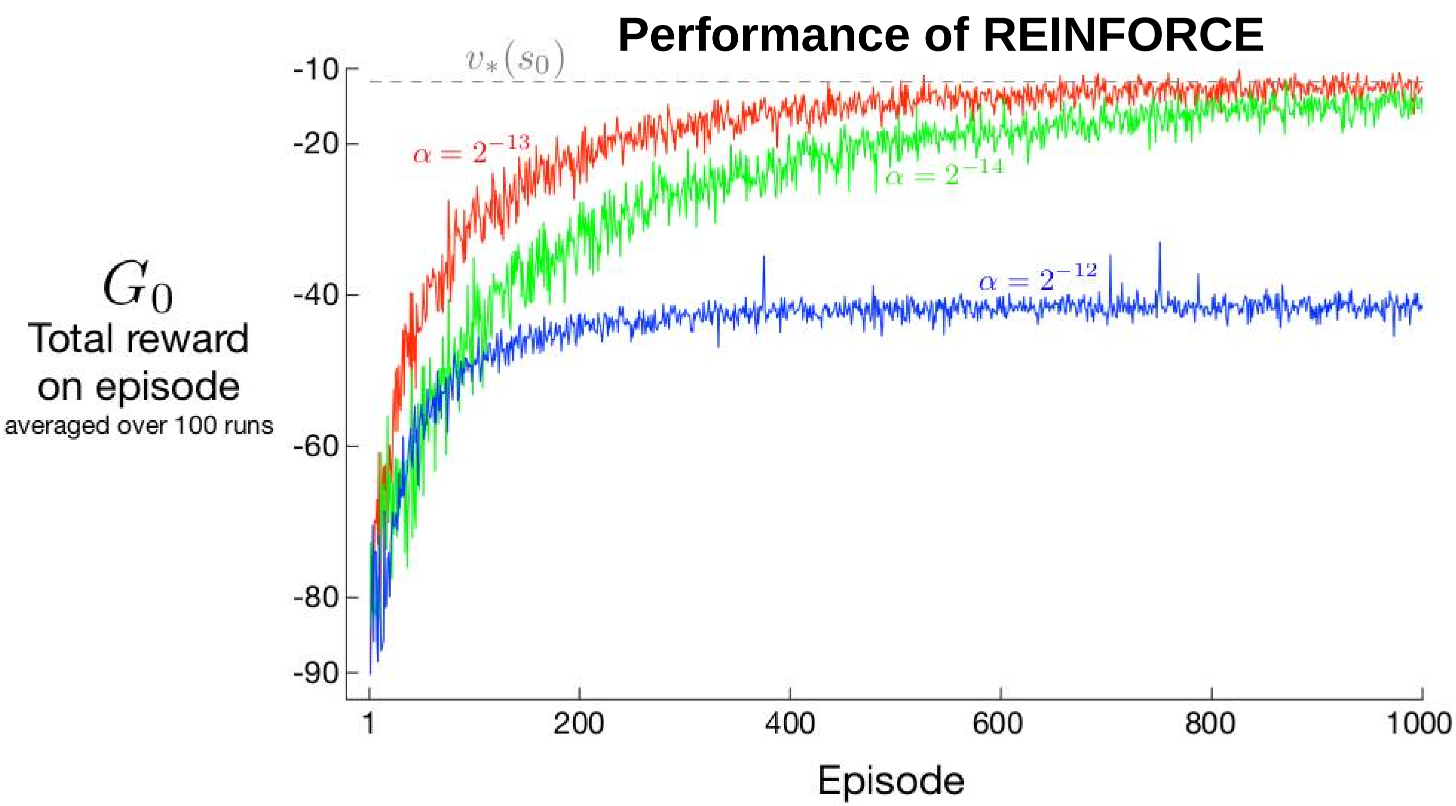
- This assures **improvement of expected performance** for sufficiently **small** α and **convergence to a local optimum** under standard stochastic approximation conditions (Ch. 2 Sutton and Barto) for decreasing α

- As a **Monte Carlo** method REINFORCE **may** be of **high variance** and thus produce **slow learning**

Example: Short corridor with switched actions with REINFORCE



- Reward: -1 per step
- Actions: left, right
- All the states appear identical under
  the function approximation (model free RL)
- $\mathbf{x}(s, \text{right}) = [1, 0]^\top$ $\mathbf{x}(s, \text{left}) = [0, 1]^\top$ Features for all states



REINFORCE with Baseline

REINFORCE with Baseline

- The **policy gradient theorem** can be **generalized** to include a **comparison** of the action value to an arbitrary baseline *b(s)*

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s,a) - b(s) \Big) \nabla \pi(a|s,\boldsymbol{\theta})$$

- The equation remains valid **if the baseline does not vary with *a*** because the subctracted quantity is zero:

$$\sum_a b(s) \nabla \pi(a|s,\boldsymbol{\theta}) \;=\; b(s) \nabla \sum_a \pi(a|s,\boldsymbol{\theta}) \;=\; b(s) \nabla 1 \;=\; 0$$

- A new version of **REINFORCE** can be derived using the **update rule** that includes a general **baseline**:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \Big(G_t - b(S_t) \Big) \frac{\nabla \pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}$$

REINFORCE with Baseline

- The baseline leaves the expected value of the update unchanged

- But it can have a **positive effect on the variance**

- **The baseline can vary with the state**
  - In some states all actions have high values → **high baseline** to differentiate the higher valued actions from the less highly valued ones
  - In other states all actions have low values → **low baseline** is appropriate

- One **natural choice** for the baseline is an **estimate** of the **state value** $\hat{v}(S_t, \mathbf{w})$ where **w** is learned with **value based methods**

- If we use **Monte Carlo** to learn **both w** and $\boldsymbol{\theta}$ we obtain the following algorithm

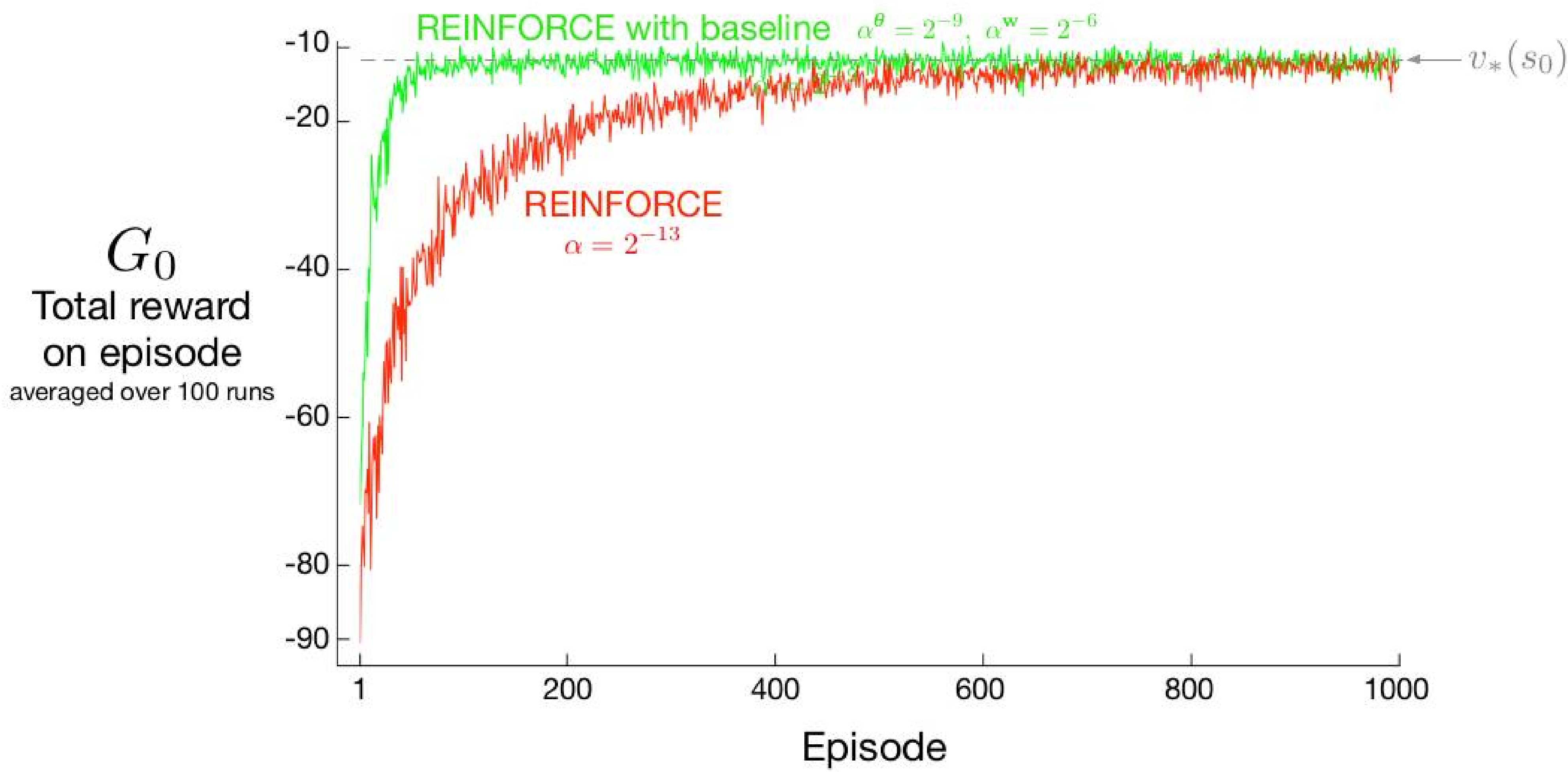
REINFORCE with Baseline

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 $\boxed{\hat{v}(s,\mathbf{w})}$
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)
    $\boxed{\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})}$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \boxed{\delta \nabla \hat{v}(S_t,\mathbf{w})}$ → See Eq. 9.7 of the book
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \boxed{\delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})}$

REINFORCE with Baseline



Actor-Critic Methods

Actor-Critic Methods

- **REINFORCE-with-baseline** learns both:
  - A **policy**
  - A **state-value** function (i.e., the baseline)

- However, we do **not** consider it an **actor-critic** method, because the **state-value function** is **not** used as a **critic**

- The value-function is **not used for bootstrapping**, namely, for **updating the value estimate for a state** from the **estimated values of subsequent states**

- It is used only as a **baseline** for the **state** whose estimate is being updated

Actor-Critic Methods

- As seen before, the bias introduced by bootstrapping is often beneficial because it reduces variance and accellerate learning

- **REINFORCE with baseline** is **unbiased** and converges asymptotically to a local minimum

- **Monte-Carlo methods** tend to learn slowly (estimates have high variance) and to be inconvenient to implement online of for continuing problems

- **Temporal-Difference methods** can eliminate these inconveniences

- To gain advantages of TD in policy gradient methods we introduce
- **Actor-Critic methods with a Bootstrapping critic**

Actor-Critic Methods

- **One-step actor-critic methods** are the **policy-gradient analog** of the TD methods introduced before, i.e., **TD, Sarsa, Q-learning**

- They are fully online and incremental

- They **replace** the **full return** of REINFORCE with the **one-step-return** and use a learned state-value function as the baseline

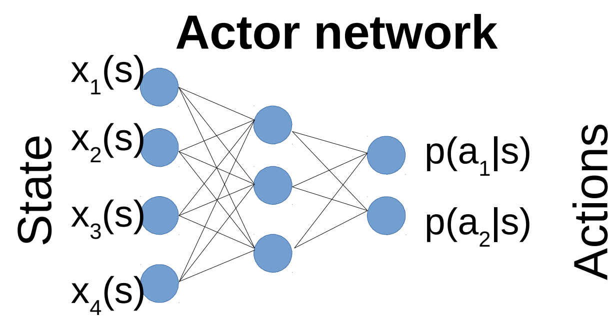
- The update rule becomes:

Update rule of REINFORCE with baseline

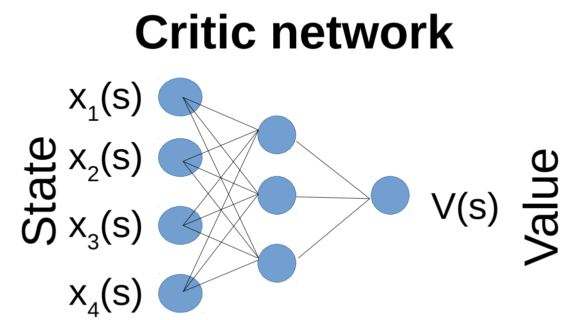
$$
\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha\Big(\underbrace{\boxed{G_{t:t+1}}}_{\text{Target}} - \underbrace{\boxed{\hat{v}(S_t,\mathbf{w})}}_{\text{Baseline}}\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha\Big(\boxed{R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w})} - \hat{v}(S_t,\mathbf{w})\Big)\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha\delta_t\frac{\nabla\pi(A_t|S_t,\boldsymbol{\theta}_t)}{\pi(A_t|S_t,\boldsymbol{\theta}_t)}.
\end{aligned}
$$

Actor-Critic Methods

- The natural way to **learn the state-value function** in this context is **semi-gradient TD(0)** (see methods for On-policy prediction with approximation)



- $\pi(A_t|S_t,\boldsymbol{\theta}_t)$: actor



- $\hat{v}(S_t,\mathbf{w})$: critic
- $R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w})$: target
- $R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$: **advantage** (=TD error)
- The **advantage** tells us if a state is better or worse than expected. If the action is better than expected (advantage > 0) then we want to incourage this action. Aotherwise (advantage < 0) we want to encourage the opposite action

Actor-Critic Methods

- The natural way to **learn the state-value function** in this context is **semi-gradient TD(0)** (see methods for On-policy prediction with approximation)

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$ (A2C)

Input: a differentiable policy parameterization $\boxed{\pi(a|s, \boldsymbol{\theta})}$
Input: a differentiable state-value function parameterization $\boxed{\hat{v}(s,\mathbf{w})}$
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):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 13