function). The sum of the forward-view update over time is

$$\begin{aligned}
\sum_{t=0}^{\infty}(\mathbf{w}_{t+1}-\mathbf{w}_t) &\approx \sum_{t=0}^{\infty}\sum_{k=t}^{\infty}\alpha\rho_t\delta_k^s\nabla\hat{v}(S_t,\mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i \\
&= \sum_{k=0}^{\infty}\sum_{t=0}^{k}\alpha\rho_t\nabla\hat{v}(S_t,\mathbf{w}_t)\delta_k^s\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i \\
&\qquad \text{(using the summation rule: } \textstyle\sum_{t=x}^{y}\sum_{k=t}^{y}=\sum_{k=x}^{y}\sum_{t=x}^{k}\text{)} \\
&= \sum_{k=0}^{\infty}\alpha\delta_k^s\sum_{t=0}^{k}\rho_t\nabla\hat{v}(S_t,\mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i,
\end{aligned}$$

which would be in the form of the sum of a backward-view TD update if the entire expression from the second sum on could be written and updated incrementally as an eligibility trace, which we now show can be done. That is, we show that if this expression was the trace at time k, then we could update it from its value at time $k-1$ by:

$$\begin{aligned}
\mathbf{z}_k &= \sum_{t=0}^{k}\rho_t\nabla\hat{v}(S_t,\mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i \\
&= \sum_{t=0}^{k-1}\rho_t\nabla\hat{v}(S_t,\mathbf{w}_t)\prod_{i=t+1}^{k}\gamma_i\lambda_i\rho_i \quad + \quad \rho_k\nabla\hat{v}(S_k,\mathbf{w}_k) \\
&= \gamma_k\lambda_k\rho_k\underbrace{\sum_{t=0}^{k-1}\rho_t\nabla\hat{v}(S_t,\mathbf{w}_t)\prod_{i=t+1}^{k-1}\gamma_i\lambda_i\rho_i}_{\mathbf{z}_{k-1}} \quad + \quad \rho_k\nabla\hat{v}(S_k,\mathbf{w}_k) \\
&= \rho_k\big(\gamma_k\lambda_k\mathbf{z}_{k-1}+\nabla\hat{v}(S_k,\mathbf{w}_k)\big),
\end{aligned}$$

which, changing the index from k to t, is the general accumulating trace update for state values:

$$\mathbf{z}_t \doteq \rho_t\big(\gamma_t\lambda_t\mathbf{z}_{t-1}+\nabla\hat{v}(S_t,\mathbf{w}_t)\big), \tag{12.25}$$

This eligibility trace, together with the usual semi-gradient parameter-update rule for TD(λ) (12.7), forms a general TD(λ) algorithm that can be applied to either on-policy or off-policy data. In the on-policy case, the algorithm is exactly TD(λ) because ρ_t is alway 1 and (12.25) becomes the usual accumulating trace (12.5) (extended to variable λ and γ). In the off-policy case, the algorithm often works well but, as a semi-gradient method, is not guaranteed to be stable. In the next few sections we will consider extensions of it that do guarantee stability.

A very similar series of steps can be followed to derive the off-policy eligibility traces for *action*-value methods and corresponding general Sarsa(λ) algorithms. One could start with either recursive form for the general action-based λ-return, (12.19) or (12.20), but the latter (the Expected Sarsa form) works out to be simpler. We extend (12.20) to the

As a stochastic gradient method, REINFORCE has good theoretical convergence properties. By construction, the expected update over an episode is in the same direction as the performance gradient. This assures an improvement in expected performance for sufficiently small α, and convergence to a local optimum under standard stochastic approximation conditions for decreasing α. However, as a Monte Carlo method REINFORCE may be of high variance and thus produce slow learning.

Exercise 13.3 In Section 13.1 we considered policy parameterizations using the soft-max in action preferences (13.2) with linear action preferences (13.3). For this parameterization, prove that the eligibility vector is

$$\nabla \ln \pi(a|s, \boldsymbol{\theta}) = \mathbf{x}(s, a) - \sum_b \pi(b|s, \boldsymbol{\theta})\mathbf{x}(s, b), \tag{13.9}$$

using the definitions and elementary calculus. □

13.4 REINFORCE with Baseline

The policy gradient theorem (13.5) can be generalized to include a comparison of the action value to an arbitrary *baseline* $b(s)$:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a \Big(q_\pi(s, a) - b(s)\Big) \nabla \pi(a|s, \boldsymbol{\theta}). \tag{13.10}$$

The baseline can be any function, even a random variable, as long as it does not vary with a; the equation remains valid because the subtracted quantity is zero:

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

The policy gradient theorem with baseline (13.10) can be used to derive an update rule using similar steps as in the previous section. The update rule that we end up with is a new version of REINFORCE that includes a general baseline:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \Big(G_t - b(S_t)\Big) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}. \tag{13.11}$$

Because the baseline could be uniformly zero, this update is a strict generalization of REINFORCE. In general, the baseline leaves the expected value of the update unchanged, but it can have a large effect on its variance. For example, we saw in Section 2.8 that an analogous baseline can significantly reduce the variance (and thus speed the learning) of gradient bandit algorithms. In the bandit algorithms the baseline was just a number (the average of the rewards seen so far), but for MDPs the baseline should vary with state. In some states all actions have high values and we need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions will have low values and a low baseline is appropriate.

One natural choice for the baseline is an estimate of the state value, $\hat{v}(S_t,\mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^d$ is a weight vector learned by one of the methods presented in previous chapters.

Because REINFORCE is a Monte Carlo method for learning the policy parameter, $\boldsymbol{\theta}$, it seems natural to also use a Monte Carlo method to learn the state-value weights, $\mathbf{w}$. A complete pseudocode algorithm for REINFORCE with baseline using such a learned state-value function as the baseline is given in the box below.

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 $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

This algorithm has two step sizes, denoted $\alpha^{\boldsymbol{\theta}}$ and $\alpha^{\mathbf{w}}$ (where $\alpha^{\boldsymbol{\theta}}$ is the α in (13.11)). Choosing the step size for values (here $\alpha^{\mathbf{w}}$) is relatively easy; in the linear case we have rules of thumb for setting it, such as $\alpha^{\mathbf{w}} = 0.1/\mathbb{E}\big[\|\nabla \hat{v}(S_t,\mathbf{w})\|_\mu^2\big]$ (see Section 9.6). It is much less clear how to set the step size for the policy parameters, $\alpha^{\boldsymbol{\theta}}$, whose best value depends on the range of variation of the rewards and on the policy parameterization.

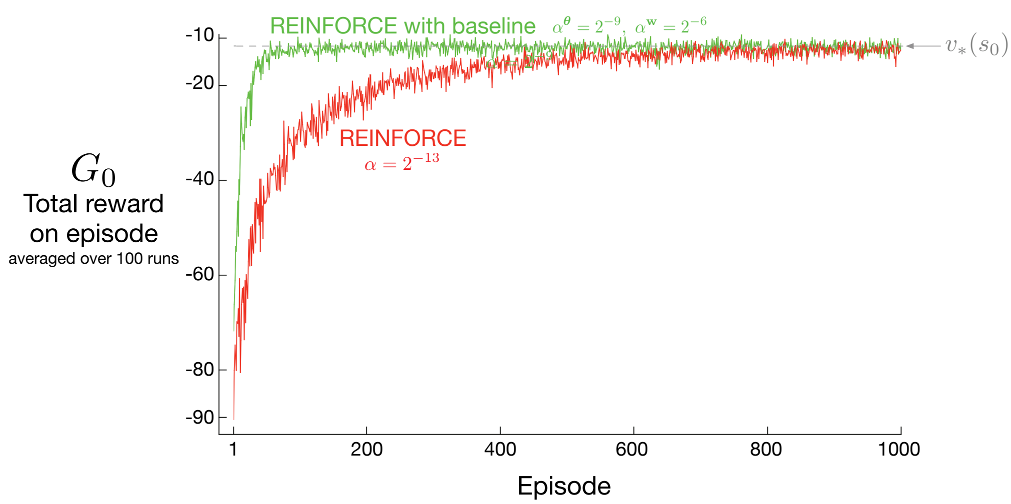

Figure 13.2: Adding a baseline to REINFORCE can make it learn much faster, as illustrated here on the short-corridor gridworld (Example 13.1). The step size used here for plain REINFORCE is that at which it performs best (to the nearest power of two; see Figure 13.1).