of both left and right actions. Note that if there were more than two possible actions, the neural network would have to output one probability per action so you would not need the concatenation step.

Okay, we now have a neural network policy that will take observations and output actions. But how do we train it?

Evaluating Actions: The Credit Assignment Problem

If we knew what the best action was at each step, we could train the neural network as usual, by minimizing the cross entropy between the estimated probability and the target probability. It would just be regular supervised learning. However, in Reinforcement Learning the only guidance the agent gets is through rewards, and rewards are typically sparse and delayed. For example, if the agent manages to balance the pole for 100 steps, how can it know which of the 100 actions it took were good, and which of them were bad? All it knows is that the pole fell after the last action, but surely this last action is not entirely responsible. This is called the *credit assignment problem*: when the agent gets a reward, it is hard for it to know which actions should get credited (or blamed) for it. Think of a dog that gets rewarded hours after it behaved well; will it understand what it is rewarded for?

To tackle this problem, a common strategy is to evaluate an action based on the sum of all the rewards that come after it, usually applying a *discount rate r* at each step. For example (see Figure 16-6), if an agent decides to go right three times in a row and gets +10 reward after the first step, 0 after the second step, and finally –50 after the third step, then assuming we use a discount rate $r = 0.8$, the first action will have a total score of $10 + r \times 0 + r^2 \times (-50) = -22$. If the discount rate is close to 0, then future rewards won't count for much compared to immediate rewards. Conversely, if the discount rate is close to 1, then rewards far into the future will count almost as much as immediate rewards. Typical discount rates are 0.95 or 0.99. With a discount rate of 0.95, rewards 13 steps into the future count roughly for half as much as immediate rewards (since $0.95^{13} \approx 0.5$), while with a discount rate of 0.99, rewards 69 steps into the future count for half as much as immediate rewards. In the CartPole environment, actions have fairly short-term effects, so choosing a discount rate of 0.95 seems reasonable.

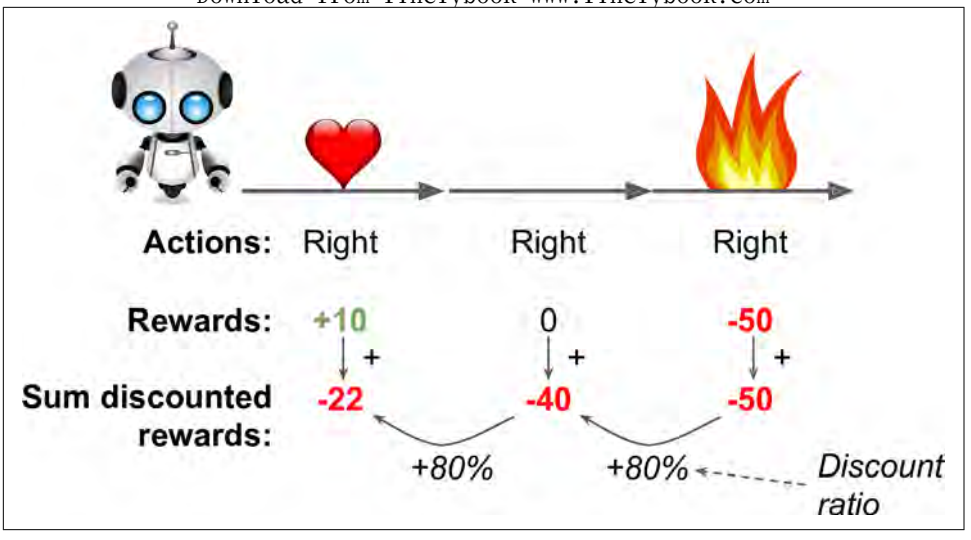

Figure 16-6. Discounted rewards

Of course, a good action may be followed by several bad actions that cause the pole to fall quickly, resulting in the good action getting a low score (similarly, a good actor may sometimes star in a terrible movie). However, if we play the game enough times, on average good actions will get a better score than bad ones. So, to get fairly reliable action scores, we must run many episodes and normalize all the action scores (by subtracting the mean and dividing by the standard deviation). After that, we can reasonably assume that actions with a negative score were bad while actions with a positive score were good. Perfect—now that we have a way to evaluate each action, we are ready to train our first agent using policy gradients. Let's see how.

Policy Gradients

As discussed earlier, PG algorithms optimize the parameters of a policy by following the gradients toward higher rewards. One popular class of PG algorithms, called *REINFORCE algorithms*, was introduced back in 1992[9] by Ronald Williams. Here is one common variant:

1. First, let the neural network policy play the game several times and at each step compute the gradients that would make the chosen action even more likely, but don't apply these gradients yet.

9 "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," R. Williams (1992).