On the flip side, of course, there is the problem that there may be no function in the chosen space of implicit representations that faithfully approximates the true utility function. As in all inductive learning, there is a trade-off between the size of the hypothesis space and the time it takes to learn the function. A larger hypothesis space increases the likelihood that a good approximation can be found, but also means that convergence is likely to be delayed.

Let us now consider exactly how the inductive learning problem should be formulated. We begin by considering how to learn utility and action-value functions, and then move on to learning the transition function for the environment.

In the LMS (least mean squares) approach, the formulation is straightforward. At the end of each training sequence, the LMS algorithm associates a reward-to-go with each state visited along the way. The $\langle state, reward \rangle$ pair can be used directly as a labelled example for any desired inductive learning algorithm. This yields a utility function $U(i)$.

It is also possible for a TD (temporal-difference) approach to apply inductive learning directly, once the U and/or Q tables have been replaced by implicit representations. The values that would be inserted into the tables by the update rules (20.2 and 20.7) can be used instead as labelled examples for a learning algorithm. The agent has to use the learned function on the next update, so the learning algorithm must be incremental.

One can also take advantage of the fact that the TD update rules provide small changes in the value of a given state. This is especially true if the function to be learned is characterized by a vector of weights $\mathbf{w}$ (as in linear weighted functions and neural networks). Rather than update a single tabulated value of U, as in Equation (20.2), we simply adjust the weights to try to reduce the temporal difference in U between successive states. Suppose that the parameterized utility function is $U_{\mathbf{w}}(i)$. Then after a transition $i \rightarrow j$, we apply the following update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[r + U_{\mathbf{w}}(j) - U_{\mathbf{w}}(i)]\nabla \mathbf{w} U_{\mathbf{w}}(i) \tag{20.8}$$

This form of updating performs gradient descent in weight space, trying to minimize the observed local error in the utility estimates. A similar update rule can be used for Q-learning (Exercise 20.9). Because the utility and action-value functions have real-valued outputs, neural networks and other continuous function representations are obvious candidates for the performance element. Decision-tree learning algorithms that provide real-valued output can also be used (see for example Quinlan's (1993b) **model trees**), but cannot use the gradient descent method.

MODEL TREES

The formulation of the inductive learning problem for constructing a model of the environment is also very straightforward. Each transition provides the agent with the next state (at least in an accessible environment), so that labelled examples consist of a state-action pair as input and a state as output. It is not so easy, however, to find a suitable implicit representation for the model. In order to be useful for value and policy iteration and for the generation of pseudo-experiences in TD learning, the output state description must be sufficiently detailed to allow prediction of outcomes several steps ahead. Simple parametric forms cannot usually sustain this kind of reasoning. Instead, it may be necessary to learn general action models in the logical form used in Chapters 7 and 11. In a nondeterministic environment, one can use the conditional-probability-table representation of state evolution typical of dynamic belief networks (Section 17.5), in which generalization is achieved by describing the state in terms of a large set of features and using only sparse connections. Although model-based approaches have advantages in terms of their ability to learn value functions quickly, they are currently hampered by a lack