the single scalar stored by an LSTM or GRU memory cell. There are two reasons to increase the size of the memory cell. One reason is that we have increased the cost of accessing a memory cell. We pay the computational cost of producing a coefficient for many cells, but we expect these coefficients to cluster around a small number of cells. By reading a vector value, rather than a scalar value, we can offset some of this cost. Another reason to use vector-valued memory cells is that they allow for **content-based addressing**, where the weight used to read to or write from a cell is a function of that cell. Vector-valued cells allow us to retrieve a complete vector-valued memory if we are able to produce a pattern that matches some but not all of its elements. This is analogous to the way that people can recall the lyrics of a song based on a few words. We can think of a content-based read instruction as saying, "Retrieve the lyrics of the song that has the chorus 'We all live in a yellow submarine.'" Content-based addressing is more useful when we make the objects to be retrieved large—if every letter of the song was stored in a separate memory cell, we would not be able to find them this way. By comparison, **location-based addressing** is not allowed to refer to the content of the memory. We can think of a location-based read instruction as saying "Retrieve the lyrics of the song in slot 347." Location-based addressing can often be a perfectly sensible mechanism even when the memory cells are small.

If the content of a memory cell is copied (not forgotten) at most time steps, then the information it contains can be propagated forward in time and the gradients propagated backward in time without either vanishing or exploding.

The explicit memory approach is illustrated in figure 10.18, where we see that a "task neural network" is coupled with a memory. Although that task neural network could be feedforward or recurrent, the overall system is a recurrent network. The task network can choose to read from or write to specific memory addresses. Explicit memory seems to allow models to learn tasks that ordinary RNNs or LSTM RNNs cannot learn. One reason for this advantage may be because information and gradients can be propagated (forward in time or backwards in time, respectively) for very long durations.

As an alternative to back-propagation through weighted averages of memory cells, we can interpret the memory addressing coefficients as probabilities and stochastically read just one cell (Zaremba and Sutskever, 2015). Optimizing models that make discrete decisions requires specialized optimization algorithms, described in section 20.9.1. So far, training these stochastic architectures that make discrete decisions remains harder than training deterministic algorithms that make soft decisions.

Whether it is soft (allowing back-propagation) or stochastic and hard, the