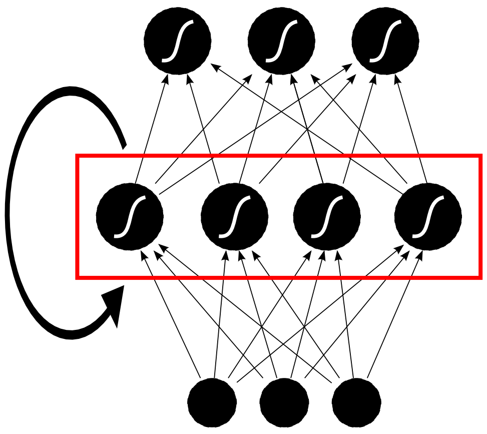

Figure 3.3: **Recurrent neural network**

3.2.1 Forward Pass

The forward pass of an RNN is the same as that of an MLP with a single hidden layer, except that activations arrive at the hidden layer from both the current external input and the hidden layer activations one step back in time. Consider a length T input sequence $\mathbf{x}$ presented to an RNN with I input units, H hidden units, and K output units. Let x_i^t be the value of input i at time t, and let a_j^t and b_j^t be respectively the network input to unit j at time t and the activation of unit j at time t. For the hidden units we have

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1} \tag{3.29}$$

Nonlinear, differentiable activation functions are then applied exactly as for MLPs

$$b_h^t = \theta_h(a_h^t) \tag{3.30}$$

The complete sequence of hidden activations can be calculated by starting at $t = 1$ and recursively applying (3.29) and (3.30), incrementing t at each step. Note that this requires initial values b_i^0 to be chosen for the hidden units, corresponding to the network's state before it receives any information from the data sequence. In this thesis, b_i^0 is always set to zero. However, other researchers have found that in some cases, RNN stability and robustness to noise can be improved by using nonzero initial values (Zimmermann et al., 2006a).

The network inputs to the output units can be calculated at the same time as the hidden activations:

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t \tag{3.31}$$

For sequence classification and segment classification tasks (see Section 2.3) the same output activation functions can be used for RNNs as for MLPs (i.e. softmax and logistic sigmoid), with the classification targets typically presented at the ends of the sequences or segments. It follows that the same objective functions can be used too. Chapter 7 introduces an output layer specifically designed for temporal classification with RNNs.

3.2.2 Backward Pass

Given the partial derivatives of the objective function with respect to the network outputs, we now need the derivatives with respect to the weights. Two well-known algorithms have been devised to efficiently calculate weight derivatives for RNNs: real time recurrent learning (RTRL; Robinson and Fallside, 1987) and backpropagation through time (BPTT; Williams and Zipser, 1995; Werbos, 1990). We focus on BPTT since it is both conceptually simpler and more efficient in computation time (though not in memory).

Like standard backpropagation, BPTT consists of a repeated application of the chain rule. The subtlety is that, for recurrent networks, the objective function depends on the activation of the hidden layer not only through its influence on the output layer, but also through its influence on the hidden layer at the next timestep, i.e.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right), \tag{3.32}$$

where

$$\delta_j^t \overset{\text{def}}{=} \frac{\partial O}{\partial a_j^t} \tag{3.33}$$

The complete sequence of δ terms can be calculated by starting at $t = T$ and recursively applying (3.32), decrementing t at each step. (Note that $\delta_j^{T+1} = 0 \,\forall j$, since no error is received from beyond the end of the sequence). Finally, bearing in mind that the weights to and from each unit in the hidden layer are the same at every timestep, we sum over the whole sequence to get the derivatives with respect to each of the network weights

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t}\frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t \tag{3.34}$$

3.2.3 Bidirectional RNNs

For many sequence labelling tasks, we would like to have access to future as well as past context. For example, when classifying a particular written letter, it is helpful to know the letters coming after it as well as those before. However, since standard RNNs process sequences in temporal order, they