Now we just have to see that this is linear in the inputs:

$$O_i = c^2 \sum_k I_k \sum_j W_{k,j} W_{j,i} + d\left(1 + c \sum_j W_{j,i}\right)$$

Thus we can compute the same function as the two-layer network using just a one-layer perceptron that has weights $W_{k,i} = \sum_j W_{k,j} W_{j,i}$ and an activation function $g(x) = c^2 x + d\left(1 + c \sum_j W_{j,i}\right)$.

**b.** The above reduction can be used straightforwardly to reduce an $n$-layer network to an $(n-1)$-layer network. By induction, the $n$-layer network can be reduced to a single-layer network. Thus, linear activation function restrict neural networks to represent only linearly functions.

**20.18** The implementation of neural networks can be approached in several different ways. The simplest is probably to store the weights in an $n \times n$ array. Everything can be calculated as if all the nodes were in each layer, with the zero weights ensuring that only appropriate changes are made as each layer is processed.

Particularly for sparse networks, it can be more efficient to use a pointer-based implementation, with each node represented by a data structure that contains pointers to its successors (for evaluation) and its predecessors (for backpropagation). Weights $W_{j,i}$ are attached to node $i$. In both types of implementation, it is convenient to store the summed input $in_i = \sum_j W_{j,i} a_j$ and the value $g'(in_i)$. The code repository contains an implementation of the pointer-based variety. See the file `learning/algorithms/nn.lisp`, and the function `nn-learning` in that file.

**20.19** This question is especially important for students who are not expected to implement or use a neural network system. Together with 20.15 and 20.17, it gives the student a concrete (if slender) grasp of what the network actually does. Many other similar questions can be devised.

Intuitively, the data suggest that a probabilistic prediction $P(Output = 1) = 0.8$ is appropriate. The network will adjust its weights to minimize the error function. The error is

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2 = \frac{1}{2}[80(1 - a_1)^2 + 20(0 - a_1)^2] = 50O_1^2 - 80O_1 + 50$$

The derivative of the error with respect to the single output $a_1$ is

$$\frac{\partial E}{\partial a_1} = 100a_1 - 80$$

Setting the derivative to zero, we find that indeed $a_1 = 0.8$. The student should spot the connection to Ex. 18.8.

**20.20** The application of cross-validation is straightforward—the methodology is the same as that for any parameter selection problem. With 10-fold cross-validation, for example, each size of hidden layer is evaluated by training on 90% subsets and testing on the remaining 10%. The best-performing size is then chosen, trained on all the training data, and the result is returned as the system's hypothesis. The purpose of this exercise is to have the student