Neural Networks continued

Backpropagation

Applying all of the training patterns to the network is called an **epoch**. There are three algorithms for an epoch of training

1. On-line:
   For each training pattern (input),
   - Apply the training pattern as discussed on Wed (i.e. compute $\Delta w_{ji}$ for all edges)
   - Apply the weight changes to the network
   - Repeat

2. Stochastic: This is the same as on-line, but randomly permutes the order in which the training patterns are applied.

3. Batch:
   For each training pattern (input),
   - Apply the training pattern as discussed on Wed (i.e. compute $\Delta w_{ji}$ for all edges).
   - Store the weight change.

   Apply the weight changes to the network. These weight changes may be the sum of all the weight changes computed in the above loop, or the mean, or the median.

Key factors in ANN design

- Architecture complexity: big enough to learn the task, small enough to train and run

Since the input and output layers are generally determined by the task, the complexity of the ANN is determined by the number and structure of the hidden nodes. Some networks reduce complexity (the number of weights) by not fully connecting the layers. On a fully connected network, reducing or increasing the complexity of the network means adding or deleting hidden nodes.
An empirical method of network design is to start with a small number of hidden nodes and add nodes until overtraining is possible: when the training and test set errors begin to diverge after sufficient training. Alternatively, if the initial guess at the number of hidden nodes enables overtraining, then reduce the number of hidden nodes until the network’s performance begins to degrade.

- Creating adequate training and testing sets: both quantity and coverage of likely inputs

ANNs learn based on their training data. They are good at learning a function in areas where there is lots of training data, and they can generalize to nearby parts of the input space. Inputs that are very different from any of their training examples will produce an output, but the output is not likely to be meaningful. To guard against outputs that are meaningless, the network either needs training examples from all parts of the input space or it needs to produce some kind of confidence measure that indicates whether it knows how to process the input.

Note: Stephanie found the Wikipedia page on over fitting to be helpful.

- Building in expert knowledge

If you know something about the structure of the problem or relationships in the input, then you can sometimes build that knowledge into the structure of the network. A hidden node in a network works similarly to a compression method like PCA. The training process forces it to learn the set of connection weights that minimize the total error of the network. In doing so, it is compressing a set of values—the inputs—to a single value.

If you have prior knowledge that a set of inputs are strongly related, it is worth considering a network architecture where only those inputs connect to one or more hidden nodes in a pipeline separate from other inputs. The output of the specialized hidden node connects with the rest of the network at a higher layer. This architecture forces the network to learn the relationships between the limited set of inputs before learning the group’s relationship to other inputs. If the prior knowledge is correct, then the network may train faster and work better than a more generic network trained on the same data.

Given an infinite amount of data, a generic architecture can learn as well or better than a specialized architecture. However, given a specific data set, researchers have shown that a modular, specialized network will generally perform better than a generic architecture.

- Confidence measures

Careful design of the network outputs enables the network to provide an indication as to whether its output is meaningful. For example, when dealing with a two class problem, it is possible to design the network with a single output node, using a low value to indicate one class and a high value to indicate the other. A single output, however, gives no indication of the confidence the network has in the value.

An alternative is to design the network with two outputs, one for each class, and train it to make only one of the outputs high. In that situation, two high outputs or two low outputs indicates that the network is not well trained for that input pattern. While it is also possible for the network to give a meaningless 1/0 or 0/1 output, having the two outputs makes that less likely.

Note: These notes are adapted from those of Bruce Maxwell.