N E U R A L N E T W O R K T Y P E S
(for CS 773b, Spring 2008)

1. Standard BP NNs (also known as MLPs for multiple layered perceptron, and sometimes called feedforward NNs). They use *steepest descent* on the weights from the hidden layer to the output layer (these should be linear functions with the weights as the coefficients), and also on the weights from the input layer to the hidden layer (summed and the sums put through the logistic, or "S" shaped curve). These take a lot of training to find a good local minimum for the weight set, but once it is found, they are fast on-line recognizers of unknown input vectors. Hornik's theorem states that 3 layers are sufficient for any learning.

2. Radial Basis Function NNs (called RBFNNs). These use Gaussian functions centered on vectors in the feature space (it is called radial because it has equal radii in all directions from the center). No weights are used from the input (fan out) nodes to the hidden layer nodes. The weights from the hidden nodes to the output node(s) provide linear combinations at the output and so the steepest descent training converges to the single global minimum. This makes training very quick and relatively accurate.

3. Probabilistic NNs (PNNs). This type uses sums of Gaussian (normal) functions to form probability distribution functions (pdf) centered on vectors in the feature space. The pdf with the highest value is the winner. The assumption that the pdf's are of this form usually holds.

4. Fuzzy NNs with Trainable Weights. This is a type of BP NN that has fuzzy inputs, or fuzzy weights, or both. It can be trained by steepest descent.

5. Fuzzy Rule NNs. The values at the input layer go to the second layer where they are fuzzified by fuzzy set membership functions (e.g., x(1) is HIGH with fuzzy truth f(1)). The second layer fuzzy variables then connect to one or more of the antecedents from at the third layer of antecedent nodes to provide them with fuzzy truth. One or more antecedent nodes at the third layer connect to the appropriate consequent in the fourth layer, which forms a rule (there are not complete connections, but only those connections determined by the rule implications from antecedent nodes to consequent nodes). The consequents are defuzzified at the output, or fifth layer to form the adjusted (weighted by fuzzy values) output.

6. Simple Fuzzy NNs. These are just like the PNNs except that the Gaussians are not summed to form a probability density function. Instead, each Gaussian is centered on a feature vector whose class is known. It is set up on the exemplar training vectors that are labeled. When an unknown feature vector is presented to the system, it is put through each Gaussian to get the fuzzy truth that it belongs to the same class as that Gaussian center. The maximum fuzzy truth is the winner and determines the class.

7. Radial Basis Functional Link Nets (RBFLNs). Here we start with an RBFNN and add lines with weights from the inputs layer to the output layer, bypassing the center layer. At the output layer, the results coming in from the input layer, and from the hidden layer, are summed in the error function and used in a steepest descent algorithm to determine both sets of weights. It learns very quickly, but is slightly slower in on-line recognition that RBFNNs.

8. Hopfield (Recurrent) NNs. These are set up on a set of final feature vectors (that the NN will learn to output) by computing the weights as outer products of these vectors. Then an unknown feature vector is input, whereupon it goes through the network and the outputs are fed back as inputs. After a number of iterations, the outputs stabilize to a fixed feature vector that is the result that associates with the input. The dimension of the vectors must be large in comparison with the number of classes (about 8 times as many features as classes to be safe). These are rarely useful in applications, but hold great promise if more research leads to useful strategies and multiple combinations of such networks or combinations with other NNs.

9. Competitive Learning Algorithms. Given a set of feature vectors for training, a set of prototypes is drawn randomly to be in the feature space. One feature vector at a time is presented to these prototypes, and the prototype that is nearest to it is moved a step toward that feature vector. The other prototypes may be move a step away from it, or may not move at all (different strategies are used). While this is not a NN as such, it is a process known as Learning Vector Quantization and is used in a the Kohonen networks. We will allow that this is a part of a NN algorithm whenever the prototypes are learned and then used in a NN, such as centers for pdf's or Gaussian fuzzy set membership functions.

10. Combinations/Mixes with Other Algorithms. There are many different combinations of NNs with other NNs and with non-NN algorithms. For example, there is a Fuzzy Support Vector NN that has very good results. There are algorithms where a clustering algorithm is used to cluster unknown feature vectors in classes, class centers found, and radial basis functions centered on these centers in an RBFNN.

11. Support Vector Machines. This is not a NN in the strict sense, but it is a supervised learning system that allows labeled feature vectors to be separated, so we include it here. It is an algorithm that maps feature vectors from N-dimensions into a higher M-dimensions, so it is easier to separate them with hyperplanes in the higher dimensional space (that would be unknown hyperspheres in the original feature space). It is especially good for 2 classes. A pair of vectors with one in Class 1 and the other in Class 2 is found that are the closest from the 2 classes. Then another closest pair is found. A hyperplane is passed through the two vectors in Class 1 and another hyperplane is passed through the other two in Class 2. These are the support vectors. These hyperplanes provide a separating margin between the two classes and are used to classify unknown feature vectors on line.