Neural Network Models

Neural network models are just complicated nonlinear statistical models. Like projection pursuit models, they make use of derived variables which correspond to directions in the predictor space, and a sum (for regression) or sums (for classification) of nonlinear transformations of these derived variables is/are used to obtain the predictions.

I will first describe a single hidden layer regression model based on input variables $x_1, x_2, \ldots, x_{p-1}$, and x_p. For the "hidden layer," M linear combinations

of the p predictors are obtained, and all of these linear combinations are transformed using the same nonlinear "activation" function, σ, to obtain M variables to use for the next step. So the original inputs are used to obtain

$$z_m = \sigma(\alpha_{0m} + \alpha_{1m} x_1 + \cdots + \alpha_{pm} x_p) \quad (m = 1, \ldots, M).$$

Then the predicted values are obtained using

$$\hat{y} = \beta_0 + \beta_1 z_1 + \cdots + \beta_M z_M.$$

So it's similar to PPR, except that a fixed f'n, σ, is used to nonlinearly transform the linear combinations of the inputs, instead of using smoothers to obtain the nonlinear transformations (as PPR does). Since ridge f'ns used in PPR can have greater complexity than σ, many more terms are generally needed w/ neural nets.

The α_{ij} and the β_j are fit iteratively using the training data. (See Sec. 11.4 and Sec. 11.5 of HTF for some details.) If M is large, which means that there are a lot of parameters, one needs to worry about overfitting — if too many iterations are used in the fitting process, the $\hat{y}_i$ can be very close to the observed y_i, but the fitted model can perform poorly with regard to generalization errors. One approach is to use a validation set, and stop the iterations when the validation error starts to increase (even though the training error may still be decreasing). Note that unlike PPR, with which terms are added to the model

one by one, and estimates of the generalization error are used to control the complexity of the model and avoid overfitting; with neural networks, M is generally set to be a large number before the training process begins, and overfitting is avoided by "early stopping" — stopping the iterations before the errors on the training data are collectively minimized.

A popular choice for the activation function is the sigmoid function, for which

$$\sigma(v) = \frac{1}{1 + e^{-v}}.$$

To better understand how the sigmoid fn can nonlinearly transform the linear combinations of

the inputs, I think that is good to view

$$\sigma(\alpha_0 + \alpha_1 x_1 + \cdots + \alpha_p x_p)$$

as

$$\sigma(\alpha + \gamma \vec{\omega}^T \vec{x}),$$

where $\alpha = \alpha_0$, $\vec{\omega}$ is a unit vector having the same direction as $(\alpha_1, \alpha_2, \ldots, \alpha_p)^T$, and γ is the length of $(\alpha_1, \ldots, \alpha_p)^T$. For some combinations of α and γ, the

$$v_i = \alpha + \gamma \vec{\omega}^T \vec{x}_i$$

values may all be in the interval $(-1, 1)$, on which

$$\sigma(v) = (1 + e^{-v})^{-1}$$

is approximately linear. For other choices of α and γ, the v_i values can be in an interval on which $\sigma(v)$ is monotonically increasing with an

increasing derivative, or the v_i values can belong to an interval on which $\sigma(v)$ is monotonically increasing with a decreasing derivative. (Note: To get decreasing behavior, the sigmoid can be multiplied by a negative coefficient.) For still other choices of α and γ, $\sigma(v)$ can produce roughly a binary response, with the values of the $\sigma(v_i)$ being either real close to 0 or real close to 1. (To get this dichotomous behavior, γ needs to be suitably large. To get linear behavior, γ needs to be suitably small. In both of these cases, α needs to be a suitable value. For linear behavior, α should be such that the middle of the $v_i - \alpha$ values is close to 0. For the dichotomous behavior, changing α will change the "threshold.")

For K-class classification, there are K target variables — for each case, exactly one of y_1, $y_2, \ldots, y_{K-1}$, and y_K is coded with a 1, and the rest of the target variable values are 0. (For K=2, I think one could just use a single binary target.) One forms K different linear combinations from the "hidden units" (the z_m $(m=1,\ldots,M)$). From the

$$t_k = \beta_{0k} + \beta_{1k} z_1 + \cdots + \beta_{Mk} z_k \quad (k=1,\ldots,K),$$

the

$$g_k(t_1,\ldots,t_K) = e^{t_k} / \sum_{\ell=1}^{K} e^{t_\ell}$$

are obtained. For each observation, the predicted class is

$$\arg\max_k \; g_k(t_1,\ldots,t_K).$$

It's better to have too many hidden units than too few. With too few, there may not be enough flexibility to capture the nonlinearity.

One can use a neural network model having more than one hidden layer, and the hidden units in each layer can be connected in different ways — it's not necessary to have them fully connected. Sec. 11.7 of HTF describes five different neural network models for the same setting. Three models have two hidden layers, with units connected in different ways, one model has one hidden layer, and one model doesn't have any hidden layers (and is equivalent to a multinomial logistic regression model). The architecture used can make an appreciable difference.