Neural Networks

Thomas Lonon

Financial Engineering
Stevens Institute of Technology

November 2, 2017

Assume we have an input vector X with p components, and a target Y (standard supervised learning setup). Let $\omega_m, m = 1, 2, \ldots, M$ be unit p-vectors of unknown parameters. The projection pursuit regression (PPR) model has the form

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X)$$

Note that this is an additive model of the derived features ($V_m = \omega_m^T X$). The functions g are estimated along with the directions ω_m.

The function $g_m(\omega_m^T X)$ is called a *ridge function* in $\mathbb{R}^p$.

The scalar variable $V_m = \omega_m^T X$ is the projection of X onto unit vector ω_m.

We seek these ω_m's so that the model fits well and hence the name projection pursuit regression.

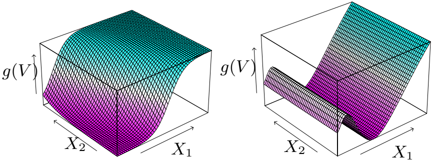


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1 + \exp(-5(V - 0.5))]$, *where* $V = (X_1 + X_2)/\sqrt{2}$.
(Right:) $g(V) = (V + 0.1)\sin(1/(V/3 + 0.1))$, *where* $V = X_1$.

If M is taken to be arbitrarily large, for appropriate choices of g_m, the PPR model can approximate any continuous function in $\mathbb{R}^p$. Such a class of models is called a *universal approximator*.

The PPR model is most useful for prediction and not for producing an understandable model for the data

The $M = 1$ model is the exception, it is known as the *single index model* in econometrics.

To fit the PPR model, we seek approximate minimizers of the error function

$$\sum_{i=1}^{N}\left[y_i - \sum_{m=1}^{M} g_m(\omega_m^T x_i)\right]^2$$

over functions g_m and direction vectors ω_m.

PPR Implementation Details

- Although any smoothing method can in principle be used, it is convenient if the method provides derivatives. Local regression and smoothing splines are convenient
- After each step the g_m's from previous steps can be readjusted using the backfitting procedure described in Chapter 9. While this may lead ultimately to fewer terms, it is not clear whether it improves prediction performance.
- Usually the ω_m are not readjusted (partly to avoid excessive computation), although in principle the could be as well
- The number of terms M is usually estimated as part of the forward stage-wise strategy. The model building stops when the next term does not appreciably improve the fit of the model. Cross-validation can also be used to determine M.

[1]

Neural Networks

A neural network is a two-stage regression or classification model, typically represented by a network diagram.

For regression, typically $K = 1$ and there is only one output at the top (Y_1)

For K-class classification, there are K units at the top, with the k^{th} unit modeling the probability of class k.

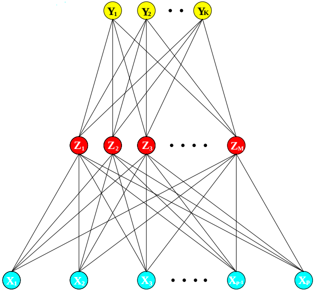


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

Features Z_m are created from linear combinations of the inputs and the target Y_k is modeled as a function of linear combinations of the Z_m.

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \ldots, M \\ T_k &= \beta_{0k} + \beta_k^T Z, k = 1, \ldots, K \\ f_k(X) &= g_k(T), k = 1, \ldots, K \end{aligned}$$

where $Z = (Z_1, Z_2, \ldots, Z_M)$ and $T = (T_1, T_2, \ldots, T_K)$[1]

The activation function $\sigma(\nu)$ is usually chosen to be the *sigmoid*

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

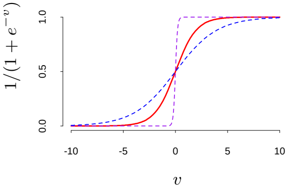


FIGURE 11.3. *Plot of the sigmoid function $\sigma(v) = 1/(1+\exp(-v))$ (red curve), commonly used in the hidden layer of a neural network. Included are $\sigma(sv)$ for $s = \frac{1}{2}$ (blue curve) and $s = 10$ (purple curve). The scale parameter s controls the activation rate, and we can see that large s amounts to a hard activation at $v = 0$. Note that $\sigma(s(v - v_0))$ shifts the activation threshold from 0 to v_0.*

The output function $g_k(T)$ does a final transformation of the vector T. For regression this is typically $g_k(T) = T_k$.

For K-classification we use the *softmax* function

$$g_k(T) = \frac{e^{T_k}}{\sum_{\ell=1}^{K} e^{T_\ell}}$$

These Z_m are hidden units that are expressed as a basis expansion of the original inputs X.

The neural network with one hidden layer has exactly the same form as the PPR model.

$$
\begin{aligned}
g_m(\omega_m^T X) &= \beta_m \sigma(\alpha_{0m} + \alpha_m^T X) \\
&= \beta_m \sigma(\alpha_{0m} + \|\alpha_m\| (\omega_m^T X))
\end{aligned}
$$

where

$$\omega_m = \frac{\alpha_m}{\|\alpha_m\|}$$

is the m^{th} unit-vector

Fitting Neural Networks

In the neural network, we have unknown parameters which we denote *weights*. We label the set of these weights θ which consist of:

$$\{\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M\} M(p+1) \text{weights}$$
$$\{\beta_{0k}, \beta_k; k = 1, 2, \ldots, K\} K(M+1) \text{weights}$$

For regression, we use sum-of-squared errors as our measure of fit

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

For classification we use either squared error or cross-entropy

$$R(\theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

and the corresponding classifier is $G(x) = \arg\max_k f_k(x)$[1]

We will minimize this $R(\theta)$ through gradient descent, called *back-propagation*. The steps involved for the squared error loss are as follows:

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ with $z_i = (z_{1i}, z_{21}, \ldots, z_{Mi})$. Then:

$$
\begin{aligned}
R(\theta) &= \sum_{i=1}^{N} R_i \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2
\end{aligned}
$$

This has derivatives:

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}$$

so the gradient descent update at $r+1$ has the form:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(} r)}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}$$

where γ_r is the *learning rate*

We can now write the partials as:

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$$
$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = s_{mi} x_{i\ell}$$

These quantities δ_{ki} and s_{mi} are "errors" from the current model which satisify:

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

which are known as the *back-propagation equations*

Issues in Training Neural Networks

Starting Values

Overfitting a method to avoid this is *weight decay* which is analogous to ridge regression for linear models. We add a penalty to the error function $R(\theta) + \lambda J(\theta)$ where

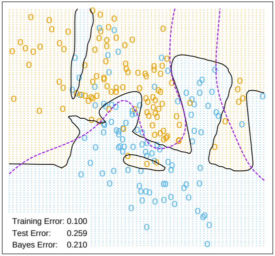
$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{m\ell} \alpha_{m\ell}^2$$

and $\lambda \geq 0$ is a tuning parameter. Or we could express the penalty as

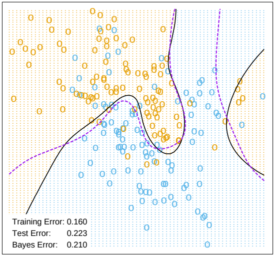
$$J(\theta) = \sum_{km} \frac{\beta_{km}^2}{1 + \beta_{km}^2} + \sum_{m\ell} \frac{\alpha_{m\ell}^2}{1 + \alpha_{m\ell}^2}$$

known as *weight elimination* penalty

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02



Issues Cont.

Scaling of the Inputs

Number of Hidden Units and Layers

Multiple Minima

Given training data $\mathbf{X}_{tr}, \mathbf{y}_{tr}$, we assume a sampling model with parameters θ. Given a prior distribution $\mathbb{P}(\theta)$, the posterior distribution of the parameters is

$$\mathbb{P}(\theta|\mathbf{X}_{tr}, \mathbf{y}_{tr}) = \frac{\mathbb{P}(\theta)\mathbb{P}(\mathbf{y}_{tr}|\mathbf{X}_{tr}, \theta)}{\int \mathbb{P}(\theta)\mathbb{P}(\mathbf{y}_{tr}|\mathbf{X}_{tr}, \theta)d\theta}$$

and for a test case with feature X_{new}, the predictive distribution for Y_{new} is

$$\mathbb{P}(Y_{new}|X_{new}, \mathbf{X}_{tr}, \mathbf{y}_{tr}) = \int \mathbb{P}(Y_{new}|X_{new}, \theta)\mathbb{P}(\theta|\mathbf{X}_{tr}, \mathbf{y}_{tr})d\theta$$

We can write all of the models in the form:

$$\hat{f}(\mathbf{x}_{new}) = \sum_{\ell=1}^{L} w_\ell \mathbb{E}[Y_{new} | \mathbf{x}_{new}, \hat{\theta}_\ell]$$

In each:

- Bayesian model: $w_\ell = 1/L$, the average estimates the posterior mean by sampling θ_ℓ from the posterior distribution
- Bagging: $w_\ell = 1$, $\hat{\theta}_\ell$ are the parameters refit to bootstrap re-samples of the training data
- Boosting: $w_\ell = 1$, $\hat{\theta}_\ell$ are typically chosen in nonrandom sequential fashion to constantly improve the fit

[1]

[1] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Stastical Learning: Data Mining, Inference, and Prediction*. Number v.2 in Springer Series in Statistics. Springer, 2009.