$\boxed{\text{Recall}}$ Features $\underline{x}$ (p-variate) + response Y (let's say continuous), neural network model is a model for $f(\underline{x})$ in

$$Y = f(\underline{x}) + \varepsilon$$

where

$$f(\underline{x}) = \beta_0 + \sum_{k=1}^{K_L} \beta_k A_k^{(L)}$$

$$A_k^{(L)} = g\left(w_{k0}^{(L)} + \sum_{j=1}^{K_{L-1}} w_{kj}^{(L)} A_j^{(L-1)}\right) \qquad k = 1, \ldots, K_L$$

$\vdots$

$$A_k^{(1)} = g\left(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} x_j\right) \qquad k = 1, \ldots, K_1$$

where L hidden layers, the kth of which has K_k activations

using the activation function g (usually ReLU).

Note If we set $\underline{x} = (1, x_1, \ldots, x_p)^T$ to include the intercept, then we can write

$$A_0^{(0)} = 1 \quad (= \text{bias holder for next level})$$

$$A_1^{(1)} = g\left(w_{10}^{(1)} + \sum_{j=1}^{p} w_{1j}^{(1)} x_j\right) \Rightarrow g\left(\overbrace{W^{(1)} \underline{x}}^{(K_1+1)\times 1}\right)$$

apply to each element

$(p+1) \times 1$ has a 1 in 1st entry

$(1+K_1) \times (p+1)$

$$A_2^{(1)} = g\left(w_{20}^{(1)} + \sum_{j=1}^{p} w_{2j}^{(1)} x_j\right)$$

$$\vdots$$

$$A_{K_1}^{(1)} = g\left(w_{K_1 0}^{(1)} + \sum_{j=1}^{p} w_{K_1 j}^{(1)} x_j\right)$$

$$W^{(1)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ w_{10}^{(1)} & w_{11}^{(1)} & \cdots & w_{1p}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & \cdots & w_{2p}^{(1)} \\ \vdots & & & \\ w_{K,0}^{(1)} & w_{K,1}^{(1)} & \cdots & w_{K,p}^{(1)} \end{pmatrix}$$

10.3 Estimation

Estimation can be difficult due to the # of parameters in a NN.

One approach: minimize

$$\sum_{i=1}^{n} \left(y_i - f(\underline{x}_i)\right)^2$$

problem: nonconvex, not guaranteed to have unique soln.

$$R(\underline{\theta}) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f_{\underline{\theta}}(\underline{x}_i)\right)^2$$

where $\underline{\theta}$ = vector of (β's + all w's)

In practice we use a gradient descent algorithm to estimate $\underline{\theta}$:

① Start with a guess for $\underline{\theta}^{(0)}$ + set $t=0$.
[start at rid N's]

② Iterate until MSE doesn't improve:

ⓐ Find vector $\underline{\delta}$ reflecting a small change in $\underline{\theta}$ such that $\underline{\theta}^{t+1} = \underline{\theta}^{t} + \underline{\delta}$ and $R(\underline{\theta}^{t+1}) < R(\underline{\theta}^{t})$

ⓑ set $t \leftarrow t+1$

The $\underline{\delta}$ vector is just a scaled gradient:

$$\underline{\theta}^{t+1} = \underline{\theta}^{t} - \rho \nabla R(\underline{\theta}^{t})$$

where $\nabla R(\underline{\theta}^{t})$ is gradient of R evaluated at current value, and $1 > \epsilon > 0$ is the learning rate, and is typically small.

$\boxed{\text{Note}}$ The gradients for NNs end up being straightforward.

Example: single layer feed forward network

$$R_i(\underline{\theta}) = \frac{1}{2}\left(y_i - f_{\underline{\theta}}(\underline{x}_i)\right)^2$$

$$= \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k\, g\Big(\underbrace{w_{k0} + \sum_{j=1}^{p} w_{kj}\, x_{ij}}_{z_{ik}}\Big)\right)^2$$

$$= \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k\, g(z_{ik})\right)^2$$

$$\frac{\partial R_i(\underline{\theta})}{\partial \beta_k} = \frac{\partial R_i(\underline{\theta})}{\partial f_{\underline{\theta}}(\underline{x}_i)} \; \frac{\partial f_{\underline{\theta}}(\underline{x}_i)}{\partial \beta_k}$$

$$= -(y_i - f_{\underline{\theta}}(\underline{x}_i)) \cdot g(z_{ik})$$

$$\frac{\partial R_i(\underline{\theta})}{\partial w_{kj}} = \frac{\partial R_i(\underline{\theta})}{\partial f_{\underline{\theta}}(\underline{x}_i)} \frac{\partial f_{\underline{\theta}}(\underline{x}_i)}{\partial g(z_{ik})} \frac{\partial g(z_{ik})}{\partial (z_{ik})} \frac{\partial z_{ik}}{\partial w_{kj}}$$

$$= -(y_i - f_{\underline{\theta}}(\underline{x}_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}$$

Each deriv gets the residual $(y_i - f_i)$, which is called backpropagation.

Need some additional tricks b/c $\underline{\theta}$ is high dimensional

- For sample is size n, a smaller random subset of data is used in each gradient step, "mini batch", resulting in <u>stochastic gradient descent</u>.

- Regularization is often done using lasso or ridge-like penalty.
- Dropout learning uses a subset of activation nodes at each step, allowing us to focus on only a subset of weights.