Backpropagation

The treatment here is taken mainly from

Efron, B. and T. Hastie (2016). *Computer Age Statistical Inference*, Cambridge University Press, pp 355-357.

Consider a deep artificial neural network, also known as a multilayer perceptron. The number of layers is K, and layers are indexed by $k = 1, 2, \ldots, K$. Thus the input layer corresponds to $k = 1$, and the output layer to $k = K$. Instances have p features, $x_1, \ldots, x_p$, and, for each instance that is fed to the network, the input layer gets p nodes containing the features, x_j, $j = 1, \ldots, p$.

Hidden layers are indexed by $k = 2, \ldots, K-1$. The hidden layer k contains p_k units (or nodes, or neurons). Units ℓ in layer k receive inputs a_j^{k-1}, $j = 1, \ldots, p_{k-1}$ from the units in the layer underneath. For $k = 2$, the inputs are $a_j^1 = x_j$. The inputs are then subjected to an affine transformation, which, for hidden layer k, can be written in vector-matrix notation as follows:

$$\boldsymbol{z}^k = \boldsymbol{W}^{k-1}\boldsymbol{a}^{k-1},$$

where $\boldsymbol{z}^k$ is a p_k-vector, $\boldsymbol{W}^{k-1}$ is a $p_k \times p_{k-1}$ matrix of weights and biases, and $\boldsymbol{a}^{k-1}$ is a p_{k-1} vector of inputs. If layer k is not dense, or fully connected, then some elements of $\boldsymbol{W}^{k-1}$ are equal to zero.

The vector $\boldsymbol{z}^k$ is now transformed by a nonlinear activation function g^k which acts element-wise on the vector $\boldsymbol{z}^k$ in order to generate the p_k-vector $\boldsymbol{a}^k$ of activations that are input into layer $k+1$. We may write $\boldsymbol{a}^k = g^k(\boldsymbol{z}^k)$ is slightly unconventional notation.

When we get to the output layer, the inputs from layer $K-1$ are handled in a way that depends on what the task is. For instance, if the network is a classifier of M classes, the number of nodes in this layer would be $p_K = M$. Input would first be subjected to an affine transformation as usual, with matrix $\boldsymbol{W}^{K-1}$. Activation might then generate an M-vector of probabilities using the softmax function. This would look like

$$a_m^K = \frac{\exp(z_m^K)}{\sum_{j=1}^M \exp(z_j^K)},$$

where the z_j^K are the components of the vector $\boldsymbol{z}^K$. For a regression task, the activation might well be just the identity function.

Next comes the computation of the contribution to the loss function from this instance. Denote this by $L(\boldsymbol{a}^K)$. The above describes a forward pass. The entire set of matrices $\boldsymbol{W}^k$ is present throughout in memory, and node ℓ in layer k saves its inputs z_ℓ^k and its activation a_ℓ^k, $\ell = 1, \ldots, p_k$.

Backpropagation is now used in order to compute the partial derivatives of the contribution $L(\boldsymbol{a}^k)$ from this instance with respect to all the elements of all the matrices $\boldsymbol{W}^k$, $k = 1, \ldots, K$.

1. For unit ℓ in the output layer, $\ell = 1, \ldots, p_k$, compute a (partial) derivative:

$$\delta_\ell^K = \frac{\partial L}{\partial z_\ell^K} = \frac{\partial L}{\partial a_\ell^K} \, \dot{g}^K(z_\ell^K),$$

   where the dot denotes differentiation of a scalar function with respect to its scalar argument.

2. For layers $k = K-1, \ldots, 2$, and for node ℓ in layer k, $\ell = 1, \ldots, p_k$, compute

$$\delta_\ell^k = \Big(\sum_{j=1}^{p_{k+1}} w_{j\ell}^k \delta_j^{k+1} \Big) \dot{g}^k(z_\ell^k),$$

   where $w_{j\ell}^k$ is an element of the $p_{k+1} \times p_k$ matrix $\boldsymbol{W}^k$.

3. The partial derivatives for the elements of $\boldsymbol{W}^k$ are then

$$\frac{\partial L}{\partial w_{j\ell}^k} = a_j^k \delta_\ell^{k+1}.$$

For stochastic gradient descent, the partial derivatives calculated as above are averaged over all the instances in a mini-batch, and this is used as the gradient in the updating of all the matrices $\boldsymbol{W}^k$ before the next epoch, that is, pass through the data.