11 Neural Network

11.1 Feedforward neural networks

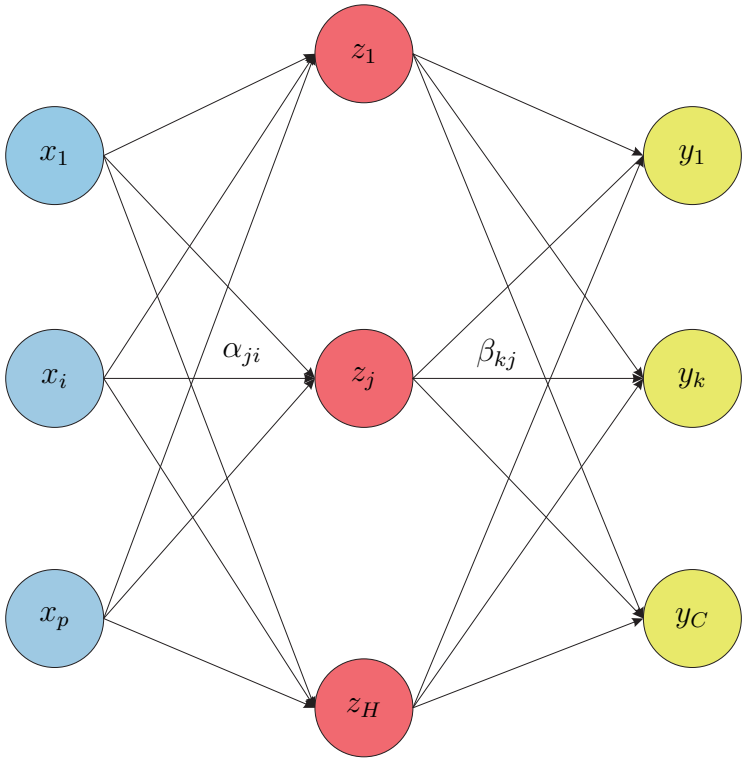


Figure 1: Neural Network with single hidden layer

We denote the complete set of weights by $\boldsymbol{\theta}$, which consists of

$$\boldsymbol{\theta} = \left\{ \begin{array}{l} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{array} \right.$$

Each of the α_j and β_k is a subset vector of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$,

$$\boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_j & \dots & \boldsymbol{\alpha}_H \end{pmatrix} = \begin{array}{c} \\ i \end{array} \begin{array}{c} \begin{array}{cccc} & & j & \\ \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \cdots & \boldsymbol{\alpha}_H \end{array} \\ \begin{pmatrix} \alpha_{11} & \alpha_{21} & \cdots & \alpha_{H1} \\ \alpha_{12} & \alpha_{22} & \cdots & \alpha_{H2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{1D} & \alpha_{2D} & \cdots & \alpha_{HD} \end{pmatrix} \end{array}$$

$$\boldsymbol{\alpha}^T = \begin{array}{c} \\ j \end{array} \begin{array}{c} \begin{array}{cccc} & & i & \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1D} \end{array} \\ \begin{pmatrix} \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{H1} & \alpha_{H2} & \cdots & \alpha_{HD} \end{pmatrix} \end{array}$$

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_k & \dots & \boldsymbol{\beta}_C \end{pmatrix} = \begin{array}{c} \\ j \end{array} \begin{array}{c} \begin{array}{cccc} & & k & \\ \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \cdots & \boldsymbol{\beta}_C \end{array} \\ \begin{pmatrix} \beta_{11} & \beta_{21} & \cdots & \beta_{C1} \\ \beta_{12} & \beta_{22} & \cdots & \beta_{C2} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{1H} & \beta_{2H} & \cdots & \beta_{CH} \end{pmatrix} \end{array}$$

$$\boldsymbol{\beta}^T = \begin{array}{c} \\ k \end{array} \begin{array}{c} \begin{array}{cccc} & & j & \\ \beta_{11} & \beta_{12} & \cdots & \beta_{1H} \end{array} \\ \begin{pmatrix} \beta_{21} & \beta_{22} & \cdots & \beta_{2H} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{K1} & \beta_{K2} & \cdots & \beta_{CH} \end{pmatrix} \end{array}$$

We can write the overall model as follows:

$$x_n \xrightarrow{\boldsymbol{\alpha}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\boldsymbol{\beta}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

x_n is a vector of length D containing D variables for the nth observation. That is,

$$x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{nD} \end{pmatrix} \tag{11.1}$$

In general, we have the following models extract from the overall model,

$$\begin{aligned} a_{nj} &= \boldsymbol{\alpha}_j^T x_n, \ j = 1, \ldots, H \\ z_{nj} &= g(a_{nj}), \ j = 1, \ldots, H \\ b_{nk} &= \boldsymbol{\beta}_k^T \mathbf{z}_n, \ k = 1, \ldots, C \\ \hat{y}_{nk} &= h(b_{nk}), \ k = 1, \ldots, C \end{aligned}$$

From the above equations, it's not hard to see that $\hat{y}_{nk}$ consisted from a bunch of other functions, this can be interpreted as,

$$\hat{\mathbf{y}}_n = h(\mathbf{b}_n) = h(\boldsymbol{\beta}^T \mathbf{z}_n) = h(\boldsymbol{\beta}^T g(\mathbf{a}_n)) = h(\boldsymbol{\beta}^T g(\boldsymbol{\alpha}^T x_n)) \tag{11.2}$$

Summarize the points above, x_n is the nth observation, $\mathbf{a}_n = \boldsymbol{\alpha}^T x_n$ be the pre-synaptic hidden layer, and $\mathbf{z}_n = g(\mathbf{a}_n)$ be the post-synaptic hidden layer, where g is some transfer funciton. We typically use $g(a) = sigm(a)$, but we may also use $g(a) = tanh(a)$. We now convert this hidden layer to the output layer. Let $\mathbf{b}_n = \boldsymbol{\beta}^T \mathbf{z}_n$ be the pre-synaptic output layer, and $\hat{\mathbf{y}}_n = h(\mathbf{b}_n)$ be the post-synaptic output layer, where h is another non-linearity. For a regression model, we use $h(\mathbf{b}) = \mathbf{b}$; for binary classification, we use $h(\mathbf{b}) = [sigm(b_1), \ldots, sigm(b_c)]$; for multi-class classification, we use $h(\mathbf{b}) = \mathcal{S}(\mathbf{b})$.

11.2 Backpropagation neural networks

We now discuss how to compute the gradient vector of the NLL(negative log likelihood) by applying the chain rule of calculus. The resulting algorithm is known as backpropagation.

In the regression case, with C outputs, the NLL is given by the by the squared error:

$$J(\boldsymbol{\theta}) = -\sum_n \sum_k (\hat{y}_{nk}(\boldsymbol{\theta}) - y_{nk})^2 \tag{11.3}$$

In the classification case, with C classes, the NLL is given by the cross entropy,

$$J(\boldsymbol{\theta}) = -\sum_n \sum_k y_{nk} \log \hat{y}_{nk}(\boldsymbol{\theta}) \tag{11.4}$$

Our task is to compute $\frac{d}{d\boldsymbol{\theta}} J$. The overall gradient is obtained by summing over n, we will derive this for each n separately or we will perform mini-batch and summing the batch size. Let us start by considering the output layer weights. We have

$$\nabla_{\boldsymbol{\beta}_k} J_n = \frac{\partial J_n}{\partial b_{nk}} \nabla_{\boldsymbol{\beta}_k} b_{nk} = \frac{\partial J_n}{\partial b_{nk}} \mathbf{z}_n \tag{11.5}$$

Assuming h is canonical link function for the output GLM, then

$$\frac{\partial J_n}{\partial b_{nk}} = \delta^{\beta}_{nk} = (\hat{y}_{nk} - y_{nk}) \tag{11.6}$$

So the overall gradient is

$$\nabla_{\boldsymbol{\beta}_k} J_n = \delta^{\beta}_{nk} \mathbf{z}_n \tag{11.7}$$

which is the pre-synaptic input to the output layer, namely $\mathbf{z}_n$, times the error signal, namely δ^{β}_{nk}.

For the input layer weights, we have

$$\nabla_{\boldsymbol{\alpha}_j} J_n = \frac{\partial J_n}{\partial a_{nj}} \nabla_{\boldsymbol{\alpha}_j} a_{nj} = \delta^{\alpha}_{nj} x_n \tag{11.8}$$

The first level error signal δ^{α}_{nj},

$$\delta^{\alpha}_{nj} = \frac{\partial J_n}{\partial a_{nj}} = \sum_k \frac{\partial J_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial a_{nj}} \tag{11.9}$$

where,

$$\frac{\partial b_{nk}}{\partial a_{nj}} = \frac{\partial b_{nk}}{\partial z_{nj}} \frac{\partial z_{nj}}{\partial a_{nj}} = \beta_{kj} g'(a_{nj}) \tag{11.10}$$

So, the first level error signal δ_{nj}^{α} is,

$$\delta_{nj}^{\alpha} = \frac{\partial J_n}{\partial a_{nj}} = \sum_k \delta_{nk}^{\beta} \beta_{kj} g'(a_{nj}) \tag{11.11}$$

In the end, the input layer weights looks like

$$\nabla_{\boldsymbol{\alpha}_j} J_n = \delta_{nj}^{\alpha} x_n = \sum_k \delta_{nk}^{\beta} \beta_{kj} g'(a_{nj}) x_n \tag{11.12}$$

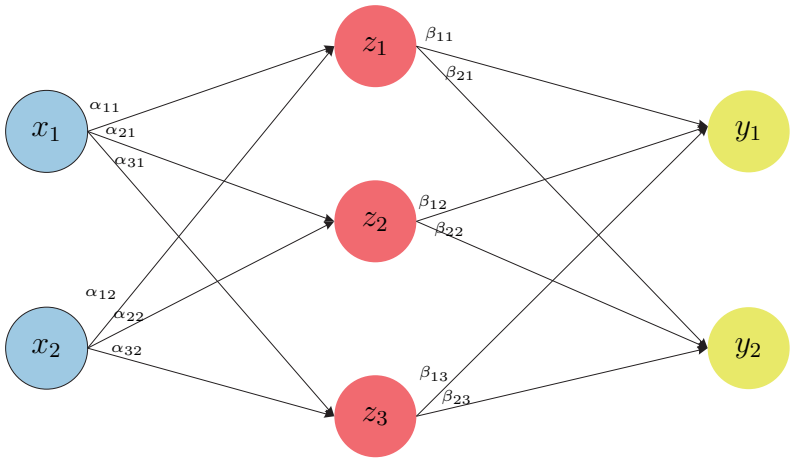
Given the derivatives, a gradient descent update at the $(r+1)$st iteration has the form

$$\boldsymbol{\beta}_k^{(r+1)} = \boldsymbol{\beta}_k^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\beta}_k} J_n$$
$$\boldsymbol{\alpha}_j^{(r+1)} = \boldsymbol{\alpha}_j^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\alpha}_j} J_n$$

Learning can be carried out online - processing each observation one at a time, updating the gradient after each training case, and cycling through the training cases many times. In this case, the sums of n in the update equations are replaced with single summation k. Online training allows the network to handle large training sets, and also update the weights as new observations come in.

The learning rate η for batch learning is usually taken to be a constant, and can be optimized by a **line search**.

11.2.1 Example 1



$$\boldsymbol{\alpha} = \begin{pmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \boldsymbol{\alpha}_3 \end{pmatrix} \tag{11.13}$$

$$\boldsymbol{\alpha}^T = \begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \\ \alpha_{31} & \alpha_{32} \end{pmatrix} \tag{11.14}$$

$$\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 \end{pmatrix} \tag{11.15}$$

$$\boldsymbol{\beta}^T = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} \\ \beta_{21} & \beta_{22} & \beta_{23} \end{pmatrix} \tag{11.16}$$

The first output layer,

$$a_1 = \boldsymbol{\alpha}_1^T x = \sum_{i=1}^{2} \alpha_{1i} x_i$$
$$a_2 = \boldsymbol{\alpha}_2^T x = \sum_{i=1}^{2} \alpha_{2i} x_i$$
$$a_3 = \boldsymbol{\alpha}_3^T x = \sum_{i=1}^{2} \alpha_{3i} x_i$$

So,

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{11.17}$$

The post-synaptic hidden layer

$$\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} g(a_1) \\ g(a_2) \\ g(a_3) \end{pmatrix} \tag{11.18}$$

The second output layer,

$$b_1 = \boldsymbol{\beta}_1^T \mathbf{z} = \sum_{j=1}^{3} \beta_{1j} z_j$$
$$b_2 = \boldsymbol{\beta}_2^T \mathbf{z} = \sum_{j=1}^{3} \beta_{2j} z_j$$

The estimated response,

$$\hat{\mathbf{y}} = \begin{pmatrix} h(b_1) \\ h(b_2) \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix}$$

a gradient descent update at the $(r+1)$st iteration has the form

$$\boldsymbol{\beta}_1^{(r+1)} = \boldsymbol{\beta}_1^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\beta}_1} J_n = \begin{pmatrix} \beta_{11} \\ \beta_{12} \\ \beta_{13} \end{pmatrix} - \eta \left(\sum_n (\hat{y}_{n1} - y_{n1}) \begin{pmatrix} z_{n1} \\ z_{n2} \\ z_{n3} \end{pmatrix} \right)$$

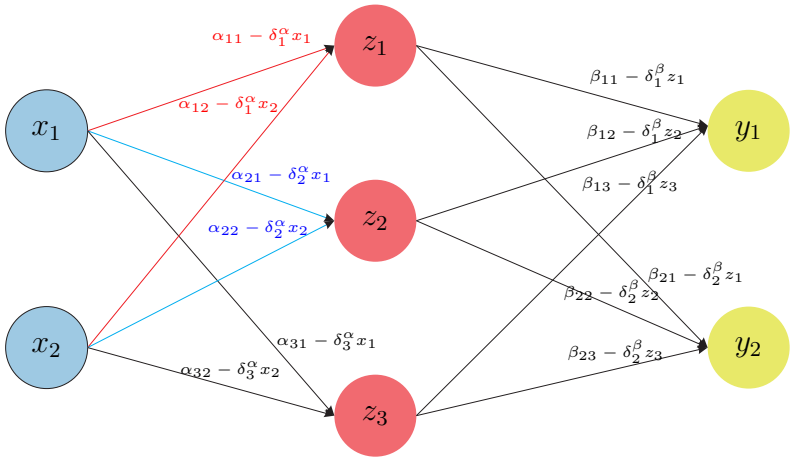
$$\boldsymbol{\beta}_2^{(r+1)} = \boldsymbol{\beta}_2^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\beta}_2} J_n = \begin{pmatrix} \beta_{21} \\ \beta_{22} \\ \beta_{23} \end{pmatrix} - \eta \left(\sum_n (\hat{y}_{n2} - y_{n2}) \begin{pmatrix} z_{n1} \\ z_{n2} \\ z_{n3} \end{pmatrix} \right)$$

$$\boldsymbol{\alpha}_1^{(r+1)} = \boldsymbol{\alpha}_1^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\alpha}_1} J_n = \begin{pmatrix} \alpha_{11} \\ \alpha_{12} \end{pmatrix} - \eta \left(\sum_n \sum_k (\delta_{nk}^{\beta} \beta_{k1}) g'(a_1) \begin{pmatrix} x_{n1} \\ x_{n2} \end{pmatrix} \right)$$

$$\boldsymbol{\alpha}_2^{(r+1)} = \boldsymbol{\alpha}_2^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\alpha}_2} J_n = \begin{pmatrix} \alpha_{21} \\ \alpha_{22} \end{pmatrix} - \eta \left(\sum_n \sum_k (\delta_{nk}^{\beta} \beta_{k2}) g'(a_2) \begin{pmatrix} x_{n1} \\ x_{n2} \end{pmatrix} \right)$$

$$\boldsymbol{\alpha}_3^{(r+1)} = \boldsymbol{\alpha}_3^{(r)} - \eta \sum_n \nabla_{\boldsymbol{\alpha}_3} J_n = \begin{pmatrix} \alpha_{31} \\ \alpha_{32} \end{pmatrix} - \eta \left(\sum_n \sum_k (\delta_{nk}^{\beta} \beta_{k3}) g'(a_3) \begin{pmatrix} x_{n1} \\ x_{n2} \end{pmatrix} \right)$$

The graph of back propagation



I didn't add learning rate on the update, since the tag is too small to add any words on it.