1 Definition of our Neural Network

Consider the following network: The input consists of D_I dimensions, there is one hidden layer with dimension D_H and the output has dimension D_O. Given two matrices $\Theta^{(1)} \in \mathbb{R}^{(D_I+1)\times D_H}$ and $\Theta^{(2)} \in \mathbb{R}^{(D_H+1)\times D_O}$, the forward propagation of an input vector $x \in \mathbb{R}^{D_I}$ (a column vector) works in the following way:

$$\begin{aligned}
a^{(1)} &= (\Theta^{(1)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} \quad \in \mathbb{R}^{D_H} \\
z &= \sigma\big(a^{(1)}\big) \quad \in \mathbb{R}^{D_H} \\
a^{(2)} &= (\Theta^{(2)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ z \end{pmatrix} \quad \in \mathbb{R}^{D_O} \\
y &= \sigma\big(a^{(2)}\big) \quad \in \mathbb{R}^{D_O}
\end{aligned}$$

For training purposes (and for computational issues) we will want to propagate multiple input vectors $x_1, x_2, ..., x_M$ with every $x_m \in \mathbb{R}^{D_I}$ at the same time. We collect them into a big input matrix

$$X = (x_1, x_2, ..., x_M) \in \mathbb{R}^{D_I \times M}$$

Each column represents a different input sample, each row contains an input feature (for example a pixel in a scanned image). Forward propagation then looks as follows:

$$\begin{aligned}
A^{(1)} &= (\Theta^{(1)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ X \end{pmatrix} \quad \in \mathbb{R}^{D_H \times M} \\
Z &= \sigma\big(A^{(1)}\big) \quad \in \mathbb{R}^{D_H \times M} \\
A^{(2)} &= (\Theta^{(2)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ Z \end{pmatrix} \quad \in \mathbb{R}^{D_O \times M} \\
Y &= \sigma\big(A^{(2)}\big) \quad \in \mathbb{R}^{D_O \times M}
\end{aligned}$$

where σ is the sigmoid function $\sigma(a) = \frac{1}{1+\exp(-a)}$

So in short

$$Y = \sigma\left\{ (\Theta^{(2)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ \sigma\left[(\Theta^{(1)})^{\mathrm{T}} \cdot \begin{pmatrix} 1 \\ X \end{pmatrix} \right] \end{pmatrix} \right\}$$

The ones of course represent rows containing M ones to match X and Z. For easier calculation we collect the following submatrix definitions:

$$\begin{aligned}
\Theta^{(1)} &= \begin{pmatrix} \Theta_0^{(1)} \\ \vdots \\ \Theta_{D_I}^{(1)} \end{pmatrix}, \text{ where } \left(\Theta_j^{(1)}\right)^{\mathrm{T}} \in \mathbb{R}^{D_H} \\
\Theta^{(2)} &= \begin{pmatrix} \Theta_0^{(2)} \\ \vdots \\ \Theta_{D_H}^{(2)} \end{pmatrix}, \text{ where } \left(\Theta_j^{(2)}\right)^{\mathrm{T}} \in \mathbb{R}^{D_O} \\
X &= (X_1, ..., X_M) \\
X_m &= \begin{pmatrix} X_{1,m} \\ \vdots \\ X_{D_I,m} \end{pmatrix}
\end{aligned}$$

$$
\begin{aligned}
Z &= (Z_1, ..., Z_M) \\
Z_m &= \begin{pmatrix} Z_{1,m} \\ \vdots \\ Z_{D_H,m} \end{pmatrix} \\
Y &= (Y_1, ..., Y_M) \\
Y_m &= \begin{pmatrix} Y_{1,m} \\ \vdots \\ Y_{D_O,m} \end{pmatrix} \\
A^{(1)} &= \begin{pmatrix} A_1^{(1)}, ..., A_M^{(1)} \end{pmatrix} \\
A_m^{(1)} &= \begin{pmatrix} A_{1,m}^{(1)} \\ \vdots \\ A_{D_H,m}^{(1)} \end{pmatrix} \\
A^{(2)} &= \begin{pmatrix} A_1^{(2)}, ..., A_M^{(2)} \end{pmatrix} \\
A_m^{(2)} &= \begin{pmatrix} A_{1,m}^{(2)} \\ \vdots \\ A_{D_O,m}^{(2)} \end{pmatrix}
\end{aligned}
$$

We will stick to the convention, that the summing indices are as follows:

- $i = (0,)1, ..., D_I$
- $j = (0,)1, ..., D_H$
- $k = 1, ..., D_O$
- $m = 1, ..., M$

where the 0th index always denotes the bias variable (so $X_{0,m} = Y_{0,m} = 1$ for all $m = 1, ..., M$)

2 Training the Neural Network

Neural Networks are supervised machine learning algorithms: We need a training set with the correct results to tune the weights $\Theta^{(r)}, r = 1, 2$. The training set is

- $X = (X_1, ..., X_M)$ is the training set input (and for convenience of length M). In our example, each X_m describes the set of pixels in the image to be resolved,
- $Y = (Y_1, ..., Y_M)$ is the training set output (for the currently – incorrect – chosen weights $\Theta^{(r)}$), i.e. the Neural Network's current guess for the digit the image contains,
- $L = (L_1, ..., L_M)$ is the given correct result, i.e. L_m contains information which digit the image actually shows. L follows the same notation as Y, so L_m is a column vector containing $L_{1,m}, ..., L_{D_O,m}$.

The training is supposed to tune the weights $\Theta^{(r)}$ so that $L \approx Y$ where the approximation is interpreted in terms of the following error function (called cross-entropy error function):

$$E(\Theta) = -\sum_{m=1}^{M} \sum_{k=1}^{D_O} L_{k,m} \cdot \ln Y_{k,m} + (1 - L_{k,m}) \cdot \ln(1 - Y_{k,m})$$

Our objective is to minimize the error function, i.e. find a set of weights $\Theta^{(r)}$ so that $E(\Theta)$ is small, which means that $L \approx Y$. This will be done in terms of a gradient descent algorithm, so we need the gradient of E with respect to Θ.

3 Learning by Backward Propagation

The gradients of E can be obtained in a reversed fashion, i.e. we start from Y and L and work our way back. This is called Backward Propagation and is done as follows. First notice that

$$\frac{\partial E(\Theta)}{\partial \Theta^{(r)}_{l_1,l_2}} = -\sum_{m=1}^{M} \sum_{k=1}^{D_O} \frac{L_{k,m} - Y_{k,m}}{Y_{k,m} \cdot (1 - Y_{k,m})} \cdot \frac{\partial Y_{k,m}}{\partial \Theta^{(r)}_{l_1,l_2}} \tag{1}$$

The first step of Backward Propagation is the last step of Forward Propagation: Consider first

$$Y_{k,m} = \sigma\left\{\sum_{j=0}^{D_H} \Theta^{(2)}_{j,k} \cdot Z_{j,m}\right\}, \text{ where } Z_{0,m} = 1, \tag{2}$$

so

$$\begin{aligned} \frac{\partial Y_{k,m}}{\partial \Theta^{(2)}_{j,k}} &= \sigma'\left\{\sum_{j=0}^{D_H} \Theta^{(2)}_{j,k} \cdot Z_{j,m}\right\} \cdot Z_{j,m} \\ &= \sigma'\Big(A^{(2)}_{k,m}\Big) \cdot Z_{j,m}, \end{aligned} \tag{3}$$

now we notice that $\sigma'(a) = \sigma(a)\,(1 - \sigma(a))$ and $Y_{k,m} = \big[\sigma\big(A^{(2)}\big)\big]_{k,m} = \sigma\Big(A^{(2)}_{k,m}\Big)$, hence

$$\sigma'\Big(A^{(2)}_{k,m}\Big) = Y_{k,m} \cdot (1 - Y_{k,m}). \tag{4}$$

Combined, we get

$$\begin{aligned} \frac{\partial E(\Theta)}{\partial \Theta^{(2)}_{j,k}} &= -\sum_{m=1}^{M} \frac{L_{k,m} - Y_{k,m}}{Y_{k,m} \cdot (1 - Y_{k,m})} \cdot Y_{k,m} \cdot (1 - Y_{k,m}) \cdot Z_{j,m} \\ &= -\sum_{m=1}^{M} (L_{k,m} - Y_{k,m}) \cdot Z_{j,m} \end{aligned} \tag{5}$$

It will be useful to abbreviate

$$\begin{aligned} \delta^{(3)}_{m} &= Y_m - L_m \\ \delta^{(3)}_{k,m} &= Y_{k,m} - L_{k,m}, \end{aligned}$$

hence

$$\boxed{\frac{\partial E(\Theta)}{\partial \Theta^{(2)}_{j,k}} = \sum_{m=1}^{M} \delta^{(3)}_{k,m} \cdot Z_{j,m},} \tag{6}$$

or (for compactness fetishists)

$$\frac{\partial E(\Theta)}{\partial \Theta^{(2)}} = \sum_{m=1}^{M} \delta^{(3)}_{m} \cdot Z^{\mathrm{T}}_{m}$$

Expanding $Y_{k,m}$ in terms of $\Theta^{(1)}$ gives us

$$\begin{aligned}
Y_{k,m} &= \sigma\left\{\sum_{j=0}^{D_H} \Theta^{(2)}_{j,k} \cdot Z_{j,m}\right\} \\
Z_{0,m} &= 1 \\
Z_{j,m} &= \sigma\left\{\sum_{i=0}^{D_I} \Theta^{(1)}_{i,j} \cdot X_{i,m}\right\}, \\
X_{0,m} &= 1
\end{aligned}$$

Derivation with respect to $\Theta^{(1)}$:

$$\begin{aligned}
\frac{\partial Y_{k,m}}{\partial \Theta^{(1)}_{i,j}} &= \sigma'\Big(A^{(2)}_{k,m}\Big) \cdot \Theta^{(2)}_{j,k} \cdot \frac{\partial Z_{j,m}}{\partial \Theta^{(1)}_{i,j}} \\
&= Y_{k,m} \cdot (1 - Y_{k,m}) \cdot \sigma'\left\{\sum_{i=0}^{D_I} \Theta^{(1)}_{i,j} \cdot X_{i,m}\right\} \cdot X_{i,m} \\
&= Y_{k,m} \cdot (1 - Y_{k,m}) \cdot \sigma'\Big(A^{(1)}_{j,m}\Big) \cdot \Theta^{(2)}_{j,k} \cdot X_{i,m} \\
\frac{\partial E(\Theta)}{\partial \Theta^{(1)}_{i,j}} &= -\sum_{m=1}^{M} X_{i,m} \cdot \sigma'\Big(A^{(1)}_{j,m}\Big) \cdot \sum_{k=1}^{D_O} (L_{k,m} - Y_{k,m}) \cdot \Theta^{(2)}_{j,k} \\
&= -\sum_{m=1}^{M} X_{i,m} \cdot \sigma'\Big(A^{(1)}_{j,m}\Big) \cdot \Theta^{(2)}_{j} \cdot (L_m - Y_m) \\
&= \sum_{m=1}^{M} \Theta^{(2)}_{j} \cdot \delta^{(3)}_{m} \cdot \sigma'\Big(A^{(1)}_{j,m}\Big) \cdot X_{i,m}
\end{aligned}$$

Hence,

$$\boxed{\frac{\partial E(\Theta)}{\partial \Theta^{(1)}_{i,j}} = \sum_{m=1}^{M} \Theta^{(2)}_{j} \cdot \delta^{(3)}_{m} \cdot \sigma'\Big(A^{(1)}_{j,m}\Big) \cdot X_{i,m}} \tag{7}$$

Using some gradient descent algorithm, we can find a local minimum for the cost function, i.e. problem adapted values for Θ.