Feed-forward neural network: backpropagation

Felipe Salvatore
felipessalvador@googlemail.com

February 23, 2017

General

For $n \geq 1$, let $D_0, D_1, \dots, D_{n+1} \in \mathbb{N}$ (all grater than 0); let $f^{(1)}, \dots, f^{(n)}\mathbb{R}^{\mathbb{R}}$ be a sequence of non-linear functions; let $(x^1, y^1), \dots, (x^N, y^N)$ be the observations of a dataset (where $x^d \in \mathbb{R}^{D_0}$, and $y^d \in \mathbb{R}^{D_{n+1}}$;let $W^{(k)} \in \mathbb{R}^{D_{k-1}, D_k}$ and $b^{(k)} \in \mathbb{R}^{D_k}$ for $k = 1, \dots, n$. A neural network with n hidden layers is a function applied to the observation x^d from the dataset as follows:

$$a^{(1)} = x^d \tag{1}$$

$$z_i^{(k)} = \sum_{s=1}^{D_{(k-1)}} W_{s,i}^{(k)} a_s^{(k)} + b_i^{(k)} \quad \text{with } i = 1, \dots, D_k \text{ and } k = 1, \dots, n \tag{2}$$

$$a^{(k)} = f^{(k-1)}(z^{(k-1)}) \quad \text{with } k = 2, \dots, n+1 \tag{3}$$

$$\hat{y} = g(z^{(n+1)}) \tag{4}$$

The Figure 1 shows a visual representation of this model. We called **forward propagation** the calculation of $\hat{y}$. If we want to be explicit we should write $\hat{y}(W^{(1)}, \dots, W^{(n+1)}, b^{(1)}, \dots, b^{(n+1)}, x^d)$, but to make things simple, we will only write $\hat{y}(x^d)$. Let $Error$ be a function to measure the error between the hypothesis and the target, thus the error for a single observation is:

$$J = Error(\hat{y}(x^d), y^d) \tag{5}$$

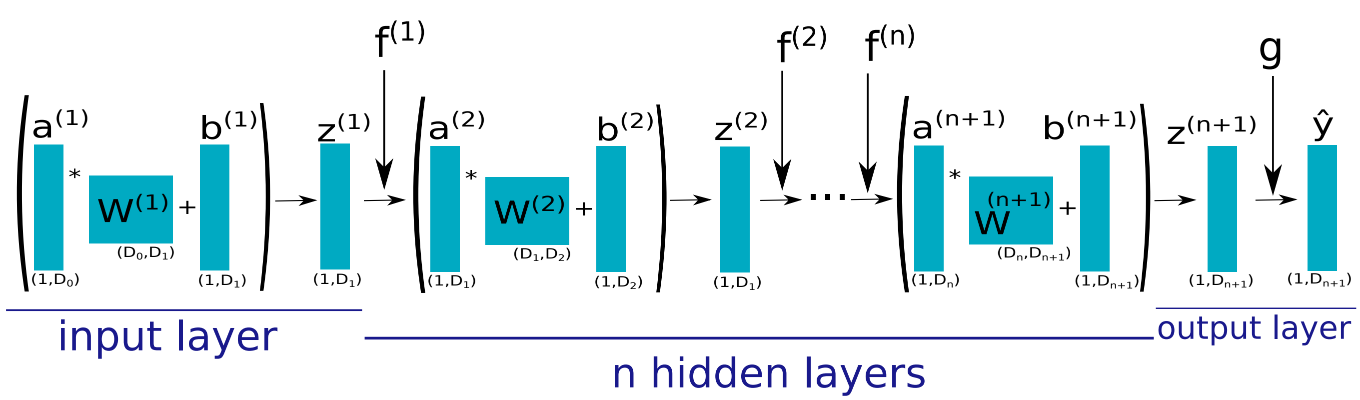

Figure 1: A neural network with n hidden layers

if we use regularization

$$J = Error(\hat{y}(x^d), y^d) + \sum_{k=1}^{n+1} \frac{1}{2}\lambda \sum_{i=1}^{D_{k-1}} \sum_{j=1}^{D_k} (W_{i,j}^{(k)})^2 \tag{6}$$

and for the whole dataset

$$J = \frac{1}{N} \sum_{d=1}^{N} (Error(\hat{y}(x^d), y^d) + \sum_{k=1}^{n+1} \frac{1}{2}\lambda \sum_{i=1}^{D_{k-1}} \sum_{j=1}^{D_k} (W_{i,j}^{(k)})^2) \tag{7}$$

To apply SGD we need to calculate the derivatives for each parameter $(W^{(1)}, \ldots, W^{(n+1)}, b^{(1)}, \ldots, b^{(n+1)}$. The **back propagation algorithm** give us a easy method for doing that. We will compute error signals $\delta^{(2)}, \ldots, \delta^{(n+2)}$ recursively in the reverse order as follows:

$$\hat{y}(x^d) - y^d \text{ [1]} \tag{8}$$

$$\delta_j^{(k)} = \sum_{s=1}^{D_k} \delta_s^{(k+1)} W_{j,s}^{(k)} f'^{(k-1)}(z_j^{(k-1)}) \tag{9}$$

with $j = 1, \ldots, D_{(k-1)}$ and $k = 2, \ldots, n+1$.

$$\frac{\partial J}{\partial W_{i,j}^{(k)}} = a_i^{(k)} \delta_j^{(k+1)} \tag{10}$$

with $i = 1, \ldots, D_{(k-1)}$, $j = 1, \ldots, D_{(k)}$ and $k = 1, \ldots, n+1$. If the cost function is 7, then

$$\frac{\partial J}{\partial W_{i,j}^{(k)}} = a_i^{(k)} \delta_j^{(k+1)} + \lambda W_{i,j}^{(k)} \tag{11}$$

$$\frac{\partial J}{\partial b_j^{(k)}} = \delta_j^{(k+1)} \tag{12}$$

with $j = 1, \ldots, D_{(k)}$ and $k = 1, \ldots, n+1$.

To make things simpler, we can simplify the equations (9),(10) and (12) using vector notation. For $A, B \in \mathbb{R}^{n,m}$ $A \circ B$ is the Hadamard product, i.e., $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$. Similarly for $u, v \in \mathbb{R}^n$, $(v \circ u)_i = v_i u_i$. Hence,

$$\boldsymbol{\delta^{(k)} = (\delta^{(k+1)} (W^{(k)})^T) \circ f'^{(k-1)}(z^{(k-1)})} \tag{13}$$

$$\boldsymbol{\frac{\partial J}{\partial W^{(k)}} = a^{(k)} \delta^{(k+1)T}} \tag{14}$$

$$\boldsymbol{\frac{\partial J}{\partial b^{(k)}} = \delta^{(k+1)}} \tag{15}$$

[1] This is not the general form, since it depends of the error function. To be honest, it is not clear to me if $\delta^{(n+2)}$ is $\frac{\partial J}{\partial \hat{y}}$ or $\frac{\partial J}{\partial z^{n+1}}$

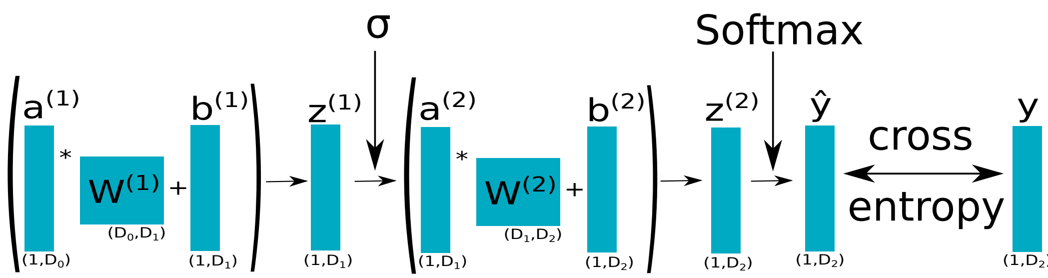

Figure 2: A neural network with 1 hidden layer

Example

Figure 2 shows a neural network with a single hidden layer. The only non-linear function is the sigmoid function σ and the function in the output layer is the softmax function. As an error measure we use cross entropy, assuming $y \in \mathbb{R}^{D_2}$ is an one-hot vector. Hence

$$z_i^{(1)} = \sum_{s=1}^{D_x} W_{s,i}^{(1)} x_s + b_i^{(1)} \quad \text{with } i = 1, \dots, D_1 \tag{16}$$

$$a_i^{(2)} = \sigma(z_i^{(1)}) \quad \text{with } i = 1, \dots, D_1 \tag{17}$$

$$z_j^{(2)} = \sum_{s=1}^{D_1} W_{s,j}^{(2)} a_s^{(2)} + b_j^{(2)} \quad \text{with } j = 1, \dots, D_2 \tag{18}$$

$$\hat{y}_j = softmax(z_j^{(2)}) \quad \text{with } j = 1, \dots, D_2 \tag{19}$$

$$J(y, \hat{y}) = CE(y, \hat{y}) = -\sum_{s=1}^{D_2} y_s \log(\hat{y}_s) \tag{20}$$

where CE stands for *cross-entropy*.

Note that $\frac{\partial J}{\partial z^{(2)}} = \hat{y} - y$. So taking $\delta^{(3)} = \hat{y} - y$, we can apply the back propagation algorithm. For $i \in 1, \dots, D_1$ and $j \in 1, \dots, D_2$ we have

$$\begin{aligned}\frac{\partial J}{\partial W_{i,j}^{(2)}} &= a_i^{(2)} \delta_j^{(3)} \\ &= a_i^{(2)} (\hat{y}_j - y_j) \, .\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_j^{(2)}} &= \delta_j^{(3)} \\ &= (\hat{y}_j - y_j) \, .\end{aligned}$$

For $j \in 1, \ldots, D_1$ we have

$$\begin{aligned}\delta_j^{(2)} &= \sum_{s=1}^{D_2} \delta_s^{(3)} W_{j,s}^{(2)} \sigma'(z_j^{(1)}) \\ &= \sum_{s=1}^{D_2} (\hat{y}_s - y_s) W_{j,s}^{(2)} \sigma'(z_j^{(1)}) \ .\end{aligned}$$

For $i \in 1, \ldots, D_0$ and $j \in 1, \ldots, D_1$ we have

$$\begin{aligned}\frac{\partial J}{\partial W_{i,j}^{(1)}} &= a_i^{(1)} \delta_j^{(2)} \\ &= a_i^{(1)} \sum_{s=1}^{D_2} (\hat{y}_s - y_s) W_{j,s}^{(2)} \sigma'(z_j^{(1)}) \ .\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_j^{(1)}} &= \delta_j^{(2)} \\ &= \sum_{s=1}^{D_2} (\hat{y}_s - y_s) W_{j,s}^{(2)} \sigma'(z_j^{(1)}) \ .\end{aligned}$$