1 Forward evaluation

Let $\mathcal{D} = \{(x, y)\}$ be a dataset of images x and labels y.

Let $N(x; W, b)$ be the neural network function, with weights W and biases b, taking inputs from a square $n \times n$ black-and-white image x with pixel values in $[0, 1]$ and outputting c categories, that is

$$N : [0,1]^{n\times n} \longrightarrow [0,1]^c$$

As always this function is a composition of d layers $N(x) = (N_1 \circ \cdots \circ N_d)(x)$. For each layer let

$$x^k = N_k(x^{k-1}) = \sigma(W^k x^{k-1} + b^k) \qquad \forall\, k = 1, \ldots, d$$
$$x_0 = x \in [0,1]^{n^2} \quad \text{(the flattened image)}$$

Here W^k is a matrix of dimension $n_k \times n_{k-1}$, and b a vector in $\mathbb{R}^{n_k}$. As usual the dimensions of the affine transformations $\{n_k\}_{k=0}^d$ are to choose as part of the network design process. Therefore we will refer to W and b as the lists of the matrices $\{W^k\}$ and vectors $\{b^k\}$ respectively. Furthermore we assume the activation function σ to be the same for all layers.

In the code, the evaluation of the neural network function N is performed in the function `guess`, while `read` preserves and returns the vectors

$$v^k = W^k x^{k-1} + b^k$$

resulting from the affine transformation alone (without applying the activation σ). This extra step allows backpropagation.

2 Backpropagation

Again, let $\mathcal{D} = \{(x, y)\}$ be a dataset of images x and corresponding labels $y \in \{0, 1\}^c$ encoded as one-hot vectors. Given any loss function

$$L : [0,1]^c \times \{0,1\}^c \longrightarrow \mathbb{R}$$

we use L to quantify the neural network's performance by evaluating $L(N(x), y)$. The derivative of this last expression with respect to each parameter W^k, b^k is then used to adjust the parameters themselves through gradient descent.

By the standard chain-rule for derivatives we get

$$\partial_{W^d} L(N(x), y) = \partial_N L(N(x), y)\ \partial_{W^d} x^d$$
$$\partial_{b^d} L(N(x), y) = \partial_N L(N(x), y)\ \partial_{b^d} x^d$$

for the last (d-th) level of the network. More generally, for every $0 \le k \le d-1$

$$\begin{aligned} \partial_{W^k} L(N(x), y) &= \partial_N L(N(x), y) \left[\prod_{h=0}^{d-k-1} \partial_{x^{d-h-1}} x^{d-h} \right] \partial_{W^k} x^k \\ \partial_{b^k} L(N(x), y) &= \partial_N L(N(x), y) \left[\prod_{h=0}^{d-k-1} \partial_{x^{d-h-1}} x^{d-h} \right] \partial_{b^k} x^k \end{aligned} \tag{1}$$

Of course

$$\partial_{x^{k-1}} x^k = diag(\sigma'(W^k x^{k-1} + b^k))\ W^k$$

where $diag(v)$ is the diagonal matrix having the vector v as diagonal. Instead $\partial_{W^k} x^k$ are tensors of rank 3 which we prefer to write component-wise:

$$\partial_{W^k_{ij}} x^k_t = \sigma'(v^k_t) x^{k-1}_j \delta_{it}$$

while

$$\partial_{b^k_i} x^k_t = \sigma'(v^k_t) x^{k-1}_j \delta_{it}$$

The vectors $v^k = W^k x^{k-1} + b^k$ were returned from the `read` function.
The function `gradErr` computes $\partial_{W^k} L(N(x), y)$ and $\partial_{b^k} L(N(x), y)$ going backwards from $k = d$ to $k = 1$ and stores them in two separate lists, reverses them, and returns them.