1 Background

We want to determine gradients of a Loss function to adjust parameters of the model: $\{w, b\}$. To do this, I have to use the chain rule, for:

$$a_j^{(L)} = \sigma(z_j^{(L)}) = \sigma(w_{ji}^{(L)} a_i^{(L-1)} + b_l^{(L)}) \quad (1)$$

$$C = \frac{1}{2}[f(x) - a_j^{(L)}]^2 \quad (2)$$

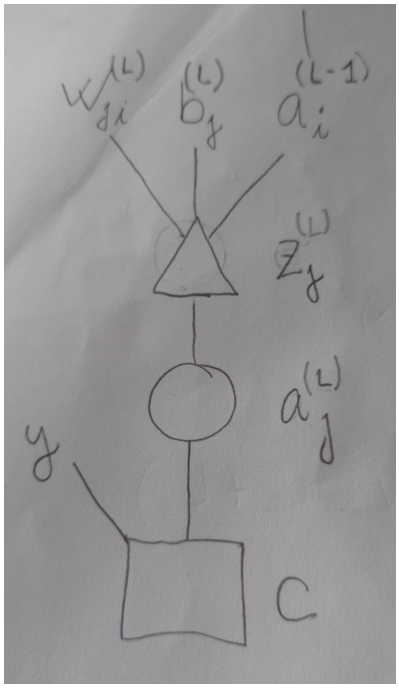

Figure 1: The proposed NN.

$$\frac{\partial C}{\partial w_{ji}^{(L)}} = \frac{\partial C}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ji}^{(L)}} \quad (3)$$

$$\frac{\partial C}{\partial b_j^{(L)}} = \frac{\partial C}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}\,, \quad (4)$$

$$\frac{\partial C}{\partial w_{ji}^{(L-1)}} = \frac{\partial C}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \frac{\partial z_i^{(L-1)}}{\partial w_{ik}^{(L-1)}} \quad (5)$$

resulting in

$$\frac{\partial C}{\partial w_{ji}^{(L)}} = [a_j^{(L)} - f(x)] \times \sigma'(z_j^{(L)}) \times a_i^{(L-1)} \quad (6)$$

$$\frac{\partial C}{\partial b_j^{(L)}} = [a_j^{(L)} - f(x)] \times \sigma'(z_j^{(L)}) \quad (7)$$

$$\frac{\partial C}{\partial w_{ji}^{(L-1)}} = [a_j^{(L)} - f(x)] \times \sigma'(z_j^{(L)}) \times w_{ji}^{(L)} \times w_{ik}^{(L-1)} \times a_k^{(L-2)} \,. \quad (8)$$

2 1-3-1 architecture

We will try to reproduce a function $f(x)$, such as cuadratic, cubic, etc..

Training set: Choose a large number N of pairs $x, f(x)$.

Activation: a sigmoid will only fit curves inside $f(x) = [0, 1]$; the ReLU only positive $f(x)$. So, it should be useful not to use a linear activation for the last layer: $\sigma = x$.

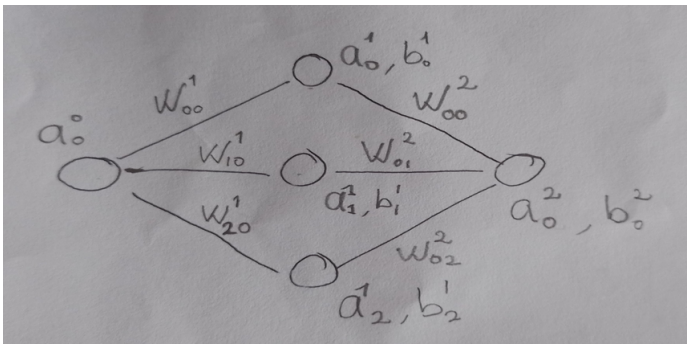

Figure 2: The proposed NN.

3 Equations

$$a_0^{(2)} = \sigma(z_0^{(2)}) = \sigma(w_{00}^{(2)} a_0^{(1)} + w_{01}^{(2)} a_1^{(1)} + w_{02}^{(2)} a_2^{(1)} + b_0^{(2)}) \quad (9)$$

$$a_0^{(1)} = \sigma(z_0^{(1)}) = \sigma(w_{00}^{(1)} a_0^{(0)} + b_0^{(1)}) \quad (10)$$

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \sigma(w_{10}^{(1)} a_0^{(0)} + b_1^{(1)}) \quad (11)$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \sigma(w_{20}^{(1)} a_0^{(0)} + b_2^{(1)}) \quad (12)$$

Loss function (stochastic GD):

$$C = \frac{1}{m}\sum_{x=0}^{m-1}\sum_{i=0}^{0}\frac{1}{2}[f(x) - a_i^{(2)}]^2 = \frac{1}{m}\sum_{x=0}^{m-1} C_x\,, \tag{13}$$

with $C_x = \frac{1}{2}\,[a_0^{(2)} - f(x)]^2$. The number m is the subgroup of points to apply the stochastic GD method (mini-batch).

4 Back-propagation equations

Model parameters: $\{w_{00}^{(2)}, w_{01}^{(2)}, w_{02}^{(2)}, w_{00}^{(1)}, w_{10}^{(1)}, w_{20}^{(1)}, b_0^{(2)}, b_0^{(1)}, b_1^{(1)}, b_2^{(1)}\}$.

$$\frac{\partial C_x}{\partial w_{00}^{(2)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})a_0^{(1)} \tag{14}$$

$$\frac{\partial C_x}{\partial w_{01}^{(2)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})a_1^{(1)} \tag{15}$$

$$\frac{\partial C_x}{\partial w_{02}^{(2)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})a_2^{(1)} \tag{16}$$

$$\frac{\partial C_x}{\partial w_{00}^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{00}^{(2)}\sigma'(z_0^{(1)})a_0^{(0)} \tag{17}$$

$$\frac{\partial C_x}{\partial w_{10}^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{01}^{(2)}\sigma'(z_1^{(1)})a_0^{(0)} \tag{18}$$

$$\frac{\partial C_x}{\partial w_{20}^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{02}^{(2)}\sigma'(z_2^{(1)})a_0^{(0)} \tag{19}$$

$$\frac{\partial C_x}{\partial b_0^{(2)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)}) \tag{20}$$

$$\frac{\partial C_x}{\partial b_0^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{00}^{(2)}\sigma'(z_0^{(1)}) \tag{21}$$

$$\frac{\partial C_x}{\partial b_1^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{01}^{(2)}\sigma'(z_1^{(1)}) \tag{22}$$

$$\frac{\partial C_x}{\partial b_2^{(1)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})w_{02}^{(2)}\sigma'(z_2^{(1)}) \tag{23}$$

4.1 Using error notation

We see a lot of repetition of (now called error) $\delta_0^{(2)} = \frac{\partial C_x}{\partial z_0^{(2)}} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)})$ and $\frac{\partial C_x}{\partial z_i^{(1)}}$:

$$\delta_0^{(1)} = \frac{\partial C_x}{\partial z_0^{(1)}} = \delta_0^{(2)} w_{00}^{(2)} \sigma'(z_0^{(1)}) \tag{24}$$

$$\delta_1^{(1)} = \frac{\partial C_x}{\partial z_1^{(1)}} = \delta_0^{(2)} w_{01}^{(2)} \sigma'(z_1^{(1)}) \tag{25}$$

$$\delta_2^{(1)} = \frac{\partial C_x}{\partial z_2^{(1)}} = \delta_0^{(2)} w_{02}^{(2)} \sigma'(z_2^{(1)}) \tag{26}$$

The back-propagation equations read

$$\frac{\partial C_x}{\partial w_{00}^{(2)}} = \delta_0^{(2)} a_0^{(1)} \tag{27}$$

$$\frac{\partial C_x}{\partial w_{01}^{(2)}} = \delta_0^{(2)} a_1^{(1)} \tag{28}$$

$$\frac{\partial C_x}{\partial w_{02}^{(2)}} = \delta_0^{(2)} a_2^{(1)} \tag{29}$$

$$\frac{\partial C_x}{\partial w_{00}^{(1)}} = \delta_0^{(1)} a_0^{(0)} \tag{30}$$

$$\frac{\partial C_x}{\partial w_{10}^{(1)}} = \delta_1^{(1)} a_0^{(0)} \tag{31}$$

$$\frac{\partial C_x}{\partial w_{20}^{(1)}} = \delta_2^{(1)} a_0^{(0)} \tag{32}$$

$$\frac{\partial C_x}{\partial b_0^{(2)}} = \delta_0^{(2)} \tag{33}$$

$$\frac{\partial C_x}{\partial b_0^{(1)}} = \delta_0^{(1)} \tag{34}$$

$$\frac{\partial C_x}{\partial b_1^{(1)}} = \delta_1^{(1)} \tag{35}$$

$$\frac{\partial C_x}{\partial b_2^{(1)}} = \delta_2^{(1)} \tag{36}$$

which in compact form for layer 2 is:

$$\frac{\partial C_x}{\partial w_{0i}^{(2)}} = \delta^{(2)} a_i^{(1)} \quad (\text{for } i = 0, 1, 2) \tag{37}$$

$$\frac{\partial C_x}{\partial b_0^{(2)}} = \delta^{(2)} \tag{38}$$

and for layer 1 is:

$$\frac{\partial C_x}{\partial w_{i0}^{(1)}} = \delta_i^{(1)} a_0^{(0)} \quad (\text{for } i = 0, 1, 2) \tag{39}$$

$$\frac{\partial C_x}{\partial b_i^{(1)}} = \delta_i^{(1)} \quad (\text{for } i = 0, 1, 2) \tag{40}$$

supplemented by

$$\delta_0^{(2)} = [a_0^{(2)} - f(x)]\sigma'(z_0^{(2)}) \tag{41}$$

$$\delta_i^{(1)} = \delta_0^{(2)} w_{0i}^{(2)} \sigma'(z_i^{(1)}) \tag{42}$$

were the activation functions are different for each layer. 1) layer 1 (hidden): ReLu; 2) layer 2 (output): linear, just z. These choices will prevent the output to be bound to 1 (case 1) and speed up learning (case 2).

I note that $a_0^{(0)} = x$ and $a_0^{(2)} = f(x)$.

4.2 Pseudocode

```
N = 100 // size of training set
x_interval = (0, 1)
// input: a function f(x) and an interval
// generate_Nps() returs an N-long array of pairs {x, f(x)}:
// the training set
N_pair_set = generate_Nps(f(x), x_interval, N)
// using sql, generate_sql_Ndb() creates the training database
training_Ndb = generate_sql_Ndb(N_pair_set)
// choose  arbitrarily a set of weights and biases using normal
// distr around norm_mean
wb = initialize_w_and_b(norm_mean, norm_deviation)
delta = 0.01 // threshold for C
eta = 0.01 // learning rate


mini_batch_size = 10
epoch_number = 100

FOR epoch IN 0..epoch_number - 1:
    // shuffle_Ndb() shuffles and separates into mini-batches
    // of size m
    training_Ndb_shuffled = shuffle_Ndb(training_Ndb, m)
    C[epoch] = 0
    FOR mini_batch IN training_Ndb_shuffled:
        C_mb = 0
        gradC_mb = 0
        FOR x IN mini_batch:
            // calculate vector of a's with eqs. 1-4
            a_0^0 = x
```

```
            a = forward_pass(wb, x)
            // calculate Loss function;
            // f(x) extracted from mini_batch
            C_x = calc_Cx(a, f(x))
            C_mb = C_mb + C_x
            // calculate gradient with eqs. 6-15;
            // f(x) extracted from mini_batch
            grad_C_x = grad(a, f(x), wb)
            gradC_mb =  gradC_mb +  grad_C_x
            C_mb = C_mb / m // this is C in the latex notes
        gradC_mb = gradC_mb / m
        // new weights and biases
        wb = wb { eta *  gradC_mb
        // --- accumulating C for all mini-batches
        C[epoch] = C[epoch] + C_mb
    // C[epoch] is average over the entire epoch (N points)
    C[epoch] = C[epoch] / (N/m)
    IF epoch % 5 = 0 AND C[epoch] < delta : BREAK
    IF epoch > 0:
        IF Abs( 2*(C[epoch-1] { C[epoch])/(C[epoch-1] + C[epoch]) )
            < 0.01 : \eta = 0.8 * \eta

FUNCTIONS

// generate_Nps() returs an N-long array of pairs {x, f(x)}:
// the training set
generate_Nps(f(x), x_interval, N)
// using sql, generate_sql_Ndb() creates the training database
training_Ndb = generate_sql_Ndb(N_pair_set, m)
// calculate Loss function for a single x point
calc_Cx(a, training_Ndb_shuffled)
// calculate gradient with eqs. 6-15
grad(a, training_Ndb_shuffled, wb)
```