Hand Written Digits Project

- Activation Function: Sigmoid Function:

$$\boxed{\sigma(z) = \frac{1}{1+e^{-z}}}$$

↳ Considering the input of an activation function $z = \sum_j w_j x_j + b$

$$\sigma(z) = \frac{1}{1+e^{-(\sum_j w_j x_j + b)}}$$

↳ Derivative of it:

$$\frac{d\sigma}{dz} = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{0\cdot(1+e^{-z}) - 1\cdot(-e^{-z})}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right.$$

$$\Rightarrow \boxed{\frac{\partial\sigma}{\partial z} = \sigma(1-\sigma)}$$

- Input:
  - ↳ Hand Written Digit image of size $28 \times 28 = 784$ pixels
  - ↳ The input pixels are greyscale, with $0.0 \rightarrow$ white and $1.0 \rightarrow$
  - ↳ Thus, we have an input layer with 784 "neurons".
  - ↳ 50.000 training inputs/samples

y

- Output / Desired output:

↳ Desired output: column array one hot -encoded where 1 means the desired Digit. In other words: $y[i] = 1 \Rightarrow$ the image is a hand written number i. Ex.: $(0,0,0,1,0,0,0,0,0,0)^T \Rightarrow$ Image with a 3

↳ Thus, the output layer has 10 neurons.

↳ the predicted output is given by the activation function of the last layer.

- Loss/Cost Function:

$$\boxed{C = \frac{1}{2n} \sum_{x=0}^{n} \| y(x) - \hat{y}(x) \|^2}$$

- Gradient Descent:

↳ Let's suppose C as a function of many variables $v_1 \dots v_m$:

$$C(v_1 \dots v_m) \Rightarrow \Delta C \approx \sum_{i=1}^{m} \frac{\partial C}{\partial v_i} \Delta v_i \text{ , for a tiny } \Delta v_i .$$

$$\Rightarrow \Delta C \approx \nabla C \cdot \Delta v$$

↳ Our goal is to choose wisely Δv in order to make ΔC negative (decrease the loss)

↳ A way of doing that is choosing: $\Delta v = -\underbrace{\eta}_{\text{learning rate}} \nabla C$,

because, in that way, $\Delta C = -\eta \|\nabla C\|^2$

↳ After a small Δv choosen that way we will have:

$$v' = v - \eta \nabla C$$

↳ Finally, since C is a function of all weights and biases we want to modify and adjust to increase accuracy, the rules we will apply is:

$$\boxed{\begin{aligned} w_k' &= w_k - \eta \frac{\partial C}{\partial w_k} \\ b_l' &= b_l - \eta \frac{\partial C}{\partial b_l} \end{aligned}}$$

↳ To speed up gradient descent algorithm we will implement the stochastic gradient descent, assuming that:

$$\underline{m << n} \qquad \frac{\sum_{j=1}^{m} \nabla C_{s_j}}{m} \approx \frac{\sum_{j=1}^{n} \nabla C_{s_j}}{n} = \nabla C$$

– Each loop shuffle the samples array and take $\left(\frac{\text{samples}}{\text{mini-batch-size}}\right)$ batches.

– Use each mini-batch to update the weights and biases. Repeat this process some amount of times. For instance: In one loop take 5.000 batches of 10 samples each. Repeat this process 30 times.

– It is important to say that after implementing a neural network, the values choose to η, m and the repetitions might require an adjustment to reach a satisfactory result.

- Back Propagation Algorithm :

1- Imput a mimi-batch

2- For each Sample s perform the following steps:

- Feedforward: for each layer compute:

$$\begin{cases} z^l = w^l \cdot a^{l-1} + b \\ a^l = \sigma(z^l) \end{cases}$$

- Output error: Compute the error vector regarding the last layer:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- Back propagate the error: For each layer starting from L-1 use the eq 2 to compute the error vector of this layer:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- Compute the derivatives regarding weights and biases:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad ; \quad \frac{\partial C}{\partial w_{k,j}^l} = a_k^{l-1} \cdot \delta_j^l$$

3- Gradient descent: After calculate all $\frac{\partial C}{\partial b}$ and $\frac{\partial C}{\partial w}$ update the wheights and biases:

$$\begin{cases} (w_{kj}^l)' = w_{kj}^l - \frac{\eta}{m} \sum_{s=0}^{m} \frac{\partial C_s}{\partial w_{kj}^l} \\ (b_m^l)' = b_m^l - \frac{\eta}{m} \sum_{s=0}^{m} \frac{\partial C_s}{\partial b_m^l} \end{cases}$$

↳ Finally, after each mini-batch of training inputs:

$$\boxed{\begin{aligned} w'_k &= w_k - \frac{\eta}{m}\sum_{j=0}^{m}\frac{\partial C_{s_j}}{\partial w_k} \\ b'_l &= b_l - \frac{\eta}{m}\sum_{j=0}^{m}\frac{\partial C_{s_j}}{\partial b_l} \end{aligned}}$$

• Back Propagation

- Relation between the activation output of a layer and the activation output of the previous layer:

$$a_j^l = \sigma\Big(\underbrace{\sum_k w_{kj}^l a_k^{l-1} + b_j^l}_{z_j^l}\Big)$$

- The quadratic cost for a single training sample:

$$C_x = \frac{1}{2}\|y - a^L\|^2 \hookrightarrow \frac{\partial C_x}{\partial a^L} = a^L - y$$

Error Function:

- Let's suppose that in the neuron (l, j) a small Δz_j^l is applied in the input z_j^l of the activation function. Then, the overall cost would change by an amount of: $\Delta C = \dfrac{\partial C}{\partial z_j^l}\Delta z_j^l$

- Notice that we will try to find a Δz_j^l which makes the cost smaller.
- Notice that, if $\frac{\partial C}{\partial z_j^l}$ has a large absolute value, then we must choose a Δz_j^l which has the opposite sign to $\frac{\partial C}{\partial z_j^l}$. But, if $\frac{\partial C}{\partial z_j^l}$ is small, then any choise of a small Δz_j^l will not be sufficient to impact ΔC. That means that, when $\frac{\partial C}{\partial z_j^l}$ has a big absolute value, we can use it to decrease the cost function. This neuron must be adjusted. In another hand, when $\frac{\partial C}{\partial z_j^l}$ is small, we can't adjust this neuron to reach a smaller cost. we need to look for other neurons to adjust. This one, then, is considered already adjusted.
- Thus, we can choose $\frac{\partial C}{\partial z_j^l}$ as the neuron error, since we whant to decrease it to reach a adjusted neuron.

Error: $\boxed{\delta_j^l = \frac{\partial C}{\partial z_j^l}}$

Fundamental equations:

1) $$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \Longleftrightarrow \delta^L = \nabla_a C \odot \sigma'(z^L)$$

Computing the error in the last layer

2) computing the relation between the error of one layer and the error of the next layer

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

3) the rate of change of cost function with respect to a bias:

what we want → $$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$ (Yes, it is the same error)

why? ⇒ $$\delta_j^l = \frac{\partial C}{\partial z_j^l} \Rightarrow \frac{\partial C}{\partial b_j^l} = \underbrace{\frac{\partial C}{\partial z_j^l}}_{\delta_j^l} \cdot \underbrace{\frac{\partial z_j^l}{\partial b_j^l}}_{1} \Rightarrow \frac{\partial C}{\partial b_j^l} = \delta_j^l$$ //

4) An equation for the rate of change of the cost with respect to any weight:

$$\frac{\partial C}{\partial w_{kj}^l} = a_k^{l-1} \cdot \delta_j^l$$

↑ what we actually want