Implementing a Neural Network for MNIST digit classification

→ The neural network to implement will allow MNIST digit classification

↳ Training images are 28x28 pixels $\boxed{784}\,28$ 28

↳ Each pixel has a pixel value [0, 255], where 0 = Black and 255 = White

→ The images can be represented as a matrix

↳ Normaly the matrix would be something like this : $A = \begin{bmatrix} — x^{(1)} — \\ — x^{(2)} — \\ — x^{(3)} — \\ — x^{(m)} — \end{bmatrix}$ ↱ each row constitutes an example

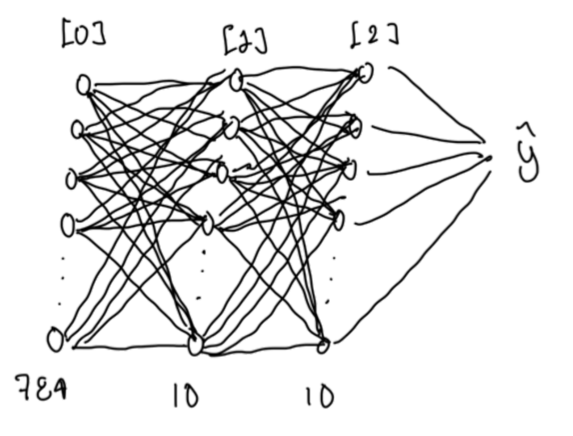
however, I want to work with columns, so I will traspose it $A^T = \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} & x^{(m)} \\ | & | & | & | \end{bmatrix}$ ↱ Each column is an example, so each column has 784 rows corresponding to each pixel.

→ The neural network will have 2 layers

↳ 0th : 784 nodes (input layer)

↳ 1st : 10 units (hidden layer)

↳ 2nd : 10 units (output layer) → digit



1) forward Propagation

→ Taking an image and running it trough the network and computing the output

$A^{[0]} = X \quad (784 \times m) \rightarrow$ Input layer

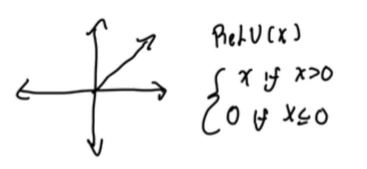
Unactivated 1st layer $\leftarrow \underset{10\times m}{Z^{[1]}} = \underset{10\times 784}{W^{[1]}}\,\underset{784\times m}{A^{[0]}} + \underset{10\times 1}{b^{[1]}}$ { This basically means to multiply the input by the weight of each connection to the 1st layer. and add a bias

$W^{[1]}$ ↓ Weight; $b^{[1]}$ ↓ Bias

Now we need to apply an activation function, otherwise each node would just be a linear combination of the nodes before it plus a bias. without a activation function it would just be a linear regression.

$A^{[1]} = g(Z^{[1]}) = \text{ReLU}(Z^{[1]})$

↓ (Rectified Linear Unit)



$\text{ReLU}(x) \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \le 0 \end{cases}$

Now to get from layer 1 to layer 2 first we add the weights on the bias

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$ { multiply the 1st layer by the weight of each connection to the 2nd layer and add a bias

→ Now the second activation function will be Softmax

$A^{[2]} = \text{Softmax}(Z^{[2]})$

↳ This will give a probability (between 0 and 1)

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

2) Backwards Propagation (Back Prop.)

→ Backwards propagation serves as a way to adjust the weights and biases.

→ It is the opposite process to forward propagation, so it starts with a prediction and finds how much the prediction diviated from the actual label (so instead of giving a success probability it gives an epsilon (error rate)) So it is possible to see how much did the previus weights and biases contributed to the actual error, and then adjust them accordingly.

$\underset{10\times m}{dZ^{[2]}} = \underset{10\times m}{A^{[2]}} - \underset{10\times m}{Y}$ → this Y will need to be one-hot encoded

↳ Error of the 2nd layer; ↳ Predictions; ↳ Actual label

$\underset{10\times 10}{dW^{[2]}} = \frac{1}{m}\,\underset{10\times m}{dZ^{[2]}}\,\underset{m\times 10}{A^{[1]T}}$

↳ Avarage of the absolute error, how much the 2nd layer was off by from the prediction, and how much should the weights and bias of the 2nd layer be adjusted

$\underset{10\times 1}{db^{[2]}} = \frac{1}{m}\sum \underset{10\times 1}{dZ^{[2]}}$

$\underset{10\times m}{dZ^{[1]}} = \underset{10\times 10}{W^{[2]T}}\,\underset{10\times m}{dZ^{[2]}} \cdot \underset{10\times m}{g'(Z^{[1]})}$

↳ How much was the hidden layer off by, does propagation in reverse → Error from the second layer and aplaying the weights to it in reverse to get to the 1st layer, and then undoing the activation function (by it's derivative) to get the proper error for the 1st layer

$\underset{10\times 784}{dW^{[1]}} = \frac{1}{m}\,\underset{10\times m}{dZ^{[1]}}\,\underset{m\times 784}{X^T}$

↳ " "

$\underset{10\times 1}{db^{[1]}} = \frac{1}{m}\sum dZ^{[1]}$

3) Then, the parameters are updated:

$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$
$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$
$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$
$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$

} Where alpha (α) is the learning rate and is a hyper parameter → Means it is not trained by the model, when the cicle is excecuted (when gradient descent is applied) alpha is set by whoever excecutes the code.

4) go back to forward propagation