How to vectorize the equations that we have derived so far so that we can exploit parallelism available due to GPU ?

Instead of passing one example through the network we want to pass one batch of examples so that we get prediction for everything at same time and we can propagate all the gradients back at the same time.

Instead of computing gradient wrt one example we are going to do vectorization for m examples.

We will see what changes in equation we have to make to make this work ?

For 1 Example	For m Examples
Error in output layer ⇝ $\delta^L = (a^L - y^L)$	$\delta^L = a^L - y^L$
hidden layer ⇝ $\delta^l = (w^{l+1}\ \delta^{l+1}) \odot \sigma'(z^l)$	$\delta^l = (\delta^{l+1}, w^{l+1^T}) \odot \sigma'(z^l)$
updation of weights and biases: $\frac{\partial L}{\partial b^l} = \delta^l$	$\frac{\partial L}{\partial b^l} = \frac{1}{m}$ np.sum $(\delta^l, axis = 0)$
$\frac{\partial L}{\partial w^l} = a^{l-1}(\delta^l)^T$	$\frac{\partial L}{\partial w^l} = a^{l-1^T} \cdot \delta^l$

In 1 example, each example is a column vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix}_{(n,1)}$$

$$x = \begin{bmatrix} \text{---} & x^{(1)} & \text{---} \\ \text{---} & x^{(2)} & \text{---} \\ & \vdots & \\ \text{---} & x^{(m)} & \text{---} \end{bmatrix}_{m \times n}$$

Error in output layer:

$a^{(i)}$

$a_L = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_c \end{bmatrix}$

Then you take argmax over these to find out which class the input belongs to

activation vector (if you have c units in last layer)

m examples and each example has n features

activation values for 1^{st} example

$a_L = \begin{bmatrix} \_\_ a^{(1)} \_\_ \\ \_\_ a^{(2)} \_\_ \\ \_\_ a^{(i)} \_\_ \\ \_\_ a^{(m)} \_\_ \end{bmatrix}_{m \times c}$

m : no. of example
c : no. of output classes / no. of units in output layer

Everything which was a column vector earlier will now become a row vector.

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ ← Cat, ← Dog, ← Horse

$y = \begin{bmatrix} 0\,0\,0\,1\,0\,0\,0 \\ \_\_\_\_\_ \\ \_\_\_\_\_ \\ \\ \_\_\_\_\_ \end{bmatrix}_{m \times c}$

We can easily do $a_L - y$ which will give $m \times c$ matrix.

δ^L will be $m \times c$ matrix

Updation of biases:

$b^l = \begin{bmatrix} b_1^l \\ b_2^l \\ b_3^l \end{bmatrix}$ $\delta^l = \begin{bmatrix} \delta b_1 \\ \delta b_2 \\ \delta b_3 \end{bmatrix}$

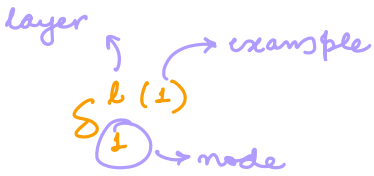
Each nodes has bias b_1, b_2 and b_3

$b^l = b^l - \eta\, \delta^l$

Error wrt 1^{st} example

$\begin{bmatrix} \_\_ \delta^{l(1)} \_\_ \\ \_\_ \delta^{l(2)} \_\_ \\ \vdots \\ \_\_ \delta^{l(m)} \_\_ \end{bmatrix}$

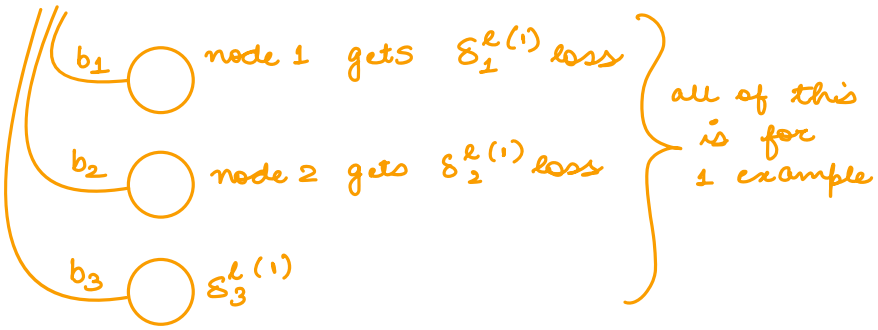
Error wrt m^{th} example



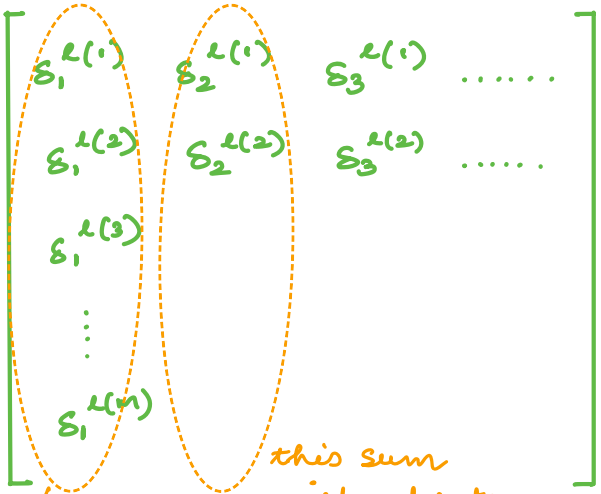
When you apply gradient descent and you want to update b_1:

$$b_1^{l} = b_1^{l} - \eta \sum_{i=1}^{m} \frac{\partial L}{\partial b_1^{l(i)}}$$

we want to keep hidden unit (subscript) same and iterate over all examples



If there are more examples then you will have $\delta_1^{l(2)}$, $\delta_2^{l(2)}$ .....



$$\frac{\partial L}{\partial b^{l}} = \frac{1}{m} \; np.sum\left(\delta^{l}, axis=0\right)$$

In matrix notation we will use this formula

taking average

along columns

we are finding average gradient along bias

Updation of weights

w^l will have a shape of h_{l-1}, h_l

- h_{l-1}: hidden unit in previous layer
- h_l: hidden unit in current layer

$$\frac{\partial L}{\partial w^l} = a^{l-1^T} \delta^l$$

$$\begin{bmatrix} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix} \begin{bmatrix} \\ \\ \end{bmatrix}$$

$(h^{l-1}, h^l) \quad (h^{l-1}, m) \quad (m, h^l)$

$$\begin{bmatrix} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix} \begin{bmatrix} \\ \end{bmatrix}$$

$(h_{l-1}, h_l) \quad (h_{l-1}, 1) \quad (1, h_l)$

$$\frac{\partial L}{\partial w^l} = a^{l-1} (\delta^l)^T$$