1. Given model : neural net with linear activation of arbitrary depth.

Let the number of layers be n.

Since, we have linear activation

$$a_1 = w_1 x + b_1$$

[where w_i is the weight matrix for layer i
b_i " " bias vector " " i]

$$\Rightarrow a_2 = w_2 a_1 + b_2$$

$$= w_2(w_1 x + b_1) + b_2$$

$$= w_2 w_1 x + (w_2 b_1 + b_2)$$

$\Rightarrow$ a_2 is of the form $wx + b$

similarly $a_n = wx + b$ for some w, b

$\Rightarrow$ neural net behaves like a linear classifier.

We can see that this similar to using SVM with linear kernel.

Since XOR is not linearly separable.

Our, neural net will not be able to classify.

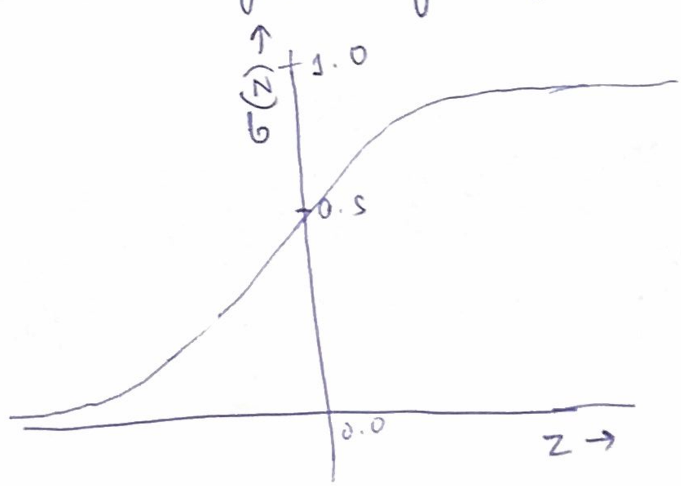
2. $\sigma(z) = \frac{1}{1+e^{-z}}$

We know that $x_i \in [0,1000]$

$\Rightarrow \bar{z} = w\bar{x} + \bar{b}$

will have a higher magnitude since $\bar{z} \propto \bar{x}$

The graph of $\sigma(z)$ is as follows



$\therefore \forall z$ with high magnitude

$\sigma(z) \rightarrow 1$

$\Rightarrow \sigma'(z) = \sigma(z)(\sigma(z) - 1)$

approaches zero.

We know that $\delta^L = \nabla_a C_x \odot \sigma'(z)$

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

$\approx$ Since $\sigma'(z)$ approaches zero it stagnates learning.

Another problem this poses is the overflow problem.

for high z, e^z is very large and we might not be able to handle it.

Using ReLu function in the hidden layer

ReLu being a linear function

does not stagnate learning.

Thus it definitely betters the problem.

However we see $f'(z)$ for ReLu

$$= \begin{cases} 1 & \forall z > 0 \\ 0 & \text{else} \end{cases}$$

$\Rightarrow$ ReLu will not be able to handle the big range of [0, 100] since gives the same derivative for the complete range.

Pre-processing required:

To overcome sigmoid problem:

① We can scale the data to a smaller range → like [0, 1]

To overcome ReLu problem:

② We can normalize the data about its mean to a range like [-1, 1]

3. In quadratic cost

$$C_x = \frac{1}{2}\| A_x^L - y\|^2$$

$$\Rightarrow \nabla_a C_x = \| A_x^L - y\|$$

So. according to BP1,

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

$$= \| A_x^L - y\| \odot \sigma'(z)$$

In this expression, $\sigma'(z)$ slows down the learning as $\sigma'(z)$ nears 0 as z approaches maxima/minima.

$\Rightarrow$ If we are able to remove this term, we will be able to fasten the learning process.

This is exactly what the cross entropy cost function does.

Cross- entropy cost:

$$C_x = -(y \ln a_x^L + (1-y)\ln(1-a_x^L))$$

$$\Rightarrow \nabla_a C_x = -\left[\frac{y}{a_x^L} + \frac{(1-y)}{(1-a_x^L)}\cdot(-1)\right]$$

$$= \frac{1-y}{1-a_x^L} - \frac{y}{a_x^L} = \frac{a_x^L(1-y) - y(1-a_x^L)}{a_x^L(1-a_x^L)}$$

$$\Rightarrow \nabla_a C_a = \frac{a_x^L - y a_x^L - y + y a_x^L}{a_x^L (1 - a_x^L)}$$

$$= \frac{a_x^L - y}{\sigma(z)[1 - \sigma(z)]}$$

$$= \frac{a_x^L - y}{\sigma'(z)} \qquad [\because \sigma'(z) = \sigma(z)(1 - \sigma(z))]$$

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

$$= \frac{a_x^L - y}{\cancel{\sigma'(z)}} \cdot \cancel{\sigma'(z)}$$

$$= a_x^L - y$$

Hence, cross-entropy fastens the learning process