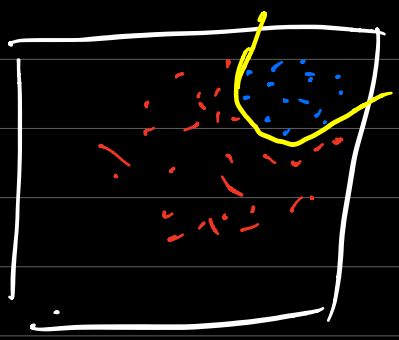
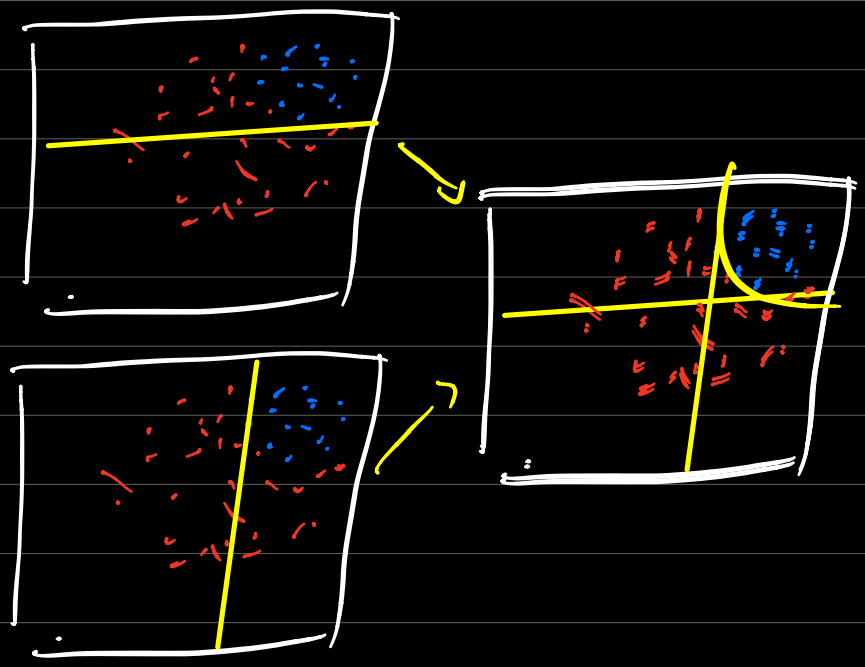
Non - linearly separable data



We can't draw a line to separate them

We use a simple trick:

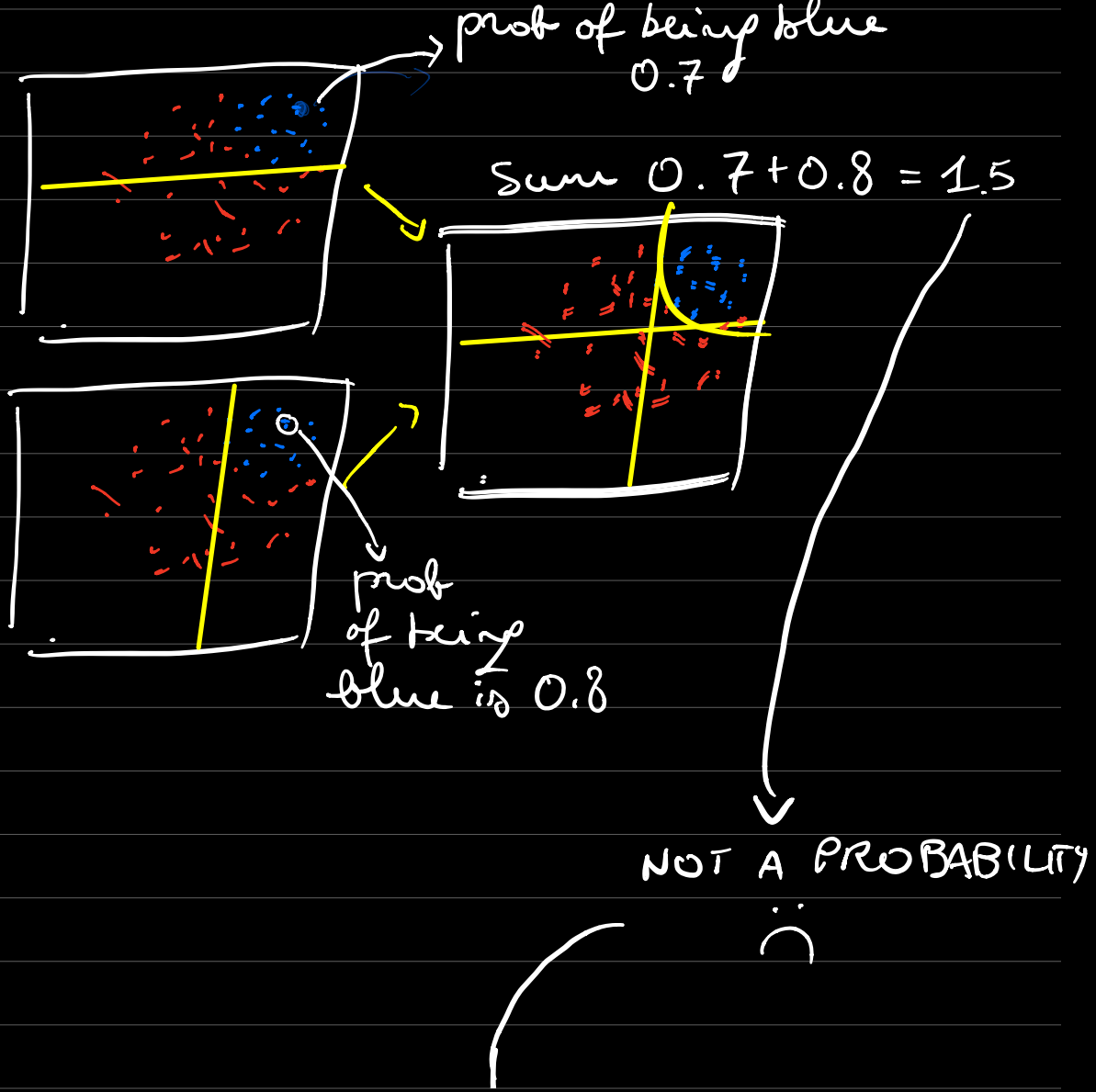
- We combine two linear models into a non linear one!



If we overimpose the two linear models we kind of getting the desired

one

How do we combine them?
We know that our linear models assign a probability to each point of being either blue or red.

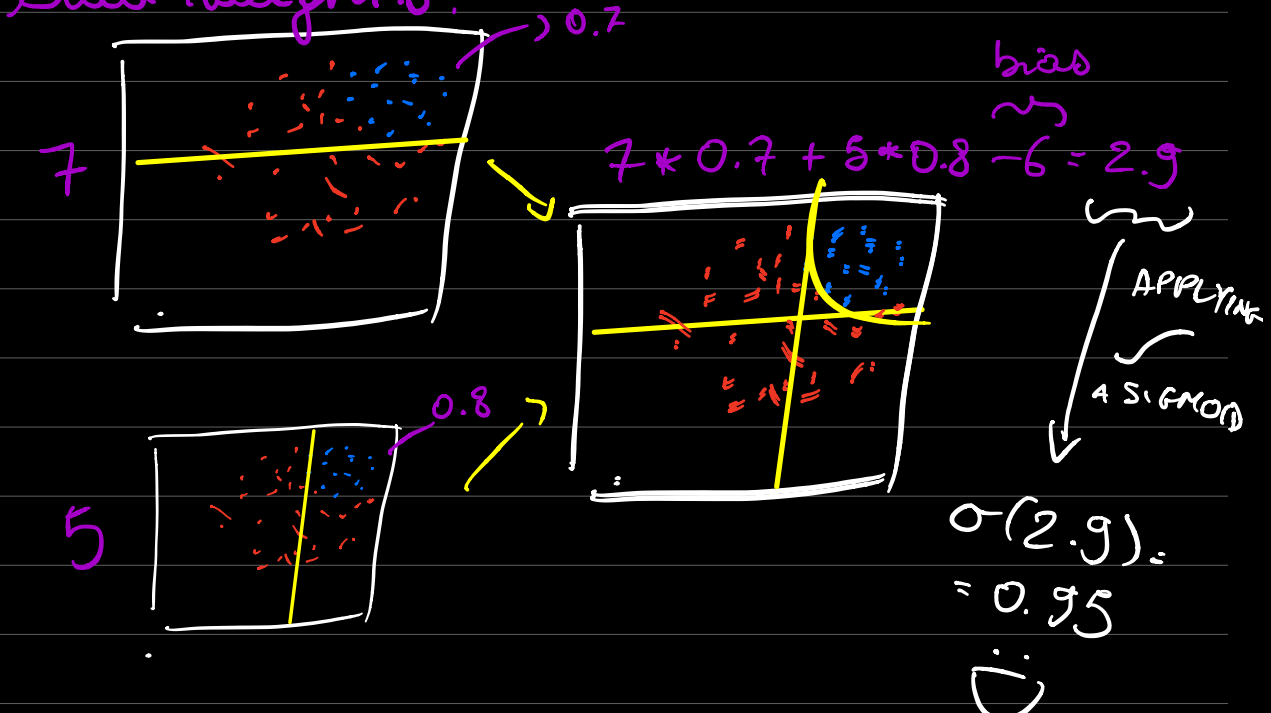


We can use a sigmoid ☺

$$\sigma(1.5) = \frac{1}{1 - e^{-1.5}} = 0.82$$

What if we want to give more weights to the upper linear model than the bottom one?

Add weights!

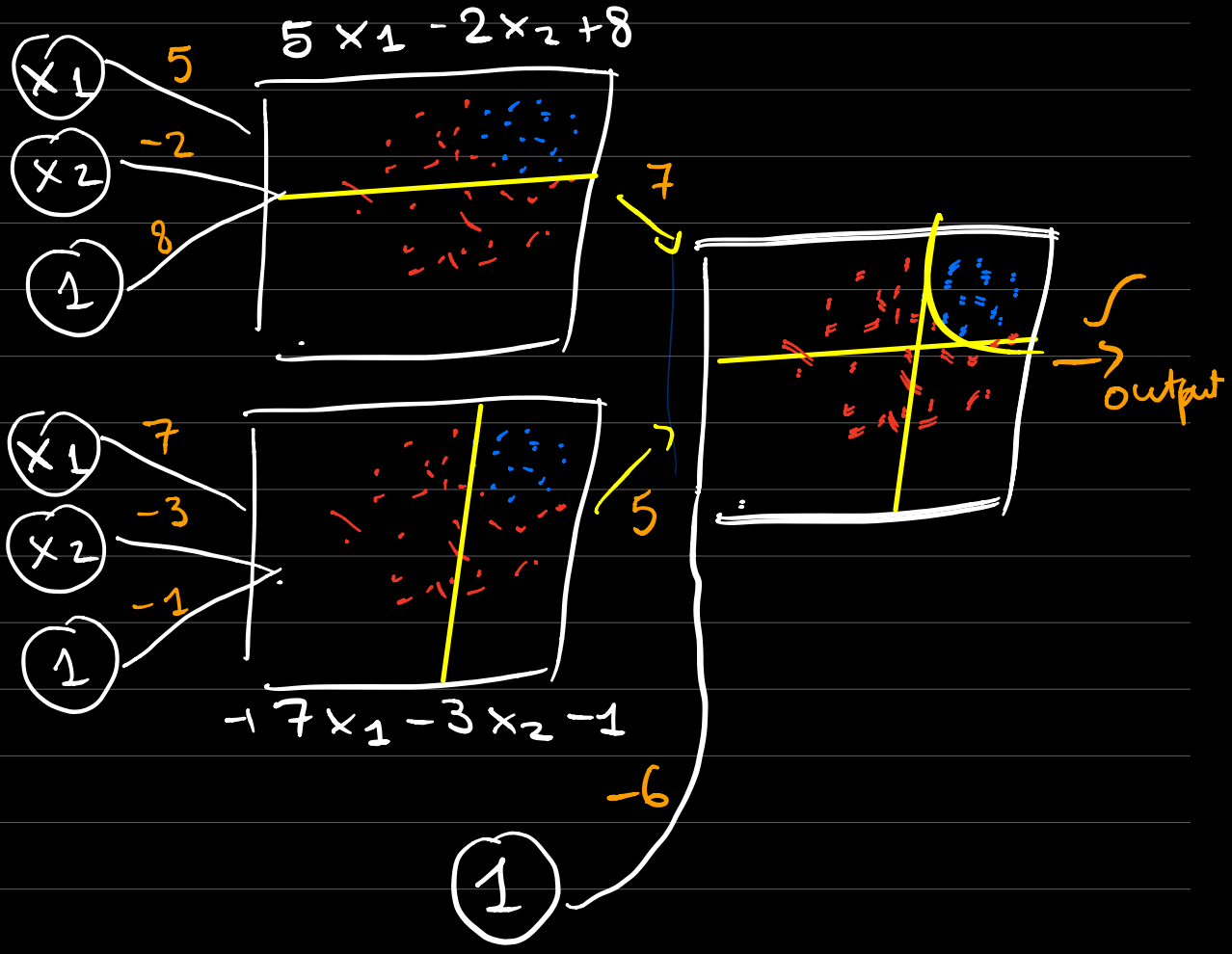


This is where we start to build a Neural Network.

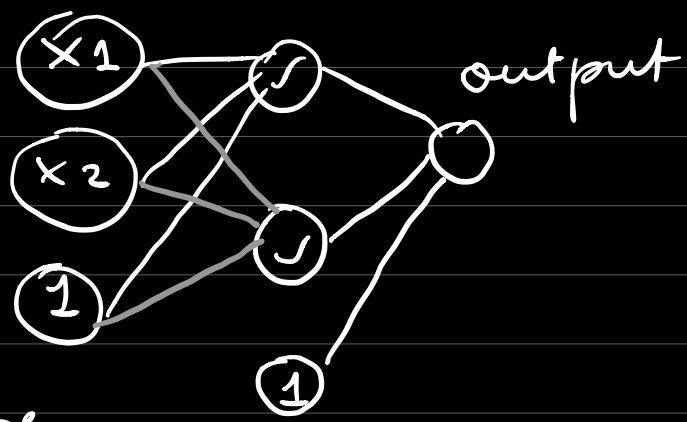
And as you could have noticed, we

did the same thing we did already with perceptrons, but using as input of the new perceptron the output of the previous ones.

Let's go into more Math, using the example of above



We represents neural networks in the following way

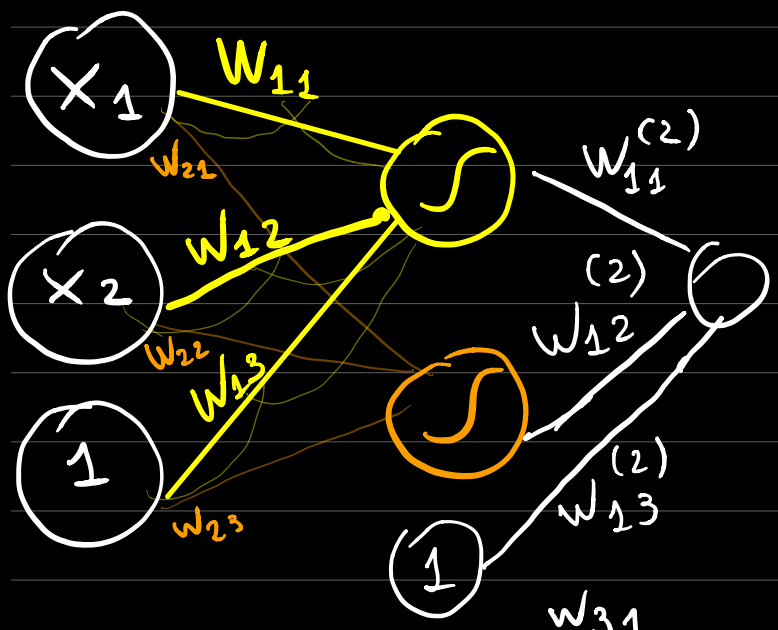


In training a neural network there are two main steps:
Feed forward, forward pass
Backword, Backpropagation

Feed forward

input ⟶ output

⚠ MATH ALERT ⚠



$$w_{31}$$

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

$w^{(1)}$

w^{2}

$$\begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{pmatrix} \begin{pmatrix} w_{11}x_1 + w_{12}x_2 + \underbrace{w_{13}\cdot 1}_{\text{bias}} \\ w_{21}x_1 + w_{22}x_2 + \underbrace{w_{23}\cdot 1}_{\text{bias}} \end{pmatrix}$$

$$1$$

As for the perceptron, our Neural Network will produce an error too.

How did the ERROR worked for the Perceptron?

So for each data point

(x_i, y_i)

$(x_1, x_2, \dots, x_m)$ label (0 or 1)

we computed a score by doing

$$0 < \underbrace{\sigma(Wx + b)}_{\hat{y}} < 1$$

$\hat{y}$ → gives a probability.

What we want is to maximize the probability of being correct, that is minimize the probability of being wrong.

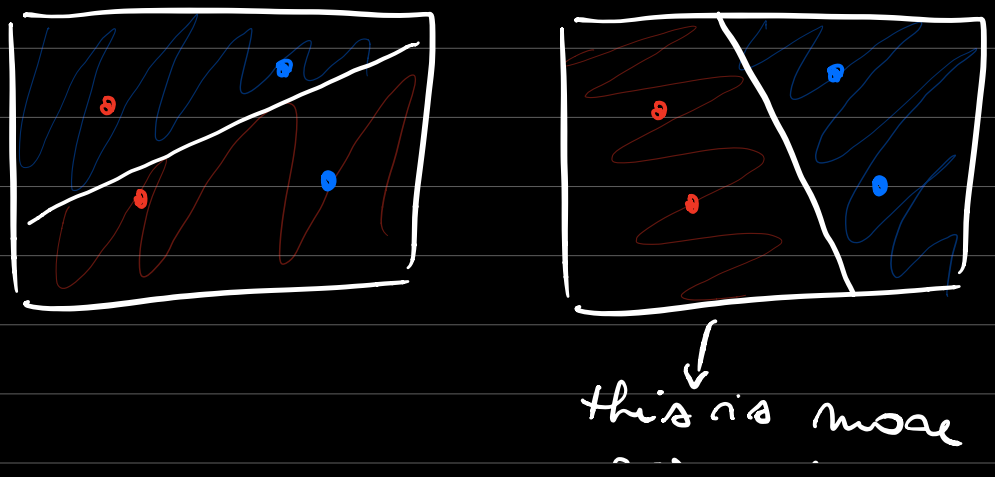
Let's start with an example:

Two models give me the probability of being accepted at the University:

- One says 80% chance
- The other 55% "

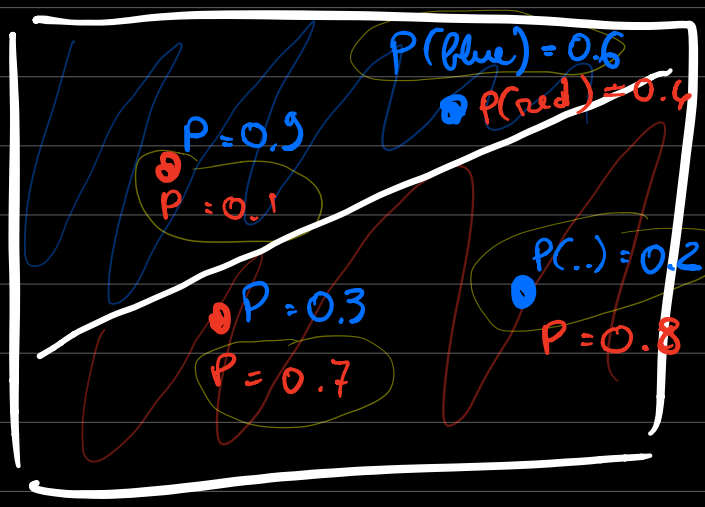
Which is better?
Well, if I actually get accepted, the first one.
So the one it is more likely to predict the correct answer.
This is called

MAXIMUM LIKELIHOOD.



correct

From a probability perspective

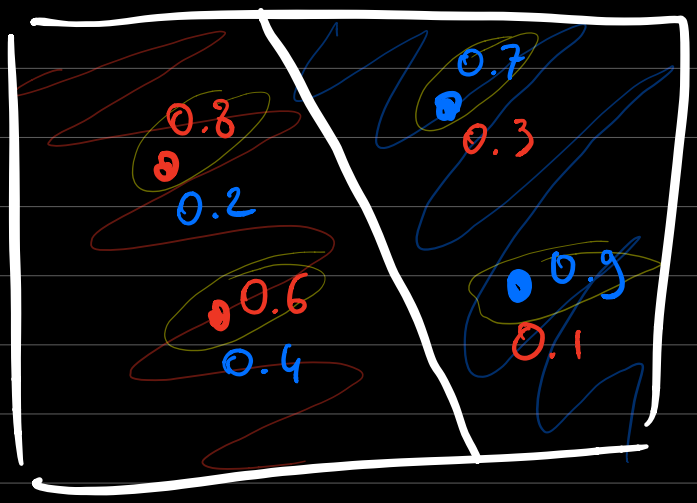


P(blue) = σ(W x + b)

since the color of the points are INDEPENDENT EVENTS, then the probability of the model to guess the right label is the product:

$$
\begin{aligned}
& P(\text{red}) = 0.1 \\
\times\ & P(\text{blue}) = 0.6 \\
\times\ & P(\text{red}) = 0.7 \\
\times\ & P(\text{blue}) = 0.2 \\
\hline
& P(\text{all}) = 0.0084
\end{aligned}
$$

→ Prob of being correct for all



$0.7 \times 0.9 \times 0.8 \times 0.6$

$P(\text{all}) = 0.3024$

greater than the other

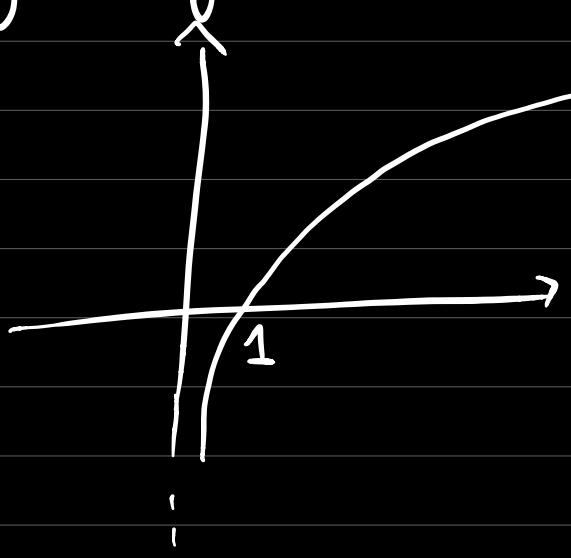
MINIMIZING THE ERROR FUNCTION IS EQUIVALENT TO MAXIMIZING THE PROBABILITY OF BEING CORRECT.

Problem: the probability we are computing requires products.

harder to compute

Transform them into sums by

using $\log(ab) = \log(a)\log(b)$



So instead of having

$$0.6 \times 0.2 \times 0.1 \times 0.7 = 0.0084$$

$$\log(0.6) + \log(0.2) + \log(0.1) + \log(0.7)$$

$$-0.51 \quad -1.61 \quad -2.3 \quad -0.36 = -4.8$$

The negative of this sum is called

CROSS-ENTROPY!

So a good model will give a

LOWER CROSS ENTROPY

(the other example gives 1.2)

So the triad is:

INPUT x, if label y is Blue
$P(x \text{ is Blue}) = \hat{y}$
$\text{Error} = -\ln(\hat{y})$

INPUT x, if label y is Red
$P(x \text{ is Red}) = 1 - P(x \text{ is Blue}) = 1 - \hat{y}$
$\text{Error} = -\ln(1 - \hat{y})$

We can combine these things

$$\text{Error} = (1 - y)(\ln(1 - \hat{y})) - y \ln(\hat{y})$$

if y is 1,
$1 - 1 = 0$
so we use

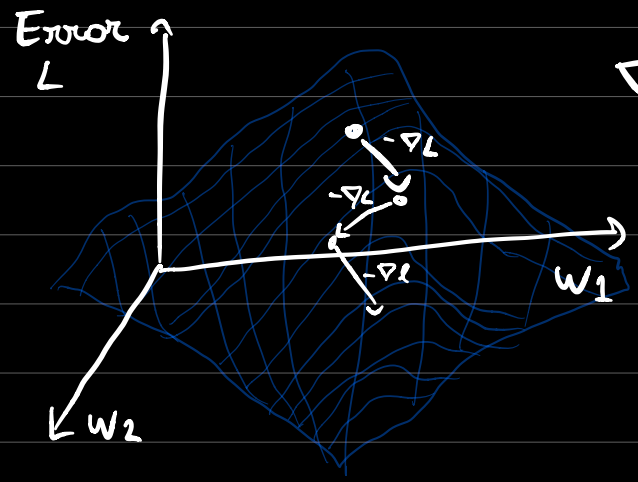
if the label is 0, this is not used

The total error will be the mean for each of the data points

$$\text{Error} = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)(\ln(1-\hat{y}_i)) + y_i \ln(\hat{y}_i)$$

How do we minimize the ERROR?

Gradient Descent



$$\nabla L = \left(\frac{\partial L}{w_1}, \frac{\partial L}{w_2}\right)$$

∇L : tells us the direction in which the function increases

$-\nabla L$ = tells us the direction in which the function decreases

To avoid making too big updates, we will use a **learning rate** α

So we

- Compute probability $\hat{y} = \sigma(Wx)$ where $W = [w_1, w_2, b]$ in our example
- Compute the error

$$L = (1-y)(\ln(1-\hat{y})) - y \ln(\hat{y}) =$$

$$= (1-y)(\ln(1 - \sigma(w_1 x_1 + w_2 x_2 + b)) - y \ln(\sigma(w_1 x_1 + w_2 x_2 + b))$$

- Compute the gradient

$$\nabla L = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial b}\right) =$$

$$\left(\frac{(1-y)}{1-\sigma(\dots)} \cdot -\sigma'(w_1 x_1 + w_2 x_2 + b) \cdot x_1\right) -$$

$$- y \frac{1}{\sigma(w_1 x_1 \dots)} \cdot \sigma'(w_1 x_1 + w_2 x_2 + b) \cdot x_1$$

We know that $\sigma(x) = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$

$$\Rightarrow \sigma'(x) = -(1+e^{-x})^{-2} \cdot e^{-x} \cdot (-1) = \frac{1}{(1+e^{-x})^2} \cdot e^{-x}$$

$$\Rightarrow e^{-x} \sigma^2(x) \text{ or } \sigma(x)(1-\sigma(x))$$

$$\Rightarrow -(1-y)\frac{1}{1-\sigma(\dots)} \cdot \sigma(\dots)(1-\sigma(\dots)) \cdot x_1 =$$

$$y(1-\sigma(\dots)) \cdot x_1 = x_1(\sigma(\dots) - \sigma(\dots)y =$$

$$-y + y\sigma(\dots)) = -x_1(y - \sigma(\dots))$$

Indeed our update algorithm is what we did yesterday:

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} = w_i + \alpha(y - \hat{y})x_i$$

Clap clap clap!

Let's code now!

The problem in computing the error on a neural network is that we don't

have the "target" of the internal nodes.

Chain Rule

If you have a composition of functions:

$$g(f(x))$$

↳ That means

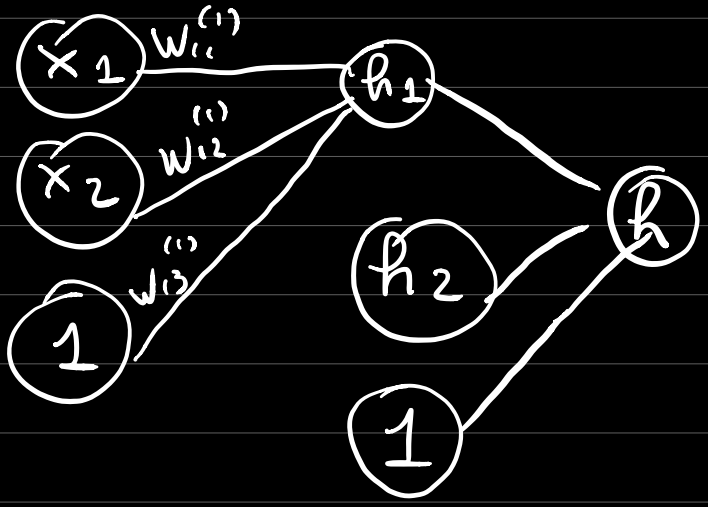
$$x \xrightarrow{f} A \xrightarrow{g} B$$

input $\quad A = f(x) \quad \hookrightarrow B = g(f(x))$

the derivative is $\dfrac{\partial g(f(x))}{\partial x} = \dfrac{\partial g}{\partial f(x)} \dfrac{\partial f}{\partial x}$

so the derivative of compositions is just a multiplication.

In a Neural Network, what we are doing is composing multiple functions



$$h_1 = W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)}$$

$$h_2 = W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)}$$

Then $h = W_{11}^{(2)} \sigma(h_1) + W_{12}^{(2)} \sigma(h_2) + W_{13}^{(2)}$

$\Rightarrow \hat{y} = \sigma(h)$

We condense it into a matrix notation

$$\hat{y} = \sigma(W^{(2)}(\sigma(W^{(1)}(x))))$$

We will compute the error

$$L(W) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \ln(\hat{y}_i) + (1-y_i)\ln(1-\hat{y}_i)\right)$$

$$\nabla L = \left(\frac{\partial L}{\partial W_{11}^{(1)}}, \dots, \frac{\partial L}{\partial W_{31}^{(2)}}\right)$$

$$\frac{\partial E}{\partial W_{11}^{(2)}} = \frac{\partial E}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h}\frac{\partial h}{\partial h_1}\frac{\partial h_1}{\partial W_{11}^{(1)}}$$

And if you will notice a nice pattern, that is:

Remember $h = W_{11}^{(2)}\sigma(h_1) + W_{12}^{(2)}\sigma(h_2) + W_{21}^{(2)}$

$$\frac{\partial h}{\partial h_1} = W_{11}^{(2)}\sigma(h_1)(1-\sigma(h_1))$$

↓

we have it from the forward pass!

In practice, by doing the forward

pass we provide already the ingredient for computing the backward pass, that will give us the partial derivative of the error w.r.t. whatever weight!