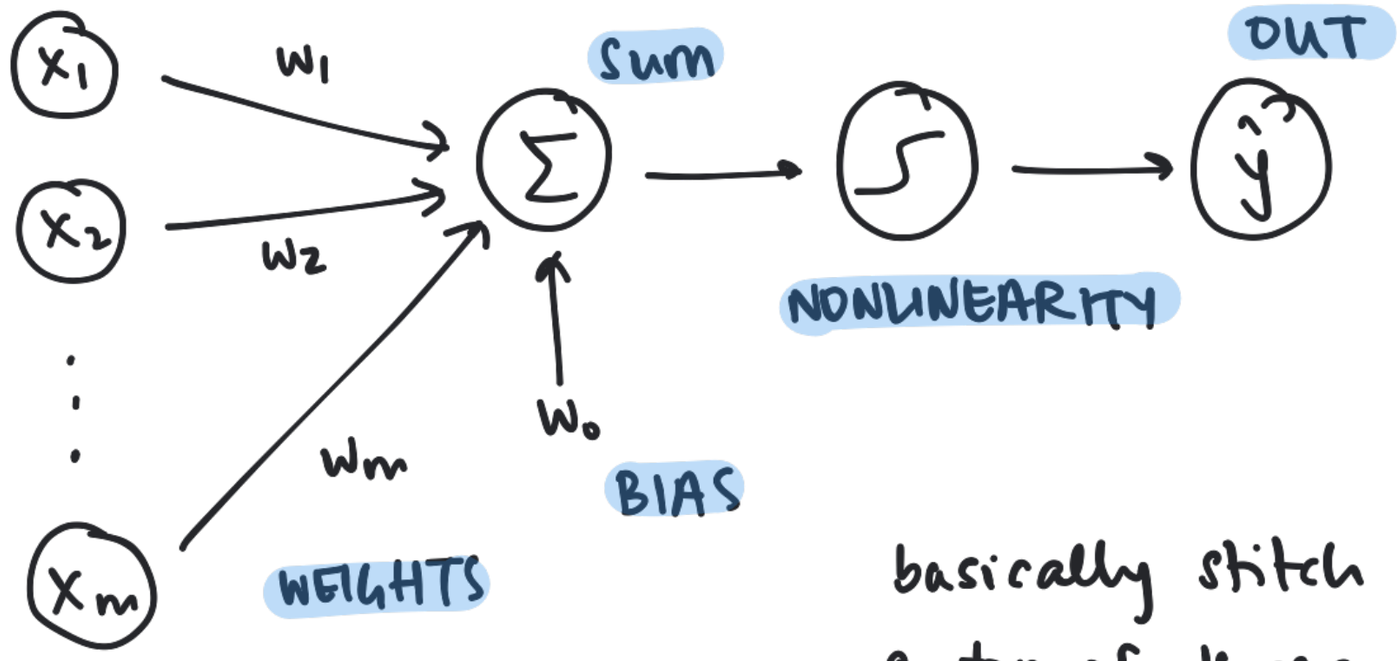
NEURAL NETS

building block is the neuron:



basically stitch a ton of these together → NEURAL NET!

(we'll do this later)

can write this mathily:

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right) \equiv g(z)$$

$$= g(w_0 + X^T W), \quad X = \begin{bmatrix} x_1 \\ \vdots \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ \vdots \end{bmatrix}$$

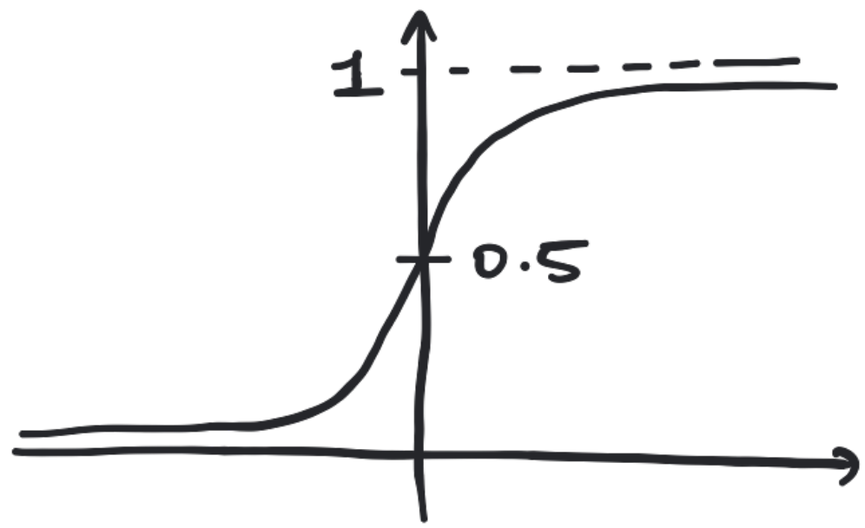
what is g?? ACTIVATION FUNCTION ⟹

a popular choice of g is the SIGMOID:

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

$\sigma : \mathbb{R} \longrightarrow (0,1)$

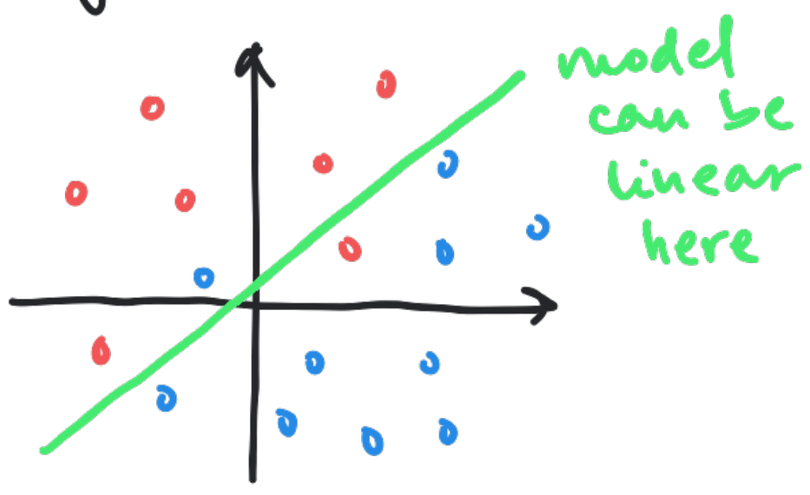
↳ can be interpreted as a probability!

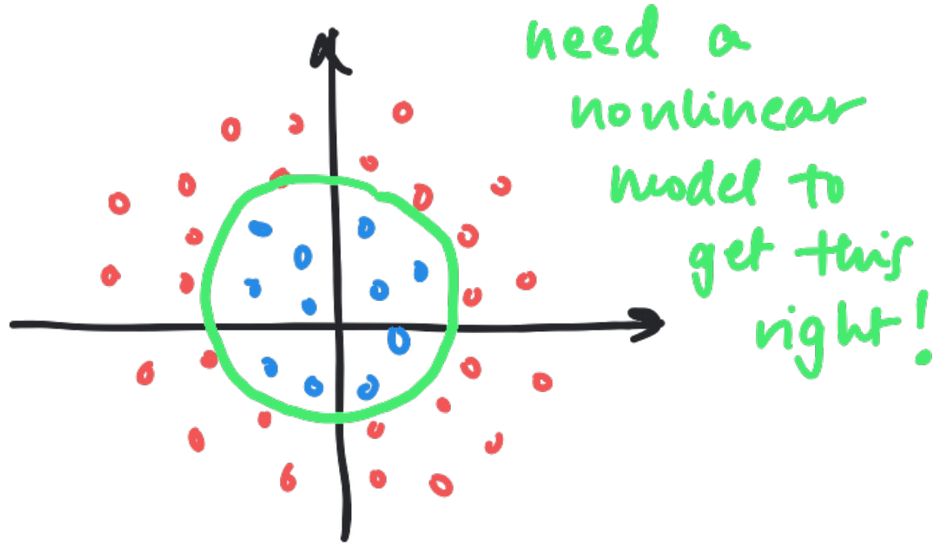


the choice of activation function is pretty important!

↳ builds essential nonlinearities into the model.

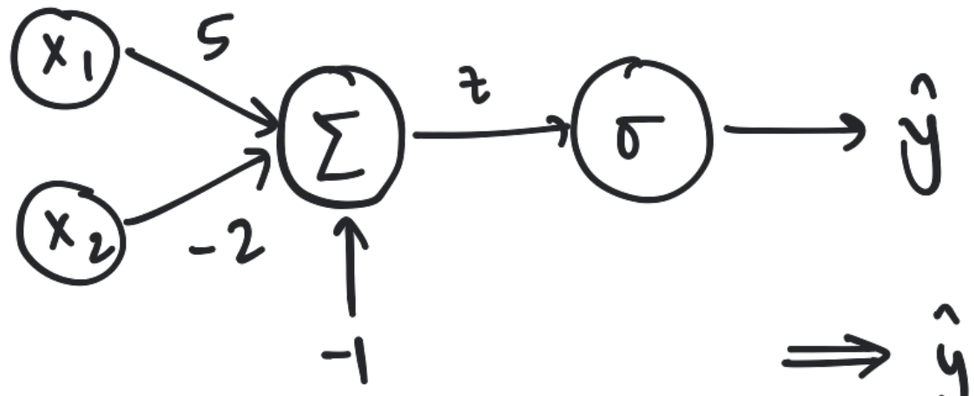
e.g. classification problems:



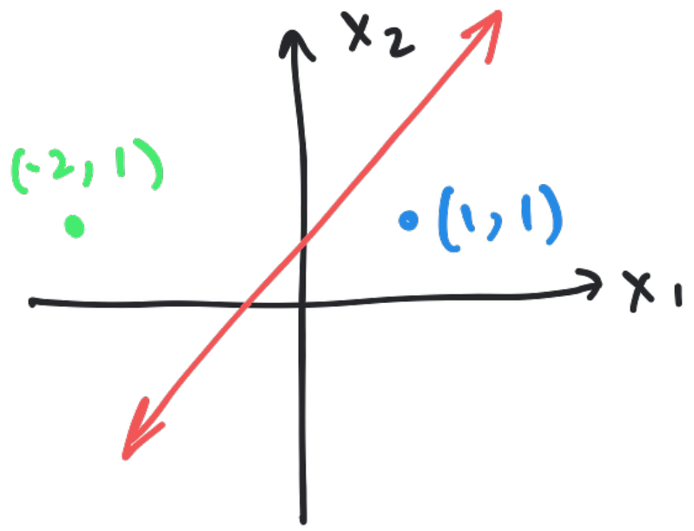


how does classification work?

Simplest case. say we have the right weights.
using just one neuron for now.



$$\Rightarrow \hat{y} = \sigma(1 + 5x_1 - 2x_2)$$



$$\hat{y} = \sigma(1 + 5 - 2)$$
$$= \sigma(4) = 0.982$$
$$> 0.5$$

$$\hat{y} = \sigma(1 - 10 - 2)$$
$$= \sigma(-11) = 1.7E5$$
$$< 0.5$$

this is a **LINEAR LOGISTIC CLASSIFIER**.

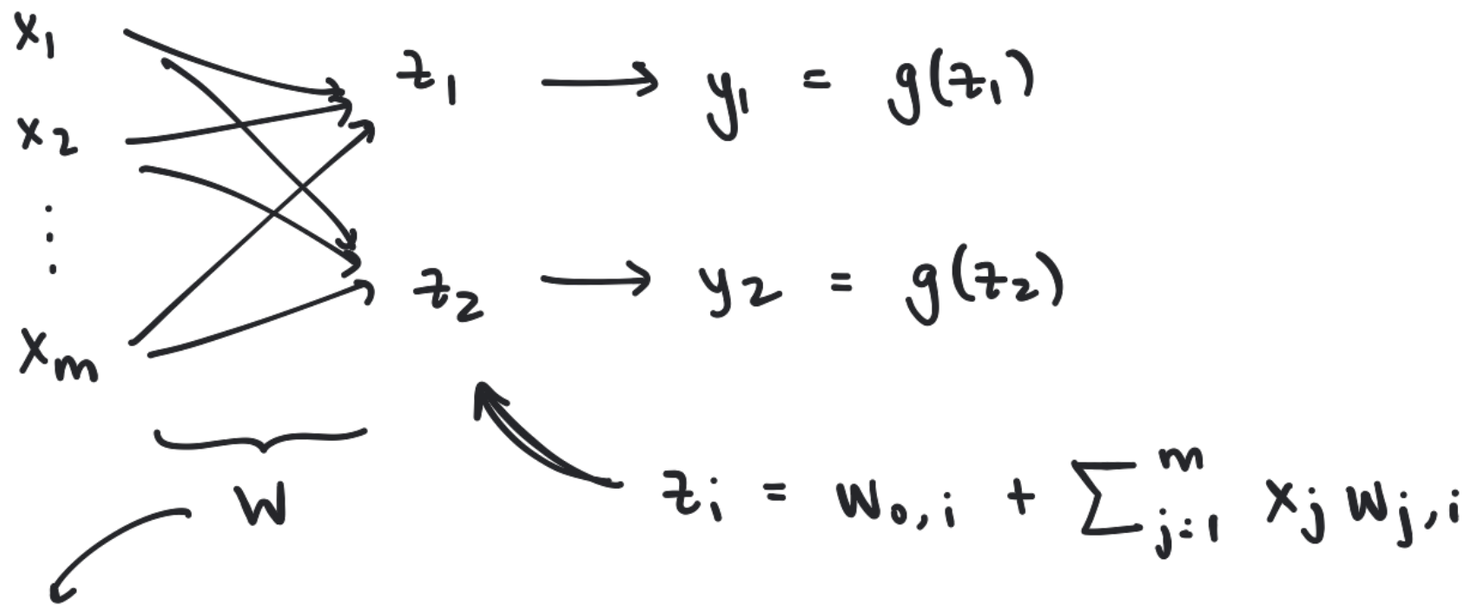
↳ predicts A if $\hat{y} > 0.5$
B if $\hat{y} < 0.5$

} we can set the threshold!
$\hat{y}$ gives us probabilities basically.

putting it together...

so that's ONE neuron. put a bunch together and you can get all kinds of fancy behavior!

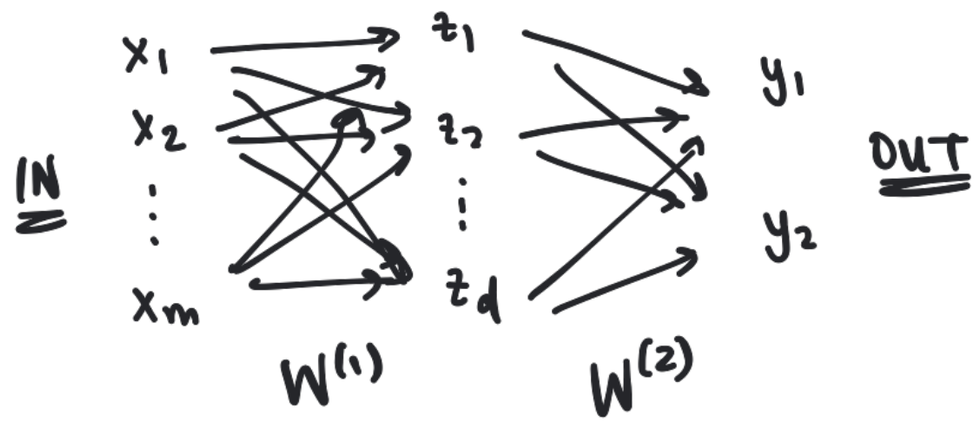
↳ hard to draw out graphically bc LOTS of dimensions.



these weights define a LAYER of the neural net.

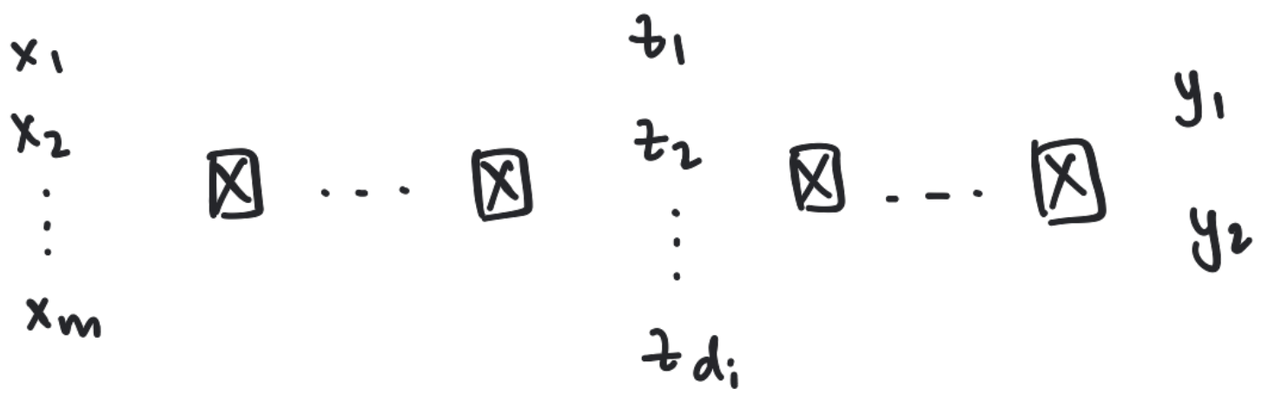
because it's so connected, call it a DENSE LAYER.

a **SINGLE LAYER NEURAL NET**:



- the z's form a **HIDDEN LAYER**.
  ↳ neither IN or OUT.
- 2 weight matrices $W^{(i)}$

can stack these! (⊠ means dense layer)



okay so you have a bunch of layers. now what???

↳ need to TRAIN the neural net to optimize W.

have to quantify how wrong its predictions are.

↳ use a **LOSS FUNCTION**.

$$\mathcal{L}\left(f(x^{(i)}; W), y^{(i)}\right)$$

$f(x, W)$: prediction

y: actual (from training data)

eg. for classification problem:

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} y^{(i)} \cdot \log\left(f(x^{(i)}; W)\right) + \left(1 - y^{(i)}\right) \cdot \log\left(1 - f(x^{(i)}; W)\right)$$

"objective function"

⇒ here, $y^{(i)}$ is 0/1. $f(x, W)$ is $[0, 1]$.

how to train your dragon

LOSS OPTIMIZATION is the goal.

↳ want to be as correct as you can* on the training data

→ want to minimize $J(W)$.

* barring overfitting!

to get: $W^* = \underset{W}{\operatorname{argmin}} J(W)$ ← best weights for the NN

can use a minimization algorithm like...

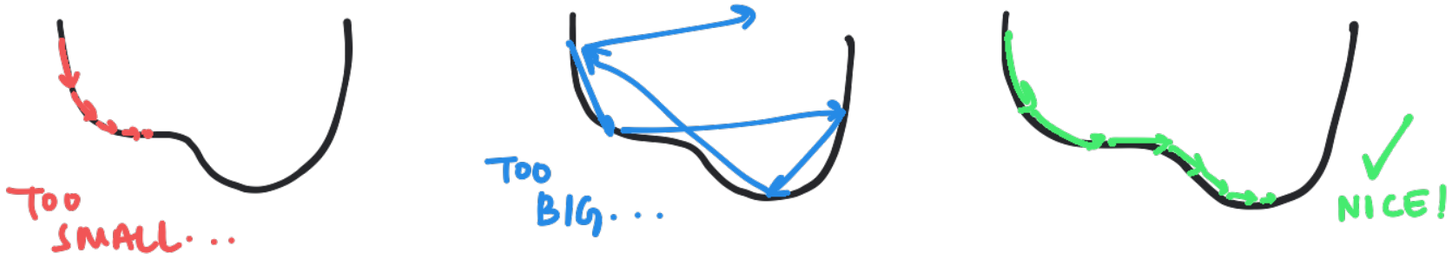
GRADIENT DESCENT

calculate gradient + update weights.

$$W_{new} = W - \eta \cdot \frac{\partial J(W)}{\partial W}$$ η: learning rate

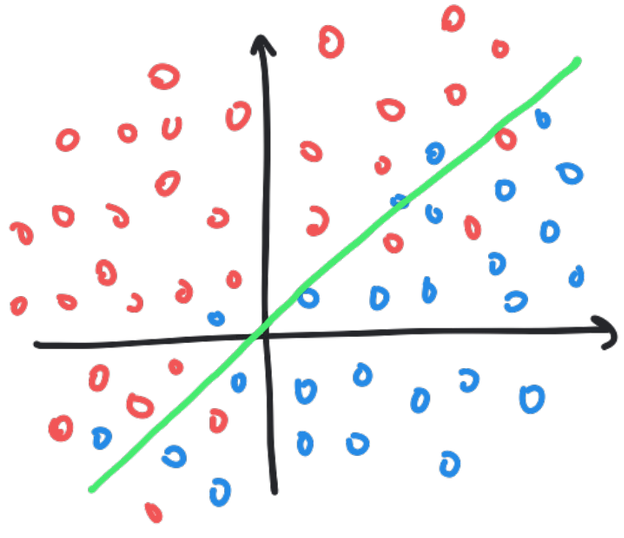
finding $\partial J / \partial W$ is nontrivial (take 036 to understand!)

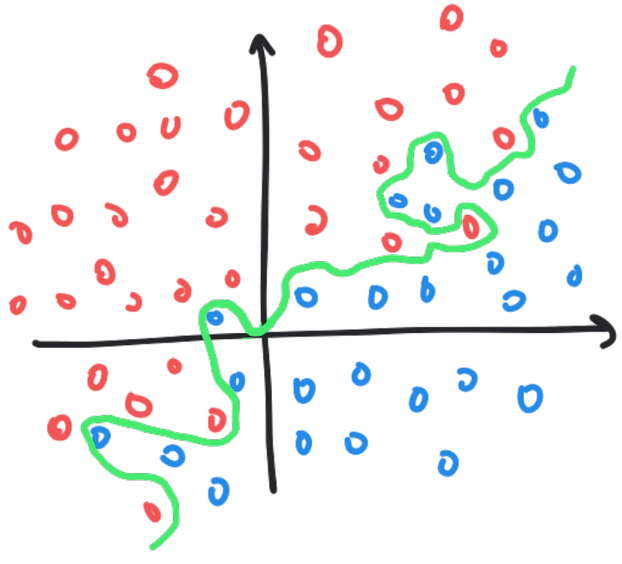
setting η can make a huge difference too!



can have adaptive learning rates that scale in various helpful ways.

overfitting ?





which looks more reasonable?

↳ want to prevent overcomplicated models.

good way to tackle this is not to overtrain.

↳ "early stopping"

× STOP TRAINING HERE!

let's check out some examples!

TO THE NOTEBOOK!