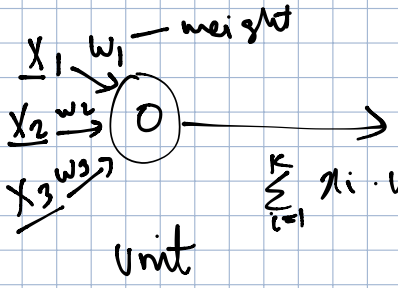
* Neural Networks

* They are computational units.

*Artificial Neural Networks



These computational units are called Perceptrons

Quiz:

x_1 1 w_1 1/2
x_2 0 w_2 3/5
x_3 −1.5 w_3 1

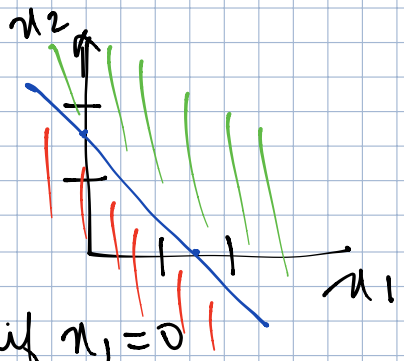
Threshold $\theta \geq 0$

activation $\Rightarrow 1 \times \frac{1}{2} + 0 \times \frac{3}{5} - 1.5 \times 1$

$\Rightarrow -1$

output = ? $y = 0$

$-1 \leq \theta$ where $\theta \geq 0$ $\therefore y = 0$ (binary output)



0, 1

weights matter

$w_1 = 1/2$

$w_2 = 3/4$

$\theta = 3/4$

return 1

return 0

if $x_1 = 0$

$x_2 = ?$

set $x_1 = 0$

$0 \times \frac{1}{2} + \square \times \frac{3}{4} = \frac{3}{4}$

solution

$\Rightarrow x_2 \, 1/2 = 3/4$

$x_2 = 3/2$

eg. same case ↑,

$x_1 \in \{0,1\}$ $w_1 = \frac{1}{2}$ $\theta = \frac{3}{4}$

$x_2 \in \{0,1\}$ $w_2 = \frac{1}{2}$

X_1	X_2		
0	0	$= 0$	$y = 0$
0	1	$= 0.5$	$y = 0$
1	0	$= 0.5$	$y = 0$
1	1	$= 1$	$y = 1$

only time $y=1$ is when X_1 & X_2 are True: conjunction AND Boolean

* This gives us a way to define a logic gate!

* we want Y to be a OR LOGIC GATE.

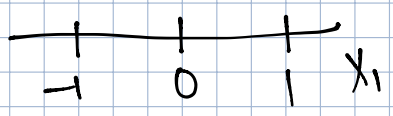
X_1	X_2		
0	0	$= 0$	$Y = 0$
0	1	$= \boxed{0.5}$	$y = 1$
1	0	$= \boxed{0.5}$	$y = 1$
1	1	$= \boxed{1}$	$y = 1$

$w_1 = \boxed{0.5}$

$w_2 = \boxed{0.5}$

$\theta = \boxed{0.5}$

* Not (one variable)



$w_1 = -1$

$\theta = 0$

* AND, OR, NIT are all expressible as preceptron units

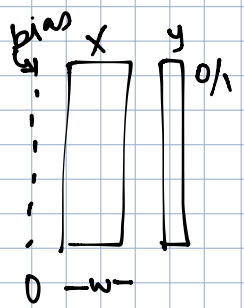
* Lets design a XOR using a network of preceptron

* XOR

OR − (AND * 2)

x_1	x_2	AND	w	XOR	OR
0	0	0	1	0	0
0	1	0	−1	1	1
1	0	0	−1	1	1
1	1	1	−2	0	1

$\theta = 0$



* Perceptron Rule

$w_i = w_i + \Delta w_i$ } we change weights by Δw_i

$\Delta w_i = \eta (y - \hat{y}) x_i$

$\hat{y} = \left(\sum_i w_i x_i \geq 0 \right)$

y : target

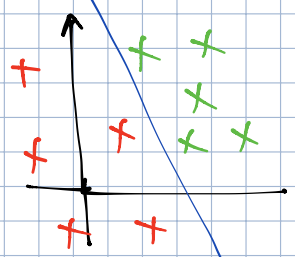
$\hat{y}$ = output

η : learning rate

x : input

only stop if it does not need to be stopped then it is not linearly seperable

y	$\hat{y}$	$y - \hat{y}$
0	0	0
0	1	−1
1	0	1
1	1	0



data is linearly seperatable in this case.

and perception rule will find it!. The algorith should only be run when data is linearly seperatable.

* Gradient descent: a more robust algorithm for non-linear seperatability
* imagine the output is not thresholded.

$a = \sum_i x_i w_i \qquad y = \{ a \geq 0 \}$

$E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a^2)$

use chain rule derivative

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{(x,y)\in D} (y-a)^2$$

$$= \sum_{(x,y)\in D} (y-a) \frac{2}{2} \frac{\partial}{\partial w_i} - \sum_i x_i w_i'$$

$$= \sum_{(x,y)\in D} (y-a)(-x_i)$$ ← looks just like perceptron rule!

* let's take a look further:

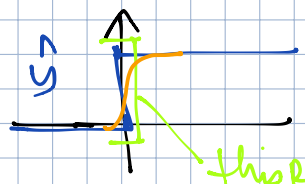
$\Delta w_i = \eta (y-\hat{y}) x_i$ perceptron : guarantee : finite convergence

$\Delta w_i = \eta (y-a) x_i$ → not thresholding : gradient descent : calculus
(y: target, a: activation)
robust to data not linearly seperatable,
converge to local optima

move the weights in the $-w$ direction.

* So why don't we use gradient descent all the time at even $\hat{y}$

* non-differentiable : it a discontinuous function



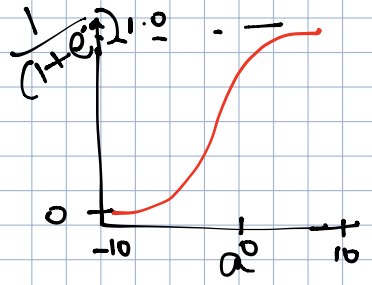
this part makes it discontinuous, lets make it smooth using sigmoid S

* Sigmoid : - differentiable threshold

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

$a \to \infty \quad \sigma(a) \to 0$

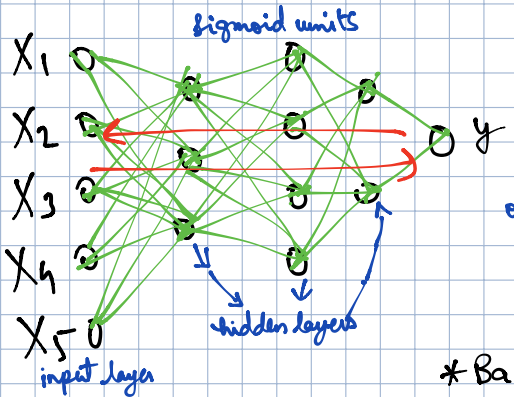
$a \to +\infty \quad \sigma(a) \to 1$



activation get to 0 ⇒ as a get negative derivatives goes to 0
when a grows in size, derivative goes towards 1. (As activation get +ve)

* Neural Networks

Sigmoid units



* This mapping from input to output is differentiable. i.t terms of weight
* we can move all the weights from input to output to get a desirable result.
* This lead to the idea of back propogation
* how moving the weigh up or down wards to produce the out put we want.
* Back Propagation : computing derivatives . Chain rule with respect to all different weights of the network.

* Many local optima : (it analogious to a perceptron since we are not doing the thresholding we trying to set the weights so the error is low, some time even after setting all the weights it makes the error worse, but in fact it gets stuck.

* Gradient descent: minimizing errors → optimizing weights

→ gradient descent
→ use advanced methods :
* momentum descent : as we are doing gradient descent, and we get stuck in an error, we continues (bounce out) forward in the direction we have be working towards

* higher order derivatives → instead of just using variation in weights

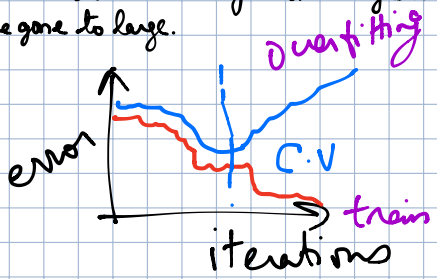
* randomized optimization : apply to make things more robust
* penality for complexity : we just dont want to minimize errors, but we want to penalize for using a a system to complex . (overfitting ex : Decision Tree growing too much, then we used pruning to filling out the tree by backing up, limiting min_samples_split)
↳ regression overfitting (order of polynominal)
↳ size of the Decision Tree
* Here in Neural Network → More and more node with make mapping more complex
→ more nodes
→ more layers
→ we can have complex network by larger numbers of the weight. So sometimes we want to penilize the network by keeping the numbers in a reasonable range

* Restriction Bias & Inductive Bias
  * Restriction Bias: tells us about the representation power of the data structure we are using ex in this case A Network of Neurons and set of hypotheses you are able to consider
* we are restricting to a subset of models

* Neural Networks
  * Restrictions:
1) we started with linear planes – Perceptrons
2) Sigmoids: much more complex
hence not much of restriction

→ it represents → Boolean function: using network of threshold-like units
→ Continuous function? → no discontinuity: we can do it with single hidden
→ Arbitrary: stitch together – two hidden layers

* There is a worry of OVERFITTING!
→ we limit it by using CV, to decide how many hidden layers to use, how many nodes to use, when to stop training when weights have gone too large.



error keeps dropping in neural networks as we perform more iterations

* Preference Bias:
  * Something about the algorithm we are to learn, why we prefer 1 over the other
  * ex: Decision Trees: we preferred nodes near the top with high information gain, correct trees, trees that were shorter to deeper
* Neural Network: How do we start the algorithm: do we start with 0 or 1?

* let's initialize the weights to something, we can't update something which is undefined.
* Small random value: since we run the algorithm multiple time, what if the algorithm gets stuck, we don't want the algorithm to get stuck there again. It gives some variability which helps avoiding local minima.
* why we start with small values → if weights get to big it sometimes lead to overfitting. (high complexity)
* So Preference Bias → small values (low complexity), we prefer simpler explanations to complex.
* we prefer correct answers, the simpler explanation is preferred Occum's Razor.

*Occum's Razor: entities should not be multiplied unnecessarily, given Neural Networks!, there there is often a lot of unnecessary multiplication. We we are not doing any better at fitting data we should not multiply further and make it more complex. Choose the simple solution.

* we get better generalization with simpler solutions in Supervised Learning.

* ReCap
  * Perceptrons → threshold units
* networks can produce any Boolean function
* perceptron rule — finite time for linearly seperable
* general differentiable rule → ( ) back Sigmoid Propogation & gradient descent
* perfernce restriction bias of neural networks

Quiz 1:

Network inputs: [1, 2, 3]

layered network weights

[1, 1, -5] & [3, -4, 2]

[2, -1]

$X_1 W_1 \Rightarrow 1 * 1 = 1$

$X_2 W_2 \Rightarrow 2 \times 1 = 2$

$X_3 W_3 \Rightarrow 3 * -5 = -15$

activation = -12

$X_1 W_1 \Rightarrow 1 * 3 = 3$

$X_2 W_2 \Rightarrow 2 * -4 = -8$

$X_3 W_3 \Rightarrow 3 * 2 = 6$

activation : 1

[-12, 1] hidden layer

$X_1 W_1 \Rightarrow -12 * 2 = -24$

$X_2 W_2 \Rightarrow 1 * -1 = -1$

[-25] //

Quiz 3:

```
[
  [input, input],
 [[3,2] , [-1, 4] , [3, -5] ],
[[1, 2 , -1]]
]
```

Input

$[a,b] * \begin{matrix}[3,2] \\ [-1,4] \\ [3,-5]\end{matrix} = \begin{matrix}[3a, 2b] \\ [-a, 4b] \\ [3a, -5b]\end{matrix}$

output

$* [1,2,-1] = \begin{matrix}[3a, 2b] \\ [-2a, 8b] \\ [-3a, 5b]\end{matrix}$

$= [-2a, 15b]$ //

Design a XOR Network

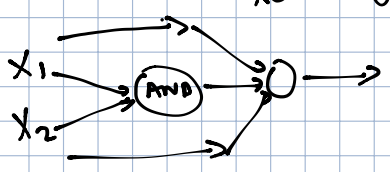
$\theta = 1$

X_1	X_1	w	Actv	AND	OR	OR-AND	XOR
0	0			0	0	0	0
1	0			0	1	1	
0	1			0	1	1	
1	1			1	1	0	

design OR & AND then OR- AND = XOR

X_1	X_2	OR w	Act	OR

XOR $\theta = 0$



OR gates Network [[3,2]
[4,5]
[2,5] , [1,1]]

input

[0,0] [[3,2] [0,0] ✓
[4,5] =
[2,5]

[0,1] [[3,2] =[[0,2] = [0,12]
[4,5] [0,5]
[2,5] [0,5]]

[1,0] [[3,2], [[3,0] [9,0]
[4,5] = [4,0] =
[2,5]] [2,0]]

[1,1] [[3,2] [3,2]
[4,5] = [4,5] = [9,12]
[2,5]] [2,5]

output

OR Gate

[0,0] * [1,1] = 0 y = 0

[0,12] * [1,1] = 12 y = 1

[9,0] * [1,1] = 9 y = 1

[9,12]] x [1,1] = 21 y = 1

θ = 8 : OR

- θ = least x_{out} - 1 @ [0,1]
[1,0]

AND $\theta = 20$ $\theta = (x_{1out} + x_{2out}) - 1$ @ [1,1]

output

[0,0] * [1,1] = 0 $y = 0$

[0,12] * [1,1] = 12 $y = 0$

[9,0] * [1,1] = 9 $y = 0$

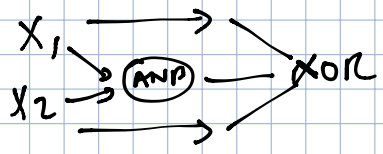
[9,12] * [1,1] = 21 $y = 1$ for AND gate if output $y = 1$ then $y = y * -2$ $\Rightarrow \therefore$ $y = -2$

XOR: $\theta = 0$

Now truth table to generate XOR

Input	OR	AND	OR-AND	Truth XOR
[0,0]	0	0	0	0
[0,1]	1	0	1	1
[1,0]	1	0	1	1
[1,1]	2	-2	0	0

XOR:



when Input

a) [0,0] = $X_1 = 0$ $X_2 = 0$

XOR = 0

b) [0,1] = $X_1 = 0$, $X_2 = 1$

XOR = 1

c) [1,0] = $X_1 = 1$, $X_2 = 0$

XOR = 1

d) [1,1] = $X_1 = 1$, $X_2 = 1$, AND = -2

= $x_1 + x_2 - AND = 1 + 1 - 2 = 0$

$\therefore$ XOR = 0