

Chapter 4: Neural Networks

* Introduction to NN → What & Why ?

Notation ←

* Feed Forward

Vectorization ←

Computation graph ← * Activation functions & non-linearity of NN → Universality

* Back-propagation

* Optimization technique: From SGD to Adam

* Batch Optimization

* TensorFlow & Pytorch

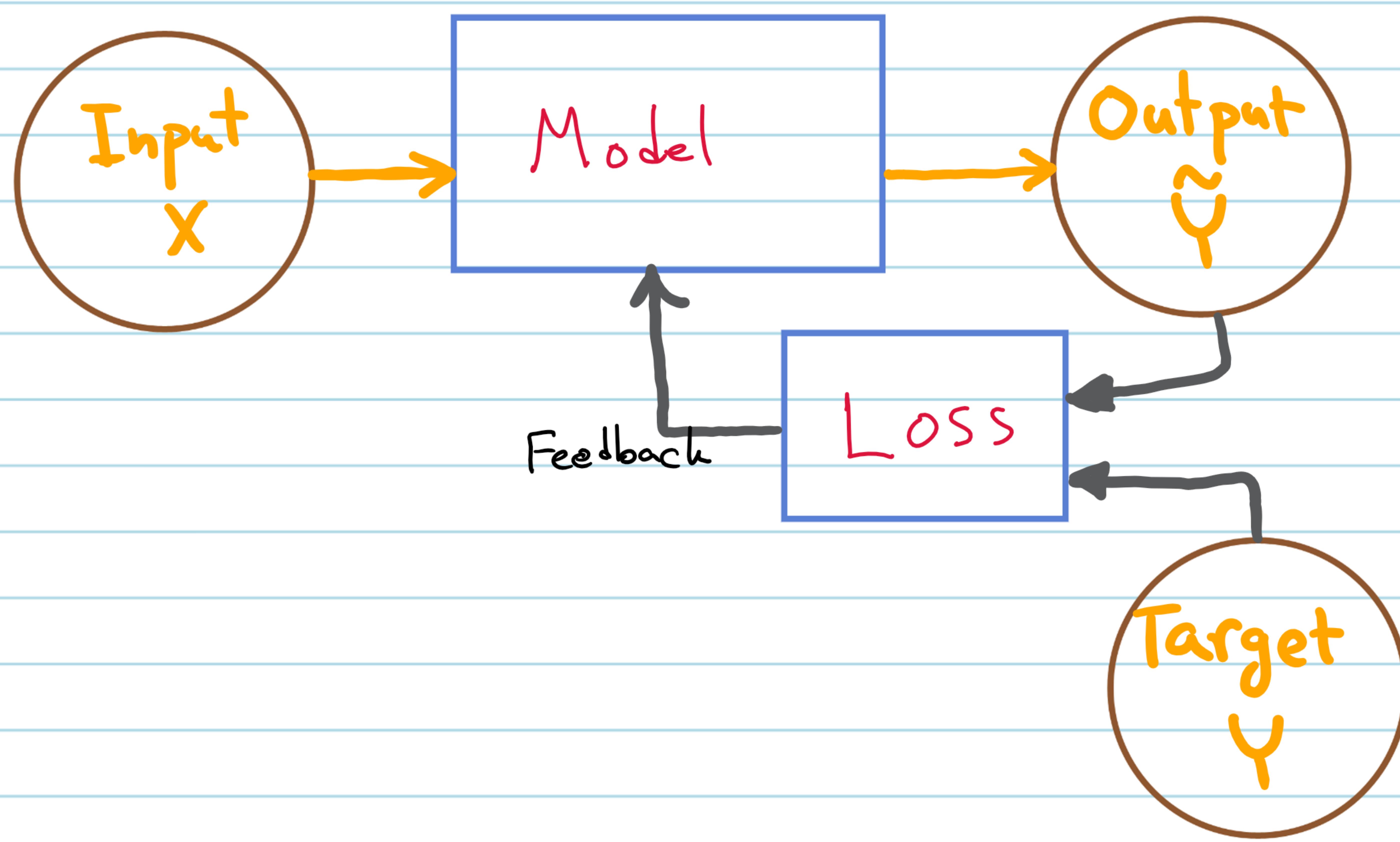
* Overfitting in NN

* Architectures

- Convolutional NN

- Recurrent NN

Reminder:



Schematic picture of classification/Regression model training process.

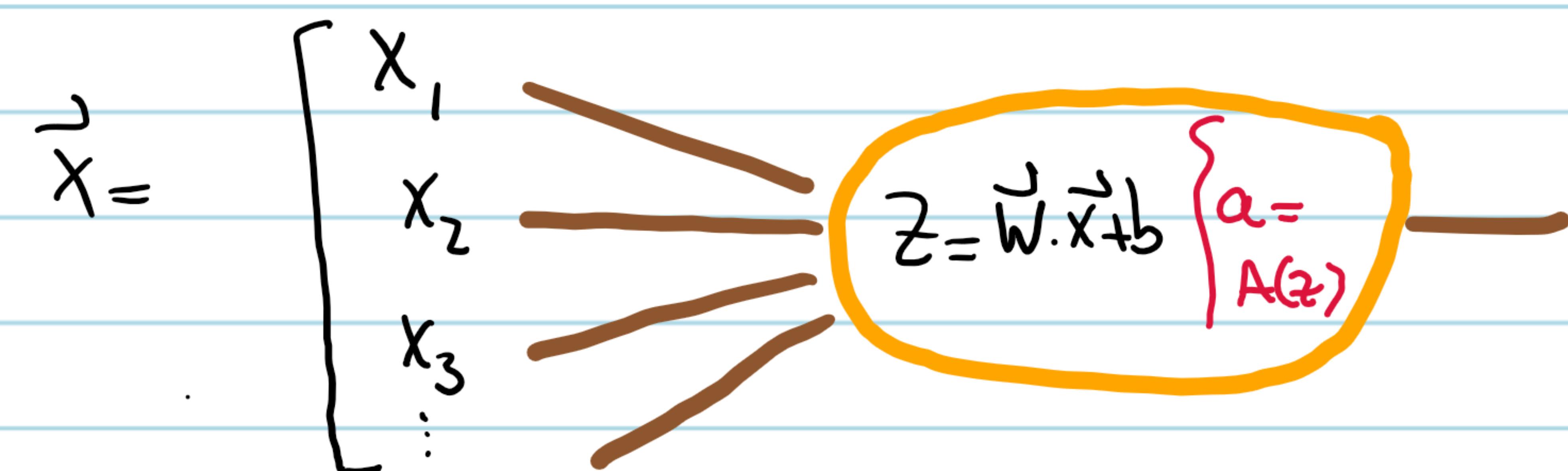
Here we want to introduce NN as a new kind of models.

These models are fairly similar to some of the models we've already seen. Specifically, one can describe them as an extension of Logistic Reg. (LR) or perceptrons.

Perceptron / LR

We already introduced logistic Reg. There's a similar model that instead of the sigmoid in LR, uses a step function

and $f(x) = (\vec{w} \cdot \vec{x} + b) \geq 0$.



This is similar to some of the models that are used to describe a neuron. Namely, it takes some inputs with different weights and if the result $\vec{w} \cdot \vec{x}$ is above some threshold, the neuron would fire, otherwise, it stays silent.

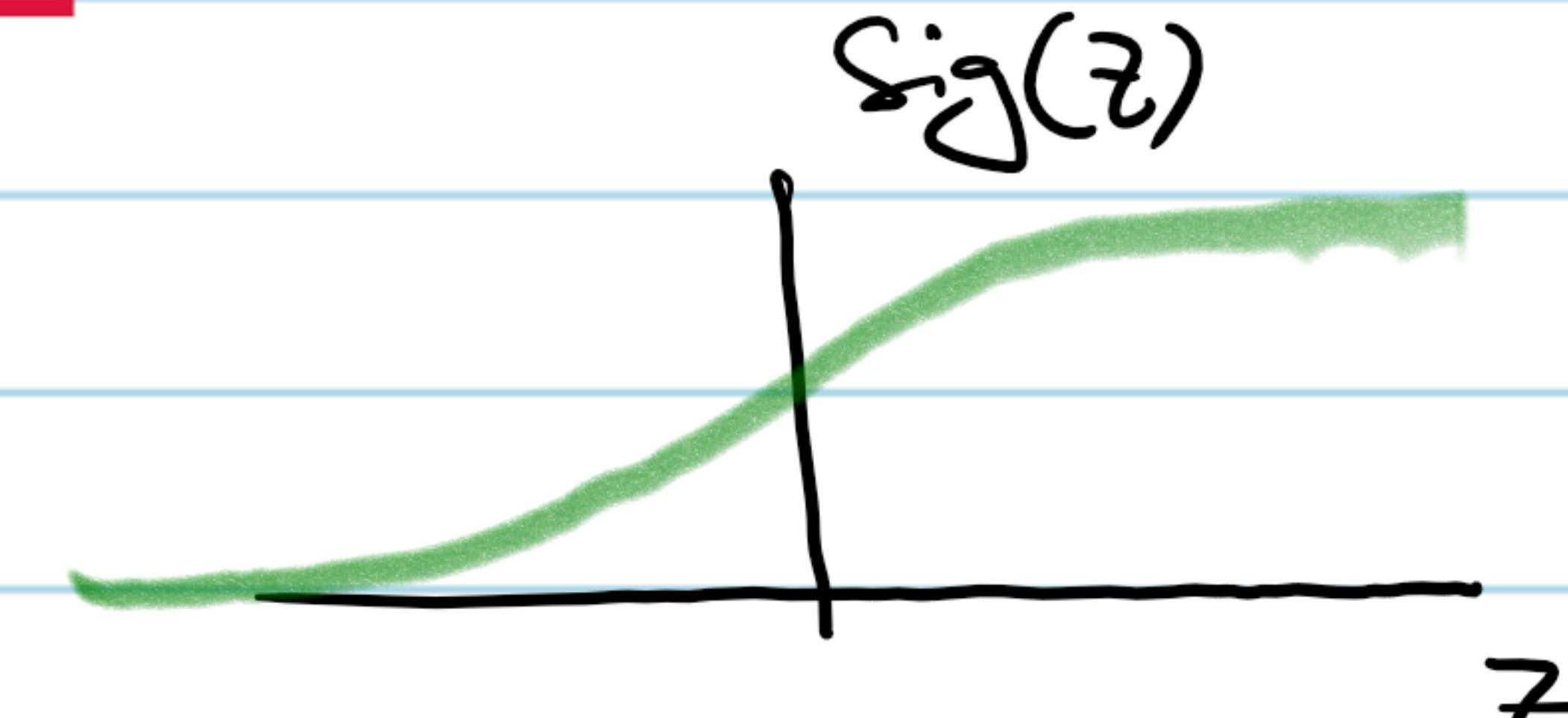
A NN can be described as combination of a bunch of these units (perceptron/LR).

We will briefly revisit these simple models and see why / how we end up with NN. For the rest of this discussion, we'll stick to LR. But almost everything would be similar for a perceptron.

LR

$$x_1 \quad x_2 \rightarrow z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \text{sig}(z)$$

$\text{sig}(z)$



For a classification problem

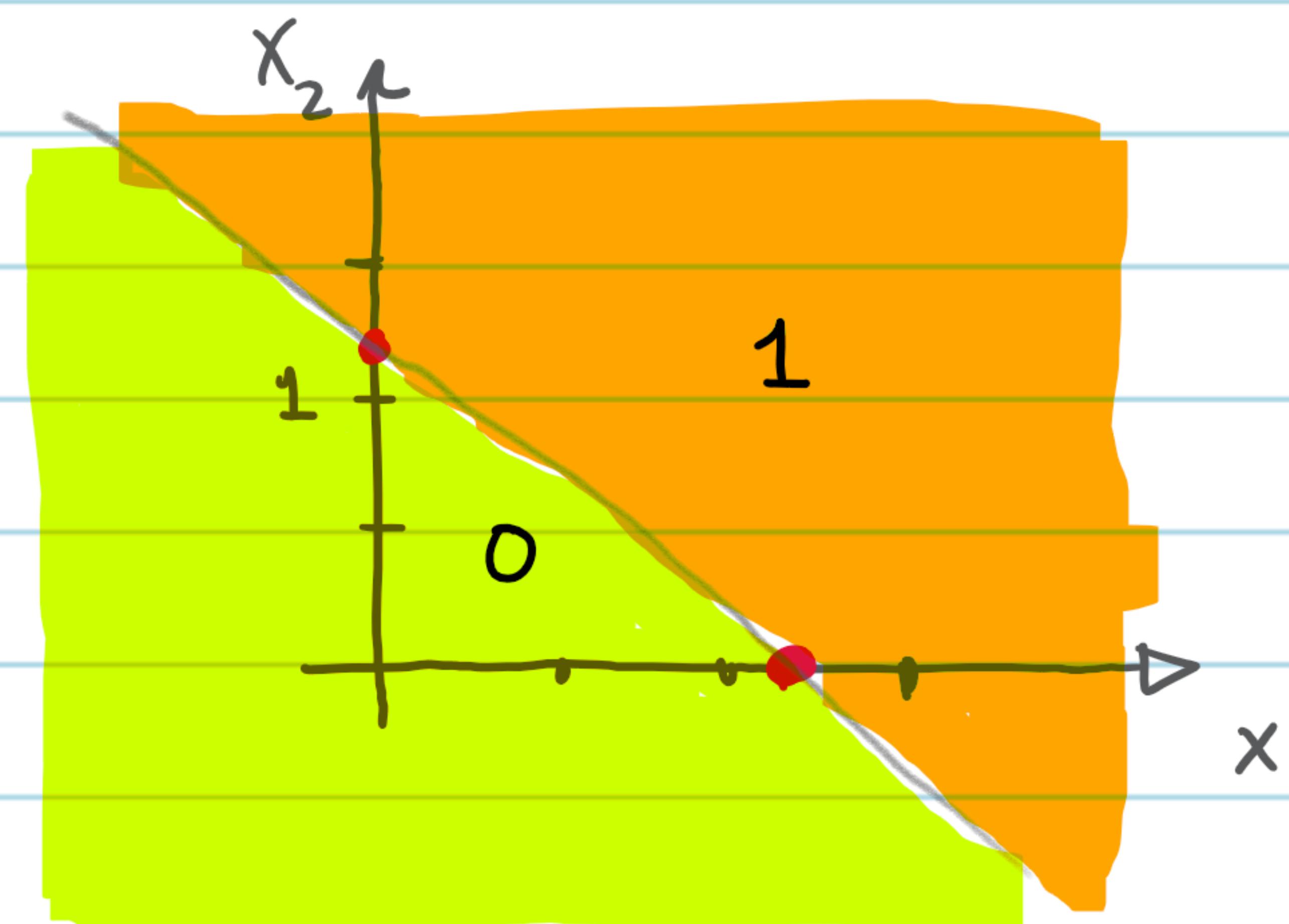
$$\bar{y} = (a \geq 1/2) \begin{cases} 0 \rightarrow \text{True} \\ 1 \rightarrow \text{False} \end{cases}$$

→ This would be
a step-function
for a perceptron.

Example:

$$w_1 = -1, w_2 = -1, b = 1.7 \Rightarrow -x_1 - x_2 + 1.7 \geq 0.5$$

$$a = 1/2 \Rightarrow \begin{cases} x_1 = 0, x_2 = 1.2 \\ x_2 = 0, x_1 = 1.2 \end{cases}$$



$$1.2 - x_1 = x_2$$

Consider the example above and restrict inputs to $x_{1,2} \in \{0, 1\}$.

x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

⇒ This is the And gate.

Assignment: Design a LR for the OR gate

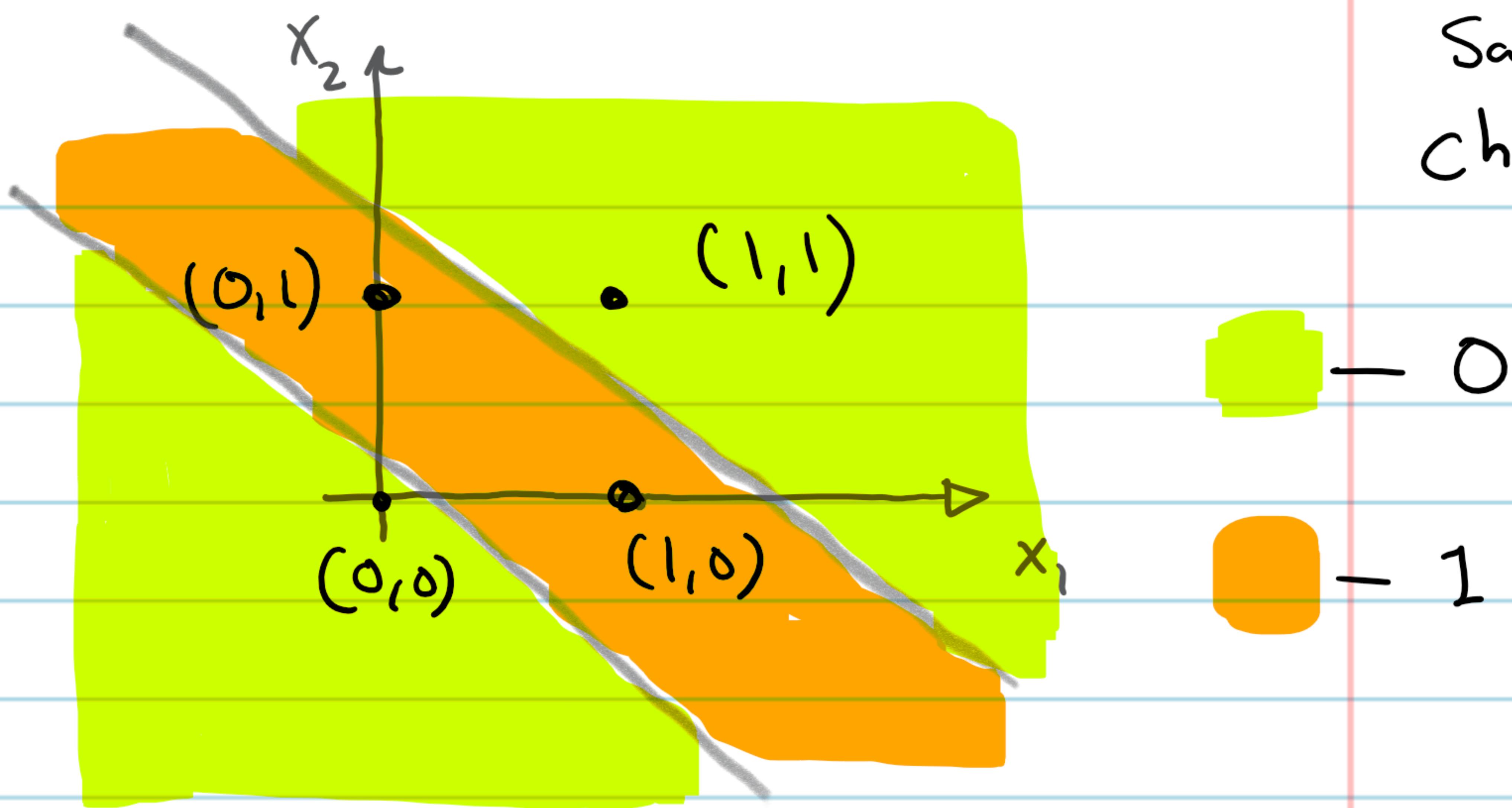
Design a LR for a NOT gate.

Question:

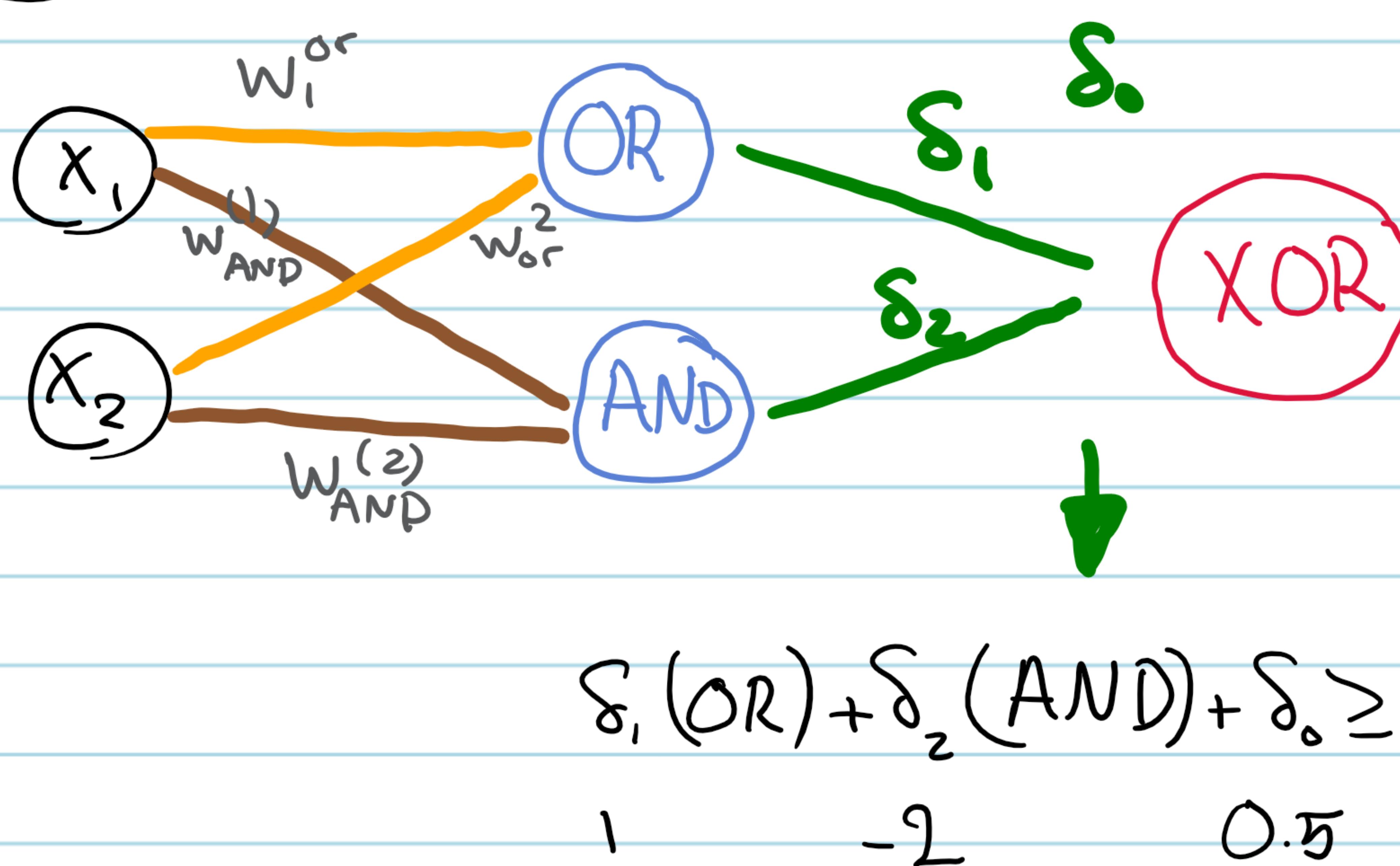
Can we do XOR with a single LR unit?

For the XOR we need:

Is it possible with a single unit?



What's the solution? One idea is to use more than one unit.



x_1	x_2	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

The idea is that a single unit cannot sometimes directly give some desired outcome, but we can combine multiple units to get the result we want.

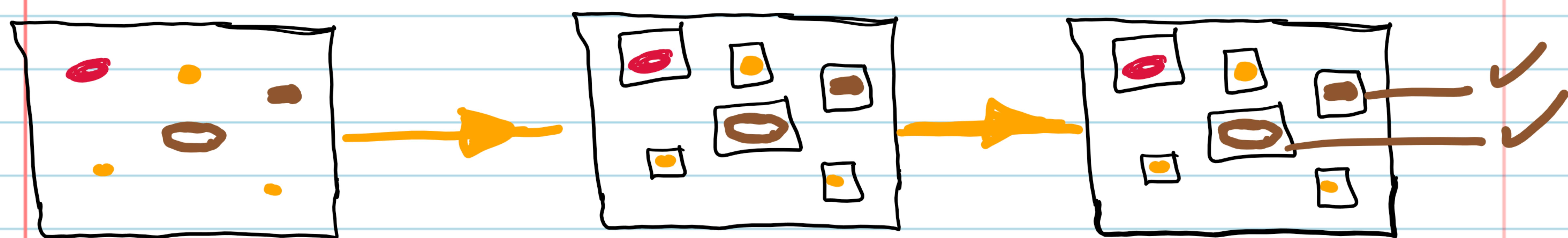
Also note that the gates AND & OR are universal which suggests that any logical gate can be built with combinations of different units in layers.

Hidden layers:

The units in the middle, between the input & output are called Hidden Layers. These serve as middle calculations that help get to the final output.

A more intuitive example:

Imagine you want to find a specific type of galaxy in an image



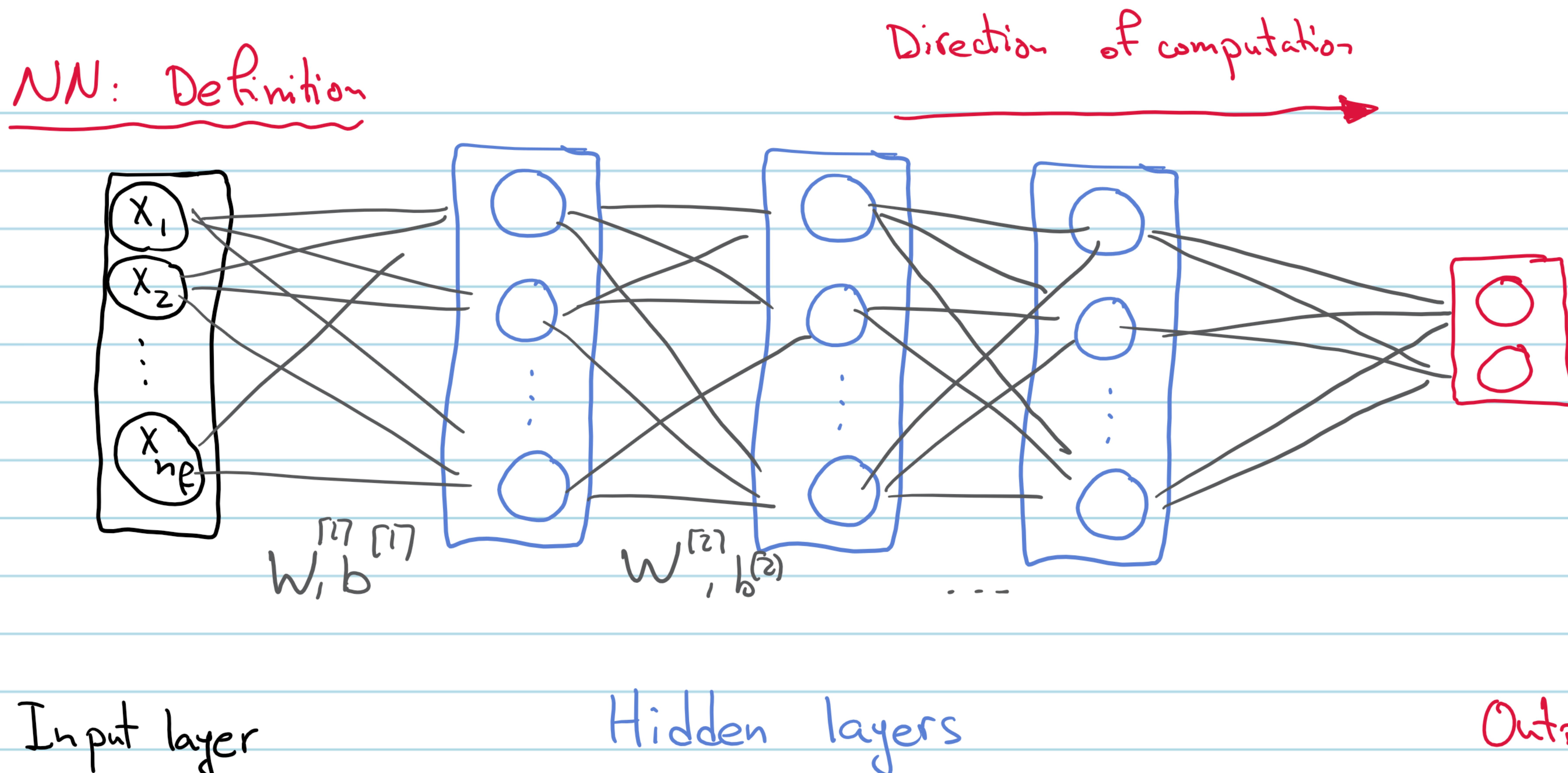
Input

Identify Objects

Classify
the Objects

↑
This is a hidden layer.
We're not interested in the results
of this layer, but we use it
to get to our final output.

NN: Definition



Each circle is a node and is similar to a LR with respect to the proceeding layer.

Notation

$$\vec{X}^{(i)} : \text{Input } \rightsquigarrow i^{\text{th}} \text{ sample. } \vec{X}^{(i)} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_p} \end{pmatrix}$$

X : Shape: (n_p, n_s)

Layer 1

$W^{[1]}$ → Weights for connecting inputs to the 1st layer. : Shape: (n_p, n_{h_1})

$b^{[1]}$ → Biases for the first Hidden layer. : Shape $(n_{h_1}, 1)$

$Z^{[1]}$ → Linear part of the 1st layer : $Z^{[1]} = (W^{[1]})^T \cdot X + b^{[1]}$

$A^{[1]}$ = Activation function for the 1st layer. e.g. sigmoid.

$a^{[1]} = A^{[1]}(Z^{[1]})$ → The outputs of the 1st layer.

Broadcasting

Notation

* Superscript $[l]$ index of the layer

* Inputs X

Outputs \tilde{Y} or Y

Hidden layers
output a

Shape

(n_r, n_s)

$(1, n_s)$

(n_e, n_s)

$W^{[l]}$: Weight connecting the outcomes of layer $l-1$ to l .

(n_{l-1}, n_e)

$b^{[l]}$: The Bias for the layer l .

(n_e, n_s)

$\tilde{z}^{[l]}$: linear outcome of layer l

(n_e, n_s)

$$\tilde{z}^{[l]} = (W^{[l]})^T \cdot a^{[l-1]} + b^{[l]}$$

$A^{[l]}$: Activation Function for layer l .

$a^{[l]}$: Outcomes of layer l :

(n_e, n_s)

$$a^{[l]} = A^{[l]}(\tilde{z}^{[l]}) = A^{[l]}((W^{[l]})^T \cdot a^{[l-1]} + b^{[l]}).$$

Question: Implement this function, such that given
 $(W^{[l]}, b^{[l]}, A^{[l]})$ it outputs the function
 above.

Activation Functions and non-linearity

Question: What would happen if we $A^{[l]}(x) = x \quad \forall l.$?

$$\begin{aligned}
 a^{[l]} &= z^{[l]} = (w^{[l]})^T \cdot a^{[l-1]} + b^{[l]} \\
 &= (w^{[l]})^T \cdot (w^{[l-1] T} \cdot a^{[l-2]} + b^{[l-1]}) + b^{[l]} \\
 &= \dots \\
 &= w^{[l] T} \cdot w^{[l-1] T} \dots x + (b^{[l]} + w^{[l] T} b^{[l-1]} + \dots) \\
 &= \Omega^T \cdot X + B
 \end{aligned}$$

The whole network reduces
to a linear reg. model.

So w/o the activation functions, it would not be the powerful model we need and the depth does not help.

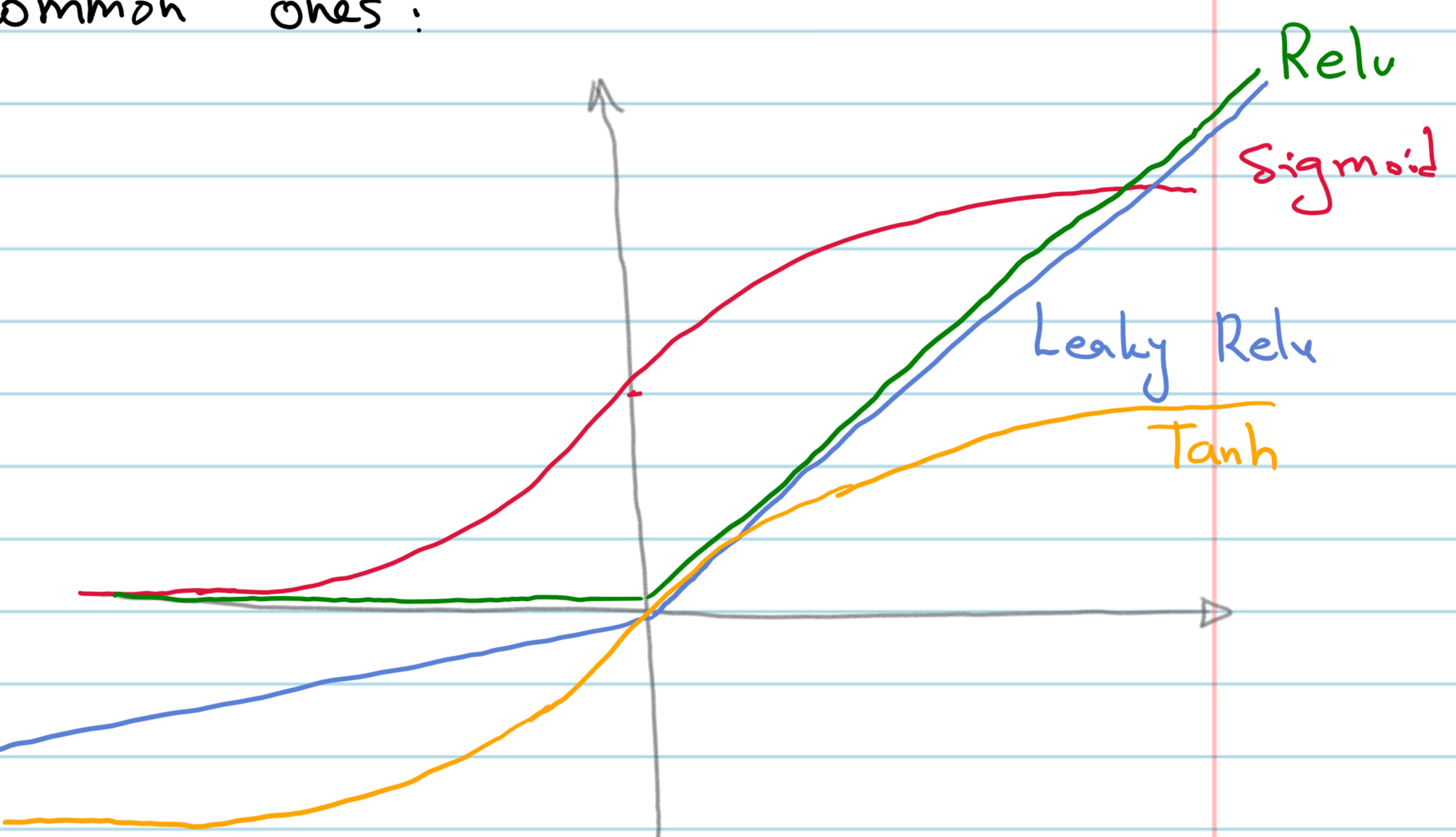
Different choices of activation functions exist, here are some of the more common ones:

Sigmoid: $A(x) = \frac{1}{1+e^{-x}}$

Tanh : $A(x) = \tanh(x)$

ReLU : $A(x) = \max(0, x)$

Leaky ReLU : $A(x) = \max(0.1x, x)$



The choice of activation function is one of the hyperparameters that depending on application, could change.

An obvious example is that for a classification problem, i.e. $y \in \{0, 1\}$, the activation for the output layer cannot be ReLU.

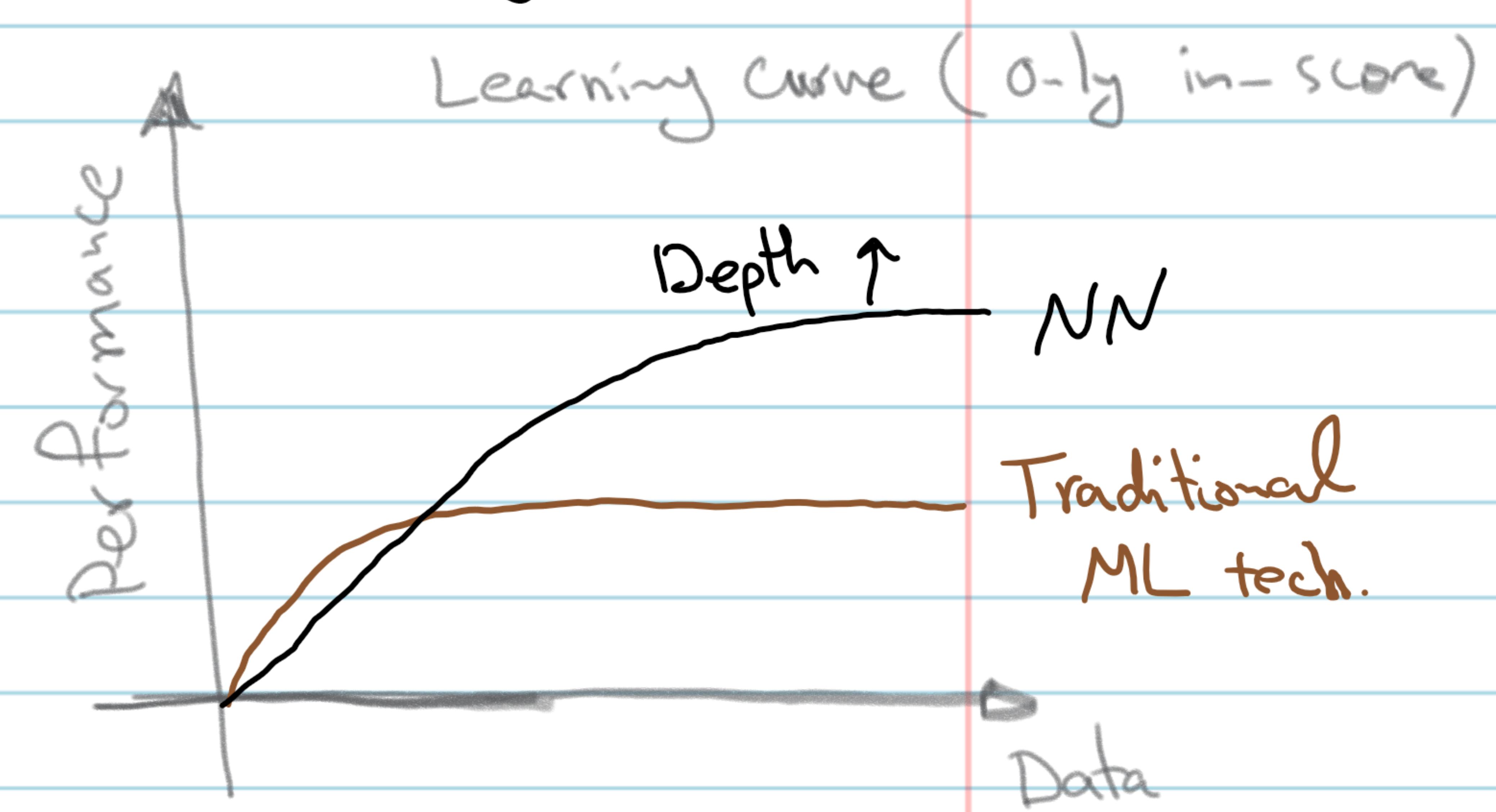
But, there's more to it. Often, the activation function can significantly affect the training time.

What's so good about NN?

Probably the main advantage is the tunable complexity of NN.

For large enough data, NN could achieve higher performances.

That is to say the bias of the models reduces with more layers.



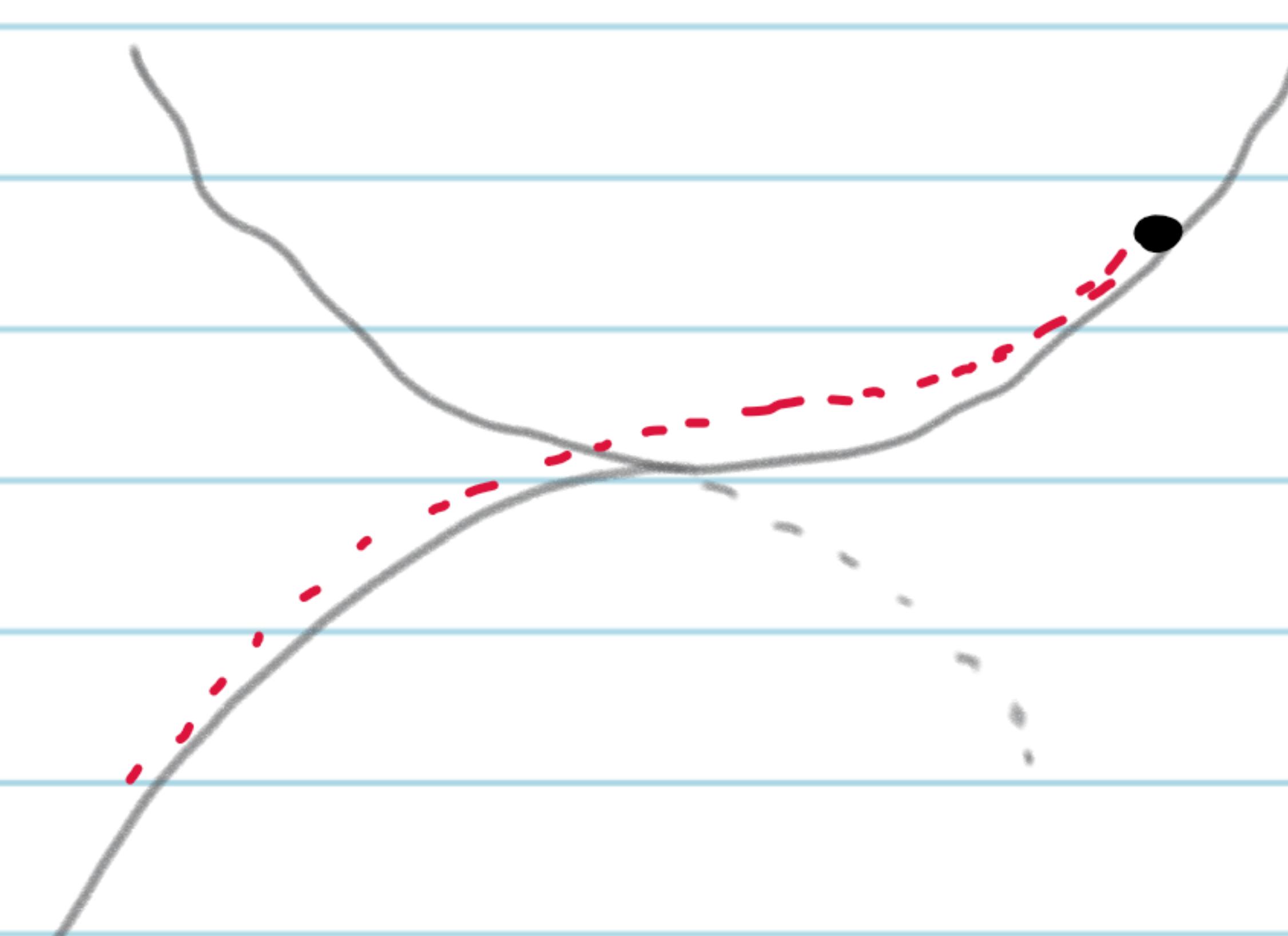
• Local minima problem

As there are too many param in the model ($\sim 10^4 - 10^7$), it is unlikely to find local minimum.

There are often saddle points and the optimization does not get stuck.

What should scare us

• Too large, too many param



* ...

Disadvantage:

Require too much data (often) and there are too many params to optimize over.

Universality :

Universal approx. Theorem.

* With enough width($O(\exp n_f)$), γ_{NN} can approximate continuous functions on compact subsets of \mathbb{R}^n .

* With enough depth with width as few as $n+c$ it would be possible to approximate wide classes of functions. (See relevant refs for more details).

→ It is interesting that $O(\exp(n))$ width reduces to linear by allowing more depth.

Training : Gradient Descent

Let's say that we want to train an LR to fit
 $\{X, Y\}$.

We need a loss function. We take Cross-entropy :

$$L_{ce}(Y, \tilde{Y}) = - \sum_i Y_i \log \tilde{Y}_i + (1-Y_i) \log (1-\tilde{Y}_i).$$

For LR, we have $\tilde{Y} = \sigma(W \cdot X + b)$ $W \rightarrow W^T$
(for simplicity)

Each step of the GD:

$$W \rightarrow W - \alpha \frac{\partial L_{ce}}{\partial W}$$

$$b \rightarrow b - \alpha \frac{\partial L_{ce}}{\partial b}$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \tilde{Y}} \frac{\partial \tilde{Y}}{\partial w_i} = \dots \underset{\text{(Chain rule)}}{=} \left(-\frac{Y}{\tilde{Y}} + \frac{1-Y}{1-\tilde{Y}} \right) \frac{\partial \tilde{Y}}{\partial w_i}$$

$$\frac{\partial \tilde{Y}}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z)) = \tilde{Y}(1-\tilde{Y})$$

$$\Rightarrow \frac{\partial L}{\partial w_i} = \left(-\frac{Y}{\tilde{Y}} + \frac{1-Y}{1-\tilde{Y}} \right) \tilde{Y}(1-\tilde{Y})(x_i) \\ \frac{(-Y(1-\tilde{Y}) + \tilde{Y}(1-Y))}{(\tilde{Y}-Y)} x_i = (\tilde{Y}-Y)x_i$$

$$\frac{\partial L}{\partial b} = (\tilde{Y}-Y)$$