

# Overview

What is Deep Learning?

Single neuron

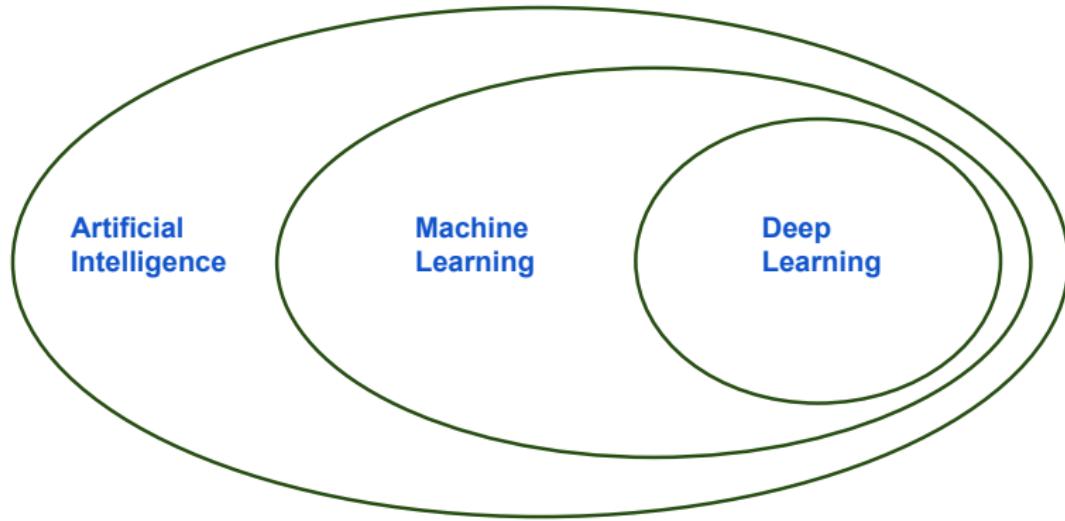
Single layer network

Multi-Layer network

Parameter learning

Further reading

# WHAT IS DEEP LEARNING



- Deep learning is a subfield of ML based on artificial neural networks.

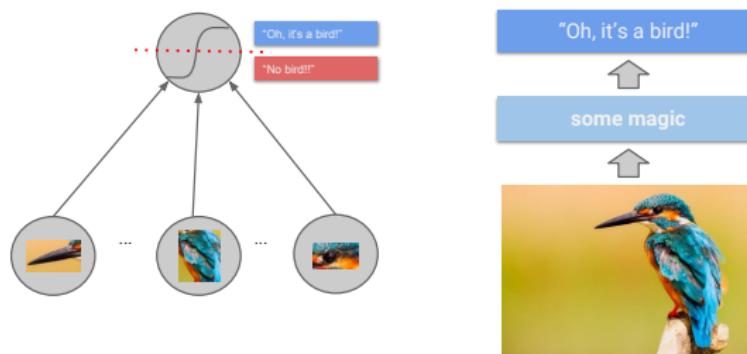
# DEEP LEARNING AND NEURAL NETWORKS

- Deep learning itself is not *new*:
  - Neural networks have been around since the 70s.
  - *Deep* neural networks, i.e., networks with multiple hidden layers, are not much younger.
- Why everybody is talking about deep learning now:
  - ① Specialized, powerful hardware allows training of huge neural networks to push the state-of-the-art on difficult problems.
  - ② Large amount of data is available.
  - ③ Special network architectures for image/text data.
  - ④ Better optimization and regularization strategies.

# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*“Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction.”*

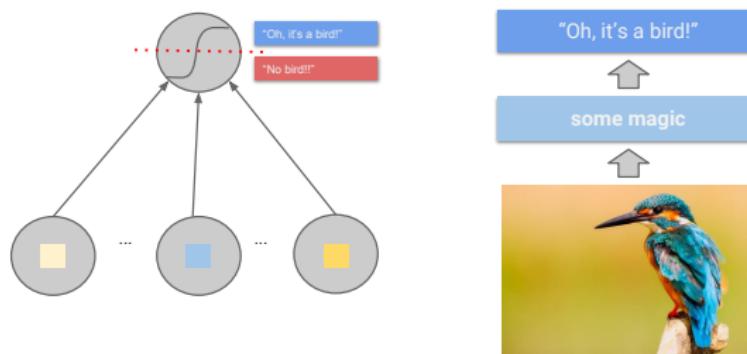
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*“Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction.”*

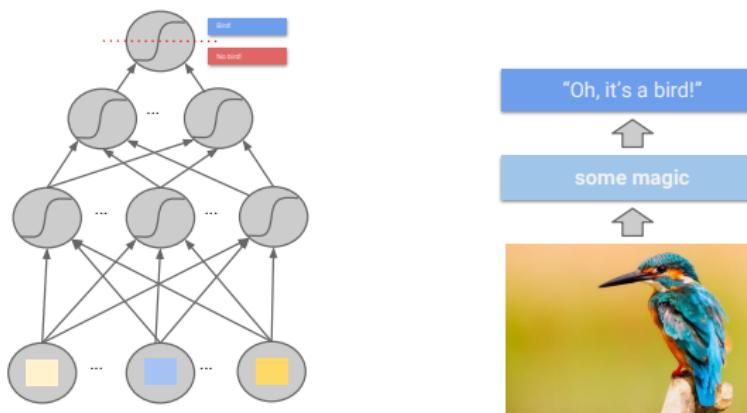
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*“Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction.”*

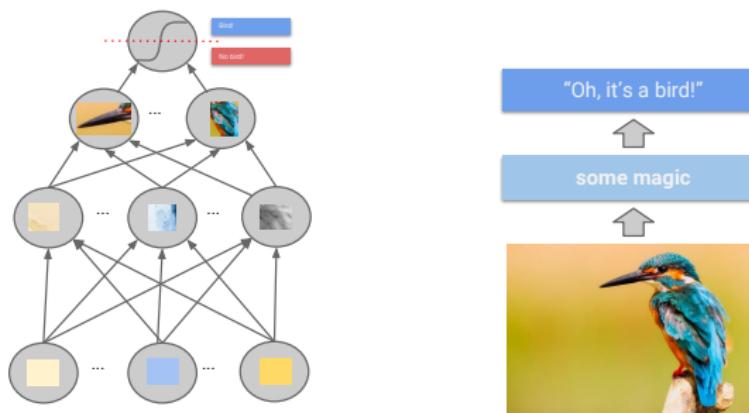
Y. Bengio



# IMAGE CLASSIFICATION WITH NEURAL NETWORKS

*“Machine learning algorithms, inspired by the brain, based on learning multiple levels of representation/abstraction.”*

Y. Bengio



# POSSIBLE USE-CASES

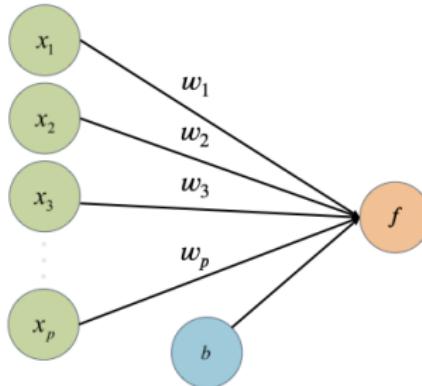
**Deep learning can be extremely valuable if the data has these properties:**

- It is high dimensional.
- Each single feature itself is not very informative but only a combination of them might be.
- There is a large amount of training data.

**This implies that for tabular data, deep learning is rarely the correct model choice.**

- Without extensive tuning, models like random forests or gradient boosting will outperform deep learning most of the time.
- One exception is data with categorical features with many levels.

# A SINGLE NEURON



Perceptron with **input features**  $x_1, x_2, \dots, x_p$ , **weights**  $w_1, w_2, \dots, w_p$ , **bias term**  $b$ , and **activation function**  $\tau$ .

- The perceptron is a single artificial neuron and the basic computational unit of neural networks.
- It is a weighted sum of input values, transformed by  $\tau$ :

$$f(x) = \tau(w_1x_1 + \dots + w_px_p + b) = \tau(\mathbf{w}^T \mathbf{x} + b)$$

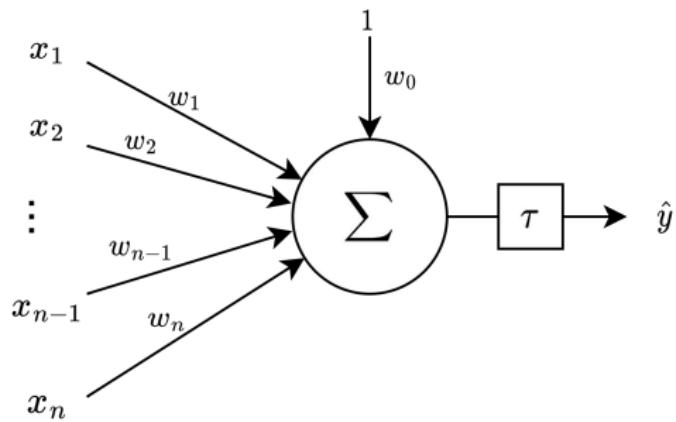
$$\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n)$$

$$\begin{aligned}\hat{y} &= \sigma(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n) \\ &= \sigma(\vec{w}^\top \vec{x})\end{aligned}$$

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{array}{c} \text{"Activation"} \\ \downarrow \\ \hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n) \\ = \sigma(\vec{w}^\top \vec{x}) \\ \uparrow \\ \text{"Activation function"} \\ \vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad \vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ \uparrow \\ \text{"Weight vector"} \quad \uparrow \\ \text{"Feature vector"} \end{array}$$

$$\hat{y} = \tau(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n)$$



# A SINGLE NEURON

We consider a perceptron with 3-dimensional input, i.e.

$$f(\mathbf{x}) = \tau(w_1x_1 + w_2x_2 + w_3x_3 + b).$$

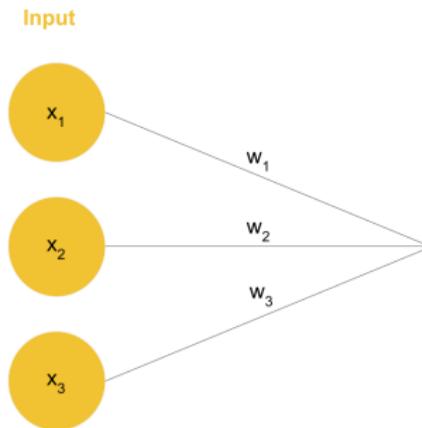
- Input features  $\mathbf{x}$  are represented by nodes in the “input layer”.



- In general, a  $p$ -dimensional input vector  $\mathbf{x}$  will be represented by  $p$  nodes in the input layer.

# A SINGLE NEURON

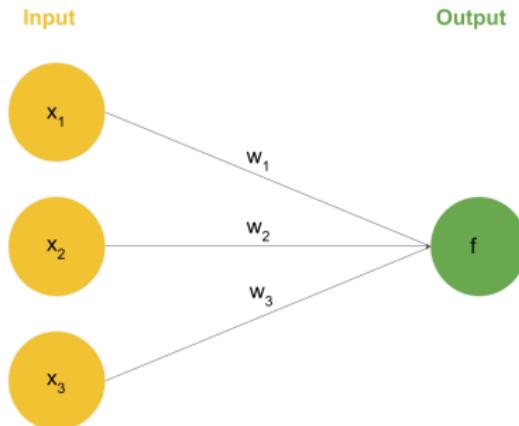
- Weights  $w$  are connected to edges from the input layer.



- The bias term  $b$  is implicit here. It is often not visualized as a separate node.

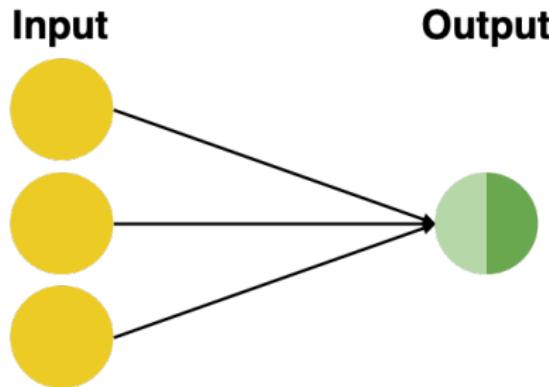
# A SINGLE NEURON

- The computation  $\tau(w_1x_1 + w_2x_2 + w_3x_3 + b)$  is represented by the neuron in the “output layer”.



# A SINGLE NEURON

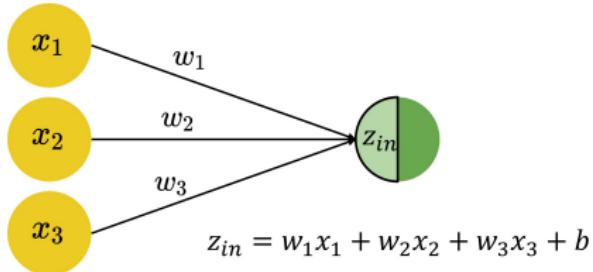
- You can picture the input vector being "fed" to neurons on the left followed by a sequence of computations performed from left to right. This is called a **forward pass**.



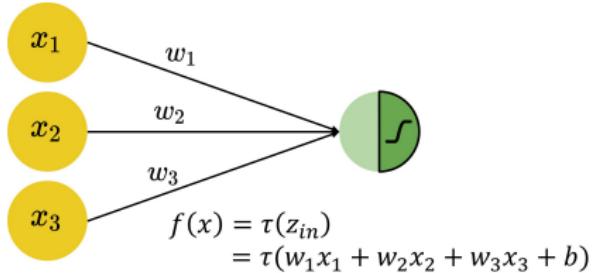
# A SINGLE NEURON

A neuron performs a 2-step computation:

- ➊ **Affine Transformation:** weighted sum of inputs plus bias.

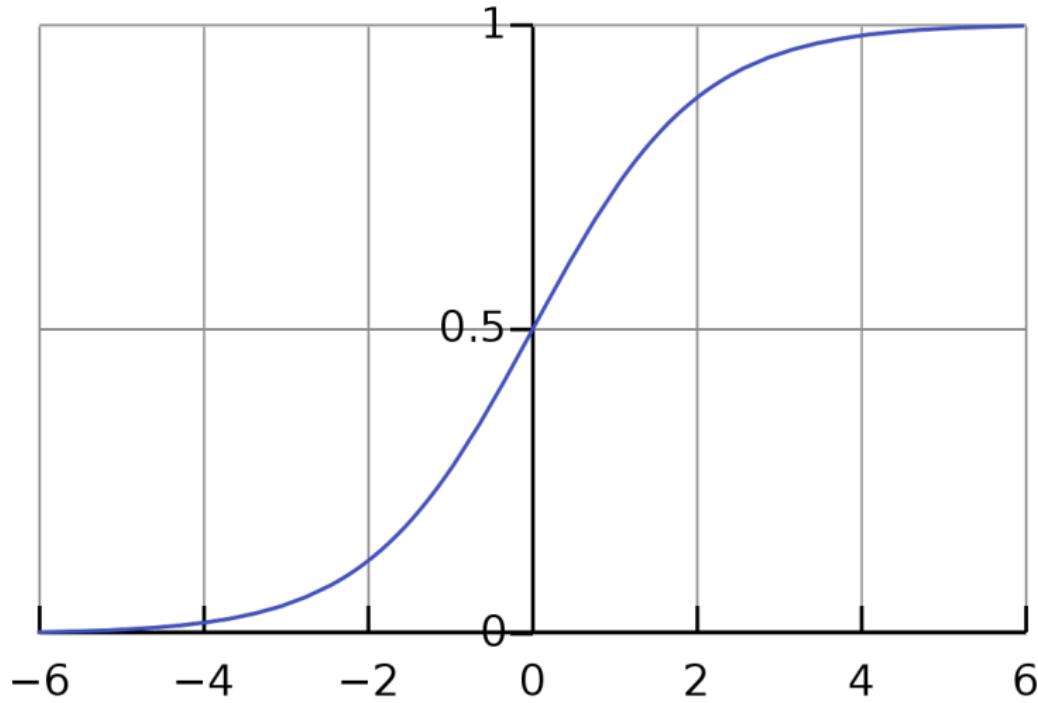


- ➋ **Non-linear Activation:** a non-linear transformation applied to the weighted sum.



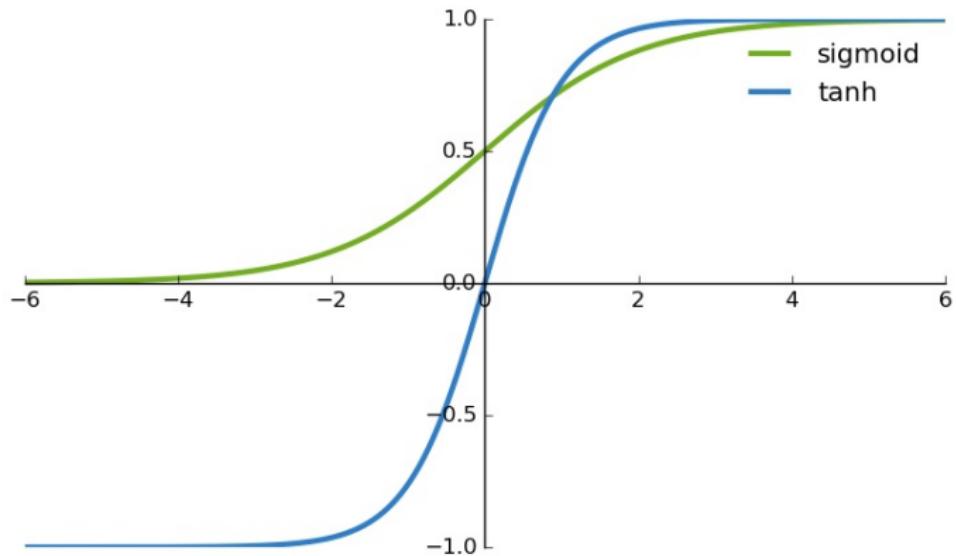
# Activation functions

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



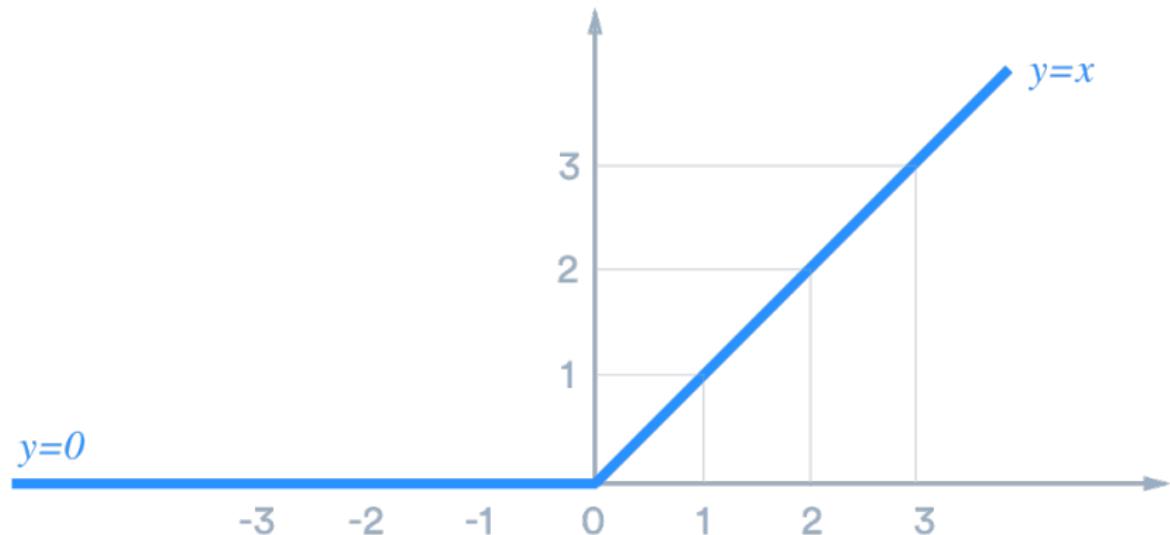
# Activation functions

2. Tanh:  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



# Activation functions

3. Rectified linear unit:  $ReLU(x) = \max(0, x)$



# A SINGLE NEURON: OPTIMIZATION

- To optimize this model, we minimize the empirical risk

$$\mathcal{R}_{\text{emp}} = \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})),$$

where  $L(y, f(\mathbf{x}))$  is a loss function. It compares the network's predictions  $f(\mathbf{x})$  to the ground truth  $y$ .

- For regression, we typically use the L2 loss (rarely L1):

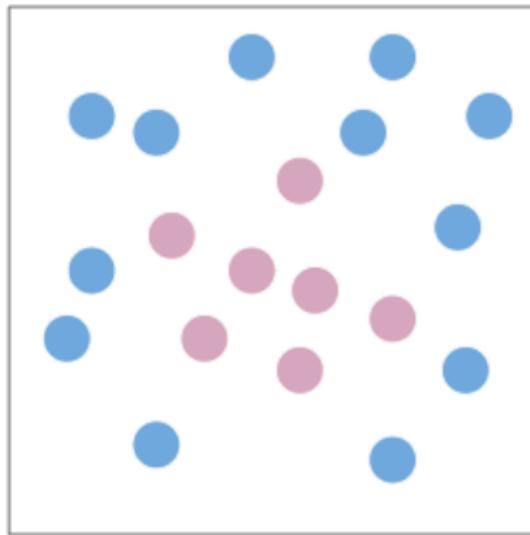
$$L(y, f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2$$

- For binary classification, we typically apply the cross entropy loss (also known as Bernoulli loss):

$$L(y, f(\mathbf{x})) = -(y \log f(\mathbf{x}) + (1 - y) \log(1 - f(\mathbf{x})))$$

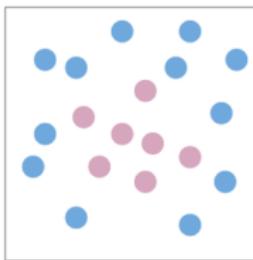
# MOTIVATION

Can a single neuron perform binary classification of these points?

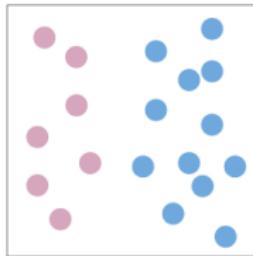


# MOTIVATION

- As a single neuron is restricted to learning only linear decision boundaries, its performance on the following task is quite poor:

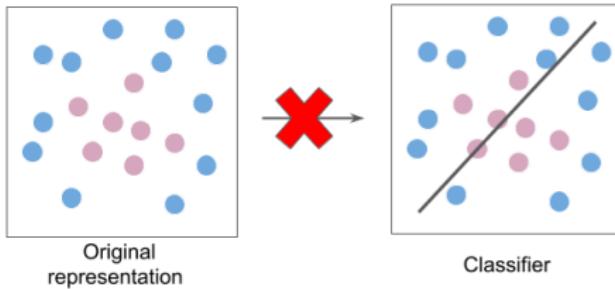


- However, the neuron can easily separate the classes if the original features are transformed (e.g., from Cartesian to polar coordinates):



# MOTIVATION

- Instead of classifying the data in the original representation,

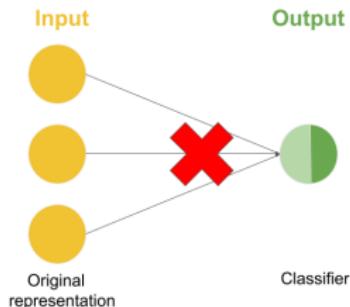


- we classify it in a new feature space.

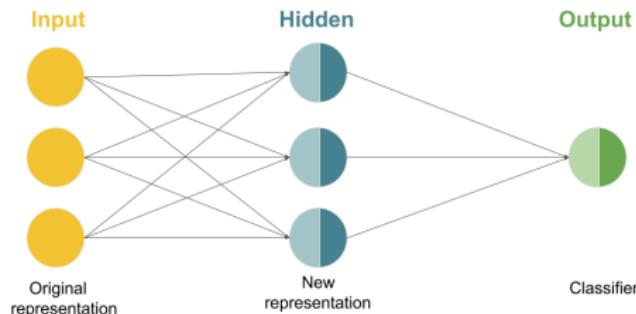


# MOTIVATION

- Analogously, instead of a single neuron,



- we use more complex networks.



# REPRESENTATION LEARNING

- It is *very* critical to feed a classifier the “right” features in order for it to perform well.
- Before deep learning took off, features for tasks like machine vision and speech recognition were “hand-designed” by domain experts. This step of the machine learning pipeline is called **feature engineering**.
- DL automates feature engineering. This is called **representation learning**.

# SINGLE HIDDEN LAYER NETWORKS

**Single neurons** perform a 2-step computation:

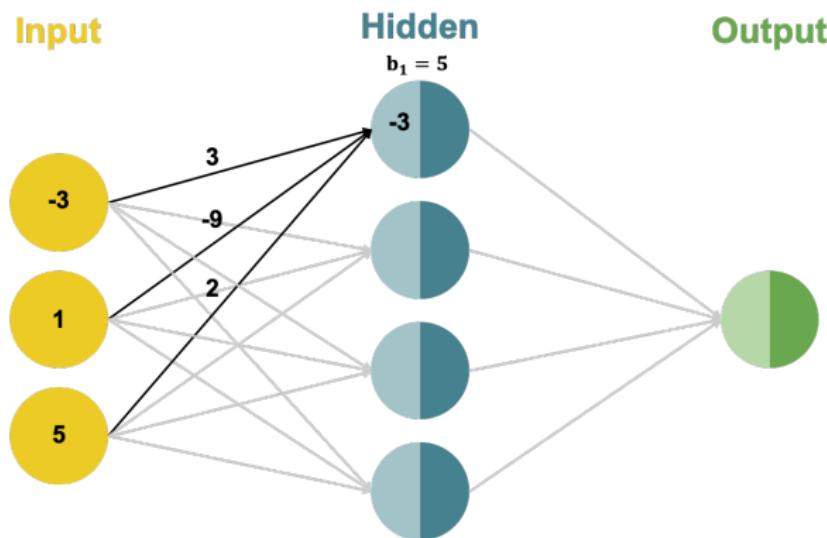
- ❶ **Affine Transformation:** a weighted sum of inputs plus bias.
- ❷ **Activation:** a non-linear transformation on the weighted sum.

**Single hidden layer networks** consist of two layers (without input layer):

- ❶ **Hidden Layer:** having a set of neurons.
  - ❷ **Output Layer:** having one or more output neurons.
- Multiple inputs are simultaneously fed to the network.
  - Each neuron in the hidden layer performs a 2-step computation.
  - The final output of the network is then calculated by another 2-step computation performed by the neuron in the output layer.

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

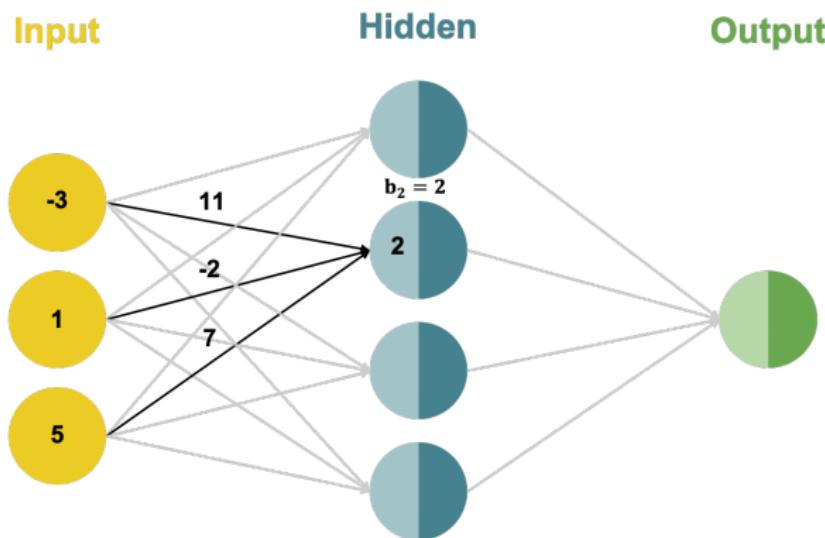


$$z_{\text{in}}^{(1)} = w_{11}x^{(1)} + w_{21}x^{(2)} + w_{31}x^{(3)} + b_1$$

$$z_{\text{in}}^{(1)} = 3 * (-3) + (-9) * 1 + 2 * 5 + 5 = -3$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

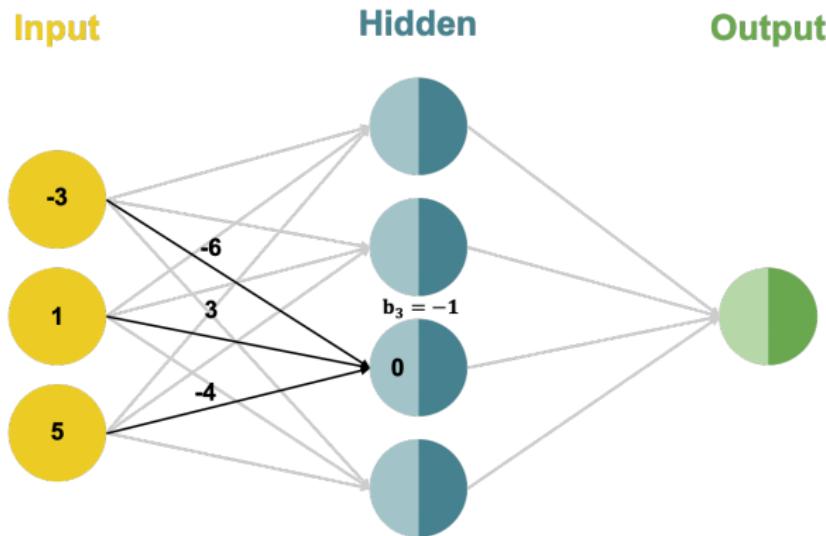


$$z_{\text{in}}^{(2)} = w_{12}x^{(1)} + w_{22}x^{(2)} + w_{32}x^{(3)} + b_2$$

$$z_{\text{in}}^{(2)} = 11 * (-3) + (-2) * 1 + 7 * 5 + 2 = 2$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

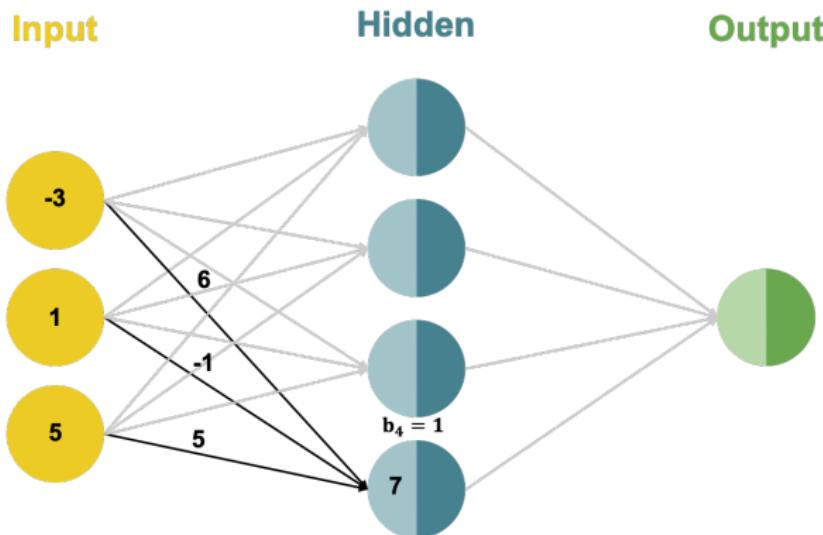


$$z_{\text{in}}^{(3)} = w_{13}x^{(1)} + w_{23}x^{(2)} + w_{33}x^{(3)} + b_3$$

$$z_{\text{in}}^{(3)} = (-6) * (-3) + 3 * 1 + (-4) * 5 - 1 = 0$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:

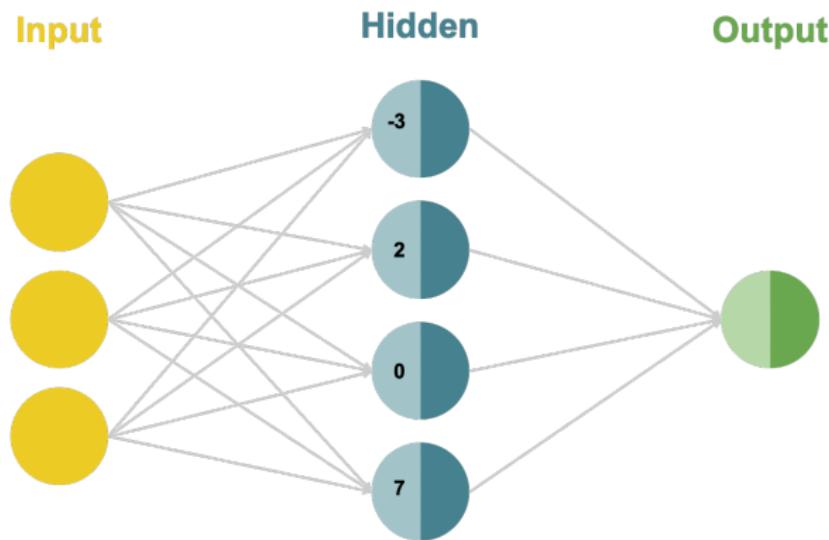


$$z_{\text{in}}^{(4)} = w_{14}x^{(1)} + w_{24}x^{(2)} + w_{34}x^{(3)} + b_4$$

$$z_{\text{in}}^{(4)} = 6 * (-3) + (-1) * 1 + 5 * 5 + 1 = 7$$

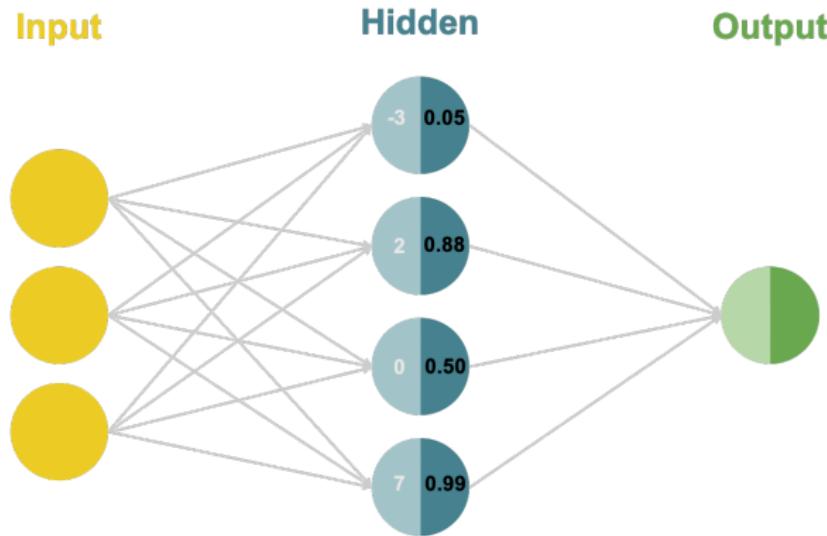
# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

Each neuron in the hidden layer performs an **affine transformation** on the inputs:



# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

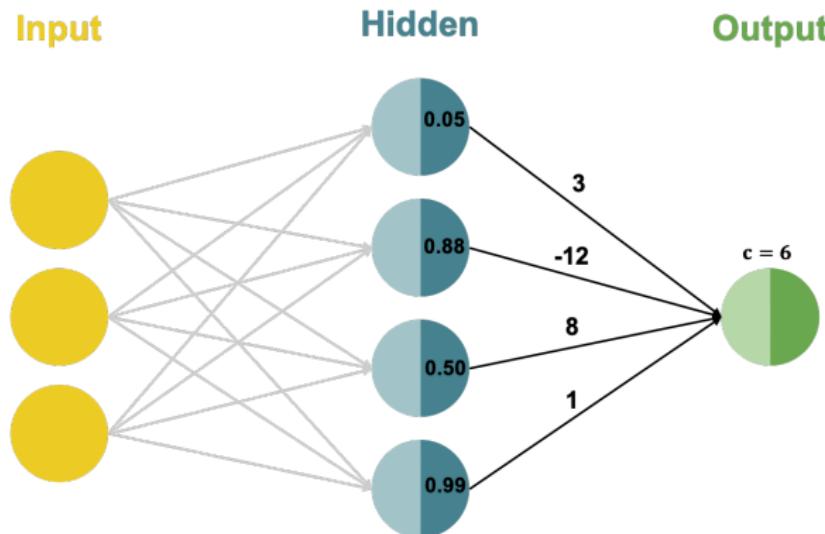
Each hidden neuron performs a non-linear **activation** transformation on the weight sum:



$$z_{\text{out}}^{(i)} = \sigma(z_{\text{in}}^{(i)}) = \frac{1}{1+e^{-z_{\text{in}}^{(i)}}}$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

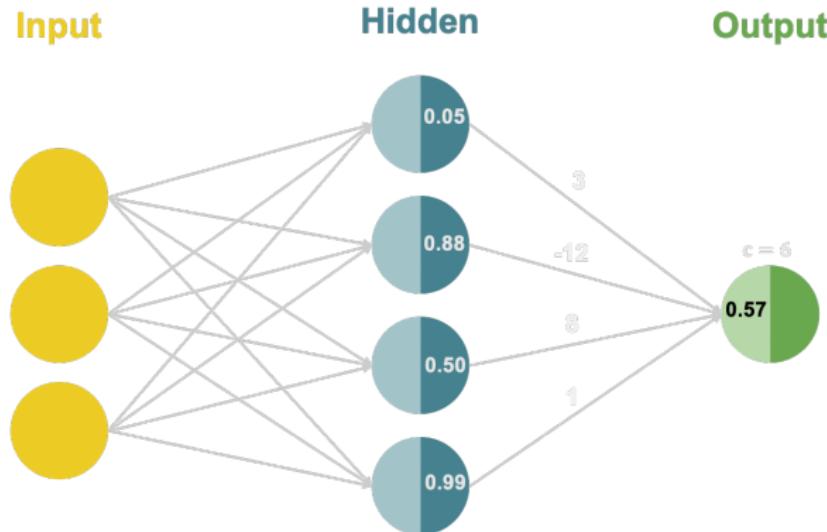
The output neuron performs an **affine transformation** on its inputs:



$$f_{\text{in}} = u_1 z_{\text{out}}^{(1)} + u_2 z_{\text{out}}^{(2)} + u_3 z_{\text{out}}^{(3)} + u_4 z_{\text{out}}^{(4)} + c$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs an **affine transformation** on its inputs:

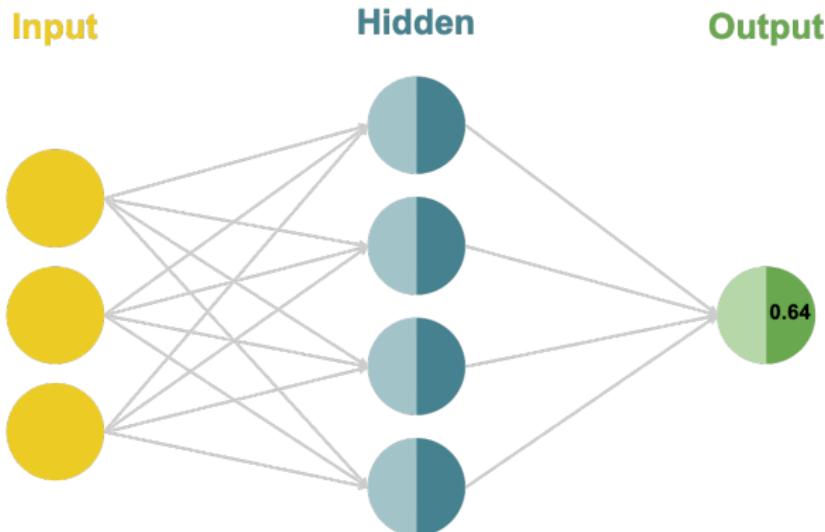


$$f_{\text{in}} = u_1 z_{\text{out}}^{(1)} + u_2 z_{\text{out}}^{(2)} + u_3 z_{\text{out}}^{(3)} + u_4 z_{\text{out}}^{(4)} + c$$

$$f_{\text{in}} = 3 * 0.05 + (-12) * 0.88 + 8 * 0.50 + 1 * 0.99 + 6 = 0.57$$

# SINGLE HIDDEN LAYER NETWORKS: EXAMPLE

The output neuron performs a non-linear **activation** transformation on the weight sum:



$$f_{\text{out}} = \sigma(f_{\text{in}}) = \frac{1}{1+e^{f_{\text{in}}}}$$

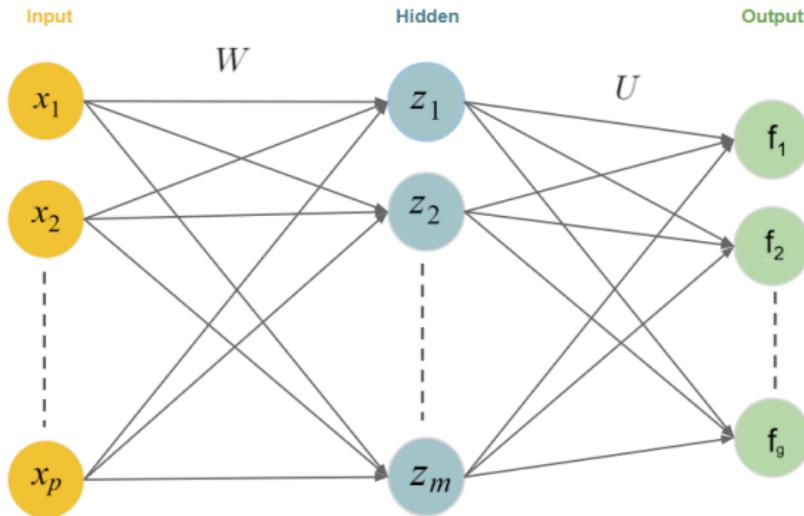
$$f_{\text{out}} = \frac{1}{1+e^{0.57}} = 0.64$$

# MULTI-CLASS CLASSIFICATION

- We have only considered regression and binary classification problems so far.
- How can we get a neural network to perform multiclass classification?

# MULTI-CLASS CLASSIFICATION

- The first step is to add additional neurons to the output layer.
- Each neuron in the layer will represent a specific class (number of neurons in the output layer = number of classes).



**Figure:** Structure of a single hidden layer, feed-forward neural network for  $g$ -class classification problems (bias term omitted).

# MULTI-CLASS CLASSIFICATION

## Notation:

- For  $g$ -class classification,  $g$  output units:

$$\mathbf{f} = (f_1, \dots, f_g)$$

- $m$  hidden neurons  $z_1, \dots, z_m$ , with

$$z_j = \sigma(\mathbf{W}_j^T \mathbf{x}), \quad j = 1, \dots, m.$$

- Compute linear combinations of derived features  $z$ :

$$f_{in,k} = \mathbf{U}_k^T \mathbf{z}, \quad \mathbf{z} = (z_1, \dots, z_m)^T, \quad k = 1, \dots, g$$

# MULTI-CLASS CLASSIFICATION

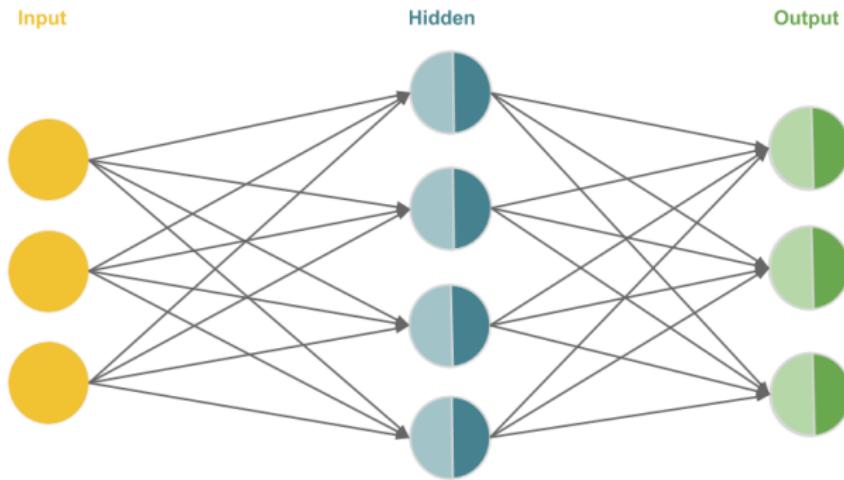
- The second step is to apply a **softmax** activation function to the output layer.
- This gives us a probability distribution over  $g$  different possible classes:

$$f_{out,k} = \tau_k(f_{in,k}) = \frac{\exp(f_{in,k})}{\sum_{k'=1}^g \exp(f_{in,k'})}$$

- This is the same transformation used in softmax regression!
- Derivative  $\frac{\partial \tau(\mathbf{f}_{in})}{\partial \mathbf{f}_{in}} = \text{diag}(\tau(\mathbf{f}_{in})) - \tau(\mathbf{f}_{in})\tau(\mathbf{f}_{in})^T$
- It is a “smooth” approximation of the argmax operation, so  $\tau((1, 1000, 2)^T) \approx (0, 1, 0)^T$  (picks out 2nd element!).

# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\begin{pmatrix} 3 & -9 & 2 \\ 11 & -2 & 7 \\ -6 & 3 & -4 \\ 6 & -1 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 2 \\ -1 \\ 1 \end{pmatrix}$$

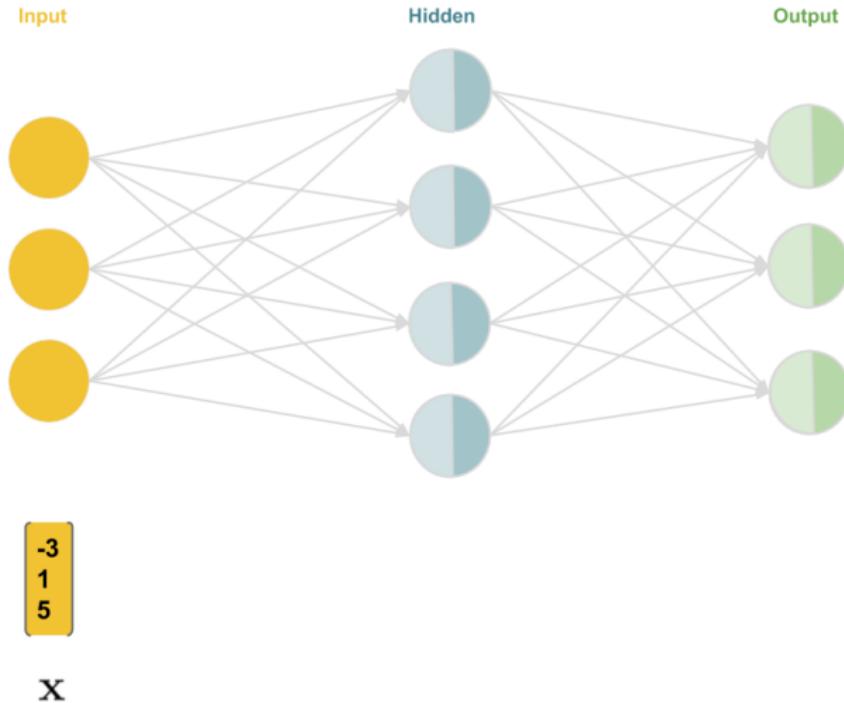
$$W^T \quad b$$

$$\begin{pmatrix} 3 & -12 & 8 & 1 \\ 2 & -3 & 9 & 1 \\ -5 & 1 & -1 & 7 \end{pmatrix} \begin{pmatrix} 6 \\ 0 \\ -8 \end{pmatrix}$$

$$U^T \quad c$$

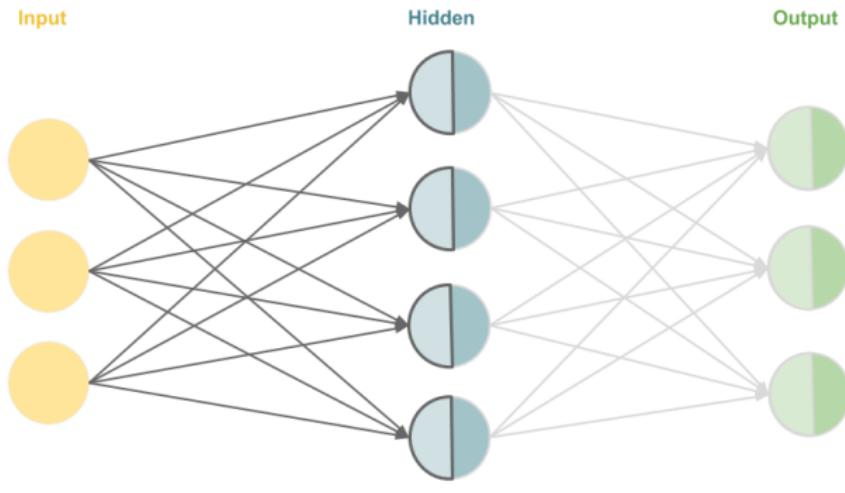
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).

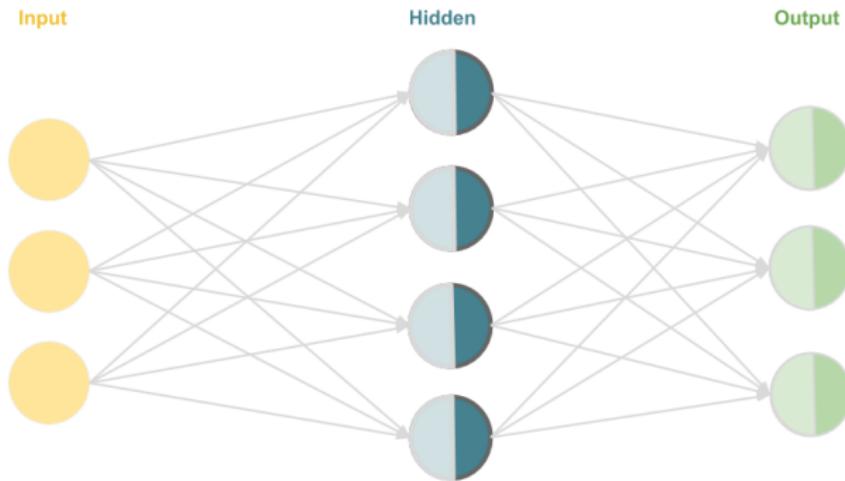


$$\begin{bmatrix} -3 \\ 1 \\ 5 \end{bmatrix} \quad \left( \begin{array}{l} (-3)^*3 + 1^*(-9) + 5^*2 + 5 \\ (-3)^*11 + 1^*(-2) + 5^*7 + 2 \\ (-3)^*(-6) + 1^*3 + 5^*(-4) + (-1) \\ (-3)^*6 + 1^*(-1) + 5^*5 + 1 \end{array} \right)$$

$$\mathbf{x} \quad \mathbf{z}_{in} = \mathbf{w}^T \mathbf{x} + \mathbf{b}$$

# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



$$\begin{bmatrix} -3 \\ 1 \\ 5 \end{bmatrix}$$

**X**

$$\begin{bmatrix} -3 \\ 2 \\ 0 \\ 7 \end{bmatrix}$$

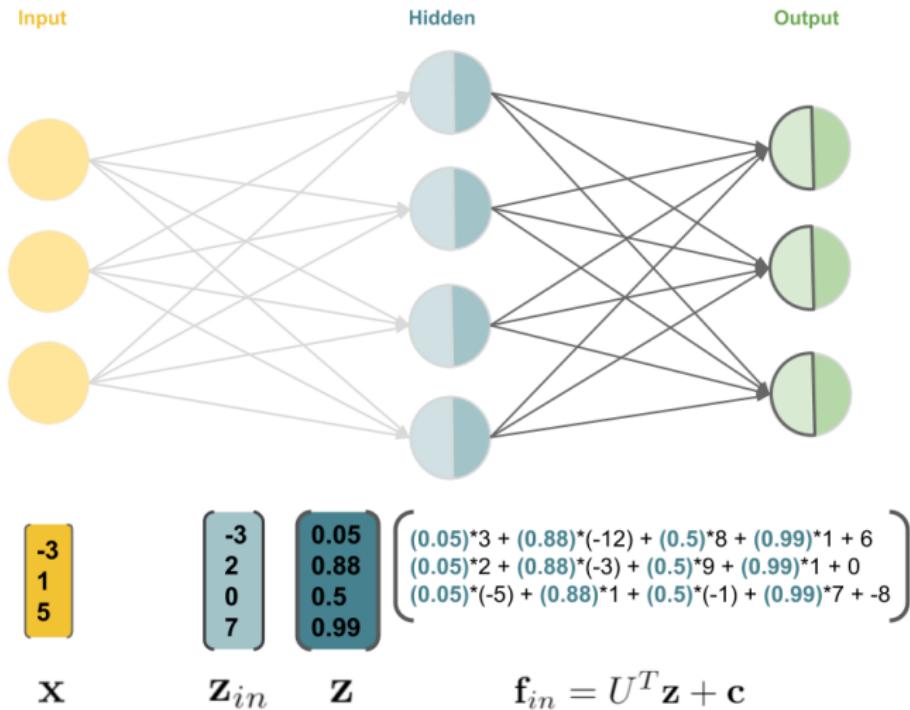
**$z_{in}$**

$$\begin{bmatrix} 1 / (1 + \exp(-(-3))) \\ 1 / (1 + \exp(-2)) \\ 1 / (1 + \exp(-0)) \\ 1 / (1 + \exp(-7)) \end{bmatrix}$$

$$\mathbf{z} = \mathbf{z}_{out} = \sigma(\mathbf{z}_{in})$$

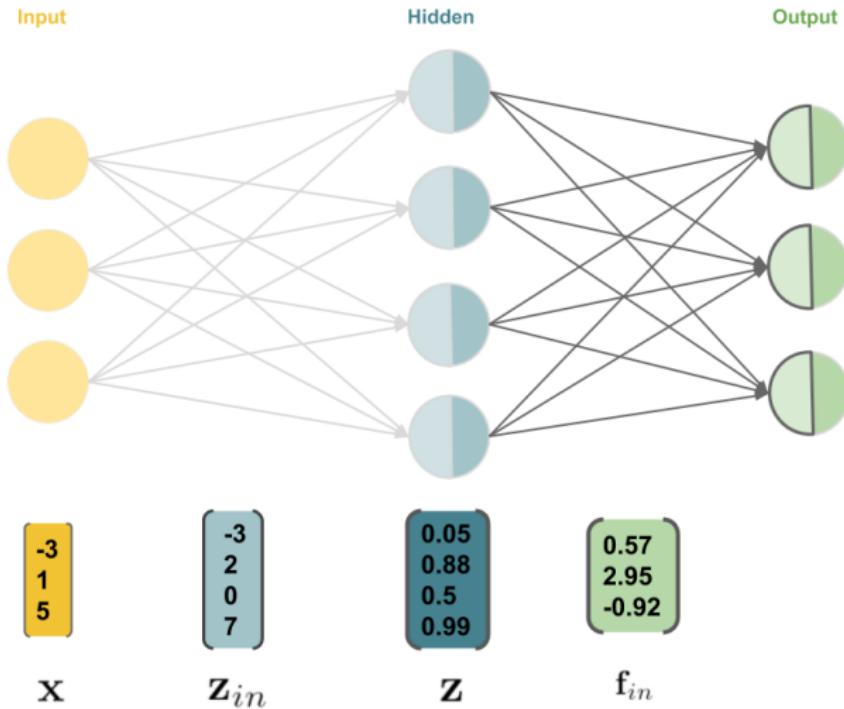
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



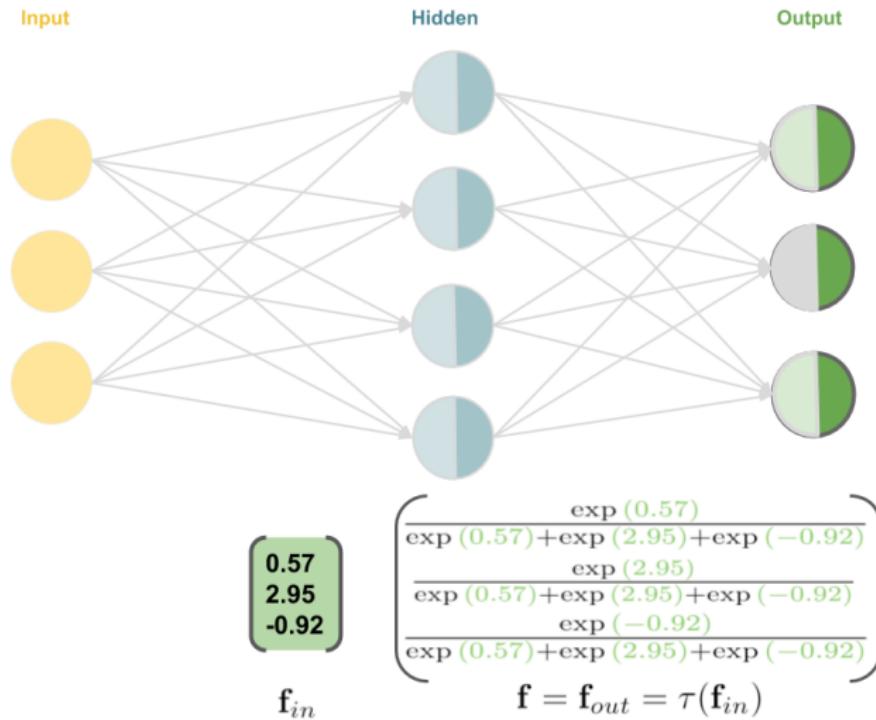
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



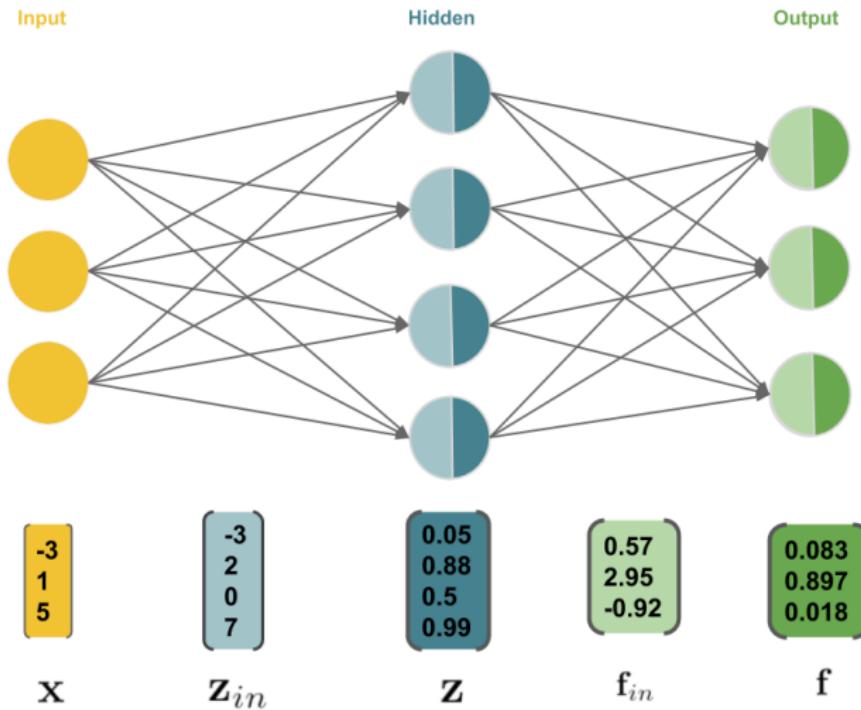
# MULTI-CLASS CLASSIFICATION: EXAMPLE

Forward pass (Hidden: Sigmoid, Output: Softmax).



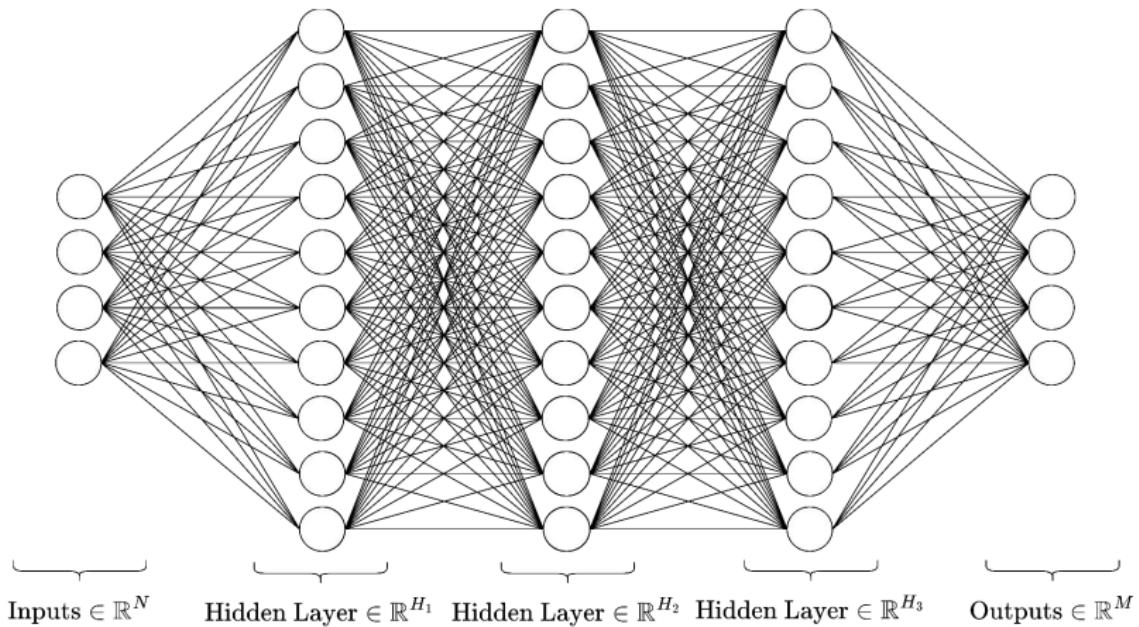
# MULTI-CLASS CLASSIFICATION: EXAMPLE

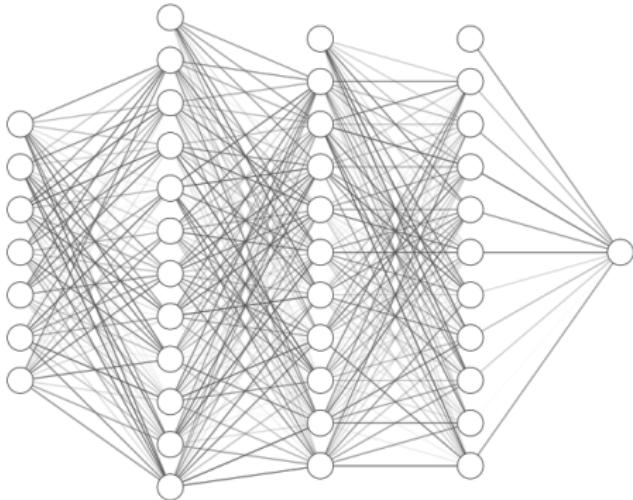
Forward pass (Hidden: Sigmoid, Output: Softmax).



# FEEDFORWARD NEURAL NETWORKS

- We will now extend the model class once again, such that we allow an arbitrary amount  $l$  of hidden layers.
- The general term for this model class is (multi-layer) **feedforward networks** (inputs are passed through the network from left to right, no feedback-loops are allowed)



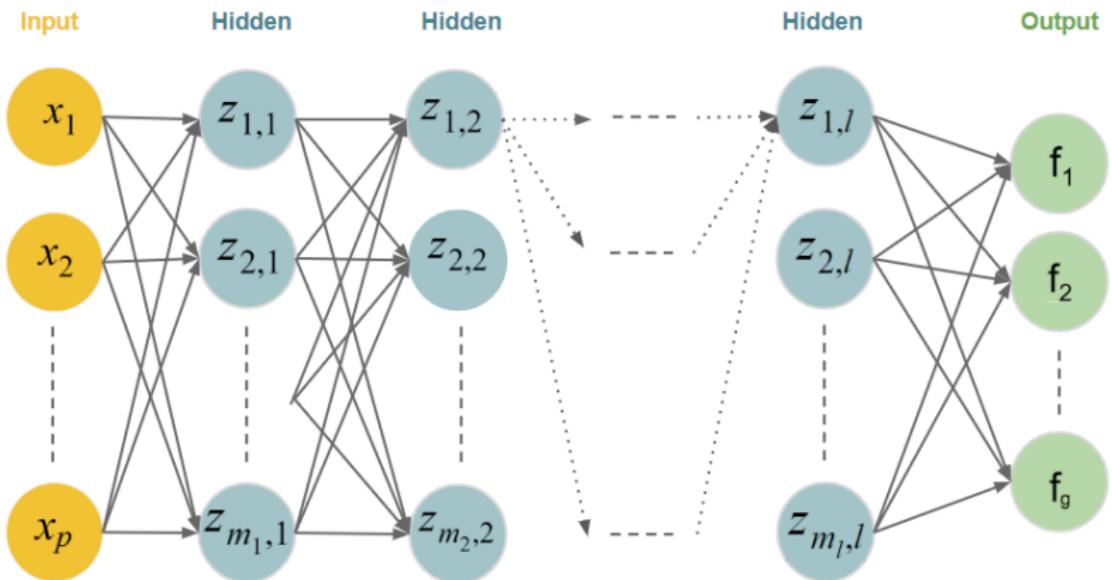


A multi-layer perceptron is a series of affine transformations of an input vector, each of which is wrapped in a non-linear activation function.

$$\mathcal{N} : \mathbb{R}^N \rightarrow \mathbb{R}^M$$
$$N, M \in \mathbb{N}$$

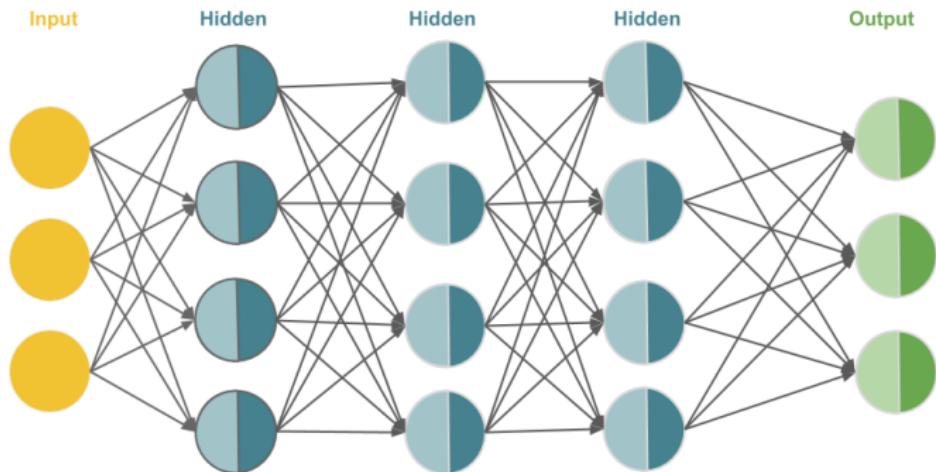
(Translation: an MLP is a fancy function)

# FEEDFORWARD NEURAL NETWORKS



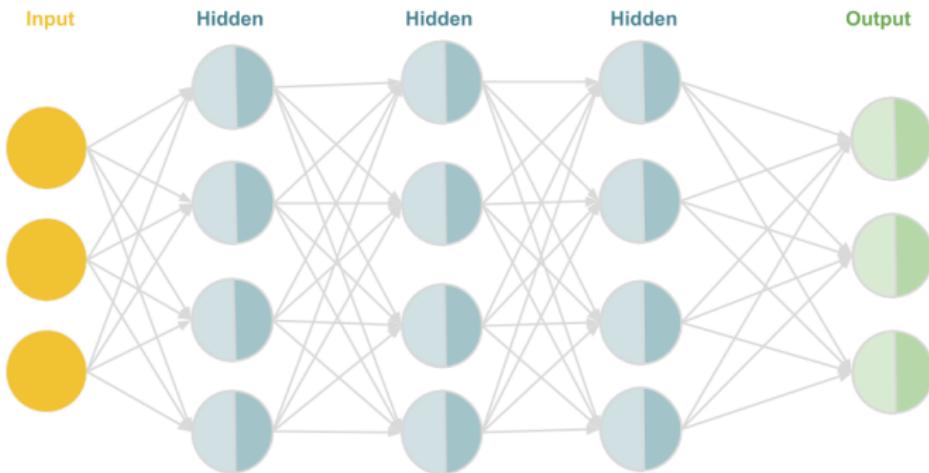
**Figure:** Structure of a deep neural network with  $l$  hidden layers (bias terms omitted).

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{pmatrix} 13 & -9 & 2 \\ -8 & 0 & 3 \\ 4 & -1 & 5 \\ -3 & 12 & 7 \end{pmatrix} \begin{pmatrix} 5 \\ -2 \\ 2 \\ 11 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & -4 & 1 \\ 0 & 11 & 2 & -14 \\ -1 & 5 & -2 & 16 \\ 0 & -9 & -3 & 4 \end{pmatrix} \begin{pmatrix} -5 \\ 3 \\ 1 \\ -8 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & -18 & -7 \\ 3 & -4 & 8 & 0 \\ -2 & 1 & 21 & 5 \\ 2 & -2 & 11 & -13 \end{pmatrix} \begin{pmatrix} 4 \\ -6 \\ 1 \\ -17 \end{pmatrix} \quad \begin{pmatrix} 9 & 3 & -1 & -4 \\ -8 & -2 & 14 & 3 \\ 13 & 2 & -9 & -1 \end{pmatrix} \begin{pmatrix} -1 \\ -4 \\ -30 \end{pmatrix}$$
$$(W^{(1)})^T \quad \mathbf{b}^{(1)} \quad (W^{(2)})^T \quad \mathbf{b}^{(2)} \quad (W^{(3)})^T \quad \mathbf{b}^{(3)} \quad U^T \quad \mathbf{c}$$

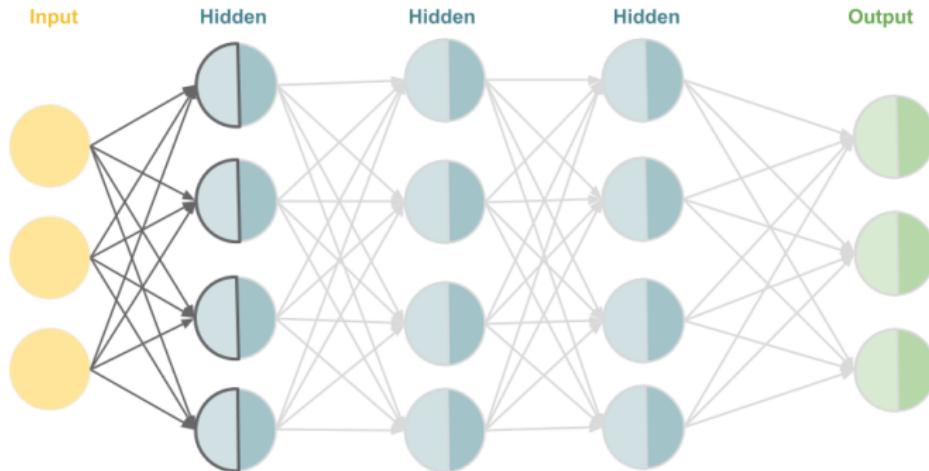
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



7  
1  
-4

x

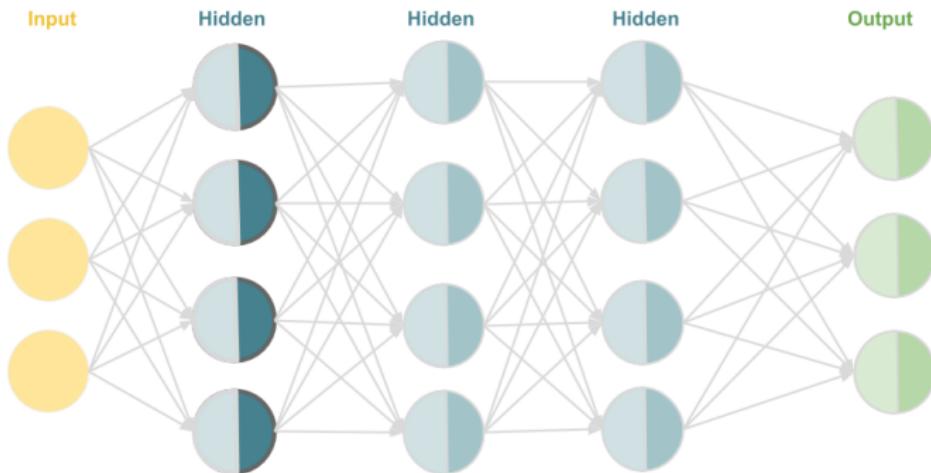
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix} \left[ \begin{array}{l} 7*13 + 1*(-9) + (-4)*2 + 5 \\ 7*(-8) + 1*0 + (-4)*3 + (-2) \\ 7*4 + 1*(-1) + (-4)*5 + 2 \\ 7*(-3) + 1*12 + (-4)*7 + 11 \end{array} \right]$$

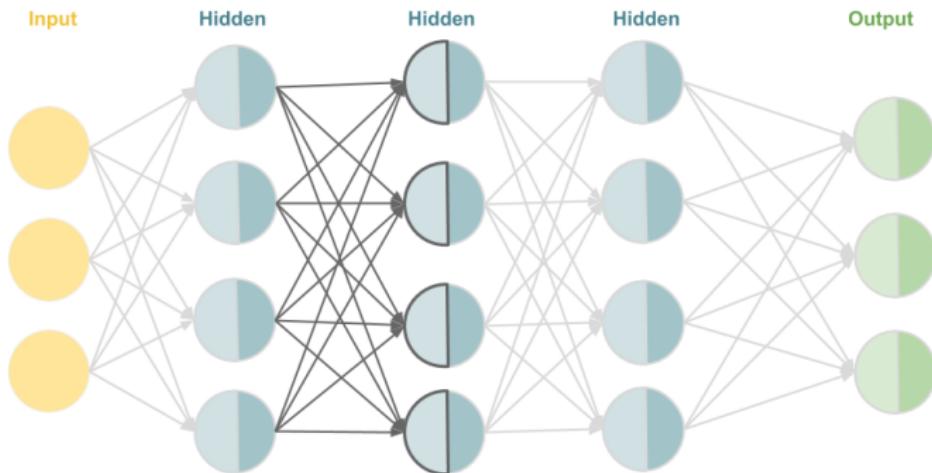
$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} = W^{(1)T} \mathbf{x} + \mathbf{b}^{(1)}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} = \mathbf{z}_{out}^{(1)} = \sigma(\mathbf{z}_{in}^{(1)})$$
$$\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix} \quad \begin{bmatrix} 79 \\ -70 \\ 9 \\ -26 \end{bmatrix} \quad \begin{bmatrix} \max(0, 79) \\ \max(0, -70) \\ \max(0, 9) \\ \max(0, -26) \end{bmatrix}$$

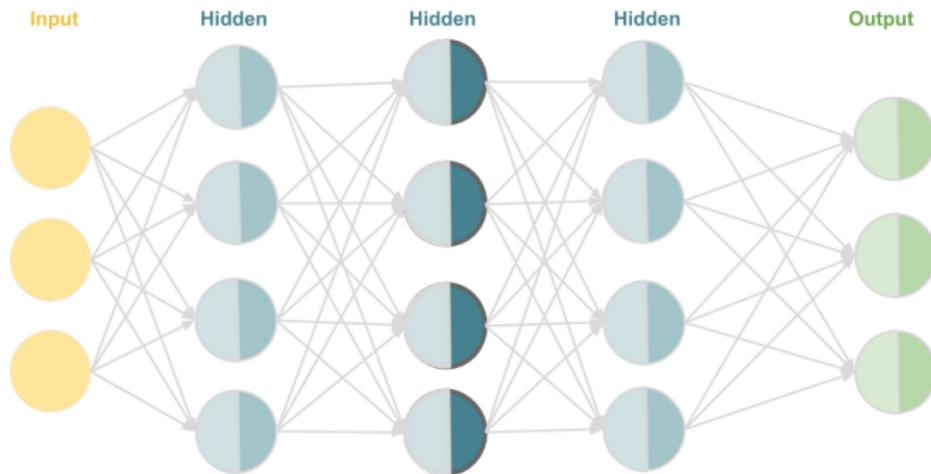
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} = W^{(2)T} \mathbf{z}^{(1)} + \mathbf{b}^{(2)}$$

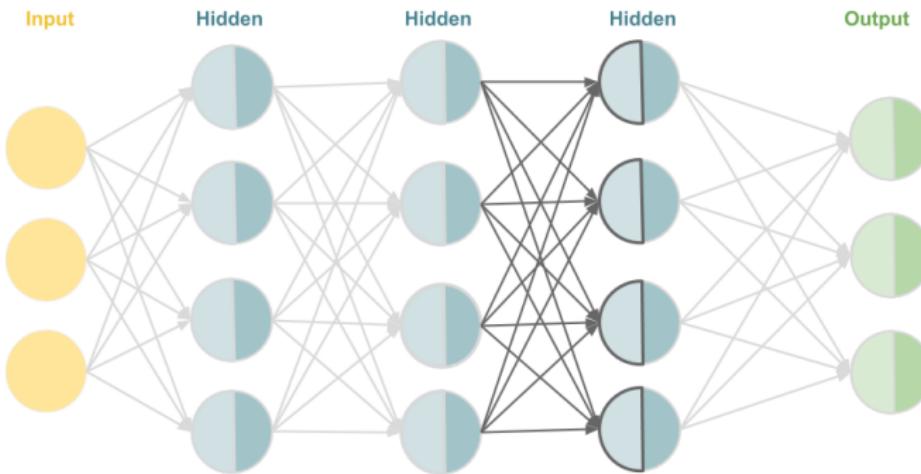
$$\begin{pmatrix} 7 \\ 1 \\ -4 \end{pmatrix} \quad \begin{pmatrix} 79 \\ -70 \\ 9 \\ -26 \end{pmatrix} \quad \begin{pmatrix} 79 \\ 0 \\ 9 \\ 0 \end{pmatrix} \quad \left( \begin{array}{l} 79 \cdot 1 + 0 \cdot 0 + 9 \cdot (-4) + 0 \cdot 1 + (-5) \\ 79 \cdot 0 + 0 \cdot 11 + 9 \cdot 2 + 0 \cdot (-14) + 3 \\ 79 \cdot (-1) + 0 \cdot 5 + 9 \cdot (-2) + 0 \cdot 16 + 1 \\ 79 \cdot 0 + 0 \cdot (-9) + 9 \cdot (-3) + 0 \cdot 4 + (-8) \end{array} \right)$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{array}{c} \mathbf{x} \\ \left[ \begin{matrix} 7 \\ 1 \\ -4 \end{matrix} \right] \\ \mathbf{z}_{in}^{(1)} \\ \left[ \begin{matrix} 79 \\ -70 \\ 9 \\ -26 \end{matrix} \right] \\ \mathbf{z}^{(1)} \\ \left[ \begin{matrix} 79 \\ 0 \\ 9 \\ 0 \end{matrix} \right] \\ \mathbf{z}_{in}^{(2)} \\ \left[ \begin{matrix} 38 \\ 21 \\ -96 \\ -35 \end{matrix} \right] \\ \mathbf{z}^{(2)} = \mathbf{z}_{out}^{(2)} = \sigma(\mathbf{z}_{in}^{(2)}) \\ \max(0, 38) \\ \max(0, 21) \\ \max(0, -96) \\ \max(0, -35) \end{array}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\mathbf{x} \quad \mathbf{z}_{in}^{(1)} \quad \mathbf{z}^{(1)} \quad \mathbf{z}_{in}^{(2)} \quad \mathbf{z}^{(2)} \quad \mathbf{z}_{in}^{(3)} = W^{(3)T} \mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

Below the input vector  $\mathbf{x}$  and the first hidden layer's input  $\mathbf{z}_{in}^{(1)}$ , their values are shown:

|    |     |
|----|-----|
| 7  | 79  |
| 1  | -70 |
| -4 | 9   |
|    | -26 |

Below the first hidden layer's output  $\mathbf{z}^{(1)}$  and the second hidden layer's input  $\mathbf{z}_{in}^{(2)}$ , their values are shown:

|    |
|----|
| 79 |
| 0  |
| 9  |
| 0  |

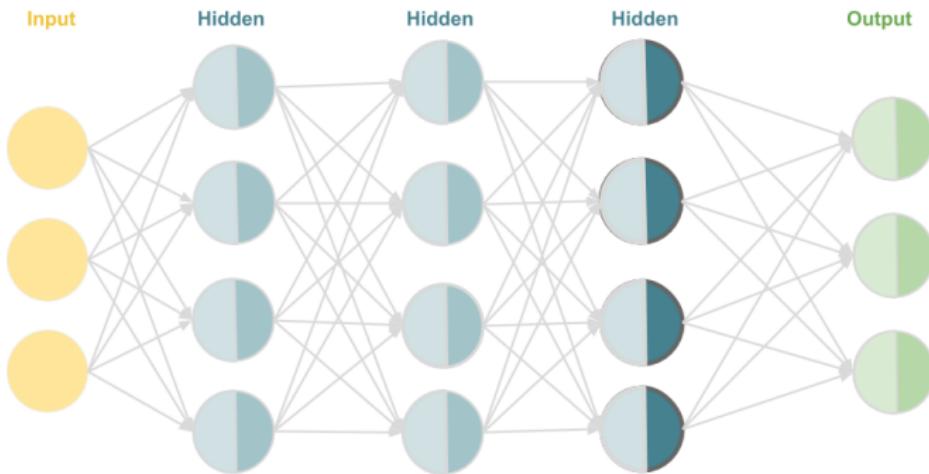
Below the second hidden layer's output  $\mathbf{z}^{(2)}$  and the third hidden layer's input  $\mathbf{z}_{in}^{(3)}$ , their values are shown:

|     |
|-----|
| 38  |
| 21  |
| -96 |
| -35 |

Below the third hidden layer's output  $\mathbf{z}^{(3)}$ , the calculation for  $\mathbf{z}_{in}^{(3)}$  is shown:

$$\begin{aligned} & 38*1 + 21*(-2) + 0*(-18) + 0*(-7) + 4 \\ & 38*3 + 21*(-4) + 0*8 + 0*0 + (-6) \\ & 38*(-2) + 21*1 + 0*21 + 0*5 + 1 \\ & 38*2 + 21*(-2) + 0*11 + 0*(-13) + (-17) \end{aligned}$$

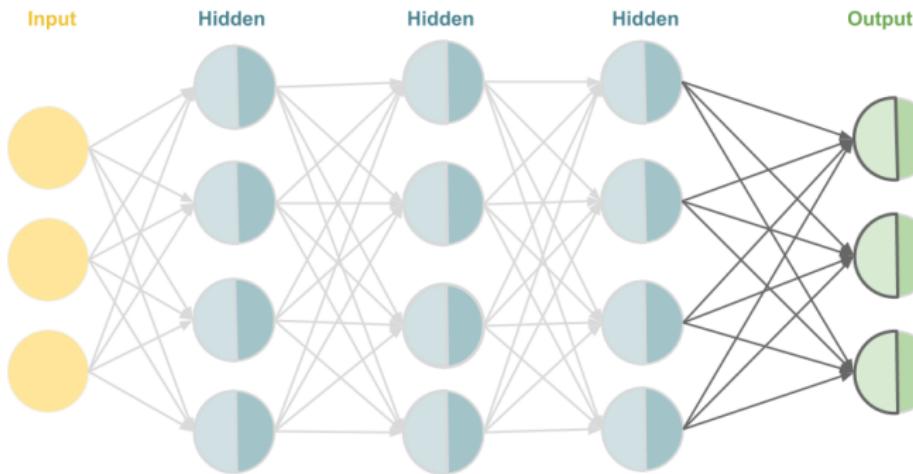
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{matrix} \mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} & \mathbf{z}^{(2)} & \mathbf{z}_{in}^{(3)} & \mathbf{z}^{(3)} = \mathbf{z}_{out}^{(3)} = \sigma(\mathbf{z}_{in}^{(3)}) \end{matrix}$$

|    |     |    |     |    |     |                |
|----|-----|----|-----|----|-----|----------------|
| 7  | 79  | 79 | 38  | 38 | 0   | $\max(0, 0)$   |
| 1  | -70 | 0  | 21  | 21 | 24  | $\max(0, 24)$  |
| -4 | 9   | 9  | -96 | 0  | -54 | $\max(0, -54)$ |
|    | -26 | 0  | -35 | 0  | 17  | $\max(0, 17)$  |

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



$$\begin{matrix} \mathbf{x} & \mathbf{z}_{in}^{(1)} & \mathbf{z}^{(1)} & \mathbf{z}_{in}^{(2)} & \mathbf{z}^{(2)} & \mathbf{z}_{in}^{(3)} & \mathbf{z}^{(3)} & \mathbf{f}_{in} = U^T \mathbf{z}^{(3)} + \mathbf{c} \end{matrix}$$

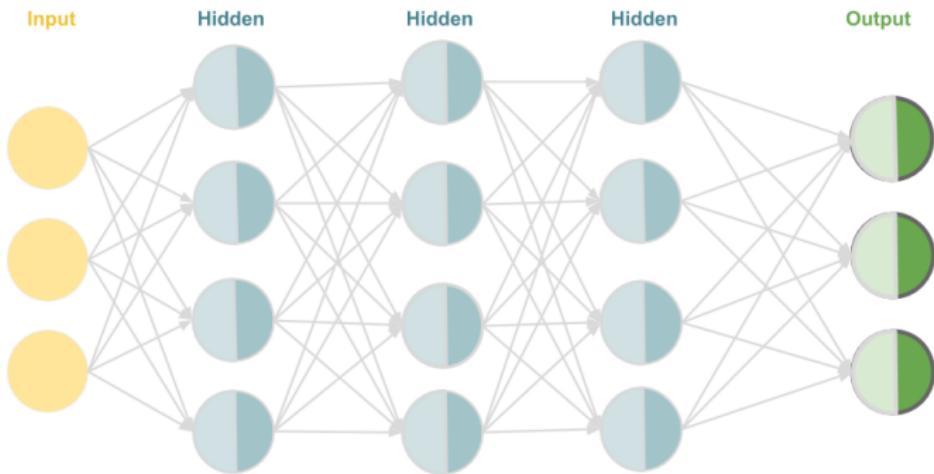
Below the input vector  $\mathbf{x}$  and hidden states  $\mathbf{z}_{in}^{(1)}, \mathbf{z}^{(1)}, \mathbf{z}_{in}^{(2)}, \mathbf{z}^{(2)}, \mathbf{z}_{in}^{(3)}, \mathbf{z}^{(3)}$ , their corresponding calculated values are shown in brackets:

$$\begin{aligned} \mathbf{z}_{in}^{(1)} &= \begin{pmatrix} 79 \\ -70 \\ 9 \\ -26 \end{pmatrix} \\ \mathbf{z}^{(1)} &= \begin{pmatrix} 79 \\ 0 \\ 9 \\ 0 \end{pmatrix} \\ \mathbf{z}_{in}^{(2)} &= \begin{pmatrix} 38 \\ 21 \\ -96 \\ -35 \end{pmatrix} \\ \mathbf{z}^{(2)} &= \begin{pmatrix} 38 \\ 21 \\ 0 \\ 0 \end{pmatrix} \\ \mathbf{z}_{in}^{(3)} &= \begin{pmatrix} 0 \\ 24 \\ -54 \\ 17 \end{pmatrix} \\ \mathbf{z}^{(3)} &= \begin{pmatrix} 0 \\ 24 \\ 0 \\ 17 \end{pmatrix} \end{aligned}$$

Calculated values for  $\mathbf{f}_{in}$ :

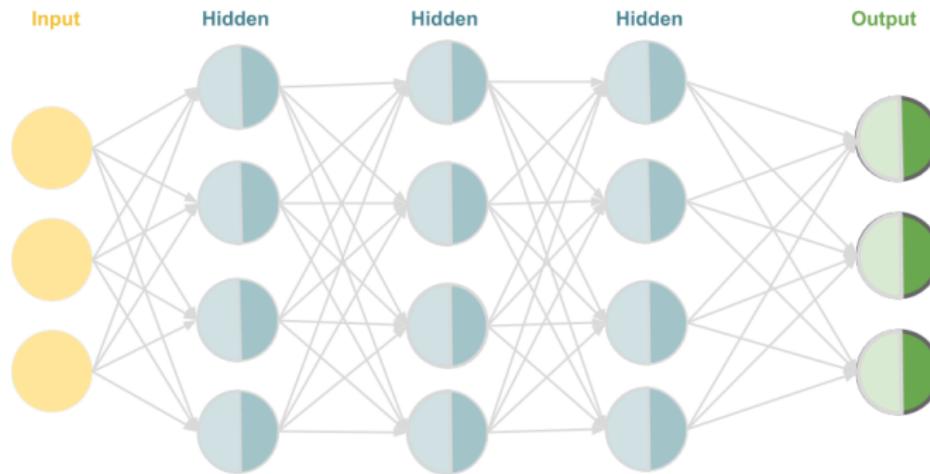
$$\begin{aligned} \mathbf{f}_{in} &= \begin{pmatrix} 24*3 + 17*(-4) + (-1) \\ 24*(-2) + 17*3 + (-4) \\ 24*2 + 17*(-1) + (-30) \end{pmatrix} \end{aligned}$$

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



|  |   |   |  |  |  |  |  |
|--|---|---|--|--|--|--|--|
| $\mathbf{x}$                                 | $\mathbf{z}_{in}^{(1)}$                               | $\mathbf{z}^{(1)}$                                | $\mathbf{z}_{in}^{(2)}$                                | $\mathbf{z}^{(2)}$                                 | $\mathbf{z}_{in}^{(3)}$                              | $\mathbf{z}^{(3)}$                                 | $\mathbf{f}_{in}$                            |
| $\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ -70 \\ 9 \\ -26 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ 0 \\ 9 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ -96 \\ -35 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ 0 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ -54 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ 0 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$ |

# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



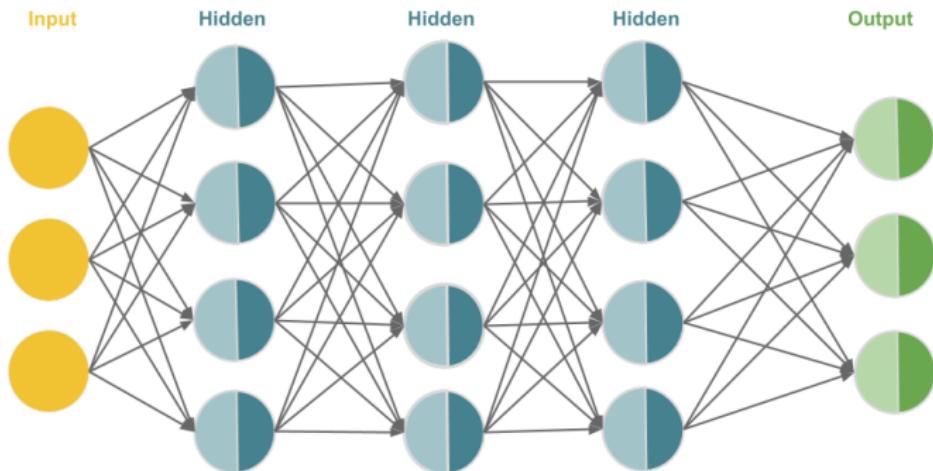
$$\begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$$

$\mathbf{f}_{in}$

$$\left\{ \begin{array}{l} \frac{\exp(3)}{\exp(3) + \exp(-1) + \exp(1)} \\ \frac{\exp(-1)}{\exp(3) + \exp(-1) + \exp(1)} \\ \frac{\exp(1)}{\exp(3) + \exp(-1) + \exp(1)} \end{array} \right.$$

$$\mathbf{f} = \mathbf{f}_{out} = \tau(\mathbf{f}_{in})$$

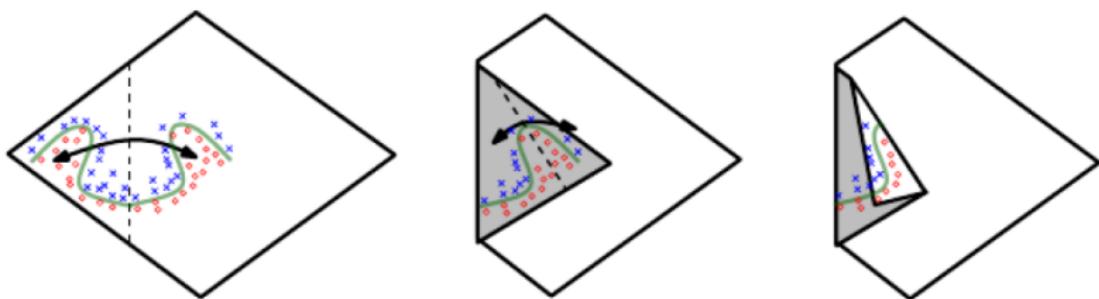
# FEEDFORWARD NEURAL NETWORKS: EXAMPLE



|  |   |   |  |  |  |  |  |   |
|--|---|---|--|--|--|--|--|---|
| $x$  | $z_{in}^{(1)}$  | $z^{(1)}$   | $z_{in}^{(2)}$   | $z^{(2)}$  | $z_{in}^{(3)}$                                       | $z^{(3)}$  | $f_{in}$                                     | $f$   |
| $\begin{bmatrix} 7 \\ 1 \\ -4 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ -70 \\ 9 \\ -26 \end{bmatrix}$ | $\begin{bmatrix} 79 \\ 0 \\ 9 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ -96 \\ -35 \end{bmatrix}$ | $\begin{bmatrix} 38 \\ 21 \\ 0 \\ 0 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ -54 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 0 \\ 24 \\ 0 \\ 17 \end{bmatrix}$ | $\begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}$ | $\begin{bmatrix} 0.866 \\ 0.015 \\ 0.117 \end{bmatrix}$ |

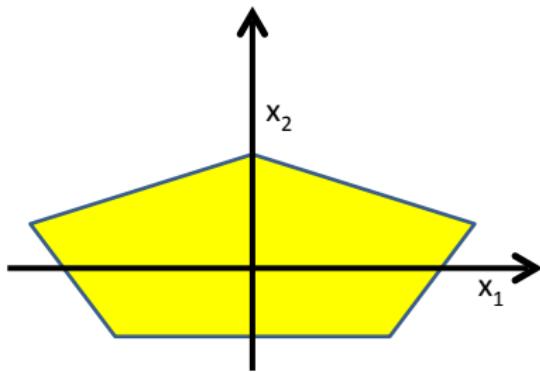
# WHY ADD MORE LAYERS?

- Multiple layers allow for the extraction of more and more abstract representations.
- Each layer in a feed-forward neural network adds its own degree of non-linearity to the model.



**Figure:** An intuitive, geometric explanation of the exponential advantage of deeper networks formally (Montúfar et al. (2014)).

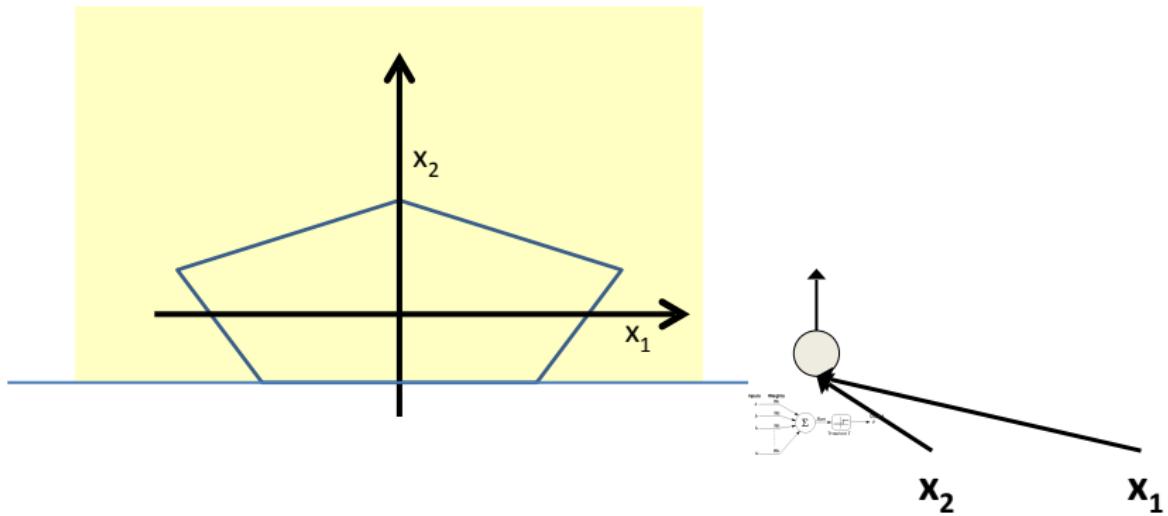
# Composing complicated “decision” boundaries



Can now be composed into  
“networks” to compute arbitrary  
classification “boundaries”

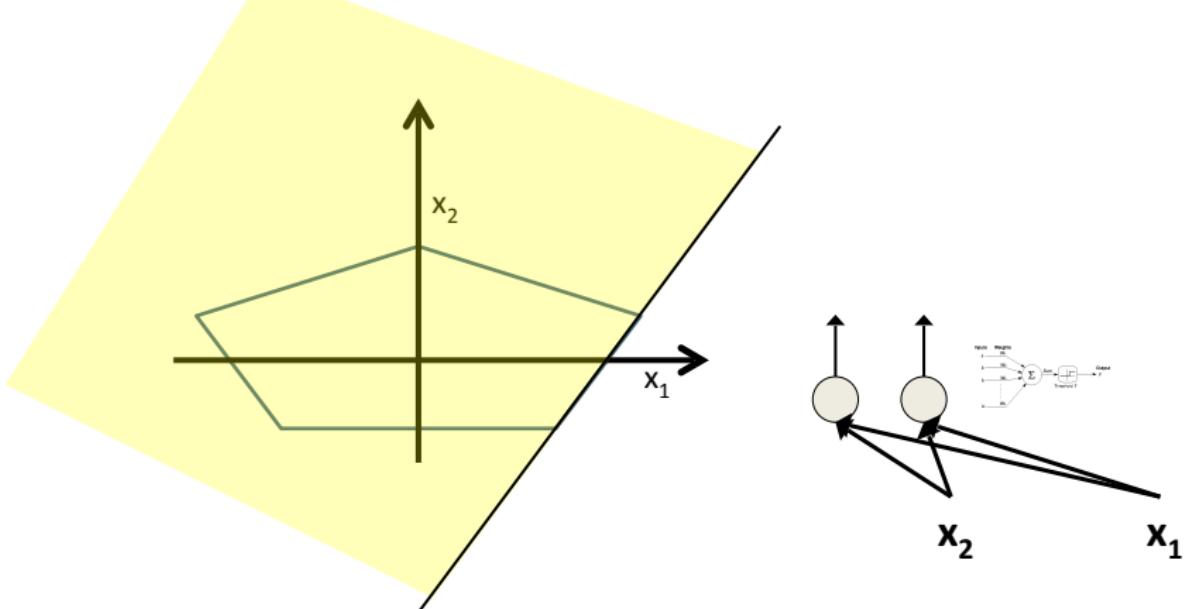
- Build a network of units with a single output that fires if the input is in the coloured area

# Booleans over the reals



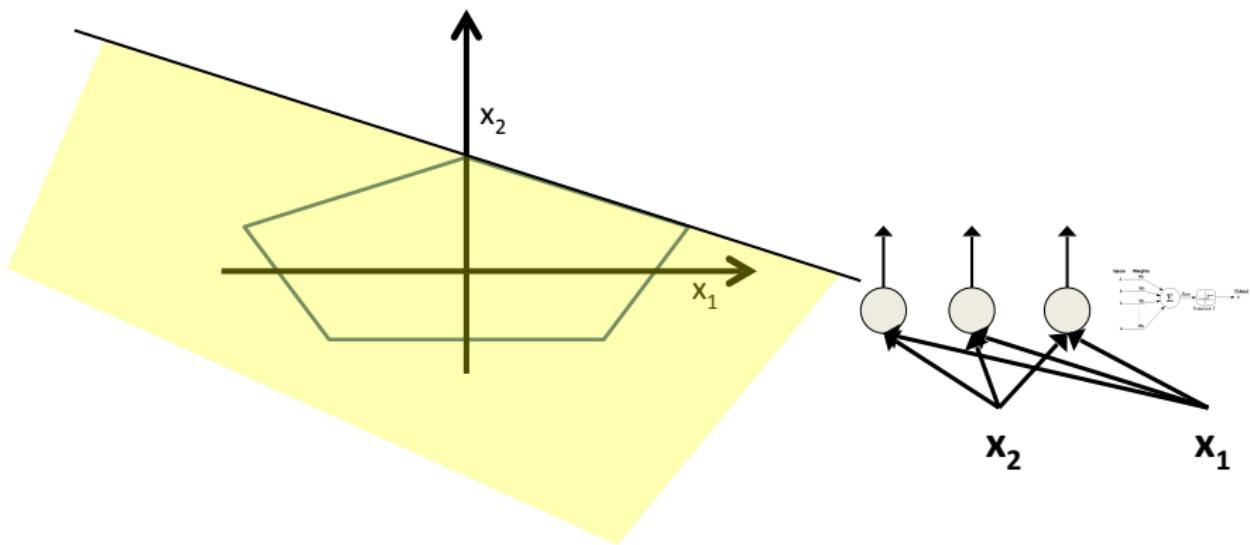
- The network must fire if the input is in the coloured area

# Booleans over the reals



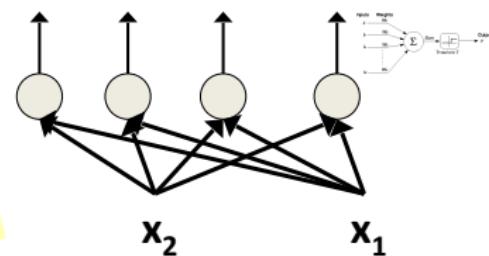
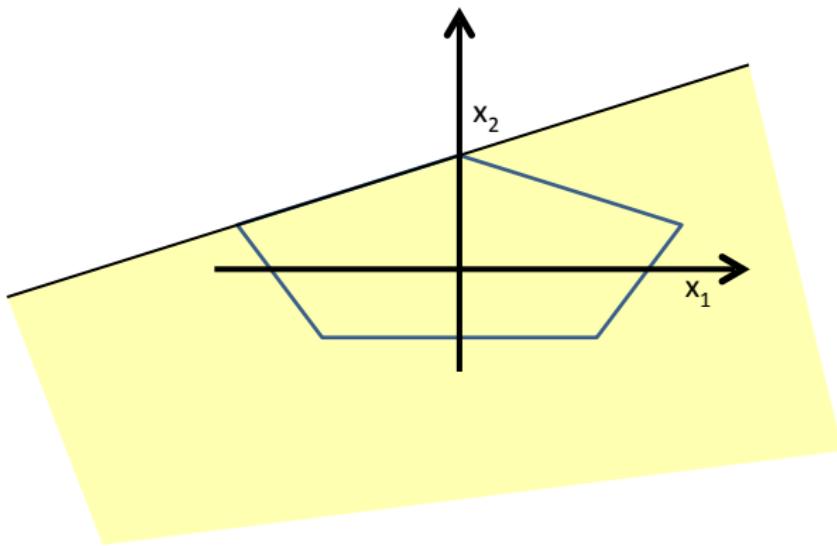
- The network must fire if the input is in the coloured area

# Booleans over the reals



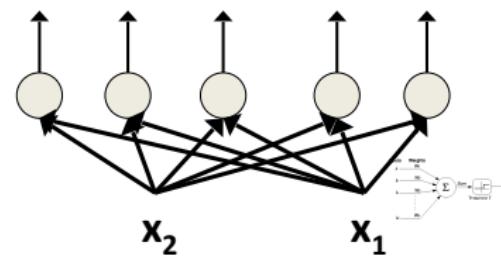
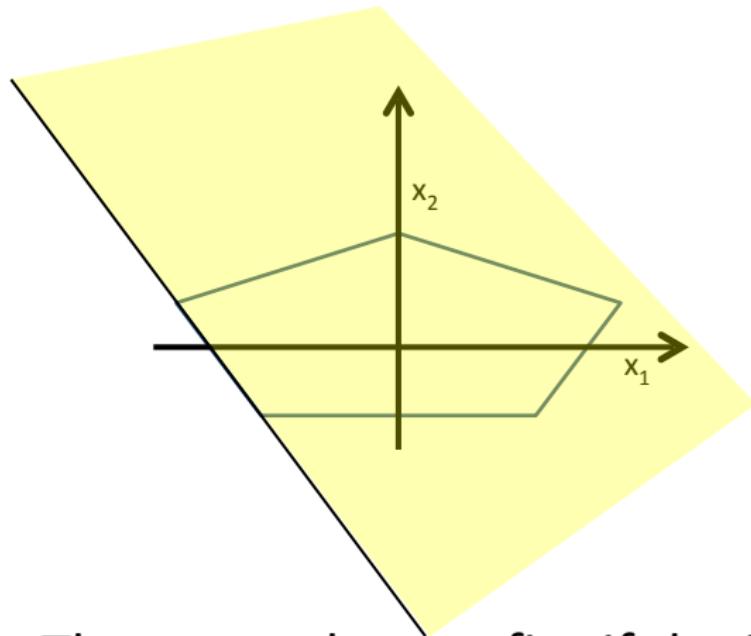
- The network must fire if the input is in the coloured area

# Booleans over the reals



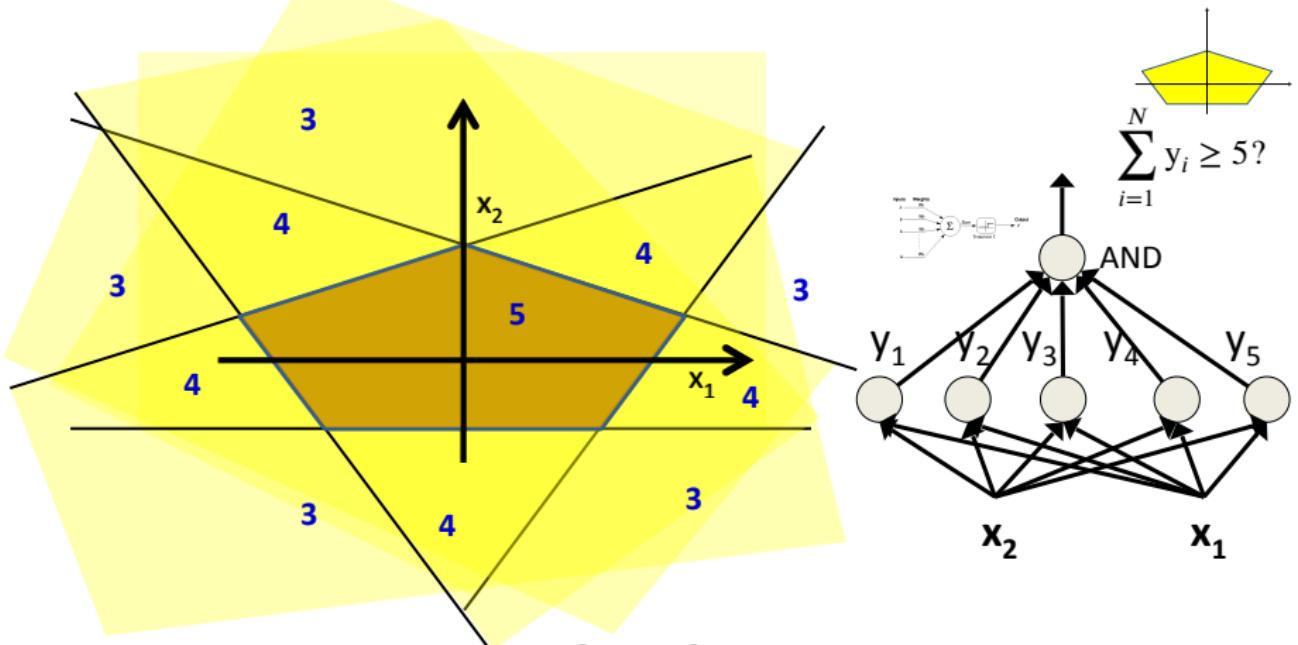
- The network must fire if the input is in the coloured area

# Booleans over the reals



- The network must fire if the input is in the coloured area

# Booleans over the reals



- The network must fire if the input is in the coloured area

---

## Implementing learning: Gradient Descent

Given:

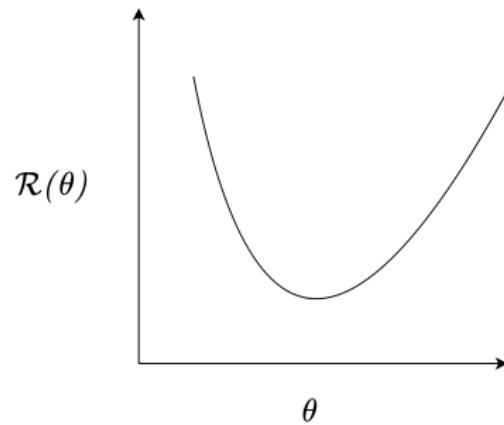
- Family of parameters  $\Theta$  (e.g. possible weights of a NN)
- Differentiable risk function  $\mathcal{R}(\theta)$

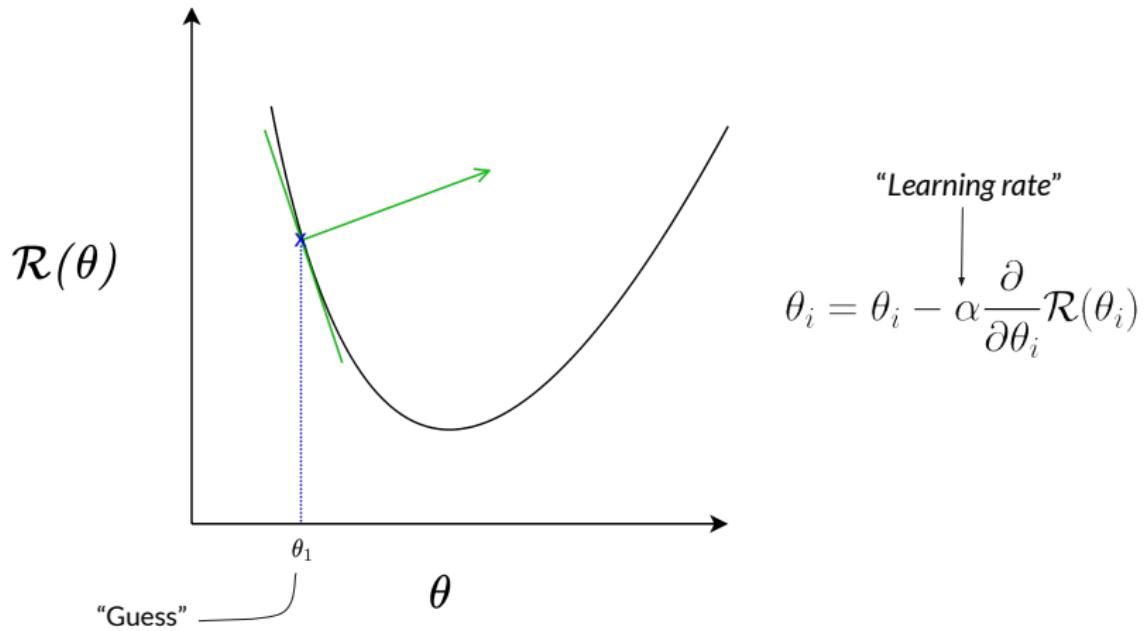
Goal:

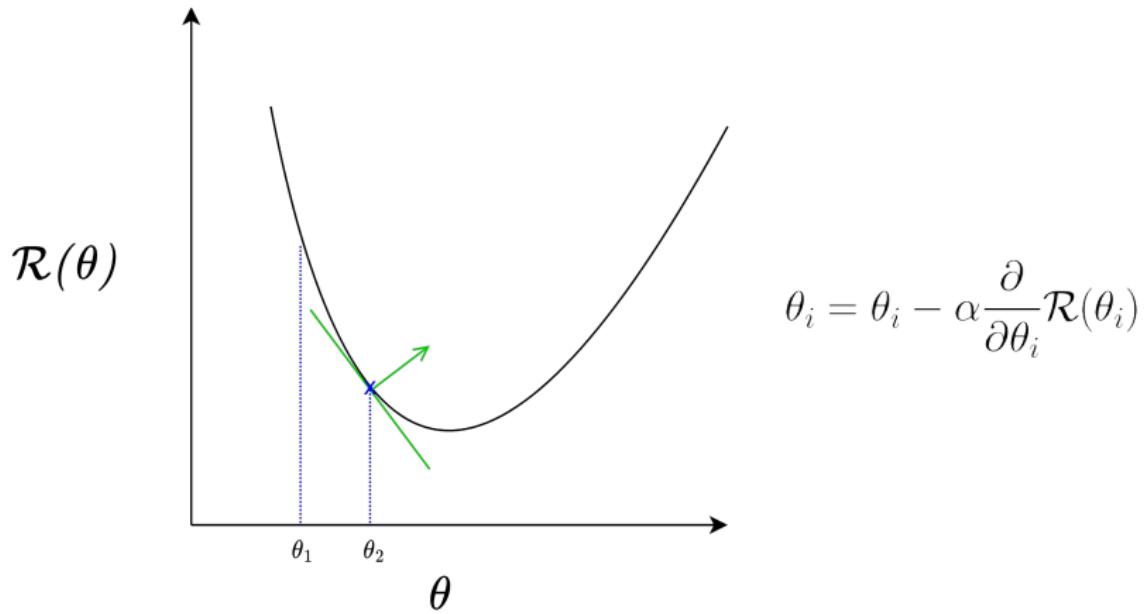
$$\theta_{opt} = \operatorname{argmin}_{\theta \in \Theta} \mathcal{R}(\theta)$$

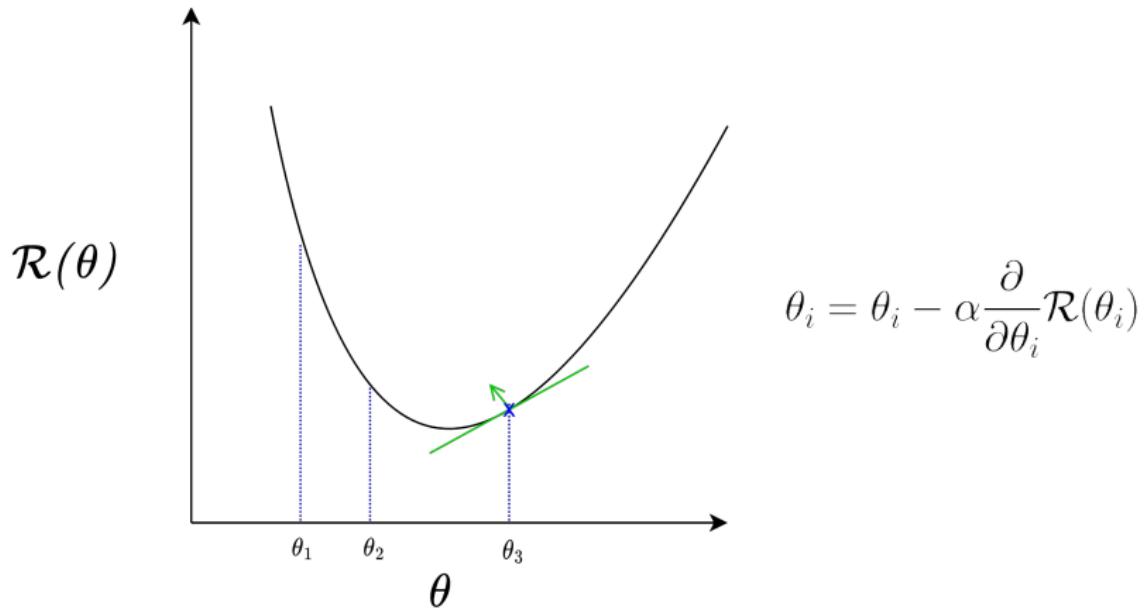
---

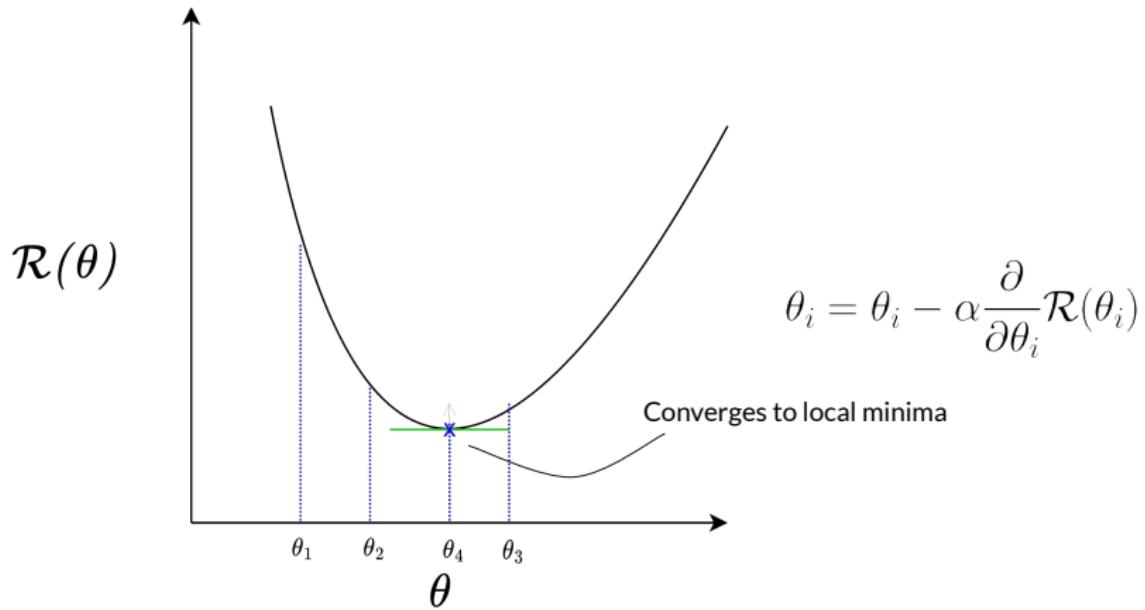
## Backprop: Gradient descent





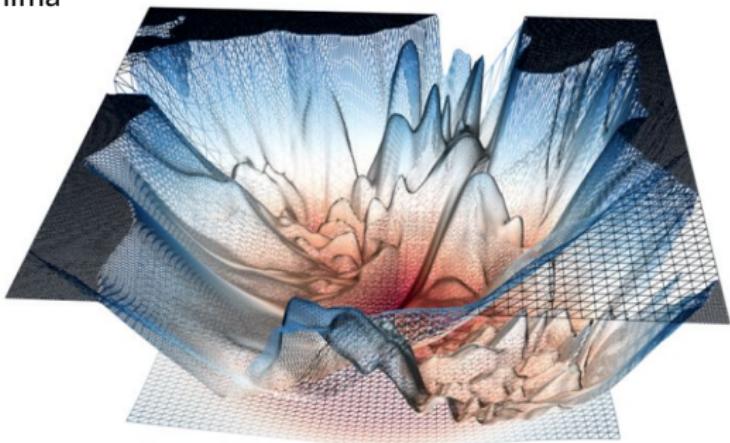






A “real” loss landscape:

- Many (many many) local minima
- Saddle points



<http://www.telesens.co/2019/01/16/neural-network-loss-visualization/>

Moreover, as the number of the parameters grows with the network depth, the computations become exponentially large. How to deal with it?

*(see in the next lecture)*

## Further reading

- ▶ Amazing short lecture series by 3Blue1Brown,  
chapter: [1](#), [2](#), [3](#)
- ▶ Neural Network interactive playground:  
<https://playground.tensorflow.org>
- ▶ Visual explanation of Neural Networks:  
<https://mlu-explain.github.io/neural-networks>