

# Neural Methods for NLP

Master LiTL --- 2023-2024

[chloe.braud@irit.fr](mailto:chloe.braud@irit.fr)

[https://gitlab.irit.fr/melodi/andiamo/teaching\\_cbraud/master\\_litl](https://gitlab.irit.fr/melodi/andiamo/teaching_cbraud/master_litl)

**Course 2: Introduction to Deep Learning**



# Schedule 2023-2024

1	28.11	13h30-15h30	2	(C1) ML Reminder + <del>(TP-ML)</del> + TP POO
2	05.12	13h30-15h30	2	(C2) Intro DL + TP2
3	12.12	13h30-16h	2.5	(C3) Embeddings + TP3
4	19.12	13h30-16h	2.5	TP4 + Start projects
(holidays)				
5	09.01	13h-16h	3	(C4) Training a NN + TP5 + TP6 →Assignments Part 1 due
6	16.01	13h-16h	3	(C5) CNN, RNN + TP7 + TP8
7	23.01	13h-16h	3	Projects
8	01.02	13h-16h	3	(C6) Encoder-decoder, transformer + TP9
9	06.02	13h-16h	3	(C7) Current challenges → Assignments Part 2 due + project defenses
	ou 13?			

# Neural methods for NLP

- 1980's: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990's: 'Statistical' NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- ≈ 2010: 'Neural' NLP
  - combine linear and non-linear functions, over dense inputs
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret ("black box")



# Neural methods for NLP

- 1980's: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990's: 'Statistical' NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- ≈ 2010: 'Neural' NLP
  - **combine linear and non-linear functions, over dense inputs**
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret ("black box")



# Neural methods for NLP

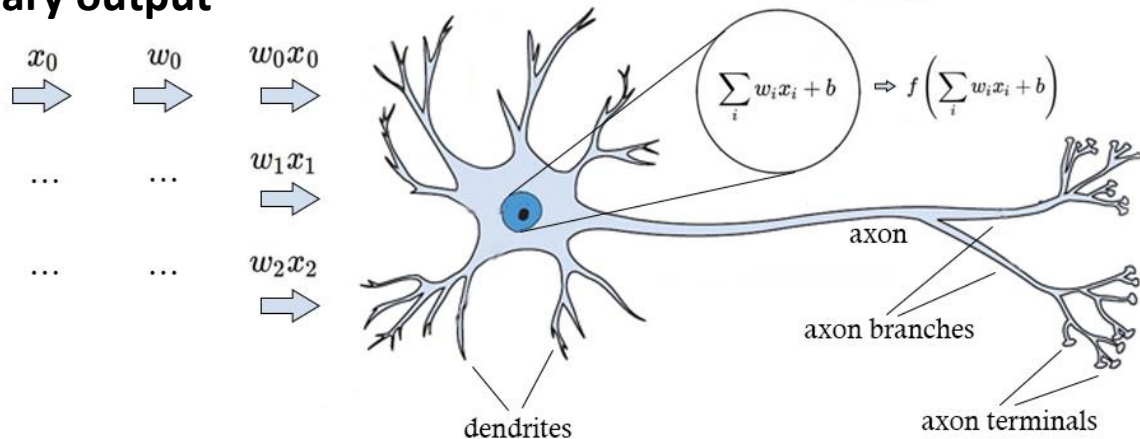
- 1980's: Symbolic NLP
  - rule-based approach, hand-written rules
  - advantages: based on linguistics expertise, very precise
  - inconvenients: lack of coverage, time consuming
- 1990's: 'Statistical' NLP
  - learn rules automatically = (mostly linear) functions, with high-dimensional, sparse feature vectors
  - large annotated corpora
  - handcrafted features
  - rather fast to train, still good baselines
- ≈ 2010: 'Neural' NLP
  - **combine linear and non-linear functions, over dense inputs**
  - (very) large annotated corpora and very large unannotated corpora
  - improved performance (in general), no feature engineering
  - harder to interpret ("black box")



# A bit more of history: The brain inspired metaphor

Brain's computation is based on computation units called neurons (Perceptron, Rosenblatt, 1957):

- A neuron has scalar inputs and outputs
- Each input has an associated **weight** to control its importance
- The neuron multiplies each input by its weight and then sums them = **linear combination**
- If the weighted sum is greater than the activation potential, the neuron is said to “fire” = produce a **single binary output**
- The neurons are connected to each other, forming a **network**: the output of a neuron may feed into the inputs of one or more neurons

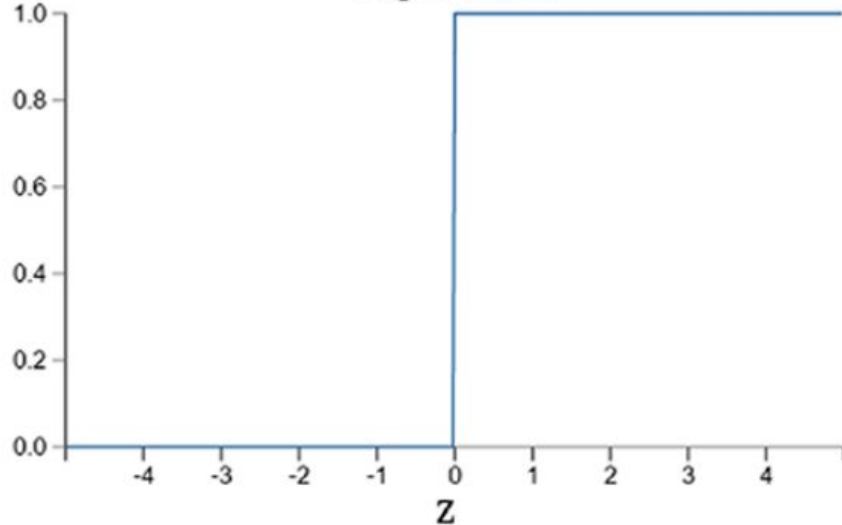


# Artificial neuron

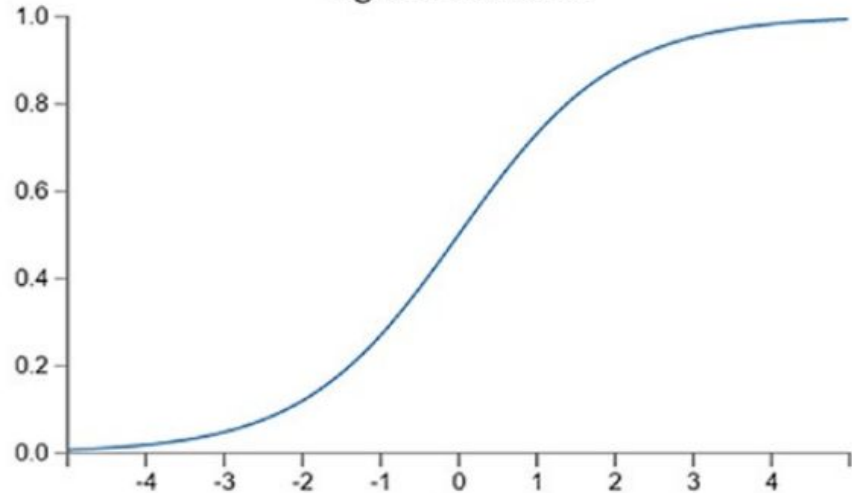
Using a binary output: not very practical

→ We prefer having small change in weight leading to small change in output

step function



sigmoid function



# 'Statistical' vs 'neural' models

Standard approach:

- **linear model** trained over high-dimensional but **very sparse** feature vectors

Neural approach:

- **non-linear** neural networks over **dense input vectors**



# Content

Introduction to Deep Learning

1. Introducing non linearity
2. Feed-forward architecture
3. Common activation functions
4. Output Transformation functions

**Practical session 2:** walk through code in PyTorch

---

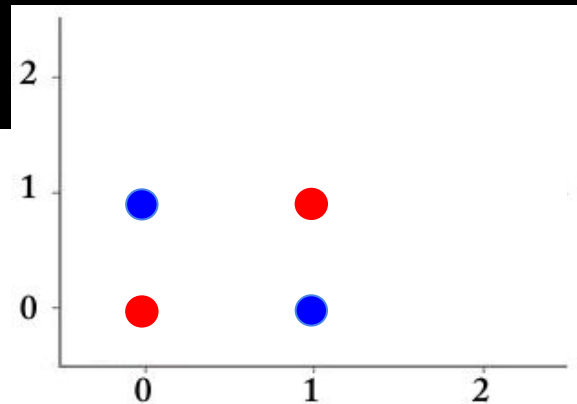
# Introducing non-linearity

## XOR problem: exclusive “or”

→ return a true value if the two inputs are not equal and a false value if they are equal.

But it's impossible to find  $\mathbf{w}$ ,  $b$  such that:

- $(0,0) \cdot \mathbf{w} + b < 0$
- $(0,1) \cdot \mathbf{w} + b \geq 0$
- $(1,0) \cdot \mathbf{w} + b \geq 0$
- $(1,1) \cdot \mathbf{w} + b < 0$



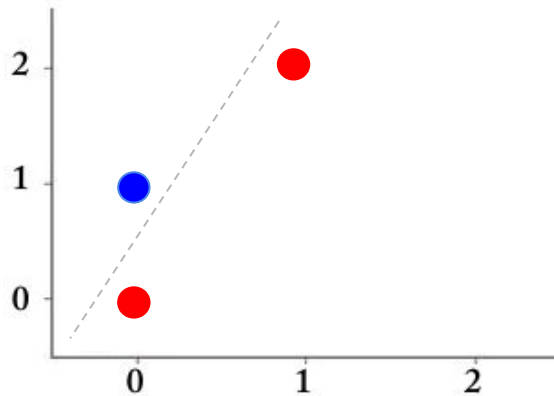
Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

# Solution: non-linear input transformations

If we transform the points using:  $\phi(\mathbf{x}_1, \mathbf{x}_2) = [\mathbf{x}_1 \times \mathbf{x}_2, \mathbf{x}_1 + \mathbf{x}_2]$

The problem becomes linearly separable:

- $\phi(0,0) = (0, 0) \rightarrow \text{red}$
- $\phi(0,1) = (0, 1) \rightarrow \text{blue}$
- $\phi(1,0) = (0, 1) \rightarrow \text{blue}$
- $\phi(1,1) = (1, 2) \rightarrow \text{red}$



The function  $\phi$  mapped the data into a representation that is suitable for linear classification, we can find:

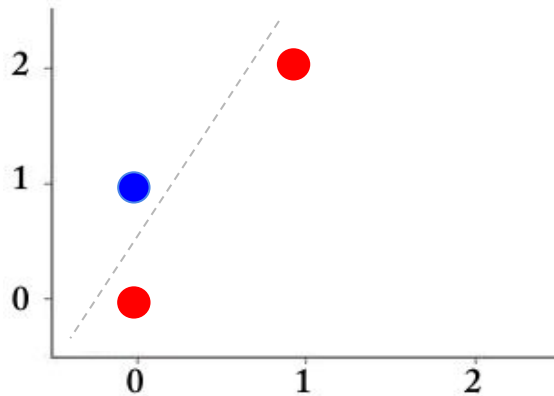
$$f(\mathbf{x}) = \phi(\mathbf{x}).\mathbf{W} + \mathbf{b}$$

# Solution: non-linear input transformations

If we transform the points using:  $\phi(\mathbf{x}_1, \mathbf{x}_2) = [\mathbf{x}_1 \times \mathbf{x}_2, \mathbf{x}_1 + \mathbf{x}_2]$

The problem becomes linearly separable:

- $\phi(0,0) = (0, 0) \rightarrow \textcolor{red}{0}$
- $\phi(0,1) = (0, 1) \rightarrow \textcolor{blue}{1}$
- $\phi(1,0) = (0, 1) \rightarrow \textcolor{blue}{1}$
- $\phi(1,1) = (1, 2) \rightarrow \textcolor{red}{0}$



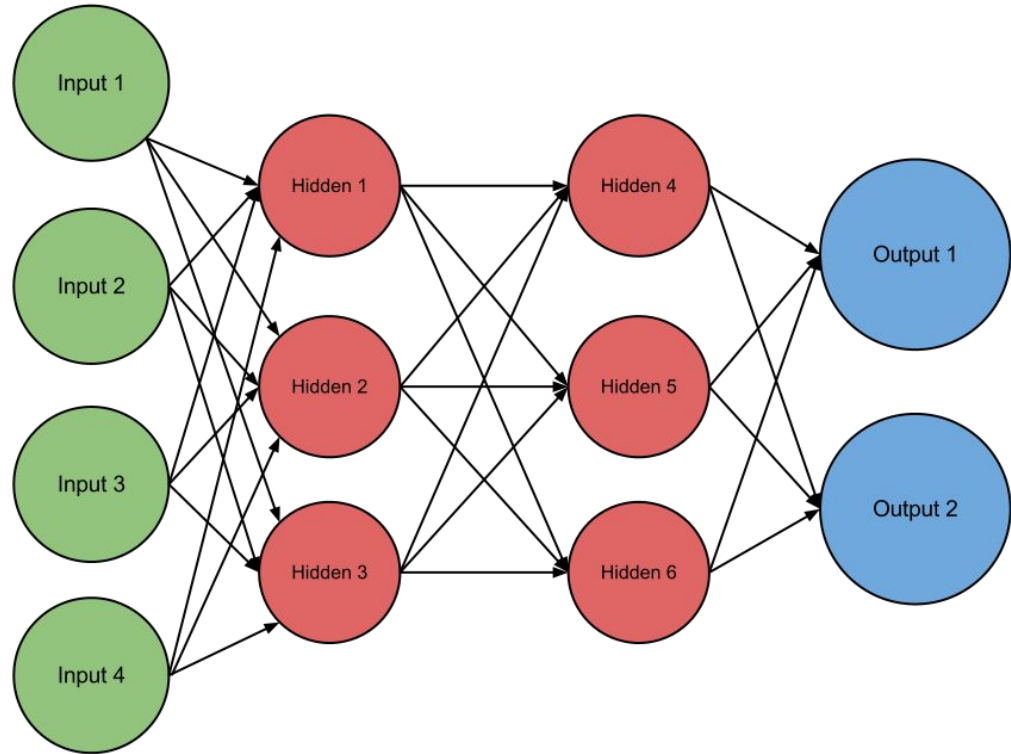
The function  $\phi$  mapped the data into a representation that is suitable for linear classification, we can find:

$$f(\mathbf{x}) = \phi(\mathbf{x}) \cdot \mathbf{W} + \mathbf{b}$$

Note: here the transformed data has the same dimension as the original, but often we'll need to map to a higher dimensional space.

Note: SVM = defining *a priori* generic mapping functions vs NN = trainable mapping functions

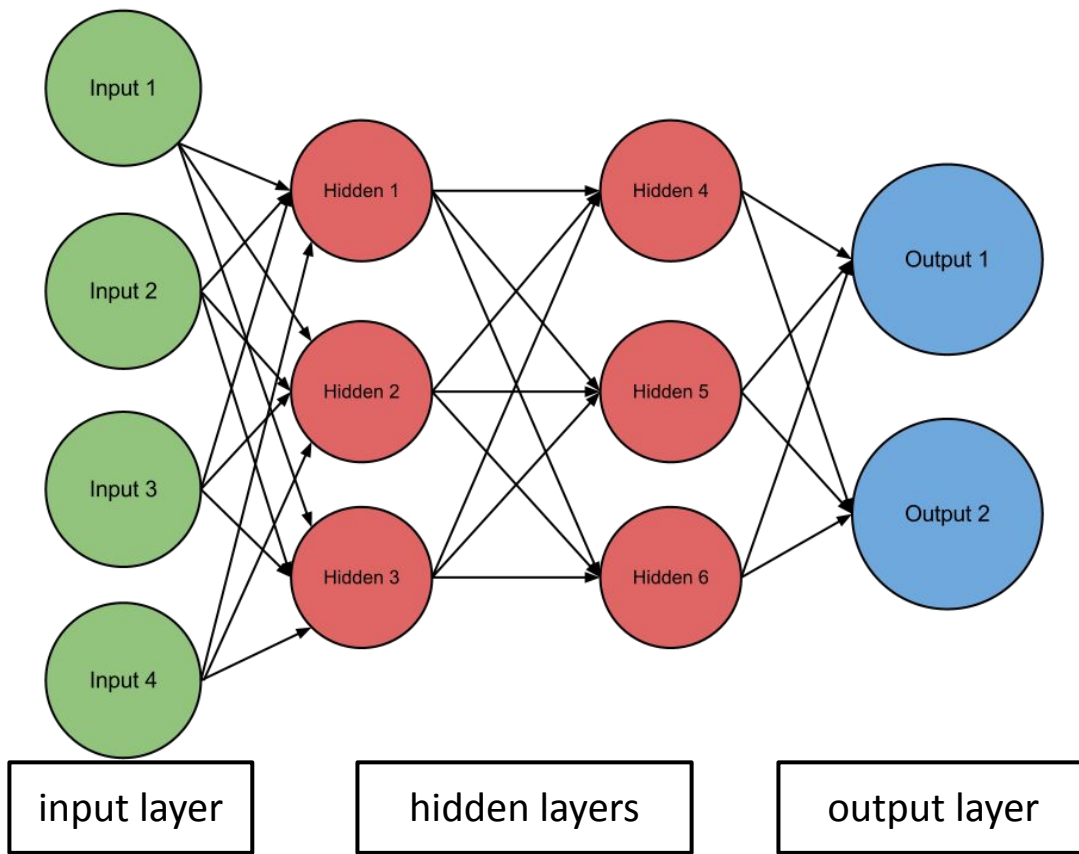
# Feed-Forward Architecture



# Feed-Forward Architecture

## Multi-layer Perceptron

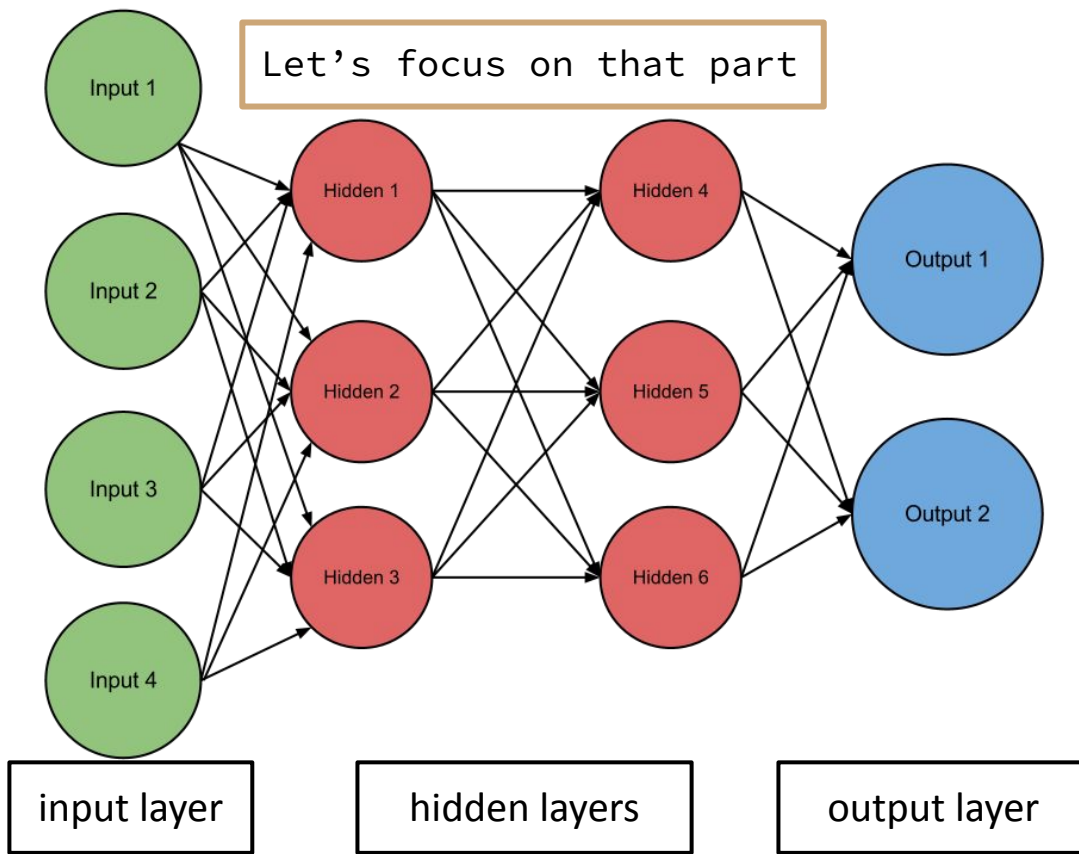
- best known, standard neural network approach
- Fully connected layers
- Can be used as drop-in replacement for typical classifiers



# Feed-Forward Architecture

## Multi-layer Perceptron

- best known, standard neural network approach
- Fully connected layers
- Can be used as drop-in replacement for typical classifiers



# Neural networks: basics

- Layers are made of neurons = building blocks of neural networks
- A single neuron works like logistic regression, we know that!



# Neural networks: basics

- Layers are made of neurons = building blocks of neural networks
- A **single neuron works like logistic regression**, we know that!

Reminder: LR model yields **probability** of an instance belonging to a particular class **based on the instance's features weighted by the model's parameters**

# Logistic Regression

- Spam (binary) classification
- We take word frequency as features

feature $f_i$	weight $w_i$
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

# Logistic Regression

- Spam (binary) classification
- We take word frequency as features

$$\text{score}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

$\mathbf{x}_1$ : “**Pharmacie** en ligne: **viagra** **meilleure** **offre**!”

$$\begin{aligned} \text{score}(\mathbf{x}_1) &= 0.4 \times 1 + 1.2 \times 1 + 0.2 \times 1 + 0.2 \times 1 + (-0.8) \times 0 + (-1.7) \\ &\times 0 + 0.1 = 2.1 \end{aligned}$$

feature $f_i$	weight $w_i$
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

# Logistic Regression

- Spam (binary) classification
- We take word frequency as features

$$\text{score}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

$\mathbf{x}_1$ : “**Pharmacie** en ligne: **viagra** **meilleure** **offre**!”

$$\text{score}(\mathbf{x}_1) = 0.4 \times 1 + 1.2 \times 1 + 0.2 \times 1 + 0.2 \times 1 + (-0.8) \times 0 + (-1.7) \times 0 + 0.1 = 2.1$$

feature $f_i$	weight $w_i$
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

# Logistic Regression

- Spam (binary) classification
- We take word frequency as features

$$\text{score}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

$\mathbf{x}_1$ : “**Pharmacie** en ligne: **viagra** **meilleure** **offre**!”

$$\text{score}(\mathbf{x}_1) = 0.4 \times 1 + 1.2 \times 1 + 0.2 \times 1 + 0.2 \times 1 + (-0.8) \times 0 + (-1.7) \times 0 + 0.1 = 2.1$$

feature $f_i$	weight $w_i$
pharmacie	0.4
viagra	1.2
meilleure	0.2
offre	0.2
demande	-0.8
transmets	-1.7
bias	0.1

# Logistic Regression

- Spam (binary) classification
- We take word frequency as features

$$\text{score}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

$\mathbf{x}_1$ : “**Pharmacie** en ligne: **viagra** **meilleure** **offre**!”

$$\text{score}(\mathbf{x}_1) = 0.4 \times 1 + 1.2 \times 1 + 0.2 \times 1 + 0.2 \times 1 + (-0.8) \times 0 + (-1.7) \times 0 + 0.1 = 2.1$$

$\mathbf{x}_2$ : “Suite à votre demande, je vous transmets notre meilleure offre”

$$\text{score}(\mathbf{x}_2) = 0.4 \times 0 + 1.2 \times 0 + 0.2 \times 1 + 0.2 \times 1 + (-0.8) \times 1 + (-1.7) \times 1 + 0.1 = -2.0$$

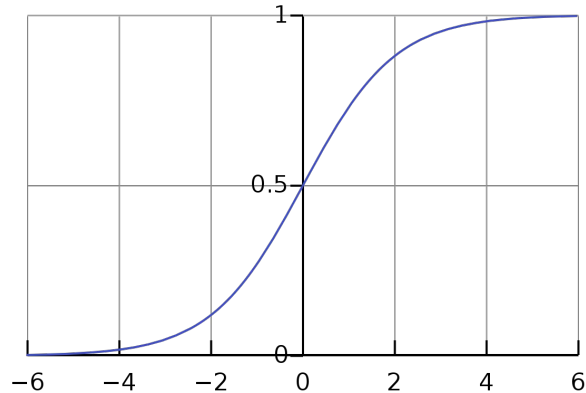
feature $f_i$	weight $w_i$
<b>pharmacie</b>	<b>0.4</b>
viagra	1.2
meilleure	0.2
offre	0.2
<b>demande</b>	<b>-0.8</b>
transmets	-1.7
bias	0.1

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear logistic function**  $\rightarrow$  range  $[0,1]$



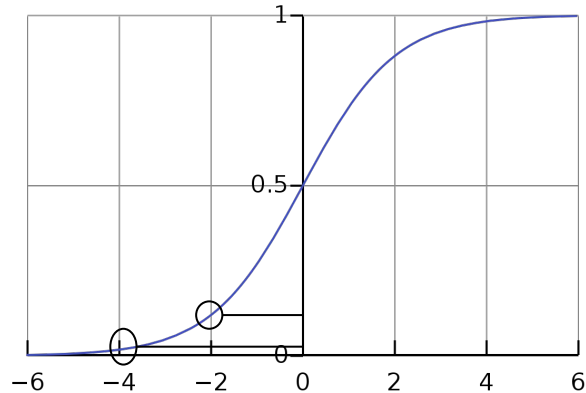
what is a non-linear function ?

- the corresponding graph is not a line: **“a function that does not graph into a straight line and does not have a constant slope”**



# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear logistic function**  $\rightarrow$  range  $[0,1]$



what is a non-linear function ?

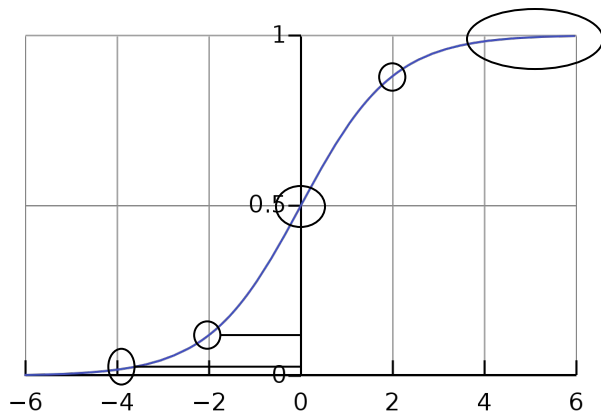
- the corresponding graph is not a line: “a function that does not graph into a straight line and does not have a constant slope”

how do we get probabilities?

- map the values to the range we want e.g.:
  - -4 here is mapped to stg close to 0
  - -2 still below 0.5
  - 0 mapped to ?
  - 2 mapped to ?
  - closer to 4-6 mapped to ?

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear logistic function**  $\rightarrow$  range  $[0,1]$



what is a non-linear function ?

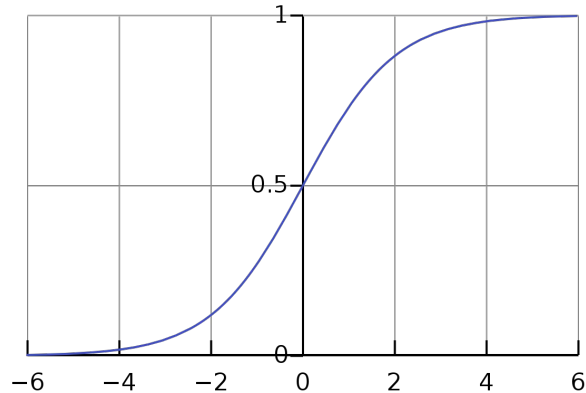
- the corresponding graph is not a line: “a function that does not graph into a straight line and does not have a constant slope”

how do we get probabilities?

- map the values to the range we want e.g.:
  - -4 here is mapped to stg close to 0
  - -2 still below 0.5
  - 0 mapped to ?  $\rightarrow$  0.5
  - 2 mapped to ?  $\rightarrow$  close to 1
  - closer to 4-6 mapped to ?  $\rightarrow$  closer and closer to 1

# Logistic Regression

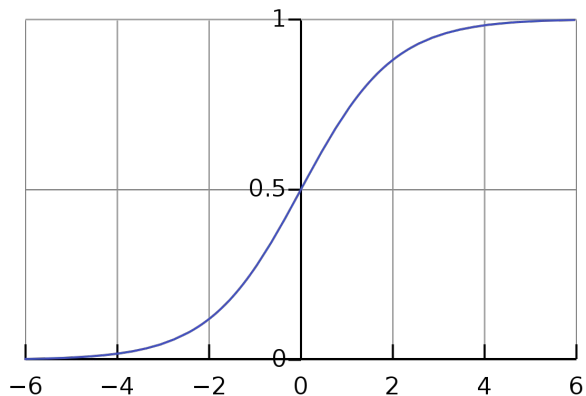
- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear *logistic function***  $\rightarrow$  range  $[0,1]$



logistic function  $\left( \frac{1}{1+e^{-x}} \right)$

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear *logistic function***  $\rightarrow$  range  $[0,1]$



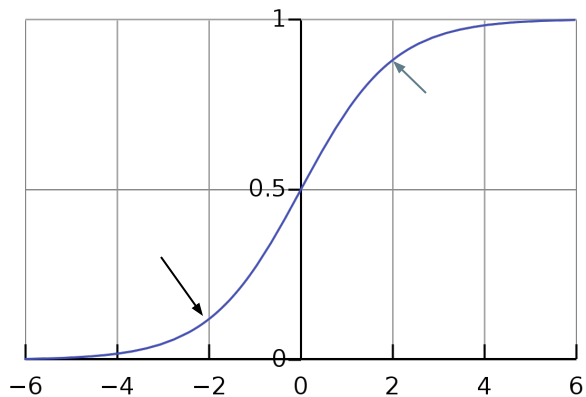
The logistic function does exactly these desired computations:

- It maps an input real number  $\rightarrow [0, 1]$
- Large negative number  $\rightarrow 0$
- Large positive number  $\rightarrow 1$

$$\text{logistic function } \left( \frac{1}{1+e^{-x}} \right)$$

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear logistic function**  $\rightarrow$  range  $[0,1]$



logistic function  $\left(\frac{1}{1+e^{-x}}\right)$

The logistic function does exactly these desired computations:

- It maps an input real number  $\rightarrow [0, 1]$
- Large negative number  $\rightarrow 0$
- Large positive number  $\rightarrow 1$

i.e. from our previous example on spam:

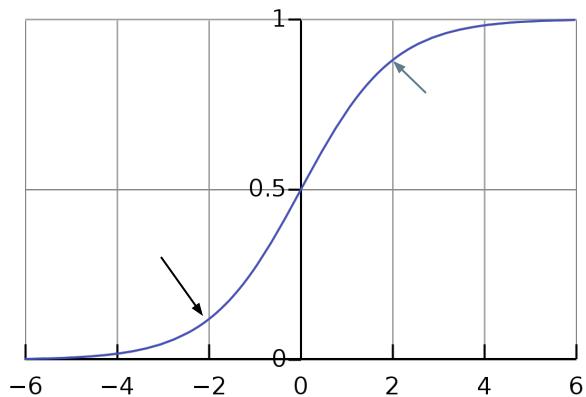
$$f(\text{score}(x_1)) = \frac{1}{1+e^{2.1}} = .89$$

$$f(\text{score}(x_2)) = \frac{1}{1+e^{-2}} = .12$$

**SPAM!**

# Logistic Regression

- Linear scores in range  $[-\infty, +\infty]$ : difficult to interpret, we'd rather have **probabilities**
- Linear scores are transformed using **non-linear logistic function**  $\rightarrow$  range  $[0,1]$



logistic function  $\left(\frac{1}{1+e^{-x}}\right)$

The logistic function does exactly these desired computations:

- It maps an input real number  $\rightarrow [0, 1]$
- Large negative number  $\rightarrow 0$
- Large positive number  $\rightarrow 1$

i.e. from our previous example on spam:

$$f(\text{score}(x_1)) = \frac{1}{1+e^{2.1}} = .89$$

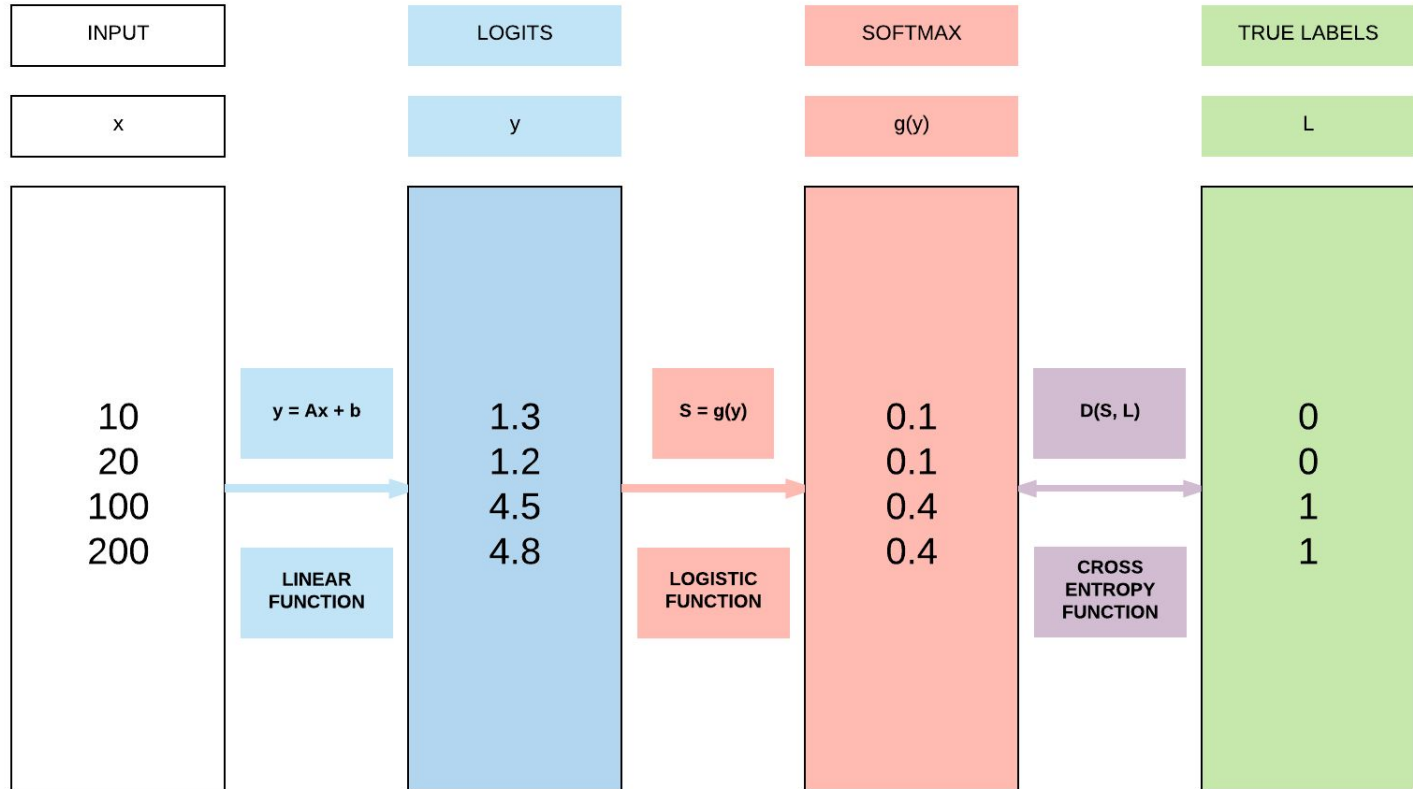
**SPAM**

$$f(\text{score}(x_2)) = \frac{1}{1+e^{-2}} = .12$$

$$\text{score}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$$

$\rightarrow \text{score}(\mathbf{x}) = \mathbf{f}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$  avec  $f$  une fct non linéaire

# Logistic Regression



# Logistic Regression

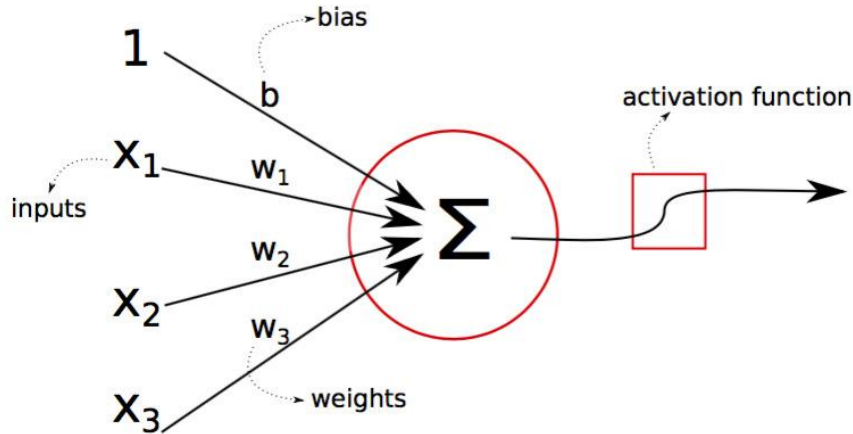
softmax = generalisation of the **logistic function** taking a **vector** as input





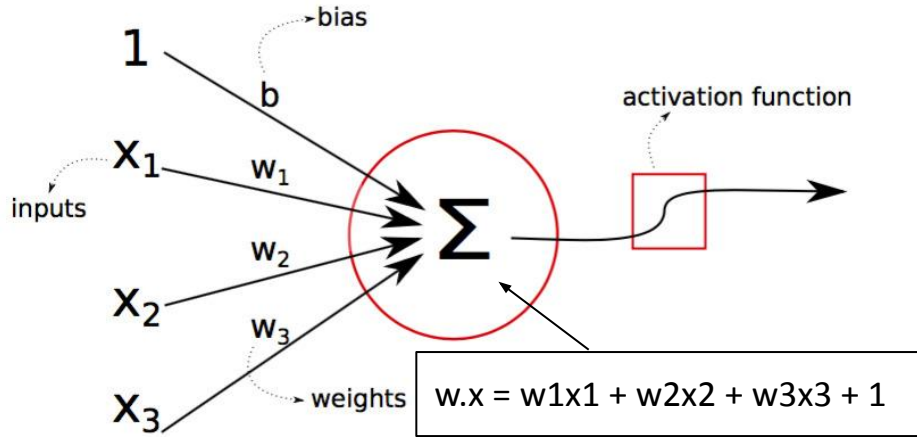
# Neuron = Logistic Regression

These are the exact computations of a single neuron



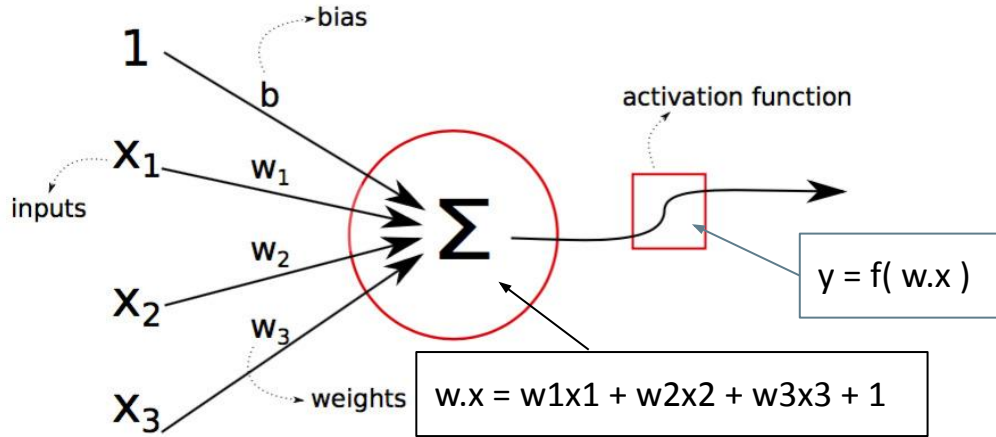
# Neuron = Logistic Regression

These are the exact computations of a single neuron



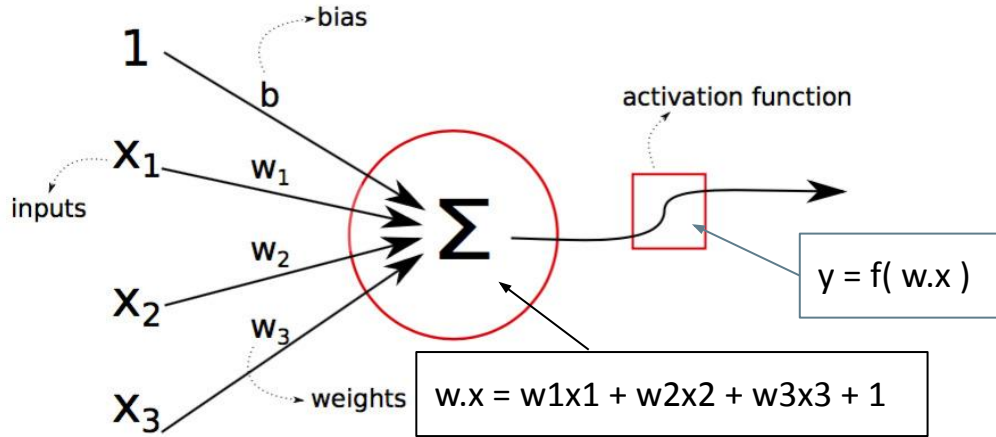
# Neuron = Logistic Regression

These are the exact computations of a single neuron



# Neuron = Logistic Regression

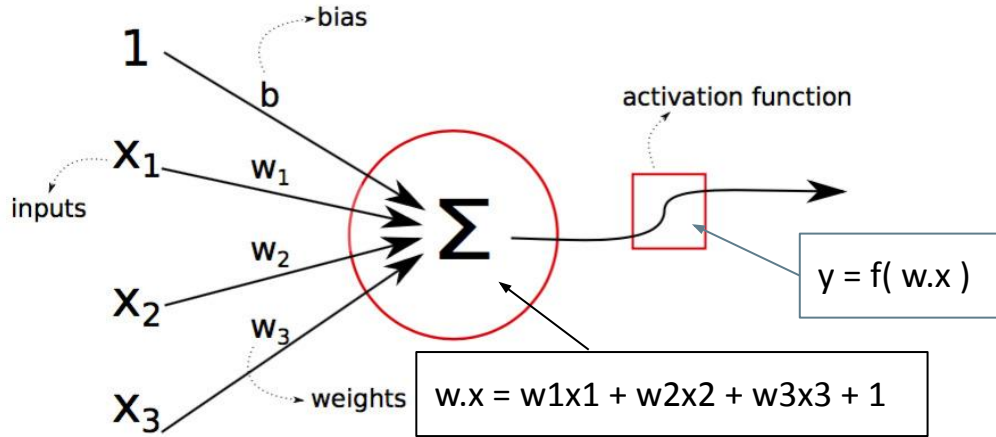
These are the exact computations of a single neuron



- We can feed an input vector to a bunch of LR functions and get an output vector

# Neuron = Logistic Regression

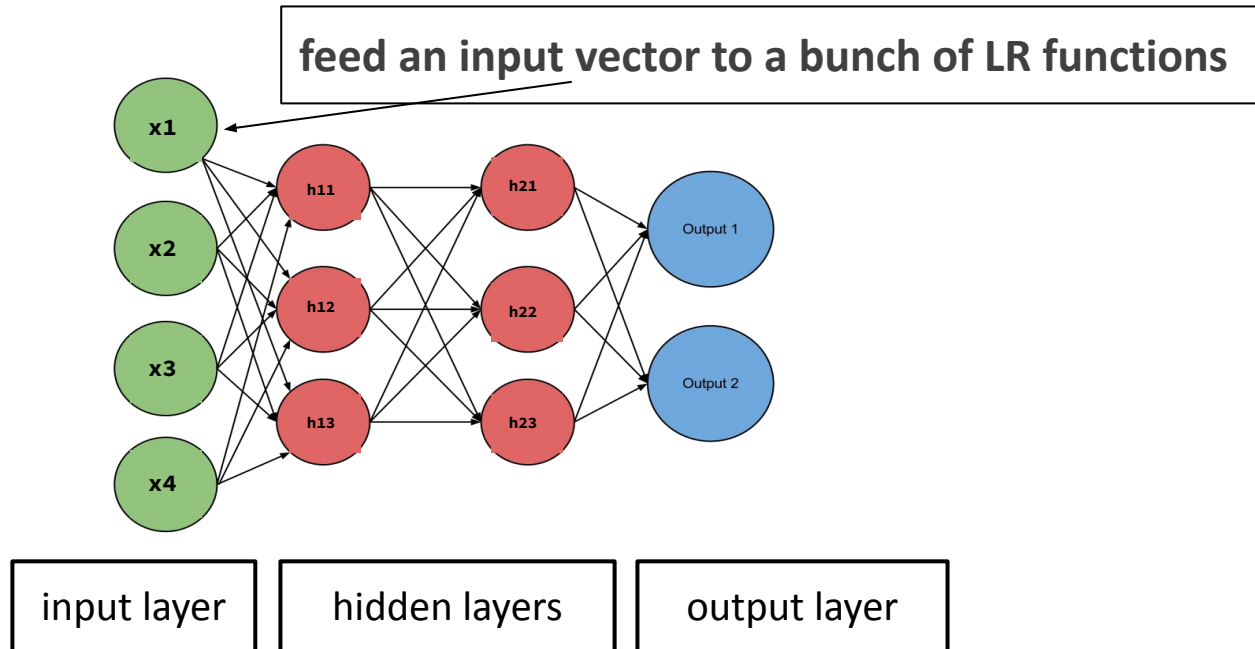
These are the exact computations of a single neuron



- We can feed an input vector to a bunch of LR functions and get an output vector
- which can be fed to another layer of LR functions

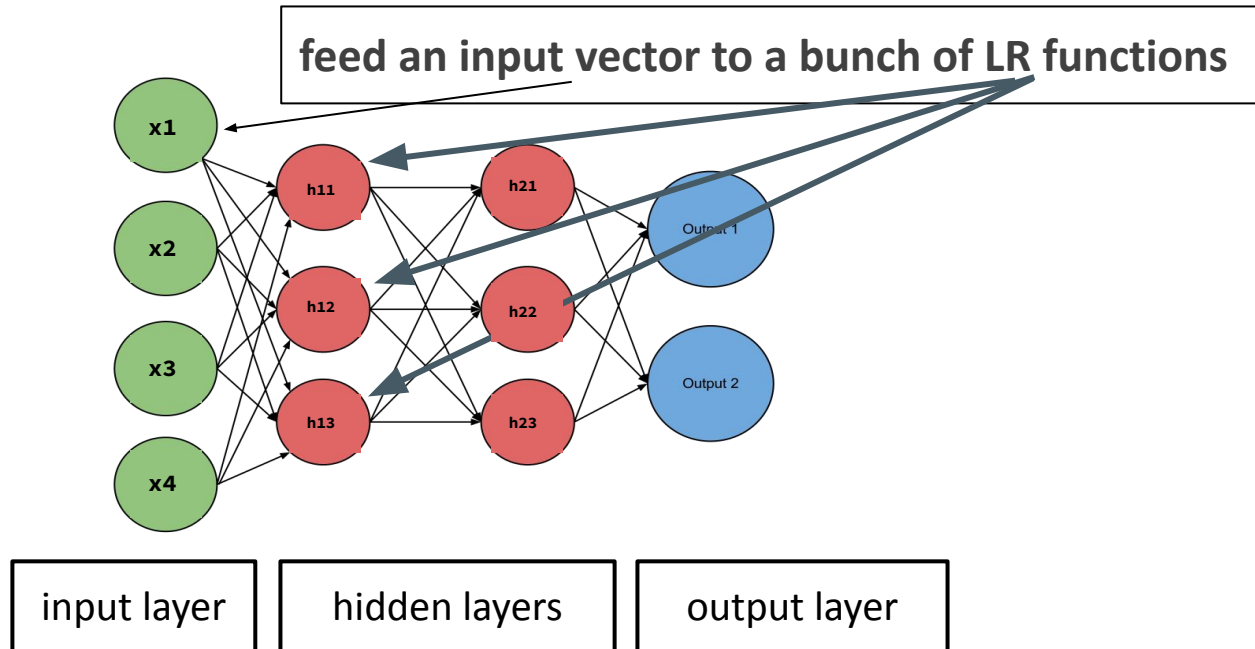
# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



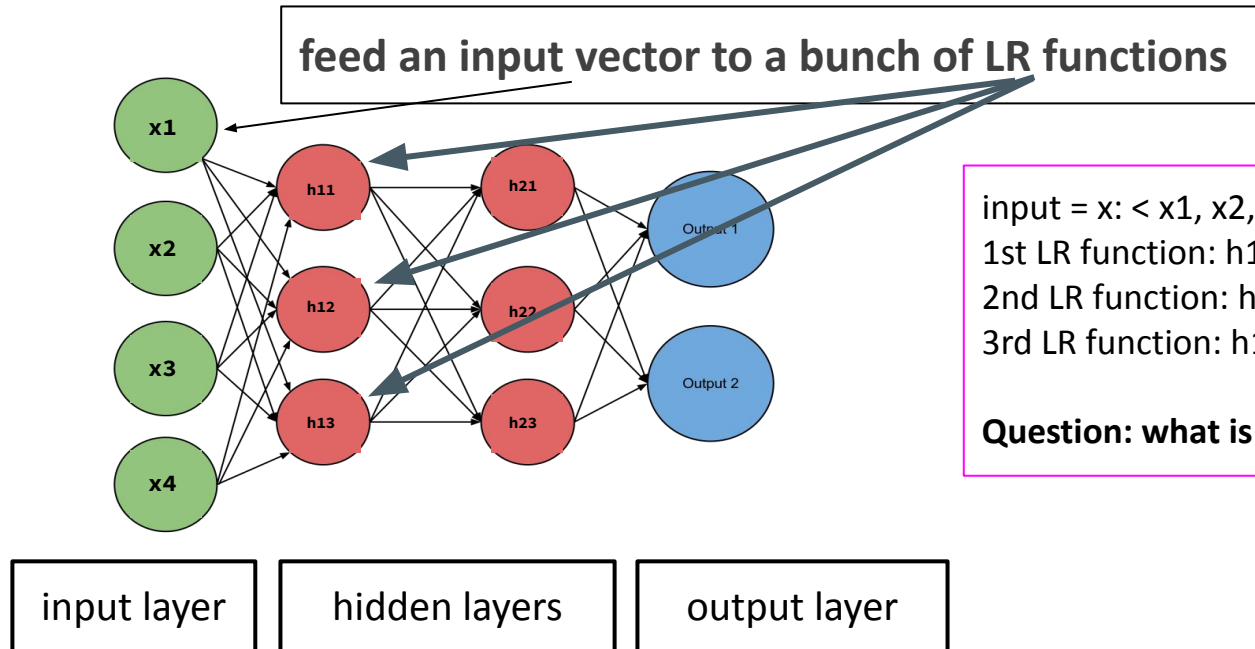
# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input =  $x$ :  $\langle x_1, x_2, x_3, x_4 \rangle$

1st LR function:  $h_{11} = f(w_1.x)$

2nd LR function:  $h_{12} = f(w_2.x)$

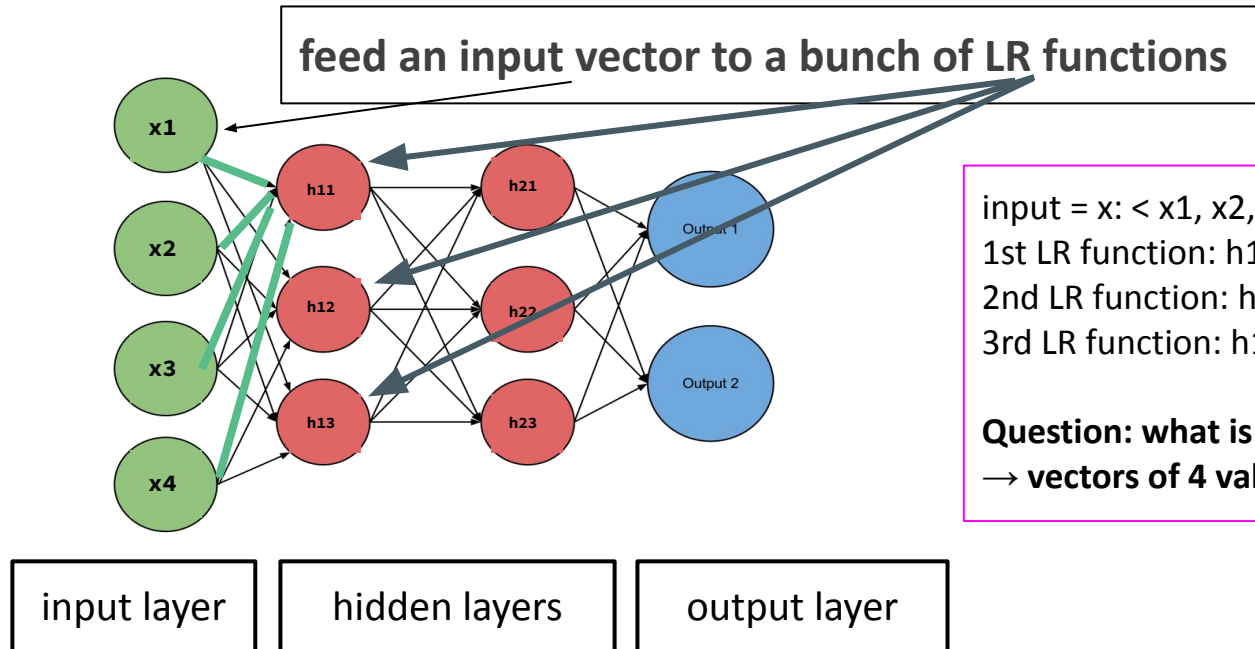
3rd LR function:  $h_{13} = f(w_3.x)$

**Question: what is the shape of the  $w_1/w_2/w_3$ ?**



# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input =  $x$ :  $\langle x_1, x_2, x_3, x_4 \rangle$

1st LR function:  $h_{11} = f(w_1.x)$

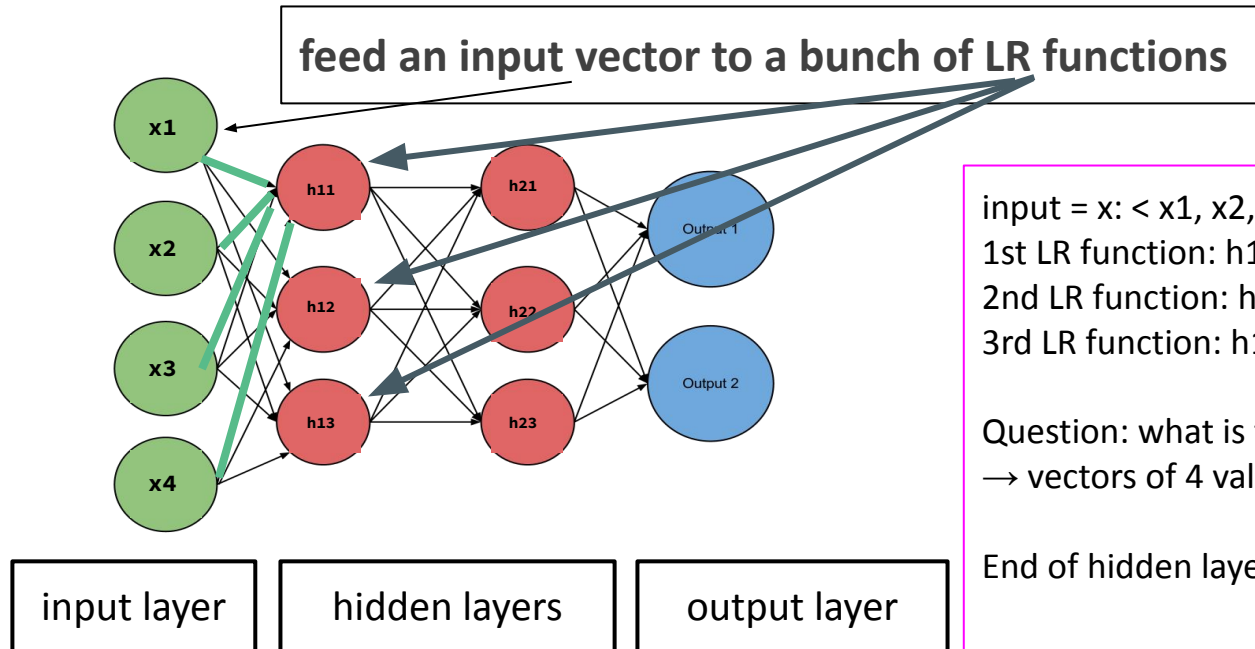
2nd LR function:  $h_{12} = f(w_2.x)$

3rd LR function:  $h_{13} = f(w_3.x)$

**Question: what is the shape of the  $w_1/w_2/w_3$ ?**  
→ vectors of 4 values, one for each input feature

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input =  $x$ :  $\langle x_1, x_2, x_3, x_4 \rangle$

1st LR function:  $h_{11} = f(w_1.x)$

2nd LR function:  $h_{12} = f(w_2.x)$

3rd LR function:  $h_{13} = f(w_3.x)$

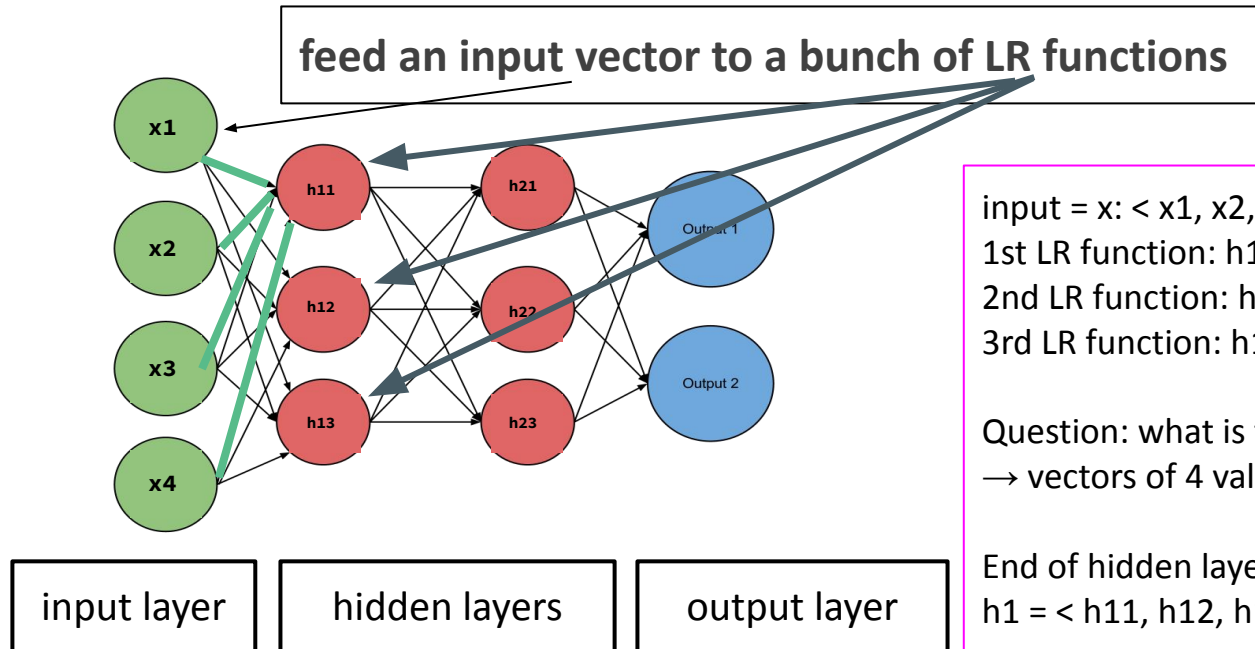
Question: what is the shape of the  $w_1/w_2/w_3$ ?

→ vectors of 4 values, one for each input feature

End of hidden layer 1 = a vector  **$h_1$**  of dimension?

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input =  $x$ :  $\langle x_1, x_2, x_3, x_4 \rangle$

1st LR function:  $h_{11} = f(w_1.x)$

2nd LR function:  $h_{12} = f(w_2.x)$

3rd LR function:  $h_{13} = f(w_3.x)$

Question: what is the shape of the  $w_1/w_2/w_3$ ?

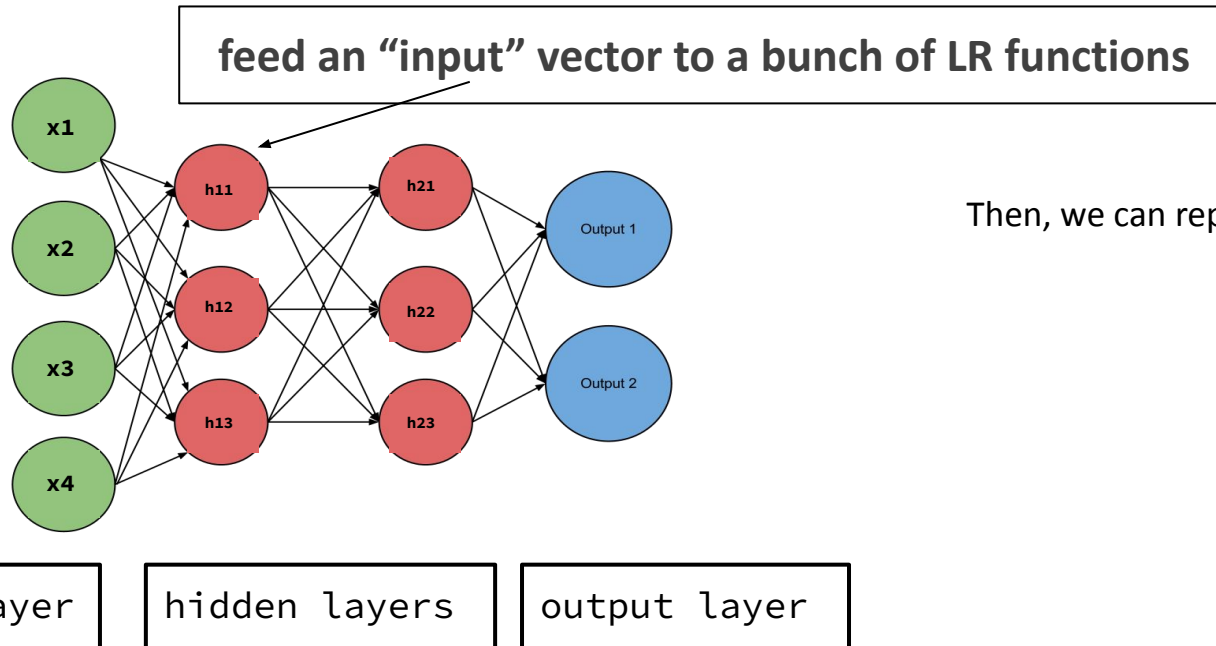
→ vectors of 4 values, one for each input feature

End of hidden layer 1 = a vector  **$h_1$  of dimension 3**

$h_1 = \langle h_{11}, h_{12}, h_{13} \rangle$

# Feed-forward architecture

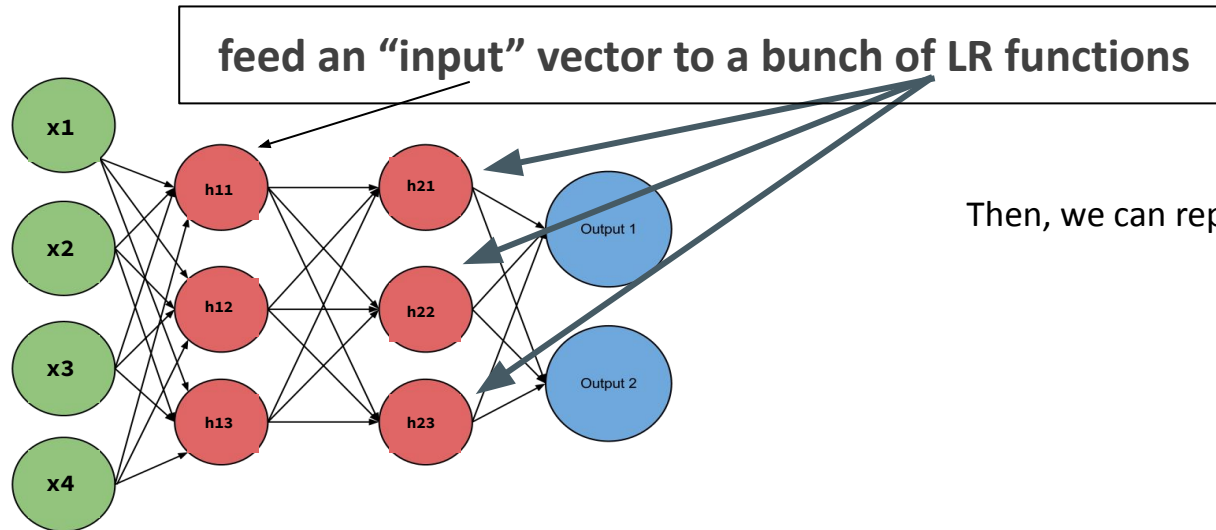
Multi-layer perceptron with 2 hidden layers



Then, we can repeat the process!

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



Then, we can repeat the process!

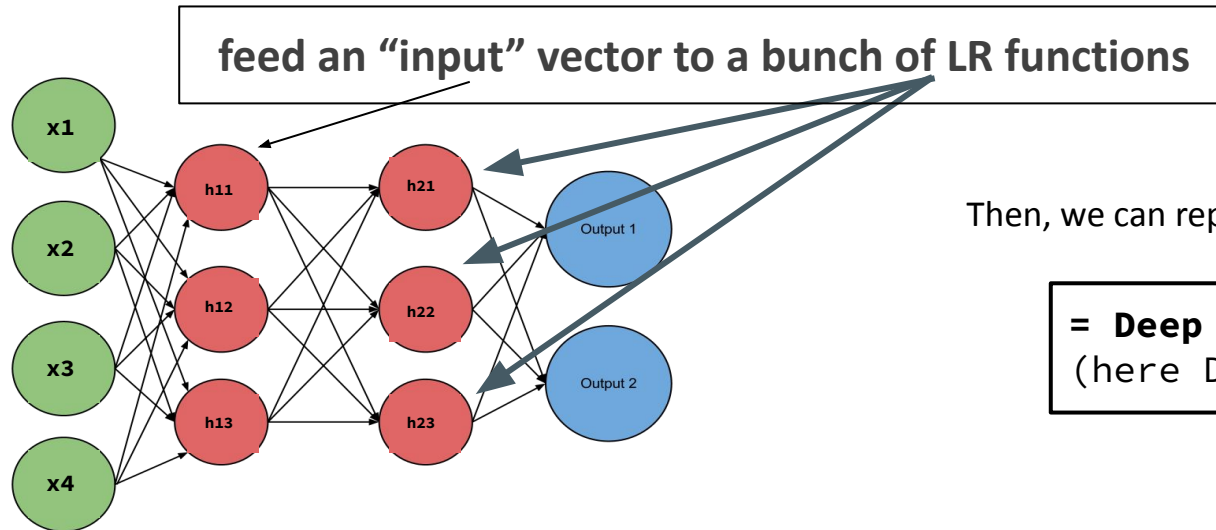
input layer

hidden layers

output layer

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



Then, we can repeat the process!

= **Deep** learning  
(here Depth = 2)

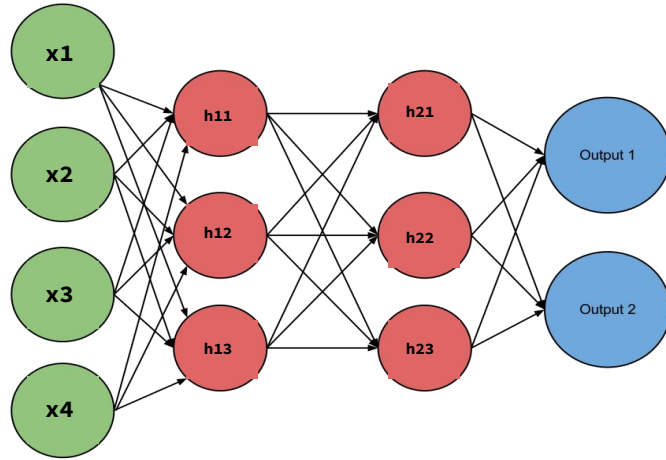
input layer

hidden layers

output layer

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



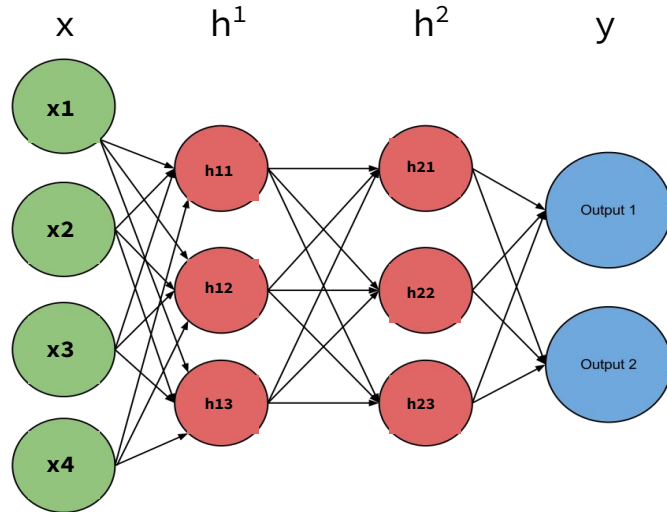
input layer

hidden layers

output layer

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

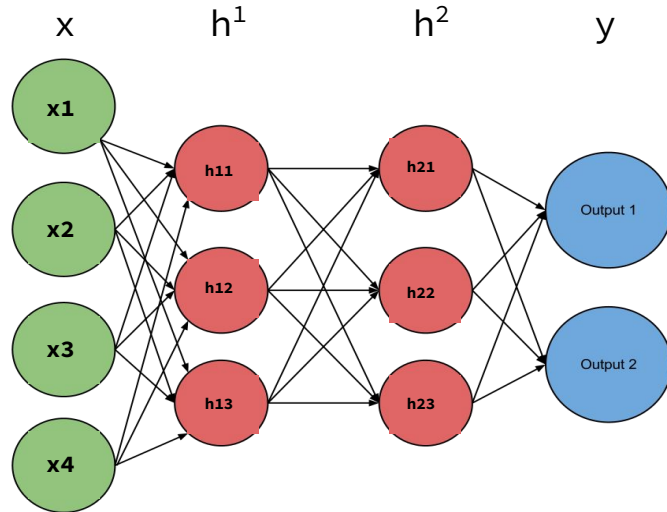
hidden layers

output layer



# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

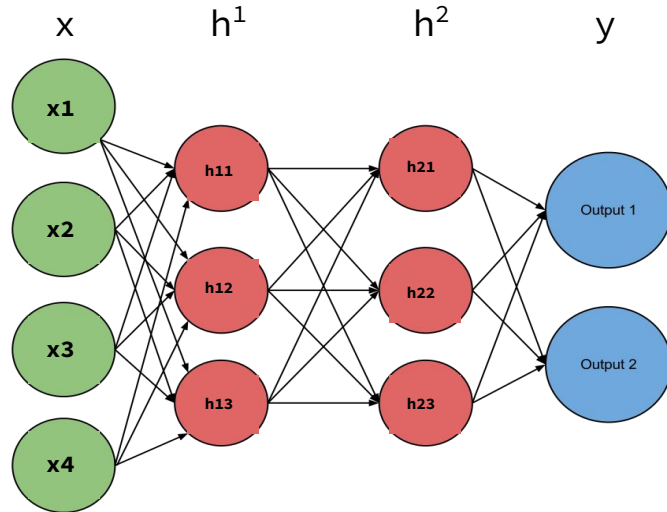
input layer

hidden layers

output layer

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

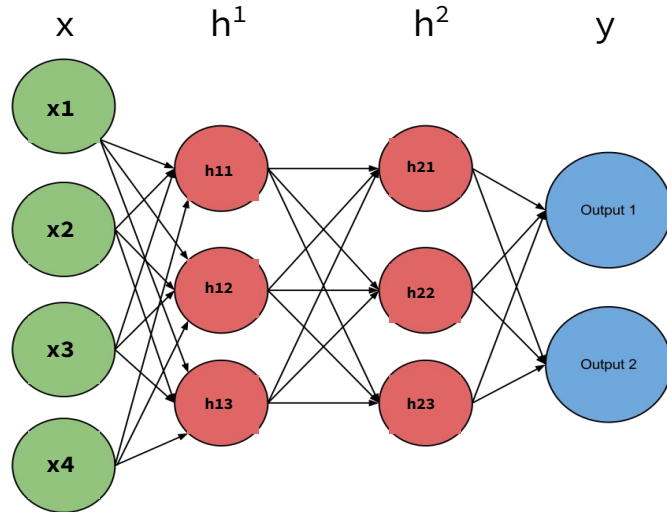
$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

fonctions  
linéaires

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

fonctions  
linéaires

fonctions non  
linéaires

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

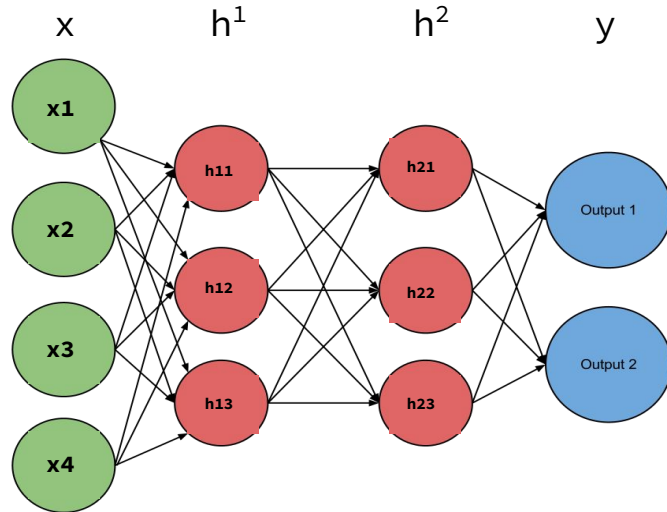
$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

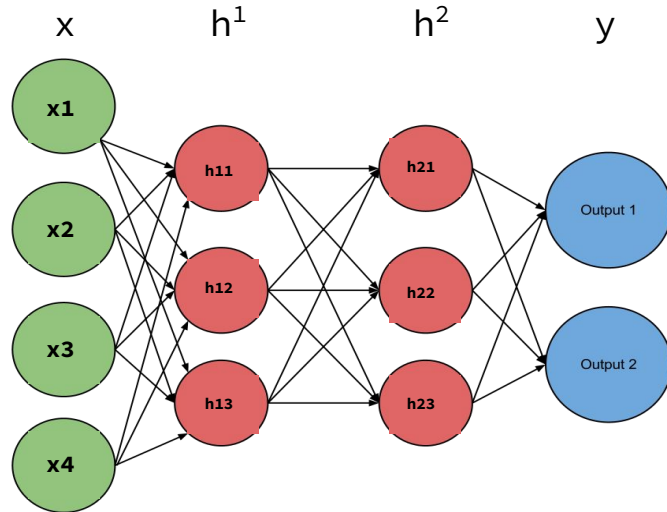
$\mathbf{x}$ : vector of size  $d_{in} =$

$\mathbf{y}$ : vector of size  $d_{out} =$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} =$

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} =$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} =$

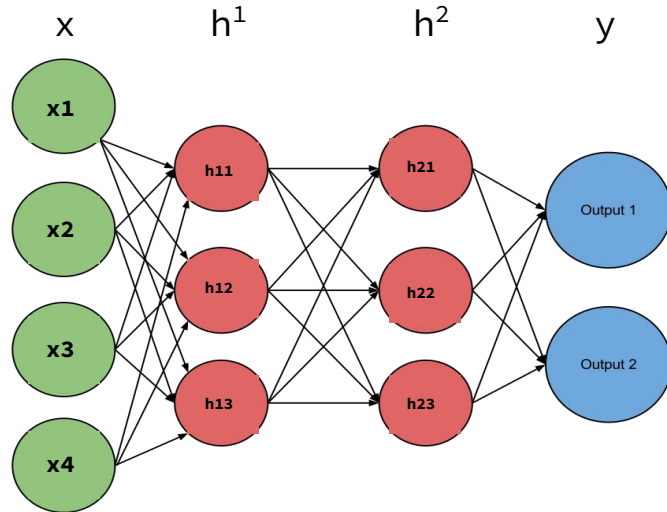
input layer

hidden layers

output layer

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

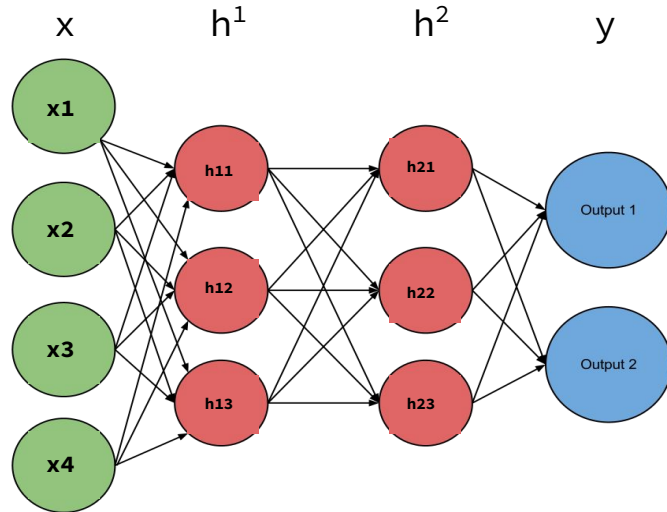
$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} =$

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

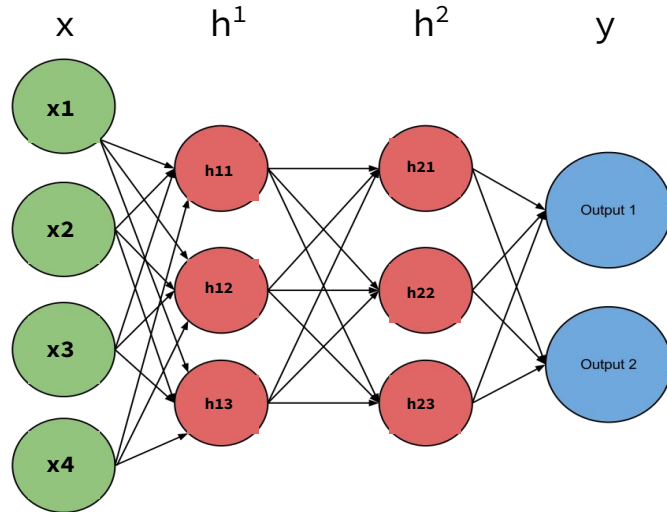
$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} = 3$

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} = 3$

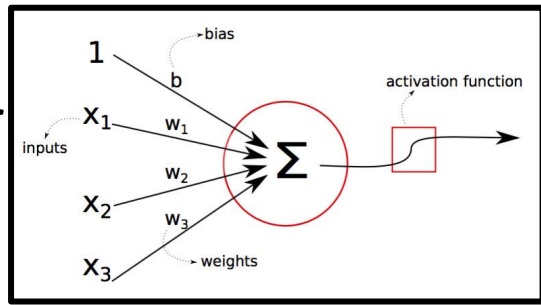
$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$ : matrices of size  $[4 \times 3], [3 \times 3], [3 \times 2]$

$\mathbf{b}^1, \mathbf{b}^2$ : 'bias' vectors of size  $d_{hidden} = 3$

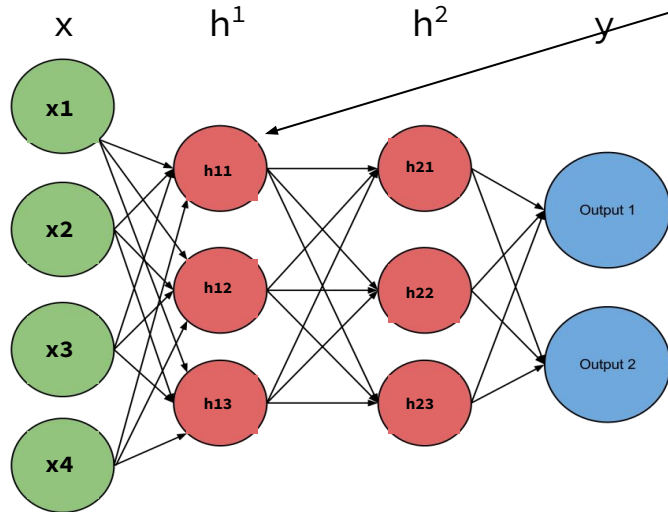
$g(\cdot)$ : non-linear activation function (elementwise)



# Feed-forward



Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$$\mathbf{W}^1 = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix}$$

$$= w_1 = w_2 = w_3$$

$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} = 3$

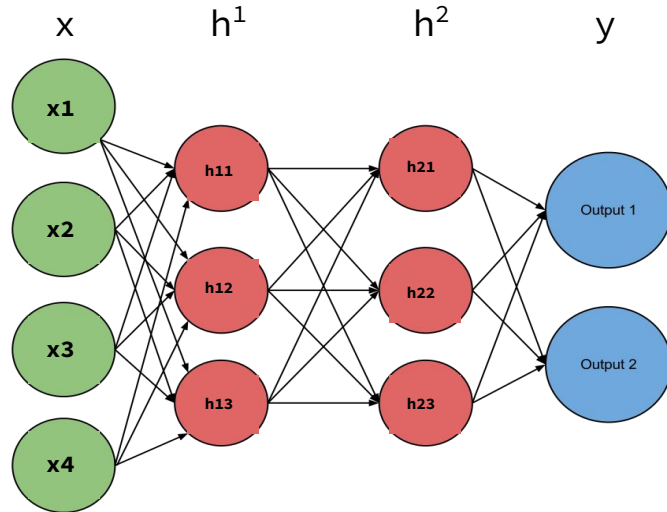
$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$ : matrices of size  $[4 \times 3], [3 \times 3], [3 \times 2]$

$\mathbf{b}^1, \mathbf{b}^2$ : 'bias' vectors of size  $d_{hidden} = 3$

$g(\cdot)$ : non-linear activation function (elementwise)

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} = 3$

Parameters of the network ( $\theta$ )

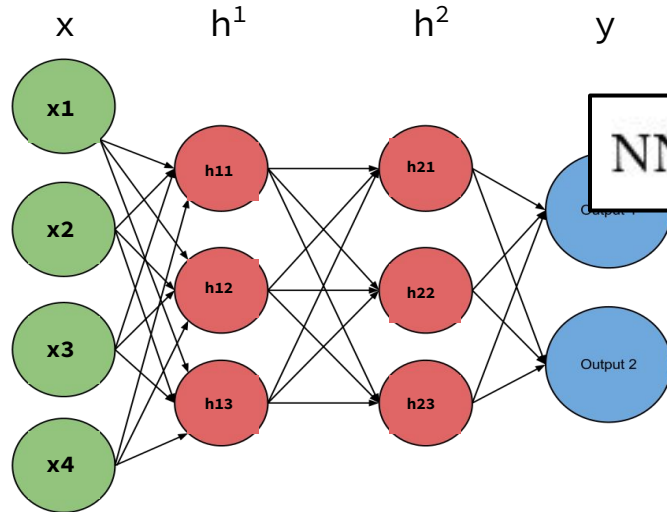
$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$ : matrices of size  $[4 \times 3], [3 \times 3], [3 \times 2]$

$\mathbf{b}^1, \mathbf{b}^2$ : 'bias' vectors of size  $d_{hidden} = 3$

$g(\cdot)$ : non-linear activation function (elementwise)

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$$NN_{MLP2}(\mathbf{x}) = (g^2(g^1(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)\mathbf{W}^2 + \mathbf{b}^2))\mathbf{W}^3$$

# Linear algebra (what? why?)

***Linear algebra** is the branch of mathematics concerning **linear** equations and **linear** functions and their representations through matrices and vector spaces. (Wikipedia)*

→ “Under the hood, the feed forward neural network is just **a composite function, that multiplies some matrices and vectors together**. It is not that vectors and matrices are the only way to do these operations but they become highly **efficient** if you do so. (..) neural networks are computationally expensive, so they require this nice trick to make them compute faster. **It’s called vectorization. They make computations extremely faster. This is one of the main reasons why GPUs are required for deep learning, as they are specialized in vectorized operations like matrix multiplication.**”

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- Matrix multiplication

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3}$$

Original matrix  
of order 2 x 3



$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{3 \times 2}$$

Transpose matrix  
of order 3 x 2

## Operations:

- **Transpose of a matrix:** mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- Matrix multiplication

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

$$\begin{bmatrix} a & b \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = [ax + by]$$

## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- **Dot product**: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum(x_i)^2}$
- Matrix multiplication

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

$$\begin{bmatrix} 3 & 1 & 4 \\ 2 & 5 & 0 \end{bmatrix}_{2 \times 3} \quad \begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 0 & 4 \\ 5 & 6 \end{bmatrix}_{4 \times 2} \quad \begin{bmatrix} 1.6 & 0.2 & 1.0 \end{bmatrix}_{1 \times 3}$$

$$\begin{bmatrix} 2 & 0 \\ 0 & 3.5 \end{bmatrix}_{2 \times 2} \quad \begin{bmatrix} 5 \\ 3 \end{bmatrix}_{2 \times 1} \quad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 40 & 20 & 1 \end{bmatrix}_{3 \times 3}$$

## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- **Dimension / shape of a matrix:** row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- Matrix multiplication



# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- **Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum(x_i)^2}$**
- Matrix multiplication

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

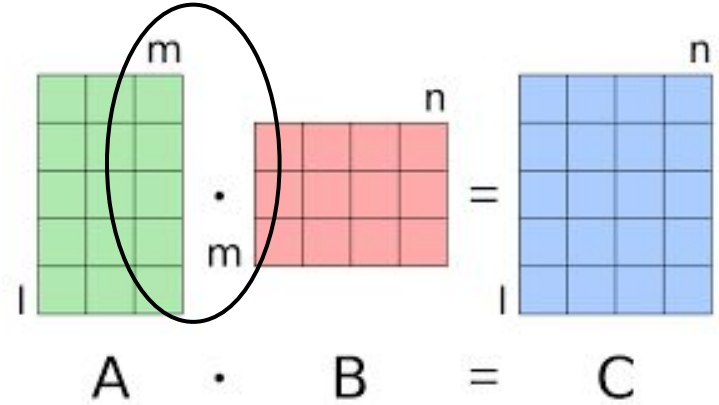
$$\begin{array}{ccc}
 \begin{array}{c} \vec{a_1} \rightarrow \\ \vec{a_2} \rightarrow \end{array} & \begin{array}{c} \vec{b_1} \quad \vec{b_2} \\ \downarrow \quad \downarrow \end{array} \\
 \begin{bmatrix} 1 & 7 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 & 3 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \vec{a_1} \cdot \vec{b_1} & \vec{a_1} \cdot \vec{b_2} \\ \vec{a_2} \cdot \vec{b_1} & \vec{a_2} \cdot \vec{b_2} \end{bmatrix} \\
 A & B & C
 \end{array}$$

## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum(x_i)^2}$
- **Matrix multiplication:**
  - The new matrix takes the **rows of the 1st matrix and columns of the 2nd matrix**
  - The number of columns of the 1st matrix must equal the number of rows of the 2nd
  - The product of an M x N matrix and an N x K matrix is an M x K matrix.

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.

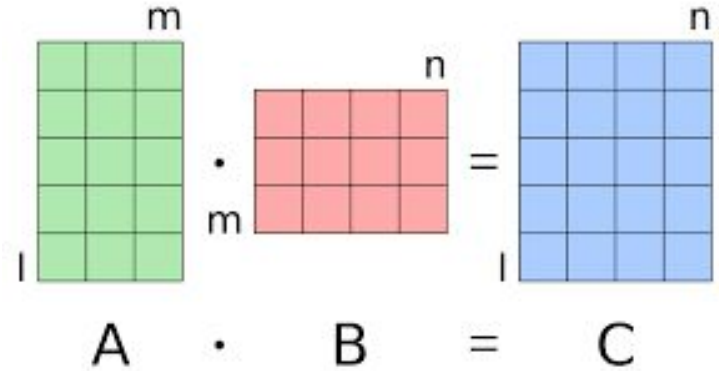


## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- **Matrix multiplication:**
  - The new matrix takes the rows of the 1st matrix and columns of the 2nd matrix
  - The **number of columns of the 1st matrix must equal the number of rows of the 2nd**
  - The product of an M x N matrix and an N x K matrix is an M x K matrix.

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank  $n$ .

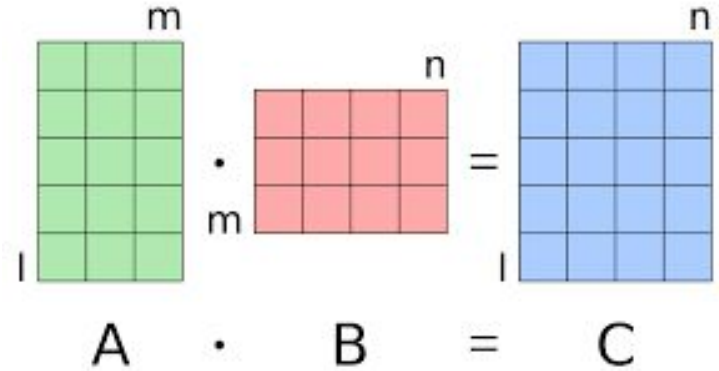


## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- **Matrix multiplication:**
  - The new matrix takes the rows of the 1st matrix and columns of the 2nd matrix
  - The number of columns of the 1st matrix must equal the number of rows of the 2nd
  - The product of an  $\mathbf{M} \times \mathbf{N}$  matrix and an  $\mathbf{N} \times \mathbf{K}$  matrix is an  $\mathbf{M} \times \mathbf{K}$  matrix.

# Linear algebra

- Scalar: A single number (rank 0 tensor)
- Vector : A list of values (rank 1 tensor)
- Matrix: A two dimensional list of values (rank 2 tensor)
- Tensor: A multi dimensional matrix with rank n.



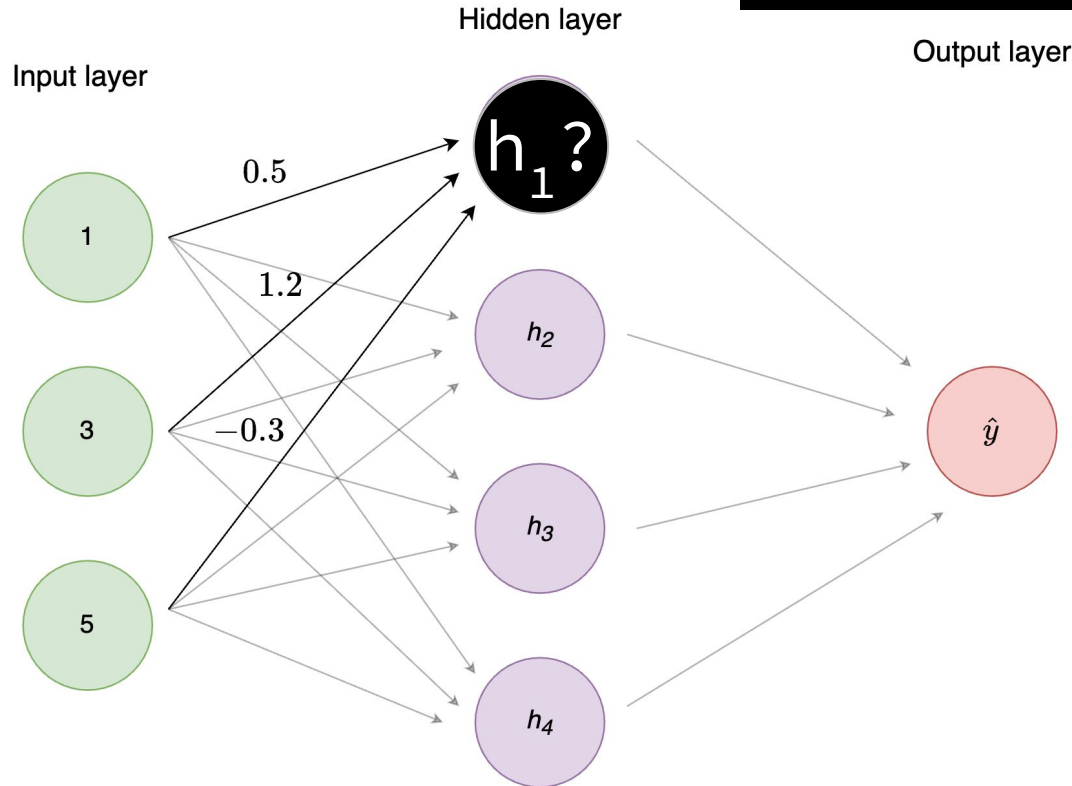
## Operations:

- Transpose of a matrix: mirror image of the matrix across the diagonal line (from top left to the bottom right of the matrix)
- Dot product: between 2 vectors, return a scalar
- Dimension / shape of a matrix: row by column
- Norm is the size of the vector, e.g.  $L2 = \sqrt{\sum (x_i)^2}$
- **Matrix multiplication:**
  - The new matrix takes the rows of the 1st matrix and columns of the 2nd matrix
  - The number of columns of the 1st matrix must equal the number of rows of the 2nd
  - The product of an  $\mathbf{M} \times \mathbf{N}$  matrix and an  $\mathbf{N} \times \mathbf{K}$  matrix is an  $\mathbf{M} \times \mathbf{K}$  matrix.

$$\mathbf{M} \times \mathbf{N} \cdot \mathbf{N} \times \mathbf{K} = \mathbf{M} \times \mathbf{K}$$

# Let's try

```
>>> import numpy as np
>>> def sig(x):
...     return 1/(1+np.exp(-x))
```



Compute the value for the first neuron in the hidden layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

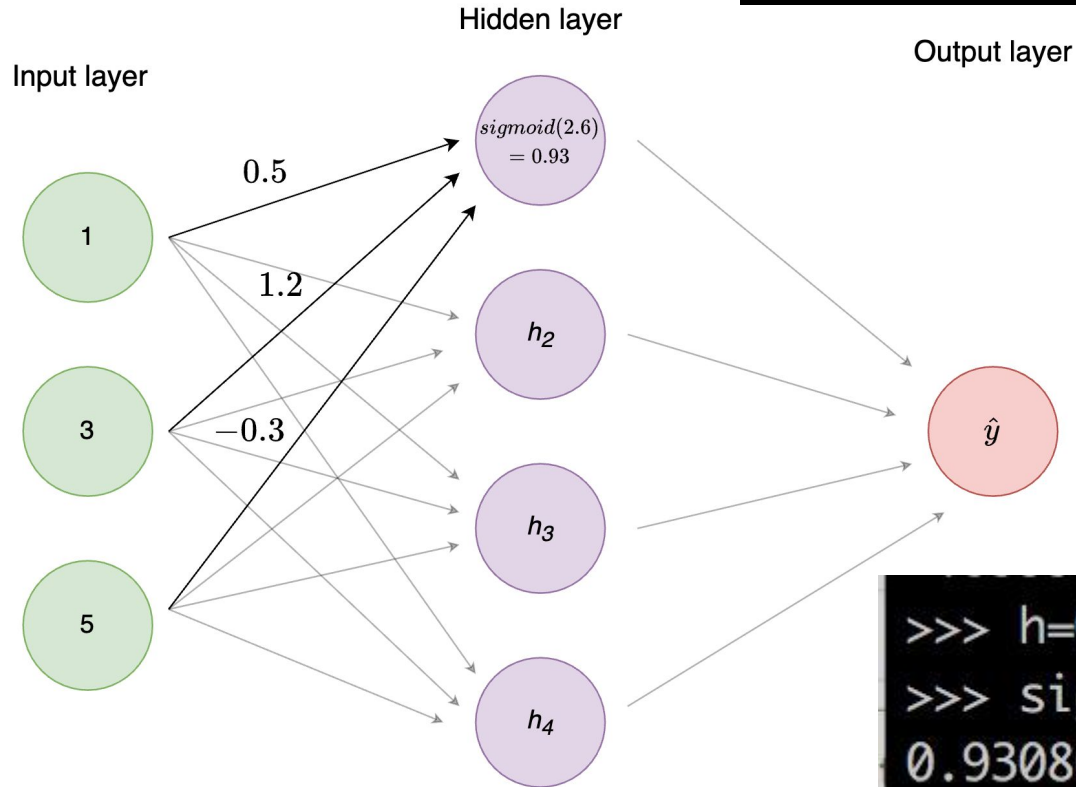
$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

# Let's try

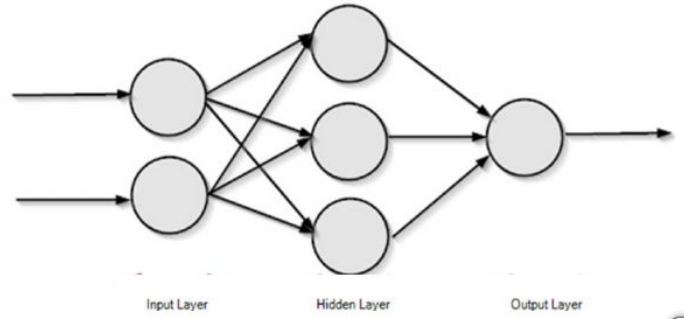
```
>>> import numpy as np
>>> def sig(x):
...     return 1/(1+np.exp(-x))
```



Compute the value for the first neuron in the hidden layer

```
>>> h=0.5*1+1.2*3+5*-0.3
>>> sig(h)
0.9308615796566531
```

# Let's try

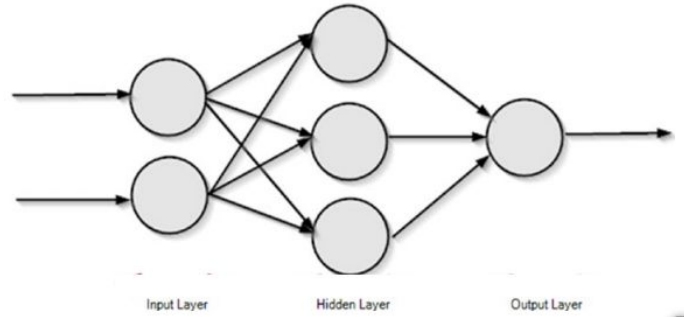


input layer:  $x = \langle x_1 \ x_2 \rangle$  shape: 1x2

weights for the hidden layer:  
w shape?



# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

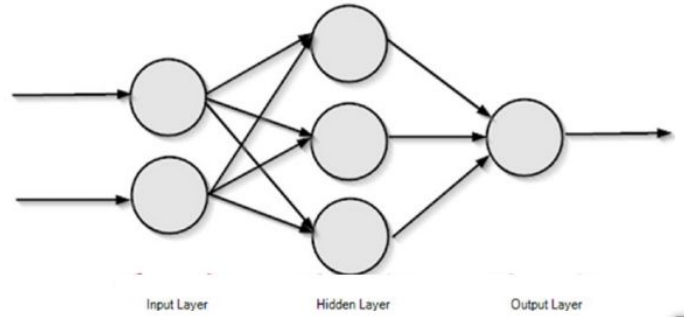
weights for the hidden layer:

$w$  shape?  **$2 \times 3$**  (input  $\times$  hidden)

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

output:  $x.W \rightarrow$  shape?

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

$w$  shape?  **$2 \times 3$**  (input  $\times$  hidden)

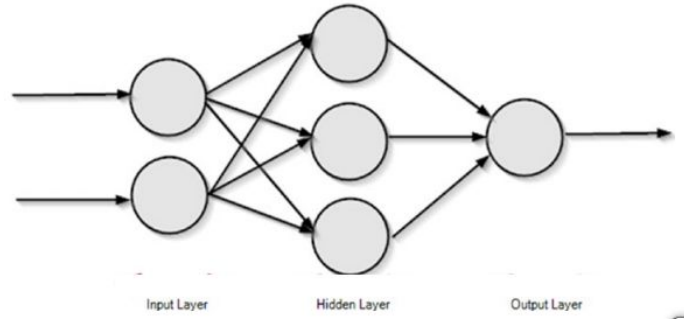
$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow 0k$

$h = \langle$

$\rangle$

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

$w$  shape?  **$2 \times 3$**  (input  $\times$  hidden)

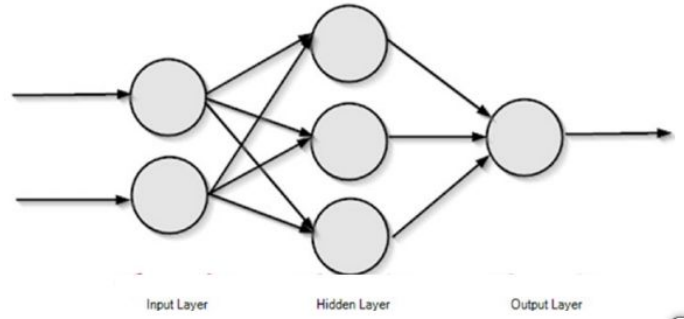
$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow 0k$

$h = \langle x_1 * w_{11} + x_2 * w_{21}$

$\rangle$

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

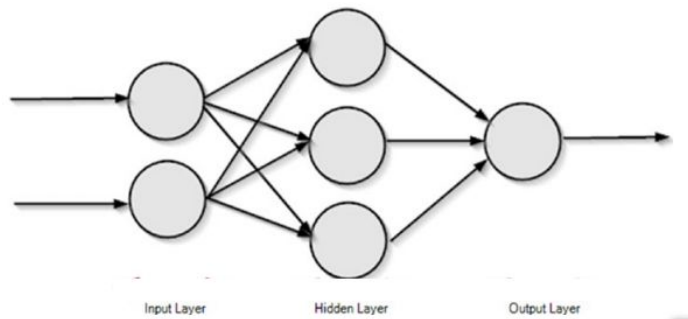
$w$  shape?  **$2 \times 3$**  (input  $\times$  hidden)

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow \text{Ok}$

$h = \langle x_1 \cdot w_{11} + x_2 \cdot w_{21} \quad x_1 \cdot w_{12} + x_2 \cdot w_{22} \quad x_1 \cdot w_{13} + x_2 \cdot w_{23} \rangle$

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

$w$  shape?  **$2 \times 3$**  (input  $\times$  hidden)

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

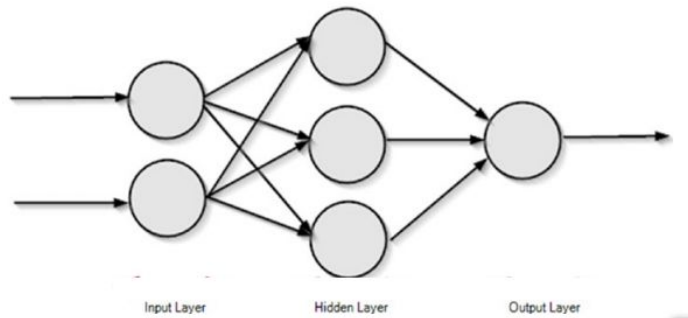
output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow 0k$

$h = \langle x_1 \cdot w_{11} + x_2 \cdot w_{21} \quad x_1 \cdot w_{12} + x_2 \cdot w_{22} \quad x_1 \cdot w_{13} + x_2 \cdot w_{23} \rangle$

Try with:  $x = \langle 0 \ 1 \rangle$  and  $W = \begin{bmatrix} 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$

$x.W = \langle ? \ ? \ ? \rangle$

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

w shape?  **$2 \times 3$**  (input  $\times$  hidden)

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

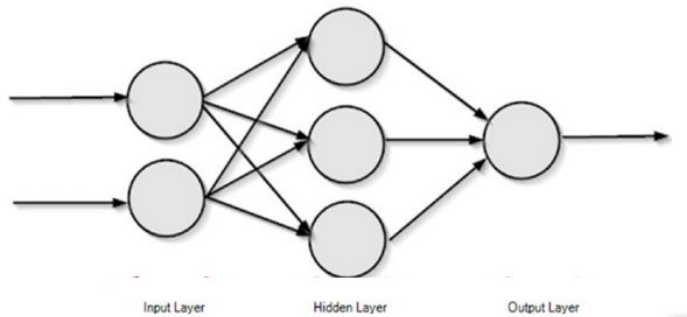
output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow 0k$

$$h = \langle x_1 \cdot w_{11} + x_2 \cdot w_{21} \quad x_1 \cdot w_{12} + x_2 \cdot w_{22} \quad x_1 \cdot w_{13} + x_2 \cdot w_{23} \rangle$$

Try with:  $x = \langle 0 \ 1 \rangle$  and  $W = \begin{bmatrix} 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$

$$x.W = \langle 100 \ 200 \ 300 \rangle$$

# Let's try



input layer:  $x = \langle x_1 \ x_2 \rangle$  shape:  $1 \times 2$

weights for the hidden layer:

$w$  shape?  **$2 \times 3$**  (input  $x$  hidden)

$$w = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

output:  $x.W \rightarrow$  shape?  $1 \times 2 \cdot 2 \times 3 = 1 \times 3 \rightarrow \text{Ok}$

$$h = \langle x_1 \cdot w_{11} + x_2 \cdot w_{21} \quad x_1 \cdot w_{12} + x_2 \cdot w_{22} \quad x_1 \cdot w_{13} + x_2 \cdot w_{23} \rangle$$

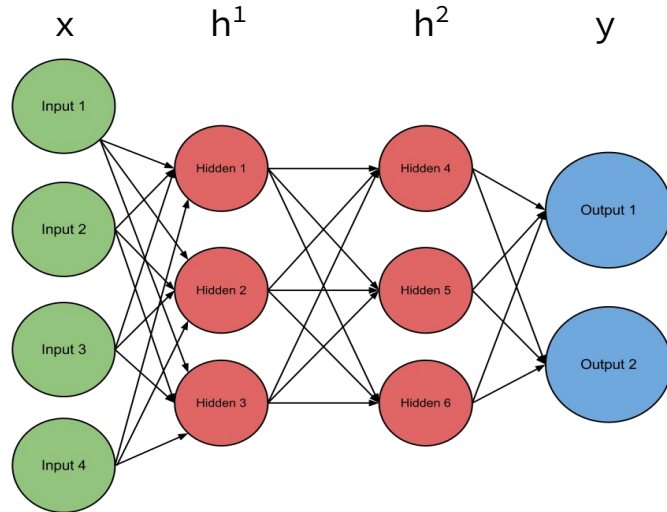
Try with:  $x = \langle 0 \ 1 \rangle$  and  $W = \begin{bmatrix} 10 & 20 & 30 \\ 100 & 200 & 300 \end{bmatrix}$

$$x.W = \langle 100 \ 200 \ 300 \rangle$$

```
>>> x=np.array([0,1])
>>> W = np.array( [[10,20,30],[100, 200, 300]] )
>>> x
array([0, 1])
>>> W
array([[ 10,  20,  30],
       [100, 200, 300]])
>>> np.dot(x, W)
array([100, 200, 300])
```

# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input layer

hidden layers

output layer

$$NN_{MLP2}(\mathbf{x}) = \mathbf{y} \quad (1)$$

$$\mathbf{h}^1 = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) \quad (2)$$

$$\mathbf{h}^2 = g(\mathbf{h}^1\mathbf{W}^2 + \mathbf{b}^2) \quad (3)$$

$$\mathbf{y} = \mathbf{h}^2\mathbf{W}^3 \quad (4)$$

$\mathbf{x}$ : vector of size  $d_{in} = 4$

$\mathbf{y}$ : vector of size  $d_{out} = 2$

$\mathbf{h}^1, \mathbf{h}^2$ : vectors of size  $d_{hidden} = 3$

$$h1 \rightarrow \mathbf{x} \cdot \mathbf{W}^1 = (1 \times d_{in}) \cdot (d_{in} \times d_{h1}) \rightarrow (1 \times 3)$$

$$h2 \rightarrow \mathbf{h}^1 \cdot \mathbf{W}^2 = (1 \times d_{h1}) \cdot (d_{h1} \times d_{h2}) \rightarrow (1 \times 3)$$

$$\mathbf{y} \rightarrow \mathbf{h}^2 \cdot \mathbf{W}^3 = (1 \times d_{h2}) \cdot (d_{h2} \times d_{out}) \rightarrow (1 \times 2)$$

$\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$ : matrices of size  $[4 \times 3], [3 \times 3], [3 \times 2]$

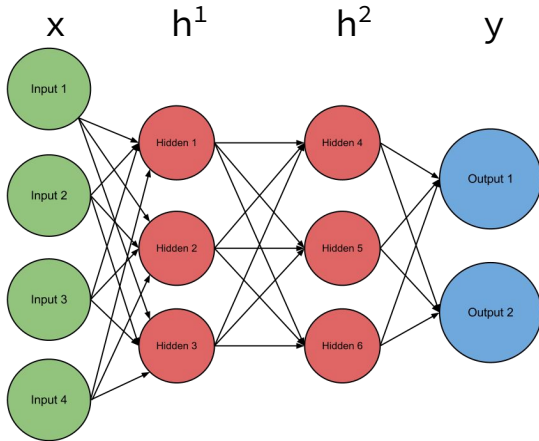
$\mathbf{b}^1, \mathbf{b}^2$ : 'bias' vectors of size  $d_{hidden} = 4$

$g(\cdot)$ : non-linear activation function (elementwise)



# Feed-forward architecture

Multi-layer perceptron with 2 hidden layers



input  
layer

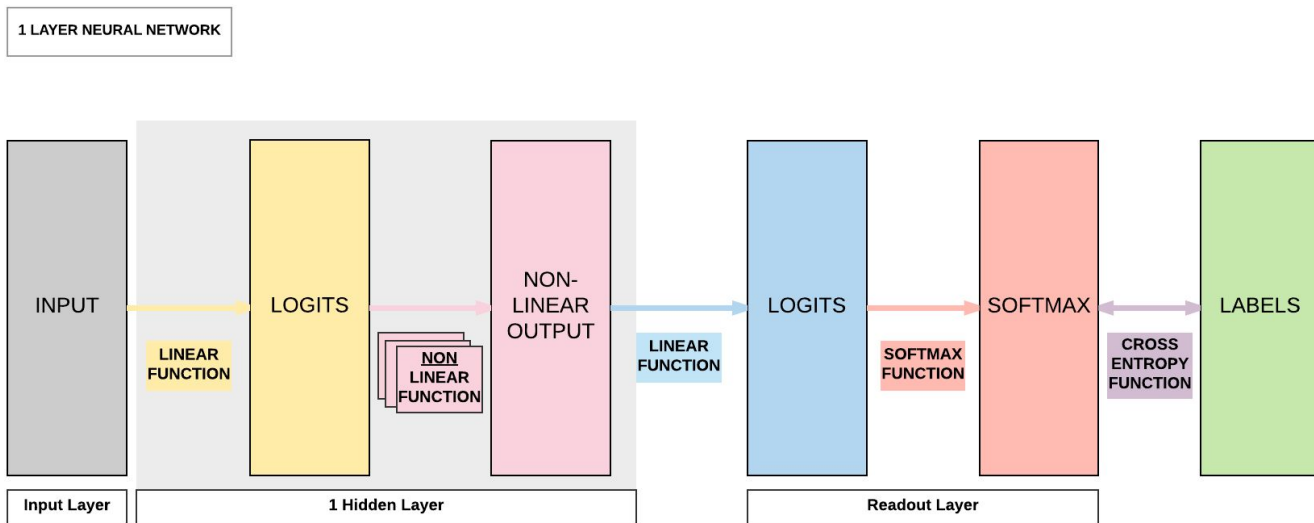
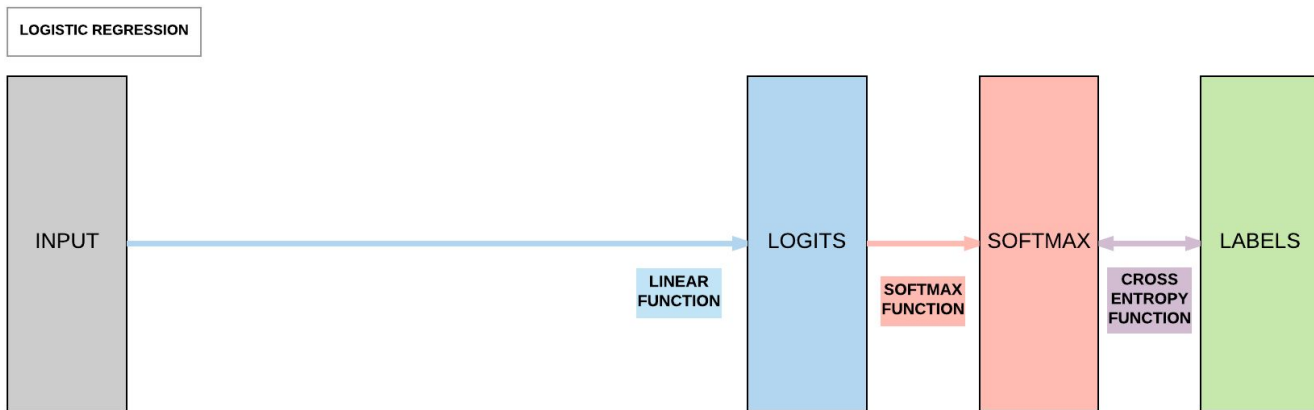
hidden  
layers

output  
layer

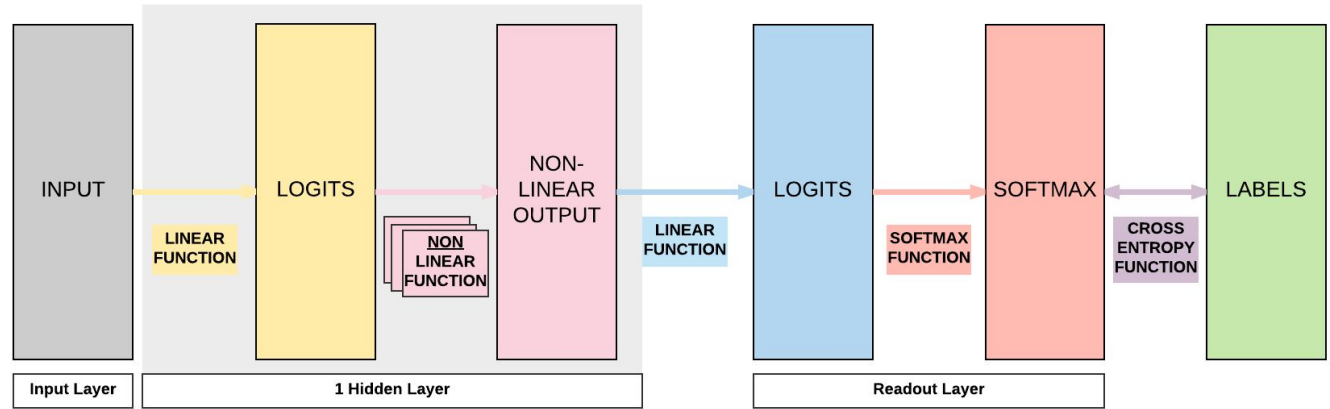
- **layer** = vector resulting from each linear transformation
- the outer-most linear transform results in the **output layer**
  - if  $dout = 1$  : regression or binary classif
  - if  $dout > 1$ : classif MC
- the other linear transforms result in **hidden layers**
- each hidden layer is followed by a **non-linear activation**
- the bias vector can be forced to 0 (= “**dropped**”) as here in the last layer
- layers resulting from linear transformations are often referred to as **fully connected** or **affine** (other types: *pooling* or *convolutional* layers)

# Summary

*a feed-forward network is simply a stack of linear models separated by nonlinear functions*



# Summary



Representation power:

- MLP1 = universal approximator: it can approximate a large family of functions [Hornik et al. 1989; Cybenko, 1989]. Why going beyond MLP1?

- theoretical results do not discuss the learnability of the NN: a representation exists, but we don't know how easy or hard it is to set the parameters based on training data and learning algorithm
- + does not guarantee that a training algorithm will find the correct function
- + it does not state how large the hidden layer should be

→ in real worlds conditions: there is benefit at trying more complex architectures

# Practical session

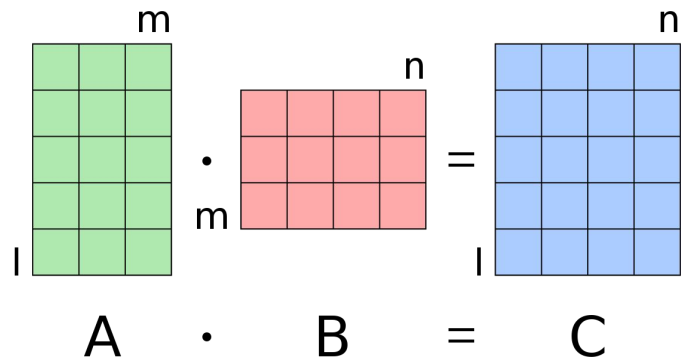
Walk through code in PyTorch:

- sentiment classification
- feed-forward Neural Network, with BoW representation of documents

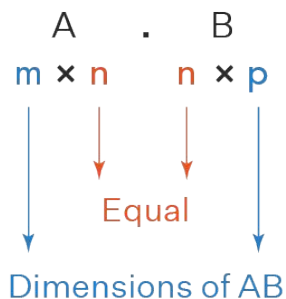
# Sources

- Parts of the course borrowed, with a few modifications, to P. Muller
- Softmax: [https://www.wikiwand.com/fr/Fonction\\_softmax](https://www.wikiwand.com/fr/Fonction_softmax)
- <https://www.deeplearningbook.org/>
- <https://towardsdatascience.com/linear-algebra-explained-in-the-context-of-deep-learning-8fcb8fca1494>
- [https://ml-cheatsheet.readthedocs.io/en/latest/linear\\_algebra.html](https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html)
- <https://blog.paperspace.com/data-loaders-abstractions-pytorch/>
- [https://www.deeplearningwizard.com/deep\\_learning/practical\\_pytorch/pytorch\\_feedforward\\_neuralnetwork/](https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neuralnetwork/)
- <https://www.i2tutorials.com/explain-softmax-activation-function-and-difference-between-sigmoid-and-softmax-function/>
- [http://perso.ens-lyon.fr/jacques.jayez/Cours/LHPST/Deep\\_Learning\\_in\\_NLP\\_1.pdf](http://perso.ens-lyon.fr/jacques.jayez/Cours/LHPST/Deep_Learning_in_NLP_1.pdf)
- <https://krisbolton.com/a-quick-introduction-to-artificial-neural-networks-part-1>
-

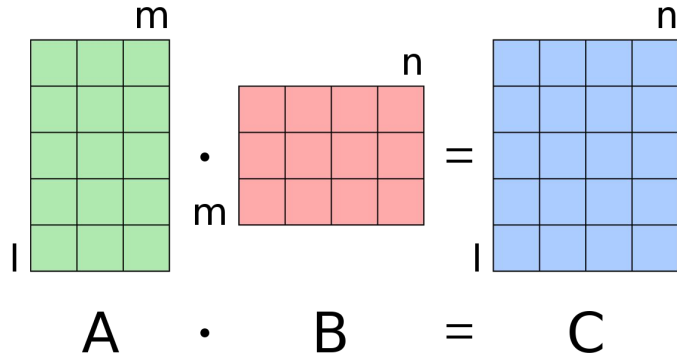
# Matrix multiplication



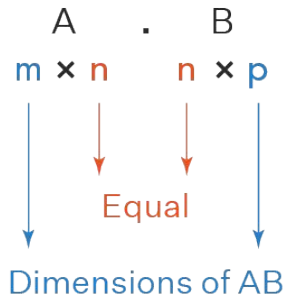
Multiplication of Matrices



# Matrix multiplication



Multiplication of Matrices



$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

$2 \times 4$        $4 \times 3$        $2 \times 3$

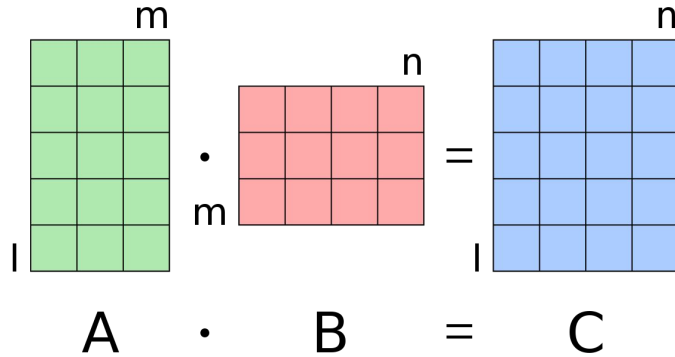
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

Dans la matrice résultante :

- ligne 1, col 1 (c11) = ligne 1 A X col 1 B

# Matrix multiplication



Multiplication of Matrices

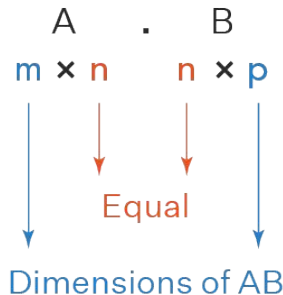


Diagram showing the calculation of the element  $c_{11}$  in the resulting matrix C. It is the dot product of the first row of A and the first column of B.

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

Dimensions:  $2 \times 4$        $4 \times 3$        $2 \times 3$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

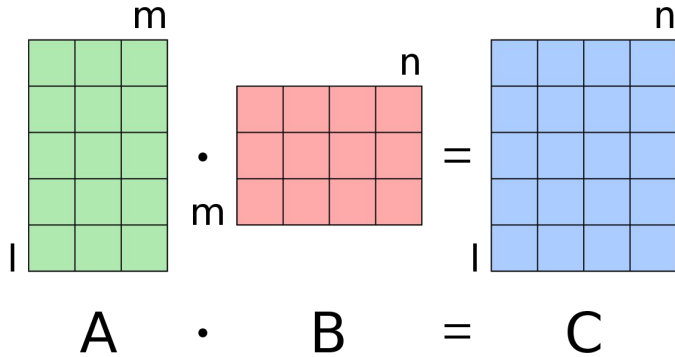
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$



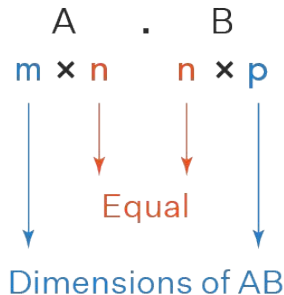
# Matrix multiplication

Dans la matrice résultante :

- ligne 1, col 1 (c11) = ligne 1 A X col 1 B
- ligne 1, col 2 (c12) = ligne 1 A X col 2 B



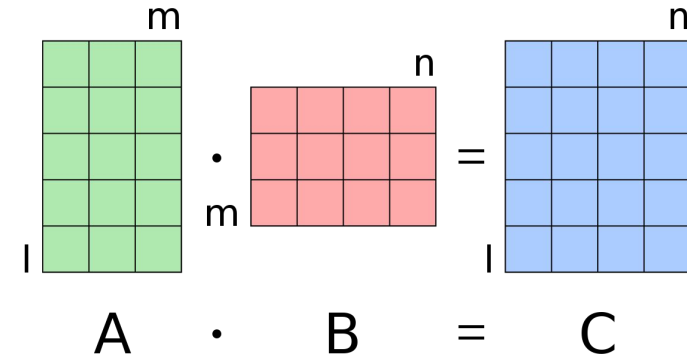
Multiplication of Matrices



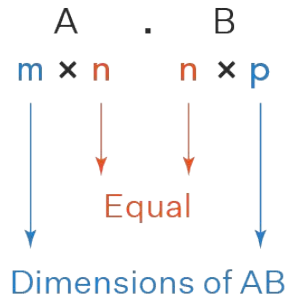
$$\begin{array}{c}
 c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} \\
 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \\
 2 \times 4 \qquad 4 \times 3 \qquad 2 \times 3
 \end{array}$$
  

$$\begin{array}{c}
 c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \\
 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}
 \end{array}$$

# Matrix multiplication



Multiplication of Matrices



Dans la matrice résultante :

- ligne 1, col 1 ( $c_{11}$ ) = ligne 1 A X col 1 B
- ligne 1, col 2 ( $c_{12}$ ) = ligne 1 A X col 2 B
- ligne **1**, col **3** ( $c_{13}$ ) = ligne **1** A X col **3** B

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

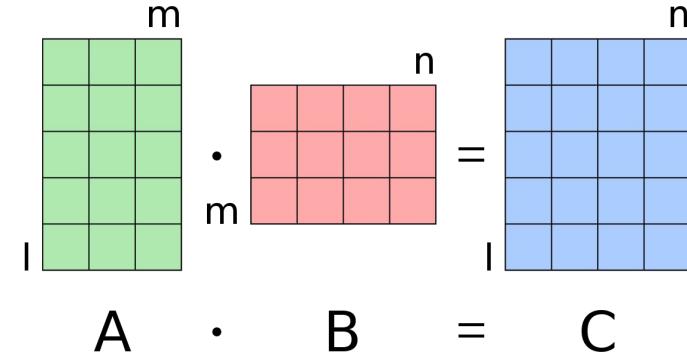
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

Dimensions:  $2 \times 4$ ,  $4 \times 3$ ,  $2 \times 3$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

# Matrix multiplication



$$A \cdot B = C$$

Multiplication of Matrices



$$\begin{array}{ccc}
 A & \cdot & B \\
 m \times n & & n \times p \\
 \downarrow & \text{Equal} & \downarrow \\
 \text{Dimensions of } AB & & 
 \end{array}$$

Dans la matrice résultante :

- ligne 1, col 1 (c11) = ligne 1 A X col 1 B
- ligne 1, col 2 (c12) = ligne 1 A X col 2 B
- ligne 1, col 3 (c13) = ligne 1 A X col 3 B
- ligne 2, col 1 (c21) = ligne 2 A X col 1 B

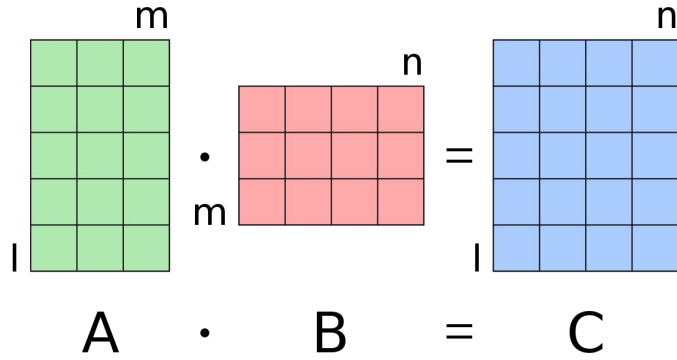
$$\begin{array}{ccc}
 c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} \\
 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \\
 2 \times 4 \qquad \qquad 4 \times 3 \qquad \qquad 2 \times 3
 \end{array}$$

$$\begin{array}{ccc}
 c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \\
 \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}
 \end{array}$$

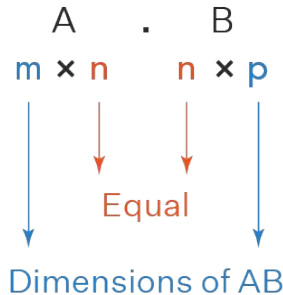
# Matrix multiplication

Dans la matrice résultante :

- ligne 1, col 1 ( $c_{11}$ ) = ligne 1 A X col 1 B
- ligne 1, col 2 ( $c_{12}$ ) = ligne 1 A X col 2 B
- ligne 1, col 3 ( $c_{13}$ ) = ligne 1 A X col 3 B
- ligne 2, col 1 ( $c_{21}$ ) = ligne 2 A X col 1 B
- ligne 2, col 2 ( $c_{22}$ ) = ligne 2 A X col 2 B
- ligne 2, col 3 ( $c_{23}$ ) = ligne 2 A X col 3 B



Multiplication of Matrices



$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

$2 \times 4$        $4 \times 3$        $2 \times 3$

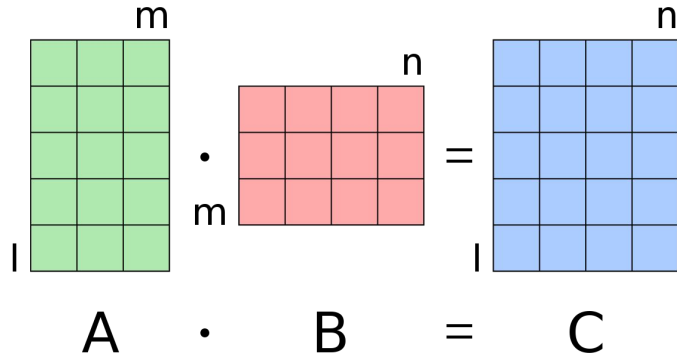
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix}$$

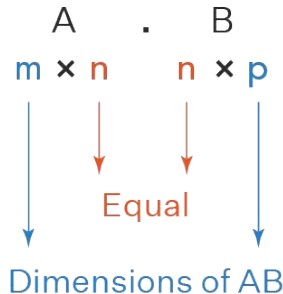
# Matrix multiplication

Dans la matrice résultante :

- ligne 1, col 1 ( $c_{11}$ ) = ligne 1 A X col 1 B
- ligne 1, col 2 ( $c_{12}$ ) = ligne 1 A X col 2 B
- ligne 1, col 3 ( $c_{13}$ ) = ligne 1 A X col 3 B
- ligne 2, col 1 ( $c_{21}$ ) = ligne 2 A X col 1 B
- ligne 2, col 2 ( $c_{22}$ ) = ligne 2 A X col 2 B
- ligne 2, col 3 ( $c_{23}$ ) = ligne 2 A X col 3 B



Multiplication of Matrices



on garde :

- le nb de lignes de A
- le nb de colonnes de B

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}$$

Diagram showing the calculation of  $c_{11}$  as the dot product of the first row of A and the first column of B. The dimensions are  $2 \times 4$  for A,  $4 \times 3$  for B, and  $2 \times 3$  for the result.

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42}$$

Diagram showing the calculation of  $c_{22}$  as the dot product of the second row of A and the second column of B. The dimensions are  $2 \times 4$  for A,  $4 \times 3$  for B, and  $2 \times 3$  for the result.

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

$$A * B = \begin{pmatrix} & & \\ & & \end{pmatrix}$$

$$A * B = \begin{pmatrix} & \\ & \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

$$A * B = \begin{pmatrix} & & \\ & & \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} & & \\ & & \end{pmatrix}$$



# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} ? \\ ? \end{pmatrix}$$
$$A * B = \begin{pmatrix} \phantom{0} & \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} & \phantom{0} \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} 1*5 + 2*8 & & \\ & & \\ & & \end{pmatrix}$$
$$A * B = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ & & \end{pmatrix}$$
$$A * B = \begin{pmatrix} & & \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$
$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 29 & 36 & 43 \end{pmatrix}$$

# Matrix multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

Dimension?

A: (2 x 2)

B: (2 x 3)

A\*B: (2 x 3)

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$
$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

# What was wrong

$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \text{ matrix ( the weights for hidden layer 1)}$$

*the output is given by*

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ (the product of vector and matrices)}$$

*this is done by taking each row of the first matrix and doing element wise multiplication with each column of the second matrix.*

*thus,*

$$h1 = w1.x1 + w2.x2$$

$$h2 = w3.x1 + w3.x2$$

$$h3 = w5.x1 + w6.x2$$

# What was wrong

$$X = \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ vector (input layer)}$$

$$W = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \text{ matrix ( the weights for hidden layer 1)}$$

*the output is given by*

$$\begin{bmatrix} h1 \\ h2 \\ h3 \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w4 & w5 \\ x3 & w6 \end{bmatrix} \cdot \begin{bmatrix} x1 \\ x2 \end{bmatrix} \text{ (the product of vector and matrices)}$$

*this is done by taking each row of the first matrix and doing element wise multiplication with each column of the second matrix.*

*thus,*

$$h1 = w1.x1 + w2.x2$$

$$h2 = w3.x1 + w3.x2$$

$$h3 = w5.x1 + w6.x2$$

$$h2 = w4.x1 + w5.x2$$

$$h3 = w3.x1 + w5.x2$$

or

change W (but w3  
appears twice 😞)