RÉSEAUX NEURONAUX : UN APERÇU

PHI 3330

Séance 3

Jonathan Simon

PROGRAMME

- 1) Qu'est-ce qu'un réseau neuronal artificial? (ANN)
- 2) Le cas le plus simple : un perceptron
- 3) Les neurons en tant que transfomations géometriques
- 4) Les perceptrons peuvent mettre en ouvre les systèmes de symboles (portes NAND)

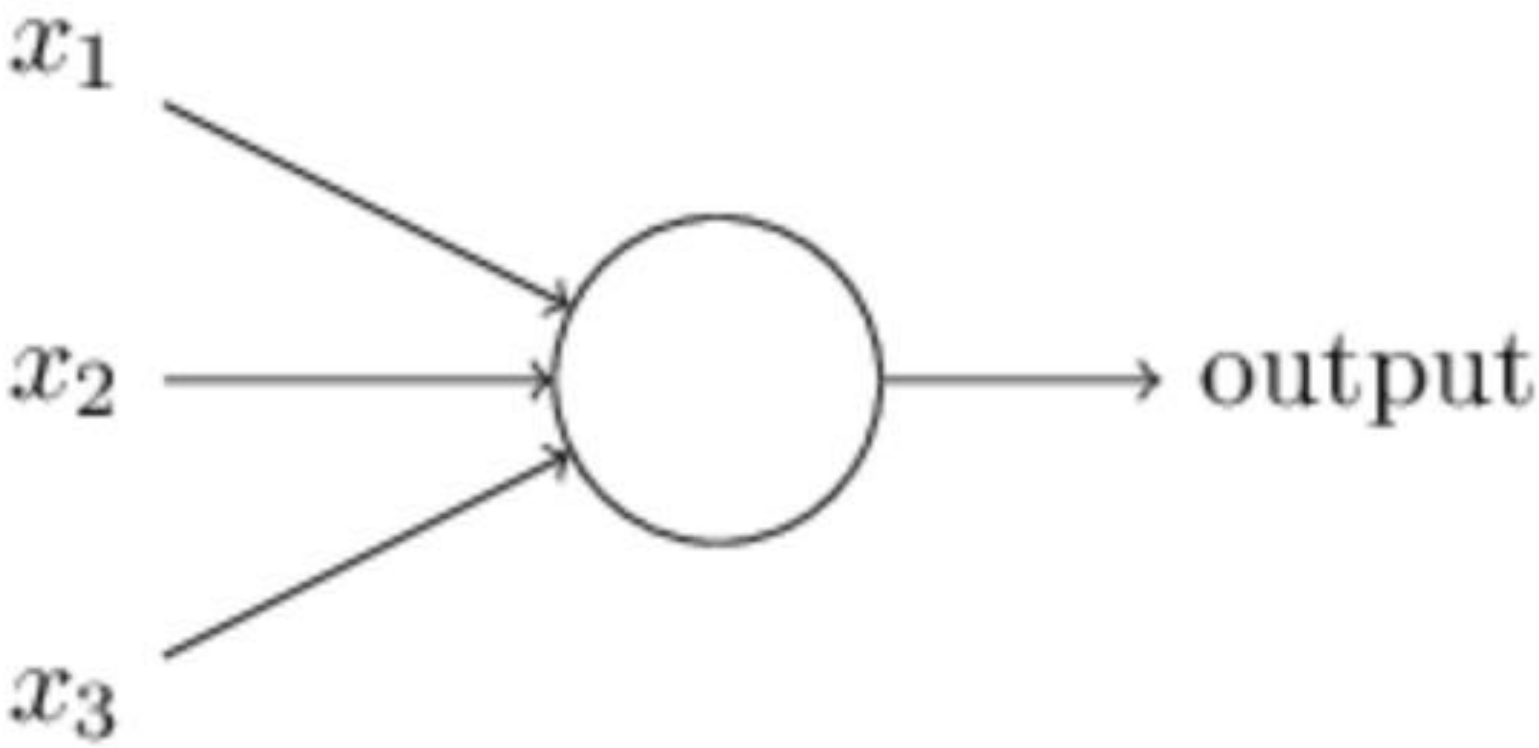
PROGRAMME

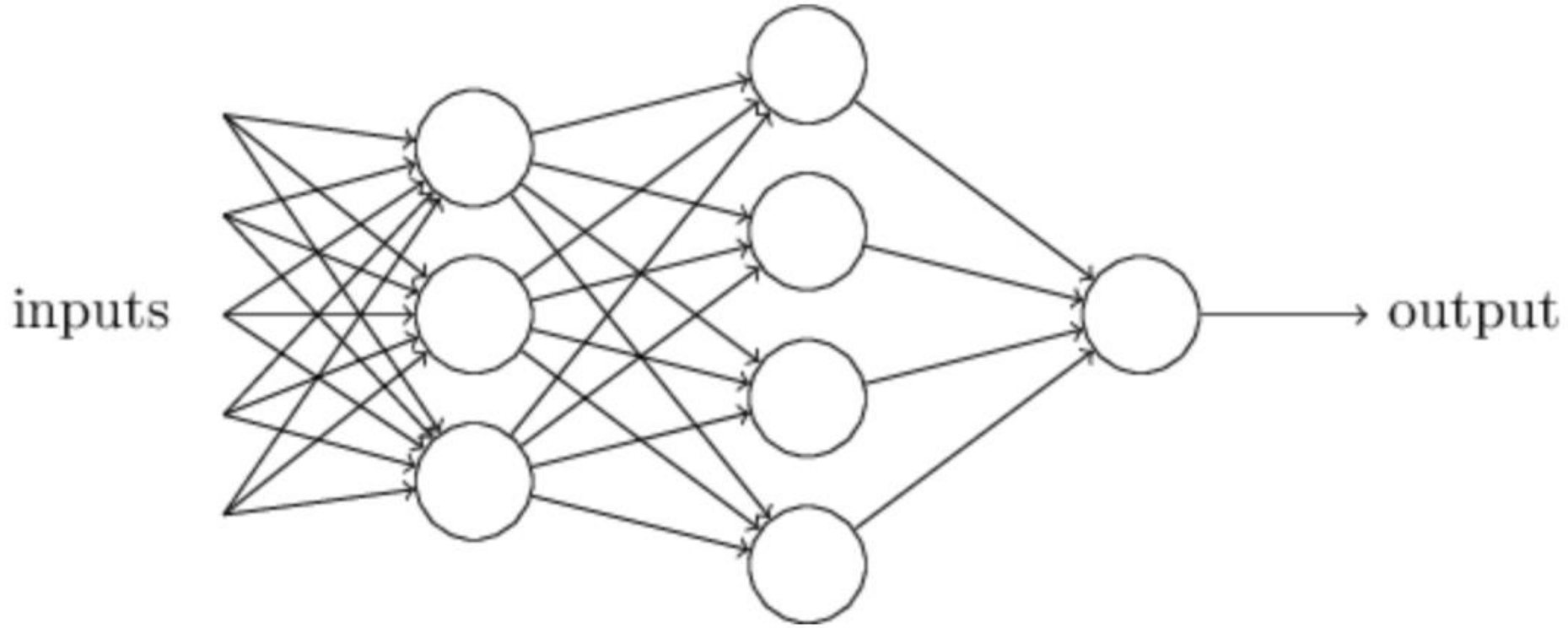
- 5) Activation sigmoïde et réseaux neuronaux capables d'apprendre
- 6) l'apprentissage des fonctions comme l'apprentissage de modèles prédictifs du monde
- 7) Le théorème d'approximation universelle : comment un réseau neuronal peut approximer toute fonction continue
- 8) Descente de gradient et backpropagation : comment les réseaux neuronaux apprennent
- 9) Pourquoi c'est difficile : pourquoi nous avons besoin de réseaux neuronaux profonds (réseaux neuronaux avec de nombreuses couches) et d'architectures compliquées pour permettre l'apprentissage de choses compliquées.

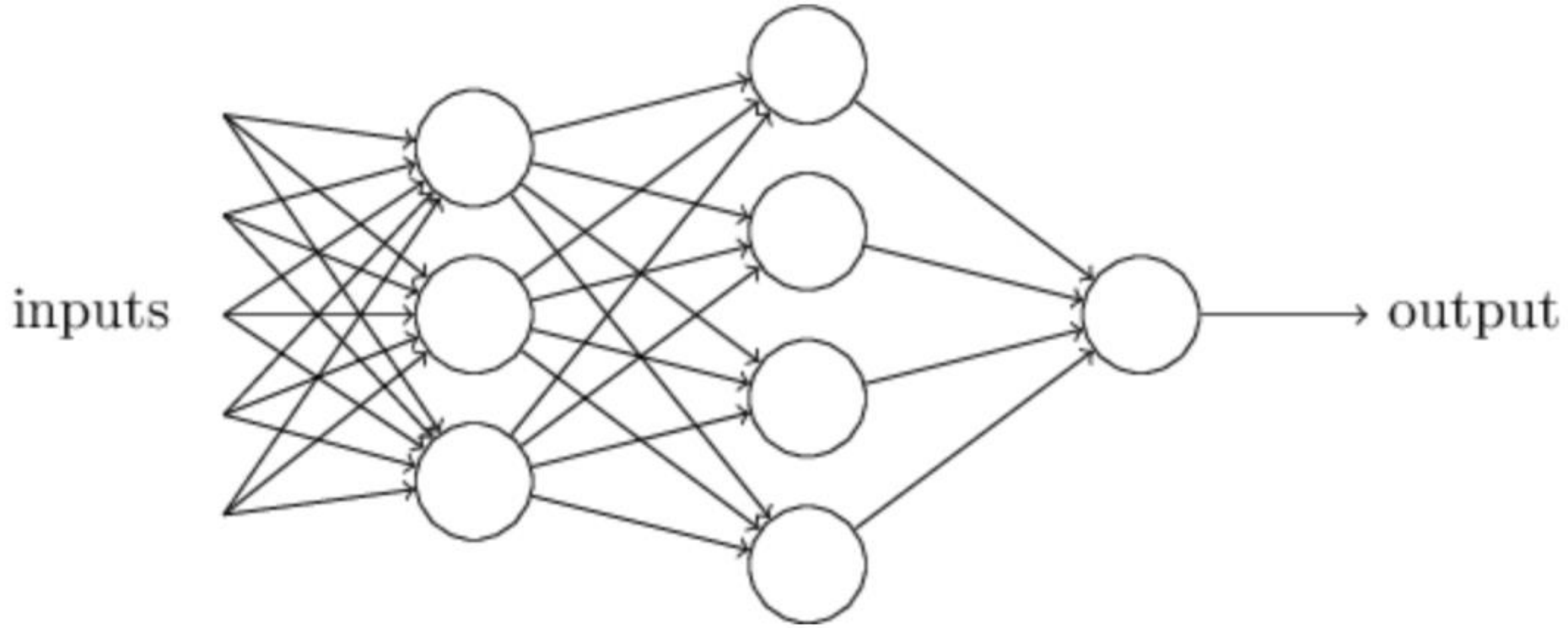
QU'EST-CE QU'UN RÉSEAU NEURONAL ARTIFICIAL? (ANN)

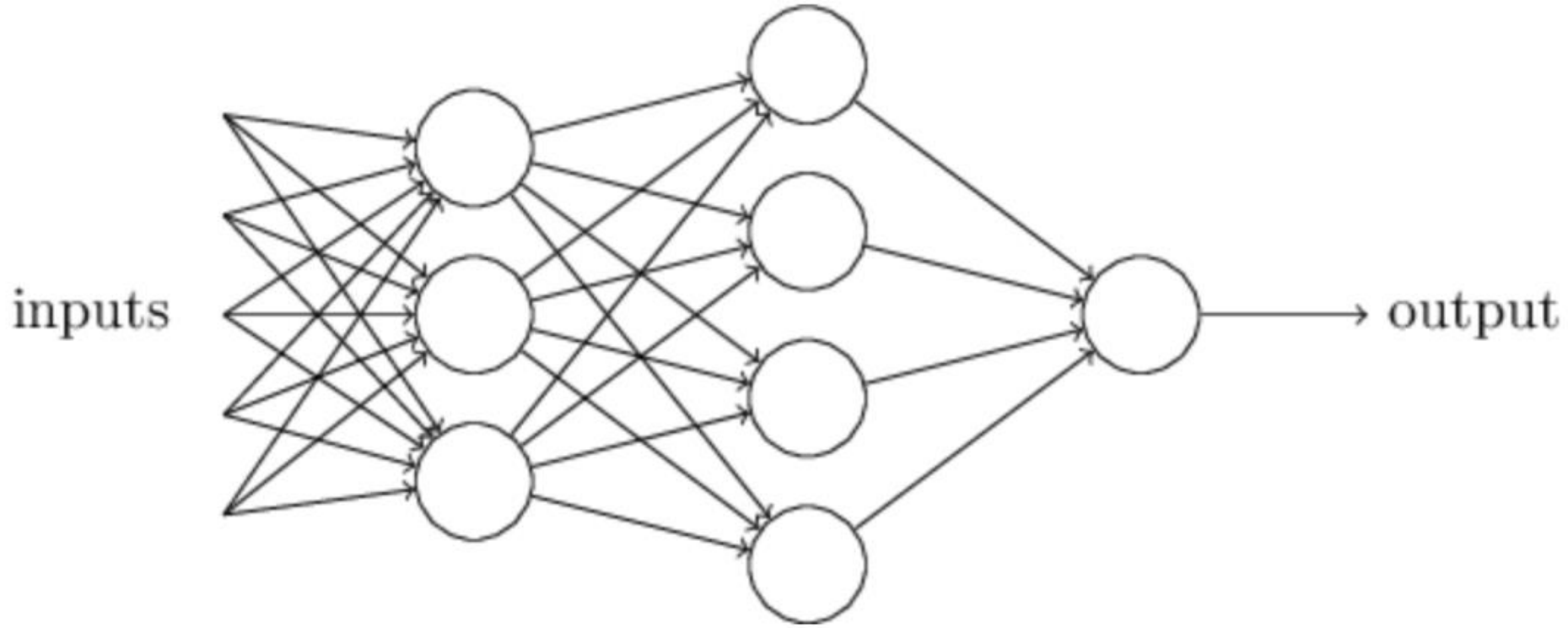
QU'EST-CE QU'UN RÉSEAU NEURONAL ARTIFICIAL? (ANN)

- Un réseau neuronal est une fonction complexe, composée de fonctions plus petites : F(G(x))
- Les plus petites fonctions sont les neurones : un neurone prend un nombre fixe d'entrées et délivre une sortie unique en fonction de ces entrées. (intuitivement : il décide de se décharger ou non, en fonction de ses entrées)


x_1
x_2
x_3
output


inputs
output


inputs
output
X
H
G
F

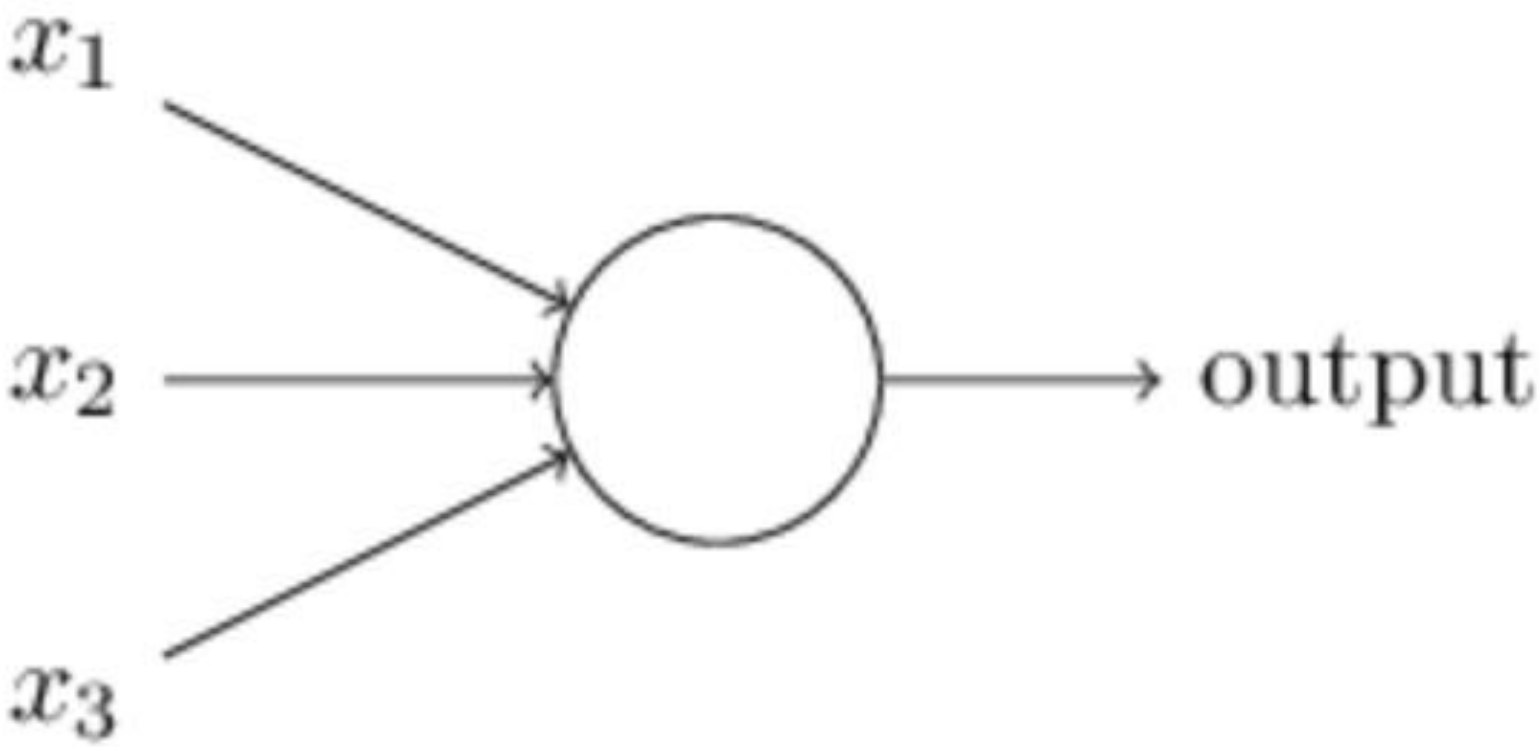


X H(X) G(H(X)) F(G(H(X)))

QU'EST-CE QU'UN RÉSEAU NEURONAL ARTIFICIAL? (ANN)

- Observez que ce sont des entités mathématiques : abstraites, plutôt qu'*artificielles*.
- En tant que telles, elles peuvent être mises en œuvre par des systèmes biologiques aussi bien que par des systèmes synthétiques.

LE CAS LE PLUS SIMPLE : UN PERCEPTRON

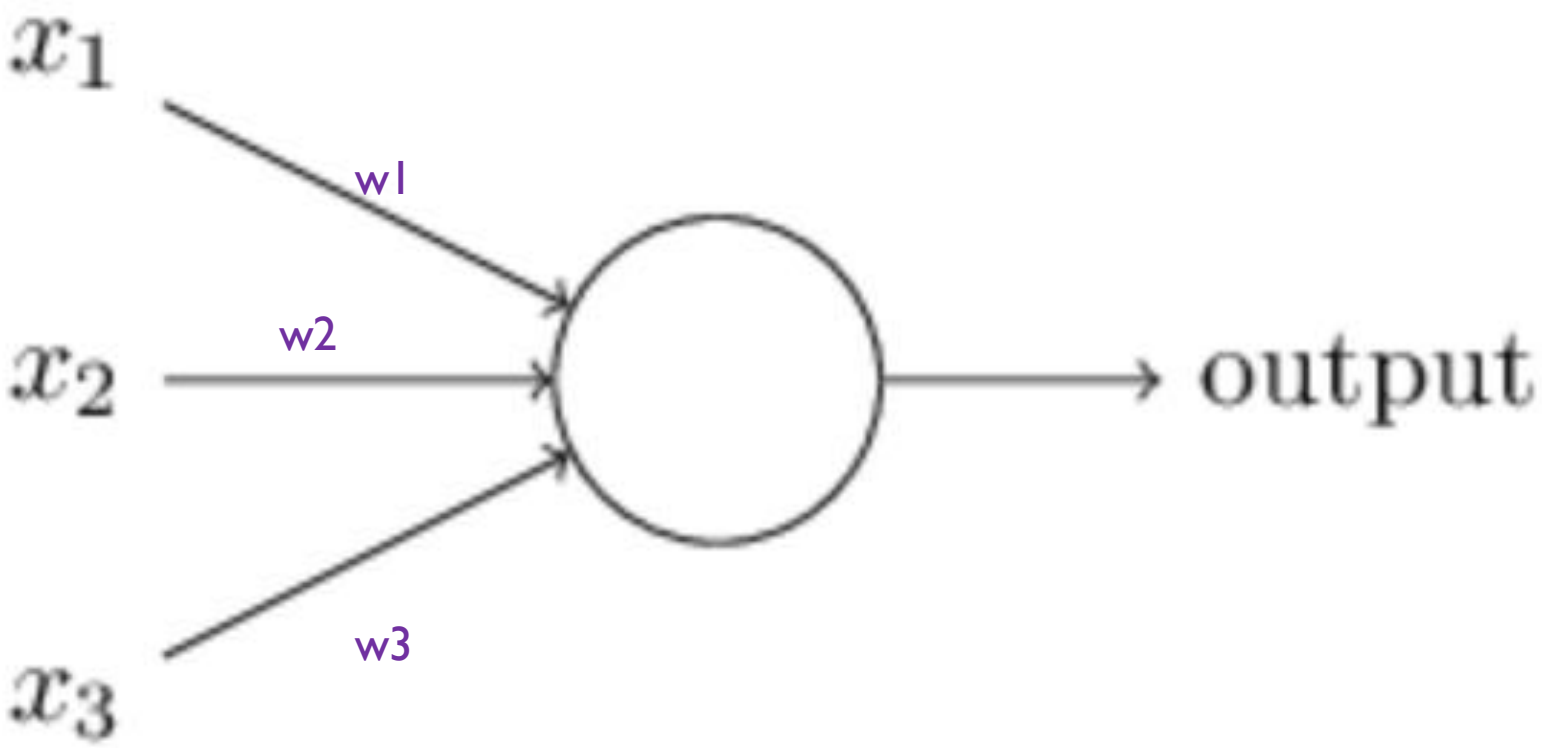

x_1
x_2
x_3
output

UN PERCEPTRON

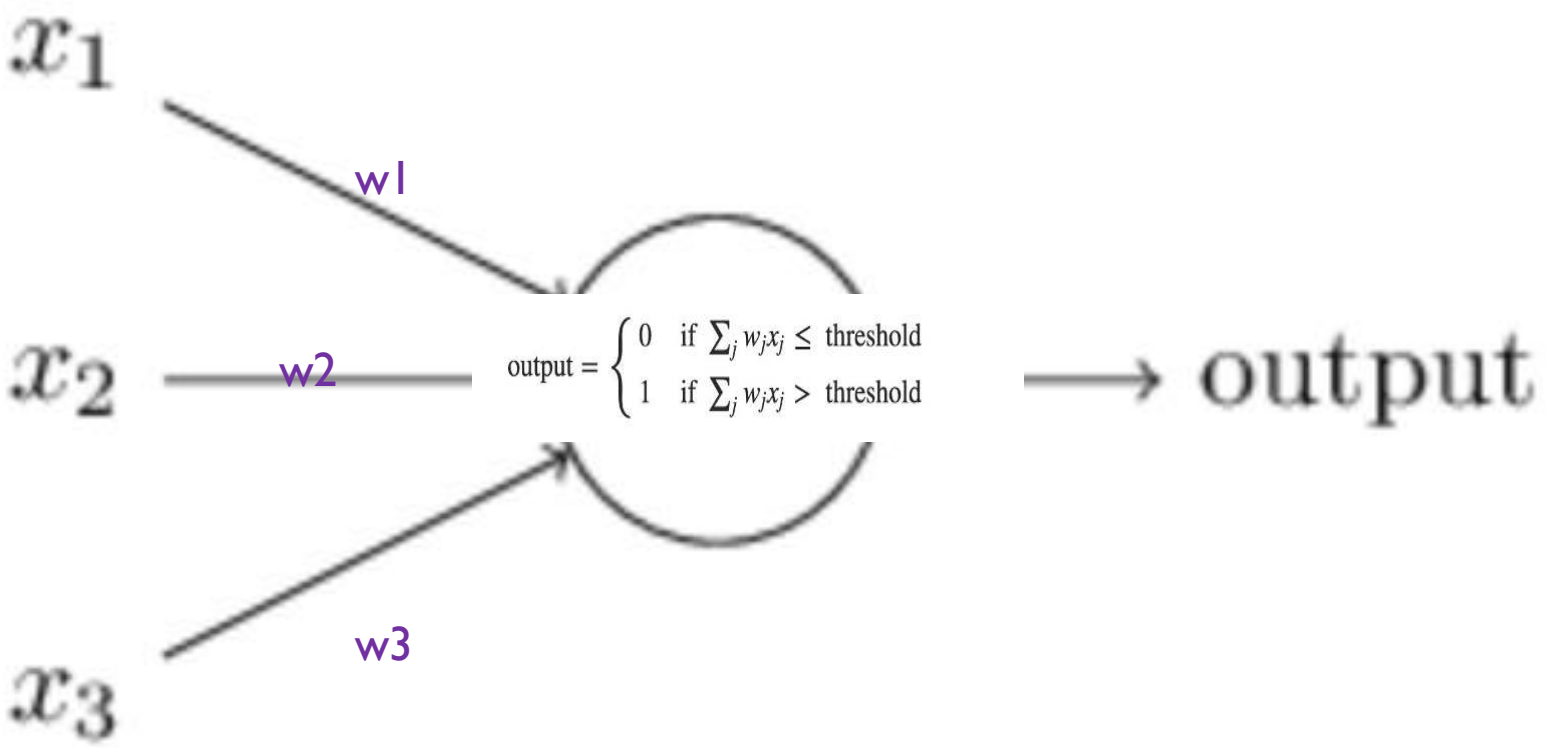
- Un perceptron est un neurone dont l'activation est fixée par un seuil : il sort 0 si le seuil n'est pas atteint, 1 s'il l'est.
- (note : cela signifie que les perceptrons ne sont pas différentiables, et que de très petits changements dans l'entrée conduisent à des sauts de 0 à 1 dans la sortie)

UN PERCEPTRON

- Nous comprenons également que le neurone attache un poids, *w*, à chaque entrée *x* : intuitivement, combien il se soucie de cette entrée
- (alternativement le *volume* de ce canal)


x_1
x_2
x_3
w1
w2
w3
output

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$


x_1
x_2
x_3
w1
w2
w3
output = $\begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$
output

UN PERCEPTRON

- Ainsi, pour décider s'il se décharge sur (x1,x2,x3), nous multiplions d'abord : x1 * w1, puis x.2 * w2, puis x3 * w3 ... puis nous les additionnons, et si la somme est supérieure à la valeur seuil, le neurone tire

- (j'utilize « * » pour symboliser multiplication)

UN PERCEPTRON

- Exemple : le neurone décide si vous allez au festival.
- *Entrée 1* = x1 = probabilité qu'il va pleuvoir.
- *Entrée 2* = x2 = probabilité que votre ami va venir.
- *Entrée 3* = x3 = probabilité que ca va être facile de s'y rendre.

UN PERCEPTRON

- Peut-être que vous vous souciez surtout de la météo, alors w1 = 6.
- Cela ne vous dérange pas vraiment d'y aller seul, donc w2 = 2,
- et vous vous souciez moyennement du temps que cela prendra, donc w3 = 4.
- Alors si nous fixons le seuil à 5 :

UN PERCEPTRON

- Les probabilités: certain qu'il pleut, votre ami vient, et c'est proche=
- (0 * 6) + (1 * 2) + (1 * 4) = 6, qui est plus grand que cinq, donc le neurone sort 1 = tu y vas

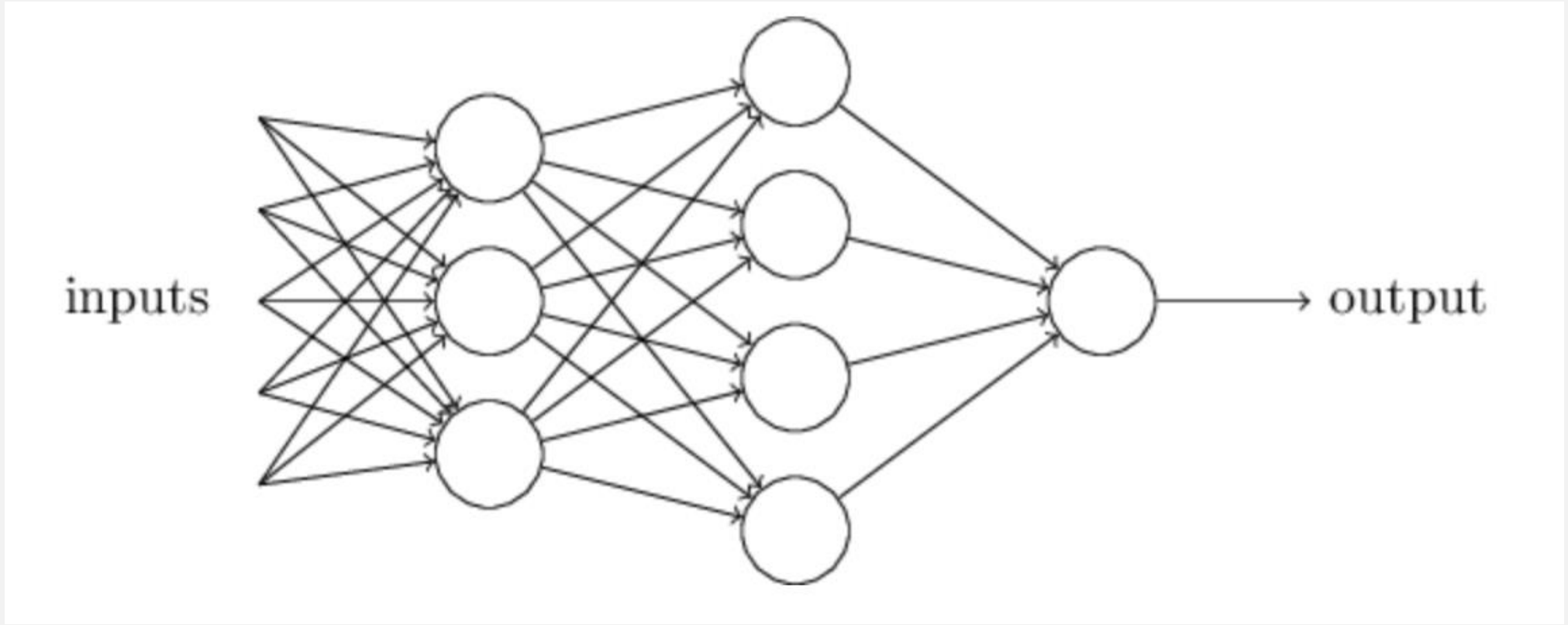
UN PERCEPTRON

- Plus typique : 85 % de chances de pluie, 80 % de chances que ton ami vienne, 75 % de chances qu'il n'y ait pas d'embouteillage=

- (.15 * 6) + (.8 * 2) + (.75 * 4) = 5.5, qui est plus grand que cinq, donc le neurone sort 1 = tu y vas

UN PERCEPTRON

- C'est l'idée générale. Notez que vous pouvez composer des perceptrons dans un réseau, ce qui permettra une hiérarchie de décisions plus complexes :


inputs
output

UN PERCEPTRON

- Peut-être que la deuxième couche de décision est de savoir si vous demandez plus d'argent à vos parents : un facteur est de savoir si vous allez au festival, un autre est de savoir si vous voulez acheter un nouvel ordinateur, etc...
- À chacun de ces facteurs vous pourriez attribuer un poids de combien il devient alors plus important pour vous de demander de l'argent...

UN PERCEPTRON

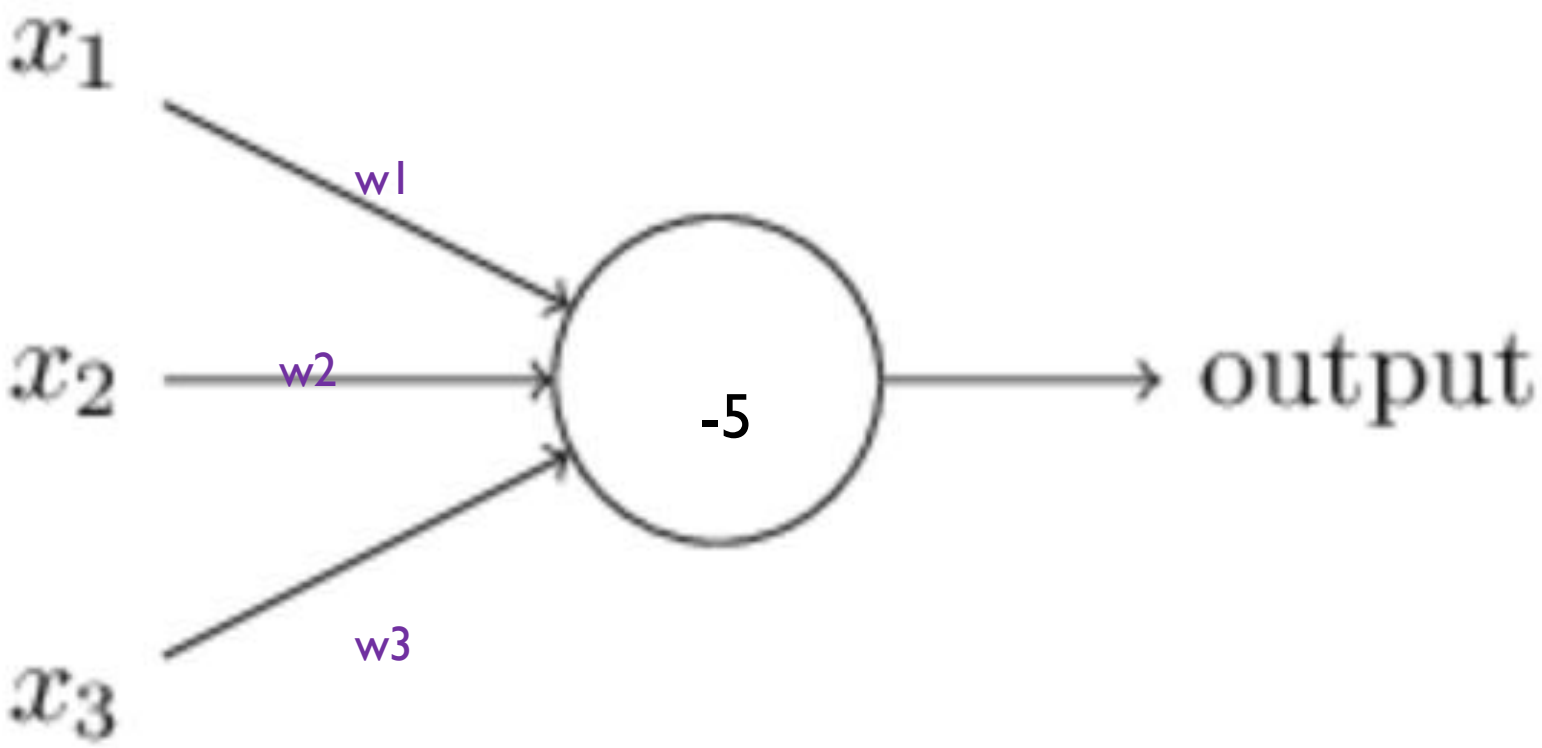
- Mais notez que si les entrées de ce deuxième niveau seront la sortie d'un premier niveau de perceptrons, alors ses entrées doivent toutes être 0 ou 1....

UN PERCEPTRON

- Au lieu de permettre au seuil d'être un nombre arbitraire, il peut être plus intuitif de penser que le seuil est toujours zéro. Mais nous voulons toujours capturer l'idée que « *la somme des termes doit être égale à 5 pour que le neurone se décharge* ». Si le seuil est zéro, cela signifie que nous avons besoin d'un terme supplémentaire de -5. C'est le *biais*.

UN PERCEPTRON

- $(x1 * w1) + (x2 * w2) + (x3 * w3) \geq 5$
- Si et seulement si :
- $(x1 * w1) + (x2 * w2) + (x3 * w3) \text{ -5} \geq 0$

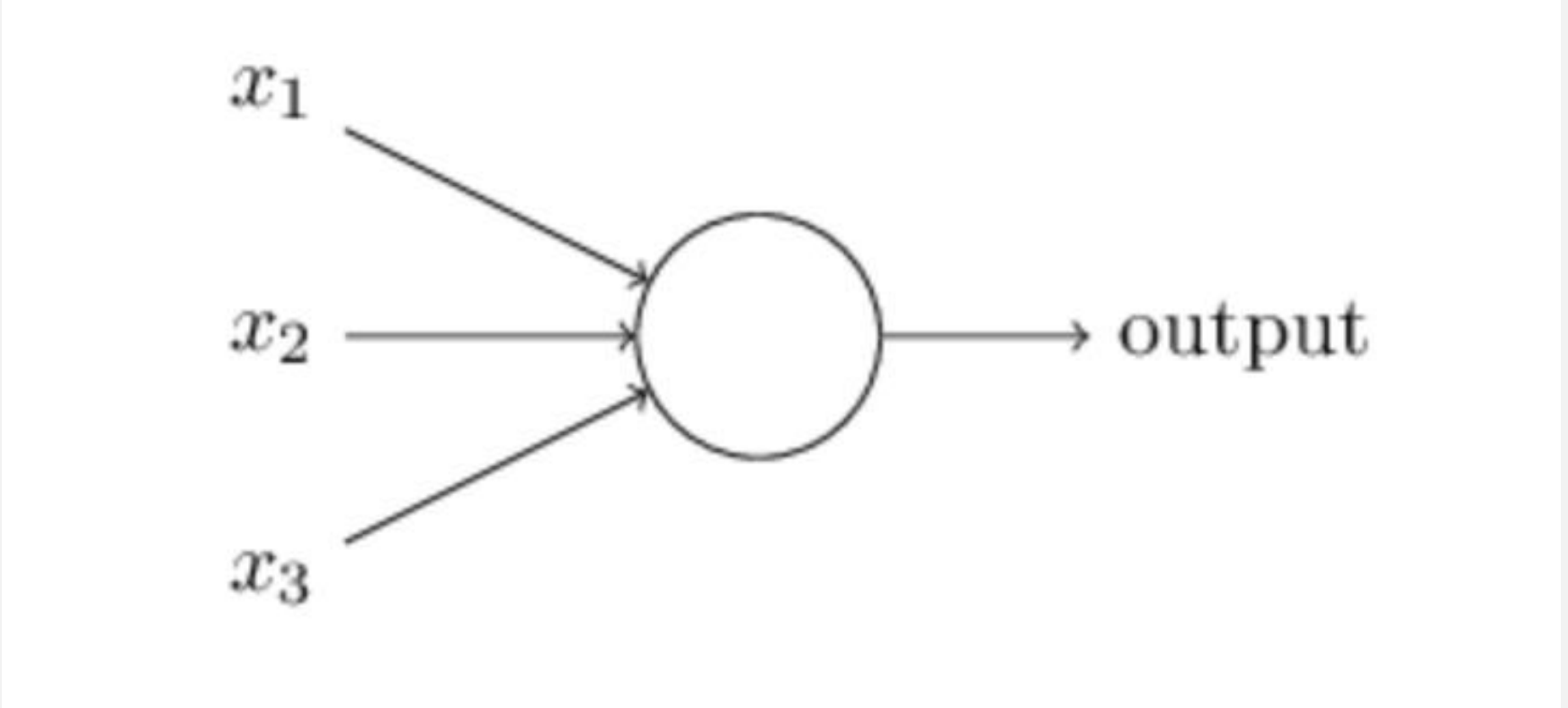

x_1
x_2
x_3
w1
w2
w3
-5
output

LES NEURONS EN TANT QUE TRANSFOMATIONS GÉOMETRIQUES

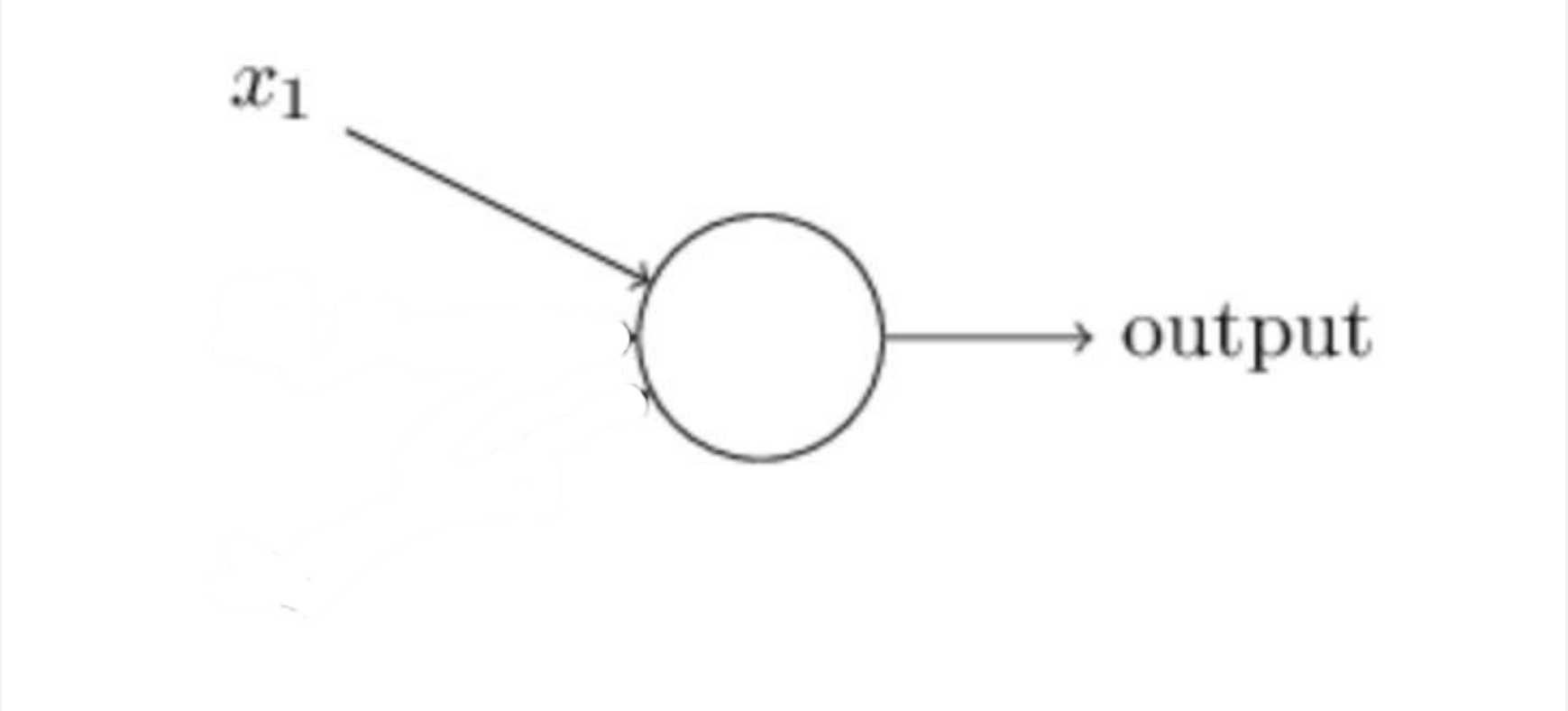
TRANSFORMATIONS

- 1) Neurones à 1 entrée
- 2) Neurones à 2 entrées
- 3) Neurones avec n entrées...

NEURONES À I ENTRÉE



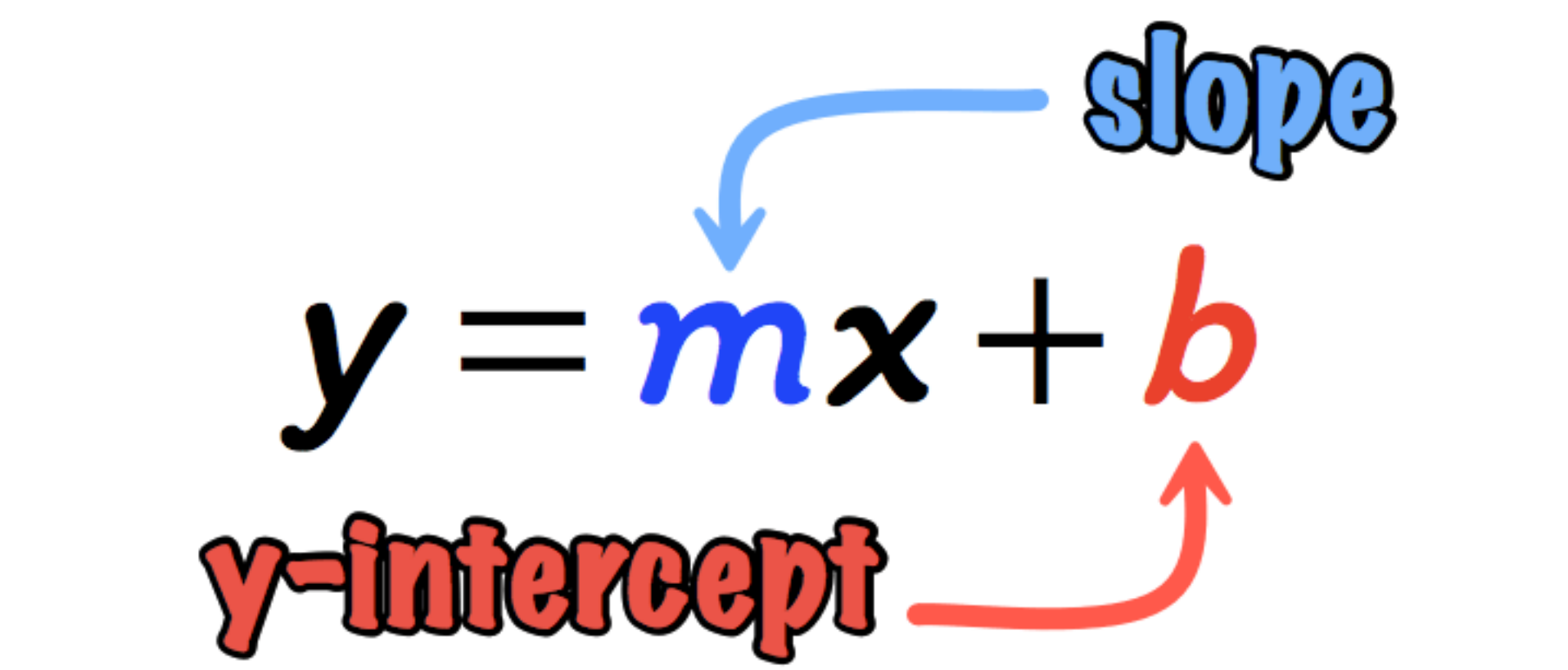
NEURONES À 1 ENTRÉE

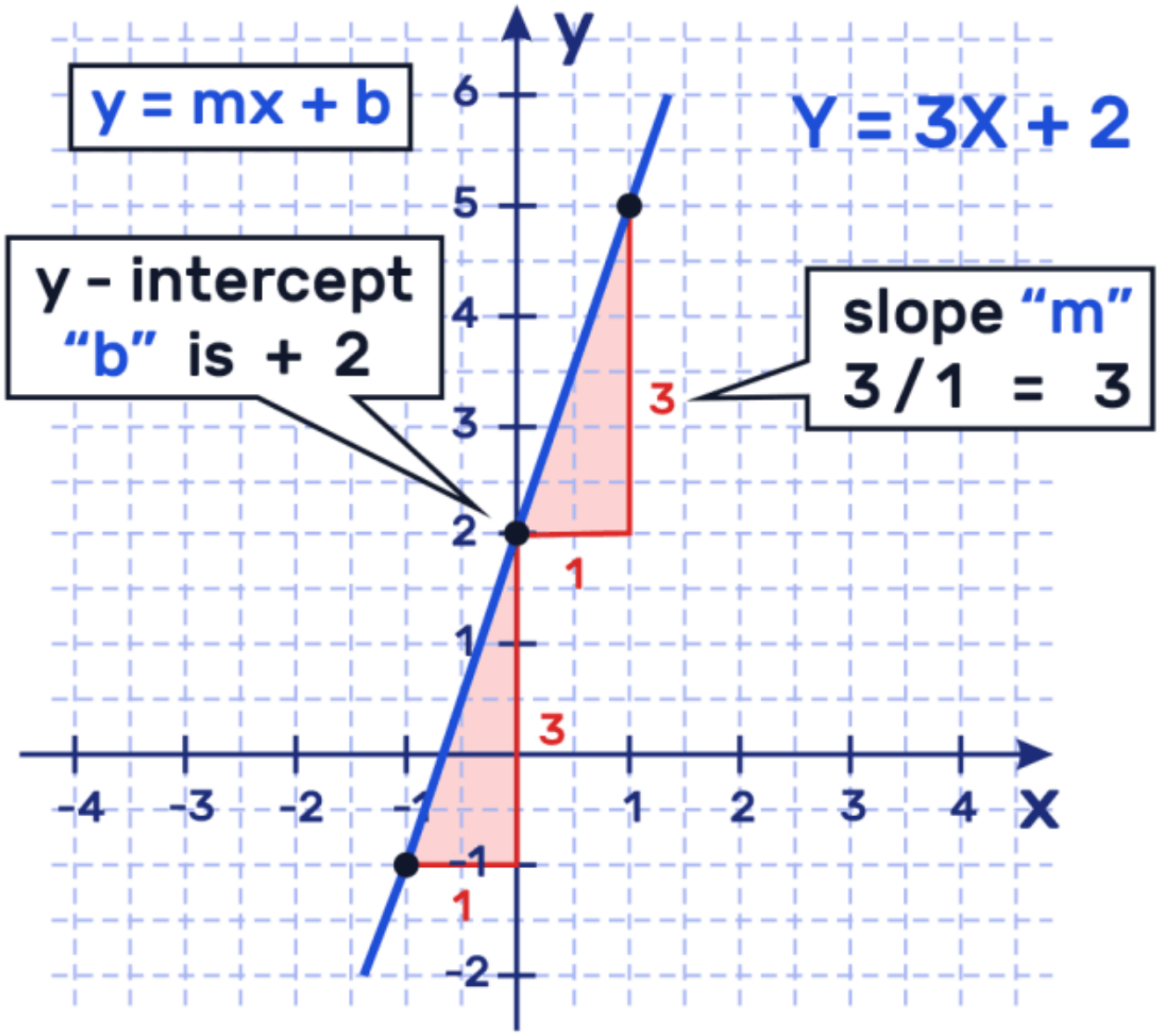


MX + B

- $(x1 * w1) \geq 5$
- Si et seulement si :
- $(x1 * w1) -5 \geq 0$

- $Mx + b \geq 0$

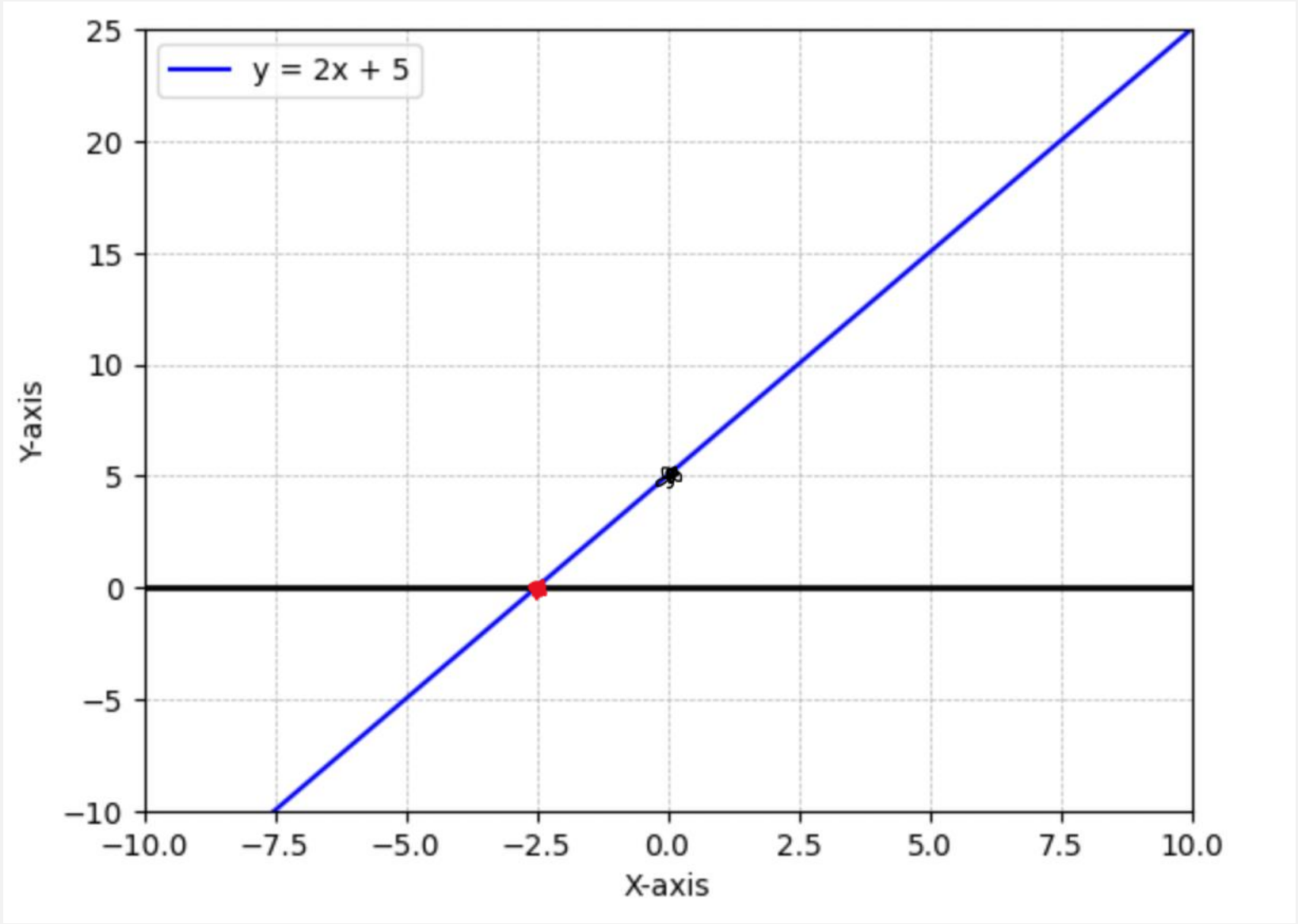

slope
$y = mx + b$
y-intercept

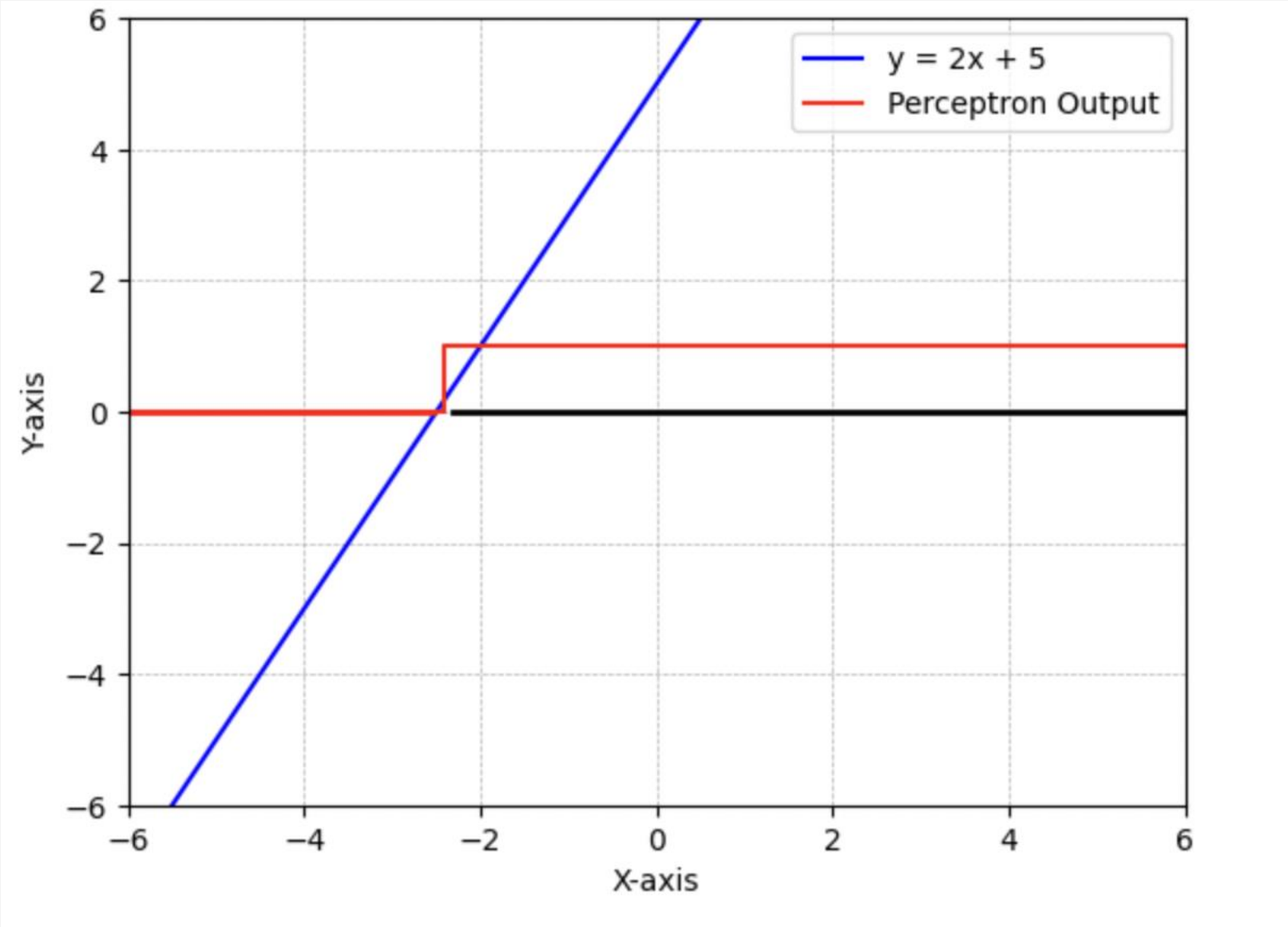

y = mx + b
Y = 3X + 2
y - intercept
"b" is + 2
slope "m"
3 / 1 = 3
y
X
6
5
4
3
2
1
-1
-2
-4
-3
-2
-1
1
2
3
4
3
1
3
1

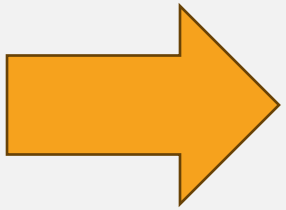
Brighterly

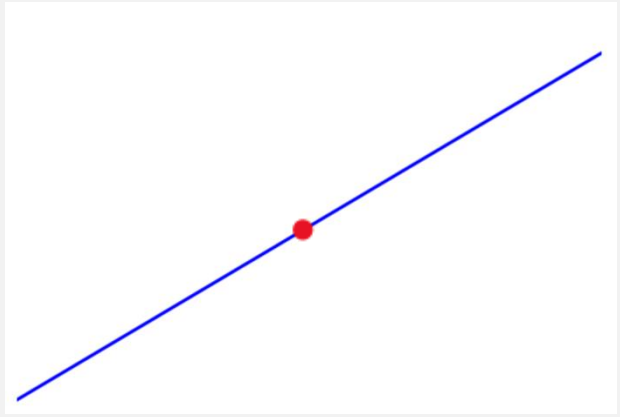
MX + B

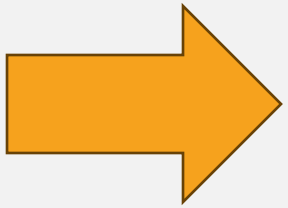
- On peut considérer qu'un tel neurone va:
- 1) Faire correspondre x à mx + b (un point sur la ligne y = mx + b)
- 2) Vérifier si la valeur obtenue est supérieure à 0 (cela signifie que y = 0 est un point très important sur la droite y = mx + b).
- 3) Si le neurone est un perceptron, sa sortie est une non-linéarité, donnée par le fait que la valeur résultante est supérieure à zéro (alors la sortie est 1) ou que la valeur résultante est égale ou inférieure à zéro (alors la sortie est 0).

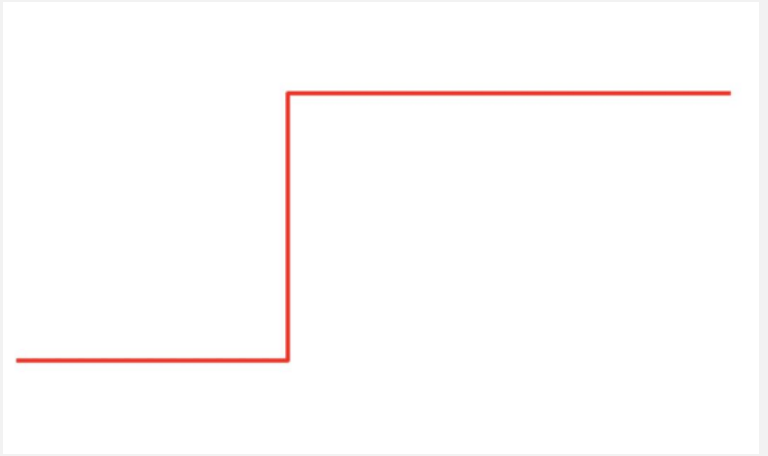

y = 2x + 5
25
20
15
10
5
0
−5
−10
Y-axis
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
X-axis


y = 2x + 5
Perceptron Output
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6







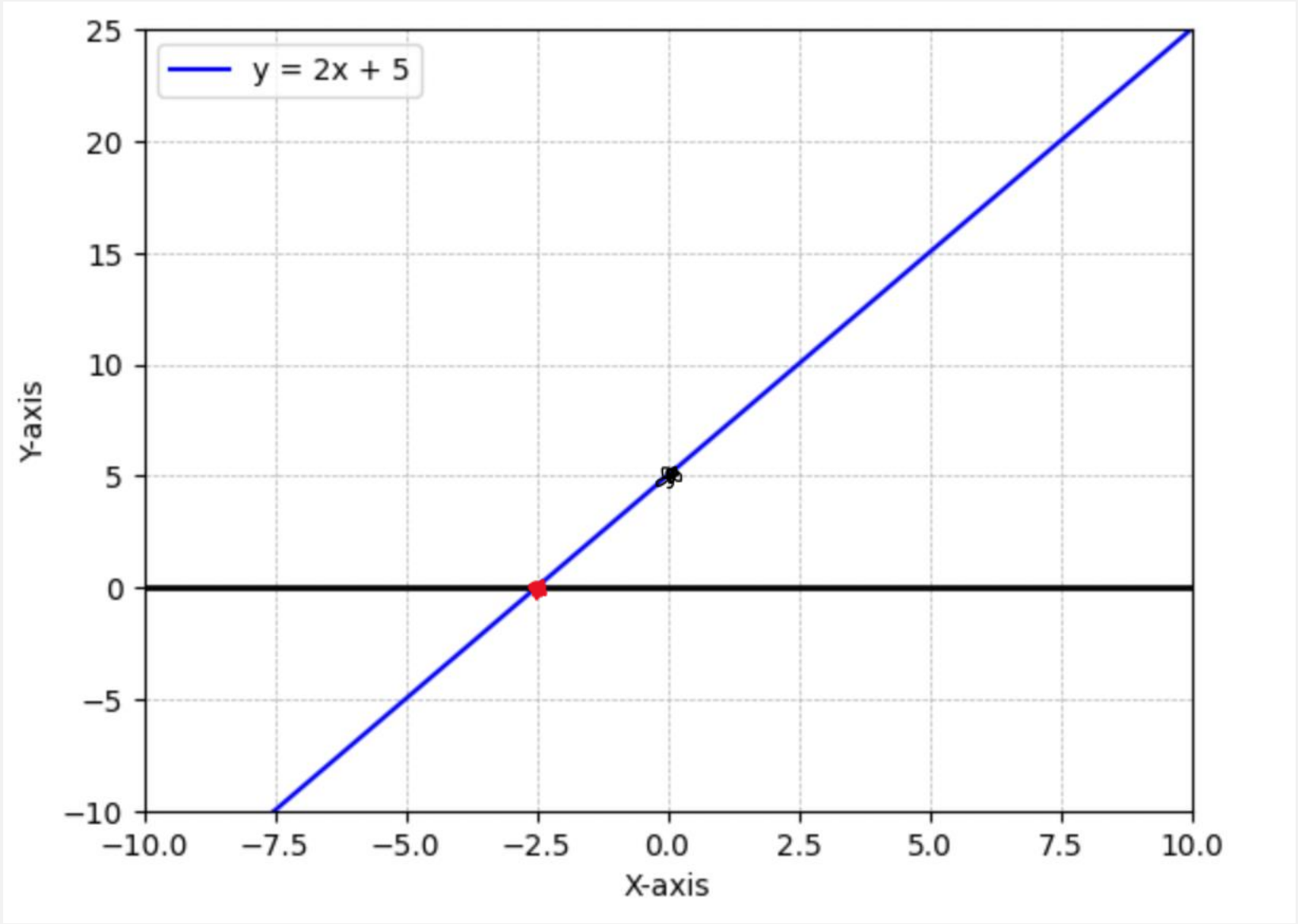


PERCEPTRON

- Peut être divisée entre:
- 1) Transformation linéaire
- 2) Transformation non linéaire (fonction en escalier)

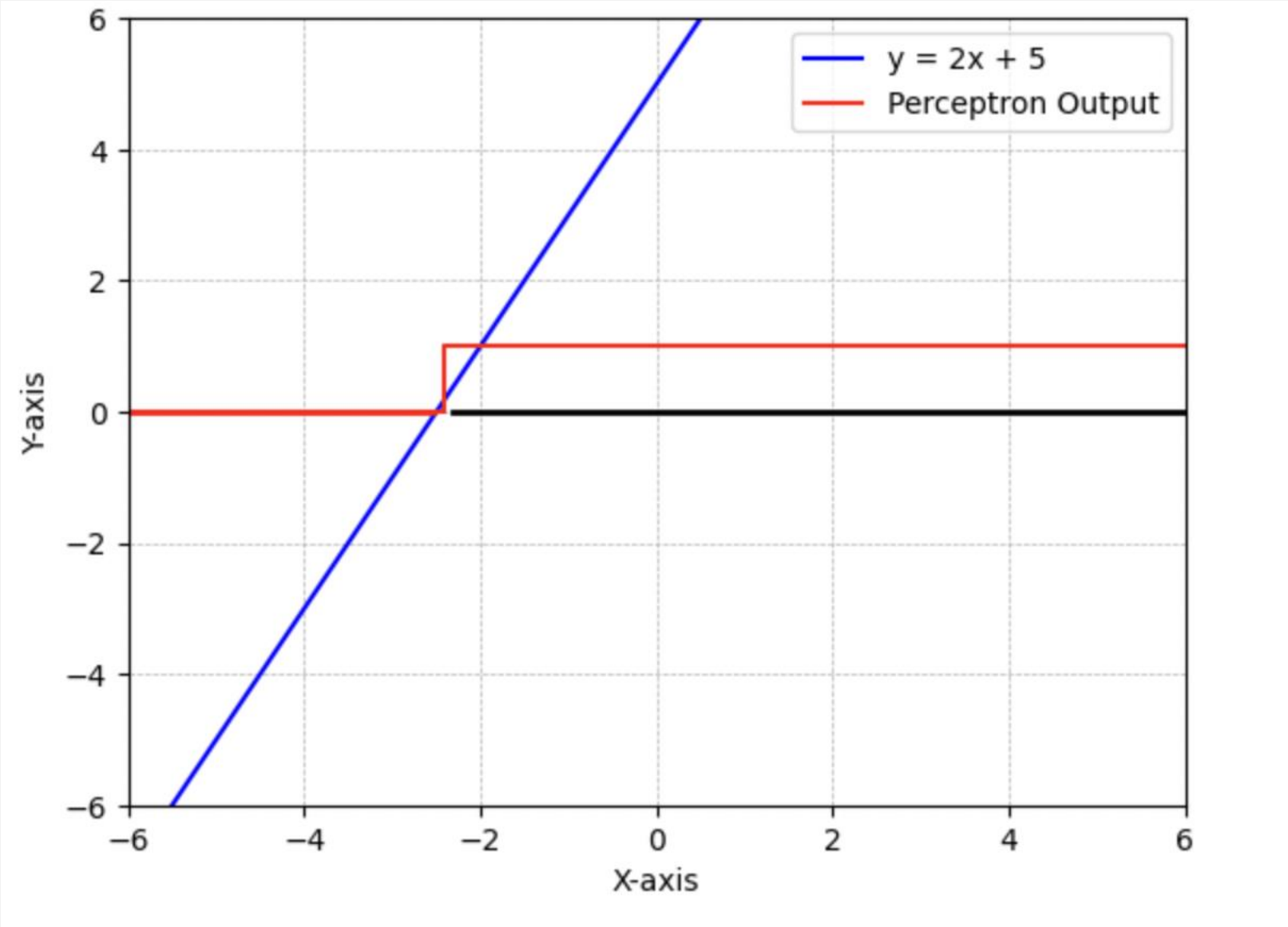
PERCEPTRON

- 1) Transformation linéaire
- Fonction linéaire prenant toutes les valeurs d'entrée et produisant une seule valeur en sortie
- Le « zéro » donne une surface de décision : les valeurs émises supérieures à zéro correspondent à « oui », les valeurs émises inférieures ou égales à zéro correspondent à « non ».

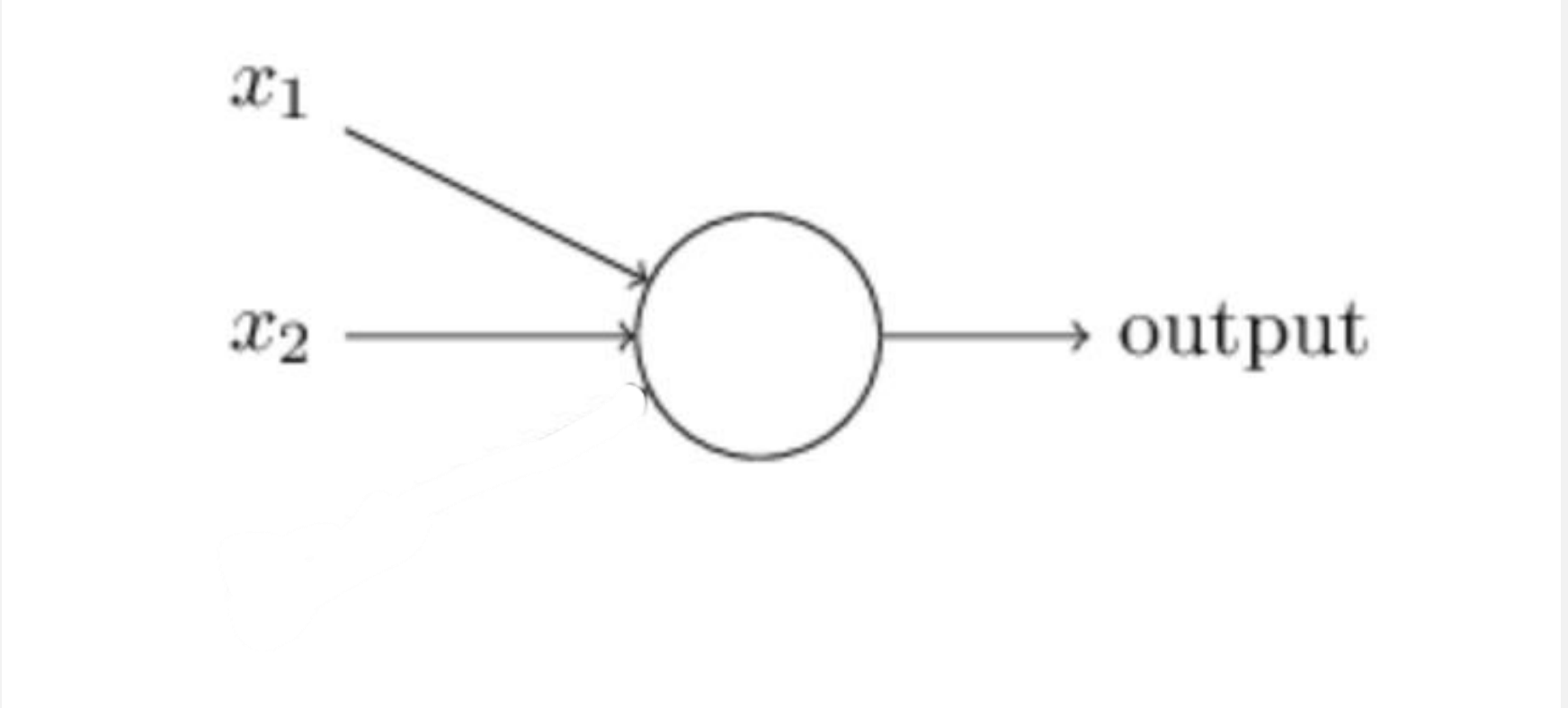

y = 2x + 5
25
20
15
10
5
0
−5
−10
Y-axis
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
X-axis

PERCEPTRON

- 1) Transformation non linéaire
- Prend une décision sur la base de la surface de décision (pour un perceptron, il s'agit d'une fonction en escalier : envoie les valeurs 0 ou inférieures de la surface de décision à 0, les valeurs supérieures à 0 de la surface de décision à 1).


y = 2x + 5
Perceptron Output
Y-axis
X-axis
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6

NEURONES À 2 ENTRÉES



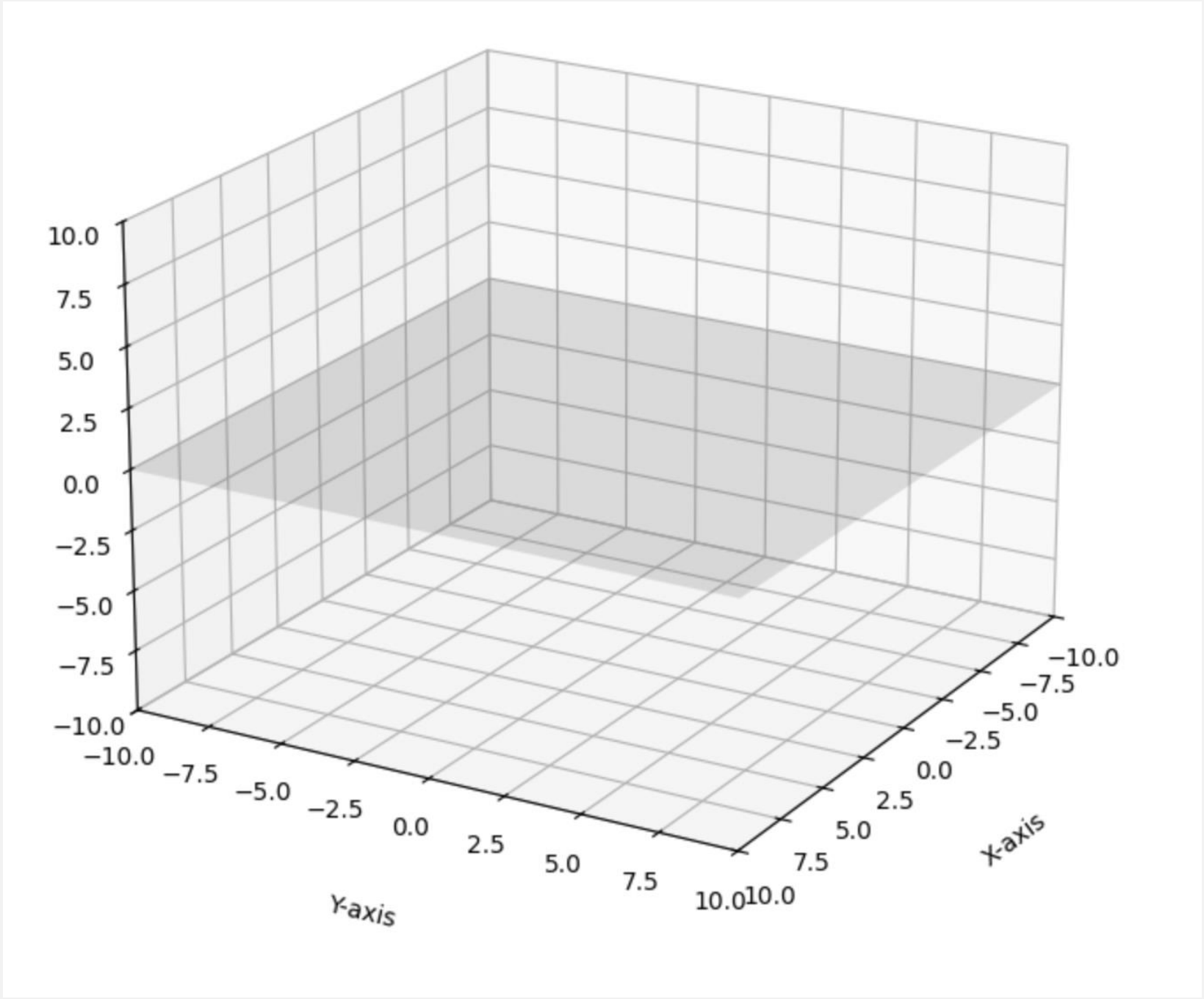
SIMPLIFIONS

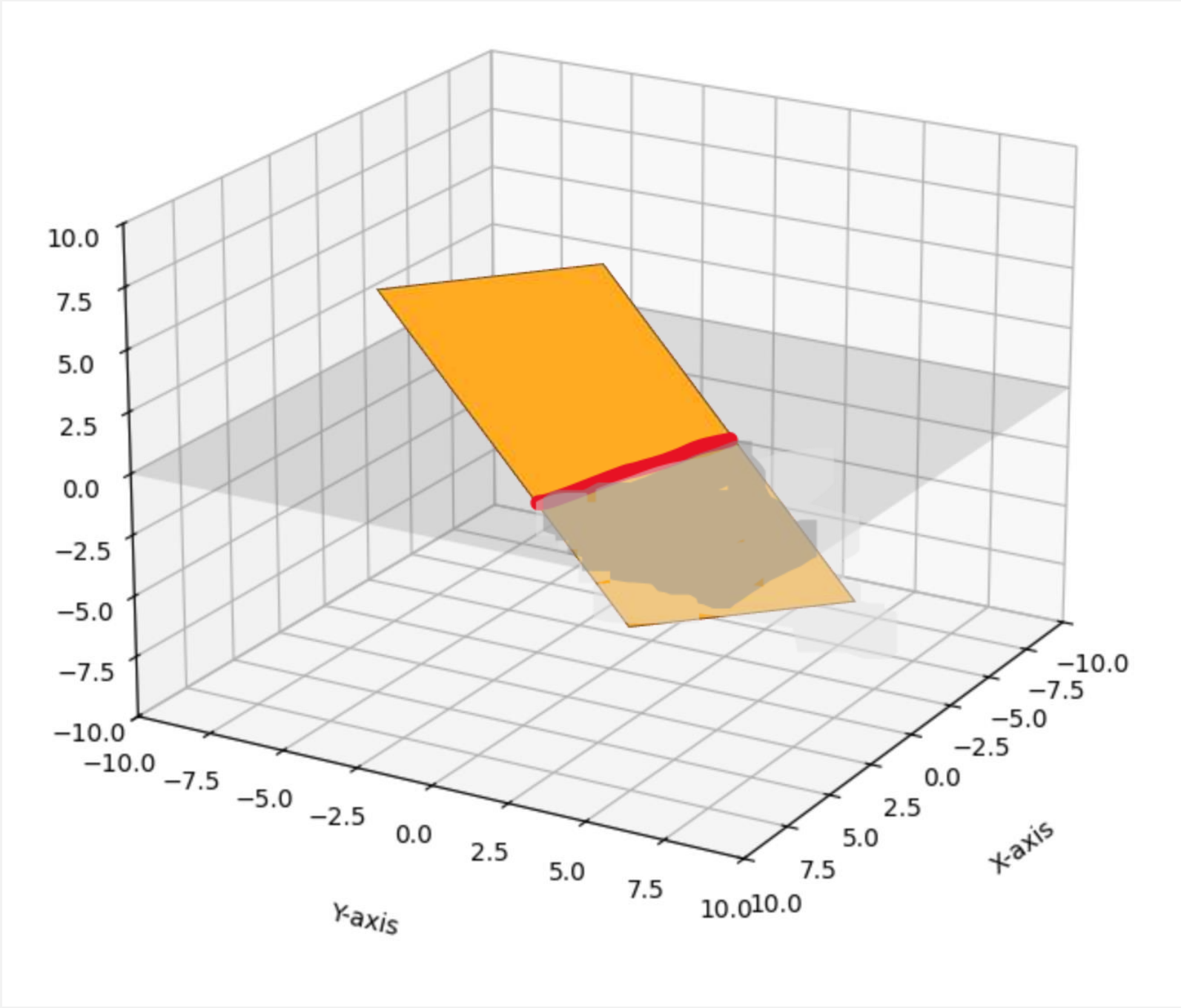
- $(x1 * w1) + (x2 * w2) \geq 5$
- Si et seulement si :
- $(x1 * w1) + (x2 * w2) - 5 \geq 0$

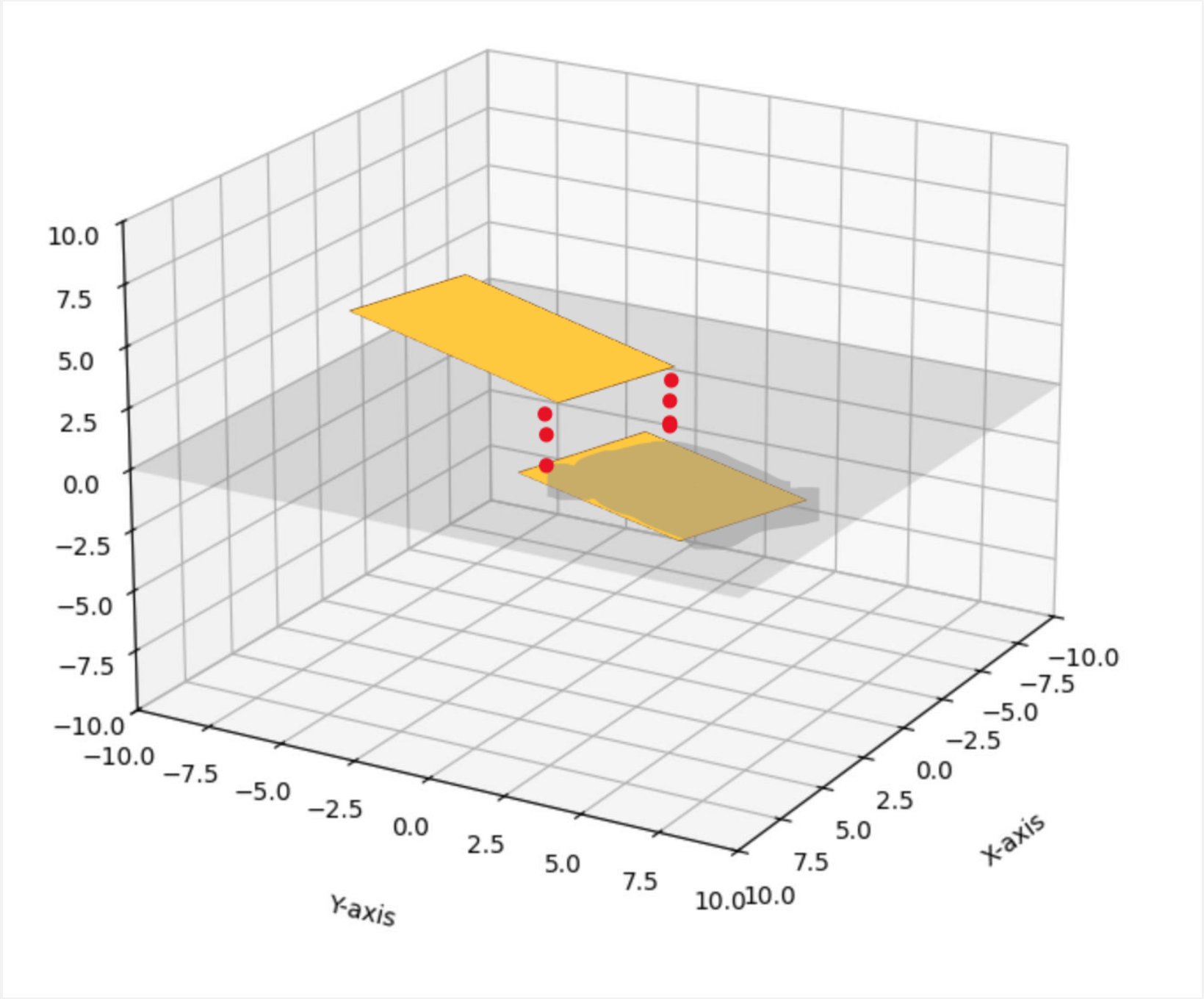
MX + B

- (x1 * w1) + (x2 * w2) + -5 ≥ 0
- (x1 * w1) + (x2 * w2) -5
- M = [w1 w2], X = $\begin{bmatrix} x1 \\ x2 \end{bmatrix}$, b = -5
- MX + b

NEURONES À 2 ENTRÉES


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Y-axis
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
X-axis


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Y-axis
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
X-axis


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Y-axis
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
X-axis

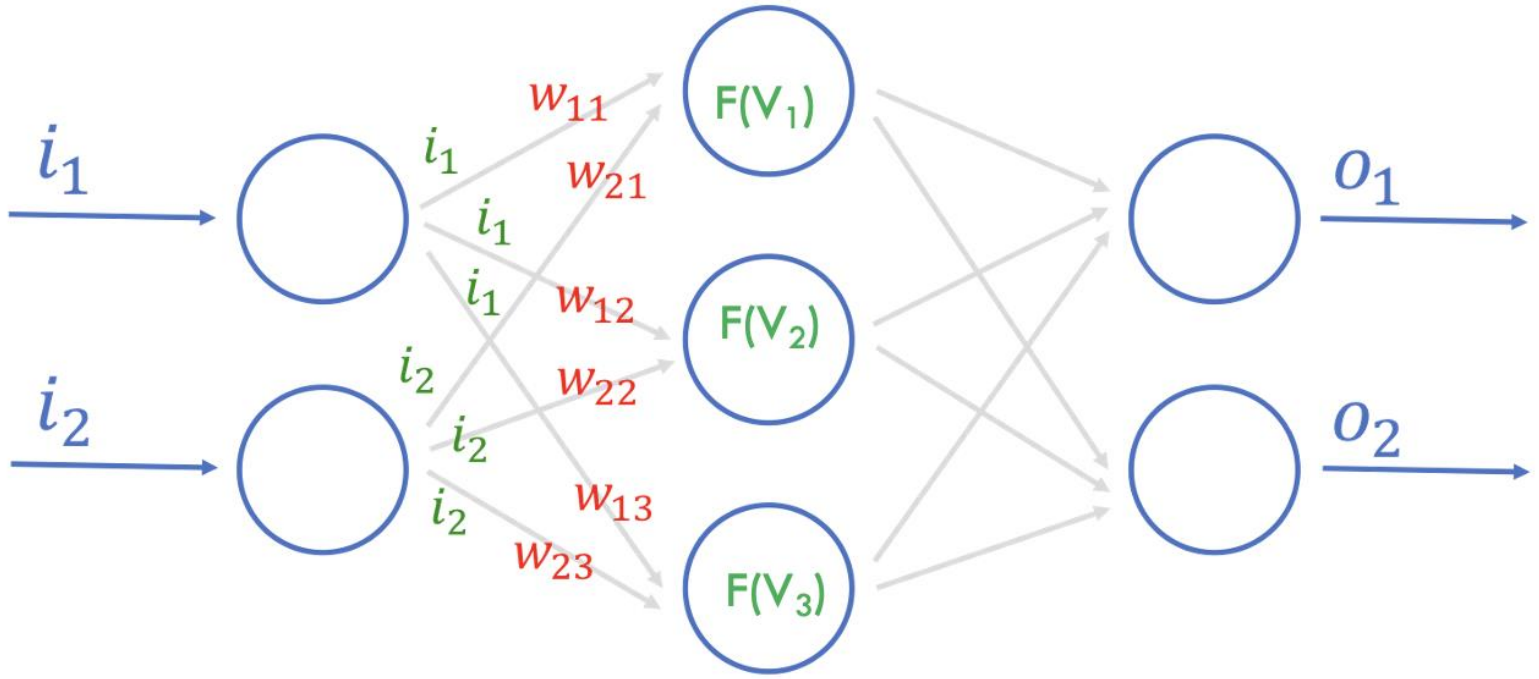
NEURONES À N ENTRÉES

- Même concept ! En général :
- Étant donné N entrées :
- 1) Une fonction linéaire de R^n à R^1 (dont le graphe serait un hyperplan de dimension R^n dans un espace de dimension R^{n+1}).
- 2) Le zéro dans cette dimension R^{n+1} est aussi un hyperplan de dimension R^n – il est le seuil, qui divise la surface de décision en deux: intuitivement, se décharger ou ne se décharger pas.
- 3) Une function non-linéare est appliqué ensuite, qui «répresente» le fait de se décharger ou non

DES NEURONES AUX COUCHES

- En fin de compte, un neurone fait correspondre R^n à R^1
- Une couche de neurones est une transformation aussi : si vous avez deux neurones, vous avez deux correspondances de R^n à R^1…
- … de manière équivalente, une correspondance de R^n à R^2.
- Si vous avez cinq neurones (qui reçoivent des données de trois neurones), vous avez une correspondance de R^3 à R^5.

$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix} \begin{matrix} = V_1 \\ = V_2 \\ = V_3 \end{matrix}$$



DES NEURONES AUX COUCHES

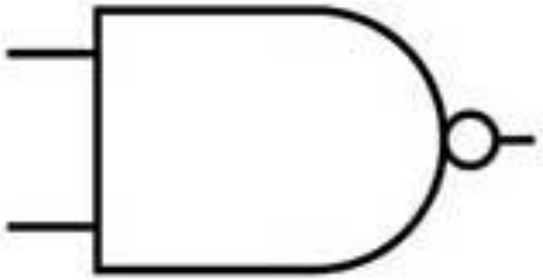
- Si nous ne considérons que la partie linéaire, il s'agit de transformations linéaires d'espaces vectoriels (déterminées par la façon dont elles mettent en correspondance les vecteurs de base, par exemple 0,1 et 1,0).

DES NEURONES AUX COUCHES

- Ensuite, nous autorisons les non-linéarités, ce qui permet aux mappings d'être n'importe quoi

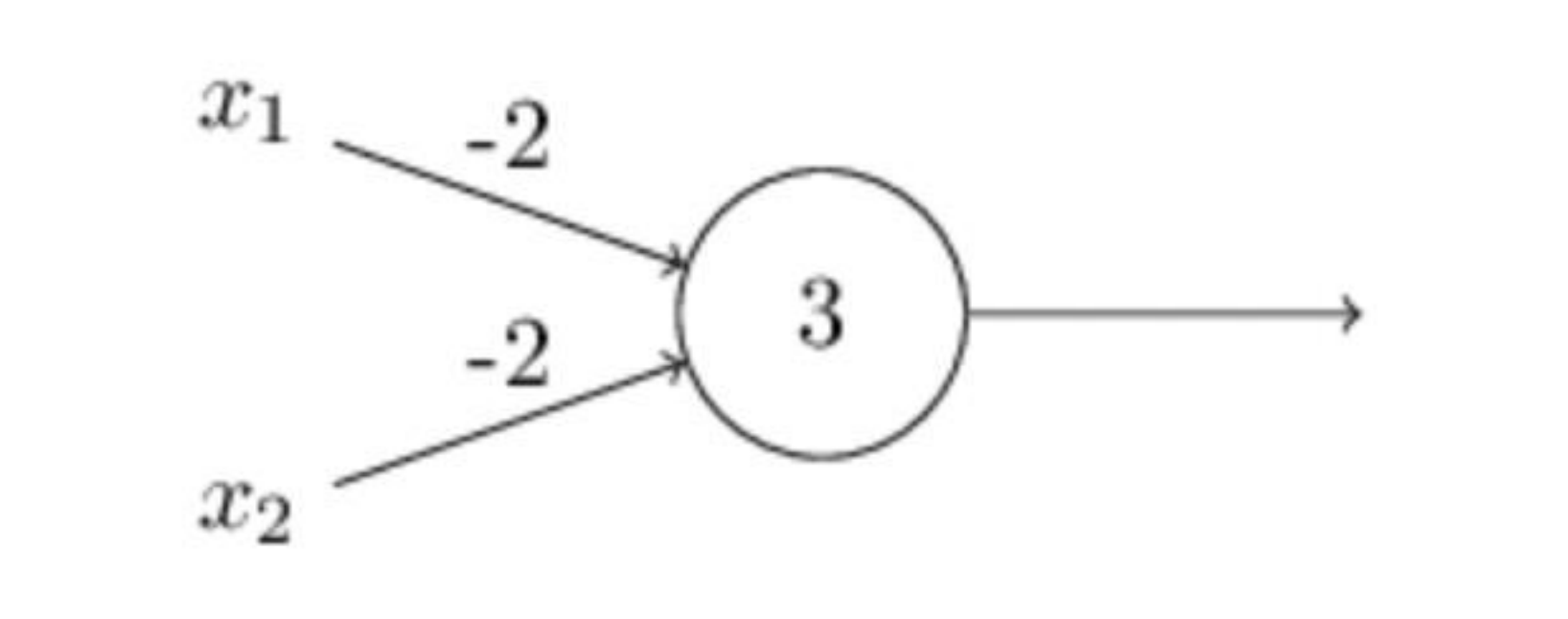
LES PERCEPTRONS EN TANT QU'IMPLÉMENTATIONS DE SYSTÈMES DE SYMBOLES (PORTES NAND)

NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

PERCEPTRONS EN TANT QU'IMPLEMENTATIONS



PERCEPTRONS EN TANT QU'IMPLEMENTATIONS

- Les poids de ce perceptron sont -2 et -2. Son biais est +3 .... en d'autres termes, son seuil est -3.

PERCEPTRONS EN TANT QU'IMPLEMENTATIONS

- Nous voyons alors que l'entrée 00 produit la sortie 1, puisque $(-2*0)+(-2*0)+3=3$ est positif.
- *Autrement dit :* $(-2*0)+(-2*0)=0$ est supérieur au seuil de -3, donc il se déclenche.
- Des calculs similaires montrent que les entrées 01 et 10 produisent la sortie 1.
- Mais l'entrée 11 produit la sortie 0, puisque $(-2*1)+(-2*1)+3=-1$ est négatif.
- Et donc notre perceptron implémente une porte NAND ! (NAND = tout sauf les deux)

PERCEPTRONS EN TANT QU'IMPLEMENTATIONS

- On peut montrer que toute porte logique peut être construite à l'aide de circuits NAND, donc les perceptrons peuvent implémenter des machines de Turing. (Cf. les affirmations de Fodor et Pylyshyn selon lesquelles ils ne sont peut-être bons qu'à cela)

PERCEPTRONS EN TANT QU'IMPLEMENTATIONS

- MAIS: Cependant, les réseaux neuronaux sont bons pour bien plus que cela (bien que les perceptrons ne le soient peut-être pas). Le secret réside dans la fonction d'activation.
- En passant d'une simple fonction d'activation à seuil à une fonction différentiable (c'est-à-dire qui ne saute pas de un à zéro), nous pouvons permettre l'apprentissage.

DES RÉSEAUX NEURONAUX CAPABLES D'APPRENDRE

DES RÉSEAUX NEURONAUX CAPABLES D'APPRENDRE

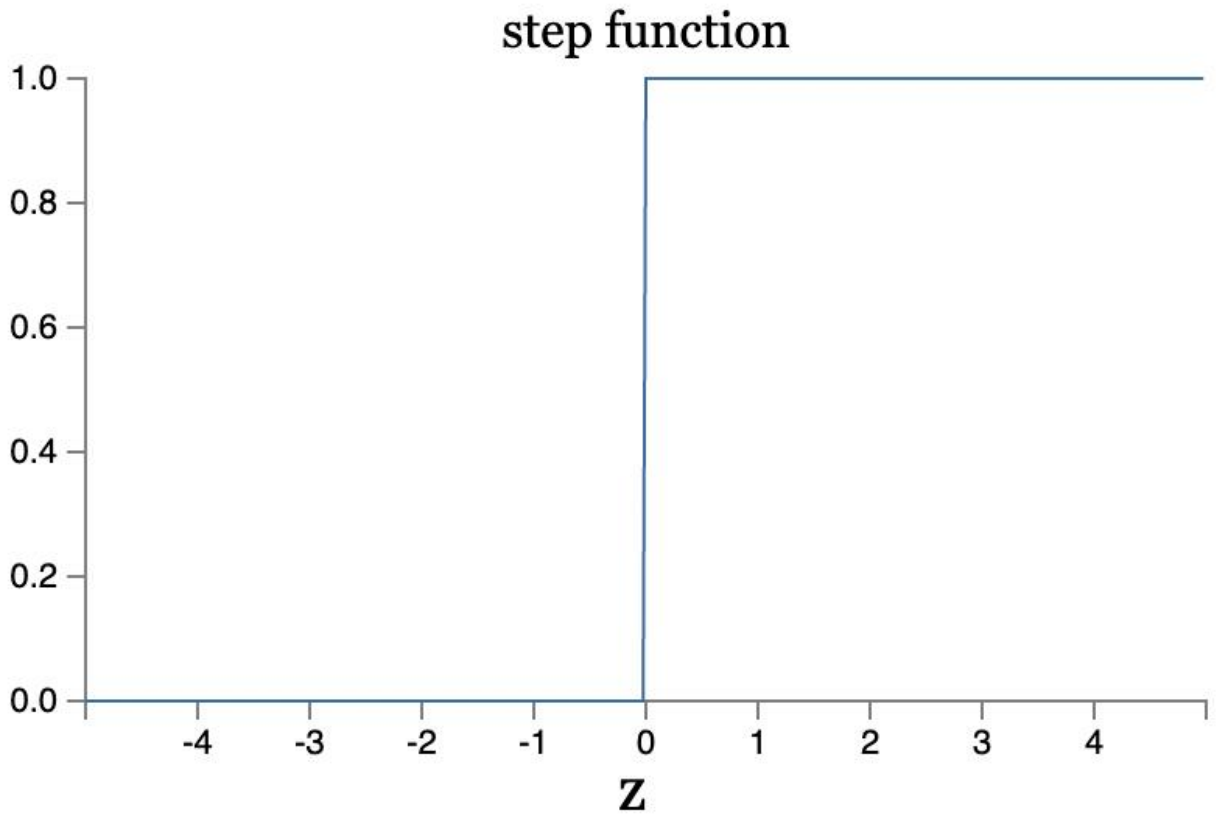
- Que serait l'apprentissage d'un réseau neuronal ? Disons que nous avons une certaine sortie cible, les sorties que nous voulons que le système fournisse.

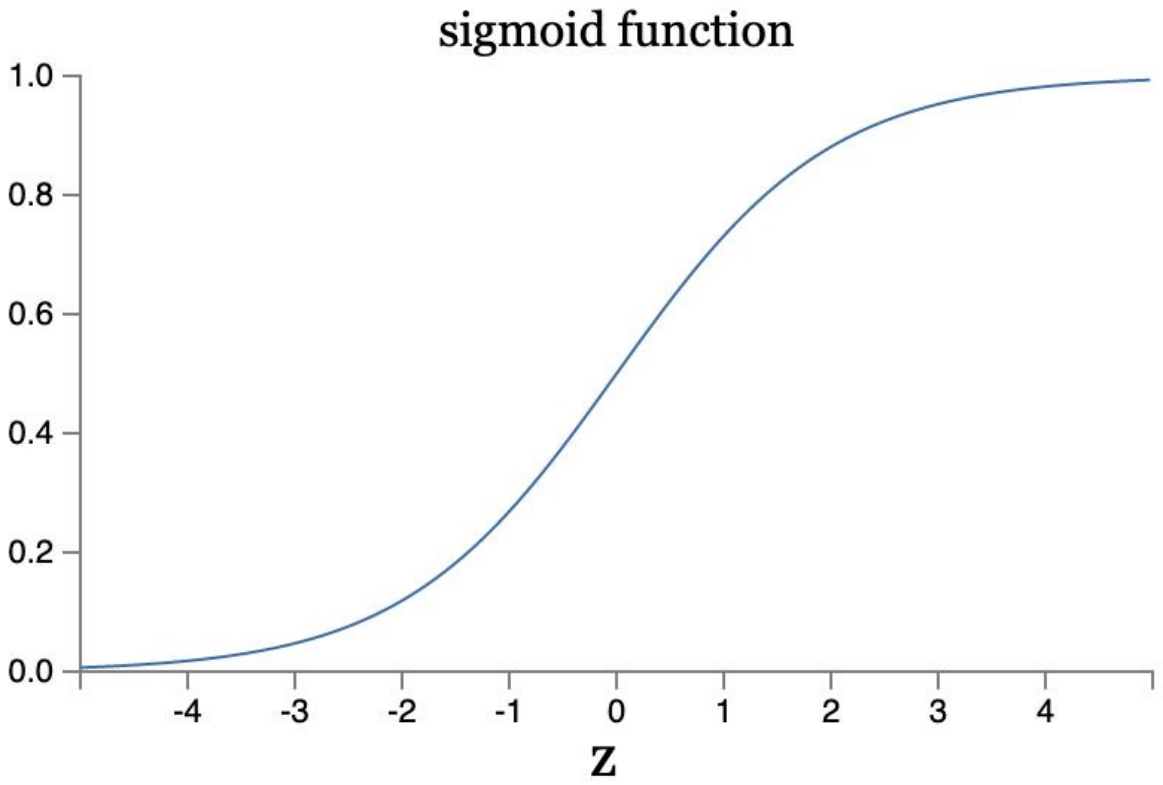
- L'apprentissage serait alors une méthode d'ajustement des poids et des biais du réseau (progressivement), jusqu'à ce que vous obteniez le résultat souhaité.

DES RÉSEAUX NEURONAUX CAPABLES D'APPRENDRE

- Le problème avec les perceptrons est que, comme ils ont des activations à seuil, de petites modifications de l'entrée peuvent produire de très grands changements dans la sortie.

- La solution : au lieu d'une véritable activation à seuil (fonction échelon), une fonction continue qui ne fait qu'approximer une fonction échelon (la fonction sigmoïde)


step function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
z


sigmoid function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
Z

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}. \tag{3}$$

To put it all a little more explicitly, the output of a sigmoid neuron with inputs $x_1, x_2, \ldots$, weights $w_1, w_2, \ldots$, and bias b is

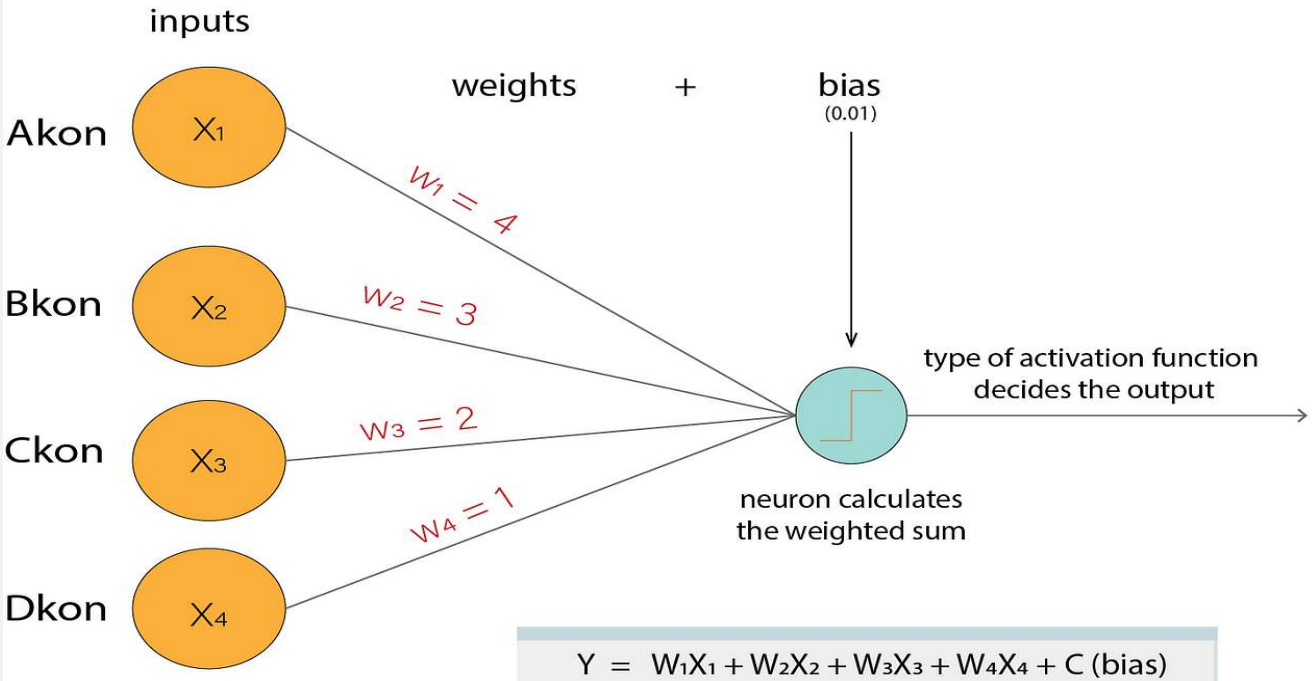
$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}. \tag{4}$$

DES RÉSEAUX NEURONAUX CAPABLES D'APPRENDRE

- En utilisant une telle fonction, nous pouvons nous assurer que des changements suffisamment petits dans les poids et les biais produiront des changements suffisamment petits dans la sortie, ce qui signifie que si nous pouvons trouver une méthode pour déterminer de quelles manières « pousser » les poids et les biais de notre réseau, nous pourrions lui faire apprendre une fonction de sortie souhaitée...

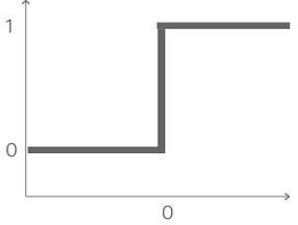
DES RÉSEAUX NEURONAUX CAPABLES D'APPRENDRE

- À noter:
- Il existe de nombreuses autres fonctions d'activation (différentiables) …



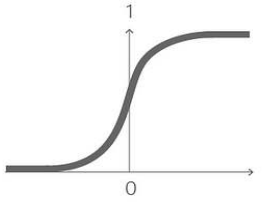
$$Y = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + C \text{ (bias)}$$

Thailand	(4x-9) + (3x10) + (2x3) + (1x-2)	=	- 2
Jamaica	(4x-4) + (3x5) + (2x-1) + (1x8)	=	5
Dubai	(4x-3) + (3x6) + (2x-2) + (1x6)	=	8



Step

Thailand: 0
Jamaica: 1
Dubai: 1



Sigmoid
(Probabilities)

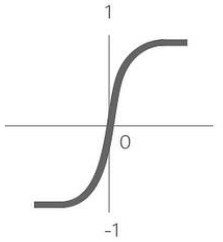
Thailand: 0.002
Jamaica: 0.85
Dubai: 0.94

*probabilities do not add to 1

Softmax
(Probabilities)

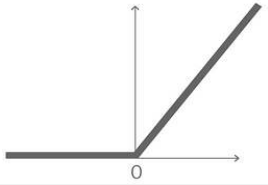
Thailand: 0.02
Jamaica: 0.22
Dubai: 0.76

*probabilities add to 1



tanh

Thailand: -0.2
Jamaica: 0.5
Dubai: 0.8

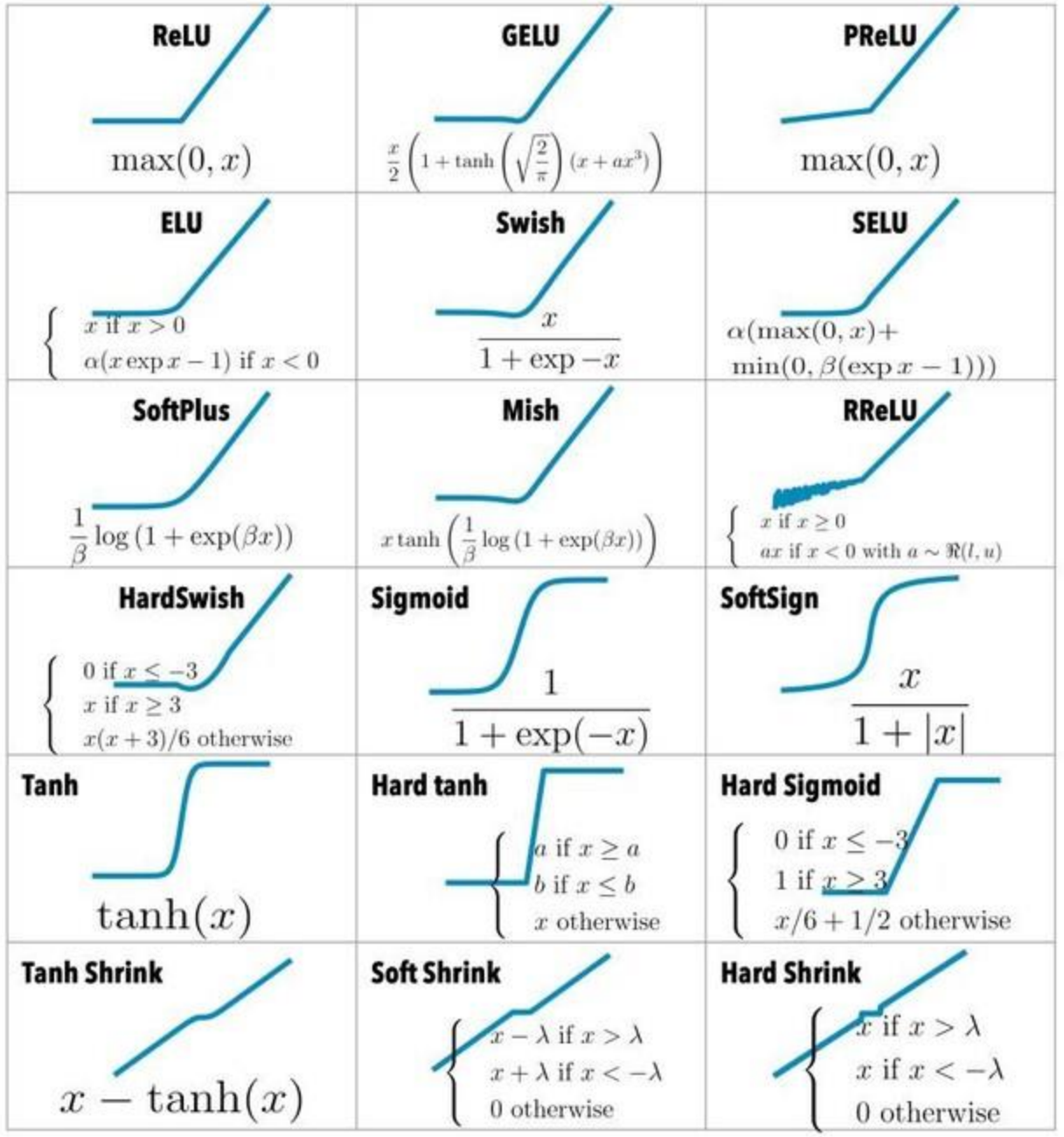


ReLU

Thailand: 0
Jamaica: 5
Dubai: 8

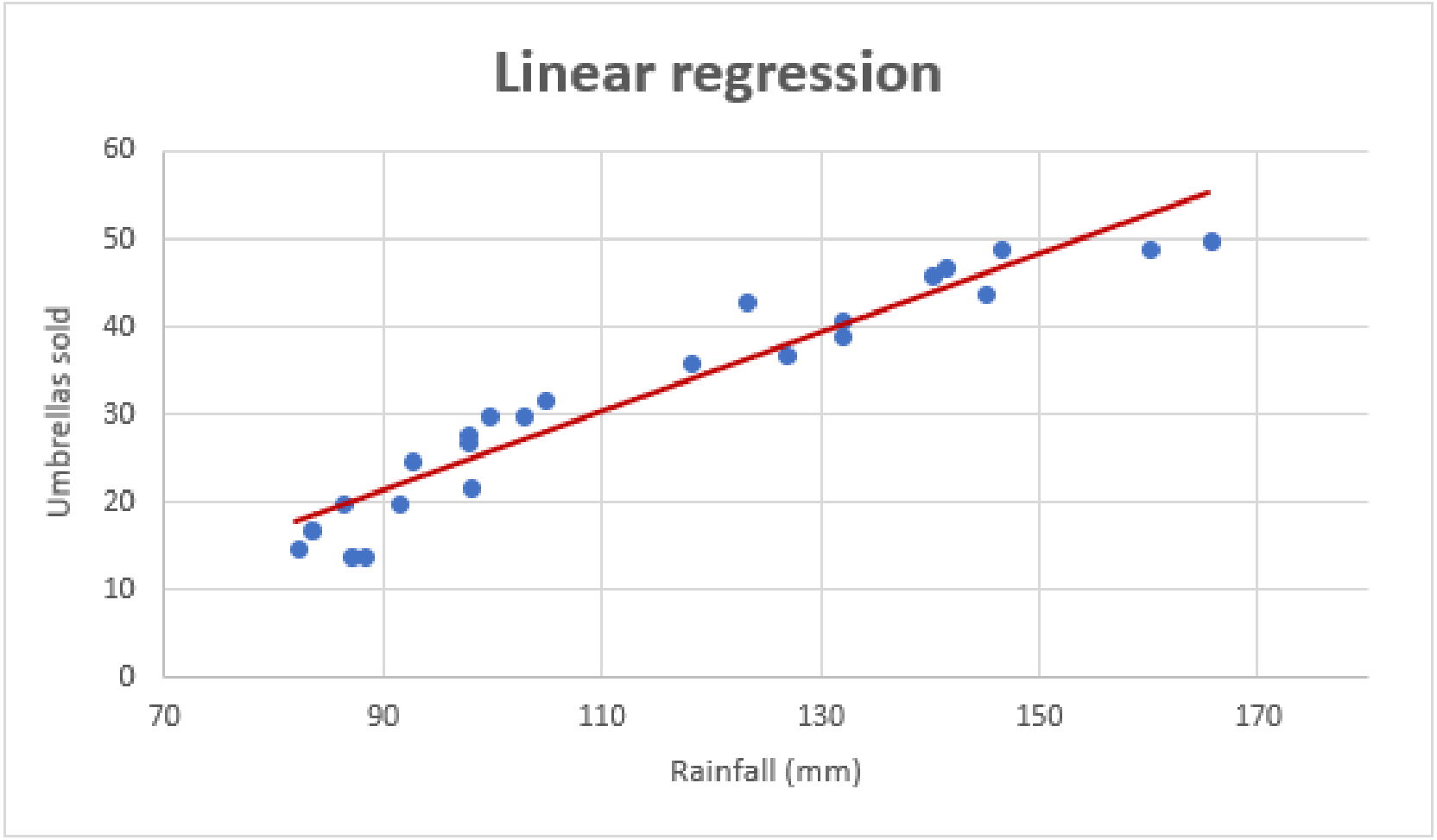
https://towardsdatascience.com/using-activation-functions-in-neural-nets-c119ad80826

Neural Network Activation Functions: a small subset!



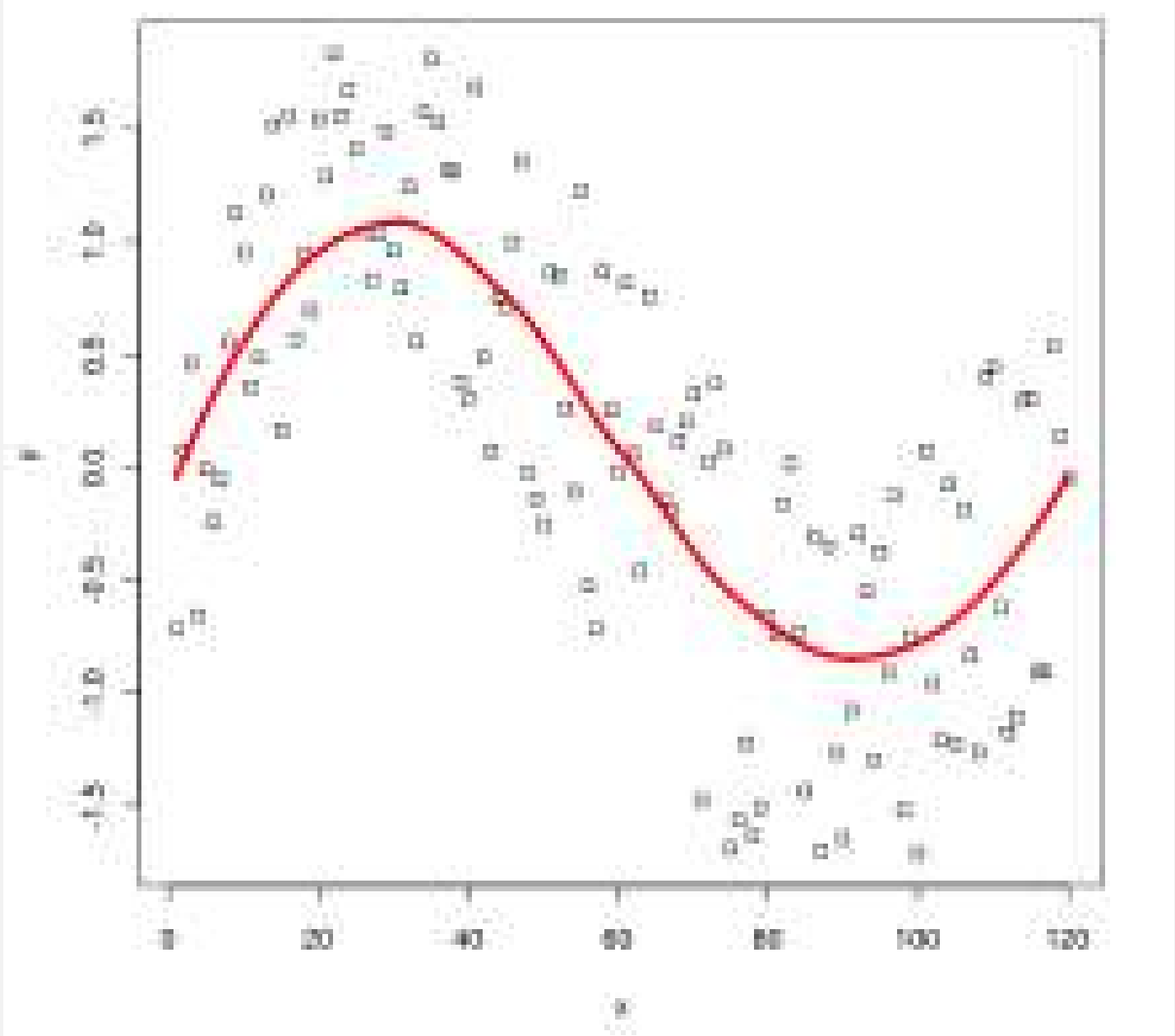
Rahul Jayarwardana 2021

L'APPRENTISSAGE DES FONCTIONS COMME L'APPRENTISSAGE DE MODÈLES PRÉDICTIFS DU MONDE


Linear regression
Umbrellas sold
60
50
40
30
20
10
0
70
90
110
130
150
170
Rainfall (mm)

MODÈLES PRÉDICTIFS

- Erreur : la différence entre la prédiction et la valeur réelle
- Un bon modèle prédictif est une ligne ou une courbe (un hyperplan) qui minimise l'erreur (étant donné les degrés de liberté dont il dispose)


1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
y
0
20
40
60
80
100
120
x


Linear Regression Curve
Daily Chart - E-mini S&P 500 Future (ES)
Sell when Price deviates above
Linear Regression Curve
Sell
Buy
Sell
Buy
9-day Moving
Linear Regression
Buy when Price deviates below
Linear Regression Curve
Buy
Commodity.com - all rights reserved

MODÈLES PRÉDICTIFS

- Notez la possibilité d'un surajustement et d'un sous-ajustement : vous pouvez relier tous les points directement si vous avez suffisamment de paramètres ... mais cela ne fera pas une bonne projection en dehors de la distribution (surajustement / overfitting).

MODÈLES PRÉDICTIFS

- D'autre part, il est parfois évident que vous avez besoin d'une courbe ou d'une fonction périodique plutôt que d'une ligne droite, là, une ligne droite serait sous-adaptée. (c'est un art sombre que de savoir comment l'obtenir juste comme il faut cependant)

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Nous avons noté que les réseaux neuronaux pourraient être en mesure d'apprendre certaines fonctions, et qu'il existe certaines fonctions qu'ils pourraient apprendre. Il reste à établir :

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Il reste à établir:
- a) que les réseaux neuronaux sont capables de représenter toute fonction qu'il serait souhaitable de représenter
- b) qu'il existe une méthode générale d'apprentissage qui pourrait permettre à un réseau d'apprendre une telle fonction

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Le théorème d'approximation universelle : pour toute fonction continue F, pour tout degré d'approximation, un réseau neuronal peut approximer F à ce degré.

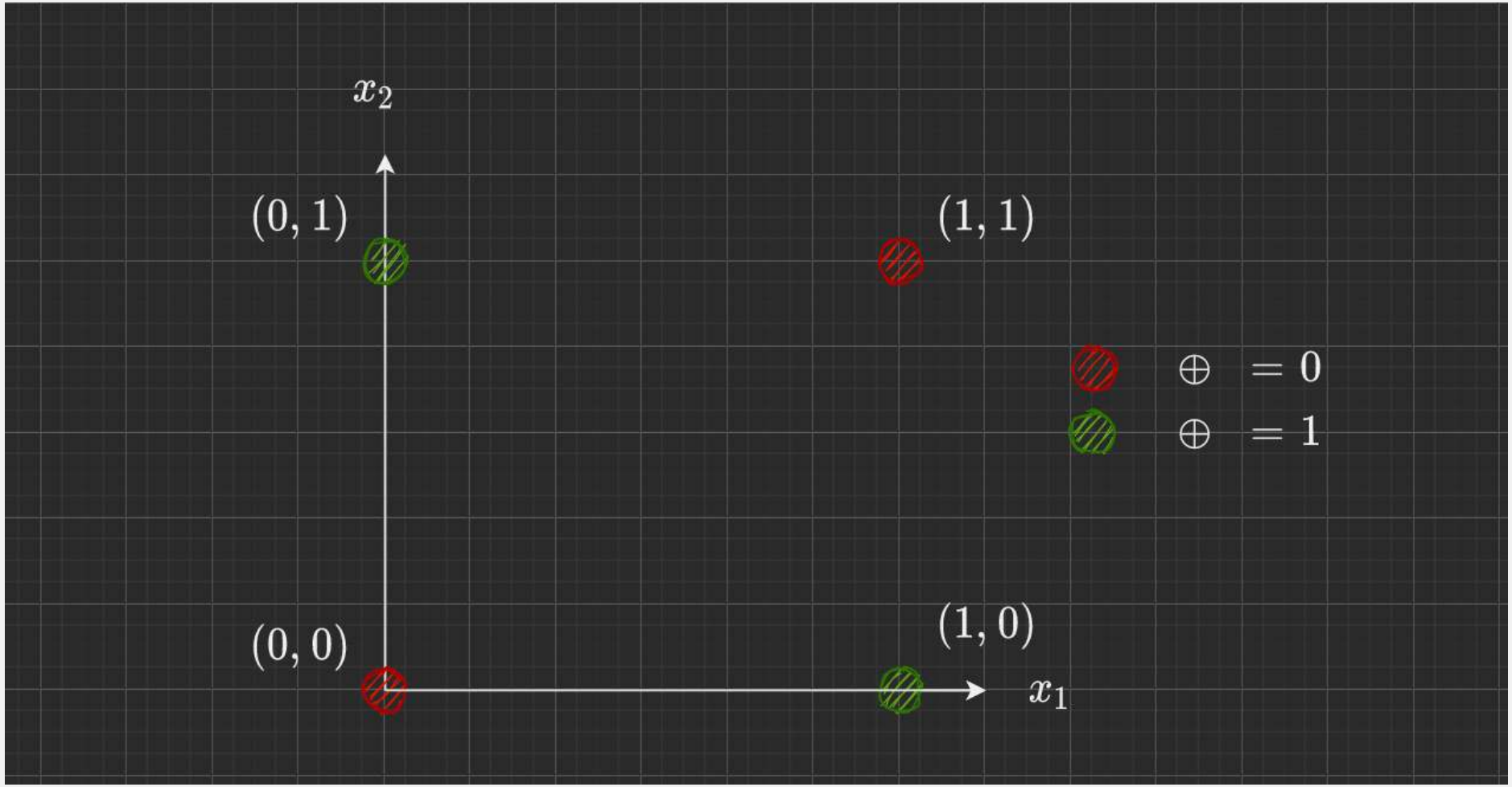
LE THÉORÈME D'APPROXIMATION UNIVERSELLE

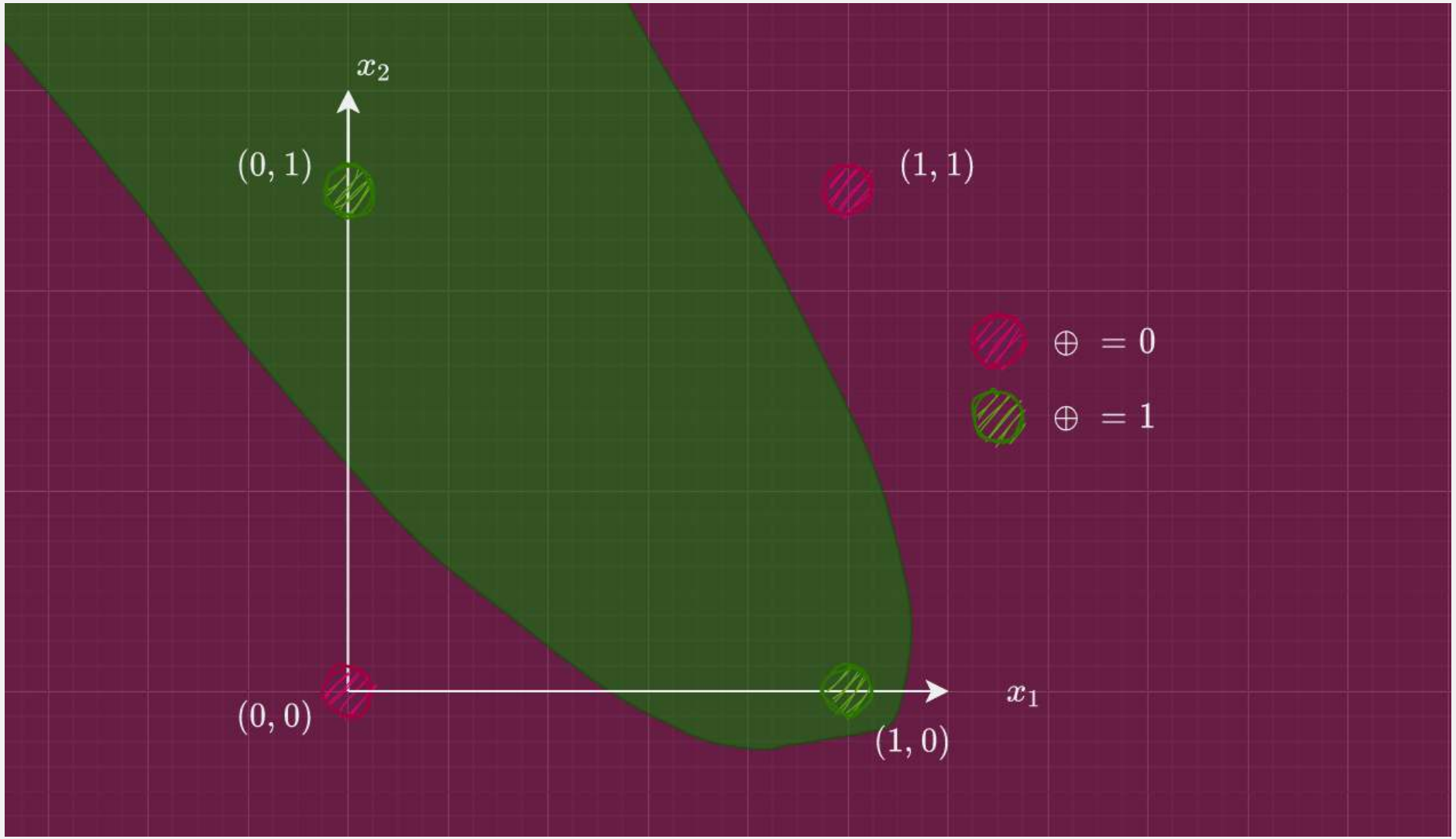
- Comment ? Pas possible avec un seul perceptron

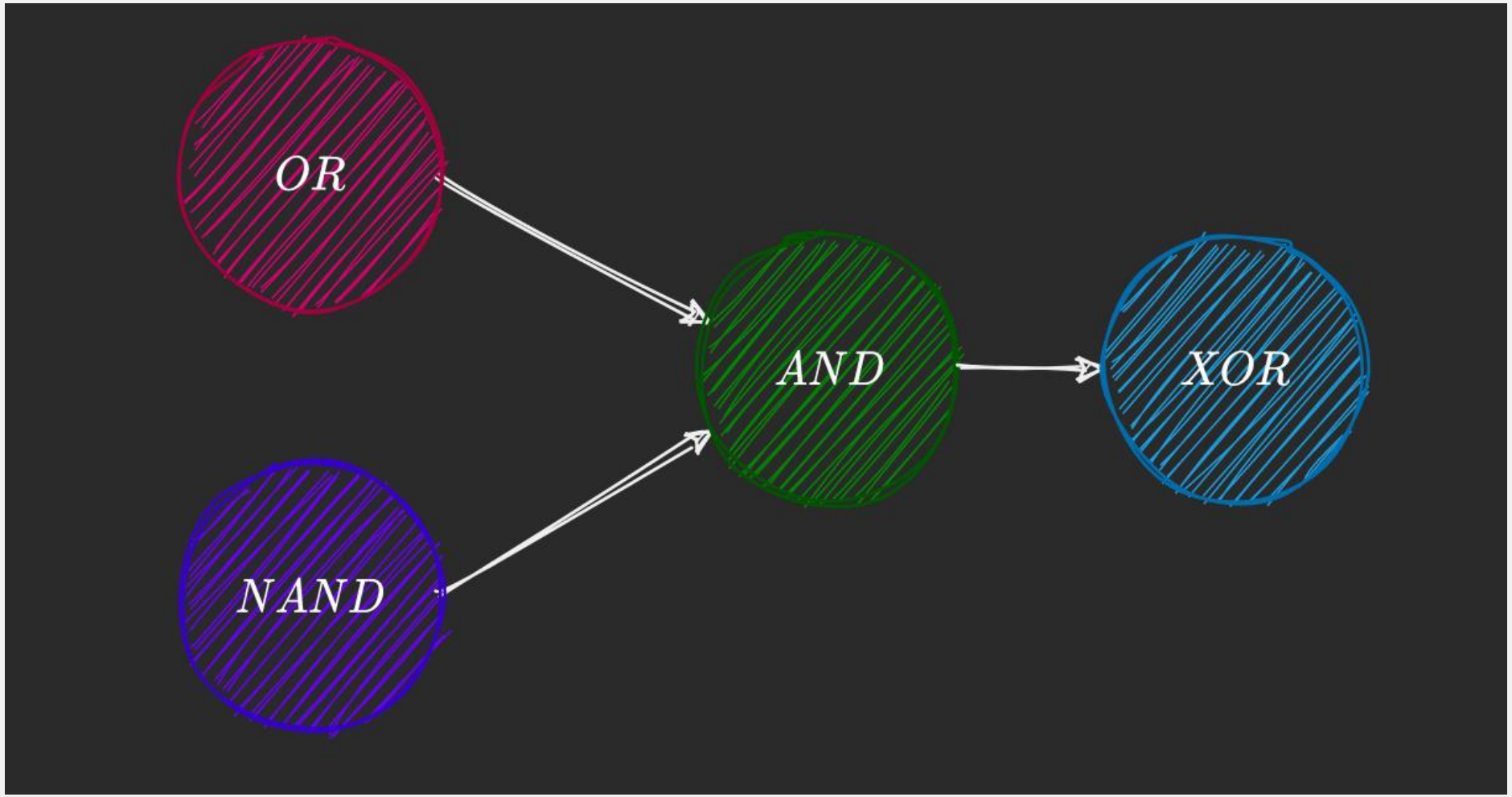
PROFONDEUR

- Exemple : XOR (la fonction de vérité «*ou* exclusif» - vrai si exactement un des deux disjonctifs est vrai)

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

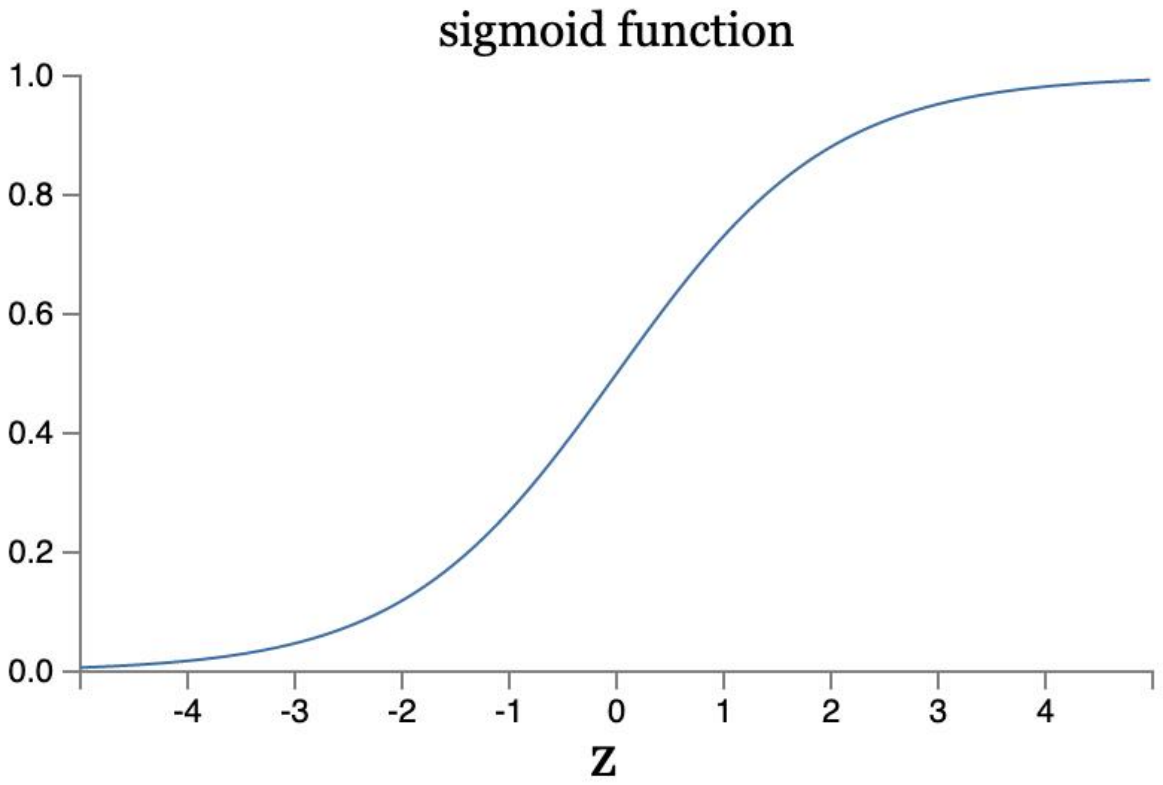

x_2
(0, 1)
(1, 1)
$\oplus = 0$
$\oplus = 1$
(0, 0)
(1, 0)
x_1


x_2
$(0, 1)$
$(1, 1)$
$\oplus = 0$
$\oplus = 1$
$(0, 0)$
$(1, 0)$
x_1


OR
AND
XOR
NAND

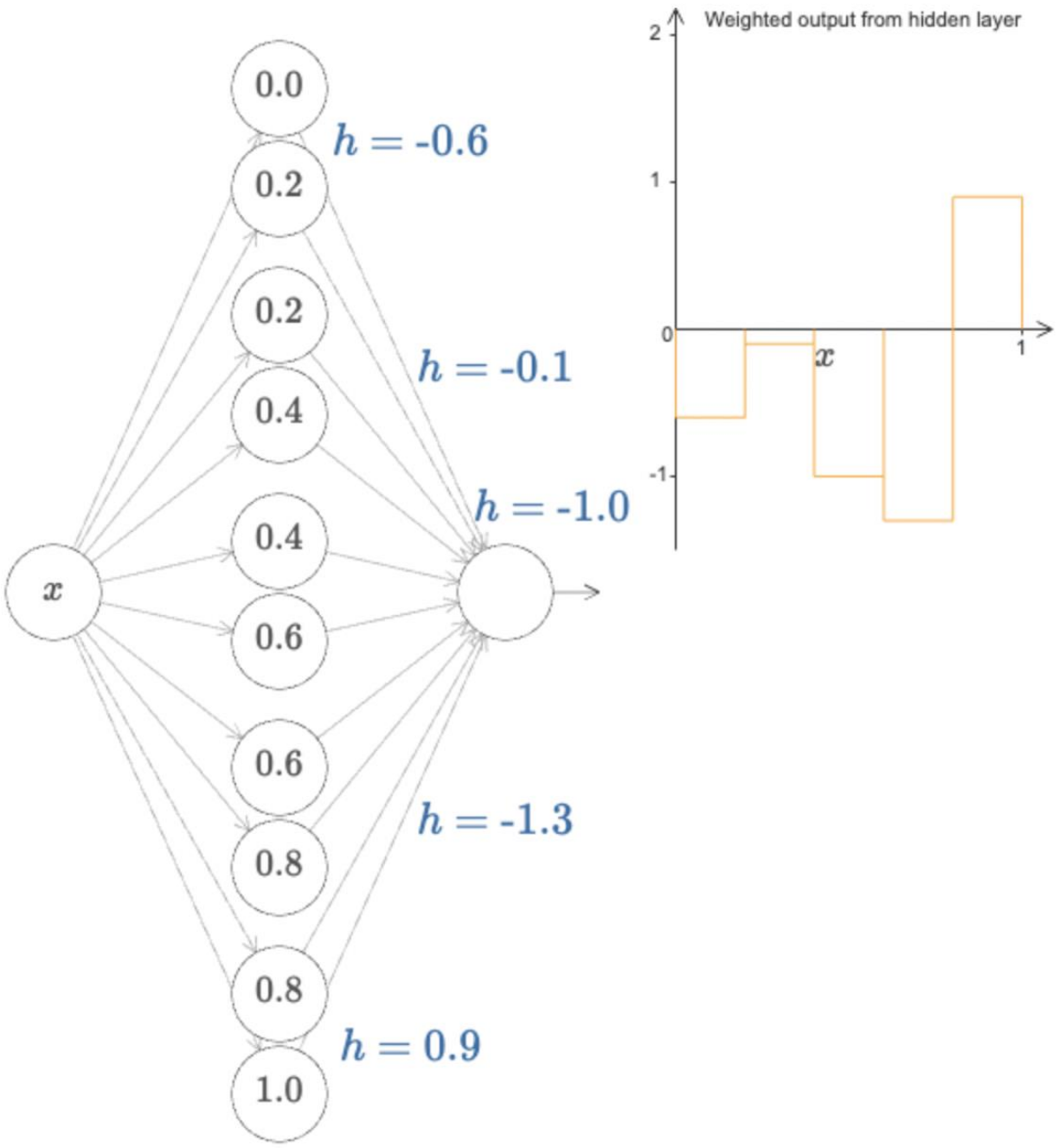
LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Comment ? Intuitivement, chaque neurone vous donne une courbe :


sigmoid function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
Z

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Vous pouvez ajuster la forme exacte de la courbe et l'endroit exact où elle se situe sur les axes x-y en ajustant les pondérations et le biais.
- Deux neurones, et vous avez deux courbes. Pour approximer une fonction arbitrairement courbe, vous avez juste besoin de suffisamment de neurones


0.0
0.2
0.2
0.4
0.4
0.6
0.6
0.8
0.8
1.0
x
h = -0.6
h = -0.1
h = -1.0
h = -1.3
h = 0.9
Weighted output from hidden layer
2
1
0
-1
1
x

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Pour en savoir plus sur ce fonctionnement, consultez le chapitre :
- http://neuralnetworksanddeeplearning.com/chap4.html
- (les outils graphiques interactifs illustrent le propos bien mieux que je ne pourrais le faire)

LE THÉORÈME D'APPROXIMATION UNIVERSELLE

- Note : ce théorème s'applique même pour le cas d'un réseau avec une seule couche cachée avec beaucoup de neurones dans celle-ci entre les entrées et les sorties

DESCENTE DE GRADIENT PAR BACKPROPAGATION

DESCENTE DE GRADIENT PAR BACKPROPAGATION

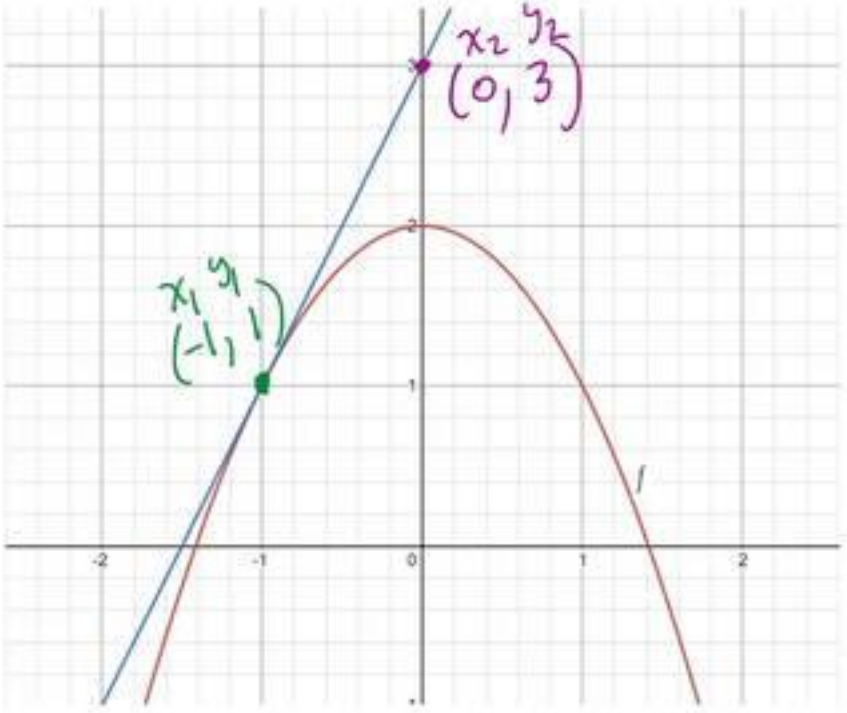
- J'ai mentionné qu'avec la régression linéaire, vous trouvez la ligne qui minimise l'erreur.

- Là, cela peut être fait analytiquement : vous pouvez en fait résoudre cette ligne (par exemple par la méthode des moindres carrés)

DESCENTE DE GRADIENT ET BACKPROPAGATION

- Dans les cas plus complexes, vous ne pouvez pas le faire de manière analytique.
- Cependant, vous pouvez identifier une « direction » (de la pente / du gradient) dans laquelle chaque paramètre doit être légèrement modifié afin de réduire l'erreur.
- (l'approche, appelée rétropropagation / backpropagation, utilise des outils issus du calcul : la règle de la chaîne et les dérivées partielles, pour déterminer la « direction » dans laquelle l'erreur d'une fonction change lorsque vous la modifiez)

Ex. 1: Given the graph of f and its tangent line, estimate $f'(-1)$.



$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$f'(-1) = m$

$m = \frac{3 - 1}{0 - -1} = \frac{2}{1}$

$m = 2$

$f'(-1) = 2$

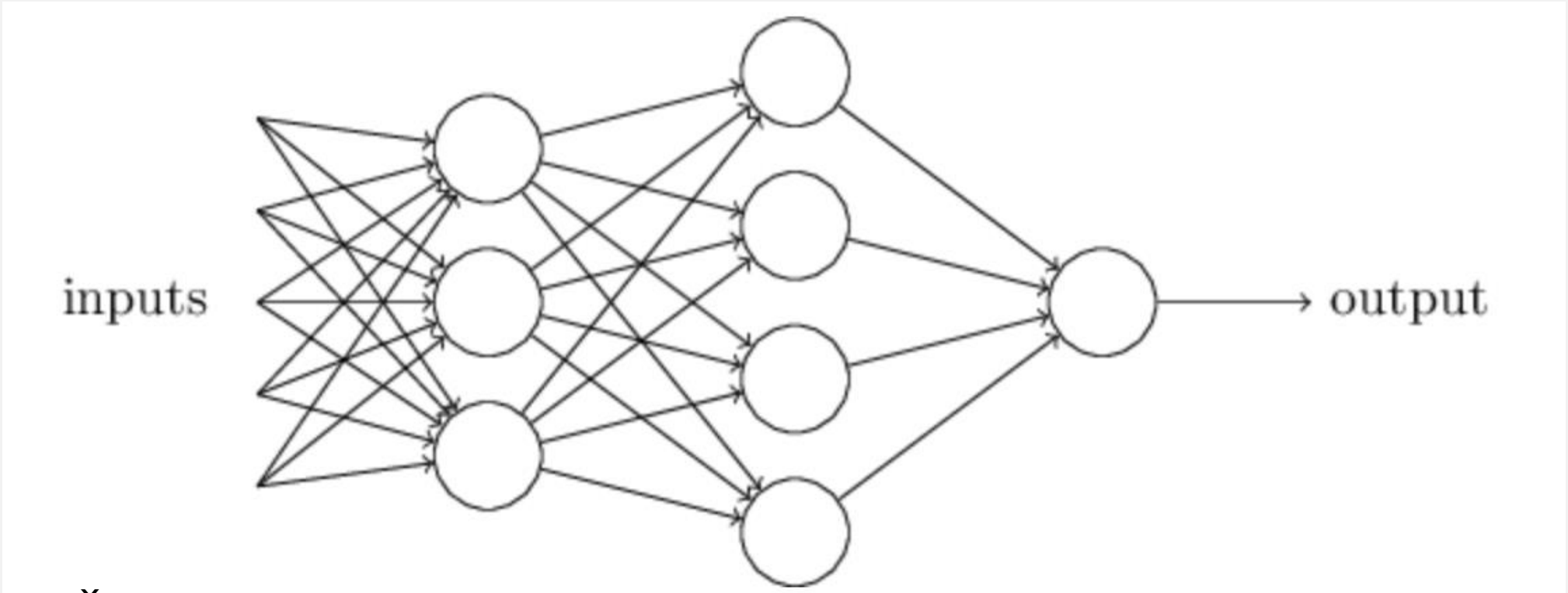
©Study.com

DESCENTE DE GRADIENT ET BACKPROPAGATION

- … comme ça, mais ici en général c'est pour une fonction compliquée, multivariée, la fonction de coût de la fonction du réseau neuronal.

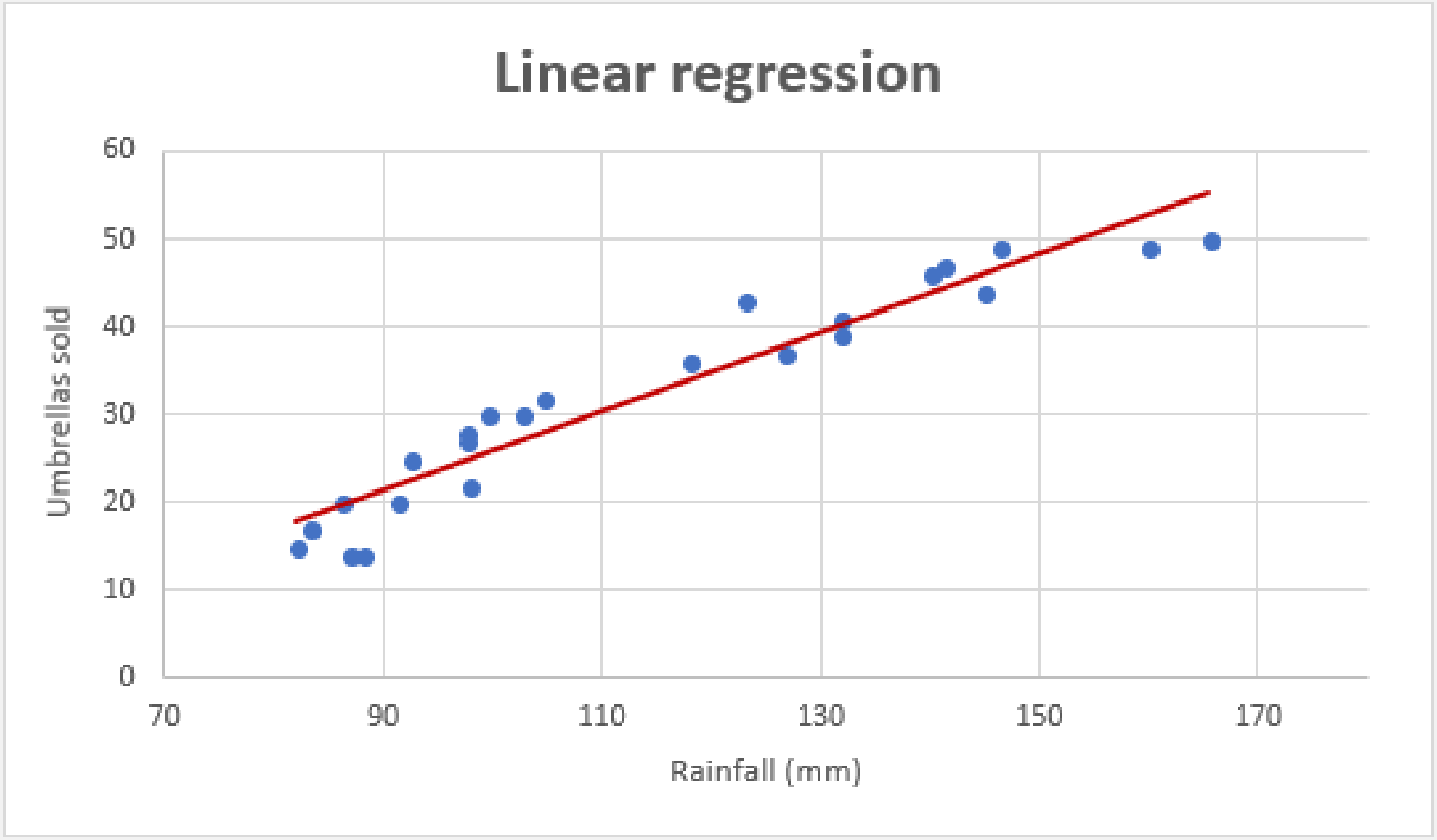
DESCENTE DE GRADIENT ET BACKPROPAGATION

- la fonction du réseau neuronal = F(G(H(x)))


inputs
output
x
H(X)
G(H(x)
F(G(H(x)))

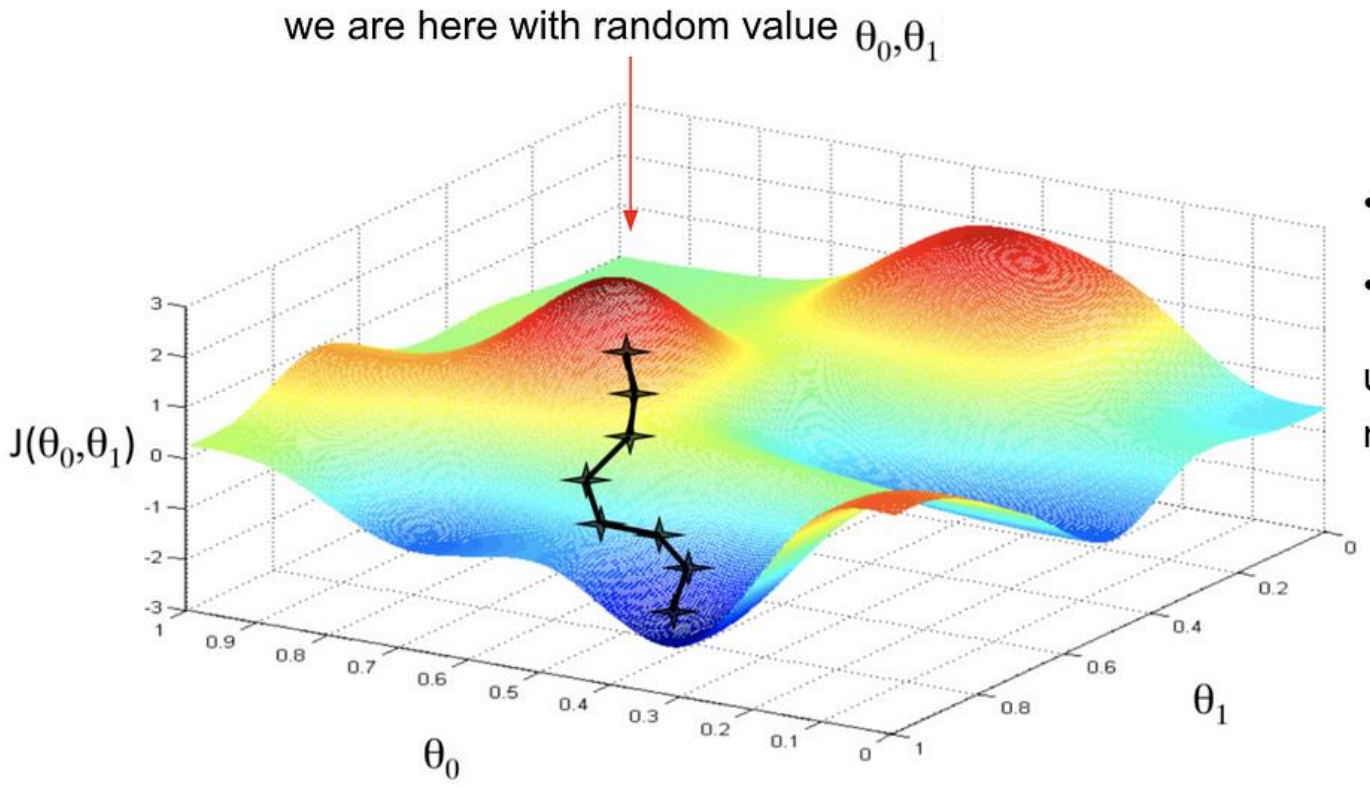
DESCENTE DE GRADIENT ET BACKPROPAGATION

- la fonction du réseau neuronal = F(G(H(x)))
- Sa fonction de coût C mesure la distance qui le sépare de la bonne réponse, pour chaque entrée.


Linear regression
Umbrellas sold
60
50
40
30
20
10
0
70
90
110
130
150
170
Rainfall (mm)

DESCENTE DE GRADIENT ET BACKPROPAGATION

- la fonction du réseau neuronal = F(G(H(x)))
- Sa fonction de coût C mesure la distance qui le sépare de la bonne réponse, pour chaque entrée.
- L'idée est de prendre la dérivée, ou C = cout, de
- C (F(G(H(x)))) :
- (comment le cout change quand on varie les entrées)

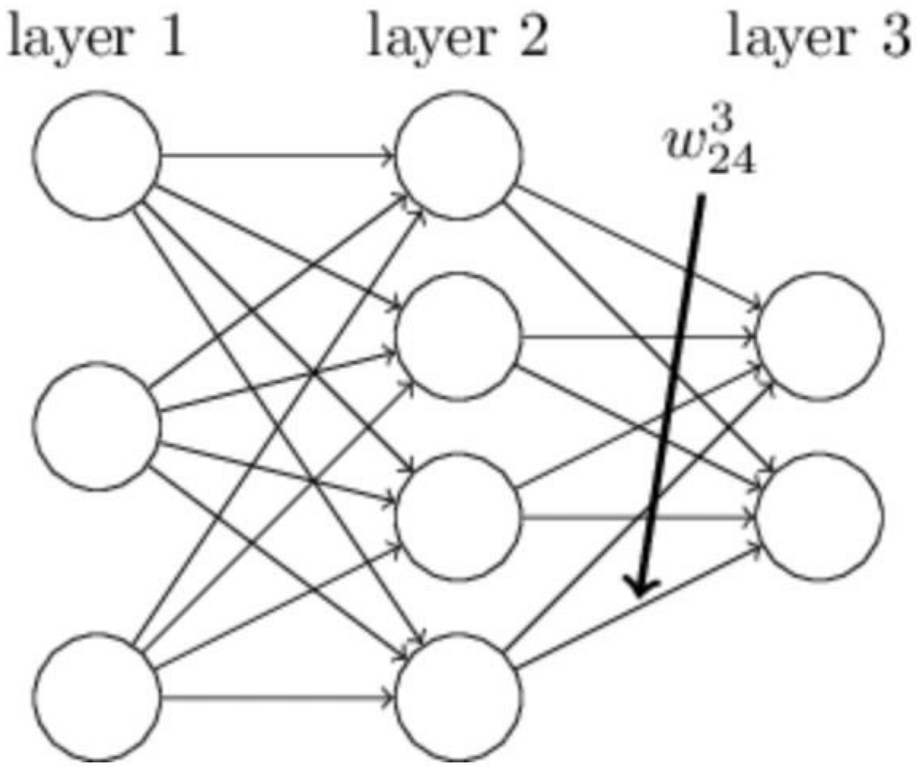


- Start with some θ_0,θ_1
- Keep changing θ_0,θ_1 to reduce $J(\theta_0,\theta_1)$

until we hopefully end up at a minimum

DESCENTE DE GRADIENT ET BACKPROPAGATION

- la fonction du réseau neuronal = F(G(H(x)))
- Sa fonction de coût C mesure la distance qui le sépare de la bonne réponse, pour chaque entrée.
- L'idée est de prendre la dérivée de
- C (F(G(H(x)))) :
- et ensuite ajuster contre la direction de cette dérivée (la direction ou la pente tombe / le gradient descend), c'est-à-dire dans la direction qui pointe vers le bas vers l'erreur minimale.



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer