Il faut mettre le lien avec la svt en premier je pense.

Les schéma sont dans deep learning → schemas etc

Explication mathématique :

J’ai utilisé ce site <https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba> ainsi que des connaissances perso.

Derrière les IA se cachent énormément de mathématiques.  
Ces IA effectuent un très grand nombre de calculs basiques ; ce qui amène a une très grande quantité de nombres et de variables.

Une intelligence artificielle est constituée :

* d’un ensemble de neurones connectés entre eux
* ainsi qu’un jeu de coefficients réglés lors des phases d’apprentissage.

Lors de la phase d’apprentissage, on fait rentrer dans le réseau une très grande série de données connues (d’où l’appellation de deep learning ou apprentissage profond), et on règle les coefficients afin d’obtenir la bonne sortie,

Pour notre projet nous avons choisi de reconnaitre des chiffres entre 0 et 9, représentés par une image de 28\*28 pixels.

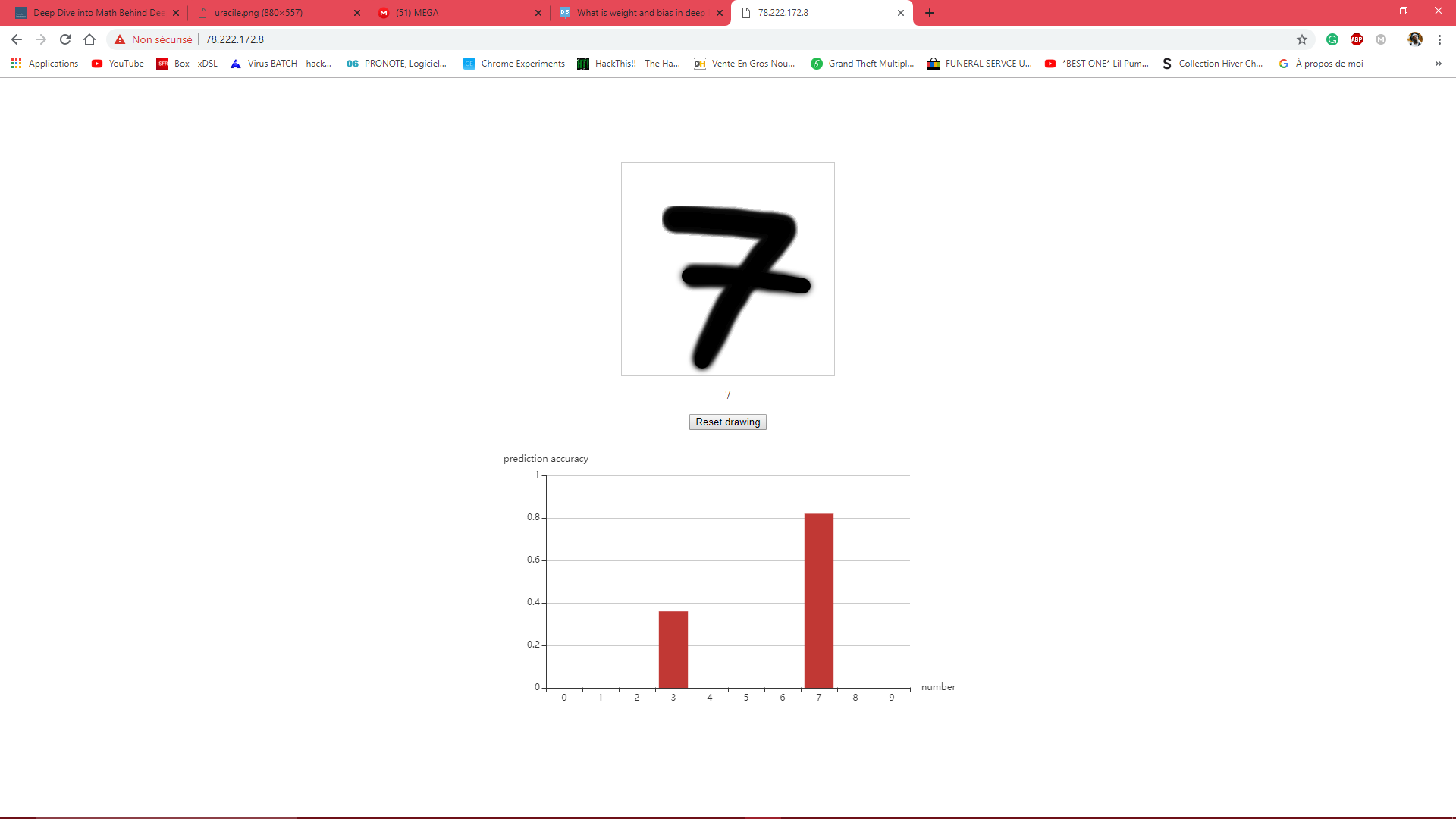
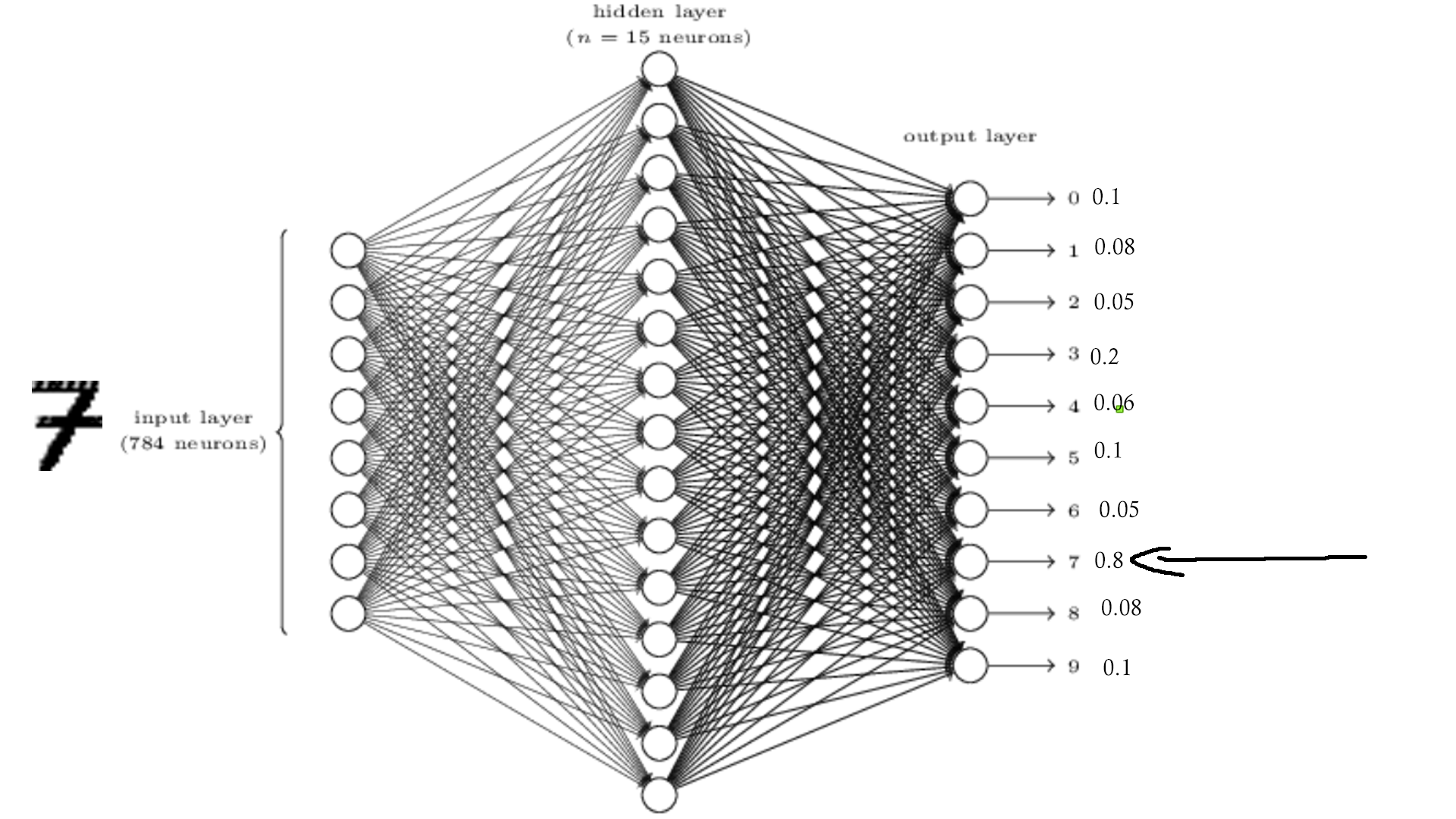


roach.png

Chaque image a donc un total de 28\*28=784 pixels.  
Les pixels sont séparés les uns des autres et sont positionnés dans des « entrées ».  
1 pixel = 1 entrées.

Expliquer les 10 sorties avec une proba

Izebi.png boumboumtamtam.png

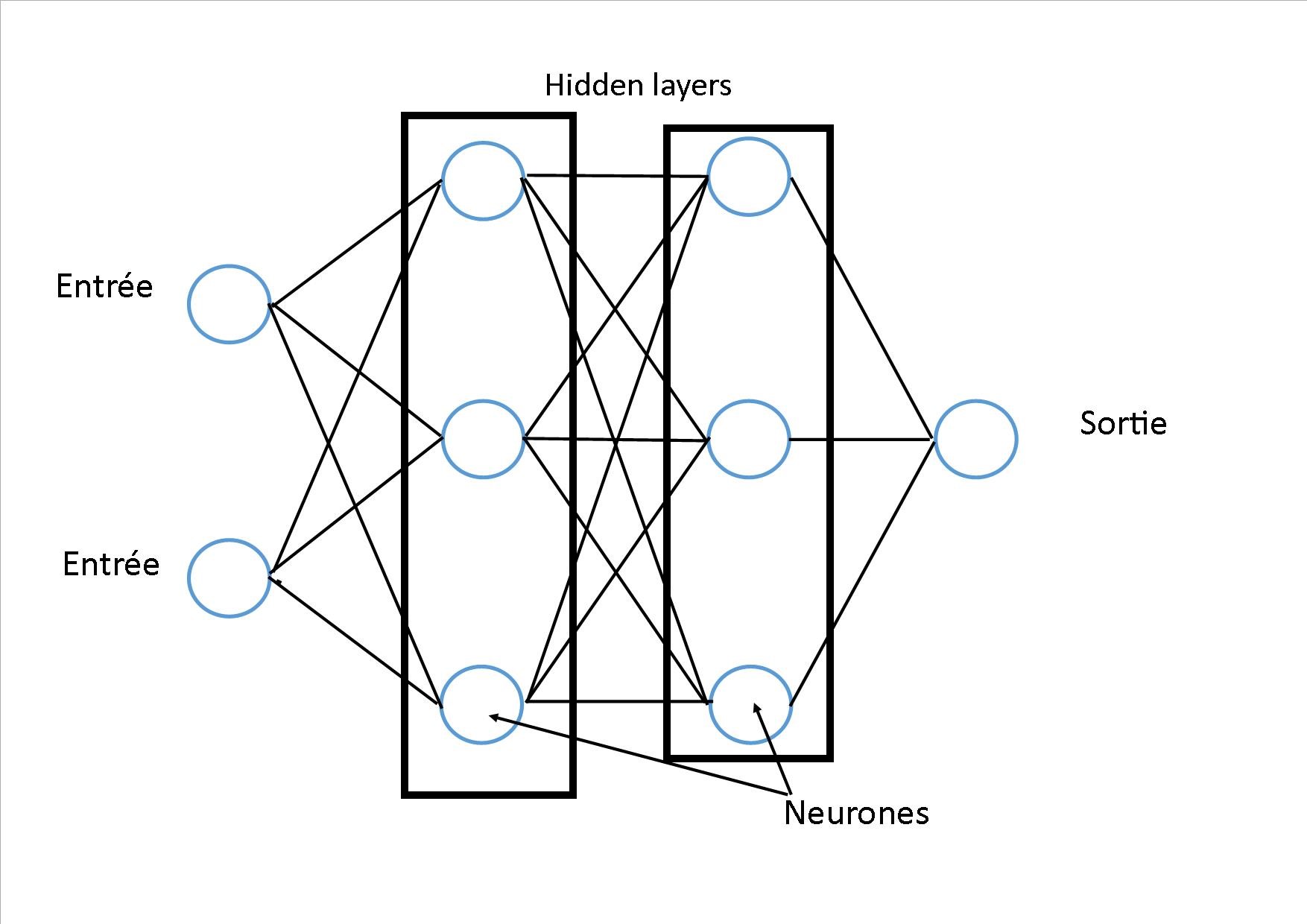


Dans ce cas on rentre une image d’un 7 de 28\*28 pixels dans notre réseau, ces pixels sont répartis en 784 entrées.

On obtient un 7 avec 0.8 de probabilité. Dans ce cas le réseau a trouvé le résultat avec succès.

1/ Description du réseau de neurones

Voici comment peut être schématisé un réseau de neurones « artificiels » : (Aurélien prends le fichier truc moche fait par clément V2 sur méga).



Un rond représente un **neurone** (node en anglais).

Ces **neurones** sont réliés entre eux par les **synapses**, représentées par un trait.

Un vecteur possède un **poids** (« weight » (noté w).

Ce **poids** correspond à l’importance de l’information qui passe par ce vecteur (Si le **poids** vaut 2,15 l’information prend une grande importance).

C’est dans le **neurone** que des calculs vont être effectués.

Un **neurone** possède un « **bias »** qui, comme le poids est réglé aléatoirement au début de l’apprentissage.

Un réseau neuronal possède une ou plusieurs **entrées** (deux en l’occurrence).

Un ou plusieurs « **Hidden layers** » (couches cachées) sont présents

Il y a deux « **Hidden layers** » dans le schéma ci-dessus (ils se trouvent dans les rectangles).

Une ou plusieurs **sorties** sont présentes (il n’y en a qu’une seule dans ce cas).

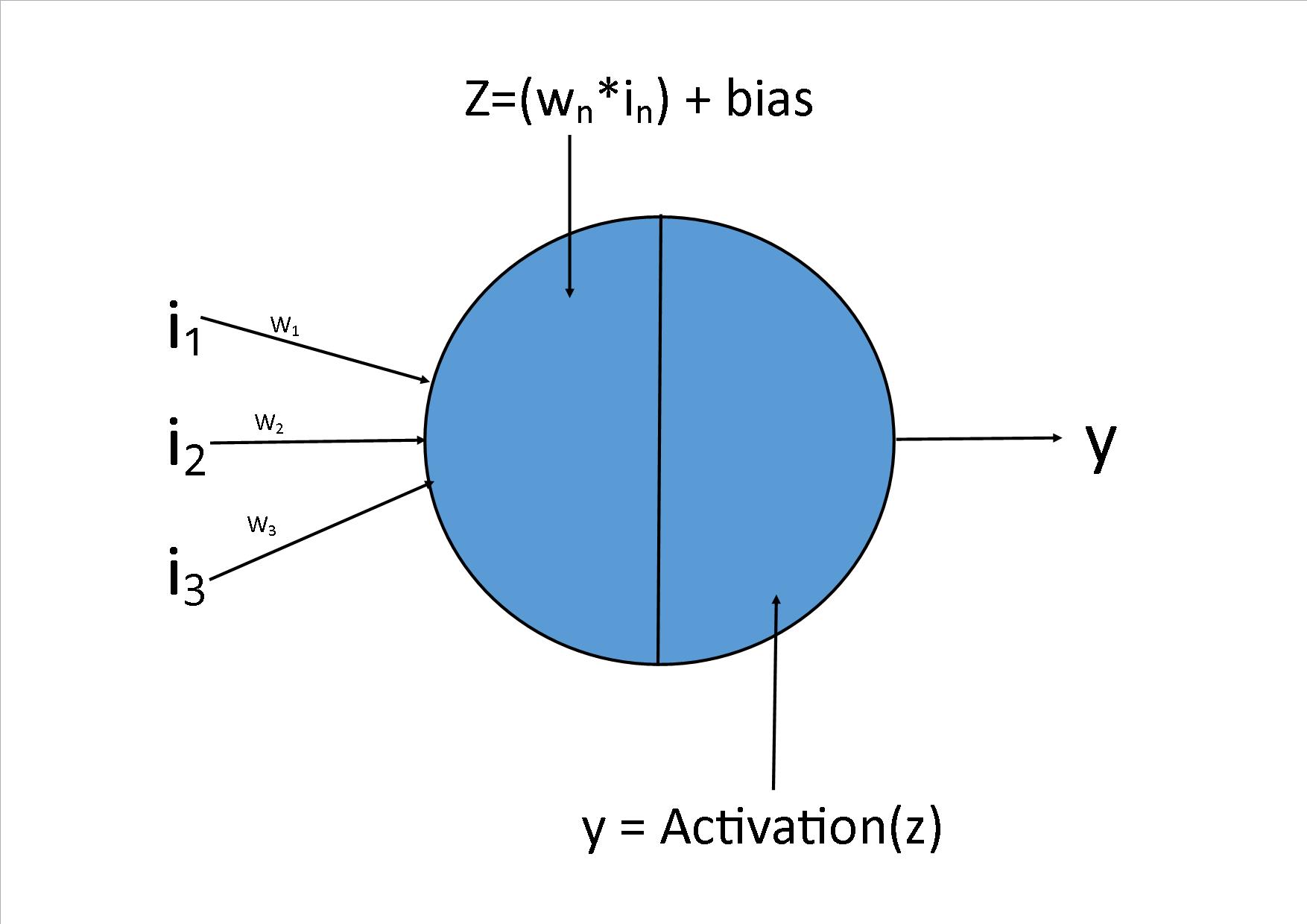
Dans le cas suivant il y a 18 **synapses** qui relient les neurones entre eux.

Les réseaux neuronaux sont des assemblages de « **matrices** »

2/ Description des coefficients du réseau neuronal

(Mettre modèle neurone x et y en dessous)

Voici ce qui se passe dans un neurone informatique :

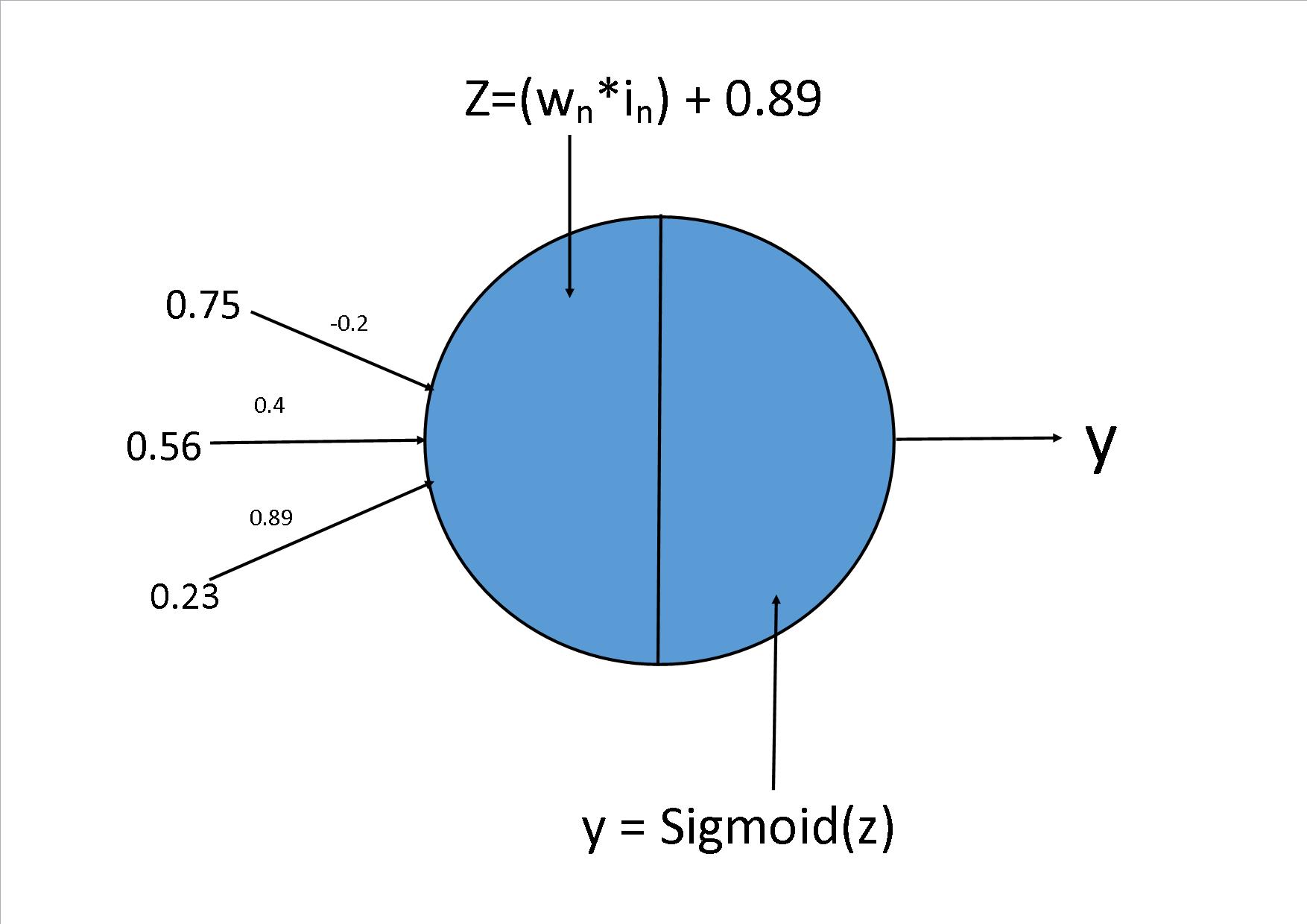


«i» (pour **input**/entrée en français) correspond à l’information envoyée par la synapse précédente (les y précédents).   
Dans ce cas il y a 3 **synapses** en entrée et elles se situent à gauche du **neurone**.  
« i » est obligatoirement compris entre 0 et 1.  
  
« W » correspond au **poids** de l’information (à son importance).  
Ce **poids** peut être positif ou négatif.  
  
Le « **bias** » permet de savoir où l’on se situe dans l’apprentissage (si on se rapproche du résultat espéré)   
Celui-ci peut être positif ou négatif.  
Il n’est pas très important pour la reconnaissance d’image en faible résolution donc l’on ne s’en sert pas.  
J’en parle tout de même car son utilisation est assez rependue.

Z correspond aux **Input** \* **poids** + **bias**Or cela ne donne pas forcément un résultat entre 0 et 1.  
On utilise donc « **l’activation** » qui est une fonction permettant d’obtenir un résultat entre 0 et 1.  
  
Les fonctions « **d’activation** » les plus utilisées sont les suivantes : reLU, Sigmoïde, Linéaire, Tanh.  
Pour l’exemple ci-dessous on utilise la fonction Sigmoïde (sigmoid en anglais)

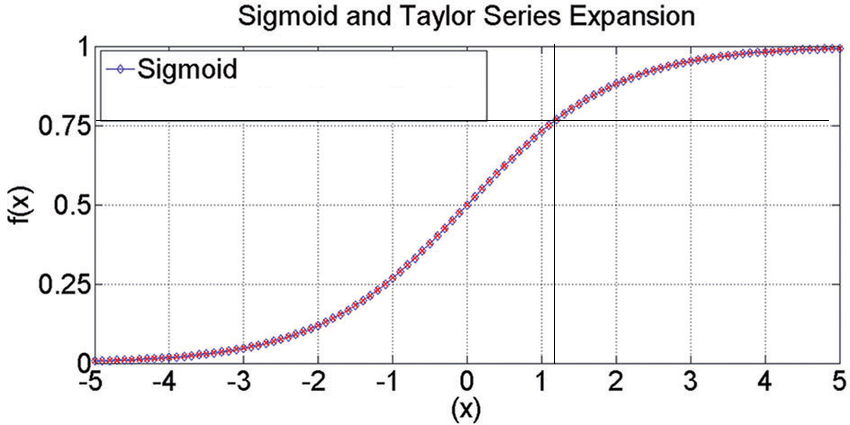
Exemple chiffré:

(Utilises « Modèle neurone (z et y) et Sigmoid.png).



On cherche d’abord Z.

Pour cela on multiplie les **inputs** par les **poids** et on les additionne (on effectue une moyenne) :   
0.75\*(-0.2) + 0.56\*0.4+0.23\*0.89 =0.2787  
  
On ajoute maintenant le bias : 0.2787+ 0.89 = 1.1687  
Z = 1.1687  
Or Z > 1  
On utilise donc la fonction « **D’activation** » (Sigmoïde) :   
Sigmoid (1.1687) ≈ 0.76



Cela donne donc un résultat compris entre 0 et 1.  
y = 0,76

#### Rajouter les dessins de titi

#### 3/ Réglage du réseau via l’apprentissage

#### Nous entrons maintenant dans la partie qui nous a posé le plus de problèmes. ( dsl c’est du travail en plus pour toi Aurélien mais ce serai bien que tu mettes des exemples des problèmes que tu as rencontrés (tu mets au fur et a mesure que j’explique)).

#### L’apprentissage permet de régler les coefficients du réseau neuronal en fournissant une entrée connue et en comparant le résultat du réseau avec le résultat attendu. Par exemple l’image d’un chien est présentée en entrée du réseau et la sortie est comparée avec « chien ».

#### 3,1/ Calcul de l’erreur

#### La Loss function (ou coût/cost en anglais) sert à connaitre le taux d’apprentissage et le taux d’erreur du réseau. Elle permet de savoir où l’on se trouve par rapport à la situation d’apprentissage « idéale ». On peut utiliser le « binary crossentropy » pour les réseaux complexes, cependant le nôtre ne requiert pas des résultats extrêmement précis. Nous avons donc décidé de prendre le résultat final auquel on soustrait le résultat attendu. Donc y(n) calculé – y prédit. (Le (n) sert à préciser le numéro du y, si c’est le 1er, le 2nd ...) Faire le calcul en élevant au carré permet d’augmenter l’écart et d’être plus précis. Nous avons testé avec l’élévation au carré et cela ne change rien.

#### Soit (y(n) calculé)² – (y prédit)² Voici à quoi ressemble la binary crossentropy à titre informatif:

#### 

#### Je n’explique pas son fonctionnement car on ne l’utilise pas et elle n’a pas une importance vitale dans un réseau de neurones artificiels. La méthode que l’on a choisie est beaucoup plus simple et quasiment aussi efficace à notre échelle.

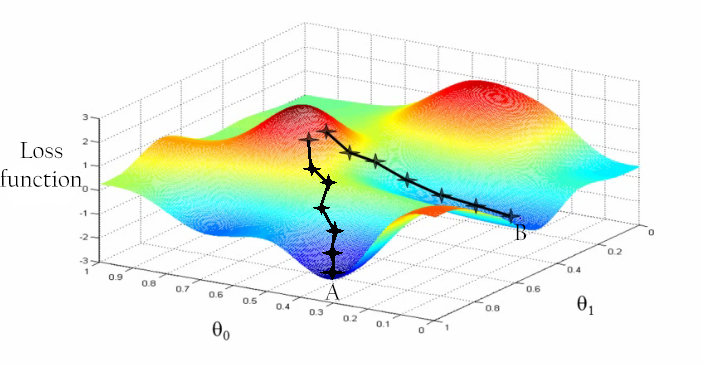
#### 3,2/ Optimisation des coefficients (poids et biais)

#### L’information remonte à travers le réseau grâce à la « back-propagation », que j’aborde plus tard.

#### Pour que l’apprentissage se fasse correctement on utilise la fonction « ****gradient descent »** (descente en gradient) qui a pour objectif de régler la valeur des **bias** et des **poids**. Le but est de faire en sorte que le résultat de la « lost function », décrite ci-dessus, soit le plus proche de zéro et donc du résultat idéal.**

#### (gradient\_descent\_demystified.png)

#### Ce qui donne cela dans un espace en 3D : ( prends « zebi.jpg »)



Plusieurs minimum « locaux » peuvent être atteints (il y en a deux : A et B dans notre exemple). On compare les sorties du réseau pour le réglage « A » au réglage « B » et on garde celui qui donne le résultat le plus proche de l’attendu.



Mais avant, il faut régler la «**vitesse d’apprentissage** » manuellement et un peu au hasard (à tâtons).  
Le nôtre est de 0,01 (je crois). S’il est trop élevé on rate le résultat et s’il est trop faible, on met beaucoup trop de temps à l’atteindre. (**izi.png**)

#### 3.3/ La diffusion des coefficients et valeurs dans le réseau. (pas forcément utile)

#### La « Forward propagation » correspond à l’ensemble des actions qu’une valeur subit de l’entrée du réseau à la sortie.

#### La « Backward propagation » permet de régler les coefficients du réseau. Elle parcourt donc le sens inverse de la « Forward propagation » (uracile.png)