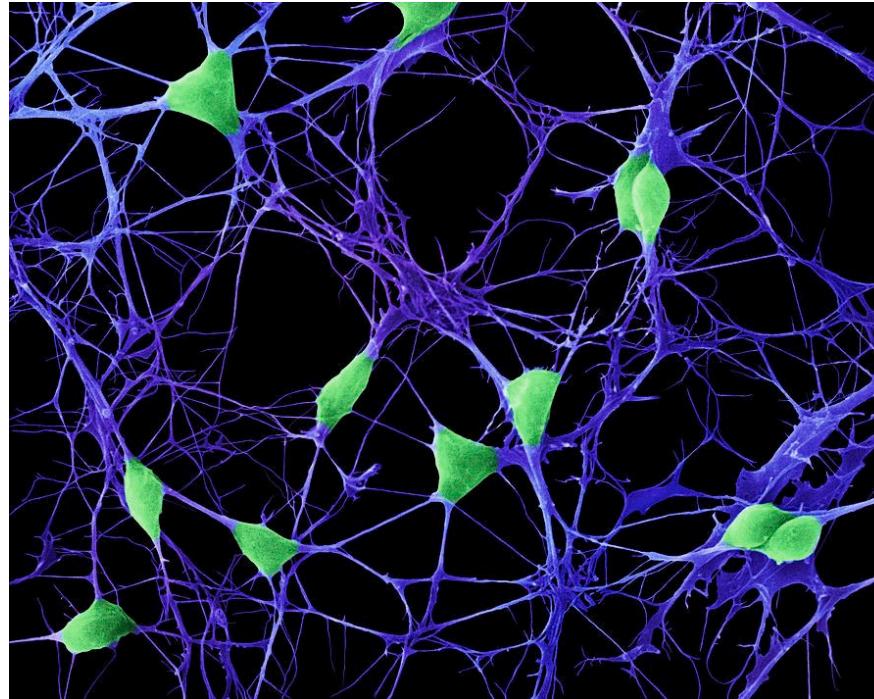
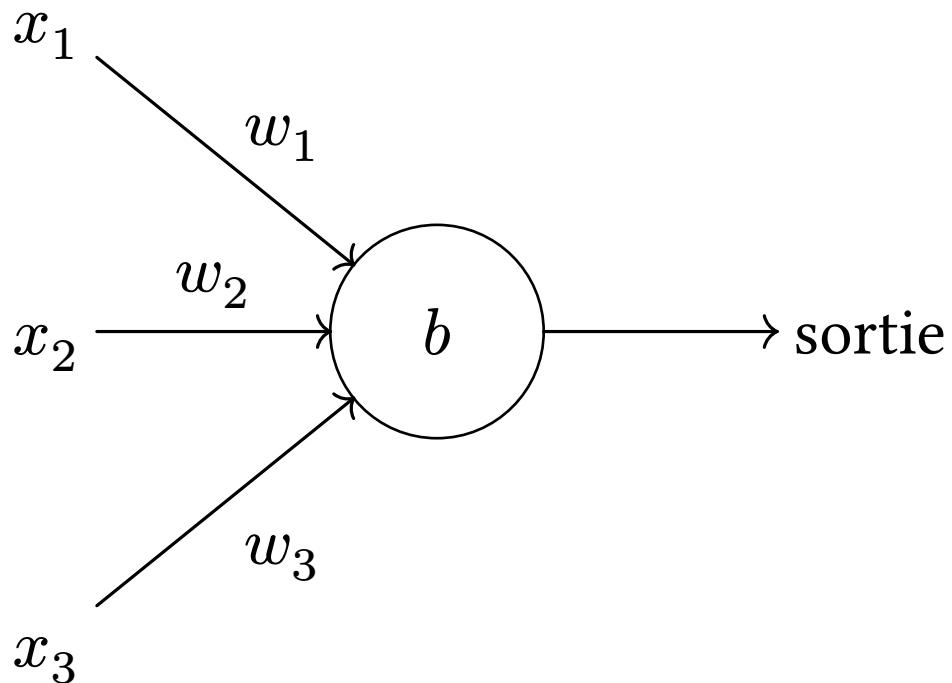


Deep Learning

L'intuition biologique



Un premier neurone artificiel : le perceptron



x_1, x_2, x_3 : entrées

w_1, w_2, w_3 : poids (*weights*)

b : biais (*bias*)

Renvoie « oui » si $w_1x_1 + w_2x_2 + w_3x_3 + b > 0$, renvoie « non » sinon.

Exemple de prise de décision avec un perceptron

Question : *Est-ce que vous irez au festival ?*

Les facteurs à prendre en compte sont :

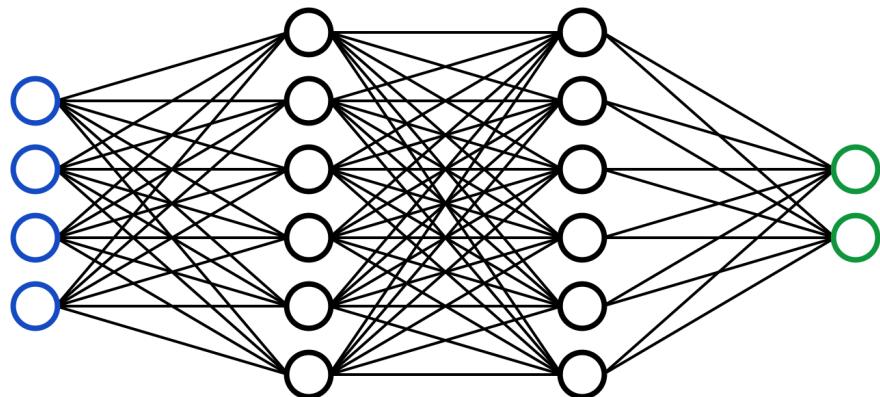
x_1 = est-ce qu'il fait beau ?

x_2 = y a-t-il un bus pour aller au festival ? (vous n'avez pas de voiture)

x_3 = est-ce qu'un ami veut bien vous accompagner ?

Question : à quoi correspondent les poids w_1, w_2, w_3 et le biais b ?

Un exemple de Deep Learning



L'exemple qu'on va mettre en place

0	4	1	9	2	1	3	1	4	3
5	3	6	1	7	2	8	6	9	4
0	9	1	1	2	4	3	2	7	3
8	6	9	0	5	6	0	7	6	1
8	7	9	3	9	8	5	9	3	3
0	7	4	9	8	0	9	4	1	5
4	6	0	4	5	6	1	0	0	1
7	1	6	3	0	2	1	1	1	9
0	2	6	7	8	3	9	0	4	6
7	4	6	8	0	7	8	3	1	5

Exercice : la fonction NAND

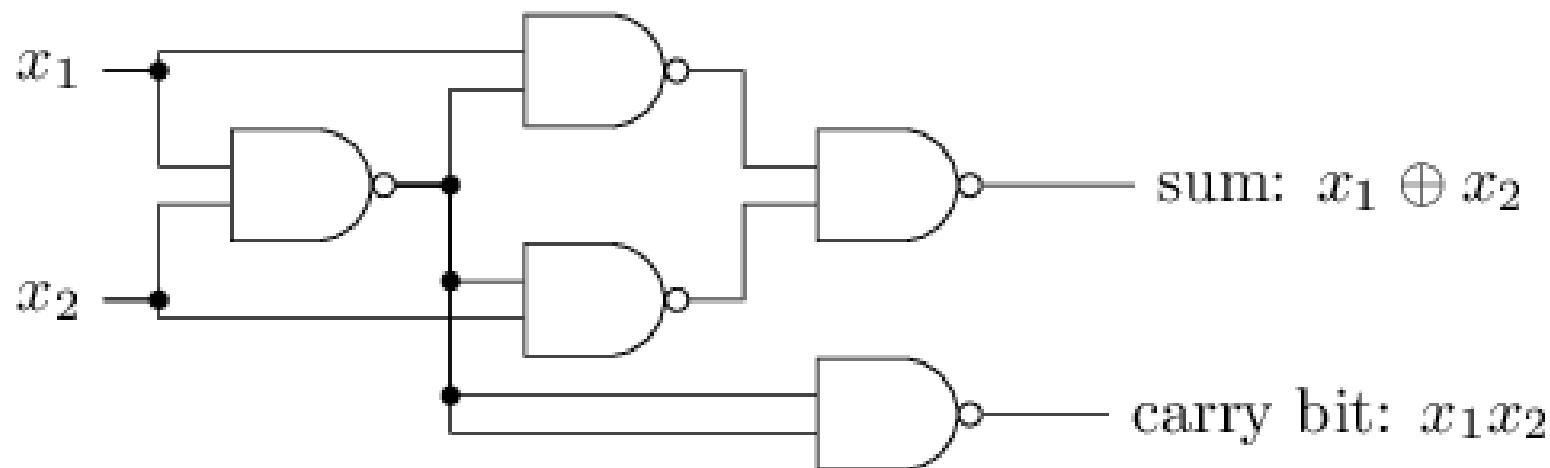
NAND est une fonction logique qui a la table de vérité suivante :

x_1	x_2	$\text{NAND}(x_1, x_2)$
1	1	0
1	0	1
0	1	1
0	0	1

Comment peut-on faire un perceptron qui implémente la fonction NAND ?

L'universalité des portes NAND

En combinant des portes NAND, on peut programmer n'importe quelle porte logique. Par exemple, voici une série de NAND qui fait une addition de deux chiffres binaires :



Les réseaux de neurone : quel intérêt ?

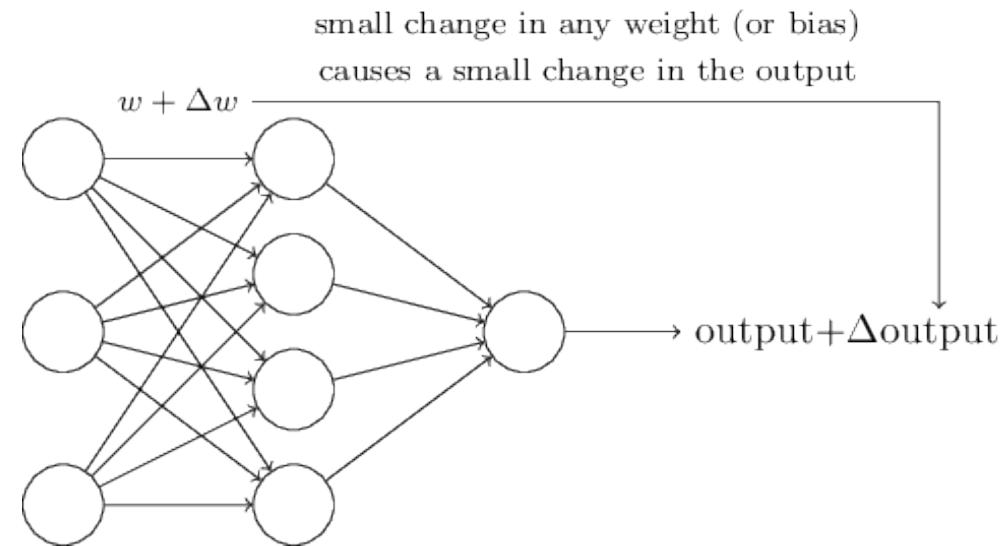
- Les perceptrons permettent de coder n’importe quelle fonction logique, donc n’importe quelle fonction informatique !
- C’est une bonne nouvelle, mais c’est aussi un peu décevant... Quel intérêt d’utiliser des perceptrons plutôt que des portes logiques classique ?

Les réseaux de neurone : quel intérêt ?

- Les perceptrons permettent de coder n’importe quelle fonction logique, donc n’importe quelle fonction informatique !
- C’est une bonne nouvelle, mais c’est aussi un peu décevant... Quel intérêt d’utiliser des perceptrons plutôt que des portes logiques classique ?
- L’intérêt c’est qu’on peut faire en sorte que le réseau de neurone apprenne les poids et les biais sans intervention humaine, juste à partir d’exemples.

Le problème des perceptrons

On aimerait bien être dans la situation suivante pour que le réseau apprenne :



La solution : les neurones « sigmoïdes »

Un neurone sigmoïde fonctionne comme un perceptron, sauf que la sortie, au lieu de valoir 0 ou 1, vaut :

$$\sigma(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

avec :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

La solution : les neurones « sigmoïdes »

Un neurone sigmoïde fonctionne comme un perceptron, sauf que la sortie, au lieu de valoir 0 ou 1, vaut :

$$\sigma(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

avec :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(Un peu de notation : plutôt que d'écrire $w_1x_1 + w_2x_2 + \dots$, on écrira : $w \cdot x$)

La solution : les neurones « sigmoïdes »

Un neurone sigmoïde fonctionne comme un perceptron, sauf que la sortie, au lieu de valoir 0 ou 1, vaut :

$$\sigma(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

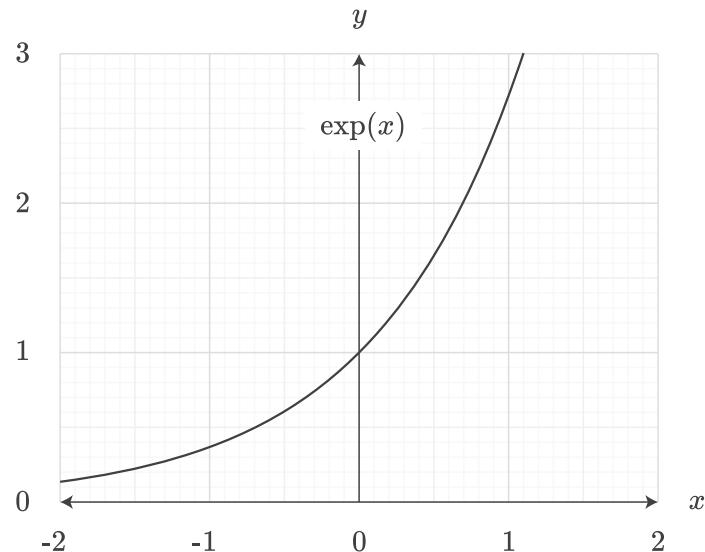
avec :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

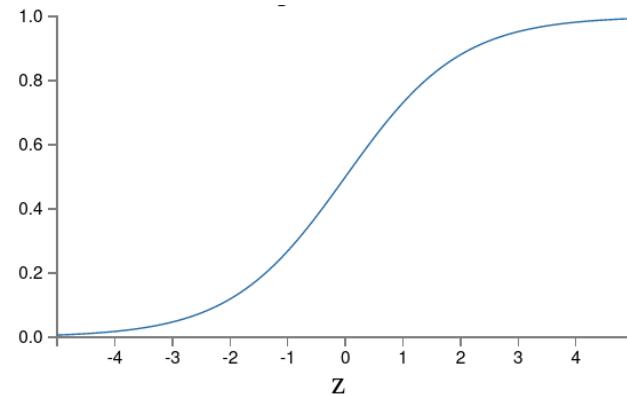
(Un peu de notation : plutôt que d'écrire $w_1x_1 + w_2x_2 + \dots$, on écrira : $w \cdot x$)

Question : coder la fonction `sigmoid(z)` et afficher son graphe.

Comprendre la fonction sigmoïde visuellement

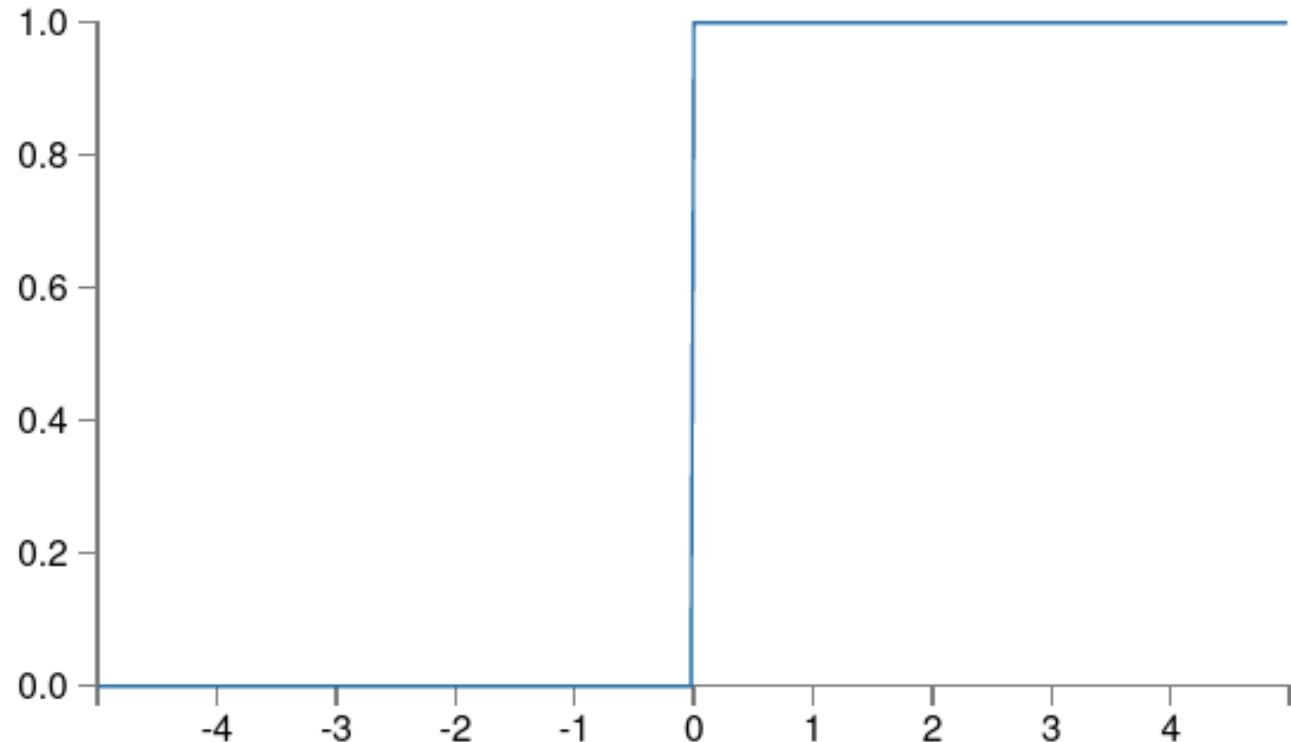


Fonction exponentielle



Fonction sigmoïde

On n'est pas si loin du perceptron



L'intérêt de la fonction sigmoïde

$$\Delta \text{ sortie} = \sum_j \frac{\partial \text{ sortie}}{\partial w_j} \Delta w_j + \frac{\partial \text{ sortie}}{\partial b} \Delta b$$

Quelle architecture pour notre réseau de neurones ?

- Combien d'entrées ? (Les images de chiffre sont de taille 28×28 .)

Quelle architecture pour notre réseau de neurones ?

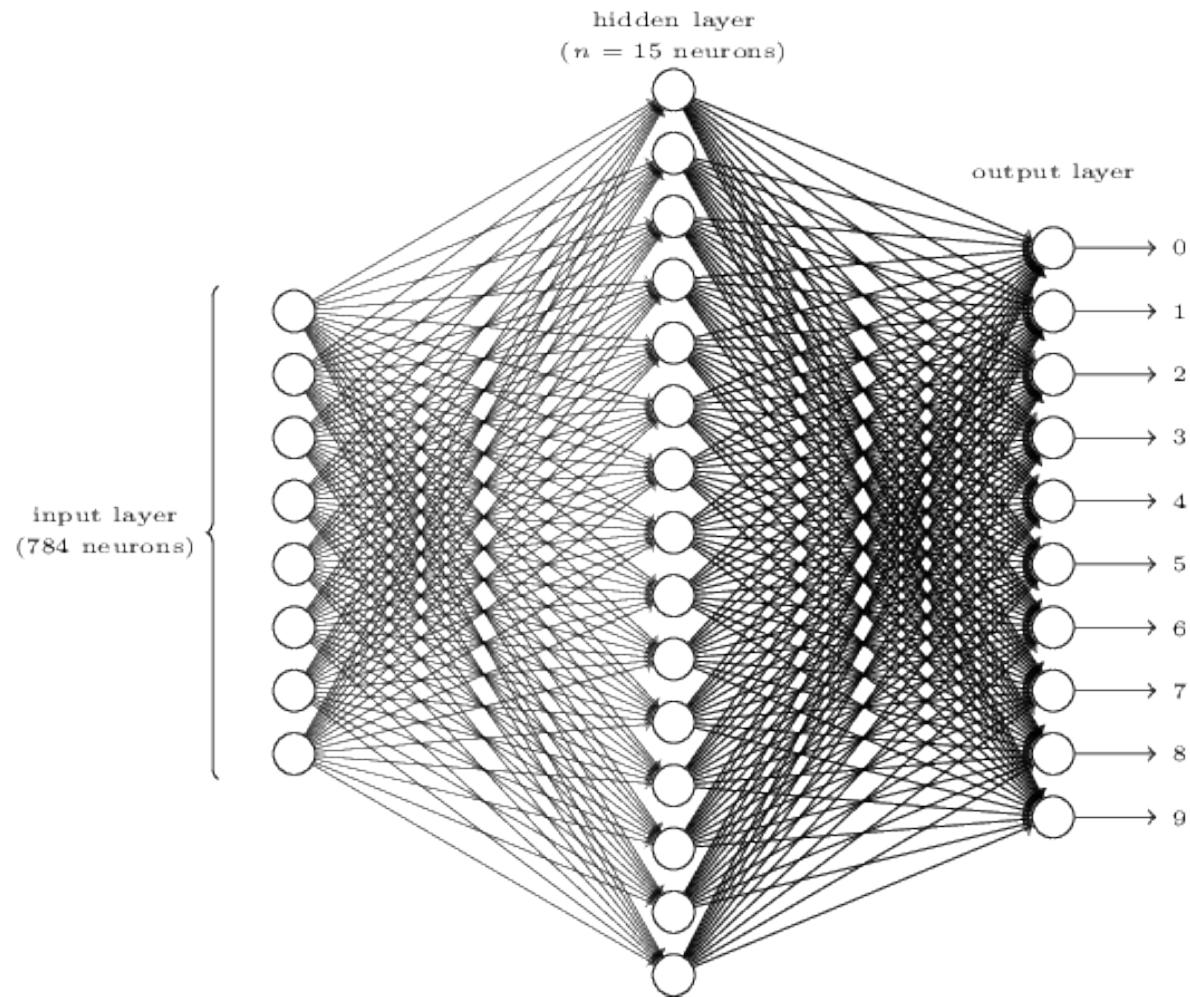
- Combien d'entrées ? (Les images de chiffre sont de taille 28×28 .)
- Combien de neurones en sortie ?

Quelle architecture pour notre réseau de neurones ?

- Combien d'entrées ? (Les images de chiffre sont de taille 28×28 .)
- Combien de neurones en sortie ?
- Est-ce qu'on ajoute autre chose ?

Quelle architecture pour notre réseau de neurones ?

- Combien d'entrées ? (Les images de chiffre sont de taille 28×28 .)
- Combien de neurones en sortie ?
- Est-ce qu'on ajoute autre chose ?



Comment mesurer la performance de notre réseau ?

Un peu de notation :

- On note une entrée d'entraînement par x . C'est un vecteur de taille $28 \times 28 = 784$.
- On note la sortie désirée correspondant à x par $y(x)$. Par exemple, si x est une image du chiffre 4, $y(x) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)$.

Comment mesurer la performance de notre réseau ?

Un peu de notation :

- On note une entrée d'entraînement par x . C'est un vecteur de taille $28 \times 28 = 784$.
- On note la sortie désirée correspondant à x par $y(x)$. Par exemple, si x est une image du chiffre 4, $y(x) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)$.

On définit la fonction *coût* (aussi appelée *perte* ou *objectif*) par...

Comment mesurer la performance de notre réseau ?

Un peu de notation :

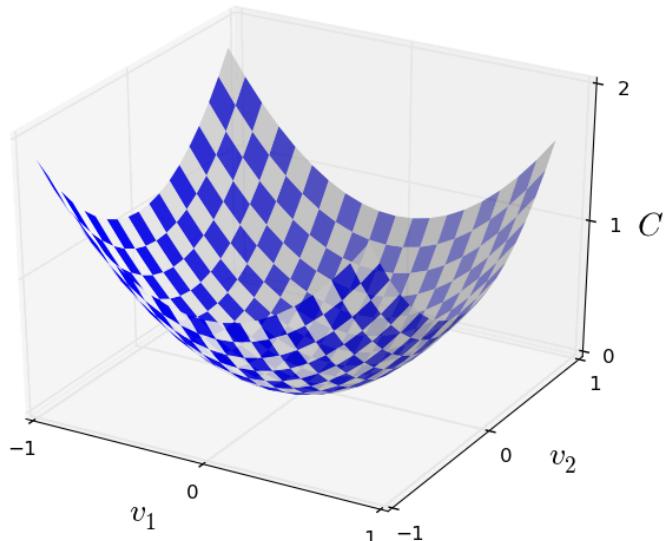
- On note une entrée d'entraînement par x . C'est un vecteur de taille $28 \times 28 = 784$.
- On note la sortie désirée correspondant à x par $y(x)$. Par exemple, si x est une image du chiffre 4, $y(x) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)$.

On définit la fonction *coût* (aussi appelée *perte* ou *objectif*) par...

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Comment minimiser la fonction coût ? (1)

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2$$



On cherche Δv_1 et Δv_2 de sorte à ce que ΔC soit négatif.

Comment minimiser la fonction coût ? (2)

On définit $\Delta v = (\Delta v_1, \Delta v_2)$ et $\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)$. On a :

$$\Delta C \approx \nabla C \cdot \Delta v$$

∇C s'appelle le *gradient* de C , c'est un peu comme la « dérivée » de C .

Pour que ΔC soit négatif, on peut choisir la valeur suivante pour Δv :

Comment minimiser la fonction coût ? (2)

On définit $\Delta v = (\Delta v_1, \Delta v_2)$ et $\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)$. On a :

$$\Delta C \approx \nabla C \cdot \Delta v$$

∇C s'appelle le *gradient* de C , c'est un peu comme la « dérivée » de C .

Pour que ΔC soit négatif, on peut choisir la valeur suivante pour Δv :

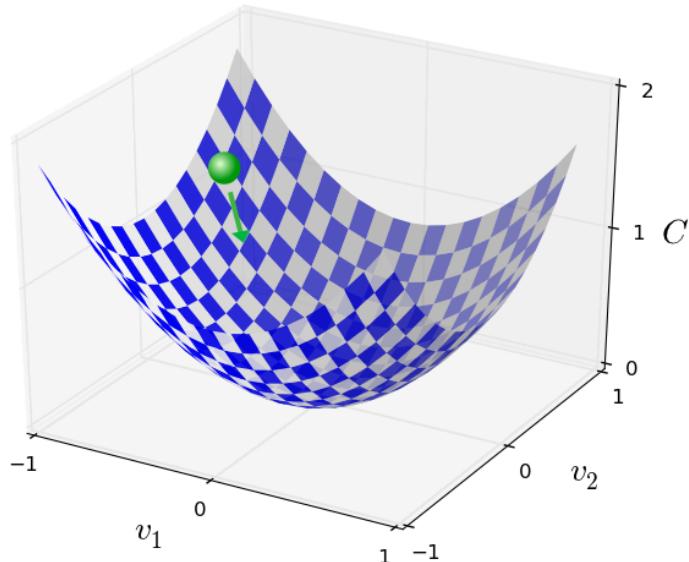
$$\Delta v = -\eta \nabla C$$

avec $\eta > 0$ un petit nombre qu'on appelle le *pas*.

En effet :

$$\Delta C \approx \nabla C \cdot \Delta v = -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 < 0$$

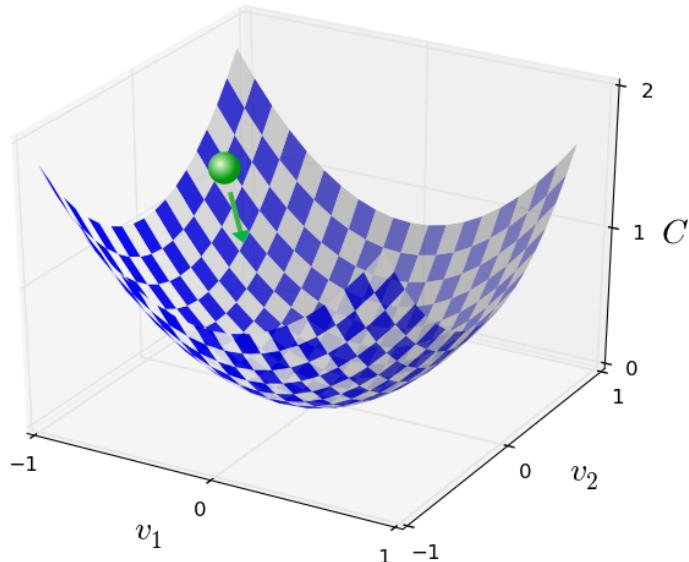
L'algorithme de descente de gradient



Pour minimiser $C(v)$, répéter :

$$v \rightarrow v - \eta \nabla C$$

L'algorithme de descente de gradient



Pour minimiser $C(v)$, répéter :

$$v \rightarrow v - \eta \nabla C$$

Exercice : effectuer 3 itérations de l'algorithme avec $\eta = 0.1$ et :

1. $f(x) = x^4 - x^3$
2. $C(v_1, v_2) = -v_1^3 + v_2^2$
3. $C(v_1, v_2) = v_1^2 v_2^2$

Appliquer la descente de gradient à notre réseau de neurones

$$C = \frac{1}{2n} \sum_x \|y(x) - x\|^2, w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k}, b_l \rightarrow b_l - \eta \frac{\partial C}{\partial b_l}$$

Appliquer la descente de gradient à notre réseau de neurones

$$C = \frac{1}{2n} \sum_x \|y(x) - x\|^2, w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k}, b_l \rightarrow b_l - \eta \frac{\partial C}{\partial b_l}$$

On reparque que $C = \frac{1}{n} \sum_x C_x$ avec $C_x = \|y(x) - x\|^2/2$, et donc :

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

En pratique, on calcule les ∇C_x individuellement pour calculer ∇C .

La descente de gradient stochastique

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Problème de cette formule : s'il y a beaucoup de x (= beaucoup de données d'entraînement), ∇C prend longtemps à calculer.

La descente de gradient stochastique

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Problème de cette formule : s'il y a beaucoup de x (= beaucoup de données d'entraînement), ∇C prend longtemps à calculer.

Solution : on calcule ∇C avec seulement m exemples d'entraînements choisis au hasard, qu'on appelle un *mini-batch* :

$$\nabla C \approx \frac{1}{m} \sum_{x \in \text{mini-batch}} \nabla C_x$$

Si on a 60000 exemples en tout mais qu'on prend des mini-batchs de taille 10, on calcule le gradient ~ 6000 plus vite.

Implémentation en Python : la classe Réseau

Exercice : définir une classe Reseau qui prend comme paramètre d'initialisation une liste d'entier tailles. Cette classe a deux attributs qui sont initialisés avec des valeurs aléatoires entre -1 et 1 :

- biais est une liste de `len(tailles)` *ndarrays* telle que `biais[0]` est de dimension `(taille[0], 1)`, `biais[1]` de dimension `(taille[1], 1)`, etc.
- poids est une liste de `len(tailles) - 1` *ndarrays* telle que `poids[0]` est de dimension `(tailles[0], tailles[1])`, `poids[1]` de dimension `(tailles[1], tailles[2])`, etc.

La classe Réseau : correction

```
class Reseau:

    def __init__(self, tailles):
        self.nb_couches = len(tailles)
        self.tailles = tailles

        self.biais = [np.random.uniform(-1., 1., size=(y, 1))
                      for y in tailles[1:]]
        self.poids = [np.random.uniform(-1., 1., size=(y, x))
                      for (y, x) in zip(tailles[:-1], tailles[1:])]
```

Implémentation en Python : la méthode `forward`

Implémentez la méthode `forward(self, x)`, qui renvoie la sortie du réseau pour une entrée `x`. Indice : si `x` et `w` sont des *ndarrays*, la méthode `np.dot(x, w)` permet de calculer $x \cdot w$.

Implémentation en Python : la méthode `forward`

Implémentez la méthode `forward(self, x)`, qui renvoie la sortie du réseau pour une entrée `x`. Indice : si `x` et `w` sont des *ndarrays*, la méthode `np.dot(x, w)` permet de calculer $x \cdot w$.

Correction courte :

```
def forward(self, x):
    for b, w in zip(self.biais, self.poids):
        x = sigmoide(np.matmul(w.T, x)+b)
    return x
```

Implémentation en Python : la méthode `evaluer`

Implémentez la méthode `evaluer(self, test_data)`, qui prend en entrée une liste de paires (x, y) et qui renvoie le nombre de fois où le réseau prédit correctement y à partir de x .

Lancez l'entraînement !

Exécutez le fichier `main.py`.

La méthode DGS (*Descente de Gradient Stochastique*)

```
def DGS(self, training_data, nb_epoques, taille_mini_batch,
eta, test_data=None):
    if test_data is not None: n_test = len(test_data)
    n = len(training_data)
    for j in range(nb_epoques):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k+taille_mini_batch]
            for k in range(0, n, taille_mini_batch)]
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, eta)
        if test_data is not None:
            print(f"Époque {j}: {self.evaluer(test_data)} /
{n_test}")
        else:
            print(f"Époque {j} complete")
```

La méthode update_mini_batch

```
def update_mini_batch(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biais]
    nabla_w = [np.zeros(w.shape) for w in self.poids]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b,
delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w,
delta_nabla_w)]
    self.poids = [w-(eta/len(mini_batch))*nw
                 for w, nw in zip(self.weights, nabla_w)]
    self.biais = [b-(eta/len(mini_batch))*nb
                  for b, nb in zip(self.biases, nabla_b)]
```

L'entraînement devrait être (presque) fini.

Quel pourcentage de réussite obtenez-vous
sur les données de test ?

Une question pour préparer la suite

Qu'est-ce qu'on pourrait faire différemment ?

Une question pour préparer la suite

Qu'est-ce qu'on pourrait faire différemment ?

- L'architecture du réseau
- Le pas η .
- Le nombre de mini-batch
- Le nombre d'époques

Une question pour préparer la suite

Qu'est-ce qu'on pourrait faire différemment ?

- L'architecture du réseau
- Le pas η .
- Le nombre de mini-batch
- Le nombre d'époques
- La fonction d'activation
- La fonction *coût*
- L'initialisation des poids et des biais
- ...