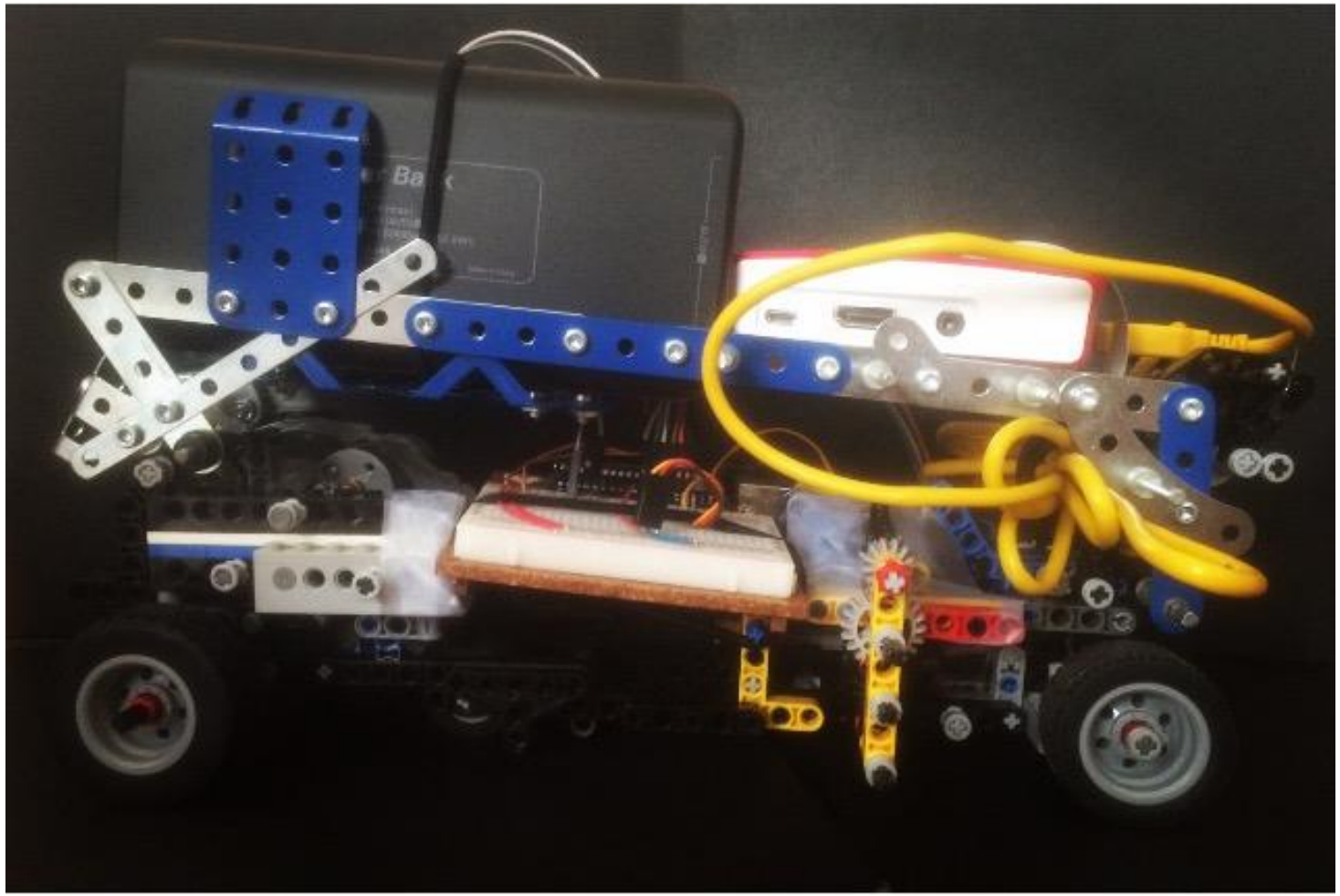
VOITURE AUTONOME À RÉSEAUX DE NEURONES



Amaury De Miguel

Pierre-Louis Guillou

SOMMAIRE

- INTRODUCTION
- ÉLABORATION DU MODÈLE
- INTELLIGENCE ARIFICIELLE
- ADPATION AU PROBLÈME
- CONCLUSION

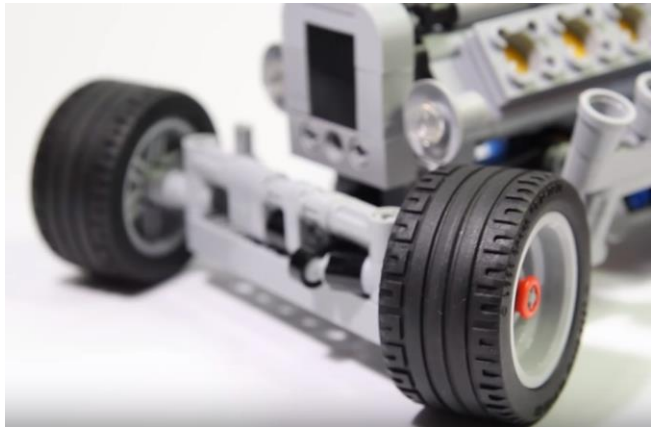
INTRODUCTION

INTRODUCTION

ÉLABORATION DU MODÈLE

CAHIER DES CHARGES

- Modèle manipulable avec des dimensions raisonnables
- Liaison pour les roues avant particulière
- Utilisation variée du système LEGO



PROPULSION

Choix du moteur

- Calcul de couple avec masse évaluée à 1.5kg et 230 tours par minutes

Vitesse linéaire : $v = 0,6m.s^{-1}$

Rayon des roues : $r = 2,5cm$

$\omega = \frac{v}{r} = 24rad.s^{-1}$

$F_{poids} = mg = 14,7N$

P=$F_{poids} \cdot v = 8,82N.m.s^{-1}$

Or par définition on a aussi $P = C\omega$ avec C le couple recherché

D'où $C = \frac{F_{poids} \cdot v}{\omega} = 0,37Nm$

PROPULSION

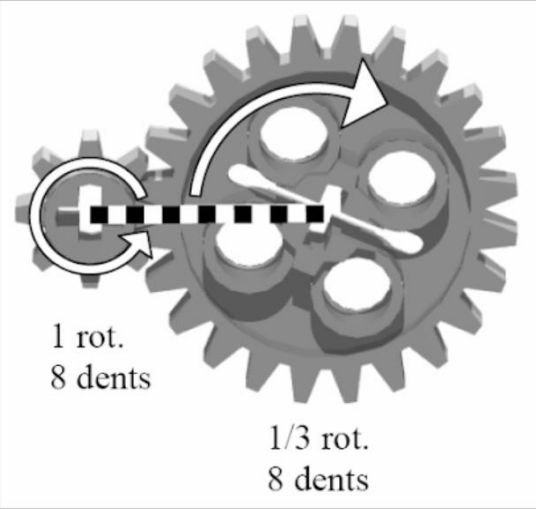
Choix du moteur

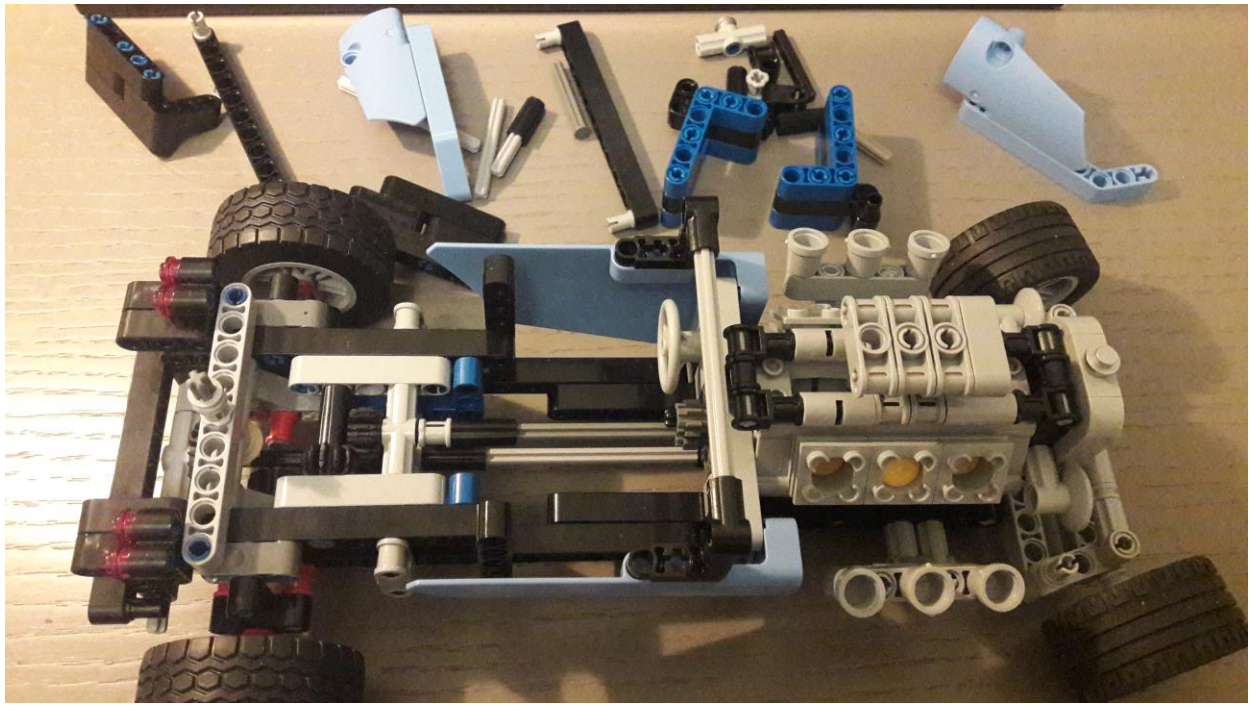
- Moteur à courant continu avec couple de $0.7\mathrm{N.m}$ et 485 tours par minutes
- Alimentation avec batterie légère de $12\mathrm{V}$

PROPULSION

Adaptation du modèle

- Modification de l'architecture de base pour relier le moteur aux roues
- Réduction du nombre de tours par minutes

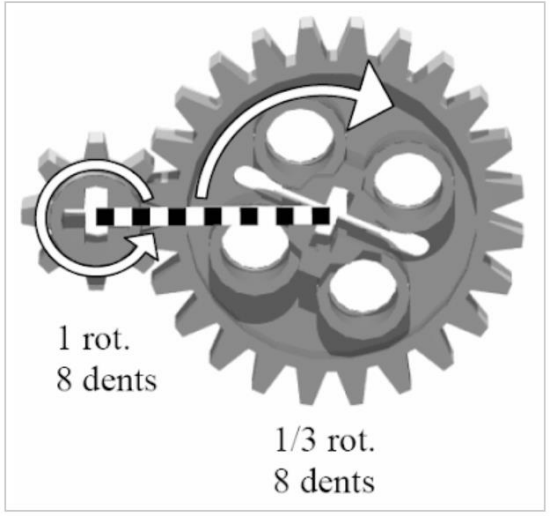


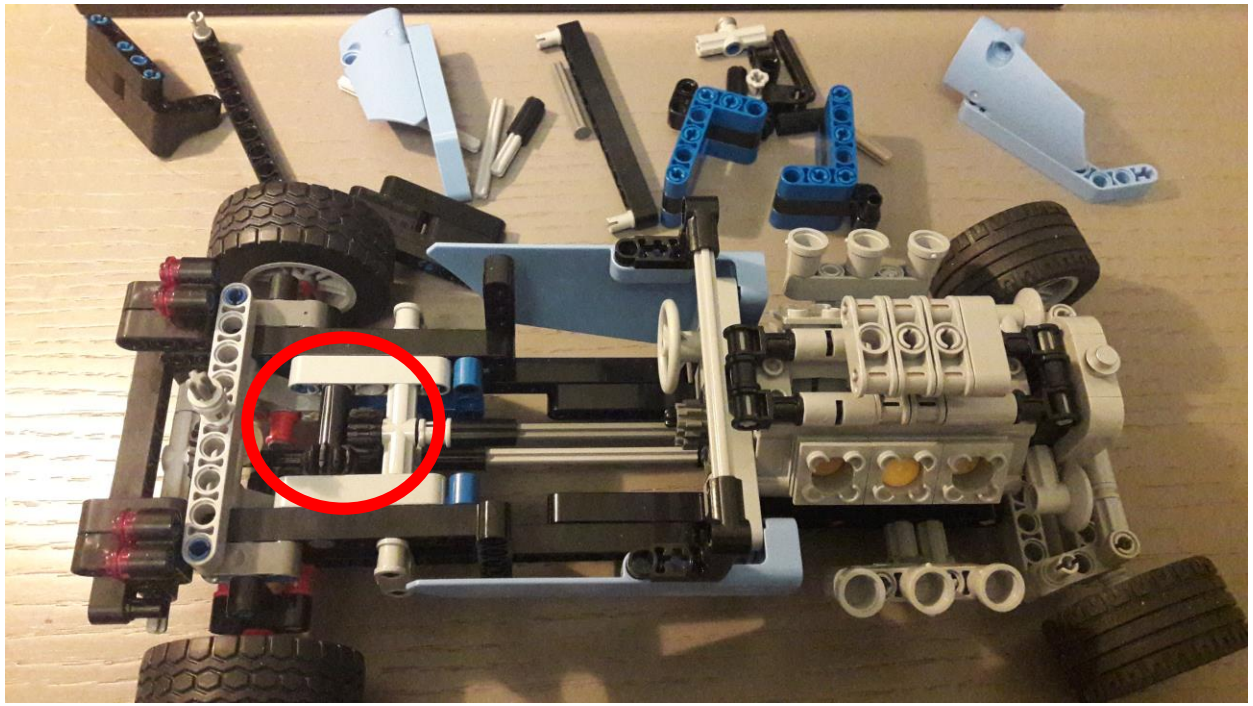


PROPULSION

Adaptation du modèle

- Modification de l'architecture de base pour relier le moteur aux roues
- Réduction du nombre de tours par minutes

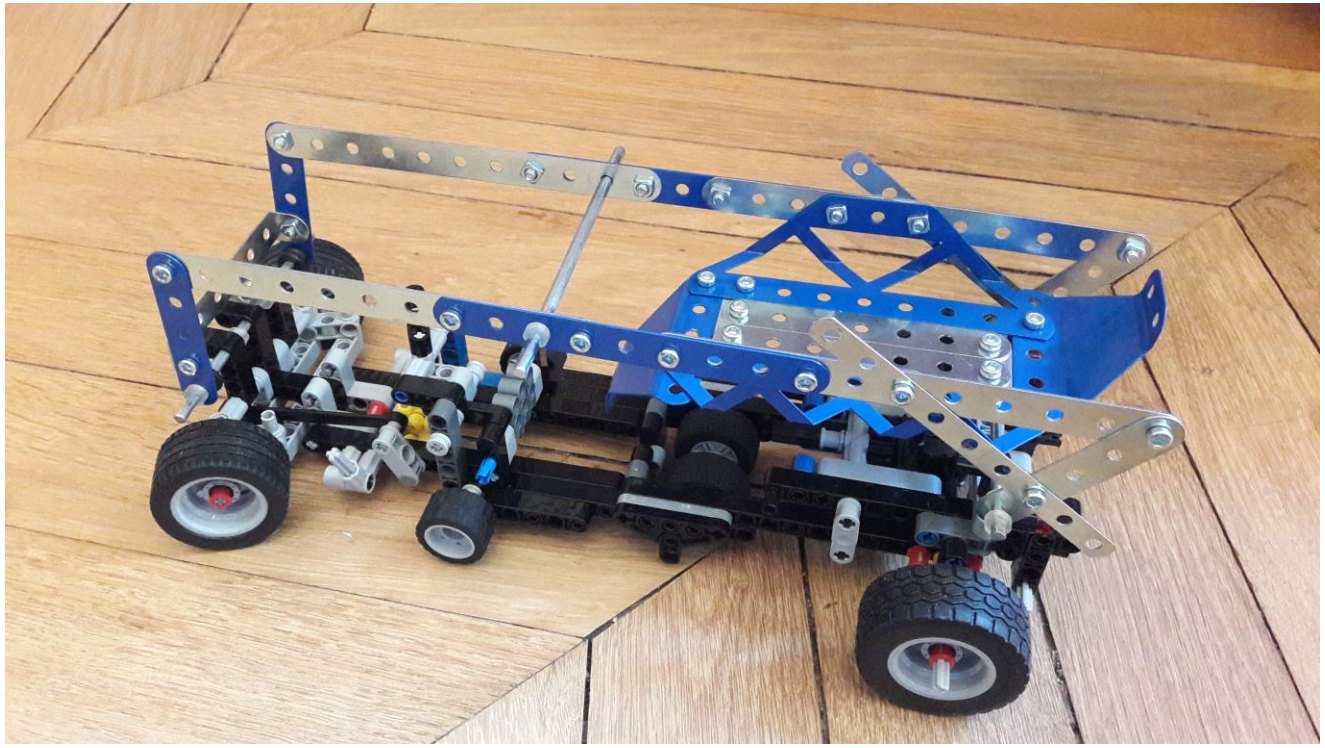




PROPULSION

Adaptation du modèle

- Allongement de la base pour tous les éléments à prévoir
- Intérêt superstructure métallique à la fois pour supporter les éléments et pour favoriser le contact entre roues et sol



DIRECTION

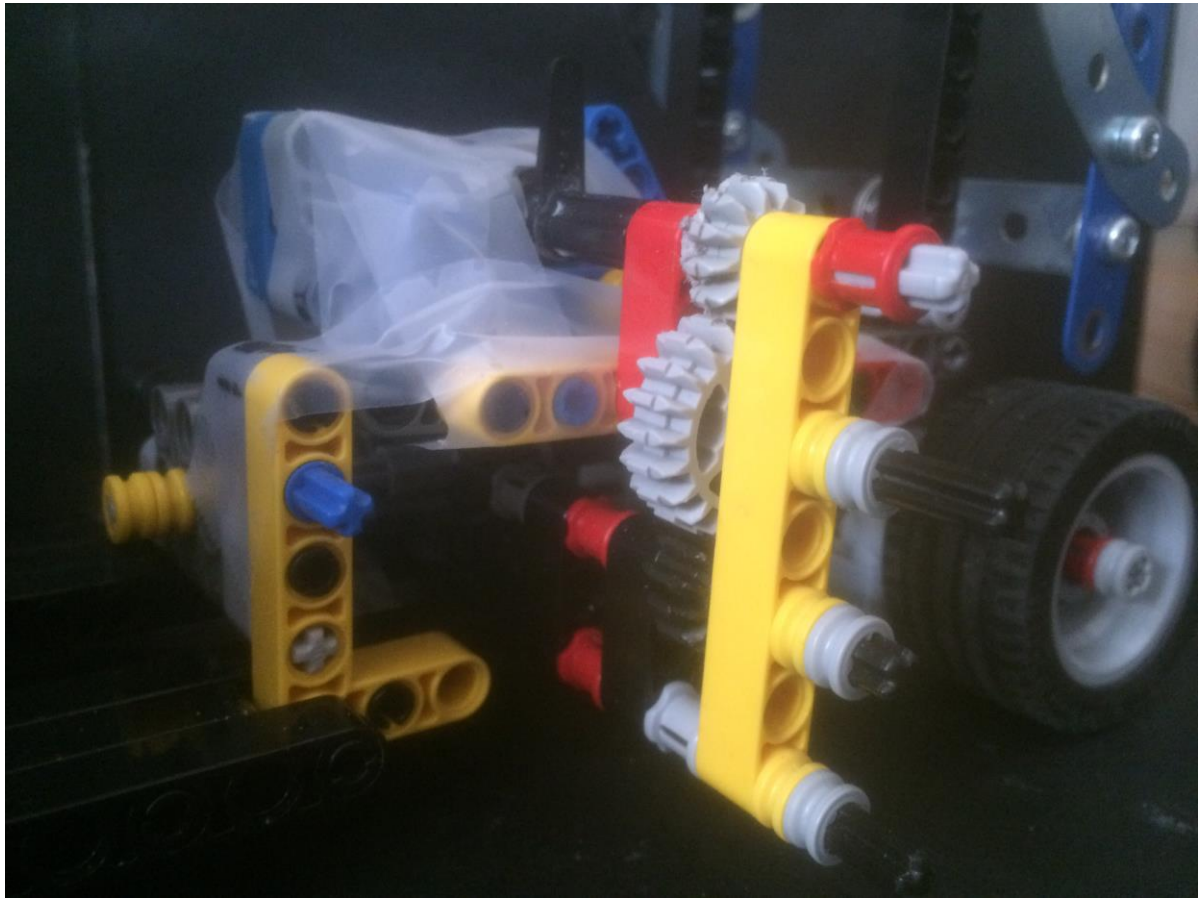
Choix du moteur

- Couple évalué par tests successifs sur un moteur de couple $0.12\mathrm{N.m}$
- Servo-moteur de couple $0.2\mathrm{N.m}$
- Alimentation via la carte de commande des moteurs

DIRECTION

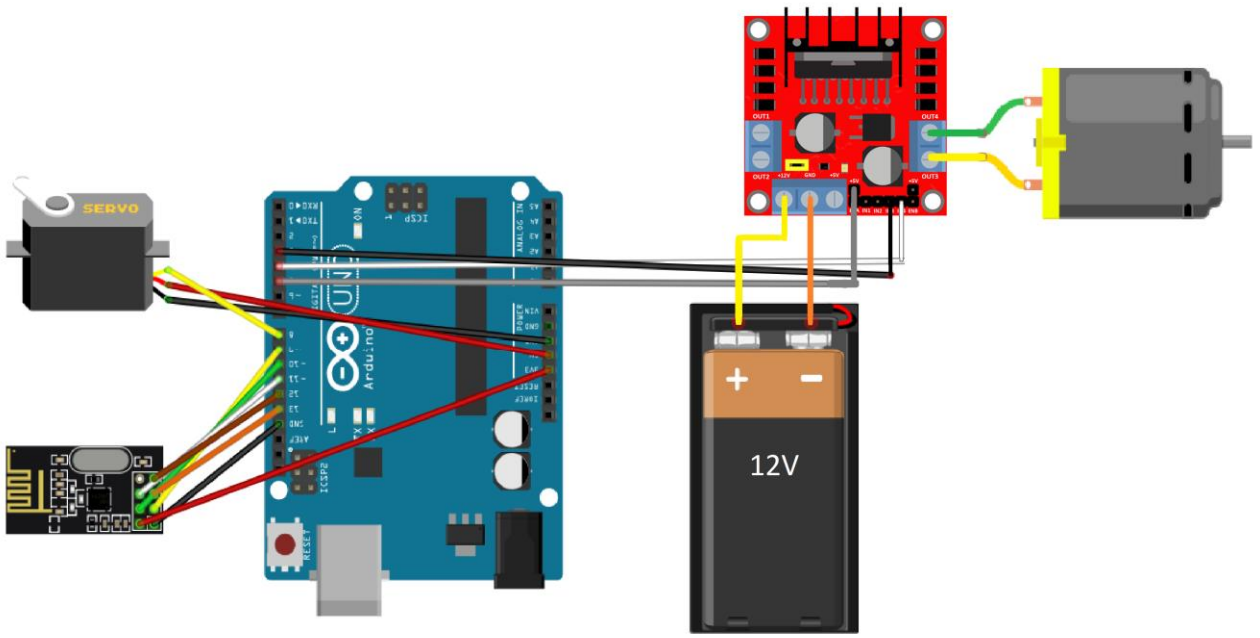
Adaptation du modèle

- Modifications de la structure de base pour pouvoir relier plus facilement le moteur aux roues avant
- Conserver la liaison qui donne son intérêt au modèle
- Utilisation des engrenages LEGO pour transmettre



FONCTIONNEMENT

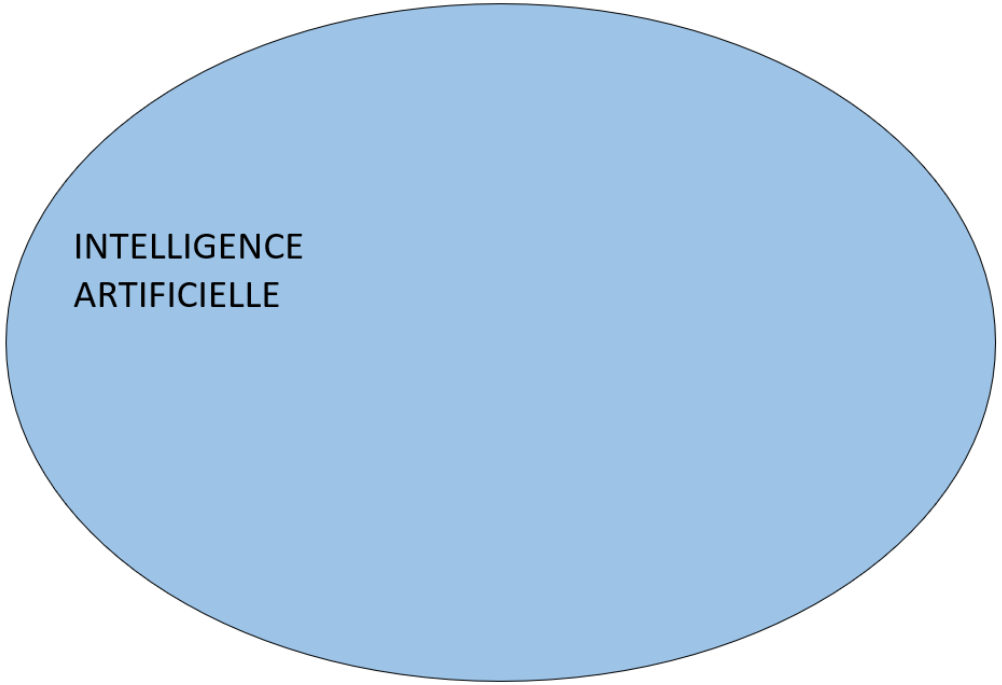
- Commande des moteurs via une carte Arduino
- Alimentation en 5V : utilisation d'un relais pour le moteur à courant continu alimenté en 12V



INTELLIGENCE ARTIFICIELLE

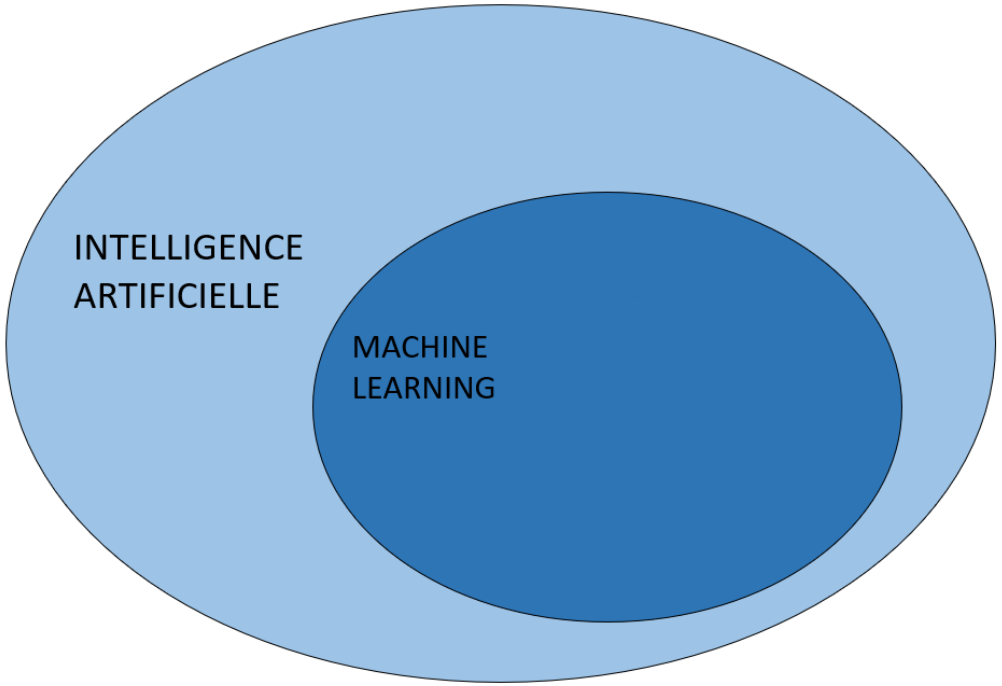
GÉNÉRALITÉS

- IA : le programmeur doit définir à la main tous les cas de figures souvent simples qui se présenteront lors de l'exécution du programme



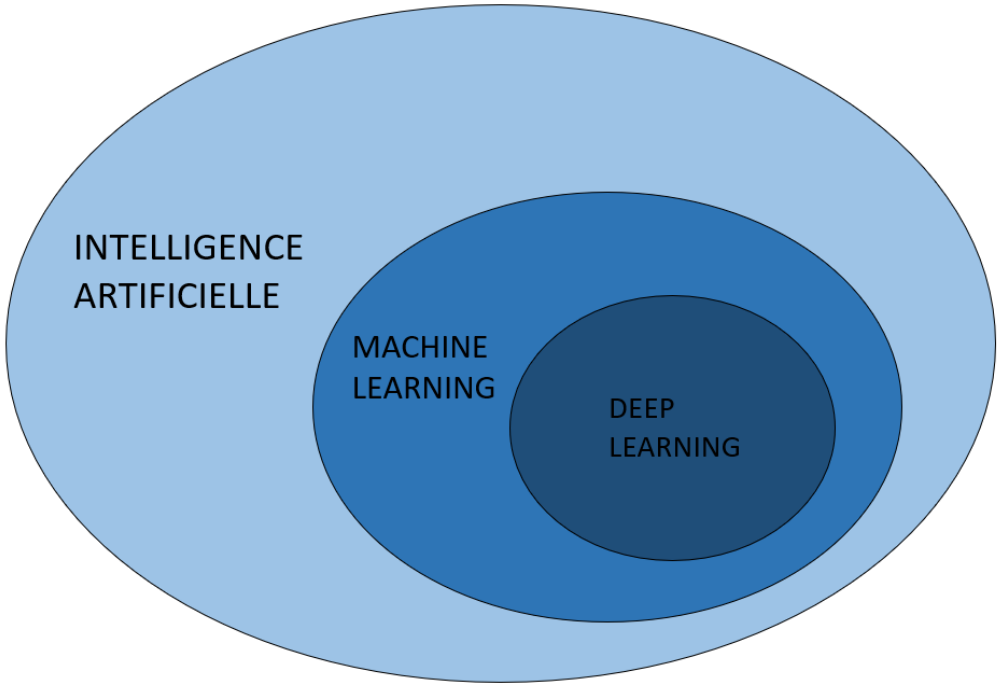
GÉNÉRALITÉS

- IA : le programmeur doit définir à la main tous les cas de figures souvent simples qui se présenteront lors de l'exécution du programme
- ML : type d'IA qui, grâce à des données d'entraînement complexes souvent prétraitées, va apprendre à reconnaître ces cas de figures



GÉNÉRALITÉS

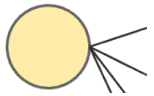
- IA : le programmeur doit définir à la main tous les cas de figures souvent simples qui se présenteront lors de l'exécution du programme
- ML : type d'IA qui, grâce à des données d'entraînement complexes souvent prétraitées, va apprendre à reconnaître ces cas de figures
- DL : type de ML qui apprend à réagir à des situations encore plus complexe avec des données peu voire pas traitées en amont



GÉNÉRALITÉS

Représentation d'un réseau de neurone

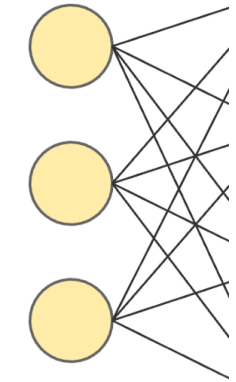
- Cellule : fonction qui applique des opérations basiques à une entrée donnée



GÉNÉRALITÉS

Représentation d'un réseau de neurone

- Cellule
- Couche (layer) : mise en parallèle de cellules



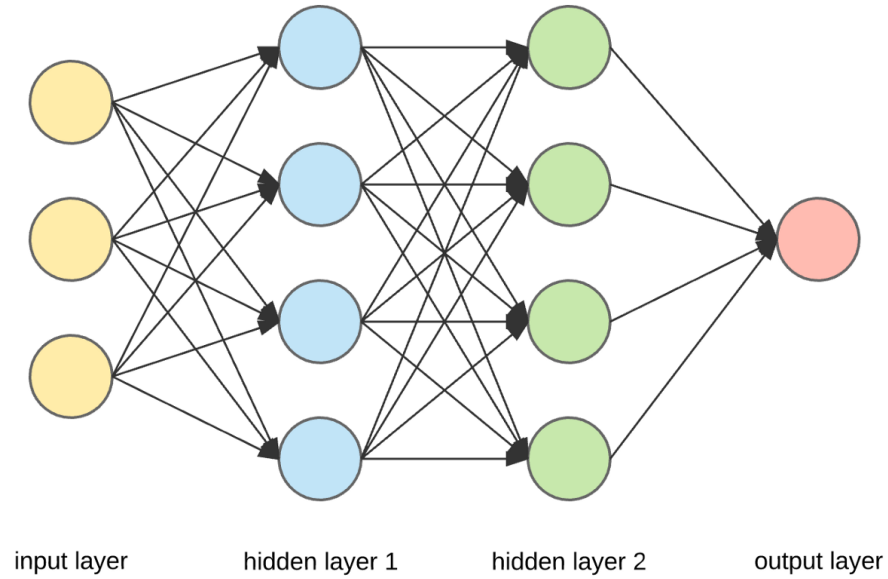
input layer

GÉNÉRALITÉS

Représentation d'un réseau de neurone

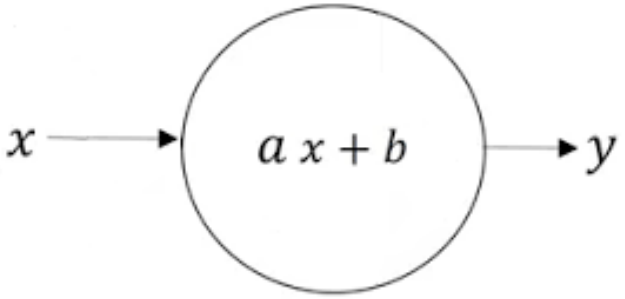
- Cellule
- Couche (layer)
- Réseau de neurones : mise à la suite de couches

L'objectif initial de cette architecture était d'imiter celle du cerveau humain

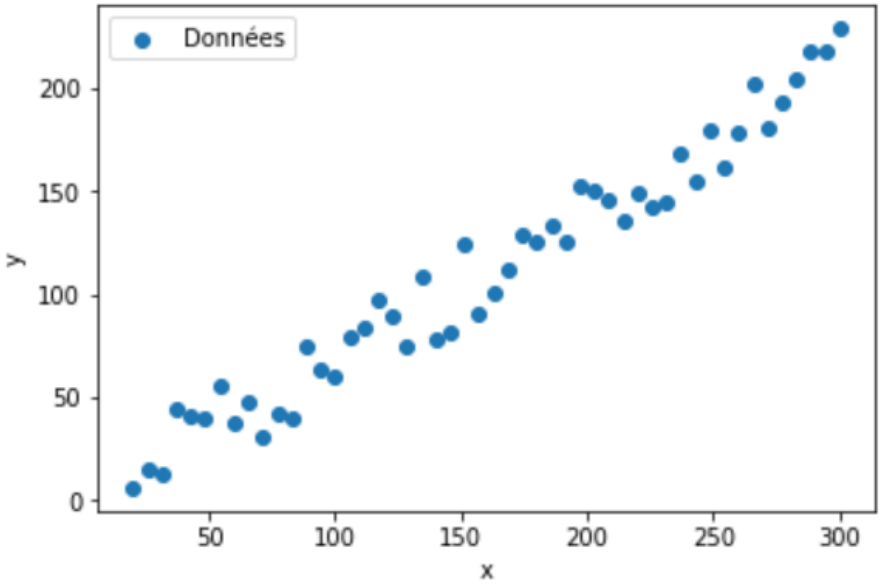


RÉGRESSION LINÉAIRE

- Étude de l'entraînement d'une cellule simple
- Cas particulier : $b = 0$



- Entraînement de cette cellule pour déterminer la meilleure relation entre x et y
- Pente de la droite : 0.7



RÉGRESSION LINÉAIRE

- Initialisation de la pente

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44 
45     da = da/m
46 
47     a = a - lr*da
48 
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

- Initialisation de la pente

- Réinitialisation

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44
45     da = da/m
46
47     a = a - lr*da
48
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

- Initialisation de la pente
- Réinitialisation
- Calcul de prédiction

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44
45     da = da/m
46
47     a = a - lr*da
48
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

- Initialisation de la pente
- Réinitialisation
- Calcul de prédiction
- Calcul de l'erreur

$$Erreur = \sum_{i=1}^{m} (y_i - p_i)^2$$

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44
45     da = da/m
46
47     a = a - lr*da
48
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

- Initialisation de la pente
- Réinitialisation
- Calcul de prédiction
- Calcul de l'erreur
- Ajustement de la pente prédite

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44
45     da = da/m
46
47     a = a - lr*da
48
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

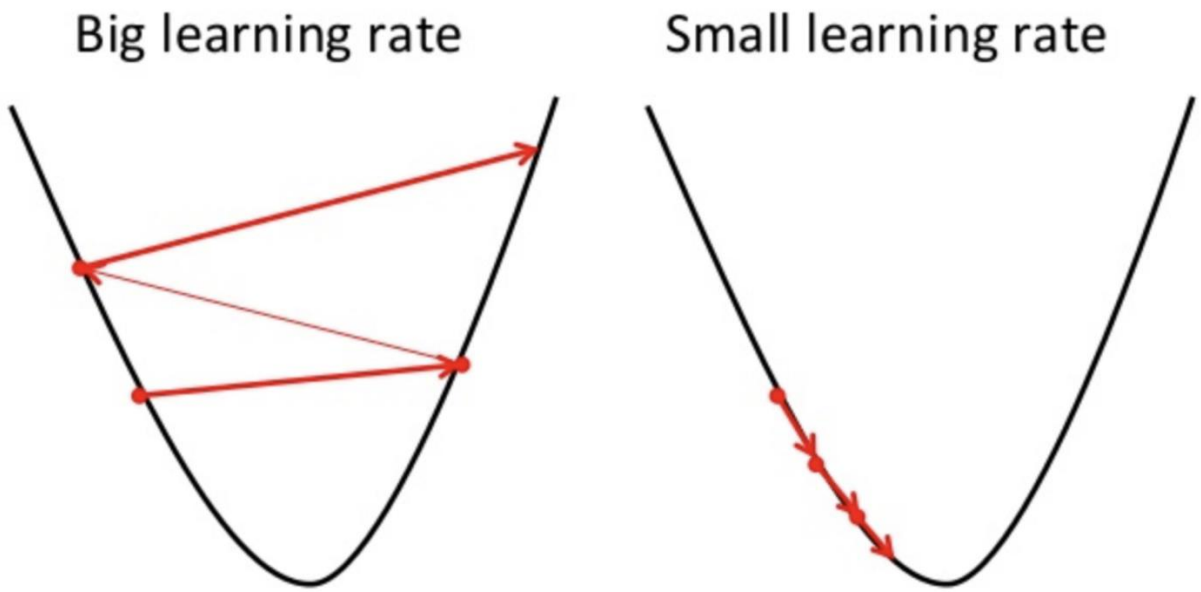
- Initialisation de la pente
- Réinitialisation
- Calcul de prédiction
- Calcul de l'erreur
- Ajustement de la pente prédite
- Ramener à l'échelle d'un exemple

```
25 for iteration in range(iterations):
26     print("iteration :", iteration)
27     Erreur = 0
28     da = 0
29     for i in range(m):
30         # Calcul de la prédiction
31         p = a*x[i]
38         # Calcul de l'erreur
39         Erreur += (p - y[i])**2
40         # Dérivée de l'erreur par rapport à p
41         dp = 2*(p - y[i])
42         # Dérivée de l'erreur par rapport à a
43         da += dp*x[i] # chain rule
44
45     da = da/m
46
47     a = a - lr*da
48
49     Erreur = Erreur/(2*m)
```

RÉGRESSION LINÉAIRE

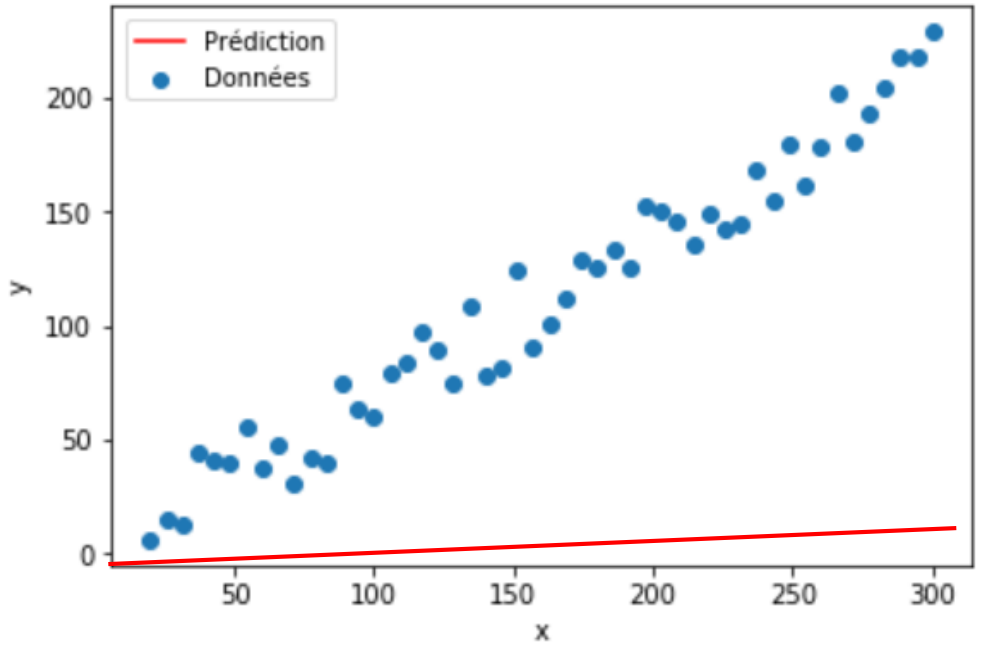
Learning rate

- Paramètre simple qui participe pourtant beaucoup à l'efficacité des réseaux de neurones au travers de leur entraînement
- Calcul de minimisation de l'erreur : permet de converger plus certainement vers le minimum de la fonction d'erreur



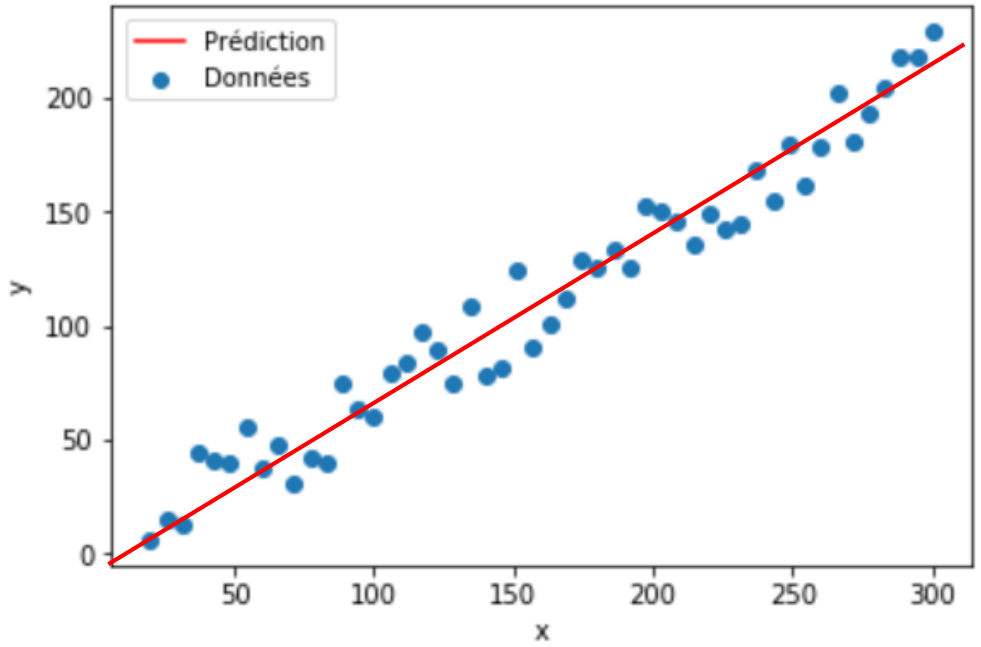
RÉGRESSION LINÉAIRE

- Résultats



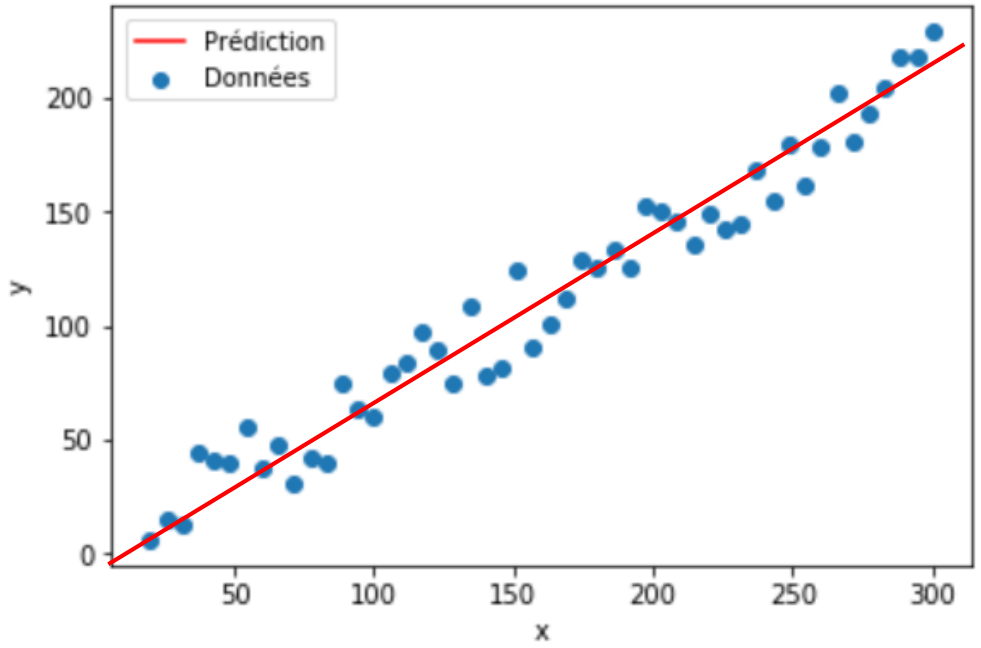
RÉGRESSION LINÉAIRE

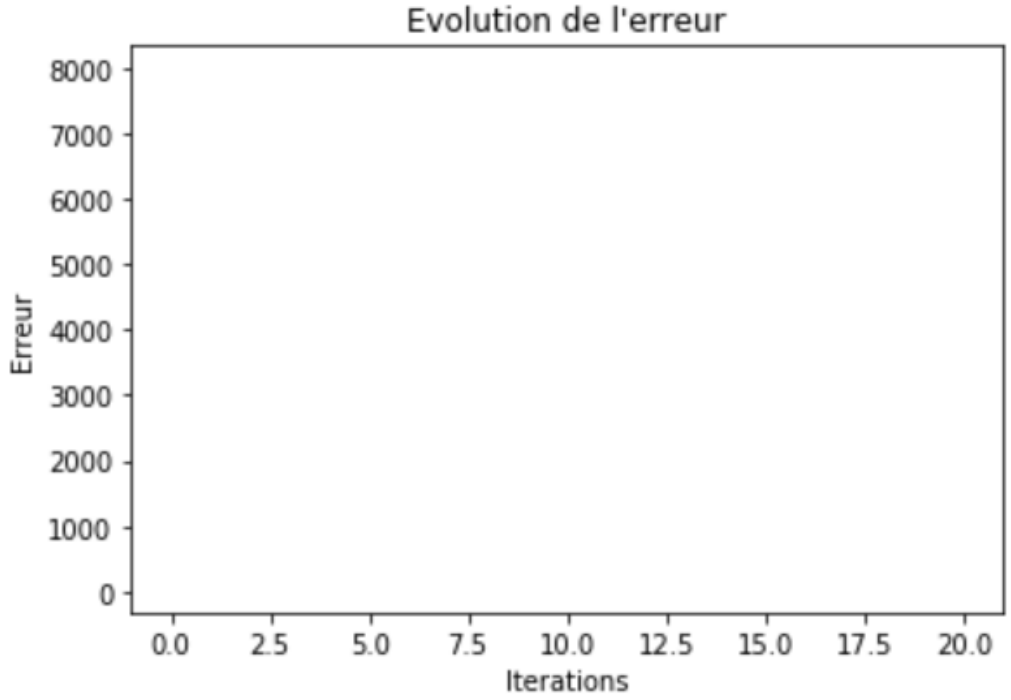
- Résultats



RÉGRESSION LINÉAIRE

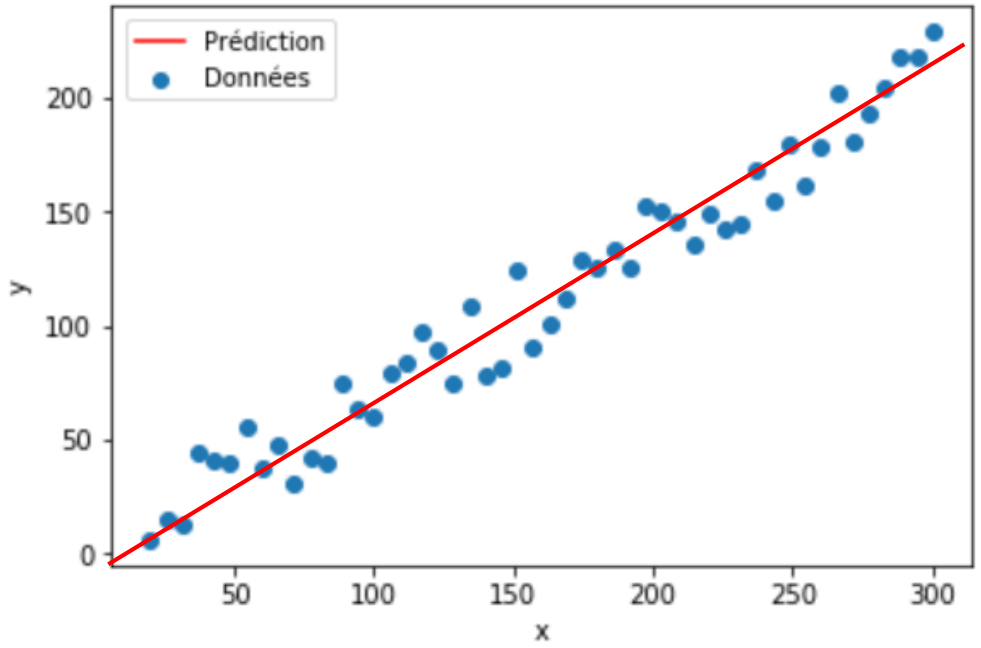
- Résultats

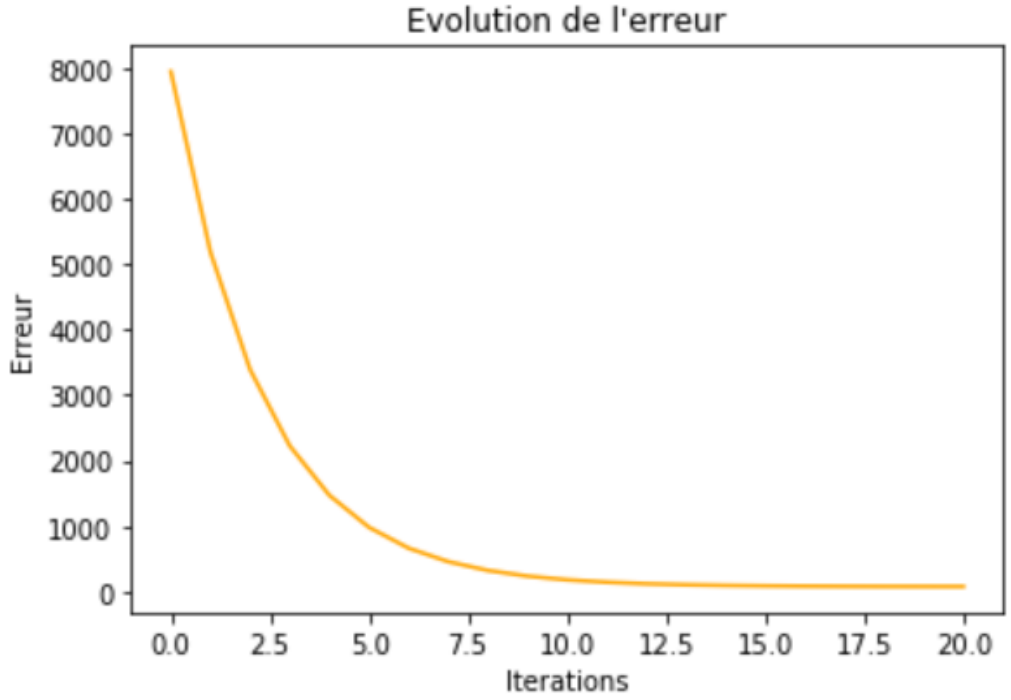




RÉGRESSION LINÉAIRE

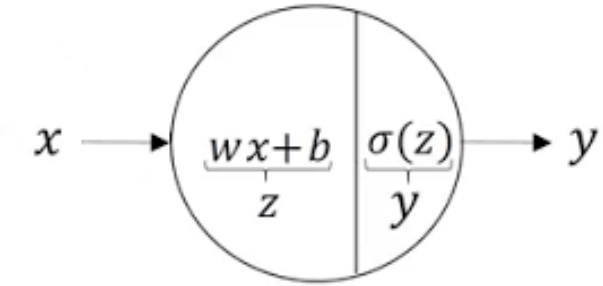
- Résultats



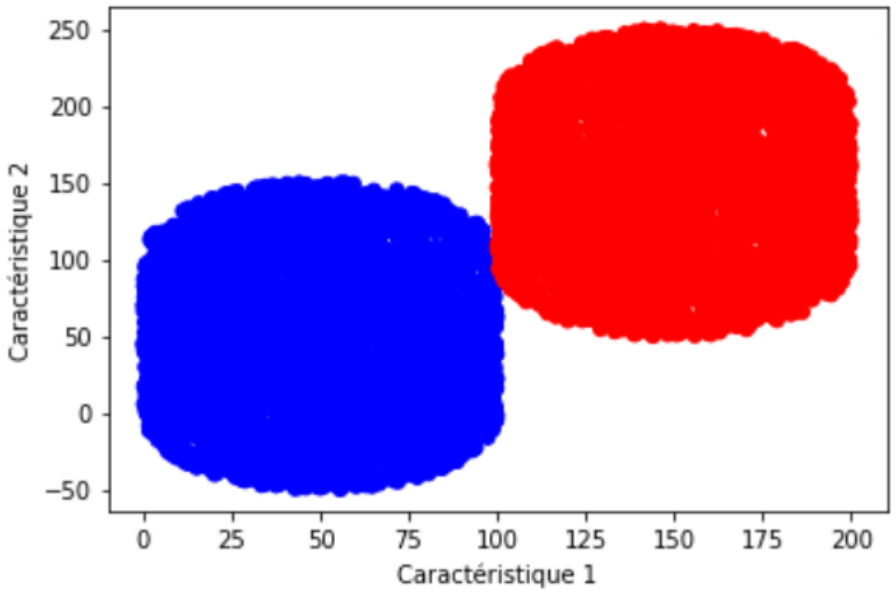


RÉGRESSION LOGISTIQUE

- Autre type de cellule simple
- Nouvelle situation avec classification de groupes



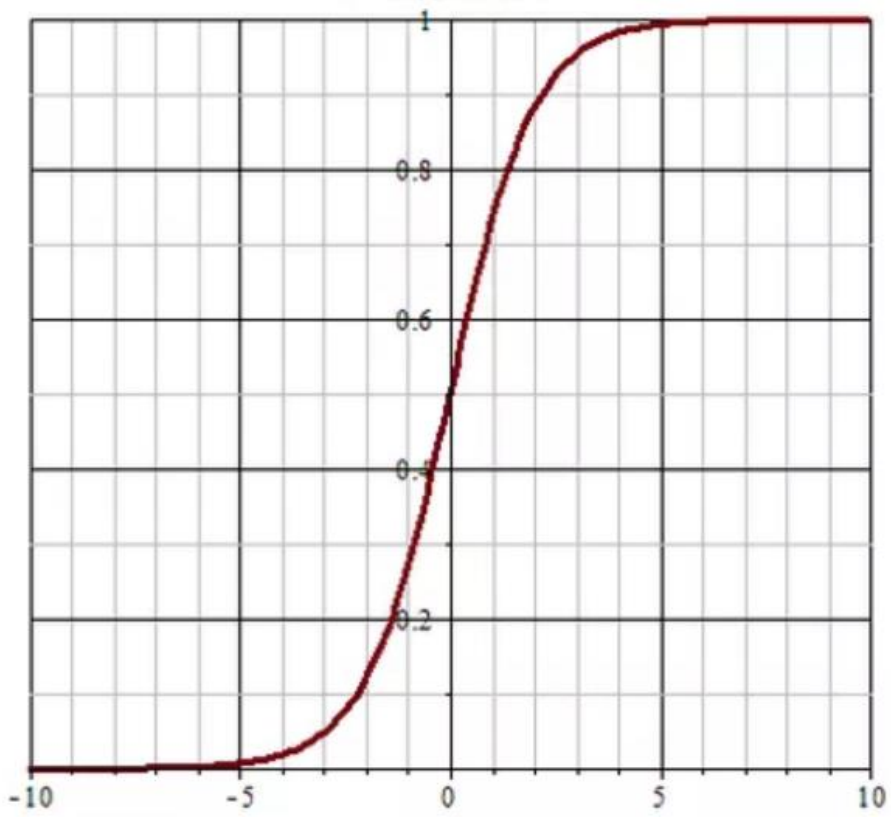
- Entraînement de la cellule pour déterminer la meilleure droite qui sépare les deux groupes



RÉGRESSION LOGISTIQUE

- Fonction d'activation « sigmoïde »
- Permet la classification d'éléments nouveaux dans un des deux groupes

$$\sigma : \left(\begin{array}{ccc} \mathbb{R} & \rightarrow & [0,1] \\ x & \mapsto & \frac{1}{1+e^{-x}} \end{array} \right)$$



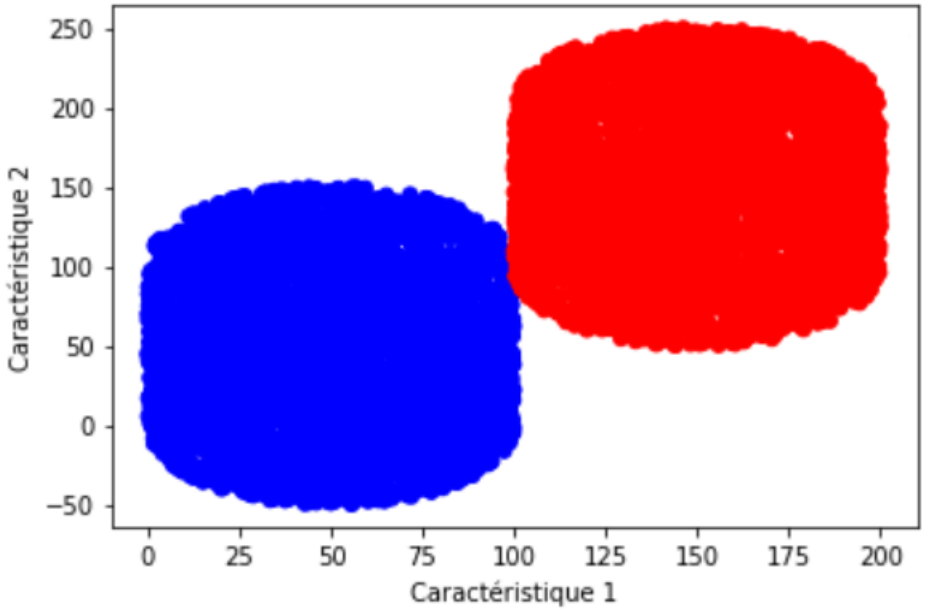
RÉGRESSION LOGISTIQUE

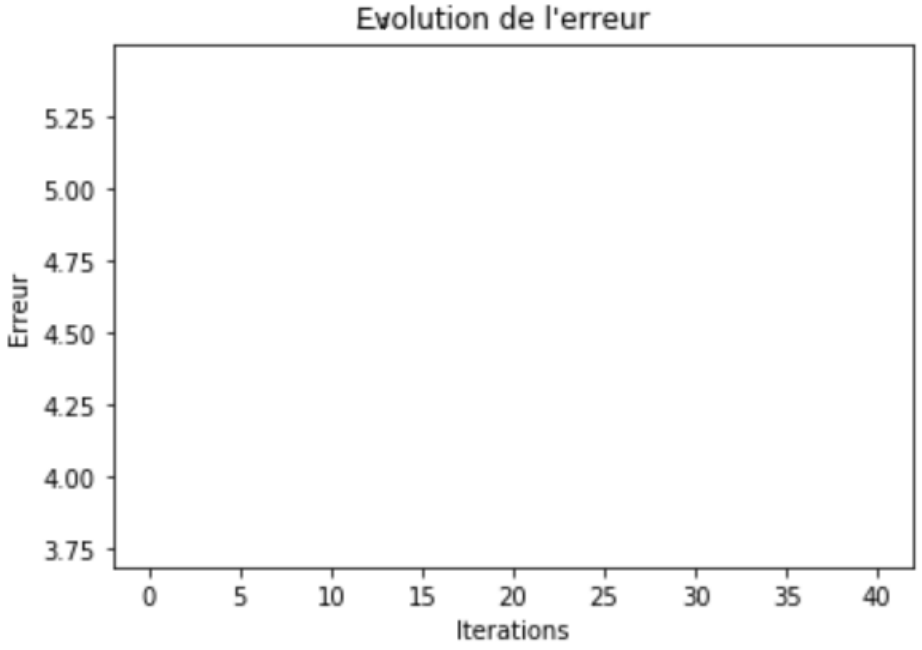
- Structure similaire à la régression linéaire pour les étapes de l'entraînement
- Fonction de l'erreur adaptée au problème de classification

$$Erreur = -\sum_{i=1}^{m} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

RÉGRESSION LOGISTIQUE

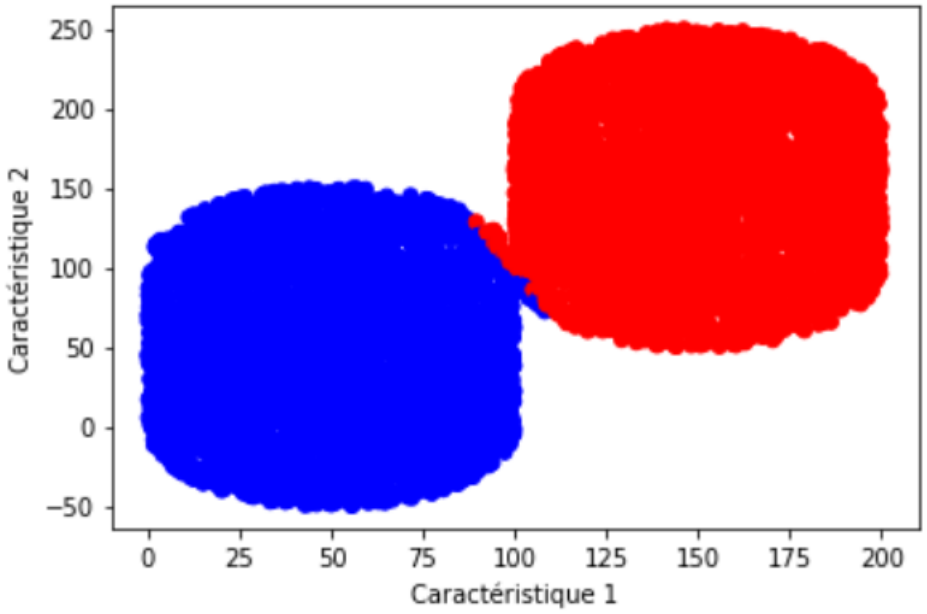
- Résultats

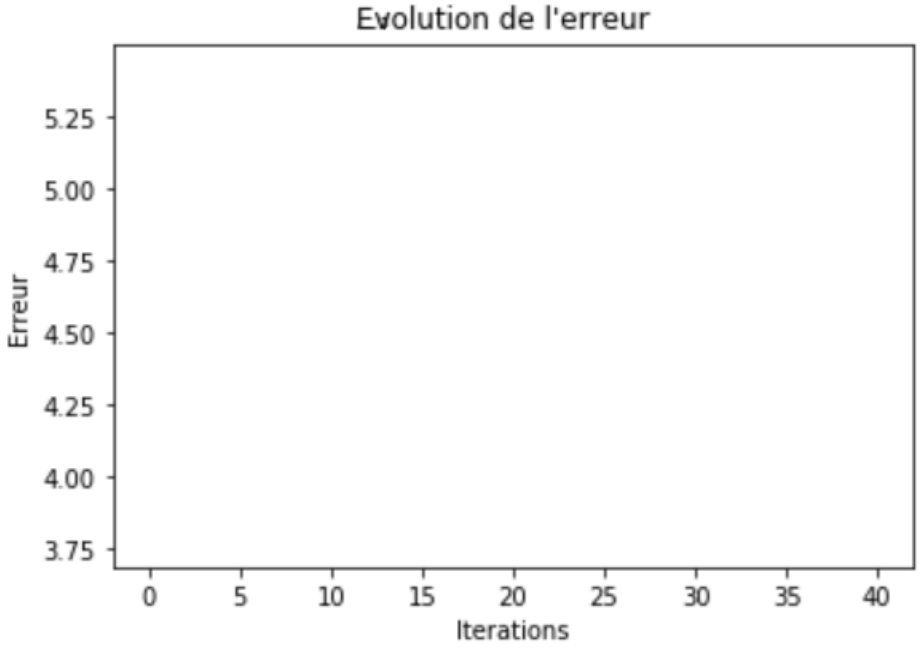




RÉGRESSION LOGISTIQUE

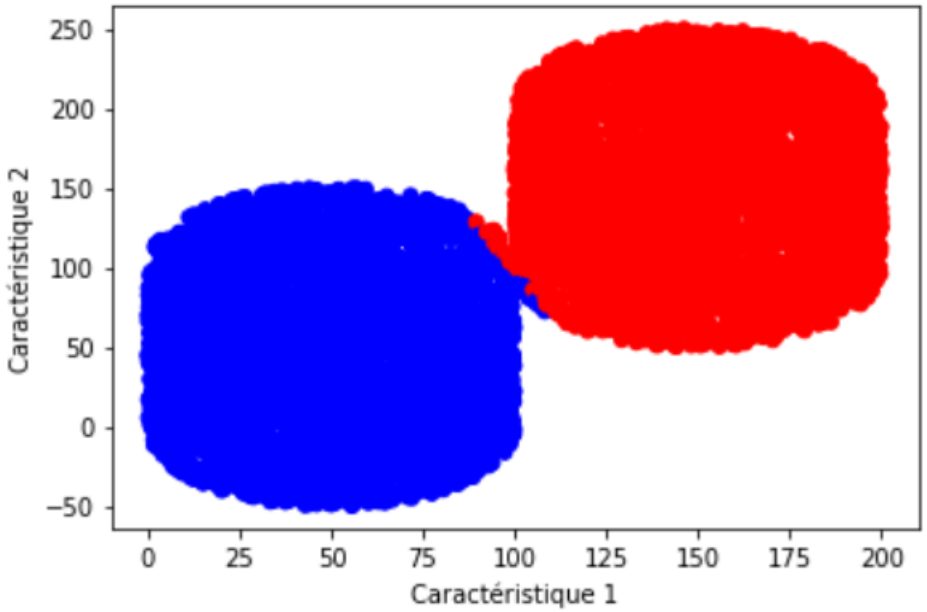
- Résultats

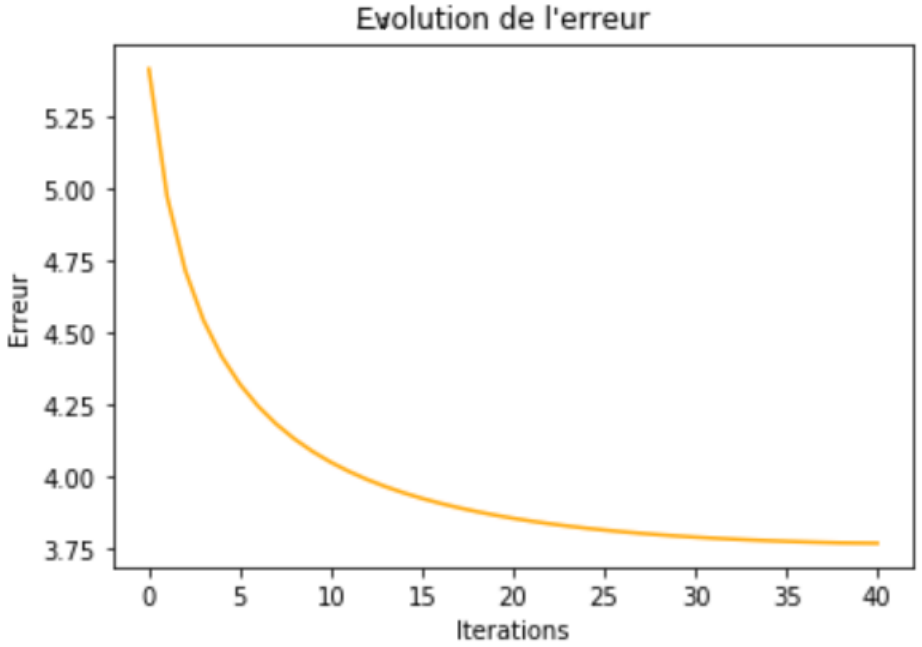




RÉGRESSION LOGISTIQUE

- Résultats





FILTRES

Convolution

- Principe de la convolution discrète : f et g deux fonctions définies de manière discrète alors on a h la fonction convolution des deux définie par :

$$h(i) = (f * g)(i) = \sum_{k=-\infty}^{+\infty} f_{i-k} g_k$$

- Intérêt pour le traitement d'images dans les réseaux de neurones

FILTRES

Convolution

- Cas simple : deux fonctions discrètes de une variable

f :	10	50	60	10	20	40	30

g :	1/3	1/3	1/3

FILTRES

Convolution

f :

10	50	60	10	20	40	30

g :

1/3	1/3	1/3

$$h(2) = \frac{50 + 60 + 10}{3} = 40$$

FILTRES

Convolution

f :

10	50	60	10	20	40	30

g :

1/3	1/3	1/3

h :

40	40	30	20	30

FILTRES

Principe

- Convolution sur des images considérées comme des tableaux de valeurs
- Même principe que le cas à une variable et à la fin de la ligne on recommence au début de la suivante en appliquant le masque à chaque étape

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

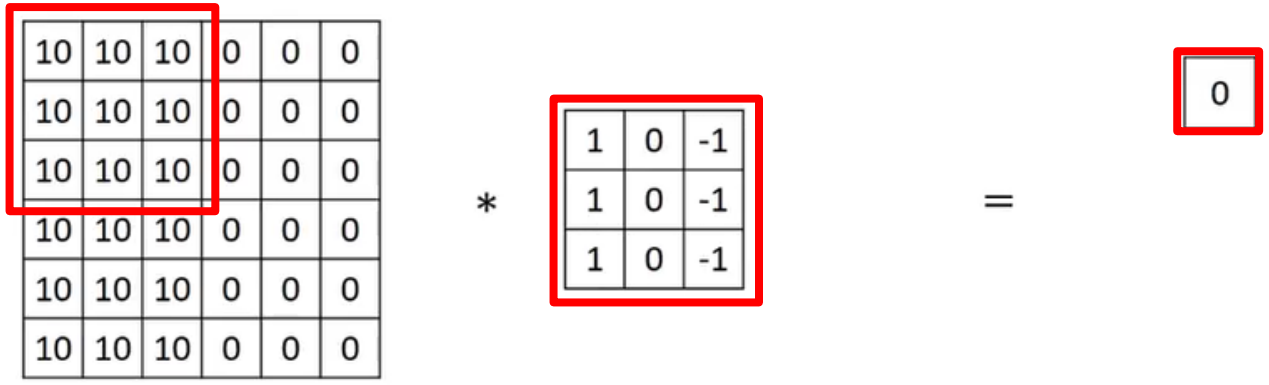
*

1	0	-1
1	0	-1
1	0	-1

FILTRES

Principe

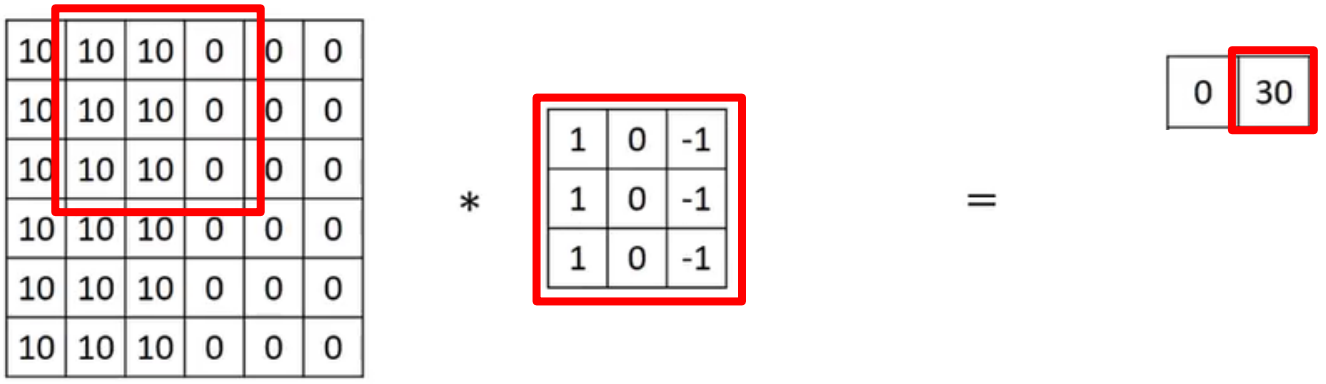
- Convolution sur des images considérées comme des tableaux de valeurs
- Même principe que le cas à une variable et à la fin de la ligne on recommence au début de la suivante en appliquant le masque à chaque étape



FILTRES

Principe

- Convolution sur des images considérées comme des tableaux de valeurs
- Même principe que le cas à une variable et à la fin de la ligne on recommence au début de la suivante en appliquant le masque à chaque étape



FILTRES

Principe

- Convolution sur des images considérées comme des tableaux de valeurs
- Même principe que le cas à une variable et à la fin de la ligne on recommence au début de la suivante en appliquant le masque à chaque étape

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

1	0	-1
1	0	-1
1	0	-1

$=$

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

FILTRES

Principe

- Filtre dit de « Prewitt »
- Objectif de détecter les contours d'une image en utilisant la différence des valeurs des pixels entre deux couches

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

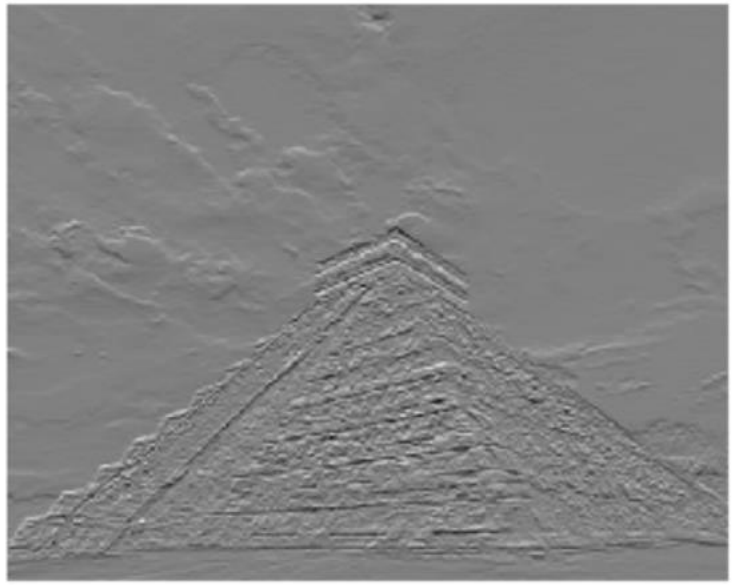
=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

FILTRES

Principe

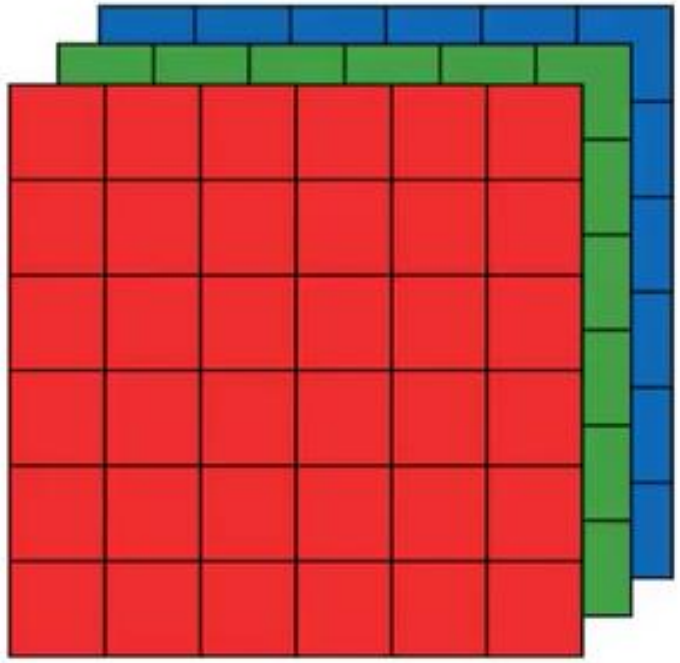
- Utilisation du filtre pour la détection de contour sur une image en nuances de gris de taille 488x390 pixels



FILTRES

Généralisation

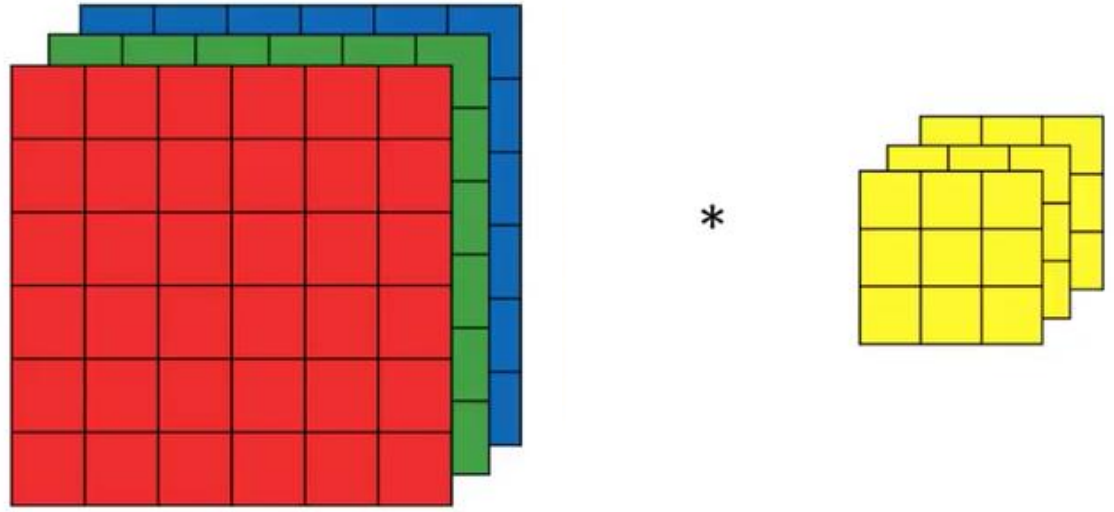
- Images avec trois couches : RGB



FILTRES

Généralisation

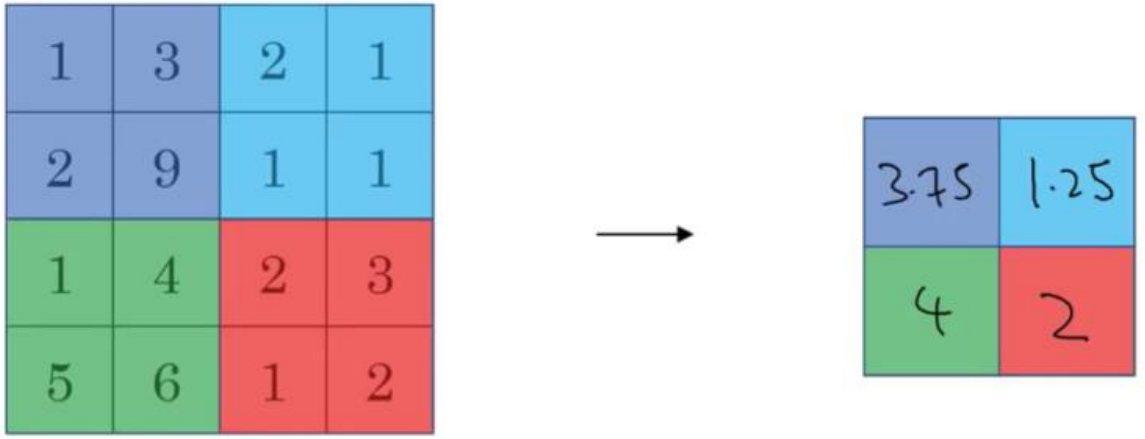
- Images avec trois couches ou à volume : représentation RGB
- Filtres adaptés pour ce genre de situations, on parle de convolution sur volume



FILTRES

Filtres de regroupement

- Intérêt dans le traitement lourd d'images effectué par un réseau de neurones
- Conservation de l'information et diminution du nombre de données
- Différents types : minimum, moyenne, maximum des pixels dans le filtre



FILTRES

Filtres de regroupement

- Aperçu sur une figure représentative pour un filtre de moyenne
- Réduction de la taille de $488x390$ à $422x340$ mais on distingue toujours le contenu de l'image

ADAPTATION AU PROBLÈME

ADAPTATION AU PROBLÈME

- Objectif du projet : à l'aide d'une caméra embarquée à l'avant, un programme d'intelligence artificielle se charge d'analyser la situation pour modifier ou non la situation de la voiture

ADAPTATION AU PROBLÈME

- Objectif du projet : à l'aide d'une caméra embarquée à l'avant un programme d'intelligence artificielle se charge d'analyser la situation pour modifier ou non la situation de la voiture

- Il faut mettre en place un montage capable de répondre à toutes les attentes relatives au projet

UTILISATION DU RASPBERRY

Le Raspberry

- Mini ordinateur : possibilité d'utiliser une caméra et Python
- Envoi de commandes à la carte Arduino pour modifier les moteurs

CRÉATION DES DONNÉES

Données d'entraînement

- Utilisation du module radio Arduino pour commander la voiture à distance
- Récupération d'images issues de la caméra pour l'entraînement du réseau de neurones
- Adaptation de ces images à l'entrée du programme en taille $224x224$

CRÉATION DES DONNÉES

Données d’entraînement

LE RÉSEAU DE NEURONES

Le programme

- Différentes architectures préexistantes, il suffit donc de choisir celle qui est la plus adaptée au problème
- Intérêt particulier pour les réseaux de neurones qui doivent analyser les images tout en ayant un temps d'exécution faible pour avoir une conduite fluide

LE RÉSEAU DE NEURONES

Le programme

- Différentes architectures préexistantes, il suffit donc de choisir celle qui est la plus adaptée au problème
- Intérêt particulier pour les réseaux de neurones qui doivent analyser les images tout en ayant un temps d'exécution faible pour avoir une conduite fluide

- Utilisation du **« MobileNet »** dans sa deuxième version

LE RÉSEAU DE NEURONES

MobileNet V2

- Intérêt principal dans son temps d'exécution pour le traitement d'images
- Convolution différente de celle standard : « depthwise » suivie de « pointwise » convolutions
- Différence qui réduit le nombre d'opérations à effectuer donc le temps d 'exécution au total

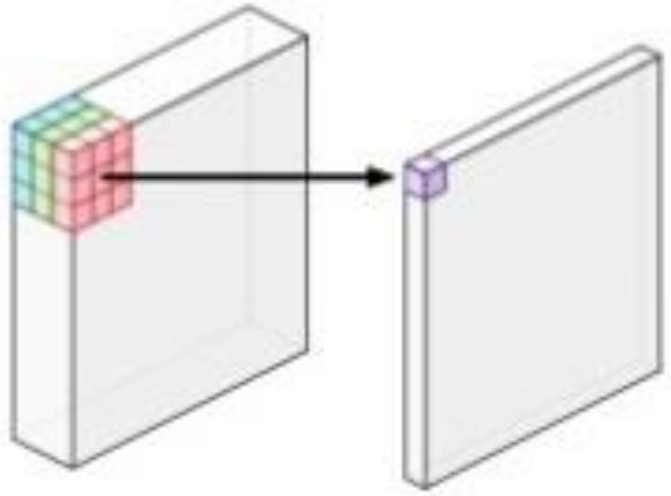
LE RÉSEAU DE NEURONES

MobileNet V2

- Intérêt principal dans son temps d'exécution pour le traitement d'images
- Convolution différente de celle standard

Convolution standard
Nombre d'opérations pour un tableau de taille $D_F \text{x} D_F \text{x} M$
convolution avec N filtres de taille $D_K \text{x} D_K \text{x} M$

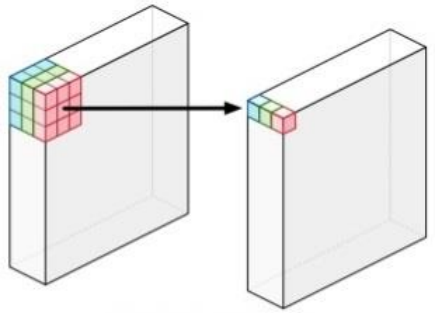
$$D_K^2 \times D_F^2 \times M \times N$$



LE RÉSEAU DE NEURONES

MobileNet V2

- Intérêt principal dans son temps d'exécution pour le traitement d'images
- Convolution différente de celle standard



Depthwise convolution

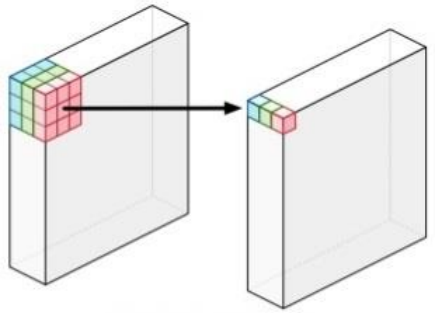
Convolution MobileNet
Nombre d'opérations pour un tableau de taille $D_F \text{x} D_F \text{x} M$
convolution avec N filtres de taille $D_K \text{x} D_K \text{x} M$

$$D_K^2 \times D_F^2 \times M$$

LE RÉSEAU DE NEURONES

MobileNet V2

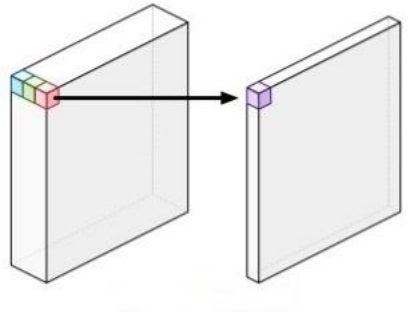
- Intérêt principal dans son temps d'exécution pour le traitement d'images
- Convolution différente de celle standard



Convolution MobileNet
Nombre d'opérations pour un tableau de taille $D_F \text{x} D_F \text{x} M$
convolution avec N filtres de taille $D_K \text{x} D_K \text{x} M$

$$D_K^2 \times D_F^2 \times M + D_F^2 \times M \times N$$

Pointwise convolution



LE RÉSEAU DE NEURONES

MobileNet V2

- Intérêt principal dans son temps d'exécution pour le traitement d'images
- Convolution différente de celle standard

- Rapport fonction du nombre de filtres et de la taille du tableau d'entrée

$$\frac{D_K^2 \times D_F^2 \times M + D_F^2 \times M \times N}{D_K^2 \times D_F^2 \times M \times N} = \frac{1}{N} + \frac{1}{D_K^2}$$

ENTRAÎNEMENT

Descente de gradient

- Vecteur des dérivées partielles d'une fonction évaluée en un point noté ∇f (a)
- Propriétés remarquables :
  1. ∇f (a) est normal à la tangente de la courbe de f en a
  2. ∇f (a) est orienté vers les valeurs positives de f

ENTRAÎNEMENT

Descente de gradient

- Propriétés remarquables :
  1. ∇f (a) est normal à la tangente de la courbe de f en a
  2. ∇f (a) est orienté vers les valeurs positives de f
- Utile lors de la minimisation de l'erreur pendant l'entraînement du réseau de neurones

ENTRAÎNEMENT

Descente de gradient

- Propriétés remarquables :
  1. ∇f (a) est normal à la tangente de la courbe de f en a
  2. ∇f (a) est orienté vers les valeurs positives de f
- Utile lors de la minimisation de l'erreur pendant l'entraînement du réseau de neurones

ENTRAÎNEMENT

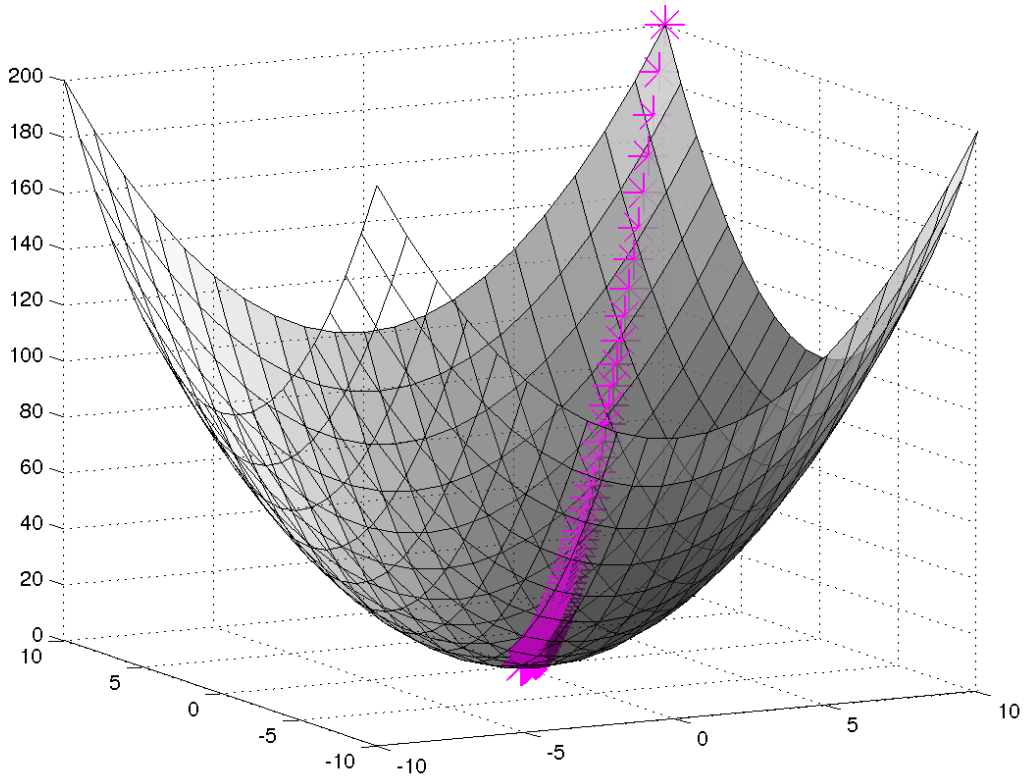
Descente de gradient

1. On prend a_0 quelconque sur la courbe de f
2. On choisit un pas λ équivalent du « learning rate »
3. On crée une suite (a_n) telle que $\forall n \in \mathbb{N} a_n = a_{n-1} - \lambda \cdot \nabla f(a_{n-1})$

```
a = a - lr*da
```

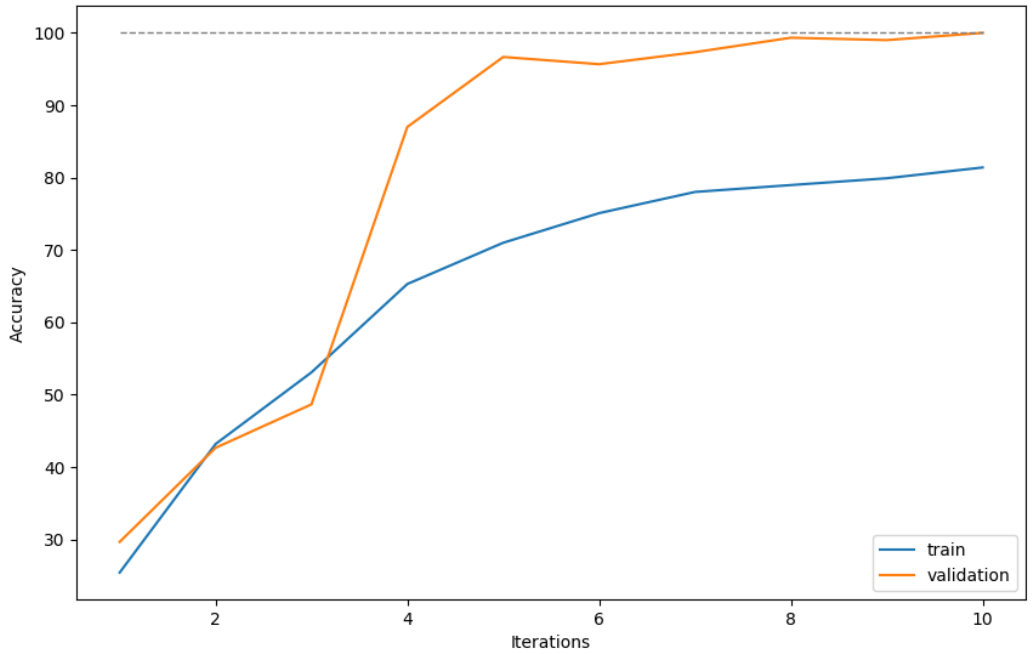
ENTRAÎNEMENT

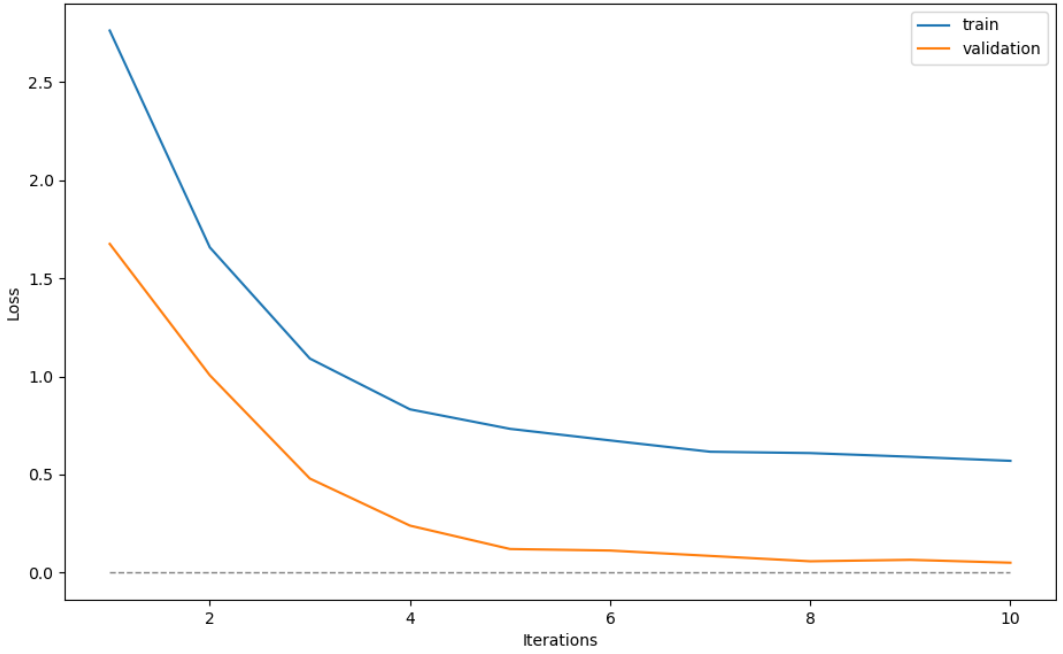
Descente de gradient



CONCLUSION

RÉSULTATS


Accuracy
Iterations
train
validation


Loss
Iterations
train
validation