QUELQUES CONCEPTS DE L'INTELLIGENCE ARTIFICIELLE

tsi.tuxfamily.org/club.html

AJOUTER NUMERO DE pAGE

Sommaire

Première partie

Apprentissage supervisé

I) Les réseaux de neurones

1) Pile ou Face

Règle n° .....

Pièce 1	Pièce 2	Verdict

Exercice 1 *On cherche à programmer un réseau de neurones pour la règle 1 :*

- *En entrée 2 neurones* p_1 *et* $p_2 \in \{0;1\}$ *(*$0 =$ *pile et* $1 =$ *face)*
- *En sortie : gagné* (1) *ou perdu* (0).

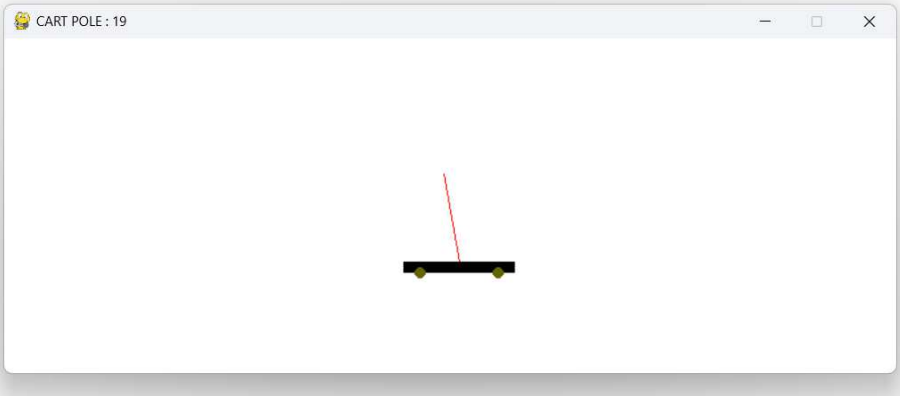
1. *Aller sur le site du club info et recopier et compléter le tableau ci-dessus.*
2. *Déclarer la fonction de Heaviside définie sur* $\mathbb{R}$ *par :* $\mathscr{H}(x) = \begin{cases} 0 \text{ si } x \leq 0 \\ 1 \text{ sinon.} \end{cases}$
3. *Écrire une fonction* `rezo1(p1, p2, a, b)` *qui reçoit en paramètre l'état* `p1` *et* `p2` *des 2 pièces, les coefficients* `a` *et* `b` *et renvoie la valeur de* $\mathscr{H}(a \times p_1 + b \times p_2)$.
4. *Écrire une fonction* `test(a,b)` *qui reçoit les coefficients* `a` *et* `b` *et renvoie le taux de réussite de ce réseau sur les 4 possibilités*
5. *En cherchant aléatoirement, existe-il des valeurs* `a` *et* `b` *pour atteindre les 100% ?*

Exercice 2 *Reprendre l'exercice avec la règle 2. On pourra faire un script qui test 10 000 valeurs de couples* $(a;b)$ *et affiche les meilleurs scores obtenus*

Exercice 3 *Reprendre l'exercice avec la règle 3*

2) Acrocirque !

La librairie **gym** (`https://www.gymlibrary.dev/`) permet d'utiliser des petits défis déjà programmés pour tester vos intelligences artificielles. Commençons par un premier défi facile si vous avez l'équilibre, il s'agit de tenir en équilibre une baguette sur une planche de skate :



Exercice 4 *Allez sur le site du club info pour télécharger la version jouable du défi et tentez votre chance... Vous verrez que ce n'est pas si simple car il y a un peu d'inertie dans la machine ! Vous pouvez ralentir le jeu en modifiant la ligne :*

```
clock.tick(10) # Nombre d'images par secondes
```

Lors de la partie, un certain nombre d'informations peuvent être récupérées lorsque l'on effectue une action :

```
obs, reward, done, timeout, info = env.step(action)
```

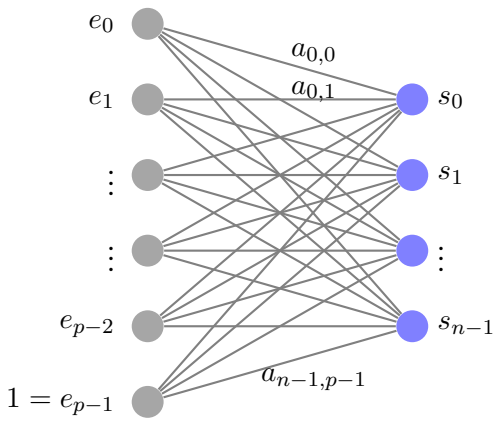
- L'`action` réalisée est codée par un entier pouvant prendre 2 valeurs :
  - ◇ 0 : accélérer vers la gauche
  - ◇ 1 : accélérer vers la droite

Les informations retournées :

- `obs` est un tableau contenant 4 flottants :
  - ◇ `obs[0]` : est la position (sur l'axe des abscisses) de la voiture. La partie est perdue si on quitte l'intervalle $[-2{,}4; 2{,}4]$.
  - ◇ `obs[1]` : est la vitesse (algébrique) de la voiture.
  - ◇ `obs[2]` : est l'angle (en radians) que fait le bâton avec la verticale. La partie est perdue si on quitte l'intervalle $[-0{,}2095; 0{,}2095]$ soit un décalage d'environ $12°$ de l'axe.
  - ◇ `obs[3]` : est la vitesse angulaire du bâton.
- `reward` est une "récompense" que nous n'utiliserons ici (d'ailleurs, elle vaut toujours 1 pour indiquer que l'on gagne des points à chaque tour de boucle)
- `done` est un booléen (qui vaut donc `True` ou `False` qui indique si la partie est terminée ou non. Celle-ci se termine lorsque le bâton va tomber.
- `timeout` est un booléen indiquant on est arrivé au bout du temps (partie gagnée).
- `info` ne sera pas utilisé dans ce défi.

Pour relever ce défi, nous allons utiliser un réseau de neurones. Celui-ci contient 4 neurones en entrées $[o_0, o_1, o_2, o_3]$ (les 4 observations) et en sortie 2 neurones : $[s_0, s_1]$. Vu qu'il a fallut une couche cachée pour jouer à pile ou face, on peut imaginer qu'il va aussi en falloir une aussi pour ce jeu. Plaçons donc n neurones sur la couche cachée $[c_0, c_1, ..., c_{n-1}]$.

Pour passer d'une couche à une autre, la structure est toujours la même :



Si il y a p entrées sur une couche et n sorties sur la couche suivante, on cherche à réaliser ce calcul avec $n \times p$ poids :

$$\begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \dots & \alpha_{0,p-1} \\ \alpha_{1,0} & \alpha_{1,1} & \dots & \alpha_{1,p-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{n-1,0} & \alpha_{n-1,1} & \dots & \alpha_{n-1,p-1} \end{pmatrix} \text{ et } \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{p-1} \end{pmatrix} \text{ donne : } \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{n-1} \end{pmatrix} = \begin{pmatrix} \mathscr{H}(a_{0,0}e_0 + a_{0,1}e_1 + ... + a_{0,p-1}e_{p-1}) \\ \mathscr{H}(a_{1,0}e_0 + a_{1,1}e_1 + ... + a_{1,p-1}e_{p-1}) \\ \vdots \\ \mathscr{H}(a_{n-1,0}e_0 + a_{n-1,1}e_1 + ... + a_{n-1,p-1}e_{p-1}) \end{pmatrix}$$

Pour ceux qui connaissent le module, on peut utiliser `numpy` qui est fait pour cela !

Exercice 5 *Écrire une fonction `produit(A,B)` qui reçoit une matrice `A` à p lignes et n colonnes et une liste de n valeurs et reçoit la sortie présentée ci-dessus (sans la fonction d'activation). Par exemple :*

```
>>> A = [[1,2,3],[4,5,6]]
>>> B = [3,-1,1]
>>> produit(A,B)
[4, 13]
```

La fonction `init(n, p)` suivante crée et renvoie un tableau de `n` lignes et `p` colonnes remplis de 0 :

```
def init(n,p) :
    return [[0]*p for i in range(n)]
```

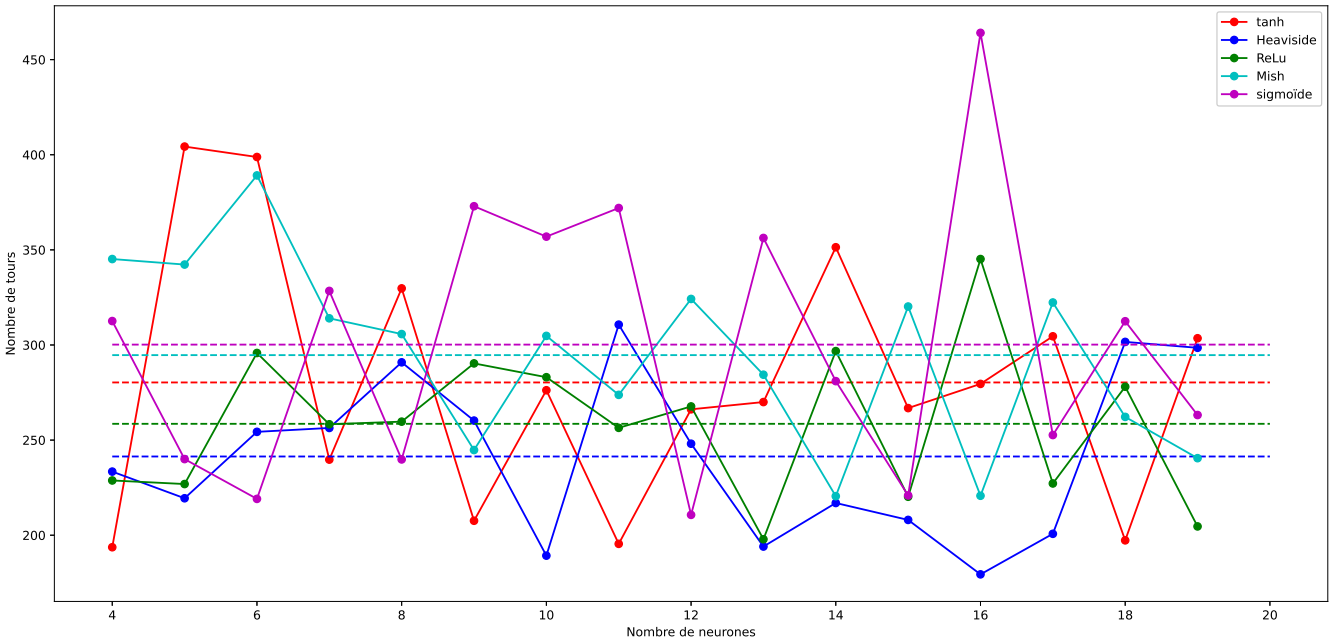
Exercice 6 *Écrire une fonction `tableau_aleatoire(n, p)` qui génère une matrice à n lignes et p colonnes avec des nombres aléatoires de l'intervalle $[-2,2]$ (on pourra utiliser la fonction `uniform` du module `random`. Par exemple :*

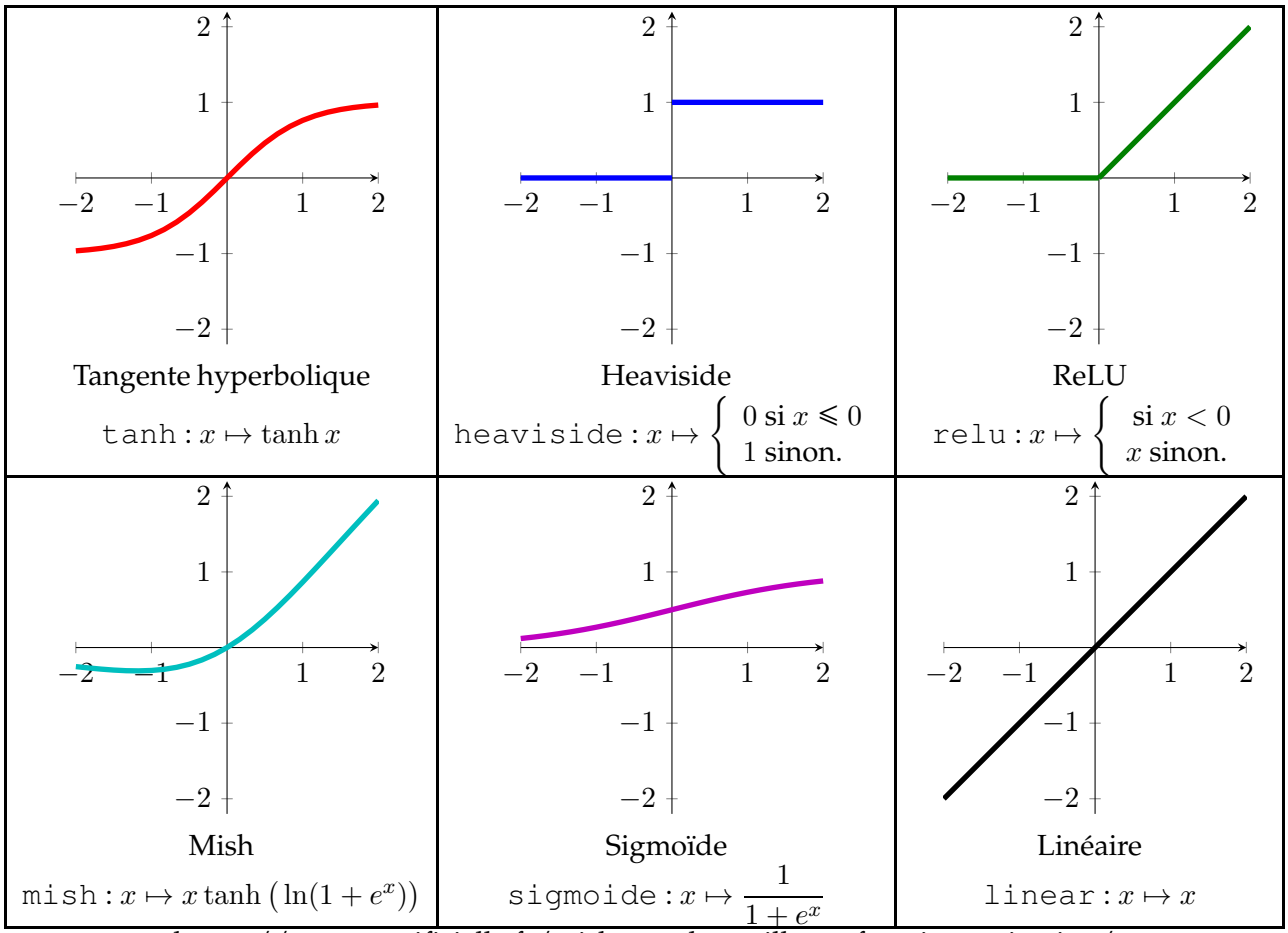
```
>>> tableau_aleatoire(3,2)
[[0.88196730162, 0.142133297005], [-0.96669320183, -1.79216445340], [-1.64652062194, 1.819393390001]]
```

Exercice 7 *Écrire une fonction `couche(entree, poids, activ)` qui reçoit une liste de valeurs d'entrée, une matrice de poids et une fonction d'activation et renvoie une liste correspondant à la sortie demandée.*

Exercice 8 *Essayez alors de résoudre le problème !*

3) Les fonctions d'activation.





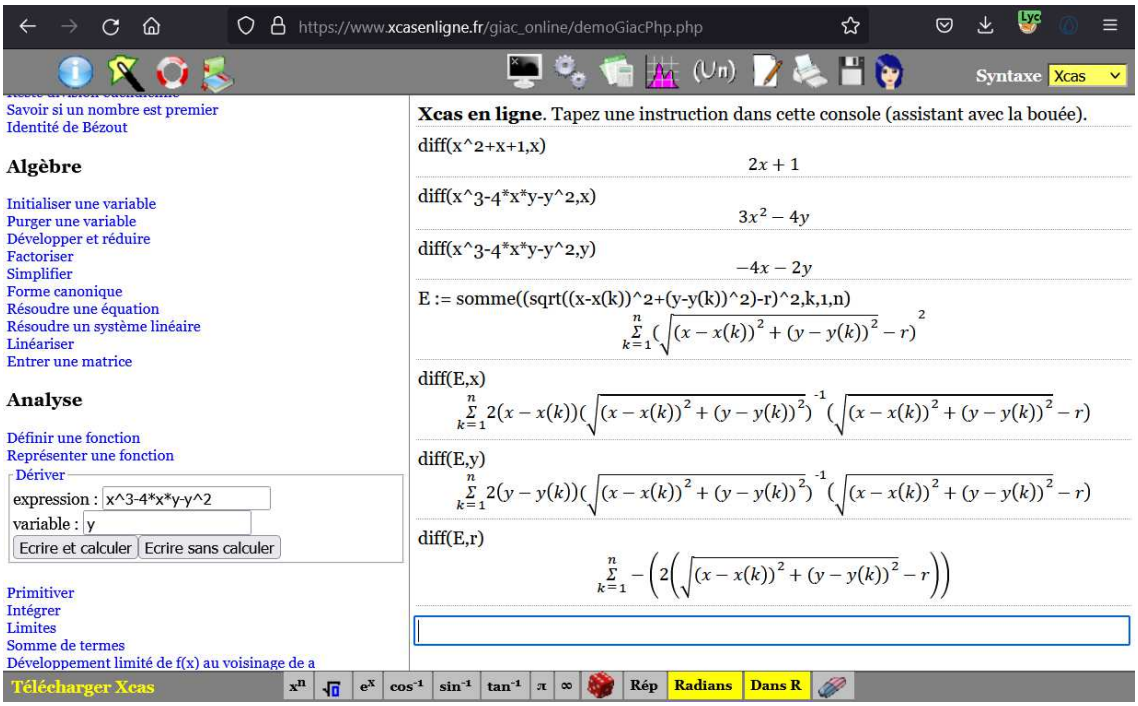
https://penseeartificielle.fr/mish-vs-relu-meilleure-fonction-activation/

Exercice 9 *Pourquoi est-il important de mettre une fonction d'activation ? Que se passe-t-il si on en met pas ? Quelques réponses sur le playground de tensorflow*

II) Descente de gradient

Pour cette partie, tu dois connaitre :

- La dérivée : le cours et les exercices d'Yvan Monka.
- Les dérivées partielles. A noter que le logiciel XCAS utilisable en propose d'effectuer les calculs pour vous !



- Sorties graphiques et Numpy :

Numpy pyplot

1) Petit mémo Numpy

Une synthèse des fonctions utilisées dans cette formation :

- `linspace(xmin, xmax, nb)` :
- `zeros(t)` :
- `shape` : d
- `T[i,j]` :
- `T[:,j]` et `T[i,:]` :
- `load`

2) Minimum d'une fonction

Exercice 10 *On considère la fonction f définie sur l'intervalle $[-3;2]$ par $f(x) = x^2 + x + 1$*

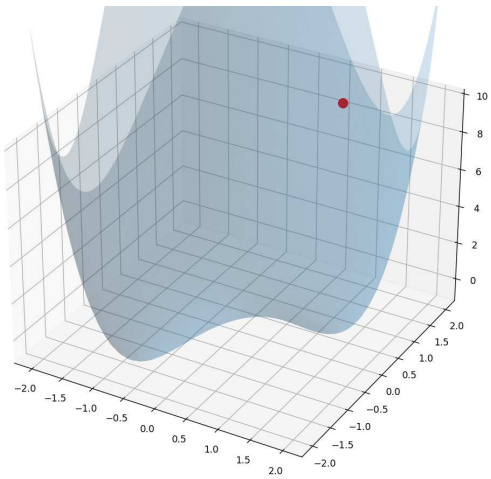
1. *Tracer la courbe de la fonction sur cet intervalle à l'aide de Python.*
2. *On tire une valeur aléatoire $x_0 \in [-3;2]$*
3. *Répéter jusqu'à ce que l'écart entre x_n et x_{n+1} soit inférieur à 10^{-4} : calculer $x_{n+1} = x_n - \alpha f'(x_n)$.*

*α est appelé **taux d'apprentissage** ou **learning rate**.*

4. *Que se passe-t-il avec $\alpha = 1$? $\alpha = 0{,}9$? $\alpha = 1{,}1$?*

Exercice 11 *On souhaite adapter l'exemple précédent à la fonction*

$$f : (x,y) \longmapsto x^4 + y^4 - 4xy$$



1. *Télécharger l'exemple sur le site et l'exécuter pour visualiser la surface représentant la fonction f.*
2. *Modifier les lignes 21 et 22 pour obtenir le résultat souhaité.*

```
# Ces 2 lignes à modifier :
x = uniform(-2,2)
y = uniform(-2,2)
```

3) Découpage d'un panneau de circulation

On dispose de photos de panneaux routiers que l'on a pré-traité : on repère les points "assez rouges", on détermine l'enveloppe convexe de cette zone rouge.

Toute cette partie a déjà été réalisée et le but de l'exercice qui suit et de déterminer le centre et le rayon du panneau de manière à ne conserver que celui-ci et supprimer le tour.


Photo


Points rouges


Enveloppe convexe


Points de l'enveloppe

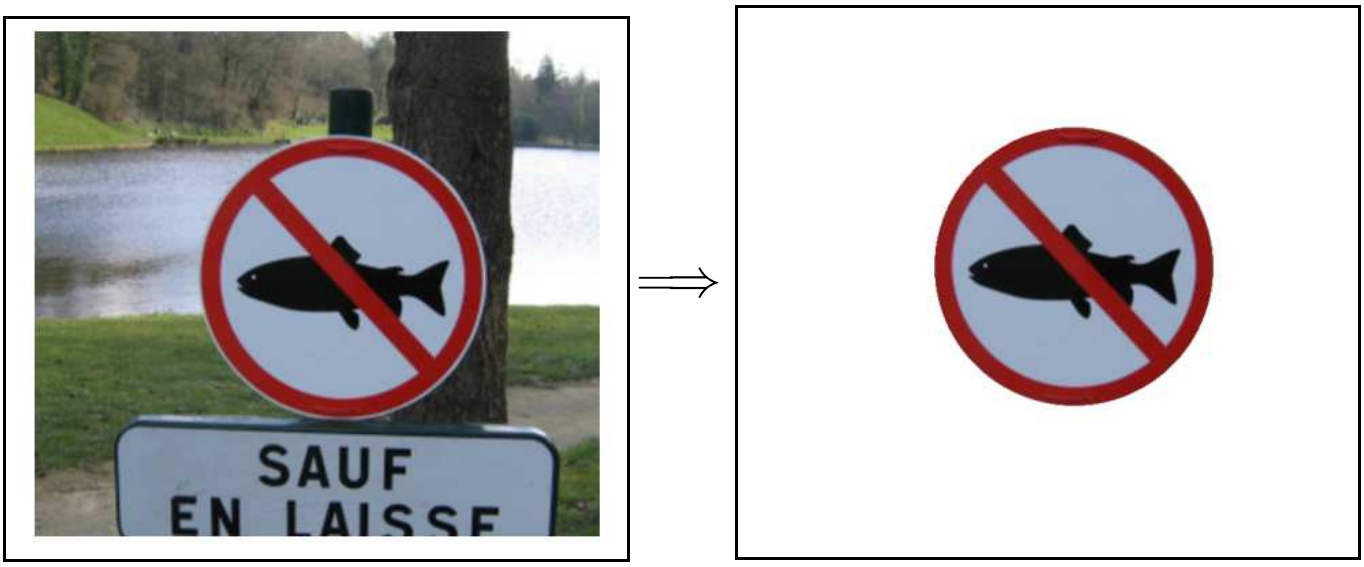
voir les animations des panneaux : pigeon poisson sens interdit pietons

Télécharger le fichier zip et le décompresser. Le dossier comprends :

- 4 photos de panneaux routiers (fichiers au format .jpg)
- 4 fichiers de tableaux numpy (fichiers au format .npy) représentant des points du contour du panneau. Les tableaux sont de taille $(2,N)$ où N est le nombre de points. La première colonne correspond aux abscisses, la seconde aux ordonnées de ces points.
- Un fichier python qui charge un panneau et l'affiche et charge le nuage de points dans une variable `envel`.

1. Compléter le programme pour tracer le nuage de points comme sur la dernière photo.

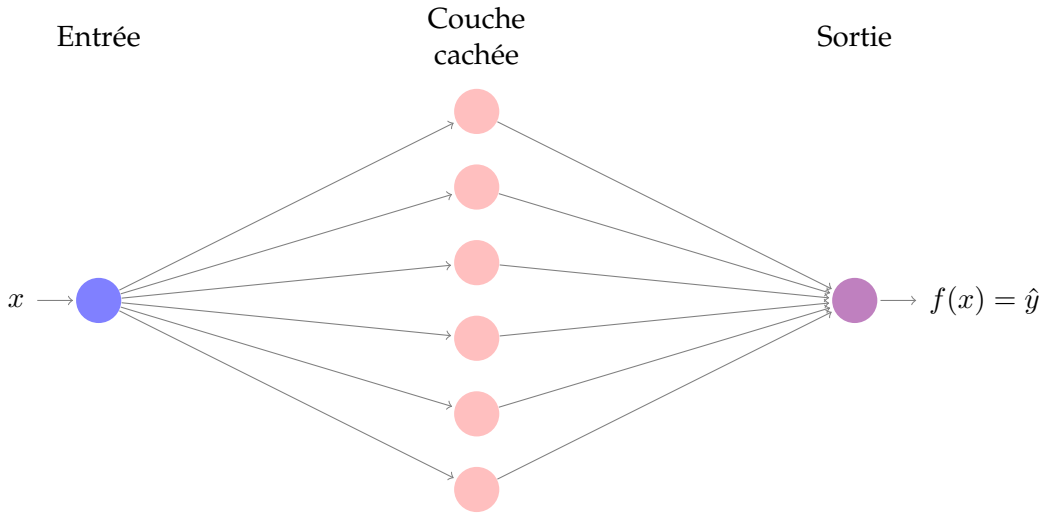
2. Trouver un candidat "pas trop mal" pour le centre et le rayon et tracer le cercle obtenu.
3. Trouver une expression possible pour une erreur $E(x,y,r)$ en fonction de x, y et r.
4. Exprimer $\frac{\partial E}{\partial x}(x,y,r)$, $\frac{\partial E}{\partial y}(x,y,r)$ et $\frac{\partial E}{\partial r}(x,y,r)$ en fonction de x, y et r (vous pouvez utiliser XCAS).
5. Appliquer alors la descente de gradient pour optimiser les valeurs de x, y et z.
6. Tracer alors le résultat obtenu. Ce nouveau cercle doit être meilleur que le premier.....
7. ★ Réaliser une animation de l'évolution du cercle.
8. ★ Que se passe-t-il si on prend (x,y,r) totalement aléatoire comme initialisation ?
9. ★ Détourer alors l'image comme voulu au départ :



4) Quel rapport avec nos réseaux de Neurones ?

Dernier exercice entièrement programmé sans module promis ! On cherche à modéliser une fonction à l'aide d'un réseau de neurones par exemple la fonction $x \longmapsto \cos(x)$ sur l'intervalle $[-3,2]$.

Pour cela, on réalise un réseau de neurones ayant cette structure :



Au niveau des notations :

- x est la valeur de l'entrée,
- on note $c_0, c_1, ..., c_{n-1}$ les n neurones cachés,
- $\hat{y}$ est la sortie calculée par le réseau,
- y la sortie attendue : ici $y = \cos x$,
- $\mathscr{H}$ est la fonction d'activation (nous prendrons la tangente hyperbolique)
- Pour $i \in [0; n-1]$, α_i est le poids du neurones c_i de la couche cachée et β_i son biais.
- Enfin pour $i \in [0; n-1]$, γ_i est le poids de la sortie et γ_n le biais.

Ainsi, on a :

$$\hat{y} = \gamma_n + \sum_{i=0}^{n-1} \gamma_i \mathscr{H}(\alpha_i x + \beta_i)$$

Pour une donnée x, on définit l'erreur comme le carré de l'écart entre la valeur calculée et la valeur attendue :

$$E = (y - \hat{y})^2 = \left(\gamma_n + \sum_{i=0}^{n-1} \gamma_i \mathscr{H}(\alpha_i x + \beta_i) - \hat{y}\right)^2$$

Pour une valeur donnée x, cette erreur est donc une fonction à $3n+1$ variables :

$$(\alpha_0, \alpha_1, ...\alpha_{n-1}, \beta_0, \beta_1, ..., \beta_{n-1}, \gamma_0, ..., \gamma_n).$$

Exercice 12 *Nous allons opérer une descente de gradient pour minimiser cette erreur.*

1. *Posons $\epsilon = y - \hat{y}$, montrer que (on peut admettre le résultat ☺) :*

   a. $\dfrac{\partial E}{\partial \gamma_n} = 2\epsilon$

   b. $\dfrac{\partial E}{\partial \gamma_i} = 2\epsilon \times \mathscr{H}(\alpha_i x + \beta_i)$

   c. $\dfrac{\partial E}{\partial \alpha_i} = 2x\epsilon \times \mathscr{H}'(\alpha_i x + \beta_i)$

   d. $\dfrac{\partial E}{\partial \beta_i} = 2\epsilon \times \mathscr{H}'(\alpha_i x + \beta_i)$

   Avec ici $\mathscr{H}(x) = tanh(x)$ et donc $\mathscr{H}(x) = 1 - tanh^2(x)$

2. *Remplir 3 listes `alpha`, `beta`, `gamma` avec des valeurs aléatoire.*
3. *Représenter la fonction désirée et la fonction prédit.*
4. *Appliquer une descente de gradient avec cet algorithme par exemple :*
   - ◇ *Choisir une valeur de $x \in [-3; 2]$ aléatoire.*
   - ◇ *Calculer le gradient de E en ce point.*
   - ◇ *Corriger les poids α_i, β_i et γ_i.*
   - ◇ *Afficher cette nouvelle courbe*
   - ◇ *Recommencer.*

Visualiser un exemple du résultat attendu.

III) Exemples en utilisant scikit-learn

1) Retour sur l'exemple précédent

Le module https://scikit-learn.org/stable/ propose de programmer très facilement des réseaux de neurones. Il n'est pas aussi puissant les modules comme tensorflow ou PyTorch, mais il permet déjà de faire des choses. Voici un code qui réalise la même chose que précédemment, vous pouvez le télécharger sur le site.

```
from sklearn.neural_network import MLPRegressor
import numpy as np
from matplotlib import pyplot as plt

nb_points = 1000                    # Nombre de points sur la courbe.
X = np.linspace(-2,3, nb_points)    # On se crée un tableau de nb_points nombres.
y = np.cos(X)                       # On calcule leurs images.
X = X.reshape(nb_points,1)          # Chaque entrée doit être sur une ligne...

modele = MLPRegressor(              # On crée le réseau de neurones de type "regression quadratique"
    hidden_layer_sizes = 100,       # Une couche cachée de 100 neurones,
    activation = 'tanh',            # Fonction d'activation
    solver = 'adam',                # Méthode de minimisation d'erreur
    max_iter = 10000,               # Nombre maximal d'itération
    batch_size = 1)                 # On prends les points 1 à 1

modele.fit(X, y)                    # On fait apprendre le modèle
score = modele.score(X, y)
print('Score du modèle :', score)
y_predit = modele.predict(X)        # On calcule les images avec ce modèle
plt.plot(X,y,'r-', label='y=cos(x)')
plt.plot(X,y_predit,'k--', label='y prédit')
plt.legend()
plt.show()
```

- `hidden_layer_sizes` : un entier ou un tuple indiquant le nombre de neurones sur les couches cachées. Par exemple `hidden_layer_sizes = (10, 15)` signifie qu'il y a 2 couches cachées : 10 neurones sur la première et 15 sur la seconde. (Par défaut : une couche cachée de 100 neurones)
- `activation` : les fonctions d'activations (se sont les mêmes sur toutes les couches). Ce paramètre peut prendre les valeurs :
  - ◇ `"identity"` : fonction identité (pas de fonction d'activation)
  - ◇ `"logistic"` : fonction sigmoïde
  - ◇ `"tanh"` : tangente hyperbolique
  - ◇ `"relu"` : fonction reLU
- `solver` : c'est la technique de régression. Peut prendre les valeurs :
  - ◇ `"lbfgs"` : Méthode de quasi-Newton.
  - ◇ `"sgd"` : Descente du gradient (stochastique).
  - ◇ `"adam"` : Descente du gradient stochastique avec une détermination automatique du taux d'apprentissage.
- `verbose` : un booléen indiquant si on veut ou non voir l'évolution de l'apprentissage lors du `fit`
- `max_iter` : maximum d'itération dans l'apprentissage (200 par défaut). Il s'agit d'un maximum car l'apprentissage peut stopper s'il ne progresse plus.
- `n_iter_no_change` : nombre de tentatives pour améliorer l'apprentissage (10 par défaut). Si aucune amélioration ne actée, le processus s'arrête.

2) coefficient de détermination

Le score obtenu par la méthode : `modele.score(X, y)` s'appelle le coefficient de détermination et est définit ainsi :

$$s = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2}$$

A compléter - plus le score est proche de 1, mieux c'est

3) Dernier problème de régression : création d'un pipeline

Un dernier problème pour essayer d'estimer une fonction à l'aide d'un réseau de neurones : considérons un rectangle le longueur L et de largeur ℓ (donc avec $\ell \leqslant L$) dans l'intervalle $[20; 100]$. Il est facile de déterminer l'aire $\mathscr{A}$ et le périmètre p de celui-ci.

Et si on tentait l'inverse ? Connaissant $\mathscr{A}$ et p, est-il possible de retrouver ℓ et L ?

En fait, oui, on peut d'ailleurs s'en sortir par le calcul, mais ce n'est pas notre but ici.

Exercice 13

1. *Réaliser une fonction `donnees(N)` qui reçoit un entier `N` et renvoie 2 tableaux de `N` lignes et 2 colonnes où pour chaque ligne du premier on trouve dans cet ordre ℓ et L et dans le deuxième $\mathscr{A}$ et p.*
2. *Tirer 2 tableaux de 300 lignes et faire apprendre les résultats à un réseau de neurones on commence par laisser les paramètres par défaut.*
3. *On peut définir l'erreur entre le couple largeur / longueur (ℓ, L) attendu et le couple $(\hat{\ell},\hat{L})$ estimé par le réseau de la manière suivante (comme une distance) : $E = \sqrt{(\ell - \hat{\ell})^2 + (L - \hat{L})^2}$.*
   a. *En fin d'apprentissage retirer un tableau de 100 lignes.*
   b. *Calculer l'erreur moyenne obtenue sur ce nouveau jeu de test.*
4. *Essayer d'améliorer le modèle en modifiant les paramètres.*

Avant de traiter le problème, il est souvent bienvenu de **normaliser** les valeurs. En effet des valeurs trop grosses et surtout dans des ordres de grandeur très différents amènent les poids du réseau de neurones à être très disparates que qui entraine des soucis de convergence.

Il existe différentes solutions de normaliser les valeurs (l'objectif est de les amener à être en grande partie dans l'intervalle $[-1; 1]$!).

- sklearn.preprocessing. StandardScaler : permet de centrer et réduire les données ($\tilde{x} = \dfrac{x - \mu}{\sigma}$)
- sklearn.preprocessing. MinMaxScaler : Amène de manière affine le minimum sur 0 et le maximum sur 1.
- sklearn.preprocessing. RobustScaler : Comme StandardScaler, mais l'arrivée est dans $[-2; 2]$ à préférer si les données possèdent quelques données aberrantes.

Par exemple le code :

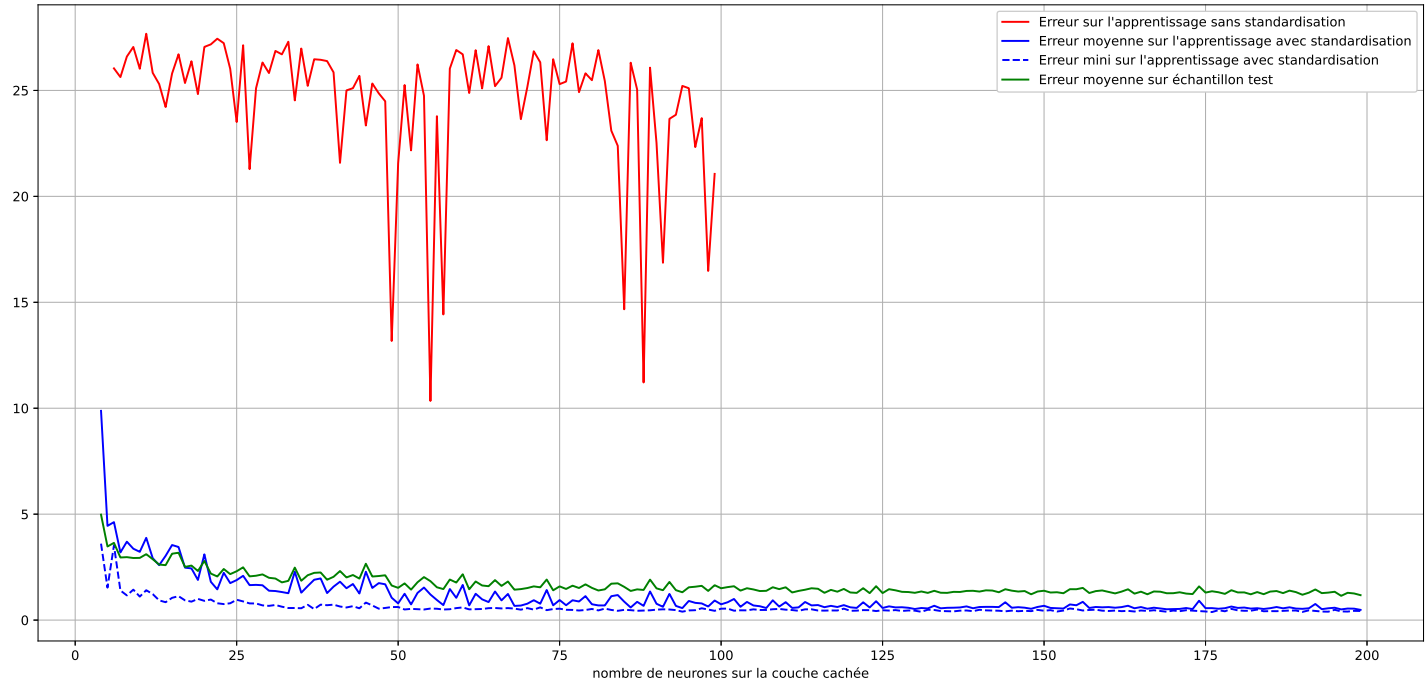
```
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
import numpy as np

A = np.array([[1,2,1], [1,3,2], [1,2,3]])
print('Départ : \n', A)
print('StandardScaler :\n', StandardScaler().fit(A).transform(A))
print('MinMaxScaler :\n', MinMaxScaler().fit(A).transform(A))
print('RobustScaler :\n', RobustScaler().fit(A).transform(A))
```

Donne le résultat :

```
Départ :
 [[1 2 1]
  [1 3 2]
  [1 2 3]]
StandardScaler :
 [[ 0.          -0.70710678 -1.22474487]
  [ 0.           1.41421356  0.        ]
  [ 0.          -0.70710678  1.22474487]]
```

```
Normalizer :
 [[0.  0.  0. ]
  [0.  1.  0.5]
  [0.  0.  1. ]]
RobustScaler :
 [[ 0.  0. -1.]
  [ 0.  2.  0.]
  [ 0.  0.  1.]]
```

Et cela a des effets sur la qualité d'apprentissage :



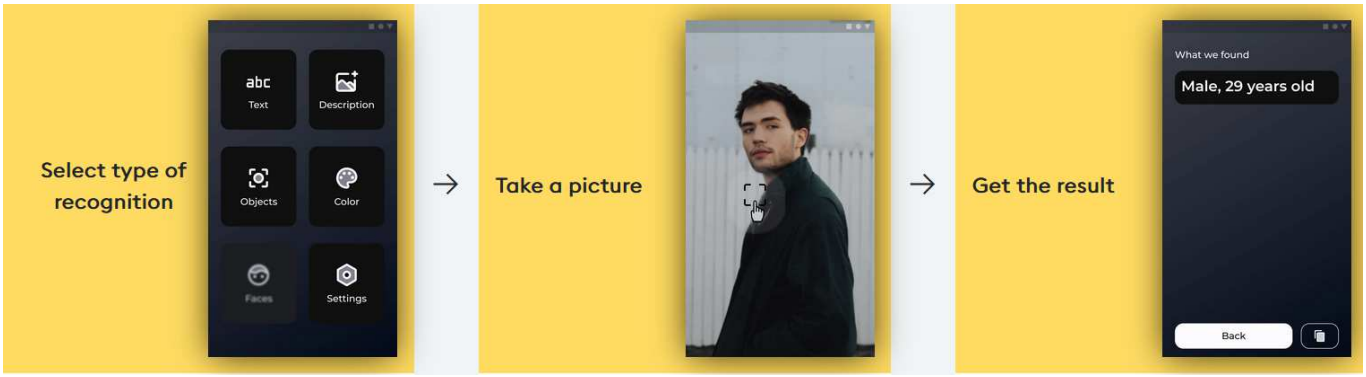
Plus simple, on peut créer un **Pipeline** qui va enchainer les processus suivants :

Données ⟶ Normalisation ⟶ Réseau ⟶ Sortie

```
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScale
from sklearn.pipeline import make_pipeline

modele = make_pipeline(
        StandardScaler(),
        MLPRegressor(....)
        )

modele.fit(X, y)
...
```

4) Un défi pour finir... : problème de sur apprentissage

L'application My Eyes disponible sous Androïd permet aux aveugles et mal voyant de lire, reconnaitre des scènes, des visages, ...



A vous de jouer en essayant de prédire l'âge d'une personne à partir d'une photo. Vous pouvez télécharger une grosse banque d'environ 10 000 images de taille 200×200 dont on connait l'âge du visage sur la photo. Ces photos proviennent du site https://www.kaggle.com/

Exercice 14 *A vous de jouer pour apprendre à la machine à estimer l'âge sur une photo quelconque. Peut-être aurez-vous besoin de ses fonctions :*

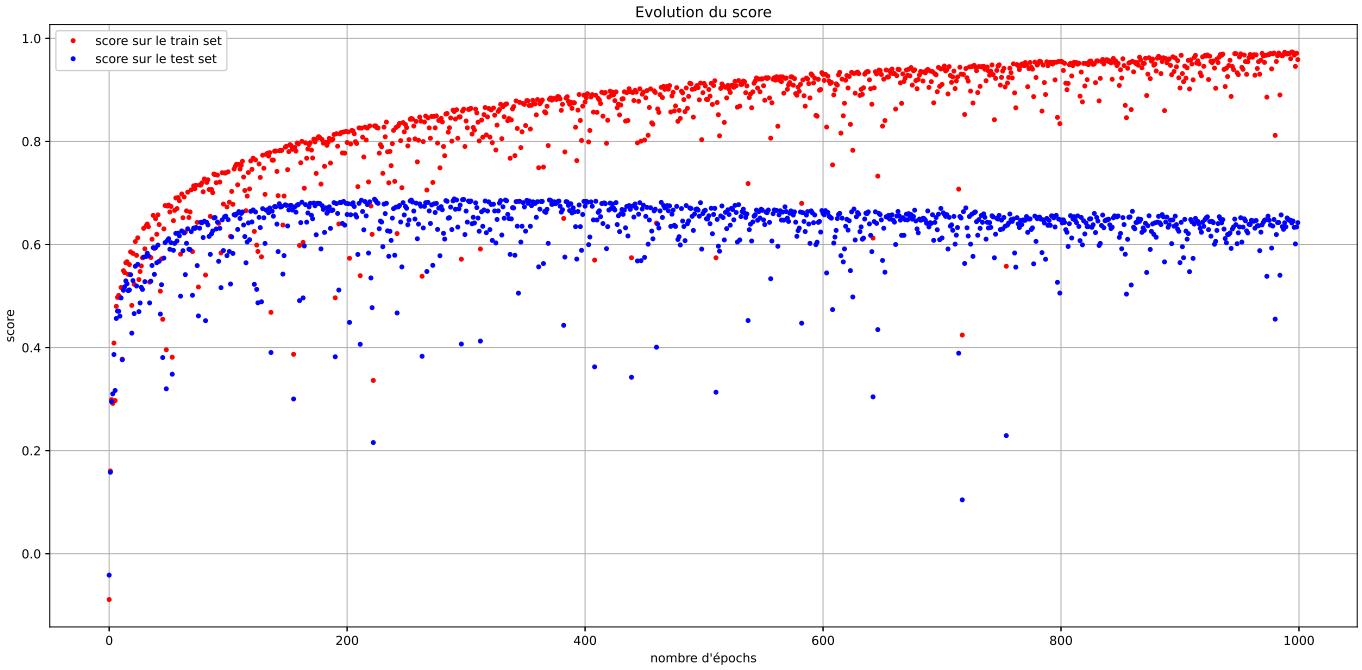
- *Module* `glob` *:* `glob` *pour lister les fichiers.*
- *Module* `PIL` *:* `Image.open`, `resize`, `convert` *pour traiter les images.*
- *Mondule* `numpy` *:* `zeros`, `array`, `reshape`, `load`, `save` *pour créer les tableau de données.*
- *Module* `pyplot` *:* `imgshow`, `cmap`, `show` *pour afficher les résultats.*

1. *Création données : la première étape consiste pour toutes les images) :*
   - *La charger (on peut aussi l'afficher).*
   - *La passer en niveau de gris.*
   - *La redimensionner en taille* 50×50 *pour limiter l'information.*
   - *L'aplatir en un vecteur de longueur* 2500 *que l'on normalisera.*
   - *Placer ce vecteur dans tableau numpy à* N *lignes et* 2500 *colonnes.*
   - *En parallèle créer un tableau de* N *valeurs représentant les âges dans le même ordre.*
2. *Apprentissage : Une fois ces données prêtes, vous pouvez faire apprendre à votre réseau ces données et afficher quelques sorties. Quelle est l'erreur moyenne d'âge ?*

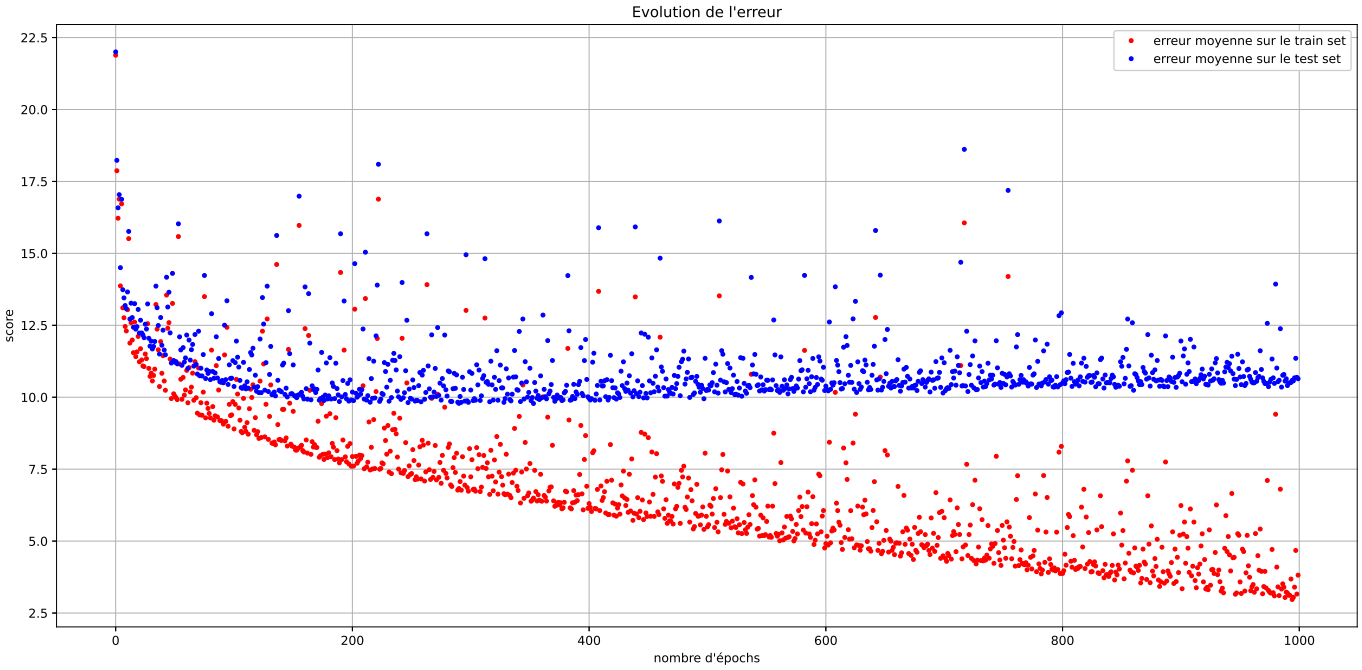
En fait, le problème est mal posé : au fur et à mesure que la modèle apprend des données d'entrainement, le modèle désapprend des généralités sur des données qu'il ne connait pas ! C'est le problème d'over fitting (ou de sur-apprentisage). On peut illustrer avec le code suivant :

```
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
import numpy as np
# Chargement données
d = np.load('data.npy')
t = np.load('target.npy')
# découpage
X_train, X_test, y_train, y_test = train_test_split(d, t, test_size=0.2, random_state = 20100)
# Normallisation
X_train = X_train / 255
X_test = X_test / 255
# modèle :
modele = MLPRegressor(
    hidden_layer_sizes = 500,
    activation = 'relu',
    solver = 'adam' , verbose = True, max_iter = 1000, n_iter_no_change = 100
)
# Apprentissage pas à pas et mémorisation des résultats sur 1000 epochs
S_train, S_test, E_train, E_test = [], [], [], []
for i in range(1000) :
```

```
    modele.partial_fit(X_train, y_train)
    S_train.append(modele.score(X_train, y_train))
    S_test.append(modele.score(X_test, y_test))
    E_train.append(np.mean(np.abs(y_train - modele.predict(X_train))))
    E_test.append(np.mean(np.abs(y_test - modele.predict(X_test))))
```




```
plt.plot(S_train, 'r.', label = 'score sur le train set')
plt.plot(S_test, 'b.', label = 'score sur le test set')
plt.title("Evolution du score")
plt.xlabel("nombre d'époch")
plt.ylabel("score")
plt.grid('on')
plt.legend()
plt.show()
```




```
plt.plot(E_train, 'r.', label = 'erreur moyenne sur le train set')
plt.plot(E_test, 'b.', label = 'erreur moyenne sur le test set')
```

```
plt.title("Evolution de l'erreur")
plt.xlabel("nombre d'épochs")
plt.ylabel("erreur")
plt.grid('on')
plt.legend()
plt.show()
```

IV) Classification

Jusque là nous avons fait des prédictions sur les variables "continues" comme l'âge, on parle de régression. Nous allons nous intéresser à présent à des problèmes de classification : par exemple à partir de la photo du visage, on voudrait déterminer la classe de la personne parmi {bébé, enfant, ados, jeune, adulte, sénior, âgé}.

1) Nouvelle idée de sortie

Le site quickdraw permet de s'amuser à dessiner et l'ordinateur reconnait votre œuvre ! Google qui propose cette application propose aussi de télécharger des banques d'images pour tester vos propres réseaux de neurones, il y a 345 catégories, mais nous n'allons en garder que 15 que nous allons numéroter :

A MODIFIER

0	Chien	1	Chat	2	Tour Eiffel	3	Pomme	4	Avion
5	Banane	6	Vélo	7	Appareil photo	8	Bougie	9	Fleur
10	Guitare	11	Chapeau	12	Glace	13	Souris	14	Pizza

Pour chaque catégorie est proposé un fichier `npy` contenant plus de 100 000 images de taille 28×28 en niveau de gris (codé sous forme d'entiers entre 0 et 255).

On pourrait on imaginer en sortie un seul neurone qui donnerai un nombre entre 0 et 14, mais nous allons faire autrement et utiliser un **encodage one-hot** : nous allons coder la sortie avec 15 neurones (autant que de classes) en mettant un 1 sur la composante qui correspondant à la classe de l'entrée. Par exemple :

$$\text{Chien : } \begin{pmatrix}1\\0\\0\\\vdots\\0\\0\end{pmatrix} ; \quad \text{Chat } \begin{pmatrix}0\\1\\0\\\vdots\\0\\0\end{pmatrix} ; \quad \text{Tour Eiffel } \begin{pmatrix}0\\0\\1\\\vdots\\0\\0\end{pmatrix} ; \quad \text{... Pizza : } \begin{pmatrix}0\\0\\0\\\vdots\\0\\1\end{pmatrix}.$$

L'activité se découpe en 2 étapes : création du tableau blanc, puis reconnaissance de celle-ci

Exercice 15 *Le code d'un tableau blanc sous PyGame peut être téléchargé pour commencer le projet. L'objectif est d'obtenir un tableau blanc pour dessiner en noir avec ces fonctionnalités :*

1. *Quand o*

2) Une nouvelle idée de sortie

A la sortie de notre réseau de neurones, on applique alors la fonction **softmax** σ définie de $\mathbb{R}^n$ dans $[0,1]^n$ ainsi : si $\overrightarrow{u} = (x_1, x_2, ..., x_n)$, alors $\sigma(\overrightarrow{u}) = \dfrac{1}{\sum_{i=1}^{n} e^{x_i}}(e^{x_1}, e^{x_2}, ..., e^{x_n})$.

Par exemple : $\sigma(2; 1.5; -0.4; 0; 1) = (\mathbf{0{,}45}; 0{,}28; 0{,}04; 0{,}06; 0{,}17)$ qui peut appréhender comme une distribution de probabilité. Ainsi sur cet exemple on conviendra de dire qu'avec 45% de certitudes, l'entrée correspond à un objet de la première classe.

L'erreur ..... CROSS-ENTROPY

3) Avec Sklearn

score = pourcentage réussite (et plus la cefficd f de determination
loss = A revoir

4) Cross validation

grid search cv

5) Zalando

Deuxième partie

Apprentissage non supervisé

V) K-mean

1) Algorithme

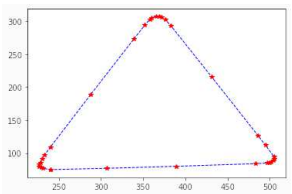
Problématique : Comment regrouper des données D_j en N paquets (**clusters**) ?
⓪ On choisit N centres $C_0, C_1, ...C_{N-1}$ au hasard.
① On assigne à chaque valeur D_j un numéro de cluster correspondant au centre le plus proche.
② On calcule alors les nouveaux centres C_i comme les positions moyenne des points du clusteur numéro i.
③ On recommence les étapes ① et ② jusqu'à ce que les positions des centres n'évoluent plus.

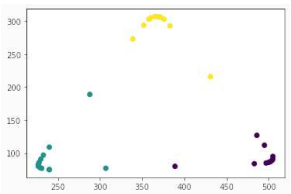
2) Exemple 1 : panneaux triangulaires

On cherche à déterminer les 3 sommets d'un panneau routier triangulaire (image à télécharger). Pour cela on a déjà récupéré et détouré la zone "rouge" (tableau à télécharger).

Image d'origine

Détourage


Points importants


Clusterisation

Exercice 16 *A vous !*

3) Le problème de l'initialisation

Inertie d'une solution : c'est la somme des carrés des distances de chaque valeur à son centre le plus proche :

$$I = \sum_j \min_i \|\overrightarrow{D_j C_i}\|^2$$

Cela quantifie la dispersion des données autour des centres (on peut l'interpréter comme une variance).

4) Avec Sklearn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Génération et affichage des données
X, y = make_blobs(n_samples=100, centers=3, cluster_std=1.5, random_state=20100)
plt.scatter(X[:,0], X[:,1])
plt.show()

# Apprentissage
model = KMeans(n_clusters=3)
model.fit(X)

# Affichage du résultat
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], c='r')
plt.show()
```

5) Comment choisir le nombre de clusters ?

Méthode Elbow

6) Exemple 2 : compression des couleurs

Avec cette idée, on peut passer cette photo de toucan en une image 16 ou même 4 couleurs :