Réseau de neurone

L'objectif est de reconnaître des chiffres en écriture manuscrite à l'aide d'un réseau de neurone. La donnée d'entrée est une image de $20 \times 20 = 400$ pixels. Le réseau de neurone renvoie 10 scores (un par chiffre). Le meilleur score donne la prédiction du réseau de neurone. Un exemple est donné ci – dessous.

Fig : *A droite, l'image filmée par la webcam branchée à l'ordinateur. Après un premier traitement on obtient une image de* 20×20 *pixels qui est affichée en insert en haut à gauche. Ces données sont alors traitées par le réseau de neurone. On obtient 10 scores (un pour chaque chiffre). Ils sont affichés dans la fenêtre de gauche (sous la forme d'une liste). Le meilleur score est de* 96% *pour le chiffre 8*

Chiffre	*Score*
0	0.0003460
1	0.0000057
2	0.0113035
3	0.0003263
4	0.0000876
5	0.0005120
6	0.0031184
7	0.0002562
8	**0.9679236**
9	0.0006966

Table : *Score renvoyé par le réseau de neurone correspondant à la figure ci – dessus. Il y a un score par chiffre. Le meilleur score correspond à la prédiction du réseau de neurone. Dans ce cas c'est le chiffre* 8 *qui est détecté.*

Avant de se lancer dans la reconnaissance de chiffre on va s'entraîner sur un problème plus simple : *Prévoir si une personne va mettre un bonnet en fonction de la température*. On va en profiter pour présenter la fonction de prédiction dont la qualité est testée par la fonction d'évaluation.

Fonction d'évaluation

On souhaite prédire un résultat (qui ne peut être que vrai ou faux) à partir de données que l'on va noter x. Par exemple prédire si une personne va mettre un bonnet en fonction de la température extérieure ($vrai$ si on met le bonnet et $faux$ sinon). Dans ce cas x est la température. Pour améliorer la prédiction on peut ajouter d'autres critères comme l'âge de la personne, son poids, sa taille. Dans ce cas x est alors un vecteur à 4 composantes $x = (T_{ext}, age,\ poids, taille)$.

Rem : *Dans la suite du document on remplacera indifféremment* $vrai$ *par* 1 *et* $faux$ *par* 0 *comme le ferait l'interpréteur* $python$.
Rem : *Un autre exemple que l'on aurait pu traiter est la prédiction du vote d'une personne pour un candidat à des élections en fonction de l'âge de la personne, son pouvoir d'achat, …*

Une prédiction n'est possible que si on s'est entraîné au préalable avec des données dont on connaît le résultat. On a donc pour reprendre l'exemple ci – dessus constaté sur un échantillon de N personnes lesquelles mettent un bonnet ou non en fonction de la température extérieure, l'âge, la taille, le poids. On a donc à disposition N couples de données (x_i, y_i) où $y_i = vrai$ si la $i^{ème}$ personne met un bonnet et $y_i = faux$ sinon pour une température x_i. L'objectif est alors à partir des données accumulées de prédire ensuite pour d'autres personnes qui va se couvrir la tête ou pas.

Concrètement c'est une fonction de prédiction qui va donner le résultat. On la note $h(x)$. On cherche la fonction $h(x)$ qui va minimiser la fonction d'évaluation $J(h)$ qui est définie ci – dessous.

$$\begin{cases} h(x) & : La\ fonction\ de\ prédiction \\ J(h) = -\frac{1}{N}\left[\sum_{i=0}^{N-1} y_i \times ln\big(h(x_i)\big) + (1-y_i) \times ln\big(1-h(x_i)\big)\right] & : La\ fonction\ d'évaluation \end{cases}$$

Rem : *La définition de la fonction d'évaluation implique que la grandeur renvoyée par* $h(\quad) \in]0,1[$.

L'objectif est de trouver une fonction de prédiction qui minimise la fonction d'évaluation. Ainsi la notation est inversée. Plus le résultat renvoyé par $J(h)$ est proche de zéro mieux c'est.

Commentaire sur la fonction d'évaluation

Dans ce paragraphe on discute de la pertinence de la définition de la fonction d'évaluation. y_i ne peut prendre que les valeurs 0 *(faux)* ou 1 *(vrai).*

Ainsi dans la somme soit c'est $y_i \times ln\big(h(x_i)\big)$ qui est nul, soit c'est $(1 - y_i) \times ln\big(1 - h(x_i)\big)$. Sur les deux termes de la somme celui qui peut être non nul va être d'autant plus proche de zéro que la prédiction $h(x_i)$ est bonne (c'est – à – dire proche de y_i).

Le tableau ci – dessous traite différents cas que l'on peut rencontrer. On peut constater que les termes de la fonction d'évaluation sont d'autant plus proches de zéro (en restant toujours positif) que la prédiction est bonne.

$h(x_i)$	y_i	$-y_i \times ln\big(h(x_i)\big)$	$-(1 - y_i) \times ln\big(1 - h(x$	$-y_i \times ln\big(h(x_i)\big) - (1 - y_i) \times ln\big(1 - h(x_i)\big)$
0.07	0	0	0.072570	0.072570 *bonne prédiction*
0.97	0	0	3.506557	3.506557 *mauvaise prédiction*
0.98	1	0.02020	0	0.02020 *bonne prédiction*
0.05	1	2.99573	0	2.99573 *mauvaise prédiction*

Ainsi comme annoncé, la fonction d'évaluation renvoie une valeur d'autant plus proche de zéro que la prédiction renvoyée par $h(x)$ est correcte.

Mise en œuvre

On considère les données dans le tableau ci – dessous. On cherche la fonction de prédiction sous la forme ci – dessous (choix imposé par le concepteur du sujet) :

$$h(x) = \frac{1}{1 + e^{ax+b}}$$

a et b sont des constantes

Rem : *Avec cette fonction le résultat est forcément compris dans l'intervalle* $]0,1[$.
Rem : *On peut envisager d'autres fonctions. Libre à vous d'en imaginer d'autres.*

Données : *On dispose d'un échantillon de* 20 *personnes. Pour la deuxième ligne* 1 *signifie que la personne a mis un bonnet et* 0 *sinon. Les* x_i *correspondent à la première ligne et les* y_i *à la deuxième.*

$T(°C)$	14	-1	8	3	-9	-24	-3	4	3	-17	9	6	41	-16	-6	-4	-8	19	17	-9
Bonnet	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1

Objectif : Trouver a et b qui minimise la fonction d'évaluation $J(h)$.

Rem : *Dans la suite du document vous devez comprendre que lorsqu'on écrit* $J(h)$*, cette grandeur dépend en réalité de* a*,* b *et des séquences* x_i *et* y_i*.*

1. Ecrire une fonction $F1(x, a, b)$ qui renvoie à partie des paramètres x, a et b la valeur de la fonction de prédiction $h(x)$ définie ci – dessus.

   Rem : *Toutes les fonctions dans ce document sont appelées $Fi(\ )$. Vous pouvez choisir un autre nom si vous le souhaitez.*

2. Ecrire une fonction $F2(xi, yi, a, b)$ qui renvoie à partir de deux séquences xi et yi et de deux flottants a et b la valeur de la fonction d'évaluation $J(h)$. Vous devez donc rentrer « à la main » les deux séquences x_i et y_i du tableau ci – dessus.

3. En choisissant au hasard les paramètres a et b proposer des valeurs qui donne une valeur de la fonction d'évaluation la plus faible possible. ***Vous devez trouver des valeurs de $a \approx 0.20$ et $b \approx -0.50$.***

Indication : *Tester un grand nombre de couples (a, b) choisis aléatoirement. Vous allez ainsi augmenter vos chances de vous approcher du minimum de $J(h)$. On vous donne ci – dessous le code pour générer des nombres aléatoires avec une distribution gaussienne centrée en m avec un écart – type σ (noté $sigma$ dans le code).*

Python 3.4

```
 8 import random as r
 9 sigma = 1.  # la moyenne : Choisissez une valeur
10 m = 1. # l'écart - type : Choisissez une valeur
11 val = r.gauss(m,sigma) # Appeler r.gauss(m,sigma) autant de fois que nécessaire
```

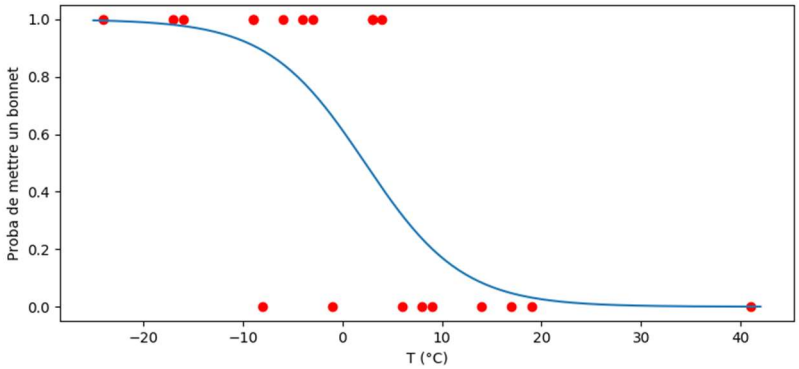


Fig : *En trait plein la fonction de prédiction en fonction de la température pour $a \approx 0.20$ et $b \approx -0.50$. Les points représentent les données de l'échantillon de 20 personnes.*

Méthode du gradient pour trouver les paramètres a et b qui donnent la valeur minimale de $J(h)$

La fonction d'évaluation $J(h)$ dépend en réalité de a, b, et des deux séquences x_i et y_i. Comme les données x_i et y_i sont fixées, il n'y a en réalité que deux paramètres dont dépend $J(h) = f(a, b)$. On a :

$$df = \frac{\partial f}{\partial a} da + \frac{\partial f}{\partial b} db = \underbrace{\begin{vmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{vmatrix}}_{gradient} \cdot \underbrace{\begin{vmatrix} da \\ db \end{vmatrix}}_{\substack{déplacement \\ élémentaire}}$$

Comme on veut trouver le minimum de $J(h) = f(a, b)$ il suffit de se déplacer en sens opposé du gradient (défini ci – dessus). En effet dans ce cas le produit scalaire entre le vecteur gradient et le déplacement élémentaire est alors négatif.

La méthode du gradient consiste à se déplacer toujours selon la ligne de plus grande pente. Le vecteur gradient est donc toujours perpendiculaire aux lignes à f constant. On peut le voir sur la figure ci – dessous.

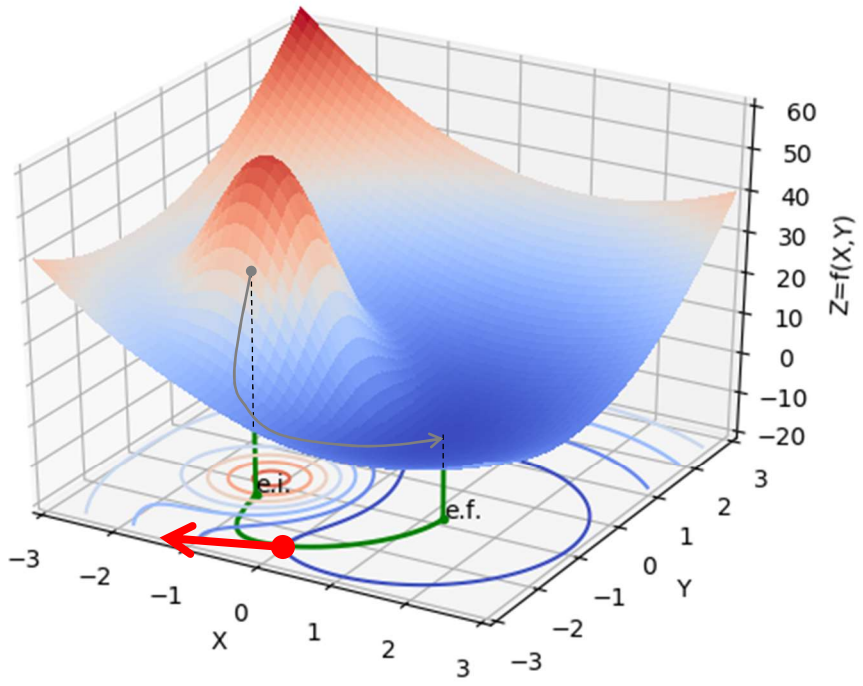


Fig : *Illustration de la méthode du gradient pour une surface définie par $Z = f(X, Y)$. On part d'un état initial (e.i.) qu'on a placé dans le plan OXY. En se déplaçant en sens opposé du gradient on arrive au point le plus bas de la surface (état final, e.f.). Le vecteur gradient a été représenté en un point (flèche en trait gras). Les lignes de niveau (à f constant) sont représentées dans le plan OXY. La méthode du gradient consiste à se déplacer toujours suivant la ligne de plus grande pente.*

On va donc appliquer la procédure suivante :

- ***(i)*** *On note a_n et b_n les paramètres calculés à la $n^{ème}$ itération. On calcule $\frac{\partial f}{\partial a}$ et $\frac{\partial f}{\partial b}$*
- ***(ii)*** *On détermine les coordonnées du point suivant dans l'espace des paramètres en se déplaçant dans la direction opposée au gradient. On a donc besoin de se donner un pas $epsilon$. On calcule alors les coordonnées du point suivant dans l'espace des paramètres à l'aide des formules ci – dessous :*

$$\begin{cases} a_{n+1} = a_n - \dfrac{\partial f}{\partial a} \times epsilon \\ b_{n+1} = b_n - \dfrac{\partial f}{\partial b} \times epsilon \end{cases}$$

Rem : *Il ne faut pas prendre $epsilon$ trop grand sous peine d'instabilité numérique (la suite peut diverger au lieu de converger). Inversement il ne faut pas le choisir trop petit sous peine d'augmenter inutilement le nombre d'étapes du calcul.*

4. Mettez en œuvre cette méthode du gradient. ***Vous devez trouver pour a et b les valeurs $a = 0.204$ et $b = -0.502$.***

Méthode pour trouver les paramètres a et b : algorithme de Nelder – Mead

L'algorithme de Nelder – Mead permet de trouver le minimum d'une fonction de N variables. Dans notre cas $N = 2$ car la fonction d'évaluation dépend de deux paramètres a et b.

L'algorithme exploite la notion de simplex qui est un polytope de $N + 1$ sommets dans un espace à N dimensions. Dans notre cas le simplex sera un triangle. Partant initialement d'un simplex celui-ci va subir des transformations simples au cours des itérations : il se déforme, se déplace et se réduit progressivement jusqu'à ce que ses sommets se rapprochent d'un point où la fonction est localement minimale.

Les étapes de l'algorithme sont définies précisément ci – dessous dans le cas $N = 2$:

- ***(i)*** *Choix de 3 points de l'espace $X_1 = (a_1, b_1)$, $X_2 = (a_2, b_2)$ et $X_3 = (a_1, b_1) = (a_3, b_3)$.*
- ***(ii)*** *Calcul des valeurs de la fonction $J(h) = f(X_i)$ en ces points. On réindexe les points de façon à avoir $f(X_1) \leq f(X_2) \leq f(X_3)$. Il suffit en fait de connaître le premier et les deux derniers.*
- ***(iii)*** *Calcul de X_0 centre de gravité de X_1 et X_3.*
- ***(iv)*** *Calcul de $X_r = X_0 + (X_0 - X_3)$ (réflexion de X_3 par rapport à X_0.*
- ***(v)*** *Si $f(X_r) < f(X_2)$ on calcule $X_e = X_0 + 2 \times (X_0 - X_3)$ (étirement du simplexe). Si $f(X_e) < f(X_r)$ alors on affecte X_e à X_3 Sinon on affecte X_r à X_3. On recommence à l'étape (ii).*
- ***(vi)*** *Si $f(X_2) \leq f(X_r)$ alors on calcule $X_e = X_3 + 1/2\,(X_0 - X_3)$ (contraction du simplex). Si $f(X_e) \leq f(X_2)$ alors on affecte X_e à X_3 et retour à l'étape (ii), sinon aller à l'étape (vii).*
- ***(vii)*** *Homothétie de rapport $1/2$ et de centre x_1 : On applique à tous les points la relation $X_i = X_1 + 1/2\,(X_i - X_1)$.*

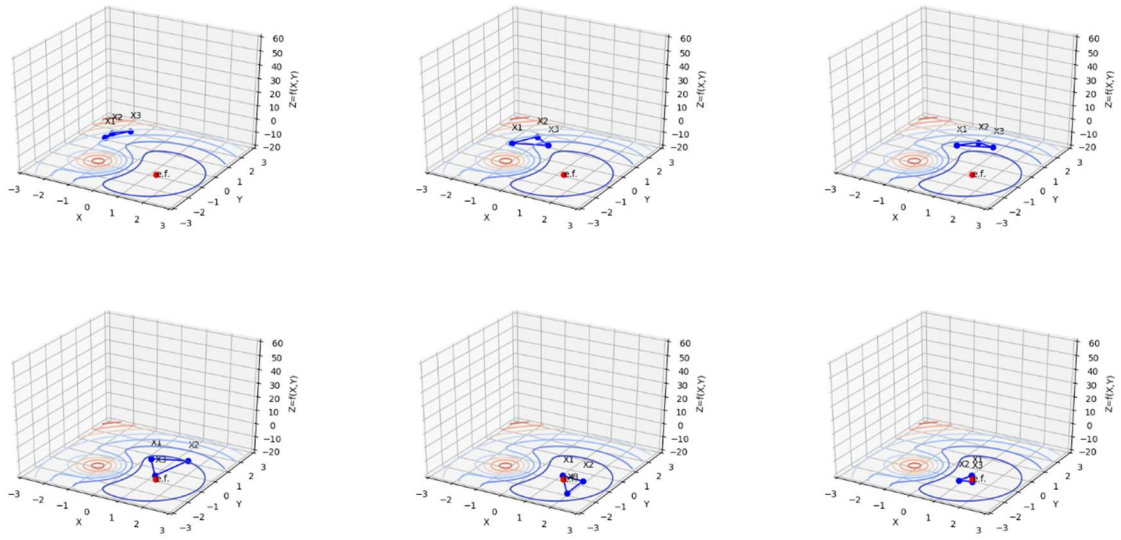


Fig : *Représentation dans le plan OXY des lignes de niveau et de la position des trois sommets du simplex (qui forme un triangle) pour les itérations $n = 0,3,6,9,12$ et 15. Pour des raisons de clarté la surface $Z = f(X,Y)$ n'est pas représentée.*

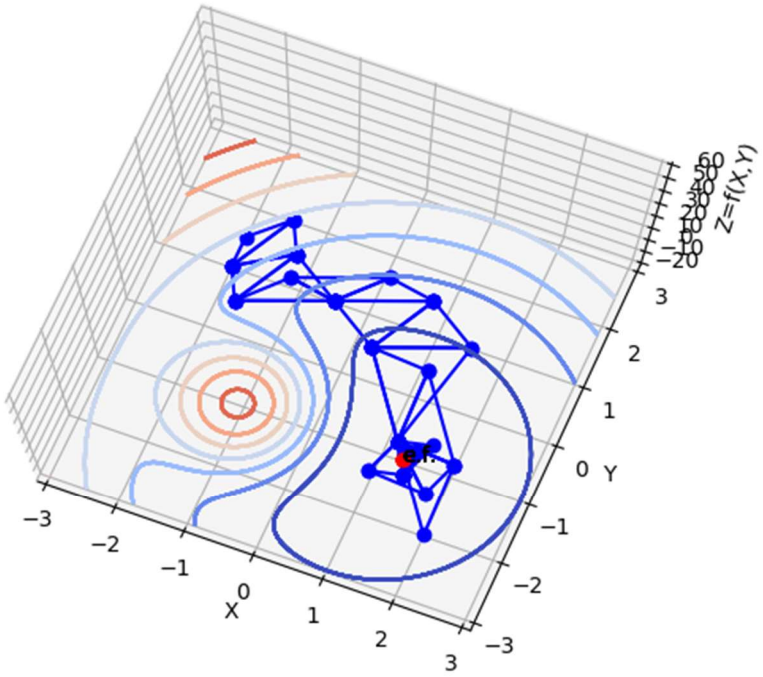


Fig : *Superposition de tous les simplex (ici des triangles) des 20 premières itérations de l'algorithme de Nelder – Mead. Les lignes de niveau sont également représentées. On voit ainsi le chemin qui mène au minimum de la fonction $Z = f(X,Y)$. Pour des raisons de clarté la surface $Z = f(X,Y)$ n'est pas représentée.*

5. Mettre en œuvre l'algorithme de Nelder – Mead. Reprendre l'exemple de la méthode du gradient. ***Vous devez trouver pour a et b les mêmes valeurs que précédemment.***

Indication : *Vous pouvez écrire une fonction $F3(X, Y, f)$ où X (respectivement Y) est une liste de 3 éléments contenant les abscisses (respect. ordonnée) des sommets du simplex et f la fonction dont on cherche le minimum. Cette fonction $F3$ doit renvoyer les coordonnées des sommets du simplex à l'itération suivante. A vous de choisir sous quelle forme récupérer ces coordonnées.*

Un seul neurone

On présente dans cette partie un modèle numérique de neurone. On commence par décrire le fonctionnement d'un neurone à deux entrées. La valeur des entrées est comprise entre 0 et 1. On ajoute une troisième entrée qui est toujours à 1 et qui permet d'avoir un biais le cas échéant même si les deux autres entrées sont nulles.

Le résultat en sortie est obtenu en deux temps. On calcule un paramètre qui est une fonction affine des entrées (biais inclus). On appelle z ce paramètre. $z = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ (θ_0, θ_1 et θ_2 sont des constantes). On applique à ce résultat une fonction $g(z) = \frac{1}{1+e^{-z}}$ dont le résultat est compris entre 0 et 1.

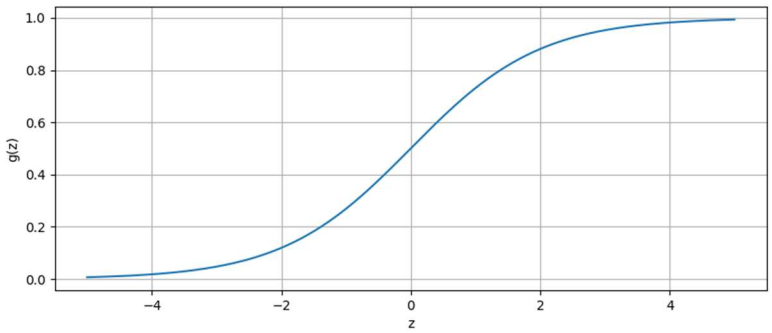


Fig : *La fonction $g(z)$ qui renvoie toujours un résultat $\in [0,1]$. Cette fonction ressemble à une fonction seuil avec « des bords arrondis ».*

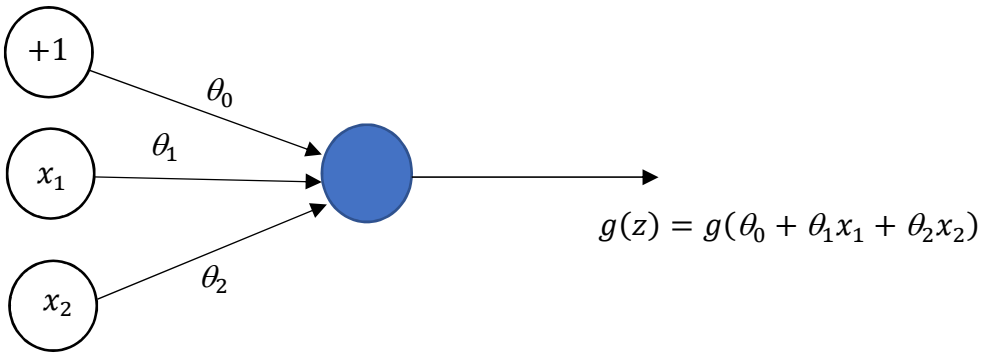


Fig : *Modèle d'un neurone à deux entrées.*

6. Ecrire une fonction $F4(X, Teta)$ qui renvoie le résultat du neurone (où X est une séquence (tableau ou liste) qui contient deux éléments x_1 et x_2 et $Teta$ une séquence de 3 constantes θ_0, θ_1 et θ_2).

7. Testez votre code avec $\theta_0 = -10$, $\theta_1 = 20$ et $\theta_2 = 20$, $x_1 = \pm 1$ et $x_1 = \pm 1$. Quelle porte logique a-t-on dans ce cas ?

8. Comment choisir θ_0, θ_1 et θ_2 pour avoir une porte logique ET ?

Dans la suite on s'intéresse à un réseau de neurone. Le modèle que l'on va utiliser est décrit dans la section suivante.

Réseau de neurone

Le modèle utilisé est un empilement de plusieurs couches de neurones dont le résultat d'une couche est envoyé à la couche suivante. La figure ci – dessous illustre la structure d'un réseau constitué de deux couches de respectivement 4 et 3 neurones. On notera dans la suite N_k le nombre de neurones de la couche (k).

Notation : On adopte la notation suivante pour les coefficients $\theta_{ij}^{(k)}$ qui relient les neurones :

$$\theta_{ij}^{(k)} \begin{cases} k : indice\ de\ la\ couche\ de\ départ \\ i : indice\ du\ neurone\ d'arrivée\ dans\ la\ couche\ k+1 \\ j : indice\ du\ neurone\ de\ départ\ dans\ la\ couche\ k \end{cases}$$

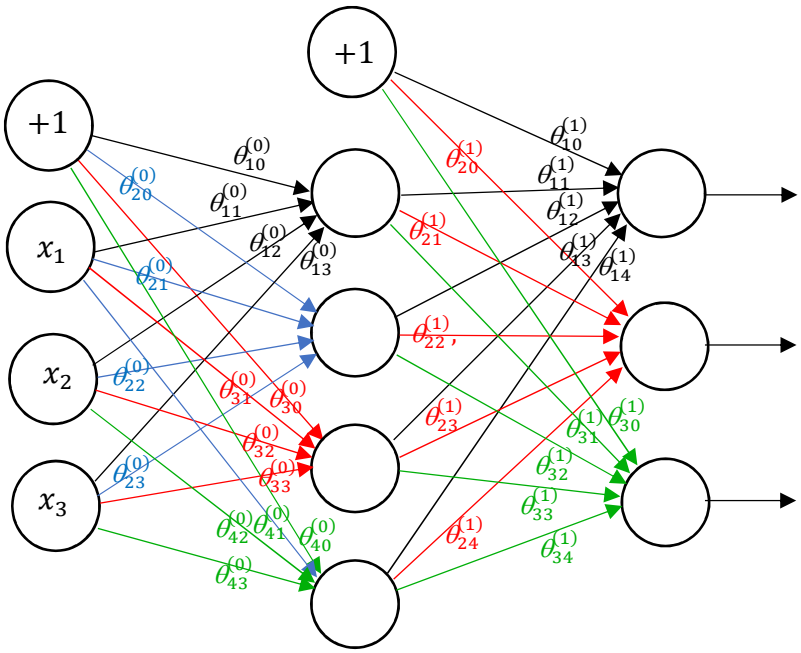


Fig : *Modèle d'un réseau de neurone avec deux couches de respectivement $N_1 = 4$ et $N_2 = 3$ neurones.*

Les calculs à faire pour obtenir le résultat de ce réseau se déduisent de ceux pour un seul neurone. Ils sont donnés ci – dessous :

$$\begin{cases} z_i^{(k+1)} = \theta_{i0}^{(k)} \times \underbrace{a_0^{(k)}}_{=+1} + \theta_{i1}^{(k)} \times a_1^{(k)} + \theta_{i2}^{(k)} \times a_2^{(k)} + \theta_{i3}^{(k)} \times a_3^{(k)} = \sum_{j=0}^{3} \theta_{ij}^{(k)} \times a_j^{(k)} = 0 \\ a_i^{(k+1)} = g\left(z_i{}^{(k+1)}\right) \end{cases}$$

Rem : *On a pour la première couche de neurone* $x_i = a_i^{(0)}$.

Rem : *On a en particulier* $a_0^{(k)} = +1$. *Cela correspond au biais.*

On peut réécrire ces relations sous forme matricielle. En posant les vecteurs $Z^{(k)} = \left[z_1^{(k)}, z_2^{(k)}, \dots, z_{N_k}^{(k)}\right]$ et $A^{(k)} = \left[a_0^{(k)}, a_1^{(k)}, \dots, a_{N_k}^{(k)}\right]$ on a la formulation ci – dessous. Notez que le vecteur $Z^{(k)}$ démarre avec un indice 1 et le vecteur $A^{(k)}$ avec un indice 0. On écrit indifféremment les vecteurs sous forme de ligne ou de colonne.

$$Z^{(k+1)} = \begin{bmatrix} \theta_{10}^{(k)} & \cdots & \theta_{1N_k}^{(k)} \\ \vdots & \ddots & \vdots \\ \theta_{N_{k+1}0}^{(k)} & \cdots & \theta_{N_{k+1}N_k}^{(k)} \end{bmatrix} A^{(k)} \qquad (*)$$

$$Puis \quad \begin{cases} a_i^{(k+1)} = g\left(z_i^{(k+1)}\right) \ pour \ i \in [\![1; N_{k+1}]\!] \\ a_0^{(k+1)} = 1 \quad (le \ biais) \end{cases}$$

Mise en œuvre : reconnaissance de chiffre

Le réseau de neurone que l'on va utiliser comporte 400 entrées autant que de pixels dans l'image. On doit ajouter 1 entrée (le biais toujours à $+1$). On a donc en tout 401 entrées. Il y a ensuite une première couche de $N_1 = 25$ neurones puis une deuxième couche de $N_2 = 10$ neurones (un par chiffre). On a donc en reprenant dans ce cas particulier les relations $(*)$:

$$\underbrace{\begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_{25}^{(1)} \end{bmatrix}}_{=Z^{(1)}} = \underbrace{\begin{bmatrix} \theta_{1,0}^{(0)} & \cdots & \theta_{1,400}^{(0)} \\ \vdots & \ddots & \vdots \\ \theta_{25,0}^{(0)} & \cdots & \theta_{25,400}^{(0)} \end{bmatrix}}_{matrice\ 25\times401} \underbrace{\begin{bmatrix} +1 = a_0^{(0)} \\ x_1 \ ou \ a_1^{(0)} \\ \vdots \\ x_{400} ou \ a_{400}^{(0)} \end{bmatrix}}_{=A^{(0)}}$$

$$\underbrace{\begin{bmatrix} z_1^{(2)} \\ \vdots \\ z_{10}^{(2)} \end{bmatrix}}_{=Z^{(2)}} = \underbrace{\begin{bmatrix} \theta_{1,0}^{(1)} & \cdots & \theta_{1,25}^{(1)} \\ \vdots & \ddots & \vdots \\ \theta_{10,0}^{(1)} & \cdots & \theta_{10,25}^{(1)} \end{bmatrix}}_{matrice\ 10\times26} \underbrace{\begin{bmatrix} +1 = a_0^{(1)} \\ g\left(z_1^{(1)}\right) = a_1^{(1)} \\ \vdots \\ g\left(z_{25}^{(1)}\right) = a_{25}^{(1)} \end{bmatrix}}_{=A^{(1)}}$$

$$a_{0<i\leq1}^{(2)} = g\left(z_i^{(2)}\right) \quad score\ pour\ le\ chiffre\ (i-1)$$

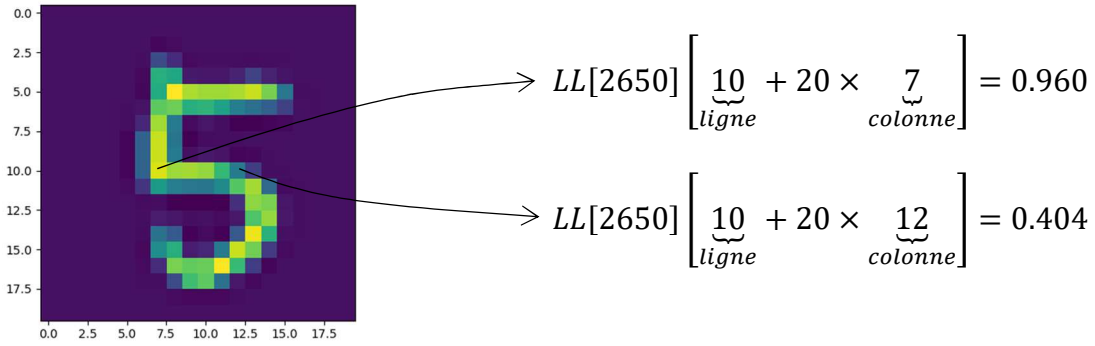
Avant de se lancer dans une reconnaissance en temps réel avec les images issues d'une caméra (objectif un peu ambitieux dans un premier temps) on va utiliser celles stockées dans un fichier (conçu par l'université de Stanford). On dispose de 5000 écritures de chiffres de 0 à 9. Cela fait 500 écritures différentes pour chaque chiffre. Les 500 premières images (d'indice 0 à 499) correspondent au chiffre 0. Les 500 suivantes au chiffre 1, et ainsi de suite jusqu'aux 500 dernières correspondant au chiffre 9. Ces données sont dans le fichier $DonneesMachineLearning01.py$.

Les valeurs des coefficients $\theta_{i,j}^{(k)}$ utilisées par le réseau de neurone sont dans le fichier $Teta451.py$. Les coefficients sont déjà optimisés pour donner de bons résultats. On peut les utiliser tels quels.

L'importation des fichiers $DonneesMachineLearning01.py$ et $Teta451.py$ se fait avec la même syntaxe que celle utilisée pour l'importation d'un module (ligne 9 et 10 ci – dessous). Une fois importé ces deux fichiers, les données sont dans deux listes $\boldsymbol{LL}$ et $\boldsymbol{LL\_Teta}$ dont la structure est donnée ci – dessous.

- ***La valeur en niveau de gris du pixel à la ligne a et la colonne b de la $n^{ème}$ image est dans la variable :***

$$\boldsymbol{LL[n][a+20\times b]}$$



$Image\ d'indice\ 2650$

- ***Le coefficiente $\theta_{i,j}^{(k)}$ est dans la variable :*** $\boldsymbol{LL\_Teta[k][i][j]}$
- ***Les données en entrée du réseau de neurone sont choisies de la manière suivante (choix imposé par le concepteur du sujet) :***

$$\begin{cases} \boldsymbol{a_0^{(0)}=1} \\ \boldsymbol{a_i^{(0)}=LL[n][i-1]} \quad \boldsymbol{pour\ 0<i<401} \end{cases}$$

Le code ci – dessous permet l'affichage du nombre 5 dont on voit l'image ci – dessus. Il faut veiller à avoir tous les fichiers (votre code plus les deux fichiers de données) dans le même répertoire. Dans le code ci – dessous on commence par convertir la liste des 400 valeurs en tableau (ligne 12). On redimensionne le tableau en 20×20 avec la méthode $.reshape()$ (ligne 13). On doit enfin faire tourner l'image autour de sa bissectrice pour avoir une image lisible (cela revient à permuter les lignes et les colonnes) (ligne 14). On affiche le résultat avec la fonction $plt.imshow(\ \ )$.

Python 3.x

```
import numpy as np
import matplotlib.pyplot as plt
from DonneesMachineLearning01 import *
from Teta451 import *

_ = np.array(LL[2650])  # LL[2650] est une séquence de 400 flottants (le 2650ème échantillons)
__ = _.reshape((20,20)) # On en fait une image de 20*20 pixels
__ = __.transpose()     # On tourne l'image par rapport à sa bisectrice
fig = plt.figure()
plt.imshow(__)
plt.show()
```

9. Ecrire une fonction $h(A0, LL\_Teta)$ prenant comme paramètre $A0$ une séquence de 401 données (on rappelle que la première donnée est le biais égal à $+1$) et les coefficients $\theta_{i,j}^{(k)}$ et qui renvoie la séquence $\left[a_1^{(2)}, \dots, a_{10}^{(2)}\right]$ en sortie du réseau de neurone.

Indication : *Vous pouvez faire un code très concis pour $h(a0, LL\_Teta)$ si vous écrivez au préalable une fonction $Mult(A, B)$ qui multiplie la matrice A par le vecteur B. Dans cette fonction la matrice A est codée par une liste de liste et B par une liste. Le résultat doit être un tableau (type $array$).*

10. Testez la fonction pour l'image d'indice 2650. Vérifier que le meilleur score correspond bien au sixième élément $a_6^{(2)}$ de la séquence renvoyée par la fonction $h(a0, LL\_Teta)$. ***La fonction doit renvoyer comme résultat la liste :*** $[6.2e^{-6}, 2.3e^{-7}, 8.2e^{-9}, 0.0114,\ 7.8e^{-6}, \mathbf{0.996},\ 2.7e^{-6}, 7.3e^{-7}, 1.7e^{-7}, 2.1e^{-5}]$

La fonction d'évaluation est donnée ci – dessous. On a besoin cette fois ci de deux sommes imbriquées (sur les 5000 images en entrées et sur les 10 données en sortie du réseau de neurone).

$$J(h) = -\frac{1}{N}\left[\sum_{n=0}^{N-1}\sum_{i=0}^{P-1} y_{n,i} \times ln\big(h_i(x_n)\big) + \big(1 - y_{n,i}\big) \times ln\big(1 - h_i(x_n)\big)\right]$$

(avec $N = 5000$ et $P = 10$)

Les notations sont les suivantes : *Pour la $n^{ème}$ image $h(x_n) = [h_0(x_n), \dots, h_9(x_n)]$ est le résultat renvoyé par la fonction $h(\ \ )$, et $y_n = \left[y_{n,0}, \dots, y_{n,9}\right]$ est le résultat attendu. Par exemple pour le chiffre 5 on attend en sortie du réseau de neurone la séquence $y =$* $[0,0,0,0,0,1,0,0,0,0]$.

11. Ecrire la fonction d'évaluation $J(LL, LL\_Teta)$ des 5000 échantillons. LL et LL_Teta sont les deux variables contenant les images et les coefficients $\theta_{i,j}^{(k)}$. ***La fonction doit renvoyer la valeur 0.0902310852378.***

Entraînement d'un réseau de neurone – version naïve

Implémenter un réseau de neurone est relativement simple. La difficulté majeure est d'entraîner ce réseau. Il faut trouver les coefficients $\theta_{i,j}^{(k)}$ qui donnent la valeur la plus basse possible de la fonction d'évaluation.
On va utiliser dans cette section la méthode du gradient déjà présenté au début du document. On va constater que cette méthode devient inefficace (très lente) lorsqu'il y a trop de paramètres. On présentera dans la section suivante une autre méthode beaucoup plus rapide.

On doit donc calculer les termes suivant pour tous les $\theta_{i,j}^{(k)}$:

$$\frac{\partial\big(J(h)\big)}{\partial\,\theta_{i,j}^{(k)}}$$

On peut alors actualiser la valeur des coefficients $\theta_{i,j}^{(k)}$ avec les relations (où $epsilon$ est le pas) :

$$\underbrace{\theta_{i,j}^{(k)}}_{\substack{\text{à } l'itération \\ suivante}} = \theta_{i,j}^{(k)} - \frac{\partial\big(J(h)\big)}{\partial\,\theta_{i,j}^{(k)}} \times epsilon$$

12. Ecrire une fonction $F5(LL, LL\_Teta, k, i, j, \varepsilon = 10^{-4})$ qui calcule numériquement la dérivée partielle $\frac{\partial(J(h))}{\partial\theta_{i,j}^{(k)}} \approx \frac{1}{2\varepsilon}\big(J(h^+) - J(h^+)\big)$ où $J(h^+)$ est calculé avec le coefficient $\theta_{i,j}^{(k)}$ auquel on a additionné ε et $J(h^-)$ est calculé avec le coefficient $\theta_{i,j}^{(k)}$ auquel on a soustrait ε, les autres coefficients restant inchangés. On donne un exemple ci – dessous. **Avec $k = 0, i = 5, j = 5$ la fonction $F5$ doit renvoyer $-1.812508e^{-9}$.**

Exemple : *Avec $k = 0$, $i = 5$ et $j = 5$ on additionne ε (que l'on peut choisir égal à 10^{-4}) au coefficient $\theta_{5,5}^{(0)}$. On appelle alors la fonction $J(LL, LL\_Teta)$. On soustrait ε à la valeur initiale de $\theta_{5,5}^{(0)}$ (Attention comme on a déjà additionné ε à l'étape d'avant on doit donc soustraire 2ε à $\theta_{5,5}^{(0)}$). On appelle une seconde fois la fonction $J(LL, LL\_Teta)$. Il n'y a plus qu'à faire la différence et diviser par 2ε comme on le fait habituellement pour calculer une dérivée numérique. Penser à additionner une dernière fois ε à $\theta_{5,5}^{(0)}$ pour qu'il retrouve sa valeur initiale.*

13. Ecrire une fonction $F6(LL, LL\_Teta, \varepsilon = 10^{-4})$ qui calcule toutes les dérivées partielles (pour toutes les valeurs possibles de k, i et j). Le résultat renvoyé par cette fonction est une liste de liste de liste avec la même structure que la variable LL_Teta puisqu'il y a autant de dérivées partielles que de coefficients $\theta_{i,j}^{(k)}$.

Attention : *Lorsque vous allez lancer le code de cette fonction vous risquez d'attendre très longtemps. C'est normal. Le code proposé est très peu efficace.*

Entraînement d'un réseau de neurone – version plus rapide – rétropropagation du gradient

La méthode proposée dans cette section est beaucoup plus rapide que la méthode de calcul précédente pour le calcul du gradient. Le gain est spectaculaire.

Le principe est de partir de l'écart entre le résultat prédit par le réseau de neurone et la réponse attendue au niveau de la dernière couche de neurone. On procède ensuite à une rétro propagation de ces écarts de la dernière à la première couche de neurone. Sans rentrer dans les explications on peut alors calculer toutes les composantes du gradient (c'est – à – dire tous les coefficients $\frac{\partial(J(h))}{\partial\theta_{i,j}^{(k)}}$). On procède de la manière suivante :

(i) *On initialise deux matrices $\Delta_{i,j}^{(0)}$ et $\Delta_{i,j}^{(1)}$ avec uniquement des zéros. Ces matrices ont les mêmes dimensions que $\theta_{i,j}^{(0)}$ et $\theta_{i,j}^{(1)}$.*

(ii) *Pour chaque image contenue dans la variable LL on calcule $Z^{(1)}$, $A^{(1)}$, $Z^{(2)}$ puis $A^{(2)}$. On reprend les notations données plus haut dans l'énoncé.*

(iii) *On calcule l'écart $\delta^{(2)} = A^{(2)} - y$*

(iv) *On calcule l'écart $\delta^{(1)}$ au niveau de la première couche de neurone à l'aide de la formule ci – dessous :*

$$\begin{cases} tmp = \left[\theta_{i,j}^{(1)}\right]^T \delta^{(2)} \\ \delta^{(1)}{}_i = tmp_i \times a_i^{(1)} \times \left(1 - a_i^{(1)}\right) \end{cases}$$

Notation : *$\left[\theta_{i,j}^{(1)}\right]^T$ est la transposée de la matrice $\left[\theta_{i,j}^{(1)}\right]$. La transposition consiste à changer les lignes en colonne. Par exemple si $A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ alors $A^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$.*

Rem : *On rappelle que $a_i^{(1)}$ sont les coordonnées de $A^{(1)}$.*

Rem : *tmp est un intermédiaire de calcul.*

Rem : *Inutile de calculer $\delta^{(0)}$ car étant au niveau de l'entrée du réseau de neurone il est nul par définition.*

(v) *On actualise alors les deux matrices $\Delta_{i,j}^{(0)}$ et $\Delta_{i,j}^{(1)}$ et on reprend à l'étape (ii).*

$$\begin{cases} \Delta_{i,j}^{(0)} = \Delta_{i,j}^{(0)} + \delta_i^{(1)} \times a_j^{(0)} \\ \Delta_{i,j}^{(1)} = \Delta_{i,j}^{(1)} + \delta_i^{(2)} \times a_j^{(1)} \end{cases}$$

(vi) *Une fois toutes les images de la variable LL traitées il suffit de diviser par $N = 5000$ les deux matrices $\Delta_{i,j}^{(0)}$ et $\Delta_{i,j}^{(1)}$ pour avoir le gradient.*

$$\frac{\partial(J(h))}{\partial\theta_{i,j}^{(k)}} = \frac{\Delta_{i,j}^{(k)}}{N}$$

14. Ecrire une fonction $F7(A0, Y, LL\_Teta)$ où $A0$ est une séquence de 401 données (correspondant à une image avec en premier élément le biais qui vaut $+1$) et Y la réponse attendue en sortie du réseau de neurone. La variable LL_Teta contient les coefficients $\theta_{i,j}^{(k)}$. Cette fonction renvoie les deux matrices $\left[\delta_i^{(1)} \times a_j^{(0)}\right]$ et $\left[\delta_i^{(2)} \times a_j^{(1)}\right]$.

Indication étape (i) : *Pour créer un tableau à deux dimensions avec uniquement des zéros on peut utiliser la syntaxe* $np.zeros\big((NbLigne, NbColonne)\big)$.

Indication étape (iv) : *Pour faire la transposée d'un tableau vous pouvez utiliser la syntaxe* $np.transpose(A)$.

Indication étape (iv) : *La module numpy permet facilement de multiplier des coordonnées de vecteurs deux à deux. Si* $a = [1,2]$ *et* $b = [3,5]$ *sont deux tableaux alors* $a * b$ *est le tableau* $[3,10]$.

Indication étape (v) : *Si* $a = [1,2]$ *et* $b = \begin{bmatrix}3\\5\end{bmatrix}$ *sont deux tableaux alors* $a * b$ *est le tableau* $\begin{bmatrix}3 & 6\\5 & 10\end{bmatrix}$. *Par défaut un tableau à une dimension est en ligne. Pour transformer un tableau* t *de* K *éléments en colonne on peut utiliser la syntaxe* $t.reshape\big((K, 1)\big)$.

15. Ecrire une fonction $F7(LL, LL\_Teta)$ où LL et LL_Teta sont les deux variables contenant les images et les coefficients $\theta_{i,j}^{(k)}$. Cette fonction renvoie le gradient de la fonction d'évaluation, c'est – à – dire toutes les dérivées partielles $\frac{\partial(J(h))}{\partial\theta_{i,j}^{(k)}}$. Le résultat peut être renvoyé sous la forme de deux tableaux à deux dimensions $\Delta_{i,j}^{(0)}$ *et* $\Delta_{i,j}^{(1)}$. **Vérifier que pour $k = 0$, $i = 5$, $j = 5$ la dérivée partielle vaut bien la même valeur que celle trouvée précédemment : $-1.812508e^{-9}$**

Il est maintenant possible de procéder efficacement à l'apprentissage d'un réseau de neurone. Vous pouvez maintenant partir de coefficients $\theta_{i,j}^{(k)}$ choisis aléatoirement calculer le gradient et modifier les coefficients $\theta_{i,j}^{(k)}$ dans le sens de $J(h)$ décroissant.
Avoir un algorithme aussi efficace pour l'apprentissage d'un réseau de neurone permet d'envisager des applications ambitieuses et même une adaptation du réseau de neurone en temps réel.

Sur apprentissage. Commentaire sur le travail effectué

En intelligence artificielle un écueil que l'on rencontre souvent est le sur apprentissage. Sans rentrer dans les détails un réseau de neurone très performant avec les données utilisées pour l'apprentissage peut s'avérer assez mauvais avec de nouveau échantillons. Dans ce genre de situations le réseau de neurone s'est « trop spécialisé ».

Pour contourner cette difficulté une recommandation est de n'utiliser qu'une partie des données pour l'apprentissage (80% environ). Les 20% de données restantes servent alors à tester la qualité de l'apprentissage.

Cette façon de procéder permet également de répondre à des questions que vous vous êtes certainement posées : Combien de couches de neurone sont nécessaires ? Combien de neurones par couche ? Questions auxquelles on a envie de répondre naïvement : Le plus grand nombre possible.

Une réponse possible est de choisir le nombre de neurone et de couche de neurone qui donnent le meilleur résultat sur les 20% de données tests après un entraînement sur les autres 80% de données.

Il est également possible pour limiter la spécialisation du réseau de neurone d'ajouter à la fonction d'évaluation une pénalité si les coefficients $\theta_{i,j}^{(k)}$ sont trop grands en valeur absolue. On aurait pu sans grande modification de l'approche proposée ajouter cette contrainte.

Traitement temps réel des images d'une webcam

Pour aller plus loin on peut utiliser le flot continu de données d'une webcam. On utilise le module *opencv* qui permet de récupérer les données issues de la caméra connectée à l'ordinateur. Le code ci – dessous est donné à titre informatif. Il traite les données en temps réel.

Python 3.5

```
def CHIFFRE(frame) :
    _H, _L = 20, 20
    ima = np.zeros((_H,_L))
    Nb = np.zeros((_H,_L))
    Dj, Di = HHH/_H, LLL/_L
    # On change la résolution de l'image
    for i in range(LLL) :
        for j in range(HHH) :
            ima[int(j/Dj),int(i/Di)] = ima[int(j/Dj),int(i/Di)] + frame[j,i,0] + frame[j,i,1] + frame[j,i,2]
            Nb[int(j/Dj),int(i/Di)] = Nb[int(j/Dj),int(i/Di)] + 1
    # On divise par Nb pour ne pas avoir de pixels surévaluée
    for i in range(_L) :
        for j in range(_H) :
            ima[j,i] = ima[j,i]/(Nb[j,i]+10**-10)
    # On met les niveaux entre 0 et 1
    # On inverse l'image (foncé devient clair et inversement)
    Min = np.min(ima)
    Max = np.max(ima)
    ima1 = np.zeros((_H,_L))
    for i in range(_L) :
        for j in range(_H) :
            ima1[j,i] = 1 - (ima[j,i]-Min)/(Max-Min)
    # On seuille l'image
    seuil = Level(list(ima1.reshape(_H*_L)),POURCENT)
    for i in range(_L) :
        for j in range(_H) :
            ima1[j,i] = 1*(ima1[j,i]>seuil)
```

```
    # On floutte l'image
    ima2 = np.zeros((_H,_L))
    for i in range(_L) :
        for j in range(_H) :
            imin, imax = max([i-1,0]), min([i+2,_L])
            jmin, jmax = max([j-1,0]), min([j+2,_H])
            kmin, kmax = max([jmin-j+1,0]), min([jmax-j+1,3])
            lmin, lmax = max([imin-i+1,0]), min([imax-i+1,3])
            ima2[j,i] = np.sum(ima1[jmin:jmax,imin:imax]*M[kmin:kmax,lmin:lmax])
    # On remet l'image entre 0 et MAX (MAX est légèrement supérieur à 1)
    Min = np.min(ima2)
    Max = np.max(ima2)
    for i in range(_L) :
        for j in range(_H) :
            ima2[j,i] = (ima2[j,i]-Min)/(Max-Min)*MAX
    # Cette dernière étape n'est peut être pas indispensable
    # On refait un seuillage pour éviter des valeurs extrèmes isolées
    # Ce seuillage est "plus doux"
    for i in range(_L) :
        for j in range(_H) :
            _ = ima2[j,i]
            ima2[j,i] = _/(AA+(1-AA)*_)
    # On fabrique à partir de l'image la liste d'entrée x
    x = []
    for i in range(_L) :
        for j in range(_H) :
            x.append(ima2[j,i])
    # On calcule la sortie du réseau de neurone
    res = h(x,L_Teta)
    print(res)
    c = 0
    Max = res[0]
    for i in range(10) :
        if res[i] > Max :
            Max = res[i]
            c = i
    print("Le chiffre est : ",c,res[c])
    out = np.zeros((PPAS*_H,PPAS*_L))
    for i in range(_L) :
        for j in range(_H) :
            for k in range(PPAS) :
                for l in range(PPAS) :
                    _ = ima2[j,i]
                    if _ < 1 :
                        out[j*PPAS+k,i*PPAS+l] = _
                    else :
                        out[j*PPAS+k,i*PPAS+l] = 0.9999
    return out, c, res[c]
############### PARAMETRE ##############
POURCENT = 90
MAX = 1.2
AA = 0.2
PPAS = 3
M = np.array([[0.20,0.40,0.20],
              [0.40,1.00,0.40],
              [0.20,0.40,0.20]])
HH, LL = 480, 640
HHH, LLL = 180, 180
_H, _L = 20, 20
############### PARAMETRE ##############

# Il faudra ajouter une entrée à chaque couche (sauf la dernière couche)
#_l = [10,10,5,4] # nombre de neurone par couche
_l = [400,25,10]
L_Teta = []
for i in range(len(_l)-1) :
    L_Teta.append(np.zeros((_l[i+1],_l[i]+1)))
#######################################
for l in range(len(LL_Teta)) :
    _H, _L = len(LL_Teta[l]), len(LL_Teta[l][0])
    for j in range(_H) :
        for i in range(_L) :
            L_Teta[l][j,i] = LL_Teta[l][j][i]
#######################################
```

```
cv2.namedWindow('frame')
cap = cv2.VideoCapture(1)
while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    j0 = HH//2-HHH//2
    j1 = HH//2+HHH//2
    for i in range(LL//2-LLL//2,LL//2+LLL//2) :
        frame[j0,i,0] = 255 #0:Bleu 1:Vert 2:Rouge
        frame[j1,i,0] = 255
    j2 = HH//2-HHH//2-1
    j3 = HH//2+HHH//2+1
    for i in range(LL//2-LLL//2-1,LL//2+LLL//2+1) :
        frame[j2,i,2] = 255
        frame[j3,i,2] = 255
    i0 = LL//2-LLL//2
    i1 = LL//2+LLL//2
    for j in range(HH//2-HHH//2,HH//2+HHH//2) :
        frame[j,i0,0] = 255
        frame[j,i1,0] = 255
    i2 = LL//2-LLL//2-1
    i3 = LL//2+LLL//2+1
    for j in range(HH//2-HHH//2-1,HH//2+HHH//2+1) :
        frame[j,i2,2] = 255
        frame[j,i3,2] = 255

    out, c, re = CHIFFRE(frame[j0:j1+1,i0:i1+1])

    _L, _H = 20, 20
    for i in range(PPAS*_L) :
        for j in range(PPAS*_H) :
            _ = out[j,i]
            _ = int(_*256)
            frame[j,i,0] = 0
            frame[j,i,1] = 0
            frame[j,i,2] = _

    cv2.putText(frame,"Le chiffre est : "+str(c)+"("+str(int(100*re))+"%)",
                (150,120),
                cv2.FONT_HERSHEY_SIMPLEX,
                1,
                255,2)

    # Display the resulting frame
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Il y a tout un travail de mise en forme des données : seuillage, effet de flou, éclaircissement des zones les plus sombres de l'image, … pour rendre les données exploitables par le réseau de neurone. Le code correspondant n'est pas présenté.

Installation d'OpenCV

J'ai trouvé un site internet où l'installation d'opencv est bien expliquée. C'est à l'adresse :

Sécurisé | https://chrisconlan.com/installing-python-opencv-3-windows/

Install Python OpenCV 3 on Windows with Anaconda Environments

May 31, 2017 By Chris Conlan — 35 Comments

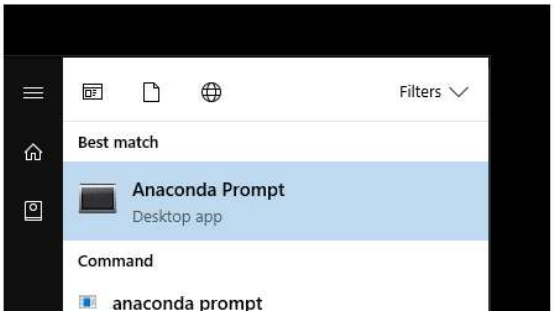
Recently, Satya Mallick, founder of learnopencv.com, posted an impressive (but complicated) method for installing OpenCV 3 on Windows that supports both the C++ and Python API's. Since a lot of users will be interested in solely Python OpenCV, I figured it would be helpful to post a relatively quick method for getting Python OpenCV 3 up and running on Windows.

1) Install Anaconda

Head over to continuum.io/downloads/ and install the latest version of Anaconda. Make sure to install the "Python 3.6 Version" for the appropriate architecture. Install it with the default settings.

2) Open the Anaconda Prompt

Anaconda installs a few programs on your computer when you run the installer. These include the Anaconda Navigator, Anaconda Cloud, Spyder, and the Anaconda Prompt. Search in your Windows taskbar for the Anaconda Prompt. This is a modified version of the Windows Command Prompt that support specific Anaconda commands. All of the code we discuss in these instructions will be run directly in the Anaconda Prompt.



3) Create an Anaconda Environment

This section is essentially a Windows distillation of https://conda.io/docs/using/envs.html#create-an-environment.

Anaconda environments are similar to a Python **virtualenv**, except they use Anaconda's superb package managers. When we install OpenCV 3, we will do so in an Anaconda environment that uses specifically Python 3.5, and that version of Python will only be accessible through the environment. Below are some basics of Anaconda environment management.

```
### Basics of Anaconda environment management ###
# Creating an environment
conda create --name myNewEnv python=x.x.x


# Activating an environment
activate myNewEnv

# Deactivating an environment
deactivate myNewEnv

# Listing environments
conda info --envs

# Removing an environment
conda remove --name myNewEnv --all
```

Run the following to create and activate a new Anaconda environment for Python 3.5. We use Python 3.5 for compatibility with the OpenCV 3 distribution we will be using.

```
conda create --name myWindowsCV python=3.5
activate myWindowsCV
```

Now, you should see "(myWindowsCV)" prepended to the command line. When the environment's name is prepended to the command line, the environment is active, and Anaconda has modified the $PATH variables of the console to point to various directories in anaconda3/envs/myWindowsCV. Running "python –version" should now return Python 3.5.* as opposed to your system Python version.

```
(myWindowsCV) C:\Users\Chris>python --version
Python 3.5.2 :: Continuum Analytics, Inc.
```

4) Install OpenCV 3 and Dependencies

In the Anaconda Prompt, with your Anaconda environment active, run:

```
conda install numpy
conda install anaconda-client
conda install --channel https://conda.anaconda.org/menpo opencv3
```

Now, enter Python and check that OpenCV 3 is installed.

```
(myWindowsCV) C:\Users\Chris>python
>>>import cv2
>>>cv2.__version__
'3.1.0'
```