

Reconnaissance de pathologies sur des radiographies grâce à un réseau de neurones

Robin BRESSOUD (n°14641)

2021

Table des matières

1 Présentation

2 Réseau de neurones

- Définitions
- Retropropagation
- Deux exemples

3 Application à la radiographie

- Enjeux
- Résolution du problème
- Traitement des échantillons
- Résultats

4 Conclusion

5 Annexes

Table des matières

1 Présentation

2 Réseau de neurones

- Définitions
- Retropropagation
- Deux exemples

3 Application à la radiographie

- Enjeux
- Résolution du problème
- Traitement des échantillons
- Résultats

4 Conclusion

5 Annexes

Présentation du problème

Nous disposons de radiographies du thorax de plusieurs milliers de patients comme sur les images suivantes. Nous souhaitons alors tenter d'implémenter une solution informatique afin de confirmer l'avis médical voire l'accélérer.



Fig.1



Fig.2

Une idée

Nous disposons de la base de données plubié par Xaosong Wang qui contient plus de 100 000 échantillons. Il s'agit donc de concevoir un réseau de neurones capable de reconnaître si le patient dont on donne la radiographie du buste est atteint d'une pathologie parmi différentes possibles.

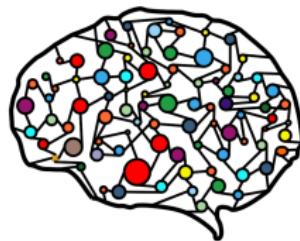


Table des matières

- 1 Présentation
- 2 Réseau de neurones
 - Définitions
 - Retropropagation
 - Deux exemples
- 3 Application à la radiographie
 - Enjeux
 - Résolution du problème
 - Traitement des échantillons
 - Résultats
- 4 Conclusion
- 5 Annexes

Chez l'Homme

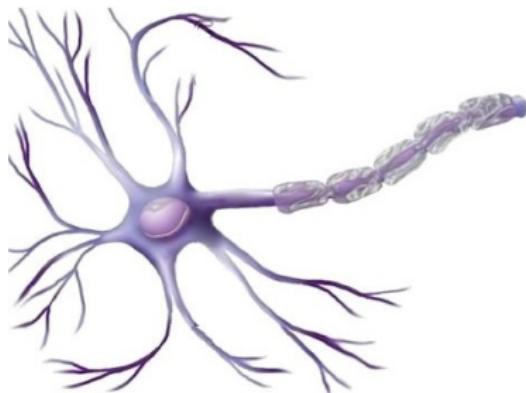
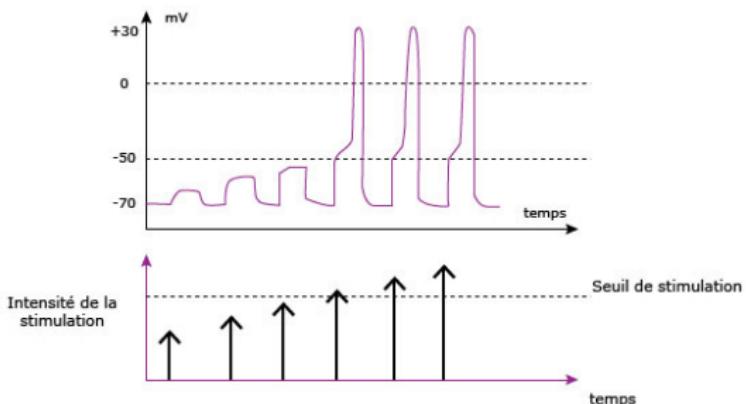


Schéma de neurone



Seuil de stimulation du neurone

Le perceptron

Définition: Le perceptron

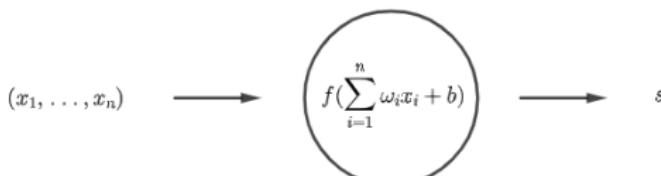
Soient $n \in \mathbb{N}, \omega = (\omega_1, \dots, \omega_n) \in \mathbb{R}^n$

Soient $b \in \mathbb{R}$

Soit une fonction d'activation $f : \mathbb{R} \rightarrow \{0, 1\}$ telle que

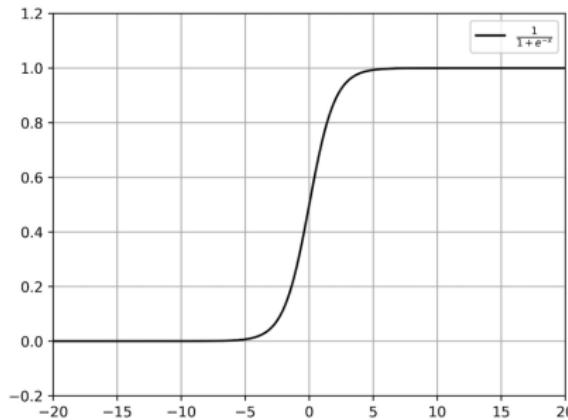
$$f(x_1, \dots, x_n) = \begin{cases} 0 & \text{si } \sum_{i=1}^n \omega_i x_i + b < 0 \\ 1 & \text{sinon} \end{cases}$$

On définit ainsi un perceptron à n -entrées. La sortie s du perceptron est le résultat de la fonction f lorsqu'on lui donne une entrée.



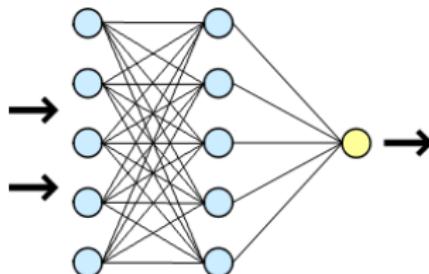
Le perceptron

Remarque : En règle générale dans les codes, on adopte une approche continue et on pose $f(x) = \frac{1}{1+e^{-x}}$ pour $x = \sum_{i=1}^n \omega_i x_i - \theta$. Cette fonction se nomme sigmoïde.



Réseau de neurones

Si le perceptron peut faire la distinction entre deux informations simples, il ne peut pas apprendre traiter des cas plus compliqués. Pour remédier à cela on peut connecter plusieurs perceptrons entre eux pour former un réseau de neurones.

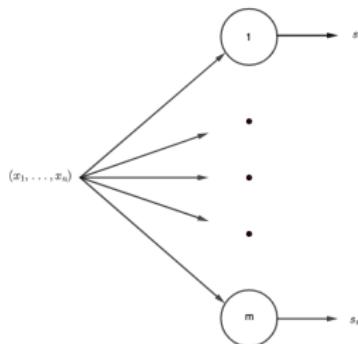


Remarque: On supposera que la fonction d'activation f de tous les neurones est la même et se trouve être la fonction sigmoïde définie précédemment.

Couches - Définition

Définition : Une couche

Une couche de taille m est un m -uplet de perceptrons à k -entrées. La sortie s d'une couche obtenue à partir d'un k -uplet (x_1, \dots, x_k) est le m -uplet des m sorties des perceptrons.



Réseau de neurones - Définition

Définition : Réseau de neurones

Un réseau de neurones à n couches est un n -uplet de couches (C_1, \dots, C_n) de tailles compatibles telles que l'entrée de la $k+1$ -ième couche soit la sortie de la k -ième.

La première couche est nommée *couche d'entrée*.

La dernière couche est la *couche de sortie*.

Correction de l'erreur pour le perceptron de Rosenblatt

Pour que le neurone soit capable de reconnaître une information qui peut prendre deux valeurs on l'entraîne et on corrige les poids selon la formule suivante :

$$\forall i, \omega_i \leftarrow \omega_i + t(s_a - s)x_i$$

t le taux d'apprentissage,
 s_a la sortie attendue

L'algorithme du gradient

Algorithme du gradient

Soit $n \in \mathbb{N}, \varepsilon > 0$.

On munit \mathbb{R}^n de son produit scalaire canonique.

Soit f une fonction différentiable de $\mathbb{R}^n \rightarrow \mathbb{R}$.

Soit x_0 la valeur initiale, t le taux d'apprentissage.

Supposons x_0, \dots, x_k construits.

Si $\|\nabla f(x_k)\| \leq \varepsilon$, on s'arrête.

Sinon on pose $x_{k+1} = x_k - t \nabla f(x_k)$

Conventions & notations

On a :

- n le nombre de couches
- f la fonction d'activation globale
- $z = (z_1, \dots, z_p)$ la sortie théorique
- $x = (x_1, \dots, x_q)$ l'entrée
- t le taux d'apprentissage
- $\omega_{\alpha\beta}^m$ le β -ième poids du neurone α de la couche m . C'est celui qui multiplie la sortie du β -ième neurone de la couche précédente
- x_k^m la sortie du k -ième neurone de la couche m
- b_α^m le biais du neurone α de la couche m

Cas du réseau de neurones

Pour le réseau de neurones on prend une fonction dite fonction de coût, notée E .

$$E(x) = \sum_{i=1}^p (x_i^n - z_i)^2$$

Lorsqu'on applique l'algorithme du gradient on a donc :

$$\omega_{\alpha\beta}^m \leftarrow \omega_{\alpha\beta}^m - t \frac{\partial E}{\partial \omega_{\alpha\beta}^m}(x)$$

Formule de rétropropagation

On note e_α^m l'erreur à la couche m définie comme ce qui suit

$$\frac{\partial E}{\partial \omega_{\alpha\beta}^m}(x) = x_\beta^{m-1} e_\alpha^m$$

On effectue le calcul par récurrence descendante pour trouver

$$e_i^n = f'(\sum_k w_{ik}^n x_k^{n-1} + b_i^n)(x_i^n - z_i)$$

$$e_r^{n-p-1} = f'(\sum_k w_{ik}^{n-p-1} x_k^{n-p-2} + b_r^{n-p-1}) \sum_i w_{ir}^{n-p} e_i^{n-p}$$

Remarque : Il s'agit bien d'une rétropropagation comme l'erreur de la couche $k-1$ dépend de celle de la couche k .

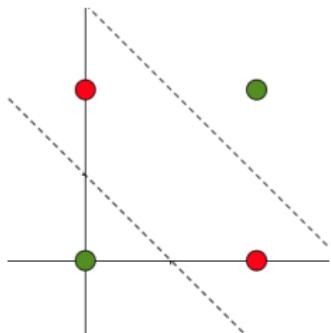
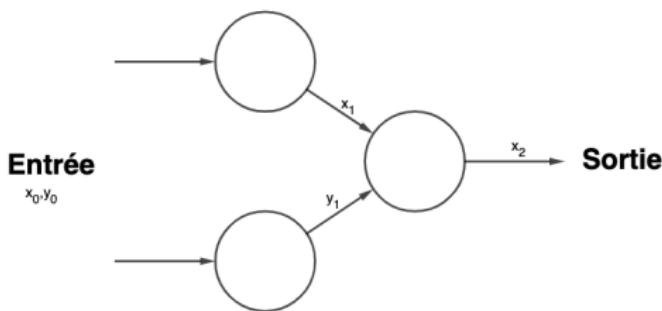
Premier exemple - le XOR

Le XOR est une opération logique simple définie par la table suivante

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Graphiquement

Un seul perceptron peut séparer l'espace en deux par un hyperplan or ici ce n'est pas possible. On peut donc faire le réseau de neurones simple suivant :



Mes exemples de code

```
def forward(x0,y0,a,b,c):
    x1 = sigmo(a[0]*x0 + a[1]*y0 + a[2])
    y1 = sigmo(b[0]*x0 + b[1]*y0 + b[2])
    x = sigmo(c[0]*x1 + c[1]*y1 + c[2])
    return (x1,y1,x)
```

```
def erreur(x0,y0,s,a,b,c):
    x1,y1,x = forward(x0,y0,a,b,c)
    es = s - x
    e0 = es*c[0]
    e1 = es*c[1]
    return (es,e0,e1)
```

```
def retropropagation(x0,y0,s,a,b,c):
    x1,y1,x = forward(x0,y0,a,b,c)
    es,e0,e1 = erreur(x0,y0,s,a,b,c)
    x1_p = sigmo_prime(a[0]*x0 + a[1]*y0 + a[2])
    y1_p = sigmo_prime(b[0]*x0 + b[1]*y0 + b[2])
    x_p = sigmo_prime(c[0]*x1 + c[1]*y1 + c[2])
    #Modification correction
    t = 1
    a[0] = a[0] + t*x0*e0*x_p*x1_p
    a[1] = a[1] + t*y0*e0*x1_p*x_p
    a[2] = a[2] + t*e0*x1_p*x_p

    b[0] = b[0] + t*x0*e1*x_p*y1_p
    b[1] = b[1] + t*y0*e1*x_p*y1_p
    b[2] = b[2] + t*e1*x_p*y1_p

    c[0] = c[0] + t*x1*es*x_p
    c[1] = c[1] + t*y1*es*x_p
    c[2] = c[2] + t*es*x_p
```

Résultats

On a 89% de réussite avec cet algorithme de manière empirique en entraînant 50 000 fois le réseau de neurones défini précédemment avec un taux d'apprentissage $t = 1$. Avec une approche non continue comme dans la définition, le système ne converge pas. Avec un réseau de deux couches de 4 neurones et une couche de 1 neurone on a 98% de réussite.

Second exemple - Reconnaissance de chiffres

La base de données du MNIST (Modified ou Mixed National Institute of Standards and Technology) est composée de chiffres manuscrits.

Le format est 28×28 pixels



Code - Class layer

Class	Methode	Arguments	Sortie
Layer	__init__	nb_input $nb_neurone$ $f_{activation}$ $f'_{activation}$	<i>None</i>
	forward	$input$	Sortie de la couche
	erreur	err_couche_sup w_couche_sup	Liste des erreurs sur une couche
	correction		<i>None</i>

Code - Class Network

Class	Methode	Arguments	Sortie
Network	<code>__init__</code>	<code>nb_input</code> <code>nb_neurone</code> <code>nb_couches</code> <code>nb_sortie</code> $f_{activation}$ $f'_{activation}$	<code>None</code>
	<code>res</code>	<code>input</code>	Sortie du réseau
	<code>retropropagation</code>	<code>resultat_theorique</code>	<code>None</code>

Résultats

Malgré plusieurs tentatives, pas de résultats positifs.

Problèmes :

- Pas d'optimisation sur le code : gros calculs à faire beaucoup de fois
- L'implémentation a un problème : il aurait fallu ajouter des couches avec de moins en moins de neurones au fur et à mesure. Cela permettrait de réduire le nombre de calculs.

Avec d'autres types de réseaux et plus de tests il est possible de dépasser les 99% de réussite.

Table des matières

- 1 Présentation
- 2 Réseau de neurones
 - Définitions
 - Retropropagation
 - Deux exemples
- 3 Application à la radiographie
 - Enjeux
 - Résolution du problème
 - Traitement des échantillons
 - Résultats
- 4 Conclusion
- 5 Annexes

Enjeux

Reconnaitre différentes maladies :

- Cardiomégalie (cardiaque)
- Emphysème (pulmonaire)
- Épanchement pleural (pulmonaire)
- Hernie
- Infiltration
- Syndrôme MASS

Utilisation de modules

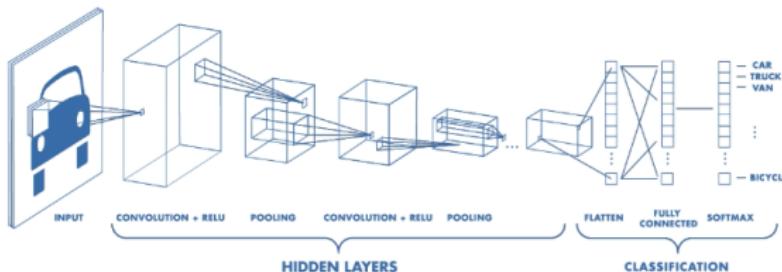
Pour simplifier le problème, on utilise TensorFlow et le cours *AI for Medical Diagnosis* de DeepLearning.AI. Cela permet d'avoir un réseau pré-entraîné :

- Permet de passer au dessus du problème de puissance de calcul.
- Permet d'avoir un réseau de neurones plus performant : un réseau de neurones convolutifs. Ce dernier est corrigé à l'aide d'une fonction d'entropie et non une fonction de coût comme précédemment.

Le CNN

Un Convolutional Neural Network se décompose en étapes :

- étape de convolution : apprend à filtrer les principales informations par rétropropagation du gradient.
- étape de pooling : réduit la taille de l'image.
- étape de correction RELU : applique RELU à la sortie
- étape fully-connected : réseau de neurones classique



Fonction de cross entropie

$$\rightarrow L(z, y) = -(y \log(z) + (1 - y) \log(1 - z))$$

On utilise alors cette fonction de la manière suivante

$$L_{mass}(X, y) = \begin{cases} -\log(P(Y = 1|X)) & \text{si } y = 1 \\ -\log(P(Y = 0|X)) & \text{si } y = 0 \end{cases}$$

où $P(Y = 1|X)$ est la probabilité que ce soit un syndrôme de MASS sachant que l'image donnée en possède un.

→ Après avoir donné une image avec le(s) symptôme(s) on pose :

$$L(X, y) = \sum_{s \in \text{symptômes}} L_s(X, y_s)$$

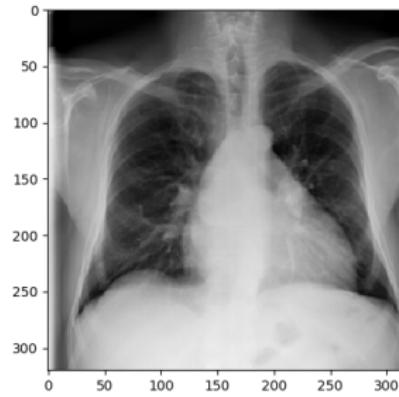
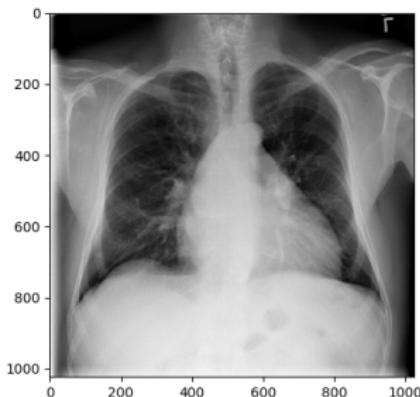
C'est la perte totale. On corrige de la manière suivante :

$$\omega \leftarrow \omega - t \frac{\partial L}{\partial \omega}$$

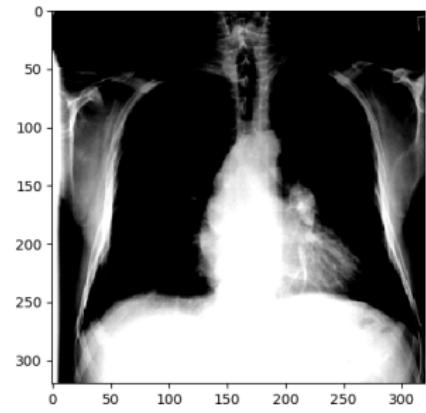
Redimensionage & normalisation

On réduit la taille en prenant la moyenne d'une zone comme valeur.

On normalise de la manière suivante : $x \leftarrow \frac{x-\mu}{\sigma}$

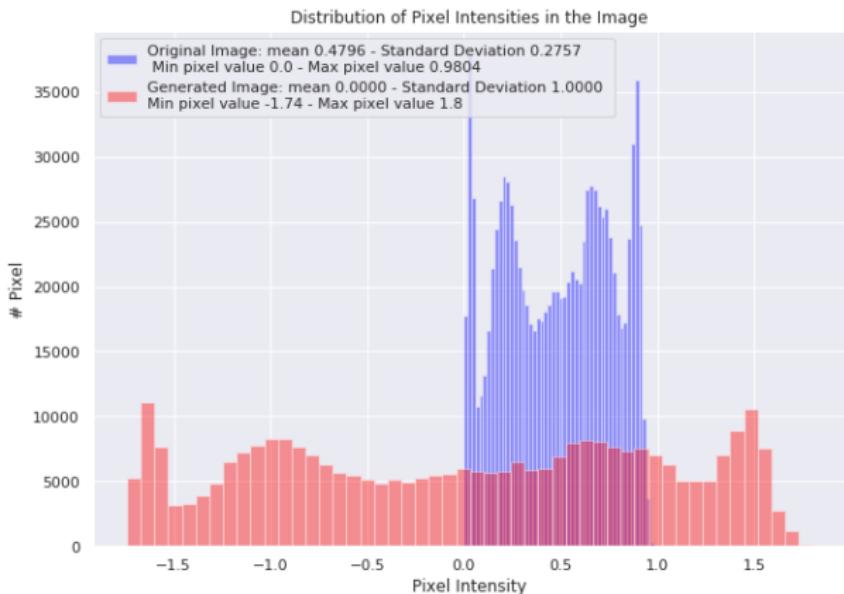


$$\mu = 0.50, \sigma = 0.25$$



$$\mu = 0.0, \sigma = 1.0$$

Redimensionnage & normalisation



Source : AI for Medical Diagnosis par deeplearning.ai

Nombre d'échantillons

Pour augmenter le nombre d'échantillons on peut utiliser diverses méthodes :



Rotation



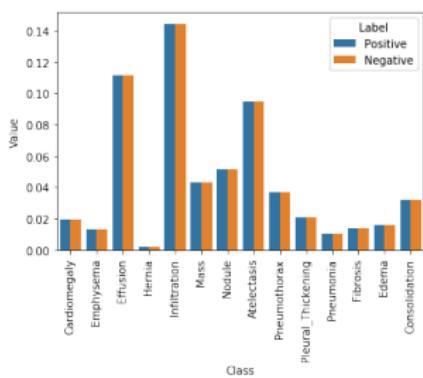
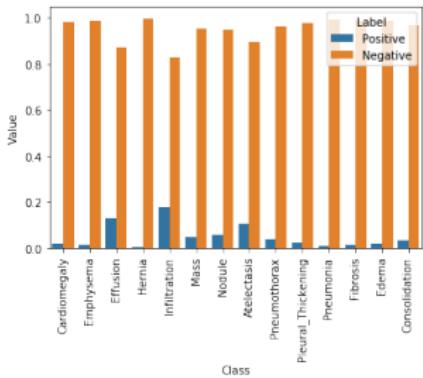
Inversion
horizontale



Zoom

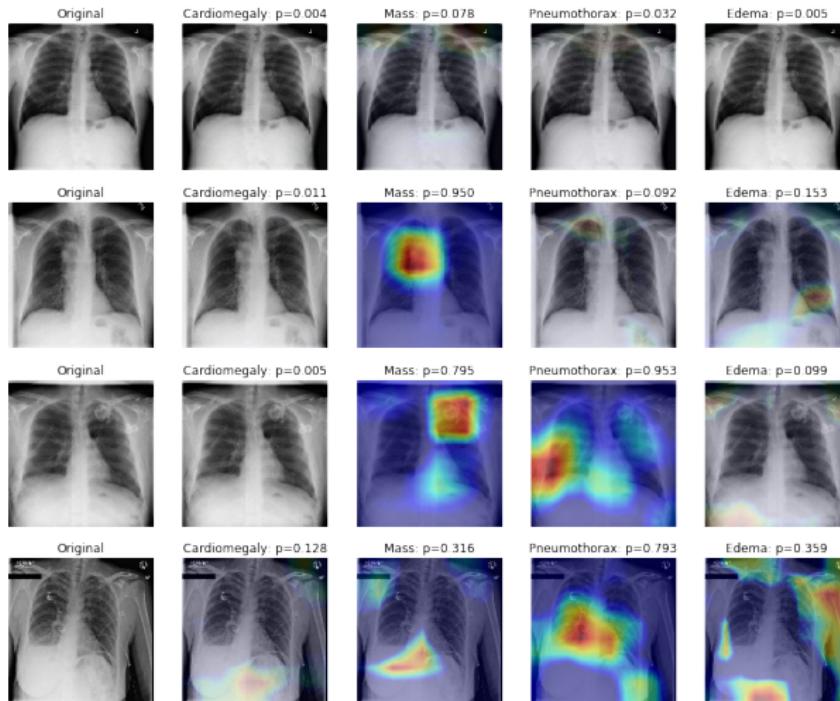
Intérêt : éviter d'avoir des faux positifs et surtout faux négatifs à cause d'un surplus d'images de patients sains.

Nombre d'échantillons - Ajout de poids de correction



On corrige $L(x, y) = -(w_p y \log(f(x)) + w_n(1 - y) \log(1 - f(x)))$
 afin que $w_p \cdot freq_{positive} = w_n \cdot freq_{negative}$
 (par exemple : $w_p = \frac{\text{nombre de cas negatifs}}{\text{nombre de cas}}$ et $w_n = \frac{\text{nombre de cas positifs}}{\text{nombre de cas}}$)

Résultats



Quelques résultats illustrés avec GradCam (module qui regarde sur quoi convergent les neurones dans l'image)

Résultats

Pathology	Radiologists (95% CI)	Algorithm (95% CI)	Algorithm – Radiologists Difference (99.6% CI) ^a	Advantage
Atelectasis	0.808 (0.777 to 0.838)	0.862 (0.825 to 0.895)	0.053 (0.003 to 0.101)	Algorithm
Cardiomegaly	0.888 (0.863 to 0.910)	0.831 (0.790 to 0.870)	-0.057 (-0.113 to -0.007)	Radiologists
Consolidation	0.841 (0.815 to 0.870)	0.893 (0.859 to 0.924)	0.052 (-0.001 to 0.101)	No difference
Edema	0.910 (0.886 to 0.930)	0.924 (0.886 to 0.955)	0.015 (-0.038 to 0.060)	No difference
Effusion	0.900 (0.876 to 0.921)	0.901 (0.868 to 0.930)	0.000 (-0.042 to 0.040)	No difference
Emphysema	0.911 (0.866 to 0.947)	0.704 (0.567 to 0.833)	-0.208 (-0.508 to -0.003)	Radiologists
Fibrosis	0.897 (0.840 to 0.936)	0.806 (0.719 to 0.884)	-0.091 (-0.198 to 0.016)	No difference
Hernia	0.985 (0.974 to 0.991)	0.851 (0.785 to 0.909)	-0.133 (-0.236 to -0.055)	Radiologists
Infiltration	0.734 (0.688 to 0.779)	0.721 (0.651 to 0.786)	-0.013 (-0.107 to 0.067)	No difference
Mass	0.886 (0.856 to 0.913)	0.909 (0.864 to 0.948)	0.024 (-0.041 to 0.080)	No difference
Nodule	0.899 (0.869 to 0.924)	0.894 (0.853 to 0.930)	-0.005 (-0.058 to 0.044)	No difference
Pleural thickening	0.779 (0.740 to 0.809)	0.798 (0.744 to 0.849)	0.019 (-0.056 to 0.094)	No difference
Pneumonia	0.823 (0.779 to 0.856)	0.851 (0.781 to 0.911)	0.028 (-0.087 to 0.125)	No difference
Pneumothorax	0.940 (0.912 to 0.962)	0.944 (0.915 to 0.969)	0.004 (-0.040 to 0.051)	No difference

Abbreviations: AUC, area under the receiver operating characteristic curve; CI, confidence interval.

<https://doi.org/10.1371/journal.pmed.1002686.t001>

Il n'y a pas d'écart de plus de 10 à 20 % par rapport à la prediction des médecins.

Il n'y a pas vraiment de meilleure prédition : les sorties attendues dans la phase d'entraînement sont données par les médecins.

Table des matières

1 Présentation

2 Réseau de neurones

- Définitions
- Retropropagation
- Deux exemples

3 Application à la radiographie

- Enjeux
- Résolution du problème
- Traitement des échantillons
- Résultats

4 Conclusion

5 Annexes

Conclusion

Les réseaux de neurones permettent de donner la probabilité qu'un patient soit atteint d'une pathologie visible par radiographie, et ce avec une précision presque médicale. Cela représente un enjeu pour l'avenir permettant la sécurité des individus en assistant les médecins et donnant un contre avis.

Merci de votre attention !

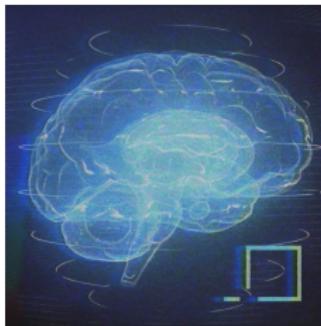


Table des matières

- 1 Présentation
- 2 Réseau de neurones
 - Définitions
 - Retropropagation
 - Deux exemples
- 3 Application à la radiographie
 - Enjeux
 - Résolution du problème
 - Traitement des échantillons
 - Résultats
- 4 Conclusion
- 5 Annexes

XOR 1

```
import numpy as np
import numpy.random as rd

sigmo = lambda x:1/(1+np.exp(-x))
sigmo_prime = lambda x:(np.exp(-x))/((1+np.exp(-x))**2)

def forward(x0,y0,a,b,c):
    x1 = sigmo(a[0]*x0 + a[1]*y0 + a[2])
    y1 = sigmo(b[0]*x0 + b[1]*y0 + b[2])
    x = sigmo(c[0]*x1 + c[1]*y1 + c[2])
    return (x1,y1,x)

def erreur(x0,y0,s,a,b,c):
    x1,y1,x = forward(x0,y0,a,b,c)
    es = s - x
    e0 = es*c[0]
    e1 = es*c[1]
    return (es,e0,e1)
```

XOR 2

```
def retropropagation(x0,y0,s,a,b,c):  
    x1,y1,x = forward(x0,y0,a,b,c)  
    es,e0,e1 = erreur(x0,y0,s,a,b,c)  
    x1_p = sigmo_prime(a[0]*x0 + a[1]*y0 + a[2])  
    y1_p = sigmo_prime(b[0]*x0 + b[1]*y0 + b[2])  
    x_p = sigmo_prime(c[0]*x1 + c[1]*y1 + c[2])  
    #Modification correction  
    t = 1  
  
    a[0] = a[0] + t*x0*e0*x_p*x1_p  
    a[1] = a[1] + t*y0*e0*x1_p*x_p  
    a[2] = a[2] + t*e0*x1_p*x_p  
  
    b[0] = b[0] + t*x0*e1*x_p*y1_p  
    b[1] = b[1] + t*y0*e1*x_p*y1_p  
    b[2] = b[2] + t*e1*x_p*y1_p  
  
    c[0] = c[0] + t*x1*es*x_p  
    c[1] = c[1] + t*y1*es*x_p  
    c[2] = c[2] + t*es*x_p  
  
    return None
```

XOR 3

```
L_XOR = [[(0, 0), 0], [(0, 1), 1], [(1, 0), 1], [(1, 1), 0]]  
L_NOT_XOR = [[(0, 0), 1], [(0, 1), 0], [(1, 0), 0], [(1, 1), 1]]  
  
def initialisation():  
    a = rd.random(3)  
    b = rd.random(3)  
    c = rd.random(3)  
    return (a,b,c)  
  
def entrainement(L):  
    a,b,c = initialisation()  
    for i in range(50000):  
        echantillon = L[rd.randint(0,4)]  
        x0,y0 = echantillon [0]  
        s = echantillon [1]  
        retropropagation(x0,y0,s,a,b,c)  
    test = [int(round(forward(x0, y0, a, b, c)[2])) for [(x0, y0), s] in L]  
    return (test)
```

Réseau de neurones 1

```
import numpy as np
import matplotlib.pyplot as plt
import random
import time
import csv

DATA = ''

#####
# RESEAU DE NEURONES #
#####

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))
def sigmoid_prime(x):
    return (np.exp(-x))/((1+np.exp(-x))**2)

lr = 1
```

Réseau de neurones 2

```
class Layer:  
    def __init__(self, nb_input, nb_neurone, fun_activ=sigmoid, deriv_fun_activ=sigmoid_p  
                self.weight = np.random.randn(nb_neurone, nb_input) # poids aleatoires  
                self.biais = np.random.randn(nb_neurone) # constante  
                self.fun = fun_activ # fonction utilisee  
                self.der = deriv_fun_activ # sa derivee  
                self.lr = taux_apprentissage  
                self.out = np.zeros(nb_neurone)  
                self.err = np.zeros(nb_neurone)  
                self.neur = nb_neurone  
                self.input = nb_input  
                self.list_input = []  
    def forward(self, input):  
        self.out = self.fun(np.dot(self.weight, input) + self.biais)  
        self.list_input = input  
        return self.out
```

Réseau de neurones 3

```
def erreur(self, liste_erreur_couche_plus, poids_couche_plus):
    L_erreur = []
    for r in range(self.neur):
        somme1 = 0
        somme2 = 0
        for k in range(self.input):
            somme1 += self.weight[r][k] * self.list_input[k]
        for i in range(len(poids_couche_plus)):
            somme2 += liste_erreur_couche_plus[i] * poids_couche_plus[i][r]
        L_erreur.append(self.der(somme1 + self.biais[r]) * somme2)
    self.err = np.array(L_erreur)
    return self.err
def correction(self):
    for r in range(self.neur):
        for i in range(self.input):
            self.weight[r][i] -= self.lr * self.err[r] * self.list_input[i]
            self.biais[r] -= self.lr * self.err[r]
    return None
```

Réseau de neurones 4

```
class Network:  
    def __init__(self, nb_input, nb_couches, nb_neurone, nb_sortie=1, fun_activ=sigmoid, c  
        L = [Layer(nb_input, nb_neurone, fun_activ, deriv_fun_activ, taux)]  
        self.couche = L # Liste de couches  
        for i in range(1, nb_couches):  
            self.couche.append(Layer(nb_neurone, nb_neurone, fun_activ, deriv_fun_activ, taux))  
        self.exit = Layer(nb_neurone, nb_sortie, fun_activ, deriv_fun_activ, taux)  
        self.nb_exit = nb_sortie  
        self.len = nb_couches # Longueur  
        self.neur = nb_neurone # Largeur  
        self.out = np.zeros((nb_couches, nb_neurone))  
        self.func = fun_activ # fonction utilisee  
        self.deriv = deriv_fun_activ # sa derivee  
    def res(self, input):  
        # Input sous forme de liste  
        self.out[0] = (self.couche[0]).forward(input)  
        for i in range(1, self.len):  
            self.out[i] = (self.couche[i]).forward(self.out[i-1])  
        return (self.exit).forward(self.out[self.len - 1])  
        # Renvoie un liste de nb_sortie element
```

Réseau de neurones 5

```
def retropropage(self, expect):
    #Recuperation de l'erreur de la sortie
    for i in range(self.nb_exit):
        intermediaire = 0
        for k in range(self.neur):
            intermediaire += (self.exit).weight[i][k] * self.out[self.len - 1][k]
        intermediaire += (self.exit).biais[i]
        (self.exit).err[i] = self.deriv(intermediaire) * ((self.exit).out[i] - expect)
        (self.exit).correction()
    # Retropagation
    (self.couche[self.len - 1]).erreur((self.exit).err, (self.exit).weight)
    (self.couche[self.len - 1]).correction()
    for i in range(self.len - 2, -1, -1):
        (self.couche[i]).erreur((self.couche[i+1]).err, (self.couche[i+1]).weight)
        (self.couche[i]).correction()
    return None
```

Réseau de neurones 6

```
#####
# DONNEES #
#####

def nbr_to_binary_list(n):
    L = []
    for i in range(10):
        if i == n: L.append(1)
        else: L.append(0)
    return L
```

Réseau de neurones 7

```
def load_mnist_data():
    """ Telecharge et prepare la base MNIST"""

    # transformer en list
    with open(DATA, newline='') as f:
        reader = csv.reader(f)
        donnees = [list(map(int, rec)) for rec in reader]

    X,Y,Z = [],[],[]

    for i in range(0, len(donnees)):
        val = donnees[i].pop()
        Y.append((nbr_to_binary_list(val)))
        X.append(donnees[i])
        Z.append([x/255 for x in donnees[i]])

    # X est une matrice de taille (70000, 784)
    # X[0] est la premiere image de la liste
    # X[0][0] est le premier pixel de cette image
    # Y est une matrice de taille (70000,)
    # Y[0] est la valeur representee par l'image X[0]

    return X,Y,Z
```

Réseau de neurones 8

```
#####
# LE CODE A EXECUTER #
#####

# N'excuter qu'une seul fois la ligne ci-dessous quand on ouvre python
X,res,donnees = load_mnist_data()

# Parametres a modifier
nbr_couche = 5
nbr_neurone = 10
fonction = sigmoid
fonction_deriv = sigmoid_prime

def egal(tab1,tab2):
    res = True
    for i in range(len(tab1)):
        res = res and (tab2[i] == int(round(tab1[i])))
    return res
```

Réseau de neurones 9

```
def test_reseau(couche ,neurones ,nb_train ,nb_test ,func ,func_prime):
    Reseau = Network(nb_input = 784,nb_couches = couche , nb_neurone = neurones , nb_sortie = 1)
    # Training
    pourcent = 1
    for i in range(nb_train):
        j = random.randint(0,69999)
        oublie = Reseau.res(donnees[j])
        Reseau.retropropage(res[j])
        if (i/nb_train) >= (pourcent/100):
            if pourcent%10 == 0: print("{}%".format(pourcent))
        pourcent+=1
    # Test
    reussi = 0
    for i in range(nb_test):
        j = random.randint(0,69999)
        if egal(Reseau.res(donnees[j]) , res[j]):
            reussi += 1
    return (couche ,neurones ,nb_train ,nb_test ,reussi)

# A executer pour faire un test
test_reseau(nbr_couche ,nbr_neurone ,42000,1000,fonction ,fonction_deriv)
```

Réduction 1

```
from math import *
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg

dir = ""
img = mpimg.imread(dir)

def stat(image):
    return (np.mean(image), np.std(image))

def f(x,m,sigma): return (x-m)/sigma

def redimensionage(image, taille):
    """ On donne une image carree pour en ressortir une image carree"""
    res = np.zeros((taille,taille))
    n = len(image)
    pas = floor(n/taille)
    for i in range(0,taille):
        for j in range(0,taille):
            extrait = image[i*pas:min((i+1)*pas, n),j*pas:min((j+1)*pas, n)]
            res[i][j] = np.mean(extrait)
    return res
```

Réduction 2

```
def reduit(image):
    m,sigma = stat(image)
    n = len(image)
    res = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            res[i][j] = f(image[i][j], m, sigma)
    return res

def aff(image ,taille):
    rd = redimensionage(image ,taille)
    m1,s1 = stat(rd)
    mi1,ma1 = np.amin(rd),np.amax(rd)
    rdu = reduit(rd)
    m2,s2 = stat(rdu)
    mi2,ma2 = np.amin(rdu),np.amax(rdu)

    plt.subplot(311)
    plt.imshow(image, cmap=plt.cm.gray ,vmin = 0., vmax = 1.)
    plt.subplot(312)
    plt.imshow(rd,cmap='gray' ,vmin = 0., vmax = 1.)
    plt.title(label = "{}-{}-{}-{}-{}".format(m1,s1,mi1,ma1))
    plt.subplot(313)
    plt.imshow(rdu,cmap='gray' ,vmin = 0., vmax = 1.)
    plt.title(label = "{}-{}-{}-{}-{}".format(m2,s2,mi2,ma2))
    plt.show()
    return None
```