Majdoub Khalid
khalid-majdoub@telecom-bretagne.eu

Aplication des réseaux de neurones à la reconnaissance de chiffre manuscrit

rapport technique du projet développement

Formation d'ingénieur généraliste
Version :1 24/12/2014
Encadrant : Didier Guériot
Année Scollaire : 2014/2015

TELECOM Bretagne

Résumé

Ce document présente le traitement du projet de développement intitulé : application des réseaux de neurones à la reconnaissances de chiffre manuscrit, ainsi que son déroulement et les différents résultats obtenus. Ce projet entre dans le cadre de l'apprentissage automatique. Dans ce contexte, nous allons implémenter un algorithme d'apprentissage en l'occurrence l'algorithme du gradient descendant stochastique et l'appliquer à un modèle de réseau de neurones (perceptron multicouche) affin de l'entraîner à reconnaître des chiffres manuscrits.on fera alors plusieurs simulations en variant les différents paramètres du modèle affin de bien dégager l'influence de ces derniers sur ses performances .à la fin on discutera les limitations du modèle et on ouvrera la fenêtre sur d'autres modèles.

mots clés : intelligence artificielle, apprentissage automatique, réseaux de neurones, algorithme de rétro propagation, perceptron multicouche

Abstract

In the context of their profession,...

Table des matières

Table des figures

Liste des tableaux

1 Introduction

les réseaux de neurones artificiels sont une famille d'algorithmes inspirés du neurones biologiques qui tentent d'approximer une fonction qui peut dependre de plusieurs variables et qui est en générale inconnu.

Les premiers à proposer un modèle sont deux bio-physiciens de Chicago, McCulloch et Pitts, qui inventent en 1943 le premier réseau de neurones artificiel .

entre 1943 et 1979 plusieurs autres modèles seront proposer notamment celui du perceptron par Rosenblatt en 1958 qui sera critiquer violemment après par Minsky et Papert dans leurs livre « perceptrons » pour son incapacité à résoudre des problèmes de classification non linéaires (les bases de données qui ne sont pas linéairement séparables), pendant cette période malgré les efforts de quelques chercheurs les réseaux de neurones resterons trop limités et ne susciterons pas beaucoup l'intérêt des chercheurs du domaine de l'intelligence artificielle.

il faudra attendre le début des années 80 avec les réseaux de neurones récurrents de Hopfeild et l'algorithme de rétro-propagation de Werbos pour que l'intérêt pour ce domaine soit de nouveau présent.

de nos jours les réseaux de neurones sont aux cœur de la recherche dans le domaine de l'intelligence artificielle ,ils donnent des résultats spectaculaires dans la branche du big-data et fuie de données (data mining),et sont utiliser par les géants du monde comme Google ,Microsoft et IBM pour réaliser des tâches de reconnaissance , classification ou de fuie de données.

2 modèle du perceptron multicouches

2.1 percetron

2.1.1 neurone biologique

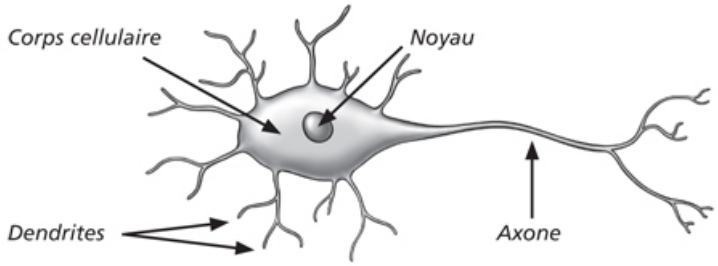
le neurone est une cellule biologique ,c'est grâce à un réseau comptent a peu près 100milliards de ces cellules sous forme d'arbre (figure 1) que nous arrivons à concevoir ce monde et à réaliser des taches de complexité très élevée.

un neurone est composé principalement de son corps cellulaire ,son Noyau ,son axone et ces dendrides.

les dendrides du neurone sont trop nombreuses et forment des liaisons avec les axones de ces voisins dans des nœuds qu'on appelle synapses à travers les quelles les différents neurones du réseau communiquent des signaux sous forme chimique (neurotransmetteurs) ,ces signaux sont alors collectés par le neurone et réagira en réponse ,on parle d'excitation.

un réseau de neurones peut donc être vu comme un graphe pondéré si on prend en compte que le degré de liaison entre le neurone et ses voisins n'est pas le même.

Figure 1 – schéma illustrant un neurone biologique

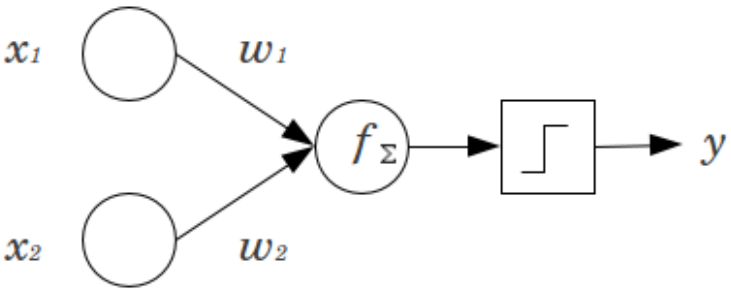


2.1.2 modèle du perceptron

le modèle du perceptron qu'on présentera ici explore une unité d'apprentissage similaire à un neurone si l'on considère que ces dendrines forment ses entrées et son axone sa sortie.donc un perceptron est une unité qui à plusieurs entrées pondérées et une sortie.

un classifieur du type perceptron ou perceptron tout court peut être vu comme une unité qui réalise une fonction .les entrées sont pondéré par des poids , sommé et après comparé à un seuil ,si le résultat est supérieur renvoie 1 si non renvoi 0.

Figure 2 – schéma d'un perceptron à deux entrées



plus formellement soient :

$w = (w_1, w_2, .., w_n)$ le vecteur regroupant les poids associés aux différentes entrées.

$x = (x_1, x_2, ..., x_n)$ le vecteur regroupant les entrées du perceptron.

b un réelle .

le perceptron peut être représenter par la fonction $\widetilde{f}_{w,b}$ qu'on appellera 'hypothèse' définit par :

$$\tilde{f}_{w,b}(x) = \begin{cases} 1 & si \quad F_\Sigma(x) > b \\ 0 & sinon \end{cases}$$

avec :

$F_\Sigma(x) = \sum_{i=1}^{n} x_i w_i =< X, W >$

2.1.3 apprentissage supervisé du modèle

le perceptron est utilisé pour la classification binaire puisque sa sortie n'admet que deux valeurs binaire.

étant donnés une base de données D de taille N,partitionnée en de deux classes C_1 et C_2 ,on veut que notre perceptron puisse prédire la classe correspondante a un élément quelconque de la base.

si f est la fonction qui a $x \in D$ associe 1 si $x \in C_1$ et 0 si non qu'on appellera cible 0, on veut que l'hypothèse $\tilde{f}_{w,b}$ du perceptron approxime f.

le problème d'apprentissage est donc un problème d'optimisation, et comme pour tout problème d'optimisation on a besoin d'une fonction de coût E qui mesurera la performance de l'approximation .

l'apprentissage revient a minimiser E sur l'ensemble des hypothèses possibles qui est dans notre cas paramétré par les poids w du perceptron.

la fonction de coût E qu'on adoptera pour cette section est l'erreur quadratique moyenne :

$$E(w) = \frac{1}{N} \sum_{x \in D} \parallel \widetilde{f}_{w,b}(x) - f(x) \parallel^2$$

le choix de l'erreur quadratique est souvent fait parce que la minimiser revient à minimiser la distance moyenne entre les éléments de sorties de l'hypothèse et ceux de la cible et donc approximer la cible par l'hypothèse puisque ces les paramètres de hypothèse qui varient.

pour minimiser cette erreur il faudra chercher les points ou son gradient s'annule et qui présentes des minima globaux,on a pour cela dans nos main la méthode du gradient .

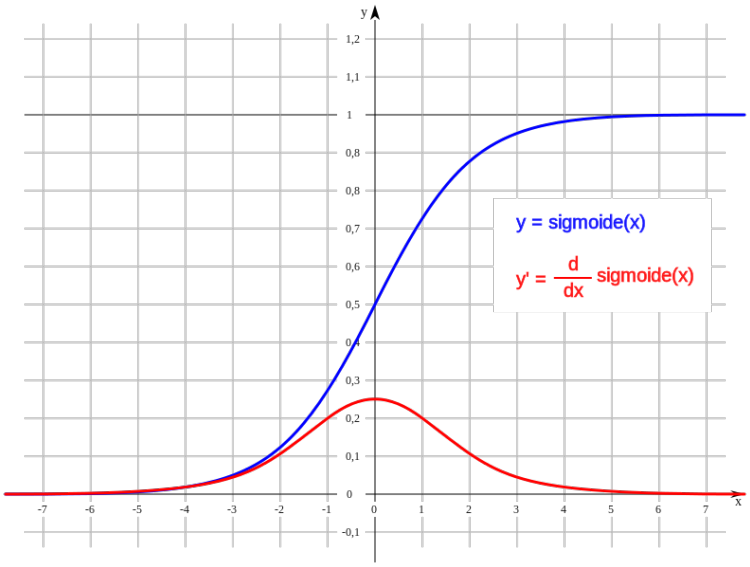
puisque la méthode du gradient requière une fonction dérivable on remplace souvent la fonction seuil du perceptron par une fonction plus lisse , dans cette section on adoptera la fonction sigmoïde σ.

l'intérêt de la fonction sigmoïde est qu'elle est indéfiniment continuement dérivable donc très lisse ,de plus elle se comporte linéairement aux voisinage de 0 .

la définition de la fonction sigmoïde et sa dérivé

$$\sigma(z) = \frac{1}{1 + e^{-z}} \text{ , } \sigma'(z) = \sigma(z).(1 - \sigma(z))$$

Figure 3 – le tracé de la fonction sigmoïde et de sa dérivé



l'hypothèse s'écrit :

$$\widetilde{f}_{w,b}(x) = \sigma(< w, x > +b)$$

calcule du gradient :

$$\frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{x\in D}(\widetilde{f}_{w,b}(x) - f(x)).\frac{\partial \widetilde{f}_{w,b}(x)}{\partial wi}$$

$$= \frac{1}{N}\sum_{x\in D}(\widetilde{f}_{w,b}(x) - f(x)).\widetilde{f}_{w,b}(x).(1 - \widetilde{f}_{w,b}(x)).x_i$$

$$\frac{\partial E}{\partial b} = \frac{1}{N}\sum_{x\in D}(\widetilde{f}_{w,b}(x) - f(x)).\widetilde{f}_{w,b}(x).(1 - \widetilde{f}_{w,b}(x))$$

dans un apprentissage supervisé on essaie de minimiser la fonction cible juste sur un sous échantillon de la base qu'on appelle base d'apprentissage ,par argument statistique(inégalité de Hoeffding) on peut montrer que si l'hypothèse approxime la fonction cible sur un sous échantillon de la base (en anglais on dit 'in sample') alors elle la rapproche aussi sur la base (en anglais out of sample).

on entraîne alors le perceptron sur cette une base d'apprentissage et puis généralement on teste ces performance sur une autre base d'échantillons étiquetés qu'on appelle base de teste.

soient $(x_1, y_1), (x_1, y_2), ..., (x_n, y_n)$ les éléments de la base d'apprentissage étiqueté et $\hat{y}_1, \hat{y}_2, ...\hat{y}_n$ leurs sorties respectives.

l'apprentissage supervisé de ce modèle consiste à minimiser :

$$E(w, y, \hat{y}), = \frac{1}{n}\sum_{i=1}^{n} \mid \hat{y}_i - y_i \mid^2$$

2.1.4 algorithme du gradient descendant stochastique

pour minimiser la fonction E donnée dans la section dessus on aura recours à des méthodes heuristiques , l'algorithme du gradient descendent en est un, il faut donc notée que l'on est confronté au problème des minima locaux .

dans l'algorithme du gradient descendant ,pour trouver un minima on navigue sur la surface de l'erreur le vecteur w en le déplaçant sur plusieurs itérations à chaque fois vers la direction qui minimise le gradient en l'occurrence $-\nabla_w E$.

au début de l'algorithme on initialise w_0 le vecteur de poids initiale avec des valeurs aléatoires et le vecteur b_0 à 0.

après à chaque itération on le déplace à un facteur près η vers la direction du gradient descendant.

la règle d'actualisation dans l'algorithme du gradient descendant est la suivante :

$$[w_{t+1}]_k = [w_t]_k - \eta.\frac{\partial E}{\partial w_k}$$

$$[w_{t+1}]_k = [w_t]_k - \eta.\frac{1}{n}.\sum_{i=1}^{n}(\hat{y}_i - y_i).\hat{y}_i.(1 - \hat{y}_i).[x_i]_k$$

$$b_{t+1} = b_t - \eta.\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i).\hat{y}_i.(1-\hat{y}_i)$$

pour l'algorithme du gradient descendant stochastique qui n'est qu'une variante de celui dessus on adopte la règle d'actualisation suivante :

pour chaque couple (x, y) de la base d'apprentissage :

$$[w_{t+1}]_k = [w_t]_k - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y}).[x]_k$$

$$b_{t+1} = b_t - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y})$$

qui s'écrit matriciellement :

$$w_{t+1} = w_t - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y}).x$$

$$b_{t+1} = b_t - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y})$$

Algorithme 1 algorithme du gradient descendant stochastique

initialise w par des valeurs aléatoires
initialise b par 0
pour i dans $1...N_{iter}$:
pour chaque couple (x, y) de la base d'apprentissage :
calculer sa sortie respective $\hat{y}$
$w = w - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y}).x$
$b = b - \eta.(\hat{y} - y).\hat{y}.(1-\hat{y})$

l'intérêt de cette variante de l'algorithme est d'éviter de tomber dans des minimas locaux triviaux facilement .

en effet ,cette fois ci au lieu de se diriger tout droit vers la direction qui minimise le plus rapidement E (qui peut être la direction qui mène vers un minima local trivial) on se dirige vers celle qui minimise juste la contribution d'un exemple d'apprentissage à chaque fois pour chaque exemple de la base.

puisque E est la moyenne de ces contributions alors statistiquement ,sur une base qui contient un grand nombre d'exemples , on se dirigera vers la direction qui la minimise.mais d'une façon stochastique ce qui fait que w va osciller beaucoup et aura moins de chance d'être piégé dans une cavité de faible profondeur.

pour s'en convaincre on peut visualiser les déplacements w pendant la recherche de la solution en navigant sur la surface d'erreur comme les déplacements d'une particule ,plus elle oscille plus elle a d'énergie et moins elle a de chance d'être piégé dans un puits de potentiel de cavité peu profonde.

2.1.5 implémentation de l'algorithme et présentation des résultats :

note : l'algorithme a été implémenté dans le langage python et se trouve dans le fichier 'test(1)_perceptron_simple.py' dans le dossier du projet .

les tests ci dessous ont étaient effectués sur une base d'apprentissage constituée d'échantillons de deux nuages gaussiens distingués dans le plan (figure 4) ,le nombre d'échantillons pris pour l'apprentissage est 100 et pour le test 50.

pour le pas d'apprentissage ,comme le montre la figure 5,autour de la valeur $\eta = 0.5$ l'algorithme ,dans ce cas, converge vers le minima le plus bas et plus rapidement ,généralement dans l'apprentissage du modèle du perceptron on choisir un pas d'erreur entre 0.1 et 0.5.

plus le pas d'apprentissage est grand plus l'erreur oscille et plus il est petit plus elle est lisse , néanmoins le pas d'apprentissage doit rester relativement petit pour que le terme $-\eta.\nabla E$ ne soit pas dominant il faut qu'il reste juste une petite perturbation qui nous mène vers un minima acceptable.

Figure 4 – représentation spatiale de la base d'apprentissage (la couleur désigne la classe)

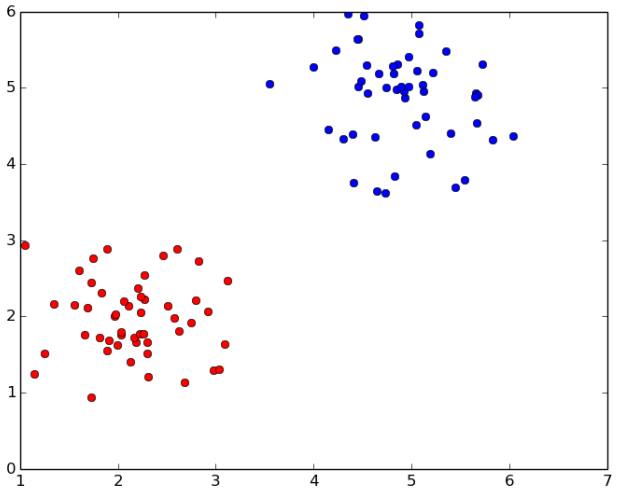
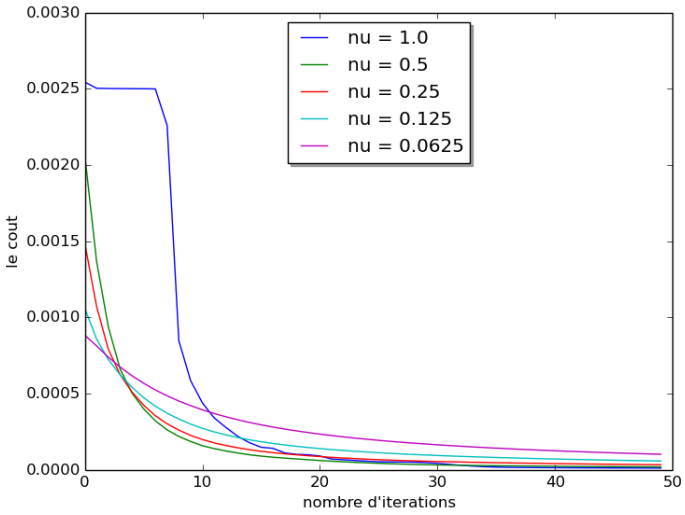


Figure 5 – cette figure représente l'évolution de cout pour différents pas d'aprentissage en moyenne sur 100 executions



le taux d'erreur obtenu est de 0%,le perceptron arrive a parfaitement classifier des nuages de points linéairement séparables.

quand on l'entraîne sur deux nuages gaussiens qui se recouvrent le perceptron a de terrible performances .c'est la plus grande limitation de ce modèle .

cette limitation vient du fait que la solution de l'équation $< w, x > +b = 0$,qui définit la limite qui est supposé séparer les deux classes ,est un hyperplan et que pour qu'un hyper plan puisse séparer parfaitement deux régions d'un espace il faut que ces dernières soient linéairement séparables.

on peut deviner que la combinaisons de plusieurs perceptrons aura un plan de séparation plus tordu et pourra remédier a ce problème et c'est ce qu'on va voir dans la section suivante.

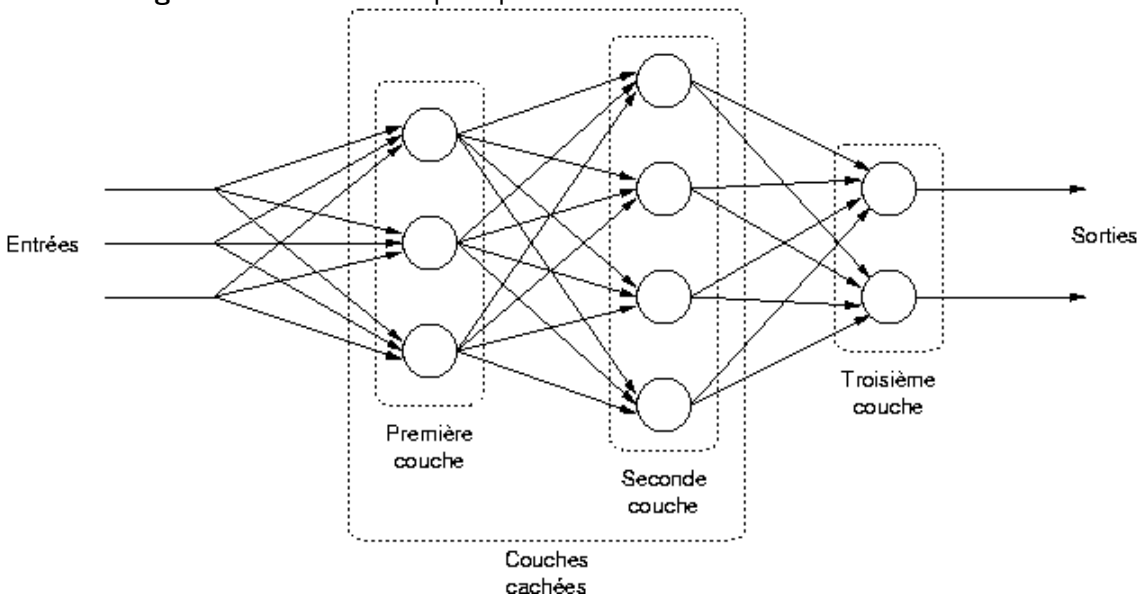
2.2 perceptron multicouche

le perceptron multicouche est une combinaison de couches de perceptrons entièrement connectées ,c'est un système qui comme le perceptron a des entrées et des sorties , les sorties de chaque couche sont renvoyé vers la suivante ainsi de suite jusqu'à la dernière couche ,les sorties de cette dernière constituent la réponse du système et sont en nombre de classes à distinguées.

les couches qui se trouvent entre l'entré et la sortie sont appelées couches cachées.

la sortie de chaque perceptron de la dernière couche peut être vue comme la probabilité pour que l'entré appartient à une des classes.

Figure 6 – schème d'un perceptron multicouche à deux couches cachées



2.2.1 traitement mathématique :

comme pour le perceptron l'apprentissage du perceptron multicouches consiste à trouver les poids de chaque perceptron de chaque couches qui minimisent une fonction de coût.

les notations suivantes seront fixées pour cette section :

— soit f est une fonction de $\mathbb{R}$ vers $\mathbb{R}$ est $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ on notera $f(x) = (f(x_1), f(x_2), ..., f(x_n))$.

— soit $f\ : u \in \mathbb{R}^n \to f(u) \in \mathbb{R}^m$, on notera $\dfrac{\partial f(u)}{\partial u} = \left[\dfrac{\partial f(u)_i}{\partial v_j}\right]_{i,j} \quad i : 1..n, j : 1..m$.

— on notera $u \bullet v = (u_1 v_1, u_2 v_2, ..., u_n v_n)$ pour deux vecteurs de même taille n.

— puisque les couches sont entièrement connectées leurs perceptrons respectifs ont le même nombre de poids ,ces derniers peuvent donc être rangé dans une matrice $w^{(l)}$ l'indice l indique le numéro de la couche.

— on notera $z^{(l)}$ le vecteur qui regroupe les sorties de la couche l avant le passage par la fonction seuil (on dira avant activation) et $a^{(l)}$ le vecteur regroupant les sorties de la couche après activation.

— la sortie du système sera notée $a^{(max)}$et l'entré $a^{(0)}$.

— notant $w_1^{(l)}, w_2^{(l)}, ..., w_{n_l}^{(l)}$ les vecteurs poids des perceptrons de la couche l (les lignes de $w^{(l)}$) et $b_1^{(l)}, b_2^{(l)}, ..., b_{n_l}^{(l)}$ leurs seuils respectives.

— f désignera la fonction d'activation (par exemple la fonction sigmoïde).

on a $a_i^{(l)} = f(z_i^{(l)})$ pour tout i dans $1...n_l$ qu'on écrira pour plus de simplification :

$$a^{(l)} = \sigma(z^{(l)})$$

les entrés de la couche l sont les sorties de la couche $l-1$ après activation donc :
$z_i^{(l)} =< w_i^{(l)}, a_i^{(l-1)} > +b_i^{(l)}$ pour tout i dans $1...n_l$.
qui s'écrit matriciellement :

$$z^{(l)} = w^{(l)}.a^{(l-1)} + b^{(l)}$$

soit une entrée x, étiqueté par y qui est dans le cas de l'apprentissage du perceptron multicouche,un vecteur de taille le nombre de classes et qui contient 1 dans l'indice correspondant à la classe qu'il désigne et 0 dans le reste.

la contribution de ce vecteur dans la fonction de coût est donnée par :

$$E_y = \frac{1}{2} \parallel y - a^{(max)} \parallel^2$$

calcule du gradient suivant cette composante :

E_y dépend des poids de la dernière couche mais aussi ceux de toutes les couches précédentes donc on aura affaire a une règles de chaîne lors du calcule des dérivées partielles.

— pour plus de simplifications dans les calcules on notera : $\delta^{a(l)} = \dfrac{\partial E_y}{\partial a^{(l)}}$ et $\delta^{z(l)} = \dfrac{\partial E_y}{\partial z^{(l)}}$.

on commence par le calcule du gradient de la dernière couche.

$$\delta^{a(max)} = \frac{\partial E_y}{\partial a^{(max)}} = (a^{(max)} - y)$$

$$\frac{\partial a^{(max)}}{\partial z^{(max)}} = \frac{\partial \sigma(z^{(max)})}{\partial z^{(max)}} = f'(a^{(max)})$$

$$\delta^{z(max)} = \frac{\partial E_y}{a^{(max)}} \circ \frac{\partial a^{(max)}}{\partial z^{(max)}} = \delta^{a(max)} \bullet f'(a^{(max)})$$

$$\frac{\partial z_i^{(max)}}{\partial w^{(max)}} = \frac{\partial (w_i^{(max)} a^{(max-1)} + b_{(max)})}{\partial w^{(max)}} = i \begin{pmatrix} 0 \\ . \\ . \\ a^{(max-1)} \\ . \\ . \\ 0 \end{pmatrix}$$

$$\frac{\partial E_y}{\partial w^{(max)}} = \frac{\partial E_y}{\partial z^{(max)}} \circ \frac{\partial z^{(max)}}{\partial w^{(max)}} = \sum_{i=1}^{n_{max}} \frac{\partial E_y}{\partial z_i^{(max)}} \frac{\partial z_i^{(max)}}{\partial w^{(max)}}$$

$$= \sum_{i=1}^{n} \delta_i^{z(max)} . \begin{pmatrix} 0 \\ . \\ . \\ a^{(max-1)} \\ . \\ . \\ 0 \end{pmatrix} i = \begin{pmatrix} \delta_1^{z(max)} a^{(max-1)} \\ . \\ . \\ . \\ \delta_n^{z(max)} a^{(max-1)} \end{pmatrix} =^t \delta^{z(max)} a^{(max-1)}$$

pour les seuils :

$$\frac{\partial E_y}{\partial b^{(max)}} = \sum_{i=1}^{n} \delta_i^{z(max)}$$

finalement pour la dernière couche on obtient les formules matricielles suivantes :

$$\frac{\partial E_y}{\partial w^{(max)}} =^t \delta^{z(max)} a^{(max-1)}$$

$$\frac{\partial E_y}{\partial b^{(max)}} = \sum_{i=1}^{n_{max}} \delta_i^{z(max)}$$

$$\delta^{z(max)} = (a^{(max)} - y) \bullet f'(a^{(max)})$$

le calcule du gradient pour les couches cachées se fera par récurrence descendante en exploitant la règle de chaîne des dérivées partielles.

$\frac{\partial E_y}{\partial w^{(l)}} =^t \delta^{z(l)} a^{(l-1)}$

$\frac{\partial z_i^{(l+1)}}{\partial z^{(l)}} = \frac{\partial z_i^{(l+1)}}{\partial a^{(l)}} \circ \frac{\partial a^{(l)}}{\partial z^{(l)}} = \sum_{j=1}^{n_l} \frac{\partial z_i^{(l+1)}}{\partial a_j^{(l)}} . \frac{\partial a_j^{(l)}}{\partial z^{(l)}}$

$$= \sum_{j=1}^{n_l} w_{ij}^{(l+1)} . \begin{pmatrix} 0 \\ . \\ . \\ f'(a_j^{(l)}) \\ . \\ 0 \end{pmatrix} j = \begin{pmatrix} f'(a_1^{(l)}).w_{i1}^{(l+1)} \\ . \\ . \\ . \\ f'(a_{n_l}^{(l)}).w_{in_l}^{(l+1)} \end{pmatrix}$$

$\frac{\partial z_i^{(l+1)}}{\partial z^{(l)}} = f'(a^{(l)}) \bullet^t w_i^{(l+1)}$

$\frac{\partial E_y}{\partial z^{(l)}} = \frac{\partial E_y}{\partial z^{(l+1)}} \circ \frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \sum_{i=1}^{n_{l+1}} \frac{\partial E_y}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial z^{(l)}} = \sum_{i=1}^{n_{l+1}} \delta_i^{z(l+1)} f'(a^{(l)}) \bullet^t w_i^{(l+1)}$

$= \frac{\partial E_y}{\partial z^{(l)}} = f'(a^{(l)}) \bullet ({}^t\delta^{(l+1)} . {}^t w^{(l+1)})$

pour les seuils :

$$\frac{\partial E_y}{\partial b^{(l)}} = \sum_{i=1}^{n_l} \delta_i^{z(l)}$$

on obtient finalement la relation de récurrences suivante :

$$\delta^{(l)} = \frac{\partial E_y}{\partial z^{(l)}} = f'(a^{(l)}) \bullet ({}^t\delta^{z(l+1)} . {}^t w^{(l+1)})$$

$$\frac{\partial E_y}{\partial w^{(l)}} =^t \delta^{z(l)} a^{(l-1)}$$

$$\frac{\partial E_y}{\partial b^{(l)}} = \sum_{i=1}^{n_l} \delta_i^{z(l)}$$

2.2.2 algorithme de rétro-propagation :

l'algorithme de rétro-propagation est une version de l'algorithme du gradient descendant stochastique adapté au perceptron multicouches.

il est appelé algorithme de rétro-propagation parce que dans l'algorithme on commence par propager l'entrée à travers les différentes couches et on rebondi en arrière pour propager l'erreur i.e calcule du vecteur δ en actualisant les poids des couches suivant la même règle que celle du gradient descendant.

Algorithme 2 algorithme de rétro-propagation

initialise la matrice de poids $w^{(l)}$ de chaque couche par des valeurs aléatoires
initialise le vecteur de seuils $b^{(l)}$ de chaque couche par 0
pour n dans $1...N_{iter}$:
pour tout couple $(x = a^{(0)}, y)$ de la base d'apprentissage :
calcule les sorties $a^{(1)}, a^{(2)}, ..., a^{(max)}$ des différents couches
$\delta^{z(max)} = f'(a^{(max)}) \bullet (a^{(max)} - y)$
$w^{(max)} = w^{(max)} - \eta.{}^t\delta^{z(max)}.z^{(max-1)}$
$b^{(max)} = b^{(max)} - \eta.\sum_{i=1}^{n_{max}} \delta_i^{z(max)}$
pour l dans $(max-1)...1$:
$\delta^{z(l)} = f'(a^{(l)}) \bullet ({}^t\delta^{z(l+1)}.{}^tw^{(l+1)})$
$w^{(l)} = w^{(l)} - \eta.{}^t\delta^{z(l)}.a^{(l-1)}$
$b^{(l)} = b^{(l)} - \eta.\sum_{i=1}^{n_l} \delta_i^{z(l)}$

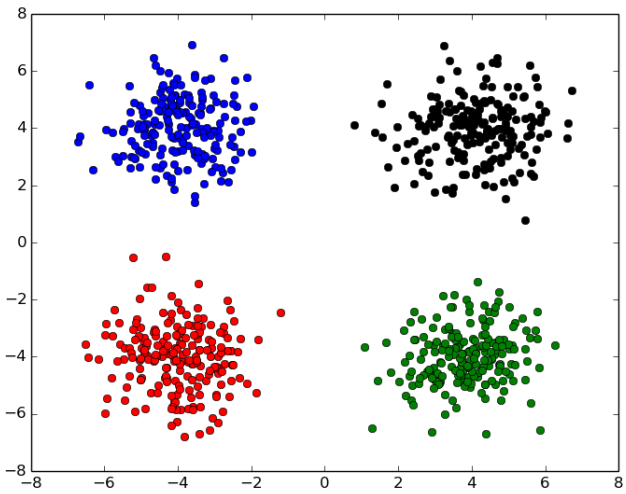
2.2.3 implémentation de l'algorithme et présentation des résultats(cas simple) :

l'implémentation de l'algorithme à été faites sur python .le fichier 'test(2)_perceptron_multicouche.py' dans le dossier du projet contient le code de cette simulation.

on va tester l'apprentissage avec cet algorithme sur la classification de 4 nuages gaussiens (figure 7).

la fonction d'activation choisie pour les tests et la fonction sigmoïde .

Figure 7 – base d'apprentissage 800 échantillons (la couleur indique la classe)



le perceptron multicouches qu'on va soumettre aux teste suivant est très simple , il est composé juste d'une seule couche contenant 4 perceptrons chacun ayant 2 entrés.

la figure du coût par nombre d'itération est similaire à celle obtenue pour 1 perceptron .

Figure 8 – représentation du cout par itération pour $\eta = 0.1$

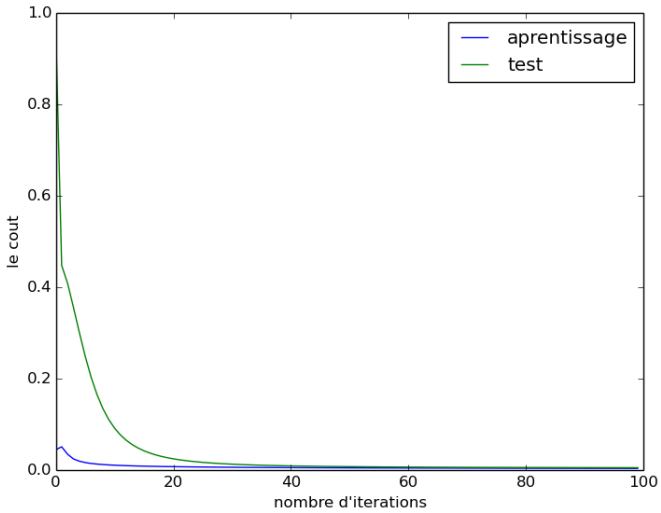
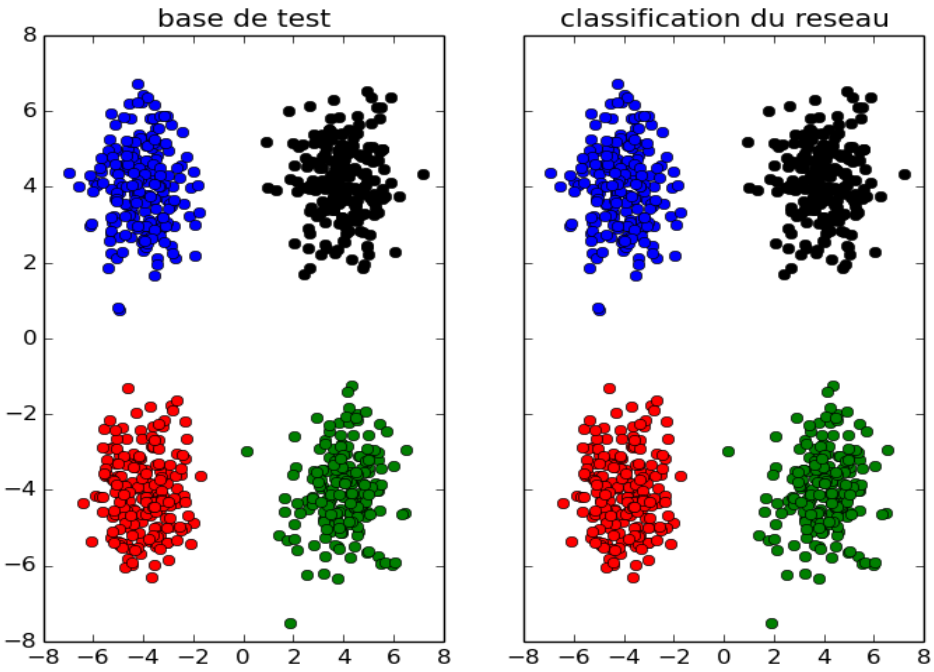


Figure 9 – base de test et réponse du réseau (800 échantillons)



encore une fois ce modèle distingues les 4 classes (qui cette fois ne sont pas linéairement séparables) avec un taux d'erreur de 0% sur la base de test.

pour 4 nuages gaussiens non distingués :

Figure 10 – base d'apprentissage

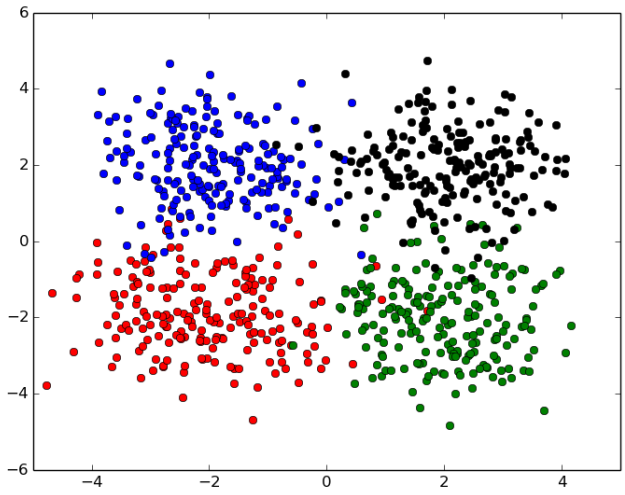
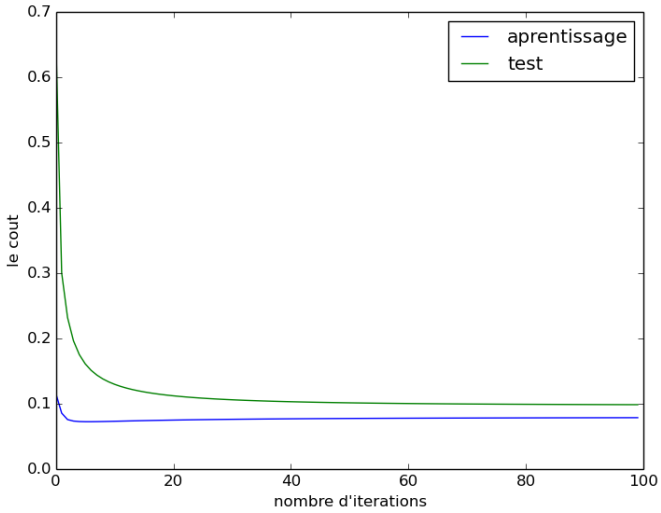
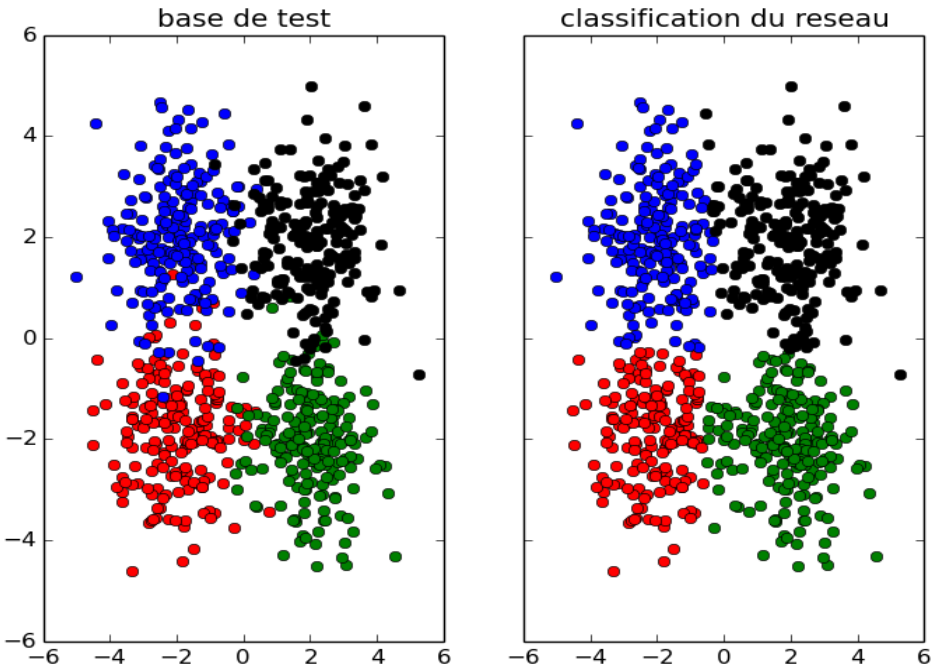


Figure 11 – coût



le taux d'erreur obtenu est cette fois 6% .comme le montre la figure 12 le modèle fait des erreurs face à la distinction de classes non distingués ce qui est prévisible puisque pour les points ou le modèle a fait des erreurs il n'est pas claire à quel classe ils appartiennent . cette limitation revient toujours dans les problèmes de classification ou il y a une sorte de corrélation entre les différents classes et ou les données ne sont pas suffisantes pour y remédier comme dans notre cas .

Figure 12 – base de test et réponses du réseau



il faut noter que les paramètres des modèles basées sur le perceptrons multicouches sont souvent choisie de façon empirique ,il n'y a pas de formule mathématique nous indiquant qu'elle doit être par exemple le nombre d'éléments dans la couche caché ou qu'elle pas d'apprentissage choisir .

ceci étant, le modèle du perceptron multicouche donc dépasse de loin celui du perceptron simple ,il peut effectuer des tâches de classification beaucoup plus complexes et peut reconnaître un tas de fonctions y compris toutes les fonctions logiques .

Ça ce vois bien donc pourquoi ce modèle a était une avancée majeure dans le domaine des réseaux de neurones.

3 application à la reconnaissance de chiffre manuscrit :

dans cette section nous allons appliquer les résultats précédents sur une situation plus réaliste en l'occurrence la reconnaissance de chiffre manuscrit , pour cela nous avons développer une interface permettant de dessiner un chiffre à la main(figure 13) , on a alors former une base d'apprentissage d'échantillons de 847 chiffres manuscrits (figure 14) de tailles 28x28 qu'on va soumettre à un réseau de neurones artificiel du type perceptron multicouches,ainsi qu'une base de test pour évaluer ces performances.

Figure 13 – interface

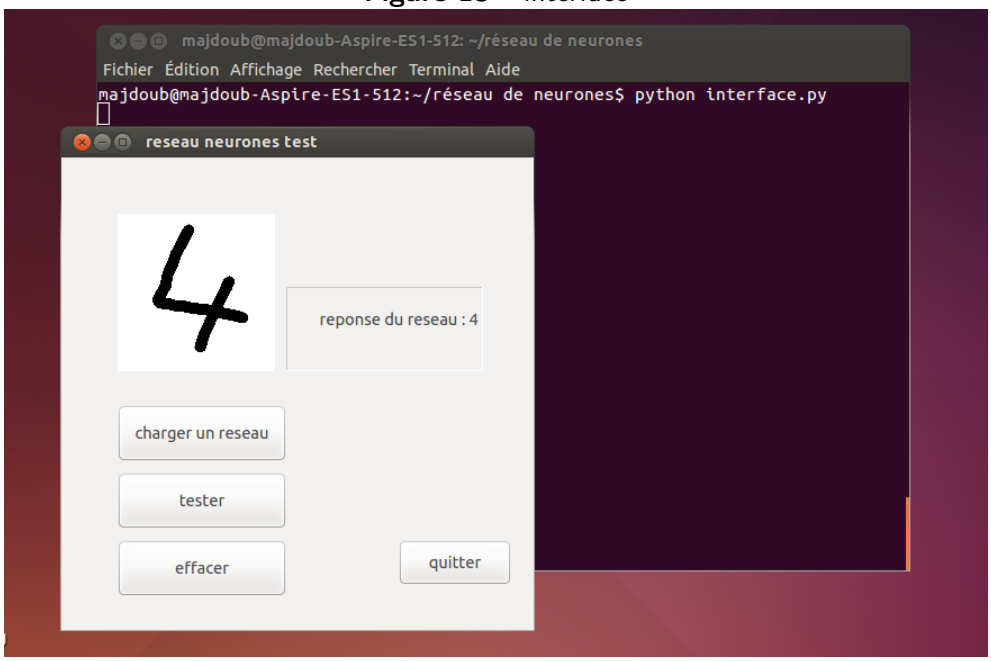
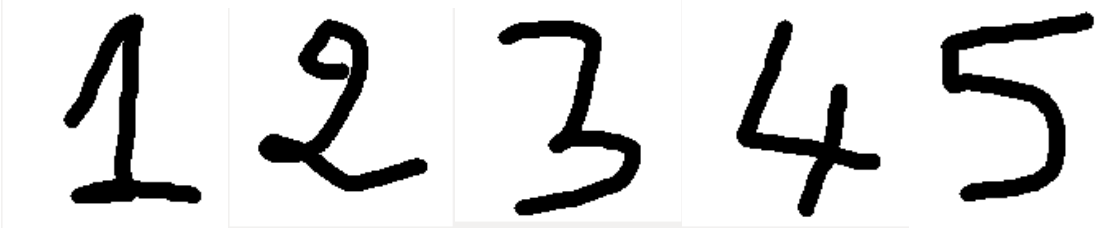


Figure 14 – exemples d'éléments de la base générée

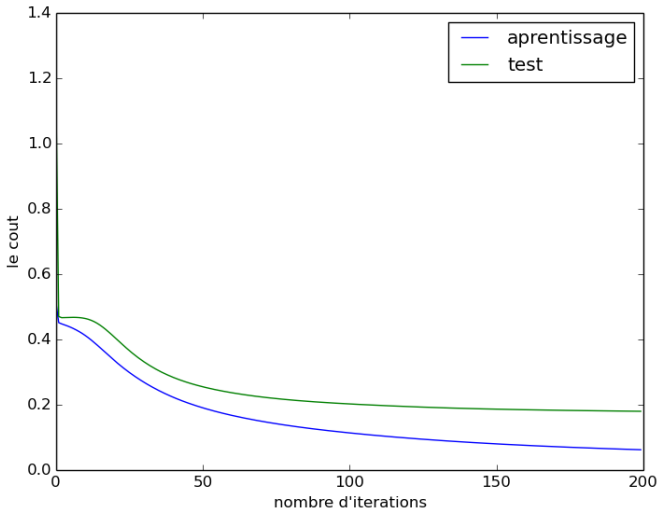


le modèle qu'on va entraîner est un perceptron à deux couches ,une couche cachée contenant 120 perceptrons et la couche de sortie contenant 10 perceptrons pour les dix classes (chiffres de 0 à 9).

l'algorithme utilisé est celui de la rétro-propagation avec un petit changement , au lieu de à chaque itération actualiser les poids pour chaque éléments de la base on découpe la base en de petit paquets et on actualise les poids pour chaque paquets .

cela permet a l'algorithme de s'exécuter plus rapidement.

Figure 15 – coût par itération obtenu pour $\eta = 0.1$



le taux d'erreur obtenu est de 22% ce qui est acceptable vu la taille assez limité de la base d'apprentissage , un apprentissage de 40 000 chiffres de la base de MNIST pour un réseau de 300 unités dans la couche caché a pu nous donner un taux d'erreur de 2% sur une base de test de 10 000 exemples.

on a mis à disposition du lecteur une sauvegarde du réseau entraîné (réseau.txt) que vous pouvez charger sur l'interface et tester manuellement en dessinant des chiffres , le ficher interface.py permet le lancement de l'interface .

on a pu se convaincre à travers ces différents test de la possibilité de résoudre des problèmes concret avec un tel modèle .

le modèle du perceptrons multicouches a lui aussi des limitations on en cite :

— la complexité du modèle augmente terriblement avec la dimension de la base de données ,pour la reconnaissance d'images de taille 1028x480 il faudra un système de 491 520 ! unités juste dans la première couche.

— le modèle n'est pas robuste ,il ne cerne pas le problème de translation et de déformation de données.

pour remédier à ces limitations en générale on précède le perceptrons multicouche par des couches de prétraitement de données pour réduire la taille des données et pour la transformer de façon a ce que l'apprentissage soit plus résistant aux transformations.

4 ouverture : réseaux de neurones à convolution :

un réseau de neurones à convolution (convolutinal neural network en anglais) est une extension du perceptron multicouche ou on ajoute au début deux couches ,une charger de filtrage des images (convolutinal layer en anglais) de la base et l'autre fait un sous échantillonnage (pooling layer en anglais).

au niveau de la couche de filtrage on convolue l'image avec des filtres entraînables générant ainsi un groupe d'images filtrées (feature maps en anglais) qui vont passées après par une

fonction d'activation, sous échantillonnées et finalement renvoyées au classifieur (perceptron multicouche) .

Figure 16 – schéma d'un réseau de neurones à convolution

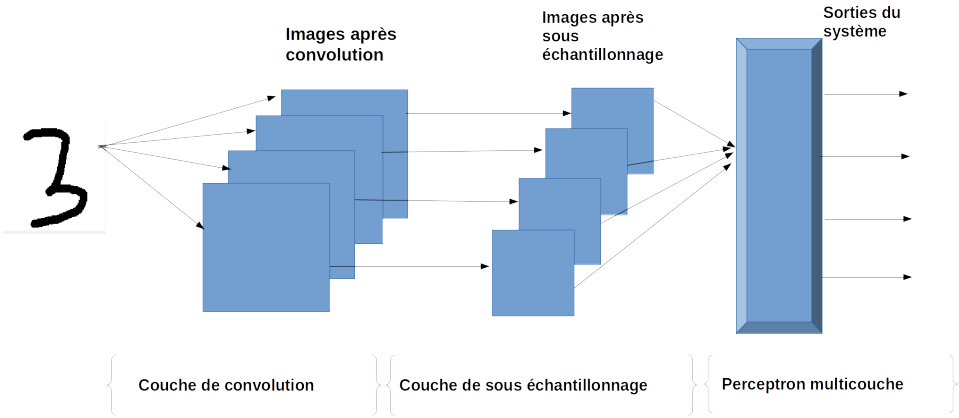
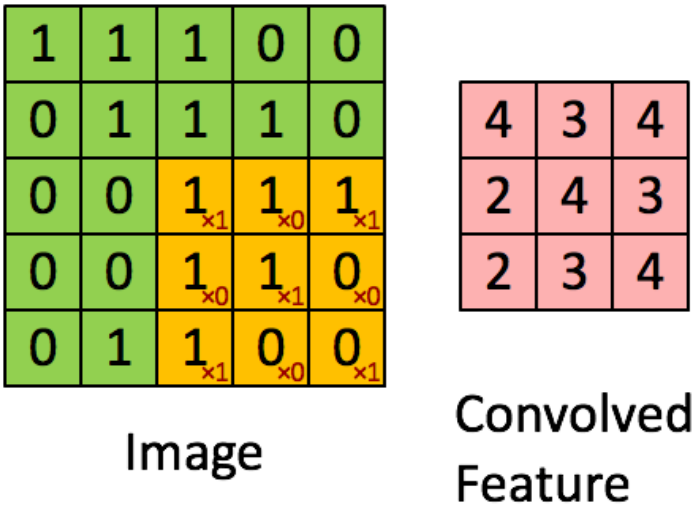


Figure 17 – exemple de convolution



la dimension des images extraite par convolution est égale à la dimension de l'image originale moins la dimension du filtre plus un, en générale pour le sous échantillonnage en prend la moyenne ou le maximum de chaque 4 pixels voisin de l'image cela permet de réduire encore la dimension des images de moitié .

l'apprentissage des poids des filtres se fait par rétro-propagation, on ne va pas présenter ici les calcules néanmoins on a mis à disposition du lecteur dans le dossier du projet une classe CNN pour convolutional neural network qui gère ce genre de réseaux.

faute de manque de matériel puissant nous n'avons pas pu faire des simulations permettant d'avoir des résultats concret (les opérations de convolution sont très gourmandes).

5 conclusion

l'étude des réseaux de neurones était une occasion pour explorer de plus près le domaine de l'apprentissage automatique qui est à la fois très fascinant et prometteur.

à la fin de ce projet on ne peut qu'exprimer une très grande admiration face à un algorithme tel l'algorithme de rétro-propagation qui est simple et ne fait intervenir que des méthodes d'optimisation élémentaires mais qui donne des résultats assez surprenant.

on a pu se convaincre aussi de l'importance de coopération entre différents domaines de la science comme dans ce cas les mathématiques , l'informatique et la biologie qui peut parfois donner naissance a des solutions élégantes face à des problèmes qui paressent très complexes du premier abord.

il faut aussi souligné que la nature est encore une source d'inspiration très riches de solutions innovantes, elle recèle encore des secrets qui une fois percés nous permettrons de venir à termes de plusieurs problèmes d'enjeux majeurs dans ce siècle comme la compréhension du fonctionnement cerveau qui non seulement une fois éclairé donnera naissance à des générations de machines intelligentes qui dépassent de loin nos calculateurs actuels mais peut être mènera à un changement radicale de l'humanité comme le prédisent les transhumanistes.

[3, 4, 5, 1, 2, 6]

Références

[1] Numpy ,scipy documentation disponible sur http ://docs.scipy.org/doc/.

[2] Pyside documentation disponible dur http ://qt-project.org/wiki/pysidedocumentation.

[3] Martin-Luther-Univ. Halle-Wittenberg Germany ; Pfister M. ; Rojas R. Behnke, S. ; Inst. of Comput. Sci. Recognition of handwritten digits using structural information. volume vol.3, pages 1391 – 1396. IEEE, jun 1997.

[4] Jake Bouvrie. Notes on convolutional neural networks. November 2006.

[5] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[6] Central Queensland Univ. Rockhampton-Qld. Australia Lu J. ; Ghosh M. ; Ghosh R. Verma, B. Fac. of Inf. & Commun. A feature extraction technique for online handwriting recognition. volume vol.2, pages 1337 – 1341. IEEE, july 2004.

www.telecom-bretagne.eu

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
Tél : + 33 (0)2 29 00 11 11
Fax : + 33 (0)2 29 00 10 00

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
Tél : + 33 (0)2 99 12 70 00
Fax : + 33 (0)2 99 12 70 19

Campus de Toulouse
10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
Tél : + 33 (0)5 61 33 83 65
Fax : + 33 (0)5 61 33 83 75