Rapport N 3 TP SDD

ISIMA

Imad ENNEIYMY
enneiymy.i@gmail.com

Yassir Karroum
ukarroum17@gmail.com

6 juin 2017

Table des matières

1 Introduction

Le présent document vise à décrire le projet ZZBrain, ce projet fut réalisé dans le cadre du module *Projet de deuxième semestre de la première année* à l'ISIMA. Il vise à créer une bibliothéque basique permettant la création et l'entrainement d'un classification basée sur les reseaux de neurons, c'est un projet purement pédagogique, et le lecteur intéréssé par une bibliothéque pareil trouvera des alternatives bien plus puissantes tel que : *TensorFlow* ou *DLib*.

Les notations, alorithmes et formules utilisés sont fortement basés sur le cours *Machine Learning* de *Andrew Ng* meme si ce dernièr est principalement proposé en *Octave*[1] alors que ZZBrain est codée en C++.

Nous utilisons la biblothéque *Dlib* pour la minimisation de la fonction $J(\Theta)$.

Dans ce qui suit nous présenterons de façon générale l'idée derrière les reseaux de neurons, nous présenterons également des conventions et notations qui seront utilisés un peu plus bas our expliquer les algorithmes et les structures de données utilisés. Nous proposerons également des pistes pour d'éventuels améliorations de ce projet.

Une multitude d'exemples (implémentés en utilisant ZZBrain), seront également présentés et expliqués un peu plus bas.

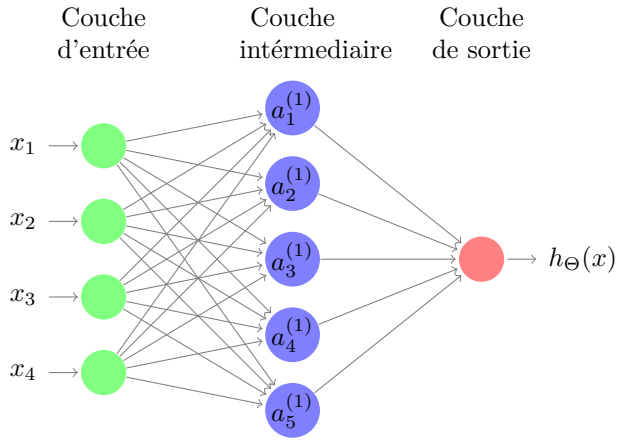
2 Reseaux de neurons

2.1 Introduction

Les reseaux des neurons representent une des methodes les plus utilisés en Machine Learning, ils sont utlisés dans de nombreux domaines comme la vision par ordinateur entre autre. Leur popularité vient du fait qu'ils permettent de modéliser des problèmes trés complexes. Le champion du monde jeu Go vient d'etre battu il y a deux semaines par un réseau de neurons, dans le jeu que les informaticiens considérent comme étant l'un des jeu les plus dur à automatiser puisque le nombre de possibilités est infinement grand. Cette methode nécéssite néamoins une puissance de calcul et une mémoire assez conséquente, dés que le nombre des noeuds commencent à croitre.

2.2 Structure du réseau

Un réseau de neurons peut etre representé comme suit :



La couche d'entrée est ce qui est fourni au programme dans le cas d'une vision par ordinateur cela peut etre les pixels de l'image, dans le cas d'un filtre à spam, celà peut etre des booléans representant l'existance de certains mots-clés. Les couches intérmédiaires permettent de complexifier le réseau et de lui permettre de modéliser des problèmes de plus en plus complexes, le problème c'est qu'elles augmentent énormement

1. Alternative libre à *Matlab*

le temps de calcul. La couche de sortie represente le résultat retourné par le réseau celà est un vecteur de booléans (souvent il est de taille 1) qui nous informe si oui ou non le vecteur d'entrée appartient à la classe donnée (par exemple si une image donnée en entrée est une voiture, ou si un message est un spam).

Le vecteur $a^{(1)} = \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \\ a_5^{(1)} \end{pmatrix}$

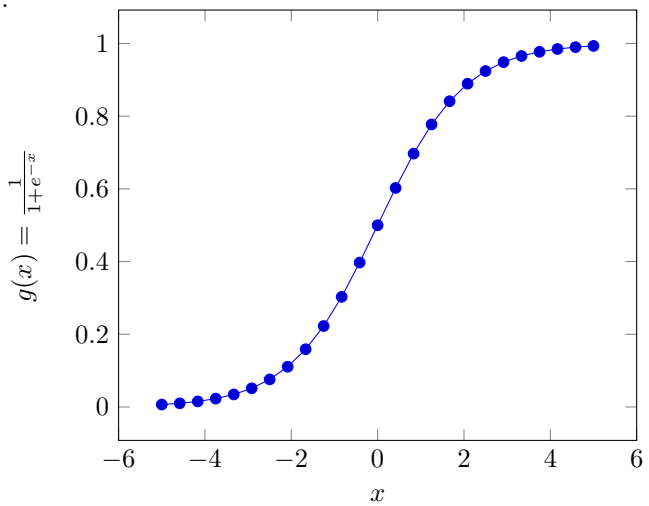
permet de calculer la sortir $h_\Theta(x)$ en effet on pourrait considérer le vecteur $a^{(1)}$ comme la nouvelle entrée est ainsi de suite. Ce procédé s'appelle la *Forward Propagation*. il sera expliqué un peu plus bas. Les poids sont stockés dans une matrice tri-dimensionnelle notée Θ, chaque poids est dénoté par : $\Theta_{ij}^{(k)}$ où k est le numéro de la couche, i le numero du noeud de la couche 2 et j le numéro du noeud de la couche 1.

On peut alors calculer $h_\Theta(x)$ comme suit :

$z_1^{(1)} = \Theta_{11}^{(0)} x_1 + \Theta_{12}^{(0)} x_2 + \Theta_{13}^{(0)} x_3 + \Theta_{14}^{(0)} x_4$
$a_1^{(1)} = g(z_1^{(1)})$
$z_2^{(1)} = \Theta_{21}^{(0)} x_1 + \Theta_{22}^{(0)} x_2 + \Theta_{23}^{(0)} x_3 + \Theta_{24}^{(0)} x_4$
$a_2^{(1)} = g(z_2^{(1)})$
$z_3^{(1)} = \Theta_{31}^{(0)} x_1 + \Theta_{32}^{(0)} x_2 + \Theta_{33}^{(0)} x_3 + \Theta_{34}^{(0)} x_4$
$a_3^{(1)} = g(z_3^{(1)})$
$z_4^{(1)} = \Theta_{41}^{(0)} x_1 + \Theta_{42}^{(0)} x_2 + \Theta_{43}^{(0)} x_3 + \Theta_{44}^{(0)} x_4$
$a_4^{(1)} = g(z_4^{(1)})$
$z_5^{(1)} = \Theta_{51}^{(0)} x_1 + \Theta_{52}^{(0)} x_2 + \Theta_{53}^{(0)} x_3 + \Theta_{54}^{(0)} x_4$
$a_5^{(1)} = g(z_5^{(1)})$
$h_\Theta(x) = g(\Theta_{11}^{(1)} a_1 + \Theta_{12}^{(1)} a_2 + \Theta_{13}^{(1)} a_3 + \Theta_{14}^{(1)} a_4)$

Ce procédé est communement appelé : *Forward Propagation*.

Ici la fonction $g(x)$ est ce que l'on désigne une fonction d'activation, dans le programme nous avons choisi la fonction de Sigmoid (ce choix pourra être changé dans le futur), il existe une multitude de fonctions, notre choix est basé sur celui fait par *Andrew*. La fonction *Sigmoid* est définie par : $g(x) = \frac{1}{1+e^{-x}}$ dont la courbe est :



Cette fonction a la particularité de converger trés vite vers 1 ou 0 ce qui s'avére trés utile puisque le résultat final doit etre un booléan.

2.2.1 Fonction du cout

La fonction du cout permet de mesurer à quel point notre réseau est précis, cette fonction dépend de theta est la minimiser revient à augmenter la précision de notre réseau. cette fonction est définie par : $J(\Theta) = -\frac{1}{m}[\sum_{i=0}^{m}\sum_{k=0}^{K} y_k^{(i)} log(h_\theta(x^{(i)})_k) + (1 - y_k^{(i)}) log(1 - h_\theta(x^{(i)})_k)] + \frac{\lambda}{2m}\sum_{l=0}^{L-1}\sum_{i=0}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_j^{(l)})^2$ Avec :

m **:** Taille du dataset donnée en entrée (pour l'entrainement du classificateur).

K **:** Taille du vecteur de sortie.

y **:** Réponse à une donnée particulière par exemple 1 si l'image en entrée est une voiture.

$h_\theta(x^{(i)})$ **:** Réponse donnée par le réseau de neurons (n'est pas nécéssairement égale à y).

λ **:** : Parametre fournie par l'utilisateur qui permet d'éviter *l'overfitting*[2].

L **:** Nombre de couches.

s_l **:** Nombre de noeuds dans la couche l.

Entrainer le réseaux des neurons revient donc à trouver les thetas qui minimiseront la fonction, celà se fera grace à des fonctions préfournies par la bibliothéque *Dlib*. Il faut tout de même noter que cette fonction n'est pas convexe, l'algorithme de minimisation ne garantit donc en rien qu'on trouvera un minima global, les minima locals peuvent donner des résultas satisfaisants si le réseaux posséde assez de couches intermediaires.

Pour minimiser cette fonction nous utilisons le BFGS, un algorithme quasi-newtonien. Cet algorithme est déja implémenté en Dlib, et prends deux parametres, une fonction et son gradient dans un point donnée. Pour calculer le gradient nous utiliserons un procédé nommé la *Back Propagation*. Il faut noter qu'il est également possible d'approcher ce gradient par des méthodes numérique, cette dernière à l'avantage d'etre plus facile et plus rapide à implémenter mais elle est beaucoup plus couteuse et ne sera donc pas utilisé, ces méthodes numériques permettent tout de même de vérifier si une implémentation de la *Back Propagation* est correcte, mais ne devra être utilisé que pour des fins de déboguage.

2.2.2 Back Propagation

La *Back Propagation* nous permettera de calculer les gradients de la fonction du cout.

Algorithme 1 : Back Propagation

Procedure *backPropagation()*
  Initialiser $\Delta_{ij}^{(l)} = 0$ pour tout l, i, j ;
  pour *i allant 1 à m* **faire**
    Calculer les différents $a^{(l)}$ par la forward propagation ;
    Calculer $\delta^{(L)} = a^{(L)} - y^{(i)}$;
    pour *l allant de L - 1 à 2* **faire**
      $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* {}^a g'(z^{(l)})$
    fin pour
    $\Delta_{kj}^{(l)} = \Delta_{kj}^{(l)} + a_j^{(l)} \delta_k^{(l+1)}$
  fin pour
  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)}$ (si $j \neq 0$)
  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)}$ (si $j = 0$)
Fin

a. Multiplication élément par élément

2. L'overfitting est le fait d'avoir un réseau qui fournit d'excellents réseultats sur le dataset d'entrainements mais a des performences médiocres sur des données qu'il n'a jamais vu