Université de Montréal

Devoir 3 - IFT6390

Auteur :
Alexandre Bonhomme

Professeur :
Dr. Pascal Vincent

23 décembre 2012 - BONA20128906

1 Calcul du gradient pour l'optimisation des paramètres d'un réseau de neurones

a) Notre couche caché contient d_h neurones on obtient donc un vecteur de poids $\mathbf{W}^{(1)}$ de taille $d_h \times d$ avec d la dimensionnalité de l'entrée ($x^{(i)} \in \mathbb{R}^d$)) et un vecteur de biais $\mathbf{b}^{(1)}$ logiquement de taille d_h, car on a un biais par neurone.
On calcul le vecteur d'activation $\mathbf{h}^a$ comme cela :

$$\mathbf{h}^a = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{1}$$

Si l'on détail l'expression pour chaque composante on obtient :

$$\begin{aligned}\mathbf{h}^a_j &= \mathbf{W}^{(1)}_j\mathbf{x} + \mathbf{b}^{(1)}_j \\ &= \sum_{i=1}^{d} \mathbf{W}^{(1)}_{j,i}\mathbf{x}_i + \mathbf{b}^{(1)}_j\end{aligned}$$

Enfin, pour déterminer $\mathbf{h}^s$ on applique une transformation non linéaire :

$$\mathbf{h}^s = \tanh(\mathbf{h}^a) \tag{2}$$

b) On a m cible et d_h neurones dans la couche caché, donc la dimension de $\mathbf{W}^{(2)}$ est égale à $m \times d_h$ et un vecteur de biais $\mathbf{b}^{(2)}$ de taille m. On calcul le vecteur d'activation $\mathbf{o}^a$ comme cela :

$$\mathbf{o}^a = \mathbf{W}^{(2)}\mathbf{h_s} + \mathbf{b}^{(2)} \tag{3}$$

Si l'on détail l'expression pour chaque composante on obtient :

$$\begin{aligned}\mathbf{o}^a_j &= \mathbf{W}^{(2)}_j\mathbf{h_s} + \mathbf{b}^{(2)}_j \\ &= \sum_{i=1}^{m} \mathbf{W}^{(2)}_{j,i}\mathbf{h_s}_i + \mathbf{b}^{(2)}_j\end{aligned}$$

c) On sait que la fonction exponentiel est définit strictement positive les o^s_k seront donc obligatoirement positifs. De plus on démontre que les o^s_k sommes à 1 :

$$\begin{aligned}\sum_{i=1}^{m} \mathbf{o}^s_i &= \sum_{i=1}^{m} \frac{\exp(\mathbf{o}^a_i)}{\sum_{k'=1}^{m}\exp(\mathbf{o}^a_{k'})} \\ &= \frac{\sum_{i=1}^{m}\exp(\mathbf{o}^a_i)}{\sum_{k'=1}^{m}\exp(\mathbf{o}^a_{k'})} \\ \sum_{i=1}^{m} \mathbf{o}^s_i &= 1\end{aligned}$$

d) On formule le risque empirique $\hat{R}$ comme ceci :

$$\begin{aligned}\hat{R}(f_\theta, D_n) &= \frac{1}{n}\sum_{i=1}^{n} L(f_\theta(x^{(i)}), y^{(i)}) \\ &= \frac{1}{n}\sum_{i=1}^{n} -\log \mathbf{o}^s_{y^{(i)}}(x^{(i)})\end{aligned}$$

Soit l'ensemble des paramètres $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}\}$ sa dimensionnalité sera donc :

$$\begin{aligned} n_\theta &= d_h \times d + d_h + m \times d_h + m \\ &= d_h \times (d+1) + m \times (d_h + 1) \end{aligned}$$

On cherche à minimiser le risque empirique en trouvant la valeur optimale des paramètres :

$$\theta^\star = \underset{\theta}{arg\,min}\ \hat{R}(f_\theta, D_n)$$

Algorithme 1: Minimisation du risque empirique $\hat{R}$ par descente de gradient batch

---

Données : $\mathbf{x}, t, \eta$
W1 $\leftarrow$ `random(`*dh, d*`)`;
b1 $\leftarrow 0$;
W2 $\leftarrow$ `random(`*m, dh*`)`;
b2 $\leftarrow 0$;
$\theta \leftarrow \{\mathbf{W1}, \mathbf{b1}, \mathbf{W2}, \mathbf{b2}\}$;

e)

// k représente les époques
pour $k \leftarrow 1$ **à** 10^5 **faire**
  pour $i \leftarrow 1$ **à** n **faire**
    $\theta \leftarrow \theta - \eta \sum \frac{\partial L(\mathbf{x}^{(i)}, t^{(i)})}{\partial \theta}$;
  fin
fin
retourner (θ);

---

h) On a $\mathbf{o}^s$ de taille m et une fonction onehot(y) qui retourne un vecteur de taille m :

$$\frac{\partial L}{\partial \mathbf{o}^a} = \mathbf{o}^s - \text{onehot}(y)$$

On peut traduire ce calcul en «numpy» :

```
grad_oa = ( numpy.exp( oa ) / numpy.sum(numpy.exp( oa )) ) - onehot( y )
```

j) On a $\frac{\partial L}{\partial \mathbf{o}^a}$ de taille m et le vecteur de sortie de la couche cachée h^s de taille d_h. On traduit les calculs de $\frac{\partial L}{\partial \mathbf{W}^{(2)}}$ et $\frac{\partial L}{\partial \mathbf{b}^{(2)}}$ en «numpy» :

```
grad_W2 = numpy.dot( grad_oa, numpy.transpose( hs ) )
grad_b2 = grad_oa
```

l) On a $\frac{\partial L}{\partial \mathbf{o}^a}$ de taille m et la matrice de poids $\mathbf{W}^{(2)}$ de taille $d_h \times m$.
On traduit le calcul de $\frac{\partial L}{\partial \mathbf{h}^s}$ en «numpy» :

```
grad_hs = numpy.dot( numpy.transpose( W2 ), grad_oa )
```

m) On calcul d'abord la dérivés de $\tanh(x)$:

$$\begin{aligned}
\tanh'(x) &= \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)' \\
&= \frac{(e^x - e^{-x})'(e^x + e^{-x}) - (e^x - e^{-x})(e^x + e^{-x})'}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
\tanh'(x) &= 1 - \tanh^2(x)
\end{aligned}$$

On peut ensuite calculer $\frac{\partial L}{\partial \mathbf{h}_j^a}$:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{h}_j^a} &= \frac{\partial L}{\partial \mathbf{h}_j^s}\frac{\partial \mathbf{h}_j^s}{\partial \mathbf{h}_j^a} \\
&= \frac{\partial L}{\partial \mathbf{h}_j^s}\frac{\partial \tanh(\mathbf{h}_j^a)}{\partial \mathbf{h}_j^a} \\
\frac{\partial L}{\partial \mathbf{h}_j^a} &= \frac{\partial L}{\partial \mathbf{h}_j^s}\left(1 - \tanh^2(\mathbf{h}_j^a)\right)
\end{aligned}$$

n) On peut définir le calcul précédent sous la forme d'une expression matricielle, avec le vecteur $\frac{\partial L}{\partial \mathbf{h}^s}$ et le vecteur résultant de l'application de la fonction $\tanh^2$ sur $\frac{\partial L}{\partial \mathbf{h}^a}$, tout les deux de taille d_h.

$$\frac{\partial L}{\partial \mathbf{h}^a} = \frac{\partial L}{\partial \mathbf{h}^s}\left(1 - \tanh^2(\mathbf{h}^a)\right)$$

On traduit ce calcul en «numpy» :

```
grad_ha = grad_hs * (1 - numpy.square(numpy.tanh( ha ))
```

o) On calcul les gradients $\frac{\partial L}{\partial \mathbf{W}_{kj}^{(1)}}$ et $\frac{\partial L}{\partial \mathbf{b}_k^{(1)}}$:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}_{kj}^{(1)}} &= \frac{\partial L}{\partial \mathbf{h}_k^a}\frac{\partial \mathbf{h}_k^a}{\partial \mathbf{W}_{kj}^{(1)}} \\
&= \frac{\partial L}{\partial \mathbf{h}_k^a}\frac{\partial}{\partial \mathbf{W}_{kj}^{(1)}}\sum_{j'} \mathbf{W}_{kj'}^{(1)}\boldsymbol{x}_{j'} + \mathbf{b}_k^{(1)} \\
\frac{\partial L}{\partial \mathbf{W}_{kj}^{(1)}} &= \frac{\partial L}{\partial \mathbf{h}_k^a}\boldsymbol{x}_j
\end{aligned}$$

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{b}_k^{(1)}} &= \frac{\partial L}{\partial \mathbf{h}_k^a}\frac{\partial \mathbf{h}_k^a}{\partial \mathbf{b}_k^{(1)}} \\
&= \frac{\partial L}{\partial \mathbf{h}_k^a}\frac{\partial}{\partial \mathbf{b}_k^{(1)}}\sum_{j'} \mathbf{W}_{kj'}^{(1)}\boldsymbol{x}_{j'} + \mathbf{b}_k^{(1)} \\
\frac{\partial L}{\partial \mathbf{b}_k^{(1)}} &= \frac{\partial L}{\partial \mathbf{h}_k^a}
\end{aligned}$$

p) On exprime le calcul précédent sous forme matricielle :

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{h}^a} \boldsymbol{x}^T$$
$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{\partial L}{\partial \mathbf{h}^a}$$

avec $\frac{\partial L}{\partial \mathbf{h}^a}$ et $\boldsymbol{x}$ deux vecteurs colonne de tailles respectives d_h et d. On traduit ce calcul en «numpy» :

```
grad_W1 = numpy.dot( grad_ha, numpy.transpose( x ) )
grad_b1 = grad_ha
```

q) On sait que L dépend d'un neurone, de la couche d'entrée, $\boldsymbol{x}_j$ au travers de tous les neurones de la couche cacher $\mathbf{h}^a$ relié à se neurone, on a, grâce à la règle de dérivation en chaine :

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{x}_j} &= \sum_{k=1}^{d_h} \frac{\partial L}{\partial \mathbf{h}_k^a} \frac{\partial \mathbf{h}_k^a}{\partial \boldsymbol{x}_j} \\
&= \sum_{k=1}^{d_h} \frac{\partial L}{\partial \mathbf{h}_k^a} \frac{\partial}{\partial \boldsymbol{x}_j} \sum_{j'} \mathbf{W}_{kj'}^{(1)} \boldsymbol{x}_{j'} + \mathbf{b}_k^{(1)} \\
\frac{\partial L}{\partial \boldsymbol{x}_j} &= \sum_{k=1}^{d_h} \frac{\partial L}{\partial \mathbf{h}_k^a} \mathbf{W}_{kj}^{(1)}
\end{aligned}$$

r) On veut cette fois minimiser le risque régularisé $\tilde{R}$:

$$\begin{aligned}
\tilde{R}(f_\theta, D_n) &= \hat{R}(f_\theta, D_n) + \lambda \mathcal{L}(\theta) \\
&= \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x^{(i)}), y^{(i)}) + \lambda \left(\|\mathbf{W}^{(1)}\|^2 + \|\mathbf{W}^{(2)}\|^2 \right)
\end{aligned}$$

Il nous faut donc calculer le gradient de $\tilde{R}$. Ce qui revient à calculer le gradient

de $\hat{R}$ et de $\mathcal{L}$.

$$\begin{aligned}
\frac{\partial \mathcal{L}(\theta)}{\partial \theta} &= \begin{pmatrix} \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{W}^{(1)}} \\ \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{b}^{(1)}} \\ \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{W}^{(2)}} \\ \frac{\partial \mathcal{L}(\theta)}{\partial \mathbf{b}^{(2)}} \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i,j} \frac{\partial}{\partial \mathbf{W}^{(1)}} \left(\mathbf{W}^{(1)}_{i,j}\right)^2 + \sum_{i',j'} \frac{\partial}{\partial \mathbf{W}^{(1)}} \left(\mathbf{W}^{(2)}_{i',j'}\right)^2 \\ \sum_{i,j} \frac{\partial}{\partial \mathbf{b}^{(1)}} \left(\mathbf{W}^{(1)}_{i,j}\right)^2 + \sum_{i',j'} \frac{\partial}{\partial \mathbf{b}^{(1)}} \left(\mathbf{W}^{(2)}_{i',j'}\right)^2 \\ \sum_{i,j} \frac{\partial}{\partial \mathbf{W}^{(2)}} \left(\mathbf{W}^{(1)}_{i,j}\right)^2 + \sum_{i',j'} \frac{\partial}{\partial \mathbf{W}^{(2)}} \left(\mathbf{W}^{(2)}_{i',j'}\right)^2 \\ \sum_{i,j} \frac{\partial}{\partial \mathbf{b}^{(2)}} \left(\mathbf{W}^{(1)}_{i,j}\right)^2 + \sum_{i',j'} \frac{\partial}{\partial \mathbf{b}^{(2)}} \left(\mathbf{W}^{(2)}_{i',j'}\right)^2 \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i,j} \frac{\partial}{\partial \mathbf{W}^{(1)}} \left(\mathbf{W}^{(1)}_{i,j}\right)^2 \\ 0 \\ \sum_{i',j'} \frac{\partial}{\partial \mathbf{W}^{(2)}} \left(\mathbf{W}^{(2)}_{i',j'}\right)^2 \\ 0 \end{pmatrix} \\
\frac{\partial \mathcal{L}(\theta)}{\partial \theta} &= 2 \begin{pmatrix} \sum_{i,j} \mathbf{W}^{(1)}_{i,j} \\ 0 \\ \sum_{i',j'} \mathbf{W}^{(2)}_{i',j'} \\ 0 \end{pmatrix}
\end{aligned}$$

On en déduit le gradient de $\tilde{R}$:

$$\begin{aligned}
\frac{\partial \tilde{R}}{\partial \theta} &= \frac{\partial \hat{R}}{\partial \theta} + \lambda \frac{\partial \mathcal{L}(\theta)}{\partial \theta} \\
&= \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \frac{\partial L}{\partial \mathbf{h}^a} (\mathbf{x}^{(i)})^T \\ \frac{\partial L}{\partial \mathbf{h}^a} \\ \frac{\partial L}{\partial \mathbf{o}^a} (h^s)^T \\ \frac{\partial L}{\partial \mathbf{o}^a} \end{pmatrix} + 2\lambda \begin{pmatrix} \sum_{i,j} \mathbf{W}^{(1)}_{i,j} \\ 0 \\ \sum_{i',j'} \mathbf{W}^{(2)}_{i',j'} \\ 0 \end{pmatrix}
\end{aligned}$$

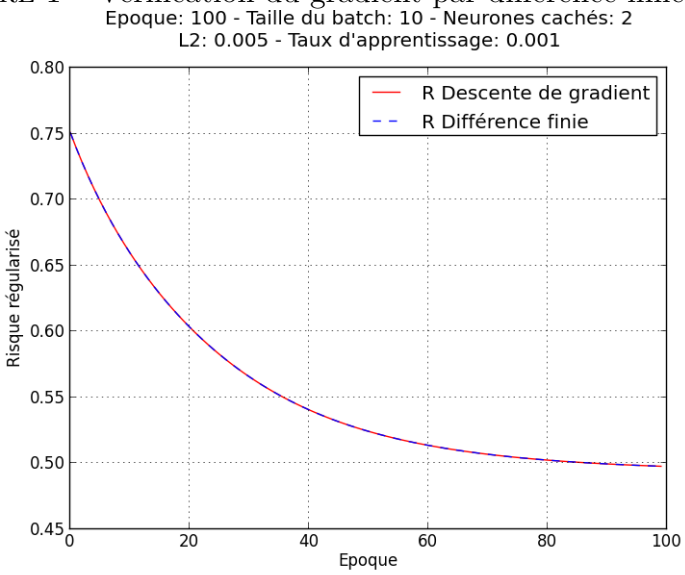
2 Implémentation du réseau de neurones

4. Pour la vérification du gradient par différence finie j'ai choisi d'effectué deux descentes de gradients sur les même données (mélangé de la même manière avec les même «graine» d'initialisation). L'une utilisant la différence finie centrée :

$$\delta[f](x) = f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

et l'autre utilisant mon implémentation de la descente de gradient par «mini-lots».

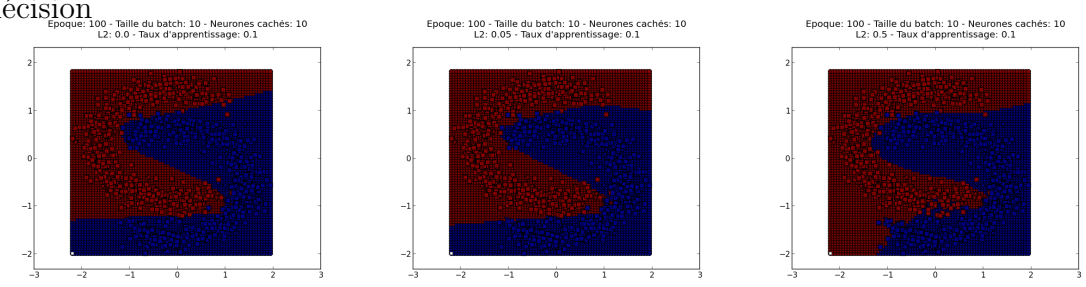
FIGURE 1 – Vérification du gradient par différence finie centrée



Comme on le constate sur cette figure la descente de gradient et la différence finie donne les même résultats. On peu donc conclure que le calcule de la descente de gradient est correct.

5. Le taux d'apprentissage et la pénalité sont très liés. Après plusieurs essai j'ai décidé de fixer le taux d'apprentissage à 0.1 pour pouvoir comparer l'effet de la pénalité sur la position de la frontiere de decision.

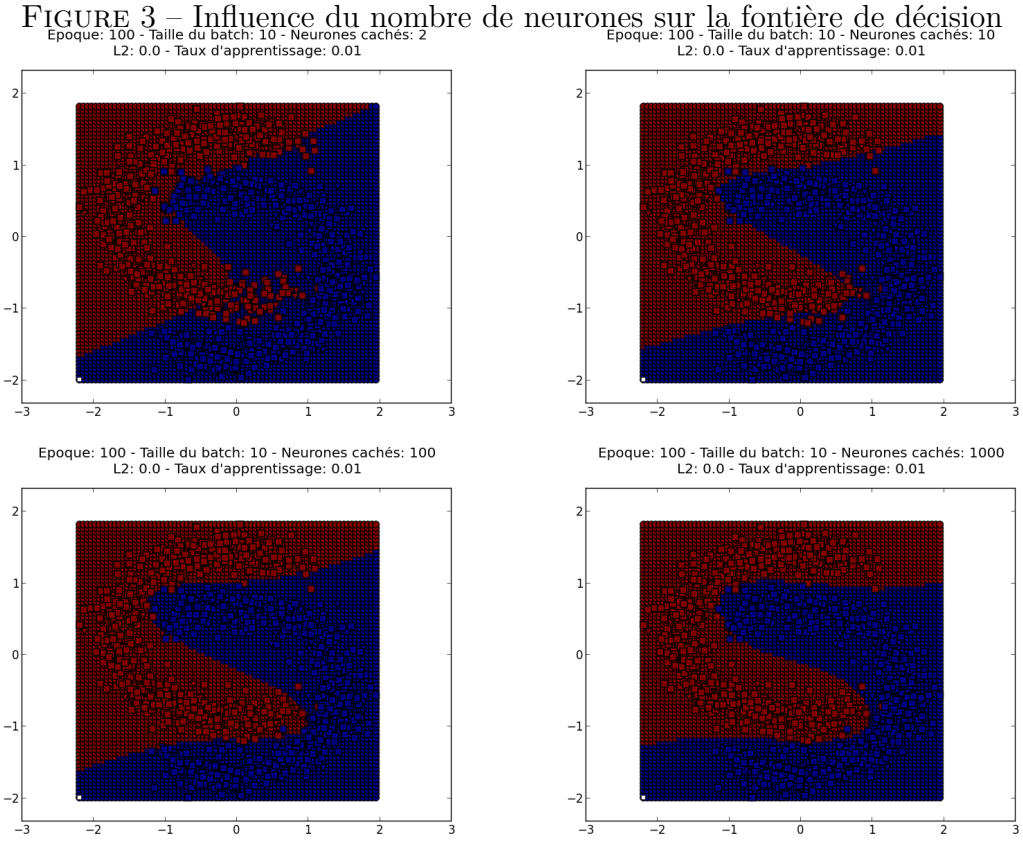
FIGURE 2 – Influence de la pénalité (de type «weight decay») sur la fontière de décision



On constate que pour une valeur relativement modérée de λ on obtient une frontière de décision moins linéaire et une meilleur «adaptation» aux données. En revanche pour une valeur trop élevé on obtient de plus mauvais résultats.

Pour comparer l'influence du nombre de neurones j'ai choisi de fixer $\lambda = 0$.

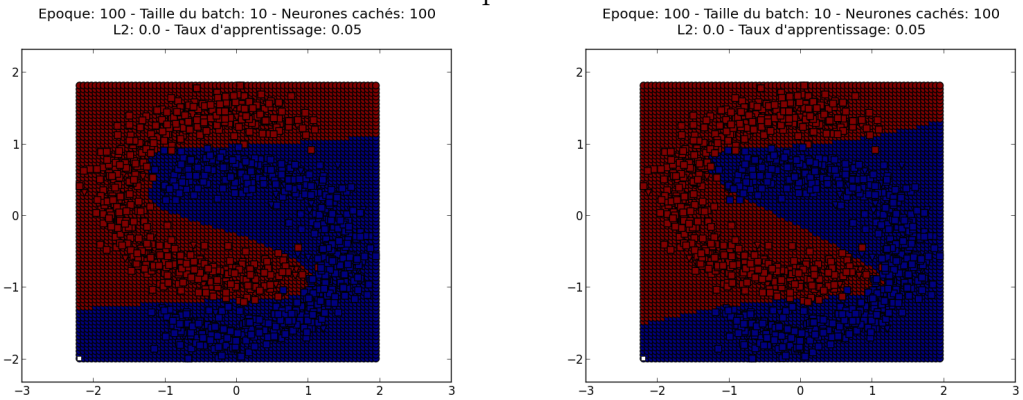
FIGURE 3 – Influence du nombre de neurones sur la fontière de décision



L'augmentation du nombre de neurones cachés influ considérablement sur la frontière de décision. Plus le nombre de neurones est grand plus la frontière de décision va «suivre» les poins d'entrainements. Mais il y a alors un risque de surapprentissage qui advient.

Pour comparer l'arrêt prématuré j'ai fixé le nombre époques à 100.

FIGURE 4 – Influence de l'arrêt prématuré sur la fontière de décision



Dans cet exemple l'arrêt prématuré (à droite) a été effectué à la $26^{ème}$ époque. On constate que l'arrêt prématuré à tendance éviter le surapprentissage. Mais il est trop dépendant du taux d'apprentissage selon moi, et l'on risque de rester bloqué dans un minima local.

7. J'ai choisi de comparer les frontière de décision pour juger mon implémentation matricielle.

FIGURE 5 – Comparaison de la fontière de décision généré par les deux algorithmes (par boucle à gauche et matriciel à droite) pour K = 1

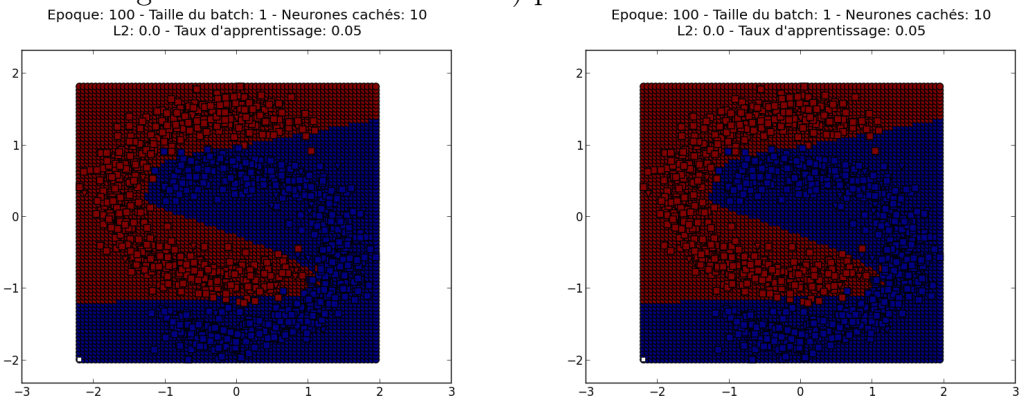
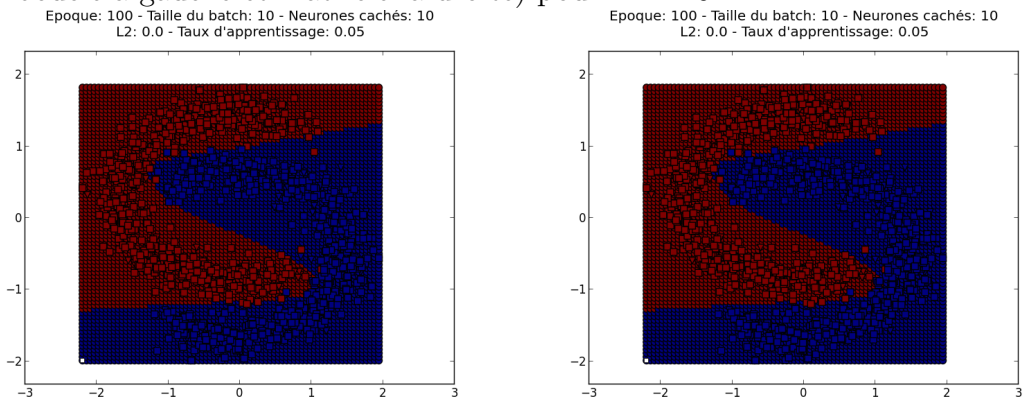


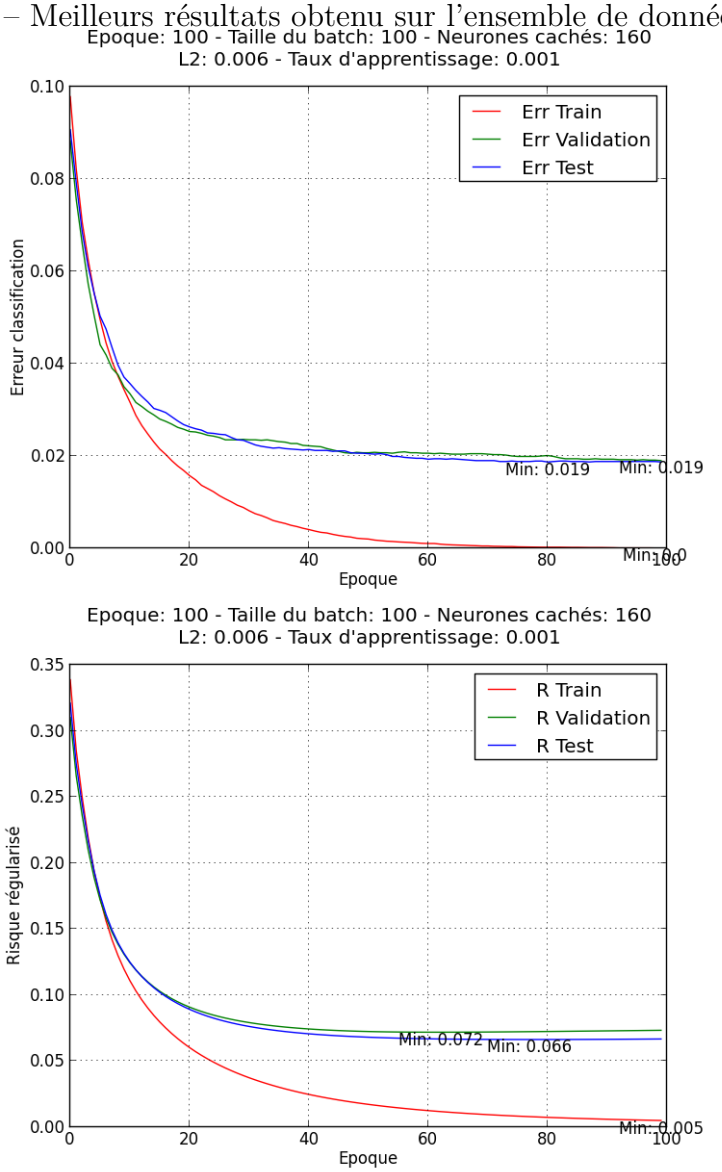
FIGURE 6 – Comparaison de la fontière de décision généré par les deux algorithmes (par boucle à gauche et matriciel à droite) pour K = 10



Les résultats sont exactement les mêmes, le calcul matricielle est donc correct.

8. Les résultats mesurés sur une époque de MNIST sont les suivants :
   – 7.750s pour le calcul par boucle
   – 1.770s pour le calcul matriciel
   Soit une diminution du temps de calcul de 70% environ.
9. Les données sont enregistrer dans un fichier nommé **data.csv** dans le dossier courrant.
10. J'ai réussi à obtenir jusqu'à 1.9% d'erreur sur les ensembles de test et de validation avec les hyper paramètres indiqués sur la figure suivante et sans arrêt prématuré.

FIGURE 7 – Meilleurs résultats obtenu sur l'ensemble de données MNIST



3 Annexes

3.1 Utilisation du programme

Le programme accepte un certain nombre d'arguments listé ci-dessous :

-fast active le calcul matriciel (activé par défaut)

-slow active le calcul par boucle

-finiteDif calcul la différence finie (et produit des courbes comparatives)

-display 0/1 affiche les graphiques (0 pour désactiver, 1 pour activer)

-data MNIST/2MOONS permet de choisir le jeu de données

-lr x indique le taux d'apprentissage

-wd x indique la pénalité (λ)

-epoch x indique le nombre d'époques à effectuer

-batch x indique la taille des lots

-hidden x indique le nombre de neurones dans la couche cachée

-es 0/1 active/désactive l'arrêt prématuré (0 pour désactiver, 1 pour activer)