IFT 3395/6390 Automne 2015
Fondements de l'apprentissage machine
Professeur: Pascal Vincent

Devoir 2 (partie théorique) et Devoir 3 (partie pratique)

REMISE: Remettez un rapport en format électronique (rapport.pdf). Pour un rapport avec des équations correctement typographiées, vous pouvez par exemple utiliser le logiciel Lyx. Tous les fichiers de code source que vous aurez créé ou adapté, et les fichiers de résultat produits (ex. courbes) devront également être remis. La remise se fait via le système StudiUM. Si vous avez de nombreux fichiers faites en une archive (.zip) et téléversez le fichier d'archive.

Ce devoir est à faire par groupe de 2 ou 3. Ne remettez qu'un seul rapport par groupe sur StudiUM (un de smembres du groupe effectue la remise). Mais assurez-vous d'avoir clairement indiqué le nom de tous les coéquipiers en tête de votre rapport.

Ce travail a pour objectif de vous familiariser avec le calcul de gradients par rétropropagation dans les réseaux de neurones, de vous permettre d'implanter un réseau de neurone de type MLP pour la classification multiclasse, et d'expérimenter avec. Comme point de départ, référez-vous à la section des notes de cours sur les réseaux de neurones. Aussi, pour vous simplifier la tâche, la première partie du devoir comporte un corrigé partiel de certaines questions.

Dans votre rapport remis, ne recopiez les réponses de la partie 1 qui sont déjà dans ce corrigé partiel. Mais n'oubliez pas d'inclure les réponses de la partie 1 qui ne sont pas dans ce corrigé...

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

On suppose qu'on dispose d'un ensemble de données $D_n = \{(\boldsymbol{x}^{(1)}, y^{(1)}), ..., (\boldsymbol{x}^{(n)}, y^{(n)})\}$ avec $x^{(i)} \in \mathbb{R}^d$ et $y^{(i)} \in \{1, ..., m\}$ indiquant la classe parmi m classes. Pour les expressions vectorielles et matricielles les vecteurs sont par défaut considérés comme des vecteurs colonnes.

Soit un réseau de neurones de type *Perceptron multicouche* avec une seule couche cachée (donc 3 couches en tout si on compte la couche d'entrée et la couche de sortie). La couche cachée est constituée de d_h neurones complètement connectés à la couche d'entrée. Nous allons considérer pour la couche cachée une non-linéarités de type **rectifieur** (Rectified Linear Unit ou **RELU**). La couche de sortie est constituée de m neurones, complètement connectés à la couche cachée. Ils ont une non-linéarité de type **softmax**. La sortie du $j^{\text{ème}}$ neurone de la couche de sortie donnera un score pour la classe j interprété comme la probabilité que l'entrée x soit de cette classe j.

Il vous est fortement conseillé de dessiner le réseau de neurones au fur et à mesure afin que vous puissiez mieux suivre les étapes (mais pas besoin de nous fournir un dessin!)

a) Soit $\boldsymbol{W}^{(1)}$ la matrice $d_h \times d$ de poids et soit $\boldsymbol{b}^{(1)}$ le vecteur de biais caractérisant des connexions synaptiques allant de la couche d'entrée à la couche cachée. Indiquez la dimension de $\boldsymbol{b}^{(1)}$. Donnez la formule de calcul du vecteur d'activations (i.e. avant non-linéarité) des neurones de la couche cachée $\boldsymbol{h}^a$ à partir d'une observation d'entrée $\boldsymbol{x}$, d'abord sous la forme d'une expression de calcul matriciel, puis détaillez le calcul d'un éléments $\boldsymbol{h}^a_j$. Exprimez le vecteur des sorties des neurones de la couche cachée $\boldsymbol{h}^s$ en fonction de $\boldsymbol{h}^a$. Indication: référez-vous aux notes de cours pour la non-linéarité de type **rectifieur**

b) Soit $\boldsymbol{W}^{(2)}$ la matrice de poids et soit $\boldsymbol{b}^{(2)}$ le vecteur de biais caractérisant les connexions synaptiques allant de la couche cachée à la couche de sortie. Indiquez les dimensions de $\boldsymbol{W}^{(2)}$ et $\boldsymbol{b}^{(2)}$. Donnez la formule de calcul du vecteur d'activations des neurones de la couche de sortie $\boldsymbol{o}^a$ à partir de leurs entrées $\boldsymbol{h}^s$ sous la forme d'une expression de calcul matriciel, puis détaillez le calcul de $\boldsymbol{o}^a_k$.

c) La sortie des neurones de sortie est donnée par

$$\boldsymbol{o}^s = \text{softmax}(\boldsymbol{o}^a)$$

Précisez l'équation des $\boldsymbol{o}^s_k$ en utilisant explicitement la formule du softmax (formule avec des exp). *Démontrez* que les $\boldsymbol{o}^s_k$ sont positifs et somment à 1. Pourquoi est-ce important?

d) Le réseau de neurones calcule donc, pour un vecteur d'entrée $\boldsymbol{x}$, un vecteur de scores (probabilités) $\boldsymbol{o}^s(\boldsymbol{x})$. La probabilité, calculée par le réseau de neurones, qu'une observation $\boldsymbol{x}$ soit de la classe y est donc donnée par la $y^{\text{ième}}$ sortie $\boldsymbol{o}^s_y(\boldsymbol{x})$. Ceci suggère d'utiliser la fonction de perte:

$$L(\boldsymbol{x}, y) = \text{-log } \boldsymbol{o}^s_y(\boldsymbol{x})$$

Précisez l'équation de L directement en fonction du vecteur $\boldsymbol{o}^a$. Il suffit pour cela d'y substituer convenablement l'équation exprimée au point précédent.

e) L'entraînement du réseau de neurones va consister à trouver les paramètres du réseau qui minimisent le risque empirique $\hat{R}$ correspondant à cette fonction de perte. Formulez $\hat{R}$. Indiquez précisément de quoi est constitué l'ensemble θ des paramètres du réseau. Indiquez à combien de paramètres scalaires n_θ cela correspond. Formulez le problème d'optimisation qui correspond à l'entraînement du réseau permettant de trouver une valeur optimale des paramètres.

f) Pour trouver la solution à ce problème d'optimisation, on va utiliser une technique de descente de gradient. Exprimez sous forme d'un bref pseudo-code la technique de descente de gradient (batch) pour ce problème.

g) Notez que le calcul du vecteur de gradient du risque empirique $\hat{R}$ par rapport à l'ensemble des paramètres θ peut s'exprimer comme

$$\begin{pmatrix} \frac{\partial \hat{R}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \hat{R}}{\partial \theta_{n_\theta}} \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \frac{\partial L(\boldsymbol{x}_i, y_i)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\boldsymbol{x}_i, y_i)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

Il suffit donc de savoir calculer le gradient du coût L encouru pour un exemple $(\boldsymbol{x}, y)$ par rapport aux paramètres, que l'on définit comme:

$$\frac{\partial L}{\partial \theta} = \begin{pmatrix} \frac{\partial L}{\partial \theta_1} \\ \vdots \\ \frac{\partial L}{\partial \theta_{n_\theta}} \end{pmatrix} = \begin{pmatrix} \frac{\partial L(\boldsymbol{x}, y)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\boldsymbol{x}, y)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

Pour cela on va appliquer la technique de **rétropropagation du gradient**, en partant du coût L, et en remontant de proche en proche vers la sortie $\boldsymbol{o}$ puis vers la couche cachée $\boldsymbol{h}$ et enfin vers l'entrée $\boldsymbol{x}$.

Démontrez que

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

Indication: Partez de l'expression de L en fonction de $\boldsymbol{o}^a$ que vous avez précisée plus haut. Commencez par calculer $\frac{\partial L}{\partial \boldsymbol{o}^a_k}$ pour $k \neq y$ (en employant au début l'expression de la dérivée du logarithme). Procédez de manière similaire pour $\frac{\partial L}{\partial \boldsymbol{o}^a_y}$.

h) Donnez l'expression correspondante du gradient en numpy (possiblement en 2 opérations successives).

grad_oa = ...

...

IMPORTANT: Dorénavant quand on demande de "calculer" des gradients ou dérivées partielles, il s'agit simplement d'exprimer leur calcul en fonction d'éléments déjà calculés aux questions précédentes (**ne substituez pas les expressions de dérivées partielles déjà calculées lors des questions d'avant!**)

i) Calculez les gradients par rapport aux paramètres $\boldsymbol{W}^{(2)}$ et $\boldsymbol{b}^{(2)}$ de la couche de sortie. Comme L ne dépend des $\boldsymbol{W}^{(2)}_{kj}$ et $\boldsymbol{b}^{(2)}_k$qu'au travers de $\boldsymbol{o}^a_k$ la règle de dérivation en chaîne nous donne:

$$\frac{\partial L}{\partial \boldsymbol{W}^{(2)}_{kj}} = \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial \boldsymbol{o}^a_k}{\partial \boldsymbol{W}^{(2)}_{kj}}$$

et

$$\frac{\partial L}{\partial \boldsymbol{b}^{(2)}_k} = \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial \boldsymbol{o}^a_k}{\partial \boldsymbol{b}^{(2)}_k}$$

RÉPONSE: Gradient par rapport aux paramètres $\boldsymbol{W}^{(2)}$ et $\boldsymbol{b}^{(2)}$

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{b}^{(2)}_k} &= \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial \boldsymbol{o}^a_k}{\partial \boldsymbol{b}^{(2)}_k} \\
&= \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial}{\partial \boldsymbol{b}^{(2)}_k} \sum_{j'} \boldsymbol{W}^{(2)}_{kj'} h^s_{j'} + \boldsymbol{b}^{(2)}_k \\
&= \frac{\partial L}{\partial \boldsymbol{o}^a_k} \\
\frac{\partial L}{\partial \boldsymbol{W}^{(2)}_{kj}} &= \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial \boldsymbol{o}^a_k}{\partial \boldsymbol{W}^{(2)}_{kj}} \\
&= \frac{\partial L}{\partial \boldsymbol{o}^a_k} \frac{\partial}{\partial \boldsymbol{W}^{(2)}_{kj}} \sum_{j'} \boldsymbol{W}^{(2)}_{kj'} h^s_{j'} + \boldsymbol{b}^{(2)}_k \\
&= \frac{\partial L}{\partial \boldsymbol{o}^a_k} h^s_j
\end{aligned}$$

j) Exprimez le calcul du gradient de la question précédente sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulés.

RÉPONSE :

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} &= \frac{\partial L}{\partial \boldsymbol{o}^a} \\
\frac{\partial L}{\partial \boldsymbol{W}^{(2)}} &= \frac{\partial L}{\partial \boldsymbol{o}^a} (h^s)^T
\end{aligned}$$

où $\frac{\partial L}{\partial \boldsymbol{o}^a}$ et h^s sont des vecteurs colonnes, et ^{T} dénote la transposée.

Précisez les dimensions...

Assurez-vous de bien comprendre pourquoi ces expressions matricielles sont équivalentes aux expressions de la question précédente.

Donnez l'expression correspondante en numpy.

grad_b2 = ...

grad_W2 = ...

k) Calculez les dérivées partielles du coût L par rapport aux sorties des neurones de la couche cachée. Comme L dépend d'un neurone caché $\boldsymbol{h}_j^s$ au travers des activations de tous les neurones de sortie $\boldsymbol{o}^a$ reliés à ce neurone caché, la règle de dérivation en chaîne nous donne:

$$\frac{\partial L}{\partial \boldsymbol{h}_j^s} = \sum_{k=1}^{m} \frac{\partial L}{\partial \boldsymbol{o}_k^a} \frac{\partial \boldsymbol{o}_k^a}{\partial \boldsymbol{h}_j^s}$$

RÉPONSE: Gradient par rapport aux sorties h_j^s des neurons de la couche cachée

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{h}_j^s} &= \sum_{k=1}^{m} \frac{\partial L}{\partial \boldsymbol{o}_k^a} \frac{\partial \boldsymbol{o}_k^a}{\partial \boldsymbol{h}_j^s} \\ &= \sum_{k=1}^{m} \frac{\partial L}{\partial \boldsymbol{o}_k^a} \left(\frac{\partial}{\partial \boldsymbol{h}_j^s} \sum_{j'} \boldsymbol{W}_{kj'}^{(2)} \boldsymbol{h}_{j'}^s + \boldsymbol{b}_k^{(2)} \right) \\ &= \sum_{k=1}^{m} \frac{\partial L}{\partial \boldsymbol{o}_k^a} \boldsymbol{W}_{kj}^{(2)}\end{aligned}$$

l) Exprimez le calcul de la question précédente sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulées.

RÉPONSE :

$$\frac{\partial L}{\partial \boldsymbol{h}^s} = \boldsymbol{W}^{(2)^T} \frac{\partial L}{\partial \boldsymbol{o}^a}$$

où $\frac{\partial L}{\partial \boldsymbol{o}^a}$ est un vecteurs colonne, et ^{T} dénote la transposée.

Précisez les dimensions...

Assurez-vous de bien comprendre pourquoi cette expression matricielle est équivalente aux calculs de la question précédente.

Donnez l'expression correspondante en numpy.

grad_hs = ...

m) Calculez les dérivées partielles par rapport aux activations des neurones de la couche cachée. Comme L ne dépend de l'activation $\boldsymbol{h}_j^a$ d'un neurone de la couche cachée qu'au travers de la sortie $\boldsymbol{h}_j^s$ de ce neurone, la règle de dérivation en chaîne donne:

$$\frac{\partial L}{\partial \boldsymbol{h}_j^a} = \frac{\partial L}{\partial \boldsymbol{h}_j^s} \frac{\partial \boldsymbol{h}_j^s}{\partial \boldsymbol{h}_j^a}$$

Notez que $\boldsymbol{h}_j^s = \text{rect}(\boldsymbol{h}_j^a)$: la fonction rectifieur s'applique élément par élément. Comme étape intermédiaire, commencez par exprimer la dérivée de la fonction rectifieur $\frac{\partial \text{rect}(z)}{\partial z} = \text{rect}'(z) = \ldots$.

n) Exprimez le calcul de la question précédente sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulés. Donnez l'expression équivalente en numpy.

o) Calculez les gradients par rapport aux éléments des paramètres $\boldsymbol{W}^{(1)}$ et $\boldsymbol{b}^{(1)}$ de la couche cachée.

Indication: c'est le même principe que pour une question antérieure

p) Exprimez ce calcul du gradient de la question précédente sous forme d'une expression matricielle, en définissant la dimension de chacune des matrices ou vecteurs manipulées. Donnez l'expression équivalente en numpy.

Indication: c'est le même principe que pour une question antérieure

q) Calculez les dérivées partielles du coût L par rapport au vecteur d'entrée $\boldsymbol{x}$.
Indication: c'est le même principe que pour une question antérieure

r) Nous allons maintenant considérer un risque empirique **régularisé**: $\tilde{R} = \hat{R} + \lambda \mathcal{L}(\theta)$, où θ est le vecteur de tous les paramètres du réseau et $\mathcal{L}(\theta)$ calcule une pénalité scalaire en fonction des paramètres θ, plus ou moins importante selon une préférence à priori qu'on a sur les valeurs de θ. λ est un hyper-paramètre (scalaire, positif ou nul) qui contrôle le compromis entre trouver des valeurs des paramètres qui minimisent le risque empirique ou qui minimisent cette pénalité. On va considérer ici une régularisation de type "weight decay" quadratique qui pénalise la norme carrée (norme L_2) des poids (mais pas des biais):

$$\begin{aligned}\mathcal{L}(\theta) &= \|\boldsymbol{W}^{(1)}\|^2 + \|\boldsymbol{W}^{(2)}\|^2 \\ &= \sum_{i,j} \left(\boldsymbol{W}^{(1)}_{ij}\right)^2 + \sum_{i',j'} \left(\boldsymbol{W}^{(2)}_{i'j'}\right)^2\end{aligned}$$

On veut en fait minimiser le risque régularisé $\tilde{R}$ plutôt que $\hat{R}$. Comment cela change-t-il le gradient par rapport aux différents paramètres?

2 PARTIE PRATIQUE: Implémentation du réseau de neurones

On vous demande d'implémenter le réseau de neurones avec le calcul du gradient pas à pas tel que vous l'avez dérivé dans la question précédente (incluant le weight decay). Vous ne pouvez pas utiliser une implémentation existante de réseau de neurones, mais devez suivre la structure de la derivation faite à la question 1 (avec des noms de variables s'en inspirant, etc.). Notez qu'avec les démos faites précédemment en Python vous avez déjà toute une structure d'algorithme d'apprentissage, de calcul et d'affichage de régions de décision 2D dont vous pouvez vous inspirer et peut-être (partiellement) réutiliser...

Détails utiles sur l'implémentation:

- **Implémentation numériquement stable de softmax**. Vous aurez besoin de pouvoir calculer un softmax de manière numériquement stable. Référez-vous aux notes de cours pour la manière de calculer un softmax numériquement stable. Commencez par écrire le softmax d'un unique vecteur. Puis étendez votre code pour qu'il fonctionne aussi avec un mini-lot (min-batch) de plusieurs vecteurs stockés dans une matrice.
- **Initialisation des paramètres.** Comme vous le savez, il est nécessaire d'initialiser aléatoirement les paramètres du réseau (dans le but d'éviter les symétries et la saturation des neurones et idéalement pour se situer au point d'inflexion de la non-linéarité de façon à avoir un comportement non-linéaire). Nous vous proposons d'initialiser les *poids* d'une couche en les tirant d'une uniforme sur $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$, où n_c est le nombre d'entrées de **cette couche** (le nombre de neurones d'entrée auxquels chaque neurone de cette couche est connecté, donc ça change typiquement d'une couche à l'autre). Les *biais* peuvent quant à eux être initialisés à 0. Justifiez votre choix de toute autre initialisation.
- **fprop** et **bprop**. On vous suggère d'écrire des méthodes fprop et bprop. fprop fait la propagation "avant" c.a.d. le calcul étape par étape depuis l'entrée, jusqu'à la sortie et au coût, des activations de chaque couche. bprop se sert des activations calculées par le précédent appel à fprop et effectue la rétropropagation du gradient depuis le coût jusqu'à l'entrée en suivant précisément les étapes dérivées dans la partie 1.

- **Verification du gradient par différence finie.** On peut estimer le gradient numériquement par différences finies. Vous devez implémenter cette estimation de façon à vérifier votre calcul du gradient (c'est un outil de développement). Pour ce faire, calculez d'abord la valeur de la perte pour la valeur courante des paramètres (sur un exemple, ou un lot). Ensuite, pour chaque paramètre θ_k (un scalaire), modifiez ce paramètre par une petite valeur ($10^{-6} < \varepsilon < 10^{-4}$) et recalculez la perte (même exemple ou lot) puis ramenez le paramètre à sa valeur de départ. La dérivée partielle (le gradient) par rapport à chaque paramètre est alors estimé en divisant la variation de la perte par ε. Le ratio de votre gradient calculé par rétropropagation et du gradient estimé par différence finie devrait se situer entre 0.99 et 1.01.

- **Taille des lots**. On demande que votre calcul et descente de gradient opère sur des mini-lots (minibatch, par opposition à batch) de taille ajustable par un hyper-paramètre K. Dans le cas de mini-lots, on ne manipule pas un unique vecteur d'entrée, mais plutôt un lot de vecteurs d'entrée, groupés dans une matrice (qui donneront de même une matrice au niveau de la couche cachée, et de la sortie). Dans le cas d'une taille de lot de un, on obtient l'équivalent d'un gradient stochastique. Étant donné que numpy est efficace sur les opérations matricielles, il est possible de faire les calculs efficacement pour un mini-lot au complet. Ceci affecte grandement le temps d'exécution.

Manipulations: on utilisera les problèmes des deux lunes et le problème de classification de chiffres manuscrits MNIST (voir liens sur la page du cours).

1. Dans un premier temps, commencez par une implémentation qui calcule le gradient pour **un** exemple, et vérifiez que le calcul est correct avec la technique de vérification du gradient par différence finie expliquée ci-dessus.

2. Verification du gradient : produire un affichage de verification du gradient par difference finie pour votre réseau (pour un petit réseau, par ex. $d = 2$ et $d_h = 2$ initialisé aléatoirement) sur 1 exemple.

3. Ajoutez à cette version un hyperparamètre de taille de lot K, pour permettre le calcul du gradient par mini-lot de K exemples (présentés sous forme de matrices), en faisant **une boucle** sur les K exemples (c'est une petit ajout à votre code précédent).

4. Verification du gradient : produire un affichage de verification du gradient sur les paramètres, par difference finie pour votre réseau (pour un petit réseau, par ex. $d = 2$ et $d_h = 2$ initialisé aléatoirement) pour un lot de 10 exemples (vous pouvez prendre des exemples des deux classes du jeux de donnée des 2 lunes).

5. Entrainer votre réseau de neurones par descente de gradient sur les données du problème des deux-lunes. Afficher les régions de décision pour differentes valeurs d'hyper-paramètres (weight decay, nombre d'unites cachées, arrêt prématuré) de façon à illustrer leur effet sur le contrôle de capacité.

6. Dans un deuxième temps, faites une copie de votre implémentation en vue d'en faire une version modifiée efficace qui manipulera les lots de K exemples avec des calculs matriciels (plutôt qu'une boucle). **Reprenez les expressions matricielles numpy établies dans la première partie, et adaptez-les au cas des mini-lots de taille K. Indiquez dans votre rapport comment vous les avez adaptées (précisez les anciennes et nouvelles expressions avec les dimensions de chaque matrice).**

7. Comparez vos deux implémentations (avec et sans boucle sur les exemples du lot) pour vérifier qu'elles donnent le même gradient total sur les paramètres, d'abord avec $K = 1$. Puis comparez-les avec $K = 10$. Joignez à votre rapport les affichages numériques effectués pour cette comparaison.

8. Mesurez le temps que prend une époque sur MNIST (1 époque = 1 passage complet à travers l'ensemble d'entraînement) pour $K = 100$ avec chacune des deux implémentation (mini-lot par boucle, et mini-lot avec calcul matriciel).

9. Adaptez votre code pour qu'il calcule au vol, pendant l'entraînement, l'erreur de classification totale sur l'ensemble d'entraînement, en plus du coût optimisé total (somme des L encourus), ceci pour chaque époque d'entraînement, et qu'après chaque époque d'entraînement, il calcule aussi erreur et coût moyen sur l'ensemble de validation et de test. Faites en sorte qu'il les affiche après chaque époque les 6 nombres correspondants (erreur et coût moyen sur train, valid, test) et les écrive dans un fichier.

10. Entrainer votre réseau sur les données de MNIST. Produisez les courbes d'entrainement, de validation et de test (courbes de l'erreur de classification et du coût en fonction du nombre d'époques d'entraînement, qui correspondent à ce que vous avez enregistré dans un fichier à la question précédente). Joignez à votre rapport les courbes obtenues avec votre meilleure valeur d'hyper-paramètres, c.a.d. pour lesquels vous avez atteint la plus basse erreur de classification sur l'ensemble de validation. On suggère deux graphiques: un pour les courbes de taux d'erreurs de classification (train, valid, test avec des couleurs différentes, bien précisées dans la légende) et l'autre pour la perte moyenne (le L moyen sur train, valid , test). Normalement vous devriez pouvoir atteindre moins de 5% d'erreur en test. Indiquez dans votre rapport la valeur des hyper-paramètres retenue et correspondant aux courbes que vous joignez. Points boni pour une erreur de test inférieure à 2%.