IFT 6390, Devoir 3

Adel Nabli, Myriam Laiymani

07 Novembre 2018

1 Relations et dérivées de quelques fonctions de base:

Dans toute la suite, nous utiliserons la notation σ pour la fonction *sigmoid*.

1. $$\frac{1}{2}\big(\tanh(\frac{x}{2})+1\big) = \frac{1}{2}\Big(\frac{e^{x/2}-e^{-x/2}}{e^{x/2}+e^{-x/2}} + \frac{e^{x/2}+e^{-x/2}}{e^{x/2}+e^{-x/2}}\Big) = \frac{1}{2}\frac{2e^{x/2}}{e^{x/2}+e^{-x/2}} = \frac{1}{1+e^{-x}} = \sigma(x)$$

2. $$\ln(\sigma(x)) = \ln(\frac{1}{1+e^{-x}}) = -\ln(1+e^{-x}) = -softplus(-x)$$

3. On va utiliser la formule pour dériver une composée de fonctions dérivables $(f\circ g)' = g'.f'\circ g$

$$\sigma'(x) = -e^{-x}(1+e^{-x})^{-2} = \frac{e^{-x}}{1+e^{-x}}\frac{1}{1+e^{-x}} = \sigma(x)\frac{1-1+e^{-x}}{1+e^{-x}} = \sigma(x)\big(1-\sigma(x)\big)$$

4. On va utiliser la formule pour dériver un produit de fonctions dérivables $(uv)' = u'v + uv'$ sachant que $\tanh(x) = (e^x - e^{-x})(e^x+e^{-x})^{-1}$

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

5. La fonction *sign* vaut 1 quand son argument est strictement positif et -1 dans le cas contraire. Ainsi, on a:
$$sign(x) = \mathbb{1}_{x>0} - \mathbb{1}_{x\leq 0}$$

6. La fonction *abs* n'est usuellement pas dérivable en 0 car sa dérivée à droite est différente de celle à gauche en ce point (cela peut se voir en revenant à la définition de la dérivée utilisant des quantificateurs ϵ et une limite).
Ce problème disparaît si l'on étudie *abs* sous la prisme de la théorie des distributions: *abs* est bien dérivable au sens des distributions (prendre $\Omega = \mathbb{R}$, remarquer que $abs \in \mathbf{L}^1_{loc}(\Omega)$, prendre un $\phi \in \mathcal{D}(\Omega)$ et utiliser la définition de dérivée au sens des distributions).
Nous allons nous contenter ici de la convention donnée par l'énoncé, ce qui donne: $abs'(x) = sign(x)$ (en effet, sur $\mathbb{R}_+^*$, $abs(x) = x$ soit de dérivé égale à 1, sur $\mathbb{R}_-^*$, $abs(x) = -x$ de dérivé -1, et si on ajoute la convention d'une dérivée nulle en 0, cela donne bien la fonction *sign*).

7. On a $rect(x) = x\mathbb{1}_{x>0}$. Cette fonction est dérivable sur $\mathbb{R}^*$, mais en ajoutant la convention $rect'(0) = 0$, on obtient $rect'(x) = \mathbb{1}_{x>0}$.

8. On a, $\forall x \in \mathbb{R}^d,\ ||x||_2^2 = \sum_{i=1}^d x_i^2$. Posons $f:\ x \mapsto ||x||_2^2$. On sait que $\nabla_f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{pmatrix}$.

   Or, $\forall\ i \in [\![1, d]\!],\ \frac{\partial f}{\partial x_i} = 2x_i$. Ainsi, on a $\nabla_f(x) = 2x$.

9. Sachant que $\forall x \in \mathbb{R}^d,\ ||x||_1 = \sum_{i=1}^d |x_i|$, en posant $f:\ x \mapsto ||x||_1$ et sachant que, par la question 6, $\forall\ i \in [\![1, d]\!],\ \frac{\partial f}{\partial x_i} = sign(x_i)$, on a: $\nabla_f(x) = sign(x)$ (en posant la convention que la fonction *sign* retourne sur $\mathbb{R}^d$ un vecteur dans $\mathbb{R}^d$ contenant le signe de chacune des composantes d'entrée).

2 Calcul du gradient pour l'optimisation des paramètres d'un réseau de neurones pour la classification multiclasse:

1. On a $\dim(b^{(1)}) = d_h$. Ainsi, pour une observation $x\ \in \mathbb{R}^d$, $h^a = W^{(1)}x + b^{(1)}$, soit:

$$\forall\ j \in [\![1, d_h]\!],\ h_j^a = b_j^{(1)} + \sum_{l=1}^{d} w_{j,l}^{(1)} x_l$$

   On a alors $h^s = rect(h^a)$.

2. On a $\dim(W^{(2)}) = m \times d_h$, $\dim(b^{(2)}) = m$ et $o^a = W^{(2)}h^s + b^{(2)}$, ce qui donne:

$$\forall\ k \in [\![1, m]\!],\ o_k^a = b_k^{(2)} + \sum_{j=1}^{d_h} w_{k,j}^{(2)} h_j^s$$

3. On sait que $o^s = softmax(o^a)$. Ainsi, on a:

$$\forall\ k \in [\![1, m]\!],\ o_k^s = \frac{e^{o_k^a}}{\sum_{k=1}^m e^{o_k^a}}$$

   ce qui est bien positif $\forall k$ par propriété de la fonction exponentielle et on a bien $\sum_{k=1}^m o_k^s = 1$. On a ainsi en sortie une distribution de probabilité sur $[\![1, m]\!]$.

4. On a $\forall\ y\ \in [\![1, m]\!],\ L(x, y) = -\log(o_y^s)$, ce qui donne, d'après la question précédente:

$$\forall\ y \in [\![1, m]\!],\ L(x, y) = -o_y^a + \log(\sum_{k=1}^{m} e^{o_k^a})$$

5. Par définition du risque empirique, on a:

$$\hat{R} = \frac{1}{n}\sum_{i=1}^{n} -\log\left(o_{y^{(i)}}^s(x^{(i)})\right) = \log(\sum_{k=1}^{m} e^{o_k^a}) - \frac{1}{n}\sum_{i=1}^{n} o_{y^{(i)}}^a$$

   On a $\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$, ce qui donne $n_\theta = d_h \times d + d_h + m \times d_h + m$. Le problème d'optimisation à considérer est alors celui de trouver $\theta^* = \underset{\theta}{\operatorname{argmin}}\ \hat{R}$.

6. La descente de gradient est un algorithme itératif qui calcule pas après pas une meilleure estimation de l'objectif θ^*. La "largeur" de chaque pas est indiqué par un *learning rate* η_t qui peut dépendre de numéro de l'itération t à laquelle on est. La "direction" de la mise à jour est donnée par le gradient $\frac{\partial \hat{R}}{\partial \theta}$. On arrête l'algorithme à partir du moment où l'écart entre deux itérations des valeurs estimées de θ^* dépassent un certain seuil ϵ.
Cela se traduit par le pseudo code suivant:

   - **Définition** du seuil ϵ
   - **Initialisation** aléatoire de θ_0, $ecart = +\infty$
   - **Tant que** $ecart > \epsilon$:

     $$\theta_{t+1} = \theta_t - \eta_t \frac{\partial \hat{R}}{\partial \theta}(\theta_t)$$
     $$ecart = ||\theta_{t+1} - \theta_t||$$
     $$\theta_t = \theta_{t+1}$$
   - **Retourner** θ_t

   Remarque: à la place de se donner un ϵ, on peut se donner un nombre n_{epoch} total d'itérations.

7. En reprenant les résultats de la question 4, en utilisant la formule de dérivée des fonctions composées, on a:
$$\forall k \neq y,\ \frac{\partial L}{\partial o^a_k} = -0 + \frac{e^{o^a_k}}{\sum_{k=1}^m e^{o^a_k}} \quad ; \quad \frac{\partial L}{\partial o^a_y} = -1 + \frac{e^{o^a_y}}{\sum_{k=1}^m e^{o^a_k}}$$
Ainsi, en rassemblant les résultats et en se rappelant de l'expression de o^s_k vu en question 3, on a:
$$\frac{\partial L}{\partial o^a} = -onehot_m(y) + o^s$$

8. `grad_oa = os - np.array([int(i == y) for i in range(m)])`

9. On a $\forall (j,k) \in [\![1, d_h]\!] \times [\![1, m]\!]$:
$$\frac{\partial o^s}{\partial w^{(2)}_{k,j}} = \frac{\partial o^s}{\partial o^a_k}\frac{\partial o^a_k}{\partial w^{(2)}_{k,j}} = \Big(\frac{e^{o^a_k}}{\sum_{k=1}^m e^{o^a_k}} - \frac{(e^{o^a_k})^2}{\big(\sum_{k=1}^m e^{o^a_k}\big)^2}\Big) h^s_j = (o^s_k - (o^s_k)^2).h^s_j$$
$$\frac{\partial o^s}{\partial b^{(2)}_k} = \frac{\partial o^s}{\partial o^a_k}\frac{\partial o^a_k}{\partial b^{(2)}_k} = \Big(\frac{e^{o^a_k}}{\sum_{k=1}^m e^{o^a_k}} - \frac{(e^{o^a_k})^2}{\big(\sum_{k=1}^m e^{o^a_k}\big)^2}\Big).1 = o^s_k - (o^s_k)^2$$

10. En reprenant les expressions précédentes, on a:
$$\frac{\partial o^s}{\partial W^{(2)}} = \underbrace{(o^s_k - (o^s_k)^2)(h^s)^T}_{\in \mathbb{R}^{m \times d_h}} \quad ; \quad \frac{\partial o^s}{\partial b^{(2)}} = \underbrace{o^s - (o^s)^2}_{\in \mathbb{R}^m} \quad ; \quad o^s \in \mathbb{R}^m,\ h^s \in \mathbb{R}^{d_h}$$
Ce qui donne en python:

`grad_b2 = os - os**2` et `grad_W2 = np.dot(grad_b2, hs.T)`

Note: Cependant, ce qui est utile comme valeur dans la backpropagation n'est pas ce qui vient d'être calculé et était demandé par l'énoncé (la dérivée de la couche de sortie par rapport aux paramètres) mais bien la dérivée de la fonction de coût par rapport aux paramètres. Avec un calcul similaire à celui effectué ici, on trouve donc:
$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial o^a}(h^s)^T \in \mathbb{R}^{m \times d_h} \quad ; \quad \frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial o^a} \quad ; \quad \frac{\partial L}{\partial o^a} \in \mathbb{R}^m,\ h^s \in \mathbb{R}^{d_h}$$

Dans notre *Notebook*, nous utiliserons plutôt les notations `grad_b2` et `grad_W2` pour ces deux dérivées avec:

`grad_W2 = np.dot(grad_oa, hs.T)` et `grad_b2 = grad_oa`

11. On a:

$$\forall\, j \in [\![1, d_h]\!],\ \frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{m} \frac{\partial L}{\partial o_k^a} \frac{\partial o_k^a}{\partial h_j^s}$$

Or, nous avons déjà calculé $\frac{\partial L}{\partial o_k^a}$ à la q.7, et avons, en reprenant l'expression donné q.2, $\frac{\partial o_k^a}{\partial h_j^s} = w_{k,j}^{(2)}$.
Ce qui donne:

$$\forall\, j \in [\![1, d_h]\!],\ \frac{\partial L}{\partial h_j^s} = \sum_{k=1}^{m} (-\mathbb{1}_{k=y} + o_k^s) w_{k,j}^{(2)} = -w_{y,j}^{(2)} + \sum_{k=1}^{m} o_k^s w_{k,j}^{(2)}$$

12. On a donc, en version matricielle:

$$\frac{\partial L}{\partial h^s} = \underbrace{W^{(2)T}}_{\in \mathbb{R}^{d_h \times m}} \underbrace{(o^s - onehot_m(y))}_{= \frac{\partial L}{\partial o^a} \in \mathbb{R}^m} \in \mathbb{R}^{d_h}$$

Ce qui donne, en python:

`grad_hs = W2.T.dot(grad_oa)`

13. On a:

$$\forall\, j \in [\![1, d_h]\!],\ \frac{\partial L}{\partial h_j^a} = \frac{\partial L}{\partial h_j^s} \frac{\partial h_j^s}{\partial h_j^a}$$

Or, nous venons de calculer $\frac{\partial L}{\partial h_j^s}$ à la question précédente et nous savons que $h^s = rect(h^a)$. Ainsi, en utilisant le résultat de la question 1.7, cela donne:

$$\forall\, j \in [\![1, d_h]\!],\ \frac{\partial L}{\partial h_j^a} = \Big(-w_{y,j}^{(2)} + \sum_{k=1}^{m} o_k^s w_{k,j}^{(2)} \Big) \mathbb{1}_{h_j^a > 0}$$

14. En version matricielle, en notant $*$ l'opérateur de multiplication termes à termes et $\forall\, v \in \mathbb{R}^{d_h},\ \mathbb{1}_{x>0}(v)$ le vecteur de $\{0,1\}^{d_h}$ dont la j^{eme} composantes vaut 1 si $v_j > 0$ et 0 sinon, cela donne:

$$\frac{\partial L}{\partial h^a} = \underbrace{\frac{\partial L}{\partial h^s}}_{\in \mathbb{R}^{d_h}} * \underbrace{\mathbb{1}_{x>0}(h^a)}_{\in \mathbb{R}^{d_h}} \in \mathbb{R}^{d_h}$$

On peut donc écrire cela en python de la façons suivante:

`grad_ha = grad_hs*(ha > 0)`

15. On a, au vu des calculs fait q.1 et q.13, $\forall (l, j) \in [\![1, d]\!] \times [\![1, d_h]\!]$:

$$\frac{\partial h^s}{\partial w_{j,l}^{(1)}} = \frac{\partial h^s}{\partial h_j^a} \frac{\partial h_j^a}{\partial w_{j,l}^{(1)}} = \mathbb{1}_{h_j^a > 0}.x_l$$

$$\frac{\partial h^s}{\partial b_j^{(1)}} = \frac{\partial h^s}{\partial h_j^a} \frac{\partial h_j^a}{\partial b_j^{(1)}} = \mathbb{1}_{h_j^a > 0}.1$$

16. Cela donne donc, en version matricielle et en rappelant que $\mathbb{1}_{x>0}(h^a) \in \mathbb{R}^{d_h}$:

$$\frac{\partial h^s}{\partial W^{(1)}} = \mathbb{1}_{x>0}(h^a)(\underbrace{x}_{\in \mathbb{R}^d})^T \in \mathbb{R}^{d_h \times d} \quad ; \quad \frac{\partial h^s}{\partial b^{(1)}} = \mathbb{1}_{x>0}(h^a) \in \mathbb{R}^{d_h}$$

Cela donne en python:

```
grad_b1 = np.maximum(0, ha)
grad_W1 = np.dot(grad_b1, x.T)
```

Note: De la même manière que précédemment, ce qui est utile comme valeur dans la backpropagation n'est pas ce qui vient d'être calculé et était demandé par l'énoncé(la dérivée de la couche cachée par rapport aux paramètres) mais bien la dérivée de la fonction de coût par rapport aux paramètres. Avec un calcul similaire à celui effectué ici, on trouve donc:

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial h^a} x^T \in \mathbb{R}^{d_h \times d} \quad ; \quad \frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial h^a} \quad ; \quad \frac{\partial L}{\partial h^a} \in \mathbb{R}^{d_h},\ x \in \mathbb{R}^d$$

Dans notre *Notebook*, nous utiliserons plutôt les notations `grad_b1` et `grad_W1` pour ces deux dérivées avec:

`grad_W1 = np.dot(grad_ha, x.T)` et `grad_b1 = grad_ha`

17. On a, par règle de la chaîne:

$$\forall\, l \in [\![1, d]\!],\ \frac{\partial L}{\partial x_l} = \sum_{j=1}^{d_h} \frac{\partial L}{\partial h_j^a} \frac{\partial h_j^a}{\partial x_l} = \sum_{j=1}^{d_h} \underbrace{\frac{\partial L}{\partial h_j^a}}_{cf\ q.13} w_{j,l}^{(1)}$$

Soit, en version matricielle:

$$\frac{\partial L}{\partial x} = W^{(1)T} \underbrace{\frac{\partial L}{\partial h^a}}_{\in \mathbb{R}^{d_h}} \in \mathbb{R}^d$$

Et en python:

```
grad_x = W1.T.dot(grad_ha)
```

18. On a donc ici $\tilde{R} = \hat{R} + \mathcal{L}(\theta)$. Ainsi, cela donne:

$$\tilde{R} = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}) + \mathcal{L}(\theta)$$

Donc, pour avoir accès à $\frac{\partial \tilde{R}}{\partial \theta}$ il suffit d'ajouter $\frac{\partial \mathcal{L}(\theta)}{\partial \theta}$ à la somme des $\frac{\partial L}{\partial \theta}$ déjà calculés. Cela se calcule en utilisant les résultats des q.1.8 et q.1.9. On a alors, $\forall\, \theta_p \in \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}\}$:

$$\frac{\partial \tilde{R}}{\partial \theta_p} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(x^{(i)}, y^{(i)})}{\partial \theta_p} + \Big(\lambda_{11} sign(W^{(1)}) + 2\lambda_{12} W^{(1)}\Big) \mathbb{1}_{\theta_p = W^{(1)}} + \Big(\lambda_{21} sign(W^{(2)}) + 2\lambda_{22} W^{(2)}\Big) \mathbb{1}_{\theta_p = W^{(2)}}$$

Cela se répercute dans la "backpropagation" en ajoutant le terme de régularisation qui convient aux différents gradients.

3 Implémentation du réseau de neurones et expérimentation

1. Voir *Notebook*

2. Sur un exemple, voilà ce que retourne notre algorithme:

```
The true gradient for the parameter W1 :
[[ 0.29787656 -0.28182156] [-0.18937933 0.17917213]]
The estimated gradient for the parameter W1 :
[[ 0.297877 -0.28182117] [-0.18937916 0.17917229]]
The ratio between the two grads for the parameter W1 :
[[1.00000146 0.99999862] [0.99999907 1.00000088]]
The true gradient for the parameter b1 :
[[ 0.22068429] [-0.14030323]]
The estimated gradient for the parameter b1 :
[[ 0.22068453] [-0.14030313]]
The ratio between the two grads for the parameter b1 :
[[1.00000108] [0.99999931]]
The true gradient for the parameter b2 :
[[ 0.50498663] [-0.50498663]]
The estimated gradient for the parameter b2 :
[[ 0.50498788] [-0.50498538]]
The ratio between the two grads for the parameter b2 :
[[1.00000248] [0.99999752]]
The true gradient for the parameter W2 :
[[ 0.06220098 0.06158102] [-0.06220098 -0.06158102]]
The estimated gradient for the parameter W2 :
[[ 0.062201 0.06158104] [-0.06220096 -0.061581 ]]
The ratio between the two grads for the parameter W2 :
[[1.0000003 1.0000003] [0.9999997 0.9999997]]
```

   Il semble donc bien que l'on ait réussi à implémenter un calcul de gradient qui fonctionne.

3. Voir *Notebook*

4. Sur $K = 10$ exemples, voilà ce que retourne notre algorithme:

```
The true gradient for the parameter W1 :
[[-0.01582565 0.0160922 ] [ 0.01370027 0.0649131 ]]
The estimated gradient for the parameter W1 :
[[-0.01582553 0.01609226] [ 0.01370063 0.06491337]]
The ratio between the two grads for the parameter W1 :
[[0.99999217 1.00000374] [1.00002627 1.00000421]]
The true gradient for the parameter b1 :
[[0.00110748] [0.01495923]]
The estimated gradient for the parameter b1 :
[[0.00110755] [0.01495942]]
```

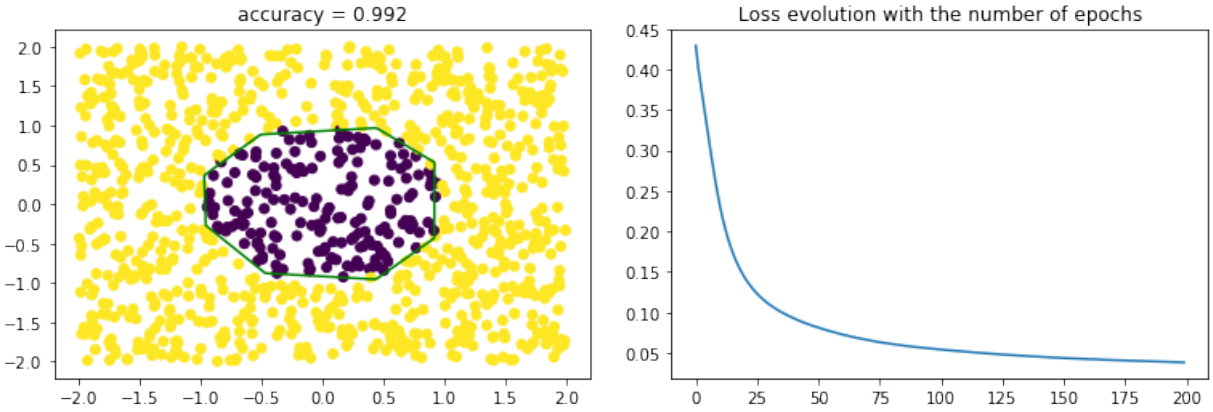
```
The ratio between the two grads for the parameter b1 :
[[1.00006481] [1.00001263]]

The true gradient for the parameter b2 :
[[ 0.04078587] [-0.04078587]]

The estimated gradient for the parameter b2 :
[[ 0.04078709] [-0.04078464]]

The ratio between the two grads for the parameter b2 :
[[1.00003003] [0.99996997]]

The true gradient for the parameter W2 :
[[ 0.06534015 0.00279866] [-0.06534015 -0.00279866]]

The estimated gradient for the parameter W2 :
[[ 0.06534097 0.00279868] [-0.06533934 -0.00279863]]

The ratio between the two grads for the parameter W2 :
[[1.00001243 1.00001034] [0.99998757 0.99998966]]
```

De nouveau, cela nous confirme que nous avons bien calculé les gradients.

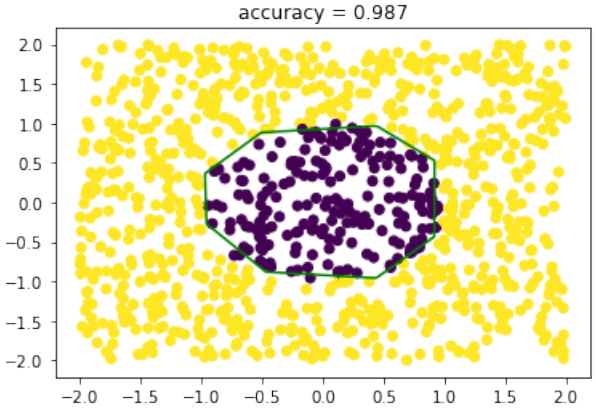
5. • Sans régularisation, pour `dh=4, K=10, n_epoch=200, learning_rate=0.03`, voilà les résultats obtenus sur le dataset *cercle*:

```
Duration of the training = 10.84952712059021 seconds
Mean duration of an epoch = 0.05424763560295105 seconds
```

Training data



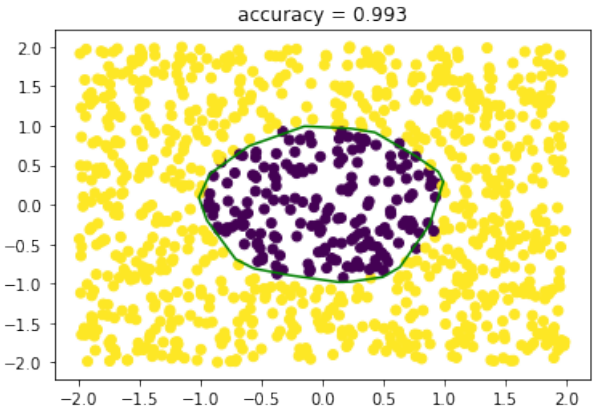
Test data

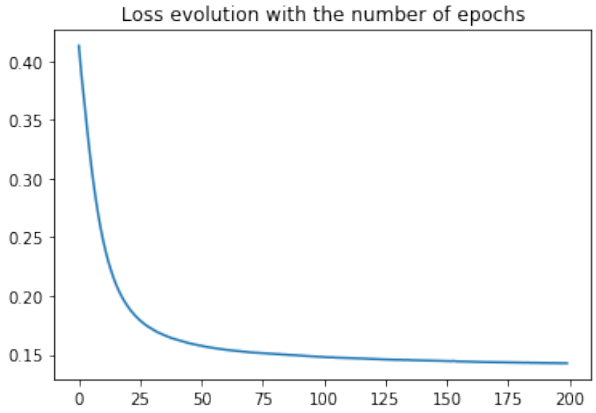


- Avec régularisation, pour `dh=10, K=10, n_epoch=200, learning_rate=0.03, Lambda=[1e-3]*4`, voilà les résultats obtenus sur le dataset *cercle*:

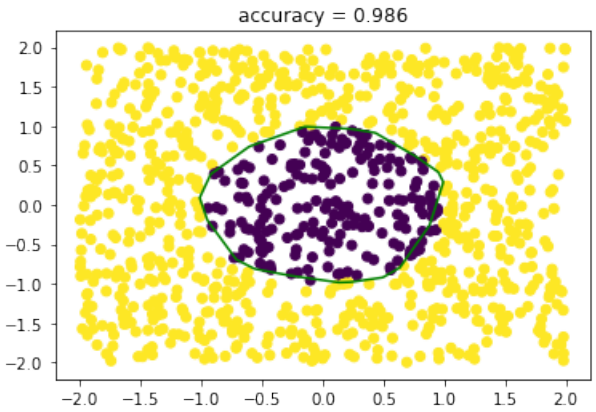
```
Duration of the training = 12.038864850997925 seconds
Mean duration of an epoch = 0.06019432425498963 seconds
```

Training data





Test data



6. Ce qui change avec l'implémentation matricielle est la *backpropagation*. La *forward propagation* est, elle, inchangée. Ainsi, ce que l'on remarque, c'est qu'en version matricielle, les valeurs h_a, h_s, o_a, o_s ne sont plus des vecteurs mais bien des matrices de K colonnes (le nombre de ligne reste inchangé dans notre implémentation). Il faut donc adapter notre code en conséquence, et notamment, nous ne pouvons plus confondre $\frac{\partial L}{\partial b^{(2)}}$ avec $\frac{\partial L}{\partial o^a}$ et $\frac{\partial L}{\partial b^{(1)}}$ avec $\frac{\partial L}{\partial h^a}$ comme on pouvait le faire avant. Ainsi, la *back propagation* passe des `grad` définis dans la partie précédente à:

```
n = len(Y)
grad_oa = self.os.copy() # dim = m × n
grad_oa[Y, range(n)] -= 1
grad_b2 = 1/n * np.sum(grad_oa, axis=1, keepdims=True) # dim = m
grad_W2 = 1/n * np.dot(grad_oa, self.hs.T) # dim = m × d_h
grad_hs = self.W2.T.dot(grad_oa) # dim = d_h × n
grad_ha = grad_hs * (self.ha>0) # dim = d_h × n
grad_b1 = 1/n * np.sum(grad_ha, axis=1, keepdims=True) # dim = d_h
grad_W1 = 1/n * np.dot(grad_ha, X.T) # dim = d_h × d
```

7. Pour comparer les gradients il faut commencer par initialiser les paramètres avec les même valeur, à savoir utiliser la commande `random.seed`.

8. Nous avons fixé les hyperparamètres aux valeurs suivantes: `dh=10, K=100, learning_rate=0.03, use_regularization=True, Lambda=[1e-3]*4`

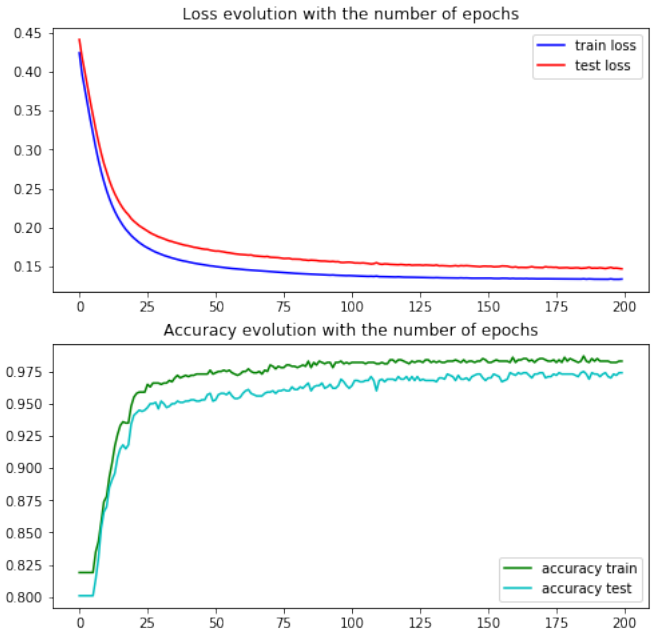
   - 1 epoch avec la méthode avec boucle prend `5.395621681213379 seconds` sur MNIST.
   - 1 epoch avec la méthode matricicielle prend `0.7701480746269226 seconds` sur MNIST.

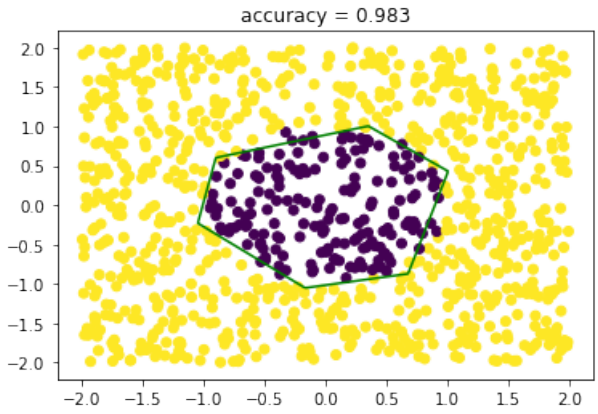
9. Voir *Notebook*

10. Nous n'avons pas réussi à trouver de "bons" hyperparamètres pour l'entraînement sur MNIST. Cependant, voici nos résultats sur le dataset *cercle*:

```
model = NN(2, 4, 2, Lambda=[1e-3]*4)
model.gradient_descent(data_train, data_test, K=10, n_epoch=200, learning_rate=0.03,
use_regularization=True)
```

Training data





Test data