Compte rendu de deep learning
TP 4 - 5

Elodie Difonzo
Roqyun Ko

L'objectif de ce TP est de mettre en place un réseau de neurones simple pour en comprendre le fonctionnement et la façon de l'entraîner.
Nous débuterons par une partie mathématique puis nous implémenterons le réseau avec la librairie PyTorch.

Partie 1 : Formalisation mathématique

1. À quoi servent les ensembles d'apprentissage, de validation et de test ?

- L'ensemble d'apprentissage
  - On calcule les paramètres (les poids et les biais) qui minimisent le résultat de fonction de coût à partir de cet ensemble. Ces données permettent à la machine d' « apprendre ».
- L'ensemble de validation
  - On ré-applique le calcul pondéré avec les poids obtenus lors de l'apprentissage. Puisque les vérités terrains sont données, on peut vérifier la validité du modèle à partir du résultat. Ainsi, si jamais la machine sur-apprend l'ensemble d'apprentissage, on modifie les hyper-paramètres comme le nombre de couches, le taux d'apprentissage, etc pour mieux entraîner la machine.
- L'ensemble de test
  - On évalue la performance du modèle à partir de ces données. On exclut cet ensemble de l'apprentissage. Ainsi, on peut tester la validité du modèle devant les données sur lesquelles il n'est pas entraîné.

2. Quelle est l'influence du nombre N d'exemples ?

La performance du modèle dépend largement de N. Plus il est grand, mieux c'est.

3. Pourquoi est-il important d'ajouter des fonctions d'activation entre des transformations linéaires ?

Un neurone est soit activé soit non-activé. Le modèle juge si l'information est importante ou triviale avec la sortie et c'est une caractéristique importante du réseau de neurones.

Donc, la valeur obtenue à partir d'un neurone a une plage limitée comme [0, 1] ou entre [-1, 1]. La fonction d'activation permet de rendre les transformations non-linéaires pour limiter la plage et appliquer plusieurs transformations de suite.

$$\text{La transformation affine : } \tilde{h} = W \cdot x + b$$
$$\text{La fonction d'activation appliquée : } h = f_{activ}(\tilde{h}) = f_{activ}(W \cdot x + b)$$

4. Quelles sont les tailles n_x, n_h, n_y sur la figure 1 ? En pratique, comment ces tailles sont-elles choisies ?

- n_x
  - La taille d'entrées : Dans notre cas, chaque entrée prend un pixel d'une image. Le nombre d'entrée est fixe et vaut $hauteur \times largeur$.

- n_y
    - La taille de sorties : Ce nombre correspond normalement au nombre de classes (une seule sortie peut être suffisante pour une classification binaire)
- n_h
    - Le nombre de neurones dans une couche : C'est un hyper-paramètre à régler selon le résultat d'un modèle.

5. Que représentent les vecteurs $\hat{y}$ et y ? Quelle est la différence entre ces deux quantités ?

$\hat{y}$ est une prédiction et y est une vérité terrain.
La différence entre ces deux quantités est exprimée en utilisant une fonction de coût.

6. Pourquoi utiliser une fonction SoftMax en sortie ?

La fonction Softmax impose que les sorties de chaque neurone soient comprises entre 0 et 1. Ainsi elles peuvent être interprétées comme des probabilités. Elle nous permet de quantifier la magnitude d'activation de neurones de manière équitable.

7. Écrire les équations mathématiques permettant d'effectuer la passe forward du réseau de neurones, c'est-à-dire permettant de produire successivement $\tilde{h}$,h, $\tilde{y}$ et $\hat{y}$ à partir de x.

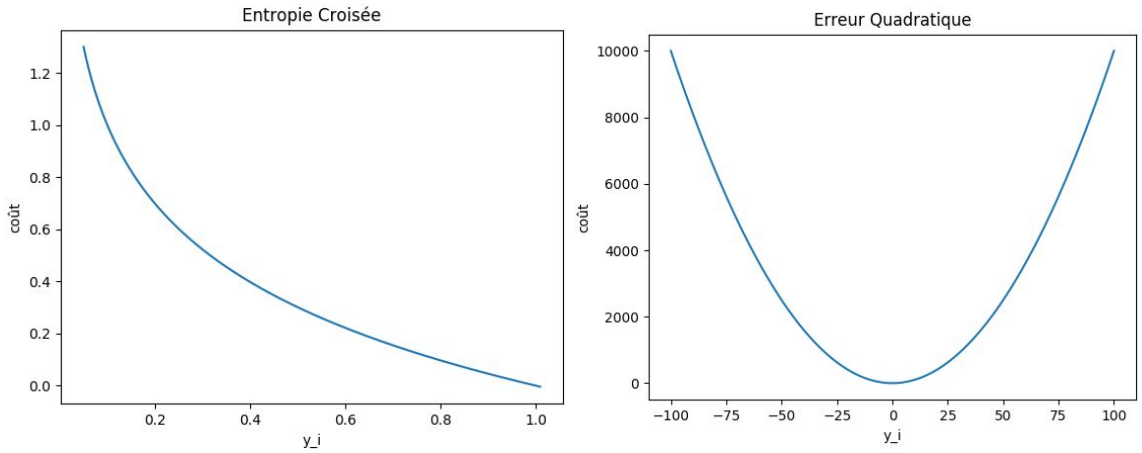
$$\tilde{h_i} = \sum_{j=1}^{n_x} W_{h_{ij}} \cdot x_j + b_{h_i} \quad où \quad i \in \{1, \ldots, n_h\}$$

$$h_j = f_{activ}(\tilde{h}_j) = tanh(\tilde{h}_j) = tanh(\sum_{m=1}^{n_x} W_{x_{m,j}} \cdot x_m + b_j)$$

$$\tilde{y}_l = \sum_{k=1}^{n_h} W_{y_{lk}} \cdot h_k + b_{y_l} \quad où \quad l \in \{1, \ldots, n_y\}$$

$$\hat{y}_l = \frac{e^{\tilde{y}_l}}{\sum_{m=1}^{n_y} e^{\tilde{y}_m}}$$

8. Pendant l'apprentissage, on cherche à minimiser la fonction de coût. Pour l'entropie croisée et l'erreur quadratique, comment les $\hat{y}_i$ doivent-ils varier pour faire diminuer la loss ?



- *Entropie croisée*
  - $l(y, \hat{y}) = -\sum y_i log(\hat{y}_i)$
  - Lorsque la valeur de $\hat{y}_i$ s'approche de 1, le coût devient minimal.
- *Erreur Quadratique*
  - $l_i(y_i, \hat{y}_i) = \sum(-y_i - \hat{y}_i)^2$
  - Lorsque la valeur de $\hat{y}_i$ s'approche de y (vérité terrain), le coût devient minimal.

9. En quoi ces fonctions sont-elles plus adaptées aux problèmes de classification ou de régression ?

L'entropie croisée est mieux adaptée pour les problèmes de classification. Au-delà une valeur de 1 pour $\hat{y}_i$, le coût tend vers l'infini négatif et un coût négatif n'a pas de signification réelle.
On ne peut jamais atteindre un minimum. Donc, l'entropie croisée est adaptée aux problèmes de classification qui utilisent « one-hot encoding ».

En revanche, l'erreur quadratique nous permet de quantifier l'écart entre la vérité terrain y et la prédiction $\hat{y}$. Le résultat de cette fonction de coût devient de plus en plus petite lorsque $\hat{y}$ est près de la vérité terrain y. Alors l'erreur quadratique est mieux adaptée pour les problèmes de régression.

10. Quels semblent être les avantages et inconvénients des diverses variantes de descente de gradient entre les versions classique, stochastique sur mini-batch et stochastique online ? Laquelle semble la plus raisonnable à utiliser dans le cas général ?

La descente de gradient est l'algorithme pour trouver le minimum de fonction de coût.

La version stochastique online calcule le gradient à chaque entraînement (changement de poids). Le calcul est plus rapide et le gradient est plus fréquemment mis à jour. La mise à jour fréquente peut rendre l'optimisation « noisy ». C'est-à-dire, le gradient ne tend pas forcément vers un minimum ou un seul minimum. ***La version stochastique online*** a moins de risque d'obtenir un minimum local grâce à ce comportement. Alors que la convergence se fait plus vite grâce à la rapidité de calcul, ce comportement « noisy » peut la ralentir également.

La version classique calcule le gradient moyen sur l'ensemble de données. Donc, le calcul est souvent intensif. S'il y a trop de données à traiter, le temps du calcul peut être long. Mais on peut vectoriser le calcul pour qu'il soit plus rapide. Le processus de l'optimisation peut avoir une stabilité qui permet d'obtenir une convergence stable grâce au fait de pouvoir prendre une moyenne des données pour le calcul. Mais cela veut dire que c'est plus facile d'obtenir un minimum local au lieu d'un minimum global à l'inverse de la version stochastique online.

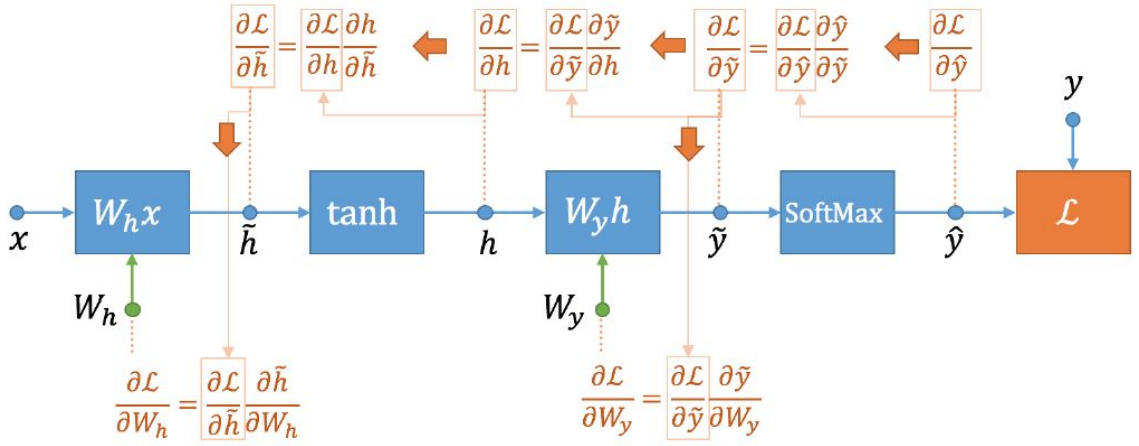
La version stochastique mini-batch est un compromis entre les deux versions. Il prend un sous-ensemble de données pour trouver le minimum. Donc, le calcul est plus stable que ***la version stochastique online*** et plus robuste que ***la version classique*** contre l'erreur d'obtenir un minimum local. Il est plus rapide que ***la version classique*** et plus efficace que ***la version stochastique online*** car on peut vectoriser le calcul.

La version stochastique mini-batch est donc la plus raisonnable d'utiliser dans le cas général.

11.Quelle est l'influence du learning rate η sur l'apprentissage ?

Une grande valeur de η modifie largement les poids lors de l'entraînement. Une petite valeur fait l'inverse. Si cette valeur est trop grande, les poids peuvent dépasser les bonnes valeurs pour obtenir le minimum de la fonction de coût. Si le learning rate est trop petit, les poids vont progresser trop lentement pour obtenir les bonnes valeurs.

12. Comparer la complexité (en fonction du nombre de couches du réseau) du calcul des gradients de la loss par rapport aux paramètres, en utilisant l'approche naı¨ve et l'algorithme de backprop.



La méthode naïve recommence le calcul des gradients depuis la loss *L* (en orange dans la figure ci-dessus).

Donc, ***la méthode naïve :***

1er itération :

1. $\frac{\partial L}{\partial \hat{\mathrm{y}}}$

2ème itération :

- Recalculer $\frac{\partial L}{\partial \hat{\mathrm{y}}}$
- $\frac{\partial L}{\partial \tilde{\mathrm{y}}} = \frac{\partial L}{\partial \hat{\mathrm{y}}}\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}$

3ème itération :

- Recalculer $\frac{\partial L}{\partial \hat{\mathrm{y}}}$ et $\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}$
- $\frac{\partial L}{\partial \mathrm{h}} = \frac{\partial L}{\partial \tilde{\mathrm{y}}}\frac{\partial \tilde{\mathrm{y}}}{\partial h} = \frac{\partial L}{\partial \hat{\mathrm{y}}}\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}\frac{\partial \tilde{\mathrm{y}}}{\partial h}$

4ème itération :

- Recalculer $\frac{\partial L}{\partial \hat{\mathrm{y}}}$ et $\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}$
- $\frac{\partial L}{\partial W_y} = \frac{\partial L}{\partial \tilde{y}}\frac{\partial \tilde{y}}{\partial W_y} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \tilde{y}}{\partial W_y}$

5ème itération:

- Recalculer $\frac{\partial L}{\partial \hat{\mathrm{y}}}$, $\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}$ et $\frac{\partial \tilde{\mathrm{y}}}{\partial h}$

- $\frac{\partial L}{\partial \tilde{h}} = \frac{\partial L}{\partial \mathrm{h}} \frac{\partial h}{\partial \tilde{h}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h} \frac{\partial h}{\partial \tilde{h}}$

6ème itération :

- Recalculer $\frac{\partial L}{\partial \hat{\mathrm{y}}}$, $\frac{\partial \hat{\mathrm{y}}}{\partial \tilde{\mathrm{y}}}$ et $\frac{\partial \tilde{\mathrm{y}}}{\partial h}$
- $\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial \tilde{h}} \frac{\partial \tilde{h}}{\partial W_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial h} \frac{\partial h}{\partial \tilde{h}} \frac{\partial \tilde{h}}{\partial W_h}$

Or, l'algorithme de backpropagation ré-utilise les dérivées calculées (au lieu de recalculer comme l'approche naïve) sans les recalculer et applique la règle de chaîne afin de parcourir les couches 1 seule fois pour obtenir le coût.

13. Quel critère doit respecter l'architecture du réseau pour permettre la backpropagation ?

Pour qu'on puisse utiliser la backpropagation, il faut que les fonctions de coût soient dérivables.

14. La fonction SoftMax et la loss de cross-entropy sont souvent utilisées ensemble et leur gradient est très simple. Montrez que la loss se simplifie en :

$$l = -\sum_i y_i \tilde{y}_i + log(\sum_i e^{\tilde{y}_i})$$

1. La formule de ***la fonction de coût*** de cross entropy est :

$$l(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i)$$

2. La formule de ***la fonction d'activation*** softmax est :

$$\hat{y}_i = \frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}$$

3. On assigne ***la fonction d'activation*** dans ***la fonction de coût***:

$$l(y, \tilde{y}) = -\sum_i y_i log(\frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}})$$

4. On applique la règle de quotient de logarithme:

$$l(y, \tilde{y}) = -\sum_i y_i \left(log(e^{\tilde{y}_i}) - log(\sum_j e^{\tilde{y}_j}) \right)$$

5. Le logarithme naturel d'une fonction exponentielle donne l'exposant. Donc on peut simplifier la loss en :

$$l(y, \tilde{y}) = -\left(\sum_i y_i \tilde{y}_i + \sum_i y_i \cdot log(\sum_j e^{\tilde{y}_j}) \right)$$

La somme de y_i est égale à 1 (one hot encoding)

donc,

$$l(y, \tilde{y}) = -\sum_i y_i \tilde{y}_i + log(\sum_j e^{\tilde{y}_j})$$

15. Écrire le gradient de la loss (cross-entropy) par rapport à la sortie intermédiaire

$$l_i(y_i, \tilde{y}_i) = -y_i \tilde{y}_i + log(\sum_j e^{\tilde{y}_j})$$

$$\frac{\partial l}{\partial \tilde{y}_i} = -y_i + \frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}$$

Rappelons qu'on peut simplifier $\hat{y}_i = \frac{e^{\tilde{y}_i}}{\sum_j^{n_y} e^{\tilde{y}_m}}$ en :

$$\frac{\partial l}{\partial \tilde{y}_i} = -y_i + \hat{y}_i$$

$$\nabla \tilde{y} l = \begin{bmatrix} \hat{y}_1 - y_1 \\ \vdots \\ \hat{y}_{n_y} - y_{n_y} \end{bmatrix}$$

16. En utilisant la backpropagation, écrire le gradient de la loss par rapport aux poids de la couche de sortie $\nabla W_y l$. Notez que l'écriture de ce gradient utilise $\nabla \tilde{y} l$. Faire de même pour $\nabla b_y l$.

$$\nabla W_y l = \begin{bmatrix} (\hat{y}_1 - y_1)\, h_1 & \cdots & (\hat{y}_1 - y_1)\, h_{n_h} \\ \vdots & \ddots & \vdots \\ (\hat{y}_{n_y} - y_{n_y})\, h_1 & \cdots & (\hat{y}_{n_y} - y_{n_y})\, h_{n_h} \end{bmatrix}$$

et

$$\nabla b_y l = \begin{bmatrix} (\hat{y}_1 - y_1) \\ \vdots \\ (\hat{y}_{n_y} - y_{n_y}) \end{bmatrix}$$

car

$$\frac{\partial l}{\partial W_{y\,i,j}} = (\hat{y}_i - y_i)\, h_j$$

et

$$\frac{\partial l}{\partial b_y} = \frac{\partial l}{\partial \tilde{\mathrm{y}}} \cdot 1$$

Pour le calcul, on utilise la règle de chaîne pour trouver les *loss*.

Pour $\nabla W_y l$:

$$\frac{\partial l}{\partial W_y} = \frac{\partial l}{\partial \tilde{\mathrm{y}}} \frac{\partial \tilde{y}}{\partial W_y}$$

$$\tilde{y} = \sum_i^{n_y} \sum_j^{n_h} \left(W_{y_{ij}} \cdot h_j + b_{y_i} \right)$$

$$\frac{\partial \tilde{y}}{\partial W_{y_{i,j}}} = \frac{\partial}{\partial W_{y_{i,j}}} \left(W_{1,1} \cdot h_1 + \ldots + W_{1,n_h} \cdot h_{n_h} + \ldots + W_{n_y,1} \cdot h_1 + \ldots + W_{n_y,n_h} \cdot h_{n_h} + \sum_i b_{y_i} \right)$$

Donc :

$$\frac{\partial \tilde{y}}{\partial W_{y_{i,j}}} = h_j, \forall i$$

Enfin :

$$\frac{\partial l}{\partial W_{y\,i,j}} = (\hat{y}_i - y_i)\, h_j$$

$$\nabla W_y l = \begin{bmatrix} (\hat{y}_1 - y_1)\, h_1 & \cdots & (\hat{y}_1 - y_1)\, h_{n_h} \\ \vdots & \ddots & \vdots \\ \left(\hat{y}_{n_y} - y_{n_y}\right) h_1 & \cdots & \left(\hat{y}_{n_y} - y_{n_y}\right) h_{n_h} \end{bmatrix}$$

Pour $\nabla b_y l$:

$$\frac{\partial l}{\partial b_y} = \frac{\partial l}{\partial \tilde{\mathrm{y}}} \frac{\partial \tilde{y}}{\partial b_y}$$

$$\frac{\partial \tilde{y}}{\partial b_{y_i}} = \frac{\partial}{\partial b_{y_i}} \left(\sum_j^{n_h} \left(W_{y_{ij}} \cdot h_j\right) + b_{y_1} + \ldots + b_{y_{n_y}} \right)$$

$$\frac{\partial \tilde{y}}{\partial b_{y_i}} = \frac{\partial}{\partial b_{y_i}} \left(b_{y_i}\right) = 1, \forall i$$

Donc :

$$\frac{\partial l}{\partial b_y} = \frac{\partial l}{\partial \tilde{\mathrm{y}}} \cdot 1$$

Enfin :

$$\nabla b_y l = \begin{bmatrix} (\hat{y}_1 - y_1) \\ \vdots \\ \left(\hat{y}_{n_y} - y_{n_y}\right) \end{bmatrix}$$

17. Calculer les autres gradients : $\nabla\tilde{h}l$, $\nabla W_h l$,$\nabla b_h l$.

$$\nabla\tilde{h}l = \begin{bmatrix} \left(1-tanh^2(\tilde{h_1})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,1}} \\ \vdots \\ \left(1-tanh^2(\tilde{h_j})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,n_h}} \end{bmatrix},$$

$$\nabla W_h l = \begin{bmatrix} \left(1-tanh^2(\tilde{h_1})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,1}}x_1 & \cdots & \left(1-tanh^2(\tilde{h_{n_h}})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,n_h}}x_1 \\ \vdots & \ddots & \vdots \\ \left(1-tanh^2(\tilde{h_1})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,n_h}}x_{n_x} & \cdots & \left(1-tanh^2(\tilde{h_{n_h}})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,n_h}}x_{n_x} \end{bmatrix}$$

et

$$\nabla b_h l = \begin{bmatrix} \left(1-tanh^2(\tilde{h_1})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,1}} \\ \vdots \\ \left(1-tanh^2(\tilde{h_j})\right)\sum_i(\hat{y}_i - y_i)\,W_{y_{i,n_h}} \end{bmatrix}$$

car

$$\frac{\partial l}{\partial \tilde{h_j}} = \sum_i (\hat{y}_i - y_i)\,W_{i,j}\cdot(1-tanh(\tilde{h_j}))$$

$$\frac{\partial l}{\partial W_{h_{k,j}}} = \sum_i (\hat{y}_i - y_i)\,W_{y_{i,j}}\cdot(1-tanh(\tilde{h_j}))x_k$$

et

$$\frac{\partial l}{\partial b_h} = \frac{\partial l}{\partial \tilde{\mathrm{h}}}\cdot 1$$

Pour le calcul, on utilise la règle de chaîne pour trouver les *loss*.

1. Pour ∇hl :

$$\frac{\partial L}{\partial \mathrm{h}} = \frac{\partial L}{\partial \tilde{\mathrm{y}}}\frac{\partial \tilde{\mathrm{y}}}{\partial h}$$

on calcul d'abord $\dfrac{\partial \tilde{y}}{\partial h_i}$ **:**

$$\frac{\partial \tilde{y}}{\partial h_j} = \frac{\partial}{\partial h_j}\left(W_{y_{1,1}}\cdot h_1 + \ldots + W_{y_{1,n_h}}\cdot h_{n_h} + \ldots + W_{y_{n_y,1}}\cdot h_1 + \ldots + W_{y_{n_y,n_h}}\cdot h_{n_h} + \sum_i b_{y_i}\right)$$

Donc :

$$\frac{\partial \tilde{y}}{\partial h_{y_j}} = W_{i,j}, \forall i$$

$$\frac{\partial l}{\partial h_j} = \sum_i (\hat{y}_i - y_i) W_{y_{i,j}}$$

$$\nabla hl = \begin{bmatrix} \sum_i (\hat{y}_i - y_i) W_{y_{i,1}} \\ \vdots \\ \sum_i (\hat{y}_i - y_i) W_{y_{i,n_h}} \end{bmatrix}$$

2. Pour $\nabla \tilde{h} l$:

$$\frac{\partial L}{\partial \tilde{\mathrm{h}}} = \frac{\partial L}{\partial \mathrm{h}} \frac{\partial h}{\partial \tilde{h}}$$

$$h_j = tanh(\tilde{h}_j) = tanh(\sum_{k=1}^{n_x} W_{h_{kj}} \cdot x_k + b_j)$$

$$\frac{\partial \cdot tanh(\tilde{h_j})}{\partial \tilde{h}} = 1 - tanh^2(\tilde{h_j})$$

Donc :

$$\frac{\partial l}{\partial \tilde{h_j}} = \sum_i (\hat{y}_i - y_i) W_{y_{i,j}} \cdot (1 - tanh^2(\tilde{h_j}))$$

Enfin :

$$\nabla \tilde{h} l = \begin{bmatrix} \left(1 - tanh^2(\tilde{h_1})\right) \sum_i (\hat{y}_i - y_i) W_{y_{i,1}} \\ \vdots \\ \left(1 - tanh^2(\tilde{h_j})\right) \sum_i (\hat{y}_i - y_i) W_{y_{i,n_h}} \end{bmatrix}$$

3. Pour $\nabla W_h l$:

$$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial \tilde{h}} \frac{\partial \tilde{h}}{\partial W_h}$$

$$\tilde{h}_j = \sum_k W_{h_{kj}} \cdot x_k + b_{h_j}$$

$$\frac{\partial \tilde{h}}{\partial W_{h_{k,j}}} = \frac{\partial}{\partial W_{h_{k,j}}} \left(Wh_{1,1} \cdot x_1 + \ldots + W_{h_{1,n_h}} \cdot x_1 + \ldots + W_{h_{n_y,1}} \cdot x_{n_x} + \ldots + W_{h_{n_x,n_h}} \cdot x_{n_x} + \sum_k b_{h_k} \right)$$

Donc :

$$\frac{\partial \tilde{h}}{\partial W_{h_{k,j}}} = x_k$$

Enfin :

$$\frac{\partial l}{\partial W_{h_{k,j}}} = \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,j}} \cdot (1 - tanh(\tilde{h_j}))x_k$$

$$\nabla W_h l = \begin{bmatrix} \left(1 - tanh^2(\tilde{h_1})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,1}} x_1 & \cdots & \left(1 - tanh^2(\tilde{h_{n_h}})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,n_h}} x_1 \\ \vdots & \ddots & \vdots \\ \left(1 - tanh^2(\tilde{h_1})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,n_h}} x_{n_x} & \cdots & \left(1 - tanh^2(\tilde{h_{n_h}})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,n_h}} x_{n_x} \end{bmatrix}$$

4. Pour $\nabla b_h l$:

$$\frac{\partial l}{\partial b_h} = \frac{\partial l}{\partial \tilde{\mathrm{h}}} \frac{\partial \tilde{h}}{\partial b_h}$$

$$\frac{\partial \tilde{h}}{\partial b_{h_j}} = \frac{\partial}{\partial b_{h_j}} \left(\sum_k^{n_x} \left(W_{h_{kj}} \cdot x_k\right) + b_{h_1} + \ldots + b_{h_{n_x}} \right)$$

$$\frac{\partial \tilde{h}}{\partial b_{h_j}} = \frac{\partial}{\partial b_{h_j}} \left(b_{h_j}\right) = 1, \forall i$$

Donc :

$$\frac{\partial l}{\partial b_h} = \frac{\partial l}{\partial \tilde{\mathrm{h}}} \cdot 1$$

Enfin :

$$\nabla b_h l = \begin{bmatrix} \left(1 - tanh^2(\tilde{h_1})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,1}} \\ \vdots \\ \left(1 - tanh^2(\tilde{h_j})\right) \sum_i \left(\hat{y}_i - y_i\right) W_{y_{i,n_h}} \end{bmatrix}$$

Partie 2 : Implémentation

```
inconnu@inconnu-900X3M:/media/inconnu/Data/VisualRecognition/TP4-5$ ./circles.py
==== epoch 0 ====
Loss : 75.17329406738281 / Acc 100.0
Val Loss : nan / Val Acc 64.0
==== epoch 1 ====
Loss : nan / Acc 70.0
Val Loss : nan / Val Acc 36.0
==== epoch 2 ====
Loss : nan / Acc 30.0
Val Loss : nan / Val Acc 36.0
==== epoch 3 ====
Loss : nan / Acc 30.0
Val Loss : nan / Val Acc 0.0
==== epoch 4 ====
Loss : nan / Acc 0.0
Val Loss : nan / Val Acc 100.0
==== epoch 5 ====
Loss : nan / Acc 100.0
Val Loss : nan / Val Acc 100.0
==== epoch 6 ====
Loss : nan / Acc 100.0
Val Loss : nan / Val Acc 100.0
==== epoch 7 ====
```