Introduction aux réseaux de neurones

Loüet Joseph & Karmim Yannis
Spécialité **DAC**

Table des matières

1 Formalisation mathématique

1.1 Jeu de données

QUESTIONS :

1) L'ensemble d'apprentissage est utilisé pour entraîner notre modèle. Cet ensemble ne doit pas contenir d'élément de l'ensemble de validation ou de l'ensemble de test. L'ensemble de validation est utilisé pour choisir les meilleurs hyper-paramètres sur notre modèle, on choisit les hyper-paramètres qui nous donnent le meilleur score avec l'ensemble de validation. L'ensemble de test est utilisé pour tester notre modèle et prédire les scores obtenues si on applique notre modèle sur un jeu de données inconnues.

2) Plus on a d'exemples, plus on peut corriger sur notre ensemble d'apprentissage. De ces faits, le modèle obtenue sera plus robuste.

1.2 Architecture du réseau (phase forward)

QUESTIONS :

3) Les fonctions permet de transformer nos fonctions linéaires en fonction non-linéaires, en effet, l'utilisation de fonction non-linéaires est indispensable pour résoudre des problèmes complexes.

4)n_x est la taille des données en entrée. n_x est le nombre de neurones de la couche cachée. n_y est la taille des prédictions. Nous ne choisissons pas la taille des données en entrée. En revanche, le nombre de neurones de la couche cachée est un hyper-paramètre à choisir et nous choisissons la taille des prédictions en fonction du problème à résoudre, s'il s'agit d'un problème de classification multi-classes ou d'un problème de température d'une ville.

5)La valeur y représente le label d'un élément x. En revanche, la valeur $\hat{y}$ représente le label prédit par notre modèle. Si f est notre modèle et x est un élément de notre jeu de données, alors y est le label associé à x et $f(x) = \hat{y}$

6) Nous utilisons la fonction *SoftMax* en sortie pour obtenir une distribution de probabilité sur les différentes classes de la prédiction. Cela permet d'en déduire la classe la plus probable.

7) $\widetilde{h_i} = \sum_{j=1}^{n_x} w_{ij}^h x_j + b_i^h =$. Donc, $\widetilde{h} = W_h \star X + B_h$

$h_i = tanh(\widetilde{h_i})$. Donc, $h = tanh(\widetilde{h})$

$\widetilde{y_i} = \sum_{j=1}^{n_y} w_{ij}^y h_j + b_i^y$. Donc, $\widetilde{y} = W_y \star h + B_y$

$\hat{y} = SoftMax(\widetilde{y})$

1.3 Fonction de coût

QUESTIONS :

8) Pour diminuer la loss, il faut que $\hat{y}$ tende vers y

9) Ces deux fonctions sont adaptées aux problèmes et de classification et de régression car ces deux fonctions sont convexes.

1.4 Méthode d'apprentissage

QUESTIONS :

10) L'avantage du batch est une descente de gradient plus stable que le stochastique. L'inconvénient du batch est un temps de calcul supérieur au stochastique. L'avantage du stochastique est un temps de calcul efficace et son inconvénient est un descente de gradient peu stable. Pour cela, la variante de descente de gradient la plus raisonnable à utiliser est le mini-batch.

11) Si le learning rate est trop grand, la descente de gradient ne va pas converger dû a des trop grand "bonds". Si le learning rate est trop petit, la descente de gradient sera longue.

12) L'algorithme de la backpropagation sera moins coûteux en compléxité que l'approche naïve car l'algorithme de la backpropagation s'appuie uniquement des éléments de la couche précédente pour calculer les gradients alors que l'approche naïve va calculer l'ensemble des gradients de toutes les couches pour obtenir le gradient voulu.

13) Pour permettre la *backpropagation*, il faut que l'ensemble des couches et des fonctions d'activation soit linéaire.

14) Pour rappel y_i est en encodage *one-hot* donc une seule composante de ce vecteur est égale à 1. Donc $\sum_k y_k \log(\sum_j e^{\widetilde{y_j}}) = \log(\sum_j e^{\widetilde{y_j}})$

$$\begin{aligned} loss(y, SoftMax(\widetilde{y})) &= -\sum_i y_i \log(\frac{e^{\widetilde{y_i}}}{\sum_j e^{\widetilde{y_j}}}) \\ &= -\sum_i (y_i \widetilde{y_i} - \widetilde{y_i} \log(\sum_j e^{\widetilde{y_j}})) \\ &= -\sum_i y_i \widetilde{y_i} + \sum_k y_k \log(\sum_j e^{\widetilde{y_j}}) \\ &= -\sum_i y_i \widetilde{y_i} + \log(\sum_j e^{\widetilde{y_j}}) \end{aligned}$$

15)

$$\frac{\partial \ell}{\partial \widetilde{y_i}} = -y_i + \frac{e^{\widetilde{y_i}}}{\sum_j e^{\widetilde{y_j}}}$$

16) On a

$$\frac{\partial \widetilde{y}_k}{\partial W_{y,ij}} = h_k$$

Donc,

$$\sum_k \frac{\partial \ell}{\partial \widetilde{y}_k} \frac{\partial \widetilde{y}_k}{\partial W_{y,ij}} = \sum_k (-y_k + \frac{e^{\widetilde{y_k}}}{\sum_j e^{\widetilde{y_j}}}) h_k$$

17) Calcul des autres gradients :

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \tilde{h_i}} &= \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial \tilde{h_i}} \\ &= W_{y,i}(-y_i + \frac{e^{\widetilde{y_i}}}{\sum_j e^{\widetilde{y_j}}})(1 - tan^2(\tilde{h_i})) \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W_{h,i}} &= \frac{\partial \mathcal{L}}{\partial \tilde{h_i}} \frac{\partial \tilde{h_i}}{\partial W_{h,i}} \\ &= W_{y,i}(-y_i + \frac{e^{\widetilde{y_i}}}{\sum_j e^{\widetilde{y_j}}})(1 - tan^2(\tilde{h_i}))X_i \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b_{h,i}} &= \frac{\partial \mathcal{L}}{\partial \tilde{h_i}} \frac{\partial \tilde{h_i}}{\partial b_{h,i}} \\ &= W_{y,i}(-y_i + \frac{e^{\widetilde{y_i}}}{\sum_j e^{\widetilde{y_j}}})(1 - tan^2(\tilde{h_i})) \end{aligned}$$

2 Implémentation

2.1 CirclesData

Le calcul des dérivées de la première section sur l'exemple du modèle donné nous a permis d'implémenter les fonctions demandées afin de calculer la frontière sur les données en cercle et de classifier les deux classes de points. À noter que classifier des données non linéaire est très difficile pour un algorithme de machine learning classique, voire impossible si on utilise pas de Kernel Trick comme dans un SVM. Les réseaux de neurones vont donc nous permettre de résoudre ce problème.

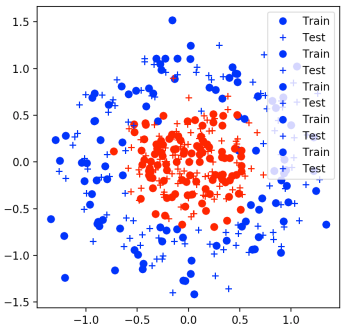
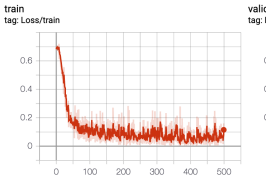
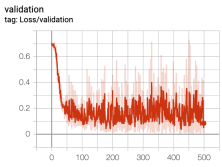


FIGURE 1 – Circles Data

Puisque l'on apprend d'une distribution de probabilité sur deux classes, on utilise une Cross Entropy Loss pour l'apprentissage du réseau. À l'aide de tensorboard nous pouvons visualiser comment évolue nos valeurs de loss en apprentissage et en validation.



(a) Loss en train pour 500 epochs.



(b) Loss en validation pour 500 epochs.

FIGURE 2 – Courbe de loss.

FIGURE 3 – Courbe d'accuracy pour l'exemple sur les Circles Data.

On voit donc que notre réseau apprend correctement à classifier les différents points sur nos données. On obtient un score d'accuracy très proche de 1 même en validation.

2.2 MNIST Data

Le second jeu de données sur lequel on va appliquer notre modèle est le dataset MNIST pour la reconnaissance de nombre écrit à la main.
On a utilisé cette fois directement les modules pytorch pour l'implémentation du modèle. On a ensuite tracé les courbes de Loss et d'accuracy.

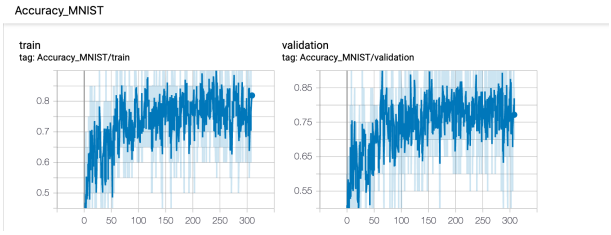


FIGURE 4 – Courbe d'accuracy sur le dataset MNIST.

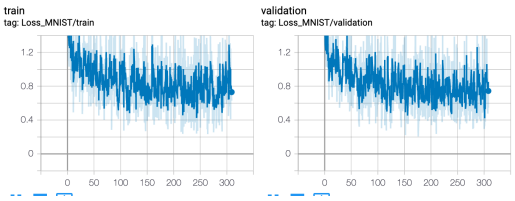


FIGURE 5 – Courbe Loss sur le dataset MNIST.

On remarque que notre modèle a plus de mal à apprendre sur ces données, qui sont ici des images 2D. On verra dans le prochain TP sur les réseaux de convolutions, pourquoi ces derniers sont plus efficace et ont des meilleures résultats