OPT8 : Deep learning
TP2

Adrien Pavao

Decembre 2017

1 Introduction

L'objectif de ce TP est l'implémentation d'un réseau de neurones avec couches cachées. Nous avons vu précédemment un modèle simple, utilisant la fonction **softmax**.

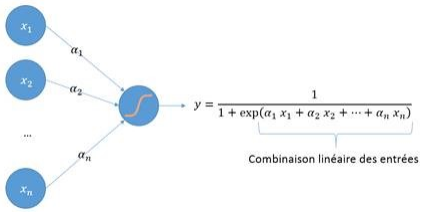

Figure 1: Schéma explicatif de la régression logistique

L'idée générale à présent est d'utiliser plusieurs module de ce type : la sortie du premier devenant l'entrée du second, et ainsi de suite[1].

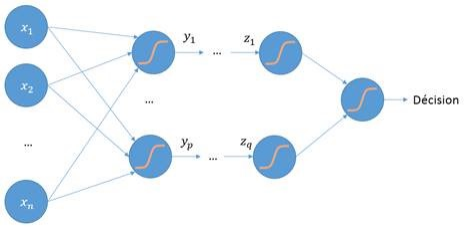

Figure 2: Schéma explicatif du réseau de neurones

Les trois étapes principales qui nous permettront l'entraînement de notre modèle sont les suivantes :

- La forward propagation

[1] Source des images : https://blogs.msdn.microsoft.com/mlfrance/2014/10/15/mthodes-probabilistes-en-machine-learning/

- La backpropagation
- La mise à jour des paramètres.

2 Implémentation

2.1 Code

Forward propagation

La fonction permettant de passer de l'input à l'ouput, avec *act_fun* la fonction d'activation :

```
    Y, Yp = [X.T], []

    for i in range(len(W)):

        y, yp = act_func(np.dot(W[i], Y[i]) + B[i])
        Y.append(y)
        Yp.append(yp)

    return Y, Yp
```

Backward propagation

La fonction permettant l'apprentissage, avec *gradB* le gradient :

```
gradB = [Yp[-1] * error]

    for i in range(len(W) - 1, 0, -1):

        gradB.append(Yp[i - 1] * np.dot(W[i].T, gradB[len(W) - 1 - i]))

    return list(reversed(gradB))
```

Mise à jour des paramètres

Et la mise à jour des paramètres *theta* avec leur variation *dtheta* et *eta* le pas de la descente de gradient :

```
 if regularizer == None:
        return theta - eta * dtheta
    elif regularizer == 'L1':
        return theta - eta * dtheta + lamda * np.abs(theta)
    elif regularizer == 'L2':
        return theta - eta * dtheta + lamda * np.square(theta)
```

2.2 Paramètres

Nous pouvons modifier différents paramètres pouvant influencer les performances du modèle.

- Structure du réseau neuronal :
  - Nombre de couches cachées
  - Taille de chaque couche : Nombre de neurones composant chaque couche cachée
- Nombre d'époques : Nombre d'étapes d'apprentissage sur toutes les données d'apprentissage.
- Taux d'apprentissage : Ce paramètre joue sur le taux de modification des poids à chaque étape.
- Fonction d'activation : Le choix de la fonction d'activation.
- Le paramètre de régularisation : λ.
- Taille du batch. On utilisera 500 dans la plupart des tests.

3 Résultats et analyse

Les taux d'erreur décrits plus bas sont des taux d'erreurs sur l'ensemble de validation.

3.1 Sur la base de données MNIST

Test 1 : Une couche

Commençons avec un réseau simple possédant une couche cachée de taille 100.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[100]	20	0.05	tanh

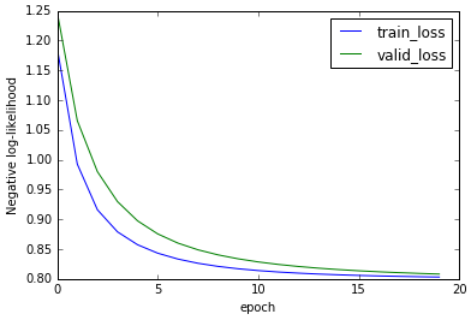

Figure 3: Évolution de la perte

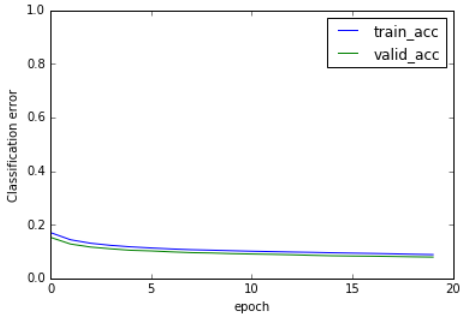

Figure 4: Évolution du taux d'erreur

Ces résultats sont cohérents et plutôt classiques. On voit que le taux d'erreur converge vite. Il atteint **0.08** au bout de 20 époques. L'évolution du likelihood est normal : on converge doucement et la loss sur l'ensemble d'entraînement est toujours plus faible que la loss sur l'ensemble de validation.

Test 2 : Trois couches

Dans ce test, on a 3 couches cachées de taille 50. On garde les mêmes paramètres pour le reste.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[50, 50, 50]	20	0.05	tanh

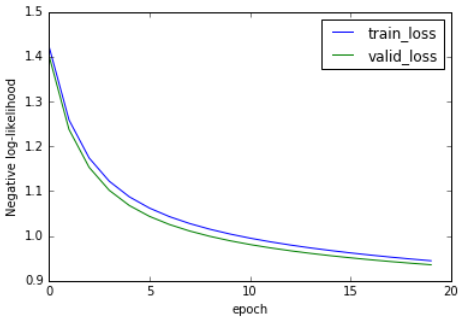

Figure 5: Évolution de la perte

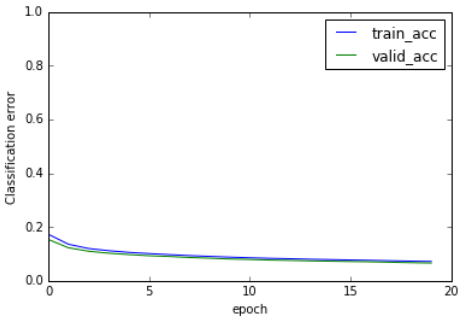

Figure 6: Évolution du taux d'erreur

On voit que le taux d'erreur s'améliore sur les 5 premières époques puis se stabilise à un taux d'erreur d'environ **0.066**. C'est un peu meilleur que précédemment. Cette fois-ci, la loss sur l'ensemble d'entraînement est supérieure à celle sur l'ensemble de validation.

Test 3 : Un plus grand réseau

Essayons à présent avec 5 couches cachées de taille 100.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[100, 100, 100, 100, 100]	20	0.05	tanh

Les graphiques étant similaires et n'apprenant rien de particulier, nous ne les afficheront pas pour ce test, ni pour le suivant. Le taux d'erreur à l'époque 19 est de **0.056**. On remarque jusqu'ici que plus le réseau possède de neurones et la précision s'améliore.

Test 4 : Un réseau complexe

Une fois de plus, on ne fait varier que la topologie du réseau de neurones. Essayons à présent un réseau plus ambitieux : 5 couches cachées, avec respectivement des tailles de 1024, 512, 256, 128 et 64 (puissance de 2 décroissante).

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[1024, 512, 256, 128, 64]	20	0.05	tanh

On atteint un taux record : **0.05**. La performance est donc légèrement meilleure mais le temps d'apprentissage est plus long.

Test 5 : Fonction d'activation Rectified Linear Unit

Essayons une autre fonction d'activation. Pour pouvoir comparer, le test 2 (les mêmes paramètres sauf la fonction). Le nombre d'époques à 20 semble pratique : suffisamment

grand pour voir le comportement jusqu'à convergence, et suffisamment petit pour que le calcul ne soit pas interminable.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[50, 50, 50]	20	0.05	**relu**

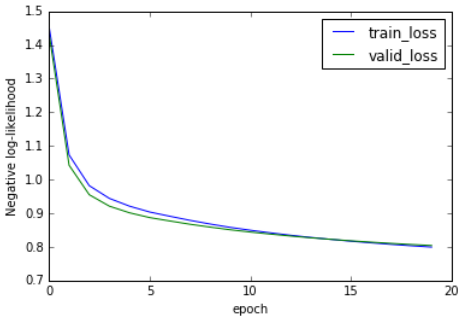

Figure 7: Évolution de la perte

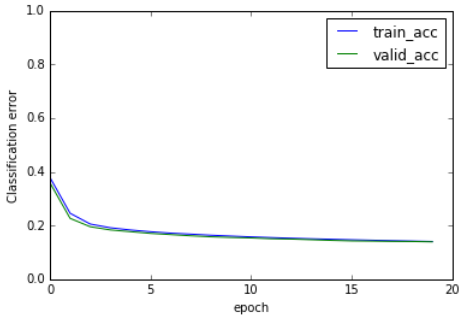

Figure 8: Évolution du taux d'erreur

Le taux d'erreur est de **0.14**. C'est moins bon qu'avec la tangente hyperbolique (0.066).

Test 6 : Fonction d'activation Sigmoid

Essayons à présent la fonction d'activation sigmoid. Nous utiliserons à nouveau les conditions du test 2.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[50, 50, 50]	20	0.05	**sigmoid**

Une fois de plus, les graphiques n'apportaient pas d'information particulière; on choisit donc de ne pas les afficher ici.

Le taux d'erreur est de **0.2**. C'est moins bon qu'avec relu (0.14) et qu'avec la tangente hyperbolique (0.066). Il serait intéressant de tester différents paramètres pour voir si cette fonction d'activation peut surpasser les autres dans certaines conditions. J'ai essayé avec des petits réseaux neuronaux, cependant je ne suis pas parvenu à obtenir une meilleure performance avec la fonction sigmoid.

Test 7 : Taux d'apprentissage

Augmentons le taux d'apprentissage à 0.5.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[50, 50, 50]	20	**0.5**	tanh

On obtient un taux d'erreur de **0.03**. Il s'agit du meilleur que l'on ai obtenu jusqu'à maintenant. Le taux d'apprentissage n'est donc pas à prendre à la légère. Attention tout de même au sur-apprentissage.

3.2 Sur la base de données CIFAR-10

Test 8 : CIFAR-10

Nous allons à présent tester notre modèle neuronal sur CIFAR-10. Voici les paramètres utilisés :

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[80, 80]	20	0.1	tanh

Tout se passe bien, la perte et la précision converge. Le taux d'erreur final est de **0.561**. C'est loin d'être excellent, même si la classification sur CIFAR-10 est plus difficile que sur MNIST.

Test 9 : Sur-apprentissage

Essayons d'améliorer les résultats avec plus de neurones et plus d'époques.

Couches de neurones	Époques	Taux d'apprentissage	Fonction d'activation
[100, 100, 100, 100, 100]	**80**	0.1	tanh

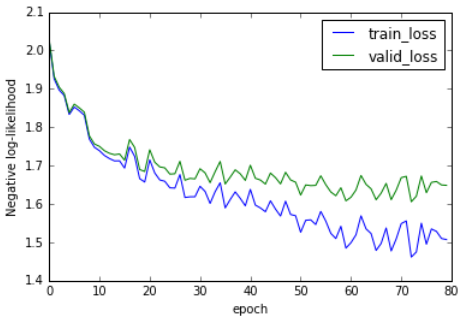

Figure 9: Évolution de la perte

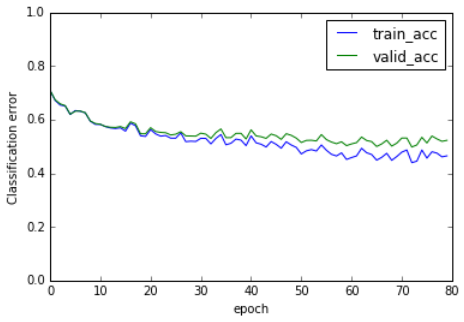

Figure 10: Évolution du taux d'erreur

Avec ces paramètres, on atteint un taux d'erreur de **0.50** à l'époque 65 et **0.52** à l'époque 79. On constate que le taux d'apprentissage α est trop élevé : la loss fluctue beaucoup. Parfois elle remonte. On remarque aussi un début de **sur-apprentissage** : la perte sur l'ensemble d'apprentissage continue de baisser alors que la perte sur l'ensemble de validation se stabilise, voir commence à remonter vers les dernières époques.

Conclusion

Les réseaux neuronaux offrent pleins de possibilités à explorer à travers les différents paramètres. Il est très intéressants d'en programmer un à la main afin d'en saisir le fonctionnement interne et le comportement.