Feed Forward Neural Network with Back Propagation Application on MNIST Dataset

Eden BELOUADAH

10 décembre 2017

1 Introduction

Le domaine d'apprentissage profond est l'un des domaines les plus récents qui s'inspirent du domaine biomédical. Il trouve des applications dans plein de problématiques réelles parmi lesquelles la classification d'image.

Le but de ce travail pratique est de mettre en œuvre un réseau de neurones Feed Forward avec la rétropropagation de l'erreur, et de l'appliquer sur la base de données MNIST.

La base de données MNIST est contruite de 50000 exemples d'images binaires numérisées. Chaque image contient un chiffre manuscrit et le but du réseau est d'associer l'image au bon chiffre correspondant parmi 10.

2 Apprentissage du réseau de neurones

Afin de faire cette tâche, il a fallu implémenter ces fonctions :

— Forward : Faire le calcul matriciel nécessaire pour propager les données sur le réseau de neurones à partir de la couche d'entrée jusqu'à la couche de sortie. La fonction, au plus des activations, fait sortir la dérivé de chaque activation.

$$z_j = \sum_{i \to j} W_{ij} x_i + b_j$$

— Softmax : Prendre les activations de la couche de sortie et les transformer en une probabilité d'appartenance des exemples aux différentes classes,

$$\sigma_i = P(t = i|x, W, b) = \frac{e^{z_i - \alpha}}{\sum_k e^{z_k - \alpha}}; \alpha = max(z_k)$$

La stabilité de $Softmax$ a été testée. La fonction ne cause pas de débordement si la valeur d'activation est très grande.

— Calcul du gradient : Se fait de cette manière :

$$\delta z_i = \sigma_i - 1_{i=l}$$

— BackPropagation : Rétropropagation de l'erreur sur le réseau à partir de la couche de sortie jusqu'à la première couche cachée et mise à jour des paramètres (W, B) du réseau,

$$\delta^{(l-1)} = f'^{(l-1)}(a^{(l-1)}) \circ (W^{(l)^t}\delta^{(l)})$$

$$W^{(l)} = W^{(l)} + reg^{(l)} - \eta_t \delta^{(l)} x^{(l)^t}$$

où $reg^{(l)}$ est le terme de régularisation et égale à :

— 0 si la régularisation n'est pas utilisée,

— $\lambda|W^{(l)}|$ dans le cas de la régularisation L1,

— $\lambda {W^{(l)}}^2$ dans le cas de la régularisation L2,

avec λ une constante scalaire.

— Calcul d'erreur : Calculer l'erreur et la précision du réseau de neurones en comparant les sorties du réseau avec les vrais classes des exemples,

$$c = -\sum_{(x_i,y_i)\in\mathcal{D}} \log P(y = y_i|x_i, W, b)$$

Trois fonctions d'activations ont été testées :

— Fonction $relu$:

$$relu(x) = max(x, 0)$$

$$relu'(x) = \begin{cases} 1 & \text{si} \quad x > 0 \\ 0 & \text{sinon} \end{cases} \tag{1}$$

— Fonction $sigmoid$:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$sigmoid'(x) = sigmoid(x)(1 - sigmoid(x))$$

— Fonction $tanh$:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$tanh'(x) = 1 - tanh^2(x)$$

La fonction d'activation finale utilisée est la fonction *relu*. Les deux fonctions *sigmoid* et *tanh* n'ont pas donné les meilleurs résultats surtout en augmentant le nombre de couches cachées. En effet, nous remarquons que lorsque les dérivées sont très petites, les deux fonctions *sigmoid* et *tanh* ne deviennent plus efficaces et l'effet du gradient disparait.

3 Architecture du réseau de neurones

L'architecture du réseau de neurones utilisée est la suivante :

— Nombre de couches cachées : 1,

— Nombre de neurones dans la couche cachée : 300,

— Fonction d'activation : ReLU,

— Erreur : Quadratique,

— Taille de Batch : 500.

L'instabilité numérique de la fonction *relu* par rapport aux deux autres fonctions nous mène à adapter le taux d'apprentissage au fur et à mesure des itérations, En effet, η est initialisé au début à 0.001 et changera de valeur dynamiquement selon la valeur d'erreur qu'on cherche à optimiser. Si l'erreur diminue, on augmente le pas d'apprentissage par 20%, et si elle augmente, on retourne aux valeurs précédentes de W et de B et on diminue le pas d'apprentissage par 20%.

4 Analyse des résultats

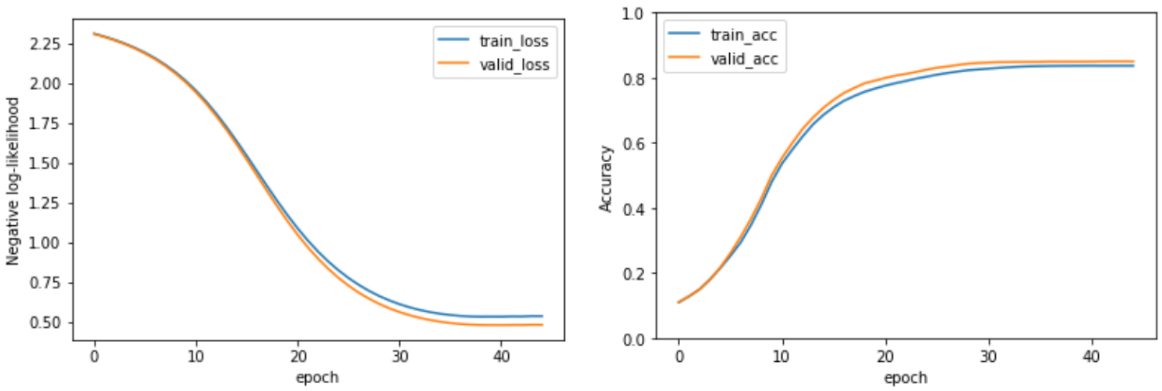
Avec moins de 45 itérations et en un temps réduit, le réseau de neurones est capable d'atteindre une précision de 86%. Cela n'empêche pas d'avoir un meilleure précision, mais le temps d'exécution augmentera en compliquant l'architecture du réseau. Les résultats sont les suivants :

Ensemble	Précision	Erreur
Apprentissage	85.89%	0.49
Validation	86.93%	0.45

Tableau 1. Précision et Erreur sur les ensembles d'apprentissage et de validation

Les résultats sont assez bons pour un réseau de neurones à architecture simple.

Les courbes d'apprentissage sont les suivantes :



(a) Erreur *(b) Précision*

Figure 1. Précision et erreur sur les ensembles d'apprentissage et de validation

Nous remarquons que l'erreur sur les ensembles d'apprentissage et de validation diminue efficacement au fil des itérations, puis commence à se stabiliser à la fin lorsque on se trouve dans un optimium local.

Nous remarquons que même sans utiliser la régularisation, le modèle ne souffre pas de sur apprentissage, vu que la courbe d'erreur sur l'ensemble de validation ne diverge pas de celle de l'ensemble d'apprentissage.

Quant à la précision, nous remarquons qu'elle est en train d'augmenter pour les deux ensembles, jusqu'à ce qu'elle se stabilise aux dernières itérations.

5 Conclusion

Ce travail pratique était une occasion pour voir de plus proche le fonctionnement des réseaux de neurones. Ces derniers peuvent changer d'architecture en variant le type du problème, sa difficulté, ses particularités...

La base de données MNIST représente une tâche de classification simple qui peut être résolue avec un réseau de neurone simple, même sans couches cachées, comme nous l'avons vu dans le travail pratique précédent. En effet, il était possible d'atteindre une précision de 87% facilement et en un peu de temps.

Les architectures complexes sont plus adaptées aux problèmes complexes qui nécessitent vraiment un réseau profond, c'est la raison pour laquelle on s'est contenté d'une seule couche cachée dans ce travail.