Master TRIED

Module : **Neural Networks and Deep Learning**

Rapport du TP

« Réseaux de neurones et Apprentissage profond »

Réalisé par :

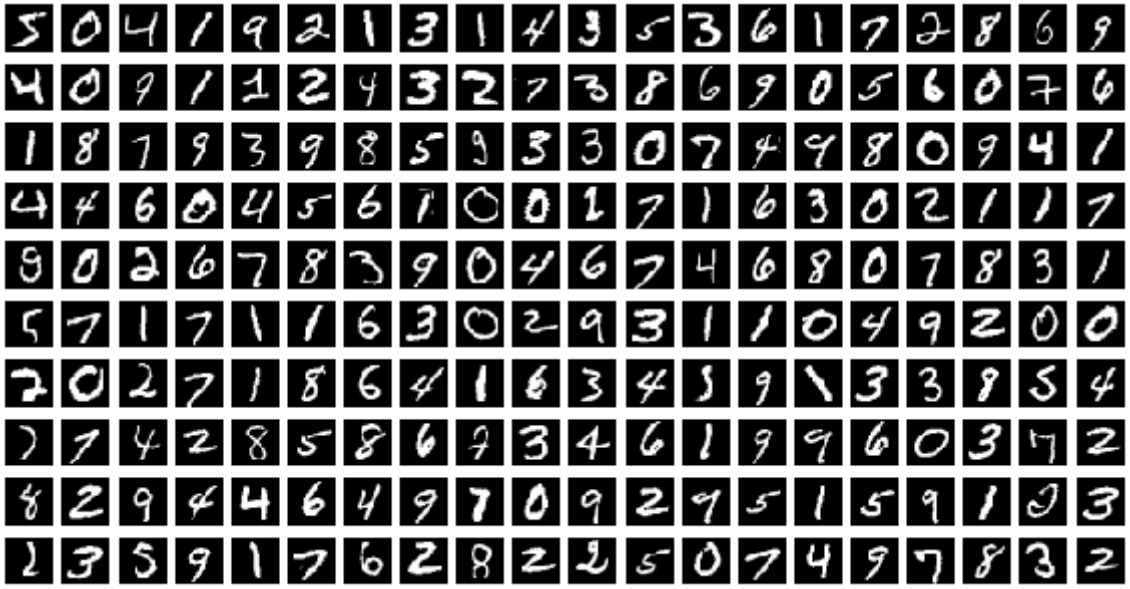
Ahmed ALLALI

Zakaria SAADI

Algorithme de rétro-propagation de l'erreur

Exercice 0 (visualisation)

On commence par afficher les 200 premières images de la base d'apprentissage



Question :

Quel est l'espace dans lequel se trouvent les images ? Quel est sa taille ?

Réponse :

L'image est stocké dans la mémoire dans sous forme d'un vecteur de taille 784

Exercice 1 (regression logistique)

Question :

Quel est le nombre de paramètres du modèle ? Justifier le calcul.

Réponse :

Le nombre de parametres est eagle a (le nombre d elements de la matrice W et le B)

784*10+10=7850

Car on a une seule couche et le reseau est entièrement connecté + des biais

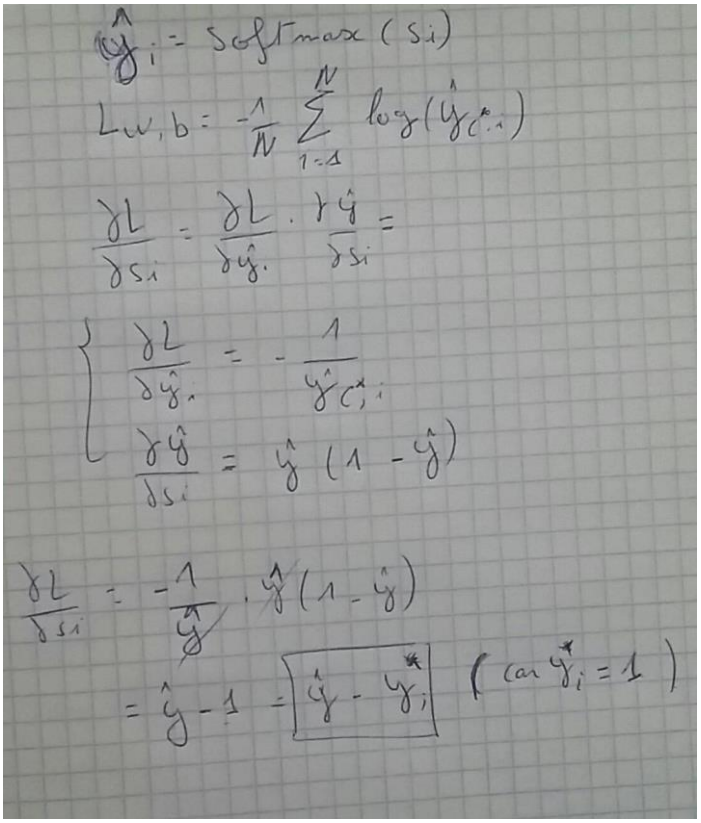
Question :

La fonction de coût de l'Eq. (3) est-elle convexe par rapports aux paramètres W, b du modèle ? Avec un pas de gradient bien choisi, peut-on assurer la convergence vers le minimum global de la solution ?

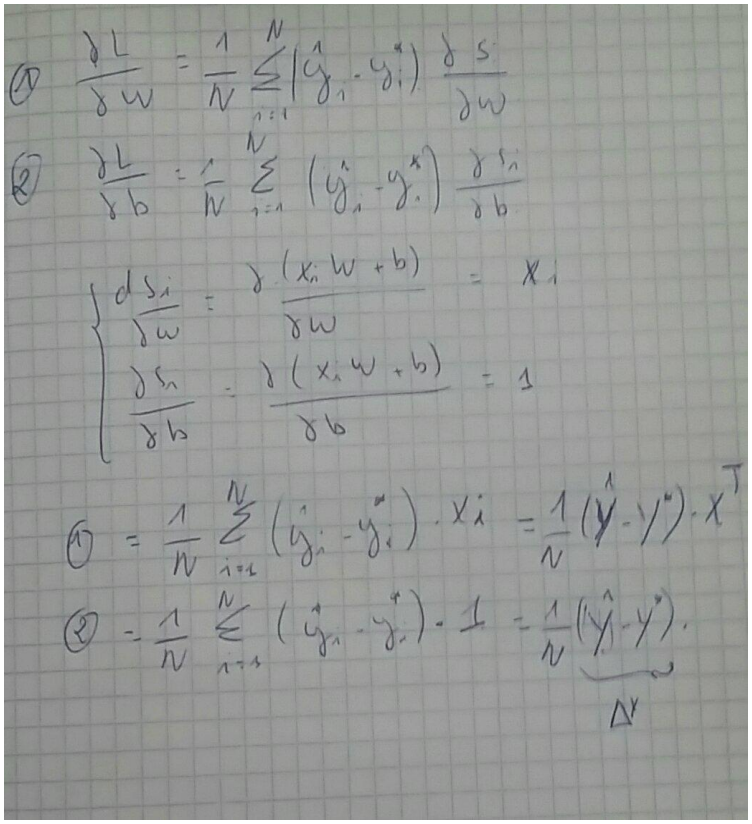
Réponse :

La fonction de cout n’est pas convexe donc il n’existe pas un un seul minimum global .et pour assurer qu’on va tomber au-dessus il faut choisir un pas de gradient grand au début et le diminuer durant l’exécution .

Démonstration des formules :



Donc on aura :



Deep Learning avec Keras

Exercice 1 : Régression Logistique avec Keras

Dans cet exercice, on a écrit un script permettant de créer le réseau de neurone ayant l'architecture suivante :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 10)                7850
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

c'est un modèle de régression logistique.

Ensuite, pour ce modèle, on a défini l'entropie croisée comme un loss , la descente de gradient stochastique comme une méthode d'optimisation, et le taux de bonne prédiction des catégories comme une métrique d'évaluation.

Pour un batch_size de 300 et un nombre de « epoch » de 10, on a eu :

loss: 27.37% acc: 92.26%

Pour un batch_size de 300 et un nombre de « epoch » de 15, on a eu :

loss: 26.87% acc: 92.38%

Pour un batch_size de 300 et un nombre de « epoch » de 20, on a eu :

loss: 27.04% acc: 92.42%

Pour un batch_size de 400 et un nombre de « epoch » de 10, on a eu :

loss: 26.62% acc: 92.60%

Pour un batch_size de 500 et un nombre de « epoch » de 10, on a eu :

loss: 26.59% acc: 92.54%

On remarque bien que la prédiction s'améliore avec l'augmentation du nombre d'époques, ce qui est logique car Une époque est un passage complet à travers toutes les données d'entraînement. Un réseau de neurones continue à «apprendre» jusqu'à ce que le taux d'erreur soit acceptable, ce qui nécessite souvent plusieurs passages dans l'ensemble des données.

Par contre, on remarque qu'un batch plus grand ne veut pas dire nécessairement un meilleur apprentissage car le gradient stochastique (SGD) repose sur le calcul du gradient négatif du critère après avoir traité seulement quelques exemples d'apprentissage d'une manière aléatoire.

Exercice 2 : Perceptron avec Keras

Dans cet exercice, on crée obtenir un réseau de neurones à une couche cachée « le Perceptron », dont la façon de l'entraîner va être strictement identique à ce qui a été fait dans l'exercice 1, l'algorithme de rétro-propagation du gradient de l'erreur permettant de mettre à jour l'ensemble des paramètres du réseau.

Dans les mémes conditions d'apprentissage que l'exercice 1 (un batch_size de 300 et un nombre de « epoch » de 10), on aura :

loss: 18.07% acc: 94.66%

On remarque qu'on a eu une meilleure performance que celle obtenu dans l'exercice précédent, cela est dû au fait que la couche cachée du Perceptron crée et maintient une représentation interne des données.

Si on change la valeur du «learning_rate» et on le met égal à 0.2 au lieu de 0.5, on aura :

loss: 25.16% acc: 92.89%

Si «learning_rate» a une valeur de 0.3, on obtient :

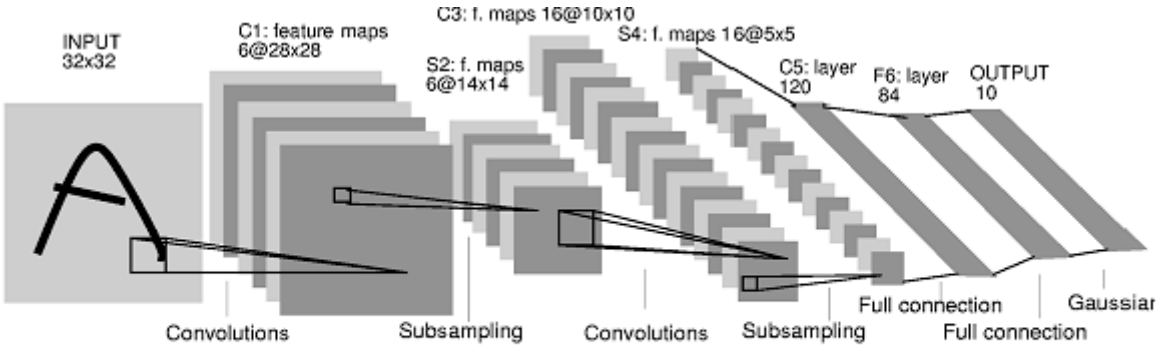
loss: 22.21% acc: 93.70%

on remarque que le fait de choisir un pas d'apprentissage trop petit n'apporte pas toujours d'améliorations à la performance, cela pourrait être le cas si ça se passe à la fin de l'apprentissage (learning rate schedules).

Exercice 3 :

On va s'intéresser dans cet exercice aux réseaux convolutifs. Pour traiter les images issus du MNIST, on a créé un réseau avec la structure suivante :

- Des couches de convolution avec 32 filtres de taille spatiale (5,5) une fonction d'activation de type sigmoid
- Des couches d'agrégation spatiale (pooling) de taille spatiale (2, 2)



On détaillera par la suite le fonctionnement et l'intérêt de ces deux couches :

Convolutions

Les images naturelles ont la propriété d'être «stationnaires», ce qui signifie que les statistiques d'une partie de l'image sont les mêmes que pour n'importe quelle autre partie. Cela suggère que les caractéristiques que nous apprenons à une partie de l'image peuvent également être appliquées à d'autres parties de l'image, et nous pouvons utiliser les mêmes caractéristiques à tous les endroits.

Plus précisément, après avoir appris des caractéristiques sur de petits échantillons (5x5) échantillonnés au hasard à partir de l'image plus grande, nous pouvons ensuite appliquer ce détecteur de caractéristiques 5x5 appris n'importe où dans l'image. Plus précisément, nous pouvons prendre les représentations apprises (5x5) et leurs appliquer une convolution avec l'image plus grande, obtenant ainsi une valeur d'activation différente à chaque endroit de l'image.

Pooling

Après avoir obtenu des caractéristiques en utilisant la convolution, nous aimerions ensuite les utiliser pour la classification. Considérons par exemple des images de taille 96x96 pixels, et supposons que nous avons appris 400 caractéristiques sur 8x8 entrées. Chaque convolution donne une sortie de taille (96-8 + 1) * (96-8 + 1) = 7921 (96-8 + 1) * (96-8 + 1) = 7921, et comme nous avons 400 caractéristiques, il en résulte un vecteur de 892 * 400 = 3,168,400892 * 400 = 3, 168,400 caractéristiques par exemple. L'apprentissage d'un classifieur avec des entrées ayant plus de 3

millions de fonctionnalités peut être compliqué et peut également être sujet à un ajustement excessif.

Pour résoudre ce problème, il faut se rappeler que nous avons décidé d'obtenir des caractéristiques convoluées parce que les images ont la propriété de «stationnarité», ce qui implique que les caractéristiques utiles dans une région sont également susceptibles d'être utiles pour d'autres régions. Ainsi, pour décrire une grande image, une approche naturelle consiste à agréger des statistiques de ces caractéristiques à divers endroits. Par exemple, on pourrait calculer la valeur moyenne (ou max) d'une caractéristique particulière sur une région de l'image. Ces statistiques récapitulatives ont une dimension beaucoup plus faible (par rapport à l'utilisation de toutes les fonctionnalités extraites) et peuvent également améliorer les résultats (moins de sur-apprentissage).

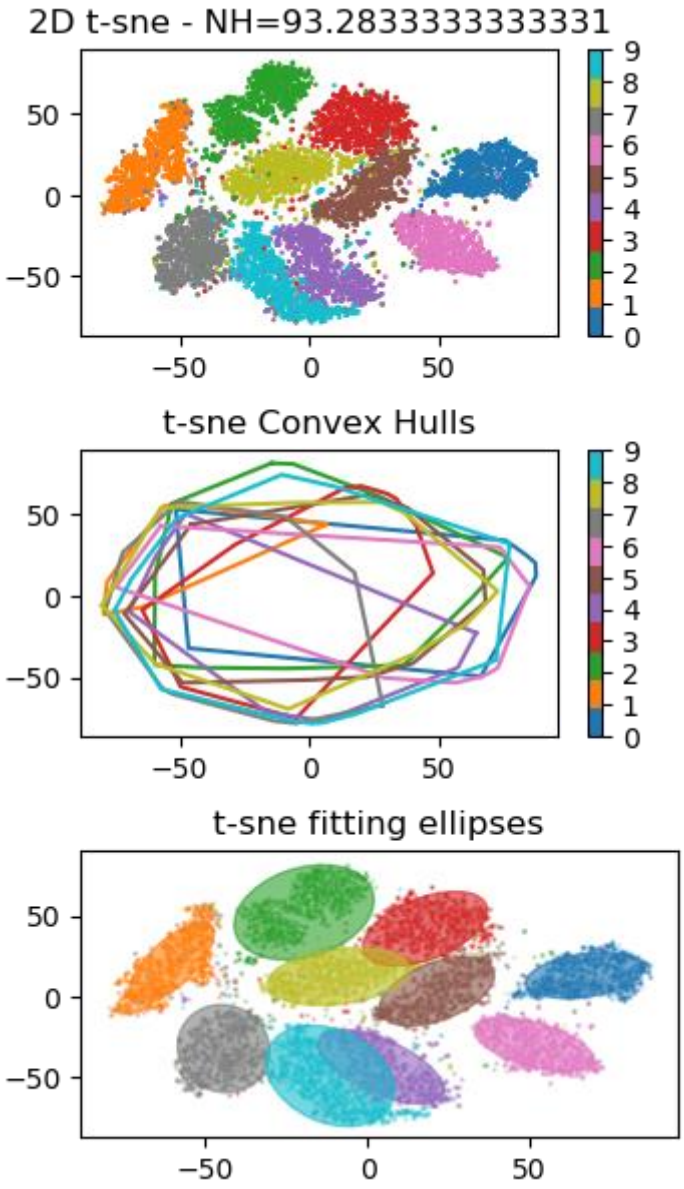
- On a effectué l'apprentissage du modèle et évalué les performances du réseau sur la base de test.
  Pour ce modèle ConvNet, on a obtenu le score suivant :

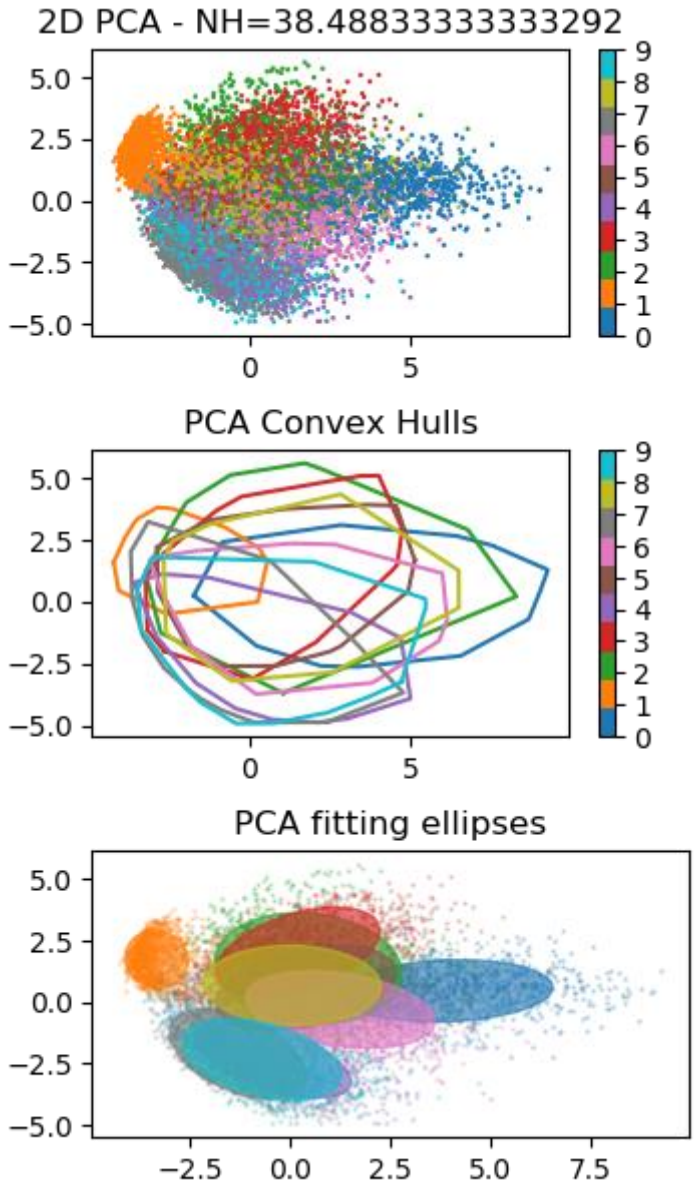
  loss: 10.03% acc: 98.26%

Deep Learning et Manifold Untangling

On va utiliser deux outils de visualisation : t-SNE et une Analyse en Composantes Principales (ACP), qui vont permettre de représenter chaque donnée (par exemple une image de la base MNIST) par un point dans l'espace 2D. Ces même outils vont permettre de projeter en 2D les représentations internes des réseaux de neurones, ce qui va permettre d'analyser la séparabilité des points et des classes dans l'espace d'entrée et dans les espaces de représentions appris par les modèles.

L'application de la méthode visualisation sur les données de test de la base MNIST produit les résultats suivants :


2D t-sne - NH=93.2833333333331
50
0
−50
−50
0
50
9
8
7
6
5
4
3
2
1
0
t-sne Convex Hulls
50
0
−50
−50
0
50
9
8
7
6
5
4
3
2
1
0
t-sne fitting ellipses
50
0
−50
−50
0
50



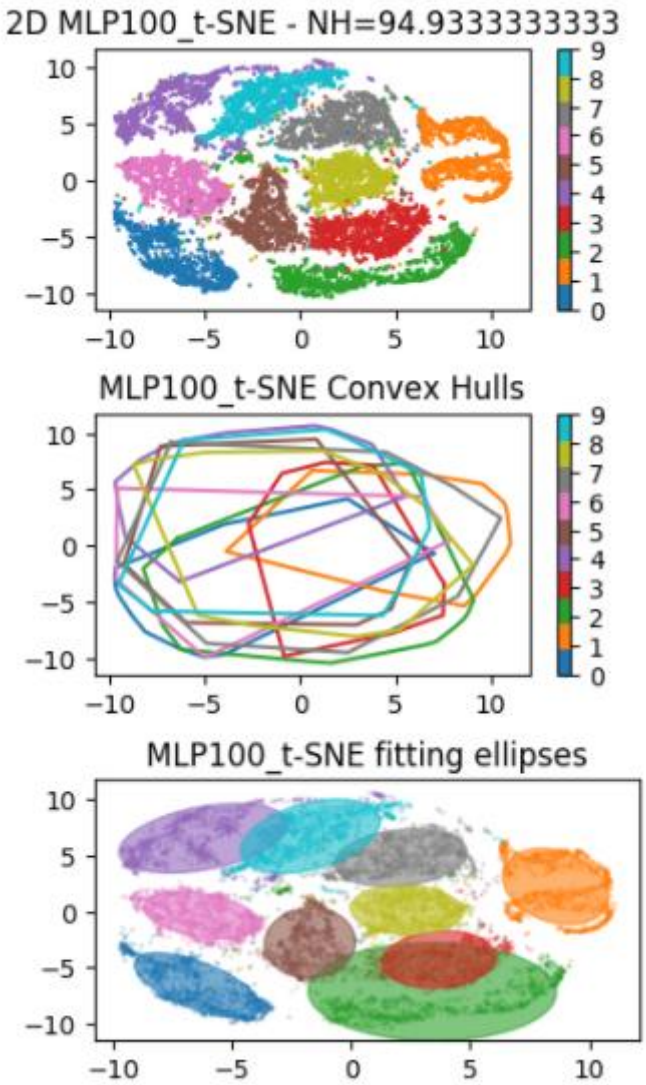
Comparaison de la méthode t-SNE à une Analyse en Composantes Principales (ACP)

t-SNE	**Analyse en Composantes Principales**
elle est non linéaire, donc elle peut détecter la structure des manifolds plus difficiles. Elle implique des hyper paramètres contrairement à l'ACP	Peut être calculé de manière itérative, donc si vous avez déjà calculé k composantes de principe, mais alors vous décidez que vous voulez une représentation dimensionnelle (k + 1) qui est seulement un peu plus de calcul. Les composants principaux sont une base orthogonale triée par la quantité de variance le long de la dimension particulière. Cela fait que la direction des principaux vecteurs du composant raconte

	une histoire à propos de vos données d'origine: quelles sont les variables qui expliquent le plus ou le moins de variation? Une fois ajusté, vous obtenez une transformation linéaire pour la réduction de la dimensionnalité d'autres points non inclus dans l'ensemble de données. On ne peut pas en dire autant de T-SNE qui minimise directement la distance entre l'ensemble de données et sa réduction de dimensionnalité par descente de gradient. Cela donne une correspondance pour les points connus, mais pas une fonction pour les nouveaux points, donc vous devrez faire une interpolation post-hoc ou partir de zéro.

Exercice 3 :

On va maintenant s'intéresser à visualisation de l'effet de "manifold untangling" permis par les réseaux de neurones.



Optimisation: Descente de gradient stochastique

Aperçu

Les méthodes batch, telles que BFGS à mémoire limitée, qui utilisent l'ensemble d'apprentissage complet pour calculer la prochaine mise à jour des paramètres à chaque itération tendent à converger très bien vers les optima locaux. Ils sont aussi simples à travailler, à condition d'avoir une bonne implémentation standard (par exemple minFunc) car ils ont très peu d'hyper-paramètres à régler. Cependant, dans la pratique, le calcul du coût et du gradient pour l'ensemble de l'ensemble de la formation peut souvent être très lent et parfois difficile sur une seule machine si l'ensemble de données est trop grand pour tenir dans la mémoire principale. Un autre problème avec les méthodes d'optimisation par lots est qu'elles ne permettent pas d'intégrer facilement de nouvelles données dans un environnement "en ligne". La descente de gradient stochastique (SGD) aborde ces deux problèmes en suivant le gradient négatif de l'objectif après avoir vu seulement un ou quelques exemples d'entraînement. L'utilisation de SGD Dans le cadre du réseau de neurones est motivé par le coût élevé de la rétrogradation sur l'ensemble complet de formation. SGD peut surmonter ce coût et conduire à une convergence rapide.