UNIVERSITÉ DE MONTRÉAL

Rapport : Devoir3

Pierre Gérard
Mathieu Bouchard

IFT3395-6390 Fondements de l'apprentissage machine
Pascal Vincent, Alexandre de Brébisson et César Laurent

Année académique 2015 - 2016

1 Partie pratique : Implémentation du réseau de neurones

1.1 Utilisation

Pour tester notre programme, exécutez simplement la commande dans le dossier *src* :

```
python main.py
```

Les exercice 5 et 9/10 s'exécutent en dernier car ils prennent le plus de temps.

1.2 Notes relative à l'implémentation

1.3 Exercices 1 et 2

Nous avons implémentée la vérification de gradient par différence finie. Pour chaque scalaire "perturbé", on conserve le ratio dans une liste. Une boucle vérifie ensuite que les ratios sont bien entre 0.99 et 1.01 .Les résultats semblent indiqué que notre implémentation est correcte.

Pour un réseau $2 \times 2 \times 2$, on obtient :

```
>>EXERCICE 1 et 2
Liste des ratio W1, b1, W2, b2
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0.99997500062302613, 1.0000250006249738]
>Tout les ratio sont bien entre 0.99 et 1.01
```

1.4 Exercices 3 et 4

On procède comme à l'exercice 1 et 2 sauf qu'on somme l'ensemble des gradients pour K exemples avant de calculer le ratio. Les résultats semblent eux-aussi indiqué que notre implémentation est correcte.

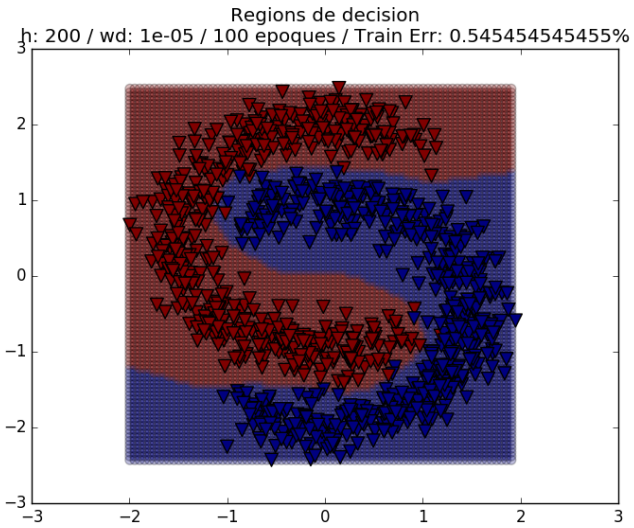
Pour un réseau $2 \times 5 \times 2$, on obtient :

```
>>EXERCICE 3 et 4
Liste des ratio W1, b1, W2, b2
[1, 1, 1.000002072572213, 1.000008244958942, 1, 1, 0.99999469791365814, 0.9999789080394954
>Tout les ratio sont bien entre 0.99 et 1.01
```

1.5 Exercices 5

On a ensuite entrainé notre réseau par descente de gradient mini-batch. L'entrainement semble un succès.

Les résultats semblent plus on augmente le nombre d'époques et de neurones cachés, plus l'erreur sera petite.



1.6 Exercices 6 et 7

1.6.1 Implémentation

Explication des modifs

1.6.2 Verification

Pour vérifier que les gradients soient équivalents, on commence par instancier les deux classes correspondant aux deux implémentations $NeuralNetwork$ et $NeuralNetworkEfficient$. On initie ensuite $W1$ et $W2$ des deux classes aux mêmes valeurs. Ensuite on execute l'entrainement pour K=1. On utilise ensuite une méthode qui pour deux réseaux de neurones vérifie que les gradients soient équivalent. On recommence le procédé pour K=10.

Les tests de comparaison nous donne (ok signifie que les deux gradients sont égaux) :

```
--- K=1 ---
Test Ok : gradient b2
Test Ok : gradient w2
Test Ok : gradient b1
Test Ok : gradient w1
 --- K=10 ---
Test Ok : gradient b2
Test Ok : gradient w2
Test Ok : gradient b1
Test Ok : gradient w1
```

1.7 Exercices 8

Les résultats des mesures de temps sont les suivant :

```
>>EXERCICE 8 MNIST
--- Reseau de depart ---
Cela a mis : 0.419163 secondes
```

```
--- Reseau optimise ---
Cela a mis : 0.123445 secondes
```

Sur cette exécution, on remarque que le temps d'exécution sur le réseau avec matrice est entre 3 et 4 fois plus rapide que sur le réseau avec boucles. Le gain de performance n'est donc pas négligeable.

1.8 Exercices 9 et 10

Pour alléger la charge de calcul et ainsi augmenter la vitesse d'exécussion sur les données MNIST, nous avons décider de calculer les valeurs d'erreur et de coût moyen à toutes les 10 épques.

Pendant l'exécution, les informations sont affichées comme ceci :

```
>>EXERCICE 9-10
Train Err;Valid Err;Test Err;Avg Cost Train;Avg Cost Valid;Avg Cost Test
89.598;88.860;90.020;2.300;2.299;2.300 (0)
77.486;75.380;76.970;2.225;2.222;2.224 (10)
64.096;62.000;64.990;2.156;2.150;2.154 (20)
57.392;55.690;58.460;2.058;2.050;2.054 (30)
49.040;46.590;48.940;1.932;1.920;1.924 (40)
45.362;43.740;45.610;1.804;1.788;1.795 (50)
37.700;35.720;37.270;1.669;1.649;1.656 (60)
32.322;29.390;31.920;1.530;1.507;1.516 (70)
24.604;22.070;23.910;1.391;1.364;1.372 (80)
25.958;23.320;24.950;1.254;1.221;1.235 (90)
23.490;20.810;22.470;1.143;1.107;1.121 (100)
23.094;20.320;21.760;1.053;1.015;1.031 (110)
21.656;18.420;20.360;0.970;0.930;0.946 (120)
19.366;16.830;18.000;0.898;0.854;0.872 (130)
18.676;16.130;17.380;0.839;0.794;0.814 (140)
18.384;15.880;17.270;0.803;0.757;0.778 (150)
18.462;16.030;17.370;0.758;0.713;0.732 (160)
17.796;15.560;16.500;0.720;0.675;0.695 (170)
18.480;16.080;17.460;0.707;0.662;0.684 (180)
16.594;14.350;15.560;0.669;0.622;0.645 (190)
```

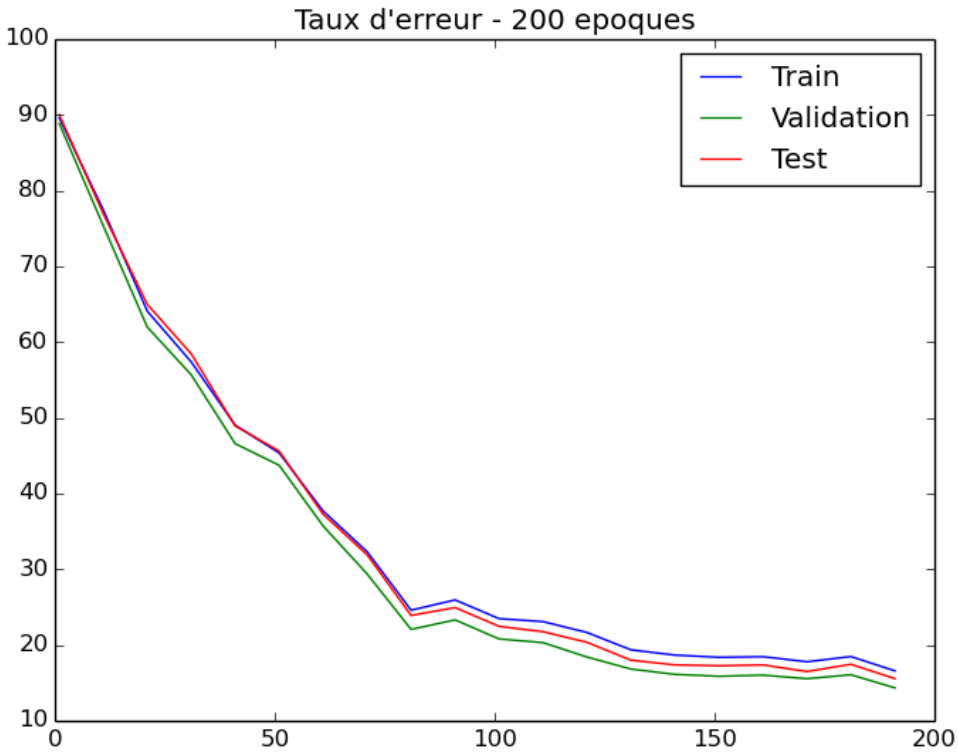
On retrouve alors dans l'ordre les 3 valeurs d'erreur de classification (train, validation, test) suivies des 3 valeurs de coût moyen (train, validation, test). Finalemnet, la valeur entre parenthèses représente le numéro de l'époque pour laquelle ces valeurs ont été calculées.

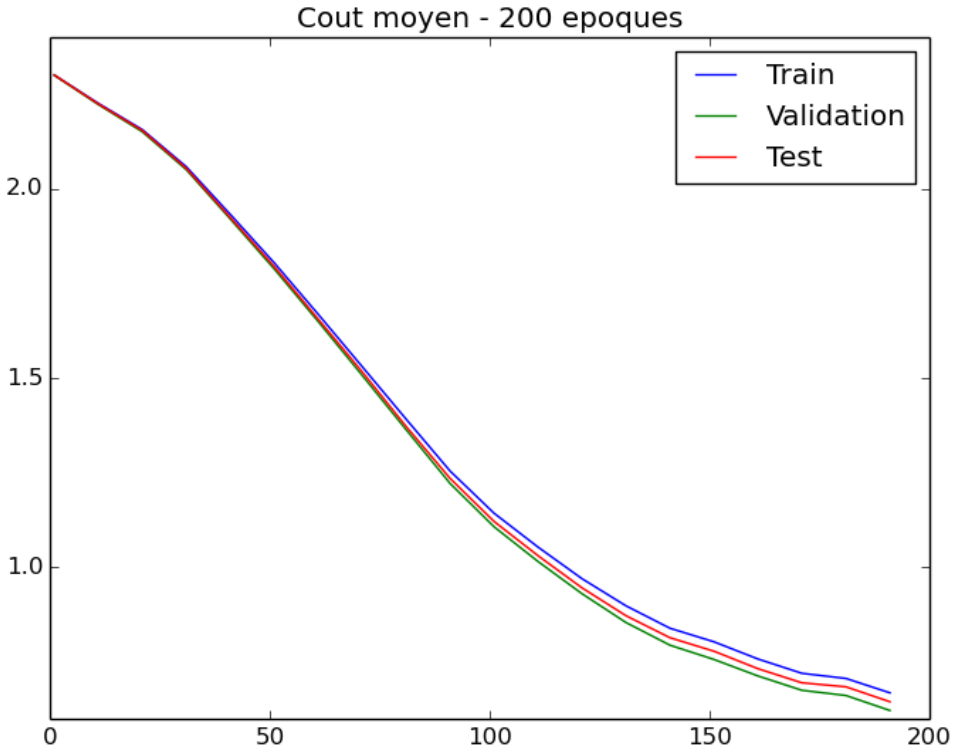
À la toute fin de l'exécution, ces valeurs sont également enregistrées dans un fichier texte nommé 9.txt se trouvant à la racine du dossier src.

Nous avons obtenus des résultats optimaux lorsque nos hyper-paramètres avaients les valeurs suivantes :

```
h = 30
wd = 0.0001
maxIter = 200
K = 50
```

On obtient alors les résultats suivants :




Cout moyen - 200 epoques
Train
Validation
Test
2.0
1.5
1.0
0
50
100
150
200