SERVAIS Nikita
N°463055

Info-F-106 : Projet d'Informatique
Classification à l'aide de réseaux de neurones

Table des matières

1. Introduction

Normalement quand il y a un problème à résoudre en programmation, il faut trouver la façon de « dire » à l'ordinateur comment le résoudre. En comprenant le problème, il est possible de trouver des algorithmes précis et performant que l'ordinateur va pouvoir effectuer facilement. Mais certains problèmes sont beaucoup plus complexe à résoudre avec les manières conventionnelles de l'informatique, comme la reconnaissance d'images, la reconnaissance de paterne dans des données. Et comme ces problèmes sont résolu avec une certaine aisance par les animaux et particulièrement les Humains. C'est de cette idée qu'est né les réseaux neuronaux qui imitent vaguement le comportement des neurones dans le cerveau. Les réseaux neuronaux ne reçoivent pas d'explication quand a la façon de résoudre le problème mais ils apprennent via l'observation de donnée, comme nous. Comment est-il possible de faire apprendre un ordinateur ? L'idée est de prendre un grand nombre de données (les données d'entrainements) et de développer un système qui va apprendre depuis ces données. Ce système est appelé perceptron qui représente un neurone qui peut donc être activé ou pas en combinant une multitude de perceptron on peut obtenir un système complexe. Pour apprendre le système utilise le concept mathématique du gradient qui est le dérivé partiel d'une fonction et grâce au gradient le réseau neuronal est capable de trouver un minimum local.

2. Perceptron multi classe et multicouche

Dans les premières parties du projet on utilisait des perceptrons avec uniquement 2 classes : active ou pas activer. Mon premiers choix d'implémentation est de faire un perceptron multi class (10 classe). Et du coup d'utilisé une fonction de coût (je reviendrai sur la fonction de cout plus bas) pour déterminer l'erreur de la couche de sortie au lieu de faire une validation croisé des différents perceptrons.

3. Normalisation des données

La normalisation des données permet d'avoir chaque input comprit entre 0 et 1. Cela facilite l'initialisation des poids.

4. Bais

L'implémentation des bais consiste simplement a ajouté b une constante à chaque couche. Les bais sont ajouté pour augmenter la flexibilité du model.

5. Fonction de coût

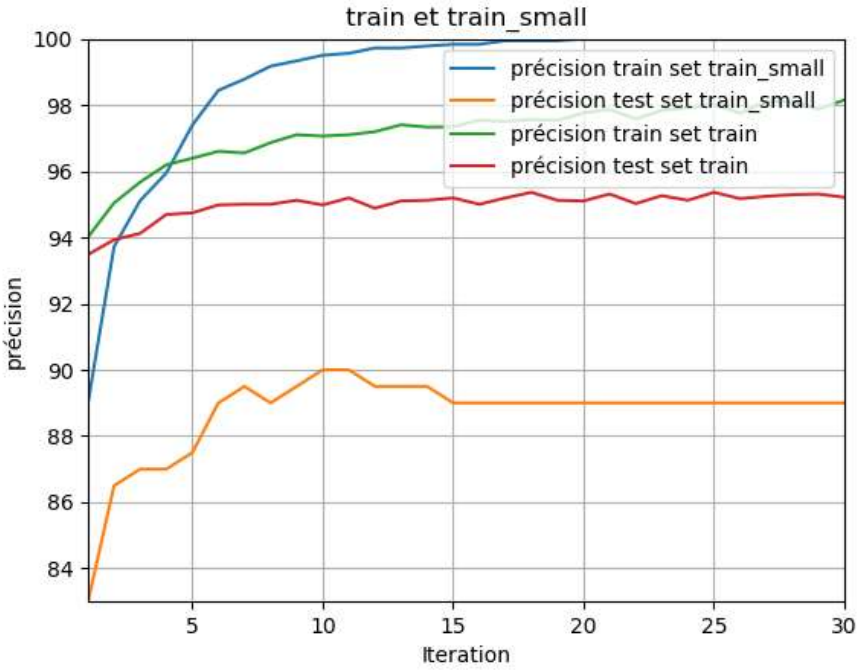
Lors de l'implémentation du perceptron multi classe, une fonction de coût a du être utiliser pour calculer l'erreur du réseau. La fonction proposé dans l'énonce est appelé fonction quadratique. Mais elle crée un problème qui se trouve être très récurant dans les réseaux neuronaux, le ralentissement de l'apprentissage. Et comme l'apprentissage est déterminé par le dérivé partiel de la fonction de cout, dire que l'apprentissage est lent revient à dire que le dérivé partiel est petit. Avec la fonction quadratique la dérivé partielle est calculer par le produit de la dérivé de la fonction logistique et de la différence entre l'activation de la couche de sortie et y. le problème vient donc du fait que la dérivé de la sigmoïde est très petite quand l'input est très grand ou très petit. Pour remédier à ce problème il faudrait ce débarrasse de la dérivé de la fonction logistique. C'est plus intuitif que plus l'erreur ne soit grande et plus le réseau n'apprend.

Une solution très courante dans les réseaux neuronaux est l'utilisation de la fonction de cout appelé entropie croisée qui via des manipulations mathématiques que je ne sais pas expliqué arrive supprimé la dérivé de la fonction logistique du calcul du cout du réseau. Ce qui est bien

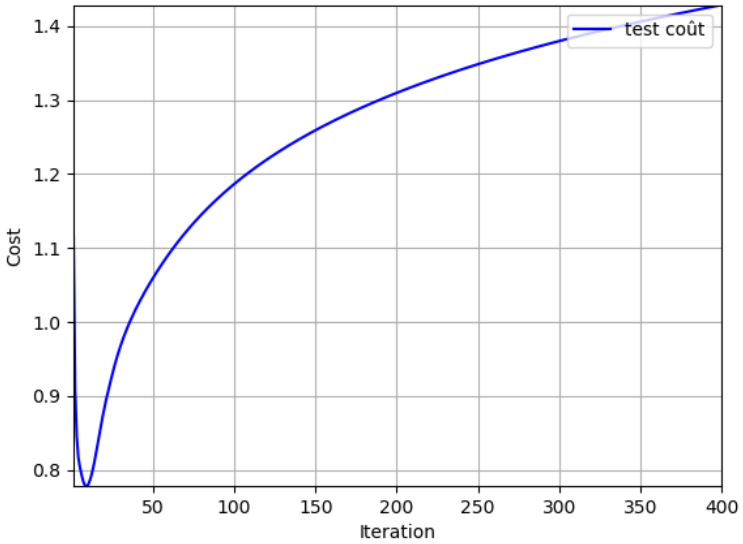
c'est que en pratique l'algorithme de backpropagation change que minimalement, car il suffit d'enlever le produit du cout du réseau et de la dérivé de la fonction logistique. Le reste est inchangé. Bien sur la formule du cout des réseaux pour chaque input n'est plus vraiment la même fonction, la fonction quadratique $C = \frac{(y-a)^2}{2n}$ et la fonction entropie croisée $C = -\frac{1}{n}\sum_x [ylna + (1-y)\ln(1-a)]$

6. Surentrainement

Un autre problème des réseaux neuronaux est le surentrainement. C'est quand les améliorations que réseau effectuer ne s'applique plus a des donnée différentes des donnée d'entrainement. Le réseau apprend les données d'entrainement par cœur. Il existe plusieurs façons de résoudre le problème, le premier étant simplement d'avoir plus de données d'entrainement mais parfois c'est assez difficile d'avoir plus de données. Et donc différente technique existe pour remédier à cela.



Ici on peut voir la différence entre la précision du train set et du test set avec 2000 images est de plus de 10% (ce qui montre un surentrainement du réseau). Et donc en augmentant les données à 50000 images la différence de précision est seulement de plus ou moins 3%. Ce qui résout déjà une grande partie du surentrainement.

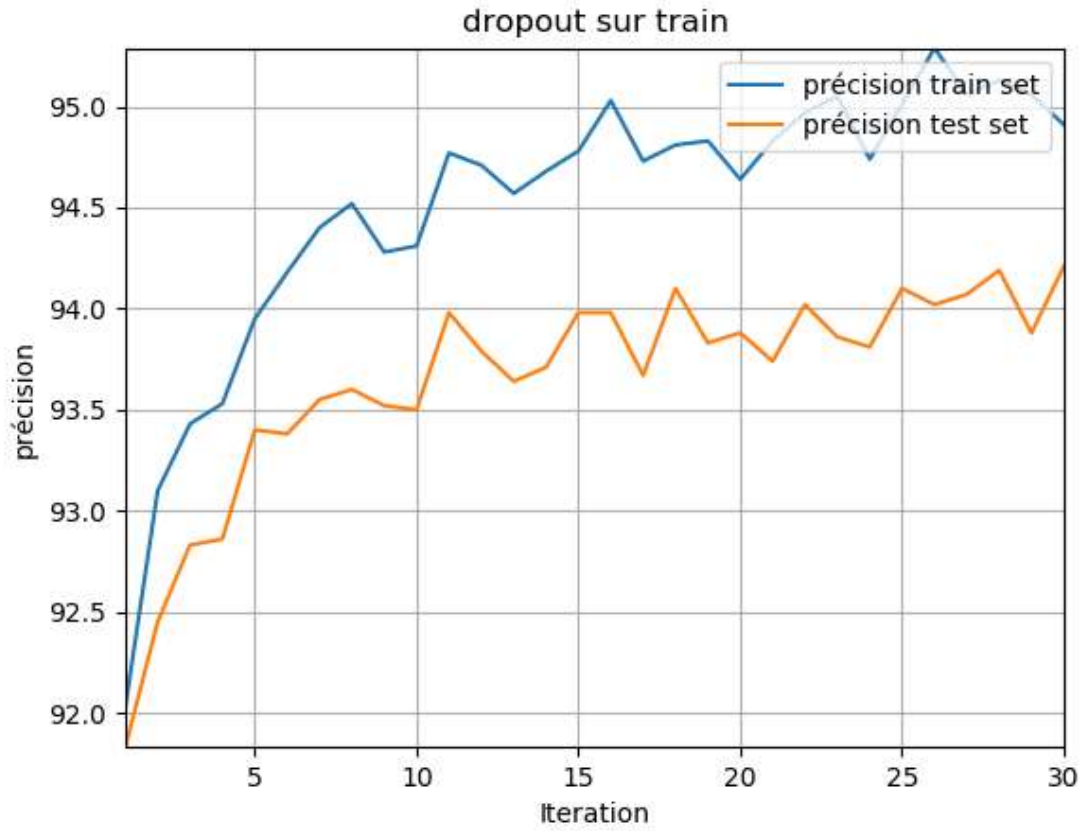


On peut aussi voir que malgré que le réseau s'entraine, le coût du réseau sur le test set s'empire avec les itérations. (2000 images)

1. Dropout

Le dropout consiste à désactivé avec une certain probabilité chaque neurone. Cela a pour effet d'empêcher les réseaux de trop s'appuyer sur des neurones en particulier(le dropout ajout un bruit qui perturbe le réseau et le force à mieux apprendre). Et donc les forces à apprendre des éléments différents des données. Ceci réduit grandement le problème de surentrainement. En pratique, on crée un masque de 1 et 0 avec une certaine probabilité de 0 et on multiplie ce masque avec les poids des neurones. Ensuite normalement on doit multiplier (par 1- la probabilité de dropout) les poids lors du test parce que les poids sont plus petit que ce qui est attendu. Mais ce qui est plus pratique à implémenté c'est le dropout inversé. Au lieu de multiplie lors de du test, on divise les activations par 1-p.

Concrètement dans mon programme j'utilise un dropout différent pour la couche d'input car si le dropout est trop grand sur la couche d'input cela a un effet négatif (dropout=0.2 pour la couche d'entré).



Learning rate=0 .05, dropout=0.2, alpha=0.9
On peut voir que la différence de précision entre le train et le test set n'est plus que de 1%.

2. Max norme

Ceci consiste à restreindre la norme des poids avec un constante c. cette technique est connue pour être efficace avec dropout. Utilisé une contrainte pour les poids permet aussi d'utilisé un plus grand learning rate car la contrainte empêche les poids d'explosé.

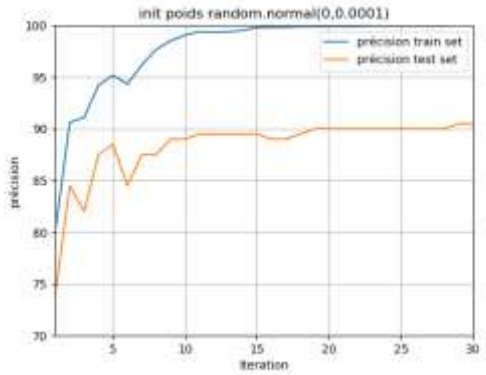
3. L2 régularisation

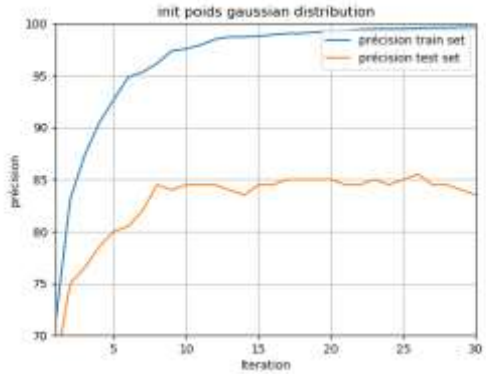
Une autre technique pour prévenir le surentrainement c'est la décadence des poids (ou L2 régularisation). Cela consiste a pénalisé les poids qui sont grand en fonction du cout du réseau, ce qui a pour effet d'empêcher le réseau d'apprendre des spécificités des données d'entrainement. Et ainsi régularisé ce qu'il apprend au test set.
Learning rate=0.05, lambda=5,alpha=0.9

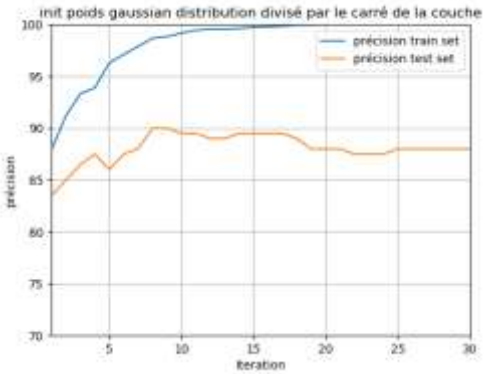
On peut voir que comme avec le dropout la différence entre le train set et test set est plus faible que sans régularisation.

7. Initialisation des poids

Etant donné que je normalise les données l'Initialisation des poids est plus facile. Une technique que courante consiste random.randn(y, x) (gaussien distribution) et de divisé par la racine carré de la couche précédente.







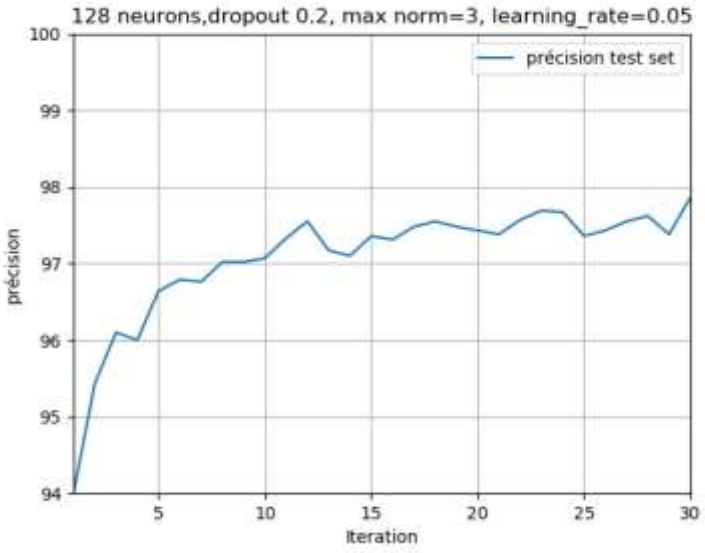
On peut voir que l'initialisation permet d'avoir de meilleur résultat plus rapidement. Dans le troisième graphe on voit que déjà à la première itération on obtient presque 85% alors que le deuxième atteint à peine 85%. Et dans le premier graphe est seulement à 75% après la première itération. L'initialisation permet surtout d'accéléré le début de l'entrainement.

8. Améliorations graphique

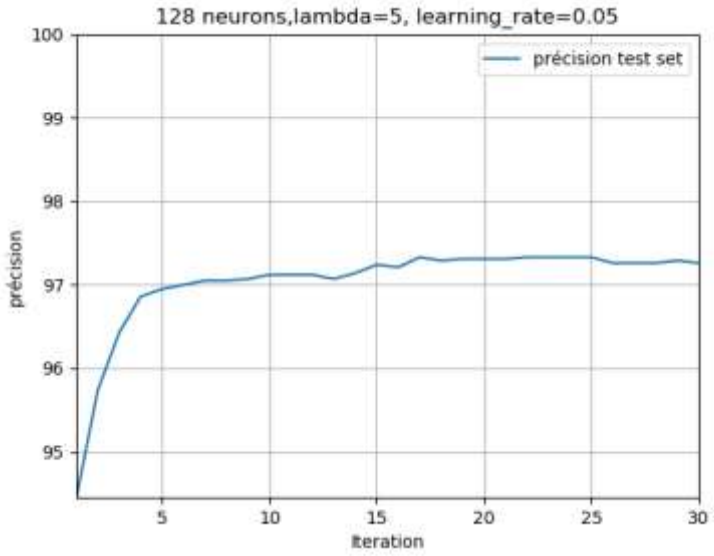
Avant de lancer l'entrainement avec le buttons Start, vous pouvez choisir un dataset, la taille, la portion du dataset qui va servir à l'entrainement, le nombre de couche intermédiaire, le nombre de neurone qui seront reparti entre les différentes couche de façon équitable (exemple : 2 couches, et 60 neurones va donner un réseau avec 30 neurones par couche intermédiaire), la probabilité de chaque neurones de dropout (sauf la couche input qui aura 0,2 de prob de dropout), la contrainte de la norme maximal des poids, lambda le facteur de régularisation L2, le learning rate et le nombre d'itération de l'entrainement.
Dans la premier tab, on peut choisir de suivre le coût et la précision lors de l'entrainement du réseau sur le train set et le test set. Grace à ces informations on peut ajuster les paramètres du réseau pour un prochain entrainement.
Dans la deuxième tab, une fois que l'entrainement est terminé on peut voir les images du dataset sélectionné.
Dans la troisième tab, aussi une fois que l'entrainement est terminé on peut voir la visualisation des poids entre la couche d'entre et la 1er couche intermédiaire. Seulement si le nombre de neutrons des couches intermédiaires forme un carre parfait alors on peut visualiser les poids entre les couche intermédiaire et la couche de sortie.

9. Conclusion

En conclusion, les réseaux neuronaux sont des programmes qui demandent une grande précision dans les choix d'implémentation et des paramètres pour utiliser son potentiel. Mais malgré cela les résultats que j'ai été capable d'atteindre sont très impressionnant. En modifiant les paramètres et avec dropout et max norme on peut atteindre 97.86% avec juste 30 itérations.



Et aussi sans dropout avec L2 régularisation 97.33%.



Avec plus de puissance de calcul on pourrait trouver des meilleurs paramètres. Les meilleures performances sont réalisées avec des réseaux de plusieurs couches avec des milliers de neurones.
Pour finir, ce projet a été très intéressant autant dans l'implémentation en python des algorithmes que l'interface graphique et surtout de naviguer dans les tonnes d'informations et d'application qu'apport les réseaux neuronaux. De plus, ces dernières années on voit surgir beaucoup d'utilisation des réseaux. Maintenant la plupart des entreprises technologique tell que les voitures autonomes de Telsa ou la reconnaissance d'image de Google sont des applications « avancé » des réseaux.