Dropout pour la régularisation de réseaux de neurones

Emmanuel NOUTAHI
Laboratoire de Biologie Informatique et Théorique (LBIT)
Département d'Informatique et de Recherche Opérationnelle (DIRO)
Université de Montréal
fmr.noutahi@umontreal.ca

Abstract

Les réseaux de neurones constituent une classe d'algorithmes d'apprentissage très robustes inspirés des neurones biologiques. Toutefois, comme la plupart des algorithmes d'apprentissages, ils n'échappent pas au phénomène de sur-apprentissage et nécessitent donc l'utilisation de méthodes de régularisation telles que *early stopping, weight decay* et depuis peu *dropout*. Dropout consiste à abandonner de façon stochastique les neurones des différentes couches du réseau lors de l'apprentissage avec une certaine probabilité p. Bien que plusieurs expériences aient montré l'efficacité de la méthode pour $p = 0.5$, les autres variations de la technique ont été peu explorées. Afin de déterminer l'efficacité de dropout et de ses différentes variations, nous avons évalué la méthode sur un extrait des données de MNIST. Nos résultats montrent que dropout fonctionne bien lorsque $p \leq 0.6$ sur les couches cachées et confirment que la désactivation de moins de 50% des neurones en entrée pourrait également améliorer la prédiction.

1 Introduction

Les réseaux de neurones sont des modèles puissants d'apprentissage machine avec une large gamme d'applications qui s'étendent de la reconnaissance d'objets (visage, langages naturels, forme syntaxique) à la bio-informatique (classification de séquence d'ADN, de cancer, prédiction de fonction de protéines).

Les réseaux de neurones peuvent contenir plusieurs couches cachées avec des activations non-linéaires, ce qui leur permet d'apprendre des relations complexes entre les entrées et les sorties [1]. Toutefois, comme la plupart des algorithmes d'apprentissage, les réseaux de neurones sont sujets au sur-apprentissage, particulièrement lorsque les données disponibles pour l'entraînement sont limitées. Plusieurs méthodes ont été développées pour résoudre le problème de sur-apprentissage. On peut ainsi arrêter l'entraînement dès que l'erreur de prédiction sur les données de validation empire ou faire recours aux poids de régularisation L1 et L2.

L'un des meilleurs moyens de régularisation est la combinaison de modèles. Ainsi, pour un modèle à capacité finie, il faut donc prendre l'espérance mathématique des prédictions obtenues à partir de différents paramètres du modèle en se basant sur une approche bayésienne [2]. Cependant, cette méthode bien que très performante est computationnellement intensive, puisqu'elle nécessite l'entraînement de plusieurs modèles avec optimisation de leurs hyper-paramètres. Une alternative récemment proposée est $dropout$.

Dropout empêche le sur-apprentissage et combine plusieurs architectures de réseaux de neurones, en retirant temporairement des unités du réseaux (Figure 1), ainsi que leurs connections [3][4]. Les versions décrites de dropout consistent à mettre l'activation des neurones d'une couche à 0 avec un probabilité p. Cette probabilité peut être estimé avec des données de validation, mais en

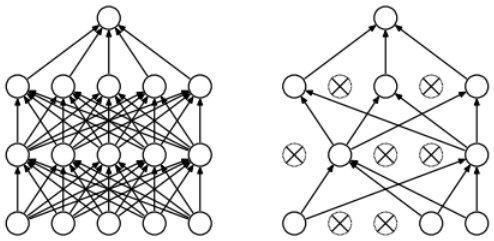


Figure 1: Dropout dans un réseau de neurones [3]. Le réseau décrit ici possède deux couches cachées. À droite, on peut noter l'impact de dropout sur le réseau avec p=0.5

règle générale, elle est fixée à 0.5. Il est également possible d'appliquer dropout sur la couche d'entrée en s'inspirant des travaux récents sur l'addition de bruits aux entrées des auto-encodeurs [5]. L'activation d'un neurone j de la couche $l+1$ devient alors:

$$a_j^{l+1} = m^{l+1} h(\sum_i w_{ji}^l a_i^l)$$

avec w_{ji}^l le poids entre les neurones i et j, h la fonction d'activation et $m^{l+1} \sim \text{Bern}\left(1 - p^{l+1}\right)$, p^{l+1} étant la probabilité de désactivation des neurones de la couche $l+1$.

Lors de la phase de prédiction, tous les neurones sont présents et les poids entre chaque neurone i^l et un neurone j^{l+1} de la couche suivante sont multipliés par ${p_i}^l$, la probabilité de désactivation du neurone i^l précédemment utilisée.

Dans le cas d'un réseau avec une seule couche cachée, prendre $p = 0.5$ pour dropout sur la couche cachée donne exactement en sortie la moyenne de prédiction pour les différentes architectures entraînées. Ce n'est toutefois pas le cas lorsque le réseau possède plusieurs couches. Il est donc important de choisir les probabilités adéquates pour obtenir des résultats optimaux.

Dans les prochaine sections, nous comparerons l'efficacité de dropout par rapport à celle d'une simple régularisation L2, ainsi que l'effet de la variation de la probabilité de désactivation des neurones des couches d'entrée et cachées.

2 Résultats expérimentaux

Afin d'évaluer l'efficacité de dropout, nous avons entraîné plusieurs modèles de réseaux de neurones pour différentes variations de dropout sur les données de MNIST.

2.1 Modèle et dataset

Nous avons utilisé les images de MNIST. Il s'agit d'un dataset de chiffres écrits manuellement comprenant 60000 images d'entraînement et 10000 de tests. La taille en pixels de chaque image est 28 x 28 et l'intensité des pixels est compris en 0 et 255. Pour notre entraînement, nous avons d'abord transformé chaque image en un vecteur de taille 28 x 28 =784. Chacun de ces vecteurs est ensuite normalisé en divisant chaque élément du vecteur par 255.

Nos tests ont été effectué sur plusieurs modèles de réseaux de neurones avec un nombre de couches et d'unités cachées variables s'approchant pour la plupart de ceux utilisés dans l'article originale de dropout. En raison de l'intensité des calculs requis pour tester plusieurs modèles, nous avons fixé la plupart des hyper-paramètres nos modèles. Le taux d'apprentissage η, par exemple est diminué à chaque époque i d'entraînement et est défini :

$$\eta_i = \frac{\eta_0}{1 + decay * i}$$

avec $decay$ fixé à 0.1 et η_0 à 1.

Pour la fonction d'activation des neurones cachés, nous avons choisi le Rectificateur de linéarité (ReLU) qui semble donner des résultats assez satisfaisant lorsque combiné à dropout [4]. Enfin, la classification est effectuée en utilisant une softmax sur les données de la dernière couche. Le code est écrit en python et utilise la librairie theano [1].

2.2 Efficacité de dropout

Pour tester l'efficacité de dropout par rapport à la régularisation L2 (*weight decay*), nous avons initialement prévu d'effectuer une validation croisée pour optimiser la valeur de l'hyper-paramètre L2 (entre : 10^{-2},10^{-3}, 10^{-4} et 10^{-5}) en divisant aléatoirement notre ensemble d'entraînement en 50000 données d'entrainement et 10000 données de validation. Toutefois, des limites computationnelles nous ont contraint à limiter la taille de nos données à uniquement 5000 données d'entrainements et 1000 données de test choisies au hasard. L'architecture utilisé est 784-512-512-10 soit 784 neurones pour la couche d'entrée, 512 pour les deux couches cachées et 10 classes en sortie.

Table 1: Erreur de prédiction de chaque méthode sur les données de tests. Dropout + L2 donne les meilleurs résultats.

méthode	**erreur sur test**
Dropout	5.6%
Dropout + L2	6%
L2	5.8%

Les résultats obtenus montrent que dropout performe mieux qu'une simple régularisation L2, supportant ainsi les résultats de l'article originale de dropout [3]. Nos taux de prédictions sont toutefois très différents, puisque nous utilisons un sous-ensemble réduit des données de MNIST. Contre toute attente, dropout seul, performe mieux que la combinaison des deux méthodes.

Ces résultats confirment l'efficacité de dropout mais doivent être pris avec précaution, puisque les différents modèles ne sont pas forcément optimaux et le nombre d'époques d'entraînement est réduit à 400 en raison de limites computationnelles.

2.3 Effet sur l'apprentissage des features

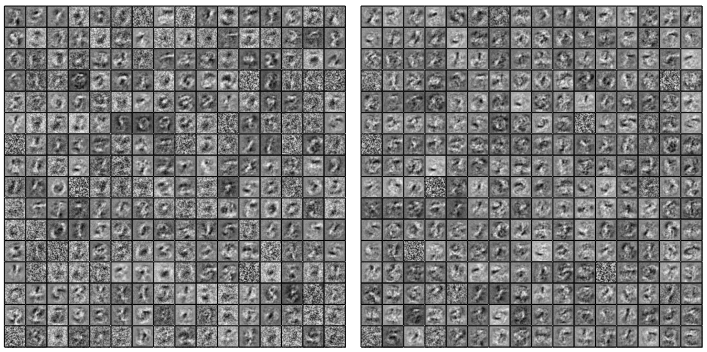
Dans un réseaux de neurone, les poids entre les neurones de deux couches dictent comment les neurones de la couche ultérieure doivent s'adapter lors de l'apprentissage. Pour déterminer l'effet de dropout sur ces poids, nous avons utilisé une architecture à une seule couche cachée avec 256 neurones (764-256-10). La Figure 2 présente les poids W^1 entre l'entrée et la couche cachée du réseau après entraînement sur 2000 images issues de MNIST.

Comme le montre les Figure 2a et Figure 2b, dropout réduit la co-adaptation entre différents neurones, forçant ainsi chaque neurone à apprendre les features des données en entrées sans trop dépendre des autres neurones. Ceci s'observe par la présence de plus de neurones avec des motifs structurés au niveau de la Figure 2b. L'effet est encore plus remarquable lorsque dropout est utilisé sur la couche d'entrée conjointement avec la régularisation L2 (Figure 2d). Il est toutefois important de noter qu'en abandonnant un grand nombre de neurones de la couche d'entrée, le modèle aura de la difficulté à apprendre les relations entre les pixels des images, ce qui pourrait causer un sous-apprentissage.

2.4 Variation de la probabilité de désactivation des neurones

Afin d'évaluer l'impact des variations de la probabilité p de désactivation des neurones d'une couche cachée sur l'apprentissage, nous avons entraîné un modèle de réseau de neurone (784-512-512-10) en utilisant 2000 images d'entrainement de MNIST et 1000 images de test. Pour chaque couche

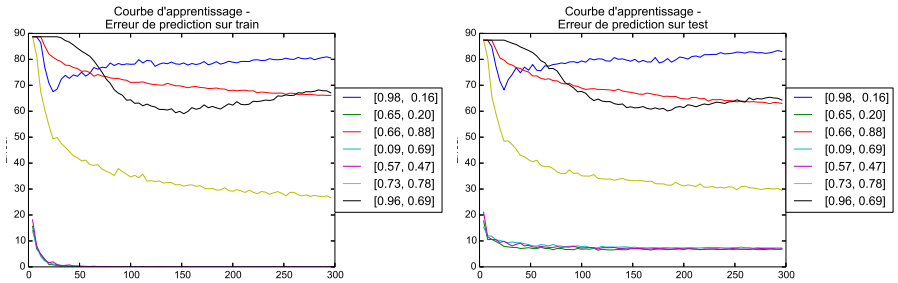
[1] https://github.com/maclandrol/dropout



(a) Sans dropout (b) Dropout: $p_{in} = 0$, $p_{hidden} = 0.5$

(c) Dropout: $p_{in} = 0.5$, $p_{hidden} = 0.5$ (d) Dropout: $p_{in} = 0.5$, $p_{hidden} = 0.5$ + L2

Figure 2: Effet de dropout sur les poids de la couche cachée dans un reseau avec pour architecture initiale 784-256-10 sur les données de MNIST.



(a) Entrainement (2000 images) (b) Test (1000 images)

Figure 3: Erreur de prédiction d'un réseau 784-512-512-10 entrainé sur données de MNIST avec dropout comme régularisation, pour des p variables. Le label indiqué pour chaque courbe, est [p_{hid1}, p_{hid2}]

cachée l, La probabilité p_l d'abandonner les neurones est tirée de $U[0-1]$ où U désigne la loi uniforme, tandis que la probabilité de désactivation des neurones de la couche d'entrée est fixée à 0.

Comme on aurait pu s'y attendre, la Figure 3 montre qu'en règle générale, dropout semble fonctionner mieux lorsque les probabilités p_{hid} sont faibles, en particulier $p_{hid} < 0.7$. Il est aussi intéressant de noter que les résultats de prédiction sont désastreux lorsque la probabilité de désactivation des neurones d'une couche est très élevée ($p_{hid} > 0.9$), alors qu'elle est faible pour les neurones de la couche suivante.

Nous avons ensuite considéré l'architecture 784-512-1024-10 et fait varier p_{hid} entre [0.2, 0.4, 0.6, 0.8, 1], p_{hid} étant identique sur toutes les couches (Figure 4). Comme le suggèrent déja les résultats précédents, les meilleurs résultats de prédiction sont obtenus pour $p_{hid} \leq 0.6$, indiquant en outre la perte de performance lorsque plus de 60% des neurones sont désactivés sur chaque couche.

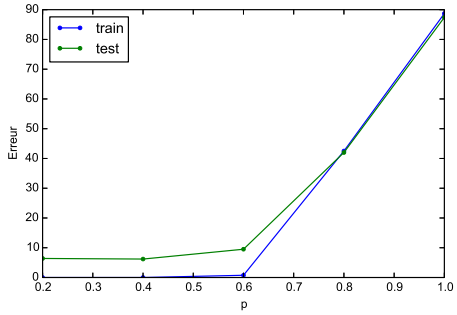
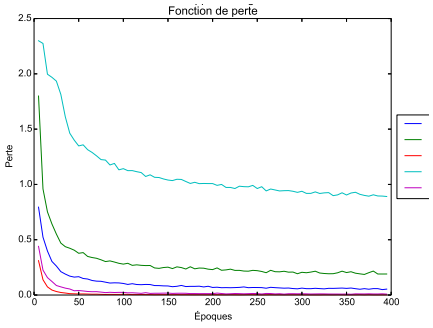


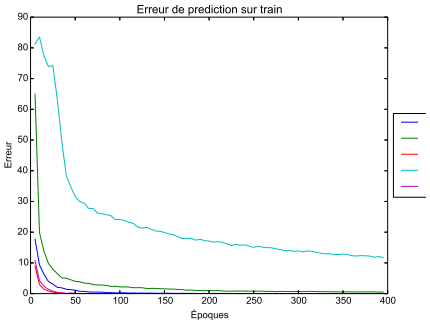
Figure 4: Variation de l'erreur de prédiction en fonction de différentes valeurs de p_{hid}

2.5 Dropout sur les entrées

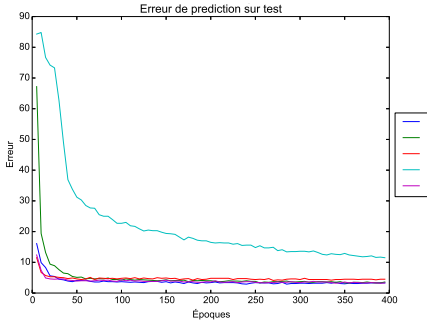
Pour évaluer l'impact de dropout sur les entrées, nous avons entraîné un modèle à trois couches cachées (784-512-512-1024-10) dont la probabilité de désactivation des neurones des couches cachées est fixée à 0.5 tandis que nous faisons varier cette probabilité entre [0, 0.2, 0.5, 0.7, 0.9] pour la couche d'entrée (Figure 5). L'entraînement s'est fait avec 5000 images et les tests avec 1000.



(a) Fonction de perte-train



(b) Erreur de prediction-train



(c) Erreur de prediction-test

Figure 5: Effet de dropout sur la couche d'entrée. Les probabilités utilisées pour chaque courbe sont respectivement : **Cyan-0.9, Vert-0.7, Bleu-0.5, Violet-0.2 et Rouge-0**

Nos résultats montrent que le nombre d'époques nécessaire pour la convergence de SGD augmente lorsque la probabilité de désactivation des neurones de la couche d'entrée est grande (> 0.5). Ceci s'illustre aussi par la comparaison des fonctions de perte sur les données d'entraînement (pertes plus élevées lorsque p_{in} est plus grand). L'erreur sur la prédiction est donc plus élevée lorsque p_in est très grand.

Ces résultats montre également que pour $p_{in} = 0.2$ et $p_{in} = 0.5$, nous obtenons un taux de prédiction légèrement meilleur, par rapport à l'absence totale de dropout sur la couche d'entrée.

Ainsi, il serait possible d'améliorer les résultats de la prédiction avec dropout, en désactivant quelques dimensions (pixels) des entrées. Cette méthode semble fonctionner le mieux pour des valeurs de $p_{in} \leq 0.5$.

3 Conclusion

Nous avons testé dropout sur des données issues de MNIST et montré son efficacité. En règle générale choisir une probabilité de désactivation des neurones des couches cachées 0.5 ± 0.2 semble être efficace. Il est également possible d'appliquer le principe sur la couche d'entrée et des valeurs plus faible (≤ 0.5) semble bien fonctionner. Cependant, il est possible que nos résultats bien que confirmant les paramètres généralement utilisés pour dropout ne soient valables que pour nos données. Nous pensons qu'une meilleure méthode pour choisir les hyper-paramètres est la validation croisée. Toutefois, cette méthode fait perdre le gain en temps de calcul de dropout et annule de ce fait l'avantage conféré par la méthode. Une méthode d'amélioration possible de dropout pourrait être d'utiliser différentes probabilités de désactivation pour les neurones d'une même couche, ou de choisir ces probabilités dépendamment des neurones qui leurs sont connectés.

4 References

[1] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *Nature* 323, 533 (1986).

[2] Salakhutdinov & A. Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo.*In Proceedings of the 25th International Conference on Machine Learning*. ACM, (2008).

[3] SRIVASTAVA, Nitish, HINTON, Geoffrey, KRIZHEVSKY, Alex, et al. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Researc*h, vol. 15, no 1, p. 1929-1958, (2014).

[4] HINTON, Geoffrey E., SRIVASTAVA, Nitish, KRIZHEVSKY, Alex, et al. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[5] P.Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *In Proceedings of the 27th International Conference on Machine Learning*, pages 3371–3408. ACM, 2010.