Impact des fonctions d'activation dans les réseaux de neurones profonds

Matthieu SPEISMANN - 2409862
Jacob ST-GERMAIN - 2162654
Génie Informatique – Polytechnique Montréal

Abstract

Les fonctions d'activation jouent un rôle central dans la capacité des réseaux de neurones profonds à modéliser des relations complexes et le plus souvent non linéaires. Alors que des fonctions classiques comme la sigmoïde ou Tanh ont longtemps été utilisées, des approches plus récentes comme ReLU et ses variantes (Leaky-ReLU, PReLU, …) ainsi que des fonctions adaptatives comme Swish ou Mish ont démontré des performances supérieures dans certains contextes d'utilisation.

Cet article propose une étude comparative de l'impact de ces fonctions sur l'apprentissage profond, en s'appuyant à la fois sur une revue de littérature scientifique et sur une analyse expérimentale.

1 Introduction

Dans le domaine de l'apprentissage profond, les fonctions d'activation jouent un rôle fondamental dans la capacité des réseaux de neurones à apprendre des représentations complexes et non linéaires. Celles-ci permettent notamment de rompre la linéarité des couches empilées dans les réseaux et rendent possible l'apprentissage de modèles riches et expressifs. Sans elles, même les réseaux profonds les plus sophistiqués ne pourraient être que de simples combinaisons linéaires.

Historiquement, des fonctions comme **la sigmoïde** ou **Tanh** ont été très utilisées, notamment en raison de leur continuité et de leur capacité à normaliser les sorties. Cependant, ces fonctions souffrent d'un temps de calcul relativement important. C'est dans ce contexte qu'est apparue la **fonction ReLU (Rectified Linear Unit)**, devenue aujourd'hui une référence dans le domaine, en raison de sa simplicité, de sa rapidité de calcul et de ses performances généralement supérieures.

Depuis, de nombreuses variantes de ReLU ont vu le jour, telles que **Leaky ReLU, PReLU ou eLU**, visant à corriger certaines de ses limitations, notamment l'inactivité des neurones pour les valeurs négatives, qui ralentit l'apprentissage par rétropropagation. Par ailleurs, des fonctions plus récentes comme **Swish**, combinant la linéarité des valeurs positives de ReLU et la souplesse de la sigmoïde, proposent une approche hybride prometteuse. Ces fonctions ont été l'objet de recherches approfondies, aussi bien théoriques qu'empiriques, comme en témoignent les travaux de Dubey et al. (2022) et Kulathunga et al. (2020), qui analysent leur comportement selon la complexité du réseau et le type de données utilisé.

L'objectif de ce travail est de **comparer expérimentalement différentes fonctions d'activation**, classiques et modernes, à partir d'un modèle simple appliqué à la classification d'images manuscrites du jeu de données MNIST. On comparera également les résultats obtenus avec le comportement théorique attendu de chaque fonction. Cette étude vise finalement à illustrer les effets pratiques de chaque fonction sur les performances du modèle, tant en termes de précision que de rapidité d'apprentissage, afin de mieux comprendre les forces et les faiblesses de chacune.

2 Catalogue des fonctions d'activation et effets théoriques attendus

2.1 ReLU et ses dérivés

La fonction **ReLU** est une fonction d'activation standard car elle introduit de la non linéarité et est rapide à calculer:

$$ReLU(x) = max(0, x) \ = \ x \, si \, x \ > \ 0 \text{ et } 0 \, si \, x \ < \ 0$$

Cependant, pour les valeurs négatives, la fonction ReLU ne permet pas une propagation efficace du gradient car celles-ci deviennent nulles.

En définissant donc arbitrairement un paramètre $\alpha \ \in]0, 1[$, on obtient la fonction **Leaky ReLU** qui conserve la non-linéarité de ReLU et une simplicité de calcul avec la définition suivante.

$$LeakyReLU(x) = max(\alpha x \, , x)$$
$$= x \, si \, x \ > \ 0 \text{ et } \alpha x \ si \, x \ < \ 0$$

Il est également possible d'apprendre automatiquement le paramètre α lors de l'entraînement du modèle (comme un poids du modèle supplémentaire). Cette fonction est nommée **Parametric ReLU (PReLU)**.

Elle a l'avantage d'être adaptative aux données d'entraînement et peut donc améliorer les performances spécifiquement à ces données. Il faut cependant rester attentif au risque accru de sur-apprentissage dans ce cas.

Enfin, il est possible d'ajouter encore plus de non linéarité avec **l'exponential LU (eLU)** définie ainsi, avec un paramètre α arbitrairement fixé:

$$eLU(x) = max\left(\alpha(e^x - 1) \, , x\right)$$
$$= x \, si \, x \ > \ 0 \text{ et } \alpha(e^x - 1) \ si \, x \ < \ 0$$

Cette fonction permet de contrôler les sortie négative puisqu'elles décroît exponentiellement vers $-\ \alpha$..

Ici, l'inconvénient majeur réside dans le coût plus important de cette fonction d'activation. Il existe aussi un risque important de saturation négative vers $-\ \alpha$ qui peut rendre l'apprentissage plus lent.

2.2 Sigmoid et ses dérivés

La fonction **sigmoïde** est une autre fonction d'activation standard, qui elle aussi introduit de la non-linéarité et qui a un relativement faible coût de calcul. De plus, ses sorties sont contrôlées entre 0 et 1:

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

La fonction **Swish** utilise la fonction sigmoïde et un paramètre β fixé (le plus souvent égal à 1) et est définie ainsi:

$$Swish(x) \ = \ x.\sigma(\beta x) \ = \ \frac{x}{1+e^{-\beta x}}$$

Cette fonction d'activation a beaucoup de caractéristiques intéressantes. En effet, avec $\beta \ = \ 1$, on a les approximations suivantes:

$$Swish(x) \ = \ x \text{ si } x \ > \ 0$$
$$Swish(x) \ = \ 0 \text{ si } x \ < \ 0$$

Le comportement de Swish est donc proche de celui de la fonction ReLU, avec l'avantage d'être différentiable en 0, ce qui facilite grandement l'optimisation lors de l'apprentissage du modèle.

Son caractère non monotone permet également de modéliser des relations plus complexes entre les données d'apprentissage. Son inconvénient majeur est quant à lui son coût de calcul bien supérieur à ReLU.

3 Expérimentation de ces fonctions d'activation

3.1 Le modèle utilisé

Pour tester les différentes fonctions d'activations, nous avons construit un réseau de neurones dans le but de classifier les données MNIST. Ce jeu de données contient des images de tailles 28 x 28 qui représente les chiffres de 0 à 9 écrit de manière manuscrite. On souhaite donc les classer en fonction du chiffre qui est écrit.

On a séparé de manière aléatoire les données MNIST en un ensemble d'entraînement (qui comporte 85% des données) et un ensemble de validation (qui comporte 15% des données).

A noter que pour l'apprentissage, on utilise l'optimiseur Adam sur des batchs de taille 128. Pour toute la suite de l'étude, voici le modèle utilisé:

- **La couche d'entrée** de taille 784 (28x28) pour accueillir les valeurs en niveau de gris de chaque pixel des images.

- **Une couche cachée** de 256 neurones, dont les liaisons avec la couche d'entrée est complète, mais subit un dropout aléatoire de 30% (pour éviter un sur-apprentissage sur des pixels spécifiques). La couche est activée par la fonction d'activation à tester.

- **Une autre couche cachée** de 128 neurones, elle aussi activée par la même fonction d'activation à tester. Elle est relié de manière complète à la couche précédente, cette fois sans dropout.

- **Une couche de sortie** avec 10 neurones pour la classification, activée par la fonction softmax.

3.2 Comparaison expérimentale des fonctions ReLU et dérivés

La première chose à étudier est l'impact du paramètre α utilisé par la fonction LeakyReLU. Pour la suite, on fixera une valeur de α pour comparer LeakyReLU aux autres fonctions d'activation.

3.2.1 Etude de l'impact du paramètre α sur les performance de LeakyReLU

On applique donc à toutes les couches cachées la fonction d'activation LeakyReLU avec des valeurs de $\alpha \in \{0, 0.01, 0.1, 0.2, 0.5, 0.8, 1\}$. A noter que lorsque $\alpha = 0$ on utilise en fait le ReLU standard et lorsque $\alpha = 1$, on utilise simplement l'identité linéaire. On obtient donc les résultats présentés sur le graphe 1 (ci-dessous)

On remarque alors que pour les faibles valeurs de α (jusqu'à 0.2) les performances sont sensiblement similaires et satisfaisantes. Ensuite, lorsque α augmente davantage, les performances des modèles concernés diminuent fortement. Cela peut s'expliquer par le fait que des valeurs de α proches de 1 ou égal à 1 modélisent des relations presque voire parfaitement linéaires, ce qui serait insuffisant pour classifier les chiffres de MNIST

Dans la suite, **nous utiliserons** $\alpha = 0.1$ **lors de l'utilisation de LeakyReLU**. Ce choix est fait afin de conserver des performances satisfaisantes tout en ayant une valeur de α suffisamment élevée pour distinguer LeakyReLu et ReLU standard.

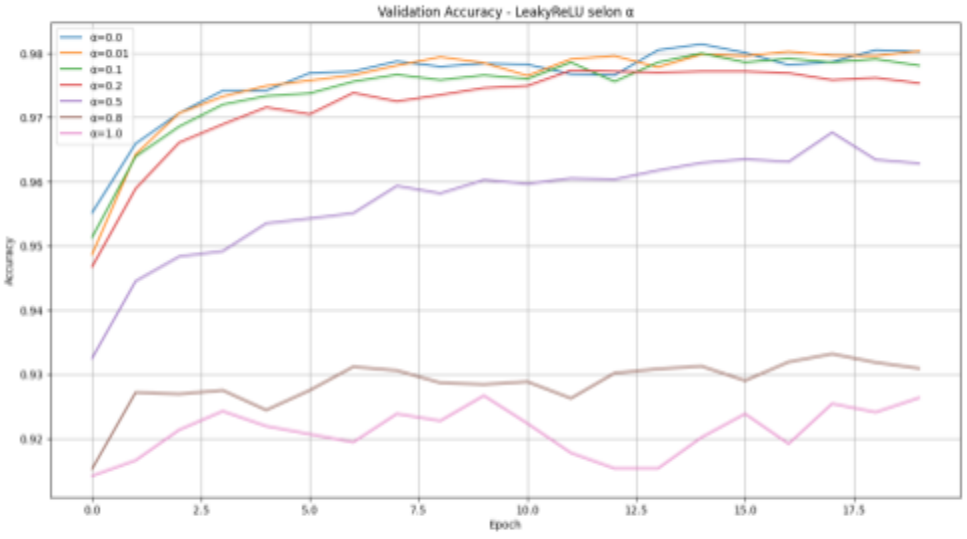
3.2.2 Comparaison des performances des fonctions d'activation ReLU (et dérivés)

En appliquant chaque fonction d'activation de type ReLU, tour à tour sur les couches cachées du modèle défini précédemment, on obtient cette fois-ci les graphes 2 et 3 (voir page suivante). On y remarque alors que:

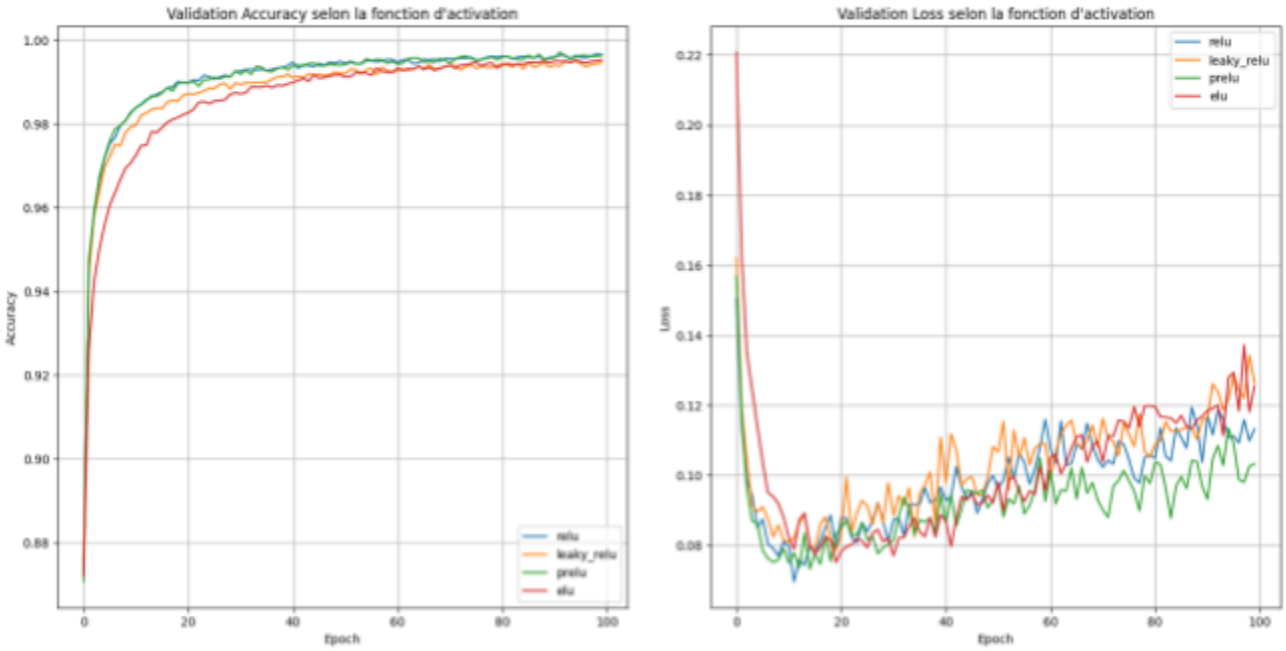
- **ReLU et PReLU ont des performances quasiment identiques et supérieures aux autres fonctions** dans cet exemple. Cependant on remarque que PReLU semble moins sur apprendre que ReLU (comme en témoigne la courbe de perte plus basse)

- **LeakyReLU est un peu moins bon** (son entraînement étant plus lent), ce qui est normal vis à vis de PReLU, puisque ce dernier effectue exactement le même calcul, mais en adaptant la valeur du paramètre aux données d'entraînement présentées.

- **eLU est quant à elle bien moins bonne que les autres fonctions étudiées**. Cela peut se comprendre par la simplicité de l'exemple étudié, qui n'est pas adapté à des relations plus complexes exprimées par des exponentielles. De plus, la saturation des valeurs négative vers $-\alpha$ peut également jouer un rôle



Graphique 1



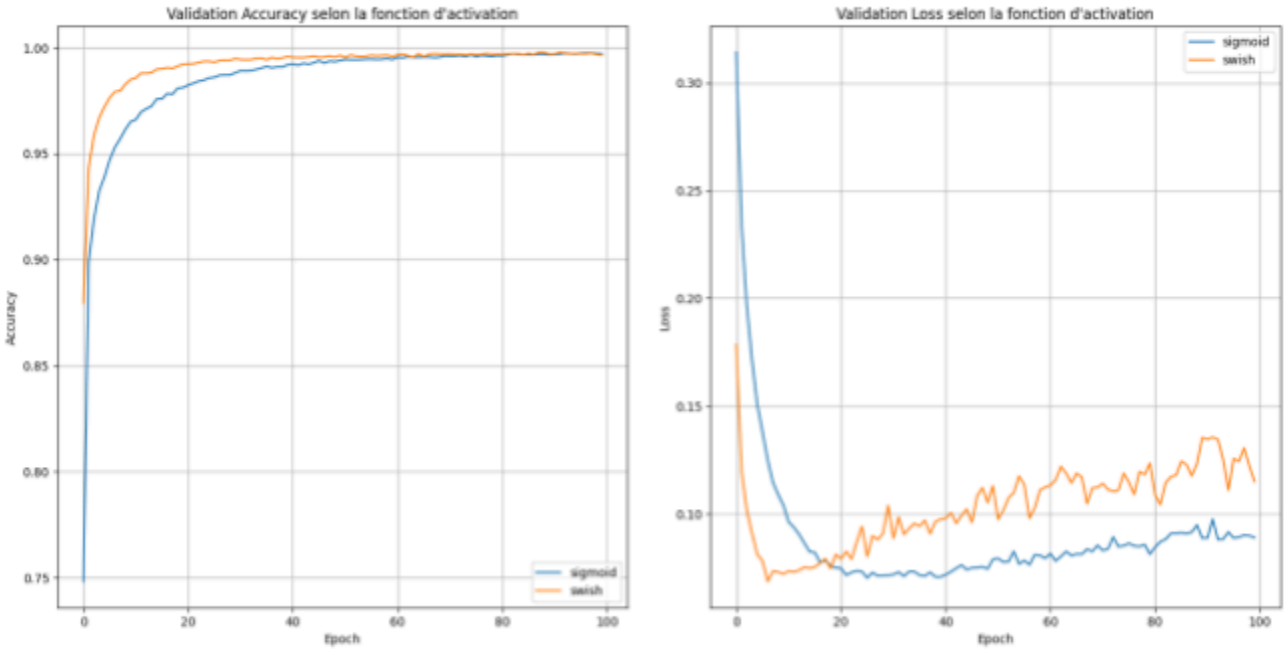
Graphiques 2 et 3

3.2.3 Comparaison du coût des fonctions d'activation ReLU (et dérivés)

On a également mesuré le temps d'apprentissage du modèle selon les différentes fonctions d'activation utilisées sur les couches cachées:

Fonction d'activation	Temps d'entraînement (s)
ReLU	171.1
LeakyReLU	181.4
PReLU	185.7
eLU	182.6

- On remarque tout d'abord que les temps d'apprentissage sont tous dans les mêmes ordres de grandeur et les différences entre eux ne dépassent pas 10%.
- Cependant, on remarque que **la plus rapide est ReLU**. Cela est normal puisqu'il s'agit de la fonction d'activation la plus simple et ne nécessite pas de calcul (soit il vaut 0, soit il prend la valeur de x).
- **Suivent, respectivement LeakyReLU et eLU**. A nouveau ce résultat est attendu car elles calculent une nouvelle valeur dans le cas où x est négatif (respectivement un produit et une exponentielle).
- Enfin, **PReLU est le plus lent**. A nouveau, cela est conforme aux attentes car cette fonction est adaptative et rajoute donc un poids d'entraînement au modèle.



Graphiques 4 et 5

3.3 Comparaison expérimentale des fonctions sigmoid et dérivés

3.3.1 Comparaison des performances des fonctions d'activation sigmoid(et dérivés)

En appliquant chaque fonction d'activation de type ReLU, tour à tour sur les couches cachées du modèle défini précédemment, on obtient cette fois-ci les graphes 4 et 5 (ci-dessus). On y remarque que **la fonction Swish est sensiblement plus efficace que la fonction sigmoid.** En effet, celle-ci apprend plus rapidement comme en témoignent les courbes de validation et de perte des deux modèles.

Cependant, **il semble également que la fonction swish soit davantage soumise au sur-apprentissage** que la fonction sigmoïde, puisque sa courbe de perte augmente bien plus que la sigmoid au-delà des 10 époques.

3.3.2 Comparaison du coût des fonctions d'activation sigmoid (et dérivés)

Voici également les résultats obtenus en mesurant le temps d'entraînement avec chaque fonction d'activation:

Fonction d'activation	**Temps d'entraînement (s)**
Sigmoide	209.6
Swish	223.4

La première remarque à faire est que **les temps d'entraînement des modèles avec des fonctions d'activation de type sigmoïde sont supérieurs à ceux du type ReLU** (respectivement +22% et +30% par rapport à ReLU) Ce comportement s'explique par le fait qu'il faut calculer de nouvelles valeurs, basées sur des exponentielles pour **toutes des valeurs de x** (là ou pour les fonction type ReLU, c'était le cas au pire uniquement pour les valeurs négatives)

De plus, **on note que la fonction Swish est longue à calculer que la fonction sigmoïde**. Cette fois, l'explication peut résider dans la fonction à calculer, légèrement plus complexe (avec un produit et une division de plus par rapport à la simple sigmoïde)

4 Discussions

4.1 Considérations sur le contexte

Il est important de rappeler, comme le faisaient les articles qui servent de référence à cette étude, que **le contexte d'utilisation du réseau de neurones est essentiel dans le choix de la fonction d'activation**. En effet, les performances décrites ici (dans le cadre d'une classification automatique) pourraient être très différentes si l'on souhaite réaliser d'autres tâches, par exemple de la génération de texte ou d'images.

Aussi, **la profondeur du réseau utilisé ainsi que la taille du jeu de données sont d'autres paramètres importants à prendre en compte**. En effet, dans notre étude, ces deux critères sont relativement faibles, mais dans l'article de Kulathunga et al. [2], deux jeu de données riches et des réseaux plus importants ont permis de mettre davantage en lumière les performances de Swish et PReLU, mais a également souligné leur temps de calcul élevé en contrepartie.

4.2 Limites identifiées de l'étude et piste d'approfondissement envisagées

Notre étude s'est intéressée à un certain nombre de fonctions standard et leurs dérivées, mais il reste un certain nombre de fonctions d'activation que nous n'avons pas testées (PeLU, Mish, GELU, APL, FReLU, …). Ces fonctions plus récentes, mais également plus spécifiques, pourraient proposer des performances plus élevées encore selon le contexte de l'étude.

Nous avons également étudié ces fonctions sur l'unique jeu de données MNIST. On pourrait vérifier la robustesse des résultats ici énoncés en testant les mêmes fonctions d'activation sur d'autres jeu de données pour la classification (Fashion-MNIST, CIFAR10 ou des données non visuelles comme Iris par exemple).

De même, il serait intéressant de confronter ces résultats avec ceux obtenus sur d'autres tâches traitées par réseaux de neurones (NLP, image génération, segmentation, ...) en vue de les généraliser, ou d'en spécifier la portée.

References

[1] Shiv Ram Dubey, Satish Kumar Singh and Bidyut Baran Chaudhuri. Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark. June 2022.

[2] Nalinda Kulathunga, Nishath Rajiv Ranasinghe, Daniel Vrinceanu, Zachary Kinsman, Lei Huang and Yunjiao Wang. Effects of the Nonlinearity in Activation Functions on the Performance of Deep Learning Methods, October 2020