La rétropropagation de l'erreur avec gradient substitué pour la classification des caractères manuscrits

"Nous attestons que nous sommes les auteurs du présent travail et que tout ce qui a été emprunté est attribué à sa source et proprement référencé."

Yahya Ettahi (22h), Yasmine Wahbi (25h), Sidi Yaya Arnaud Yarga (20h)

GEI723 - Problème III - Automne 2020

INTRODUCTION ET CONTEXTE

Les réseaux de neurones à décharges s'inspirent du comportement des neurones biologiques pour les modéliser et offrent de nouvelles opportunités pour atteindre l'objectif de l'intelligence artificielle en se basant sur l'intelligence humaine. Ce projet est basé sur la méthode d'apprentissage supervisé pour la classification d'images qui utilise bien la rétropropagation d'erreur. Pour se faire, l'usage du notebook d'Ismaël Balafrej et Ahmad El Ferdaoussi ainsi que la bibliothèque PyTorch de Python et l'environnement Google Colab est requis.

OBJECTIFS

- Comprendre l'impact du choix de la forme de la dérivée associée à une décharge dans l'algorithme de rétropropagation de l'erreur
- Savoir résoudre et implémenter les équations différentielles qui caractérisent un réseau de neurones;
- Pouvoir étudier et analyser l'impact des méta-paramètres sur les résultats;
- Pouvoir réaliser l'architecture d'un réseau de neurones à décharges avec apprentissage par rétropropagation de l'erreur pour une application en classification;
- Être capable de faire le lien entre la littérature scientifique proposée dans ce problème et l'implémentation informatique en langage python.

PARTIE I : RÉSOLUTION DES ÉQUATIONS DIFFÉRENTIELLES

Pour résoudre les équations suivantes:

$$\frac{dIi(t)}{dt} = -\frac{Ii(t)}{\tau I} + \sum_{j=1}^{N} wijSj(t) \qquad (1)$$

$$\frac{dUi(t)}{dt} = \frac{-Ui(t) + Ii(t)}{\tau U} \qquad (2)$$

nous avons utilisé la méthode d'intégration par Euler. Les solutions obtenues sont ci-dessous :

- Pour l'équation (1) :

$$Ii(t) = Ii(t-\Delta t)(1-\frac{\Delta t}{\tau I}) + \Delta t \sum_{j=1}^{N} wijSj(t-\Delta t)$$

- Pour l'équation (2) :

$$Ui(t) = Ui(t-\Delta t)(1-\frac{\Delta t}{\tau U}) + \Delta t \frac{Ii(t-\Delta t)}{\tau U}$$

PARTIE II : LECTURE DE L'ARTICLE DE RÉFÉRENCE

Les équations de la partie précédente diffèrent de ceux de l'article [1]. Cette différence réside dans le dernier terme des équations de ce dernier. En effet, les équations de l'article prennent en considération la récurrence contrairement à ceux de la partie I où on remarque que juste la propagation en avant qui a été considérée.

PARTIE III : CODE À TROUS

Après avoir complété le code, nous avons effectué l'évaluation de plusieurs implémentations de la dérivée en plus de celle fournie dans le code. Ces implémentations sont Celu, Selu, Tanh, Sigmoide (toutes 4 issues de la bibliothèque torch et dont les allures sont disponibles en annexe) et Surrogate (inspiré de [2]). Les résultats consignés dans le tableau 1 permettent de constater une meilleure précision pour la sigmoide suivi de la surrogate. On confirme également cela à travers l'évolution de l'erreur au cours des itérations (figure 1).

implémentation de la dérivée	Initiale avec (Relu) Alpha = 0	Surrogate	(torch) Celu	(torch) Selu	(torch) tanh	(torch) Sigmoide
Précision (10 ittérations)	79.7	93,1	61,2	45	9,6	96,6

Tableau 1 : Précision en fonction de la fonction d'activation implémentée

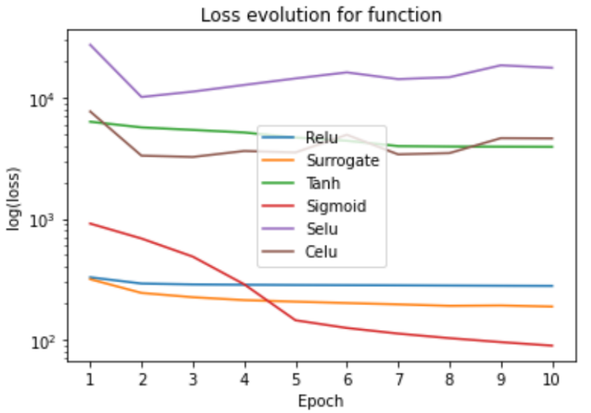

Figure 1 : Evolution de l'erreur des différentes fonctions d'activation au cours des itérations

PARTIE IV : ANALYSE, ÉVALUATION ET COMPRÉHENSION DE L'IMPACT DES FORMES ET PARAMÈTRES DES DÉRIVÉES

Dans cette partie, nous avons varié le alpha dans la fonction Relu pour voir son impact sur les résultats. Nous avons cherché la précision pour nos alpha {0, 0.01, -1} :

Alpha relu	0	0.01	-1
Précision (20 itérations)	86,9	9,6	14,2

Tableau 2 : Précision en fonction de la valeur de alpha

Les résultats obtenus prouvent qu'avec un alpha nul, la précision est meilleure versus les autres expériences. Par ailleurs, pour nos autres paramètres (nombre de couches, de neurones, taux d'apprentissage...) nous avons gardés ceux déjà fourni dans le code à trous.

PARTIE V : ANALYSE, ÉVALUATION ET COMPRÉHENSION DE L'IMPACT DES MÉTA-PARAMÈTRES

Pour cette partie, nous avons gardé le alpha nul et nous avons essayé de faire des modifications sur les méta-paramètres; Ainsi, nos résultats en général (voir tableau 3 au 7) ont une tendance de baisse de précision avec l'augmentation du paramètre modifié (nombre de couches cachées, de neurones, d'itérations, taux d'apprentissage et pas de discrétisation). Ceci est expliqué par le fait que plus les valeurs sont grandes plus le nombre de paramètres à ajuster augmente.

De plus, la piste du surapprentissage comme explication a été explorée (voir tableau 8) mais elle n'est pas concluante car dans le cas du nombre de neurone, la précision sur les données d'apprentissage est similaire à celle sur les données de validation.

PARTIE VI : RÉSEAU DE TYPE "APPRENTISSAGE EXTRÊME"

Dans cette partie, nous avons dans un premier temps ignoré la 1ère couche lors de l'apprentissage puis dans un second temps l'apprentissage a été fait uniquement sur la dernière couche (voir les résultats des expériences faites en annexe) . On a remarqué que la diminution du nombre de neurones, des couches cachées et le taux d'apprentissage assure une augmentation de la précision par rapport à l'expérience initial de la partie précédente (avec le même nombre de couche cachée et sans apprentissage extrême)Pour comparer l'apprentissage classique et celui extrême, on a constitué un réseau avec deux couches cachées de 64 neurones chacune et un taux d'apprentissage de 0.01. Après un entraînement de 10 itérations nous avons obtenu les résultats consignés dans

le tableau(9). Nous avons constaté une même durée d'entrainement (16 mn) pour toutes les méthodes mais une meilleure précision pour l'apprentissage classique. Le constat sur la précision est conforme à nos attentes en raison du nombre de poids à ajuster en moins. Cependant sur la durée nous nous attendions à avoir une durée plus courte pour les apprentissages extrêmes.

REMERCIEMENT

Nos sincères remerciements à l'équipe qui a mis à notre disposition l'accès à la bilbiothèque PyTorch et à Google Colab, A Ismaël Balafrej pour avoir préparé les notebooks python et à Ahmad El Ferdaoussi pour les avoir mis à jour.

RÉFÉRENCES

[1]: EO Neftci, H Mostafa & F Zenke, "Surrogate gradient learning in spiking neural net-works", IEEE SIGNAL PROCESSING MAGAZINE

[2]: https://github.com/fzenke/spytorch/blob/master/notebooks/SpyTorchTutorial1.ipynb

[3]: SLAYER: Spike Layer Error Reassignment in Time (2018) https://arxiv.org/pdf/1810.08646.pdf

CONCLUSION

A partir des expériences réalisées, on peut constater que l'augmentation des valeurs de certains paramètres n'induis pas systématiquement une augmentation de la précision, d'où la nécessité de trouver un compromis pour avoir la meilleure performance.En outre en utilisant les paramètres du tableau (10) nous avons obtenu une précision de 96.6%. Ce qui est inférieure au 99.36% de [3] mais est tout de même satisfaisant.

Nb de couches cachées	1	2	3
Précision (10 itérations)	79,7	24,2	09,6

Tableau 3 : Précision en fonction du nombre de couches cachées avec Relu alpha = 0 et nombre de neurones dans chaque couches est 128

relu nb_hidden	2^6 = 64	2^7 = 128	2^8 = 256	2^9 = 512
Précision (10 itérations)	85,9	79,7	65,4	30,2

Tableau 4 :Précision en fonction du nombre de neurones avec Relu alpha = 0 pour une couche cachée

Nombre d'itération	5	10	20	25
Précision	79,7	79,7	79,4 (GPU) 86,9 (CPU)	79,1

Tableau 5 : Précision en fonction du nombre d'itérations:

LR	0.001	0.01	0.1	1
Précision	86.9	79.4	69.6	70.2

Tableau 6 : Précision en fonction du taux d'apprentissage

dt (ms)	1	10	100
précision	79.4	09.6	09.6

Tableau 7 : Précision en fonction du pas de discrétisation

Nb_hidden	64	512
Précision	Train: 85.9 Test: 85.5	Train: 30.3 Test: 30.1

Tableau 8 : Expérience de surapprentissage

Méthode	Durée d'entrainement	Précision
Classique	16.00449961423874 mn	0.839
Extrême 1	16.310380001862843 mn	0.660
Extrême 2	16.237150386969248 mn	0.642

Tableau 9 :Comparaison entre apprentissage classique et celui extrême en terme de durée d'entrainement et de précision

Apprentissage	Nbde couches cachées	Nb de neurones par couche cachée	Taux d'apprentissage	Fonction d'activation	Paramètre de discrétisation	Nombre d'itération
classique	1	128	0.01	Sigmoide	1 ms	10

Tableau 10 : Paramètres pour avoir une précision de 96,6%

Annexe

init: sans A. extrême avec une précision de 0.242	init: sans A. Extrême avec une précision de 0.096
• 2 couches , 128 neurones, lr=0.01 précision 0.194	• 3 couches , 128 neurones, lr=0.01 précision 0.096
• 2 couches , 64 neurones, lr=0.01 précision 0.459 • 2 couches , 256 neurones, lr=0.01 précision 0.096	• 3 couches , 64 neurones, lr=0.01 précision 0.235 • 3 couches , 256 neurones, lr=0.01 précision 0.096
• 2 couches , 128 neurones, lr=0.1 précision 0.096 • 2 couches , 128 neurones, lr=0.001 précision 0.744	• 3 couches , 128 neurones, lr=0.1 précision 0.096 • 3 couches , 128 neurones, lr=0.001 précision 0.202

init: A. extrême avec une précision de 0.242	init: A. extrême avec une précision de 0.096
• 2 couches, 128 neurones, lr=0.01 précision 0.536	• 3 couches , 128 neurones, lr=0.01 précision 0.096
• 2 couches , 64 neurones, lr=0.01 précision 0.641 • 2 couches , 256 neurones, lr=0.01 précision 0.182	• 3 couches , 64 neurones, lr=0.01 précision 0.459 • 3 couches , 256 neurones, lr=0.01 précision 0.096
• 2 couches , 128 neurones, lr=0.1 précision 0.283 • 2 couches , 128 neurones, lr=0.001 précision 0.725	• 3 couches , 128 neurones, lr=0.1 précision 0.143 • 3 couches , 128 neurones, lr=0.001 précision 0.678

Tableau 11 : Expériences sur les résultats de l'apprentissage "extrême"

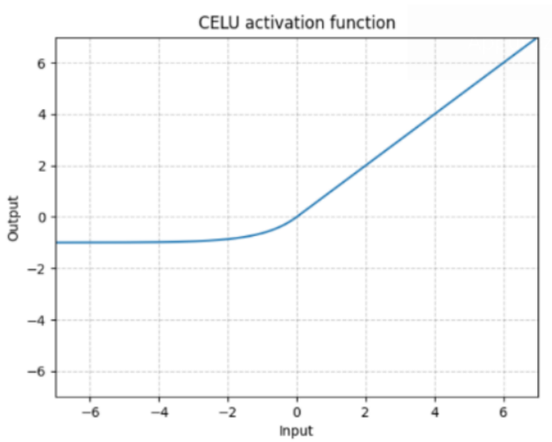

Figure 2 : Allure de la fonction d'activation Celu

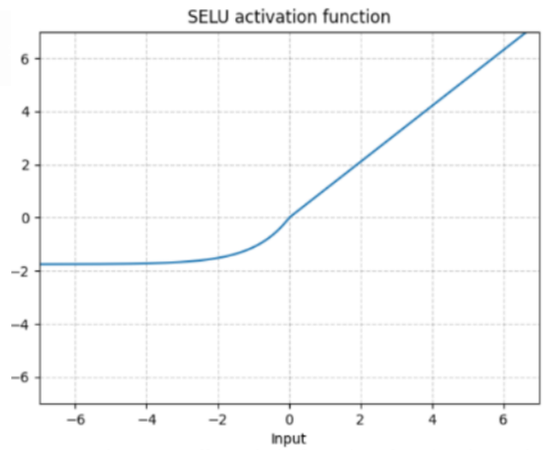

Figure 3 : Allure de la fonction d'activation Selu

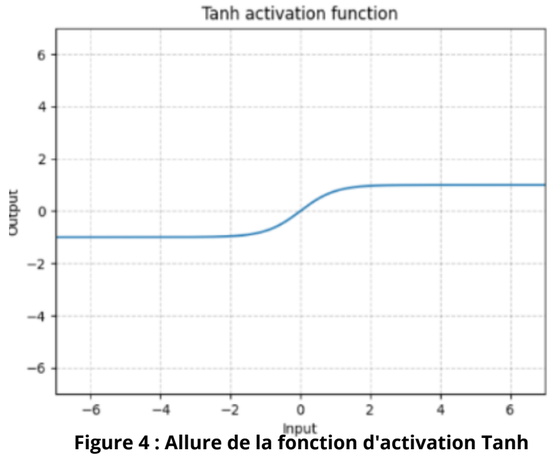

Figure 4 : Allure de la fonction d'activation Tanh

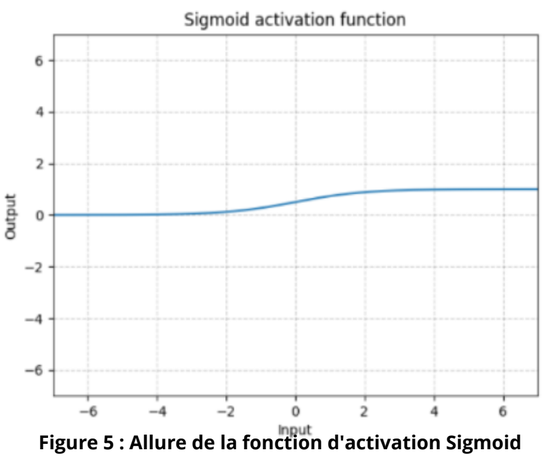

Figure 5 : Allure de la fonction d'activation Sigmoid