Rétropropagation de l'erreur et classification

github.com/victor-yon/gei723

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
Auteurs: Soline Bernard, Antoine Marion, Victor Yon Groupe : P2 Temps de travail: 45h/personne Date: 10/12/2020

Objectif

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.
Etre capable d'étudier et d'analyser l'impact des méta-paramètres sur les résultats.
Etre capable de définir l'architecture d'un réseau de neurones à décharges avec apprentissage par rétropropagation de l'erreur pour une application en classification.
Etre capable de faire le lien entre la littérature scientifique proposée dans ce problème et l'implémentation informatique en language python.

Type de réseau

Nous étudions la classification de la base de données MNIST avec une réseau de neurones à décharge et avec le mécanisme de rétropopagation du gradient de l'erreur. C'est un algorithme d'optimisation permettant d'ajuster les paramètres d'un réseau de neurones multicouches pour mettre en correspondance des entrées et des sorties référencées dans une base d'apprentissage. Cette rétropropagation se fait de la dernière couche vers la première.
Les poids du réseau sont initialisés de façon aléatoire. L'algorithme suit les étapes suivantes :
- Une image est présentée au réseau. Les valeurs de chaque pixel sont converties en valeurs entre 0 et 1, puis, en instants de décharges échelonnés de 0 à 100. L'arrière-plan va décharger instantanément, alors que les pixels blancs, représentant le chiffre sur l'image, vont décharger à des valeurs de temps plus élevées.
- La propagation vers l'avant se calcule avec une fonction d'activation *h(x)*.

$$S_j^{(l)} = h^{(l)}\left(\sum_{i=1}^{N} w_{ij}^{(l)} S_j^{(l-1)}(t)\right)$$

- Lorsque la propagation est terminée, la dernière couche annonce une valeur de sortie (ici entre 0 et 10)
- Cette valeur est comparée à la valeur désirée et l'erreur est calculée.

$$y_i - x_i^{tar}$$

- L'erreur est propagée par l'intermédiaire de la dérivée de la fonction d'activation sur toutes les couches de neurones. Les poids sont alors mis à jour.
(Le calcul de l'erreur peut être effectué de différentes façon)

$$e_i^{sortie} = h'^{(l)}\left(\sum_{j=1}^{N} w_{ij}^{(Sortie)} S_j^{(Sortie-1)}(t)\right)(y_i - x_i^{tar})$$

$$e_j^{(l-1)} = h'^{(l-1)}\left(\sum_{j=1}^{N} w_{ij}^{(l-1)} S_j^{(l-2)}(t)\right)(y_i - x_i^{tar})$$

Equations différentielles

On utilise un réseau de neurones modélisé avec des neurones à décharges et fuite. Le courant d'entrée est uniquement influencé par les neurones de la couche précédente (pas de rétroaction au sein d'une même couche de neurone). Le réseau est modélisé par les équations suivantes :

$$\frac{dI_i^{(l)}}{dt} = -\frac{I_i^{(l)}(t)}{\tau_I} + \sum_{j=1}^{N} w_{ij}^{(l)} S_j^{(l-1)}(t)$$

$$\frac{dU_i^{(l)}(t)}{dt} = -\frac{\left(U_i^{(l)}(t) - U_{rest}\right) + RI_i^{(l)}(t)}{\tau_u} + S_i^{(l)}(t)(U_{rest} - v)$$

I représente le courant du neurone, U le potentiel membranaire, w_ij le,poids synaptique du neurone j vers le neurone i, S_j la séqeunce d'implusions de sortie du neurone j, N le nombre de neurones de la couche présynaptique et (l) l'indice de la couche de neurone.
On fixe le potentiel au repos à 0, la résistance à 1.
On discrétise l'équation grâce à la méthode d'Euler.

$$\frac{dX(t)}{dt} \approx \frac{X(t+\Delta t) - X(t)}{\Delta t}$$

$$\frac{dI_i(t)}{dt} = -\frac{I_i(t)}{\tau_I} + \sum_{j=1}^{N} w_{ij} S_j(t)$$

$$\frac{dU_i(t)}{dt} = \frac{-U_i(t) + I_i(t)}{\tau_u}$$

On obtient les équations suivantes :

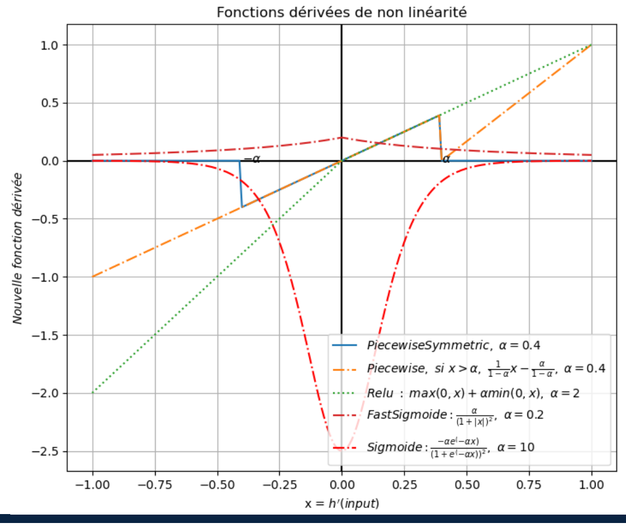
$$U(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau_u}\right) U(t) + I(t+\Delta t)$$

$$I(t+\Delta t) = \left(1 - \frac{\Delta t}{\tau_I}\right) I(t) + \sum_{j=1}^{N} w_{ij}^{(l)} S_j^{(l-1)}(t+\Delta t)$$

Dérivée de non linéarité

Nous avons étudié 3 formes de dérivées de non linéarité :
- La fonction de type Relu
- Une fonction de type sigmoïde rapide
- Une fonction continue par morceau

Pour chacune des fonctions, nous avons fait varier les paramètres afin de déterminer la meilleure configuration.
La variation des paramètres alpha pour chaque type est donnée en annexe.



Fonction Sigmoïde

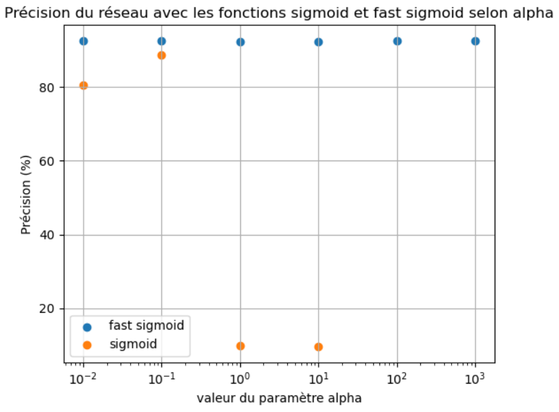
Nous avons testé 2 formes de sigmoïdes.

Fonction Fast Sigmoïde

$$z = \frac{\alpha}{(1+|x|)^2}$$

Fonction Sigmoïde

$$z = -\frac{\alpha e^{-\alpha x}}{(1+e^{-\alpha x})^2}$$



La fonction sigmoïde rapide semble plus robuste à différentes valeurs d'alpha comparativement à la fonction sigmoïde standard. Les simulations ont été effectuées avec 3 epochs et le jeu complet des données.

Fonction continue par morceaux

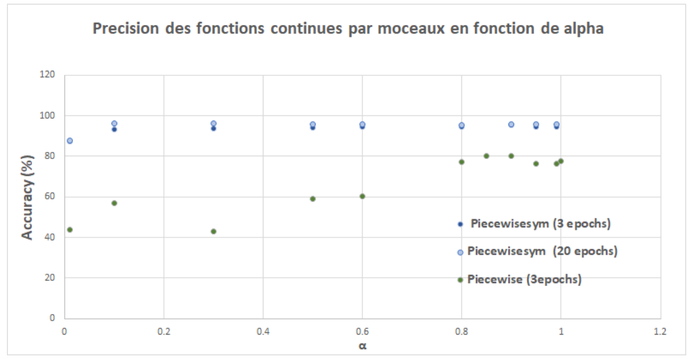
Nous avons testé 2 formes de fonction continues par morceaux.

Piecewisesym

$$z = \begin{cases} 0 \; si \; x \geq \alpha \; ou \; x \leq -\alpha \\ h'(x) \; si \; -\alpha \leq x \leq \alpha \end{cases}$$

Piecewise

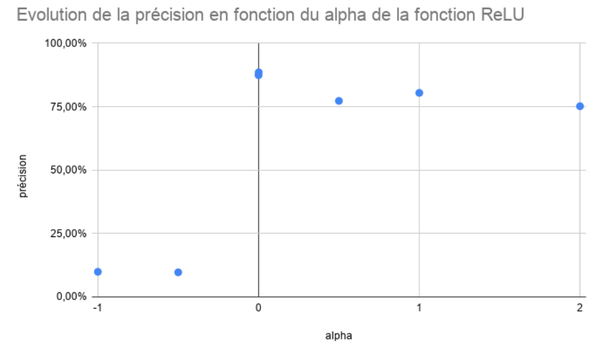
$$z = \begin{cases} \frac{1}{1-\alpha} x - \frac{\alpha}{1-\alpha} \; si \; x \geq \alpha \\ h'(x) \; si \; x < \alpha \end{cases}$$



Le meilleur apprentissage est obtenue pour la fonction continue par morceaux symétrique. Sur 3 epochs, le meilleur apprentissage est obtenu pour un alpha de 0.9 (95,57%) , alors que pour 20 epoch, il est obtenu pour un alpha de 0.1 (96,4%). Dans cette dernière configuration, les poids mettent plus de temps à s'adapter. Dans tous les cas, la précision est supérieure à 93%.

Fonction ReLU

La fonction Relu est définie par : $z = \max(0, y(x)) + \alpha \min(0, y(x))$



Nous avons fait varier la pente de la fonction ReLU pour la partie inférieure à zero de la courbe avec un paramètre *alpha*. Lorsque sa valeur est positive ou égale à zéro le réseau peut apprendre efficacement et ainsi obtenir de bons résultats de classification, avec une valeur optimale de *alpha* proche de 0 (87% de précision sur 3 époques).
Lorsque la valeur de *alpha* est négative, la direction de la courbe en dessous de zéro est inversée et le réseau ne converge pas vers une solution, ce qui est cohérent avec le fait que le gradient n'a plus le bon signe et certains paramètres sont augmentés lorsqu'ils devraient être diminués.

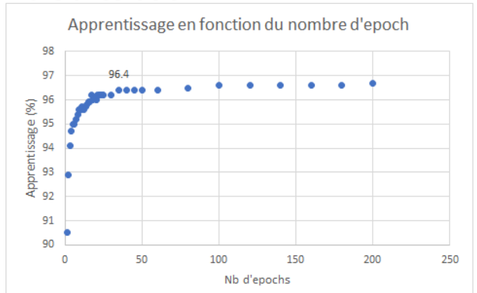
Notre meilleure apprentissage est obtenue avec la fonction continue par morceaux symétrique. Nous choisissons de comparer l'ensemble des résultats sur 3 epochs. Notre meilleure configuration est obtenue pour un **alpha de 0,5** qui nous permet d'obtenir **94,06%** d'apprentissage.

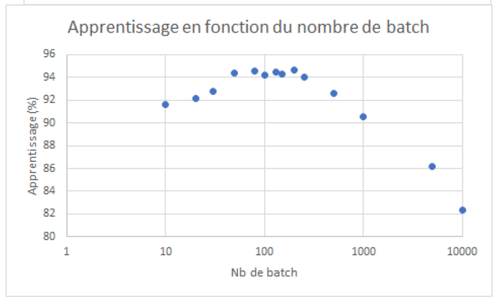
Réseau de neurones pour classification MNIST

Valeurs par défaut

Nb d'images d'entrainement	Nb d' image de test	Nb d'epoch	Taille du batch	Taux d'apprentissage
50000	10000	3	256	0.01
Temps par image	**Pas de temps**	**Tau_v**	**Tau_i**	**Taille et nombre de couche cachée**
100 ms	1 ms	20ms	5ms	128

Nombre d'itération

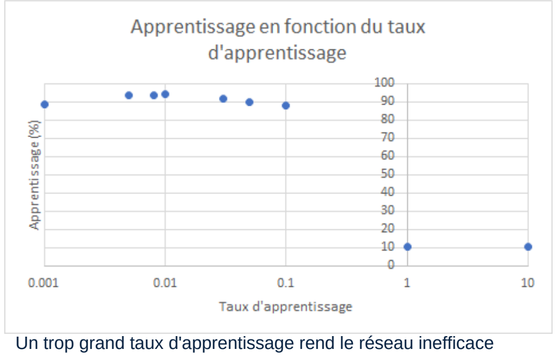




Avec les paramètres par défaut de notre réseau de neurone, l'apprentissage converge après 35 épochs (96.4%). L'apprentissage maximal est obtenue après 200 epochs (96.7%). La rapidité de la convergence dépend des autres paramètres du système.

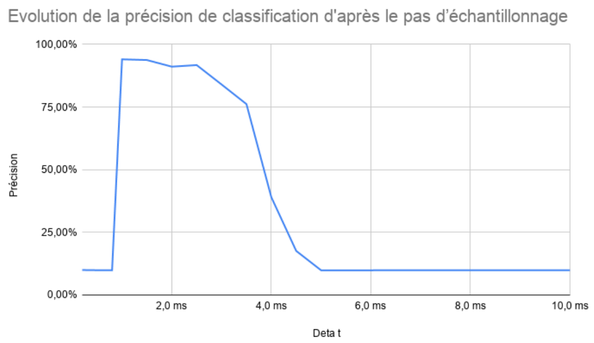
Avec 3 epochs, nous obtenons la meilleure configuration pour une taille de batch comprise entre 50 (94.43%) et 300 avec une valeur maximale à 200 (94.7%). Comme on peut s'y attendre, si la taille du batch est trop grande, l'apprentissage est moins bon car il n'y a pas assez de feedback. Cependant, pour des tailles de batch inférieures à 200, l'erreur diminue mais oscille sur une plage d'environ 150 valeurs (Voir annexe 2). Pour éviter ces oscillations, nous définissons la taille de batch optimale sur 256 (94.06%)..

Taux d'apprentissage



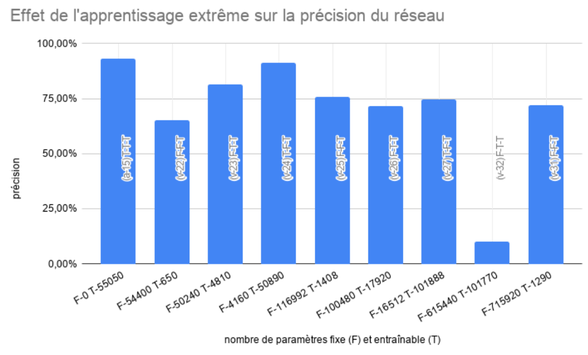
Un trop grand taux d'apprentissage rend le réseau inefficace

Pas d'échantillonnage (delta t)

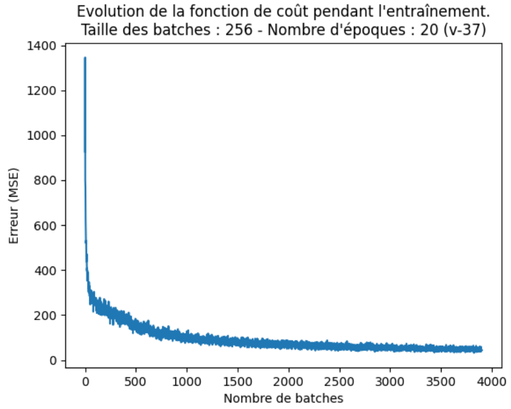


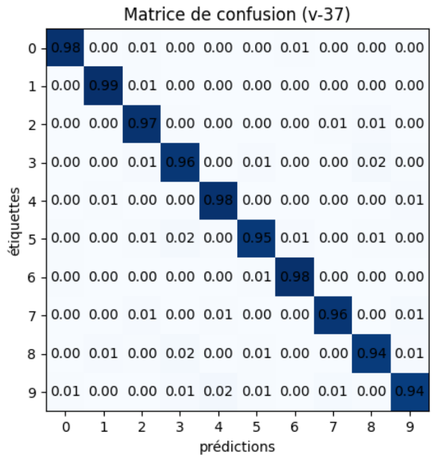
Les meilleures performances de classification ont été obtenues avec un pas d'échantillonnage de **1 ms**. Lorsque cette valeur augmente, le temps de calcul diminue significativement tout comme la précision. **Au-delà de 4 ms,** l'écart est trop grand et la précision passe en **dessous des 40%**. On peut supposer qu'à cette échelle, des décharges peuvent être manquées et les paramètres ne peuvent plus être appris efficacement.
En dessous de 1 ms, le temps de calcul augmente fortement et la précision diminue soudainement. Ce comportement est inattendu puisqu'un écart plus faible devrait augmenter la précision du réseau et au minimum maintenir les performances. Nous supposons que ces résultats sont causés par la méthode de calcul et la façon dont python gère les opérations sur les variables à virgule flottante ayant une valeur fiable.

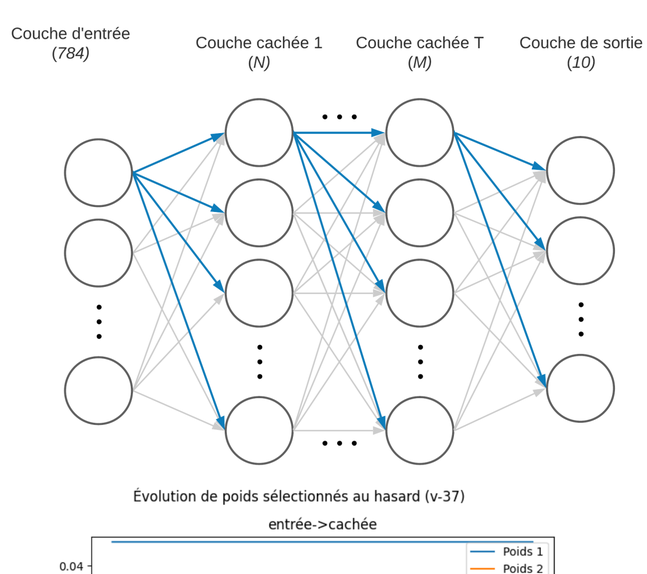
Apprentissage extrême

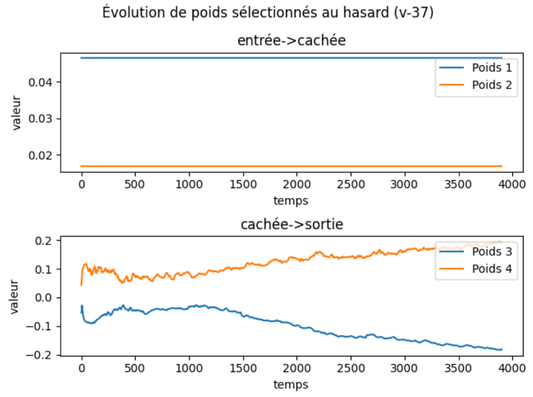


Nous avons fixé certains paramètres en fonction de leur emplacement dans le réseau afin d'en observer l'effet sur l'apprentissage. La vitesse d'entraînement est inversement proportionnelle au nombre de paramètres variables puisque le gradient doit être calculé pour chacun d'entre eux. Avec des couches fixes, il est donc envisageable de créer des réseaux contenant beaucoup de neurones. Nous avons constaté que la performance du réseau reste très fortement **corrélée au nombre paramètres variables** et qu'un grand nombre de paramètres fixes n'est pas forcément bénéfique.
Le meilleur résultat reste celui où toutes les couches sont variables (a-15) mais une version avec une couche fixe centrale (v-24) s'approche de ses performances avec 91% de précision. Nous avons noté qu'il est important d'avoir une ou **plusieurs couches variables avant la sortie** pour réaliser efficacement la classification. Et qu'un trop grand nombre de paramètres à entraîner mène systématiquement vers une mauvaise classification de jeu de test (v-32).

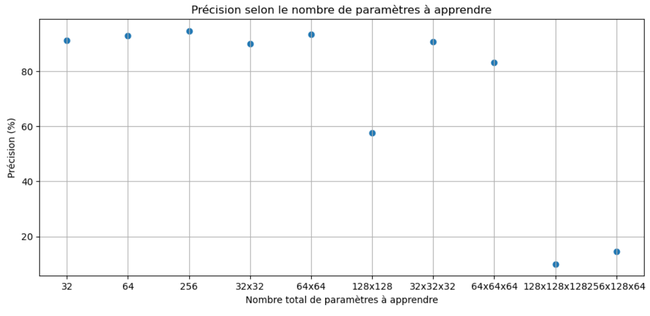








Effet du nombre de couches cachées et du nombre de neurones par couche



Le nombre de paramètres du réseaux est exprimé en abcisse. Le plus petit réseau essayé comportait une couche cachée de 32 neurones alors que le réseau le plus lourd comportait 3 couches cachées composées respectivement de 256, 128 et 64 neurones. Les essais ont été réalisés avec 3 epochs au départ pour toutes les configurations. Ensuite, les configurations pour lesquelles la précision n'était pas assez élevée ont été entraînées avec 20 epochs. La précision n'a cependant pas augmenté avec le nombre d'epochs.
L'augmentation du nombre de paramètres n'est pas corrélée de manière linéaire avec la précision. Un trop grand nombre de paramètres peut rendre difficile la convergence du réseau. Si on se fie sur la descente de gradient, un minimum local qui n'est pas le minimum global peut faire plafonner la précision du réseau. Une hypothèse retenue est le fait que l'apprentissage d'un plus grand nombre de paramètres prend plus de temps. Par contre, les résultats ont montré que lorsque l'on augmente le nombre d'itération, les réseaux plus grands n'ont pas montré de meilleure performance. On peut conclure que le problème est soi lié à un sur-apprentissage ou alors une difficulté de convergence

Discussion des résultats, conclusion

Lorsque l'on observe les simulations qui ont été effectuées, on arrive à la conclusion que la majorité des méta-paramètres étaient proche des valeurs optimales. Le gain le plus important que nous avons observé a été lorsque la fonction d'approximation du gradient a été remplacée par la fonction continue par morceaux symétrique par rapport à l'axe des ordonnées (*piecewise_sym*).
Le meilleurs résultat que nous avons obtenu est composé de 2 couches cachés (128-64) avec 96,37% de classification correcte sur le jeu de test et 96,47% sur le jeu de validation. En plus des paramètres par défaut définit précédemment nous avons utilisé la méthode continue par morceaux symétrique avec un alpha de 0,5 et 20 époques. Nous sommes assez fière de ces résultats qui sont proche de ceux de l'état de l'art.
D'autres paramètres pourraient être évalués à différentes valeurs. Il serait par exemple intéressant d'évaluer différents taux d'apprentissage à des valeurs non fixes, on pourrait essayer de le faire diminuer au cours de l'entrainement. D'autres travaux montrent une décroissance progressive dans la valeur du taux d'apprentissage améliorant les résultats.
Sinon, par souci de temps, le nombre d'époques a été maintenu à 20 pour le simulations sur les paramètres optimaux que nous avions identifiés. En observant le graphique de la perte en fonction des époques, on constate qu'elle ne s'est pas complètement stabilisée même après les 20 époques. Il serait intéressant d'observer les effets de l'augmentation du nombre d'époques sur notre simulation mais faisant attention au risque de sur-apprentissage.

Bibliographie

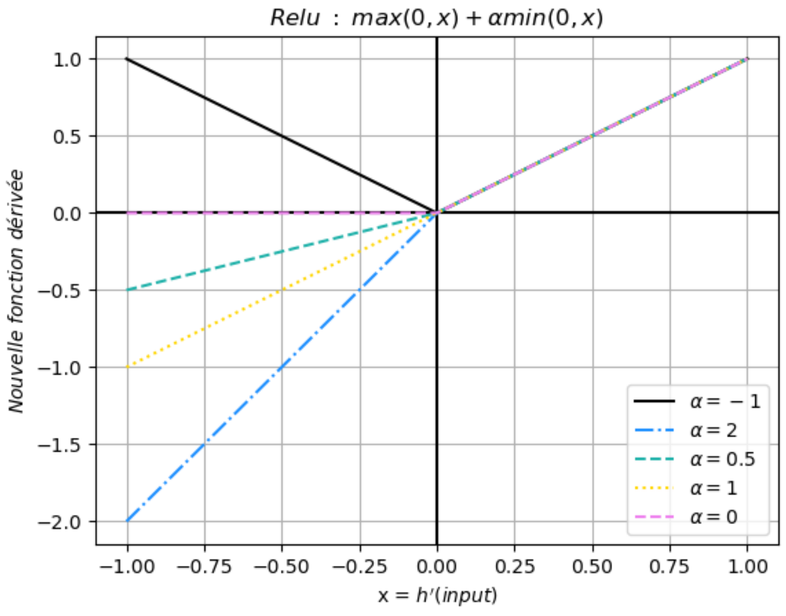
[1] Neftci, Emre O., Hesham Mostafa, and Friedemann Zenke. "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks." IEEE Signal Processing Magazine 36, no. 6 (November 2019): 51–63. https://doi.org/10.1109/MSP.2019.2931595.

Remerciements

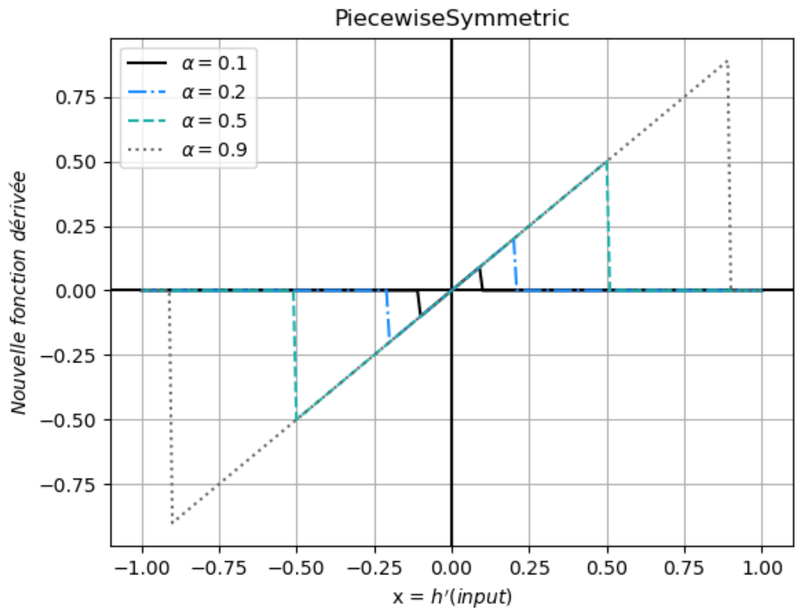
PyTorch, numpy, GitHub, LucidChart et StackOverflow
Base de données MNIST : http://yann.lecun.com/exdb/mnist
Surrogate Gradient Learning : https://github.com/surrogate-gradient-learning

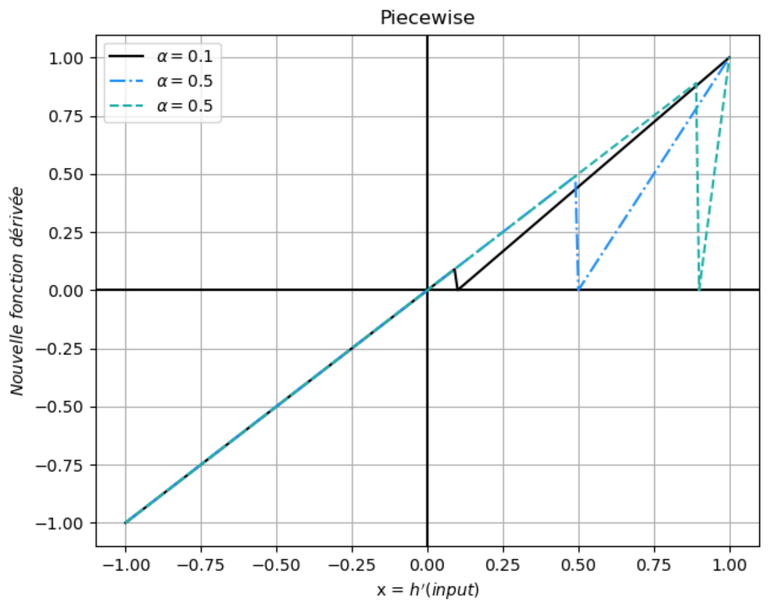
Annexe 1 – Fonctions de substitution évaluées

Fonction Relu

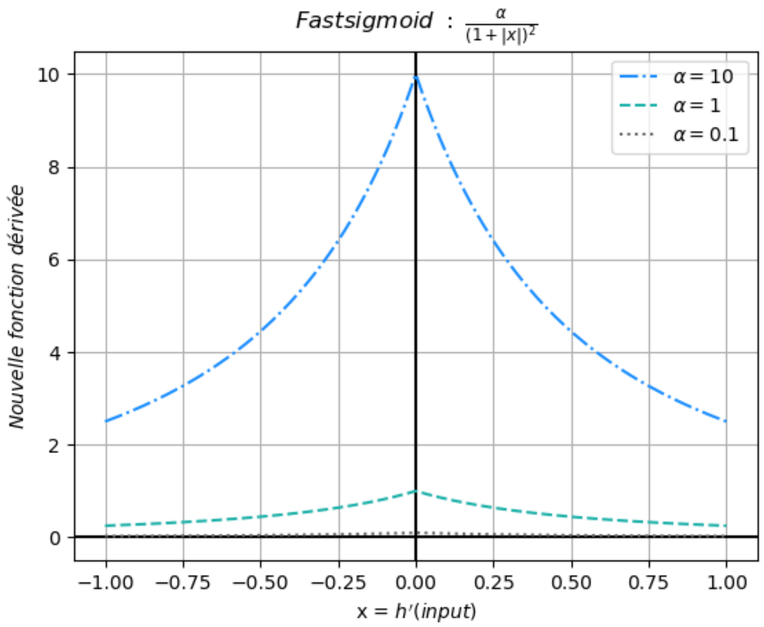


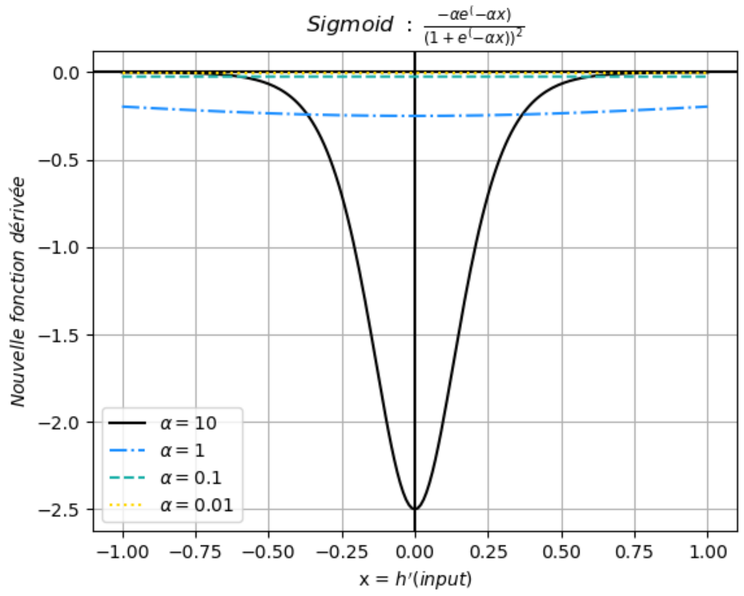
Fonctions continues par morceaux





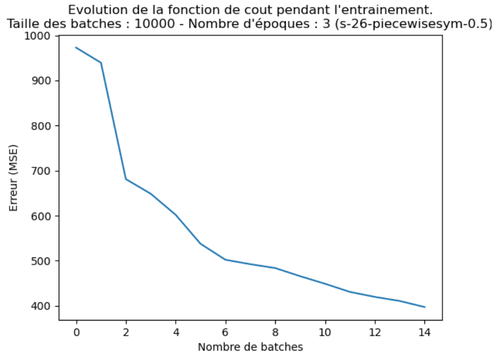
Fonctions sigmoïdes

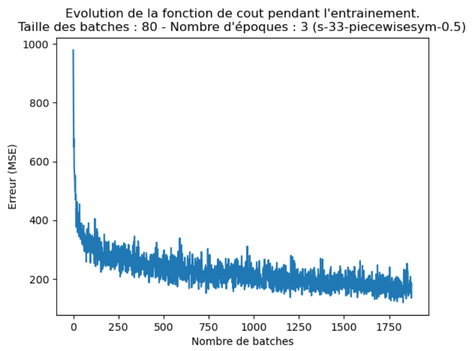


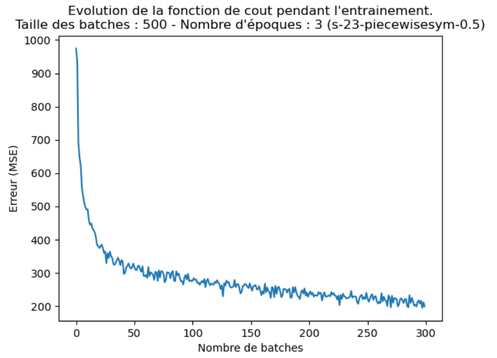


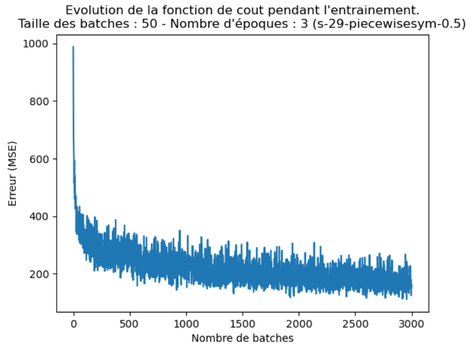
Annexe 2

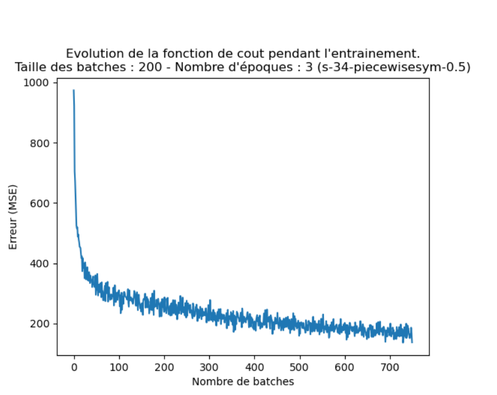
Impact du nombre de la taille du batche sur l'apprentissage

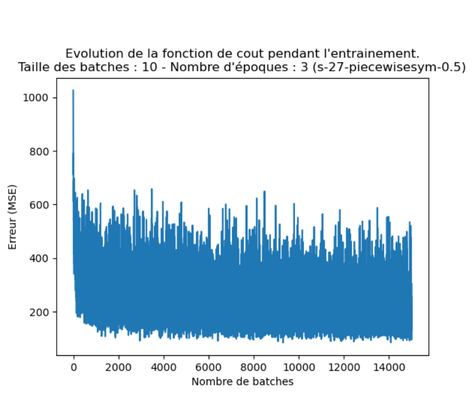


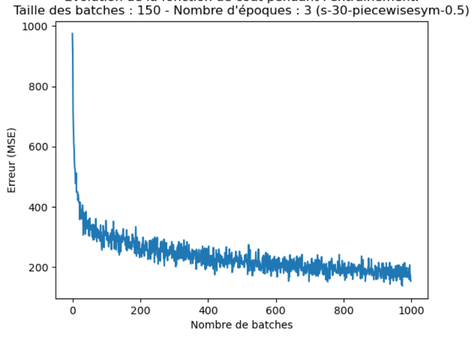


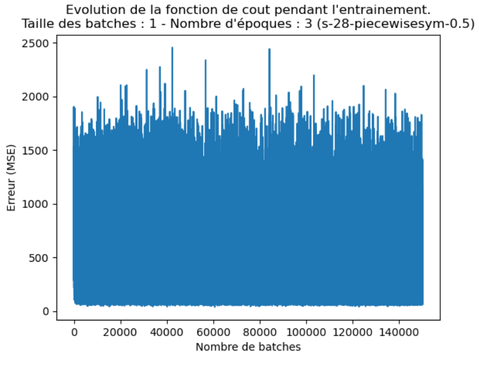




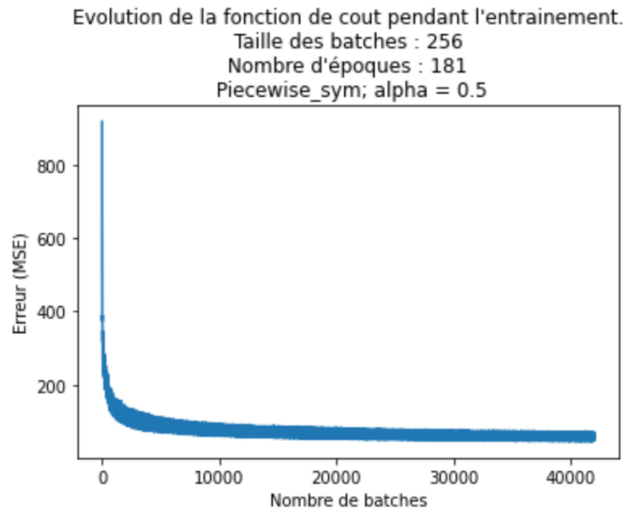




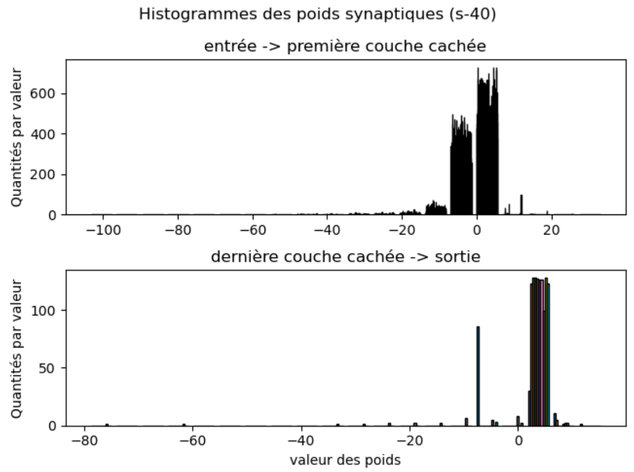


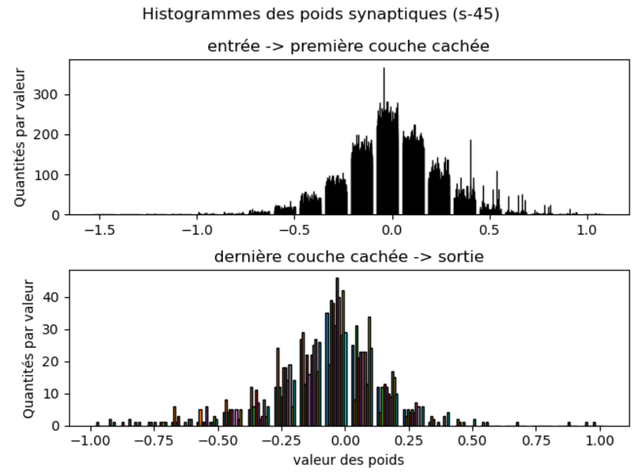



```
Model accuracy on test set: 0.966
Epoch 180 -- loss : 55.7944
```

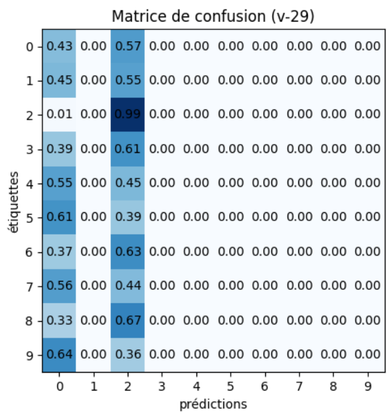


Répartition des poids pour un apprentissage avec un taux d'apprentissage élevé (à gauche) ne permettant pas un bon apprentissage et un taux plus faible (à droite)

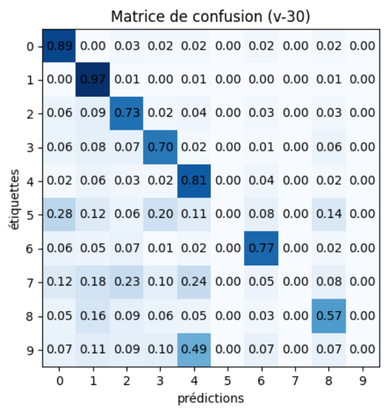




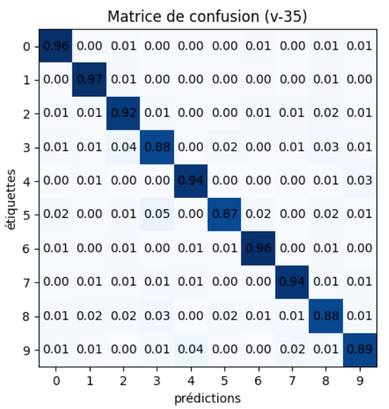
Différence de classification entre 3 apprentissages extrêmes, du moins précis (à gauche) au plus précis (à droite)



Couches cachées : 128-128-128
Apprentissage : False-False-True



Couches cachées : 128-128-128
Apprentissage : False-False-False



Couches cachées : 64-128
Apprentissage : True-False

Annexe 3 – Résultats en fonction des paramètres de simulation

Résultats en faisant varier la fonction de substitution

Avec 50 000 image d'entraînement; 10 000 images de test; taille de batch 256; taux d'apprentissage 0,01; 100 ms par images; pas de temps dt 1ms; tau v 20 ms; tau i 5 ms; une couche cachée de 128 neurones

Variables :	run_name	nb_epoch	surrogate_gradient	surrogate_alpha	test accuracy
Valeur défaut :	-	20	relu	None	10,00
Simulations	a-4	3	fast sigmoid	100	92,68
	a-7	3	fast_sigmoid	0,01	92,55
	a-3	3	fast_sigmoid	0,1	92,59
	a-8	3	fast_sigmoid	1	92,44
	a-9	3	fast_sigmoid	10	92,44
	a-10	3	fast_sigmoid	1000	92,54
	s-7	3	piecewise	0,01	43,88
	s-2	3	piecewise	0,1	56,73
	s-9	3	piecewise	0,3	43,04
	s-1	3	piecewise	0,5	59,12
	s-8	3	piecewise	0,6	60,13
	s-5	3	piecewise	0,8	77,30
	s-10	3	piecewise	0,85	80,10
	s-3	3	piecewise	0,9	80,18
	s-6	3	piecewise	0,95	76,17
	s-4	3	piecewise	0,99	76,26
	s-17	3	piecewise_sym	0,01	87,46
	s-13	3	piecewise_sym	0,1	93,30
	s-20	3	piecewise_sym	0,3	93,52
	s-11	**3**	**piecewise_sym**	**0,5**	94,06
	s-15	**10**	**piecewise_sym**	**0,5**	95,57
	s-21	3	piecewise_sym	0,6	94,69
	s-18	3	piecewise_sym	0,8	94,40
	s-12	3	piecewise_sym	0,9	94,32
	s-14	10	piecewise_sym	0,9	95,57
	s-19	3	piecewise_sym	0,95	94,47
	s-16	3	piecewise_sym	0,99	94,52
	v-3	3	relu	-1	9,91
	v-2	3	relu	0	87,43
	v-1	3	relu	0,001	88,58
	v-4	3	relu	1	80,46
	v-5	10	relu	1	77,56
	s	3	relu	2	75,20
	s	3	relu	0,5	77,30
	s	3	relu	-0,5	9,70
	a-5	3	sigmoid	0,01	80,60
	a-1	3	sigmoid	0,1	88,62
	a-6	3	sigmoid	1	9,91
	a-2	3	sigmoid	10	9,50

Résultats en faisant varier les paramètres sur réseau avec la fonction de substitution "piecewise_sym" et un alpha de 0,5

Variables :	run_name	nb_epoch	batch_size	learning_rate	delta_t	size_hidden_layers	trainable_layers	test accuracy
Ref values :	-	3	256	0.01	1,0 ms	128	TRUE	94,06%
Number of hidden layers	a-11	3				32		91,16%
	a-12	3				64		92,95%
	a-13	3				256		94,74%
	a-14	3				32,32		90,04%
	a-15	3				64,64		93,33%
	a-16	3				128,128		57,70%
	a-17	3				32,32,32		90,65%
	a-18	3				64,64,64		83,19%
	a-19	3				128,128,128		9,91%
	a-20	3				256,128,64		14,51%
	a-21	10				64,64		94,99%
	a-22	20				256,256		10,00%
	a-23	20				512,512		10,00%
	v-37	20				128,64		96,37%
Delta t	v-18	3			0,2 ms			9,98%
	v-17	3			0,5 ms			9,91%
	v-19	3			0,8 ms			9,91%
	v-16	3			1,5 ms			93,77%
	v-9	3			2,0 ms			91,13%
	v-10	20			2,0 ms			94,44%
	v-15	3			2,5 ms			91,76%
	v-11	3			3,0 ms			84,02%
	v-14	3			3,5 ms			76,19%
	v-12	3			4,0 ms			39,03%
	v-13	3			4,5 ms			17,60%
	v-8	3			5,0 ms			9,87%
	v-6	3			10,0 ms			9,91%
	v-7	20			10,0 ms			9,92%
Batch size	s-28	3	1					91,09%
	s-27	3	10					91,61%
	s-31	3	20					92,17%
	s-32	3	30					92,82%
	s-29	3	50					94,43%
	s-33	3	80					94,57%
	s-22	3	100					94,24%
	s-35	3	130					94,46%
	s-30	3	150					94,33%
	s-34	3	200					94,70%
	s-23	3	500					92,59%
	s-24	3	1000					90,52%
	s-25	3	5000					86,15%
	s-26	3	10 000					82,37%

Variables :	run_name	nb_epoch	batch_size	learning_rate	delta_t	size_hidden_layers	trainable_layers	test accuracy
Ref values :	-	3	256	0.01	1,0 ms	128	TRUE	94,06%
Learning rate	s-38	3		0,001				88,67%
	s-43	3		0,005				93,26%
	s-45	3		0,008				93,68%
	s-11	3		0,01				94,06%
	s-44	3		0,03				91,62%
	s-42	3		0,05				89,59%
	s-39	3		0,1				87,74%
	s-40	3		1				10,64%
	s-41	3		10				10,64%
Extreme learning	v-20	3					FALSE	78,79%
	v-21	20					FALSE	79,35%
	v-22	3				64,64	FALSE, FALSE	65,05%
	v-23	3				64,64	FALSE, TRUE	81,31%
	v-24	3				64,64	TRUE, FALSE	91,47%
	v-35	3				64,128	TRUE, FALSE	92,23%
	v-36	3				64,256	TRUE, FALSE	83,39%
	v-25	3				128,128	FALSE, FALSE	75,91%
	v-26	3				128,128	FALSE, TRUE	71,81%
	v-27	3				128,128	TRUE, FALSE	74,55%
	v-32	3				784,128	FALSE, TRUE	9,91%
	v-31	3				784,128	FALSE, FALSE	72,17%
	v-33	3				512,16	FALSE, TRUE	87,18%
	v-28	3				128,128,128	F, T, T	9,91%
	v-29	3				128,128,128	F, F, T	14,02%
	v-30	3				128,128,128	F, F, F	54,78%
	v-34	3				128,128,128	T, F, T	9,91%
Number of neurons	s-46	3				784		47,90%
	s-46b	5				784		65,70%
	s-46t	10				784		84,80%
	s-46q	15				784		85,70%
	s-46c	20				784		86,10%

Paramètre du réseau :

- **Nombre d'image d'entraînement** (nb_train_samples) Nombre d'images que l'on présente au réseau pendant la phase d'entraînement.
- **Nombre d'image de test** (nb_test_samples) Nombre d'images que l'on présente au réseau pendant la phase de test.
- **Taille de batch** (batch_size) Nombre d'images présentées entre chaque rétropropagation d'erreur.
- **Taux d'apprentissage** (learning_rate) Permet de réguler l'apprentissage pendant la rétropropagation de l'erreur. Cela permet d'éviter d'atteindre des valeurs de poids trop élevées trop rapidement.
- **Nombre de neurones par couches cachées** et **Nombre de couches cachées** (size_hidden_layers) Nombre de couche de neurone entre la couche d'entrée (présentation de l'image) et la couche de sortie (prédiction du label)
- **Nombre de fois où l'on présente le jeu de données** (nb_epoch) Permet d'améliorer l'apprentissage sur certains types d'images et certaines configuration.
- **Apprentissage sur les couches** (trainable_layers) Nous pouvons activer ou désactiver le mécanisme de rétropropagation de l'erreur sur chacune de couche du réseau.
- **Temps de présentation d'une image au réseau** (duration_per_image) Temps total pendant lequel une image génère des décharges.
- **Discrétisation du temps** (delta_t) La mise à jour des calculs est effectuée tous les pas de temps delta t.
- **Type de fonction de non linéarité** (surrogate_gradient) Cinq types de fonctions de non linéarité sont étudiées. Une fonction relu, 2 fonctions continues par morceaux, et 2 fonctions sigmoïdes rapides.
- **Valeur alpha** (surrogate_alpha) de la fonction de subsitution.

Répartition des tâches

Choix des types de dérivée de non linéarité : Antoine Marion, Soline Bernard
Adaptation code python : Victor Yon
implémentation, choix et affichage des graphiques : Antoine Marion
Simulations et tests : Antoine Marion, Soline Bernard, Victor Yon
Poster et interprétation des résultats : Antoine Marion, Soline Bernard, Victor Yon