

MISE À JOUR MODERNE DES POIDS
RECENT ADVANCES IN MACHINE LEARNING

Théo Lopès-Quintas

BPCE Payment Services,
Université Paris Dauphine

24 mai 2024

QUEL SCHEMA D'OPTIMISATION CHOISIR ?

- 1 Quel schéma d'optimisation choisir? 1**
 - 1.1 Avec momentum 4
 - 1.2 AdaGrad 5
 - 1.3 RMSProp 7
 - 1.4 Adam 8
- 2 Régularisations 12
 - 2.1 Weight Decay 13
 - 2.2 Gradient clipping 15
- 3 Normalisation d'un réseau de neurones 17
 - 3.1 Couche *Batch Normalization* 18
 - 3.2 Layer Normalization 22
 - 3.3 RMSNorm 24

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

FORMULATION D'UN PROBLÈME DE MACHINE LEARNING

Dans le cadre supervisé, nous avons accès à un dataset \mathcal{D} défini comme :

$$\mathcal{D} = \left\{ (x_i, y_i) \mid \forall i \leq n, x_i \in \mathbb{R}^{d'}, y_i \in \mathcal{Y} \right\}$$

Nombre d'observations Nombre d'informations

Avec $\mathcal{Y} \subseteq \mathbb{R}$ pour un problème de régression et $\mathcal{Y} \subset \mathbb{N}$ dans le cadre d'une classification. Les problèmes de Machine Learning supervisé peuvent souvent s'écrire sous la forme d'une optimisation d'une fonction de perte $\mathcal{L} : \mathbb{R}^d \times \mathcal{M}_{n,d'} \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ comme :

Vecteur des paramètres optimaux

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta, X, y)$$

Dimension du vecteur de paramètres

Dans la suite, pour simplifier les notations, nous omettrons la dépendance de \mathcal{L} en X (matrice des informations) et y (vecteur réponse). Notons qu'en général, nous avons $d \neq d'$ et dans le cas du deep learning, très souvent $d \gg d'$.

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

MÉTHODE

La méthode la plus utilisée pour résoudre ce genre de problème est la descente de gradient :

$$\theta_{t+1} = \theta_t - \underbrace{\eta_t}_{\text{Learning rate}} \nabla \mathcal{L}(\theta_t)$$

Quand on travaille avec des grands datasets, le coût de calcul/temps est grand si l'on calcule $\nabla \mathcal{L}(\theta_t)$ pour la totalité de la base. On peut donc considérer d'autres approches :

- **Stochastique (SGD)** : on sélectionne au hasard une observation et on met à jour θ
- **Stochastique par batch** : on sélectionne aléatoirement une partie de la base (batch) et on met à jour θ à chaque batch

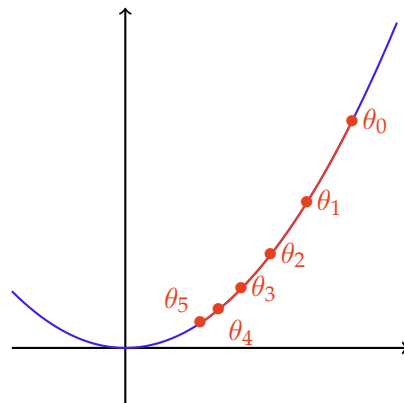


Figure – Exemple d'une descente de gradient pour $f(x) = x^2$

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

AVEC MOMENTUM

Paramètre du *momentum*

$$\begin{cases} v_{t+1} &= \gamma_t v_t + (1 - \gamma_t) \nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta_t v_{t+1} \end{cases} \quad (1)$$

Vecteur de vélocité

Avec cette version, on conserve dans la mise à jour des poids la *tendance* de déplacement des poids dans l'espace des paramètres pour accélérer la descente. A noter que la valeur du momentum γ_t doit être dans l'intervalle $[0, 1]$.

La descente de gradient avec momentum peut parfois *rater* le minimum et faire machine arrière. Ce phénomène peut être mitigé à l'aide d'un choix précis du *learning rate*. Cet hyper-paramètre est de loin le paramètre le plus sensible sur l'ensemble des schémas que nous présenterons

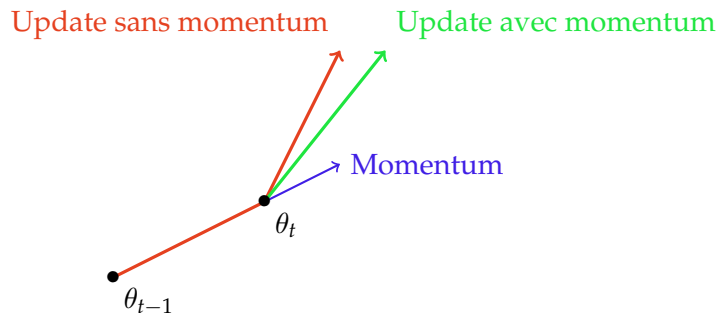


Figure – Effet du momentum sur la mise à jour des poids

QUEL SCHEMA D'OPTIMISATION CHOISIR ?

ADAGRAD : UNE PREMIERE REPONSE

$$\begin{cases} g_{t+1} = g_t + \nabla \mathcal{L}(\theta_t)^2 & \text{avec } g_0 = 0 \\ \theta_{t+1} = \theta_t - \eta \frac{\nabla \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \varepsilon} \end{cases} \quad (\text{AdaGrad, 2011})$$

Nombre pour éviter des problèmes numériques

Si le gradient a une magnitude plus importante dans une direction, elle sera privilégiée pendant l'optimisation.

AdaGrad [Duchi et al., 2011] propose de normaliser le gradient pour avoir un apprentissage *uniforme*.

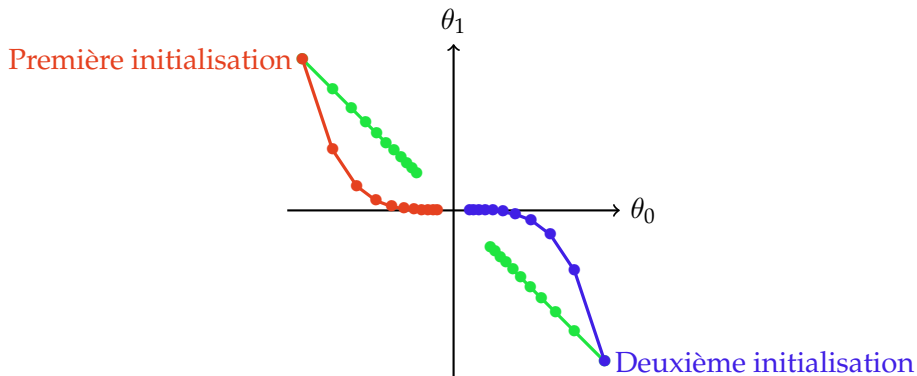


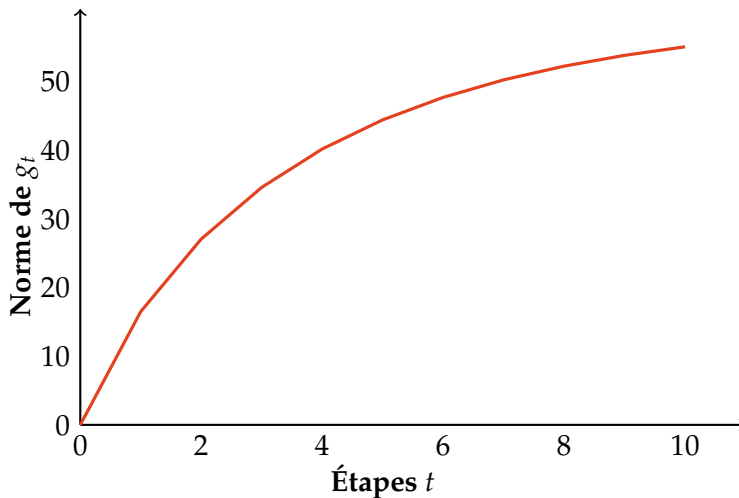
Figure – 10 étapes de descente de gradient pour la fonction $f(x, y) = x^2 + 3y^2$ avec **AdaGrad** pour deux initialisations différentes

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

ADAGRAD : RÉPONSE IMPARFAITE

$$\begin{cases} g_{t+1} = g_t + \nabla \mathcal{L}(\theta_t)^2 & \text{avec } g_0 = 0 \\ \theta_{t+1} = \theta_t - \eta \frac{\nabla \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \epsilon} \end{cases} \quad (\text{AdaGrad, 2011})$$

Cependant avec ce schéma, AdaGrad tend à avoir un apprentissage qui ralentit au fil de l'entraînement puisque $(g_t)_t$ est croissante.



QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

RMSProp : DEUXIÈME RÉPONSE

Contrôle la *mémoire* des précédents gradients, $\alpha \geq 0$

$$\begin{cases} g_{t+1} = \alpha g_t + (1 - \alpha) \nabla \mathcal{L}(\theta_t)^2 & \text{avec } g_0 = 0 \\ \theta_{t+1} = \theta_t - \eta \frac{\nabla \mathcal{L}(\theta_t)}{\sqrt{g_{t+1}} + \varepsilon} \end{cases} \quad (\text{RMSProp, 2012})$$

RMSProp [Hinton et al., 2012] cherche aussi à permettre un apprentissage uniforme dans toutes les directions, mais le fait avec une moyenne mobile exponentielle. Cela permet à $(g_t)_t$ de pouvoir décroître.

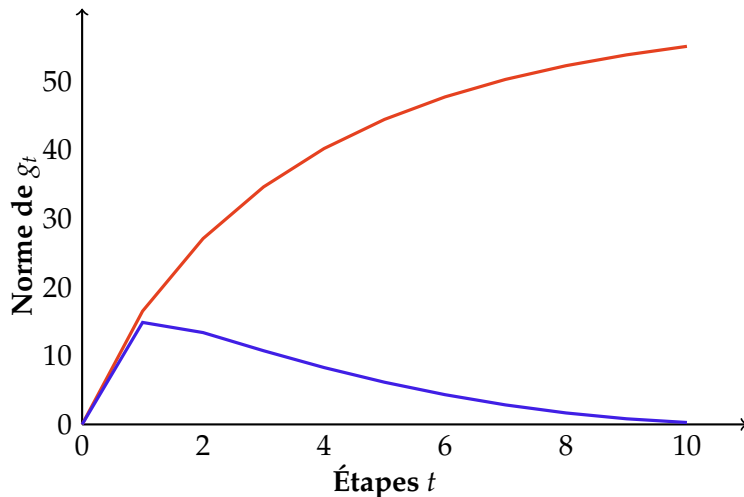


Figure – Norme de g_t pour RMSProp et AdaGrad

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

ADAM : COMBINER ADA GRAD ET RMS PROP

$$\left\{ \begin{array}{lcl} m_{t+1} & = & \beta_1 m_t + (1 - \beta_1) \nabla \mathcal{L}(\theta_t) \quad \text{avec } m_0 = 0 \\ \hat{m}_{t+1} & = & \frac{m_{t+1}}{1 - \beta_1^{t+1}} \\ v_{t+1} & = & \beta_2 v_t + (1 - \beta_2) \nabla \mathcal{L}(\theta_t)^2 \quad \text{avec } v_0 = 0 \\ \hat{v}_{t+1} & = & \frac{v_{t+1}}{1 - \beta_2^{t+1}} \\ \theta_{t+1} & = & \theta_t - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \varepsilon} \end{array} \right. \quad (\text{Adam, 2014})$$

Adam [Kingma and Ba, 2015] s'est rapidement imposé comme un excellent schéma d'optimisation pour l'apprentissage d'un réseau de neurones. Il ressemble à une version accélérée de RMSProp, étudions plus en détail ses moyennes mobiles.

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

ADAM : CORRECTION DU BIAIS DE L'INITIALISATION

Si l'on reprend une partie d'Adam, par exemple pour la suite $(v_t)_{t \in \mathbb{N}}$:

$$\begin{cases} v_{t+1} &= \beta_2 v_t + (1 - \beta_2) \nabla \mathcal{L}(\theta_t)^2 \quad \text{avec } v_0 = 0 \\ \hat{v}_{t+1} &= \frac{v_{t+1}}{1 - \beta_2^{t+1}} \end{cases}$$

On remarque que :

$$\begin{aligned} v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \mathcal{L}(\theta_t)^2 \\ &= \beta_2^2 v_{t-2} + (1 - \beta_2) \left[\beta_2 \mathcal{L}(\theta_{t-1})^2 + \mathcal{L}(\theta_t)^2 \right] \\ &= \beta_2^t v_0 + (1 - \beta_2) \sum_{i=1}^t \beta_2^{t-i} \mathcal{L}(\theta_i)^2 \end{aligned}$$

Si l'on suppose que les gradients sont identiquement distribués, on obtient :

$$\mathbb{E}[v_t] = (1 - \beta_2) \mathbb{E} \left[\mathcal{L}(\theta_t)^2 \right] \sum_{i=1}^t \beta_2^{t-i} \iff \mathbb{E}[v_t] = \mathbb{E} \left[\mathcal{L}(\theta_t)^2 \right] (1 - \beta_2^t)$$

D'où on déduit la correction apportée par la suite $(\hat{v}_t)_{t \in \mathbb{N}}$. Le calcul est le même pour la suite $(\hat{m}_t)_{t \in \mathbb{N}}$.

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

ADAM : INTERPRÉTATION DU LEARNING RATE

Puisque les suites $(\hat{v}_t)_{t \in \mathbb{N}}$ et $(\hat{m}_t)_{t \in \mathbb{N}}$ sont des estimateurs non biaisés de $\mathbb{E}[\mathcal{L}(\theta_t)]$ et $\mathbb{E}[\mathcal{L}(\theta_t)^2]$, on doit pouvoir obtenir une information sur la mise à jour des paramètres.

On considère une variable aléatoire X telle que $\mathbb{E}[X]$ existe et $\mathbb{E}[X^2]$ existe et n'est pas nulle. Alors, puisque la fonction $x \mapsto x^2$ est une fonction convexe, avec l'inégalité de Jensen on a :

$$\mathbb{E}[X^2] \geq \mathbb{E}[X]^2 \iff \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]} \leq 1 \iff \frac{|\mathbb{E}[X]|}{\sqrt{\mathbb{E}[X^2]}} \leq 1$$

Dans notre cas, avec les mêmes hypothèses pour $\mathcal{L}(\theta_t)$ que pour X , on définit $\Delta_t = \theta_t - \theta_{t-1}$ et on a :

$$\begin{aligned} \Delta_t &= \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}} \quad \text{donc} \quad |\Delta_t| = \eta \frac{|\hat{m}_t|}{\sqrt{\hat{v}_t}} \\ |\Delta_t| &\sim \eta \frac{|\mathbb{E}[\mathcal{L}(\theta_t)]|}{\sqrt{\mathbb{E}[\mathcal{L}(\theta_t)]^2}} \quad \text{ainsi} \quad |\Delta_t| \lesssim \eta \end{aligned}$$

QUEL SCHÉMA D'OPTIMISATION CHOISIR ?

QUEL OPTIMIZER CHOISIR ?

De nombreuses autres possibilités existent, et chaque année plusieurs optimiseurs sont proposés. C'est pourquoi [Schmidt et al., 2021] propose de comparer équitablement quinze optimiseurs sur différentes tâches. Il en ressort deux informations :

- ▶ Adam est l'optimiseur le plus performant sur le plus de tâches, sans pour autant être clairement supérieur. RMSProp et l'accélération de Nesterov restent des alternatives très intéressantes.
- ▶ Choisir les meilleurs hyperparamètres est tout aussi efficace voire plus efficace que de changer d'optimiseur.

Perhaps the most important takeaway from our study is hidden in plain sight : the field is in danger of being drowned by noise. Different optimizers exhibit a surprisingly similar performance distribution compared to a single method that is re-tuned or simply re-run with different random seeds. It is thus questionable how much insight the development of new methods yields, at least if they are conceptually and functionally close to the existing population.

— Robin Schmidt, Frank Schneider et Philipp Hennig (2021)

RÉGULARISATIONS

- 1 Quel schéma d’optimisation choisir? 1
 - 1.1 Avec momentum 4
 - 1.2 AdaGrad 5
 - 1.3 RMSProp 7
 - 1.4 Adam 8
- 2 Régularisations 12
 - 2.1 Weight Decay 13
 - 2.2 Gradient clipping 15
- 3 Normalisation d’un réseau de neurones 17
 - 3.1 Couche *Batch Normalization* 18
 - 3.2 Layer Normalization 22
 - 3.3 RMSNorm 24

RÉGULARISATIONS

PROBLÈME DE LA RÉGULARISATION

[Krogh and Hertz, 1991] montre qu'avoir un réseau de neurones avec une magnitude de poids faible permet de limiter le sur-apprentissage.

Une manière classique pour régulariser un réseau de neurones est de modifier la fonction de perte que l'on optimise. Le plus souvent on exploite la régularisation \mathcal{L}_2 :

$$\mathcal{L}_\lambda(w) = \mathcal{L}(w) + \frac{\lambda}{2} \|w\|_2^2 \quad \text{avec } \lambda \geq 0$$

Dans le cas d'une descente de gradient classique, cela donne le schéma d'optimisation :

$$\begin{aligned} w_{t+1} &= w_t - \eta_t \nabla \mathcal{L}_\lambda(w_t) && \text{par définition de la descente de gradient} \\ &= w_t - \eta_t \nabla \mathcal{L}(w_t) - \eta_t \lambda w_t && \text{par définition de } \mathcal{L}_\lambda \end{aligned}$$

Le terme supplémentaire $\eta_t \lambda w_t$ est appelé le weight decay.

RÉGULARISATIONS

PROBLÈME DE LA RÉGULARISATION

Mais si l'on considère une descente de gradient avec momentum :

$$\begin{aligned}v_{t+1} &= \beta v_t + (1 - \beta) \nabla \mathcal{L}_\lambda(w_t) && \text{par définition} \\&= \beta v_t + (1 - \beta) [\nabla \mathcal{L}(w_t) + \lambda w_t] && \text{par définition de } \mathcal{L}_\lambda\end{aligned}$$

$$\begin{aligned}w_{t+1} &= w_t - \eta_t (\beta v_t + (1 - \beta) [\nabla \mathcal{L}(w_t) + \lambda w_t]) \\&= w_t - \eta_t \beta v_t - \eta_t (1 - \beta) \nabla \mathcal{L}(w_t) - \eta_t (1 - \beta) \lambda w_t\end{aligned}$$

Autrement dit, nous avons une propagation de la régularisation dans le schéma de descente, ce qui n'est pas souhaité. Pour conserver le comportement observé dans une descente de gradient classique, [Loshchilov and Hutter, 2019] propose simplement de modifier le schéma de descente plutôt que la fonction de perte.

Ainsi, dès lors que l'on utilise une méthode *adaptive*, il faut préférer le **weight decay** à la régularisation \mathcal{L}_2 . L'omniprésence d'Adam dans la littérature a amené à la création d'AdamW qui est Adam intégrant le weight decay.

RÉGULARISATIONS

GRADIENT CLIPPING

Le weight decay cherche à contraindre la norme des poids, mais dans un entraînement avec une fonction de perte hautement non convexe, il est possible que des gradients de norme importante soit propagé : c'est l'explosion des gradients.

Une première manière de réduire ce phénomène est d'imposer que chacune des coordonnées, indépendamment, soit comprise dans $[-\alpha, \alpha]$ avec $\alpha \in \mathbb{R}^+$: c'est le **gradient clipping**. On risque cependant de ne plus conserver la *direction* du vecteur.

Une seconde manière est d'imposer une norme maximale au vecteur : c'est le **gradient clipping par norme**.

$$\tilde{g} = \begin{cases} \alpha \frac{g}{\|g\|} & \text{si } \|g\| \geq \alpha \\ g & \text{sinon} \end{cases}$$

Cependant on risque de ne pas apprendre efficacement si une des coordonnées *écrase* les autres.

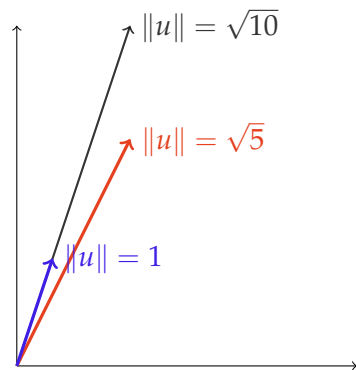
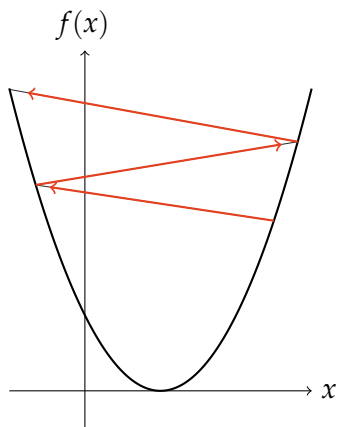


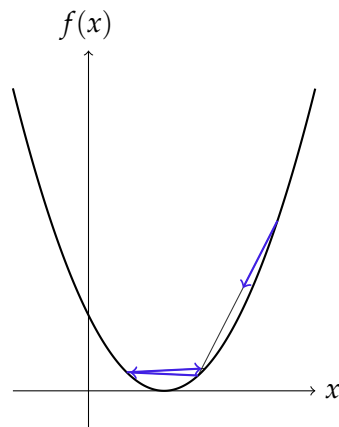
Figure – Gradient initial, **gradient** après clipping et **gradient** après clipping par norme

RÉGULARISATIONS

GRADIENT CLIPPING - EXEMPLE



(a) Gradients sans gradient clipping par norme



(b) Gradients avec gradient clipping par norme

Figure – Descente de gradient pour $f(x) = (x - 1)^2$ et un learning rate de 1.05

NORMALISATION D'UN RÉSEAU DE NEURONES

- 1 Quel schéma d'optimisation choisir? 1
 - 1.1 Avec momentum 4
 - 1.2 AdaGrad 5
 - 1.3 RMSProp 7
 - 1.4 Adam 8
- 2 Régularisations 12
 - 2.1 Weight Decay 13
 - 2.2 Gradient clipping 15
- 3 Normalisation d'un réseau de neurones 17
 - 3.1 Couche *Batch Normalization* 18
 - 3.2 Layer Normalization 22
 - 3.3 RMSNorm 24

NORMALISATION D'UN RÉSEAU DE NEURONES

COUCHE *Batch Normalization*

Sergey Ioffe et Christian Szegedy publie en 2015 l'article *Batch normalization : Accelerating deep network training by reducing internal covariate shift* [Ioffe and Szegedy, 2015] où l'on peut lire :

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities

— Sergey Ioffe, Christian Szegedy (2015)

Ce phénomène est appelé l'*internal covariate shift* et fait partie de la famille des *shifts* qui existent en Machine Learning¹. Pour essayer de contrer ses effets est proposé une nouvelle couche dans un réseau de neurones : Batch Normalization.

1. Il s'agit de l'ensemble des changements qui peuvent se produire pendant l'entraînement ou l'utilisation d'un algorithme. Par exemple le changement de distribution entre le dataset de train et le dataset de test.

NORMALISATION D'UN RÉSEAU DE NEURONES

COUCHE *Batch Normalization* : PREMIÈRE APPROXIMATION

L'idée est de pousser l'idée de normaliser les inputs à l'ensemble des couches, pas seulement la première. Si nous le faisons systématiquement avec l'ensemble du dataset, alors nous devons calculer les nouvelles valeurs pour normaliser pour chaque couche à chaque époques. Donc calculer des matrices de variance-covariance ainsi que leurs racine carré inverse : c'est très coûteux. Pour tout de même approcher l'idée de normaliser chaque couche, l'article propose une première simplification. La première est que chaque dimensions du vecteur d'input sera normalisée indépendamment :

Dimension k du vecteur input $x \in \mathbb{R}^d$

$$\hat{x}^{(k)} = \frac{x^{(k)} - \overline{x^{(k)}}}{\sigma_{x^{(k)}}}$$

Ecart-type de $x^{(k)}$ calculé sur le training set

Ce genre de transformation, appliqué à chaque couche, peut changer ce que chaque couche *représente*. Normaliser les inputs d'une fonction d'activation sigmoid les contraints à être proche de 0 donc du régime presque linéaire de sigmoid.

NORMALISATION D'UN RÉSEAU DE NEURONES

COUCHE *Batch Normalization* : SECONDE APPROXIMATION

Pour ne pas le perdre, on introduit deux paramètres pour chaque couche $l \leq L$ que le réseau va apprendre pour permettre de conserver le pouvoir de représentation du réseau. Si l'on note z l'output de la couche Batch-Normalization, on a :

$$z^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Les paires $(\gamma^{(k)}, \beta^{(k)})$ sont apprises avec le modèle. Si l'on utilise une descente de gradient stochastique alors on ne peut plus utiliser l'ensemble du dataset pour normaliser. C'est ici qu'intervient la seconde simplification : chaque mini-batch \mathcal{B} produit une estimation de la moyenne et de la variance.

The diagram illustrates the calculation of the normalized input \hat{x}_i for a mini-batch \mathcal{B} . It shows the following components:

- The batch mean $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{j=1}^m x_j$ is calculated from the batch elements x_j .
- The batch variance $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{j=1}^m (x_j - \mu_{\mathcal{B}})^2$ is calculated from the batch elements x_j and the batch mean $\mu_{\mathcal{B}}$.
- The normalized input \hat{x}_i is calculated as $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$.
- A note indicates that $\epsilon > 0$ is used to prevent numerical instabilities.

Et le résultat de la couche de batch-normalization comme : $z_i = \gamma \hat{x}_i + \beta$

NORMALISATION D'UN RÉSEAU DE NEURONES

COUCHE *Batch Normalization* : EN PRATIQUE

L'introduction de cette nouvelle brique dans l'architecture d'un réseau de neurones permet d'accélérer sa convergence, de manière saine, si l'on modifie également le reste des paramètres. Dans l'article, et dans la pratique, il est conseillé quand on utilise Batch Normalization d'augmenter le learning rate et de supprimer ou réduire le dropout par exemple.

Position de la couche

Si l'article est clair sur la position de cette couche par rapport à l'activation (fonction d'activation puis Batch Normalization), dans la pratique ce n'est pas ce qui semble fonctionner le mieux. En témoigne le commentaire de François Chollet² suite à une question sur GitHub :

I haven't gone back to check what they are suggesting in their original paper, but I can guarantee that recent code written by Christian [Szegedy] applies relu before BN. It is still occasionally a topic of debate, though.

— François Chollet (2016)

Par expérience, nous conseillons de suivre cette recommandation.

2. François Chollet est un chercheur chez Google qui a, entre autre, écrit Keras et un des livres références (si ce n'est le) sur le Deep Learning.

NORMALISATION D'UN RÉSEAU DE NEURONES

LAYER NORMALIZATION

Une petite taille de batch peut donner lieu à des estimations erronées. De plus, il n'est pas évident comment nous pouvons appliquer la couche BatchNormalization à des réseaux de neurones récurrents. Partant de ce constat, [Ba et al., 2016] propose la couche **Layer Normalization** : au lieu de normaliser chaque features indépendamment, normalisons chaque observations du batch indépendamment !

Formellement, si l'on considère un vecteur avec d features, Layer Normalization normalise comme suit :

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \quad (2)$$

$\mu_l = \frac{1}{d} \sum_{j=1}^d x_j$

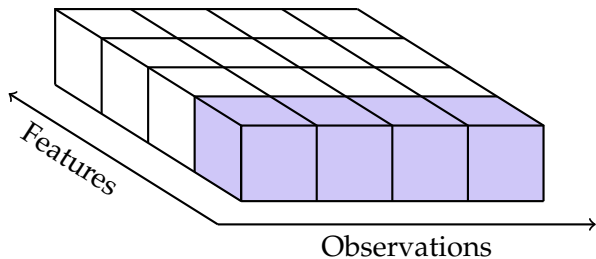
$\sigma_l^2 = \frac{1}{d} \sum_{j=1}^d (x_j - \mu_l)^2$

$\epsilon > 0$ pour prévenir les instabilités numériques

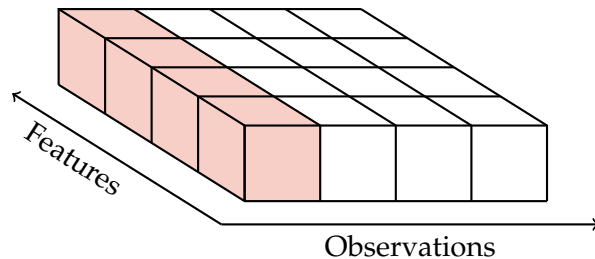
Puis, de manière identique à Batch-Normalization pour conserver la puissance de représentation du réseau, l'output de la couche est définie comme $z_i = \gamma \hat{x}_i + \beta$ avec γ et β des paramètres appris en même temps que les autres paramètres du modèle.

NORMALISATION D'UN RÉSEAU DE NEURONES

LAYER NORMALIZATION



(a) Batch Normalization



(b) Layer Normalization

Figure – Pour un dataset donné, comparaison de la normalisation entre les deux méthodes décrites

NORMALISATION D'UN RÉSEAU DE NEURONES

RMSNorm

La couche Layer Normalization induit un surplus de calcul qui peut devenir coûteux quand on entraîne des LLM. [Zhang and Sennrich, 2019] propose alors **RMSNorm** dont l'hypothèse de départ est que la stabilisation de l'entraînement ne vient pas du retrait de la moyenne, mais de la mise à l'échelle.

La question du placement de la couche de BatchNormalization se pose également pour la couche LayerNorm et RMSNorm. [Xiong et al., 2020] préconise de faire précéder directement les blocs d'attention et de Feed Forward Network par la couche de normalisation.

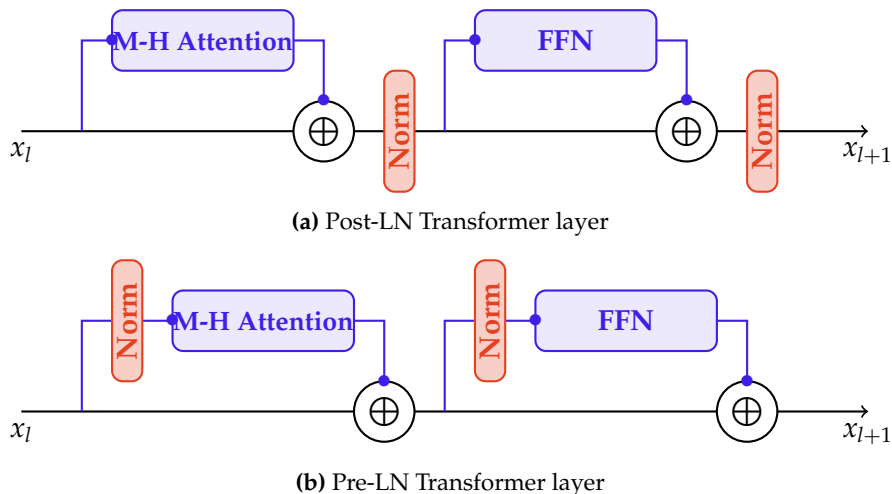












Figure – Placement de la couche de normalisation dans l'architecture Transformers

BIBLIOGRAPHIE I

-  Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016).
Layer normalization.
arXiv preprint arXiv :1607.06450.
-  Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., Pham, H., Dong, X., Luong, T., Hsieh, C.-J., et al. (2023).
Symbolic discovery of optimization algorithms.
arXiv preprint arXiv :2302.06675.
-  Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of machine learning research.
-  Hinton, G., Srivastava, N., and Swersky, K. (2012).
Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
Cited on.
-  Ioffe, S. and Szegedy, C. (2015).
Batch normalization : Accelerating deep network training by reducing internal covariate shift.
International conference on machine learning.

BIBLIOGRAPHIE II

-  Kingma, D. P. and Ba, J. (2015).
Adam : A method for stochastic optimization.
In International Conference on Learning Representations (ICLR).
-  Krogh, A. and Hertz, J. (1991).
A simple weight decay can improve generalization.
Advances in neural information processing systems.
-  Loshchilov, I. and Hutter, F. (2019).
Decoupled weight decay regularization.
International Conference on Learning Representations (ICLR).
-  Schmidt, R. M., Schneider, F., and Hennig, P. (2021).
Descending through a crowded valley-benchmarking deep learning optimizers.
In International Conference on Machine Learning. PMLR.
-  Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. (2020).
On layer normalization in the transformer architecture.
In International Conference on Machine Learning. PMLR.

BIBLIOGRAPHIE III



Zhang, B. and Sennrich, R. (2019).

Root mean square layer normalization.


Advances in Neural Information Processing Systems, 32.

ANNEXE : COMPLÉMENTS

OPTIMIZER LION

[Chen et al., 2023] Une équipe de chercheur de Google et UCLA propose un nouvel optimizer nommé Lion (EvoLved Sign Momentum) [Chen et al., 2023]. Il est le fruit d'une recherche symbolique du meilleur optimizer possible. AdamW est le point de départ de la recherche. Cet optimizer s'écrit avec nos notations sous la forme :

$$\begin{cases} c_{t+1} &= \beta_1 m_t + (1 - \beta_1) \nabla \mathcal{L}(\theta_t) \quad \text{avec } m_0 = 0 \\ m_{t+1} &= \beta_2 m_t + (1 - \beta_2) \nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta_t (\text{sign}(c_t) + \lambda \theta_t) \end{cases} \quad (\text{Lion, 2023})$$

Weight decay, $\lambda \geq 0$

Lion ne travaille pas avec l'amplitude des gradients mais seulement le signe. On a également deux hyperparamètre β_1 et β_2 qui n'ont pas un rôle aussi clair que pour Adam. L'approche globale reste très proche de celle d'Adam et le comportement important du weight decay a été conservé.

Des améliorations sensibles sont décrites dans l'article, mais la communauté n'a pas trouvé les mêmes résultats de manière consistante, nous renvoyant à la conclusion de [Schmidt et al., 2021].

L'approche est cependant suffisamment innovante et l'utilisation uniquement du signe suffisent, à notre avis, pour que l'on connaisse ce travail.