Réseau de neurones - Compléments

Introduction au deep learning

Théo Lopes Quintas

BPCE Payment Services,
Université Paris Dauphine

2023-2026

Meilleure descente de gradient

Formulation d'un problème de Machine Learning

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

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

Nombre d'informations

Nombre d'observations

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 $\mathcal{L} : \mathbb{R}^d \times \mathcal{M}_{n,d'} \times \mathbb{R}^n \to \mathbb{R}_+$ comme :

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

Vecteur des paramètres optimaux

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 $d >> d'$.

Meilleure descente de gradient

Avec momentum

Paramètre du *momentum*

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

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

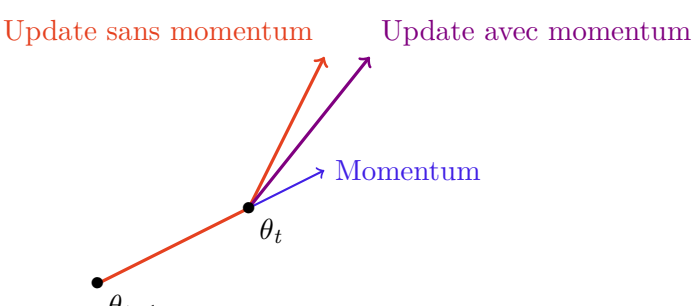


Figure – Effet du momentum sur la mise à jour des poids

Meilleure descente de gradient

AdaGrad : Une première réponse

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

(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*.

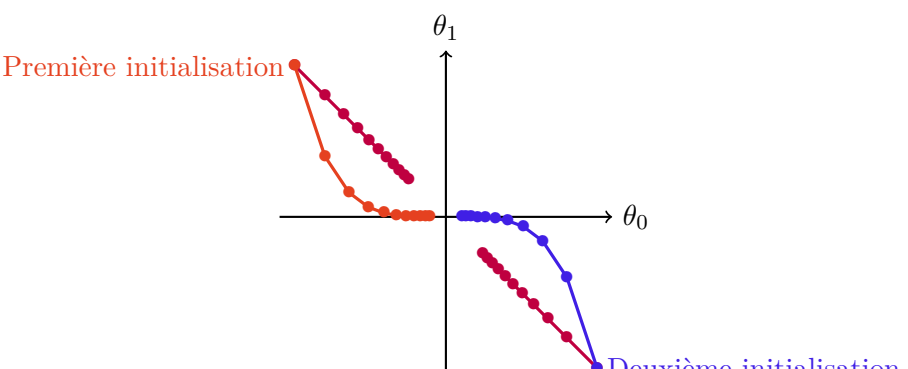


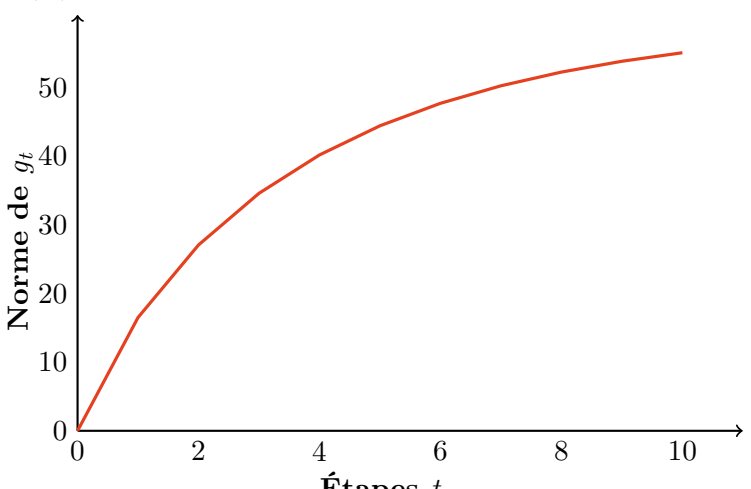
Figure – 10 étapes de descente de gradient pour la fonction $f(x, y) = x^2 + 3y^2$ avec AdaGrad pour deux initialisations différentes

Meilleure descente de gradient

AdaGrad : Réponse imparfaite

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

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



Meilleure descente de gradient

RMSProp : Deuxième réponse

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

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

(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.

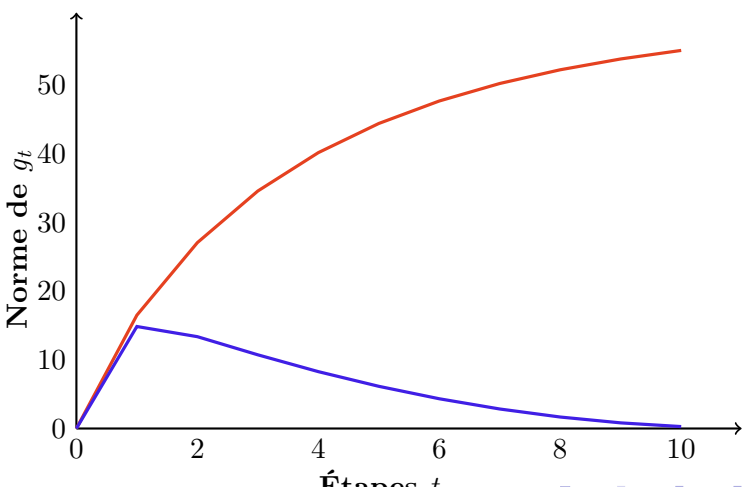


Figure – Norme de g_t pour RMSProp et AdaGrad

Meilleure descente de gradient

Adam : Combiner AdaGrad et RMSProp

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

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.

Meilleure descente de gradient

Adam : Correction du biais de l'initialisation

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

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

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 :

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

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

Meilleure descente de gradient

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 $\mathbb{E}\left[\mathcal{L}(\theta_t)\right]$ et $\mathbb{E}\left[\mathcal{L}(\theta_t)^2\right]$, on doit pouvoir obtenir une information sur la mise à jour des paramètres.
On considère une variable aléatoire X telle que $\mathbb{E}\left[X\right]$ existe et $\mathbb{E}\left[X^2\right]$ 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 :

$$\mathbb{E}\left[X^2\right] \geqslant \mathbb{E}\left[X\right]^2 \iff \frac{\mathbb{E}\left[X\right]^2}{\mathbb{E}\left[X^2\right]} \leqslant 1 \iff \frac{\left|\mathbb{E}\left[X\right]\right|}{\sqrt{\mathbb{E}\left[X^2\right]}} \leqslant 1$$

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 :

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

Meilleure descente de gradient

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 restant 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)

En résumé

Échéancier

Il faut un échéancier pour SGD

On se place dans le cadre d'une optimisation d'une fonction de perte $\mathcal{L}$ non convexe, mais β-smooth. On conserve les hypothèses (**??**) de l'annexe de la séance 1, et on note $\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$

Théorème 1 (Pas de descente décroissant)

On considère une descente de gradient stochastique avec η_t une suite décroissante telle que $\eta_t \in \left]0, \frac{1}{\beta}\right]$ et que :

$$\sum_{t=0}^{+\infty} \eta_t = +\infty \quad et \quad \sum_{t=0}^{+\infty} \eta_t^2 < +\infty$$

Alors, pour tout $T \geqslant 1$:

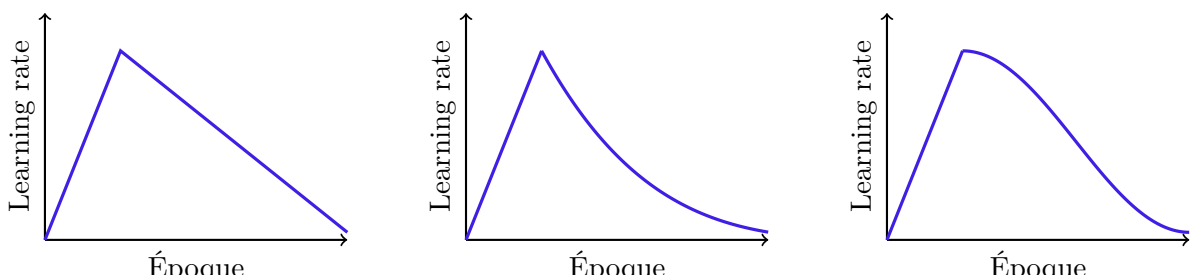
$$\lim_{T \to +\infty} \mathbb{E}\left[\frac{1}{\sum_{t=0}^{T-1} \eta_t} \sum_{t=0}^{T-1} \eta_t \|\nabla \mathcal{L}(\theta_t)\|^2 \right] = 0$$

Nous n'avons plus cette fois une convergence dans un intervalle proche du minimum, mais une convergence vers un point de gradient nul. D'où la nécessité d'avoir un échéancier pour le choix du learning rate. De même pour Nesterov, nous avions besoin d'un échéancier pour le momentum.

Échéancier

Classique

Les succès de l'ensemble des schémas précédents sont conditionnés à un bon choix du learning rate η, même dans le cas adaptatif. Il est possible de le conserver constant pour
la totalité de l'entraînement mais il existe des **échéanciers** pour modifier η au cours de l'entraînement.



(a) Linear Decay **(b)** Polynomial Decay **(c)** Cosinus Decay

Figure – Exemples d'échanciers du learning rate avec échauffements

Il n'y a pas de consensus sur l'utilisation ou non d'un échauffement. De même, ces échéanciers tendent vers 0, mais il est plus utile de le conserver à une valeur non nulle.

Échéancier

Par cycle

L'article *Cyclical Learning Rates for Training Neural Network* [Smith, 2017] propose une nouvelle manière de définir un échéancier [1]

> *The essence of this learning rate policy comes from the observation that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect*
>
> — Leslie Smith (2015)

La première manière de faire varier la valeur est triangulaire :

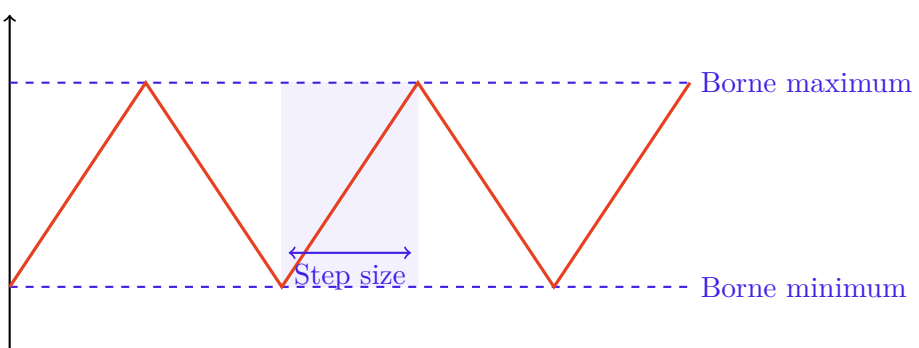


Figure – Échéancier triangulaire pour le learning rate

1. A noter que cette proposition est antérieure à *Attention is all you need*, elle s'applique donc à l'ensemble des réseaux par construction.

Échéancier

Par cycle et avec restart

L'échéancier cosinus introduit en 2016 dans l'article *SGDR : Stochastic gradient descent with warm restart* [Loshchilov and Hutter, 2016] est devenu standard dans les réseaux Transformers [2].

Nombre d'itérations réalisées dans le cycle i

$$\eta_t = \eta_{\min}^i + \frac{1}{2}\left(\eta_{\max}^i - \eta_{\min}^i\right)\left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right)$$

Nombre d'itérations à réaliser dans le cycle i

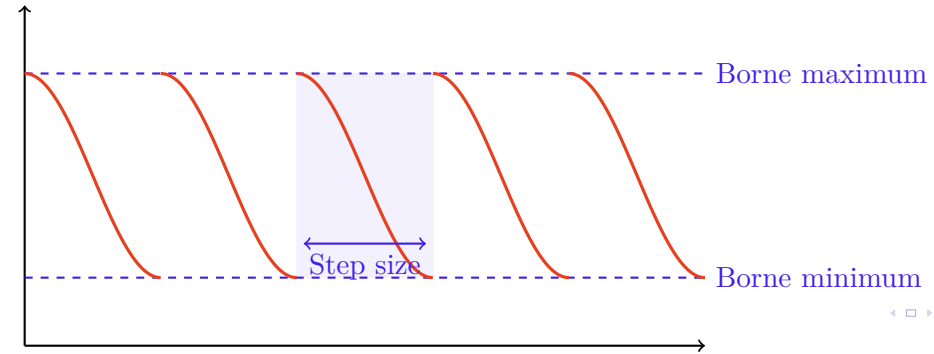


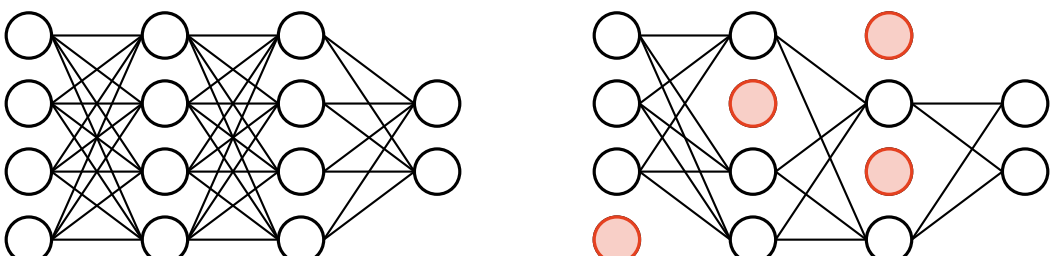
Figure – Échéancier cosinus pour le learning rate

En résumé

Régularisations

Dropout

Le dropout [Srivastava et al., 2014] est introduit en 2014 par Nitish Srivastava, Geoffrey Hinton, Alex krizhevsky, Ilya Sutskever et Ruslan Salakhutinov. Lors de l'entraînement, le dropout va *supprimer* de la chaîne d'entraînement des neurones avec une proportion $1 - p$ à chaque couche, dont l'input, sauf l'output. On obtient à chaque passe forward un sous-ensemble du réseau initial.



(a) Entraînement classique

(b) Entraînement avec dropout

Figure – Réseau de neurones avec et sans dropout

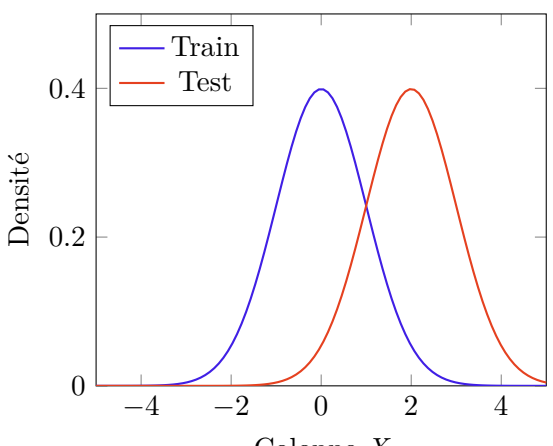
L'exploitation du réseau de neurones nécessite comportement déterministe. Classiquement, la prédiction finale d'un ensemble est construite en faisant la moyenne arithmétique des prédictions. Ici, conserver l'ensemble des réseaux est trop coûteux en mémoire, donc nous allons approcher ce comportement en faisant la moyenne géométrique : chaque poids est multipliés par la proportion p de dropout.

Régularisations - Parenthèse

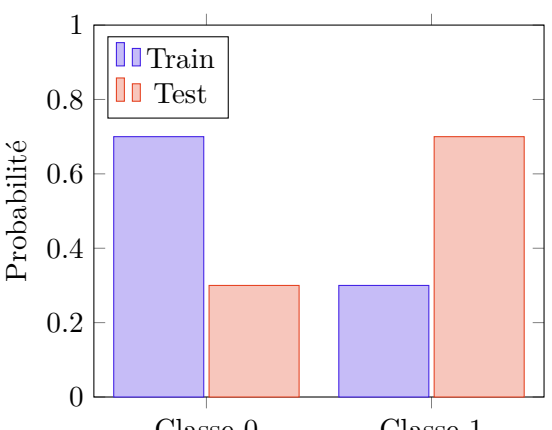
Les datasets shifts

On parle de **dataset shift** quand les distributions du dataset d'entraînement et du dataset de test sont différentes.

Covariate shift : changement de distribution des features X.



Prior probability shift : changement de distribution de la cible y.



Le **Concept shift** correspond à un changement de relation entre X et y.

Régularisations - Parenthèse

Les datasets shift : internal covariate shift

Un cas particulier du *covariate shift* est l'**internal covariate shift** : la distribution en sortie des neurones changent en *traversant* le réseau de neurones.

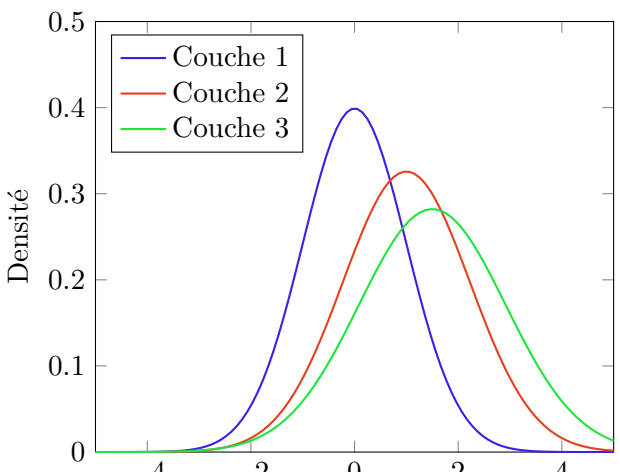


Figure – Internal covariate shift dans un réseau de neurone

Régularisations

Couche Batch Normalization

[Ioffe and Szegedy, 2015] propose la couche **BatchNormalization** qui standardise les données qu'elle reçoit. Calculer les matrices de variance-covariance étant coûteux, on normalise chaque dimension 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

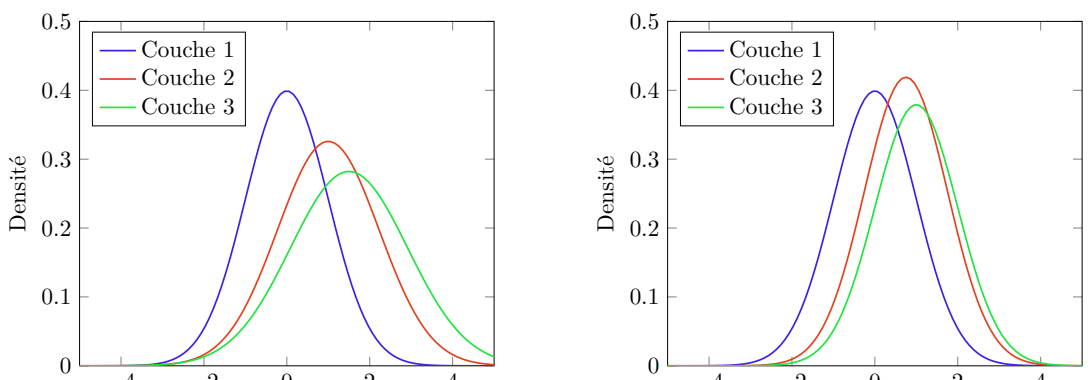
Régularisations

Couche Batch Normalization

Ce genre de transformation appliqué à une couche peut changer ce que la couche *représente*. Pour ne pas perdre cela, après la standardisation on calcule la sortie de la couche z comme :

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

Les paires $(\gamma^{(k)}, \beta^{(k)})$ sont apprises avec le modèle.



(a) Avant **(b)** Après

Figure – Internal covariate shift et BatchNormalization dans un réseau de neurone

Régularisations

Couche Batch Normalization

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.

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{j=1}^{m} x_j$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{j=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$\varepsilon > 0$ **pour prévenir les instabilités numérique**

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

L'introduction de celle nouvelle brique permet d'accélérer la convergence, de manière saine, si l'on modifie également le reste des paramètres. Dans l'article il est conseillé 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 Chollet [3] 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)

3. 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.

En résumé

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 $\mathcal{L}_2$:

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

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.

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

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

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

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 *adaptative*, 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.

En résumé

Bibliographie I

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.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022).
Training compute-optimal large language models.
arXiv preprint arXiv :2203.15556.

Ioffe, S. and Szegedy, C. (2015).
Batch normalization : Accelerating deep network training by reducing internal covariate shift.
International conference on machine learning.

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.

Bibliographie II


Loshchilov, I. and Hutter, F. (2016).
Sgdr : Stochastic gradient descent with warm restarts.
arXiv preprint arXiv :1608.03983.


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.


Smith, L. N. (2017).
Cyclical learning rates for training neural networks.
In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE.


Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
Dropout : a simple way to prevent neural networks from overfitting.
The journal of machine learning research.