

INTRODUCTION AUX RÉSEAUX DE NEURONES

Le processus d'apprentissage en pratique

Pascal Germain*, 2019

* Merci spécial à [Philippe Giguère](#) pour m'avoir permis de réutiliser une partie de ces transparents.

Le choc de la réalité

Algorithme de descente en gradient stochastique (avec momentum).

ENTRÉES: Ensemble d'apprentissage S , taux d'apprentissage η , vélocité α , nombre d'itérations T

- Initialiser $\mathbf{z} \in \mathbb{R}^d$ aléatoirement
- $\mathbf{v} \leftarrow 0$
- Pour t de 1 à T :
 - Choisir aléatoirement $i \in \{1, \dots, n\}$
 - $\mathbf{g} \leftarrow \nabla F_i(\mathbf{z})$
 - $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\eta}{\sqrt{t}} \mathbf{g}$
 - $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$
- Retourner \mathbf{z}

Apprentissage par «minibatch»

- La variance du gradient diminue avec la taille m de la minibatch

$$\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$$

- Pour $m=1$, on retrouve la descente de gradient stochastique standard
- Pour $m=n=|S|$, on retrouve la descente en gradient «batch» (non-stochastique)
- Valeurs typiques : 32, 64, 128, 256, 512
- **Avantage principal:** Augmente la vitesse de calcul grâce à la parallélisation des opérations matricielles sur CPU / GPU
- Mais la taille de la minibatch influence la solution obtenue...

Apprentissage par «minibatch»

- À chaque itération, le gradient est calculé sur un petit sous-ensemble de taille fixe de l'ensemble d'apprentissage

Algorithme de descente en gradient avec momentum – version minibatch.

ENTRÉES: Taux d'apprentissage η , vélocité α , taille de minibatch m , nombre d'époques T

- Initialiser $\mathbf{z} \in \mathbb{R}^d$ aléatoirement

- $\mathbf{v} \leftarrow 0$

- Pour t de 1 à T :

Un passage sur tous les exemples d'apprentissage se nomme une «époque» (en anglais: «epoch»)

- Diviser aléatoirement S en $\lfloor n/m \rfloor$ sous-ensembles disjoints $\hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$

Ce sont les «minibatch»

- Pour $\hat{S}_i = \hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$:

- * $\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$

Moyenne du gradient sur les m exemples de la «minibatch»

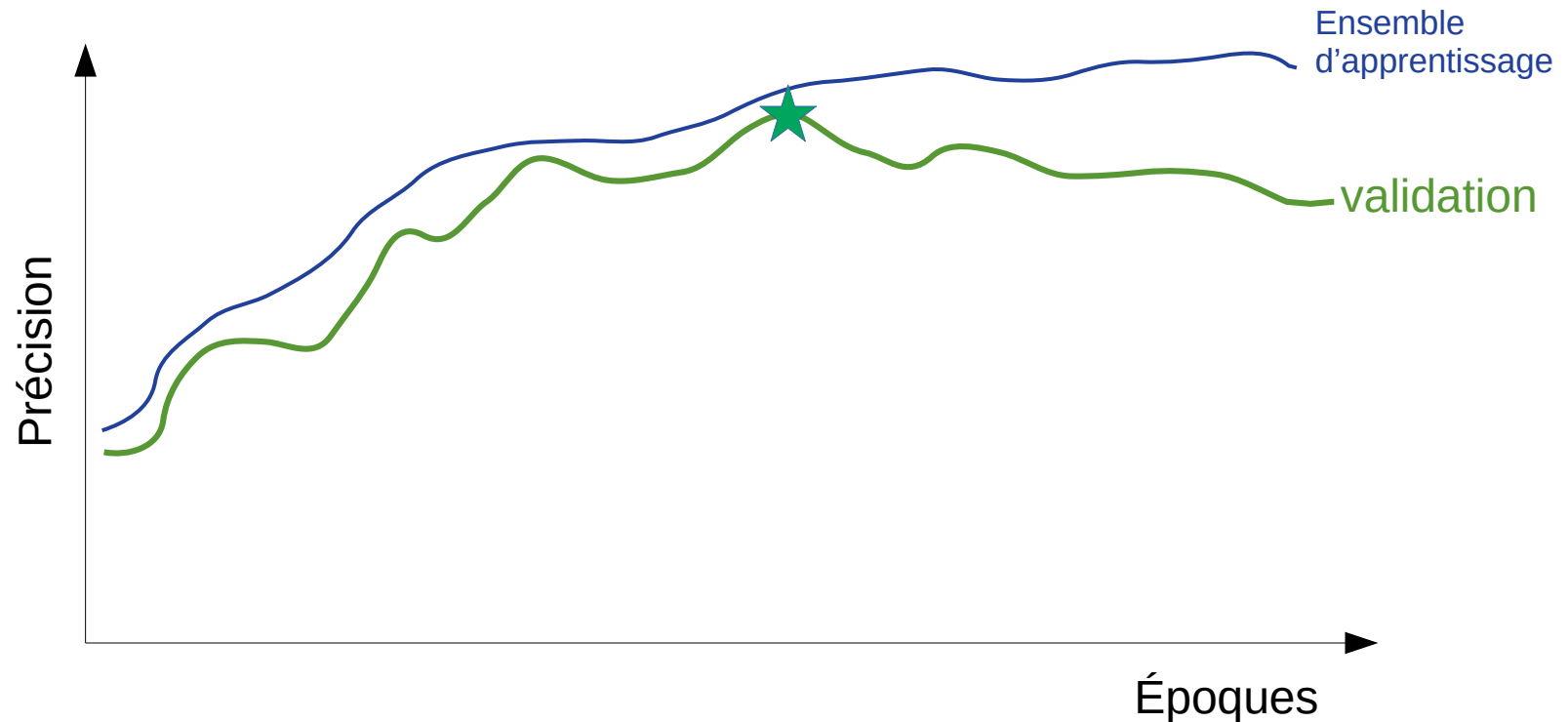
- * $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\eta}{\sqrt{t}} \mathbf{g}_t$

- * $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$

- Retourner \mathbf{z}

Le «early stopping»

- Séparer les données en un ensemble d'apprentissage S et un ensemble de validation S_v .
- Arrêter la descente de gradient lorsque la précision est maximale sur S_v .



La précision n'est pas nécessairement évaluée par la même métrique que la fonction de perte optimisée (exemple: précision zéro-un vs perte logistique)

Horaire d'apprentissage

- On désire généralement un taux d'apprentissage élevé en début d'apprentissage et un taux d'apprentissage bas vers la fin
- Toutefois, le taux de décroissance idéal dépend du problème
- En pratique, il est commun d'ajuster le taux d'apprentissage «à la main»

Algorithme de descente en gradient avec momentum – version horaire d'apprentissage

ENTRÉES: Ensemble d'apprentissage S , horaire d'apprentissage $\{\eta_t\}_{t=1}^T$,
vélocité α , taille de minibatch m , nombre d'époques T

- Initialiser $\mathbf{z} \in \mathbb{R}^d$ aléatoirement
- $\mathbf{v} \leftarrow 0$
- Pour t de 1 à T :
 - Diviser aléatoirement S en $\lfloor n/m \rfloor$ sous-ensembles disjoints $\hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$
 - Pour $\hat{S}_i = \hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$:
 - * $\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$
 - * $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta_t \mathbf{g}$
 - * $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$
- Retourner \mathbf{z}

Le «early stopping»

Algorithme de descente en gradient avec momentum – version early stopping.

ENTRÉES: Ensemble d'apprentissage S , ensemble de validation S_V , patience p
taux d'apprentissage η , vélocité α , taille de minibatch m

- Initialiser $\mathbf{z} \in \mathbb{R}^d$ aléatoirement

- $\mathbf{v} \leftarrow 0$

- $t \leftarrow 1, T \leftarrow p, \mathbf{z}^* \leftarrow \mathbf{z}$

\mathbf{z}^* représente le meilleur modèle rencontré

Le paramètre de «patience» remplace le nombre d'époques

- Tant que $t \leq T$:

– Diviser aléatoirement S en $\lfloor n/m \rfloor$ sous-ensembles disjoints $\hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$

– Pour $\hat{S}_i = \hat{S}_1, \dots, \hat{S}_{\lfloor n/m \rfloor}$:

- * $\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$

- * $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\eta}{\sqrt{t}} \mathbf{g}_t$

- * $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$

Si la précision du modèle représenté par les paramètres \mathbf{z} sur l'ensemble de Validation S_V est plus élevée que le meilleurs modèle rencontré jusqu'à maintenant

– Si $\text{Précision}(\mathbf{z}, S_V) > \text{Précision}(\mathbf{z}^*, S_V)$:

- * $\mathbf{z}^* \leftarrow \mathbf{z}$

- * $T \leftarrow t + p$

– $t \leftarrow t + 1$

Allons voir si la précision augmente au cours des p prochaines itérations

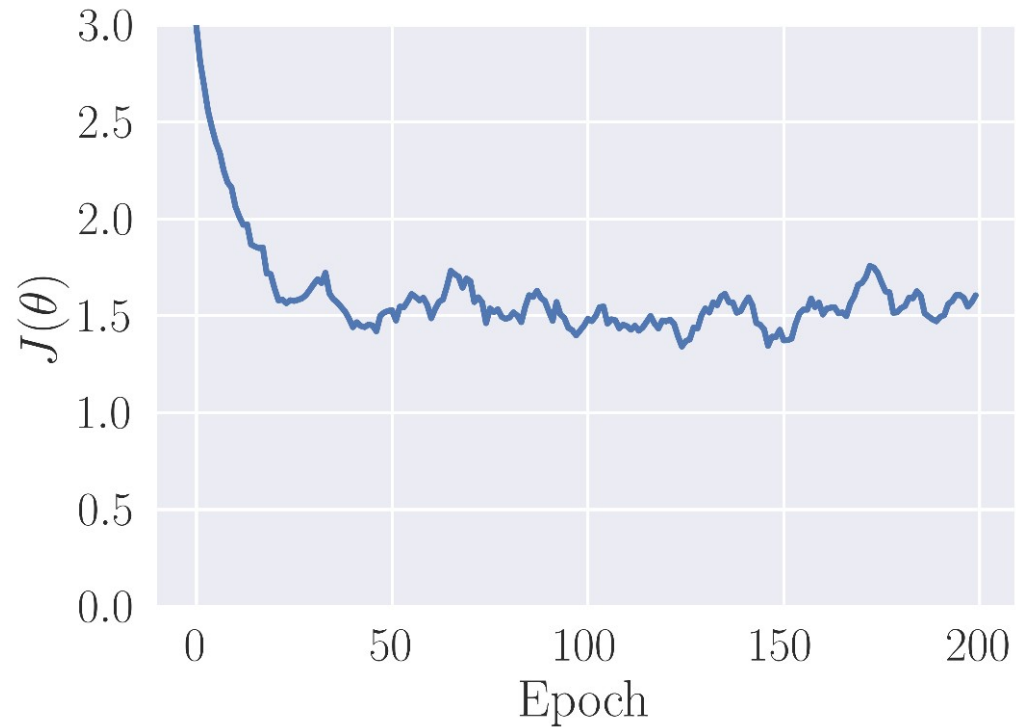
Tant que ma patience n'est pas épuisée !

Retourne le meilleur modèle rencontré sur l'ensemble de validation

- Retourner \mathbf{z}^*

Horaire d'apprentissage

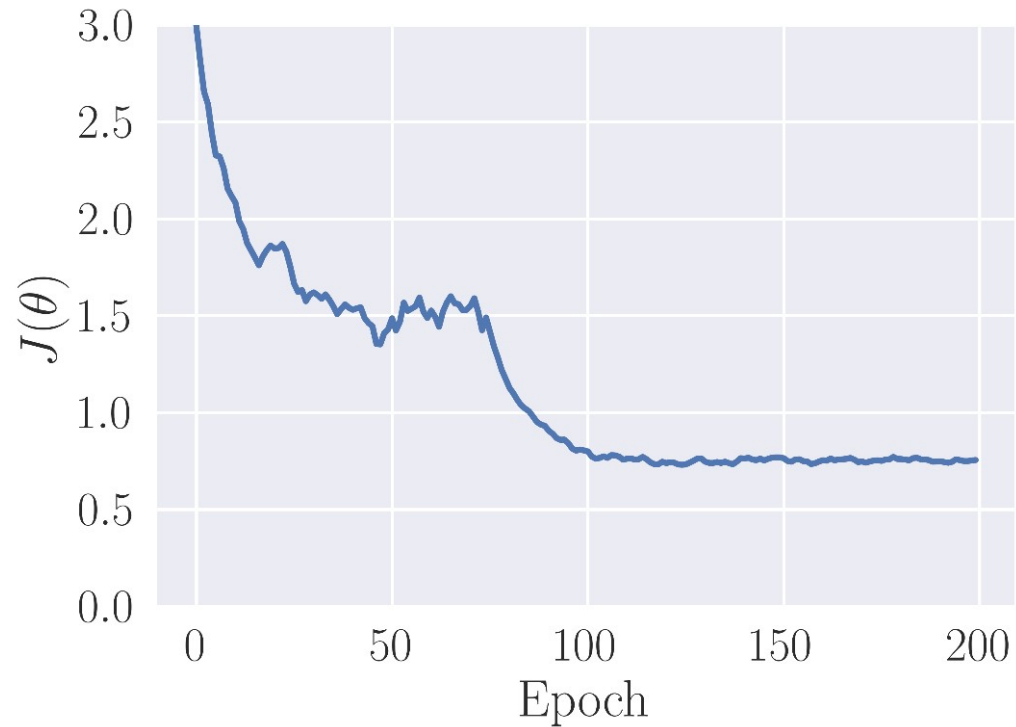
Convergence aux alentours de l'époque 75.



Époque	0			
Taux d'apprentissage	0.1			

Horaire d'apprentissage

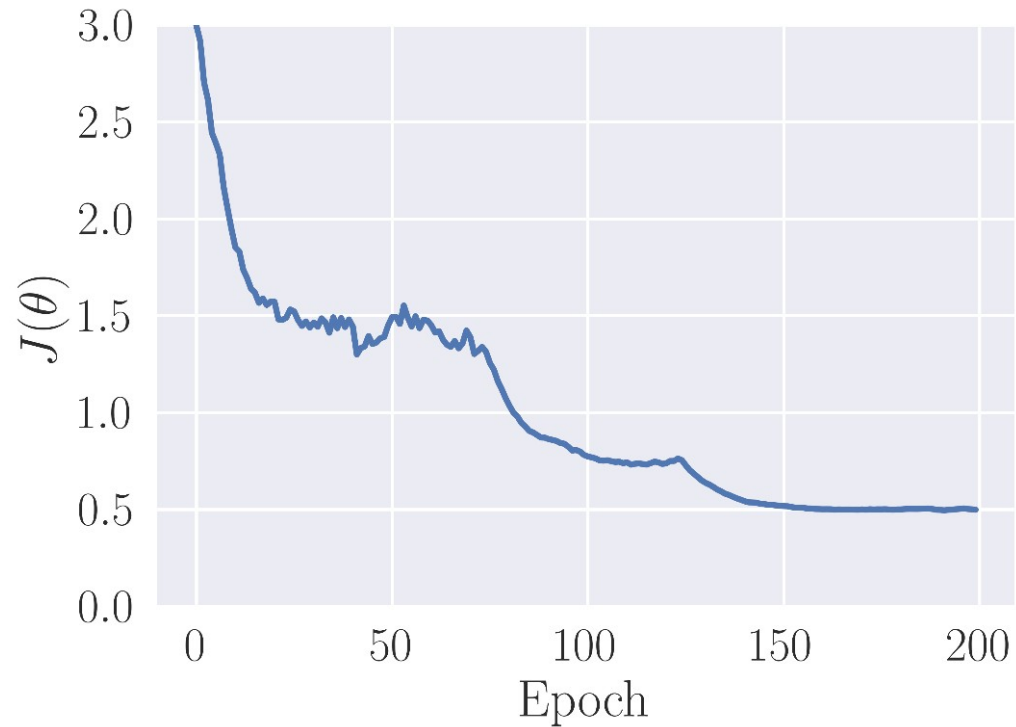
Convergence aux alentours de l'époque 125.



Époque	0	75		
Taux d'apprentissage	0.1	0.01		

Horaire d'apprentissage

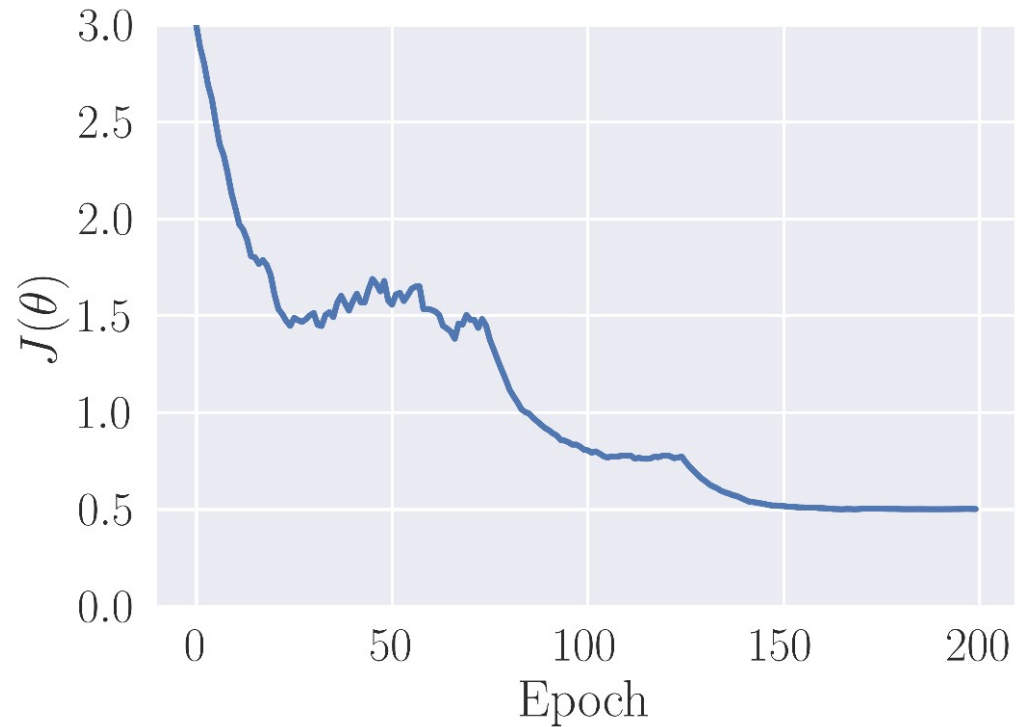
Convergence aux alentours de l'époque 175.



Époque	0	75	125	
Taux d'apprentissage	0.1	0.01	0.001	

Horaire d'apprentissage

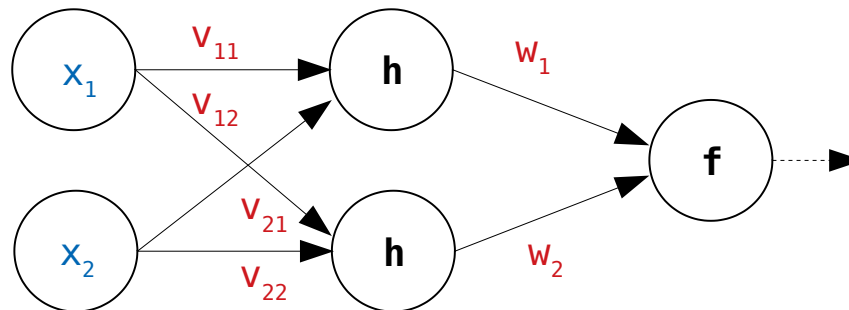
Aucun gain
après époque
175.



Époque	0	75	125	175
Taux d'apprentissage	0.1	0.01	0.001	0.0001

Horaire d'apprentissage

- Une mauvaise initialisation des poids du réseau est susceptible de:
 - Ralentir la vitesse de convergence
 - Converger vers un minimum local indésirable
- L'initialisation aléatoire des poids permet de briser les symétries

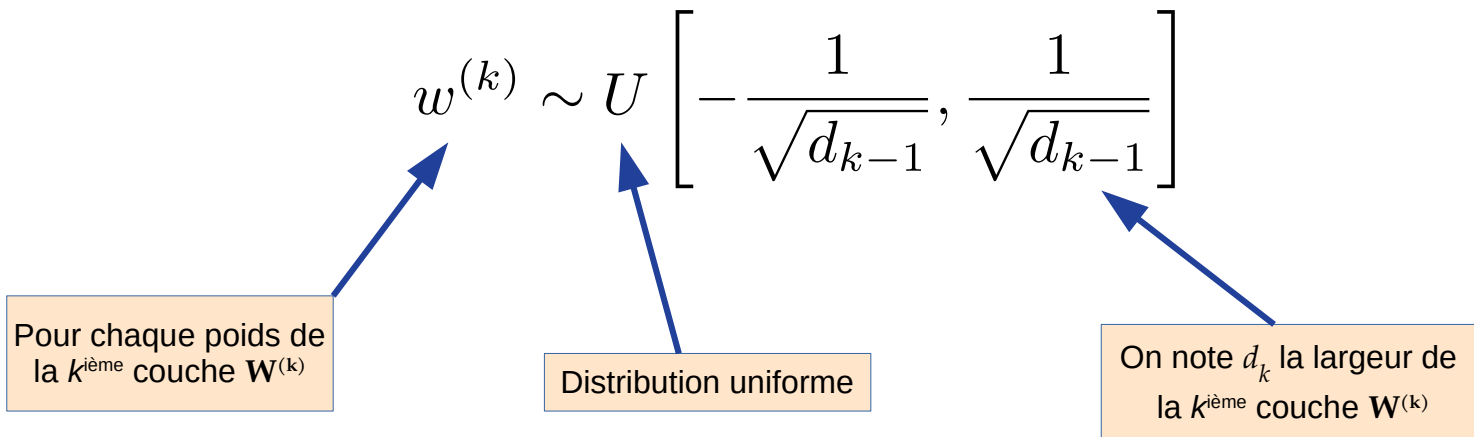


- **Question:** Qu'arrive-t-il si j'initialise tous les poids à la même valeur?

Initialisation des paramètres

Valeurs par défaut de Pytorch

<https://pytorch.org/docs/stable/nn.html#torch.nn.Linear>



$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{1,1}^{(k)} & \cdots & w_{1,d_{k-1}}^{(k)} \\ \vdots & \ddots & \vdots \\ w_{d_k,1}^{(k)} & \cdots & w_{d_k,d_{k-1}}^{(k)} \end{bmatrix} \in \mathbb{R}^{d_k \times d_{k-1}}.$$

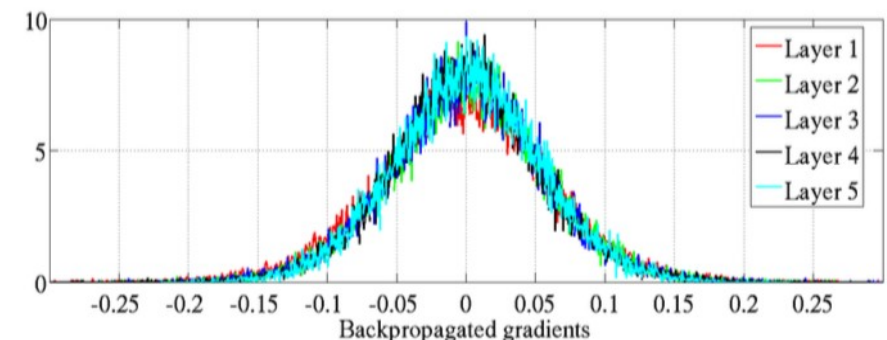
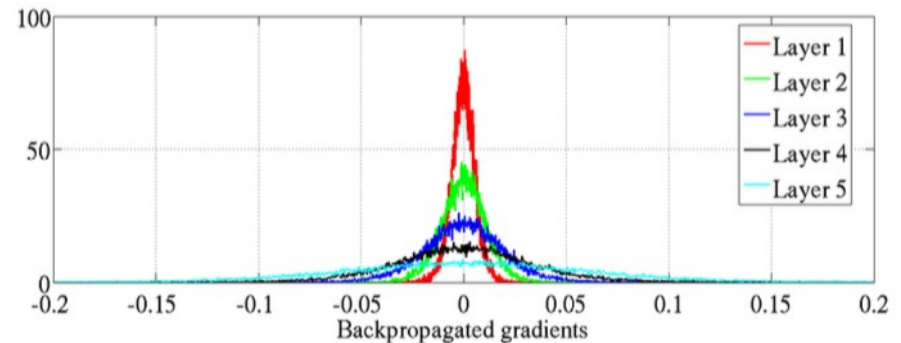
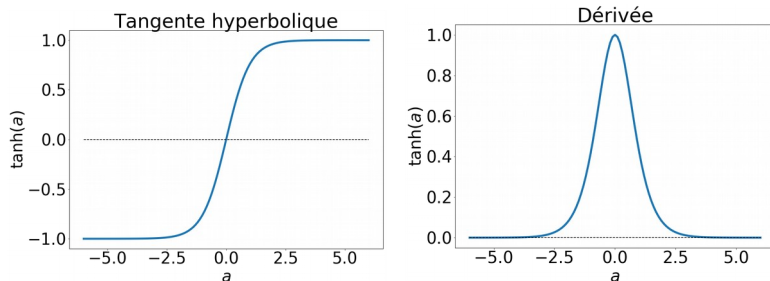
Initialisation des paramètres

Objectif: Conserver la variance des gradients similaire à l'entrée et à la sortie de chaque couche du réseau

$$w^{(k)} \sim U \left[-\frac{\sqrt{6}}{\sqrt{d_{k-1} + d_k + 1}}, \frac{\sqrt{6}}{\sqrt{d_{k-1} + d_k + 1}} \right]$$

Pytorch: https://pytorch.org/docs/stable/nn.init.html?highlight=kaiming#torch.nn.init.xavier_uniform_

Pour les fonction d'activation de type tangente hyperbolique



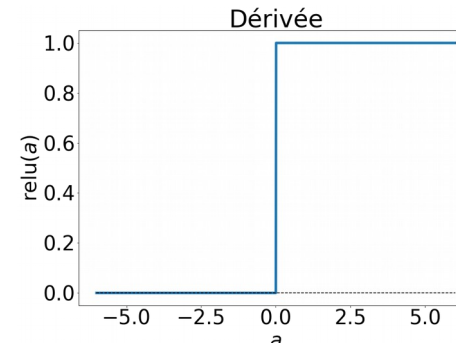
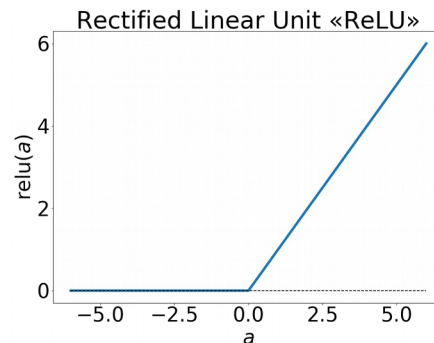
Initialisation des paramètres

Objectif: Conserver la variance des gradients similaire à l'entrée et à la sortie de chaque couche du réseau

$$w^{(k)} \sim U \left[-\frac{2}{\sqrt{d_{k-1}}}, \frac{2}{\sqrt{d_{k-1}}} \right]$$

Pytorch: https://pytorch.org/docs/stable/nn.init.html?highlight=kaiming#torch.nn.init.kaiming_uniform_

Pour les fonction d'activation de type ReLU



Terme de régularisation

Paramètre de « *weight decay* » à déterminer.

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n L(R_{\mathbf{w}}(\mathbf{x}_i), y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\text{où } \mathbf{w} = \text{vec}(\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \dots)$$

Somme des carrés de tous les poids du réseau,
excluant les paramètres de biais

Interprétation(s) :

- Empêche de «surapprendre» en se concentrant exclusivement sur la minimisation de la perte sur l'ensemble d'apprentissage.
- Favorise les prédicteurs «simples»
- Favorise les prédicteurs «stables»

Possède de belles propriétés pour les problèmes d'apprentissage convexes, ce qui n'est pas le cas ici !

Terme de régularisation

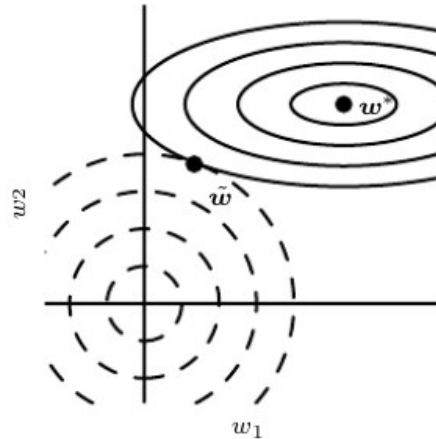


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal \mathbf{w} . The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\mathbf{w}}$, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from \mathbf{w}^* . Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to zero. In the second dimension, the objective function is very sensitive to movements away from \mathbf{w}^* . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

Source : Goodfellow, Bengio, Courville

<https://www.deeplearningbook.org/contents/regularization.html>

Terme de régularisation

Paramètre de « *weight decay* » à déterminer.

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n L(R_{\mathbf{w}}(\mathbf{x}_i), y_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Pourquoi le terme « *weight decay* » pour désigner le paramètre de régularisation λ ?

Gradient de la fonction de perte

Gradient du terme de régularisation

$$\nabla F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \nabla L(R_{\mathbf{w}}(\mathbf{x}_i), y_i) + \lambda \mathbf{w}$$

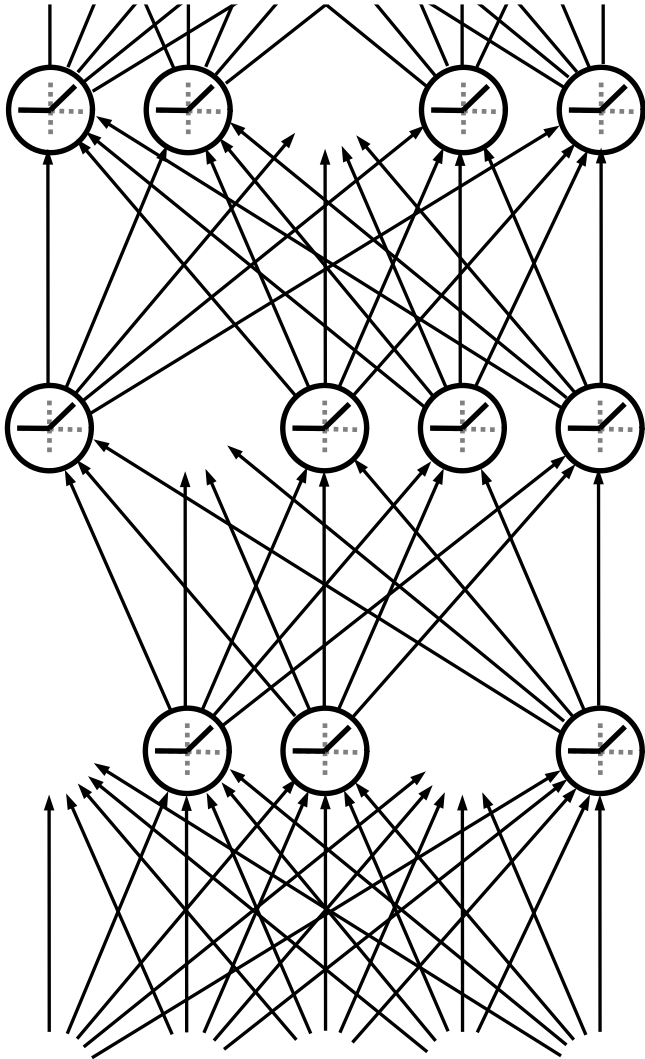
Voir : <https://pytorch.org/docs/stable/optim.html?highlight=sgd#torch.optim.SGD>

Dropout

- Retirer aléatoirement et temporairement des neurones du réseau à chaque itération de la descente de gradient
- Paramètre : Probabilité d'inclusion p_{inc}
- Peut se faire aisément en multipliant la sortie des neurones retirées par 0

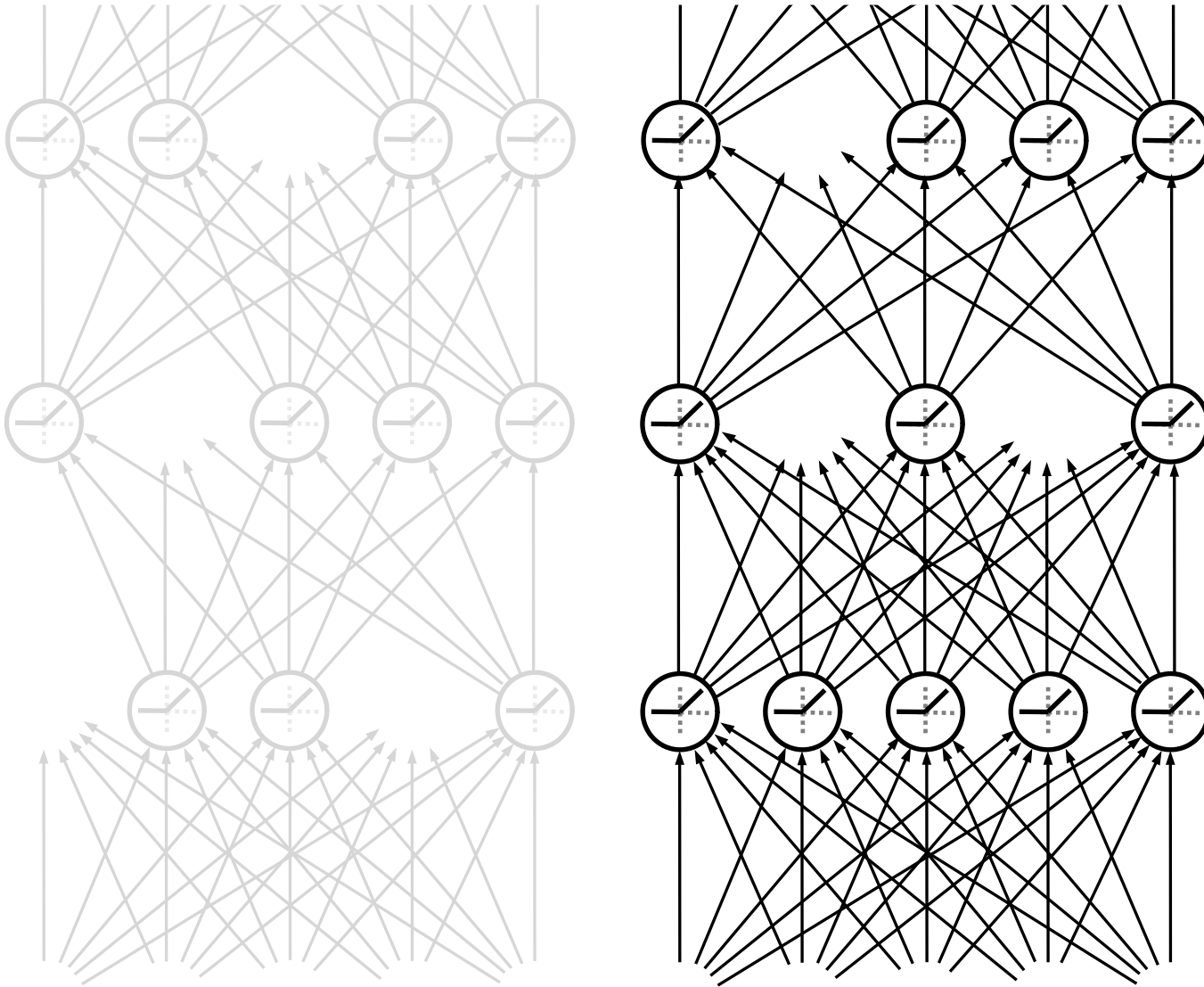
Dropout

Peut être vu comme une agrégation de plusieurs modèles



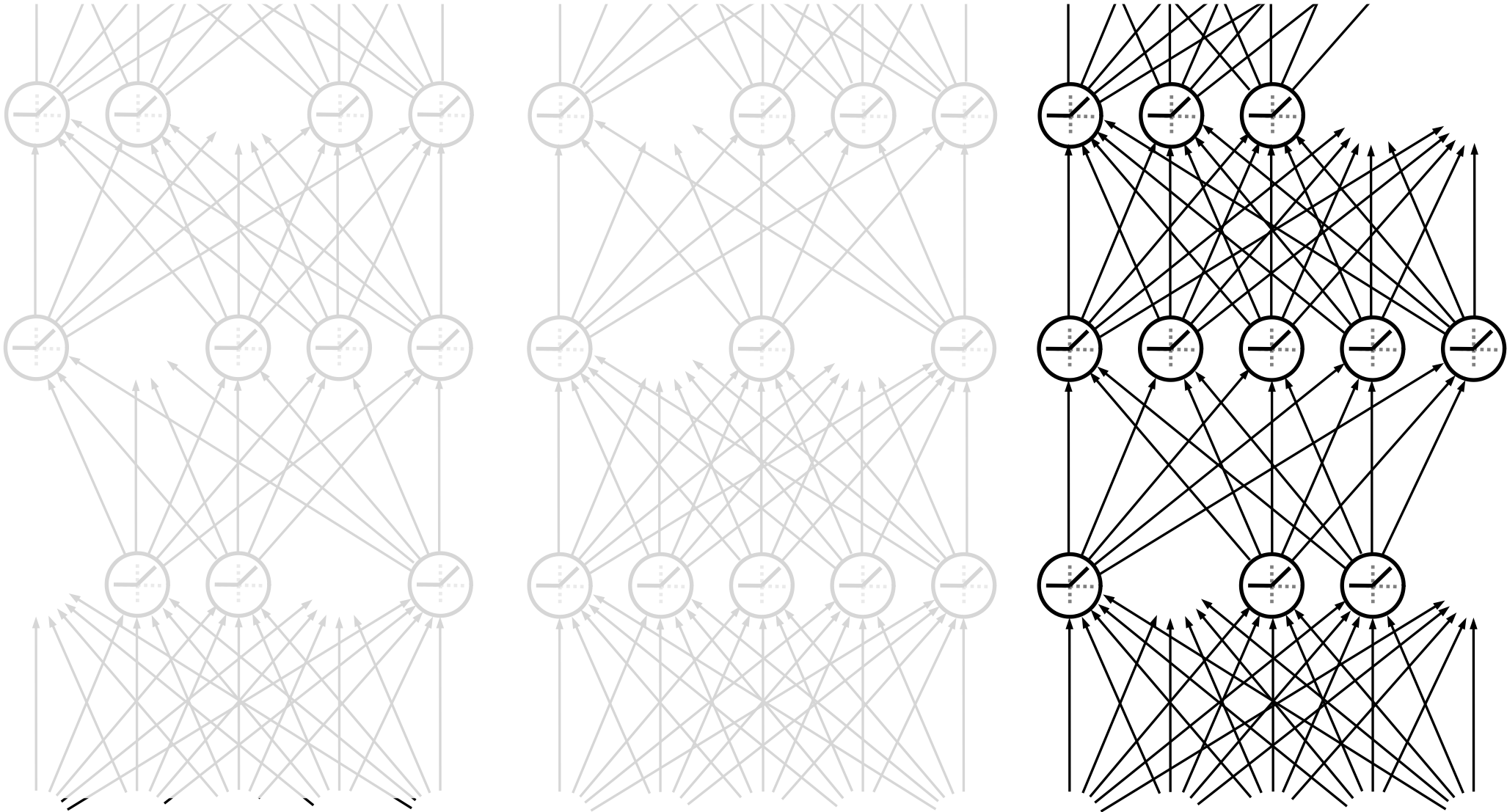
Dropout

Peut être vu comme une agrégation de plusieurs modèles



Dropout


Peut être vu comme une agrégation de plusieurs modèles



Dropout

- Choisir un taux d'inclusion :
 - par couche ($p_{inc}=0.8$ en entrée, $p_{inc}=0.5$ ailleurs)
 - pour le réseau entier (sauf sortie)
- Limite la coadaptation des neurones : un *neurone* doit être bon en plusieurs contextes
- On doit parfois compenser la perte de capacité en augmentation la taille du réseau.

Dropout

- À l'inférence, deux stratégies possibles :
 - A** On sélectionne 10-20 masques de dropout et l'on fait la moyenne des prédictions
 -  **B** On conserve tous les neurones, mais on pondère les connections sortantes par p_{inc}

$$\mathbf{E}a = \sum_{i=1}^D p_{inc} w_i \phi_i$$

Note : PyTorch fait **B**, mais en pondérant par $1/(1-p_{drop})$ durant l'apprentissage.

Normalisation par *batch* (ou «*batchnorm*»)

Lors de l'apprentissage par «minibatch», normaliser et centrer la distribution des sorties de chaque neurone individuellement

Considérons un neurone. Lors de la phase de propagation avant de l'apprentissage, ce neurone est traversé par une minibatch de m éléments au temps t .

1. Notons $a_{t;1}, \dots, a_{t;m}$ l'ensemble des valeurs sorties du neurone.

Normalisation par *batch* (ou «*batchnorm*»)

Lors de l'apprentissage par «minibatch», normaliser et centrer la distribution des sorties de chaque neurone individuellement

Considérons un neurone. Lors de la phase de propagation avant de l'apprentissage, ce neurone est traversé par une minibatch de m éléments au temps t .

1. Notons $a_{t;1}, \dots, a_{t;m}$ l'ensemble des valeurs sorties du neurone.
2. On calcule

$$\mu_{a_t} = \frac{1}{m} \sum_{i=1}^m a_{t;i}$$
$$\sigma_{a_t} = \sqrt{\epsilon + \frac{1}{m} (\mu_{a_t} - a_{t;i})^2} \quad (\epsilon \approx 10^{-8})$$

3. Puis, on transforme chaque a_i :

$$\tilde{a}_k = \gamma_a \frac{a_{t;i} - \mu_{a_t}}{\sigma_{a_t}} + \beta_a,$$

où γ_a et β_a sont des paramètres associés au neurone au même titre que les poids, initialisés au hasard et appris par rétropropagation du gradient.

⇒ Lors de la prédiction, γ_a et β_a sont remplacés par des valeurs estimées sur tout l'ensemble d'apprentissage.

Augmentation des données

- Créer des *nouvelles données* à partir de l'ensemble d'apprentissage
- Fonctionne particulièrement bien pour classification visuelle, car on y cherche l'invariance à certaines transformations :
 - Rotation
 - Translation
 - Réflexion horizontale ou verticale
 - Échelle
 - Intensité lumineuse, etc...
- Nécessité de comprendre le processus (*physique*) de création des données

Augmentation des données

Horizontal flip



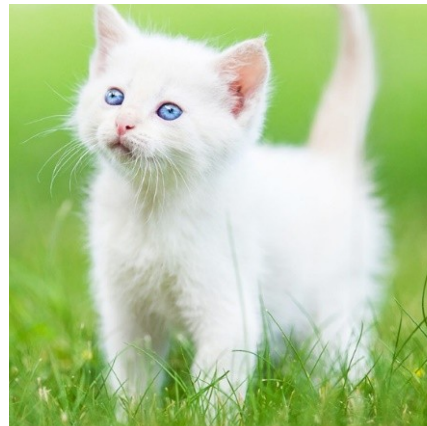
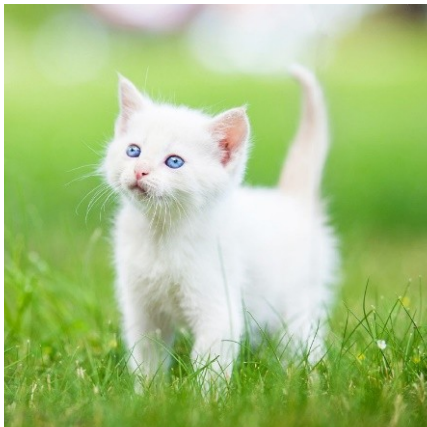
Vertical flip



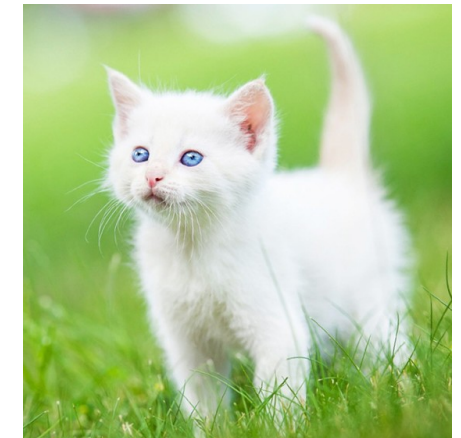
Random crop



Random scale



Random rotation



Combinaison de ces opérations
Attention à ne pas changer un 6 en 9...

Exemples de «torchvision»

- RandomCrop
- RandomHorizontalFlip
- RandomVerticalFlip
- RandomResizedCrop
- RandomSizedCrop
- ColorJitter

transforms.ColorJitter



Brightness

Trial #0



Trial #1



Trial #2



Trial #3

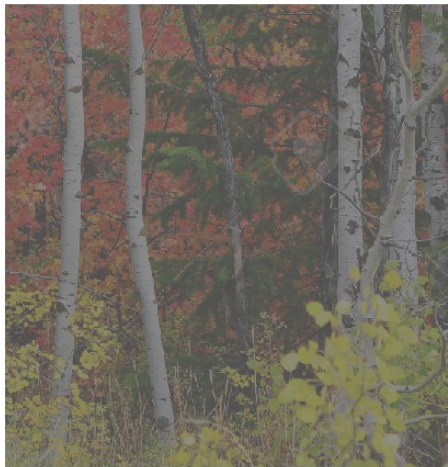


transforms.ColorJitter



Contrast

Trial #0



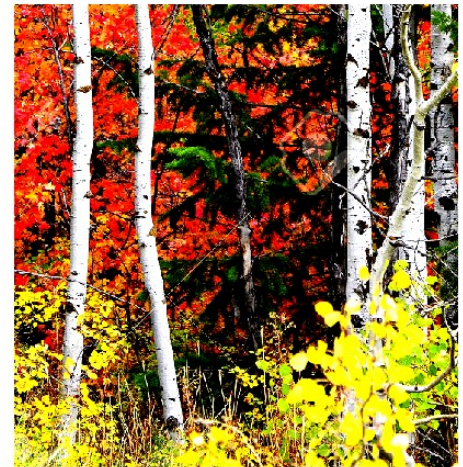
Trial #1



Trial #2



Trial #3



`transforms.ColorJitter`



Saturation

Trial #0



Trial #1



Trial #2



Trial #3

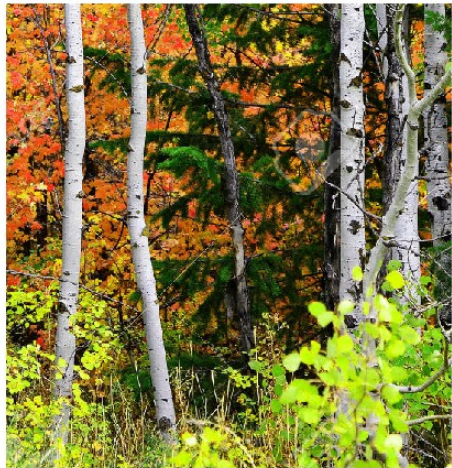


transforms.ColorJitter



Hue

Essai #0



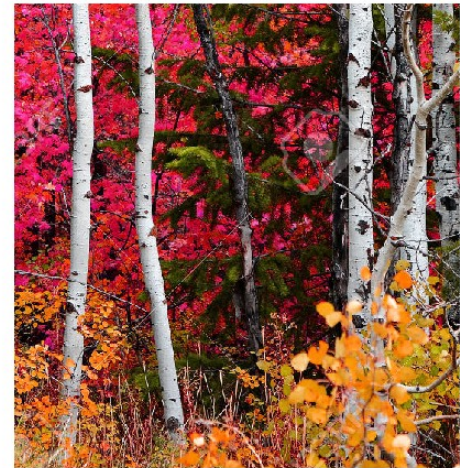
Essai #1



Essai #2



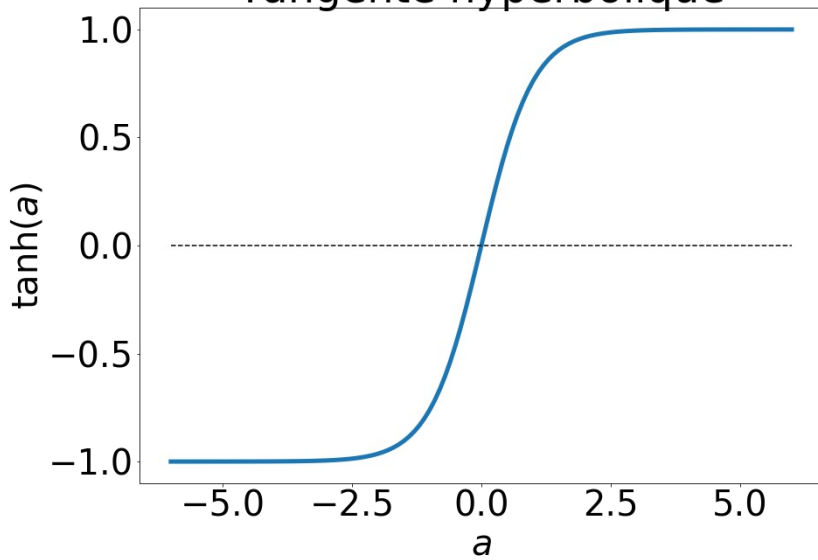
Essai #3



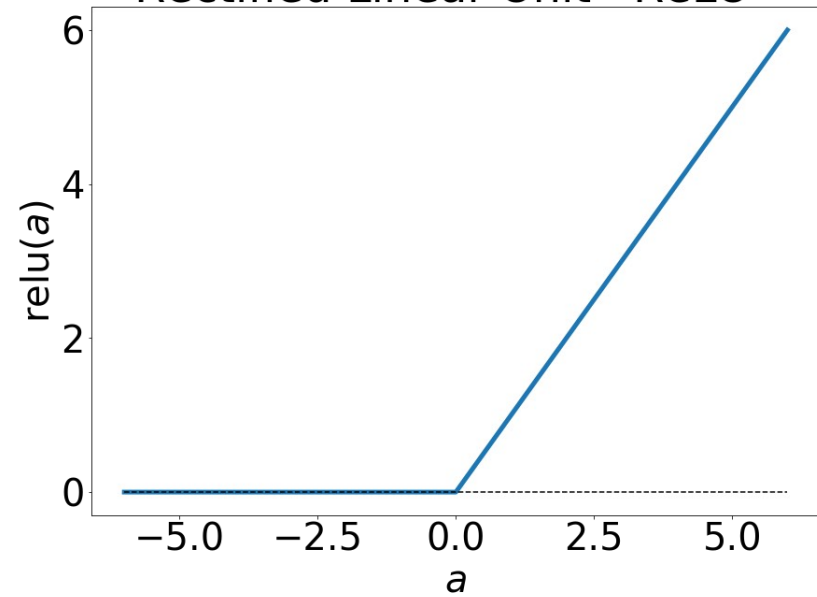
Remarques sur les fonctions d'activations

Disparition du gradient

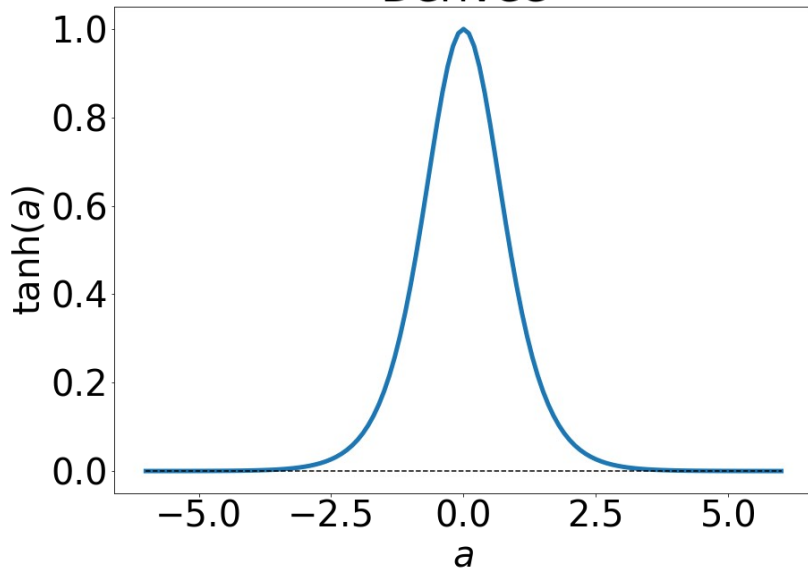
Tangente hyperbolique



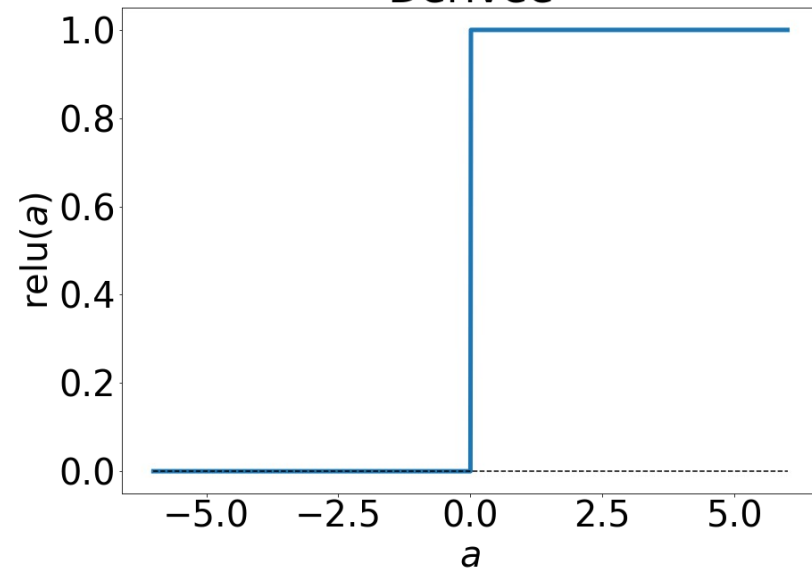
Rectified Linear Unit «ReLU»



Dérivée

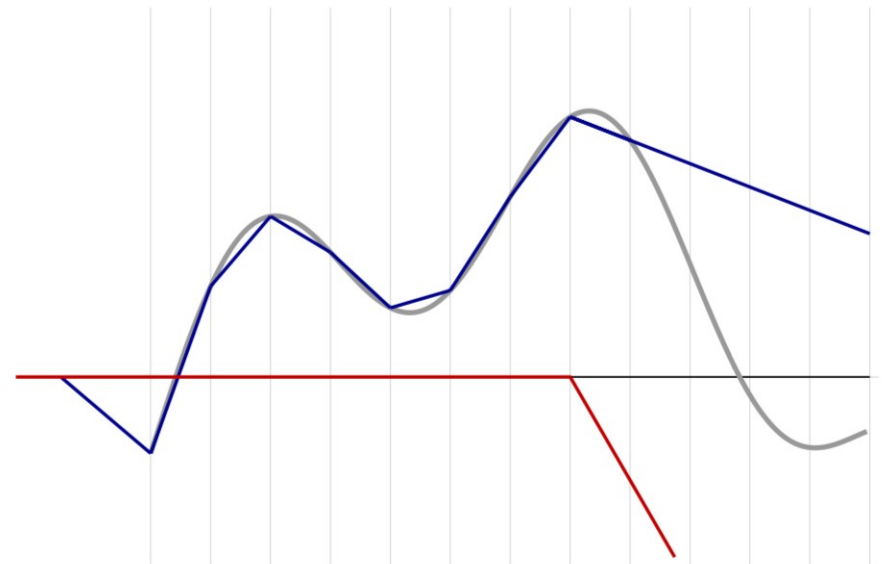
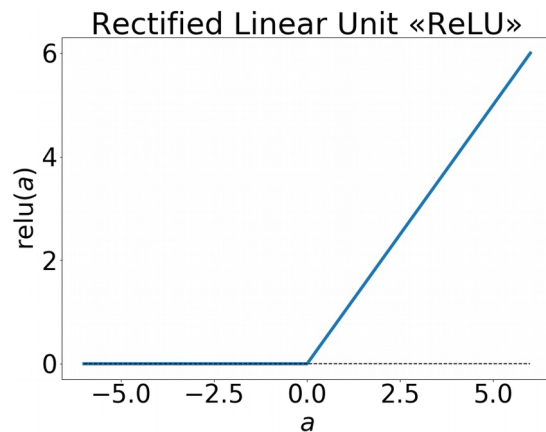


Dérivée



Approximation universelle

Toute fonction continue peut être exprimée comme une combinaison linéaire de fonctions ReLU



Exemple tiré de :
<https://fleuret.org/ee559/> (section 3.4)