INTRODUCTION AUX RÉSEAUX DE NEURONES

Le processus d'apprentissage en pratique

Pascal Germain*, 2018

* 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 0 à T:
  - Choisir aléatoirement $i \in \{1, \ldots, d\}$
  - $\mathbf{g} \leftarrow \nabla F_i(\mathbf{z})$
  - $\mathbf{v} \leftarrow \alpha\, \mathbf{v} - \dfrac{\eta}{\sqrt{t}} \mathbf{g}$
  - $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$
- Retourner $\mathbf{z}$

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 mini-batch.
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 0 à T:
  - Diviser aléatoirement S en $\lfloor n/m \rfloor$ sous-ensembles disjoints $\hat{S}_1, \ldots, \hat{S}_{\lfloor n/m \rfloor}$
  - Pour $\hat{S}_i = \hat{S}_1, \ldots, \hat{S}_{\lfloor n/m \rfloor}$:
    - $\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$
    - $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\eta}{\sqrt{t}} \mathbf{g}_t$
    - $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$
- Retourner $\mathbf{z}$

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

Ce sont les «minibatch»

Moyenne du grandient sur les *m* exemples de la «minibatch»

Apprentissage par «minibatch»

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

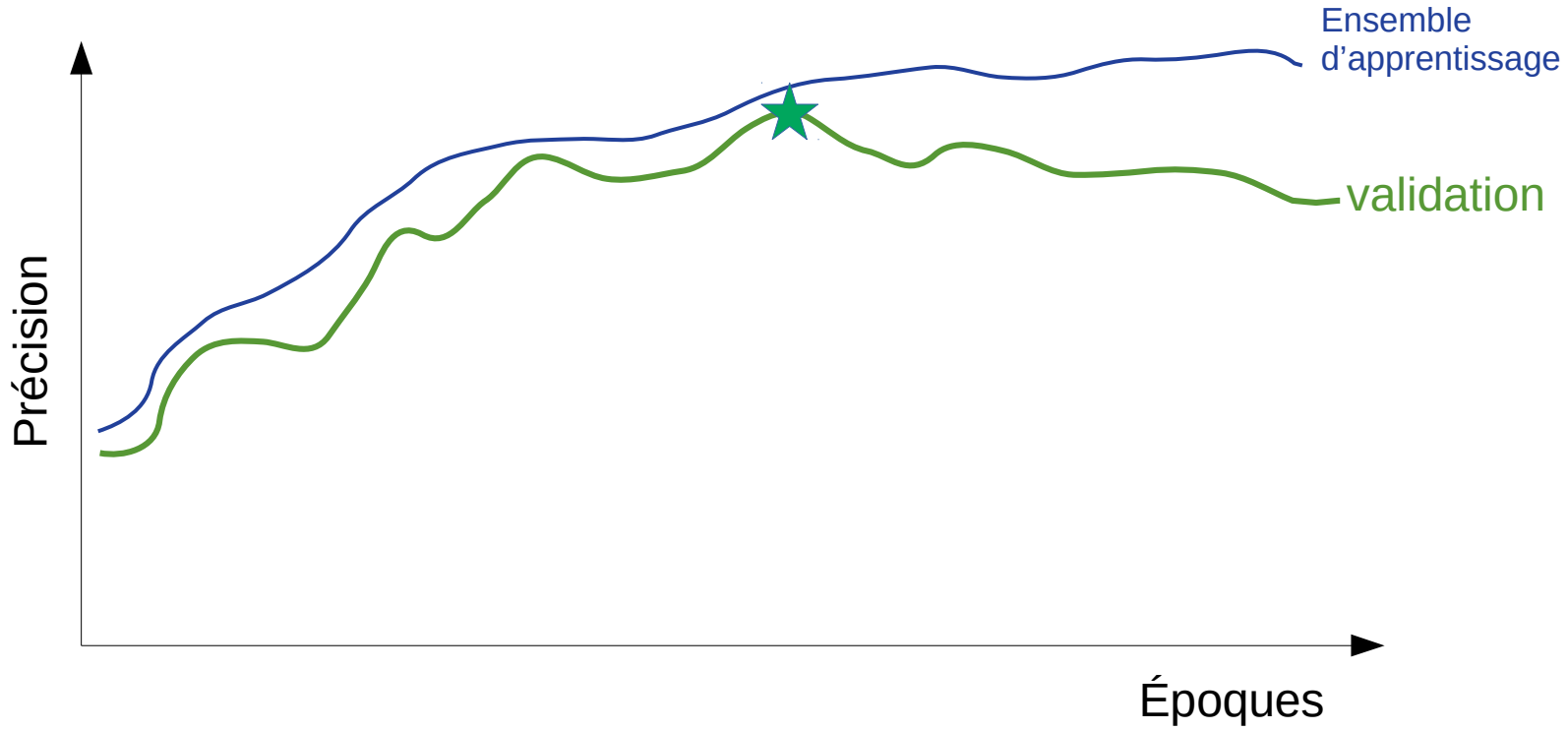
$$\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$$

- 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 opération matricielles sur CPU / GPU
- Mais la taille de la minibatch influence la solution obtenue...

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)

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
- $\mathbf{v} \leftarrow 0$
- $t \leftarrow 0$, $T \leftarrow p$, $\mathbf{z}^* \leftarrow \mathbf{z}$
- Tant que $t < T$:
  - Diviser aléatoirement S en $\lfloor n/m \rfloor$ sous-ensembles disjoints $\hat{S}_1, \ldots, \hat{S}_{\lfloor n/m \rfloor}$
  - Pour $\hat{S}_i = \hat{S}_1, \ldots, \hat{S}_{\lfloor n/m \rfloor}$:
    - $\mathbf{g} \leftarrow \frac{1}{m} \sum_{(\mathbf{x}_k, y_k) \in \hat{S}_i} \nabla F_k(\mathbf{z})$
    - $\mathbf{v} \leftarrow \alpha\, \mathbf{v} - \frac{\eta}{\sqrt{t}}\, \mathbf{g}_t$
    - $\mathbf{z} \leftarrow \mathbf{z} + \mathbf{v}$
  - Si Précision$(\mathbf{z}, S_V) >$ Précision$(\mathbf{z}^*, S_V)$:
    - $\mathbf{z}^* \leftarrow \mathbf{z}$
    - $T \leftarrow t + p$
  - $t \leftarrow t + 1$
- Retourner $\mathbf{z}^*$

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

z* représente le meilleur modèle rencontré

Tant que ma patience n'est pas épuisée !

Si la précision du modèle représenté par les paramètres ***z*** sur l'ensemble de Validation S_v est plus élevée que le meilleurs modèle rencontré jusqu'à maintenant

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

Retourne le meilleur modèle rencontré sur l'ensemble de validation

Horaire d'apprentissage

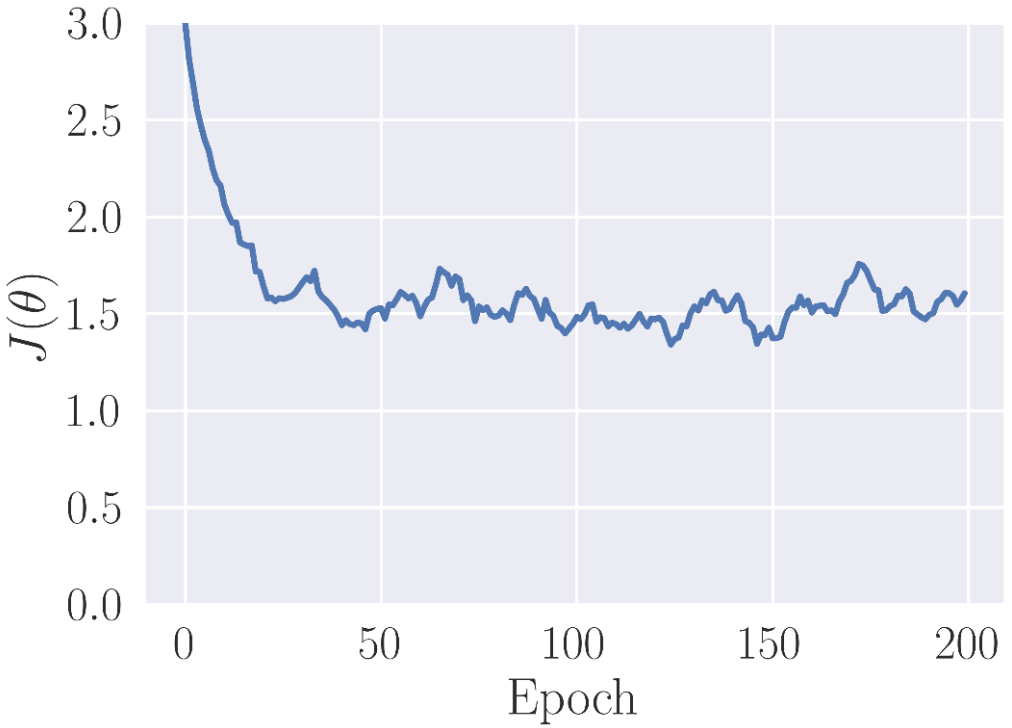
- On désire généralement un taux d'apprentissage élevé en début d'apprentissage et un taux d'apprentissage bas vers la fin
- Toutefois, le taux de décroissance idéal dépend du problème
- En pratique, il est commun d'ajuster le taux d'apprentissage «à la main»

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

Horaire d'apprentissage

Convergence aux alentours de l'époque 75.

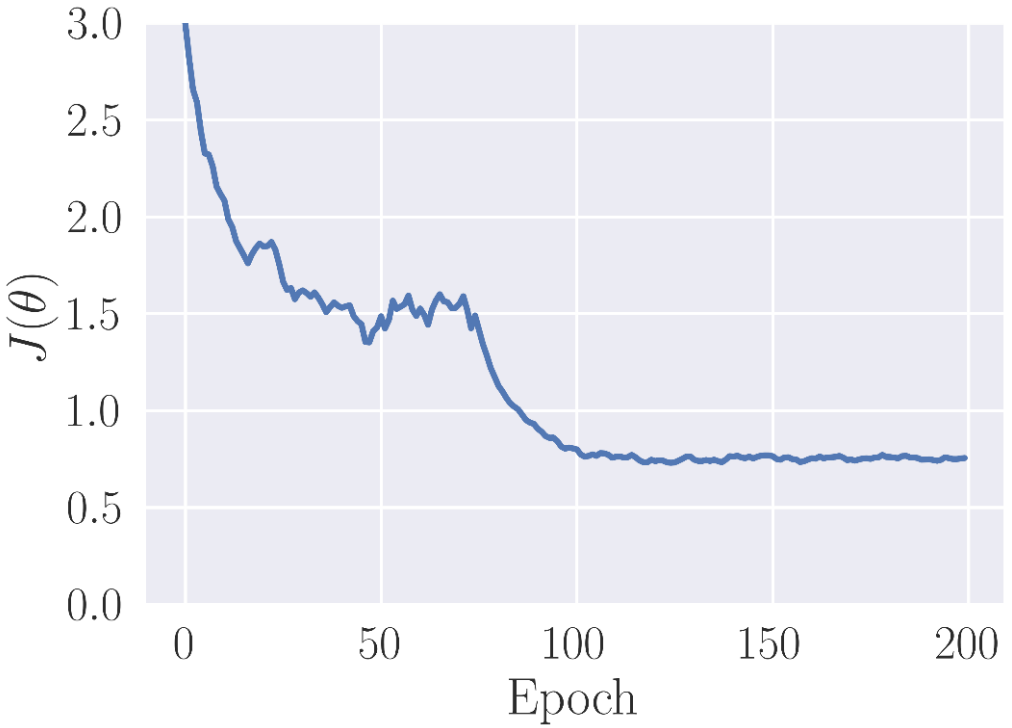


Époque	0			
Taux d'apprentissage	0.1			

Crédit: Ludovic Trottier

Horaire d'apprentissage

Convergence aux alentours de l'époque 125.

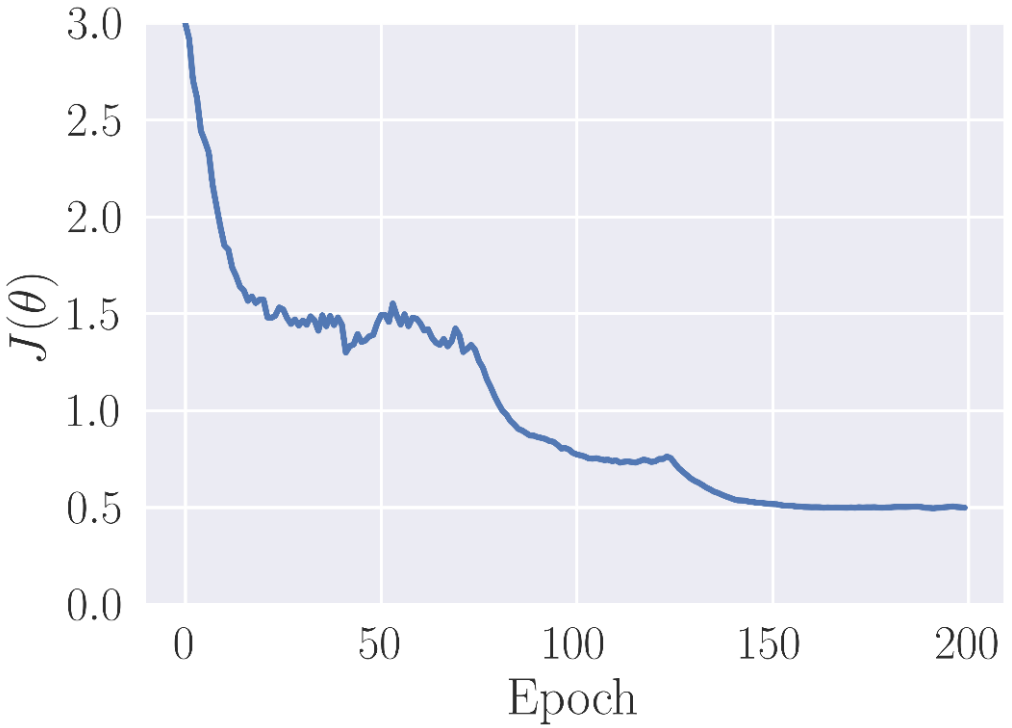


Époque	0	75		
Taux d'apprentissage	0.1	0.01		

Crédit: Ludovic Trottier

Horaire d'apprentissage

Convergence aux alentours de l'époque 175.

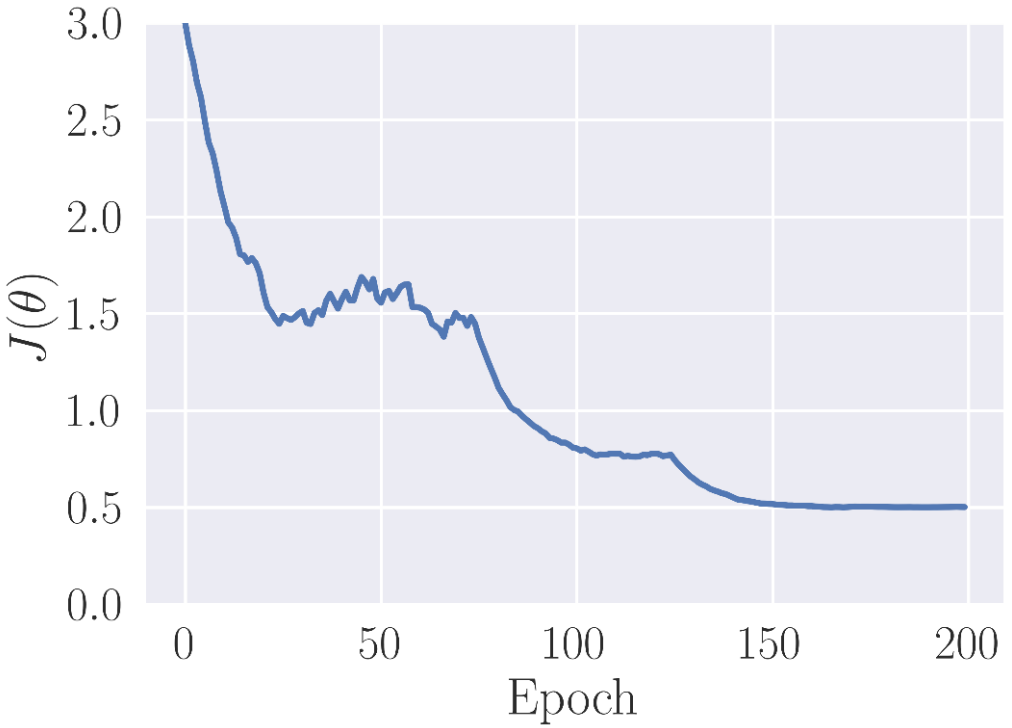


Époque	0	75	125	
Taux d'apprentissage	0.1	0.01	0.001	

Crédit: Ludovic Trottier

Horaire d'apprentissage

Aucun gain après époque 175.

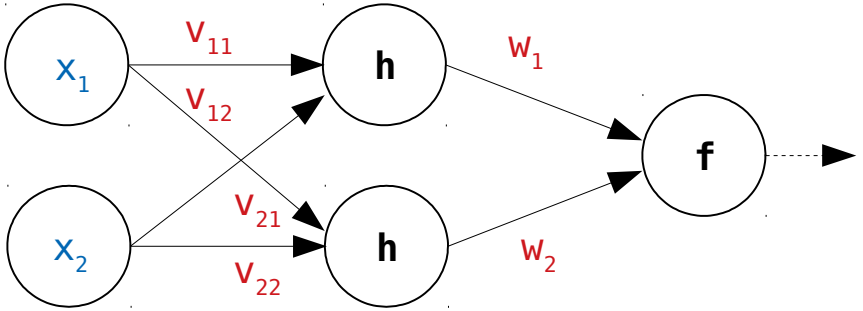


Époque	0	75	125	175
Taux d'apprentissage	0.1	0.01	0.001	0.0001

Crédit: Ludovic Trottier

De l'importance de l'initialisation

- 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 par défaut de pytorch

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

Pour chaque poids de la $k^{\text{ième}}$ couche $\mathbf{W}^{(k)}$

On note d_k la largeur de la $k^{\text{ième}}$ couche $\mathbf{W}^{(k)}$

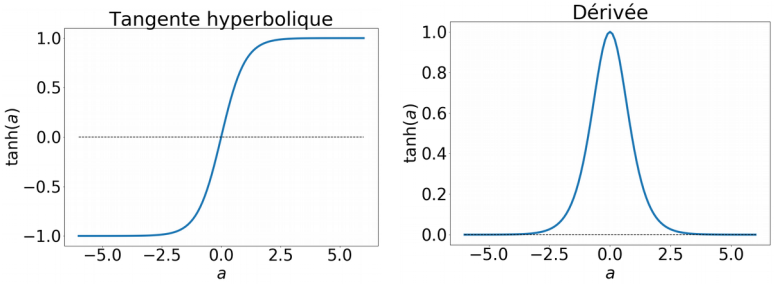
$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{1,1}^{(k)} & \cdots & w_{1,d_{k-1}}^{(k)} \\ \vdots & \ddots & \vdots \\ w_{d_k,1}^{(k)} & \cdots & w_{d_k,d_{k-1}}^{(k)} \end{bmatrix} \in \mathbb{R}^{d_k \times d_{k-1}}.$$

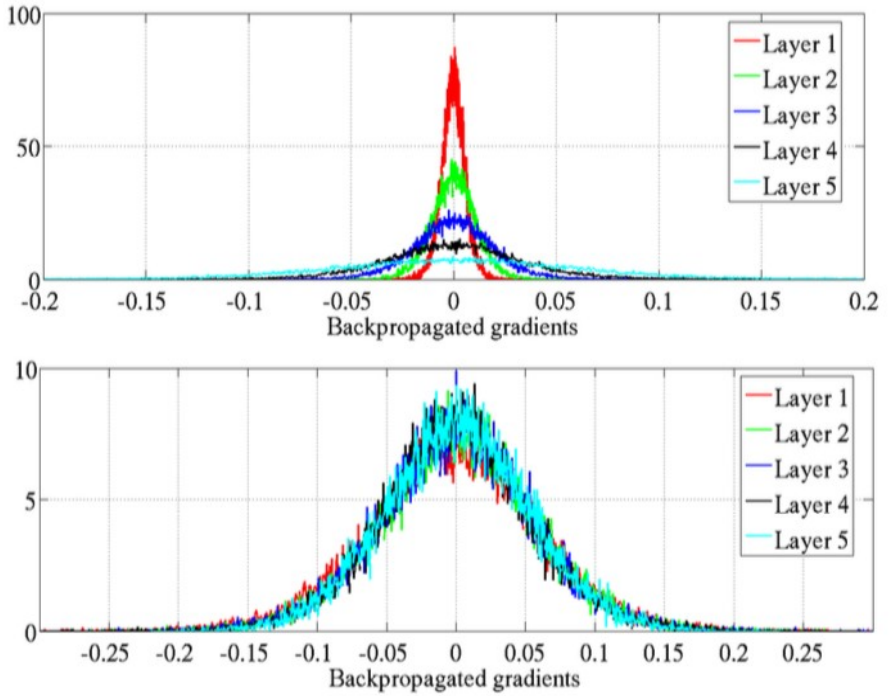
Initialisation «Xavier»

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

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

Pour les fonction d'activation de type tangente hyperbolique





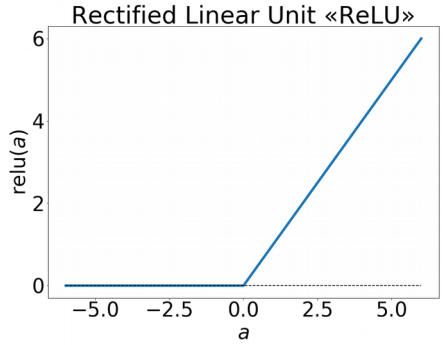
Xavier Glorot et Yoshua Bengio (2010): Understanding the difficulty of training deep feedforward neural networks, AISTATS

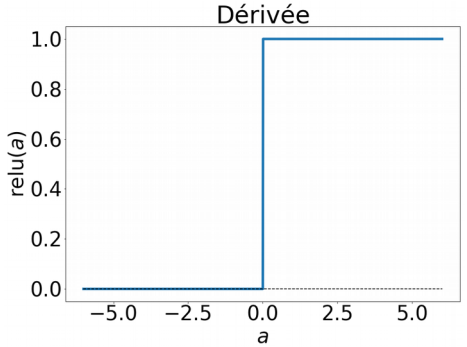
Initialisation «Kaiming»

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

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

Pour les fonction d'activation de type ReLU





He Kaiming et al. (2015): Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, ICCV

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 de l'apprentissage, ce neurone est traversé par une minibatch de m éléments au temps t.

1. Notons $a_{t;1}, \ldots, a_{t;m}$ l'ensemble des valeurs sortie 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 de l'apprentissage, ce neurone est traversé par une minibatch de m éléments au temps t.

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

$$\mu_{a_t} = \frac{1}{m} \sum_{i=1}^{m} a_{t;i}$$

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

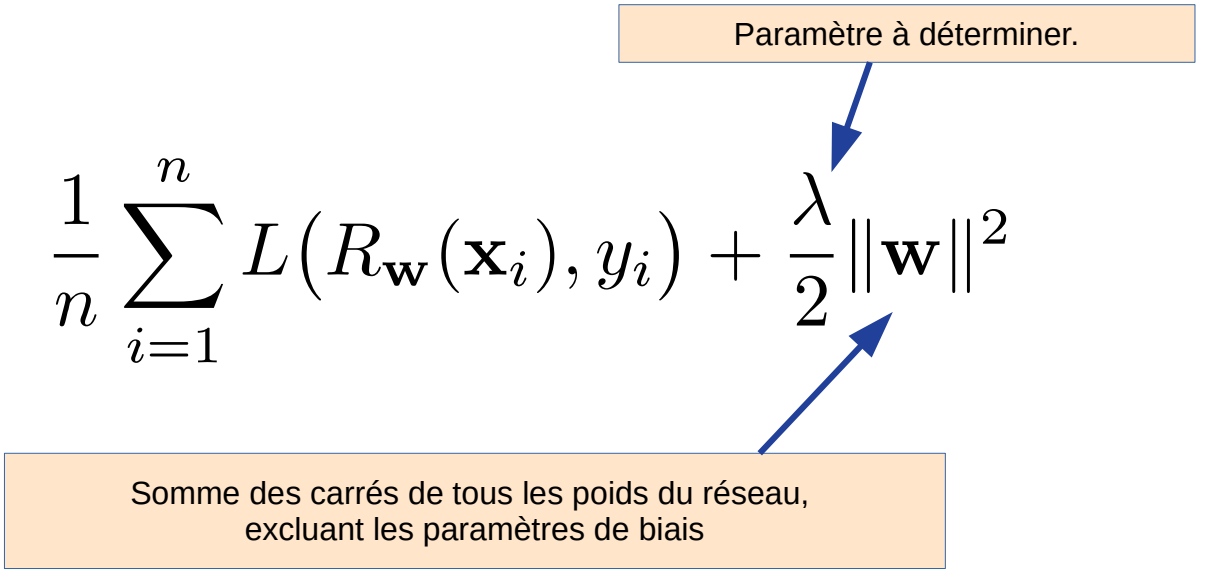
3. Puis, on transforme chaque a_i:

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

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.

$\Rightarrow$ Lors de la prédiction, γ_a et $\beta)a$ sont remplacés par des valeurs estimés sur tout l'ensemble d'apprentissage.

Terme de régularisation



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

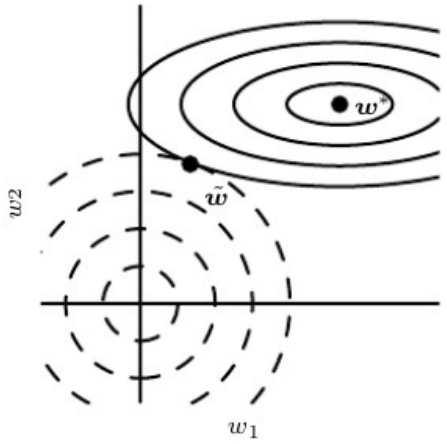


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\boldsymbol{w}}$, 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 $\boldsymbol{w}^*$. 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 $\boldsymbol{w}^*$. The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

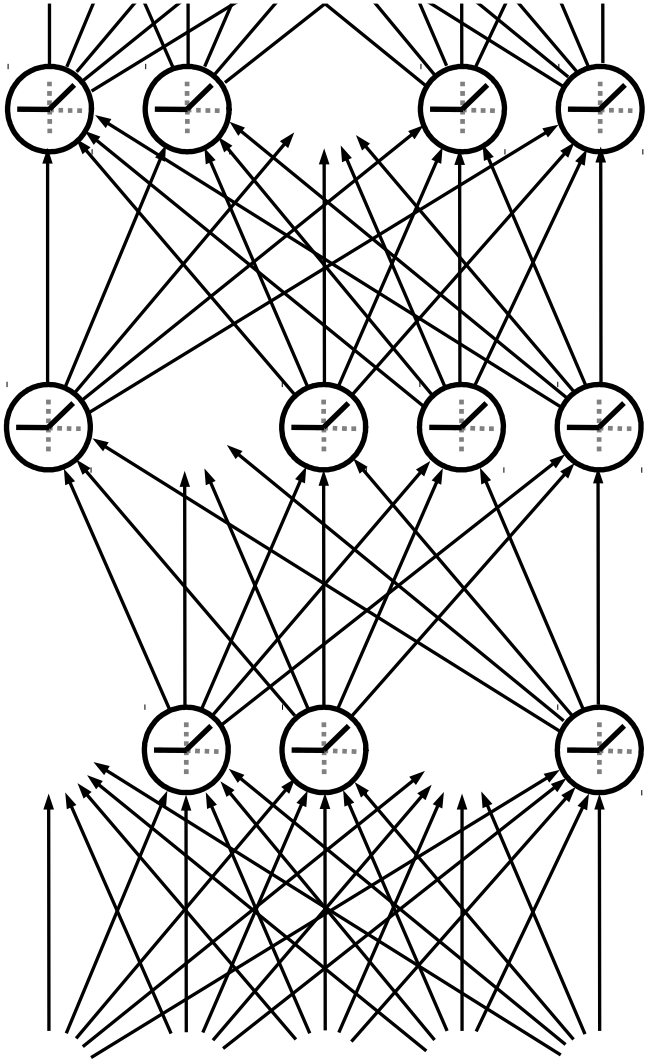
Source : Goodfellow, Bengio Courville
https://www.deeplearningbook.org/contents/regularization.html

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

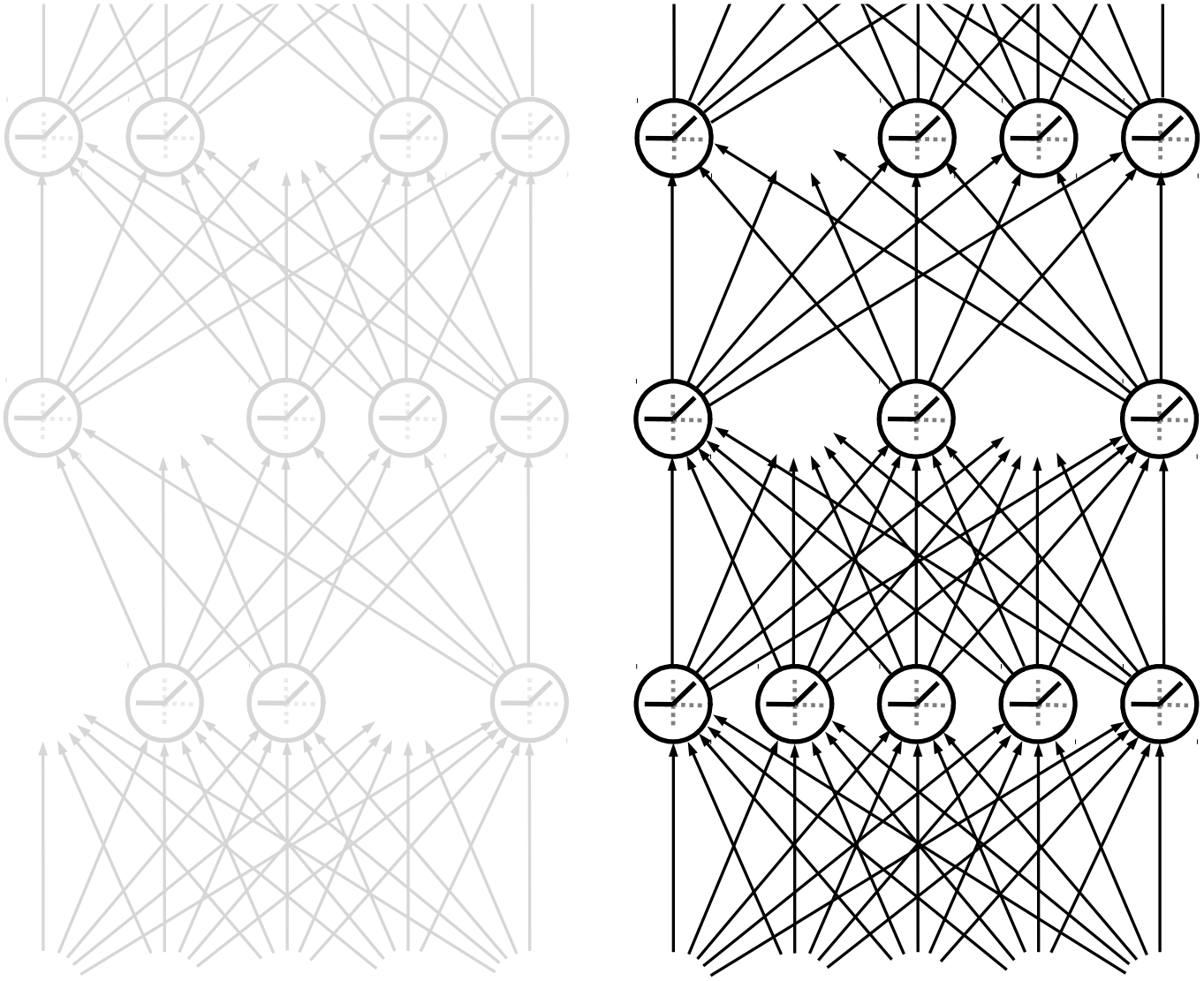
Dropout

Peut être vu comme une agrégation de plusieurs modèles



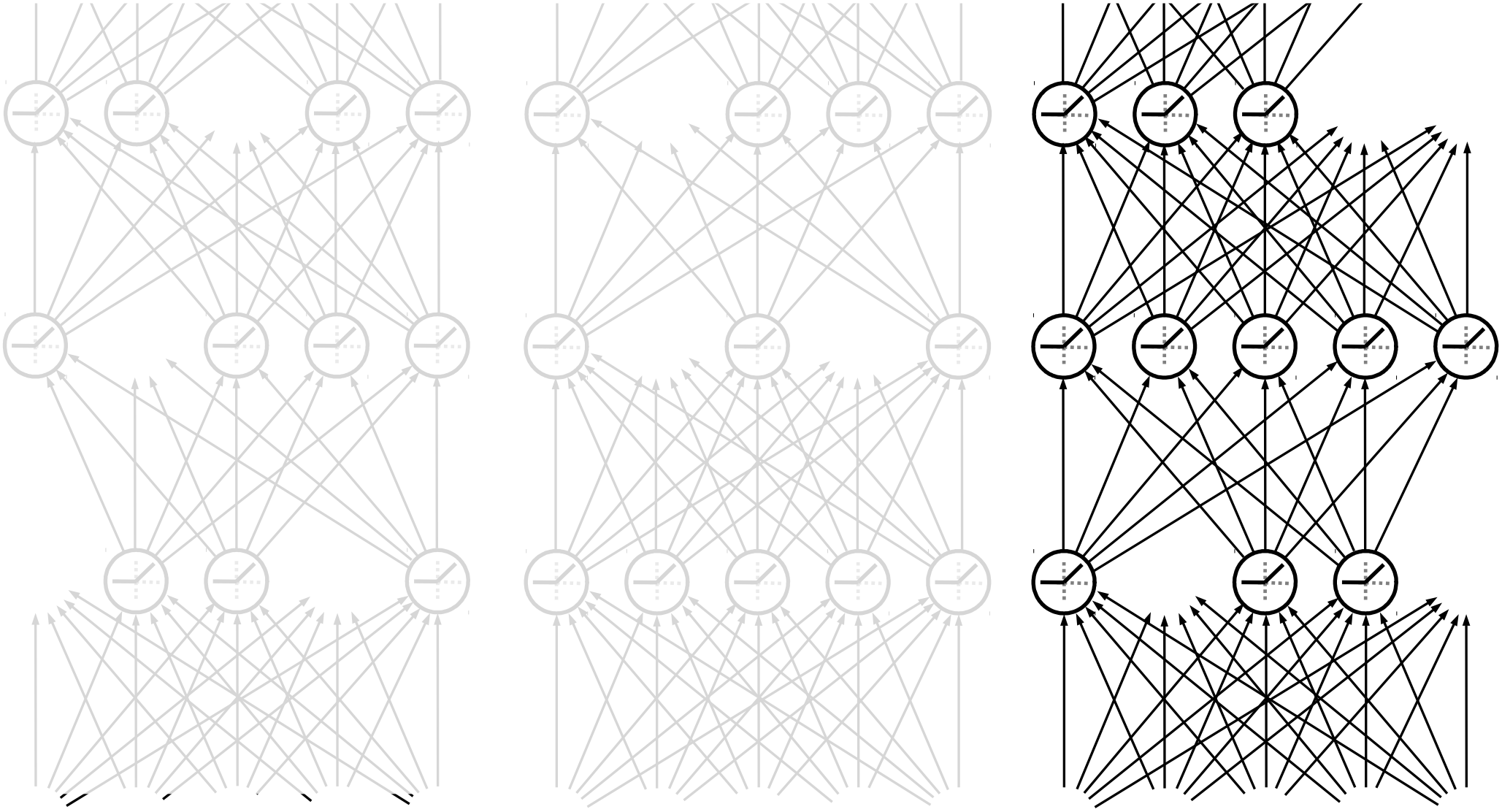
Dropout

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

  +commun

  B On conserve tous les neurones, mais on pondère les connections sortantes par p_{inc}

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

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

Exemples dans `torchvision`

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

https://github.com/pytorch/vision/blob/master/torchvision/transforms.py

Augmentation des données

Horizontal flip

Vertical flip

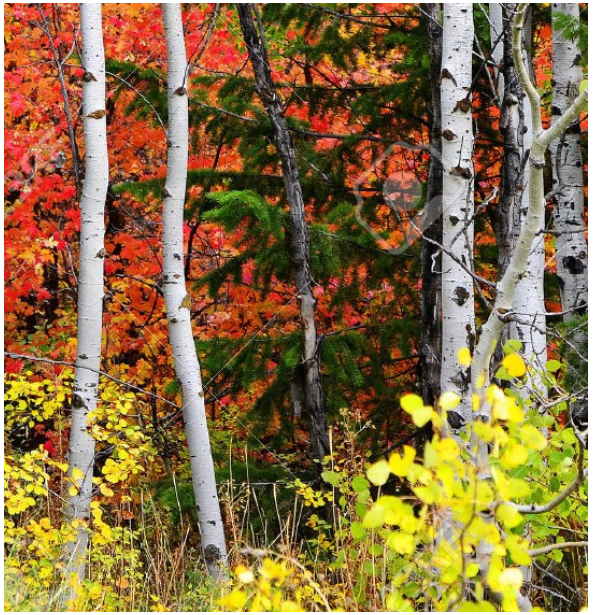
Random crop

Random scale

Random rotation

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

transforms.ColorJitter

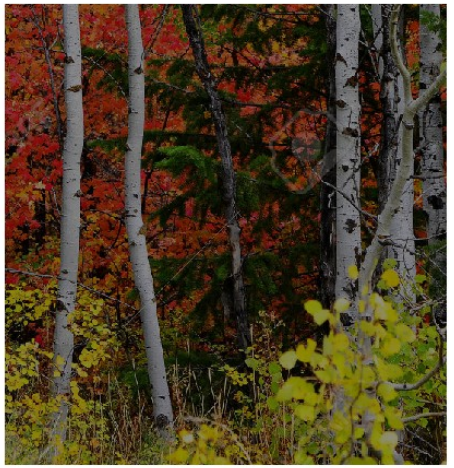


Brightness

Trial #0

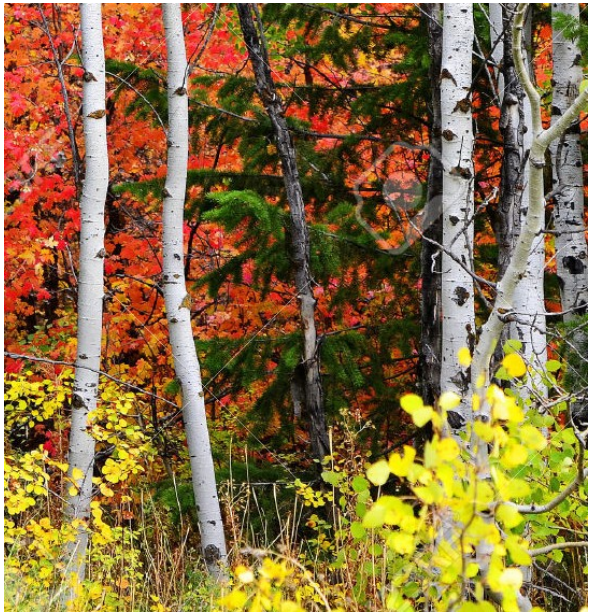
Trial #1

Trial #2



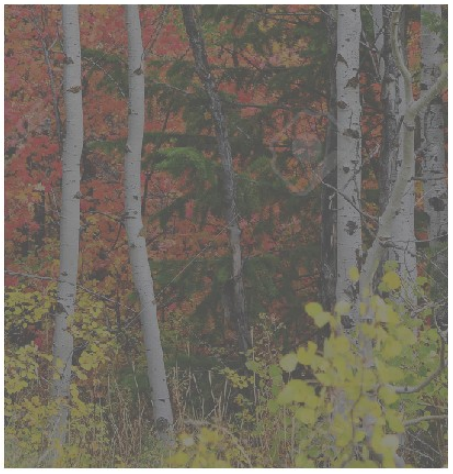
Trial #3

transforms.ColorJitter



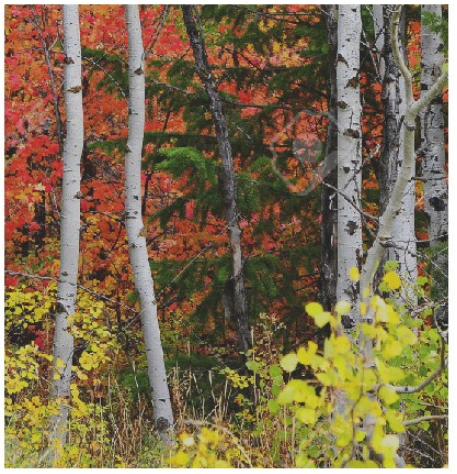
Contrast

Trial #0



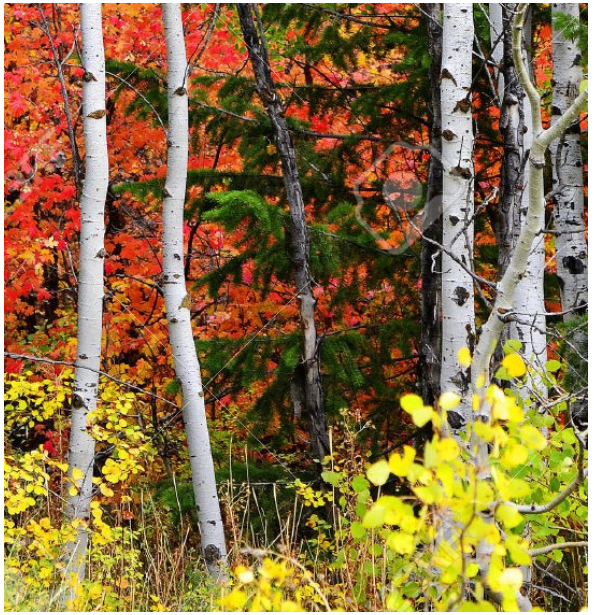
Trial #1

Trial #2



Trial #3

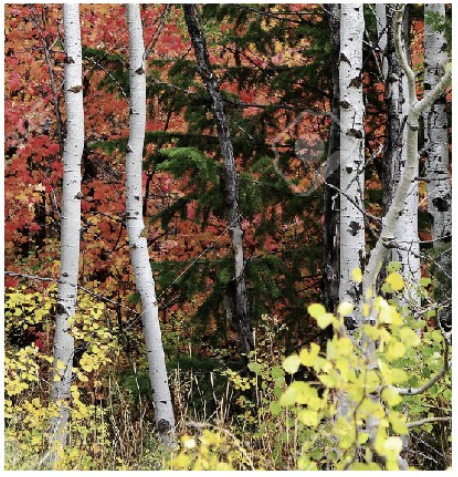
transforms.ColorJitter



Saturation

Trial #0

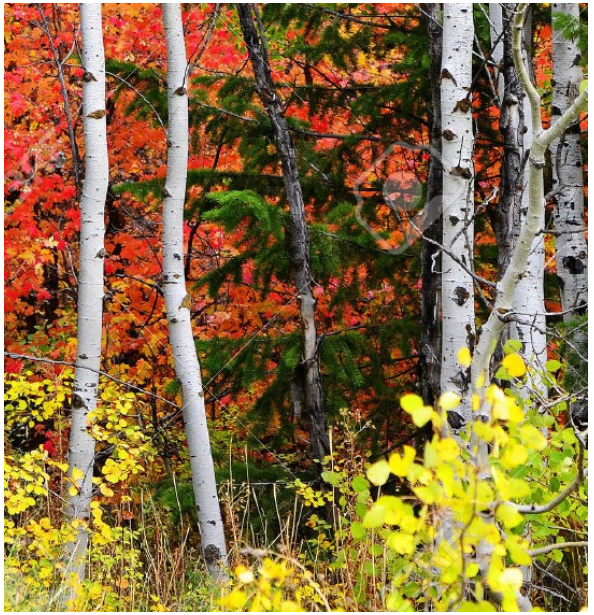
Trial #1



Trial #2

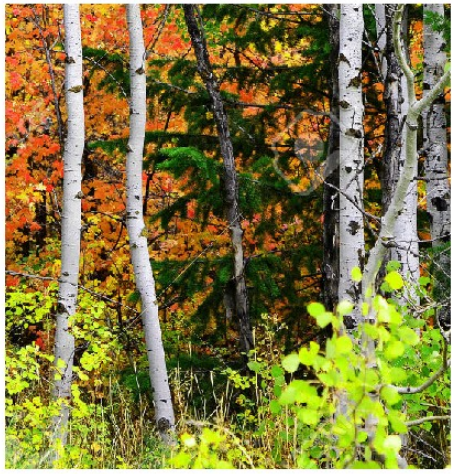
Trial #3

transforms.ColorJitter



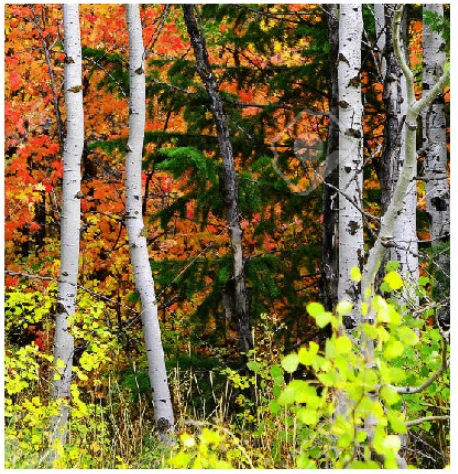
Hue

Essai #0



Essai #1

Essai #2



Essai #3

Méthode d'ensemble

- Entraîne quelques modèles (réseaux de neurones)
- Faire un vote / moyennage des sorties
- Fonctionne sous l'hypothèse que les modèles font des erreurs différentes
- Pour des architectures identiques, les prédictions varient quand même dues à l'initialisation, mini-batch aléatoire
- Donne souvent un gain de 1-2%

Méthode d'ensemble

Réseau	year	# crops	# modèles	Gain Top-1 (Erreur)	Gain Top-5 (Erreur)
AlexNet	2012		7		2.8% (15.4)
VGGnET	2014	-	2	0.7% (23.7%)	0% (6.8%)
GoogLeNet	2013	144	7	-	1.22% (6.7%)
BatchNorm-Inception	2015	144	6	1.9% (20.1%)	0.92% (4.9%)
Inception-v3	2015	144	4	1.57% (17.2%)	0.62% (3.6%)
Inception v4 + 3x Inception-ResNet-v2	2016	144	4	1.3% (16.5%)	0.6% (3.1%)

(La valeur entre parenthèse est le taux d'erreur, après ensemble)

Szegedy et al., Rethinking the Inception Architecture for Computer Vision, CPVR 2015.