

# GLO-4030/7030

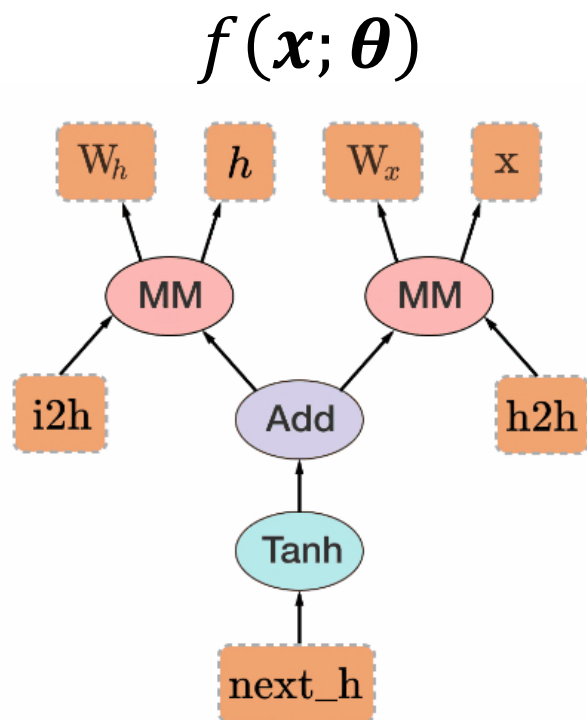
# APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

Optimisation pour  
l'apprentissage profond

# Plan

- 1) Rappel sur notions d'apprentissage
- 2) Algorithmes d'optimisation
- 3) Stratégies d'optimisation
- 4) Diagnostique

# Retour sur la semaine dernière



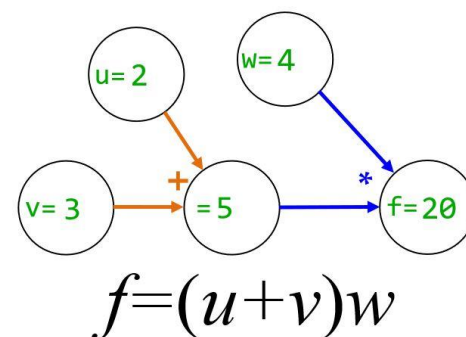
[pytorch.org](https://pytorch.org)

$$L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

Entropie croisée:  
 $L(\hat{y}, y) = -\log \hat{y}_{cible}$

$$\nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

Backprop



Cette semaine: comment améliorer les performances du réseau?

# Rappel sur notions d'apprentissage

# Notion d'apprentissage

- **Risque empirique  $J(\boldsymbol{\theta})$**

- Fonction de coût  $L$  + distribution de données empirique  $\hat{p}_{data}$ :

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{data}} L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

- **Vrai risque  $J^*(\boldsymbol{\theta})$**

- Fonction de coût  $L$  + vraie distribution de données  $p_{data}$  :

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{data}} L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

# Notion d'apprentissage

- Accès à  $p_{data} \rightarrow$  vrai risque  $J^*(\boldsymbol{\theta}) \rightarrow$  **optimisation** standard
- Accès à  $\hat{p}_{data} \rightarrow$  risque empirique  $J(\boldsymbol{\theta}) \rightarrow$  problème d'**apprentissage**
- On a jamais accès à  $p_{data}$ .
- En résumé
  - On minimise directement  $J(\boldsymbol{\theta})$ .
  - Espérer minimiser aussi  $J^*(\boldsymbol{\theta})$ .

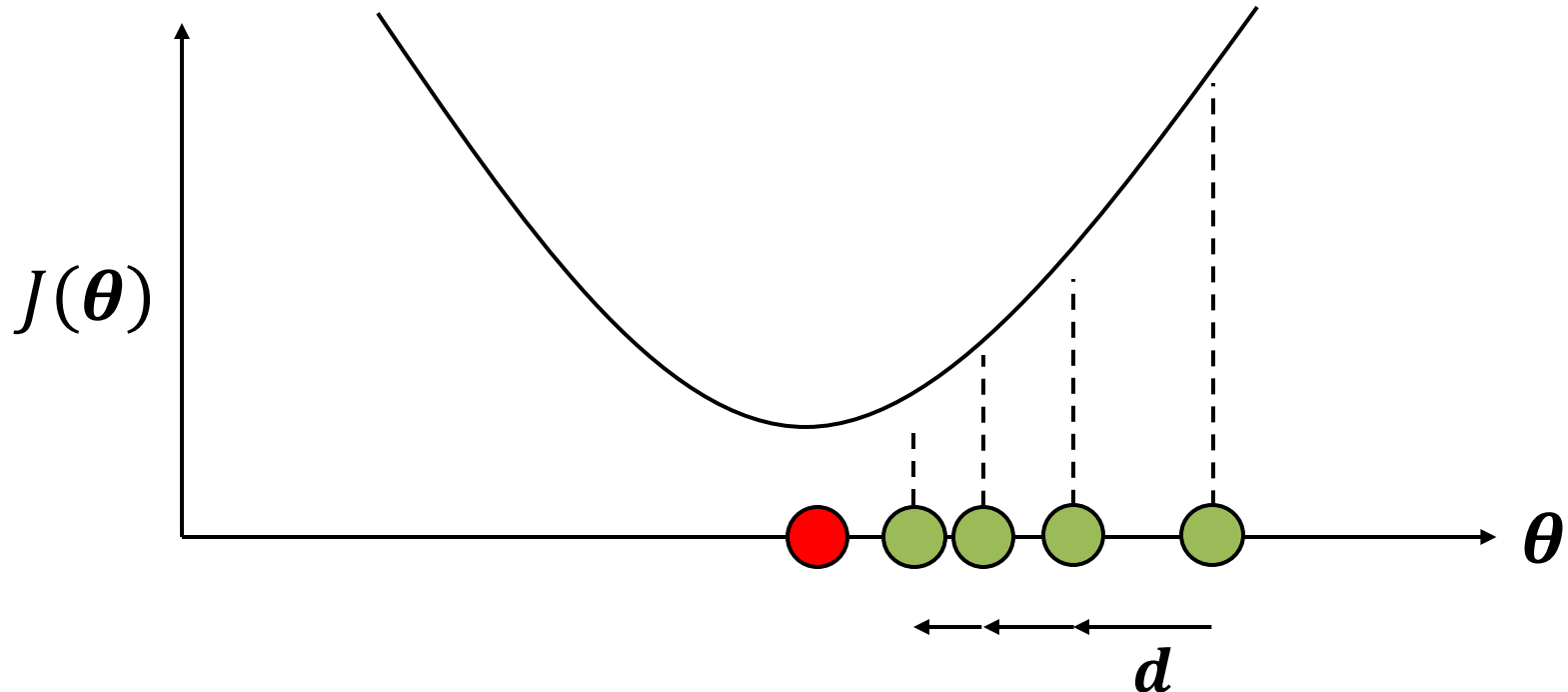
# Algorithmes d'optimisation

# Descente du gradient

- Concept
  - Déterminer un vecteur directionnel  $\mathbf{d}$  basé sur le gradient  $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ .
  - Mettre à jour les paramètres:
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$$
  - Répéter jusqu'à convergence.
- Pourquoi est-ce que ça fonctionne ?
  - $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbf{0} \rightarrow \boldsymbol{\theta}$  est un point critique.
  - Aide à trouver les extremums.



# Descente du gradient



- Convergence
  - $J(\theta)$  cesse de diminuer.
  - $J(\theta)$  oscille autour d'une certaine valeur (petite variance).

# Notion de gradient

- Rappel

$$\begin{aligned} J: \mathbb{R}^D &\rightarrow \mathbb{R} \\ \boldsymbol{\theta} &\mapsto J(\boldsymbol{\theta}) \end{aligned}$$

- Gradient

- $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \in \mathbb{R}^d \equiv$  **vecteur** des dérivées partielles de  $J$  par rapport aux entrées de  $\boldsymbol{\theta}$ :

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left[ \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1}, \dots, \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_D} \right]$$

# Notion de gradient

- Dérivation du gradient

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{(x,y) \sim \hat{p}_{data}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y) \\ &= \nabla_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^N L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \\ &= \frac{1}{N} \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})\end{aligned}$$

- $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \equiv \text{moyenne}$

# Algorithmes d'optimisation

- Descente du gradient:
  1. par batch
  2. stochastique
  3. avec momentum
  4. accéléré de Nesterov
  5. Adagrad
  6. RMSprop
  7. Adam

# 1. Descente du gradient par batch

- Vecteur directionnel

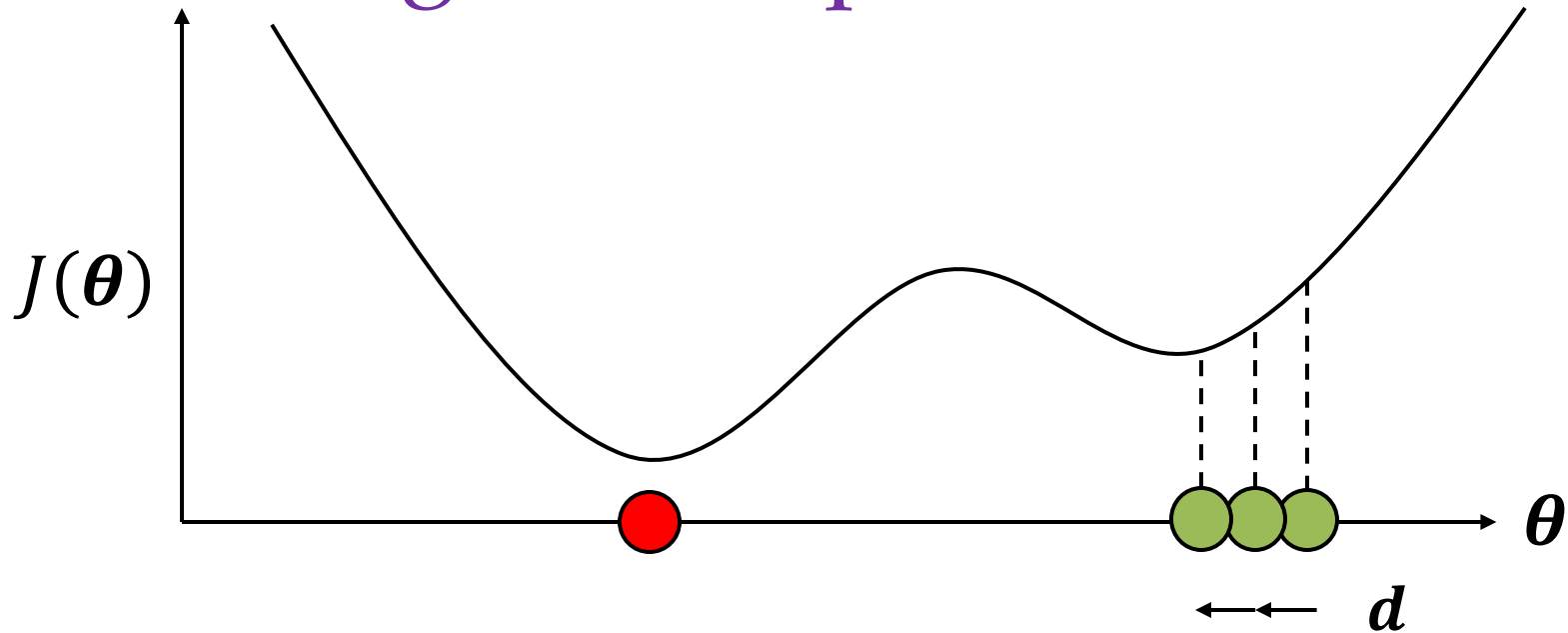
$$\begin{aligned} \mathbf{d} &= \epsilon_t \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\ &= \epsilon_t \cdot \frac{1}{N} \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \end{aligned}$$

$\epsilon_t$  = **taux d'apprentissage** à l'itération  $t$ .

- MAJ:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$$

# Inconvénients de la descente du gradient par batch



- Trop sensible au minimum / maximum locaux pour être utilisable en pratique.

# Inconvénients de la descente du gradient par batch

- Autres inconvénients
  - Traiter tous les exemples d'entraînement  $(\mathbf{x}^{(i)}, y^{(i)})$  donne **une seule** mise-à-jour.
  - Pas de MAJ entre  $(\mathbf{x}^{(i)}, y^{(i)})$  et  $(\mathbf{x}^{(i+j)}, y^{(i+j)})$
  - Erreur semblable  $\rightarrow$  gradient semblable  $\rightarrow$  redondance

## 2. Descente du gradient stochastique

- SGD
  - *stochastic gradient descent*
  - Une **batch** de taille  $m \rightarrow$  une MAJ

- Vecteur directionnel

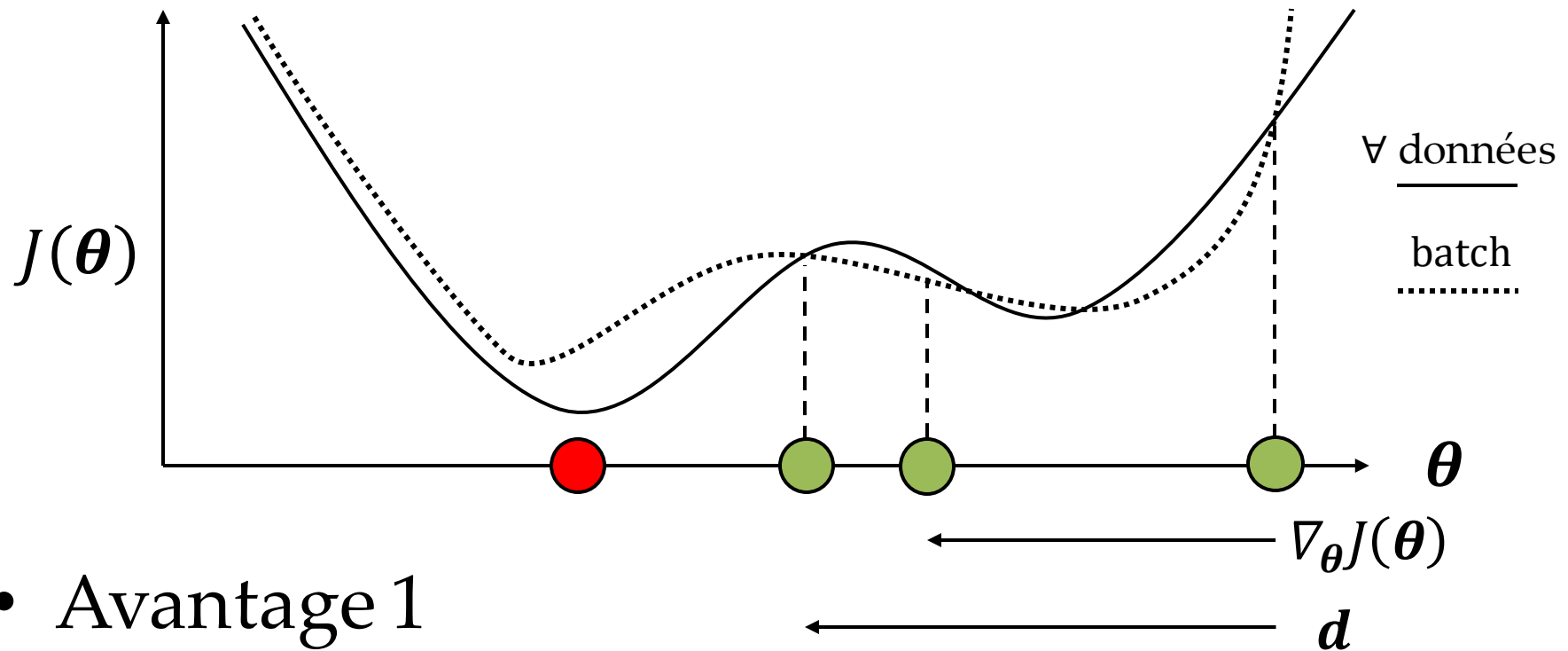
$$\mathbf{d} \leftarrow \epsilon_t \cdot \hat{\mathbf{g}} = \epsilon_t \cdot \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$$

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
2. Gradient:  $\hat{\mathbf{g}}$
3. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$



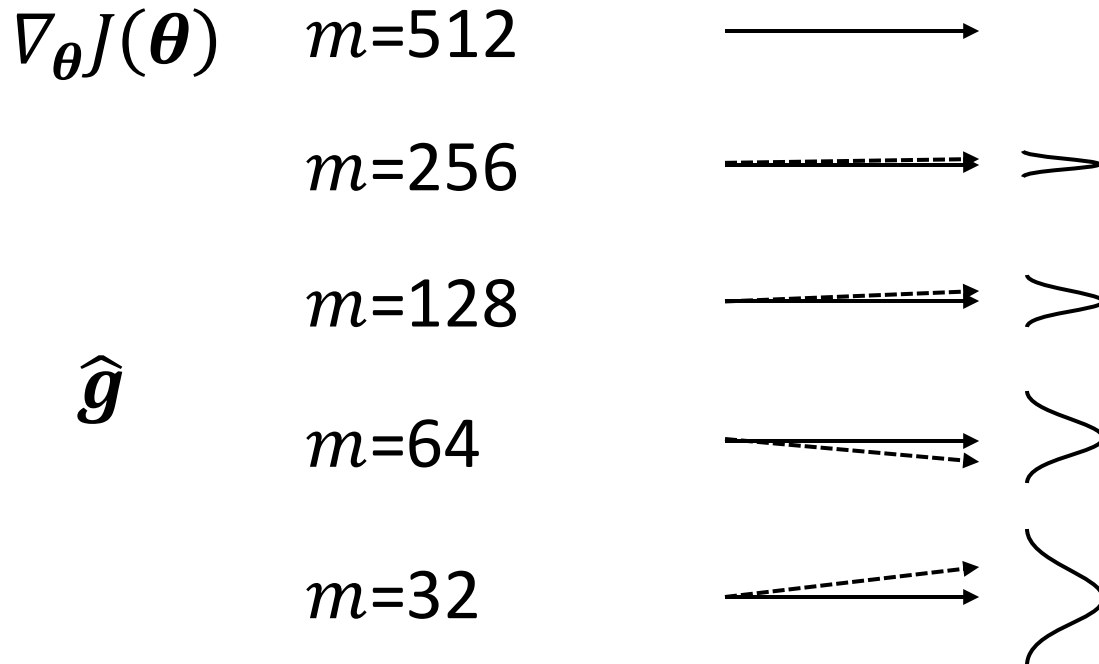
# Avantages de la descente du gradient stochastique



- Avantage 1
  - $\nabla_{\theta} J(\theta) \approx \hat{g}$
  - $\hat{g}$  = version bruitée du gradient.
  - Bruit aide à sortir des min / max locaux.

# Avantages de la descente du gradient stochastique

- Avantage 2
  - Une batch de taille  $m$  réduit l'incertitude du gradient de  $O(\sqrt{m})$ .
  - Direction semblable avec moins de données.

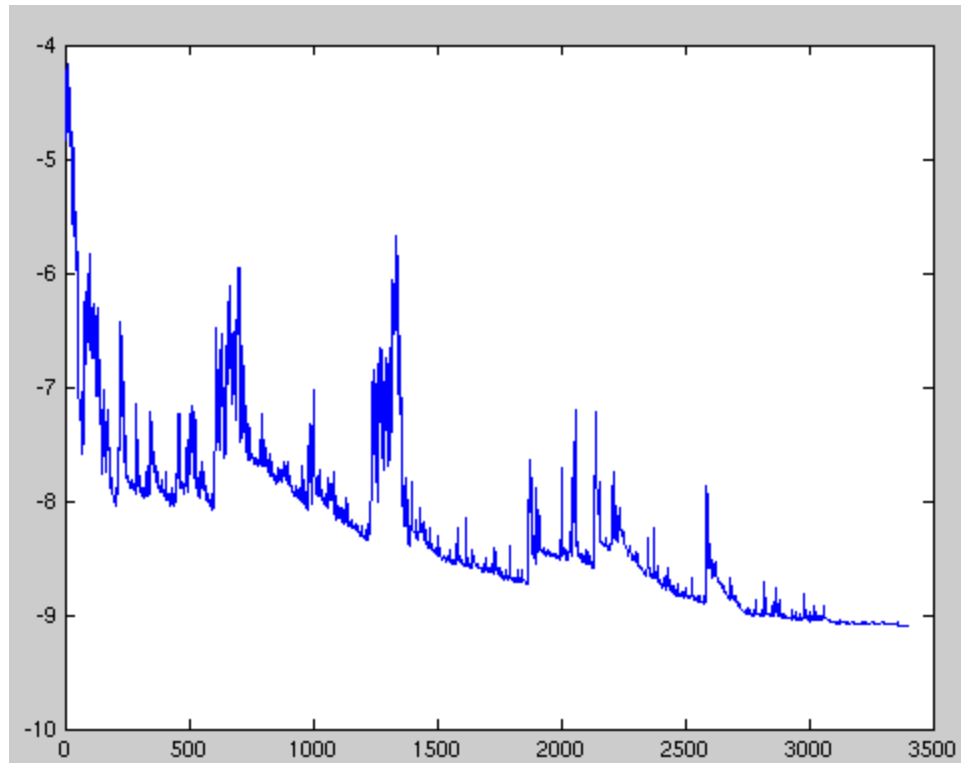


# Avantages de la descente du gradient stochastique

- Avantage 3
  - MAJ entre  $(\mathbf{x}^{(i)}, y^{(i)})$  et  $(\mathbf{x}^{(i+j)}, y^{(i+j)})$
  - Diminue la redondance si erreur semblable.
  - Convergence possible avant même de voir tous les exemples.
- Avantage 4
  - Traitement parallèle de la batch.
- Avantage 5
  - Optimisation hardware quand  $m = 2^k$

# Inconvénient #1 de la descente du gradient stochastique

- Variance  $\rightarrow$  fluctuation



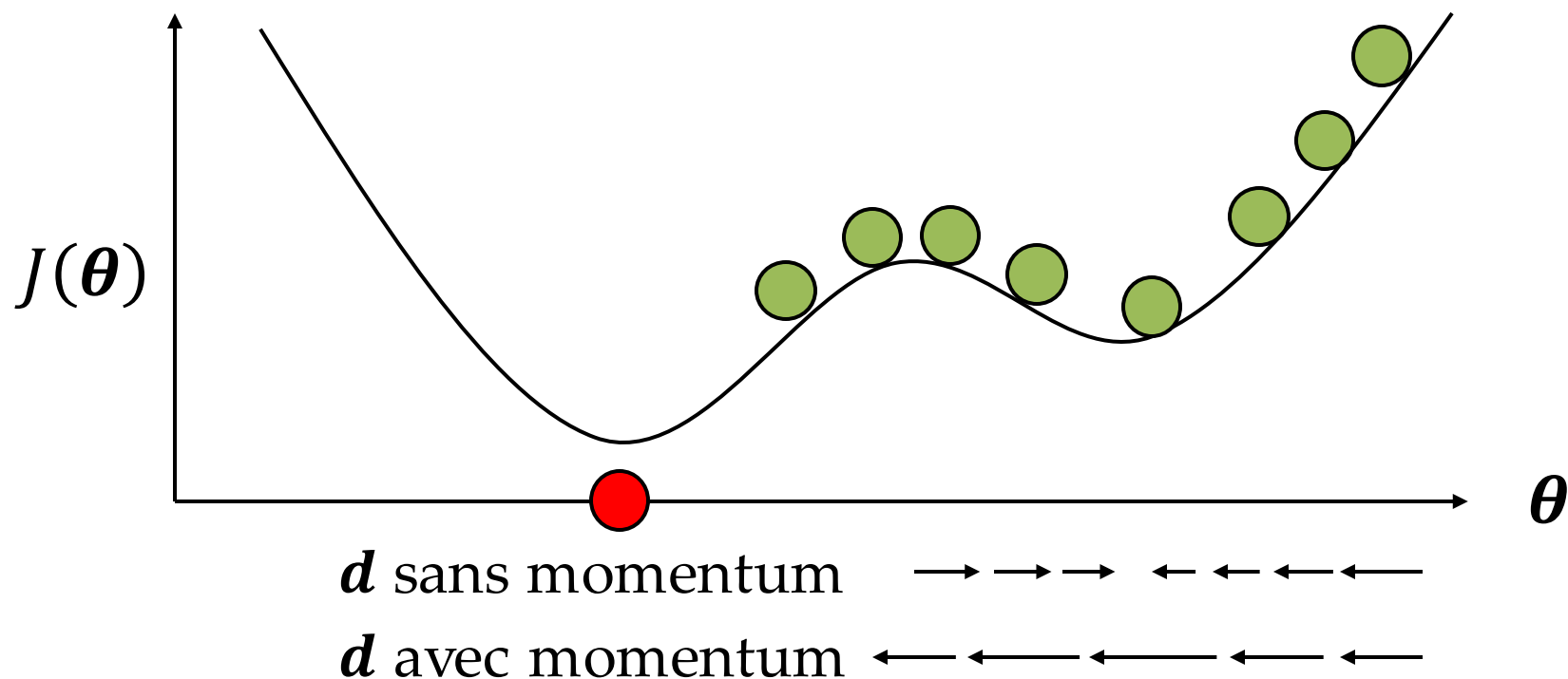
[Wikipédia](#)

# Inconvénient #2 de la descente du gradient stochastique

- Taux d'apprentissage  $\epsilon$  a une grande influence
  - À suivre... (section stratégies d'optimisation)

# 3. SGD + Momentum

- Concept
  - Ajouter à SGD une dynamique Newtonnienne.

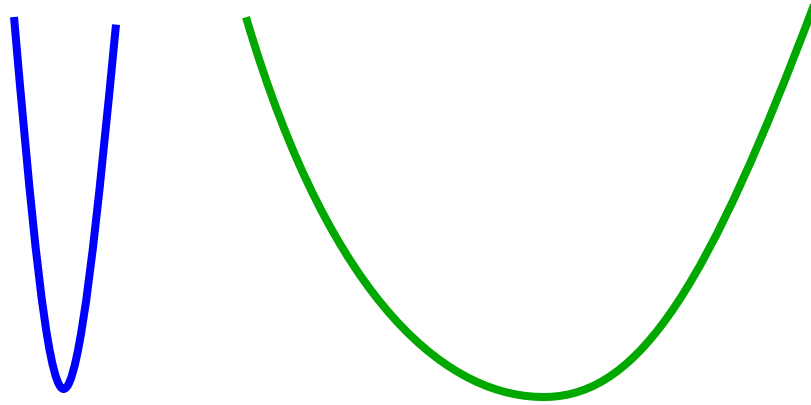


# SGD + Momentum

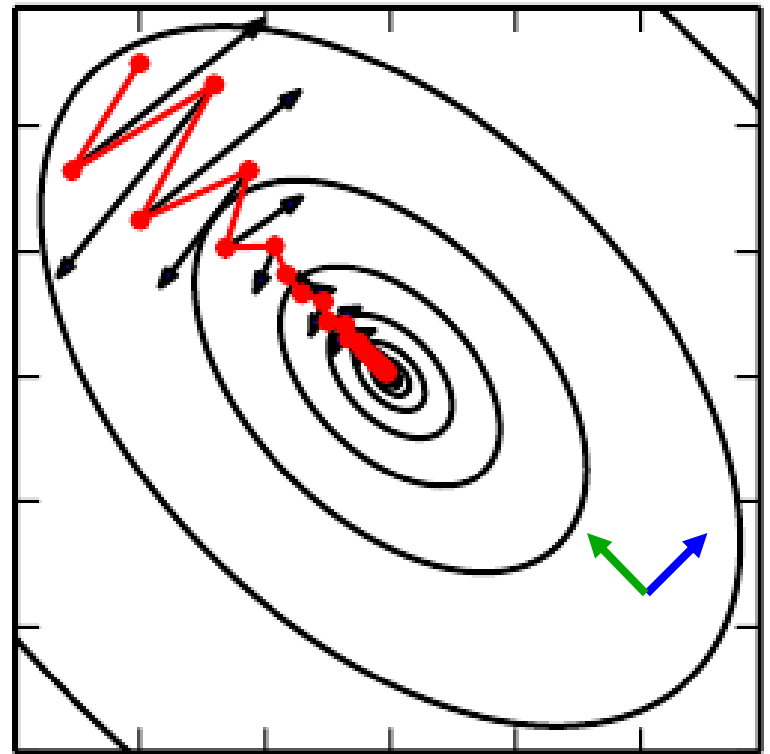
- Vitesse  $\mathbf{v} \in \mathbb{R}^D$ 
  - Estimation des directions précédentes.
- Facteur d'oubli  $\alpha \in [0, 1)$ 
  - $\alpha \rightarrow 0 \equiv$  SGD standard
  - Valeur suggérée:  $\alpha = 0.9$
- Algorithme
  1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient:  $\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  3. Vitesse:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon_t \cdot \hat{\mathbf{g}}$
  4. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

# Avantage de SGD + Momentum

- Avantage
  - Le momentum aide lorsqu'il y a des ravins.
  - Les MAJ de  $\theta$  tendent à s'aligner.

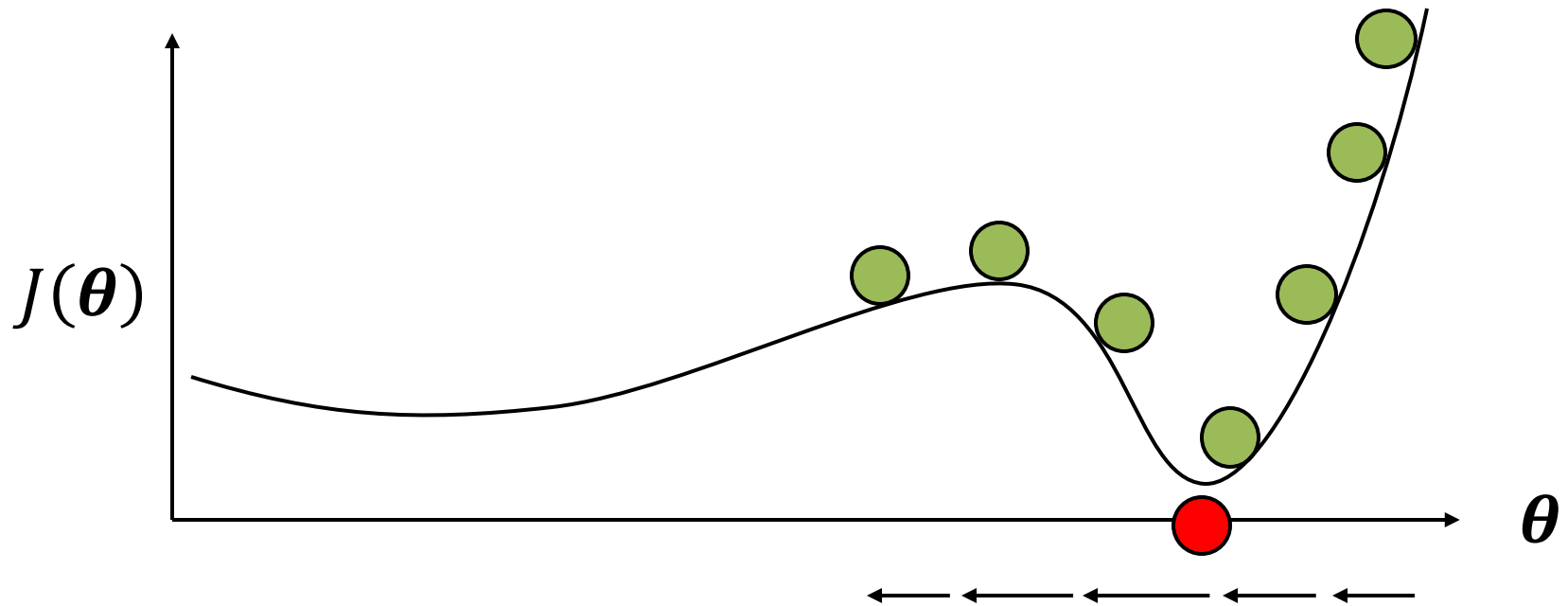


rouge = avec momentum  
noir = sans momentum



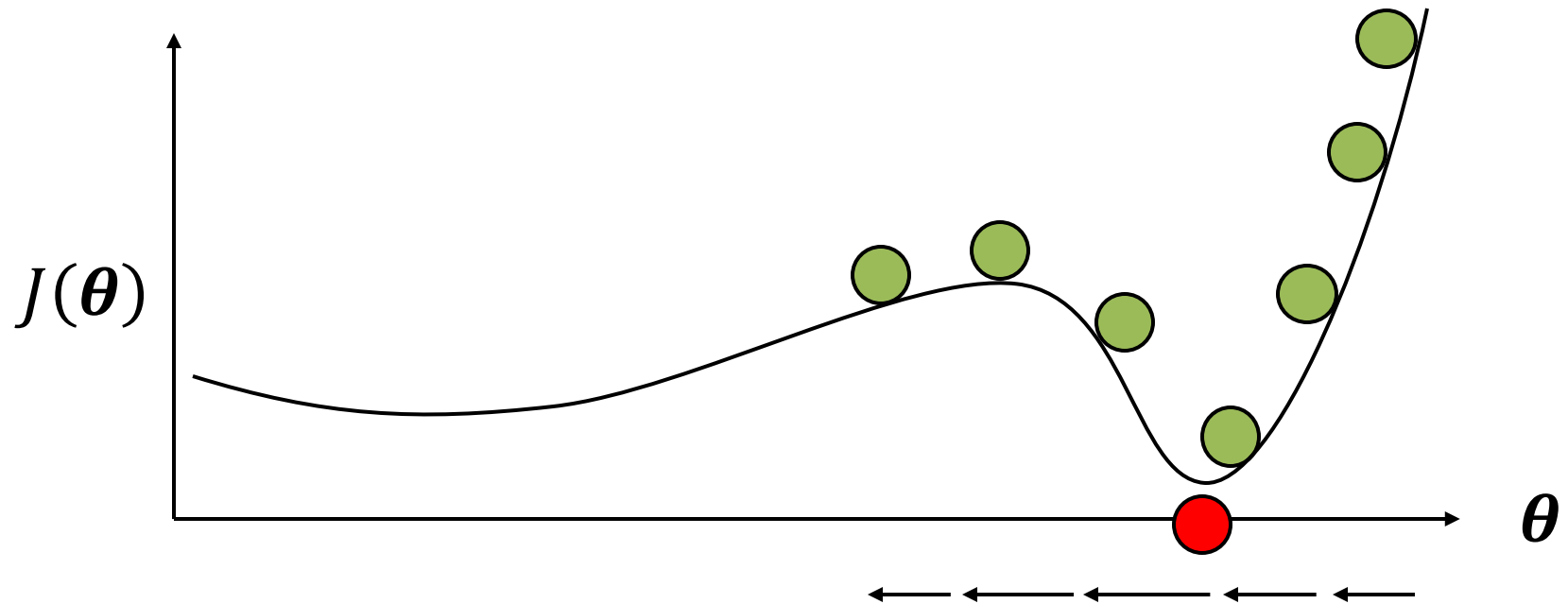


# Inconvénient de SGD + Momentum



- Problème
  - Peut survoler les min / max locaux.
  - Mais, peut dépasser le min global.

# Inconvénient de SGD + Momentum



- Problème
  - Peut survoler les min / max locaux.
  - Mais, peut dépasser le min global.

# 4. SGD + Momentum accéléré de Nesterov

- But
  - MAJs s'adaptent plus rapidement à la courbe.
- Concept
  - Appliquer une **correction** sur le *jump* du momentum.
- Différence importante
  - La position où on évalue le gradient.

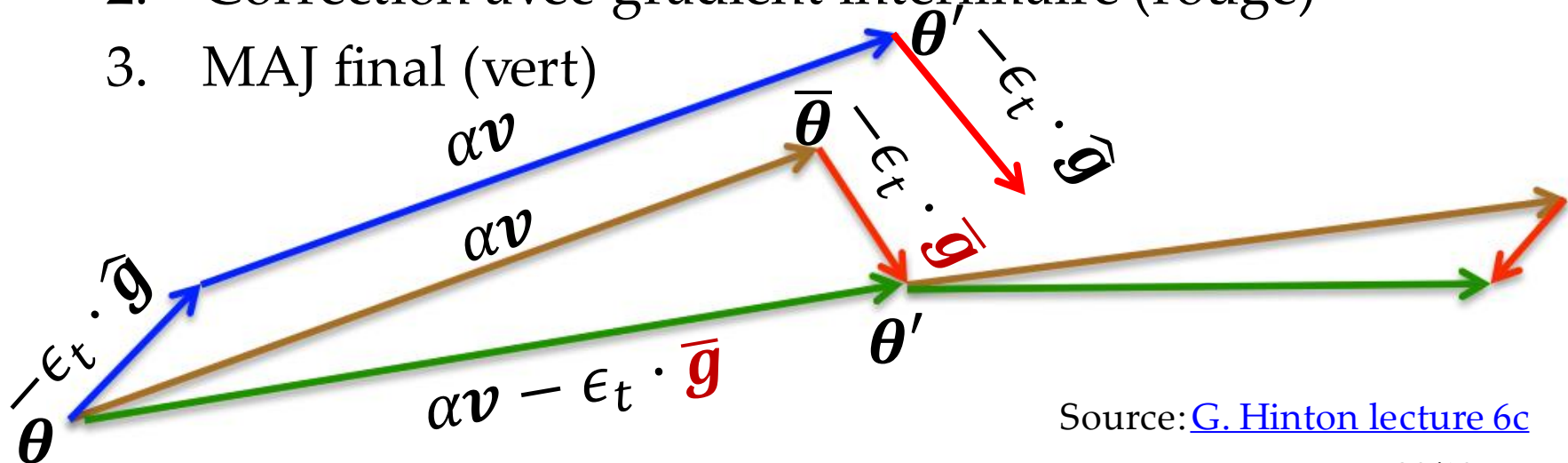
# SGD + Momentum de Nesterov

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
2. Point intérimaire:  $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \mathbf{v}$
3. Gradient:  $\bar{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\bar{\boldsymbol{\theta}}} L(f(\mathbf{x}^{(i)}; \bar{\boldsymbol{\theta}}), y^{(i)})$
4. Vitesse:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon_t \cdot \bar{\mathbf{g}}$
5. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

# SGD + Momentum de Nesterov

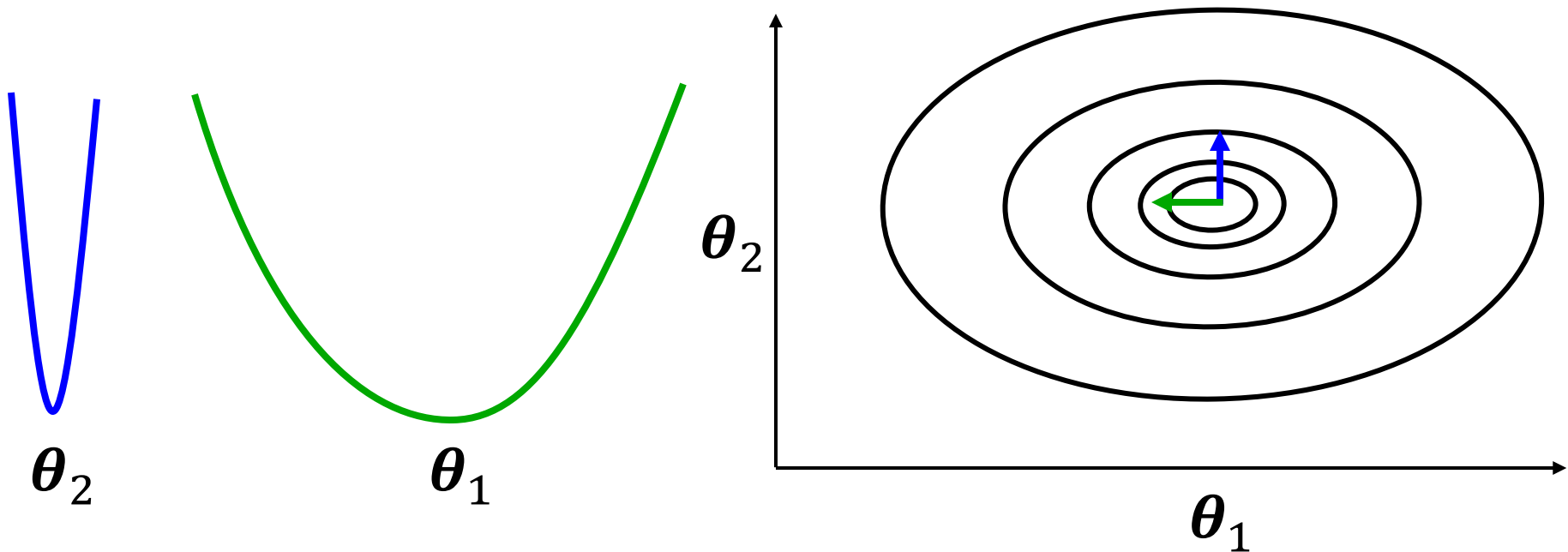
- Momentum standard
  1. Gradient (petit bleu)
  2. Jump direction des gradients accumulés (grand bleu)
- Nesterov
  1. Jump direction des gradients accumulés (brun)
  2. Correction avec gradient intérimaire (rouge)
  3. MAJ final (vert)



Source: [G. Hinton lecture 6c](#)

# 5. Adagrad

- Concept
  - Adapter la MAJ de chaque paramètre  $\theta_d$  **individuellement** pour uniformiser le taux de changement.



# Adagrad

- Principe
  - $\theta_k$  historique petite MAJ  $\rightarrow$  augmenter la MAJ à  $k$ .
  - $\theta_k$  historique grande MAJ  $\rightarrow$  diminuer la MAJ à  $k$ .
- Méthode
  - **Mettre à l'échelle** la MAJ avec l'inverse de la racine carrée de la somme des gradients.
- Accumulation des gradients  $\mathbf{r} \in \mathbb{R}^D$ 
  - Accumule le carré des gradients.
- Petite constante  $\delta \in \mathbb{R}$  pour stabilité numérique.

# Adagrad

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$

2. Gradient:  $\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

3. Accumulation:  $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

4. Mise à l'échelle:  $\mathbf{d} \leftarrow \frac{\epsilon_t}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

5. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$



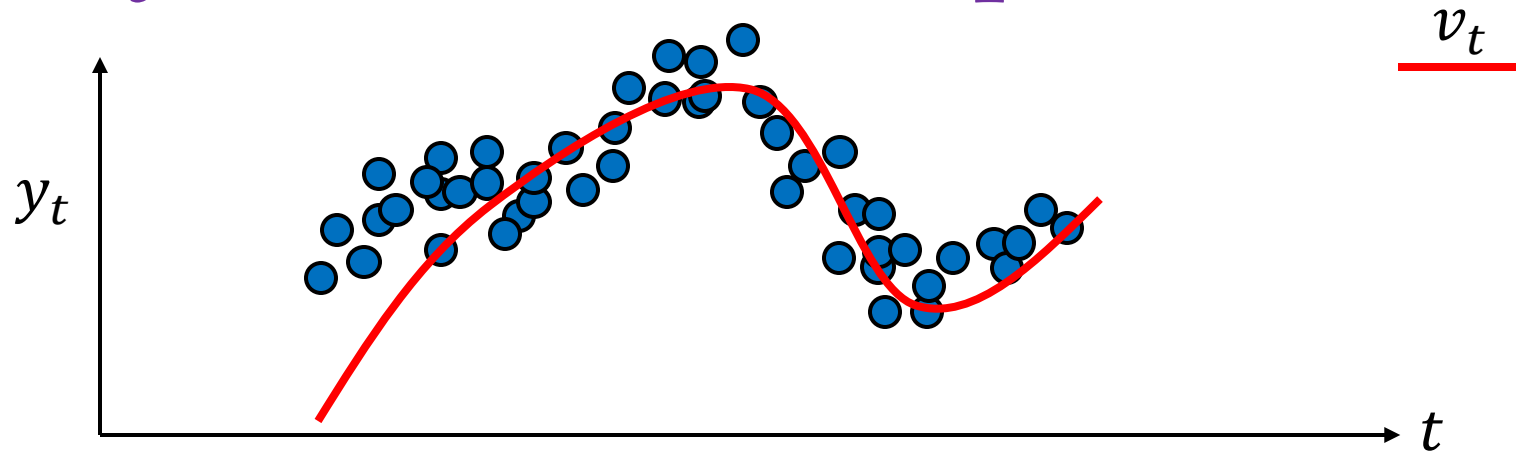
# Inconvénient de Adagrad

- Croissance excessive de  $\mathbf{r}$ 
  - Accumulation de valeurs positives  $\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ .
  - Les taux d'apprentissage mis à l'échelle  $\frac{\epsilon_t}{\delta + \sqrt{\mathbf{r}}} \rightarrow \mathbf{0}$  rapidement.

## 6. RMSprop

- Solution pour régler la croissance de  $r$ .
- Concept
  - Remplacer:
    - Accumuler  $\hat{g} \odot \hat{g}$  pour tous les  $\hat{g}$  depuis le début.
  - Par:
    - Accumuler  $\hat{g} \odot \hat{g}$  que pour les plus récents  $\hat{g}$ .
- Méthode
  - Utiliser une moyenne mobile exponentielle à taux de décroissance  $\rho \in [0, 1]$ .
  - Valeur suggérée:  $\rho = 0.99$

# Moyenne mobile exponentielle



$$v_t = \rho v_{t-1} + (1 - \rho) y_t$$

$$v_0 = 0$$

$$\begin{aligned} v_1 &= 0.99 \cdot v_0 + 0.01 \cdot y_1 \\ &= 0.01 \cdot y_1 \end{aligned}$$

$$\begin{aligned} v_2 &= 0.99 \cdot v_1 + 0.01 \cdot y_2 \\ &= 0.0099 \cdot y_1 + 0.01 \cdot y_2 \end{aligned}$$

# RMSprop

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$

2. Gradient:  $\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

3. Accumulation:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

4. Mise à l'échelle:  $\mathbf{d} \leftarrow \frac{\epsilon_t}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

5. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$

# 7. RMSprop + Momentum

- Méthode:
  - Comme pour SGD, on peut ajouter du momentum aux direction calculées.
- Vitesse  $\mathbf{v} \in \mathbb{R}^D$ 
  - Estimation des directions **mises à l'échelle** précédentes.
- Facteur d'oubli  $\alpha \in [0, 1)$ 
  - $\alpha \rightarrow 0 \equiv$  RMSprop standard

# RMSprop + Momentum

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$

2. Gradient:  $\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

3. Accumulation:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

4. Vitesse:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\epsilon_t}{\sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

5. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$

## 8. RMSprop + Nesterov

- Méthode:
  - Comme pour SGD + Nesterov, on peut utiliser le momentum accéléré de Nesterov.
  - Appliquer une **correction mise à l'échelle** du *jump* du momentum.

# RMSprop + Nesterov

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$

2. Point intérimaire:  $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \mathbf{v}$

3. Gradient:  $\bar{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\bar{\boldsymbol{\theta}}} L(f(\mathbf{x}^{(i)}; \bar{\boldsymbol{\theta}}), y^{(i)})$

4. Accumulation:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \bar{\mathbf{g}} \odot \bar{\mathbf{g}}$

5. Vitesse:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\epsilon_t}{\sqrt{\mathbf{r}}} \odot \bar{\mathbf{g}}$

6. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$



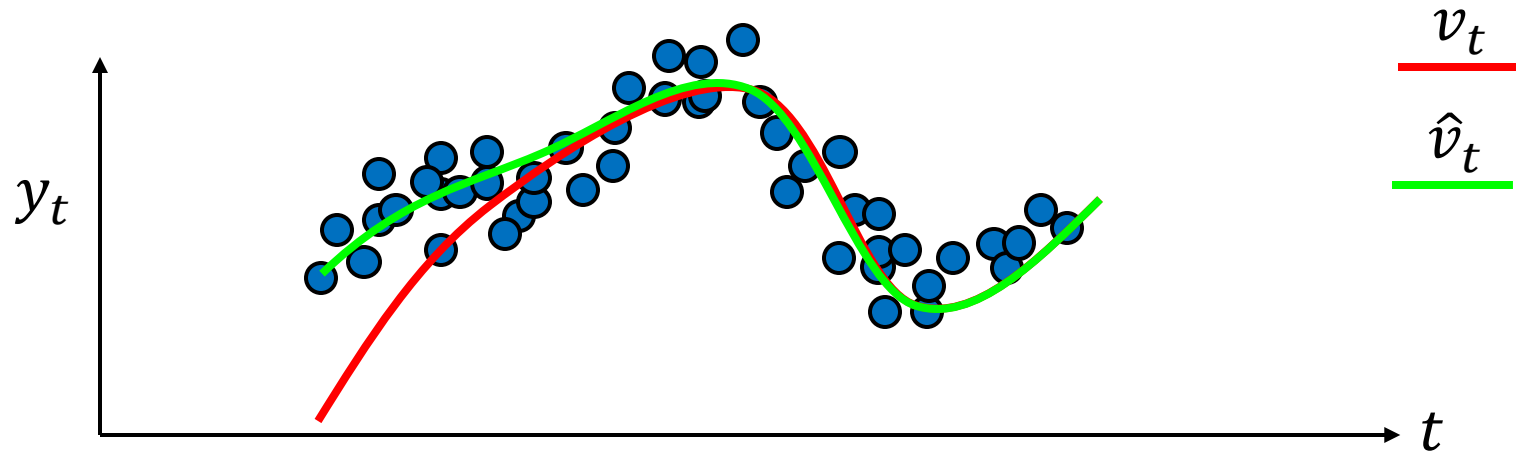
# Inconvénients de RMSprop

- Inconvénient 1
  - $\mathbf{r}$  est une estimation du second moment (variance décentrée) du gradient  $\hat{\mathbf{g}}$ .
  - Cet estimateur a un large biais positif au début de la descente du gradient.
- Inconvénient 2
  - L'utilisation du momentum en combinaison avec de la mise-à-l'échelle (étape vélocité) n'a pas une justification théorique claire.

## 9. Adam

- Solution pour résoudre les inconvénients de RMSprop + momentum / Nesterov.
- Méthode:
  - Calculer une estimation du premier et second moments avec une moyenne mobile exponentielle à taux  $\rho_1, \rho_2 \in [0, 1)$ .
    - Valeurs suggérées:  $\rho_1 = 0.9$  et  $\rho_2 = 0.999$
  - Corriger les biais des moments.
  - Le premier moment normalisé par le second moment donne la direction de MAJ.

# Correction des biais



$$v_t = \rho v_{t-1} + (1 - \rho) y_t$$

$$v_0 = 0$$

$$\begin{aligned} v_1 &= 0.99 \cdot v_0 + 0.01 \cdot y_1 \\ &= 0.01 \cdot y_1 \end{aligned}$$

$$\begin{aligned} v_2 &= 0.99 \cdot v_1 + 0.01 \cdot y_2 \\ &= 0.0099 \cdot y_1 + 0.01 \cdot y_2 \end{aligned}$$

$$\hat{v}_t = \frac{v_t}{1 - (\rho)^t}$$

$$\begin{aligned} \hat{v}_2 &= \frac{v_2}{1 - 0.99^2} \\ &= \frac{0.0099 \cdot y_1 + 0.01 \cdot y_2}{0.0199} \\ &= 0.497 \cdot y_1 + 0.503 \cdot y_2 \end{aligned}$$

# Adam

- Algorithme

1. Échantillonnage:  $(\mathbf{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$

2. Gradient:  $\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$

3. 1er moment:  $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \hat{\mathbf{g}}$

4. 2e moment:  $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

5. Correction biais 1er moment :  $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - (\rho_1)^t}$

6. Correction biais 2e moment :  $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - (\rho_2)^t}$

7. Direction:  $\mathbf{d} \leftarrow \epsilon_t \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}$

8. MAJ:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{d}$

# Tableau comparatif

Approche	Hyper-paramètre	Utilisation mémoire
Batch	$\epsilon_t$	D
<b>SGD</b>	$\epsilon_t, m$	D
SGD + Momentum	$\epsilon_t, m, \alpha$	D*2
<b>SGD + Nesterov</b>	$\epsilon_t, m, \alpha$	D*2
Adagrad	$\epsilon_t, m, \delta$	D*2
RMSProp	$\epsilon_t, m, \rho$	D*2
RMSProp + Momentum	$\epsilon_t, m, \rho, \alpha$	D*3
RMSProp + Nesterov	$\epsilon_t, m, \rho, \alpha$	D*3
<b>Adam</b>	$\epsilon_t, m, \rho_1, \rho_2, \delta$	D*3

[PyTorch documentation](#)

# Adam

- Inconvénient
  - Augmentation rapide du taux d'apprentissage effectif causé lorsque le 2e moment précédent est beaucoup plus grand que le gradient estimé.

# Yogi

- Amélioration de Adam
- Le but est de contrôler l'augmentation rapide du taux d'apprentissage effectif
- Taille de la minibatch augmente avec le temps

# Yogi

- Algorithme

- Échantillonnage:  $(x^{(i)}, y^{(i)}) \sim \hat{p}_{data}$

- Gradient:  $\hat{g}_t = \frac{1}{m} \sum_{i=1}^m \nabla_{\Theta} L(f(x^{(i)}; \Theta), y^{(i)})$

- 1er moment:  $s_t = \rho s_{t-1} + (1 - \rho) \hat{g}_t$

- **2e moment:**  $r_t = r_{t-1} - (1 - \beta_2) \text{sign}(r_{t-1} - \hat{g}_t^2) \hat{g}_t^2$

- Correction biais 1er moment:  $\hat{s}_t = \frac{s}{1 - \rho_1^t}$

- Correction biais 2e moment:  $\hat{r}_t = \frac{r}{1 - \rho_2^t}$

- Direction:  $d = \epsilon_t \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$

- MAJ:  $\theta \leftarrow \theta - d$



# Yogi

Table 1: Train and test loss comparison for Deep AutoEncoders. Standard errors with  $2\sigma$  are shown over 6 runs are shown. All our experiments were run for 5000 epochs utilizing the ReduceLROnPlateau schedule with patience of 20 epochs and decay factor of 0.5 with a batch size of 128.

Method	LR	$\beta_1$	$\beta_2$	$\epsilon$	MNIST	
					Train Loss	Test Loss
PT + NCG [20]	-	-	-	-	2.31	2.72
RAND+HF [20]	-	-	-	-	1.64	2.78
PT + HF [20]	-	-	-	-	1.63	2.46
KSD [36]	-	-	-	-	1.8	2.5
HF [36]	-	-	-	-	1.7	2.7
ADAM (Default)	$10^{-3}$	0.9	0.999	$10^{-8}$	$1.85 \pm 0.19$	$4.36 \pm 0.33$
ADAM (Modified)	$10^{-3}$	0.9	0.999	$10^{-3}$	$0.91 \pm 0.04$	$1.88 \pm 0.07$
YOGI (Ours)	$10^{-2}$	0.9	0.9	$10^{-3}$	$0.78 \pm 0.02$	$1.70 \pm 0.03$
YOGI (Ours)	$10^{-2}$	0.9	0.999	$10^{-3}$	$0.88 \pm 0.02$	<b><math>1.36 \pm 0.01</math></b>

# Stratégies d'optimisation

# Stratégies d'optimisation

Stratégies pour améliorer l'optimisation:

1. Choisir les hyperparamètres
2. Initialisation des paramètres
3. Normalisation

# 1. Choisir les hyperparamètres

- Trois façons de choisir les hyperparamètres:
  1. Horaire d'entraînement
  2. Taux d'apprentissage cyclique
  3. Recherche en grille
  4. Recherche aléatoire
  5. "Learning rate finder"

# 1.1 Horaire d'entraînement

- Principe
  - Établir à l'avance un horaire indiquant la valeur des hyperparamètres en fonction de l'époque.
  - Le plus souvent utilisé pour définir **les taux d'apprentissage  $\epsilon_t$** .
  - *Plus un art qu'une science.*

# Exemple d'horaire d'entraînement

- Nombre d'époque maximum: 200
- Taille de la batch:  $m = 32$
- Algorithme d'optimisation: SGD + Nesterov  $\alpha = 0.9$
- Horaire:

Époque	0	75	125	175
Taux d'apprentissage	0.1	0.01	0.001	0.0001
Weight decay	1e-4	5e-4	1e-5	5e-5

# Stratégie #1 pour choisir l'horaire

- Principe
  - $J(\boldsymbol{\theta})$  atteint un plateau  $\rightarrow$  changer l'hyperparamètre.
  - Produira une courbe en forme d'**escalier**.
- Méthode
  1. Définir la valeur initiale.
  2. Définir le taux de décroissance  $\beta \in (0, 1]$ .
  3. Définir les époques de MAJ en identifiant les plateaux de  $J(\boldsymbol{\theta})$ .

# Stratégie #1 pour choisir l'horaire du taux d'apprentissage $\epsilon$

- Exemple: taux d'apprentissage  $\epsilon$ .
- Valeur initiale  $\epsilon_0$ 
  - Trop large: grandes oscillations  $\rightarrow J(\boldsymbol{\theta})$  augmente  $\rightarrow$  divergence (NaNs, Inf)
  - Trop petit: convergence local  $\rightarrow J(\boldsymbol{\theta})$  grande valeur  $\rightarrow$  piètre solution  $\boldsymbol{\theta}$
  - Stratégie pour  $\epsilon_0$ :
    - Prendre approximativement la plus grande valeur de  $\epsilon_0$  qui ne fait pas diverger  $J(\boldsymbol{\theta})$ .



# Stratégie #1 pour choisir l'horaire du taux d'apprentissage $\epsilon$

- Taux de décroissance
  - MAJ:  $\epsilon \leftarrow \epsilon \cdot \beta$
  - Représente l'aggressivité de la convergence.
  - Convergence rapide quand  $\beta \rightarrow 0$
  - Convergence lente quand  $\beta \rightarrow 1$
  - Valeurs suggérées:  $\beta \in \{0.1, 0.2, 0.5\}$

# Stratégie #1 pour choisir l'horaire du taux d'apprentissage $\epsilon$

- Époques de MAJ
  - Observer  $J(\boldsymbol{\theta})$
  - Soit  $t_e$  l'époque où  $J(\boldsymbol{\theta})$  converge.
  - Ajouter à l'horaire d'entraînement une entrée avec époque =  $t_e$ .
  - Recommencer avec la nouvelle horaire.
  - Arrêter lorsque la MAJ de  $\epsilon$  ne fait plus diminuer  $J(\boldsymbol{\theta})$ .

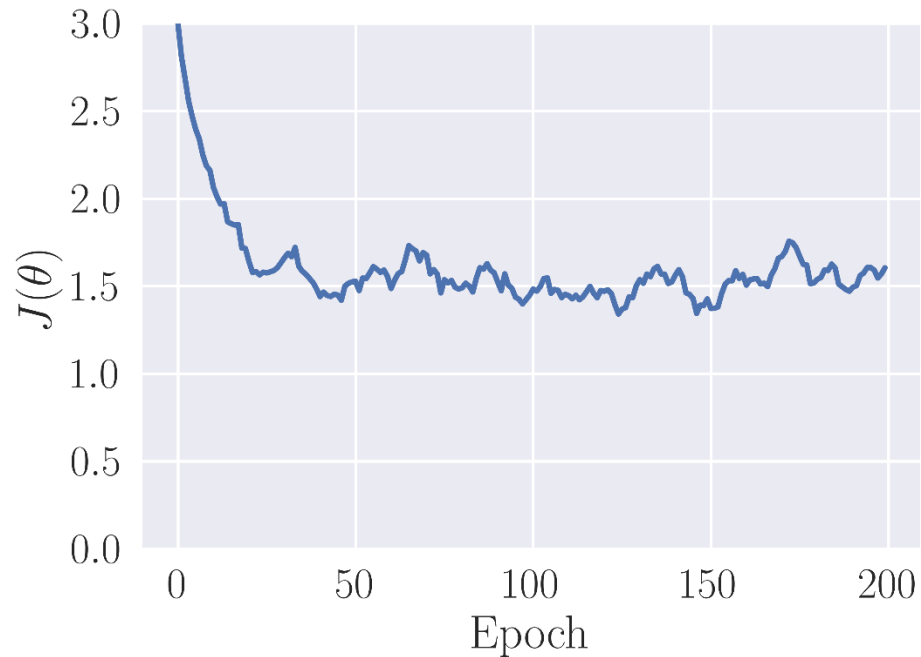
# Exemple d'horaire du taux d'apprentissage $\epsilon$

- Exemple
- Valeur initiale  $\epsilon_0$ 
  - 10  $\rightarrow$  nan, 1  $\rightarrow$  nan, 0.1  $\rightarrow$  ok
- Taux de décroissance
  - $\beta = 0.1$
- Époques de MAJ
  - Commençons avec l'horaire suivante:

Époque	0
Taux d'apprentissage	0.1

# Exemple d'horaire du taux d'apprentissage $\epsilon$

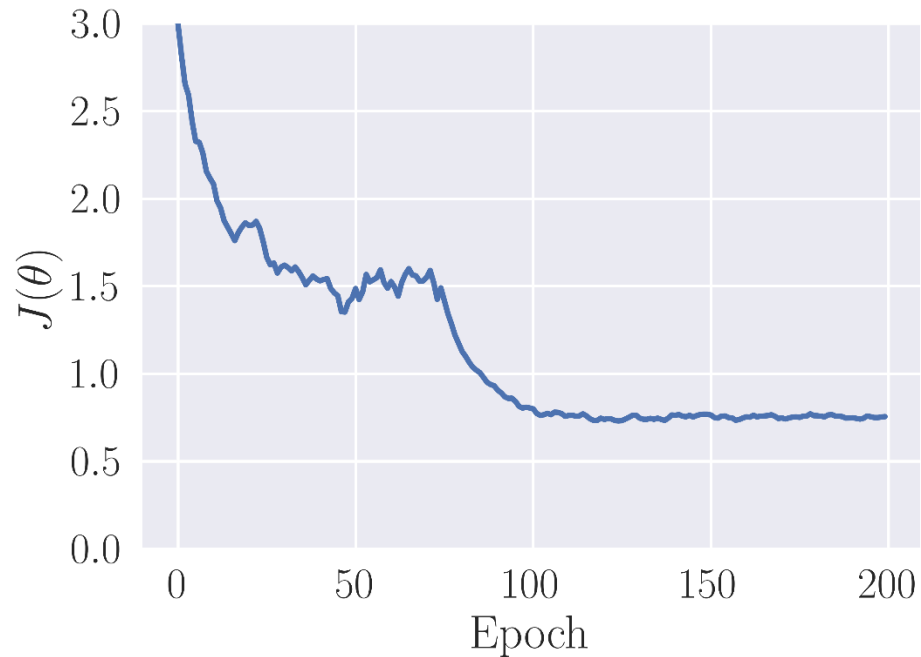
Convergence  
aux alentours  
de l'époque 75.



Époque	0
Taux d'apprentissage	0.1

# Exemple d'horaire du taux d'apprentissage $\epsilon$

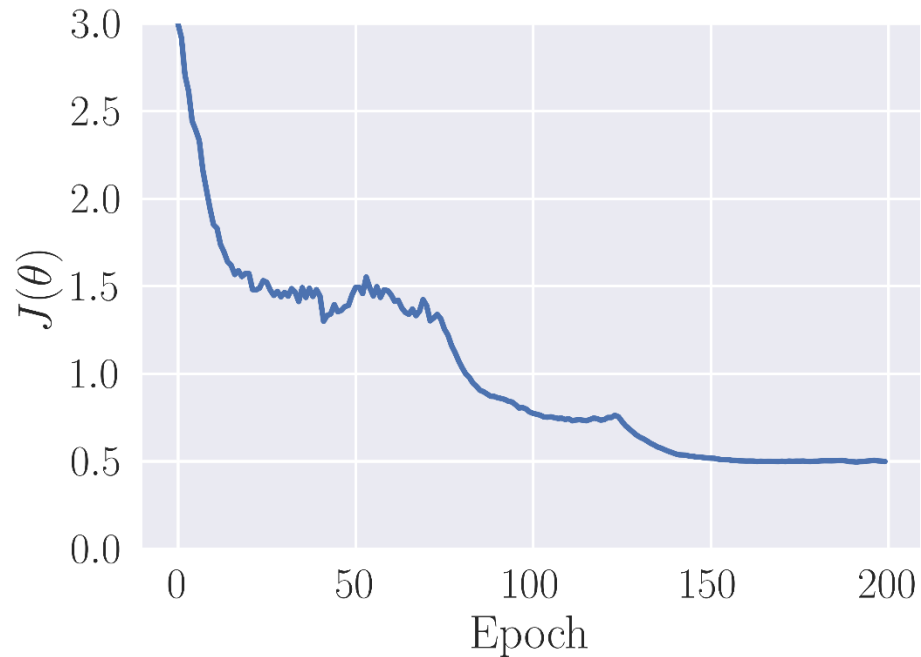
Convergence  
aux alentours  
de l'époque  
125.



Époque	0	75
Taux d'apprentissage	0.1	0.01

# Exemple d'horaire du taux d'apprentissage $\epsilon$

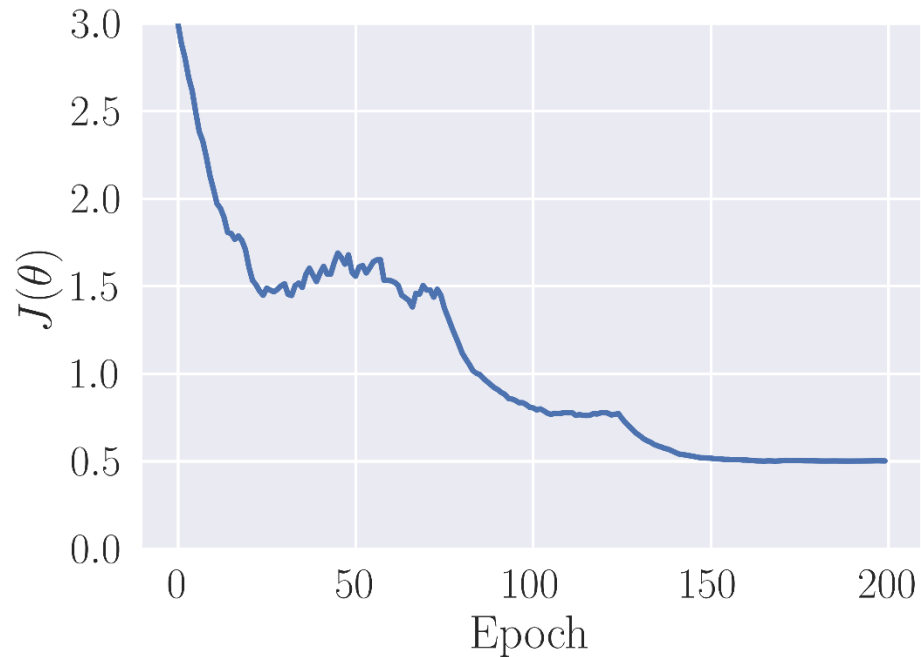
Convergence  
aux alentours  
de l'époque  
175.



Époque	0	75	125
Taux d'apprentissage	0.1	0.01	0.001

# Exemple d'horaire du taux d'apprentissage $\epsilon$

Aucun gain après époque 175.



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

# Stratégie #2 pour choisir l'horaire

- Principe
  - Changer l'hyper-parameter à chaque époque  $t_e$ .
  - Produira une courbe **sans plateau**.
- Méthode
  1. Définir la valeur initiale.
  2. Définir le taux de décroissance  $\beta \in (0, 1]$ .
  3. Définir la règle de décroissance.



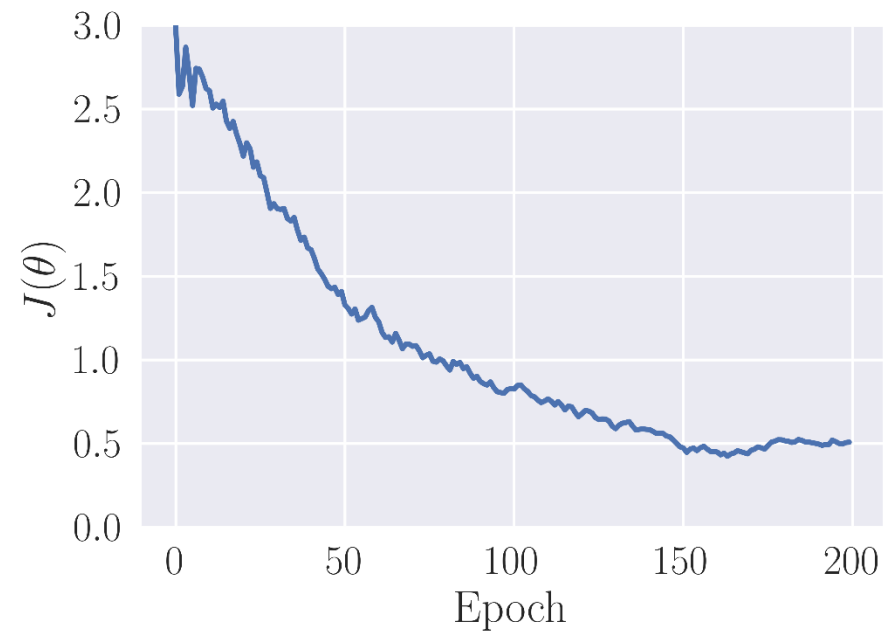
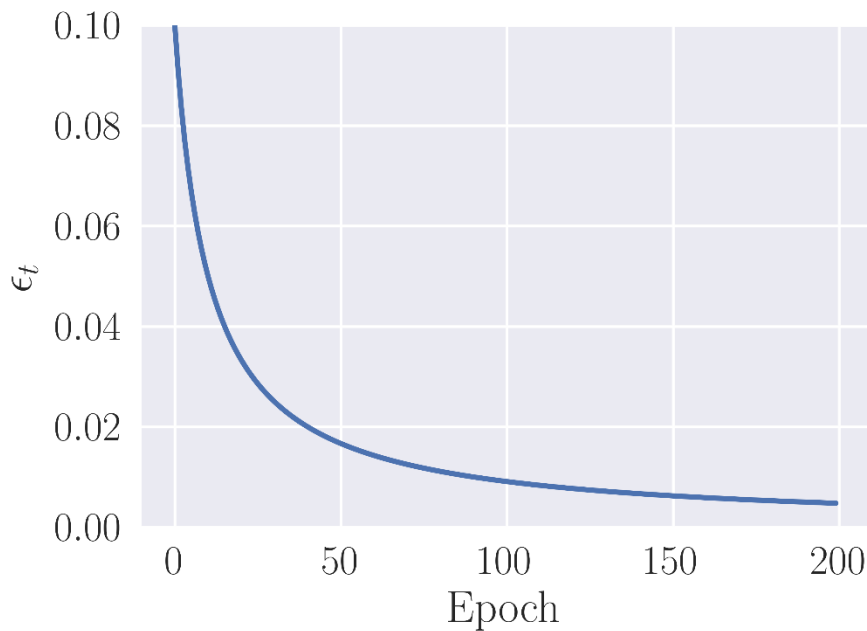
# Stratégie #2 pour choisir l'horaire du taux d'apprentissage $\epsilon$

- Exemple: taux d'apprentissage  $\epsilon$ .
- Valeur initiale  $\epsilon_0$ 
  - Prendre approximativement la plus grande valeur de  $\epsilon_0$  qui ne fait pas diverger  $J(\boldsymbol{\theta})$ .
- Taux de décroissance
  - Dépend du problème.
- Règle de décroissance

$$\epsilon_t = \frac{\epsilon_0}{1 + \beta \cdot t_e} \quad \text{ou} \quad \epsilon_t = \epsilon_0 \cdot \exp\{-\beta \cdot t_e\}$$

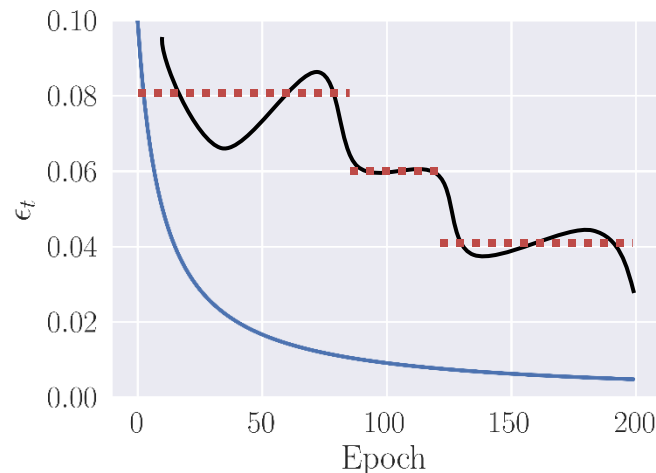
# Exemple d'horaire du taux d'apprentissage $\epsilon$

- $\epsilon_0 = 0.1$ ,  $\beta = 0.1$ ,  $\epsilon_t = \frac{\epsilon_0}{1+\beta \cdot t_e}$



# Stratégie #1 > Stratégie #2

- Avantage de Stratégie #1
  - Relation non-linéaire entre le taux de décroissance de  $J(\theta)$  et celui de  $\epsilon_t$ .
  - Difficile de caractériser cette relation.
  - Stratégie #1 approxime la relation non-linéaire avec une fonction en escalier.



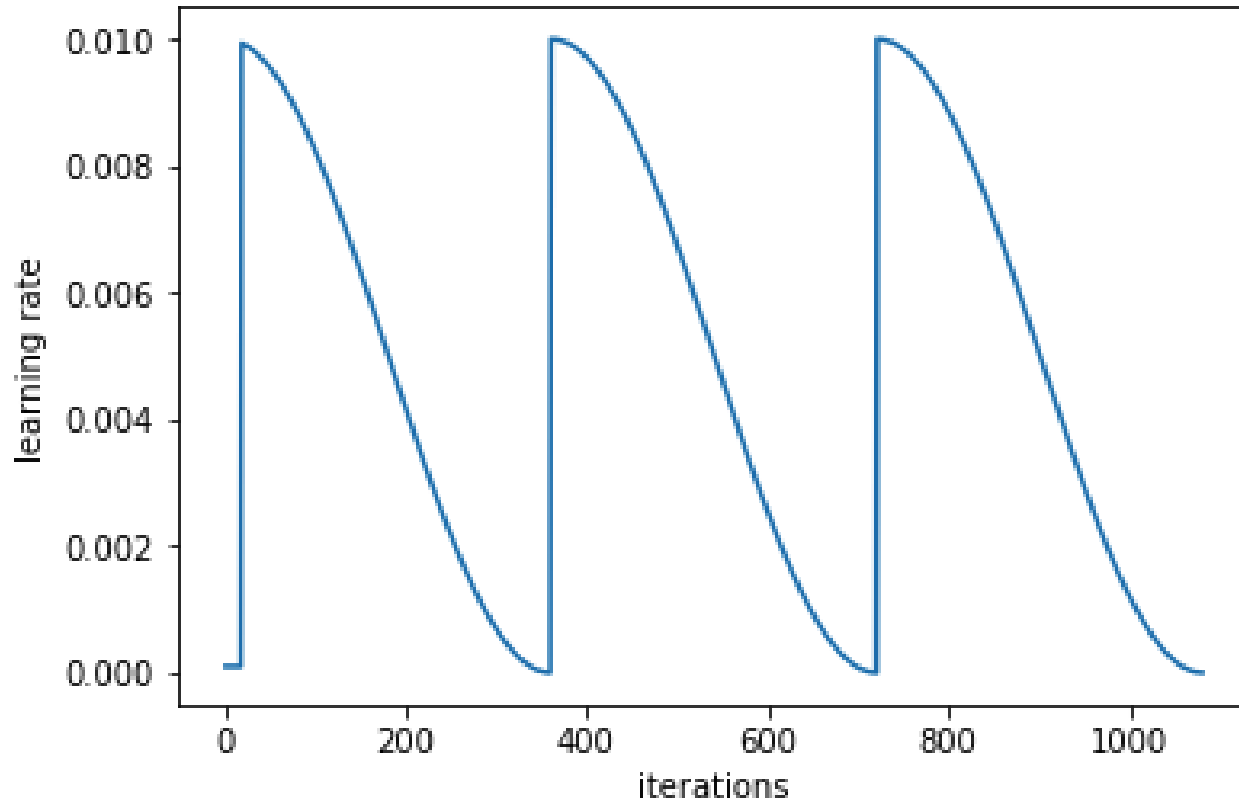
# Avantages et inconvénients des horaires d'entraînement

- Avantages
  - Simple à expliquer.
  - Facile à reproduire.
- Inconvénients
  - Développer son intuition pour comprendre l'effet des hyper-paramètres.
  - Peut sembler arbitraire.
- Somme toute
  - Souvent utilisé en pratique

## 1.2 Taux d'apprentissage cyclique

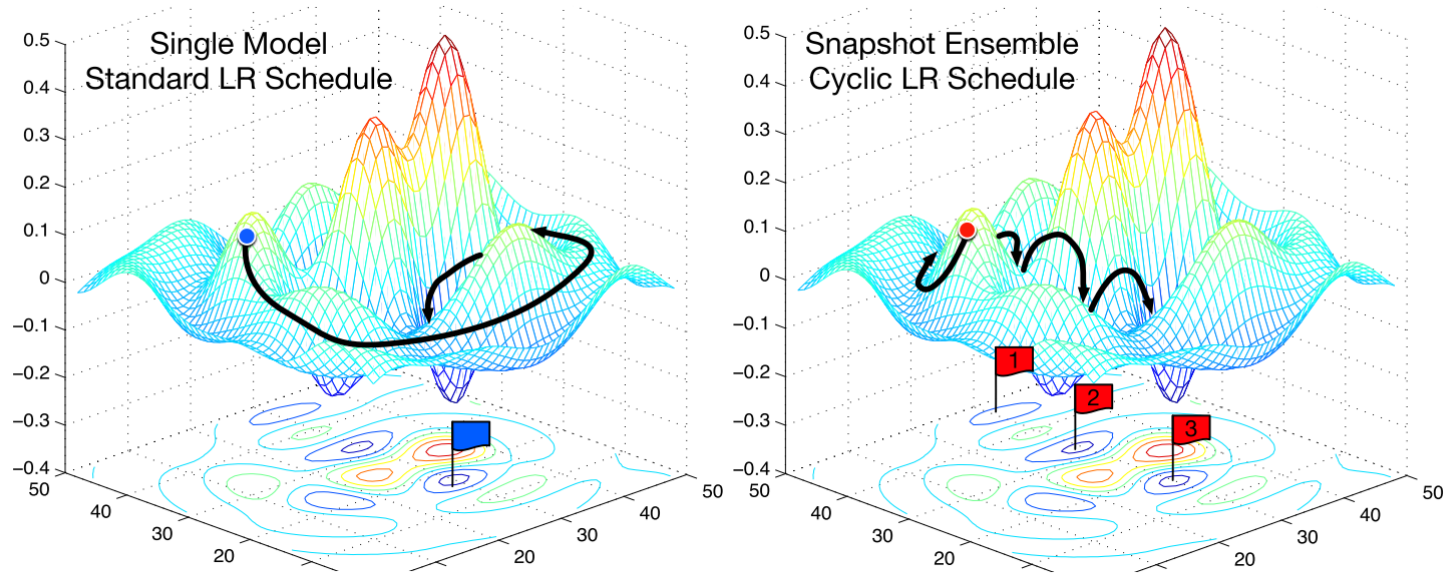
- Méthode d'optimisation en soit
  - Descente de gradient stochastique avec redémarrage
  - Ensemble de modèles conservé pour faire la prédiction finale sur l'ensemble de test

# 1.2 Taux d'apprentissage cyclique



fast.ai

# 1.2 Taux d'apprentissage cyclique



<https://arxiv.org/abs/1704.00109>

# 1.3 Recherche en grille

- Certains hyperparamètres doivent restés fixes pour toute la durée de l'optimisation.
  - Nombre de neurones
  - Fonction d'activation
  - Probabilité de dropout (Semaine 4)
  - Dimension des filtres à convolution (Semaine 5)
- D'autres peuvent être fixés même si pas nécessaire.
  - Weight decay (Semaine 4)
  - Taux d'apprentissage
- Comment trouver leur valeur optimale ?

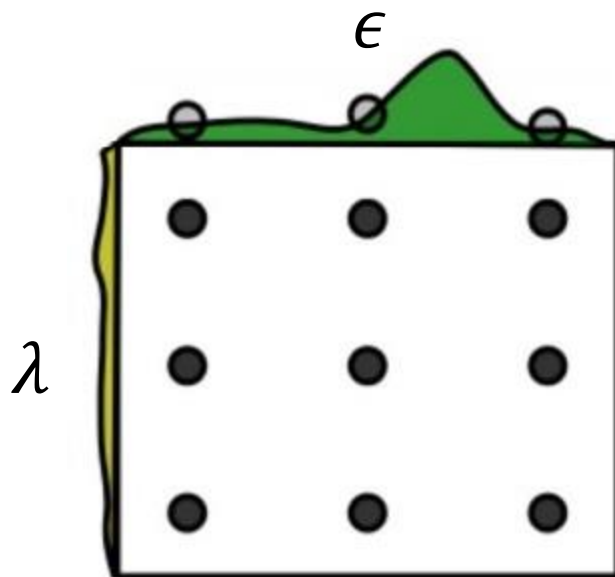


# Recherche en grille

- Principe
  - Déterminer **un ensemble de valeur** à tester pour chaque hyperparamètre.
  - Utiliser une échelle logarithmique.
- Méthode
  - Minimiser  $J(\boldsymbol{\theta})$  pour chaque combinaison du produit cartésien.

# Exemple de recherche en grille

- Hyperparamètres:
  - Taux d'apprentissage:  $\epsilon \in \{0.1, 0.01, 0.001\}$
  - Weight decay  $\lambda \in \{0.001, 0.0005, 0.0001\}$



# Avantages et inconvénients

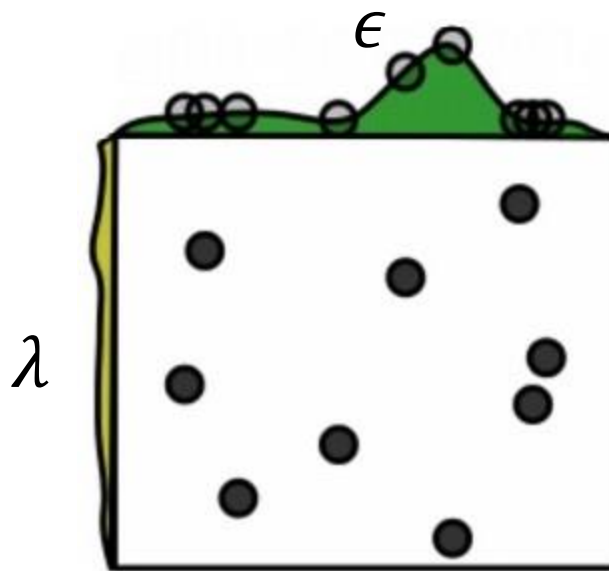
- Avantages
  - Profiter de la connaissance d'un expert.
  - Recherche exhaustive (brute force).
  - Facilement parallélisable.
- Inconvénients
  - Demande computationnelle élevée.
  - Calcul redondant, car un seul hyperparamètre change à la fois.

# 1.4 Recherche aléatoire

- Principe
  - Déterminer une **distribution de probabilité** pour chaque hyperparamètre.
  - Utiliser une échelle logarithmique.
- Méthode
  - Échantillonner les distributions  $T$  fois.
  - Minimiser  $J(\boldsymbol{\theta})$   $T$  fois, une fois pour chaque combinaison.

# Exemple de recherche aléatoire

- Hyperparamètres:
  - Taux d'apprentissage:  $\epsilon = 10^{\gamma_1}$  où  $\gamma_1 \sim U(-1, -3)$
  - Weight decay:  $\lambda = 10^{\gamma_2}$  où  $\gamma_2 \sim U(-3, -4)$



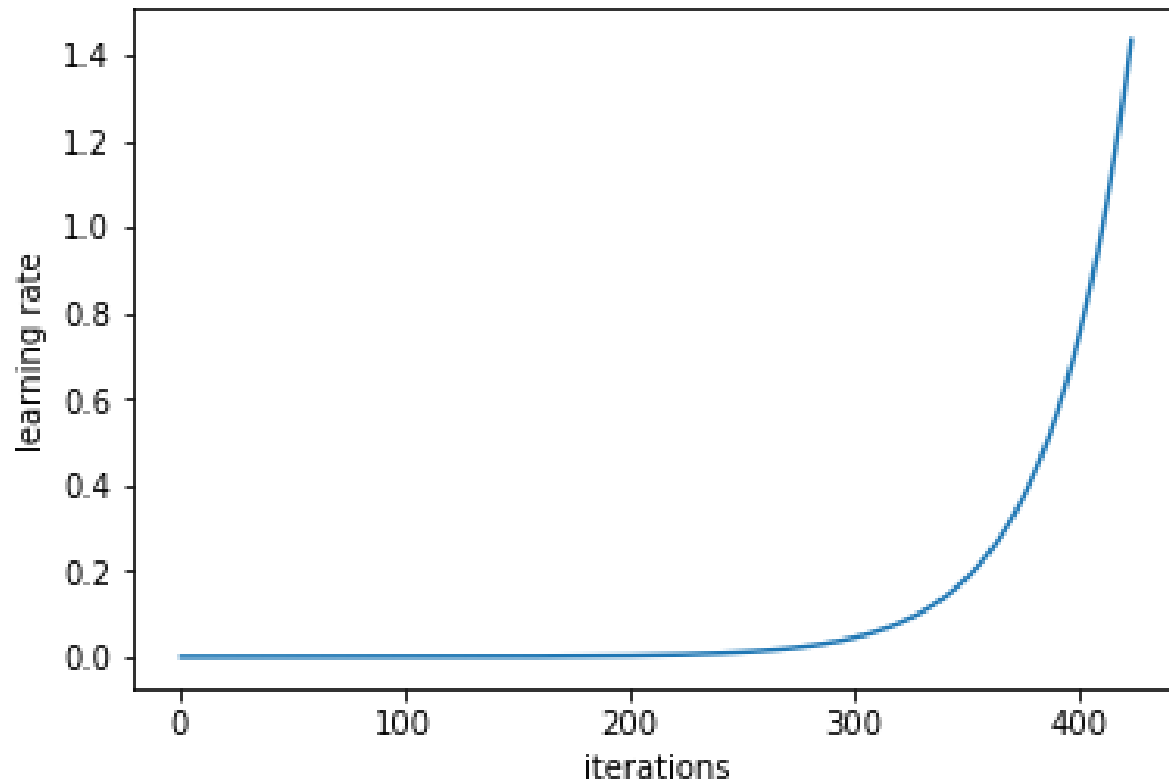
# Avantages et inconvénients

- Avantages
  - Mêmes avantages que recherche en grille.
  - Moins redondant, car hyperparamètre change à chaque fois.
  - Anytime: peut être arrêté n'importe quand.
  - Plus il roule, meilleure est la recherche.
- Inconvénients
  - Demande computationnelle élevée.
  - Difficile à reproduire (random seed).

# 1.5 Learning rate finder

- Principe
  - Trouver un bon taux d'apprentissage pour débiter l'entraînement
- Importance
  - Un taux d'apprentissage trop petit occasionne une descente de gradient longue
  - Un taux d'apprentissage trop élevé peut faire diverger le réseau

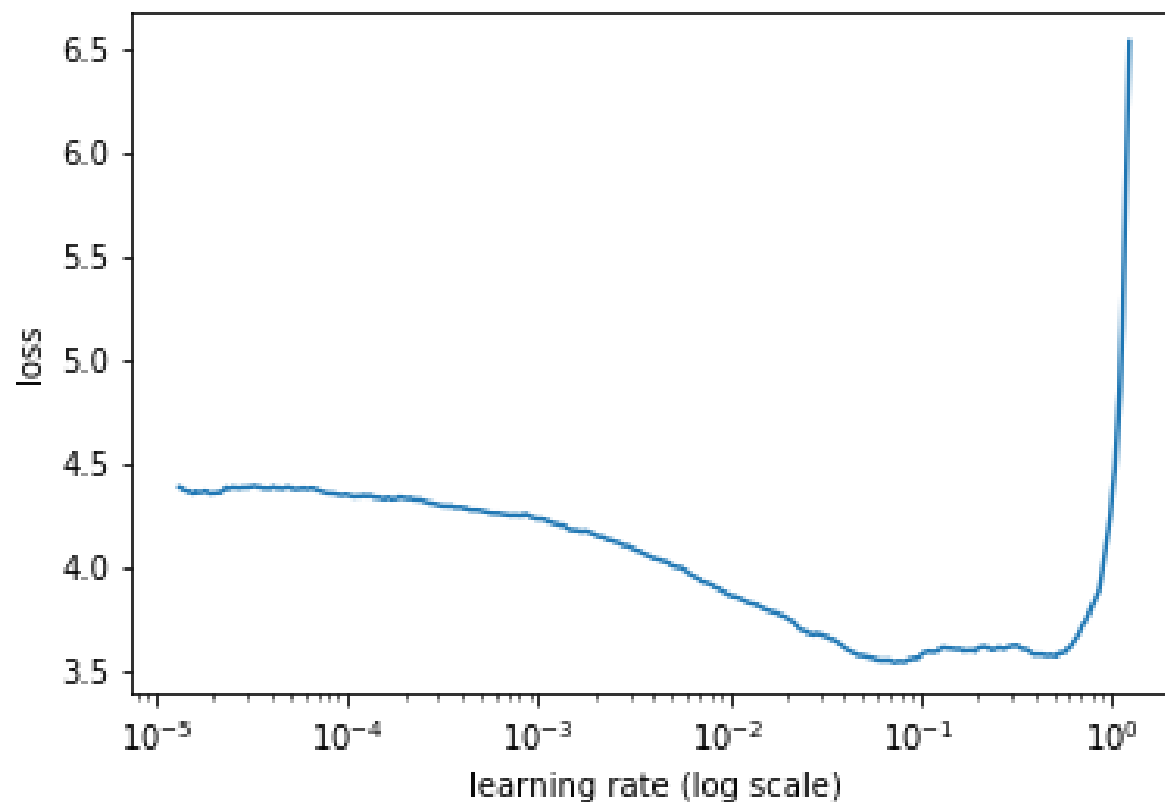
# 1.5 Learning rate finder



fast.ai



# 1.5 Learning rate finder



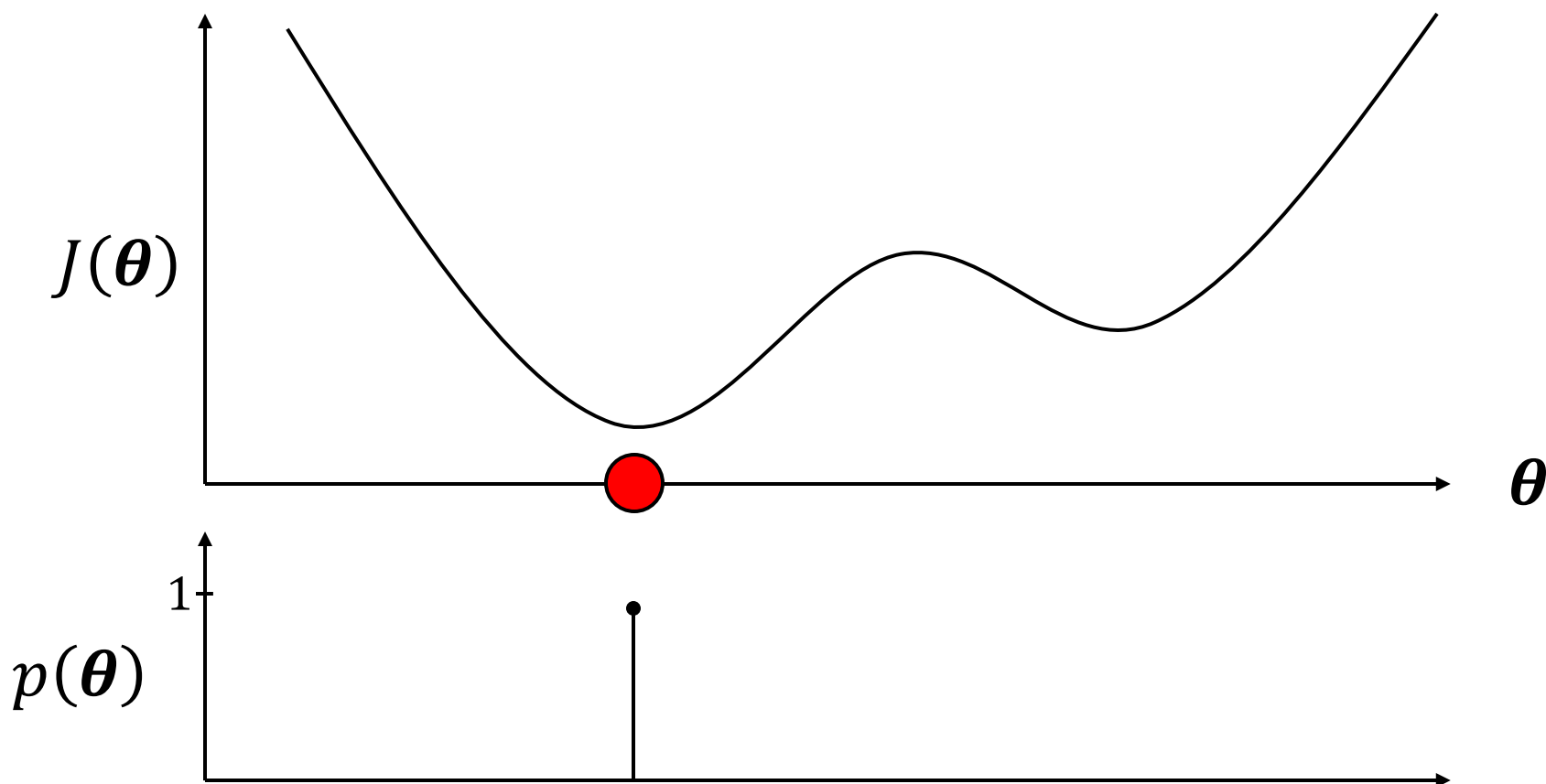
fast.ai

## 2. Initialisation des paramètres

- Principe
  - Choisir la valeur initial de  $\theta$  avant l'optimisation
- Importance
  - Souvent laisser de côté.
  - Un réseau mal initialisé peut être difficile à optimiser.
  - $w = -1,000,000, x = 1$
  - $\text{sigmoid}(w \cdot x) = \frac{1}{1 + \exp(-w \cdot x)} = \frac{1}{1 + \exp(1,000,000)}$

## 2. Initialisation des paramètres

- A un grand impact sur l'optimisation.



## 2. Initialisation des paramètres

Deux façons:

1. Initialisation à partir de rien
  - La seule façon pour beaucoup de problèmes.
2. Initialisation par pré-entraînement
  - Utilisée lorsqu'un réseau a déjà été entraîné sur le même type de problème.

# Initialisation à partir de rien

- Initialisation des biais
  - Généralement initialisés à 0.
- Initialisation des poids
  1. Choisir une distribution de probabilité.
  2. Choisir ses paramètres.
  3. Échantillonner les poids iid.
- Définition:
  - $n_i$ : nombre de neurones en entrée.
  - $n_o$ : nombre de neurones en sortie.

# Initialisation aléatoire des poids

Nom	Uniforme $w \sim U(-a, a)$	Normale $w \sim N(0, s^2)$	Article
Glorot / Xavier	$a = \sqrt{6/(n_i + n_o)}$	$s = \sqrt{2/(n_i + n_o)}$	<a href="#">pdf</a>
Kaiming He	$a = \sqrt{6/n_i}$	$s = \sqrt{2/n_i}$	<a href="#">pdf</a>

- Glorot / Xavier
  - À utiliser avec sigmoid et tanh.
- Kaiming
  - À utiliser avec ReLU
- Dérivation des équations
  - [lien](#)

# Initialisation PyTorch

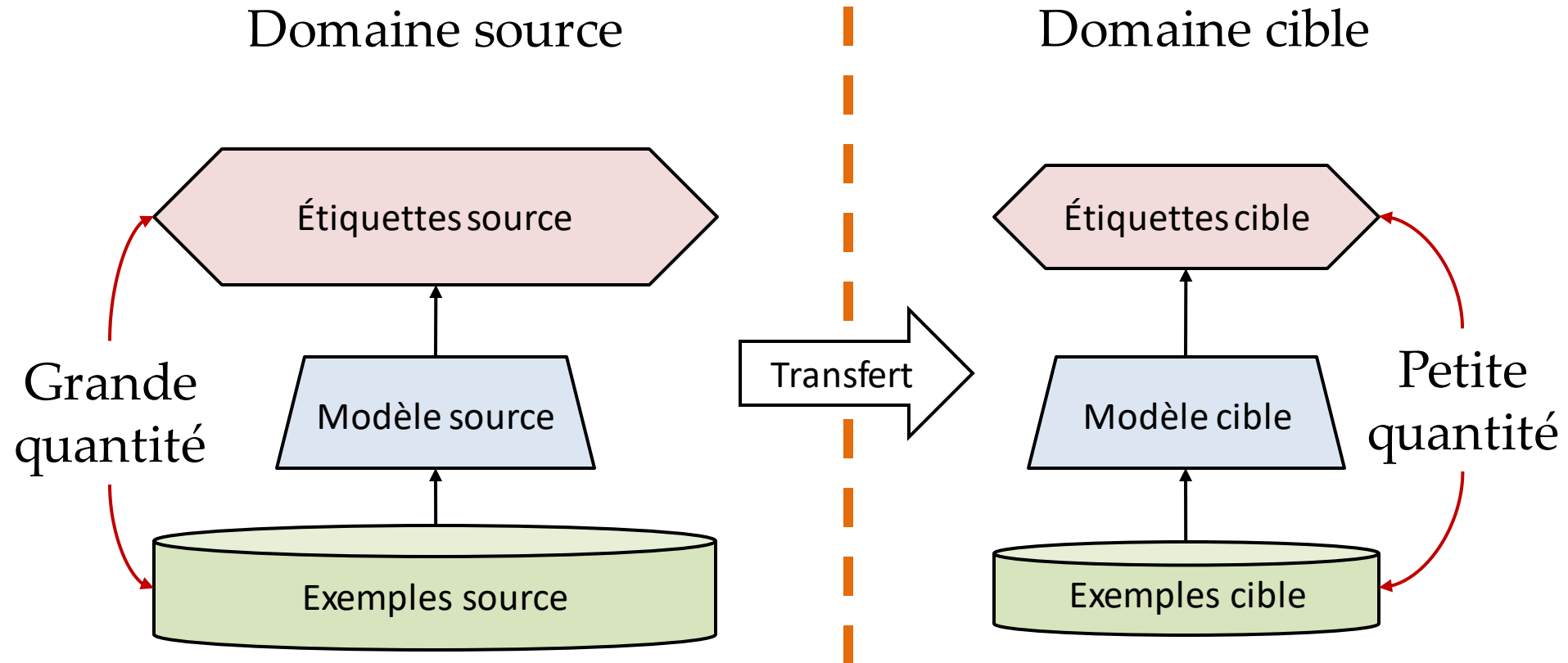
- Par défaut pour linéaire et à convolution ([lien](#)):
  - $w \sim U(-a, a), a = \sqrt{1/n_i}$
  - Pourquoi ?  $\neg\_(\text{ツ})\_/\neg$
- Module [init.py](#)
  - kaiming\_normal, kaiming\_uniform, xavier\_normal, xavier\_uniform

# Initialisation constante des poids

- Principe
  - On peut aussi initialiser à une constante  $k$ .
  - $w = k$
- Inconvénient
  - Poids identique  $\rightarrow$  MAJ identique
  - $y_1 = w_{11}x_1 + w_{12}x_2$
  - $y_2 = w_{21}x_1 + w_{22}x_2$
  - $$-\frac{\partial J}{w_{11}} = \frac{\partial J}{y_1} \frac{\partial y_1}{w_{11}} = \frac{\partial J}{y_1} x_1 \leftrightarrow \frac{\partial J}{w_{21}} = \frac{\partial J}{y_2} \frac{\partial y_2}{w_{21}} = \frac{\partial J}{y_2} x_1$$



# Initialisation par pré-entraînement



# Initialisation par pré-entraînement

- Méthode
  - Initialiser les paramètres du modèle cible avec ceux du modèle source.
  - Entraînement devient *fine-tuning*.
- Fonctionne si  $\text{Dom}(\text{source}) \approx \text{Dom}(\text{cible})$ .

# 3. Normalisation

- Principe
  - Transformer l'input pour mieux conditionner l'optimisation.
- Nous verrons:
  - Normalisation Min-Max
  - Normalisation du *z-score*
  - Normalisation par batch

# Normalisation Min-Max

- Principe
  - Transforme l'input au range  $[0, 1]$ .
- Méthode
  - $\bar{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
- Sensibilité au *outliers*.
  - Utilisé seulement quand  $x$  est borné.
  - Exemple: image  $I \in \{0, 255\}^3 \rightarrow I \in [0, 1]^3$

# Normalisation du *z-score*

- Principe
  - Aussi appelée **normalisation** tout court.
  - Transforme l'input pour que moyenne = 0 et variance = 1.
- Méthode
  - $\bar{x} = \frac{x - x_{\mu}}{\sqrt{x_{\sigma^2} + \delta}}$
  - $x_{\mu}$  et  $x_{\sigma^2}$  calculés avec tous les  $x^{(i)}, i = 1 \dots N$ , d'entraînement.
- Également sensible aux *outliers*.
- Souvent utilisée.

# Normalisation par batch (BN)\*

- Principe ([lien](#))
  - Appliquer *normalisation du z-score* sur les neurones des couches cachées du réseau.
- Méthode
  - $\mathbf{h}^l = \mathcal{F}(\mathbf{h}^{l-1})$
  - $\bar{\mathbf{h}}^l = \alpha \cdot \frac{\mathbf{h}^l - \mathbf{h}^l_{\mu}}{\sqrt{\mathbf{h}^l_{\sigma^2} + \delta}} + \beta$
  - $\mathbf{h}^l_{\mu}$  et  $\mathbf{h}^l_{\sigma^2}$  calculés seulement sur les  $\mathbf{h}^l$  de la batch.
  - $\alpha$  et  $\beta$  paramètres additionnels à apprendre.

# Normalisation par batch

- Pour le mode test
  - Moyenne mobile:  $\mathbf{m} \leftarrow \rho \mathbf{m} + (1 - \rho) \mathbf{h}^l_{\mu}$
  - Variance mobile:  $\mathbf{s} \leftarrow \rho \mathbf{s} + (1 - \rho) \mathbf{h}^l_{\sigma^2}$
  - $\bar{\mathbf{h}}^l = \alpha \cdot \frac{\mathbf{h}^l - \mathbf{m}}{\sqrt{\mathbf{s} + \delta}} + \beta$
- Initialisation
  - $\alpha = 1, \beta = 0, \mathbf{m} = 0, \mathbf{s} = 1$

# Avantages de la normalisation par batch

- Avantage #1
  - Apprentissage end-to-end avec backprop.
- Avantage #2
  - Diminue l'impact d'un taux d'apprentissage initial trop grand.
  - $\epsilon_0 = 0.1$  fonctionne souvent très bien.
- Avantage #3
  - Réseaux profonds facile à entraîner.



# Avantages de la normalisation par batch

- Avantage #4
  - Réseau converge plus rapidement.
- Avantage #5
  - Diminue le nombre de neurones morts / désactivés de la ReLU.

# Avantages de la normalisation par batch

- Tout ces avantages sont en grande partie causés par un lissage de la fonction de perte
- La normalisation par batch devrait **toujours être utilisée.**
  - Placer avant activation ou après fc / conv.

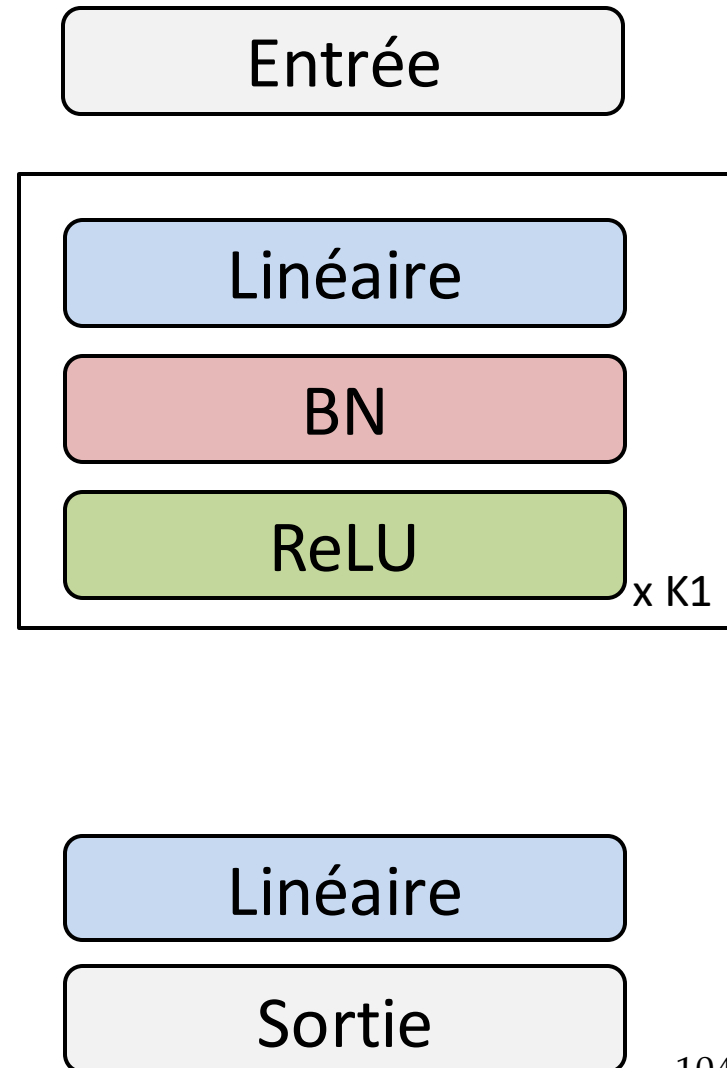
Diagnostic

# Diagnostic

- Quelques recettes à suivre pour partir du bon pied
  1. Structure neuronale de base.
  2. Information à enregistrer.
  3. Méthode rouleau compresseur.
  4. Analyse

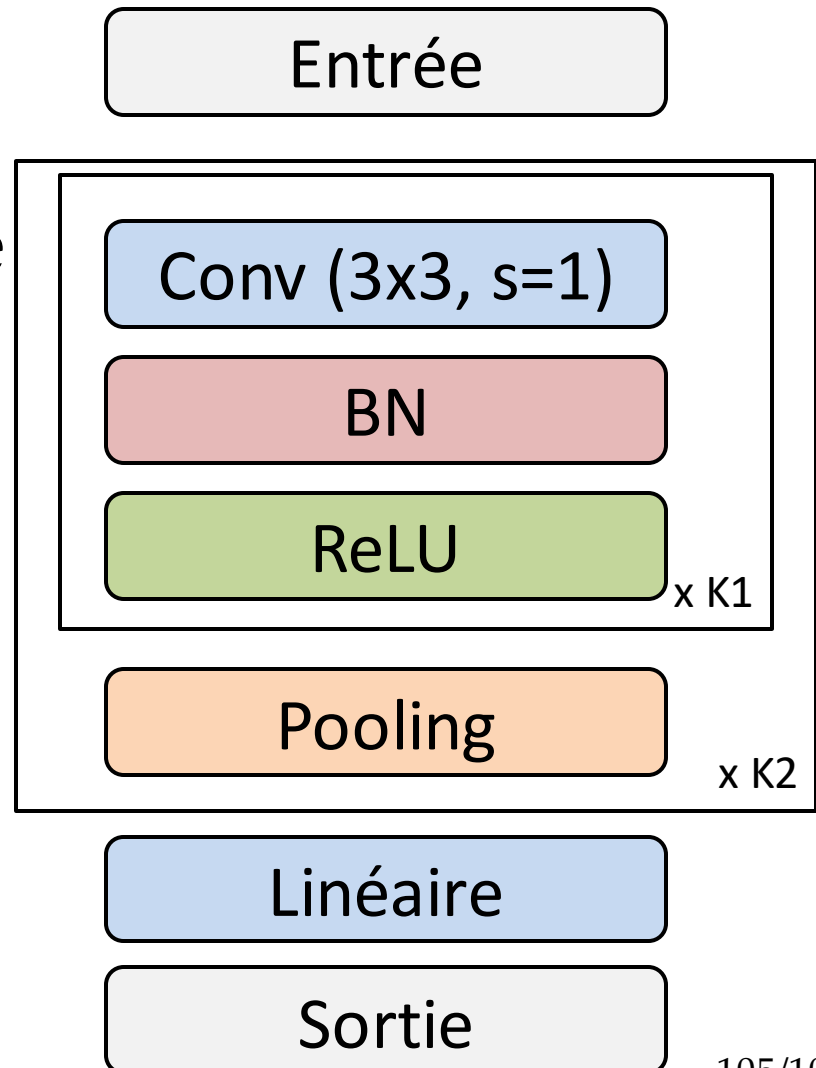
# 1. Structure neuronale de base

- Fully Connected
  - Initialisation  
Kaiming He  
uniforme
  - bias = False
  - BN:  $\alpha = 1, \beta = 0,$   
 $m = 0, s = 1$

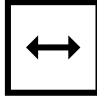


# Structure neuronale de base

- Convolution
  - Initialisation  
Kaiming He normale
  - bias = False
  - BN:  $\alpha = 1$ ,  $\beta = 0$ ,  
 $m = 0$ ,  $s = 1$
  - Max Pooling
  - Global Average Pooling



## 2. Information à enregistrer

- Bien diagnostiquer  accès à toute l'information
  - But: s'assurer que tout se passe comme voulu.
- Voici quelques exemples d'information à enregistrer.

# Information de performance

- Valeur de la fonction de coût
  - Peut voir s'il y a des nan, inf, si ça augmente au lieu de diminuer.
- Valeur de la métrique de performance
  - Donne une meilleure idée de la performance que la fonction de coût (coût=1.2, err=5%).



# Information sur le réseau

- Structure du réseau
  - Peut voir si on entraîne le bon réseau.
- Afficher le nombre de paramètres
  - `np.sum([p.numel() for p in network.parameters()])`
- Afficher les prédictions d'une seule batch en début d'époque
  - Peut voir l'évolution des prédictions.

# Information sur les données

- Nombre d'exemples d'entraînement / de validation / de test.
- Ratio des classes
  - Détecter un déséquilibre de classes.
- Afficher les exemples d'une seule batch en début d'époque
  - Seulement si c'est informatif (e.g. images)

# Information sur l'optimisation

- La norme / moyenne / variance des MAJ  $d$
- La norme / moyenne / variance / % = 0 des neurones cachées
  - Peut donner une idée du nombre d'unités mortes.
- Afficher le temps de calcul d'une batch
  - Estimer le temps total de l'apprentissage.
- Valeur des hyperparamètres

### 3. Méthode rouleau compresseur

- Pour mieux se familiariser avec un nouveau problème d'apprentissage.
- Procéder en 3 étapes:
  1. Sous-ensemble des données + sous-ensemble des classes.
  2. Toutes les données + sous-ensemble des classes.
  3. Toutes les données + toutes les classes.

# Étape #1: sous-ensemble données + sous-ensemble classes

- But
  - Démontrer que le réseau peut **sur-apprendre**.
  - Si pas bon en train → pas bon en val.
- Astuces
  - Pas de régularisation.
  - Pas d'augmentation de donnée.
- Exemple
  - Classification binaire avec 10 exemples par classe.

# Étape #2: toutes les données + sous-ensemble classes

- But
  - Démontrer que le réseau peut **généraliser**.
- Astuces
  - Ajouter un peu de régularisation.
  - Ajouter un peu d'augmentation de donnée.
- Exemple
  - Classification binaire avec 100,000 exemples par classe.

# Étape #3: toutes les données + toutes les classes

- But
  - Débuter l'entraînement normal.
- Astuces
  - Petit réseau → gros réseau
  - Horaire d'entraînement
  - Augmentation de donnée maximale
- Exemple
  - Classification 100 classes avec 100,000 exemples par classe.

# 4. Analyse

- But
  - Obtenir une retrospective après l'apprentissage.
- Permet de
  - Détecter des étiquettes fausses
  - Détecter un mauvais traitement de données.
  - Donner une intuition sur la difficulté de la tâche.



# Analyse d'erreur

- Exemple
  - Aléatoirement, regarder quelques exemples de predictions correctes et erronées.
  - L'exemple ayant la meilleure performance.
  - L'exemple ayant la pire performance.
  - L'exemple le plus incertain (prob 0.5 classification binaire)

# Analyse des batches

- S'assurer que:
  - Tous les exemples sont différents
  - Les étiquettes sont correctes
- Afficher l'augmentation de données
  - Regarder si elle respecte l'invariance de classe.

