UNIVERSITÉ LAVAL

GLO-4030/7030
APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

Optimisation pour l'apprentissage profond

Hiver 2020

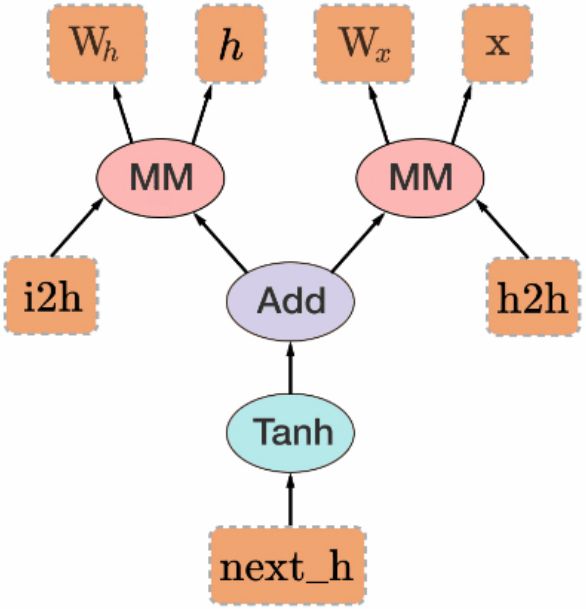
Philippe Giguère

Plan

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

Retour sur la semaine dernière

$f(\boldsymbol{x};\boldsymbol{\theta})$



pytorch.org

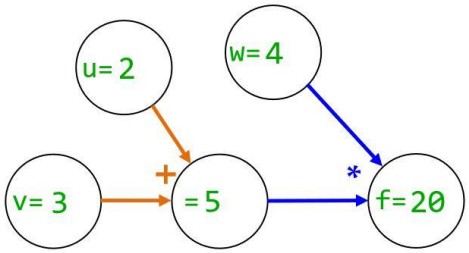
$L(f(\boldsymbol{x};\boldsymbol{\theta}), y)$

Entropie croisée:

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

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

Backprop



$f=(u+v)w$

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 $\hat{p}_{data}$:

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

- **Vrai risque $J^*(\boldsymbol{\theta})$**
  - Fonction de coût L + vraie distribution de données p_{data} :

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

Notion d'apprentissage

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

Algorithmes d’optimisation

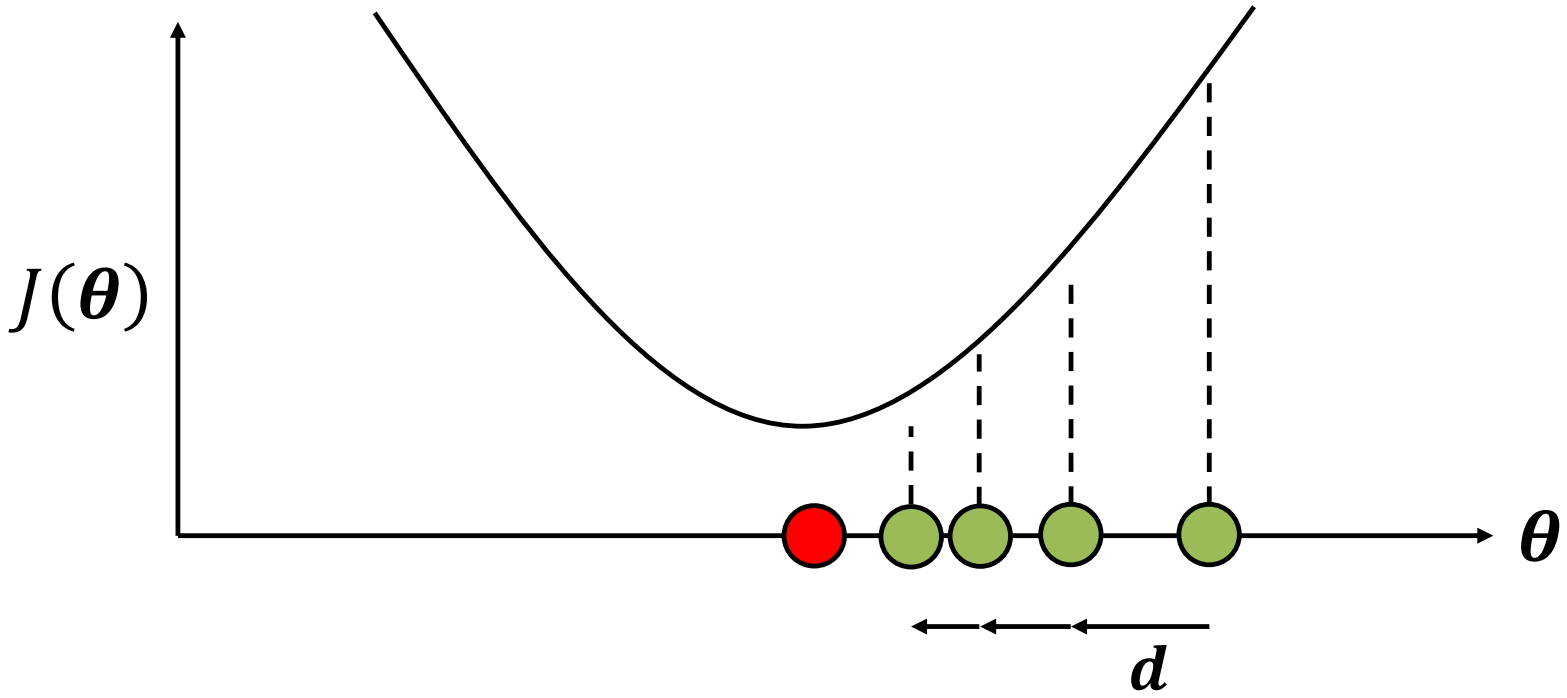
Descente du gradient

- Concept
  - Déterminer un vecteur directionnel $\boldsymbol{d}$ basé sur le gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.
  - Mettre à jour les paramètres:

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

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

Notion de gradient

- 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

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

$$= \nabla_{\boldsymbol{\theta}} \, \frac{1}{N} \sum_{i=1}^{N} L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right), y^{(i)}\right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \, L\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right), y^{(i)}\right)$$

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

- Batch signifie ici TOUT le jeu de données
  - pas une *minibatch*
- Vecteur de correction $\boldsymbol{d}$

$$\boldsymbol{d} = \epsilon_t \cdot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

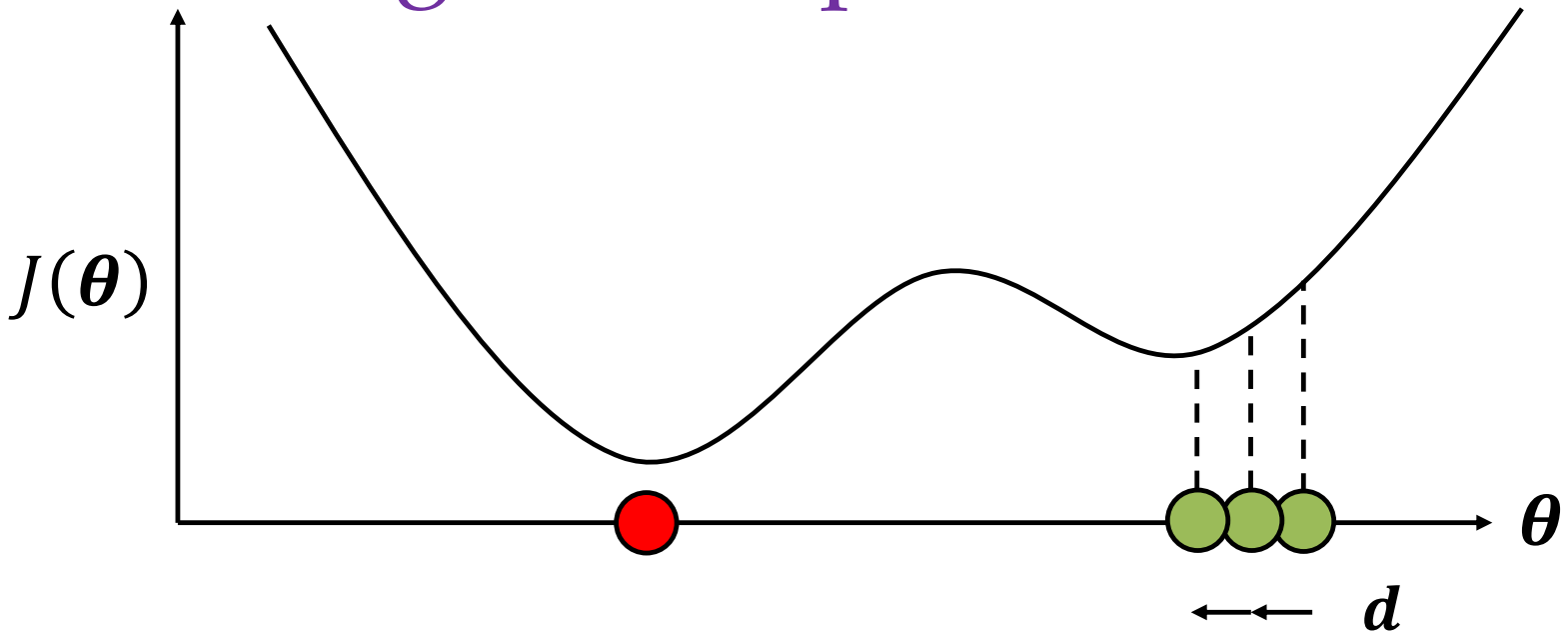
$$= \epsilon_t \cdot \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$$

ϵ_t = **taux d'apprentissage** à l'itération t.

- MAJ:

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

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 $(\boldsymbol{x}^{(i)}, y^{(i)})$ donne **une seule** mise-à-jour.
  - Pas de MAJ entre $(\boldsymbol{x}^{(i)}, y^{(i)})$ et $(\boldsymbol{x}^{(i+j)}, y^{(i+j)})$
  - Erreur semblable → gradient semblable → redondance

2. Descente du gradient stochastique

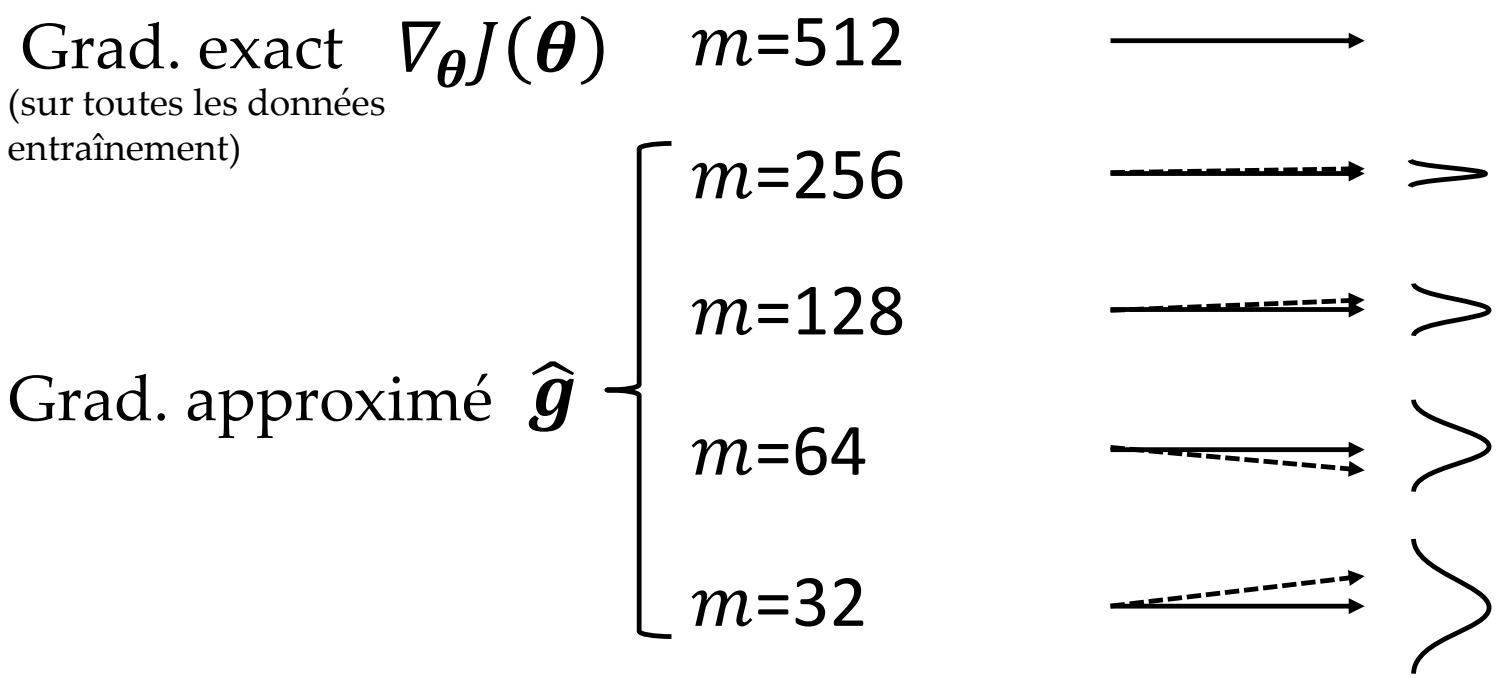
- SGD
  - *stochastic gradient descent*
  - Une **minibatch** de taille m → une MAJ
- Vecteur de correction

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

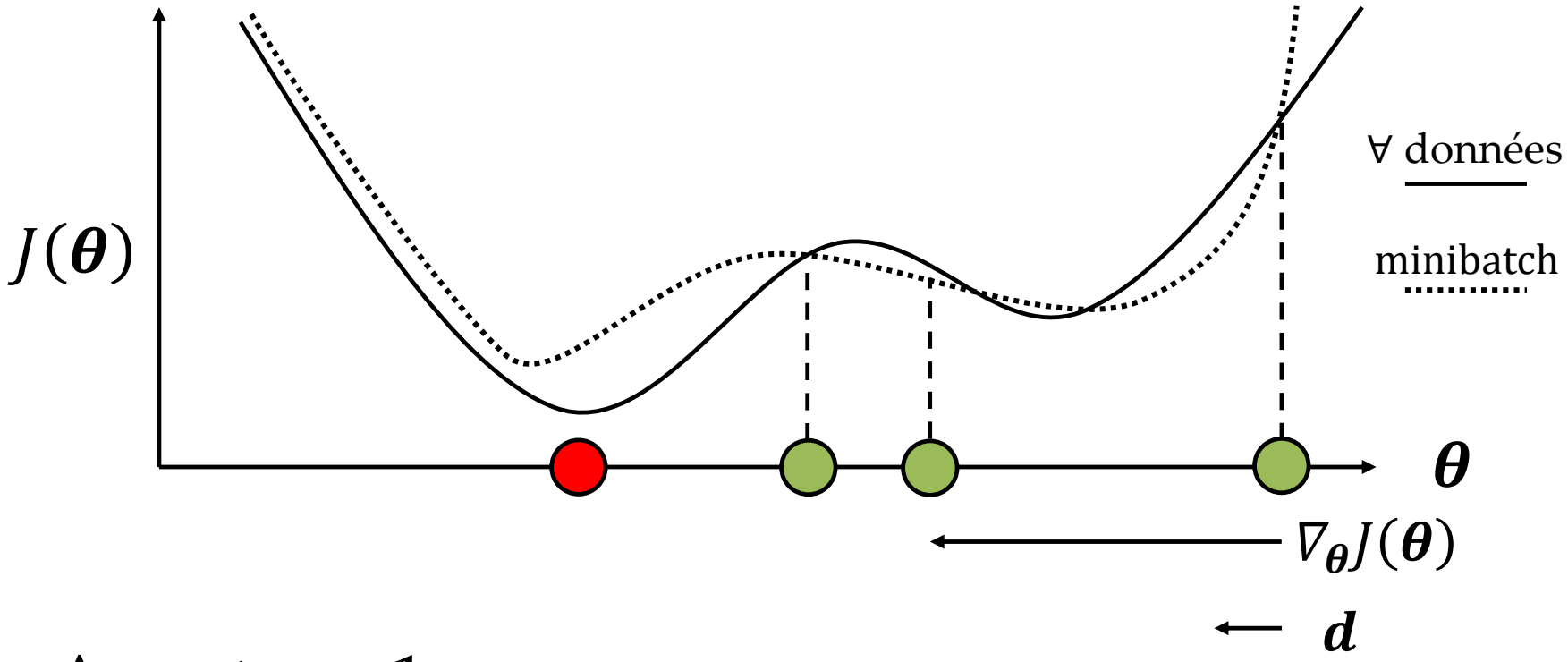
- Algorithme
  1. Échantillonnage : $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}}$
  3. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{d}$

Descente du gradient stochastique

- L'incertitude sur le gradient dépend de la taille m de la minibatch : $\sigma = O(1/\sqrt{m})$.
- Direction semblable avec moins de données.



Avantages de la descente du gradient stochastique



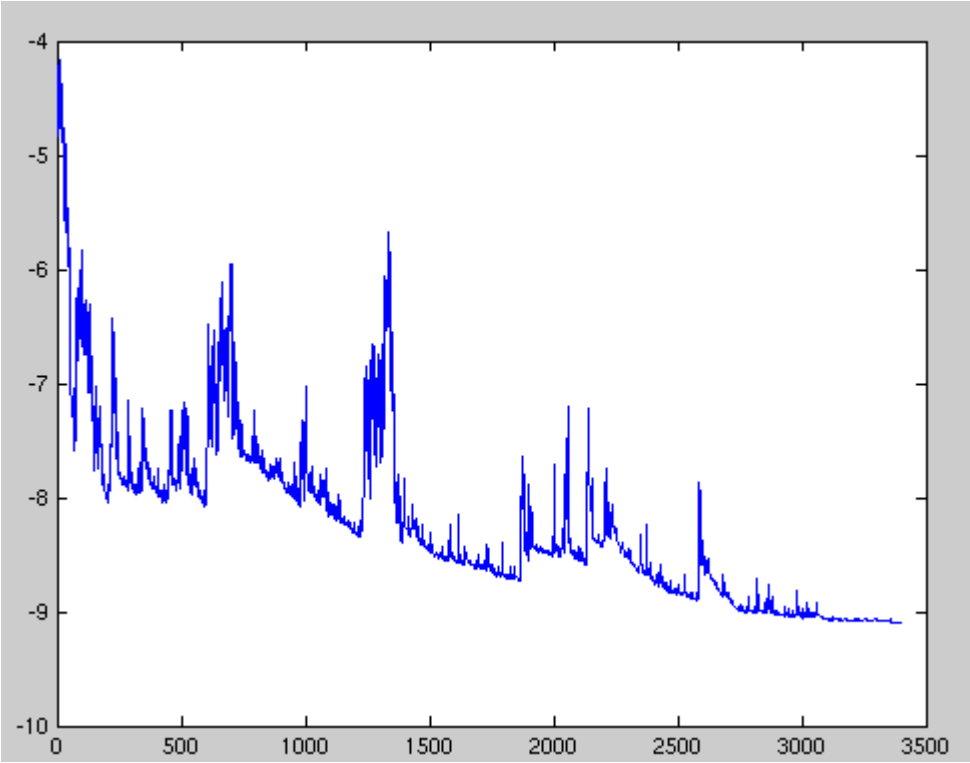
- Avantage 1
  - $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \hat{\boldsymbol{g}}$ = version bruitée du gradient.
  - Bruit aide à sortir des min / max locaux.

Avantages de la descente du gradient stochastique

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

Inconvénient #1 de la descente du gradient stochastique

- Variance → fluctuation



Wikipédia

Inconvénient #2 de la descente du gradient stochastique

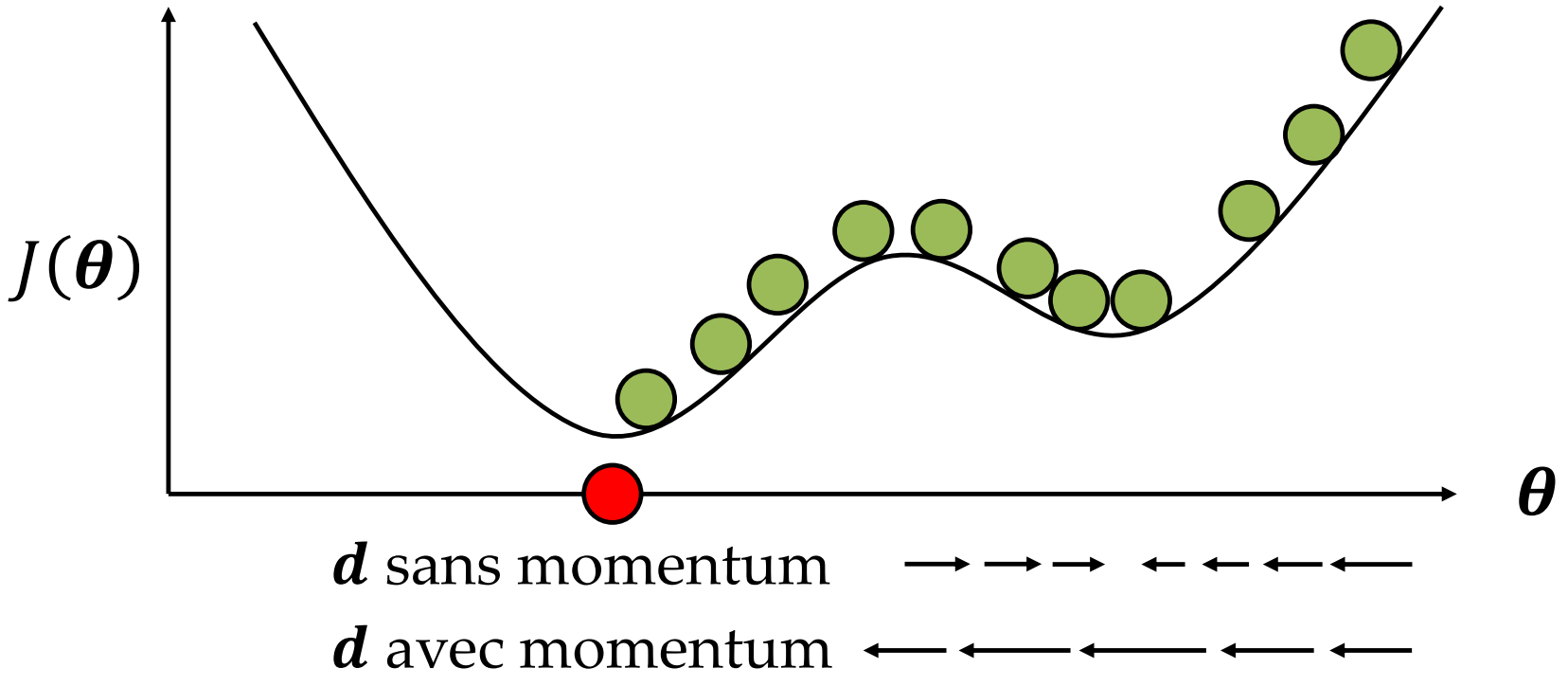
- Taux d'apprentissage ϵ a une grande influence
  - À suivre… (section stratégies d'optimisation)

Note pratique

- Sélection au hasard sur un grand jeu de données (millions) peut être long
- Simplification
  - Une seule permutation aléatoire initiale (*shuffle*) du jeu de données
  - Prends ensuite les données en round-robin
- Fonctionne aussi bien en pratique qu'une vraie pige au hasard

3. SGD + Momentum

- Concept
  - Ajouter à SGD une dynamique Newtonienne.

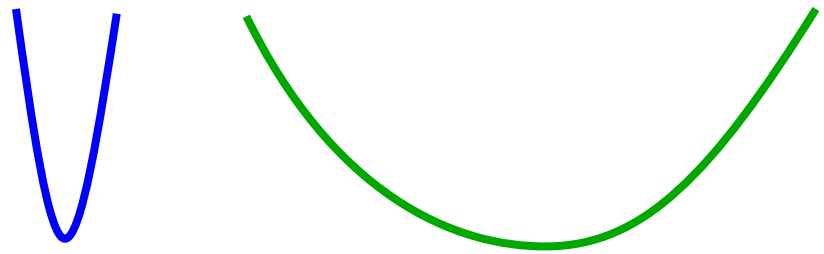


SGD + Momentum

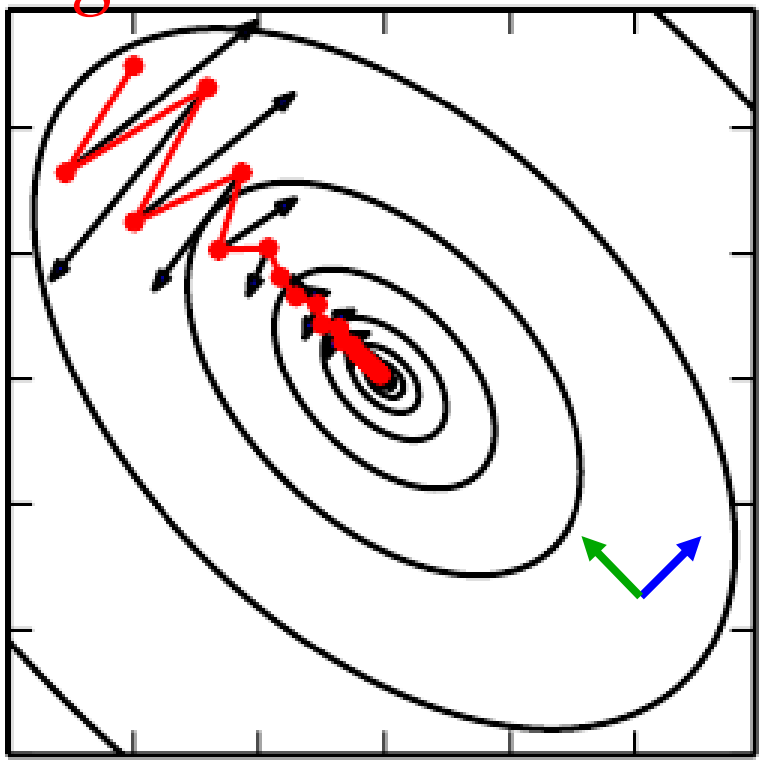
- Velocité $\boldsymbol{v} \in \mathbb{R}^D$
  - Estimation des directions $\boldsymbol{d}$ précédentes.
- Algorithme
  1. Échantillonnage : $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}}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

Avantage de SGD + Momentum

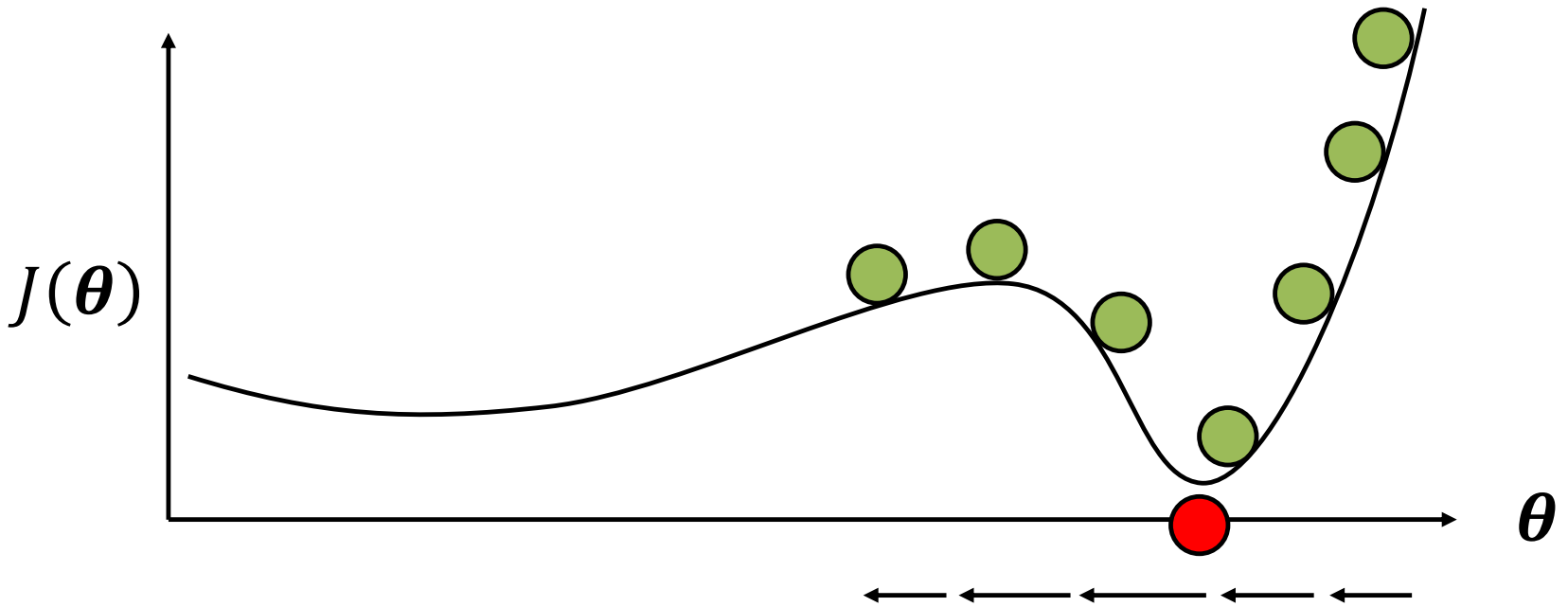
- Le momentum aide lorsqu'il y a des ravins.
- Les MAJ de $\boldsymbol{\theta}$ tendent à s'aligner.
- Atténue les oscillations causées par les grandes courbures



noir = sans momentum
rouge = avec momentum



Inconvénient de SGD + Momentum



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

Note pratique

- Si le réseau est initialisé au hasard
  - peut avoir de très gros gradients en début d'entraînement
  - judicieux d'utiliser un momentum α plus petit (0.5) en début d'optimisation, au lieu des valeurs usuelles $\alpha = [0.9\ 0.99]$
  - augmente graduellement α par la suite

4. SGD + Momentum accéléré de Nesterov

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

SGD + Momentum de Nesterov

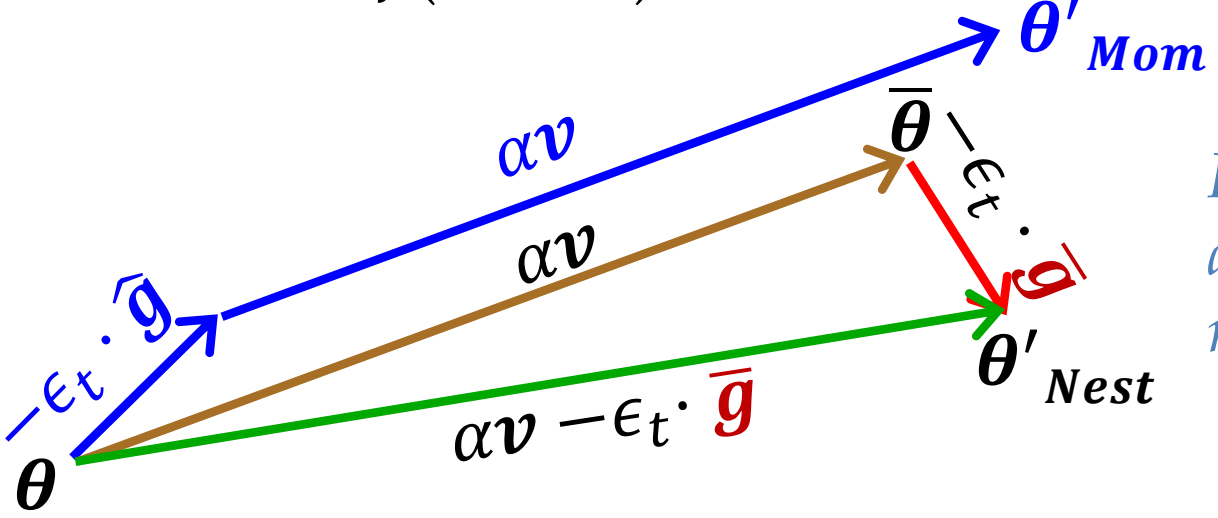
- Algorithme
  1. Échantillonnage : $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Point intérimaire: $\overline{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  3. Gradient: $\overline{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\overline{\boldsymbol{\theta}}}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

SGD + Momentum de Nesterov

- **Momentum standard**
  1. Gradient (petit bleu)
  2. Jump direction des gradients accumulés (grand bleu)

Rappel : $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon_t \cdot \hat{\boldsymbol{g}}$
MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

- **Nesterov**
  1. Jump direction des gradients accumulés (brun)
  2. Gradient (rouge) au point intérimaire $\overline{\boldsymbol{\theta}}$
  3. MAJ (en vert)

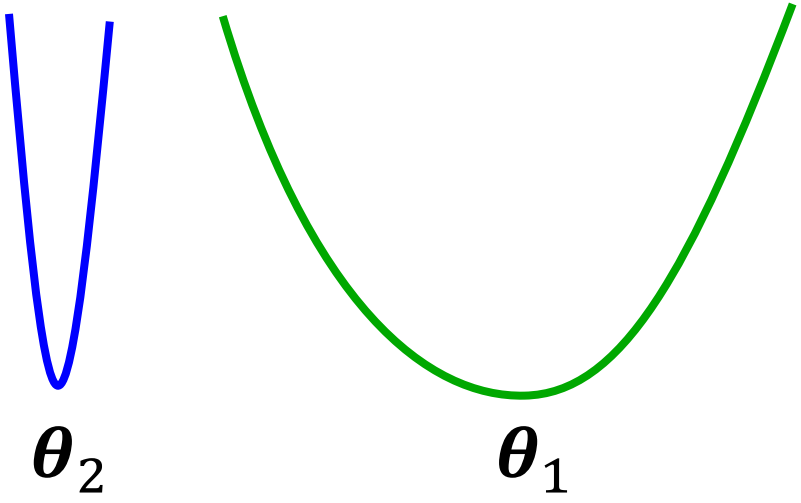


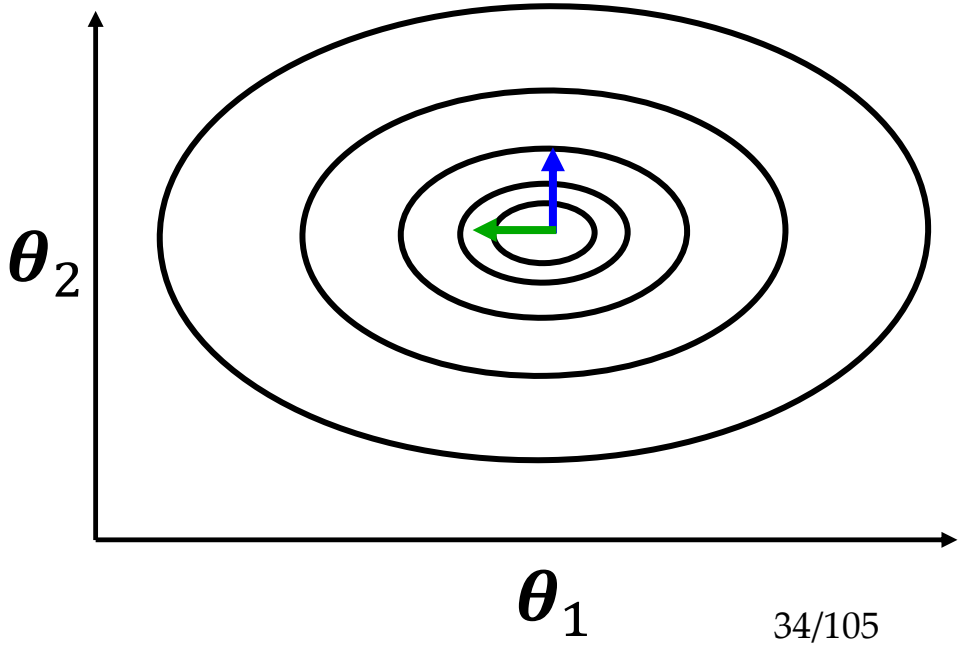
Intuition : fait le gros ajustement $\alpha \boldsymbol{v}$, puis on raffine avec le gradient

Adapté de : G. Hinton lecture 6c

Méthodes par taux d'apprentissage adaptatif

- Optimisation est souvent très sensible dans une direction, et insensible dans d'autres
- Adapter la MAJ de chaque paramètre $\boldsymbol{\theta}_d$ **individuellement** pour uniformiser le taux de changement.





Adagrad

- Principe
  - $\boldsymbol{\theta}_k$ historique petite MAJ → augmenter la MAJ à k.
  - $\boldsymbol{\theta}_k$ historique grande MAJ → 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 $\boldsymbol{r} \in \mathbb{R}^D$
  - Accumule le carré des gradients.
- Petite constante $\delta \in \mathbb{R}$ pour stabilité numérique.

Adagrad

- Algorithme
  1. Échantillonnage: $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  3. Accumulation: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  4. Mise à l'échelle: $\boldsymbol{d} \leftarrow \frac{\epsilon_t}{\delta + \sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$
  5. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{d}$

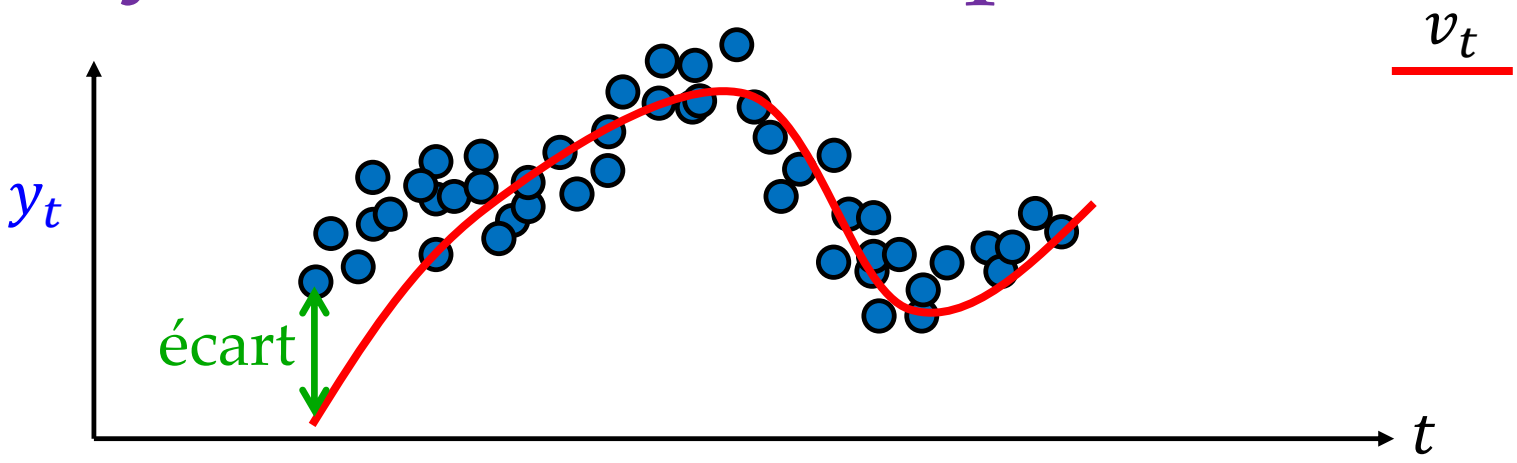
Inconvénient de Adagrad

- Croissance excessive de $\boldsymbol{r}$
  - Accumulation de valeurs positives $\hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$.
  - Les taux d'apprentissage mis à l'échelle $\frac{\epsilon_t}{\delta + \sqrt{\boldsymbol{r}}} \rightarrow \mathbf{0}$ rapidement.
- Particulièrement sensible aux grands gradients de départ pour réseaux initialisés au hasard

6. RMSprop

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

Moyenne mobile exponentielle



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

Problème d'initialisation $v_0 = 0$

$$v_1 = 0.99 \cdot v_0 + 0.01 \cdot y_1$$
$$= 0.01 \cdot y_1$$

valeurs très loin des y_t au début

$$v_2 = 0.99 \cdot v_1 + 0.01 \cdot y_2$$
$$= 0.0099 \cdot y_1 + 0.01 \cdot y_2$$

(Corrigé avec ADAM)

RMSprop

- Algorithme
  1. Échantillonnage: $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  3. Accumulation: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  4. Mise à l'échelle: $\boldsymbol{d} \leftarrow \frac{\epsilon_t}{\delta + \sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$
  5. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{d}$

7. RMSprop + Momentum

- Méthode:
  - Comme pour SGD, on peut ajouter du momentum aux direction calculées.
- Vélocité $\boldsymbol{v} \in \mathbb{R}^D$
  - sur les gradients précédents, incluant la **mise à l'échelle**.
- Facteur d'oubli $\alpha \in [0, 1)$
  - $\alpha \to 0 \equiv$ RMSprop standard

RMSprop + Momentum

- Algorithme
  1. Échantillonnage: $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  3. Accumulation: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  4. Vélocité: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon_t}{\sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$
  5. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

8. RMSprop + Nesterov

- Méthode:
  - Comme pour SGD + Nesterov, on peut utiliser le momentum accéléré de Nesterov.

RMSprop + Nesterov

- Algorithme
  1. Échantillonnage: $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Point intérimaire: $\bar{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
  3. Gradient: $\bar{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\bar{\boldsymbol{\theta}}} L\left(f\left(\boldsymbol{x}^{(i)}; \bar{\boldsymbol{\theta}}\right), y^{(i)}\right)$
  4. Accumulation: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \bar{\boldsymbol{g}} \odot \bar{\boldsymbol{g}}$
  5. Vélocité: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon_t}{\sqrt{\boldsymbol{r}}} \odot \bar{\boldsymbol{g}}$
  6. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

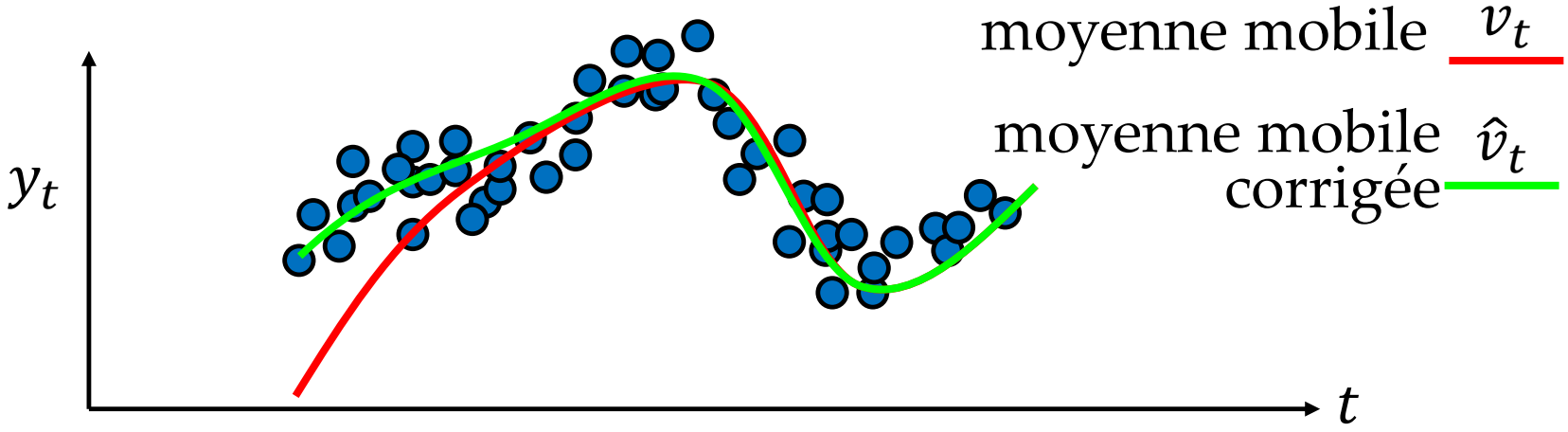
Inconvénients de RMSprop

- Inconvénient 1
  - $\boldsymbol{r}$ est une estimation du second moment (variance décentrée) du gradient $\widehat{\boldsymbol{g}}$.
  - 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 des 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 du biais de départ



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

$$v_0 = 0$$

$$v_1 = 0.99 \cdot v_0 + 0.01 \cdot y_1$$
$$= 0.01 \cdot y_1$$

$$v_2 = 0.99 \cdot v_1 + 0.01 \cdot y_2$$
$$= 0.0099 \cdot y_1 + 0.01 \cdot y_2$$

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

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

Vidéo

Ludovic Trottier

Adam

- Algorithme
  1. Échantillonnage: $(\boldsymbol{x}^{(i)}, y^{(i)}) \sim \hat{p}_{data}, 1 \leq i \leq m.$
  2. Gradient: $\hat{\boldsymbol{g}} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), y^{(i)}\right)$
  3. 1er moment: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\hat{\boldsymbol{g}}$
  4. 2e moment: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  5. Correction biais 1er moment : $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - (\rho_1)^t}$
  6. Correction biais 2e moment : $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - (\rho_2)^t}$
  7. Direction: $\boldsymbol{d} \leftarrow \epsilon_t \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$
  8. MAJ: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{d}$

Tableau comparatif

Approche	Hyper-paramètre	Utilisation mémoire
Batch	ϵ_t	D
SGD	ϵ_t, m	D
SGD + Momentum	ϵ_t, m, α	D*2
SGD + Nesterov	ϵ_t, m, α	D*2
Adagrad	ϵ_t, m, δ	D*2
RMSProp	ϵ_t, m, ρ	D*2
RMSProp + Momentum	$\epsilon_t, m, \rho, \alpha$	D*3
RMSProp + Nesterov	$\epsilon_t, m, \rho, \alpha$	D*3
Adam	$\epsilon_t, m, \rho_1, \rho_2, \delta$	D*3

D: nombre de paramètres pour θ

PyTorch documentation sur **`torch.optim`**

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. Recherche en grille
  3. Recherche aléatoire

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 ϵ_t.**
  - *Plus un art qu'une science.*

Exemple d'horaire d'entraînement

- Nombre d'époque maximum: 200
- Taille de la minibatch: $m = 32$
- 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 → changer l'hyper-paramètre.
  - Produira une courbe en forme d'**escalier.**
- Méthode
  1. Définir la valeur initiale de l'hyperparamètre
  2. Définir son 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 ϵ

- Exemple: taux d'apprentissage ϵ.
- Valeur initiale ϵ_0
  - Trop large: grandes oscillations → $J(\boldsymbol{\theta})$ augmente → divergence (NaNs, Inf)
  - Trop petit: convergence locale → $J(\boldsymbol{\theta})$ grande valeur → piètre solution $\boldsymbol{\theta}$
  - Stratégie pour ϵ_0:
    - Prendre approximativement la plus grande valeur de ϵ_0 qui ne fait pas diverger $J(\boldsymbol{\theta})$.

Stratégie #1 pour choisir l'horaire du taux d'apprentissage ϵ

- Taux de décroissance
  - MAJ: $\epsilon \leftarrow \epsilon \cdot \beta$
  - Représente l'agressivité 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 ϵ

- 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 le nouvel horaire.
  - Arrêter lorsque la MAJ de ϵ ne fait plus diminuer $J(\boldsymbol{\theta})$.

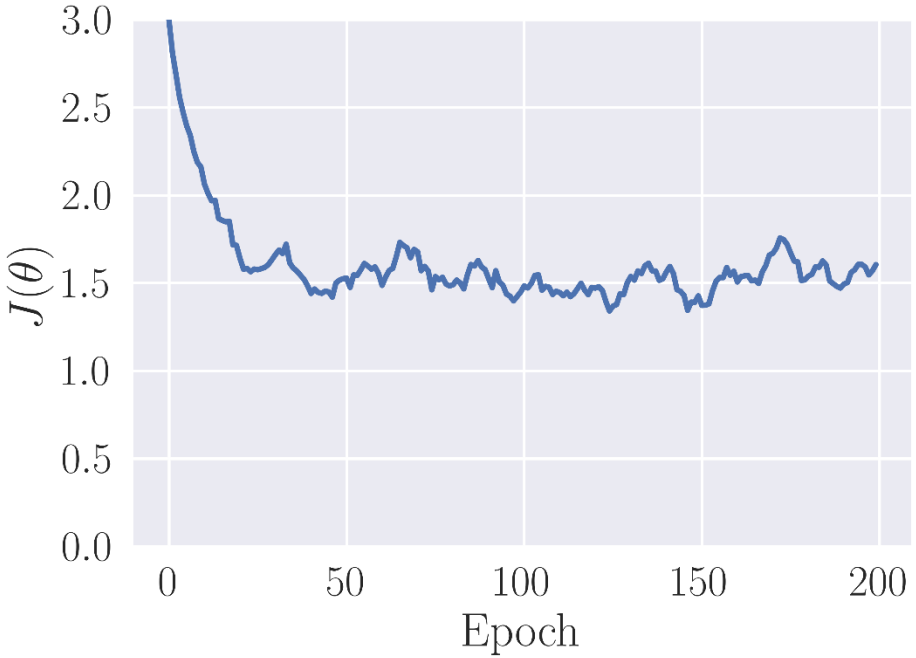
Exemple d'horaire du taux d'apprentissage ϵ

- Valeur initiale ϵ_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 ϵ

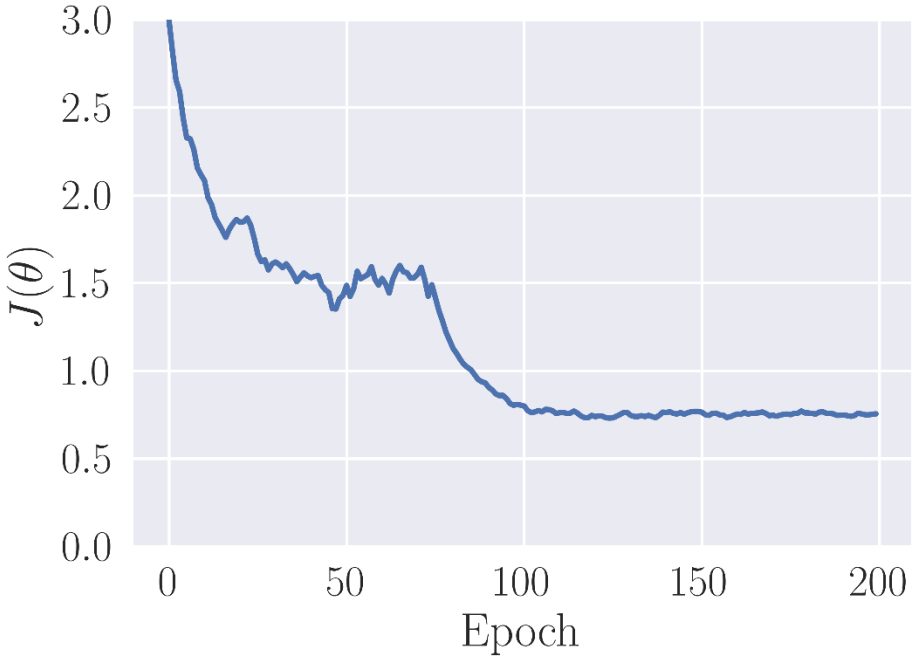
Convergence aux alentours de l'époque 75.



Époque	0
Taux d'apprentissage	0.1

Exemple d'horaire du taux d'apprentissage ϵ

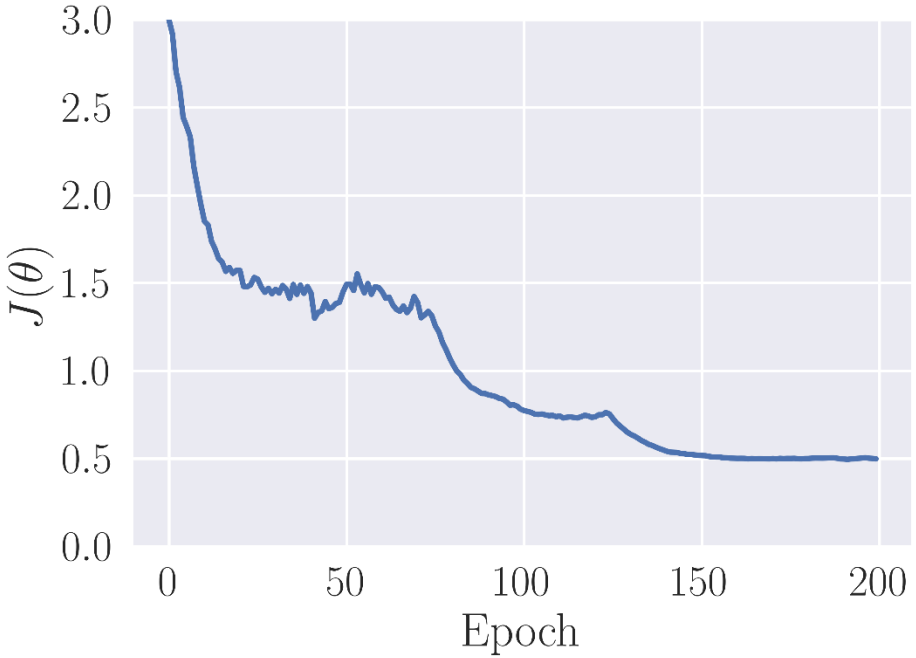
Convergence aux alentours de l'époque 125.



Époque	0	75
Taux d'apprentissage	0.1	0.01

Exemple d'horaire du taux d'apprentissage ϵ

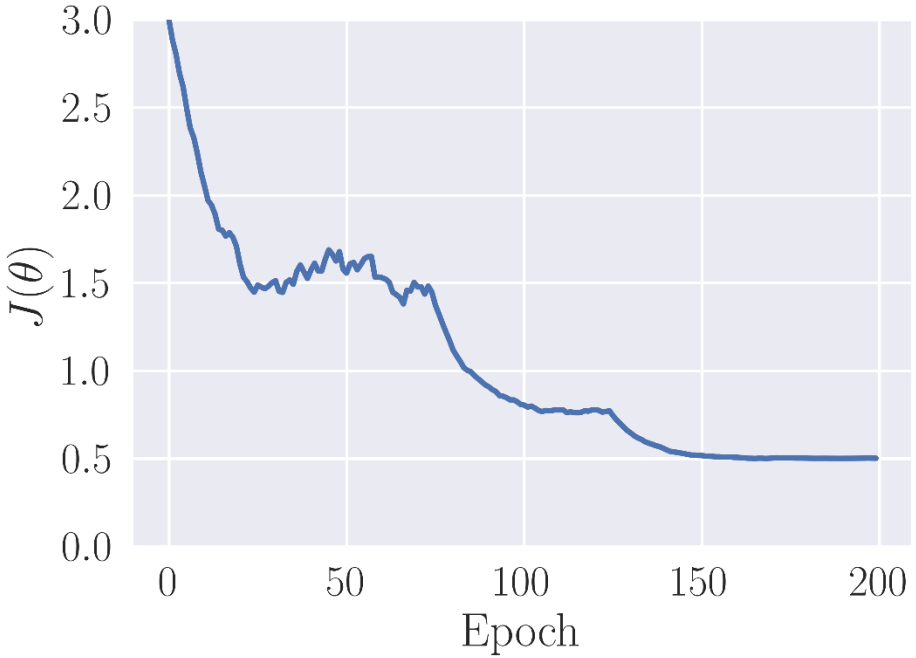
Convergence aux alentours de l'époque 175.



Époque	0	75	125
Taux d'apprentissage	0.1	0.01	0.001

Exemple d'horaire du taux d'apprentissage ϵ

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-paramètre 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.

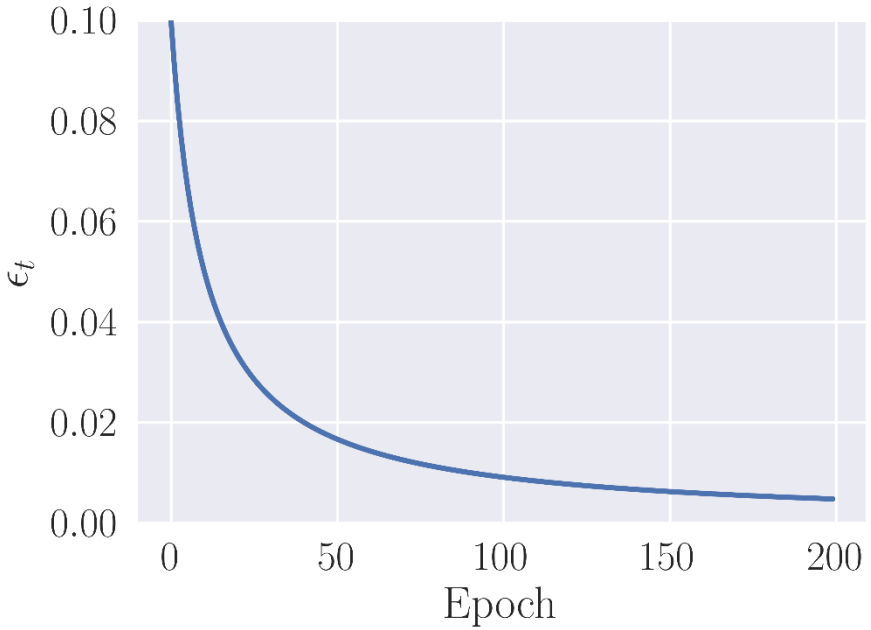
Stratégie #2 pour choisir l'horaire du taux d'apprentissage ϵ

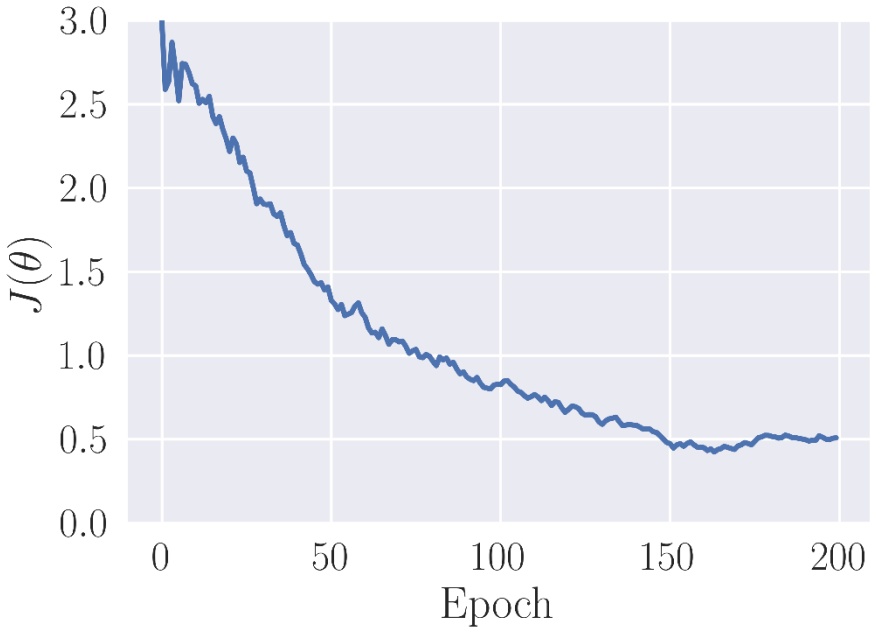
- Exemple: taux d'apprentissage ϵ.
- Valeur initiale ϵ_0
  - Prendre approximativement la plus grande valeur de ϵ_0 qui ne fait pas diverger $J(\boldsymbol{\theta})$.
- Taux de décroissance
  - Dépend du problème.
- Règle de décroissance

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

Exemple d'horaire du taux d'apprentissage ϵ

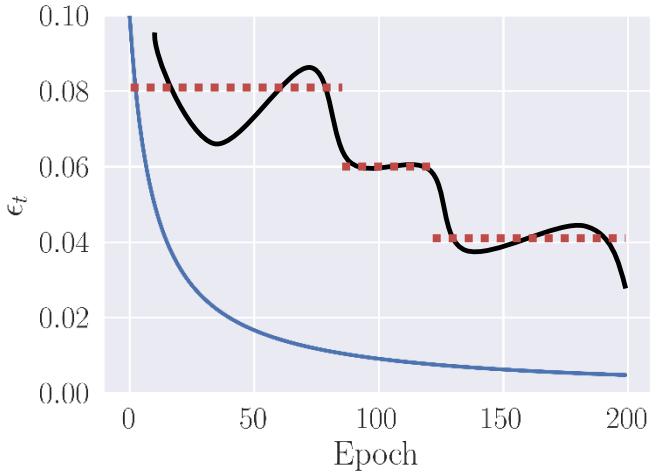
- $\epsilon_0 = 0.1,\ \beta = 0.1,\ \epsilon_t = \frac{\epsilon_0}{1+\beta \cdot t_e}$





Stratégie #1 > Stratégie #2

- Avantage de Stratégie #1
  - Relation non-linéaire entre le taux de décroissance de $J(\boldsymbol{\theta})$ et celui de ϵ_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 Recherche en grille

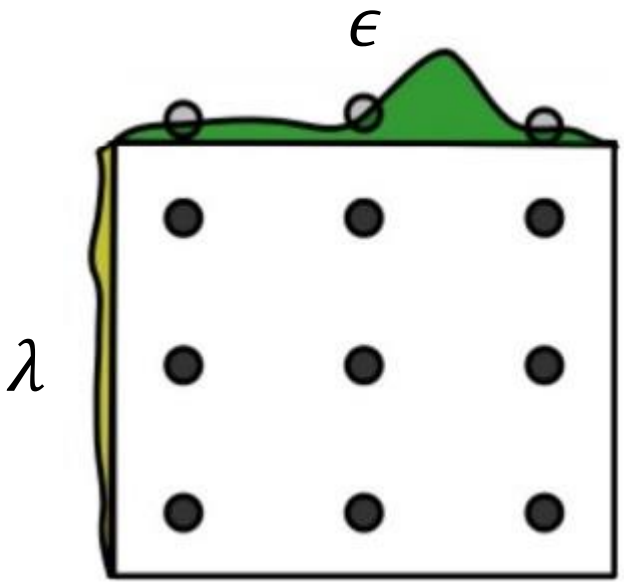
- Certains hyperparamètres doivent rester fixes pour toute la durée de l'optimisation.
  - Nombre de neurones
  - Fonction d'activation
  - Probabilité de dropout (Semaine 4)
  - Dimension des filtres à convolution (Semaine 5)
- D'autres peuvent être fixés même si pas nécessaire.
  - Weight decay (Semaine 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.
    - [0.01 0.02 0.05 0.1 0.2 0.5 1.0]
- 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

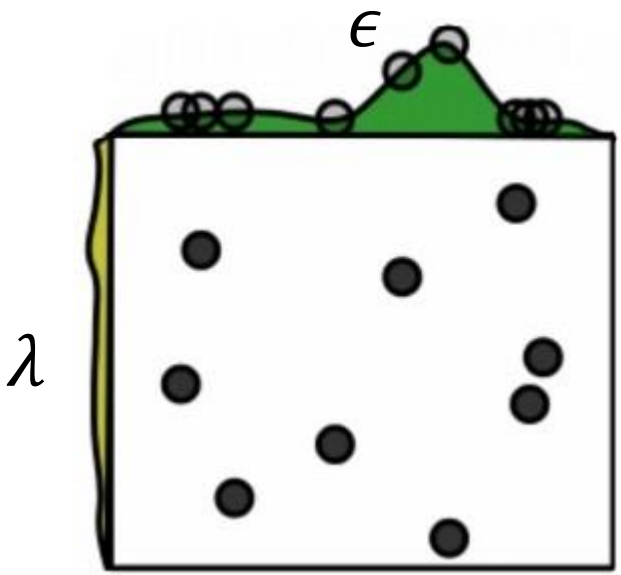
- **Avantages**
  - Profiter de la connaissance d'un expert.
  - Recherche exhaustive (brute force).
  - Facilement parallélisable.
- **Inconvénients**
  - Demande computationnelle élevée.
  - Calcul redondant, car un seul hyperparamètre change à la fois.

1.3 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.
  - Algorithme *Anytime*: peut être arrêté n'importe quand.
    - Plus il roule, meilleure est la recherche.
- **Inconvénients**
  - Demande computationnelle élevée.
  - Difficile à reproduire (random seed).

2. Initialisation des paramètres

- Principe
  - Choisir la valeur initiale de $\boldsymbol{\theta}$ avant l'optimisation
- Importance
  - Aspect souvent négligé.
  - Un réseau mal initialisé peut être difficile à optimiser.
  - $w = -1{,}000{,}000, \ x = 1$
  - $sigmoid(w \cdot x) = \frac{1}{1 + \exp(-w \cdot x)} = \frac{1}{1 + \exp(1{,}000{,}000)}$

2. Initialisation des paramètres

- Va affecter :
  - la vitesse de convergence de l'optimisation
  - la convergence ou non
  - la capacité à généraliser : l'optimisateur va rester somme toutes dans le voisinage de l'initialisation
    - early stopping, par exemple

2. Initialisation des paramètres

Deux façons:

1. Initialisation à partir de rien
   - La seule façon pour des problèmes non-standards, ou une architecture nouvelle.
2. Initialisation par pré-entraînement
   - Utilisée lorsqu'un réseau a déjà été entraîné sur le même type de problème.

Initialisation à partir de rien

- Biais *b* généralement initialisés à 0.
- Exceptions
  - biais sur les scores de la dernière couche, pour avoir une sortie softmax proche de la distribution marginale des exemples d'entraînement
  - légèrement positive pour activer les ReLU
  - pour les gates, avoir un biais de 1, pour qu'elle soit activé raisonnablement (et puisse apprendre)

Initialisation à partir de rien

- Initialisation des poids W

1. Choisir une distribution de probabilité.
2. Choisir ses paramètres.
3. Échantillonner les poids iid.

Initialisation aléatoire des poids

n_i: nombre de neurones en entré.
n_o: nombre de neurones en sortie

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

- 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 ? ¯_(ツ)_/¯
- Module init.py
  - kaiming_normal, kaiming_uniform, xavier_normal, xavier_uniform

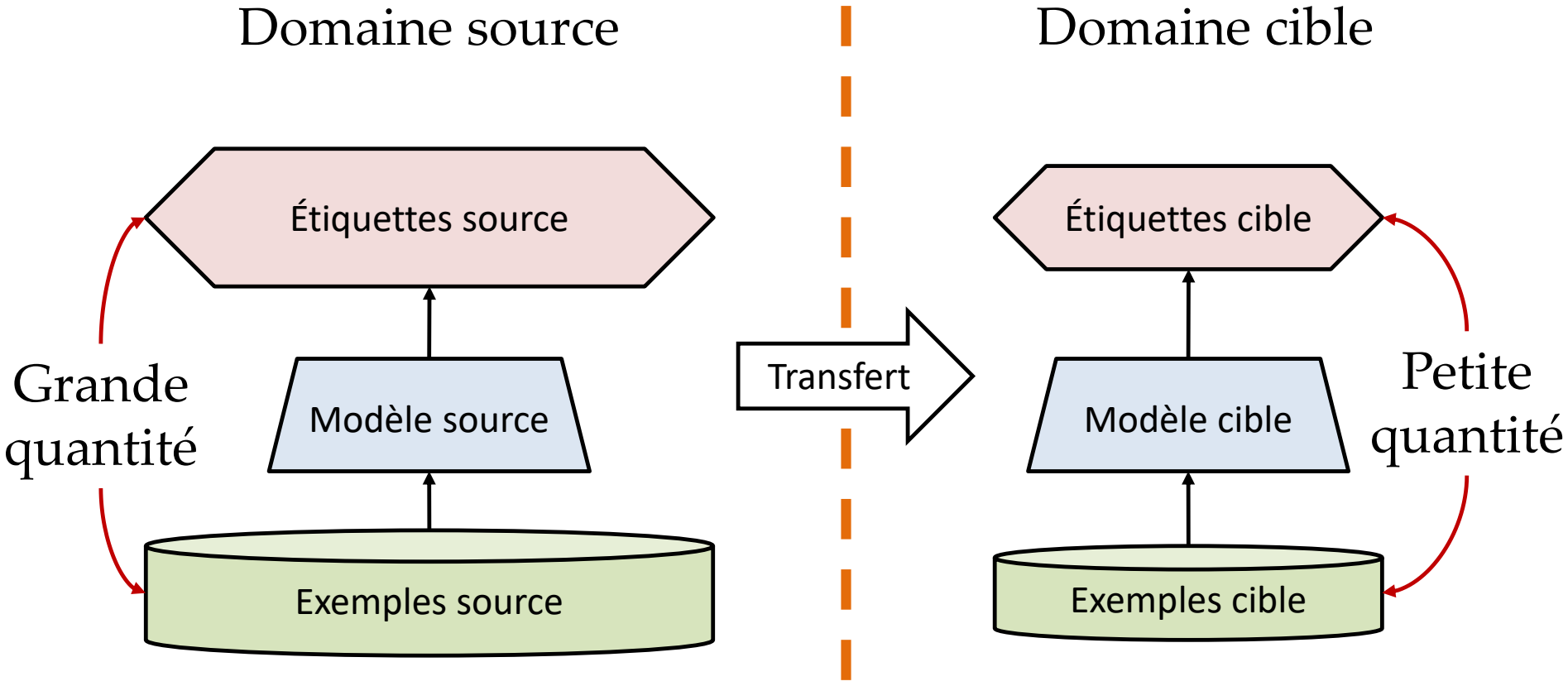
Initialisation identique ?

- Que se passe-t-il si on initialise tous les poids W à une seule valeur constante ?

$$W = \begin{bmatrix} k & k & \cdots & k \\ k & k & \cdots & k \\ \vdots & \vdots & \ddots & k \\ k & k & k & k \end{bmatrix}$$

- Impossible de briser les symétries durant l'entrainement
  - les gradients seront les mêmes
- Équivalent d'avoir un seul neurone!

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 Dom(source) ≈ Dom(cible).

3. Normalisation

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

Normalisation Min-Max

- Principe
  - Transpose l'entrée dans la plage $[0, 1]$
- Méthode

$$\overline{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{x}_{min}}{\boldsymbol{x}_{max} - \boldsymbol{x}_{min}}$$

- Sensibilité aux *outliers*
  - Utilisé seulement quand $\boldsymbol{x}$ 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'entrée pour que moyenne = 0 et variance = 1.
- Méthode

$$\overline{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{x}_{\mu}}{\sqrt{\boldsymbol{x}_{\sigma^2} + \delta}}$$

  - $\boldsymbol{x}_{\mu}$ et $\boldsymbol{x}_{\sigma^2}$ calculés avec tous les $\boldsymbol{x}^{(i)}, i = 1 \ldots N$, 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
  - $\boldsymbol{h}^l = \mathcal{F}\left(\boldsymbol{h}^{l-1}\right)$
  - $\overline{\boldsymbol{h}}^l = \boldsymbol{\alpha} \cdot \frac{\boldsymbol{h}^l - {\boldsymbol{h}^l}_{\mu}}{\sqrt{{\boldsymbol{h}^l}_{\sigma^2} + \delta}} + \boldsymbol{\beta}$
  - ${\boldsymbol{h}^l}_{\mu}$et ${\boldsymbol{h}^l}_{\sigma^2}$ calculés seulement sur les $\boldsymbol{h}^l$ de la batch. Elle doit être donc assez grande.
  - $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$ paramètres additionnels à apprendre.

Normalisation par batch

- Pour le mode test
  - Moyenne mobile: $\boldsymbol{m} \leftarrow \rho \boldsymbol{m} + (1 - \rho)\, \boldsymbol{h}^{l}{}_{\mu}$
  - Variance mobile: $\boldsymbol{s} \leftarrow \rho \boldsymbol{s} + (1 - \rho)\, \boldsymbol{h}^{l}{}_{\sigma^2}$
  - $\overline{\boldsymbol{h}}^{l} = \boldsymbol{\alpha} \cdot \frac{\boldsymbol{h}^{l} - \boldsymbol{m}}{\sqrt{\boldsymbol{s} + \delta}} + \boldsymbol{\beta}$
- Initialisation
  - $\boldsymbol{\alpha} = 1, \boldsymbol{\beta} = 0, \boldsymbol{m} = 0, \boldsymbol{s} = 1$
- Note : une paire de valeur α, β par neurone.

Avantages de la normalisation par batch

- Avantage #1
  - Apprentissage des paramètres α, β 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.

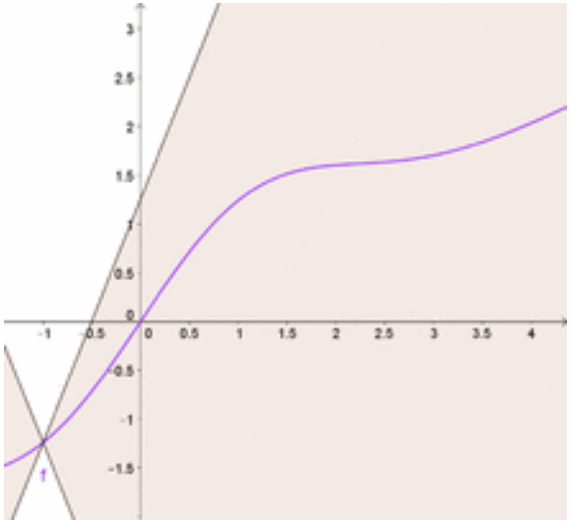
Avantages de la normalisation par batch

- Avantage #4
  - Réseau converge plus rapidement.
- Avantage #5
  - Diminue le nombre de neurones morts / désactivés de la ReLU.
- Avantage #6
  - Diminue l'impact de l'initialisation.

Pourquoi BatchNorm fonctionne

- Son succès est indéniable
- Les causes de ce succès sont *nébuleuses*
  - initialement attribué à la diminution du covariate shift
  - plus récemment, invoque plutôt un lissage de la fonction objectif via le critère de Lipschitz [1]

Lipschitz-continue



[1] Santurkar S. et al, How Does Batch Normalization Help Optimization, arXiv, 2018.

Notes pratiques

- La normalisation par batch (BatchNorm) devrait **toujours être utilisée**
- Endroit où placer le BN ne fait pas l'unanimité
  - Placer avant activation non-linéaire ? après ?
  - Ne semble pas causer de différences majeures de performance
- Ne pas combiner avec le *Drop-out* (Semaine 4)

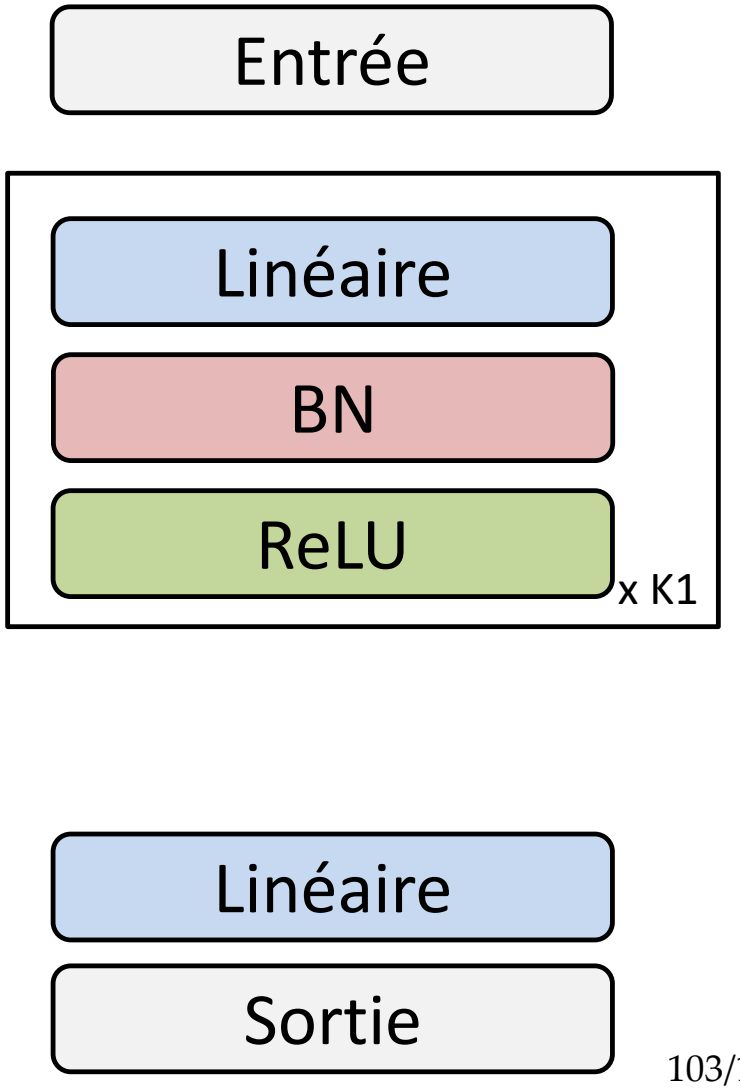
Diagnostique

Diagnostique

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

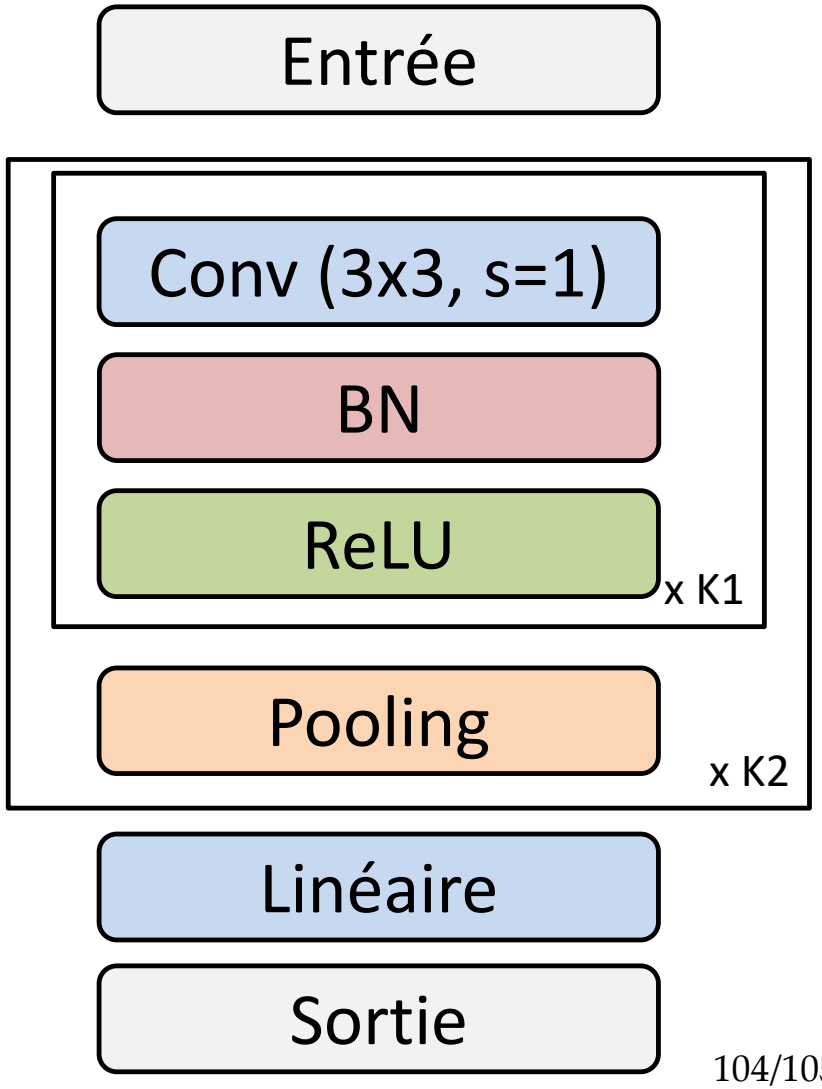
1. Structure neuronale de base

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



Structure neuronale de base

- Convolution (Sem 5-6)
  - Initialisation Kaiming He normale
  - bias = False
  - BN: $\boldsymbol{\alpha} = 1$, $\boldsymbol{\beta} = 0$, $\boldsymbol{m} = 0$, $\boldsymbol{s} = 1$
  - 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 signifie?
    - taux d'erreur=5% est plus parlant

Information sur le réseau

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

Information sur les donnés

- 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 $\boldsymbol{d}$
- 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 graduellement 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'entrainement
  - Augmentation de donnée maximale
- Exemple
  - Classification 100 classes avec 100,000 exemples par classe.

4. Analyse

- But
  - Obtenir une rétrospective 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 prédictions 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 minibatches

- 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.