Deep Learning Optimisé - Jean Zay

Entraînement et large batches

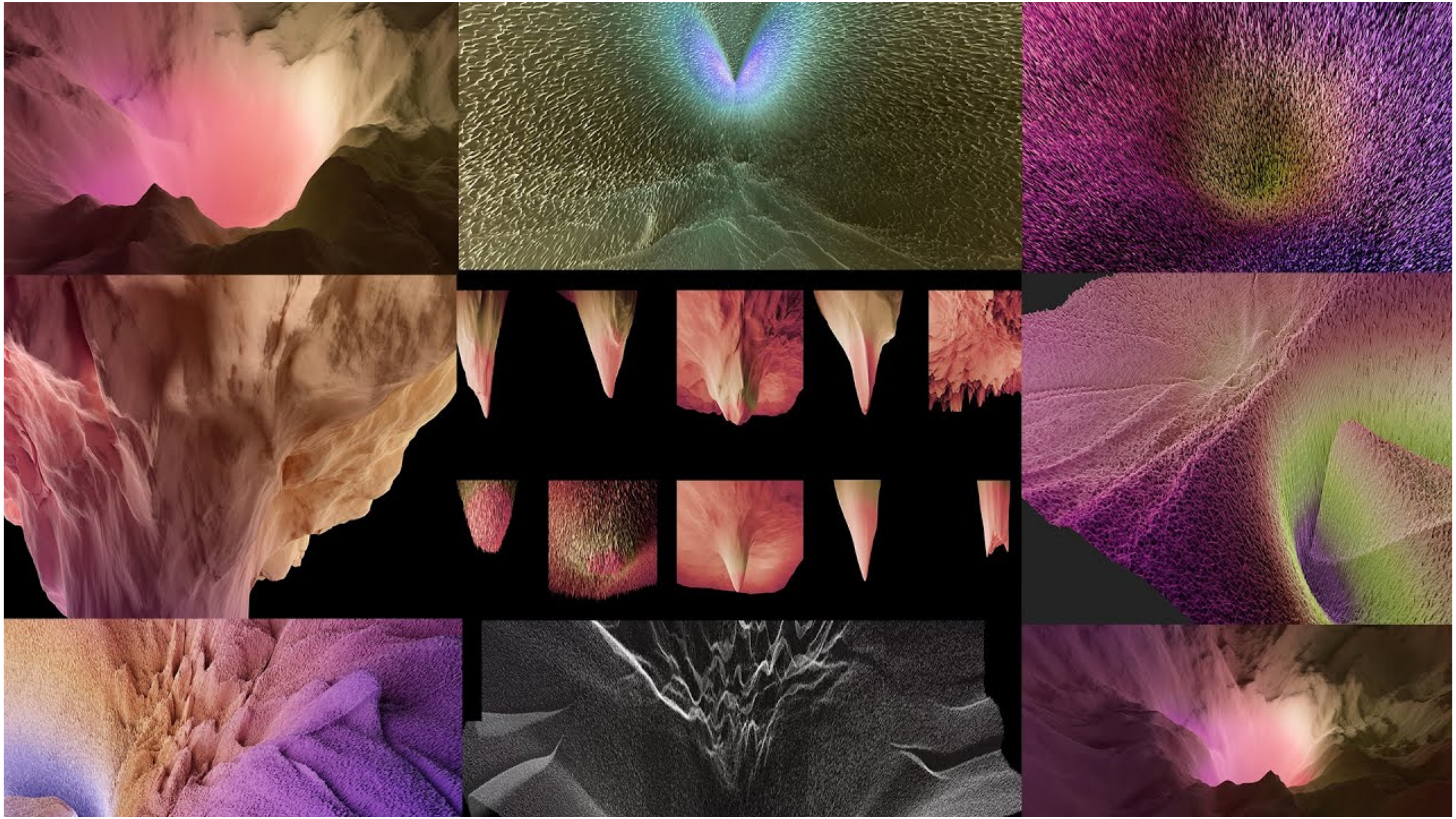
cnrs

Institut du Développement et des Ressources en Informatique Scientifique

Loss Landscape

Loss Landscape

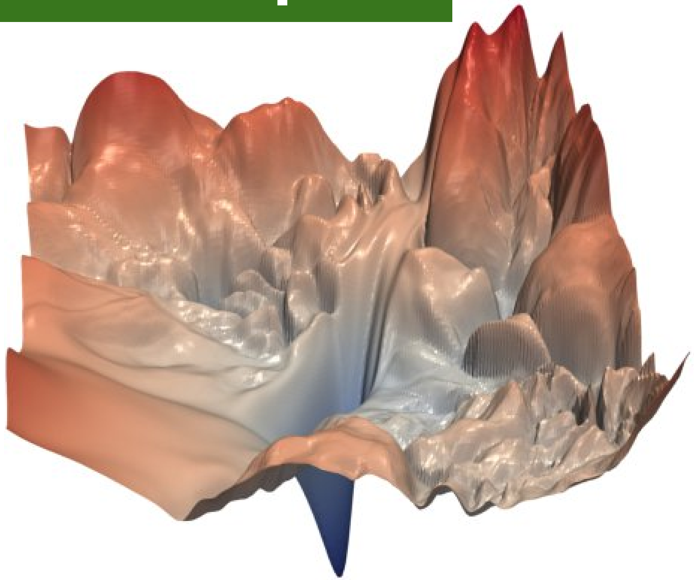
https://losslandscape.com/



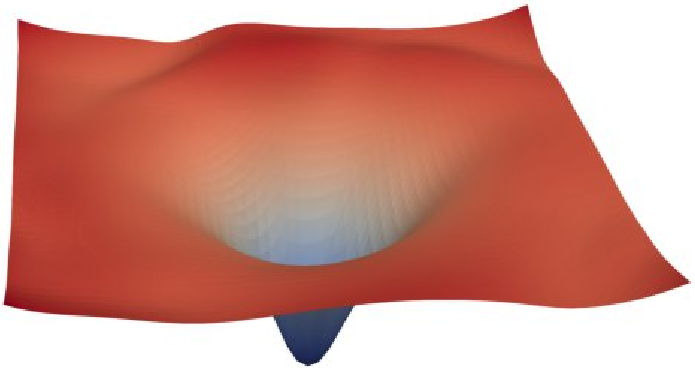
Loss Landscape

https://arxiv.org/pdf/1712.09913.pdf

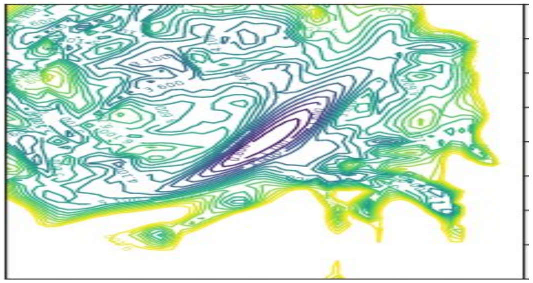
Residual Learning
Depuis les Resnets (2015) ...

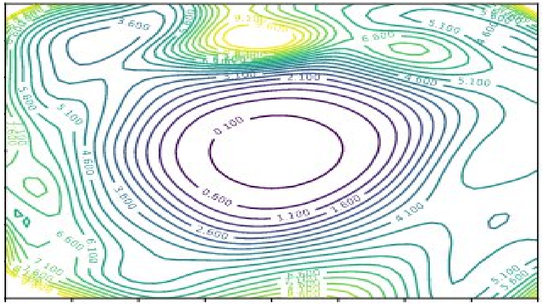


(a) without skip connections

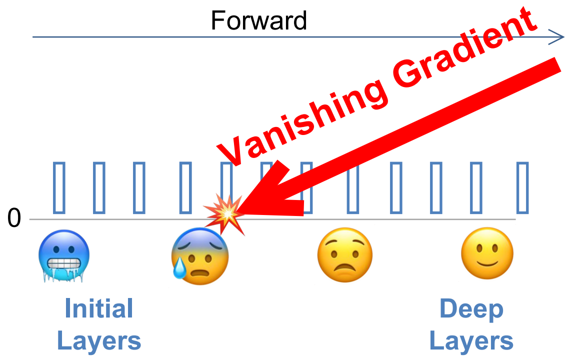


(b) with skip connections



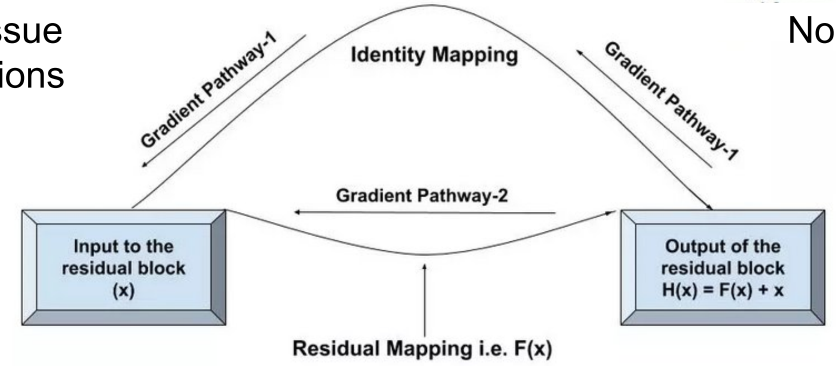


Residual Learning

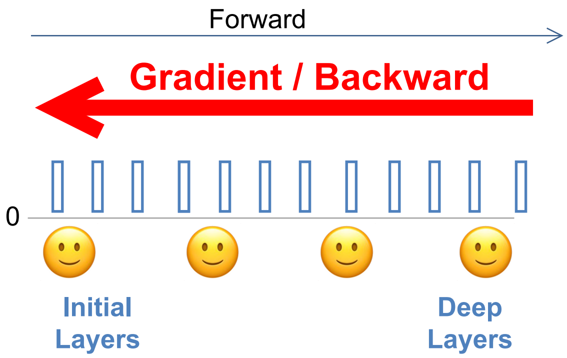


Vanishing Gradient issue
without skip connections

Residual Block
F(x)+x



Gradient Pathways in ResNet

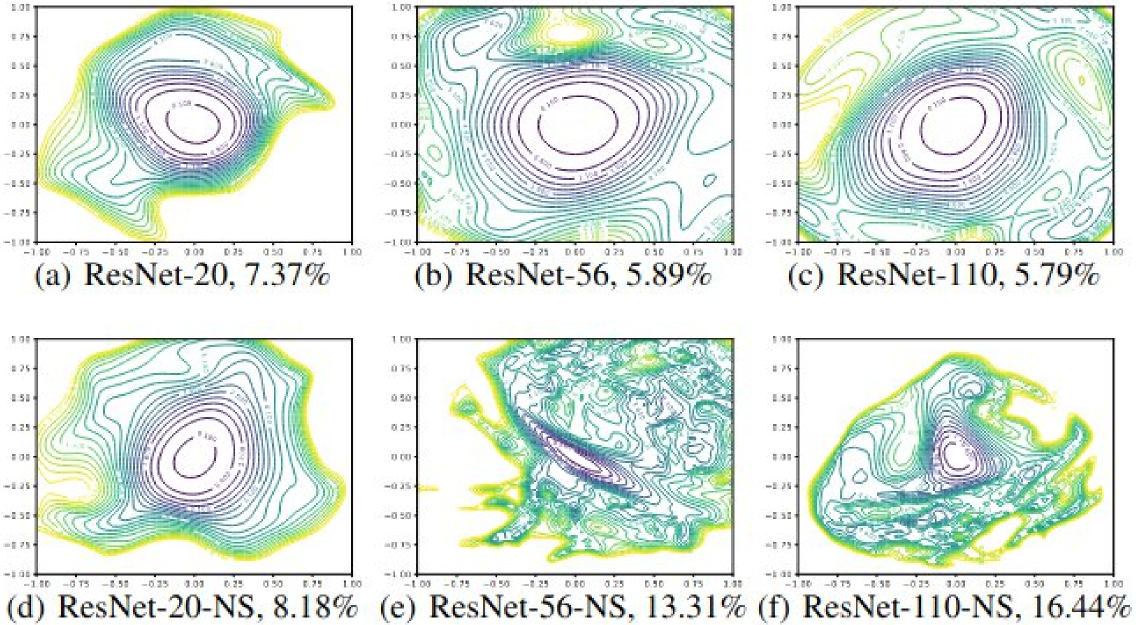


No Vanishing Gradient issue
with skip connections

Residual Learning – impact sur la profondeur

ResNet

ResNet -
No Short
without skip connections

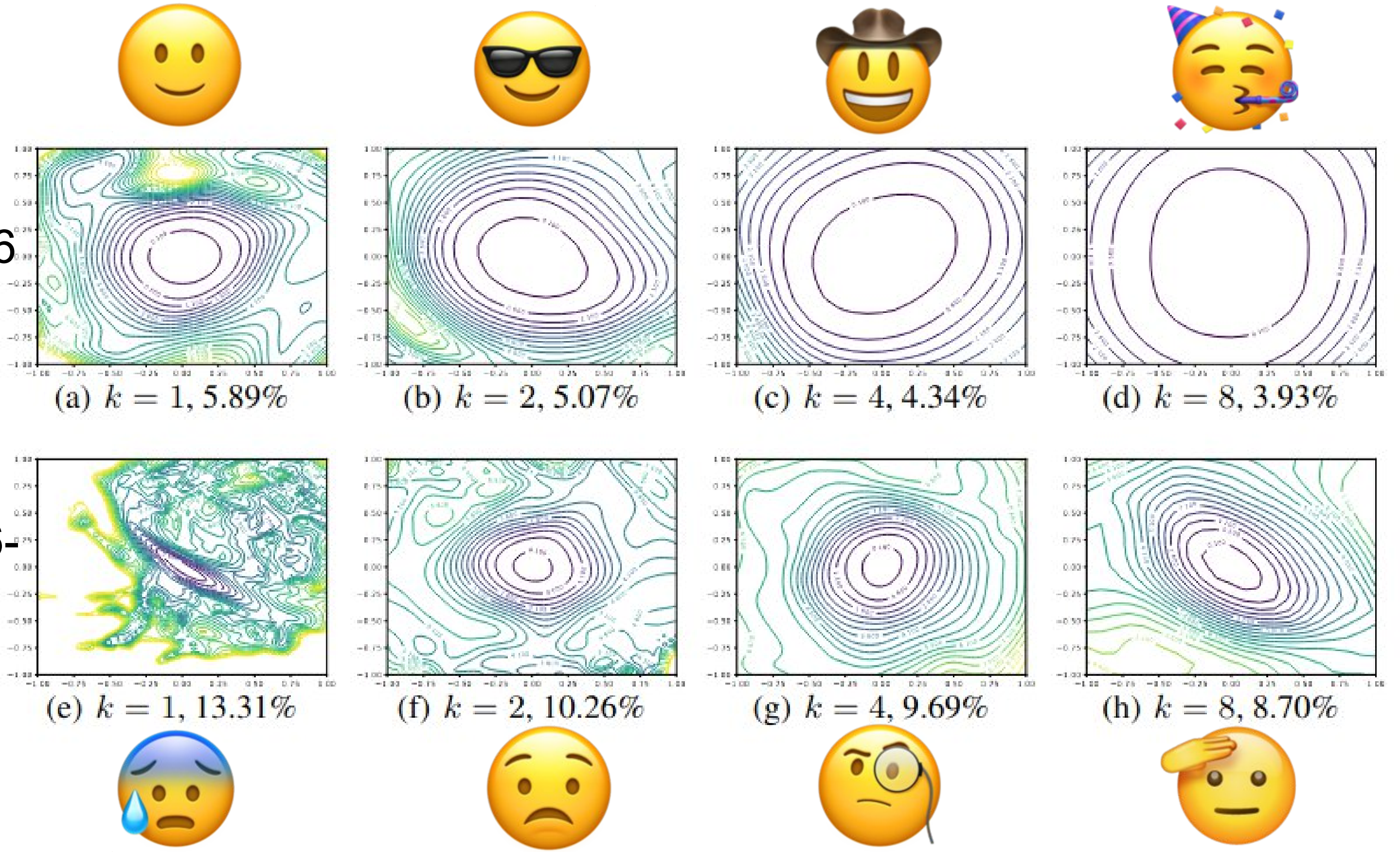


Residual Learning – impact sur la largeur

Wide-ResNet-56

Wide-ResNet-56-No Short

without skip connections



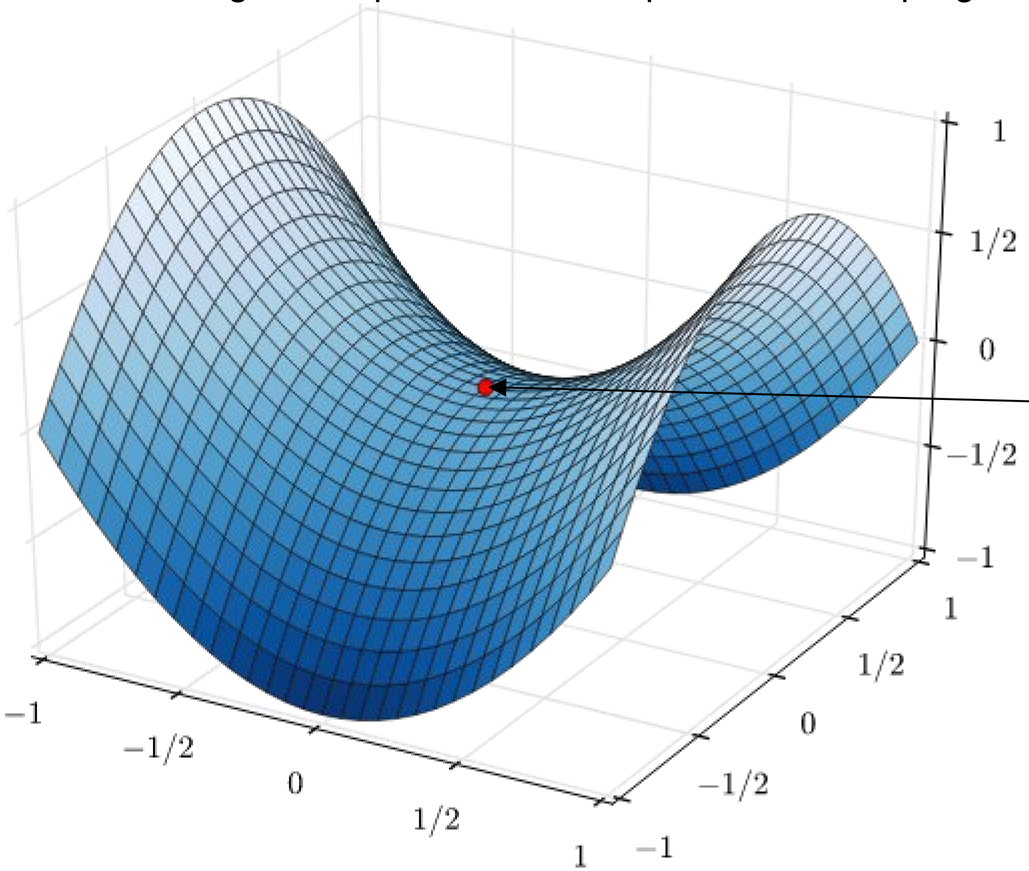
(a) $k = 1, 5.89\%$ (b) $k = 2, 5.07\%$ (c) $k = 4, 4.34\%$ (d) $k = 8, 3.93\%$

(e) $k = 1, 13.31\%$ (f) $k = 2, 10.26\%$ (g) $k = 4, 9.69\%$ (h) $k = 8, 8.70\%$

k = coefficient largeur des *channels par rapport à ResNet*

Problème du point-selle

Point-selle : Un gradient proche de zéro qui va rendre la progression du modèle très lente



Comment quitter le point-selle ?

Initialisation des paramètres du modèle

The Blessing of Dimensionality :

- Xavier Initialization
  - uniform
  - normal

- Kaiming Initialization
  - uniform
  - normal

Par défaut dans *PyTorch* :

- Meilleur algorithme d'initialisation selon le type de couche (linéaire, convolutionnel, transformer, ...).
- Aujourd'hui, il n'est plus nécessaire de chercher à optimiser l'initialisation.

Learning rate scheduler

Learning rate scheduler ◂

Cyclic scheduler ◂

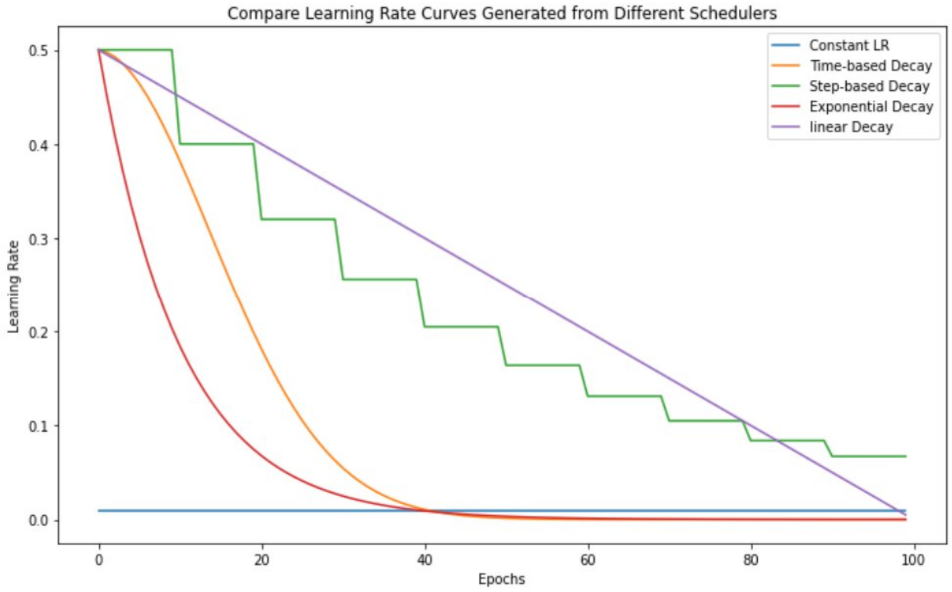
One cycle scheduler ◂

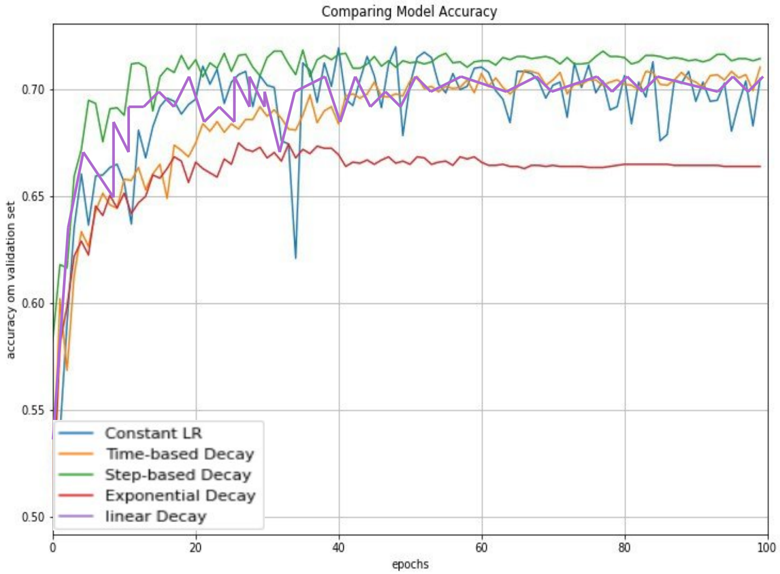
LR finder ◂

LR large batch ◂

Learning Rate Scheduler

Learning rate decay

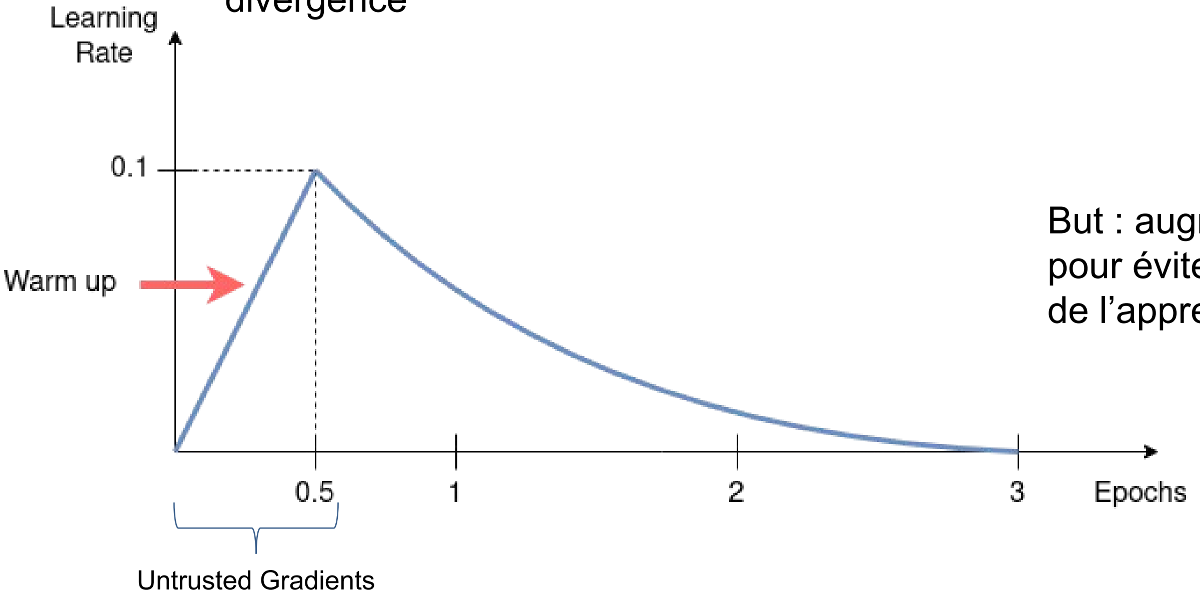




Learning Rate Scheduler

WARMUP pour *large batches*

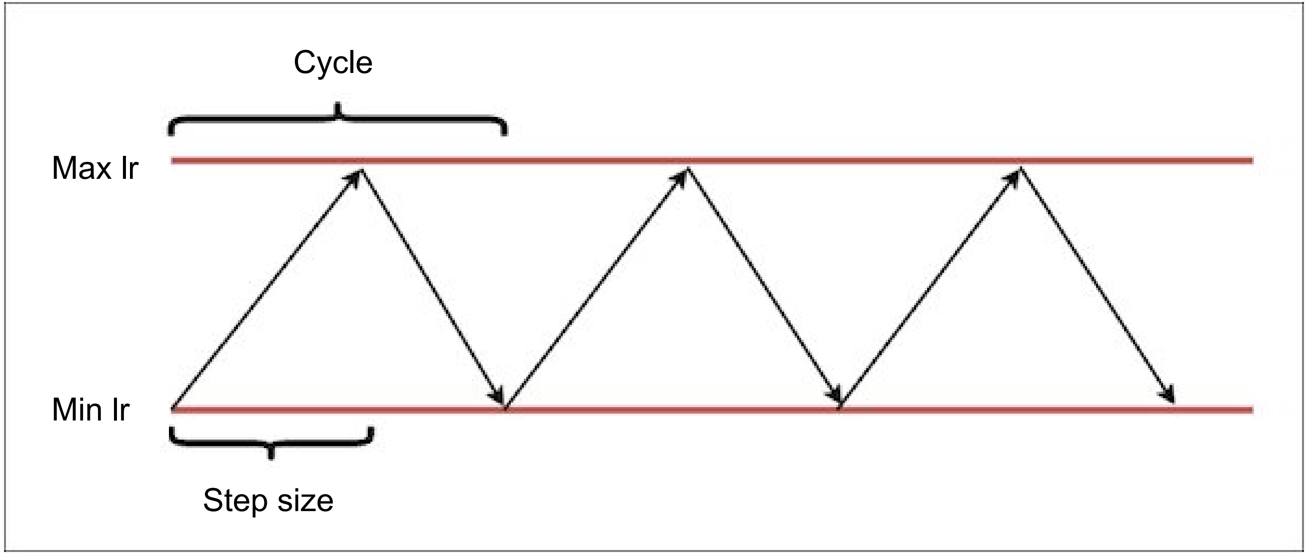
Problèmes : Les premières itérations ont trop d'effet sur le modèle (loss importantes, gradients élevés, biais, …), un *learning rate* élevé peut provoquer une forte instabilité ou une divergence



But : augmenter petit à petit le *learning rate* pour éviter le risque de divergence au début de l'apprentissage

Cyclic Learning Rate Scheduler

Cyclical Learning Rates for Training Neural Networks - Leslie N. Smith 2017

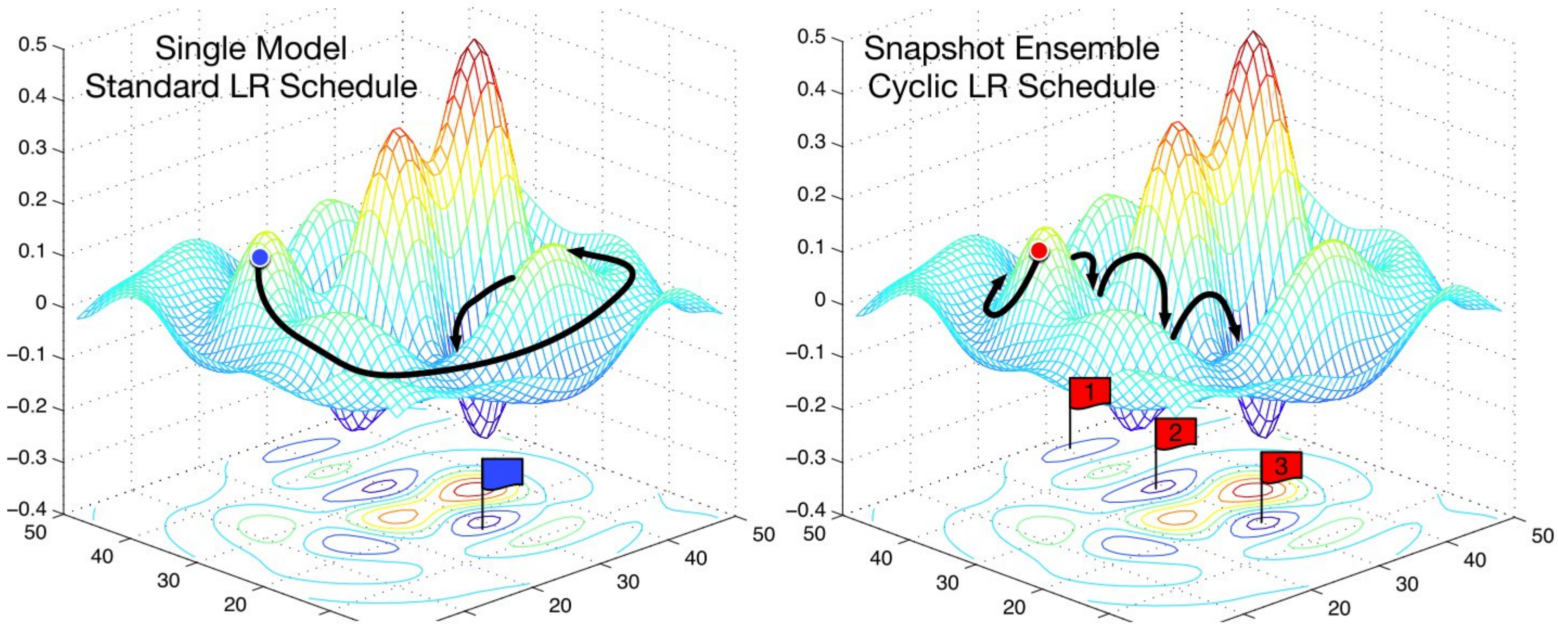


Paramètres :

- Step_size = x * epoch (2 ≤ x ≤ 10)
- Base_lr -> valeur minimum de convergence
- max_lr -> valeur maximale avant divergence

Succession de *warmups* et de *learning rate decays*

Cyclic Learning Rate Scheduler



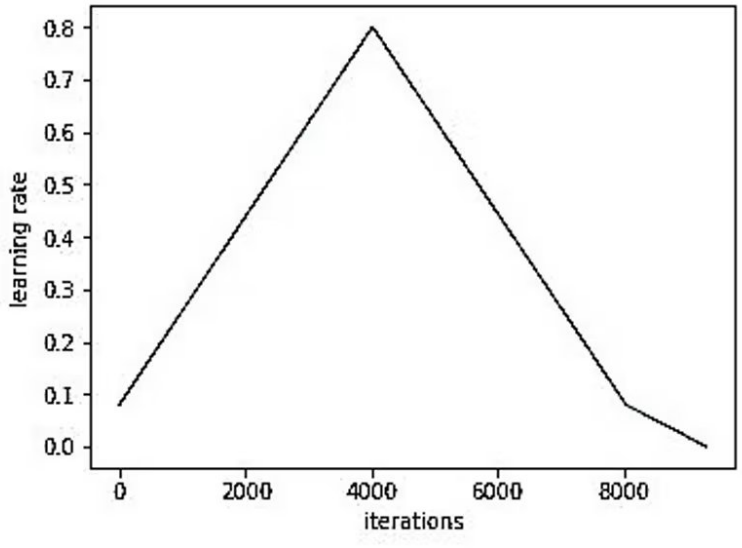
SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE
Gao Huang, Yixuan Li, Geoff Pleiss

One Cycle Learning Rate

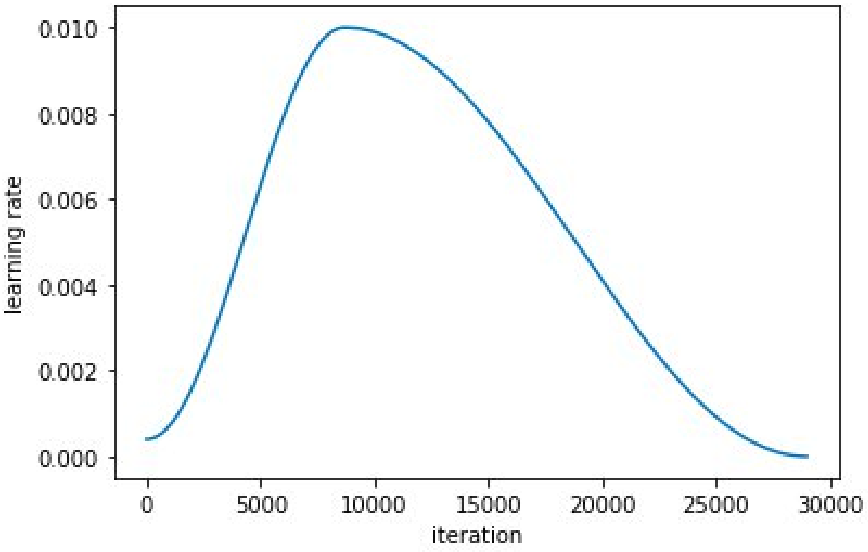
Un seul cycle suffit !

A disciplined approach to neural network hyper-parameters - [Leslie N. Smith]

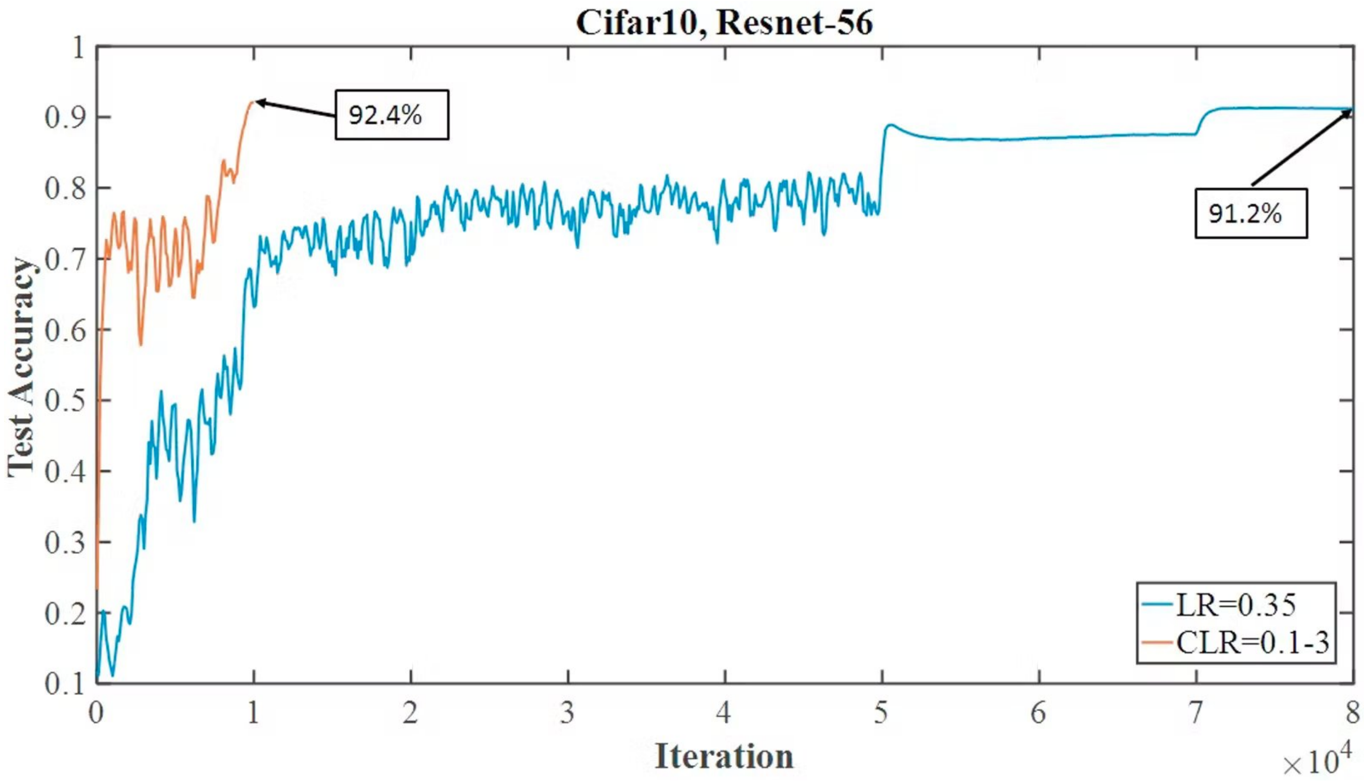
Proposition initiale



cosine annealing : Recommandation par FastAI



One Cycle Learning Rate - Super convergence



Une convergence plus rapide pour une précision finale équivalente

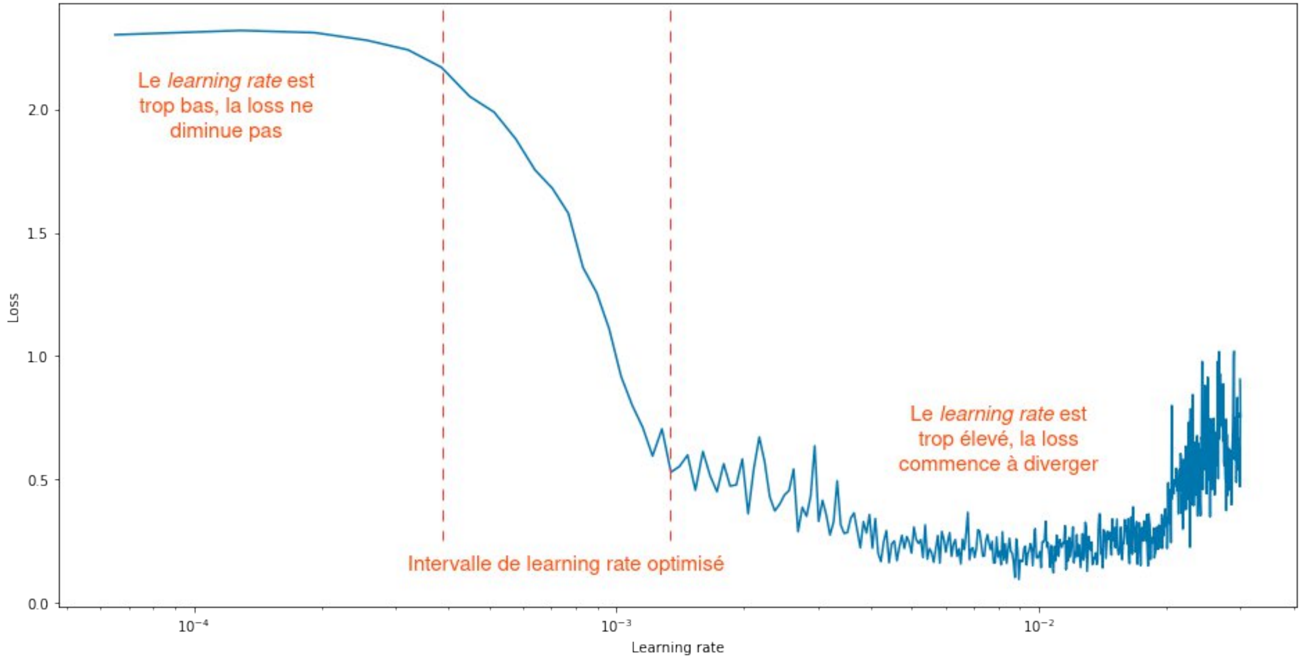
Learning Rate Finder

But : Trouver les valeurs de *learning rate* optimales pour son modèle, particulièrement pour la valeur maximale d'un *cyclic scheduler*

- Faire tourner son modèle sur quelques *epochs* en faisant augmenter son *learning rate*

- Début de baisse de la *loss* → *Learning rate* minimal

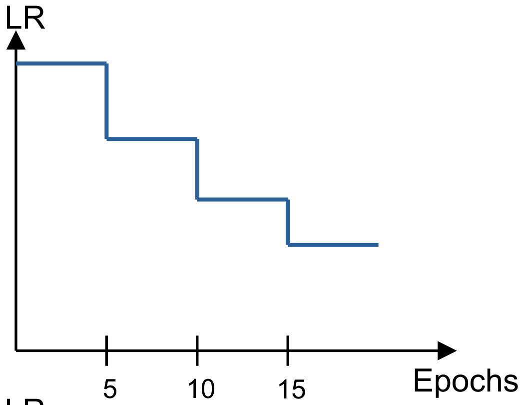
- Début de variation de la *loss* → ***Learning rate* maximal**



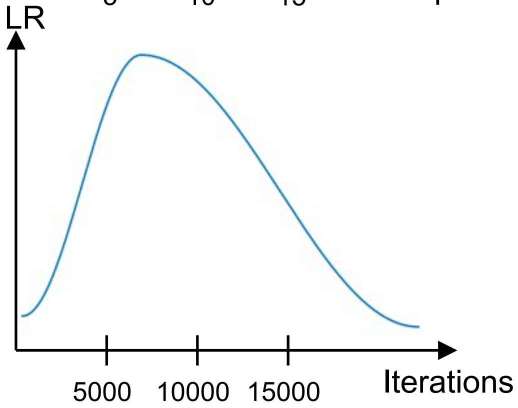
Learning Rate Scheduler

Chaque *scheduler* a ses propres paramètres

```
import torch.optim as opt

scheduler = opt.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

for epoch in range(100):
        train(...)
        validate(...)
        scheduler.step()
```




```
import torch.optim as opt

scheduler = opt.lr_scheduler.CyclicLR(optimizer, base_lr=0.01, max_lr=0.1)


for epoch in range(10):
        for batch in data_loader:
                train_batch(...)
                scheduler.step()
```



Optimiseur de descente de gradient

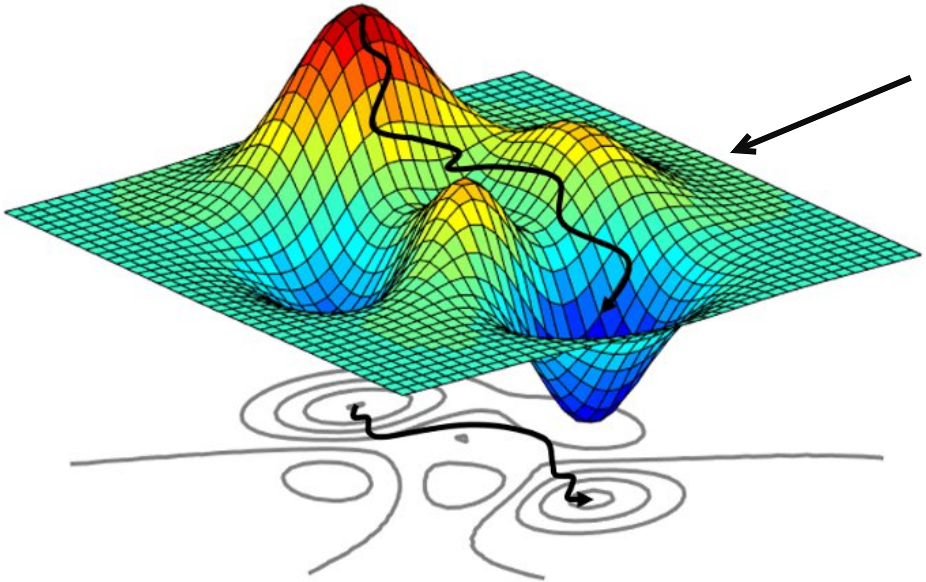
SGD ◂

ADAM◂

ADAMW ◂

Optimiseur - SGD

L'optimiseur est l'algorithme qui pilote la descente de gradient et la recherche de minimum avec pour but d'optimiser le temps d'apprentissage et la métrique finale.



SGD = *Stochastic Gradient Descent*

Calcul du Gradient et mise à jour des poids

à chaque *batch*

\+ Taille de *batch et learning rate* adaptables selon les besoins contradictoires :

- D'exploration pour trouver le meilleur minimum local
- D'accélération de la descente de gradient

SGD with Momentum

Coefficient de *momentum*

$$m_0 = 0$$

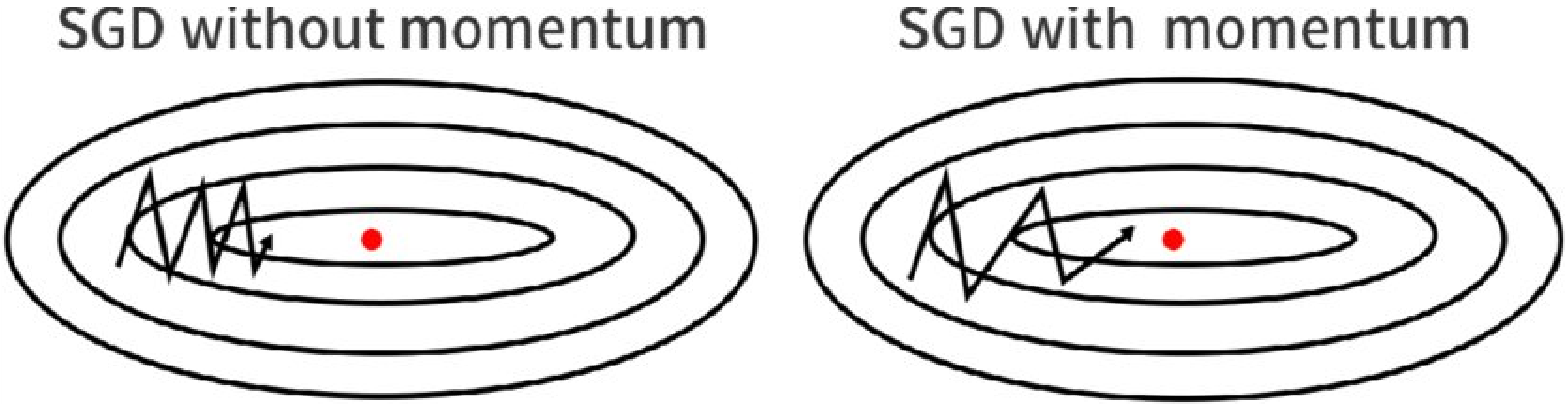
$$m_i = \boxed{\beta} * m_{i-1} + (1 - \beta) * g_i$$

$$\theta_i = \theta_{i-1} - \alpha * m_i$$

Objectif : Prendre en considération les gradients précédents pour une descente de gradient plus rapide.

Valeur initiale conseillée : 0,9

$$0.85 < \beta < 0.95$$

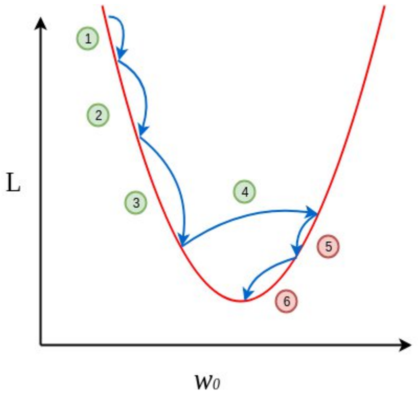


+ Permet de converger plus rapidement

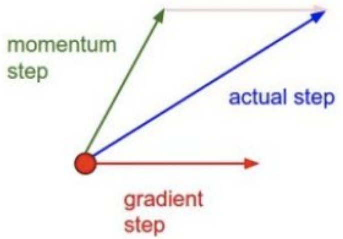
- Pas de garantie que le *momentum* nous amène dans la bonne direction

Types de *Momentum*

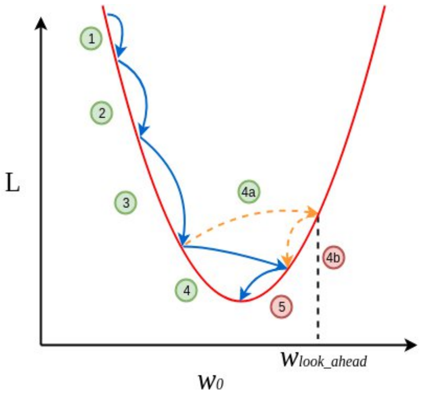
Momentum



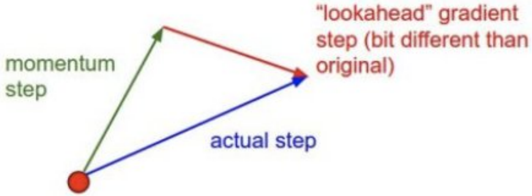
Momentum update

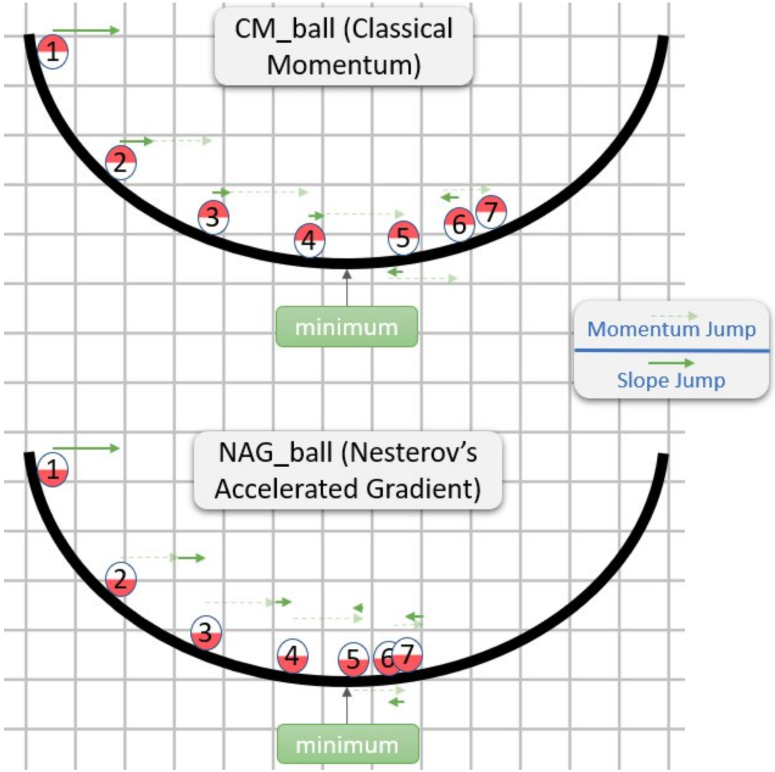


Nesterov momentum



Nesterov momentum update

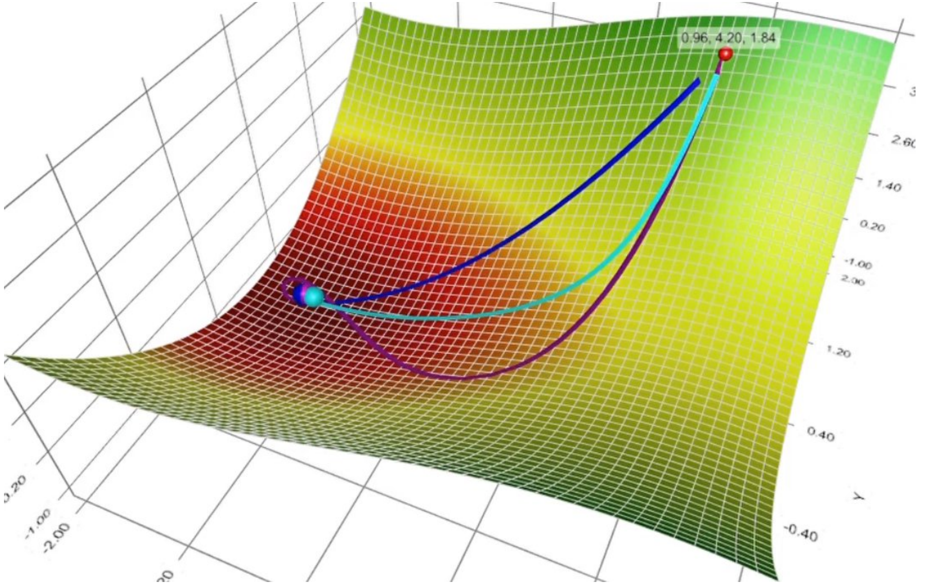




Optimiseurs adaptatifs

Plutôt que de piloter la descente de gradient manuellement avec le *learning rate* …

SGD (no *momentum*)
SGD (with *momentum*)
Adam



Nous pouvons adapter le *learning rate* **pour chaque poids du modèle** en fonction du gradient, du gradient², ou de la norme des poids de la couche !!

Exemples :

- AdaGrad,
- AdaDelta,
- RMSprop
- **Adam**

Spécialisés pour les batches larges :

- **LARS**
- **LAMB**

Adam

Adam : Adaptative moment estimation

$$m_i = \beta_1 * m_{i-1} + (1 - \beta_1) * g_i$$

Premier moment : moyenne glissante

$$v_i = \beta_2 * v_{i-1} + (1 - \beta_2) * g_i^2$$

Second moment : variance non centrée glissante

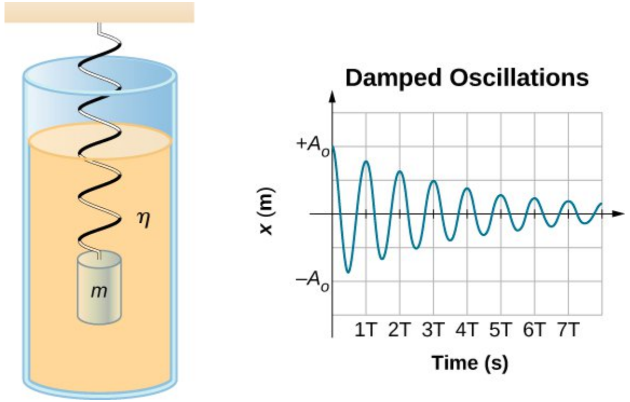
$$\hat{m}_i = \frac{m_i}{1 - \beta_1^i}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_2^i}$$

Correction des biais des premières itérations

$$\theta_i = \theta_{i-1} - \frac{\alpha}{\sqrt{\hat{v}_i + \epsilon}} * \hat{m}_i$$

Amortit les oscillations



Paramètres :

β_1 & β_2 = Taux de régression (β_1 = 0.9 & β_2 = 0.999)

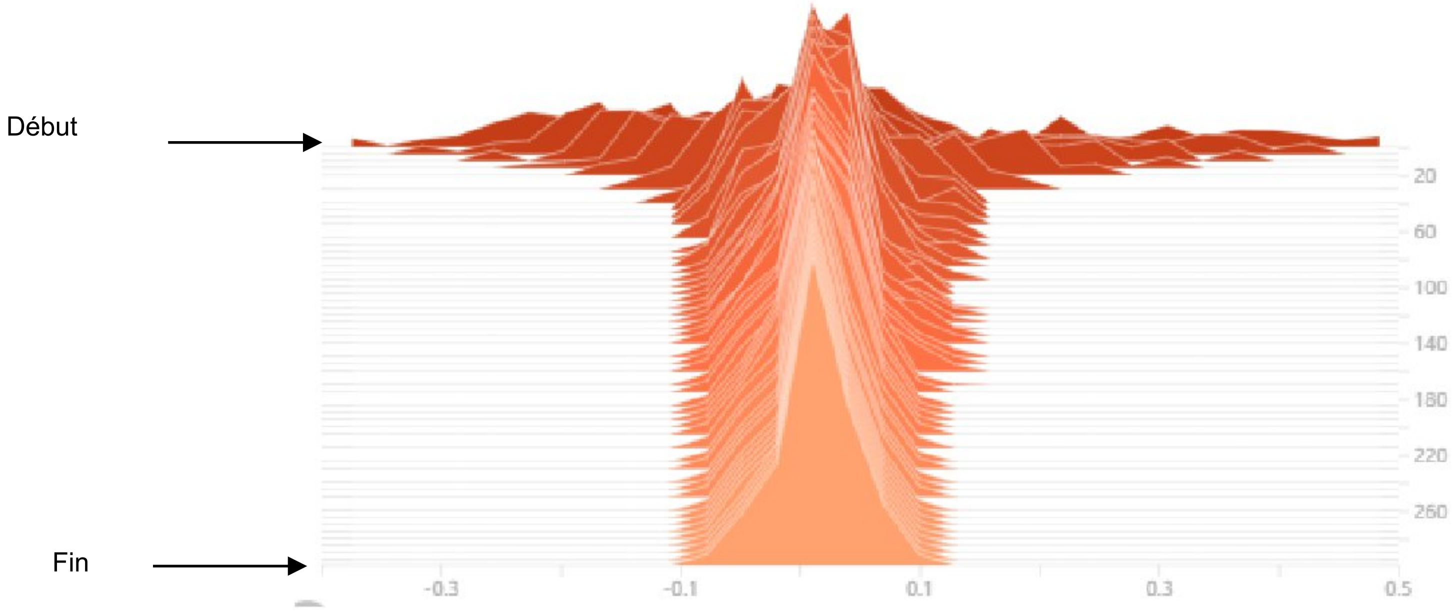
ϵ — Très petite valeur pour éviter une division par zéro

But : Adapter l'importance des mise à jours de poids en fonction des précédents gradients et de la variabilité du gradient.

Weight decay

Un réseau de neurones qui converge et généralise correctement (ni sous-apprentissage ni sur-apprentissage) a généralement **des poids qui tendent vers 0.**

Distribution des poids durant l’apprentissage :



Weight decay

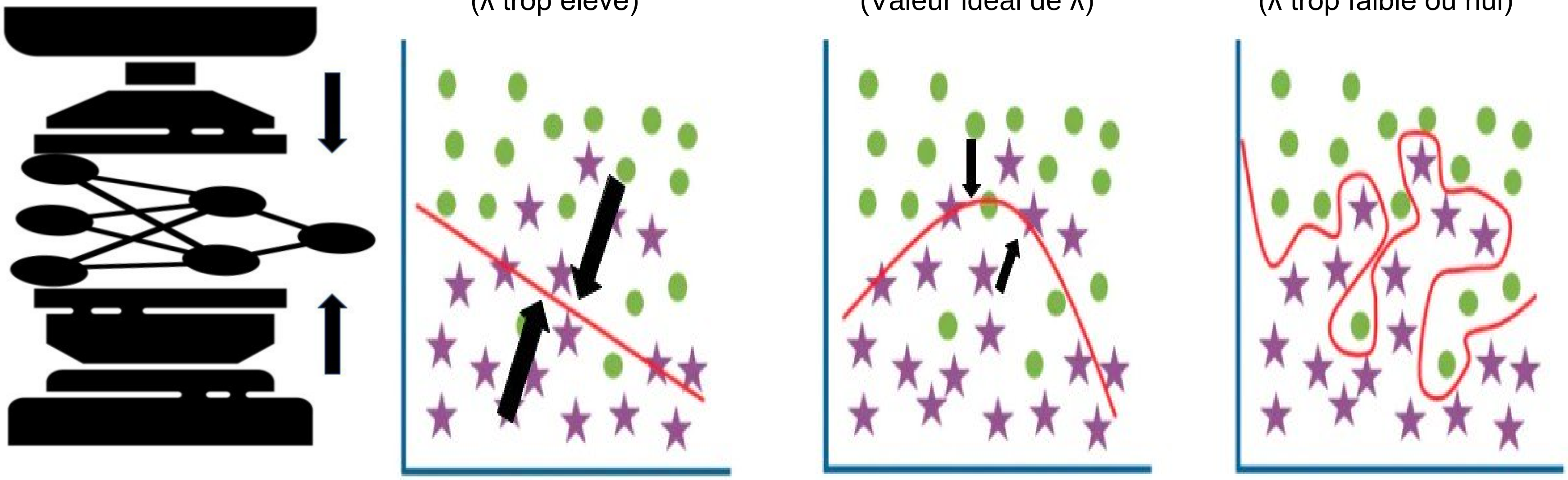
Préférable à la régularisation L2 standard définie dans la fonction de perte

λ : paramètre du *weight decay (généralement entre 0 et 0.1)*

Underfitting
(λ trop élevé)

Weight decay correct
(Valeur idéal de λ)

Overfitting
(λ trop faible ou nul)



La technique de *weight decay*, définie dans *l'optimizer* permet de forcer les poids à converger vers des valeurs proches de zéro.

Weight decay and decoupled weight decay

ADAM

For i = 1 to ...

$$g_i = \nabla_\theta f_i(\theta_{i-1}) \boxed{+ \lambda\theta_{i-1}}$$
$$m_i = \beta_1 * m_{i-1} + (1-\beta_1) * g_i$$
$$v_i = \beta_2 * v_{i-1} + (1-\beta_2) * g_i^2$$
$$\hat{m}_i = \frac{m_i}{1-\beta_1^i}$$
$$\hat{v}_i = \frac{v_i}{1-\beta_2^i}$$
$$\theta_i = \theta_{i-1} - \frac{\alpha}{\sqrt{\hat{v}_i + \epsilon}} * \hat{m}_i$$

Return θ_i

Weight decay

ADAMW

For i = 1 to ...

$$g_i = \nabla_\theta f_i(\theta_{i-1})$$
$$m_i = \beta_1 * m_{i-1} + (1-\beta_1) * g_i$$
$$v_i = \beta_2 * v_{i-1} + (1-\beta_2) * g_i^2$$
$$\hat{m}_i = \frac{m_i}{1-\beta_1^i}$$
$$\hat{v}_i = \frac{v_i}{1-\beta_2^i}$$
$$\theta_i = \theta_{i-1} - \frac{\alpha}{\sqrt{\hat{v}_i + \epsilon}} * \hat{m}_i \boxed{- \alpha\lambda\theta_{i-1}}$$

Return θ_i

Decoupled weight decay

Évolution du *weight decay*: *Decoupled weight decay* (découplé du momentum !!)

- SGD et Adam avec le *weight decay*
- SGDW et AdamW avec le *decoupled weight decay*

SGD et SGDW sont à peu près équivalents en performance.

Cependant AdamW est notablement meilleur que Adam !!

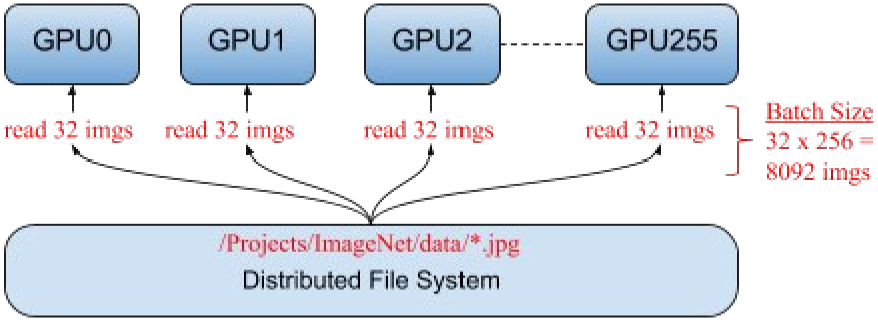
Optimisation des larges batch

Problématiques larges batches ◂

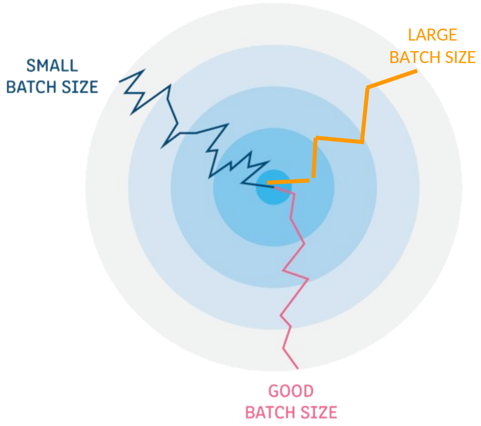
Learning Rate Scaling & Batch Schedulers ◂

Optimiseurs larges batches ◂

Large Batches avec le parallélisme de données



Parallélisme de données : La parallélisation implique une grande taille de *batch*



Problème : Les *batches* trop grands (> 512) ont tendance à engendrer de moins bonnes performances

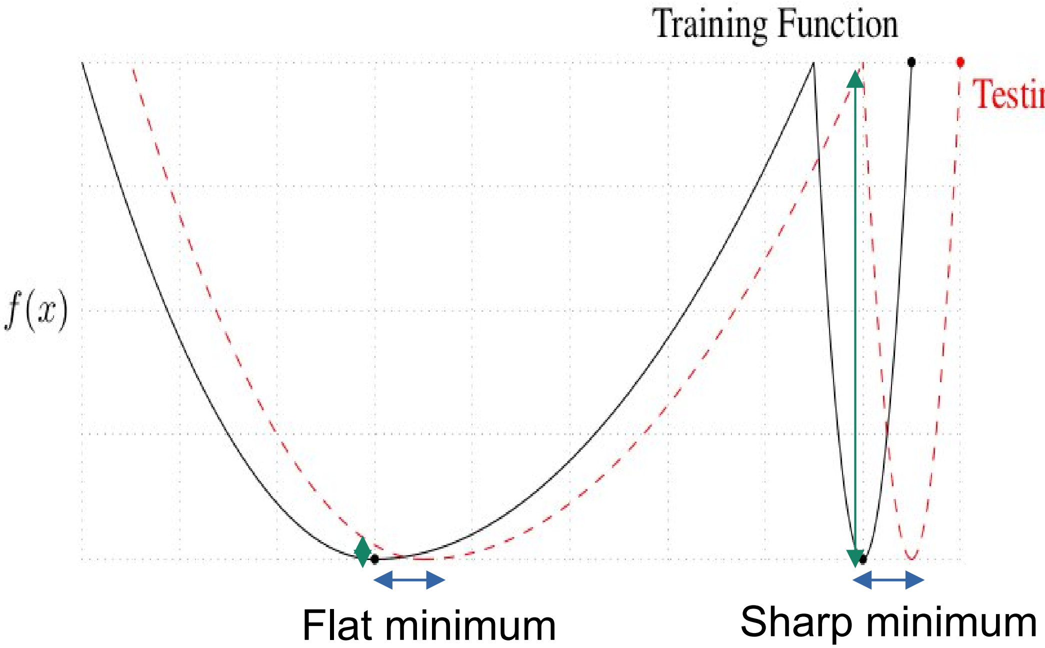
512 1K 2K 4K 8K 16K 32K …

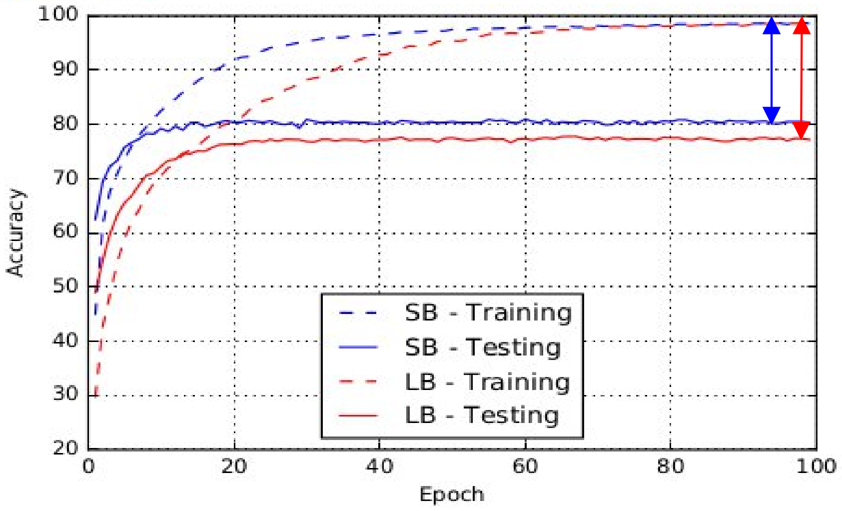
Batch | **Large Batch**

Large Batches

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang





Comparaison d’entraînement d’un réseau convolutionnel avec des petits batch (SB) et large batch (LB) sur CIFAR 10

Plus le *batch* est grand, plus le modèle a tendance à converger vers des minimums pentus et étroits.

Large Batches : Learning rate scaling

Lorsqu'on augmente considérablement la taille du *batch* global, il est souvent nécessaire de mettre le *learning rate* à l'échelle :

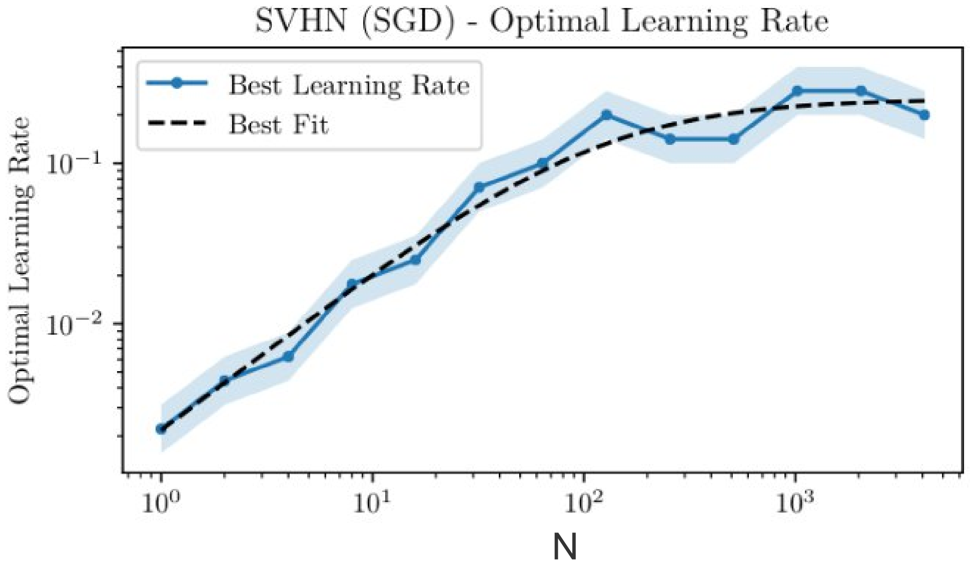
N = Nombre de processus parallèles

Croissance linéaire du *learning rate* :

$$\alpha \rightarrow N * \alpha$$

Croissance en racine carrée du *learning rate* :

$$\alpha \rightarrow \sqrt{N} * \alpha$$



An Empirical Model of Large-Batch Training
Sam McCandlish, Jared Kaplan, Dario Amodei

Optimal : croissance linéaire au début puis en racine carrée (recommandé par OpenAI)

Batch Size Scheduler

=> Alternative au *Learning Rate Scheduler*

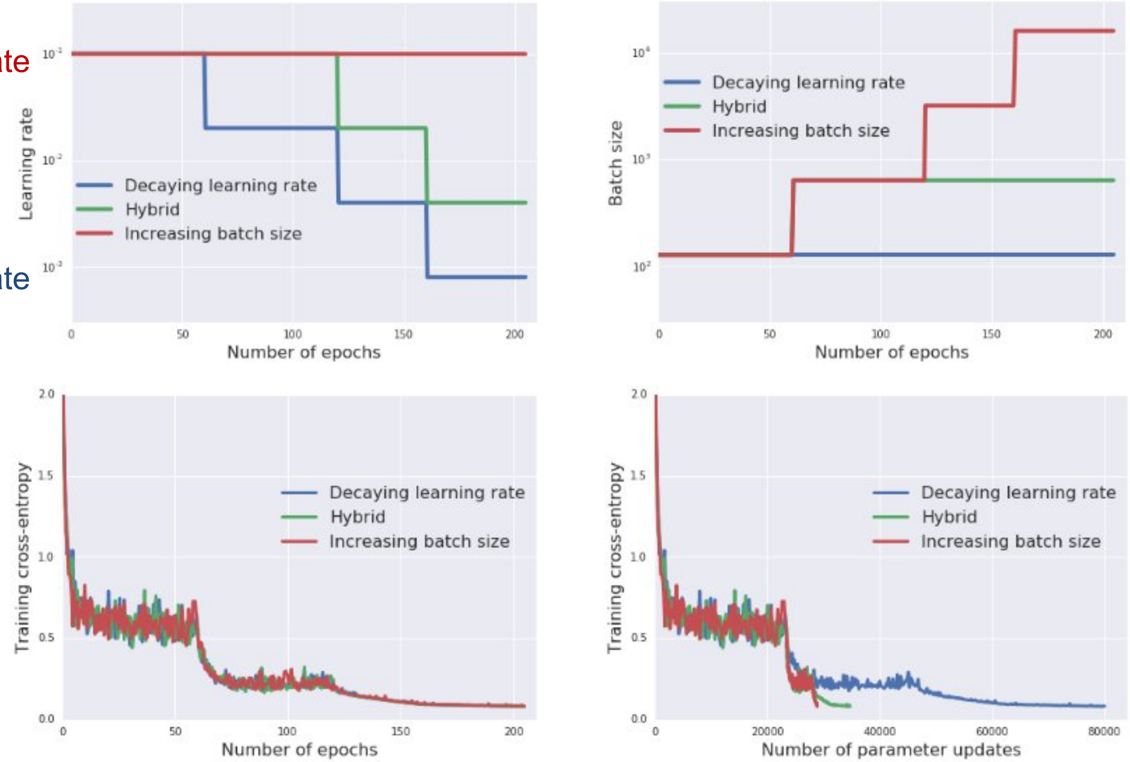
[DON’T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE]

High Learning Rate

Low Learning Rate

Large Batch

Small Batch



Large Batches

Tendances :

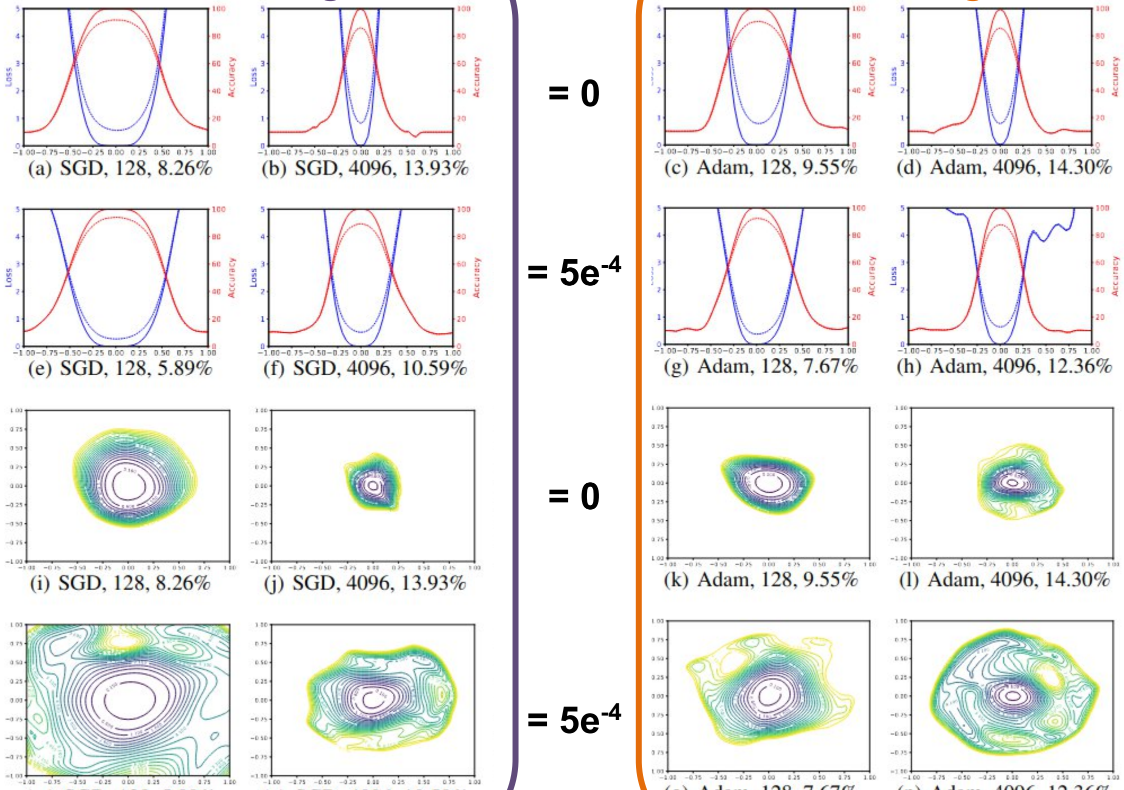
Flat Minimum	Sharp Minimum
- Test Loss	+ Test Loss
Slow Descent	Fast Descent

Small Batch Large Batch

SGD ADAM

Weight Decay Sans W Decay

	SGD		Weight Decay	Adam	
	Small Batch	Large Batch		Small Batch	Large Batch



Weight Decay: = 0, = $5e^{-4}$, = 0, = $5e^{-4}$

(a) SGD, 128, 8.26% (b) SGD, 4096, 13.93% (c) Adam, 128, 9.55% (d) Adam, 4096, 14.30%

(e) SGD, 128, 5.89% (f) SGD, 4096, 10.59% (g) Adam, 128, 7.67% (h) Adam, 4096, 12.36%

(i) SGD, 128, 8.26% (j) SGD, 4096, 13.93% (k) Adam, 128, 9.55% (l) Adam, 4096, 14.30%

(m) SGD, 128, 5.89% (n) SGD, 4096, 10.59% (o) Adam, 128, 7.67% (p) Adam, 4096, 12.36%

Optimiseurs *Large Batches* - LARS

LARS pour “Layer-wise Adaptive Rate Scaling.”

Adaptation de *SGD with momentum* avec l’ajout d’un trust ratio pour chaque couche qui dépend de l ‘évolution du gradient de la couche

r = Trust ratio

l = Numéro de couche

$$m_i = \beta * m_{i-1} + (1-\beta) * (g_i + \underbrace{\lambda\theta_{i-1}}_{\text{Weight decay}})$$

$$r_1 = \left\|\theta^l_{i-1}\right\|_2$$

$$r_2 = \left\|m^l_i\right\|_2$$

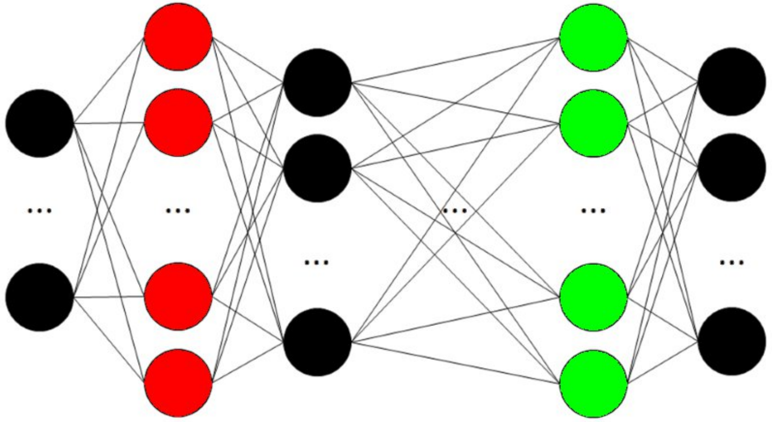
$$r = r_1 / r_2$$

$$\alpha^l = r * \alpha$$

$$\theta^l_i = \theta^l_{i-1} - \alpha^l * m^l_i$$

Peu de confiance
Valeur faible de r

Beaucoup de confiance
Valeur élevée de r



But : Adapter l’importance des mise à jours des poids en fonction d’un *trust ratio* calculé pour chaque couche du réseau.

Optimiseurs *Large Batches* - LAMB

LAMB pour “Layer-wise Adaptive Moments optimizer for Batch training.”

Adaptation de Adam avec l’ajout d’un trust ratio pour chaque couche qui dépend de l ‘évolution du gradient de la couche

r = Trust ratio

l = Numéro de couche

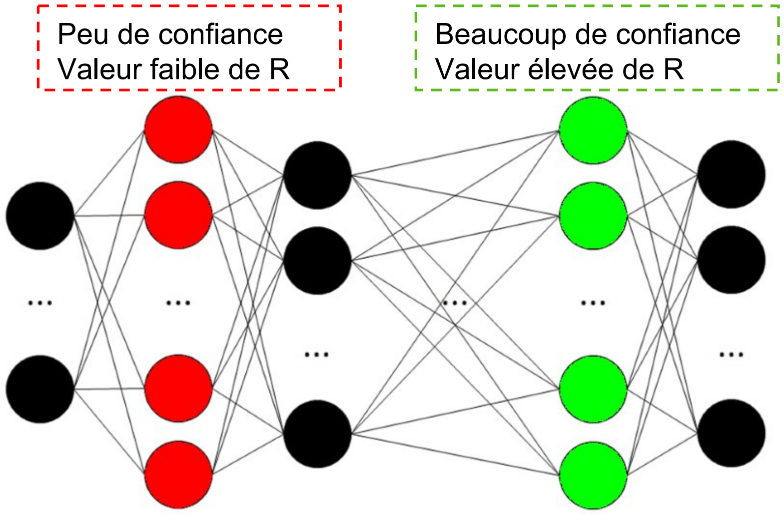
$$r_1 = \| \theta^l_{i-1} \|_2$$

$$r_2 = \| \frac{\hat{m}^l_i}{\sqrt{\hat{v}^l_i + \epsilon}} + \underline{\lambda \theta^l_{i-1}} \|_2$$

$$r = r_1 / r_2$$

$$\alpha^l = r * \alpha$$

$$\theta^l_i = \theta^l_{i-1} - \alpha^l * \left(\frac{\hat{m}^l_i}{\sqrt{\hat{v}^l_i + \epsilon}} + \underbrace{\lambda \theta^l_{i-1}}_{\text{Decoupled weight decay}} \right)$$



But : Adapter l’importance des mise à jours des poids en fonction d’un *trust ratio* calculé pour chaque couche du réseau.

Implémentation des optimiseurs

Chaque optimiseur a ses propres paramètres

SGD

```
import torch.optim as opt

SGD_optimizer = opt.SGD(params, lr, momentum=0, weight_decay=0, nesterov=False, ...)
```

ADAMW

```
import torch.optim as opt

ADAM_optimizer = opt.AdamW(params, lr=0.001, betas=(0.9, 0.999), weight_decay=0.05,..)
```

LAMB

```
from apex.optimizers import FusedLamb

LAMB_optimizer = FusedLamb(params, lr=0.001, betas=(0.9, 0.999), weight_decay=0,
adam_w_mode=True)
```

LARC

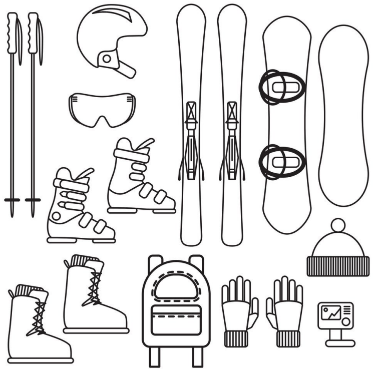
Optimisation apex de LARS

```
import torch.optim as opt
from apex.parallel.LARC import LARC

base_optimizer = opt.SGD(params, lr=0.001, momentum=0.9, weight_decay=0)
optimizer = LARC(base_optimizer)
scheduler = opt.lr_scheduler.CyclicLR(base_optimizer, base_lr=0.01, max_lr=0.1)
```

Large Batches Rider

SGD AdamW

Weight Decay



LARS

LAMB

Batch Scheduler

LR Scheduler

LR scaling

Warmup

LR Decay

BLOOM example

176B params · 59 languages · Open-access

95281 steps (116.8 days)

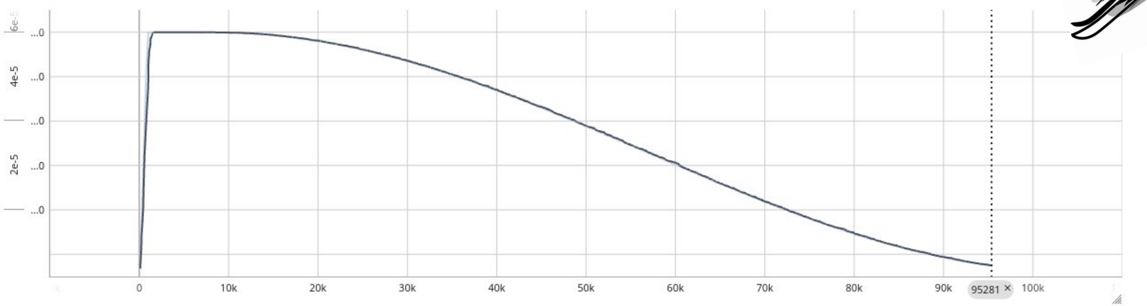
AdamW,
β1=0.9, β2=0.95, eps=1e−8

Weight Decay of 0.1

LR Scheduler
- peak=6e-5
- warmup over 375M tokens
- cosine decay for learning rate down to 10% of its value, over 410B tokens

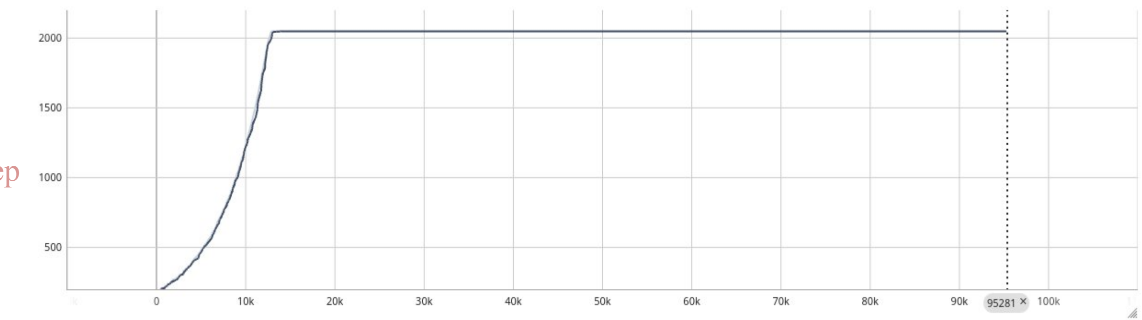
Batch Scheduler
- start from 32k tokens (GBS=16)
- increase linearly to 4.2M tokens/step (GBS=2048) over ~20B tokens
- then continue at 4.2M tokens/step (GBS=2048) for 430B tokens

learning-rate/learning-rate



Run ↑	Smoothed	Value	Step	Relative
tensorboard/main	0	0	95,281	116.8 day

batch-size/batch-size



Run ↑	Smoothed	Value	Step	Relative
tensorboard/main	2,048	2,048	95,281	116.8 day

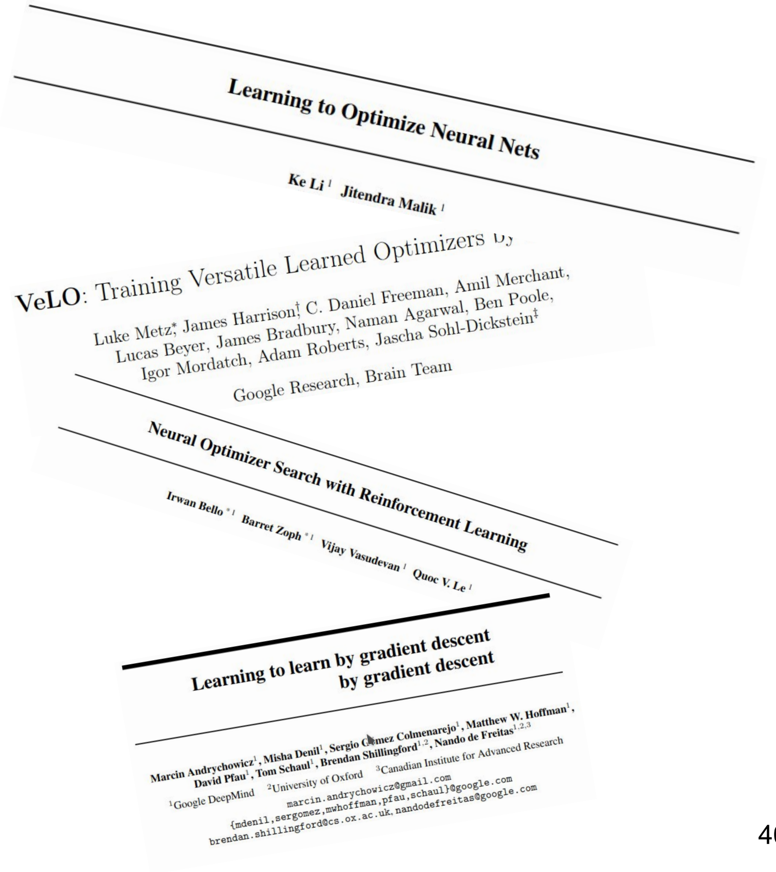
Nouveaux optimiseurs

Nouvelle tendance : l’apprentissage d’optimiseur ◂

Gouffre des nouveaux optimiseurs ◂

LION : exemple d’une nouvelle approche ◂

Nouvelle tendance : l’apprentissage d’optimiseur


Learning to Optimize Neural Nets

Ke Li [1] Jitendra Malik [1]

VeLO: Training Versatile Learned Optimizers by

Luke Metz*, James Harrison†, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agarwal, Ben Poole, Igor Mordatch, Adam Roberts, Jascha Sohl-Dickstein‡

Google Research, Brain Team

Neural Optimizer Search with Reinforcement Learning

Irwan Bello *1 Barret Zoph *1 Vijay Vasudevan [1] Quoc V. Le [1]

Learning to learn by gradient descent by gradient descent

Marcin Andrychowicz[1], Misha Denil[1], Sergio Gómez Colmenarejo[1], Matthew W. Hoffman[1], David Pfau[1], Tom Schaul[1], Brendan Shillingford[1,2], Nando de Freitas[1,2,3]

[1]Google DeepMind [2]University of Oxford [3]Canadian Institute for Advanced Research

marcin.andrychowicz@gmail.com
{mdenil,sergomez,mwhoffman,pfau,schaul}@google.com
brendan.shillingford@cs.ox.ac.uk, nandodefreitas@google.com

Gouffre des nouveaux optimiseurs

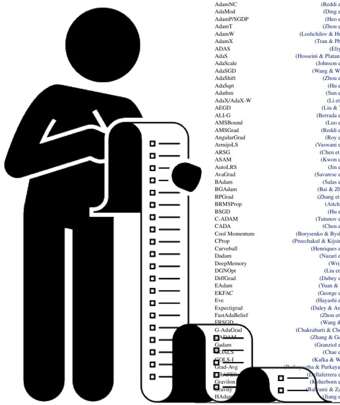
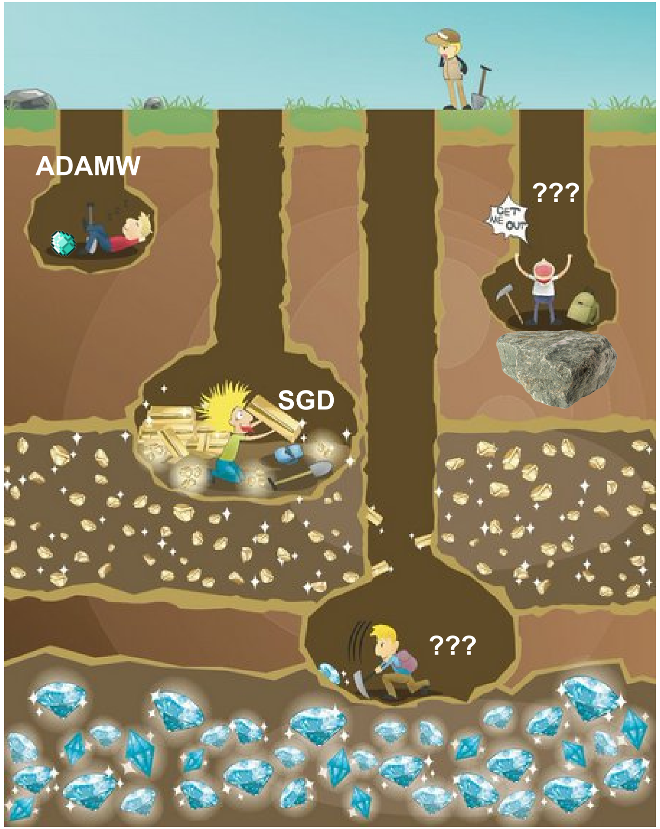


Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam^{+}	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamP/SGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
[illegible]ADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
[illegible]OALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
[illegible]OLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
[illegible]	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
[illegible]ravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAda[illegible]	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)



Schmidt, Robin M., Frank Schneider, and Philipp Hennig. "Descending through a crowded valley-benchmarking deep learning optimizers." *International Conference on Machine Learning*. PMLR, 2021.

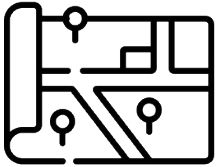
LION : un nouvel optimiseur

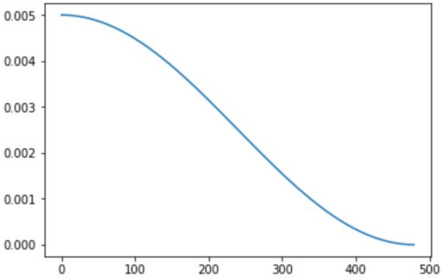
Algorithm 1 AdamW Optimizer

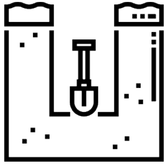
given $\beta_1, \beta_2, \epsilon, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update EMA of g_t **and** g_t^2
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  bias correction
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  update model parameters
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1})$
end while
return θ_t

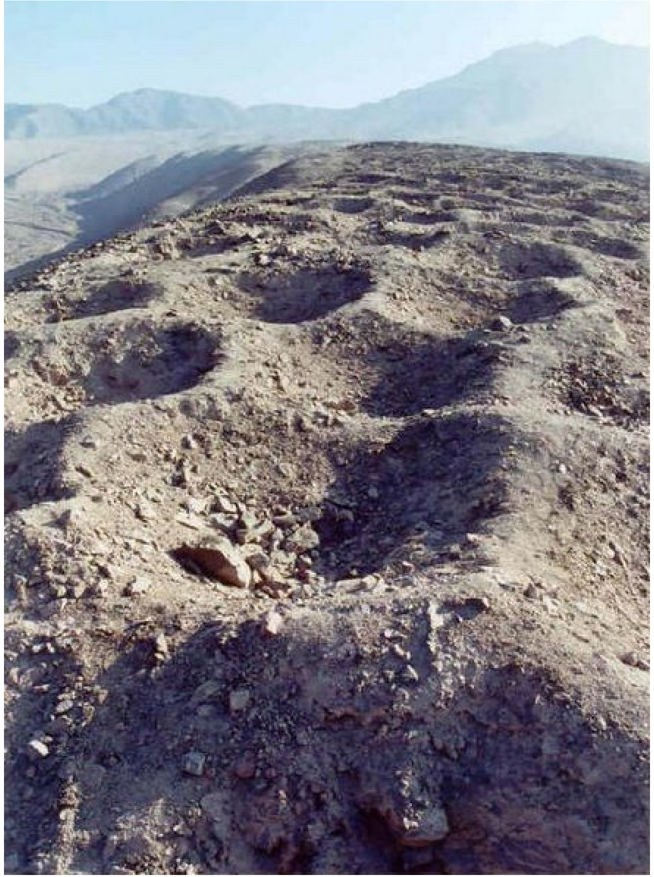
Algorithm 2 Lion Optimizer (ours)

given $\beta_1, \beta_2, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0$
while θ_t not converged **do**
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update model parameters
  $c_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\text{sign}(c_t) + \lambda \theta_{t-1})$
  update EMA of g_t
  $m_t \leftarrow \beta_2 m_{t-1} + (1 - \beta_2) g_t$
end while
return θ_t









Chen, Xiangning, et al. "Symbolic discovery of optimization algorithms." *arXiv preprint arXiv:2302.06675* (2023).

TP : Learning rate + Optimiseurs

Objectifs :
- Modifier le learning rate scheduler
- Modifier l’optimiseur
- Faire des entraînements avec des large batches

Depuis JupyterHub :
– Lancer une instance SLURM CPU
– Allez dans le dossier tp_optimiseurs
– Ouvrez le notebook DLO-JZ_Optimiseurs