SAISON 22/23

Formation

Introduction au Deep Learning

16

Apprendre plus vite
Optimiser l'apprentissage

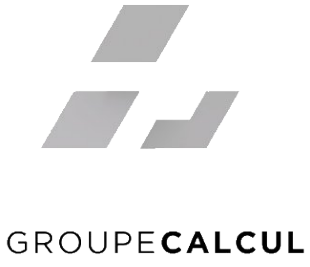
FIDLE

List and Networks

You can also subscribe to :

http://fidle.cnrs.fr/listeinfo
Fidle information list

https://listes.services.cnrs.fr/wws/info/devlog
List of ESR* « Software developers » group



https://listes.math.cnrs.fr/wws/info/calcul
List of ESR* « Calcul » group

(*) ESR is Enseignement Supérieur et Recherche, french universities and public academic research organizations

Ressources

https://fidle.cnrs.fr

Powered by CNRS CRIC, and UGA DGDSI
of Grenoble, Thanks !

Course materials (pdf)

Practical work environment*

Corrected notebooks

Videos (YouTube)

(*) Procedure via Docket or pip
Remember to get the latest version !

Program

20 Séquences
du 17 novembre
au 14 mai 2023

1 ✓
History, Fundamental Concepts
Basic Regression DNN
Basic Classification DNN

2 3 ✓
Hight Dimensionnal Data
CNN

4 ✓
Demystify **mathematics** for neural networks.

5 ✓
Training strategies Evaluation
Sparse data (text) Embedding

6 ✓
Sequences data
RNN

7 ✓
PyTorch
A small detour with **PyTorch**.

8 ✓
«Attention is All You Need»
Transformers

9 ✓ New !
Graph Neural Network
GNN

10 ✓
Autoencoder networks
AE

11 ✓
Variational Antoencoder
VAE

12 JDLS 2023 JDLS 12 mai
Project session
«My project in 180 s»

13 ✓
Generative Adversarial Networks
GAN

14 ✓ New !
Diffusion Model
Text to image

15 ✓
AI, Law, Society and **Ethics**

16 New !
Model and training optimization
Resource efficiency

17
Jean-Zay
GPU acceleration

18
Deep Reinforcement Learning
RL

19 New !
Physics-Informed Neural Networks
PINNS

20
JDLS 2023
Deep Learning for Science !

FIDLE

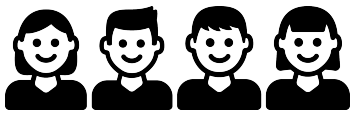
SAISON 22/23

Roadmap

16 New !

Model and training optimization

Resource efficiency



16.1 **Why optimize your training?**

16.2 **Software optimization techniques**

- ➔ Learning Rate & Scheduler
- ➔ Gradient descent and Optimizers
- ➔ Dataset improvement
- ➔ Model & Training tricks

16.3 **A successful training is all you need !**

- ➔ Fine-Tuning Vs Tranfert Learning
- ➔ Hyperparameter Search (HPO)

16.4 **Visualization tools**

- ➔ Tensorboard & Profiler
- ➔ Experiences tracking

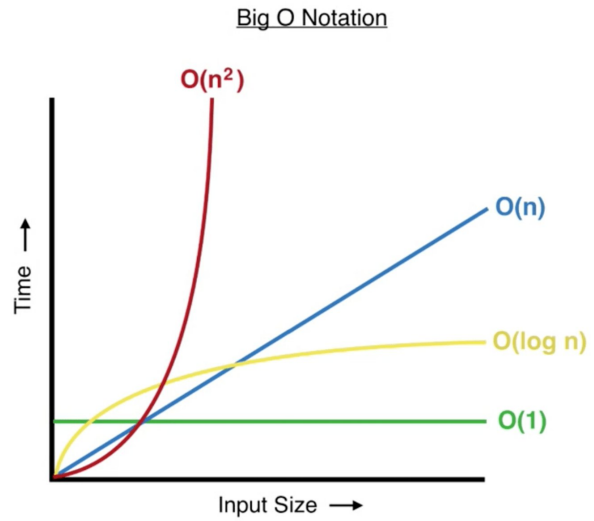
16.5 **Example : Optimize your train**

- ➔ Training Optimization & Tensorboard

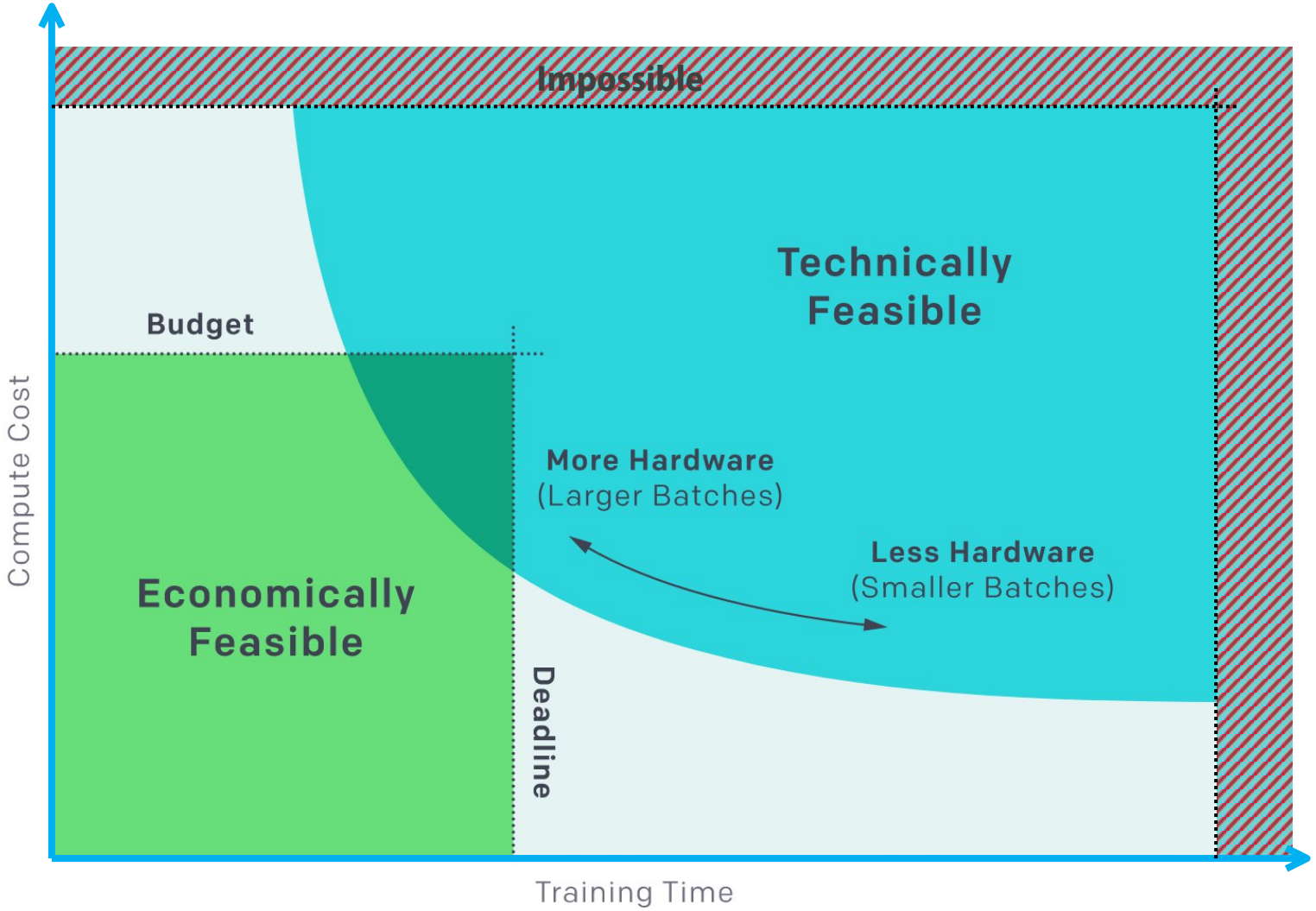
Why optimize your training?

FIDLE

Time, money and feasibility

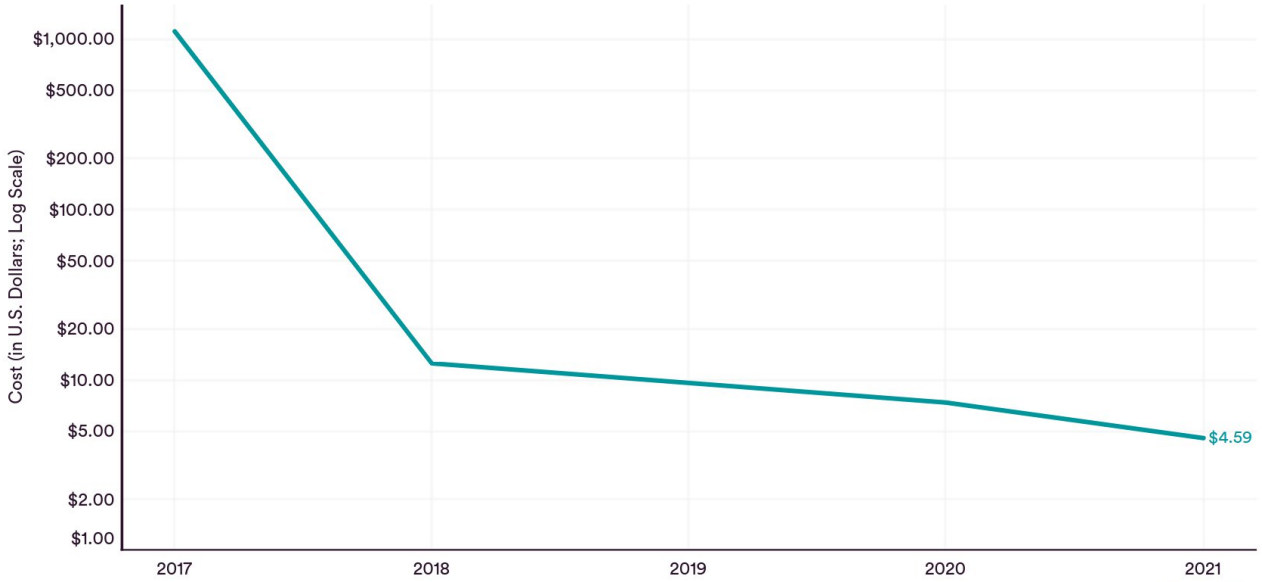


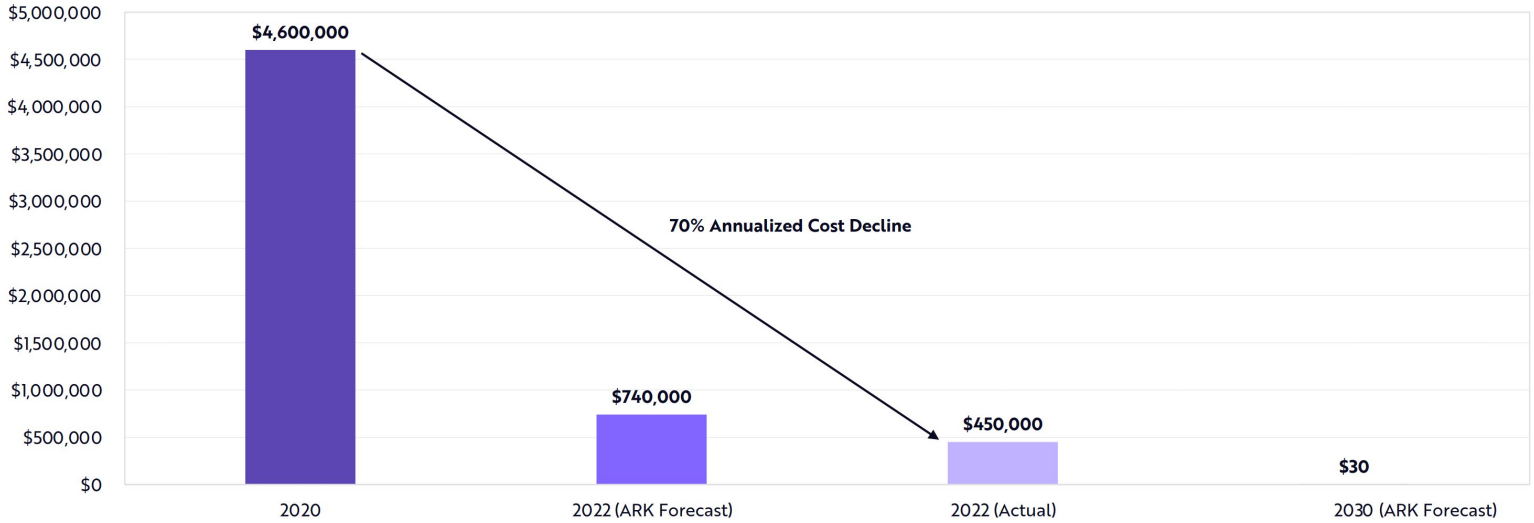
Time, money and feasibility



https://openai.com/blog/science-of-ai/

Same with less

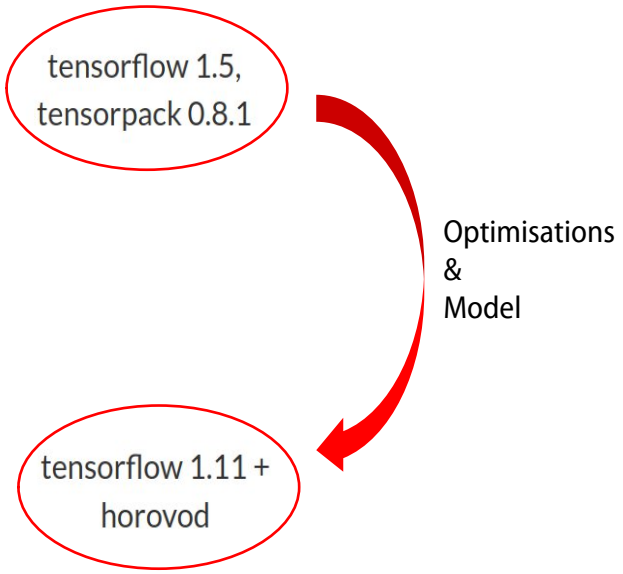




https://ark-invest.com/big-ideas-2023/
https://aiindex.stanford.edu/wp-content/uploads/2022/03/2022-AI-Index-Report_Master.pdf

Better with as much

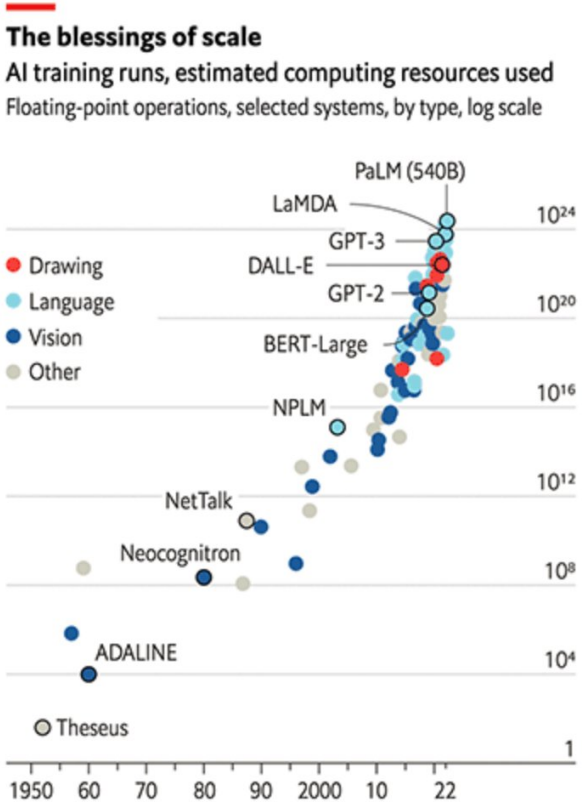
Jan 2018	ResNet50 *DIUX* source	14:37:59	$358.22	93.07%	p3.16xlarge
		÷ 8.4	= $/time	+ 0.17%	= Hardware
Feb 2019	Resnet 50 v1 *GE Healthcare (Min Zhang)* source	1:44:34	$42.66	93.24%	8*V100 (single p3.16xlarge)

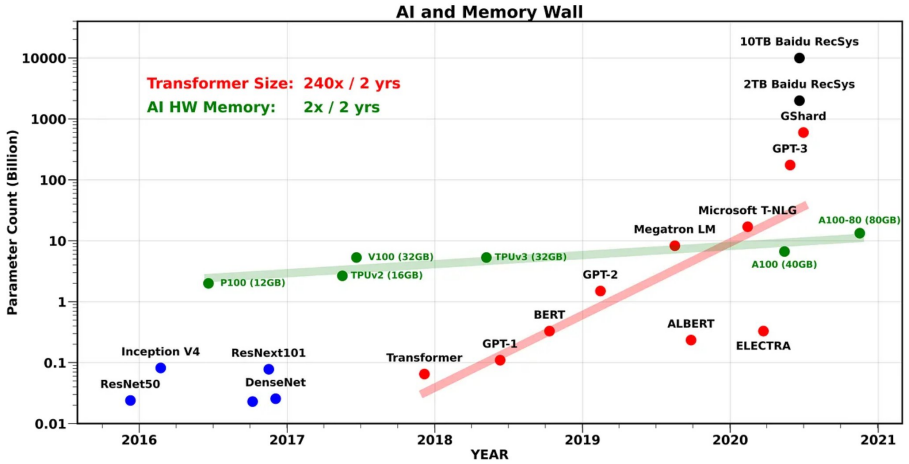


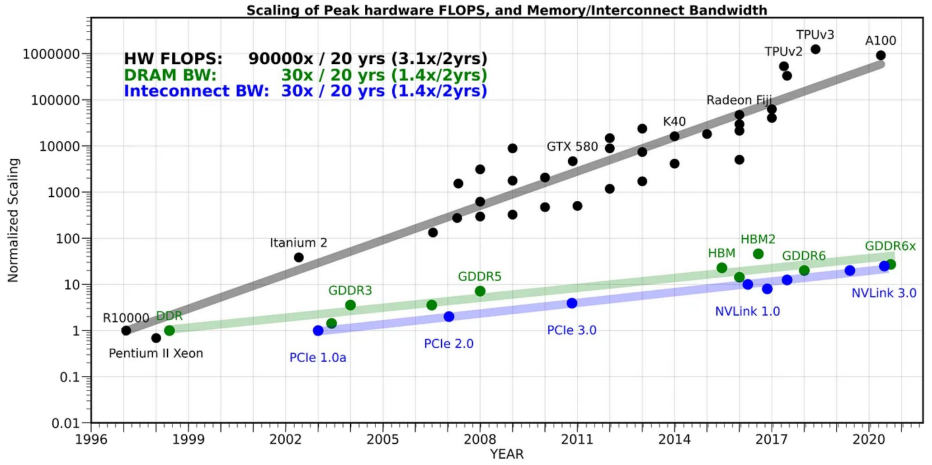
https://dawn.cs.stanford.edu/benchmark/ImageNet/train.html

Better with less

Bigger is Better ?
(snowball effect)

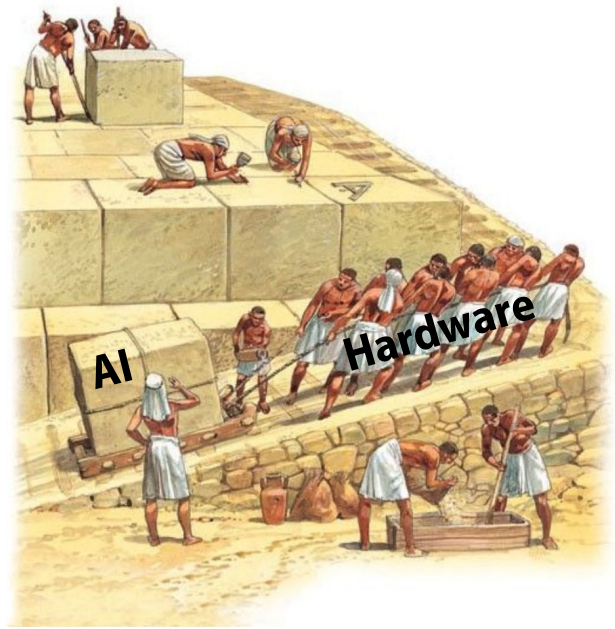






Hardware Wall

AGI



https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

Optimized training => Environment friendly

a BigScience initiative

BLOOM

176B params · 59 languages · Open-access

= **50T CO2 equivalent** =

x50 Paris <-> New York

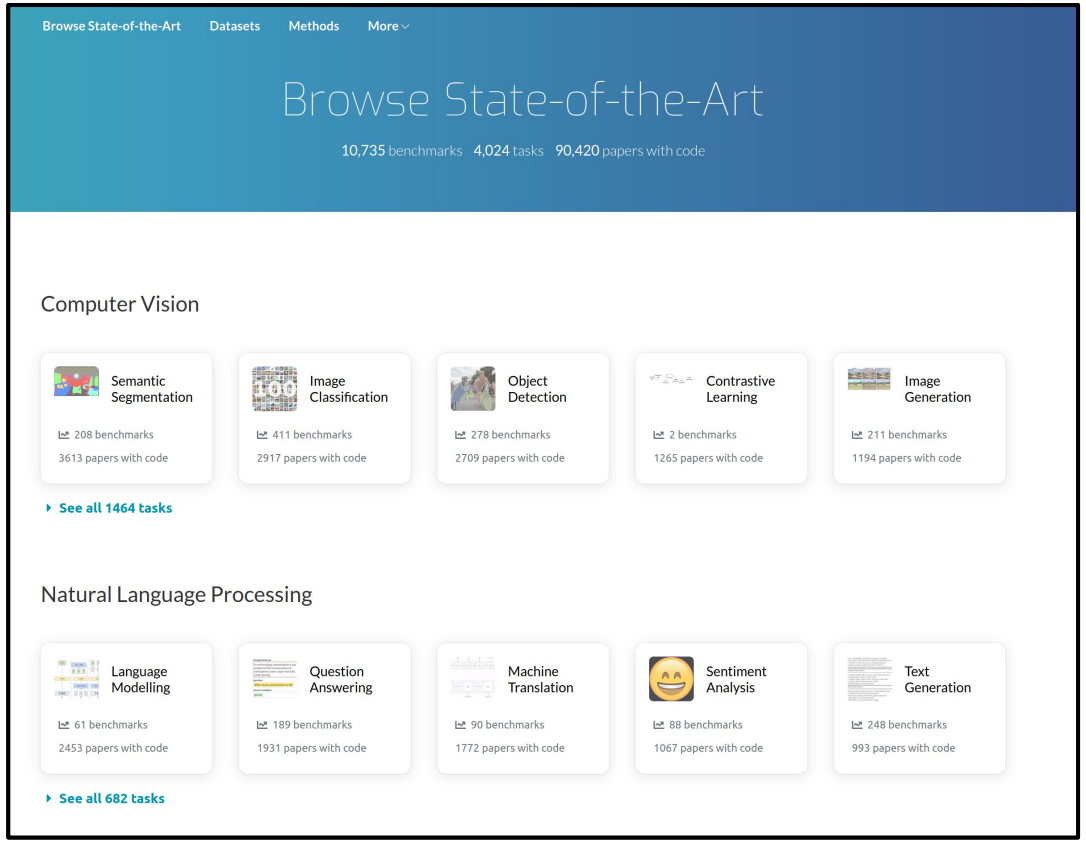
Green AI :

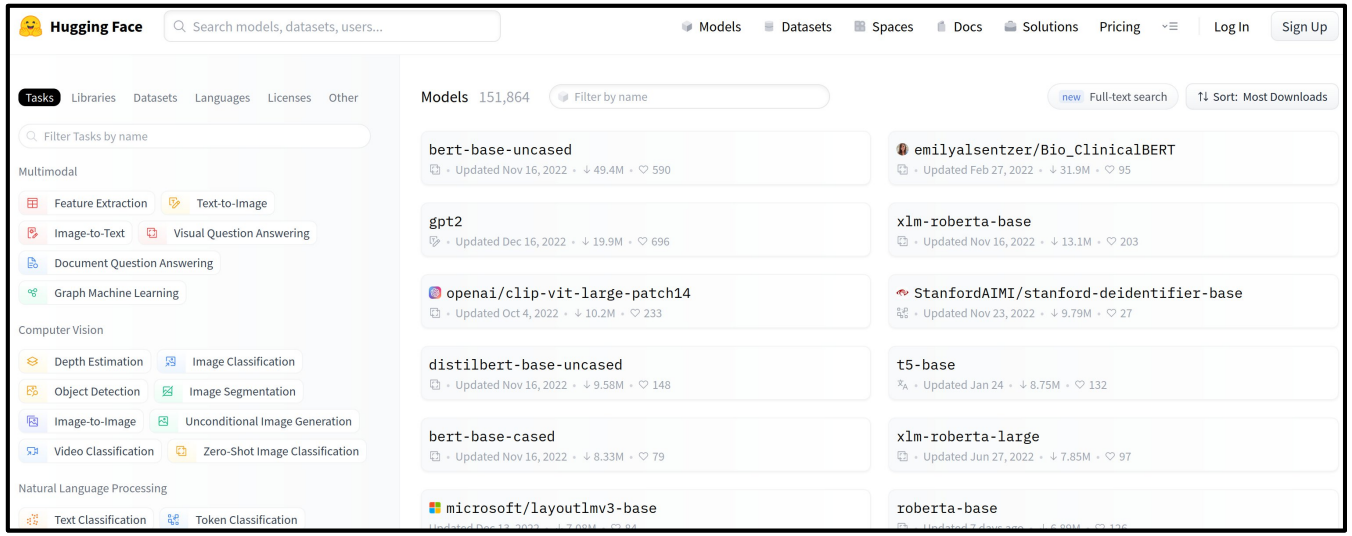
AI model with better results without increasing computational cost, and ideally reducing it.

Red AI :

AI model that results from rapid computational escalation (and thus carbon) costs.

Never reinvent the wheel !





Optimizing AI training made easy: Just don't run it!

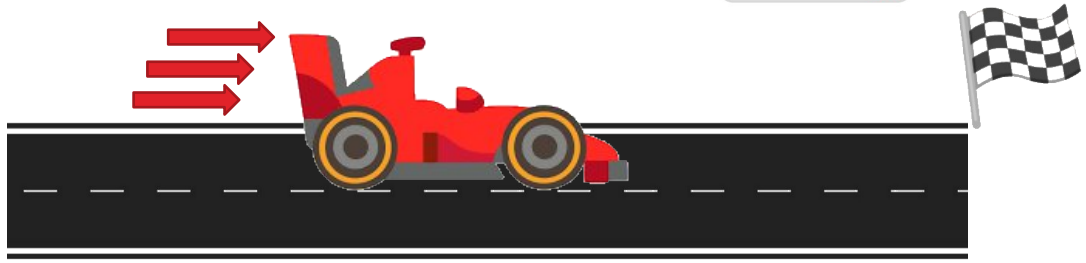
https://arxiv.org/
https://paperswithcode.com/sota
https://huggingface.co/models

2 ways to speed up a AI training

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}$$

Hardware Optimisation :
We increase our speed

17 Jean-Zay
GPU acceleration



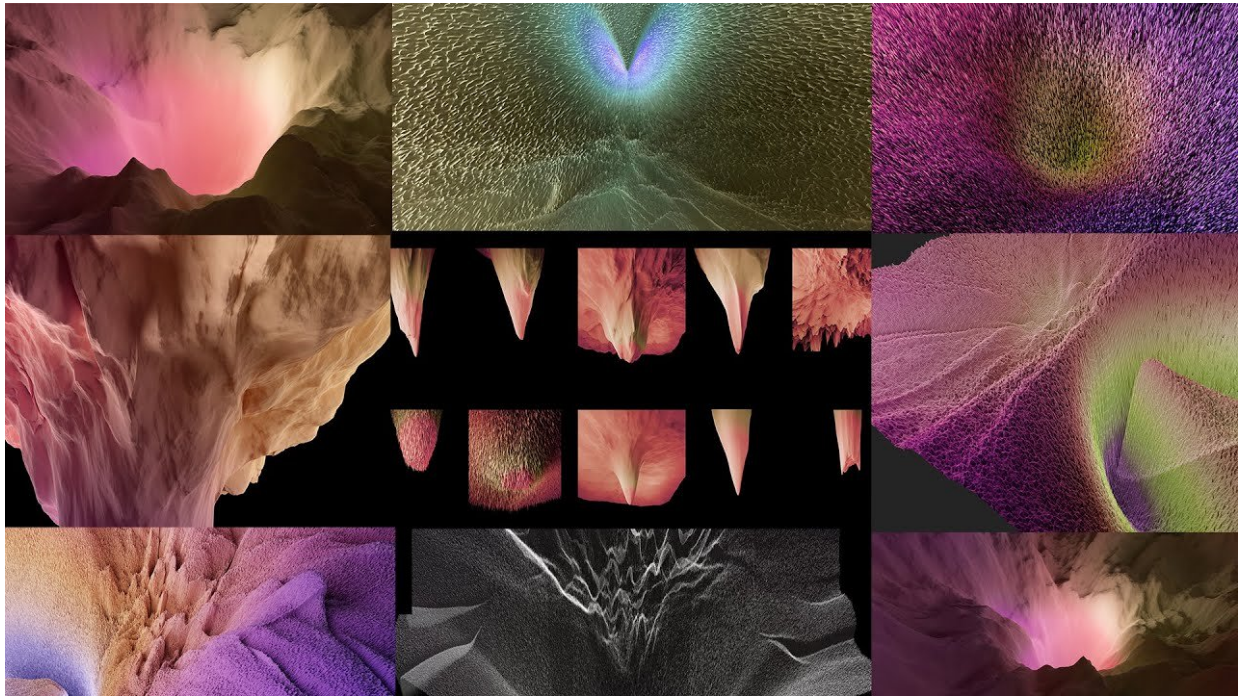
Software Optimisation :
We reduce the distance

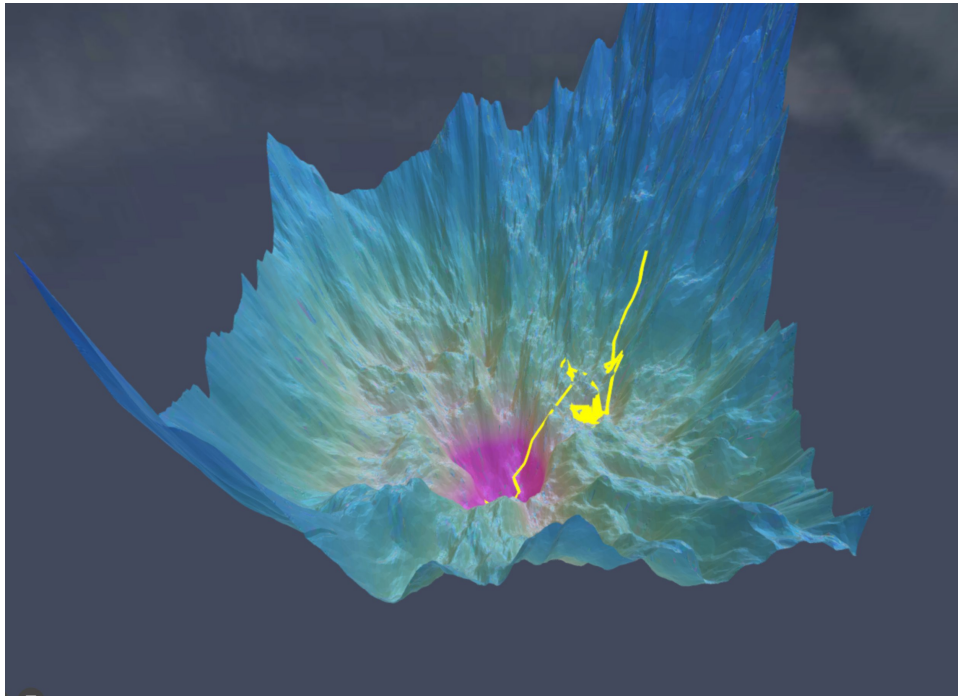
16 New !
Model and training optimization
Resource efficiency

Software optimization techniques

FIDLE

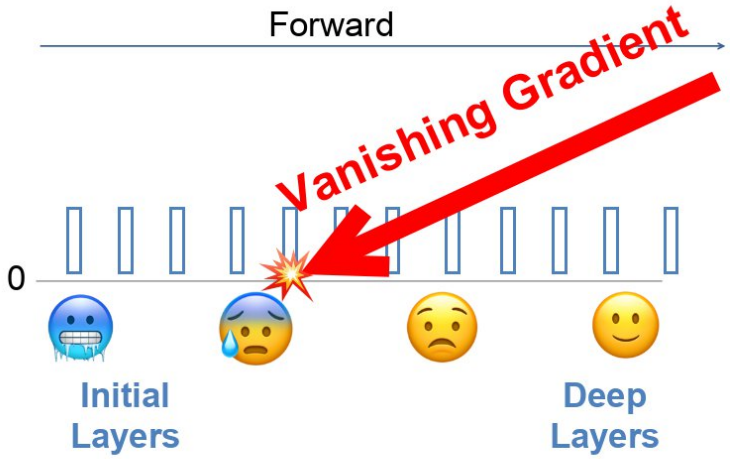
Loss Landscape





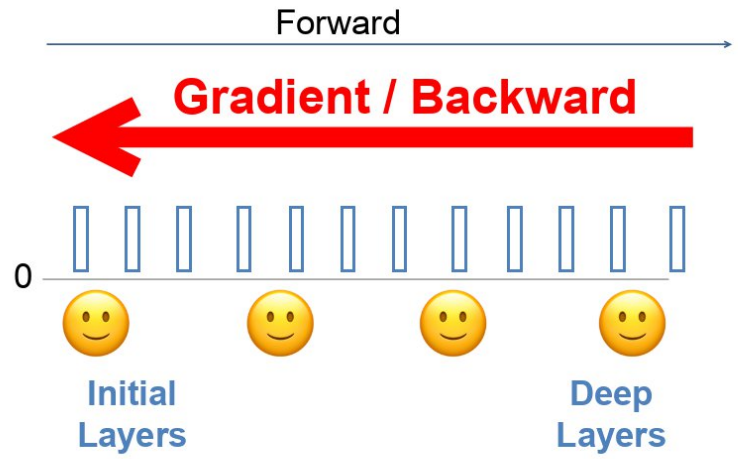
https://losslandscape.com/

Residual Learning

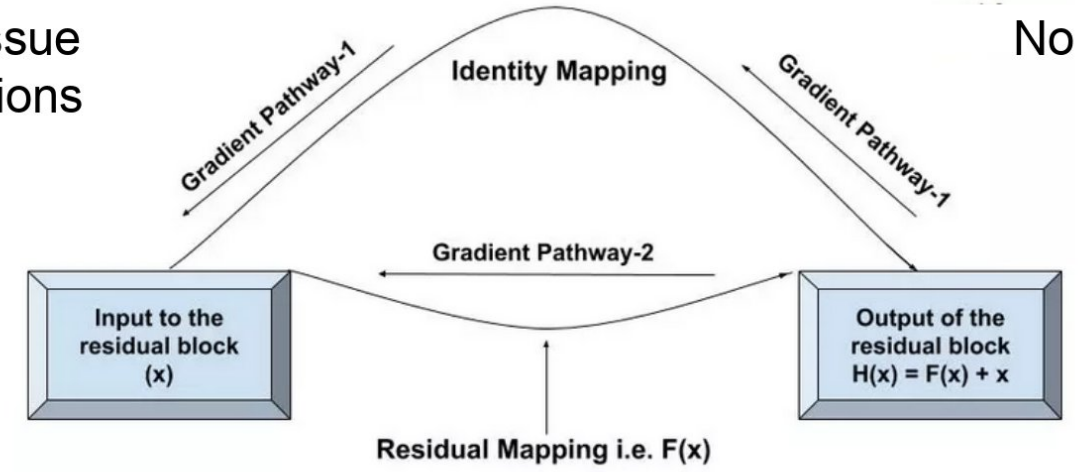


Vanishing Gradient issue **without** skip connections

Residual Block
F(x)+x



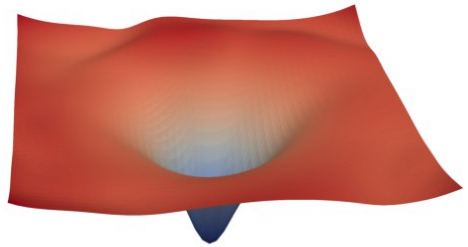
No Vanishing Gradient issue **with** skip connections



Gradient Pathways in ResNet

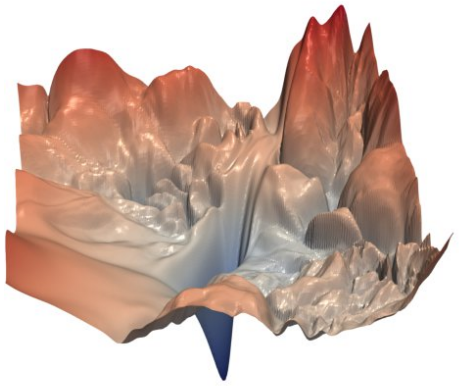
Residual Learning – depth impact

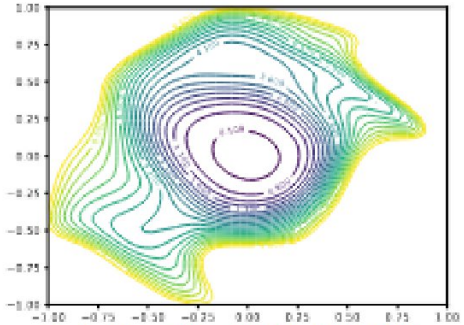
ResNet



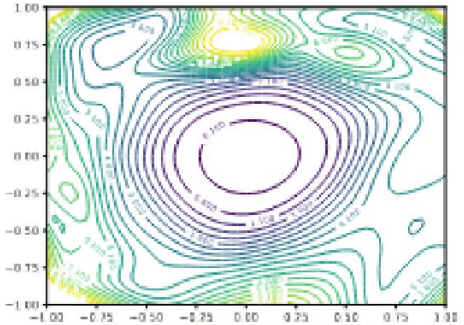
ResNet -
No Short

without skip connections

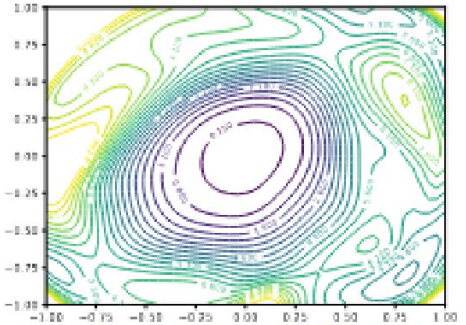




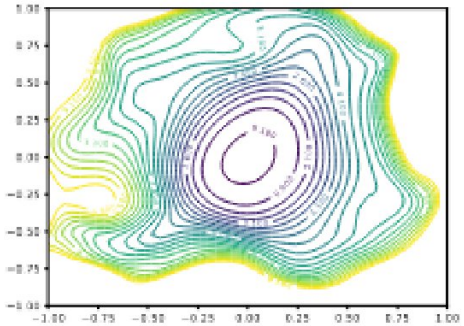
(a) ResNet-20, 7.37%



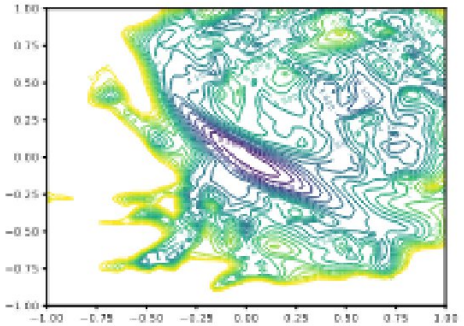
(b) ResNet-56, 5.89%



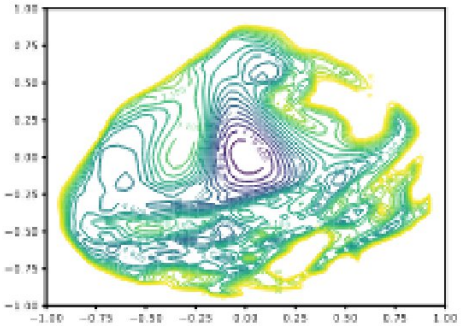
(c) ResNet-110, 5.79%



(d) ResNet-20-NS, 8.18%



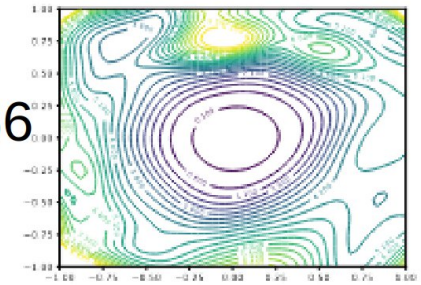
(e) ResNet-56-NS, 13.31%



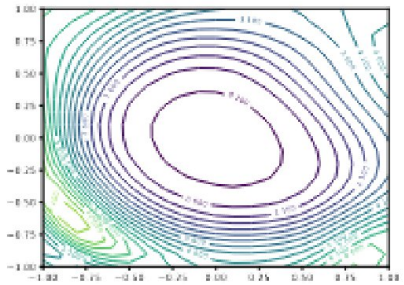
(f) ResNet-110-NS, 16.44%

Residual Learning – width impact

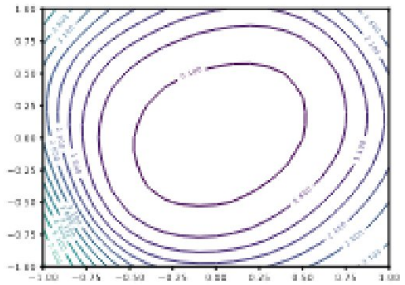
Wide-ResNet-56



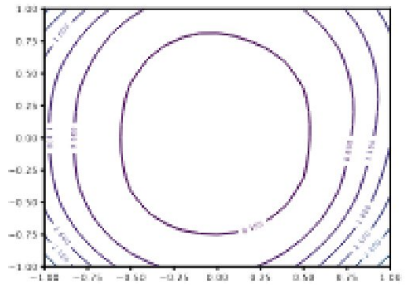
(a) $k = 1$, 5.89%



(b) $k = 2$, 5.07%



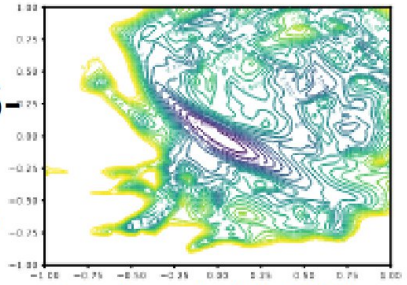
(c) $k = 4$, 4.34%



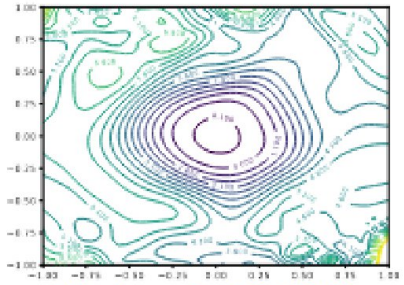
(d) $k = 8$, 3.93%

Wide-ResNet-56-No Short

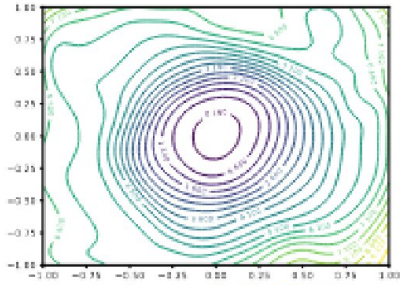
without skip connections



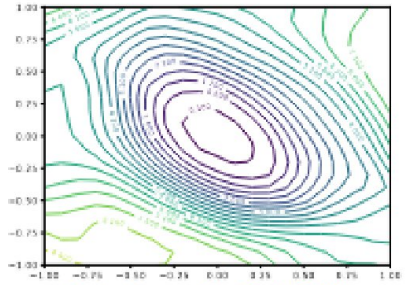
(e) $k = 1$, 13.31%



(f) $k = 2$, 10.26%



(g) $k = 4$, 9.69%



(h) $k = 8$, 8.70%

k = width coefficient of the channels compared to ResNet

Model Parameters Initialization

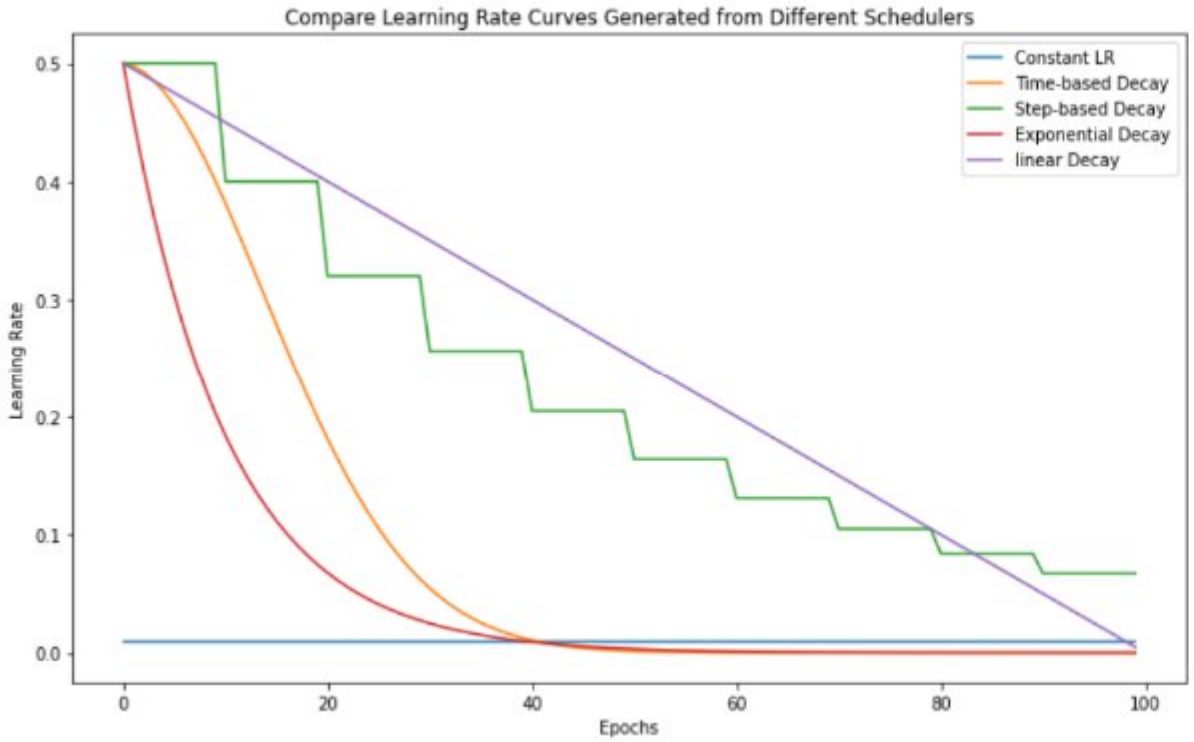
The Blessing of Dimensionality :

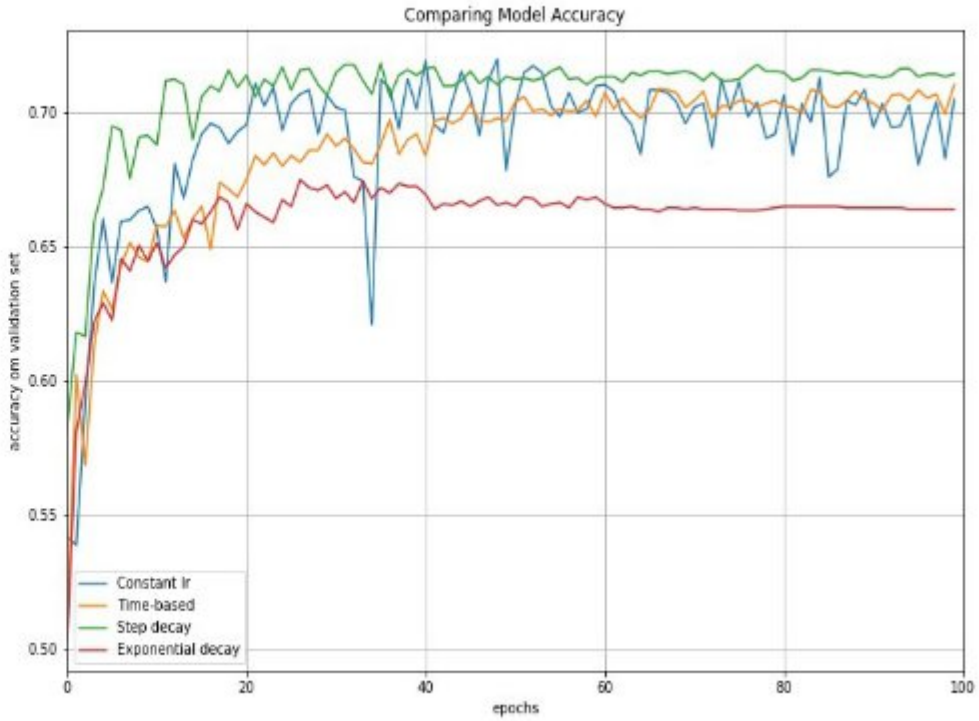
- Xavier Initialization
  - uniform
  - normal
- Kaiming Initialization
  - uniform
  - normal

By default in PyTorch:

- Best initialization algorithm depending on the type of layer (linear, convolutional, transform, ...).
- Today, it is no longer necessary to try to optimize the initialization.

Learning rate decay

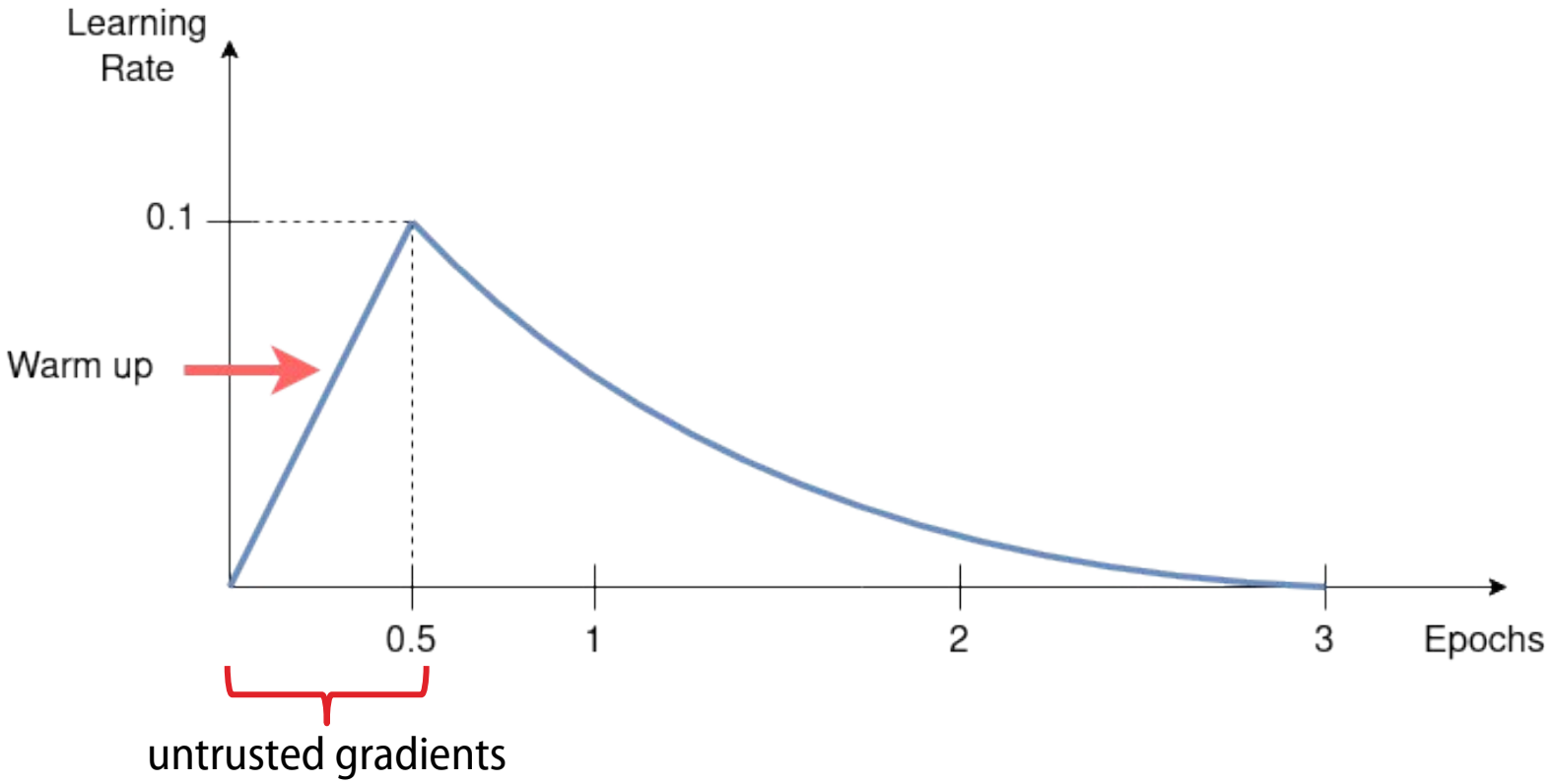




Learning rate scheduler

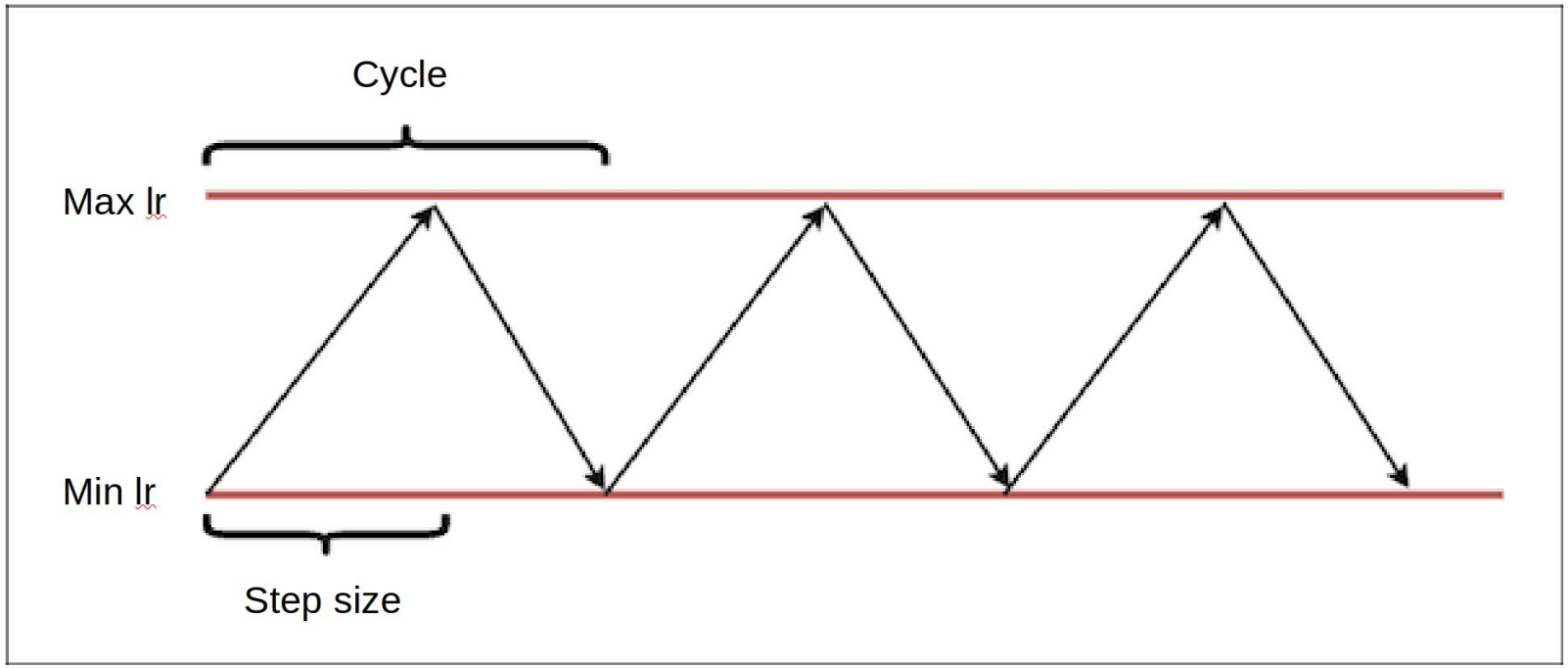
Problem: **Early iterations** have too much effect on the model (large loss, high gradients, bias, …), **high learning rate can cause high instability or divergence**

LR Scheduler Goal: **gradually increase the learning rate** to avoid the risk of divergence at the start of learning



Learning rate (constant)

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

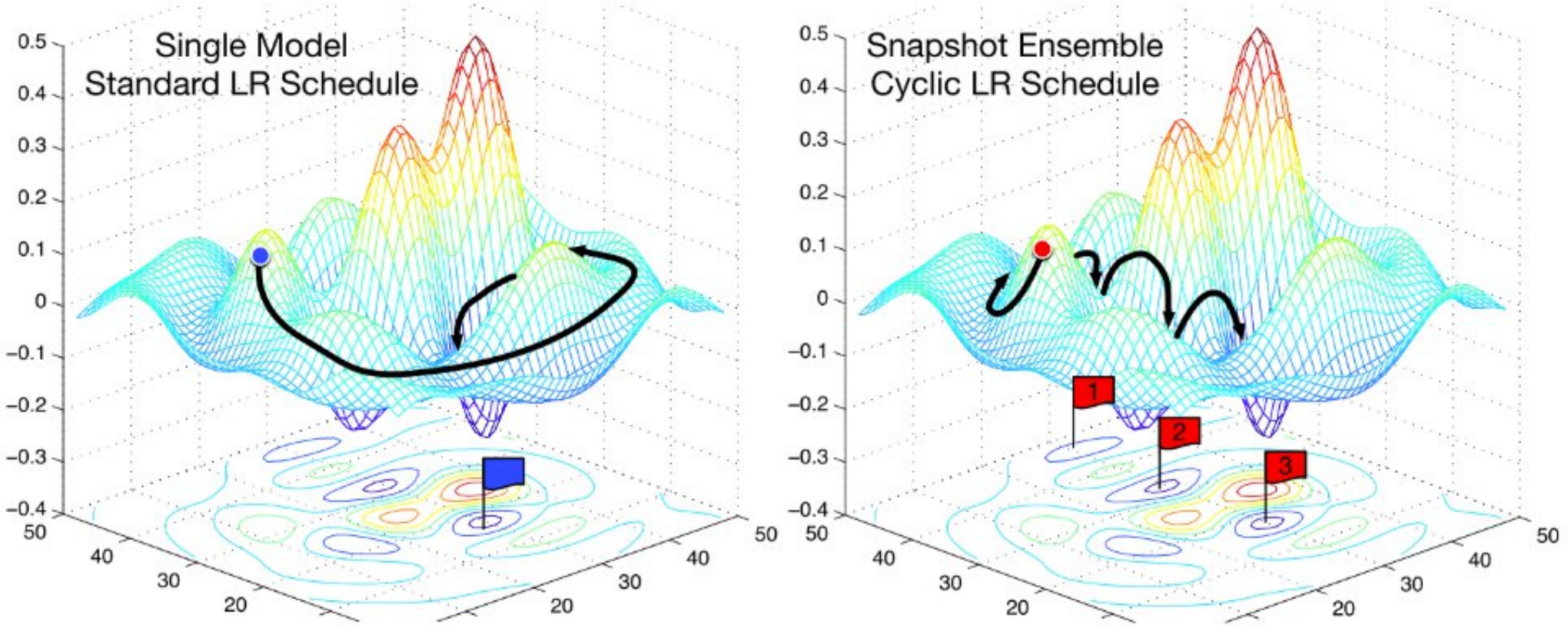


Parameters :

- Step_size : x *epoch($2 \leq x \leq 10$)
- Base_lr = > min convergence value
- max_lr => max value before divergence

Succesions of warmups
and learning rate decays

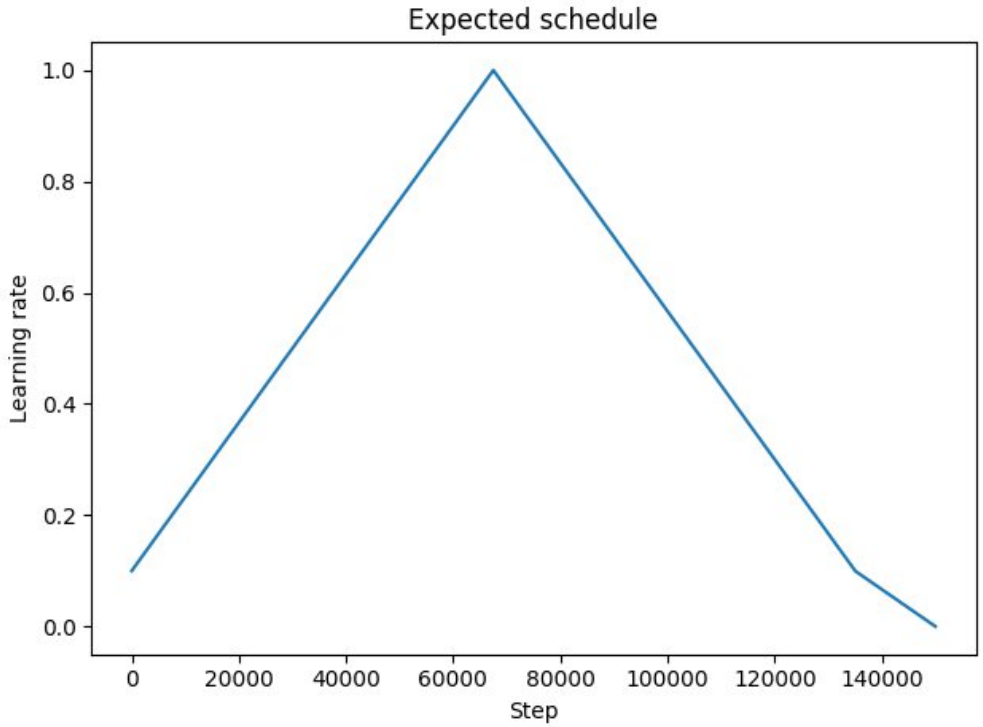
Learning rate

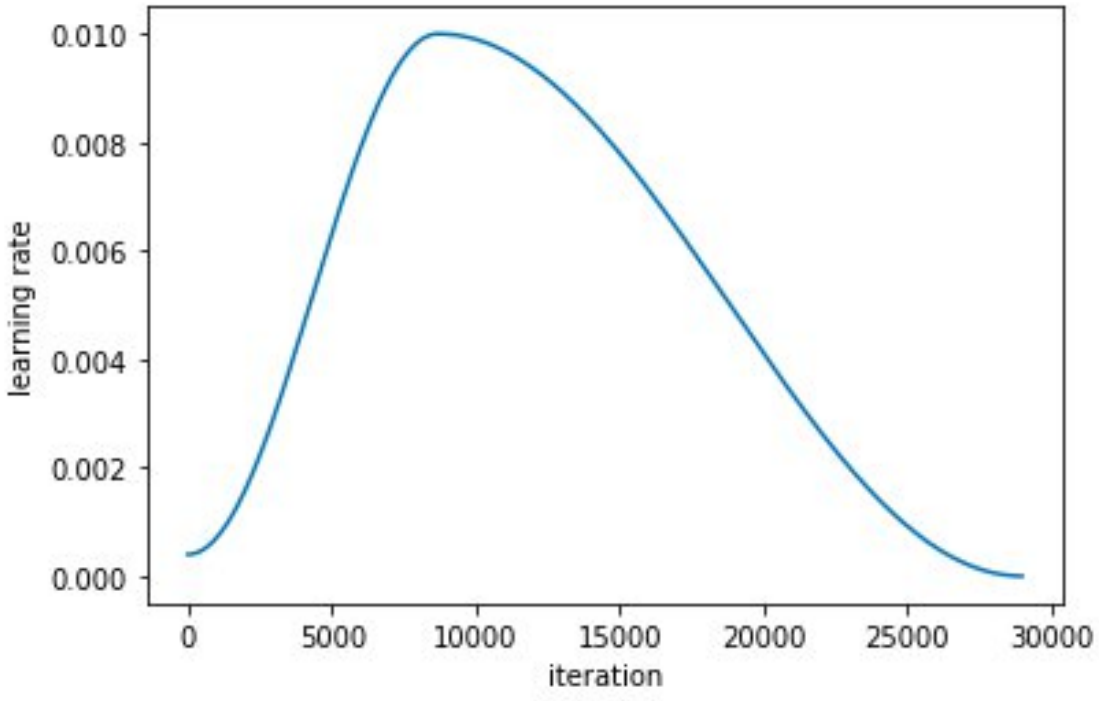


SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE

Gao Huang, Yixuan Li, Geoff Pleiss

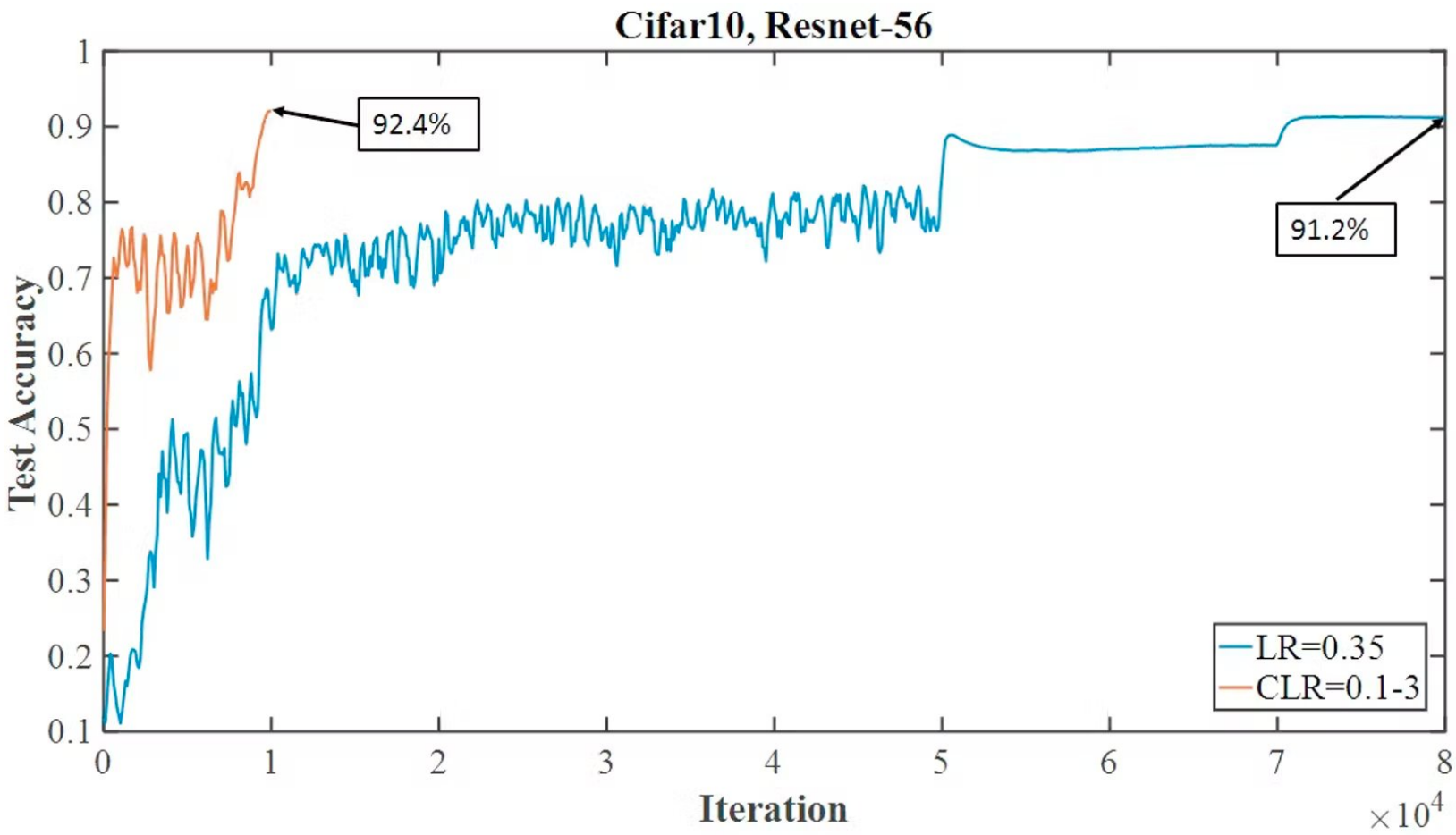
Learning rate



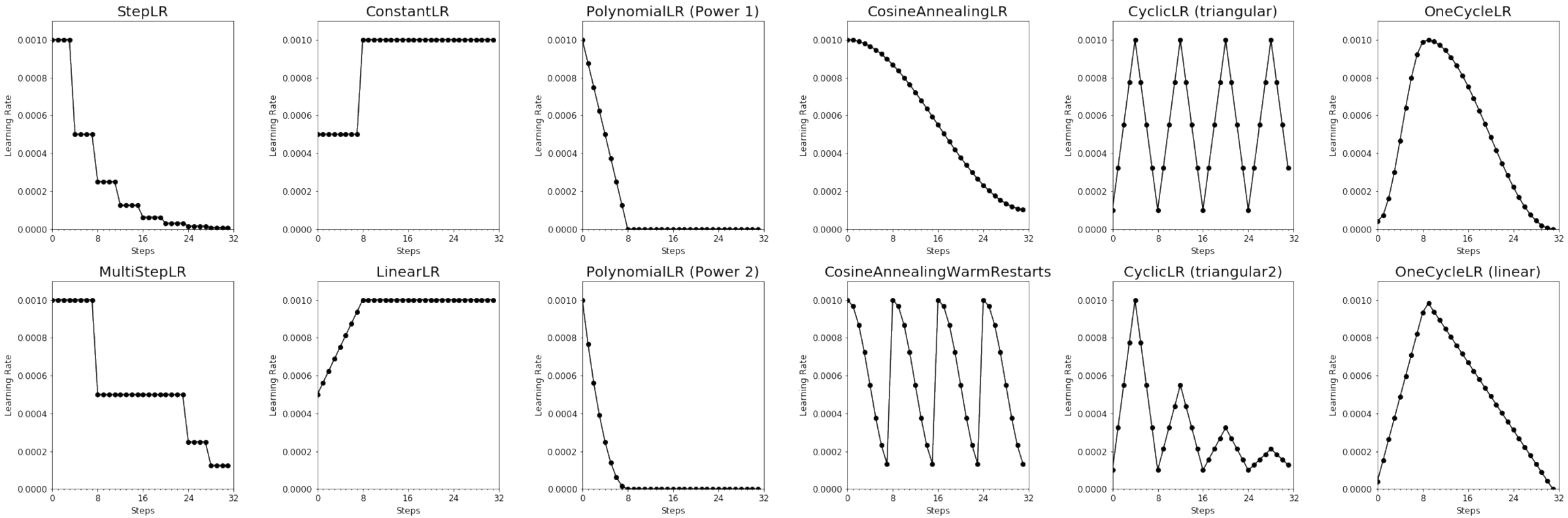


A disciplined approach to neural netwok hyper-parameters, Leslie N. Smith

One Cycle Learning Rate - Super convergence



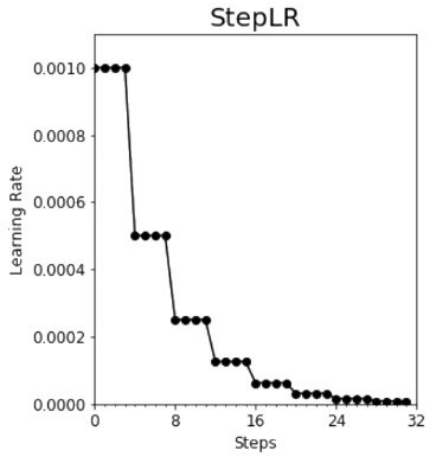
Learning rate



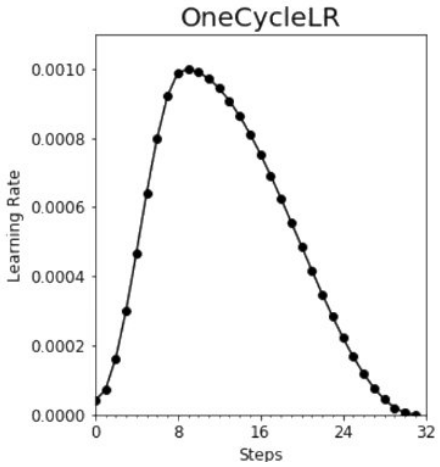
https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863#eec7

Learning rate

```
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()
```

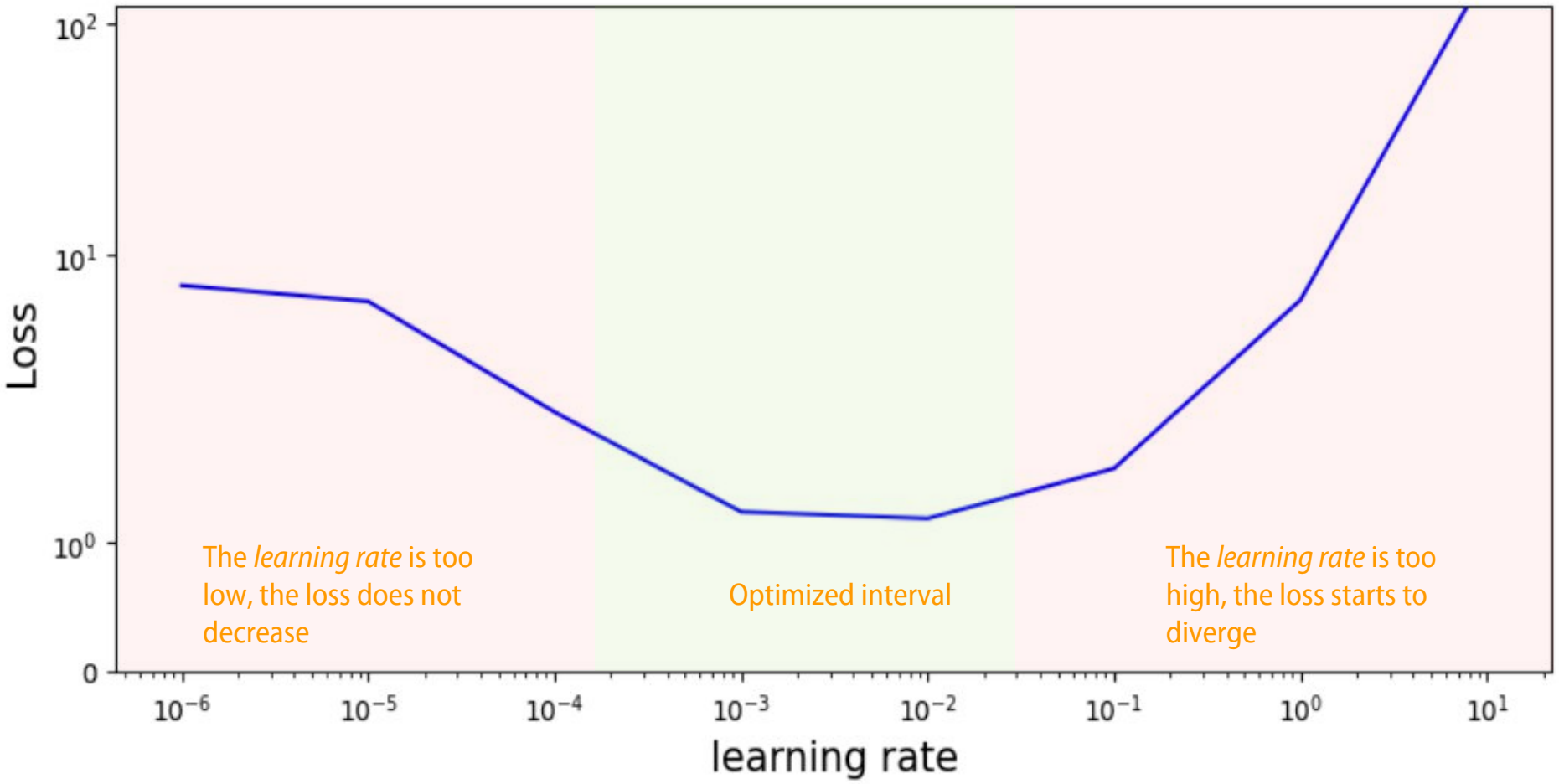



```
import torch.optim as opt

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()
```



Learning rate finder

Goal: Find the optimal learning rate values for his model, especially for the maximum value of a cyclic scheduler



Run your model over a few step/epochs by increasing its learning rate (with/without model reset)

- Start of decline in loss

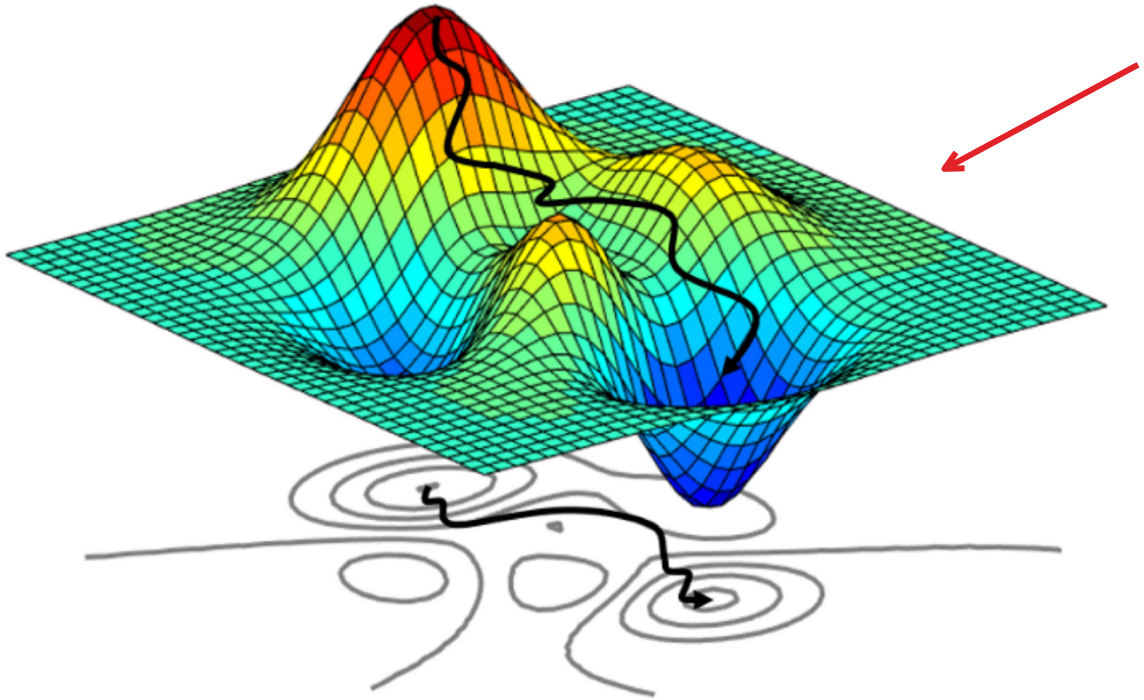
→ **min learning rate**

- Start of loss variation

→ **max learning rate**

Questions Break

Optimizers -SGD



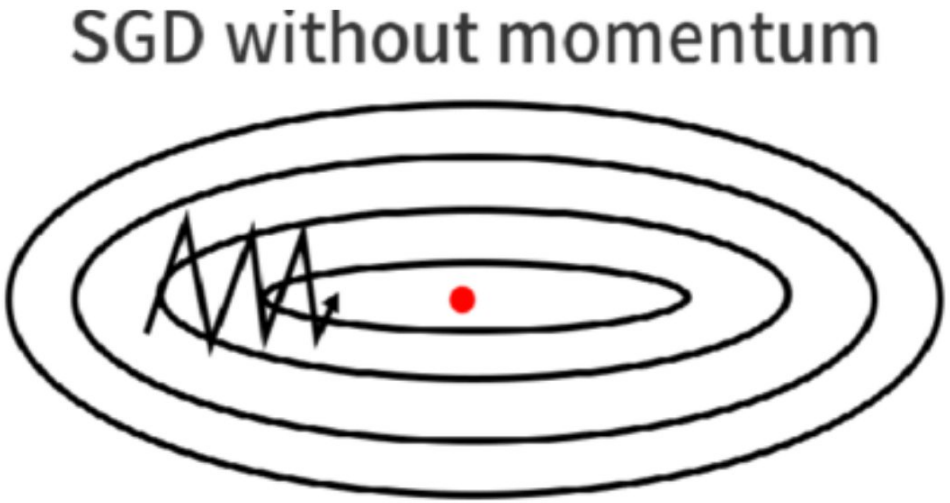
The optimizer is the algorithm that drives the gradient descent and the search for minimum with the aim of optimizing the learning time and the final metric.

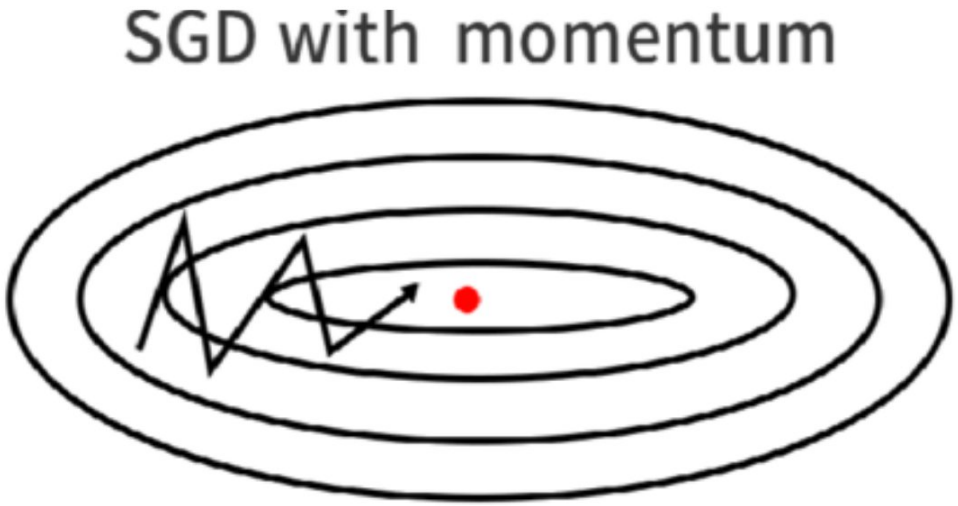
Batch size and learning rate adaptable to conflicting needs:

- Exploration to find the best local minimum
- Gradient descent acceleration

SGD = Stochastic Gradient Descent
Gradient calculation and weight update **at each batch**

SGD with Momentum





$$m_0 = 0$$

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

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

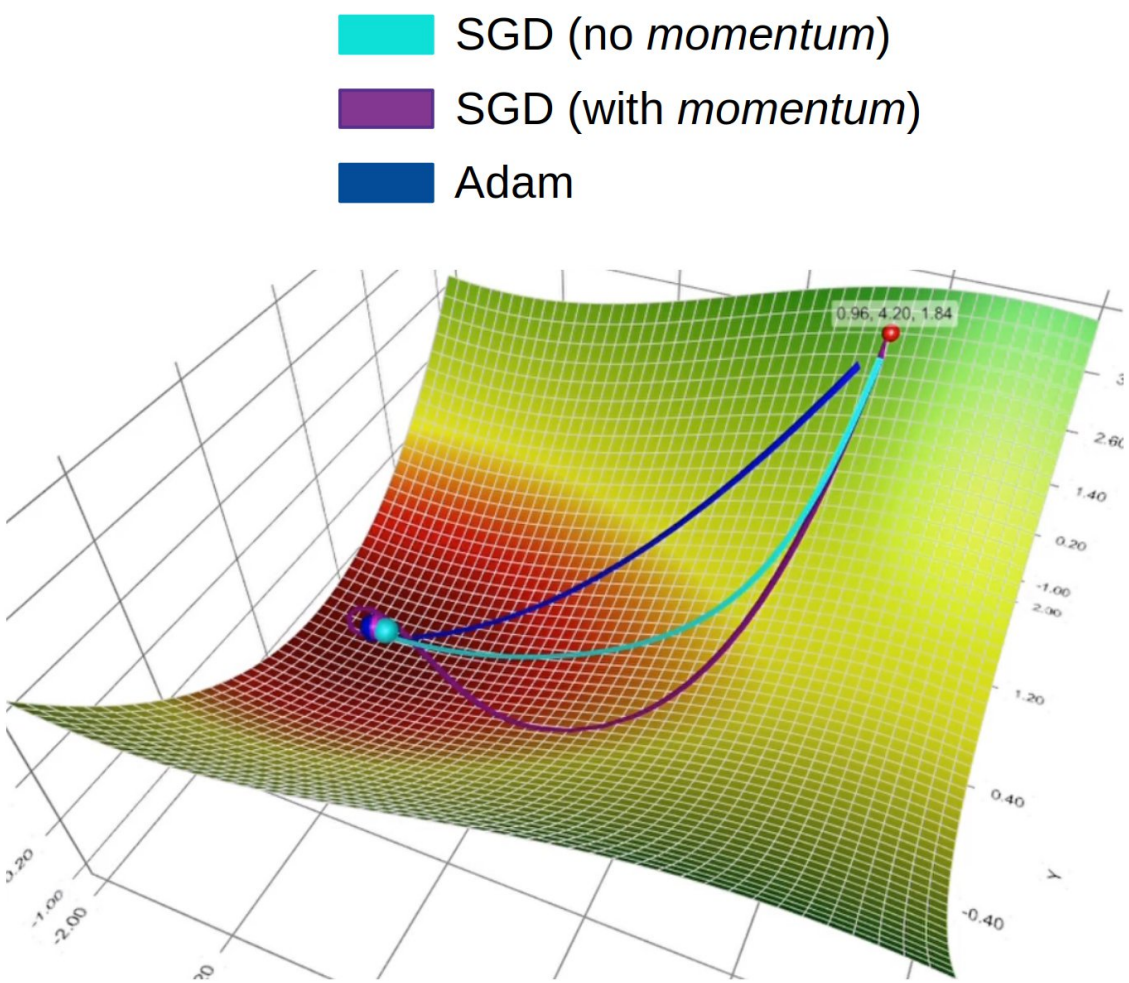
Momentum Factor

Goal: Consider previous gradients for faster gradient descent.

$$0.85 < \beta < 0.95$$

+ Allows you to converge faster
- No guarantee that the momentum will take us in the right direction

Adaptive Optimizers



Rather than driving the gradient descent manually with the learning rate...

We can adapt the learning rate for **each weight** of the model according to the gradient, the gradient2, or the norm of the weights of the layer!!

Examples :

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

Adam

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

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

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

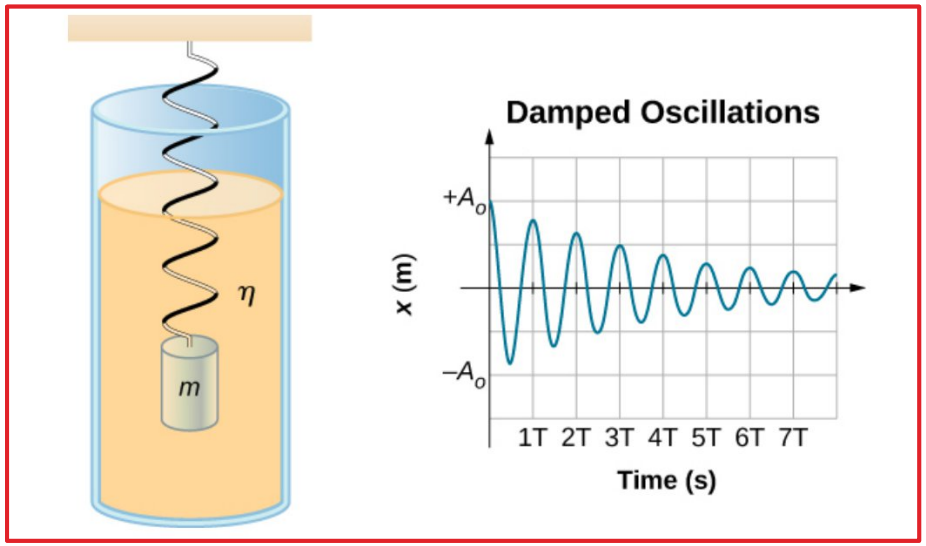
$$\hat{v}_i = \frac{v_i}{1 - \beta_2^i}$$

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

Correction of the biases of the first iterations

First moment: moving average

Second moment: sliding uncentered variance

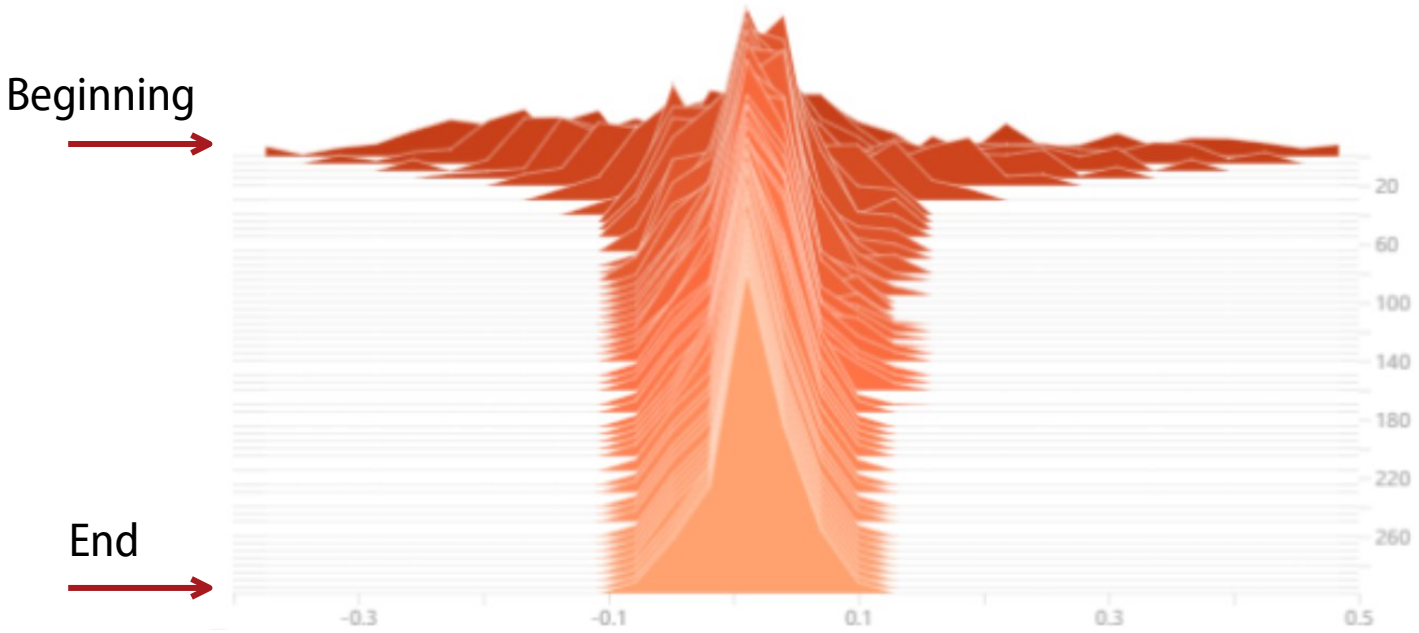


Parameters :

β1 & β2 = Regression rate (β1 = 0.9 & β2 = 0.999)

ϵ — Very small value to avoid division by zero

Weight decay



Weight distribution during learning

A neural network that converges and generalizes correctly (neither underfitting nor overfitting) generally has weights that tend to the same value.

Weight decay and decoupled weight decay

ADAM

For i = 1 to ...

$$g_i = \nabla_\theta f_i(\theta_{i-1}) + \lambda\theta_{i-1}$$

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

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

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

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

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

Return θ_i

Weight decay

ADAMW

For i = 1 to ...

$$g_i = \nabla_\theta f_i(\theta_{i-1})$$

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

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

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

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

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

Return θ_i

Decoupled weight decay

Evolution of weight decay: Decoupled weight decay (decoupled from momentum!!)

- SGD and Adam with the weight decay
- SGDW and AdamW with decoupled weight decay

SGD and SGDW are roughly equivalent in performance.

However AdamW is noticeably better than Adam!!

Weight decay and decoupled weight decay

SGD

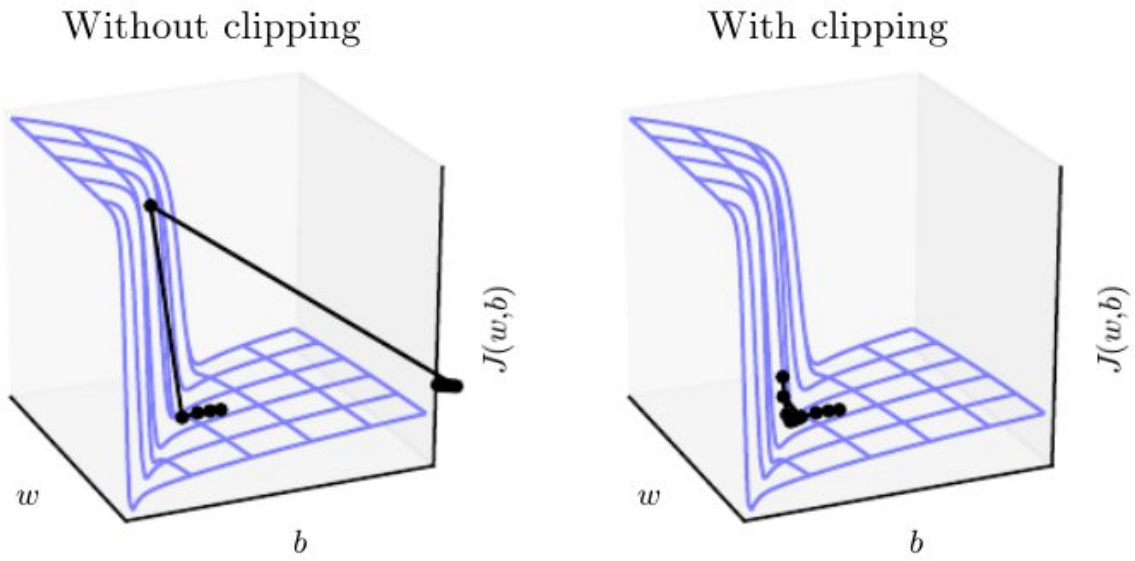
```
import torch.optim as opt

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,..)
```

Questions Break

Gradient clipping

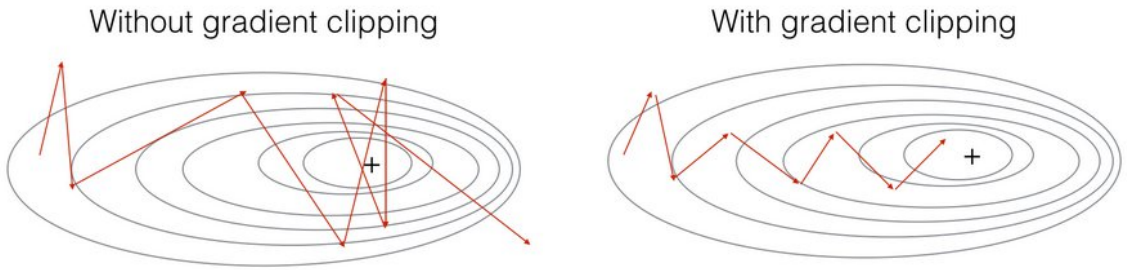


Problem : **Exploding gradients**

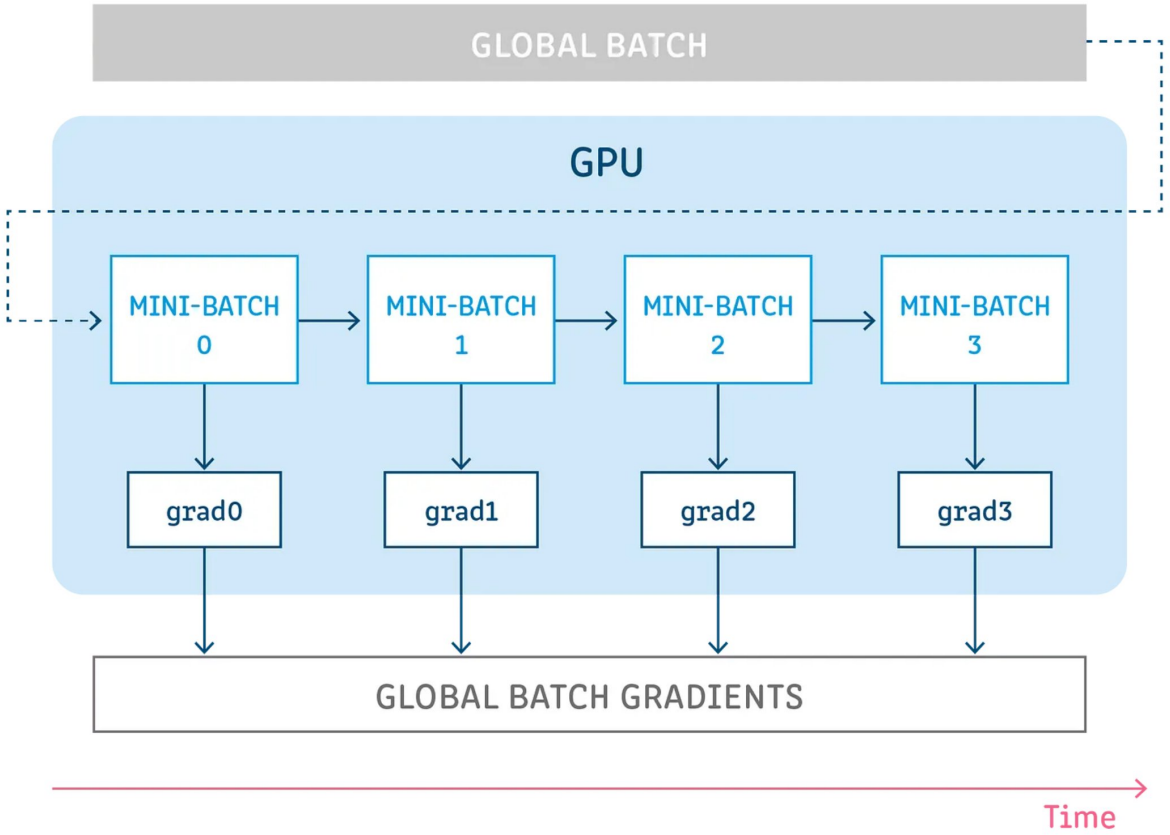
- Large increase in the norm of the gradient during training
- Produce unstable network, the gradient descent step impossible to execute

Gradient clipping solution :
The error derivative is changed or clipped to a threshold during backward propagation through the network. The clipped gradients is used to update the weights.

Other solutions : Regularization, weight-decay



Gradient accumulation

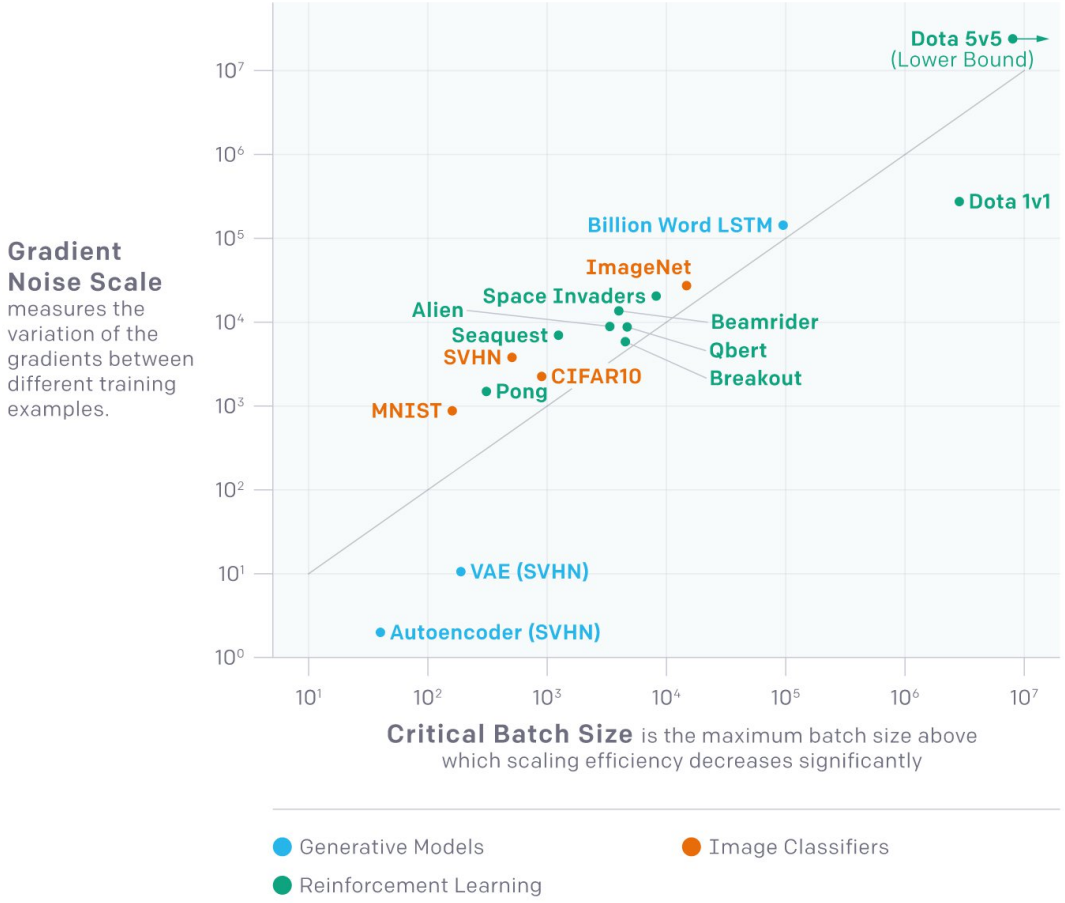


Use Case :

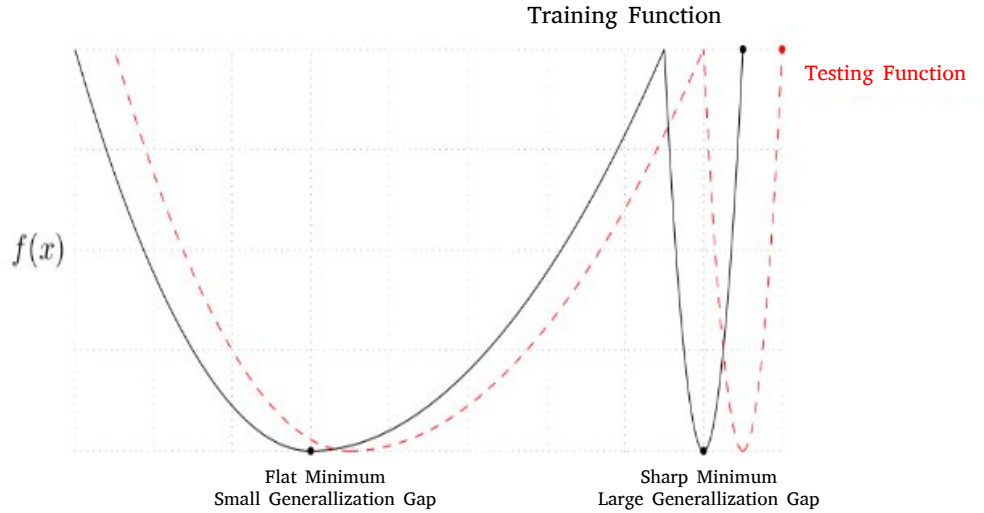
- Not enough memory to have a normal Batch Size
- Reduces the number of communication between host and device (accelerator)

Batch size

The **best batch size** is the **maximum size above which scaling efficiency decreases** (and that fits on the system).

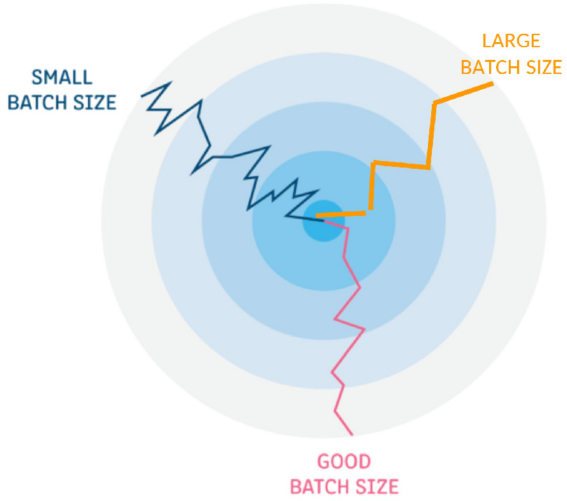


Large-Batch Training



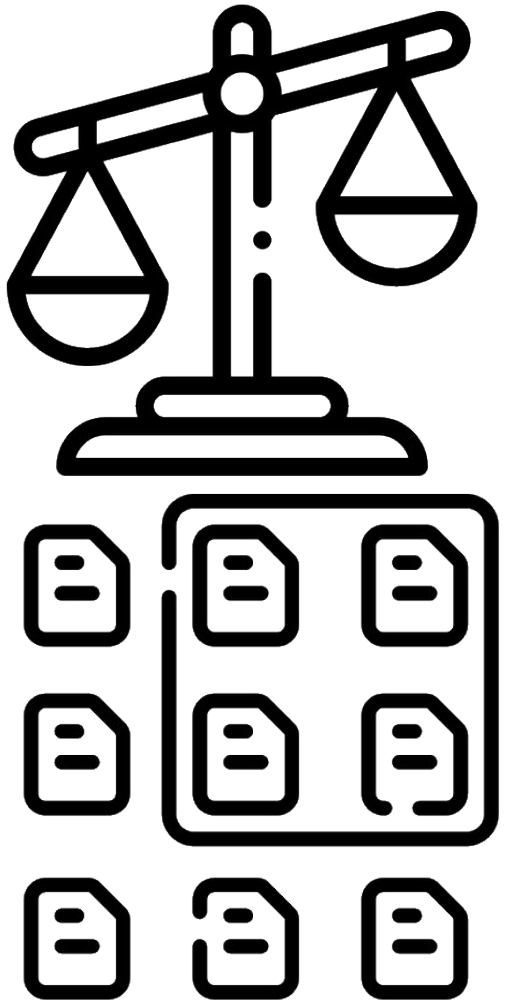
Large batch => model converge towards steep and narrow minimums

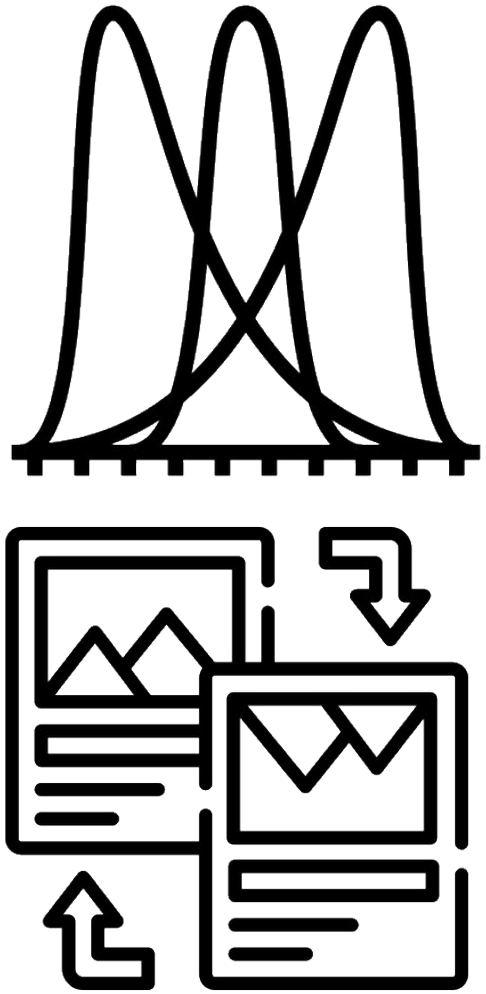
Scaling (multi-gpu/multi-nodes) => large batch



https://openai.com/blog/science-of-ai/

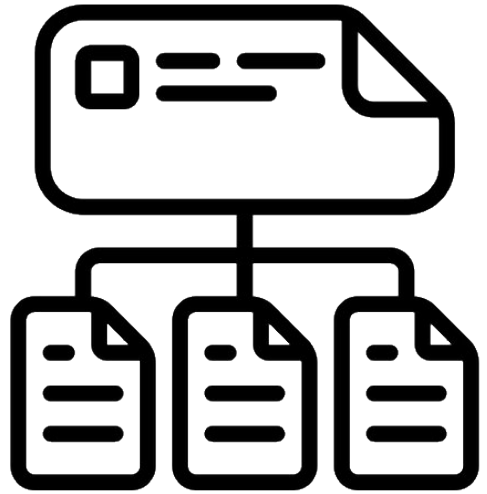
Dataset improvement



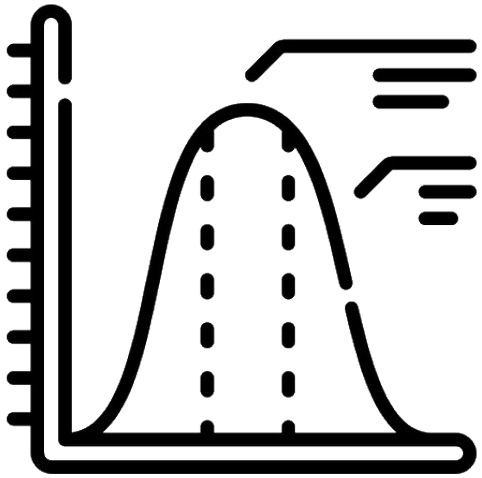


On a dataset scale :

- Split your dataset (train, test, val)
- Manage your misproportions and irregularities
- Use data augmentation to have more diversity



Data improvement

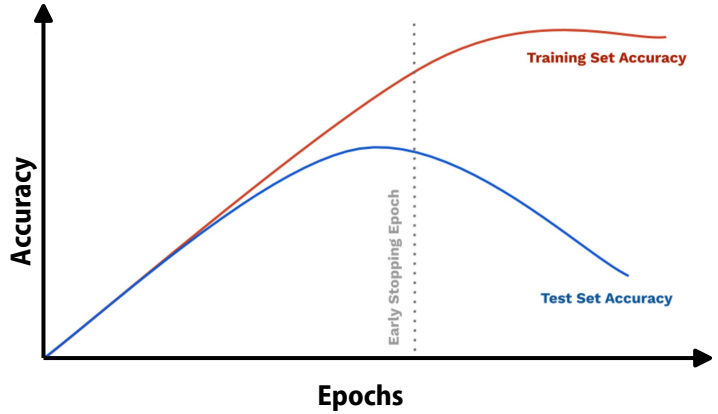


On a per data scale :

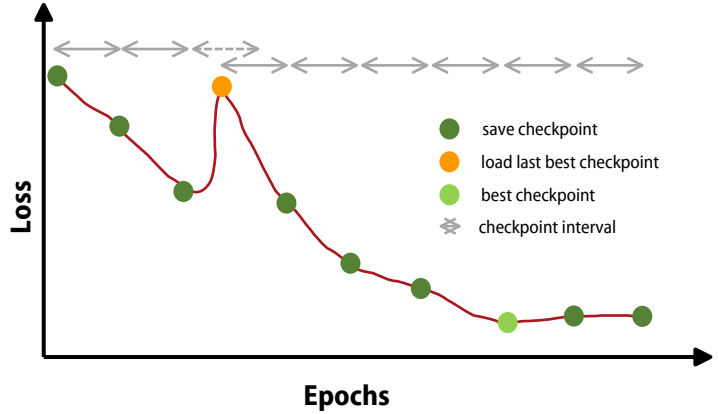
- Scale features
- Clean bad data
- Select features

Training tricks

Early Stopping



Checkpointing



- torch.no_grad()
- profiler
- render
- logs

Don’t forget to update !

Software & Libraries

- Cuda
- Pytorch 2.0 / Tensorflow
- Lightning / Keras

Model & training architecture

- Optimizer
- Kernels
- Blocs

Yourself !

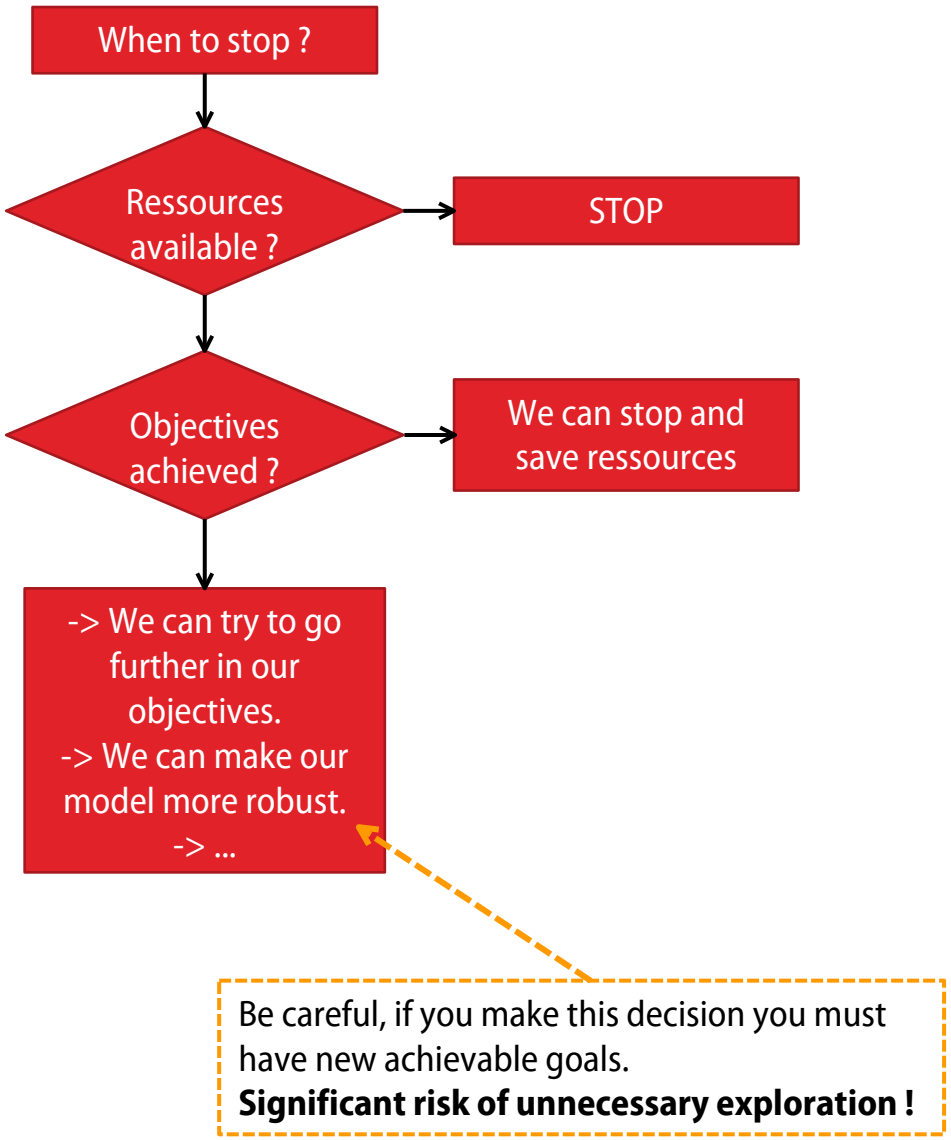
- Training
- Online courses
- Talks

Questions Break

A successful training is all you need !

FIDLE

What is a successful training ?



~~What is a successful training ?~~

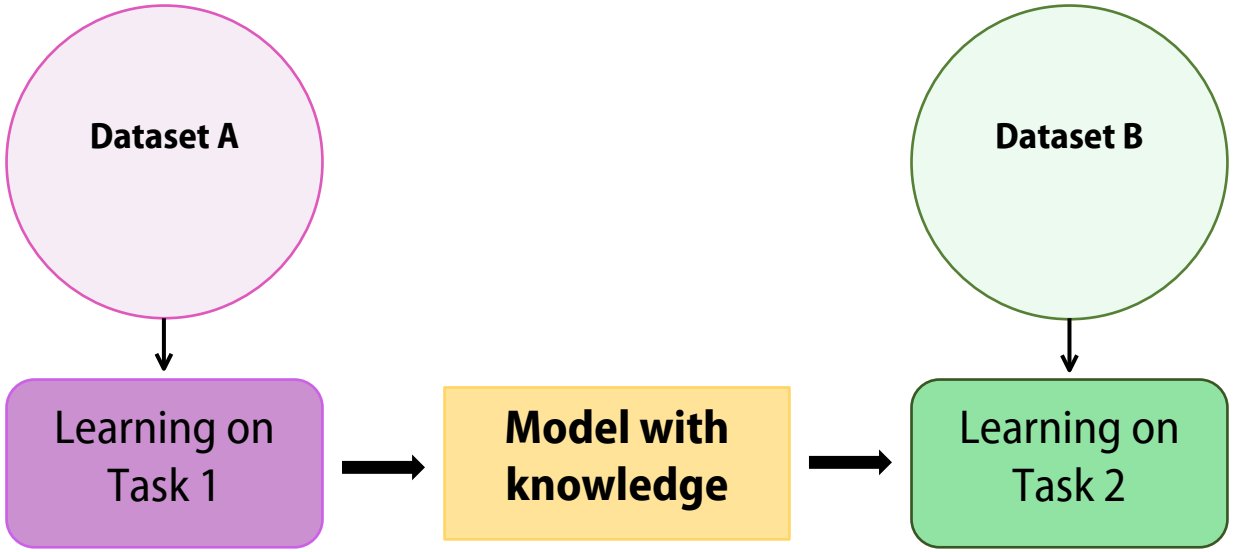
What do we want to achieve with the resources we have?

- Reach a specific metric
- Propose a new model architecture
- Improve an element of the model or training
- Pre-train an LLM only a
- ....

A successful training allows us to move towards our objectives.

THE successful training is the one that brings us closer (or exceeds) the most of these.

Transfert-Learning



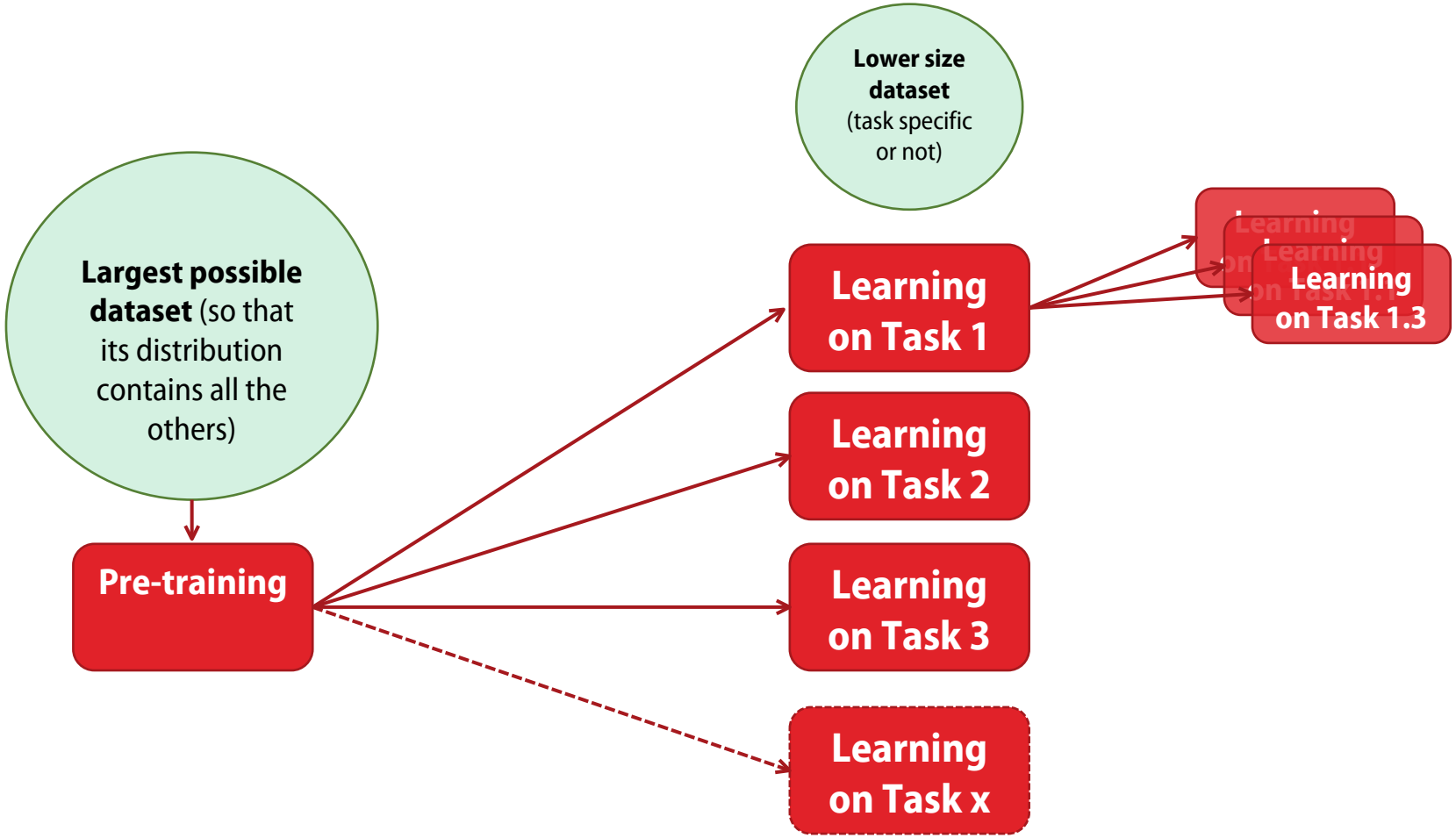
Use knowledge learned (from models) in past experiences to go faster and further with new training.

Benefits :

- **Less Training Time**
- **Less Data**
- **Better Performance**

Question to ask before a Transfert-Learning :
What to transfer ? **How** to transfer ? **When** to transfer ?

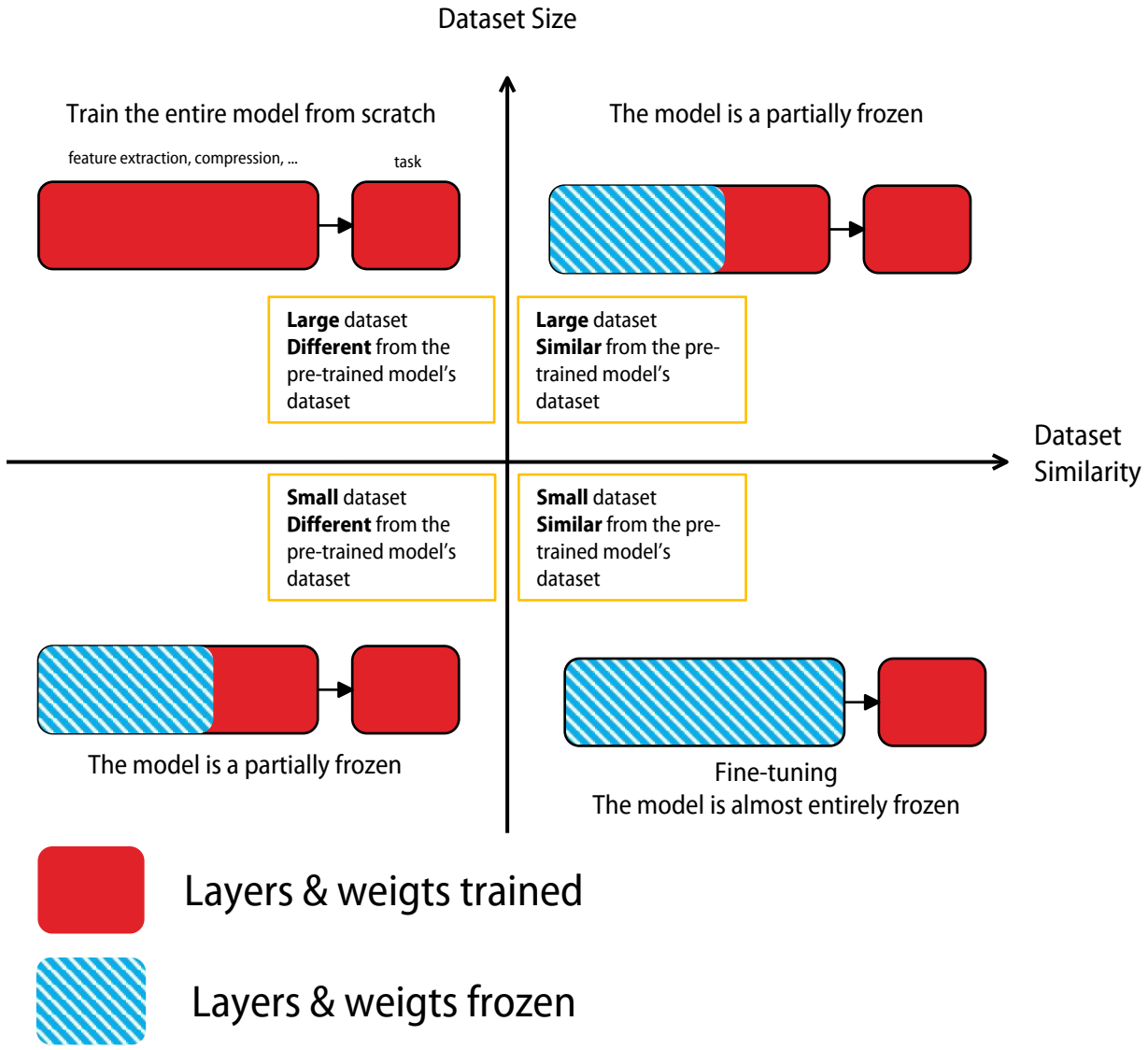
Large Model and pre-training



Large **L**anguage **M**odels are **very time consuming** and **very expensive** to train.

Fine-Tuning

Fine-Tuning in Transfer learning



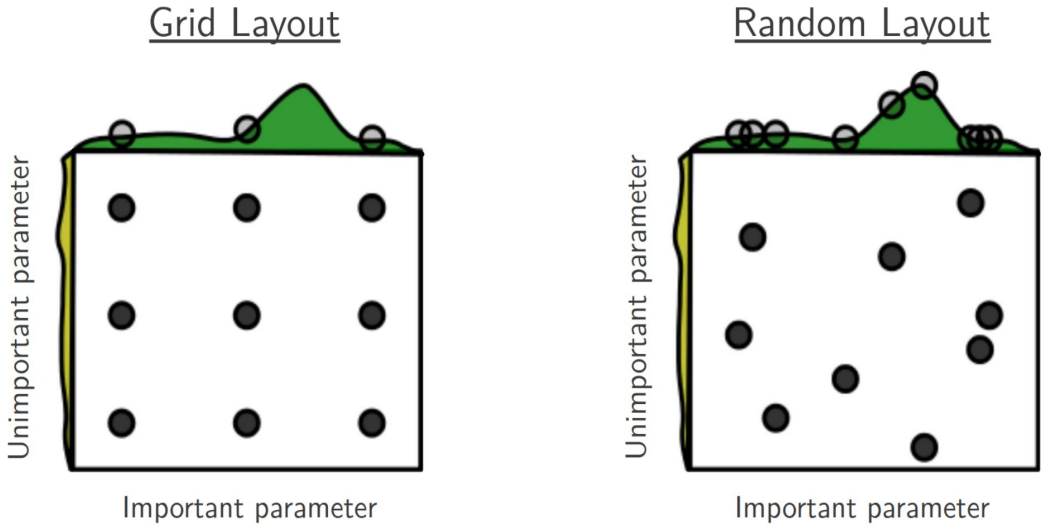
Fine-Tuning in standalone

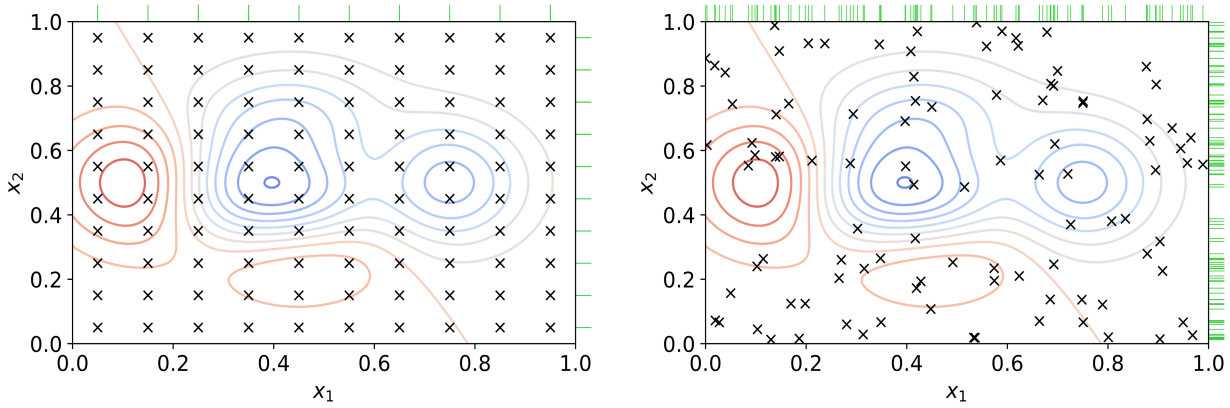
``There is *Fine* in Fine-Tuning``

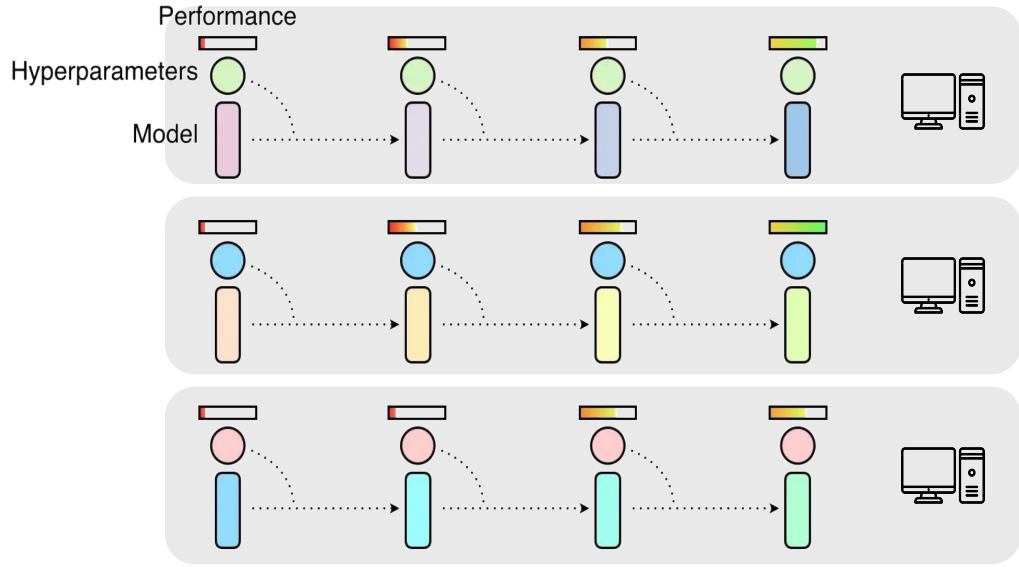
We want refinement.
The weights of the models will change very little :

- training on a dataset subset
- small learning rate

HPO : Grid & Random Search

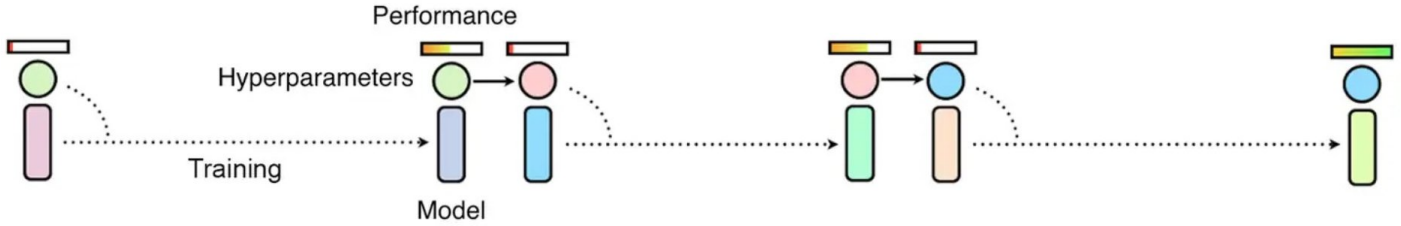




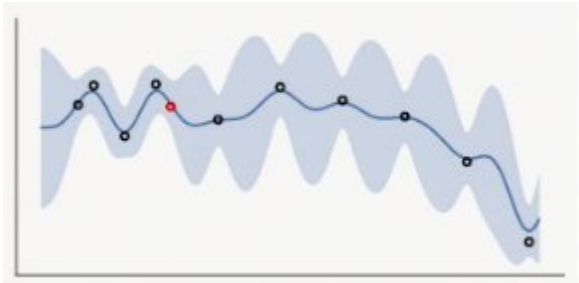


Independent tests (which can be parallelized) which test a combination of hyperparameters.
Very costly in resources and no guarantee of improved results.

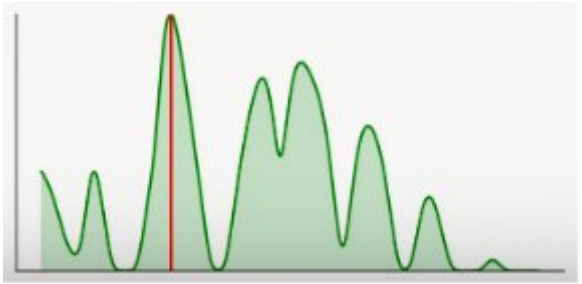
HPO : Bayesian Optimization



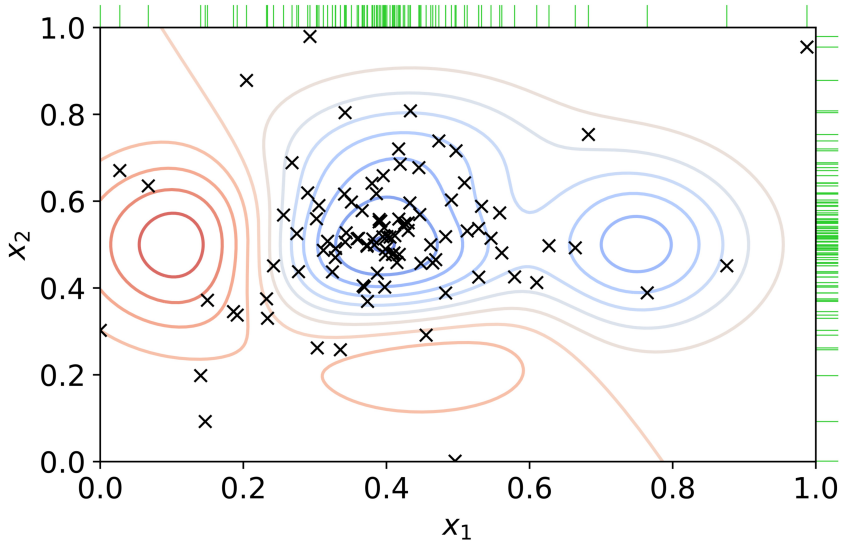
Sequential but allows to quickly find the global optimum. Proposes a new set of hyper parameters based on the scores obtained by the previous ones tested.



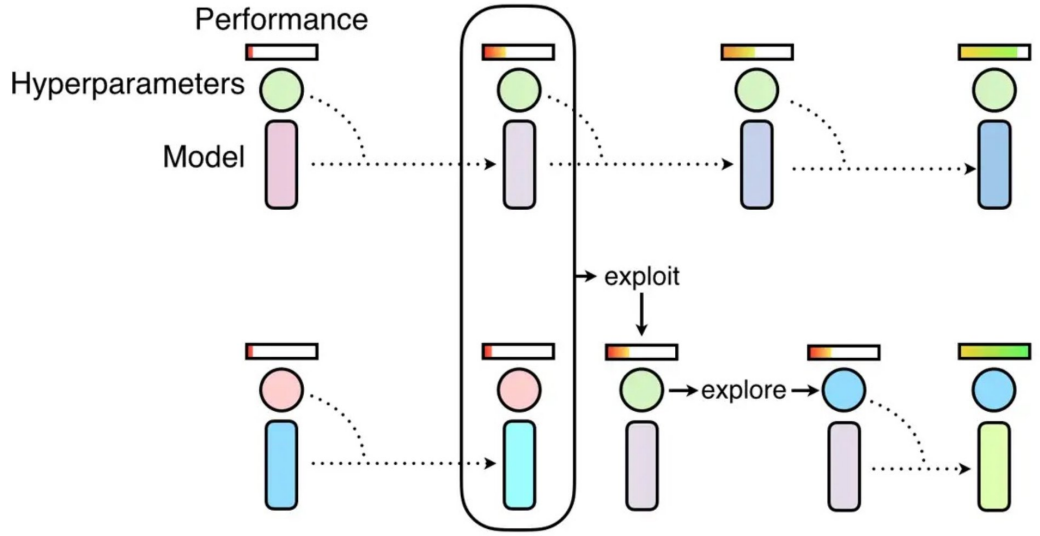
Expected metric score according to Hyper-parameters

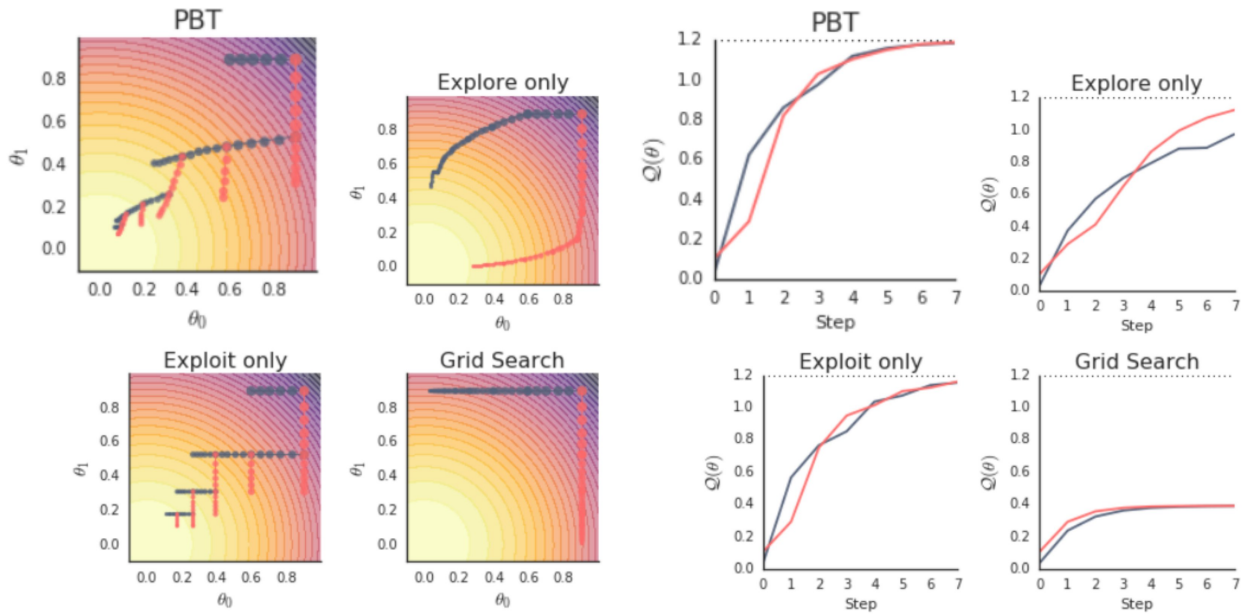


Maximize Acquisition function e.g. Expected Improvement



HPO : Population Based Training Search



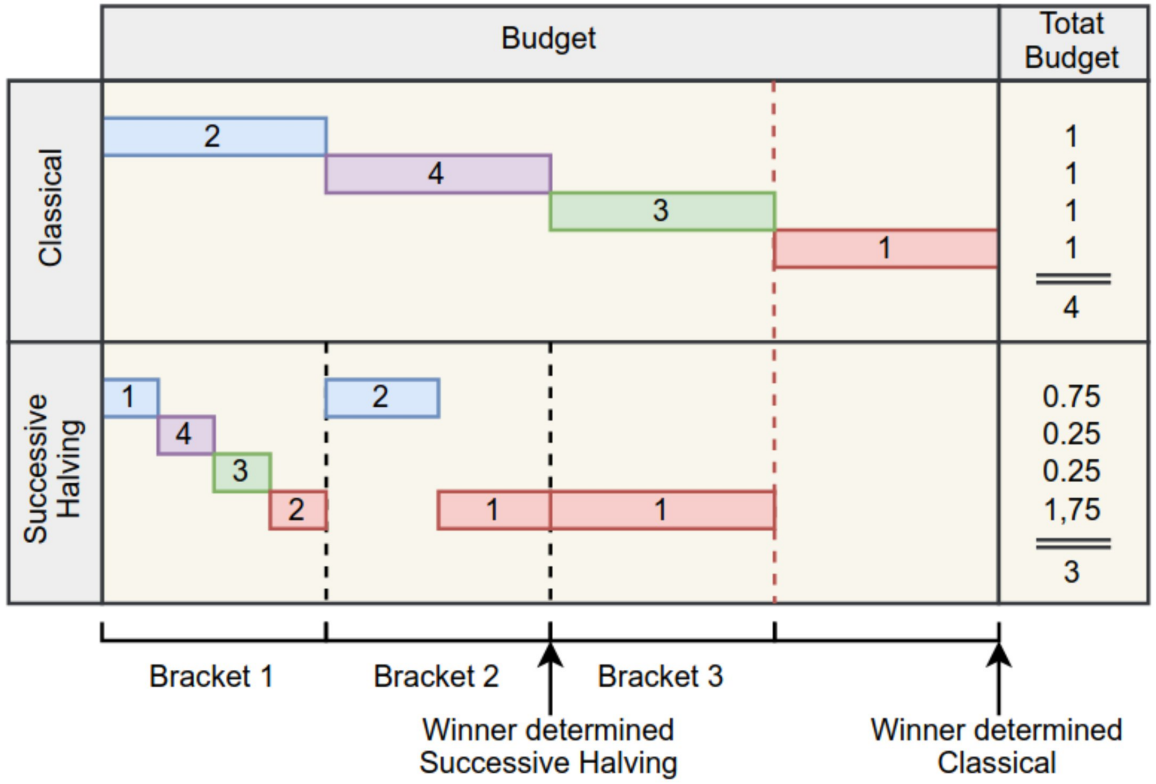


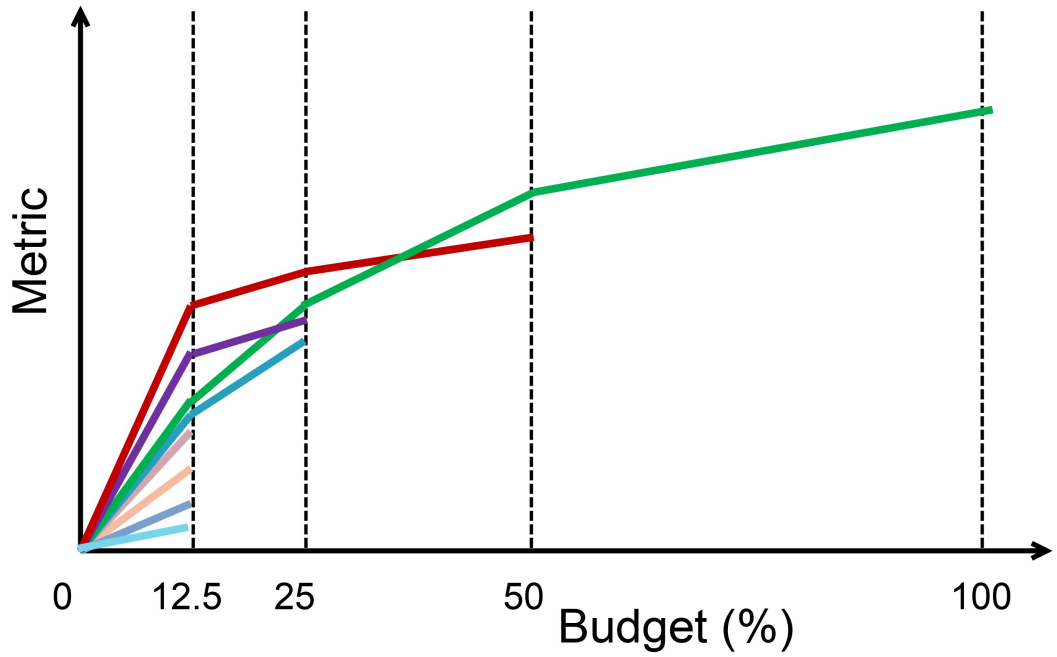
- Research and optimization of hyper parameters during training
- For large models with long and poorly parallelizable tests on a few machines.

- **Exploit** = copy of the weights of the best model
- **Explore** = Bayesian Optimization

https://www.deepmind.com/blog/population-based-training-of-neural-networks

HPO : Successive Halving Algorithm





- For sequentials trials
- Works well with small or medium model -> Trials must be fast !

- For massively parallelizable tests :
  ASHA (Asynchronous Successive Halving Algorithm)

Hyperparameter Search (HPO)

- Based on config file
- Easy to use
- Not only use for ML/DL

OPTUNA

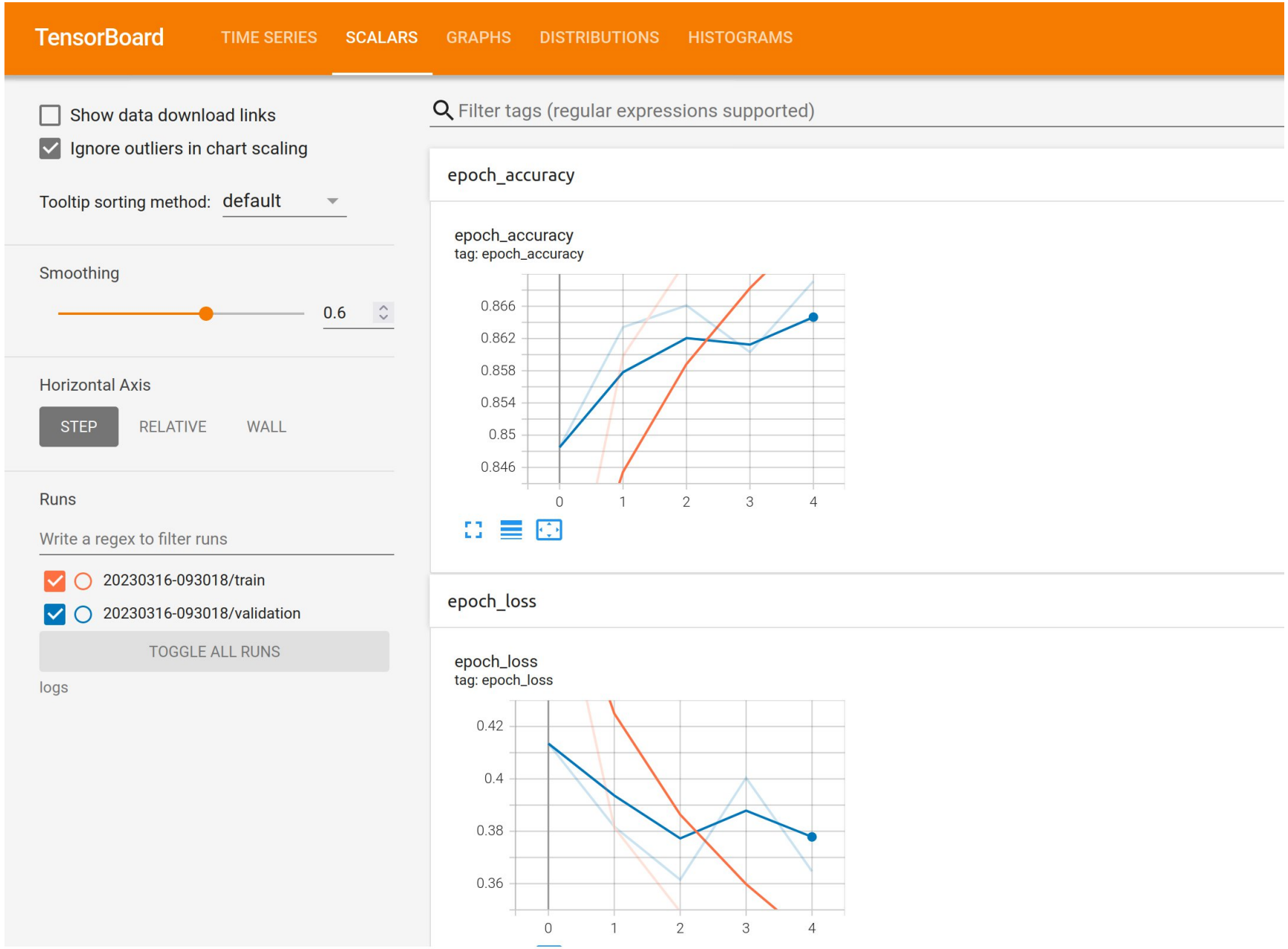
- Work with an objective function
- Efficient Optimization Algorithms

Can be use together !

Visualization tools

FIDLE

Tensorboard



Experiments tracking

Weights & Biases

Source Control

Data Version Control

Advantages :

- allows you to save and order the results
- allows easy comparison and visualization of results
- provides all the information needed to reproduce the results

Experiments tracking

Showing 21 matching runs

				Metrics		Parameters									
	Created	Duration	Run Name	Loss/train	Loss/val	batch_size	deep	device	double	epochs	in_chan	in_dim	latent_dim	lr	residual
☐	⊗ 5 months ago	28.4min	better_lat1…	2467.9	1233.4	64	4	cuda:0	True	3	1	32	256	0.001	True
☐	✓ 5 months ago	47.0min	better_lat1…	1840.8	525	64	3	cuda:0	True	3	1	32	128	0.001	True
☐	✓ 5 months ago	32.2min	better_lat1…	3443.7	652.9	64	3	cuda:0	True	3	1	32	128	0.1	True
☐	✓ 5 months ago	31.9min	better_lat1…	1680.5	3211.8	64	2	cuda:0	True	3	1	32	128	0.01	True
☐	✓ 5 months ago	16.7min	better_lat1…	1320.8	967.3	64	3	cuda:0	True	3	1	32	128	0.01	False
☐	✓ 5 months ago	15.2min	better_lat1…	1838.2	976.7	64	3	cuda:0	False	3	1	32	128	0.01	True
☐	✓ 5 months ago	1.1h	better_lat2…	1239	567.6	64	3	cuda:0	True						
☐	✓ 5 months ago	1.1h	better_lat5…	1953	1514.4	64	3	cuda:0	True						
☐	✓ 5 months ago	18.1min	better_lat6…	1669.5	668.8	64	3	cuda:0	True						
☐	5 months ago		better_lat1…	2368.2	3429.6	64	3	cuda:0	True						
☐	✓ 5 months ago	45.2min	small_lat25…	1901.2	1653.1	64	4	cuda:0	True						
☐	⊗ 5 months ago	1.1h	small_lat25…	1937.5	1039.6	64	4	cuda:0	True						
☐	✓ 5 months ago	29.2min	small_lat25…	2050.4	723	64	5	cuda:0	True						
☐	✓ 5 months ago	27.2min	small_lat25…	1958.2	519.8	64	3	cuda:0	True						
☐	✓ 5 months ago	52.5min	small_lat12…	1733.3	519.5	64	4	cuda:0	True						
☐	✓ 5 months ago	31.2min	small_lat25…	947.4	3987.8	64	4	cuda:0	True						
☐	⊗ 5 months ago	1.2h	small_lat25…	1245.4	1811.4	64	4	cuda:0	True						
☐	✓ 5 months ago	14.1min	big-166555…	2107.9	2693.3	16	8	cuda:0	True						
☐	✓ 5 months ago	20.0min	first-16655…	2276.2	1498.6	64	4	cuda:0	True						
☐	✓ 5 months ago	3.6min	first-16655…	2253.8	1435.3	32	4	cuda:0	True						
☐	✓ 5 months ago	3.9min	first-16655…	2412.2	1407.1	32	5	cuda:0	True						

Test pretrain AutoEncoder › Comparing 6 Runs from 1 Experiment › Loss/train

Loss/train

Completed Runs 6/6

Points:

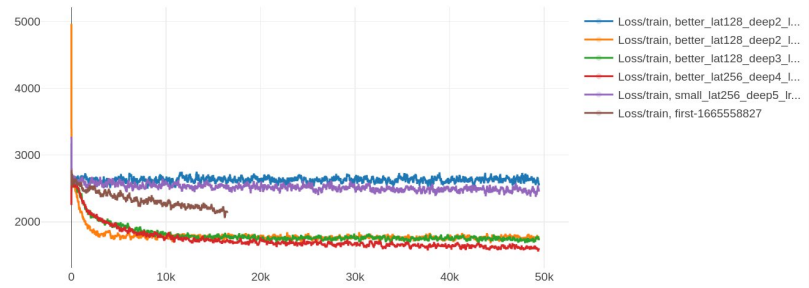
Line Smoothness 100

X-axis:
- Step
- Time (Wall)
- Time (Relative)

Y-axis: Loss/train

Y-axis Log Scale:

Download CSV



Loss/train

Run	Latest	Min	Max
better_lat128_deep2_lr01-1665582265	3443.7 (step=49439)	1069.9 (step=3773)	5205.4 (step=14284)
better_lat128_deep2_lr01-1665582227	1680.5 (step=49439)	806.5 (step=1210)	4966.9 (step=0)
better_lat128_deep3_lr01_nores-1665582196	1320.8 (step=49439)	870.7 (step=25165)	4310 (step=1550)
better_lat256_deep4_lr01-1665582151	1239 (step=49439)	820.8 (step=37359)	4494.6 (step=136)
small_lat256_deep5_lr01-1665575822	2050.4 (step=49439)	1045.7 (step=28182)	5172.2 (step=2325)
first-1665558827	2276.2 (step=16479)	1013.7 (step=16391)	4825.8 (step=398)

Profiler

Configuration

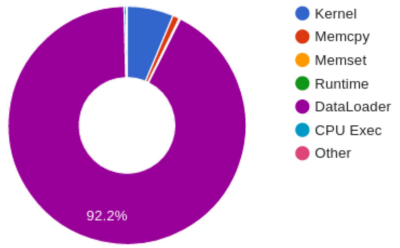
Number of Worker(s)	1
Device Type	GPU

GPU Summary

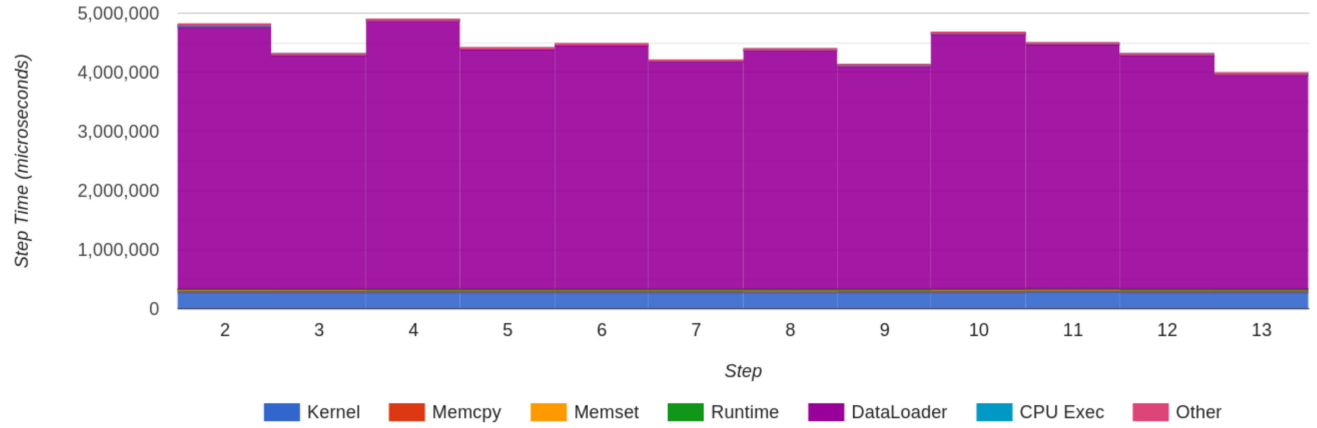
GPU 0:	
Name	Tesla V100-SXM2-32GB
Memory	31.74 GB
Compute Capability	7.0
GPU Utilization	6.33 %
Est. SM Efficiency	6.26 %
Est. Achieved Occupancy	47.2 %
Kernel Time using Tensor Cores	42.95 %

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	4,419,330	100
Kernel	279,658	6.33
Memcpy	40,750	0.92
Memset	1,316	0.03
Runtime	6,381	0.14
DataLoader	4,072,972	92.16
CPU Exec	13,462	0.3
Other	4,790	0.11



Step Time Breakdown



Performance Recommendation

- This run has high time cost on input data loading. 92.2% of the step time is in DataLoader. You could try to set num_workers on DataLoader's construction and enable multi-processes on data loading.
- GPU 0 has low utilization. You could try to increase batch size to improve. Note: Increasing batch size may affect the speed and stability of model convergence.

GPU type and memory capacity

% of time spent with an active GPU

% of time spent with an active SM

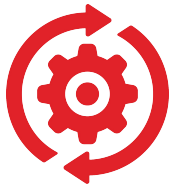
% of active wraps on a SM

% of time spent on Tensor Cores

Questions Break

Example : Resnet Cifar10

FIDLE



TP software optimisation & tracking

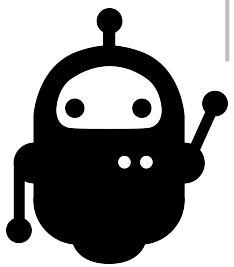
Notebook :

Objective :

Step by step, optimize a simple training

Dataset :

Cifar10



Bibliography

Papers :

Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model (https://arxiv.org/abs/2211.02001)
The cost of training NLP models (https://arxiv.org/pdf/2004.08900.pdf)
Green AI (https://arxiv.org/pdf/1907.10597.pdf)
Adam (https://arxiv.org/pdf/1412.6980.pdf)
Algorithms for Hyper-Parameter Optimization (https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf)
Compute Trends Across Three Eras of Machine Learning (https://arxiv.org/abs/2202.05924)
On large-batch training for deep learning (https://arxiv.org/pdf/1609.04836.pdf)

Other Ressources :

SGD (https://paperswithcode.com/method/sgd-with-momentum)(https://senarvi.github.io/understanding-convergence-of-sgd)
GPT3 for 500k$ (https://www.mosaicml.com/blog/gpt-3-quality-for-500k)
AI memory wall (https://github.com/amirgholami/ai_and_memory_wall)
Gradient accumultation (https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa)
Gradient clipping & many other things (https://www.deeplearningbook.org/)

Next, on Fidle :

« Two knights running very fast »
By SDXL Beta

17

Jean-Zay
GPU acceleration

Merci !

FIDLE

Jeudi 13 avril,

Épisode 17 :

Passer à la vitesse supérieure : l'accélération matérielle

- Présentation de Jean-Zay et calcul sur GPU
- Distribution du calcul - Data Parallelism - Model Parallelism
- Hybrid Parallelism - Pipelines
- Deepspeed - Optimisation du Data parallelism
- Bonnes pratiques pour un apprentissage distribué

Durée : 2h