Compléments

Julien Romero

Retour sur les fonctions d'activation

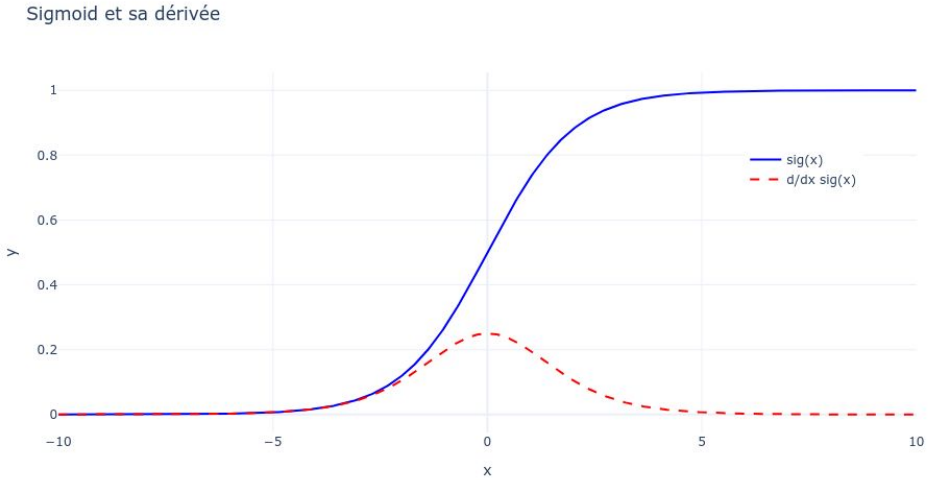
La sigmoid

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Transforme un nombre réel en un réel dans l'intervalle [0, 1]

Interprétation simple comme une probabilité dans une classification binaire

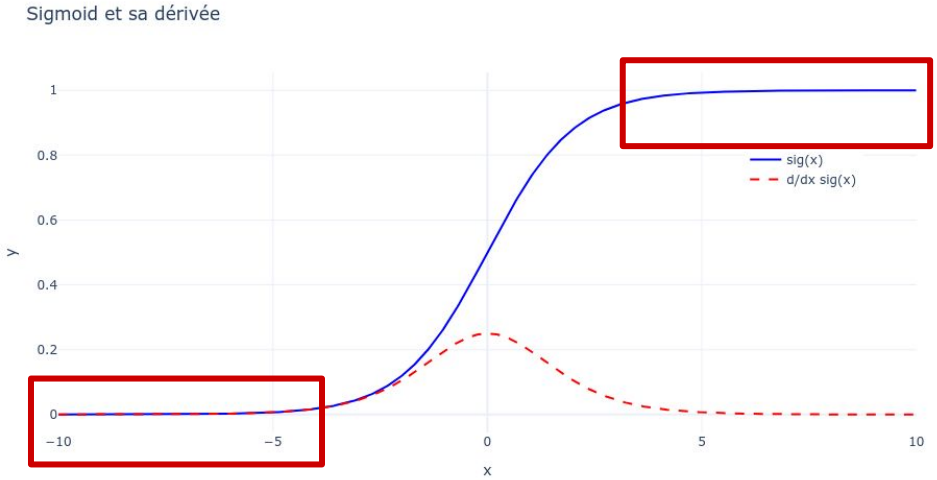


La sigmoid

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial \sigma(x)}{\partial x}} \frac{\partial L}{\partial \sigma(x)}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



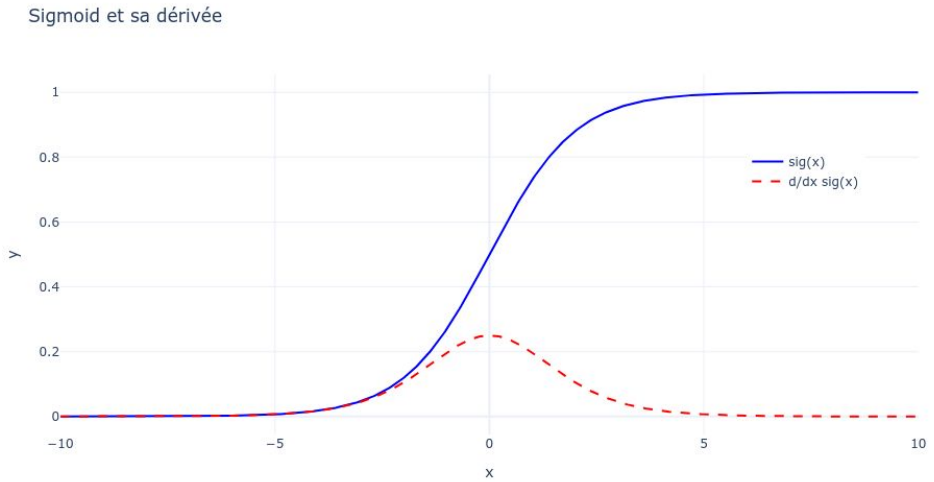
3 points négatifs:

- La saturation “tue” le gradient

La sigmoid

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



3 points négatifs:

- La saturation “tue” le gradient
- La sortie n’est pas centrée sur 0

Le problème des fonctions d'activation positives

Prenons une couche d'un MLP

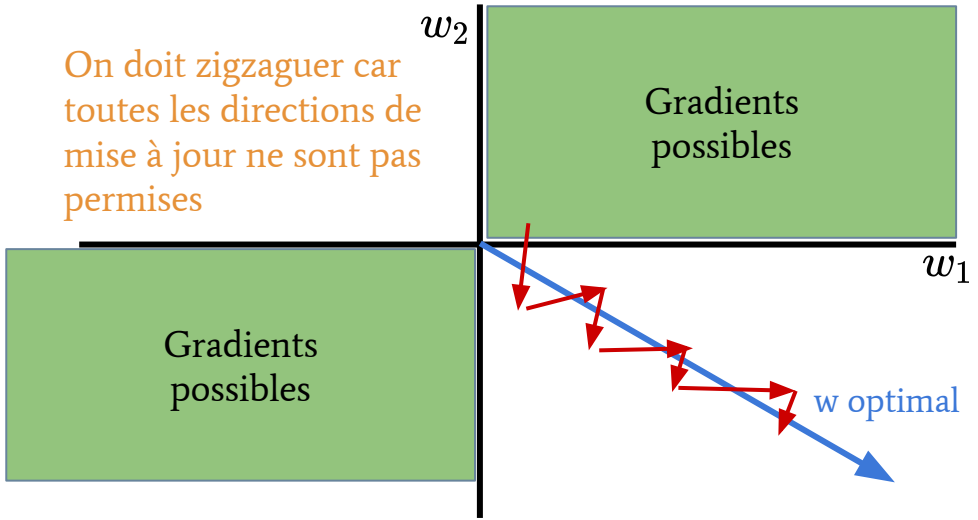
$$h_i^l = \sum_j w_{i,j}^l \sigma(h_j^{l-1}) + b_i^l$$

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial h_i^l}{\partial w_{i,j}} \frac{\partial L}{\partial h_i^l} = \sigma(h_j^{l-1}) \frac{\partial L}{\partial h_i^l}$$

Que peut-on dire sur les gradients par rapport aux poids de cette couche ?

On ne modifie pas le signe du gradient en entrée pendant la backpropagation

Pour un neurone donnée, tous les gradients ont le même signe



Le problème des fonctions d'activation positives

Prenons une couche d'un MLP

$$h_i^l = \sum_j w_{i,j}^l \sigma(h_j^{l-1}) + b_i^l$$

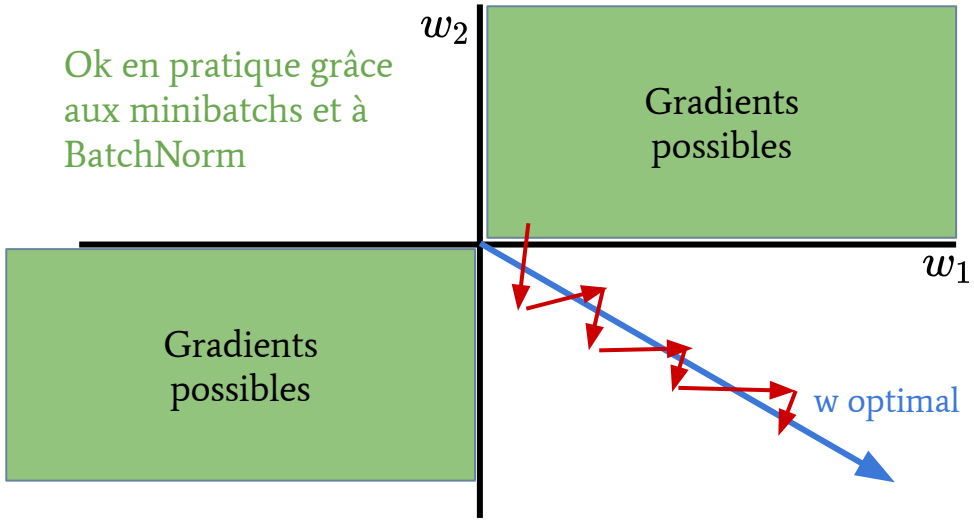
$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial h_i^l}{\partial w_{i,j}} \frac{\partial L}{\partial h_i^l} = \sigma(h_j^{l-1}) \frac{\partial L}{\partial h_i^l}$$

Que peut-on dire sur les gradients par rapport aux poids de cette couche ?

On ne modifie pas le signe du gradient en entrée pendant la backpropagation

Pour un neurone donnée, tous les gradients ont le même signe

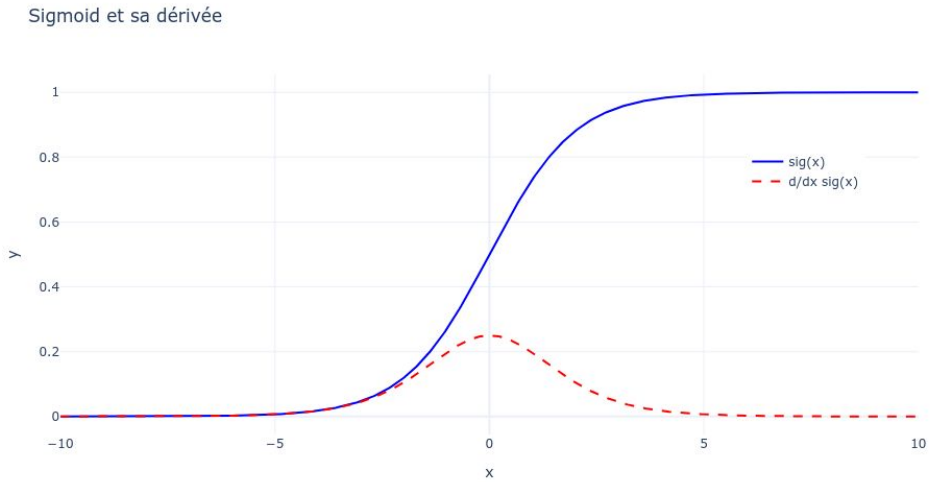
Ok en pratique grâce aux minibatchs et à BatchNorm



La sigmoid

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



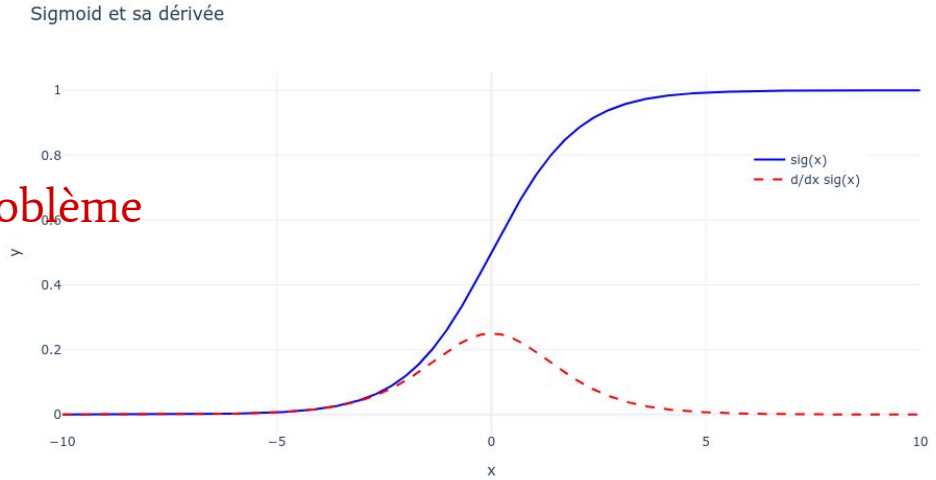
3 points négatifs:

- La saturation “tue” le gradient
- La sortie n’est pas centrée sur 0
- exp() coûte cher à calculer

La sigmoid

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Pire problème

3 points négatifs:

- **La saturation "tue" le gradient**
- La sortie n'est pas centrée sur 0
- exp() coûte cher à calculer

La tangente hyperbolique

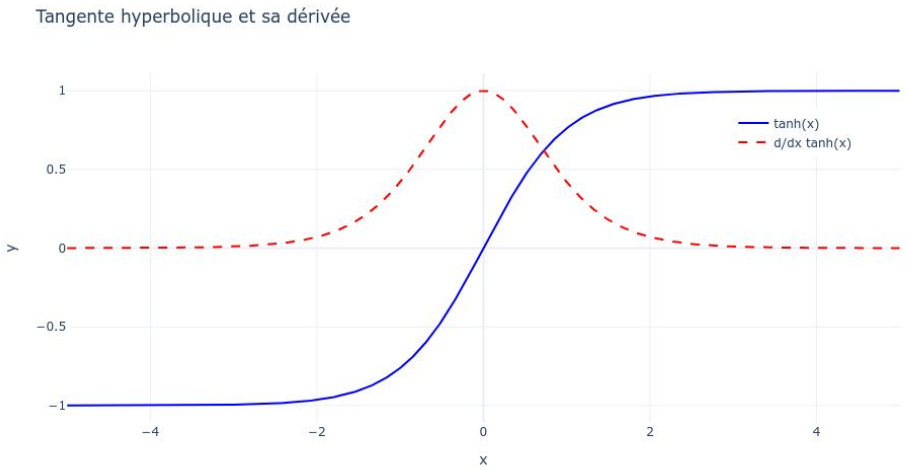
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$tanh'(x) = 1 - tanh^{2}(x)$$

Une sortie entre -1 et 1

Problèmes de saturation

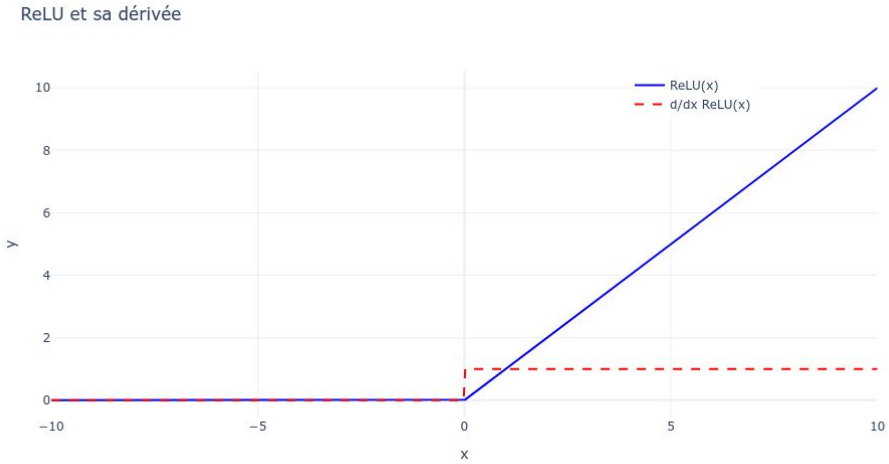
Centrée sur 0



ReLU - Rectified Linear Unit

$$ReLU(x) = max(0, x)$$

$$ReLU'(x) = 0 \text{ si} x < 0; 1 \text{ si} x > 0$$



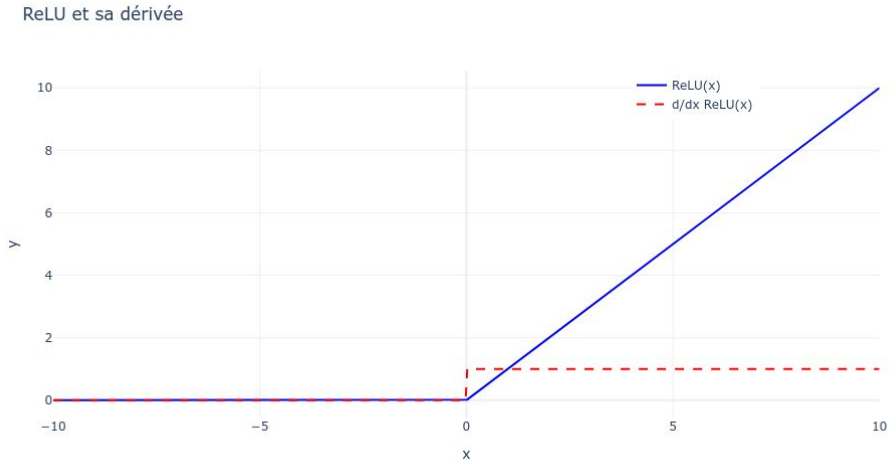
- \+ Pas de saturation dans les positifs
- \+ Efficace à calculer
- \+ Convergence plus rapide que la sigmoid et tanh en pratique

ReLU - Rectified Linear Unit

$$ReLU(x) = max(0, x)$$

$$ReLU'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$

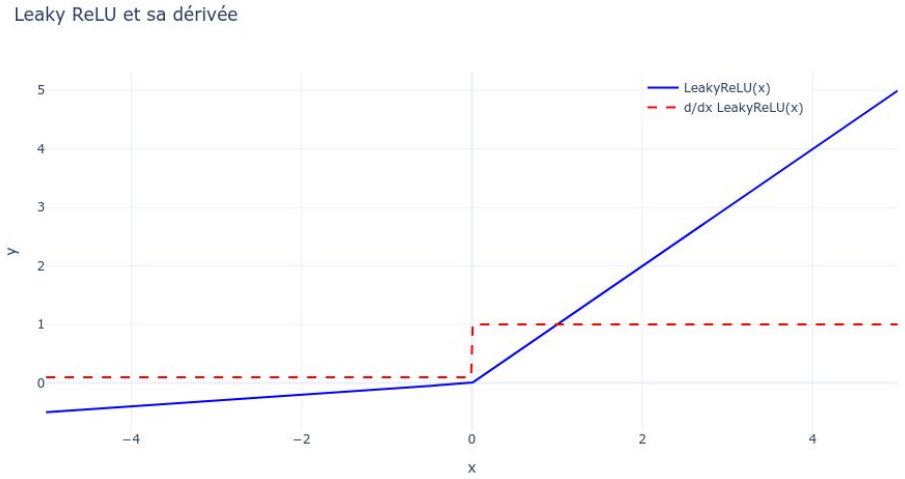
- Non centrée sur 0, positive
- Pas d'apprentissage si l'entrée est négative
- Dérivée non définie en 0
- Utilisable uniquement dans les couches cachée
- Sortie potentiellement très grande



Leaky ReLU

$$LeakyReLU(x) = max(\alpha x, x)$$

$$LeakyReLU'(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$



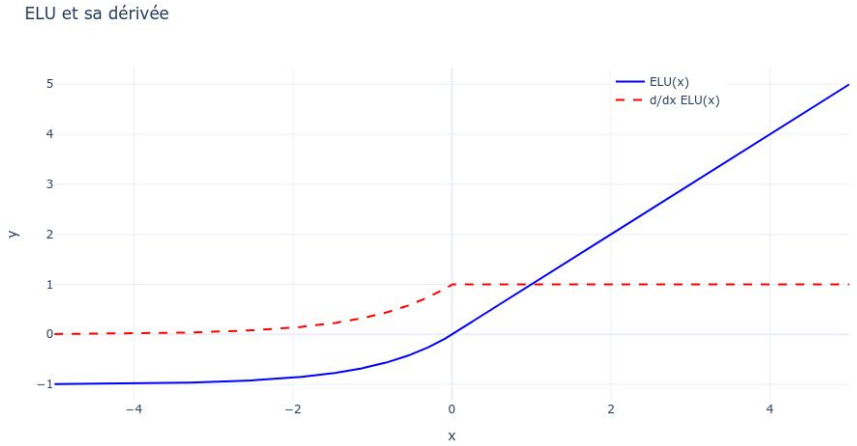
α vaut généralement 0.1 mais peut être appris (Parametric ReLU)

+ Pas de saturation
+ Calcul efficace
+ Convergence rapide
+ Gradient jamais nul

ELU - Exponential Linear Unit

$$ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

$$ELU'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$$



+ Bénéfices de ReLU
+ Moyenne plus proche de 0
+ Saturation dans les négatifs apporte de la stabilité
+ Transition plus lisse en 0
- exp coûteux

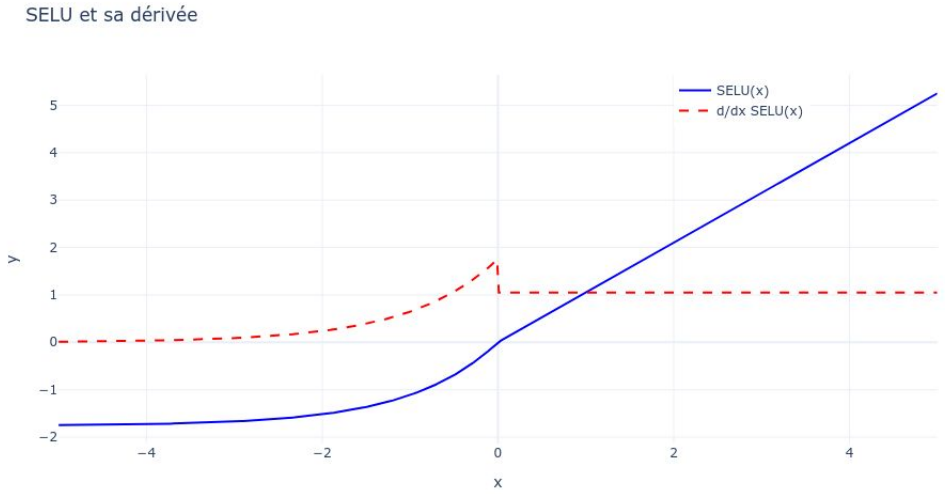
SELU - Scaled Exponential Linear Unit

$$SELU(x) = \begin{cases} \lambda\alpha(e^x - 1) & \text{if } x \leq 0 \\ \lambda x & \text{if } x > 0 \end{cases}$$

$$ELU'(x) = \begin{cases} \lambda\alpha e^x & \text{if } x < 0 \\ \lambda & \text{if } x > 0 \end{cases}$$

$$\alpha = 1.67326324235437728481704299167 17$$
$$\lambda = 1.05070098735548049341933498529 46$$

+ Marche mieux pour les réseaux profonds
+ Auto-normalisation = peut remplacer BatchNorm



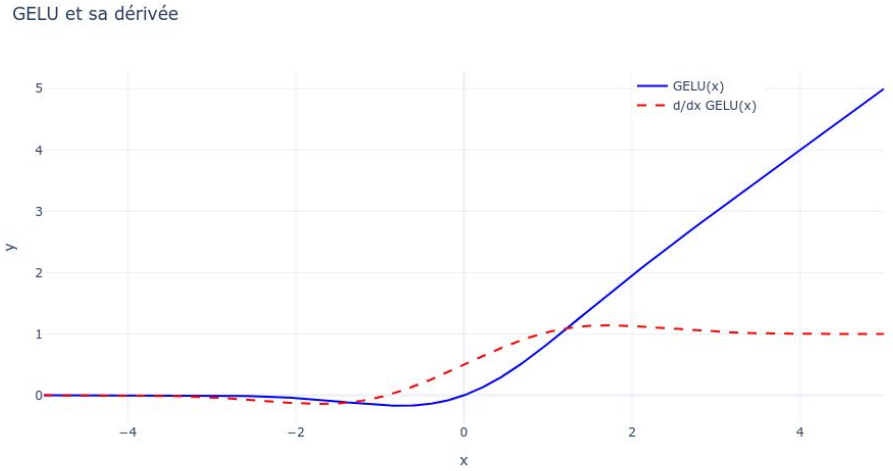
GELU - Gaussian Error Linear Unit (GELU)

$$X \sim N(0,1)$$

$$GELU(x) = x\mathbb{P}(X \leq x)$$

$$GELU(x) \approx x\sigma(1.702x)$$

Idée : Plus l'entrée est faible, plus on l'annule. Plus elle est grande, plus on la conserve.



+ Ressemble à ReLU
+ Dérivable partout !

Très courant dans Transformers

De nombreuses autres fonctions possibles

- SiLU (Sigmoid Linear Unit) : $x\sigma(x)$
- GLU (Gated Linear Unit) : SiLU avec des régressions linéaires
- GeGLU (Gated Linear Unit with GELU activation) : Semblable à GLU avec un GELU
- Swish : Similaire à SiLU avec un paramètre en plus dans la sigmoid
- SwiGLU : Semblable à GLU avec un Swish

Comparaison

Model	ResNet	WRN	DenseNet
LReLU	94.2	95.6	94.7
PReLU	94.1	95.1	94.5
Softplus	94.6	94.9	94.7
ELU	94.1	94.1	94.4
SELU	93.0	93.2	93.9
GELU	94.3	95.5	94.8
ReLU	93.8	95.3	94.8
Swish-1	94.7	95.5	94.8
Swish	94.5	95.5	94.8

Table 4: CIFAR-10 accuracy.

Model	ResNet	WRN	DenseNet
LReLU	74.2	78.0	83.3
PReLU	74.5	77.3	81.5
Softplus	76.0	78.4	83.7
ELU	75.0	76.0	80.6
SELU	73.2	74.3	80.8
GELU	74.7	78.0	83.8
ReLU	74.2	77.8	83.7
Swish-1	75.1	78.5	83.8
Swish	75.1	78.0	83.9

Table 5: CIFAR-100 accuracy.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions

En pratique

- On utilise ReLU par défaut
- Si vraiment on cherche à gagner un peu de performance, tester d'autres fonctions
- On oublie sigmoid et tanh

Pytorch

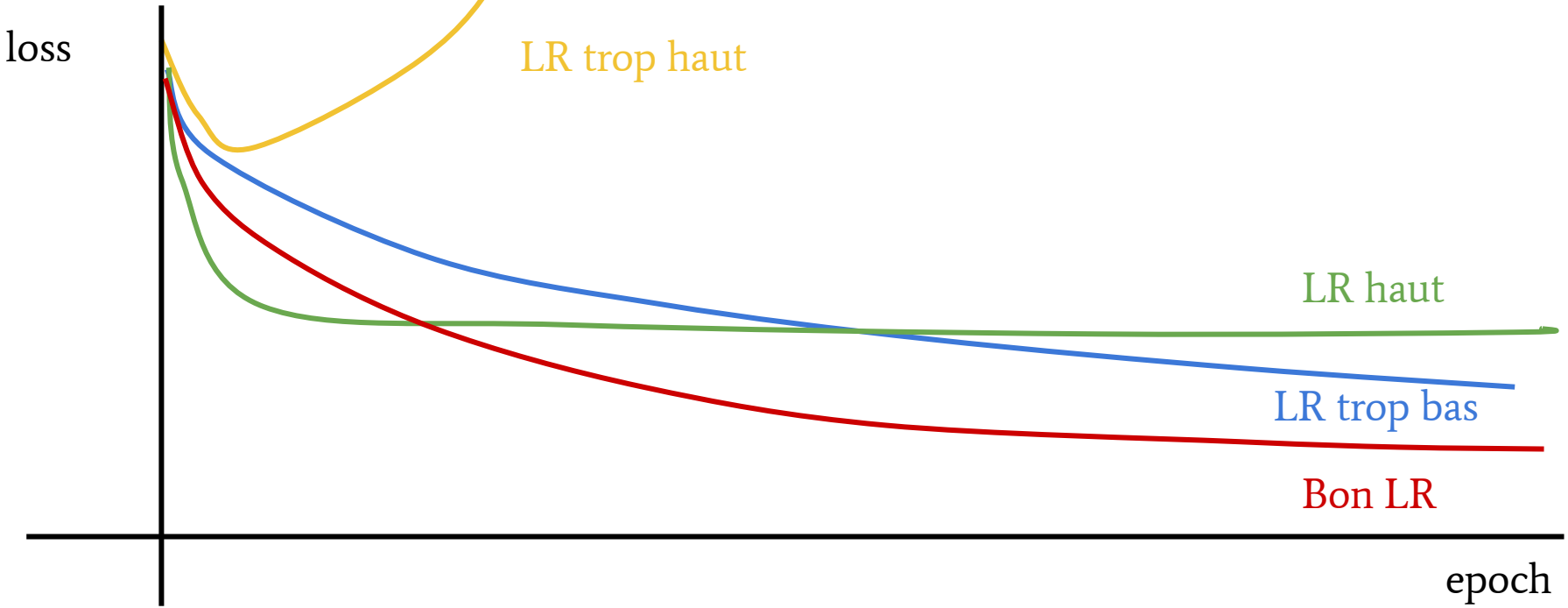
La plupart sont implémentés dans PyTorch, dans **torch.nn**

Non-linear Activations (weighted sum, nonlinearity)

`nn.ELU`	Applies the Exponential Linear Unit (ELU) function, element-wise.
`nn.Hardshrink`	Applies the Hard Shrinkage (Hardshrink) function element-wise.
`nn.Hardsigmoid`	Applies the Hardsigmoid function element-wise.
`nn.Hardtanh`	Applies the HardTanh function element-wise.
`nn.Hardswish`	Applies the Hardswish function, element-wise.
`nn.LeakyReLU`	Applies the LeakyReLU function element-wise.

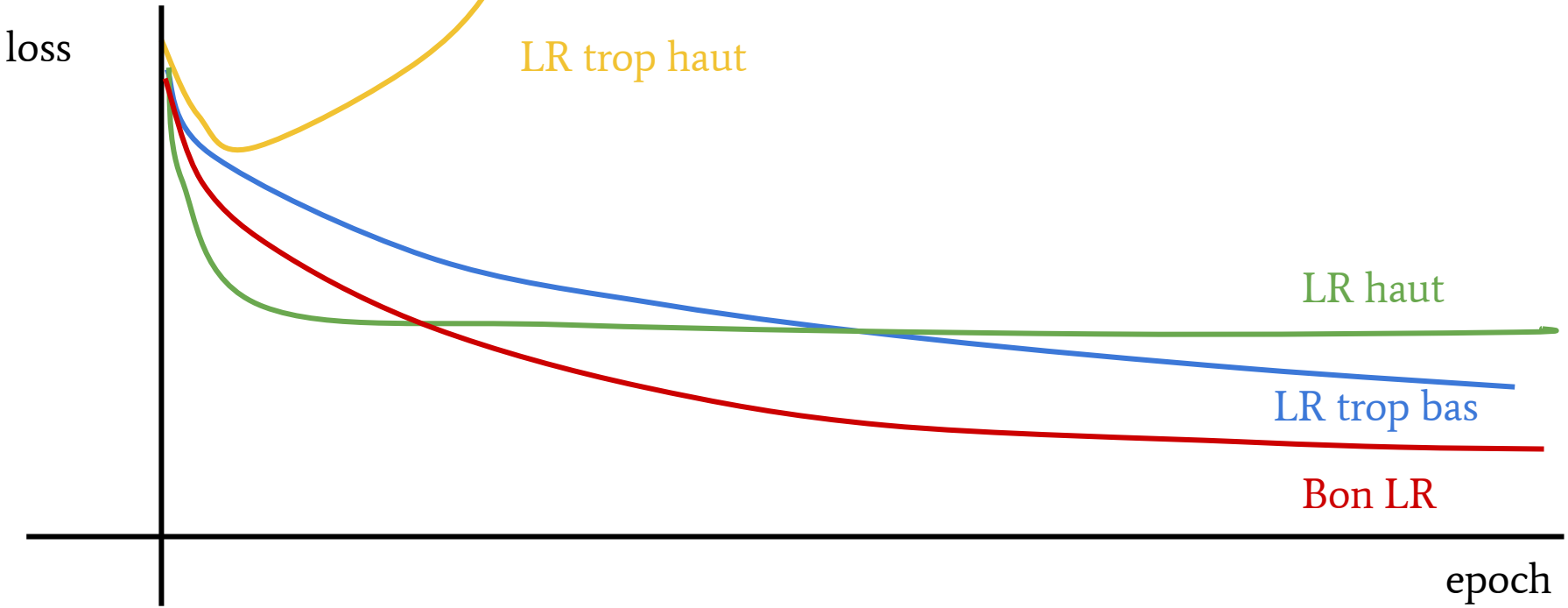
Planification du taux d'apprentissage

Comportements classiques



Comportements classiques

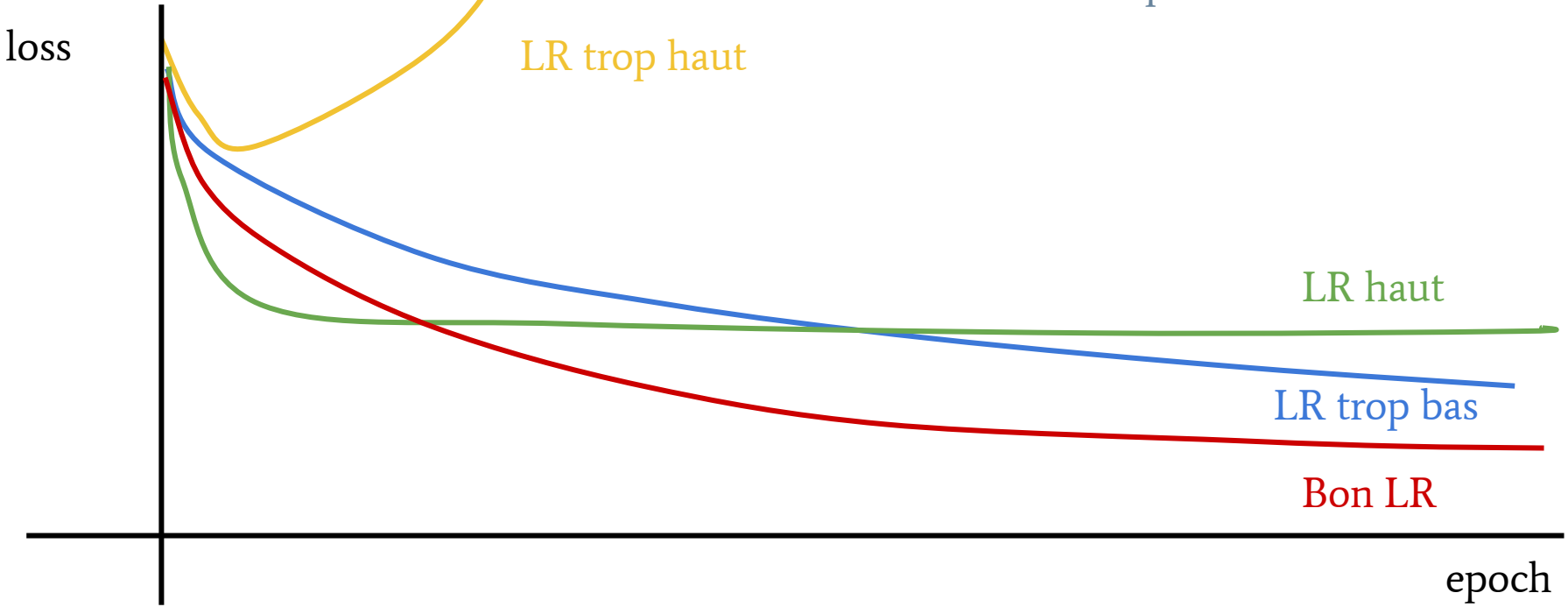
Quel learning rate choisir ?



Comportements classiques

Quel learning rate choisir ?

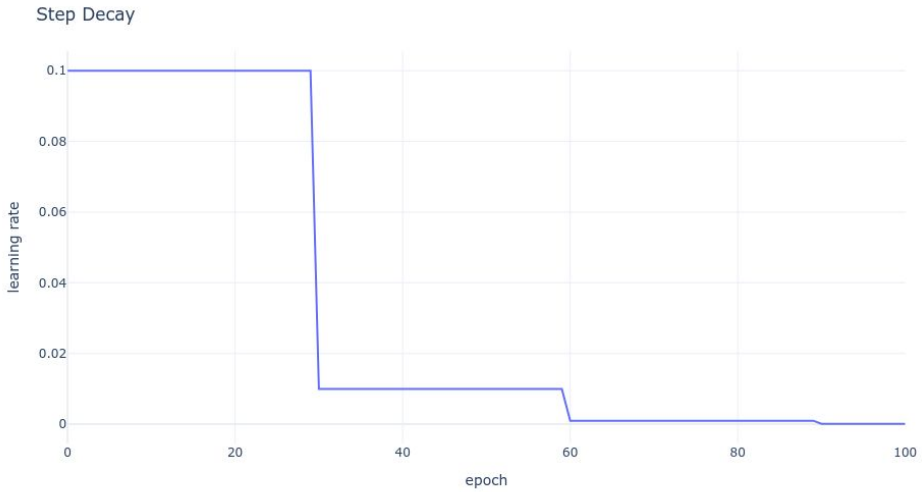
Dans l'idéal plusieurs.



Learning Rate Decay : Step

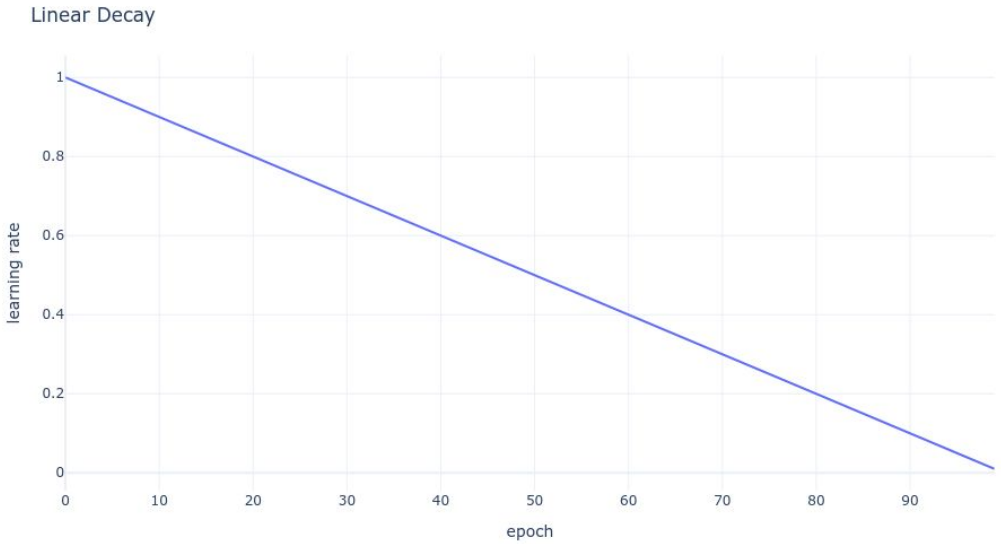
Idée : Réduire le taux d'apprentissage à des points fixes.

Par exemple : ResNet multiple le learning rate par 0.1 après 30, 60 et 90 epochs



Learning Rate Decay : Linéaire

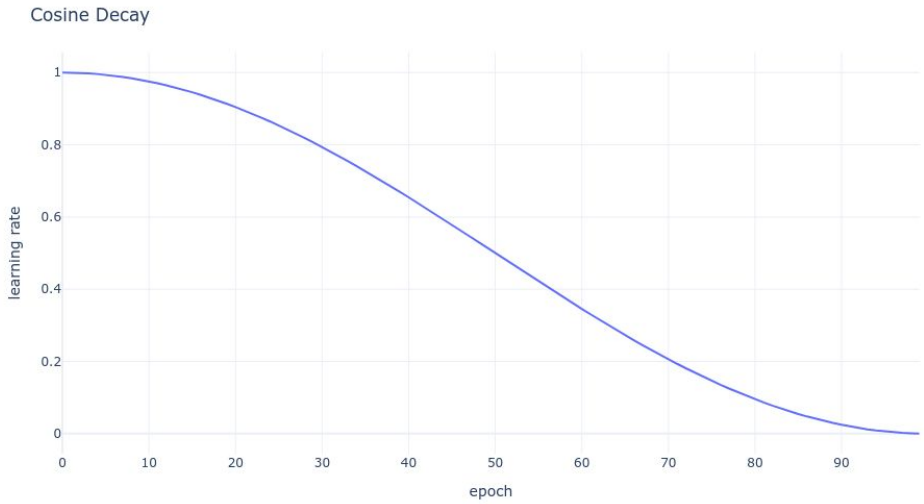
Idée : Réduire de manière régulière le learning rate entre une valeur initiale et 0.



$$\alpha_t = \alpha_0(1 - \frac{t}{T})$$

Learning Rate Decay : Cosinus

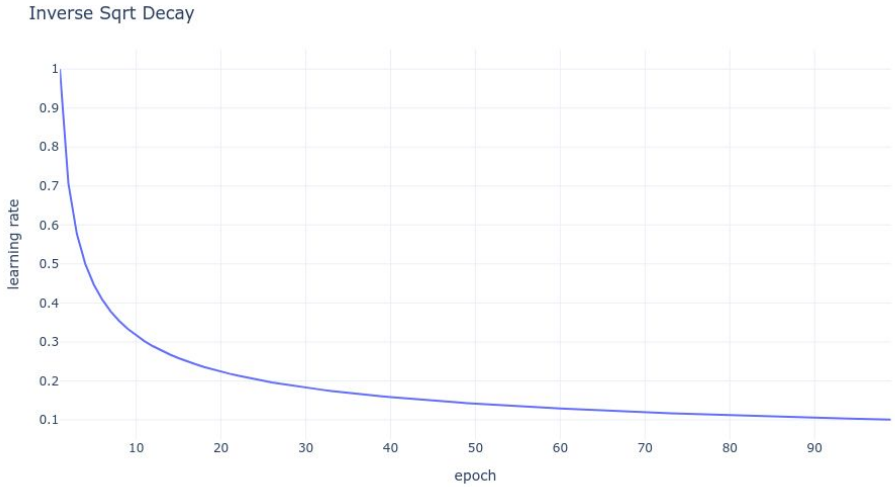
Idée : Réduire petit à petit le learning rate, d'une valeur initiale à 0. On va plus lentement au début et à la fin



$$\alpha_t = \frac{1}{2}\alpha_0(1 + cos(\frac{t\pi}{T}))$$

Learning Rate Decay : Racine carrée inversée

Idée : Réduire petit à petit le learning rate, d'une valeur initiale à 0. On va plus vite au début et très lentement à la fin



$$\alpha_t = \frac{\alpha_0}{\sqrt{t}}$$

Learning Rate Decay : Constante

Idée : On ne change pas la valeur !

$$\alpha_t = \alpha_0$$

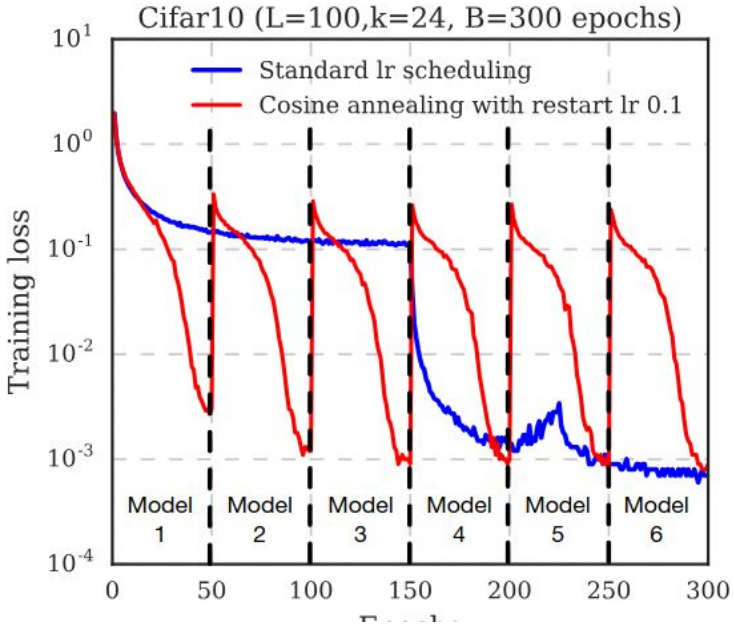
Learning Rate Warmup

Idée : Utiliser une stratégie différente au début de l'entrainement

Par exemple : Utiliser un learning rate constant pendant X itération avant de faire un cosine decay.

Learning Rate Restart

Idée : Recommencer le planificateur de zéro à intervalle régulier.



Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., & Weinberger, K. Q. (2017). Snapshot ensembles: Train 1, get m for free

PyTorch

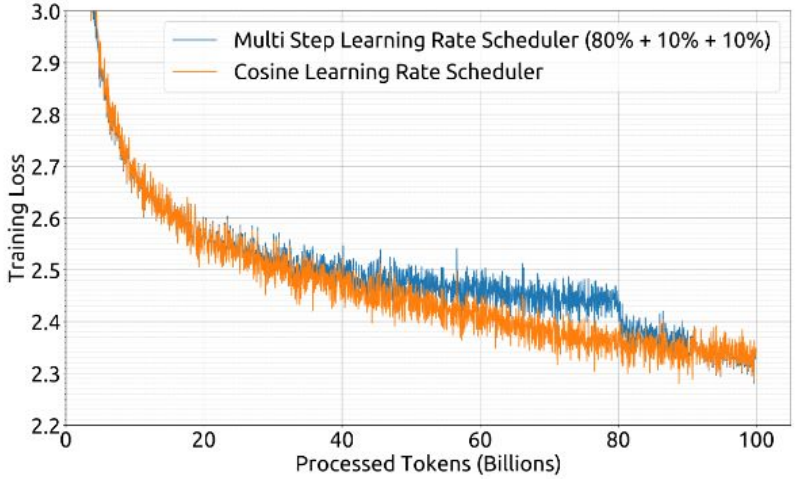
`lr_scheduler.LRScheduler`	Adjusts the learning rate during optimization.
`lr_scheduler.LambdaLR`	Sets the initial learning rate.
`lr_scheduler.MultiplicativeLR`	Multiply the learning rate of each parameter group by the factor given in the specified function.
`lr_scheduler.StepLR`	Decays the learning rate of each parameter group by gamma every step_size epochs.
`lr_scheduler.MultiStepLR`	Decays the learning rate of each parameter group by gamma once the number of epoch reaches one of the milestones.
`lr_scheduler.ConstantLR`	Multiply the learning rate of each parameter group by a small constant factor.

Exemples

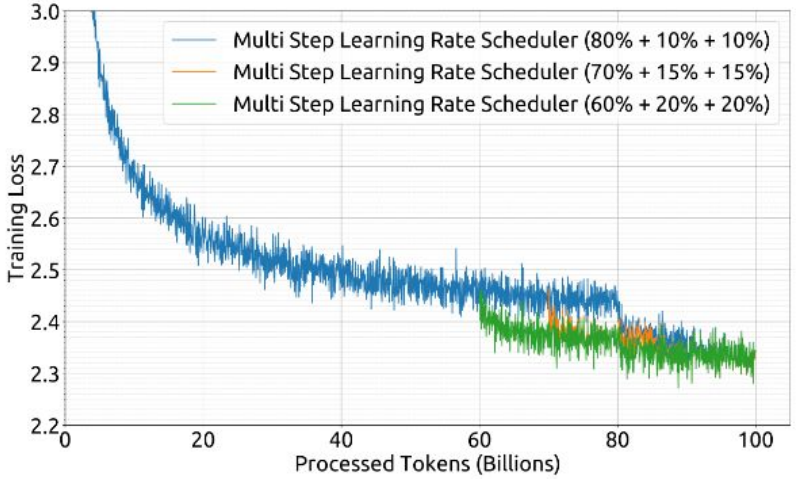
LLaMa 3 : “We pre-train Llama 3 405B using AdamW with a peak learning rate of **8 × 10−5** , a **linear warm up of 8,000 steps** , and a **cosine learning rate schedule decaying** to 8 × 10−7 over 1,200,000 steps”

DeepSeek: “A **multi-step learning rate scheduler** is employed during pre-training instead of the typical cosine scheduler. Specifically, the learning rate of the model reaches its maximum value after 2000 **warmup** steps, and then decreases to 31.6% of the maximum value after processing 80% of the training tokens. It further reduces to 10% of the maximum value after 90% of the tokens.”

Exemples - DeepSeek



(a) Multi-step v.s. cosine learning rate decay



(b) Different proportions of multi-step stages

Choix des hyperparamètres

Méthodes classiques

- Grid Search
- Random Search

Très coûteux à mettre en place suivant le dataset !

Approche pratique

1. Vérifier la perte initiale

Sans weight decay, vérifier que la loss ne fait pas n'importe quoi à l'initialisation.

Approche pratique

1. Vérifier la perte initiale
2. Overfit sur un petit échantillon

Est-ce que nous sommes capables d'overfit et d'atteindre 100% d'accuracy sur un petit échantillon ? Jouer avec le modèle, le learning rate, l'initialisation de poids, ... Pas de régularisation (on veut overfit)

La loss ne descend pas ? LR trop bas, mauvaise initialisation

La loss va à l'infini/NaN ? LR trop haut, mauvaise initialisation

Approche pratique

1. Vérifier la perte initiale
2. Overfit sur un petit échantillon
3. Trouver un LR qui fait descendre la loss

Avec l'architecture précédente, sur toutes les données d'entraînement, un peu de weight decay. Le LR doit descendre après 100 itérations (pas epoch).

LR classiques : 1e-1, 1e-2, 1e-3, 1e-4

Approche pratique

1. Vérifier la perte initiale
2. Overfit sur un petit échantillon
3. Trouver un LR qui fait descendre la loss
4. Faire une petite grid search autour des paramètres trouvés précédemment

Sur 1 à 5 époques, sur le LR et le weight decay.

Valeurs classiques de weight decay : 1e-4, 1e-5, 0

Approche pratique

1. Vérifier la perte initiale
2. Overfit sur un petit échantillon
3. Trouver un LR qui fait descendre la loss
4. Faire une petite grid search autour des paramètres trouvés précédemment
5. Pour les configurations les plus prometteuses, poursuivre l'entraînement

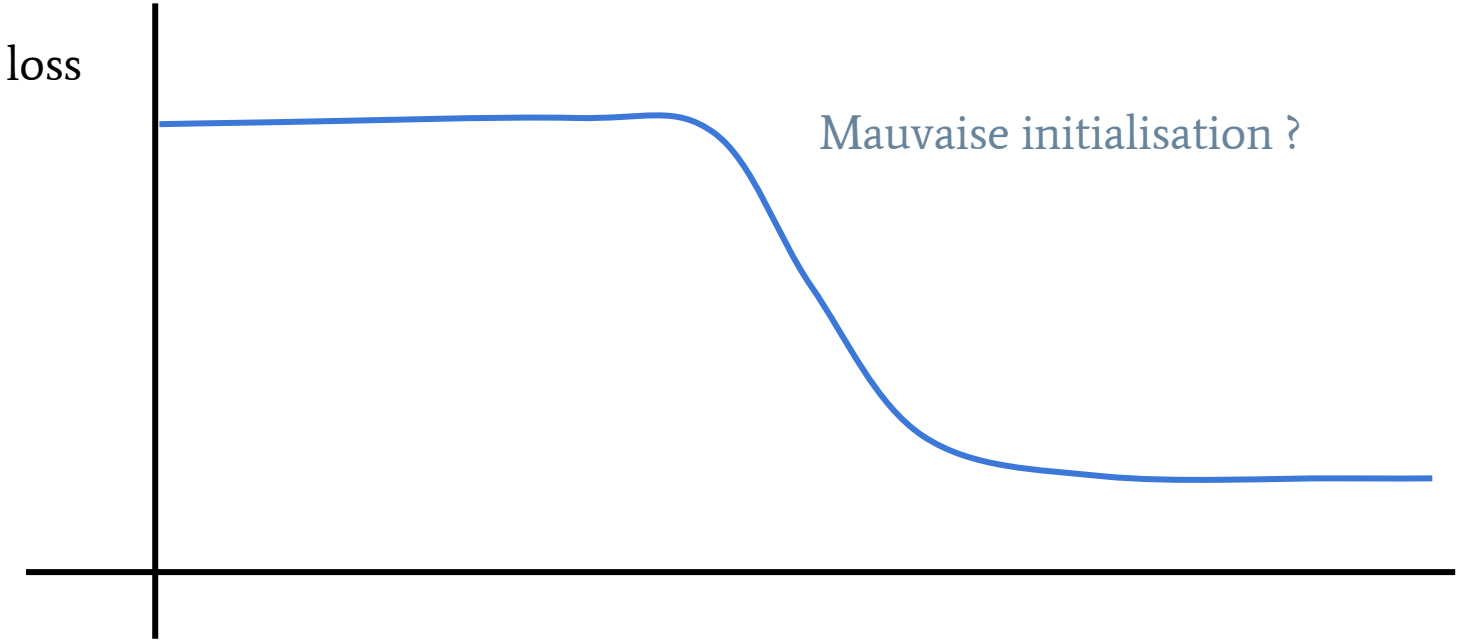
10-20 époques, sans LR decay.

Approche pratique

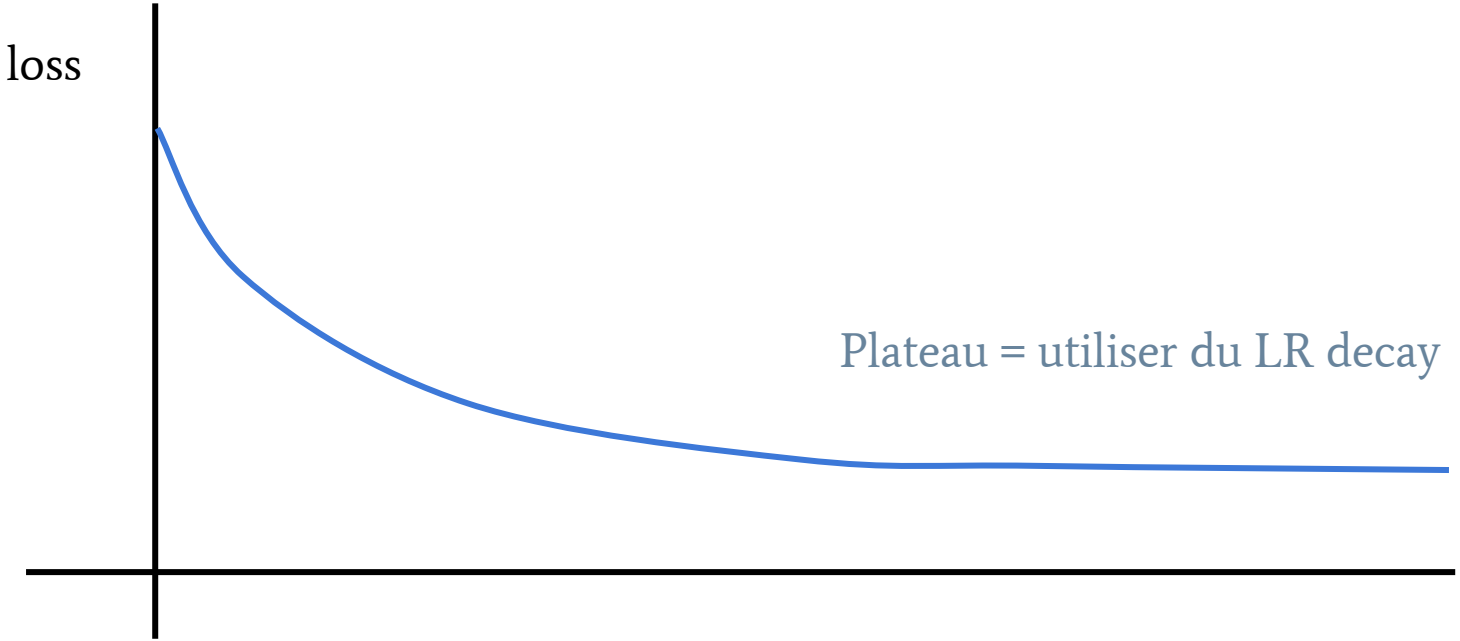
1. Vérifier la perte initiale
2. Overfit sur un petit échantillon
3. Trouver un LR qui fait descendre la loss
4. Faire une petite grid search autour des paramètres trouvés précédemment
5. Pour les configurations les plus prometteuses, poursuivre l'entraînement
6. Regarder les courbes d'apprentissage

On peut afficher la loss toutes les N itérations. La sortie peut être bruitée, mais on peut la lisser (c.f. tensorboard).

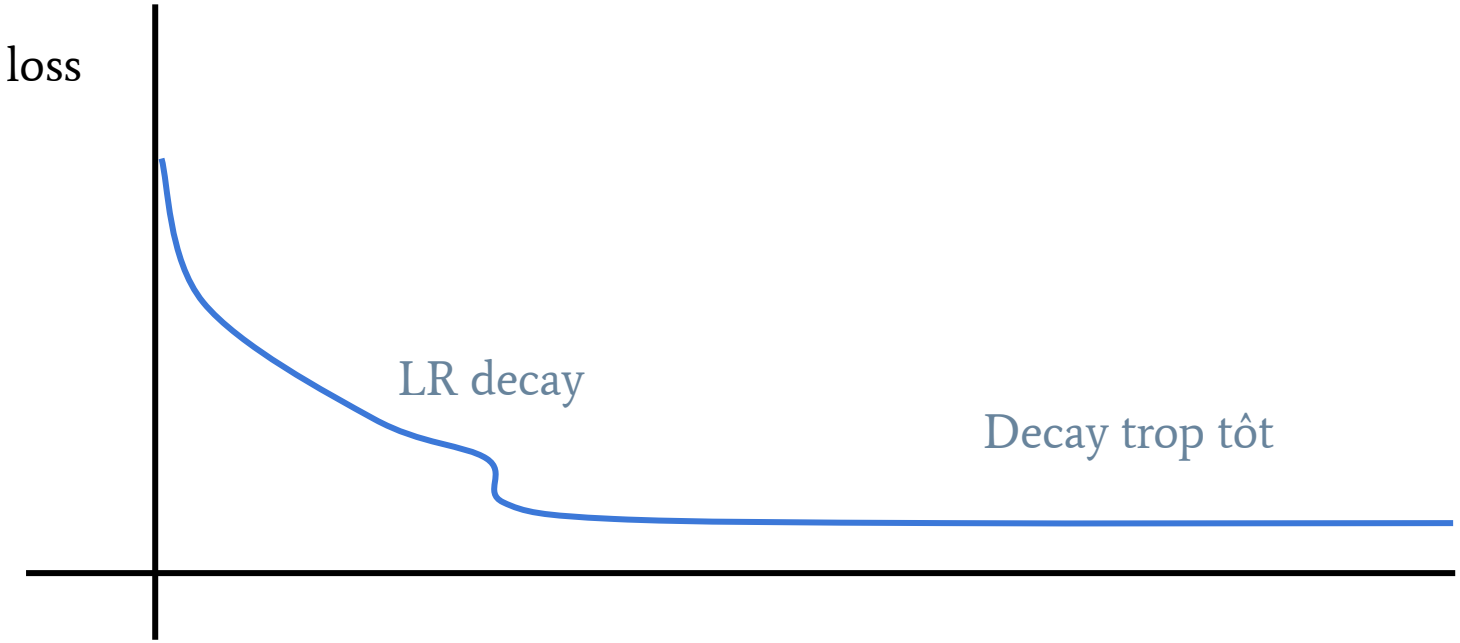
Courbes d'apprentissage



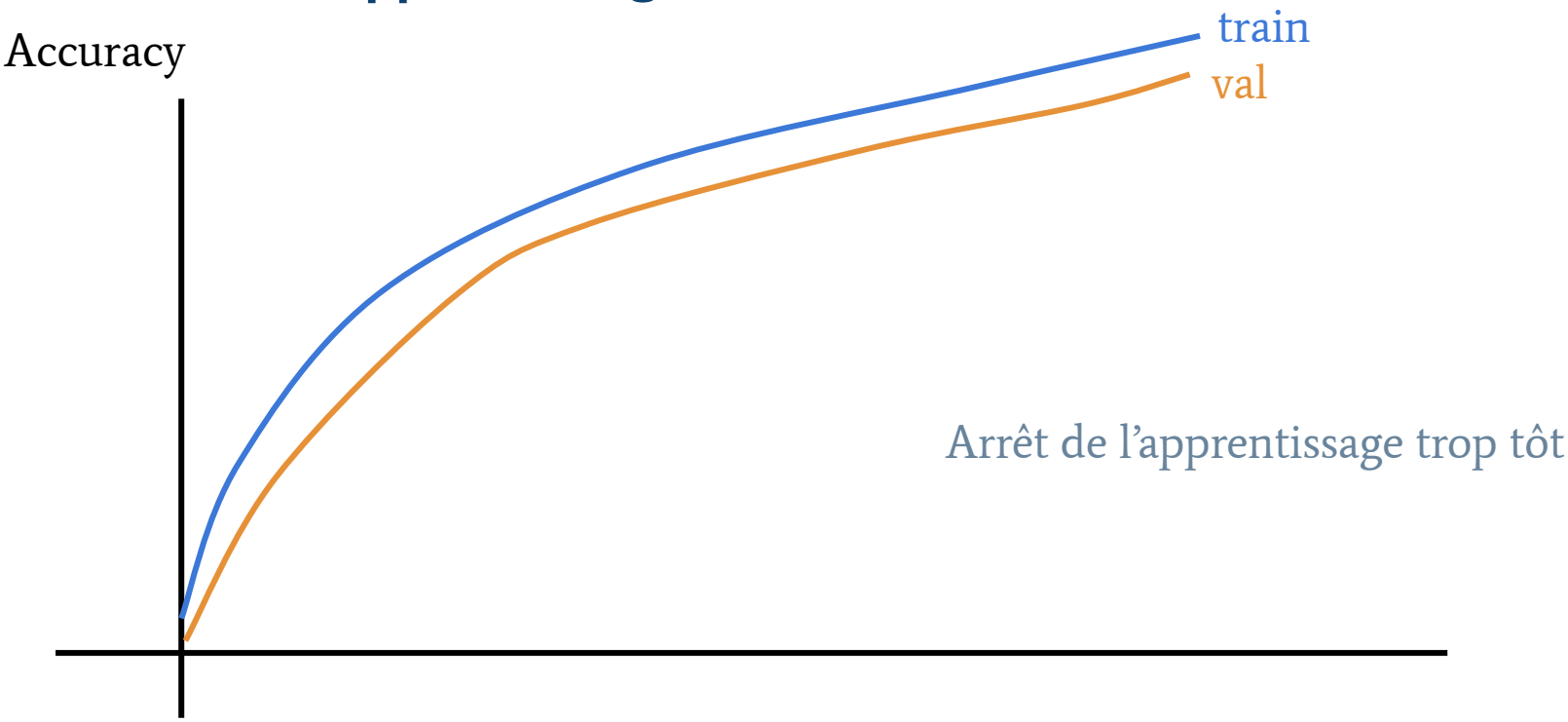
Courbes d'apprentissage



Courbes d'apprentissage

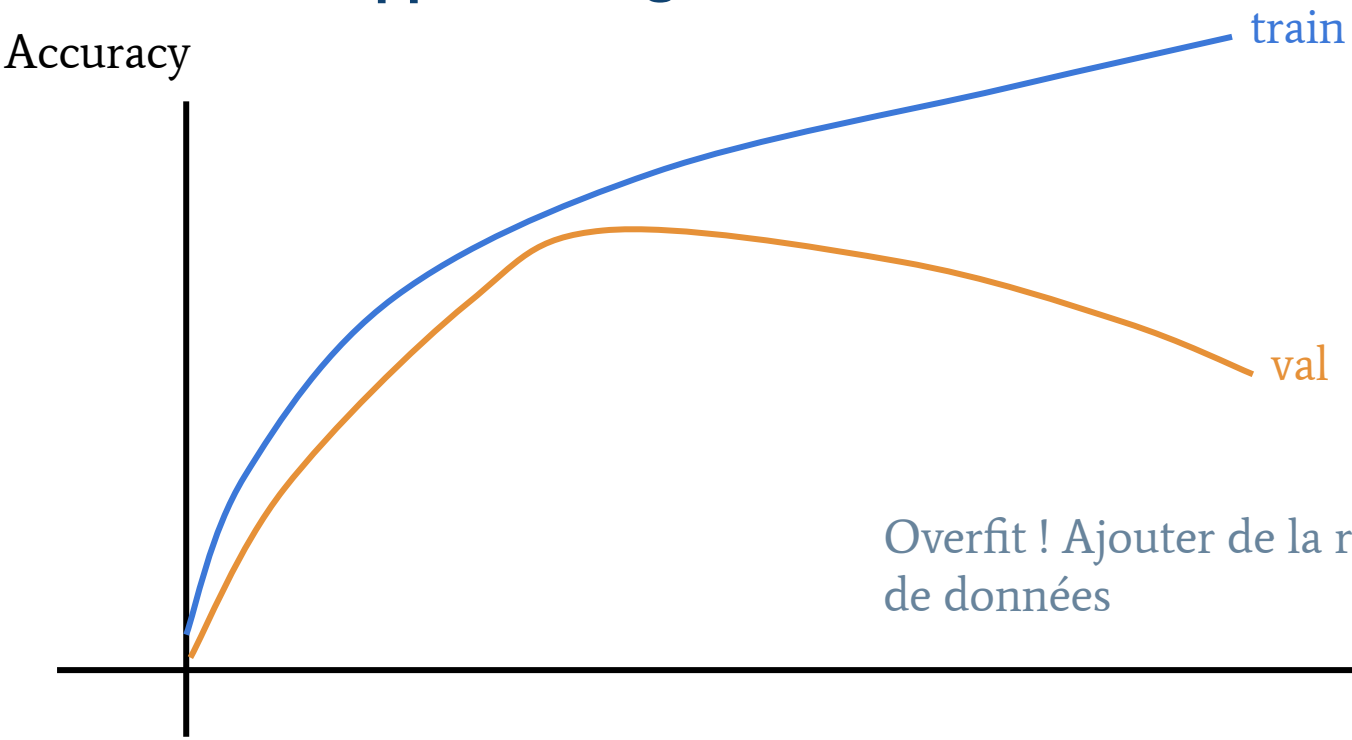


Courbes d'apprentissage



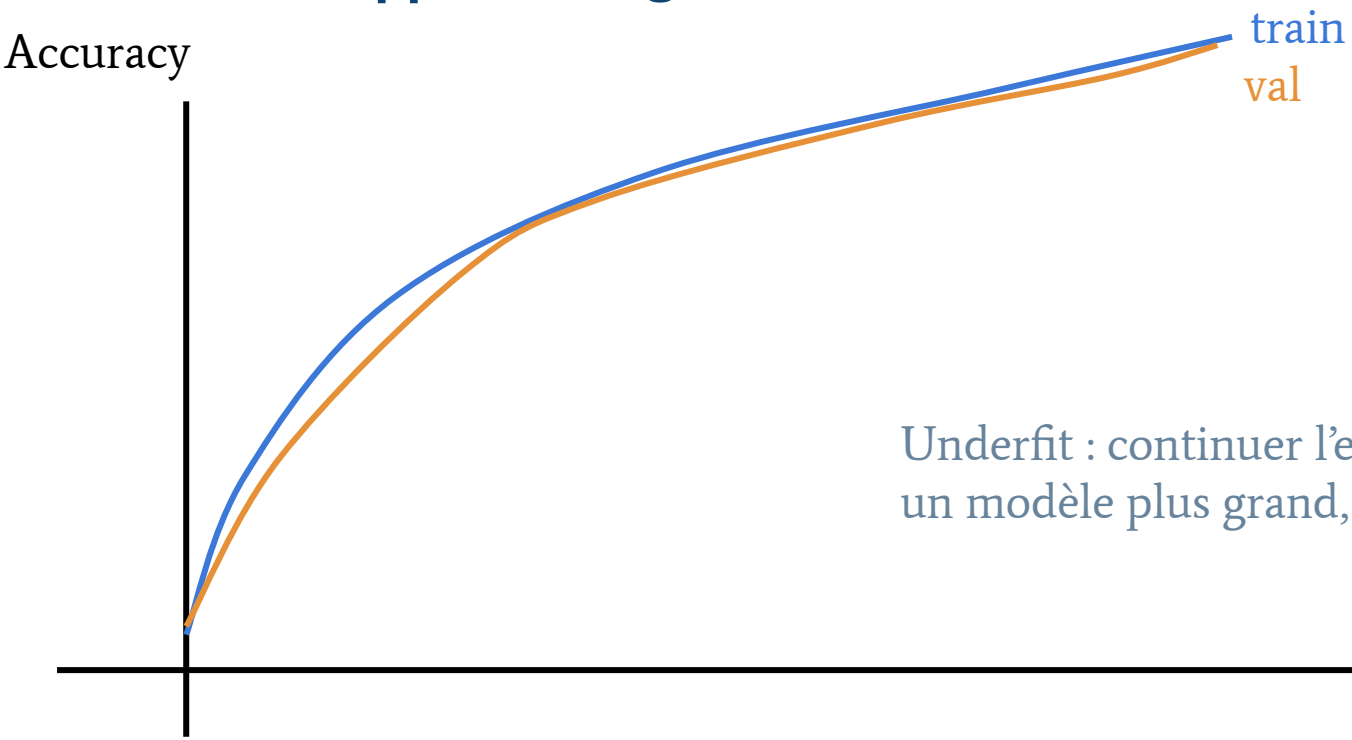
Arrêt de l'apprentissage trop tôt

Courbes d'apprentissage



Overfit ! Ajouter de la régularisation et plus de données

Courbes d'apprentissage



Underfit : continuer l'entraînement, utiliser un modèle plus grand, plus grand LR

Approche pratique

1. Vérifier la perte initiale
2. Overfit sur un petit échantillon
3. Trouver un LR qui fait descendre la loss
4. Faire une petite grid search autour des paramètres trouvés précédemment
5. Pour les configurations les plus prometteuses, poursuivre l'entraînement
6. Regarder les courbes d'apprentissage
7. GOTO 5.

Méthodes ensemblistes

Modèles ensemblistes

1. Entraîner des modèles indépendamment
2. Faire voter/prendre la moyenne de probabilités

On peut gagner 2% de performance gratuitement.

On peut aussi utiliser un seul modèle, mais à différents moments de l'entraînement

Exemple - CommonsenseQA

Human		03/10/2019	88.9	
ALBERT+DESC-KCR (ensemble model)	Microsoft Cognitive Services Research	12/02/2020		83.3
ALBERT + SFR (ensemble)	Huawei Poisson Lab & BUAA	7/17/2021	81.8	
Albert+KD (ensemble model)	HIT-SCIR-QA	12/30/2020		80.9
ALBERT+DESC-KCR (single model)	Microsoft Cognitive Services Research	12/02/2020		80.7
ALBERT+KD (single model)	HIT-SCIR-QA	12/10/2020		80.3
ALBERT+HGN (ensemble model)	USC MOWGLI / INK Lab	12/28/2020		80.0
Albert + KCR(knowledge chosen by relations, single model)	ITNLP (Harbin Institute of Technology)	07/12/2020		79.5
UnifiedQA (single model)	Allen Institute for AI	04/23/2020	79.1	
Albert+Headhunter (single model)	Anonymous	12/04/2020	78.4	
Albert + MSKF	Anonymous	05/31/2021	78.3	
Albert + PathGenerator (ensemble model)	USC MOWGLI / INK Lab	05/14/2020		78.2
T5 (single model)	Allen Institute for AI	04/23/2020	78.1	
ALBERT+HGN (single model)	USC MOWGLI / INK Lab	12/28/2020		77.3
TeGBERT (single model)	Anonymous	07/22/2020		76.8
ALBERT (ensemble model)	Zhiyan Technology	12/18/2019	76.5	
QA-GNN (single)	Stanford University	11/29/2020		76.1

Transfer Learning

Idée Préconçue

Il faut beaucoup de données pour entraîner un réseau de neurone profond et performant.

Idée Préconçue

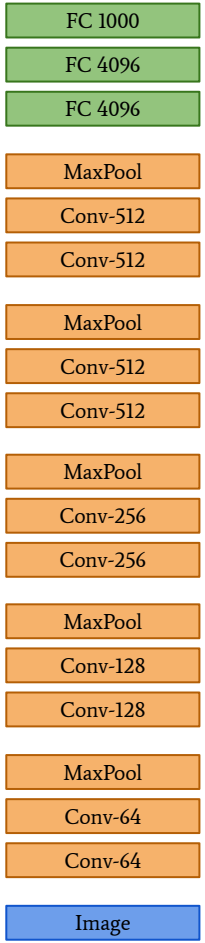
Transfer Learning

Idée : Réutiliser un réseau déjà entraîné sur une autre tâche pour l'adapter à notre problème

+ Apprentissage plus rapide et moins cher
+ Besoin de moins de données
+ Réutilisabilité des modèles
+ Amélioration des performances
+ Un bon modèle sur une tâche générique améliore les performances de pleins de tâches reliées

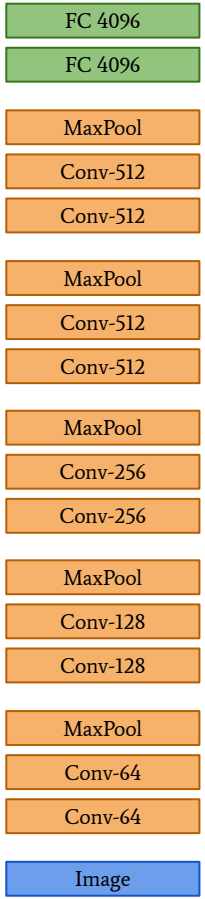
Étapes du transfer learning

1. Entraîner un modèle sur un (large) dataset
   a. Souvent fait pour nous
   b. Exemples : ResNet sur ImageNet, T5 sur C4



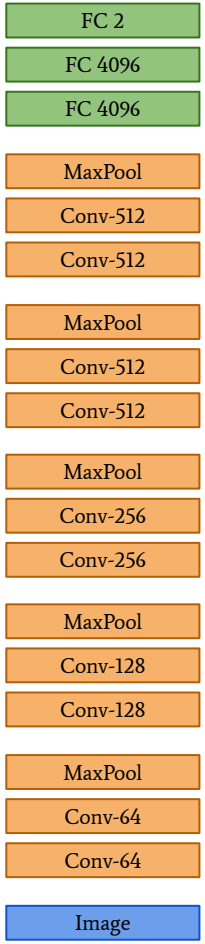
Étapes du transfer learning

1. Entraîner un modèle sur un (large) dataset
   a. Souvent fait pour nous
   b. Exemples : ResNet sur ImageNet, T5 sur C4
2. Sélectionner les partie à conserver
   a. En général, il faut enlever la dernière couche, spécifique au problème
   b. On conserve la partie qui “crée” les features



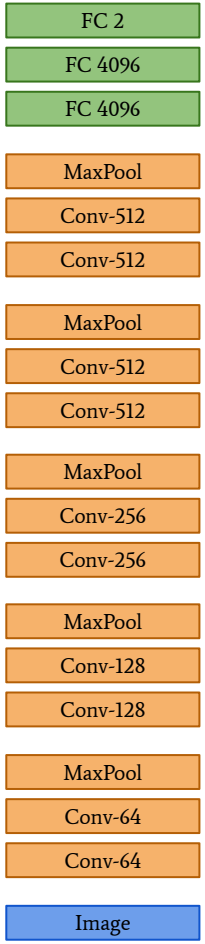
Étapes du transfer learning

1. Entraîner un modèle sur un (large) dataset
   a. Souvent fait pour nous
   b. Exemples : ResNet sur ImageNet, T5 sur C4
2. Sélectionner les partie à conserver
   a. En général, il faut enlever la dernière couche, spécifique au problème
   b. On conserve la partie qui “crée” les features
3. Modifier l’architecture pour notre problème
   a. En général, un MLP pour avoir une sortie ayant le bon nombre de neurones



Étapes du transfer learning

1. Entraîner un modèle sur un (large) dataset
   a. Souvent fait pour nous
   b. Exemples : ResNet sur ImageNet, T5 sur C4
2. Sélectionner les partie à conserver
   a. En général, il faut enlever la dernière couche, spécifique au problème
   b. On conserve la partie qui “crée” les features
3. Modifier l’architecture pour notre problème
   a. En général, un MLP pour avoir une sortie ayant le bon nombre de neurones
4. Choisir quels paramètres entraîner, et lesquels conserver (freeze)



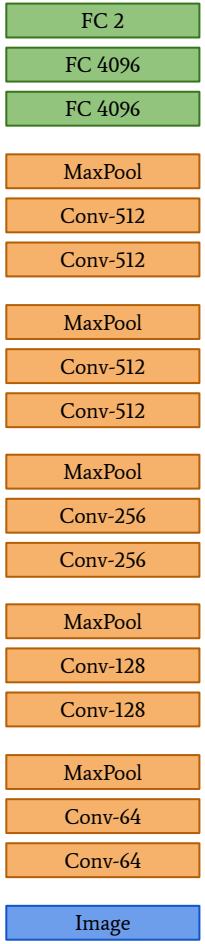
Freeze de paramètres

- Quand nous avons peu de données, il est souvent utile de ne s'entraîner que sur une partie des poids
  - Pas la peine de calculer les gradients = calculs plus rapides et moins de mémoire nécessaire
- Le reste ne change pas
- Plus la nouvelle tâche est similaire au problème original, plus on peut freeze de couches
  - Trop de différences + pas assez de données = mauvaises performances
- Pour certaines couches comme BatchNorm, on reste en mode inférence
- En Pytorch, il faut dire qu'on ne veut pas les gradients :

```
for param in model.parameters():
    param.requires_grad = False
```

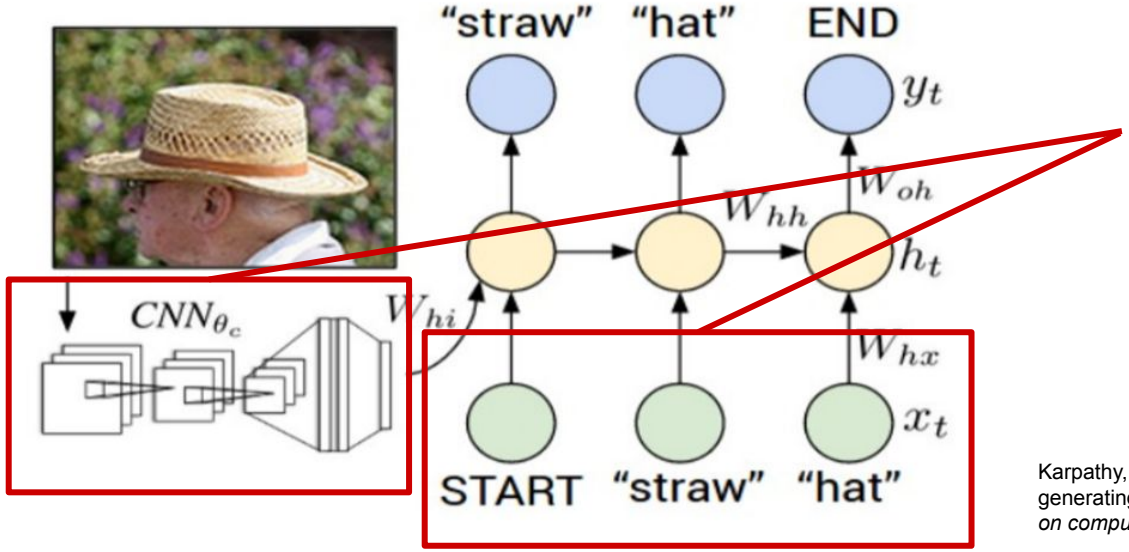
Étapes du transfer learning

1. Entraîner un modèle sur un (large) dataset
   a. Souvent fait pour nous
   b. Exemples : ResNet sur ImageNet, T5 sur C4
2. Sélectionner les partie à conserver
   a. En général, il faut enlever la dernière couche, spécifique au problème
   b. On conserve la partie qui “crée” les features
3. Modifier l’architecture pour notre problème
   a. En général, un MLP pour avoir une sortie ayant le bon nombre de neurones
4. Choisir quels paramètres entraîner, et lesquels conserver (freeze)
5. Finetuning : Continuer l’entraînement sur un nouveau dataset
   a. On prend souvent un LR dix fois plus petit que l’original



Le Transfer Learning est la norme

Il faut (presque) toujours préféré le transfer learning à un apprentissage de 0.



CNN pré-entrainé sur ImageNet
Mots pré-entrainés par Word2Vec

Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*

En résumé

- Les fonctions d'activation et leurs propriétés
- Planification du taux d'apprentissage
- Choisir les hyperparamètres
- Modèles ensemblistes
- Transfer learning