Réseaux de neurones : notions avancées

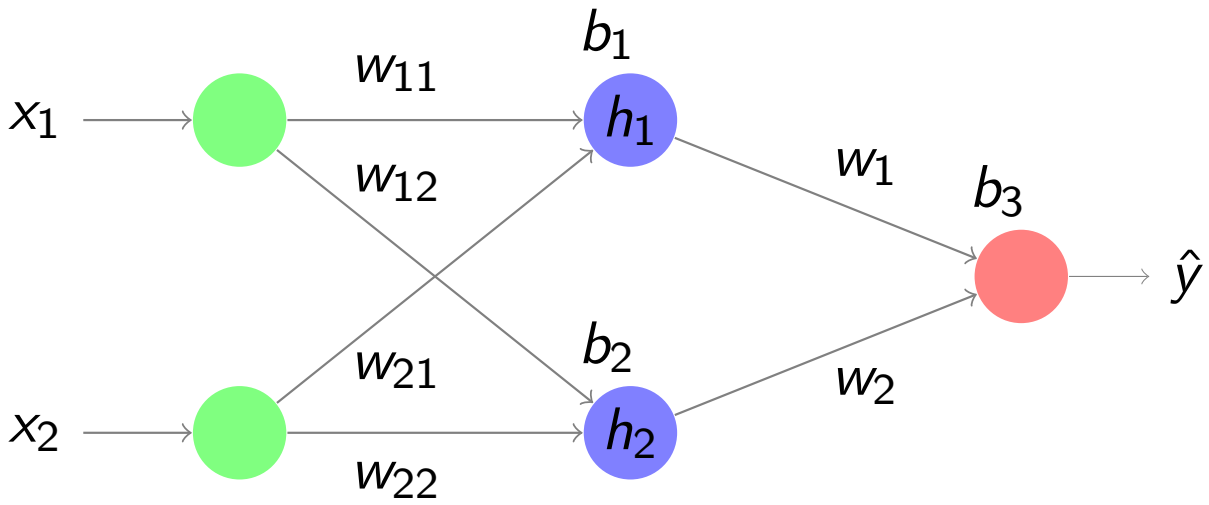
Apprentissage Profond

A. Carlier

2024

Plan du cours

Retour sur le calcul des gradients



$$\begin{cases} \hat{y} = \sigma(z) \text{ où } z = w_1 h_1 + w_2 h_2 + b_3 \\ h_1 = f(z_1) \text{ où } z_1 = w_{11} x_1 + w_{21} x_2 + b_1 \\ h_2 = f(z_2) \text{ où } z_2 = w_{12} x_1 + w_{22} x_2 + b_2 \end{cases}$$

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_j} \frac{\partial h_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

Retour sur le calcul des gradients

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_j} \frac{\partial h_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

On a :

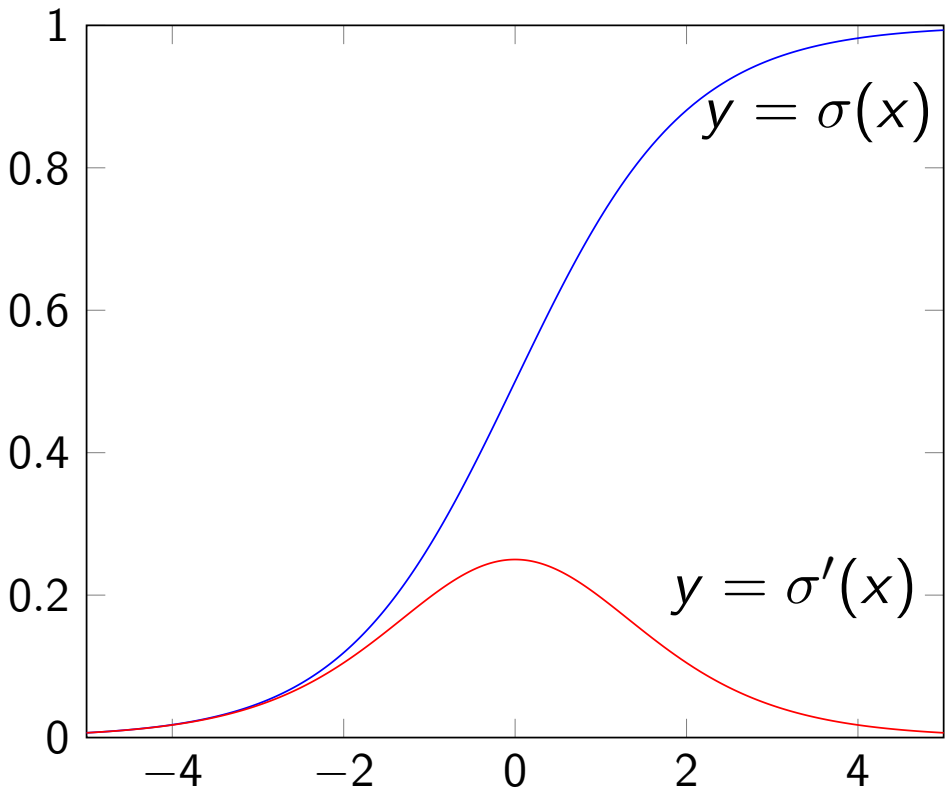
$$\frac{\partial z}{\partial h_j} = w_j$$

$$\frac{\partial h_j}{\partial z_j} = f'(z_j)$$

avec f' la dérivée de la fonction d'activation portée par le neurone.

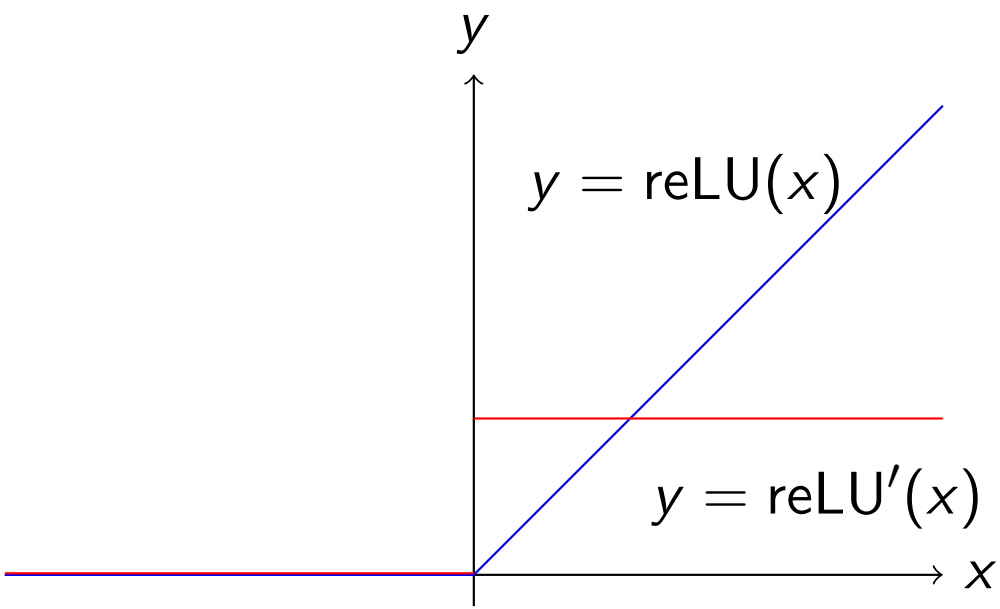
$$\frac{\partial z_j}{\partial w_{ij}} = x_i$$

Gros plan sur la fonction sigmoïde



La dérivée de la fonction sigmoïde est à valeurs dans $[0, \frac{1}{4}]$, ce qui diminue l'amplitude des gradients propagés à travers les couches du réseau de neurones. C'est le problème de l'**évanescence des gradients** (*vanishing gradients*).

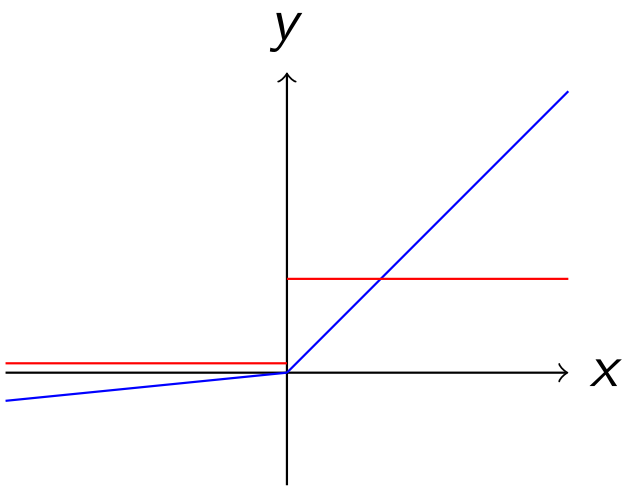
La fonction *rectified Linear Unit*



$$\text{reLU}(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{sinon} \end{cases} \tag{1}$$

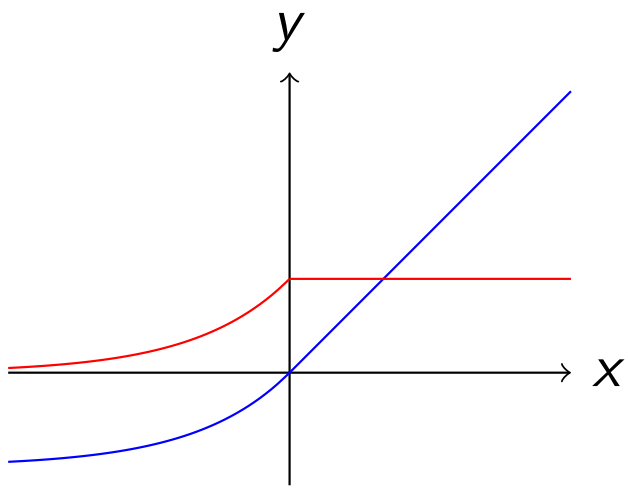
Le fait d'avoir un gradient constamment égal à 1 dans la zone activée améliore grandement la convergence.

D'autres variantes



$$\text{leakyReLU}(x) = \begin{cases} \alpha x & \text{si } x < 0 \\ x & \text{sinon} \end{cases}$$

Leaky Rectified Linear Unit



$$\text{eLU}(x) = \begin{cases} \alpha(e^x - 1) & \text{si } x < 0 \\ x & \text{sinon} \end{cases}$$

Exponential Linear Unit

Ces fonctions améliorent la convergence de la descente de gradient en limitant les occurences où le gradient est nul. De plus, en autorisant les activations négatives, le problème d'optimisation est mieux conditionné.

Plan du cours

Descente de gradient

Algorithme : Descente de gradient $(\mathcal{D}, \alpha)$
Initialiser $\theta^{\{0\}} \leftarrow 0$, $k \leftarrow 0$
TANT QUE pas convergence **FAIRE**
  POUR j de 1 à d **FAIRE**

$$\theta_j^{\{k+1\}} \leftarrow \theta_j^{\{k\}} - \alpha \frac{\partial J(\theta^{\{k\}})}{\partial \theta_j}$$

  FIN POUR
  $k \leftarrow k + 1$
FIN TANT QUE

Calculer le gradient exact est **coûteux** parce qu'il faut évaluer le modèle sur tous les m exemples de l'ensemble de données à chaque itération.

Descente de gradient stochastique

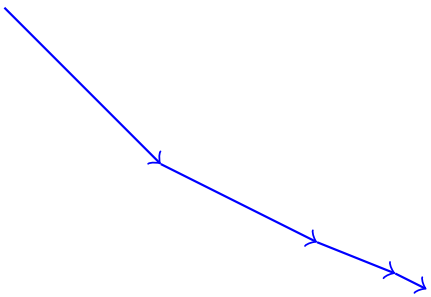
On peut en fait distinguer plusieurs alternatives pour l'algorithme de la descente de gradient :

- Algorithme **Batch** : utilise tous les m exemples de l'ensemble d'apprentissage pour calculer le gradient.
- Algorithme **Mini-batch** : approxime le gradient en le calculant en utilisant k exemples parmi les m échantillons de l'ensemble d'apprentissage, où $m >> k > 1$.
- Algorithme **Stochastique** : approche le gradient en le calculant sur un seul exemple ($k = 1$).

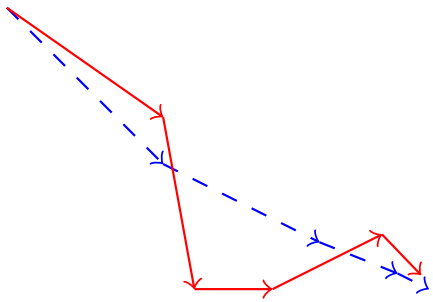
Par abus de langage, on utilise le terme de descente de gradient stochastique (SGD) y compris pour l'algorithme Mini-batch.

Descente de Gradient Stochastique

L'idée de la descente de gradient stochastique est que le gradient à calculer étant une espérance sur l'ensemble d'apprentissage, il est possible de l'estimer approximativement à l'aide d'un petit ensemble d'échantillons, appelé *mini-batch*. (C'est le même principe que pour les sondages d'opinion !)



Descente de gradient



Descente de gradient stochastique

Remarque : un mini-batch trop petit peut engendrer un bruit trop grand sur l'estimation du gradient et empêcher la convergence.

Variante de la descente de gradient

On parle d'**epoch** lorsque l'ensemble d'apprentissage a été visité entièrement pour le calcul des gradients.

Au final on a donc :

- Algorithme **Batch** : 1 itération par *epoch*.
- Algorithme **MiniBatch** : $\frac{m}{k}$ itérations par *epoch*.
- Algorithme **Stochastique** : m itérations par *epoch*.

Momentum

Pour éviter les problèmes liés à une stabilisation dans un minimum local, on ajoute un terme d'inertie (*momentum*).

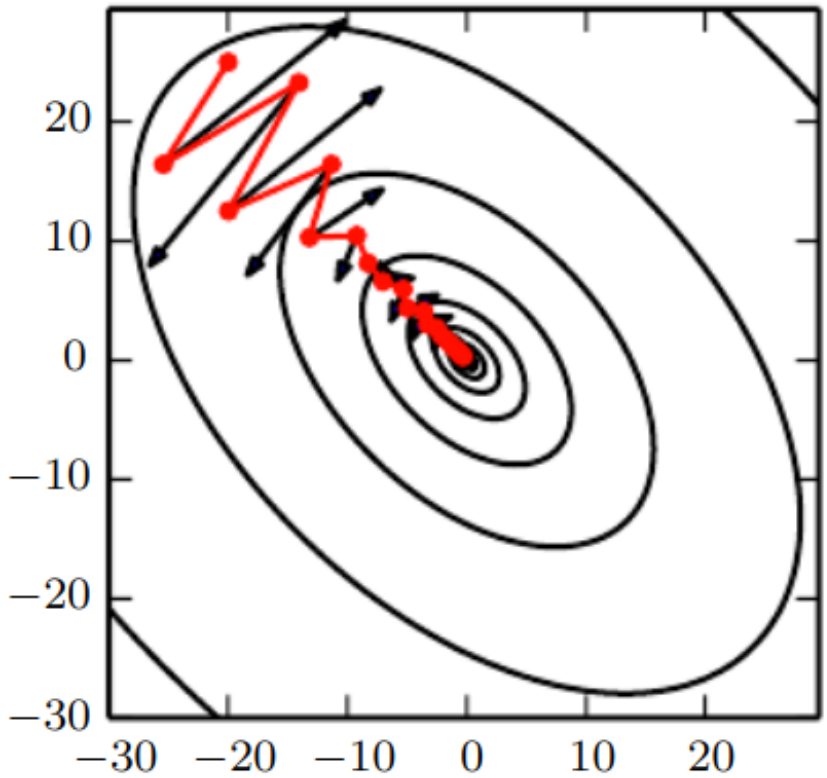


Image de [Goodfellow et al. 2015] Deep Learning

Momentum

En pratique, on adapte l'algorithme de descente du gradient, en remplaçant la mise à jour des paramètres

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}$$

par deux étapes :

$$v = \eta v - \alpha \frac{\partial J}{\partial \theta}$$

$$\theta \leftarrow \theta + v$$

où v (pour vélocité) désigne la direction dans laquelle les paramètres vont être modifiés. v prend en compte les gradients précédents via le paramètre η ($0 < \eta < 1$), qui quantifie l'importance relative des gradients précédents par rapport au gradient courant.

Optimiseurs améliorant la descente de gradient stochastique

Dans les espaces paramétriques de grande dimension, la topologie de la fonction objectif rend la descente de gradient parfois inefficace. On peut améliorer cette dernière en utilisant des optimiseurs adaptés.

L'optimiseur **AdaGrad** introduit une forme d'adaptation du taux d'apprentissage en accumulant les carrés des gradients précédents.

1. Calcul du gradient : $g = \frac{\partial J}{\partial \theta}$
2. Accumulation des gradients : $r = r + ||g||_2$
3. Mise à jour des paramètres : $\theta \leftarrow \theta - \frac{\alpha}{\sqrt{r}} g$

Optimiseurs améliorant la descente de gradient stochastique

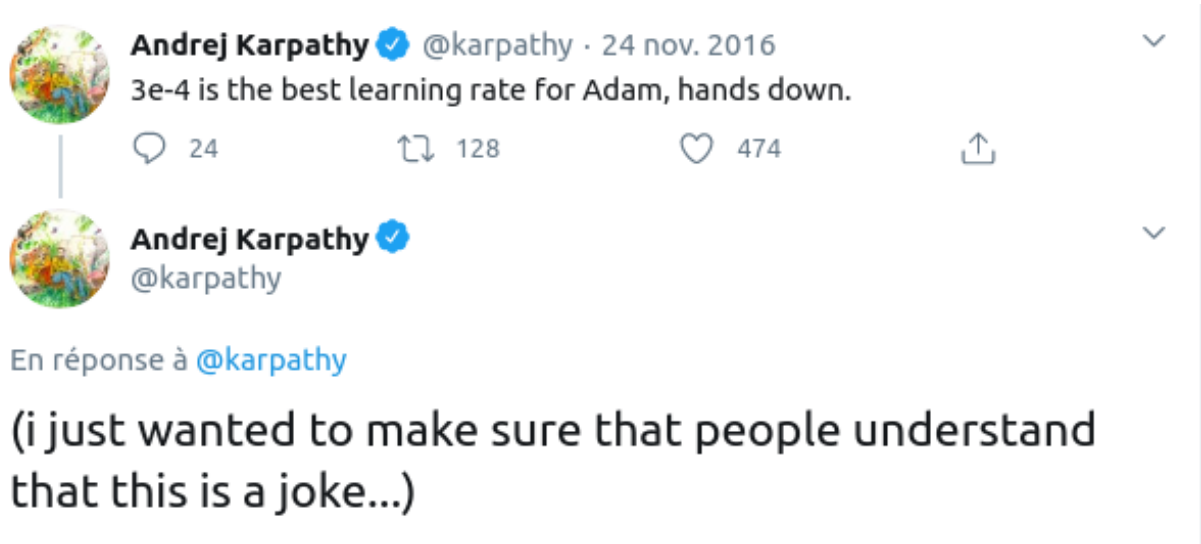
L'optimiseur **RMSProp** est presque identique à AdaGrad, mais l'impact des plus anciens gradients est altéré par un coefficient multiplicatif ρ inférieur à 1 (*weight decay*), ce qui améliore le comportement de l'algorithme dans le cas des bols allongés.

1. Calcul du gradient : $g = \frac{\partial J}{\partial \theta}$
2. Accumulation des gradients : $r = \rho r + (1 - \rho)||g||_2$
3. Mise à jour des paramètres : $\theta \leftarrow \theta - \frac{\alpha}{\sqrt{r}} g$

Enfin, l'optimiseur **Adam** est similaire à RMSProp, mais adapte également le momentum.

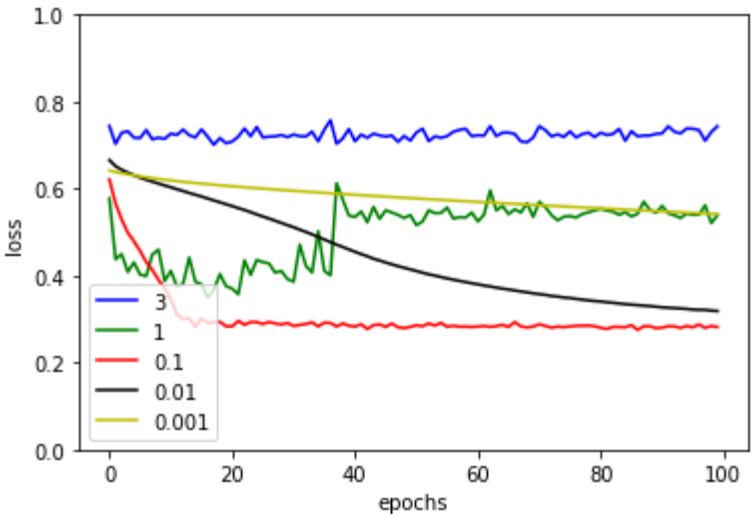
En pratique : choix de l'optimiseur

Adam est souvent un bon choix pour débuter (avec le taux d'apprentissage "magique")

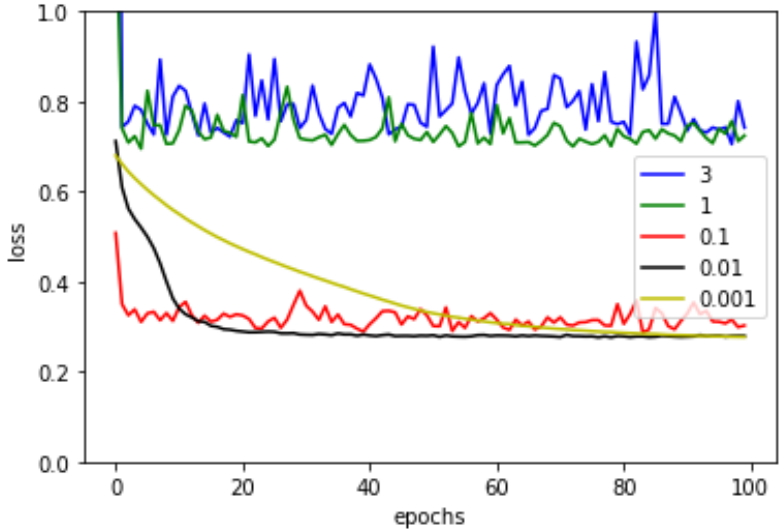


En pratique, les meilleurs résultats mis en avant dans les articles sont obtenus avec une simple descente de gradient stochastique, et une mise à jour programmée du taux d'apprentissage (cyclique, cosinus, etc.)

SGD vs. Adam sur un exemple simple



SGD

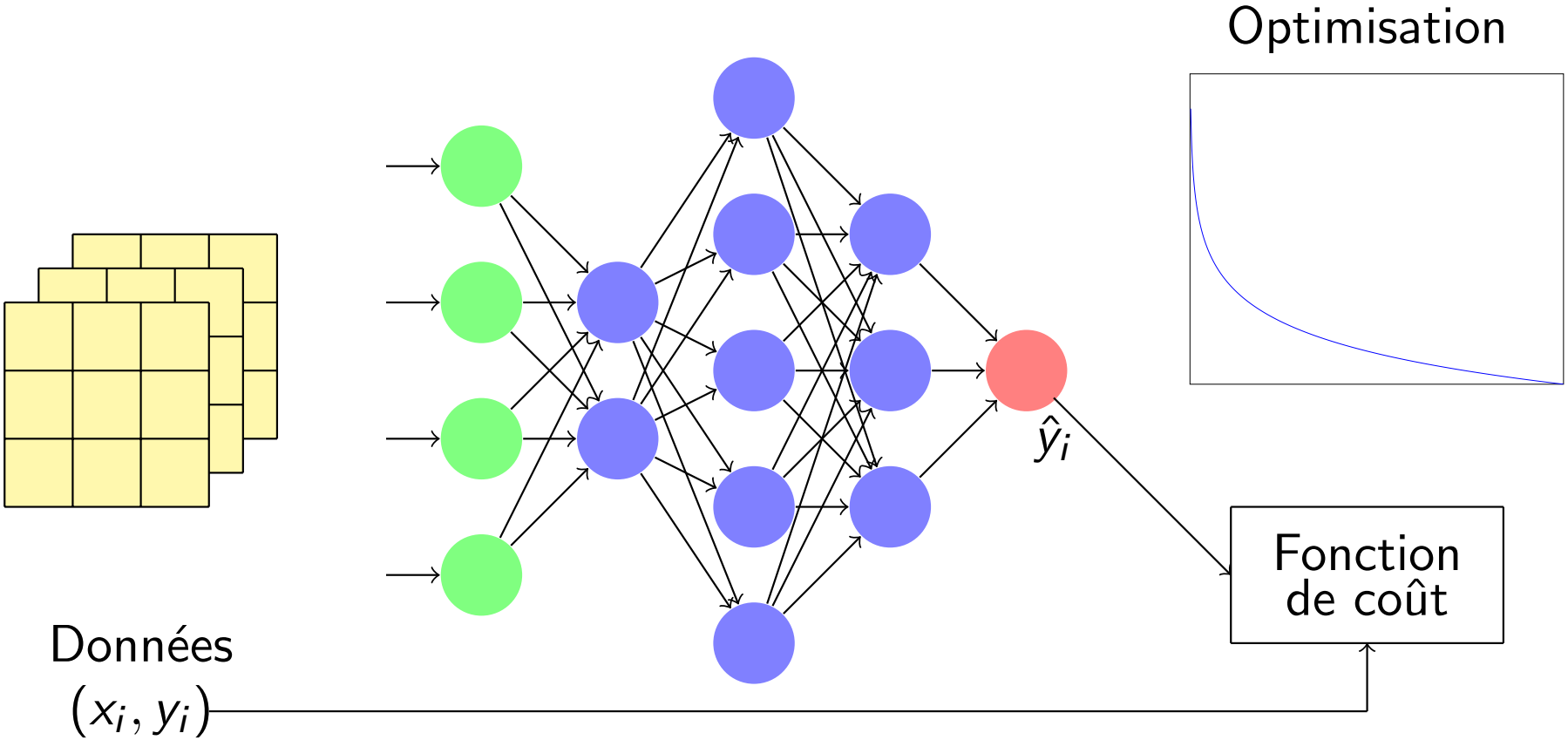


Adam

(Evolution de la perte d'apprentissage au cours de l'entraînement, pour différents taux d'apprentissage, avec SGD et Adam)

Plan du cours

Vue d'ensemble



profondeur, #neurones, activation α, momentum

$w_{i,j}, b_k$

hyperparamètres et paramètres

Risque empirique et Risque espéré

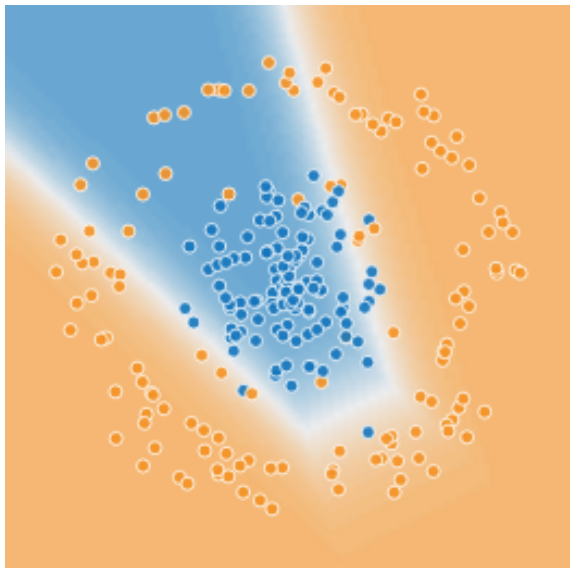
Risque empirique : erreur de prédiction moyenne sur l'ensemble d'apprentissage.
Risque espéré (ou risque de généralisation) : erreur de prédiction moyenne sur la population cible... **Inconnu !**

L'objectif principal d'un algorithme d'apprentissage est de minimiser le **risque espéré**, mais nous ne sommes capables d'évaluer que le **risque empirique**.

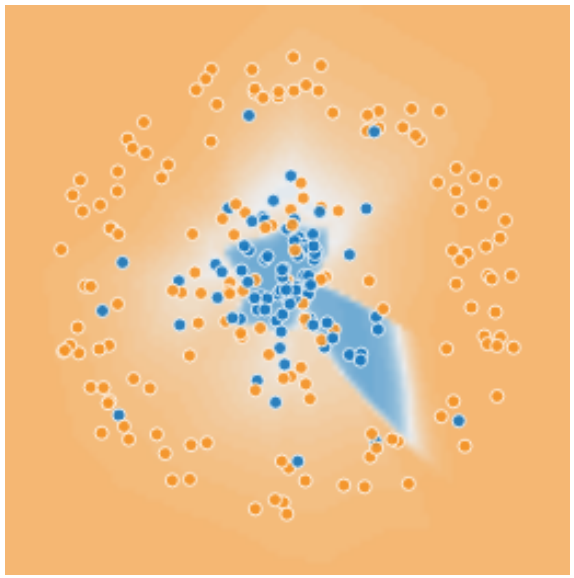
Sous/Sur-apprentissage

On parle de **sous-apprentissage** (*underfitting*) lorsque le modèle appris explique trop mal l'ensemble d'apprentissage.

On parle de **sur-apprentissage** (*overfitting*) lorsque le modèle appris explique à l'inverse trop bien l'ensemble d'apprentissage ; ce modèle se généralise alors mal à la population cible.



Sous-apprentissage



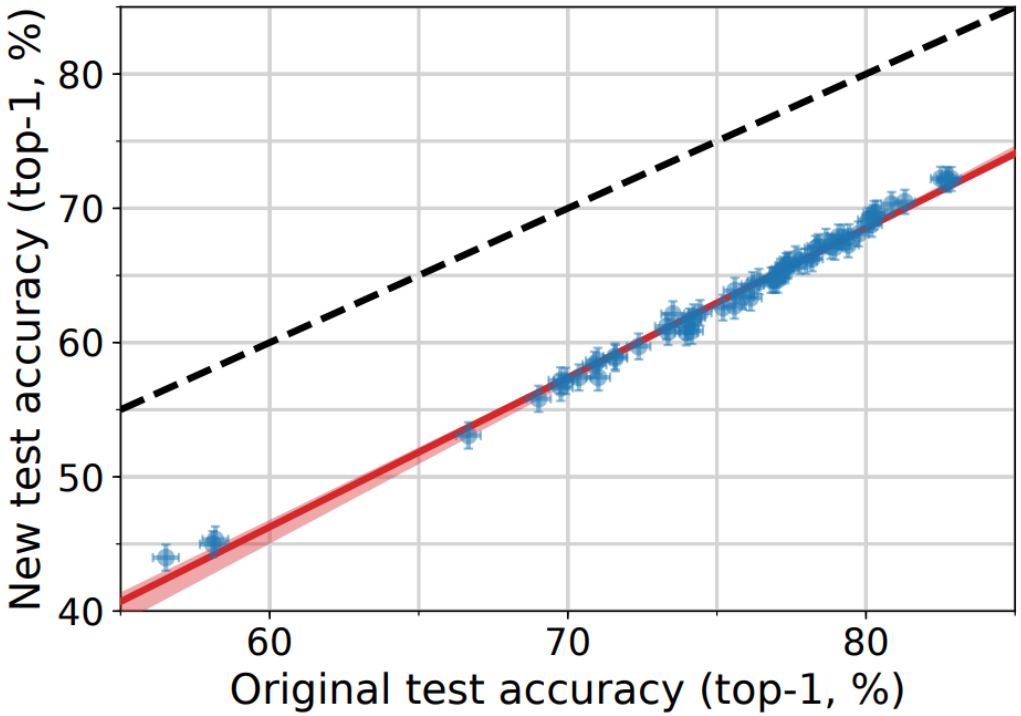
Sur-apprentissage

Sous/Sur-apprentissage

Solution : gérer 2 ensembles de données distincts

- l'ensemble **d'apprentissage**, sur lequel on va effectuer la descente de gradient et donc optimiser les **paramètres** du modèle.
- l'ensemble de **test** : qui détermine la performance objective du réseau de neurones.

L'importance de l'ensemble de validation



Les classifieurs sont en moyenne 10 à 15% plus performants sur l'ensemble de test original d'ImageNet que sur un nouvel ensemble généré par la même procédure.

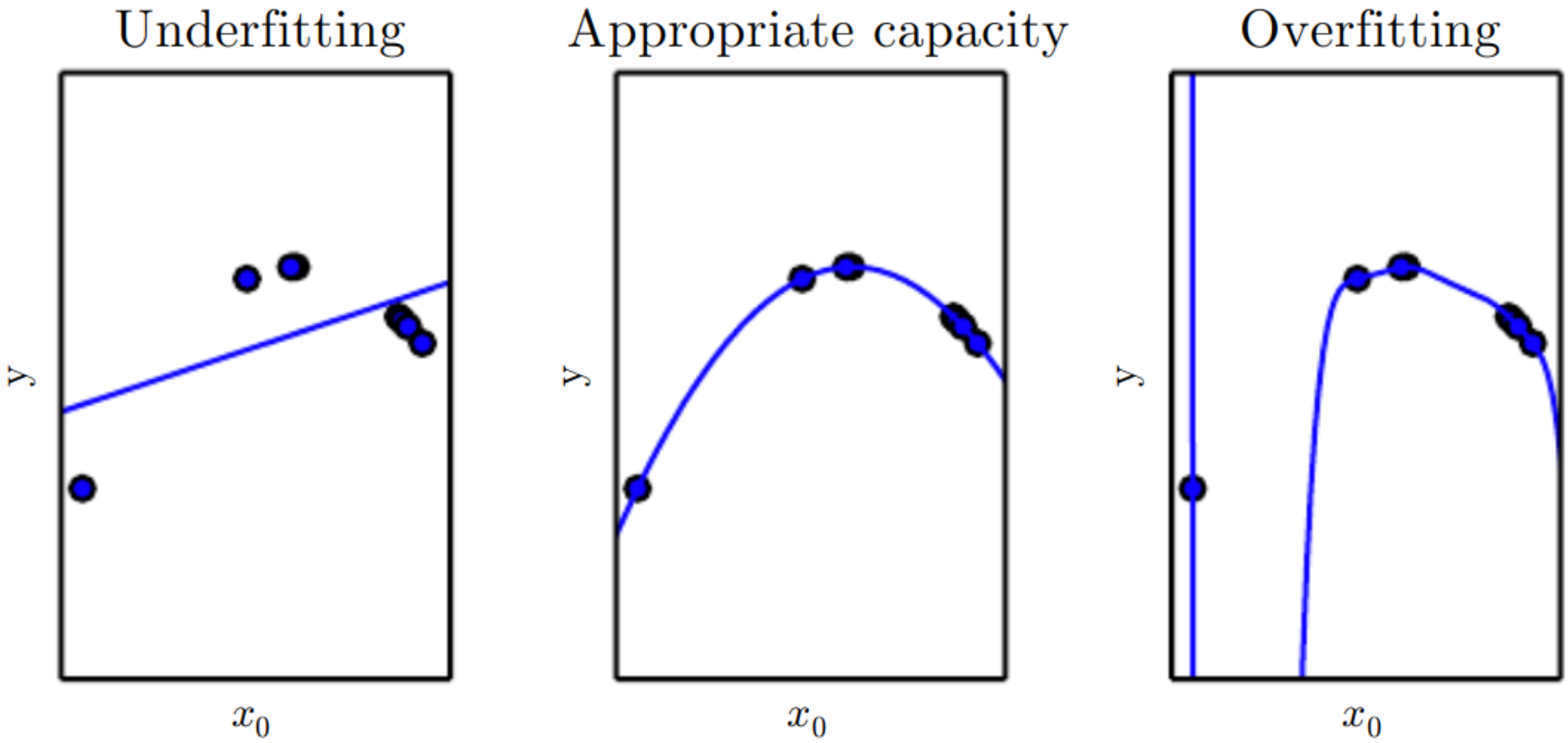
[Recht et al. 2019] Do ImageNet Classifiers Generalize to ImageNet ?

Sous/Sur-apprentissage

Méthodologiquement, il est donc plus juste de gérer 3 ensembles de données distincts

- l'ensemble **d'apprentissage**, sur lequel on va effectuer la descente de gradient et donc optimiser les **paramètres** du modèle.
- l'ensemble de **validation** : qui va nous fournir une estimation de l'erreur de généralisation, et nous permettre d'optimiser les **hyperparamètres** du modèle.
- l'ensemble de **test** : qui détermine la performance objective du réseau de neurones.

Sous/Sur-apprentissage : le problème de la **capacité** du modèle

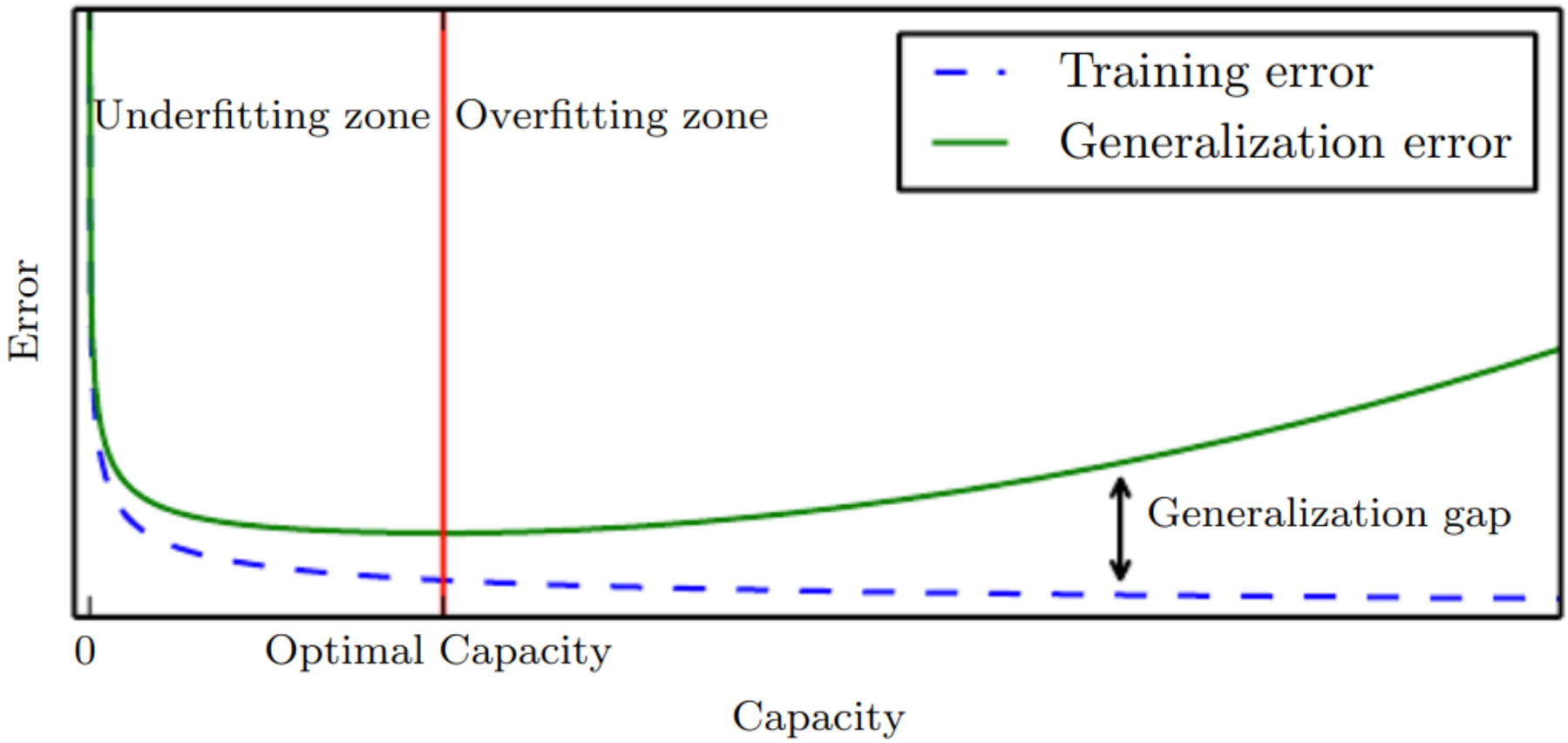


Un modèle de trop faible capacité engendre du sous-apprentissage.
Un modèle de trop large capacité engendre du sur-apprentissage.

Image de [Goodfellow et al. 2015] Deep Learning

Sous/Sur-apprentissage

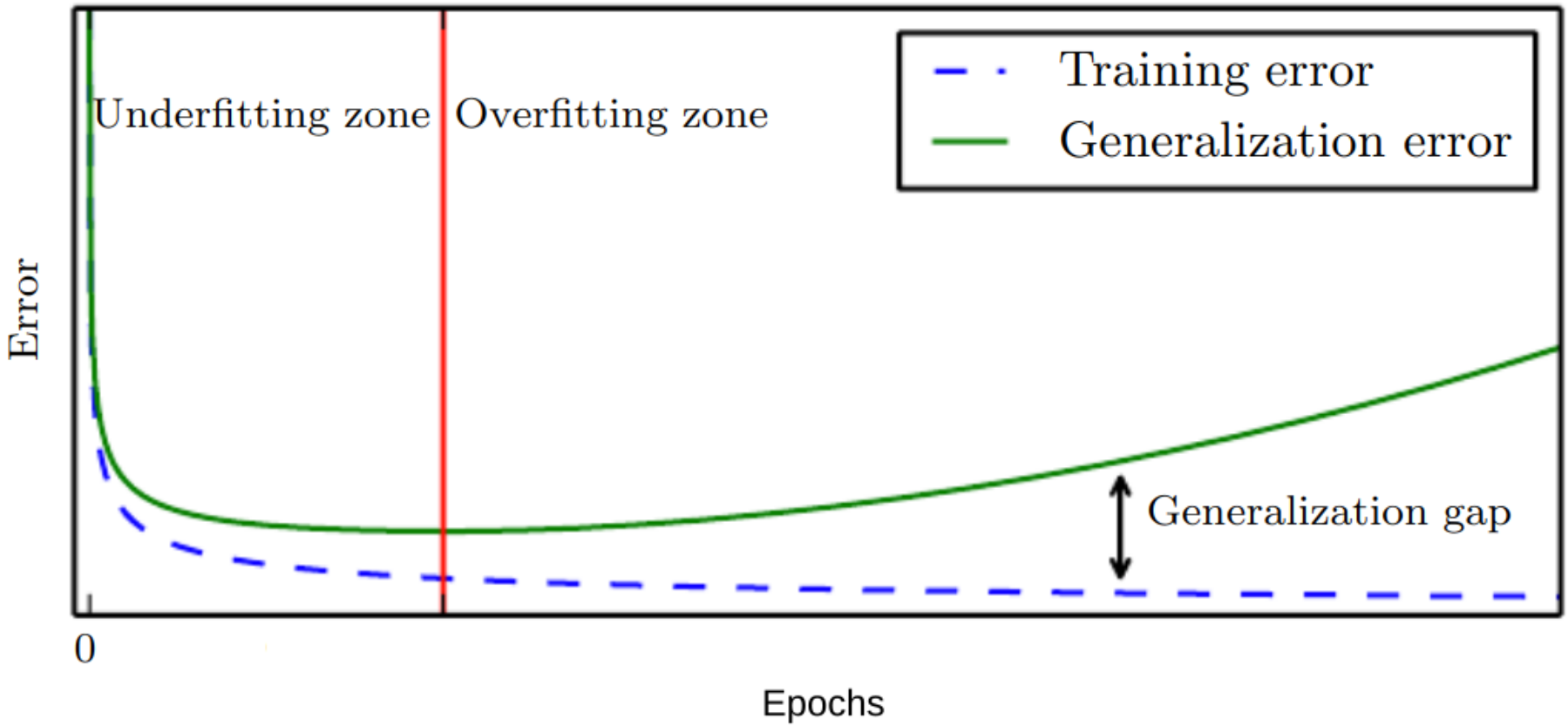
Illustration graphique :



Il est nécessaire d'identifier la capacité (profondeur, nombre de neurones) du réseau qui induit un apprentissage optimal.

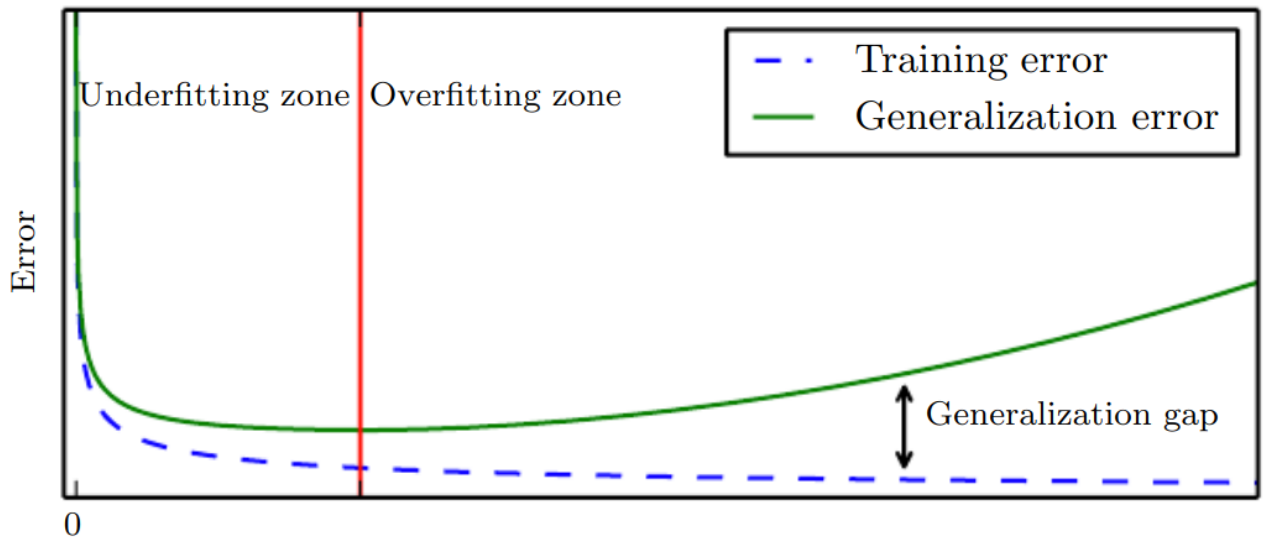
Image de [Goodfellow et al. 2015] Deep Learning

Sous/Sur-apprentissage



La courbe ci-dessus est valable pour la **capacité** du réseau, mais aussi pour la **durée** d'entraînement.

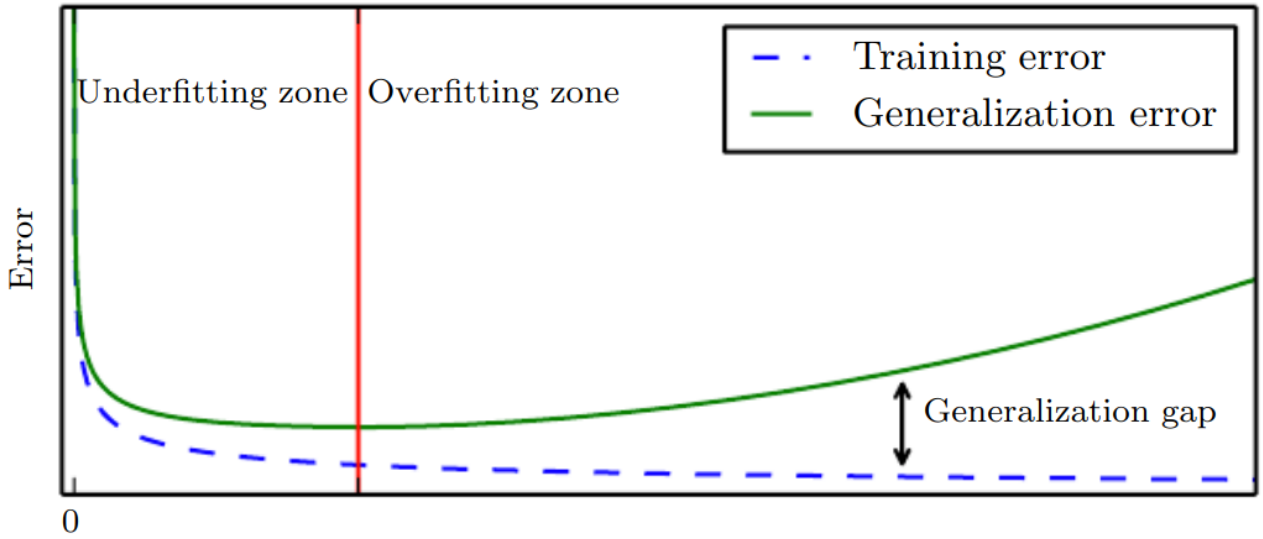
Que faire en cas de sous-apprentissage ?



On peut :

- Augmenter la capacité du réseau : augmenter la profondeur, ajouter des neurones dans les couches cachées, changer d'architecture.
- Améliorer l'entraînement : augmenter le nombre d'*epochs*, changer le taux d'apprentissage, ajouter du momentum, éventuellement changer d'optimiseur.

Que faire en cas de sur-apprentissage ?



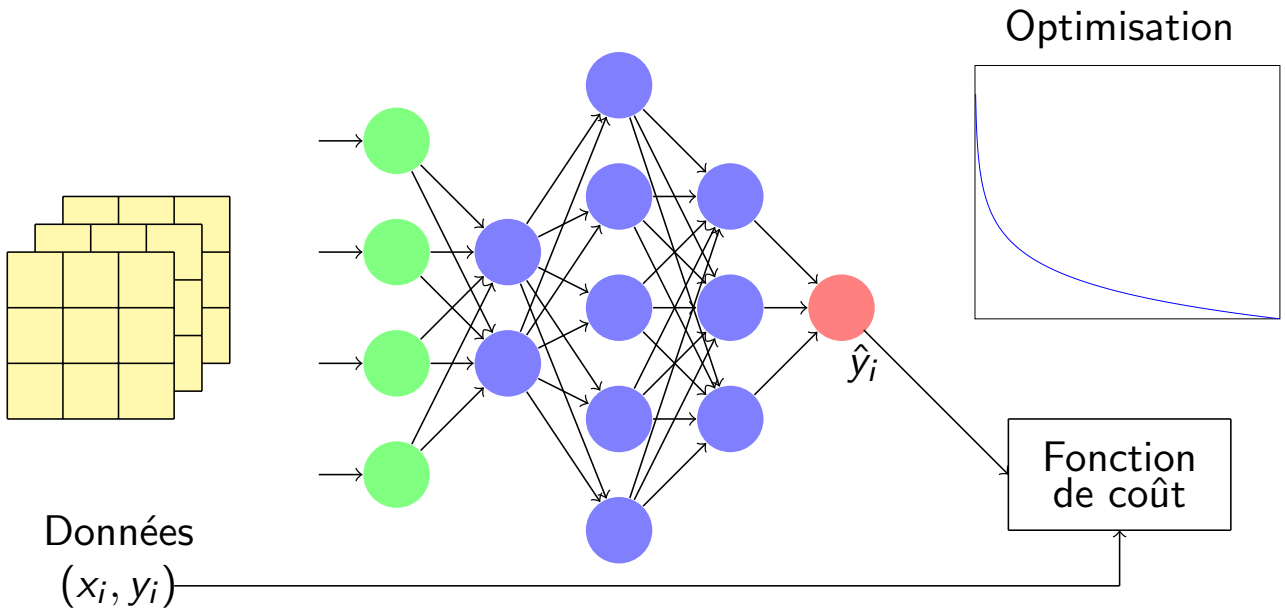
On peut :

- Diminuer la capacité du réseau : en pratique cela est déconseillé.
- Stopper l'entraînement plus tôt (diminuer le nombre d'*epochs*)
- **Régulariser**

Plan du cours

Régularisation

La régularisation consiste à imposer des contraintes sur le processus d'apprentissage afin de limiter le sur-apprentissage.
Ces contraintes peuvent s'impliquer à tous les niveaux : sur les données, sur les paramètres du réseau, dans la fonction de coût, ou encore dans l'optimiseur.



Arrêt anticipé (*early stopping*)

L'arrêt anticipé est une stratégie de régularisation qui consiste à observer l'erreur commise sur l'ensemble de validation et mettre un terme à l'apprentissage quand cette erreur commence à remonter.
En pratique : l'erreur sur l'ensemble de validation est bruitée, il faut attendre un peu avant de s'arrêter pour de bon.

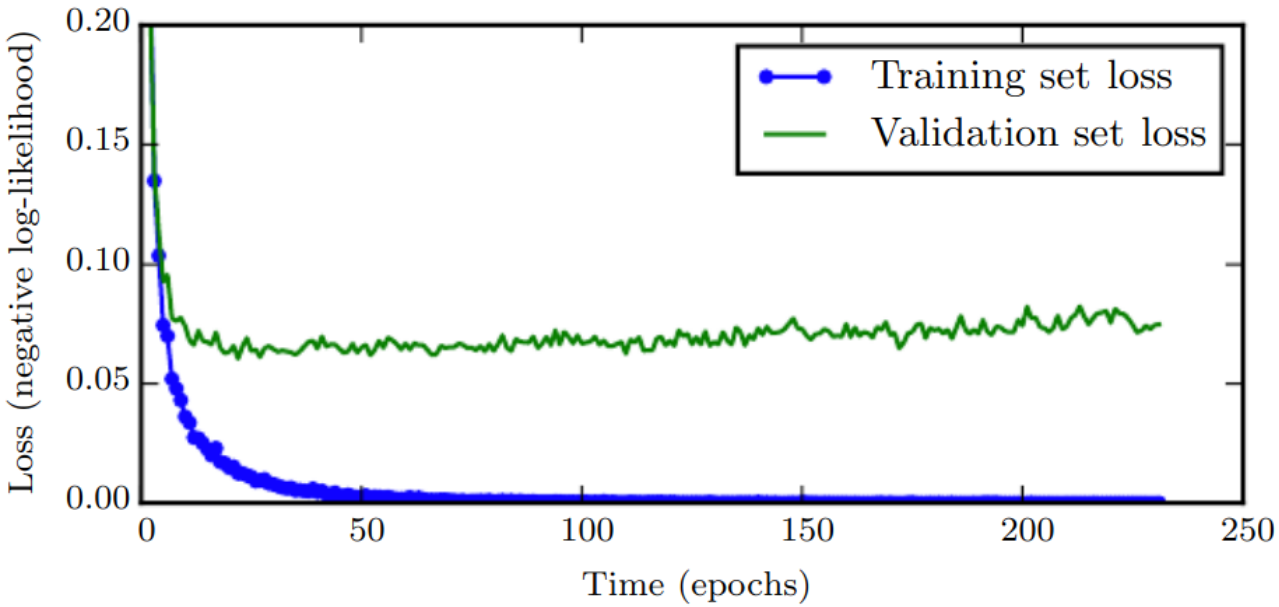


Image de [Goodfellow et al. 2015] Deep Learning

Weight decay

Ajout d'une contrainte sur les paramètres du réseau :

- Régularisation $\mathcal{L}^2$ ou **Ridge** maintient les coefficients du modèle aussi petits que possible :

$$J(\theta) = \text{RisqueEmpirique}(\theta) + \lambda \frac{1}{2} \sum_{i=1}^{m} \theta_i^2$$

où λ contrôle la qualité de régularisation souhaitée

- Régularisation $\mathcal{L}^1$ ou **Lasso** : tend à éliminer complètement les poids des variables les moins importantes ($\Rightarrow$ produit un modèle creux) :

$$J(\theta) = \text{RisqueEmpirique}(\theta) + \lambda \sum_{i=1}^{m} |\theta_i|$$

[Krogh, Hertz 1992] A simple weight decay can improve generalization

Régularisation

- Régularisation **Elastic net** (filet élastique) : compromis entre Ridge et Lasso exprimé par le paramètre $r \in [0, 1]$:

$$J(\theta) = \text{RisqueEmpirique}(\theta) + r\lambda\frac{1}{2}\sum_{i=1}^{m}|\theta_i| + \frac{1-r}{2}\lambda\sum_{i=1}^{m}\theta_i^2$$

`https://playground.tensorflow.org/`

Augmentation de base de données

Utiliser des ensembles de données de taille réduite peut induire un surapprentissage.

$\rightarrow$ augmentation artificielle de la taille de la base de données en les altérant avec des transformations contrôlées.

Cette pratique est particulièrement utile en *traitement d'images* : un réseau de neurones ne sait pas reconnaître des formes dans une image qui sont transformées par translation, rotation ou changement d'échelle.

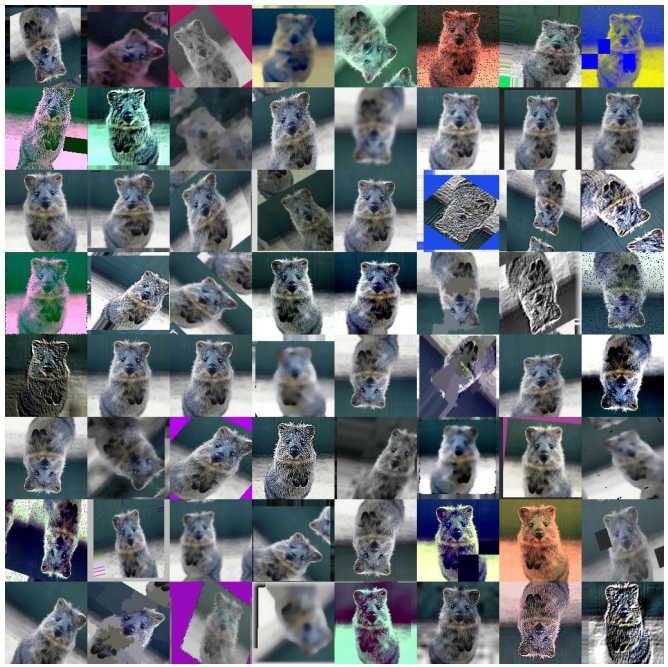


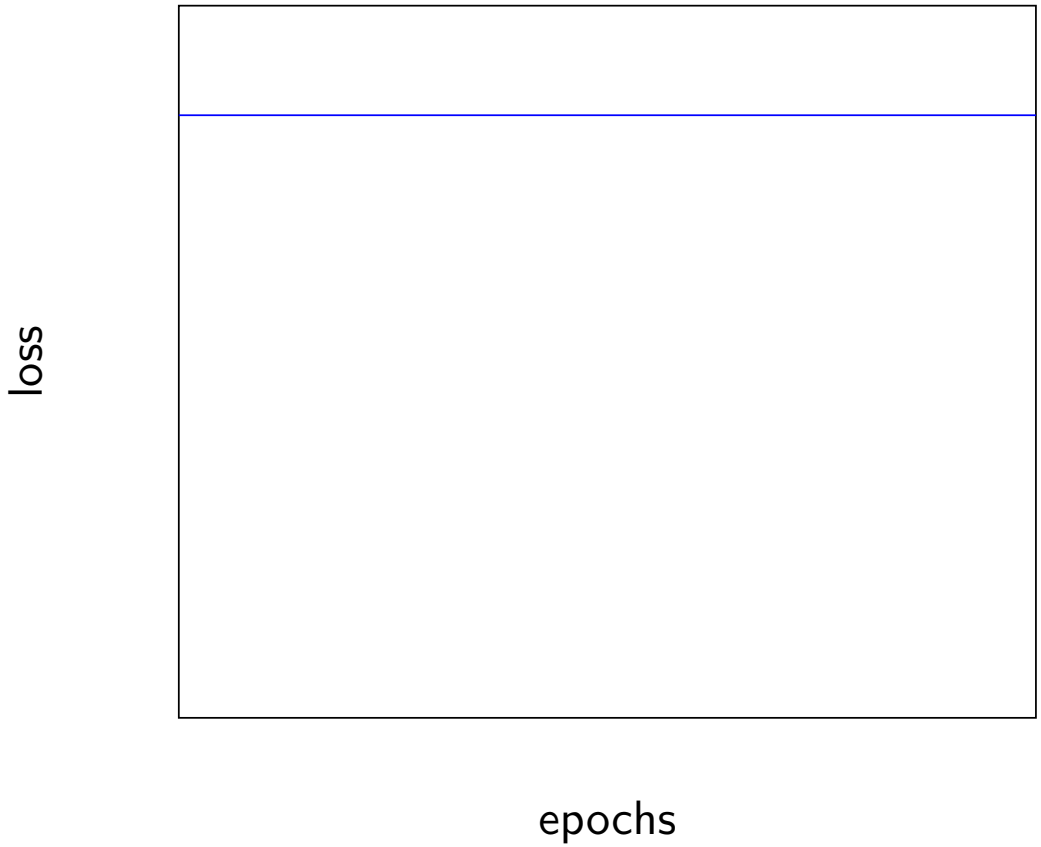
Image de `https://github.com/aleju/imgaug`

Bagging

Pour réduire le risque espéré, on peut entraîner plusieurs modèles (formes de réseau) différents, et les faire voter pour dégager la prédiction la plus populaire.
L'intuition derrière cette méthode est que les différents modèles se tromperont à différentes reprises...

Plan du cours

"Ça marche pas"



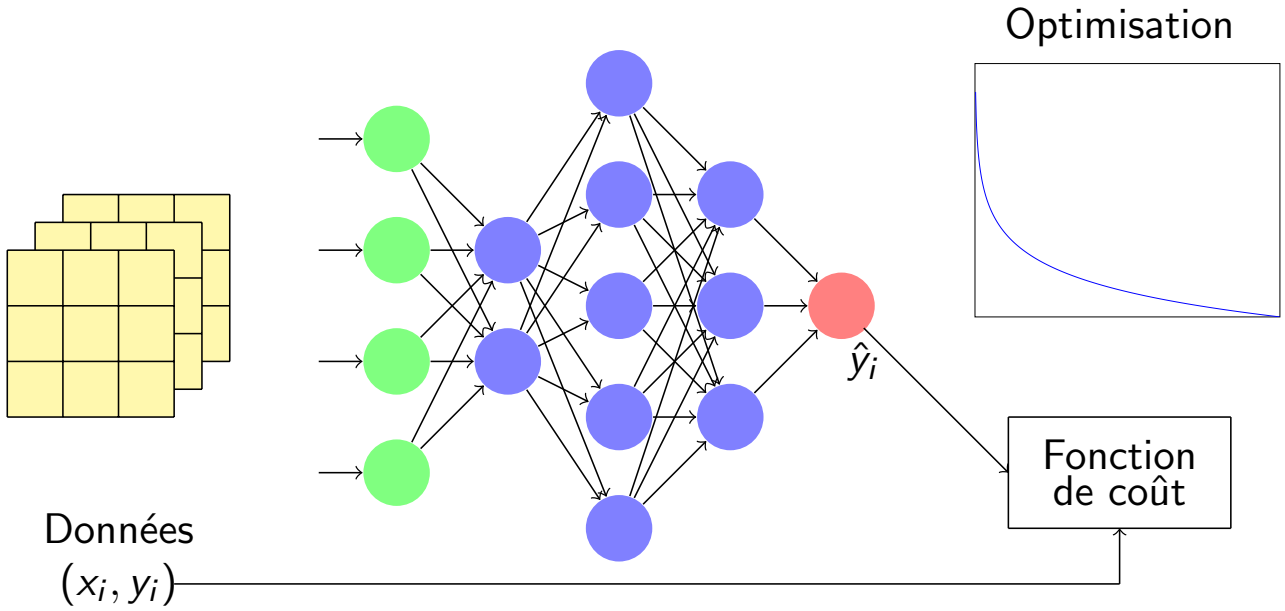
D'après vous, quel problème peut causer ce genre de comportement pour l'évolution de la fonction de perte ?

Plan du cours

Difficulté de l'apprentissage profond

Vous connaissez maintenant tous les éléments constitutifs d'un algorithme basé sur l'apprentissage profond.

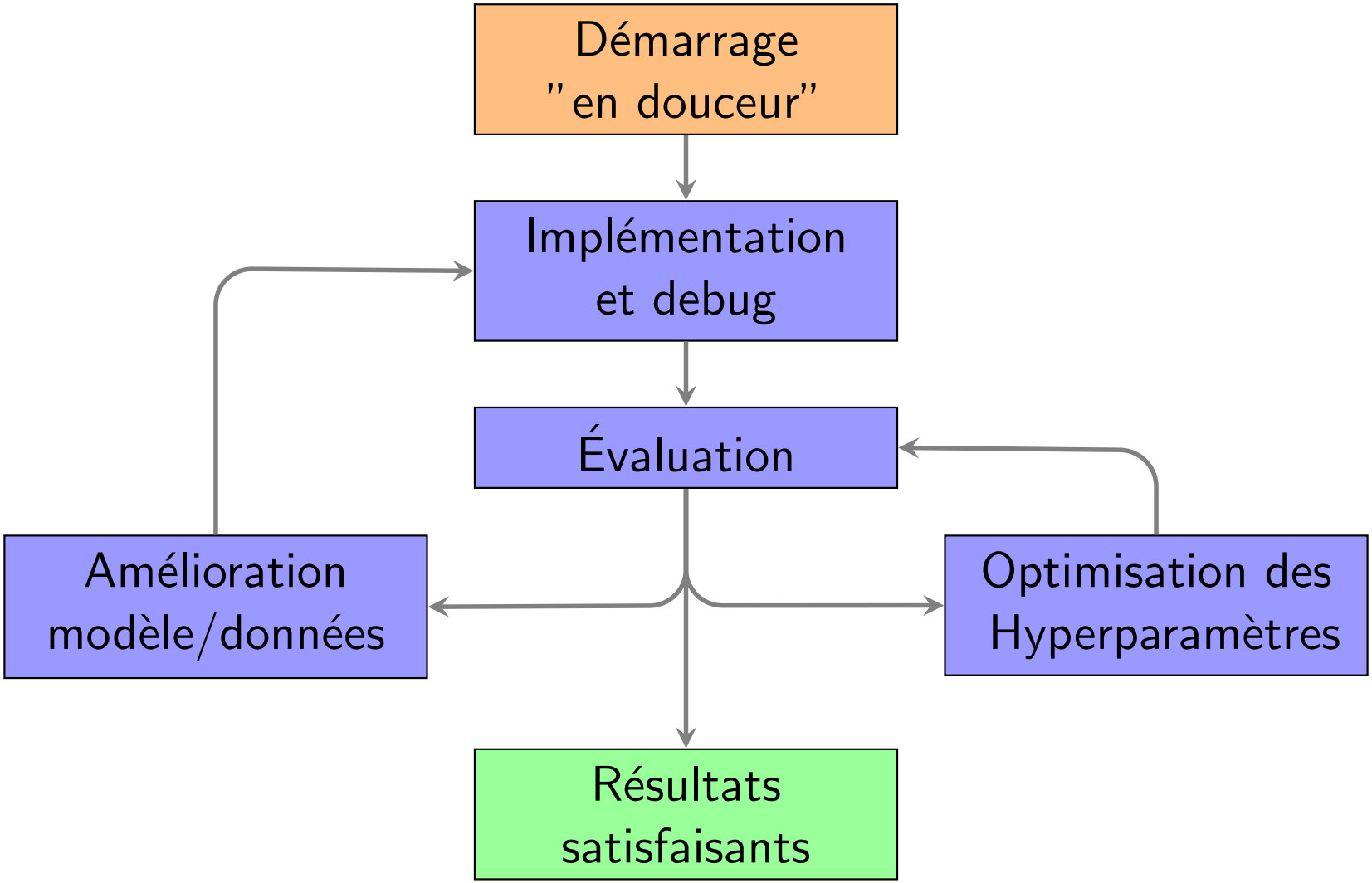
Comment s'y retrouver parmi toutes les combinaisons possibles d'architectures, d'hyperparamètres, de type de régularisation, etc. ? Tout n'est pas toujours nécessaire, ni désirable, ni aussi efficace en fonction des problèmes, des données, et du contexte.



profondeur, #neurones, activation α, momentum

$w_{i,j}, b_k$

Méthodologie générale



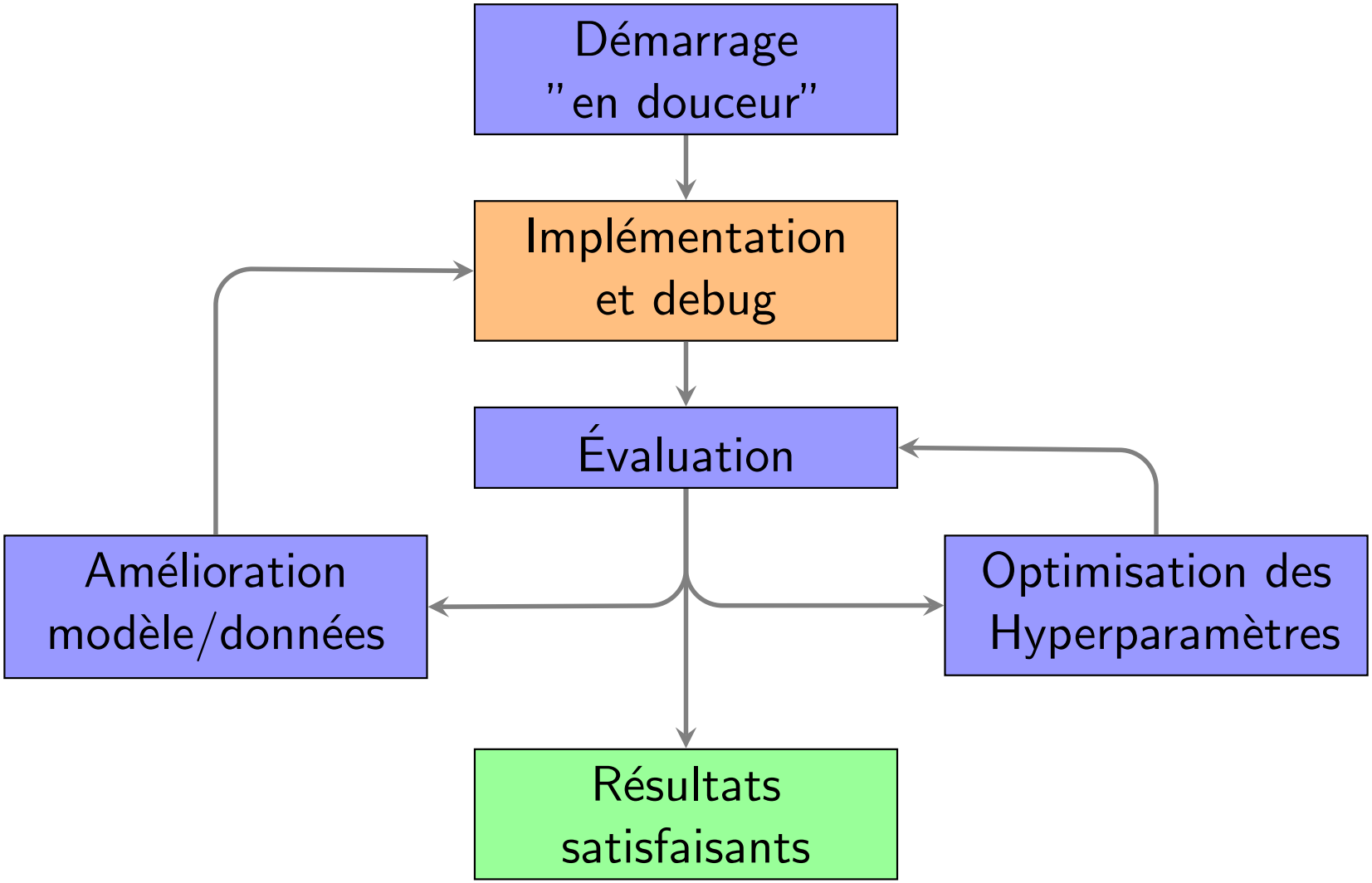
Adapté de *Troubleshooting Deep Neural Networks* de Josh Tobin

Démarrer simplement

Pour commencer sur un nouveau problème, l'objectif est de mettre au point une première version simpliste, mais qui fonctionne, à partir de laquelle on va itérer. Pour cela :

- Choisir une **architecture simple**
  Perceptron multi-couche, LeNet, LSTM
- Utiliser des **paramètres par défauts** efficaces
  Adam (avec taux d'apprentissage magique !), activation `reLU`, pas de régularisation
- **Normalisation** des données
  centrer-réduire, ramener entre 0 et 1
- Éventuellement **simplifier** le problème
  ensemble d'apprentissage réduit, diminuer le nombre de classes, la dimension des images

Méthodologie générale



Adapté de *Troubleshooting Deep Neural Networks* de Josh Tobin

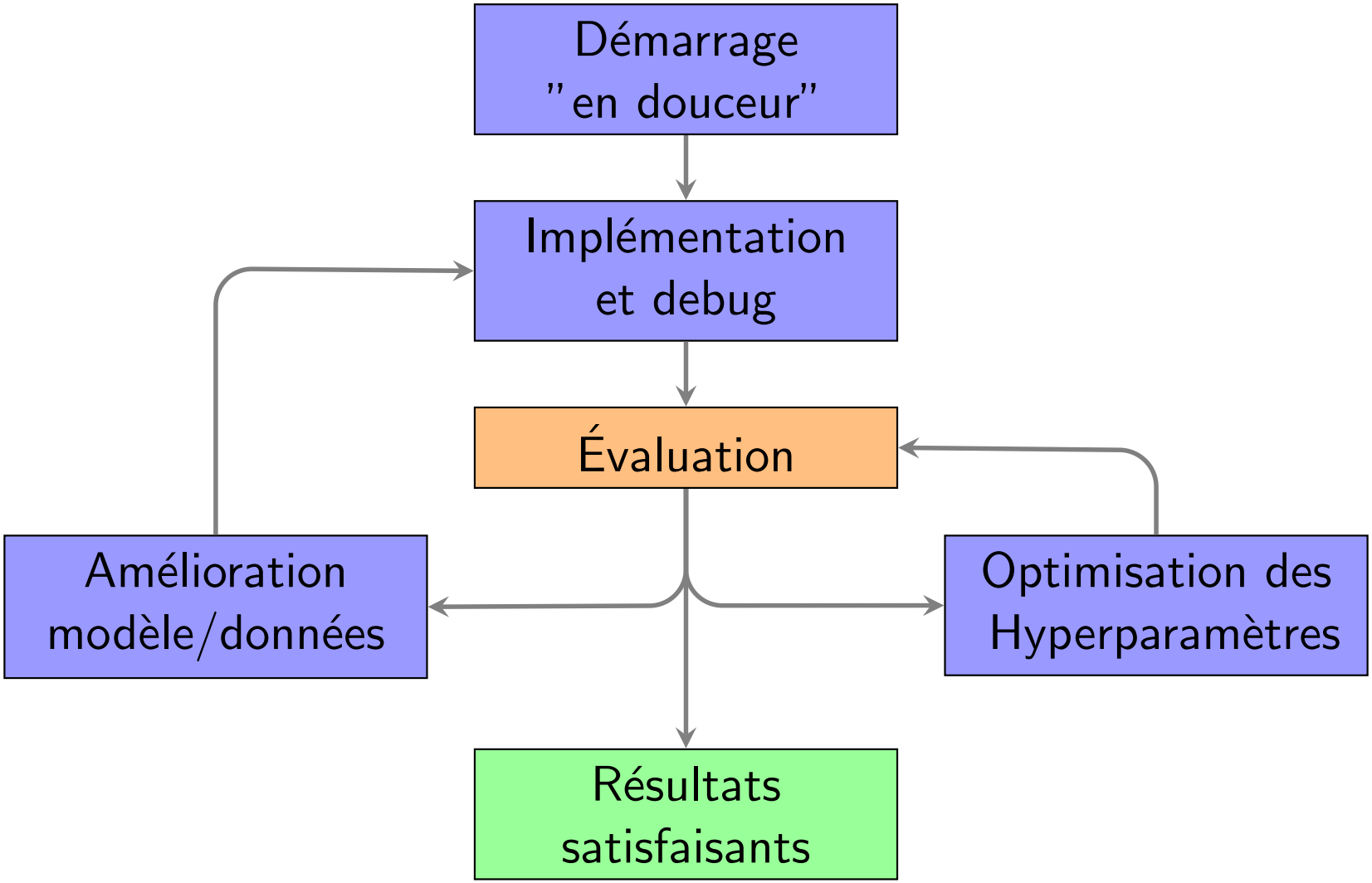
Débugger

Quelques erreurs et problèmes courants :

- Taille ou type des tenseurs incorrects
- Pré-traitement des données oublié ou fait plusieurs fois
- Sortie non adaptée à la fonction de perte
- Instabilité numérique (`NaN`, `Inf`) liée aux exponentielles, logarithmes, divisions, etc.
- Déficit de RAM GPU

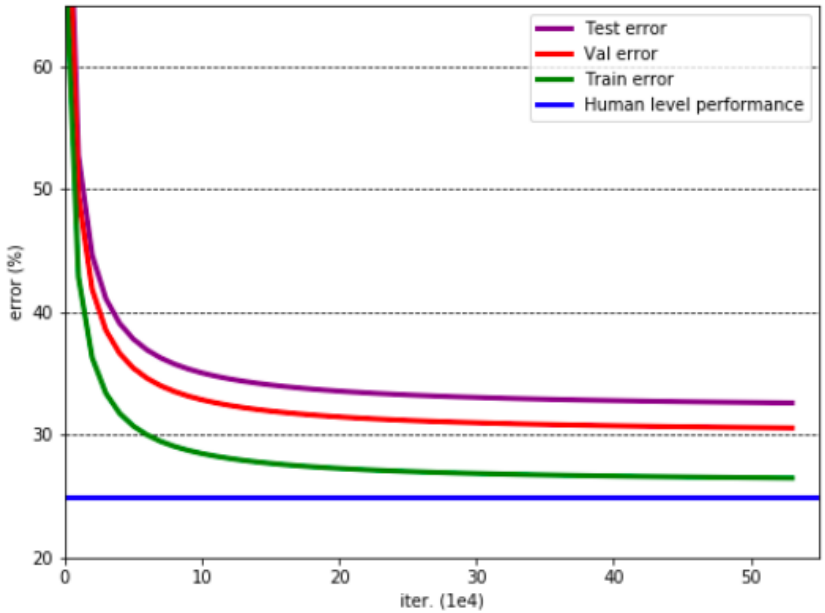
Conseil important : commencer par essayer de surapprendre un *batch* de données (voire une seule donnée)

Méthodologie générale



Adapté de *Troubleshooting Deep Neural Networks* de Josh Tobin

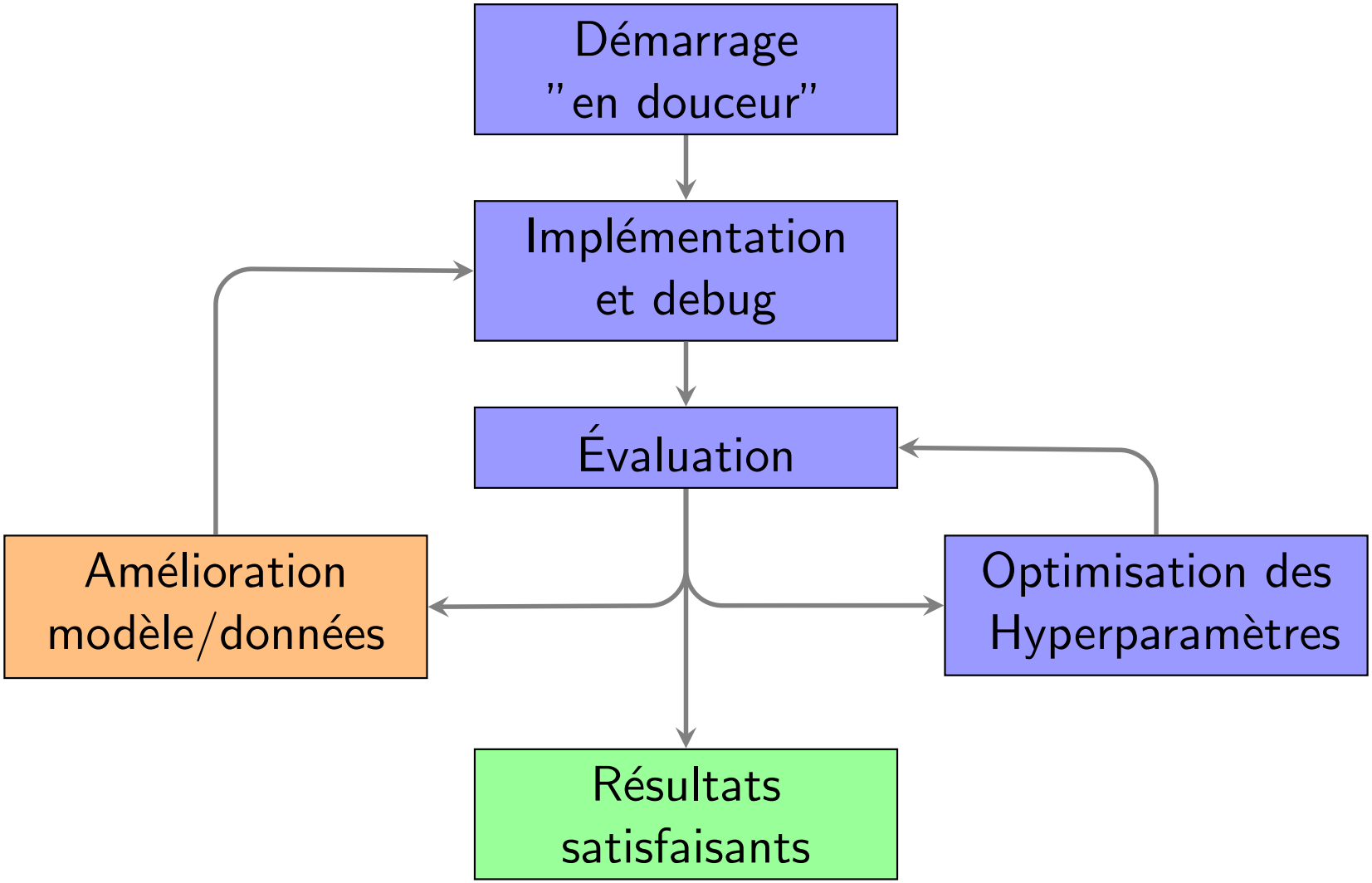
Évaluer la source des erreurs



L'erreur sur l'ensemble de test peut provenir de différentes sources :

- **Erreur irréductible** : meilleure performance objectivement atteignable
- **Sous-apprentissage**
- **Sur-apprentissage**
- **Sur-apprentissage de l'ensemble de validation**

Méthodologie générale



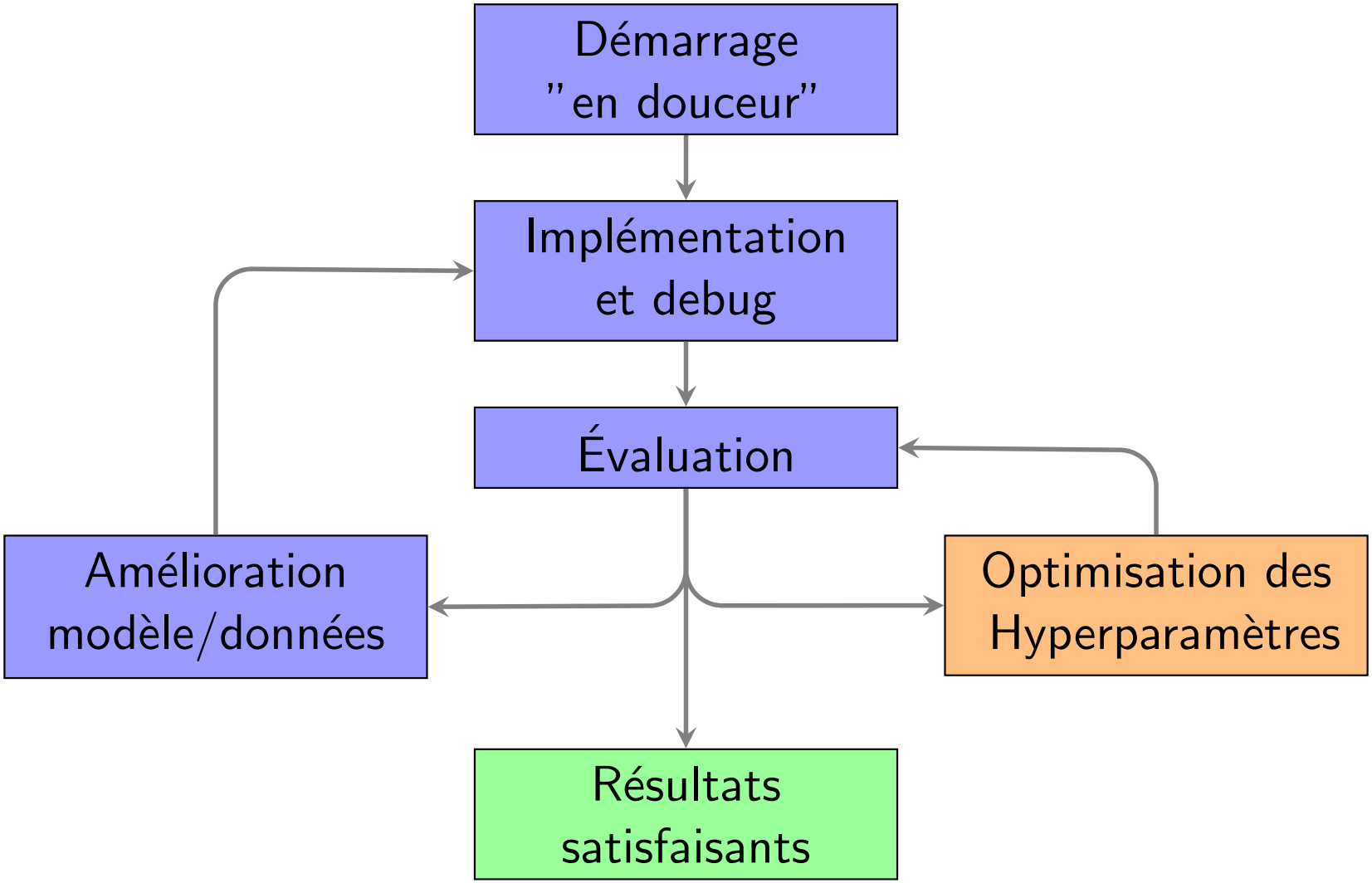
Adapté de *Troubleshooting Deep Neural Networks* de Josh Tobin

Améliorer le modèle et/ou les données

Pour corriger les problèmes mis à jour lors de l'évaluation on va privilégier l'ordre suivant :

1. Correction du sous-apprentissage
   Augmentation de la taille du modèle, changement d'architecture
   Réduction de la régularisation
2. Correction du sur-apprentissage
   Ajout de données (quand c'est possible !), ou augmentation des données
   Régularisation
3. Correction des problèmes de bases de donnée
   Déséquilibre de classes, distributions train/val/test différentes

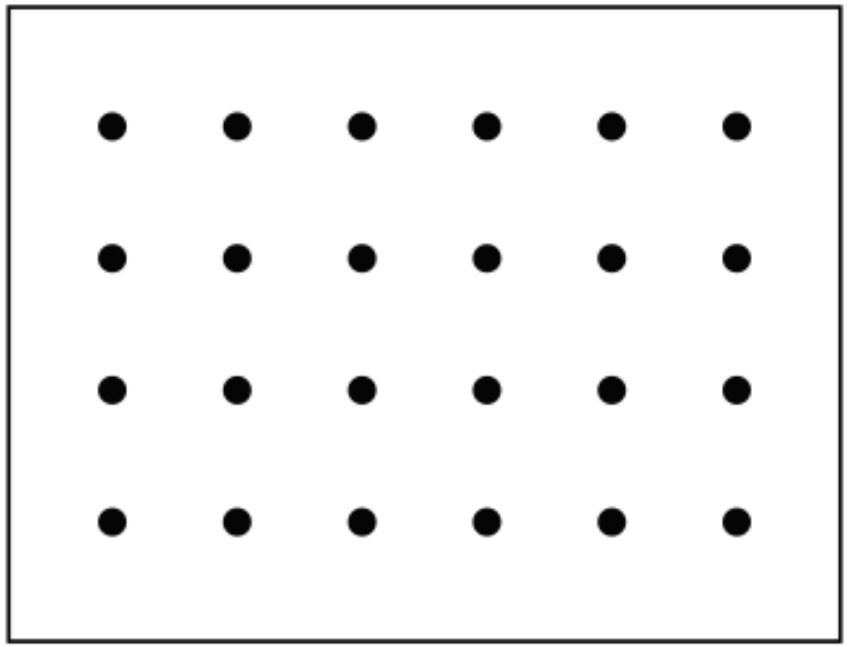
Méthodologie générale



Adapté de *Troubleshooting Deep Neural Networks* de Josh Tobin

Hyperparamètres

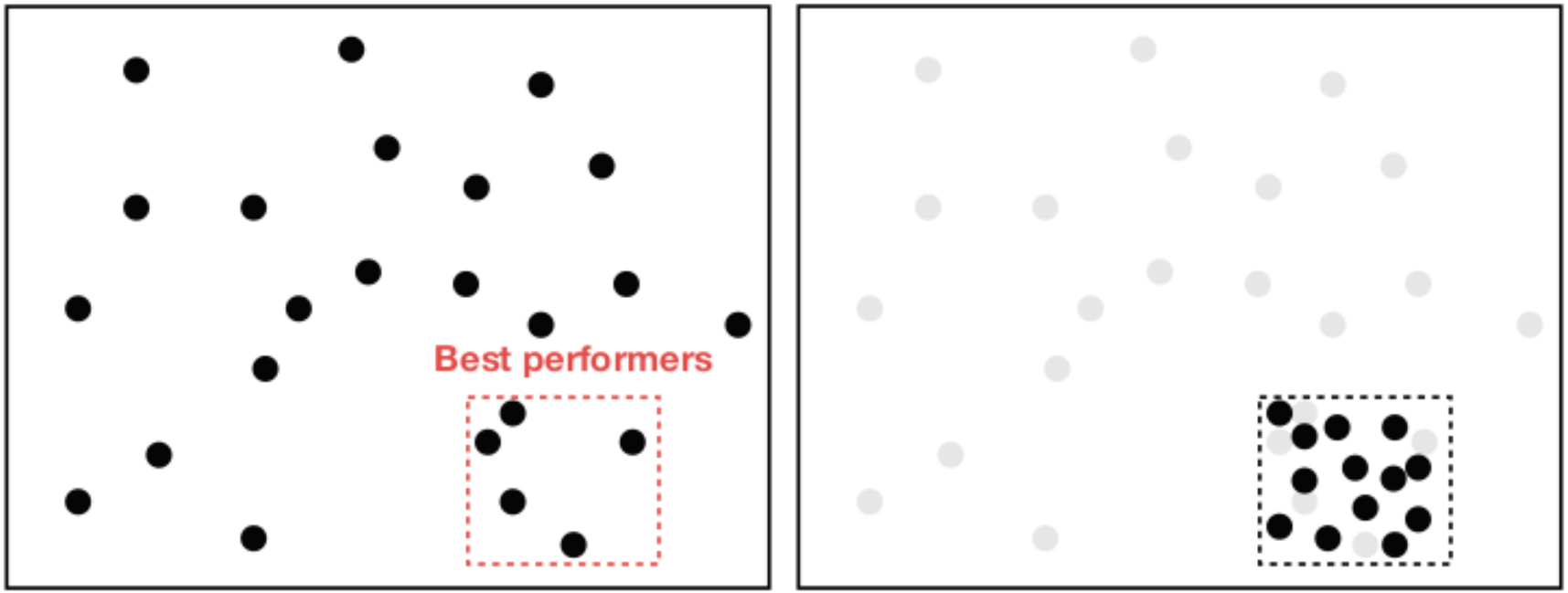
Pour mettre au point les valeurs optimales d'hyperparamètres (taux d'apprentissage, momentum, nombre de neurones par couche, etc.), une pratique commune est de déterminer un ensemble de valeurs possibles pour chaque hyperparamètre et de tester toutes les combinaisons. On conserve la combinaison qui minimise l'erreur sur l'ensemble de validation.



Chaque point correspond à l'entraînement complet d'un réseau de neurones (peut prendre plusieurs heures/jours/semaines...)

Hyperparamètres

Une variante plus couramment utilisée est d'implémenter une recherche aléatoire *coarse-to-fine*.



Chaque point correspond à l'entraînement complet d'un réseau de neurones (peut prendre plusieurs heures/jours/semaines...)

Crédits

Ce cours a été conçu en collaboration avec Sandrine Mouysset (IRIT - UPS) : `sandrine.mouysset@irit.fr`.

Ce cours s'appuie en outre en partie sur l'ouvrage *Deep Learning* de Ian Goodfellow, Joshua Bengio et Aaron Courville.

Un point sur le projet

Les groupes sont constitués !

Prochaine étape : **choix du sujet**. Pour cela, deux solutions :

- Vous venez me demander ce que je pense de votre idée de sujet, ou
- Vous m'envoyez un mail avec votre proposition de sujet

Avant le 2 février ! !