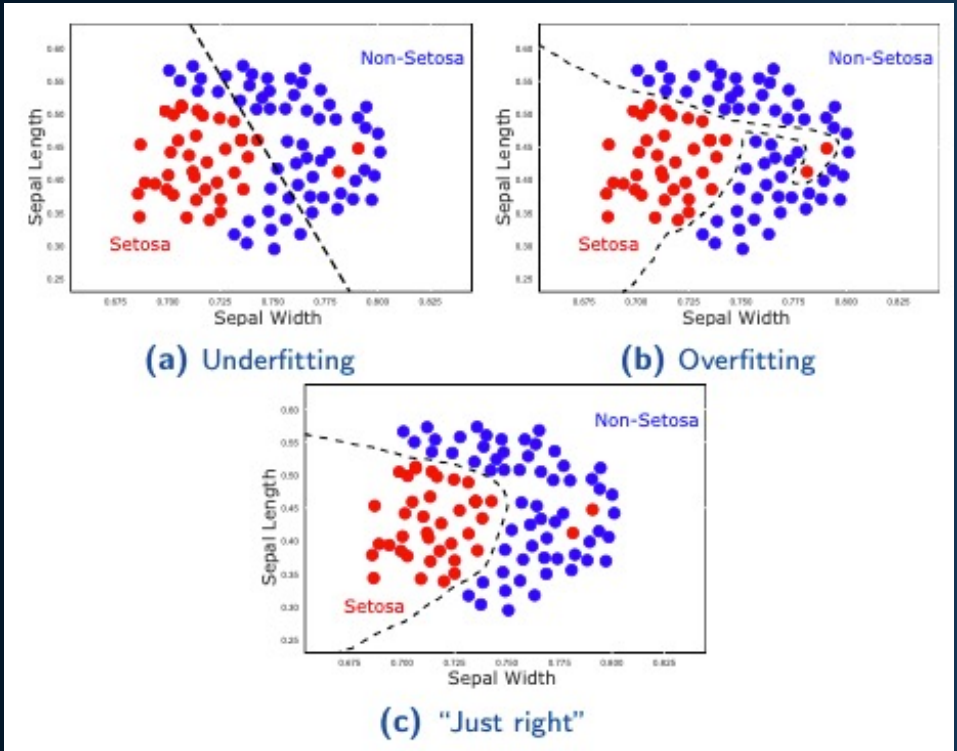
IA Géosciences - Deep learning: Régularisation des réseaux de neurones

Romain Wenger (Laboratoire Image Ville Environnement)

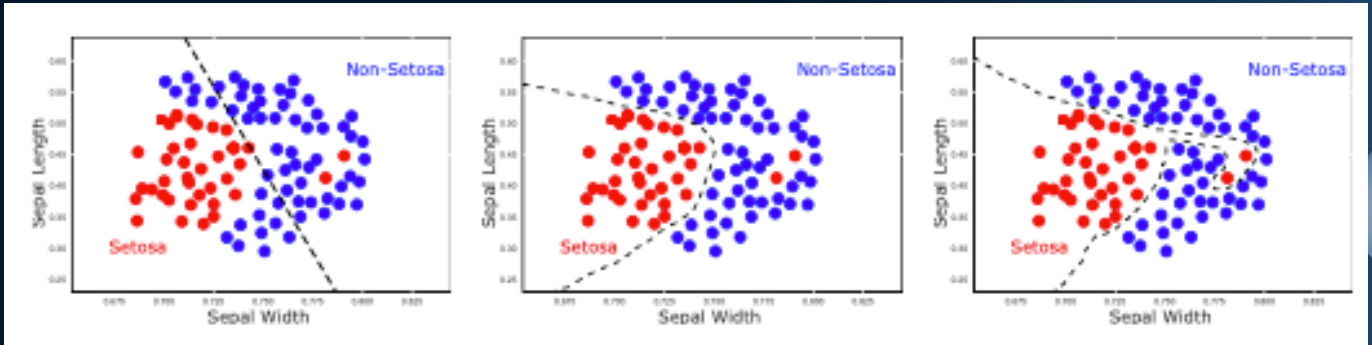
Table des matières

Table des matières

Sur/Sous-apprentissage

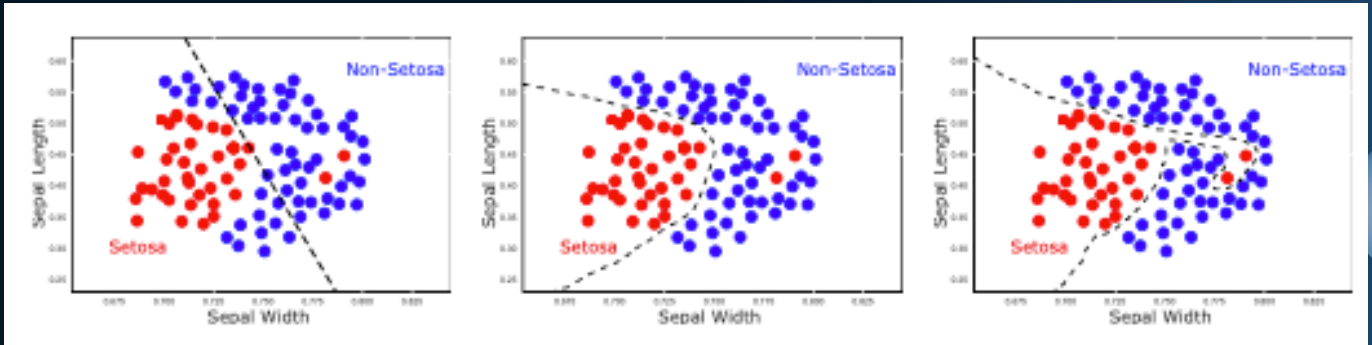


Sur-apprentissage < Notre but < Sous-apprentissage



Notre but est de trouver un modèle qui est assez puissant pour mitiger le sous-apprentissage mais pas trop afin d'éviter le sur-apprentissage

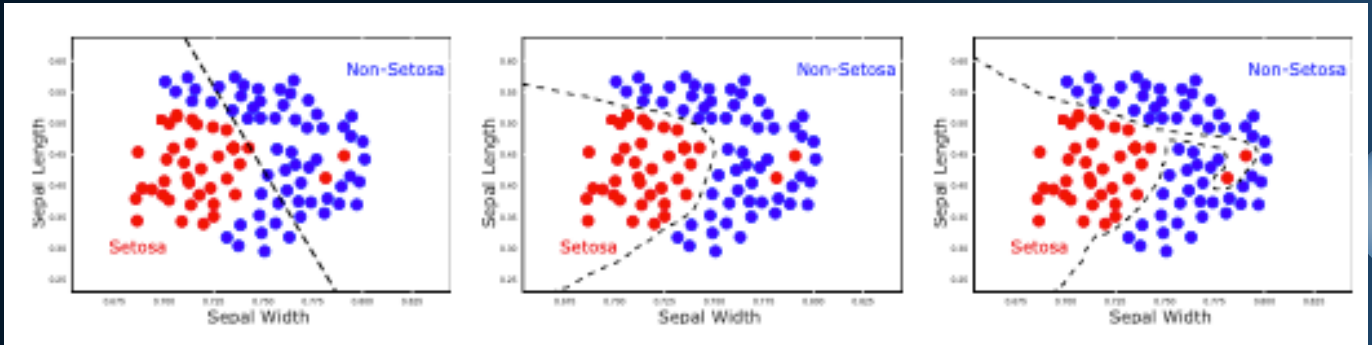
Sous-apprentissage



En général il est facile d'éviter le sous-apprentissage.

- En augmentant la complexité du modèle (e.g. nombre de neurones)
- Avec les GPUs il est facile d'entraîner un modèle complexe
- Les réseaux de neurones sont capables de mémoriser un dataset

Bonne généralisation



En général c'est difficile d'atteindre une bonne généralisation

- Un modèle trop complexe ⇒ sur-apprentissage
- Un modèle trop petit ⇒ sous-apprentissage

Stratégie

1. Faire du sur-apprentissage (on est sûr de la capacité du modèle)
2. Utiliser des techniques de régularisation pour empêcher l'overfitting

Table des matières

Régularisation explicite et implicite

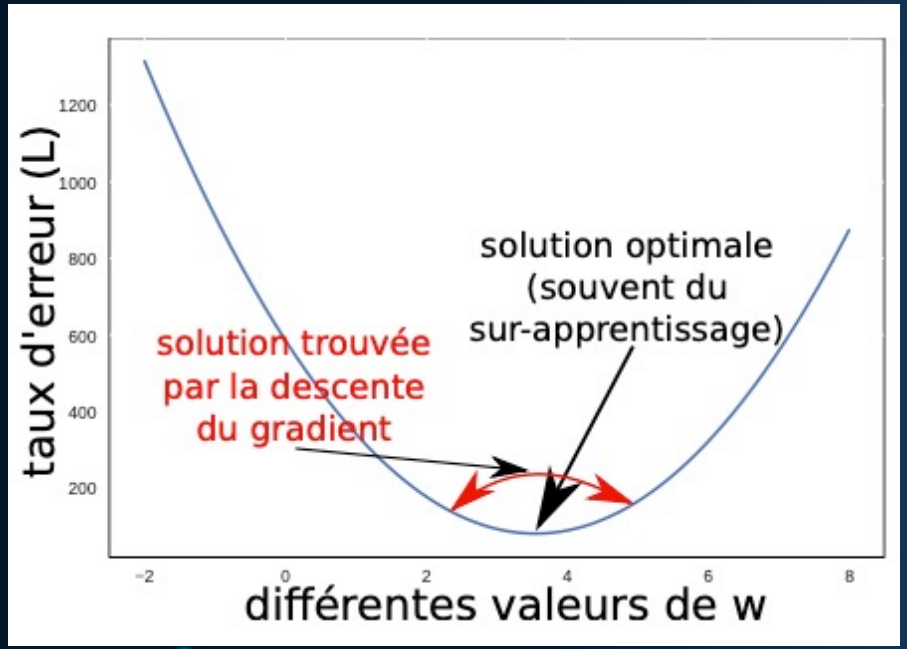
Explicite le but principal de la technique **est** la régularisation

- Weight decay (dégradation des poids)
- Dropout (désactivation aléatoire de neurones)
- Data augmentation (augmentation de données)

Implicite le but principal de la technique **n'est pas** la régularisation

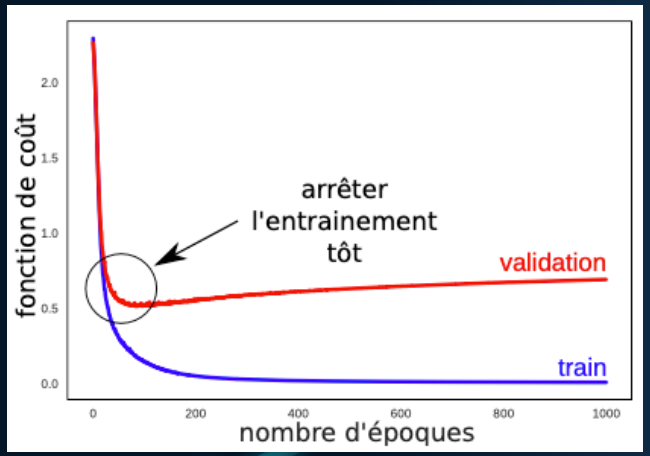
- Early stopping
- Batch normalization
- Descente de gradient
- Transfer learning

Descente du gradient : régularisation implicite



Early stopping : régularisation implicite

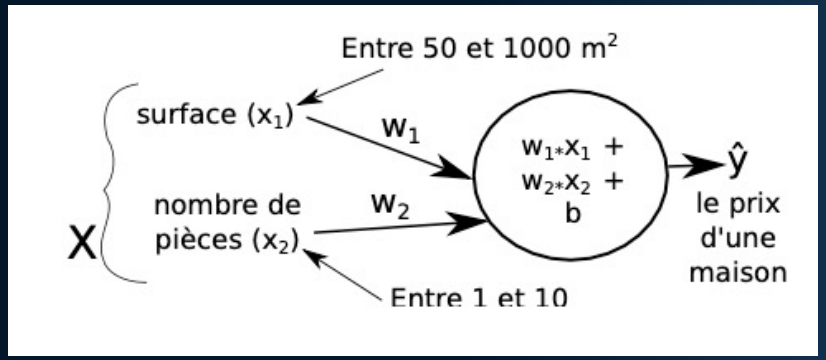
Cette technique consiste à arrêter l'entraînement plus tôt. ⇒ elle a un effet qui évite le sur-apprentissage



Une idée qui est proche de celle du modèle checkpoint.

Batch normalization : régularisation implicite

On reprend l'exemple de prédiction des prix des maisons



- Il faut donc normaliser l'entrée des neurones

⇒ les deux attributs auront la même contribution à la prédiction

Batch normalization : régularisation implicite

Une méthode pour normaliser :

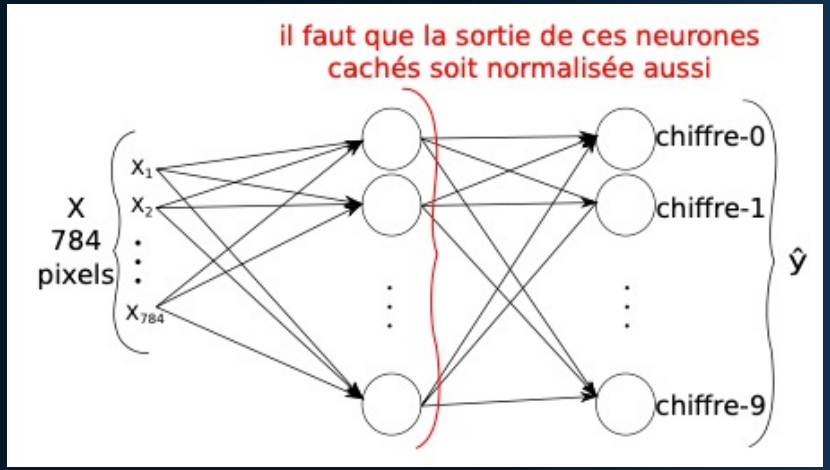
$$X_2 = \frac{X_1 - \mu}{\sigma}$$

- X_1 étant la valeur de l'attribut (avant et après normalisation)
- μ étant la moyenne de l'attribut sur tout le jeu de données
- σ étant l'écart type de l'attribut sur tout le jeu de données

Le résultat sera un attribut ayant une moyenne (μ) de 0 et un écart type (σ) de 1 après la normalisation

Batch normalization : régularisation implicite

Reprenons l'exemple d'un réseau avec une couche cachée



⇒ L'entraînement devient facile

⇒ Les neurones cachés auront la même contribution à la classification

- But initial améliorer l'entraînement ⇒ régularisation implicite
- On calcule μ et σ sur un batch au lieu de l'ensemble d'entraînement

Batch normalization : régularisation implicite

La normalisation statique traditionnelle appliquée aux attributs

$$z = \frac{z - \mu}{\sigma}$$

En développant cette fraction on aura

$$z = \frac{z}{\sigma} - \frac{\mu}{\sigma}$$

Soit $\gamma = \frac{1}{\sigma}$ et $\beta = \frac{-\mu}{\sigma}$ en remplaçant dans l'équation 3 on aura

$$z = \gamma * z + \beta$$

Idée importante : Au lieu de fixer γ et β calculer et en utilisant la formule de la moyenne et l'écart type, on pourra les apprendre avec la descente du gradient. Ainsi, « Batch Normalization » pourra s'adapter aux données et prendre la valeur de la vraie moyenne/écart type si c'est nécessaire

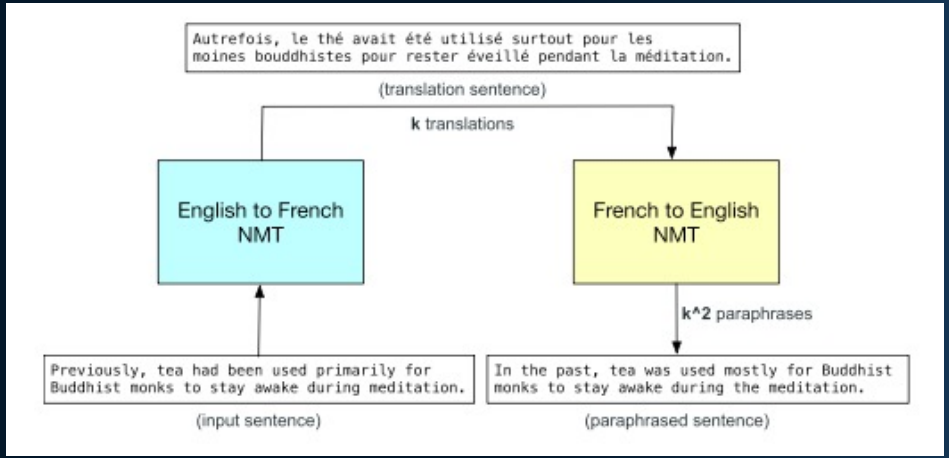
Augmentation des données : régularisation explicite

Pour les images on peut :

- Tourner (classifier un objet peu importe sa rotation)
- Bruiter (rendre le modèle robuste contre les images floues)
- Couper (classifier un objet à partir d'une partie)

Augmentation des données : régularisation explicite

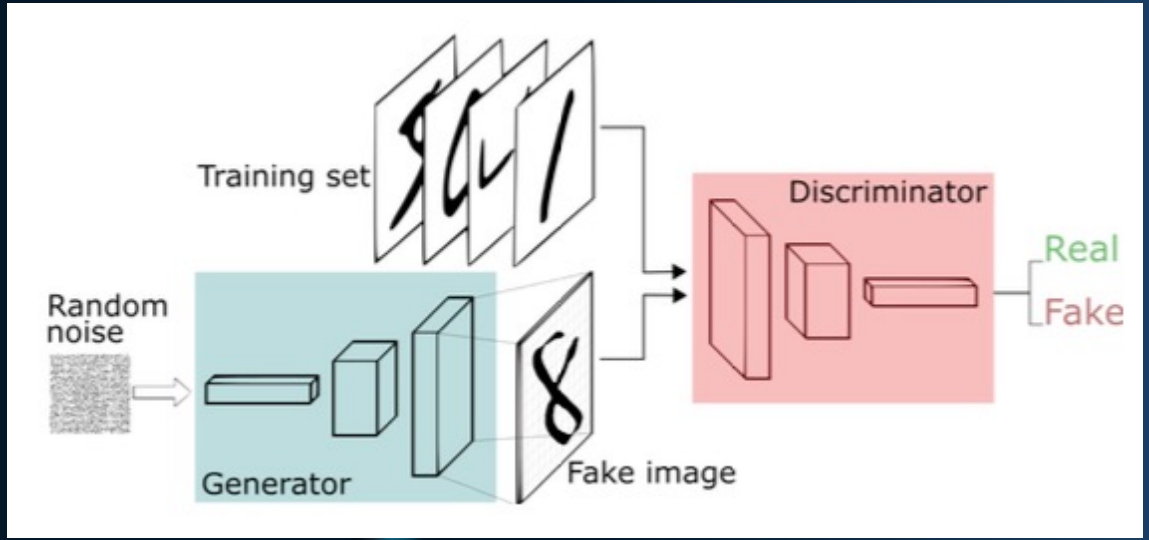
Pour les données textuelles



- Traduction EN -> FR ->EN
- Augmentation des synonymes

Augmentation des données : régularisation explicite

Méthode plus sophistiquée : entraîner un réseau qui génère des données : Generative Adversial Networks (GANs)



Augmentation des données : régularisation explicite

- Le but de l'augmentation des données est d'améliorer la généralisation

⇒ Régularisation explicite

- On aide le réseau à apprendre à classifier des images coupées, tournées, etc.

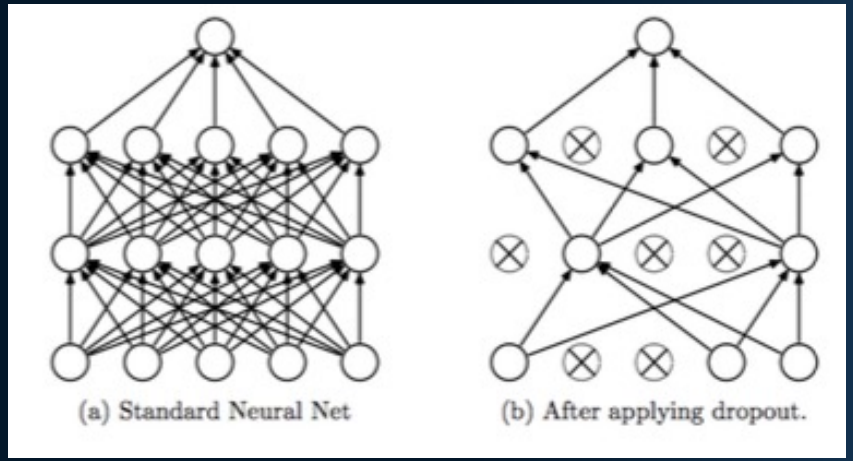
Par définition :

La régularisation (explicite) est toute sorte de modification dans le réseau qui a comme but d'améliorer la généralisation d'un modèle en diminuant le taux d'erreur sur la validation sans diminuer celui du train.

Solution idéale :

Avoir une très grande quantité de données réelles, mais il est souvent difficile de trouver des données annotées, notamment dans les géosciences.

Dropout : régularisation explicite

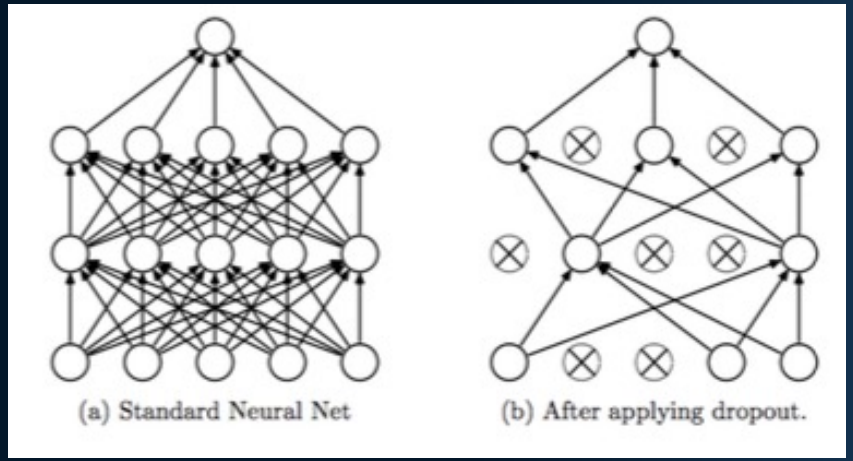


- Consiste à désactiver des neurones aléatoirement durant l'entrainement

⇒ Le réseau ne peut plus compter sur un seul neurone

⇒ Difficile de faire du sur-apprentissage

Dropout : régularisation explicite



- Consiste à désactiver des neurones aléatoirement durant l'entrainement

⇒ Le réseau ne peut plus compter sur un seul neurone

⇒ Difficile de faire du sur-apprentissage

Dégradation des poids (norme/2) : régularisation explicite

Fonction de coût générale

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i)$$

La dégradation des poids a comme but de pénaliser une grande valeur des poids $w_j \in W = (w_1, w_2, \dots, w_m)$ l'ensemble des paramètres à apprendre, en ajoutant à la fonction de coût :

$$\lambda \sum_{j=1}^{m} w_j^2$$

La fonction de coût devient

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i) + \lambda \sum_{j=1}^{m} w_j^2$$

Avec λ un hyper-paramètre qui détermine le degré de pénalisation des poids w_j

Dégradation des poids (norme/2) : régularisation explicite

La fonction de coût avec régularisation devient

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i) + \lambda \sum_{j=1}^{m} w_j^2$$

La fonction de coût change ⇒ la dérivée change

$$\frac{\boldsymbol{\theta L(w_j)}}{\boldsymbol{\theta w_j}} = \frac{\boldsymbol{\theta}(\frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i))}{\boldsymbol{\theta w_j}} + \frac{\boldsymbol{\theta}(\lambda \sum_{j=1}^{m} w_j^2)}{\boldsymbol{\theta w_j}}$$

Le premier terme ne change pas par rapport à ce qu'on a déjà vu

Pour le second terme de la dérivée, on a :

$$\frac{\boldsymbol{\theta w_k^2}}{\boldsymbol{\theta w_j}} = 0 \mid \forall k \neq j$$

Alors que pour w_j

$$\frac{\boldsymbol{\theta w_j^2}}{\boldsymbol{\theta w_j}} = 2 * \boldsymbol{w_j}$$

Dégradation des poids (norme/2) : régularisation explicite

La dérivée était

$$\frac{\theta L(w_j)}{\theta w_j} = \frac{\theta(\frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i))}{\theta w_j} + \frac{\theta(\lambda \sum_{j=1}^{m} w_j^2)}{\theta w_j}$$

La dérivée devient

$$\frac{\theta L(w_j)}{\theta w_j} = \frac{\theta(\frac{1}{n}\sum_{i=1}^{n} erreur(y_i, \hat{y}_i))}{\theta w_j} + \frac{2\lambda}{m} w_j$$

Dans le cas d'une régression logistique

$$\frac{\theta L(w_j)}{\theta w_j} = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i) * x_i + \frac{2\lambda}{m} w_j$$

A noter que pour les b on ne fait pas de régularisation

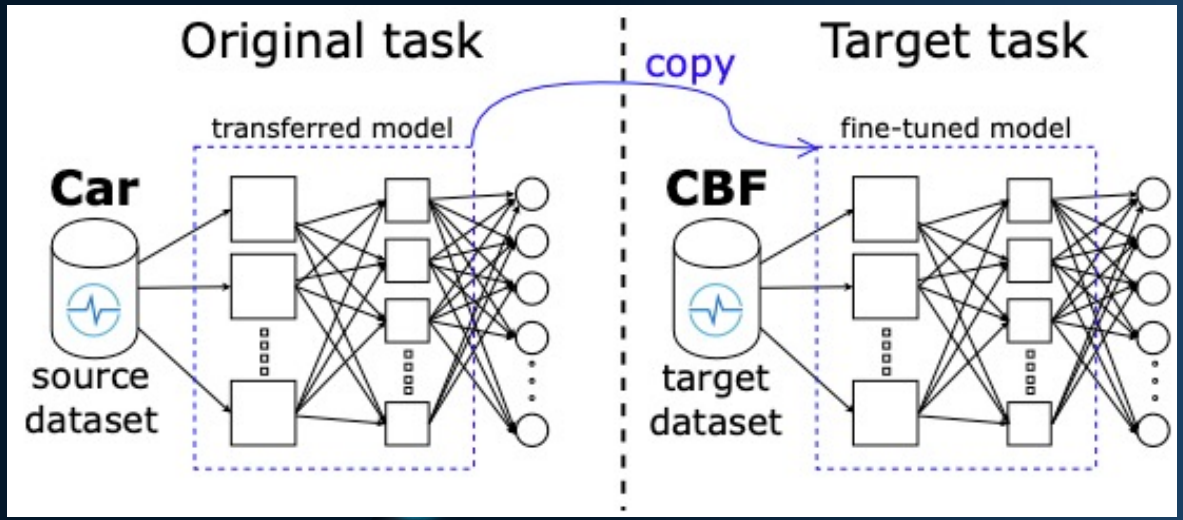
Apprentissage par transfert (Transfer Learning)

Il existe deux moyens pour initialiser les paramètres d'un réseau

1. Aléatoirement
    - Nécessite une énorme capacité de calcul pour converger
    - Introduit une instabilité dans les réseaux
    - Peut aboutir à une situation de sur-apprentissage
2. Par des poids d'un réseau déjà appris
    - Nécessite moins de calcul pour converger
    - Réduit l'instabilité due à l'initialisation aléatoire
    - Peut aboutir à une meilleure généralisation

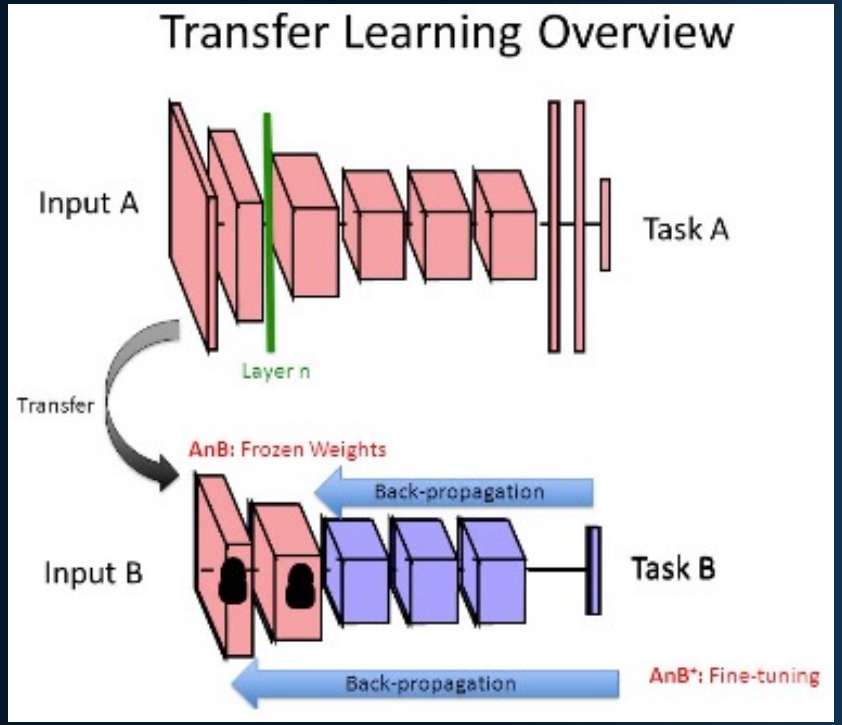
Apprentissage par transfert (Transfer Learning)

Le principe est de prendre un réseau qui a déjà été entraîné sur une tâche source et re-entraîner sur une tache cible



Apprentissage par transfert : blocage des couches

Une idée : ne pas re-entraîner toutes les couches



Apprentissage par transfert : blocage des couches

Le blocage des couches

- Permet de réduire encore le temps d'entraînement

Intuition

- Une couche profonde est plus proche des classes : spécifiques aux classes
- Une couche moins profonde extrait des caractéristiques plus génériques

Par exemple pour la classification des images chat.chien

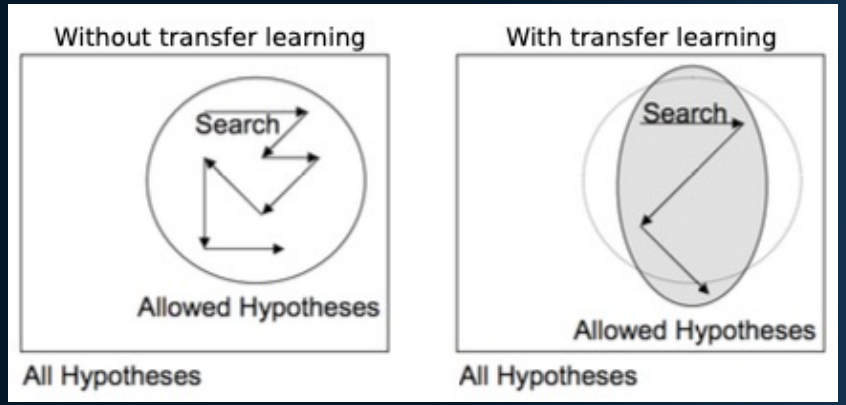
- Une couche profonde va extraire les yeux et oreilles des animaux
- Les deux premières couches vont extraire des contours verticaux et obliques qui seront utiles pour n'importe quelle tache de classification d'images ⇒ on peut ne pas les re-entraîner

Apprentissage par transfert : régularisation implicite

L’apprentissage par transfert : régularisation implicite

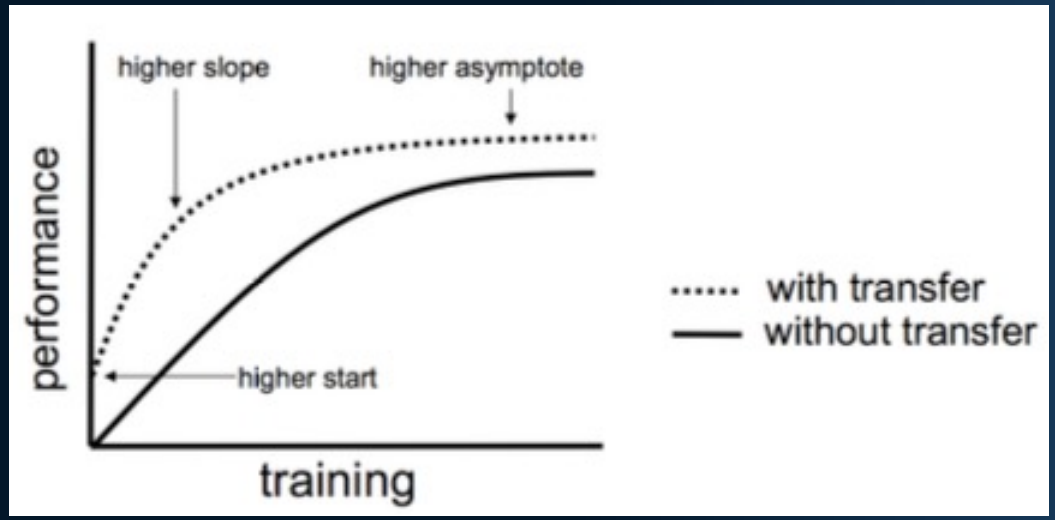
- But initial : réduire le temps d’entraînement
- Effet : réduire le sur-apprentissage

Apprentissage par transfert : réduit l'espace de recherche



En initialisant par des poids non-aléatoires, on limite où commence la descente du gradient ⇒ on limite l'espace des paramètres des poids w explorables par cette méthode

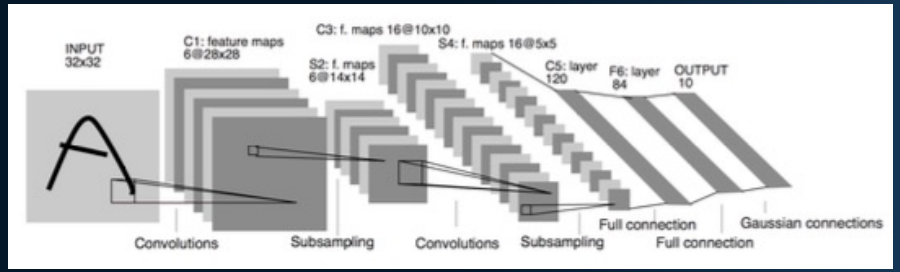
Apprentissage par transfert : meilleure généralisation



En initialisant par des poids qui ont déjà du sens sur une tâche source, on initialise le réseau avec une meilleure performance que l'initialisation aléatoire

Table des matières

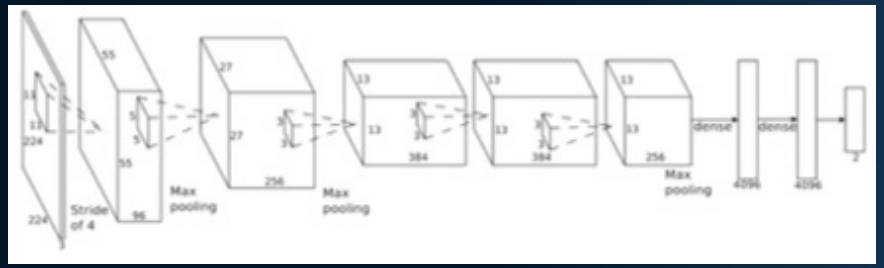
LeNet-5 (1998) pour MNIST-like



Caractéristiques

- Proposée pour des images de niveau de gris
- Des convolutions suivies par des poolings ensuite un MLP final
- Utilise les fonctions linéaires : sigmoid/tanh
- Des fonctions d’activation après le pooling
- Les mêmes filtres étaient appliqués pour chaque canal
- Contient 60k paramètres à apprendre

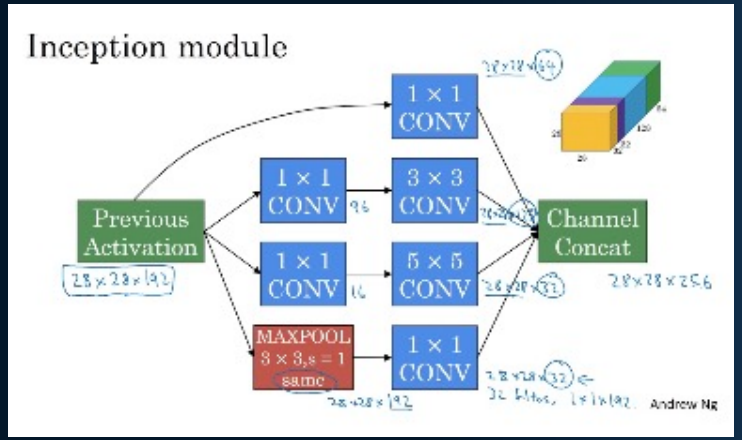
AlexNet (2012) pour ImageNet



Caractéristiques

- Proposée pour des images RGB
- Similaire à LeNet-5 mais beaucoup pour grand
- Contient 60M de paramètres comparés
- Utilise la fonction d’activation ReLU
- C’est le modèle qui a eu l’impact le plus important en deep learning

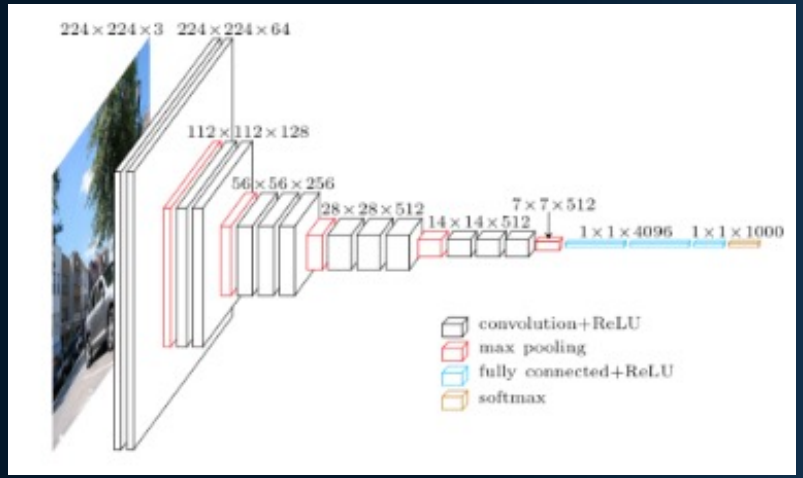
Inception (2014) pour ImageNet



Caractéristiques

- Implémente la convolution 1*1 pour réduire le temps de calcul
- Utilise plusieurs longueurs de filtre en parallèle
- On n’a pas besoin de faire le choix de ces hyper-paramètres

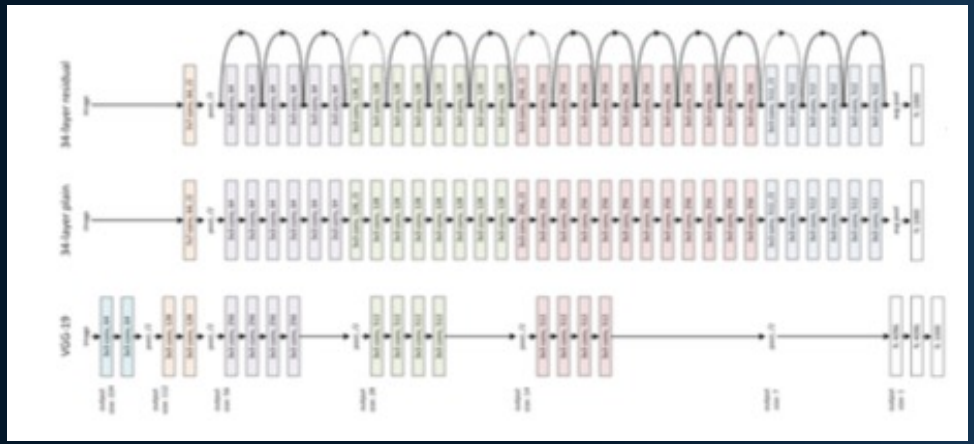
VGG-16 (2015) pour ImageNet



Caractéristiques

- Pour toutes les couches : la même taille des filtres (3*3)
- Pour toutes les couches : la même taille des max-pooling (2*2)
- Contient 138M de paramètres à apprendre
- On n’a pas besoin de faire le choix des filtres/pooling

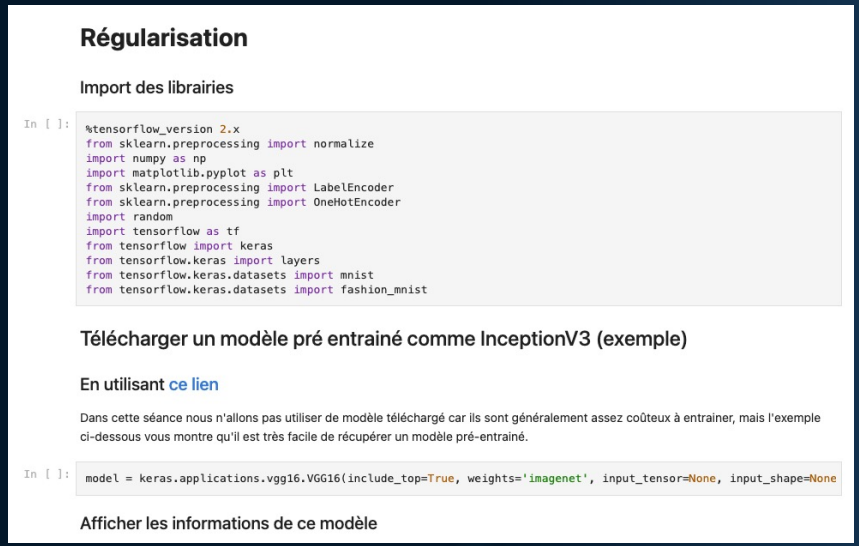
ResNet (2016) pour ImageNet



Caractéristiques

- Des connections résiduelles (les arcs en noir)
- Permet d’entraîner plus facilement des réseaux très profonds
- Ces connections permettent au réseau de sauter des convolutions inutiles en prenant ces « courts-circuits »

De la théorie vers la pratique



Régularisation

Import des librairies

In [ ]:

```
%tensorflow_version 2.x
from sklearn.preprocessing import normalize
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import random
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.datasets import fashion_mnist
```

Télécharger un modèle pré entrainé comme InceptionV3 (exemple)

En utilisant ce lien

Dans cette séance nous n'allons pas utiliser de modèle téléchargé car ils sont généralement assez coûteux à entrainer, mais l'exemple ci-dessous vous montre qu'il est très facile de récupérer un modèle pré-entrainé.

In [ ]:

```
model = keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', input_tensor=None, input_shape=None
```

Afficher les informations de ce modèle

Télécharger le fichier (https://github.com/r-wenger/cours_m1-m2-OTG/blob/main/IA_geosciences_M2/CM/6_DL_R%C3%A9gularisation.ipynb) et l'importer sur Google Colab