Stratégies d'entraînement
-
Workshop THINKII

Raphaël BERTRAND
Aix Marseille Univ, CNRS/IN2P3, CPPM, Marseille, France

Algorithmes classiques VS deep learning

Algorithme classique

Inputs + Fonction → Outputs

VS

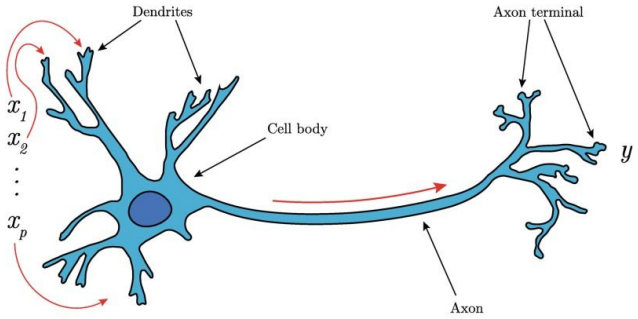
Machine learning

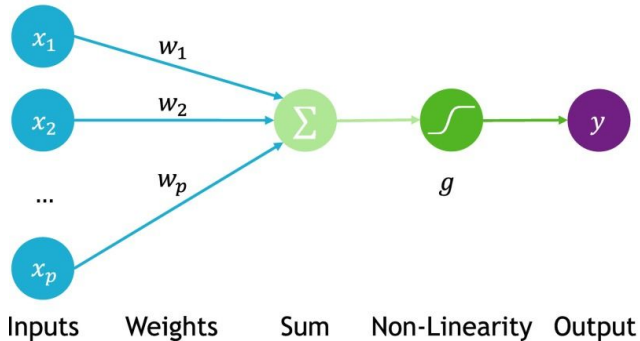
Inputs + Outputs → Fonction

- But : **calculer les données de sorties** via une fonction et des données d'entrée

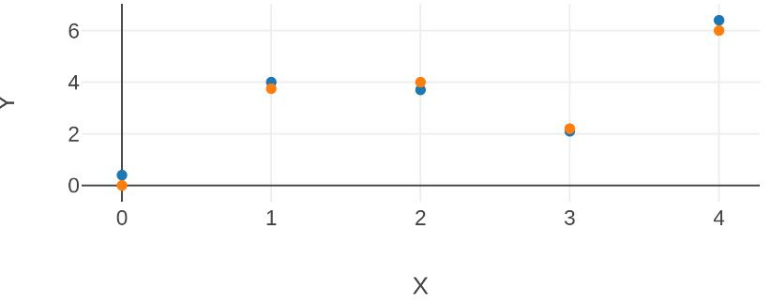
- But : **trouver le lien entre les données d'entrée et de sortie**
- **Assomption sur** la nature de **la fonction**
  - Poids et biais calculés

Deep learning :





Forward et backward propagation



Fonction de coût J(θ)

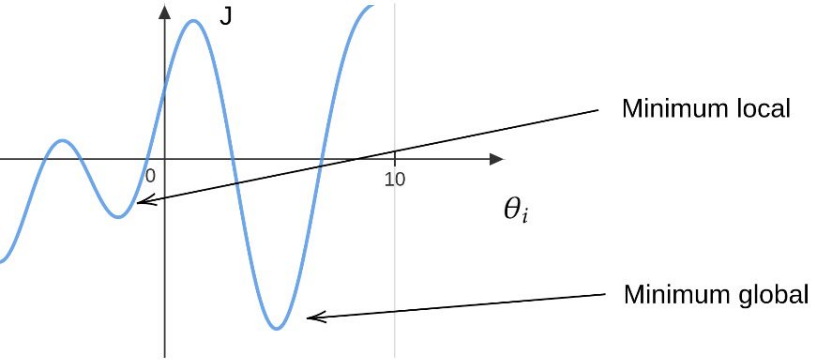
θ : paramètres du modèle

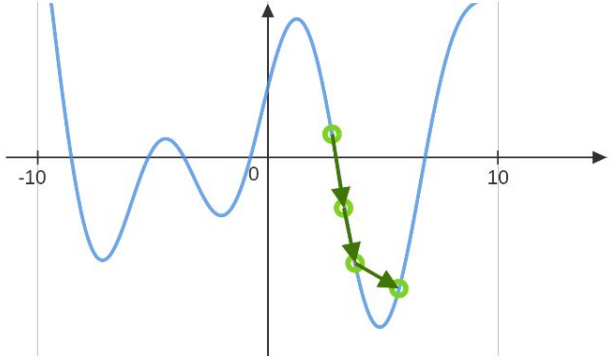
ex : MSE

$$J = \frac{1}{n}\sum_{k=0}^{n}(y_{pred} - y_{true})^2$$

Algorithme du gradient descendant

Minimiser la fonction de coût J





Mise à jour des poids :

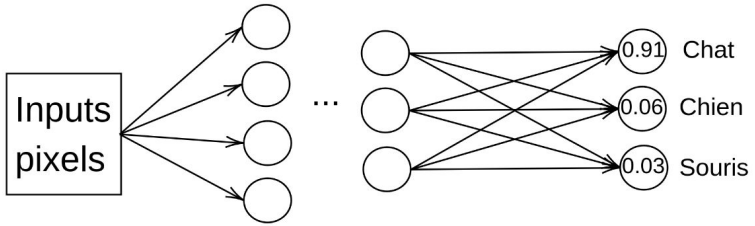
$$\theta_{i+1} = \theta_i - \alpha \frac{dJ(\theta)}{d\theta_i}$$

α : taux d'apprentissage

Principaux problèmes pour l'utilisation de NN (1/2)

Classification

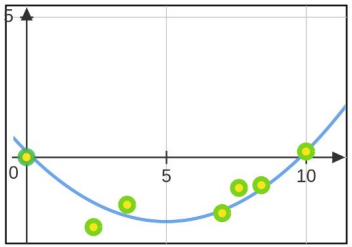
Image de chat



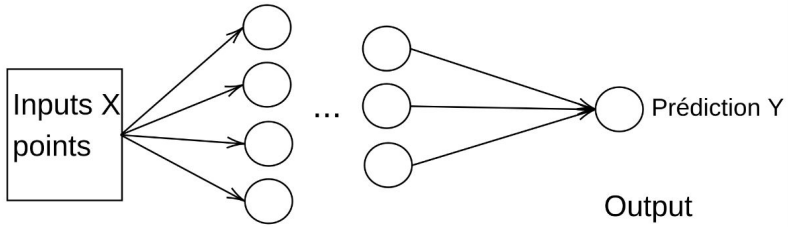
Couches intermédiaires
Repérer des features

Output

Régression



Fonction depuis un
espace de points

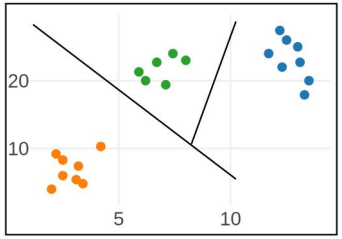


Output

Couches intermédiaires

Principaux problèmes pour l'utilisation de NN (2/2)

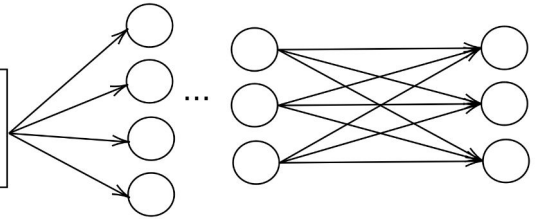
Clusterisation



Nuage de points

1
2
3

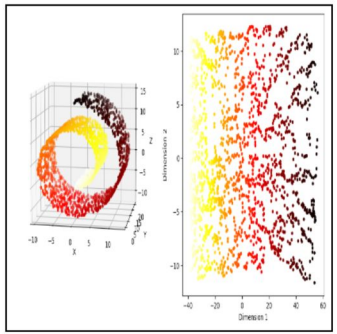
Inputs points



Couches intermédiaires
Fonction de séparation

Output

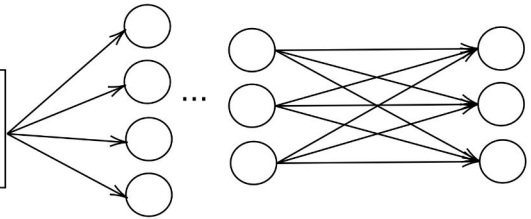
Réduction de dimensionnalité



N dimensions vers
N-p dimensions

Dimension outputs
< dimension inputs

Inputs points

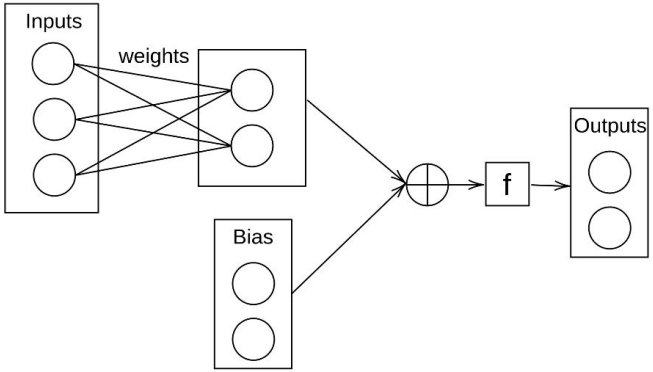


Couches intermédiaires

Output

Principales couches en deep learning : Dense et CNN

Dense → Brique élémentaire d'un réseau de neurones

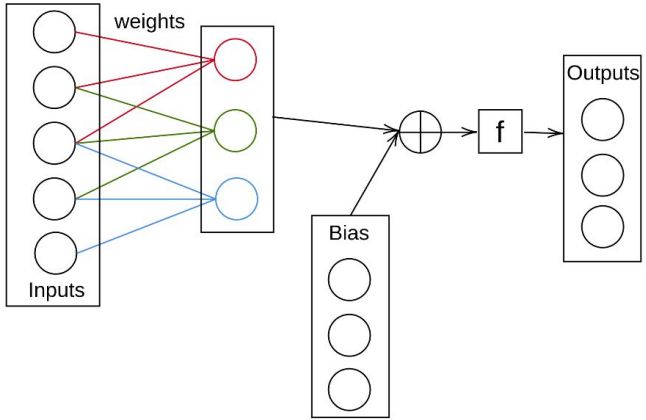


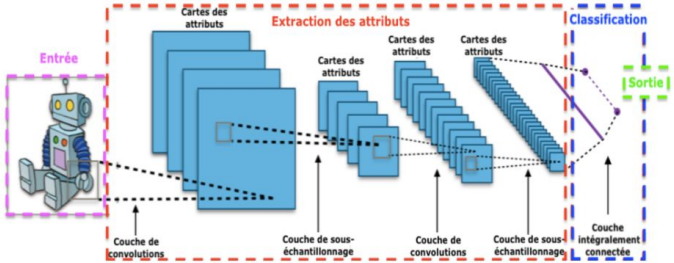
$$y_i = f(\sum_{k=0}^{n} w_{ik} x_k + \sum_{k=0}^{n} b_k)$$

inputs, weights, bias, fonction d'activation

n : dimension de l'output

CNN → Réseau de neurones convolutifs



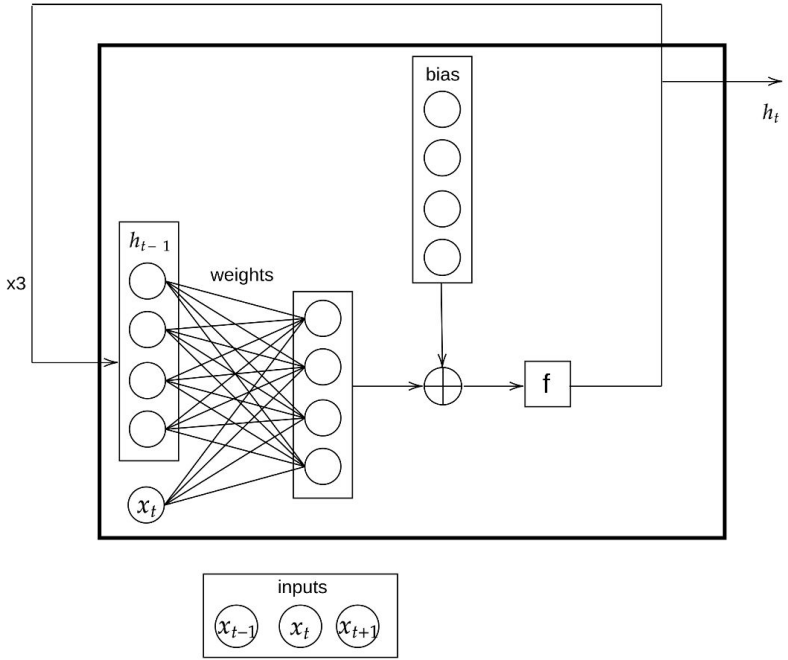


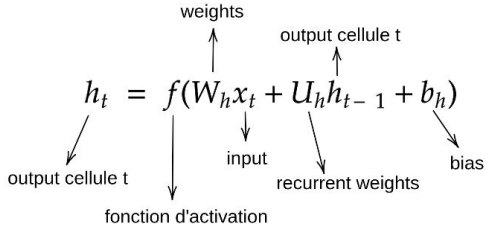
Paramètre d'un CNN :
- kernel/filtre : dimension de la fenètre glissante
- nombre de kernel/filtres
- stride : pas de déplacement du kernel

Les neurones d'entrée ne sont pas connectés à tous les neurones de sortie

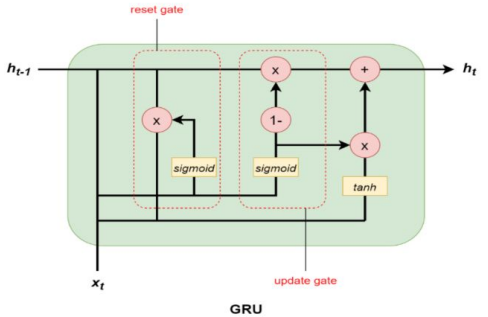
Principales couches en deep learning : Couches RNNs

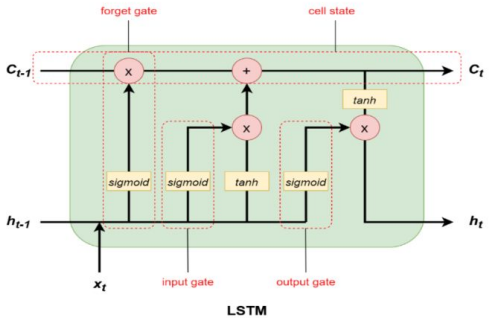
RNN ⟶ Prédiction de séries temporelles





← Vanilla RNN

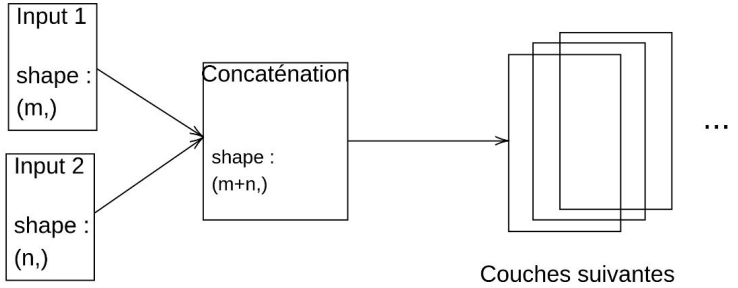




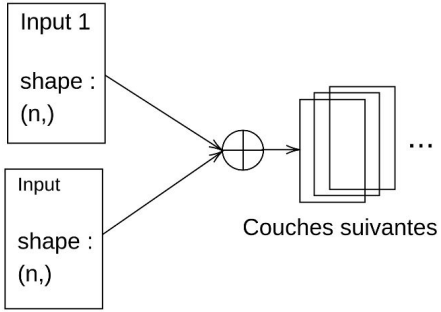
- Les RNNs utilisent une **mémoire court terme** au sein d'une séquence temporelle de données
- Les couches **GRU** et **LSTM** utilisent des portes supplémentaires :
  - Pour **oublier** des données
  - Pour **garder en mémoire long terme** des données

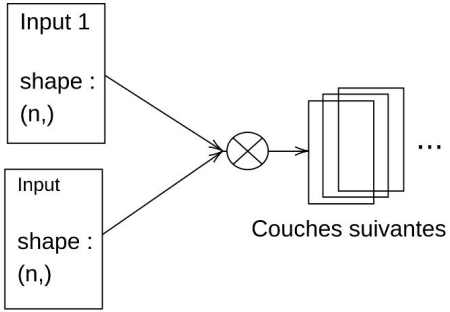
Quelques couches non configurables

Couche de concaténation : concaténer plusieurs couches en une seule



Couche d'addition/multiplication : additionner/multiplication les résultats de plusieurs couches





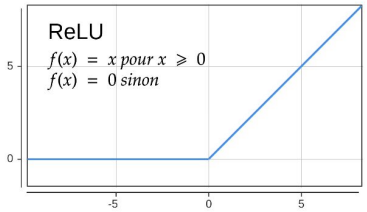
Max pooling

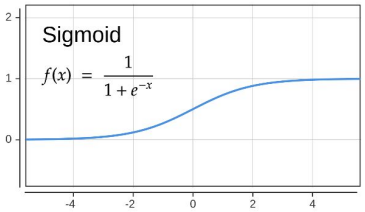
$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 2 & 1 \\ -1 & -2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 3 \\ 2 & 2 \end{bmatrix}$$

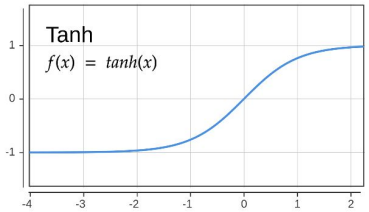
Average pooling

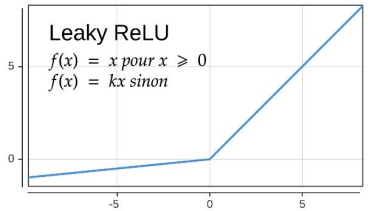
$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 2 & 1 \\ -1 & -2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1.75 & 2 \\ 0 & 0.25 \end{bmatrix}$$

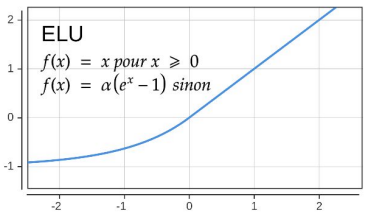
Fonctions d'activation

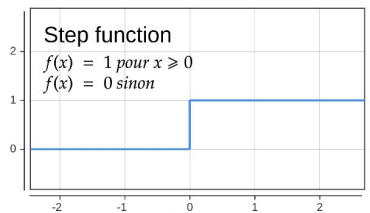


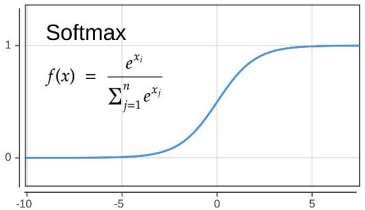


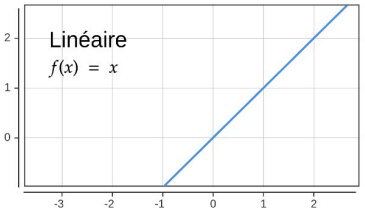












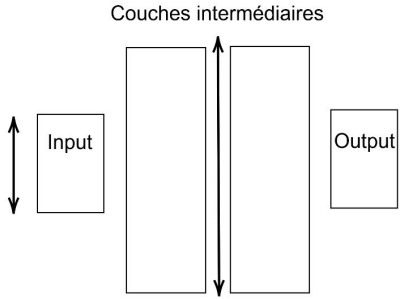
- Permet d'introduire la **non-linéarité** du modèle
  - Il n'est pas obligatoire d'en mettre une dans chaque couche
- **Restreint les données de sortie dans un intervalle**
  - ex : ReLU qui ne renvoie pas de valeurs < 0
- Certaines sont plus difficiles à faire converger le modèle : step function

	Intervalle d'arrivée	Dérivabilité	Utilité	Commentaire
ReLU	[0;+∞[	$\mathbb{R} \setminus \{0\}$	Seuil d'activation	Blocage des poids quand < 0
Sigmoide	]0;1[	$\mathbb{R} \setminus \{0\}$	Probabilités et classification binaire	Blocage des poids sur les extrémités
Tanh	]-1;1[	$\mathbb{R}$	Données équilibrées autour de 0	Blocage des poids sur les extrémités, Symétrique
Leaky ReLU	]−∞;+∞[	$\mathbb{R} \setminus \{0\}$	Meilleure robustesse que ReLU	
ELU	]-1;+∞[	$\mathbb{R} \setminus \{0\}$	Meilleure robustesse que ReLU	
Step function	{0} U {1}	$\mathbb{R} \setminus \{0\}$	Décision binaire	Difficile d'utiliser le gradien descendant
Softmax	[0;1]	$\mathbb{R}$	Probabilités	Blocage des poids sur les extrémités, Symétrique
Linéaire	$\mathbb{R}$	$\mathbb{R}$	Pas une fonction d'activation	

Complexité d'un problème → Largeur vs Profondeur

Largeur

+ d'unités de neurones
pour les couches intermédiaires

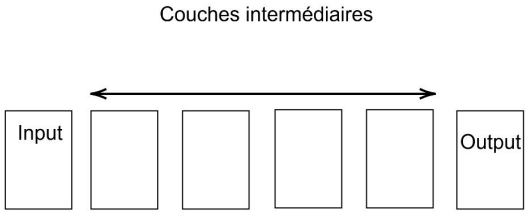


Dimension couches intermédiaires > Dimension input

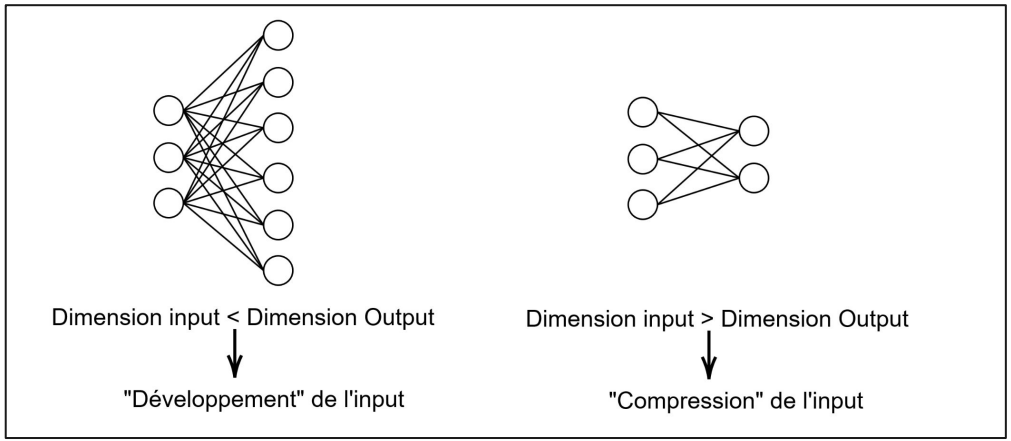
VS

Profondeur

+ de couches intermédiaires



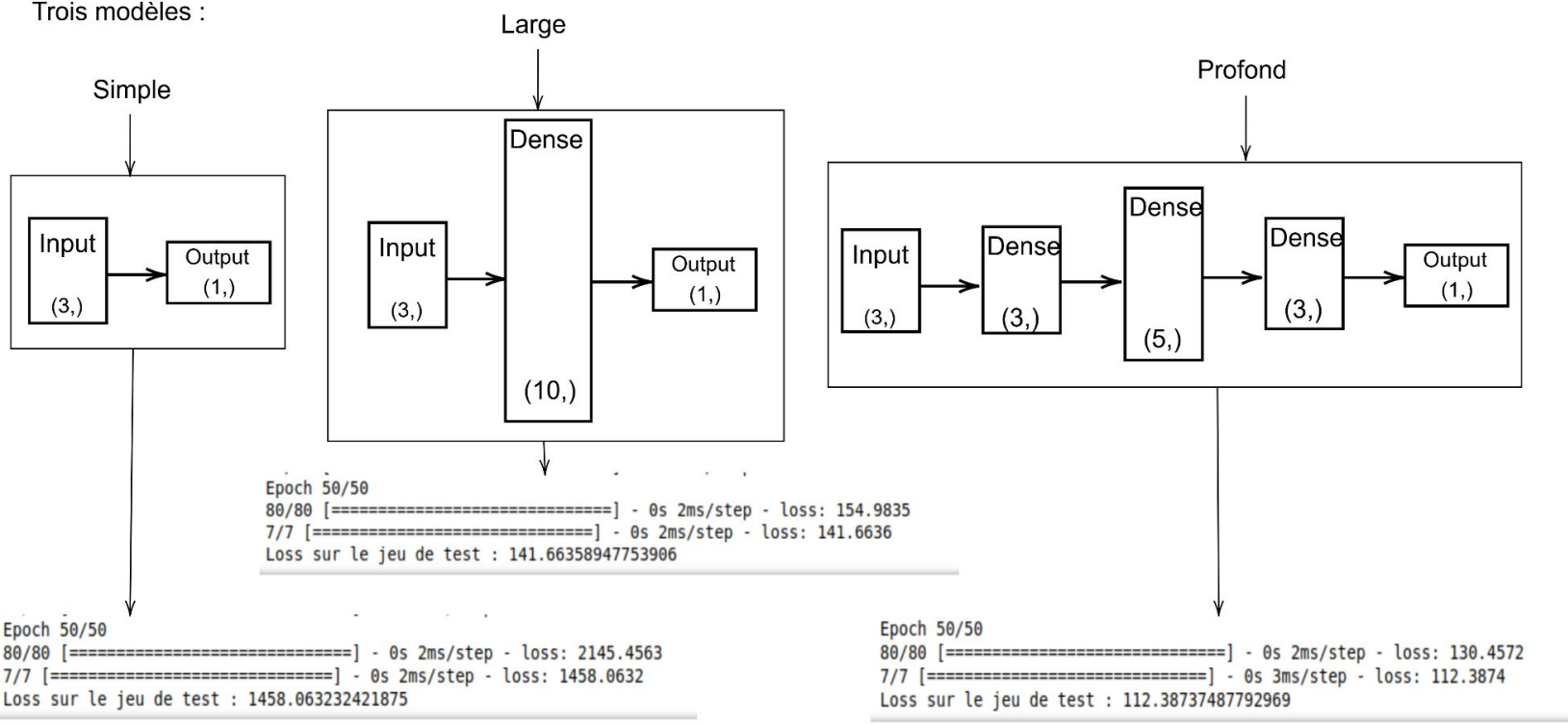
- Il faut trouver un **équilibre entre largeur et profondeur**
- **Largeur → Plus de calculs en parallèle**
- **Profondeur → Plus de complexité de calculs**



Complexité d'un problème : exemples

Régression $f(x, y) = x^2 + xy$

Trois modèles :



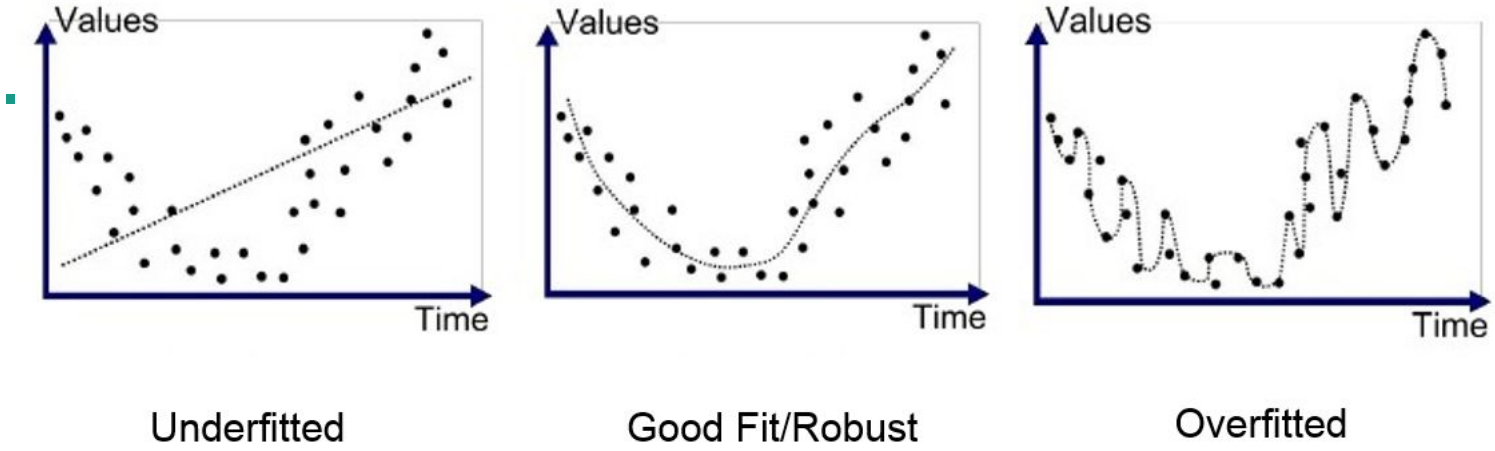

```
Epoch 50/50
80/80 [==============================] - 0s 2ms/step - loss: 154.9835
7/7 [==============================] - 0s 2ms/step - loss: 141.6636
Loss sur le jeu de test : 141.66358947753906
```

```
Epoch 50/50
80/80 [==============================] - 0s 2ms/step - loss: 2145.4563
7/7 [==============================] - 0s 2ms/step - loss: 1458.0632
Loss sur le jeu de test : 1458.063232421875
```

```
Epoch 50/50
80/80 [==============================] - 0s 2ms/step - loss: 130.4572
7/7 [==============================] - 0s 3ms/step - loss: 112.3874
Loss sur le jeu de test : 112.38737487792969
```

→ Pour les multiplications de variables, il faut baser son architecture sur la profondeur

Overfitting/Underfitting



- **Underfitting** :
  - Le **modèle** est trop **peu complexe** pour décrire le problème
  - Le **modèle** ne s'est **pas** entraîné sur **assez de données**
- **Equilibré** :
  - Le modèle décrit relativement bien le problème
  - **Bon équilibre entre complexité du modèle et nombre de données d'entraînement**
- **Overfitting** :
  - Le **modèle** est **trop complexe** pour le problème
  - Le modèle s'est **entraîné trop de fois** sur les mêmes données

Fonctions de coût

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{pred} - y_{true})^2$$

Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{pred} - y_{true}|$$

Régression

- La MSE pénalise plus les grandes erreurs
  - est sensible aux outliers
- La MAE est linéaire
  - non sensible aux outliers

Binary Cross Entropy

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_{true} * log(y_{pred}) + (1 - y_{true}) * log(1 - y_{pred})]$$

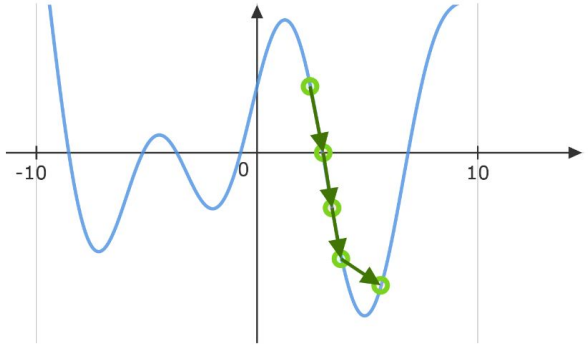
Categorical Cross Entropy

$$CCE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j}^{true} * log\left(y_{i,j}^{pred}\right)$$

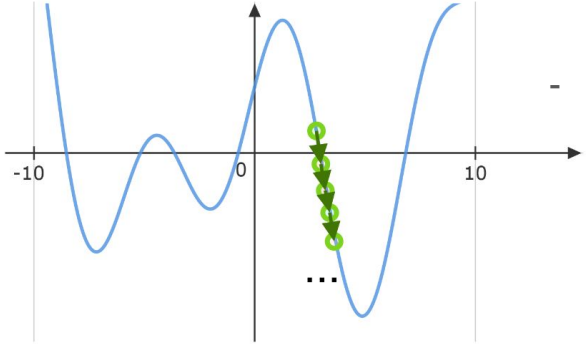
Classification

Taux d'apprentissage et Optimizer

Taux d'apprentissage ⟶ Pas de la mise à jour du gradient



Taux d'apprentissage trop grand



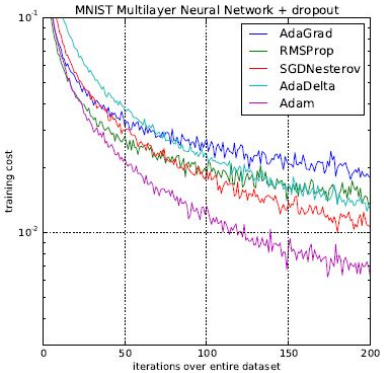
Taux d'apprentissage trop petit

- **Grand taux d'apprentissage**
  - Converge vers un **minimum non optimal**
  - **Evite les minimums locaux**
- **Petit taux d'apprentissage**
  - **Converge lentement** vers un minimum optimal
  - **Vulnérable aux minimums locaux**

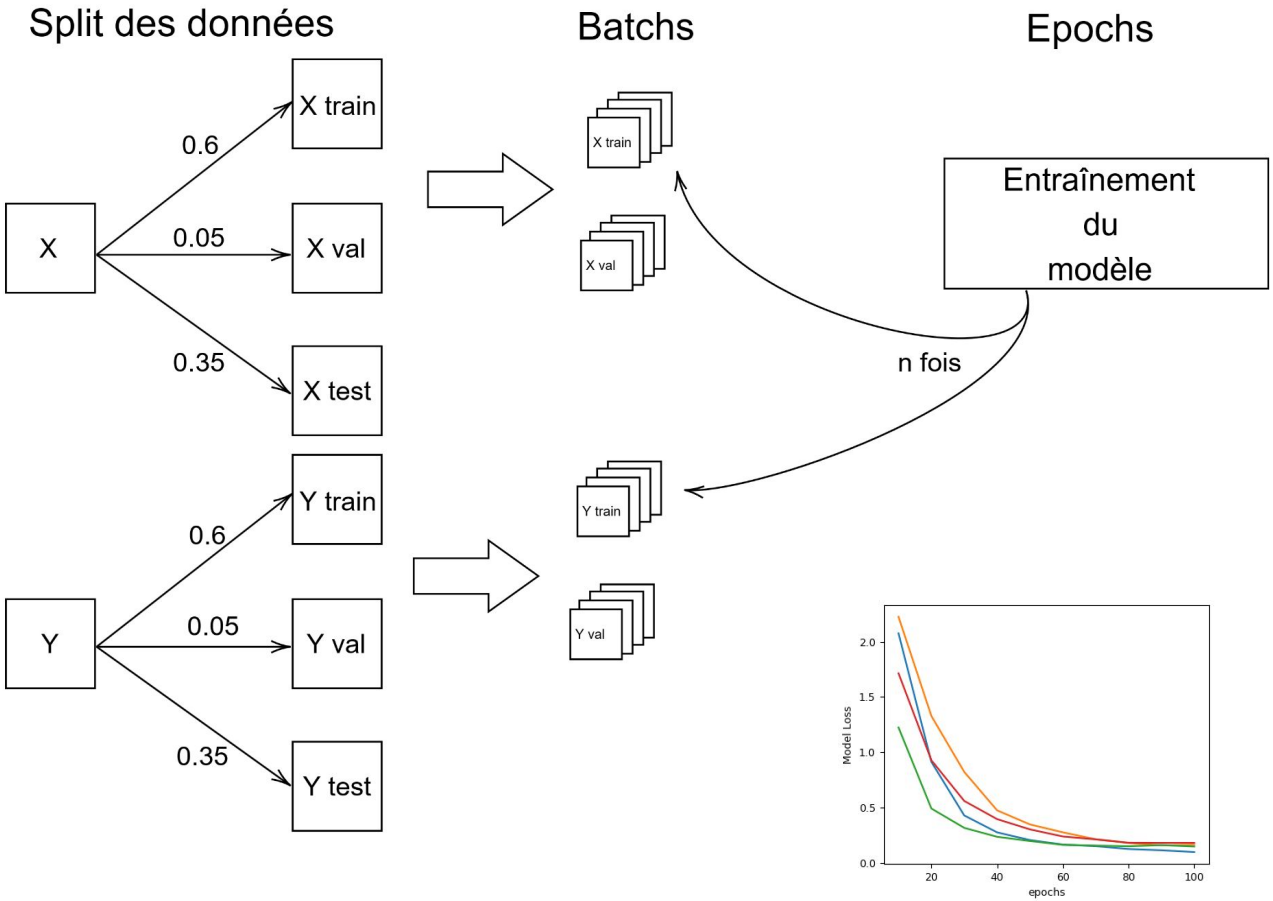
Optimizers ⟶ Amélioration de l'algorithme de gradient descendant

- Stochastic Gradient Descent : Mise à jour des poids à chaque batch
- Momentum : Garde en mémoire la somme des gradients précédents (vitesse)
- Adagrad : Adaptation du taux d'apprentissage en fonction des gradients précédents
- Adam : Mix entre Momentum et Adagrad

→ **Le meilleur optimizer dépend du problème donné**



Split, batch et epochs



- **Split** :
  - train : données d'entraînement
  - val : données de test lors de l'entraînement
  - test : données de test après l'entraînement

→ Un bon split donne le plus de données d'entraînement possible tout en gardant assez de données de validation et de test

- **Batch** :
  - Les données sont groupées dans des batchs pour l'entraînement
  - La mise à jour des poids se fait à chaque batch
  - Permet d'éviter de donner trop d'importance à des outliers
- **Epoch** : nombre de fois où le modèle s'entraîne sur les données
  - Permet d'affiner la précision du modèle petit à petit

→ L'optimisation du nombre d'epochs et de la taille des batchs permet d'optimiser le graphique de diminution des loss

Normalisation et initialisation

Normalisation

$$x' = \frac{x - min(X)}{max(X) - min(X)}$$

x normalisé

Intervalle : [0,1]

Standardisation

$$x' = \frac{x - \hat{x}}{\sigma}$$

moyenne

x normalisé — écart type

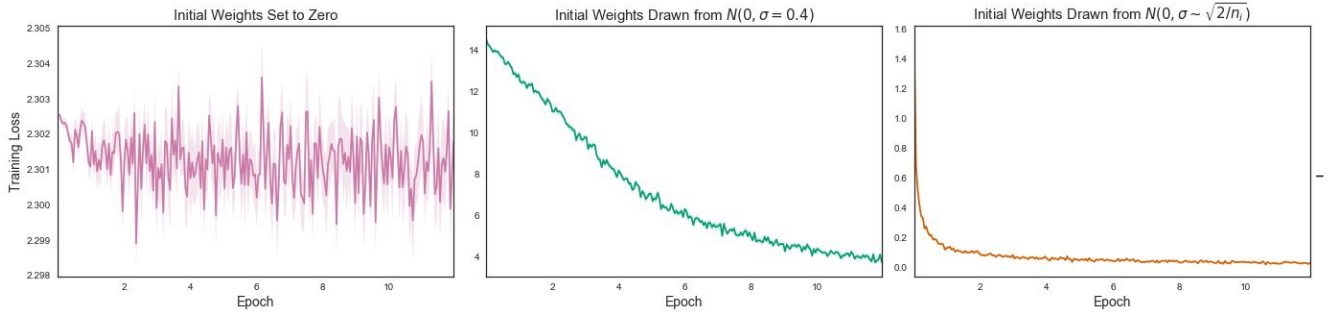
Intervalle : [-1,1]

Division

$$x' = \frac{x}{k}$$

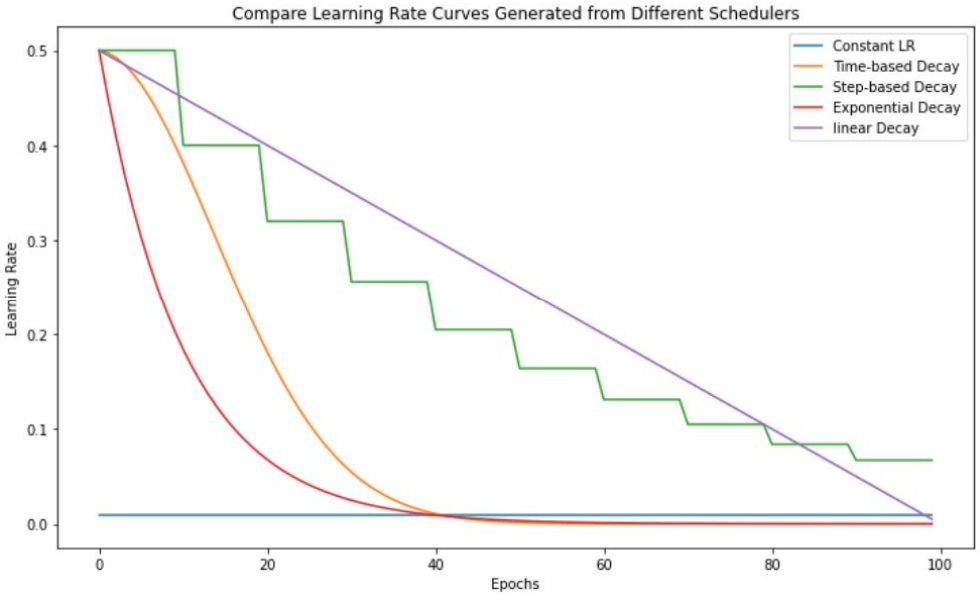
x normalisé — constante

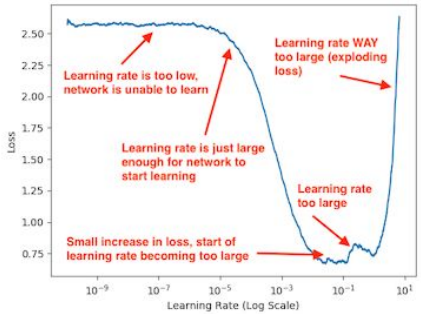
Intervalle : dépend de λ



- **Normalisation**
  - Plusieurs façons de normaliser :
    - Normalisation mathématique
    - Standardisation
    - Division par un entier
  - **Ramener les données proche des intervalles d'activation des fonctions d'activation**
- **Initialisation des poids**
  - **Permet au modèle de converger** plus souvent lors de l'entraînement
  - Exemples d'initialisation :
    - Xavier/Glorot → fonction d'activation symétriques (tanh, sigmoid)
    - He → ReLU et ses variantes
  - Pour forcer notre modèle à converger, on peut commencer l'entraînement sur une petite partie du jeu de données (ou sur une epoch) et de recréer un modèle si le modèle diverge.

Learning rate schedulers





- **Learning rate constant** (par défaut)
  - **Simple** à implémenter
  - Peut conduire à des **problèmes de convergence** trop rapide ou trop lente
- **Réduction graduelle par epoch**
  - Décroissance **continue**
  - Le **choix du taux de décroissance** est **délicat**
- **Réduction pas à pas**
  - Changement à intervalles réguliers
  - Les **changements** peuvent être trop **brusques**
- **Réduction exponentielle**
  - Ajustement rapide au début puis stabilisation
  - **Learning rate** devenant trop **petit trop rapidement**
- **Réduction linéaire**
  - Décroissance douce et linéaire
  - Solution générale mais **pas assez flexible**

Équilibrer les données d'entraînement

Personnalisation de la fonction de coût

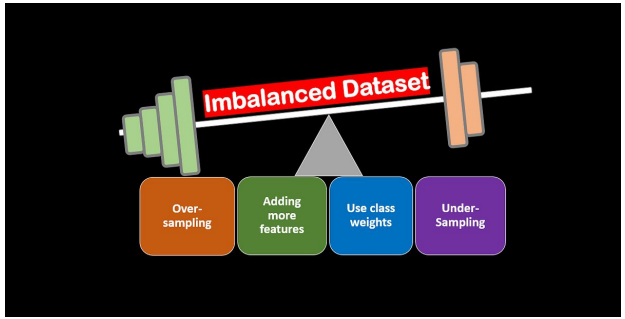
Pour des valeurs entières : $balanced\ MSE = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{f_{y_i^{true}}}\left(y_i^{true} - y_i^{pred}\right)^2$

fréquence

Pour des valeurs continues : $balanced\ MSE = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{f_{intervalle\ de\ y}}\left(y_i^{true} - y_i^{pred}\right)^2$

fréquence

On définit des intervalles pour lesquelles on équilibre les données d'entraînement



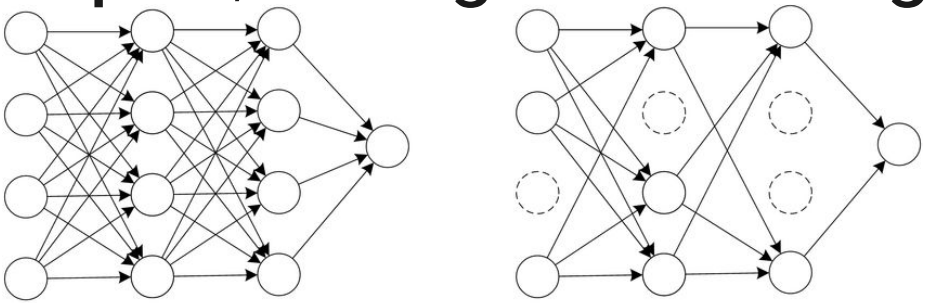
Problème :

- Les données d'entraînement ne reflète pas les données de test
- Les données d'entraînement ne sont pas équilibrées → trop de petites/grandes valeurs

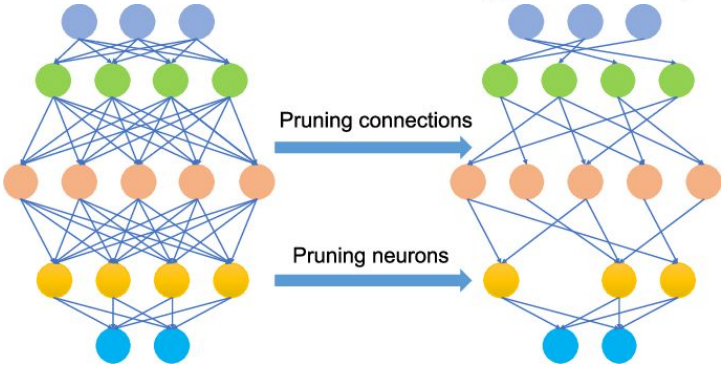
Que faire :

- **Personnalisation de la fonction de coût**
  - **Donner plus d'importance à certaines données d'entraînement**
  - (Favoriser la précision sur un output)
- **Donner un poids aux données d'entraînement**
  - **Donner plus d'importance aux données d'entraînement suivant leur occurrence**

Dropout, Pruning et L1 et L2 régularisation



(a) Standard Neural Network (b) Network after Dropout



$$L1\ Regularization\ =\ (loss\ function) + \alpha \sum_{j=1}^{p} |b_j|$$

$$L2\ Regularization\ =\ (loss\ function) + \alpha \sum_{j=1}^{p} b_j^2$$

- **Dropout : Désactiver aléatoirement des neurones durant l'entraînement**
  - Permet au réseau de ne pas trop dépendre d'un neurone
- **Pruning : Enlever les neurones dont les weights sont proches de 0**
  - Peut servir à entraîner plus vite son modèle
  - Peut servir à alléger le modèle en terme de complexité
- Les L1 et L2 régularisations consistent à **limiter la valeur des weights d'une couche**
  - Des poids trop grands peuvent refléter un modèle trop complexe