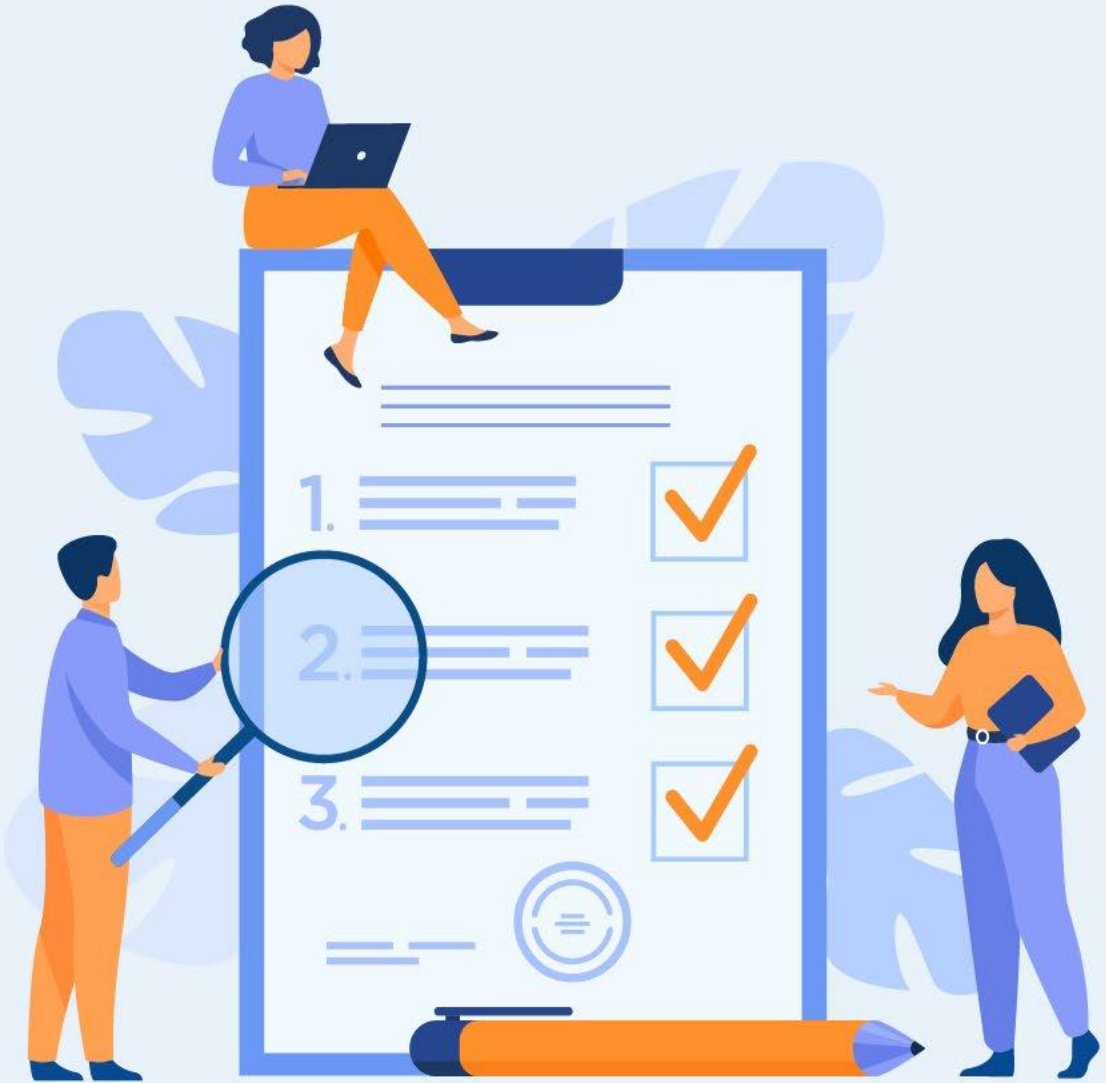


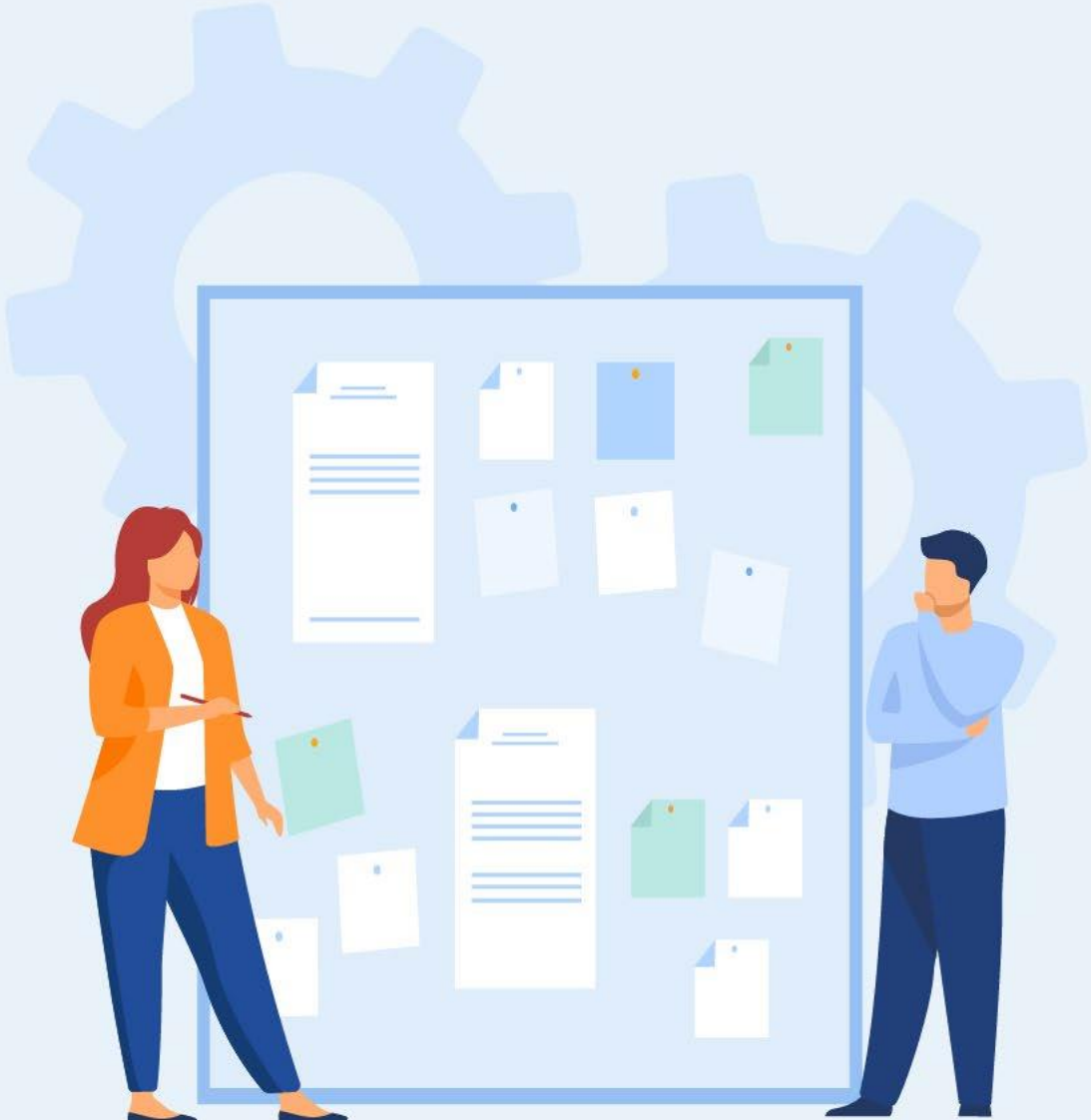
CHAPITRE 2

MAÎTRISER LES TECHNIQUES DE RÉGULARISATION

Ce que vous allez apprendre dans ce chapitre :

- Les défis de l'apprentissage
- Les différentes techniques de régularisation

08 heures



CHAPITRE 2

MAÎTRISER LES TECHNIQUES DE RÉGULARISATION

1. **Généralisation en apprentissage**
2. Techniques de régularisation
3. Avantages et inconvénients

Le Problème Principal

En apprentissage automatique, le but est de créer un modèle qui fonctionne bien **sur des données qu'il n'a jamais vues** (comme un étudiant qui réussit un examen sans avoir vu les questions à l'avance).

- **Exemple** : Si vous apprenez *par cœur* un livre de maths (*données d'entraînement*), vous échouerez face à des exercices nouveaux (*données de test*).
- **Objectif** : Apprendre les **règles générales**, pas mémoriser les exemples !

PARTIE 1

Les 2 Pièges à Éviter

1. **Sous-apprentissage (Underfitting)**

•**C'est quoi ?** Le modèle est **trop simple** pour comprendre les données.

Exemple : Votre modèle dit "Tous les animaux à 4 pattes sont des chiens".

•**Symptômes** :

- Erreurs élevées **même sur les données d'entraînement**.
- Le modèle n'apprend rien, comme un élève qui n'écoute pas en cours.

•**Solution** : Rendre le modèle plus complexe (ex: ajouter des couches en deep learning).

2. **Sur-apprentissage (Overfitting)**

•**C'est quoi ?** Le modèle est **trop compliqué** et mémorise le "bruit" des données.

Exemple : Votre modèle reconnaît *uniquement* les photos de chiens prises en plein jour avec un collier rouge.

•**Symptômes** :

- Performances parfaites sur les données d'entraînement… mais catastrophiques sur les nouvelles données.
- Comme un élève qui mémorise les réponses sans comprendre.

•**Solution** : Simplifier le modèle ou ajouter de la régularisation (ex: dropout).

Capacité d'un modèle

La **capacité**, c'est la "flexibilité" du modèle à apprendre des motifs complexes dans les données.

•**Exemple** : Imaginez un **peintre** :

- Un pinceau fin (*faible capacité*) ne peut dessiner que des formes simples.
- Un pinceau ultra-précis (*haute capacité*) peut reproduire chaque détail, même les imperfections de la toile.

Biais et variance

Le biais et la variance sont deux sources d'erreur importantes en apprentissage qui peuvent affecter les performances d'un modèle. Comprendre ces concepts est essentiel pour diagnostiquer et améliorer les modèles d'apprentissage.

Biais

Le biais, c’est quand votre modèle est trop simple pour comprendre les données.

Un modèle avec un biais élevé a tendance à être trop simpliste pour capturer la complexité des données. Cela peut conduire à une sous-représentation des informations importantes et à des performances médiocres à la fois sur les données d'entraînement et de test.

En d'autres termes, un biais élevé peut conduire à un modèle qui est trop simplifié et ne peut pas bien généraliser.

- Exemple : Comme essayer de dessiner un chat avec seulement un cercle et deux triangles.
- Problème : Le modèle rate les détails importants, fait des erreurs même sur les données d’entraînement, et ne s’améliore pas avec de nouvelles données.
- On appelle ça : Sous-ajustement (underfitting).

Variance

La **variance**, c'est quand votre modèle est **trop compliqué** et s'accroche aux détails inutiles.

- **Exemple** : Comme mémoriser par cœur une liste de mots au lieu de comprendre la règle de grammaire.
- **Problème** : Le modèle colle trop aux données d'entraînement (y compris le bruit) et échoue sur les **nouvelles données**.
- **On appelle ça** : **Surajustement** (*overfitting*).

En résumé

- **Biais élevé** → Modèle trop simple → **Underfitting** ❌.
- **Variance élevée** → Modèle trop complexe → **Overfitting** ❌.
- **Objectif** : Trouver un équilibre entre les deux (ni trop simple, ni trop compliqué),

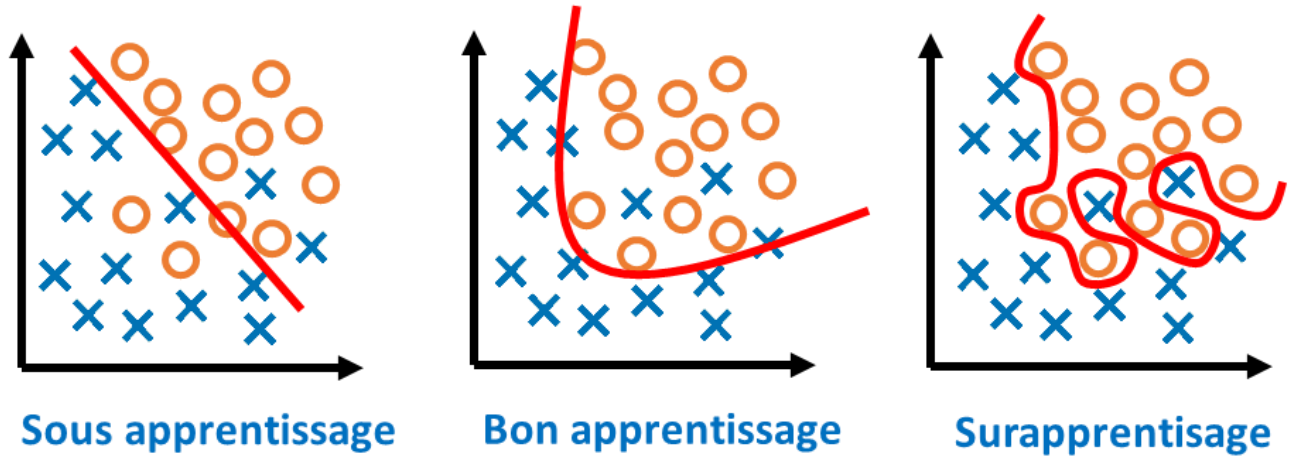
Exemple : Imaginez apprendre à reconnaître un chien :

- **Biais** : "Tout ce qui a 4 pattes est un chien" → Erreurs fréquentes (confondre avec des chats).
- **Variance** : "Seul mon chien est un vrai chien" → Impossible de reconnaître d'autres races.

Sur et sous-apprentissage

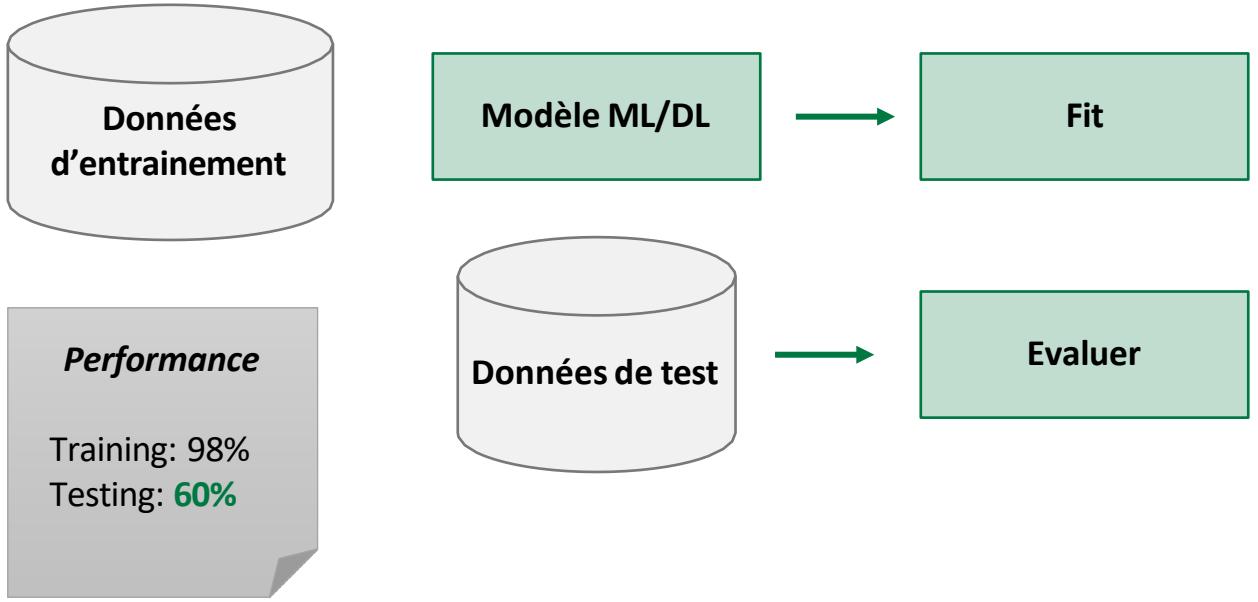
Si nous avons deux classes (notées X et O) dans un espace 2D et que le classificateur trace une ligne très droite, on a un classificateur avec un biais élevé. Cette ligne généralisera bien, ce qui signifie que l'erreur de classification pour les nouveaux points (erreur de test) sera très similaire à l'erreur de classification pour le anciens points (erreur d'entraînement). Mais le problème est que l'erreur sera trop importante en premier lieu. C'est ce qu'on appelle le **sous-ajustement**.

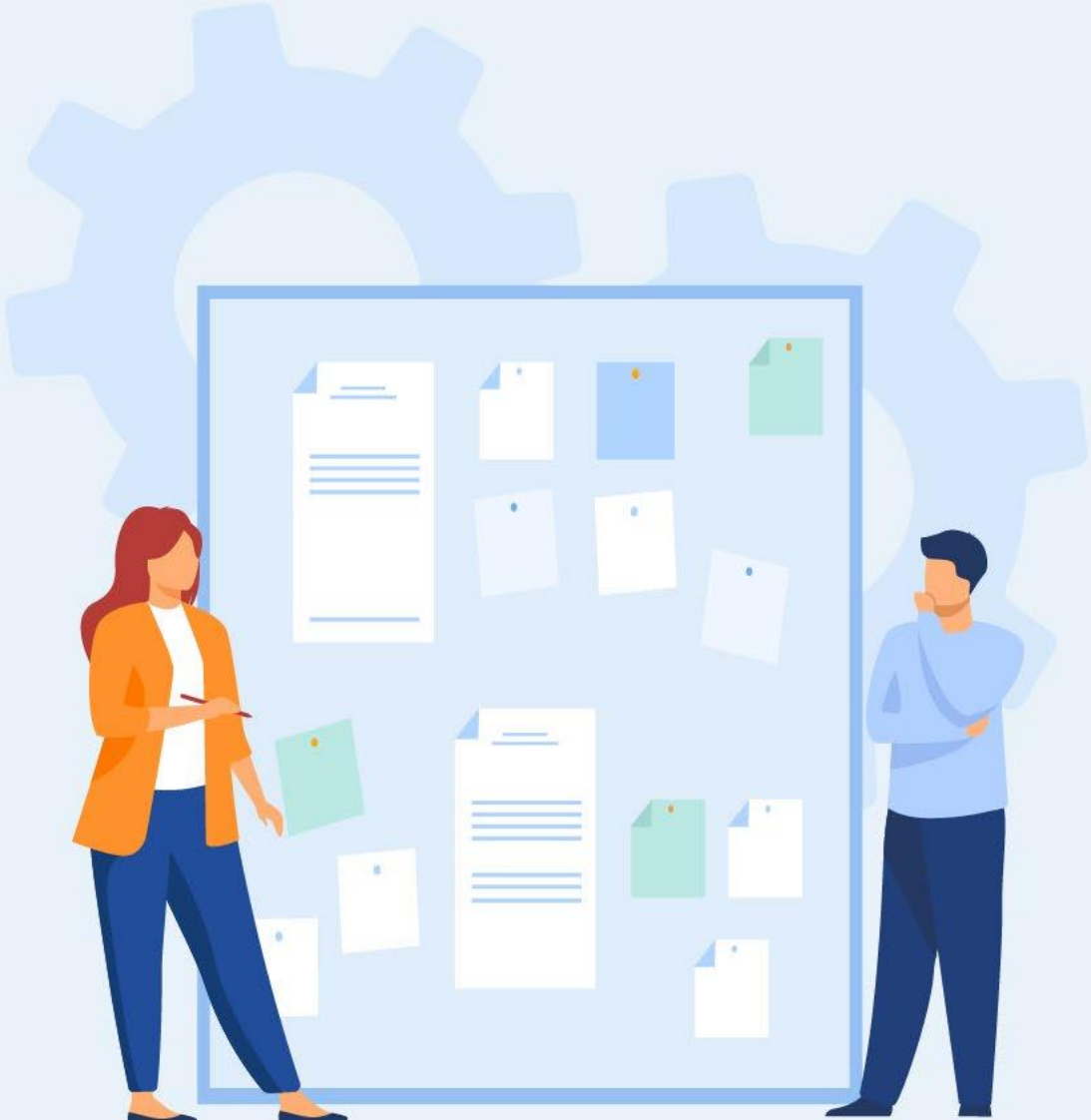
D'un autre côté, si on a un classificateur qui trace une ligne complexe pour inclure tous les X et aucun des O, alors on a une variance élevée (et un faible biais), ce qui est appelé **surajustement**. Dans ce cas, on obtient une erreur d'apprentissage relativement faible et une erreur de test beaucoup plus importante.



Comment détecter le surapprentissage ?

- ✓ Si l'erreur d'entraînement diminue rapidement tandis que l'erreur de validation augmente ou stagne, cela peut indiquer un surapprentissage.
- ✓ Si les métriques de performance de l'ensemble d'entraînement sont plus élevées que celles de l'ensemble du test, on constate que c'est un surapprentissage.





CHAPITRE 2

MAÎTRISER LES TECHNIQUES DE RÉGULARISATION

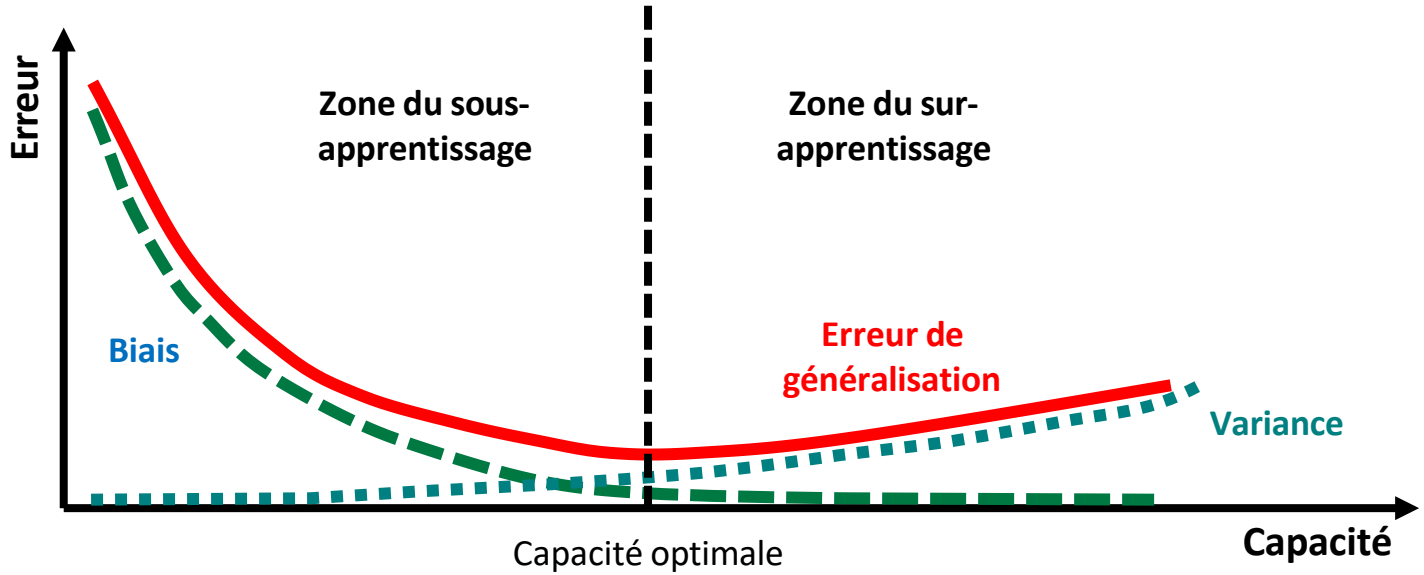
1. Généralisation en apprentissage
2. **Techniques de régularisation**
3. Avantages et inconvénients

Définition

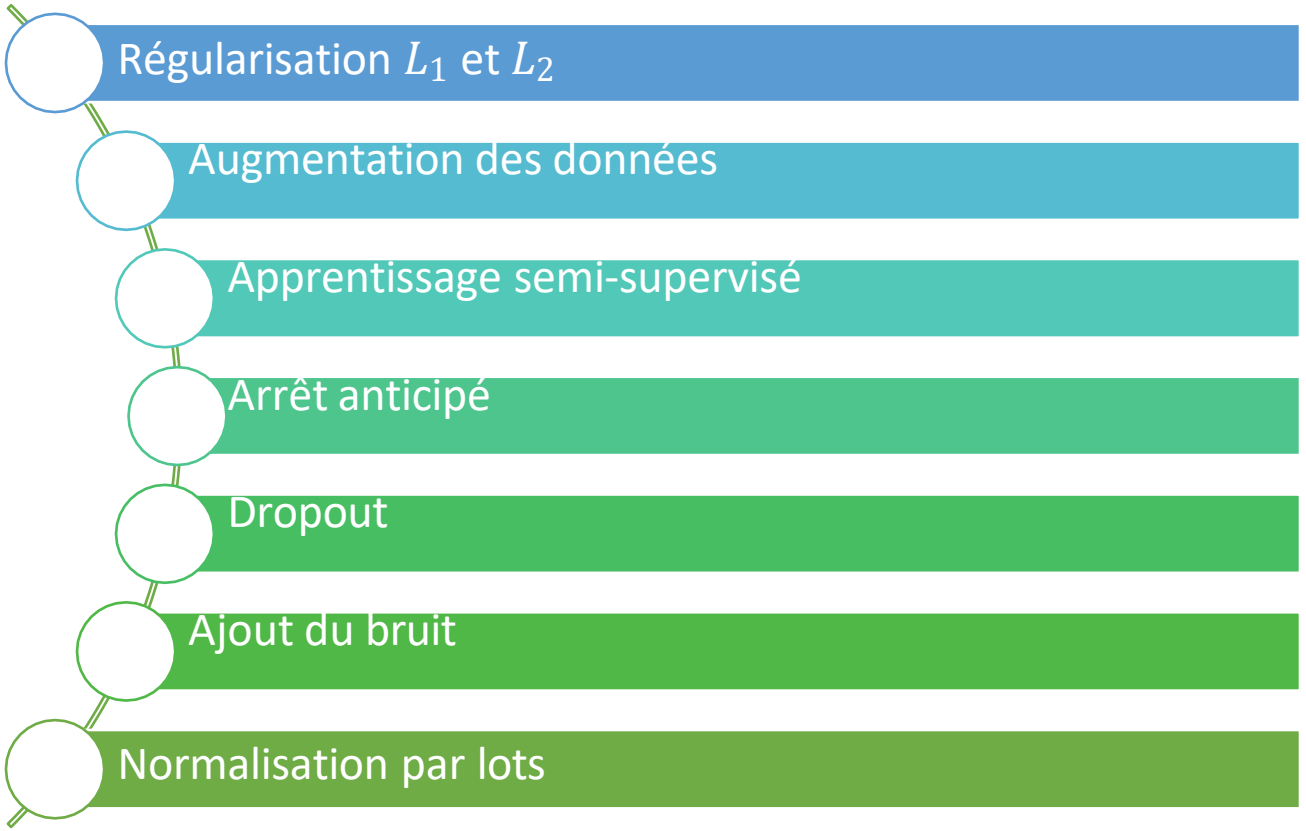
La régularisation est toute modification qu'on apporte à un algorithme d'apprentissage qui vise à réduire son erreur de généralisation mais pas son erreur d'apprentissage. On doit choisir une forme de régularisation bien adaptée à la tâche particulière.

La variance est l'erreur due à la sensibilité aux petites fluctuations de l'échantillon d'apprentissage.

Une variance élevée peut entraîner un surapprentissage, c'est-à-dire modéliser le bruit aléatoire des données d'apprentissage plutôt que les sorties prévues.



Les techniques de régularisation sont des stratégies utilisées en apprentissage automatique pour prévenir le surapprentissage (overfitting) et améliorer les performances de généralisation des modèles. Voici quelques-unes des techniques de régularisation couramment utilisées :



Régularisation L-norme

Une technique qui ajoute des "règles" au modèle pour l'empêcher de mémoriser les données.

- **L1 (Lasso)** : Ajoute une pénalité sur la *valeur absolue* des poids du réseau.

Encourage le modèle à utiliser moins de caractéristiques (comme un coach qui impose un régime strict).

Exemples :

1. Si 100 pixels décrivent un chat, L1 lui apprend à n'utiliser que les 10 plus importants.
2. Sur une image de chat, le modèle pourrait ignorer le fond (herbe/ciel) et se concentrer uniquement sur les moustaches/yeux.

```
Dense(128, activation='relu', kernel_regularizer=l1(0.01))
```

- **L2 (Ridge)** : Pénalise les *carrés* des poids pour éviter les valeurs extrêmes.

Empêche les poids du réseau de devenir trop grands (comme un coach qui interdit les excès).

Exemple :

Le modèle analyse tous les pixels, mais sans sur-interpréter ceux du fond de l'image.

Le modèle utilise tous les pixels de l'image, mais sans donner 1000x plus d'importance à un pixel spécifique.

```
Dense(128, activation='relu', kernel_regularizer=l2(0.01))
```

Augmentation des données

Un modèle d'apprentissage est mieux généralisé lorsqu'il est entraîné sur plus de données.

En pratique, la quantité de données est limitée.

Pour certaines tâches d'apprentissage, il est simple de synthétiser les données.

Le classifieur prend une entrée de grande dimension x et la labélise avec une identité de catégorie unique *y*:

- ✓ La tâche principale du classifieur est d'être invariant à une grande variété de transformations.

Générez de nouveaux échantillons *(x, y)* simplement en transformant les entrées.

L'augmentation des données est une approche difficilement généralisable à d'autres problèmes.

Augmentation des données

- ✓ Augmentation des données très efficace pour le problème de classification d'objets
- ✓ Les images sont de grande dimension et incluent une variété de variations, peuvent facilement être simulées
- ✓ La translation de quelques pixels de l'image améliore les performances
- ✓ La rotation et le changement d'échelle sont également efficaces

Original

Symétrie axiale

Rotation

Recadrage aléatoire

Changement de couleur

Addition de bruit

Perte d'informations

Changement de contraste

PARTIE 1

02 - MAÎTRISER LES TECHNIQUES DE RÉGULARISATION

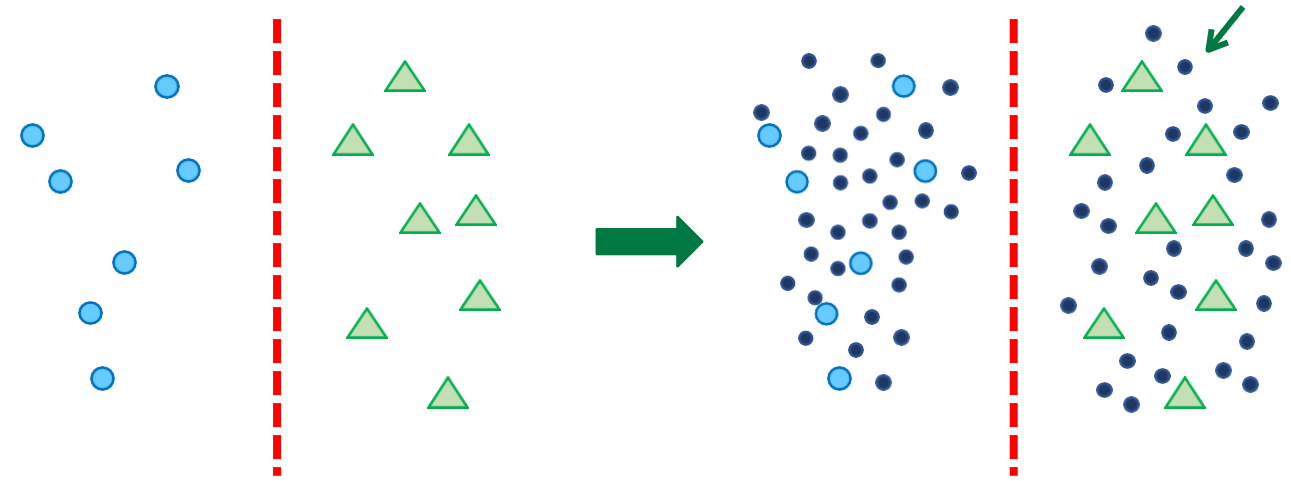
Techniques de régularisation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Création d'images modifiées
datagen = ImageDataGenerator(
    rotation_range=30,      # Rotation aléatoire jusqu'à 30°
    width_shift_range=0.2, # Décalage horizontal aléatoire (20% de la largeur)
    horizontal_flip=True,  # Miroir horizontal
    brightness_range=[0.8, 1.2] # Variation de luminosité
)

# Application sur une image de chat
img = load_img('chat.jpg')
x = img_to_array(img)
x = x.reshape((1,) + x.shape)

i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.imshow(batch[0].astype('uint8'))
    plt.axis('off')
    i += 1
    if i % 4 == 0:  # Affiche 4 variations
        break
```

Apprentissage semi-supervisé

L'apprentissage semi-supervisé est une approche de l'apprentissage automatique qui vise à exploiter à la fois des données étiquetées et non étiquetées pour entraîner un modèle. Contrairement à l'apprentissage supervisé traditionnel qui se base uniquement sur des données étiquetées, l'apprentissage semi-supervisé utilise également des données non étiquetées pour améliorer les performances du modèle

Par exemple, dans le cadre de l'analyse imagerie médicale, comme les données de tomodensitométrie ou d'IRM, les radiologues peuvent examiner et annoter une petite partie d'une tumeur ou d'une maladie. Il peut facilement collecter les données, mais l'annotation manuelle de toutes les analyses est longue et coûteuse. Par conséquent, il se compare à l'apprentissage non supervisé, l'apprentissage d'un modèle pour aider à étiqueter les données avec la conjonction de réseaux d'apprentissage profond peut être bénéfique à partir d'une petite proportion de données étiquetées et améliorer sa précision.



Pseudo-étiquetage (Pseudo-Labeling)

- **Principe** :
  - Le modèle est d'abord entraîné sur les données étiquetées.
  - Il prédit des « pseudo-labels » pour les données non-étiquetées, qu'on ajoute au jeu d'entraînement si la confiance est suffisante.
  - On ré-entraîne le modèle sur l'ensemble élargi, éventuellement de façon itérative .
- **Avantages** :
  - Très simple à mettre en œuvre.
  - Peut apporter jusqu'à +20 points de pourcentage sur des petits jeux d'images.
- **Limites** :
  - **Confirmation bias** : si les premières pseudo-étiquettes sont erronées, le modèle peut s'auto-confirmer dans ses erreurs.

Scénario

•**Données étiquetées** : 1000 images (500 chats, 500 chiens)

•**Données non-étiquetées** : 10 000 images d'animaux divers

Étapes :

1.Entraînez initialement le modèle sur les 1000 images étiquetées.

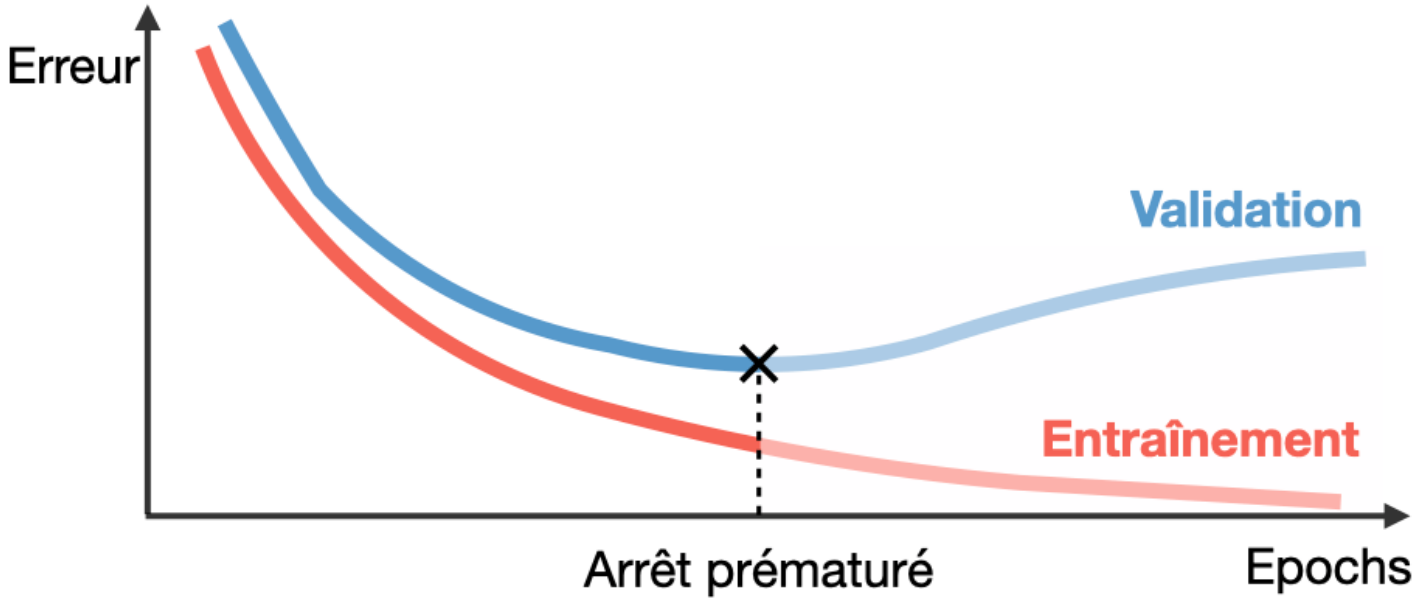
2.Utilisez ce modèle pour prédire des **pseudo-labels** sur les 10 000 images non-étiquetées.

3.Ré-entraînez avec toutes les images (étiquetées + pseudo-étiquetées).

```
# Étape 2 : Génération de pseudo-labels
unlabeled_images = load_unlabeled_data() # 10 000 images
pseudo_labels = model.predict(unlabeled_images)
# Étape 3 : Ré-entraînement
model.fit(
    x=[labeled_images, unlabeled_images],
    y=[true_labels, pseudo_labels],
    epochs=10
)
```

Arrêt anticipé (Early stopping)

L'arrêt anticipé est une technique utilisée lors de l'entraînement de modèles d'apprentissage automatique pour éviter le surapprentissage (overfitting) et améliorer la généralisation du modèle. Elle consiste à arrêter l'entraînement prématurément lorsque les performances du modèle sur un ensemble de validation commencent à se détériorer.

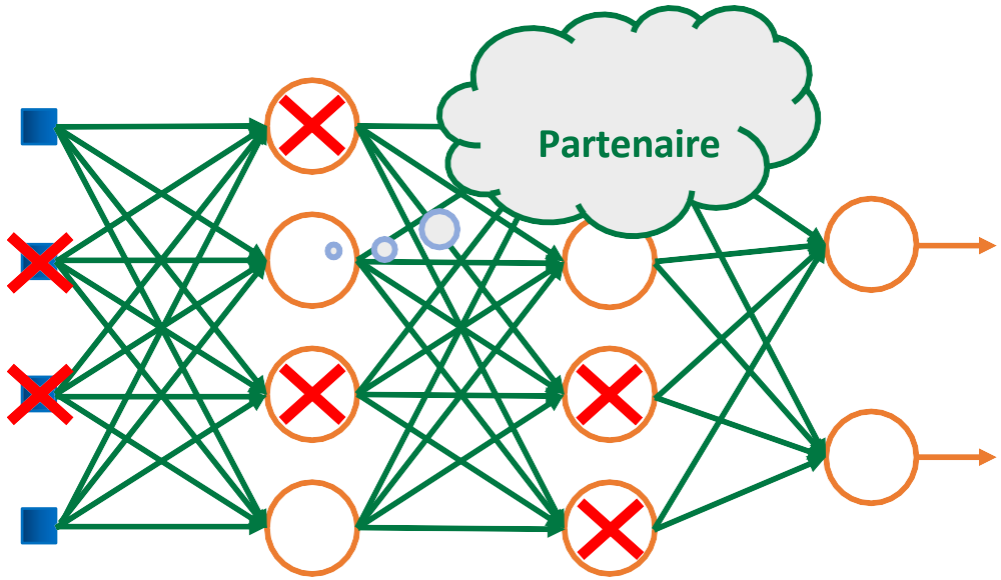


```python
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5, # Attendre 5 époques sans amélioration
    restore_best_weights=True # Garde les meilleurs poids
)

model.fit(..., callbacks=[early_stop])
```

Dropout

- ✓ Le dropout consiste à aléatoirement désactiver un certain nombre de neurones pendant l'entraînement, ce qui force le réseau à apprendre des représentations plus robustes et généralisables.
- ✓ L'idée principale derrière le dropout est d'empêcher les neurones de co-adapter ou de devenir trop dépendants les uns des autres. Lorsqu'un certain pourcentage de neurones est désactivé, le réseau doit s'appuyer sur les autres neurones qui sont toujours actifs pour effectuer les prédictions. Cela permet de répartir la responsabilité de la représentation des caractéristiques entre tous les neurones du réseau, plutôt que de laisser certains neurones dominer le processus.
- ✓ le taux de dropout (souvent autour de 20–50%).

```
model = Sequential([
    Dense(128, activation='relu'),
    Dropout(0.5), # Désactive 50% des neurones
    Dense(64, activation='relu'),
    Dropout(0.3), # Désactive 30% des neurones
    Dense(2, activation='softmax')
])
```



```
# Combinaison Optimale (Exemple Chat/Chien/Oiseau)
model = Sequential([
    # Couche d'entrée avec augmentation
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.1),

    # Architecture + régularisation
    Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.3),

    # Sortie
    Dense(3, activation='softmax')
])

# Entraînement avec early stopping
model.compile(...)
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    callbacks=[EarlyStopping(patience=3)]
)
```