Université de Monastir

Faculté des Sciences de Monastir

Resumé Deep Learning

MPDS2

November 5, 2024

Contents

1 *Applications du deep learning*

- Reconnaissance de caractères
- Reconnaissance d'Image
- Détection d'objets
- Description d'images (sous forme de text)
- Reconnaissance voix
- Traduction automatique
- Jeux

2 *Les éléments essentiels qui impactent les avancées en deep learning*

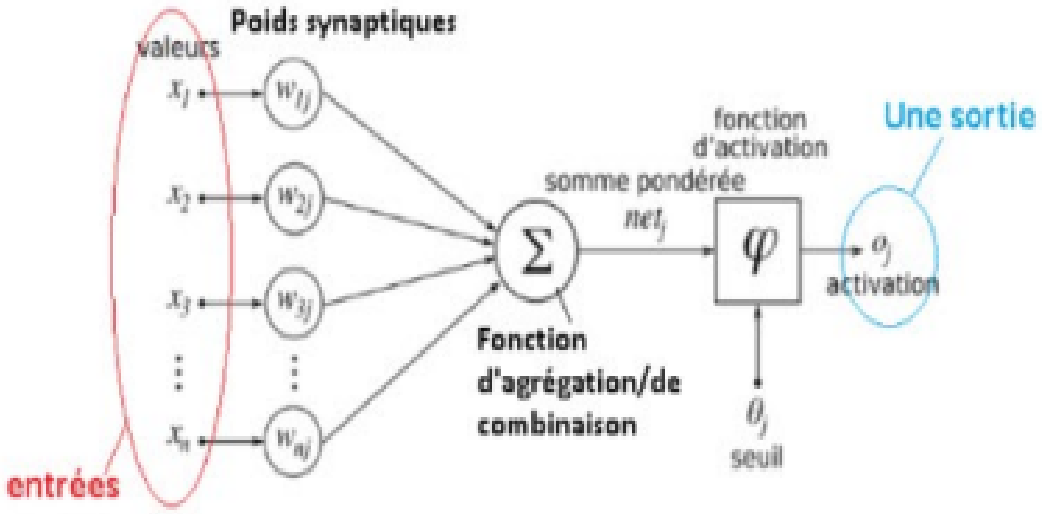
- Données variées et pertinentes
- Les capacités de calcul, généralement fournies par des processeurs puissants comme les GPU (unités de traitement graphique) ou des architectures distribuées
- L'amélioration des algorithmes de deep learning, tels que les réseaux neuronaux convolutifs (CNN), les réseaux récurrents (RNN) ou les transformers

3 *Reseaux de Neurones Artificiels ANN*

Un réseau de neurones artificiels (ANN) est un modèle d'apprentissage automatique inspiré par le fonctionnement des neurones dans le cerveau humain. • Ce modèle est utilisé pour résoudre des tâches comme la classification et la régression en apprenant à partir de données.

3.1 *Composition d'un ANN*

Un ANN est composé d'un ensemble de nœuds, appelés neurones, connectés entre eux à travers des couches (couche d'entrée, couches cachées, couche de sortie). Les ANN sont particulièrement efficaces pour modéliser des relations complexes et non linéaires dans les données. Chaque neurone reçoit des entrées pondérées, appliquant ensuite une fonction d'agrégation (somme pondérée des entrées plus un biais), suivie d'une fonction d'activation pour produire une sortie.



Rôle des fonctions d'activation : introduire la non-linéarité dans les réseaux de neurones

Importance : Permet au réseau d'apprendre des relations complexes dans les données.

Les fonctions d'activation les plus populaires :

La fonction sigmoïde

est souvent utilisée pour les problèmes de classification binaire, car elle comprime les valeurs dans l'intervalle (0, 1).

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

Fonction Tangente Hyperbolique (Tanh):

La fonction Tanh est similaire à la sigmoïde, mais elle est centrée autour de 0 et comprime les valeurs dans l'intervalle (-1, 1).

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

Fonction ReLU (Rectified Linear Unit)

La fonction ReLU est largement utilisée en raison de sa simplicité et de son efficacité pour résoudre des problèmes de gradient. Elle fixe toutes les valeurs négatives à zéro.

$$\mathbf{ReLU}(x) = \max(0, x)$$

Fonction Softmax

La fonction Softmax est principalement utilisée dans la couche de sortie pour la classification multi-classes. Elle convertit un vecteur de valeurs en probabilités, qui s'additionnent à 1.

$$\textbf{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

3.2 *Architecture d'un Réseau de Neurones*

Couche d'entrée (Input Layer):

La couche d'entrée est la première couche du réseau, où les données brutes sont introduites. Chaque neurone de cette couche représente une caractéristique spécifique des données d'entrée.
Les neurones de cette couche ne font aucun calcul ; ils transfèrent simplement les données d'entrée vers la couche suivante.

Couche cachée (Hidden Layer):

Elles reçoivent les données de la couche d'entrée, les traitent en appliquant des calculs (généralement des fonctions non linéaires comme ReLU, Sigmoid, etc.), puis les transfèrent vers la couche suivante.

Couche de sortie (Output Layer):

La couche de sortie produit la réponse finale du réseau de neurones. Elle reçoit les données transformées des couches cachées et génère la prédiction du modèle. Le nombre de neurones dans cette couche dépend de la tâche. Pour la classification multi-classes (par exemple, reconnaître si une image contient un chat, un chien ou un oiseau), la couche de sortie peut contenir un neurone pour chaque catégorie, avec une activation softmax pour fournir une probabilité pour chaque classe.

4 *Notions de base sur les réseaux neuronaux*

4.1 *Classification Binaire*

Un exemple d'entraînement unique est représenté par une paire (x, y) où x est un vecteur de caractéristiques de dimension Nx et y la classe, est soit 0, soit 1.
m exemples d'entraînement : L'ensemble de données d'entraînement se compose de m exemples, représentés comme (x(1),y(1)),(x(2),y(2)),..,(x(m),y(m))

où chaque paire (x(i),y(i)) correspond à un vecteur de caractéristiques et à son étiquette associée(classe associée)

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix}$$

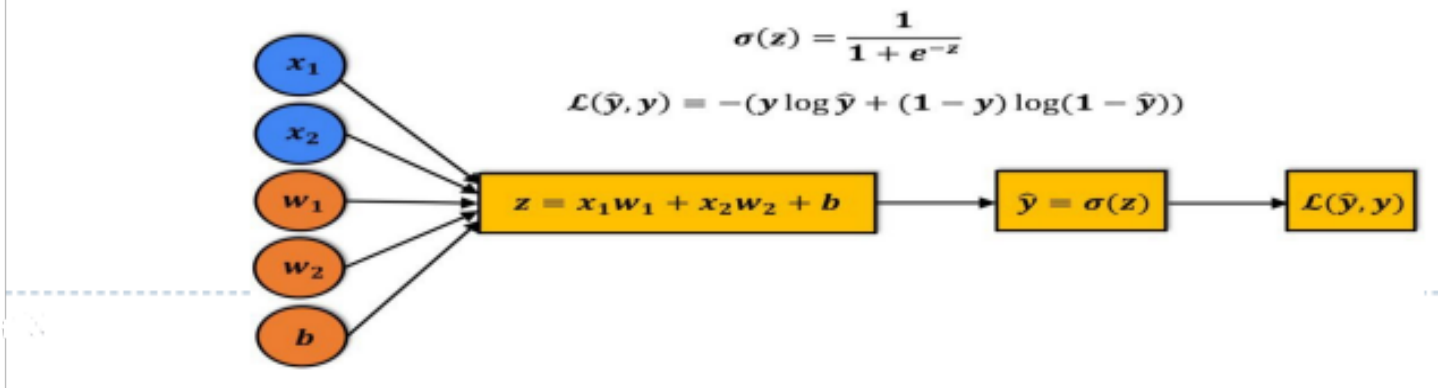
$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

La matrice X a des dimensions Nx*m où Nx est le nombre des caractéristiques (features) et m est le nombre des exemples

4.1.1 Regression Logistique

étant donné x, on cherche $\hat{y}=P(y=1|x)$, avec $0 \leq \hat{y} \leq 1$.

- **Données d'entrée** :
  - x: est un vecteur de caractéristiques avec n_x dimensions.
- **Paramètres** :
  - ω : Un vecteur de poids associé à chaque caractéristique de x.
  - B : Un biais (constante).
- **Sortie** : $\hat{y}=\sigma(z)$ où $z = \omega^T x+b$



$\sigma(z)$: **Est la fonction sigmoïde, qui transforme z en une probabilité comprie entre 0 et 1.**

- La fonction de coût pour une seule instance est définie comme suit :
  - $L(\hat{y},y)=-[y\log(\hat{y})+(1-y)\log(1-\hat{y})]$ avec $y\in\{0,1\}$ est la vraie étiquette (0 ou 1) et $\hat{y}$ est la probabilité prédite.
- Cette formule gère les deux cas possibles :
  - **Si y=1**: La fonction d'erreur devient $-\log(\hat{y})$
  - **Si y=0** : La fonction d'erreur devient $-\log(1-\hat{y})$.
- Pour l'ensemble des m exemples de notre jeu de données, la fonction de coût globale (ou moyenne) est :

$$J(\omega,b)=\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)},y^{(i)})$$

- Le rôle de la fonction d'erreur est donc d'indiquer au modèle comment ajuster les paramètres (via des techniques comme la descente de gradient) pour réduire cet écart et améliorer les prédictions.

- ## Descente de Gradient

La fonction de coût J(w,b) est la fonction qu'on cherche à minimiser pour ajuster les paramètres du modèle de régression logistique (w et b). Les paramètres sont mis à jour après chaque itération de la descente de gradient en utilisant les dérivées partielles de la fonction de coût par rapport à w et b.
La règle de mise à jour pour w et b est :

$$w := w - \alpha\frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha\frac{\partial J(w,b)}{\partial b}$$

où α est le taux d'apprentissage, un hyperparamètre qui détermine la taille des étapes.

Algorithme de Descente de Gradient:

1. Initialiser w et b avec des valeurs aléatoires.
2. Calculer ŷ à partir de la fonction sigmoïde :

$$\hat{y} = \sigma(w^T x + b)$$

3. Calculer l'erreur (fonction de coût) J(w,b).
4. Calculer les gradients pour w et b.
5. Mettre à jour w et b en suivant la règle de descente de gradient.
6. Répéter jusqu'à convergence

La Propagation en Avant

Calculer la sortie à partir des entrées en appliquant les paramètres w et b à la formule de la régression logistique

1. Calcul du Z (Combinaison linéaire des entrées)
2. Application de la Fonction d'Activation (Sigmoïde)
3. Interprétation de la Sortie :
   Si ŷ est proche de 1, le modèle prédit une probabilité élevée que y=1, sinon il prédit que y=0

La propagation en avant est la première étape clé de l'entraînement et de la prédiction d'un modèle de régression logistique. Elle permet de transformer les données d'entrée en sorties (prédictions) en appliquant les poids et les biais du modèle.

La Propagation en Arrière (Backpropagation)

Objectif : Minimiser l'erreur entre les prédictions (ŷ) et les valeurs réelles (y) en ajustant les poids et biais du réseau.
Mettre à jour les poids et les biais d'un modèle en calculant l'erreur et en propageant cette erreur en sens inverse (de la sortie vers les entrées).

1. Calcul de l'erreur (Fonction de coût)
2. Calcul des gradients (dérivées): Ces dérivées représentent comment une petite modification du poids ou du biais affecte l'erreur globale.
3. Mise à jour des poids et biais

La propagation en arrière est le mécanisme clé qui permet aux réseaux de neurones d'apprendre en ajustant leurs paramètres.

5 *KERAS :*

Keras est une bibliothèque open-source dédiée à la création de modèles de deeplearning.
Simplifie la construction de réseaux de neurones complexes
Construit sur des backends populaires comme TensorFlow
Utilisé pour des tâches telles que :
• Classification
• Régression
• Segmentation d'image
• NLP (Traitement du Langage Naturel)
Avantages :
• Facilité d'utilisation : Une interface simple et intuitive.
•Modularité : Composants indépendants facilement assemblables.
• Extensibilité : Possibilité de créer ses propres couches et modèles.
• Support GPU : Accélération avec l'utilisation des GPU.
• Large Communauté : Beaucoup de ressources et de tutoriels disponibles
voir code keras ch2 pages 22-28

6 *Réseau de Neurones à Faible Profondeur(peu profond)*

Un réseau de neurones à faible profondeur est un modèle d'apprentissage automatique avec une seule couche cachée.
• Moins complexe, plus rapide à entrainer
• Adapté aux problèmes simples
Applications :
• Tâches de classification simples (ex : reconnaissance de chiffres)
• Régression linéaire et logistique
Limites :
• Expressivité réduite : Difficulté à modéliser des relations complexes.
• Dépendance aux données : Moins performant sur les problèmes nécessitant plusieurs niveaux d'abstraction.

7 *Réseaux Profonds*

•Plusieurs couches caches
•Capacité à modéliser des relations complexes (ex : image, langage)
• Requiert plus de données et de puissance de calcul

8 *Pourquoi une représentation profonde?*

voir les exemples ch3 pages 7-12

9 *Paramètres vs Hyperparamètres dans un Réseau de Neurones*

Catégorie	**Description**
Paramètres (Modifiables lors de l'entraînement)	
Définition	Ce sont les valeurs internes que le modèle apprend automatiquement au cours de l'entraînement.
Exemples	• **Poids (Weights, W)** : Déterminent l'influence de chaque connexion entre neurones. • **Biais (Biases, b)** : Ajustent la sortie des neurones pour un meilleur ajustement.
Rôle	Les paramètres sont modifiés pendant le processus de rétropropagation (backpropagation) pour minimiser l'erreur (fonction de perte). Ils permettent au modèle de s'adapter aux données spécifiques.
Hyperparamètres (Définis par l'utilisateur)	
Définition	Ce sont les valeurs qui ne sont pas apprises automatiquement, mais qui doivent être définies avant le début de l'entraînement.
Exemples	• **Taux d'apprentissage (α)** : Contrôle la vitesse à laquelle les paramètres du modèle sont mis à jour. • **Nombre d'epochs** : Le nombre de fois que l'ensemble de données est passé dans le modèle. • **Nombre de couches cachées et neurones par couche** : Déterminent l'architecture du réseau. • **Taille du batch** : Le nombre d'exemples utilisés pour calculer la mise à jour des paramètres. • **Fonction d'activation** : ReLU, Sigmoid, etc., contrôlent le passage de l'information entre les couches.

Table 1: Différences entre les Paramètres et les Hyperparamètres dans un Réseau de Neurones

10 *comment Améliorer des réseaux profonds???*

•Ajuster le nombre de couches cachées pour équilibrer la complexité et la capacité de généralisation.
•Tester différentes tailles de couches cachées.

•Expérimenter différentes fonctions d'activation selon les besoins spécifiques.
•Learning rate : Le taux d'apprentissage doit être choisi avec soin pour éviter une convergence trop rapide ou trop lente.
•Algorithmes d'optimisation : Utilisation de méthodes comme Adam, RMSprop ou SGD avec momentum pour accélérer la convergence.

11 *la Configuration des Jeux de Données(Train, Dev, Test)*

Dans l'approche classique, la séparation des données est réalisée de la manière suivante :

- Entraînement (Train) : 60-70% des données totales.
- Développement (Validation) (Dev) : 20.%
- Test : 10-20% .

Ces ratios sont adaptés aux petits ensembles de données (par exemple 1 000 ou 10000 exemples).

Avec l'émergence des big data, ces pourcentages ne sont plus nécessaires :

- Dev et Test peuvent représenter une plus petite portion des données totales. Par exemple, pour 1 million d'exemples : 98% pour l'entraînement. 1% pour le dev. 1% pour le test (soit environ 10 000 exemples pour chaque set)

Interpritation des erreurs :

Cas	Erreur Entraînement	Erreur Validation	Interprétation
Cas 1	1%	11%	Haute variance (surapprentissage)
Cas 2	15%	16%	Haut biais (sous-apprentissage)
Cas 3	15%	30%	Haut biais et haute variance
Cas 4	0.5%	1%	Faible biais et faible variance

- Biais élevé : Le modèle n'apprend pas bien même sur les données d'entraînement. = Sous-apprentissage
- Variance élevée : Le modèle apprend trop bien les données d'entraînement mais généralise mal. = surapprentissage
- Faible biais et faible variance : Idéal, le modèle généralise bien sur les données non vues.

Pour Optimisation de Modèle

• Biais élevé :
Utiliser des modèles plus complexes (augmentation de capacité).
• Variance élevée :
Plus de régularisation (L2, dropout).
Plus de données d'entraînement pour améliorer la généralisation.
Réduction de la complexité du réseau : si le réseau est trop complexe par rapport à la tâche ou aux données disponibles.

12 *La Régularisation*

est une technique d'ajustement appliquée aux modèles d'apprentissage automatique pour réduire la complexité du modèle et éviter le surapprentissage .Elle restreint la capacité du modèle à ajuster les données d'entraînement en limitant les poids des paramètres.
Le modèle devient moins sensible au bruit dans les données d'entraînement et se concentre davantage sur les tendances générales, ce qui améliore la généralisation

Types :

•Régularisation L1 (Lasso) :
Encourage les poids des paramètres à être nuls, ce qui conduit à des modèles plus simples.
•Régularisation L2 (Ridge) :
Réduit les grandes valeurs des poids pour éviter que le modèle ne devienne trop complexe.
•Dropout :
À chaque itération, certaines unités (neurones) sont aléatoirement ignorées pendant l'entraînement, évitant ainsi la co-dépendance des neurones.

13 les Algorithmes d'Optimisation

est une méthode utilisée pour minimiser ou maximiser une fonction (généralemer la fonction de perte en deep learning).
Ces algorithmes ajustent les poids du réseau de neurones pour réduire l'erreur lors de l'entraînement.
Les algorithmes sont :

- Descente de Gradient Stochastique (SGD)
  - SGD avec Momentum, RMSprop , Adagrad
- Adam

Commencez avec Adam si vous travaillez sur un projet complexe avec beaucoup de données.

SGD est meilleur pour : Petites bases de données ou lorsqu'on veut maîtriser la convergence.

14 Descente de gradient par lots

additionne les erreurs de chaque point de l'ensemble d'entraînement et met à jour le modèle après avoir évalué tous les exemples.

ALGORITHME

Entrées :

- X **: Ensemble de données d'entraînement**
- y **: Labels associés**
- θ **: Paramètres initiaux du modèle**
- α **: Taux d'apprentissage**
- $nb\_époques$ **: Nombre d'époques**

1. **Initialiser les paramètres θ aléatoirement.**
2. **Pour chaque époque de 1 à $nb\_époques$ faire :**
   (a) **Calculer les prédictions pour toutes les données.**
   (b) **Calculer l'erreur entre les prédictions et les étiquettes réelles.**
   (c) **Calculer le gradient en utilisant toutes les données.**
   (d) **Mettre à jour les paramètres θ.**
3. **Fin pour**

Sortie : Retourner θ (les paramètres optimisés)

15 Descente de gradient stochastique

met à jour les paramètres après chaque exemple d'entraînement, en exécutant une époque d'entraînement pour chaque point.

ALGORITHME

Entrées :

- X **: Ensemble de données d'entraînement**

- y : **Labels associés**
- θ : **Paramètres initiaux du modèle**
- α : **Taux d'apprentissage**
- $nb\_époques$: **Nombre d'époques**

1. **Initialiser les paramètres θ aléatoirement.**
2. **Pour chaque époque de 1 à $nb\_époques$ faire :**
   (a) **Pour chaque exemple (x_i, y_i) dans X, y faire :**
      i. **Calculer la prédiction pour x_i.**
      ii. **Calculer l'erreur pour cet exemple.**
      iii. **Calculer le gradient pour cet exemple.**
      iv. **Mettre à jour les paramètres θ.**
   (b) **Fin pour**
3. **Fin pour**

Sortie : Retourner θ (les paramètres optimisés)

16 Descente de gradient par mini-lots

divise les données en petits lots et met à jour les paramètres pour chaque lot.

ALGORITHME

Entrées :

- X : **Ensemble de données d'entraînement**
- y : **Labels associés**
- θ : **Paramètres initiaux du modèle**
- α : **Taux d'apprentissage**
- $nb\_époques$: **Nombre d'époques**
- $taille\_mini\_lot$: **Nombre d'exemples par mini-lot**

1. **Initialiser les paramètres θ aléatoirement.**
2. **Pour chaque époque de 1 à $nb\_époques$ faire :**
   (a) **Diviser X et y en mini-lots de taille $taille\_mini\_lot$.**
   (b) **Pour chaque mini-lot (X_batch, y_batch) faire :**

i. Calculer les prédictions pour le mini-lot (X_batch)**.**

ii. Calculer l'erreur pour le mini-lot.

iii. Calculer le gradient pour le mini-lot.

iv. Mettre à jour les paramètres : $\theta = \theta - \alpha \times gradient\_batch$**.**

(c) Fin pour

3. Fin pour

Sortie : Retourner θ **(les paramètres optimisés)**

Méthode	Avantages	Inconvénients
Descente de gradient par lots	• Convergence généralement stable vers le minimum local. • Efficace en termes de calcul pour les petits ensembles de données.	• Temps de traitement long pour les grands ensembles de données, car toutes les données doivent être stockées en mémoire. • Risque de rester bloqué dans un minimum local.
Descente de gradient stochastique	• Rapidité des mises à jour et nécessite moins de mémoire. • Peut échapper au minimum local grâce aux fluctuations des gradients.	• Convergence plus bruyante (moins stable) par rapport à la descente de gradient par lots. • Moins efficace en termes de calcul pour des ensembles de données très grands.
Descente de gradient par mini-lots	• Compromis entre la descente par lots et la descente stochastique. • Moins de bruit que la descente stochastique, tout en étant plus rapide que la descente par lots.	• Nécessite un choix judicieux de la taille des mini-lots pour optimiser l'efficacité et la stabilité.

17 Ajustement dynamique du taux d'apprentissage

Technique consistant à réduire progressivement le taux d'apprentissage (learning rate) au cours de l'entraînement d'un modèle.
Démarrer avec un taux d'apprentissage relativement élevé pour avancer rapidement, puis le réduire afin de stabiliser les derniers pas vers un optimum.

Objectif :

Optimiser la performance de convergence pour obtenir de meilleurs résultats finaux sans sauts brusques autour du minimum global.

Comment?

- **Décroissance Fixe (Step Decay)**
  - Réduire le taux d'apprentissage après un nombre fixe d'époques
  - Exemple : Diviser le taux d'apprentissage par 2 toutes les 10 époques.
- **Décroissance Exponentielle**
  - Le taux d'apprentissage est réduit de manière exponentielle après chaque itération ou époque:
    Lr= Lr(initial)*exp(-decayrate * nbepoque)
    decayrate : C'est le taux de décroissance exponentielle. Un nombre plus élevé de decayrate va réduire plus rapidement le taux d'apprentissage.
- **Décroissance par Validation**
  - Basé sur les performances sur un ensemble de validation
  - Si la performance n'améliore pas, réduire le taux d'apprentissage (méthode couramment utilisée en Keras)

Pourquoi?

Stabilité de l'entraînement ,Convergence plus rapide et meilleure

18 Pourquoi ajuster les hyperparamètres

- Taux d'apprentissage (alpha) : Le plus important.

Paramètres de second ordre :

- Taille du mini-batch (pour efficacité).
- Nombre d'unités cachées (réglage de capacité).
- Nombre de couches et décroissance du taux d'apprentissage

Type de recherche	Principe	Avantage	Limites
Recherche Manuelle	• Fixer les hyperparamètres secondaires à des valeurs par défaut raisonnables. • Optimiser le taux d'apprentissage en explorant plusieurs valeurs pour identifier celle qui donne les meilleures performances. • Ajuster chaque hyperparamètre dans l'ordre d'importance, en maintenant les meilleurs paramètres trouvés.	• Réduit le nombre total de combinaisons testées. • Évite de modifier trop de paramètres en même temps, ce qui améliore la stabilité.	• Certains hyperparamètres peuvent avoir des effets croisés qui nécessitent d'être ajustés ensemble.
Recherche par Grille (Grid Search)	• Essayer toutes les combinaisons d'un petit nombre de valeurs pour chaque hyperparamètre. • Exemple de valeurs testées : – Taux d'apprentissage : 0.01, 0.001, 0.0001 – Nombre d'époques : 10, 20 – Neurones par couche : 32, 64	• Couvre toutes les combinaisons possibles dans l'espace de recherche défini.	• Coût élevé en calcul, particulièrement si plusieurs hyperparamètres doivent être testés.

Table 2: Comparaison des techniques de recherche d'hyperparamètres

Principe de normalisation mini-batch:

• La normalisation de mini-batch applique une normalisation aux activations de chaque couche dans un réseau de neurones.
• Pour chaque mini-lot d'entraînement :
- Calculer la moyenne et l'écart-type des activations pour chaque caractéristique
- Normaliser les activations en les centrant autour de la moyenne et en les redimensionnant à l'aide de l'écart-type.

Avantages :

• Amélioration de la convergence : Accélère l'apprentissage et améliore les performances du modèle.
• Réduction de l'overfitting : Agit comme une régularisation

19 Structuration des projets de machine learning

19.1 Stratégie de structuration :

Afin de maximiser l'efficacité, il est crucial d'identifier rapidement les techniques ayant un fort impact. Par exemple, dans un projet de classification d'images de chats, plusieurs solutions comme l'augmentation des

données, la diversité des exemples ou l'ajustement de l'architecture peuvent être envisagées pour améliorer la performance.

19.2 Critères de performance

-Bonne performance sur l'ensemble d'entraînement
-Bonne généralisation sur le jeu de validation
-Excellente performance sur le jeu de test
-Performance satisfaisante en conditions réelles.

19.3 Gestion des données d'entraînement, de développement et de test

Lorsqu'il existe des différences entre les ensembles, plusieurs stratégies peuvent être adoptées :

-Data Augmentation : Appliquer des transformations pour mieux représenter les conditions réelles.
-Transfert d'apprentissage : Utiliser un modèle pré-entraîné pour améliorer la généralisation sur de nouveaux ensembles de données.

19.4 Analyse des erreurs

Analyser les erreurs est essentiel pour des améliorations ciblées. Le processus inclut :
-La collecte d'exemples mal classifiés
-La classification des erreurs par type
-La quantification de l'impact de chaque catégorie d'erreurs pour prioriser les améliorations.

Pour corriger les erreurs les plus fréquentes, on peut :

- Ajouter des données spécifiques : Par exemple, si des races de chiens sont confondues avec des chats, on peut ajouter des exemples supplémentaires de ces races.
- Data Augmentation ciblée : Augmenter la diversité des images en simulant diverses conditions (angles, éclairage).
- Affinage du modèle : Entraîner un modèle plus profond ou utiliser des techniques de transfert d'apprentissage.