IA : Théorie et Algorithmes – Réseaux de neurones

make it **clever**

Sommaire

Sommaire

Sommaire

Introduction aux Réseaux de Neurones

Un réseau de neurones artificiel est un modèle computationnel inspiré du fonctionnement du cerveau humain. Il est composé d'unités de traitement simples appelées **neurones artificiels** ou **perceptrons**, connectées entre elles par des liaisons pondérées.

Principe de Base

- **Entrées** : Le réseau reçoit des données d'entrée
- **Traitement** : Les neurones traitent l'information en appliquant des fonctions mathématiques
- **Sortie** : Le réseau produit une réponse basée sur l'apprentissage effectué

Applications Principales

- Classification d'images
- Reconnaissance vocale
- Traduction automatique
- Prédiction de séries temporelles
- Diagnostic médical

Architecture des Réseaux de Neurones

Un réseau de neurones est organisé en **couches** :

2.1 Couche d'Entrée (Input Layer)

- Reçoit les données d'entrée
- Chaque neurone correspond à une caractéristique des données
- Pas de traitement, juste transmission des valeurs

2.2 Couches Cachées (Hidden Layers)

- Effectuent le traitement principal
- Peuvent être multiples (deep learning)
- Chaque neurone applique une transformation aux données

2.3 Couche de Sortie (Output Layer)

- Produit le résultat final
- Le nombre de neurones dépend du type de problème :
  - **Classification binaire** : 1 neurone
  - **Classification multi-classes** : n neurones (n = nombre de classes)
  - **Régression** : 1 neurone (valeur continue)

Types d'Architecture

Architecture Feed-Forward

```
Entrée → Couche Cachée 1 → Couche Cachée 2 → Sortie
```

- Information circule dans un seul sens
- Pas de cycles ou boucles
- Architecture la plus simple et courante

Architecture Récurrente

```
Entrée → Couche → Sortie
           ↓ ↑
         Mémoire
```

- Connexions de retour vers les couches précédentes
- Capable de traiter des séquences
- Utilisée pour le traitement du langage naturel

Fonctions d'Activation

La fonction d'activation détermine si un neurone doit être "activé" ou non. Elle introduit de la **non-linéarité** dans le réseau, permettant d'apprendre des relations complexes.

3.1 Fonction Sigmoïde

Formule : $\sigma(x) = 1 / (1 + e^{(-x)})$

Caractéristiques :

- Sortie entre 0 et 1
- Dérivable partout
- Saturante (gradient s'annule pour grandes valeurs)

Usage : Classification binaire, couches cachées (anciennement)

Fonctions d'Activation

3.2 Fonction Tanh

Formule : tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))

Caractéristiques :

- Sortie entre -1 et 1
- Centrée sur zéro
- Meilleure que sigmoïde pour les couches cachées

Fonctions d'Activation

3.3 Fonction ReLU (Rectified Linear Unit)

Formule : ReLU(x) = max(0, x)

Caractéristiques :

- Simple à calculer
- Pas de saturation pour valeurs positives
- Problème : "neurones morts" (gradient nul pour x < 0)

Usage : Fonction la plus populaire pour les couches cachées

Fonctions d'Activation

3.4 Fonction Leaky ReLU

Formule : LeakyReLU(x) = max(αx, x) où α = 0.01

Avantages :

- Résout le problème des neurones morts
- Gradient non nul même pour valeurs négatives

Fonctions d'Activation

3.5 Fonction Softmax

Formule : softmax(x_i) = e^(x_i) / Σ(e^(x_j))
Usage : Couche de sortie pour classification multi-classes
Propriété : La somme des sorties = 1 (distribution de probabilité)

Pondération des Activations

Principe des Poids

Chaque connexion entre neurones a un **poids** (weight) qui détermine l'importance de cette connexion.

Calcul de l'Activation

Pour un neurone j :

```
net_j = Σ(w_ij × a_i) + b_j
a_j = f(net_j)
```

Où :

- w_ij = poids de la connexion du neurone i vers j
- a_i = activation du neurone i
- b_j = biais du neurone j
- f = fonction d'activation

Pondération des Activations

Rôle des Biais

- **Biais** : terme constant ajouté à chaque neurone
- Permet de décaler la fonction d'activation
- Améliore la capacité d'apprentissage du réseau

Initialisation des Poids

Initialisation Aléatoire

- Poids initialisés de manière aléatoire
- Évite la symétrie parfaite
- Valeurs petites pour éviter la saturation

Pondération des Activations

Méthodes Avancées

- **Xavier/Glorot** : adapté aux fonctions sigmoïde/tanh
- **He** : adapté aux fonctions ReLU
- **LeCun** : pour réseaux profonds

Apprentissage d'un Réseau de Neurones

L'apprentissage consiste à ajuster les poids pour minimiser l'erreur entre les prédictions et les vraies valeurs.

Apprentissage Supervisé

- Utilise des données étiquetées
- Compare prédiction avec vérité terrain
- Exemples : classification, régression

Apprentissage d'un Réseau de Neurones

Apprentissage Non-Supervisé

- Pas d'étiquettes
- Trouve des structures cachées
- Exemples : clustering, réduction de dimension

Apprentissage par Renforcement

- Apprend par récompenses/punitions
- Agent interagit avec environnement
- Exemples : jeux, robotique

Apprentissage d'un Réseau de Neurones

Algorithme de Rétropropagation

Étape 1 : Propagation Avant (Forward Pass)

1. Calculer les activations de chaque couche
2. Obtenir la prédiction finale
3. Calculer l'erreur

Étape 2 : Propagation Arrière (Backward Pass)

1. Calculer le gradient de l'erreur par rapport aux poids
2. Propager l'erreur vers les couches précédentes
3. Mettre à jour les poids

Apprentissage d'un Réseau de Neurones

Formule de Mise à Jour

```
w_new = w_old - α × ∂E/∂w
```

Où :

- α = taux d'apprentissage
- ∂E/∂w = gradient de l'erreur par rapport au poids

Apprentissage d'un Réseau de Neurones

Fonctions de Coût

Erreur Quadratique Moyenne (MSE)

Usage : Régression

Formule : $MSE = (1/n) \times \Sigma(y\_true - y\_pred)^2$

Entropie Croisée

Usage : Classification

Formule : $CE = -\Sigma(y\_true \times \log(y\_pred))$

Apprentissage d'un Réseau de Neurones

Optimiseurs

Gradient Descent Stochastique (SGD)

- Mise à jour après chaque exemple
- Rapide mais bruyant

Mini-Batch Gradient Descent

- Compromis entre SGD et Batch GD
- Mise à jour sur petits groupes d'exemples

Adam

- Optimiseur adaptatif
- Combine momentum et RMSprop
- Très populaire en pratique

Modélisation d'un Réseau de Neurones

Modélisation des Données d'Entrée

Normalisation

- **Min-Max** : (x - min) / (max - min)
- **Z-score** : (x - μ) / σ
- **Objectif** : éviter la domination par certaines features

Encodage des Variables Catégorielles

- **One-Hot Encoding** : chaque catégorie devient une colonne binaire
- **Label Encoding** : attribution de nombres aux catégories

Traitement des Données Manquantes

- Imputation par moyenne/médiane
- Suppression des lignes/colonnes

Modélisation d'un Réseau de Neurones

Modélisation des Données de Sortie

Classification Binaire

- **Sortie** : 1 neurone avec sigmoïde
- **Encodage** : 0 ou 1

Classification Multi-classes

- **Sortie** : n neurones avec softmax
- **Encodage** : one-hot encoding des classes

Régression

- **Sortie** : 1 neurone linéaire
- **Valeurs** : continues, normalisées

Modélisation d'un Réseau de Neurones

Structure du Réseau

Choix du Nombre de Couches

- **Peu de couches** : problèmes simples
- **Nombreuses couches** : problèmes complexes (deep learning)
- **Règle empirique** : commencer simple, complexifier si nécessaire

Choix du Nombre de Neurones

- **Couche d'entrée** : nombre de features
- **Couches cachées** : entre taille entrée et sortie
- **Couche de sortie** : selon le problème

Modélisation d'un Réseau de Neurones

Fonction de Combinaison

La fonction de combinaison calcule l'entrée nette du neurone :

```
net = Σ(w_i × x_i) + b
```

- **Combinaison linéaire** : la plus courante
- **Autres possibles** : produit, maximum, minimum

Modélisation d'un Réseau de Neurones

Propagation de l'Information

Propagation Avant

1. **Entrée** → activation couche d'entrée
2. **Couche par couche** → calcul activations
3. **Sortie** → prédiction finale

Modélisation d'un Réseau de Neurones

Propagation de l'Information

Exemple de Calcul

Pour un réseau 2-3-1 :

```
# Couche cachée
h1 = f(w11×x1 + w21×x2 + b1)
h2 = f(w12×x1 + w22×x2 + b2)
h3 = f(w13×x1 + w23×x2 + b3)

# Couche de sortie
y = f(w1×h1 + w2×h2 + w3×h3 + b_out)
```

Les MLP (Multi-Layer Perceptron)

Le **Perceptron Multi-Couches** (MLP) est un type de réseau de neurones feed-forward avec au moins une couche cachée.

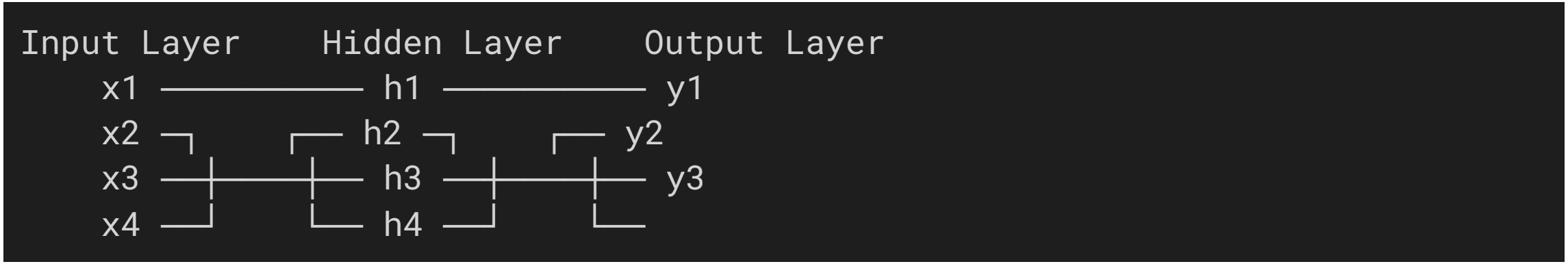
Structure d'un MLP

Caractéristiques

- **Architecture** : feed-forward
- **Couches** : au moins 3 (entrée, cachée, sortie)
- **Connexions** : complètement connecté
- **Activation** : non-linéaire dans les couches cachées

Les MLP (Multi-Layer Perceptron)

Représentation



Les MLP (Multi-Layer Perceptron)

Apprentissage du MLP

1. **Initialisation** : poids et biais aléatoires
2. **Pour chaque époque** :
    - Pour chaque batch :
        - Forward pass
        - Calcul de l'erreur
        - Backward pass
        - Mise à jour des paramètres
3. **Évaluation** : test sur données de validation

Les MLP (Multi-Layer Perceptron)

Hyperparamètres Importants

- **Taux d'apprentissage** (α) : 0.001 - 0.1
- **Nombre d'époques** : 100 - 1000+
- **Taille des batches** : 16 - 256
- **Architecture** : nombre de couches et neurones
- **Fonction d'activation** : ReLU, tanh, sigmoid

Les MLP (Multi-Layer Perceptron)

Avantages et Inconvénients

Avantages

- **Universalité** : peut approximer toute fonction continue
- **Flexibilité** : adaptable à différents problèmes
- **Performance** : excellents résultats sur de nombreuses tâches

Inconvénients

- **Boîte noire** : difficile à interpréter
- **Surapprentissage** : risque d'overfitting
- **Hyperparamètres** : nombreux paramètres à tuner

Convolutional Neural Networks (CNN)

Introduction aux CNN

Les réseaux de neurones convolutifs (**CNN**) constituent une révolution dans le traitement des données structurées spatialement. Contrairement aux réseaux de neurones traditionnels qui "aplatissent" les images en vecteurs 1D, perdant ainsi l'information spatiale cruciale, les CNN préservent et exploitent cette structure bidimensionnelle.

Principe fondamental : Réduction progressive des dimensions tout en préservant l'information essentielle. Une image de 224x224 pixels (150 000 valeurs) est transformée en quelques centaines de caractéristiques représentatives.

Inspiration biologique : Le cortex visuel humain traite les informations visuelles de manière hiérarchique. Les neurones des premières couches détectent des éléments simples (lignes, contours), puis les couches supérieures combinent ces informations pour reconnaître des formes complexes (visages, objets). Les CNN reproduisent ce processus avec des filtres apprenables.

Applications concrètes : Reconnaissance d'images médicales, détection d'objets dans la conduite autonome, classification de produits en e-commerce, filtres de réseaux sociaux, systèmes de surveillance intelligente.

Couche de Convolution (CONV) - Détails Approfondis

La convolution est l'opération mathématique au cœur des CNN. Elle simule la façon dont notre cerveau détecte des motifs dans les images.

Mécanisme technique :

- Un **filtre (noyau)** de taille fixe (ex: 3x3, 5x5) contient des poids apprenables
- Ce filtre "balaye" systématiquement l'image d'entrée pixel par pixel
- À chaque position, il calcule le produit scalaire entre ses poids et la zone locale de l'image
- Le résultat forme une **feature map** révélant où le motif recherché est présent

Exemple concret : Un filtre Sobel horizontal `[[-1,0,1], [-1,0,1], [-1,0,1]]` détecte les contours verticaux. Quand il passe sur une transition noir→blanc, le produit scalaire est élevé, révélant un contour.

Formule mathématique :

$$s(i,j) = \sum_m \sum_n X(i+m, j+n) \cdot W(m,n) + b$$

Avantages clés :

- **Partage des poids :** Un même filtre est réutilisé sur toute l'image, réduisant drastiquement le nombre de paramètres
- **Translation invariance :** Un motif est détecté quelle que soit sa position dans l'image
- **Connectivité locale :** Chaque neurone ne traite qu'une zone restreinte, mimant la réalité biologique

Paramétrage d'une Convolution - Guide Pratique

Le choix des paramètres de convolution détermine la qualité et l'efficacité du réseau.

1. Taille du filtre (Kernel Size) :

- **3x3** : Standard actuel, optimal pour la plupart des tâches
  - Capte des détails fins (textures, petits objets)
  - Empilable : deux 3x3 = champ réceptif d'un 5x5 mais avec moins de paramètres
- **5x5, 7x7** : Pour des motifs plus larges, coûteux en calculs
- **1x1** : "Pointwise convolution", change seulement la profondeur, utilisé dans les architectures modernes

2. Stride (Pas de déplacement) :

- **Stride=1** : Maximum d'information préservée, sortie de taille similaire
- **Stride=2** : Réduit la taille de moitié, alternative au pooling
- **Stride>2** : Perte d'information significative, rarement utilisé

Paramétrage d'une Convolution - Guide Pratique

3. Padding (Remplissage) :

- **"Valid"** (pas de padding) : Taille_sortie = (Taille_entrée - Kernel_size + 1)
- **"Same"** (avec padding) : Taille_sortie = Taille_entrée
- Le padding évite la perte d'information aux bordures, crucial pour les petites images

Calcul de la taille de sortie :

$$[Taille_{sortie} = \frac{Taille_{entré\,e} + 2 \times Padding - Kernel_{size}}{Stride} + 1]$$

Couche de Pooling (POOL) - Réduction Intelligente

Le pooling constitue une étape de sous-échantillonnage intelligent qui préserve l'information essentielle tout en réduisant la complexité computationnelle.

Objectifs multiples :

- **Réduction dimensionnelle :** Une image 100x100 devient 50x50 avec un pooling 2x2
- **Invariance aux translations :** Un objet légèrement déplacé reste détectable
- **Réduction du sur-apprentissage :** Moins de paramètres = meilleure généralisation
- **Augmentation du champ réceptif :** Les couches suivantes "voient" une zone plus large

Couche de Pooling (POOL) - Réduction Intelligente

Types de Pooling détaillés :

Max Pooling (le plus utilisé) :

- Prend la valeur maximale dans chaque fenêtre
- Préserve les activations fortes (caractéristiques saillantes)
- Exemple : fenêtre [2,1,3,0] → sortie = 3
- Ajoute une robustesse au bruit

Average Pooling :

- Calcule la moyenne des valeurs
- Plus lisse, préserve l'information globale
- Utilisé dans les dernières couches de certaines architectures

Global Average Pooling :

- Remplace les couches fully-connected finales
- Réduit drastiquement le nombre de paramètres
- Une feature map 7x7 devient une seule valeur

Paramètres typiques : Fenêtre 2x2 avec stride=2 (pas de recouvrement)

Couches de Correction (Activation) - Non-linéarité Essentielle

Les fonctions d'activation introduisent la non-linéarité indispensable pour apprendre des relations complexes. Sans elles, un réseau de neurones reste une simple transformation linéaire.

ReLU (Rectified Linear Unit) - Le standard actuel :

Avantages :

- Calcul ultra-rapide (simple comparaison)
- Pas de saturation pour les valeurs positives
- Résout le problème du gradient vanishing
- Sparsité naturelle (beaucoup de zéros en sortie)

Variantes de ReLU :

- **Leaky ReLU :** f(x) = max(0.01x, x) → évite les "neurones morts"
- **ELU :** Smooth pour les valeurs négatives
- **Swish :** f(x) = x × sigmoid(x) → performances supérieures sur certaines tâches

Couches de Correction (Activation) - Non-linéarité Essentielle

Sigmoïde - Usage spécialisé :

- Sortie entre 0 et 1, interprétable comme probabilité
- Utilisée principalement en sortie pour classification binaire
- Problème : saturation aux extrêmes (gradient ≈ 0)

Tanh - Alternative centrée :

- Sortie entre -1 et 1, centrée sur 0
- Meilleure que sigmoïde pour les couches cachées
- Toujours le problème de saturation

Couche Entièrement Connectée (FC) - Intégration Finale

La couche fully-connected (dense) représente le "cerveau décisionnel" du CNN, intégrant toutes les caractéristiques extraites pour produire la classification finale.

Architecture et fonctionnement :

- **Connexion totale :** Chaque neurone reçoit toutes les sorties de la couche précédente
- **Transformation :** Vecteur_entrée × Matrice_poids + Biais = Vecteur_sortie
- **Rôle :** Combiner les caractéristiques locales en décision globale

Exemple détaillé - Classification CIFAR-10 :

1. **Entrée :** Image 32x32x3 (3072 pixels)
2. **Après convolutions/pooling :** Feature map 4x4x128 (2048 valeurs)
3. **Flatten :** Conversion en vecteur 1D de 2048 éléments
4. **FC1 :** 2048 → 512 neurones (apprentissage de combinaisons)
5. **FC2 :** 512 → 10 classes (chiens, chats, avions...)

Couche Entièrement Connectée (FC) - Intégration Finale

Calcul des paramètres :

- FC1 : (2048 × 512) + 512 = 1,049,088 paramètres
- FC2 : (512 × 10) + 10 = 5,130 paramètres

Problématiques modernes :

- **Surparamètrage :** Les FC représentent souvent 80% des paramètres totaux
- **Solutions :** Global Average Pooling, architectures fully-convolutional
- **Dropout :** Essentiel pour éviter le surapprentissage dans ces couches denses

Couche de Perte (LOSS) - Optimisation Dirigée

La fonction de perte quantifie l'écart entre prédictions et réalité, guidant l'apprentissage via la rétropropagation. C'est le "GPS" qui indique au réseau comment s'améliorer.

Cross-Entropy Loss - Classification multi-classes :

$$[L = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})]$$

- **N** : nombre d'échantillons, **C** : nombre de classes
- **y** : vraies étiquettes (one-hot), **ŷ** : prédictions (softmax)

Exemple concret : Classification chat/chien

- Vraie étiquette : [1, 0] (chat)
- Prédiction : [0.8, 0.2] → Loss = -log(0.8) = 0.22
- Mauvaise prédiction : [0.3, 0.7] → Loss = -log(0.3) = 1.20

Propriétés importantes :

- **Pénalité asymétrique :** Plus on s'éloigne, plus la pénalité augmente exponentiellement
- **Probabilités :** Force le réseau à donner des probabilités calibrées
- **Gradient informatif :** Fournit des signaux d'erreur proportionnels à l'ampleur de l'erreur

Couche de Perte (LOSS) - Optimisation Dirigée

Mean Squared Error (MSE) - Régression :

$[L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2]$

- Utilisé pour prédire des valeurs continues (prix, température...)
- Pénalise quadratiquement les erreurs importantes
- Plus sensible aux outliers que l'erreur absolue

Exemples de CNN connus - Évolution Architecturale

LeNet-5 (1998) - Le précurseur :

- **Structure :** CONV→POOL→CONV→POOL→FC→FC
- **Innovation :** Première application réussie sur MNIST (chiffres manuscrits)
- **Limitation :** Seulement ~60k paramètres, inadapté aux images complexes

AlexNet (2012) - La révolution :

- **Impact :** Victoire écrasante sur ImageNet (15.3% → 10.9% d'erreur)
- **Innovations clés :**
  - Premier usage massif du GPU (2 cartes NVIDIA GTX 580)
  - ReLU au lieu de tanh/sigmoïde
  - Dropout pour éviter le surapprentissage
  - Data augmentation (rotation, flip, crop)
- **Architecture :** 8 couches, 60M paramètres

Exemples de CNN connus - Évolution Architecturale

VGGNet (2014) - La profondeur systématique :

- **Principe :** Filtres 3x3 exclusivement, empilage profond
- **Versions :** VGG-16 (16 couches), VGG-19 (19 couches)
- **Avantage :** Architecture très régulière, facile à implémenter
- **Inconvénient :** 138M paramètres, très lourd en mémoire

ResNet (2015) - Le saut quantique :

- **Innovation révolutionnaire :** Skip connections (connexions résiduelles)
- **Problème résolu :** Dégradation des réseaux très profonds (50+ couches)
- **Principe :** f(x) = x + F(x) au lieu de f(x) = F(x)
- **Impact :** Permet des réseaux de 152+ couches avec de meilleures performances

Choix des Hyperparamètres - Stratégies Pratiques

Le choix des hyperparamètres détermine le succès ou l'échec d'un projet CNN. Voici une approche méthodique basée sur l'expérience pratique.

1. Nombre de filtres - Progression géométrique :

- **Première couche :** 16-64 filtres (détection de contours basiques)
- **Couches intermédiaires :** Doubler à chaque niveau (64→128→256→512)
- **Dernières couches :** Jusqu'à 512-1024 filtres (concepts complexes)

Exemple progressif - Classification d'images :

```
Input (224x224x3)
→ CONV 64 filtres 3x3 → (224x224x64)
→ POOL 2x2 → (112x112x64)
→ CONV 128 filtres 3x3 → (112x112x128)
→ POOL 2x2 → (56x56x128)
→ CONV 256 filtres 3x3 → (56x56x256)
```

Choix des Hyperparamètres - Stratégies Pratiques

2. Forme du filtre - 3x3 domine :

- **3x3** : Optimal dans 90% des cas
  - Champ réceptif efficace avec peu de paramètres
  - Deux 3x3 = un 5x5 mais avec 18 vs 25 paramètres
- **1x1** : Réduction de dimensionnalité, changement de channels
- **5x5, 7x7** : Réservés aux premières couches ou cas spéciaux

3. Pooling - Configuration standard :

- **Max Pooling 2x2, stride=2** : Réduit de moitié sans recouvrement
- **Position :** Après chaque bloc de 2-3 convolutions
- **Alternative moderne :** Convolution avec stride=2 (learned pooling)

Méthodes de Régularisation (1/2) - Contrôle du Surapprentissage

La régularisation est cruciale pour éviter que le modèle mémorise les données d'entraînement au lieu d'apprendre des règles générales.

Approche Empirique - Validation croisée :

- **Principe :** Tester différentes configurations sur un ensemble de validation
- **Méthodologie :**
    1. Diviser les données : 60% train, 20% validation, 20% test
    2. Entraîner plusieurs modèles avec hyperparamètres différents
    3. Sélectionner le meilleur sur validation
    4. Évaluation finale sur test (une seule fois)
- **Outils :** Grid search, random search, optimisation bayésienne

Dropout - Désactivation aléatoire :

- **Mécanisme :** Pendant l'entraînement, mettre aléatoirement des neurones à 0
- **Taux typiques :** 0.2-0.3 pour convolutions, 0.5-0.8 pour FC
- **Effet :** Force chaque neurone à être utile indépendamment des autres
- **Analogie :** Comme s'entraîner avec une équipe réduite pour être plus polyvalent

Exemple pratique :

```
# Avec dropout 0.5 sur 1000 neurones
# À chaque forward pass : ~500 neurones actifs aléatoirement
# Force le réseau à ne pas dépendre de neurones spécifiques
```

Méthodes de Régularisation (1/2) - Contrôle du Surapprentissage

Data Augmentation - Multiplication artificielle :

- **Transformations géométriques :** Rotation (±15°), flip horizontal, zoom (0.8-1.2x)
- **Transformations photométriques :** Changement luminosité/contraste, bruit gaussien
- **Objectif :** Un dataset de 10k images devient virtuellement 100k+ variations
- **Impact :** Amélioration typique de 2-5% de précision

Méthodes de Régularisation (2/2) - Techniques Avancées

Régularisation Explicite - Pénalisation mathématique :

L2 Regularization (Weight Decay) :

$$[L_{total} = L_{data} + \lambda \sum_i w_i^2]$$

- **Effet :** Pénalise les poids de grande magnitude
- **Résultat :** Poids plus petits, décisions plus "douces"
- **λ typique :** 10^{-4} à 10^{-2} selon la taille du réseau
- **Avantage :** Évite les oscillations brutales dans l'espace des décisions

L1 Regularization :

$$[L_{total} = L_{data} + \lambda \sum_i |w_i|]$$

- **Effet :** Favorise la sparsité (beaucoup de poids = 0)
- **Usage :** Compression de modèle, sélection automatique de features

Méthodes de Régularisation (2/2) - Techniques Avancées

Contrôle de la Taille du Réseau :

- **Principe de parcimonie :** Commencer petit, augmenter si nécessaire
- **Méthode :** Doubler progressivement les couches/filtres jusqu'à convergence
- **Indicateurs :**
  - Surapprentissage : Écart train/validation > 5-10%
  - Sous-apprentissage : Plateau précoce sur les deux courbes

Weight Clipping - Limitation brutale :

- **Mécanisme :** Si |w| > seuil, alors w = sign(w) × seuil
- **Usage spécialisé :** GANs (Wasserstein), réseaux récurrents
- **Seuil typique :** 0.01 à 0.1 selon l'architecture

Early Stopping - Arrêt optimal :

- **Surveillance :** Loss de validation qui remonte pendant N époques
- **Patience :** N = 5-20 selon la vitesse de convergence
- **Sauvegarde :** Garde les poids du meilleur epoch sur validation

Introduction aux Réseaux Récurrents

Les **Réseaux de Neurones Récurrents (RNN)** révolutionnent le traitement des données séquentielles en introduisant la notion de "mémoire" dans les réseaux de neurones. Contrairement aux réseaux feedforward qui traitent chaque entrée indépendamment, les RNN maintiennent un état interne qui évolue au fil du temps.

Principe fondamental : À chaque pas de temps t, le réseau reçoit une entrée x(t) et produit une sortie y(t), tout en mettant à jour son état caché h(t) basé sur l'entrée actuelle et l'état précédent h(t-1). Cette récurrence permet de capturer les dépendances temporelles.

Applications révolutionnaires :

- **Traitement du langage naturel :** Traduction automatique, génération de texte, analyse de sentiment
- **Analyse de séries temporelles :** Prédiction boursière, météorologie, monitoring IoT
- **Reconnaissance vocale :** Conversion parole-texte, assistants vocaux
- **Vision séquentielle :** Analyse vidéo, tracking d'objets, description automatique d'images

Avantage clé : Capacité à traiter des séquences de longueur variable et à maintenir un contexte sur de longues périodes, essentiel pour comprendre la structure temporelle des données.

Défi majeur : Les RNN traditionnels souffrent du problème du gradient vanishing, limitant leur capacité à apprendre des dépendances à long terme. C'est précisément ce défi qui a motivé le développement des LSTM.

Problème du Gradient Vanishing dans les RNN

Le **gradient vanishing** constitue l'obstacle principal des RNN traditionnels, limitant drastiquement leur capacité d'apprentissage sur de longues séquences.

Mécanisme du problème :

Lors de la rétropropagation à travers le temps (BPTT), le gradient doit traverser de nombreuses étapes temporelles. À chaque étape, il est multiplié par la matrice de poids et la dérivée de la fonction d'activation. Si ces valeurs sont inférieures à 1, le gradient diminue exponentiellement.

Illustration mathématique :

Pour une séquence de longueur T, le gradient au temps t=1 est proportionnel à :

$$[\prod_{k=2}^{T} \frac{\partial h^{(k)}}{\partial h^{(k-1)}} = \prod_{k=2}^{T} W \cdot \sigma'(z^{(k-1)})]$$

Problème du Gradient Vanishing dans les RNN

Conséquences pratiques :

- **Oubli rapide :** Les informations au début de la séquence sont "oubliées"
- **Apprentissage limité :** Seules les dépendances à court terme (5-10 pas) sont apprises
- **Convergence lente :** Les paramètres des premières couches ne s'ajustent presque pas

Exemple concret : Dans la phrase "Le chat, qui était très mignon et jouait dans le jardin depuis ce matin, dort", un RNN classique aura du mal à associer "dort" avec "chat" à cause de la distance.

Solutions existantes avant LSTM :

- Gradient clipping (limitation des gradients)
- Initialisation soignée des poids
- Fonctions d'activation alternatives (ReLU)

Ces solutions partielles ont mené au développement d'architectures spécialisées comme les LSTM.

Architecture LSTM - Vue d'Ensemble

Les **Long Short-Term Memory (LSTM)** networks, introduits par Hochreiter et Schmidhuber en 1997, révolutionnent l'apprentissage des dépendances à long terme grâce à une architecture sophistiquée de "portes" contrôlant le flux d'information.

Innovation architecturale : L'LSTM introduit un **état cellulaire (Cell State) C(t)** distinct de l'état caché h(t). Cet état cellulaire agit comme une "autoroute d'information" permettant aux informations importantes de traverser de nombreuses étapes temporelles sans dégradation.

Architecture LSTM - Vue d'Ensemble

1. État Cellulaire C(t) :

- Mémoire à long terme du réseau
- Flux d'information contrôlé par les portes
- Permet la propagation directe des gradients

2. État Caché h(t) :

- Version filtrée de l'état cellulaire
- Sortie visible du réseau à chaque pas de temps
- Combinaison de mémoire court et long terme

3. Système de Portes :

- **Forget Gate :** Décide quelles informations effacer de l'état cellulaire
- **Input Gate :** Détermine quelles nouvelles informations stocker
- **Output Gate :** Contrôle quelles parties de l'état cellulaire révéler

Avantage révolutionnaire : Cette architecture permet aux LSTM d'apprendre des dépendances sur des centaines, voire des milliers de pas de temps, résolvant efficacement le problème du gradient vanishing.

Porte d'Oubli (Forget Gate) - Mécanisme de Sélection

La **Forget Gate** constitue le premier mécanisme de décision dans une cellule LSTM, déterminant quelles informations de l'état cellulaire précédent doivent être conservées ou effacées.

Rôle fonctionnel :

La porte d'oubli analyse l'état caché précédent h(t-1) et l'entrée actuelle x(t) pour produire un vecteur de décision f(t) compris entre 0 et 1 pour chaque élément de l'état cellulaire.

Équation mathématique :

$$[f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)]$$

Où :

- σ = fonction sigmoïde (sortie entre 0 et 1)
- W_f = matrice de poids de la porte d'oubli
- [h_{t-1}, x_t] = concaténation des vecteurs
- b_f = vecteur de biais

Porte d'Oubli (Forget Gate) - Mécanisme de Sélection

Interprétation des valeurs :

- **f(t) = 0** : Effacer complètement l'information correspondante
- **f(t) = 1** : Conserver totalement l'information
- **0 < f(t) < 1** : Conservation partielle, atténuation progressive

Exemple pratique - Analyse de sentiment :

Dans la phrase "Ce film était excellent au début mais devient ennuyeux", la porte d'oubli peut apprendre à diminuer l'impact de "excellent" quand elle rencontre "mais", permettant au modèle de capturer le changement de sentiment.

Application : Cette porte permet au LSTM de gérer la capacité finie de sa mémoire en effaçant sélectivement les informations devenues non pertinentes.

Porte d'Entrée (Input Gate) - Sélection des Nouvelles Informations

La **Input Gate** détermine quelles nouvelles informations doivent être stockées dans l'état cellulaire, travaillant en tandem avec un candidat de nouvelles valeurs pour mettre à jour sélectivement la mémoire.

Mécanisme en deux étapes :

Étape 1 - Sélection (Input Gate) :

$$[i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)]$$

Cette équation produit un vecteur i(t) indiquant quelles valeurs mettre à jour.

Porte d'Entrée (Input Gate) - Sélection des Nouvelles Informations

Étape 2 - Candidats (Candidate Values) :

$$[\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)]$$

Cette équation génère un vecteur de nouvelles valeurs candidates Ĉ(t).

Combinaison intelligente :

Les deux composants sont multipliés élément par élément :

$$[\text{Nouvelle information} = i_t \odot \tilde{C}_t]$$

Le symbole ⊙ représente la multiplication élément par élément (Hadamard product).

Porte d'Entrée (Input Gate) - Sélection des Nouvelles Informations

Mise à jour de l'état cellulaire :

$$[C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t]$$

Cette équation combine l'information oubliée et la nouvelle information.

Exemple concret - Traduction :

Pour traduire "The cat sits on the mat" → "Le chat s'assoit sur le tapis", l'input gate peut apprendre à encoder de nouvelles informations grammaticales (genre, nombre) tout en préservant le sens principal transporté par l'état cellulaire.

Rôle crucial : Cette porte permet un contrôle fin de la mise à jour mémoire, évitant la corruption de l'information importante par du bruit.

Porte de Sortie (Output Gate) - Contrôle de l'État Caché

La **Output Gate** constitue le mécanisme final de contrôle, déterminant quelles parties de l'état cellulaire seront révélées dans l'état caché et donc transmises aux couches suivantes.

Fonction de filtrage :

La porte de sortie ne modifie pas l'état cellulaire mais contrôle sa visibilité, permettant au LSTM de maintenir des informations internes tout en ne révélant que les éléments pertinents pour la tâche actuelle.

Équations mathématiques :

Calcul de la porte de sortie :

$$[o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)]$$

Génération de l'état caché :

$$[h_t = o_t \odot \tanh(C_t)]$$

Porte de Sortie (Output Gate) - Contrôle de l'État Caché

Analyse des composants :

- **o(t)** : Vecteur de contrôle (0 à 1) déterminant la visibilité
- **tanh(C(t))** : État cellulaire normalisé entre -1 et 1
- **h(t)** : État caché final, combinaison filtrée

Rôle de tanh :

La fonction tanh normalise l'état cellulaire, évitant que des valeurs extrêmes dominent l'état caché et perturbent l'apprentissage.

Exemple pratique - Génération de texte :

Dans un modèle de génération de texte, l'état cellulaire peut contenir des informations sur le style, le contexte, et la grammaire. L'output gate peut apprendre à ne révéler que les informations nécessaires pour prédire le mot suivant, gardant le contexte global en mémoire interne.

Impact sur les performances : Cette séparation entre mémoire interne (C(t)) et sortie visible (h(t)) permet aux LSTM de maintenir des représentations riches tout en fournissant des sorties focalisées.

Flux d'Information dans les LSTM - Vision Intégrée

Le **flux d'information** dans un LSTM suit un processus orchestré où chaque porte contribue à une gestion sophistiquée de la mémoire, permettant l'apprentissage de dépendances complexes sur de longues séquences.

Séquence d'opérations à chaque pas de temps :

1. Analyse contextuelle :

Toutes les portes analysent simultanément l'entrée actuelle x(t) et l'état caché précédent h(t-1) pour prendre leurs décisions respectives.

2. Oubli sélectif :

$[f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)]$

Détermine quelles informations de C(t-1) conserver.

3. Acquisition d'information :

$[i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)]$

$[\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)]$

Sélectionne et génère de nouvelles informations à stocker.

Flux d'Information dans les LSTM - Vision Intégrée

4. Mise à jour de la mémoire :

$[C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t]$

Combine oubli et acquisition pour mettre à jour l'état cellulaire.

5. Génération de sortie :

$[o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)]$

$[h_t = o_t \odot \tanh(C_t)]$

Filtre l'état cellulaire pour produire l'état caché.

Analogie : Comme un flux de données avec des écluses : chaque porte agit comme une écluse contrôlant le débit et la direction de l'information, permettant une navigation précise dans l'espace informationnel temporel.

Backpropagation Through Time (BPTT) - Algorithme d'Apprentissage

La **Backpropagation Through Time (BPTT)** adapte l'algorithme de rétropropagation classique aux architectures récurrentes, permettant l'apprentissage des paramètres à travers les dépendances temporelles.

Principe fondamental :

Le réseau récurrent est "déroulé" dans le temps, créant une architecture feedforward équivalente où chaque pas de temps correspond à une couche. Les gradients sont ensuite calculés en remontant cette chaîne temporelle.

Étapes de l'algorithme BPTT :

1. Forward Pass :

Calcul séquentiel des états cachés et cellulaires pour toute la séquence :

$[h_0, C_0 \rightarrow h_1, C_1 \rightarrow \ldots \rightarrow h_T, C_T]$

Backpropagation Through Time (BPTT) - Algorithme d'Apprentissage

2. Calcul de la perte :
Évaluation de la fonction de perte sur la séquence complète ou seulement sur les sorties finales selon la tâche.
3. Backward Pass :
Calcul des gradients en remontant le temps :
$[\frac{\partial L}{\partial h_T} \rightarrow \frac{\partial L}{\partial h_{T-1}} \rightarrow \ldots \rightarrow \frac{\partial L}{\partial h_0}]$
Avantage des LSTM :
Grâce à l'état cellulaire et aux portes, les gradients peuvent "sauter" par-dessus de nombreux pas de temps sans dégradation significative, résolvant le problème du gradient vanishing.
Complexité computationnelle :

- **Temps :** O(T × M) où T = longueur séquence, M = taille du modèle
- **Mémoire :** O(T × M) pour stocker tous les états intermédiaires

Optimisation pratique : Truncated BPTT limite la rétropropagation à K pas récents pour réduire la complexité mémoire.

Techniques d'Optimisation pour LSTM

L'entraînement efficace des LSTM nécessite des techniques d'optimisation spécialisées pour gérer la complexité des dépendances temporelles et la haute dimensionnalité des paramètres.

Algorithmes d'optimisation avancés :

Adam Optimizer (recommandé) :

- Adapte le learning rate pour chaque paramètre individuellement
- Combine momentum et RMSprop pour une convergence stable
- Learning rate typique : 0.001 à 0.01
- Particulièrement efficace pour les séquences de longueur variable

RMSprop :

- Normalise les gradients par leur moyenne quadratique mobile
- Évite les oscillations dans les directions de forte courbure
- Alternative robuste à Adam pour certaines architectures

Techniques d'Optimisation pour LSTM

Gradient Clipping :

$$[\text{Si } ||\nabla|| > \text{seuil, alors } \nabla \leftarrow \frac{\nabla \cdot \text{seuil}}{||\nabla||}]$$

- Limite l'ampleur des gradients pour éviter l'explosion
- Seuil typique : 1.0 à 5.0
- Essentiel pour stabiliser l'entraînement des LSTM

Stratégies d'apprentissage :

Learning Rate Scheduling :

- **Decay exponentiel :** LR × 0.95 tous les N epochs
- **Cosine annealing :** Réduction cyclique du learning rate
- **ReduceLROnPlateau :** Réduction automatique lors de stagnation

Batch Size dynamique :

- Commencer avec de petits batches (32-64) pour stabilité
- Augmenter progressivement jusqu'à 128-256 pour efficacité
- Équilibrer vitesse d'entraînement et qualité des gradients

Stratégies de Régularisation pour LSTM

Les LSTM, avec leur architecture complexe et nombreux paramètres, nécessitent des techniques de régularisation sophistiquées pour éviter le surapprentissage et améliorer la généralisation.

Dropout spécialisé pour RNN :

Dropout standard :

- Application sur les connexions feedforward (x(t) → gates)
- Taux typique : 0.2 à 0.5
- Évite de perturber les connexions récurrentes critiques

Recurrent Dropout :

- Application sur les connexions récurrentes (h(t-1) → h(t))
- Taux plus faible : 0.1 à 0.3
- Préserve la continuité temporelle

Variational Dropout :

- Même masque de dropout maintenu sur toute la séquence
- Évite les changements brutaux de capacité du réseau
- Particulièrement efficace pour les longues séquences

Stratégies de Régularisation pour LSTM

Techniques de régularisation avancées :

Weight Decay (L2 regularization) :

$$[L_{total} = L_{data} + \lambda \sum_i w_i^2]$$

- λ typique : 10^{-4} à 10^{-2}
- Appliqué principalement aux matrices de poids des portes

Early Stopping :

- Surveillance de la perte de validation
- Patience : 5-15 époques selon la taille du dataset
- Sauvegarde du meilleur modèle

Batch Normalization :

- Normalisation des activations des portes
- Accélère la convergence et régularise implicitement
- Adaptation spéciale pour les RNN (Layer Normalization)

Gestion des Séquences Variables et Optimisations

Le traitement efficace de séquences de longueurs variables constitue un défi majeur dans l'entraînement des LSTM, nécessitant des stratégies spécialisées pour optimiser les performances computationnelles et mémorielles.

Stratégies de gestion des séquences :

Padding et Masking :

- **Padding :** Compléter les séquences courtes avec des zéros
- **Masking :** Ignorer les positions paddées dans le calcul de la perte
- **Position :** Padding en fin de séquence (post-padding) généralement préférable

Bucketing (Groupement par taille) :

- Regrouper les séquences de longueurs similaires dans les mêmes batches
- Réduction significative du padding nécessaire
- Amélioration de 20-40% des performances d'entraînement

Dynamic RNN :

- Arrêt automatique du traitement à la vraie longueur de chaque séquence
- Économie de calculs pour les séquences courtes
- Implémentation native dans TensorFlow/PyTorch

Gestion des Séquences Variables et Optimisations

Optimisations mémorielles :

Truncated BPTT :

- Limitation de la rétropropagation à K pas temporels récents
- K typique : 35-50 pour l'équilibre performance/mémoire
- Maintien de l'état caché pour la continuité

Gradient Checkpointing :

- Sauvegarde sélective des activations intermédiaires
- Trade-off temps de calcul vs mémoire
- Permet l'entraînement de séquences plus longues

Mixed Precision Training :

- Utilisation de float16 pour les forward/backward passes
- Conservation de float32 pour les mises à jour de paramètres
- Réduction de 50% de l'usage mémoire

Variantes et Extensions des LSTM

L'architecture LSTM originale a inspiré de nombreuses variantes et améliorations, chacune ciblant des limitations spécifiques ou des domaines d'application particuliers.

GRU (Gated Recurrent Unit) :

- **Simplification :** Fusion des portes forget et input en une "update gate"
- **Réduction :** Deux portes au lieu de trois (update gate + reset gate)
- **Avantages :** Moins de paramètres, entraînement plus rapide
- **Performance :** Comparable aux LSTM sur de nombreuses tâches
- **Usage :** Préférable quand les ressources computationnelles sont limitées

Bidirectional LSTM :

- **Principe :** Deux LSTM parallèles, un vers l'avant, un vers l'arrière
- **Concaténation :** Les sorties sont combinées à chaque pas de temps
- **Avantage :** Accès au contexte futur et passé simultanément
- **Applications :** Traduction, étiquetage morpho-syntaxique, analyse de sentiment
- **Limitation :** Impossibilité de traitement en temps réel

Variantes et Extensions des LSTM

Peephole LSTM :

- **Modification :** Les portes accèdent directement à l'état cellulaire
- **Équations enrichies :** Addition de termes $W_c \times C(t-1)$ dans les calculs de portes
- **Bénéfice :** Meilleur contrôle basé sur le contenu de la mémoire
- **Usage :** Tâches nécessitant un timing précis

LSTM avec Attention :

- **Mécanisme :** Pondération dynamique des états cachés précédents
- **Application :** Séquence-à-séquence avec focus adaptatif
- **Impact :** Précurseur des architectures Transformer
- **Révolution :** A mené au développement de l'attention auto-supervisée

Bayesian Network

Un **réseau bayésien** est un modèle graphique probabiliste défini par :

Composants essentiels :

- **Graphe orienté acyclique (DAG)** $G = (V, E)$
- **Variables aléatoires** $X = \{X_1, X_2, ..., X_n\}$ représentées par les nœuds V
- **Dépendances conditionnelles** représentées par les arcs E
- **Paramètres** θ : tables de probabilité conditionnelle (CPT)

Propriétés fondamentales :

- Chaque nœud correspond à une variable aléatoire
- Chaque arc ($X_i \rightarrow X_j$) indique que X_i influence directement X_j
- L'absence d'arc encode une hypothèse d'indépendance conditionnelle
- La structure capture les relations causales ou statistiques

Notation :

- Parents(X_i) : ensemble des parents directs de X_i
- Descendants(X_i) : tous les nœuds atteignables depuis X_i
- Ancêtres(X_i) : tous les nœuds dont X_i est descendant

Loi de Probabilité Jointe

Théorème de factorisation fondamental :

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Parents(X_i))$$

Exemple concret :

Pour un réseau : Météo → Arrosage → Gazon Mouillé ← Pluie

$$P(M, A, G, P) = P(M) \cdot P(A|M) \cdot P(P) \cdot P(G|A, P)$$

Avantages de la factorisation :

- **Réduction exponentielle** du nombre de paramètres
- Distribution jointe générale : $O(2^n)$ paramètres
- Avec factorisation : $O(n \times 2^k)$ où $k = \max(|Parents(X_i)|)$

Conditions de validité :

La factorisation est valide si et seulement si toutes les indépendances conditionnelles encodées par le graphe sont respectées dans la distribution.

Applications :

- Calcul efficace de probabilités marginales
- Inférence conditionnelle
- Simulation de données

Propriété de Markov Globale

Énoncé : Chaque variable est conditionnellement indépendante de tous ses non-descendants, étant donnés ses parents.

Formellement :

$$X_i \perp NonDescendants(X_i) \mid Parents(X_i)$$

D-séparation - Critères graphiques :

1. **Configuration en chaîne (A → B → C) :**
   - A et C sont indépendants sachant B : A ⊥ C | B
2. **Configuration en fourche (A ← B → C) :**
   - A et C sont indépendants sachant B : A ⊥ C | B
3. **Configuration en V (A → B ← C) - Collider :**
   - A et C sont indépendants marginalement : A ⊥ C
   - Mais dépendants sachant B : A ⫫ C | B

Blanket de Markov :

Ensemble minimal MB(X) = Parents(X) ∪ Enfants(X) ∪ CoParents(X)
tel que X ⊥ Reste | MB(X)

Inférence - Définition et Complexité

Définition :
L'inférence consiste à calculer $P(X_i \mid Evidence)$ où $Evidence = \{X_j = x_j\}$
Types de requêtes :

- **Probabilité marginale** : $P(X = x)$
- **Probabilité conditionnelle** : $P(X = x \mid E = e)$
- **MAP (Maximum A Posteriori)** : $\text{argmax } P(X \mid E)$
- **MPE (Most Probable Explanation)** : $\text{argmax } P(X, E)$

Complexité théorique :

- **Inférence exacte** : NP-difficile dans le cas général
- **Complexité** : exponentielle en la largeur d'arbre (treewidth) du graphe
- **Cas particuliers** : polynomial pour les arbres, poly-arbres

Facteurs d'influence :

- Structure du réseau (dense vs sparse)
- Taille du domaine des variables
- Nombre de variables observées
- Type de requête (marginale vs MAP)

Inférence Exacte - Méthodes

1. Élimination de Variables :

```
Algorithme VE(Variables, Evidence, Query):
1. Instantier les variables observées
2. Pour chaque variable à éliminer Z :
   a. Collecter tous les facteurs contenant Z
   b. Multiplier ces facteurs
   c. Sommer sur Z pour l'éliminer
3. Multiplier les facteurs restants
4. Normaliser pour obtenir P(Query|Evidence)
```

Complexité : $O(n \times d^{w+1})$ où W = largeur d'arbre, d = taille domaine

2. Propagation de Croyance (Junction Tree) :

- **Moralisation** : ajouter arcs entre co-parents
- **Triangulation** : éliminer cycles de longueur > 3
- **Construction arbre de cliques** maximalités
- **Propagation de messages** entre cliques

3. Cas Efficaces :

- **Arbres** : inférence en O(n)
- **Poly-arbres** : propagation locale
- **Faible largeur d'arbre** : algorithmes polynomiaux

Inférence Approchée - Techniques

1. Méthodes de Monte Carlo :

Échantillonnage Direct :

- Générer échantillons selon $P(X_1,...,X_n)$
- Estimer P(Query|Evidence) par comptage

MCMC (Markov Chain Monte Carlo) :

- **Gibbs Sampling** : échantillonnage cyclique

```
Pour chaque variable Xᵢ :
  Échantillonner Xᵢ ~ P(Xᵢ | X₋ᵢ, Evidence)
```

- **Metropolis-Hastings** : acceptation/rejet

2. Méthodes Variationnelles :

- **Approximation** par distribution plus simple q(x)
- **Minimisation** de D_KL(q||p)
- **Mean Field** : $q(x) = \prod_i q_i(x_i)$

3. Critères de Choix :

- Précision vs temps de calcul
- Taille du réseau et structure
- Ressources computationnelles disponibles

Apprentissage Automatique - Paramètres

Contexte : Structure du réseau connue, données d'entraînement disponibles D = {$x^{(1)}$, $x^{(2)}$, ..., $x^{(N)}$}

1. Maximum de Vraisemblance (ML) :

$$\hat{\theta} = \arg\max_{\theta} \prod_{n=1}^{N} P(x^{(n)}|\theta) = \arg\max_{\theta} \sum_{n=1}^{N} \log P(x^{(n)}|\theta)$$

Avantages : Simple, asymptotiquement optimal

Inconvénients : Surapprentissage avec peu de données

2. Estimation Bayésienne :

- **Prior** : P(θ) (ex: Dirichlet pour variables discrètes)
- **Posterior** : P(θ|D) ∝ P(D|θ)P(θ)
- **MAP** : $\hat{\theta}$ = argmax P(θ|D)

Apprentissage Automatique - Paramètres

3. Algorithme EM (Variables Latentes) :

```
E-step: Q(θ|θ^(t)) = E[log P(X,Z|θ) | X,θ^(t)]
M-step: θ^(t+1) = argmax Q(θ|θ^(t))
```

4. Cas Particuliers :

- Variables discrètes : comptage + lissage de Laplace
- Variables continues : estimation paramètres gaussiens

Apprentissage Automatique - Structure

Problème : Apprendre simultanément structure G et paramètres θ
Complexité : Super-exponentielle en nombre de variables

- Nombre de DAG sur n variables : O(n! × 2^(n(n-1)/2))

1. Approches Score-Based :
Scores populaires :

- **BIC** : log P(D|G,θ̂) - (k/2)log(N)
- **AIC** : log P(D|G,θ̂) - k
- **MDL** : -log P(D|G,θ̂) + (k/2)log(N)

où k = nombre de paramètres
Algorithmes de recherche :

- **Hill-climbing** : modifications locales
- **Recherche tabou** : éviter cycles
- **Algorithmes génétiques** : population de solutions

2. Approches Constraint-Based :

- Tests d'indépendance conditionnelle
- **Algorithme PC** : squelette puis orientation
- Construction incrémentale du graphe

3. Approches Hybrides : Score + contraintes

Variantes - Réseau Bayésien Dynamique

Définition : Extension temporelle des réseaux bayésiens pour modéliser des processus séquentiels.
Structure :

- **Variables** : X_t = {X_1^t, X_2^t, ..., X_n^t} à chaque instant t
- **Dépendances temporelles** : X_t dépend de X_{t-1}
- **Transition** : P(X_t | X_{t-1}) stationnaire

Formulation :

$$P(X_{0:T}) = P(X_0) \prod_{t=1}^{T} P(X_t|X_{t-1})$$

Modèles particuliers :

- **HMM** : une seule variable latente
- **Filtre de Kalman** : variables continues, dynamique linéaire

Variantes - Réseau Bayésien Dynamique

Inférence temporelle :

- **Filtrage** : P(X_t | observations$_0$:t)
- **Lissage** : P(X_t | observations$_0$:T)
- **Prédiction** : P(X_{t+k} | observations$_0$:t)

Applications :

- Reconnaissance vocale, traitement signal
- Finance quantitative, épidémiologie
- Robotique, véhicules autonomes

Variantes - Classifieur Bayésien Naïf

Hypothèse forte : Indépendance conditionnelle des attributs étant donnée la classe.
Structure : C → X_1, C → X_2, ..., C → X_n (étoile)
Modèle :

$$P(C, X_1, \ldots, X_n) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

Classification :

$$\hat{c} = \arg\max_c P(C = c) \prod_{i=1}^{n} P(X_i|C = c)$$

Variantes :

- **Gaussien** : attributs continus ~ N(μc, σc²)
- **Multinomial** : comptage de mots (NLP)
- **Bernoulli** : attributs binaires

Variantes - Classifieur Bayésien Naïf

Paradoxe :

- Hypothèse souvent **violée** en pratique
- Performance **surprenamment bonne** dans de nombreux domaines
- Explication : décision optimale ne nécessite qu'ordre correct des probabilités

Applications :

- Classification de texte, filtrage spam
- Diagnostic médical
- Reconnaissance de formes simples

Variantes - Diagramme Causal

Objectif : Distinguer corrélation et causation, permettre inférence causale.
Extension des réseaux bayésiens :

- Arcs représentent des **relations causales directes**
- Permet d'analyser l'effet d'**interventions**

Do-Calculus de Pearl :
Trois règles pour transformer P(Y|do(X)) :

1. **Insertion/suppression observations :**
   $P(Y|do(X), Z, W) = P(Y|do(X), W)$ si $Y \perp Z \mid X,W$ dans $G_{\bar{X}}$
2. **Action/observation exchange :**
   $P(Y|do(X), do(Z), W) = P(Y|do(X), Z, W)$ si $Y \perp Z \mid X,W$ dans $G_{\bar{X},\bar{Z}}$
3. **Insertion/suppression actions :**
   $P(Y|do(X), do(Z), W) = P(Y|do(Z), W)$ si $Y \perp Z \mid X,W$ dans $G_{\bar{X},Z(W)}$

Variantes - Diagramme Causal

Applications :

- **Épidémiologie** : effet de traitements
- **Économétrie** : politiques publiques
- **Sciences sociales** : relations causales complexes

AutoEncoder

Un **autoencodeur** est un réseau de neurones non supervisé conçu pour apprendre une représentation comprimée (encodage) des données d'entrée.

Objectif principal :

Apprendre une fonction identité $f(x) \approx x$ sous contrainte de dimensionalité réduite.

Composants architecturaux :

- **Encodeur** $\varphi : X \to Z$ (compression)
- **Espace latent** Z de dimension réduite
- **Décodeur** $\psi : Z \to X$ (reconstruction)

Intuition :

Forcer le réseau à passer par un "goulot d'étranglement" pour capturer uniquement les caractéristiques essentielles des données.

Différence avec PCA :

- PCA : transformation linéaire optimale
- AutoEncoder : transformation non-linéaire apprise

Applications :

- Réduction de dimensionnalité
- Détection d'anomalies, Débruitage d'images, Pré-entraînement de réseaux profonds

AutoEncoder - Architecture

Structure générale :

```
Input(d) → Hidden(h₁) → ... → Latent(k) → ... → Hidden(h₁) → Output(d)
        ↳──────── Encodeur ────────↲    ↳──────── Décodeur ────────↲
```

Encodeur $\varphi : \mathbb{R}^d \to \mathbb{R}^k$:

- Séquence de couches avec dimensions décroissantes
- Fonctions d'activation : ReLU, tanh, sigmoid
- $z = \varphi(x) = \sigma(W_2\sigma(W_1x + b_1) + b_2)$

Espace latent :

- Dimension k << d (contrainte de compression)
- Représentation dense des caractéristiques principales

Décodeur $\psi : \mathbb{R}^k \to \mathbb{R}^d$:

- Architecture "miroir" de l'encodeur
- Dimensions croissantes
- $\hat{x} = \psi(z) = \sigma'(W_4\sigma'(W_3z + b_3) + b_4)$

AutoEncoder - Architecture

Symétrie :

Souvent $W_4 = W_1^T$, $W_3 = W_2^T$ (poids liés) pour réduire paramètres.

Activation finale :

- Sigmoid pour données [0,1]
- Linéaire pour données continues
- Softmax pour données catégorielles

AutoEncoder - Formalisation Générale

Formulation mathématique :

$$z = f_\theta(x) \text{ (encodeur)}$$
$$\hat{x} = g_\varphi(z) \text{ (dé codeur)}$$
$$\hat{x} = (g_\varphi \circ f_\theta)(x)$$

Fonction de perte :

$$L(x, \hat{x}) = ||x - \hat{x}||^2 + \lambda R(\theta, \varphi)$$

où R(θ, φ) est un terme de régularisation.

Types de régularisation :

1. **Sparse AutoEncoder :**

$$R = \lambda \sum_i |h_i| \$\$(ré\ gularisation L1 sur activations)$$

$$R = \lambda || \frac{\partial h}{\partial x} ||^2_F \$\$(pé\ nalise sensibilité\ aux variations)$$

Entrée corrompue : $\tilde{x}$ = x + ε, objectif : reconstruire x original

AutoEncoder - Formalisation Générale

Optimisation :

- Algorithme : descente de gradient (SGD, Adam)
- Backpropagation standard
- Mini-batches pour efficacité

Word2Vec Model

Objectif : Apprendre des représentations vectorielles de mots capturant la sémantique.
Intuition : "You shall know a word by the company it keeps" (Firth, 1957)
Deux architectures principales :
1. Skip-gram : Prédire le contexte à partir du mot central

```
Input: mot central wi → Output: mots contexte {wi-c, ..., wi+c}
```

$$P(w_o | w_i) = \frac{\exp(v'_{w_o}^T v_{w_i})}{\sum_{w=1}^{|V|} \exp(v'_w^T v_{w_i})}$$

2. CBOW (Continuous Bag-of-Words) : Prédire mot central depuis contexte

```
Input: contexte {wi-c, ..., wi+c} → Output: mot central wi
```

Optimisations computationnelles :

- **Hierarchical Softmax** : arbre binaire, complexité O(log|V|)
- **Negative Sampling** : échantillonner mots négatifs

$$L = \log \sigma(v'_{w_O}^T v_{w_I}) + \sum_{i=1}^k E_{w_i \sim P_n(w)}[\log \sigma(-v'_{w_i}^T v_{w_I})]$$

Propriétés remarquables :

- Relations linéaires : roi - homme + femme ≈ reine
- Clustering sémantique dans l'espace vectoriel

GloVe Model

Global Vectors for Word Representation

Motivation : Combiner avantages des méthodes globales (factorisation matricielle) et locales (fenêtre contextuelle).

Matrice de co-occurrence :

- X_{ij} = nombre d'occurrences du mot j dans le contexte du mot i
- Statistiques globales du corpus

Observation clé :

Les ratios de co-occurrence révèlent des relations sémantiques :

$$\frac{P_{ik}}{P_{jk}} = \frac{X_{ik}}{X_{jk}}$$

Fonction objectif :

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Fonction de pondération :

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{si } x < x_{max} \\ 1 & \text{sinon} \end{cases}$$

avec α = 3/4, x_{max} = 100 typiquement.

Avantages vs Word2Vec :

- Utilisation efficace des statistiques globales
- Training plus rapide sur gros corpus
- Performance comparable sur tâches d'analogie

Reinforcement Learning

Paradigme d'apprentissage : Un agent apprend à prendre des décisions optimales en interagissant avec un environnement.

Acteurs principaux :

- **Agent** : système qui prend les décisions
- **Environnement** : monde dans lequel évolue l'agent
- **Actions** : choix possibles de l'agent
- **États** : situations dans lesquelles se trouve l'agent
- **Récompenses** : feedback de l'environnement

Boucle d'interaction :

```
État St → Action At → Récompense Rt+1 → État St+1 → ...
```

Reinforcement Learning

Différence avec autres paradigmes :

- **Supervisé** : apprend depuis exemples étiquetés
- **Non-supervisé** : trouve structure dans données
- **Reinforcement** : apprend par essai-erreur avec feedback différé

Caractéristiques clés :

- **Apprentissage séquentiel** : décisions influencent le futur
- **Trade-off exploration/exploitation**
- **Récompense différée** : actions présentes affectent récompenses futures
- **Apprentissage en ligne** : adaptation continue

Formulation d'un Problème de RL

Processus de Décision Markovien (MDP) :
Un MDP est défini par le tuple ⟨S, A, P, R, γ⟩ :

- **S** : ensemble fini d'états
- **A** : ensemble fini d'actions
- **P** : fonction de transition P(s'|s,a)
- **R** : fonction de récompense R(s,a) ou R(s,a,s')
- **γ** ∈ [0,1] : facteur d'escompte

Propriété de Markov :

$$P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \dots) = P(S_{t+1}|S_t, A_t)$$

Politique π :

- **Déterministe** : π(s) = a
- **Stochastique** : π(a|s) = probabilité d'action a dans état s

Return (récompense cumulée) :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Objectif : Trouver politique optimale π* maximisant E[G_t]
Extension : POMDP
Partially Observable MDP : agent n'observe que partiellement l'état

Cas d'Usage du Reinforcement Learning

1. Jeux et Divertissement :

- **AlphaGo/AlphaZero** : maîtrise du Go, échecs, shogi
- **OpenAI Five** : Dota 2 professionnel
- **Atari Games** : DQN sur jeux classiques
- **StarCraft II** : AlphaStar

2. Robotique :

- **Contrôle de bras robotiques** : manipulation d'objets
- **Locomotion** : apprentissage de la marche
- **Navigation autonome** : évitement d'obstacles
- **Assembly tasks** : assemblage précis

3. Finance :

- **Trading algorithmique** : optimisation de portefeuille
- **Market making** : provision de liquidité
- **Risk management** : gestion du risque
- **Pricing** : tarification dynamique

Cas d'Usage du Reinforcement Learning

4. Systèmes et Réseaux :

- **Load balancing** : répartition de charge
- **Routing** : optimisation de chemins réseau
- **Resource allocation** : allocation de ressources cloud
- **Traffic control** : gestion du trafic urbain

5. Autres Applications :

- **Personnalisation** : recommandations adaptatives
- **Publicité** : enchères temps réel (RTB)
- **Santé** : optimisation de traitements

Caractéristiques de l'Apprentissage par Renforcement

1. Apprentissage par Essai-Erreur :

- Agent teste différentes actions
- Apprend des conséquences de ses choix
- Amélioration progressive par expérience

2. Feedback Différé :

- Récompenses peuvent arriver avec retard
- Actions présentes influencent récompenses futures
- Nécessité d'attribution de crédit temporel

3. Trade-off Exploration vs Exploitation :

- **Exploration** : tester nouvelles actions pour découvrir
- **Exploitation** : utiliser connaissances actuelles
- **Dilemme** : optimiser court terme vs apprentissage long terme

Caractéristiques de l'Apprentissage par Renforcement

4. Apprentissage Séquentiel :

- Décisions interdépendantes dans le temps
- État futur dépend des actions passées
- Planification à horizon temporel

5. Apprentissage en Ligne :

- Adaptation continue à l'environnement
- Pas besoin de dataset fixe pré-constitué
- Capacité d'adaptation aux changements

6. Optimisation Multi-Objectif :

- Maximiser récompense cumulée
- Minimiser risque/variance
- Respecter contraintes temporelles

Types d'Apprentissage par Renforcement

1. Classification par Connaissance du Modèle :

Model-Free :

- Agent ne connaît pas P(s'|s,a) ni R(s,a)
- Apprend directement depuis expérience
- **Exemples** : Q-Learning, SARSA, Policy Gradient

Model-Based :

- Agent apprend/utilise modèle de l'environnement
- Planification possible avec modèle appris
- **Exemples** : Dyna-Q, MCTS, Model-Predictive Control

Types d'Apprentissage par Renforcement

2. Classification par Type d'Apprentissage :

Value-Based :

- Apprend fonction de valeur V(s) ou Q(s,a)
- Politique dérivée de la fonction de valeur
- **Exemples** : Q-Learning, DQN, Double DQN

Policy-Based :

- Apprend directement la politique $\pi(a|s)$
- Optimisation par gradient de politique
- **Exemples** : REINFORCE, PPO, TRPO

Actor-Critic :

- Combine value-based et policy-based
- Actor : politique, Critic : fonction de valeur
- **Exemples** : A3C, SAC, TD3

Types d'Apprentissage par Renforcement

3. Classification par Données :

On-Policy :

- Utilise données de la politique courante
- **Exemples** : SARSA, PPO, A3C

Off-Policy :

- Peut utiliser données d'autres politiques
- **Exemples** : Q-Learning, DQN, SAC

Défis de l'Apprentissage par Renforcement

1. Sample Efficiency :

- **Problème** : besoin de millions d'échantillons
- **Causes** : exploration inefficace, credit assignment
- **Solutions** : experience replay, prioritized sampling

2. Exploration vs Exploitation :

- **Problème** : équilibre difficile à maintenir
- **Stratégies** : ε-greedy, UCB, Thompson Sampling
- **Exploration sophistiquée** : curiosity-driven, count-based

3. Non-Stationnarité :

- **Problème** : distribution des données change pendant apprentissage
- **Causes** : politique évolue, environnement adaptatif
- **Solutions** : target networks, experience replay

Défis de l'Apprentissage par Renforcement

4. Dimensionnalité :

- **Problème** : espaces d'états/actions très grands
- **Solutions** : approximation fonctionnelle, hierarchical RL

5. Sparse Rewards :

- **Problème** : récompenses rares ou différées
- **Solutions** : reward shaping, intrinsic motivation, HER

6. Stabilité d'Entraînement :

- **Problème** : convergence non garantie
- **Causes** : approximation, bootstrapping
- **Solutions** : clipping, regularization, careful hypertuning

Avantages et Inconvénients du RL

Avantages :

1. Adaptabilité :

- Apprentissage continu et adaptation
- Pas besoin d'expertise domaine a priori
- Découverte de stratégies non-intuitives

2. Généralisation :

- Capable de gérer nouveaux scénarios
- Transfer learning possible entre tâches similaires
- Robustesse à certains changements d'environnement

3. Autonomie :

- Apprentissage sans supervision humaine constante
- Optimisation automatique des stratégies
- Découverte de solutions créatives

4. Performance :

- Peut dépasser performance humaine (jeux, robotique)
- Optimisation globale plutôt que locale

Avantages et Inconvénients du RL

Inconvénients :

1. Coût Computationnel :

- Entraînement très long et coûteux
- Besoin de millions d'échantillons
- Infrastructure de calcul importante

2. Stabilité :

- Convergence non garantie
- Sensible aux hyperparamètres
- Variance élevée des résultats

3. Sécurité :

- Exploration peut être dangereuse
- Comportements imprévisibles pendant apprentissage
- Difficile de garantir contraintes de sécurité

GAN - Architecture

Generative Adversarial Networks : Framework d'apprentissage antagoniste entre deux réseaux.
Composants :
1. Générateur G :

- **Entrée** : vecteur de bruit z ~ p(z) (ex: Gaussien)
- **Sortie** : données synthétiques G(z)
- **Objectif** : tromper le discriminateur
- Architecture typique : Deconvolution, BatchNorm, ReLU

2. Discriminateur D :

- **Entrée** : données réelles x ou fausses G(z)
- **Sortie** : probabilité que l'entrée soit réelle D(x) $\in$ [0,1]
- **Objectif** : distinguer vraies vs fausses données
- Architecture typique : Convolution, BatchNorm, LeakyReLU

GAN - Architecture

3. Entraînement Antagoniste :

```
Pour chaque itération :
1. Entraîner D : maximiser détection vraies/fausses données
2. Entraîner G : minimiser détection des données générées
```

Architecture DCGAN (Deep Convolutional GAN) :

- Convolutions/Deconvolutions transposées
- BatchNormalization (sauf couches input/output)
- ReLU pour G, LeakyReLU pour D
- Pas de fully connected layers cachées

GAN - Propriétés Mathématiques

Formulation en Théorie des Jeux :

GAN résout le problème minimax :

$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$

Interprétation :

- **Premier terme** : D maximise probabilité de classifier correctement données réelles
- **Deuxième terme** : D maximise probabilité de détecter données fausses
- **G minimise** : rendre ses données indétectables

Équilibre de Nash Théorique :

- **Solution optimale** : p_g = p_data (générateur reproduit distribution réelle)
- **Discriminateur optimal** : D*(x) = p_data(x)/(p_data(x) + p_g(x))
- **À l'équilibre** : D*(x) = 1/2 partout

GAN - Propriétés Mathématiques

Divergence de Jensen-Shannon :

Le critère GAN minimise JSD entre p_data et p_g :

$JSD(p_{data}||p_g) = \frac{1}{2}KL(p_{data}||M) + \frac{1}{2}KL(p_g||M)$

où M = (p_data + p_g)/2

Propriétés Théoriques :

- Convergence globale vers p_data sous conditions idéales
- En pratique : oscillations, mode collapse, instabilité

Training GAN

Algorithme d'Entraînement Standard :

```
Pour nombre_epochs :
    Pour k étapes :
        # Entraîner Discriminateur
        Échantillonner {x⁽¹⁾,...,x⁽ᵐ⁾} depuis p_data
        Échantillonner {z⁽¹⁾,...,z⁽ᵐ⁾} depuis p_z
        Mettre à jour D par gradient ascendant :
        ∇_θd [1/m Σᵢ log D(x⁽ᵢ⁾) + log(1-D(G(z⁽ᵢ⁾)))]

    # Entraîner Générateur
    Échantillonner {z⁽¹⁾,...,z⁽ᵐ⁾} depuis p_z
    Mettre à jour G par gradient descendant :
    ∇_θg [1/m Σᵢ log(1-D(G(z⁽ᵢ⁾)))]
```

Training GAN

Problèmes d'Entraînement :

1. Mode Collapse :

- G génère échantillons de faible diversité
- Solutions : Unrolled GAN, WGAN, spectral normalization

2. Vanishing Gradients :

- D trop bon → gradients nuls pour G
- Solutions : feature matching, modified losses

3. Instabilité :

- Oscillations, non-convergence
- Solutions : learning rate scheduling, architectural tricks

Améliorations Techniques :

- **Spectral Normalization** : stabilise training
- **Progressive Growing** : entraînement multi-résolution
- **Self-Attention** : capture dépendances long-terme

Évaluation GAN

Défis de l'Évaluation :

- Pas de métrique objective universelle
- Trade-off qualité vs diversité
- Évaluation subjective importante

1. Métriques Quantitatives :

Inception Score (IS) :

$$IS = \exp(E_{x \sim p_g}[D_{KL}(p(y|x)||p(y))])$$

- Mesure qualité et diversité simultanément
- **Limitations** : biais vers classes ImageNet

Fréchet Inception Distance (FID) :

$$FID = ||\mu_r - \mu_g||^2 + Tr(C_r + C_g - 2\sqrt{C_r C_g})$$

- Compare statistiques features Inception
- **Plus bas = meilleur**, corrèle avec qualité humaine

Évaluation GAN

2. Métriques de Précision/Rappel :

- **Précision** : proportion échantillons générés de haute qualité
- **Rappel** : couverture de la distribution réelle
- **Trade-off** : qualité vs diversité

3. Évaluation Humaine :

- **AMT studies** : évaluation par humains
- **Turing test** : distinguer réel vs généré
- **Études perceptuelles** : préférence humaine

4. Autres Métriques :

- **LPIPS** : similarité perceptuelle
- **PPL** : Perceptual Path Length
- **SWD** : Sliced Wasserstein Distance