

Deep learning

Redha Moulla

Juillet 2025

Plan de la formation

- Introduction au deep learning
- Vision par ordinateur
- Traitement du langage naturel
- Large Language Models
- Techniques plus avancées

Introduction au deep learning

Jalons Importants du Deep Learning

Années 1940-1980 : Premières idées

- 1943 : Modèle de neurone artificiel de McCulloch et Pitts.
- 1958 : Perceptron de Rosenblatt, premier modèle entraînable.
- 1986 : Backpropagation popularisée par Rumelhart, Hinton et Williams.

Années 1990-2000 : Bases modernes

- 1995 : SVM et Random Forest dominent le Machine Learning.
- 1998 : LeNet-5 de Yann LeCun, pionnier des CNNs.
- 2006 : Redécouverte des réseaux de neurones profonds grâce à Hinton.

Années 2010-2020 : Révolution du Deep Learning

- 2012 : AlexNet révolutionne la vision par ordinateur.
- 2014 : GANs (Goodfellow) et Deep Q-Networks (DeepMind).
- 2017 : Transformer (Vaswani et al.), révolution dans le NLP.
- 2020 : GPT-3, BERT et diffusion des LLMs.

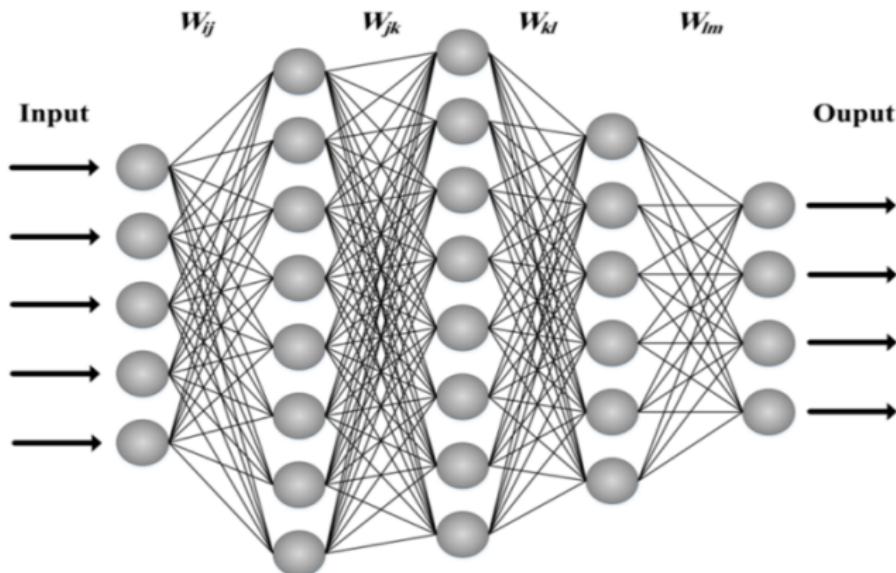
Introduction aux réseaux de neurones

Définition : Les réseaux de neurones sont des modèles computationnels inspirés par le fonctionnement des neurones dans le cerveau humain. Ils sont capables d'apprendre des tâches complexes en modélisant des relations non linéaires entre les entrées et les sorties.

Caractéristiques :

- **Extraction automatique des features** : Capacité d'adaptation et d'extraction des features à partir des données sans programmation explicite.
- **Modélisation non linéaire** : Aptitude à capturer des relations complexes dans les données.
- **Flexibilité** : Applicables à un large éventail de tâches et de typologies de données (images, langage naturel, données graphiques, etc.).

Illustration d'un réseau de neurones classique



Le neurone artificiel

Modèle Mathématique : Chaque neurone artificiel effectue une somme pondérée de ses entrées, ajoute un biais, et passe le résultat à travers une fonction d'activation pour obtenir la sortie.

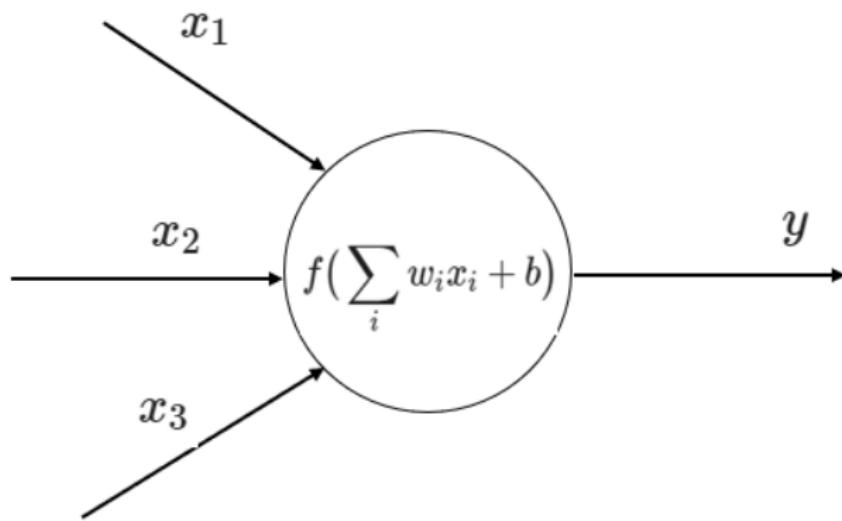
$$z_i = \sum_{j=1}^m w_{ij}x_j + b_i$$

$$a_i = f(z_i)$$

où :

- x_j représente l'entrée du neurone,
- w_{ij} est le poids associé à l'entrée x_j ,
- b_i est le biais du neurone,
- f est la fonction d'activation,
- z_i est le potentiel d'action pré-synaptique,
- a_i est la sortie activée du neurone.

Illustration d'un neurone artificiel



Fonctions d'activation

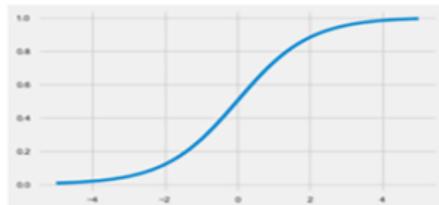
Les fonctions d'activation permettent aux modèles d'apprendre des relations plus complexes en capturant les non-linéarités dans les données.

- **Sigmoïde** : $\sigma(z) = \frac{1}{1+e^{-z}}$, plage de sortie (0, 1), utilisée pour la probabilité dans la classification binaire.
- **Tangente Hyperbolique (tanh)** : $\tanh(z)$, plage de sortie (-1, 1), version centrée et normalisée de la sigmoïde.
- **ReLU (Unité Linéaire Rectifiée)** : $f(z) = \max(0, z)$, non saturante, favorise la convergence rapide et permet d'éviter le problème de disparition des gradients.
- **Leaky ReLU** : $f(z) = \max(\alpha z, z)$, variante de ReLU qui permet un petit gradient lorsque $z < 0$.
- **Softmax** : Utilisée pour la couche de sortie des problèmes de classification multi-classes.

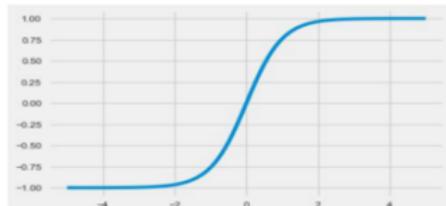
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Illustration des fonctions d'activation

Sigmoïde



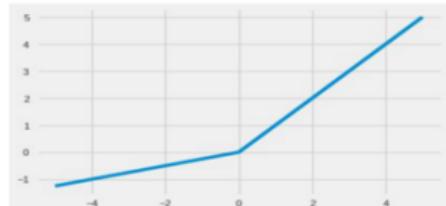
Tanh



ReLU



ReLU paramétrique



Entraînement d'un réseau de neurones artificiel

Principe d'Entraînement : L'entraînement d'un réseau de neurones consiste à ajuster ses poids pour minimiser une fonction de coût qui mesure l'erreur entre les prédictions et les vraies valeurs.

Descente de gradient stochastique (SGD) :

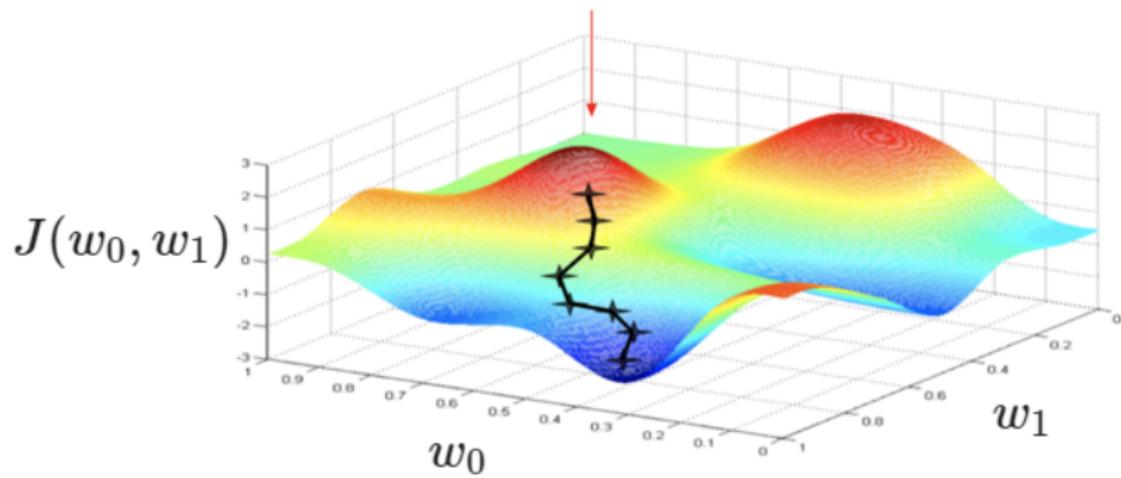
- Méthode d'optimisation utilisée pour mettre à jour les poids du réseau de manière itérative.
- À chaque itération, un sous-ensemble (batch) de données est utilisé pour calculer le gradient de la fonction de coût.
- Les poids sont mis à jour dans la direction opposée du gradient pour réduire l'erreur.

$$w_{new} = w_{old} - \eta \cdot \nabla_w J(w)$$

où :

- w_{old} et w_{new} sont les valeurs des poids avant et après la mise à jour,
- η est le taux d'apprentissage,
- $\nabla_w J(w)$ est le gradient de la fonction de coût par rapport aux poids.

Illustration graphique de la SGD



Entraînement des réseaux de neurones profonds

Dans les réseaux de neurones profonds, l'erreur calculée à la sortie du réseau dépend indirectement des poids des couches cachées. Ce lien indirect rend difficile de savoir comment ajuster ces poids pour réduire l'erreur.

Implications pour l'Entraînement :

- **Propagation de l'erreur** : Sans un mécanisme pour propager l'erreur de la sortie vers les couches antérieures, il est impossible de déterminer l'impact de chaque poids sur l'erreur finale.
- **Complexité de l'ajustement des Poids** : Chaque poids dans les couches cachées affecte l'erreur de sortie de manière complexe, nécessitant une méthode précise pour leur ajustement.



Backpropagation : Objectif et principe

Objectif : Trouver comment ajuster les poids $W^{(n)}$ et les biais $b^{(n)}$ pour minimiser l'erreur de prédiction du réseau.

Ce que nous voulons calculer :

- Les gradients $\frac{\partial J}{\partial W^{(n)}}$ et $\frac{\partial J}{\partial b^{(n)}}$.
- Ces gradients nous permettent de mettre à jour les paramètres avec la descente de gradient :

$$W^{(n)} \leftarrow W^{(n)} - \eta \frac{\partial J}{\partial W^{(n)}}, \quad b^{(n)} \leftarrow b^{(n)} - \eta \frac{\partial J}{\partial b^{(n)}} \quad (1)$$

Problème : Comment calculer ces gradients en partant de la sortie du réseau ?

Définition de l'erreur locale :

$$\delta^{(n)} = \frac{\partial J}{\partial z^{(n)}} \quad (2)$$

Nous devons donc propager cette erreur pour obtenir tous les gradients.

Rétropropagation de l'erreur

$$\delta^{(N)} = \frac{\partial J}{\partial a^{(N)}} \cdot \sigma'(z^{(N)}) \quad (3)$$

2. Rétropropagation de l'erreur :

$$\delta^{(n)} = (W^{(n+1)})^T \delta^{(n+1)} \cdot \sigma'(z^{(n)}) \quad (4)$$

3. Calcul des gradients des poids et biais :

$$\frac{\partial J}{\partial W^{(n)}} = \delta^{(n)} (a^{(n-1)})^T \quad (5)$$

$$\frac{\partial J}{\partial b^{(n)}} = \delta^{(n)} \quad (6)$$

4. Mise à jour des paramètres :

$$W^{(n)} \leftarrow W^{(n)} - \eta \frac{\partial J}{\partial W^{(n)}}, \quad b^{(n)} \leftarrow b^{(n)} - \eta \frac{\partial J}{\partial b^{(n)}} \quad (7)$$

Surapprentissage dans les réseaux de neurones

Le surapprentissage (overfitting) se produit lorsqu'un réseau de neurones apprend trop bien les détails et le bruit des données d'entraînement, au détriment de sa capacité à généraliser sur de nouvelles données.

Le surapprentissage dans les réseaux de neurones peut se produire pour différentes raisons :

- Complexité excessive du modèle
- Manque de diversité dans les données d'entraînement
- Entraînement prolongé

Stratégies pour atténuer le surapprentissage :

- **Régularisation** : Techniques de régularisation telles que L1/L2, Dropout, pour limiter la complexité du modèle.
- **Dropout** : "Eteindre" un ensemble de neurones choisis aléatoirement pendant l'entraînement.
- **Early Stopping** : Arrêt de l'entraînement lorsque la performance sur un ensemble de validation cesse de s'améliorer.
- **Augmentation de Données** : Augmenter la variété des données d'entraînement pour améliorer la robustesse du modèle.

Computer vision

Introduction à la vision par ordinateur

Qu'est-ce que la vision par ordinateur ? La vision par ordinateur est un domaine de l'intelligence artificielle qui vise à permettre aux machines de comprendre et d'interpréter des images et des vidéos, comme le fait le système visuel humain.

Applications :

- Classification d'images
- Détection et suivi d'objets
- Segmentation d'images
- Reconnaissance d'images
- Génération d'images

Défis :

- Variabilité des conditions d'éclairage et de perspective
- Bruit et occlusions dans les images
- Complexité des scènes réelles
- Besoin de grandes quantités de données annotées

Brève histoire des réseaux de neurones convolutifs (CNN)

Principaux jalons historiques

- **1959 - Neurophysiologie de la vision** Découverte des neurones sensibles aux motifs dans le cortex visuel (Hubel Wiesel).
- **1980 - Neocognitron (Fukushima)** Premier modèle de réseau inspiré des mécanismes biologiques de la vision.
- **1989 - Apprentissage supervisé des CNN (LeCun et al.)** Introduction de l'apprentissage des poids avec le backpropagation.
- **1998 - LeNet-5 (LeCun et al.)** Première architecture CNN réussie, appliquée à la reconnaissance de chiffres manuscrits.
- **2012 - AlexNet (Krizhevsky, Sutskever, Hinton)** Victoire au challenge ImageNet, exploitant le calcul sur GPU.
- **2014 - VGGNet et GoogLeNet (Simonyan Zisserman, Szegedy et al.)** Exploration de réseaux plus profonds et introduction d'architectures optimisées.
- **2015 - ResNet (He et al.)** Introduction des connexions résiduelles, permettant d'entraîner des réseaux très profonds.
- **2017+ - Évolution des CNN** Hybridation avec des Transformers (Vision Transformers - ViT) et optimisation des architectures.

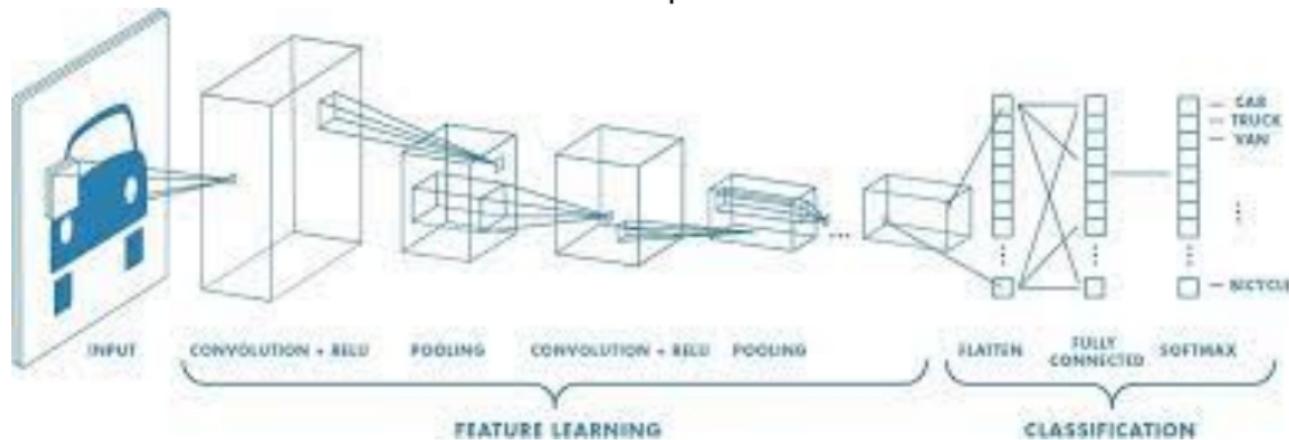
Architecture basique d'un CNN

Les CNNs sont structurés en plusieurs couches qui transforment progressivement l'entrée (image) pour aboutir à une sortie (décision).

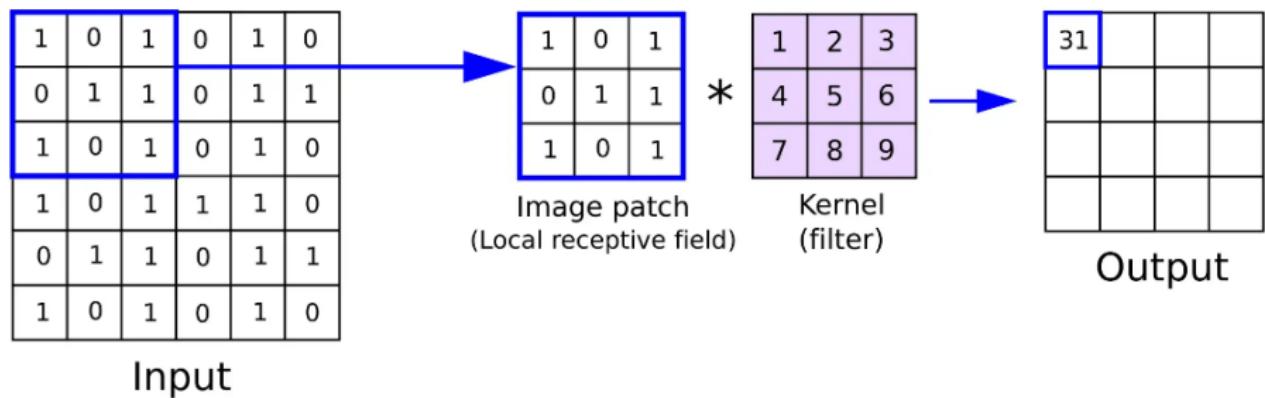
- ① **Couche de convolution** : Applique des filtres pour extraire des caractéristiques de bas niveau comme les bords et les textures.
- ② **Couche de pooling (ou Sous-échantillonnage)** : Réduit la dimensionnalité de chaque carte de caractéristiques tout en préservant les informations les plus importantes.
- ③ **Couche de normalisation** : Normalise les activations de la couche précédente, souvent pour améliorer la convergence.
- ④ **Couches entièrement Connectées (Fully Connected)** : Utilisées en fin de réseau pour classer les images en fonction des caractéristiques extraites.

Architecture d'un CNN

Le fonctionnement d'un CNN peut être visualisé comme un enchaînement de transformations où chaque couche traite et transforme les informations provenant de la couche précédente.



Fonctionnement des filtres d'un CNN



Pourquoi utiliser plusieurs filtres dans un CNN ?

Principe Dans un CNN, chaque couche de convolution applique plusieurs filtres simultanément. Chaque filtre apprend à détecter un motif spécifique (**bords, textures, formes, objets**).

Pourquoi plusieurs filtres ?

- Capturer différentes caractéristiques d'une image (ex. bords horizontaux, diagonaux, textures).
- Obtenir une représentation plus riche et discriminante.
- Augmenter la profondeur de l'apprentissage sans ajouter trop de paramètres.

Comment cela fonctionne ?

- Chaque filtre produit une **feature map** en appliquant une convolution sur l'image d'entrée.
- Toutes les feature maps sont ensuite empilées pour former la sortie de la couche.
- À chaque nouvelle couche, les filtres combinent les informations des couches précédentes.

Fonctionnement du pooling dans les CNN

Pourquoi le pooling ? Le pooling est une opération qui réduit la dimensionnalité des feature maps tout en conservant les informations importantes. Il permet de :

- Réduire le nombre de paramètres et de calculs
- Rendre le réseau plus robuste aux translations et variations locales
- Prévenir le sur-apprentissage (overfitting)

Types de pooling :

- **Max pooling** : conserve la valeur maximale dans chaque région du filtre.
- **Average pooling** : calcule la moyenne des valeurs dans chaque région.

Hyperparamètres influents :

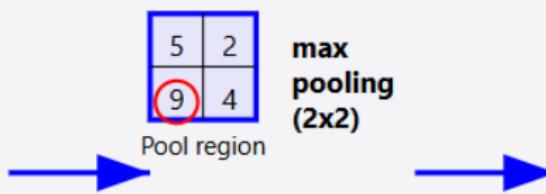
- **Taille du filtre** ($k \times k$) : ex. 2×2 ou 3×3
- **Stride** : contrôle le déplacement du filtre (souvent égal à la taille du filtre)

Impact sur l'architecture CNN : - Réduit la taille des feature maps → moins de mémoire et de calculs - Maintient les caractéristiques les plus importantes - Supprime des détails moins pertinents, favorisant l'invariance spatiale

Fonctionnement du pooling

5	2	7	1	8	3
9	4	6	2	7	5
3	8	2	9	4	1
7	6	5	8	3	2
1	5	9	4	7	6
4	2	8	3	6	1

Input



9	7	8
8	9	4
5	9	7

Output

La normalisation dans les CNN

Pourquoi la normalisation ? La normalisation permet de stabiliser l'entraînement des réseaux de neurones et d'accélérer la convergence en contrôlant la distribution des activations. Elle aide à :

- Réduire le décalage de covariances interne (internal covariate shift).
- Accélérer l'apprentissage en permettant des taux d'apprentissage plus élevés.
- Améliorer la généralisation et réduire l'overfitting.

Types de normalisation :

- **Batch Normalization (BatchNorm)** Normalise les activations par mini-batch pour assurer une distribution stable.
- **Layer Normalization (LayerNorm)** Appliquée sur chaque exemple indépendamment en normalisant sur les neurones d'une couche.
- **Instance Normalization (InstanceNorm)** Similaire à LayerNorm mais appliquée indépendamment sur chaque canal d'une image.
- **Group Normalization (GroupNorm)** Divise les canaux en groupes et applique une normalisation à chaque groupe.

Batch Normalization - Formule : Pour une activation x_i dans un mini-batch :

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Les couches fully connected dans les CNN

Qu'est-ce qu'une couche fully connected ? Une couche fully connected (FC) est une couche où chaque neurone est connecté à tous les neurones de la couche précédente. Elle permet de transformer les caractéristiques extraites en une sortie interprétable (ex. classification).

Fonctionnement :

- Prend en entrée un vecteur aplati (**flatten**) issu des dernières feature maps d'un CNN.
- Applique une transformation linéaire :

$$y = Wx + b$$

où W est la matrice des poids et b le biais.

- Applique une **fonction d'activation** (ex. ReLU, softmax).

Pourquoi utiliser des couches fully connected ?

- Convertir les features extraites en classes ou valeurs de sortie.
- Capturer des combinaisons complexes des caractéristiques apprises.

Limites des CNN profonds

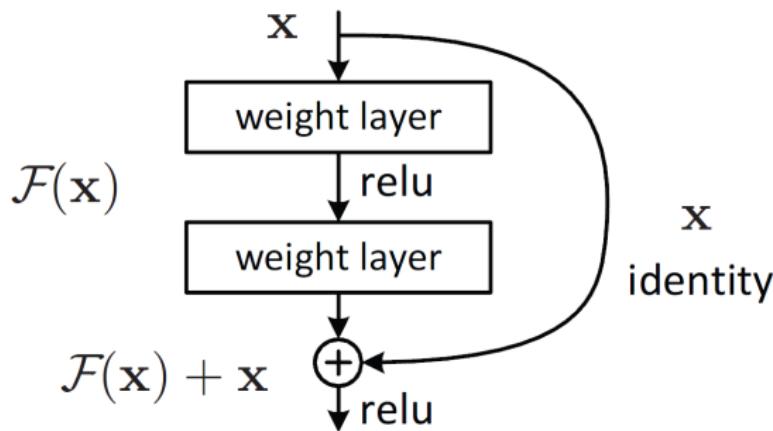
- L'intuition initiale : plus de couches = plus de capacité à modéliser des fonctions complexes.
- En pratique, au-delà d'une certaine profondeur :
 - La précision sur le set de test stagne ou diminue.
 - La fonction d'erreur devient plus difficile à optimiser.
 - Les gradients deviennent extrêmement faibles dans les premières couches (vanishing gradients).
- Problème contre-intuitif : un réseau plus profond peut apprendre moins bien qu'un plus simple.
- Les couches supplémentaires n'apprennent pas l'identité, ce qui perturbe l'apprentissage.

Le bloc résiduel

- Chaque bloc apprend une fonction $F(x)$, puis ajoute x à la sortie :

$$y = F(x, \{W_i\}) + x$$

- Schéma d'un bloc simple :



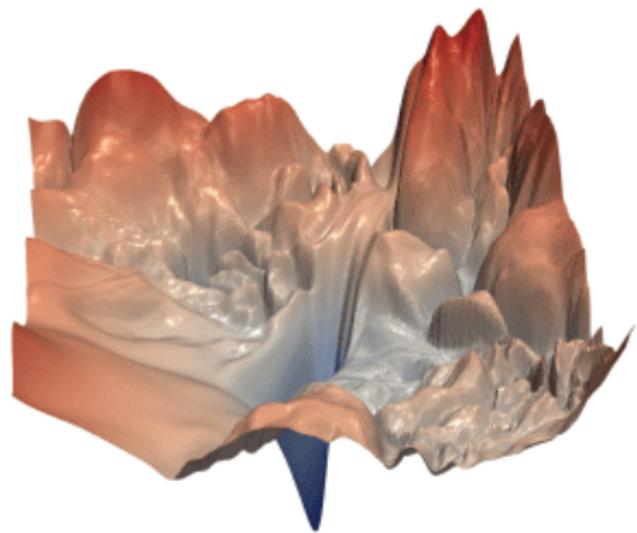
Pourquoi les réseaux résiduels ?

- Objectif : faciliter l'apprentissage en contournant les couches inutiles si besoin.
- Idée clé : forcer les couches à apprendre une fonction résiduelle $F(x) := H(x) - x$.
- On réécrit donc $H(x)$ (la transformation souhaitée) comme :

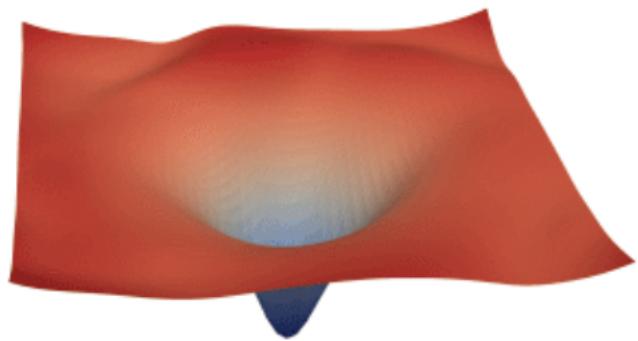
$$H(x) = F(x) + x$$

- Si $F(x) = 0$, alors $H(x) = x$: le réseau peut simplement apprendre l'identité si nécessaire.
- Cette approche permet une meilleure propagation du gradient et une meilleure convergence.

Intuition derrière les connexions résiduelles



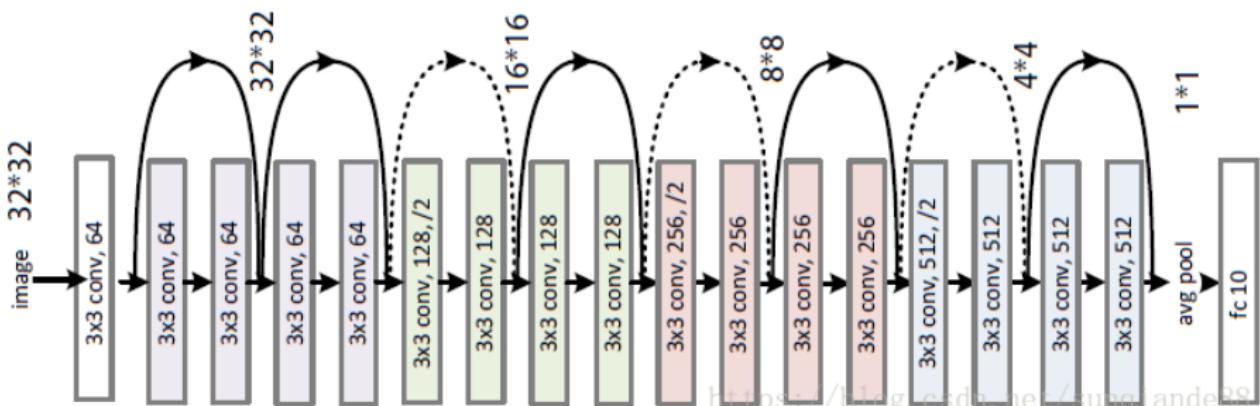
(a) without skip connections



(b) with skip connections

Architecture typique : ResNet-18

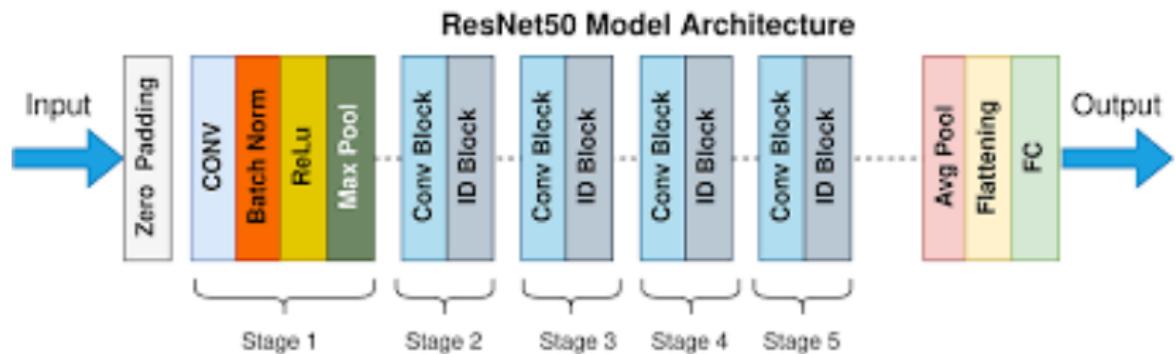
- Blocs simples avec deux conv 3×3
- $[2, 2, 2, 2]$ blocs par groupe



<https://t11g.csduheilh/funcande28>

Architecture typique : ResNet-50

- Blocs "bottleneck" avec conv $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$
 - $[3, 4, 6, 3]$ blocs par groupe



Auto-encodeurs

Introduction aux autoencodeurs

Définition et objectif :

- Les autoencodeurs sont des réseaux de neurones utilisés pour apprendre des représentations efficaces des données, généralement dans le but de les compresser et de les reconstruire.
- Ils sont composés d'un encodeur et d'un décodeur :
 - L'**encodeur** compresse l'entrée en une représentation dans un espace latent.
 - Le **décodeur** reconstruit les données d'entrée à partir de cette représentation compressée.

Applications en traitement d'image :

- Réduction de dimensionnalité
- Réduction du bruit
- Détection d'anomalies dans les images

Objectif des autoencodeurs :

- Minimiser l'erreur de reconstruction, c'est-à-dire la différence entre l'entrée originale et la sortie après encodage et décodage.

Architecture de base d'un autoencodeur 1/2

Les autoencodeurs sont composés de trois éléments principaux :

- **Encodeur :**

- Série de couches qui réduisent la dimensionnalité de l'entrée.
- Comporte généralement des couches convolutionnelles pour les données d'image afin de capturer les caractéristiques locales.

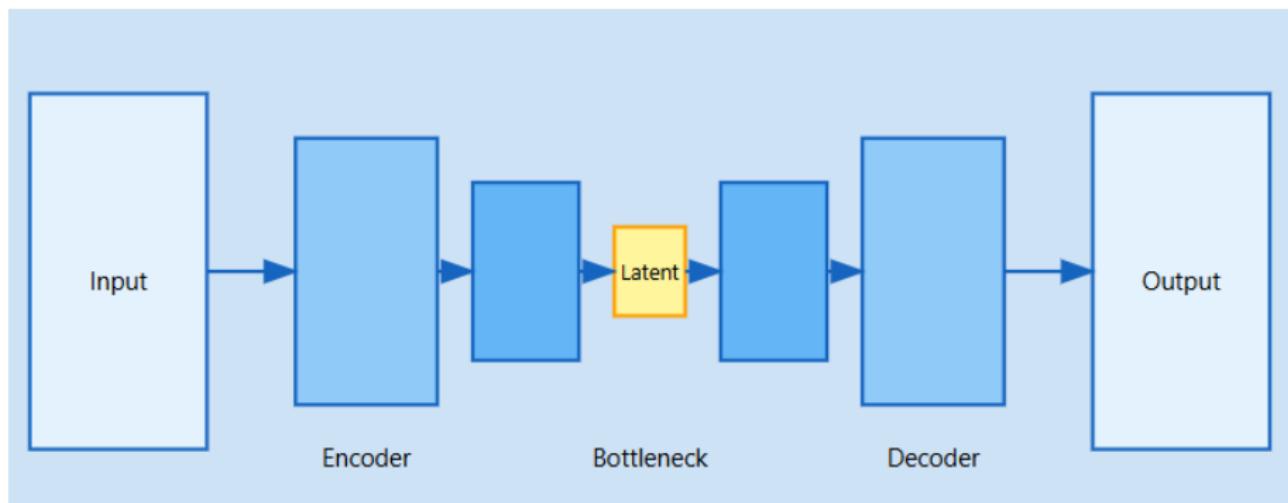
- **Goulot d'étranglement :**

- Dimension la plus réduite du réseau, représentant les caractéristiques essentielles de l'entrée.
- Agit comme une contrainte, forçant le modèle à apprendre des représentations compactes.

- **Décodeur :**

- Série de couches qui effectuent un suréchantillonnage progressif pour reconstruire l'entrée.
- Utilise des couches de convolution transposée pour récupérer les dimensions spatiales dans les données d'image.

Architecture de base d'un autoencodeur 2/2



Encodage avec des couches convolutionnelles

Rôle des couches convolutionnelles dans l'encodeur :

- Les couches convolutionnelles permettent de capturer des hiérarchies spatiales en apprenant des motifs locaux dans l'image d'entrée.
- Elles sont particulièrement efficaces pour les images, car elles permettent au modèle de détecter des bords, textures et formes de manière hiérarchique.

Fonctionnement des couches convolutionnelles :

- **Filtres** : Petites matrices (ex. : 3x3 ou 5x5) qui se déplacent sur l'image pour apprendre des caractéristiques dans différentes régions.
- **Strides et padding** :
 - **Stride (pas)** : Contrôle la taille du déplacement du filtre, influençant la taille de sortie.
 - **Padding** : Maintient les dimensions spatiales en ajoutant une bordure de zéros autour de l'image.
- **Fonctions d'activation** : ReLU est souvent utilisée pour introduire de la non-linéarité et aider le modèle à apprendre des motifs complexes.

Couches convolutionnelles

Opération de filtre dans les couches convolutionnelles :

- **Filtres (noyaux)** : Petites matrices (ex. : 3x3, 5x5) utilisées pour détecter des motifs spécifiques dans des régions locales de l'entrée.
- **Processus de convolution** :
 - Le filtre se déplace sur l'image d'entrée, effectue une multiplication élément par élément suivie d'une sommation pour créer une carte de caractéristiques.
 - Plusieurs filtres sont utilisés pour extraire différents types de caractéristiques (bords, textures, etc.).

Pooling pour la réduction de dimension :

- **Max pooling** : Réduit les dimensions spatiales de la carte de caractéristiques, en conservant les éléments les plus saillants et en réduisant la charge de calcul.
- Les couches de pooling sont généralement ajoutées après les couches convolutionnelles pour réduire progressivement la taille de l'image et se concentrer sur les caractéristiques importantes.

Couche convolutionnelle : exemple

1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1

Input Image 5x5

1	0	1
0	1	0
1	0	1

Filter 3x3

5	3	5
3	5	3
5	3	5

Feature Map 3x3

5	3	5
3	5	3
5	3	5



2x2 Pool

5	5
5	5

Max Pooling

Couches de convolution transposée (déconvolution)

Objectif de la convolution transposée :

- Les convolutions transposées (aussi appelées déconvolutions) sont utilisées pour suréchantillonner les cartes de caractéristiques et restaurer les dimensions spatiales dans le décodeur.
- Elles sont essentielles pour reconstruire les images à partir de la représentation compressée dans l'espace latent.

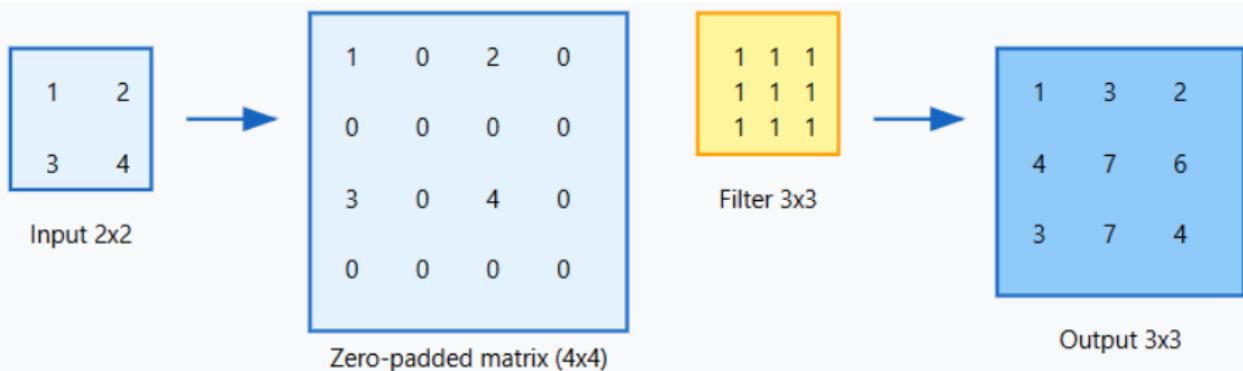
Fonctionnement de la convolution transposée :

- Contrairement à la convolution standard, la convolution transposée augmente les dimensions spatiales.
- **Inversion du processus de convolution :**
 - Dans une convolution transposée, des zéros sont généralement insérés entre les pixels de la carte de caractéristiques avant d'appliquer la convolution.
 - Cette technique "étend" efficacement les dimensions de l'entrée, permettant au décodeur de générer une sortie de plus haute résolution.

Stride et padding dans la convolution transposée :

- **Stride** : Contrôle le facteur de suréchantillonnage en définissant l'espacement entre chaque pixel de sortie.
- **Padding** : Comme pour la convolution standard, le padding peut être

Déconvolution : exemple



Transposed Convolution Process:

1. Input 2x2 is expanded with zeros (4x4)
2. Apply 3x3 convolution filter
3. Result is a 3x3 feature map

Couche de l'espace latent (goulot d'étranglement)

Objectif de l'espace latent :

- L'espace latent, également appelé goulot d'étranglement, contient une représentation compressée des données d'entrée.
- Il force l'autoencodeur à apprendre les caractéristiques les plus essentielles, en ignorant les détails moins importants.

Caractéristiques de la représentation dans l'espace latent :

- L'espace latent a une dimension plus faible que l'entrée, ce qui pousse le modèle à capturer des caractéristiques de haut niveau.
- Souvent, les caractéristiques dans l'espace latent ne sont pas directement interprétables, mais elles représentent des structures ou motifs importants dans les données.

Introduction aux autoencodeurs variationnels (VAE)

Qu'est-ce qu'un autoencodeur variationnel (VAE) ?

- Un VAE est un modèle génératif qui apprend à encoder les données dans un espace latent structuré, puis à les décoder pour approximer l'entrée d'origine.
- Contrairement aux autoencodeurs classiques, les VAE considèrent l'espace latent comme une distribution probabiliste, permettant des représentations plus souples et significatives.

Différences clés avec les autoencodeurs standards :

- Les VAE introduisent un aspect probabiliste en encodant l'entrée comme une distribution plutôt qu'un point fixe.
- Cela permet aux VAE de générer de nouvelles données en échantillonnant la distribution latente, ce qui les rend utiles pour des tâches de génération.

Applications des VAE :

- Génération et synthèse d'images (par exemple, création d'images réalistes).
- Interpolation dans l'espace latent, permettant des transitions fluides entre les points de données.
- Utile pour la détection d'anomalies, où les entrées anormales s'écartent de la distribution apprise.

Architecture d'un autoencodeur variationnel (VAE)

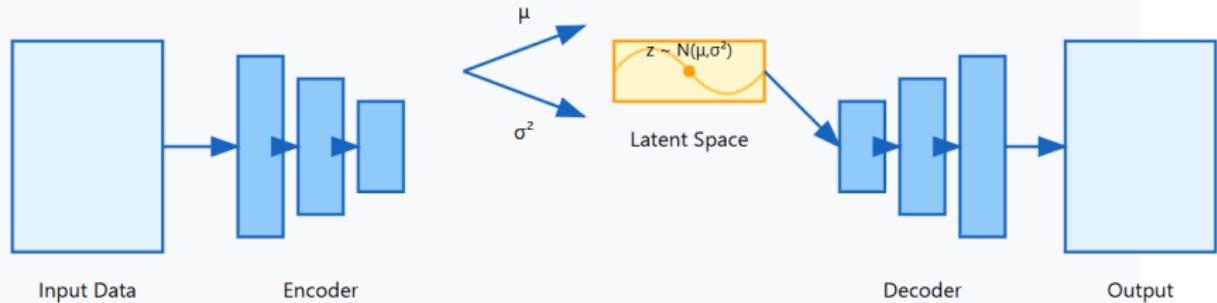
Vue d'ensemble de la structure du VAE :

- Comme un autoencodeur classique, un VAE est composé d'un encodeur, d'un espace latent, et d'un décodeur.
- Cependant, dans les VAE, l'encodeur projette l'entrée dans une distribution dans l'espace latent plutôt que dans un point fixe.

Composants d'un VAE :

- **Encodeur :**
 - Projette les données d'entrée vers une distribution latente, produisant une moyenne (μ) et une variance (σ^2) pour chaque dimension de l'espace latent.
- **Espace latent (probabiliste) :**
 - Représente les données comme une distribution, typiquement gaussienne, à partir de laquelle on peut échantillonner de nouveaux points.
 - Permet une génération structurée des données et des transitions fluides dans l'espace latent.
- **Décodeur :**
 - Reconstruit l'entrée à partir d'un point échantillonné dans la distribution latente, en générant de nouvelles données basées sur la structure apprise.

Architecture d'un VAE



Key Components:

- Encoder progressively reduces dimensions to latent space
- Latent space captures probabilistic distribution
- Decoder progressively expands dimensions to reconstruction

Fonction de perte dans les autoencodeurs variationnels (VAE)

Aperçu de la fonction de perte d'un VAE :

- La perte d'un VAE combine deux termes : la **perte de reconstruction** et la **perte de divergence KL**.
- Cette combinaison permet au modèle d'apprendre à la fois une reconstruction fidèle et un espace latent structuré.

1. Perte de reconstruction :

- Mesure la différence entre l'entrée et sa reconstruction, en utilisant typiquement l'erreur quadratique moyenne (MSE) ou l'entropie croisée binaire.
- Encourage le décodeur à produire des sorties proches de l'entrée originale.

2. Perte de divergence KL :

- Mesure l'écart entre la distribution latente apprise et une distribution normale standard ($N(0, 1)$).
- Sert de régularisation, en forçant l'espace latent à être lisse et structuré, avec des échantillons proches d'une loi normale.

Fonction de perte dans les VAE

Fonction de perte d'un VAE :

$$\mathcal{L}_{\text{VAE}}(x) = -\mathbb{E}_{q(z|x)} [\log p(x|z)] + \text{KL}(q(z|x)\|p(z))$$

Divergence de Kullback-Leibler (formule générale) :

$$\text{KL}(q(z)\|p(z)) = \int q(z) \log \left(\frac{q(z)}{p(z)} \right) dz$$

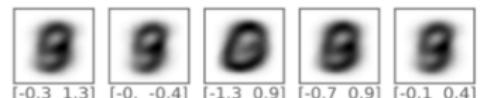
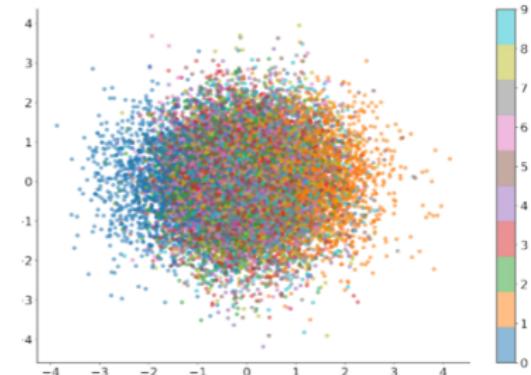
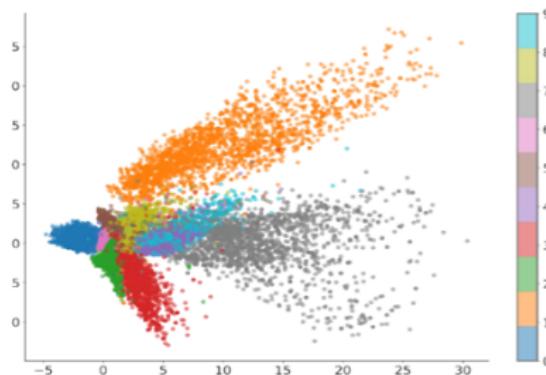
Cas gaussien : $q(z|x) = \mathcal{N}(\mu, \sigma^2)$, $p(z) = \mathcal{N}(0, I)$:

$$\text{KL}(q(z|x)\|\mathcal{N}(0, I)) = \frac{1}{2} \sum_{i=1}^d (\mu_i^2 + \sigma_i^2 - \log \sigma_i^2 - 1)$$

Astuce de reparamétrisation :

$$z = \mu + \sigma \cdot \epsilon, \quad \text{où } \epsilon \sim \mathcal{N}(0, I)$$

Décomposition de la fonction de perte



Entraînement et optimisation du VAE 1/2

Objectif d'entraînement :

- L'objectif est de minimiser la perte combinée du VAE, en équilibrant précision de reconstruction et structure de l'espace latent.
- L'apprentissage ajuste les poids de l'encodeur et du décodeur pour optimiser à la fois la reconstruction et la régularisation.

Truc de reparamétrisation :

- Pour rétropropager à travers l'opération d'échantillonnage dans l'espace latent, on utilise le truc de reparamétrisation.
- Au lieu d'échantillonner directement $z \sim q(z|x)$, on échantillonne $\epsilon \sim N(0, 1)$ et on calcule $z = \mu + \sigma \cdot \epsilon$.
- Cela permet aux gradients de circuler à travers μ et σ , rendant l'entraînement différentiable.

Entraînement et optimisation du VAE 2/2

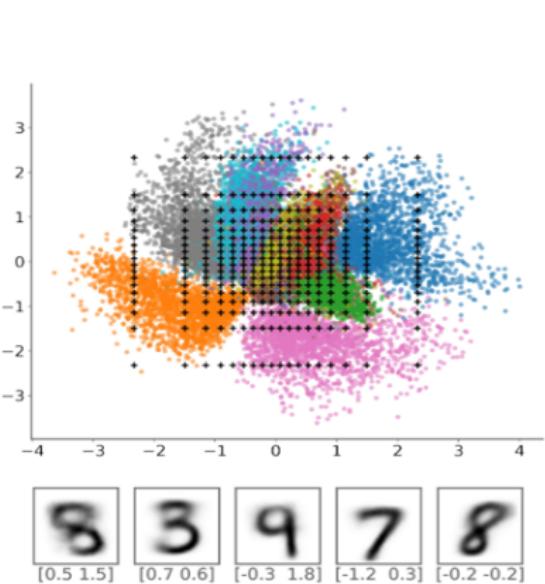
Processus d'optimisation :

- La descente de gradient stochastique (SGD) ou l'optimiseur Adam est généralement utilisé pour minimiser la perte du VAE.
- L'entraînement consiste à ajuster itérativement les poids de l'encodeur et du décodeur pour affiner la distribution latente apprise et la qualité de reconstruction.

Compromis entre reconstruction et régularisation :

- Ajuster le poids β sur le terme de divergence KL permet de contrôler l'équilibre entre fidélité de la reconstruction et régularisation de l'espace latent.
- Un β plus élevé favorise des espaces latents plus lisses mais peut réduire la précision de reconstruction.

Régularisation des VAE



Représentations distribuées

Introduction aux Embeddings

• Qu'est-ce qu'un Embedding ?

- Un embedding est une représentation vectorielle d'un mot qui capture le contexte du mot dans un document, des relations sémantiques et syntaxiques avec d'autres mots.
- Ils transforment des mots en vecteurs de nombres pour que les algorithmes de machine learning puissent les traiter efficacement.

• Pourquoi sont-ils importants ?

- Les embeddings permettent de capturer non seulement l'identité d'un mot mais aussi ses aspects sémantiques et contextuels.
- Ils facilitent des tâches telles que la classification de texte, la traduction automatique, et la détection des sentiments.

• Évolution des embeddings

- Historiquement, les mots étaient représentés comme des indices ou des vecteurs one-hot, où chaque mot est indépendant des autres.
- Les embeddings modernes, tels que Word2Vec et GloVe, représentent les mots dans des espaces vectoriels continus où les mots de sens similaires sont proches les uns des autres.

Différence entre One-hot Encoding et Embeddings

• One-hot Encoding

- Chaque mot est représenté par un vecteur avec un '1' dans la position qui lui est propre et des '0' partout ailleurs. Ce type de représentation est simple mais très inefficace en termes d'espace et ne capture pas les relations entre les mots.
- Exemple : pour un vocabulaire de dimension 100 000 :

Véhicule = [0, 0, 1, 0, 0, 0, 0, 0, ..., 0, 0]

Voiture = [0, 0, 0, 0, 0, 1, 0, 0, 0, ..., 0, 0]

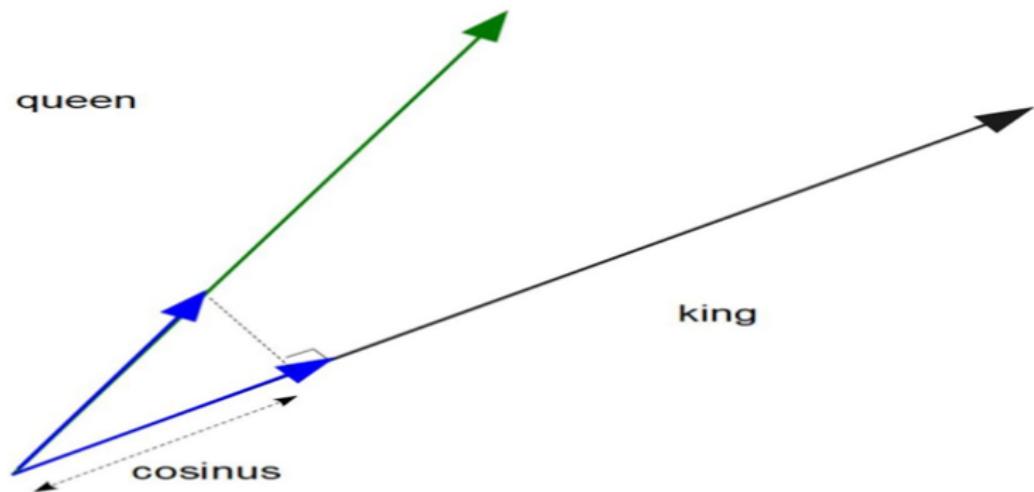
Cette représentation ne permet pas de prendre en compte la dimension sémantique

• Embeddings

- Les embeddings représentent les mots comme des vecteurs denses de nombres flottants (généralement entre 50 et 300 dimensions).
- Cette représentation est beaucoup plus riche et peut capturer des relations complexes entre les mots, comme la similarité sémantique.

Similarité sémantique

Nous sommes intéressés par des représentations de mots qui capturent la distance sémantique, sous forme de produit scalaire par exemple (ou distance cosiné).



Représentations distribuées : Principes

“You shall know a word by the company it keeps”

— J.R. Firth (1957)

Les mots sont similaires s'ils apparaissent fréquemment dans le même contexte.

- Il conduit son *véhicule* pour rentrer à la maison.
- Il conduit sa *voiture* pour rentrer chez lui.

Construction d'une représentation distribuée

Considérons le corpus suivant à titre d'exemple : **cnn in crop analysis**

cnn and svm are widely used.

linear_regression performed along with svm

linear_regression for crop and farm

svm being used for farm monitoring

Do cnn, svm and linear_regression appear in the same context?

Le vocabulaire est alors :

[cnn, in, crop, analysis, and, svm, are, widely, used,
linear_regression, performed, along, with, for, farm, being,
monitoring, do, appear, the, same, context]

$$\dim(\text{vocabulaire}) = 22$$

Similarité sémantique

La matrice de co-occurrence représente la fréquence à laquelle les mots apparaissent ensemble deux à deux.

cnn in crop ...

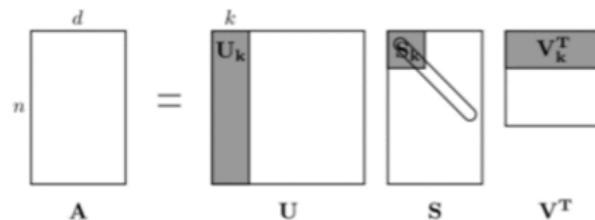
```
cnn [[0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
in [2, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1],  
crop [1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
... [1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[2, 1, 0, 0, 1, 0, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],  
[1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 0, 0, 0, 1, 2, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],  
[1, 1, 1, 0, 1, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1],  
[0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
[0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],  
[1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1],  
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0]]]
```

Réduction de la dimension

La décomposition en valeurs singulières est une technique permettant de réduire la dimension des données (similaire à l'ACP).

Étant donné une matrice $A \in \mathbb{R}^{n \times d}$

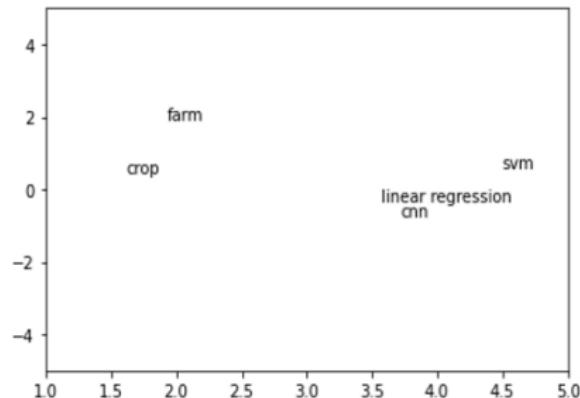
$$A = UDV^T \quad \text{où} \quad U \in \mathbb{R}^{n \times r}, D \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{d \times r}.$$



U est la matrice contenant les représentations (vecteurs de mots)

Visualisation des représentations distribuées

```
cnn :[ [ 3.72113518, -0.73233585],  
In [ 2.7940387 , -1.40864784],  
crop [ 1.61178082, 0.44786021],  
... [ 0.77784994, -0.31230389],  
[ 2.12245048, 1.29571556],  
[ 4.49293777, 0.58090417],  
[ 1.33312748, 0.66758743],  
[ 1.33312748, 0.66758743],  
[ 2.25882809, 1.80738544],  
[ 3.56593605, -0.31006976],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 0.95394179, 0.079159 ],  
[ 1.92921553, 1.94783497],  
[ 1.92921553, 1.94783497],  
[ 1.12301312, 1.42126096],  
[ 1.12301312, 1.42126096],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175],  
[ 2.26025282, -1.31571175]]
```



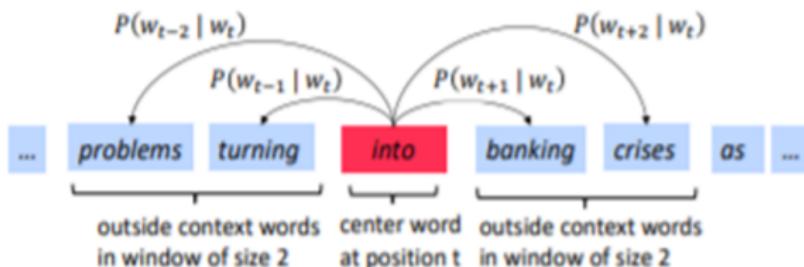
Word2Vec : Principes

- **Principes de base :**

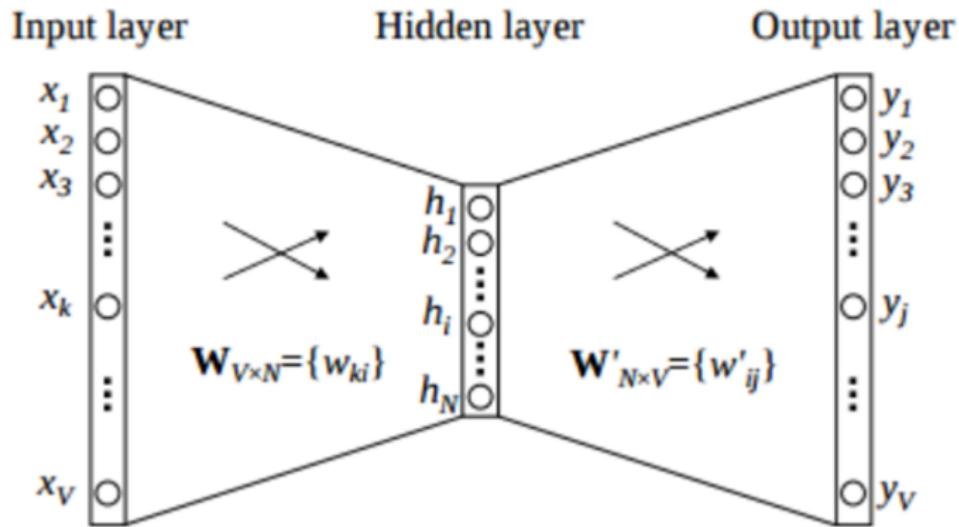
- Word2Vec est basé sur l'hypothèse distributionnelle : les mots qui apparaissent dans des contextes similaires ont des significations similaires.
- Utilise un réseau de neurones peu profond pour apprendre des embeddings de mots à partir de grands corpus.

- **Deux architectures principales :**

- ① *Continuous Bag of Words (CBOW)* : Prédit le mot cible à partir du contexte.
- ② *Skip-Gram* : Prédit le contexte à partir du mot cible.



Architecture du modèle CBOW



Modèle CBOW : Formulation mathématique

- **Objectif** : Prédire le mot cible w_t en fonction du contexte C .
- **Fonction de prédiction** :

$$P(w_t|C) = \frac{e^{\mathbf{v}_{w_t}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (8)$$

- **Où** :
 - \mathbf{v}_w est le vecteur de l'embedding du mot w .
 - \mathbf{h} est le vecteur du contexte, la moyenne des embeddings des mots du contexte.
 - W est l'ensemble de tous les mots du vocabulaire.
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

Modèle Skip-Gram : Formulation mathématiques

- **Objectif** : Prédire les mots de contexte à partir du mot cible w_t .
- **Fonction de prédiction** :

$$P(C|w_t) = \prod_{w_c \in C} \frac{e^{\mathbf{v}_{w_c}^T \cdot \mathbf{h}}}{\sum_{w \in W} e^{\mathbf{v}_w^T \cdot \mathbf{h}}} \quad (9)$$

- **Où** :
 - \mathbf{v}_{w_c} est le vecteur de l'embedding du mot de contexte w_c .
 - \mathbf{h} est le vecteur de l'embedding du mot cible w_t .
- **Optimisation** : Minimisation de la fonction de coût, souvent une forme de cross-entropy ou log-likelihood négatif.

Optimisation

- **Negative sampling :**
 - Pour chaque paire de mots (cible, contexte), des paires négatives sont générées en échantillonnant des mots aléatoires du vocabulaire.
 - Améliore l'efficacité de l'entraînement en réduisant le nombre de calculs nécessaires pour chaque étape de mise à jour.
- **Sous-échantillonnage des mots fréquents :**
 - Les mots très fréquents (par exemple, "le", "est") sont sous-échantillonnés pour améliorer la qualité des embeddings et accélérer l'entraînement.
 - Réduit la dominance des mots fréquents dans la formation des embeddings.
- **Fonction de coût :**
 - Utilisation typique de la log-likelihood négative comme fonction de coût.
 - L'objectif est de maximiser la probabilité des mots réels (positifs) tout en minimisant celle des mots échantillonnés négativement.
- **Mise à jour des poids :**
 - Les poids du réseau sont mis à jour en utilisant des méthodes de descente de gradient stochastique.
 - Les gradients sont calculés par backpropagation à travers le réseau.

Negative sampling : Formulation mathématique

- **Objectif du Negative Sampling :**

- Simplifier l'optimisation de la fonction de coût en remplaçant le problème de classification multinomiale par une série de classifications binaires.

- **Formule du Negative Sampling :**

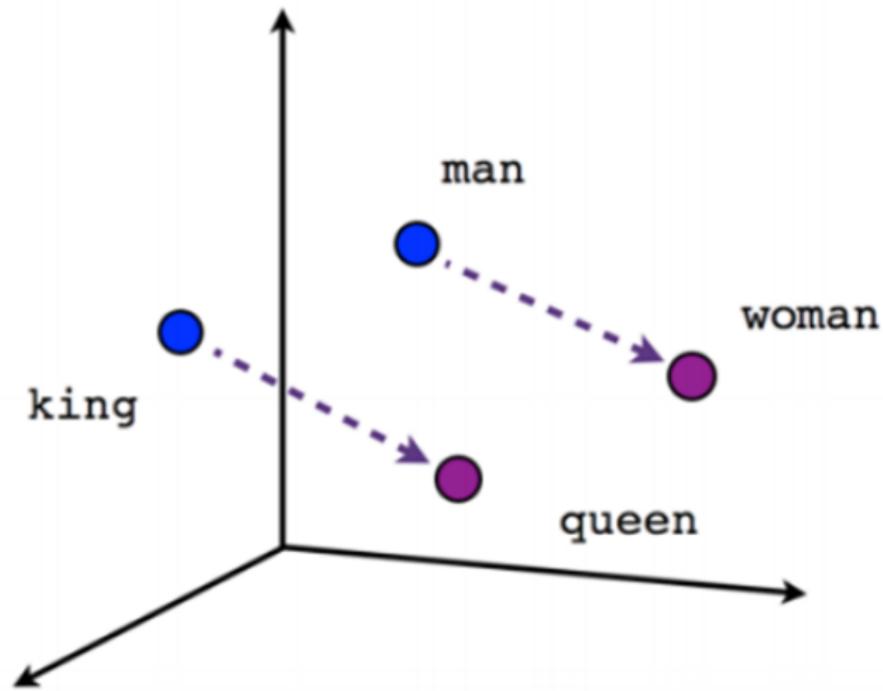
$$L(\theta) = \log \sigma(\mathbf{v}_{w_O}^T \mathbf{h}) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\mathbf{v}_{w_i}^T \mathbf{h})] \quad (10)$$

- Où $\sigma(x) = \frac{1}{1+e^{-x}}$ est la fonction logistique.
- \mathbf{v}_{w_O} est le vecteur du mot cible et \mathbf{h} est le vecteur du mot d'entrée (pour Skip-Gram) ou le vecteur du contexte (pour CBOW).
- k est le nombre de mots négatifs à échantillonner, tirés selon une distribution de probabilité $P_n(w)$.

- **Distribution de probabilité des mots négatifs ($P_n(w)$) :**

- Généralement, les mots négatifs sont échantillonnés selon une distribution qui favorise les mots fréquents, mais pas autant que la distribution de fréquence des mots dans le corpus.
- Une pratique courante est d'utiliser $P_n(w) \propto (U(w))^{3/4}$, où $U(w)$ est la fréquence brute du mot w .

Représentations Word2Vec



Applications pratiques de Word2Vec

- **Analogie de mots :**

- Word2Vec est célèbre pour capturer des relations complexes, comme "homme est à femme ce que roi est à reine".
- Permet de résoudre des analogies en utilisant des opérations arithmétiques simples sur les vecteurs de mots.

- **Clustering sémantique :**

- Les embeddings peuvent être utilisés pour regrouper des mots sémantiquement similaires, facilitant l'analyse de textes volumineux.

- **Amélioration des systèmes de recommandation :**

- Les vecteurs de mots de Word2Vec peuvent être utilisés pour améliorer la précision des recommandations en comprenant mieux les préférences des utilisateurs.

Autres représentations distribuées

Il existe d'autres représentations distribuées :

- Glove (2014) : proposé par une équipe de Stanford, il combine à la fois les techniques modernes de deep learning et les techniques statistiques (co-occurrences entre les mots). Il permet d'avoir des représentations des mots plus globales que Word2Vec.
- Fasttext (à partir de 2016) : la librairie a été créée par Facebook. Le modèle est entraîné sur des subwords (n-grams de caractères). Il est ainsi plus efficace pour traiter les mots inconnus (out of vocabulary).
- Représentations contextuelles (Transformers), etc.

Réseaux de neurones récurrents

Introduction aux RNN

Les Réseaux de Neurones Récurrents (RNN) sont une classe de réseaux de neurones artificiels spécialement conçus pour traiter des séquences de données, tels que des séries temporelles ou des séquences textuelles.

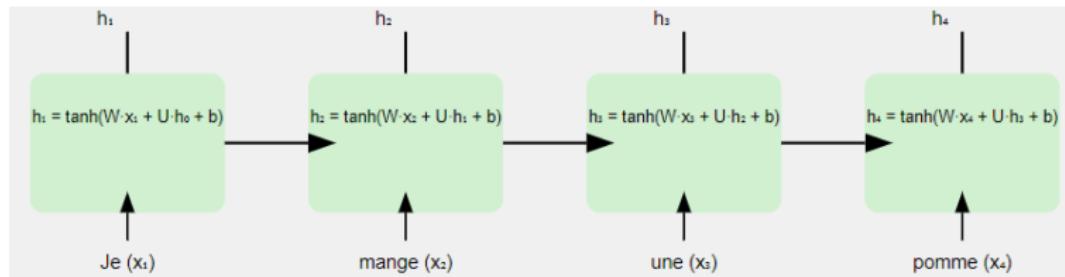
Caractéristiques:

- **Mémoire à court terme** : Les RNN ont la capacité de se "souvenir" d'informations passées grâce à leurs connexions récurrentes.
- **Traitement de séquences** : Ils sont particulièrement adaptés pour des tâches où les données sont séquentielles et où le contexte est important (ex: langage naturel, musique).
- **Modélisation de dépendances temporelles** : Les RNN peuvent capturer des dépendances temporelles et contextuelles dans les données.

Architecture des RNN

Cette architecture est dépliée pour montrer la séquence complète:

- Chaque bloc A est le même RNN à différents instants temporels.
- Le bloc A prend une entrée x_t et produit un état caché h_t , qui est alors passé à la même cellule au pas de temps suivant.
- L'état initial h_0 est souvent initialisé à zéro ou une petite valeur aléatoire.



Formulation mathématique des RNN

Un RNN est un réseau de neurones conçu pour traiter des séquences de données, caractérisé par sa capacité à maintenir une 'mémoire' des entrées antérieures dans ses connexions récurrentes.

Formulation mathématique d'un RNN simple pour une séquence de temps t :

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

où :

- x_t est l'entrée à l'instant t
- h_t est l'état caché à l'instant t
- y_t est la sortie à l'instant t
- W et b sont les paramètres du réseau
- σ est une fonction d'activation non-linéaire

Problème de disparition du gradient

Le problème de disparition du gradient survient lors de la rétropropagation dans les RNN traditionnels. Les gradients des paramètres peuvent devenir très petits, rendant l'apprentissage des dépendances à long terme difficile.

Mathématiquement, pendant la rétropropagation, si les valeurs de $\frac{\partial h_t}{\partial W}$ sont petites, le gradient total peut tendre vers zéro à mesure que T augmente.

Solutions au problème de disparition du gradient :

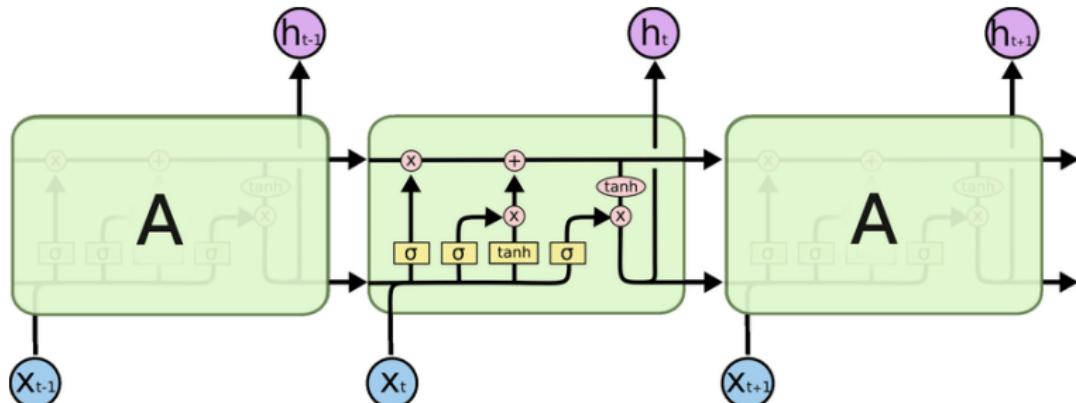
Pour atténuer le problème de disparition du gradient, différentes solutions ont été proposées :

- Utilisation de fonctions d'activation comme ReLU au lieu de sigmoïde ou tanh.
- Introduction d'architectures RNN avancées comme LSTM et GRU.
- Techniques de clipping de gradient pour éviter l'explosion des gradients.

RNN avec Long Short-Term Memory (LSTM)

Les LSTM sont une variante des RNN conçus pour mieux capturer les dépendances à long terme. Ils introduisent des 'cellules mémoire' avec des portes de régulation :

- Porte d'oubli : contrôle la quantité d'informations à retenir de l'état précédent.
- Porte d'entrée : contrôle la quantité d'informations à ajouter de l'entrée actuelle.
- Porte de sortie : détermine la quantité d'informations à transmettre à l'état suivant.



Formulation mathématique des LSTM

La formulation d'un LSTM pour un instant t :

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

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

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

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

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

$$h_t = o_t * \tanh(C_t)$$

où :

- C_t est l'état de la cellule à l'instant t
- h_t est l'état caché à l'instant t
- f_t, i_t, o_t sont respectivement les états des portes d'oubli, d'entrée, et de sortie
- Les W et b représentent les poids et biais
- σ est la fonction sigmoïde
- \tanh est la tangente hyperbolique

RNN avec Gated Recurrent Unit (GRU)

Les GRU sont une autre variante des RNN qui simplifient l'architecture des LSTM en combinant la porte d'oubli et la porte d'entrée en une seule porte de mise à jour.

Formule d'un GRU pour un instant t :

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

où :

- z_t est l'état de la porte de mise à jour
- r_t est l'état de la porte de réinitialisation
- \tilde{h}_t est l'état candidat pour h_t
- h_t est l'état caché final à l'instant t

Applications des RNN

Les RNN sont largement utilisés dans divers domaines tels que :

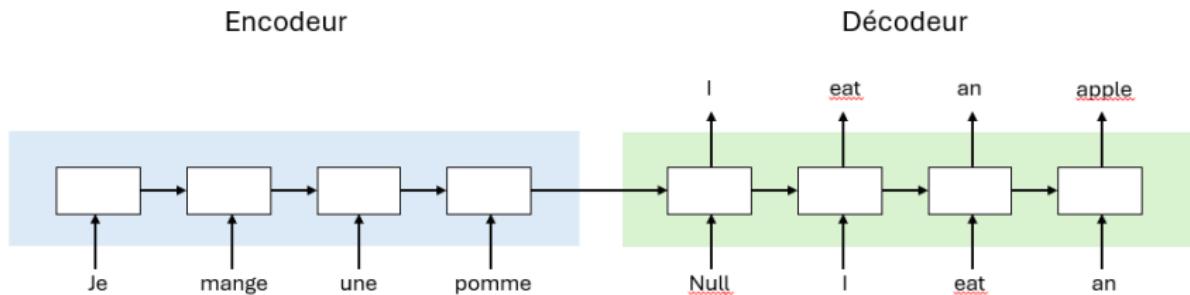
- Traitement du langage naturel : traduction automatique, génération de texte.
- Reconnaissance vocale : conversion de la parole en texte.
- Prévision de séries temporelles : prévision météorologique, analyse boursière.

Modèles Seq2Seq

Architecture de base

L'architecture seq-to-seq comprend deux composants principaux :

- **Encodeur** : Un réseau de neurones qui lit et encode la séquence d'entrée en un vecteur de contexte (une représentation dense de la séquence).
 - **Décodeur** : Un autre réseau de neurones qui lit le vecteur de contexte pour générer la séquence de sortie, un élément à la fois.



Fonctionnement de l'Encodeur

- L'encodeur transforme la séquence d'entrée, souvent textuelle, en une série d'états cachés représentant l'information contenue dans la séquence.
- Utilise généralement des réseaux de neurones récurrents (RNN), LSTM ou GRU pour traiter les dépendances temporelles.

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

où \mathbf{x}_t est l'entrée à l'instant t , et \mathbf{h}_t est l'état caché.

Fonctionnement du décodeur

- Le décodeur commence par le vecteur de contexte fourni par l'encodeur pour commencer la génération de la séquence de sortie.
- À chaque étape, le décodeur est alimenté par son propre état caché précédent et parfois par une partie de la sortie précédemment générée.
- Le processus continue jusqu'à ce qu'un symbole de fin de séquence soit généré.

$$s_t = f(y_{t-1}, s_{t-1}, C)$$

où y_{t-1} est la sortie générée à l'instant $t - 1$, s_t est l'état caché du décodeur, et C est le vecteur de contexte.

Introduction au mécanisme d'Attention

Le mécanisme d'attention est une amélioration clé apportée aux modèles seq-to-seq traditionnels, permettant au modèle de se "concentrer" sur différentes parties de la séquence d'entrée lors de la génération de la séquence de sortie.

- **Objectif** : Surmonter les limitations des encodeurs seq-to-seq qui compressent toute l'information d'une séquence d'entrée dans un vecteur de contexte fixe.
- **Avantage** : Améliore la capacité du modèle à gérer de longues séquences d'entrée, en rendant le processus de génération de la séquence de sortie plus dynamique et contextuellement informé.

Illustration du mécanisme d'Attention



Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

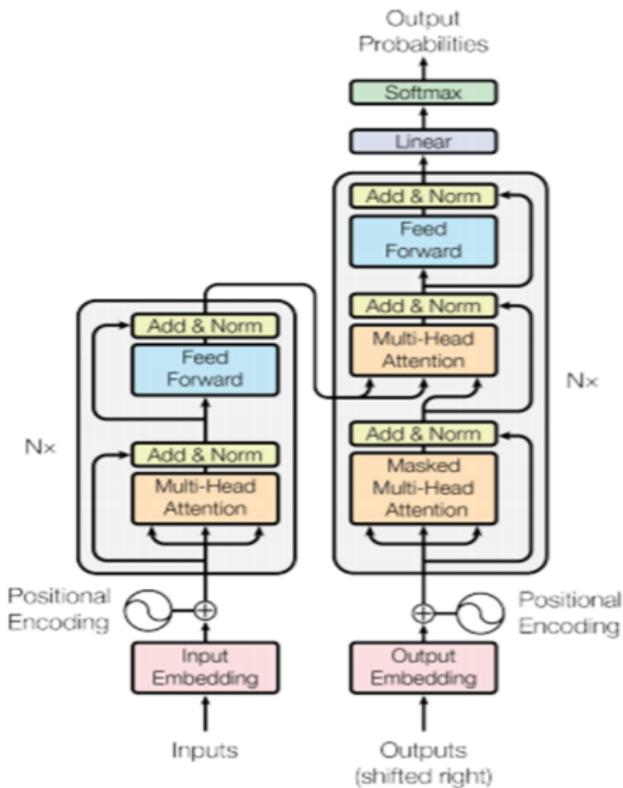
Lukasz Kaiser*
Google Brain
lukasz.kaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

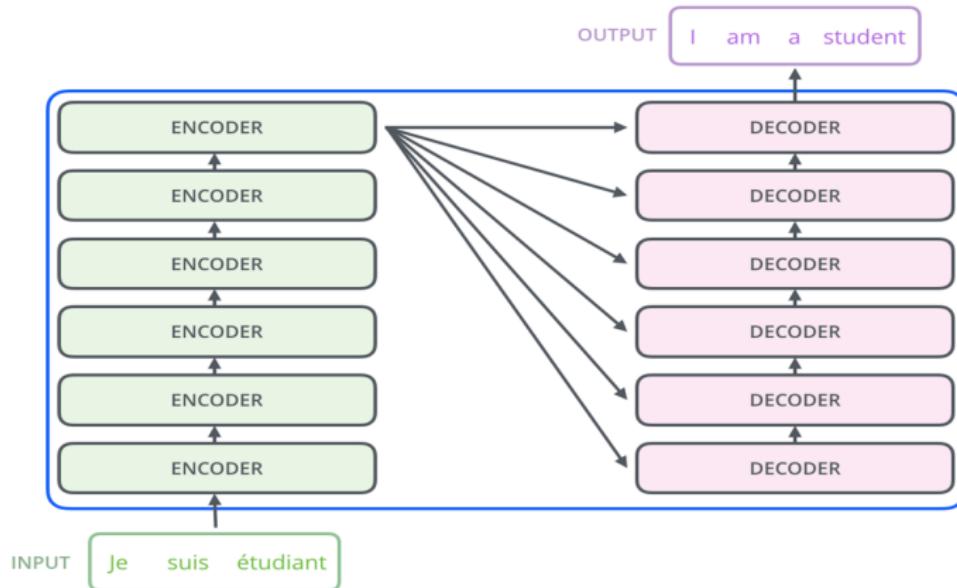
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to

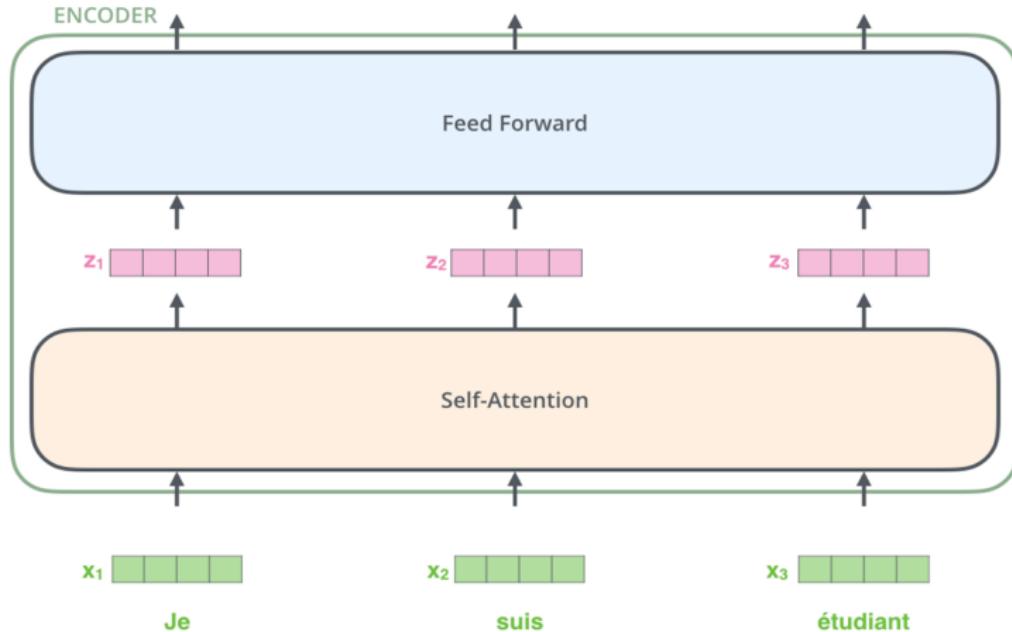
Transformer originel



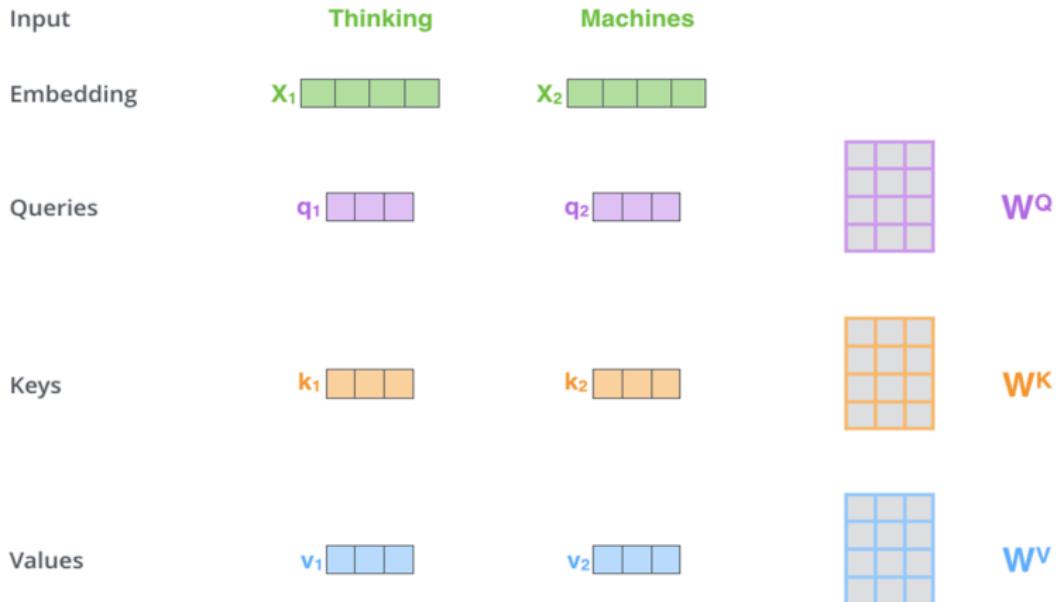
Mécanisme de cross-attention



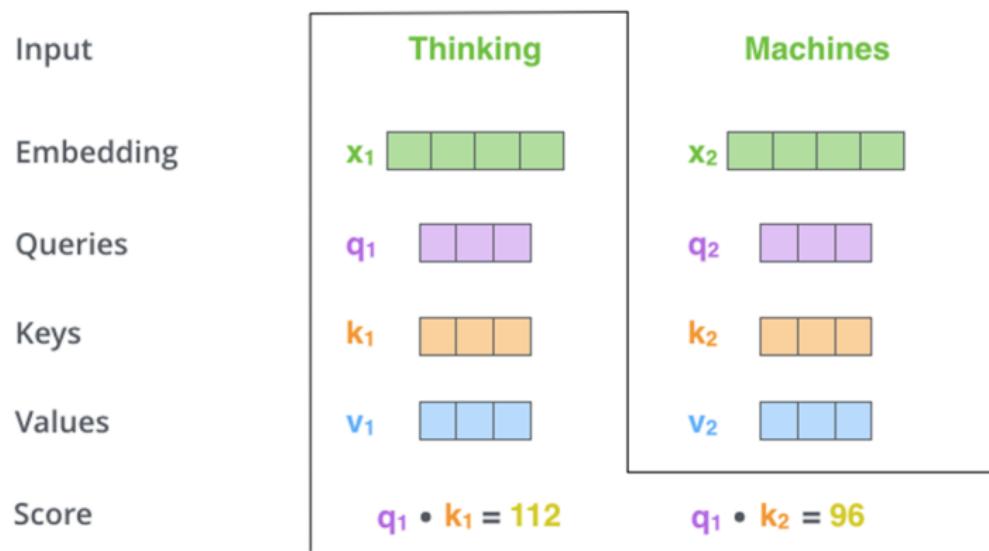
Mécanisme de self-attention



Fonctionnement du mécanisme de self-attention



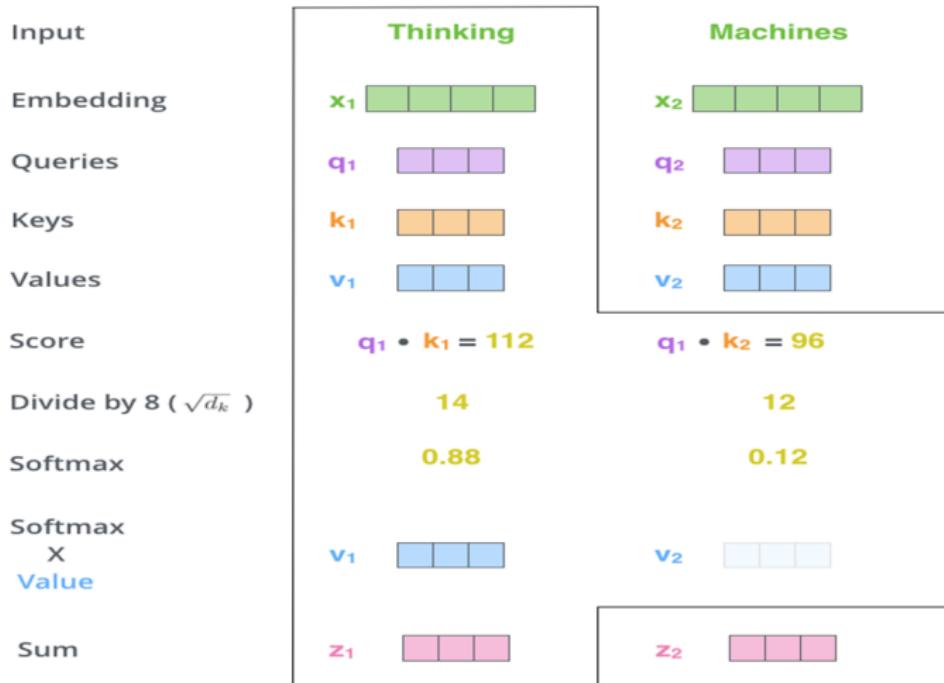
Calcul des scores de self-attention



Calcul des scores de self-attention

Input	Thinking		Machines	
Embedding	x_1		x_2	
Queries	q_1		q_2	
Keys	k_1		k_2	
Values	v_1		v_2	
Score	$q_1 \bullet k_1 = 112$		$q_1 \bullet k_2 = 96$	
Divide by 8 ($\sqrt{d_k}$)	14		12	
Softmax	0.88		0.12	

Calcul de la nouvelle représentation



Formulation matricielle

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

Formulation matricielle du mécanisme de self attention

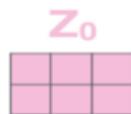
$$\text{softmax} \left(\frac{\begin{matrix} \mathbf{Q} & \times & \mathbf{K}^T \\ \begin{matrix} \text{purple} \end{matrix} & \times & \begin{matrix} \text{orange} \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \mathbf{V}$$
$$= \mathbf{Z}$$

Multihead attention

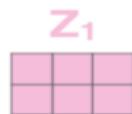


Multihead attention

ATTENTION
HEAD #0



ATTENTION
HEAD #1



...

ATTENTION
HEAD #7



\times



Les embeddings de position dans le Transformer

- Le modèle Transformer ne fonctionne pas de manière récurrente et ne sait pas, **a priori**, à quelle position se trouve chaque token dans la séquence.
- Pour fournir cette information d'ordre, on ajoute **un vecteur de position** à chaque embedding de mot.
- Dans la version d'origine, ces vecteurs sont **codés par des fonctions sinus et cosinus** :

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

- **pos** : position du token dans la séquence (de 0 à la longueur du texte).
- **i** : index d'une dimension particulière dans l'espace d'embedding.
- **d_model** : dimension du vecteur d'embedding.
- Cette méthode apprend au modèle à **distinguer** les positions et à capturer des informations de distance de manière efficace.

Tokénisation

- **Définition :** La tokénisation consiste à découper un texte en unités élémentaires appelées **tokens**. Ces tokens peuvent représenter des mots, des sous-mots, des caractères ou même des morceaux de mots.
- **Pourquoi est-ce important ?**
 - Les modèles de langage manipulent des indices ou des représentations numériques plutôt que des mots bruts.
 - Une bonne tokénisation permet de mieux gérer la morphologie (préfixes, suffixes), la gestion des mots rares, et de réduire la taille du vocabulaire.
- **Méthodes courantes :**
 - Découpage par espaces et ponctuation (simpliste).
 - Sous-mots (Byte-Pair Encoding, WordPiece, SentencePiece) : on segmente les mots en fragments fréquents pour mieux gérer les mots inconnus.
 - Caractères purs (dans certaines applications spécifiques).
- **Exemple (WordPiece) :**
 - Phrase : "J'adore le traitement automatique du langage"
 - Tokens : ["J", "'", "adore", "le", "traitement", "auto", "matique", "du", "langage"]

Byte Pair Encoding (BPE) : procédure

- **Objectif :**

- Réduire la taille du vocabulaire tout en gérant les mots inconnus.
- Capturer les éléments récurrents (préfixes, racines, suffixes) sous forme de sous-mots.

- **Principe :**

- ➊ Découper chaque mot en caractères au départ.
- ➋ Repérer la paire de symboles la plus fréquente dans tout le corpus.
- ➌ Fusionner cette paire pour former un nouveau symbole (sous-mot).
- ➍ Répéter les fusions jusqu'à la taille de vocabulaire souhaitée.

- **Avantages :**

- **Évite** les énormes vocabulaires composés de mots entiers.
- **Gère** de façon flexible les mots rares ou inconnus (en les segmentant en sous-unités).
- **Capture** la structure morphologique (préfixes, suffixes) via la répétition.

Byte Pair Encoding (BPE) : exemple

- **Phrase (mini-corpus)** : "la souris rit sous la pluie"

Au départ, on segmente en caractères (y compris les espaces) :

```
[ "l", "a", " ", "s", "o", "u", "r", "i", "s", " ", "r", "i", "t", " ",  
  "s", "o", "u", "s", " ", "l", "a", " ", "p", "l", "u", "i", "e"]
```

- **Étapes de fusion itératives** :

- ① On compte la fréquence de chaque paire de symboles (par ex. "l"+ "a" qui forme la, "s"+ "o", "o"+ "u", etc.).
- ② On **fusionne** la paire la plus fréquente pour former un nouveau **token**.
- ③ On répète jusqu'à atteindre la taille de vocabulaire souhaitée (ici, on fera seulement quelques fusions pour l'exemple).

- **Exemple de fusions probables** :

- Fusion de "l" et "a" → "la" (vu deux fois dans la phrase).
- Fusion de "s" et "o" → "so".
- Fusion de "so" et "u" → "sou" (présent dans "souris" et "sous").

- **Résultat final** :

```
[ "la", "sou", "ri", "i", "s", "i", "t", "la", "p", "l", "u", "e"]
```

Performances du tranformer originel

Entraîné sur WMT 2014 English-German dataset, comprenant près de 4.5 millions de phrases sentence, le WMT 2014 English-French dataset, comprenant près de 36 millions de phrases.

- 8 NVIDIA P10 GPUs
- **Base** : 12 heures
- **Large** : 3.5 jours

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

BERT

BERT (Bidirectional Encoder Representations from Transformers) représente une avancée majeure dans le NLP, introduisant une approche novatrice pour la modélisation de langage.

- **Contextualisation profonde** : Première architecture à utiliser pleinement la bidirectionnalité pour comprendre le contexte des mots, permettant une compréhension plus fine du langage.
- **Pré-entraînement et fine-tuning** : Méthodologie en deux étapes offrant une flexibilité pour l'adaptation à diverses tâches de TALN sans architecture spécifique à la tâche.
- **Impact sur le NLP** : Avant BERT, les approches de modélisation de langage étaient limitées par la compréhension unidirectionnelle ou des représentations statiques des mots. BERT a changé la donne en permettant des représentations contextuelles dynamiques.
- **Performances révolutionnaires** : À son introduction, BERT a établi de nouveaux standards de performance sur une gamme de benchmarks de NLP, y compris GLUE, SQuAD, etc.

Pré-entraînement de BERT

Objectifs de pré-entraînement :

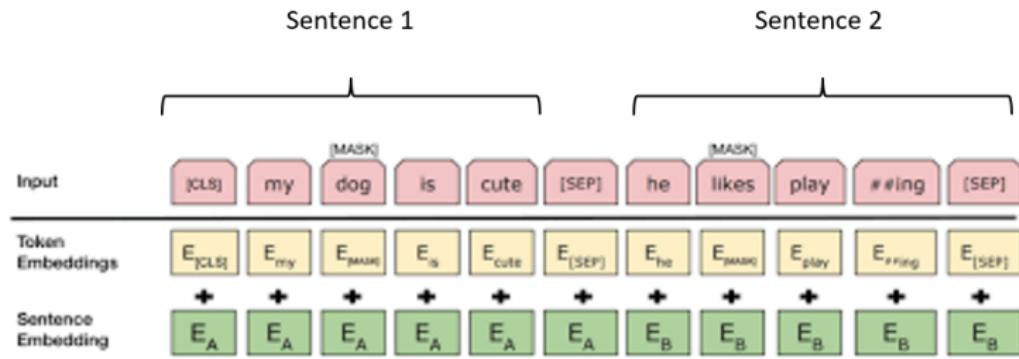
- **Masquage de token (MLM)** : 15% des tokens masqués aléatoirement, prédits par le modèle.

$$L_{\text{MLM}} = - \log P(\text{token masqué} | \text{contexte}) \quad (11)$$

- **Prédiction de la prochaine phrase (NSP)** : Déterminer si une phrase B suit logiquement une phrase A.

$$L_{\text{NSP}} = - [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \quad (12)$$

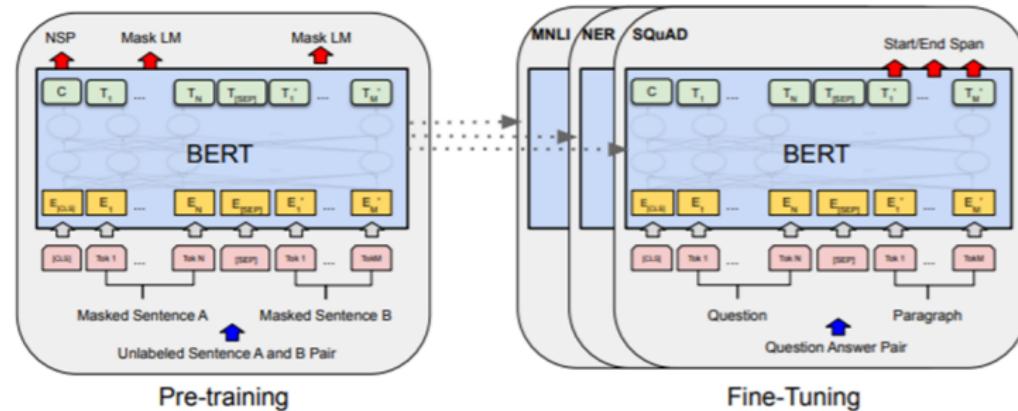
- Combinaison des pertes pour l'optimisation.



Fine-tuning de BERT pour des tâches spécifiques

Après pré-entraînement, BERT est affiné pour des tâches spécifiques:

- Ajout d'une couche de sortie spécifique à la tâche (classification, NER, QA).
 - Fine-tuning de tous les paramètres du modèle pré-entraîné sur le corpus de la tâche.



Performances de BERT

BERT a été pré-entraîné sur le BookCorpus (800 millions de mots) et English Wikipedia (2,500 millions de mots).

Deux modèles :

- **BERT Base** : 12 couches avec 110 millions de paramètres.
- **BERT Large** : 24 couches avec 340 millions de paramètres.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Improving Language Understanding by Generative Pre-Training

Alec Radford

OpenAI

alec@openai.com

Karthik Narasimhan

OpenAI

karthikn@openai.com

Tim Salimans

OpenAI

tim@openai.com

Ilya Sutskever

OpenAI

ilyasu@openai.com

Abstract

Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of our approach on a wide range of benchmarks for natural language understanding. Our general task-agnostic model outperforms discriminatively trained models that use architectures specifically crafted for each task, significantly improving upon the state of the art in 9 out of the 12 tasks studied. For instance, we achieve absolute improvements of 8.9% on commonsense reasoning (Stories Cloze Test), 5.7% on question answering (RACE), and 1.5% on textual entailment (MultiNLI).

GPT-3

- Développé par OpenAI, publié en 2020.
- Modèle auto-régressif basé sur l'architecture Transformer.

Caractéristique	Valeur / Détail
Date	2020
Paramètres	175 milliards
Corpus d'entraînement	Environ 300 milliards de tokens
Contexte maximum	2048 tokens
Ressources GPU	Plusieurs centaines de GPU (type V100)
Coût d'entraînement (estim.)	4,6 M\$ à 12 M\$ (selon les sources)

- **Applications** : génération de texte, Q&R, résumé, traduction, assistance à la programmation.

Le few-shot learning

- Les grands modèles (comme GPT-3) peuvent réaliser des tâches **sans être explicitement réentraînés** dessus.
- Le principe repose sur **l'utilisation de quelques exemples** (exemples étiquetés) directement dans le prompt ou l'entrée.
- **Zero-shot** : aucune démonstration fournie, le modèle doit comprendre la tâche à partir de sa connaissance générale.
- **One-shot / Few-shot** : on inclut un nombre très réduit d'exemples (un ou quelques-uns) pour guider le modèle dans la résolution de la tâche.
- Exemple :

English: "cat" → French: "chat"

English: "house" → French: "maison"

English: "car" → French: "voiture"

English: "tree" → French:

ChatGPT

ChatGPT, développé par OpenAI, est un modèle de langage basé sur l'architecture GPT (Generative Pre-trained Transformer) optimisé pour comprendre et générer des dialogues naturels. L'entraînement de ChatGPT se décline selon les étapes suivantes :

- ① Pré-entraînement sur un corpus volumineux :** Comme GPT-3, ChatGPT est d'abord pré-entraîné sur un vaste ensemble de données textuelles, englobant un large éventail de la littérature disponible sur Internet, pour apprendre une compréhension générale du langage.
- ② Fine-tuning supervisé :** Ensuite, ChatGPT est affiné sur des dialogues spécifiques pour améliorer ses compétences conversationnelles. Cette étape utilise des paires question-réponse et des conversations pour enseigner au modèle des structures de dialogue et des réponses contextuellement appropriées.
- ③ Reinforcement Learning from Human Feedback (RLHF):** Utilisation de techniques de renforcement pour ajuster les réponses du modèle basées sur les préférences et les corrections fournies par des évaluateurs humains, raffinant davantage la pertinence et la naturalité des réponses.

Apprentissage multimodal

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford ^{*} ¹ Jong Wook Kim ^{*} ¹ Chris Hallacy ¹ Aditya Ramesh ¹ Gabriel Goh ¹ Sandhini Agarwal ¹
Girish Sastry ¹ Amanda Askell ¹ Pamela Mishkin ¹ Jack Clark ¹ Gretchen Krueger ¹ Ilya Sutskever ¹

Abstract

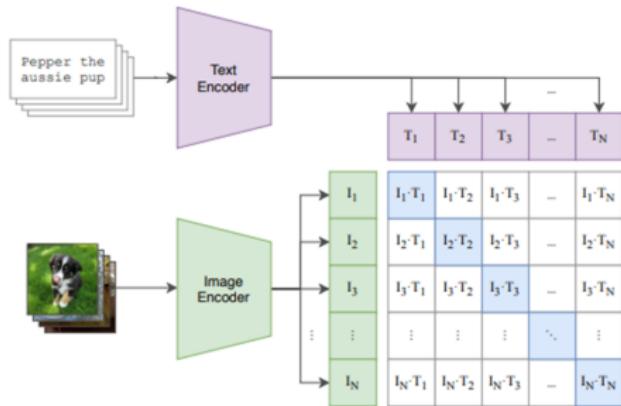
State-of-the-art computer vision systems are trained to predict a fixed set of predetermined object categories. This restricted form of supervision limits their generality and usability since additional labeled data is needed to specify any other visual concept. Learning directly from raw text about images is a promising alternative which leverages a much broader source of supervision. We demonstrate that the simple pre-training task of predicting which caption goes with which image is an efficient and scalable way to learn SOTA image representations from scratch on a dataset of 400 million (image, text) pairs collected from the internet. After pre-training, natural language is used to reference learned visual concepts (or

Task-agnostic objectives such as autoregressive and masked language modeling have scaled across many orders of magnitude in compute, model capacity, and data, steadily improving capabilities. The development of “text-to-text” as a standardized input-output interface (McCann et al., 2018; Radford et al., 2019; Raffel et al., 2019) has enabled task-agnostic architectures to zero-shot transfer to downstream datasets removing the need for specialized output heads or dataset specific customization. Flagship systems like GPT-3 (Brown et al., 2020) are now competitive across many tasks with bespoke models while requiring little to no dataset specific training data.

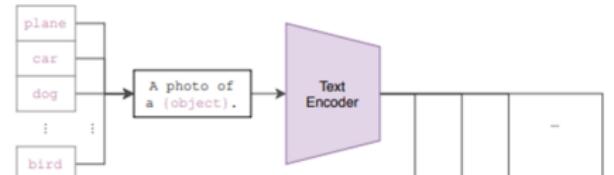
These results suggest that the aggregate supervision accessible to modern pre-training methods within web-scale collections of text surpasses that of high-quality crowd-labeled NLP datasets. However, in other fields such as computer

Architecture et entraînement de CLIP

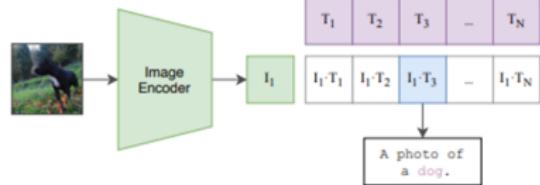
(1) Contrastive pre-training



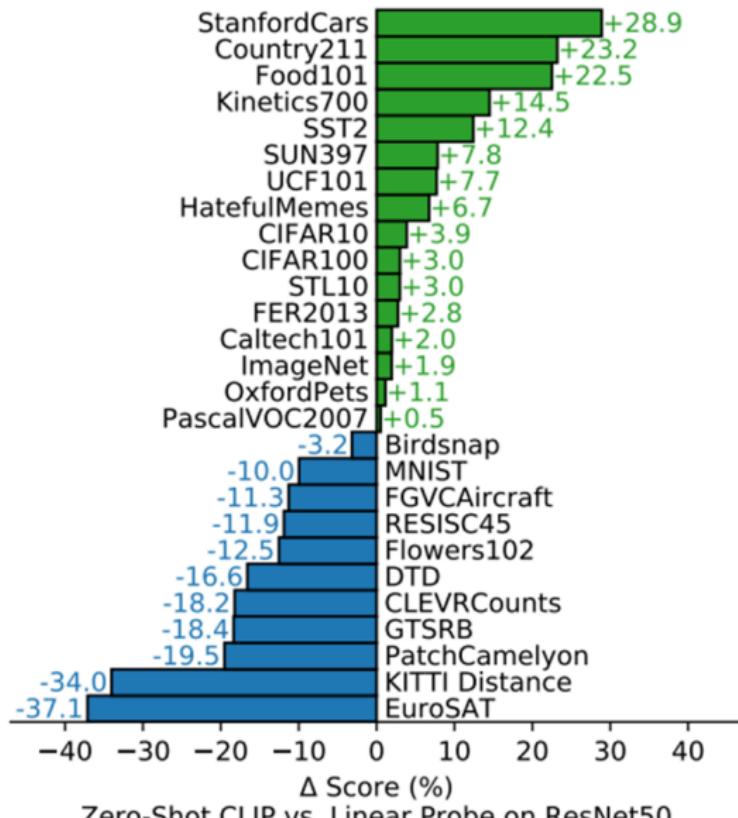
(2) Create dataset classifier from label text



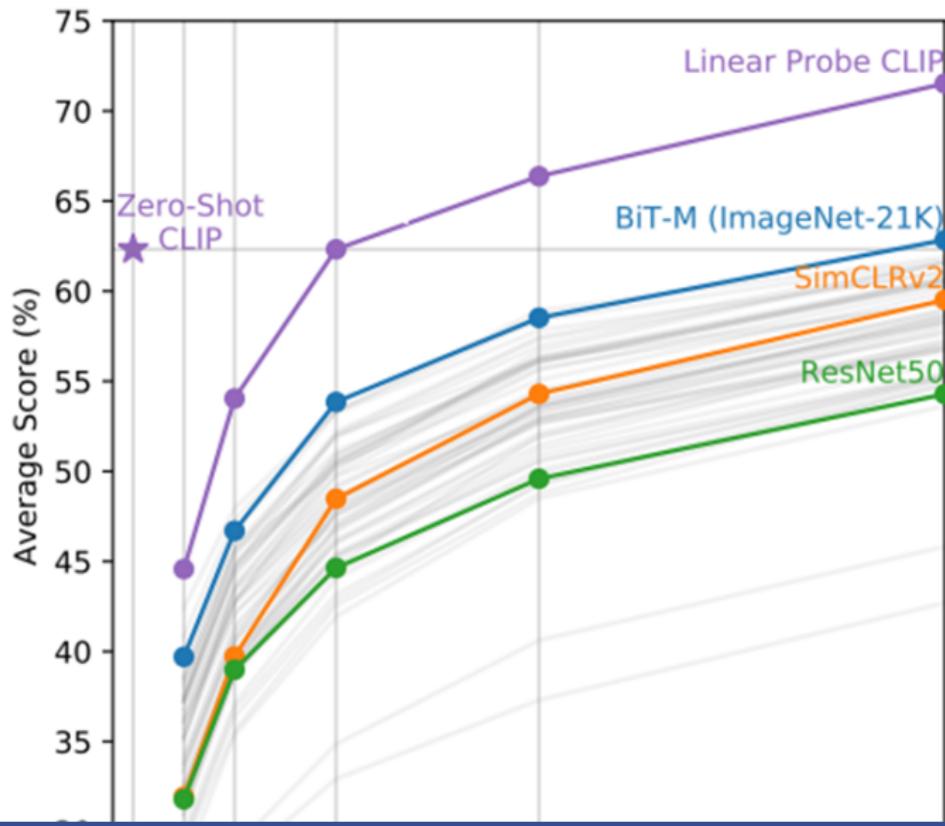
(3) Use for zero-shot prediction



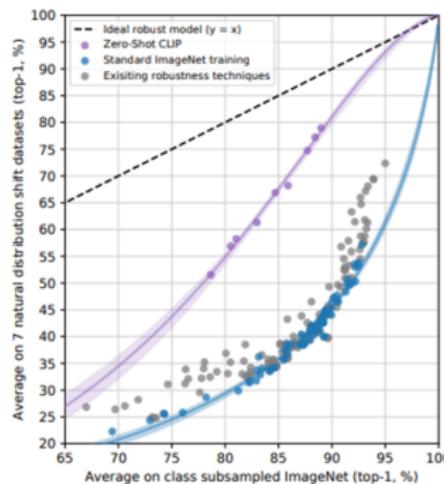
Performances de CLIP 1/2



Performances de CLIP 2/2



Robustesse de CLIP



	ImageNet	ResNet101	Zero-Shot CLIP	Δ Score
ImageNet	76.2	76.2		0%
ImageNetV2	64.3	70.1		+5.8%
ImageNet-R	37.7	88.9		+51.2%
ObjectNet	32.6	72.3		+39.7%
ImageNet Sketch	25.2	60.2		+35.0%
ImageNet-A	2.7	77.1		+74.4%

Dataset Examples

The table provides a quantitative comparison of the top-1 accuracy for ImageNet and ResNet101 across six different datasets. The 'Zero-Shot CLIP' column is empty for all rows except the first. The 'Δ Score' column indicates the performance improvement of Zero-Shot CLIP over ResNet101 for each dataset. The last row, 'ImageNet-A', shows a significant performance boost of 74.4%.

Modèles de diffusion

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley

jonathanho@berkeley.edu

Ajay Jain
UC Berkeley

ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley

pabbeel@cs.berkeley.edu

Abstract

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset, we obtain an Inception score of 9.46 and a state-of-the-art FID score of 3.17. On 256x256 LSUN, we obtain sample quality similar to ProgressiveGAN. Our implementation is available at <https://github.com/jonathanho/diffusion>.

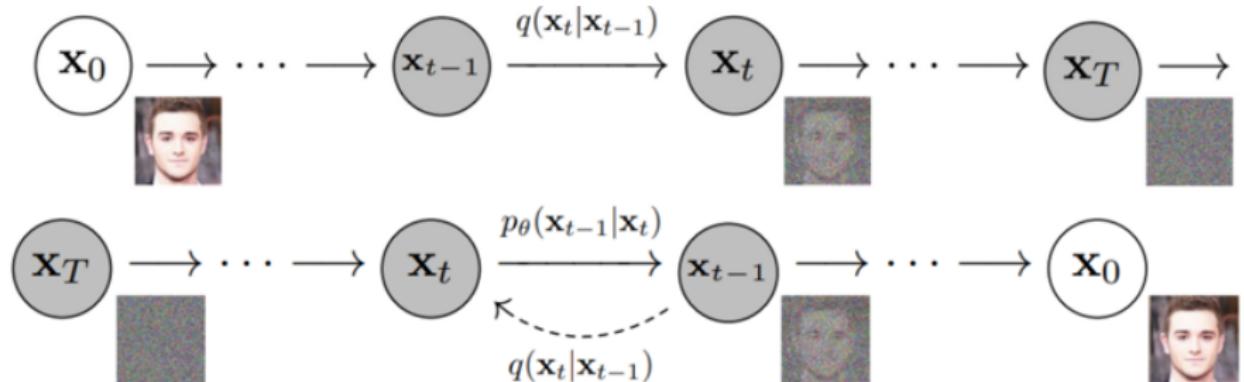
1 Introduction

Deep generative models of all kinds have recently exhibited high quality samples in a wide variety of data modalities. Generative adversarial networks (GANs), autoregressive models, flows, and variational autoencoders (VAEs) have synthesized striking image and audio samples [14, 27, 3, 58, 38, 25, 10, 32, 44, 57, 26, 33, 45], and there have been remarkable advances in energy-based modeling and score matching that have produced images comparable to those of GANs [11, 55].

Introduction

- **Définition générale :** Les modèles de diffusion (p.ex. DDPM) apprennent à générer des échantillons en inversant un processus de diffusion (ajout de bruit).
- **Concept clé :**

Processus direct $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \quad \longleftrightarrow \quad$ Processus inverse $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$.



Processus direct (Forward Process)

Idée : On ajoute progressivement du bruit gaussien sur un échantillon réel \mathbf{x}_0 pour obtenir une suite $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$.

Équations de base :

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t ; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}),$$

où β_t est un coefficient de bruit (p.ex. β_t croît linéairement ou de façon personnalisée).

Distribution en un seul pas :

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t ; \sqrt{\alpha_t} \mathbf{x}_0, (1 - \alpha_t) \mathbf{I}), \quad \text{avec } \alpha_t = \prod_{s=1}^t (1 - \beta_s).$$

Remarques :

- Plus t est grand, plus \mathbf{x}_t est bruitée.
- À $t = T$, \mathbf{x}_T est proche d'une distribution de bruit pur.

3. Processus inverse (Reverse Process)

But : Apprendre $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ pour “débruiter” la donnée étape par étape et retrouver \mathbf{x}_0 .

Forme approchée (Gaussian) :

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)).$$

Approximation courante :

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1 - \beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \alpha_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right),$$

où $\boldsymbol{\epsilon}_\theta$ représente le bruit prédit par le réseau (type U-Net).

Interprétation :

- Le modèle tente d'inférer la “propre” version de \mathbf{x}_t (en retirant le bruit).
- À chaque étape, on recule d'un pas dans le temps jusqu'à \mathbf{x}_0 .

4. Objectif d'entraînement (Loss)

Formulation générale : L'objectif est de minimiser la divergence KL entre $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)$ et $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)$.

Dans la pratique, il est courant de simplifier cette KL en une *loss* basée sur le bruit : on tire un t aléatoire, on échantillonne \mathbf{x}_t depuis $q(\mathbf{x}_t | \mathbf{x}_0)$, et on apprend à prédire le bruit ϵ .

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{x}_0, \epsilon, t} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon}, t)\|^2 \right].$$

Idée intuitive :

- Au lieu de prédire directement \mathbf{x}_0 , on prédit le bruit injecté.
- Cela facilite la convergence et stabilise l'apprentissage.

Node embedding

Node embedding

Definition

Le **node embedding** est une méthode de représentation des nœuds d'un graphe dans un espace vectoriel de faible dimension, tout en préservant la structure et les propriétés du graphe.

Utilité :

- Facilite l'application de techniques de machine learning sur les graphes en fournissant une représentation numérique des nœuds.
- Permet de capturer à la fois les informations topologiques et les attributs des nœuds.

Techniques communes :

- **DeepWalk, Node2Vec** : Utilisent des séquences de marche aléatoire pour apprendre les embeddings.
- **Graph Neural Networks (GNNs)** : Apprennent des embeddings en intégrant les caractéristiques des voisins d'un nœud.

Node embedding

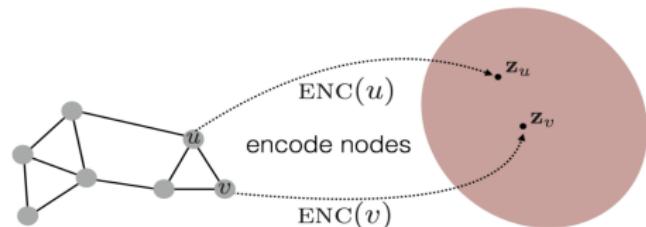


Figure: Transformation des noeuds en vecteurs dans un espace de dimension plus réduite.

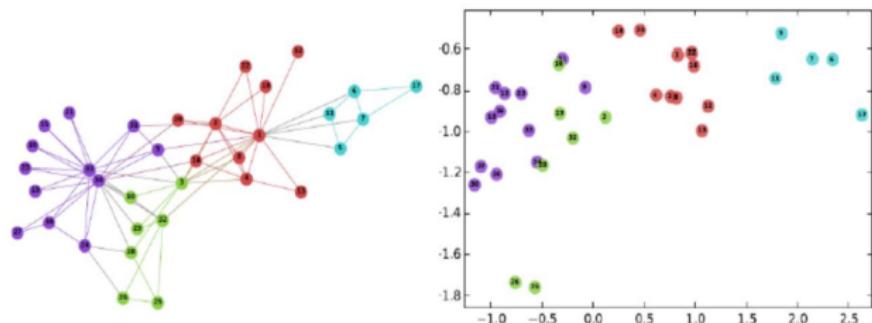


Figure: Représentation des embeddings dans l'espace cible.

Node2Vec - Formalisme mathématique

Node2Vec est un algorithme qui apprend des représentations vectorielles de nœuds dans un graphe en utilisant des marches aléatoires biaisées.

Définition de la marche aléatoire biaisée : Une marche aléatoire biaisée est définie par deux paramètres, p et q , qui déterminent la probabilité de transition d'un nœud à l'autre :

- p (paramètre de retour) : Contrôle la probabilité de revisiter un nœud précédent.
- q (paramètre d'exploration) : Gère l'équilibre entre l'exploration des voisins immédiats et des régions éloignées.

Probabilité de transition : La probabilité de transition d'un nœud v à un nœud x est donnée par :

$$\pi_{vx} = \begin{cases} \frac{1}{p} & \text{si } x \text{ est le nœud précédent} \\ 1 & \text{si } x \text{ est un voisin direct de } v \\ \frac{1}{q} & \text{si } x \text{ est éloigné de } v \end{cases} \quad (13)$$

Ces probabilités sont normalisées pour chaque nœud.

Génération de marches aléatoires

Node2Vec génère des marches aléatoires biaisées pour explorer efficacement le graphe :

- **Initialisation** : Chaque marche aléatoire commence à un nœud choisi aléatoirement.
- **Sélection des voisins** : À chaque étape, le prochain nœud est choisi parmi les voisins du nœud actuel, en utilisant la probabilité de transition basée sur p et q .

Formulation mathématique : La probabilité de transition d'un nœud v vers un nœud voisin x est donnée par :

$$\pi_{vx} = \frac{\alpha_{pq}(v, x)}{Z} \quad (14)$$

où $\alpha_{pq}(v, x)$ est un poids basé sur p et q , et Z est un facteur de normalisation.

- $\alpha_{pq}(v, x)$ définit la probabilité de se déplacer vers le nœud x en prenant en compte la structure locale et globale du graphe.

Node2Vec

Après avoir générée des séquences de nœuds via des marches aléatoires, Node2Vec utilise un modèle de type Skip-Gram pour apprendre les embeddings.

Modèle Skip-Gram : Le modèle vise à maximiser la probabilité conditionnelle des nœuds voisins dans les séquences générées.

$$\max \sum_{v \in V} \sum_{u \in N(v)} \log \Pr(u|v) \quad (15)$$

où

$$\Pr(u|v) = \frac{\exp(u \cdot v)}{\sum_{w \in V} \exp(w \cdot v)} \quad (16)$$

- V est l'ensemble des nœuds dans le graphe.
- $N(v)$ est l'ensemble des nœuds voisins de v dans les marches aléatoires.
- $f(v)$ est l'embedding du nœud v .

Optimisation : Node2Vec utilise des techniques comme le Negative Sampling pour optimiser l'apprentissage des embeddings.

Node2Vec

Node2Vec utilise le modèle Skip-Gram pour apprendre les embeddings de nœuds :

- **Objectif d'apprentissage** : Maximiser la similarité des embeddings des nœuds qui apparaissent fréquemment ensemble dans les marches aléatoires.
- **Negative sampling** : Améliore l'efficacité de l'apprentissage en échantillonnant des nœuds "négatifs" pour chaque nœud cible, accélérant ainsi la convergence.

Fonction objective :

$$\max \sum_{v \in V} \left(\sum_{u \in N(v)} \log \sigma(u \cdot v) + \sum_{k \in K} \mathbb{E}_{k \sim P(n)} [\log \sigma(-k \cdot v)] \right) \quad (17)$$

où σ est la fonction sigmoid, $N(v)$ est l'ensemble des nœuds voisins de v , et K est un ensemble de nœuds négatifs échantillonés selon la distribution $P(n)$.

Introduction à DeepWalk

Definition

DeepWalk est une technique d'embedding de graphes qui utilise des marches aléatoires uniformes pour apprendre des représentations vectorielles des nœuds dans un espace de faible dimension.

Caractéristiques principales :

- **Marches aléatoires uniformes** : Explore le graphe en effectuant des marches aléatoires, traitant chaque transition de nœud avec une probabilité égale.
- **Modèle de langage** : Utilise le modèle Skip-Gram de Word2Vec pour apprendre des embeddings à partir des séquences de nœuds générées par les marches aléatoires.
- **Capturer la structure du voisinage** : Les embeddings appris reflètent les structures de voisinage des nœuds dans le graphe.

Génération de marches aléatoires dans DeepWalk

DeepWalk génère des marches aléatoires pour chaque nœud du graphe, servant de base à l'apprentissage des embeddings.

Processus de génération :

- ① Chaque marche aléatoire commence à un nœud sélectionné aléatoirement.
- ② La marche se poursuit en choisissant aléatoirement un voisin à chaque étape, pour une longueur prédéfinie.

Importance des marches aléatoires :

- Les séquences de nœuds générées imitent les "phrases" dans le langage naturel, capturant les contextes locaux des nœuds dans le graphe.
- Permettent d'extraire des informations sur les structures locales et globales du graphe sans nécessiter de connaissances spécifiques sur sa topologie.

Apprentissage des embeddings avec DeepWalk

DeepWalk utilise le modèle Skip-Gram pour transformer les marches aléatoires en embeddings de nœuds.

Méthodologie :

- Chaque nœud dans une marche aléatoire est traité comme un "mot" et la séquence complète comme une "phrase".
- Le modèle Skip-Gram apprend à prédire les nœuds voisins dans une séquence, en maximisant la probabilité des nœuds voisins étant donné un nœud cible.

Fonction objective :

$$\max \sum_{v \in V} \left(\sum_{u \in N(v)} \log \Pr(u|f(v)) \right) \quad (18)$$

où V est l'ensemble des nœuds, $N(v)$ est l'ensemble des nœuds dans le voisinage de la marche aléatoire de v , et $f(v)$ est l'embedding du nœud v .

Optimisation : Utilise des techniques comme le Negative Sampling pour améliorer l'efficacité de l'apprentissage.

Réseaux de neurones graphiques (GNN)

Introduction aux Graph Convolutional Networks (GCN)

Definition

Les **Graph Convolutional Networks (GCN)** sont une classe de réseaux de neurones utilisés pour l'apprentissage sur graphes, combinant la structure des graphes avec des techniques de convolution.

Principes de Base :

- Intègrent à la fois les caractéristiques des nœuds et la structure topologique du graphe.
- Adaptent les méthodes de convolution, typiques des CNNs, au domaine des graphes.

Applications :

- Classification de nœuds et de graphes.
- Analyse de réseaux sociaux, bioinformatique, chimie computationnelle.

Architecture des GCN

Structure des couches :

- Les GCN sont généralement composés de plusieurs couches de convolution, suivies de couches de pooling et de couches entièrement connectées.
- Chaque couche convulsive capte des informations de voisinage à un degré plus élevé.

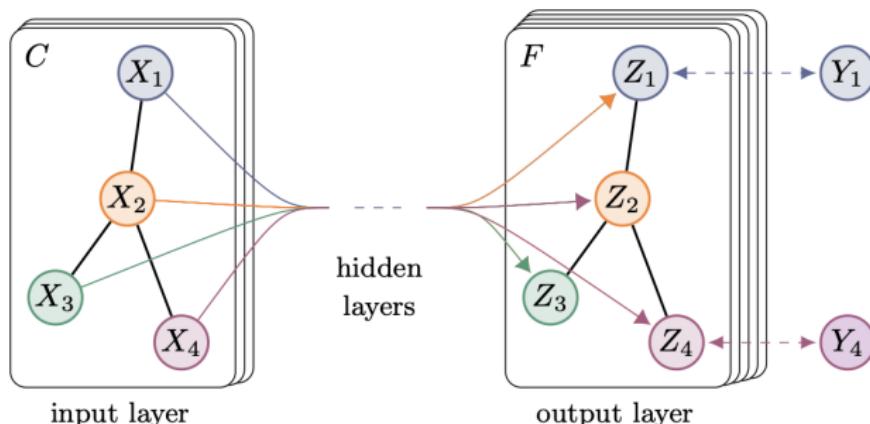


Figure: Architecture d'un Graph Convolutional Network

Fonctionnement des GCN

Mécanisme de convolution :

- Chaque couche d'un GCN applique une opération de convolution qui agit sur les nœuds et leurs voisins immédiats.
- Les caractéristiques des nœuds sont mises à jour en combinant leurs propres caractéristiques avec celles de leurs voisins.

Formule de convolution :

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$$

où \tilde{A} est la matrice d'adjacence avec des boucles ajoutées, \tilde{D} est le degré de \tilde{A} , $H^{(l)}$ est la représentation des nœuds à la couche l , et σ est une fonction d'activation.

Apprentissage des GCN

Processus d'apprentissage :

- Les GCNs sont entraînés en utilisant la rétropropagation, similaire aux CNNs traditionnels.
- Les poids des couches convolutives sont ajustés pour minimiser une fonction de perte définie, souvent une perte de classification croisée.

Optimisation :

- Des techniques comme la descente de gradient stochastique ou Adam sont utilisées pour l'optimisation.
- Les régularisations, telles que le dropout, peuvent être appliquées pour éviter le surajustement.

Défis et limitations des GCN

Défis:

- **Échelle** : La taille et la complexité des graphes peuvent rendre l'entraînement des GCNs coûteux en termes de calculs, surtout pour les grands réseaux.
- **Hétérogénéité** : Les graphes avec divers types de nœuds et d'arêtes peuvent nécessiter des adaptations spécifiques des modèles GCN.

Limitations :

- **Sur-smoothing** : Avec un trop grand nombre de couches, les caractéristiques des nœuds peuvent devenir indifférenciables, réduisant ainsi l'efficacité du modèle.
- **Indépendance spatiale** : Les GCNs se basent sur l'hypothèse que les nœuds proches sont similaires, ce qui n'est pas toujours le cas.
- **Manque de flexibilité** : Les architectures GCN standard peuvent ne pas être optimales pour tous les types de structures de graphes ou de tâches d'apprentissage.

Introduction aux Graph Attention Networks (GAT)

Definition

Les **Graph Attention Networks (GAT)** sont une forme de réseaux de neurones pour graphes qui utilisent des mécanismes d'attention pour pondérer les contributions des voisins d'un nœud.

Principes :

- Intègrent les informations des voisins d'un nœud en mettant l'accent sur les plus pertinents, grâce à l'attention.
- Permettent de gérer les graphes hétérogènes et de capter des relations complexes entre les nœuds.

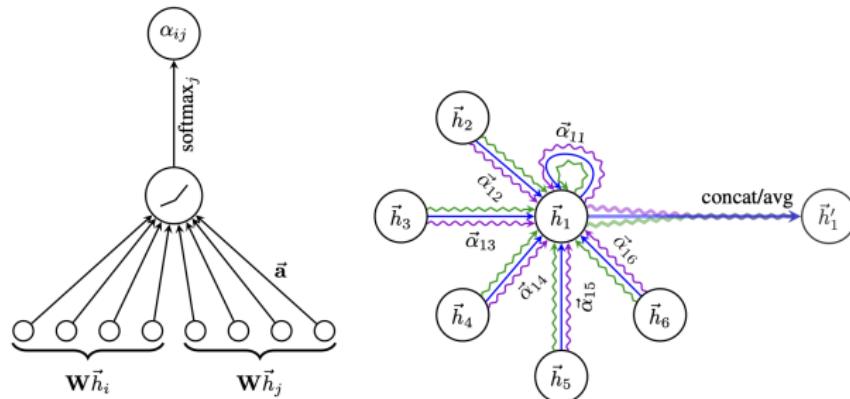
Applications :

- Classification de nœuds, analyse de réseaux sociaux, bioinformatique.
- Tâches nécessitant une compréhension fine des relations entre entités.

Architecture des GAT

Structure des couches :

- Les GAT sont composés de plusieurs couches d'attention, chacune mettant à jour les caractéristiques des nœuds en se basant sur leurs voisins.
- L'attention multi-tête peut être utilisée pour capturer divers aspects de la relation entre les nœuds.



Mécanisme d'Attention et mise à jour des états dans les GAT

Fonctionnement de l'Attention :

- Chaque noeud reçoit des informations de ses voisins avec un poids d'attention, calculé dynamiquement.
- L'attention permet de se concentrer sur les informations les plus pertinentes venant des voisins.

Calcul de l'Attention :

$$\alpha_{ij} = \frac{\exp(\sigma(a^T[Wh_i \| Wh_j])))}{\sum_{k \in N(i)} \exp(\sigma(a^T[Wh_i \| Wh_k])))}$$

où α_{ij} est le coefficient d'attention entre les noeuds i et j , h sont les caractéristiques des noeuds, W est une matrice de poids et a est un vecteur d'attention appris.

Mise à Jour des États des Nœuds :

- Les états des noeuds sont mis à jour en combinant les informations pondérées par l'attention de leurs voisins.
- $h'_i = \sigma(\sum_{j \in N(i)} \alpha_{ij} Wh_j)$ où h'_i est le nouvel état du noeud i et σ est une fonction d'activation non linéaire.

Mécanisme de Multi-Head Attention dans les GAT

Mécanisme de Multi-Head Attention :

Dans la pratique, plusieurs mécanismes d'attention (têtes) sont appliqués à chaque nœud, capturant ainsi divers aspects des informations.

Combinaison des résultats des têtes :

- **Concaténation** (couches cachées) :

$$h'_i = \left\|_{k=1}^K \sigma \left(\sum_{j \in N(i)} \alpha_{ij}^k W^k h_j \right) \right\|$$

- **Moyenne** (couche de sortie) :

$$h'_i = \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in N(i)} \alpha_{ij}^k W^k h_j \right)$$

Avantages :

- Augmente la capacité du modèle à extraire et combiner diverses caractéristiques des nœuds et de leurs relations.
- Améliore la représentation globale des nœuds en intégrant plusieurs perspectives.

Apprentissage des GAT

Processus d'apprentissage :

- Les GAT sont entraînés en utilisant la rétropropagation, avec une attention ajustée pour optimiser la tâche spécifique.
- Les coefficients d'attention sont appris pour maximiser la performance sur la tâche de prédiction.

Optimisation :

- Des techniques d'optimisation comme la descente de gradient stochastique ou Adam sont employées.
- Le dropout et d'autres stratégies de régularisation peuvent être utilisés pour améliorer la généralisation.

Défis des GAT

Défis :

- Gestion de grands graphes : La complexité computationnelle peut augmenter avec la taille du graphe.
- **Interprétabilité** : Comprendre comment les mécanismes d'attention prennent leurs décisions reste un défi.
- **Hétérogénéité des Graphes** : Adapter les GAT pour fonctionner efficacement sur des graphes avec des types de nœuds et d'arêtes variés

Impact : Les GAT offrent une approche puissante et flexible pour l'apprentissage sur graphes, avec un potentiel significatif pour des avancées dans divers domaines.

Merci de votre attention

redha.moulla@axia-conseil.com