Deep learning

Redha Moulla

février - mars 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.

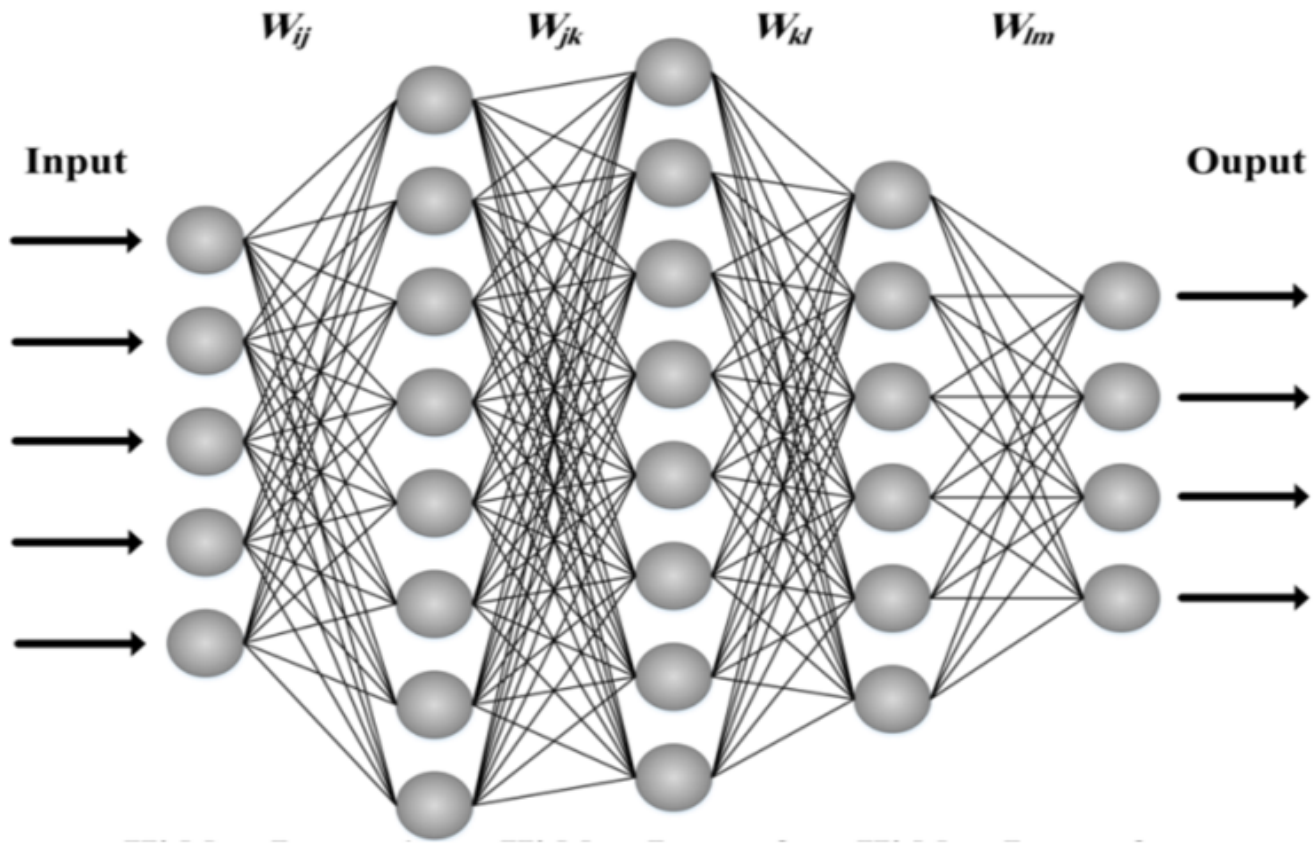
Introduction aux réseaux de neurones

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.

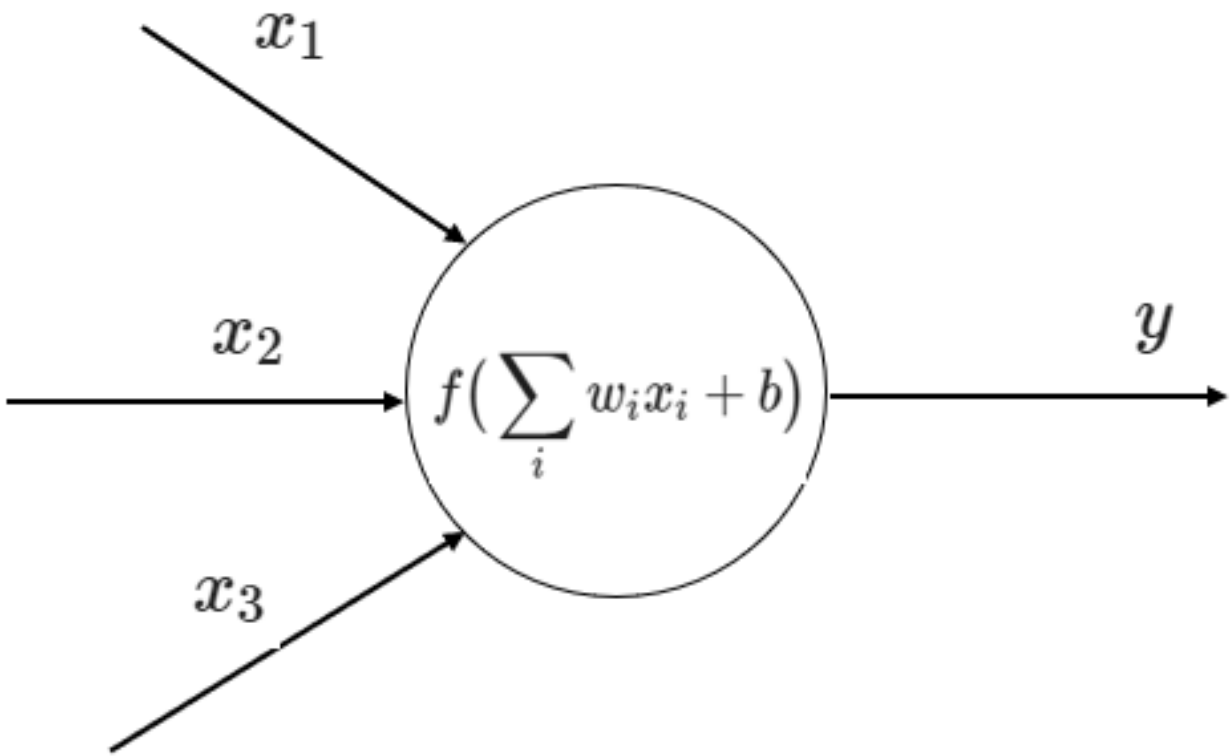
$$z_i = \sum_{j=1}^{m} w_{ij} x_j + b_i$$

$$a_i = f(z_i)$$

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 aritificiel



Fonctions d'activation

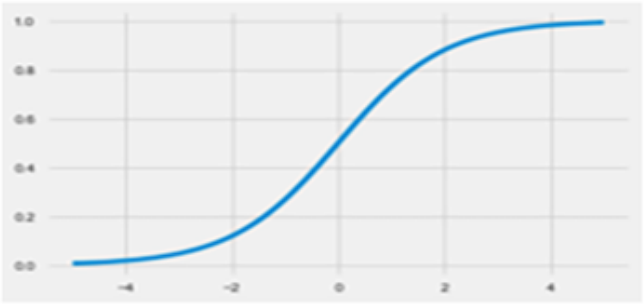
Les fonctions d'activation permettent au modèles d'apprendre des relations plus complexes en cpaturant les non-linéarités dans les données.

- **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.

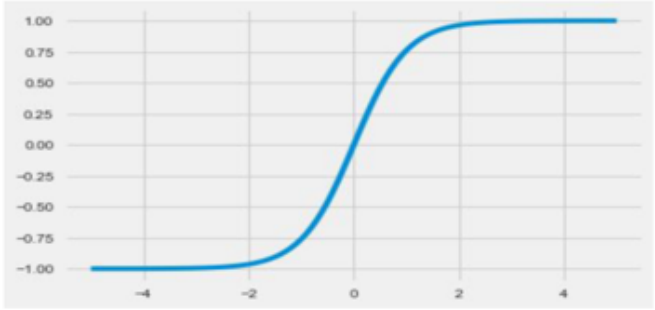
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Illustration des fonctions d'activation

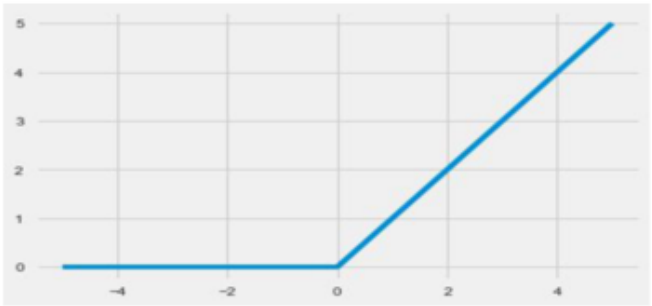
Sigmoïde



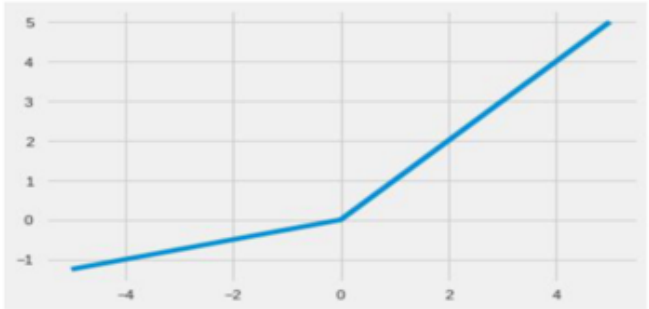
Tanh



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) :

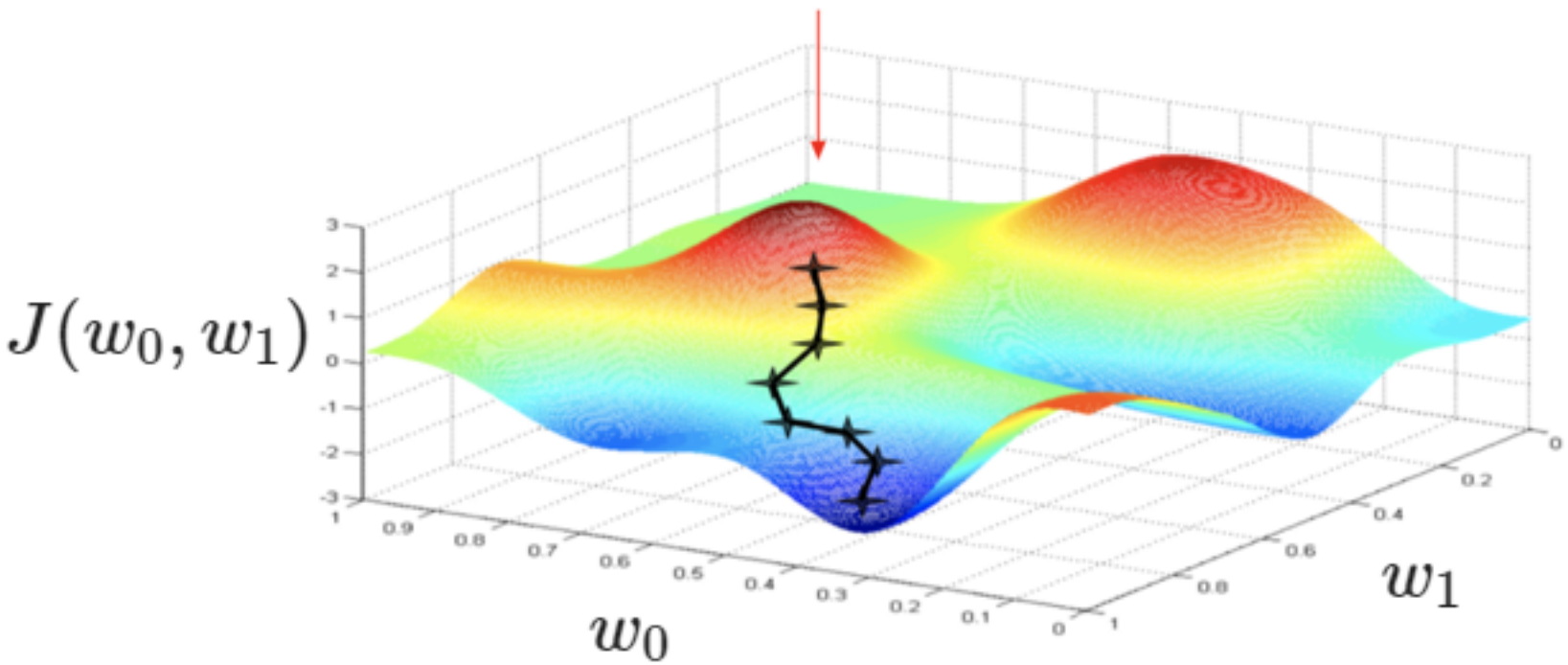
- Méthode d'optimisation utilisée pour mettre à jour les poids du réseau de manière itérative.
- À chaque itération, un sous-ensemble (batch) de données est utilisé pour calculer le gradient de la fonction de coût.
- Les poids sont mis à jour dans la direction opposée du gradient pour réduire l'erreur.

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

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 :

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

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

Définition de l'erreur locale :

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

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

Rétropropagation de l'erreur

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

2. Rétropropagation de l'erreur :

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

3. Calcul des gradients des poids et biais :

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

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

4. Mise à jour des paramètres :

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

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.

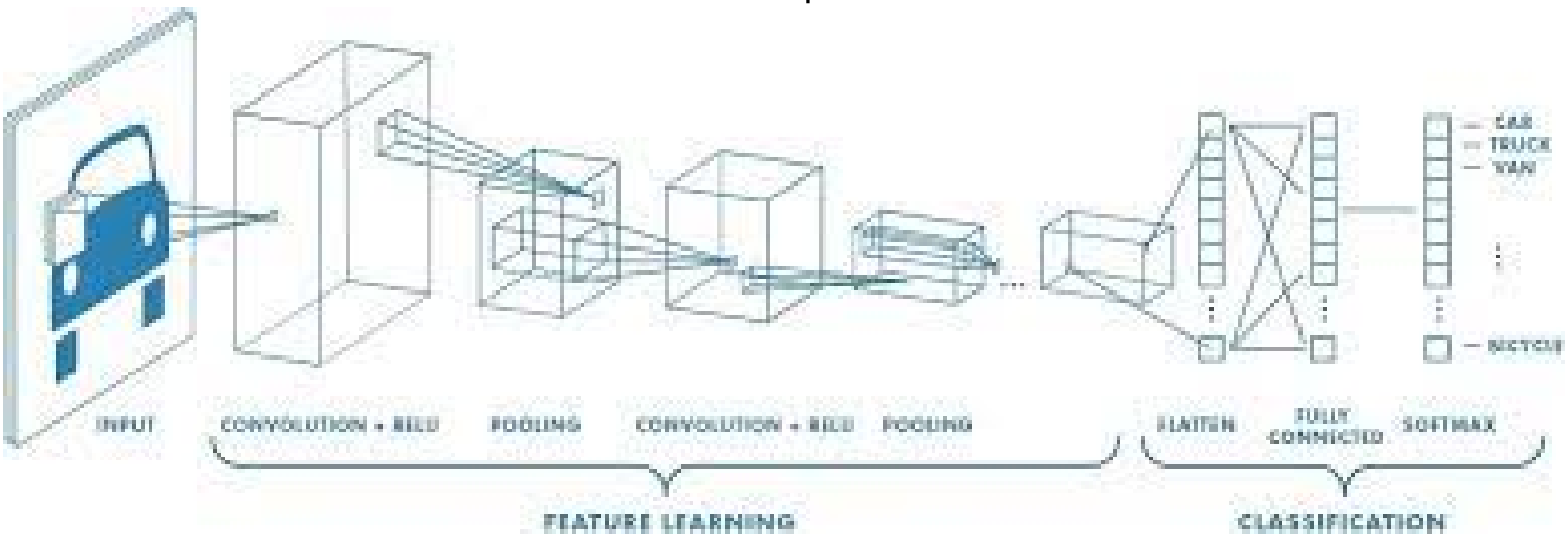
Architecture basique d'un CNN

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

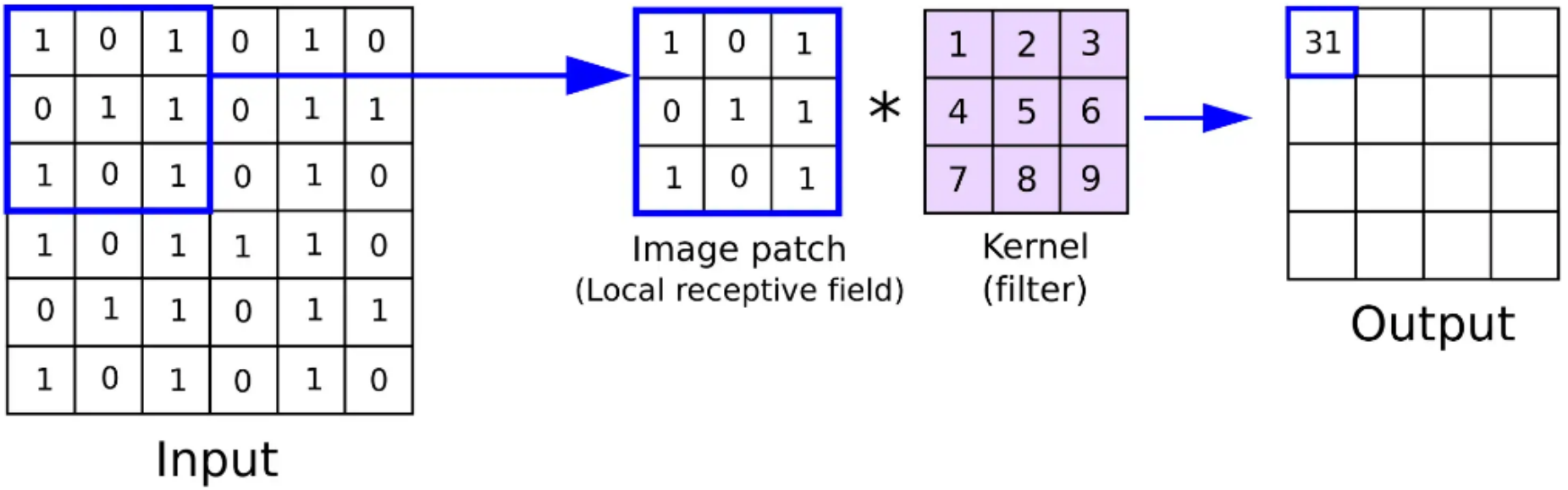
1. **Couche de convolution :** Applique des filtres pour extraire des caractéristiques de bas niveau comme les bords et les textures.
2. **Couche de pooling (ou Sous-échantillonnage) :** Réduit la dimensionnalité de chaque carte de caractéristiques tout en préservant les informations les plus importantes.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

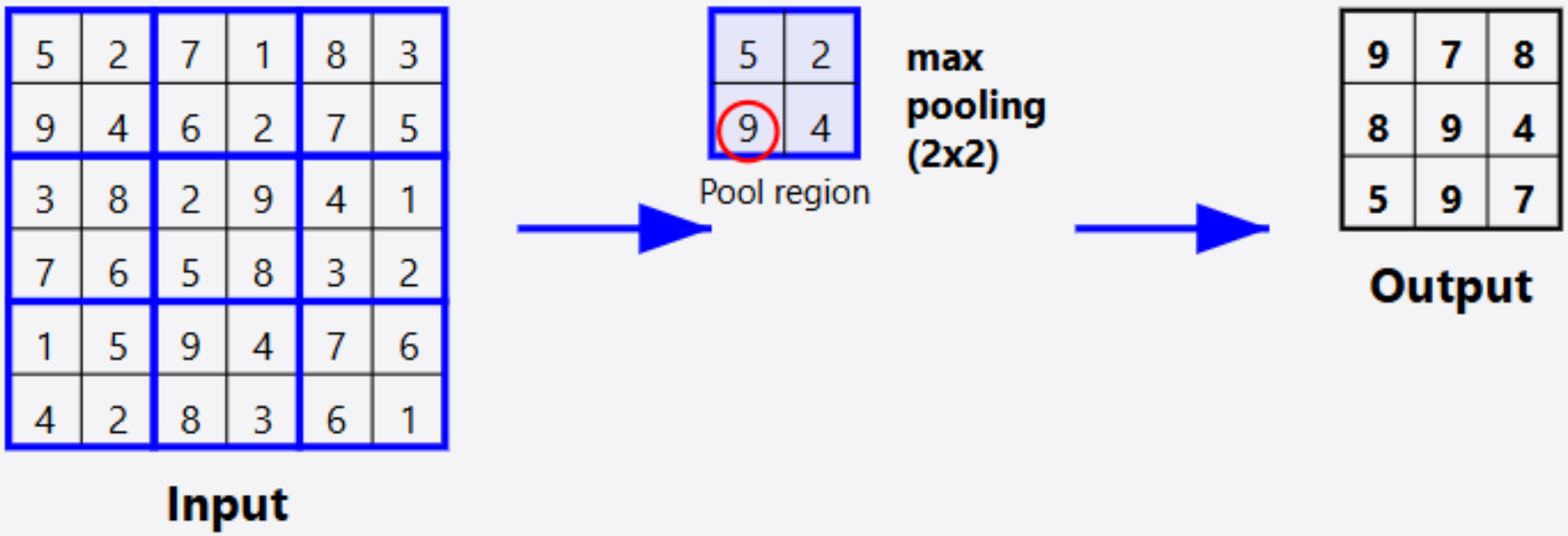
- **Max pooling** : conserve la valeur maximale dans chaque région du filtre.
- **Average pooling** : calcule la moyenne des valeurs dans chaque région.

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 $\rightarrow$ moins de mémoire et de calculs - Maintient les caractéristiques les plus importantes - Supprime des détails moins pertinents, favorisant l'invariance spatiale

Fonctionnement du pooling



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.

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 :

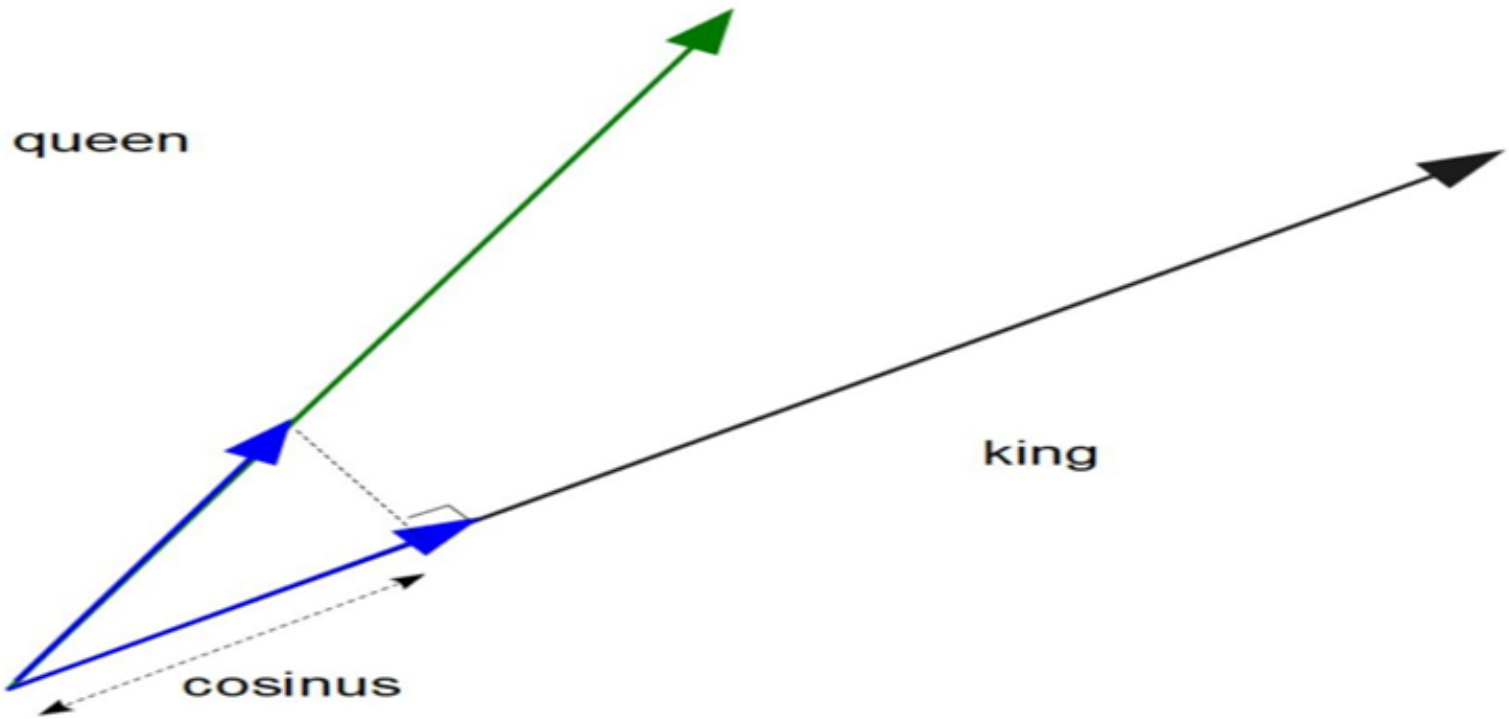
    Véhicule = [0, 0, 1, 0, 0, 0, 0, 0, 0,. . . ,0 ,0]
    Voiture = [0, 0, 0, 0, 0, 1, 0, 0, 0,. . . ,0 ,0]

    Cette representation 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 cosine).



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.

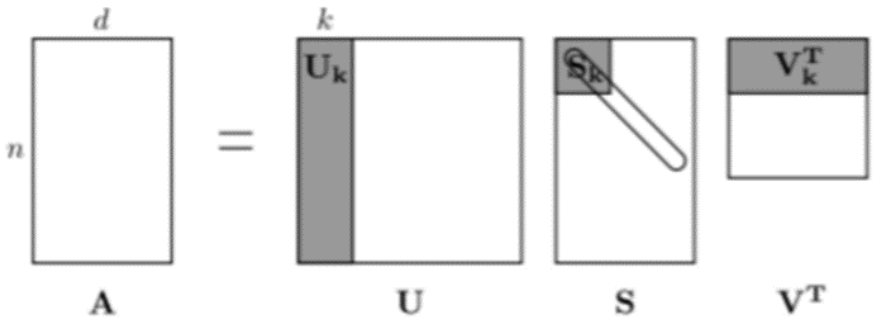
	cnn	in	crop	...
cnn	0	2	1	1	1	2	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1
in	2	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1
crop	1	1	0	1	1	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0
...	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	0	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0
	2	1	0	0	1	0	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1
	1	0	0	0	1	1	0	1	1	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
	1	0	0	0	1	2	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	1	1	1	0	1	2	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1
	0	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	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	1	0	0	0	1	1	1	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	1	0	1	1	0	0	1	1	0	0	0	2	0	1	1	0	0	0	0	0
	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	1	0	0	0	0	0
	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0
	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1
	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1
	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	1
	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	1
	1	1	0	0	0	1	0	0	0	1	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 $A \in \mathbb{R}^{n \times d}$

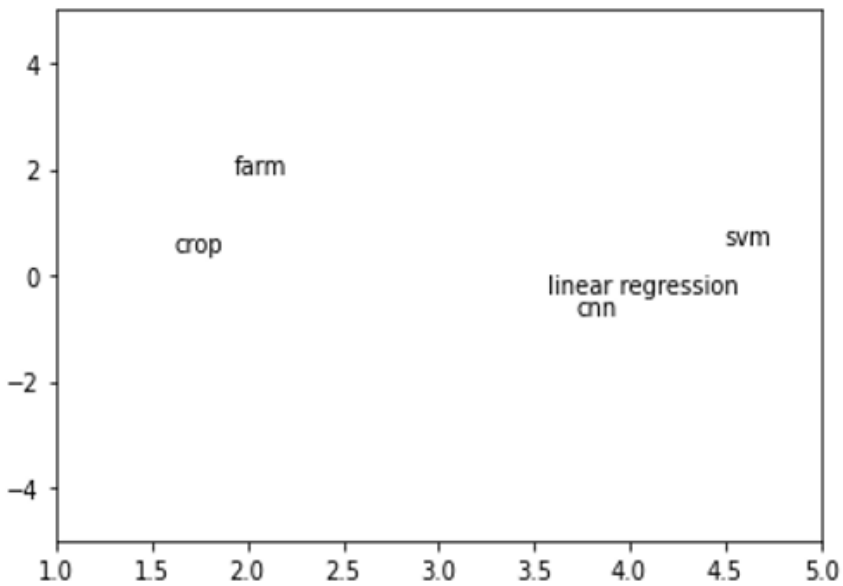
$$A = UDV^T \quad \text{où} \quad U \in \mathbb{R}^{n \times r}, D \in \mathbb{R}^{r \times r}, V \in \mathbb{R}^{d \times r}.$$



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

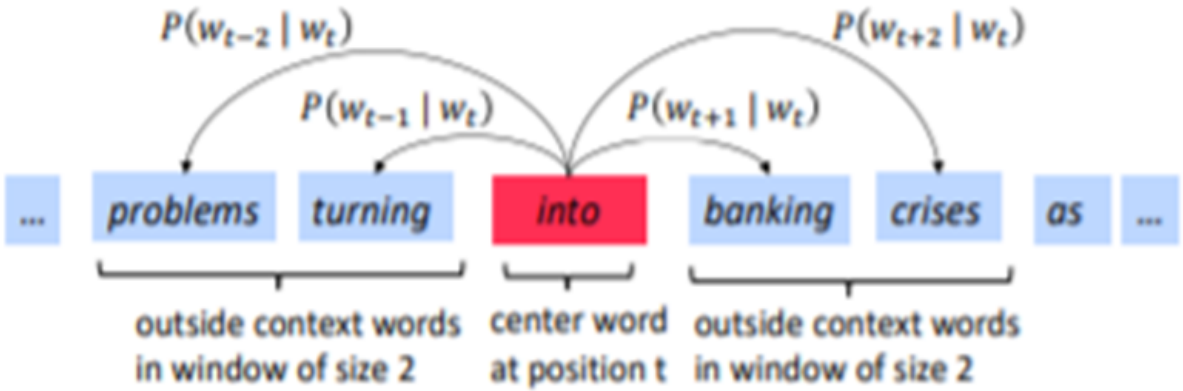
Vsiualisation des représentations distribuées

```
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  ],
     [ 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]]
```

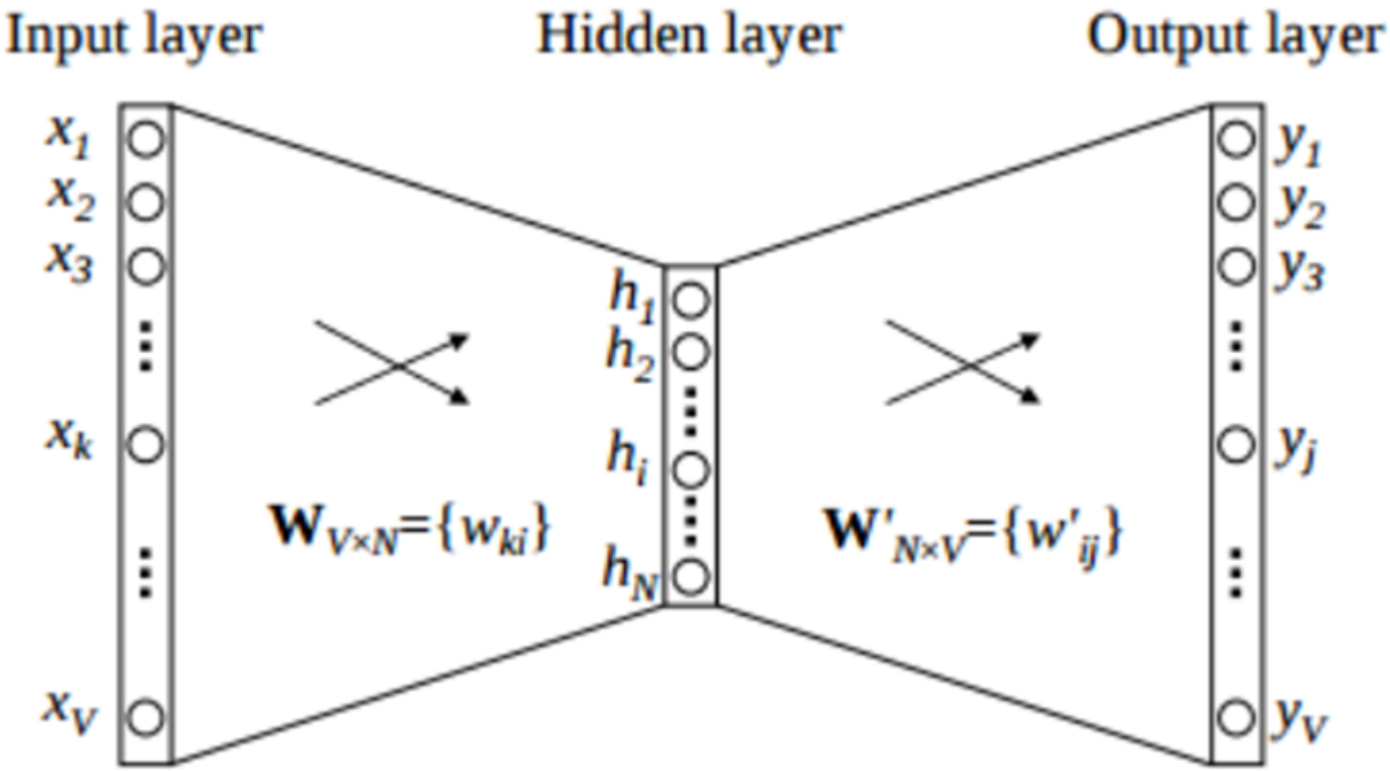


Word2Vec : Principes

- **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 :**

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

- **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 :**

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

- **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.

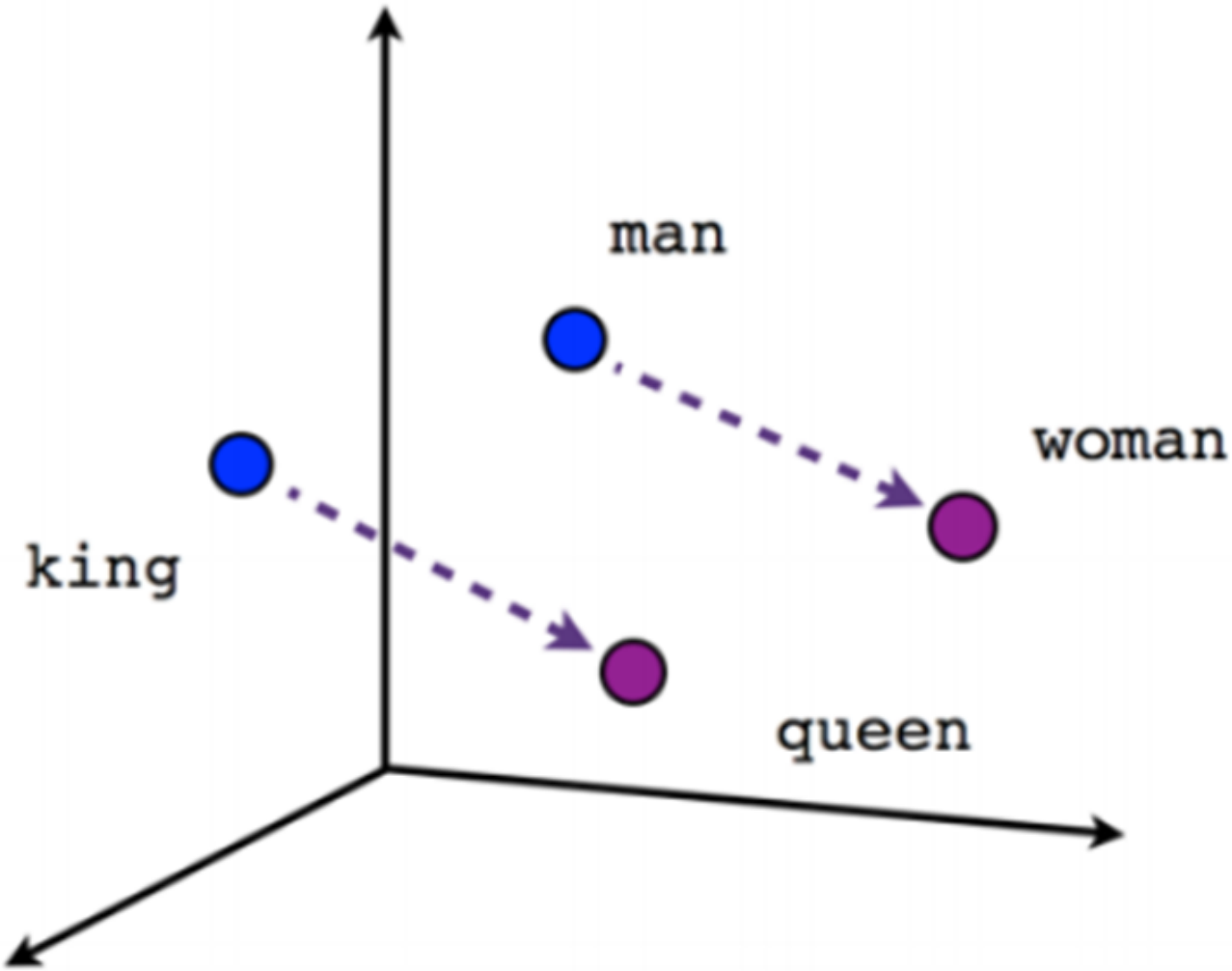
Negative sampling : Formulation mathématique

- **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 :**

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

  - 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 répsentations 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.

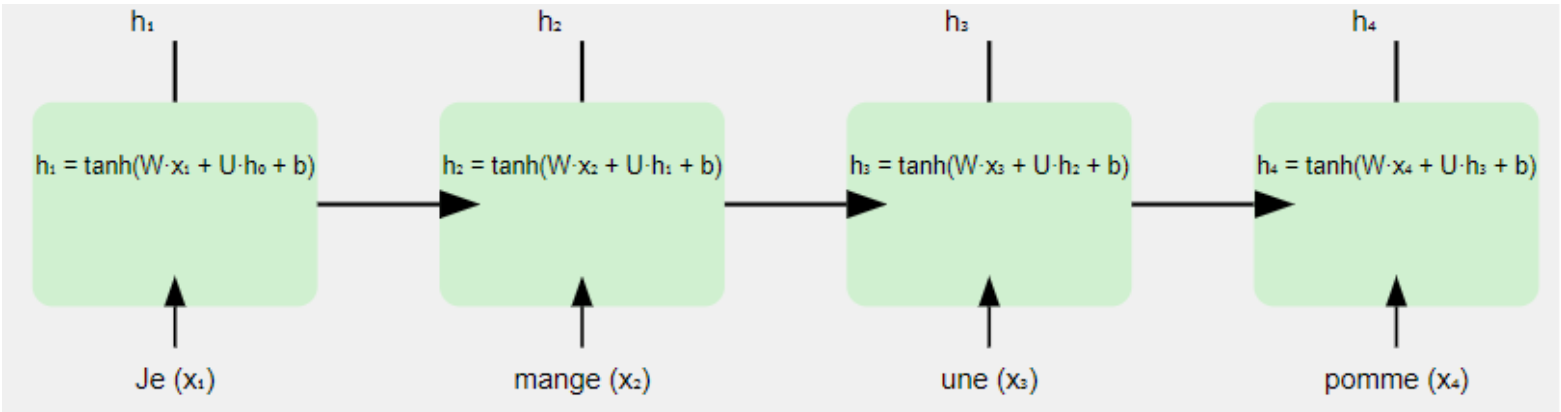
Caractéristiques:

- **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.

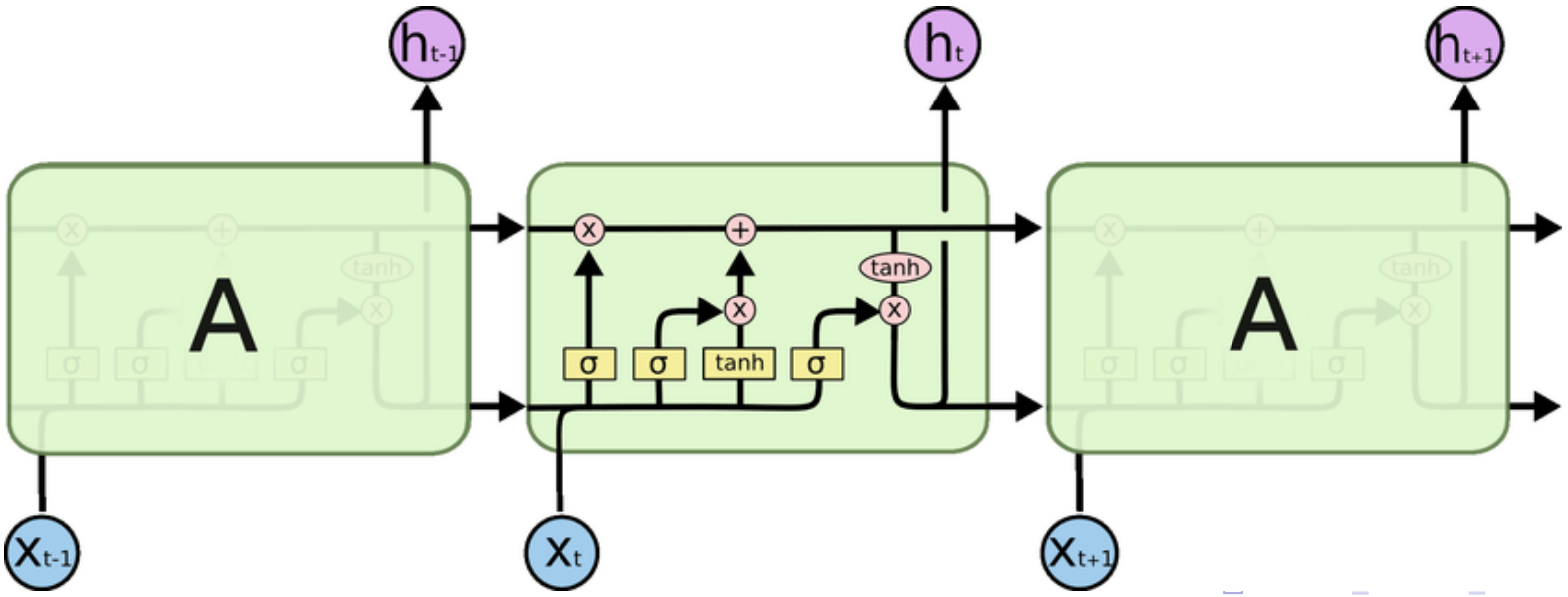
Solutions au problème de disparition du gradient :
Pour atténuer le problème de disparition du gradient, différentes solutions ont été proposées :

- 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.

Merci de votre attention

redha_moulla@yahoo.fr