Hes·so VALAIS WALLIS
Haute Ecole de Gestion
Hochschule für Wirtschaft

Filière Informatique de gestion
Studiengang Wirtschaftsinformatik

Initiez-vous au Deep Learning

LI : 2ème

Étudiant : Zotrim Uka

Compétence : B4, M7, M8, P12

HES-SO Valais-Wallis • rte de la Plaine 2 • CP 80 • 3960 Sierre
+41 58 606 93 11 • info.informatique@hevs.ch • www.hevs.ch/ig

Table des matières

Table des illustrations

Préambule

Ce résumé, basé sur le cours Initiez-vous au Deep Learning d'OpenClassrooms, a été conçu pour fournir aux professionnels une vue d'ensemble des fondamentaux et des meilleures pratiques en apprentissage profond. Les informations présentées ici proviennent directement des principes modernes de deep learning et visent à offrir une compréhension approfondie des architectures neuronales, des techniques d'entraînement, et des applications pratiques. Ce document est destiné à un usage interne et a pour objectif de fournir une synthèse claire et pratique des sujets abordés, permettant aux ingénieurs, data scientists et responsables techniques de comprendre les enjeux et les avantages des réseaux de neurones dans un contexte professionnel.

1. Introduction

AlphaZero, une intelligence artificielle de DeepMind, filiale de Google, est capable de surpasser les meilleurs joueurs de Go au monde. L'intelligence artificielle s'insère déjà dans la vie quotidienne, que ce soit pour dicter une note, utiliser des assistants vocaux ou la reconnaissance faciale sur un appareil photo.

Ce cours introduit les réseaux de neurones artificiels, qui sont à l'origine de ces avancées technologiques. Il explore le concept de l'apprentissage profond (Deep Learning) et présente divers types de réseaux de neurones, chacun adapté à des tâches spécifiques comme le traitement de l'image, du son ou du texte.

En fin de parcours, les participants sauront expliquer les bases des réseaux de neurones artificiels, créer un modèle de Deep Learning et ajuster ses paramètres pour en améliorer la performance.

2. Découvrez le neurone formel

Le neurone formel, ou perceptron, est présenté comme un modèle mathématique inspiré du neurone biologique, le composant de base des réseaux de neurones artificiels. Ce dernier imite certains aspects de l'activité biologique, mais de manière simplifiée.

Un neurone biologique comprend un corps cellulaire, des dendrites (qui reçoivent les signaux extérieurs) et un axone (qui transmet les signaux vers d'autres neurones). Les dendrites captent l'influx nerveux des stimuli, et lorsqu'une certaine excitation est atteinte, un signal est émis le long de l'axone, reliant ainsi les neurones à travers des synapses.

Pour modéliser ce processus, le perceptron utilise des **entrées** (notées x) sous formes de vecteurs, représentant les dendrites, une **sortie** (y), représentant l'axone, et des **paramètres** (w et b) influençant le calcul. La sortie du neurone se base sur la somme des entrées pondérées par des **poids** (w), additionnée à un **biais** (b), le tout passant par une fonction de **transfert** f, souvent non linéaire.

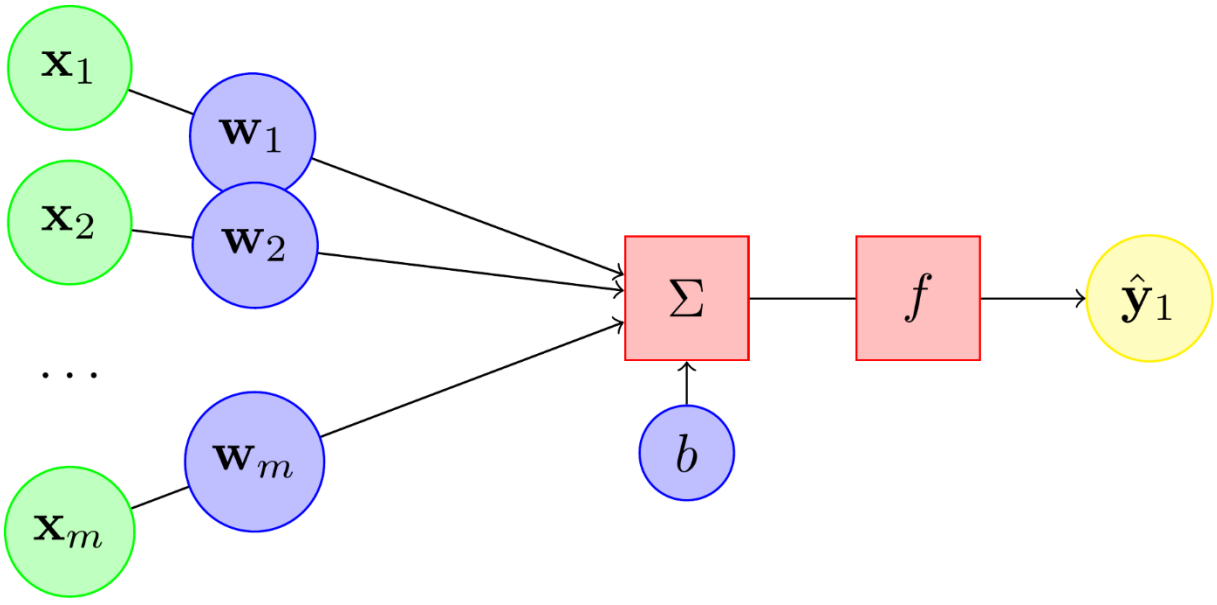


Figure 1 : Un neurone formel

L'équation formelle du neurone est :

$$\hat{\mathbf{y}} = f(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

Les valeurs absolues des entrées influencent l'amplitude de la sortie, ce qui diffère de la modulation de fréquence dans un neurone biologique.

En guise d'exemple, on examine un neurone formel appliqué à un problème de classification binaire. Les valeurs des paramètres (w=[0.985,2.186] et b=−0.522) permettent de séparer deux ensembles de points sur un graphique. L'apprentissage par descente de gradient optimise les paramètres w et b pour minimiser une fonction de perte, mesurant l'écart entre la prédiction y^ et la valeur réelle y. Pour la classification, on utilise la log-vraisemblance négative :

$$L(\hat{y}, y) = -(y.log(\hat{y}) + (1-y).log(1-\hat{y}))$$

À chaque étape, les paramètres sont ajustés dans la direction qui réduit cette perte, en se basant sur le gradient de la fonction. Cette adaptation continue des poids permet de déplacer la fonction de décision, séparant progressivement les données en deux classes, avec la zone **cyan** pour les valeurs négatives et la **magenta** pour les positives.

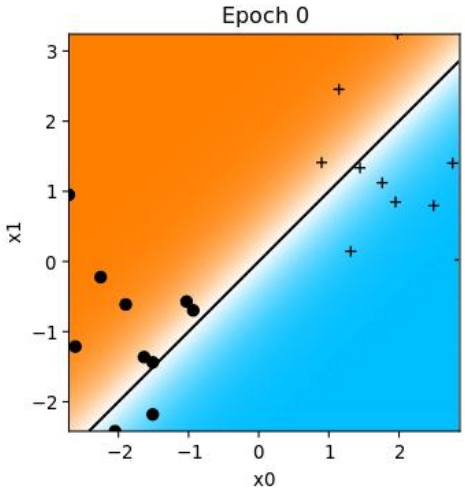


Figure 2 : Évolution de la fonction de décision à chaque étape de la descente

3. Explorez les réseaux de neurones en couches

Un réseau de neurones multicouche, ou perceptron multicouche, permet de résoudre des problèmes plus complexes qu'un simple neurone formel.

Un seul neurone peut classer des données lorsqu'elles sont séparables par une droite. Cependant, il échoue pour des problèmes plus complexes, comme des exemples non linéairement séparables où une simple droite ne peut pas diviser les données en deux groupes.

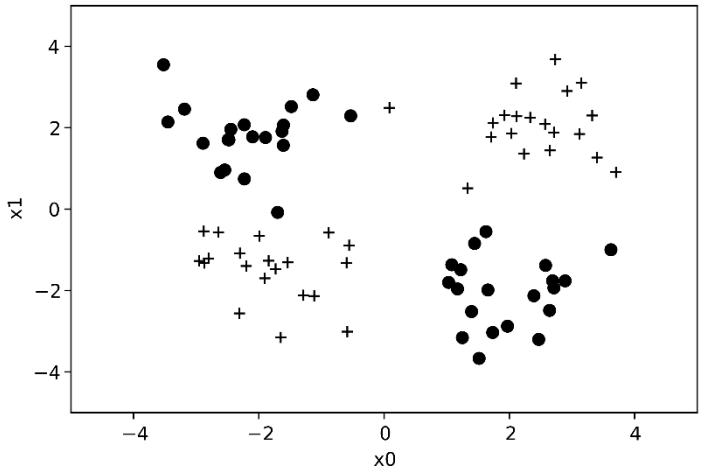


Figure 3 : Un exemple non linéairement séparable

Pour dépasser ces limites, plusieurs neurones peuvent être connectés ensemble, formant un réseau où la sortie d'un neurone devient l'entrée d'un autre.

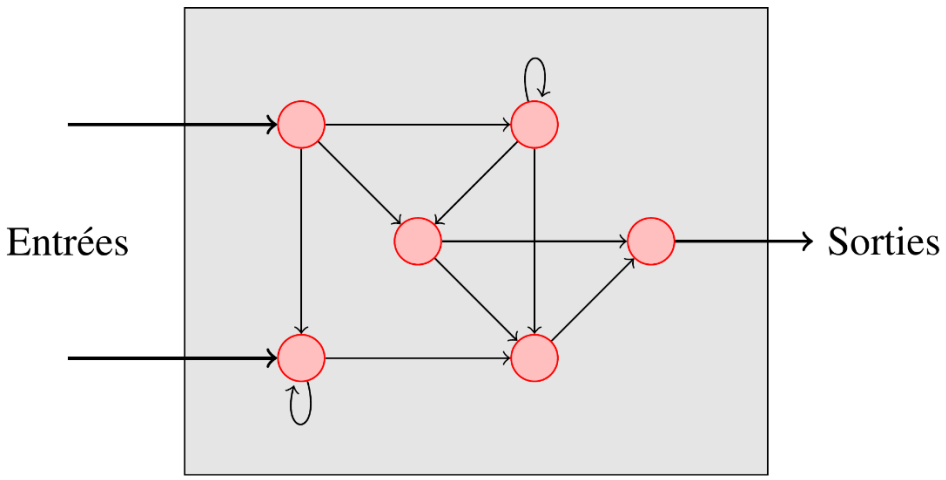


Figure 4 : Une topologie quelconque

Dans ces réseaux, les neurones sont souvent organisés en couches : chaque couche reçoit les données de la couche précédente et envoie ses résultats à la suivante. Ce type de réseau, adapté aux données de taille fixe comme les images, est appelé réseau multicouche ou perceptron multicouche (MLP). Pour les données de taille variable, comme des séquences de texte ou d'audio, des réseaux récurrents sont utilisés, bien qu'ils soient plus complexes à entraîner.

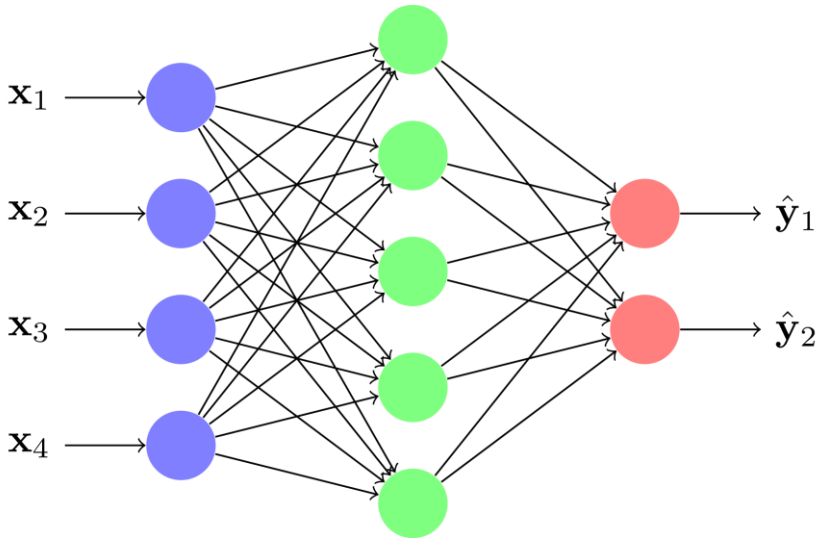


Figure 5 : Un réseau multicouche

Pour entraîner un réseau multicouche, on utilise la descente de gradient pour ajuster les poids et minimiser l'erreur du réseau. Ce processus, appelé rétropropagation, consiste à calculer l'erreur à partir de la couche de sortie et à la propager couche par couche jusqu'à la première, en ajustant les poids de chaque couche en fonction du gradient, ce qui permet au réseau d'apprendre des représentations plus complexes pour les données.

Un réseau multicouche avec deux couches, par exemple avec trois neurones dans la première couche et deux dans la seconde, permet de résoudre un problème non linéaire. Grâce à cet apprentissage, le réseau peut progressivement diviser les exemples en deux groupes, illustré par l'évolution de la courbe de séparation au fil de l'apprentissage.

Un réseau peut être organisé en une seule couche ou en plusieurs couches.

Dans un réseau à **une seule couche**, plusieurs neurones fonctionnent en parallèle pour traiter les données. Chaque neurone de cette couche prend les mêmes entrées et produit

une sortie, puis ces sorties sont combinées pour former une seule réponse. Cela reste cependant limité pour résoudre des problèmes très complexes, car une seule couche n'est pas suffisante pour capturer toutes les relations possibles dans des données complexes.

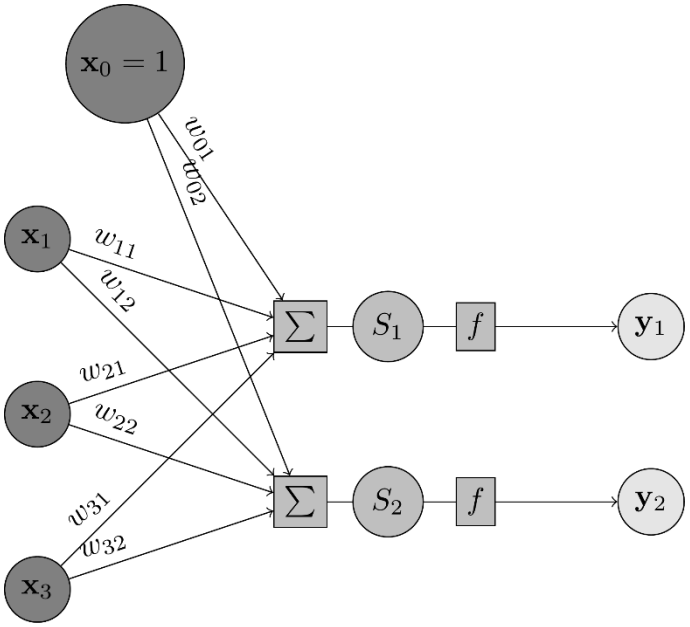


Figure 6 : Schéma d'une couche isolée

En passant à une organisation avec **plusieurs couches**, on obtient ce qu'on appelle un perceptron multicouche (MLP). Ici, chaque couche intermédiaire apprend des représentations plus sophistiquées des données en passant les informations traitées de la première couche à la seconde, et ainsi de suite. Chaque couche extrait des caractéristiques de plus en plus complexes, ce qui permet de résoudre des problèmes non linéaires.

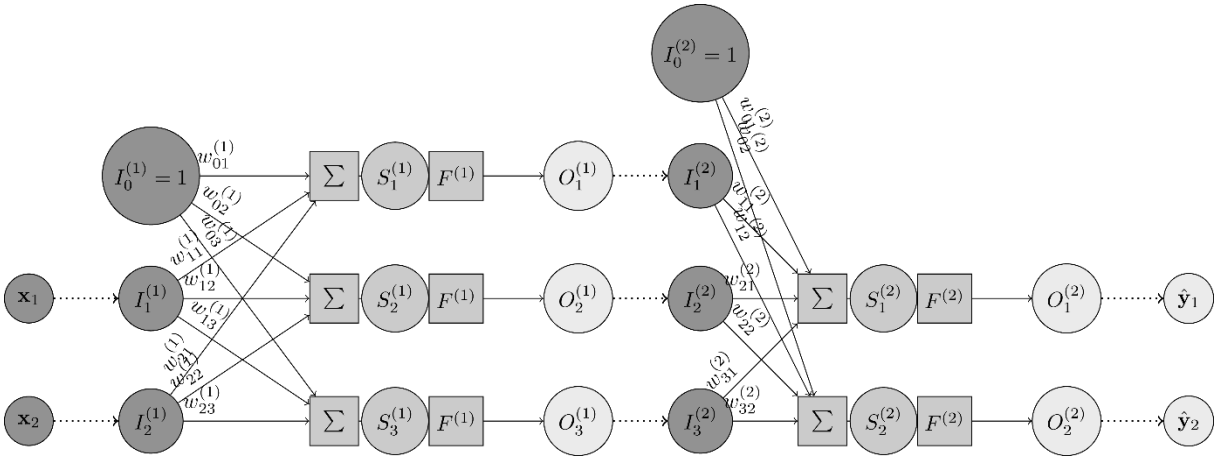
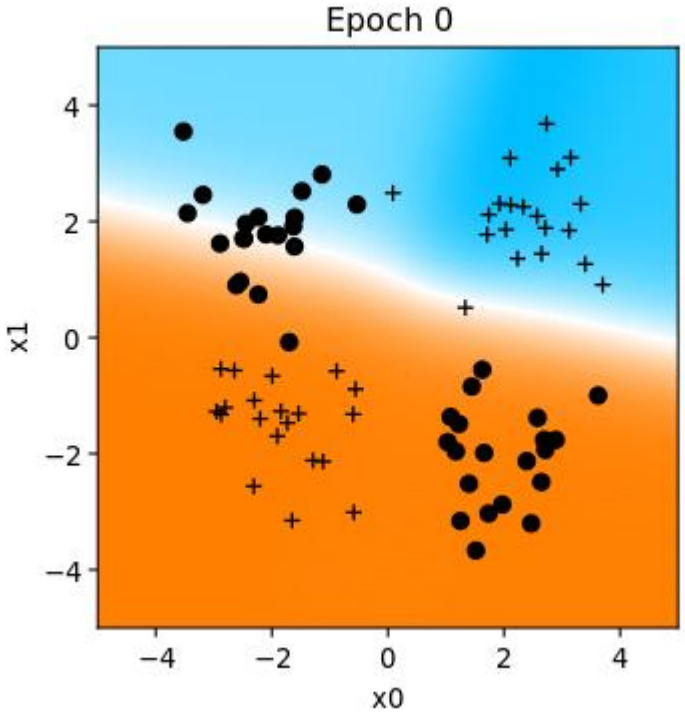


Figure 7 : Un réseau complet à deux couches

L'apprentissage d'un réseau multicouche se fait via la **rétropropagation du gradient**. Cette méthode permet d'ajuster les paramètres du réseau (les poids et les biais) pour réduire progressivement l'erreur. La rétropropagation commence par la couche de sortie : on calcule l'erreur par rapport aux valeurs attendues, puis cette erreur est propagée en sens inverse dans le réseau, couche par couche, jusqu'à la couche d'entrée. Chaque couche ajuste alors ses poids en fonction du gradient, c'est-à-dire la direction et la mesure de changement pour réduire l'erreur. En répétant ce processus, le réseau apprend à donner des réponses de plus en plus précises.



4. Initiez-vous aux autoencodeurs

Les autoencodeurs (AE) sont des réseaux de neurones utilisés en apprentissage non supervisé, où l'on dispose uniquement des caractéristiques des exemples (notées x) sans étiquettes (notées y). Leur but est de réduire la dimension des données en les compressant dans une représentation plus petite, tout en conservant l'information essentielle.

L'architecture d'un autoencodeur, appelée architecture diabolo, est composée de deux parties : l'encodeur et le décodeur. L'encodeur compresse les données de grande dimension en une représentation de plus petite dimension, tandis que le décodeur décompresse ces données pour tenter de recréer les données d'origine. La valeur centrale de l'autoencodeur est appelée le code et représente une version compressée de l'entrée. L'apprentissage se fait par rétropropagation du gradient en utilisant les données d'entrée comme cibles.

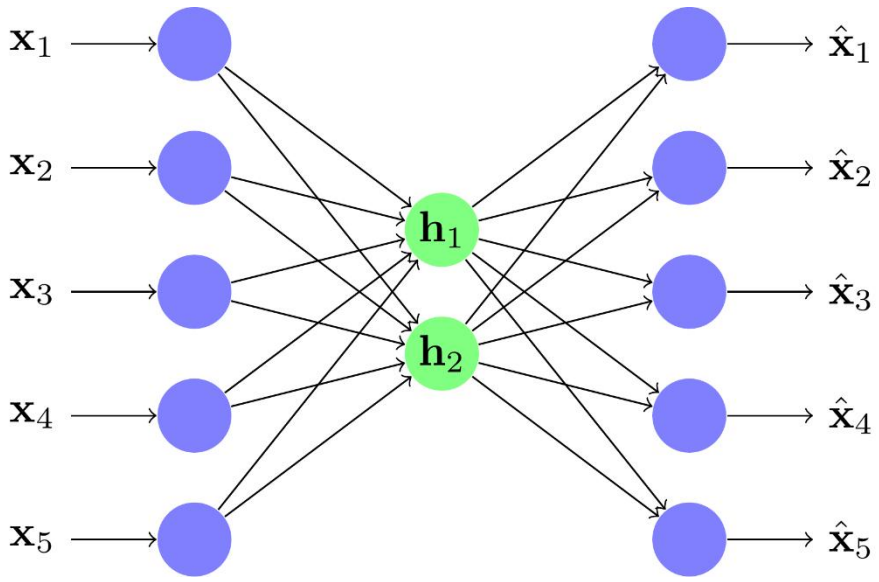


Figure 8 : Architecture diabolo simple, avec une couche pour l'encodeur et une couche pour le décodeur

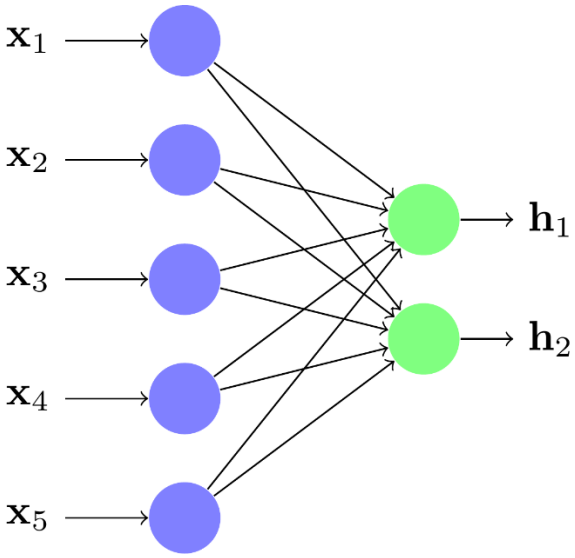


Figure 9 : La partie Encodeur d'un réseau diabolo

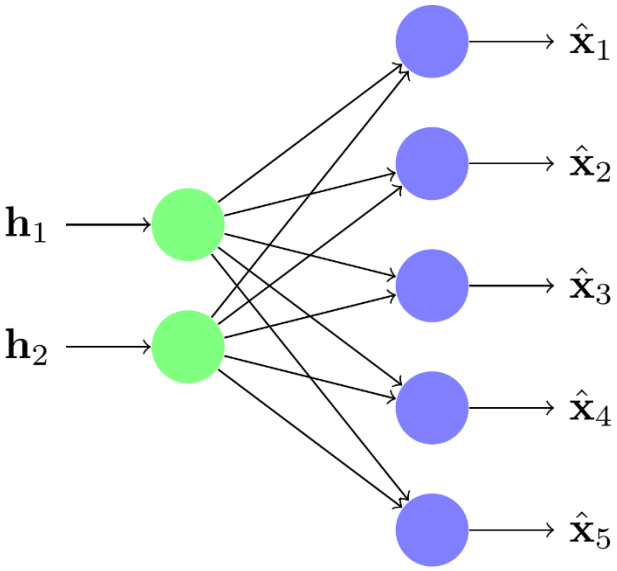


Figure 10 : La partie Décodeur d'un réseau diabolo

Il existe deux types d'autoencodeurs. Les under-complete ont un code plus petit que les entrées, permettant une réduction de dimension pour extraire les caractéristiques essentielles des données. Les over-complete, avec un code de dimension supérieure aux entrées, permettent une meilleure représentation des données pour des traitements ultérieurs. Pour éviter que le réseau ne se contente de recopier les données, on déconnecte des neurones de manière aléatoire durant l'apprentissage, ce qui empêche les solutions triviales.

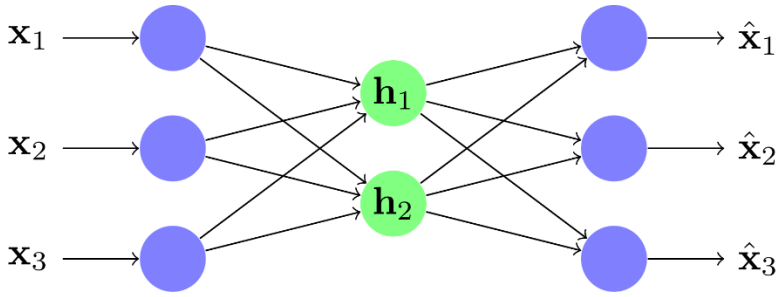


Figure 11 :Un exemple de réseau under-complete

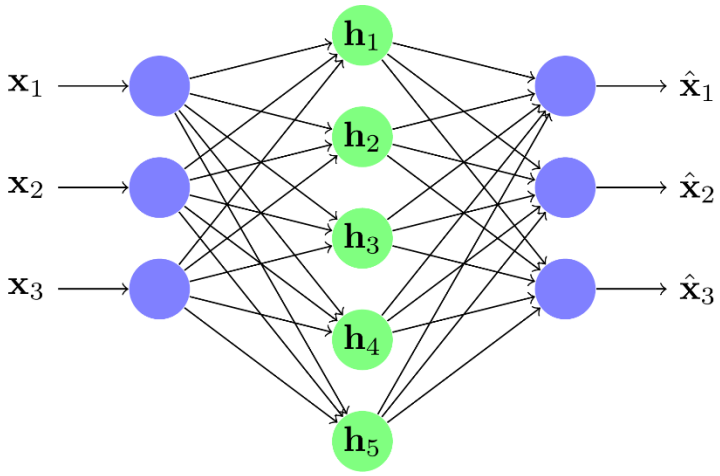


Figure 12 : Un exemple de réseau over-complete

Les autoencodeurs under-complete sont utilisés pour des applications de débruitage, apprenant à ignorer le bruit dans les données en ajoutant des perturbations à l'entrée, une méthode qui agit comme une régularisation. Ils servent également à l'extraction automatique de caractéristiques : en analysant les poids de l'encodeur, un autoencodeur apprend des filtres caractéristiques des données.

Un réseau profond peut être construit en empilant plusieurs autoencodeurs, où chaque autoencodeur est préentraîné de manière non supervisée sur les sorties de l'encodeur précédent. Ensuite, un apprentissage supervisé classique est utilisé pour ajuster toutes les couches, une étape appelée fine-tuning.

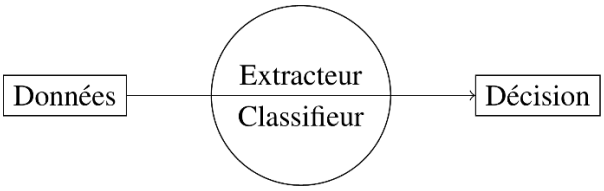


Figure 13 : Apprentissage profond

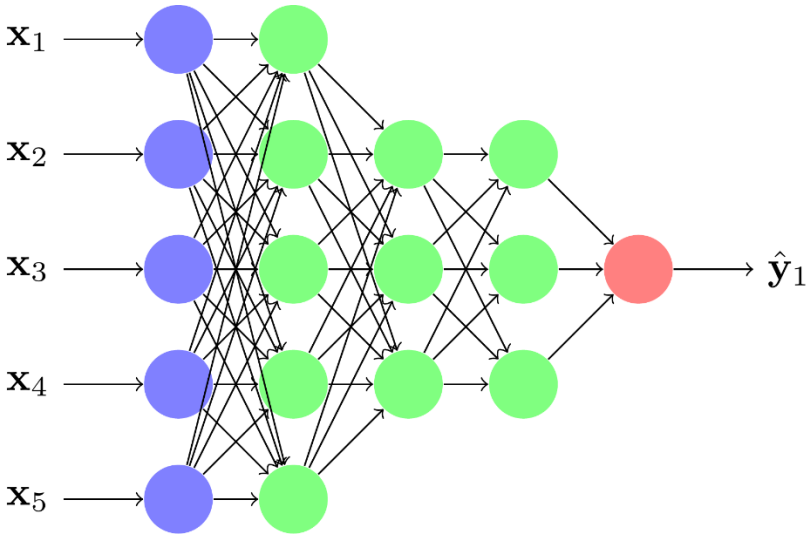


Figure 14 : Apprentissage final, ou fine-tuning

Les réseaux profonds font face au problème du gradient évanescent. Plus un réseau contient de couches, plus les gradients deviennent faibles dans les premières couches, rendant l'apprentissage difficile. Ce phénomène, surtout problématique pour des fonctions de transfert comme tanh, pose un défi majeur dans l'entraînement des réseaux profonds, notamment pour le traitement des images.

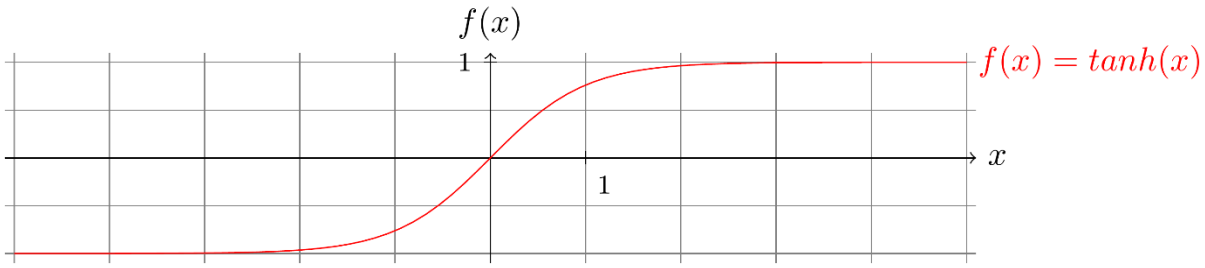


Figure 15 : La fonction de transfert tanh

5. Construisez des réseaux profonds grâce aux couches convolutionnelles

Les réseaux profonds rencontrent le problème du gradient évanescent, où les gradients deviennent trop faibles pour atteindre les premières couches, rendant l'apprentissage difficile. Dans un réseau de neurones classique appliqué aux images, chaque pixel est relié à toutes les unités de la première couche, formant une couche complètement connectée. Cette configuration génère un nombre énorme de paramètres, difficiles à optimiser.

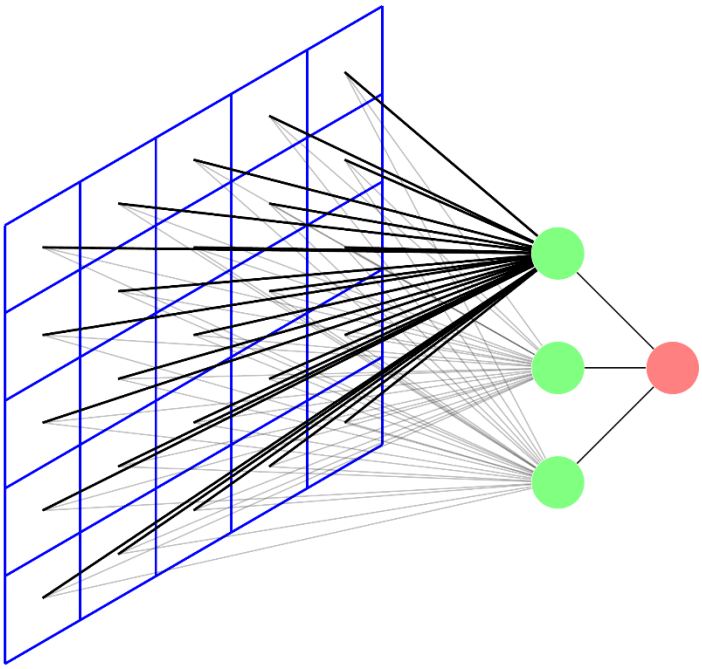


Figure 16 : Un réseau de neurones naïf sur une image

Pour résoudre cela, il est possible de tenir compte de la structure de l'image en utilisant des **couches convolutionnelles**. Un filtre de convolution balaie l'image pour extraire des caractéristiques spécifiques, comme les contours, en se concentrant uniquement sur certains pixels. Contrairement aux filtres fixes, les couches convolutionnelles apprennent les valeurs optimales des filtres à partir des données, ce qui réduit le nombre de paramètres et atténue le problème du gradient évanescent.

Un réseau convolutionnel comprend généralement des couches convolutionnelles empilées, des couches de **pooling** et des couches de **batch normalization**. Les couches de pooling réduisent la taille des cartes de caractéristiques en ne gardant que les valeurs maximales sur une petite région, ce qui conserve les caractéristiques importantes tout en simplifiant le réseau. Les couches de batch normalization stabilisent les données et réduisent le phénomène de saturation, rendant le réseau plus robuste à l'apprentissage. En fin de réseau, des couches complètement connectées sont souvent ajoutées pour finaliser la tâche de classification.

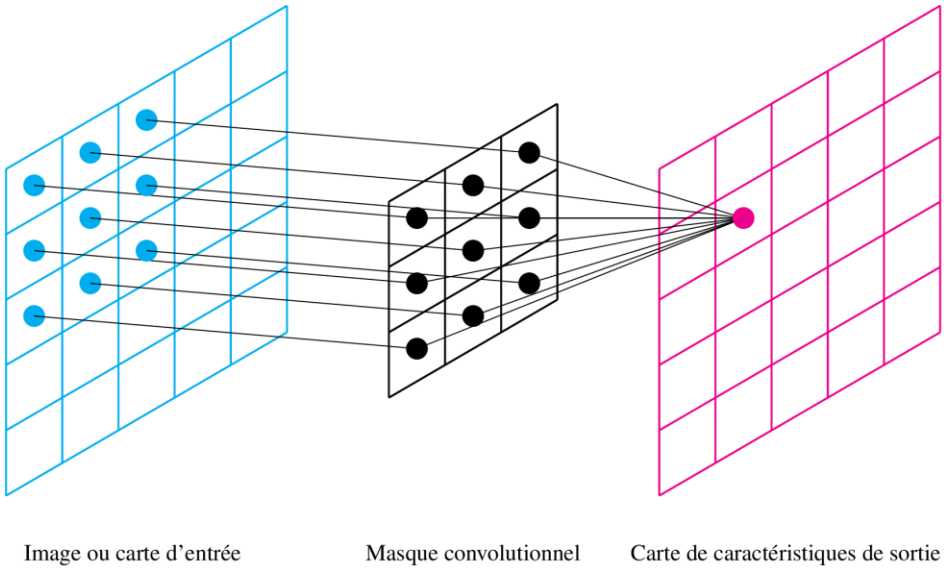


Figure 17 : Une couche convolutionnelle en action

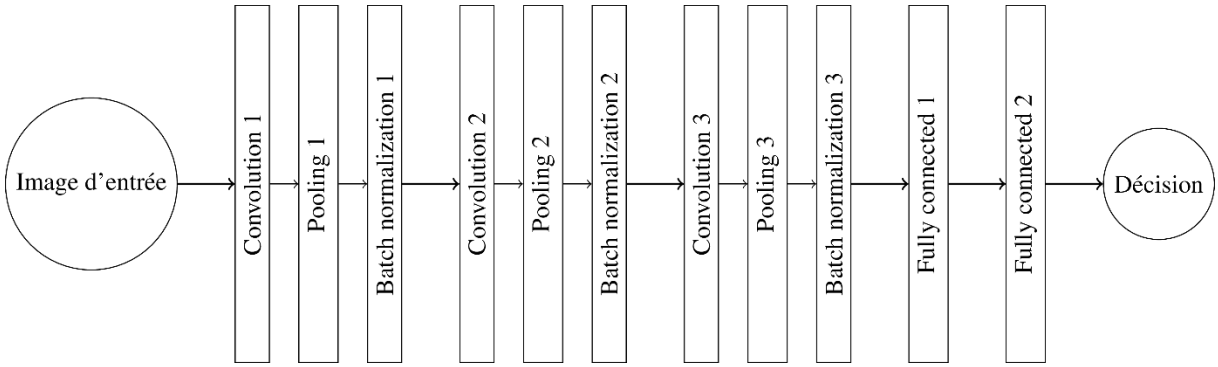


Figure 18 : Un réseau profond complet

Les fonctions d'activation comme ReLU ou Softplus sont couramment utilisées dans les couches convolutionnelles, car elles ne saturent pas, maintenant un gradient suffisamment grand pour faciliter l'apprentissage. De multiples filtres appliqués en parallèle génèrent plusieurs cartes de caractéristiques en sortie, chaque filtre apprenant à extraire une caractéristique différente de l'image.

Des réseaux convolutionnels populaires, tels qu'AlexNet et VGG, sont préentraînés sur de vastes bases de données comme ImageNet et peuvent être utilisés en **transfer learning**. Dans ce cadre, les premières couches du réseau préentraîné sont conservées pour une nouvelle tâche, et de nouvelles couches sont ajoutées pour adapter le réseau à cette tâche. Ce transfert d'apprentissage permet de réutiliser les connaissances acquises pour des

applications spécifiques, même avec peu de nouvelles données, en réentraînant l'ensemble du réseau pour affiner les résultats.

6. Construisez des modèles génératifs grâce aux réseaux de neurones

Les modèles génératifs permettent de créer de nouvelles données synthétiques en apprenant la distribution d'un ensemble de données d'origine. Les modèles profonds génératifs sans vraisemblance, comme les GANs et les VAE, cherchent à capturer cette distribution pour générer des exemples réalistes sans évaluer précisément leur vraisemblance.

Un **autoencodeur** peut être utilisé pour générer des données en conservant uniquement sa partie décodeur. On tire un code aléatoire, qui est ensuite passé dans le décodeur pour produire une donnée synthétique. Toutefois, sans connaître la distribution de ce code, les données générées risquent de manquer de réalisme. Les **VAE (Variational Autoencoders)** pallient cette limite en contraignant la distribution des codes latents pour mieux guider la génération.

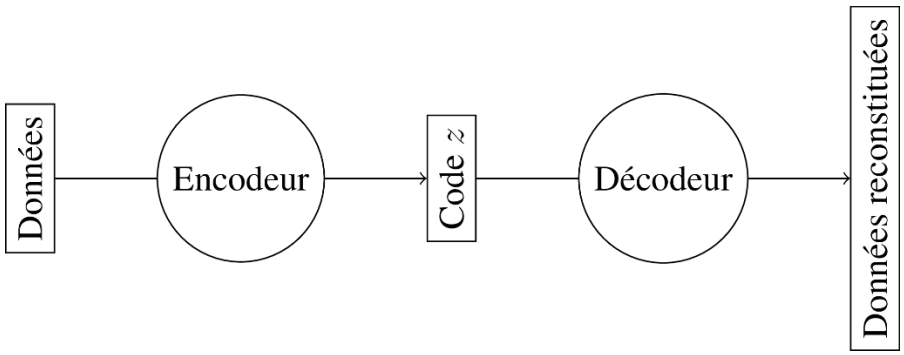


Figure 19 : Schéma général d'un autoencodeur

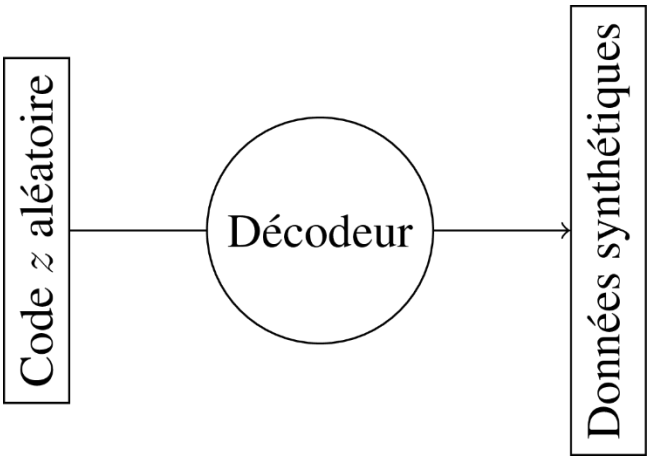


Figure 20 : La partie décodeur d'un autoencodeur

Les **GANs (Generative Adversarial Networks)** représentent une autre approche. Composés de deux réseaux, le **générateur** et le **discriminateur**, ils s'affrontent dans un apprentissage compétitif : le générateur crée des exemples synthétiques pour tromper le discriminateur, tandis que le discriminateur tente de distinguer les exemples réels des faux. Cette opposition conduit les GANs à générer des exemples de plus en plus réalistes. Leur fonction objective cherche un équilibre où les deux réseaux atteignent une sorte de compromis (équilibre de Nash), souvent ajustée pour plus de stabilité lors de l'apprentissage.

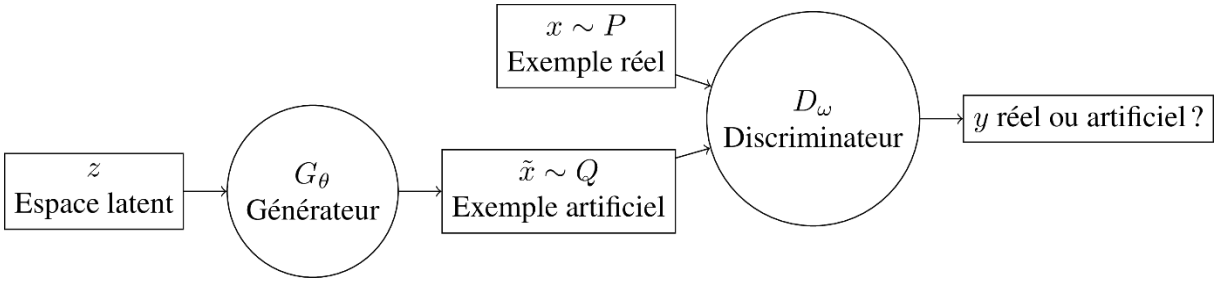


Figure 21 : Schéma général d'un GAN

Les GANs sont utilisés dans de nombreuses applications, telles que la colorisation d'images, la segmentation, la génération d'images à partir de cartes, le remplissage de zones manquantes (inpainting) et la prédiction de trames vidéo. En combinant des architectures GAN avec des autoencodeurs, il est possible de manipuler l'espace latent et d'obtenir des résultats impressionnants, comme le morphing d'images. Des avancées récentes, comme les réseaux GAN à croissance progressive, améliorent la qualité et la stabilité des données générées.

7. Initiez-vous aux problématiques liées au traitement de séquences

Les réseaux de neurones **feedforward**, comme les perceptrons multicouches (MLP) et les réseaux convolutionnels (CNN), sont très performants dans de nombreux domaines, surtout en traitement d'images. Leur architecture fait transiter l'information des entrées aux sorties et permet l'extraction automatique de caractéristiques grâce à des algorithmes efficaces comme la rétropropagation du gradient. Cependant, ils présentent des limites importantes, notamment face aux données de tailles variables.

Ces réseaux imposent que toutes les entrées et sorties aient des dimensions fixes, ce qui n'est pas réaliste dans des contextes comme la reconnaissance vocale, l'analyse de texte ou la traduction automatique, où la longueur des données peut varier. Par exemple, dans la traduction, une phrase source et sa traduction peuvent ne pas avoir le même nombre de mots. Idéalement, un modèle capable de gérer des données de longueur variable en entrée et en sortie serait nécessaire.

Une solution consiste à utiliser une **fenêtre glissante** qui parcourt les données de manière séquentielle pour les traiter morceau par morceau. Cette approche fonctionne pour des tâches comme la reconnaissance de caractères manuscrits, où la fenêtre se déplace sur chaque lettre et les classe séparément. Cependant, elle pose un problème en cas de dépendances contextuelles, où un élément de la séquence dépend des précédents ou des suivants.

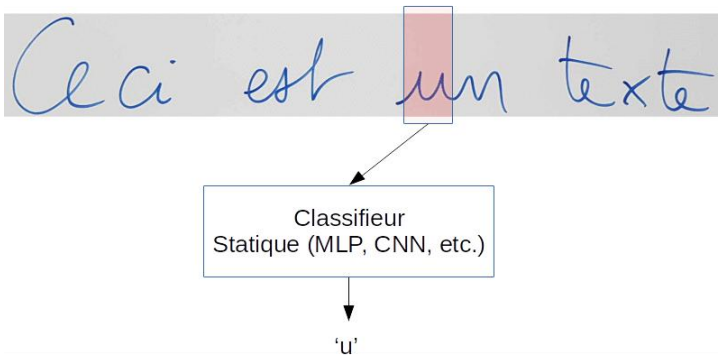


Figure 22 : Fenêtre glissante pour la reconnaissance d'écriture

Par exemple, une phrase comme "Un deux ___ quatre cinq" ou "En me rasant ce matin, je me suis ___" est simple à compléter pour un humain grâce au contexte, mais difficile pour un réseau de neurones qui traite chaque élément indépendamment. La fenêtre glissante seule ne suffit pas, car elle ne permet pas de se souvenir du contexte ou des éléments précédents.

Pour surmonter cette limite, les réseaux de neurones doivent être dotés d'une forme de **mémoire** pour gérer des séquences tout en intégrant le contexte. Les **réseaux de neurones récurrents (RNN)** sont conçus pour cette tâche : ils reprennent l'idée de la fenêtre glissante tout en offrant une mémoire capable de conserver et de traiter l'information sur l'ensemble de la séquence, rendant possible l'analyse des signaux de taille variable avec des dépendances internes.

8. Découvrez le fonctionnement des réseaux de neurones récurrents

Les réseaux de neurones récurrents (RNN) introduisent une mémoire qui permet de traiter des séquences de données de taille variable, utiles pour des tâches où chaque élément dépend du contexte des éléments précédents. En plus de l'astuce de la fenêtre glissante, les RNN utilisent des **connexions récurrentes** pour réinjecter la sortie de la couche dans la couche elle-même au pas de temps suivant, créant ainsi une "mémoire" potentiellement infinie. Cette capacité permet au RNN de se souvenir de ce qu'il a vu à chaque étape de la séquence.

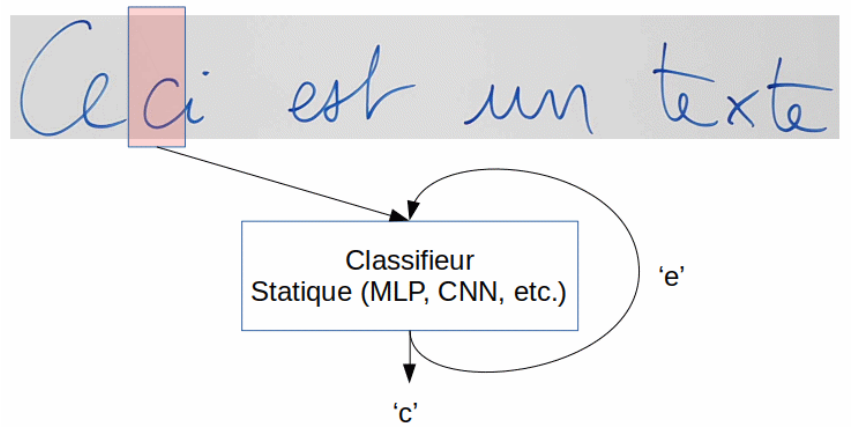


Figure 23 : Le réseau récurrent parcourt le signal de gauche à droite à l'aide d'une fenêtre glissante. Il décide du caractère à reconnaître, en se basant sur le contenu de la fenêtre et sur sa décision sur la fenêtre précédente.

Les réseaux récurrents peuvent être combinés avec des couches denses ou de convolution pour renforcer leurs capacités. Dans cette configuration, les RNN disposent de plusieurs matrices de poids : une pour les connexions d'entrée, une pour les connexions récurrentes, et une pour les connexions avec les couches denses. Une représentation **dépliée** du réseau, montrant le temps, permet de visualiser comment l'information est propagée entre chaque instant de la séquence.

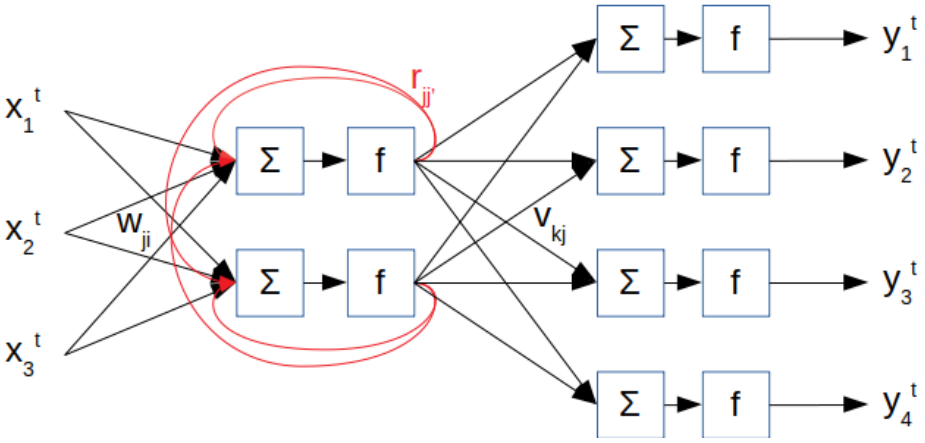


Figure 24 : Réseau de neurones avec une couche récurrente et une couche dense. Les entrées sont indicées i, les neurones récurrents j et les sorties k

Les RNN peuvent être appliqués à plusieurs types de tâches :

1. **Étiquetage de séquences** : le réseau produit une sortie à chaque étape de la séquence d'entrée, comme en reconnaissance de texte ou d'écriture, où chaque caractère est étiqueté au fur et à mesure.

2. **Classification de séquences** : le réseau parcourt toute la séquence d'entrée avant de produire une seule sortie, comme en analyse de sentiments, où le modèle peut classifier un texte entier comme positif, négatif ou neutre.

3. **Génération de séquences** : ce mode utilise la séquence d'entrée pour prédire le prochain élément. Utilisé dans la prédiction de séries temporelles, il permet également de générer du texte en anticipant chaque caractère ou mot suivant en fonction de la séquence précédente.

Grâce à leur mémoire, les RNN peuvent gérer les dépendances internes d'une séquence, offrant une solution puissante pour les tâches où les éléments sont interdépendants et où le contexte est essentiel.

9. Maitrisez les algorithmes d'apprentissage des réseaux récurrents

L'apprentissage des réseaux de neurones récurrents (RNN) nécessite des adaptations car la présence de connexions récurrentes interdit l'utilisation de la rétropropagation classique. La solution consiste à utiliser une version dépliée du réseau, une approximation dans laquelle les cycles de récurrence sont remplacés par des copies du réseau sur plusieurs étapes de temps. Ce procédé, appelé **Rétropropagation à Travers le Temps** (BPTT), applique la rétropropagation classique sur un réseau déplié, partageant les poids à chaque étape temporelle.

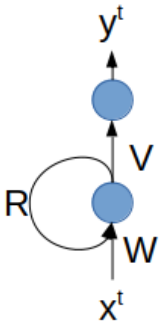


Figure 25 : RNN avec une couche récurrente et une couche dense

Cependant, cet apprentissage déplié peut entraîner deux problèmes majeurs : la **disparition du gradient** (ou gradient évanescent) et l'**explosion du gradient**. Ces phénomènes apparaissent en raison de la profondeur accrue du réseau lors du dépliement, provoquant une diminution ou une augmentation excessive du gradient. Pour contrer cela, on utilise des techniques comme le **gradient clipping**, qui limite l'amplitude du gradient, et d'autres méthodes comme le dropout, la batch normalization, et la régularisation L1 ou L2.

Les **RNN simples** (ou Vanilla RNN) conviennent aux dépendances de courte et moyenne durée, mais montrent leurs limites avec des dépendances temporelles longues, comme dans le traitement du langage naturel, où il est nécessaire de garder des informations sur des centaines de pas de temps. La solution consiste à introduire des cellules à mémoire interne, telles que les **LSTM (Long Short-Term Memory)** et les **GRU (Gated Recurrent Units)**, qui permettent au réseau de contrôler et conserver un état interne en fonction du contexte.

L'apprentissage des RNN peut nécessiter une annotation pour chaque fenêtre temporelle de la séquence, ce qui est fastidieux et imprécis lorsque des fenêtres couvrent plusieurs éléments. Pour simplifier ce processus, l'algorithme **CTC (Connectionist Temporal Classification)** fournit un apprentissage basé sur des étiquettes de séquence globales plutôt que pour chaque étape de temps, rendant l'étiquetage plus efficace et réduisant la complexité de l'apprentissage pour des tâches telles que la reconnaissance de parole ou d'écriture manuscrite.

Figure 26 : Étiquetage global d'une séquence en vue de l'utilisation de l'algorithme d'apprentissage CTC

10. Découvrez les cellules à mémoire interne : les LSTM

Les **cellules LSTM (Long Short Term Memory)** permettent aux réseaux récurrents de maintenir un état sur une longue période, résolvant les problèmes liés aux dépendances à long terme. Une cellule LSTM contient une mémoire interne et trois portes de contrôle : la **porte d'entrée** qui décide si l'entrée actuelle doit modifier la mémoire, la **porte d'oubli** qui peut remettre à zéro la mémoire, et la **porte de sortie** qui contrôle si la mémoire influence la sortie. Chacune de ces portes utilise des fonctions d'activation (souvent des sigmoïdes) pour moduler les informations, ce qui permet un contrôle précis de la mémoire de la cellule.

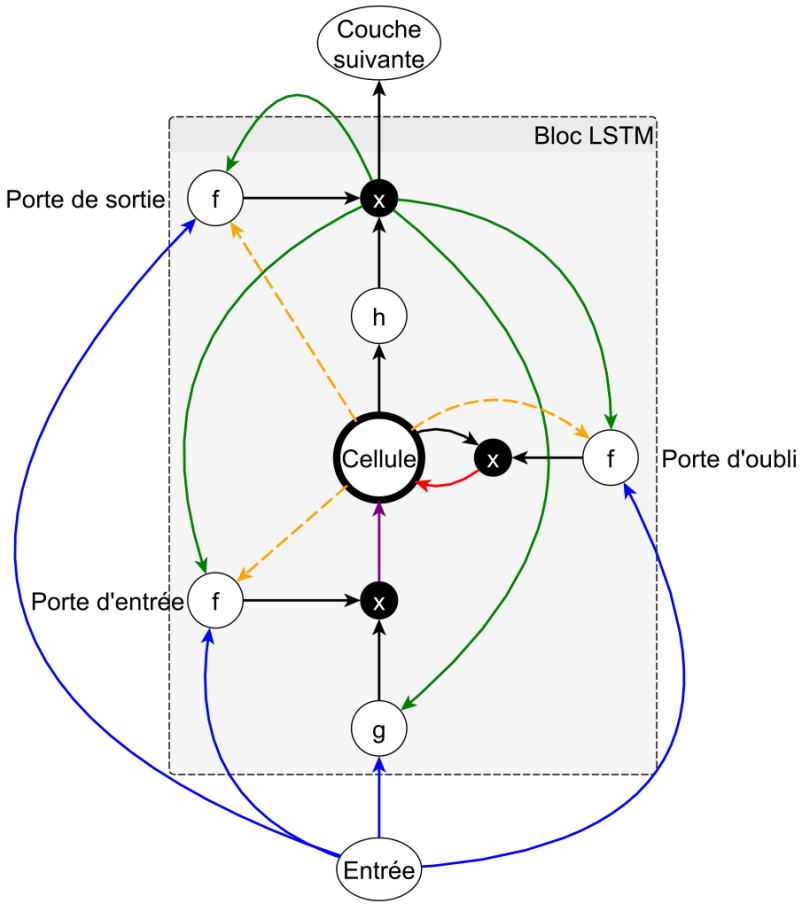


Figure 27 : Schéma d'un neurone LSTM comprenant une mémoire interne (cell) contrôlée par les trois portes d'entrée

L'ensemble des poids des LSTM est appris au cours de l'entraînement via la rétropropagation à travers le temps (BPTT), tout comme les réseaux récurrents simples, bien que les LSTM nécessitent deux fois plus de paramètres, ce qui les rend plus lourds à utiliser.

Les **BLSTM (Bidirectional LSTM)** sont des LSTM bidirectionnels, permettant d'analyser une séquence dans les deux directions, utile lorsque le contexte futur est accessible et peut aider à la décision. Cela permet de prendre les meilleures décisions en utilisant l'information présente avant et après l'élément étudié. Les BLSTM sont adaptés aux tâches d'analyse de séquences enregistrées mais ne conviennent pas aux applications en temps réel, comme la reconnaissance de la parole en direct.

Les **MDLSTM (Multi-dimensional LSTM)** sont une extension qui permet de traiter des données à plusieurs dimensions, telles que les images. Ils parcourent les données dans plusieurs directions : haut-bas, bas-haut, gauche-droite, et droite-gauche. Ce type de réseau est particulièrement efficace pour des tâches comme la reconnaissance d'écriture manuscrite, où il est crucial de modéliser des dépendances bidirectionnelles dans une image.

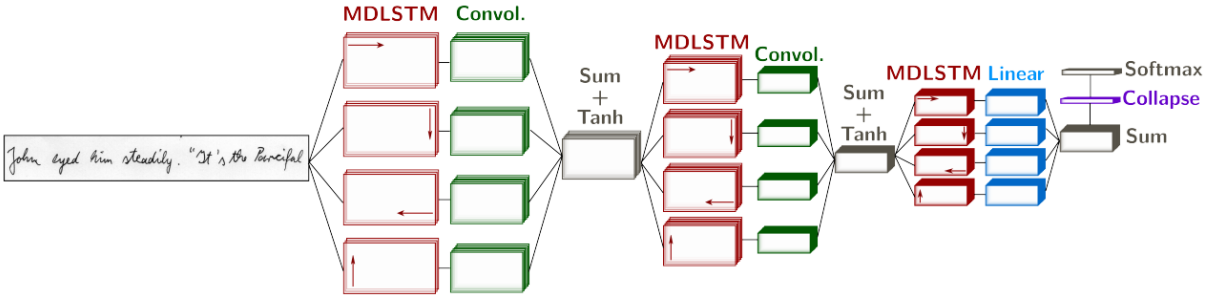


Figure 28 : Exemple d'utilisation d'un MDLSTM pour la reconnaissance d'écriture

Ces architectures avancées de LSTM peuvent être combinées avec des couches convolutionnelles ou denses, offrant des possibilités de traitement adaptées aux différentes caractéristiques des signaux.

11. Construisez des architectures neuronales modulaires

Les réseaux de neurones sont modulaires et permettent de combiner différents types de couches (denses, convolutionnelles, récurrentes, etc.) pour répondre aux besoins d'une tâche spécifique, tout en étant entraînés par la rétropropagation du gradient. Cette flexibilité d'empilement des couches rend possible la construction d'architectures adaptées aux différents types de données.

Les couches denses sont historiquement les premières couches utilisées dans les réseaux de neurones et comportent une interconnexion totale entre les neurones d'une couche et ceux de la suivante. Elles sont généralement placées en fin de réseau pour prendre la décision finale, notamment dans des tâches de classification ou de régression.

Les couches convolutionnelles sont plus légères en termes de paramètres car elles utilisent des filtres, qui permettent de capturer efficacement les caractéristiques des images, comme les contours et textures. Utilisées en association avec des couches de pooling, elles sont idéales pour les couches basses du réseau dans les tâches de traitement d'image, d'analyse de texte ou de signal sonore, où elles capturent des caractéristiques essentielles.

Les couches récurrentes sont adaptées au traitement des séquences et permettent de modéliser les dépendances temporelles. Elles sont couramment employées pour des tâches impliquant des séquences (comme l'écriture, la parole ou les séries temporelles). Elles peuvent également être utilisées en mode génératif pour prédire les éléments suivants d'une séquence. Cependant, les couches récurrentes sont lourdes en calculs et nécessitent souvent des couches de type LSTM pour gérer des dépendances longues.

Exemples de Modèles Profonds

1. **Classification d'images** : L'architecture VGG16, conçue pour la classification d'images dans la base ImageNet, utilise des couches convolutionnelles avec des filtres et du max pooling pour extraire les caractéristiques visuelles. En sortie, plusieurs couches denses avec une fonction softmax effectuent la classification parmi plusieurs catégories.

2. **Reconnaissance d'écriture manuscrite** : Un modèle de reconnaissance d'écriture combine des couches convolutionnelles, des activations leaky ReLU, et des couches récurrentes bidirectionnelles (BLSTM) pour capturer l'aspect séquentiel des caractères. La structure est adaptée aux séquences d'écriture en représentant chaque colonne de pixels, permettant une reconnaissance précise de chaque caractère dans la séquence.
3. **Génération de légendes pour les images** : Une architecture complexe pour générer des descriptions d'images utilise d'abord des couches convolutionnelles pour l'extraction des caractéristiques visuelles, suivies d'un réseau LSTM pour la génération séquentielle de texte. Un module d'attention identifie les zones importantes de l'image pour chaque mot généré, ajustant l'attention selon le contexte de la légende en cours.

Ces exemples montrent comment les architectures modulaires des réseaux de neurones permettent de résoudre des tâches complexes en adaptant l'empilement des couches et en tirant parti des avantages de chaque type de couche.

13. Référence

1. *Initiez-vous au Deep Learning*. (n.d.).

   https://openclassrooms.com/fr/courses/5801891-initiez-vous-au-deep-learning