Projet Recherche

Groupe 1

Réseau de neurones et métaheuristiques pour une prédiction fiable de la production d'énergie photovoltaïque de bâtiments à énergie positive.

Hugo CHAMPY, Louis DUMONT, Maxime EY, Tom SEITZ, Quentin STUBECKI

A4 FISE Info 2023/2024

SOMMAIRE

1. RESUME ET ABSTRACT

1.1. RESUME

Dans le cadre de la décentralisation de la production d'énergie, les bâtiments évoluent pour devenir des unités de production énergétique grâce à l'intégration de la technologie photovoltaïque (PV). Cette étude se concentre sur le développement d'un outil prédictif précis de la production énergétique des bâtiments en phase de conception, en utilisant des variables telles que les données météorologiques et la géolocalisation dans diverses régions.

Le cœur de cette recherche réside dans l'optimisation des hyperparamètres des modèles de prédiction par des métaheuristiques pour maximiser la précision des estimations énergétiques. Les hyperparamètres influencent fortement les performances des modèles d'apprentissage automatique, et leur réglage optimal est crucial pour obtenir des prédictions fiables. Nous employons des techniques métaheuristiques, telles que les algorithmes génétiques, les colonies de fourmis et le recuit simulé, pour explorer systématiquement l'espace des hyperparamètres. La précision des modèles est évaluée en utilisant l'erreur quadratique moyenne (Mean Squared Error, MSE) comme critère de validation. Les métaheuristiques permettent de minimiser la MSE en identifiant les combinaisons d'hyperparamètres les plus performantes. Les résultats montrent une amélioration significative de la précision des prédictions par rapport aux méthodes d'optimisation traditionnelles.

Cette approche offre une solution robuste et adaptative aux variations des conditions météorologiques et des configurations urbaines. Les implications sont importantes pour le développement de bâtiments à énergie positive, facilitant l'adoption de technologies photovoltaïques optimisées et contribuant à des stratégies énergétiques durables en milieu urbain.

1.2. ABSTRACT

In the context of decentralizing energy production, buildings are evolving into energy production units through the integration of photovoltaic (PV) technology. This study focuses on developing an accurate predictive tool for the energy production of buildings during the design phase, using variables such as meteorological data and geolocation across various regions.

The core of this research lies in optimizing the hyperparameters of prediction models using metaheuristics to maximize the accuracy of energy estimates. Hyperparameters significantly influence the performance of machine learning models, and their optimal tuning is crucial for reliable predictions. We employ metaheuristic techniques, such as genetic algorithms, ant colony optimization, and simulated annealing, to systematically explore the hyperparameter space. The accuracy of the models is evaluated using the Mean Squared Error (MSE) as the validation criterion. Metaheuristics allow for minimizing MSE by identifying the most effective hyperparameter combinations. The results show a significant improvement in prediction accuracy compared to traditional optimization methods.

This approach offers a robust and adaptive solution to variations in weather conditions and urban configurations. The implications are significant for the development of positive energy buildings, facilitating the adoption of optimized photovoltaic technologies and contributing to sustainable energy strategies in urban environments.

2. INTRODUCTION

2.1. DESCRIPTION DU SUJET

Dans le contexte de la décentralisation de la production d'énergie, les infrastructures et les bâtiments évoluent progressivement pour devenir des unités de production énergétique. L'utilisation de la technologie photovoltaïque (PV) se généralise, en particulier en milieu urbain, en tant que l'une des rares sources d'énergie renouvelable pouvant s'intégrer harmonieusement. Cela implique une collaboration essentielle entre architectes, artisans, bureaux d'étude et l'industrie photovoltaïque, afin de trouver des solutions adaptées à chaque situation tout en préservant l'aspect esthétique et fonctionnel de notre patrimoine immobilier.

L'étude dans laquelle s'inscrit ce projet vise à développer un outil polyvalent capable de prédire précisément la production énergétique des bâtiments en phase de conception, dans diverses régions européennes. Cette prédiction s'appuie sur plusieurs variables, telles que les données météorologiques et la géolocalisation.

Nous allons nous intéresser à l'optimisation de l'apprentissage automatique. Pour se faire, il est nécessaire de régler minutieusement les nombreux éléments constitutifs du modèle, appelés hyperparamètres, car ils exercent une influence significative sur son comportement. Pour cela, on utilise des méthodes à base d'heuristiques et de métaheuristiques.

L'objectif est d'utiliser des métaheuristiques, afin de trouver la meilleure combinaison de valeurs des hyperparamètres, pour une plus grande précision des prédictions réalisées concernant les panneaux photovoltaïques dans les bâtiments à énergie positive (BIPV).

2.2. QUESTION DE RECHERCHE

Nous nous poserons ainsi la question suivante :

Quels hyper paramètres du modèle est-il nécessaire d'optimiser pour améliorer la précision de nos résultats ?

Pour répondre à cette problématique nous explorerons les pistes des métaheuristiques pour faire varier nos hyper paramètres dans l'optique de trouver la combinaison optimale à notre modèle. Les métaheuristiques sont choisies pour leur capacité à explorer de vastes espaces de recherche et à éviter les pièges des minima locaux. Le recuit simulé, les algorithmes génétiques et l'optimisation par essaim particulaire sont des techniques particulièrement adaptées pour ce type de tâche en raison de leur robustesse et de leur capacité à découvrir de bonnes solutions dans des espaces complexes.

L'analyse des résultats aidera à comprendre les interactions entre les hyperparamètres et leur influence sur la précision du modèle. L'utilisation de visualisations aidera à illustrer ces relations et à fournir des insights clairs sur les meilleures pratiques en matière de configuration du modèle.

3. ETAT DE L'ART

3.1. LES METAHEURISTIQUES

Dans le contexte de notre activité de recherche, les métaheuristiques jouent un rôle crucial pour trouver des solutions bonnes ou proches de l'optimalité dans des temps de calcul raisonnables, surtout pour les instances de grandes tailles où les méthodes exactes deviennent impraticables. Les métaheuristiques sont des stratégies de niveau supérieur qui guident le processus de recherche de solutions afin d'explorer l'espace des solutions de manière efficace. Contrairement aux heuristiques, qui sont généralement spécifiques à un problème, les métaheuristiques sont conçues pour être applicables à une large gamme de problèmes. Elles visent à trouver une solution optimale ou suboptimale en évitant de rester bloqué dans des optimums locaux.

Voici des exemples de métaheuristiques pour tenter de résoudre notre problème :

3.1.1. LE RECUIT SIMULE [1] [2]

Le recuit simulé est une technique d'optimisation inspirée par le processus de refroidissement des métaux. Ce processus, également connu sous le nom de recuit, consiste à chauffer un matériau à une température élevée avant de le refroidir lentement pour atteindre une structure plus stable et minimiser les défauts. En se basant sur cette analogie, le recuit simulé explore l'espace de recherche de solutions possibles en acceptant des solutions pires avec une certaine probabilité, probabilité qui diminue progressivement au fil du temps. Cela permet à l'algorithme d'échapper aux minima locaux et d'augmenter les chances de trouver la solution globale optimale.

La première étape du recuit simulé consiste à choisir une solution initiale aléatoire dans l'espace de recherche et à définir une température initiale élevée. La solution initiale sert de point de départ pour l'exploration de l'espace de recherche, tandis que la température initiale contrôle la probabilité d'acceptation des solutions pires au début du processus.

À chaque itération, une nouvelle solution est générée en perturbant légèrement la solution actuelle. Cette perturbation peut prendre différentes formes, selon le problème à résoudre, mais elle doit être suffisamment petite pour permettre une exploration locale tout en étant capable de couvrir l'ensemble de l'espace de recherche.

La nouvelle solution est ensuite évaluée en calculant sa performance selon une métrique prédéfinie. Cette évaluation permet de déterminer si la nouvelle solution est meilleure ou pire que la solution actuelle. La nouvelle solution est acceptée selon deux critères :

- Si elle est meilleure que la solution actuelle, elle est automatiquement acceptée.
- Si elle est pire, elle peut être acceptée avec une certaine probabilité. Cette probabilité est définie par une fonction exponentielle qui dépend de la différence de performance entre les solutions et de la température actuelle. Au fur et à mesure que la température diminue, la probabilité d'accepter des solutions pires diminue également.

La température est progressivement réduite selon un schéma de refroidissement prédéterminé. Cette réduction progressive de la température diminue la probabilité d'accepter des solutions pires, ce qui permet à l'algorithme de se concentrer davantage sur l'exploitation des régions les plus prometteuses de l'espace de recherche.

Les étapes de perturbation, d'évaluation, d'acceptation et de refroidissement sont répétées jusqu'à ce que la température atteigne un seuil suffisamment bas ou qu'un critère d'arrêt soit satisfait (par exemple, un nombre maximal d'itérations ou une convergence des solutions).

3.1.2. L'OPTIMISATION PAR ESSAIM DE PARTICULES (PSO) [3]

L’optimisation par essaime de particules (Particle Swarm Optimization ou PSO en anglais) est une méthode d'optimisation stochastique inspirée par le comportement collectif des organismes vivants, tels que les bancs de poissons et les vols d'oiseaux. Elle est largement utilisée pour résoudre des problèmes d'optimisation complexes dans divers domaines, notamment l'ingénierie, la finance et la recherche opérationnelle.

Le principe est simple, il nous faut simuler un groupe de particules (agents) qui se déplacent dans l'espace de recherche d'un problème d'optimisation. Chaque particule représente une solution candidate et ajuste sa position en fonction de sa propre expérience et de celle de ses voisines. Les particules se déplacent vers des positions prometteuses en combinant des aspects d'exploration (recherche de nouvelles solutions) et d'exploitation (affinement des solutions existantes).

Le mécanisme de fonctionnement du PSO commence par l'initialisation des particules avec des positions et des vitesses aléatoires dans l'espace de recherche. Chaque particule possède une position actuelle, une vitesse et une position personnelle optimale. Ensuite, chaque particule évalue la qualité de sa position actuelle en utilisant une fonction objectif spécifique au problème à résoudre. Les vitesses des particules sont mises à jour en tenant compte de trois composants : l'inertie, qui représente la tendance de la particule à conserver sa vitesse actuelle ; le composant cognitif, qui incite la particule à revenir vers sa position personnelle optimale ; et le composant social, qui pousse la particule à se diriger vers la meilleure position connue par l'ensemble de l'essaim. Les positions personnelles optimales et la position globale optimale sont mises à jour si de meilleures positions sont trouvées. Les étapes d'évaluation et de mise à jour sont répétées jusqu'à ce qu'un critère d'arrêt soit atteint, tel qu'un nombre maximal d'itérations ou une convergence suffisante.

Le PSO présente plusieurs avantages. Il est relativement simple à comprendre et à implémenter comparé à d'autres algorithmes d'optimisation évolutive. Il est capable de converger rapidement vers des solutions de haute qualité pour de nombreux types de problèmes d'optimisation et peut gérer des problèmes d'optimisation avec des paysages de fonction objectif complexes, incluant des fonctions non linéaires, non différentiables et multimodales. De plus, le PSO peut être facilement adapté et combiné avec d'autres algorithmes d'optimisation pour améliorer ses performances.

Cependant, le PSO a également des inconvénients. Il peut parfois converger prématurément vers des optima locaux, surtout dans des espaces de recherche complexes. La performance de PSO est sensible aux choix des paramètres, tels que le facteur d'inertie et les coefficients d'accélération, ce qui peut nécessiter une expérimentation ou une optimisation préalable. Pour des problèmes de très grande dimension, le PSO peut devenir moins efficace et nécessiter des adaptations spécifiques. Le PSO a été appliqué avec succès dans divers domaines tels que l'ingénierie, pour l'optimisation de la conception des structures et le réglage des paramètres des systèmes de contrôle ; la finance, pour l'optimisation de portefeuilles, la gestion des risques et les prévisions financières ; la recherche opérationnelle, pour la planification de la production, l'optimisation de la logistique et la gestion des ressources ; et le machine learning, pour l'entraînement de réseaux de neurones, la sélection de caractéristiques et l'optimisation des hyperparamètres.

3.1.3. PLAN D'EXPERIENCES [4]

Un plan d'expérience est une méthode systématique utilisée pour organiser et structurer une expérience. Son objectif principal est de maximiser l'information obtenue tout en minimisant les ressources nécessaires, telles que le temps, les coûts et les efforts.

Les composants essentiels d'un plan d'expérience incluent d'abord les objectifs de l'expérience, qui définissent clairement ce que l'on cherche à découvrir ou à vérifier. Ensuite, il y a les facteurs et niveaux, qui sont les variables indépendantes que l'on souhaite étudier et leurs différentes valeurs ou conditions respectives. Les variables de réponse sont également cruciales ; ce sont les résultats mesurables de l'expérience, comme le rendement ou la qualité du produit. Les hypothèses, qui sont des prédictions ou des suppositions à tester, font également partie intégrante du plan. Pour contrôler les effets des variables confondantes, un plan de randomisation est élaboré, distribuant les traitements de manière aléatoire. Les réplicas, sont utilisés pour évaluer la variabilité et la reproductibilité des résultats.

Il existe plusieurs types de plans d'expérience. Les plans factoriels examinent toutes les combinaisons possibles des niveaux des facteurs. Un plan factoriel complet teste toutes les combinaisons, tandis qu'un plan factoriel fractionné en teste une sous-partie pour réduire le nombre d'expériences nécessaires. Les plans à blocs regroupent des unités expérimentales similaires pour réduire la variabilité. Les plans de surface de réponse modélisent et optimisent des processus en ajustant des modèles quadratiques aux données expérimentales. Les plans de mélanges sont appliqués lorsque les facteurs sont des proportions d'un mélange, comme dans la formulation de produits. Les plans séquentiels permettent d'ajuster le plan en fonction

des résultats obtenus au fur et à mesure de l'expérience. Les avantages d'un plan d'expérience sont nombreux. Il permet de réduire le nombre d'expériences nécessaires tout en maximisant les informations obtenues, ce qui le rend efficace. Il offre une estimation précise des effets des facteurs et de leurs interactions, améliorant ainsi la précision des résultats. Il aide également à identifier les conditions optimales pour obtenir le résultat souhaité, contribuant à l'optimisation des processus. Enfin, en contrôlant les sources de variation non désirées, il assure la robustesse et la fiabilité des résultats obtenus.

3.2. ETUDE DU MODELE

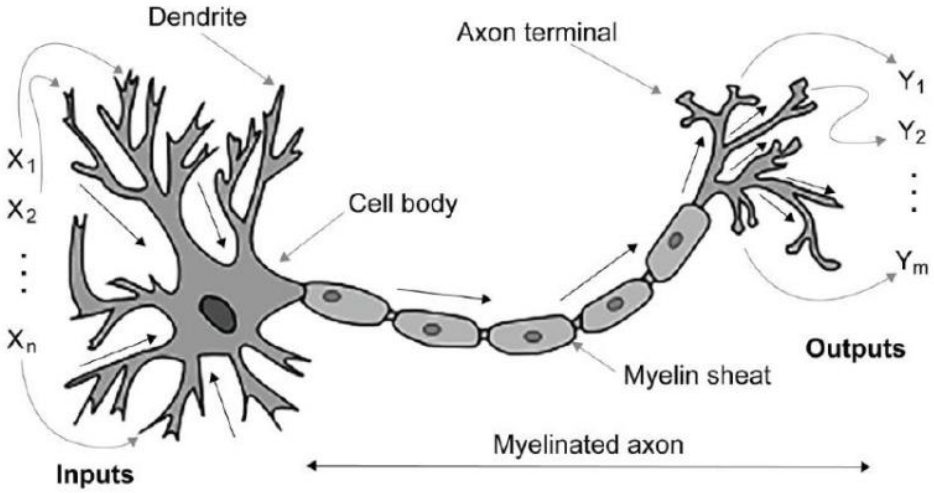
Nous cherchons à comprendre le fonctionnement du modèle afin de pouvoir déduire les hyperparamètres à optimiser pour améliorer le modèle.

Le modèle est un réseau de neurones. Il s'agit donc de comprendre initialement le fonctionnement d'un perceptron, qui correspond à un neurone artificiel. Puis leurs différentes implémentations qu'utilisent le modèle.

3.2.1. LE PERCEPTRON [5] [6]

Un neurone constitue l'unité fondamentale du système nerveux chez l'humain, servant à la transmission de signaux électriques à travers l'organisme. Les milliards de neurones biologiques qui composent le cerveau humain sont en communication constante, échangeant de petits signaux électriques binaires par activation ou désactivation. Un réseau neuronal se définit comme un ensemble de ces neurones liés entre eux. Les réseaux de neurones artificiels (RNA), sont conçus en s'inspirant des réseaux neuronaux biologiques. L'expression "intelligence artificielle" prend racine dans l'existence de l'intelligence naturelle au sein du cerveau humain (ou de tout cerveau, en général), intelligence que nous cherchons à reproduire de manière artificielle. Pour rendre cela plus compréhensible, nous allons initier notre explication par une comparaison entre le neurone naturel et son équivalent artificiel, le perceptron.

Une version simplifiée d'un neurone biologique est représentée ci-dessous :

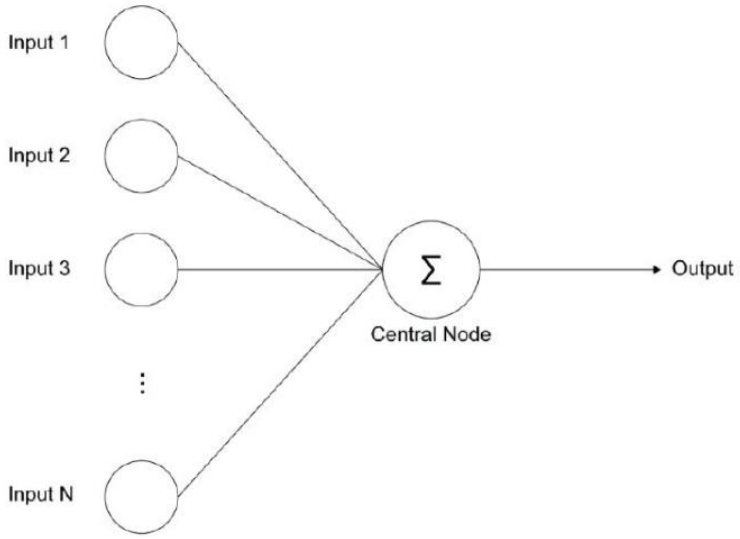


Cette représentation est grandement simplifiée dont voici les 3 composants principales :

- Les dendrites, qui correspondent aux signaux d'entrés.
- Le corps cellulaire, où signal des dendrites est traité.
- Les terminaisons axonales, via lesquels le neurone transmet le signal de sortie aux autres neurones.

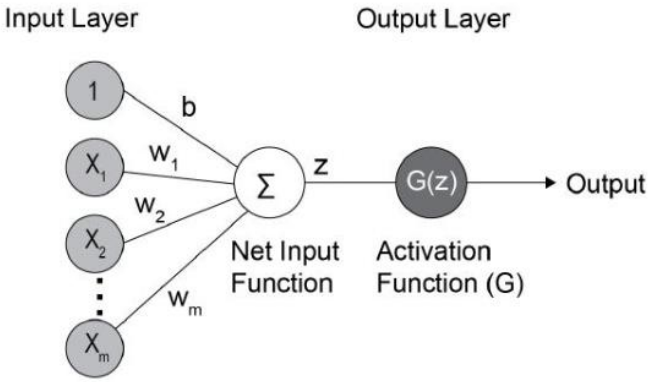
A partir de cette représentation simplifiée nous pouvons faire le parallèle avec le perceptron, dont la conception est directement inspirée du fonctionnement des neurones biologiques.

La figure ci-dessous représente une version haut-niveau du perceptron :



Dans un neurone biologique, tout comme dans neurone artificielle, le nœud central combine l'ensemble des signaux d'entrées et active un signal de sortie si l'assemblage des signaux d'entrée dépassent un certain seuil.

Une représentation plus détaillée du perceptron est montrée ci-dessous :



Un perceptron est donc constitué de plusieurs composants que nous allons décrire ci-dessous.

Les **entrée** (X_1 ,X_2, …, X_m) :

Chaque neurone reçoit des données d'entrée représentées par X1, X2, …, Xm. Les données d'entrée peuvent être des données structurées (comme un fichier CSV) ou des données non structurées comme une image. Ces entrées sont appelées features.

Les **poids** (W_1, W_2, …, W_m) :

Chaque neurone possède des pondérations correspondant au nombre d'entrées qu'il reçoit. Ainsi, pour m entrées (X_1, X_2, …, X_m), il y aura m pondérations (W_1, W_2, …, W_m), chacune étant un nombre réel qui peut être positif ou négatif. Ces valeurs ne sont pas fixées à l'avance mais sont déterminées au cours du processus d'apprentissage.

Le **biais** :

Le rôle du biais est de permettre un ajustement de la fonction de sortie indépendamment des valeurs d'entrée. Il joue un rôle essentiel pour augmenter la flexibilité du modèle en lui permettant de mieux s'ajuster aux données

La **somme pondérée** :

La somme pondérée correspond à la somme des produits des entrées et de leurs poids correspondants, additionnée au biais.

Elle est décrite par la formule mathématique suivante :

$$z = \sum_{i=0}^{m} (W_i * X_i) + b$$

Où W_i correspond aux poids.

X_i correspond aux valeurs d'entrée.

B correspond au biais.

La fonction d'activation (G(z)) :

L'issue de la somme pondérée est utilisée comme entrée pour la fonction d'activation.

L'issue de cette fonction d'activation constitue la sortie finale du neurone.

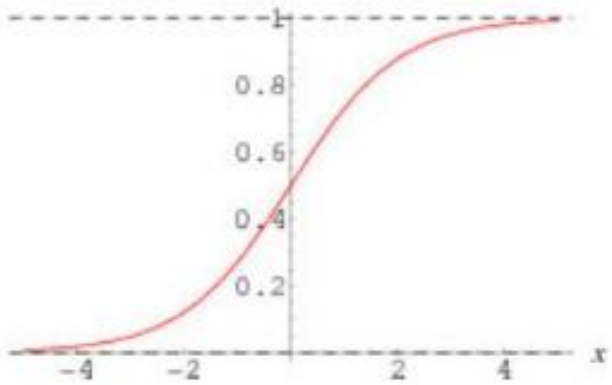
La fonction d'activation instaure une non-linéarité. Il existe une diversité de fonctions d'activation. La plus connue est la fonction d'activation Sigmoïde, qui génère une sortie binaire, permettant ainsi au perceptron de se comporter comme un neurone biologique présentant deux états possibles : activé ou désactivé.

L'équation mathématique de la fonction sigmoïde est :

$$Sortie = \frac{1}{1 + e^{-z}}$$

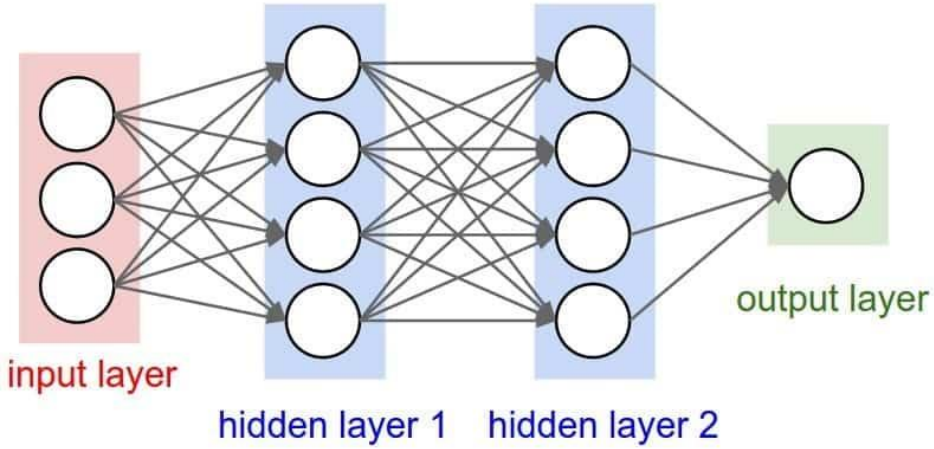
Où z correspond au résultat de la somme pondérée.

Sa représentation mathématique est la suivante :



La sortie de la fonction sigmoïde est toujours binaire grâce au seuil qu'on définit à l'avance. Si la valeur de sortie de la sigmoïde est supérieure au seuil, la valeur de sortie du perceptron est 1, dans le cas contraire, la valeur de sortie sera 0.

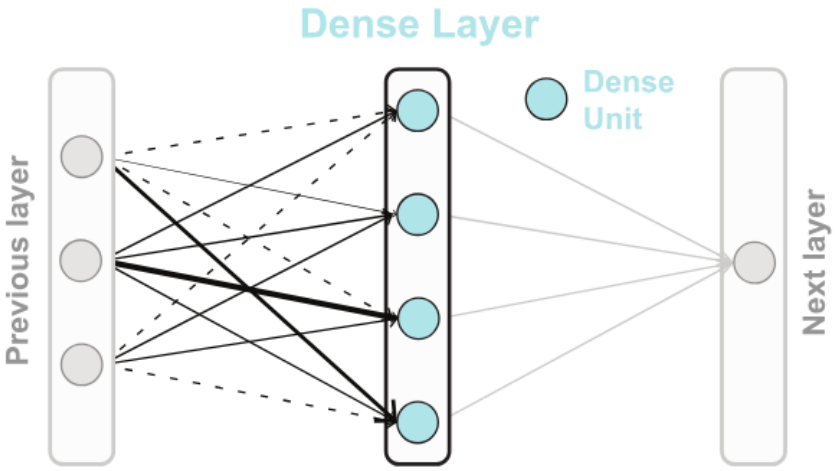
Le perceptron cherche donc à imiter le fonctionnement d'un neurone. L'association de plusieurs neurones correspond à un réseau de neurones artificiel. Ils sont souvent disposés sous la forme de couches.



3.2.2. COUCHE DENSES [7]

Une couche dense également appelée couche entièrement connectée est une couche utilisée dans les étapes finales du réseau neuronal. Cette couche aide à modifier la dimensionnalité de la sortie de la couche précédente afin que le modèle puisse facilement définir la relation entre les valeurs des données dans lesquelles le modèle fonctionne.

Dans tout réseau neuronal, une couche dense est une couche profondément connectée à la couche précédente, ce qui signifie que les neurones de la couche sont connectés à chaque neurone de sa couche précédente.



3.3. LONG SHORT TERM MEMORY (LSTM)

Les réseaux LSTM sont une variation des réseaux de neurones récurrents (RNN). En effet, les RNN classiques ont des difficultés à traiter les dépendances à long terme en raison du problème du "vanishing gradient", les LSTM ont été spécialement conçus pour surmonter ce problème.

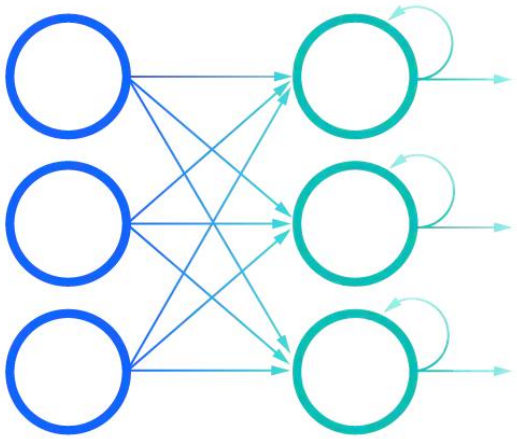
Nous allons faire un point rapide sur les RNN afin de comprendre les avantages de l'utilisation des LSTM.

3.3.1. RECURRENT NEURAL NETWORK (RNN) [8] [9]

Les réseaux de neurones récurrents (RNN) sont particulièrement adaptés aux problèmes où l'ordre des données est plus significatif que les données isolées elles-mêmes. Ils sont fréquemment employés pour le traitement de séquences de données, telles que les séquences temporelles ou les textes dans divers formats.

Grâce à leur capacité à stocker des informations grâce à une mémoire interne, les RNN peuvent retenir des détails essentiels des données traitées, ce qui les rend extrêmement efficaces pour anticiper les événements ou éléments à suivre. Contrairement aux réseaux de neurones traditionnels où l'information progresse uniquement dans un sens, de l'entrée vers la sortie en passant par les couches cachées sans jamais revisiter un même nœud, les RNN fonctionnent différemment.

Dans le cas des RNN, l'information circule à travers une boucle permettant au réseau de prendre en compte à la fois les données actuelles et les enseignements tirés des données antérieures lorsqu'il réalise des prédictions.



Toutefois, les réseaux de neurones récurrents (RNNs) sont confrontés à une difficulté connue sous le nom de « gradient vanishing ». Cette complication découle d'un processus de rétropropagation utilisé par les réseaux neuronaux pour affiner leur apprentissage.

Dans ce modèle, le réseau évalue l'écart entre sa prédiction et la réponse attendue, puis utilise cette information pour ajuster ses paramètres, tels que les poids, en fonction d'une quantité dénommée gradient. Un gradient plus élevé signifie des modifications plus importantes des paramètres, et inversement. Ce cycle se répète jusqu'à atteindre un niveau acceptable de précision.

Les RNNs mettent en œuvre la rétropropagation à travers le temps, ce qui signifie que les calculs actuels dépendent des résultats précédents. Cependant, si la valeur du gradient devient trop faible à un certain point, elle diminuera encore davantage à l'étape suivante, menant à une réduction exponentielle des gradients. À terme, cela empêche le modèle de progresser dans son apprentissage.

Ce phénomène est désigné comme le problème de disparition du gradient et il limite la capacité des RNNs à conserver des informations sur le long terme, rendant les données antérieures de moins en moins influentes sur les résultats actuels.

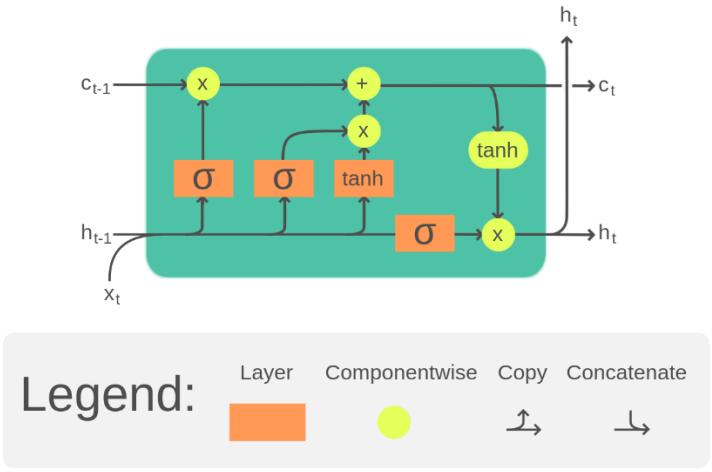
3.3.2. FONCTIONNEMENT DU MODELE LSTM [10] [11] [12]

Les réseaux LSTM ont été développés spécifiquement pour adresser cette problématique. Ils incorporent des mécanismes de contrôle pour contrer le problème de disparition du gradient et pour saisir de manière efficace les relations à long terme au sein des séquences.

Les LSTM introduisent un élément novateur, la cellule de mémoire, qui confère au réseau la faculté de conserver et d'exploiter des informations sur une durée prolongée. Le modèle est donc composé de de deux types de mémoire, une mémoire longue qui stocke les informations importantes tout le long du processus et une mémoire courte, qui traite les informations plus temporaires.

La structure de la cellule de mémoire dans un LSTM inclut diverses portes : une porte pour l'ajout d'informations nouvelles (la porte d'entrée), une pour le retrait d'informations (la porte d'oubli) et une autre pour la sortie d'informations (la porte de sortie). Ces portes facilitent la gestion du flux d'informations au sein de la cellule de mémoire, offrant la possibilité de décider quelles données garder ou éliminer.

Ce mécanisme permet aux LSTM de retenir des données significatives sur de longs intervalles tout en écartant les informations jugées non essentielles.



Le modèle LSTM est donc composé de 3 étapes pour gérer le flux d'informations dans la cellule, qui correspondent aux 3 portes.

La porte d'oubli :

Cette première étape de notre LSTM consiste à déterminer les informations à éliminer de la cellule de mémoire.

Durant cette phase, l'attention est portée sur h_{t-1} et X_t, produisant un score entre 0 et 1 pour chaque élément dans la cellule de mémoire C_{t-1}. Un score de 1 signifie « conserver intégralement », tandis qu'un score de 0 indique « supprimer entièrement ».

La porte d'entrée :

Cette phase implique la sélection des informations nouvelles à stocker dans la cellule mémoire et se divise en deux segments.

Le premier segment emploie une couche sigmoïde pour déterminer quelles informations seront mises à jour.

Le second segment utilise une couche tanh pour générer un vecteur de nouvelles valeurs potentielles, $C_{\sim t}$, qui pourraient être incorporées dans la cellule mémoire.

Par la suite, les résultats issus de ces deux processus sont fusionnés, permettant ainsi l'ajout des valeurs sélectionnées dans la cellule mémoire.

La porte de sortie :

Au cours de cette dernière étape, la cellule détermine ce qu'elle produira comme résultat de la mémoire à court terme.

Initialement, une couche sigmoïde est appliquée sur les valeurs h_{t-1} et X_t pour pour décider quelles informations doivent être stockées dans la mémoire courte et qui seront transmises à la cellule suivante.

Par la suite, une fonction tanh est appliquée sur une copie de la mémoire longue pour normaliser les valeurs dans l'intervalle de −1 à 1.

Les résultats de ces 2 opération sont ensuite multipliés pour déterminer quelles informations de la mémoire courte seront utilisées pour générer la sortie de la cellule LSTM.

3.3.3. LIMITATIONS

Complexité et Coût Computationnel : L'une des principales contraintes des LSTM est leur architecture complexe qui les rend significativement plus coûteux en termes de calcul et de mémoire par rapport aux RNN simples. Chaque cellule LSTM intègre plusieurs portes (par exemple, les portes d'entrée, de sortie et d'oubli) qui contrôlent le flux d'information. Ce degré de complexité augmente le nombre de paramètres à entraîner et, par conséquent, exige des ressources computationnelles plus importantes, ce qui peut se traduire par un temps d'entraînement plus long et un besoin accru en puissance de calcul.

Difficultés de Parallélisation : Du fait de leur nature séquentielle, où chaque calcul dépend de l'état précédent, les LSTM tout comme les RNN sont difficiles à paralléliser. Cette limitation impacte la vitesse d'entraînement et la scalabilité des LSTM.

Risque de Surajustement : Les LSTM, de par leur capacité à capturer des dépendances complexes dans de longues séquences, sont également susceptibles de surajustement, en particulier lorsqu'ils sont entraînés sur des ensembles de données de petite ou moyenne taille. Sans une régularisation appropriée, comme l'application de techniques de dropout ou l'usage de normes de régularisation, les LSTM peuvent s'adapter excessivement aux données d'entraînement, perdant ainsi en généralisation sur de nouvelles données.

Gestion des Séquences Extrêmement Longues : Bien que mieux équipés que les RNN standards pour gérer des dépendances à long terme, les LSTM peuvent encore rencontrer des difficultés avec des séquences extrêmement longues. Dans de tels cas, même les LSTM pourraient ne pas être capables de conserver toute l'information pertinente au fil des séquences, résultant en une performance moindre pour certaines applications spécifiques exigeant la mémorisation de contextes prolongés.

3.4. TENSORFLOW [13]

TensorFlow est une bibliothèque open-source de machine learning développée par l'équipe Google Brain. Publiée en 2015, elle est rapidement devenue une référence dans le domaine du deep learning, grâce à sa flexibilité, sa performance et son large écosystème d'outils. TensorFlow permet de construire, entraîner et déployer des modèles de machine learning sur divers dispositifs, allant des ordinateurs de bureau aux appareils mobiles et aux infrastructures de cloud computing.

TensorFlow se distingue par sa flexibilité et sa modularité. Il permet de définir des modèles de machine learning de manière déclarative et impérative, ce qui le rend adapté à la recherche ainsi qu'à la production. Il supporte divers niveaux d'abstraction, allant des opérations de bas niveau aux API de haut niveau comme Keras. De plus, TensorFlow fonctionne sur des CPU, GPU et TPU (Tensor Processing Units) de Google, optimisant ainsi l'entraînement et l'inférence des modèles. Son large écosystème comprend des outils comme TensorBoard pour la visualisation, TensorFlow Lite pour le déploiement sur des appareils mobiles, et TensorFlow Extended (TFX) pour la mise en production des modèles. Une communauté active de chercheurs et de développeurs contribue à son amélioration continue, avec une documentation abondante, des tutoriels et des exemples disponibles pour aider à l'apprentissage et à l'utilisation. TensorFlow utilise des graphes de calcul pour représenter les opérations mathématiques. Chaque nœud du graphe représente une opération, et les arêtes représentent les flux de données, appelées tensors. Initialement, TensorFlow utilisait des sessions pour exécuter les graphes, nécessitant une phase de construction et une phase d'exécution distinctes. De plus, TensorFlow 2.0 intègre Keras en tant qu'API de haut niveau pour la construction et l'entraînement des modèles, facilitant la création rapide de prototypes tout en permettant des ajustements fins pour des recherches avancées.

Les applications de TensorFlow sont nombreuses et variées. En vision par ordinateur, il est utilisé pour la reconnaissance d'objets, la segmentation d'images et la génération d'images avec des modèles GANs (Generative Adversarial Networks). Dans le domaine du traitement du langage naturel (NLP), TensorFlow est employé pour la traduction automatique, la génération de texte, l'analyse des sentiments et les chatbots. Pour les séries temporelles et les prévisions, il est utilisé pour la prévision de la demande, la détection d'anomalies et la modélisation financière. Enfin, TensorFlow est également utilisé pour les systèmes de recommandation et la personnalisation, comme dans les plateformes de contenu et la personnalisation des publicités.

Comparé à d'autres frameworks, TensorFlow a ses avantages et ses inconvénients. PyTorch, par exemple, est connu pour son approche dynamique et intuitive, souvent préféré pour la recherche et le prototypage rapide. Cependant, TensorFlow est souvent choisi pour les déploiements en production en raison de son écosystème complet. MXNet, principalement utilisé par Amazon, est également performant et flexible, mais TensorFlow dispose d'une communauté et d'un écosystème plus larges. Caffe, optimisé pour la vision par ordinateur, est utilisé pour des applications spécifiques nécessitant des modèles de deep learning rapides et bien optimisés, mais TensorFlow offre une flexibilité et un support plus larges pour diverses applications. Les avantages de TensorFlow incluent sa grande flexibilité et sa capacité à gérer

des tâches variées, son support multi-plateforme et son optimisation pour le hardware spécialisé, ainsi que son écosystème riche et ses outils complémentaires facilitant le développement et le déploiement. De plus, sa documentation complète et sa communauté active sont des atouts majeurs. Cependant, TensorFlow a aussi des inconvénients. Sa courbe d'apprentissage initiale est plus élevée comparée à certains autres frameworks, et sa flexibilité peut parfois conduire à une complexité supplémentaire dans les projets plus simples. De plus, TensorFlow 1.x était souvent critiqué pour sa verbosité et sa difficulté d'utilisation, bien que cela ait été largement amélioré dans TensorFlow 2.0.

3.5. HYPER PARAMETRES

Pour faire varier notre modèle, nous allons faire varier différents paramètres qui vont directement influer sur la précision et la rapidité du modèle.

Epochs :

Le nombre d'epochs est un hyperparamètre qui définit le nombre de fois que l'algorithme d'apprentissage travaillera sur l'ensemble du jeu de données d'apprentissage.

Taille du lot :

Dans notre modèle, la taille du lot est le nombre d'échantillons traités avant la mise à jour du modèle

Fonction d'activation :

Cette fonction détermine si un neurone doit être activé (ou non), et elle introduit des non-linéarités dans le modèle, permettant aux réseaux de neurones d'apprendre des relations complexes et non-linéaires dans les données. Nous avons les fonctions sigmoïdes, hyperboliques, Relu, Leaky Relu et Softmax.

Taux d'apprentissage :

Il détermine la taille des pas que l'algorithme fait lors de la mise à jour des poids du modèle en fonction du gradient de la fonction de perte. Le taux d'apprentissage contrôle la vitesse et la qualité de la convergence du modèle vers un minimum de la fonction de perte.

Nombre de neurones :

Le nombre de neurones par couche dans un réseau de neurones est un hyperparamètre essentiel qui influence la capacité du modèle à apprendre des motifs à partir des données. La sélection du nombre approprié de neurones dans chaque couche est cruciale pour obtenir de bonnes performances tout en évitant le surapprentissage ou le sous-apprentissage.

Couches d'Entrée et de Sortie :

Le nombre de neurones dans la couche d'entrée doit correspondre au nombre de caractéristiques (features) dans les données d'entrée, tandis que le nombre de neurones dans la couche de sortie dépend du type de tâche. Pour une classification binaire, 1 neurone avec une fonction d'activation sigmoïde est nécessaire, pour une classification multi-classe, il faut un neurone par classe avec une fonction d'activation softmax, et pour une régression, il faut 1 neurone (ou plus, selon la nature du problème) sans fonction d'activation ou avec une activation linéaire.

Couches Cachées :

Dans un réseau de neurones, les couches cachées sont des couches intermédiaires situées entre la couche d'entrée et la couche de sortie. Elles jouent un rôle crucial en permettant au réseau d'apprendre et de modéliser des relations complexes dans les données.

Le choix du nombre de neurones dans les couches cachées dépend de la complexité du problème et de la quantité de données disponibles.

Trop peu de neurones peuvent entraîner un sous-apprentissage (le modèle est trop simple pour capturer les motifs dans les données).

Trop de neurones peuvent entraîner un surapprentissage (le modèle est trop complexe et peut mémoriser le bruit dans les données d'entraînement).

3.6. RESEAU DE NEURONES ARTIFICIELS [14]

Un réseau neuronal artificiel, initialement inspiré par le fonctionnement des neurones biologiques, a progressivement évolué vers des approches plus proches des méthodes statistiques. Ces réseaux sont souvent perfectionnés via des méthodes d'apprentissage probabiliste, notamment bayésien, et se situent à l'intersection des applications statistiques et des techniques d'intelligence artificielle. Ils enrichissent les applications statistiques avec des paradigmes qui facilitent des classifications rapides, comme les réseaux de Kohonen, tout en offrant à l'intelligence artificielle un mécanisme perceptif qui fonctionne indépendamment des préconceptions de celui qui l'implémente, et qui ajoute des données d'entrée au raisonnement logique formel, comme illustré par l'apprentissage profond.

3.6.1. PREMIER RESEAU NEURONAL

McCulloch (qui voulait « comprendre comment on comprend ») et Pitts ont publié en 1943 l'un des articles considérés comme ayant lancé l'intelligence artificielle (en) Warren Sturgis McCulloch et Walter Pitts, « A logical calculus of the ideas immanent in nervous activity », Bulletin of Mathematical Biophysics, no 5, 1943, p. 115-133. Ils considèrent que le fonctionnement des neurones naturels évoque celui des portes logiques en mathématique. Mais ils ne donnent pas d'indication méthodologique pour adapter les coefficients synaptiques. Cette question au cœur des réflexions sur l'apprentissage a connu un début de réponse grâce aux travaux du physiologiste canadien Donald Hebb sur l'apprentissage en 1949 décrits dans son ouvrage The Organization of Behaviour. Hebb a proposé une règle simple permettant de modifier la valeur des « coefficients synaptiques » en fonction de l'activité des unités qu'ils relient. Cette règle aujourd'hui connue sous le nom de « règle de Hebb » est presque partout présente dans les modèles actuels, même les plus sophistiqués.

Cet article suscite une idée, qui se diffuse, notamment chez Frank Rosenblatt, qui en 1958 produit le modèle du perceptron. C'est le premier système artificiel capable d'apprendre par expérience, y compris quand son instructeur commet quelques erreurs (ce en quoi il diffère nettement d'un système d'apprentissage logique formel). Les médias américains, dont le New-York Times publient une vague d'articles enthousiastes annonçant pour bientôt une machine capable de tout reconnaitre, de penser, de marcher, se reproduire. Certains déclarent que le perceptron pourra bientôt faire de l'exploration spatiale. Le magazine Science dit que le perceptron pourra apprendre, prendre des décisions et traduire des langues.

Warren McCulloch, interviewé par la radio tv canada en 1969 (peu avant son décès) explique que :

« les machines informatiques pensent avec un langage qui est celui des nombres ; nous transformons tout en chiffre pour les faire fonctionner. Elles sont nulles en matière de perception. Elles n'ont pas de langage naturel sous-jacent, comme nous ou les animaux.

Lorsqu'on veut les faires développer un modèle de monde autour d'elles-mêmes, on s'aperçoit que ce n'est pas le bon dispositif.

Vous voulez qu'elles jouent aux échecs, très bien. Notre programme de jeux d'échec au MIT est déjà un joueur de 2ème rang ; dans un an je pense qu'il sera de 1er rang ; ce ne sera pas un maitre au bout d'un an mai dans 6 ans environ. Dans 6 ans, il deviendra une technologie de 1er rang qu'aucun humain ne pourra dépasser.

Je souhaite que ça aille plus loin, et ça ira plus loin, car nous pouvons réduire ces choses-là à des nombres. »

Cette même année 1969, un coup d'arrêt est donné à cette piste par Marvin Lee Minsky et Seymour Papert, qui estiment qu'imiter le cerveau est trop complexe. Ces derniers co-publient un ouvrage conséquent, pédagogique et apparemment complet, intitulé Perceptrons, mettant en exergue certaines les limitations théoriques, selon eux insurmontables, du perceptron, et plus généralement des classifieurs linéaires ; ces deniers ne peuvent pas traiter des problèmes non linéaires ou de connexité (par exemple, le cas de la fonction XOR). Minsky et Paperts étendent implicitement ces limitations à tous modèles de réseaux de neurones artificiels. Paraissant alors dans une impasse, la Recherche sur les réseaux de neurones perd l'essentiel de ses financements publics, et le secteur industriel s'en détourna aussi. Les fonds destinés à l'intelligence artificielle furent redirigés plutôt vers la logique formelle. Cependant, les solides qualités de certains réseaux de neurones en matière adaptative (exemple : Adaline), leur permettant de modéliser de façon évolutive des phénomènes eux-mêmes évolutifs, les amèneront à être intégrés sous des formes plus ou moins explicites dans le corpus des systèmes adaptatifs ; utilisés dans le domaine des télécommunications ou celui du contrôle de processus industriels.

En 1982, John Joseph Hopfield donna un nouveau souffle au cadre neuronal en publiant un article introduisant un nouveau modèle de réseau de neurones dit de Hopfield. Cet article eut du succès pour plusieurs raisons, dont la principale était de teinter la théorie des réseaux de neurones de la rigueur propre aux physiciens. Le neuronal redevint un sujet d'étude acceptable, bien que le modèle souffrît des principales limitations des modèles des années 1960, notamment l'impossibilité de traiter les problèmes non linéaires.

3.6.2. STRUCTURE DES RESEAUX

Les réseaux de neurones artificiels sont des modèles informatiques conçus pour imiter la structure et le fonctionnement du cerveau humain. Ils sont composés d'un ensemble d'unités de traitement appelées neurones, organisés en couches, qui transmettent et transforment les informations.

3.6.3. COUCHES DE NEURONES

La structure typique d'un réseau neuronal comprend trois types de couches :

La couche d'entrée : Cette couche reçoit les données initiales. Chaque neurone dans cette couche représente une caractéristique spécifique de l'entrée.

Les couches cachées : Entre la couche d'entrée et de sortie, il peut y avoir une ou plusieurs couches cachées. Ces couches traitent les signaux provenant de la couche d'entrée et effectuent des transformations complexes grâce à des fonctions mathématiques. Elles sont essentielles pour apprendre des caractéristiques et des représentations non linéaires des données.

La couche de sortie : La dernière couche produit le résultat final du réseau, qui peut être une classification, une prédiction, etc.

3.6.4. CONNEXIONS ET POIDS

Les neurones au sein des différentes couches sont interconnectés. Chaque connexion entre deux neurones est associée à un poids qui modifie la force du signal transmis. Ces poids sont ajustés lors de l'apprentissage du réseau pour minimiser les erreurs de prédiction.

3.6.5. FONCTIONS D'ACTIVATION

Les fonctions d'activation sont utilisées pour introduire de la non-linéarité dans le processus de traitement, permettant ainsi au réseau de modéliser des relations complexes entre les entrées et les sorties. Des exemples courants incluent la fonction ReLU, la sigmoïde et la tangente hyperbolique.

3.6.6. PROCESSUS D'APPRENTISSAGE

L'apprentissage dans un réseau neuronal se fait généralement par un algorithme appelé rétropropagation du gradient. Lors de cet apprentissage, le réseau ajuste ses poids pour minimiser la différence entre la sortie prévue et la sortie réelle, en utilisant une fonction de coût, souvent via une méthode d'optimisation comme la descente de gradient.

3.6.7. VARIANTES DE RESEAUX

Il existe de nombreuses variantes de réseaux de neurones, adaptées à différents types de problèmes :

Réseaux de neurones convolutifs (CNN) : Spécialement conçus pour le traitement des images, où les neurones sont arrangés de manière à reconnaître des patterns spatiaux.

Réseaux de neurones récurrents (RNN) : Idéaux pour les données séquentielles comme le texte et le son, ces réseaux ont des connexions qui bouclent, permettant de maintenir une forme de mémoire.

4. METHODOLOGIE DE RECHERCHE

Pour cette activité de recherche, nous avons mis en place une méthodologie de recherche afin de déterminer une solution.

Nous commençons par récupérer les données globales de notre jeu de données, notre modèle sera entrainé en utilisant ces données.

Comme l'optimisation des hyperparamètres d'un modèle de machine learning, en particulier des réseaux de neurones comme les LSTM, est souvent un problème complexe et non linéaire, avec de nombreux minima locaux, nous mettons ensuite en place des métaheuristiques afin de trouver un optimum global à la fonction objective. Les métaheuristiques permettent de rechercher simultanément plusieurs solutions potentielles, augmentant ainsi les chances de trouver une solution proche de l'optimal global plutôt que de se limiter à des solutions locales suboptimales.

Pour ne pas surcharger notre GPU, et ainsi augmenter considérablement le temps d'exécution, nous avons choisi un échantillon. Pour ce faire nous avons choisi aléatoirement 20% des données de chacun des bâtiments du dataset, nous conservons donc la proportion de chacun d'entre eux et garantissons un échantillon représentatif.

Une fois que nous avons nos métaheuristiques prêtes avec les données que nous souhaitons utiliser, nous devons sélectionner un critère afin de quantifier la qualité de notre modélisation. Nous avons choisi l'erreur quadratique moyenne (Mean-Squared Error ou MSE en anglais) car il s'agit d'un critère facilement interprétable (étant la différence au carré entre les valeurs prédites par le modèle et les valeurs réelles), une MSE plus faible indique une meilleure performance du modèle. De plus, avec la MSE les erreurs sont élevées au carré avant d'être moyennées, elle donne un poids plus important aux erreurs plus grandes. Cela signifie que les modèles avec des erreurs systématiquement grandes sur quelques points de données seront pénalisés plus sévèrement.

Une potentielle piste serait de prendre en compte le temps d'exécution pour chaque algorithme et ainsi comparé le temps nécessaire pour obtenir ces résultats. Cependant, cette piste ne peut pas être prise en compte par notre équipe car nous avons travaillé sur plusieurs machines physiques et virtuelles avec des capacités différentes, ce qui fausserait complètement les résultats

5. ANALYSE DES RESULTATS

Après avoir utilisé nos métaheuristiques nous avons obtenu des combinaisons d'hyper paramètres qui nous permettraient d'optimiser notre modèle. Vous trouverez ci-dessous deux exemples de combinaisons d'hyper paramètres que nos modèles nous ont fournis.

Pour le recuit simulé nous avons eu les combinaisons suivantes :

LSTM1 units	152
LSTM1 activation	linear
Dropout 1 rate	0.11
LSTM 2 unit	240
LSTM 2 activation	Tanh
DROPOUT 2 rate	0.1
DENSE 1 units	232
DENSE 1 activation	Tanh
DENSE 2 units	1
DENSE 2 activation	Linear
Optimizer learning rate	0.0005
EPOCHS	370
BATCH SIZE	112

MSE de 3.92 obtenue sur les 20% de données choisies

MSE de 6.97 avec ces paramètres sur le dataset complet.

Avec notre PSO, nous avons les valeurs suivantes :

LSTM1 units	188
LSTM1 activation	Tanh
Dropout 1 rate	0.0
LSTM 2 unit	169
LSTM 2 activation	sigmoid
DROPOUT 2 rate	0.0
DENSE 1 units	124

DENSE 1 activation	Sigmoid
Optimizer learning rate	0.001
EPOCHS	381
BATCH SIZE	100

Nous obtenons une MSE de 4.4

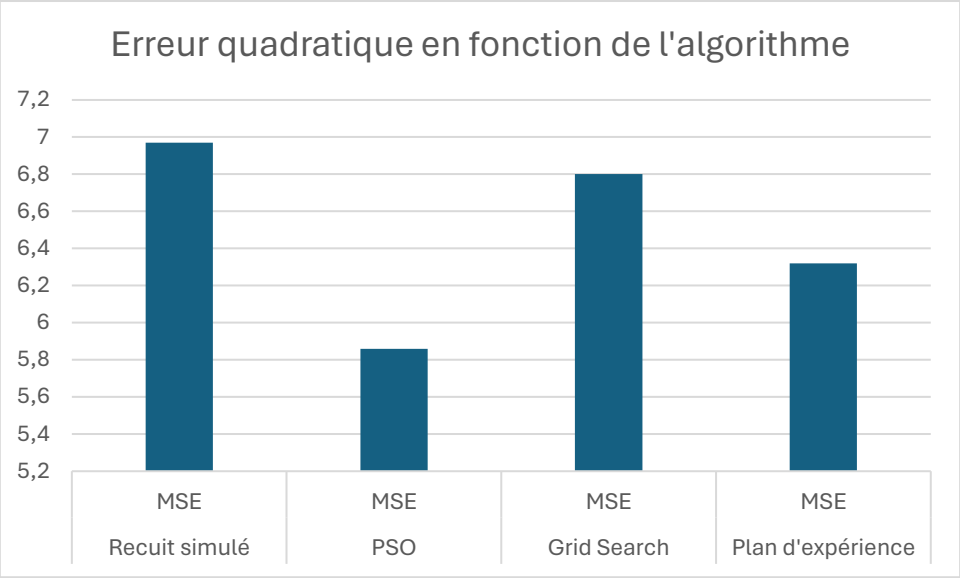
Enfin, le plan d'expérience avec un plan fractionnaire L8(2^7) :

LSTM1 units	320
LSTM1 activation	Tanh
Dropout 1 rate	0.1
LSTM 2 unit	208
LSTM 2 activation	Tanh
DROPOUT 2 rate	0.1
Learning rate	0.001

Nous obtenons une MAE de 3.5

Nous répétons cette méthode sur l'ensemble de nos algorithmes et nous récupérons l'erreur quadratique moyenne ressortie par notre modèle.

Afin de mettre en exergue les résultats obtenus, nous avons construit ce diagramme représentant l'erreur quadratique moyenne en fonction de chaque algorithme utilisé.


Erreur quadratique en fonction de l'algorithme
7,2
7
6,8
6,6
6,4
6,2
6
5,8
5,6
5,4
5,2
MSE
Recuit simulé
MSE
PSO
MSE
Grid Search
MSE
Plan d'expérience

6. CONCLUSION

Cette étude a exploré l'utilisation de métaheuristiques pour optimiser les hyperparamètres d'un modèle de réseau de neurones LSTM dans le but de prédire avec précision la production d'énergie photovoltaïque des bâtiments. Plusieurs algorithmes métaheuristiques, notamment le recuit simulé, l'optimisation par essaim de particules (PSO) et les algorithmes génétiques, ont été appliqués pour rechercher les combinaisons d'hyperparamètres optimales minimisant l'erreur quadratique moyenne (MSE) sur un échantillon représentatif des données.

Les résultats obtenus démontrent l'efficacité des métaheuristiques pour améliorer les performances du modèle LSTM. En comparant les valeurs de MSE pour chaque algorithme, nous avons pu identifier les configurations d'hyperparamètres les plus prometteuses. Le recuit simulé a fourni une MSE de 3,92 sur l'échantillon de données et de 6,97 sur l'ensemble des données, tandis que le PSO a atteint une MSE de 4,4. Ces résultats représentent une amélioration significative par rapport aux méthodes d'optimisation traditionnelles.

Cette approche offre une solution robuste et adaptative pour prédire la production d'énergie photovoltaïque dans divers environnements urbains, en tenant compte des variations météorologiques et des configurations des bâtiments. Les implications sont importantes pour le développement de bâtiments à énergie positive, facilitant l'adoption de technologies photovoltaïques optimisées et contribuant à des stratégies énergétiques durables en milieu urbain.

Bien que prometteurs, ces résultats ouvrent la voie à de nouvelles pistes de recherche. L'exploration d'autres métaheuristiques ou l'hybridation de plusieurs techniques pourrait permettre d'atteindre des performances encore supérieures. De plus, l'intégration de variables supplémentaires, telles que les caractéristiques architecturales des bâtiments, pourrait enrichir le modèle et améliorer davantage sa précision.

En somme, cette étude démontre le potentiel des métaheuristiques pour l'optimisation des hyperparamètres dans les réseaux de neurones, offrant une approche puissante pour relever les défis de la prédiction de la production d'énergie photovoltaïque dans un contexte urbain complexe.

7. BIBLIOGRAPHIE

[1] Dimitris Bertsimas. John Tsitsiklis. "Simulated Annealing." Statist. Sci. 8 (1) 10 - 15, February, 1993. »

[2] Xin-She Yang. Nature-Inspired Optimization Algorithms. Livre 2021 [Online] Disponible sur : <<https://www.sciencedirect.com/book/9780128219867/nature-inspired-optimization-algorithms>> visité le 17 mai 2024

[3] Adriam Tam. “A Gentle Introduction to Particle Swarm Optimization” Disponisble sur : https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/ . Visité le 18 mai

[4] Laurent Muller. « LES PLANS D’EXPERIENCES METHODE TAGUCHI » Présentation 2020, visité le 18 mai 2024

[5] Mirza RAHIM BAIG, Thomas V. JOSEPH, Nipun SADVILKAR, Mohan KUMAR SILAPARASETTY, Anthony SO. THE DEEP LEARNING WORKSHOP. Packt juillet 2020. [Online] Disponible sur : <<https://univ.scholarvox.com/reader/docid/88900550/page/74>>. Visité le 17 mai 2024.

[6] Mohan KUMAR SILAPARASETTY. Beginning with Deep Learning Using TensorFlow. BPB 2022. [Online] Disponible sur : <<https://univ.scholarvox.com/reader/docid/88937931/page/122>>. Visité le 17 mai 2024.

[7] Yugesh VERMA. Dense Layers. Analytics India Magazine 10 avril 2024. [Online] Disponible sur : <<https://analyticsindiamag.com/topics/what-is-dense-layer-in-neural-network/>>. Visité le 17 mai 2024.

[8] Raphael KASSEL. Recurrent Neural Network (RNN) : de quoi s’agit-il ?. DataScientest 2 juillet 2021. [Online] Disponible sur : <<https://datascientest.com/recurrent-neural-network>>. Visité le 17 mai 2024.

[9] Equipe Blent. Réseaux récurrents (RNN) : on vous explique tout. Blent 28 juin 2022. [Online] Disponible sur : <<https://blent.ai/blog/a/rnn-on-vous-explique-tout>>. Visité le 17 mai 2024.

[10] Christopher OLAH. Understanding LSTM Networks. Colah’s blog 27 août 2015. [Online] Disponible sur : <<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>>. Visité le 17 mai 2024.

[11] Daniel. Long Short Term Memory (LSTM) : de quoi s’agit-il ?. Datascientest 2 octobre 2023. [Online] Disponible sur : <<https://datascientest.com/long-short-term-memory-tout-savoir>>. Visité le 17 mai 2024.

[12] Rebeen HAMAD. What is LSTM? Introduction to Long Short-Term Memory. Medium 3 décembre 2023. [Online] Disponible sur : <<https://medium.com/@rebeen.jaff/what-is-lstm-introduction-to-long-short-term-memory-66bd3855b9ce>>. Visité le 17 mai 2024.

[13] Documentation Tensorflow. Disponible sur : https://www.tensorflow.org/?hl=en Visité le 20 mai 2024

[14] Kevin SWINGLER, « APPLYING NEURAL NETWORKS, A Practical Guide ». Disponible sur : <<https://books.google.fr/books?hl=fr&lr=&id=bq0YnP4BNKsC&oi=fnd&pg=PP7&dq=neuronal+networks+base&ots=BYDwMWaLMj&sig=w6Qo4BawA64RfLDgekeZC4xFe5A&redir_esc=y#v=onepage&q&f=false>>. Visité le 18 mai 2024