

Projet de Calcul Parallèle

Prédiction de dépense Energétique dans les
bâtiments, avec : CUDA, PSO, Réseaux de
Neurones



Réalisé par : Meriem ZOGARH
Youssef OUKASSOU



Encadré par : Pr. Hala KHANKHOUR

RÉSUMÉ

L'objectif principal de ce projet est d'explorer les avantages de l'utilisation de la technologie CUDA dans le domaine de l'apprentissage automatique appliquée à la prédiction des dépenses énergétiques dans les bâtiments. L'énergie étant un enjeu majeur de notre époque, il est crucial de développer des modèles de prédiction précis pour optimiser l'utilisation de l'énergie dans les bâtiments.

Pour atteindre cet objectif, nous avons combiné plusieurs approches avancées, notamment l'algorithme d'optimisation par essaim de particules (PSO) et les réseaux de neurones. Le PSO est utilisé pour ajuster les paramètres du réseau de neurones afin d'obtenir les meilleures performances de prédiction possibles.

L'une des contributions majeures de ce projet réside dans l'utilisation de CUDA, une plateforme de calcul parallèle de NVIDIA, pour accélérer les calculs intensifs nécessaires à l'entraînement des réseaux de neurones. Grâce à la parallélisation offerte par CUDA, nous avons pu réduire considérablement le temps nécessaire à l'apprentissage du modèle, permettant ainsi d'explorer plus rapidement différentes configurations et d'optimiser les performances globales.

Les principales étapes de ce projet comprennent la collecte de données sur la consommation énergétique des bâtiments, la conception et l'entraînement du réseau de neurones, l'implémentation de l'algorithme PSO pour optimiser les paramètres du réseau, et enfin, l'exploitation de CUDA pour accélérer l'entraînement du modèle.

Les résultats préliminaires montrent que notre approche combinant CUDA, PSO et les réseaux de neurones offre une amélioration significative des performances de prédiction par rapport aux méthodes traditionnelles. Les avantages de la parallélisation GPU offerts par CUDA se sont révélés cruciaux pour obtenir des résultats en temps réel, ce qui pourrait avoir un impact positif sur la gestion de l'énergie dans les bâtiments.

Ce projet démontre ainsi le potentiel de CUDA pour accélérer les tâches d'apprentissage automatique dans le domaine de la prédiction énergétique, ouvrant la voie à des applications plus avancées de l'informatique parallèle dans des domaines similaires.

TABLE DES MATIERES

Résumé.....	2
Table des Figures.....	6
Chapitre 1: Le site NVIDIA	8
Présentation de NVIDIA.....	8
Historique	8
Spécialité	8
Technologie CLés	8
Composants principale du site Nvidia	9
Rubrique « Pour Vous ».....	9
Rubrique « Industries »	9
Rubrique « Solutions »	10
Chapitre 2: Installation Cuda	11
Présentation de la technologie CUDA	11
étape d'installation de CUDA sur WIndows 10	11
1 ^{er} étape : Télécharger Cuda Toolkit & cuDnn.....	11
2 ^{ème} étape : Installer CUDA	14
3 ^{ème} étape : Extrayer le fichier d'archive cuDNN	15
4 ^{ème} étape : placement des fichiers de configurations.....	16
5 ^{ème} étape : ajouter les variables d'environnement	16
6 ^{ème} étape : télécharger et installer visual studio.....	17
7 ^{ème} étape : tester cuda.....	18
Chapitre 3 : Technologies utilisées	19
Réseau de neurones.....	19
L'algorithme PSO.....	20
PSO avec Réseau de neurones	21

CUDA (Compute Unified Device Architecture	24
Utilisation de CUDA pour l'accélération des calculs	25
Vérification de la compatibilité matérielle avec CUDA	26
Réseau de neurones + CUDA + PSO	27
Chapitre 4:Mise en pratique du projet.....	29
Origine des données.....	29
Description de la source de données	29
Signification des attributs.....	30
Objective de prédiction.....	31
Preprocessing.....	32
gestion des valeurs manquantes	33
Corrélation.....	33
Feature selection.....	37
Data splitting et normalisation	37
Création du modèle de réseau de neurones	38
Initialisation du réseau de neuones	38
Création des couches du réseau de neurones	39
Initialisation de l'optimisateur et de la fonction de perte.....	39
Stockage du poids de régularisation L2 :.....	40
Impression des informations sur les couches.....	40
Méthode « forward »	41
Méthode ‘compute_regularization_loss	41
Création de la fonction objective	42
Préparation des données et de l'initialisation.....	42
Définition de la fonction objectif	42
Optimisation avec PSO	44
Définition des plages d'hyperparamètres	44

Configuration et exécution de l'optimisateur PSO	45
Tester les meilleurs hyperparamètres.....	46
Résultat d'optimisation.....	46
Comparaison d'exécution avec cuda et son cuda.....	46
évaluation du modèle	47
Visualisation des résultats.....	49
Valeur actuel VS. Valeur actuel (Scatter plot)	49
évolution des pertes d'entraînement et de validation.....	50
Comparaison des valeurs réelles et prédictives.....	51
Conclusion	53
Références.....	54

TABLE DES FIGURES

Figure 1: rubrique "Pour vous" dans le site Nvidia	9
Figure 2: Rubrique " Industries " dans le site Nvidia	10
Figure 3: Rubrique "Solutions" du site Nvidia.....	10
Figure 4: Télécharger Cuda Toolkit	11
Figure 5: accéder au site de téléchargement de cuDNN.....	13
Figure 6: accès non requis sans compte.....	13
Figure 7: Création de compte sur le site Nvidia.....	13
Figure 8: télécharger cuDNN	14
Figure 9: choisir le path d'extraction de cuda	14
Figure 10: extraction en cours	14
Figure 11: choisir l'option d'installation	15
Figure 12: extraction du dossier cuDNN.....	15
Figure 13: les dossiers bin, include et lib de CUDA et cuDNN.....	16
Figure 14: Modifier la variable d'environnement	16
Figure 15: télécharger Visual Studio 2019	17
Figure 16: télécharger les packages C++ avec l'installation de Visual Studio Community 2019.	17
Figure 17: Code CUDA	18
Figure 18: compilation et exécution du code CUDA	18
Figure 19: Architecture d'un réseau de neurones.....	19
Figure 20: fonctionnement de l'algorithme PSO	21
Figure 21: PSO avec Réseau de neurones.....	23
Figure 22: Flux de traitement dans CUDA	24
Figure 23: GPU vs CPU (part 1)	25
Figure 24: GPU vs CPU (part 2)	26
Figure 25: présentation des attributs de la source de données	29
Figure 26: importer packages et data source	32
Figure 27: information sur les colonnes de la data source	32
Figure 28: Handling missing values.....	33
Figure 29: répartition des colonnes de la donnée source	33
Figure 30: matrice de corrélation du 1er groupe.....	34
Figure 31: Matrice de corrélation du 2ème groupe.....	34
Figure 32: Matrice de corrélation du 3ème groupe.....	35

Figure 33: Matrice de corrélation du 4ème groupe.....	35
Figure 34: Matrice de corrélation du 5ème groupe.....	36
Figure 35: Matrice de corrélation du 6ème groupe.....	36
Figure 36: sélection de caractéristiques basée sur la corrélation.....	37
Figure 37: La classe 'NeuralNetwork'.....	38
Figure 38: Création des couches de réseau de neurones	39
Figure 39: Initialisation de l'optimiseur et de la fonction perte.....	39
Figure 40: Stockage du poids de régularisation L2	40
Figure 41: Impression des informations sur les couches	40
Figure 42: Méthode "forward"	41
Figure 43: Méthode "compute_regularization_loss"	41
Figure 44: préparation et initialisation des données.....	42
Figure 45: définition de la fonction objective	42
Figure 46: configuration d'hyperparamètres	42
Figure 47: Initialisation du modèle	43
Figure 48: préparation des données avec cuda.....	43
Figure 49: Initialisation de l'optimiseur et de la fonction perte.....	43
Figure 50: entraînement du modèle.....	43
Figure 51: évaluation du modèle sur les données de validation	44
Figure 52: définition des plages d'hyperparamètres	44
Figure 53: configuration et exécution de l'optimiseur pso.....	45
Figure 54: tester les meilleurs hyperparamètres	46
Figure 55: résultat de l'optimisation	46
Figure 56: PSO + ANN sans CUDA	47
Figure 57: évaluation du modèle.....	48
Figure 58: graphe de dispersion.....	49
Figure 59: évolution des pertes d'entraînement et de validation.....	50
Figure 60: comparaison des valeurs réelles et prédites	51

CHAPITRE 1: LE SITE NVIDIA

PRÉSENTATION DE NVIDIA

Nvidia Corporation est une entreprise de technologie de renommée mondiale qui se concentre principalement sur le développement de matériel informatique, de logiciels et de technologies liées au traitement graphique, à l'intelligence artificielle (IA) et au calcul haute performance.

HISTORIQUE

Nvidia a été fondée en 1993 par Jensen Huang, Chris Malachowsky et Curtis Priem. L'entreprise a son siège à Santa Clara, en Californie, aux États-Unis.

SPECIALITE

CARTES GRAPHIQUES

Nvidia est particulièrement connue pour ses cartes graphiques, y compris la célèbre gamme GeForce destinée aux joueurs et les cartes Quadro pour les professionnels de la création graphique et du design.

INTELLIGENCE ARTIFICIELLE

Nvidia est un leader dans le domaine de l'AI, grâce à ses GPU spécialement conçus pour les tâches d'apprentissage automatique et d'AI. Les GPU Tesla et les GPU A100 sont largement utilisées pour l'entraînement de modèles d'IA.

CALCUL HAUTE PERFORMANCE

Les produits Nvidia sont également utilisés pour des applications de calcul haute performance, notamment la simulation, la modélisation, et la recherche scientifique.

TECHNOLOGIE CLES

Parmis les technologies de Nvidia on retrouve : CUDA, Ray Tracing en temps réel, DLSS (Deep Learning Super Sampling) et Nvidia CUDA AI

CUDA

Nvidia a développé CUDA (Compute Unified Device Architecture), une plateforme de calcul parallèle qui permet aux développeurs d'utiliser la puissance de calcul des GPU Nvidia pour des applications de calcul général.

NVIDIA CUDA AI

Nvidia propose également des logiciels et des bibliothèques d'apprentissage automatique, ainsi que des plates-formes d'IA comme Nvidia DGX pour soutenir les chercheurs et les entreprises dans leurs projets d'IA

COMPOSANTS PRINCIPALE DU SITE NVIDIA

RUBRIQUE « POUR VOUS »

La section "Pour Vous" du site Nvidia a été la plus précieuse en explorant le site. Cette rubrique offre une personnalisation de l'expérience utilisateur en fournissant des informations spécifiques en fonction de ses besoins et de ses intérêts. On y trouve des recommandations de produits et de services qui correspondent à ses préférences, que ce soit pour le gaming, le développement professionnel, ou l'intelligence artificielle. Cette fonctionnalité simplifie la navigation sur le site en permettant de trouver rapidement les informations les plus pertinentes pour des besoins spécifiques.

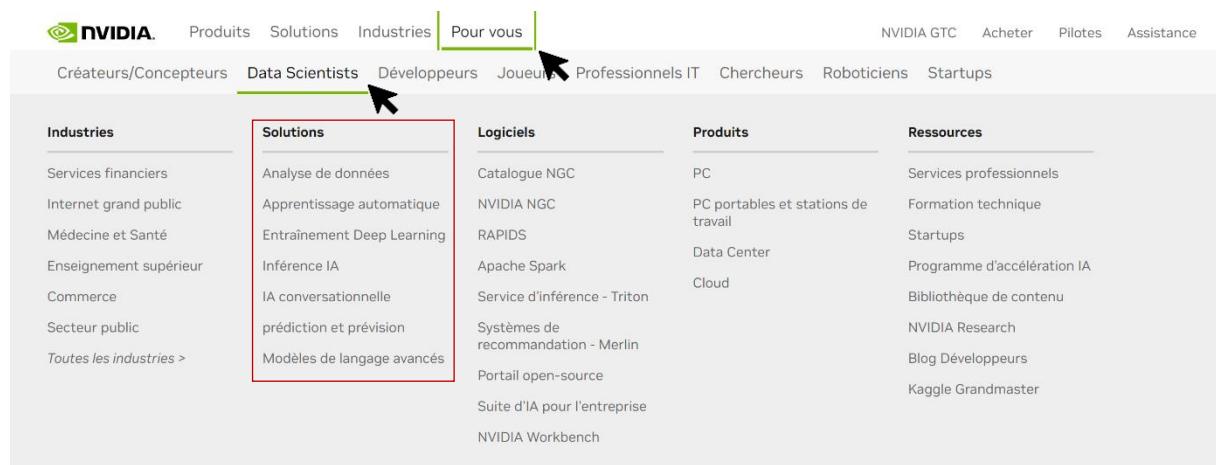


Figure 1: rubrique "Pour vous" dans le site Nvidia

RUBRIQUE « INDUSTRIES »

La rubrique "Industries" du site Nvidia est une section essentielle qui met en avant la polyvalence des solutions de Nvidia dans divers secteurs. Cette section offre un aperçu détaillé de la manière dont les produits Nvidia sont utilisés pour répondre aux besoins spécifiques de différentes industries. En explorant cette rubrique, les visiteurs peuvent découvrir comment les GPU Nvidia, les technologies d'intelligence artificielle et de calcul haute performance sont déployés pour résoudre des problèmes complexes dans des domaines tels que la santé, les sciences, l'automobile, l'énergie et bien d'autres. Les études de cas, les témoignages de clients et les exemples concrets illustrent comment les technologies Nvidia contribuent à des avancées

significatives et à des améliorations dans chaque secteur, montrant ainsi l'impact concret et la pertinence de ces solutions au-delà du simple jeu vidéo et de la conception graphique. En bref, la rubrique "Industries" du site Nvidia démontre l'engagement de l'entreprise à répondre aux besoins spécifiques de chaque domaine en offrant des solutions sur mesure et performantes.

NVIDIA GTC Acheter Pilotes Assistance

Industries

Aperçu	Recherche énergétique	Production industrielle	Robotics
Architecture, Ingénierie, Construction et Opérations	Services financiers	Médias et divertissement	Villes intelligentes
Automobile	Santé et sciences de la vie	Services publics mondiaux	Supercomputing
Internet grand public	Enseignement supérieur	Restaurants	Télécommunications
Cybersécurité	Développement de jeux	Commerce et CPG	Transport

Figure 2: Rubrique "Industries" dans le site Nvidia

RUBRIQUE « SOLUTIONS »

La rubrique "Solutions" du site Nvidia est une ressource essentielle qui présente des solutions technologiques complètes et adaptées à divers besoins et industries. Cette section vise à simplifier la recherche de solutions spécifiques en regroupant les offres de Nvidia par domaine d'application. Elle permet aux visiteurs de découvrir comment les produits Nvidia, tels que les cartes graphiques, les GPU d'IA, les logiciels et les services, sont utilisés pour résoudre des problèmes concrets et relever des défis spécifiques dans des domaines tels que la médecine, l'automobile, la finance, l'architecture, la recherche scientifique, etc.

Chaque sous-catégorie de la rubrique "Solutions" offre une description détaillée des solutions disponibles, des études de cas, des témoignages de clients et des informations sur les avantages de l'utilisation des technologies Nvidia dans ces domaines. Les visiteurs peuvent ainsi obtenir une compréhension approfondie de la manière dont les produits et les services Nvidia peuvent être adaptés à leurs besoins, améliorant ainsi leur productivité, leur efficacité et leur capacité d'innovation.

NVIDIA GTC Acheter Pilotes Assistance

Solutions

IA et science des données	Data Center et Cloud Computing	Conception et simulation	Robotique et Edge Computing	Calcul haute performance	Véhicules autonomes
Aperçu	Aperçu	Aperçu	Aperçu	Aperçu	Aperçu
Analyse de données	Calcul accéléré pour l'informatique d'entreprise	Réalité virtuelle et augmentée	IA sur 5G	HPC et IA	Chauffeur
Apprentissage automatique	Cloud Computing	Multi-écrans	Analyse vidéo intelligente	Simulation et modélisation	Concierge
Entraînement Deep Learning	Colocation	Rendu graphique	Industrie	Visualisation scientifique	Entraînement
Inférence IA	Edge Computing	Métavers	Robotics	Gestion des déplACEMENTS sur l'Edge	Simulation
IA conversationnelle	Mise en réseau	Virtualisation graphique	Solutions Edge		Cartographie HD
Speech AI	Virtualisation	Simulations d'ingénierie			
Modèles de langage avancés	MLOps	Diffusion			

Figure 3: Rubrique "Solutions" du site Nvidia

CHAPITRE 2: INSTALLATION CUDA

PRÉSENTATION DE LA TECHNOLOGIE CUDA

CUDA (Compute Unified Device Architecture) est une plateforme de calcul parallèle développée par NVIDIA. Elle permet d'exploiter la puissance de calcul des cartes graphiques (GPU) pour accélérer un large éventail d'applications, en particulier les tâches intensives en calcul. CUDA repose sur un modèle de programmation parallèle qui permet aux développeurs de concevoir des programmes capables de tirer parti de milliers de coeurs de calcul présents dans les GPU modernes. En utilisant CUDA, les développeurs peuvent accélérer des applications dans des domaines tels que la modélisation et la simulation scientifique, l'apprentissage automatique, le traitement d'images, la recherche en intelligence artificielle, et bien plus encore. Grâce à son adoption généralisée et à son support continu, CUDA est devenu un outil essentiel pour les chercheurs, les développeurs et les ingénieurs cherchant à exploiter la puissance de calcul parallèle des GPU pour résoudre des problèmes complexes de manière plus efficace et plus rapide.

ÉTAPE D'INSTALLATION DE CUDA SUR WINDOWS 10

1^{ER} ÉTAPE : TÉLÉCHARGER CUDA TOOLKIT & CUDNN

TÉLÉCHARGER CUDA TOOLKIT

Pour télécharger le Cuda Toolkit on accède au site « nvidia.developer » <https://developer.nvidia.com/cuda-downloads>.

The screenshot shows the download page for the CUDA Toolkit. It has four filter sections: 'Operating System' (Windows), 'Architecture' (x86_64), 'Version' (10, 11, Server 2019, Server 2022), and 'Installer Type' (exe (local) selected). Below these is a green bar with the text 'Download Installer for Windows 10 x86_64'. A note says 'The base installer is available for download below.' Under 'Base Installer', there's an 'Installation Instructions' box with steps 1 and 2, and a 'Download (3.0 GB)' button.

Figure 4: Télécharger Cuda Toolkit

il suffit de choisir les préférences en termes de types de système d'exploitation, l'architecture de ce dernier ainsi que sa version, et finalement le type d'installation soit : local ou network.

Noté bien qu'il faut respecter les propriétés de votre machine, il faut choisir la même architecture et version que celle de votre système d'exploitation. En ce qui concerne le type d'installation il dépend principalement de l'environnement dans lequel vous travaillez et des besoins spécifiques de votre configuration. Pourtant il y a plusieurs raisons pourquoi le type local peut être préférable par rapport au type network :

1. **INDÉPENDANCE DU RÉSEAU** : L'installation en mode local ne nécessite pas de connexion Internet constante pendant le processus d'installation. Cela peut être particulièrement utile si vous travaillez dans un environnement où la connectivité Internet est limitée ou peu fiable.
2. **CONTRÔLE TOTAL** : L'installation en mode local vous donne un contrôle total sur les fichiers d'installation. Vous pouvez choisir où les fichiers seront stockés sur votre système, ce qui peut être utile si vous avez des préférences spécifiques pour la gestion de l'espace disque ou si vous devez personnaliser l'installation.
3. **SIMPLICITÉ** : L'installation en mode local tend à être plus simple, car elle ne nécessite généralement qu'un seul fichier d'installation exécutable. En revanche, l'installation en mode réseau peut impliquer le téléchargement de plusieurs fichiers, ce qui peut rendre le processus plus complexe.

TÉLÉCHARGER CUDNN

cuDNN (CUDA Deep Neural Network) est une bibliothèque logicielle développée par NVIDIA pour accélérer les opérations de calcul intensif dans les réseaux de neurones profonds (DNN) en utilisant des GPU NVIDIA. cuDNN optimise les opérations fréquemment utilisées dans les réseaux de neurones, telles que la convolution, la normalisation, la rétropropagation, et d'autres, en exploitant la puissance de calcul parallèle des GPU. Cette bibliothèque est largement utilisée dans le domaine de l'apprentissage profond (deep learning) pour accélérer la formation de modèles de réseaux de neurones, ce qui permet d'obtenir des performances nettement meilleures par rapport à l'utilisation de CPU seuls. En résumé, cuDNN est un outil essentiel pour accélérer l'apprentissage automatique et la recherche en intelligence artificielle en exploitant les capacités de calcul parallèle des GPU NVIDIA.

- Afin de télécharger le fichier de cuDNN on accède au site :
<https://developer.nvidia.com/cudnn>

NVIDIA cuDNN

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers.

Deep learning researchers and framework developers worldwide rely on cuDNN for high-performance GPU acceleration. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. cuDNN accelerates widely used deep learning frameworks, including Caffe2, Chainer, Keras, MATLAB, MxNet, PaddlePaddle, PyTorch, and TensorFlow. For access to NVIDIA optimized deep learning framework containers that have cuDNN integrated into frameworks, visit NVIDIA GPU CLOUD to learn more and get started.

[Download cuDNN](#)

[Developer Guide](#)

[Forums](#)

[Latest Release Notes](#)

Figure 5: accéder au site de téléchargement de cuDNN

Par contre la page de téléchargement de cuDNN nécessite une souscription, il faudra alors créer un compte sur la plateforme Nvidia.

NVIDIA Developer Program Membership Required

The file or page you have requested requires membership in the NVIDIA Developer Program. Please either log in or join the program to access this material. [Learn more](#) about the benefits of the NVIDIA Developer Program.

Figure 6: accès non requis sans compte

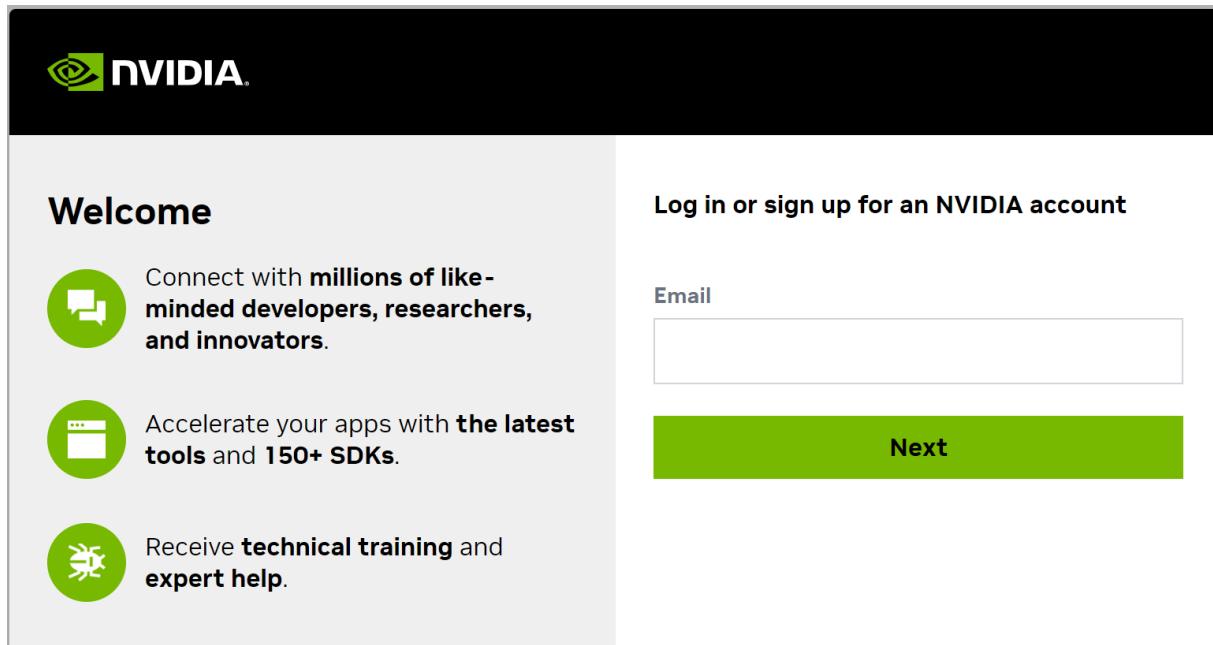


Figure 7: Création de compte sur le site Nvidia

Après la création de compte et la vérification de l'email, vous serez rediriger vers la page de téléchargement : <https://developer.nvidia.com/rdp/cudnn-download>, il suffit d'accepter les condition du contrat de la licence du logiciel cuDNN et de choisir la version souhaité.

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

Download cuDNN v8.9.4 (August 8th, 2023), for CUDA 12.x

Download cuDNN v8.9.4 (August 8th, 2023), for CUDA 11.x

[Archived cuDNN Releases](#)

Figure 8: télécharger cuDNN

2^{EME} ETAPE : INSTALLER CUDA

Après avoir télécharger le fichier .exe d'installation de cuda toolkit et le lancer une fenêtre de CUDA Setup package s'affiche.



Figure 9: choisir le path d'extraction de cuda

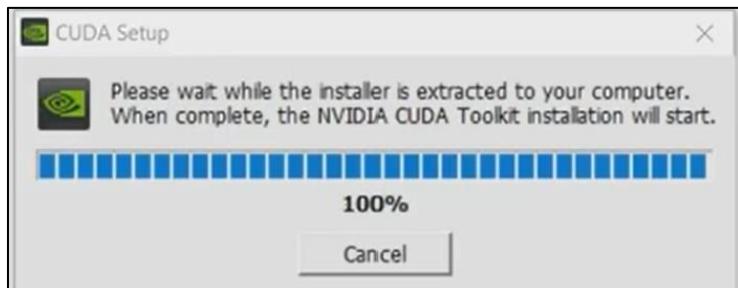


Figure 10: extraction en cours

Après la terminaison de l'extraction avec le CUDA Setup, une fenêtre NVIDIA Installer est lancé, d'abord il y aura une vérification du système afin de vérifier sa compatibilité, ensuite et après avoir accepter le contrat de la licence, il faut choisir l'option d'installation, soit : express ou custom, la version express est généralement recommandée.

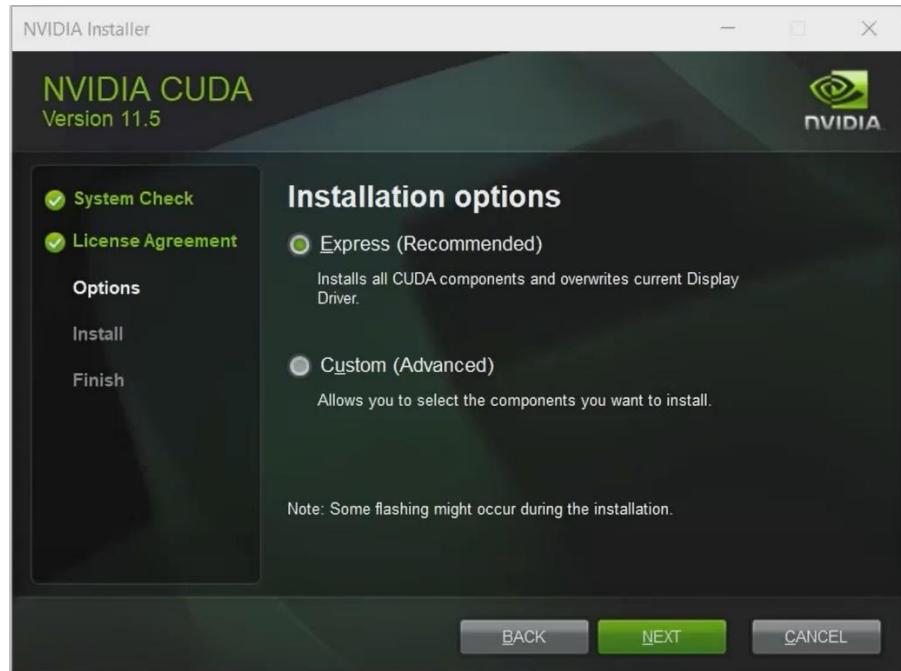


Figure 11: choisir l'option d'installation

Maintenant il suffit d'attendre la fin de l'installation qui peut prendre un peu de temps.

3^{ème} ÉTAPE : EXTRAYER LE FICHIER D'ARCHIVE CUDNN

On extrait le fichier zip de cuDNN dans le C-drive, vous pouvez changer le nom du dossier après extraction à cudnn pour y faciliter l'accès.

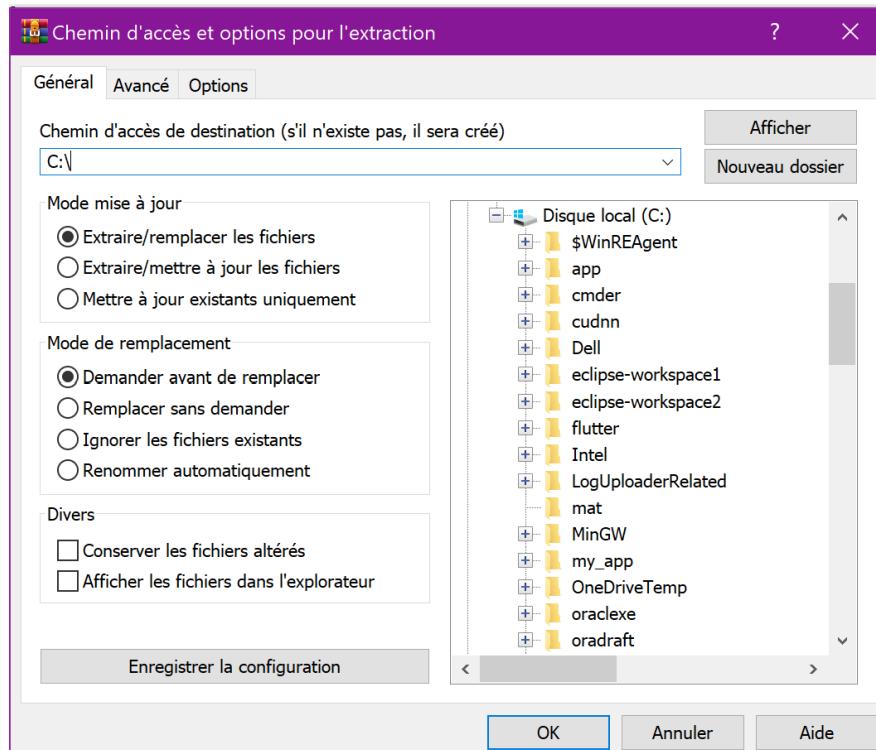


Figure 12: extraction du dossier cuDNN

4^{ÈME} ÉTAPE : PLACEMENT DES FICHIERS DE CONFIGURATIONS

Naviguer vers le dossier CUDA, le path de l'emplacement doit être un peu prêt comme celui-ci: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.2. Naviguer ensuite dans une autre fenêtre vers le dossier cuDNN C:\cudnn

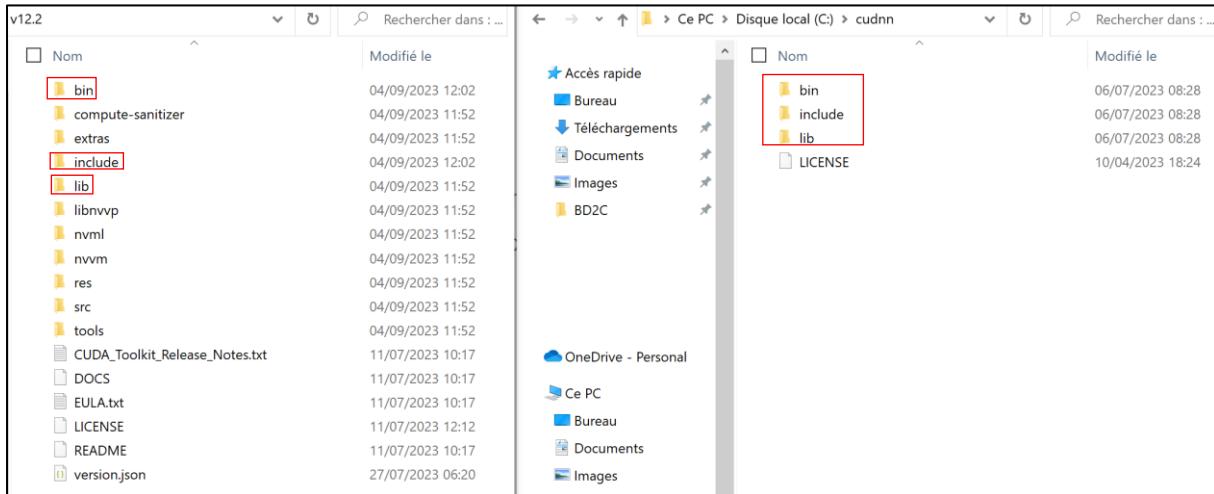


Figure 13: les dossiers bin, include et lib de CUDA et cuDNN

Il faut copier tous les fichiers dans les dossiers bin, include et lib de cuDNN et les coller respectivement dans les dossiers bin, include et lib de CUDA.

5^{ÈME} ÉTAPE : AJOUTER LES VARIABLES D'ENVIRONNEMENT

On doit ajouter deux variables d'environnement dans le path de la variable système.

Une contenant le path du dossier bin de cuda, soit : C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.2\bin

La deuxième contenant le path du dossier libnvvp de cuda, soit : C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.2\libnvvp

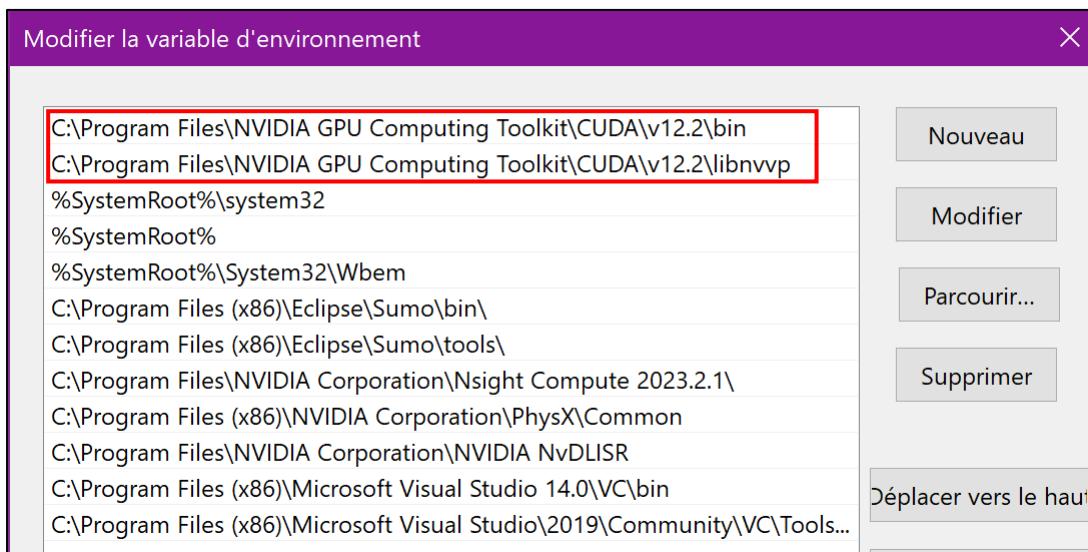


Figure 14: Modifier la variable d'environnement

6^{ÈME} ÉTAPE : TÉLÉCHARGER ET INSTALLER VISUAL STUDIO

La dernière version qui est compatible avec Nvidia est la version de 2019, alors lors de l'installation il faut chercher dans les archives des anciens version et non pas télécharger la dernière version disponible.

Pour accéder aux versions antérieures naviguer vers le site :

<https://visualstudio.microsoft.com/fr/vs/older-downloads/>



Figure 15: télécharger Visual Studio 2019

Choisissez la version Visual Studio Community 2019 qui est la version gratuite et disponible à tous.

Après le téléchargement et le lancement de fichier .exe, le Visual Studio Installer sera lancer, assurer vous avons d'installer de bien télécharger tout ce qui concerne le C++.

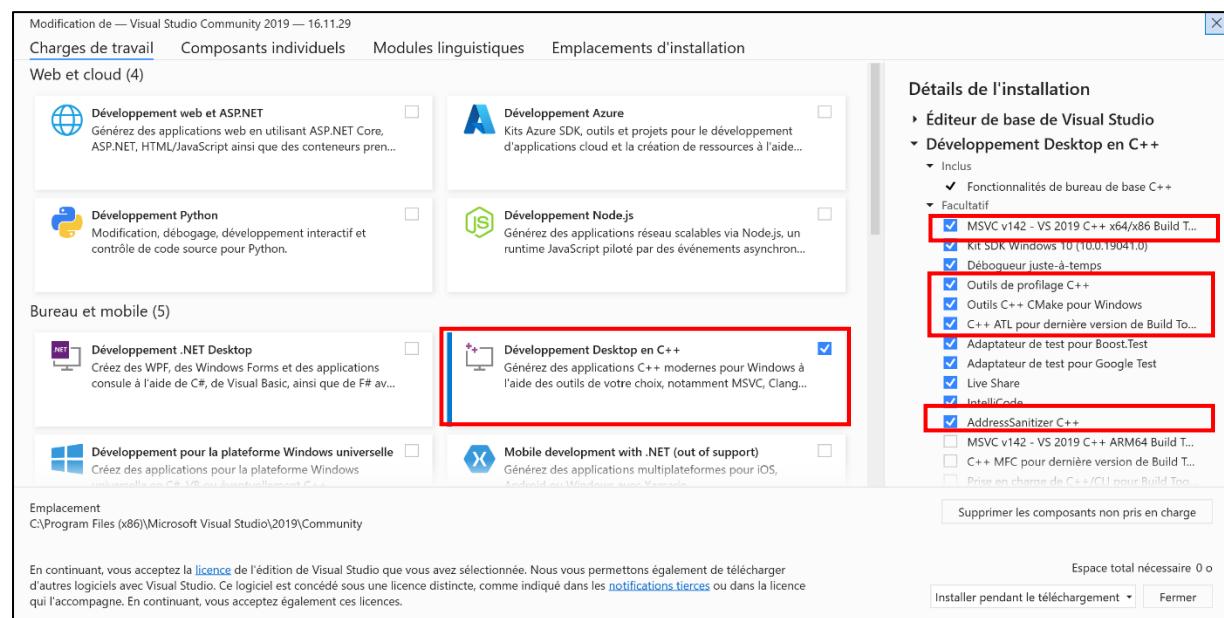


Figure 16: télécharger les packages C++ avec l'installation de Visual Studio Community 2019

7^{ÈME} ÉTAPE : TESTER CUDA

Afin de s'assurer que CUDA à bien été installer, on va tester avec un simple code

```
#include <iostream>

// Fonction CUDA pour ajouter deux tableaux
__global__ void addArrays(int *a, int *b, int *c, int size) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < size) {
        c[tid] = a[tid] + b[tid];
    }
}

int main() {
    const int arraySize = 10;
    int a[arraySize], b[arraySize], c[arraySize];
    int *dev_a, *dev_b, *dev_c;

    // Allouer de la mémoire sur le GPU
    cudaMalloc((void**)&dev_a, arraySize * sizeof(int));
    cudaMalloc((void**)&dev_b, arraySize * sizeof(int));
    cudaMalloc((void**)&dev_c, arraySize * sizeof(int));

    // Initialiser les tableaux a et b sur le CPU
    for (int i = 0; i < arraySize; i++) {
        a[i] = i;
        b[i] = i;
    }

    // Copier les tableaux a et b du CPU vers le GPU
    cudaMemcpy(dev_a, a, arraySize * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, arraySize * sizeof(int), cudaMemcpyHostToDevice);

    // Appeler la fonction CUDA pour ajouter les tableaux
    addArrays<<<1, arraySize>>>(dev_a, dev_b, dev_c, arraySize);

    // Copier le résultat du GPU vers le CPU
    cudaMemcpy(c, dev_c, arraySize * sizeof(int), cudaMemcpyDeviceToHost);

    // Afficher le résultat
    for (int i = 0; i < arraySize; i++) {
        std::cout << a[i] << " + " << b[i] << " = " << c[i] << std::endl;
    }

    // Libérer la mémoire sur le GPU
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);

    return 0;
}
```

Figure 17: Code CUDA

```
C:\Users\anasz\Downloads>nvcc -o test test.cu
test.cu
tmpxft_00001cd0_00000000-10_test.cudafe1.cpp
    Création de la bibliothèque test.lib et de l'objet test.exp

C:\Users\anasz\Downloads>test.exe
0 + 0 = 0
1 + 1 = 2
2 + 2 = 4
3 + 3 = 6
4 + 4 = 8
5 + 5 = 10
6 + 6 = 12
7 + 7 = 14
8 + 8 = 16
9 + 9 = 18
```

Figure 18: compilation et exécution du code CUDA

Comme On peut le constater le code CUDA se compile et s'exécute sans aucune erreur.

CHAPITRE 3 : TECHNOLOGIES UTILISÉES

Dans ce chapitre de réalisation du projet, nous explorons une approche de régression novatrice pour prédire la dépense énergétique dans les bâtiments. Notre solution repose sur un réseau de neurones optimisé par l'algorithme PSO (Particle Swarm Optimization), et grâce à l'utilisation de CUDA, nous parvenons à accélérer significativement les temps d'exécution, ouvrant ainsi la voie à une gestion énergétique plus efficace des bâtiments.

RÉSEAU DE NEURONES

Un réseau de neurones, également connu sous le nom de réseau neuronal artificiel (RNA) ou perceptron multicouche, est un modèle mathématique et informatique inspiré du fonctionnement du cerveau humain. Il est utilisé pour résoudre un large éventail de problèmes complexes, notamment la classification d'images, la reconnaissance de la parole, la traduction automatique, la prédiction de séries temporelles, et bien d'autres applications.

Le concept fondamental derrière un réseau de neurones est de simuler le comportement des neurones biologiques dans le cerveau. Un neurone biologique reçoit des signaux électriques de ses synapses, les traite et génère éventuellement un signal de sortie. De manière similaire, un neurone artificiel reçoit des entrées pondérées, les somme, les passe à travers une fonction d'activation, puis produit une sortie.

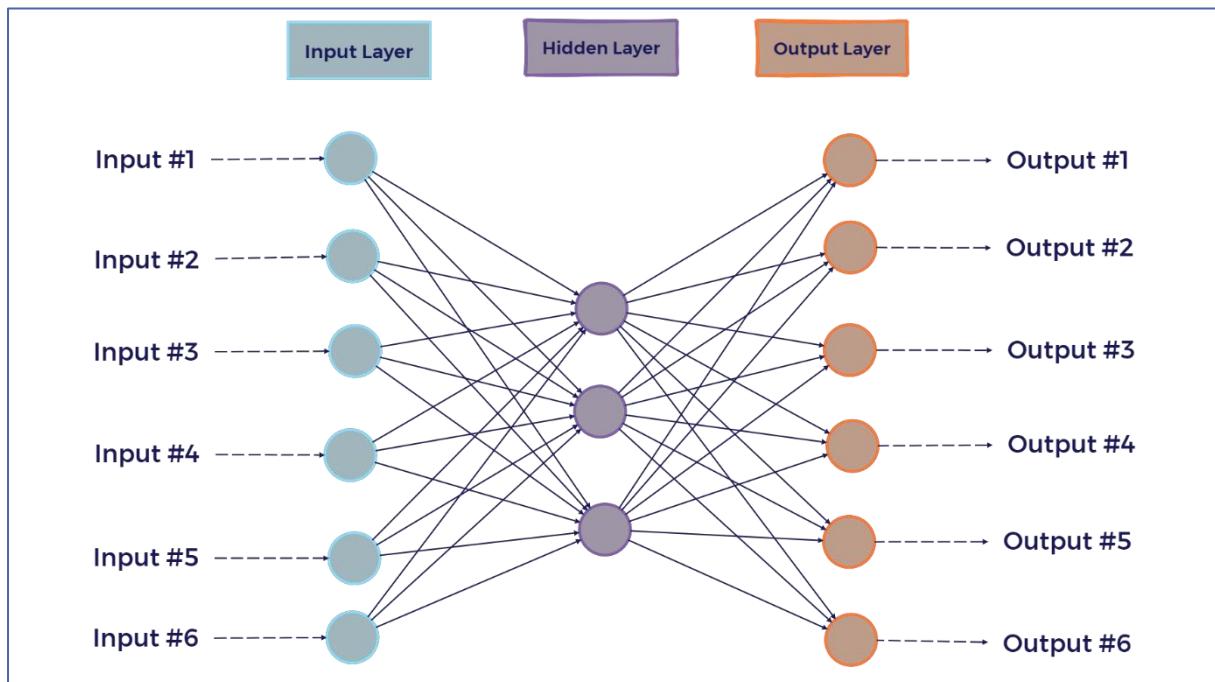


Figure 19: Architecture d'un réseau de neurones

Voici les principales composantes et étapes du fonctionnement d'un réseau de neurones :

- **LES COUCHES DE NEURONES** : Un réseau de neurones est généralement organisé en couches. La première couche est la couche d'entrée, où les données sont introduites. La dernière couche est la couche de sortie, qui produit les résultats du réseau. Entre ces deux couches, il peut y avoir une ou plusieurs couches cachées, qui permettent d'extraire des caractéristiques plus complexes des données.
- **LES POIDS ET LES BIAIS** : Chaque connexion entre les neurones est associée à un poids. Les poids déterminent l'importance de chaque entrée pour le neurone suivant. De plus, chaque neurone est associé à un biais, qui permet de déplacer la fonction d'activation et d'ajuster la sortie.
- **LA FONCTION D'ACTIVATION** : Après avoir sommé les entrées pondérées, chaque neurone applique une fonction d'activation. Cette fonction introduit une non-linéarité dans le modèle, ce qui lui permet d'apprendre des relations complexes dans les données. Les fonctions d'activation couramment utilisées incluent la fonction sigmoïde, la fonction ReLU (Rectified Linear Unit), et la fonction tanh (tangente hyperbolique).

L'ALGORITHME PSO

L'algorithme PSO (*Particle Swarm Optimization*) est une méthode d'optimisation inspirée par le comportement social des oiseaux et des poissons en groupe. Il fonctionne en simulant un ensemble de particules qui se déplacent dans un espace de recherche pour trouver la meilleure solution à un problème donné. Chaque particule représente une solution potentielle, et elles communiquent entre elles pour s'améliorer progressivement. L'algorithme PSO est largement utilisé pour résoudre divers problèmes d'optimisation, y compris l'optimisation de réseaux de neurones, où il peut être appliqué pour trouver les meilleurs paramètres afin d'améliorer les performances du réseau.

L'algorithme PSO fonctionne de manière simple et intuitive :

1. **INITIALISATION:** Des particules sont placées aléatoirement dans l'espace de recherche, chacune avec sa propre solution potentielle.
2. **ÉVALUATION:** Chaque particule évalue la qualité de sa solution en utilisant une fonction objectif.
3. **MISE À JOUR LOCALE:** Chaque particule compare sa performance actuelle avec la meilleure performance qu'elle ait jamais atteinte ("*pbest*"). Si elle trouve une meilleure solution, elle met à jour "*pbest*".

4. **MISE À JOUR GLOBALE:** Les particules partagent leurs meilleures solutions locales, et la meilleure de toutes est déterminée ("*gbest*").
5. **MISE À JOUR DES POSITIONS:** Les particules se déplacent vers "*gbest*" en ajustant leurs positions et leurs vitesses en fonction de "*pbest*" et "*gbest*".
6. **RÉPÉTITION:** Les étapes précédentes sont répétées pendant un certain nombre d'itérations ou jusqu'à ce qu'un critère d'arrêt soit atteint.
7. **RÉSULTATS:** À la fin, la meilleure solution trouvée ("*gbest*") représente la solution optimale ou de haute qualité pour le problème d'optimisation.

L'algorithme PSO est itératif et efficace pour explorer de grands espaces de recherche, ce qui en fait un outil puissant pour l'optimisation de diverses tâches, y compris l'optimisation des paramètres d'un réseau de neurones.

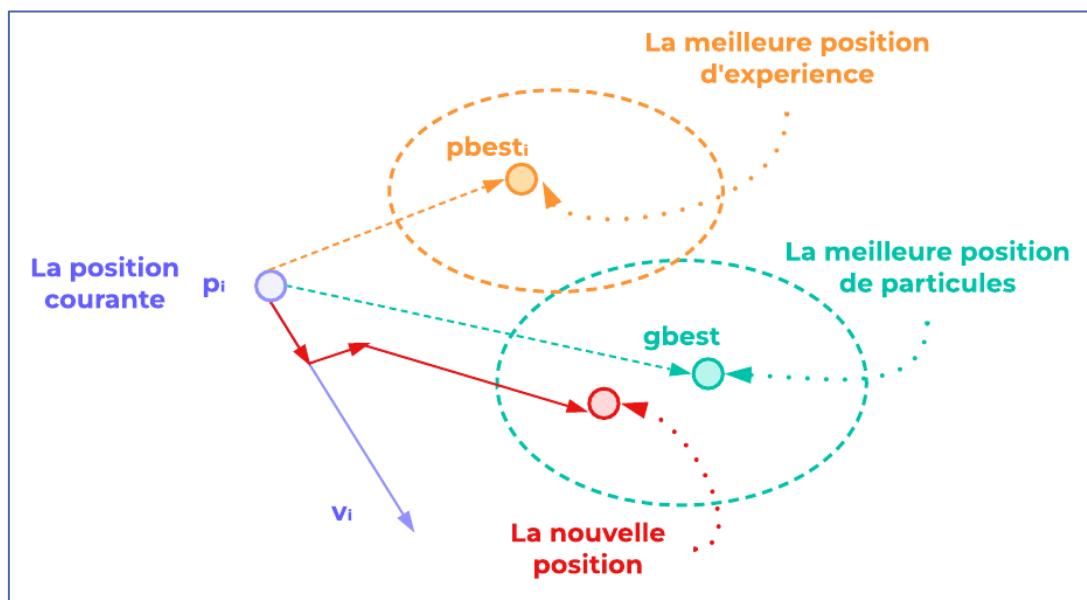


Figure 20: fonctionnement de l'algorithme PSO

PSO AVEC RÉSEAU DE NEURONES

L'utilisation de l'algorithme PSO pour l'optimisation d'un réseau de neurones, généralement appelée "optimisation de réseau de neurones par PSO", s'appuie sur le même principe fondamental que l'optimisation standard, mais avec une application spécifique aux paramètres du réseau de neurones. Voici comment fonctionne l'optimisation de réseau de neurones par PSO :

1. INITIALISATION DES PARTICULES:

- Chaque particule représente une configuration de paramètres pour le réseau de neurones, tels que les poids et les biais des connexions entre les neurones.

- Les paramètres de chaque particule sont initialisés de manière aléatoire ou avec des valeurs de départ préétablies.

2. ÉVALUATION DE LA PERFORMANCE DU RÉSEAU DE NEURONES:

- Pour chaque particule, le réseau de neurones correspondant est construit avec les paramètres associés.
- La performance du réseau de neurones est évaluée en utilisant une métrique appropriée, telle que l'exactitude pour la classification, l'erreur quadratique moyenne pour la régression, ou une autre métrique spécifique au problème.

3. MISE À JOUR DE LA MEILLEURE PERFORMANCE LOCALE (PBEST):

- Chaque particule conserve en mémoire la meilleure performance (la meilleure métrique de performance) qu'elle a obtenue jusqu'à présent avec sa configuration de paramètres.
- Si la performance actuelle est meilleure que la performance "pbest", la particule met à jour sa meilleure performance locale.

4. MISE À JOUR DE LA MEILLEURE PERFORMANCE GLOBALE (GBEST) :

- Les particules communiquent entre elles pour partager leurs informations sur les performances locales.
- La meilleure performance globale, appelée "gbest", est déterminée en fonction des performances locales de toutes les particules. Il s'agit de la meilleure performance parmi toutes les particules.

5. MISE À JOUR DES PARAMÈTRES DU RÉSEAU DE NEURONES:

- Chaque particule ajuste ses paramètres (les poids et les biais du réseau de neurones) en fonction de sa meilleure performance locale ("pbest") et de la meilleure performance globale ("gbest").
- Les règles de mise à jour des paramètres sont spécifiques au problème et à l'architecture du réseau de neurones, mais elles sont généralement basées sur des variations proportionnelles aux performances et peuvent utiliser des coefficients d'apprentissage.

6. CRITÈRE D'ARRÊT:

- L'algorithme PSO continue de mettre à jour les configurations de paramètres du réseau de neurones jusqu'à ce qu'un critère d'arrêt soit atteint. Ce critère peut être le nombre maximal d'itérations, l'atteinte d'une performance souhaitée, ou d'autres conditions définies par l'utilisateur.

7. RÉSULTATS:

- Une fois que l'algorithme PSO s'est arrêté, la meilleure configuration de paramètres du réseau de neurones, associée à la meilleure performance globale ("gbest"), est utilisée comme modèle optimisé pour résoudre la tâche spécifique.

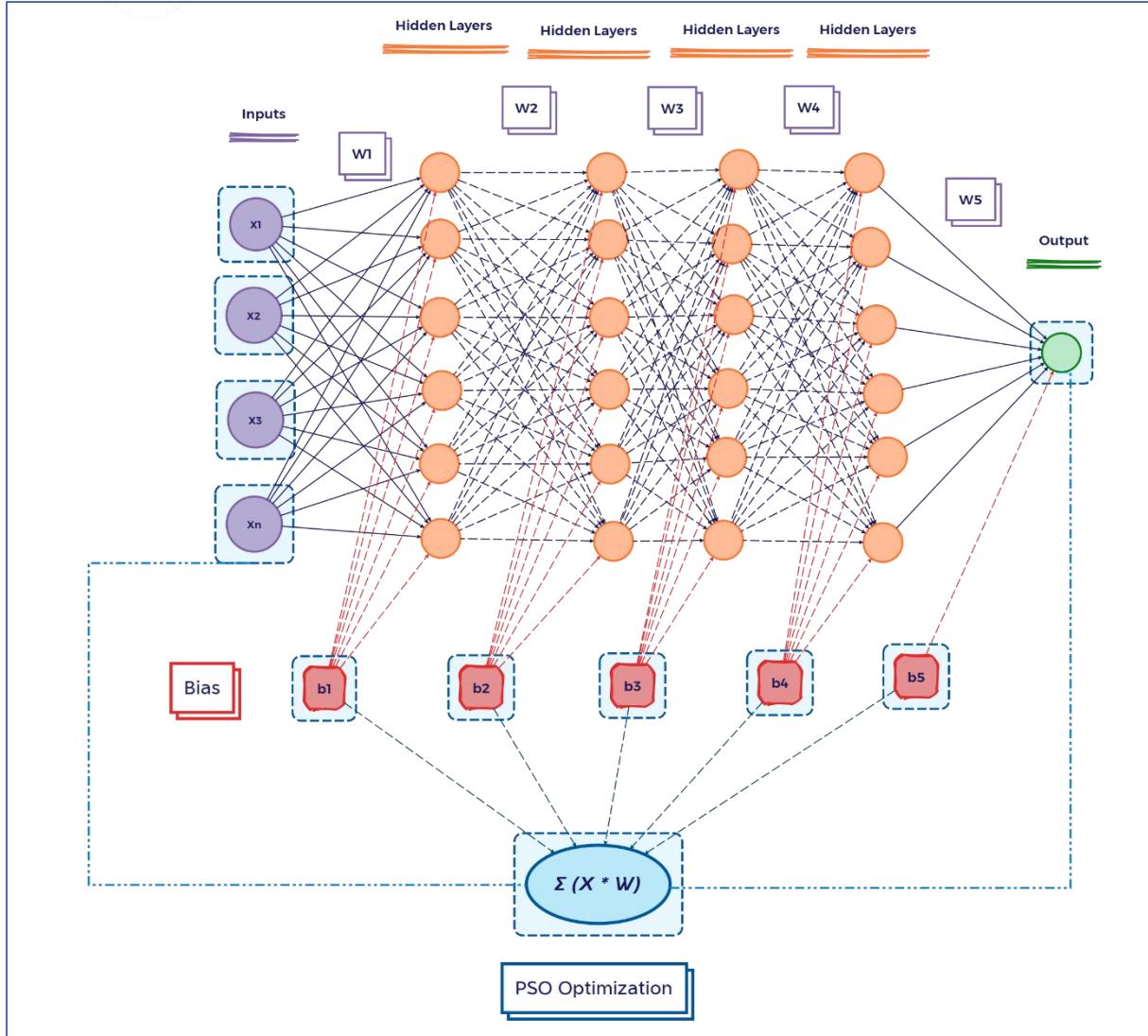


Figure 21: PSO avec Réseau de neurones

L'utilisation de PSO pour l'optimisation de réseaux de neurones permet de trouver des ensembles de paramètres qui maximisent ou minimisent la performance du réseau pour une tâche donnée, ce qui est essentiel dans l'entraînement efficace de réseaux de neurones profonds et complexes. Cette approche peut aider à accélérer le processus d'ajustement des paramètres, en particulier lorsque l'optimisation par recherche aléatoire ou par grille serait inefficace.

CUDA (COMPUTE UNIFIED DEVICE ARCHITECTURE)

CUDA est une technologie développée par NVIDIA qui révolutionne la manière dont les calculs intensifs sont effectués. Contrairement à l'exécution séquentielle traditionnelle sur les processeurs centraux (CPU), CUDA exploite la puissance de calcul massivement parallèle offerte par les cartes graphiques ou GPU (Graphics Processing Units).

À la base de CUDA se trouve le concept de programmation parallèle, où des milliers de coeurs de traitement sur un GPU peuvent effectuer simultanément des calculs sur des données. Cette architecture parallèle permet de réduire considérablement le temps nécessaire pour effectuer des calculs complexes, ce qui en fait un choix idéal pour des applications gourmandes en calcul, telles que l'apprentissage automatique, la simulation, la modélisation numérique et bien d'autres. Un aspect essentiel de CUDA est sa capacité à diviser les tâches en threads, petites unités de travail, qui peuvent être exécutées en parallèle. Les threads peuvent coopérer pour accomplir des tâches complexes, et les blocs de threads peuvent être organisés en grilles pour une gestion efficace des calculs. Cette flexibilité permet aux développeurs de personnaliser et d'optimiser les calculs pour répondre aux besoins spécifiques de leur projet.

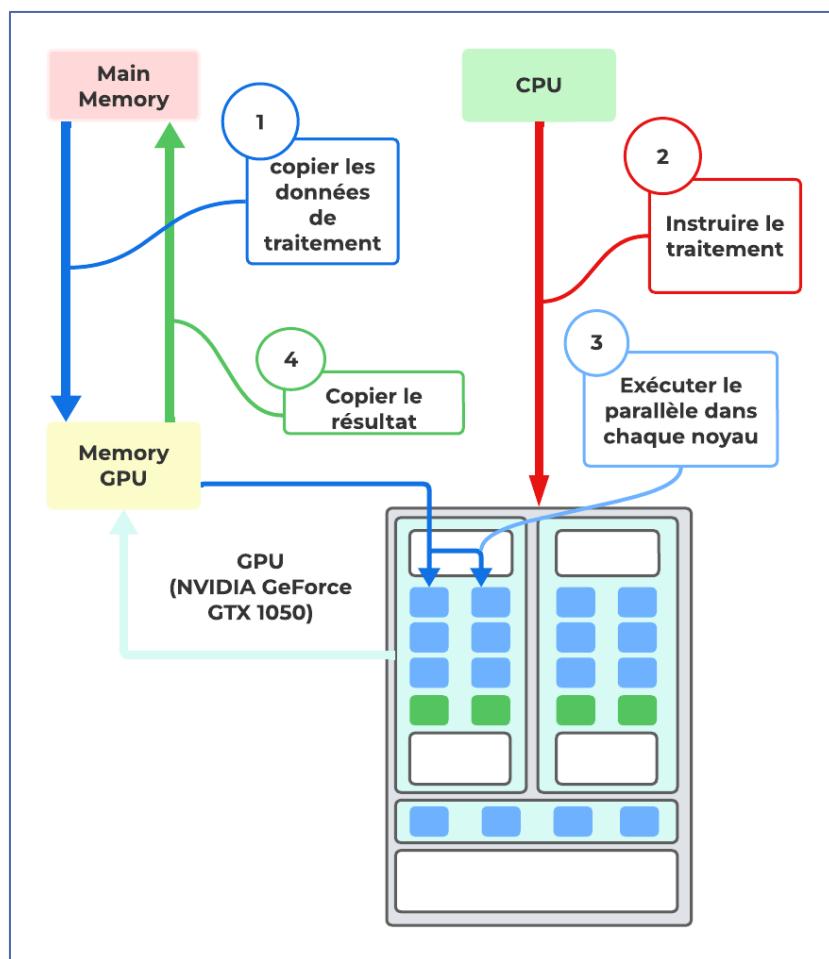


Figure 22: Flux de traitement dans CUDA

Outre l'accélération des calculs, CUDA offre également la possibilité de transférer des données entre la mémoire du CPU et du GPU de manière efficace, ce qui est crucial pour des applications qui manipulent de grandes quantités de données.

En résumé, CUDA est une technologie puissante qui exploite les capacités parallèles des GPU pour accélérer les calculs et améliorer les performances des applications scientifiques et de calcul intensif. Son adoption dans ce projet vise à exploiter ces avantages pour résoudre efficacement des problèmes complexes et gourmands en calcul.

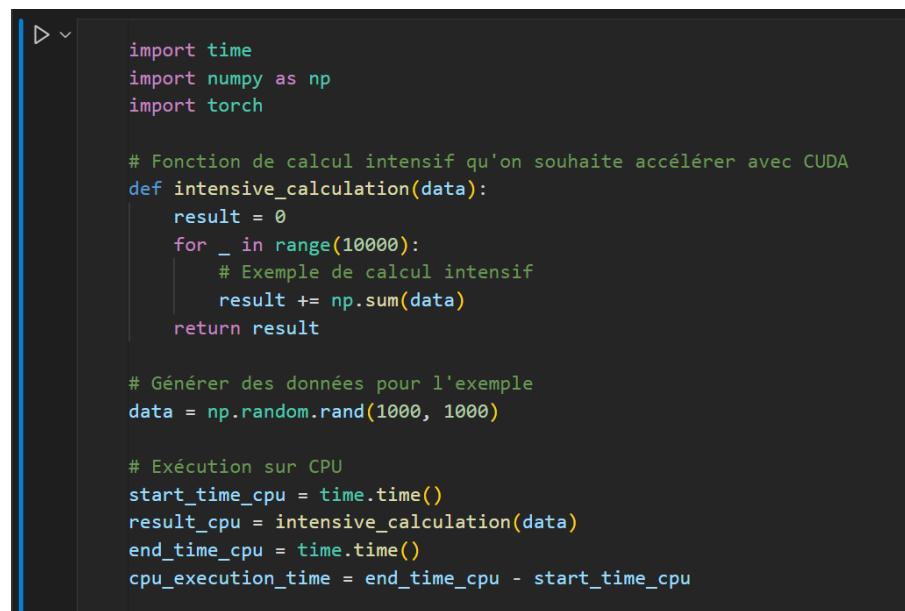
UTILISATION DE CUDA POUR L'ACCELERATION DES CALCULS

La décision de l'utilisation de CUDA découle de la nécessité d'optimiser les performances de calcul, en particulier pour les tâches intensives que notre application doit accomplir. CUDA nous offre la possibilité d'exploiter pleinement la puissance de calcul massivement parallèle des cartes graphiques (GPU), ce qui se traduit par des gains significatifs en termes de vitesse d'exécution.

DEMONSTRATION

La justification de l'utilisation de CUDA peut être renforcée en montrant des performances comparatives entre l'exécution sur CPU et l'exécution sur GPU. Dans ce code de test nous utilisons des mesures de temps d'exécution pour mettre en évidence les avantages de CUDA.

Ce code génère des données aléatoires et effectue une opération de calcul intensif sur ces données à la fois sur le CPU et le GPU. Il mesure ensuite le temps d'exécution pour les deux cas et compare les résultats pour s'assurer qu'ils sont identiques.



```
▶ ▾
import time
import numpy as np
import torch

# Fonction de calcul intensif qu'on souhaite accélérer avec CUDA
def intensive_calculation(data):
    result = 0
    for _ in range(10000):
        # Exemple de calcul intensif
        result += np.sum(data)
    return result

# Générer des données pour l'exemple
data = np.random.rand(1000, 1000)

# Exécution sur CPU
start_time_cpu = time.time()
result_cpu = intensive_calculation(data)
end_time_cpu = time.time()
cpu_execution_time = end_time_cpu - start_time_cpu
```

Figure 23: GPU vs CPU (part 1)

```

# Transférer les données sur GPU
data_gpu = torch.tensor(data, dtype=torch.float32)

# Exécution sur GPU
start_time_gpu = time.time()
# Transférer les résultats de nouveau sur CPU
result_gpu = intensive_calculation(data_gpu.cpu().numpy())
end_time_gpu = time.time()
gpu_execution_time = end_time_gpu - start_time_gpu

# Comparaison des temps d'exécution
print("Temps d'exécution sur CPU :", cpu_execution_time, "secondes")
print("Temps d'exécution sur GPU :", gpu_execution_time, "secondes")

# Vérification des résultats (les résultats doivent être identiques)
if np.allclose(result_cpu, result_gpu):
    print("Les résultats CPU et GPU sont identiques.")
else:
    print("Les résultats CPU et GPU sont différents.")

[1] ✓ 20.4s
...
Temps d'exécution sur CPU : 12.75890302658081 secondes
Temps d'exécution sur GPU : 4.759001731872559 secondes
Les résultats CPU et GPU sont identiques.

```

Figure 24: GPU vs CPU (part 2)

Nous avons réalisé une comparaison pratique pour illustrer l'impact de CUDA. En effectuant une opération de calcul intensif sur des données générées aléatoirement, nous avons constaté que le temps d'exécution sur CPU était de 12.7 SECONDES, tandis que sur GPU, il se réduisait à 4.7 SECONDES. Cette différence est remarquable et démontre l'efficacité de CUDA pour accélérer les calculs.

Grâce à ce code, on peut démontrer de manière empirique que l'utilisation de CUDA pour les calculs intensifs permet d'obtenir des performances significativement meilleures par rapport à l'exécution sur CPU, renforçant ainsi la justification de l'utilisation de CUDA.

VERIFICATION DE LA COMPATIBILITE MATERIELLE AVEC CUDA

Nous avons consacré le 2^{ème} chapitre à l'installation de CUDA dans notre système, dans cette partie nous allons vérifier la compatibilité de l'installation faite avec le système.

Nous avons utilisé la commande '*deviceQuery*' dans un terminal pour vérifier la compatibilité de notre carte graphique avec CUDA.

EXECUTION DE 'DEVICEQUERY'

```

C:\Users\anasz>deviceQuery
deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce GTX 1050"
  CUDA Driver Version / Runtime Version      12.2 / 12.2
  CUDA Capability Major/Minor version number: 6.1
  Total amount of global memory:             4096 MBytes (4294836224 bytes)
  ( 5 ) Multiprocessors, (128) CUDA Cores/MP: 640 CUDA Cores
  GPU Max Clock rate:                      1493 MHz (1.49 GHz)
  Memory Clock rate:                       3504 MHz
  Memory Bus Width:                        128-bit
  L2 Cache Size:                           524288 bytes
  Maximum Texture Dimension Size (x,y,z): 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers: 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers: 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:          zu bytes
  Total amount of shared memory per block:  zu bytes
  Total number of registers available per block: 65536
  Warp size:                             32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:       1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                   zu bytes
  Texture alignment:                     zu bytes
  Concurrent copy and kernel execution:   Yes with 5 copy engine(s)
  Run time limit on kernels:              Yes
  Integrated GPU sharing Host Memory:    No
  Support host page-locked memory mapping: Yes
  Alignment requirement for Surfaces:     Yes
  Device has ECC support:                Disabled
  CUDA Device Driver Mode (TCC or WDDM):  WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA): Yes
  Device supports Compute Preemption:    Yes
  Supports Cooperative Kernel Launch:    Yes
  Supports MultiDevice Co-op Kernel Launch: No
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

DeviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 12.2, CUDA Runtime Version = 12.2, NumDevs = 1, Device0 = NVIDIA GeForce GTX 1050
Result = PASS

```

L'outil `deviceQuery` a fourni des informations détaillées sur notre carte graphique, y compris le modèle, la capacité de calcul, le nombre de coeurs CUDA, etc. Nous avons vérifié que notre carte était correctement détectée et compatible avec la version de CUDA que nous avons installée.

RÉSEAU DE NEURONES + CUDA + PSO

Notre projet à pour but de prédire la dépense énergétiques dans les bâtiments à l'aide d'un model de réseau de neurones avec une optimisation d'hyperparamètres à l'aide de l'algorithme PSO tout en utilisant CUDA pour l'accélération matérielle.

CUDA est utilisé pour profiter de l'accélération matérielle offerte par les GPU (Graphics Processing Units) lors de l'entraînement et de l'évaluation d'un réseau de neurones. Le modèle de réseau de neurones et les données d'entraînement sont transférés sur le GPU. Ensuite, lors des phases cruciales d'entraînement et d'évaluation du modèle, les calculs, tels que la propagation avant, la rétropropagation du gradient et le calcul de la perte, sont effectués en parallèle sur les nombreux coeurs de calcul du GPU. Ce parallélisme massif accélère de manière significative les calculs par rapport à un calcul séquentiel sur un CPU, permettant ainsi une optimisation plus rapide et efficace du modèle de réseau de neurones.

UTILISATION DE CUDA AVEC LE RÉSEAU DE NEURONES

Dans ce projet nous avons opter pour l'incorporation de CUDA dans le réseau de neurones et non pas dans l'optimisation à l'aide de PSO pour plusieurs raisons :

1. **NATURE DES CALCULS** : Le réseau de neurones implique des calculs intensifs, tels que les multiplications de matrices et les opérations sur les tenseurs, qui bénéficient considérablement de l'accélération matérielle offerte par les GPU via CUDA. En revanche, PSO implique principalement des opérations de météouristique (mouvement des particules dans un espace de recherche) qui sont moins gourmandes en calcul par rapport aux opérations de rétropropagation et de mise à jour de poids dans un réseau de neurones.
2. **PARALLÉLISME INTRINSÈQUE** : Le réseau de neurones est intrinsèquement parallèle, car les calculs pour chaque exemple d'entraînement peuvent être effectués indépendamment les uns des autres. Cela signifie que les calculs peuvent être répartis sur de nombreux coeurs de GPU, ce qui accélère l'entraînement. En revanche, l'algorithme PSO n'a pas la même structure parallèle intrinsèque, car les particules se déplacent dans l'espace de recherche de manière coopérative, mais chaque itération dépend des positions des autres particules.
3. **COÛT DE TRANSFERT DES DONNÉES** : Transférer des données entre la mémoire centrale (CPU) et la mémoire du GPU (VRAM) a un coût. Dans le cas du réseau de neurones, les données d'entraînement sont généralement volumineuses, et les transférer vers le GPU une fois pour l'entraînement peut être plus efficace que de les transférer à chaque itération de PSO.
4. **FOCALISER SUR LA TÂCHE PRINCIPALE** : L'objectif principal de l'utilisation de CUDA dans ce contexte est d'accélérer l'entraînement du réseau de neurones, car il s'agit de la tâche la plus coûteuse en termes de calcul dans le processus. Le PSO, en revanche, est principalement utilisé pour l'optimisation des hyperparamètres, et l'accélération matérielle pour cette phase pourrait ne pas avoir autant d'impact sur les performances globales de l'algorithme.

CHAPITRE 4:MISE EN PRATIQUE DU PROJET

ORIGINE DES DONNÉES

DESCRIPTION DE LA SOURCE DE DONNÉES

Les données utilisées dans ce projet proviennent d'une collecte réalisée dans divers bâtiments afin de comprendre et de prédire la dépense énergétique. Chacune des observations de données représente des mesures prises à un moment donné dans un bâtiment spécifique. Voici une description des attributs de la source de données :

Date	La date à laquelle les mesures ont été prises
Id	L'identifiant unique associé à chaque observation
Total electricity consumption	La consommation totale d'électricité enregistrée dans le bâtiment
Air Temperature	La température de l'air à l'intérieur du bâtiment
Radiant Temperature	La température radiante, qui mesure la sensation thermique des occupants
Operative Temperature	La température opérative, qui est une combinaison de la température de l'air et de la température radiante
Outside Dry-Bulb Temperature	La température extérieure
Glazing	Informations sur le vitrage ou les fenêtres du bâtiment
Walls	Caractéristiques des murs du bâtiment
Ceilings (int)	Caractéristiques des plafonds intérieurs
Floors (int)	Caractéristiques des planchers intérieurs
Ground Floors	Caractéristiques des planchers au niveau du sol
Partitions (int)	Informations sur les cloisons intérieures
Roofs	Caractéristiques du toit du bâtiment
External Infiltration	Les infiltrations d'air extérieur dans le bâtiment
External Vent.	Les systèmes de ventilation extérieure
General Lighting	L'éclairage général dans le bâtiment
Computer + Equip	La consommation électrique des ordinateurs et de l'équipement
Occupancy	Le nombre d'occupants dans le bâtiment
Solar Gains Interior Windows	Les gains solaires à travers les fenêtres intérieures
Solar Gains Exterior Windows	Les gains solaires à travers les fenêtres extérieures
Zone Sensible Heating	Le chauffage sensible dans une zone spécifique
Zone Sensible Cooling	Le refroidissement sensible dans une zone spécifique
Sensible Cooling	Le refroidissement sensible global
Total Cooling	Le refroidissement total
Mech Vent + Nat Vent + Infiltration	La ventilation mécanique, naturelle et les infiltrations d'air

Figure 25: présentation des attributs de la source de données

SIGNIFICATION DES ATTRIBUTS

Chacun des attributs de nos données joue un rôle clé dans la compréhension de la performance énergétique des bâtiments.

1. **DATE** : L'attribut "Date" indique le moment auquel les mesures ont été enregistrées. Il permet de prendre en compte les variations saisonnières et temporelles qui peuvent influencer la consommation d'énergie.
2. **TOTAL ELECTRICITY CONSUMPTION** : Cet attribut représente la consommation totale d'électricité enregistrée dans le bâtiment. Il s'agit de notre variable cible, que nous cherchons à prédire.
3. **AIR TEMPERATURE** : La température de l'air à l'intérieur du bâtiment est un facteur clé pour évaluer le confort thermique des occupants et la demande de chauffage ou de refroidissement.
4. **RADIANT TEMPERATURE** : La température radiante mesure la sensation thermique des occupants, ce qui peut influencer leurs besoins en chauffage ou en refroidissement.
5. **OPERATIVE TEMPERATURE** : L'attribut "Operative Temperature" est une combinaison de la température de l'air et de la température radiante, donnant une indication globale du confort thermique.
6. **OUTSIDE DRY-BULB TEMPERATURE** : La température extérieure joue un rôle majeur dans la détermination de la charge thermique du bâtiment.
7. **GLAZING** : Cet attribut fournit des informations sur les caractéristiques des vitrages ou des fenêtres du bâtiment, ce qui peut influencer les pertes et les gains de chaleur.
8. **WALLS, CEILINGS (INT), FLOORS (INT), GROUND FLOORS, PARTITIONS (INT), ROOFS** : Ces attributs décrivent diverses caractéristiques de la construction du bâtiment, qui peuvent avoir un impact sur l'isolation thermique et l'efficacité énergétique.
9. **EXTERNAL INFILTRATION** : L'infiltration d'air extérieur dans le bâtiment peut entraîner des pertes de chaleur ou de froid, ce qui influence la demande de chauffage ou de refroidissement.
10. **EXTERNAL VENTILATION** : La ventilation extérieure peut également avoir un impact sur la consommation d'énergie en influençant la qualité de l'air intérieur.
11. **GENERAL LIGHTING** : Cet attribut représente la consommation électrique liée à l'éclairage général du bâtiment.

12. COMPUTER + EQUIP : Il indique la consommation électrique des ordinateurs et de l'équipement électrique.
13. OCCUPANCY : Le nombre d'occupants dans le bâtiment peut affecter la demande de chauffage, de refroidissement et d'éclairage.
14. SOLAR GAINS INTERIOR WINDOWS, SOLAR GAINS EXTERIOR WINDOWS : Ces attributs mesurent les gains de chaleur solaire à travers les fenêtres, ce qui peut influencer la température intérieure.
15. ZONE SENSIBLE HEATING, ZONE SENSIBLE COOLING, SENSIBLE COOLING, TOTAL COOLING : Ces attributs sont liés à la charge thermique du bâtiment, qui dépend de nombreux facteurs, dont la température, la ventilation et les gains solaires.
16. MECH VENT + NAT VENT + INFILTRATION : Cet attribut combine les différentes composantes de la ventilation, y compris la ventilation mécanique, naturelle et les infiltrations d'air.

OBJECTIVE DE PRÉDICTION

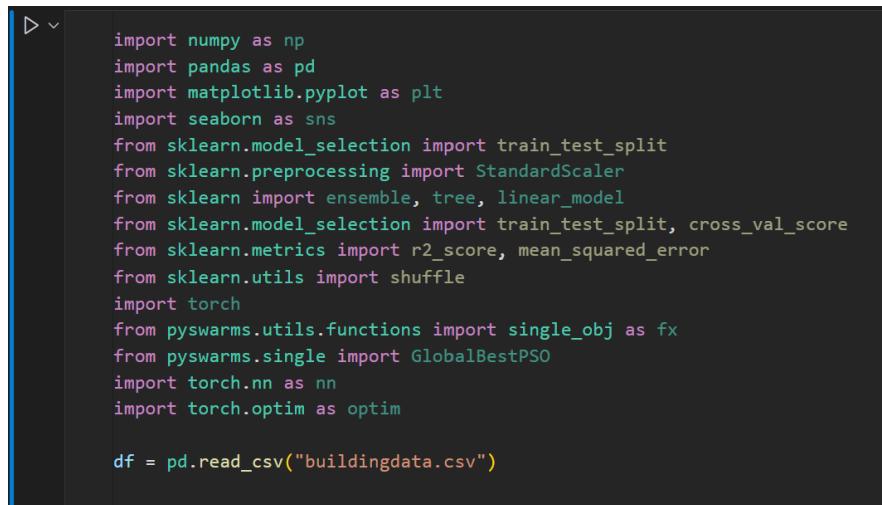
L'objectif principal de ce projet est de développer un modèle de prédiction de la dépense énergétique dans les bâtiments en utilisant les données disponibles. Plus précisément, nous cherchons à atteindre les objectifs suivants :

1. *Prédiction de la Consommation d'Électricité* : Notre variable cible principale est la "Total Electricity Consumption," qui représente la consommation totale d'électricité dans les bâtiments. Nous souhaitons développer un modèle capable de prédire cette consommation en fonction des divers attributs environnementaux, de construction et de comportement des occupants.
2. *Compréhension des Facteurs d'Influence* : Nous cherchons à identifier les facteurs qui ont le plus d'influence sur la consommation d'électricité dans les bâtiments. Cela nous permettra de mieux comprendre les déterminants de la dépense énergétique et d'orienter les décisions d'efficacité énergétique.
3. *Optimisation de la Gestion Énergétique* : En développant un modèle de prédiction précis, nous visons à fournir des informations utiles pour l'optimisation de la gestion énergétique des bâtiments. Cela peut inclure des recommandations pour réduire la consommation d'énergie, améliorer le confort thermique et réduire l'empreinte carbone.

4. *Prise de Décisions Informatisées* : À terme, nous espérons intégrer le modèle de prédiction dans un système d'aide à la décision pour les gestionnaires de bâtiments. Cela leur permettra de prendre des décisions éclairées en temps réel pour optimiser l'efficacité énergétique.

PREPROCESSING

La 1^{er} étape du code est d'importer tous les packages nécessaires ainsi que le fichier .csv de la donnée source et de récupérer des informations sur les types celle-ci.



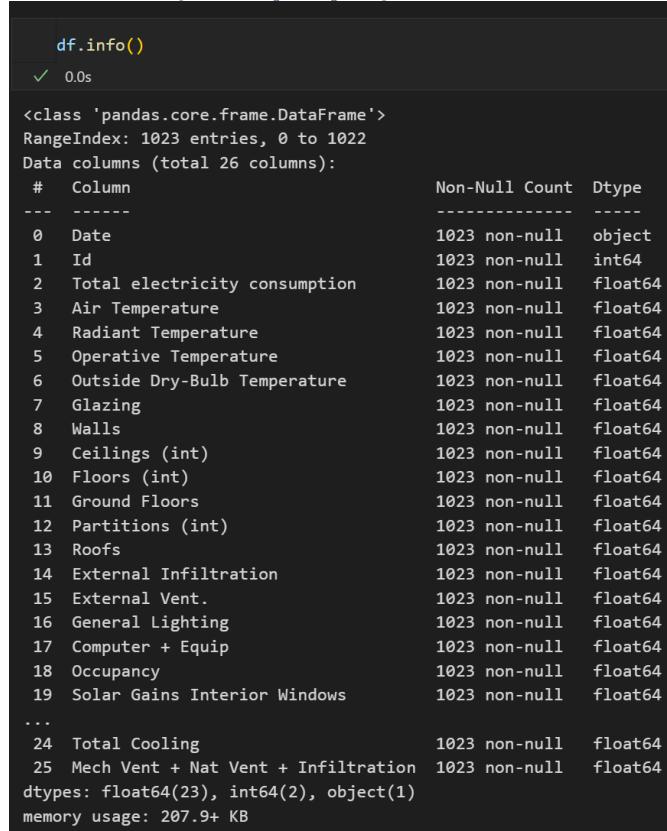
```

> v
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import ensemble, tree, linear_model
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.utils import shuffle
import torch
from pyswarms.utils.functions import single_obj as fx
from pyswarms.single import GlobalBestPSO
import torch.nn as nn
import torch.optim as optim

df = pd.read_csv("buildingdata.csv")

```

Figure 26: importer packages et data source



df.info()			
	#	Column	Non-Null Count Dtype
✓	0.0s		
	<class 'pandas.core.frame.DataFrame'>		
	RangeIndex: 1023 entries, 0 to 1022		
	Data columns (total 26 columns):		
	#	Column	Non-Null Count Dtype
	0	Date	1023 non-null object
	1	Id	1023 non-null int64
	2	Total electricity consumption	1023 non-null float64
	3	Air Temperature	1023 non-null float64
	4	Radiant Temperature	1023 non-null float64
	5	Operative Temperature	1023 non-null float64
	6	Outside Dry-Bulb Temperature	1023 non-null float64
	7	Glazing	1023 non-null float64
	8	Walls	1023 non-null float64
	9	Ceilings (int)	1023 non-null float64
	10	Floors (int)	1023 non-null float64
	11	Ground Floors	1023 non-null float64
	12	Partitions (int)	1023 non-null float64
	13	Roofs	1023 non-null float64
	14	External Infiltration	1023 non-null float64
	15	External Vent.	1023 non-null float64
	16	General Lighting	1023 non-null float64
	17	Computer + Equip	1023 non-null float64
	18	Occupancy	1023 non-null float64
	19	Solar Gains Interior Windows	1023 non-null float64
	...		
	24	Total Cooling	1023 non-null float64
	25	Mech Vent + Nat Vent + Infiltration	1023 non-null float64
		dtypes:	float64(23), int64(2), object(1)
		memory usage:	207.9+ KB

Figure 27: information sur les colonnes de la data source

À partir de l'output de la ligne « `df.info()` » on constate que mise à part la colonne Date toutes les features sont de type numérique, d'ici on peut conclure qu'on n'a pas de données catégoriels qui va nécessiter d'être supprimer ou séparer lors du preprocessing.

GESTION DES VALEURS MANQUANTES

L'étape de preprocessing la plus cruciale est de manipuler les valeurs manquantes, il sera difficile voir impossible de procéder avec une analyse efficace avec des lignes dont des valeurs sont manquantes, alors pour remédier à cela nous cherchons les valeurs manquantes et on l'est remplace par la moyenne de la colonne.

```
# Check for missing values
missing_values = df.isnull().sum()

# Print columns with missing values
print("Columns with missing values:")
print(missing_values[missing_values > 0])

# Fill missing values with mean (for numerical columns)
df.fillna(df.mean(), inplace=True)

# After filling missing values, check again for any remaining missing values
remaining_missing = df.isnull().sum().sum()
print("Remaining missing values:", remaining_missing)

# Save the cleaned dataset
df.to_csv("cleaned_data.csv", index=False)
✓ 0.1s

Columns with missing values:
Series([], dtype: int64)
Remaining missing values: 0
```

Figure 28: Handling missing values

CORRELATION

Après une analyse initiale de notre données source, on peut constater que les colonnes peuvent être répartie en plusieurs groupes.

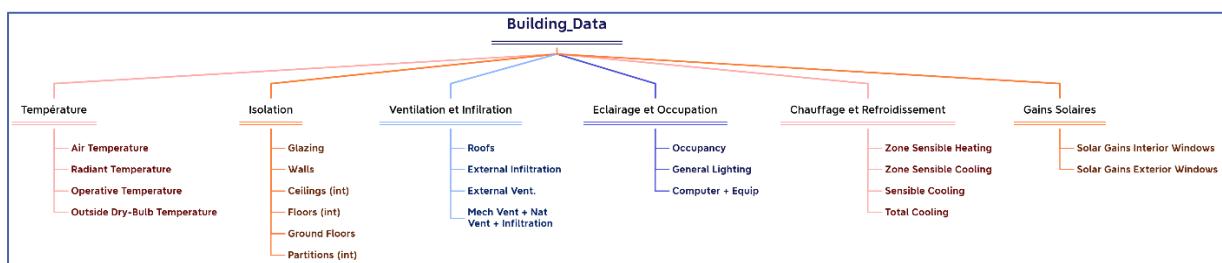


Figure 29: répartition des colonnes de la donnée source

Ainsi, et afin de mieux capturer l'impact de chacune de ces features sur la cible « Total Energy Consumption » nous avons suivis la répartition dans le schéma ci-dessous afin de générer plusieurs matrice de corrélation sous forme de heatmaps.

GROUPE 1 : TEMPÉRATURE

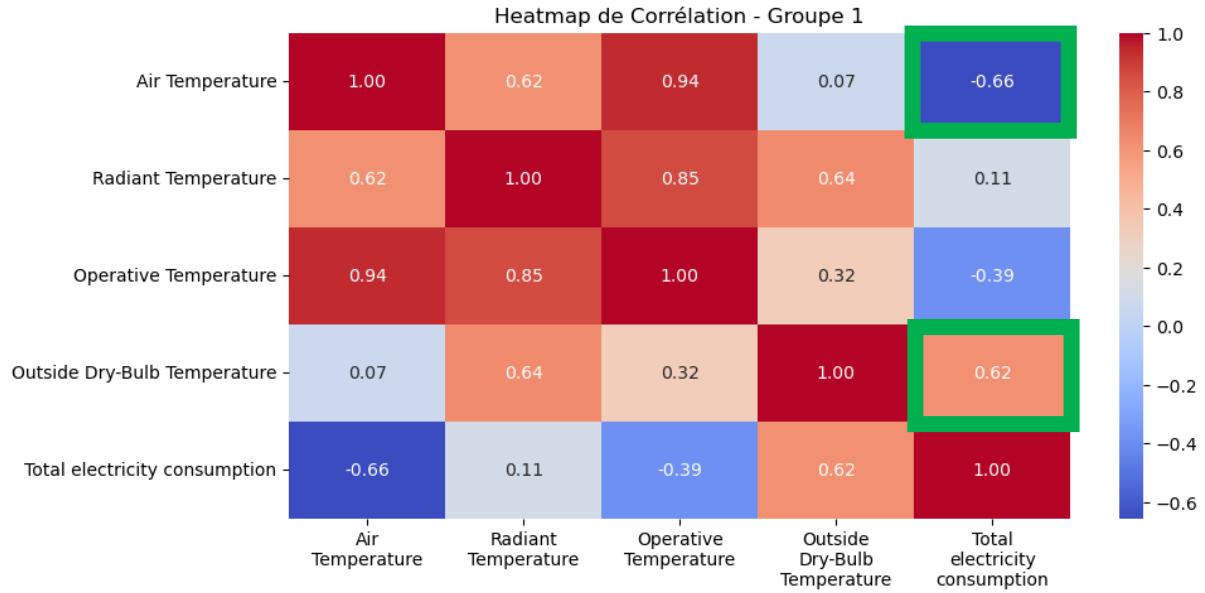


Figure 30: matrice de corrélation du 1er groupe

Dans ce groupe de colonnes on constate que les features qui peuvent avoir le plus d'impact sur la cible sont : « *Air Temperature* » et « *Outside Dry-Bulb Temperature* ». On peut aussi voir que la correlation entre « *Air Temperature* » et la cible est négative ce qui implique que si la température de l'air du bâtiment augmente la consommation d'énergie dans le bâtiment diminue et vis-vers-ça.

GROUPE 2 : ISOLATION

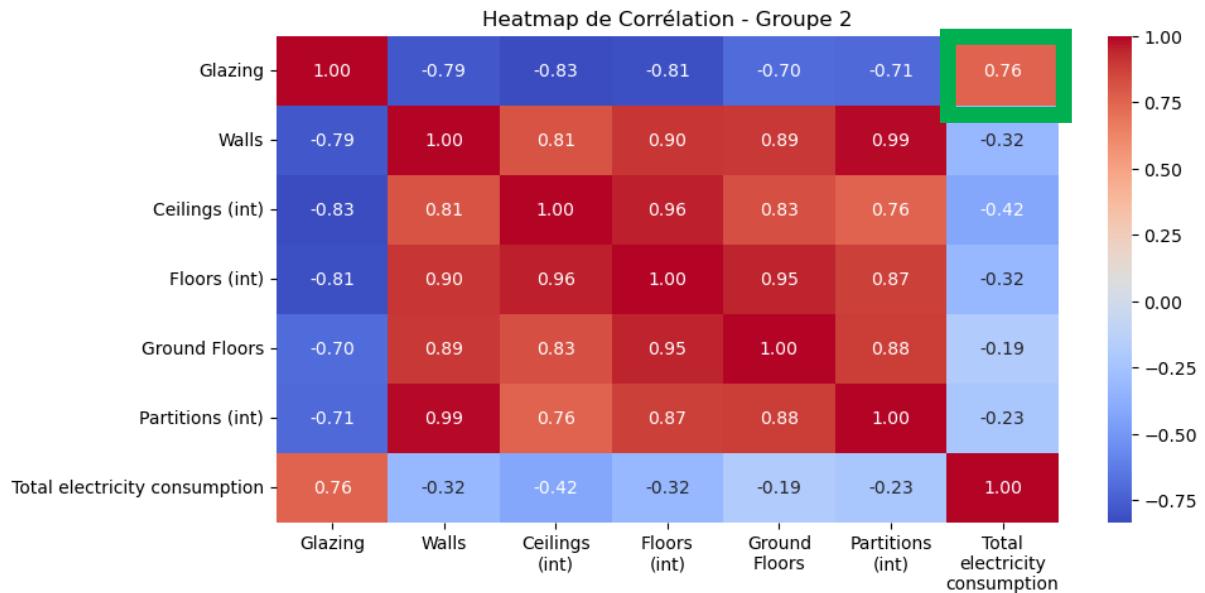


Figure 31: Matrice de corrélation du 2ème groupe

Dans le groupe de colonnes liée à l'isolation on remarque que la seule colonne qui peut avoir un impact significatif sur la cible est « *Glazing* », alors la qualité de vitrage dans ce cas influence le taux de consommation de l'énergie dans le bâtiment.

GROUPE 3 : VENTILATION ET INFILTRATION

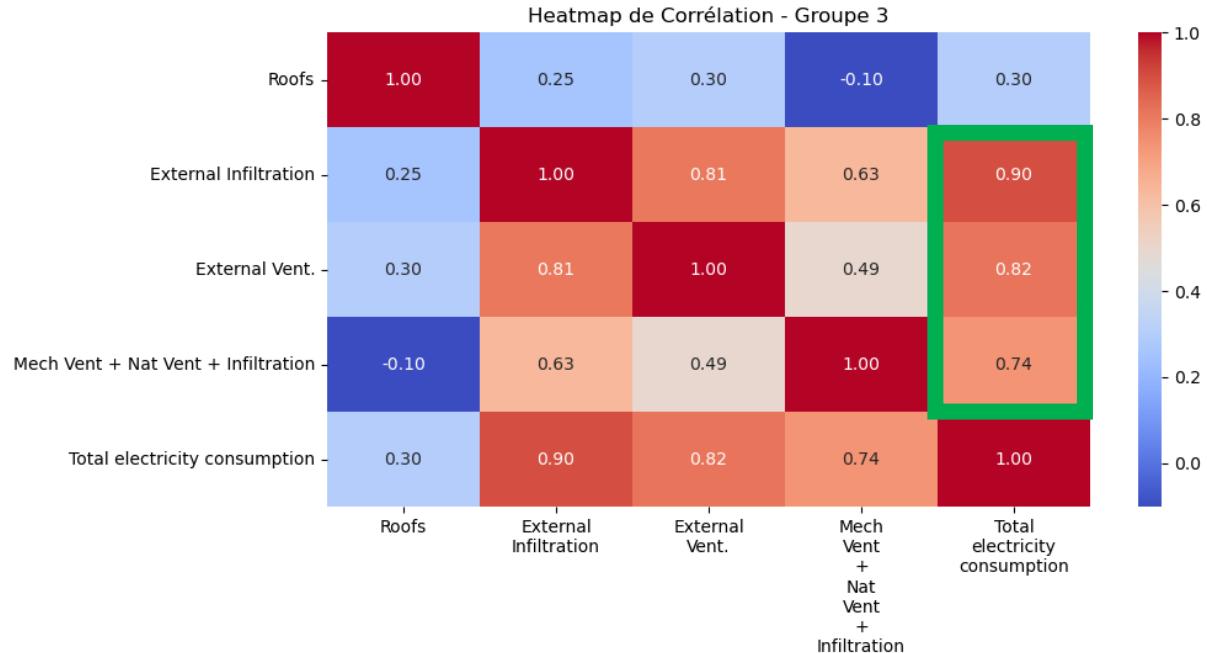


Figure 32: Matrice de corrélation du 3ème groupe

Dans le groupe de ventilation et infiltration on constate que la colonne « *External Infiltration* » a un très grand impact sur la cible. Sans oublier aussi les colonnes « *External Vent.* » et « *Mech Vent + Nat Vent + Infiltration* » qui ont aussi une assez grande corrélation avec la cible.

GROUPE 4 : ÉCLAIRAGE ET OCCUPATION

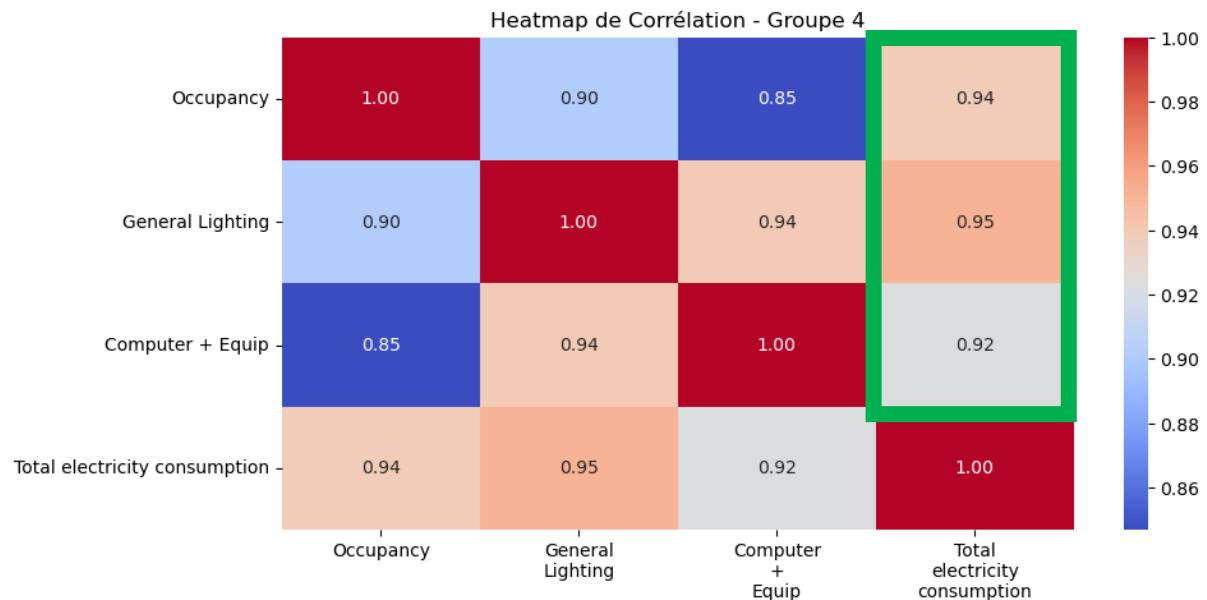


Figure 33: Matrice de corrélation du 4ème groupe

À travers cette matrice de corrélation on peut constater que toutes les colonnes de ce groupe soit: « *Occupancy* », « *General Lighting* » et « *Computer + Equip* » ont une très grande corrélation avec la cible.

GROUPE 5 : CHAUFFAGE ET REFROIDISSEMENT

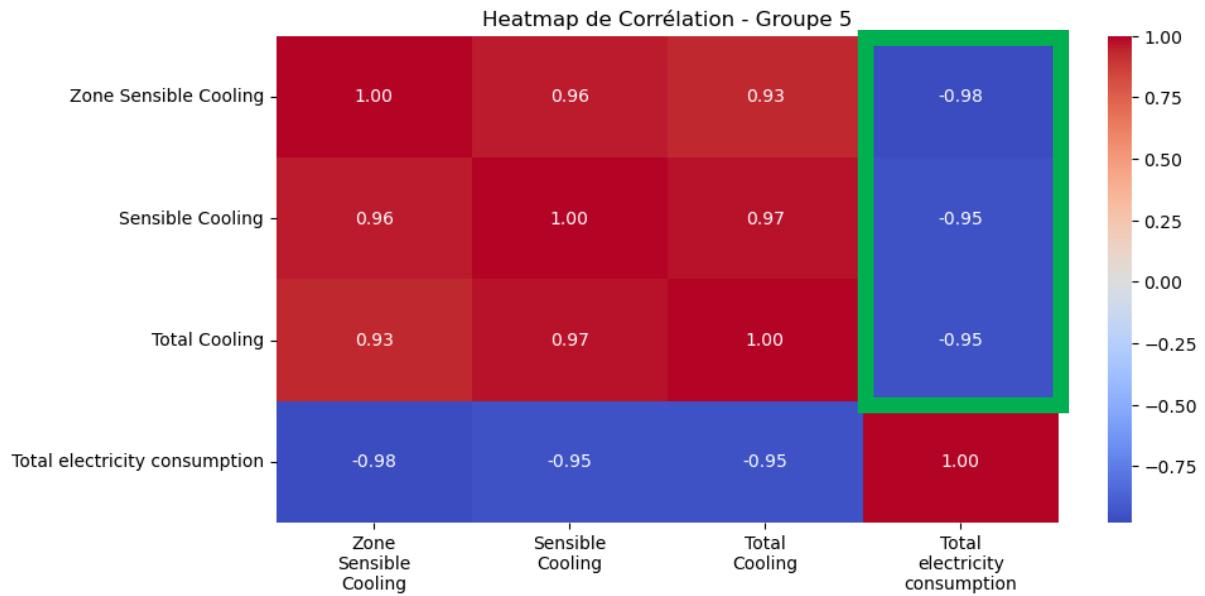


Figure 34: Matrice de corrélation du 5ème groupe

Les colonnes de ce groupe qui sont : « *Zone Sensible Cooling* », « *Sensible Cooling* » et « *Total Cooling* » ont une très forte corrélation négative avec la cible, ce qui implique que plus le taux de ces features augmente le taux de la consommation de l'énergie dans le bâtiment diminue.

GROUPE 6 : GAINS SOLAIRES

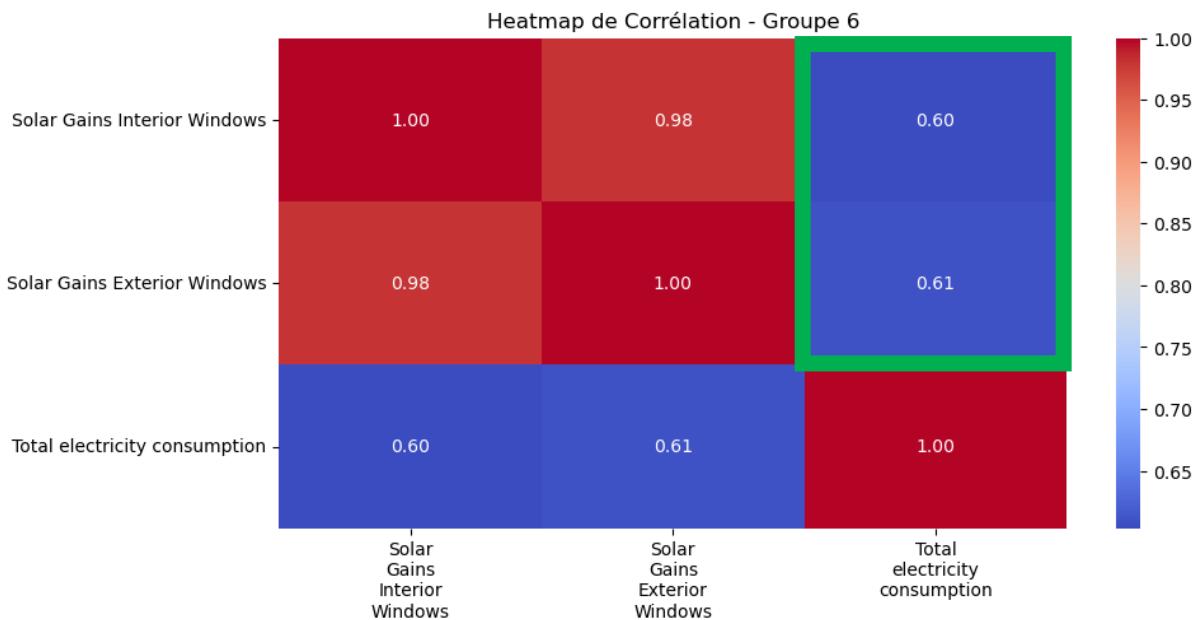


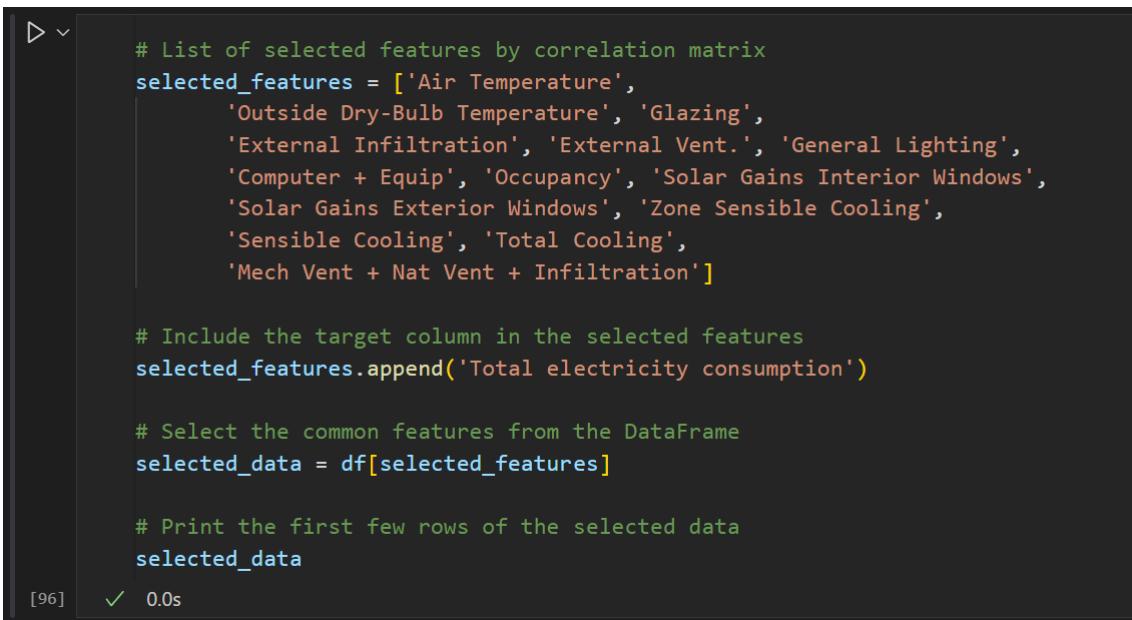
Figure 35: Matrice de corrélation du 6ème groupe

On peut constater que les colonnes du groupe gains solaires qui sont : « *Solar Gains interior Windows* » et « *Solar Gains Exterior Windows* » peuvent avoir un impact significatif sur la cible.

FEATURE SELECTION

Notre dataset contient 26 colonnes, c'est un grand nombre de features par rapport à 1023 lignes, afin d'améliorer au maximum les performances du modèle nous allons appliquer une « feature selection ». L'application d'une sélection de features lors du prétraitement des données est une étape importante dans le processus de préparation des données. Cette étape consiste à choisir un sous-ensemble des caractéristiques disponibles dans la donnée, en éliminant celles qui sont moins importantes ou redondantes.

En se basant sur les matrices de corrélation que nous avons réaliser plus tôt, on peut faire une sélection de caractéristiques selon leurs corrélation avec la cible.



```
# List of selected features by correlation matrix
selected_features = ['Air Temperature',
    'Outside Dry-Bulb Temperature', 'Glazing',
    'External Infiltration', 'External Vent.', 'General Lighting',
    'Computer + Equip', 'Occupancy', 'Solar Gains Interior Windows',
    'Solar Gains Exterior Windows', 'Zone Sensible Cooling',
    'Sensible Cooling', 'Total Cooling',
    'Mech Vent + Nat Vent + Infiltration']

# Include the target column in the selected features
selected_features.append('Total electricity consumption')

# Select the common features from the DataFrame
selected_data = df[selected_features]

# Print the first few rows of the selected data
selected_data
```

[96] ✓ 0.0s

Figure 36: sélection de caractéristiques basée sur la corrélation

Ainsi nous gardant uniquement les colonnes qui ont une corrélation significatif avec la cible et qui donc peuvent influencer la valeur de celle-ci.

DATA SPLITTING ET NORMALISATION

Avant de passer à la création du modèle d'apprentissage automatique, il est nécessaire de diviser les données. Dans notre cas on divise la donnée en 3 ensembles : un ensemble d'entraînement, un ensemble de validation et un ensemble de test.

Une étape qui ne doit pas être négliger est la normalisation, car elle permet de rendre les données plus adaptées à des algorithmes d'apprentissage automatique et plus précisément ceux qui sont sensibles à l'échelle des caractéristiques. Pour notre problème, il est préférable d'utiliser le « StandardScaler » pour la normalisation, car il est particulièrement utile avec des modèles basés sur la descente de gradient comme les réseaux de neurones.

« StandardScaler » est un outil de prétraitement couramment utilisé en apprentissage automatique pour normaliser ou mettre à l'échelle les données. Il fait partie du module `sklearn.preprocessing` de la bibliothèque scikit-learn (ou `sklearn`). StandardScaler fonctionne de la manière suivante :

1. *Calcul des statistiques descriptives* : Lorsque vous utilisez `StandardScaler`, il commence par calculer la moyenne (la valeur moyenne) et l'écart type (la mesure de la dispersion) pour chaque caractéristique (variable) dans l'ensemble de données d'entraînement.
 2. *Mise à l'échelle des données* : Ensuite, il soustrait la moyenne de chaque valeur de caractéristique et divise le résultat par l'écart-type. Cette opération de soustraction de la moyenne et de division par l'écart-type standardise les données, les centrant autour de zéro (moyenne nulle) et les mettant à l'échelle de manière à avoir une variance de 1.
 3. *Transformation des données* : Une fois que les statistiques ont été calculées sur l'ensemble de données d'entraînement, vous pouvez utiliser les mêmes paramètres pour mettre à l'échelle les ensembles de données de validation et de test.
-

CRÉATION DU MODÈLE DE RÉSEAU DE NEURONES

Pour créer le réseau de neurones, ont défini une classe ‘NEURALNETWORK’ qui représente un réseau de neurones artificiels à plusieurs couches pour la régression en utilisant PyTorch, qui est un package couramment utilisé pour créer et former des réseaux de neurones.

INITIALISATION DU RÉSEAU DE NEUONES

```
import torch.nn as nn
import torch.optim as optim

class NeuralNetwork(nn.Module):
    def __init__(self, num_hidden_layers, num_neurons_per_layer, learning_rate, l2_weight):
        super(NeuralNetwork, self).__init__()
        self.layers = nn.ModuleList()
        input_size = len(selected_features) - 1
```

Figure 37: La classe 'NeuralNetwork'

La classe ‘NeuralNetwork’ est définie en tant que sous-classe de ‘nn-Module’.

Dans la méthode ‘`__init__`’, le constructeur de la classe, plusieurs paramètres sont passés pour personnaliser le réseau de neurones. Ces paramètres sont :

- *num_hidden_layers* : Le nombre de couches cachées dans le réseau.
- *num_neurons_per_layer* : Le nombre de neurones dans chaque couche cachée.
- *learning_rate* : Le taux d'apprentissage utilisé pour l'optimisation des poids du réseau.
- *l2_weight* : Le poids de régularisation L2 (pénalisation L2) à appliquer lors de l'optimisation.

CRÉATION DES COUCHES DU RÉSEAU DE NEURONES

```
# Input layer
self.layers.append(nn.Linear(input_size, num_neurons_per_layer))
self.layers.append(nn.LeakyReLU(negative_slope=0.01))

# Hidden layers
for _ in range(num_hidden_layers - 1):
    self.layers.append(nn.Linear(num_neurons_per_layer, num_neurons_per_layer))
    self.layers.append(nn.LeakyReLU(negative_slope=0.01))

# Output layer
self.layers.append(nn.Linear(num_neurons_per_layer, 1))
```

Figure 38: Création des couches de réseau de neurones

- On commence par créer une liste de couches neuronales vide appelée ‘*self.layers*’.
- La taille de l’entrée du réseau (‘input-size’) est calculée en soustrayant 1 de la longueur des ‘*selected_features*’.
- Ensuite, les couches du réseau sont construites et ajoutées à la liste ‘*self.layers*’. Le réseau commence par une couche d’entrée, puis il y a plusieurs couches cachées (dont le nombre est déterminé par ‘*num_hidden_layers*’), et enfin une couche de sortie avec une seule sortie (pour la régression).

INITIALISATION DE L’OPTIMISATEUR ET DE LA FONCTION DE PERTE

```
self.optimizer = optim.Adam(self.parameters(), lr=learning_rate)
self.criterion = nn.MSELoss()

self.l2_weight = l2_weight
```

Figure 39: Initialisation de l’optimisateur et de la fonction perte

- L’optimisateur Adam est créé avec les paramètres du réseau en tant qu’arguments.
- La fonction perte utilisée pour la régression est l’erreur quadratique moyenne, initialisée avec ‘*nn.MSELoss()*’.

L’optimisateur Adam (*adaptive moment estimation*) est un algorithme d’optimisation populaire et efficace utilisé pour entraîner des réseaux de neurones. Il est particulièrement bien adapté à l’entraînement de réseaux de neurones profonds pour plusieurs raisons :

1. *Adaptabilité du taux d’apprentissage* : Adam ajuste automatiquement le taux d’apprentissage pour chaque paramètre du réseau en fonction de l’historique des gradients.
2. *Contrôle du moment* : Adam intègre un terme de moment (comme dans la SGD avec *moment*) qui aide à accélérer la convergence vers le minimum global en réduisant les oscillations indésirables.
3. *Contrôle de la normalisation des gradients* : Adam utilise une estimation de la racine carrée de la moyenne des carrés des gradients pour normaliser les mises à jour de poids, ce qui évite les problèmes de convergence associés à des gradients de grande magnitude.
4. *Biais corrigé* : Adam effectue une correction de biais pour compenser la perte d’information due à l’initialisation des moments à zéro. Cela améliore la stabilité de l’optimisation au début de l’entraînement.

L’utilisation d’Adam est courante dans de nombreux cadres d’apprentissage profond, car il permet d’entraîner efficacement des réseaux de neurones profonds tout en minimisant les besoins en réglage manuel des hyperparamètres.

STOCKAGE DU POIDS DE RÉGULARISATION L2 :

```
self.l2_weight = l2_weight
```

Figure 40: Stockage du poids de régularisation L2

- Le poids de régularisation L2 ('l2_weight') est stocké dans 'self.l2_weight' et peut être utilisé plus tard dans l'optimisation pour ajouter de la régularisation L2.

La régularisation L2, également appelée régularisation de poids L2 ou pénalisation L2, est une technique couramment utilisée en apprentissage automatique, en particulier dans le contexte de l'apprentissage profond, pour éviter le surapprentissage (overfitting) et améliorer la généralisation d'un modèle. Elle est particulièrement utile lorsque vous entraînez des modèles avec de nombreux paramètres, comme les réseaux de neurones. La régularisation L2 fonctionne en ajoutant un terme de pénalité à la fonction de perte (ou fonction d'objectif) du modèle, qui dépend des poids du modèle. La forme de cette pénalité est généralement proportionnelle à la somme des carrés des valeurs des poids (norme L2) du modèle. Le terme de pénalité L2 est calculé comme suit :

$$L2 = \lambda + \sum_{i=1}^N w_i^2$$

Où :

- *L2 : également notée 'Term L2' est le terme de pénalité L2 ajouté à la fonction de perte.*
- *λ : un hyperparamètre positif appelé « coefficient de régularisation » qui contrôle l'intensité de la régularisation. Plus λ est grand, plus la régularisation est forte.*
- *w_i : les poids du modèle*

En pratique, l'optimisation de la fonction perte lors de l'apprentissage d'un modèle revient à minimiser à la fois la perte d'entraînement (qui mesure la qualité de l'ajustement aux données d'entraînement) et le terme de pénalité L2. La régularisation L2 encourage les poids du modèle à rester petits, ce qui a pour effet de prévenir le surapprentissage en réduisant la complexité du modèle. Elle permet également de favoriser des solutions plus robustes en évitant que certains

IMPRESSION DES INFORMATIONS SUR LES COUCHES

```
# Print layer information for linear layers only
for i, layer in enumerate(self.layers):
    if isinstance(layer, nn.Linear):
        print(f"Layer {i}: Input Size = {layer.in_features}, Output Size = {layer.out_features}")
```

Figure 41: Impression des informations sur les couches

Enfin, une boucle parcourt les couches du réseau et affiche des informations sur les couches linéaires uniquement. Cela inclut la taille de l'entrée et de la sortie de chaque couche.

MÉTHODE « FORWARD »

La méthode « forward » est une méthode requise dans les classes PyTorch qui héritent de ‘nn.Module’. Elle définit la manière dont les données sont propagées à travers le réseau de neurones lors de l’inférence ou de l’entraînement.

```
def forward(self, x):
    for layer in self.layers:
        x = layer(x)
        #print("Layer output shape:", x.shape) # Add this line to print the tensor shape
    return x
```

Figure 42: Méthode "forward"

- Dans cette méthode, on itère sur les couches stockées dans ‘self.layers’. Pour chaque couche, on applique cette couche aux données d’entrée ‘x’. Cela signifie qu’on propage les données à travers chaque couche réseau.
- La variable ‘x’ est mise à jour à chaque itération pour contenir la sortie de la couche précédente. Ainsi, elle contient finalement la sortie de la dernière couche (couche de sortie) lorsque la boucle se termine.
- Enfin, la méthode retourne la sortie de la dernière couche, qui est la sortie du réseau de neurones après avoir propagé les données d’entrée.

MÉTHODE ‘COMPUTE_REGULARIZATION_LOSS

```
def compute_regularization_loss(self):
    l2_loss = 0.0

    for param in self.parameters():
        l2_loss += torch.sum(param ** 2)

    return 0.5 * self.l2_weight * l2_loss
```

Figure 43: Méthode "compute_regularization_loss"

- Cette méthode calcule la perte de régularisation L2 pour les poids du réseau de neurones. La régularisation L2 est utilisée pour éviter le surapprentissage en décourageant les poids de devenir trop grands.
- La méthode parcourt tous les paramètres du réseau de neurones à l’aide de ‘self.parameters()’. Ces paramètres incluent tous les poids du modèle.
- Pour chaque paramètre, on calcule le carré de sa valeur avec ‘param ** 2’ et on ajoute ces carrés pour tous les paramètres du modèle à ‘l2_loss’.
- Finalement, la perte de régularisation L2 est calculée en multipliant ‘l2_loss’ par ‘0.5’ (ou ‘1/2’) et en multipliant le résultat par le coefficient de régularisation L2 ‘self.l2_weight’.

CRÉATION DE LA FONCTION OBJECTIVE

La fonction ‘objective_function’ sert à évaluer la performance d'un modèle de réseau de neurones pour différentes combinaisons d'hyperparamètres. Elle est utilisée dans le contexte de l'optimisation des hyperparamètres pour sélectionner les meilleures valeurs d'hyperparamètres pour le modèle.

PRÉPARATION DES DONNÉES ET DE L'INITIALISATION

```
y_train_array = np.array(y_train).reshape(-1, 1)
y_val_array = np.array(y_val).reshape(-1, 1)
```

Figure 44: préparation et initialisation des données

- Les valeurs cibles ‘y_train’ et ‘y_val’ sont converties en tableaux NumPy et redimensionnées en vecteurs colonnes à l'aide de ‘reshape(-1, 1)’. Cela suggère que ces valeurs cibles sont transformées en un format attendu pour l'entraînement et la validation.

DÉFINITION DE LA FONCTION OBJECTIF

```
def objective_function(hyperparameters):
    num_particles, num_dimensions = hyperparameters.shape
    # Initialize an array to store validation losses for each particle
    validation_losses = np.zeros(num_particles)
```

Figure 45: définition de la fonction objective

- Cette fonction prend comme argument ‘hyperparameters’, une matrice qui contient différentes combinaisons d'hyperparamètres à évaluer. Chaque ligne de cette matrice représente une configuration d'hyperparamètres à tester.
- La fonction parcourut les différentes combinaisons d'hyperparamètres pour évaluer les performances du modèle sur les données de validation.

BOUCLE SUR LES CONFIGURATIONS D'HYPERPARAMÈTRES

```
for i in range(num_particles):
    num_hidden_layers = int(hyperparameters[i, 0])
    num_neurons_per_layer = int(hyperparameters[i, 1])
    log_learning_rate = hyperparameters[i, 2]
    learning_rate = 10 ** log_learning_rate
    l2_weight = hyperparameters[i, 3]
```

Figure 46: configuration d'hyperparamètres

- À l'intérieur de la boucle, les hyperparamètres spécifiques à chaque configuration (nombre de couche cachées, nombre de neurones par couche, taux d'apprentissage et poids de régularisation L2) sont extraits à partir de ‘hyperparameters[i, :]’.

INITIALISATION DU MODÈLE

```
# Reset model and optimizer for each run
model = NeuralNetwork(num_hidden_layers, num_neurons_per_layer, learning_rate, l2_weight)
if torch.cuda.is_available():
    model.cuda()
```

Figure 47: Initialisation du modèle

- Un modèle de réseau de neurones ('model') est initialisé avec les hyperparamètres spécifiques de la configuration actuelle.
- Si une carte GPU est disponible (vérifié avec `torch.cuda.is_available()`), le modèle est déplacé sur le GPU.

PRÉPARATION DES DONNÉES

```
X_train_tensor = torch.tensor(X_train_scaled.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_array, dtype=torch.float32)
X_val_tensor = torch.tensor(X_val_scaled.values, dtype=torch.float32)
y_val_tensor = torch.tensor(y_val_array, dtype=torch.float32)
if torch.cuda.is_available():
    X_train_tensor, y_train_tensor = X_train_tensor.cuda(), y_train_tensor.cuda()
    X_val_tensor, y_val_tensor = X_val_tensor.cuda(), y_val_tensor.cuda()
```

Figure 48: préparation des données avec cuda

- Les données d'entraînement ('`X_train_scaled`') et de validation ('`X_val_scaled`') sont converties en tenseurs PyTorch et, si une carte GPU est disponible, sont également déplacées sur le GPU.

INITIALISATION DE L'OPTIMISEUR ET DE LA FONCTION PERTE

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
criterion = torch.nn.MSELoss()
```

Figure 49: Initialisation de l'optimiseur et de la fonction perte

- Un optimiseur Adam est créé pour le modèle avec le taux d'apprentissage spécifié.
- La fonction de perte est définie comme l'erreur quadratique moyenne (MSE) en utilisant '`torch.nn.MSELoss()`'.

ENTRAÎNEMENT DU MODÈLE

```
num_epochs = 50
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    # Add regularization loss
    regularization_loss = model.compute_regularization_loss()
    total_loss = loss + regularization_loss
    total_loss.backward()
    optimizer.step()
```

Figure 50: entraînement du modèle

- Le modèle est entraîné pendant un certain nombre d'époques ('num_epochs'), où les prédictions sont calculées sur les données d'entraînement, et la perte est calculée à l'aide de la MSE.
- Une perte de régularisation L2 est également calculée en utilisant la méthode 'compute_regularization_loss' définie précédemment dans la classe du modèle.
- La perte totale est la somme de la perte MSE et de la perte de régularisation L2.
- Les gradients sont calculés et les poids du modèle sont mis à jour en utilisant l'optimiseur.

EVALUATION DU MODÈLE SUR LES DONNÉES DE VALIDATION

```

model.eval()
with torch.no_grad():
    val_outputs = model(X_val_tensor)
    val_loss = criterion(val_outputs, y_val_tensor)
    validation_losses[i] = val_loss.item()
return validation_losses

```

Figure 51: évaluation du modèle sur les données de validation

- Le modèle est mis en mode évaluation ('model.eval()'), et les prédictions sont calculées sur les données de validation ('X_val_tensor').
- La perte de validation est calculée en utilisant la MSE entre les prédictions et les vraies valeurs de validation ('y_val_tensor').
- La perte de validation pour la configuration actuelle d'hyperparamètres est stockée dans le tableau 'validation_losses'.
- Une fois toutes les configurations d'hyperparamètres ont été évaluées, la fonction 'objective_function' renvoie le tableau des pertes de validation pour toutes les configurations.

OPTIMISATION AVEC PSO

DÉFINITION DES PLAGES D'HYPERPARAMÈTRES

```

# Modified hyperparameter ranges
hyperparameters = [(1, 6), (16, 512), (-6, -2), (1e-6, 1e-2)]

```

Figure 52: définition des plages d'hyperparamètres

- Les plages d'hyperparamètres sont définies dans la liste 'hyperparameters'. Chaque élément de cette liste représente une plage pour un hyperparamètre spécifique.

- (1,6) : Indique que le nombre de couches cachées doit être compris entre 1 et 6 inclus.
- (16, 512) : Indique que le nombre de neurones dans chaque couches doit être compris entre 16 et 512.
- (-6, -2) : Indique que la valeur du taux d'apprentissage doit être compris entre -6 et -2.
- (1e-6, 1e-2) : Indique que la valeur du poids de régularisation L2 doit être compris entre 1e-6 et 1e-2.

CONFIGURATION ET EXÉCUTION DE L'OPTIMISATEUR PSO

```
#Increase the number of particles and iterations
options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
optimizer = GlobalBestPSO(n_particles=50, dimensions=4, options=options)

# Perform PSO optimization with more iterations
best_objective, best_hyperparameters = optimizer.optimize(objective_function, iters=50)
```

Figure 53: configuration et exécution de l'optimisateur pso

- Les paramètres de l'optimisateur PSO sont configurés dans le dictionnaire ‘options’. Ces paramètres contrôlent le comportement de l'algorithme PSO, tels que les coefficients ‘c1’, ‘c2’ et ‘w’ qui déterminent comment les particules (solutions) évoluent dans l'espace de recherche.

Les paramètres c1, c2 et w sont utilisés pour régler le comportement et les performances de l'algorithme en contrôlant comment les particules dans l'essaim se déplacent et convergent vers une solution optimale.

- *c1 (coefficent cognitif) : Il contrôle dans quelle mesure chaque particule est influencée par sa propre expérience personnelle. Un c1 élevé incite les particules à se rappeler de leurs meilleures positions passées, favorisant ainsi l'exploration de solutions potentiellement meilleures.*
- *c2 (coefficent social) : Il détermine l'influence des meilleures positions trouvées par d'autres particules de l'essaim sur chaque particule. Un c2 élevé encourage les particules à se regrouper autour des meilleures positions trouvées par leurs pairs, ce qui aide à converger plus rapidement vers une solution optimale.*
- *w (poids d'inertie) : Il contrôle la vitesse de convergence de l'algorithme. Un w élevé permet aux particules de maintenir leur vitesse actuelle, favorisant l'exploration et aidant à éviter de rester coincé dans des minima locaux. Un w faible fait ralentir les particules au fil du temps, ce qui favorise la convergence vers une solution, mais peut potentiellement les piéger dans des minima locaux.*

TESTER LES MEILLEURS HYPERPARAMÈTRES

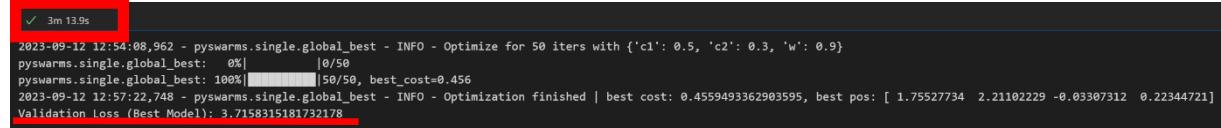
```
# Instead of unpacking hyperparameters in the objective_function, use it as is
validation_loss = objective_function(np.array([best_hyperparameters]))[0]

# Print the validation loss of the best model
print("Validation Loss (Best Model):", validation_loss)
```

Figure 54: tester les meilleurs hyperparamètres

- On passe les hyperparamètres à la fonction ('objective_function') en tant que tableau unique dans 'np.array([best_hyperparameters])'.
- La perte de validation du meilleur modèle est imprimée avec le message 'Validation Loss (Best Model) :' suivi de la valeur de la perte.

RÉSULTAT D'OPTIMISATION



```
✓ 3m 13.9s
2023-09-12 12:54:08,962 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 0%|██████████| 0/50
pyswarms.single.global_best: 100%|██████████| 50/50, best_cost=0.456
2023-09-12 12:57:22,748 - pyswarms.single.global_best - INFO - Optimization finished | best cost: 0.4559493362903595, best pos: [ 1.75527734  2.21102229 -0.03307312  0.22344721]
Validation Loss (Best Model): 3.7158315181732178
```

Figure 55: résultat de l'optimisation

C'est la fin de l'optimisation, et elle fournit un résumé des résultats.

- Le "best cost" (meilleur coût) indique la valeur de l'objectif la plus basse atteinte au cours de l'optimisation PSO. Dans ce cas, le meilleur coût est d'environ 0,456.
- "best pos" (meilleure position) indique les hyperparamètres correspondant au meilleur coût. Ces valeurs sont [1,75527734, 2,21102229, -0,03307312, 0,22344721].
- Validation Loss (Best Model): 3.7158315181732178 : Après avoir trouvé les meilleurs hyperparamètres, le modèle associé à ces hyperparamètres est évalué sur les données de validation. La perte de validation obtenue avec le meilleur modèle est affichée. La perte de validation est d'environ 3,716.

Dans ce résultat on peut constater que le temps d'exécution de la création du modèle ANN et optimisation PSO avec CUDA prend 3 MIN ET 13.9 SECONDES.

COMPARAISON D'EXÉCUTION AVEC CUDA ET SON CUDA

Afin de mieux examiner l'utilité et l'efficacité d'utilisation de CUDA dans notre modèle, on teste la création du même modèle ANN et optimisation à l'aide de l'algorithme pso sans utilisation de CUDA.

```

✓ 63m 33.1s
Model: "model_118"

Layer (type)          Output Shape         Param #
=====
input_layer (InputLayer)  [(None, 10)]        0
dense_26 (Dense)       (None, 64)           704
dense_27 (Dense)       (None, 64)           4160
dense_28 (Dense)       (None, 64)           4160
output_layer (Dense)   (None, 1)            65
=====
Total params: 9,089
Trainable params: 9,089
Non-trainable params: 0

Stopping search: maximum iterations reached --> 50
Best Hyperparameters:
Number of Hidden Layers: 0
Number of Neurons per Layer: 0
Learning Rate: 1.0
Validation Loss (Best Objective): 0.0030059022828936577

```

Figure 56: PSO + ANN sans CUDA

Comme on peut remarquer à partir de ce journal d'exécution, un simple code d'optimisation de réseau de neurones prend **63 MIN ET 33.1 SECONDES**. Et on peut aussi remarquer que l'optimisation n'est même pas réussite.

Ceci prouve que l'utilisation de CUDA est cruciale dans l'optimisation de ce modèle car c'est grâce au GPU que le modèle que nous avons créé peut essayer plusieurs hyperparamètres à l'aide de l'algorithme pso et trouver les meilleurs et cela dans un temps minimale.

ÉVALUATION DU MODÈLE

Afin de s'assurer de la qualité de notre modèle, on réalise une évaluation sur un ensemble de données de validation distinct de l'ensemble d'entraînement.

Dans ce code il y a deux métriques d'évaluation de la performance de votre modèle de réseau de neurones.

1. Validation MSE (Mean Squared Error)
2. Validation R-squared (R^2)

```

# Convert best_hyperparameters to a NumPy array
best_hyperparameters = np.array(best_hyperparameters)
# Update the best hyperparameters extraction
best_num_hidden_layers = int(best_hyperparameters[0])
best_num_neurons_per_layer = int(best_hyperparameters[1])
best_log_learning_rate = int(best_hyperparameters[2])
best_learning_rate = 10 ** best_log_learning_rate
best_l2_weight = best_hyperparameters[3]
# Create the best neural network based on the best hyperparameters
best_model = NeuralNetwork(
    best_num_hidden_layers, best_num_neurons_per_layer, best_learning_rate, best_l2_weight)
# Define a function for model evaluation
def evaluate_model(model, X, y):
    model.eval() # Set the model to evaluation mode
    with torch.no_grad():
        X_tensor = torch.tensor(X, dtype=torch.float32)
        y_true = torch.tensor(y, dtype=torch.float32)
        # If GPU is available, move data to GPU
        if torch.cuda.is_available():
            X_tensor = X_tensor.cuda()
            y_true = y_true.cuda()
        # Make predictions
        y_pred = model(X_tensor)
        # Convert predictions and true labels back to CPU if necessary
        y_pred = y_pred.cpu().numpy()
        y_true = y_true.cpu().numpy()
        # Calculate Mean Squared Error (MSE) and R-squared (R²)
        mse = mean_squared_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)
    return mse, r2
mse_val, r2_val = evaluate_model([best_model, X_val_scaled.values, y_val.values])
print("Validation MSE:", mse_val)
print("Validation R-squared (R²):", r2_val)

```

[15]

```

... Validation MSE: 4.356079
Validation R-squared (R²): 0.9888293617729995

```

Figure 57: évaluation du modèle

L'output de ce code fournit deux métriques d'évaluation de la performance du modèle sur un ensemble de données de validation :

- Validation MSE (Mean Squared Error) : 4.356079
 - La MSE est une mesure de l'erreur moyenne au carré entre les valeurs prédites par votre modèle et les vraies valeurs (cibles) dans l'ensemble de données de validation.
 - Une MSE plus basse indique une meilleure performance du modèle. Dans ce cas, une MSE de 4.356079 signifie que, en moyenne, les prédictions du modèle ont une erreur au carré de 4.356079 par rapport aux vraies valeurs de l'ensemble de validation.
 - En général, une MSE proche de zéro indique que le modèle prédit très bien les données de validation, tandis qu'une MSE élevée indique que le modèle a du mal à faire des prédictions précises.
- Validation R-squared (R²) : 0.9888293617729995
 - Le R-squared, également appelé coefficient de détermination, mesure la proportion de la variance dans les données de validation qui est expliquée par les prédictions du modèle.

- Le R^2 varie de 0 à 1, où 1 signifie que le modèle explique parfaitement la variance des données et 0 signifie que le modèle n'explique aucune variance (performances médiocres).
- Un R^2 élevé, comme 0.9888293617729995 dans ce cas, indique que le modèle explique très bien la variance des données de validation. En d'autres termes, le modèle parvient à capturer la variation dans les données de validation de manière très précise.

Interprétation générale :

- Les valeurs obtenues (MSE basse et R^2 élevé) suggèrent que le modèle fonctionne très bien sur l'ensemble de données de validation. Il semble être capable de faire des prédictions très précises, avec une MSE relativement faible et un R^2 proche de 1, ce qui signifie que la majorité de la variance dans les données de validation est expliquée par les prédictions du modèle.
- En pratique, une MSE de 4.356079 et un R^2 de 0.9888293617729995 sont généralement considérés comme d'excellentes performances, indiquant un modèle qui s'ajuste bien aux données de validation.

VISUALISATION DES RÉSULTATS

VALEUR ACTUEL VS. VALEUR ACTUEL (SCATTER PLOT)

Actual vs. Predicted Values (Validation Set)

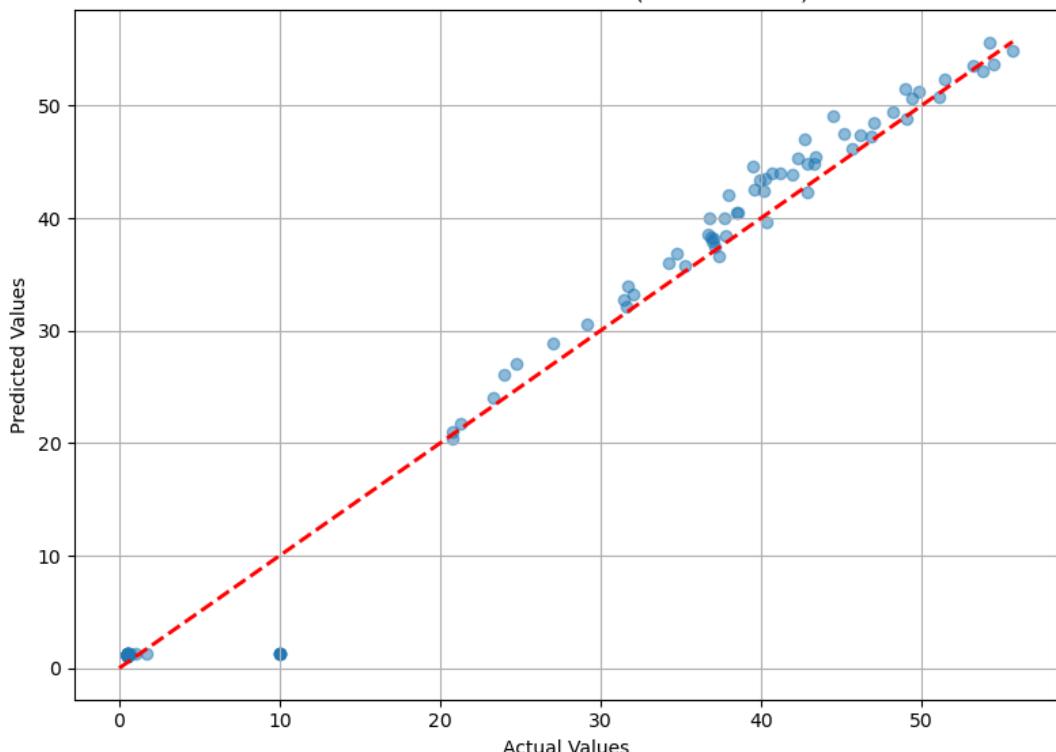


Figure 58: graphe de dispersion

- Chaque point sur le graphique représente une paire de valeurs : la valeur réelle (horizontale) et la valeur prédictée (verticale) pour un échantillon de l'ensemble de données de validation.
- Les points sont dispersés autour de la ligne diagonale en pointillés rouges. Plus les points sont proches de cette ligne, meilleure est la qualité des prédictions du modèle.
- La comparaison visuelle entre les valeurs réelles et les valeurs prédictées permet d'évaluer rapidement la performance du modèle. Si les points suivent étroitement la ligne rouge, cela indique que le modèle fait des prédictions précises. Si les points sont dispersés de manière aléatoire, cela suggère que le modèle a du mal à prédire avec précision.

Interprétation du graphique :

Dans ce graphe on peut visualiser que les points bleus sont regroupés autour de la ligne en pointillés rouge, cela indique généralement que les prédictions du modèle sont très proches des valeurs réelles dans l'ensemble de données de validation. C'est une très bonne indication de la qualité de prédictions du modèle.

ÉVOLUTION DES PERTES D'ENTRAÎNEMENT ET DE VALIDATION

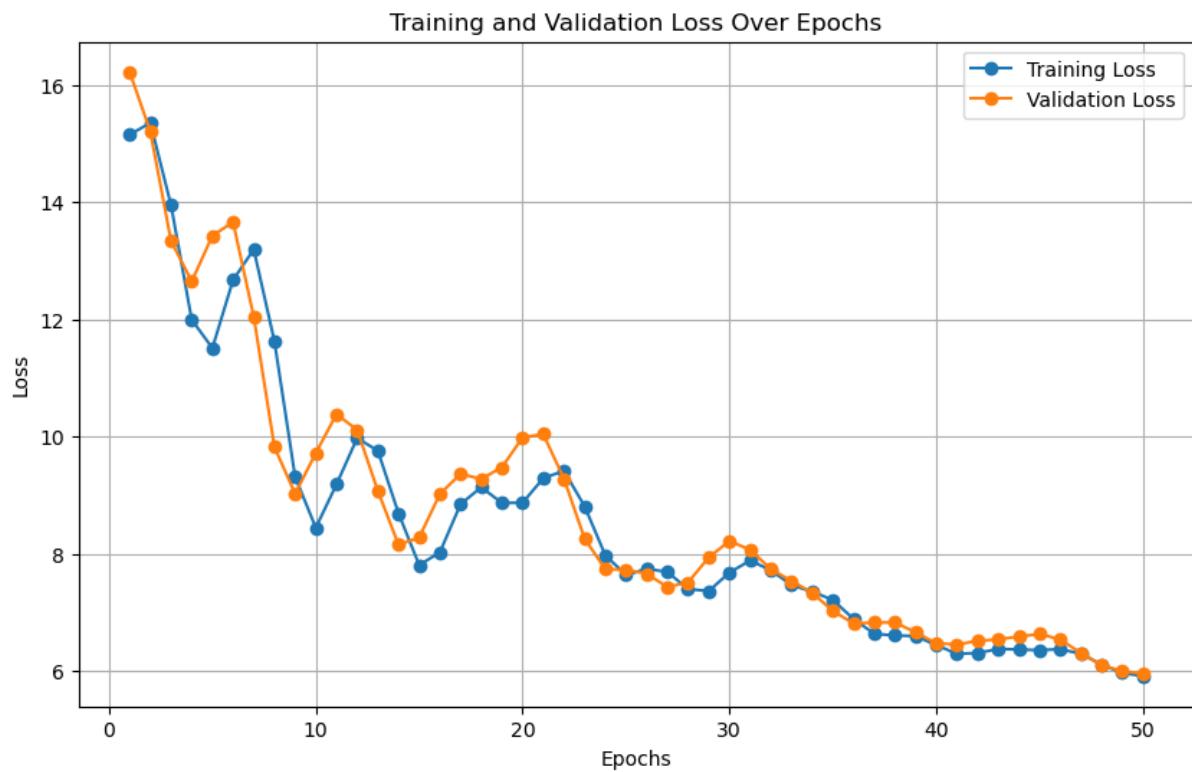


Figure 59: évolution des pertes d'entraînement et de validation

- La courbe bleue représente l'évolution de la perte d'entraînement, ce qui signifie comment la perte diminue au fur et à mesure que le modèle s'entraîne sur les données

d'entraînement. On s'attend généralement à ce que cette perte diminue au fil des époques, car le modèle s'ajuste progressivement aux données.

- La courbe orange représente l'évolution de la perte de validation, ce qui indique comment le modèle généralise sur des données qu'il n'a pas vu pendant l'entraînement. Une baisse constante de cette perte est souhaitée, mais une augmentation ou un plateau peuvent indiquer un surajustement si la perte de validation augmente tandis que la perte d'entraînement diminue.
- Lorsque les deux courbes suivent une trajectoire similaire et descendent ensemble, cela indique que le modèle s'entraîne bien et généralise bien.
- La convergence des courbes (lorsqu'elles atteignent un plateau) peut indiquer que l'entraînement a atteint un point optimal, et il peut ne pas être nécessaire de poursuivre l'entraînement.

Interprétation du graphique :

On remarque que les courbes convergent vers des valeurs de perte plus basses, cela suggère que le modèle a réussi à apprendre les modèles dans les données d'entraînement et à généraliser sur les données de validation. Le plateau à la fin peut indiquer que l'entraînement a atteint un point optimal, et de nouvelles époques d'entraînement pourraient ne pas améliorer significativement les performances.

COMPARAISON DES VALEURS RÉELLES ET PRÉDITES

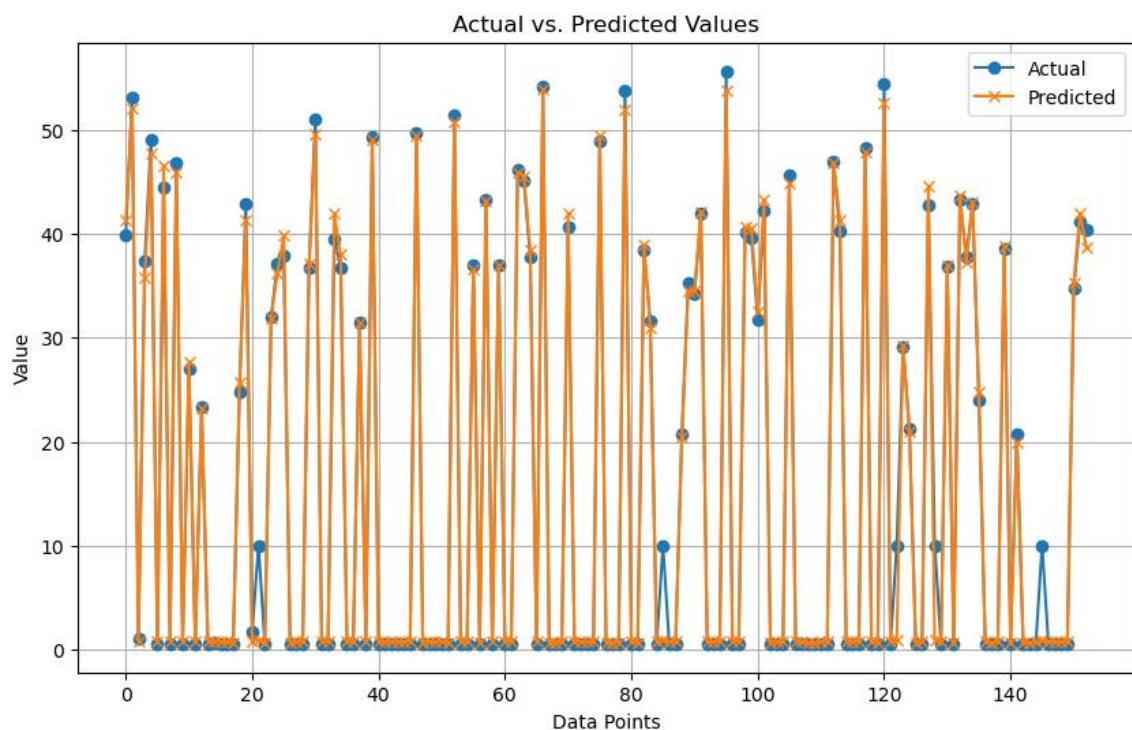


Figure 60: comparaison des valeurs réelles et prédites

- L'objectif principal de ce graphique est d'évaluer visuellement la performance du modèle en comparant les valeurs réelles aux valeurs prédictives.
- Lorsque les points bleus et les croix orange sont alignés étroitement, cela indique que le modèle effectue des prédictions précises. En d'autres termes, les valeurs prédictives sont très proches des vraies valeurs.
- Si les points orange se chevauchent fortement avec les points bleus, cela indique que le modèle a une bonne capacité à généraliser et à faire des prédictions précises sur des données qu'il n'a pas vu pendant l'entraînement.
- Cependant, si les points orange sont dispersés et ne suivent pas bien la ligne des points bleus, cela peut indiquer que le modèle a des difficultés à faire des prédictions précises.

Interprétation du graphe :

On remarque que dans le graphe les croix orange sont très proches des points bleus, cela indique que le modèle a réussi à bien apprendre et à bien généraliser, ce qui est l'objectif principal de l'entraînement du modèle. C'est un signe positif de la qualité de ce dernier.

CONCLUSION

Ce projet de recherche a exploré avec succès l'utilisation de CUDA, l'algorithme d'optimisation PSO (Particle Swarm Optimization), et les réseaux de neurones pour la prédiction des dépenses énergétiques dans les bâtiments. L'objectif était de déterminer comment CUDA peut être une ressource puissante pour améliorer les performances des modèles d'apprentissage automatique, en particulier dans le contexte de l'analyse des données énergétiques.

Les résultats obtenus démontrent clairement que l'intégration de CUDA a considérablement accéléré l'entraînement et l'exécution du modèle, offrant une augmentation significative des performances par rapport à une implémentation basée sur CPU. Cette accélération a permis de réduire considérablement les temps de calcul, ce qui est essentiel pour des applications en temps réel ou des analyses de données à grande échelle.

L'algorithme PSO a également joué un rôle crucial en optimisant les paramètres du réseau de neurones, améliorant ainsi la précision des prédictions. Cette approche d'optimisation a permis au modèle de s'adapter de manière plus efficace aux spécificités des données énergétiques, contribuant ainsi à une meilleure adéquation aux réalités du domaine.

En fin de compte, ce projet a ouvert la voie à de nouvelles opportunités pour l'utilisation de l'accélération matérielle, telle que CUDA, dans le domaine de l'apprentissage automatique appliquée à l'énergie et aux bâtiments. Les implications vont au-delà de la simple prédiction des dépenses énergétiques, car elles pourraient s'étendre à d'autres applications nécessitant des calculs intensifs en parallèle.

Cependant, il est important de noter que l'optimisation de projets d'apprentissage automatique avec CUDA peut également poser des défis, notamment en termes de gestion de la mémoire GPU et de synchronisation. Une planification minutieuse et une connaissance approfondie des caractéristiques de CUDA sont essentielles pour maximiser ses avantages.

En conclusion, ce projet a apporté une contribution significative à la manière dont CUDA, PSO et les réseaux de neurones peuvent être combinés pour résoudre des problèmes complexes en matière de dépenses énergétiques dans les bâtiments. Il a montré que l'utilisation judicieuse de technologies avancées peut ouvrir de nouvelles perspectives pour l'efficacité énergétique et l'analyse des données dans le secteur de la construction, ouvrant ainsi la voie à des développements futurs passionnants dans ce domaine.

RÉFÉRENCES

- Nvidia Cuda installation guide for Microsoft Windows
https://www.clear.rice.edu/comp422/resources/cuda/pdf/CUDA_Installation_Guide_Windows.pdf
- Training a Neural Network with PySwarms
https://pyswarms.readthedocs.io/en/development/examples/custom_objective_function.html
- Implementing Particle Swarm Optimization using Python
<https://www.analyticsvidhya.com/blog/2021/11/implementing-a-particle-swarm-optimization-with-python/>
- Designing Artificial Neural Network Using Particle Swarm Optimization: A Survey
<https://www.intechopen.com/chapters/82833>
- Particle Swarm Optimisation for Evolving Deep Neural Networks
<https://arxiv.org/pdf/1907.12659.pdf>
- CUDA Neural Network Implementation (Part 1)
<https://luniak.io/cuda-neural-network-implementation-part-1/>
- NVIDIA cuDNN
<https://developer.nvidia.com/cudnn>
- NVIDIA CUDA Basics and Best Practices
<https://www.run.ai/guides/nvidia-cuda-basics-and-best-practices>