République Algérienne Démocratique Et Populaire
Ministère De L'enseignement Supérieur Et De La Recherche Scientifique
Université Mouloud Mammeri Tizi-Ouzou
Faculté De Génie Electrique Et D'informatique
Département D'électronique

Mémoire de fin d'étude

En vue de l'obtention du diplôme de Master en électronique

Option : Systèmes embarqués

Thème

Implémentation d'un réseau de neurone ANN sur ESP32 pour la surveillance de diabète

Proposé et encadré par : Pr. M. Laghrouche

Développé par : LOUIZINI Ahmed
ALMABOUDI Massissilia

Année universitaire : 2023/2024

Remerciements

Je dédie ce travail à mes chers parents qui ont su me donner du courage et du soutien tout au long de mon cursus.

A mes chers frères et sœurs MASSINISSA, LICIA et mon cher mari qui étaient toujours à mes cotés

Mes remerciements les plus sincères vont à tous ceux qui m'ont soutenu, en particulier mon promoteur Mr Laghrouche et ses conseils et orientation.

Sans oublier mon binôme Louizini Ahmed

Je remercie aussi les membres du jury.

ALMABOUDI Massissilia

Remerciements

Je tiens à exprimer toute ma reconnaissance à mon directeur de mémoire Mr Laghrouche. Je la remercie de m'avoir encadré, orienté, aidé et conseillé.

J'adresse mes sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté de me rencontrer et de répondre à mes questions durant mes recherches.

Je remercie mes très chers parents, qui ont toujours été là pour moi. Je remercie mes sœurs pour leurs encouragements.

Enfin, je remercie mes amis qui ont toujours été là pour moi. Leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

LOUIZINI Ahmed

Sommaire

Liste des schémas et tableaux

Introduction Générale

Introduction Générale

L'intelligence artificielle (IA) est aujourd'hui une force motrice derrière des innovations technologiques majeures, touchant divers aspects de notre quotidien et transformant de nombreuses industries. Au cœur de cette révolution se trouvent les réseaux de neurones artificiels (RNA), une technologie inspirée du fonctionnement du cerveau humain. Les RNA ont démontré leur capacité à résoudre des problèmes complexes dans des domaines variés tels que la vision par ordinateur, le traitement du langage naturel et la prédiction de maladies. Ce mémoire se propose d'explorer les fondements théoriques des RNA, leurs méthodes d'implémentation, et leurs applications pratiques, avec un accent particulier sur la prédiction du diabète.

Les maladies chroniques comme le diabète représentent un défi majeur pour les systèmes de santé à travers le monde. La capacité de prédire et de détecter précocement le diabète pourrait améliorer considérablement les soins aux patients et réduire les coûts de traitement. Les réseaux de neurones artificiels offrent une solution prometteuse pour le développement de systèmes de diagnostic précis et automatisés. Ce mémoire examine comment les RNA peuvent être utilisés pour créer un système de prédiction du diabète efficace et intégré.

Objectifs

1. Comprendre les fondements théoriques des réseaux de neurones artificiels : Explorer les modèles mathématiques et les différentes architectures de RNA.
2. Implémenter des réseaux de neurones sur diverses plateformes : Étudier l'implémentation des RNA sur des logiciels comme MATLAB et des microcontrôleurs tels que l'Arduino ESP32.
3. Déployer un modèle de RNA pour la prédiction du diabète : Développer et tester un système intégré capable de prédire le diabète à partir de données réelles.

Ce mémoire est structuré en trois chapitres principaux, chacun abordant un aspect clé de l'étude des réseaux de neurones artificiels.

Le premier chapitre fournit une base théorique solide sur les RNA. Nous commençons par présenter le modèle mathématique des RNA, couvrant les réseaux de neurones artificiels (ANN), les réseaux de neurones récurrents (RNN) et les réseaux de neurones Convolutionnels (CNN). Ensuite, nous explorons les différents types de réseaux, en distinguant les réseaux feedforward et feedback. Nous détaillons également le processus d'apprentissage des RNA, incluant l'initialisation des poids, la propagation avant, le calcul de l'erreur, la rétropropagation du gradient et la mise à jour des poids. Ce chapitre se termine par une discussion sur les domaines d'application et les avantages et inconvénients des RNA.

Introduction Générale

Le deuxième chapitre est consacré à l'implémentation pratique des RNA. Nous décrivons comment les réseaux de neurones peuvent être développés et utilisés sur des logiciels comme MATLAB et des environnements de développement comme Visual Studio Code. Une attention particulière est portée à l'utilisation de microcontrôleurs, notamment l'Arduino ESP32, pour l'implémentation des RNA dans des systèmes embarqués. Ce chapitre comprend des exemples détaillés d'implémentation et des études de cas pratiques.

Le dernier chapitre se concentre sur le déploiement d'un modèle de RNA pour la prédiction du diabète. Nous décrivons une application spécifique de prédiction du diabète en utilisant un ANN, en commençant par une description de la maladie et des techniques d'apprentissage automatique pertinentes. Ensuite, nous détaillons le développement d'un système de détection de diabète intégré, incluant la recherche de bases de données, l'entraînement du modèle avec Python, la conversion vers TensorFlow Lite et la mise en œuvre sur une plateforme Arduino. Nous terminons par la création d'une application mobile pour l'interface utilisateur et une discussion sur les limitations et les choix de paramètres.

Ce mémoire explore les aspects théoriques, pratiques et applications des réseaux de neurones artificiels (RNA) pour démontrer leur potentiel dans le domaine de la santé, notamment pour la prédiction du diabète. Les résultats visent à améliorer la compréhension des RNA et leur utilisation dans des systèmes de diagnostic médical avancés.

Chapitre 1 : Les Réseaux de Neurones Artificiels

Chapitre 1 : Les Réseaux de Neurones

Introduction

Depuis l'avènement des premiers ordinateurs dans les années 40, le domaine de l'informatique a connu une évolution constante. Chaque année apporte son lot de nouveautés, rendant les ordinateurs d'aujourd'hui nettement plus puissants que ceux d'il y a quelques décennies. Malgré ces avancées, la résolution de problèmes d'optimisation complexes demeure un défi majeur pour les informaticiens. Pour surmonter ces obstacles, une multitude d'algorithmes ont été développés, parmi lesquels figurent les réseaux de neurones artificiels (Artificial Neural Networks).

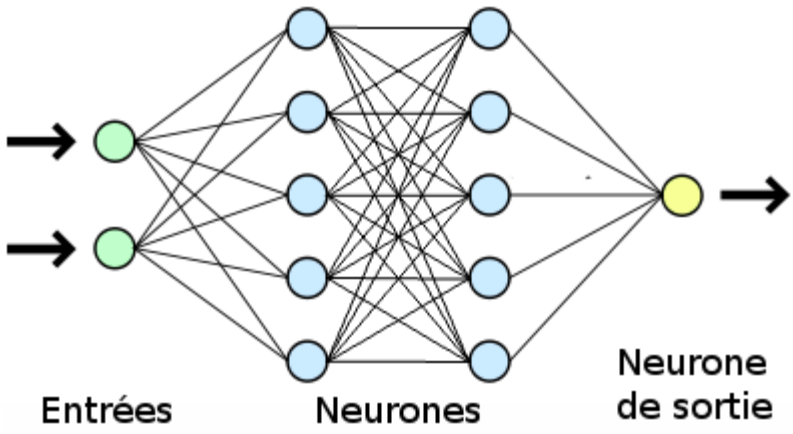


Figure 1 Réseau de neurone

Les réseaux de neurones s'inspirent du fonctionnement biologique du cerveau humain et adoptent une approche mathématique pour imiter ses processus. Ce chapitre se consacre à l'étude des réseaux de neurones artificiels, en explorant leur modèle mathématique, leurs différentes architectures, ainsi que leurs domaines d'application variés.

1. Modèle Mathématique des Réseaux de Neurones Artificiels

1.1. Réseaux de Neurones Artificiels (ANN)

Les Réseaux de Neurones Artificiels, ou ANNs, sont des modèles de réseau de neurones inspirés du fonctionnement du cerveau humain. Ils sont utilisés pour la classification, la régression et d'autres tâches liées à l'apprentissage supervisé. Les ANNs sont composés de neurones artificiels organisés en couches, comprenant une couche d'entrée, des couches cachées et une couche de sortie. Ils sont capables d'apprendre à partir de données en ajustant les poids des connexions entre les neurones.

Structure des ANNs :

- Couche d'entrée : Reçoit les données brutes.
- Couches cachées : Effectuent des transformations et extraient des caractéristiques.
- Couche de sortie : Produit la prédiction finale.

Caractéristiques des ANNs :

- Capacité à apprendre des modèles complexes : Grâce à leurs multiples couches, les ANNs peuvent capturer des relations non linéaires dans les données.
- Généralisation à de nouvelles données : Les ANNs peuvent appliquer les connaissances acquises à des données non vues pendant l'entraînement.
- Problèmes de surapprentissage : Les ANNs peuvent mémoriser les données d'entraînement au lieu de généraliser, nécessitant des techniques de régularisation.
- Nécessité d'un grand volume de données : Plus de données permettent une meilleure performance et une meilleure généralisation.

1.2. Réseaux de Neurones Récurrents (RNN)

Les Réseaux de Neurones Récurrents, ou RNNs, sont conçus pour traiter des données séquentielles et temporelles. Contrairement aux ANNs classiques, les RNNs possèdent des connexions récurrentes qui leur permettent de conserver une mémoire des informations précédentes. Cela les rend efficaces pour des tâches telles que la prédiction de séquences, la traduction automatique et le traitement du langage naturel.

Structure des RNNs :

Neurones récurrents : Chaque neurone reçoit des entrées de la couche précédente ainsi que des rétroactions de sa propre sortie précédente.

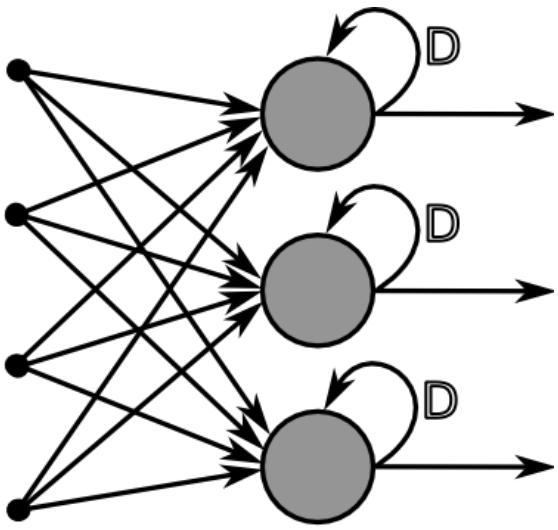


Figure 2 Schéma d'un réseau de neurones artificiels récurrents

Applications des RNNs :

- Reconnaissance de la parole : Transcription de l'audio en texte.
- Génération de texte : Production de texte à partir d'un modèle entraîné.
- Modélisation de séries temporelles : Prédiction de valeurs futures basées sur des données historiques.

Challenges des RNNs :

- Disparition/explosion du gradient : Les gradients peuvent devenir extrêmement petits ou grands, rendant l'entraînement difficile.
- Capacité limitée à traiter des séquences longues : Des architectures comme les LSTM et GRU ont été développées pour pallier ces problèmes.

1.3. Réseaux de Neurones Convolutionnels (CNN)

Les Réseaux de Neurones Convolutionnels (CNN), également connus sous le nom de ConvNets, sont des architectures de réseau de neurones conçues spécifiquement pour le traitement des données structurées en grilles, comme les images ou les séquences temporelles. Ils sont caractérisés par l'utilisation de couches de convolution qui filtrent et extraient des caractéristiques locales pertinentes à partir des données en entrée.

Structure des CNNs :

- Couches de convolution : Appliquent des filtres pour détecter des motifs locaux.
- Couches de pooling : Réduisent la dimensionnalité et la sensibilité aux variations.
- Couches entièrement connectées : Effectuent des classifications basées sur les caractéristiques extraites.

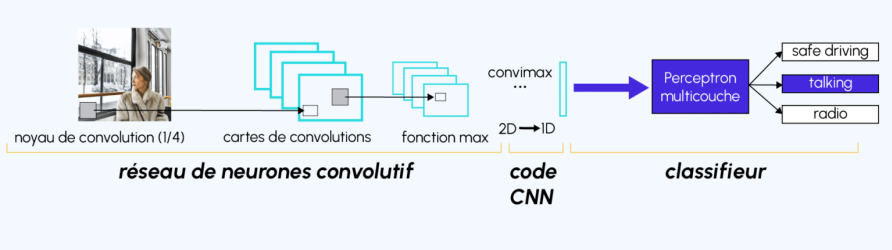


Figure 3 Schéma représentant l'architecture d'un CNN

Applications des CNNs :

- Classification d'images : Identification d'objets dans des images.
- Détection d'objets : Localisation et identification d'objets dans des images.
- Segmentation d'images : Division d'images en segments significatifs.

2. Types de Réseaux de Neurones

2.1. Réseaux Feedforward (Réseaux à propagation directe)

Les réseaux feedforward sont des réseaux de neurones où l'information circule dans une seule direction, de la couche d'entrée à la couche de sortie, sans boucles de rétroaction. Ce type de réseau est utilisé pour traiter des données statiques où la séquence des informations n'a pas d'importance.

Caractéristiques des Réseaux Feedforward :

- Flux unidirectionnel : Les données se déplacent uniquement vers l'avant.
- Application : Classification d'images, reconnaissance de motifs.

2.2. Réseaux Feedback (Réseaux récurrents)

Les réseaux feedback, également appelés réseaux récurrents, sont des réseaux de neurones où les informations peuvent circuler dans des boucles de rétroaction. Contrairement aux réseaux feedforward, les réseaux feedback ont des connexions récurrentes qui permettent à l'information de revenir à des étapes antérieures du traitement.

Caractéristiques des Réseaux Feedback :

- Rétroactions internes : Permettent la mémorisation des états précédents.
- Application : Modélisation de séquences, prédiction de séries temporelles.

3. Classification par Types de Réseaux de Neurones

Réseaux Feedforward

- ANN : Réseaux de Neurones Artificiels qui suivent le modèle classique où les données se propagent de la couche d'entrée à la couche de sortie sans boucles de rétroaction.
- CNN : Réseaux de Neurones Convolutionnels conçus pour le traitement des données spatiales, telles que les images.

Réseaux Feedback

RNN : Réseaux de Neurones Récurrents utilisant des connexions récurrentes pour traiter des données séquentielles et temporelles.

4. Processus d'Apprentissage dans les Réseaux de Neurones

4.1. Initialisation des Poids

Les poids du réseau sont initialisés de manière aléatoire ou avec des valeurs prédéfinies. Ces poids déterminent l'importance relative des différentes entrées pour les prédictions du réseau.

4.2. Propagation Avant

Lorsqu'une donnée est introduite dans le réseau, elle est propagée à travers les différentes couches (neurones) du réseau selon les poids et les fonctions d'activation définies. Cela produit une prédiction initiale du réseau.

4.3. Calcul de l'Erreur

L'erreur entre la prédiction du réseau et la vérité terrain (étiquette réelle) est calculée à l'aide d'une fonction de perte ou de coût. Cette erreur mesure la qualité de la prédiction par rapport aux données d'entraînement.

4.4. Rétropropagation du Gradient

L'algorithme de rétropropagation du gradient est utilisé pour calculer comment les poids du réseau doivent être ajustés pour réduire l'erreur. Ce processus utilise le gradient de la fonction de perte par rapport aux poids du réseau pour déterminer la direction dans laquelle les poids doivent être modifiés pour améliorer la prédiction.

4.5. Mise à Jour des Poids

Les poids du réseau sont mis à jour en fonction des gradients calculés lors de la rétropropagation. Cette mise à jour peut être effectuée à l'aide d'algorithmes d'optimisation tels que la descente de gradient stochastique (SGD) ou ses variantes comme Adam, RMSProp, etc.

4.6. Répétition du Processus

Ce processus d'ajustement des poids et de mise à jour est répété pour de multiples itérations (époques) sur l'ensemble des données d'entraînement jusqu'à ce que le réseau atteigne une performance satisfaisante ou que certains critères d'arrêt soient atteints (par exemple, convergence de l'erreur).

5. Domaines d'Application

ANNs :

- Finance : Détection de fraude, prédiction de marchés.
- Médecine : Diagnostic de maladies, analyse d'images médicales.
- Marketing : Analyse de comportements, recommandation de produits.

RNNs :

- Traduction automatique : Conversion de textes d'une langue à une autre.
- Génération de texte : Création de contenu textuel à partir d'exemples d'entraînement.
- Modélisation de séries temporelles : Prédiction de tendances futures sur la base de données historiques.

CNNs :

- Reconnaissance faciale : Identification de visages humains dans des images.
- Classification d'images : Détection et reconnaissance d'objets dans des images.
- Segmentation d'images : Division d'images en segments significatifs pour l'analyse détaillée.

6. Avantages et Inconvénients des Réseaux de Neurones

ANNs

- Avantages :

- Flexibilité : Capacité à s'adapter à une variété de tâches d'apprentissage.
- Capture des relations complexes : Capacité à modéliser des relations non linéaires dans les données.

- Inconvénients :

- Besoin de grandes quantités de données : Les performances des ANNs sont améliorées avec un grand volume de données d'entraînement.
- Risque de surapprentissage : Les ANNs peuvent mémoriser les données d'entraînement au lieu de généraliser à de nouvelles données.

RNNs

- Avantages :

- Traitement des séquences de données : Capacité à gérer les données temporelles et séquentielles.

- Inconvénients :

- Problèmes de disparition/explosion du gradient : Difficulté à entraîner les RNNs sur de longues séquences de données.

CNNs

- Avantages :

- Efficacité pour les données visuelles : Excellentes performances pour le traitement des images et des vidéos.
- Réduction du nombre de paramètres : Les couches de convolution réduisent le nombre de paramètres nécessaires par rapport aux couches entièrement connectées.

- Inconvénients :

- Inadaptés pour les données séquentielles : Les CNNs ne sont pas optimisés pour le traitement des séquences de données.
- Opacité : Difficulté à interpréter les décisions prises par le modèle.

7. Liste des Bibliothèques Utilisées

Pandas : Pandas est une bibliothèque de manipulation et d'analyse de données en Python, particulièrement adaptée pour les données tabulaires. Elle offre des structures de données flexibles comme DataFrame, qui permet de manipuler facilement les données en effectuant des opérations comme le filtrage, l'agrégation et la fusion.

Scikitlearn (SKLearn) : Scikitlearn est une bibliothèque Python qui fournit des outils simples et efficaces pour l'analyse de données et l'apprentissage automatique. Elle inclut des algorithmes pour la classification, la régression, le clustering et la réduction de dimensionnalité.

TensorFlow : TensorFlow est une bibliothèque open source de machine learning développée par Google. Elle permet de créer et d'entraîner des modèles de réseaux de neurones pour des tâches variées, telles que la classification d'images, la traduction automatique et la prédiction de séries temporelles.

Figure 4 TensorFlow logo

Keras : Keras est une API de réseau de neurones de haut niveau, écrite en Python et capable de s'exécuter sur TensorFlow. Elle permet de construire et d'entraîner facilement des modèles de Deep Learning avec une interface utilisateur intuitive.

TFLiteConverter : TFLiteConverter est un outil de TensorFlow permettant de convertir des modèles TensorFlow en format TensorFlow Lite pour une exécution efficace sur des dispositifs à ressources limitées comme les smartphones, les microcontrôleurs et les appareils IoT.

TensorFlow Lite for Microcontrôleurs : TensorFlow Lite for Microcontrollers est une version de TensorFlow Lite conçue pour exécuter des modèles de machine learning sur des microcontrôleurs avec des ressources limitées, offrant des performances optimisées pour les applications embarquées.

DroidScript : DroidScript est un environnement de développement intégré (IDE) qui permet de coder des applications Android en utilisant JavaScript.

Conclusion

Ce chapitre a examiné différents types de réseaux de neurones artificiels, notamment les ANN, CNN et RNN, en abordant leur structure, leurs caractéristiques et leurs applications, telles que la reconnaissance d'images et la prédiction de séries temporelles. Nous avons discuté des processus d'apprentissage des réseaux de neurones, y compris l'initialisation des poids, la propagation avant, le calcul de l'erreur, la rétropropagation et la mise à jour des poids, essentiels pour comprendre leur fonctionnement.

Nous avons également analysé les avantages et inconvénients de chaque type de réseau : les ANNs sont polyvalents mais nécessitent beaucoup de données et risquent le sur-apprentissage ; les RNNs sont bons pour les données séquentielles mais peuvent rencontrer des problèmes de gradients ; les CNNs sont efficaces pour les données visuelles mais pas pour les données séquentielles.

Le prochain chapitre approfondira l'implémentation des réseaux de neurones en utilisant MATLAB et Python, en se concentrant sur les microcontrôleurs. Cette exploration mettra en lumière les défis et opportunités de l'embarquement de modèles de machine learning dans des dispositifs à ressources limitées, ainsi que les meilleures pratiques pour optimiser les performances des modèles déployés.

Chapitre 02 :
L'implémentation d'un réseau de neurone

Chapitre 02 : L'implémentation d'un réseau de neurone

Introduction :

Dans ce deuxième chapitre, on parlera de "l'implémentation" ou d'un autre sens de "l'intégration" des algorithmes dans différents supports matérielles ; on a choisi de trois supports distincts, à chacun d'eux son mode de fonctionnement, ses caractéristiques, sa méthode à produire un résultat et bien évidemment à chacun son implémentation.

On commencera par introduire une définition utile à l'implémentation, ensuite on va voir les étapes implémentation sur des logiciels et des microcontrôleurs, on s'attaquera ensuite directement à la description, et à la présentation de ces supports en question, et on indiquera la procédure à suivre pour intégrer un algorithme quelconque d'un RDN.

1. Définition :

L'implémentation : dans le contexte informatique, est définie comme l'action d'installer ou de mettre en place un système d'exploitation, un logiciel, ou une fonctionnalité sur un ordinateur, adapté aux besoins et à la configuration informatique de l'utilisateur. C'est le processus par lequel un concept ou une idée est rendu opérationnel, souvent à partir de spécifications techniques. En d'autres termes, c'est la concrétisation d'un produit logiciel qui permet de transformer des instructions ou des algorithmes en un programme exécutable et fonctionnel au sein d'un environnement matériel donné.

Ce processus peut varier en complexité, allant de la simple installation d'une application à l'intégration d'une nouvelle méthode ou système complet au sein d'une infrastructure informatique existante. L'implémentation est une étape cruciale qui assure que les solutions logicielles sont non seulement théoriquement solides mais aussi pratiquement efficaces et adaptées aux contextes d'utilisation réels.

2. Réseaux de neurones pour logiciels et microcontrôleurs

L'implémentation de réseaux de neurones sur des logiciels et des microcontrôleurs est une tâche complexe qui nécessite une planification et une exécution minutieuses. Elle commence par le choix de l'architecture du réseau, une étape cruciale où l'on décide du nombre de couches cachées et de neurones dans chaque couche, ainsi que des fonctions d'activation qui seront utilisées pour introduire des non linéarités dans le modèle. Ces décisions dépendent fortement de la complexité de la tâche à apprendre et des caractéristiques des données disponibles.

La préparation des données est une autre étape fondamentale, où les données brutes doivent être soigneusement collectées, nettoyées de toute anomalie et souvent transformées via des

processus de normalisation ou de standardisation pour assurer que le modèle ne soit pas biaisé par l'échelle des variables. Cette étape permet de s'assurer que le réseau de neurones reçoit des données de qualité pour l'apprentissage.

L'entraînement du réseau est le cœur du processus, où les poids synaptiques, qui sont les paramètres du modèle, sont ajustés de manière itérative. Cet ajustement se fait généralement à l'aide d'algorithmes d'optimisation comme la descente de gradient, qui cherche à minimiser une fonction de coût représentant l'erreur entre les prédictions du modèle et les véritables valeurs. L'apprentissage peut être supervisé, semi-supervisé ou non supervisé, selon la disponibilité des étiquettes dans les données d'entraînement.

Une fois l'entraînement terminé, il est impératif de valider et tester le modèle avec un ensemble de données inédit pour évaluer sa capacité à généraliser à de nouvelles données. Cette étape est essentielle pour détecter et prévenir le surajustement, qui se produit lorsque le modèle apprend par cœur les données d'entraînement au lieu de comprendre les tendances sous-jacentes.

Enfin, le déploiement du modèle est l'étape où le réseau de neurones entraîné et validé est intégré dans des applications logicielles ou embarqué sur des microcontrôleurs pour être utilisé dans le monde réel. Pour les logiciels, des bibliothèques comme TensorFlow ou PyTorch offrent des outils avancés qui facilitent la conception, l'entraînement et le déploiement des modèles de réseaux de neurones.

Cependant, pour les microcontrôleurs, l'implémentation doit tenir compte des ressources limitées en termes de mémoire et de puissance de calcul. Cela implique souvent une optimisation rigoureuse du modèle, qui peut inclure la réduction de la précision des poids à travers la quantification, ou l'adoption de réseaux de neurones binaires ou ternaires qui sont moins gourmands en ressources. Ces techniques permettent de déployer des modèles de machine Learning sur des dispositifs avec des contraintes matérielles sévères, ouvrant la voie à une multitude d'applications intelligentes dans des domaines tels que l'IoT, l'automatisation domestique et l'industrie 4.0.

3. Utilisation de MATLAB pour les Réseaux de Neurones : Présentation et Fonctionnalités

MATLAB est un environnement de programmation interactif et un langage de programmation développé par MathWorks, principalement utilisé pour le calcul numérique, l'analyse de données et la visualisation. Il offre une gamme complète de fonctionnalités pour

résoudre des problèmes mathématiques et techniques grâce à un langage de haut niveau et une vaste bibliothèque d'outils pour le calcul numérique, l'algèbre linéaire, l'optimisation, le traitement du signal, le traitement d'images, etc.

En plus de ses capacités de calcul, MATLAB se distingue par ses fonctionnalités avancées de visualisation, permettant la création de graphiques 2D et 3D, d'animations et de diagrammes. Son interface utilisateur interactive facilite l'interaction avec le code, l'exploration des données et le test des algorithmes en temps réel. De plus, MATLAB peut être intégré avec d'autres langages de programmation comme C/C++, Python, Java, etc., ce qui en fait un outil polyvalent pour le développement d'applications.

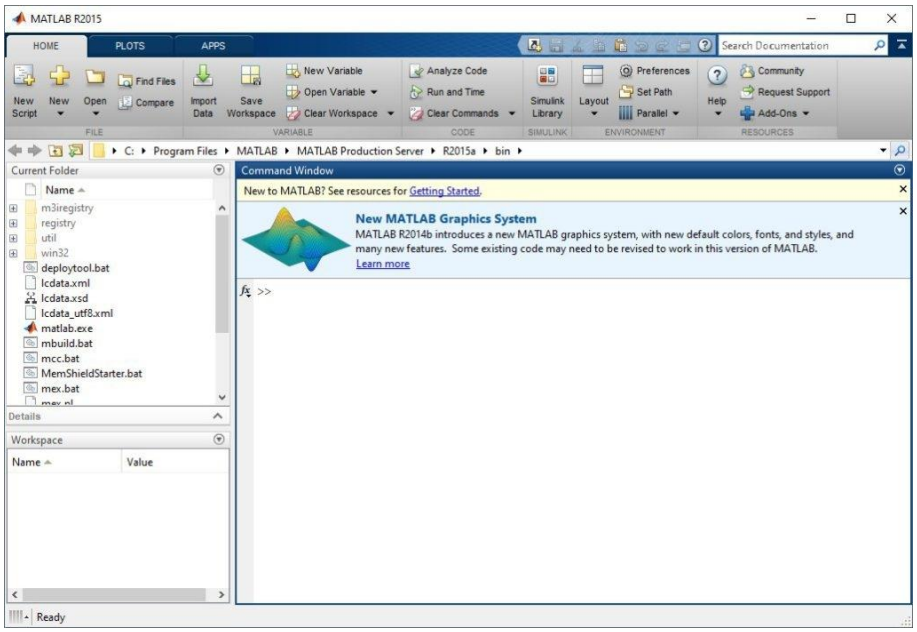


Figure 5 Accueil MATLAB R2015

Dans le domaine des réseaux de neurones, MATLAB est largement utilisé pour sa capacité à créer, entraîner et analyser différents types de réseaux neuronaux, tels que les réseaux de neurones artificiels (ANN), les réseaux de neurones récurrents (RNN), les réseaux de neurones convolutifs (CNN), et les réseaux de neurones à longue mémoire (LSTM). MATLAB offre des outils spécialisés pour la création de couches neuronales, la définition des architectures de réseau, l'optimisation des hyperparamètres, et la visualisation des performances et des résultats des réseaux neuronaux.

Nouveau projet (New) : Ce bouton permet de créer un nouvel environnement de travail dans MATLAB, ouvrant ainsi une nouvelle fenêtre de Command Windows pour démarrer un nouveau projet ou une nouvelle session de travail.

Fenêtre de commande (Command Windows) : C'est l'endroit où l'utilisateur saisit et exécute les instructions de son programme. Les instructions sont exécutées une par une dans cette zone, ce qui permet de tester et d'interagir directement avec le code.

Editeur (MATLAB Editor) : Contrairement à la Command Windows, l'Editeur permet à l'utilisateur d'écrire et de modifier ses instructions de manière plus structurée et complète. Les instructions peuvent être écrites en bloc et organisées dans des fichiers de script ou de fonction pour une exécution plus efficace.

Espace de travail (Workspace) : Cette zone affiche toutes les variables, constantes et informations sur leur format, taille et type utilisées dans la session de travail actuelle. Cela permet à l'utilisateur de surveiller et de gérer les données en temps réel pendant l'exécution du code.

Historique des commandes (Command History) : L'historique des commandes garde une trace des instructions et des actions exécutées dans les sessions de travail actuelles et précédentes. Cela permet à l'utilisateur de revoir et de réutiliser des commandes précédentes, facilitant ainsi le processus de développement et de débogage.

Neural Network Toolbox :

La Neural Network Toolbox de MATLAB est un environnement pour la conception, l'implémentation et la simulation de réseaux de neurones qui offre une interface utilisateur graphique permettant aux utilisateurs d'introduire leurs données et paramètres sans se soucier de l'implémentation sous-jacente, ce qui rend cet outil particulièrement accessible et pratique pour les débutants tout en offrant la flexibilité nécessaire aux experts; elle inclut des fonctions et des blocs de construction prédéfinis pour les tâches courantes, automatise des étapes complexes comme l'initialisation des poids et la sélection de l'algorithme d'optimisation, et permet même la génération automatique de code pour le déploiement de réseaux formés, ce qui en fait un outil précieux pour une large gamme d'applications en intelligence artificielle et en apprentissage automatique.

3.1. Implémentation d'un RdN dans MATLAB :

Dans le cadre de l'implémentation d'un réseau de neurones dans MATLAB, le processus débute par la mise en place de l'environnement de développement. La Deep Learning Toolbox est installée pour fournir les outils nécessaires à la création et à l'entraînement des réseaux de neurones. Le script commence par vérifier la disponibilité d'un modèle préentraîné, ce qui permettrait de gagner du temps en évitant de recommencer l'entraînement depuis le début. En

l'absence de modèle préentraîné, un nouveau réseau est initialisé avec une architecture feedforward comprenant une couche cachée de dix neurones.

Les données d'entrée et les cibles sont soigneusement définies pour refléter la relation que le réseau doit apprendre. Par exemple, si le but est de prédire la consommation d'énergie en fonction de divers paramètres comme la température et l'humidité, les données d'entrée (x) incluraient ces paramètres, tandis que les cibles (t) représenteraient la consommation d'énergie correspondante. Ces données sont cruciales car elles guident le réseau dans son processus d'apprentissage.

Le réseau est ensuite entraîné de manière itérative, chaque itération représentant une étape d'ajustement des poids en fonction des erreurs observées entre les sorties prédites et les cibles réelles. À chaque itération, une visualisation graphique est générée pour illustrer la performance du réseau, permettant ainsi une évaluation visuelle de sa précision et de son évolution.

Une fois l'entraînement terminé, une analyse détaillée est menée pour évaluer la performance du réseau en utilisant des métriques telles que l'erreur quadratique moyenne (MSE) ou le coefficient de détermination (R^2). Cette analyse est essentielle pour mesurer la précision des prédictions du réseau et sa capacité à généraliser à partir de nouvelles données. Si les résultats ne sont pas satisfaisants, des ajustements peuvent être apportés pour améliorer la performance, comme la modification de l'architecture du réseau ou l'ajustement des paramètres d'entraînement.

En conclusion, l'implémentation d'un réseau de neurones dans MATLAB implique une série d'étapes méthodiques, de la préparation de l'environnement à l'analyse post-entraînement. Chaque phase est cruciale pour garantir que le réseau puisse apprendre efficacement et fournir des prédictions précises sur des données inédites. La Deep Learning Toolbox offre un cadre robuste pour tester différentes architectures et optimiser les performances du modèle.

Étapes de Réalisation du Programme de Réseau de Neurones

Dans le cadre de l'implémentation d'un réseau de neurones dans MATLAB, le processus débute par la mise en place de l'environnement de développement. La Deep Learning Toolbox est installée pour fournir les outils nécessaires à la création et à l'entraînement des réseaux de neurones. Le script commence par vérifier la disponibilité d'un modèle préentraîné, ce qui permettrait de gagner du temps en évitant de recommencer l'entraînement depuis le début. En l'absence de modèle préentraîné, un nouveau réseau est initialisé avec une architecture feedforward comprenant une couche cachée de dix neurones.

Les données d'entrée et les cibles sont soigneusement définies pour refléter la relation que le réseau doit apprendre. Par exemple, si le but est de prédire la consommation d'énergie en

fonction de divers paramètres comme la température et l'humidité, les données d'entrée (x) incluraient ces paramètres, tandis que les cibles (t) représenteraient la consommation d'énergie correspondante. Ces données sont cruciales car elles guident le réseau dans son processus d'apprentissage.

Le réseau est ensuite entraîné de manière itérative, chaque itération représentant une étape d'ajustement des poids en fonction des erreurs observées entre les sorties prédites et les cibles réelles. À chaque itération, une visualisation graphique est générée pour illustrer la performance du réseau, permettant ainsi une évaluation visuelle de sa précision et de son évolution.

Une fois l'entraînement terminé, une analyse détaillée est menée pour évaluer la performance du réseau en utilisant des métriques telles que l'erreur quadratique moyenne (MSE) ou le coefficient de détermination (R^2). Cette analyse est essentielle pour mesurer la précision des prédictions du réseau et sa capacité à généraliser à partir de nouvelles données. Si les résultats ne sont pas satisfaisants, des ajustements peuvent être apportés pour améliorer la performance, comme la modification de l'architecture du réseau ou l'ajustement des paramètres d'entraînement.

En conclusion, l'implémentation d'un réseau de neurones dans MATLAB implique une série d'étapes méthodiques, de la préparation de l'environnement à l'analyse post entrainement. Chaque phase est cruciale pour garantir que le réseau puisse apprendre efficacement et fournir des prédictions précises sur des données inédites. La Deep Learning Toolbox offre un cadre robuste pour tester différentes architectures et optimiser les performances du modèle.

```
% Vérifier si un réseau pré-entraîné existe
if exist('net_pretrained.mat', 'file')
    % Charger le réseau pré-entraîné
    load('net_pretrained.mat');
else
    % Créer un nouveau réseau si aucun réseau pré-entraîné n'existe
    net = feedforwardnet(10);
end

% Définir les données d'entrée et les cibles
x = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];

% Nombre d'itérations pour l'entraînement
iterations = 20;

% Boucle pour l'entraînement itératif
for i = 1:iterations
    % Configurer le réseau avec les données d'entrée et les cibles
    net = configure(net, x, t);

    % Entraîner le réseau
    [net,tr] = train(net, x, t);

    % Simuler le réseau avec les données d'entrée
    y = net(x);

    % Afficher les résultats
    figure;
    plot(x, t, 'o', x, y, '*');
    title(['Iteration ' num2str(i)]);

    % Pause pour visualiser le graphique
    pause(1);
end
```

Figure 6 Le programme pour l'optimisation itérative d'un réseau de neurones.

Enfin, les résultats et les observations sont méticuleusement documentés. Ce rapport final inclut des graphiques, des tableaux de données et une discussion approfondie sur les performances du réseau

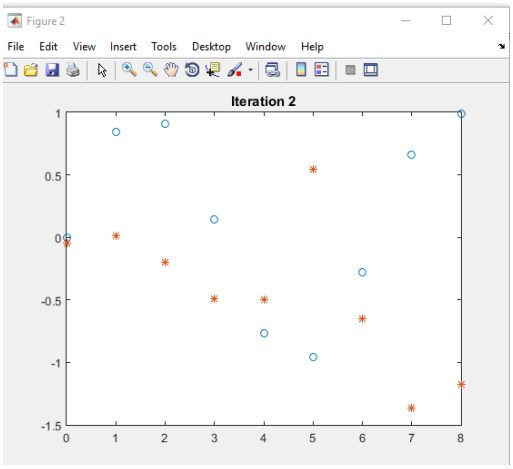


Figure 7 Résultats obtenus de la 2éme itération

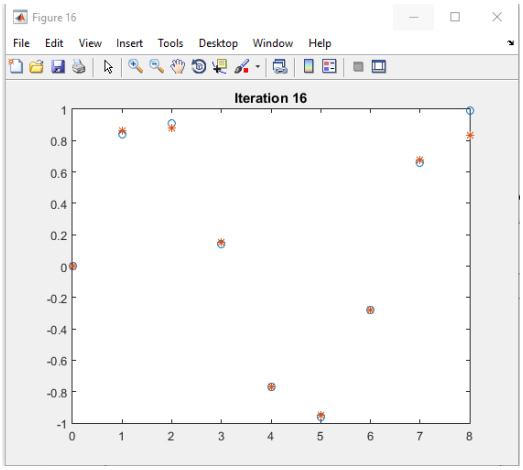


Figure 8 Résultats obtenus de la16éme itération

3.2. Les réseaux de neurone et visuel code studio :

Visuel code studio :

Visual Studio Code (VS Code) est un éditeur de code source puissant et gratuit qui prend en charge de nombreux langages de programmation et technologies, y compris Python. Avec l'extension Python de Microsoft pour VS Code, les développeurs bénéficient d'une expérience

de programmation Python simplifiée, amusante et productive, quel que soit le système d'exploitation utilisé. Voici quelques avantages de l'utilisation de Python avec VS Code :

- **IntelliSense** pour l'auto-complétion et l'aide à la saisie, facilitant ainsi la rédaction du code.
- **Linting** pour détecter les erreurs et les problèmes de style, contribuant à maintenir un code propre et conforme aux normes.
- **Débogage** pour trouver et corriger les bugs de manière efficace.
- **Tests unitaires** pour garantir la qualité et la fiabilité du code.
- **Gestion des environnements** pour passer facilement entre différents environnements Python, y compris virtuels et Conda.

Python est particulièrement bien adapté aux réseaux de neurones artificiels grâce à des bibliothèques comme TensorFlow et PyTorch, qui simplifient la création et l'entraînement de modèles. Avec VS Code, il est possible d'écrire le code pour définir, entraîner et tester un réseau de neurones, et même de visualiser les résultats directement dans l'éditeur.

En utilisant ces outils et fonctionnalités, les développeurs peuvent améliorer leur productivité et la qualité de leurs projets de machine learning. Par exemple, IntelliSense permet d'éviter les erreurs typographiques et de coder plus rapidement, tandis que le linting et le débogage aident à identifier et à résoudre les problèmes de manière proactive. La gestion des environnements permet de configurer et de tester facilement des modèles dans différents contextes, assurant ainsi une flexibilité et une adaptabilité maximales.

En résumé, Visual Studio Code, associé à l'extension Python de Microsoft, offre une plateforme robuste et versatile pour le développement de projets en Python, en particulier dans le domaine des réseaux de neurones artificiels. Les fonctionnalités avancées de VS Code

permettent aux développeurs de travailler plus efficacement et de produire des modèles de haute qualité.

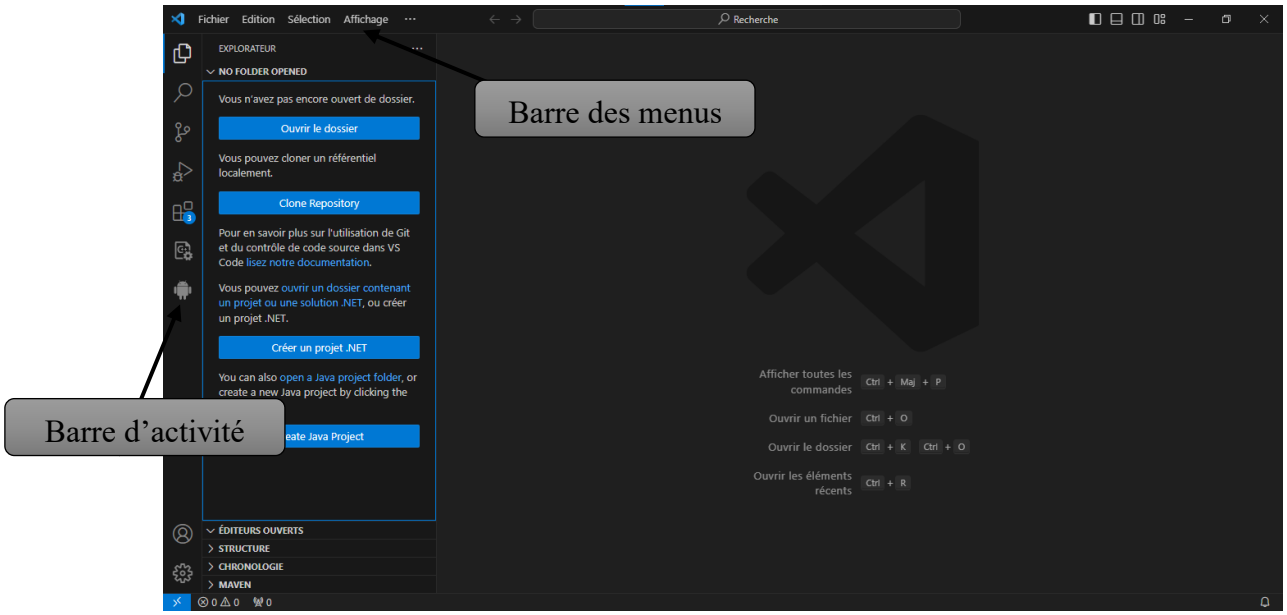


Figure 9 Accueil Visual Studio Code

Nouveau fichier / Nouveau projet : Dans Visual Studio Code, vous pouvez créer un nouveau fichier en utilisant le raccourci CTRL+N ou en sélectionnant File > New File. Pour un nouveau projet, vous pouvez simplement ouvrir un nouveau dossier qui servira d'environnement de travail en utilisant File > Open Folder.

Terminal intégré : Le terminal intégré de Visual Studio Code remplit une fonction similaire à la Command Windows de MATLAB. Vous pouvez y exécuter des commandes Shell, des scripts et interagir avec votre système. Vous pouvez l'ouvrir avec `Ctrl+`` ou via le menu View > Terminal.

Éditeur de code : L'éditeur de code de Visual Studio Code est l'endroit où vous écrivez et modifiez votre code. Il prend en charge de nombreuses langues de programmation avec la coloration syntaxique, le repliement de code, et bien plus. Vous pouvez ouvrir plusieurs fichiers en même temps dans des onglets séparés.

Explorateur : L'Explorateur de Visual Studio Code affiche la structure de fichiers de votre projet et vous permet de naviguer facilement entre eux. Cela correspond à l'Espace de travail de MATLAB, mais pour les fichiers et dossiers plutôt que pour les variables.

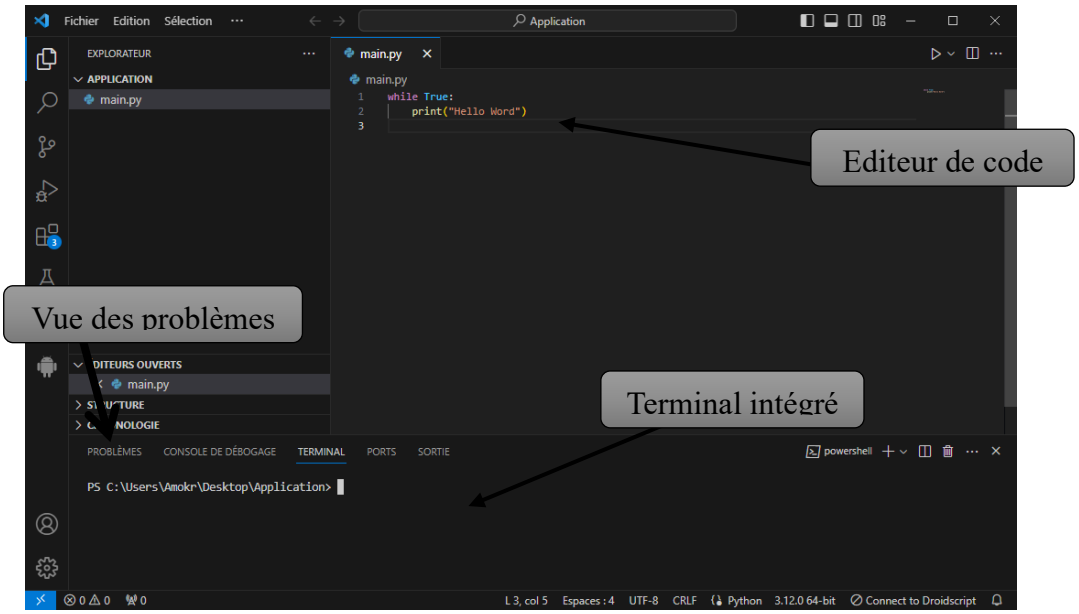


Figure 10 Espace éditeurs de VS Code

Vue des problèmes : Visual Studio Code dispose d'une vue des problèmes qui répertorie les erreurs et les avertissements dans votre code. Cela peut être considéré comme une version améliorée de l'historique des commandes de MATLAB, car elle vous aide à identifier et résoudre les problèmes dans votre code.

Historique des commandes : Bien que Visual Studio Code n'ait pas une fonctionnalité d'historique des commandes identique à celle de MATLAB, vous pouvez naviguer dans l'historique des commandes du terminal intégré en utilisant les flèches haut et bas. De plus, l'extension Command History disponible sur le Visual Studio Marketplace peut ajouter une fonctionnalité similaire pour les commandes exécutées dans le terminal1.

Implémentation d'un réseau de neurone sur visuel code :

La simulation de réseaux de neurones peut être réalisée de plusieurs façons. Pour ce projet, nous avons choisi une méthode claire et efficace en utilisant le langage de programmation Python et plusieurs bibliothèques spécialisées dans le traitement des données et l'apprentissage automatique. Le code commence par l'importation des bibliothèques nécessaires, telles que pandas pour la manipulation des données et Keras pour la construction du modèle de réseau de neurones artificiels (ANN).

Les données ont été chargées à partir du fichier diabetes.csv, qui contient des mesures cliniques pertinentes pour la prédiction du diabète. Les caractéristiques (features) ont été séparées de l'étiquette Outcome, qui indique si un patient est atteint de diabète ou non. Les données ont ensuite été divisées en ensembles d'entraînement et de test afin de permettre une évaluation rigoureuse du modèle.

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0

Figure 11 Diabète.csv

Un processus de normalisation a été appliqué aux données pour garantir que le modèle ne soit pas biaisé par l'échelle des différentes variables. La normalisation est une étape cruciale car elle assure que toutes les caractéristiques contribuent de manière équitable à l'entraînement du modèle, évitant ainsi que des variables à grande échelle dominent le processus d'apprentissage.

En résumé, la méthode choisie pour la simulation de réseaux de neurones en Python inclut les étapes suivantes :

- Importation des bibliothèques : Utilisation de pandas pour la manipulation des données et Keras pour la construction du modèle ANN.
- Chargement des données : Lecture des données à partir du fichier diabetes.csv.
- Préparation des données : Séparation des caractéristiques et de l'étiquette Outcome, et division en ensembles d'entraînement et de test.
- Normalisation des données : Application d'un processus de normalisation pour éviter les biais dus aux différentes échelles des variables.

Cette approche permet de construire un modèle ANN robuste et efficace pour la prédiction du diabète, en s'assurant que les données sont correctement préparées et que le modèle est évalué de manière rigoureuse.

Le modèle ANN a été construit avec une couche d'entrée, deux couches cachées et une couche de sortie avec une fonction d'activation sigmoïde pour la classification binaire. Le modèle a été compilé avec une fonction de perte de binary_crossentropy et un optimiseur Adam.

Le modèle a été utilisé pour prédire l'état diabétique de nouvelles données cliniques. La prédiction a été effectuée en transformant les données à l'aide du même normalisateur utilisé pour l'ensemble d'entraînement. Le résultat a été interprété comme diabétique ou non diabétique, fournissant ainsi une indication rapide de l'état de santé du patient.

```
# Charger les données
df = pd.read_csv('diabetes.csv')

# Séparer les caractéristiques et l'étiquette
X = df.drop('Outcome', axis=1).values
y = df['Outcome'].values

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normaliser les données
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Créer le modèle ANN
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compiler le modèle
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Entraîner le modèle
model.fit(X_train, y_train, epochs=1, batch_size=10)

# Évaluer le modèle
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))

# Prédire l'étiquette de diabète
predictions = (model.predict(X_test) > 0.5).astype(int)
print(predictions)
```

Lorsque nous exécutons notre modèle de prédiction du diabète dans Visual Studio Code, le terminal affiche en temps réel le déroulement de l'entraînement. À chaque époque, nous pouvons observer les métriques telles que la perte et la précision, nous permettant de suivre les progrès du modèle. Une fois l'entraînement terminé, le terminal affiche la précision finale évaluée sur l'ensemble de test, donnant une indication claire de la performance du modèle.

En outre, le terminal présente les résultats des prédictions sur de nouvelles données. Pour chaque échantillon testé, il indique si le modèle prédit l'état 'diabétique' ou 'non diabétique', ce qui est essentiel pour évaluer la capacité du modèle à généraliser à partir des données qu'il n'a jamais vues. Cette fonctionnalité est particulièrement utile pour les professionnels de la santé qui cherchent à obtenir des prédictions rapides et fiables sur l'état de santé de leurs patients.

```
# Charger les données
new_data = [[4,110,92,0,0,37.6,0.191,
new_data_scaled = scaler.transform(ne
```

Ce code charge de nouvelles données, puis applique une transformation pré-apprise (comme la normalisation ou la standardisation) pour mettre ces nouvelles données à l'échelle de la même manière que les données d'entraînement du modèle.

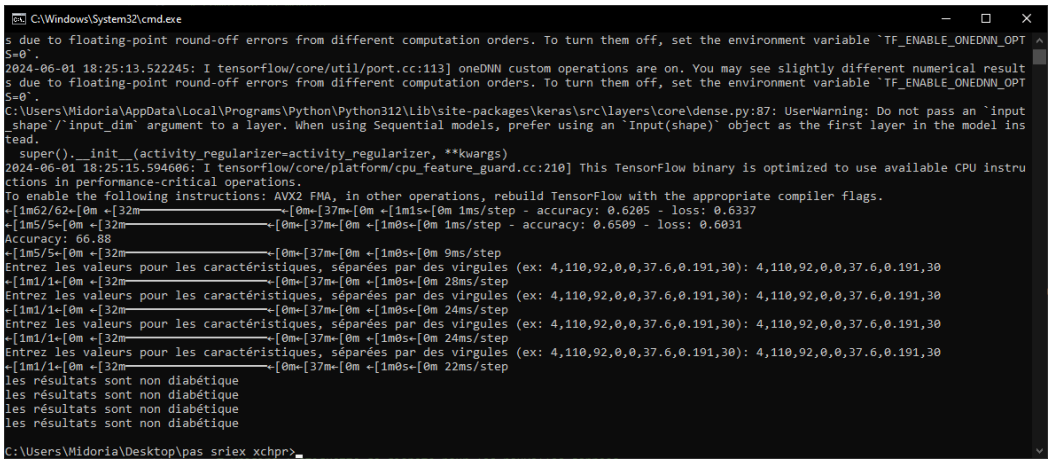
Exemple de sortie dans le terminal après l'entraînement

```
print(f"La prédiction est que cette personne est {result}.")
```

Exemple de sortie pour les prédictions

```
for i,pred in enumerate(result):
    print("le résultat est " + pred)
```

Ces lignes de code, lorsqu'elles sont ajoutées à notre script, permettent d'afficher des informations détaillées sur la performance et les prédictions du modèle, rendant les résultats immédiatement accessibles et compréhensibles pour l'utilisateur.



4. Les Réseaux de neurones et les microcontrôleurs :

4.1. Introduction aux microcontrôleurs :

Les microcontrôleurs sont des composants essentiels dans de nombreux systèmes embarqués. Ils intègrent dans un seul boîtier un microprocesseur, divers types de mémoires (RAM, ROM, Flash), et des périphériques de communication (Entrées-Sorties). Cette intégration compacte en fait des solutions idéales pour une variété d'applications, des appareils électroniques grand public aux systèmes industriels complexes.

Définition du microcontrôleur :

Un microcontrôleur est un circuit intégré qui combine les fonctionnalités d'un microprocesseur avec celles de la mémoire et des interfaces de communication. Il est conçu pour exécuter des tâches spécifiques dans des systèmes embarqués, offrant une solution complète dans un seul composant.

Architecture des microcontrôleurs :

Les microprocesseurs nécessitent plusieurs éléments pour leur fonctionnement :

- La mémoire ROM pour stocker le programme ;
- La mémoire RAM pour les données et les variables ;
- Des périphériques pour l'interaction avec les entrées et sorties ;
- Une horloge pour synchroniser les opérations.

Ces composants sont connectés par des bus de données, d'adresse et de contrôle, permettant la communication entre eux et avec le monde extérieur.

Fonctionnement général du microcontrôleur :

Lors du démarrage du système, le microprocesseur charge la première instruction à exécuter depuis la mémoire ROM. Il exécute cette instruction, puis passe à la suivante en consultant le registre d'instructions. Ce processus se poursuit séquentiellement, permettant l'exécution des programmes stockés en mémoire.

4.2. Introduction à l'Arduino ESP32 :

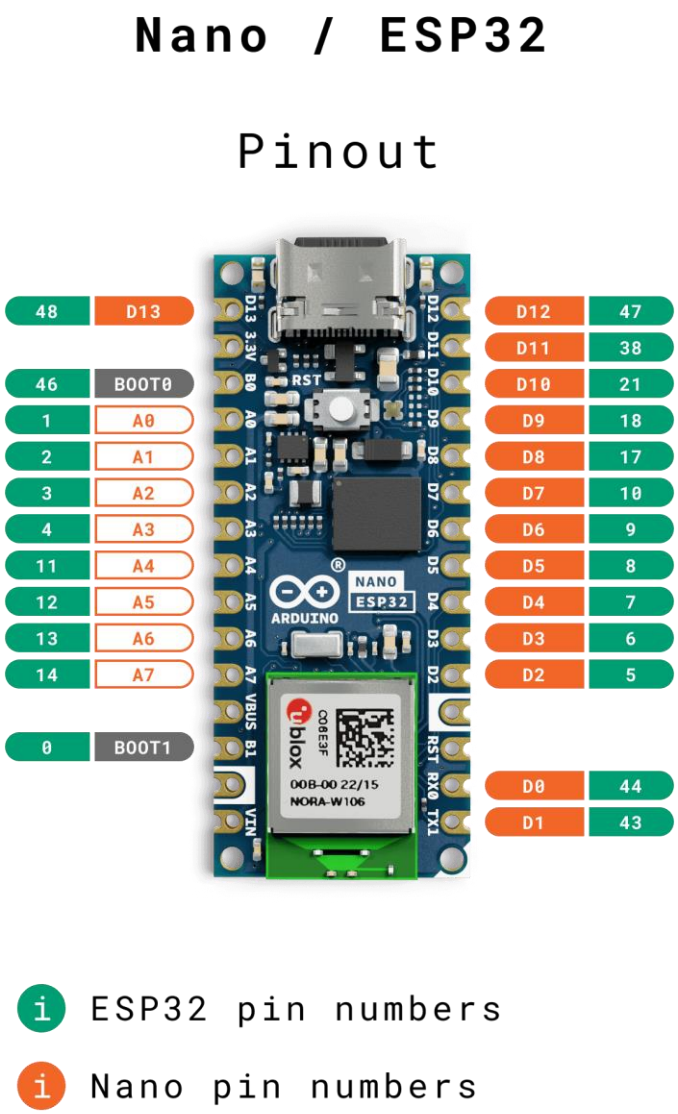


Figure 12 Pinout d'un Arduino Nano/ESP32

L'Arduino ESP32 est une plateforme de développement basée sur un microcontrôleur de l'architecture Xtensa LX6. Elle offre une multitude de fonctionnalités, notamment des broches numériques et analogiques, une connectivité Wifi et Bluetooth, ainsi que des capacités de traitement élevées.

Caractéristiques de l'Arduino ESP32 :

La carte Arduino ESP32 dispose de plusieurs broches numériques et analogiques pour l'entrée et la sortie de signaux. Elle est équipée d'un quartz de 40 MHz pour la synchronisation des opérations, et peut être alimentée via un port USB ou une source externe.

Variétés de cartes Arduino ESP32 :

Différentes variantes de cartes Arduino ESP32 sont disponibles, chacune offrant des fonctionnalités spécifiques. Parmi les plus courantes, on trouve la ESP32, la ESP32CAM et la ESP32S2, chacune ayant ses propres spécifications en termes de mémoire, de connectivité et de broches d'E/S.

Programmation de l'Arduino ESP32 :

La programmation de l'Arduino ESP32 s'effectue à l'aide du logiciel Arduino IDE, utilisant un langage basé sur C et C++. Ce langage simplifié facilite le développement d'applications pour la carte Arduino ESP32, avec de nombreuses ressources et bibliothèques disponibles en ligne.

4.3. Implémentation des Réseaux de Neurones dans l'Arduino ESP32 :

Les microcontrôleurs, tels que l'Arduino ESP32, offrent la possibilité d'implémenter des réseaux de neurones pour des applications embarquées. Cette section explore les différentes méthodes d'implémentation et leurs considérations.

Calcul sériel avec l'Arduino ESP32 :

Le calcul sériel consiste à multiplier les entrées par les poids pour chaque neurone, puis à sommer ces produits pour obtenir la sortie du neurone. Cette opération se répète pour chaque neurone du réseau, avec les résultats stockés en mémoire SRAM de la carte Arduino ESP32.

Calcul parallèle avec l'Arduino ESP32 :

Le calcul parallèle implique l'utilisation de threads pour exécuter plusieurs tâches en parallèle sur l'Arduino ESP32. Cette approche peut améliorer les performances en répartissant la charge de calcul sur plusieurs cœurs du microcontrôleur, mais nécessite une gestion complexe des ressources et des synchronisations.

Conclusion :

Nous avons évalué trois méthodes pour implémenter un réseau neuronal : MATLAB, une carte ESP32, et Python dans Visual Studio Code. MATLAB offre des performances optimales grâce à la puissance de calcul de l'ordinateur, idéal pour les projets nécessitant une grande capacité de traitement. Python dans Visual Studio Code se distingue par sa flexibilité et son accessibilité, facilitant le développement et le débogage. Les cartes ESP32 sont adaptées pour les systèmes embarqués, offrant simplicité de programmation, extensibilité matérielle, et adaptabilité aux contraintes matérielles..

Chapitre 03 : Déploiement d'un Réseau de Neurones

Chapitre 03 : Déploiement d'un Réseau de neurones

Introduction :

“Dans cette deuxième et dernière partie, nous allons déployer un réseau de neurones artificiels (ANN) sur notre carte ESP32 pour prédire le diabète, une maladie chronique. Nous exploiterons les méthodes d’apprentissage automatique, en particulier le Deep Learning, une branche avancée qui utilise des structures complexes de neurones, pour améliorer la précision et l’efficacité de notre modèle.”

1. Prédiction du diabète à l'aide d'un réseau de neurone ANN :

1.1. Description de l’application :

Notre application, conçue pour être embarquée sur une carte ESP32, utilise un réseau de neurones artificiels pour prédire le risque de diabète. Elle analyse huit paramètres de santé essentiels : le sexe, l’âge, la présence d’hypertension, d’une maladie cardiaque, l’historique de tabagisme, l’IMC (Indice de Masse Corporelle), le niveau d’HbA1c, et le taux de glucose dans le sang. En traitant ces données, l’application peut déterminer avec précision si un individu est diabétique, à risque, ou en bonne santé.

1.2. Le diabète :

Le diagnostic du diabète fait référence à un groupe de maladies qui affectent la manière dont le corps utilise le glucose sanguin. Le glucose est une source d’énergie importante pour les cellules qui composent les muscles et les tissus, ainsi que pour le cerveau. La cause principale du diabète varie selon le type, mais quelle que soit la forme de diabète, elle peut entraîner un excès de sucre dans le sang, ce qui peut conduire à de graves problèmes de santé1.

- Parmi les facteurs favorisant l’apparition ou l’aggravation du diabète, on trouve :
- Génétique et antécédents familiaux de diabète
- Surpoids, obésité et manque d’activité physique
- Hypertension artérielle et taux élevé de cholestérol
- Grossesse, pouvant mener à un diabète gestationnel
- Âge avancé, car le risque augmente avec l’âge
- Les symptômes du diabète peuvent inclure :
- Soif intense et bouche sèche (polydipsie)
- Mictions fréquentes
- Fatigue
- Vision trouble
- Perte de poids inexpliquée
- Engourdissement ou picotements dans les mains ou les pieds

Si le diabète n'est pas contrôlé, il peut s'aggraver et entraîner des symptômes supplémentaires, tels que :

- Nausées
- Fièvre (bien que moins fréquente que dans les infections)

Il est crucial de consulter un médecin si des symptômes de diabète apparaissent, car un traitement précoce peut prévenir les complications graves

1.3. L'Apprentissage Automatique :

L'apprentissage automatique, ou Machine Learning, est un pilier central de l'intelligence artificielle (IA). Il s'agit d'un domaine scientifique qui permet aux machines d'apprendre à partir de données et d'améliorer leurs performances dans la réalisation de tâches spécifiques sans être explicitement programmées pour cela.

Voici quelques points clés sur l'apprentissage automatique :

Définition et Fonctionnement

L'apprentissage automatique utilise des modèles mathématiques pour permettre aux ordinateurs d'identifier des motifs dans les données, appelés "patterns", et de faire des prédictions ou de prendre des décisions en conséquence. Ces modèles sont entraînés en utilisant un ensemble de données et s'améliorent progressivement en performance.

Rôle dans l'IA

L'apprentissage automatique est intégré à l'IA et simule la pensée humaine via des réseaux neuronaux. Il est particulièrement efficace dans des contextes où les données changent constamment ou où la codification directe d'une solution est difficile1.

Avantages

Les avantages de l'apprentissage automatique incluent l'automatisation du travail, l'amélioration de l'expérience utilisateur grâce à des ChatBots, et l'identification de modèles dans les données, ce qui est essentiel pour le développement de systèmes intelligents1.

TensorFlow : Un Outil Clé pour l'Apprentissage Automatique

TensorFlow, développé par Google, est une bibliothèque d'apprentissage automatique qui a transformé la manière dont les modèles de Machine Learning, y compris ceux du Deep Learning, sont conçus et déployés. Voici comment TensorFlow s'intègre dans l'apprentissage automatique :

Création de Modèles

TensorFlow permet de créer des modèles d'apprentissage automatique en utilisant un graphe de calcul, qui contient des nœuds et des arêtes pour un calcul parallèle et distribué. Cela accélère le processus de formation des réseaux neuronaux.

Flexibilité et Évolutivité

Il offre une flexibilité pour définir des graphiques de calcul personnalisés et peut être exécuté sur une grande variété de plateformes, ce qui le rend idéal pour l'entraînement de modèles complexes.

1.4. Deep Learning (apprentissage profond) :

Le Deep Learning est une sous-catégorie de l'apprentissage automatique qui s'inspire du fonctionnement des réseaux neuronaux biologiques. Il repose sur des algorithmes capables de reconnaître des patterns complexes à partir de données brutes

2. Architecture des Réseaux de Neurones

2.1. Couches Cachées :

- Les réseaux de neurones profonds se distinguent par leur nombre élevé de couches cachées. Ces couches permettent de modéliser des fonctions très complexes en apprenant des représentations hiérarchiques des données.
- Chaque couche cachée effectue une transformation non linéaire des entrées. La sortie d'une couche devient l'entrée de la couche suivante.
- Mathématiquement, la sortie d'une couche cachée $h_i$est calculée comme suit :

$$\boldsymbol{h_i = \sigma(W_i H_{i-1} + b_i)}$$

Où $\boldsymbol{W_i}$ sont les poids, $\boldsymbol{h_{i-1}}$est la sortie de la couche précédente, et $\boldsymbol{b_i}$ est le biais. La fonction d'activation σ introduit la nonlinéarité.

2.2. Neurones Artificiels :

- Chaque neurone reçoit des entrées pondérées et applique une fonction d'activation.
- La fonction d'activation ReLU $\boldsymbol{f(x) = \max(0, x)}$ est couramment utilisée pour introduire la non linéarité.
- Un neurone peut être représenté comme :

$$\boldsymbol{y = f(W_x + b)}$$

Où $\boldsymbol{W}$ est la matrice de poids, $\boldsymbol{x}$ est le vecteur d'entrée, et $\boldsymbol{b}$ est le biais.

2.3. Rétropropagation :

- La rétropropagation est la méthode d'entraînement des réseaux de neurones.
- Elle ajuste les poids en utilisant le gradient de la fonction de perte par rapport aux poids.
- La mise à jour des poids $\boldsymbol{W}$ se fait selon la règle :

$$\Delta \boldsymbol{W} = -\boldsymbol{\eta} \frac{\boldsymbol{\vartheta L}}{\boldsymbol{\vartheta W}}$$

Où $\boldsymbol{\eta}$ est le taux d'apprentissage et $\boldsymbol{L}$ est la fonction de perte.

2.4. Transfer Learning :

- Le Transfer Learning consiste à utiliser un modèle préentraîné sur une tâche similaire comme point de départ pour une nouvelle tâche.
- Cela permet d'économiser du temps et des ressources d'entraînement.

Applications et Cas d'Usage

- Détection de Fraudes :
- Les réseaux de neurones peuvent détecter des schémas frauduleux dans les transactions financières.
- Recommandations Personnalisées :
- Les plateformes de streaming utilisent le Deep Learning pour recommander des films, des séries, etc., en fonction des préférences de l'utilisateur.
- Prédiction de Maladies :
- Dans le domaine médical, le Deep Learning aide à prédire des maladies à partir de données cliniques et d'images médicales.

3. Implémentation de la descente de gradient : Méthodes Online et Batch

3.1. Descente de Gradient Online (Stochastic Gradient Descent - SGD)

Dans la descente de gradient online, également connue sous le nom de descente de gradient stochastique, les poids du réseau de neurones sont mis à jour après chaque exemple de formation. Voici comment cela fonctionne :

- Mise à jour fréquente : Après chaque exemple de formation, les poids sont ajustés en fonction de l'erreur pour cet exemple particulier.

- Convergence plus rapide : Puisque les poids sont mis à jour fréquemment, la descente de gradient stochastique peut souvent trouver des solutions raisonnablement bonnes beaucoup plus rapidement que la méthode par lots.
- Oscillations : En raison des mises à jour fréquentes et des variations des exemples de formation, la descente de gradient stochastique peut parfois osciller autour du minimum global plutôt que de converger directement.

3.2. Descente de Gradient par Lots (Batch Gradient Descent)

Dans la descente de gradient par lots, les poids du réseau de neurones sont mis à jour après avoir calculé l'erreur pour l'ensemble du lot de formation. Voici comment cela fonctionne :

- Mise à jour globale : Les poids sont ajustés en fonction de l'erreur moyenne sur l'ensemble du lot de formation.
- Convergence stable : Étant donné que les poids sont mis à jour moins fréquemment et que les mises à jour sont basées sur des moyennes, cette méthode tend à converger plus doucement et plus directement vers le minimum global.
- Mémoire et calcul intensifs : La descente de gradient par lots nécessite de stocker et de traiter l'ensemble du lot de formation en mémoire, ce qui peut être coûteux en termes de calcul et de mémoire, surtout pour les grands ensembles de données.

4. Choix de la méthode Batch pour le Développement du Réseau de Neurones ANN

J'ai choisi d'utiliser la descente de gradient par lots pour le développement de mon réseau de neurones ANN pour plusieurs raisons :

- Convergence plus stable : La méthode par lots offre une convergence plus stable et plus lisse vers le minimum global, réduisant les risques d'oscillation autour du minimum.
- Efficacité pour des données en grande quantité : Bien que la descente de gradient par lots soit plus intensive en termes de mémoire et de calcul, elle est plus appropriée pour les ensembles de données de grande taille, car elle permet de calculer des gradients plus précis.
- Optimisation globale : En utilisant l'erreur moyenne de l'ensemble du lot, la descente de gradient par lots tend à mieux optimiser les poids pour des performances globales du modèle, plutôt que d'être influencée par les fluctuations des exemples individuels.

La descente de gradient par lots, bien que plus exigeante en ressources, offre une optimisation plus stable et précise pour les réseaux de neurones ANN, ce qui en fait le choix idéal pour le développement de modèles complexes et de grande envergure.

5. Développement d'un Système de Détection de Diabète Intégré

5.1. Recherche d'une base de données sur Kaggle

Pour démarrer ce projet, nous avons entrepris de trouver une base de données pertinente sur Kaggle. Voici les étapes réalisées :

- Inscription sur Kaggle : La première étape a consisté à créer un compte utilisateur sur Kaggle, une plateforme réputée pour la disponibilité de datasets variés et de qualité.
- Téléchargement d'un dataset concernant le diabète : Après l'inscription, nous avons recherché et téléchargé un dataset spécifique sur le diabète. Ce dataset sera utilisé pour entraîner notre modèle de réseau de neurones.

Genre	Age	Hypertension	Maladie Cardiaque	Antécédents Tabagiques	Indice de Masse Corporelle	Niveau HbA1c	Niveau de Glucose	Diabétique
Female	36.0	0	0	Current	23.45	5.0	155	0
Male	76.0	1	1	Current	20.14	4.8	155	0
Female	20.0	0	0	Never	27.32	6.6	85	0
Female	44.0	0	0	Never	19.31	6.5	200	1
Female	69.0	0	0	Never	21.24	4.8	85	0
Female	72.0	0	1	Former	27.94	6.5	130	0
Female	4.0	0	0	No Info	13.99	4.0	140	0
Male	30.0	0	0	Never	33.76	6.1	126	0
Male	67.0	0	1	Not Current	27.32	6.5	200	1
Male	40.0	0	0	Former	27.85	5.8	80	0
Male	45.0	1	0	Never	26.47	4.0	158	0
Female	20.0	0	0	Never	22.19	3.5	100	0
Female	76.0	0	0	Never	23.55	5.0	85	0
Male	5.0	0	0	No Info	15.1	5.8	85	0
Female	15.0	0	0	No Info	21.76	4.5	130	0
Female	26.0	0	0	Never	21.22	6.6	200	0
Male	50.0	1	0	Current	27.32	5.7	260	1
Female	34.0	0	0	Never	56.43	6.2	200	0
Male	73.0	0	0	Former	25.91	9.0	160	1
Male	5.0	0	0	No Info	27.32	6.6	130	0
Female	77.0	1	1	Never	32.02	5.0	159	0

Figure 13 Base de données utilisée pour l'entrainement

5.2. Entraînement du réseau de neurones avec Python

Dropout :

- Le dropout est un moyen efficace de réduire le surapprentissage en empêchant les neurones de devenir trop spécialisés pour certaines caractéristiques.

- Pendant l'entraînement, chaque neurone a une probabilité de 50 % d'être désactivé (c'est-à-dire qu'il ne contribue pas à la propagation avant ou à la rétropropagation du gradient).
- Cela permet au modèle d'apprendre des représentations plus robustes et généralisables.

L'étape suivante du projet a consisté à entraîner un modèle de réseau de neurones en utilisant Python. Les étapes détaillées sont les suivantes :

Préparation des données :

La préparation des données est une étape cruciale pour garantir la qualité du modèle. Nous avons utilisé les bibliothèques `pandas` et `SKLearn` pour cette tâche. Les opérations spécifiques incluent :

- Chargement des données : Le dataset a été chargé dans un DataFrame Pandas pour une manipulation facile.
- Normalisation des données : Les caractéristiques ont été normalisées pour garantir que chaque caractéristique contribue également au modèle en utilisant la class LabelEncoder() de SkLearn.
- Division des données : Les données ont été divisées en ensembles d'entraînement et de test pour permettre une évaluation objective de la performance du modèle.

```
# Chargement des données
data = pd.read_csv('diabette.csv')

# Normalisation des données
le_gender = LabelEncoder()
le_smoking = LabelEncoder()
data['gender'] = le_gender.fit_transform(data['gender'])
data['smoking_history'] = le_smoking.fit_transform(data['smoking_history'])

# Séparation des caractéristiques et des étiquettes
X = data.drop('diabetes', axis=1)
y = data['diabetes']

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Création et entraînement du modèle :

Une fois les données prêtes, nous avons utilisé `TensorFlow` et `Keras` pour créer et entraîner le modèle de réseau de neurones. Les étapes spécifiques incluent :

- Définition de l'architecture du modèle : Nous avons conçu un modèle séquentiel avec plusieurs couches denses. Chaque couche utilise la fonction d'activation ReLU, sauf pour la couche de sortie qui utilise la fonction sigmoïde pour la classification binaire.

- Compilation du modèle : Le modèle a été compilé en utilisant l'optimiseur Adam et la fonction de perte binaire crossentropy, avec l'accuracy (précision) comme métrique d'évaluation.

Implémentation d'un arrêt précoce : L'entrainement du modèle s'arrêtera si la précision de l'entrainement n'évolue plus.

- Entraînement du modèle : Le modèle a été entraîné sur l'ensemble de données d'entraînement en utilisant la validation croisée pour évaluer ses performances. L'entraînement a été effectué sur plusieurs époques avec un batch size définie.

```
# Initialisation du modèle
model = Sequential()

# Ajout des couches cachées avec dropout
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.5))

# Ajout de la couche de sortie
model.add(Dense(1, activation='sigmoid'))

# Compilation du modèle
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fonction de rappel pour l'arrêt précoce
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5,
restore_best_weights=True)

# Entraînement du modèle avec la validation
model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2,
callbacks=[early_stopping])
```

La "loss" (ou perte en français) est une mesure quantitative de l'erreur d'un modèle d'apprentissage automatique. Elle indique à quel point les prédictions du modèle sont éloignées des valeurs réelles (c'est-à-dire des valeurs attendues ou des cibles). Plus précisément, la fonction de perte calcule une valeur qui quantifie cette différence pour chaque échantillon et ensuite agrège ces valeurs sur l'ensemble des données.

Entropie Croisée (Cross-Entropy) :

- Utilisée principalement pour les tâches de classification.
- Elle mesure la différence entre deux distributions de probabilité, souvent entre les distributions de probabilité prédite par le modèle et les vraies étiquettes.

$$\text{CrossEntropy} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\left(\hat{y}_{i,c}\right)$$

Où $\boldsymbol{y_{i,c}}$ est la vraie étiquette (0 ou 1) pour la classe $\boldsymbol{c}$ de l'échantillon $\boldsymbol{i}$ et $\boldsymbol{\hat{y}_{i,c}}$ est la probabilité prédite pour la classe $\boldsymbol{c}$.

Rôle de la Fonction de Perte

La fonction de perte joue un rôle crucial dans l'entraînement des modèles de machine learning. Voici comment elle est utilisée :

1. **Optimisation :**

L'objectif de l'entraînement est de minimiser la perte en ajustant les poids du modèle. Les algorithmes d'optimisation, comme la descente de gradient, utilisent les gradients de la fonction de perte pour mettre à jour les poids du modèle.

2. **Évaluation :**

La perte permet d'évaluer la performance du modèle pendant et après l'entraînement. Des valeurs de perte faibles indiquent généralement de bonnes performances, tandis que des valeurs élevées indiquent que le modèle a du mal à faire des prédictions précises.

5.3. Conversion du modèle vers TensorFlow Lite et version binaire

Pour permettre l'exécution du modèle sur un microcontrôleur Arduino ESP32, nous avons procédé à la conversion du modèle en un format plus léger et compatible :

- Conversion en TensorFlow Lite :

Le modèle Keras entraîné a été converti en TensorFlow Lite en utilisant `TFLiteConverter`. Ce format est optimisé pour les dispositifs à ressources limitées.

- Génération d'un fichier binaire :

Nous avons ensuite généré un fichier binaire du modèle TensorFlow Lite en utilisant l'outil `xxd`. Ce fichier binaire sera intégré dans le programme Arduino.

```
# Convert the model to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TensorFlow Lite model to a file
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)

import subprocess

# Convert the TensorFlow Lite model to a C header file
subprocess.run(["xxd", "-i", "model.tflite", "model_data.h"])
```

5.4. Résultat de l'entraînement

Les résultats de l'entraînement montrent une progression claire et efficace du modèle sur les 17 époques présentées. Voici quelques observations détaillées :

Précision (Accuracy) :

- La précision sur les données d'entraînement a commencé à 0.9027 à la première époque et a progressivement augmenté pour atteindre environ 0.9688 à la fin de l'entraînement.
- La précision sur les données de validation a également augmenté, débutant à 0.9588 et se stabilisant autour de 0.9727, avec une légère fluctuation.

Perte (Loss) :

- La perte sur les données d'entraînement a commencé à 0.2700 et a diminué régulièrement, atteignant environ 0.0916.
- La perte sur les données de validation a montré une tendance similaire, passant de 0.1141 à environ 0.0796.

Stabilisation :

- On observe une stabilisation des valeurs de précision et de perte à partir de l'époque 8, où les améliorations deviennent plus marginales. Ceci indique que le modèle commence à converger.

Évaluation finale :

- Lors de l'évaluation finale sur les données de test, le modèle a obtenu une précision de 0.9734 et une perte de 0.0837. Ces valeurs sont cohérentes avec celles observées pendant les époques de validation, ce qui montre que le modèle n'est ni sous-appris ni sur-appris.

Les résultats montrent que le modèle s'entraîne efficacement et atteint un haut niveau de performance, avec une bonne généralisation sur les données de validation et de test. L'entraînement aurait peut-être pu s'arrêter autour de l'époque 10 à 12, car les améliorations supplémentaires sont minimes au-delà de ce point. Cependant, le modèle est stable et performant, indiquant une bonne configuration des hyperparamètres et une architecture appropriée pour la tâche.

```
Epoch 14/30
1960/1960 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9693 - loss: 0.0907 - val_accuracy: 0.9719 - val_loss: 0.0810
Epoch 15/30
1960/1960 ━━━━━━━━━━━━━━━━━━━━ 14s 6ms/step - accuracy: 0.9689 - loss: 0.0914 - val_accuracy: 0.9726 - val_loss: 0.0807
Epoch 16/30
1960/1960 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9683 - loss: 0.0926 - val_accuracy: 0.9727 - val_loss: 0.0797
Epoch 17/30
1960/1960 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9689 - loss: 0.0922 - val_accuracy: 0.9727 - val_loss: 0.0788
613/613 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9734 - loss: 0.0824
Loss: 0.08247107267379761
Accuracy: 0.9727550745010376
```

Figure 14 Résultats de l'entraînement du modèle

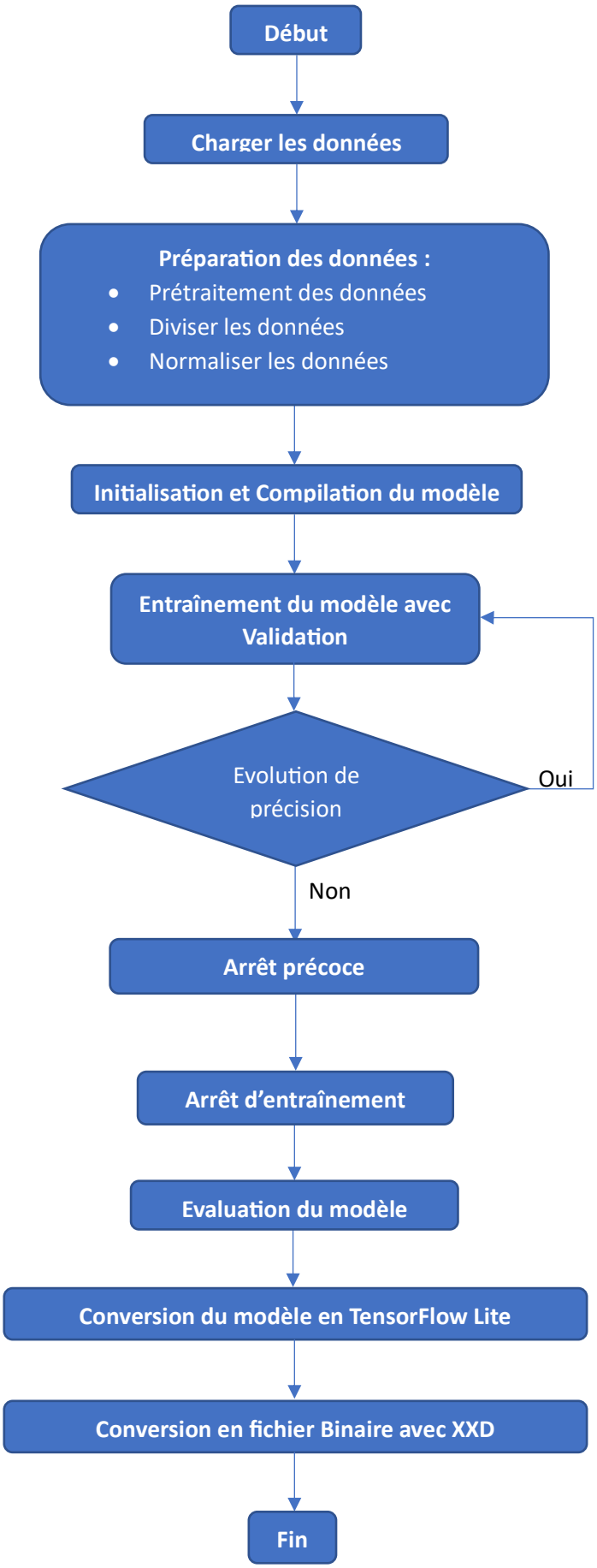


Figure 15 Organigramme des étapes utilisé lors de l'entraînement du modèle

5.5. Implémentation d'un programme Arduino pour utiliser le modèle

L'objectif de cette étape est d'intégrer le modèle de détection de diabète converti en TensorFlow Lite dans un microcontrôleur Arduino ESP32, afin qu'il puisse exécuter des prédictions en temps réel.

Préparation de l'environnement Arduino

Avant d'écrire le programme, Nous avons installé les outils et bibliothèques nécessaires :

- Arduino IDE : On télécharge et installe l'IDE Arduino depuis (https://www.arduino.cc/en/software).
- Carte ESP32 : Nous avons ajouté la prise en charge de l'ESP32 dans l'IDE Arduino en suivant les instructions sur le site officiel d'ESP32.
- Bibliothèques TensorFlow Lite pour Microcontrôleurs.

Chargement du modèle TensorFlow Lite sur l'ESP32

Le modèle TensorFlow Lite doit être converti en un format compatible avec le code Arduino. Nous avons utilisé l'outil `xxd` pour générer un fichier `.h` contenant le modèle binaire. Puis inclus ce fichier dans votre projet Arduino.

```
unsigned char diabetes_model[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
  0x88, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00, 0x00, 0x28, 0x34, 0x00, 0x00,
  0x38, 0x34, 0x00, 0x00, 0x64, 0x3b, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x86, 0xcb, 0xff, 0xff,
  0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x38, 0x00, 0x00, 0x00,
  0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f,
  0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x94, 0xff, 0xff, 0xff, 0x0d, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0x64, 0x65, 0x6e, 0x73,
  0x65, 0x5f, 0x37, 0x00, 0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
  0x5a, 0xcc, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x0d, 0x00, 0x00, 0x00,
  0x64, 0x65, 0x6e, 0x73, 0x65, 0x5f, 0x34, 0x5f, 0x69, 0x6e, 0x70, 0x75,
  0x74, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x34, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0xdc, 0xff, 0xff, 0xff, 0x10, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x43, 0x4f, 0x4e, 0x56,
  0x45, 0x52, 0x53, 0x49, 0x4f, 0x4e, 0x5f, 0x4d, 0x45, 0x54, 0x41, 0x44,
  0x41, 0x54, 0x41, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00,
  0x08, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
  0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74,
  0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00,
  0x11, 0x00, 0x00, 0x00, 0x44, 0x33, 0x00, 0x00, 0x3c, 0x33, 0x00, 0x00,
```

Figure 16 Modèle TFLite en Binaire

Initialisation du modèle et gestion des prédictions

Nous avons créé un programme Arduino qui charge et initialise le modèle TensorFlow Lite, puis effectue des prédictions basées sur les données d'entrée. Voici un exemple de programme Arduino détaillé :

Inclusions et initialisations :

- Les bibliothèques nécessaires pour exécuter TensorFlow Lite sur l'ESP32 sont incluses.
- La taille de l'arène de mémoire est définie pour stocker les tenseurs.

Fonction setup() :

- Initialisation de la communication série pour l'affichage des résultats.
- Initialisation du rapporteur d'erreurs pour gérer les erreurs éventuelles.
- Configuration de l'interpréteur TensorFlow Lite avec le modèle et les opérateurs nécessaires.
- Allocation des tenseurs et récupération des pointeurs d'entrée et de sortie.

Fonction loop() :

- Exemple de données d'entrée pour effectuer une prédiction.
- Remplissage du tenseur d'entrée avec les données d'exemple.
- Invocation du modèle pour effectuer une prédiction.
- Lecture et affichage des résultats de la prédiction sur le moniteur série.

Communication sans fil

Nous avons ajouté une partie au code pour permettre à l'Arduino de recevoir des requêtes HTTP de type POST sur le réseau Wifi avec les données de prédictions, et répondre avec la prédiction faite suivant le modèle.

5.6. Création d'une application Android avec DroidScript

Enfin, pour permettre la communication entre l'ESP32 et un appareil mobile, nous avons développé une application Android en utilisant DroidScript :

Installation de DroidScript :

DroidScript a été téléchargé et installé sur un appareil Android depuis le Google Play Store.

Configuration du projet DroidScript :

- Création d'un nouveau projet :

Nous avons ouvert DroidScript et créé un nouveau projet.

- Configuration des permissions :

Les permissions nécessaires ont été configurées dans les paramètres du projet pour permettre la communication Wifi entre l'application Android et l'ESP32.

Développement de l'interface et de la logique de communication :

Nous avons conçu une interface utilisateur simple où on peut choisir les différents paramètres à envoyer. Le code a été écrit pour envoyer des requêtes vers le serveur précédemment créé par l'Arduino contenant les données choisis par l'utilisateur et recevoir le résultat de la prédiction.

5.7. Intégration d'une Interface Mobile pour la Prédiction du Diabète

Dans cette partie, nous abordons le développement d'une interface mobile conviviale, conçue pour collecter des données cliniques et prédire le risque de diabète.

L'application, développée indépendamment, permet une interaction directe avec un microcontrôleur ESP32 via une connexion Wi-Fi, facilitant ainsi la saisie et l'analyse de données cliniques pertinentes.

Nous décrirons les huit paramètres essentiels qui constituent la base de notre modèle prédictif, reflétant fidèlement les caractéristiques du dataset utilisé pour l'entraînement de notre réseau de neurones artificiels.

Après avoir rempli les huit paramètres nécessaires, l'utilisateur appuie sur un bouton 'Envoyer' pour soumettre ses informations. Le système analyse ensuite ces données en temps réel et fournit un résultat sous forme de message, indiquant si les caractéristiques saisies suggèrent un risque de diabète. Cette fonctionnalité clé assure une interaction fluide et une réponse immédiate, rendant l'application à la fois efficace et accessible pour les utilisateurs.

Ces paramètres sont :

- Genre (Gender) : Masculin ou Féminin
- Âge (Age)
- Hypertension (Hypertension) : Présence (1) ou absence (0)
- Maladie cardiaque (Heart Disease) : Présence (1) ou absence (0)
- Historique de tabagisme (Smoking History) : Actuel, Ancien, Jamais ou Pas d'info
- Indice de masse corporelle (BMI)
- Niveau d'HbA1c (HbA1c Level)
- Niveau de glucose sanguin (Blood Glucose Level)

Les utilisateurs remplissent ces champs avec des données telles que :

Femme ; **54 ans** ; hypertension : **Non** ; maladie cardiaque : **Non** ; **pas d'info sur le tabagisme** ; IMC : **27.32** ; niveau d'HbA1c : **6.6** ; niveau de glucose sanguin : **80**.

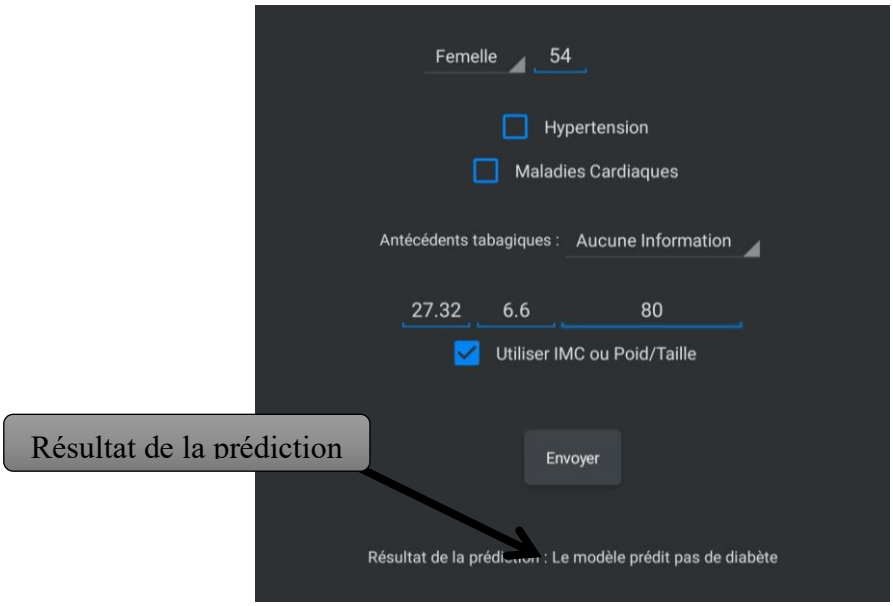


Figure 17 Démonstration du fonctionnement de l'application Android

5.8. Limites des Paramètres d'Entrée

Pour garantir la précision des prédictions de l'application, des limites ont été définies pour les paramètres d'entrée suivants :

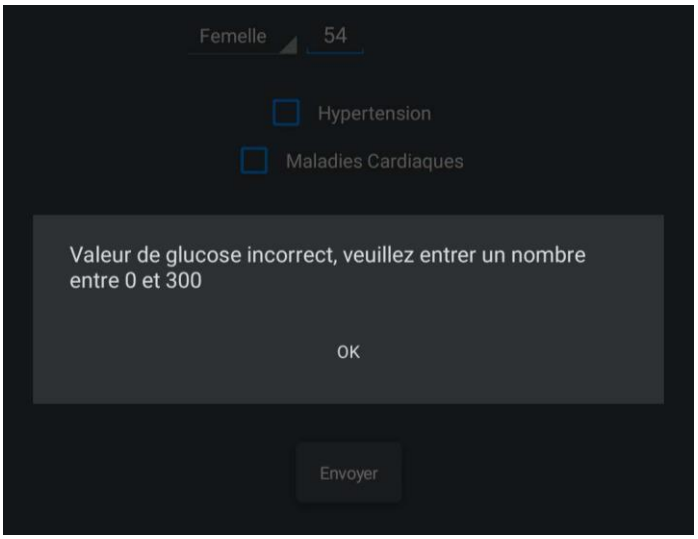


Figure 18 Message d'erreur

- **Âge (Age)** : Doit être un nombre entier compris entre 0 et 100 ans. Toute valeur en dehors de cette plage entraînera un message d'erreur indiquant : "Valeur d'âge incorrecte, veuillez entrer un nombre entre 0 et 100."
- **Niveau de glucose sanguin (Blood Glucose Level)** : Doit être un nombre compris entre 0 et 300 mg/dL. Toute valeur supérieure déclenchera un message d'erreur : "Valeur de glucose incorrecte, veuillez entrer un nombre entre 0 et 300."

- **Niveau d'HbA1c (HbA1c Level)** : Doit être un nombre compris entre 0 et 10 %. Toute valeur supérieure affichera un message d'erreur : "Valeur d'HbA1c incorrecte, veuillez entrer un nombre entre 0 et 10."
- **Indice de masse corporelle (BMI)** : Doit être un nombre compris entre 0 et 90 kg/m². Toute valeur supérieure entraînera un message d'erreur : "Valeur de BMI incorrecte, veuillez entrer un nombre entre 0 et 90."

Ces contraintes sont essentielles pour maintenir l'intégrité des données et la validité des prédictions. Elles reflètent également les plages de valeurs cliniquement pertinentes pour chaque paramètre.

5.9. Choix entre l'IMC ou le calcul avec Poid et Taille

Dans cette section, nous discutons de la flexibilité offerte aux utilisateurs quant à la saisie de l'indice de masse corporelle (IMC) ou des paramètres individuels de poids et de taille pour calculer l'IMC. Cette fonctionnalité vise à accommoder les différentes préférences et situations des utilisateurs, garantissant ainsi une expérience utilisateur plus intuitive et adaptable.

L'IMC est un indicateur couramment utilisé pour évaluer la corpulence d'une personne en fonction de son poids et de sa taille. Cependant, tous les utilisateurs ne connaissent pas nécessairement leur IMC exact, mais peuvent facilement fournir leur poids et leur taille. Pour cette raison, l'application permet deux méthodes de saisie :

Saisie directe de l'IMC :

- L'utilisateur peut directement entrer son IMC s'il connaît déjà cette valeur. Cela simplifie le processus de saisie des données, car il n'est pas nécessaire de calculer l'IMC séparément.

Calcul de l'IMC à partir du poids et de la taille :

Alternativement, l'utilisateur peut fournir son poids en kilogrammes et sa taille en mètres. L'application calcule alors automatiquement l'IMC en utilisant la formule :

$$IMC = \frac{Poid\ (Kg)}{Taille(m)^2}$$

Cette méthode est particulièrement utile pour les utilisateurs qui ne connaissent pas leur IMC, mais ont accès à leur poids et taille.

Interface Utilisateur pour la Saisie de l'IMC ou du Poids et de la Taille

L'interface utilisateur de l'application est conçue pour offrir une transition fluide entre les deux méthodes de saisie. Une case à cocher intitulée "Utiliser IMC ou Poids/Taille" est disponible. Lorsque cette case est cochée, l'utilisateur peut entrer son IMC directement. Lorsque la case est décochée, les champs de saisie pour le poids et la taille apparaissent, permettant à l'utilisateur de fournir ces informations.

- **Case cochée :**

- Le champ de saisie de l'IMC est visible.
- Les champs de saisie du poids et de la taille sont masqués.

- **Case décochée :**

- Les champs de saisie du poids et de la taille sont visibles.
- Le champ de saisie de l'IMC est masqué.

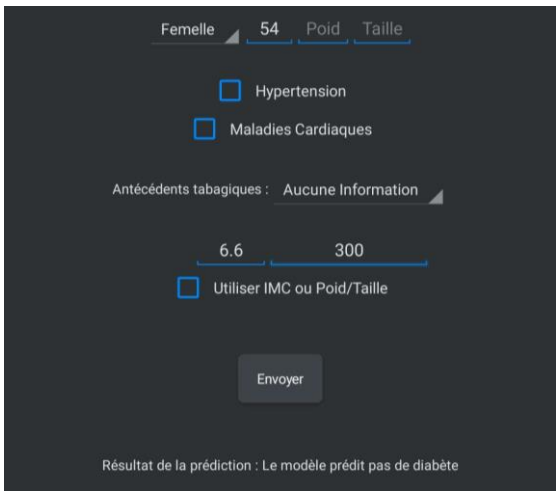


Figure 19 Utilisation du Poid et Taille

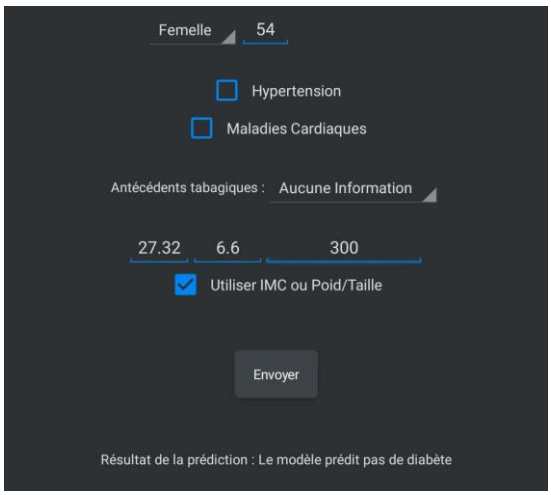


Figure 20 Utilisation de l'IMC directement

Vérification des Valeurs Entrées

Pour assurer la précision des prédictions, l'application inclut des vérifications pour s'assurer que les valeurs saisies respectent les limites définies. Par exemple, si l'utilisateur entre une valeur de poids ou de taille qui n'est pas réaliste, l'application alerte l'utilisateur avec un message d'erreur et ajuste la valeur saisie pour rester dans des plages acceptables.

Voici quelques exemples de messages d'erreur pour les saisies incorrectes :

- Pour le Poids : “Valeur de poids incorrecte, veuillez entrer un nombre valide.”
- Pour la Taille : “Valeur de taille incorrecte, veuillez entrer un nombre valide.”

En offrant ces deux méthodes de saisie et en assurant la vérification des données, l'application garantit une expérience utilisateur flexible et sécurisée, tout en maintenant l'intégrité des données nécessaires pour des prédictions précises du risque de diabète.

Conclusion

Dans le cadre de ce projet, nous avons développé un modèle neuronal unique et robuste qui a démontré une capacité remarquable à prédire le diabète. En exploitant les données cliniques et en appliquant des techniques d'apprentissage profond, notre modèle a pu identifier avec précision les cas de diabète, offrant ainsi un outil potentiellement vital pour le diagnostic précoce et la gestion de cette maladie chronique. Les résultats obtenus témoignent de l'efficacité de l'approche choisie et ouvrent la voie à de futures recherches pour affiner et améliorer la précision de la prédiction du diabète.

En somme, notre modèle neuronal unique a montré des résultats très satisfaisants dans la prédiction du diabète. Bien que nous n'ayons pas atteint une précision absolue de 100%, les performances obtenues sont prometteuses. La complexité des données biomédicales et la variabilité individuelle des patients sont des défis constants, mais notre modèle représente une avancée significative dans le domaine du diagnostic assisté par ordinateur.

Nous encourageons la poursuite de la recherche et du développement pour améliorer continuellement la précision et la fiabilité des diagnostics médicaux.

Conclusion Générale

Conclusion Générale

Ce mémoire a exploré en profondeur les réseaux de neurones artificiels (RNA), leurs bases théoriques, leurs méthodes d'implémentation et leurs applications pratiques, en mettant un accent particulier sur la prédiction du diabète. En examinant ces divers aspects, nous avons pu apprécier la puissance et la polyvalence des RNA dans la résolution de problèmes complexes et leur potentiel à transformer des domaines critiques comme la santé.

Synthèse

Nous avons commencé par une introduction détaillée aux fondements théoriques des RNA. En explorant les différents types de réseaux – les réseaux de neurones artificiels (ANN), les réseaux de neurones récurrents (RNN), et les réseaux de neurones Convolutionnels (CNN) – nous avons établi une compréhension solide des architectures et des mécanismes d'apprentissage qui sous-tendent ces modèles. Nous avons également discuté des avantages et des inconvénients des RNA et de leurs diverses applications dans des domaines tels que la vision par ordinateur et le traitement du langage naturel.

Le deuxième chapitre a détaillé l'implémentation pratique des RNA sur différentes plateformes. Nous avons démontré comment les réseaux de neurones peuvent être développés et utilisés dans des environnements logiciels comme MATLAB et des environnements de développement comme Visual Studio Code. L'implémentation sur des microcontrôleurs, en particulier l'Arduino ESP32, a montré comment les RNA peuvent être intégrés dans des systèmes embarqués, offrant des solutions puissantes et compactes pour des applications réelles.

Le dernier chapitre s'est concentré sur l'application pratique des RNA dans le domaine de la santé, spécifiquement pour la prédiction du diabète. Nous avons développé un système intégré capable de prédire le diabète à partir de données réelles, en utilisant des techniques d'apprentissage automatique et de Deep learning. Le processus a inclus la recherche de bases de données, l'entraînement du modèle, sa conversion en TensorFlow Lite et son déploiement sur une plateforme Arduino, ainsi que la création d'une application mobile pour une interface utilisateur intuitive. Cette application a démontré les possibilités offertes par les RNA pour le diagnostic médical.

Perspectives d'Avenir

Les réseaux de neurones artificiels continueront à évoluer et à s'améliorer, ouvrant la voie à de nouvelles applications et innovations. Les avancées en matière de matériel informatique, telles que les processeurs neuromorphiques et les accélérateurs d'IA, permettront des calculs plus rapides et plus efficaces, augmentant ainsi les capacités des RNA. De plus, l'intégration de

techniques avancées comme l'apprentissage par transfert et les réseaux antagonistes génératifs (GAN) pourrait encore accroître la précision et la robustesse des modèles.

Dans le domaine de la santé, l'utilisation des RNA pour le diagnostic et la prédiction des maladies deviendra de plus en plus courante. Les systèmes de prédiction du diabète développés dans ce mémoire ne sont qu'un exemple parmi tant d'autres des possibilités offertes par les RNA. En continuant à améliorer ces modèles et à les adapter à de nouvelles conditions cliniques, nous pourrons offrir des outils de diagnostic plus précis et accessibles, contribuant ainsi à une meilleure prise en charge des patients.

En conclusion, ce mémoire a mis en lumière le potentiel significatif des réseaux de neurones artificiels dans divers domaines, avec une focalisation particulière sur la prédiction du diabète. En combinant théorie, pratique et application, nous avons démontré comment les RNA peuvent être utilisés pour résoudre des problèmes complexes et offrir des solutions innovantes. Les résultats obtenus ici sont prometteurs et ouvrent la voie à de futures recherches et développements dans le domaine des réseaux de neurones et de l'intelligence artificielle en général

Bibliographie

1. Colmerauer alain, Panorama de l'intelligence artificielle : ses bases méthodologiques ses développements

   https://www.fnac.com/a7320001/Peter-Marquis-Panorama-de-l-intelligence-artificielle-ses-bases-methodologiques-ses-developpements
2. Asim Zulfiqar, Hands-on ESP32 with Arduino IDE: Unleash the power of IoT with ESP32 and build exciting projects with this practical guide, 2024
3. TensorFlow, Tutorials

   https://www.tensorflow.org/tutorials/quickstart/beginner?hl=fr
4. Kernix, TensorFlow un écosystème Open Source dédié à la création des modèles de Deep learning

   https://www.kernix.com/tensorflow-un-ecosysteme-open-source-dedie-a-la-creation-des-modeles-de-deep-learning
5. LazyProgrammer, Deep Learning and Natural Language Processing
6. Jason Brownlee, Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems, 2017
7. TensorFlow, TensorFlow Lite

   https://www.tensorflow.org/lite/guide?hl=fr
8. Karthiek Reddy Bokka, Deep Learning for Natural Language Processing
9. CEA, L'essentiel sur L'intelligence artificielle

   https://www.cea.fr/comprendre/Pages/nouvelles-technologies/essentiel-sur-intelligence-artificielle.aspx
10. GARY HALLBERG, Wireless Communications with Arduino and the ESP32

    https://www.amazon.com/Wireless-Communications-Arduino-ESP32-Short-ebook/dp/B09K5PSWLS
11. uPesy, How to connect to a WiFi network with the ESP32

    https://www.upesy.com/blogs/tutorials/how-to-connect-wifi-acces-point-with-esp32
12. The OpenEDG Python Institute ,Python Essentials 1: The Official OpenEDG Python Institute beginners course with practical exercises
13. Publishing & AI ,Python NumPy for Beginners: NumPy Specialization for Data Science
14. Arduino, Wi-Fi® / ESP32

    https://docs.arduino.cc/arduino-cloud/hardware/wifi/

15. DroidScript, Docs

    https://docs.arduino.cc/arduino-cloud/hardware/wifi/
16. Quora , Qu’est-ce que le deep learning (apprentissage profond) ?

    https://fr.quora.com/Qu-est-ce-que-le-deep-learning-apprentissage-profond
17. Forum Arduino, tension dalimentation du arduino

    https://forum.arduino.cc/t/tension-dalimentation-du-arduino-nano-33-ble-sense-rev2/1172854
18. DataScientest, Convolutional Neural Network : Tout ce qu’il y a à savoir !

    https://datascientest.com/convolutional-neural-network
19. WikiPédia, Recurrent Layer Neural Network

    https://fr.m.wikipedia.org/wiki/Fichier:RecurrentLayerNeuralNetwork.png
20. DroidScript, XMLHttpRequest Example

    https://droidscript.org/wiki/doku.php?id=sample_code:sample_xmlhttprequest