Rapport final de projet spécialisé

Reconstruction de champs d'écoulement fluide avec des réseaux de neurone informés par la physique (PINNs)

Blaise Madiega, 111 253 805

Baccalauréat en génie mécanique

RAPPORT ADRESSÉ À

Professeur: Mathieu Olivier

GMC-3550 Hiver 2023

July 25, 2023

1 Résumé

En résumé, ce projet de recherche vise à résoudre l'équation de Navier-Stokes pour un écoulement laminaire incompressible autour d'un cylindre à travers l'apprentissage profond notamment en utilisant des réseaux de neurones informés par la physique (PINNs). Les PINNs représentent une méthode innovante et prometteuse pour traiter des problèmes d'écoulement en mécanique des fluides sans recourir à un maillage ou une discrétisation, tout en intégrant les lois de la physique afin d'améliorer la précision des prédictions. Cette approche offre également des avantages en termes de coût computationnel et de flexibilité.

Dans le cadre de ce projet, un modèle PINN a été développé et entraîné pour simuler l'écoulement fluide autour d'un cylindre, en s'appuyant sur les connaissances physiques pour assurer la précision et la cohérence des résultats. Les performances du modèle sont ensuite analysées en termes d'exactitude et de respect des contraintes physiques. De plus, différentes techniques sont explorées pour optimiser l'architecture du réseau et les hyperparamètres du modèle, ainsi que pour améliorer sa robustesse face à des données limitées.

L'application réussie des PINNs à cette problématique prouve bien que l'apprentissage profond scientifique peut ouvrir la voie à de nouvelles approches dans la modélisation, la simulation et l'optimisation des écoulements pour une variété d'applications en ingénierie et en sciences. Par exemple, ce domaine de l'IA pourrait être utilisé pour concevoir des systèmes de refroidissement plus efficaces, optimiser les performances aérodynamiques des véhicules ou améliorer la compréhension des écoulements naturels tels que les courants océaniques et les mouvements atmosphériques.

Sommaire

2 Liste des symboles et abréviations

Liste des symboles et abréviations	
Symboles/Abréviations	**Définition**
IA	Intelligence Artificielle
PINNs	Physics-Informed Neural Networks
CFD	Computational Fluid Dynamics
EDP	Équation aux Dérivées Partielles
MLP	Multi-Layers Perceptron
ANN	Artificial Neural Network
AD	Auto-Différentiation
SGD	Stochastic Gradient Descent

3 Introduction

L'ingénierie est le domaine qui a tiré le plus grand profit des avancées de l'intelligence artificielle (IA) ces dernières années. Les applications de l'IA en ingénierie sont variées, allant de *la conception de produits à la planification de la production, en passant par la maintenance prédictive*. Parmi les applications les plus prometteuses figure la simulation, basée sur des outils mathématiques et informatiques. En mécanique des fluides, par exemple, la simulation numérique permet de résoudre des problèmes tels que la prédiction des performances de vol d'un avion ou la conception d'un échangeur de chaleur. Les méthodes traditionnelles de simulation numérique en mécanique des fluides, telles que les éléments finis ou les volumes finis, sont efficaces pour résoudre des problèmes simples ou de taille modérée. Cependant, elles peuvent rapidement devenir coûteuses en terme de temps de calcul pour des problèmes complexes ou de grande envergure.

Les réseaux de neurones informés par la physique (PINNs) constituent une nouvelle approche d'apprentissage profond combinant les atouts des modèles physiques et des réseaux de neurones artificiels. Les PINNs emploient un ensemble d'équations différentielles pour décrire le comportement physique et intègrent ces équations directement dans l'architecture du réseau neuronal. Cela permet aux PINNs d'utiliser les données observables pour résoudre des problèmes complexes tels que l'inférence inverse ou la synthèse fonctionnelle. En intégrant explicitement les contraintes physiques dans l'architecture du modèle, ils produisent des résultats plus robustes et stables que ceux obtenus uniquement avec les techniques traditionnelles d'apprentissage profond. Contrairement aux autres approches basées sur l'apprentissage profond, les PINNs ne nécessitent pas beaucoup de données d'entraînement car ils exploitent la structure générale connue du système étudié. De plus, grâce à leur capacité unique d'incorporer explicitement les connaissances physiques, il est possible de construire non seulement un meilleur modèle, mais également de découvrir certaines propriétés mathématiques implicites jusqu'alors inconnues.

Le présent projet porte sur la construction d'un modèle de réseau de neurones informé par la physique pour reconstruire l'écoulement de fluide autour d'un cylindre à partir de quelques données. Les deux principaux objectifs du projet sont dans un premier temps d'acquérir des données à travers une simulation numérique afin d'avoir une solution de référence du problème puis ensuite de fusionner les connaissances en physique avec les capacités d'apprentissage des réseaux neuronaux afin de créer un modèle prédictif à évaluer avec les données acquises. Ce modèle doit être capable de prédire les caractéristiques de l'écoulement du fluide (vitesse, pression, etc.) en fonction des conditions initiales et des paramètres du système, avec peu de données d'entraînement acquises par simulation numérique (CFD).

Dans les lignes qui suivent, le problème est d'abord défini de façon plus détaillée avec une revue de littérature; ensuite la théorie et les concepts impliqués dans la résolution de la problématique, la méthodologie utilisée et enfin les résultats suivis d'une analyse de ces derniers sont à présenter.

4 Description du sujet d'étude

4.1 Définition du problème

Lorsque les particules d'un fluide s'écoulent autour d'un cylindre de manière régulière et symétrique, l'écoulement est dit laminaire. Cet écoulement stable et prévisible se produit généralement à des vitesses de fluide faibles et à des nombres de Reynolds faibles (Re<2000). Le nombre de Reynolds est une grandeur adimensionnelle qui décrit le rapport entre les forces inertielles et visqueuses dans un écoulement. Toutefois, si l'écoulement laminaire rencontre un obstacle, tel qu'un cylindre, le phénomène de Von Karman peut survenir.

Le phénomène de Von Karman se caractérise par la formation de tourbillons alternatifs et instables à l'arrière du cylindre. Ces tourbillons se détachent périodiquement des bords du cylindre et se déplacent dans le sillage, créant une zone de turbulence. Ce phénomène présente une fréquence caractéristique, appelée fréquence de Strouhal, qui dépend de la vitesse d'écoulement et de la géométrie du cylindre (Williamson, 1996 [1]). Le phénomène de Von Karman a des implications pratiques dans divers domaines, tels que la conception de ponts, de bâtiments, d'aéronefs et de véhicules automobiles. Comprendre ce phénomène est crucial pour la conception efficace de ces structures, car il peut provoquer des vibrations indésirables et augmenter la traînée.

Dans ce projet, le point de mire est de reconstruire un écoulement laminaire incompressible autour d'un cylindre à bas Reynolds (pour rester en régime stationnaire et éviter l'effet Von Karman) en utilisant des réseaux de neurones informés par la physique (PINNs) à base de quelques données de simulation. Les paramètres du problème dont le domaine de calcul est illustré sur la **Figure 4.1** sont les suivants:

- **diamètre du cylindre:** $10\,m$

- **viscosité cinématique:** $0.5\,m^2/s$

- $U_\infty = 1\,m/s$

- **Nombre de Reynolds:** 20

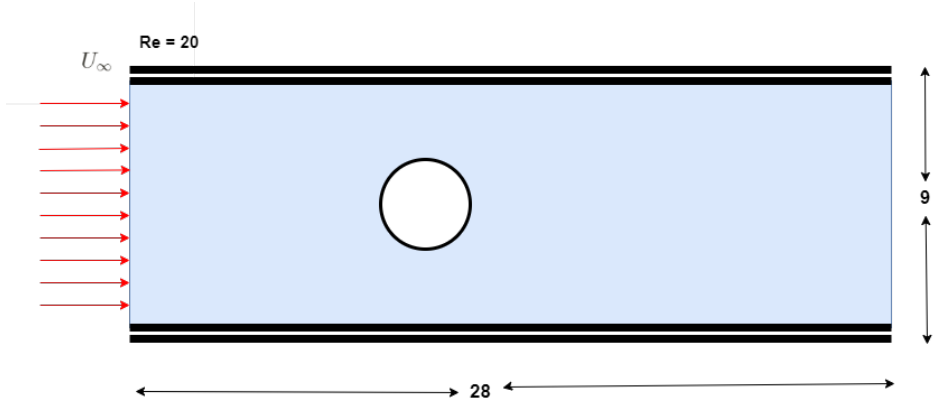

Figure 4.1 Représentation de la géométrie du cylindre et du domaine de calcul pour le champ de vitesse et de pression

4.2 Revue de littérature

4.2.1 Travaux réalisés avec les PINNs

Les réseaux de neurones sont des structures de données multicouches capables d'apprendre à partir de données d'entraînement et de réaliser des prédictions sur des données inédites. Ils ont été employés avec succès dans divers domaines, notamment la reconnaissance d'image, la traduction automatique et la prédiction de séries temporelles.

Ces réseaux ont également trouvé des applications dans la résolution d'équations différentielles, qui sont des instruments mathématiques cruciaux pour modéliser les systèmes dynamiques dans de nombreux secteurs, tels que la physique, la chimie et la biologie. Bien que les équations différentielles soient généralement ardues à résoudre analytiquement, des méthodes numériques telles que les différences finies ou les éléments finis permettent de les résoudre efficacement.

Au cours des dernières années, les réseaux de neurones informés par la physique (PINNs) ont suscité un intérêt croissant dans la résolution de problèmes complexes en science et en ingénierie. Ils ont démontré leur efficacité en intégrant des connaissances physiques a priori dans des modèles d'apprentissage profond, permettant ainsi de résoudre des problèmes non linéaires avec une précision et une efficacité accrues.

Parmi les travaux notables, Raissi et al. [2] (2019) ont développé un cadre d'apprentissage profond pour résoudre des problèmes directs et inverses impliquant des équations aux dérivées partielles non linéaires à l'aide des PINNs. Cette avancée a ouvert la voie à de nombreuses autres recherches explorant diverses applications des PINNs. Par exemple, Yang et Perdikaris [3] (2019) ont étendu l'application des PINNs aux équations différentielles stochastiques, offrant ainsi une nouvelle perspective pour traiter l'incertitude dans les modèles physiques.

Shyma Alhuwaider [4] a exploré l'utilisation de réseaux de neurones comme **solveurs d'équations différentielles partielles (EDP)** pour prédire les solutions de la dynamique des fluides compressibles. Les réseaux neuronaux apprennent à partir de données tout en étant contraints également par la physique pour garantir que les solutions générées soient précises et réalisables. Cette thèse démontre les capacités des réseaux neuronaux pour l'approximation des solutions de la dynamique des fluides compressibles et la recherche de formes aérodynamiques réalisables à travers la prédiction d'onde de choc 1D en fonction des conditions initiales et limites, la prédiction de l'écoulement du fluide en 2D autour d'ellipses, la conception inverse pour minimiser le coefficient de traînée, l'utilisation d'une **architecture PointNet** pour approximer le flux autour d'un objet donné.

Bar-Sinai et al. [5] (2019) ont proposé une approche d'apprentissage pour les discrétisations de problèmes aux dérivées partielles. Cette méthode a montré que l'apprentissage profond peut être utilisé pour développer des schémas numériques adaptatifs en fonction des caractéristiques spécifiques du problème. De plus, Li et al. [6] (2020) ont introduit l'opérateur neuronal de Fourier pour les équations aux dérivées partielles paramétriques, démontrant la capacité des PINNs à apprendre des opérateurs pour des systèmes dépendant de paramètres.

Les PINNs ont également été appliqués avec succès à la résolution de problèmes inverses soumis à des contraintes. Lu et al. [7](2020) ont développé un cadre de PINNs pour résoudre des problèmes inverses

avec des contraintes physiques explicites, ce qui améliore la robustesse et la stabilité des solutions.

En ce qui concerne les applications dans d'autres domaines d'ingénierie, Rao et al. [8] (2020) ont exploré l'utilisation des PINNs pour modéliser les élastodynamiques. Par ailleurs, Daniele Secci et al. [9] (2022) ont appliqué les PINNs pour identifier les sources de contamination dans les écoulements de fluides. Ces travaux montrent la capacité des PINNs à traiter des phénomènes physiques complexes avec une précision remarquable.

De plus, Stiasny et al. [10] (2021) démontrent l'efficacité des PINNs dans l'estimation de l'état dynamique des systèmes électriques, offrant ainsi une nouvelle approche pour résoudre des problèmes importants en génie électrique. Enfin, Sirignano et Spiliopoulos [11] (2020) ont utilisé des PINNs pour modéliser la propagation d'ondes sismiques et acoustiques en milieu hétérogène, illustrant l'énorme potentiel des PINNs dans diverses applications pratiques.

4.2.2 Avantages de l'utilisation des PINNs

Intégration des lois physiques : Les PINNs incorporent les lois physiques sous-jacentes directement dans leur structure, ce qui permet d'améliorer la généralisation et la précision de leurs prédictions [2].

Efficacité computationnelle : Ils permettent de résoudre les EDP sans nécessiter de maillage ni de discrétisation, réduisant ainsi les coûts de calcul [12].

Flexibilité : Ils sont également capables de traiter des problèmes avec des conditions aux limites complexes et des incertitudes dans les données [13].

Traitement des géométries complexes : Leur utilisation permet de modéliser l'écoulement des fluides dans des géométries complexes avec des limites irrégulières sans qu'il soit nécessaire de générer un maillage.

Capacité de traitement des données manquantes : Les PINNs peuvent combler les lacunes dans les données d'entrée et offrir des prédictions fiables même en présence de données manquantes ou de données bruitées [14] [15].

4.2.3 Défis et perspectives futures

Scalabilité : L'augmentation de la taille du problème et de la dimensionnalité des données peut poser des problèmes de mémoire et de temps de calcul pour les PINNs. De nouvelles méthodes d'optimisation et des architectures de réseaux de neurones adaptées peuvent aider à résoudre ce problème [16].

Robustesse : Les PINNs peuvent être sensibles aux erreurs d'approximation et aux erreurs numériques, en particulier dans des conditions extrêmes ou des situations fortement non linéaires. Des recherches supplémentaires sont nécessaires pour améliorer leur robustesse et leur tolérance aux erreurs[17].

Incertitudes et quantification des incertitudes : Ces modèles pourraient bénéficier de l'intégration de méthodes de quantification des incertitudes pour fournir des estimations plus fiables et robustes de la dynamique des fluides [18].

Apprentissage multi-échelle et multi-fidélité : L'intégration de méthodes d'apprentissage multi-échelle et multi-fidélité peut permettre aux PINNs de capturer efficacement des phénomènes de grande et petite échelle, ainsi que de combiner des données de simulations numériques et expérimentales de différentes résolutions et fidélités [19].

Sensibilité aux conditions limites et initiales: Ces réseaux de neurones peuvent être sensibles aux conditions initiales et aux paramètres de la fonction de perte. Les PINNs peuvent être entraînés pour minimiser la fonction de perte sur l'ensemble d'entraînement sans pour autant satisfaire les équations différentielles à cause des conditions limites.

4.3 Théorie et concepts

4.3.1 Modèle de base : le perceptron

Le **perceptron** est un concept fondamental de l'apprentissage profond, fournissant la brique de base pour des réseaux neuronaux plus complexes. Il s'agit d'un algorithme utilisé pour classer les données en deux catégories (*classification binaire*) en utilisant un ensemble de poids et de valeurs de biais qui sont ajustés à mesure que le modèle apprend de ses erreurs. En substance, on peut le considérer comme un neurone artificiel qui prend des signaux d'entrée et décide s'ils doivent être classés dans une catégorie ou une autre en fonction de leur valeur.

Proposé par Frank Rosenblatt en 1957 [20], ce modèle fonctionne de la manière suivante:

- l'entrée du modèle **x** est reçue sous forme d'un vecteur de nombres réels,

- pour chaque entrée dans le vecteur, le modèle dispose d'un poids **w** qui lui est multiplié,

- suite aux multiplications des entrées par leurs poids respectifs, les résultats sont sommés,

- un biais **b** est ensuite ajouté à la somme afin d'obtenir la sortie finale **z**.

Cela se traduit par l'équation suivante:

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^t \mathbf{x} + b \tag{1}$$

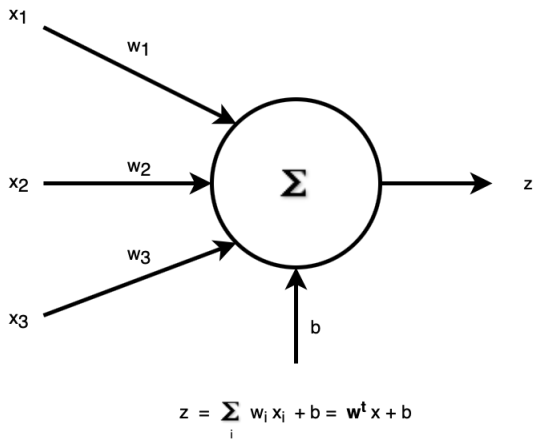

Figure 4.2 : Modèle du perceptron

La **Figure 4.2** représente les différents paramètres définissant un perceptron en l'occurrence les poids **w** et le biais **b**.

4.3.2 Fonctions d'activation

Les **fonctions d'activation** sont un élément important de l'apprentissage profond, car elles jouent un rôle clé dans le développement des réseaux neuronaux artificiels. Les fonctions d'activation déterminent comment les entrées d'une couche affectent les sorties d'une autre couche au sein d'un réseau. Elles décident essentiellement si les informations doivent ou non passer par le réseau et comment elles doivent être traitées par chaque neurone.

Pour un problème de classification binaire par exemple, il est nécessaire de corriger la sortie finale du neurone **z** afin d'obtenir une sortie booléenne qui permet de valider si l'entrée appartient à la **classe 0** ou à la **classe 1**.

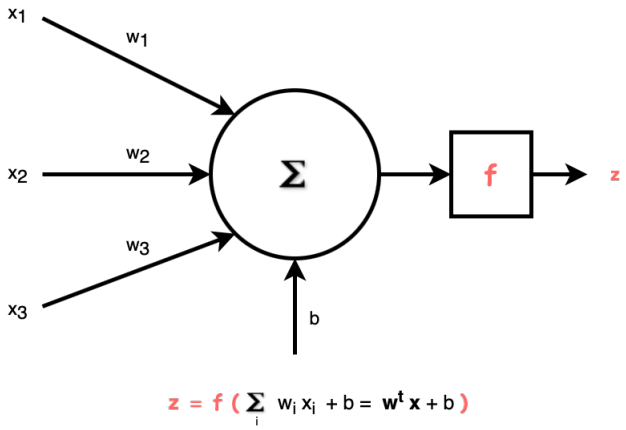

Figure 4.3 : Modèle du perceptron avec une fonction d'activation

En modifiant la sortie finale du neurone (**Figure 4.3**), les fonctions d'activation permettent d' obtenir une **sortie plus interprétable** et également d'**introduire de la non-linéarité**[21] afin de permettre aux réseaux de neurones d'atteindre de meilleures performances sur des jeux de données qui ne sont pas linéairement séparables.

Le type de fonctions d'activation utilisé dépend de la catégorie de problème d'apprentissage profond à résoudre. La fonction d'activation la plus couramment utilisée est **l'unité linéaire rectifiée (ReLU)**, qui a été largement adoptée en raison de son efficacité lorsqu'il s'agit de traiter des ensembles de données non linéaires tels que des images ou des tâches de reconnaissance vocale. ReLU prend toutes les valeurs d'entrée négatives et les transforme en **0** (voire **Figure 4.4**) tout en laissant les valeurs positives inchangées ; cela permet d'optimiser le temps d'entraînement puisque seuls les points de données pertinents doivent être pris en compte lors des calculs.

D'autres fonctions d'activation couramment utilisées comprennent les fonctions **sigmoïde, tanh, softmax, leaky ReLU**, etc., qui ont toutes leurs propres avantages en fonction du type de tâche pour laquelle elles sont appliquées. La fonction sigmoïde est plus efficace pour les problèmes de classification binaire tandis que la fonction tanh performe le mieux pour les problèmes de multi-classification comme les tâches de reconnaissance d'images impliquant plusieurs étiquettes par échantillon d'image.

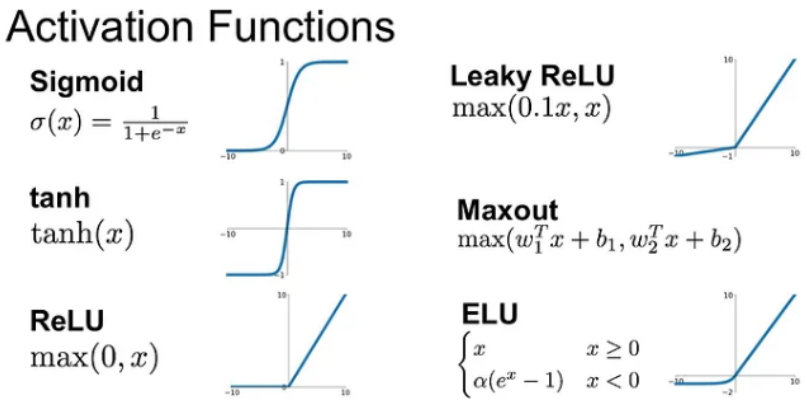

Figure 4.4 : Différentes fonctions d'activation et leurs graphes [22]

4.3.3 Perceptron Multi-Couche (MLP)

Un **perceptron multicouche (MLP)** est un type de réseau de neurones artificiels qui se compose de plusieurs couches de neurones inter-connectés. C'est un réseau neuronal à propagation avant, ce qui signifie que l'information circule dans une direction, de la couche d'entrée à travers les couches cachées jusqu'à la couche de sortie.

Chaque neurone du MLP reçoit une entrée des neurones de la couche précédente, effectue une somme pondérée des entrées et applique une fonction d'activation pour produire une sortie. La sortie de chaque neurone est ensuite transmise en entrée aux neurones de la couche suivante comme l'illustre bien la **Figure 4.5**.

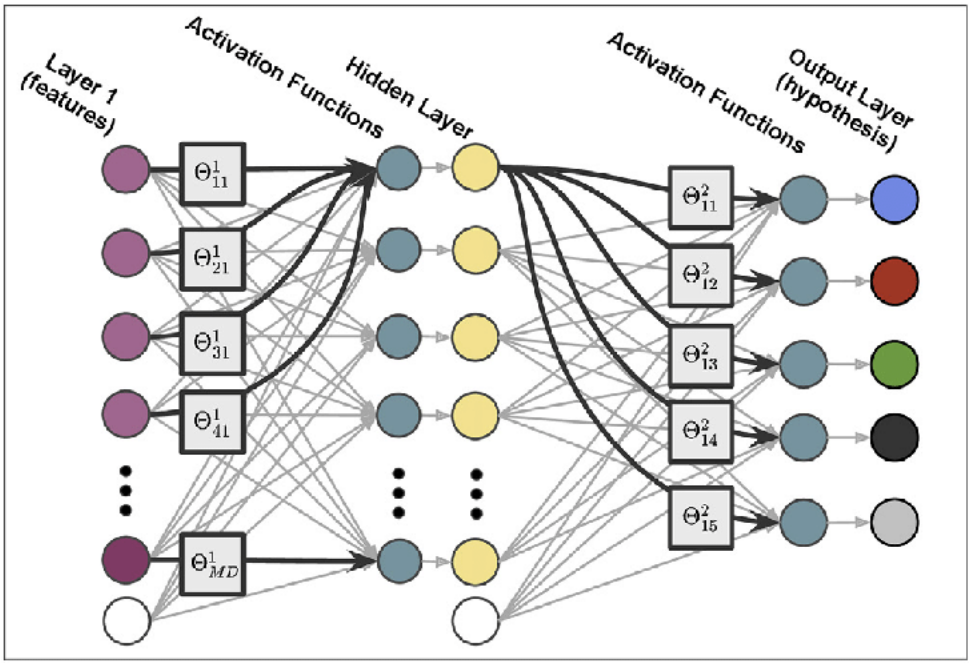

Figure 4.5 : Exemple d'architecture MLP pour un problème de multi-classification [22]

Le but des couches cachées dans le MLP est d'extraire et de transformer les caractéristiques des données d'entrée de manière à faciliter la classification ou la prédiction de la sortie. Le nombre de couches cachées et le nombre de neurones dans chaque couche sont des hyperparamètres qui peuvent être ajustés pour obtenir de meilleures performances sur une tâche spécifique.

L'algorithme de rétropropagation est utilisé pour entraîner le MLP en ajustant les poids des connexions entre les neurones. L'algorithme calcule l'erreur entre la sortie prédite et la sortie réelle et ajuste les poids pour minimiser l'erreur. La formule pour la sortie d'un neurone dans un MLP est similaire à celle d'un perceptron à une seule couche, mais avec des couches de calcul supplémentaires:

$$z_1 = \sigma(W_1 x + b_1) \tag{2}$$

$$z_{k+1} = \sigma(W_{k+1} z_k + b_{k+1}) \tag{3}$$

$$y = \mathbf{W}_{L+1} z_L + \mathbf{b}_{L+1} \tag{4}$$

Les matrices $\mathbf{W_k}$ et les vecteurs $\mathbf{b_k}$ représentent respectivement les **poids** et les **biais**. Les fonctions d'activation (σ) sont appliquées à la sortie de chaque couche cachée (L: l'indice de couche cachée) sauf au niveau de la dernière couche du MLP qui est celle de sortie le résultat ne doit pas être modifié pour une représentation fidèle de la sortie attendue dans le cas d'un problème de régression. Par contre pour un problème de classification comme celui de la **Figure 4.5** (ANN), avant la sortie on applique encore des fonctions d'activation pour avoir une sortie plus interprétable vis-à-vis du problème à l'étude notamment la multi-classification.

Les MLPs sont largement utilisés dans une variété d'applications, telles que la reconnaissance d'images et de la parole, le traitement du langage naturel et l'analyse prédictive. Ils sont particulièrement utiles dans les tâches qui impliquent des relations non linéaires entre les données d'entrée et de sortie.

4.3.4 Différentiation automatique (Autograd ou Autodiff)

`Définition et avantages`

L'un des défis de la formation des réseaux neuronaux consiste à calculer les gradients de la fonction de perte par rapport aux paramètres du modèle. Les gradients sont utilisés pour mettre à jour les paramètres du modèle pendant la formation, et ils sont nécessaires pour optimiser le modèle afin de minimiser la fonction de perte. Les méthodes traditionnelles de calcul des gradients, telles que la différentiation numérique et la différentiation symbolique, sont souvent coûteuses en terme de calcul et peuvent être sujettes à des erreurs numériques.

L'idée principale de la différentiation automatique est de chaîner les dérivées à travers des opérations élémentaires pour calculer la dérivée de la fonction entière. Pour ce faire, on applique la règle de la dérivation en chaîne, qui stipule que la dérivée d'une fonction composée est le produit des dérivées des fonctions individuelles de la chaîne[23].

$$\frac{d(g \circ f)}{dx} = \frac{dg}{df} \cdot \frac{df}{dx} \tag{5}$$

Dans le contexte de l'apprentissage profond, on peut utiliser la différentiation automatique pour calculer les gradients de la fonction de perte par rapport aux paramètres du modèle. Cela permet de mettre à jour les paramètres du modèle en utilisant la descente de gradient, qui est un algorithme d'optimisation populaire pour les réseaux neuronaux.

`Comparaison des différentes méthodes de différentiation`

- Dérivation numérique (cas des différences finies)

La dérivation numérique, en particulier les méthodes de différences finies, est une approche courante pour approximer les dérivées de fonctions à partir de données discrètes. Les avantages des méthodes de différences finies comprennent leur simplicité, leur facilité d'implémentation et leur adaptabilité à divers problèmes. Elles sont particulièrement utiles lorsque les expressions analytiques des dérivées ne sont pas disponibles ou difficiles à obtenir. Cependant, les méthodes de différences finies présentent certaines limites.

Tout d'abord, elles peuvent être sensibles **aux erreurs d'arrondi et à la discrétisation**, ce qui peut entraîner des approximations imprécises des dérivées, en particulier pour des pas de discrétisation h trop grands ou trop petits. De plus, elles ne sont pas toujours efficaces pour des fonctions complexes avec de nombreux paramètres.

- Dérivation symbolique

La différentiation symbolique est une méthode mathématique avancée qui permet de calculer la dérivée d'une fonction en utilisant des symboles plutôt que des valeurs numériques. Cette méthode est souvent utilisée en calcul différentiel pour simplifier les calculs de dérivées de fonctions complexes. Au lieu de calculer la dérivée en utilisant les règles de différentiation standard, la différentiation symbolique utilise des règles algébriques pour manipuler les expressions symboliques de la fonction. Ces règles peuvent être utilisées pour simplifier les expressions à dériver, ce qui rend le calcul de la dérivée plus facile et plus rapide.

Bien que la dérivation symbolique présente un énorme avantage en terme de **précision et d'exactitude** et pourrait être utile pour le calcul de gradients de fonctions en apprentissage profond, il est fort **limité du fait de la complexité des fonctions à dériver** dans ce domaine: les expressions symboliques peuvent devenir très grandes et difficiles à manipuler ce qui va inéluctablement ralentir le temps de calcul.

Pour illustrer ce problème, considérons l'exemple suivant.

Soit la fonction $l_{n+1} = 4l_n(1 - l_n)$ avec $l_1 = x$; $n \in [1, 5]$ et $n \in \mathbf{N}$.

Évolution des expressions en fonction de n		
n	l_n	$\frac{d}{dx} l_n$
1	x	1
2	$4x(1-x)$	$4(1-x) - 4x$
3	$-64x\,(x-1)\,(2x-1)^2\,(8x^2-8x+1)^2$	$-64\,(2x-1)\,(8x^2-8x+1)$ $(128x^4-256x^3+160x^2-32x+1)$
4	$-256x\,(x-1)\,(2x-1)^2\,(8x^2-8x+1)^2$ $(128x^4-256x^3+160x^2-32x+1)^2$	$-256\,(2x-1)\,(8x^2-8x+1)$ $(128x^4-256x^3+160x^2-32x+1)$ $(32768x^8-131072x^7+212992x^6+...)$
5	$-1024x\,(x-1)\,(2x-1)^2\,(8x^2-8x+1)^2$ $(128x^4-256x^3+160x^2-32x+1)^2$ $(32768x^8-131072x^7+...)^2$	$-1024\,(2x-1)\,(8x^2-8x+1)$ $(128x^4-256x^3+160x^2-32x+1)$ $(32768x^8-131072x^7+212992x^6+...)$ $(2147483648x^{16}-17179869184x^{15}+...)$

Tableau 4.1 Exemple illustrant l'évolution rapide des expressions symboliques

Avec les résultats consignés sur le tableau 4.1, un graphique (**Figure 4.6**) a été tracé pour illustrer l'évolution des expressions de la fonction et de sa dérivée en fonction de n. On voit bien que les expressions de la dérivée prennent une allure exponentielle au fur et à mesure que la valeur de n augmente; ce qui montre bien que cette méthode de dérivation est inefficace pour de l'optimisation algorithmique.

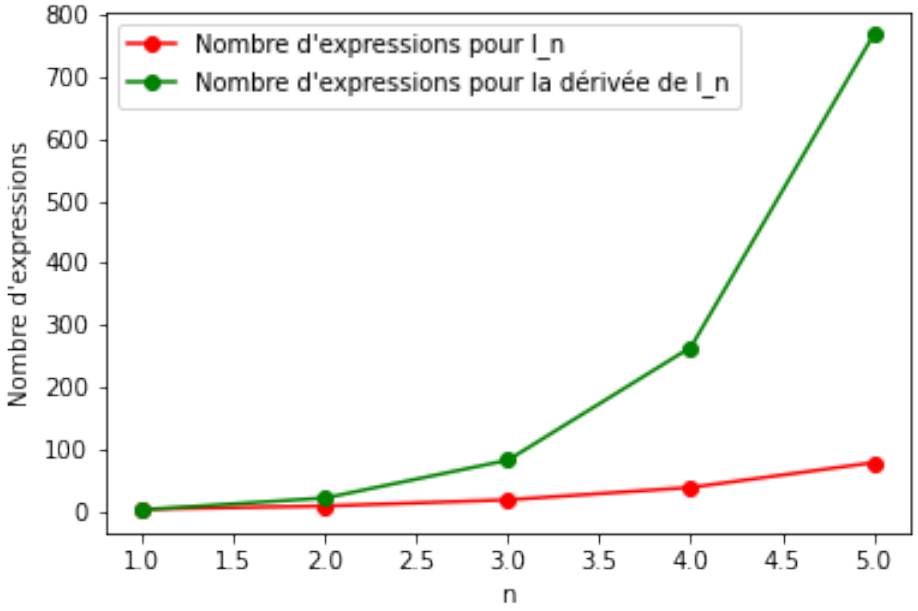

Figure 4.6 Illustration graphique de l'évolution rapide des expressions symboliques

- Autodiff (forces et limitations)

Parmi les forces de l'autodifférentiation, on compte **sa précision numérique et sa stabilité** [23], qui sont particulièrement avantageuses dans le contexte de l'apprentissage profond et de l'optimisation. L'autoDiff évite les problèmes d'erreurs d'arrondi et de discrétisation associés aux méthodes de différences finies. De plus, elle est capable de **traiter efficacement les fonctions ayant de nombreux paramètres**, ce qui la rend appropriée pour l'entraînement de réseaux de neurones profonds avec des millions de poids et biais.

Cependant, l'autoDiff présente également certaines limitations. Tout d'abord, son implémentation peut être plus complexe que celle des méthodes numériques traditionnelles, nécessitant souvent l'utilisation de bibliothèques spécialisées pour gérer les calculs de dérivées de manière optimisée. De plus, bien que l'autoDiff soit généralement plus rapide que les méthodes numériques pour les fonctions complexes, elle peut être plus lente pour les fonctions simples ou pour les modèles avec un nombre limité de paramètres [24]. Enfin, l'autoDiff ne résout pas tous les problèmes liés à l'optimisation, comme les minima locaux ou les plateaux, qui peuvent toujours impacter l'efficacité et la convergence des algorithmes d'apprentissage.

`Méthodes directe et inverse de l'autodiff`

Il existe deux modes principaux de différentiation automatique : le mode direct et le mode inverse. Le mode direct calcule la dérivée d'une fonction en propageant les dérivées des entrées à travers la séquence d'opérations élémentaires. Cela se fait en calculant les dérivées partielles de chaque opération par rapport à ses entrées et en les multipliant ensemble.

Le mode inverse, quant à lui, calcule la dérivée d'une fonction en propageant les dérivées de la sortie en arrière dans la séquence d'opérations élémentaires. Cela se fait en calculant les dérivées partielles de

chaque opération par rapport à ses sorties et en les multipliant ensemble.

En apprentissage profond, le mode inverse est généralement utilisé car il est plus efficace pour calculer les gradients par rapport à un grand nombre d'entrées, ce qui est courant dans les réseaux neuronaux.

`Implémentation de l'autodiff`

Pour mettre en œuvre la différentiation automatique dans un cadre d'apprentissage profond, il faut représenter la séquence d'opérations élémentaires comme un graphe de calcul. Chaque nœud du graphe représente une opération, et les bords représentent les entrées et les sorties des opérations.

Pendant la propagation vers l'avant, on évalue le graphe en calculant les sorties de chaque nœud du graphe. Pendant la propagation en arrière (voire **Figure 4.7**), les gradients de la fonction de perte sont calculés par rapport à chaque nœud du graphe en appliquant la règle de la dérivation en chaîne.

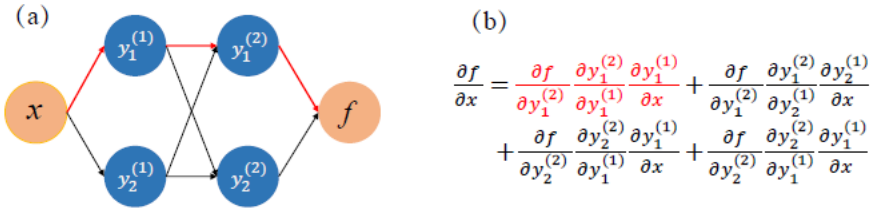

Figure 4.7 Schéma montrant les dépendances entre les différentes variables intérmédiaires, l'entrée et la sortie qui permettent d'implémenter un calcul de gradient sous forme de graphe (Autodiff) [23]. La dérivée de la sortie f par rapport à une entrée x est la somme de toutes les dérivées en chaîne des différents chemins rattachant l'entrée à la sortie.

4.3.5 Fonctions de perte

En apprentissage profond, les fonctions de coût, également appelées fonctions de perte, sont utilisées pour mesurer la différence entre la sortie prédite d'un réseau neuronal et la sortie réelle des données de formation. L'objectif du réseau neuronal est de minimiser la fonction de coût, ce qui signifie qu'il essaie de rendre ses prédictions aussi proches que possible de la sortie réelle.

Il existe plusieurs types différents de fonctions de coût qui sont couramment utilisés dans l'apprentissage profond. Le choix de la fonction de coût dépend de la tâche spécifique exécutée et du type de réseau neuronal utilisé.

Voici quelques-unes des fonctions de coût les plus couramment utilisées :

- **Erreur quadratique moyenne (EQM)** L'erreur quadratique moyenne (EQM) est une fonction de coût simple qui mesure l'erreur quadratique moyenne entre la sortie prédite et la sortie réelle. Elle est couramment utilisée dans les problèmes de régression, où l'objectif est de prédire une valeur numérique continue.

La formule de l'EQM est la suivante :

$$MSE = \frac{1}{N} * \sum (y_i - \hat{y}_i)^2 \tag{6}$$

Où N est le nombre d'échantillons d'apprentissage, y_i est la sortie réelle pour le i-ième échantillon d'apprentissage, et $\hat{y}_i$ est la sortie prédite pour le i-ième échantillon d'apprentissage.

- **L'entropie croisée binaire (BCE)** [25]

L'entropie croisée binaire (BCE) est une fonction de coût couramment utilisée dans les problèmes de classification binaire, où la sortie est une valeur binaire (0 ou 1).

La formule de la BCE est la suivante :

$$BCE = -\frac{1}{N} \sum (y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)) \tag{7}$$

Où N est le nombre d'échantillons d'apprentissage, y_i est la sortie réelle pour le i-ième échantillon d'apprentissage (soit 0 ou 1), et $\hat{y}_i$ est la sortie prédite pour le i-ième échantillon d'apprentissage (une valeur entre 0 et 1).

- **L'entropie croisée catégorielle (CCE)** [25]

L'entropie croisée catégorielle (CCE) est une fonction de coût couramment utilisée dans les problèmes de classification multi-classes, où la sortie est une valeur catégorielle (une parmi plusieurs classes possibles).

La formule pour le CCE est :

$$CCE = -\frac{1}{N} \sum \sum (y_{i,j} log(\hat{y}_{i,j})) \tag{8}$$

Où N est le nombre d'échantillons d'apprentissage, $y_{i,j}$ est un indicateur binaire (soit 0 ou 1) de l'appartenance du i-ième échantillon d'apprentissage à la j-ième classe, et $\hat{y}_i, j$ est la probabilité prédite que le i-ième échantillon d'apprentissage appartient à la j-ième classe.

Il existe plusieurs autres fonctions de coût qui sont utilisées dans l'apprentissage profond, en fonction de la tâche spécifique et de l'architecture du réseau.

4.3.6 Optimiseurs de fonction de perte

L'optimisation est une étape cruciale dans l'entraînement des réseaux de neurones artificiels, et plusieurs hyperparamètres peuvent influencer cette étape notamment l'**initialisation des paramètres**, la **fonction de perte**, la **taille des lots** et les **algorithmes d'apprentissage** (appelés également **optimiseurs**)

`Initialisation des paramètres (Poids et Biais)`

Les poids et les biais sont au cœur des réseaux de neurones, et leur initialisation a une influence considérable sur l'efficacité de l'entraînement en terme de temps et de convergence (Glorot Bengio, 2010[26]). Plusieurs méthodes d'initialisation des paramètres existent, et leur choix revêt une importance cruciale, car elles déterminent le point de départ du processus d'optimisation (He et al., 2015[27]).

En général, les biais des neurones dans un réseau sont initialisés à 0 ou à une valeur légèrement positive, telle que 0.01, en particulier lorsqu'une fonction d'activation **ReLU** est utilisée (Krizhevsky et al., 2012 [28]).

Pour l'initialisation des poids des neurones, il existe plusieurs stratégies.

- **Initialisation à une valeur constante**

L'**initialisation uniforme** des poids dans un réseau de neurones consiste à attribuer à tous les poids la même valeur initiale. Bien que cette méthode puisse sembler attrayante de par sa simplicité, elle présente toutefois des inconvénients majeurs (Glorot & Bengio, 2010 [26]). En effet, l'initialisation uniforme engendre une symétrie au sein du réseau, conduisant à une mise à jour identique des poids pendant l'optimisation (Sutskever et al., 2013 [29]). Cette situation limite considérablement la capacité d'apprentissage du réseau.

C'est pourquoi il est préférable d'adopter une initialisation aléatoire des poids, qui permet de briser la symétrie et favorise un apprentissage plus efficace.

- **Initialisation aléatoire**

Cette méthode consiste à initialiser les poids en échantillonnant à partir d'une distribution de probabilité. Parmi les méthodes d'initialisation aléatoire les plus courantes, on retrouve: l'initialisation **Xavier** et l'initialisation **He**.

Initialisation Xavier

Nommée après l'auteur de l'article scientifique qui l'a introduit, cette méthode consiste à initialiser les poids, d'une couche l , selon une distribution d'une loi normale de moyenne $\mu = 0$ et d'écart-type $\sigma = \sqrt{\frac{1}{n^{l-1}}}$ où n^{l-1} représente le nombre de neurones de la couche précédente.

$$W_i^l \sim N\left(0, \sqrt{\frac{1}{n^{l-1}}}\right) \tag{9}$$

L'initialisation Xavier est particulièrement efficace lorsque les fonctions d'activation sigmoïde et tanh sont utilisées. Cette méthode d'initialisation est aussi connue sous le nom de l'initialisation Glorot.

Initialisation He

L'initialisation He est similaire à l'initialisation Xavier tout en étant plus appropriée lors de l'utilisation de la fonction d'activation ReLU. La différence entre les deux méthodes est l'écart-type de la distribution d'une loi normale.

En effet, dans le cadre de l'initialisation He, les poids d'une couche l sont échantillonnés d'une loi normale de moyenne 0 et d'écart-type $\sqrt{\frac{2}{n^{l-1}}}$, où n^{l-1} représente le nombre de neurones de la couche précédente.

$$W_i^l \sim N\left(0, \sqrt{\frac{2}{n^{l-1}}}\right) \tag{10}$$

L'écart-type est ainsi multiplié par $\sqrt{2}$ pour l'initialisation He comparativement à l'initialisation Xavier. Cette méthode d'initialisation est aussi connue sous le nom de l'initialisation Kaiming.

`Optimiseurs de fonction de perte`

- Descente de gradient classique

La descente de gradient est une méthode d'optimisation numérique utilisée pour minimiser une fonction objectif, souvent appelée fonction de coût ou fonction de perte, dans le contexte de l'apprentissage automatique et des réseaux de neurones (Bottou et al., 2018 [30]). Cette technique repose sur le calcul du gradient de la fonction objectif par rapport aux paramètres du modèle, c'est-à-dire la dérivée partielle de la fonction objectif pour chaque paramètre. Le gradient indique la direction dans laquelle la fonction objectif croît le plus rapidement, et la descente de gradient effectue des mises à jour itératives des paramètres en les déplaçant dans la direction opposée au gradient (Ruder, 2016[31]).

La mise à jour des paramètres dans la descente de gradient est réalisée selon la formule suivante:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \tag{11}$$

où θ_t représente les paramètres du modèle à l'itération t, η est le **taux d'apprentissage** (un hyper-paramètre positif qui contrôle la taille des mises à jour) et $\nabla J(\theta_t)$ est le gradient de la fonction objectif (ou fonction de perte) J évaluée aux paramètres actuels θ_t (Goodfellow et al., 2016 [32]).

La descente de gradient peut être appliquée sous différentes formes, notamment la **descente de gradient par lot ("batch")**, la **descente de gradient stochastique (SGD)** et la **descente de gradient par petits lots ("mini-batch)"**. La principale différence entre ces variantes réside dans la manière dont les gradients sont calculés à partir des données d'entraînement (Géron, 2017 [33]).

Cet algorithme d'optimisation est largement utilisé dans les réseaux de neurones, mais il présente certains défis, tels que **la sensibilité au choix du taux d'apprentissage** et le **risque de rester bloqué dans des optima locaux ou des points selle** (Goodfellow et al., 2016 [32]). Plusieurs variantes et améliorations de la descente de gradient ont été proposées pour remédier à ces problèmes, notamment l'**optimiseur AdaGrad, RMSProp et Adam** (Kingma Ba, 2014 [34]; Duchi et al., 2011 [35];

Tieleman Hinton, 2012 [36]).

Taux d'apprentissage η

Comme énoncé précédemment, le taux d'apprentissage est un hyperparamètre important dans l'algorithme de descente de gradient. Il contrôle la taille des pas qui sont effectués sur la surface de la perte exprimée en fonction des paramètres du réseau, et dont le but est de minimiser celle-ci. Ainsi, la valeur du taux d'apprentissage influence grandement l'étape d'optimisation, car si les pas effectués sont trop petits l'entraînement risque de durer longtemps, alors que si les pas sont trop grands il risque d'être difficile d'atteindre un minimum intéressant. Considérons un exemple simple d'optimisation illustré à la **Figure 4.8** avec pour but de mener la boule bleue au bas de la courbe qui représente une fonction de perte.

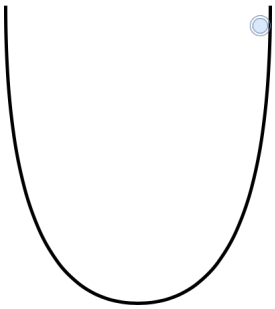

Figure 4.8 Exemple de fonction de perte simple à optimiser (trouver le minimum global)

Deux situations peuvent avoir lieu si on choisit le mauvais taux d'apprentissage (voire Figure 4.9):

- si le taux d'apprentissage est trop petit, la bulle bleue peut atteindre le minimum global de la courbe mais la convergence ne sera pas rapide et le temps d'entraînement plus long.

- si le taux d'apprentissage par contre est trop grand, la bulle fait de grands bonds et risque de manquer le minimum.

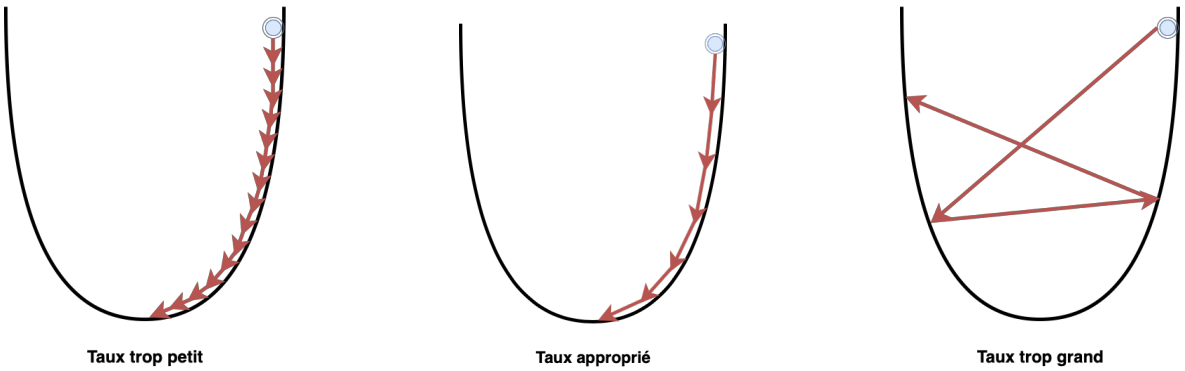

Figure 4.9 Exemple d'optimisation avec des mauvais taux d'apprentissage (*taux trop petit* ou *taux trop grand*) comparativement à un *taux approprié*.

La valeur du taux d'apprentissage choisi est très importante mais cela ne suffit pas parce que généralement les fonctions de perte sont des fonctions complexes comportant plusieurs minima locaux. Afin

de permettre une meilleure optimisation des réseaux de neurones artificiels, certaines modifications ont été apportées à l'algorithme classique de descente de gradient introduisant les concepts d'**algorithmes d'apprentissage avec "momentum"**.

- **Descente de gradient avec "momentum"**

Les algorithmes de descente de gradient avec "momentum" (dans le sens de quantité de mouvement) permettent d'adresser la sensibilité du gradient aux minima locaux.

La fonction ci-dessous (voire **Figure 4.10**) dispose de deux minima à savoir un minimum local et un minimum global. Les algorithmes sans "momentum" restent bloqués au minimum local lorsque le gradient est trop faible de façon à ne pas dépasser celui-ci pour aller ver le minimum global qui est plus intéressant pour la fonction de perte; a contrario, ceux avec "momentum" rajoutent un certain élan à la descente de gradient.

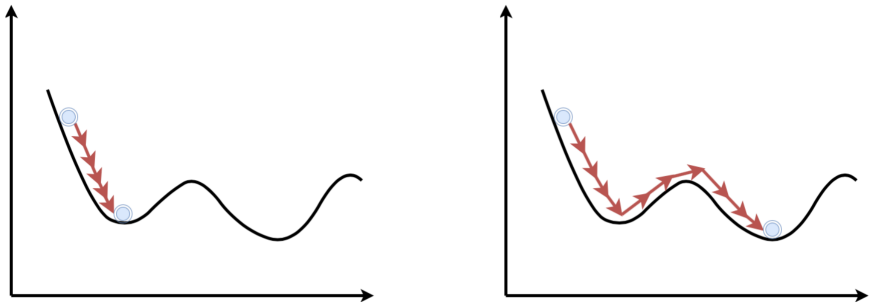

Figure 4.10 Illustration de la descente de gradient sans "momentum" (gauche) versus descente de gradient avec "momentum" (droite)

La mise à jour des paramètres pour ces types d'algorithmes est similaire de la descente de gradient classique (**Équation 11**) sauf qu'avant il faut faire une étape de calcul de la vélocité v:

$$v_{t+1} = \alpha v_t + \eta \nabla J(\theta_t) \tag{12}$$

Ensuite on effectue la mise à jour:

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{13}$$

- α est un hyperparamètre généralement fixé à 0.9

- v est la vélocité initialisée à 0.

En prenant en compte les gradients précédents, la convergence vers le minimum global est plus rapide et cela a également pour effet d'accélérer la descente de gradient dans la direction du minimum et de réduire les oscillations dans les autres directions. Le terme de quantité de mouvement agit donc comme un amortisseur qui atténue l'impact des gradients extrêmes et réduit la dépendance à l'égard de la taille du pas d'apprentissage η.

5 Méthodologie utilisée

5.1 Préparation des données

5.1.1 Type de données nécessaires

Pour entraîner et valider un modèle PINN, plusieurs types de données sont nécessaires:

Données d'entraînement: Ce sont les données utilisées pour ajuster les paramètres du modèle PINN. Elles comprennent les conditions aux limites et les points d'espace et de temps (**points de collocation**) pour lesquels la résiduelle de l'EDP est calculée. La **Figure 5.1** illustre la collecte aléatoire des points pour évaluer la fonction de perte pendant l'entraînement du modèle. Les points en **bleu** sont des points intérieurs du domaine de calcul, les points en **vert** représentent des noeuds de mesure pris aléatoirement dans les données acquises par simulation CFD et les autres points délimitant le domaine servent à évaluer les résidus au niveau des conditions limites.

Données de validation: Elles servent à évaluer la performance du modèle pendant et après l'entraînement. Elles peuvent être des solutions *analytiques, des solutions numériques de référence ou des mesures expérimentales* pour le problème d'écoulement considéré. Dans le cas présent, des **données de simulation CFD** réalisée avec OpenFOAM ont été utilisées à cette fin.

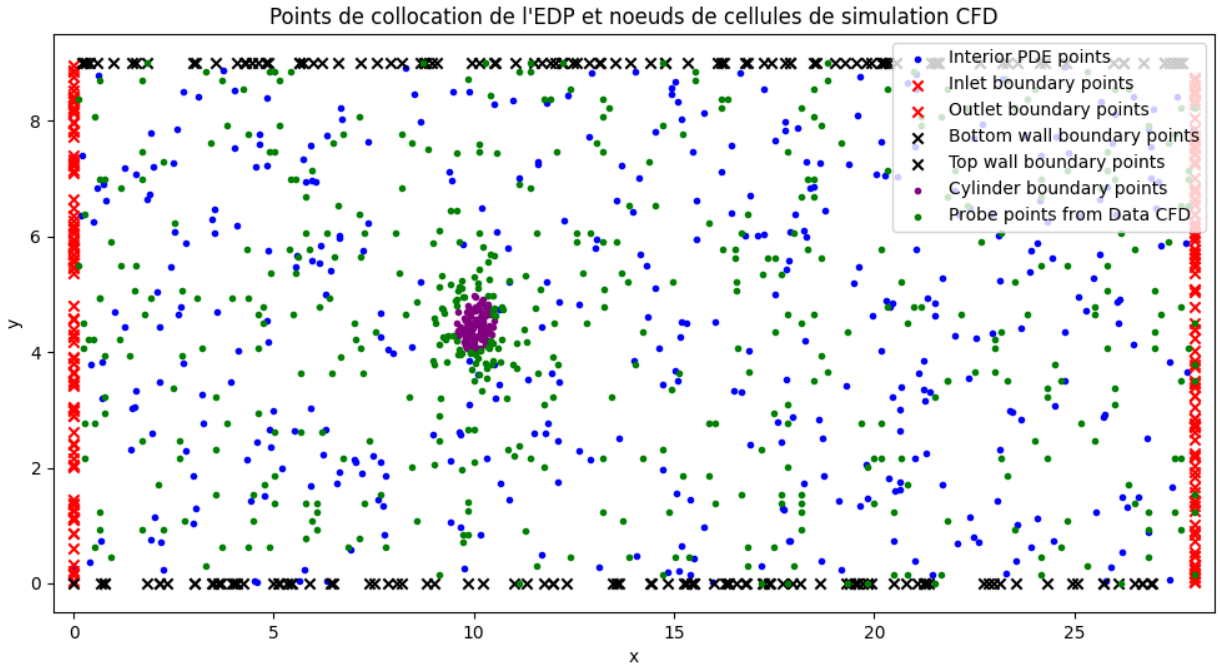

Figure 5.1 Points de collation pour fins d'évaluation de la fonction de perte du modèle PINN.

5.1.2 Collecte des données

Pour certains problèmes d'écoulement de fluides, il existe des solutions analytiques exactes ou approximatives qui peuvent être utilisées comme données d'entraînement et de validation. Par exemple, la solution de Couette pour l'écoulement entre deux plaques parallèles ou la solution de Stokes pour l'écoulement autour d'une sphère.

Pour les problèmes où aucune solution analytique n'est disponible, des simulations numériques, telles que les méthodes de volumes finis, d'éléments finis ou de différences finies, peuvent être utilisées pour générer des données d'entraînement et de validation.

Les mesures expérimentales peuvent également être utilisées comme données d'entraînement et de validation. Cependant, il est important de prendre en compte les incertitudes et les erreurs associées aux mesures expérimentales et, si nécessaire, d'effectuer une analyse d'incertitude pour évaluer l'impact de ces incertitudes sur la performance du modèle PINN.

Comme mentionné plus haut, au cours du projet la collecte de données consistait à réaliser une simulation CFD afin d'acquérir des données de validation. Le problème d'écoulement simulé est un **écoulement incompressible en 2D autour d'un cylindre à bas Reynolds ($Re = 20$) et en régime stationnaire**. Globalement les différentes étapes pour réaliser la simulation sont les suivantes:

- *la définition de la géométrie* : l'obstacle en l'occurence le cylindre en 2D, les murs (dessus et dessous), la paroi d'entrée et de sortie de l'écoulement.

- *la définition des conditions limites*: valeurs du champ de vitesse et de pression aux frontières du domaine de calcul

- *la discrétisation du domaine (maillage)*

- *le choix du solveur* : **simpleFoam**

- *le post-traitement des résultats obtenus*

5.2 Conception et configuration du modèle PINN

5.2.1 Définition des équations gouvernant l'écoulement et de la fonction de perte

Pour implémenter l'architecture du PINN, la première étape est de définir les équations aux dérivées partielles à résoudre en l'occurence celles de Navier-Stokes pour l'écoulement fluide autour d'un cylindre.

En considérant un espace bidimensionnel délimité par les coordonnées x et y dans le domaine $\mathbf{\Omega}$, un écoulement incompressible stable est caractérisé par les composantes de champ de vitesse horizontale (u) et verticale (v), ainsi que par la pression (p). Pour décrire le mouvement d'une particule de fluide de taille mésoscopique de manière adimensionnelle, les équations de Navier-Stokes peuvent être utilisées sous leur forme forte, comprenant les trois équations fondamentales du mouvement:

Équation de continuité

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{14}$$

Équations de quantité de mouvement (direction x et y)

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{R_e}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial p}{\partial x} \tag{15}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = \frac{1}{R_e}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - \frac{\partial p}{\partial y} \tag{16}$$

Conditions aux limites

Pour les problèmes d'écoulement, les conditions aux limites généralement appliquées sont les suivantes:

`Condition de non-glissement`: L'égalité de vitesse entre le fluide et la paroi est établie pour une paroi fixe, avec une vitesse de déplacement nulle, soit $(u, v) = 0$.

`Conditions de glissement`: Moins restrictives que la condition de non-glissement, elles permettent une vitesse tangentielle non nulle ($u_t \neq 0$), mais interdisent toute vitesse normale ($n \cdot u = 0$) pour respecter le principe de *non-pénétration du fluide* dans la paroi.

`Condition de symétrie` : Appliquée pour diviser le domaine en deux parties, elle impose un gradient nul de la composante tangentielle de la vitesse ($\nabla u_t = 0$) et de la pression ($\nabla p = 0$), ainsi qu'une non-pénétration pour la vitesse normale $n \cdot u = n_x u + n_y v = 0$.

`Condition d'écoulement uniforme` : Spécifiée pour les conditions d'entrée, elle impose une vitesse constante $(u, v) = (u_{BC}, v_{BC})$.

`Condition de pression` : Une valeur de pression donnée (P_o) ou une valeur de pression moyenne peut être imposée pour les conditions de sortie.

Fonction de perte

Comme énoncé à la **section 4.3.5**, la fonction de perte est le socle d'un réseau neuronal artificiel dans le sens où elle permet de mesurer la différence entre les valeurs prédites par le réseau et les valeurs réelles. Dans le cas présent, il s'agit d'évaluer l'EDP, ses conditions limites et également les résidus avec quelques données provenant de la simulation CFD. Le type d'erreur calculée est l'**erreur quadratique moyenne** (EQM ou "MSE" en anglais, voire **l'équation 6**).

Les résidus de l'EDP et ceux des points de mesure (simulation) sont calculés de la façon suivante:

$$R_1 = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{17}$$

$$R_2 = \frac{1}{R_e}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial p}{\partial x} - u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} = 0 \tag{18}$$

$$R_3 = \frac{1}{R_e}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - \frac{\partial p}{\partial y} - u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} = 0 \tag{19}$$

$$\mathscr{L}_{edp} = \frac{1}{N_{x_{in},y_{in}\in\mathbf{\Omega_{edp}}}}\sum|R_1|^2 + \frac{1}{N_{x_{in},y_{in}\in\mathbf{\Omega_{edp}}}}\sum|R_2|^2 + \frac{1}{N_{x_{in},y_{in}\in\mathbf{\Omega_{edp}}}}\sum|R_3|^2 \tag{20}$$

$$\mathscr{L}_{BC} = \frac{1}{N_{x_{bc},y_{bc}\in\mathbf{\Omega_{BC}}}}\sum|\tilde{\mathbf{q}}(\mathbf{x_{bc}},\mathbf{y_{bc}}) - \mathbf{q}(\mathbf{x_{bc}},\mathbf{y_{bc}})|^2 \tag{21}$$

$$\mathscr{L}_m = \frac{1}{N_m}\sum|\tilde{\mathbf{q}}(\mathbf{x_m},\mathbf{y_m}) - \mathbf{q}(\mathbf{x_m},\mathbf{y_m})|^2 \tag{22}$$

$$\mathscr{L}_{total} = \mathscr{L}_{edp} + \mathscr{L}_{BC} + \mathscr{L}_m \tag{23}$$

Avec:

R_1 : le résidu de l'EDP sur l'équation de continuité

R_2 : le résidu sur l'équation de quantité de mouvement en direction x.

R_3 : le résidu sur l'équation de quantité de mouvement en direction y.

$\mathscr{L}_{edp}$: la perte totale associée à l'équation de Navier-Stokes en incompressible 2D.

$\mathscr{L}_{BC}$: le résidu sur les conditions aux limites du domaine de calcul.

$\mathscr{L}_m$: le résidu sur les données acquises par simulation numérique.

$\tilde{\mathbf{q}}(\mathbf{x_{bc}},\mathbf{y_{bc}})$: vecteur des champs u, v, p prédits par le réseau de neurones.

$\mathbf{q}(\mathbf{x_{bc}},\mathbf{y_{bc}})$: vecteur des champs réels u, v, p aux conditions limites.

$\tilde{\mathbf{q}}(\mathbf{x_m},\mathbf{y_m})$: vecteur des champs u, v, p prédits par le réseau de neurones avec des points pris aléatoires des noeuds issus de la simulation CFD.

$\mathbf{q}(\mathbf{x_m},\mathbf{y_m})$: vecteur des champs u, v, p calculés par simulation numérique.

$N_{x_{in},y_{in}\in\mathbf{\Omega_{edp}}}$: le nombre de points de collocation collectés à l'intérieur du domaine de calcul.

$N_{x_{bc},y_{bc}\in\mathbf{\Omega_{BC}}}$: le nombre de points de collocation collectés aux frontières du domaine de calcul.

N_m : le nombre de points pris aléatoirement parmi les données obtenues par simulation.

5.2.2 Architecture du réseau de neurones

Une représentation précise du problème de l'écoulement des fluides nécessite un réseau de neurones informé par la physique (PINN). L'architecture du réseau doit refléter la complexité du problème et assurer une convergence régulière et rapide pendant l'entraînement. Considérations relatives à l'architecture du réseau de neurones :

`Neurones et couches`

Un réseau neuronal profond comporte plusieurs couches avec un certain nombre de neurones. Les réseaux de neurones profonds ont généralement une entrée, plusieurs étages cachés et une sortie. Le nombre de couches cachées et de neurones dans chaque couche dépend de la complexité du problème et doit être déterminé par essais et erreurs et par des connaissances préalables.

`Entrées et sorties`

Les entrées du réseau de neurones correspondent aux coordonnées spatiales (x, y). Les neurones doivent correspondre aux variables de la couche d'entrée. Les sorties du réseau représentent des variables d'intérêt, telles que les composantes u et v du champ de vitesse du fluide et la pression p (voire **figure 5.2**). Les neurones doivent correspondre aux variables de la couche de sortie.

`Connectivité et poids`

Le poids synaptique relie les neurones d'une couche à l'autre. Ces poids ajustent les paramètres du réseau pendant l'apprentissage. La connectivité peut être totale, c'est-à-dire que chaque neurone d'une couche est connecté à tous les neurones de la couche suivante, ou partielle, en fonction de la structure du problème. Dans l'architecture actuelle, il s'agit d'un **réseau de neurones pleinement connecté**.

`Fonctions d'activation`

Les fonctions non linéaires appliquées aux sorties des neurones introduisent une non-linéarité dans le modèle. Sigmoïde, tangente hyperbolique (tanh) et ReLU sont des fonctions d'activation courantes. La fonction d'activation peut affecter les performances et la convergence du modèle. La fonction $tanh$ est celle qui a été utilisée dans l'architecture du PINN grâce à ses performances pour la résolution de problèmes de régression multidimensionnelle.

Le diagramme de la **Figure 5.2** illustre comment ces différents éléments de l'architecture sont connectés les uns aux autres. On a d'abord une couche d'entrée (x, y), qui est ensuite connectée aux différentes couches cachées à l'intérieur desquelles sont utilisées les fonctions d'activation $tanh$. Enfin à la sortie du réseau neuronal, la couche de sortie se compose les champs prédits u, v et p dont les dérivées sont calculées avec la méthode AD afin de les intégrer dans la fonction de perte.

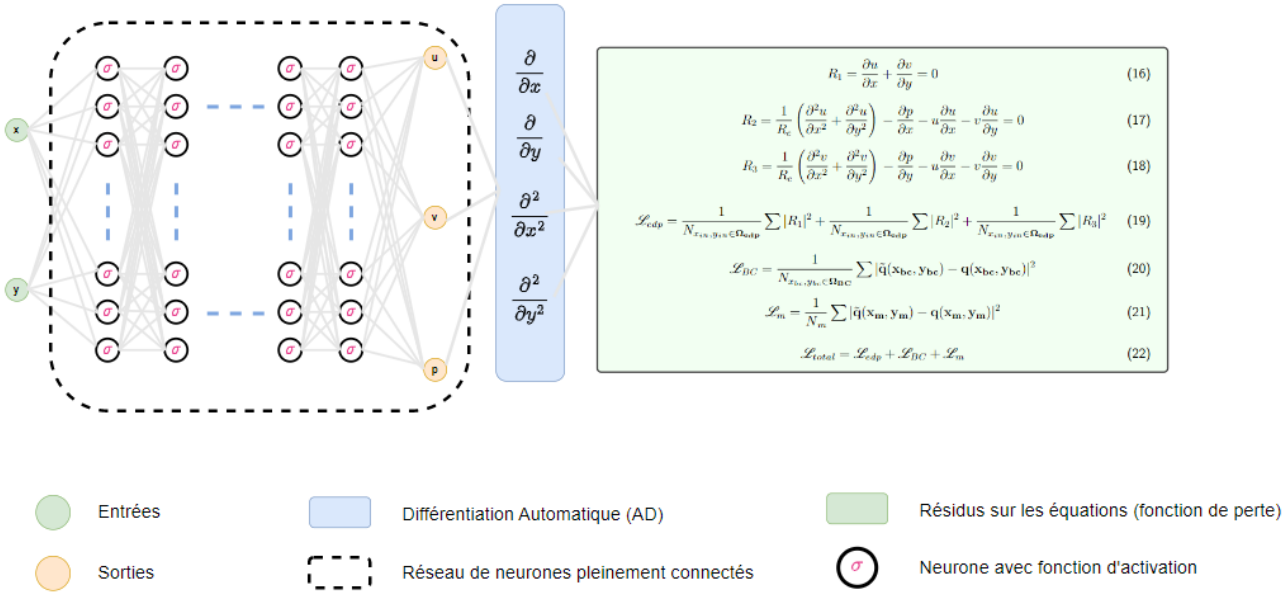

Figure 5.2 Diagramme architectural du modèle PINN développé.

5.2.3 Méthodes d'optimisation

L'apprentissage du modèle PINN consiste à ajuster le poids et le biais du réseau afin de minimiser la fonction de coût, qui comprend la contribution des résidus EDP et des conditions limites. Pour mettre à jour les paramètres du réseau sur la base du gradient de coût, des méthodes d'optimisation numérique sont utilisées.

`Taille du réseau:` Le choix de la taille, c'est-à-dire le nombre de couches cachées et le nombre de neurones par couche, se fait généralement de quatre façons à savoir la recherche **manuelle**, la recherche par **grille**, la recherche **aléatoire** et la recherche **avancée avec des librairies dédiées**. Les trois techniques exceptée celle par grille (à cause du coût computationnel important qu'elle nécessite) ont été expérimentées dans ce projet.

`Algorithmes d'optimisation` : SGD, "momentum", Adagrad, Adam et d'autres méthodes de second ordre comme Newton-Raphson ou le gradient conjugué peuvent être utilisés pour entraîner les réseaux de neurones. L'algorithme qui s'est montré efficace et a été utilisé au long du projet est **Adam**.

`Taux d'apprentissage` : L'hyperparamètre contrôle la taille des mises à jour des paramètres du réseau de neurones par itération de l'algorithme d'optimisation. Il existe également plusieurs méthodes de choix du taux d'apprentissage: un taux fixe (constant) tout au long de l'apprentissage ou un taux dynamique décroissant (la décroissance peut être linéaire ou exponentielle). La formule implémentée est une décroissance exponentielle à travers les équations suivantes:

$$a = \frac{N}{2} \tag{24}$$

$$\beta = \log(\frac{\eta_{init}}{\eta_{min}}) \tag{25}$$

$$\eta_i = max(\eta_{_{init}} e^{\frac{-\beta i}{a}}, \eta_{min}) \tag{26}$$

Avec:

N : le nombre maximal d'itérations

η_{min} : le taux minimal d'apprentissage

η_{init} : le taux d'apprentissage initial

i : l'époque actuelle (i^{me} itération).

5.3 Entraînement et validation du modèle

L'entraînement implique l'utilisation d'une fonction de perte combinant les erreurs de prédiction et les erreurs de contraintes physiques, ainsi que le choix d'un optimiseur et d'hyperparamètres adaptés, tels que le taux d'apprentissage, la taille de l'échantillon et le nombre d'époques. Le modèle est entraîné sur un ensemble de données d'entraînement en utilisant l'autodifférentiation et la rétropropagation pour ajuster les poids et les biais. Ce processus permet au modèle d'apprendre à représenter les champs d'écoulement fluide de manière précise et cohérente avec les principes physiques régissant le problème étudié. Le **pseudo-code suivant** résume l'algorithme utilisé pour entraîner le modèle.

Entrée: Hyperparamètres du réseau (taille, σ) et
Optimisation (méthode, taux d'apprentissage η, la limite d'entraînement)
Résultat:
Construire la structure du réseau PINN $x, \theta \Rightarrow y^{MLP}(x;\theta)$;
Initialiser les paramètres θ;
Préparer les données (points de collocation et données de simulation) (x, y);
Construire la fonction de perte $(x, y), \theta \Rightarrow \mathscr{L}_{tot}[(x, y), \theta]$;
while le critère d'arrêt non atteint
do
Préparer l'échantillon;
Calculer la perte;
Calculer les gradients de la fonction de perte $\frac{\partial \mathscr{L}}{\partial \theta}$;
Mettre à jour les paramètres θ;
end

Après la phase d'entraînement, il faut évaluer les performances du réseau de neurones informé par la physique (PINN). Cette évaluation permet de vérifier la capacité du modèle à généraliser sur l'ensemble du domaine de calcul et d'identifier d'éventuelles lacunes. Des métriques appropriées, telles que **l'erreur**

quadratique moyenne et l'erreur absolue moyenne, sont utilisées pour mesurer la qualité des prédictions du modèle.

6 Résultats et discussion

Afin de résoudre l'équation de Navier-Stokes décrivant l'écoulement laminaire autour d'un cylindre, une approche **pas-à-pas** a été utilisée: construire dans un premier temps un réseau de neurones afin de résoudre des équations différentielles moins complexes (l'occurence les **équations de diffusion 1D et de convection-diffusion 1D**) dont une solution de référence existe et ensuite développer ce réseau pour résoudre le problème final d'écoulement autour du cylindre et comparer le résultat obtenu à travers le PINN avec des données de simulations réalisées avec OpenFOAM.

6.1 EDP Diffusion 1D

Cette équation décrit comment la température u varie dans le temps t et l'espace x, en fonction du taux de diffusion α. L'équation de la chaleur est généralement écrite comme suit:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \tag{27}$$

où $u(x,t)$ est la température à la position x et au temps t, et α est un coefficient de diffusion qui dépend des propriétés du matériau. L'équation de la chaleur est une équation aux dérivées partielles linéaire du second ordre, qui décrit la diffusion de la chaleur dans un milieu homogène. Cette équation est utilisée pour résoudre de nombreux problèmes de diffusion de la chaleur en une dimension, tels que la propagation d'une onde de chaleur dans une barre métallique, la diffusion de chaleur dans une plaque isolante, ou encore la diffusion de la chaleur dans un milieu poreux.

L'EDP résolue dans le présent projet est celle de la diffusion à une dimension avec un **terme source** $g(x,t)$:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + g(x,t) \tag{28}$$

Avec :

$$g(x,t) = -e^{-t}\left(sin(\pi x) - \pi^2 sin(\pi x)\right) \tag{29}$$

Domaine de calcul

$$\begin{cases} x \in [-1;1] \\ t \in [0;1] \end{cases} \tag{30}$$

Condition initiale

$$u(x,0) = sin(\pi x) \tag{31}$$

Conditions limite (Dirichlet)

$$\begin{cases} u(-1,t) = 0 \\ u(1,t) = 0 \end{cases} \tag{32}$$

Réseau de neurones différentiable

L'approche des PINNs consiste à transformer l'équation aux dérivées partielles qui gouverne le porblème physique à l'étude sous forme de réseau de neurones. En appliquant ce principe on obtient:

$$NN(x,t) \simeq u(x,t) \Rightarrow \begin{cases} \frac{\partial NN(x,t)}{\partial t} \\ \frac{\partial^2 NN(x,t)}{\partial x^2} \end{cases} \tag{33}$$

L'**équation 8** devient:

$$\frac{\partial NN}{\partial t} = \frac{\partial^2 NN}{\partial x^2} - e^{-t}\Big(sin(\pi x) - \pi^2 sin(\pi x)\Big) \tag{34}$$

Pénalisation (fonction de perte) de l'EDP

La fonction de perte pour le PINN est calculée à travers le résidu de l'équation différentielle:

$$\mathscr{L}(x,t) = \frac{\partial NN}{\partial t} - \frac{\partial^2 NN}{\partial x^2} + e^{-t}\Big(sin(\pi x) - \pi^2 sin(\pi x)\Big) \tag{35}$$

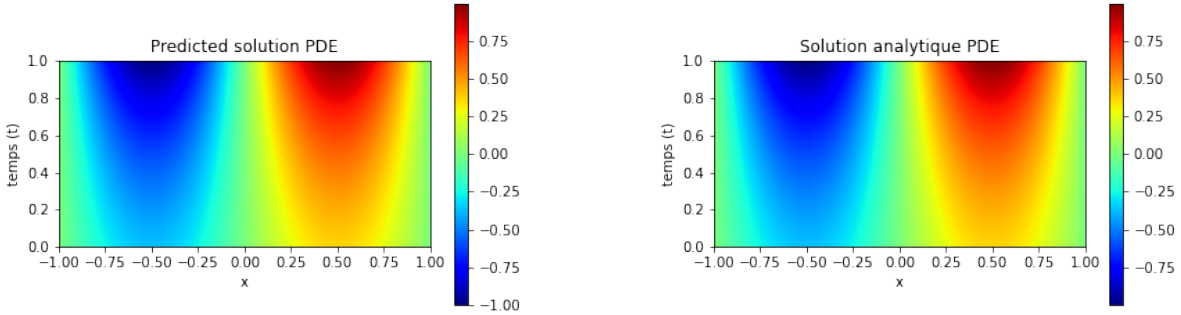

Figure 6.1 Résultats obtenus avec le PINN pour la résolution du problème de diffusion (la fonction u prédite à gauche) et (la solution de référence à droite) avec $N_{edp} = 4000$, $N_{BC} = 2000$ et $N_{IC} = 2000$

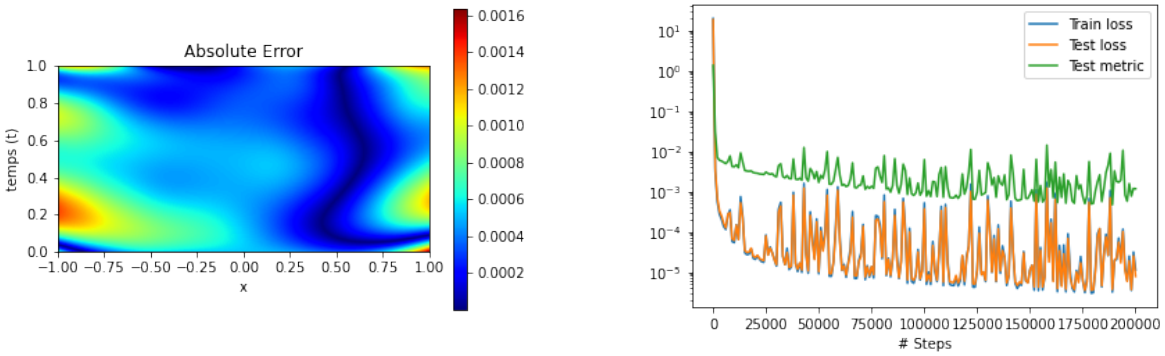

Figure 6.2 Erreur absolue de prédiction (image à droite) des résultats consignés à la figure 6.1 et l'évolution des résidus de la fonction de perte pour les données de validation et d'entraînement (image à gauche).

Les **Figures 6.1 et 6.2** relatent les résultats obtenus en utilisant le modèle PINN comme un solveur d'EDP c'est-à-dire sans recourir à des données expérimentales ou de simulation. La taille du réseau pour ce problème est de 3 x 64 en l'occurrence 64 neurones par couche et 3 couches cachées; la fonction d'activation utilisée est la tangente hyperbolique ($tanh$). Avec ce réseau, les résidus de perte atteignent un plateau ($moyenne$ avec un petit écart-type) de l'ordre de 10^{-5}. L'erreur absolue de prédiction est de 0.0016; ce qui est assez précis pour le cas étudié. La courbe de résidus sur les données d'entraînement (en bleu) est parfaitement superposée sur celle des résidus sur les données de validation (en orange); ce qui implique qu'il n'y a pas de problème de sur-apprentissage ou de sous-apprentissage. C'est également l'une des forces des PINNs comparativement aux réseaux de neurones classiques qui nécessitent des techniques de régularisation pour pallier aux problèmes de sur-apprentissage. La courbe en couleur verte représente l'erreur relative en fonction des différentes itérations: elle décroît et atteint un maximum de l'ordre de 10^{-3}.

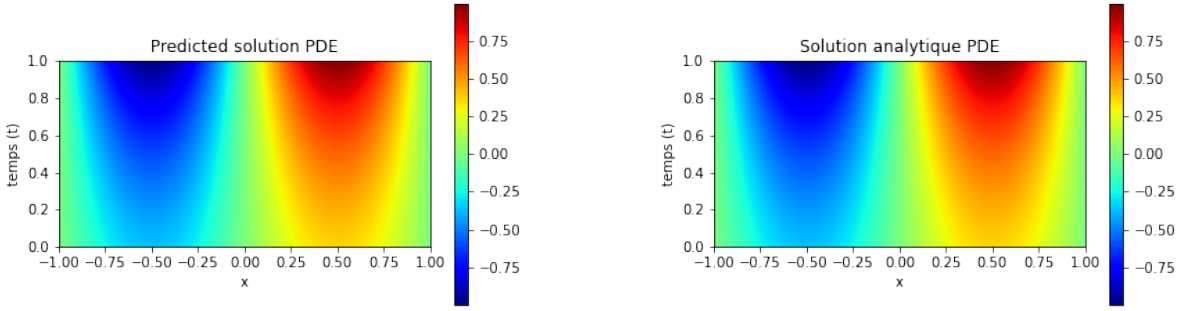

Figure 6.3 Résultats obtenus après optimisation avec le PINN pour la résolution du problème de diffusion (la fonction u prédite à gauche) et (la solution de référence à droite)

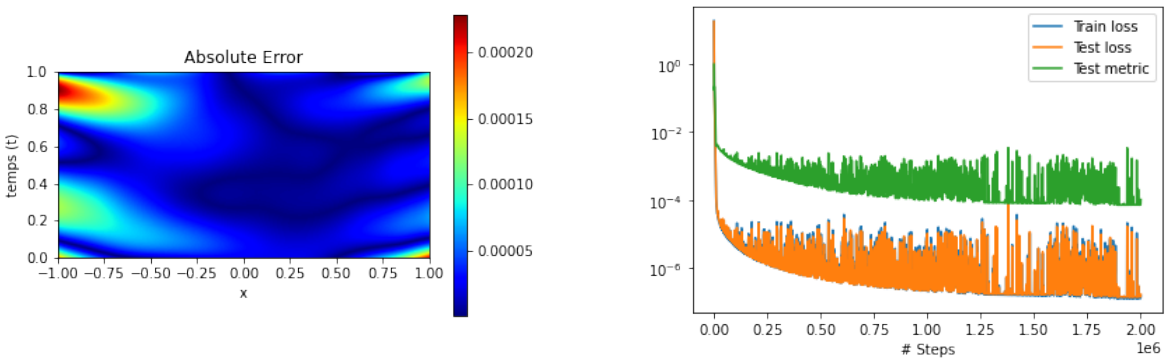

Figure 6.4 Erreur absolue de prédiction des résultats consignés à la figure 6.3 (droite) et l'évolution des résidus de la fonction de perte pour les données de validation et d'entraînement (gauche).

Pour illustrer l'analogie entre les méthodes numériques classiques avec discrétisation (maillage), le meilleur modèle avec une erreur plus petite que celle des résultats précédents a été obtenu en augmentant le nombre de points collectés ($N_{edp} = 20000, N_{BC} = 8000, N_{IC} = 8000$) et en diminuant le pas de taux d'apprentissage ($\eta = 10^{-5}$). Les **Figures 6.3 et 6.4** illustrent les résultats de cette optimisation: les résidus ont diminué davantage de 10^{-5} à 10^{-6} et l'erreur absolue moyenne de $1.6\ 10^{-3}$ à $2\ 10^{-4}$. Par contre cela implique un temps d'entraînement plus long.

6.2 EDP Convection-Diffusion 1D

Les problèmes de convection-diffusion en dynamique des fluides sont des problèmes de transport de quantités scalaires, comme la température, la concentration ou la densité, dans un fluide en mouvement. Ces phénomènes sont décrits par une équation aux dérivées partielles qui combine les effets de la convection et de la diffusion.

L'équation de convection-diffusion pour un scalaire ϕ dans un fluide incompressible 1D est donnée par :

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = D\frac{\partial^2 \phi}{\partial x^2} \tag{36}$$

où u est la vitesse du fluide dans la direction x, D est le coefficient de diffusion et $\frac{\partial}{\partial x}$ représente la dérivée partielle par rapport à x. La première partie de l'équation représente la convection, qui décrit comment la quantité scalaire est transportée par le mouvement du fluide. La deuxième partie représente la diffusion, qui décrit comment la quantité scalaire est diffusée par les gradients de concentration ou de température.

Condition initiale

Au temps initial la solution de l'équation est une fonction gaussienne:

$$\phi(x, 0) = Ae^{-(\frac{x}{\sigma})^2} \tag{37}$$

`Paramètres de la fonction gaussienne`

$A = 1$ et $\sigma = 0.05$

Conditions limites

Gauche (Dirichlet) : $\phi(x = 0, t) = f(t)$

Droite (Neumann) : $\frac{\partial \phi(x=1,t)}{\partial x} = 0$

Solution de référence

$$\phi(x, t) = A\left(\sqrt{\frac{\pi\sigma^2}{\pi(\sigma^2 + 4\nu t)}}e^{\left(\frac{-(x-ct)^2}{\sigma^2+4\nu t}\right)}\right) \tag{38}$$

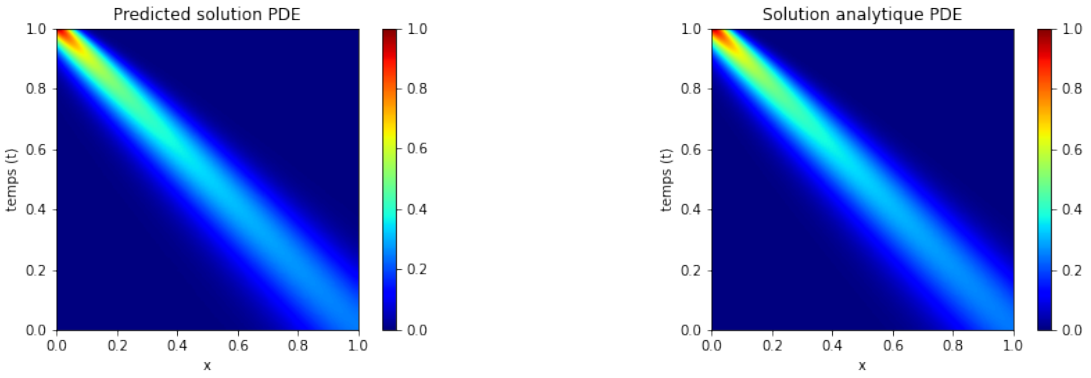

Figure 6.5 Résultats obtenus après optimisation avec le PINN pour la résolution du problème de convection-diffusion (la fonction ϕ prédite à gauche) et (la solution de référence à droite)

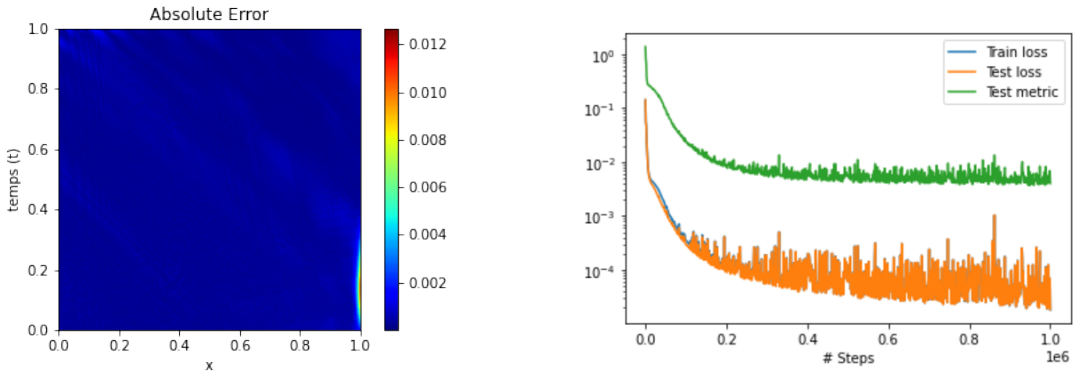

Figure 6.6 Erreur absolue de prédiction (image à droite) des résultats consignés à la figure 6.5 et l'évolution des résidus de la fonction de perte pour les données de validation et d'entraînement (image à gauche).

Les **Figures 6.5 et 6.6** illustrent les résultats obtenus pour résoudre l'équation de convection-diffusion linéaire, c'est-à-dire avec une vitesse constante $u = 1$. En utilisant un taux d'apprentissage de 10^{-4} et des points de collocation de $N_{edp} = 6000$, $N_{BC} = 3000$ et $N_{IC} = 3000$, les résidus de perte s'élèvent à 1.22×10^{-5}. Par ailleurs, l'erreur absolue moyenne obtenue est de 0.012.

Ces résultats mettent, en sus de ceux du problème de diffusion, en évidence la performance du modèle PINN dans la résolution d'équation aux dérivées partielles. Le modèle parvient à fournir des prédictions précises et cohérentes avec les contraintes physiques imposées.

6.3 EDP Navier-Stokes: Écoulement incompressible autour d'un cylindre

La présente section présente les résultats obtenus en combinant quelques données avec le modèle PINN développé pour résoudre les problèmes précédents.

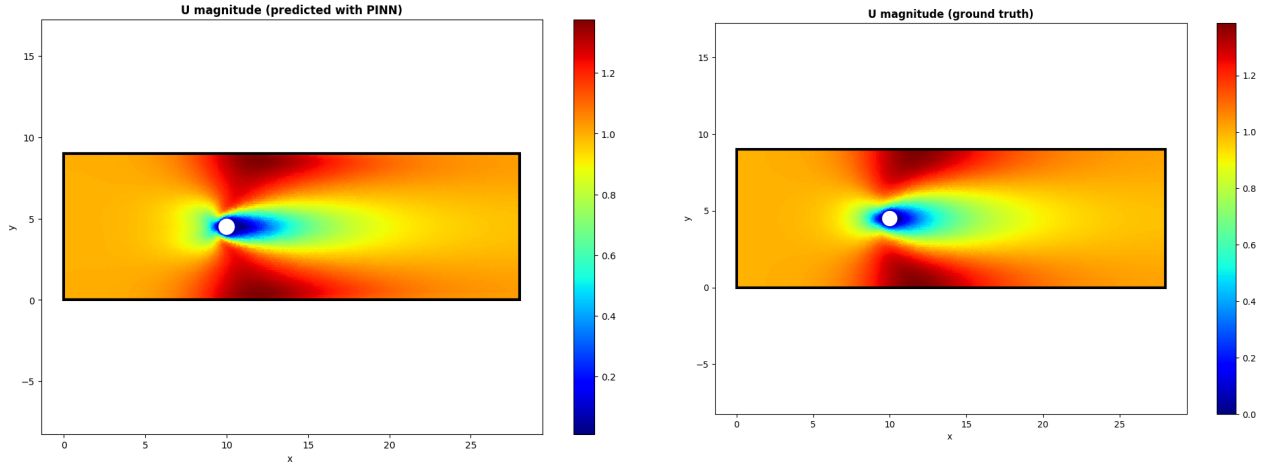

Figure 6.7 Résultats obtenus en combinant l'utilisation de quelques données $N_m = 500$ et le modèle PINN pour prédire les champs d'écoulement (la magnitude du champ de vitesse U prédite à droite et le champ de vitesse obtenu par simulation à gauche).

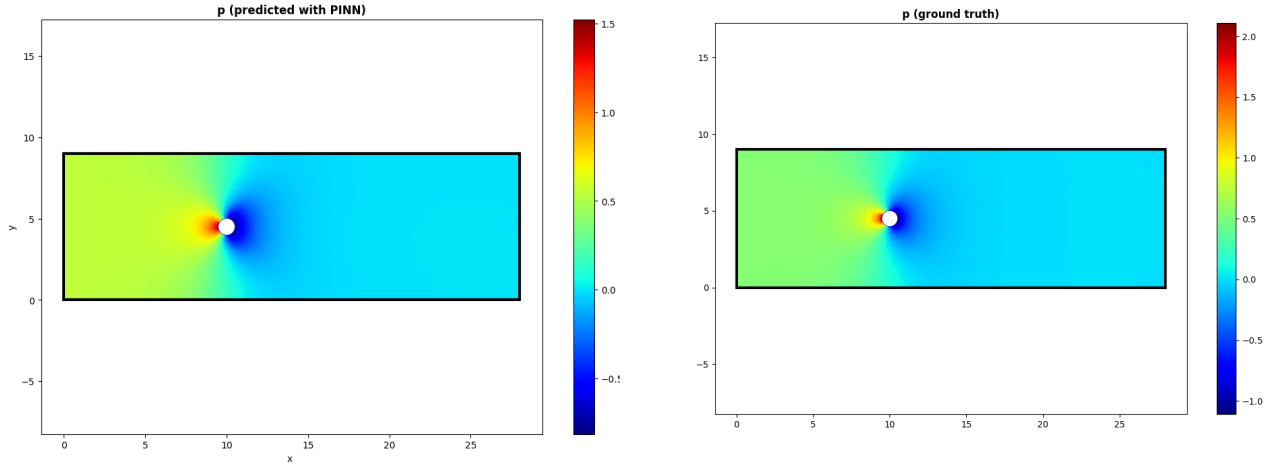

Figure 6.8 Résultats obtenus en combinant l'utilisation de quelques données $N_m = 500$ et le modèle PINN pour prédire les champs d'écoulement (la pression p prédite à droite et le champ de pression obtenu par simulation à gauche).

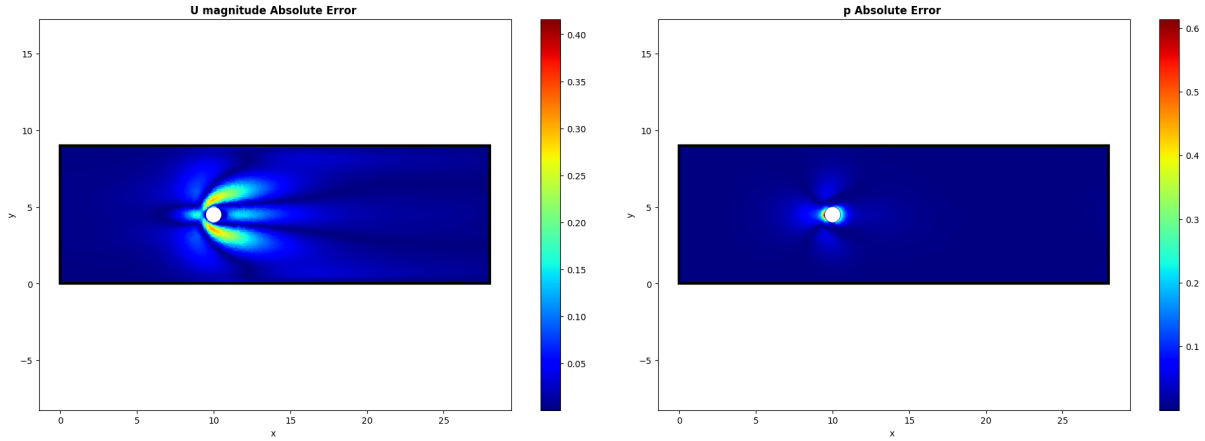

Figure 6.9 Erreurs absolues moyennes de prédiction (à droite l'erreur sur la vitesse e_u et à gauche celle sur la pression e_p)

On peut observer à travers les **Figures 6.7 à 6.9** qu'avec quelques données l'architecture PINN est capable de reconstruire significativement l'entièreté de la solution sur tout le domaine de calcul. Les erreurs absolues de prédiction pour la vitesse et la pression sont respectivement 0.4 et 0.6. Les résultats des différents modèles testés sont récapitulés dans le tableau ci-après:

Sommaire des résultats				
# NN	**EDP**	**Description**	**Résidus $\mathscr{L}$**	**Erreur absolue**
1	Diffusion	Sans donnée	$4.45\ 10^{-6}$	0.0002
2	Convection-Diffusion	Sans donnée	$1.22\ 10^{-5}$	0.012
3	Navier-Stokes	NN standard avec 60% de données	$2.89\ 10^{-5}$	$e_u = 0.02$ $e_v = 0.015$ $e_p = 0.0175$
4	Navier-Stokes	Sans donnée	$1.42\ 10^{-3}$	$e_u = 0.6$ $e_v = 0.55$ $e_p = 1.6$
5	Navier-Stokes	PINN + quelques données $N_m = 500$	$6.89\ 10^{-4}$	$e_u = 0.4$ $e_v = 0.35$ $e_p = 0.6$

Tableau 6.1 Récapitulatif des résultats obtenus avec les cas de test réalisés

À travers le tableau 6.1 on constate que le modèle est plus performant pour les cas de test $NN\ \#1, NN\ \#2$ et $NN\ \#3$ respectivement pour la résolution des équations de diffusion, de convection-diffusion et de Navier-Stokes avec que des données (réseau neuronal classique). Ce résultat n'est pas surprenant vu la complexité des équations couplées (vitesse-pression) et aussi la probable propagation d'erreurs liées à la discrétisation des volumes finis. L'analyse des résultats obtenus révèle trois points importants:

La **convergence des PINNs** est facilitée en régénérant aléatoirement le maillage d'entrée (points de collocation et de frontière) après quelques itérations d'entraînement. Cette approche permet au réseau de neurones de bénéficier d'une grande variété de points d'échantillonnage, rendant les PINNs plus robustes et efficaces sur le plan computationnel. Cette méthode permet d'atteindre les mêmes objectifs

qu'avec un maillage très fin utilisé dans les éléments ou volumes finis.

La **prédiction de la pression** demeure difficile avec la formulation adoptée pour les PINNs de l'écoulement des fluides (Navier-Stokes). Il pourrait être intéressant d'explorer différentes formulations d'équations régissant l'écoulement ou des architectures de réseaux de neurones dédiées à chaque variable (composantes de vitesse u, v et pression p).

Un **entraînement excessif** pour obtenir des valeurs de perte plus faibles ne conduit pas à un surajustement des données d'entraînement, grâce à la présence du résidu de l'EDP dans la fonction de perte. En intégrant l'information relative à la physique (équations différentielles régissant le système), la perte pour les données d'entraînement et de validation reste du même ordre de grandeur tout au long du processus d'entraînement. Cela s'explique par le fait que le réseau de neurones apprend non seulement la cartographie de la solution pour un ensemble donné de points de données, mais aussi les instances souhaitées des EDPs pour les ensembles de données d'entraînement et de validation.

7 Conclusion

En conclusion, ce projet de recherche visait à résoudre l'équation de Navier-Stokes pour un écoulement de fluide autour d'un cylindre en utilisant un modèle de réseau de neurones informé par la physique (PINN) et à le valider avec des données acquises par simulation numérique. Les résultats obtenus montrent une plus grande performance du réseau dans la résolution des équations de diffusion et de convection-diffusion sans aucune donnée, comparativement aux équations de Navier-Stokes. Néanmoins, le sens physique n'est pas perdu et la reconstruction prédictive des champs d'écoulement est satisfaisante; il convient également de prendre en compte la probable propagation d'erreurs liées à la discrétisation dans les données obtenues par simulation CFD.

En ce qui concerne les perspectives de travaux futurs, il convient de mentionner que les PINNs présentent certaines limitations pour des problèmes à petite échelle, notamment l'effort d'implémentation que cela implique. Cependant, il serait intéressant d'explorer l'application des PINNs à d'autres problèmes en mécanique des fluides ou d'autres domaines de l'ingénierie comme la prédiction de la fatigue de composantes machine, ainsi que d'étudier les perspectives offertes par les opérateurs neuronaux informés par la physique (PINOs) pour résoudre des familles d'équations différentielles.

De plus, des recherches pourraient être menées pour améliorer la robustesse du modèle face aux données bruitées et pour optimiser davantage les hyperparamètres et l'architecture du réseau de neurones. Il serait également pertinent d'examiner l'intégration de méthodes d'apprentissage non supervisé et de techniques de réduction de dimensionnalité (avec l'usage des encodeurs et décodeurs) pour réduire encore davantage la quantité de données nécessaires à l'entraînement du modèle. Enfin, le développement d'outils logiciels, d'écosystèmes ou de plateformes permettant une intégration facile des PINNs dans les processus de conception et d'optimisation en ingénierie contribuerait à étendre l'application de cette technologie prometteuse à une variété de secteurs.

References

1. Williamson, C. *Vortex Dynamics in the cylinder wake,* [Available online] https://doi.org/10.1146/annurev.fl.28.010196.002401, 1996.
2. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics, [Available online],* https://www.sciencedirect.com/science/article/abs/pii/S0021999118307125 **378,** 686–707 (2019).
3. Yibo, Y. & Perdikaris, P. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics* **394,** Pages 136 –152, [Available online], https://doi.org/10.1016/j.jcp.2019.05.027 (2019).
4. Alhuwaider, S. *Compressible flow one-shot prediction using PINNs* Thesis (King Abdullah University of Science and Technology). https://repository.kaust.edu.sa/bitstream/handle/10754/676833/Shyma_KAUST_Thesis%5C%20(4).pdf?sequence=5.
5. Yohai, B.-S., Hutzenthaler, M., Jentzen, A. & Salimova, D. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences* **116,** 15344–15349, [Available online] https://arxiv.org/pdf/1808.04930.pdf (2019).
6. Li, Z. *et al.* Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895,* [Available online], https://arxiv.org/abs/2010.08895v3 (2020).
7. Lu, L. *et al.* Physics-informed neural networks with hard constraints for inverse design. *Available online,* https://arxiv.org/abs/2102.04626 (2021).
8. Rao, C., Sun, R. & Liu, Y. Physics informed deep learning for computational elastodynamics without labeled data. *preprint arXiv:2006.08472* (2020).
9. Secci, D., Molino, L. & Zanini, A. Contaminant source identification in groundwater by means of artificial neural network, https://www.researchgate.net/publication/362152224_Contaminant_source_identification_in_groundwater_by_means_of_artificial_neural_network (2022).
10. Stiasny, J., Misyris, G. S. & Chatzivasileiadis, S. *Physics-Informed Neural Networks for Non-linear System Identification for Power System Dynamics* in *2021 IEEE Madrid PowerTech* (2021), 1–6.
11. Sirignano, J. & Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* **375,** 1339–1364 (2020).
12. Zhu, Y., Zabaras, N., Koutsourelakis, P.-S. & Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics* **394,** 56–81 (2019).
13. Li, X., Chen, W. & Karniadakis, G. E. A general purpose physics-informed deep learning framework for solving boundary value problems. *Journal of Computational Physics* **424,** 109851 (2021).
14. Raynaud, G., Houde, S. & Gosselin, F. P. ModalPINN: An extension of physics-informed Neural Networks with enforced truncated Fourier decomposition for periodic flow reconstruction using a limited number of imperfect sensors. *Journal of Computational Physics* **464,** 111271. https://doi.org/10.1016%5C%2Fj.jcp.2022.111271 (Sept. 2022).
15. Beck, A., Flad, D. & Munz, C.-D. Deep Neural Networks for Data-Driven LES Closure Models. *Journal of Computational Physics* **424,** https://www.researchgate.net/publication/335430643_Deep_Neural_Networks_for_Data\bibrangedashDriven_LES_Closure_Models (2019).
16. Han, J. & Lee, Y. *Hierarchical Learning to Solve Partial Differential Equations Using Physics-Informed Neural Networks* 2022. arXiv: 2112.01254 [cs.LG].
17. Hwang, H. J., Perdikaris, P. & Karniadakis, G. E. Robust and scalable physics-informed deep learning with noisy gradient signals. *arXiv preprint arXiv:2105.00794* (2021).
18. Bilionis, I. & Zabaras, N. Multi-output separable Gaussian process: Towards an efficient, fully Bayesian paradigm for uncertainty quantification. *Journal of Computational Physics* **354,** 423–445 (2018).
19. Peherstorfer, B. & Willcox, K. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering* **306,** 196–215 (2016).

20. Wikipedia. *Frank Rosenblatt* Page web consultée sur https://fr.wikipedia.org/wiki/Frank_Rosenblatt le (2023/03/12).
21. TowardsDataScience. *Importance and reasoning behind activations functions,* Page web consultée sur https://towardsdatascience.com/the-importance-and-reasoning-behind-activation-functions-4dc00e74db41 à la date du 2023/02/23.
22. Medium. *Introduction to Different activation functions for Deep Learning,* Page web consultée sur https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092 à la date du 2023/02/23.
23. Laue, S. & Rüping, S. A comparison of automatic differentiation and numeric differentiation in machine learning. *Machine Learning* **108,** 575–595 (2019).
24. Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research* **18,** 1–43 (2018).
25. Medium. *Overview of loss functions for machine learning* Page web consultée sur https://medium.com/analytics-vidhya/overview-of-loss-functions-for-machine-learning-61829095fa8a#:~:text=Loss%20functions%20take%20predictions%20and,sample%20of%20data%20and%20predictions. à la date du 2023/02/23.
26. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks* in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (2010), 249–256.
27. He, K., Zhang, X., Ren, S. & Sun, J. *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* in *Proceedings of the IEEE International Conference on Computer Vision* (2015), 1026–1034.
28. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *ImageNet classification with deep convolutional neural networks* in *Advances in Neural Information Processing Systems* (2012), 1097–1105.
29. Sutskever, I., Martens, J., Dahl, G. & Hinton, G. *On the importance of initialization and momentum in deep learning* in *International Conference on Machine Learning* (2013), 1139–1147.
30. Bottou, L., Curtis, F. E. & Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Review* **60,** 223–311 (2018).
31. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
32. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).
33. Géron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (O'Reilly Media, Inc., 2017).
34. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
35. Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* **12,** 2121–2159 (2011).
36. Tieleman, T. & Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* **4,** 26–31 (2012).