Prédiction d'ondes sismiques dans des milieux complexes par apprentissage de réseaux neuronaux

Sohaib Choufani,William Roche,Oussama Saadi,Samuel Sithakoul

May 2022

1 Abstract

A compléter Le projet est disponible publique GitHub : LIEN GITHUB

2 Introduction

La modélisation et la simulation de la propagation d'ondes sismiques sont des enjeux majeurs pour la sécurité dans un but de prédiction des dommages sur des zones ciblées. En effet, pour l'hydrologie, la géologie ou le génie civil, ces ondes sont riches en information puisqu'elle peuvent renseigner sur la structure interne de la Terre et donc permettre par exemple la localisation de gisements d'hydrocarbure ou la prévention de risques naturels. C'est pourquoi il est important de reproduire le comportement d'une onde sur un domaine de la manière la plus détaillée et réaliste possible. à partir d'équations aux dérivées partielles en dimension quelconque et pour des milieux pouvant être hétérogènes.

L'établissement des principales lois régissant l'univers a longtemps occupé les physiciens, ces lois prennent souvent la forme de telles équations différentielles et la difficulté réside dans la résolution de ces équations : on arrive à montrer qu'on ne peut pas toujours expliciter la solution, il s'agit d'une théorie de galois différentielle. Les chercheurs se sont alors penchés sur le problème d'approximation des solutions des différentes EDP non linéaires de grande dimension. Il s'agit de l'un des problèmes les plus difficiles en mathématiques appliquées : concevoir et analyser de telles méthodes d'approximation d'EDP. Les méthodes d'approximation standard, telles que les méthodes des différences finies, les méthodes par éléments finis, les méthodes d'approximation spectrale de Galerkin [1], les méthodes d'approximation de Monte Carlo, souffrent d'une forte dépendance de la dimensionnalité dans le sens que le nombre d'opérations de calcul du schéma d'approximation utilisé croît de façon exponentielle : pour les équations différentielles ordinaires linéaires d'ordre 2 et 3, des algorithmes

de résolution exacte avec des temps de calcul réalistes existent, se fondant sur une étude préalable précise des groupes de Galois différentiels potentiels de ces équations. [2] Quelques travaux datant de moins de 5 ans utilisent le Deep Learning comme alternative rapide aux méthodes classiques de résolution d'EDP : les résultats sont particulièrement prometteurs. Le graphe suivant montre l'évolution de l'intérêt porté pour cette nouvelle méthode : il met en avant une brusque augmentation sur les dernières années.

Ces résultats sont obtenus en utilisant la base de publications Lens [3], en cherchant les publications dont le titre, les mots clés, ou l'abstract contiennent simultanément les termes artificial networks et Partial Differential Equations . Toujours sur la base Lens, et avec la même requête, on s'aperçoit que sur les 20 universités mondiales qui publient le plus, les trois premières sont américaines : Brown, Stanford, et Purdue University. On observe également que de plus en plus d'universités qui publiaient peu dans le domaine de la mécanique ont commencé à s'y intéresser à travers l'intelligence artificielle. L'idée de base pour appliquer le deep learning aux EDP est simple : les réseaux de neurones artificiels sont des approximateurs universels qui peuvent, sur la base d'une quantité suffisante de données d'apprentissage, apprendre à associer des éléments d'entrée à des éléments de sortie. De la reconnaissance vocale [4] au contrôle de qualité d'un tissu [5], les progrès récents du deep learning ont été largement profitable au secteur scientifique. Les réseaux de neurones convolutifs et leurs dérivés, par exemple, font partie de ces réseaux très efficaces avec une grande précision applicables aux problèmes géologiques d'estimation [6] ou de quantification de l'incertitude des géo-matériaux complexes [7]. Tahmasebi réalise dans son article un tour d'horizon sur les différentes applications des méthodes d'IA en géosciences [8]. Ces réseaux (data driven) présentent le défaut de ne bien fonctionner que lorsqu'une grande quantité de données est disponible. Ainsi, de telles méthodes ne sont pas passées en revue ici. Dans la plupart des applications d'ingénierie, sinon toutes, l'acquisition de données est une tâche coûteuse et chronophage ainsi un intérêt tout particulier est porté ici sur les méthodes d'IA optimales pour remédier au problème d'acquisition des données.

Dans la plupart des cas, les algorithmes du deep learning sont considérés comme une boîte noire donc encore difficilement interprétable et généralement sans aucune contribution de la connaissance préalable du système. Cette connaissance, en géologie, peut se présenter sous la forme de lois physiques régissant la dynamique temporelle d'un système. Cette information préalable est supposée être un outil pour compléter les données disponibles et pour pouvoir alors rejeter les solutions non réalistes [9]. Les algorithmes qui encodent ces lois physiques dans leurs processus d'apprentissage sont appelés méthodes contraintes par la physique (PINNs ou physics driven) [9],[10]. Raissi montre dans son article que ces équations conduisent rapidement le processus d'apprentissage vers la bonne solution et lui permettent d' apprendre le lien entre les entrées et les sorties même avec peu de données. La startup franco-américaine Atmo a développé un outil, basé sur le deep learning, pour des prévisions météorologiques et pouvoir alors vendre ces calculateurs aux pays qui n'ont pas de services météorologiques

et climatiques capables de telles prévisions. Leur outil offre des résultats de prédiction semblables à ceux des centres météorologiques nationaux d'autres pays. De même, NVIDIA propose NVIDIA Simnet [11] ,un outil permettant d'utiliser aisément leur réseau de neurones pour la résolution d'EDP. Ces approches ne sont donc plus limitées au domaine de la recherche, mais deviennent populaires en ingénierie.

Motivé par les recherches et les développements faits pour cette méthode encore récente, ce travail a pour but d'analyser la capacité des PINN à prédire les mouvements sismiques avec des performances statistiques similaires aux techniques de résolution classiques présentées. La finalité est de proposer à la communauté scientifique une interface permettant l'accès à notre base de données et à notre algorithme et contribuer au développement de ces réseaux de neurones prometteurs. Plus spécifiquement, notre contribution se résume comme il suit:

1. Nous proposons une modélisation pour la propagation d'une onde
2. Nous implémentons une méthode d'entraînement d'un PINN
3. Nous adaptons un calcul des coefficients de pondération de la loss
4. Nous offrons une interface d'utilisation modulable

3 Travaux relatifs

3.1 Réseaux de neurones

Les Réseaux de neurones artificiels (NNs pour Neural Networks) sont des algorithmes inspirés des réseaux de neurones biologiques. Comme son nom l'indique, un NN est basé sur un réseau d'unités ou de nœuds connectés appelés neurones artificiels. Les connexions sont appelées arêtes et sont orientés. Chaque neurone correspond à un automate simple. Chaque connexion peut transmettre un signal, un nombre réel, suivent l'orientation de celle-ci. Un neurone reçoit des signaux, puis les combine et peut envoyer un signal aux autres neurones qui lui sont connectés. Le fonctionnement d'un neurone se produit en deux étapes. Les arêtes ont généralement un poids. Au cours de la première étape, dite de pré-activation, le neurone réalise une combinaison linéaire des différentes entrées suivent le poids des arêtes. Puis à l'étape d'activation, le neurone calcule l'image du réel obtenu par une fonction non-linéaire dite fonction d'activation. Par exemple, Les neurones peuvent avoir un seuil tel qu'un signal n'est envoyé que si le signal agrégé franchit ce seuil. Une fonction d'activation communément utilisée est la fonction sigmoïde définie sur $\mathbb{R}$ par:

$$x \mapsto \frac{1}{1+e^{-x}}$$

En général, les neurones sont regroupés en couches. Les signaux vont de la première couche (la couche d'entrée) à la dernière couche (la couche de sortie), éventuellement après avoir traversé plusieurs couches intermédiaires (hidden

layers). Par exemple, pour un réseau constitué de 4 couches, on peut alors écrire que la sortie est la suivante :

$$y = g^{(4)}(B^{(4)} + W^{(4)}g^{(3)}(B^{(3)} + W^{(3)}g^{(2)}(B^{(2)} + W^{(2)}g^{(1)}(B^{(1)} + W^{(1)}x))))$$

$g^{(i)}$ étant la fonction d'activation de la i-ème couche, $B^{(i)}$ et $W^{(i)}$ les vecteurs et matrices décrites plus haut.

Soit un ensemble ouvert U de $\mathbb{R}^n$. Soit un vecteur d'entrée $X \in U$ et un réseau de neurones avec n neurones en première couche et m neurones en dernière couche qui prédit une valeur y_{pred}. On note θ l'ensemble des paramètres du réseau, c'est-à-dire l'ensemble des poids et des biais. L'objectif est de modifier le réseau de neurones, en agissant sur la valeur des poids des arêtes pour obtenir en sortie y_{pred} aussi proche de y_real, les valeurs réelles associées à X à partir des données (labels).Il s'agit de l'étape d'apprentissage.

On introduit donc une fonction coût, dite loss, qui permet de quantifier la distance entre y_{pred} et y_{real}. Il s'agit d'un problème d'optimisation et l'objectif est alors de minimiser la loss. Par exemple, on peut utiliser la MSE (Mean-Squared Error):

$$L : (x,) \mapsto \frac{1}{2}\|y_{pred} - y_{real}\|_2^2$$

L'objectif est donc de trouver θ_{min} qui minimise la fonction coût L. Une méthode qui pourrait être utilisée pour calculer de manière approchée θ_{min} est la méthode dite de descente du gradient. On calcule de manière itérative :

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta}(x)$$

Où α le learning rate est choisi de manière judicieuse de sorte que la vitesse de convergence soit optimale.

3.2 PINN

Etant donnée l'opérateur D linéaire et l'équation fonctionnelle d'inconnue u :

$$\forall x \in U, D(u)(x) = h(x)$$

Si l'on introduit des conditions supplémentaires à l'équation fonctionnelles, tel que des conditions au bords, il est possible de reproduire ces conditions dans la fonction Loss suivant différentes stratégies. Nous reviendrons plus en détails sur la fonction Loss que nous avons décidé d'implanter dans l'algorithme proposé dans ce rapport.

3.3 L'équation d'onde

4 Méthode proposée

4.1 Modélisation des ondes sismiques

4.2 Entraînement du PINN

Afin de pouvoir entraîner le réseau de neurones pour approcher les solutions de l'équation d'onde définie plus haut, il est nécessaire de définir les données d'entraînement qui seront fournies au modèle pour s'y adapter en fonction du certaine fonction de coût à définir.

Tout d'abord, l'ensemble des données ne recquiert pas nécessairement de données réelles relevées sur le terrain même si celles-ci peuvent intervenir comme terme régularisant supplémentaire lors du calcul de la fonction de coût.

On considère un ensemble de points d'entraînements

$$x = (x_i, x_b, x_r)$$

où $x_i \in \mathbb{R}^{i \times n}, x_b \in \mathbb{R}^{b \times n}, x_r \in \mathbb{R}^{r \times n}$.

Ici chacun des x_k représente une matrice de points du domaine étudié à partir desquels va être calculée la solution approchée par le réseau de neurones. Dans le cas présent, n correspond à la dimensionnalité du problème en incluant la dimension temporelle et la ou les dimensions spatiales. Chaque point d'entraînement fourni au modèle est donc un vecteur de taille n : $(t, x_1, x_2, ..., x_{n-1})$. Les indices i,b et r renvoient respectivement aux points des conditions initiales, c'est-à-dire à l'instant $t = 0$, aux points des conditions au bord, qu'ils s'agissent de conditions de Neumann ou de Dirichlet, et aux points résiduels, c'est-à-dire à l'ensemble des points intérieurs du domaine qui vont servir à calculer le résidu de l'équation différentielle.

Ces points constituent alors un maillage du domaine étudié. Dans le problème des équations d'onde et par la nature même des PINN, un échantillonnage suivant une loi uniforme des points d'entraînement selon les dimensions spatiales et temporelles suffit à entraîner le modèle.

On associe aussi aux vecteurs x_i et x_b, les vecteurs $u_i = u_{real}(x_i) \in \mathbb{R}^{i \times n}$, $u_b = u_{real}(x_b) \in \mathbb{R}^{b \times n}$ et $v_b = v_{real}(x_b) \in \mathbb{R}^{b \times n}$ qui correspondent aux valeurs prises par la solution voulue selon les conditions initiales et aux bords imposées ainsi que la dérivée voulue pour la solution exacte aux bords. La situation peut facilement être élargie à un ensemble de conditions aux bords pour des dérivées d'ordres supérieures (Neumann quelconque) ou pour des valeurs de la solution en plus de frontières en dimension supérieure (Dirichlet quelconque). Ces valeurs serviront ensuite aussi au calcul de la fonction de coût.

Ensuite, le modèle utilisé est un simple multiperceptron prenant en entrée des vecteurs de taille n correspondant à la dimension temporelle et les $n-1$ données

spatiales. Une couche de rescaling est utilisée afin de normaliser l'ensemble des données d'entraînement dans l'intervalle $[-1, 1]$. Cela permet de profiter au mieux des domaines intéressants des fonctions d'activation utilisées et de l'utilisation potentielle de Batch normalization. La sortie du réseau correspond simplement à un vecteur de taille 1 qui représente la valeur scalaire de la solution approchée en fonction des points d'entrée. Dans toute la suite, on notera u la solution approchée par le PINN.

Le modèle est ensuite entraîné pour minimiser la fonction de coût:

$$\mathcal{L} = \lambda_r \mathcal{L}_r + \lambda_b \mathcal{L}_i + \lambda b \mathcal{L}_b + \lambda_{db} \mathcal{L}_{db}$$

Les coefficients $lambda_k$ sont des hyperparamètres à optimiser représentant l'importance de chaque terme dans la contribution totale. La raison pour laquelle ce coefficient est fixé identitiquement pour la condition aux bords et la condition initiale sera expliquée dans la partie suivante. En première approche, ils sont tous fixés à 1.

Pour des raisons de lisibilité, décomposons les différents termes de cette loss en dimension 1 même si l'approche présentée est facilement généralisable en dimension supérieure en remplaçant les dérivées par des gradients et les dérivées secondes par des lapalaciens.

1) Loss résiduelle :

$$\mathcal{L}_r = \frac{1}{N} \sum_{i=1}^{N} \mathcal{F}(x_r^{(i)})$$

où N est la taille des batchs utilisée et $\mathcal{F}$ l'opérateur résidu de l'équation aux dérivées partielles définie ici par:

$$\mathcal{F} = \frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2}$$

Les dérivées secondes ou les laplaciens en dimension supérieure à 1 de la solution approchée sont calculés numériquement par méthode de dérivation automatique.

2) Loss initiale :

$$\mathcal{L}_i = \frac{1}{N} \sum_{j=1}^{N} |u(x_i^{(j)}) - u_i^j|^2$$

Il s'agit simplement de l'erreur quadratique moyenne entre la prédiction du réseau de neurones et la solution exacte sur les points x_i de la condition initiale.

3) Loss aux bords:

$$\mathcal{L}_b = \frac{1}{N} \sum_{j=1}^{N} |u(x_b^{(j)}) - u_b^j|^2$$

De même c'est la MSE entre la prédiction du réseau de neurones et la solution exacte sur les points x_b de la condition aux bords.

4) Loss sur la dérivée aux bords:

$$\mathcal{L}_{db} = \frac{1}{N}\sum_{j=1}^{N} |\frac{\partial u}{\partial t}(x_b^{(j)}) - v_b^j|^2$$

C'est ici la MSE entre la dérivée de la prédiction du réseau de neurones calculée par dérivation automatique et la dérivée exacte recherchée sur les points x_b de la condition aux bords.

On peut donc réécrire l'objectif d'optimisation du réseau de neurones:

$$\frac{1}{N}\sum_{i=1}^{N} \lambda_r \mathcal{F}(x_r^{(i)}) + \lambda_b |u(x_i^{(j)}) - u_i^j|^2 + \lambda_b |u(x_b^{(j)}) - u_b^j|^2 + \lambda_{db} |\frac{\partial u}{\partial t}(x_b^{(j)}) - v_b^j|^2$$

Le modèle est ensuite entraîné pendant un certain nombre d'epochs par rétropropagation du gradient en minimisant $\mathcal{L}$ avec comme méthode d'optimisation des paramètres la descente de gradient stochastique Adam. L'ensemble des données est divisé en batchs à chaque epoch dans un objectif de parallélisation des calculs et donc une meilleure efficacité computationnelle.

4.3 Implémentation de la méthode NTK

La recherche des bonnes valeurs des poids λ_k devant chaque terme de la loss est une recherche d'hyperparamètres pouvant être longue et fastidieuse mais néanmoins importante pour prendre en compte de la manière la plus juste possible les phénomènes physiques sous-jacents.

Dans un premier temps, il est posssible de procéder de manière empirique en observant graphiquement quels facteurs de la loss sont prépondérants et en affectant un poids plus important aux facteurs dont la loss est encore plus élevée. Cette méthode a ses limites d'autant plus si les hyperparamètres ne sont pas indépendants comme c'est le cas dans la situation étudiée. D'autres méthodes d'exploration de l'espace $\mathbb{R}^3$ sont possibles comme une grid search, une optimisation baysésienne ou des algorithmes génétiques. Cependant, des méthodes sont longues et computationnellement lourdes, ce qui nuit à la volonté d'un entraînement rapide pour un problème physique donné.

C'est pourquoi il existe une méthode de calcul automatique de ces coefficients à partir des NTK (Neural tangent kernel) du réseau de neurones.

On peut montrer qu'à partir des points $\left\{x_b^i, g\left(x_b^i\right)\right\}_{i=1}^{N_b}, \left\{x_r^i, f\left(x_r^i\right)\right\}_{i=1}^{N_r}$ où f représente le second membre de l'équation et g la condition aux bords (pour une écriture plus concise, condition initiale et conditions aux bords ne sont pas distingués), la solution approchée $u(t)$ et l'opérateur de l'EDP appliqué à la solution $\mathcal{L}u(t)$ évoluent selon

$$\begin{bmatrix} \frac{du(x_b,\theta(t))}{dt} \\ \frac{d\mathcal{L}u(x_r,\theta(t))}{dt} \end{bmatrix} = -\begin{bmatrix} K_{uu}(t) & K_{ur}(t) \\ K_{ru}(t) & K_{rr}(t) \end{bmatrix} \cdot \begin{bmatrix} u\left(x_b,\theta(t)\right) - g\left(x_b\right) \\ \mathcal{L}u\left(x_r,\theta(t)\right) - f\left(x_r\right) \end{bmatrix}$$

où $K_{ru}(t) = K_{ur}^T(t)$ et $K_{uu}(t) \in \mathbb{R}^{N_b \times N_b}, K_{ur}(t) \in \mathbb{R}^{N_b \times N_r}$, et $K_{rr}(t) \in \mathbb{R}^{N_r \times N_r}$. Ces matrices sont définies par :

$$(K_{uu})_{ij}(t) = \left\langle \frac{du\left(x_b^i, \theta(t)\right)}{d\theta}, \frac{du\left(x_b^j, \theta(t)\right)}{d\theta} \right\rangle$$

$$(K_{ur})_{ij}(t) = \left\langle \frac{du\left(x_b^i, \theta(t)\right)}{d\theta}, \frac{d\mathcal{L}u\left(x_r^j, \theta(t)\right)}{d\theta} \right\rangle$$

$$(K_{rr})_{ij}(t) = \left\langle \frac{d\mathcal{L}\left(x_r^i, \theta(t)\right)}{d\theta}, \frac{d\mathcal{L}\left(x_r^j, \theta(t)\right)}{d\theta} \right\rangle$$

On peut aussi les calculer de la manière suivante:

$$K_{uu}(t) = J_u(t)J_u^T(t), \quad K_{rr}(t) = J_r(t)J_r^T(t), \quad K(t) = \begin{bmatrix} J_u(t) \\ J_r(t) \end{bmatrix} \left[J_u^T(t), J_r^T(t)\right]$$

où $J_u(t)$ et $J_r(t)$ sont les matrices jacobiennes de $u(t)$ et $\mathcal{L}u(t)$ par rapport à θ et $K(t) = \begin{bmatrix} K_{uu}(t) & K_{ur}(t) \\ K_{ru}(t) & K_{rr}(t) \end{bmatrix}$.

On calcule alors les coefficients λ par:

$$\lambda_b = \frac{Tr(K)}{Tr(K_{uu}}$$

et

$$\lambda_r = \frac{Tr(K)}{Tr(K_{rr}}$$

à une fréquence répétée pendant l'entraînement du PINN.

Dans le cas étudié ici, il est nécessaire de prendre aussi en compte la condition aux bord imposée à la dérivée de la solution approchée. Les formules précédentes se modifient en:

$$\begin{bmatrix} \frac{du(x_u, \theta(t))}{dt} \\ \frac{du_t\left(x_{u_t}, \theta(t)\right)}{dt} \\ \frac{d\mathcal{L}u(x_r, \theta(t))}{dt} \end{bmatrix} := \widetilde{K}(t) \cdot \begin{bmatrix} u\left(x_b, \theta(t)\right) - g\left(x_b\right) \\ u_t\left(x_{u_t}, \theta(t)\right) \\ \mathcal{L}u\left(x_r, \theta(t)\right) \end{bmatrix}$$

où ici u_t correspond à la dérivée de la solution approchée calculée aux points correspondants x_{u_t} par dérivation automatique. De même que dans la partie précédente on définit $\widetilde{K}$ par:

$$\widetilde{K}(t) = \begin{bmatrix} \frac{\lambda_u}{N_u} J_u(t) \\ \frac{\lambda_{u_t}}{N_{u_t}} J_{u_t}(t) \\ \frac{\lambda_r}{N_r} J_r(t) \end{bmatrix} \cdot \left[J_u^T(t), J_{u_t}^T(t), J_r^T(t)\right]$$

De même que dans le cas général introduit précédemment, on peut calculer les différentes matrices K :

$$[K_u(t)]_{ij} = \left[J_u(t)J_u^T(t)\right]_{ij} = \left\langle \frac{du\left(x_u^i, \theta(t)\right)}{d\theta} \frac{du\left(x_u^j, \theta(t)\right)}{d\theta} \right\rangle$$

$$[K_{u_t}(t)]_{ij} = \left[J_{u_t}(t)J_{u_t}^T(t)\right]_{ij} = \left\langle \frac{du_t\left(x_{u_t}^i \theta(t)\right)}{d\theta}, \frac{du\left(x_{u_t}^j, \theta(t)\right)}{d\theta} \right\rangle$$

$$[K_r(t)]_{ij} = \left[J_r(t)J_r^T(t)\right]_{ij} = \left\langle \frac{d\mathcal{L}u\left(x_r^i, \theta(t)\right)}{d\theta}, \frac{d\mathcal{L}u\left(x_r^j, \theta(t)\right)}{d\theta} \right\rangle$$

On peut alors finalement calculer les pondérations de chaque loss λ à partir de la trace des matrices NTK:

$$\lambda_u = \frac{Tr\left(K_u\right) + Tr\left(K_{u_t}\right) + Tr\left(K_r\right)}{Tr\left(K_u\right)}$$

$$\lambda_{u_t} = \frac{Tr\left(K_u\right) + Tr\left(K_{u_t}\right) + Tr\left(K_r\right)}{Tr\left(K_{u_t}\right)}$$

$$\lambda_r = \frac{Tr\left(K_u\right) + Tr\left(K_{u_t}\right) + Tr\left(K_r\right)}{Tr(K)}$$

Les propositions et preuves techniques associées au calcul des NTK sont laissées à la liberté du lecteur dans l'article correspondant. Empiriquement, cette méthode fournit des résultats de meilleure qualité que des valeurs fixées pour chaque pondération à partir d'un grand nombre d'expérimentations. Néanmoins, étant donné la complexité qu'ajoute cette méthode, il est nécessaire d'effectuer la mise à jour des coefficientsà une fréquence limitée d'itérations et éventuellement sur des batchs de données au lieu de l'ensemble du dataset.

5 Résultats expérimentaux

5.1 Détails d'implémentation

5.2 Comparaison

5.3 Interface utilisateur

6 Conclusion + Travaux futurs

Ce travail a permis d'explorer l'intérêt des Physics Informed Neural Networks pour modéliser la propagation d'ondes à partir de la résolution de l'équation d'onde aux dérivées partielles.

7 Références