Cours 12: Introduction aux opérateurs neuronaux

Emmanuel Franck[*],

28 novembre, 2024
Master CMSI, M2, Strasbourg

[*] MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France

Outline

Objectif

Opérateur neuronaux pour les EDP elliptiques

Objectif

Objectif: inversion d'opérateur

- Soit un problème elliptique de la forme

$$\begin{cases} -\Delta u = f(\mathbf{x}), & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases}$$

- On voudrait prédire rapidement une solution pourr un $f(\mathbf{x})$ ou un $g(\mathbf{x})$ généraux.
- Si la solution existe, on peut formellement dire qu'il existe $G : \mathcal{H}_x \to \mathcal{H}_y$ tel que

$$u(.) = G^{-1}(f(.))$$

Idée

Peut t'on apprendre G^{-1} avec un réseau de neurones ?

- On peut généraliser en cherchant a approcher $G^+(\mathbf{a}(.)) = u(.)$ avec $\mathbf{a}$ l'ensemble des fonctions données du problème (source, condition limite, coefficients de l'EDP).
- Soit G_θ un réseau on va résoudre

$$\min_\theta \left(\int_{\mathcal{A}} \|u(.;\mathbf{a}) - G_\theta(\mathbf{a})\|^2_{\mathcal{H}_y} d\mu(\mathbf{a}) \right)$$

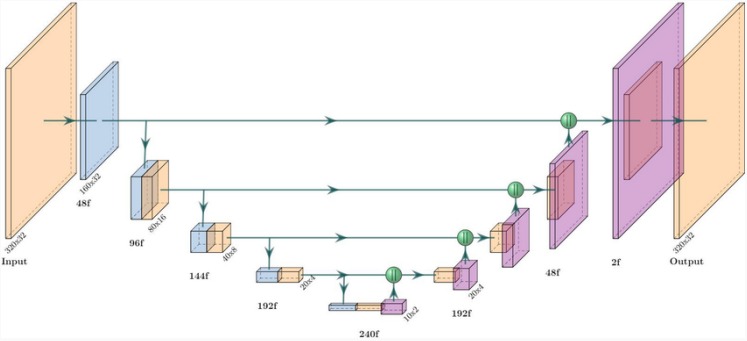
- **Questions**: Comment approcher $\mu(\mathbf{a})$ et quelle architecture prendre ?

Inversion d'opérateur et Vnet

- On va représenter nos fonctions $u(.)$ et $\mathbf{a}(.)$ sur une grille de taille N_x.
- On souhaite donc construire:

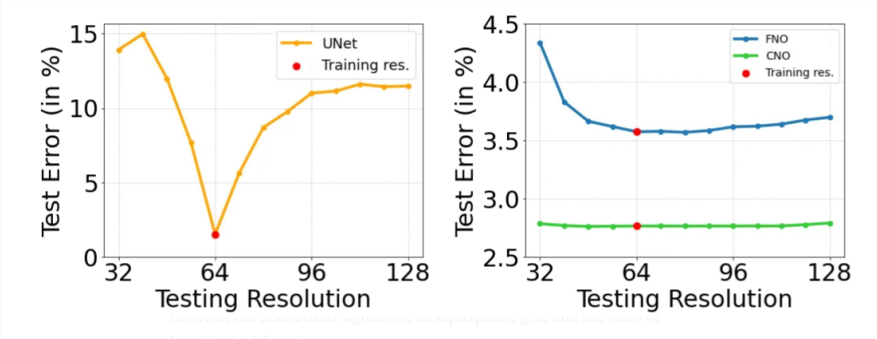
$$\mathbf{u}_h \approx G_\theta(\mathbf{a}_h)$$

- Il est assez naturel d'utiliser des réseaux de convolutifs.
- Le modèle de référence est le Vnet.



Vnet et invariance à la résolution

- On aimerait que le résultat de dependent pas de la résolution voir de la discrétisation.



- Pas le cas du Vnet.

Opérateur neuronaux

Il s'agit de réseaux dont les entrées et les sorties serait dans des espaces de Hilbert. En pratique cela veut dire que ce qui est appris ne dépend pas de la résolution voir de la discrétisation.

Objectif

Opérateur neuronaux pour les EDP elliptiques

Réseau de neurones et espace fonctionnel

- Questions:
  - Comment construire des couches de réseaux de neurones en dimension infini ?
  - Comment paramétriser/discrétiser les couches pour obtenir l'invariance par résolution.
  - Comment apprendre ?

Couche classique

Une couche classique de réseau de neurones de $\mathbb{R}^l$ dans $\mathbb{R}^{l+1}$ est donnée par

$$\mathbf{y} = \sigma(A\mathbf{x} + \mathbf{b})$$

On compose une application linéaire et une fonction nonlinéaire coordonnée par coordonnée.

- On généralise les applications linéaires par les opérateurs linéaires en dimension infini.
- Opérateur linéaire à noyau:

$$g(\mathbf{x}) = \int_\Omega k(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

- On se retrouve donc avec une couche de la forme:

$$g_2(\mathbf{x}) = \sigma(g_1(\mathbf{x})), \forall \mathbf{x} \in \Omega, \qquad g_1(\mathbf{x}) = \left(\int_\Omega k(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y} \right)$$

avec σ appliquée localement en chaque point $g_1(\mathbf{x})$.

Couche cachée

- On peut maintenant construire une couche plus général pour un vecteur en fonction en entrée.

Couche interne d'opérateur neuronal

Soit une entrée $\mathbf{v}_l(\mathbf{x}) \in H(\mathbb{R}^d)^{d_l}$ et une sortie $\mathbf{v}_{l+1}(\mathbf{x}) \in H(\mathbb{R}^d)^{d_{l+1}}$. Une couche cachée est définie par

$$\mathbf{v}_{l+1}(\mathbf{x}) = \sigma_{l+1}\left(W_l\mathbf{v}_l(\mathbf{x}) + \int_\Omega \kappa(\mathbf{x},\mathbf{y})\mathbf{v}_l(\mathbf{y})\mathrm{d}\mathbf{y} + b_l(\mathbf{x})\right)$$

avec une matrice constante $W \in \mathcal{M}_{d_{l+1},d_l}(\mathbb{R})$, $\kappa(\mathbf{x},\mathbf{y}) \in \mathcal{M}_{d_{l+1},d_l}(\mathbb{R})$ et $b_l(\mathbf{x}) \in \mathbb{R}^{l+1}$

- Le domaine pourrait pratique d'une couche à l'autre mais en pratique c'est pas vraiment utilisée.
- Le noyau pourrait avoir des dépendances plus compliquées (en $\mathbf{v}$, en paramètres) etc.

Remarque

On a donc une opération en espace locale qui mélange les fonctions et une nonlocale en espace. Si on prend $\kappa(\mathbf{x}-\mathbf{y})$ on retrouve sur une version continue des couches convolutives.

Architecture complète I

- Avec les réseaux de neurones classiques on augmente souvent la dimension avant de la redescendre. On peut généraliser ici.

Couche d'extrapolation

Soit une entrée $\mathbf{a}(\mathbf{x}) \in H(\mathbb{R}^d)^{d_{in}}$ la couche d'extrapolation génère une sortie $\mathbf{v}_0(\mathbf{x}) \in H(\mathbb{R}^d)^{d_0}$ avec $d_0 > m$ par la formule:

$$\mathbf{v}_0(\mathbf{x}) = MLP_\theta(\mathbf{x}, \mathbf{a}(\mathbf{x}))$$

Couche de projection

Soit une entrée $\mathbf{v}_L(\mathbf{x}) \in H(\mathbb{R}^d)^{d_L}$ la couche de projection génère une sortie $\mathbf{u}(\mathbf{x}) \in H(\mathbb{R}^d)^{d_{out}}$ avec $d_{out} < d_L$ par la formule:

$$\mathbf{u}(\mathbf{x}) = MLP_\theta(\mathbf{x}, \mathbf{v}_L(\mathbf{x}))$$

- La couche d'extrapolation génère des fonctions d'entrées supplémentaire par des transformations locales en $\mathbf{x}$
- La couche projection combine les fonctions de la dernière couche par des transformations locales en $\mathbf{x}$

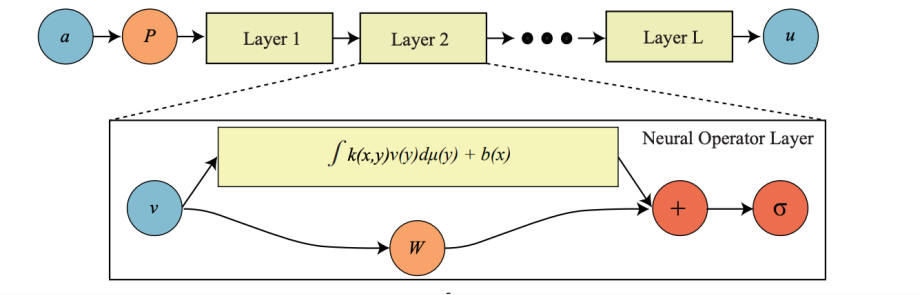
Architecture complète II

Architecture complète

Soit une entrée $\mathbf{a}(\mathbf{x}) \in H(\mathbb{R}^d)^{d_{in}}$ et $\mathbf{u}(\mathbf{x}) \in H(\mathbb{R}^d)^{d_{out}}$ la sortie. Un opérateur neural est donnée par la succession de couche suivante:

$$\mathcal{G}_\theta := \mathcal{P} \circ \sigma_L(W_L + \mathcal{K}_{TL} + b_L) \circ \cdots \circ \sigma_0(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{Q}$$

avec $\mathcal{Q}$ la couche d'extrapolation, $\mathcal{P}$ celle de projection et $(W_i + \mathcal{K}_i + b_i)$ un couche linéaire.



Entrainement

- En pratique on veut minimiser:

$$\mathcal{J}(\theta) = \int_{\mathcal{A}} \int_{\Omega} \| G_\theta(a(\mathbf{x})) - u(\mathbf{x}) \|_2^2 \, d\mathbf{x} d d\mu(\mathbf{a})$$

- Processus général:
  - On génère des données discrètes (source etc) $(\mathbf{a}_1, \ldots, \mathbf{a}_m)$ avec $\mathbf{a}_i \in \mathcal{M}_{d_{in},n}(\mathbb{R})$, d_{in} vecteurs à n coefficients
  - En utilisant un solveur on calcul les solutions associées:

$$(\mathbf{u}_1, \ldots, \mathbf{u}_m) \in \mathbb{R}^n$$

    avec $\mathbf{u}_i \in \mathcal{M}_{d_{in},n}(\mathbb{R})$ d_{out} vecteurs à n coefficients
  - On assemble la fonction de cout

$$\mathcal{J}(\theta) = \sum_{i=1}^{m} \int_{\Omega} \| G_\theta(a_i(\mathbf{x})) - u(\mathbf{x}) \|_2^2 \, d\mathbf{x}$$

  - On effectue une minimisation:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{J}(\theta)$$

- Pour échantillonner on utilise souvent des processus Gaussiens bien choisis.

Fonctions de coûts physiquement informés

- On peut aussi utiliser des fonctions de coûts tenant compte de l'équation (comme pour les PINNs).
- Cependant elles sont en générales pas suffisantes. Il faut les couplés avec les données.
- Soit une EDP :

$$\begin{cases} L_b(u(\mathbf{x})) = f(\mathbf{x}), & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \partial\Omega \end{cases}$$

- Fonctions de coûts physique:

$$\mathcal{J}_{edp}(\theta) = \sum_{i=1}^{m} \int_{\Omega} \| L_{b_i}(G_\theta(a_i(\mathbf{x}))) - f_i(\mathbf{x}) \|_2^2 \, d\mathbf{x} + \lambda \sum_{i=1}^{m} \int_{\partial\Omega} \| G_\theta(a_i(\mathbf{x})) - g_i(\mathbf{x}) \|_2^2 \, d\mathbf{x}$$

Remarque

Le calculs de fonctions de coût physique pour les opérateurs neuronales génère un coût mémoire est très importante. On détaillera les potentiels implémentations selon les architectures.