# Cours 13: Architectures d'opérateurs neuronaux

Emmanuel Franck[*],

28 novembre, 2024
**Master CMSI, M2**, Strasbourg

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


# Outline

## Opérateur neuronaux intégraux

# GreenNet et EDP linéaire I

- On propose ici une première architecture qu'on va appeler GreenNet.
- On considère ici une équation linéaire de la forme:

$$L(\mathbf{u}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$

avec $\mathbf{u}(\mathbf{x}), \mathbf{f}(\mathbf{x}) \in H^s(\Omega)^p$

**GreenNet**

Il s'agit d'un réseau à une couche, sans couche d'extrapolation et de projection, ou la couche principale est $G_\theta : H^s(\Omega)^p \to H^s(\Omega)^p$ est paramétrée de la façon suivante:

$$G_\theta(\mathbf{f}) = \int_\Omega k_\theta(\mathbf{x}, \mathbf{y})\mathbf{f}(\mathbf{y})d\mathbf{y} + u_\theta(\mathbf{x})$$

avec $k_\theta$ un réseau de type MLP, Fourier etc.

- Puisqu'on paramètrise le noyau au niveau continue, ce qui sera appris sera Indépendant de la résolution et de la discrétisation.
- Les intégrales peuvent être approchées par Monte-Carlo.

## GreenNet En pratique

La couche est donnée par:

$$G_\theta(\mathbf{f}_1, ...., \mathbf{f}_m) = \sum_{i=1}^{m} k_\theta(\mathbf{x}, \mathbf{y}_i)\mathbf{f}(\mathbf{y}_i) + u_\theta(\mathbf{x})$$

avec $k_\theta$ un réseau de type MLP, Fourier etc.

- **Est ce que cette paramètrisation est une bonne idée** ? Oui d'après la théorie de Green.
- Considérons: $-\Delta u(\mathbf{x}) = f(\mathbf{x})$.
  - On pose $\Delta G(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y})$
  - On rappelle les propriétés des Dirac:
  $$\int_\Omega \delta(\mathbf{x} - \mathbf{y}) f(\mathbf{y}) d\mathbf{y} = f(\mathbf{y})$$
  - On introduit l'identité:
  $$\int_{\partial\Omega} [\phi \nabla\psi - \psi\nabla\phi] \cdot \mathbf{n} dS = \int_\Omega \left[\phi\nabla^2\psi - \psi\nabla^2\phi\right] dV$$
  - Choisissons $\phi = u(\mathbf{x})$ et $\psi = G(\mathbf{x}, \mathbf{y})$ Cela donne
  $$A = \int_{\partial\Omega} [u(\mathbf{x})\nabla G(\mathbf{x}, \mathbf{y}) - G(\mathbf{x}, \mathbf{y})\nabla u(\mathbf{x})] \cdot \mathbf{n} dS = \int_\Omega \left[u(\mathbf{x})\nabla^2 G(\mathbf{x}, \mathbf{y}) - G(\mathbf{x}, \mathbf{y})\nabla^2 u(\mathbf{x})\right] dV$$
  $$A = \int_\Omega [u(\mathbf{x})\delta(\mathbf{x} - \mathbf{y}) - G(\mathbf{x}, \mathbf{y}) f(\mathbf{x})] dV = u(\mathbf{y}) - \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) dV.$$
  - On a donc
  $$u(\mathbf{y}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) dV + \int_{\partial\Omega} [u(\mathbf{x})\nabla G(\mathbf{x}, \mathbf{y}) - G(\mathbf{x}, \mathbf{y})\nabla u(\mathbf{x})] \cdot \mathbf{n} dS$$

## Théorie de Green

Soit le problème

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

Soit

$$\begin{cases} -\Delta G(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}), & \forall \mathbf{x}, \mathbf{y} \in \Omega \\ G(\mathbf{x}, \mathbf{y}) = 0, & \forall \mathbf{x} \in \partial\Omega \end{cases}$$

alors la solution est donnée par

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) dV + \int_{\partial\Omega} g(\mathbf{x}) (\nabla G(\mathbf{x}, \mathbf{y}) \cdot \mathbf{n}) dS$$

- Par principe de superposition on a aussi

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y} + u_{hom}(\mathbf{x})$$

avec

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

# Théorie de Green III

- Cette théorie peut être généralisée a chaque EDP elliptique linéaire.
- Ce résultat justifie l'architecture des GreenNet.

**Théorie de Green**

Soit le problème

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

Soit

$$\begin{cases} -\Delta G(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}), & \forall \mathbf{x} \in \Omega \\ \partial_{\mathbf{n}} G(\mathbf{x}, \mathbf{y}) = 0, & \forall \mathbf{x} \in \partial\Omega \end{cases}$$

alors la solution est donnée par

$$u(\mathbf{x}) = \int_{\Omega} G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) dV - \int_{\partial\Omega} G(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) dS$$

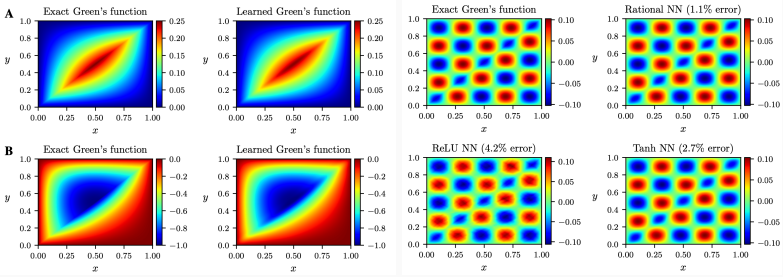
- Un certains nombre de fonctions de Green sont connues (Wikipedia). Les GreenNet peuvent en déterminer de nouvelles.

# GreenNet et EDP linéaire II

- Puisque le noyau de Green satisfait:

$$-\Delta G(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}),$$

- Il s'agit d'une fonction dont les dérivées seconde sont singulières.
- Un réseau étant une fonction régulière il va mal capturé la fonction de Green.



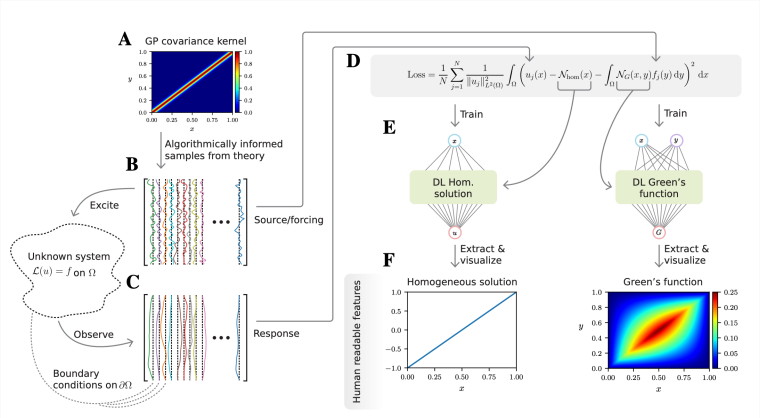

- Le GreenNet propose d'utiliser des couches d'activation rationnelles:

$$\sigma(x) = \frac{\sum_{i=0}^{3} a_i x^i}{1 + \left| \sum_{j=1}^{2} b_j x^j \right|}$$

- Cela permet d'obtenir de mieux capturer des singularités.

# GreenNet et EDP linéaire III

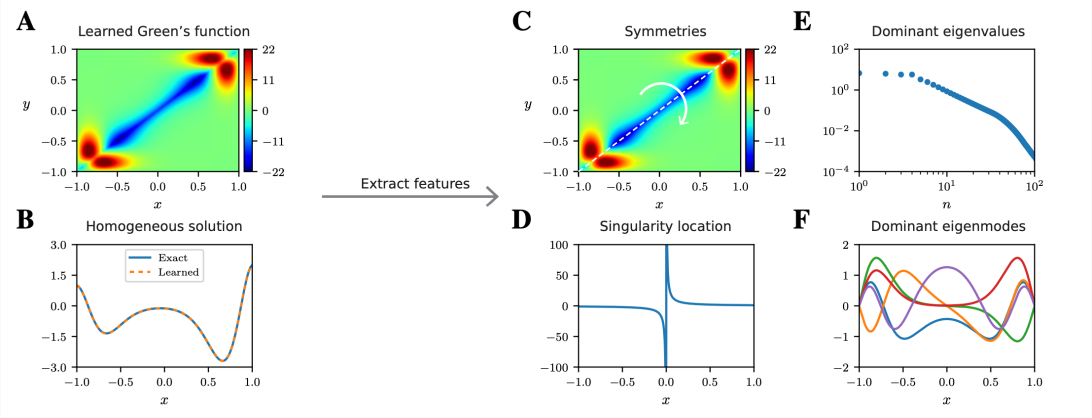
- Résumé de l'apprentissage:




- **Plusieurs exemples**: `https://greenlearning.readthedocs.io/en/latest/guide/gallery.html`

# GreenNet et EDP linéaire III

- Découverte de noyau



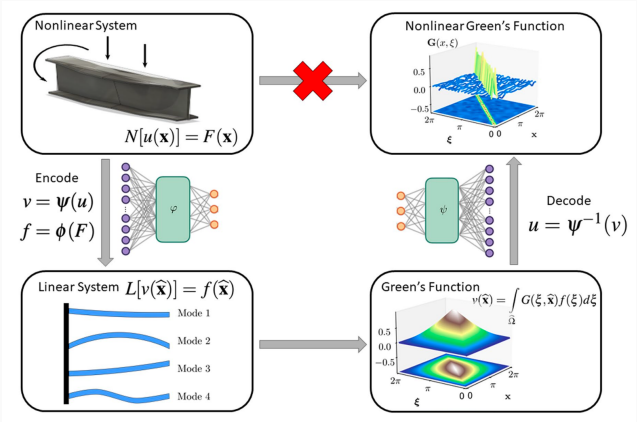

- **Plusieurs exemples**: `https://greenlearning.readthedocs.io/en/latest/guide/gallery.html`

# GreenNet et EDP nonlinéaire I

- La théorie de Green ne s'étend pas aux EDP nonlinéaires.

**Idée de DeepGreenNet**

on apprend une linéarisation du problème puis un opérateur de Green.



# GreenNet et EDP nonlinéaire II

- Pour le DeepGreenNet il est proposer d'apprendre un opérateur de Green $G_\theta$ et deux auto-encodeurs (Resnet).
- L'auto-encodeur $E_{\theta_f}$, $D_{\theta_f}$ s'appliquera à la source. L'auto-encodeur $E_{\theta_u}$, $D_{\theta_u}$ s'appliquera à la solution.
- Fonction de couts de base:

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

avec $\hat{f}_i(\mathbf{x}) = E_{\theta_f}(f_i(\mathbf{x}))$ et $\hat{u}_i(\mathbf{x}) = E_{\theta_u}(u_i(\mathbf{x}))$.

$$\mathcal{J}_f(\theta) = \sum_{i=1}^{m} \int_\Omega \| D_{\theta_f}(E_{\theta_f}(f_i(\mathbf{x}))) - f_i(\mathbf{x}) \|_2^2 \, d\mathbf{x}$$

$$\mathcal{J}_u(\theta) = \sum_{i=1}^{m} \int_\Omega \| D_{\theta_u}(E_{\theta_u}(u_i(\mathbf{x}))) - u_i(\mathbf{x}) \|_2^2 \, d\mathbf{x}$$

- Fonction de Coût additionnelle:

$$\mathcal{J}(\theta) = \sum_{i=1}^{m} \int_\Omega \| D_{\theta_u} \circ G_\theta \circ E_{\theta_f}(f_i(\mathbf{x})) - u_i(\mathbf{x}) \|_2^2 \, d\mathbf{x}$$

# GreenNet bas rang I

- Un des défauts du GreenNet est le coût d'évaluation. Si vous voulez calculer la solution en $m_1$ points, chaque point nécessitera le calcul de l'intégral avec $m_2$ points.
- Coût en $O(m_1 m_2)$ .

## GreenNet de Bas rang

On propose de prendre:

$$k_\theta(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{R} \phi_{\theta,i}(\mathbf{x}) \psi_{\theta,i}(\mathbf{y})$$

- Simplification:
  - Soit la formule de Green:

$$u(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y} = \sum_{i=1}^{R} \int_\Omega \phi_{\theta,i}(\mathbf{x}) \psi_{\theta,i}(\mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

  - Cela donne:

$$u(\mathbf{x}) = \sum_{i=1}^{R} \phi_{\theta,i}(\mathbf{x}) \int_\Omega \psi_{\theta,i}(\mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

$$u(\mathbf{x}) = \sum_{i=1}^{R} \langle f, \psi_i \rangle_{L^2} \phi_{\theta,i}(\mathbf{x})$$

# GreenNet bas rang II

## Couche du GreenNet bas rang

La couche est donnée par:

$$G_\theta(\mathbf{f}) = \sum_{k=1}^{R} \langle f, \psi_i \rangle_{L^2} \phi_{\theta,k}(\mathbf{x}_i)$$

avec

$$\langle f, \psi_i \rangle_{L^2} = \frac{1}{m} \sum_{j=1}^{m} \psi_{\theta,i}(\mathbf{x}_j) f(\mathbf{x}_j)$$

- Le coût d'évaluation est en $O(Rm_2 + Rm_1)$.
- on peut utiliser un ou deux réseaux pour les fonctions de bases.
- Le rang $R$ doit être assez grand.

## Remarque

Si vous apprenez G, effectuez un sampling aléatoire de **x** et diagonalisez la matrice, la décroissance des valeurs propres donnera une idée du rang à viser.

# DeepOnet I

- L'opérateur Neural DeepOnet un des premiers introduit dans la littérature.

**Idée**

Avoir un réseau qui prend en entrée $f(\mathbf{x})$ évalué à des points appelés "senseur" qui va prédire les coefficients $\langle f, \psi_i \rangle_{L^2}$.

**DeepONet pour approcher la fonction de Green**

Il s'agit d'un réseau à une couche, sans couche d'extrapolation et de projection, ou la couche principale est $G_\theta : H^s(\Omega)^p \to H^s(\Omega)^p$ est paramétrée de la façon suivante:

$$G_\theta(\mathbf{f}_1, ....\mathbf{f}_m) = \sum_{i=1}^{R} \alpha_i b_{\theta,i}(\mathbf{x})$$

avec $b_\theta$ un réseau de type MLP de $\mathbb{R}^d$ dans $\mathbb{R}^R$ et

$$t_\theta(\mathbf{f}_1, ....\mathbf{f}_m) \to (\alpha_1, ..., \alpha_K)$$

un réseau de $\mathbb{R}^{p \times m}$ dans $\mathbb{R}^R$

## DeepONet Général

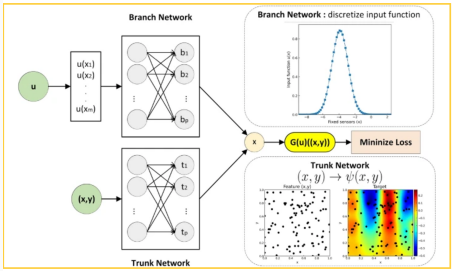
Il s'agit d'un réseau à une couche, sans couche d'extrapolation et de projection, ou la couche principale est $G_\theta : H^s(\Omega)^q \to H^s(\Omega)^p$ est paramétrée de la façon suivante:

$$G_\theta(\mathbf{a}_1, ....\mathbf{a}_m) = \sum_{i=1}^{R} \alpha_i b_{\theta,i}(\mathbf{x})$$

avec $b_\theta$ un réseau de type MLP de $\mathbb{R}^d$ dans $\mathbb{R}^R$ et

$$t_\theta(\mathbf{a}_1, ....\mathbf{a}_m) \to (\alpha_1, ..., \alpha_K)$$

un réseau de $\mathbb{R}^{q\times m}$ dans $\mathbb{R}^R$



## Approximation universelle

$X$ est un espace de Hilbert, $K_1 \subset X$ et $K_2 \subset \mathbb{R}^d$ sont deux ensembles compacts. $V$ est un ensemble compact dans $C(K_1)$, et $G$ est un opérateur non linéaire continu qui applique $V$ dans $C(K_2)$. Alors, pour tout $\epsilon > 0$, il existe $n, p, m > 0$ et des constantes $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$ tels que :

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left( \sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

avec $u \in V$ et $y \in K_2$.

## Shift-DeepONet

Il s'agit d'un réseau à une couche, sans couche d'extrapolation et de projection, ou la couche principale est $G_\theta : H^s(\Omega)^q \to H^s(\Omega)^p$ est paramétrée de la façon suivante:

$$G_\theta(\mathbf{a}_1, ....\mathbf{a}_m) = \sum_{i=1}^{R} \alpha_i b_{\theta,i}(\langle \mathbf{x}, \boldsymbol{\beta} \rangle + \gamma)$$

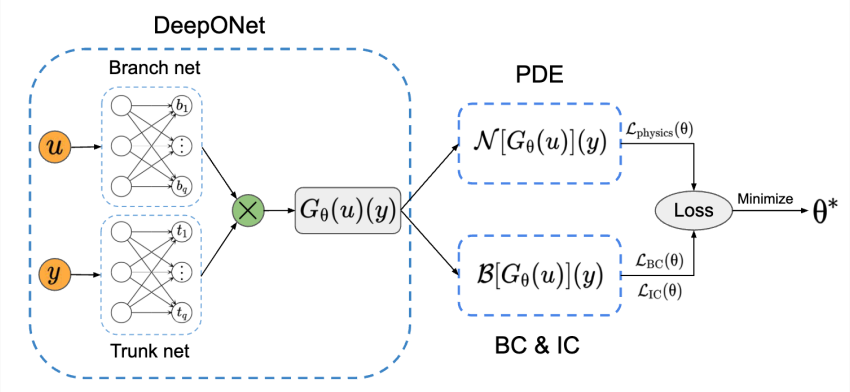
avec $b_\theta$ un réseau de type MLP de $\mathbb{R}^d$ dans $\mathbb{R}^R$ et

$$t_\theta(\mathbf{a}_1, ....\mathbf{a}_m) \to (\alpha_1, ..., \alpha_K), \quad a_\theta(\mathbf{a}_1, ....\mathbf{a}_m) \to (\beta_1, ..., \beta_d), \quad b_\theta(\mathbf{a}_1, ....\mathbf{a}_m) \to (\beta_1, ..., \beta_d)$$

# Approche physiquement informés

## Remarque

Les réseaux de type GreenNet et DeepOnet peuvent être évaluée en tout point **x**. On peut donc facilement ajouter une fonction de cout physique. Par-contre l'apprentissage devient très couteux.



# Opérateur neuronal sur graphe

- Jusqu'a présent on a vu que des opérateurs neuronaux à une couche. On va introduire des réseaux multi-couches utilisant des idées similaires.
- On va supposer que l'entrée et la sortie sont discrétisées sur un maillage.

## Opérateur neuronal sur graphe

Il s'agit d'un réseau multi-couche dont une couche de $\mathbb{R}^{d_l}$ dans $\mathbb{R}^{d_{l+1}}$ est donnée par

$$\mathbf{v}_{l+1}(\mathbf{v}) = \sigma\left(W\mathbf{v}_l(\mathbf{x}) + \frac{1}{|N(\mathbf{x})|}\sum_{\mathbf{y}\in N(\mathbf{x})} k_\theta(\mathbf{x},\mathbf{y},\mathbf{a}(\mathbf{x}),\mathbf{a}(\mathbf{y}))\mathbf{v}_l(\mathbf{y})\right)$$

avec $k_\theta$ un réseau de type MLP, Fourier etc et

$$N(\mathbf{x}) = \{\mathbf{y}, \text{tel que } \| \mathbf{y}-\mathbf{x} \| < r\}$$

- Le voisinage doit être choisi dans l'espace = indépendance à la résolution.
- Cette couche est une discrétisation de:

$$\sigma\left(W\mathbf{v}_l(\mathbf{x}) + \int_\Omega k_\theta(\mathbf{x},\mathbf{y},\mathbf{a}(\mathbf{x}),\mathbf{a}(\mathbf{y}))1_{\mathcal{B}(\mathbf{x},r)}\mathbf{v}_l(\mathbf{y})\right)$$

- On peut utiliser l'ensemble des voisins du maillage ou sous-échantillonner.

## Opérateur neuronaux Spetraux

# Méthodes spectrales

## Idée

On effectue un changement de base et on apprend dans la nouvelle base.

- On se donne une base $(\phi_1(\mathbf{x}), ...., \phi_n(\mathbf{x}))$ orthogonale.
- Dans la nouvelle base on a
$$u(\mathbf{x}) \approx \sum_{i=1}^{n} \langle u, \phi_i \rangle_{L^2} \phi_i(\mathbf{x})$$
- Pour apprendre une transformation dans une base on apprend une fonction de la forme
$$\alpha_i = \langle u, \phi_i \rangle_{L^2} \to \beta_i$$
- Un changement de résolution spatiale va peu changer l'approximation de $\langle u, \phi_i \rangle_{L^2}$
- Si on a assez de points les $\alpha_i$ bougent peu et la transformation dépend peu de la résolution.

## Couche spectrale

On propose la couche à noyau suivante:

$$\int_{\Omega} k_\theta(\mathbf{x}, \mathbf{y}) \mathbf{v}(\mathbf{y}) d\mathbf{y} = (\mathcal{B}^{-1} \circ \hat{\kappa}_\theta \circ \mathcal{B})(\mathbf{v})$$

avec $\mathcal{B}$ un changement de base dans une base $(\phi_1(\mathbf{x}), ...., \phi_n(\mathbf{x}))$ orthogonale et $\hat{\kappa}_\theta$ une application paramétrique qui transforme les modes de la fonction.

# FNO

- Les réseaux de Fourier sont les opérateurs neuraux les plus communs.

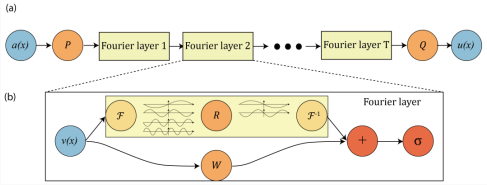
**FNO**

Les opérateurs neuronaux de Fourier utilisent comme base $\mathcal{B}$, la base de Fourier, utilise la FFT comme opérateur de changement de base et utilise:

$$\hat{\kappa}_\theta = K$$

avec $K \in \mathbb{C}^{d_{l+1} \times d_l \times n_f}$ une matrice entrainable avec $n_f$ le nombre de fréquences de Fourier qu'on va considérer.

- Si vos $d_l$ signaux ont $n$ points spatiaux le coût de la FFT et FFT inverse est en $O(d_l n log n)$ et $O(d_{l+1} n log n)$
- Le coût de l'application du filtre est de l'ordre $O(d_l d_{l+1} n_f)$. Ce coût est raisonnable si $n_f << n$.



# FNO en pratique

- Hyper-paramètres du FNO:
  - Nombre de couche (en général entre 3 et 5),
  - Largeur: nombre de fonctions générées par la couche d'extrapolation.
  - Nombre de modes: nombre de modes de Fourier sur lequel on va appliquer les filtres. Entre 20 et 40 en général.
- FNO et la dimension:
  - Le nombre de modes augmente vite avec la dimension. Les FNO deviennent couteux en grande dimension d'espace.
  - Le nombre de modes est en $O(n_{fx}^d)$ et donc la cout de la couche devient $O(d_l d_{l+1} n_{fx}^d)$

## FNO vs FNO Factorisé

Le FNO en dimension $d$ utilise:

$$\int_\Omega k_\theta(\mathbf{x},\mathbf{y})\mathbf{v}(\mathbf{y})d\mathbf{y} = (\mathcal{F}_d^{-1} \circ \hat{\kappa}_{\theta,d} \circ \mathcal{F}_d)(\mathbf{v})$$

Le FNO factorisé utilise:

$$\int_\Omega k_\theta(\mathbf{x},\mathbf{y})\mathbf{v}(\mathbf{y})d\mathbf{y} = \sum_{i=1}^{d} (\mathcal{F}_i^{-1} \circ \hat{\kappa}_{\theta,i} \circ \mathcal{F}_i)(\mathbf{v})$$

# FNO physiquement informés

- Comment étendre un réseau à des fonctions de coût physiquement informés ?

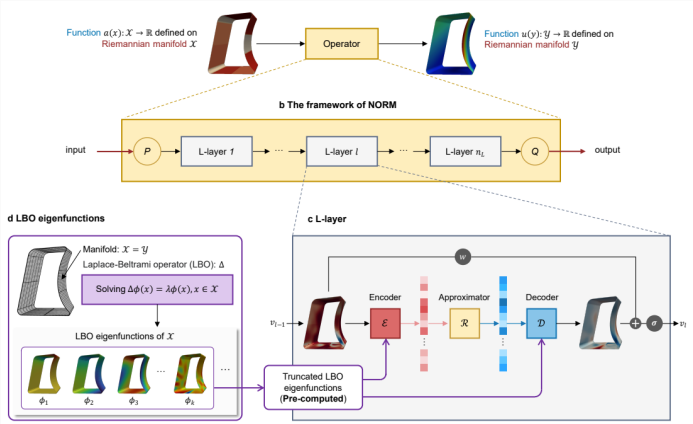
**Remarque**

Le FNO utilise des FFT donc on récupère la solution que aux $n$ points ou est localisé le signal discret d'entrée. Comment calculer la dérivée ?

- La sortie est donnée par
$$u(\mathbf{x}) = \mathcal{P}(v_L)(\mathbf{x}) = \mathcal{P}((\mathcal{W}_L v_{L-1})(\mathbf{x}) + \mathcal{K}_L v_{L-1}(\mathbf{x}))$$
- **Idée**: on va faire une transformée de Fourier inverse continue à la fin.
- On a donc:
$$u(x) = \mathcal{P} \circ \mathcal{F}^{-1}(\kappa_\theta \cdot (\mathcal{F} v_{L-1}))(x) = \mathcal{P}\left(\frac{1}{k_{\max}} \sum_{k=0}^{k_{\max}} \left(\kappa_\theta^k (\mathcal{F} v_{L-1})_k\right) \exp \frac{i2\pi k}{D}(x)\right)$$
- et donc
$$u'(x) = Q'(v_L(x)) \cdot \frac{1}{k_{\max}} \sum_{k=0}^{k_{\max}} \left(\kappa_\theta^k (\mathcal{F} v_{L-1})_k\right) \exp' \frac{i2\pi k}{D}(x)$$
- On peut aussi calculer la dérivée dans l'espace de Fourier:
$$u' = \mathcal{P}'(v_L) \cdot \mathcal{F}^{-1}\left(\frac{i2\pi}{D} K \cdot (\mathcal{F} v_L)\right)$$

# Geométries complexes: LNO

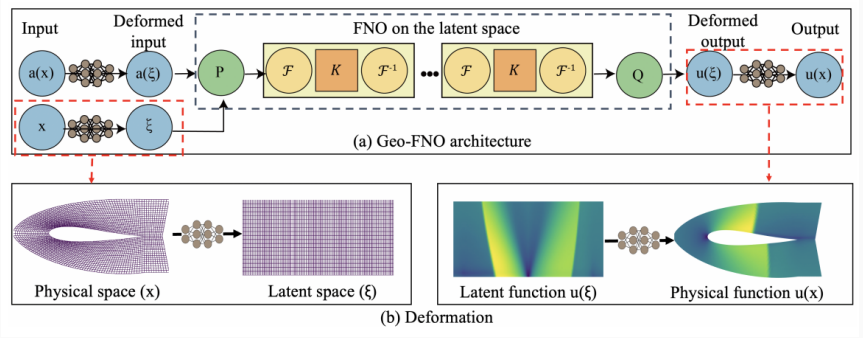
- Comme on a vu dans le cours sur les GNN, on peut généraliser Fourier sur des géométries complexes.
- On va donc calculer les modes de Fourier de la géométrie (FE) puis construire une transformée de Fourier associée.



# Geométries complexes: GNO

## Idée

On apprend une déformation du domaine physique vers un domaine rectangulaire, on applique une FNO classique et on applique la déformation inverse au résultat.




- On définit un domaine physique $\Omega$ et un ensemble de point $\Omega_h = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$. On définit le domaine de calcul $D$ et un ensemble de point $D_h = \{\xi_1, ..., \xi_n\}$

# Géométries complexes: GNO II

## Apprendre la déformation

La deformation est construire a partir d'un réseau inversible (par rapport au première variables)

$$\phi_\Omega(\xi_1, ..., \xi_d, \mathbf{a}) \to (x_1, ..., x_d)$$

avec **a** les paramètres de la géométrie. On choisit un réseau MLP ou de Fourier de la forme:

$$\phi_\Omega^{-1}(\mathbf{x}) = \mathbf{x} + f_\theta(\mathbf{x}, \mathbf{a})$$

- On va construire la transformée de Fourier sur le domaine physique.
- On le fait par changement de variable:

$$\begin{aligned}(\mathcal{F}_\Omega v)(k) &:= \int_D v(\xi) e^{-2i\pi\langle \xi, \mathbf{k}\rangle} d\xi \\ &= \int_\Omega v(\mathbf{x}) e^{-2i\pi\left\langle \phi_\Omega^{-1}(\mathbf{x}), \mathbf{k}\right\rangle} \left|\det\left[\nabla_x \phi_\Omega^{-1}(\mathbf{x})\right]\right| d\mathbf{x} \approx \frac{1}{|\Omega_h|} \sum_{\mathbf{x}\in\Omega_h} m(\mathbf{x}) v(\mathbf{x}) e^{-2i\pi\left\langle \phi_\Omega^{-1}(\mathbf{x}), \mathbf{k}\right\rangle}\end{aligned}$$

- L'inverse se définit pat l'identité $\mathcal{F}_\Omega^{-1} \circ \mathcal{F}_\Omega = I_d$

$$\left(\mathcal{F}_\Omega^{-1}\hat{v}\right)(\mathbf{x}) = \left(\mathcal{F}^{-1}\hat{v}\right)\left(\phi_\Omega^{-1}(\mathbf{x})\right) = \sum_k \hat{v}(\mathbf{k}) e^{2i\pi\left\langle \phi_\Omega^{-1}(\mathbf{x}), \mathbf{k}\right\rangle}$$

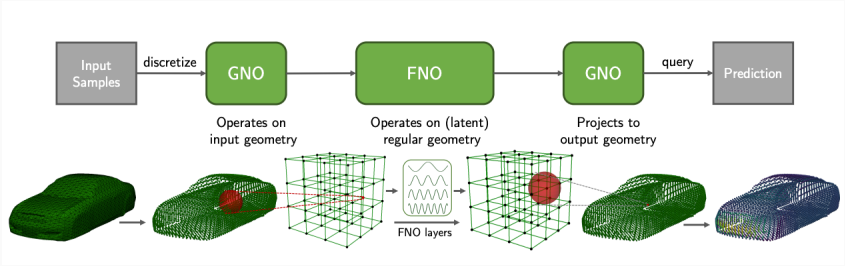
- Réseau final:

$$\mathcal{G}_\theta := \mathcal{Q} \circ \left(W_L + \mathcal{F}_\Omega{}^{-1} \circ \kappa_\theta \circ \mathcal{F} + b_L\right) \circ \cdots \circ \left(W_1 + \mathcal{F}^{-1} \circ \kappa_\theta \circ \mathcal{F}_\Omega + b_L\right) \circ \mathcal{P}$$

# Géométries complexes: GINO

## Idée

Construire une grille qui contient $\Omega$ et apprendre une transformation locale entre la grille et $\Omega_h$.




## Couche de projection géométrique

Pour passer d'une géométrie physique $\Omega$ à un domaine de calcul on va utiliser la couche suivant:

$$\mathbf{v}_0(\mathbf{x}) = \int_{B_r(\mathbf{x})} \kappa_\theta(\mathbf{x}, \mathbf{y}) \mathbf{a}(\mathbf{y}) \mathrm{d}\mathbf{y}, \qquad \to \mathbf{v}_0(\mathbf{x}_i^D) = \sum_{j=1}^{M} \kappa_\theta(\mathbf{x}_i, \mathbf{y}_j^\Omega) \mathbf{a}(\mathbf{y}_j^\Omega)$$

On peut inverser le rôle de $\Omega$ et $D$

# Opérateur neuraux spectraux

## Idée

On peut utiliser une autre base que Fourier: Chebyshev, Legendre et garder l'architecture précédente.

## Autre architecture

On fait l'ensemble des transformations dans l'espace spectral. On parle de SNO (spectral neural operator). On a donc

$$\mathcal{G}_\theta = \mathcal{B}^{-1} \circ MLP_\theta \circ \mathcal{B}$$

- En pratique on enchaine donc les couches en restant dans l'espace spectral.
- Les couche linéaires sont donc, comme précédemment, apprisent dans l'espace spectral. Les modes des différents signaux sont totalement mélangés.
- Les activations sont aussi appliquées dans le domaine spectral.

## Opérateur neuronaux et attention

## Introduction

Les transformers sont un type de réseaux de neurones introduits pour le traitement du langage naturel (traduction) puis étendus aux problèmes de traitement du signal.

- **Traduction 1er stratégie**: le modèle SeqtoSeq.
- Il s'agit d'un modèle qui va transformer une séquence (une phrase en français) en une autre séquence (en anglais).
- Si on se donne comme séquence d'entrée $(x_1, ..., x_n)$ et $(y_1, ..., y_n)$ comme séquence de sortie alors l'objectif est de construire la loi de probabilité:

$$\mathbb{P}((y_1, ..., y_n) \mid (x_1, ..., x_n))$$

## Modèles SeqtoSeq

Deux réseaux: l'encodeur et le décodeur. Le rôle de l'encodeur est de transformer

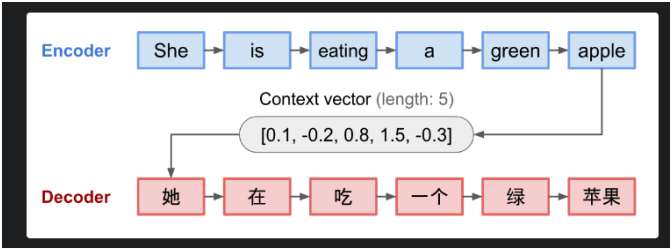
$$E : (x_1, x_2, \ldots, x_n) \to c$$

la séquence en un vecteur $c$ (longueur fixe), le vecteur de contexte. Rôle du décodeur :

$$D : (c, y_1, y_2, \ldots, y_{t-1}) \to y_t$$

est de prédire la suite de la séquence de sortie en utilisant les éléments précédents et vecteur de contexte.

# SeqtoSeq




- Après le décodeur on applique un MLP afin d'obtenir la loi de probabilité finale.
- En pratique;
  - Pour chacun des $1 \leqslant k \leqslant K$ valeurs que pourrait prendre $y_t$, le réseau dense va calculer

$$\prod_{t=1}^{p} \mathbb{P}(y_t = k \mid h, y_1, y_2, \ldots, y_{t-1}),$$

  en appliquant un softmax dont on apprendra les poids $(\mathbf{w}_j)_{1 \leqslant j \leqslant K}$:

$$\mathbb{P}(y_i = k \mid \mathbf{v}) = \text{Softmax}(\mathbf{v} \cdot \mathbf{w}_k) = \frac{e^{\mathbf{v} \cdot \mathbf{w}_k}}{\sum_{j=1}^{K} e^{\mathbf{v} \cdot \mathbf{w}_j}}$$

- Ensuite on maximise le maximum de vraisemblance et cela revient a minimiser l'entropie croisée classique en classification multi-classe.

# 1er Attention

## Remarque

- Encodeur et le décodeur des réseaux de type LSTM/GRU,
- **Défaut**: si la séquence d'entrée est longue, le fait que le vecteur de contexte soit de taille fixe fait que les informations situées à la fin de la séquence vont avoir tendance à avoir un poids trop important dans le vecteur de contexte.

## Idée

L'idée de l'attention consiste à créer des raccourcis entre le vecteur de contexte et tous les éléments de la séquence. Les poids associés aux raccourcis sont personnalisables pour chaque élément de sortie. Comme le vecteur de contexte a accès à l'ensemble de la séquence d'entrée, nous n'avons pas à nous soucier de l'oubli.

- **1er construction**: l'encodeur est un RNN bi-directionnelle. Deux RNN qui parcourent les données dans les deux sens

$$\begin{cases} \overrightarrow{h}_t = \sigma_h\left(W_{\overrightarrow{h}} x_t + U_{\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \\ \overleftarrow{h}_t = \sigma_h\left(W_{\overleftarrow{h}} x_t + U_{\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \end{cases}$$

# 1er Attention II

- On construit l'état lalent comme concaténation des deux états $h_t = [\overrightarrow{h}_t, \overrightarrow{h}_t]$
- Le décodeur prendre une forme classique:

$$s_t = f_\theta(s_{t-1}, y_{t-1}, c_t)$$

avec $s_t$ l'état latent du décodeur.

- Comment construire le vecteur de contexte:

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$

Cela revient donc a dire que le vecteur de contexte est une combinaison linéaire des états cachés de l'encodeur.

- Les coefficients $\alpha_{t,i}$ ? L'idée est de construire le coefficient $\alpha_{t,i}$ de façon à ce qu'il mesure à quel point la sortie $y_t$ et l'entrée $x_i$ se correspondre bien ( on parle d'alignement).
- Pour cela on utilise les états cachés associés à $y_t$ et $x_i$

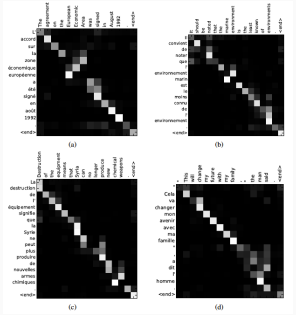
$$\alpha_{ti} = \frac{score(s_{t-1}, h_i)}{\sum_{k=1}^{n} score(s_{t-1}, h_k)}, \qquad score(s_{t-1}, h_i) = v_a^t \tanh(W_a[s_{t-1}, h_i])$$

avec $v_a^t, W_a$ des paramètres entraînables.

# 1er Attention III

## Interprétation

L'ensemble des coefficients $\alpha_{ti}$ forment une matrice qui mesure l'alignement entre tous les états cachés du décodeur et encodeur et donc entre tous les éléments des séquences. On peut l'interpréter comme une matrice de corrélation entre les éléments des deux séquences.



- l'attention au contenu: $score(s_{t-1}, h_i) = cosine([s_{t-1}, h_i])$
- l'attention basée sur le produit scalaire: $score(s_{t-1}, h_i) = \frac{s_{t-1} h_i}{\sqrt{n}}$
- L'attention dite générale: $score(s_{t-1,h_i}) = s_{t-1} W_a h_i$, avec $W_a$ une matrice entraînable.

# Intra Attention

## Principe de l'auto-Attention

Un mécanisme d'auto-attention est un mécanisme qui met en relation différentes positions d'une même séquence afin de calculer une représentation de cette même séquence.

## Opérateur d'auto-intention

On se donne une entrée $X \in \mathbb{R}^{n,d}$. Soit $W_Q, W_K, W_V \in \mathbb{R}^{d,d}$ des matrices de poids entraînable. On définit trois quantités:

- La requête: $Q = W_q X$
- La clé: $K = W_K X$
- La valeur: $V = W_V X$

L'auto attention est donnée par $Att(Q, K, V) = \text{Softmax}(\frac{QK^t}{\sqrt{n}})V$

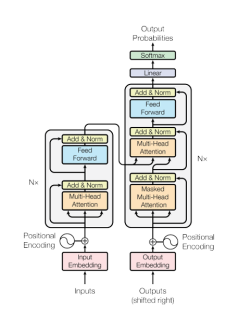
- $Q$ représente l'information que l'on souhaite corréler avec les autres. On parle de requête (comme une information que l'ont souhaite chercher dans un dictionnaire).
- L'enjeu est d'aligner $Q$ au mieux avec des clés qui indexent les données de la séquence $X$ (comme si on cherchait la clé à une requête dans un dictionnaire).
- **Softmax**: transforme cela en loi de probabilité ou plus une clé est alignée à la requête plus la probabilité est grande.
- On va donc renvoyer une matrice qui privilégie au mieux les valeurs dont l'alignement avec une donnée cible est le meilleur .

## Auto attention multi-tête

On se donne une entrée $X \in \mathbb{R}^{n,d}$. Soit $W_Q^i, W_K^i, W_V^i \in \mathbb{R}^{d,d}, \quad \forall i < n$ des matrices de poids entraînable. La transformation d'attention multi tête est donnée par

$$MTATT(X) = concat(ATT(Q_1, K_1, V_1), ...., ATT(Q_m, K_m, V_m))W_0$$

avec $W_0$ une matrice de poids entraînables.




- L'encodeur d'un transformeur est composé de plusieurs blocs qui enchaînent deux sous blocs, un mécanisme d'attention multi-tête et un MLP.
- Le décodeur prend la séquence de sortie, applique aussi une série de blocs qui enchaîne des mécanismes d'attention. Le premier sous-bloc d'attention est dit masqué, car il utilise les données du fur de la séquence $(y_1, ...y_n)$, le second sous bloc utilise aussi les résultats de l'encodeur.

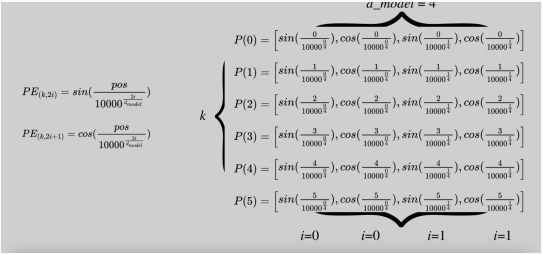
# Transformeur II

- La position des mots sont les éléments essentiels de toute langue. Les (RNN) analysent une phrase mot par mot de manière séquentielle.
- L'architecture Transformer évite la méthode de récurrence des RNN ce qui permet d'accélérer le temps d'apprentissage et de capturer des dépendances plus longues dans une phrase.
- Le transformeur traite l'ensemble de la séquence donc ignore la position des mots. Pour ajouter cette information on commence par un encodage positionnel.

## Encodage de positions

- Les mots sont encodés par un vecteur de taille $d$. L'encodage de positions du mot $k$ est donné par un vecteur de taille $d$ donnée par

$$PE_{k,2i} = \sin(\frac{f}{\frac{2i}{d}}), \quad PE_{k,2i+1} = \cos(\frac{f}{\frac{2i}{d}})$$



## Bloc d'un transformeur

Soit $\mathbf{x} \in \mathbb{R}^{n\times d}$ alors la transformation $f_\theta(\mathbf{x}) = \mathbf{z}$ est donnée par:

$$\begin{cases} Q^{(h)}(\mathbf{x}_i) & = W_{h,q}^T \mathbf{x}_i, \quad K^{(h)}(\mathbf{x}_i) = W_{h,k}^T \mathbf{x}_i, \quad V^{(h)}(\mathbf{x}_i) = W_{h,v}^T \mathbf{x}_i, \quad W_{h,q}, W_{h,k}, W_{h,v} \in \mathbb{R}^{d\times d}, \\ & \alpha_{i,j}^{(h)} = \mathrm{softmax}_j\left(\frac{\left\langle Q^{(h)}(\mathbf{x}_i), K^{(h)}\left(\mathbf{x}_j\right)\right\rangle}{\sqrt{n}}\right), \\ & \mathbf{u}_i' = \sum_{h=1}^H W_{c,h}^T \sum_{j=1}^n \alpha_{i,j}^{(h)} V^{(h)}\left(\mathbf{x}_j\right), \quad W_{c,h} \in \mathbb{R}^{d\times d}, \\ & \mathbf{u}_i = \mathrm{LayerNorm}\left(\mathbf{x}_i + \mathbf{u}_i'; \gamma_1, \beta_1\right), \quad \gamma_1, \beta_1 \in \mathbb{R}^d, \\ & \mathbf{z}_i' = W_2^T \sigma\left(W_1^T \mathbf{u}_i\right), \quad W_1 \in \mathbb{R}^{d\times m}, W_2 \in \mathbb{R}^{m\times d}, \\ & \mathbf{z}_i = \mathrm{LayerNorm}\left(\mathbf{u}_i + \mathbf{z}_i'; \gamma_2, \beta_2\right), \quad \gamma_2, \beta_2 \in \mathbb{R}^d. \end{cases}$$

Les étapes de normalisation par paquet sont définies par

$$\begin{cases} \mathrm{LayerNorm}(\mathbf{z}; \gamma, \beta) = \gamma \frac{(\mathbf{z} - \mu_\mathbf{z})}{\sigma_\mathbf{z}} + \beta, \quad \gamma, \beta \in \mathbb{R}^k. \\ \mu_\mathbf{z} = \frac{1}{k}\sum_{i=1}^k \mathbf{z}_i, \quad \sigma_\mathbf{z} = \sqrt{\frac{1}{k}\sum_{i=1}^k (\mathbf{z}_i - \mu_\mathbf{z})^2}. \end{cases}$$

# Attention continue

- On propose d'utiliser les transformeurs pour construire des opérateurs neuronaux.
- On va commencer par construire des opérateurs d'attention continue

## Probabilité pour l'attention

Soit $u : \Omega \to \mathbb{R}$ une fonction indexée par $\mathbf{x} \in \Omega$ (pas forcement l'espace). On définit une mesure de probabilité sur $\Omega$, paramétrée par $(u, \mathbf{x})$, via la densité $p(\cdot; u, \mathbf{x}) : \Omega \to \mathbb{R}^+$ définie par

$$p(\mathbf{y}; u, \mathbf{x}) = \frac{\exp\big(\langle Qu(\mathbf{x}), Ku(\mathbf{y})\rangle_{\mathbb{R}^{d_K}}\big)}{\int_D \exp\big(\langle Qu(\mathbf{x}), Ku(\mathbf{s})\rangle_{\mathbb{R}^{d_K}}\big)\mathrm{d}\mathbf{s}}, \quad \forall \mathbf{y} \in \Omega$$

## Attention continue

On définit l'opérateur d'auto-attention comme une transformation de fonctions de $\Omega$ vers $\mathbb{R}^{d_u}$ vers les fonctions $\mathrm{A}(u)$ de $\Omega$ vers $\mathbb{R}^{d_v}$

$$\mathrm{A}(u)(\mathbf{x}) = \mathbb{E}_{\mathbf{y}\sim p(\mathbf{y};u,\mathbf{x})}[Vu(\mathbf{y})], \quad \forall \mathbf{x} \in \Omega$$

équivalent à

$$\mathrm{A}(u)(\mathbf{x}) = \int_\Omega \frac{\exp\big(\langle Qu(\mathbf{x}), Ku(\mathbf{y})\rangle_{\mathbb{R}^{d_K}}\big)}{\int_D \exp\big(\langle Qu(\mathbf{x}), Ku(\mathbf{s})\rangle_{\mathbb{R}^{d_K}}\big)\mathrm{d}\mathbf{s}} Vu(\mathbf{y})d\mathbf{y}, \quad \forall \mathbf{y} \in D$$

# Attention continue II

- On peut la réécrire comme:

$$\mathrm{A}(u)(\mathbf{x}) = \int_\Omega k_\theta(u(\mathbf{x}), u(\mathbf{y}))u(\mathbf{y})d\mathbf{y}, \quad \forall \mathbf{y} \in D$$

  avec $\theta = (K, Q, V)$ des opérateurs linéaires. On va discuter les choix de $(K, Q, V)$ en dernier.
- On a donc une couche d'opérateur neuronale mais nonlinéaire.

## Lien avec l'attention

L'attention peut être vu comme une discrétisation de Monte-Carlo de l'attention continue.

## Attention multi-tête continue

Elle est donnée par

$$(\mathrm{A}_{\text{MultiHead}}(\mathbf{v}))(\mathbf{x}) = W_{\text{MultiHead}}(\mathrm{A}(\mathbf{v}; \theta_1)(\mathbf{x}), \ldots, \mathrm{A}(\mathbf{v}; \theta_H)(\mathbf{x})),$$

avec $W_{\text{MultiHead}}$ une matrice de poids.

## Opérateur neuronal transformeur

Il s'agit d'un réseau de la forme

$$\mathrm{G}(u, \theta) := (\mathrm{T}_{\text{out}} \circ \mathrm{E}_L \circ \mathrm{T}_{\text{in}})(u, \theta),$$

Les opérateurs sont des opérateurs de projection et extrapolation.

# Opérateur de type transformeur I

- Comme chez les transformeurs classiques, on ajoute des couches de normalisation et un MLP.

**Bloc d'un opérateur neuronal de type transformeur**

Le bloc est donné par

$$
\begin{aligned}
\mathbf{v}^{(l-1)}(\mathbf{x}) &\leftarrow \mathrm{W}_1\mathbf{v}^{(l-1)}(\mathbf{x}) + \mathrm{A}_{\text{MultiHead}}\left(\mathbf{v}^{(l-1)}\right),\\
\mathbf{v}^{(l-1)}(\mathbf{x}) &\leftarrow \mathrm{F}_{\text{LayerNorm}}\left(\mathbf{v}^{(l-1)}(\mathbf{x})\right),\\
\mathbf{v}^{(l-1)}(\mathbf{x}) &\leftarrow \mathrm{W}_2\mathbf{v}^{(l-1)(\mathbf{x})} + \mathrm{F}_{\mathrm{NN}}\left(\mathbf{v}^{(l-1)}(\mathbf{x})\right),\\
\mathbf{v}^{(l)}(\mathbf{x}) &\leftarrow \mathrm{F}_{\text{LayerNorm}}\left(\mathbf{v}^{(l-1)}(\mathbf{x})\right),
\end{aligned}
\qquad \text{avec}
$$

$$
\left(F_{\text{LayerNorm}}\left(\mathbf{v};\gamma,\beta\right)(\mathbf{x})\right)_k = \gamma_k \cdot \frac{(\mathbf{v}(\mathbf{x}))_k - m(\mathbf{v}(\mathbf{x}))}{\sqrt{\sigma^2(\mathbf{v}(\mathbf{x})) + \epsilon}} + \beta_k,
$$

- $\mathrm{T}_{\text{in}}$ et $\mathrm{T}_{\text{out}}$ sont des MP locaux de la forme:

$$
MLP(\mathbf{x}, \mathbf{u}_{in}(\mathbf{x})) \rightarrow \mathbf{v}^0
$$

# Opérateur de type transformeur II

## Attention 1er choix

Soit $\mathbf{v}_l(\mathbf{x}) \in H^p$ On va apprendre les matrices:

$$V, Q, K \in \mathcal{M}_{\frac{p}{H}, p}(\mathbb{R})$$

## Attention 2ème choix

Soit $\mathbf{v}_l(\mathbf{x}) \in H^p$ On va apprendre les opérateurs linéaires:

$$V, Q, K = \int_\Omega k_\theta(\mathbf{x}, \mathbf{y}) \mathbf{v}_l(\mathbf{y}) d\mathbf{y}$$

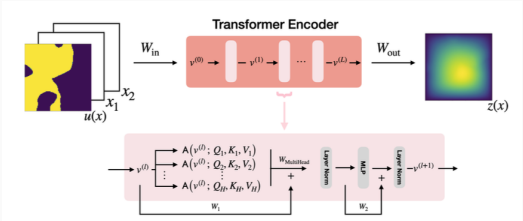



Figure 1: Transformer Neural Operator.

# Opérateur de type transformeur II

## Attention 1er choix

Soit $\mathbf{v}_l(\mathbf{x}) \in H^p$ On va apprendre les matrices:

$$V, Q, K \in \mathcal{M}_{\frac{p}{H}, p}(\mathbb{R})$$

## Attention 2ème choix

Soit $\mathbf{v}_l(\mathbf{x}) \in H^p$ On va apprendre les opérateurs linéaires:

$$V, Q, K = \int_\Omega k_\theta(\mathbf{x}, \mathbf{y}) \mathbf{v}_l(\mathbf{y}) d\mathbf{y}$$

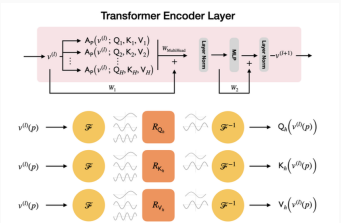



Figure 5: Encoder Layer of the Fourier Attention Neural Operator.