Séminaire MecaPhyGeo – 13-14 nov 2019 – Aix en Provence

IA & Mécanique
Le complexe du Deep ?

Stéphane BONELLI
UMR Recover Irstea, Aix-Marseille University

Conclusion

- L'IA et les réseaux de neurone datent de plusieurs dizaines d'années (<1940).
  Le Deep Learning est récent (2012, ImageNet). C'est une révolution pour la recherche. C'est un changement de paradigme total pour les mathématiciens et la science des données.

- On en est au tout début. Les développements sont pour l'heure de l'ordre de l'artisanat expérimental, sans fondement.
  On comprend très mal le fonctionnement d'un point de vue mathématique (2 couches actuellement).

- Les résultats obtenus sont spectaculaires dans tous les domaines explorés.
  Le Deep Learning a permis d'adresser des problèmes jusqu'alors inabordables.

- La Mécanique ne s'est pas encore approprié le Deep Learning.
  C'est sans aucun doute une technique qui va être d'usage courant et incontournable à moyen terme (≈5 ans).

- Cette technique permet d'adresser efficacement les très grands nombres de données (10^6 à 10^9), mais également les fortes non linéarités. Les premières applications en Mécanique (2016/2019) :
  - i) analyse de mesures expérimentales (PIV, DEM, ...)
  - ii) modélisation prédictive "temps réel" (tsunami, ...)
  - iii) modélisation fortement non-linéaire (turbulence, élastoplasticité, DEM, homogénisation FEM^2, FEM/DEM, ...)

- Le Deep Learning est tout à fait abordable,
  mais il nécessite une méthodologie rigoureuse sous peine de résultats très décevants.

Deep Learning : systèmes de très grande dimension

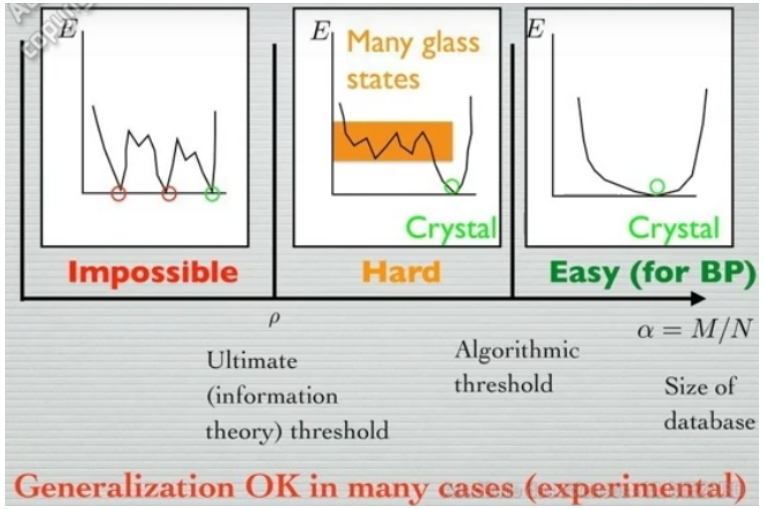
Objectif : construire un modèle sans programmation explicite des équations

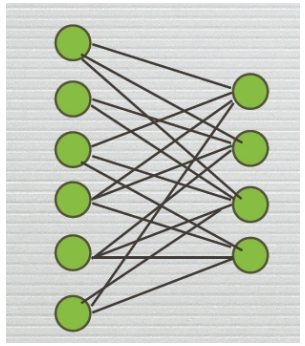
Deep Learning : grande similitude avec la physique statistique des systèmes désordonnés
grande similitude (et plus puissant) que la théorie des ondelettes

Question classique en physique statistique : construire une densité de probabilité

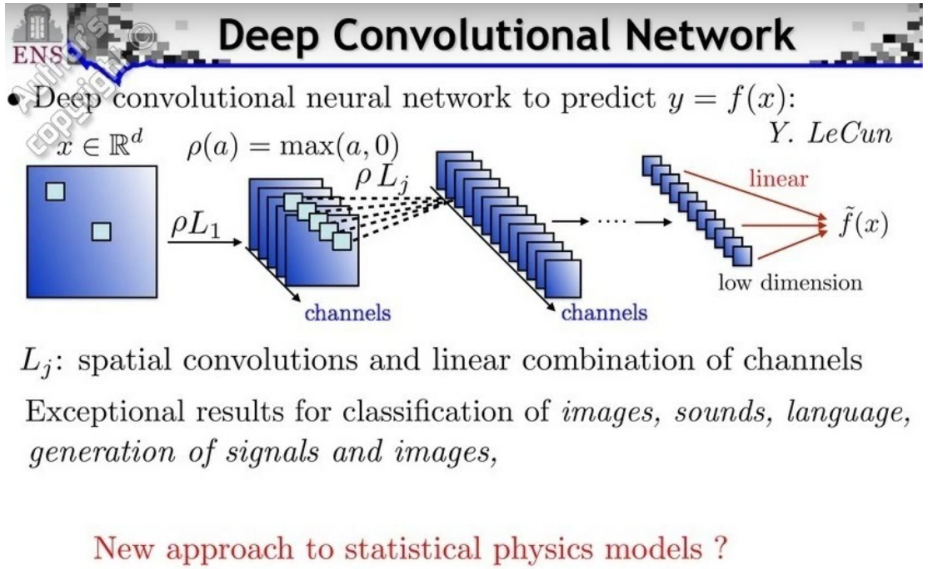
La structuration des réseaux de neurones profonds est assimilable à un processus de changement d'échelle sur plusieurs échelles intermédiaires décorrélées mais en interaction (« l'émergence » en physique statistique)

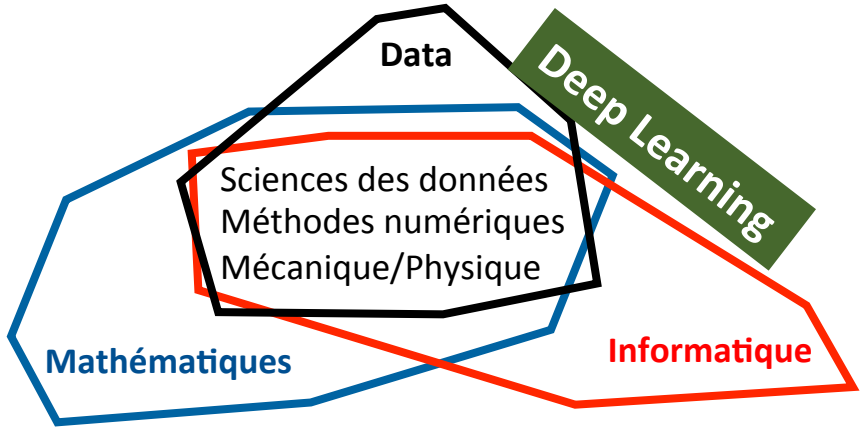
Transition de phase





Réseau de neurone des années 2000

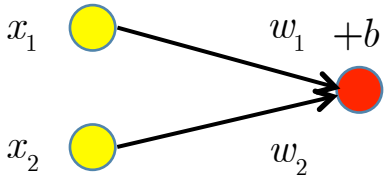




https://www.college-de-france.fr/site/stephane-mallat/course-2018-01-24-09h30.htm
https://www.academie-sciences.fr/fr/Colloques-conferences-et-debats/physique-statistique-intelligence-artificielle.html

Quelques éléments de description (FFNN)

Régression linéaire



$$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

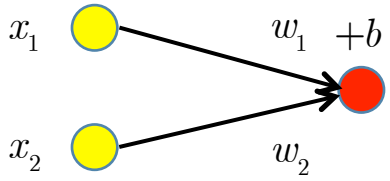
$$\{X_1^i, X_2^i, Y^i\}_{i=1,..,n}$$

Jeu de données

$$\min_{(w_1,w_2,b)} L(w_1, w_2, b) = \sum_{i=1}^{n} \left\| Y_i - f(X_1^i, X_2^i) \right\|^2$$

Quelques éléments de description (FFNN)

Régression linéaire

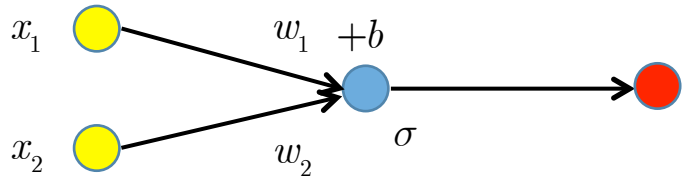


$$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

$$\{X_1^i, X_2^i, Y^i\}_{i=1,..,n}$$

Jeu de données

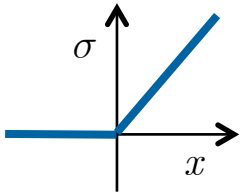
$$\min_{(w_1,w_2,b)} L(w_1, w_2, b) = \sum_{i=1}^{n} \left\| Y_i - f(X_1^i, X_2^i) \right\|^2$$

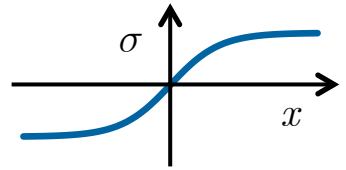


1 seul neurone

$$f(x_1, x_2) = \sigma(w_1 x_1 + w_2 x_2 + b)$$

σ n'est pas un polynôme

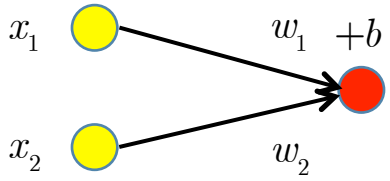




Régression non linéaire

Quelques éléments de description (FFNN)

Régression linéaire



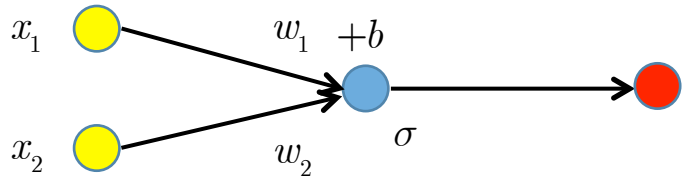
$$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

$$\{X_1^i, X_2^i, Y^i\}_{i=1,..,n}$$

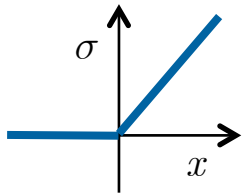
Jeu de données

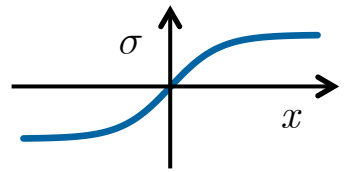
$$\min_{(w_1,w_2,b)} L(w_1, w_2, b) = \sum_{i=1}^{n} \left\| Y_i - f(X_1^i, X_2^i) \right\|^2$$

1 seul neurone



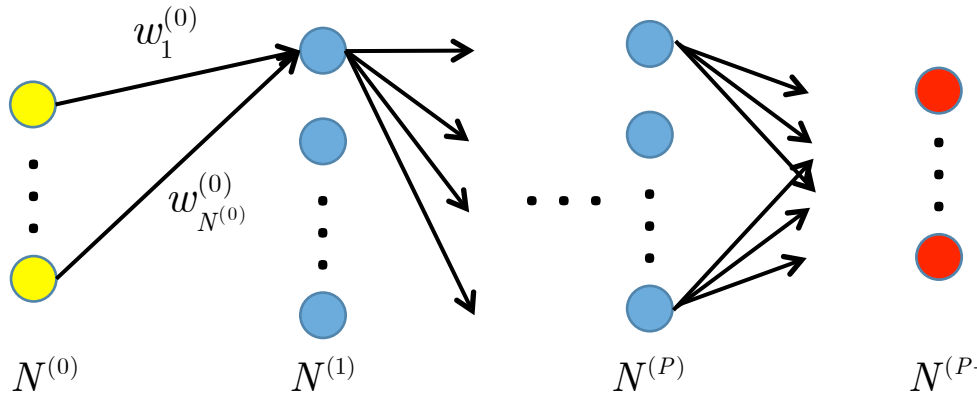
$$f(x_1, x_2) = \sigma(w_1 x_1 + w_2 x_2 + b)$$





Régression non linéaire

Réseau profond



$$\mathbf{x} = \{x_i^{(0)}\}_{1,..,N^{(0)}}$$

Entrée
(Donnée)

$$\mathbf{f}(\boldsymbol{\theta}, \mathbf{x}) = \mathbf{x}^{(P+1)}$$

$$\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$$

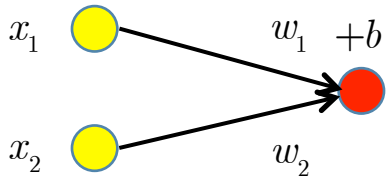
Sortie
(Fonction)

$$\mathbf{x}^{(1)} = \sigma(\mathbf{W}^{(1)}.\mathbf{x}^{(0)} + \mathbf{b}^{(1)})$$

$$\mathbf{x}^{(P+1)} = \sigma(\mathbf{W}^{(P+1)}.\mathbf{x}^{(P)} + \mathbf{b}^{(P+1)})$$

Quelques éléments de description (FFNN)

Régression linéaire



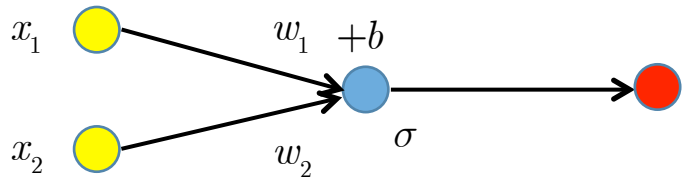
$$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

$$\{X_1^i, X_2^i, Y^i\}_{i=1,..,n}$$

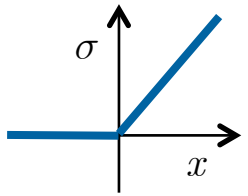
Jeu de données

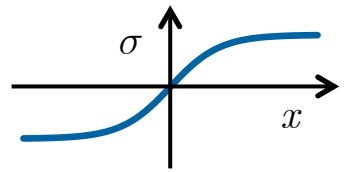
$$\min_{(w_1,w_2,b)} L(w_1, w_2, b) = \sum_{i=1}^{n} \left\| Y_i - f(X_1^i, X_2^i) \right\|^2$$

1 seul neurone



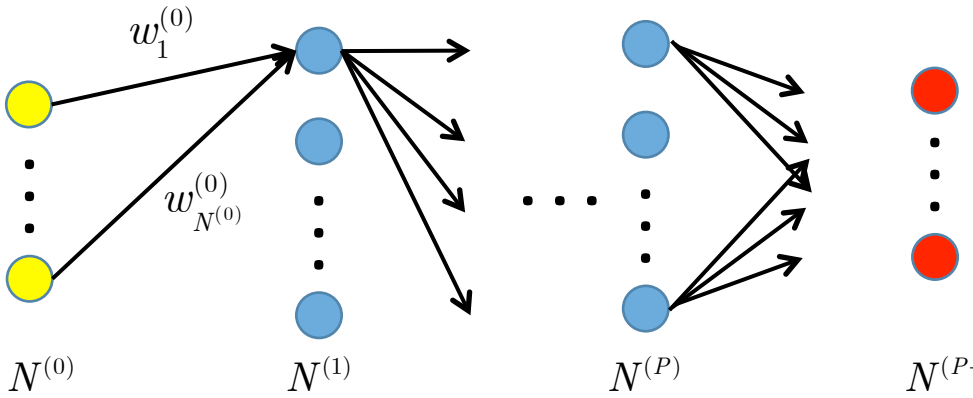
$$f(x_1, x_2) = \sigma(w_1 x_1 + w_2 x_2 + b)$$





Régression non linéaire

Réseau profond



$$\mathbf{x} = \{x_i^{(0)}\}_{1,..,N^{(0)}}$$

Entrée
(Donnée)

$$\mathbf{f}(\boldsymbol{\theta}, \mathbf{x}) = \mathbf{x}^{(P+1)}$$

$$\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$$

Sortie
(Fonction)

$$\mathbf{x}^{(1)} = \sigma(\mathbf{W}^{(1)}.\mathbf{x}^{(0)} + \mathbf{b}^{(1)})$$

$$\mathbf{x}^{(P+1)} = \sigma(\mathbf{W}^{(P+1)}.\mathbf{x}^{(P)} + \mathbf{b}^{(P+1)})$$

$$\{\mathbf{X}_i, \mathbf{Y}_i\}_{i=1,..,n}$$

Jeu de données

Minimisation non linéaire

$$\min_{\boldsymbol{\theta}=(\mathbf{W},\mathbf{b})} L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left\| \mathbf{Y}_i - \mathbf{f}(\boldsymbol{\theta}, \mathbf{X}_i) \right\|^2$$

Equations de récurrence

$$\mathbf{x}^{(c)} = \sigma(\mathbf{W}^{(c)} \cdot \mathbf{x}^{(c-1)} + \mathbf{b}^{(c)})$$

$$\frac{\partial \mathbf{x}^{(c)}}{\partial \mathbf{x}} = \sigma' \cdot \mathbf{W}^{(c)} \cdot \frac{\partial \mathbf{x}^{(c-1)}}{\partial \mathbf{x}}$$

La solution est un champ différentiable

Méthodologie de mise en oeuvre

1. Elaboration de la *Data*

Les données

- Observations et mesures
- Eventuellement hétérogènes
- Résultats de modélisations *a priori*

Suffisamment riches pour

- être représentatives (sans biais *a priori*)
- être scindées en
  - training data (sans biais)
  - test data

Data={training, test}

|Data|=M

2. Choix de la fonction *f*

Type et structure du réseau NN

- Réseaux unidirectionnels (FFNN)
- Réseaux récurrents (RNN, LSTM, GRUs)
- Réseaux convolutionnels (CNN)

Dimension de la fonction

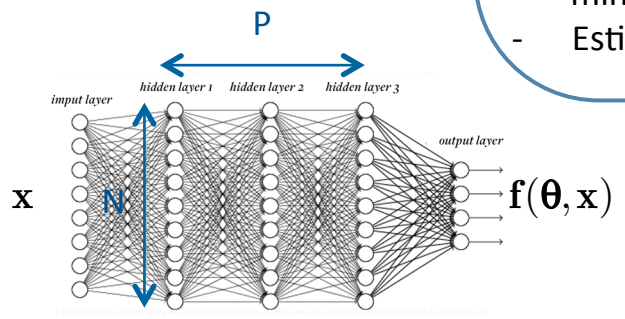
- Largeur du réseau N
- Profondeur du réseau P

$|\vartheta| \approx (\text{lin}(N), \exp(P))$

- Type de fonction d'activation σ
- Type de fonction erreur *L* (coût, loss)

|Paramètres|=$|\vartheta|$

Attention au rapport $M/|\vartheta|$
(transition de phase, overfitting, ...)



3. Construction de la fonction *f*

Type de logiciel

- Numpy, SciKit learn (Inria), TensorFlow (Google), …

Type de matériel

- HPC, GPUs
- Massivement distribué, parallèle

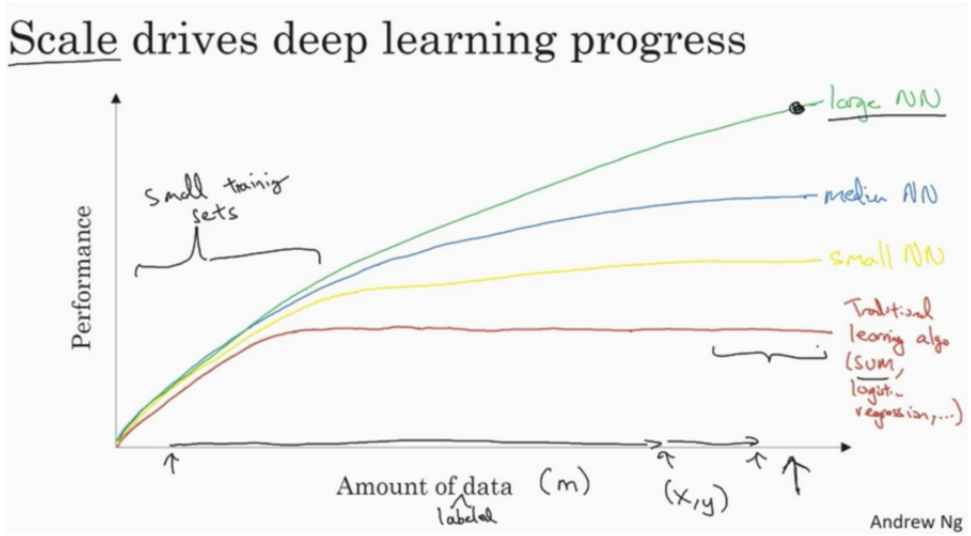
Calcul des paramètres
(apprentissage, plusieurs jours de calcul)

- Choix de la *training data* (non biaisée)
- Calcul, choix du solveur (ADAM, ...) et des techniques de régularisation (Dropout, Pénalisation L1 ou L2, Normalisation par mini-batch, ...)
- Estimation de l'erreur sur la *test data*

4. Calcul de *f* — Temps réel

$\mathbf{f}(\boldsymbol{\theta}, \mathbf{x})$, $\boldsymbol{\theta}$ connu

Intérêt du Deep Learning



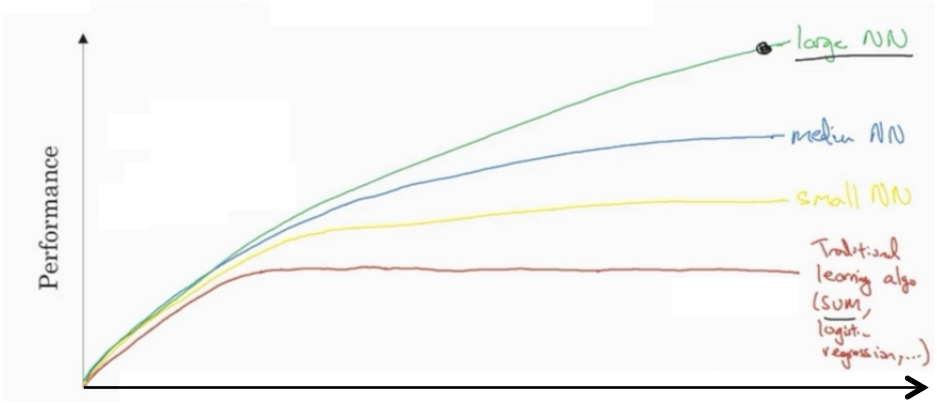
10^2 10^6 $>10^{10}$ (2017)

Progrès des 10 dernières années

- Existences de grandes bases de données
- Progrès dans les algorithmes
- Calculs GPU
- Résultats théoriques

Intérêt

- Efficacité sur les très grands nombres de données
- Efficacité sur les fortes non linéarités (même sur un petit nb de données)



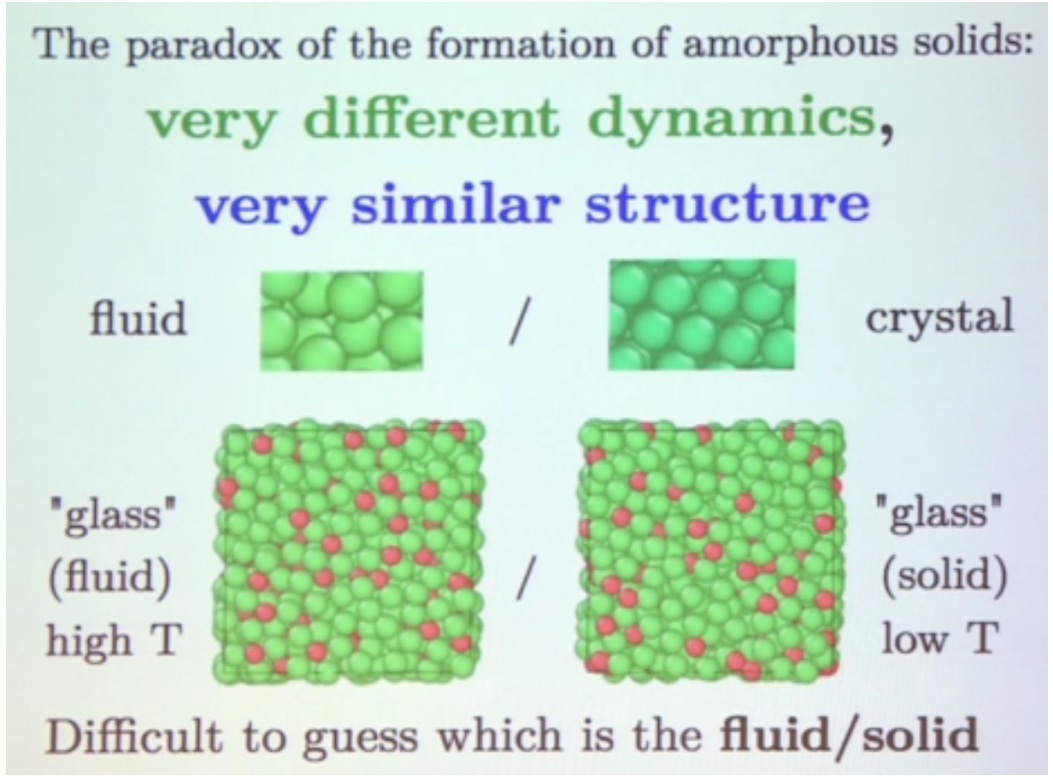
fortes non linéarités

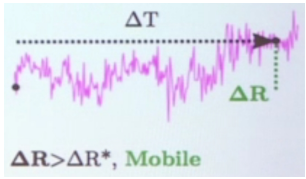
(à nombre de neurones égal, un réseau profond est plus performant)

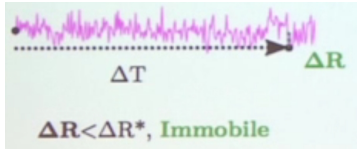
Exemple : Solides amorphes

Formation des spins amorphes : les modélisations DEM n'apprennent rien sur la physique : l'information est dans les résultats, il faut l'extraire !!!

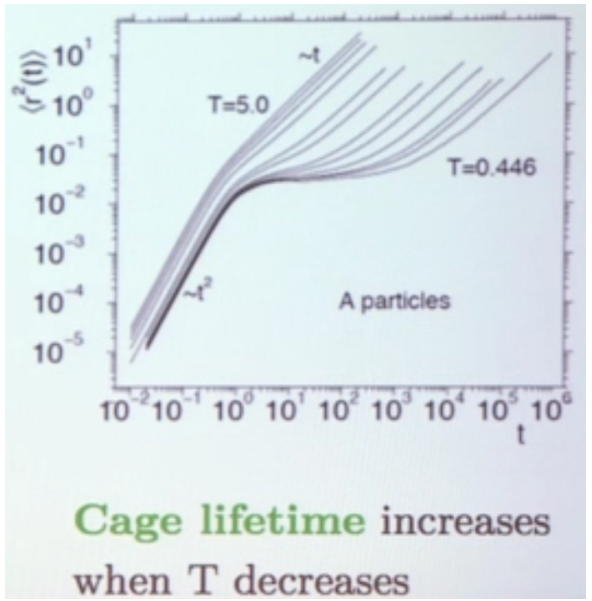
1000 particules 3D, (position et vitesse)
training data : 1877 snapshots (1.5 10^6 particules)
test data : 510 snapshots (0.4 10^6 particules)
Travail en cours de publication à l'ENS Paris ...

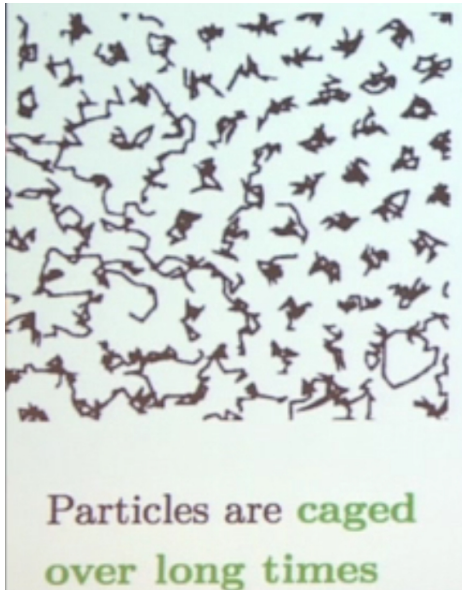






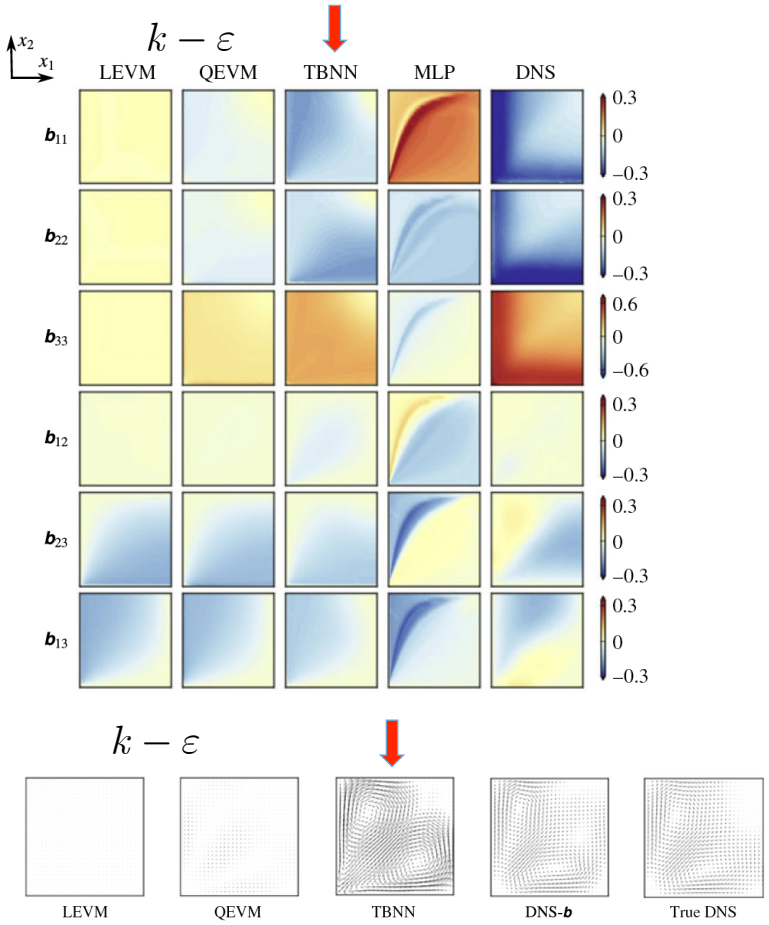
Glasses: Microscopic Dynamics



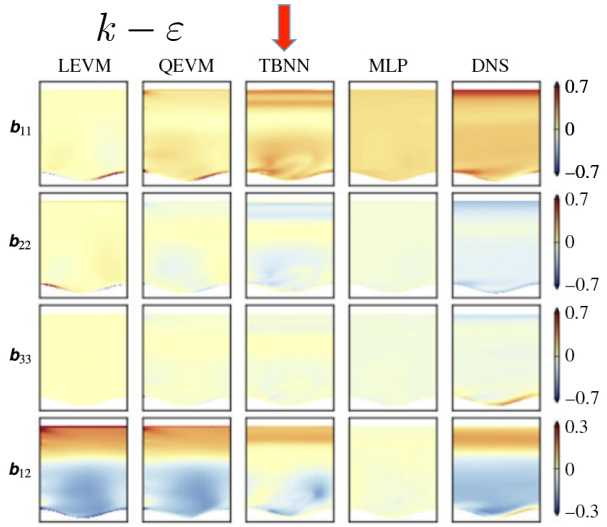


Cubuk E.D. et al., Identifying Structural Flow Defects in Disordered Solids Using Machine-Learning Methods, *Phys. Rev. L*, **114**, 108001 (2015)

Exemple : Modélisation de la turbulence



Duct flow (Re=2000)

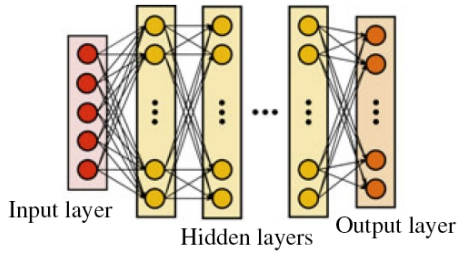


Flow over a wavy wall (Re=6850)

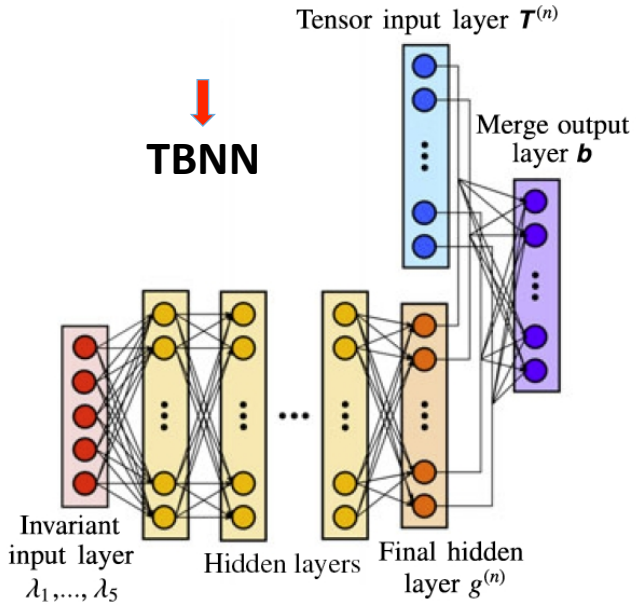
Importance du choix du réseau !!

$$L = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left\|\underline{\underline{b}}_{DNS}(x_i,t_i) - \underline{\underline{b}}_{NN}(x_i,t_i)\right\|^2}_{\text{Ecart aux résultats DNS (tenseur de Reynolds)}}$$

8 couches profondes, 30 neurones par couche, Relu
Training data : 9 resultats de simulations RANS (pipe, jet, channel, ...)
CPU : 10 jours/network, 20 networks (200 j de calcul)
Test data: résultats de simulations DNS

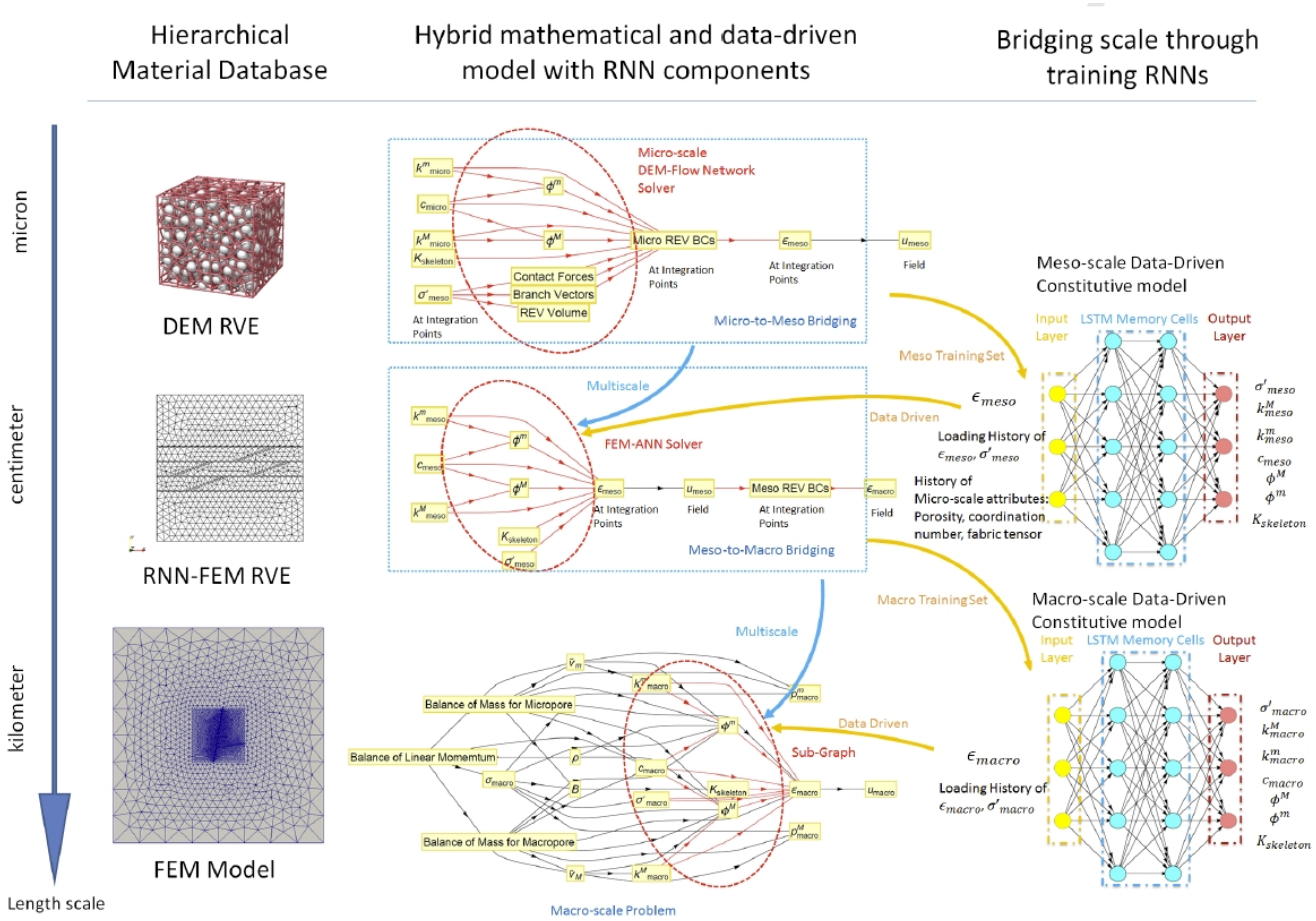


MLP



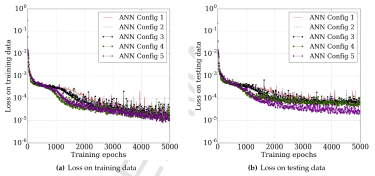
Ling J., Kurzawski A., Templeton J., Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, *J. Fluid Mech.* (2016), vol. 807, pp. 155-166
Kutz N. J., Deep learning in fluid dynamics, *J. Fluid Mech.* (2017), vol. 814, pp. 14

Exemple : Modélisation multi-échelle poroplastique



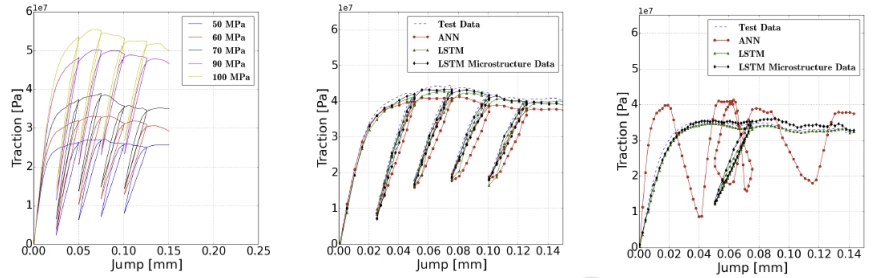
$$L = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left\| X_{DEM}(x_i,t_i) - X_{NN}(x_i,t_i)\right\|^2}_{\text{Ecart aux résultats DEM}}$$

(tenseur de texture et vecteur force)



ANN : réseau classique
LSTM : Réseau de type récurrent pour tenir compte des irréversibilités (mémoire)
2 couches profondes, 80 neur./couche, Sigmoid & ReLU
Training data (y/c Dropout) : FEM/DEM, 500 échantillons
Test data: FEM/DEM, 100 échantillons

Importance du choix du réseau !!



(a) Training data for different confining pressure σ **(b)** Forward prediction of ANN, LSTM and LSTM with micro-scale data **(e)** Forward prediction of unloading

Wang K., Sun W., A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning, *Comput. Methods Appl. Mech. Engrg.*, Vol. 334 (2018) pp. 337-380

Bock et al., A Review of the Application of Machine Learning and Data Mining Approaches in Continuum Materials Mechanics, *Frontiers in Materials*, 2019, doi: 10.3389/fmats.2019.00110

Application : Interpolation paramétrée de données

Supposons un processus physique pour lequel nous avons

Plusieurs ensembles homogènes paramétrés de mesures

$$\left\{U(p_i,t_i,x_i)\right\}_{i=1,..,N}$$

Par exemple un ensemble de résultats d'essais de laboratoire,
chaque ensemble correspond à un essai pour une valeur du paramètre *p* fixé (par exemple la température)

1ères applications en Mécanique :
- modélisation de la turbulence (2016)
- modélisation « temps réel » des tsunamis (2017)

Alors on peux calculer un champ *u(x,t)* avec un réseau de neurones en minimisant la fonction d'erreur

$$L = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left\|U(p_i,x_j,t_k) - u(p_i,x_j,t_k)\right\|^2}_{\text{Ecart aux mesures}}$$

Intérêt :
- le résultat est un champ *u(x,t)*
- *u(x,t)* peut être différencié ($\partial u/\partial x$ ou $\partial u/\partial t$)
- interpolation d'un très grand nombre de données possible (>10^6)
- analyse unifiée d'une collection d'ensemble de données

The Universal Approximation Theorems - Cybenko (1989), Hornik (1991) (largeur ➔∞) , Lin et al. (1993) (il faut que σ ne soit pas un polynôme)
Lu *et al.* (2017) (profondeur➔∞)

Application : Analyse hybride de séries non stationnaires

Supposons un processus physique pour lequel nous avons

Un modèle statistique *a priori*

$$\underbrace{u_{HS}(x_i,t)}_{\text{(ex: déplacement d'un barrage voûte)}} =$$

$$\underbrace{H(x_i,t)}_{\substack{\text{évolution non saisonnière sans dérive en temps} \\ \text{(ex : influence du plan d'eau pour un barrage)}}}$$

$$+ \quad \underbrace{S(x_i,t)}_{\substack{\text{évolution saisonnière} \\ \text{(ex: température annuelle)}}}$$

Des mesures $\left\{U(t_k,x_k)\right\}_{k=1,..,N}$

Alors on peux calculer une solution *u(x,t)* avec un réseau de neurones en minimisant la fonction d'erreur

$$L = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left\| U(x_i,t_i) - \left(H(x_i,t_i) + S(x_i,t_i) + \underbrace{u(x_i,t_i)}_{\text{Dérive en temps}} \right) \right\|^2}_{\text{Ecart aux mesures}}$$

Intérêt :

- analyse groupée de plusieurs séries homogènes (plusieurs instruments d'auscultation)
- le résultat est un champ (pas des valeurs ponctuelles)
- pas de formulation *a priori* de la dérive en temps

Xu W. et al., A hybrid modelling method for time series forecasting based on a linear regression model and deep learning, *Applied Intelligence* (in press, 2019)
Cai M. et al., Day-ahead building-level load forecasts using deep learning vs. traditional time-series techniques, *Applied Energy,* 236:1078–1088, 2019

Application : Interpolation NN+*a priori* knowldege

Supposons un processus physique pour lequel nous avons

Des équations d'évolution

Equation de champ
(conservations de la masse, qté de mouvement, ...)

$$D(u) = f \text{ sur } [0,\mathrm{T}] \times \Omega$$

Conditions aux limites

$$B(u) = g \text{ sur } [0,\mathrm{T}] \times \partial\Omega$$

Conditions initiales

$$u = u^0 \text{ à } t = 0 \text{ dans } \Omega$$

Des données $\{f, g, u^0\}$

Des mesures $\{U(t_i, x_i)\}_{i=1,..,N}$

Alors on peux calculer une solution *u(x,t)* avec un réseau de neurones en minimisant la fonction d'erreur

$$L = \underbrace{\lambda_1 \tfrac{1}{T}\tfrac{1}{\Omega} \int_0^T \int_\Omega \left\| D(u) - f \right\|^2 d\Omega\, dt}_{\text{Equation de champ}} + \underbrace{\lambda_2 \tfrac{1}{T}\tfrac{1}{\partial\Omega} \int_0^T \int_{\partial\Omega} \left\| B(u) - g \right\|^2 d\Gamma dt}_{\text{Conditions aux limites}}$$

$$+ \underbrace{\tfrac{1}{\Omega} \int_\Omega \left\| u^0 - u \right\|^2 d\Omega}_{\text{Conditions initiales}} + \underbrace{\tfrac{1}{n} \sum_{i=1}^{n} \left\| U(x_i, t_i) - u(x_i, t_i) \right\|^2}_{\text{Ecart aux mesures}}$$

Intérêt :

- pas de maillage, pas de condition de type CFL
- le résultat est un champ *u(x,t)* (pas des valeurs ponctuelles)
- problèmes de très grande taille possible (pas de matrices)
- calcul HPC sur GPU possible avec un solveur
- prise en compte de connaissance a priori (par exemple champ de vitesse à divergence nulle div(u)=0)

Raissi M., Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations, *Journal of Machine Learning Research* 19 (2018) 1-24

Application : Résolution d'un problème aux limites

Supposons un processus physique pour lequel nous avons

Des équations d'évolution

Equation de champ
(conservations de la masse, qté de mouvement, ...)

$$D(u) = f \text{ sur } [0,\mathrm{T}] \times \Omega$$

Conditions aux limites

$$B(u) = g \text{ sur } [0,\mathrm{T}] \times \partial\Omega$$

Conditions initiales

$$u = u^0 \text{ à } t = 0 \text{ dans } \Omega$$

Des données $\{f, g, u^0\}$

Alors on peux calculer une solution *u(x,t)* avec un réseau de neurones en minimisant la fonction d'erreur

$$L = \lambda_1 \underbrace{\frac{1}{T}\frac{1}{\Omega}\int_0^T \int_\Omega \left\| D(u) - f \right\|^2 d\Omega\, dt}_{\text{Equation de champ}} + \lambda_2 \underbrace{\frac{1}{T}\frac{1}{\partial\Omega}\int_0^T \int_{\partial\Omega} \left\| B(u) - g \right\|^2 d\Gamma dt}_{\text{Conditions aux limites}}$$

$$+ \underbrace{\frac{1}{\Omega}\int_\Omega \left\| u^0 - u \right\|^2 d\Omega}_{\text{Conditions initiales}}$$

Intérêt :

- pas de maillage, pas de condition de type CFL
- le résultat est un champ *u (x,t)* (pas des valeurs ponctuelles)
- interpolations possibles sur les données d'entrée
- problèmes de très grande taille possible (pas de matrices)
- calcul HPC sur GPU possible avec un solveur

Yadav N., Yadav A., Kumar M., An Introduction to Neural Network Methods for Differential Equations, Springer, 2015

Application : Analyse de sensibilité et Problème inverse

Supposons un processus physique avec

Des équations d'évolution fonction de données matérielles inconnues *p*

Equation de champ
(conservations de la masse, qté de mouvement, ...)

$$D(p)(u) = f \text{ sur } [0,\mathrm{T}] \times \Omega$$

Conditions aux limites

$$B(u) = g \text{ sur } [0,\mathrm{T}] \times \partial\Omega$$

Conditions initiales

$$u = u^0 \text{ à } t = 0 \text{ dans } \Omega$$

Des données $\{f, g, u^0\}$

Des mesures $\{U(t_i, x_i)\}_{i=1,..,N}$

Alors on peux calculer une solution *u(p,x,t)* avec un réseau de neurones en minimisant la fonction d'erreur

$$L = \lambda_1 \underbrace{\frac{1}{T}\frac{1}{\Omega}\int_0^T \int_\Omega \left\| D(p)(u) - f \right\|^2 d\Omega\, dt}_{\text{Equation de champ}} + \lambda_2 \underbrace{\frac{1}{T}\frac{1}{\partial\Omega}\int_0^T \int_{\partial\Omega} \left\| B(u) - g \right\|^2 d\Gamma dt}_{\text{Conditions aux limites}} + \underbrace{\frac{1}{\Omega}\int_\Omega \left\| u^0 - u \right\|^2 d\Omega}_{\text{Conditions initiales}}$$

puis identifier *p* par minimisation non linéaire classique

$$\min_p \underbrace{\frac{1}{n}\sum_{i=1}^{n} \left\| U(x_i, t_i) - u(p, x_i, t_i) \right\|^2}_{\text{Ecart aux mesures}}$$

Intérêt :

- pas de maillage
- **le résultat est un champ *u(p,x,t)* paramétré par l'inconnue *p* (et non pas des valeurs ponctuelles issues de plusieurs calculs pour plusieurs *p*)**
- calcul de sensibilité au données d'entrée immédiat
- *le problème est bien posé*

Jo H. et al., Deep Neural Network Approach to Forward-Inverse Problems, Preprint (in press, 2019)