Aprendizaje profundo basado en la física

Semana 3: Embebiendo física dentro de redes neuronales

Docente: José I. Robledo - 21/04/2026

Ecuaciones diferenciales

Algunos tipos que veremos

ODE: $u''(x) + u(x) = 0$, con $x \in \mathbb{R}$

PDE indep. del tiempo: $\nabla^2_{\vec{x}} u(\vec{x}) + u(\vec{x}) = 0$ con $\vec{x} \in \mathbb{R}^n$

PDE dep. del tiempo: $\nabla^2_{\vec{x},t} u(\vec{x},t) + u(\vec{x},t) + \dfrac{\partial u(\vec{x},t)}{\partial t} = 0$ con $\vec{x} \in \mathbb{R}^n$

IDE: $\dfrac{\partial u(x)}{\partial x} + \displaystyle\int_{x_0}^{x} f(y, u(y))dy = g(x, u(x))$

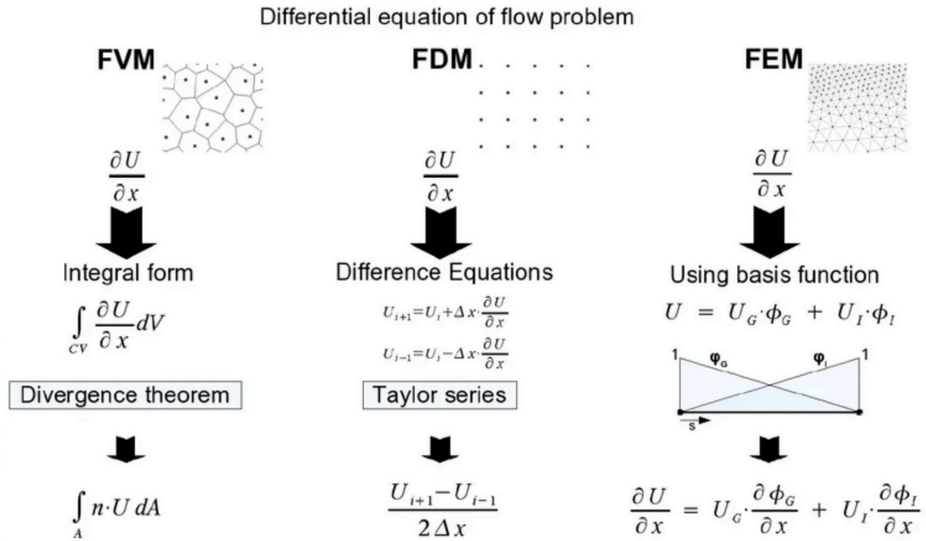
Ecuaciones diferenciales

Solvers

- **Métodos analíticos**
  - Buscan expresión cerrada para $u(x)$
  - Muy restringidos en la práctica
- **Métodos numéricos**
  - Para ODEs: Euler, Runge-Kutta, Crank-Nicolson, etc.
  - Para PDEs: FDM, FEM, FVM, Métodos espectrales, etc.

Ecuaciones diferenciales

Solvers numéricos utilizados en grillas



Método de diferencias finitas (FDM)

$\frac{du}{dx} \approx \frac{u_{i+1} - u_i}{\Delta x}$, genera sistema de ecuaciones algebraicas.

Método de volúmenes finitos (FVM): aplica leyes de conservación en pequeños volúmenes del dominio

$$\int_V \nabla \cdot F dV = \int_{\partial V} F \cdot n dS$$

Método de elementos finitos (FEM): aproxima soluciones a PDEs e IDEs utilizando un conjunto finito de funciones, tratando de llevar el problema de PDE a ODE y utilizar el método de Euler o de Runge-Kutta para encontrar la solución del sistema simplificado.

$$u(x) \approx \sum_i u_i \phi_i(x)$$

Embebiendo física dentro de redes neuronales

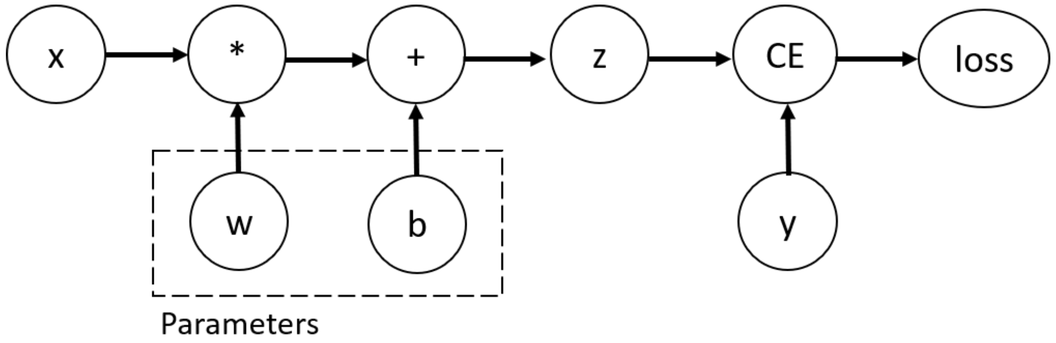
Introducción a las redes neuronales informadas por la física

Red neuronal $f_w(x)$

Autograd → deriva redes neuronales con respecto **a lo que querramos**

$$f_w(x) \rightarrow f(x, w) \implies \frac{\partial f}{\partial w}, \frac{\partial f}{\partial x}$$

PyTorch
Autograd



Embebiendo física dentro de redes neuronales

PINNs

Sea $u(\vec{x}, t)$ la solución de una ecuación diferencial en derivadas parciales (PDE).

Si definimos $u_x := \frac{\partial u}{\partial x}$, podemos expresar la PDE en término de una función $\mathcal{F}$ de sus derivadas parciales parametrizada por λ:

$$u_t = \mathcal{F}_\lambda(u_x, u_{xx}, \cdots, u_{xx\cdots x})$$

La red neuronal $f_w(\vec{x}, t)$ aproximará la solución real del problema $u(\vec{x}, t)$ a partir de los datos observados y del cumplimiento de la PDE, mediante el cálculo del **residuo** R y su minimización:

$$R = u_t - \mathcal{F}_\lambda(u_x, u_{xx}, \cdots, u_{xx\cdots x}) = 0$$

Embebiendo física dentro de redes neuronales

PINNs

La red neuronal $f_w(\vec{x}, t)$ aproximará la solución real del problema $u(\vec{x}, t)$ a partir de los datos observados y del cumplimiento de la PDE, mediante el cálculo del **residuo** R_{fisico} y su minimización.

Queremos que $R_{fisico}(f_w) = 0$, pero tenemos

$$R_{fisico} = f_t - \mathcal{F}_\lambda(f_x, f_{xx}, \cdots, f_{xx\cdots x}) \geq 0$$

Planteamos el problema

$$\arg\min_\theta \sum_i \alpha_0(f_w(\vec{x}_i, t_i) - y_i)^2 + \alpha_1 l(R_{fisico}(x_i))$$

Soft constraint

Embebiendo física dentro de redes neuronales

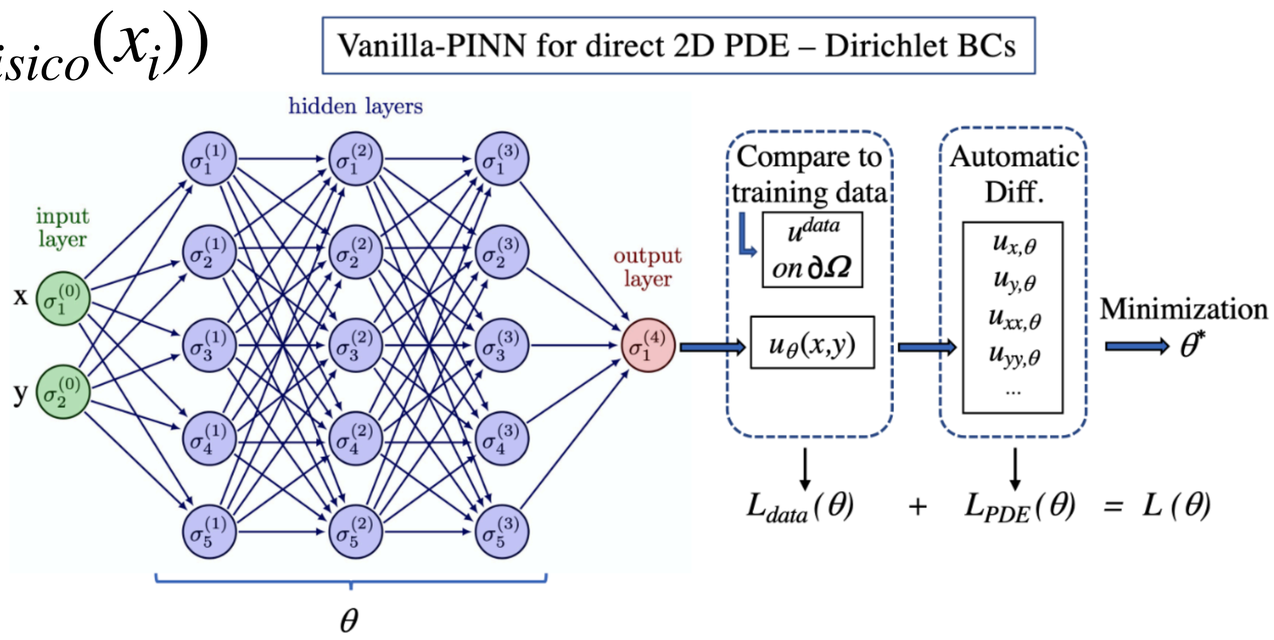
PINNs

En PINNs, proponemos que una red neuronal parametrizada por w reproduzca la solución a la ecuación diferencial planteada

$$u(\vec{x}_i, t) \approx f_w(\vec{x}_i, t) = NN_w(\vec{x}_i, t)$$

$$\arg\min_\theta \sum_i \alpha_0 (f_w(\vec{x}_i, t_i) - y_i)^2 + \alpha_1 l(R_{fisico}(x_i))$$

Soft constraint



Embebiendo física dentro de redes neuronales

Condiciones de contorno y condiciones iniciales

Cuando planteamos la solución de una ecuación diferencial, debemos ademas especificar las condiciones de contorno y/o condiciones iniciales, ya que estas determinan de forma única la solución de la PDE. Por ende, ante un problema físico real, imponemos que se cumplan las condiciones de contorno e iniciales de la misma manera, mediante alguna función l del residuo correspondiente

$$R_{contorno} = \sum_{i=1}^{n} f_w(\vec{x}_B^{(i)}, t^{(i)}) - u_B^{(i)} \qquad R_{CI} = \sum_{i=1}^{n_{CI}} f_w(\vec{x}_0^{(i)}, t_0^{(i)}) - u_0^{(i)}$$

Condiciones de contorno

Tipos

- Si tenemos una ecuación diferencial para una función $u(x)$ en un dominio $x \in [a, b]$, una condición de **Dirichlet** especifica el valor de la solución en el contorno

$$u(x_i) = c_i \text{ con } x_i \text{ en el contorno}$$

- Una condición de **Neumann** fija el valor en la derivada

$$\frac{\partial u(x_i)}{\partial x} = c_i \text{ con } x_i \text{ en el contorno}$$

- La condición de **Robin** combina ambas

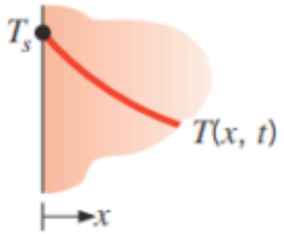
$$au(x_i) + b\frac{\partial u(x_i)}{\partial x} = c_i \text{ con } x_i \text{ en el contorno}$$

Condiciones de contorno

Tipos

1. Constant surface temperature

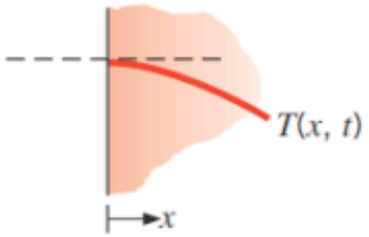
$$T(0, t) = T_s$$



Dirichlet condition

2. Adiabatic or insulated surface

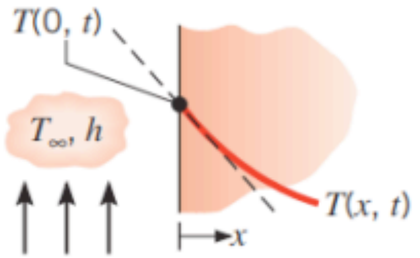
$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = 0$$



Neumann condition

3. Convection surface condition

$$-k\left.\frac{\partial T}{\partial x}\right|_{x=0} = h[T_\infty - T(0, t)]$$



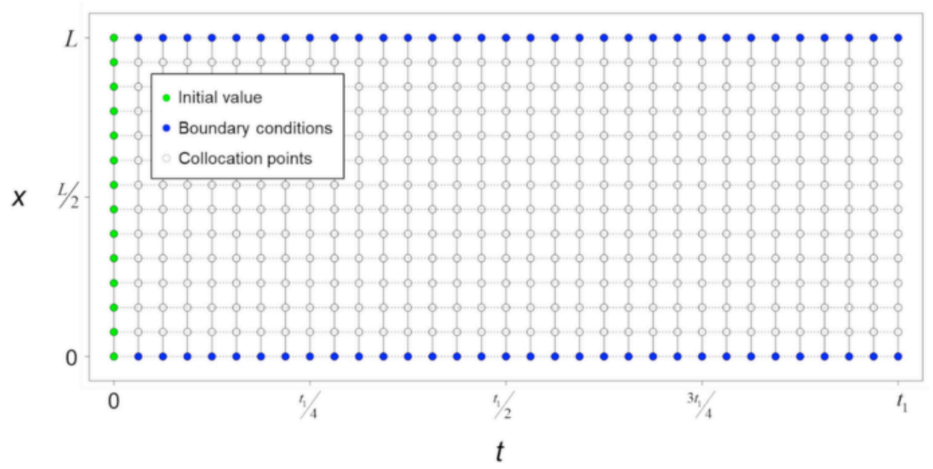
Robin condition

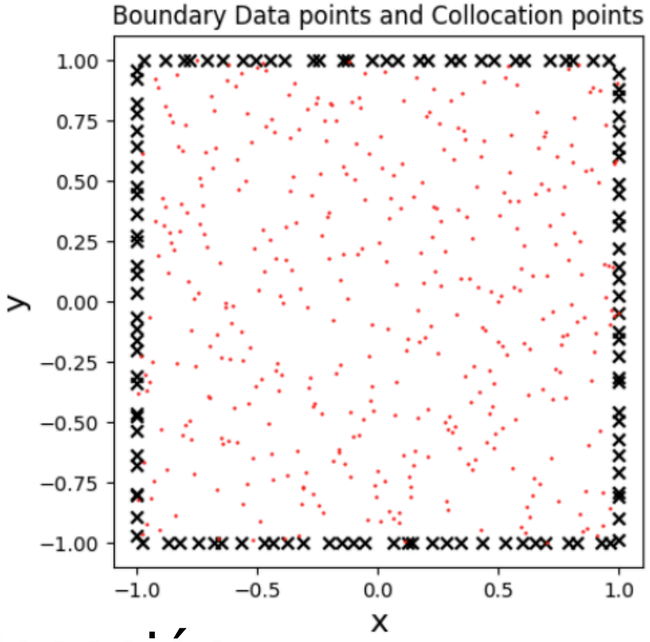
T. L. Bergman, *Fundamentals of heat and mass transfer*. John Wiley & Sons, 2011.

Collocation points

Exigir puntos donde se cumple la ecuación diferencial

Podemos armar datos ficticios!





N_c puntos de anclaje o colocación.

Pueden estar **fuera** del rango de los datos de entrenamiento

https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/

Embebiendo física dentro de redes neuronales

La función de costo de las PINNs

$$\arg\min_\theta \sum_i \alpha_0 (f_w(\vec{x}_i, t_i) - y_i)^2 + \alpha_1 l(R_{fisico}(x_i)) + \alpha_2 l(R_{contorno}) + \alpha_3 l(R_{CI})$$

Es común utilizar $l(x) = MSE(x)$

$$\mathscr{L}_{data} = \sum_i^N (f_w(\vec{x}_i, t) - y_i)^2$$

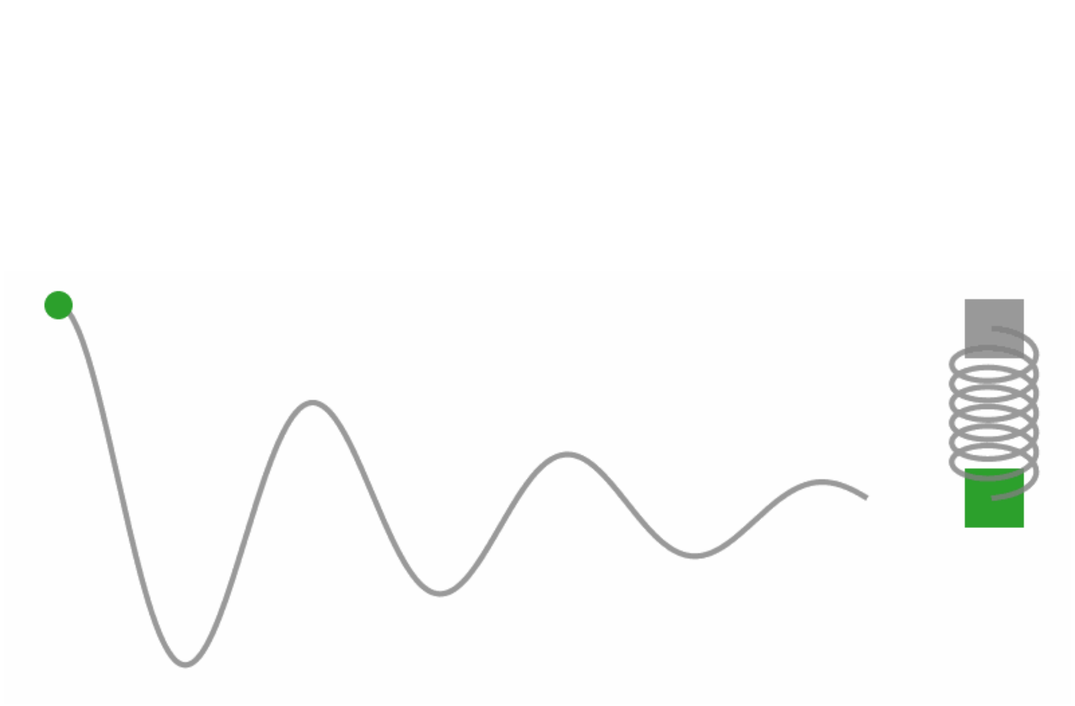
$$\mathscr{L}_{fisica} = \sum_i^{N_c} (R_{fisico}(x_i))^2$$

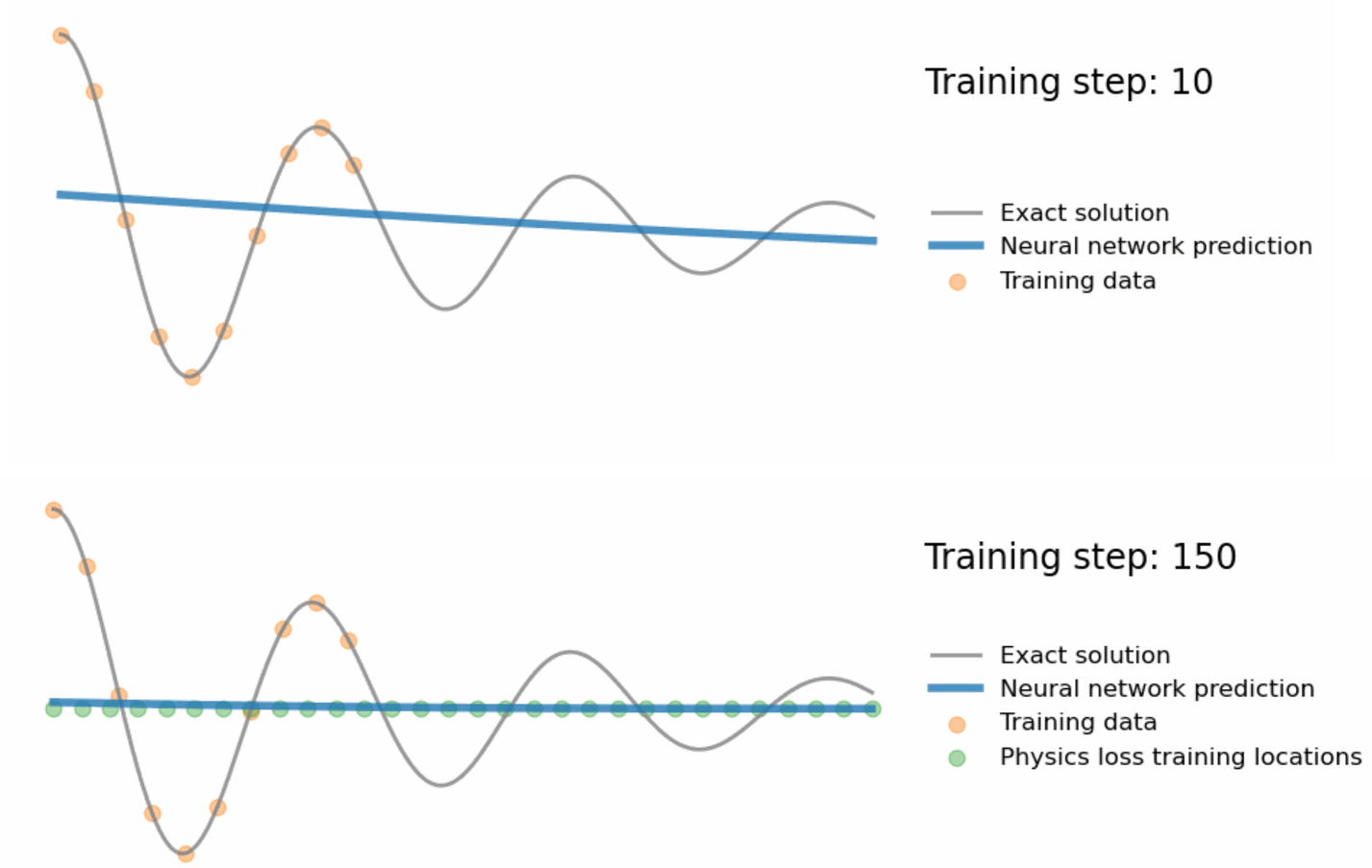
$$\mathscr{L}_{contorno} = \sum_{i=1}^{n} (f_w(\vec{x}_B^{(i)}, t^{(i)}) - u_B^{(i)})^2$$

$$\mathscr{L}_{CI} = \sum_{i=1}^{n_{CI}} (f_w(\vec{x}_0^{(i)}, t_0^{(i)}) - u_0^{(i)})^2$$

$$\arg\min_w (\alpha_0 \mathscr{L}_{data} + \alpha_1 \mathscr{L}_{fisica} + \alpha_2 \mathscr{L}_{contorno} + \alpha_3 \mathscr{L}_{CI})$$

Ejemplo Oscilador armónico amortiguado





https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/

Derivadas en batches

Tomamos un conjunto de puntos del dominio y vemos que se debe cumplir la ecuación diferencial

$$x_f \in \mathbb{R}^{N \times d}$$

Donde N es el tamaño del batch y d es la cantidad de variables

$$u_w(x_f) = f_w(x_f) \implies f_w(x_f) = [f_w(x_1), f_w(x_2), \ldots, f_w(x_N)] \in \mathbb{R}^N$$

Necesitamos un Jacobiano $N \times d$ ...

Producto Vector-Jacobiano

Vector Jacobian Product (VJP)

Supongamos que tenemos $\vec{y} = f(\vec{x})$ con $x \in \mathbb{R}^n$ e $y \in \mathbb{R}^m$. Podemos escribir el **Jacobiano** como

$$J = \frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_d} \end{bmatrix}$$

Autograd calcula el producto vector-Jacobiano (JVP) para ahorrar cómputo ya que generalmente sólo necesitamos ciertas combinaciones de derivadas, donde JVP se define como

$$v^T J \in \mathbb{R}^d, \text{ con } v \text{ elegido de manera estratégica.}$$

Producto Vector-Jacobiano

La clave está en la **independencia entre puntos**

$u(x_i)$ depende solo de x_i

Entonces el Jacobiano tiene estructura diagonal por bloques.

$$\frac{\partial u(x_i)}{\partial x_j} = \delta_k(i = j)\frac{\partial u(x_i)}{\partial x_i}$$

Vector-Jacobian Product

Ejemplo: batch con 2 elementos

$$u = \begin{bmatrix} f_w(x_1) \\ f_w(x_2) \end{bmatrix}, J = \begin{bmatrix} \frac{\partial f_w(x_1)}{\partial x_1} & 0 \\ 0 & \frac{\partial f_w(x_2)}{\partial x_2} \end{bmatrix}$$

$$\text{si } v = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow v^T J = (\frac{\partial f_w(x_1)}{\partial x_1}, \frac{\partial f_w(x_2)}{\partial x_2})$$

https://docs.pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

Teórico práctico semana 3

Aproximación fina

L-BFGS

- Los métodos de optimización que vimos actualizan los pesos en función del gradiente de la función de costo y han demostrado excelente capacidad para encontrar el mínimo global.

- Una vez que estamos en la vecindad del mínimo global, existen métodos más rápidos y precisos para encontrar el mínimo global. Estos métodos son sensibles a la inicialización, por ende sirven mayoritariamente cuando ya nos encontramos cerca del mínimo global.

- Optimización FINA

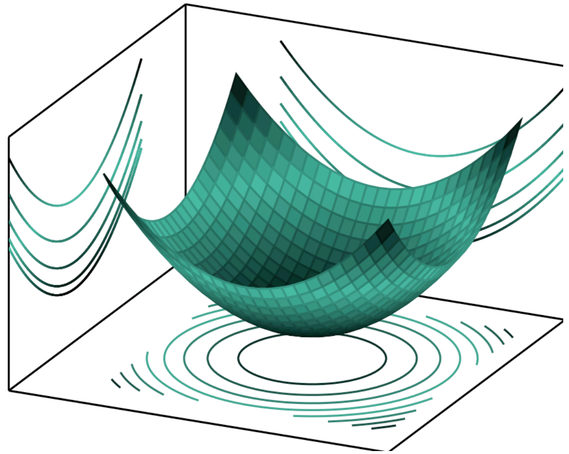
Aproximación fina
L-BFGS

Aproximación cuadrática local

Cerca del punto actual $w^{(t)}$, aproximamos por serie de Taylor

$$\mathscr{L}_w \approx \mathscr{L}_{w^{(t)}} + \nabla \mathscr{L}^T_{w^{(t)}}(w - w^{(t)}) + \frac{1}{2}(w - w^{(t)})^T H(w - w^{(t)}) + \mathcal{O}(w^3),$$

Donde $H = \nabla^2 \mathscr{L}_{w^{(t)}}$ es la **matriz Hessiana**.



Localmente, parece un un pozo cuadrático!

Aproximación fina

L-BFGS

$$\mathscr{L}_w \approx \mathscr{L}_{w^{(t)}} + \nabla \mathscr{L}^T_{w^{(t)}}(w - w^{(t)}) + \frac{1}{2}(w - w^{(t)})^T H (w - w^{(t)}) + \mathcal{O}(w^3),$$

$$Q(d) = \mathscr{L}_{w^{(t)}} + g^T d + \frac{1}{2} d^T H d, \quad \text{con } g = \nabla \mathscr{L}_{w^{(t)}} \quad \text{y} \quad d = (w - w^{(t)}).$$

Encontrar el mejor paso que nos lleva a el mínimo de un sólo paso si fuese cuadrático:

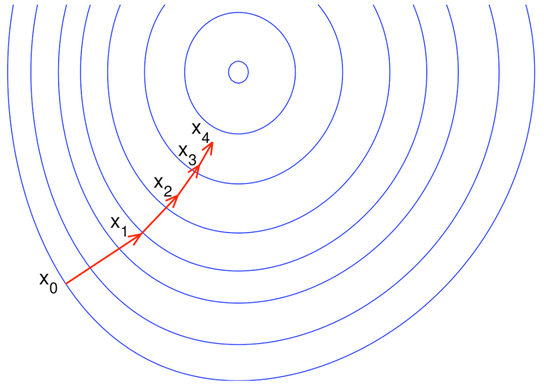
$$\nabla_d Q(d) = g + Hd = 0 \implies d = -H^{-1} g$$

Método de Newton

$$w^{(t+1)} = w^{(t)} - H^{-1} \nabla \mathscr{L}_{w^{(t)}}$$

Calcular H^{-1} cuesta mucho

Aproximación fina
L-BFGS



Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) construye una aproximación de la inversa del Hessiano

$$H^{(t)^{-1}} \approx \nabla^2 \mathscr{L}_{W^{(t)}}$$

Usando sólo gradientes del paso anterior y la actualización de los pesos del paso anterior. Almacena sólo los últimos m cambios (**L** del nombre)

$s^{(t)} = w^{(t+1)} - w^{(t)}$ (cambio en el parámetro)

$y^{(t)} = \nabla \mathscr{L}_{W^{(t+1)}} - \nabla \mathscr{L}_{W^{(t)}}$ (cambio en el gradiente)

Información sobre curvatura codificada en $\rho^{(t)} = \dfrac{1}{y^{(t)^T} s^{(t)}}$

Doble loop recursivo para aproximar Hessiano con esta info. Aplicando una secuencia de actualizaciones de rango 1 que aproximan la inversa del hessiano.

L-BFGS en PINNs

Encontrar cercanía al mínimo… y luego el mínimo

```
model.compile("adam", lr=1e-3)
model.train(iterations=3000)
model.compile("L-BFGS")
model.train()
```

Muy usado en PINNs!

https://aria42.com/blog/2014/12/understanding-lbfgs

Pesos auto-adapatativos en PINNs

Problemas con discontinuidades en CI?

$$\arg\min_w(\alpha_0\mathscr{L}_{data} + \alpha_1\mathscr{L}_{fisica} + \alpha_2\mathscr{L}_{contorno} + \alpha_3\mathscr{L}_{CI})$$

$$\arg\min_w \arg\max_\lambda(\mathscr{L}_{data} + \alpha(\lambda_{fisica})\mathscr{L}_{fisica} + \alpha(\lambda_{contorno})\mathscr{L}_{contorno} + \alpha(\lambda_{CI})\mathscr{L}_{CI})$$

$$\mathcal{L}_r(\boldsymbol{w}, \boldsymbol{\lambda}_r) = \frac{1}{2}\sum_{i=1}^{N_r} m(\lambda_r^i)\,|\mathcal{N}_{\boldsymbol{x},t}[u(\boldsymbol{x}_r^i, t_r^i; \boldsymbol{w})] - f(\boldsymbol{x}_r^i, t_r^i)|^2$$

$$\mathcal{L}_b(\boldsymbol{w}, \boldsymbol{\lambda}_b) = \frac{1}{2}\sum_{i=1}^{N_b} m(\lambda_b^i)\,|\mathcal{B}_{\boldsymbol{x},t}[u(\boldsymbol{x}_r^i, t_r^i; \boldsymbol{w})] - g(\boldsymbol{x}_b^i, t_b^i)|^2$$

$$\mathcal{L}_0(\boldsymbol{w}, \boldsymbol{\lambda}_0) = \frac{1}{2}\sum_{i=1}^{N_0} m(\lambda_0^i)\,|u(\boldsymbol{x}_0^i, 0; \boldsymbol{w}) - h(\boldsymbol{x}_0^i)|^2.$$

Descenso por el gradiente para w

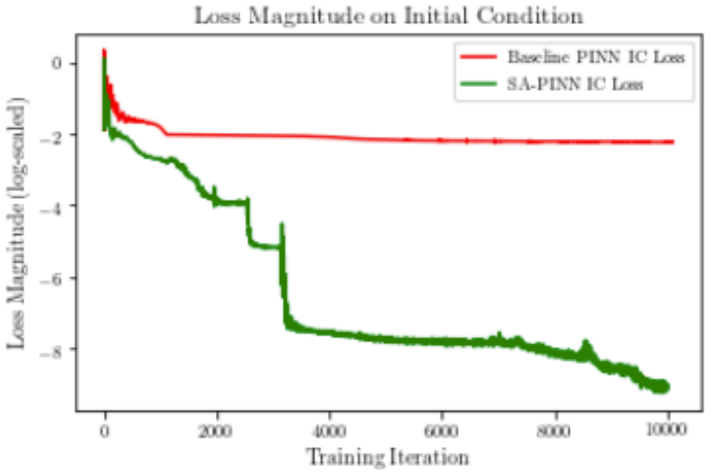
$$\begin{aligned}
\boldsymbol{w}^{k+1} &= \boldsymbol{w}^k - \eta_k\,\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k)\\
\boldsymbol{\lambda}_r^{k+1} &= \boldsymbol{\lambda}_r^k + \rho_r^k\nabla_{\boldsymbol{\lambda}_r}\mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k)\\
\boldsymbol{\lambda}_b^{k+1} &= \boldsymbol{\lambda}_b^k + \rho_b^k\,\nabla_{\boldsymbol{\lambda}_b}\mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k)\\
\boldsymbol{\lambda}_0^{k+1} &= \boldsymbol{\lambda}_0^k + \rho_0^k\,\nabla_{\boldsymbol{\lambda}_0}\mathcal{L}(\boldsymbol{w}^k, \boldsymbol{\lambda}_r^k, \boldsymbol{\lambda}_b^k, \boldsymbol{\lambda}_0^k)\,.
\end{aligned}$$

Ascenso por el gradiente para las λ's

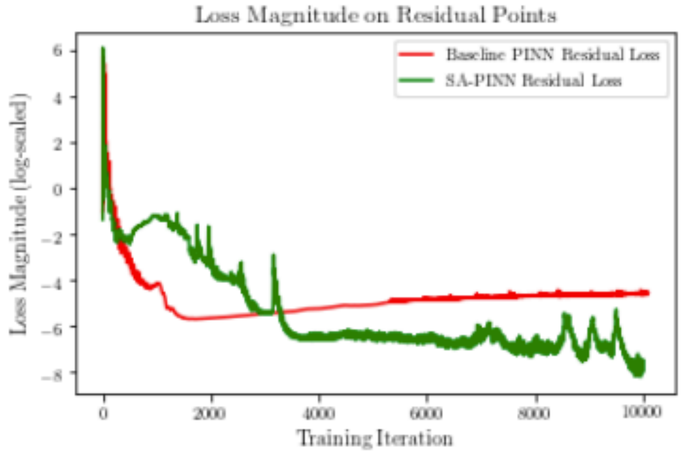
https://arxiv.org/abs/2009.04544

Pesos auto-adapatativos en PINNs

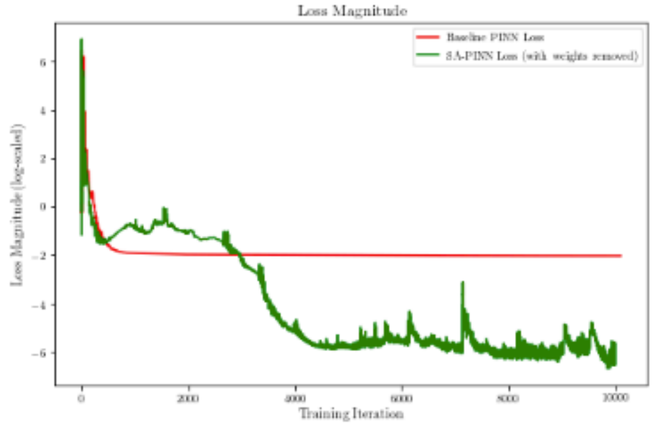
Condiciones iniciales



Residuo físico

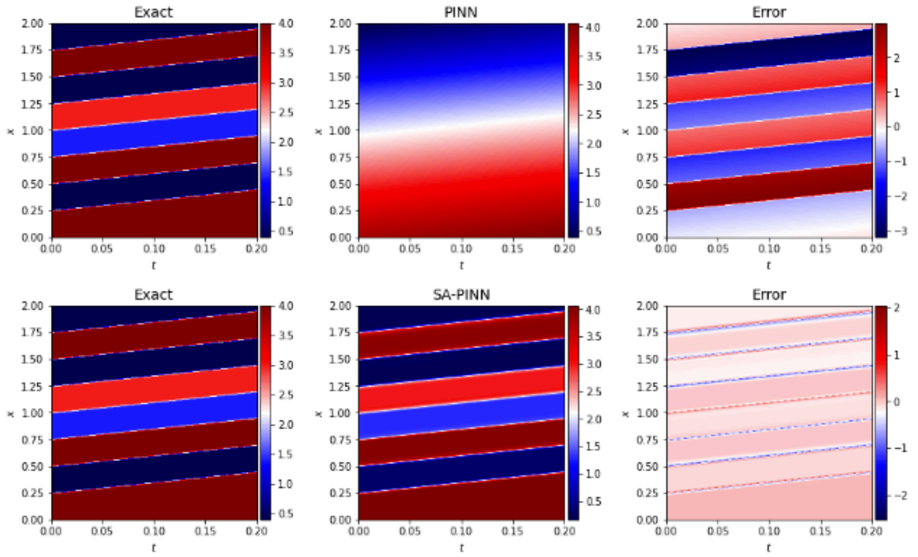


Total



Advección 1D

$$
\begin{aligned}
&q_t(x,t) + \bar{u} q_x(x,t) = 0\,, \quad x \in [0,L]\,,\ t \in [0,T]\,,\\
&u(0,t) = u(L,t) = 0,\ t \in [0,T]\,,\\
&u(x,0) = g(x)\,,\ x \in [0,L]\,.
\end{aligned}
$$



Rojo: baseline Vanilla PINN

Verde: SA-PINN

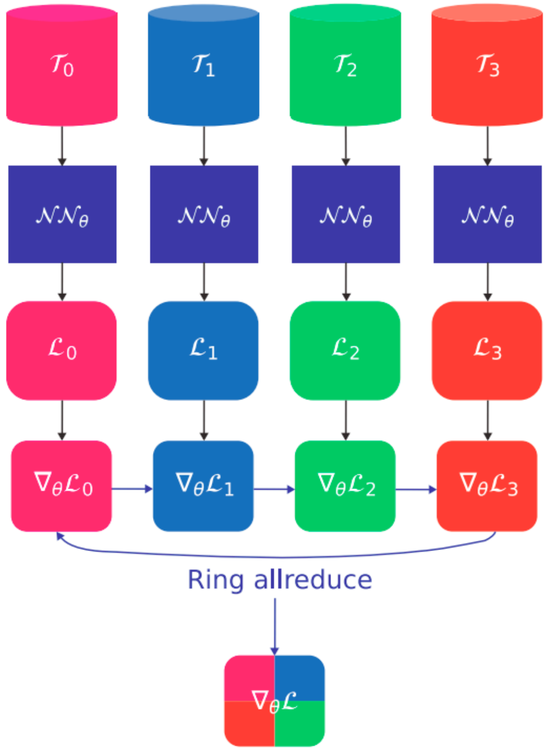
https://arxiv.org/abs/2009.04544

X-PINNs

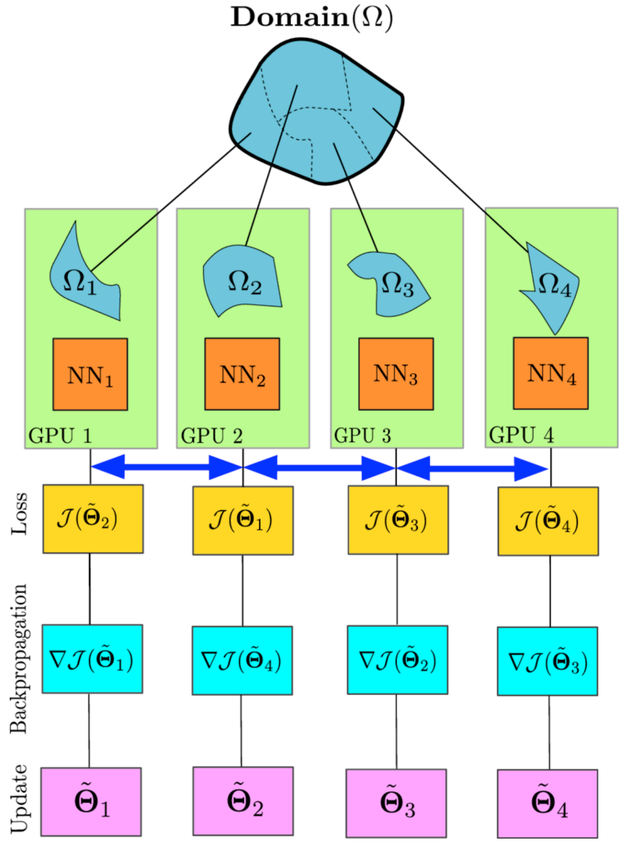
Extended Physics Informed NNs

$$\mathscr{L} = \sum_{k}^{n_{dominios}} (\alpha_0 \mathscr{L}^k_{data} + \alpha_1 \mathscr{L}^k_{fisica} + \alpha_2 \mathscr{L}^k_{contorno} + \alpha_3 \mathscr{L}^k_{CI}) + \sum_{interfaces} \mathscr{L}_{interfaces}$$

Data Distributed Parallel (DDP)

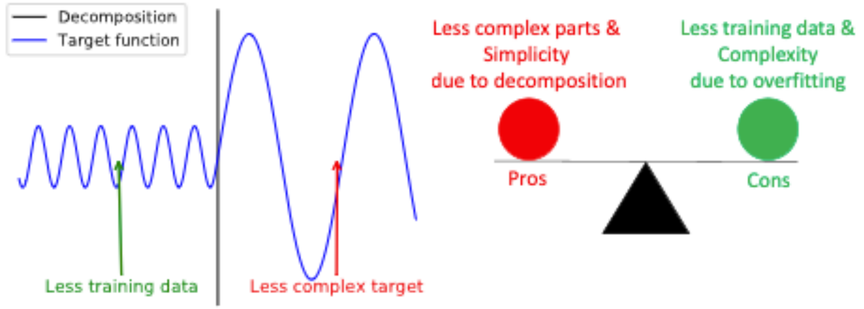


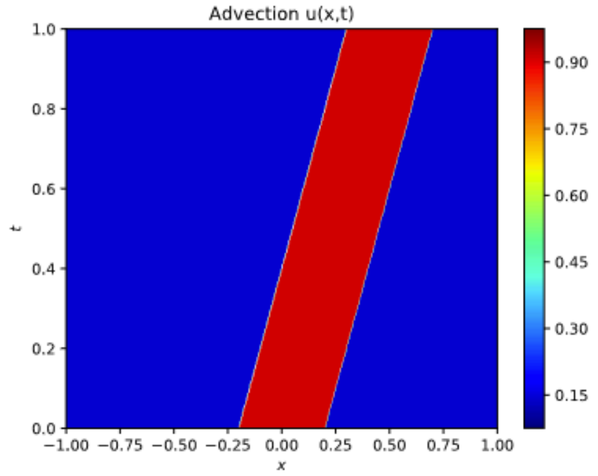
- Entrenamos $n_{dominios}$ redes…
- Paralelizable!

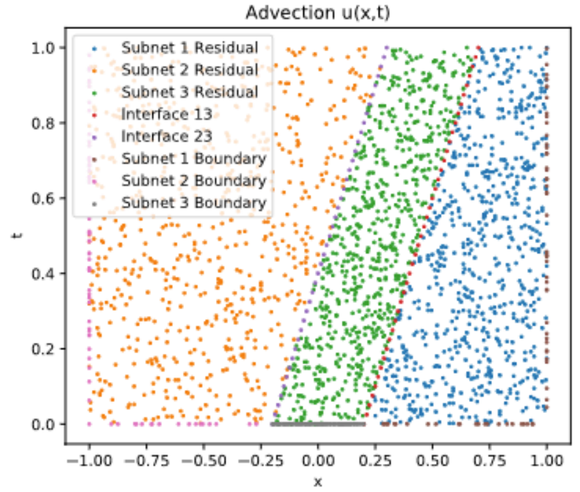


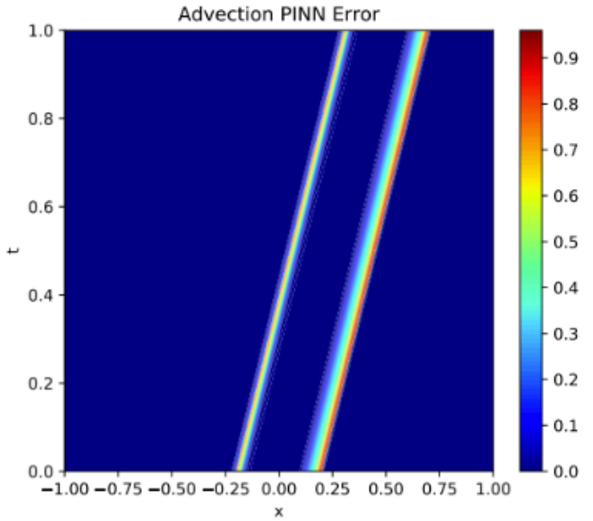
https://arxiv.org/pdf/2109.09444

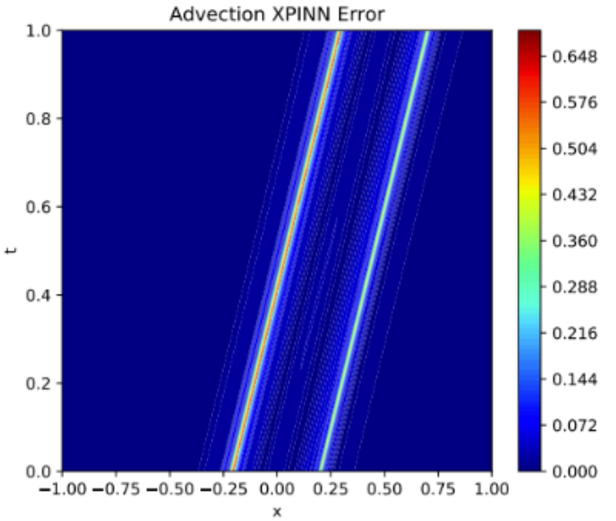
X-PINNs











https://arxiv.org/pdf/2109.09444

Optimización Bayesiana

Proceso Gaussiano

$$\mathscr{L} = \mathscr{L}(\theta_h) \text{ con } \theta_h \text{ hiperparámetros}$$

Ejemplo: $\theta_h = \{\eta, \alpha_{fisica}, \alpha_{CC}, \alpha_{CI}, n_{hidden}, n_{layers}\}$

$$\arg\min_{\theta_h} \mathscr{L}_{total}$$

Proceso Gaussiano: $\mathscr{L}(\theta_h) \sim \mathscr{GP}(\mu(\theta_h), k(\theta_h, \theta_h'))$

Distribución sobre funciones posibles que podrían explicar nuestros datos

$$\mathscr{L}(\theta_h) \,|\, \text{datos} \sim \mathcal{N}(\mu(\theta_h), \sigma^2(\theta_h))$$

https://scikit-optimize.github.io/stable/modules/generated/skopt.acquisition.gaussian_ei.html?highlight=ei#skopt.acquisition.gaussian_ei

Optimización Bayesiana

Proceso Gaussiano

- Antes de observar datos (entrenar en cierto conjunto de $\theta_h^{(i)}$ iniciales), el proceso gaussiano considera un espacio de optimización suave, que está codificado en el kernel.

- Luego de observar (realizar los entrenamientos) el proceso gaussiano se convierte en una distribución de probabilidad condicionada, que devuelve valores para $\mu(\theta_h), \sigma^2(\theta_h)$, i.e. cuando vale la media de la función de costo en el espacio de hiperparámetros, y qué **incertidumbre** tenemos al respecto.

https://scikit-optimize.github.io/stable/modules/generated/skopt.acquisition.gaussian_ei.html?highlight=ei#skopt.acquisition.gaussian_ei

Optimización Bayesiana

Proceso Gaussiano

Proceso iterativo:

1. Seleccionamos ciertos valores iniciales $\theta_h^{(i)}$ con $i = 1,...,n$
2. Entrenamos PINNs para todo $\theta_h^{(i)}$
3. Ajustamos proceso Gaussiano (GP) (maximizando log-verosimilitud)
4. El GP recomienda el siguiente punto θ_h^t a evaluar basándose en las regiones del espacio de hiperparámetros donde la incerteza sea mayor o donde la función de costo ya sea buena*

https://scikit-optimize.github.io/stable/modules/generated/skopt.acquisition.gaussian_ei.html?highlight=ei#skopt.acquisition.gaussian_ei

Optimización Bayesiana

Maxmizar la mejora esperada (Expected Improvement)

Sea $\mathscr{L}^*$ el mejor valor observado hasta el momento. La mejora al explorar θ_h se define como

$$I(\theta_h) = \max(\mathscr{L}^* - \mathscr{L}(\theta_h), 0).$$

Por ende, la **mejora esperada** se escribe como el valor esperado de la mejora:

$$EI(\theta_h) = \mathbb{E}[\max(\mathscr{L}^* - \mathscr{L}(\theta_h), 0)]$$

$$EI(\theta_h) = (\mathscr{L}^* - \mu(\theta_h))\Phi(z) + \sigma(\theta_h)\phi(z), \quad z = \frac{\mathscr{L}^* - \mu(\theta_h)}{\sigma(\theta_h)}$$

Explotación exploración

Φ CDF de la normal,

ϕ PDF de la normal

https://deepxde.readthedocs.io/en/latest/demos/pinn_forward/helmholtz.2d.dirichlet.hpo.html

Teórico práctico semana 3: optimización bayesiana