UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

PRÁCTICA PROFESIONAL III

Estudio y simulación de EDP's utilizando técnicas de deep learning

Practicante: Fernando Fêtis
Evaluador: Hugo Carrillo
Investigador post-doctoral.

Periodo de práctica: 1 de febrero – 28 de febrero.

Otoño, 2023.

Resumen práctica profesional

En la primera práctica realizada se complementó el apunte del curso de aprendizaje de máquinas del DIM, agregando demostraciones formales a los teoremas expuestos y complementando el contenido con otras referencias. En esta segunda Y tercera práctica, se continuó en la línea del aprendizaje automático pero ahora utilizando redes neuronales para simular ecuaciones en derivadas parciales.

Como continuación de la práctica II, en este trabajo se buscó implementar de manera eficiente distintos resultados adicionales a las PINN's con tal de poder hacer más robusto el estudio de este método. En particular, se estudió como extender el problema a dominios no rectangulares y a dominios rectangulares en una dimensión finita arbitraria. Además, se estudiaron e implementaron técnicas de cuantificación de incertidumbre para evaluar la simulación de la red neuronal. Por último, se desarrollo un framework en Python para unificar todos las features desarrolladas y obtener una librería robusta y consistente para la simulación de EDP's utilizando PINN's.

Contexto

Dado que este trabajo consiste en una continuación de la práctica anterior, se omitirá la explicación en detalle de una red neuronal y del método PINN para evitar redundancias y poder profundizar más los temas desarrollados.

Una PINN consiste en una red neuronal (generalmente fully connected) que busca aproximar la solución de una EDP mediante la minimización de su error de aproximación (y condiciones iniciales o de borde cuando existan). En este trabajo se mostrará como se estudió el problema de generalizar este enfoque a problemas más complejos (sistemas de ecuaciones, problema de continuidad única, dominios no rectangulares, etc.).

Objetivos y metodología

Luego de tener estudiadas las diferentes arquitecturas y maneras de ajustar los hiperparámetos de una red PINN, se comenzó la tarea de complementar dicho método con otros resultados del estado del arte. A continación se enumeran los lineamientos generales de este trabajo:

- Analizar diferentes ténicas de sampling de los puntos del dominio (uniforme, mayor masa en los bordes, etc.) de acuerdo a los distintos tipos de ecuaciones.
- Generalización del código previo a problemas en mayores dimensiones. Los trabajos realizados para la práctica II constitían únicamente en EDP's de dos variables.
- Generalización de la técnica a dominios (2D) no rectangulares. Se estudiaron diferentes técnicas de sampling en polígonos y en figuras más complejas.
- Modificar la técnica para permitir la simulación de sistemas de ecuaciones (con dominio no rectangular o de alta dimensión en caso de ser necesario).
- Unificación de todo el código en una librería única que sea fácil de usar.

Al igual que antes, se utilizó la librería PyTorch de Python para aprovechar sus propiedades de diferenciación automática.

Desarrollo

Técnicas de sampling

Una de las primeras variaciones que se le realizamos al método consistió en cambiar la forma de generar los puntos dentro del dominio. Esto fue motivado al observar que la red neuronal tenía un mal comportamiento en ciertas zonas más complicadas de aprender (posiblemente debido al spectral bias estudiado en la práctica anterior).

Para esto, se probó hacer sampling a partir de una distribución beta que le diera mayor peso a las zonas problemáticas, esto con el fin de penalizar más fuertemente el error en esas zonas. También se probó un samplig evolutivo en cuanto a la distribución de la masa dentro del dominio. Por último, se estudió la diferencia de comportamiento al hacer sampling fijo (al inicio del entrenamiento) y por época (cambiando los puntos en cada época de entrenamiento).

Si bien hubo diferencias para ciertos casos, en general pudimos concluir que la elección del método de sampleo no influye considerablemente en el performance final de la red. Por esto, se decidió dejar un sampleo uniforme ya que es el menos costoso desde el punto de vista computacional. Además, tiene la propiedad de ser el que maximiza la entropía, por lo que puede ser considerado como un prior no informativo sobre el dominio.

Problemas para dominios no rectangulares

Este problema fue resuelto utilizando dos enfoques distintos:

- Para definir los diferentes se utiliza una imagen jpg donde está dibujado el dominio. Luego, se reconoce el rectángulo minimal que encierra dicho dominio y se samplea del dominio utilizando sampling uniforme sobre el rectángulo mediante un método de aceptación rechazo. Este método tiene la ventaja de permitir dominios con formas tan compleja como se necesite. La principal desventaja es que dependiendo de la forma del dominio, el sampling a partir del rectánguloi minimal puede ser poco óptimo.
- La segunda técnica utilizada consistió en considerar dominios poligonales, aprovechando una librería ya implementada de Python para crear polígonos. Para el sampling se utilizó una técnica de sampling uniforme en un polígono, la cual consiste en particionar el polígono en triángulo y luego elegir al azar uno de los triángulos (según el área de cada triángulo) y luego hacer sampling uniforme sobre el triángulo elegido. En el reporte adjunto se indica en detalle el algoritmo.

  Dos ejemplos de sampling realizado para este tipo de dominios son los siguientes:

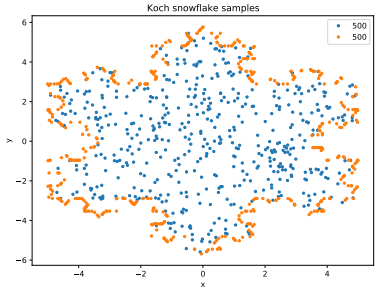


Figura 1: Sampling de un dominio con forma de copo de nieve de Koch. El color naranjo indica el sampling en el borde mientras que el color azul es un sampling en el interior.

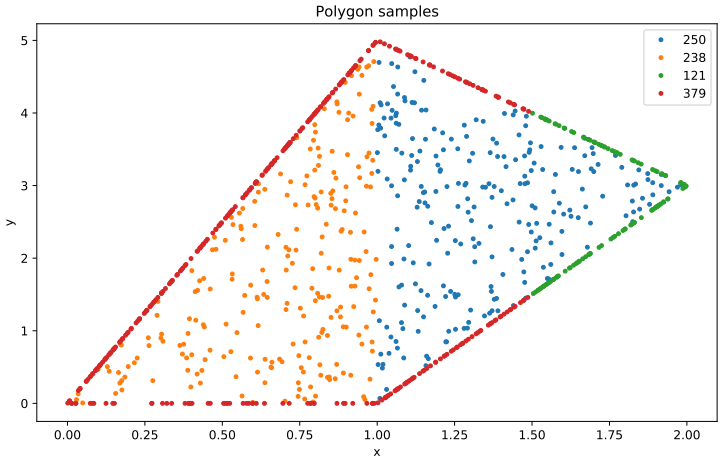


Figura 2: Sampling de un dominio poligonal definido en varias zonas (para poder colocar condiciones de borde distinta en cada región).

0.1. Desarrollo de librería para PINN

La mayor contribución de nuestro trabajo fue la creación de una librería para simular de manera fácil la solución de una EDP utilizando redes neuronales. A continuación se muestra cómo se utiliza nuestra librería para la siguiente EDP definida sobre un triángulo:

```python
# Import the library:
from pinn import geometry, pde, neuralnetworks, trainer

# Class for the PDE:
class PoissonEquation(pde.BaseClass):

    def __init__(self):
        super().__init__()
        self.domain = geometry.Domain('polygon', vertices=torch.tensor([[0.,0], [1, 0], [1, 1]]))
        self.equation = self.eq
        self.boundary_condition = {'res': self.bc}

    def eq(self, data, solution):
        x, y = data['x'], data['y']
        u = solution[0]
        u_x = self.d(u, x)
        u_y = self.d(u, y)
        u_xx = self.d(u_x, x)
        u_yy = self.d(u_y, y)
        eq = u_xx + u_yy + 10*cos(x + 2*y)
        return (eq, )

    def bc(self, data, normals, solution):
        x, y = data['x'], data['y']
        u = solution[0]
        bc = u - 2*cos(x + 2*y)  # u(x,y) - 2cos(x+2y) = 0 on the boundary.
        return (bc, )
```

Este código es para instanciar la siguiente EDP:

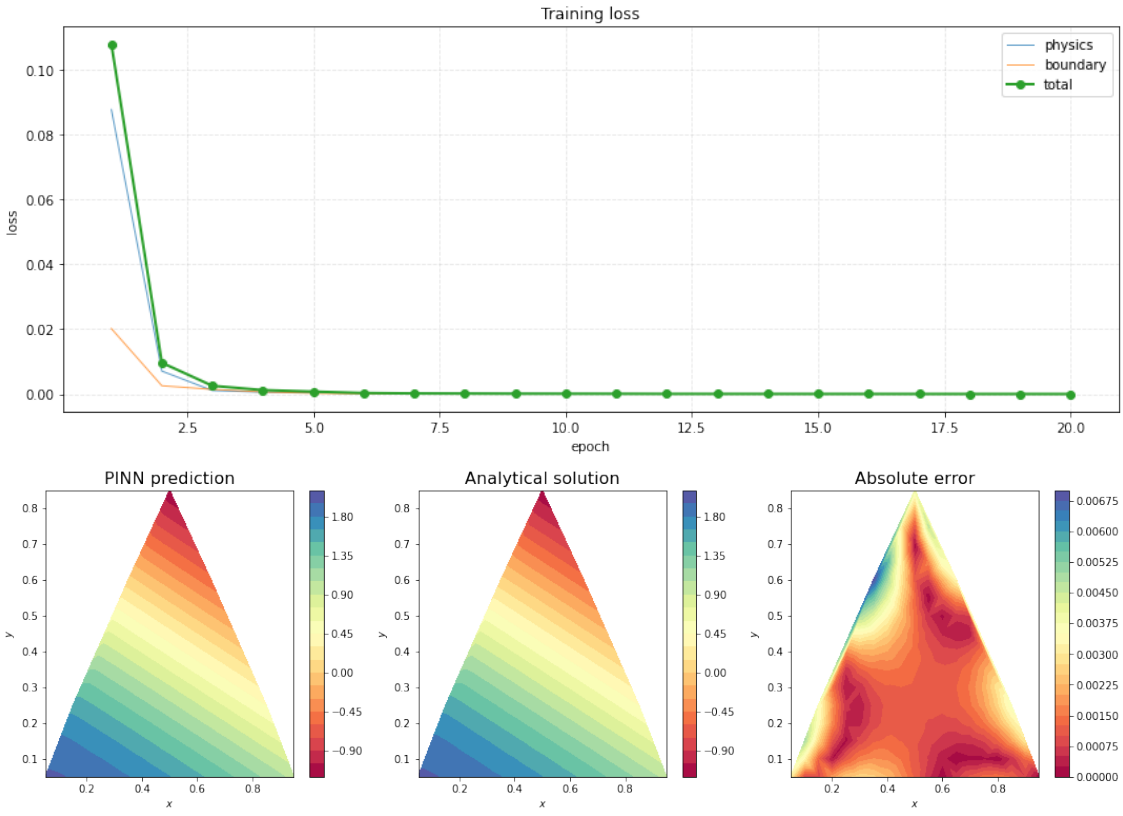
$$\begin{aligned} \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) &= -10\cos(x+2y), && (x,y) \in \Omega \\ u(x,y) &= 2\cos(x+2y), && (x,y) \in \partial\Omega \end{aligned}$$

Para el entrenamiento y gráfico de la red creada, se utilizan los siguientes comandos:

```python
# Objects:
eq = PoissonEquation()
net = neuralnetworks.FullyConnected(n_variables=2, n_output=1, n_layers=3, wide=100)
optimizer = torch.optim.LBFGS(net.parameters(), line_search_fn='strong_wolfe')
tr = trainer.Trainer(eq, net, optimizer)

# Training:
tr.train(epochs=20)
tr.plot_loss()
net.plot_pinn(eq)
```

Al ejecutar estas líneas, se obtienen lo siguiente:



Se observa que la simulación de una EDP en general es muy simple. Esto contrasta con la forma de simular con otros métodos como elementos finitos en FeniCS, donde se vuelve necesario contar con la formulación variacional del problema.

El código de esta librería fue altamente optimizado para evitar la mayor cantidad de bugs y retrasos por mala implementación.

Por último, se añade un ejemplo más para mostrar el cálculo de intervalos de confianza que se realiza mediante dropout. La justificación formal de este método puede ser encontrada en el reporte final que se adjunta en esta entrega.

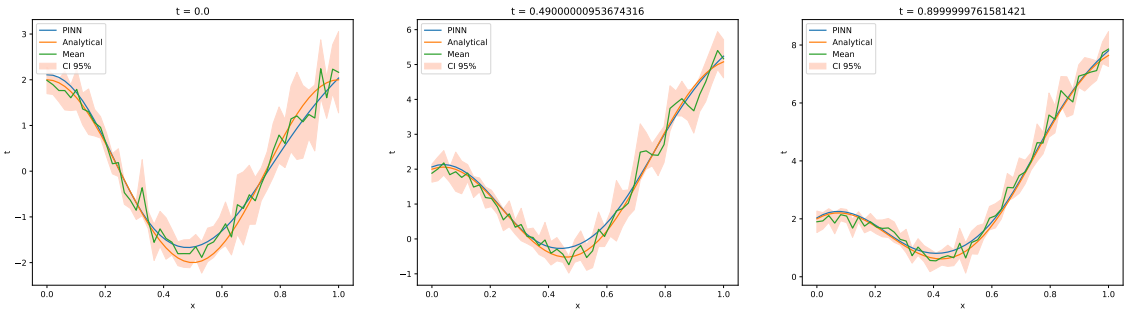


Figura 3: Análisis de incertidumbre de la solución aprendida para una EDO mediante la técnica estocástica de Dropout.

Conclusiones

En este trabajo se unieron las piezas de lo presentado en la práctica II y se agregaron características adicionales a nuestros métodos con el fin de poder tener una mejor simulación de la EDP. Adicional a todo lo que se mencionó en las secciones anteriores, es bueno destacar el siguiente trabajo adicional a modo de conclusión:

- Las redes neuronales no presentan la maldición de la dimensionalidad tan fuertemente como los otros métodos. En particular, se observó que para EDP's de alto orden (mayor a 5), la red neuronal no tuvo problemas para el aprendizaje.
- A pesar de lo anterior, sí se observó que la red tiene problemas para ecuaciones cuya solución es poco regular (o muy compleja). Esto se notó al utilizar intervalos de incertidumbre para este tipo de ecuaciones.
- Observamos que el método de sampling de los puntos para entrenamiento no afecta de forma considerable el rendimiento de la red. Por esto, se dejó el sampling uniforme (y fijo al comienzo del entrenamiento) tal como se propone el en paper original del método.
- Para sampling en dominios curvos (como toroides, elipses u otros) fue necesario el estudio de técnicas de simulación estocásticas por lo que generalizar este problema para cualquier dominio en general puede ser complejo de implementar.
- La librería desarrollada también quedó con otras arquitecturas implementadas. En particular, se dejó una arquitectura encoder-decoder, un transformer encoder y una red fully connected residual.
- Para el problema de continuación única, nuestro trabajo nos permitió observar que las PINN son capaces de aprenderlo para entornos locales (más detalles de esto en el reporte).

- Si bien aquí no se mostró, la librería desarrollada permite estudiar ecuaciones más complejas como sistemas acoplados o problemas inversos (de cualqueir orden). Además, para ecuaciones temporales el programa es capaz de identificar la variable temporal y hacer un gráfico dinámico de la solución.
- Nuestra implementación fue probada en diferentes modelos físicos y en la mayoría se obtuvo buenos resultados. Entre ellos:
  - Se estudió un modelo SIR (y un modelo SIR más avanzado) y vemos que es capaz de resolver el problema inverso para encontrar los parámetros del modelo.
  - Se simuló el enfriamiento de una CPU mediante un disipador término. Las soluciones encontradas eran bastante precisas aunque podían fallar en tiempos iniciales de la simulación.
  - Se simularon ondas en aguas poco profundas utilizando KdV.
  - Se probó con todas las ecuaciones en derivadas parciales clásicas. Como era de esperarse, el método tuvo problemas en EDP's hiperbólicas.

Por último, considero importante recalcar el potencial que tiene este método, dado que entrega buenos resultados y necesita poco conocimiento técnico de EDP's por lo que puede usarse por investigadores poco familiarizados con la teoría detrás de las EDP's.