Neural networks for PDEs

Miguel López Pérez

Universidad de Granada
e-mail: miguellopezjpc@correo.ugr.es

Abstract. En el presente trabajo repasaremos los métodos que involucran redes neuronales artificiales para resolver ecuaciones diferenciales. Expondremos unos ejemplos de un método para la resolución de EDOs y EDPs.

1 Introducción

Este trabajo versará sobre la resolución de ecuaciones diferenciales y ecuaciones en derivadas parciales mediante técnicas de redes neuronales artificales.

Estas ecuaciones son bien conocidas en campos como la física, ciencias de la computación, biología, química o economía ya que modelizan sistemas dinámicos complejos. Expresan el comportamiento de procesos como puede ser el movimiento de un fluido o los cambios de la oferta y demanda según el precio de un producto. Desde el siglo XVIII, son conocidas en matemáticas, aunque se dispone de una gran teoría muchas de ellas se escapan de una resolución explícita dejando lugar a resultados teóricos sobre la soluciones, como puede ser su existencia, unicidad, estabilidad o comportamiento asintótico. Por lo tanto, a efectos prácticos, como mucho podemos aspirar a simular las ecuaciones mediante técnicas de métodos numéricos los cuales discretizan el espacio aproximando las derivadas de las funciones, como pueden ser los métodos de Runge-Kutta. Esto requiere de gran complejidad a la hora de la resolución numérica ya que se producen inestabilidades en gran parte de los cálculos además de que obtenemos soluciones solamente en una malla discreta del dominio.

En [4] propusieron la resolución de estas ecuaciones mediante redes neuronales artificiales. Aunque ya se habían propuesto anteriormente métodos de resolución de ecuaciones lineales algebraicas, con estás técnicas, las cuales resultaban de la discretización del dominio, los autores presentaron un método general para resolver tanto ecuaciones diferenciales como ecuaciones en derivadas parciales, ya que confía en la capacidad de aproximación de las redes neuronales (son bien conocidas por ser aproximadores universales, lo cual motiva su uso en este campo). Este método tiene grandes ventajas respecto a otras formas clásicas de resolución ya que las soluciones que se obtienen son diferenciables y tienen una forma analítica cerrada, el número de parámetros del modelo es bastante reducido, puede ser aplicado tanto a ecuaciones diferenciales ordinarias como sistema de ecuaciones diferenciales ordinarias o ecuaciones en derivadas parciales y el método puede ser implementado eficientemente en arquitecturas paralelas. En

la siguiente sección explicaremos en detalle el método propuesto y adjuntaremos unos resultados obtenidos por dicho método.

A partir de este nuevo paradigma de resolución de ecuaciones han salido otros trabajos [1] los cuales hacen uso de este método para resolver ecuaciones concretas más complejas como puede ser el problema de Stokes para modelizar fluidos, el cual utiliza este método tres veces ya que es un sistema de ecuaciones derivadas parciales.

Recientemente, en [5] proponen no sólo la resolución de las ecuaciones en derivadas parciales sino encontrar las propias ecuaciones que rigen ciertos sistemas a través de datos recolectados siendo procesados mediante Deep Learning. Presentan una red feedforward llamada PDE-Net, la cual tiene la forma genérica de una ecuación de evolución. El objetivo de la red es aprender la forma de la respuesta no lineal de la función y realizar predicciones precisas. A diferencia del trabajo existente, la red propuesta solo requiere un conocimiento menor en la forma de la función de respuesta no lineal, y no requiere conocimiento sobre los operadores diferenciales involucrados (a excepción de su orden máxima posible) y sus aproximaciones discretas asociadas. La función de respuesta no lineal se puede aprender utilizando redes neuronales u otras métodos de aprendizaje automático, mientras se aprenden aproximaciones discretas de los operadores diferenciales usar núcleos de convolución (es decir, filtros, los cuales relacionan con distintas aproximaciones de los operadores diferenciales.) conjuntamente con el aprendizaje de la función de respuesta. Si tenemos un conocimiento previo sobre la forma de la función de respuesta, podemos ajustar fácilmente la arquitectura de la red aprovechando la información adicional. Esto puede simplificar el entrenamiento y mejorar los resultados.

El trabajo [2] aborda el problema de ecuaciones diferenciales con alta dimensión y ecuaciones diferenciales estocásticas. Desarrollar algoritmos numéricos eficientes para alta dimensión (por ejemplo, cientos de dimensiones) es una tarea complicada. Las ecuaciones en derivadas parciales (EDP) han sido una de las tareas más desafiantes en matemáticas aplicadas. Como es bien sabido, la dificultad radica en la "maldición de la dimensionalidad", es decir, a medida que crece la dimensionalidad, la complejidad de los algoritmos crece exponencialmente. Exploran el uso del deep learning para resolver problemas de alta dimensión general EDPs. Para este fin, es necesario formular las EDPs como un problema de aprendizaje. Exploran una conexión entre PDEs parabólicas (no lineales) y ecuaciones diferenciales estocásticas inversas (BSDE) ya que las BSDEs comparte muchas características comunes con problemas de control estocástico.

2 Método propuesto original

En esta sección hablaremos del método propuesto en el paper original [4] de este campo.

2.1 Descripción del método

Consideramos la siguiente ecuación diferencial general:

$$G(x, \Psi(x), \nabla\Psi(x), \nabla^2\Psi(x)) = 0, \qquad x \in D \subset \mathbb{R}^n. \tag{1}$$

sujeto a ciertas condiciones de contorno (por ejemplo, Dirichlet, fijamos un valor para la solución en la frontera, o Neumann, fijamos el valor de su derivada en la frontera). Sin perder generalidad, podemos considera ecuaciones de grado superior pero para simplificar notación y cálculos consideramos las de este tipo.

Consideramos una discretización del dominio $\hat{D}$ y, por tanto, una solución de (1) debe verificar el siguiente sistema de ecuaciones:

$$G(x_i, \Psi(x_i), \nabla\Psi(x_i), \nabla^2\Psi(x_i)) = 0, \qquad \forall x_i \in \hat{D} \tag{2}$$

sujeto a las condiciones de contorno.

Si $\Psi_t(x, p)$ es una solución de prueba (obtenemos una forma paramétrica de la solución e intentamos que se ajuste a nuestro problema) con parámetros p, el problema es transformado a

$$min_p \sum_{x\hat{D}} G(x, \Psi t(x_i, p), \nabla\Psi(x_i, p), \nabla^2\Psi(x_i, p)) \tag{3}$$

sujeto a las restricciones impuestas por las condiciones de contorno.

En este método de usar soluciones de prueba Ψ_t usa una red neuronal feed-foward y los parámetros p corresponde a los pesos y la bias de la arquitectura de la red.

Imponemos que cumpla las soluciones de prueba las condiciones de contorno. Para ello la escribiremos como la suma de dos términos:

$$\Psi_t(x) = A(x) + F(x, N(x, p)) \tag{4}$$

donde N(x,p) es la salida de la red neuronal feedforward con parámetros p y de entrada x.

El término $A(x)$ contiene parámetros no ajustables y satiface las condiciones de frontera. El término F está construido de manera que no afecte a la condición de contorno, ya que $\Psi_t(x)$ debe también satisfacerlas. Este término debe de ajustarse para que los parámetros (los pesos y las biases) resuelvan el problema de minimización sin restricciones.

2.2 Cálculo del Gradiente

La minimización de la función (3) puede ser considerado como el proceso de entrenamiento de una red neuronal artificial donde el error correspondiente a cada elemento x_i de entrada es el valor de $G(x_i)$ el cual tiene que convertirse en cero. El cálculo del valor de este error involucra no sólo la salida de la red (como en el entrenamiento convencional) sino también las derivadas de la salida con respecto a cualquiera de sus entradas. Por lo tanto, en el cálculo del gradiente del error con respecto a los pesos de la red, necesitamos no sólo el gradiente de la red sino también el gradiente de la red respecto a sus entradas.

Consideramos un perceptron multicapa con n unidades de entrada, una capa oculta con H unidades de sigmoide y una unidad de salida lineal. La extensión al caso de más capas ocultas puede obtenerse sin dificultad. Para un elemento $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ la salida de la red es de la forma $S = \sum_{i=1}^{H} v_i \sigma(z_i)$, donde $z_i = \sum_{i=1}^{n} w_{ij} x_j + u_i$, w_{ij} es el peso de la unidad j de entrada de la unidad oculta i, v_i es el peso desde la unidad oculta i a la salida, u_i es el bias de la unidad oculta i y $\sigma(z)$ es la función de transferencia de tipo sigmoide. Es claro que,

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{H} v_i w_{ij}^k \sigma_i^{(k)}$$

donde $\sigma_i = \sigma(z_i)$ y $\sigma^{(k)}$ denota la derivada de orden k-ésima de la sigmoide. Además es evidente que:

$$\frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}} \cdots \frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}} = \sum_{i=1}^{n} v_i P_i \sigma_i^{(\Lambda)} \tag{5}$$

donde $P_i = \prod_{k=1}^{n} w_{ik} \lambda_k$ y $\Lambda = \sum_{i=1}^{n} \lambda_i$.

La ecuación (5) indica que la derivada de la red con respecto a cualquiera de las entradas es equivalente a una red neuronal feedforward $N_h(x)$ con una capa oculta, teniendo los mismos valores para los pesos w_{ij} y umbrales u_i y reemplazando cada peso v_i por $v_i P_i$. Además la función de transferencia de cada unidad oculta es reemplazada con la derivada de orden Λ de la sigmoide.

Por tanto, el gradiente de N_g con respecto a los parámetros de la red original pueden ser obtenidos fácilmente como:

$$\frac{\partial N_g}{\partial v_i} = P_i \sigma_i^{(\Lambda)} \tag{6}$$

$$\frac{\partial N_g}{\partial u_i} = P_i \sigma_i^{(\Lambda+1)} \tag{7}$$

$$\frac{\partial N_g}{\partial w_{ij}} = x_j v_i P_i \sigma_i^{(\Lambda+1)} + v_i \lambda_j w_{ij}^{\lambda_j - 1} \left(\prod_{k=1, k \neq j}^{} w_{ij}^{\lambda_k} \right) \sigma_i^{(\Lambda)} \tag{8}$$

Una vez que la derivada del error con respecto a los parámetros de la red ha sido definida entonces podemos usar casi cualquier técnica de minimización.

Podemos usar, el algoritmo de propagación hacia atrás, el método del gradiente conjugado u otras técnicas propuestas en la literatura. En [4] proponen usar el método BFGS [3] ya que converge cuadráticamente y da buenos resultados. Nótese que para un punto del mallado dado las derivadas de cada red con respecto de los parámetros puede ser obtenido simultáneamente en el caso que contemos con hardware paralelo.

2.3 Ejemplo: Ecuación lineal y ecuación logística (EDO)

El primer ejemplo que presentamos es la EDO lineal de primer orden, correspondiente al modelo de Malthus, al crecimiento ilimitado de poblaciones. La ecuación corresponde a

$$\Psi(x)' = \Psi(x), \qquad [t \in 0, 1]$$

En la siguiente gráfica comparamos la solución analítica del problema junto a la predicción de la red neuronal y de otra solución numérica mediante el esquema de diferencias finitas.

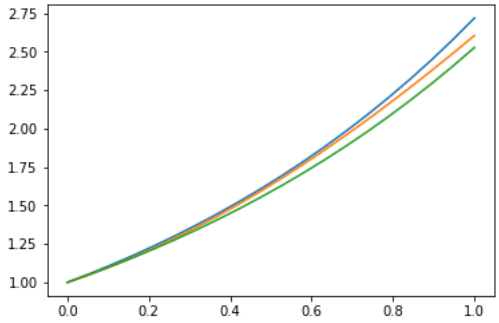


Fig. 1. Azul: Solución analítica, Rojo: Red neuronal, Verde: Diferencias finitas

El siguiente ejemplo corresponde a la ecuación logística. La cual modela el crecimiento de una población con capacidad de carga del medio (no existe el crecimiento ilimitado).

La ecuación diferencial ordinaria sería:

$$N'(t) = rN(t)\left(1 - \frac{N(t)}{P}\right), \qquad t \in [0, 2]$$

donde r es la tasa de crecimiento y P la capacidad de carga del medio, los parámetros propios de la ecuación. Para que esté bien definida y cuente con existencia y unicidad de solución, habría que dar una condición inicial, en nuestro ejemplo es: $N(0) = N_0 = 3$.

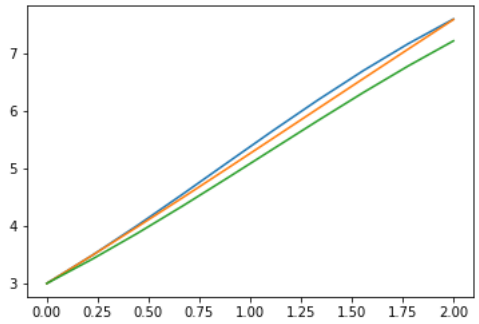


Fig. 2. Azul: Solución analítica, Rojo: Red neuronal, Verde: Diferencias finitas

2.4 Ejemplo: Ecuación en derivadas parciales

También podemos resolver problemas de contornos que involucran ecuaciones en derivadas parciales, para esto consideramos la ecuación de Laplace:

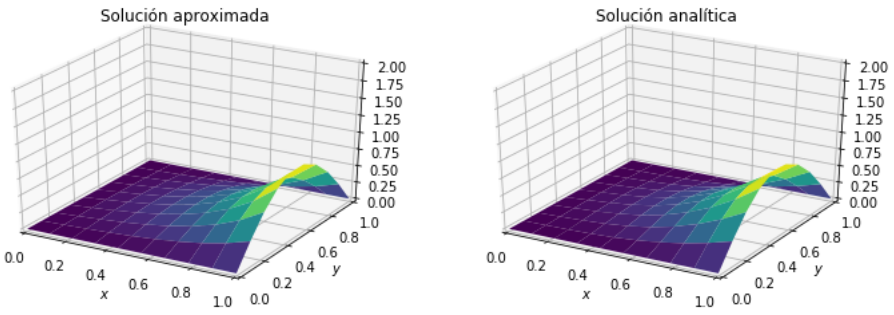
$$\frac{\partial^2}{\partial x^2}\Psi(x,y) + \frac{\partial^2}{\partial y^2}\Psi(x,y) = 0, \quad x, y \in [0,1]$$

sujeto a las siguientes condiciones de contorno:

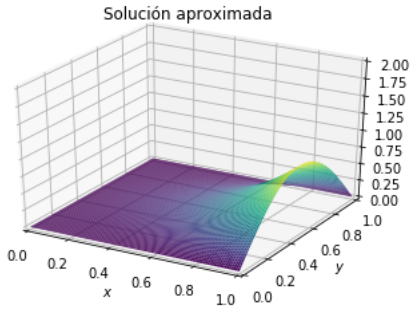
$$\Psi(0,y) = \Psi(1,y) = \Psi(x,0) = 0, \qquad \forall x, y \in [0,1]$$

$$\Psi(x,1) = sin(\pi x), \qquad \forall x \in [0,1]$$

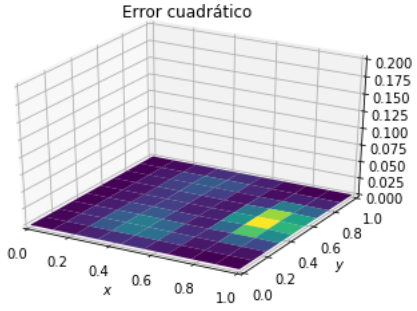
Este problema se puede resolver analíticamente, así que a continuación mostramos su solución explícita y la aproximada:



Una de las ventajas que hemos comentado antes es que gracias a las redes neuronales obtenemos soluciones diferenciables y no sólo definidas en un conjunto discreto de punto. Por ello, podemos evaluar la solución en más puntos del dominio como vemos a continuación:



La aproximación que hemos conseguido es muy buena, podemos ver un gráfico del error cuadrático el cual está muy cercano a 0 en todos los puntos.



3 Conclusiones

Teóricamente, todos los métodos que hemos repasado están bien propuestos. En la práctica, tenemos la virtudes y defectos de las redes neuronales. Aunque podamos simular cualquier ecuación aún llevando consigo no linealidades lo cual es bastante difícil con la teoría clásica. Tenemos que entrenar redes neuronales lo cual puede llevar bastante tiempo, ya que dependiendo de la tasa de aprendizaje de la red, aprenderá más rápido o más lentamente. Empíricamente, hemos comprobado que tasas de aprendizaje más altas conlleva a un aumento en la velocidad de convergencia pero algunos métodos se vuelven inestables alcanzando overflow y la posterior no convergencia del método. Para ello, podría ser interesante repasar otros métodos de minimización de funciones.

References

1. M. Baymani, S. Effati, H. Niazmand, and A. Kerayechian. Artificial neural network method for solving the navier—stokes equations. *Neural Comput. Appl.*, 26(4):765–773, May 2015.
2. Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic

differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.
3. R. Fletcher. *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA, 1987.
4. I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *Trans. Neur. Netw.*, 9(5):987–1000, September 1998.
5. Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-Net: Learning PDEs from Data. *ArXiv e-prints*, October 2017.
6. Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017.
7. Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations. *CoRR*, abs/1711.10566, 2017.