MATEMÁTICAS DEL DEEP LEARNING

FERNANDO VADILLO

Resumen. Entender el mundo a través los modelos matemáticos, ha sido siempre el gran reto de la Ciencia. Sin embargo, en los últimos años la llamada Inteligencia Artificial ha desarrollado nuevas técnicas que usando sólo grandes bolsas de datos y las modernos recurso computacionales, aproximan los modelos que parecían inaccesible. La Inteligencia Artificial que estimula pero también asusta, quizá séa el nueva paradigma que renuevo el mundo cientítido y técnico, con todas las consecuencia sociales. En este documento se explica matemáticamente una de las técnicas más exitosa de los últimos años: Deep Learning Networks (DNNs).

Índice

1. Introducción 2
2. Redes Neuronales 3
2.1. La neurona y el perceptrón 3
2.2. Las funciones de activación 4
2.3. Un ejemplo de una Red Neuronal Artificial 5
2.4. Resolución de un ejemplo práctico 6
2.5. Configuración general de una red. 7
3. Entrenamiento de Redes Neuronales Prealimentadas 9
3.1. Versión con muestreo con reemplazamiento. 9
3.2. Versión con muestreo sin reemplazamiento. 10
3.3. Versión mini-batch. 10
3.4. Propagación hacia atrás. 10
4. Algunas aplicaciones del Depp Learning 11
4.1. Sparse Identification of Nonlinear Dynamics (SINDy) 11
4.2. Physics-informed neural networks (PINN) 11
Referencias 12

1. Introducción

La Artificial Intelligence (AI) es una preocuación y comentario de muchos de nosotros, son muchos los texto publicados recientemente para divulgar su contenido [24], [16], [3], [23], [22], [7], [6], [15], con un importante lado crítico en [25] y [19] . La AI es la combinación de algoritmos planteados con el objetivo de crear máquinas

Received by the editors 20 de enero de 2025.

que presenten las mismas capacidades que los humanos, como por ejemplo, hablar, reconocer objetos y sonidos, traducir, etc..

Aunque este artículo está pensado en lectores científicos y técnico, la AI también afecta a personas de humanidades y sobre todo de humanidades cognitivas. En el reciente y divertido artículo [8] de la prestigiosa linguista Violeta Demonte, la autora comenta los modelos de generativos de lenguaje conocidos como Large Lenguage Models (MGL). La autora relata sus conversaciones con ChatGPT de la compañía Open AI cofundado por el conocido Sam Altman en 2015, capaz de generar textos escritos y hablados, responder a peticiones de información, hacer resúmenes y sumarios en una prosa, en su opinión, no excesivamente elegante, pero siempre fluido y coherente.

Por otra parte, los humanos somos una raza que aprende constantemente de su entorno, con sus aciertos y errores, incluso de otros individuos, y por ello, existe también una amplia rama dentro de la Inteligencia Artificial que intenta trasladar este acto a las máquinas: el Machine Learning (ML). El Machine Learning podría definirse como la práctica de usar algoritmos para obtener datos, aprender de ellos y realizar predicciones sobre algo en concreto; es decir, tiene como objetivo que mediante la experiencia y ciertos algoritmos predefinidos, la máquina aprenda automáticamente y obtenga respuestas, ya sea siendo supervisada o no. En nuestro caso, nos centraremos en el Deep Learning (DL), una técnica para hacer Machine Learning que mediante un algoritmo automático y estructurado, busca imitar el aprendizaje humano, con el fin de obtener ciertos conocimientos. Mientras que los algoritmos tradicionales de ML son lineales, los algoritmos de DL se basan en una estructura de complejidad y abstracción que va aumentando a cada paso que se da.

Pongamos un ejemplo para entender como funciona el DL. Imaginemos a un bebé que está aprendiendo a reconocer objetos, como por ejemplo un coche. El bebé aprende qué es y qué no es un coche, señalando objetos y diciendo la palabra coche. El padre o madre dice: *Si, eso es un coche* o *No, eso no es un coche*. A medida que el niño continúa señalando objetos, se va dando cuenta de las características que poseen todos los coches. Lo que hace el niño, sin saberlo, es aclarar el concepto de coche mediante la construcción de una estructura, en la que cada nivel de abstracción se crea con el conocimiento que se obtuvo de la capa anterior.

Los programas que usan el DL pasan por el mismo proceso: cada algoritmo en la jerarquía aplica una transformación no lineal a su entrada, y usa lo que aprende para crear un modelo estadístico como salida. Las iteraciones continúan hasta que la salida ha alcanzado un nivel de precisión aceptable que previamente se ha elegido. La cantidad de capas de procesamiento a través de las cuales deben pasar los datos es el motivo por el cual el DL lleva la palabra *Deep* en su nombre.

En el aprendizaje automático tradicional (Machine Learning), el proceso de aprendizaje se supervisa, y el programador tiene que ser muy específico cuando le dice a la máquina qué tipo de cosas debería buscar para decidir si cierta imagen contiene un coche o si no. Este es un proceso complejo, y la tasa de éxito depende completamente de la capacidad del programador para definir con precisión un conjunto de características para los coches. En el DL, en cambio, es el programa el que crea el conjunto de características por sí mismo, sin supervisión, donde además, el aprendizaje no supervisado no solo es más rápido, sino que generalmente es más

preciso.

El DL es una técnica muy estudiada porque se está aplicando con enorme éxito en muchos campos, además, presenta un acercamiento muy íntimo del modo de funcionamiento del cerebro humano lo que le hace enormemente sugerente, quizá conociendo mejor esta técnica se llége a reconocer el funciona nuestro cerebro.

Este artículo se desarrolla de la forma siguiente : primero se describen los caracteres generales de las redes neuronales que después de ilustran con un ejemplo. En la segunda parte se extiende las ideas a contextos más generales explicando los métodos matemáticos aplicados: métodos del gradiente, gradiente estocástico y el algoritmo de propagación hacia atrás. Finalmente se comentan muy brevemente algunas recientes aplicaciones en el contexto de la Matemática Computacional y los Métodos Numéricos.

2. Redes Neuronales

2.1. La neurona y el perceptrón. La neurona es la unidad fundamental del cerebro humano que conectadas entre millones crean una red neuronal. La neurona artificial representada en la Figura 1 recibe unos datos $x_1, \cdots, x_n$ que multiplicados por unos pesos $w_1, \cdots, w_n$ producen el logit de entrada $z = \sum_{j=1}^{n} w_j x_j$ a la que habitualmente se suma una constante sesgo o bias μ. Finalmente una funcióon de activación f produce la salida $y = f(z + \mu)$ que se envía a otra neurona. Esto habitualmente se escribe de forma vectorial:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \ \mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}, \quad y = f(\mathbf{w}^T \cdot \mathbf{x} + \mu).$$

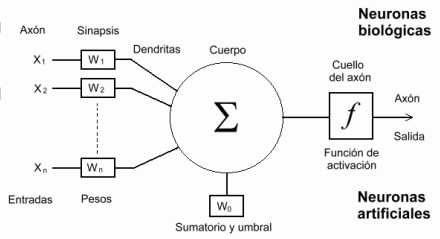

Figura 1. Modelo de Neurona Artificial

2.2. Las funciones de activación.

Definicion 2.1. La función sigmoide es un función $f : \mathbf{R} \to [0, 1]$ tal que:

$$f(z) = \frac{1}{1 + e^{-z}},$$

cuya gráfica se puede ver en la Figure 2. Cuando el logit es muy negativo, la salida es casi cero, mientas que si este es grande su salida es cercana a valor 1, Además,

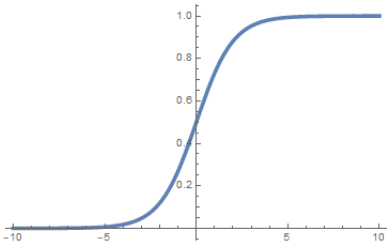

FIGURA 2. Gráfica de la función sigmoide en el intervalo [-10,10]

la función sigmoide presenta una propiedad muy conveniente y es que su derivada $f'(z) = f(z)(1 - f(z))$.

Por otra parte, la posición y lo escarpada que es la curva, viene determinado por escalar y trasladar el argumento, o lo que es lo mismo, ponderando y sesgando la entrada de la neurona. Por tanto, si por ejemplo tomamos la función sigmoide, pero decidimos trasladarla para encontrar el centro en z=-3 y hacerla más escarpada, es decir, $f(4(z-3))$, nos encontraremos con que la función tiene ahora la forma de la Figura 3.

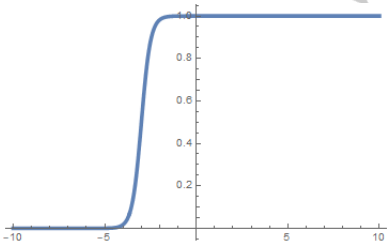

FIGURA 3. Gráfica de la función sigmoide modificada en el intervalo [-10,10].

Definicion 2.2. La función ReLU es un función $f : \mathbf{R} \to [0, +\infty]$ tal que:

$$f(z) = \text{máx}(0, z),$$

cuya gráfica se puede ver en la parte izquierda de la Figure 2.2. Esta función se puede suavizar en la versión denominada softplus

$$f(z) = \ln(1 + e^z),$$

cuya gráfica se puede ver en la parte derecha de la Figure 2.2.

2.3. Un ejemplo de una Red Neuronal Artificial. Imaginemos que tenemos la siguiente red neuronal artificial de 4 capas:

Para la red en la Figura 5 la primera capa (la entrada) viene representada por dos círculos, dado que nuestros puntos como datos de entrada tienen dos componentes. La segunda, tercera y cuarta capa tienen dos, tres y dos círculos, respectivamente, los cuales indican el número de neuronas que se utilizan en dicha capa. Las flechas

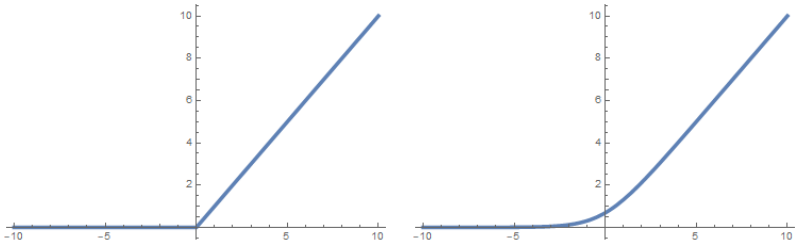

FIGURA 4. Gráficas de la funciones ReLU y Sofplus en el intervalo [-10,10]

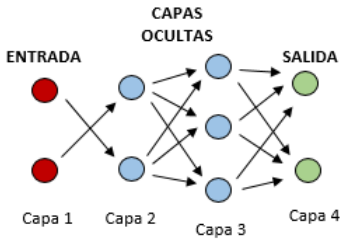

FIGURA 5. Ejemplo de una red con cuatro capas.

que se dirigen de una capa a otra indican la información que se envía desde una neurona, hasta las neuronas de la capa siguiente.

Puesto que los datos de entrada tienen la forma $\mathbf{x}\in \mathbb{R}^2$, los pesos y sesgos para la segunda capa vendrán representados por la matriz de pesos $\mathbf{W}_2 \in \mathbb{R}^{2x2}$ y el vector de sesgos $\mathbf{b}_2 \in \mathbb{R}^2$, respectivamente. Por tanto, la salida desde la segunda capa hacia la tercera tiene la forma

$$f(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2) \in \mathbb{R}^2. \tag{2.1}$$

Como podemos observar, la tercera capa con tres neuronas, cada una de ellas recibiendo datos de entrada en $\mathbb{R}^2$. Entonces, los pesos y sesgos para esta nueva capa vendrán dados por $\mathbf{W}_3 \in \mathbb{R}^{3x2}$ y $\mathbf{b}_3 \in \mathbb{R}^3$, y la salida desde esta tercera capa es tal que

$$f(\mathbf{W}_3 \underbrace{f(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2)}_{Salida\ de\ la\ capa\ 2} + \mathbf{b}_3) \in \mathbb{R}^3. \tag{2.2}$$

De manera análoga, obtendremos, por último, la salida desde la capa 4. Esta será la salida definitiva desde nuestra red de neuronas y tendrá la siguiente forma:

$$F(x) = \underbrace{f(\mathbf{W}_4\, f(\mathbf{W}_3 \underbrace{f(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2)}_{Salida\ de\ la\ capa\ 2} + \mathbf{b}_3) + \mathbf{b}_4)}_{Salida\ de\ la\ capa\ 3} \in \mathbb{R}^2, \tag{2.3}$$

donde $\mathbf{W}_4 \in \mathbb{R}^{2x3}$ y $\mathbf{b}_4 \in \mathbb{R}^2$ son, de nuevo, la matriz de pesos y el vector de sesgos. Esta última expresión define una función $F : \mathbb{R}^2 \to \mathbb{R}^2$ en términos de los 23 parámetros que hay en las matrices de pesos y los vectores de sesgo

2.4. Resolución de un ejemplo práctico. Para explicar como se entrena una red neuronal seguiremos el ejemplo de la referencia [12]: se supone que en el cuadrado unidad del plano se han realizado perforaciones para encontrar petróleo con el resultado separados en dos categorías, los puntos exitosos con un círculo verde y los fracasados con asteriscos rojos que se representan en la Figura 6.

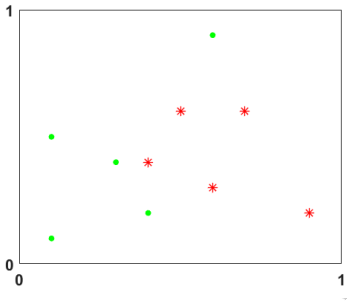

FIGURA 6. Conjunto de puntos y las dos categorías.

Nuestro objetivo ahora es optimizar los 23 parámetros a los que hemos hecho referencia tal que requeriremos que $F(x)$ sea próximo a $\binom{1}{0}$ para los puntos en la categoría A y próximo a $\binom{0}{1}$ para los puntos en la categoría B. Después, dado un punto $x \in \mathbb{R}^2$, sería razonable clasificarlo dependiendo de la componente más larga de $F(x)$; es decir, meterlo en la categoría A si $F_1(x) > F_2(x)$ y en la categoría B si $F_1(x) < F_2(x)$. Esta exigencia sobre la función F la especificaremos definiendo una función de costos sobre los pesos y los sesgos, ya que los datos de entrada son fijos. Si denotamos los diez puntos que tenemos como datos en la Figura 6 como $\{x_i\}, i \in \{1, \ldots, 10\}$, usaremos $y(x_i)$ para la salida, que se define tal que

$$y(x_i) = \begin{cases} \binom{1}{0}, & si \quad x_i \text{ está en la categoría A,} \\ \binom{0}{1}, & si \quad x_i \text{ está en la categoría B.} \end{cases} \tag{2.4}$$

Finalmente, nuestra función objetivo o de costos será:

$$C(\mathbf{W}_2, \mathbf{W}_3, \mathbf{W}_4, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4) = \frac{1}{10} \sum_{i=1}^{10} \frac{1}{2} ||y(x_i) - F(x_i)||_2^2, \tag{2.5}$$

el término $\frac{1}{2}$ en la función se incluye unicamente para simplificar las derivadas.

Para los datos que se muestran en la Figura 6, se han utilizado una herramienta para problemas de mínimos cuadrados no lineales llamada *lsqnonlin* del programa *MATLAB* sobre los 23 parámetros que definen $\mathbf{W}_2$,$\mathbf{W}_3$,$\mathbf{W}_4$,$\mathbf{b}_2$,$\mathbf{b}_3$,$\mathbf{b}_4$ (vease por ejemplo las referencias [26, Cap. 8], [13, Cap. 8],[2, Cap. 9] o [27, Cap. 7]).

Para la red ya entrenada con los diez datos y la función de activación sigmoide, en la Figura 7 se dibuja la frontera que separa los puntos en los que $F_1(x) > F_2(x)$. Este resultado indicaría que la zona sombreada, existe una mayor probabilidad de éxito en la perforación.

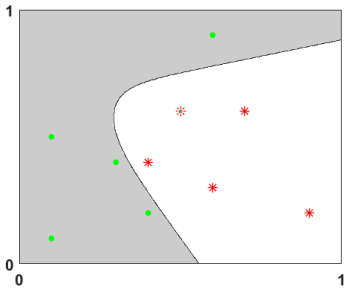

FIGURA 7. Resultado del entrenamiento.

2.5. Configuración general de una red. Para resolver problemas más sofisticado se necesitara un gran sistema que numerosas neurones en muchas capas.

En cada capa de una red neuronal artificial, toda neurona activa recibe los mismos datos de entrada (valores en $\mathbb{R}$) y produce un único dato de salida (otro valor en $\mathbb{R}$). Eso no implica que todas las neuronas de una red tengan las mismas funciones, y es por ello que las distinguimos en 3 categorías:

- **Neuronas de entrada.** Estas son las encargadas de recibir primeramente los datos de entrada. Utilizando los pesos y sesgos inviará a cada una de las neuronas de la siguiente capa, un valor de salida.
- **Neuronas ocultas.** Todas las neuronas intermedias entre la capa de entrada y la capa de salida de la red. Cada una recibe un valor de entrada por cada neurona (activa) que haya en la capa anterior, y después de realizar la operación que aparece en la ecuación, enviará el valor obtenido a todas las neuronas del siguiente nivel. Aquí es donde se realizan la mayoría de los cálculos.
- **Neuronas de salida.** Estas neuronas son las encargadas de las salidas que resulta de la red.

Si una la red consta de C capas con n_c neuronas para $c \in \{1, 2, \ldots, C\}$, n_1 será la dimensión de los datos de entrada y n_C la de salida. Entonces, es evidente que la neurona va de $\mathbb{R}^{n_1}$ en $\mathbb{R}^{n_C}$. Además, denotaremos por $\mathbf{W}_c \in \mathbb{R}^{n_c \times n_{c-1}}$ a la matriz de pesos en la capa c-ésima, por lo que w^c_{jk} es el peso que aplica la neurona j-ésima en la capa c-ésima a la salida de la neurona k-ésima en la capa anterior, es decir, la capa $c-1$-ésima. Además, $\mathbf{b}_c \in \mathbb{R}^{n_c}$ como el vector de sesgos para la capa c-ésima, por lo que la neurona j-ésima en la capa c-ésima utilizaré el valor b^c_j.

Dado un dato de entrada $\mathbf{x} \in \mathbb{R}^{n_1}$, podemos resumir la acción de la red denotando por a^c_j la activación de la salida de la neurona j-ésima en la capa c-ésima. Así, tendremos un nuevo vector $\mathbf{a}_c \in \mathbb{R}^{n_c}$ con los valores de activación a^c_j, $j \in \{1, 2, \ldots, n_c\}$, y por tanto:

$$\mathbf{a}_1 = \mathbf{x} \in \mathbb{R}^{n_1}, \tag{2.6}$$

$$\mathbf{a}_c = f(\mathbf{W}_c\, \mathbf{a}_{c-1} + \mathbf{b}_c),\ \forall c \in \{2, 3, \ldots, C\}. \tag{2.7}$$

Finalmente, si se supone N datos o puntos de entrenamiento en $\mathbb{R}^{n_1}$, es decir, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ tal que $\mathbf{x}_i \in \mathbb{R}^{n_1}$, $\forall i \in \{1, 2, \ldots, N\}$, para los cuales se conocen las salidas $\{\mathbf{y}(x_i)\}_{i=1}^N \in \mathbb{R}^{n_C}$. Entonces, la función de costos cuadrática, es decir, la función que se busca minimizar es:

$$Cost(\mathbf{W}_2, \mathbf{W}_3, \ldots, \mathbf{W}_C, \mathbf{b}_2, \mathbf{b}_3, \ldots, \mathbf{b}_C) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} ||\mathbf{y}(x_i) - \mathbf{a}_C(x_i)||_2^2. \tag{2.8}$$

En el ejemplo de la red que se muestra en la figura 8 se tiene que $C = 4$ y las dimensiones:

- $n_1 = 2$, $n_2 = 3$, $n_3 = 4$, $n_4 = 1$.
- $\mathbf{W}_2 \in \mathbb{R}^{3\times 2}$, $\mathbf{W}_3 \in \mathbb{R}^{4\times 3}$, $\mathbf{W}_4 \in \mathbb{R}^{1\times 4}$.
- $\mathbf{b}_2 \in \mathbb{R}^3$, $\mathbf{b}_3 \in \mathbb{R}^4$, $\mathbf{b}_4 \in \mathbb{R}$.

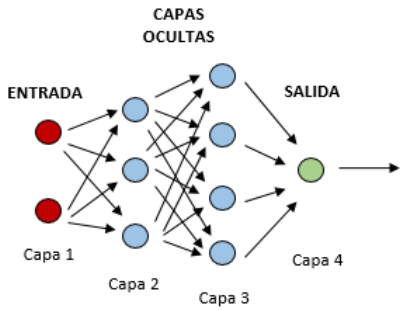

FIGURA 8. Estructura de una red neurona con cuatro capas.

En esta sección se ha explicado la estructura y configuración básica de las neuronas y de las redes neuronales y también mostrado como funcionan las redes neuronales prealimentadas. Después de estos conocimientos básicos se tratará de mejorar los entrenamientos con el fin de que aumentar su eficacia pensado en problemas de mayor complejidad y dimensión.

3. Entrenamiento de Redes Neuronales Prealimentadas

En la sección anterior se vio que para entrenar una red neuronal, se debe elegir los pesos y los sesgos que minimices la función costo (2.8). Estos pesos y sesgos que son matrices y vectores, en lo que sigue serán todos ellos un vector $\mathbf{p} \in \mathbb{R}^s$, en ejemplo de los sondéos $p = 23$. Por tanto se tiene un problema de minimización que se trata en esta sección. Ahora la función costo la se escribe de la forma

$$Cost(\mathbf{p}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} ||\mathbf{y}(x_i) - \mathbf{a}_C(x_i)||_2^2 = \frac{1}{N} \sum_{i=1}^{N} C_{x_i}(\mathbf{p}). \tag{3.1}$$

El método de optimización más conocido es el método del gradiente. Se trata de un método iterativo que calcula una sucesión de vectores en $\mathbb{R}^s$ que converge al vector que minimiza la función costo. Suponiendo que el vector actual es $\mathbf{p}$ se trata de elegir una perturbación $\triangle\mathbf{p}$ tal que $\mathbf{p}+\triangle\mathbf{p}$ séa una mejora, es decir, la función costo disminuya. Entonces si $\triangle\mathbf{p}$ es pequeño se puede ignora $||\triangle\mathbf{p}||^2$ y en el desarrollo de Taylor se tiene

(3.2)
$$Cost(\mathbf{p}+\triangle\mathbf{p}) \approx Cost(\mathbf{p}) + \sum_{r=1}^{s} \frac{\partial Cost(\mathbf{p})}{\partial p_r} \triangle p_r = Cost(\mathbf{p}) + \nabla Cost(\mathbf{p})^T \triangle\mathbf{p},$$

donde $\nabla Cost(\mathbf{p})$ es el gradiente de la función costo en $\mathbf{p}$. Ahora para reducir el valor de la función $Cost$, el producto $\nabla Cost(\mathbf{p})^T \triangle\mathbf{p}$ deberá ser tan negativo como séa posible y teniendo en cuanta la desigualdad de Cauchy-Schwarz que dice que para dos vectores $\mathbf{f},\mathbf{g}\in\mathbb{R}^s$ se tiene la desigualdad $|\mathbf{f}^T\mathbf{g}| \leq ||\mathbf{f}||_2||\mathbf{g}||_2$, es decir, lo más negativo que puede ser $\mathbf{f}^T\mathbf{g}$ es $-||\mathbf{f}||_2||\mathbf{g}||_2$ que ocurre cuando $\mathbf{f}=-\mathbf{g}$. Esto aplicado a (3) dice que la mejor elección es $\triangle\mathbf{p} = -\nabla Cost(\mathbf{p})$ que define el método de máximo descenso o método del gradiente, es decir

(3.3)
$$\mathbf{p} \rightarrow \mathbf{p} - \nabla Cost(\mathbf{p}),$$

y donde

(3.4)
$$\nabla Cost(\mathbf{p}) = \frac{1}{N}\sum_{i=1}^{N} \nabla C_{x_i}(\mathbf{p}).$$

Si ahora se considera que la función costo (3.1) tiene muchos parámetros, es decir, s es grande y además N, el número de datos, habitualmente es aún mayor, evaluar el gradiente en cada paso puede se muy caro y se necesita abaratar cada paso, lo que conduce a llamado Método del gradiente estocástico que se comenta brevemente en varias versiones.

3.1. Versión con muestreo con reemplazamiento. En cada paso se elige aleatoriamente un punto de entrenamiento para representar a todos, esto se puede resumirse de la siguiente manera:

1. Elegir un entero $i\in\{1,2,\ldots,N\}$ uniformemente al azar.
2. Actualizar

(3.5)
$$\mathbf{p}_{nuevo} = \mathbf{p}_{previo} - \alpha\ \nabla C_{x_i}(\mathbf{p}).$$

En este caso, el índice i-ésimo ha sido seleccionado y después reemplazado. el punto de entrenamiento. Observa se que incluso para pequeños valores de α, la actualización (3.5) no garantiza reducir la función de costo total.

3.2. Versión con muestreo sin reemplazamiento. La versión del método del gradiente estocástico en (3.5) es el más simple de un extenso arco de posibilidades. Una alternativa a este es hacer muestreo pero esta vez sin reemplazamiento, es decir, recorrer cada uno de los N puntos de entrenamiento en orden aleatorio. Realizar N pasos de esta manera, referido como completar una epoch, se puede resumir de la siguiente manera:

1. Mezclar los enteros $\{1,2,\ldots,N\}$ en un nuevo orden $\{k_1,k_2,\ldots,k_N\}$.
2. Desde $i=1$ hasta N, actualizar

(3.6)
$$\mathbf{p}_{nuevo} = \mathbf{p}_{previo} - \alpha\nabla C_{x_{k_i}}(\mathbf{p}).$$

3.3. Versión mini-batch. El método del gradiente estocástico aproximaba la media sobre todos los puntos de entrenamiento en (3.4). Entonces, parece lógico tomar una parte de la muestra, es decir, para algún $m << N$:

1. Seleccionar m enteros $\{k_1, k_2, \ldots, k_m\} \subset \{1, 2, \ldots, N\}$ uniformemente al azar.
2. Actualizar

$$\mathbf{p}_{nuevo} = \mathbf{p}_{previo} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x_{k_i}}(\mathbf{p}). \tag{3.7}$$

Para esta variante el subconjunto $\{x_{k_1}, x_{k_2}, \ldots, x_{k_m}\}$ se conoce como mini-batch.

Para una información más ámplia de estas técnicas se recomienda la referencia [14].

3.4. Propagación hacia atrás. Para utilizar el método del gradiente estocástico en sus distintas versiones, se necesita calcular las derivadas parciales con respecto a cada la componentes del vector $\mathbf{p} \in \mathbb{R}^s$ que son las w^c_{jk} y b^c_j, y la forma de conseguir estas estimaciones es utilizando el método de pro para propagación hacia atrás más conocido como back propagation que se describe a continuación.

Se considera la función C_{x_i} en (3.1), para un punto de entrenamiento ya fijado, es una función de pesos y sesgos, por lo que podemos olvidar la dependencia sobre el punto x_i (puesto que está ya fijado) y solamente quedarnos con

$$Cost = \frac{1}{2} ||\mathbf{y} - \mathbf{a}^{[C]}||_2^2, \tag{3.8}$$

donde se sabe que $\mathbf{a}^{[C]}$ es la salida de la red neuronal artificial y es por ello por lo que $Cost$ depende de los pesos y sesgos.

Para facilitar trabajo y posteriormente conseguir expresiones más cómodas, se definen dos nuevas variables. En primer lugar, sea

$$\mathbf{z}^{[c]} = \mathbf{W}^{[c]} \mathbf{a}^{[c-1]} + \mathbf{b}^{[c]} \in \mathbb{R}^{n_c}, \quad \text{para} \quad c = 2, 3, \ldots, C, \tag{3.9}$$

donde $z^{[c]}_j$ representa la entrada ponderada para la neurona j-ésima en la capa c-ésima. Entonces, la relación definida en (2.7) que propaga la información a través de la red se puede escribir como

$$\mathbf{a}^{[c]} = f(\mathbf{z}^{[c]}), \quad \text{para} \quad c = 2, 3, \ldots, C. \tag{3.10}$$

La segunda variable será $\delta^{[c]} \in \mathbb{R}^{n_c}$ de la forma

$$\delta^{[c]}_j = \frac{\partial Cost}{\partial z^{[c]}_j}, \quad \text{para} \quad j = 1, 2, \ldots, n_c, \quad c = 2, 3, \ldots, C, \tag{3.11}$$

esta expresión es comunmente denominada como el error de la j-ésima neurona en la capa c-ésima porque muestra la variación de la función de coste C con respecto a la entrada ponderada de la neurona j-ésima en la capa c-ésima.

En este punto se necesita definir el producto Hadamard de dos vectores.

Definicion 3.1. El producto Hadamard de dos vectores $\mathbf{x}, \mathbf{y} \in \mathbb{R}^s$ es otro vector tal que $(\mathbf{x} \circ \mathbf{y})_i = x_i y_i$ para $i = 1, \cdots, s$.

Con esta notación, el siguiente resultados permite calcula los gradiente con el algoritmo de propagación hacia atrás. En la demostración se aprecia que son consecuencia de la regla de la cadena.

Lema 3.2.

$$\delta^{[C]} = f'(\mathbf{z}^{[C]}) \circ (\mathbf{a}^{[C]} - \mathbf{y}). \tag{3.12}$$

$$\delta^{[c]} = f'(\mathbf{z}^{[c]}) \circ ((\mathbf{W}^{[c+1]})^T)\delta^{[c+1]} \quad para \quad c = 2, 3, \ldots, C-1. \tag{3.13}$$

$$\frac{\delta Cost}{\delta b_j^{[c]}} = \delta_j^{[c]}, \quad para \quad c = 2, 3, \ldots, C. \tag{3.14}$$

$$\frac{\delta Cost}{\delta w_{jk}^{[c]}} = \delta_j^{[c]}\, a_k^{[c-1]}, \quad para \quad c = 2, 3, \ldots, C. \tag{3.15}$$

Demostración. Ver [12]. □

4. Algunas aplicaciones del Depp Learning

Esta técnica DL ha tenido y sigue teniendo mucho éxito en aplicaciones y disciplinas científica, quizá la más conocida es la de clasificación de imágenes que se explica en la segunda parte de [12]. Otras muchas dichas aplicaciones se pueden consultar en recientes publicaciones como por ejemplo [10],[17], [20], [5], [1] y [11]. En MATLAB existe una Toolbox que se puede consultar en la página de MathWorks, en este documento se comentarán algunas investigaciones que han usado Deep Learning para resolver ecuaciones diferenciales.

4.1. Sparse Identification of Nonlinear Dynamics (SINDy). [4] y [9]. Uno de los mayores retos actuales en muchas áreas ciencíticas y técnicas es extraer las ecuaciones matemáticos de las grandes bancos de datos. Lo que ahora se conoce cómo técnicas Big Data, almacena grandes cantidades de datos que adémas de darles extructúra también se preocupa de relacionar lo con modelos matemáticos para descubrir nuevas aplicaciones.

4.2. Physics-informed neural networks (PINN). En [21] y [18] se utiliza deep-learning para resolver problemas directos e inversos también de ecuaciones diferenciales no lineales.

Estas técnicas nuevas son de una eficacia sorprendete y hace imposible predecir su futuro: hasta donde llegarán y el tipo de problemas que podrán resolver.

Referencias

1. G.A. Anastassiou, *Parametrized, Deformed and General Neural Networks*, Springer, 2023.
2. U.M. Ascher and C. Greif, *A Firts Course in Numerical Methods*, SIAM, 2011.
3. M.A. Boden, *Inteligencia artificial*, Turner, 2017.
4. S.L. Brunton, J.L. Proctor, and J.N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, PNAS **113** (2016), no. 15.
5. N. Buduma, *Fundamentals of Deep Learning. Designing Next-Generation Machine Intelligence Algorithms*, O'REILLY, 2017.
6. R. Caballero and E. Martín, *Bases de big data y de la inteligencia artificial*, Los libros de la catarata, 2022.
7. M. Coeckelbegh, *Ética de la Inteligencia artificial*, Cátedra, 2021.

8. V. Demonte, *Entre la exaltación y el miedo: ¿ la IA nos quitará la palabra ?*, Revista de libros **Febrero** (2024).
9. U. Fasel, J.N. Kutz B.W. Brunton, and S.L. Bruton, *Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control*, The Royal Sociaty Publishing **478** (2022).
10. I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press, 2016.
11. P. Grohs and G. Kutyniok, *Mathemticl Aspects of Deep Learning*, Cambridge University Press, 2023.
12. C.F. Higham and D.J. Higham, *Deep learning: An Introduction for Applied Mathematicians*, SIAM Review **61** (2019), 860–891.
13. M.H. Holmes, *Introduction to Scientific Computing and Data Analysis*, Springer, 2016.
14. k. Jin, J. Latz, C. Liu, and C.B. Schonlied, *A Continuous-time Stochastic Gradient Descent Method for Continuous Data*, Journal od Marching Learning Research (2023), 1–48.
15. J. Kaplan, *Artificial Intelligence*, Oxfort University Press, 2023.
16. K. Kelly, *The Inevitable*, Penguin Publishing Group, 2017.
17. P. Kim, *MATLAB Deep Learning*, Apress, 2017.
18. L. Lu, X. Meng, A.Z. Mao, and G.E. Karniadakis, *DeepXDE: A Deep Learning Library for Solving Differential Equations*, SIAM Review **63** (2021), 208–228.
19. C. O'Neail, *Armas de destrucción matemática*, Capitán Swing, 2018.
20. J. Patterson and A. Gibson, *Depp Learing: A Practitioner's Approach*, O'REILLY, 2017.
21. M Raissi, P. Perdikaris, and G.E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal od Computational Physics **378** (2021), 686.
22. P. Rodriguez, *Inteligencia artificial*, Deusto, 2018.
23. L. Rouhiainen, *Artificial Intelligence: 101 Things You Must Know Today About Our Future*, 2018.
24. S. Russel and P. Norvig, *Inteligencia artificial: Un enfoque moderno*, Alhambra, 2004.
25. V.M. Schonberger, *Big data : la revolución de los datos masivos*, Turner, 2013.
26. D.E. Stewart, *Applied Numerical Analysis: A Graduate Course*, Springer, 2023.
27. W.Y. Yang, W. Cao, T.S. Chung, and J. Morris, *Applied Numerical Methods Using* MATLAB, Wiley Interscience, 2005.

Dep. Matemática de la UPV/EHU
Email address: fernando.vadillo@ehu.es