INTRODUCCIÓN A LA SOLUCIÓN DE ECUACIONES DIFERENCIALES USANDO REDES NEURONALES

Miguel Uh Zapata
Reymundo Itzá Balam

Índice general

CAPÍTULO 2

INTRODUCCIÓN A LAS REDES NEURONALES

En este capítulo introductorio se presentan los conceptos fundamentales comúnmente utilizados en el estudio de las redes neuronales artificiales. Comenzaremos con una discusión general sobre qué es una red neuronal artificial, haciendo énfasis en su forma más simple: *el perceptrón.*

A partir de esta base, se introducirán los elementos clave involucrados en la construcción y resolución de problemas mediante esta técnica, tales como la arquitectura de la red, el proceso de entrenamiento y los mecanismos de ajuste de parámetros. Finalmente, se revisarán algunos ejemplos ilustrativos que permitirán contextualizar y afianzar los conceptos presentados.

OBJETIVOS

Al finalizar este capítulo, el lector será capaz de:

- Definir qué es una red neuronal artificial y describir sus componentes fundamentales.
- Explicar el funcionamiento y la estructura del perceptrón como el modelo más simple de una red neuronal.
- Identificar los conceptos básicos involucrados en la resolución de problemas mediante redes neuronales, tales como arquitectura, entrenamiento y ajuste de parámetros.
- Analizar ejemplos sencillos de aplicación de redes neuronales con el fin de reforzar los conceptos teóricos presentados.

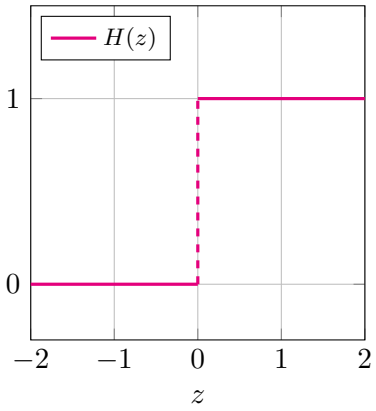


Figura 2.37 Función escalón usado como función de activación en el perceptrón.

2.2 EL PERCEPTRÓN

Desde una perspectiva matemática, una red neuronal puede interpretarse como la composición de diversas funciones matemáticas que, actuando de manera conjunta, permiten modelar y aproximar relaciones de elevada complejidad. Para comprender con mayor profundidad este concepto, resulta natural comenzar con la arquitectura más sencilla.

El *perceptrón*, introducido en 1957 por Frank Rosenblatt [**?**], representa una de las estructuras fundamentales dentro de las redes neuronales. En este modelo, tanto las entradas como la salida se expresan mediante valores numéricos, y cada conexión entre neuronas se encuentra asociada a un peso que determina la influencia de la información transmitida.

DEFINICIÓN

Sean $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ y $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)^T \in \mathbb{R}^n$ los vectores de entradas y pesos, respectivamente. El *perceptrón* es la función $\widetilde{u} : \mathbb{R}^n \to \mathbb{R}$ calculada como $\widetilde{u}(\boldsymbol{x}; \boldsymbol{w}) = H(z)$ donde

$$z = \boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n, \tag{2.1}$$

es la suma ponderada de sus entradas y

$$H(z) = \begin{cases} 1, & \text{si } z \geq 0, \\ 0, & \text{si } z < 0, \end{cases} \tag{2.2}$$

es la función escalón, denominada *función de activación*.

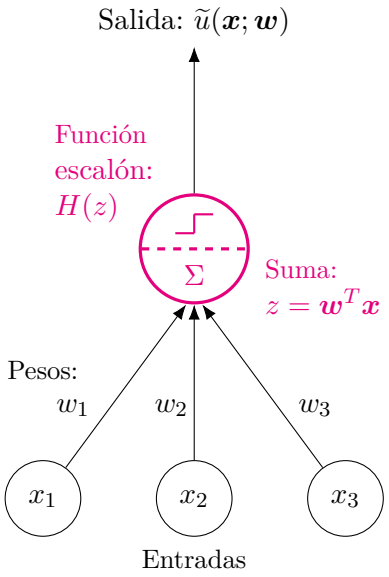


Figura 2.38 Esquema gráfico de las principales partes de un perceptrón con tres valores de entrada.

Una forma habitual de representar las redes neuronales es mediante esquemas gráficos, ya que estos permiten visualizar de manera intuitiva tanto la estructura del modelo como el flujo de información entre sus componentes. Si bien dichas representaciones pueden variar ligeramente según el contexto o la notación adoptada, en la Figura 2.38 se muestra una versión del modelo original propuesto por Rosenblatt, correspondiente a un perceptrón con tres entradas. El flujo de información en esta representación se considera de abajo hacia arriba. En el diagrama, el símbolo (Σ) denota la operación de suma ponderada que define la variable z, tal como se establece en la ecuación (2.1). Posteriormente, el símbolo ($\sqcap$) representa la aplicación de la función escalón dado en la ecuación (2.2), la cual determina la salida del modelo.

Originalmente, el perceptrón fue diseñado para resolver problemas de clasificación, lo que justifica el uso de una función escalón que genera salidas binarias. Sin embargo, la evolución de las redes neuronales ha permitido generalizar la arquitectura de la red y ha ampliado notablemente las capacidades del modelo, posibilitando su aplicación a problemas más complejos.

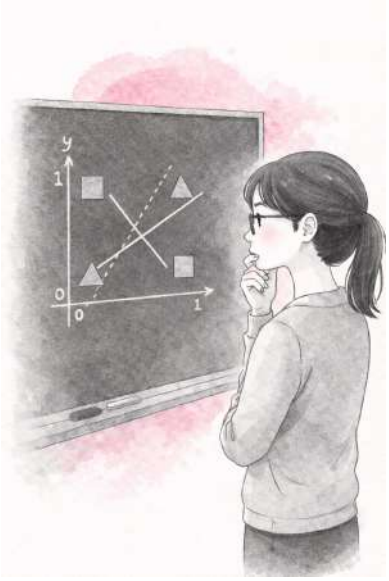


Figura 2.41 Representación geométrica del problema XOR. No existe una recta que separe correctamente las clases.

2.2.2 Ejemplo: el problema de clasificación XOR

Iniciaremos con uno de los ejemplos más representativos en redes neuronales y aprendizaje automático: el problema XOR (o exclusivo), que se plantea a continuación.

EL PROBLEMA XOR

Considérese un conjunto de datos compuesto por pares ordenados (x_1, x_2), donde $x_1, x_2 \in \{0, 1\}$. Se desea construir un modelo de clasificación binaria cuya salida $y \in \{0, 1\}$ satisfaga la siguiente regla:

- $y = 1$ cuando las entradas son distintas ($x_1 \neq x_2$),
- $y = 0$ cuando las entradas son iguales ($x_1 = x_2$).

El objetivo es determinar si un perceptrón simple es capaz de representar esta función de clasificación.

En 1969, Marvin Minsky y Seymour Papert, en su monografía *Perceptrons*, señalaron una limitación fundamental del perceptrón simple: su incapacidad para resolver problemas aparentemente elementales, como el problema XOR.

Desde una perspectiva geométrica, el problema XOR puede visualizarse en el plano bidimensional considerando cada patrón de entrada como un punto. En el caso de la función XOR, los patrones asociados a la clase 0, $(0, 0)$ y $(1, 1)$, se ubican en vértices opuestos del cuadrado unitario, mientras que los patrones de la clase 1, $(0, 1)$ y $(1, 0)$, ocupan los vértices restantes. La Figura 2.41 muestra este problema de clasificación usando cuadrados y triángulos.

Podemos ver que la disposición de las clases genera una configuración en la que no existe una única recta capaz de separar correctamente ambas clases. En otras palabras, las clases no son linealmente separables, lo que explica la incapacidad de un perceptrón simple, cuya capacidad de decisión es lineal, para representar la función XOR.

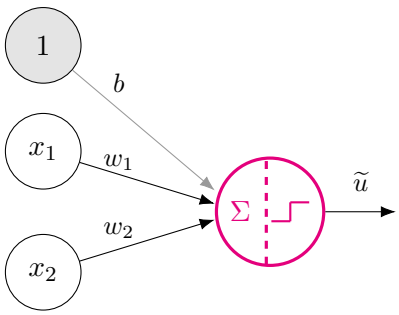


Figura 2.42 Perceptrón usado en el problema XOR.

Para establecer una perspectiva matemática más rigurosa que el perceptrón simple no puede representar el problema XOR, recurriremos a un argumento por contradicción. Consideremos un perceptrón con dos variables de entrada dada por

$$\widetilde{u}(x_1, x_2; w_1, w_2, b) = H(z), \qquad \text{donde} \qquad z = w_1 x_1 + w_2 x_2 + b,$$

y cuya representación se muestra en la Figura 2.42. La demostración inicia suponiendo que este perceptrón resuelve el problema XOR. Bajo esta hipótesis, existirían los parámetros w_1, w_2 y b tales que el modelo satisface la Tabla 2.1.

Figura 2.112 Entorno en la nube que permite ejecutar códigos en `Python` directamente desde el navegador.

2.9.1 Ejecución en la nube: Google Colab

Para realizar las implementaciones y experimentar con los ejemplos presentados en este libro, el lector necesita contar con un editor o entorno de desarrollo para `Python`. Si se tiene poca experiencia en la instalación y configuración de software, se recomienda utilizar un editor de código en la nube.

Los entornos en la nube ofrecen dos ventajas fundamentales: textcolorpinkMXno es necesario instalar programas en el equipo local y permiten compartir fácilmente el trabajo a través de internet. Entre las opciones disponibles, destacan los servicios gratuitos proporcionados por Google. En particular, *Google Colab* es un entorno de desarrollo integrado (IDE) en la nube que permite ejecutar código `Python` directamente desde el navegador.

Una de sus principales ventajas es que incluye muchas librerías preinstaladas, entre ellas `PyTorch` y `TensorFlow`, lo que facilita comenzar a trabajar de inmediato sin preocuparse por dependencias o configuraciones adicionales. Por esta razón, se recomienda especialmente a los usuarios que no estén familiarizados con la instalación y administración de entornos de desarrollo locales.

Además, existen tutoriales introductorios y ejemplos de uso de fácil acceso que permiten familiarizarse rápidamente con la plataforma y comenzar a desarrollar código de manera práctica. Un ejemplo de la interfaz de programación en Google Colab se muestra en la Figura 2.113, donde puede apreciarse la estructura típica del entorno: el editor de celdas de código, las herramientas de ejecución y los paneles de gestión de archivos y recursos.

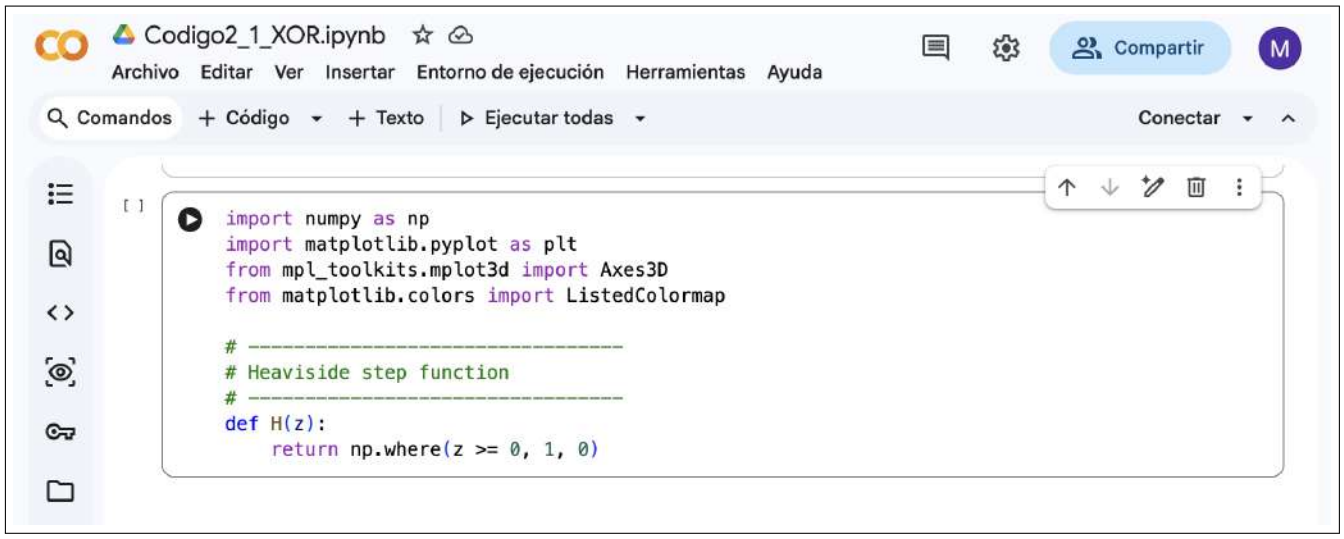


Figura 2.113 Un ejemplo de la interfaz de programación en Google Colab (2026).

Ejercicio 2.3: Visualización del problema de paridad de tres bits. Siguiendo con el ejercicio anterior, ahora nuestro objetivo es desarrollar un programa en `Python` que permita visualizar geométricamente en la computadora que los puntos correspondientes a la función de paridad no pueden serpararse linealmente mediante un solo hiperplano.

Además, este ejercicio servirá como una introducción práctica al uso de `Python` y del entorno de trabajo en Google Colab. A través de su implementación, el lector podrá familiarizarse la ejecución de código en un entorno interactivo en la nube.

El código debe incluir lo siguiente.

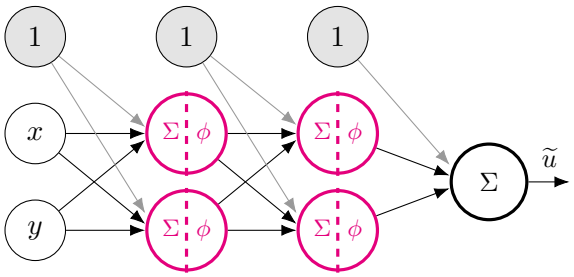
- Generar los ocho puntos del conjunto $\{0,1\}^3$ y evaluar la función de paridad f en cada uno de los ocho puntos.
- Construir una gráfica tridimensional donde:
  - Los puntos con salida $f = 1$ se representen con un color.
  - Los puntos con salida $f = 0$ se representen con otro color.
- Intentar añadir un plano (correspondiente a un perceptrón)

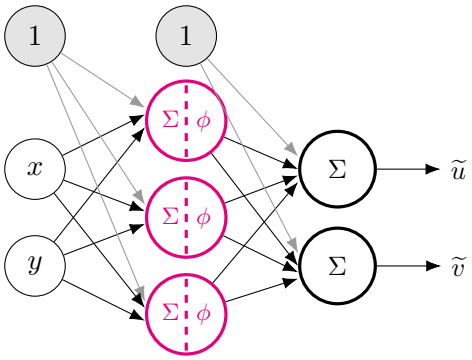
  $$w_1x_1 + w_2x_2 + w_3x_3 + b = 0,$$

  y observar que no es posible separar correctamente ambas clases con un único plano.

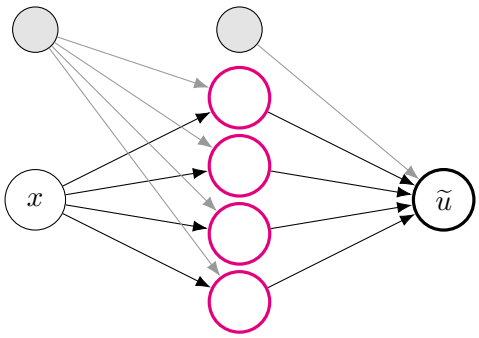
Finalmente, elaborar un breve reporte en el que se describan los principales componentes del programa desarrollado e incluir los resultados gráficos obtenidos del mismo.

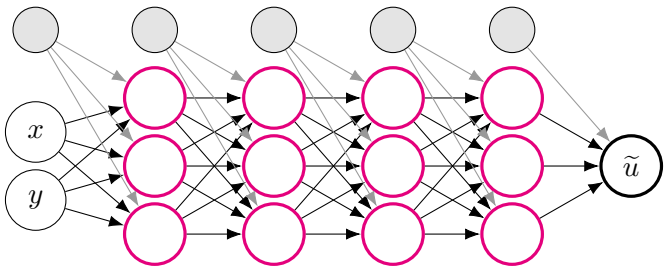
Ejercicio 2.4: Expresión matemática de una red neuronal. Escribir de manera explícita la expresión matemática de las siguientes redes neuronales en términos de sus pesos y sesgos, donde ϕ es la función tangente hiperbólica.





Ejercicio 2.5: Expresión matemática de redes neuronales más complejas. Escribir de manera explícita la expresión matemática de las siguientes redes neuronales en términos de sus pesos y sesgos, donde ϕ es la función de activación general. Ahora usaremos la notación más simplista que estaremos manejando en los siguientes capítulos.





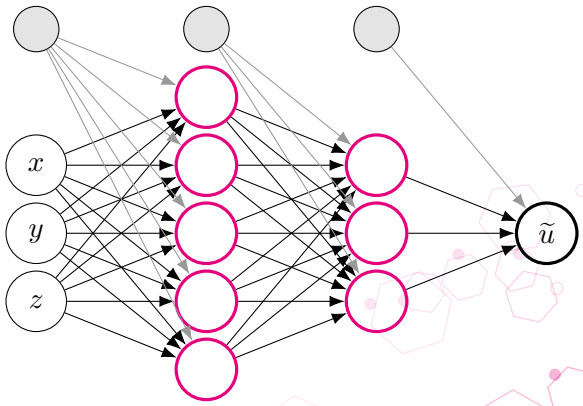


Figura 5.232 Las placas metálicas pueden presentar diferentes formas geométricas, comenzando con configuraciones simples como dominios rectangulares.

5.2 ECUACIONES ELÍPTICAS EN 2D

Con el propósito de ilustrar la implementación práctica de la metodología basada en redes neuronales y evaluar su desempeño en la aproximación de soluciones, consideraremos una clase de ecuaciones elípticas en dos dimensiones descritas a continuación.

ECUACIÓN DIFERENCIAL ELÍPTICA EN 2D

Consideremos ecuaciones elípticas de segundo orden en dos dimensiones de la forma

$$\nabla \cdot (\beta(\boldsymbol{x})\nabla u(\boldsymbol{x})) + \kappa(\boldsymbol{x})u(\boldsymbol{x}) = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega \subset \mathbb{R}^2, \quad (5.10)$$

donde $\boldsymbol{x} = (x, y) \in \Omega$, $u = u(x, y)$ es la función desconocida que se desea aproximar, $\beta(x, y)$ representa un coeficiente de difusión o conductividad espacialmente variable, $\kappa(x, y)$ es un término de reacción o absorción, y $g(x, y)$ corresponde al término fuente.

Supondremos que las funciones β, κ y g son conocidas y suficientemente suaves, de manera que el problema se encuentre bien planteado desde el punto de vista analítico. En particular, para garantizar el carácter elíptico del operador diferencial, se asume que $\beta(x, y) > 0$ en todo punto del dominio Ω.

La ecuación (5.10) aparece en una gran variedad de contextos físicos e ingenieriles, como la conducción de calor en estado estacionario, procesos de difusión, electrostática y mecánica de medios continuos, entre otros. En este sentido, la solución buscada $u(x, y)$ describe típicamente una magnitud física en equilibrio, cuya distribución espacial está determinada tanto por las propiedades del medio como por las condiciones impuestas en la frontera del dominio.

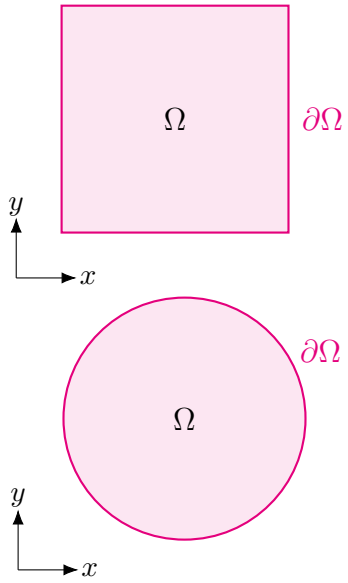


Figura 5.233 Ejemplos de dominios con geometría regular empleados para el problema elíptico en $\mathbb{R}^2$.

5.2.1 Condiciones de frontera

Para que el problema quede completamente determinado, no basta con especificar únicamente la ecuación diferencial en el interior del dominio; también es necesario imponer información adicional sobre lo que ocurre en su frontera. A esta información se le conoce como *condiciones de frontera*, y su papel es restringir el conjunto de soluciones posibles de acuerdo con el comportamiento físico que se desea modelar.

Denotemos por $\partial\Omega$ la frontera del dominio Ω. En aplicaciones físicas, esta frontera puede representar las paredes de una placa, el contorno de un material o la superficie externa de una región donde ocurre un fenómeno de difusión o transferencia. Dependiendo del problema considerado, la frontera $\partial\Omega$ puede dividirse en distintas porciones, sobre cada una de las cuales se prescribe un tipo específico de condición. A continuación se presentan sus expresiones matemáticas más generales.

Figura 5.244 A diferencia de los casos previamente estudiados, los problemas bidimensionales suelen requerir arquitecturas neuronales más profundas para obtener mejores aproximaciones.

5.4.1 El modelo basado en redes neuronales

El punto de partida de esta metodología consiste en asumir que todos los términos que definen la EDP son conocidos, mientras que la solución exacta u permanece desconocida y debe ser aproximada.

EL MODELO BASADO EN REDES

Se introduce una red neuronal multicapa como una función aproximante de la forma

$$u(x, y) \approx \widetilde{u}(x, y; W, \boldsymbol{b}), \tag{5.18}$$

donde $\widetilde{u}(x, y; W, \boldsymbol{b})$ representa la salida de la red neuronal evaluada en el punto (x, y), y W y $\boldsymbol{b}$ denotan, respectivamente, el conjunto de pesos y sesgos (*biases*) entrenables del modelo. El objetivo será ajustar dichos parámetros de tal manera que $\widetilde{u}$ satisfaga, en la medida de lo posible, tanto la EDP en el interior del dominio como las condiciones de frontera asociadas.

Aunque en los capítulos anteriores bastaba con emplear redes neuronales relativamente sencillas, compuestas por una o dos capas ocultas, en el caso bidimensional resulta conveniente utilizar arquitecturas más profundas para lograr una mejor capacidad de aproximación. La red neuronal multicapa propuesta consiste de una capa de entrada, una capa de salida y un conjunto de capas ocultas intermedias. Cada capa oculta contiene N_e neuronas, como se ilustra en la Figura 5.245. La primera capa siempre recibe como entrada el punto espacial $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$, mientras que la capa de salida produce el valor aproximado $\widetilde{u}(\boldsymbol{x}) \in \mathbb{R}$.

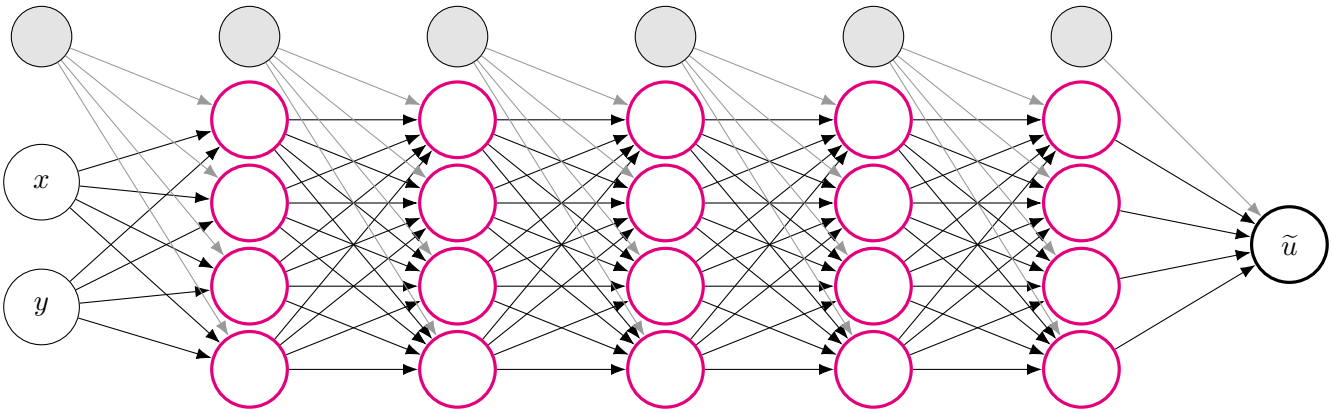


Figura 5.245 Ejemplo de una red neuronal con cinco capas ocultas de cuatro neuronas cada una.

En el ejemplo ilustrado en la Figura 5.245, la red neuronal tiene la configuración

$$\text{capas} = (2, 4, 4, 4, 4, 4, 1),$$

lo que significa que la red recibe dos entradas espaciales, contiene cinco capas ocultas con cuatro neuronas cada una y una única salida escalar.

Figura 5.238 El programa del método de diferencias finitas consiste de un sistema de ecuaciones lineales compuesta por el esquema numérico aplicado a cada punto del dominio.

Para mayores detalles sobre esta metodología, puede consultarse el libro de Burden [**?**]. A continuación, se presenta la implementación de este método para la aproximación numérica del problema, considerando una malla con $N = 20$ divisiones, lo que corresponde a un tamaño de paso de $h = 0.1$.

```python
import numpy as np
import matplotlib.pyplot as plt

#============================================================
# Solucion exacta y termino fuente
def solucion_exacta(x, y):
    return np.sin(np.pi*x) * np.sin(np.pi*y)
def termino_fuente(x, y):
    return -2.0* (np.pi**2) * np.sin(np.pi*x) * np.sin(np.pi*y)

#============================================================
# Solución de Poisson 2D con condiciones de Dirichlet
def poisson2D(F, G, h):
    N = F.shape[0]
    Ntot = N * N
    A = np.zeros((Ntot, Ntot))
    b = np.zeros(Ntot)
    for i in range(N):
        for j in range(N):
            k = i + j * N
            # Puntos en la frontera
            if i == 0 or i == N-1 or j == 0 or j == N-1:
                A[k, k] = 1.0
                b[k] = G[i, j]
            # Puntos interiores
            else:
                A[k, k]     = -4.0 / h**2
                A[k, k - 1] =  1.0 / h**2
                A[k, k + 1] =  1.0 / h**2
                A[k, k - N] =  1.0 / h**2
                A[k, k + N] =  1.0 / h**2
                b[k] = F[i, j]
    # Solucionador del sistema lineal
    uvec = np.linalg.solve(A, b)
    U = uvec.reshape((N, N), order='F') # Solución en 2D
    return U

#============================================================
# Programa principal

# Dominio
xI, xF, yI, yF = -1.0, 1.0, -1.0, 1.0

# Malla uniforme
N = 20
x = np.linspace(xI, xF, N+1)
y = np.linspace(yI, yF, N+1)
h = x[1] - x[0]
X, Y = np.meshgrid(x, y, indexing='ij')

# Solución exacta y término fuente
Uexa = solucion_exacta(X, Y)
F = termino_fuente(X, Y)

# Solución numérica
U = poisson2D(F, Uexa, h)
```

Código 5.4 Programa usando diferencias finitas para solucionar la ecuación de Poisson en 2D.

Notas

```
E = np.abs(Uexa-U)
s=np.sum(E**2)*h**2
L2 = np.sqrt(s)
LI = np.max(E)
print(f"Linf=LI:.6e")
print(f"L2=L2:.6e")
```

Notas

```
plt.figure()
plt.contourf(x,y,U.T)
plt.colorbar()
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Figura 5.239 Cálculo de las normas del error y gráficas del programa de diferencias finitas.

Notas

$N_D = 3N$
$N_{\mathcal{N}} = N$
$N_f = N^2$
$N_p = (N+1)^2$

Figura 5.312 Parámetros.

Los resultados numéricos

La Figura 5.313 muestra los resultados obtenidos utilizando $N_D = 3N$ puntos de entrenamiento para las condiciones de Dirichlet y $N_{\mathcal{N}} = N$ puntos asociados a las condiciones de Neumann, considerando los casos $N = 5$, 10 y 20. En todas las simulaciones se utilizaron además $N_f = N^2$ puntos de colocación dentro del dominio y una red neuronal multicapa con 4 capas ocultas y 20 neuronas por capa. Podemos observar que la red neuronal reproduce correctamente la solución exacta incluso cuando el número de datos es relativamente pequeño. Sin embargo, aunque el número total de puntos de entrenamiento y colocación aumenta considerablemente entre los distintos casos, el orden de magnitud del error absoluto permanece prácticamente similar.

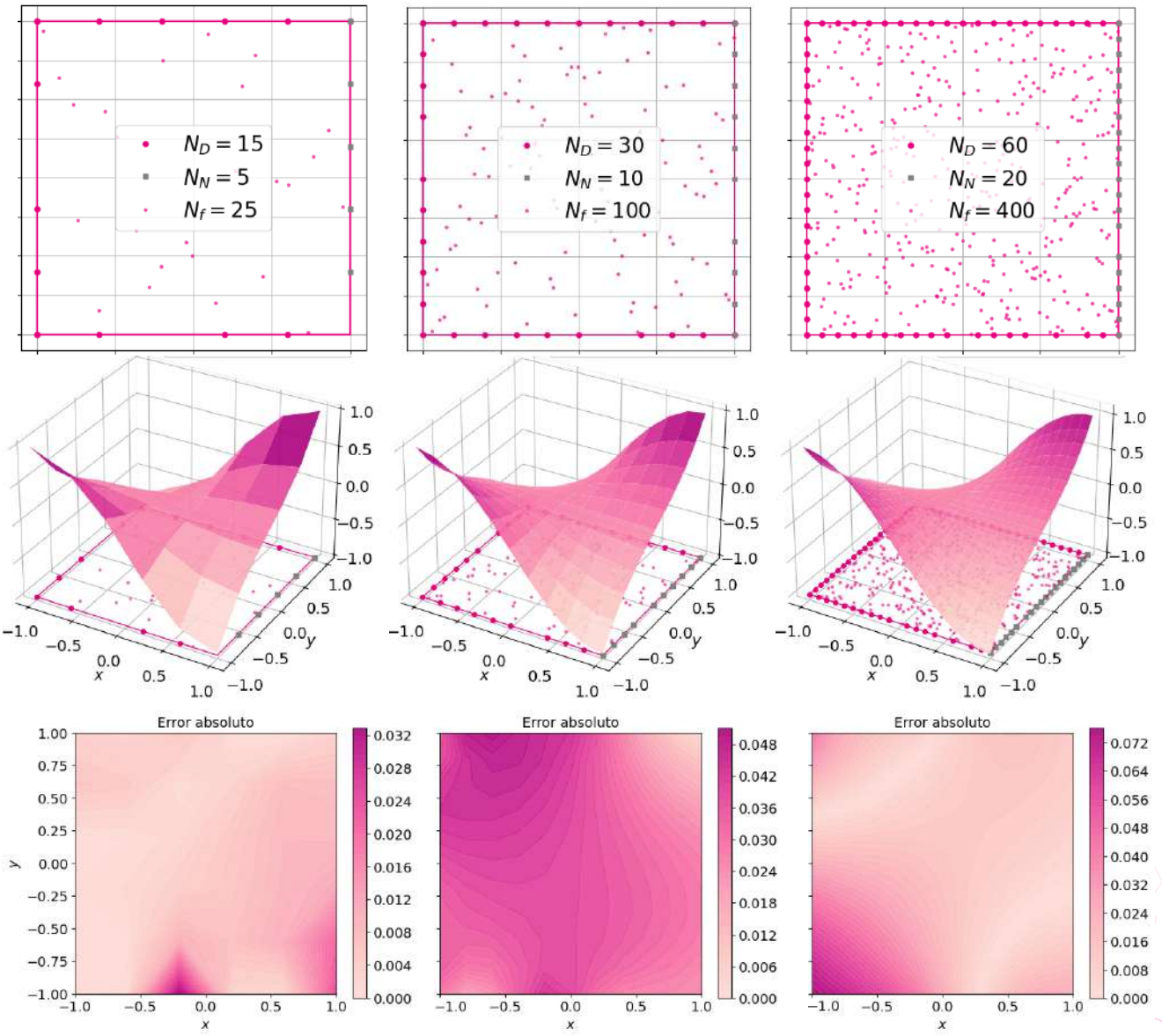


Figura 5.313 Muestra de puntos, solución aproximada y error absoluto de la red neuronal multicapa para distintos número de puntos de entrenamiento y colocación elegidos aleatoriamente.

Miguel Uh Zapata
Reymundo Itzá Balam