Ejercicios de Cálculo UD5

Ejercicio 1 (Demostrar las fórmulas utilizadas para el gradiente cuando se usa la función sigmoide (1 punto).)**.**

$$\frac{\partial e}{\partial w_i} = (\hat{y} - y)\,\hat{y}\,(1 - \hat{y})\,x_i$$

$$\frac{\partial e}{\partial b} = (\hat{y} - y)\,\hat{y}\,(1 - \hat{y})$$

Solución

1.1 Definamos primero cada elemento

1.1.1 Modelo lineal

$$z = \sum_{i=1}^{n} w_i\,x_i + b$$

1.1.2 Función sigmoide

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

1.1.3 Función de error

$$e(\hat{y}) = \frac{1}{2}\,(\hat{y} - y)^2$$

1.2 Primeras derivadas

1.2.1 Derivada del modelo de datos

$$\frac{\partial z}{\partial w_i} = \lim_{h \to 0} \frac{z(w_i + h) - z(w_i)}{h}$$

Cada término para calcular el límite es:

$$z(w_i + h) = \sum_{j \neq i} w_j x_j + (w_i + h)x_i + b$$

$$z(w_i) = \sum_{j \neq i} w_j x_j + w_i x_i + b$$

Con lo que la expresión completa es:

$$\frac{z(w_i+h)-z(w_i)}{h}=\frac{\left[\sum_{j\neq i}w_jx_j+(w_i+h)x_i+b\right]-\left[\sum_{j\neq i}w_jx_j+w_ix_i+b\right]}{h}$$

Simplificando:

$$\frac{hx_i}{h}=x_i$$

Ahora el límite es:

$$\lim_{h\to 0}x_i=x_i$$

Por lo que:

$$\frac{\partial z}{\partial w_i}=x_i.$$

1.2.2 Derivada de la función sigmoide

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

Definición de derivada:

$$f'(z)=\lim_{h\to 0}\frac{f(z+h)-f(z)}{h}$$

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

$$f'(z)=\lim_{h\to 0}\frac{\frac{1}{1+e^{-(z+h)}}-\frac{1}{1+e^{-z}}}{h}$$

$$f'(z)=\lim_{h\to 0}\frac{(1+e^{-z})-(1+e^{-(z+h)})}{h(1+e^{-(z+h)})(1+e^{-z})}$$

$$f'(z)=\lim_{h\to 0}\frac{e^{-z}-e^{-(z+h)}}{h(1+e^{-(z+h)})(1+e^{-z})}$$

Por tanto:

$$f'(z)=\frac{e^{-z}}{(1+e^{-z})^2}=\hat{y}(1-\hat{y})$$

$$\frac{\partial\hat{y}}{\partial z}=\hat{y}\,(1-\hat{y})$$

1.2.3 Derivada de la función de error

$$e(\hat{y}) = \frac{1}{2}(\hat{y} - y)^2$$

Definición de derivada:

$$\frac{\partial e}{\partial \hat{y}} = \lim_{h \to 0} \frac{\frac{1}{2}(\hat{y} + h - y)^2 - \frac{1}{2}(\hat{y} - y)^2}{h}$$

$$= \frac{1}{2} \lim_{h \to 0} \frac{(\hat{y} - y + h)^2 - (\hat{y} - y)^2}{h}$$

Expansión del cuadrado:

$$(\hat{y} - y + h)^2 = (\hat{y} - y)^2 + 2h(\hat{y} - y) + h^2$$

$$\frac{\partial e}{\partial \hat{y}} = \frac{1}{2} \lim_{h \to 0} \frac{2h(\hat{y} - y) + h^2}{h}$$

Simplificando:

$$\frac{\partial e}{\partial \hat{y}} = \frac{1}{2} \lim_{h \to 0} [2(\hat{y} - y) + h]$$

$$\frac{\partial e}{\partial \hat{y}} = \frac{1}{2} \cdot 2(\hat{y} - y) = \hat{y} - y$$

Ahora podemos aplicar la regla de la cadena, y las derivadas que nos quedan son:

$$\frac{\partial z}{\partial w_i} = x_i$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$$

y

$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y$$

Sabemos que se trata de una composicion de funciones:

$$e = e(\hat{y}(z(w_i)))$$

Por la regla de la cadena:

$$\frac{\partial e}{\partial w_i} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_i}$$

Sustituyendo :

$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial z}{\partial w_i} = x_i$$

Entonces:

$$\boldsymbol{\frac{\partial e}{\partial w_i} = (\hat{y} - y)\, \hat{y}(1 - \hat{y})\, x_i}$$

Para b:

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Pero, hay que tener en cuenta que:

$$\frac{\partial z}{\partial b} = 1$$

Por lo tanto:

$$\frac{\partial e}{\partial b} = (\hat{y} - y) \cdot \hat{y}(1 - \hat{y}) \cdot 1$$

O lo que es lo mismo::

$$\boldsymbol{\frac{\partial e}{\partial b} = (\hat{y} - y)\, \hat{y}(1 - \hat{y})}$$

Ejercicio 2 (Modelado de los datos usando una única neurona (6 puntos).)**.**

$$\hat{y} = \frac{1}{1 + e^{-(w_1 \cdot x_1 \cdot w_2 \cdot x_2 + b)}}$$

Enunciado

Este ejercicio está compuesto por los siguientes apartados:

2.1 Normalizar los datos

Para normalizar los datos, es fácil hacer una sencilla aplicación en Python. El codigo esta disponible en el Anexo 1

El algoritmo realmente no necesita mucha explicacion, el codigo implementa una inicializacion para la tabla de datos, y funciones para localizar el maximo valor de una columna, el minimo y con esos valores y un iterador, normaliza los valores de las columnas aplicando:

$$rango = x_{max} - x_{min}$$

$$x_{norm} = \frac{x - min}{rango}$$

Lo cual nos da como resultado:

Lote	Edad Normalizada	Ingresos Normalizados
1	0.5263	0.8750
2	0.0789	0.1875
3	0.8684	0.3125
4	0.4211	0.8125
5	1.0000	0.0625
6	0.2632	0.5625
7	0.6842	0.6250
8	0.1579	0.3750
9	0.7368	0.4375
10	0.3421	0.6875
11	0.0000	0.0000
12	0.4737	1.0000
13	0.9474	0.1875
14	0.6053	0.7500

Cuadro 1: Valores normalizados de edad e ingresos.

2.2 Tabla detallada para cada lote t de las constantes antes y después de utilizar el lote, así como los gradientes, para los 10 primeros lotes (2.5 puntos).

Los resultados estan obtenidos de una tabla excel que estara adjunta a este PDF, la tabla es:

Lote	x_1	x_2	y	w_1	w_2	b	h	$\hat{y}$	$\partial e/\partial w_1$	$\partial e/\partial w_2$	$\partial e/\partial b$
1	0.5263	0.8750	1	0.5000	0.5000	0.5000	0.6332		0	0	0
2	0.0789	0.1875	0	0.4988	0.4972	0.4852	0.6332	0.6532	0.01168	0.02774	0.14797
3	0.8684	0.3125	0	0.4865	0.4928	0.4711	1.0738	0.7453	0.12286	0.04421	0.14148
4	0.4211	0.8125	1	0.4886	0.4967	0.4759	1.0763	0.7458	-0.02029	-0.03916	-0.04819
5	1.0000	0.0625	0	0.4742	0.4958	0.4615	0.9955	0.7302	0.14386	0.00899	0.14386
6	0.2632	0.5625	1	0.4758	0.4993	0.4677	0.8652	0.7037	-0.01625	-0.03474	-0.06177
7	0.6842	0.6250	1	0.4790	0.5022	0.4723	1.1053	0.7512	-0.03181	-0.02905	-0.04649
8	0.1579	0.3750	0	0.4767	0.4966	0.4575	0.7363	0.6762	0.02338	0.05552	0.14806
9	0.7368	0.4375	0	0.4661	0.4904	0.4432	1.0260	0.7361	0.10536	0.06256	0.14299
10	0.3421	0.6875	1	0.4681	0.4943	0.4489	0.9398	0.7191	-0.01942	-0.03902	-0.05675
11	0.0000	0.0000	0	0.4681	0.4943	0.4344	0.4489	0.6104	0	0	0.14516
12	0.4737	1.0000	1	0.4701	0.4987	0.4388	1.1504	0.7596	-0.02080	-0.04391	-0.04391
13	0.9474	0.1875	0	0.4565	0.4960	0.4243	0.9777	0.7266	0.13674	0.02706	0.14434
14	0.6053	0.7500	1	0.4594	0.4996	0.4292	1.0726	0.7451	-0.02930	-0.03631	-0.04842

Cuadro 2: Evolución de los parámetros y gradientes de la neurona en cada lote.

2.3 Predicción para los lotes 11 a 14 (1 punto).

Una vez que disponemos de la tabla completa, podemos calcular las predicciones para los elementos reales.

Tomamos los valores finales tras el entrenamiento (lote 10):

$$w_1 = 0{,}4681, \qquad w_2 = 0{,}4943, \qquad b = 0{,}4489.$$

A partir de este punto, esos valores quedan fijas, ya que no se pide continuar el entrenamiento.

2.3.1 Lote 11

Para el lote 11 conocemos:

$$x_1(11) = 0{,}0000, \qquad x_2(11) = 0{,}0000.$$

Calculamos:

$$h(11) = w_1\, x_1(11) + w_2\, x_2(11) + b = 0 + 0 + 0{,}4489 = 0{,}4489.$$

Aplicamos la función sigmoide:

$$\hat{y}(11) = \sigma(h(11)) = \frac{1}{1 + e^{-0{,}4489}} \approx 0{,}6103.$$

Para obtener la predicción binaria usamos el criterio habitual:

$$\text{Predicción}(t) = \begin{cases} 0, & \text{si } \hat{y}(t) < 0{,}5, \\ 1, & \text{si } \hat{y}(t) \geq 0{,}5. \end{cases}$$

Dado que $\hat{y}(11) = 0{,}6103 \geq 0{,}5$, la predicción para el lote 11 es:

$$\text{Predicción}(11) = 1.$$

Ahora solo tenemos que repetir para el resto de los lotes:

2.3.2 Lote 12

Para el lote 12 conocemos:

$$x_1(12) = 0{,}4737, \qquad x_2(12) = 1{,}0000.$$

Con los pesos finales tras el entrenamiento (lote 10):

$$w_1 = 0{,}4681, \qquad w_2 = 0{,}4943, \qquad b = 0{,}4489.$$

Calculamos:

$$h(12) = w_1\, x_1(12) + w_2\, x_2(12) + b = 0{,}4681 \cdot 0{,}4737 + 0{,}4943 \cdot 1{,}0000 + 0{,}4489 \approx 1{,}1504.$$

Aplicamos la función sigmoide:

$$\hat{y}(12) = \sigma\big(h(12)\big) = \frac{1}{1 + e^{-1{,}1504}} \approx 0{,}7596.$$

Para obtener la predicción binaria usamos el criterio habitual:

$$\text{Predicción}(t) = \begin{cases} 0, & \text{si } \hat{y}(t) < 0{,}5, \\ 1, & \text{si } \hat{y}(t) \geq 0{,}5. \end{cases}$$

Dado que $\hat{y}(12) = 0{,}7596 \geq 0{,}5$, la predicción para el lote 12 es:

$$\text{Predicción}(12) = 1.$$

2.3.3 Lote 13

Para el lote 13 conocemos:

$$x_1(13) = 0{,}9474, \qquad x_2(13) = 0{,}1875.$$

Con los pesos finales tras el entrenamiento (lote 10):

$$w_1 = 0{,}4681, \qquad w_2 = 0{,}4943, \qquad b = 0{,}4489.$$

Calculamos:

$$\begin{aligned} h(13) &= w_1\, x_1(13) + w_2\, x_2(13) + b \\ &= 0{,}4681 \cdot 0{,}9474 + 0{,}4943 \cdot 0{,}1875 + 0{,}4489 \\ &\approx 0{,}9851. \end{aligned}$$

Aplicamos la función sigmoide:

$$\hat{y}(13) = \sigma\big(h(13)\big) = \frac{1}{1 + e^{-0{,}9851}} \approx 0{,}7281.$$

Para obtener la predicción binaria usamos el criterio habitual:

$$\text{Predicción}(t) = \begin{cases} 0, & \text{si } \hat{y}(t) < 0{,}5, \\ 1, & \text{si } \hat{y}(t) \geq 0{,}5. \end{cases}$$

Dado que $\hat{y}(13) = 0{,}7281 \geq 0{,}5$, la predicción para el lote 13 es:

$$\text{Predicción}(13) = 1.$$

2.3.4 Lote 14

Para el lote 14 conocemos:

$$x_1(14) = 0{,}6053, \qquad x_2(14) = 0{,}7500.$$

Con los mismos pesos finales tras el entrenamiento:

$$w_1 = 0{,}4681, \qquad w_2 = 0{,}4943, \qquad b = 0{,}4489.$$

Calculamos:

$$\begin{aligned} h(14) &= w_1\, x_1(14) + w_2\, x_2(14) + b \\ &= 0{,}4681 \cdot 0{,}6053 + 0{,}4943 \cdot 0{,}7500 + 0{,}4489 \\ &\approx 1{,}1030. \end{aligned}$$

Aplicamos la función sigmoide:

$$\hat{y}(14) = \sigma\big(h(14)\big) = \frac{1}{1 + e^{-1{,}1030}} \approx 0{,}7508.$$

De nuevo, usando el mismo criterio:

$$\text{Predicción}(t) = \begin{cases} 0, & \text{si } \hat{y}(t) < 0{,}5, \\ 1, & \text{si } \hat{y}(t) \geq 0{,}5. \end{cases}$$

Como $\hat{y}(14) = 0{,}7508 \geq 0{,}5$, la predicción para el lote 14 es:

$$\text{Predicción}(14) = 1.$$

2.4 Hacer una propuesta de otras funciones de activación que consideréis interesantes basándoos en una búsqueda bibliográfica (1 punto)

Aunque en este ejercicio se ha utilizado la función sigmoide debido a que produce valores en el intervalo $(0, 1)$, existen otras funciones de activación ampliamente utilizadas en redes neuronales modernas. La mayoría de estas funciones forman parte de las implementaciones estándar de frameworks como TensorFlow, Keras o PyTorch, que las ofrecen optimizadas y listas para su uso en entornos de aprendizaje profundo. A continuación se describen algunas, junto con orientaciones sobre los casos prácticos en los que suelen emplearse.

2.4.1 ReLU (Rectified Linear Unit)

$$a(z) = \text{máx}(0,\, z)$$

Es la función de activación más utilizada en redes profundas. TensorFlow y PyTorch la emplean por defecto en capas densas y convolucionales. Su principal ventaja es que evita la saturación de gradientes cuando z es grande, acelerando el entrenamiento.

Usos recomendados:

- Redes profundas (CNN, MLP grandes).
- Problemas con datos de alta dimensión (imágenes, audio).
- Capas ocultas donde se busca rapidez de entrenamiento.

2.4.2 Leaky ReLU

$$a(z) = \begin{cases} z, & z > 0, \\ 0{,}01z, & z \leq 0. \end{cases}$$

Es una variante de ReLU que evita que las neuronas queden "muertas" (cuando sólo devuelven 0). Frameworks como TensorFlow incluyen variantes como `LeakyReLU` y `ParametricReLU`.

Usos recomendados:

- Cuando se observa que con ReLU muchas neuronas dejan de actualizarse.
- En problemas de regresión o clasificación donde los datos tienen gran variabilidad.

2.4.3 Función tangente hiperbólica (tanh)

$$a(z) = \tanh(z)$$

Produce valores entre $(-1, 1)$ y es centrada en cero, lo que suele mejorar la convergencia respecto a la sigmoide. TensorFlow la ofrece como `tf.nn.tanh`.

Usos recomendados:

- Redes recurrentes (RNN clásicas).
- Modelos donde la salida de las capas ocultas debe estar normalizada.
- Problemas con características negativas y positivas equilibradas.

2.4.4 Función Softsign

$$a(z) = \frac{z}{1 + |z|}$$

Es una alternativa suave a la tangente hiperbólica, con saturación menos brusca.

Usos recomendados:

- Redes donde tanh produce saturación excesiva.
- Problemas que requieren transiciones más suaves en la activación.

2.4.5 Función Softplus

$$a(z) = \ln\left(1 + e^z\right)$$

Puede considerarse una versión suavizada de ReLU. Es continuamente diferenciable, lo que la hace muy útil en arquitecturas donde se necesita estabilidad numérica. TensorFlow la implementa como `tf.nn.softplus`.

Usos recomendados:

- Modelos probabilísticos y variacionales (VAE).
- Redes donde ReLU genera puntos no diferenciables problemáticos.

En este problema concreto —donde se desea obtener una probabilidad en la salida— la función sigmoide sigue siendo la más adecuada. Sin embargo, en redes más complejas (como las descritas en el Ejercicio 3), funciones como ReLU, tanh o Softplus suelen ofrecer un entrenamiento más estable y eficiente, por lo que resultan alternativas interesantes según la arquitectura y la naturaleza de los datos.

2.5 Extensión del modelo con varias neuronas

En este ejercicio hemos utilizado una única neurona con dos entradas (edad e ingresos) para estimar la probabilidad de compra. Este modelo sólo puede generar una frontera de decisión lineal en el plano (x_1, x_2), lo que limita su capacidad para separar correctamente los distintos tipos de clientes.

Una extensión natural consiste en utilizar una *capa oculta* con varias neuronas. Cada neurona oculta actúa como un *detector de patrón* en el espacio de entrada, y la neurona de salida combina estas activaciones para producir la probabilidad final de compra.

2.5.1 Capa oculta con tres neuronas

Si mantenemos dos entradas (edad normalizada x_1 e ingresos normalizados x_2) y añadimos una capa oculta con tres neuronas, el modelo puede escribirse como:

$$h_j = w_{1j}\, x_1 + w_{2j}\, x_2 + b_j, \qquad a_j = \sigma(h_j), \qquad j = 1, 2, 3,$$

donde:

- w_{1j} y w_{2j} son los pesos que conectan las entradas con la neurona oculta j,
- b_j es el sesgo de la neurona oculta j,
- a_j es la activación de la neurona oculta j.

La neurona de salida combina las activaciones de la capa oculta:

$$z_{\text{out}} = v_1 a_1 + v_2 a_2 + v_3 a_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(z_{\text{out}}),$$

donde v_1, v_2, v_3 son los pesos de la capa de salida y b_{out} es su sesgo.

2.5.2 Interpretación de las neuronas ocultas

Cada neurona de la capa oculta puede interpretarse como una "región" o "perfil de cliente" en el plano edad–ingresos:

- Una neurona puede especializarse en detectar **ingresos altos** (peso w_{2j} grande y w_{1j} pequeño).
- Otra neurona puede detectar **edad media o alta** (peso w_{1j} grande y w_{2j} pequeño).
- Una tercera neurona puede activarse sólo cuando ambos valores son altos (pesos w_{1j} y w_{2j} positivos y un sesgo b_j más elevado).

La neurona de salida aprende a combinar estas activaciones: por ejemplo, asignando una mayor probabilidad de compra cuando se activa la neurona que representa el perfil de clientes con edad e ingresos altos. De este modo, la red deja de utilizar una única recta para separar las clases y pasa a aproximar fronteras de decisión más complejas, lo que mejora la capacidad de modelar diferentes tipos de clientes.

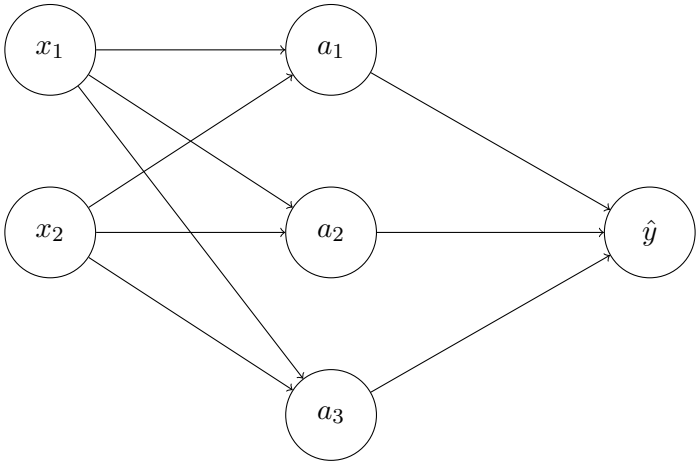

Figura 1: Ejemplo de red neuronal con una capa oculta de tres neuronas.

3 Modelado de datos usando varias neuronas

3.1 Calcular las fórmulas que faltan del gradiente, comprobando $\frac{\partial e}{\partial w_{11}}$ y calculando $\frac{\partial e}{\partial w_{12}}$, $\frac{\partial e}{\partial w_{b1}}$, $\frac{\partial e}{\partial w_{21}}$, $\frac{\partial e}{\partial w_{22}}$ y $\frac{\partial e}{\partial b_2}$

3.1.1 Comprobar $\frac{\partial e}{\partial w_{11}}$

En este apartado queremos demostrar la expresión analítica del gradiente respecto del peso w_{11}, que conecta la primera entrada x_1 con la primera neurona oculta de la red.

Recordamos que esta neurona oculta tiene:

$$h_1 = w_{11}x_1 + w_{12}x_2 + b_1,$$

y su salida es:

$$z_1 = \sigma(h_1) = \frac{1}{1+e^{-h_1}}.$$

La neurona de salida combina las salidas de las tres neuronas ocultas:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}).$$

El error utilizado es:

$$e = \frac{1}{2}(\hat{y} - y)^2.$$

Aplicación de la regla de la cadena

Como w_{11} afecta primero a h_1, luego a z_1, luego a h_{out}, y finalmente al error, aplicamos la regla de la cadena a lo largo de todo este recorrido:

$$\frac{\partial e}{\partial w_{11}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_1} \cdot \frac{\partial z_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_{11}}.$$

Ahora derivamos cada término individualmente.

1. Derivada del error respecto a la predicción

$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$

2. Derivada de la sigmoide

$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1-\hat{y}).$$

3. Derivada del modelo lineal de la neurona de salida

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}$$

solamente depende de z_1 a través del término $v_1 z_1$, por tanto:

$$\frac{\partial h_{\text{out}}}{\partial z_1} = v_1.$$

4. Derivada de la sigmoide de la neurona oculta

$$\frac{\partial z_1}{\partial h_1} = z_1(1-z_1).$$

5. Derivada del modelo lineal de la neurona oculta

$$h_1 = w_{11}x_1 + w_{12}x_2 + b_1$$

por lo que:

$$\frac{\partial h_1}{\partial w_{11}} = x_1.$$

Resultado final

Sustituyendo cada derivada:

$$\boxed{\frac{\partial e}{\partial w_{11}} = (\hat{y}-y)\,\hat{y}(1-\hat{y})\,v_1\,z_1(1-z_1)\,x_1}$$

Esta expresión coincide con la forma general del gradiente en redes neuronales de dos capas, y confirma la fórmula indicada en el enunciado.

3.1.2 Comprobar $\frac{\partial e}{\partial w_{12}}$

En este apartado queremos demostrar la expresión analítica del gradiente respecto del peso w_{12}, que conecta la segunda entrada x_2 con la primera neurona oculta de la red.

Para la primera neurona oculta recordamos:

$$h_1 = w_{11}x_1 + w_{12}x_2 + b_1, \qquad z_1 = \sigma(h_1) = \frac{1}{1+e^{-h_1}}.$$

La neurona de salida combina las salidas de las neuronas ocultas:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}),$$

y el error es:

$$e = \frac{1}{2}(\hat{y}-y)^2.$$

Aplicación de la regla de la cadena

El peso w_{12} afecta primero a h_1, luego a z_1, luego a h_{out} y finalmente al error. Por la regla de la cadena:

$$\frac{\partial e}{\partial w_{12}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_1} \cdot \frac{\partial z_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_{12}}.$$

Derivamos cada término:

- Derivada del error respecto a la predicción:

$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$

- Derivada de la sigmoide de la neurona de salida:

$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1 - \hat{y}).$$

- Derivada del modelo lineal de la neurona de salida respecto a z_1:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}} \;\Rightarrow\; \frac{\partial h_{\text{out}}}{\partial z_1} = v_1.$$

- Derivada de la sigmoide de la neurona oculta:

$$\frac{\partial z_1}{\partial h_1} = z_1(1 - z_1).$$

- Derivada del modelo lineal de la neurona oculta respecto a w_{12}:

$$h_1 = w_{11} x_1 + w_{12} x_2 + b_1 \;\Rightarrow\; \frac{\partial h_1}{\partial w_{12}} = x_2.$$

Resultado final

Sustituyendo todas las derivadas en la expresión de la regla de la cadena:

$$\boxed{\frac{\partial e}{\partial w_{12}} = (\hat{y} - y)\, \hat{y}(1 - \hat{y})\, v_1\, z_1(1 - z_1)\, x_2}$$

que es completamente análoga a la derivada respecto a w_{11}, simplemente cambiando la entrada x_1 por x_2.

3.1.3 Comprobar $\frac{\partial e}{\partial w_{21}}$

Ahora estudiamos el gradiente respecto del peso w_{21}, que conecta la primera entrada x_1 con la *segunda* neurona oculta.

Para la segunda neurona oculta:

$$h_2 = w_{21} x_1 + w_{22} x_2 + b_2, \qquad z_2 = \sigma(h_2) = \frac{1}{1 + e^{-h_2}}.$$

La neurona de salida sigue siendo:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}),$$

y el error:

$$e = \frac{1}{2}(\hat{y} - y)^2.$$

Aplicación de la regla de la cadena

El peso w_{21} afecta primero a h_2, luego a z_2, luego a h_{out} y finalmente al error. Por la regla de la cadena:

$$\frac{\partial e}{\partial w_{21}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_2} \cdot \frac{\partial z_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_{21}}.$$

Las derivadas son análogas a las de los apartados anteriores:

- Derivada del error:
$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$
- Derivada de la sigmoide de la neurona de salida:
$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1-\hat{y}).$$
- Derivada del modelo lineal de la neurona de salida respecto a z_2:
$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}} \;\Rightarrow\; \frac{\partial h_{\text{out}}}{\partial z_2} = v_2.$$
- Derivada de la sigmoide de la segunda neurona oculta:
$$\frac{\partial z_2}{\partial h_2} = z_2(1-z_2).$$
- Derivada del modelo lineal de la segunda neurona oculta respecto a w_{21}:
$$h_2 = w_{21}x_1 + w_{22}x_2 + b_2 \;\Rightarrow\; \frac{\partial h_2}{\partial w_{21}} = x_1.$$

Resultado final

Sustituyendo:

$$\boxed{\frac{\partial e}{\partial w_{21}} = (\hat{y}-y)\,\hat{y}(1-\hat{y})\,v_2\,z_2(1-z_2)\,x_1}$$

que es completamente análoga al caso de w_{11}, pero ahora la derivada se realiza a través de la segunda neurona oculta (z_2) y del peso de salida v_2.

3.1.4 Comprobar $\frac{\partial e}{\partial w_{22}}$

En este apartado queremos calcular el gradiente respecto del peso w_{22}, que conecta la segunda entrada x_2 con la *segunda* neurona oculta.

Recordamos que para esta neurona:

$$h_2 = w_{21}x_1 + w_{22}x_2 + b_2, \qquad z_2 = \sigma(h_2) = \frac{1}{1+e^{-h_2}}.$$

La neurona de salida combina las salidas de las neuronas ocultas:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}),$$

y el error es:

$$e = \frac{1}{2}(\hat{y}-y)^2.$$

Aplicación de la regla de la cadena

Dado que w_{22} afecta a h_2, luego a z_2, después a h_{out} y finalmente al error, aplicamos la regla de la cadena:

$$\frac{\partial e}{\partial w_{22}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_2} \cdot \frac{\partial z_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_{22}}.$$

A continuación derivamos los términos:

- Error respecto de la predicción:
$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$
- Derivada de la sigmoide en la neurona de salida:
$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1 - \hat{y}).$$
- Derivada del modelo lineal de la neurona de salida respecto a z_2:
$$\frac{\partial h_{\text{out}}}{\partial z_2} = v_2.$$
- Derivada de la sigmoide en la neurona oculta 2:
$$\frac{\partial z_2}{\partial h_2} = z_2(1 - z_2).$$
- Derivada del modelo lineal de la neurona oculta respecto a w_{22}:
$$h_2 = w_{21}x_1 + w_{22}x_2 + b_2 \;\Rightarrow\; \frac{\partial h_2}{\partial w_{22}} = x_2.$$

Resultado final

Sustituyendo cada derivada en la fórmula de la cadena obtenemos:

$$\boxed{\frac{\partial e}{\partial w_{22}} = (\hat{y} - y)\,\hat{y}(1 - \hat{y})\,v_2\,z_2(1 - z_2)\,x_2}$$

que es completamente análoga a los casos anteriores, con la entrada x_2 y la derivación a través de la segunda neurona oculta.

3.1.5 Comprobar $\frac{\partial e}{\partial w_{31}}$

En este apartado queremos calcular el gradiente respecto del peso w_{31}, que conecta la primera entrada x_1 con la *tercera* neurona oculta.

Para esta neurona:

$$h_3 = w_{31}x_1 + w_{32}x_2 + b_3, \qquad z_3 = \sigma(h_3) = \frac{1}{1 + e^{-h_3}}.$$

La neurona de salida combina las salidas de las tres neuronas ocultas:

$$h_{\text{out}} = v_1z_1 + v_2z_2 + v_3z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}),$$

y el error es:

$$e = \frac{1}{2}(\hat{y} - y)^2.$$

Aplicación de la regla de la cadena

El peso w_{31} afecta a h_3, luego a z_3, después a h_{out} y finalmente al error. Por la regla de la cadena:

$$\frac{\partial e}{\partial w_{31}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_3} \cdot \frac{\partial z_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial w_{31}}.$$

Derivamos cada término:

- Derivada del error:
$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$
- Derivada de la sigmoide en la neurona de salida:
$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1-\hat{y}).$$
- Derivada del modelo lineal de la neurona de salida respecto a z_3:
$$\frac{\partial h_{\text{out}}}{\partial z_3} = v_3.$$
- Derivada de la sigmoide en la tercera neurona oculta:
$$\frac{\partial z_3}{\partial h_3} = z_3(1-z_3).$$
- Derivada del modelo lineal de la tercera neurona oculta respecto a w_{31}:
$$h_3 = w_{31}x_1 + w_{32}x_2 + b_3 \;\Rightarrow\; \frac{\partial h_3}{\partial w_{31}} = x_1.$$

Resultado final

Sustituyendo todas las derivadas:

$$\boxed{\frac{\partial e}{\partial w_{31}} = (\hat{y}-y)\,\hat{y}(1-\hat{y})\,v_3\,z_3(1-z_3)\,x_1}$$

que es completamente análoga a las expresiones de w_{11} y w_{21}, pero a través de la tercera neurona oculta.

3.1.6 Comprobar $\frac{\partial e}{\partial w_{32}}$

Finalmente, calculamos el gradiente respecto del peso w_{32}, que conecta la segunda entrada x_2 con la tercera neurona oculta.

Para la tercera neurona oculta:

$$h_3 = w_{31}x_1 + w_{32}x_2 + b_3, \qquad z_3 = \sigma(h_3) = \frac{1}{1+e^{-h_3}}.$$

La neurona de salida es:

$$h_{\text{out}} = v_1z_1 + v_2z_2 + v_3z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}),$$

y el error:

$$e = \frac{1}{2}(\hat{y}-y)^2.$$

Aplicación de la regla de la cadena

El peso w_{32} afecta a h_3, luego a z_3, después a h_{out} y por último al error. Por la regla de la cadena:

$$\frac{\partial e}{\partial w_{32}} = \frac{\partial e}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_{\text{out}}} \cdot \frac{\partial h_{\text{out}}}{\partial z_3} \cdot \frac{\partial z_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial w_{32}}.$$

Las derivadas individuales son:

- Derivada del error:
$$\frac{\partial e}{\partial \hat{y}} = \hat{y} - y.$$
- Derivada de la sigmoide en la neurona de salida:
$$\frac{\partial \hat{y}}{\partial h_{\text{out}}} = \hat{y}(1 - \hat{y}).$$
- Derivada del modelo lineal de la neurona de salida respecto a z_3:
$$\frac{\partial h_{\text{out}}}{\partial z_3} = v_3.$$
- Derivada de la sigmoide de la tercera neurona oculta:
$$\frac{\partial z_3}{\partial h_3} = z_3(1 - z_3).$$
- Derivada del modelo lineal de la tercera neurona oculta respecto a w_{32}:
$$h_3 = w_{31}x_1 + w_{32}x_2 + b_3 \;\Rightarrow\; \frac{\partial h_3}{\partial w_{32}} = x_2.$$

Resultado final

Sustituyendo todas las derivadas en la expresión de la regla de la cadena:

$$\boxed{\frac{\partial e}{\partial w_{32}} = (\hat{y} - y)\,\hat{y}(1 - \hat{y})\,v_3\,z_3(1 - z_3)\,x_2}$$

que completa la familia de derivadas respecto a los pesos de la capa oculta en una red con tres neuronas ocultas y una neurona de salida.

3.1.7 Resultado general para cualquier peso w_{ij} de la capa oculta

A partir de los casos particulares anteriores, podemos obtener una expresión general del gradiente respecto de cualquier peso w_{ij} de la capa oculta, donde:

- i indica la entrada (x_1 o x_2),
- j indica la neurona oculta ($j = 1, 2, 3$).

Recordemos que la activación de cada neurona oculta es:

$$h_j = w_{1j}x_1 + w_{2j}x_2 + b_j, \qquad z_j = \sigma(h_j).$$

La neurona de salida utiliza:

$$h_{\text{out}} = v_1 z_1 + v_2 z_2 + v_3 z_3 + b_{\text{out}}, \qquad \hat{y} = \sigma(h_{\text{out}}), \qquad e = \frac{1}{2}(\hat{y} - y)^2.$$

Aplicando la regla de la cadena a lo largo de la ruta:

$$w_{ij} \longrightarrow h_j \longrightarrow z_j \longrightarrow h_{\text{out}} \longrightarrow \hat{y} \longrightarrow e,$$

obtenemos la expresión general:

$$\boxed{\frac{\partial e}{\partial w_{ij}} = (\hat{y} - y)\,\hat{y}(1-\hat{y})\,v_j\,z_j(1-z_j)\,x_i}$$

donde:

- $(\hat{y} - y)$ proviene de la derivada del error,
- $\hat{y}(1-\hat{y})$ es la derivada de la sigmoide de salida,
- v_j es el peso que conecta la neurona oculta j con la neurona de salida,
- $z_j(1-z_j)$ es la derivada de la sigmoide de la neurona oculta,
- x_i es la entrada correspondiente al peso w_{ij}.

Esta fórmula resume todos los casos particulares y constituye la regla general para el cálculo de gradientes en una red neuronal de una capa oculta con activación sigmoide.

3.2 Tabla detallada para cada lote t de las constantes antes y después de utilizar el lote, así como los gradientes, para los 10 primeros lotes

Realizar este ejercicio en Microsoft Excel haria una tabla realmente compilicada de definir, asi que, he optado por una solicion programatica, en Python, sin usar ninguna libreria especifica para redes neuronales. El codigo completo se puede revisar en el Anexo 2.

Lote	x_1	x_2	y	w_{11}	w_{12}	b_1	w_{21}	w_{22}	b_2	w_{31}	w_{32}	b_3	v_1	v_2	v_3	b	$\hat{y}$
1	0.5263	0.8750	1	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6319
2	0.0789	0.1875	0	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6153
3	0.8684	0.3125	0	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6288
4	0.4211	0.8125	1	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6298
5	1.0000	0.0625	0	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6271
6	0.2632	0.5625	1	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6240
7	0.6842	0.6250	1	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6306
8	0.1579	0.3750	0	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6196
9	0.7368	0.4375	0	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6288
10	0.3421	0.6875	1	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.4809	0.4980	0.4672	0.2053	0.2053	0.2053	0.0721	0.6270

Cuadro 3: Evolución de la red durante el entrenamiento.

3.3 Prediccion para los lotes 11 a 14

Al igual que en el ejercicio anterior, la salida es parte del mismo codigo fuente:

Lote	x_1	x_2	$\hat{y}$	Predicción
11	0.0000	0.0000	0.6108	1
12	0.4737	1.0000	0.6329	1
13	0.9474	0.1875	0.6281	1
14	0.6053	0.7500	0.6313	1

Cuadro 4: Predicciones de la red para los lotes de test.

Anexo 1: Código de normalización de datos

En este anexo se incluye el código Python utilizado en la Sección 2.1 para normalizar los datos de edad e ingresos.

```python
'''
 *
 * Copyright (C) 2025 Juan Solsona
 *
'''

import sys
from typing import Dict
from dataclasses import dataclass


@dataclass
class DatosEjemplo:
    lote: int
    edad: int
    ingresos: float
    compra: bool
    es_entrenamiento: bool

    @classmethod
    def init_con_sample(cls) -> Dict[int, 'DatosEjemplo']:
        datos = [
            cls(lote=1, edad=42, ingresos=2.6, compra=True, es_entrenamiento=True),
            cls(lote=2, edad=25, ingresos=1.5, compra=False, es_entrenamiento=True),
            cls(lote=3, edad=55, ingresos=1.7, compra=False, es_entrenamiento=True),
            cls(lote=4, edad=38, ingresos=2.5, compra=True, es_entrenamiento=True),
            cls(lote=5, edad=60, ingresos=1.3, compra=False, es_entrenamiento=True),
            cls(lote=6, edad=32, ingresos=2.1, compra=True, es_entrenamiento=True),
            cls(lote=7, edad=48, ingresos=2.2, compra=True, es_entrenamiento=True),
            cls(lote=8, edad=28, ingresos=1.8, compra=False, es_entrenamiento=True),
            cls(lote=9, edad=50, ingresos=1.9, compra=False, es_entrenamiento=True),
            cls(lote=10, edad=35, ingresos=2.3, compra=True, es_entrenamiento=True),
            cls(lote=11, edad=22, ingresos=1.2, compra=False, es_entrenamiento=False),
            cls(lote=12, edad=40, ingresos=2.8, compra=True, es_entrenamiento=False),
            cls(lote=13, edad=58, ingresos=1.5, compra=False, es_entrenamiento=False),
            cls(lote=14, edad=45, ingresos=2.4, compra=True, es_entrenamiento=False),
        ]
        return {dato.lote: dato for dato in datos}

    @staticmethod
    def normalizar(datos: Dict[int, 'DatosEjemplo'],
                   solo_entrenamiento: bool = False) -> Dict[int, 'DatosEjemplo']:
        """
        Normaliza edad e ingresos de todas las instancias en el diccionario.

        Args:
            datos: Diccionario con instancias de DatosEjemplo
            solo_entrenamiento: Si True, usa solo datos de entrenamiento para calcular rangos

        Returns:
            Nuevo diccionario con instancias normalizadas
        """
        # Calcular rangos para edad
        max_edad = calcula_max(datos, 'edad', solo_entrenamiento)
        min_edad = calcula_min(datos, 'edad', solo_entrenamiento)
        rango_edad = max_edad - min_edad
```

```
57
58          # Calcular rangos para ingresos
59          max_ingresos = calcula_max(datos, 'ingresos', solo_entrenamiento)
60          min_ingresos = calcula_min(datos, 'ingresos', solo_entrenamiento)
61          rango_ingresos = max_ingresos - min_ingresos
62
63          # Crear nuevo diccionario con instancias normalizadas
64          datos_normalizados = {}
65          for lote, instancia in datos.items():
66              edad_normalizada = (instancia.edad - min_edad) / rango_edad if rango_edad > 0 else 0
67              ingresos_normalizados = (instancia.ingresos - min_ingresos) / rango_ingresos if rango_ingresos >
68
69              datos_normalizados[lote] = DatosEjemplo(
70                  lote=instancia.lote,
71                  edad=edad_normalizada,
72                  ingresos=ingresos_normalizados,
73                  compra=instancia.compra,
74                  es_entrenamiento=instancia.es_entrenamiento
75              )
76
77          return datos_normalizados
78  # Calcula el máximo de una columna
79  def calcula_max(valores: Dict[int, DatosEjemplo], columna: str,
80                  solo_entrenamiento: bool = False) -> float:
81      max_actual: float = 0
82      for dato in valores.values():
83          if not solo_entrenamiento or dato.es_entrenamiento:
84              if dato.__dict__[columna] > max_actual:
85                  max_actual = dato.__dict__[columna]
86      return max_actual
87
88
89  # Calcula el mínimo de una columna
90  def calcula_min(valores: Dict[int, DatosEjemplo], columna: str,
91                  solo_entrenamiento: bool = False) -> float:
92      min_actual: float = sys.float_info.max
93      for dato in valores.values():
94          if not solo_entrenamiento or dato.es_entrenamiento:
95              if dato.__dict__[columna] < min_actual:
96                  min_actual = dato.__dict__[columna]
97      return min_actual
98
99
100 # Normaliza una columna (valores entre 0 y 1)
101 def normaliza_columna(valores: Dict[int, DatosEjemplo], columna: str,
102                       solo_entrenamiento: bool = False) -> Dict[
103     int, float]:
104     maximo = calcula_max(valores, columna, solo_entrenamiento)
105     minimo = calcula_min(valores, columna, solo_entrenamiento)
106     rango = maximo - minimo
107
108     valores_normalizados = {}
109     for lote, dato in valores.items():
110         if not solo_entrenamiento or dato.es_entrenamiento:
111             valor_normalizado = (dato.__dict__[columna] - minimo) / rango \
112                 if rango > 0 else 0
113             valores_normalizados[lote] = valor_normalizado
114
115     return valores_normalizados
116
117
```

```python
# Programa principal
datos_mapa = DatosEjemplo.init_con_sample()

# Normalizar edades e ingresos
edades_normalizadas = normaliza_columna(datos_mapa, 'edad')
ingresos_normalizados = normaliza_columna(datos_mapa, 'ingresos')

# Crear diccionario final con lote, edad_normalizada e ingresos_normalizados
diccionario_final = []
for lote, dato in datos_mapa.items():
    diccionario_final.append({
        'lote': lote,
        'edad_normalizada': edades_normalizadas.get(lote, None),
        'ingresos_normalizados': ingresos_normalizados.get(lote, None)
    })

print("\n" + "=" * 70)
print("Diccionario con valores normalizados:")
print("=" * 70)
for registro in diccionario_final:
    print(
        f"Lote: {registro['lote']:<3} | Edad Normalizada: "
        f"{registro['edad_normalizada']:.4f} | Ingresos Normalizados: "
        f"{registro['ingresos_normalizados']:.4f}")
```

Anexo 2: Código de la red neuronal con capa oculta

En este anexo se muestra el código Python completo que implementa el modelo de red neuronal descrito en la Sección 3.2, incluyendo el cálculo de la tabla de evolución de los pesos y las predicciones finales.

```python
"""
 *
 * Copyright (C) 2025 Juan Solsona
 *
"""
import math
from typing import List, Dict

from python.normalizar import DatosEjemplo


# Clase que define una neurona
class Neurona:
    def __init__(self, nombre: str, n_entradas: int):
        self.nombre = nombre
        self.n_entradas = n_entradas

        # Pesos y sesgo iniciales
        self.w: List[float] = [0.5 for _ in range(n_entradas)]
        self.b: float = 0.5

        # Valores intermedios
        self.h: float | None = None
        self.z: float | None = None  # salida sigmoide

        # Gradientes
        self.grad_w: List[float] = [0.0 for _ in range(n_entradas)]
```

```
        self.grad_b: float = 0.0

    def progresa_valor(self, x: List[float]) -> float:
        """Forward de una neurona: h = w·x + b, z = sigma(h)."""
        h = 0.0
        for i in range(len(self.w)):
            h += self.w[i] * x[i]
        h += self.b

        self.h = h
        self.z = 1.0 / (1.0 + math.exp(-h))
        return self.z


class Capa:
    def __init__(self, n_entradas: int, n_neuronas: int, nombre: str = ""):
        self.nombre = nombre
        self.neuronas: List[Neurona] = [
            Neurona(nombre=f"{nombre}_n{i+1}", n_entradas=n_entradas)
            for i in range(n_neuronas)
        ]

    def progresa_valor(self, x: List[float]) -> List[float]:
        return [neurona.progresa_valor(x) for neurona in self.neuronas]

class Red:
    def __init__(self):
        self.oculta = Capa(2, 3, nombre="oculta")
        self.salida = Capa(3, 1, nombre="salida")

        # Historial para 3.2 y 3.3
        self.historial_entrenamiento: list[dict] = []
        self.predicciones_test: list[dict] = []

    def progresa_valor(self, x):
        z = self.oculta.progresa_valor(x)
        yhat = self.salida.progresa_valor(z)[0]
        return z, yhat

    def backprop(self, x: list[float], y_real: float, eta: float):
        # 1. Forward
        z, yhat = self.progresa_valor(x)

        # --- CAPA DE SALIDA ---
        neurona_salida = self.salida.neuronas[0]

        # 2. Delta de la salida: (ŷ - y) * ŷ * (1-ŷ)
        delta_out = (yhat - y_real) * yhat * (1.0 - yhat)

        # 3. Gradientes de la neurona de salida
        for j in range(len(neurona_salida.w)):
            neurona_salida.grad_w[j] = delta_out * z[j]
        neurona_salida.grad_b = delta_out

        # --- CAPA OCULTA ---
        for j, neurona_oculta in enumerate(self.oculta.neuronas):
            v_j = neurona_salida.w[j]
            z_j = neurona_oculta.z

            delta_j = delta_out * v_j * z_j * (1.0 - z_j)

```

```
            for i in range(len(neurona_oculta.w)):
                neurona_oculta.grad_w[i] = delta_j * x[i]

            neurona_oculta.grad_b = delta_j

        # Actualización pesos salida
        for j in range(len(neurona_salida.w)):
            neurona_salida.w[j] -= eta * neurona_salida.grad_w[j]
        neurona_salida.b -= eta * neurona_salida.grad_b

        # Actualización pesos capa oculta
        for neurona_oculta in self.oculta.neuronas:
            for i in range(len(neurona_oculta.w)):
                neurona_oculta.w[i] -= eta * neurona_oculta.grad_w[i]
            neurona_oculta.b -= eta * neurona_oculta.grad_b

        error = 0.5 * (yhat - y_real) ** 2
        return error, yhat

    def entrenar(self, datos_mapa: dict[int, DatosEjemplo], eta: float, n_epocas: int):
        self.historial_entrenamiento = []

        for epoca in range(n_epocas):
            error_total = 0.0
            print(f"\n Época {epoca + 1} ")

            for lote_id in sorted(datos_mapa.keys()):
                dato = datos_mapa[lote_id]
                if not dato.es_entrenamiento:
                    continue

                x1 = dato.edad
                x2 = dato.ingresos
                x = [x1, x2]
                y_real = 1.0 if dato.compra else 0.0

                error, yhat = self.backprop(x, y_real, eta)
                error_total += error

                print(f"Lote {lote_id:2d} | x={x} | y_real={y_real:.1f} | "
                      f"ŷ={yhat:.4f} | e={error:.6f}")

            print(f"Error total en la época {epoca + 1}: {error_total:.6f}")

        # Al terminar la última época, guardamos el estado de la red lote a lote
        # (usando los datos de entrenamiento).
        for lote_id in sorted(datos_mapa.keys()):
            dato = datos_mapa[lote_id]
            if not dato.es_entrenamiento:
                continue

            x1 = dato.edad
            x2 = dato.ingresos
            x = [x1, x2]
            _, yhat = self.progresa_valor(x)

            n1 = self.oculta.neuronas[0]
            n2 = self.oculta.neuronas[1]
            n3 = self.oculta.neuronas[2]
            ns = self.salida.neuronas[0]

```

```
            fila = {
                "lote": lote_id,
                "x1": x1,
                "x2": x2,
                "y": int(dato.compra),

                "w11": n1.w[0],
                "w12": n1.w[1],
                "b1": n1.b,

                "w21": n2.w[0],
                "w22": n2.w[1],
                "b2": n2.b,

                "w31": n3.w[0],
                "w32": n3.w[1],
                "b3": n3.b,

                "v1": ns.w[0],
                "v2": ns.w[1],
                "v3": ns.w[2],
                "b_out": ns.b,

                "yhat": yhat
            }
            self.historial_entrenamiento.append(fila)

    def predecir_test(self, datos_mapa: dict[int, DatosEjemplo]):
        """
        Rellena self.predicciones_test con los lotes de test (no entrenamiento).
        """
        self.predicciones_test = []

        for lote_id in sorted(datos_mapa.keys()):
            dato = datos_mapa[lote_id]
            if dato.es_entrenamiento:
                continue

            x1 = dato.edad
            x2 = dato.ingresos
            x = [x1, x2]
            _, yhat = self.progresa_valor(x)
            pred = 1 if yhat >= 0.5 else 0

            self.predicciones_test.append({
                "lote": lote_id,
                "x1": x1,
                "x2": x2,
                "yhat": yhat,
                "pred": pred
            })

    def generar_tabla_entrenamiento_latex(self):
        """
        Usa self.historial_entrenamiento y escupe una tabla LaTeX con columnas limpias.
        """
        print("\n=== Tabla LaTeX para el ejercicio 3.2 (entrenamiento) ===\n")

        print(r"\begin{table}[H]")
        print(r"\centering")
        print(r"\scriptsize")
```

```
        print(r"\setlength{\tabcolsep}{3pt}")
        print(r"\begin{tabular}{|c|cc|c|ccc|ccc|cccc|c|}")
        print(r"\hline")
        print(r"Lote & $x_1$ & $x_2$ & $y$ & "
              r"$w_{11}$ & $w_{12}$ & $b_1$ & "
              r"$w_{21}$ & $w_{22}$ & $b_2$ & "
              r"$w_{31}$ & $w_{32}$ & $b_3$ & "
              r"$v_1$ & $v_2$ & $v_3$ & $b$ & $\hat y$ \\")
        print(r"\hline")

        for fila in self.historial_entrenamiento:
            print(
                f"{fila['lote']} & "
                f"{fila['x1']:.4f} & {fila['x2']:.4f} & {fila['y']} & "
                f"{fila['w11']:.4f} & {fila['w12']:.4f} & {fila['b1']:.4f} & "
                f"{fila['w21']:.4f} & {fila['w22']:.4f} & {fila['b2']:.4f} & "
                f"{fila['w31']:.4f} & {fila['w32']:.4f} & {fila['b3']:.4f} & "
                f"{fila['v1']:.4f} & {fila['v2']:.4f} & {fila['v3']:.4f} & "
                f"{fila['b_out']:.4f} & {fila['yhat']:.4f} \\\\"
            )

        print(r"\hline")
        print(r"\end{tabular}")
        print(r"\caption{Evolución de la red durante el entrenamiento.}")
        print(r"\end{table}")

    def generar_tabla_predicciones_latex(self):
        """
        Usa self.predicciones_test para generar la tabla LaTeX del 3.3.
        """
        print("\n=== Tabla LaTeX para el ejercicio 3.3 (predicciones) ===\n")

        print(r"\begin{table}[H]")
        print(r"\centering")
        print(r"\small")
        print(r"\begin{tabular}{|c|cc|c|c|}")
        print(r"\hline")
        print(r"Lote & $x_1$ & $x_2$ & $\hat y$ & Predicción \\")
        print(r"\hline")

        for fila in self.predicciones_test:
            print(
                f"{fila['lote']} & "
                f"{fila['x1']:.4f} & {fila['x2']:.4f} & "
                f"{fila['yhat']:.4f} & {fila['pred']} \\\\"
            )

        print(r"\hline")
        print(r"\end{tabular}")
        print(r"\caption{Predicciones de la red para los lotes de test.}")
        print(r"\end{table}")

    def dump(self):
        print(" Capa oculta ")
        for n in self.oculta.neuronas:
            print(f"{n.nombre}: w={n.w}, b={n.b}, h={n.h}, z={n.z}, "
                  f"grad_w={n.grad_w}, grad_b={n.grad_b}")

        print("\n Capa salida ")
        for n in self.salida.neuronas:
            print(f"{n.nombre}: w={n.w}, b={n.b}, h={n.h}, z={n.z}, "
```

```
                  f"grad_w={n.grad_w}, grad_b={n.grad_b}")

def main():
    # Inicializamos y normalizamos datos
    datos_mapa = DatosEjemplo.init_con_sample()
    datos_mapa = DatosEjemplo.normalizar(datos_mapa)  # ahora edad/ingresos están normalizados

    print(" Datos cargados (normalizados) ")
    for lote, dato in sorted(datos_mapa.items()):
        print(f"Lote {lote:2d}: edad={dato.edad:.4f}, ingresos={dato.ingresos:.4f}, "
              f"compra={dato.compra}, entreno={dato.es_entrenamiento}")

    # Creamos la red
    red = Red()

    print("\n Pesos iniciales de la red ")
    red.dump()

    # Entrenamos sobre los lotes de entrenamiento
    eta = 0.1
    n_epocas = 10
    red.entrenar(datos_mapa, eta=eta, n_epocas=n_epocas)

    print("\n Pesos de la red después del entrenamiento ")
    red.dump()

    # Predicciones para los lotes de TEST (3.3)
    red.predecir_test(datos_mapa)

    # Tablas LaTeX
    red.generar_tabla_entrenamiento_latex()
    red.generar_tabla_predicciones_latex()


if __name__ == "__main__":
    main()
```