EXAMEN PRÁCTICO: FUNDAMENTOS DE REDES NEURONALES

Luis Calderin Rodriguez
Emmanuel Londoño Madrid

Universidad Nacional de Colombia & Universidad de Córdoba
Facultad de Ciencias, Departamento de Física.
Aprendizaje Automático
Medellín, Colombia
Febrero, 2026

Índice

1. Exploración del conjunto de datos

1. Ejecute el código anterior e identifique cuántas observaciones (n) y cuántas características tiene el dataset. *(3 puntos)*

   a) ¿Cuál es el valor de n (número de registros)? *(1 pt)*

   b) ¿Cuántas características tiene cada observación? *(1 pt)*

   c) Según la notación del capítulo, escriba el vector de características $\mathbf{x}$ para la primera observación del dataset en forma de columna. *(1 pt)*

   Respuesta:

```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Cargar el dataset
california = fetch_california_housing()
X = california.data
y = california.target
feature_names = california.feature_names

print('Descripcion del dataset:')
print(california.DESCR[:1500])


# primera observacion
primera_observacion = X[0, :]
```

Listing 1: Código Inicial para cargar los datos.

   a. Después de ejecutar el código, obtenemos que el número de registros son 20640

   b. El número de características son 8

   c.

$$F = \begin{bmatrix} 8.3252 \\ 41.0000 \\ 6.9841 \\ 1.0238 \\ 322.0000 \\ 2.5556 \\ 37.8800 \\ -122.2300 \end{bmatrix}$$

2. Liste las 8 características del dataset y explique brevemente qué representa cada una. *(4 puntos)*

   Respuesta:

   Descripción de cada característica:

- MedInc: Ingreso medio del hogar en el área (en decenas de miles de dólares).
- HouseAge: Edad media de las casas en el área (en años).
- AveRooms: Promedio de habitaciones por vivienda.
- AveBedrms: Promedio de dormitorios por vivienda.
- Population: Población del área.
- AveOccup: Promedio de ocupantes por vivienda.
- Latitude: Latitud geográfica del área.
- Longitude: Longitud geográfica del área.

3. ¿Qué variable estamos tratando de predecir (etiqueta y)? ¿Qué unidades tiene? *(2 puntos)*

   Respuesta:

   La variable que estamos tratando de predecir es el valor medio de las viviendas en los distritos de California, expresado en cientos de miles de dólares.

4. Según la taxonomía del capítulo, ¿este problema es de regresión o clasificación? Justifique su respuesta explicando por qué la variable objetivo corresponde a una u otra categoría. *(3 puntos)*

   Respuesta:

   Según lo visto en el capítulo, y dado que estamos tratando de predecir el valor medio de las viviendas en los distritos de California, que e esencia corresponde a una variable numérica continua, este problema se clasifica como uno de regresión y no de clasificación. En los problemas de regresión, el objetivo es estimar un valor cuantitativo que puede tomar infinitos valores dentro de un rango determinado, como ocurre con los precios de las viviendas, que pueden variar considerablemente según múltiples factores (ubicación, tamaño, número de habitaciones, ingresos medios del distrito, entre otros). El modelo, por tanto, busca aproximar una función que relacione estas variables predictoras con un valor numérico específico.

5. **Normalización de datos:** *(8 puntos)*

   El preprocesamiento es crucial para el entrenamiento estable de redes neuronales.

   d. ¿Por qué es importante normalizar las características antes de entrenar una red neuronal? Relacione su respuesta con el proceso de descenso del gradiente. (3 pts)*(3 puntos)*

   e. ¿Qué hace exactamente StandardScaler? Escriba la fórmula matemática de la transformación. *(2 puntos)*

   f. ¿Por qué usamos fit_transform en el conjunto de entrenamiento pero solo transform en el de prueba? *(3 puntos)*

Respuestas:

```python
#Normalizacion de los datos

# Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

print(f"\nDatos divididos:")
print(f" Conjunto de entrenamiento: {X_train.shape[0]} muestras")
print(f"Conjunto de prueba: {X_test.shape[0]} muestras")

print("\nNormalizar:")

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 2: Normalización de los datos.

d. Normalizar las características es fundamental porque las redes neuronales son extremadamente sensibles a la escala de los datos. Por ejemplo, Si una característica tiene valores entre 0 y 1 y otra entre 1,000 y 1,000,000, los gradientes asociados a la característica de mayor magnitud dominarán la actualización de los pesos.

e. *StandardScaler* realiza una estandarización (Distribución normal). Su función es transformar las características para que tengan una media igual a 0 y una desviación estándar igual a 1.Esto centra la distribución de los datos y escala su varianza. La fórmula matemática para transformar cada valor x es:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

f. En el entrenamiento usamos la función *fit_transform*. El método *Fit* calcula los parámetros estadísticos (μ y σ) del conjunto de entrenamiento. Luego, *transform* aplica la fórmula anterior usando esos valores. Finalmente, en la prueba usamos solo la función *transform* ya que usamos la μ y σ que aprendimos del conjunto de entrenamiento.

2. Arquitectura del perceptrón

Ahora implementará la estructura básica de un perceptrón para regresión, conectando cada componente con la teoría del capítulo.

g. Complete la función. ¿Por qué es conveniente inicializar los pesos con valores aleatorios pequeños en lugar de ceros? *(3 pts)*

h. Según la notación del capítulo, ¿qué forma debe tener el vector $\mathbf{w}$? Verifique con `print(w.shape)` después de llamar a su función. *(2 pts)*

i. ¿Por qué el sesgo b se inicializa típicamente en cero mientras los pesos no? *(1 pts)*

Respuestas:

```python
def inicializar_parametros(n_caracteristicas):
    """
    Inicializa los pesos y el sesgo del perceptron.

    Parametros:
    -----------
    n_caracteristicas : int
        Numero de caracteristicas de entrada

    Retorna:
    --------
    w : numpy array de forma (n_caracteristicas, 1)
    b : float (escalar)
    """
    # COMPLETE EL CODIGO AQUI
    # Inicialice w con valores aleatorios peque os (usar np.random.randn)
    w = np.random.randn(n_caracteristicas, 1) * 0.01

    # Inicialice b en cero
    b = 0.0

    return w, b



# Obtener numero de caracteristicas
n_caracteristicas = X_train_scaled.shape[1]

# Inicializar parametros
w, b = inicializar_parametros(n_caracteristicas)

print(w.shape)
```

Listing 3: Inicialización de los parámetros.

g. Es conveniente inicializar los pesos con valores aleatorios pequeños en lugar de ceros ya que de lo contrario, todos se actualizarán exactamente de la misma manera. Esto se conoce como problema de simetría.

h. El vector w debe tener la forma $(8, 1)$, donde 8 es el número de características de entrada.

i. A diferencia de los pesos, inicializar el sesgo b en cero no causa el problema de simetría. Si $b = 0$, las neuronas ya estarán computando valores diferentes debido a que sus w son distintos, por lo tanto no habría ningún problema.

7. Implementación de la suma ponderada (propagación hacia adelante): *(10 puntos)*

Recuerde la ecuación fundamental: $z = \mathbf{w}^T\mathbf{x} + b$

j) Complete la función. Explique por qué usamos $\mathbf{X} @ \mathbf{w}$ en lugar de $\mathbf{w}^T @ \mathbf{X}$ cuando $\mathbf{X}$ tiene forma (m, n). *(3 pts)*

k) En un problema de regresión como este, ¿qué función de activación se usa? ¿Por qué? *(2 pts)*

l) Pruebe su función con los primeros 5 ejemplos del conjunto de entrenamiento. Muestre las predicciones iniciales y compárelas con los valores reales. ¿Son buenas predicciones? ¿Por qué? *(3 pts)*

m) Dibuje un diagrama (puede ser a mano y escaneado) mostrando el flujo de datos desde las 8 características de entrada hasta la salida y_pred, indicando dónde intervienen $\mathbf{w}$, b y la suma ponderada. *(2 pts)*

Respuestas

```
1
2  def propagacion_adelante(X, w, b):
3      """
4      Calcula la propagaci n hacia adelante del perceptr n.
5
6      Par metros:
7      -----------
8      X : numpy array de forma (m, n_caracteristicas)
9          Matriz de caracter sticas donde m es el n mero de ejemplos
10         CADA FILA es un ejemplo, CADA COLUMNA es una caracter stica
11     w : numpy array de forma (n_caracteristicas, 1)
12         Vector de pesos
13     b : float
14         Sesgo
15
16     Retorna:
17     --------
18     z : numpy array de forma (m, 1)
19         Predicciones para cada ejemplo
20     """
21     # Calcule z = X * w + b
22     # NO necesitamos transponer w porque la multiplicaci n es X @ w
23     z = np.dot(X, w) + b
24
25     return z
26 X_trained_cinco= X_train_scaled[:5,]
27 print(propagacion_adelante(X_trained_cinco, w, b))
```

Listing 4: Función de propagación hacia adelante.

j. La razón es la compatibilidad de dimensiones de la matriz y el vector. Nuestra matriz X tiene forma (m, n) (donde m son ejemplos y n características). Así mismo, el vector w tiene forma $(n, 1)$.La multiplicación de matrices requiere que las columnas de la primera coincidan con las filas de la segunda. Así, $(m, n) \times (n, 1)$ da como resultado un vector $(m, 1)$, que es exactamente una predicción por cada ejemplo.

k. Se usa la función de identidad ($f(x) = x$) ya que las funciones de activación como *Sigmoide* o *Tanh* limitan el rango de salida.

l. El vector correspondiente es:

$$Y = \begin{bmatrix} -0.00767769 \\ -0.03297133 \\ 0.02924184 \\ 0.0045381 \\ -0.02172335 \end{bmatrix}$$

m. Diagrama

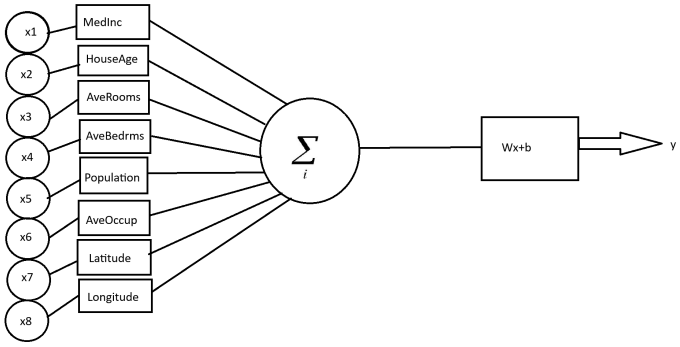

Figura 1: Flujo de datos.

8. Función de pérdida: *(8 puntos)*

   Para regresión, usaremos el Error Cuadrático Medio (MSE).

n) Complete la función. ¿Por qué elevamos al cuadrado las diferencias en lugar de usar el valor absoluto? *(3 pts)*

o) Calcule la pérdida inicial (con los pesos aleatorios). Guarde este valor para compararlo después del entrenamiento. *(2 pts)*

p) El capítulo menciona que la pérdida MSE tiene su origen en la regresión lineal de Legendre y Gauss. ¿Qué ventaja tiene el MSE para el descenso del gradiente comparado con un error absoluto medio (MAE)? *(2 pts)*

q) ¿Qué significa que la función de pérdida se "estabilice" durante el entrenamiento? Relacione con el concepto de convergencia. *(2 pts)*

Respuestas:

```
def calcular_perdida(y_pred, y_real):
    """
    Calcula el error cuadratico medio.

    Parametros:
```

```
    -----------
    y_pred : numpy array de forma (m, 1)
        Predicciones del modelo
    y_real : numpy array de forma (m, 1) o (m,)
        Valores reales

    Retorna:
    --------
    mse : float
        Error cuadratico medio
    """
    # Asegurar que y_real tenga la forma correcta
    if y_real.ndim == 1:
        y_real = y_real.reshape(-1, 1)

    # Numero de ejemplos
    m = y_real.shape[0]

    # MSE = (1/m) * sum((y_pred - y_real)^2)
    # COMPLETE EL CODIGO AQUI
    mse = (1/m) * np.sum((y_pred - y_real)**2)

    return mse



# Hacer predicciones con pesos aleatorios iniciales
y_train_reshaped = y_train.reshape(-1, 1)
y_pred_inicial = propagacion_adelante(X_train_scaled, w, b)
perdida_inicial = calcular_perdida(y_pred_inicial, y_train_reshaped)

print(perdida_inicial)
```

Listing 5: Función de pérdida.

n. Al elevar al cuadrado, los errores grandes tienen un impacto mucho mayor que los pequeños. Esto obliga al modelo a ser más preciso y evitar desviaciones grandes.

o. Al implementar el código el valor que obtenemos es: 5,66563

p. A medida que el error se acerca a cero, la pendiente (gradiente) de la función cuadrática se vuelve cada vez más pequeña. Esto permite que el descenso del gradiente pare automáticamente, dando pasos más finos para encontrar el valor óptimo sin pasarse de largo.

q. Cuando decimos que la función de pérdida se estabiliza, significa que después de muchas épocas, el valor del MSE deja de disminuir significativamente y se mantiene. La estabilización es la señal de que el modelo ha alcanzado la convergencia.

3. Retropropagación y gradiente descendente

Esta es la parte central del aprendizaje. Implementará el algoritmo que permite a la red aprender de sus errores

9. Cálculo del gradiente: *(12 puntos)*

   Para el MSE con activación lineal, derive matemáticamente las expresiones del gradiente.

r) Partiendo de $L = \frac{1}{m}\sum(\hat{y}_i - y_i)^2$ y $\hat{y} = \mathbf{X}\mathbf{w} + b$, demuestre paso a paso que: $\frac{\partial L}{\partial \mathbf{w}} = \frac{2}{m}\mathbf{X}^\top(\hat{y} - \mathbf{y})$ *(4 pts)*

s) Demuestre también que: $\frac{\partial L}{\partial b} = \frac{2}{m}\sum(\hat{y}_i - y_i)$ *(3 pts)*

t) ¿Por qué el gradiente "señala la dirección de máxima pendiente ascendente"? Explique usando la metáfora de la montaña del capítulo. *(2 pts)*

u) ¿Qué significa el signo negativo en la regla de actualización $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla L$? *(3 pts)*

Respuestas:

r) Demostración:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2\right] = \frac{1}{m}\sum_{i=1}^{m} 2(\hat{y}_i - y_i)\frac{\partial \hat{y}_i}{\partial \mathbf{w}}$$

$$= \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\mathbf{x}_i \quad \text{ya que } \hat{y}_i = \mathbf{x}_i^\top \mathbf{w} + b \Rightarrow \frac{\partial \hat{y}_i}{\partial \mathbf{w}} = \mathbf{x}_i$$

$$= \frac{2}{m}\mathbf{X}^\top(\hat{\mathbf{y}} - \mathbf{y})$$

s) Demostración:

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} 2(\hat{y}_i - y_i)\frac{\partial \hat{y}_i}{\partial b} = \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\cdot 1 = \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

10. Implemente la función de cálculo de gradientes: *(8 puntos)*

v) Complete la función siguiendo las fórmulas derivadas en la pregunta anterior. *(4 pts)*

w) Verifique que $d\mathbf{w}$ tenga la misma forma que $\mathbf{w}$. ¿Por qué es esto necesario para la actualización? *(2 pts)*

x) ¿Qué sucede con los gradientes si todas las predicciones son exactamente iguales a los valores reales? *(2 pts)*

Respuestas:

```python
def calcular_gradientes(X, y_pred, y_real):
    """
    Calcula los gradientes de la p rdida respecto a w y b.

    Par metros:
    -----------
    X : numpy array de forma (m, n)
        Matriz de caracter sticas
    y_pred : numpy array de forma (m, 1)
        Predicciones del modelo
    y_real : numpy array de forma (m, 1)
        Valores reales

    Retorna:
    --------
    dw : numpy array de forma (n, 1) - gradiente respecto a w
    db : float - gradiente respecto a b
    """
    # Obtener n mero de ejemplos
    m = X.shape[0]

    # Asegurar que y_real tenga forma correcta
    if y_real.ndim == 1:
        y_real = y_real.reshape(-1, 1)

    # Calcular el error (   - y)
    error = y_pred - y_real

    # Calcular gradiente respecto a w: dw = (2/m)  X  (   - y)
    dw = (2/m) * np.dot(X.T, error)

    # Calcular gradiente respecto a b: db = (2/m)   (   - y)
    db = (2/m) * np.sum(error)

    return dw, db
```

Listing 6: Cálculo de gradientes.

w. En efecto dw y w tienen las mismas dimmesiones, en este caso $(8, 1)$. Esto es necesario para que las respectivas operaciones matriciales estén bien definidas.

x. Cuando los gradientes son cero, esto nos indica que el modelo ha alcanzado un mínimo global en la función de pérdida. Sin embargo, esto también puede ser una señal de *Overfitting*, que el modelo no está aprendiendo, sino que esta memorizando.

11. Actualización de parámetros: *(10 puntos)*

y) Complete la función. *(2 pts)*

z) ¿Qué es la tasa de aprendizaje (*learning_rate* o η)? ¿Qué problemas pueden surgir si es muy grande? ¿Y si es muy pequeña? *(4 pts)*

aa) Ejecute una iteración completa: propagación adelante → cálculo de pérdida → gradientes → actualización. Compare la pérdida antes y después. ¿Disminuyó? *(4 pts)*

Respuestas:

```python
def actualizar_parametros(w, b, dw, db, learning_rate):
    """
    Actualiza los pesos y sesgo usando gradiente descendente.

    Par metros:
    -----------
    w : numpy array de forma (n, 1)
        Pesos actuales
    b : float
        Sesgo actual
    dw : numpy array de forma (n, 1)
        Gradiente respecto a w
    db : float
        Gradiente respecto a b
    learning_rate : float
        Tasa de aprendizaje (  )

    Retorna:
    --------
    w : numpy array de forma (n, 1)
        Pesos actualizados
    b : float
        Sesgo actualizado
    """
    # Actualizar pesos: w_nuevo = w - learning_rate * dw
    w = w - learning_rate * dw

    # Actualizar sesgo: b_nuevo = b - learning_rate * db
    b = b - learning_rate * db

    return w, b
```

Listing 7: Actualizar parámetros.

z. es un hiperparámetro que define el tamaño del paso que da el modelo en cada iteración del descenso del gradiente hacia el mínimo de la función de pérdida. Si η es muy pequeño, el modelo tarda muchísimo tiempo (muchas épocas) en llegar al mínimo, lo que consume recursos computacionales innecesarios. Por otro lado, si η el algoritmo da pasos tan largos que puede pasar por encima del mínimo de la función de pérdida.

aa. Ejecutar una iteracion completa

4. Entrenamiento completo

12. Implemente el ciclo de entrenamiento completo: *(15 puntos)*

bb. Complete la función integrando todas las funciones anteriores. *(5 pts)*

cc. Entrene el modelo con *learning_rate*=0.01 y *epochs*=1000. Grafique el historial de pérdida (época vs MSE). Incluya la gráfica en su entrega. *(4 pts)*

dd. ¿El modelo convergió? ¿Cómo lo sabe observando la gráfica? *(2 pts)*

ee. Experimente con diferentes tasas de aprendizaje: 0.001, 0.01, 0.1, 1.0. Grafique las 4 curvas de pérdida en un mismo *plot*. ¿Cuál funciona mejor? ¿Alguna diverge? *(4 pts)*

Respuestas:

bb.
```python
def entrenar_perceptron(X_train, y_train, learning_rate=0.01, epochs=1000):
    """
    Entrena un perceptron para regresion.

    Parametros:
    -----------
    X_train : datos de entrenamiento
    y_train : etiquetas de entrenamiento
    learning_rate : tasa de aprendizaje
    epochs : numero de epocas (iteraciones sobre todo el dataset)

    Retorna:
    --------
    w, b : parametros entrenados
    historial_perdida : lista con la perdida en cada epoca
    """
    # Obtener numero de caracteristicas
    n_caracteristicas = X_train.shape[1]

    # Inicializar parametros
    w, b = inicializar_parametros(n_caracteristicas)

    # Lista para guardar el historial de perdida
    historial_perdida = []

    # Asegurar que y_train tenga forma correcta
    if y_train.ndim == 1:
        y_train = y_train.reshape(-1, 1)

    # Ciclo de entrenamiento
    for epoca in range(epochs):
        # 1. Propagacion adelante
        y_pred = propagacion_adelante(X_train, w, b)

        # 2. Calcular perdida y guardar en historial
        perdida = calcular_perdida(y_pred, y_train)
        historial_perdida.append(perdida)

        # 3. Calcular gradientes
        dw, db = calcular_gradientes(X_train, y_pred, y_train)
```

```
41
42          # 4. Actualizar parametros
43          w, b = actualizar_parametros(w, b, dw, db, learning_rate)
44
45          # Imprimir progreso cada 100 epocas
46          if epoca % 100 == 0:
47              print(f"Epoca {epoca:4d}, Perdida: {perdida:.6f}")
48
49      return w, b, historial_perdida
50
51
52
53  w_entrenado, b_entrenado, historial_perdida = entrenar_perceptron(
        X_train_scaled, y_train, learning_rate=0.01, epochs=1000)
54
55  print(f"\nEntrenamiento completado")
56  print(f"Perdida final: {historial_perdida[-1]:.6f}")
57  print(f"Perdida inicial: {historial_perdida[0]:.6f}")
58  print(f"Reduccion: {historial_perdida[0] - historial_perdida[-1]:.6f}"
        )
```

Listing 8: Entrenamiento completo.

cc. Resultados del entrenamiento:

Tabla 1: Evolución de la Función de Pérdida por Época

Época	**Pérdida**
0	5.605485
100	0.710806
200	0.596081
300	0.573863
400	0.558914
500	0.548063
600	0.540151
700	0.534370
800	0.530135
900	0.527025

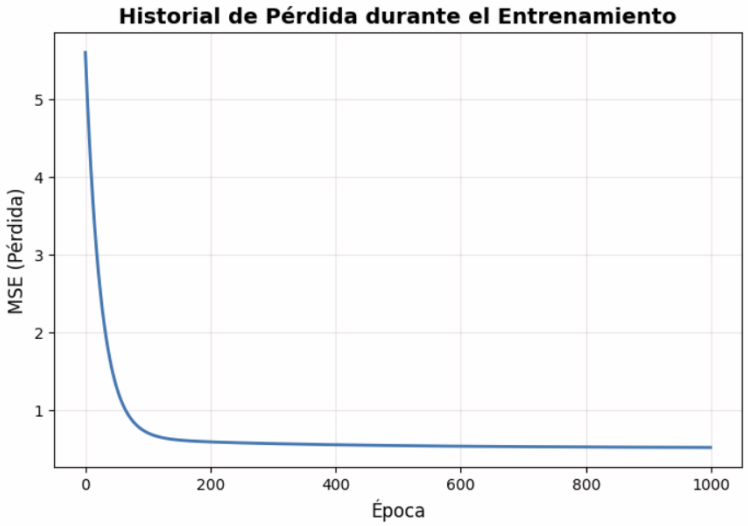

Figura 2: Épocas vs MSE.

dd. De acuerdo a los resultados, la pérdida inicial fue de 5,080730 y al finalizar el entrenamiento obtenemos una pérdida de 0,524755. Como se puede observar obtuvimos una reducción significativa en la función de pérdida, por lo tanto podemos concluir que el modelo converge.

ee. Las siguientes gráficas muestran el valor del MSE en cada iteración. Se puede observar que el modelo mas eficiente (decrece mas rápido) es en el que $\eta = 0{,}1$. Finalmente, en el caso en el que $\eta = 1{,}0$ el modelo diverge.

Figura 3: Épocas vs MSE con $\eta = 0{,}001$.

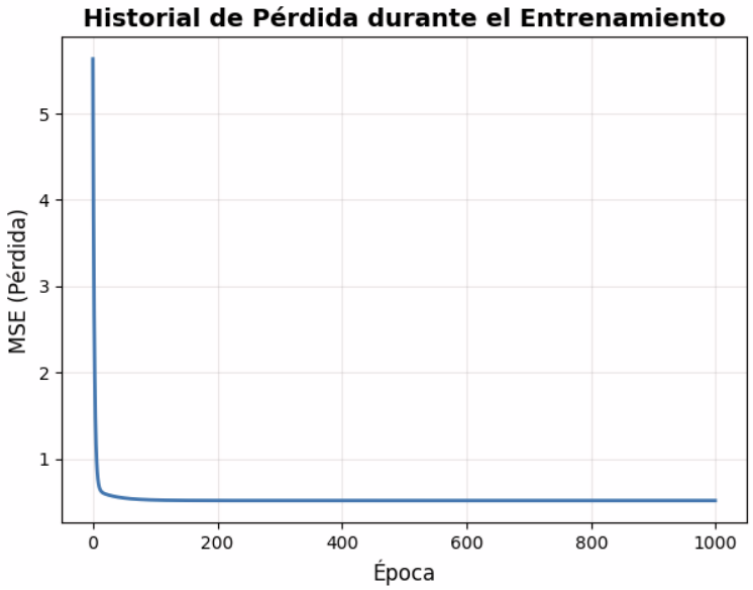

Figura 4: Épocas vs MSE con $\eta = 0{,}1$.

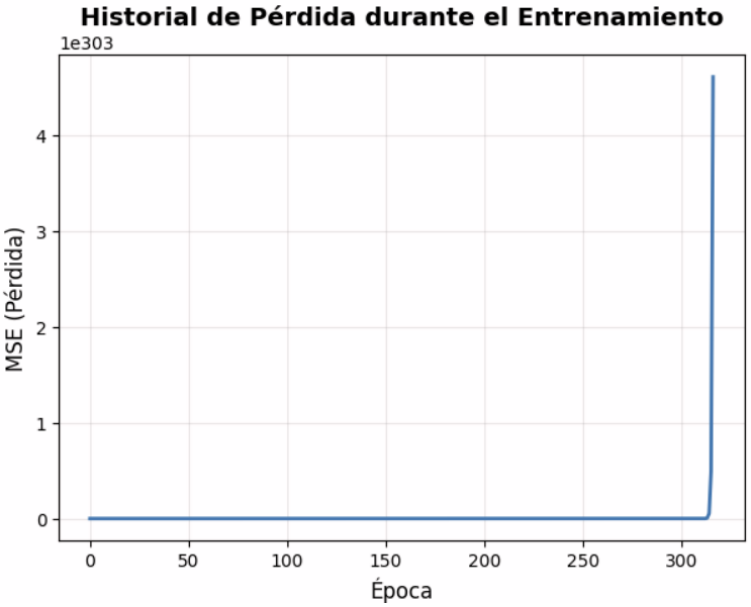

Figura 5: Épocas vs MSE con $\eta = 1{,}0$.

13. Evaluación en el conjunto de prueba.

ff) Usando los parámetros $\mathbf{w}$ y b entrenados, calcule las predicciones en *X_test_scaled*. *(2 pts*

gg) Calcule el MSE en el conjunto de prueba. ¿Es mayor o menor que el MSE del final de entrenamiento? ¿Qué indica esto sobre la generalización del modelo? *(3 pts)*

hh) Cree un *scatter plot* comparando `y_test` (eje x) vs predicciones (eje y). Añada una línea diagonal $y = x$. ¿Qué tan cerca están los puntos de esta línea ideal? *(3 pts)*

ii) Calcule el coeficiente de determinación R^2. Interprete el resultado. *(2 pts)*

Respuestas:

Respuestas de las preguntas ff. y gg.

```

learning_rates = [0.1, 0.01, 0.001, 0.0001]
resultados = {}

print("Entrenando con diferentes learning rates")

for lr in learning_rates:
    print(f"Entrenando con lr={lr}")
    w_temp, b_temp, hist = entrenar_perceptron(X_train_scaled, y_train
    , learning_rate=lr, epochs=1000)
    resultados[lr] = {'historial': hist,'w': w_temp,'b': b_temp,'
    perdida_final': hist[-1]}
    print(f"    Perdida final: {hist[-1]:.6f}")

# Mostrar comparacion de resultados
print("COMPARACION DE RESULTADOS")

for lr in learning_rates:
    perdida_final = resultados[lr]['perdida_final']
    print(f"  lr={lr:6.4f}: perdida = {perdida_final:.6f}")

y_test_reshaped = y_test.reshape(-1, 1)
y_pred_test = propagacion_adelante(X_test_scaled, resultados[mejor_lr
    ]['w'], resultados[mejor_lr]['b'])
perdida_test = calcular_perdida(y_pred_test, y_test_reshaped)

print(f"Modelo con lr={mejor_lr}:")
print(f"Perdida en entrenamiento: {resultados[mejor_lr]['perdida_final
    ']:.6f}")
print(f"Perdida en prueba: {perdida_test:.6f}")
print(f"Diferencia: {abs(perdida_test - resultados[mejor_lr]['
    perdida_final']):.6f}")

if perdida_test < resultados[mejor_lr]['perdida_final']*1.1:
    print("El modelo generaliza bien")
else:
    print("Posible overfitting")
```

Listing 9: Evaluación del modelo.

En las siguientes tablas se presenta de manera detallada los resultados arrojados al implementar el código anterior.

En la **Tabla 2** se observa la comparación del modelo con diferentes tasas de aprendizaje.

Tabla 2: Comparativa de Pérdida por Época según Learning Rate (lr)

Época	$lr = 0{,}1$	$lr = 0{,}01$	$lr = 0{,}001$	$lr = 0{,}0001$
0	5.613989	5.621401	5.617454	5.593917
100	0.524500	0.709654	3.972157	5.398999
200	0.518402	0.594920	2.872152	5.211789
300	0.517985	0.572963	2.135525	5.031980
400	0.517940	0.558229	1.641428	4.859275
500	0.517934	0.547540	1.309400	4.693393
600	0.517933	0.539750	1.085781	4.534059
700	0.517933	0.534061	0.934741	4.381013
800	0.517933	0.529896	0.832334	4.234005
900	0.517933	0.526839	0.762541	4.092794
Pérdida Final	**0.517933**	**0.524609**	**0.715038**	**3.958478**

Después de comparar las diferentes pérdidas, el modelo más óptimo es el que tiene tasa de aprendizaje igual a 0,1. Finalmente, al comparar los MSE en el periodos de entrenamiento y de prueba, se puede concluir que el modelo generaliza bien.

Tabla 3: Resumen de Generalización (Modelo Óptimo $lr = 0{,}1$)

Métrica	**Valor**
Pérdida en Entrenamiento	0.517933
Pérdida en Prueba	0.555893
Diferencia	0.037960
Conclusión	*El modelo generaliza bien*

hh. En la siguiente imagen se muestra un gráfico de dispersión comparando los valores arrojados por el modelo y sus valores reales.

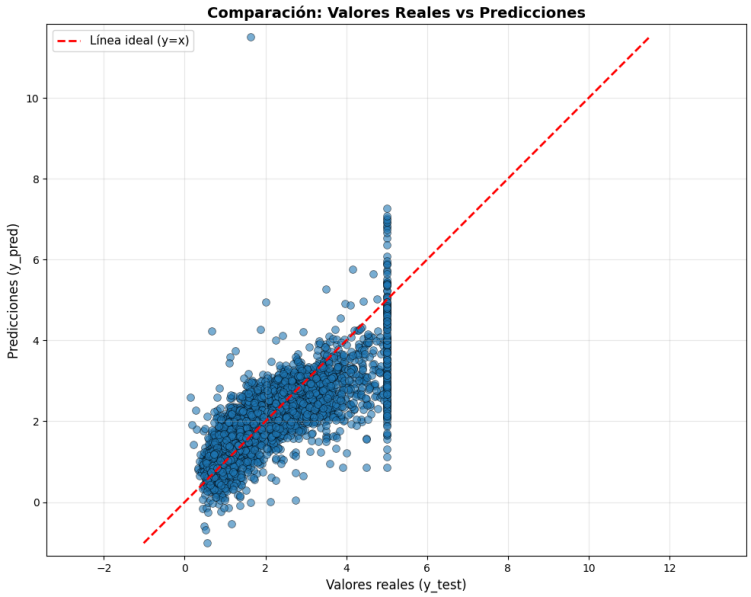

Figura 6: Test vs predicciones.

Después de analizar el gráfico, vemos que el modelo tiene un desempeño aceptable para valores pequeños, pero presenta dificultades para capturar la variabilidad de los datos más altos.

ii.
```python
SS_res = np.sum((y_test_flat - y_pred_flat)**2)
SS_tot = np.sum((y_test_flat - np.mean(y_test_flat))**2)
R2 = 1 - (SS_res / SS_tot)

print(f"Calculos intermedios:")
print(f"SS_res (suma de residuos al cuadrado): {SS_res:.6f}")
print(f"SS_tot (varianza total):                {SS_tot:.6f}")
print(f"Razon SS_res/SS_tot:                    {SS_res/SS_tot:.6f
    }")

print(f"R^2 = {R2:.6f}")
```

Listing 10: Coeficiente de determinación.

Al implementar el código, obtenemos un valor de $R^2 = 0{,}575788$; es decir que el modelo es aceptable.

5. Reflexión final

14. Limitaciones del perceptrón simple: *(5 puntos)*

jj) ¿Qué tipo de relación entre características y precio puede modelar un perceptrón simple (sin capas ocultas)? *(1 pts)*

kk) Si la relación real entre las características y el precio de las viviendas fuera altamente no lineal, ¿cómo podría extender este modelo? Relacione con el concepto de perceptrón multicapa. *(2 pts)*

ll) El capítulo menciona el Teorema de Aproximación Universal. ¿Qué nos garantiza este teorema sobre las capacidades de una red con una capa oculta? ¿Por qué entonces se usan redes profundas? *(2 pts)*

Respuestas:

jj Un perceptrón simple solo puede modelar relaciones lineales entre las características de entrada y la variable de salida (en este caso, el valor medio de las viviendas). Esto se debe a que su estructura matemática consiste básicamente en una suma ponderada de las entradas más un sesgo.

kk Para modelar relaciones no lineales, el modelo debe extenderse añadiendo capas ocultas con funciones de activación no lineales (como ReLU o Sigmoide), transformándose en un perceptrón multicapa.

ll El teorema de aproximación universal establece que una red neuronal con una sola capa oculta puede aproximar cualquier función continua con el nivel de precisión que se desee. Aunque una sola capa puede hacerlo en teoría, en la práctica suele requerir un número exponencialmente grande de neuronas, lo que es ineficiente y difícil de entrenar.