UNIVERSIDAD NACIONAL DE COLOMBIA	**Aprendizaje automático: (ML)**
Fecha: 16/02/2026	Integrantes: Daniel Giraldo Toro C.C 1000756359 Juan David Uribe Cadavid C.C 1001463742

EXAMEN PRÁCTICO: FUNDAMENTOS DE REDES NEURONALES

Predicción del Precio de Viviendas en California

Capítulo 1: Arquitectura de las Redes Neuronales

Información del Examen

Duración: 1 semana | Modalidad: Parejas | Entregables: Código Python + Documento con respuestas | Total: 100 puntos

Este examen evaluará su comprensión de los conceptos fundamentales de las redes neuronales a través de la implementación práctica de un perceptrón para regresión. Trabajará con el dataset California Housing para predecir el valor mediano de las viviendas.

Objetivo: Implementar desde cero un perceptrón simple, sin usar frameworks de deep learning (solo NumPy), respondiendo preguntas teóricas que conecten cada paso con los conceptos del capítulo.

PARTE I: EXPLORACIÓN DEL CONJUNTO DE DATOS (20 puntos)

Comenzaremos cargando y explorando el dataset. Este paso es fundamental para entender las características con las que trabajará nuestro modelo.

Código inicial para cargar los datos:

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

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 pts)*

   R// Según la información del dataset n=20640

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

   R// Cada observación tiene 8 características.

   c) Según la notación del capítulo, escriba el vector de características x para la primera observación del dataset en forma de columna. *(1 pts)*

   R// $\vec{x} = [8.3252\ 41.000\ 6.98412698\ 1.02380952\ \ 322.\ 2.55555556\ 37.88\ -122.23]$

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

   Los atributos se dividen en los siguientes:

   - MedInc: Nivel de ingreso mediano de los hogares en la zona.
   - HouseAge: Indica cuántos años, en promedio central, tienen las casas en esa zona específica.
   - AveRooms: Cantidad promedio de habitaciones por vivienda.
   - AveBedrms: Cantidad promedio de dormitorios por vivienda.

- Population: Número total de habitantes en la zona.
- AveOccup: Promedio de personas que habitan cada vivienda.
- Latitude: Ubicación geográfica de la zona en términos de latitud.
- Longitude: Ubicación geográfica de la zona en términos de longitud.

Sugerencia: Use california.feature_names y california.DESCR para obtener esta información.

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

R// La variable que se está tratando de predecir es el valor medio de las viviendas en cada zona de California. Esta variable está medida en cientos de miles de dólares estadounidenses.

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)*

R//Es un problema de regresión debido a que se busca predecir un valor continuo que en este caso sería el precio medio de las viviendas, no se trata de clasificar observaciones en categorías discretas, sino de predecir un valor real, lo que caracteriza a los problemas de regresión.

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

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

```
# 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)


# Normalizar las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

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)*

R// La normalización es fundamental ya que permite estandarizar las características del conjunto de datos, asegurando que todas tengan una escala comparable. Esto facilita que el algoritmo de descenso del gradiente actualice los pesos de manera balanceada. lo cual es importante debido a que las variables desbalanceadas generan gradientes mayores, afectando la estabilidad y velocidad de convergencia del modelo. Al normalizar las características, se evitan estos desequilibrios y se logra que el descenso del gradiente sea más eficiente, estable y rápido.

e) ¿Qué hace exactamente Standard Scaler? Escriba la fórmula matemática de la transformación. *(2 pts)*

R// StandardScaler realiza un proceso de estandarización de las características, transformando cada variable para que tenga media igual a 0 y desviación estándar igual a 1. Este procedimiento permite que todas las variables queden en una misma escala, evitando que aquellas con valores numéricamente mayores dominen el proceso de entrenamiento del modelo. La fórmula matemática de la transformación es la siguiente:

$$Z = \frac{x - \mu}{\sigma}$$

donde x es el valor original, μ es la media de la característica y σ es su desviación estándar.

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

R// Usamos fit_transform en el conjunto de entrenamiento porque en esa etapa el modelo debe aprender los parámetros de la transformación (la media y la desviación estándar). Es decir, con fit se calculan estos valores a partir de los datos de entrenamiento, y con transform se aplican para escalar los datos.
En cambio, en el conjunto de prueba solo usamos transform porque no debemos recalcular la media ni la desviación estándar con esos datos. El conjunto de prueba debe simular datos no vistos, por lo que únicamente se transforma utilizando los parámetros aprendidos en el entrenamiento.

PARTE II: ARQUITECTURA DEL PERCEPTRÓN (25 puntos)

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

Inicialización de parámetros:

6. Implemente la función de inicialización de pesos y sesgo: *(6 puntos)*

```
def inicializar_parametros(n_caracteristicas):
    '''
    Inicializa los pesos y el sesgo del perceptrón.


    Parámetros:
    -----------
    n_caracteristicas : int
        Número de características de entrada


    Retorna:
    --------
    w : numpy array de forma (n_caracteristicas, 1)
    b : float (escalar)
```

```
    '''
    # COMPLETE EL CÓDIGO AQUÍ
    # Inicialice w con valores aleatorios pequeños (usar
np.random.randn)
    # Inicialice b en cero


    return w, b
```

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

R// Es conveniente inicializar los pesos con valores aleatorios pequeños porque esto rompe la simetría del modelo, permitiendo que cada peso comience con un valor distinto y evolucione de manera diferente durante el entrenamiento. Si todos los pesos se inicializan en cero, se actualizarán de la misma forma en cada iteración, impidiendo que el modelo aprenda patrones diferenciados. Además, si los pesos fueran demasiado grandes desde el inicio, podrían generar activaciones excesivas y hacer inestable el proceso de entrenamiento, dificultando la convergencia.

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

R// El vector w debe tener forma de vector columna, es decir, una matriz de tamaño (d × 1). En cuanto a su dimensión, d corresponde al número de características de la matriz de entrenamiento X, cuya dimensión es (n × d), donde n representa el número de muestras. Por lo tanto, el tamaño de w depende directamente del número de características del conjunto de datos, ya que debe existir un peso asociado a cada una.

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

R//El sesgo b se inicializa en cero porque no genera problemas de simetría en el modelo, ya que es un único escalar. En cambio, los pesos no deben inicializarse en cero porque todos se actualizarán de la misma manera durante el entrenamiento, impidiendo que el modelo aprenda representaciones diferenciadas.

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

Recuerde la ecuación fundamental: $z = w^T x + b$

```
def propagacion_adelante(X, w, b):
    '''
    Calcula la suma ponderada para todas las observaciones.


    Parámetros:
    -----------
    X : numpy array de forma (m, n)
        m observaciones, n características
    w : numpy array de forma (n, 1)
        Vector de pesos
```

```
    b : float
        Sesgo


    Retorna:
    --------
    y_pred : numpy array de forma (m, 1)
        Predicciones del modelo
    '''
    # COMPLETE EL CÓDIGO AQUÍ
    # Calcule z = X @ w + b (producto matricial)


    return y_pred
```

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

R// porque la matriz X de entrenamiento tiene dimensión (m_train, n) y el vector de pesos w tiene dimensión (n, 1). En esta configuración, las dimensiones internas coinciden (n con n), lo que permite realizar correctamente la multiplicación matricial y obtener un resultado de dimensión (m_train, 1), es decir, una predicción por cada observación. No es necesario transponer w, ya que ya está definido como vector columna con la forma adecuada para multiplicarse directamente por X_train.

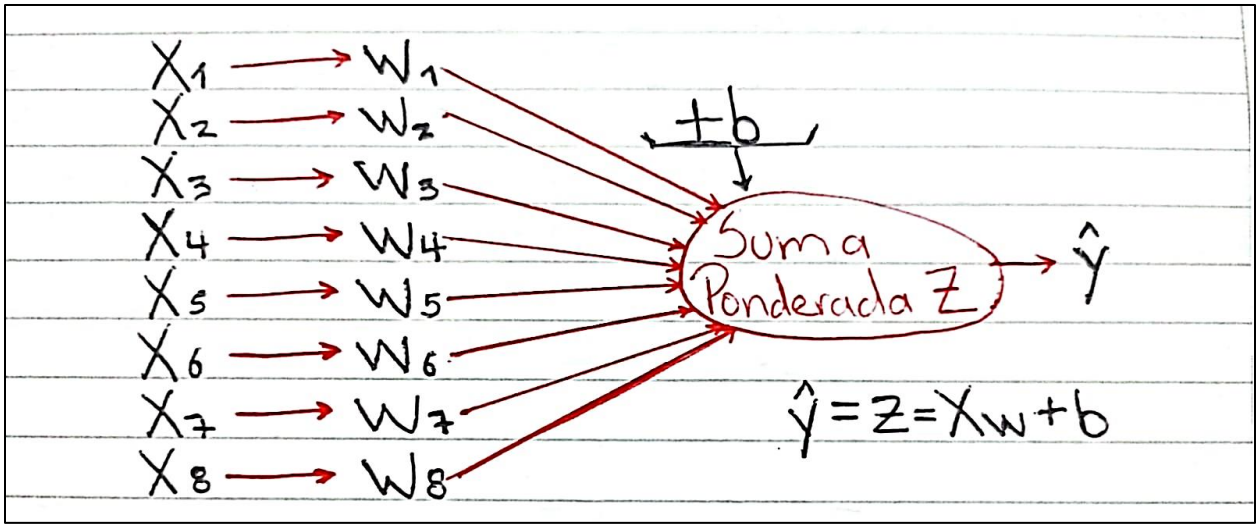
k) En un problema de regresión como este, ¿qué función de activación se usa? ¿Por qué? *(2 pts)*

R// Se utiliza la función de activación de regresión lineal. Esto se debe a que es un problema de regresión, por lo tanto, la variable objetivo es un valor continuo (el precio medio de una vivienda).

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 las predicciones? ¿Por qué? *(3 pts)*

R// Las predicciones iniciales no son buenas, ya que presentan diferencias significativas respecto a los valores reales. Esto ocurre porque los pesos fueron inicializados aleatoriamente y el modelo aún no ha sido entrenado para aprender la relación entre las variables de entrada y el precio de la vivienda.

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 w, b y la suma ponderada. *(2 pts)*



8. Función de pérdida: *(9 puntos)*

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

```
def calcular_perdida(y_pred, y_real):
    '''
    Calcula el error cuadrático medio.


    MSE = (1/m) * sum((y_pred - y_real)^2)
    '''
    # COMPLETE EL CÓDIGO AQUÍ


    return mse
```

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

R// Se elevan al cuadrado las diferencias para evitar que los errores se cancelen, penalizar más los errores grandes y facilitar el cálculo de derivadas necesarias para optimizar el modelo mediante descenso por gradiente.

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 el error absoluto medio (MAE)? *(2 pts)*

R// El MSE tiene ventaja sobre el MAE porque, como se explica en el libro, la función de pérdida determina la geometría del paisaje de optimización y la forma en que aprende el modelo. El MSE, heredado de la regresión lineal de Legendre y Gauss, es una función suave y completamente derivable, lo que permite calcular fácilmente su gradiente y aplicar la regla $w(t+1)=w(t)-\eta\nabla L(w)$. Además, en regresión lineal genera una superficie parabólica con un mínimo bien definido, facilitando la convergencia estable del descenso del gradiente. En cambio,

el MAE usa valor absoluto, que no es derivable en cero, lo que puede dificultar la optimización. Por ello, el MSE resulta más adecuado cuando se entrena mediante gradiente descendente.

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

R// Que la función de pérdida se “estabilice” significa que su valor deja de disminuir de manera significativa a medida que avanzan las épocas de entrenamiento. Es decir, después de varias actualizaciones de los pesos, el error ya no cambia mucho entre una iteración y otra. Esto está directamente relacionado con el concepto de convergencia. Decimos que el modelo converge cuando el descenso del gradiente ha llegado a un punto donde el gradiente es muy pequeño y las actualizaciones de los pesos son mínimas. En ese punto, el modelo se encuentra cerca de un mínimo (idealmente global, al menos local) de la función de pérdida.

PARTE III: RETROPROPAGACIÓN Y GRADIENTE DESCENDENTE (30 puntos)

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 $L = (1/m)\ \Sigma(\hat{y}_i - y_i)^2$ y $\hat{y} = Xw + b$, demuestre paso a paso que: $\partial L/\partial w = (2/m)\ X^T (\hat{y} - y)$ *(4 pts)*

El gradiente se obtiene mediante derivación en cadena como se muestra en la siguiente ecuación:

$$\frac{\partial L}{\partial \omega} = \frac{\partial L}{\partial \hat{y}} \cdot \left(\frac{\partial \hat{y}}{\partial \omega}\right)^T$$

Tomando en cuenta que la sumatoria de $\hat{y}_i$ se puede denotar como el vector $\hat{y}$ e igualmente para y, se realiza el cambio de notación y sus respectivas derivadas.

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

$$\left(\frac{\partial \hat{y}}{\partial \omega}\right)^T = X^T$$

Reemplazando en la regla de la cadena se llega a la ecuación descrita a continuación:

$$\frac{\partial L}{\partial \omega} = \left(\frac{2}{m}\right) X^T (\hat{y} - y)$$

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

De forma similar se empieza con la derivación en cadena como se muestra en la siguiente ecuación:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \left(\frac{\partial \hat{y}}{\partial b}\right)$$

Se realizan las derivadas por aparte para luego utilizarlas:

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

La derivada de una constante es igual a 1.

$$\frac{\partial \hat{y}}{\partial b} = 1$$

Reemplazando en la regla de la cadena se llega a la ecuación descrita a continuación:

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

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

R// El gradiente “señala la dirección de máxima pendiente ascendente” porque, matemáticamente, es el vector que indica hacia dónde crece más rápido una función en un punto dado. Si imaginamos la función de pérdida como una montaña, el gradiente nos diría cuál es la dirección más empinada para subir desde el punto en el que estamos. Es como si estuviéramos parados en una ladera y quisiéramos subir lo más rápido posible: el gradiente apunta justo hacia esa subida más pronunciada. Pero como en el entrenamiento queremos minimizar el error, no subir la montaña sino bajar hasta el punto más bajo (el mínimo), entonces nos movemos en la dirección contraria al gradiente. Por eso el descenso del gradiente utiliza −∇L, para ir bajando por la pendiente más fuerte hasta que la pérdida se estabilice y el modelo converja.

u) ¿Qué significa el signo negativo en la regla de actualización w ← w − η∇L? *(3 pts)*

R// El signo negativo indica que el movimiento se realiza en la dirección contraria al gradiente. El gradiente ∇L apunta hacia donde la función de pérdida crece más rápidamente, es decir, hacia la “subida” de la montaña en la metáfora del capítulo. Sin embargo, como el objetivo del entrenamiento es minimizar la pérdida, el modelo debe desplazarse en el sentido opuesto, es decir, hacia abajo. Por ello, en la regla de actualización w←w−η∇L, el signo negativo garantiza que los parámetros se ajusten en la dirección de máximo descenso y no de ascenso.

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

```
def calcular_gradientes(X, y_pred, y_real):
    '''
    Calcula los gradientes de la pérdida respecto a w y b.

    Parámetros:
    -----------
```

```
    X : numpy array de forma (m, n)
    y_pred : numpy array de forma (m, 1)
    y_real : numpy array de forma (m, 1)

    Retorna:
    --------
    dw : numpy array de forma (n, 1) - gradiente respecto a w
    db : float - gradiente respecto a b
    '''
    m = X.shape[0]
    error = y_pred - y_real.reshape(-1, 1)

    # COMPLETE EL CÓDIGO AQUÍ
    # dw = ...
    # db = ...

    return dw, db
```

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

w) Verifique que dw tenga la misma forma que w. ¿Por qué es esto necesario para la actualización? *(2 pts)*

R// Es necesario porque los pesos se actualizan mediante la fórmula de $w \leftarrow w - \eta \nabla L$. Esta operación implica una resta entre el vector de pesos y su gradiente. Para que dicha resta sea válida desde el punto de vista matemático y computacional, ambos deben tener exactamente las mismas dimensiones. Además, cada peso debe actualizarse con su gradiente correspondiente. Si las dimensiones no coincidieran, no sería posible realizar una actualización elemento a elemento, lo que rompería la coherencia del proceso de optimización y generaría un error en la implementación.

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

R// Significa que los gradientes son cero, por lo tanto, los pesos ya no se actualizan. En términos de optimización, el modelo ha alcanzado un mínimo de la función de pérdida. El descenso del gradiente se detiene porque no hay dirección en la cual disminuir más el error.

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

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

    w_nuevo = w - learning_rate * dw
    b_nuevo = b - learning_rate * db
```

```
    '''
    # COMPLETE EL CÓDIGO AQUÍ


    return w, b
```

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)*

R// La tasa de aprendizaje (learning rate, η) es el hiperparámetro que determina la magnitud de las actualizaciones de los pesos durante el entrenamiento de una red neuronal. En términos prácticos, define el tamaño del paso con el que el modelo se desplaza sobre la superficie de la función de pérdida en cada iteración del descenso del gradiente.

Si la tasa de aprendizaje es demasiado alta, las actualizaciones pueden ser excesivas, provocando que el modelo sobrepase el mínimo global, genere oscilaciones alrededor del mínimo o incluso diverge sin lograr converger. Por el contrario, si la tasa de aprendizaje es muy pequeña, el proceso de entrenamiento se vuelve considerablemente más lento, requiriendo un mayor número de épocas para aproximarse al mínimo de la función de pérdida.

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

R// La pérdida disminuyó después de realizar la actualización, esto se debe a que los pesos fueron ajustados utilizando el gradiente y una tasa de aprendizaje adecuada, dejando a la función del gradiente en una zona en la dirección de máximo descenso de la función de pérdida.

PARTE IV: ENTRENAMIENTO COMPLETO (25 puntos)

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

```
def entrenar_perceptron(X_train, y_train, learning_rate=0.01,
epochs=1000):
    '''
    Entrena un perceptrón para regresión.

    Parámetros:
    -----------
    X_train : datos de entrenamiento
    y_train : etiquetas de entrenamiento
    learning_rate : tasa de aprendizaje
    epochs : número de épocas (iteraciones sobre todo el dataset)

    Retorna:
    --------
    w, b : parámetros entrenados
    historial_perdida : lista con la pérdida en cada época
    '''
    n_caracteristicas = X_train.shape[1]
    w, b = inicializar_parametros(n_caracteristicas)
    historial_perdida = []

    for epoca in range(epochs):
        # COMPLETE EL CÓDIGO AQUÍ
        # 1. Propagación adelante
        # 2. Calcular pérdida y guardar en historial
        # 3. Calcular gradientes
        # 4. Actualizar parámetros

        if epoca % 100 == 0:
            print(f'Época {epoca}, Pérdida: {perdida:.4f}')

    return w, b, historial_perdida
```

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)*

R// Se puede observar que el modelo converge porque la pérdida comienza con un valor alto y disminuye progresivamente durante las primeras épocas. Posteriormente, la curva se estabiliza y se vuelve casi plana. A partir de ese punto, los cambios entre épocas son mínimos, lo que indica que el descenso del gradiente ya no realiza ajustes significativos en los pesos ni en el sesgo, señal de que el modelo alcanzó una región cercana al mínimo.

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)*

R// Al experimentar con las diferentes tasas de aprendizaje propuestas, se puede notar que para las tasas 0.001, 0.01 y 0.1 el modelo converge, ya que la pérdida disminuye progresivamente hasta estabilizarse. En particular, con una tasa igual a 0.1 el modelo converge en menos épocas, lo que hace el entrenamiento más eficiente. Por otro lado, al utilizar una tasa de aprendizaje de 1.0 se evidencia que la curva de pérdida diverge casi de inmediato, debido a que el paso es demasiado grande y termina desestabilizando el proceso de entrenamiento.

13. Evaluación en el conjunto de prueba: *(10 puntos)*

ff) Usando los parámetros 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 final de entrenamiento? ¿Qué indica esto sobre la generalización del modelo? *(3 pts)*

R// El MSE en el conjunto de prueba es menor que el MSE final de entrenamiento. Esto indica que el modelo generaliza adecuadamente y no presenta sobreajuste. La diferencia puede deberse a variaciones estadísticas entre los conjuntos o a que el conjunto de prueba sea ligeramente menos complejo que el de entrenamiento.

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)*

R// Los puntos se distribuyen alrededor de la línea ideal y=x, lo que indica que el modelo logra capturar la tendencia general entre los valores reales y las predicciones. Sin embargo, se observa una dispersión considerable respecto a la línea, especialmente en valores altos del precio, donde el modelo tiende a subestimar (muchos puntos quedan por debajo de la línea). Es decir que el modelo no es perfecto, existe un error lo cual coincide con lo obtenido en el cálculo del MSE.

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

```
# Fórmula R²
SS_res = np.sum((y_test - y_pred)**2)
SS_tot = np.sum((y_test - np.mean(y_test))**2)
R2 = 1 - (SS_res / SS_tot)
```

R// El modelo ha aprendido una relación significativa entre las entradas y la salida, pero no alcanza un ajuste perfecto.

REFLEXIÓN FINAL (Bonus: 5 puntos)

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)*

R// El perceptrón simple (sin capas ocultas) solo puede realizar relaciones lineales entre las características y el precio de las viviendas.

    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)*

R// En ese caso, el modelo podría extenderse utilizando un perceptrón multicapa (MLP), es decir, una red neuronal con una o más capas ocultas y funciones de activación no lineales como ReLU.

    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)*

R// De acuerdo con la literatura una red con una capa oculta suficientemente grande puede aproximar cualquier función continua. A pesar de esto las redes profundas siguen siendo útiles ya que requieren menos neuronas y son más eficientes computacionalmente.

CRITERIOS DE EVALUACIÓN

Rúbrica de Calificación

Código funcional y bien documentado: 40% Respuestas teóricas correctas y bien argumentadas: 35% Gráficas claras y bien etiquetadas: 15% Presentación y organización: 10%

ENTREGABLES

1. Archivo Python (.py o .ipynb) con todo el código implementado y funcionando.

2. Documento (PDF o Word) con las respuestas a todas las preguntas teóricas.

3. Gráficas generadas: curva de pérdida, comparación de learning rates, scatter plot de predicciones.

4. El código debe ejecutarse sin errores usando solo NumPy, Matplotlib y sklearn (solo para cargar datos y normalizar).

Nota sobre integridad académica

Este examen es en parejas. Puede consultar el material del curso, documentación de NumPy y recursos en línea para entender conceptos, pero el código y las respuestas deben ser de su propia autoría. Cite cualquier fuente que consulte.