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

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

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)*

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)*

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)*

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

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

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)*

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)*

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

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)*

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 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)*

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)*

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

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)*

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

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)*

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

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)*

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

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)*

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)*

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)*

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)*

```
# 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)
```

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)*

    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)*

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.