Red neuronal de clasificacion de prediccion de enfermedades cardiovasculares

La base de datos seleccionada contiene 918 registros con 18 columnas de informacion relacionada a los factores de riesgo de cardiovascular y sus metricas medicas asociadas. El objetivo del trabajo practico es aplicar los conceptos matematicos relacionados a las redes neuronales mediante el uso del lenguaje de programacion python asi como tambien algunas librerias para poder predecir la posibilidad de que un paciente contraiga una enfermedad cardiovascular.

La base de datos fue tomada del siguiente archivo, donado a la Universidad de Irvine, California en 1988. Está citada en 64 papers hasta la fecha.
International application of a new probability algorithm for the diagnosis of coronary artery disease.

In [377…
```
# Import de librerias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
import seaborn as sns
from sklearn.preprocessing import RobustScaler, StandardScaler
```

In [378…
```
# Cargando el dataset
data_path = 'heart.csv'
df = pd.read_csv(data_path, encoding='ascii', delimiter=',')
```

In [379…
```
df.head()
```

Out[379…

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exer
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

Utilizamos el siguiente comando en python para determinar los tipos de columnas de nuestro dataset.

In [380…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Esta base de datos tiene las columnas suficientes para entrenar una red neuronal de clasificación, asi como tambien tiene una columna objetivo. Vamos a transformar las columnas categoricas en numericas utilizando LabelEncoder para poder utilizar el descenso de gradiente estocastico.

In [381…

```
categorical_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slo

for col in categorical_cols:
    label_encoder = LabelEncoder()
    df[col] = label_encoder.fit_transform(df[col])
df.head()
```

Out[381…

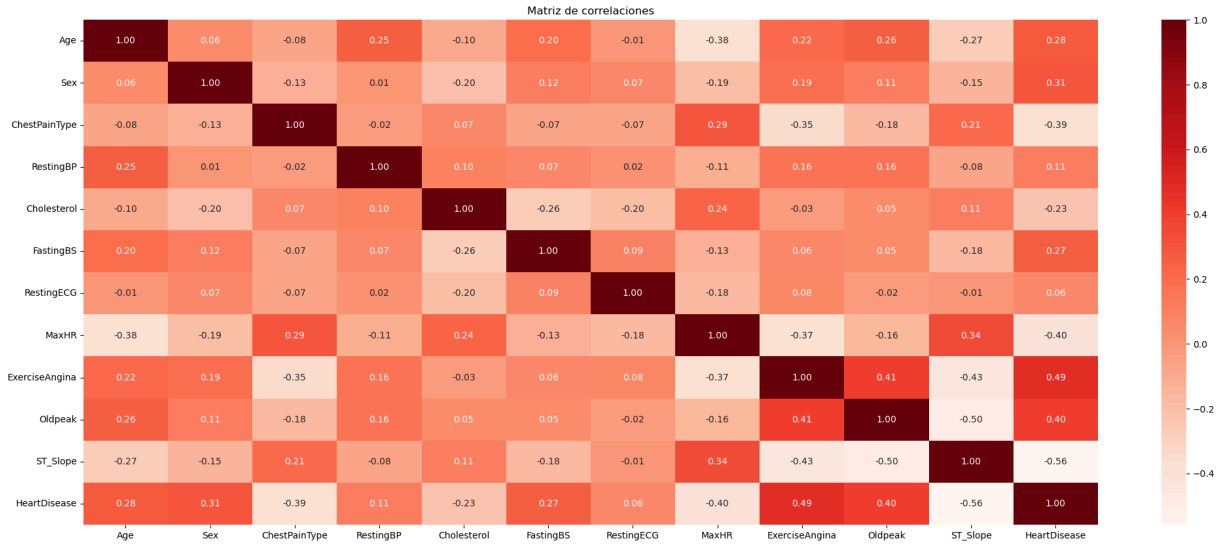
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exer
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	

2. Análisis de Correlaciones:

Usando la libreria seaborn y pandas para el analisis de correlaciones de pearson de nuestras variables numericas, queremos quedarnos con los valores que tengan una correlacion mas cercana a 1 y -1 para nuestra variable objetivo. El paper A guide to appropriate use of Correlation coefficient in medical research determina que si bien no hay un estandar universal de rango limite para determinar el tipo de correlacion, nos ofrece la siguiente tabla:

Size of Correlation	Interpretation
.90 to 1.00 (−.90 to −1.00)	Very high positive (negative) correlation
.70 to .90 (−.70 to −.90)	High positive (negative) correlation
.50 to .70 (−.50 to −.70)	Moderate positive (negative) correlation
.30 to .50 (−.30 to −.50)	Low positive (negative) correlation
.00 to .30 (.00 to −.30)	negligible correlation

In [382...
```
plt.figure(figsize=(25,10))
corr_matrix=df.corr(numeric_only=True)
sns.heatmap(corr_matrix,annot=True,cmap='Reds',fmt='.2f')
plt.title("Matriz de correlaciones")
plt.show()
```



In [383...
```
threshold = 0.4
correlation_matrix = df.corr(numeric_only=True)
high_corr_features = correlation_matrix.index[abs(correlation_matrix["HeartDisease"
high_corr_features.remove("HeartDisease")
print("Caracteristicas con mayor correlacion a variable objetivo:")
print(high_corr_features)
```

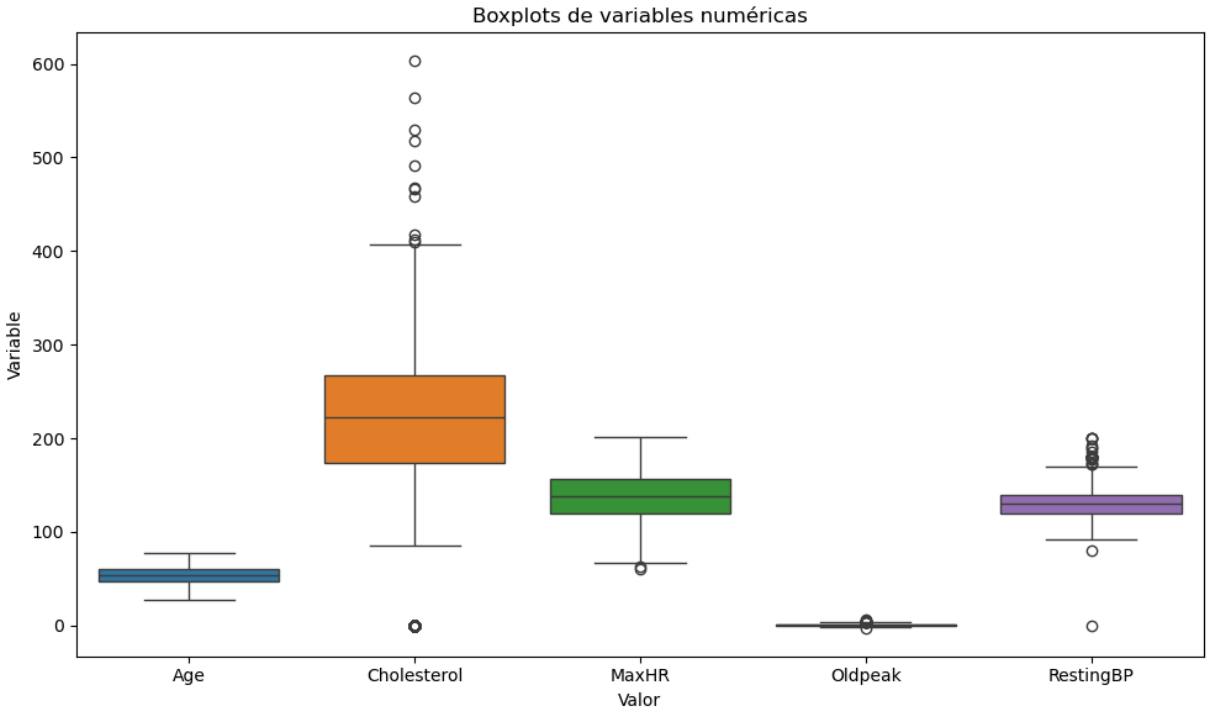
```
Caracteristicas con mayor correlacion a variable objetivo:
['MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']
```

El tamaño de la muestra es relativamente pequeño, por lo que por ahora no quisiera eliminar ninguna columna por mas de que su factor de correlacion no supere el 0.4.

3.Analisis de valores atipicos y limpieza de datos

In [384...
```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
vars_to_exclude = ['HeartDisease', 'FastingBS'] + categorical_cols
cols_to_plot = numeric_columns.difference(vars_to_exclude)
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[cols_to_plot])
plt.title('Boxplots de variables numéricas')
plt.xlabel('Valor')
plt.ylabel('Variable')
plt.tight_layout()
plt.show()

numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
numeric_columns_minus_categorical = numeric_columns.difference(vars_to_exclude)

# Calcular el rango intercuartílico (IQR) para cada columna numérica
Q1 = df[numeric_columns_minus_categorical].quantile(0.25)
Q3 = df[numeric_columns_minus_categorical].quantile(0.75)
IQR = Q3 - Q1

# Identificar valores atípicos utilizando el criterio del IQR
outliers_mask = ((df[numeric_columns_minus_categorical] < (Q1 - 1.5 * IQR)) | (df[n
rows_with_outliers = outliers_mask.any(axis=1)
# Contar el número de valores atípicos por fila
num_outliers = rows_with_outliers.sum()
print("Número de Filas con Valores Atípicos:", num_outliers)
```




```
Número de Filas con Valores Atípicos: 216
```

4. Transformaciones preliminares

Tenemos 216 filas con valores atipicos. Sin embargo, nuestro dataset viene de mediciones de pacientes reales y estos valores pueden representar casos utiles para nuestro modelo, por lo tanto no voy a removerlos. Sin embargo, me gustaria normalizar las columnas para asegurar que todas las características contribuyan de manera equilibrada al entrenamiento de la red neuronal.

Vamos a implementar el algoritmo de normalizacion basado en el desvio estandar de los datos.

In [385…

```
# 1. Ajustamos y transformamos los datos numericos

columnas_a_normalizar = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
df_s = df.copy()

# usa el algoritmo de standard scaler z = (x - mu) / sigma
def escalador_desvio_estandar(df, columnas_a_normalizar):
    for col in columnas_a_normalizar:
        mu = np.mean(df[col])
        sigma = np.std(df[col])
        df_s[col] = (df[col] - mu) / sigma
    return df_s

escalador_desvio_estandar(df, columnas_a_normalizar)
# Mostrar el DataFrame normalizado
df_s.head()
```

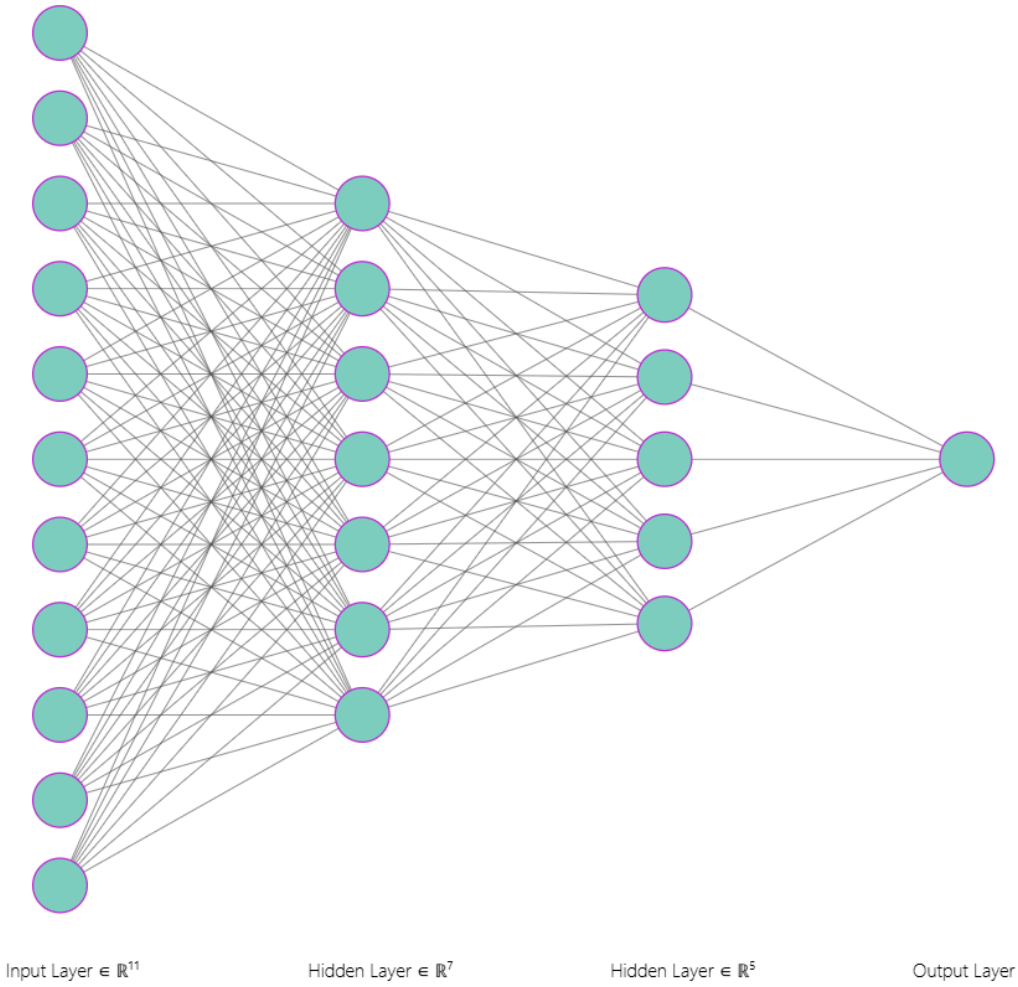
Out[385…

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
0	-1.433140	1	1	0.410909	0.825070	0	1	1.382928
1	-0.478484	0	2	1.491752	-0.171961	0	1	0.754157
2	-1.751359	1	1	-0.129513	0.770188	0	2	-1.525138
3	-0.584556	0	0	0.302825	0.139040	0	1	-1.132156
4	0.051881	1	2	0.951331	-0.034755	0	1	-0.581981

Desarrollo de la red neuronal

La arquitectura de la red tendrá inicialmente 11 neuronas en la capa de entrada, dos capas ocultas para mayor precision: la primera con 7 neuronas, la segunda con 5 neuronas, mas una neurona en la capa de salida.
El razonamiento por el cual se eligieron estos numeros de neuronas para las capas oculta esta inspirado por Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture.

Se utilizará la función de activación ReLU (Rectified Linear Unit) para los outputs de la capas ocultas, ya que la ReLu provee la suficiente no linearidad para nuestro modelo, asi como también una mejor performance computacional.
Para el output de la capa de salida,utilizaremos la función de activación Sigmoide ya que nuestro output oscila entre 0 y 1.



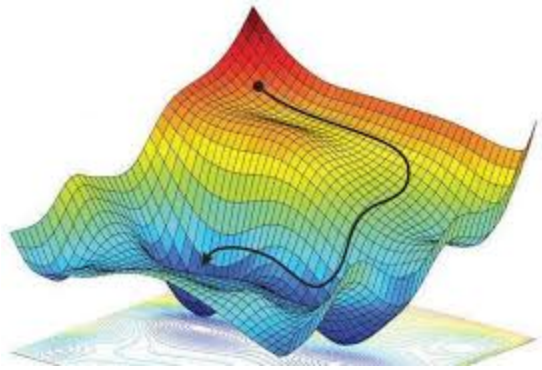
In [ ]:
```
# Extraigo las columnas de entrada del df_r
all_inputs = df_s.iloc[:, :-1].values
all_outputs = df_s.iloc[:, -1].values



# Dividir en un conjunto de entrenamiento y uno de prueba
X_train, X_test, Y_train, Y_test = train_test_split(all_inputs, all_outputs,test_si

np.random.seed(41) # Seed para nuestros valores random

n = X_train.shape[0] # número de registros de entrenamiento

# Construir una red neuronal con pesos y sesgos
# Capa 1 > Capa Oculta 1
w_hidden1 = np.random.rand(7, 11) * 2 - 1 # Dimensiones de la matriz de pesos - 7 n
# Añadimos sesgos para que los valores random puedan ser negativos
b_hidden1 = np.random.rand(7, 1) * 2 - 1 # Dimensiones de la matriz de sesgos - 1 s

# Capa Oculta 1 > Capa Oculta 2
w_hidden2 = np.random.rand(5, 7) * 2 - 1
b_hidden2 = np.random.rand(5,1) * 2 - 1

# Capa Oculta 1 > Capa de Salida
```

```
w_output = np.random.rand(1, 5) * 2 - 1
b_output = np.random.rand(1, 1) * 2 - 1


# Funciones de activacion
relu = lambda x: np.maximum(x, 0)
relu_derivative = lambda x: (x > 0).astype(float)
logistic = lambda x: 1 / (1 + np.exp(-x))
logistic_derivative = lambda x: logistic(x) * (1 - logistic(x))

# Funcion que corre la red neuronal con los datos de entrada para predecir la salid
def forward_prop(X):
    Z1 = w_hidden1 @ X + b_hidden1
    A1 = relu(Z1)
    Z2 = w_hidden2 @ A1 + b_hidden2
    A2 = relu(Z2)
    Z3 = w_output @ A2 + b_output
    A3 = logistic(Z3)
    return Z1, A1, Z2, A2, Z3, A3


test_predictions = forward_prop(X_test.transpose())[5] # el [3] para obtener A2
test_predictions = (test_predictions >= 0.5).astype(int)  # Convertir las prediccio
accuracy = np.mean(test_predictions == Y_test.reshape(1, -1))  # Calcular la precis
print("ACCURACY: ", accuracy)
```

```
ACCURACY:  0.5294117647058824
```

Una precisión de alrededor del 50% se puede esperar de una red neuronal sin entrenar. Se utilizara el descenso de gradiente estocástico para el entrenamiento de la red.

Imagen ilustrativa:



In [387…

```
L = 0.01 # La tasa de aprendizaje inicial

# Devuelve pendientes para pesos y sesgos
# usando la regla de la cadena
def backward_prop(Z1, A1, Z2, A2, Z3, A3, X, Y):
    dC_dA3 = 2 * (A3 - Y)
    dA3_dZ3 = logistic_derivative(Z3)
    dZ3_dW3 = A2
    dZ3_dA2 = w_output
    dC_dZ3 = dC_dA3 * dA3_dZ3
```

```
    dC_dW3 = dC_dZ3 @ dZ3_dW3.T
    dC_dB3 = np.sum(dC_dZ3, axis=1, keepdims=True)

    dC_dA2 = dZ3_dA2.T @ dC_dZ3
    dA2_dZ2 = relu_derivative(Z2)
    dZ2_dW2 = A1
    dZ2_dA1 = w_hidden2
    dC_dZ2 = dC_dA2 * dA2_dZ2

    dC_dW2 = dC_dZ2 @ dZ2_dW2.T
    dC_dB2 = np.sum(dC_dZ2, axis=1, keepdims=True)

    dC_dA1 = dZ2_dA1.T @ dC_dZ2
    dA1_dZ1 = relu_derivative(Z1)
    dZ1_dW1 = X
    dC_dZ1 = dC_dA1 * dA1_dZ1

    dC_dW1 = dC_dZ1 @ dZ1_dW1.T
    dC_dB1 = np.sum(dC_dZ1, axis=1, keepdims=True)

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2, dC_dW3, dC_dB3



def entrenamiento(l_rate, iters, w_h1, b_h1, w_h2, b_h2, w_o, b_o):

    # Para los graficos
    accuracy_train_l = []
    accuracy_test_l = []

    # Ejecutar descenso de gradiente
    for i in range(iters):
    # seleccionar aleatoriamente un conjunto de datos de entrenamiento
        idx = np.random.choice(n, 1, replace=False)
        X_sample = X_train[idx].transpose()
        Y_sample = Y_train[idx].reshape(1, 1)

        # pasar datos seleccionados aleatoriamente a través de la red neuronal
        Z1, A1, Z2, A2, Z3, A3 = forward_prop(X_sample)

        # distribuir error a través de la retropropagación y devolver pendientes pa
        dW1, dB1, dW2, dB2, dW3, dB3 = backward_prop(Z1, A1, Z2, A2, Z3, A3, X_samp

        # Actualizar pesos y sesgos
        w_h1 -= l_rate * dW1
        b_h1 -= l_rate * dB1
        w_h2 -= l_rate * dW2
        b_h2 -= l_rate * dB2
        w_o -= l_rate * dW3
        b_o -= l_rate * dB3

        # Cálculo de precisión del train
        train_predictions = forward_prop(X_train.transpose())[5]
        train_predictions = (train_predictions >= 0.5).astype(int)  # Convertir las
        accuracy_train = np.mean(train_predictions == Y_train.reshape(1, -1))  # Ca
```

```
        accuracy_train_l.append(accuracy_train)


        # Cálculo de precisión del test
        test_predictions = forward_prop(X_test.transpose())[5]
        test_predictions = (test_predictions >= 0.5).astype(int)  # Convertir las p
        accuracy_test = np.mean(test_predictions == Y_test.reshape(1, -1))  # Calcu
        accuracy_test_l.append(accuracy_test)
    return accuracy_train_l, accuracy_test_l
```

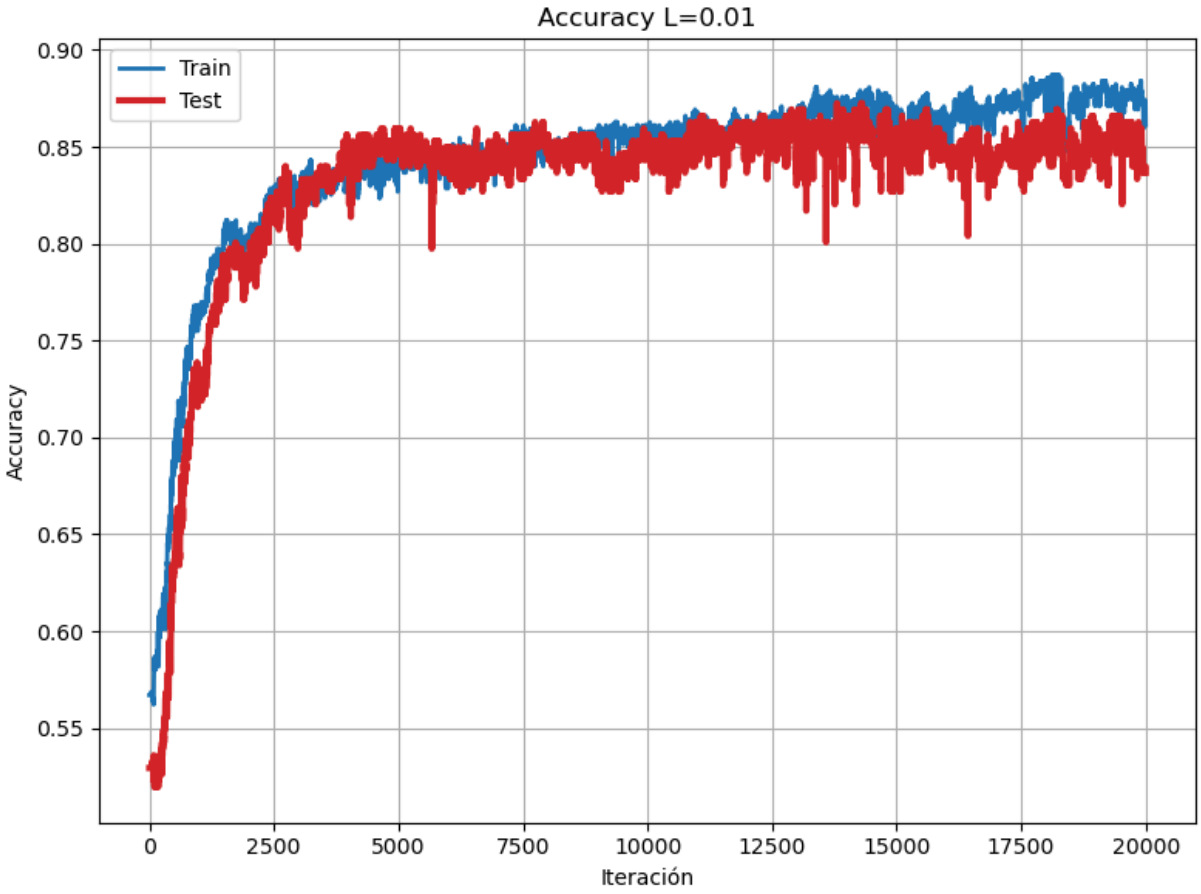
In [388…

```
acc_train_20000, acc_test_20000 = entrenamiento(L, 20000,  w_hidden1, b_hidden1, w_

# Cálculo de precisión
test_predictions = forward_prop(X_test.transpose())[5]
test_predictions = (test_predictions >= 0.5).astype(int)  # Convertir las prediccio
accuracy = np.mean(test_predictions == Y_test.reshape(1, -1))  # Calcular la precis
print("ACCURACY: ", accuracy)

# El valor de precisión (accuracy) es  83,3% (mayor que el anterior) indica la prop
# por la red neuronal en el conjunto de prueba (X_test) después de entrenarla media
```

```
ACCURACY:  0.8366013071895425
```

In [389…

```
def grafico_accuracy(accuracy_train_l,accuracy_test_l):
    iteraciones = list(range(len(accuracy_train_l)))  # Eje X = número de iteración

    fmt_test = {
        'color': 'tab:red',
        'ls': 'solid',
        'lw': 3,
    }

    fig, ax = plt.subplots(1, 1, figsize=(8, 6))
    ax.plot(iteraciones, accuracy_train_l, label='Train', color='tab:blue', lw=2)
    ax.plot(iteraciones, accuracy_test_l, label='Test', **fmt_test)

    ax.grid(which='both')
    ax.legend()
    ax.set_title(f'Accuracy L={L}')
    ax.set_xlabel('Iteración')
    ax.set_ylabel('Accuracy')

    fig.tight_layout()
    plt.show()
grafico_accuracy(acc_train_20000, acc_test_20000)
```



Como podemos observar en el grafico de accuracy, conforme se avanzan las iteraciones, el modelo empieza a ajustar sus pesos cada vez mas a los datos de entrenamiento lo cual muestra sintomas de estar "sobre ajustandose" a los datos de entrenamiento. Para evitarlo, podriamos probar distintos valores de learning rate e iteraciones de entrenamiento.

Para ver de una manera minuciosa como la red predice los resultados, podemos realizar pruebas con valores aleatorios del dataset.

In [391…

```
filas_aleatorias = df.iloc[:,:].sample(n=10)  # Obtengo una muestra de 10 filas.
filas_aleatorias
```

Out[391...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Ex
519	63	1	0	96	305	0	2	121	
679	63	1	3	145	233	1	0	150	
171	40	1	2	140	235	0	1	188	
467	63	0	1	132	0	0	1	130	
424	60	1	2	120	0	1	1	141	
540	62	1	2	138	204	0	2	122	
630	71	0	0	112	149	0	1	125	
233	41	1	0	112	250	0	1	142	
166	50	1	0	140	231	0	2	140	
750	46	0	2	142	177	0	0	160	

In [392...

```
filas_aleatorias_sin_target = filas_aleatorias.loc[:, filas_aleatorias.columns[:-1]
filas_aleatorias_sin_target
```

Out[392...

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Ex
519	63	1	0	96	305	0	2	121	
679	63	1	3	145	233	1	0	150	
171	40	1	2	140	235	0	1	188	
467	63	0	1	132	0	0	1	130	
424	60	1	2	120	0	1	1	141	
540	62	1	2	138	204	0	2	122	
630	71	0	0	112	149	0	1	125	
233	41	1	0	112	250	0	1	142	
166	50	1	0	140	231	0	2	140	
750	46	0	2	142	177	0	0	160	

In [393...

```
scaler = StandardScaler()
df_features = df.drop(columns='HeartDisease')  # O el nombre correcto de tu variabl
scaler.fit(df_features)

for i, row in filas_aleatorias_sin_target.iterrows():
    X_sample = pd.DataFrame([row.values], columns=df_features.columns)  # Datos de
    X_sample = scaler.transform(X_sample) # Normalizar

    Z1, A1, Z2, A2, Z3, A3 = forward_prop(X_sample.transpose()) # Esta función devu
```

```
    prediccion = (A3 > 0.5).astype(int) # Realiza una clasificación binaria

    print("Fila", i)
    print("Activaciones de la capa de salida:", A3)
    print("Predicción:", prediccion)
    valor = filas_aleatorias.loc[i, 'HeartDisease']
    print(f"Death Event en el Dataframe es: {valor}")
    print("\n")
```

```
Fila 519
Activaciones de la capa de salida: [[0.96338097]]
Predicción: [[1]]
Death Event en el Dataframe es: 1


Fila 679
Activaciones de la capa de salida: [[0.88053966]]
Predicción: [[1]]
Death Event en el Dataframe es: 0


Fila 171
Activaciones de la capa de salida: [[0.01301167]]
Predicción: [[0]]
Death Event en el Dataframe es: 0


Fila 467
Activaciones de la capa de salida: [[0.39035411]]
Predicción: [[0]]
Death Event en el Dataframe es: 0


Fila 424
Activaciones de la capa de salida: [[0.9991608]]
Predicción: [[1]]
Death Event en el Dataframe es: 1


Fila 540
Activaciones de la capa de salida: [[0.98710249]]
Predicción: [[1]]
Death Event en el Dataframe es: 1


Fila 630
Activaciones de la capa de salida: [[0.80964757]]
Predicción: [[1]]
Death Event en el Dataframe es: 0


Fila 233
Activaciones de la capa de salida: [[0.49015356]]
Predicción: [[0]]
Death Event en el Dataframe es: 0


Fila 166
Activaciones de la capa de salida: [[0.74369738]]
Predicción: [[1]]
Death Event en el Dataframe es: 1


Fila 750
Activaciones de la capa de salida: [[0.22994884]]
```

```
Predicción: [[0]]
Death Event en el Dataframe es: 0
```

Sin embargo, una mejor manera de visualizar esto, es con una matriz de confusion que nos demuestre los positivos y negativos verdaderos y predichos. Esta matriz compara las predicciones del modelo con los resultados reales y nos muestra donde acertó o falló.

Verdadero positivo (TP): el modelo predijo correctamente un resultado positivo, es decir, el resultado real fue positivo.
Verdadero negativo (TN): El modelo predijo correctamente un resultado negativo, es decir, el resultado real fue negativo.
Falso positivo (FP): El modelo predijo incorrectamente un resultado positivo, es decir, el resultado real fue negativo. También se conoce como error de tipo I.
Falso Negativo (FN): El modelo predijo incorrectamente un resultado negativo, es decir, el resultado real fue positivo. También se conoce como error de Tipo II.

Además, con los resultados de esta matriz se pueden calcular otras medidas como:

1. Exactitud (Accuracy):

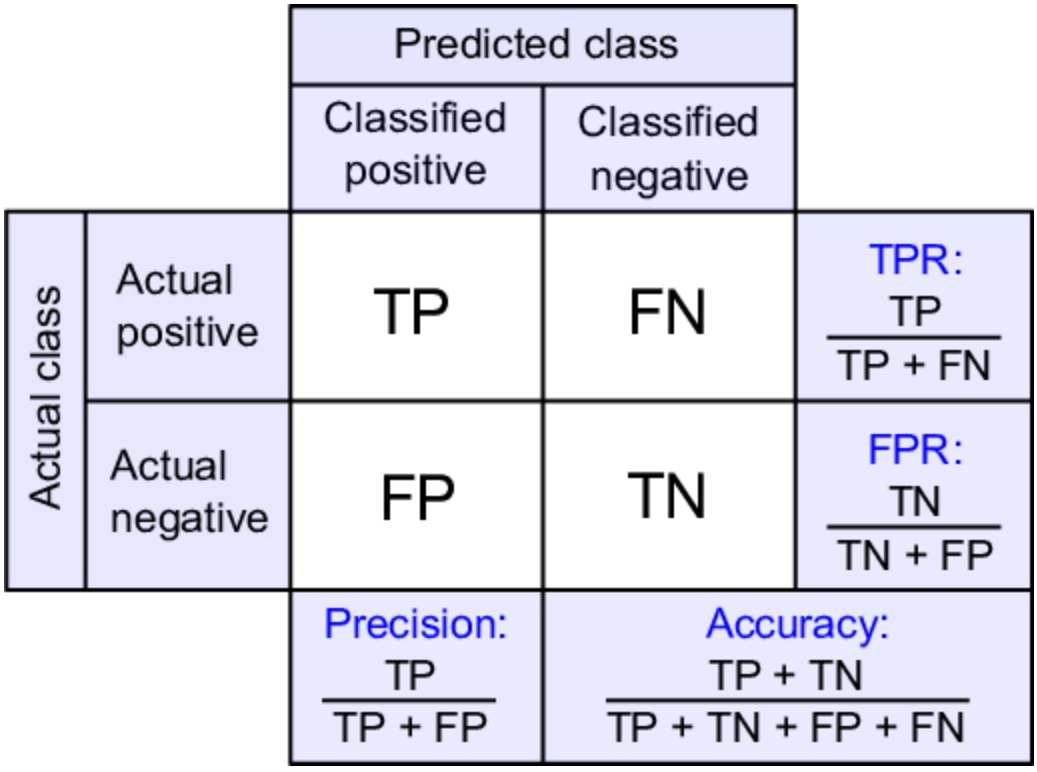
La precisión muestra cuántas predicciones acertó el modelo de entre todas las predicciones.

2. Precisión (Precision):

La precisión se centra en la calidad de las predicciones positivas del modelo. Nos indica cuántas de las predicciones "positivas" fueron realmente correctas.3. Recuperacion (TPR): La recuperación mide la eficacia del modelo para predecir casos positivos. Muestra la proporción de verdaderos positivos detectados respecto a todos los casos positivos reales.

4. Especificidad (FPR):

La especificidad mide la capacidad de un modelo para identificar correctamente instancias negativas.



In [394…

```
from sklearn.metrics import confusion_matrix, classification_report

y_true = Y_test.flatten()
y_prediction = test_predictions.flatten()
cm = confusion_matrix(y_true,y_prediction)
print("Matriz de Confusión:")
print(cm)
print("\nReporte Detallado:")
print(classification_report(y_true, y_prediction, target_names=["Sin enfermedad", "
```

```
Matriz de Confusión:
[[119  25]
 [ 25 137]]

Reporte Detallado:
                precision    recall  f1-score   support

Sin enfermedad       0.83      0.83      0.83       144
Con enfermedad       0.85      0.85      0.85       162

      accuracy                           0.84       306
     macro avg       0.84      0.84      0.84       306
  weighted avg       0.84      0.84      0.84       306
```

Podemos volver a comprobar que la precisión del modelo es relativamente alta.

Comparacion con SciKit learn

Vamos a implementar una red neuronal con la misma arquitectura que la nuestra mediante la libreria de python SciKit learn.

In [395…
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values

# Normalizar los datos
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Separar los datos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3, random_sta

# Crear el modelo
nn = MLPClassifier(solver='sgd',               #Indica el algoritmo a utilizar para
                   hidden_layer_sizes=(7, 5),  #Red con dos capas ocultas, la prime
                   activation='relu',
                   max_iter=200000,
                   learning_rate_init=0.01,     #Tasa de aprendizaje
                   alpha=0.0001)                #prevenir el sobreajuste

# Entrenar el modelo
nn.fit(X_train, Y_train)            #Durante el proceso de ajuste, el algoritmo de
                                    #entrenamiento, actualizando los pesos de la re
# Imprimir pesos y sesgos
#print("Pesos de las capas:", nn.coefs_)
#print("Sesgos de las capas:", nn.intercepts_)

# Evaluar el modelo
train_score = nn.score(X_train, Y_train)         #calcula la precisión del modelo e
test_score = nn.score(X_test, Y_test)            #calcula la precisión del modelo e
print("Red neuronal con (7,5) neuronas")
print("Puntaje del conjunto de entrenamiento: %f" % train_score)
print("Puntaje del conjunto de prueba: %f" % test_score)
```

```
Red neuronal con (7,5) neuronas
Puntaje del conjunto de entrenamiento: 0.887255
Puntaje del conjunto de prueba: 0.879085
```

Como podemos observar, el modelo provisto por la libreria nos devuelve una punteria bastante similar al modelo desarollado manualmente. Podemos probar agregando mas neuronas y capas ocultas, ya que este framework hace que su implementacion sea mucho mas rápida.

In [396…
```
# Segunda prueba con 3 capas ocultas - 15, 10, 5
nn2 = MLPClassifier(solver='sgd',               #Indica el algoritmo a utilizar par
                   hidden_layer_sizes=(15, 10, 5),  #Red con tres capas ocultas, 15
                   activation='relu',
                   max_iter=200000,
                   learning_rate_init=0.01,     #Tasa de aprendizaje
                   alpha=0.0001)                #prevenir el sobreajuste
```

```
# Entrenar el modelo
nn2.fit(X_train, Y_train)

# Evaluar el modelo
train_score = nn.score(X_train, Y_train)         #calcula la precisión del modelo e
test_score = nn.score(X_test, Y_test)            #calcula la precisión del modelo e
print("Red neuronal con (15,10,5) neuronas")
print("Puntaje del conjunto de entrenamiento: %f" % train_score)
print("Puntaje del conjunto de prueba: %f" % test_score)
```

```
Red neuronal con (15,10,5) neuronas
Puntaje del conjunto de entrenamiento: 0.887255
Puntaje del conjunto de prueba: 0.879085
```

Sin embargo, tampoco mejora mucho mas. Es interesante el parametro "alpha" que permite evitar el sobre-ajuste del modelo utilizando lo que se conoce como un termino de regularizacion L2. En el modelo creado manualmente, no se implementó la regularizacion de ridge en el cálculo del costo, por lo que tuve que ajustar las iteraciones de entrenamiento para evitarlo. Tenerlo como un parametro lo hace más eficiente y claro a la hora de implementar la red neuronal.

Documentacion de MPLC Classifier de SciKit Learn

Conclusiones

Armar una red neuronal desde cero fue un proceso tanto divertido como tedioso. Me fue muy interesante ver como conceptos teoricos tan disparejos pueden unirse para ser mucho mas que la suma de sus partes. Aprendi cosas nuevas y repasé algunas viejas. La red neuronal solo suma y multiplica pesos, pero las posibilidades son infinitas, y armarla desde cero te da todas las herramientas para poder cambiar cualquiera de los parametros e implementar los algoritmos y capas que quieras, siempre y cuando sepas como hacerlo.

SciKit learn es un utensillo más en la cocina de las redes neuronales, podes usar sus librerias y paquetes en distintos lugares de tu arquitectura para ahorrar tiempo, como pequeños bloques de funcionalidad lista para ser usada, asi como tambien implementar una red neuronal solo con librerias de python sin escribir un solo algoritmo. Sin duda alguna, una gran herramienta para tener en el bolsillo del matemáico o del desarollador.