A partir de las características celulares observadas, ¿el tumor es maligno o benigno?

El cáncer de mama es el cáncer más común entre las mujeres del mundo. Esta base de datos se centra en un conjunto de datos que contiene características clínicas y biométricas de pacientes con cáncer de mama. En este contexto, se utilizan variables como el radio promedio, la textura media y el área de las células, que son fundamentales para determinar si un tumor es benigno o maligno.

En la primera parte analizaremos la base de datos para limpiarla de valores atipicos y de valores faltantes en alguna columna de cada fila, y determinar la correlacion de las variables con la salida esperada.

En la segunda parte armaremos la red neuronal con nuestros datos listos para trabajar y la entrenaremos, luego obtendremos su rendimiento en base a como performa con un subconjunto de la base de datos apartado para testing, para despues graficarla.

Por ultimo, haremos la comparacion de la red neuronal con la libreria Scikit Learn.

Analisis de los datos

Descripcion de las columnas de la base de datos

1. **id (int):** Identificador único del paciente.
2. **diagnosis (string):** Diagnóstico del cáncer (M:maligno y B:benigno).
3. **radius_mean (float):** Promedio del radio de las células tumorales.
4. **texture_mean (float):** Promedio de la textura de las células tumorales.
5. **perimeter_mean (float):** Promedio del perímetro de las células tumorales.
6. **area_mean (float):** Promedio del área ocupada por las células tumorales.
7. **smoothness_mean (float):** Promedio de la suavidad de la superficie de las células tumorales.
8. **compactness_mean (float):** Promedio de la compacidad de las células tumorales.
9. **concavity_mean (float):** Promedio de la concavidad de las células tumorales.
10. **concave points_mean (float):** Promedio de los puntos cóncavos en las células tumorales.
11. **symmetry_mean (float):** Promedio de la simetría de las células tumorales.
12. **fractal_dimension_mean (float):** Promedio de la dimensión fractal de las células tumorales.
13. **radius_se (float):** Error estándar del radio de las células tumorales.
14. **texture_se (float):** Error estándar de la textura de las células tumorales.
15. **perimeter_se (float):** Error estándar del perímetro de las células tumorales.
16. **area_se (float):** Error estándar del área ocupada por las células tumorales.
17. **smoothness_se (float):** Error estándar de la suavidad de las células tumorales.
18. **compactness_se (float):** Error estándar de la compacidad de las células tumorales.
19. **concavity_se (float):** Error estándar de la concavidad de las células tumorales.
20. **concave points_se (float):** Error estándar de los puntos cóncavos en las células tumorales.
21. **symmetry_se (float):** Error estándar de la simetría de las células tumorales.
22. **fractal_dimension_se (float):** Error estándar de la dimensión fractal de las células tumorales.
23. **radius_worst (float):** Peor radio registrado de las células tumorales.
24. **texture_worst (float):** Peor textura registrada de las células tumorales.
25. **perimeter_worst (float):** Peor perímetro registrado de las células tumorales.
26. **area_worst (float):** Peor área ocupada por las células tumorales.
27. **smoothness_worst (float):** Peor suavidad registrada de las células tumorales.
28. **compactness_worst (float):** Peor compacidad registrada de las células tumorales.
29. **concavity_worst (float):** Peor concavidad registrada de las células tumorales.
30. **concave points_worst (float):** Peor cantidad de puntos cóncavos en las células tumorales.
31. **symmetry_worst (float):** Peor simetría registrada de las células tumorales.
32. **fractal_dimension_worst (float):** Peor dimensión fractal de las células tumorales.

¿Que tipo de variable son?

1. Categórica: diagnosis.
2. Continuas: Todas las demás variables.

Librerias a usar

In [1]:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
```

Guardo en una variable el df e Imprimo base de datos

In [3]:

```
df = pd.read_csv("cancer_de_mama.csv", index_col=0)
df
```

Out[3]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst
842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380	17.33
842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990	23.41
84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570	25.53
84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910	26.50
84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540	16.67
...	...	...	...	...	...	...	...	...	...	...	...	...	...
926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450	26.40
926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690	38.25
926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980	34.12
927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740	39.42
92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456	30.37

569 rows × 31 columns

imprimo las primeras 5 filas

In [54]:

```
df.head()
```

Out[54]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

imprimo todas las columnas con las que vamos a trabajar

Cabe aclarar que en este análisis no se tuvo en cuenta la columna ID, ya que se considera irrelevante para la evaluación de las características predictivas.

In [64]:

```
df.columns
```

Out[64]:

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

identificamos los tipos de datos de cada colomna

In [4]:

```
df.dtypes
```

Out[4]:

```
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

Como nuestra columna objetivo es diagnosis, reemplazamos la letras por valores numericos M:1 y B:0 e imprimimos las primeras 20 para chequear.

In [5]:

```
# Maligno: Significa que es agresivo, potencialmente peligroso y puede diseminarse a otras partes del cuerpo.
# En el contexto del cáncer de mama, esto implica que las células cancerosas pueden crecer de manera descontrolada
# y causar daños en los tejidos cercanos.

# Benigno: Indica que no es agresivo y no representa un riesgo significativo para la salud.
# En el caso de los tumores benignos en el seno, aunque pueden causar molestias o preocupaciones
# no se diseminan ni invaden otros tejidos.

df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

df.head(20)
```

Out[5]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst
842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.38	17.33
842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.99	23.41
84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.57	25.53
84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.91	26.50
84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.54	16.67
843786	1	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	...	15.47	23.75
844359	1	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	...	22.88	27.66
84458202	1	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	...	17.06	28.14
844981	1	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2350	...	15.49	30.73
84501001	1	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2030	...	15.09	40.68
845636	1	16.02	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	...	19.19	33.88
84610002	1	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	0.1842	...	20.42	27.28
846226	1	19.17	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.11180	0.2397	...	20.96	29.94
846381	1	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	0.1847	...	16.84	27.66
84667401	1	13.73	22.61	93.60	578.3	0.11310	0.22930	0.21280	0.08025	0.2069	...	15.03	32.01
84799002	1	14.54	27.54	96.73	658.8	0.11390	0.15950	0.16390	0.07364	0.2303	...	17.46	37.13
848406	1	14.68	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.05259	0.1586	...	19.07	30.88
84862001	1	16.13	20.68	108.10	798.8	0.11700	0.20220	0.17220	0.10280	0.2164	...	20.96	31.48
849014	1	19.81	22.15	130.00	1260.0	0.09831	0.10270	0.14790	0.09498	0.1582	...	27.32	30.88
8510426	0	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	...	15.11	19.26

20 rows × 31 columns

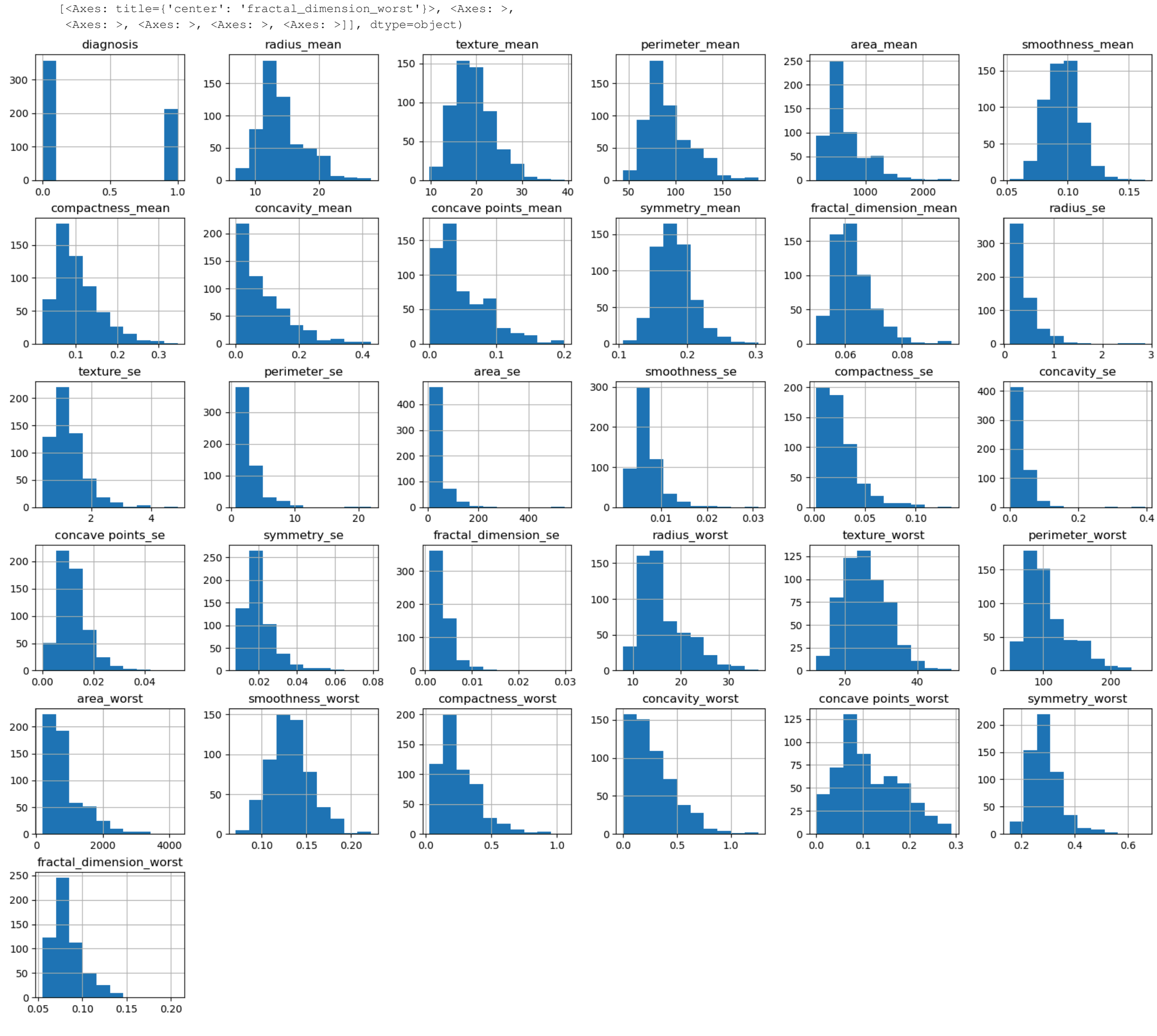
Histograma para visualizar datos atipicos

In [5]:

```
# Limpio el df
df = df.dropna()
df['diagnosis'] = df['diagnosis'].astype(int)

# Muestro histogramas de las variables de entrada para vizualizar valores atípicos
df.hist(figsize=(20, 17))
```

Out[5]:

```
array([[<Axes: title={'center': 'diagnosis'}>,
        <Axes: title={'center': 'radius_mean'}>,
        <Axes: title={'center': 'texture_mean'}>,
        <Axes: title={'center': 'perimeter_mean'}>,
        <Axes: title={'center': 'area_mean'}>,
        <Axes: title={'center': 'smoothness_mean'}>],
       [<Axes: title={'center': 'compactness_mean'}>,
        <Axes: title={'center': 'concavity_mean'}>,
        <Axes: title={'center': 'concave points_mean'}>,
        <Axes: title={'center': 'symmetry_mean'}>,
        <Axes: title={'center': 'fractal_dimension_mean'}>,
        <Axes: title={'center': 'radius_se'}>],
       [<Axes: title={'center': 'texture_se'}>,
        <Axes: title={'center': 'perimeter_se'}>,
        <Axes: title={'center': 'area_se'}>,
        <Axes: title={'center': 'smoothness_se'}>,
        <Axes: title={'center': 'compactness_se'}>,
        <Axes: title={'center': 'concavity_se'}>],
       [<Axes: title={'center': 'concave points_se'}>,
        <Axes: title={'center': 'symmetry_se'}>,
        <Axes: title={'center': 'fractal_dimension_se'}>,
        <Axes: title={'center': 'radius_worst'}>,
        <Axes: title={'center': 'texture_worst'}>,
        <Axes: title={'center': 'perimeter_worst'}>],
       [<Axes: title={'center': 'area_worst'}>,
        <Axes: title={'center': 'smoothness_worst'}>,
        <Axes: title={'center': 'compactness_worst'}>,
        <Axes: title={'center': 'concavity_worst'}>,
        <Axes: title={'center': 'concave points_worst'}>,
        <Axes: title={'center': 'symmetry_worst'}>],
       [<Axes: title={'center': 'fractal_dimension_worst'}>, <Axes: >,
        <Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```



Como observamos una varianza de datos, elegimos trabajarlo por cuartiles, para alejar los valores atípicos.

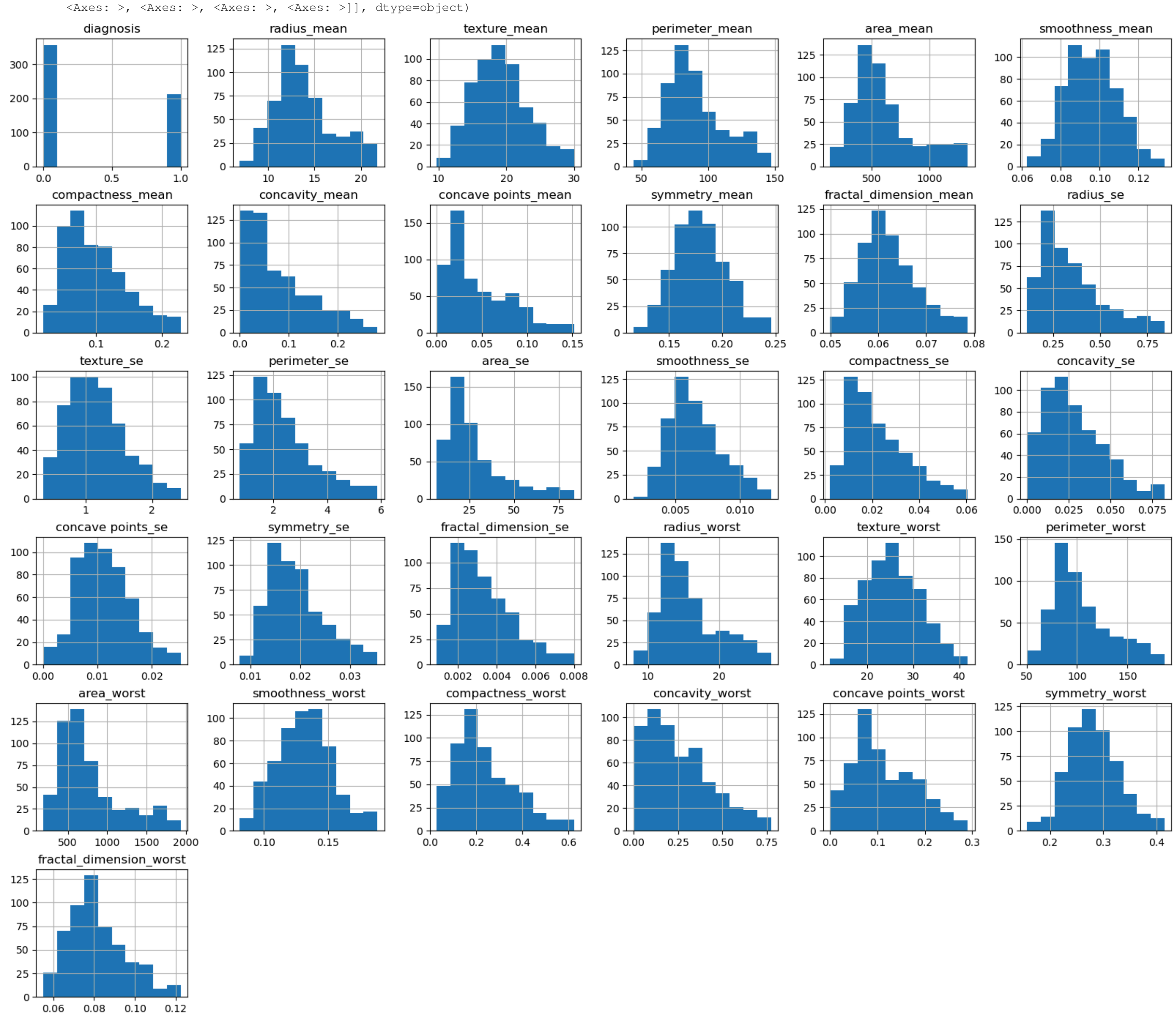
In [72]:

```
df_salida = df['diagnosis']

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

# Identificar valores atípicos utilizando el criterio del IQR
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))

# Contar el número de valores atípicos por fila
num_outliers = outliers.sum()
print("Filas con Valores Atípicos:\n", num_outliers)

# Saco del df las filas con valores atípicos (y reinserto la columna de salida original)
df = df.where(np.invert(outliers))
df['diagnosis'] = df_salida

# Histograma limpio
df.hist(figsize=(20, 17))
```

```
Filas con Valores Atípicos:
 diagnosis                  0
radius_mean               14
texture_mean               7
perimeter_mean            13
area_mean                 25
smoothness_mean            6
compactness_mean          16
concavity_mean            18
concave points_mean       10
symmetry_mean             15
fractal_dimension_mean    15
radius_se                 38
texture_se                20
perimeter_se              38
area_se                   65
smoothness_se             30
compactness_se            28
concavity_se              22
concave points_se         19
symmetry_se               27
fractal_dimension_se      28
radius_worst              17
texture_worst              5
perimeter_worst           15
area_worst                35
smoothness_worst           7
compactness_worst         16
concavity_worst           12
concave points_worst       0
symmetry_worst            23
fractal_dimension_worst   24
dtype: int64
```

Out[72]:

```
array([[<Axes: title={'center': 'diagnosis'}>,
        <Axes: title={'center': 'radius_mean'}>,
        <Axes: title={'center': 'texture_mean'}>,
        <Axes: title={'center': 'perimeter_mean'}>,
        <Axes: title={'center': 'area_mean'}>,
        <Axes: title={'center': 'smoothness_mean'}>],
       [<Axes: title={'center': 'compactness_mean'}>,
        <Axes: title={'center': 'concavity_mean'}>,
        <Axes: title={'center': 'concave points_mean'}>,
        <Axes: title={'center': 'symmetry_mean'}>,
        <Axes: title={'center': 'fractal_dimension_mean'}>,
        <Axes: title={'center': 'radius_se'}>],
       [<Axes: title={'center': 'texture_se'}>,
        <Axes: title={'center': 'perimeter_se'}>,
        <Axes: title={'center': 'area_se'}>,
        <Axes: title={'center': 'smoothness_se'}>,
        <Axes: title={'center': 'compactness_se'}>,
        <Axes: title={'center': 'concavity_se'}>],
       [<Axes: title={'center': 'concave points_se'}>,
        <Axes: title={'center': 'symmetry_se'}>,
        <Axes: title={'center': 'fractal_dimension_se'}>,
        <Axes: title={'center': 'radius_worst'}>,
        <Axes: title={'center': 'texture_worst'}>,
        <Axes: title={'center': 'perimeter_worst'}>],
       [<Axes: title={'center': 'area_worst'}>,
        <Axes: title={'center': 'smoothness_worst'}>,
        <Axes: title={'center': 'compactness_worst'}>,
        <Axes: title={'center': 'concavity_worst'}>,
        <Axes: title={'center': 'concave points_worst'}>,
        <Axes: title={'center': 'symmetry_worst'}>],
       [<Axes: title={'center': 'fractal_dimension_worst'}>, <Axes: >,
        <Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```



Como podemos observar en los histogramas, la distribución de datos se hizo mas amigable. Los valores atípicos generan un desvalance en la distribución general de los datos.

Ahora realizaremos una estadística rápida de los datos normalizados.

In [6]:

```
# describe() genera una tabla que incluye estadísticas como la media, la desviación estándar, los cuartiles
# el valor mínimo y el máximo para cada columna numérica.
# El .T al final transpuesta la tabla, convirtiendo las filas en columnas y viceversa.
df_stats = df.describe().T

# Para normalizar: (valor - promedio) / desv_estandar
df_n = (df - df_stats['mean']) / df_stats['std']

# genera estadísticas descriptivas para el df
df_n.describe()
```

Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	...	5.690000e+02
mean	2.497514e-17	-1.311195e-16	6.243785e-17	-1.123881e-16	-2.185325e-16	-8.366672e-16	1.873136e-16	2.497514e-17	-4.995028e-17	1.748260e-16	...	-8.241796e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	...	1.000000e+00
min	-7.699311e-01	-2.027864e+00	-2.227289e+00	-1.982759e+00	-1.453164e+00	-3.109349e+00	-1.608721e+00	-1.113893e+00	-1.260710e+00	-2.741705e+00	...	-1.725382e+00
25%	-7.699311e-01	-6.887793e-01	-7.253249e-01	-6.913472e-01	-6.666089e-01	-7.103378e-01	-7.464292e-01	-7.430941e-01	-7.372951e-01	-7.026215e-01	...	-6.743279e-01
50%	-7.699311e-01	-2.148925e-01	-1.045442e-01	-2.357726e-01	-2.949274e-01	-3.486040e-02	-2.217454e-01	-3.419391e-01	-3.973715e-01	-7.156354e-02	...	-2.688030e-01
75%	1.296535e+00	4.689800e-01	5.836621e-01	4.992377e-01	3.631877e-01	6.356397e-01	4.934227e-01	5.255994e-01	6.463664e-01	5.303125e-01	...	5.215568e-01
max	1.296535e+00	3.967796e+00	4.647799e+00	3.972634e+00	5.245913e+00	4.766717e+00	4.564409e+00	4.239858e+00	3.924477e+00	4.480808e+00	...	4.090590e+00

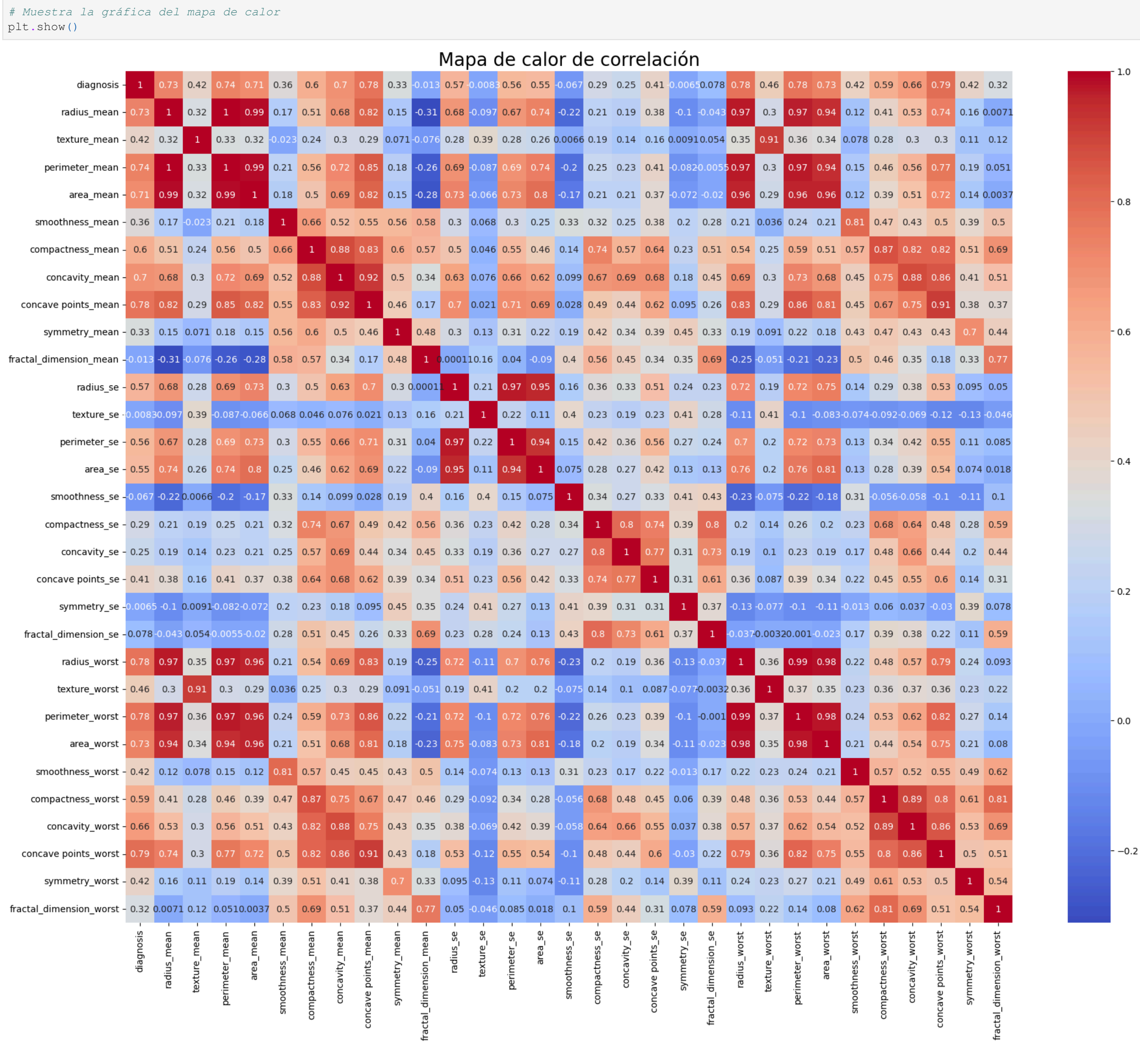
8 rows × 31 columns

Estas transformaciones nos ayudan a estandarizar los datos para que sean comparables en escala y asegurar que el data frame esté listo para usarse.

Ahora, Graficaremos las correalaciones

In [16]:

```
# Eliminar filas con valores nulos
df = df.dropna()

# creo otro arhivo con lo datos limpios
df.to_csv('cancer_de_mama_limpio.csv')

# creo el grafico y el tamaño
plt.figure(figsize=(22,17))

# Calcula la matriz de correlación de las variables en el df.
corr = df.corr()

# Crea un mapa de calor que visualiza la matriz de correlación
sns.heatmap(corr, annot=True, cmap="coolwarm")

# titulo
plt.title('Mapa de calor de correlación', fontsize=20)

# Muestra la gráfica del mapa de calor
plt.show()
```

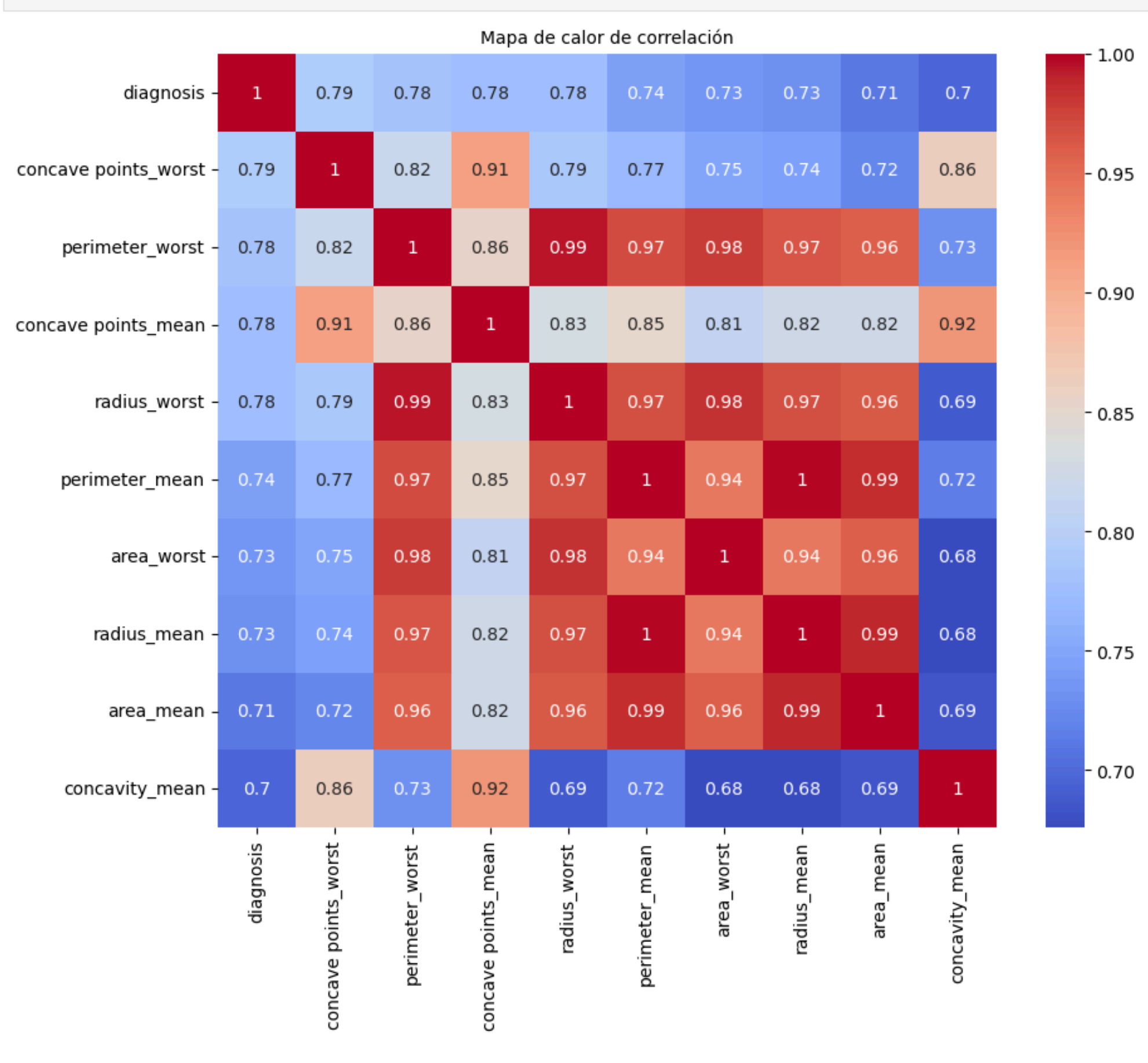


Al analizar la matriz de correlación, se observa que las variables relacionadas con las características celulares, como el radio promedio y el perímetro promedio, presentan una fuerte correlación positiva con la variable de diagnóstico (diagnosis). Por otro lado, variables como el área de la célula y la suavidad muestran una correlación inversa significativa.

Son muchas columnas, al trabajar con todas nos podria causar un overffiting. Para eso seleccionas las 9 columnas que mas correlacion tienen con nuestra columna objetivo y creamos otro archivo llamado cancer_de_mama_limpio_top10 para guardarlas

In [18]:

```
# Guardo el df en una variable
df2 = pd.read_csv("cancer_de_mama_limpio.csv", index_col=0)

# Definir la variable de salida
variable_salida = "diagnosis"

# Calcular la matriz de correlación
correlacion = df2.corr()

# Seleccionar las 10 columnas con mayor correlación (en valor absoluto) con la variable de salida
columnas_top10 = correlacion[variable_salida].abs().sort_values(ascending=False).head(10).index

# Crear un nuevo DataFrame solo con las columnas relevantes
df_top10 = df2[columnas_top10]

# creo el grafico y el tamaño
plt.figure(figsize=(10, 8))

# Crea un mapa de calor que visualiza la matriz de correlación
sns.heatmap(df_top10.corr(), annot=True, cmap="coolwarm")

# titulo
plt.title('Mapa de calor de correlación', fontsize=10)

# Muestra la gráfica del mapa de calor
plt.show()

# creo nuevo archivo
df_top10.to_csv("cancer_de_mama_limpio_top10")
```



nuestras columnas seleccionas son:

1. **concavity_mean**
2. **area_mean**
3. **radius_mean**
4. **area_worst**
5. **perimeter_mean**
6. **radius_worst**
7. **concave points_mean**
8. **perimeter_worst**
9. **concave points_worst**

Despues de un largo analisis. podemos concluír que nuestra base de datos es adecuada para entrenar una red neuronal, para determinar si un tumor es maligno o begnino.

Red neuronal

La red tendrá dos capas: una capa oculta y una capa de salida.

¿Cuántas neuronas habrá en cada capa?

- Capa de entrada: 9 neuronas
- Capa oculta: 3 neuronas
- Capa de salida: 1 neurona

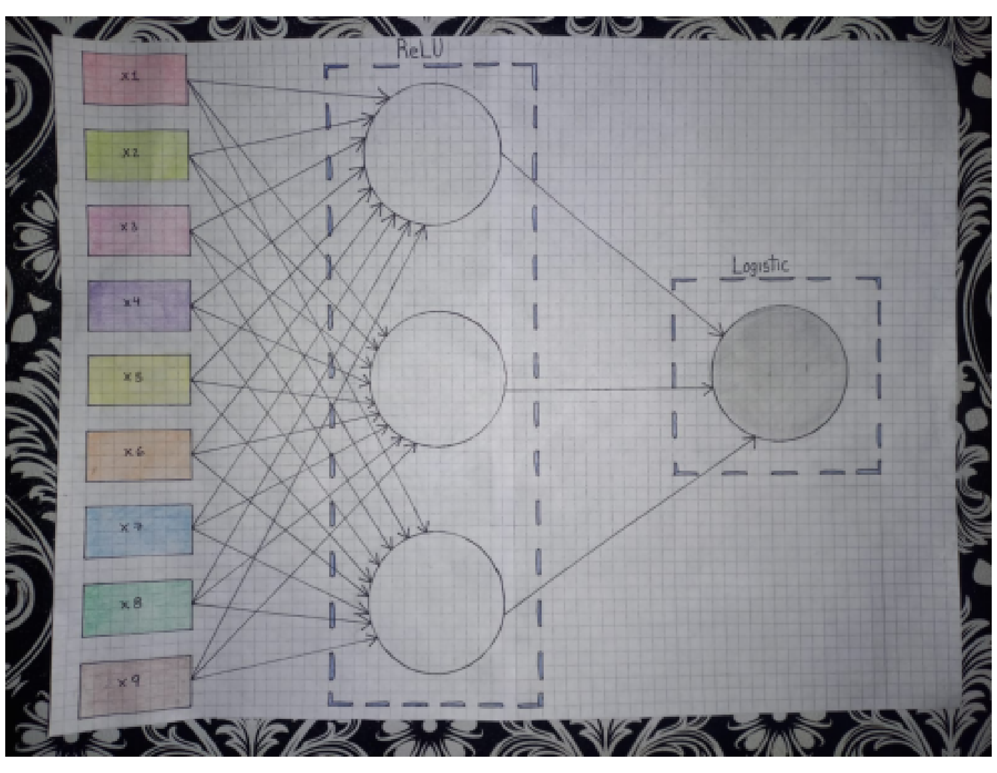
La capa oculta utilizara la función ReLu y la capa de salida logistic.

- ReLu: deja igual los valores de entrada positivos y = 0 y a los negativos = X si X > 0 y 0 si X < 0
- Logistic: transforma la salida en un valor entre 0 y 1, ideal para clasificación binaria.

Dibujo de la arquitectura de la red neuronal

In [2]:

```python
from PIL import Image
import matplotlib.pyplot as plt

# Cargar la imagen
imagen = Image.open('dibujo_red_neuronal.JPEG')

# Mostrar la imagen
plt.imshow(imagen)
plt.axis('off')
plt.show()
```



Red neuronal Libreria

In [116…

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

# Cargar el archivo limpio de cáncer de mama
df = pd.read_csv('cancer_de_mama_limpio_top10.csv')

# Extraer variables de entrada (todas las filas, columnas de 2 a 10)
X = df.iloc[:, 2:11].values

# Extraer columna de salida (todas las filas, columna 'diagnosis')
Y = df.iloc[:, 1].values

# Normalizar las características
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_state=42)

# Crear la red neuronal
nn = MLPClassifier(solver='adam',
                   hidden_layer_sizes=(3, ),  # Capas ocultas
                   activation='relu',  # Función de activación
                   max_iter=150_000,  # Máximo de iteraciones
                   learning_rate_init=0.01)  # Tasa de aprendizaje inicial

# Entrenamiento
nn.fit(X_train, Y_train)

# Evaluación del modelo
print("Porcentaje de aciertos con test: ", (nn.score(X_test, Y_test) * 100))
print("Porcentaje de aciertos con train: ", (nn.score(X_train, Y_train) * 100))
```

```
Porcentaje de aciertos con test:  95.78947368421052
Porcentaje de aciertos con train:  96.04221635883906
```

Red neuronal hecha a mano

In [4]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Guarda el archivo en una variable
df2 = pd.read_csv('cancer_de_mama_limpio_top10.csv')

# Extraer variables de entrada (todas las filas, columnas de 2 a 10)
X = df2.iloc[:, 2:11].values

# Extraer columna de salida (todas las filas, columna 'diagnosis')
Y = df2.iloc[:, 1].values

# Normalizar las características con StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Dividir en un conjunto de entrenamiento y uno de prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3)

# Número de registros de entrenamiento
n = X_train.shape[0]
print('Cantidad de filas de entrenamiento: ', n)

# Funciones de activación
relu = lambda x: np.maximum(x,0)
logistic = lambda x: 1 / (1 + np.exp(-x))

# Semilla para reproducibilidad
np.random.seed(2)

# Construir red neuronal con pesos y sesgos
# inicializados aleatoriamente
w_hidden = (np.random.rand(3, 9) * 2) - 1
w_output = (np.random.rand(1, 3) * 2) - 1

b_hidden = (np.random.rand(3, 1) * 2) - 1
b_output = (np.random.rand(1, 1) * 2) - 1

# Funcion que corre la red neuronal con los datos de entrada para predecir la salida
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2

# Cálculo de precisión
def precision(X, Y):
    test_predictions = forward_prop(X.transpose())[3] # me interesa solo la capa de salida, A2
    test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y)
    accuracy = sum(test_comparisons.astype(int) / X.shape[0])
    print("Porcentaje de aciertos: ", (accuracy*100).round(2))

print('Pre entrenamiento: \n')
print('Test')
precision(X_test, Y_test)
print('Train')
precision(X_train, Y_train)

# Tasa de aprendizaje
L = 0.01

# Derivadas de las funciones de activación
#d_leaky_relu = lambda x: np.where(x > 0, x, 0.01)
d_relu = lambda x: x > 0
d_logistic = lambda x: np.exp(-x) / (1 + np.exp(-x)) ** 2

# Devuelve pendientes para pesos y sesgos
# usando la regla de la cadena
def backward_prop(Z1, A1, Z2, A2, X, Y):
    dC_dA2 = 2 * A2 - 2 * Y
    dA2_dZ2 = d_logistic(Z2)
    dZ2_dA1 = w_output
    dZ2_dW2 = A1
    dZ2_dB2 = 1
    dA1_dZ1 = d_relu(Z1)
    dZ1_dW1 = X
    dZ1_dB1 = 1

    dC_dW2 = dC_dA2 @ dA2_dZ2 @ dZ2_dW2.T

    dC_dB2 = dC_dA2 @ dA2_dZ2 * dZ2_dB2

    dC_dA1 = dC_dA2 @ dA2_dZ2 @ dZ2_dA1

    dC_dW1 = dC_dA1 @ dA1_dZ1 @ dZ1_dW1.T

    dC_dB1 = dC_dA1 @ dA1_dZ1 * dZ1_dB1

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2


# Ejecutar descenso de gradiente
for i in range(150_000):
    # seleccionar aleatoriamente uno de los datos de entrenamiento
    idx = np.random.choice(n, 1, replace=False)
    X_sample = X_train[idx].transpose()
    Y_sample = Y_train[idx]

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

    # distribuir error a través de la retropropagación
    # y devolver pendientes para pesos y sesgos
    dW1, dB1, dW2, dB2 = backward_prop(Z1, A1, Z2, A2, X_sample, Y_sample)

    # actualizar pesos y sesgos
    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_output -= L * dW2
    b_output -= L * dB2

# Cálculo de precisión post-entrenamiento
print('Post entrenamiento: \n')
print('Test')
precision(X_test, Y_test)
print('Train')
precision(X_train, Y_train)
```

```
Cantidad de filas de entrenamiento:  379
Pre entrenamiento:

Test
Porcentaje de aciertos:  65.26
Train
Porcentaje de aciertos:  61.48
Post entrenamiento:

Test
Porcentaje de aciertos:  96.84
Train
Porcentaje de aciertos:  96.57
```

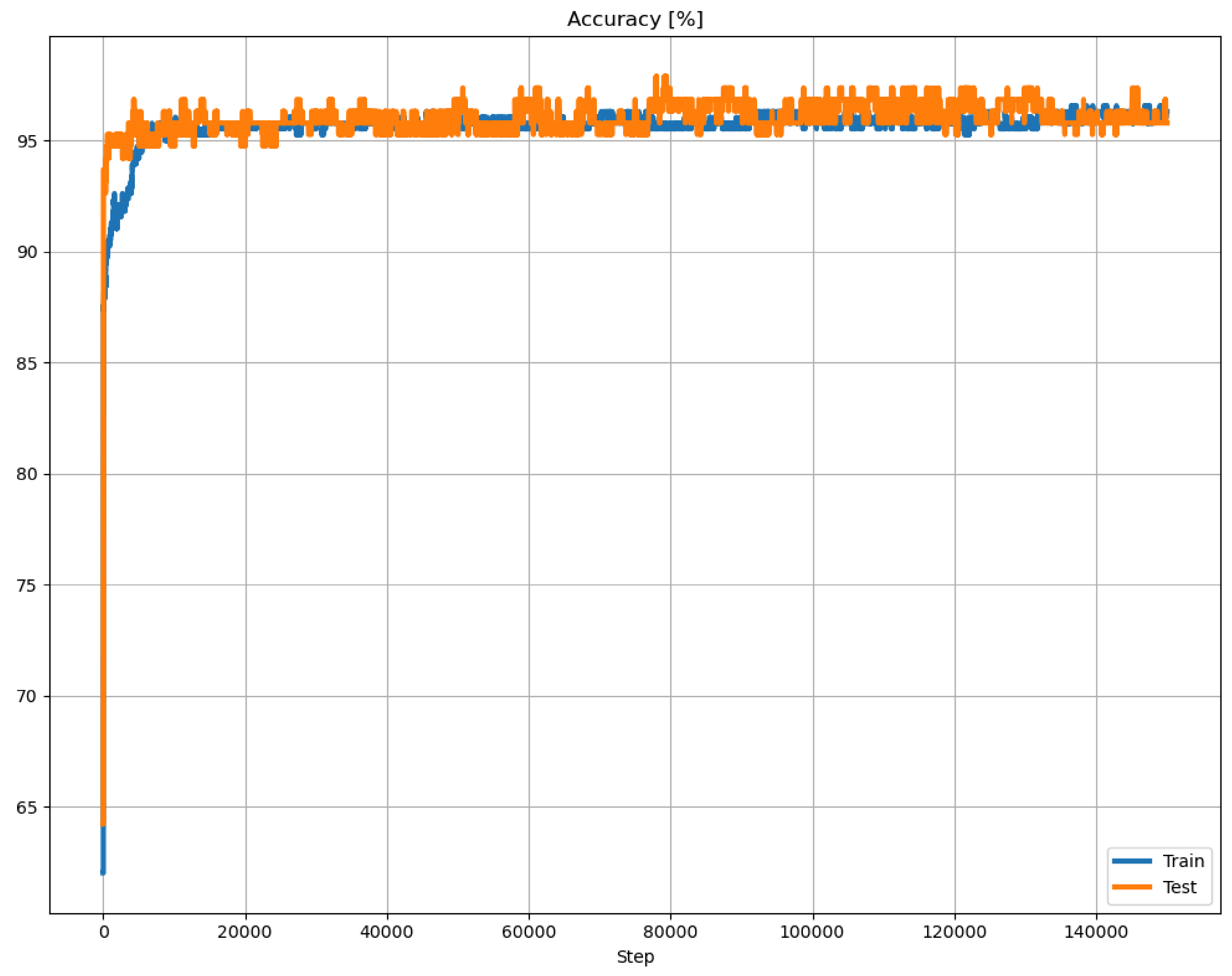
Comparación con scikit-learn

La red neuronal manual y la de scikit-learn mostraron un rendimiento muy similar, con una precisión del 95-96% en ambos conjuntos de Test y Train.

In [33]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm.auto import tqdm

# Cargar el archivo en una variable
df2 = pd.read_csv('cancer_de_mama_limpio_top10.csv')

# Extraer variables de entrada (todas las filas, columnas de 2 a 10)
X = df2.iloc[:, 2:11].values

# Extraer columna de salida (todas las filas, columna 'diagnosis')
Y = df2.iloc[:, 1].values

# Normalizar las características con StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Dividir en un conjunto de entrenamiento y uno de prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3, random_state=42)

# Número de registros de entrenamiento
n = X_train.shape[0]
print('Cantidad de filas de entrenamiento: ', n)

# Funciones de activación
relu = lambda x: np.maximum(x, 0)
logistic = lambda x: 1 / (1 + np.exp(-x))

# Semilla para reproducibilidad
np.random.seed(2)

# Construir red neuronal con pesos y sesgos inicializados aleatoriamente
w_hidden = (np.random.rand(3, 9) * 2) - 1
w_output = (np.random.rand(1, 3) * 2) - 1
b_hidden = (np.random.rand(3, 1) * 2) - 1
b_output = (np.random.rand(1, 1) * 2) - 1

# Funcion que corre la red neuronal con los datos de entrada para predecir la salida
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2

# Tasa de aprendizaje
L = 0.01

# Derivadas de las funciones de activación
d_leaky_relu = lambda x: x > 0
d_logistic = lambda x: np.exp(-x) / (1 + np.exp(-x)) ** 2

# Devuelve pendientes para pesos y sesgos usando la regla de la cadena
def backward_prop(Z1, A1, Z2, A2, X, Y):
    dC_dA2 = 2 * A2 - 2 * Y
    dA2_dZ2 = d_logistic(Z2)
    dZ2_dA1 = w_output
    dZ2_dW2 = A1
    dZ2_dB2 = 1
    dA1_dZ1 = d_relu(Z1)
    dZ1_dW1 = X
    dZ1_dB1 = 1

    dC_dW2 = dC_dA2 @ dA2_dZ2 @ dZ2_dW2.T
    dC_dB2 = dC_dA2 @ dA2_dZ2 * dZ2_dB2
    dC_dA1 = dC_dA2 @ dA2_dZ2 @ dZ2_dA1
    dC_dW1 = dC_dA1 @ dA1_dZ1 @ dZ1_dW1.T
    dC_dB1 = dC_dA1 @ dA1_dZ1 * dZ1_dB1

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2

# Listas para almacenar la precisión
accuracy_train_l = []
accuracy_test_l = []

# Ejecutar descenso de gradiente
for i in tqdm(range(150_000)):
    # Seleccionar aleatoriamente un registro de entrenamiento
    idx = np.random.choice(n, 1, replace=False)
    X_sample = X_train[idx].transpose()
    Y_sample = Y_train[idx]

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

    # Retropropagación y cálculo de pendientes para pesos y sesgos
    dW1, dB1, dW2, dB2 = backward_prop(Z1, A1, Z2, A2, X_sample, Y_sample)

    # Actualizar pesos y sesgos
    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_output -= L * dW2
    b_output -= L * dB2

    # Cálculo de precisión en el conjunto de prueba
    test_predictions = forward_prop(X_test.transpose())[3]
    test_comparisons = np.equal((test_predictions >= .5).flatten().astype(int), Y_test)
    accuracy_test = np.mean(test_comparisons) * 100
    accuracy_test_l.append(accuracy_test)

    # Cálculo de precisión en el conjunto de entrenamiento
    train_predictions = forward_prop(X_train.transpose())[3]
    train_comparisons = np.equal((train_predictions >= .5).flatten().astype(int), Y_train)
    accuracy_train = np.mean(train_comparisons) * 100
    accuracy_train_l.append(accuracy_train)

# Imprimir la precisión final
print("X_test ACCURACY post-training: ", accuracy_test_l[-1])
print("X_train ACCURACY post-training: ", accuracy_train_l[-1])

# Graficar la precisión
fmt_train = {
    'color': 'tab:blue',
    'ls': 'solid',
    'lw': 3,
}

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

fig, ax = plt.subplots(1, 1, figsize=(10, 8))

ax.plot(accuracy_train_l, label='Train', **fmt_train)
ax.plot(accuracy_test_l, label='Test', **fmt_test)


ax.grid(which='both')
ax.legend()
ax.set_title('Accuracy [%]')
ax.set_xlabel('Step')

fig.tight_layout()
plt.show()
```

```
Cantidad de filas de entrenamiento:  379
  0%|          | 0/150000 [00:00<?, ?it/s]
X_test ACCURACY post-training:  95.78947368421052
X_train ACCURACY post-training:  96.30606860158312
```



Luego de desarrollar nuestra Red Neuronal, viendo el porcentaje de aciertos y el gráfico, ¿nuestro modelo presenta overfitting?.

El overfitting es cuando una red una de ajusta demasiado bien a los datos de entrenamiento. Una forma de detectar el sobreajuste es observando si el porcentaje de aciertos en el conjunto de entrenamiento es significativamente mayor que en el conjunto de prueba, especialmente si esta diferencia se amplía al aumentar las épocas de entrenamiento. También es indicativo de sobreajuste si el rendimiento en el conjunto de prueba se estabiliza mientras que el del entrenamiento continúa mejorando.

Existen varias estrategias para prevenir o detectar el sobreajuste en redes neuronales:

- Dropout: Esta técnica consiste en desactivar aleatoriamente algunas neuronas durante el entrenamiento, lo que ayuda a evitar la dependencia de neuronas específicas.
- Reducir la complejidad de la red neuronal: Una forma de lograrlo es disminuir el número de neuronas en las capas ocultas, lo que impide que la red aprenda en exceso sobre datos específicos.
- Ajustar el número de épocas: Limitar la cantidad de iteraciones de entrenamiento puede ayudar a detener el proceso antes de que la red continúe entrenándose sin mejoras en el conjunto de prueba.
- Aumentar la cantidad de datos: Contar con más muestras puede disminuir la probabilidad de sobreajuste, ya que se facilita la generalización.

Conclusión Final

Desarrollar una red neuronal desde cero nos permitió entender en profundidad cómo funcionan las redes neuronales en cada paso del proceso, desde la inicialización de pesos y sesgos hasta el forward y el backward. Al construir manualmente cada función de activación y sus derivadas, adquirí una comprensión mucho más detallada de cómo los gradientes afectan el ajuste de los pesos y de cómo se optimizan los modelos de aprendizaje. Comparado con el uso la librería de scikit-learn, crear una red neuronal manualmente ofrece ventajas significativas en términos de aprendizaje. Trabajar desde cero ayuda a entender mejor las matemáticas y la lógica detrás del entrenamiento de los modelos. La principal desventaja de este enfoque es que resulta mucho más laborioso y es muy propenso a errores, ya que requiere gestionar manualmente cada cálculo y ajuste de parámetros.