

# Computación neuronal y evolutiva

## Práctica nº1



# UNIVERSIDAD DE BURGOS

Por: Adrián Zamora Sánchez

# Índice

<b>Motivación</b>	<b>3</b>
<b>Descripción de los datos y el problema</b>	<b>3</b>
<b>Comparativa de los resultados</b>	<b>6</b>
Con pocas neuronas (3,5)	7
Con un número intermedio de neuronas (10)	7
Con muchas neuronas (60+)	7
<b>Selección del mejor resultado</b>	<b>7</b>
<b>Etapas de explotación</b>	<b>8</b>

# Motivación

Mi elección sobre los datos relacionados con el estudio del porcentaje de grasa surge de mi afición al deporte y mi interés por la anatomía. Hace unos días vi un [video en youtube](#) en el cual un doctor explicaba el funcionamiento del Índice de Masa Corporal (IMC) y su uso en el ámbito médico para estimar la composición corporal, en especial el porcentaje de grasa. A pesar de su amplia utilización, el IMC, fue desarrollado por un matemático en 1830, presenta limitaciones significativas, ya que no contempla factores como la masa muscular o la distribución de grasa, lo que puede llevar a estimaciones erróneas en ciertos individuos.

Este método, aunque es útil en su contexto, ha sido cuestionado por expertos debido a su incapacidad para adaptarse a la diversidad corporal. Hoy en día, con el avance de la tecnología, creo que es posible superar estas limitaciones mediante el uso de inteligencia artificial, específicamente redes neuronales. Mi interés en este proyecto radica en explorar cómo, utilizando algunas medidas corporales que gracias a la medicina moderna son fáciles de conseguir, se puede desarrollar un modelo capaz de estimar con gran precisión el porcentaje de grasa corporal, ofreciendo así una herramienta más personalizada y precisa que la actual basada en el IMC.

En el siguiente enlace a mi [GitHub](#) se podrá consultar el código para la generación de las redes neuronales y el código con el que he llevado a cabo la explotación de la red seleccionada. Además en el repositorio se podrán encontrar los archivos csv con los datos utilizados.

## Descripción de los datos y el problema

Para conseguir estimar el porcentaje de grasa vamos a entrenar a la red neuronal con un set de datos que he encontrado en el siguiente [enlace](#) de GitHub, este set de datos contiene múltiples datos, todos relacionados con la composición corporal y el porcentaje de grasa. En la siguiente imagen se pueden apreciar el conjunto de datos cargado en MATLAB.

	IDNO	BODYFAT	DENSITY	AGE	WEIGHT	HEIGHT	ADIPOSI	NECK	CHEST	ABDOMEN	HIP	THIGH	KNEE	ANKLE	
1	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	Number	N
2	1	12.6	1.0708	23	154.25	67.75	23.7	36.2	93.1	85.2	94.5	59	37.3	21.9	3
3	2	6.9	1.0853	22	173.25	72.25	23.4	38.5	93.6	83	98.7	58.7	37.3	23.4	3
4	3	24.6	1.0414	22	154	66.25	24.7	34	95.8	87.9	99.2	59.6	38.9	24	2
5	4	10.9	1.0751	26	184.75	72.25	24.9	37.4	101.8	86.4	101.2	60.1	37.3	22.8	3
6	5	27.8	1.034	24	184.25	71.25	25.6	34.4	97.3	100	101.9	63.2	42.2	24	3
7	6	20.6	1.0502	24	210.25	74.75	26.5	39	104.5	94.4	107.8	66	42	25.6	3
8	7	19	1.0549	26	181	69.75	26.2	36.4	105.1	90.7	100.3	58.4	38.3	22.9	3
9	8	12.8	1.0704	25	176	72.5	23.6	37.8	99.6	88.5	97.1	60	39.4	23.2	3
0	9	5.1	1.09	25	191	74	24.6	38.1	100.9	82.5	99.9	62.9	38.3	23.8	3
1	10	12	1.0722	23	198.25	73.5	25.8	42.1	99.6	88.6	104.1	63.1	41.7	25	3
2	11	7.5	1.083	26	186.25	74.5	23.6	38.5	101.5	83.6	98.2	59.7	39.7	25.2	3
3	12	8.5	1.0812	27	216	76	26.3	39.4	103.6	90.9	107.7	66.2	39.2	25.9	3

De estos datos, no se utiliza para nada la primera columna, que contiene identificadores de los sujetos, la segunda columna será el resultado con el que se compara, también llamados vector objetivo y ajusta la red neuronal durante el entrenamiento, el resto de columnas contiene las mediciones de diferentes partes del cuerpo, con ellas se realiza el ajuste de la función que se entrena, estos son los vectores de entrada.

Se trata de un problema de ajuste de la función que se puede representar matemáticamente como  $f(x) = y$  donde “f()” es la función a ajustar, “x” es el conjunto de datos corporales de un individuo con el que se realiza la predicción y finalmente obtendremos “y”, que será la estimación del porcentaje de grasa del individuo.

Para realizar el entrenamiento se utilizarán cuatro configuraciones de neuronas en la capa oculta, he escogido probar con 3, 5, 15 y 60 neuronas, estas se prueban mediante un bucle que realiza el entrenamiento para cada configuración de la red.

Dentro del bucle, se establece la función de entrenamiento “trainlm”, la cual es recomendada para ajuste de funciones en modelos sencillos, aunque esta función es con la que he realizado el trabajo principalmente, he probado y comparado otras funciones como trainbr, la cual también ha dado grandes resultados.

Se dividen los datos con los que se entrena la red de forma aleatoria siguiendo una configuración donde el 70% son para entrenamiento, el 15% para validación y el 15% restante se utiliza para realizar pruebas. Estos ajustes se realizan en el siguiente código:

```
% Función de entrenamiento
trainFcn = 'trainlm';

% Crear y configurar la red neuronal
net = fitnet(num_neurons, trainFcn);

% División aleatoria de los datos
net.divideParam.trainRatio = 0.7;
net.divideParam.valRatio = 0.15;
net.divideParam.testRatio = 0.15;
```

La medición del tiempo que tarda la red en ser entrenada dependerá de muchos parámetros, como el número de neuronas, el tamaño de set de datos y la función con la que se entrena la red. El tiempo que tarda en entrenar es obtenido mediante el siguiente código:

```
% Se inicia el cronometro, se entrena la red y se para el cronometro,
% obteniendo el tiempo de entrenamiento
tic;
[net, tr] = train(net, X, Y); % Se entrena la red
train_time = toc;

% Almacena el valor del tiempo de entrenamiento
train_times(i) = train_time;
```

Para comprobar la validez de los resultados, he escogido un límite en el error del 2.5%, puesto que para la mayoría de aplicaciones clínicas esta aproximación es más que suficiente, además, comparado con el 1% de error que tiene la herramienta más precisa en la actualidad, el escáner DEXA, considero que el error escogido es más que válido.

A continuación una captura que muestra la información obtenida de aciertos, errores y el % de errores que muestra el programa para cada configuración de la red neuronal:

```
>> bodyfat
Con 3 neuronas: MSE: 0.3920, Aciertos: 38, Fallos: 0, Fallo del 0.00%, Tiempo de entrenamiento: 0.67 segundos
Con 5 neuronas: MSE: 0.8350, Aciertos: 37, Fallos: 1, Fallo del 2.63%, Tiempo de entrenamiento: 0.42 segundos
Con 15 neuronas: MSE: 8.3429, Aciertos: 36, Fallos: 2, Fallo del 5.26%, Tiempo de entrenamiento: 0.42 segundos
Con 60 neuronas: MSE: 2.3424, Aciertos: 34, Fallos: 4, Fallo del 10.53%, Tiempo de entrenamiento: 0.58 segundos
```

Al terminar el entrenamiento se toman los datos de las pruebas para con estos datos calcular el error en las estimaciones del porcentaje de grasa.

```
% Obtener los índices del conjunto de prueba
testInd = tr.testInd;
```

```
% Predecir en el conjunto de prueba
Y_pred = net(X(:, testInd));
Y_test = Y(testInd);
```

Para medir el error se utiliza el error cuadrático medio (MSE), el cual es muy sensible a errores grandes, es decir, errores que se alejan mucho del valor esperado. Los resultados al igual que con el tiempo, se almacenan en un vector el resultado obtenido.

```
% Calcular el MSE y lo almacena
mse_error = mse(Y_test - Y_pred);
results(i) = mse_error;
```

Estos datos son especialmente relevantes para entender el éxito que ha tenido el modelo que se ha entrenado, por eso estos datos son mostrados al final del programa para poder hacer una comparativa de todos los modelos.

```
% Mostrar el resumen de los resultados del MSE y el tiempo de entrenamiento
disp('Resultados de MSE para cada configuración de neuronas:');
disp(results);
disp('Tiempos de entrenamiento para cada configuración de neuronas:');
disp(train_times);
```

En los siguientes apartados se explican los resultados obtenidos con cada configuración de la red neuronal y utilizaremos estos datos para establecer una comparativa con la que poder elegir la mejor opción a la hora de estimar el porcentaje de grasa.

# Comparativa de los resultados

En primer lugar, los resultados a evaluar son los obtenidos al utilizar el algoritmo "trainlm", que es la implementación por MATLAB del algoritmo de retropropagación de Levenberg-Marquardt. La importancia del algoritmo utilizado en el entrenamiento es especialmente importante, como podemos ver en las siguientes capturas de resultados obtenidos con 3, 5, 10 y 60 neuronas en orden descendente.

Tiempos para **trainbr**:

```
Tiempos de entrenamiento para cada configuración de neuronas:  
0.7207  
0.6262  
5.1320  
32.8962
```

MSE para **trainbr**:

```
Resultados de MSE para cada configuración de neuronas:  
0.0924  
6.1307  
8.2760  
0.2257
```

Tiempos para **trainlm**:

```
Tiempos de entrenamiento para cada configuración de neuronas:  
0.4292  
0.3940  
0.4102  
0.5745
```

MSE para **trainlm**:

```
Resultados de MSE para cada configuración de neuronas:  
2.1716  
5.2531  
2.2759  
7.1710
```

Como podemos observar, el resultado de trainlm son menores tiempos de entrenamiento con mejores resultados para número reducido de neuronas y en el caso de trainbr un mayor número de neuronas obtiene buenos resultados, pero a costa de un muy elevado tiempo de entrenamiento.

A continuación analizaremos los resultados en mayor profundidad para **trainlm**, explicando la causa de su mejor o peor rendimiento según el número de neuronas utilizado.

## Con pocas neuronas (3,5)

Este es el modelo que ha tenido un menor MSE y dado que su tasa de fallos es la menor con respecto a las otras configuraciones de redes neuronales, es el aparentemente más adecuado en este contexto, pues el problema a resolver es relativamente sencillo, además, el número de datos utilizados para el entrenamiento es bajo, lo cual beneficia a esta red más sencilla.

## Con un número intermedio de neuronas (10)

La red con 10 neuronas está obteniendo peores resultados que los modelos con menor número de neuronas, sin embargo, los resultados se mantienen aceptables, posiblemente el mayor MSE se debe a que en este caso se da un sobreajuste también llamado “overfitting”, el cual lleva a que el modelo se ajuste demasiado a los datos específicos del conjunto de entrenamiento, en lugar de generalizar bien a nuevos datos.

## Con muchas neuronas (60+)

En este caso, la red de mayor tamaño ha obtenido los peores resultados tanto en términos de rendimiento como de eficiencia. No solo tarda más en entrenarse, sino que su MSE es considerablemente más alto que el de las redes más pequeñas, lo que indica un claro problema de sobreajuste. Esto se observa en las capturas comparativas entre los algoritmos `trainlm` y `trainbr`, donde el ajuste en la retropropagación resulta en un entrenamiento más exitoso.

Otro posible factor que penaliza a estas redes es el reducido número de muestras de entrenamiento, puesto que la red tiene capacidad para asimilar gran cantidad de datos con relaciones complejas entre ellos y el set de datos contiene pocos datos con una relación bastante directa entre estos, la red está asimilando mal los datos y llega a conclusiones incorrectas.

## Selección del mejor resultado

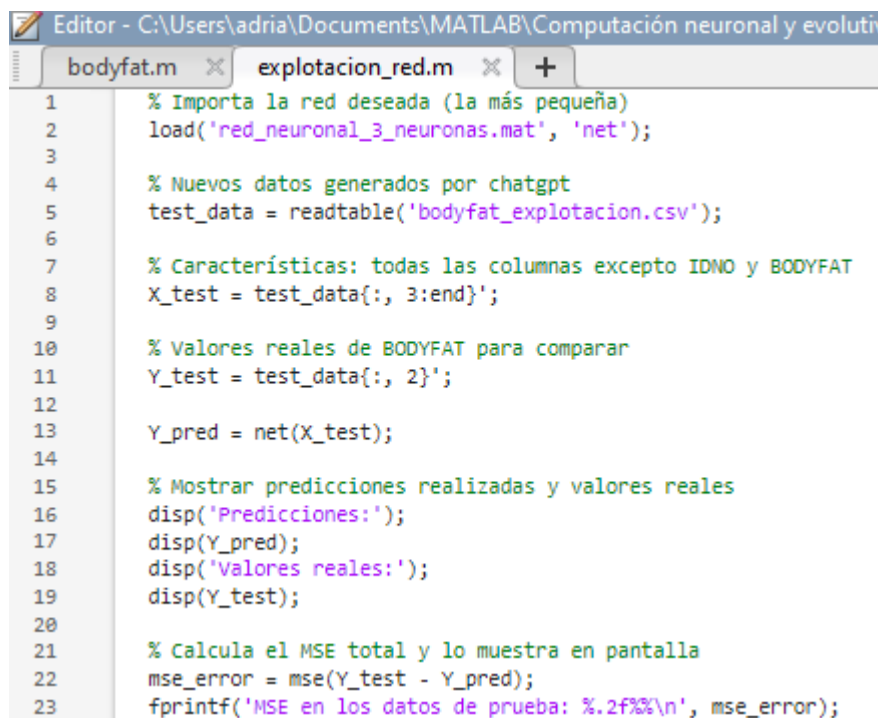
En base a los resultados obtenidos y pensando en la explotación de la red generada, considero que la red con 3 o 5 neuronas sería la que mejor resultados ofrece, pues tiene el MSE más reducido. Se trata de un modelo con un funcionamiento sencillo y con capacidad suficiente para realizar un buen trabajo en la tarea de estimar el porcentaje de grasa en base a las medidas seleccionadas.

Por los motivos ya enunciados y con los datos mostrados en el apartado anterior se procede a la explotación de la red en la tarea para la que ha sido entrenada, de esta forma podremos comprobar si la hipótesis de que esta es la mejor red es acertada.

## Etapa de explotación

Tras exportar el modelo con 3 neuronas para comprobar su eficacia en la etapa de explotación he creado este sencillo programa, con el cual puedo dar datos al modelo generado y este me devuelve su estimación, los siguientes 4 conjuntos de datos generados con inteligencia artificial para ajustarse a los datos con el formato a los utilizados durante el entrenamiento serán con los que se pruebe su eficacia.

Un pequeño programa que carga la red escogida y le pasa los datos para obtener las predicciones:



```
Editor - C:\Users\adria\Documents\MATLAB\Computación neuronal y evolutiva
bodyfat.m  explotacion_red.m  +
1  % Importa la red deseada (la más pequeña)
2  load('red_neuronal_3_neuronas.mat', 'net');
3
4  % Nuevos datos generados por chatgpt
5  test_data = readtable('bodyfat_explotacion.csv');
6
7  % Características: todas las columnas excepto IDNO y BODYFAT
8  X_test = test_data(:, 3:end)';
9
10 % Valores reales de BODYFAT para comparar
11 Y_test = test_data(:, 2)';
12
13 Y_pred = net(X_test);
14
15 % Mostrar predicciones realizadas y valores reales
16 disp('Predicciones:');
17 disp(Y_pred);
18 disp('Valores reales:');
19 disp(Y_test);
20
21 % Calcula el MSE total y lo muestra en pantalla
22 mse_error = mse(Y_test - Y_pred);
23 fprintf('MSE en los datos de prueba: %.2f%%\n', mse_error);
```

Un ejemplo de los resultados obtenidos:

```
>> explotacion_red
Predicciones:
    20.5065    4.1848    25.4052    30.5525

Valores reales:
    16.2000    5.6000    20.2000    30.4000

MSE en los datos de prueba: 11.92%
>>
```

Los resultados obtenidos son relativamente malos en comparación con el MSE obtenido durante el entrenamiento, probablemente esto se debe a que el entrenamiento se llevó a cabo con un set de datos muy reducido, de forma que la red no ha tenido suficientes datos para saber generalizar en sus predicciones con datos nuevos.