



**Universidad
Andrés Bello**

Universidad Andrés Bello
Magíster Ciencias de la Computación
Reconocimiento de Patrones

Tarea 3

Reconocimiento de Patrones

Equipo de Trabajo
Jean Contreras Leyton

Profesor
Juan Tapia

Noviembre, 2017

Tabla de Contenidos.

1.- Introducción.	2
2.- Desarrollo.	2
2.1.- Red Neuronal.	2
2.2.- Consideraciones Red Neuronal.	3
2.3.- Funciones de entrenamiento.	3
2.3.1- Gradiente Descendiente con Momentum.	3
2.3.2- Levenberg-Marquardt.	4
2.4.- Conjunto de Datos.	4
3.- Resultados.	5
3.1.- Momentum vs Tasa de aprendizaje.	5
3.2.- Cantidad óptima de unidades de capa oculta - Gradiente Descendiente.	12
3.3.- Cantidad óptima de unidades de capa oculta - Levenberg-Marquardt.	16
3.4.- Gradiente Descendiente vs Levenberg-Marquardt.	20
3.5.- Arquitectura ([29+5]-N-1) vs ([29+5+5]-N-1).	20
3.6.- Comparación porcentajes de clasificación.	23
3.7.- Clasificación conjunto de prueba tst.txt.	24
4.- Código.	25
5.- Conclusiones.	33
5.- Referencias.	33

1.- Introducción.

Las Redes Neuronales Artificiales (Warren McCulloch y Walter Pitts , 1943) son un tipo de clasificador basadas en las Redes Neuronales Naturales, su unidad mínima es denominada perceptrón (neurona) y necesita tres conjuntos de datos para entrenarse (train, test y validación). Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Cada unidad neuronal está conectada con muchas otras y los enlaces entre ellas pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Cada unidad neuronal, de forma individual, opera empleando funciones de suma.

Se dispone de una base de datos del sector de salud con información de 1.533 afiliados, esta posee distintos tipos de mediciones sobre una ventana histórica de 12 meses. Cada ejemplo consta de 29 características medidas en base a datos sociodemográficos y de comportamiento en salud. Además poseen un valor deseado donde 1: cambio de institución y 0: permanencia en la institución actual. Se desea predecir el potencial cambio de afiliado a otra institución en el mes siguiente mediante la utilización de clasificadores neuronales para discriminar entre ambas categorías. Para ello se utilizará una red neuronal tipo MLP (*Multilayer Perceptron*).

En la Sección 2 se describen los procedimientos usados para resolver el problema y sus fundamentos teóricos, en la Sección 3 se exponen los resultados obtenidos experimentalmente, representados en tablas y gráficos, en la Sección 4 se realiza una discusión respecto a los resultados, y su contraste con la teoría, por último en la Sección 5 se adjunta el código implementado con sus respectivos comentarios.

2.- Desarrollo.

2.1.- Red Neuronal.

Las Redes Neuronales Artificiales (Warren McCulloch y Walter Pitts , 1943) son un tipo de clasificador no lineales basadas en las Redes Neuronales Naturales, su unidad mínima es denominada perceptrón (neurona) y necesita tres conjuntos de datos para entrenarse (train, test y validación).

Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Cada unidad neuronal, de forma individual, trabaja en dos etapas, en la primera emplea una suma de pesos de las entradas pertenecientes a la capa entrada (*Input Layer*), mientras que la segunda evalúa una función de activación tal como se aprecia en la Imagen 2.1.1.

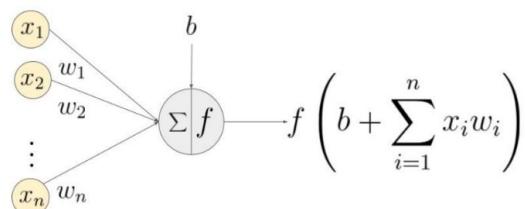


Imagen 2.1.1: Ejemplo de neurona con n entradas ($x_1 - x_n$), sus respectivos pesos ($w_1 - w_n$), un sesgo (b) y la función de activación aplicada a la suma de las entradas.

Una capa oculta (*Hidden Layer*) contiene un conjunto de neuronas (también denominadas como unidades) que están conectadas a cada una de las entradas. En una Red Neuronal pueden existir más de un capa oculta, donde las neuronas de cada conjunto están interconectadas. La última capa oculta se conecta a una capa de salida (*Output Layer*) con los resultados obtenidos, tal como se observa en la Figura 2.1.2.

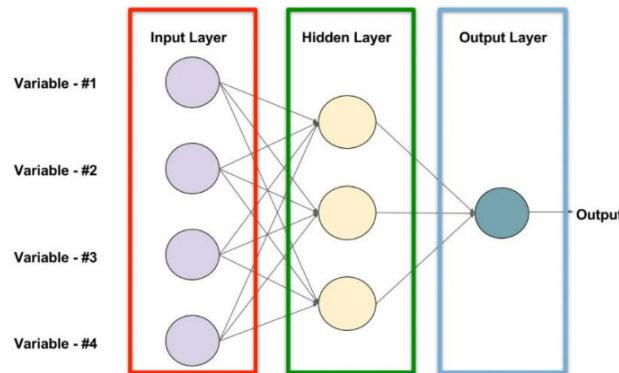


Imagen 2.1.2: Red Neuronal con 4 entradas, una capa oculta de tres neuronas y una capa de salida.

2.2.- Consideraciones Red Neuronal.

La Red neuronal a utilizar es de tipo MLP (*Multilayer Perceptron*), es decir, posee múltiples capas ocultas, para lo cual se toman en cuenta las siguientes consideraciones:

- Función a minimizar: MSE (Error cuadrático medio).
- Función de activación de capa oculta: Tangente Hiperbólica (Imagen 2.1.2).
- Función de activación de capa de salida: Lineal (Imagen 2.1.2).
- Número máximo de épocas para entrenar red: 5000.

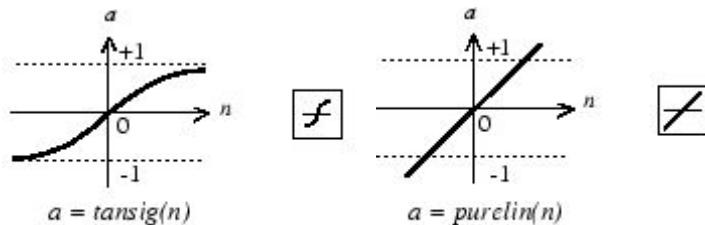


Imagen 2.1.2: Funciones: Tangente Hiperbólica (*tansig*) y Lineal (*purelin*).
Fuente: MathWorks tansig and purelin functions.

2.3.- Funciones de entrenamiento.

2.3.1- Gradiente Descendiente con Momentum.

Gradiente Descendiente con momentum (GD, *traingdm* en Matlab) es una función de entrenamiento de redes neuronales que actualiza los pesos (w) y sesgo (b) según el gradiente descendiente con momentum. Esta función permite a la red responder no solo al gradiente local, sino que también a tendencias recientes en la superficie de error. Actuando como un filtro de bajo peso, el momentum permite a la red ignorar características pequeñas en la superficie de error.

Por ejemplo, sin momentum, una red puede quedar atrapada en un mínimo local de poca profundidad, mientras que al utilizar momentum la red puede deslizarse a través de dicho mínimo. GD con momentum depende de dos parámetros de entrenamiento, el parámetro $lr (\mu)$ que indica una tasa de aprendizaje similar a una gradiente descendente simple y el parámetro $mc (\alpha)$ es la constante de momentum, mc se encuentra entre 0 (sin momentum) y valores cercanos a 1 (mucho momentum). Un mc constante de 1 resulta en una red que es completamente insensible al gradiente local y, por lo tanto, no aprende apropiadamente (Mathworks documentation, 2017).

2.3.2- Levenberg-Marquardt.

Levenberg-Marquardt (LM y *trainlm* en Matlab) es una función de entrenamiento de redes neuronales que actualiza los pesos (w) y sesgo (b) de acuerdo a la optimización LM, es a menudo el algoritmo de backpropagation más rápido del toolbox de Matlab. Al igual que los métodos cuasi Newton, el algoritmo LM fue diseñado para abordar la rapidez de entrenamiento de segundo orden sin tener que calcular la matriz de Hesse. Cuando la función de rendimiento tiene la forma de una suma de cuadrados (como es típico en las redes feedforward de entrenamiento), entonces la matriz de Hesse se puede aproximar como (1) y la gradiente puede ser computada como (2). Donde J es la matriz jacobiana que contiene las primeras derivadas de los errores de red con respecto a los pesos y los sesgos, y e es un vector de errores de red. La matriz jacobiana se puede calcular a través de una técnica de backpropagation estándar que es mucho menos compleja que la computación de la matriz de Hesse. El algoritmo de Levenberg-Marquardt usa esta aproximación a la matriz de Hesse en la siguiente actualización tipo Newton (3).

$$H = J^T J \quad (1)$$

$$g = J^T e \quad (2)$$

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e \quad (3)$$

Cuando el escalar μ es cero, este es solo el método de Newton, usando la matriz aproximada de Hesse. Cuando μ es grande, este se convierte en gradiente descendente con un tamaño pequeño de paso. El método de Newton es más rápido y más preciso cerca de un mínimo de error, por lo que el objetivo es cambiar hacia el método de Newton lo más rápido posible. Por lo tanto, μ disminuye después de cada paso exitoso (reducción en la función de rendimiento) y se incrementa solo cuando un paso tentativo podría aumentar la función de rendimiento. De esta forma, la función de rendimiento siempre se reduce en cada iteración del algoritmo (Mathworks documentation, 2017).

2.4.- Conjunto de Datos.

Se dispone de una base de datos del sector de salud con información de 1.533 afiliados, posee distintos tipos de mediciones sobre una ventana histórica de 12 meses. Cada ejemplo consta de 29 características medidas en base a datos sociodemográficos y de comportamiento en salud. Además poseen un valor deseado donde 1: cambio de institución y 0: permanencia en la institución actual. El conjunto se divide en Train (tr.txt - 800 ejemplos), Validación (va.txt - 401 ejemplos) y dos conjuntos de Test (ts.txt - 185 ejemplos y tst.txt - 147 ejemplos sin clases). Se desea predecir el potencial cambio de afiliado a otra institución en el mes siguiente mediante la utilización de clasificadores neuronales para discriminar entre ambas categorías. Para ello se utilizará una red neuronal tipo MLP (*Multilayer Perceptron*).

3.- Resultados.

En las siguientes subsecciones se describen los cinco experimentos que se llevaron a cabo con distintas configuraciones de redes neuronales. Para ello, se realizaron 10 simulaciones por cada configuración, obteniendo de esta manera diversos resultados debido a que los pesos son generados aleatoriamente. El criterio para evaluar cada red es el porcentaje de clasificaciones correctas en el conjunto *va.txt*, en caso de existir empate se resolverá obteniendo el menor valor de mse para ambos.

3.1.- Momentum vs Tasa de aprendizaje.

Utilizando una red neuronal con entrenamiento de gradiente descendiente, se realizaron una serie de experimentos para una arquitectura 29-10-1, es decir, un *Input Layer* de 29 entradas, un *Hidden Layer* de 10 unidades y el *Output Layer* de una unidad. Para ello se utilizaron los parámetros momentum $\alpha = 0, 0.9$ y tasa de aprendizaje $\mu = 0.1, 0.01, 0.001$.

- **Momentum 0.0:**

La Tabla 3.1.1 contiene los resultados de los 30 experimentos realizados para momentum 0.0. resaltando las mejores simulaciones, se observa que a menor tasa de aprendizaje, los porcentajes de clasificación correcta disminuyen, mientras que el performance (MSE) y la cantidad de épocas aumentan llegando al máximo establecido.

Esta dinámica se puede observar de forma más clara en la Tabla 3.1.2 que contiene los promedios de las diez simulaciones de la Tabla 3.1.1, donde la mejor tasa de aprendizaje es 0.1.

Simulation	Momentum: 0.0											
	learning_rate: 0.1				learning_rate: 0.01				learning_rate: 0.001			
	val (correct)	val (mse)	Total Epochs	Best Epoch	val (correct)	val (mse)	Total Epochs	Best Epoch	val (correct)	val (mse)	Total Epochs	Best Epoch
1	0.7581	0.1620	970	964	0.7656	0.1618	5000	5000	0.7057	0.2038	5000	5000
2	0.7756	0.1538	4092	4086	0.7731	0.1547	5000	5000	0.5786	0.2359	5000	5000
3	0.8030	0.1408	1864	1858	0.7830	0.1633	5000	5000	0.6608	0.2201	5000	5000
4	0.8204	0.1369	2598	2592	0.7581	0.1722	5000	5000	0.6409	0.2103	5000	5000
5	0.8030	0.1486	3177	3171	0.7531	0.1665	5000	5000	0.5536	0.2403	5000	5000
6	0.8055	0.1466	2074	2068	0.7830	0.1544	5000	5000	0.6758	0.1969	5000	5000
7	0.7955	0.1458	1903	1897	0.7681	0.1610	5000	5000	0.6160	0.2312	5000	5000
8	0.7905	0.1464	2698	2692	0.7631	0.1640	5000	5000	0.6185	0.2201	5000	5000
9	0.8080	0.1470	2723	2717	0.7382	0.1692	5000	5000	0.6833	0.2084	5000	5000
10	0.8105	0.1439	1552	1546	0.7781	0.1671	5000	5000	0.6409	0.2246	5000	5000

Tabla 3.1.1: Porcentaje de clasificaciones correctas, performance (error cuadrático medio), cantidad de épocas y mejor época para el conjunto de validación por cada una de las 10 simulaciones, cada tasa de aprendizaje y momentum 0.0. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Las imágenes (Imagen 3.1.1.1 - 3.1.1.3) poseen los gráficos de performance (MSE) vs épocas y matrices de confusión para los conjuntos de train, validación y testing de los tres mejores resultados para momentum 0.0. En ellas se observa un comportamiento similar, a excepción de la Imagen 3.1.1.1 en la que se llega a la mejor performance en un 52% del máximo de épocas, obteniendo el mejor valor de los tres.

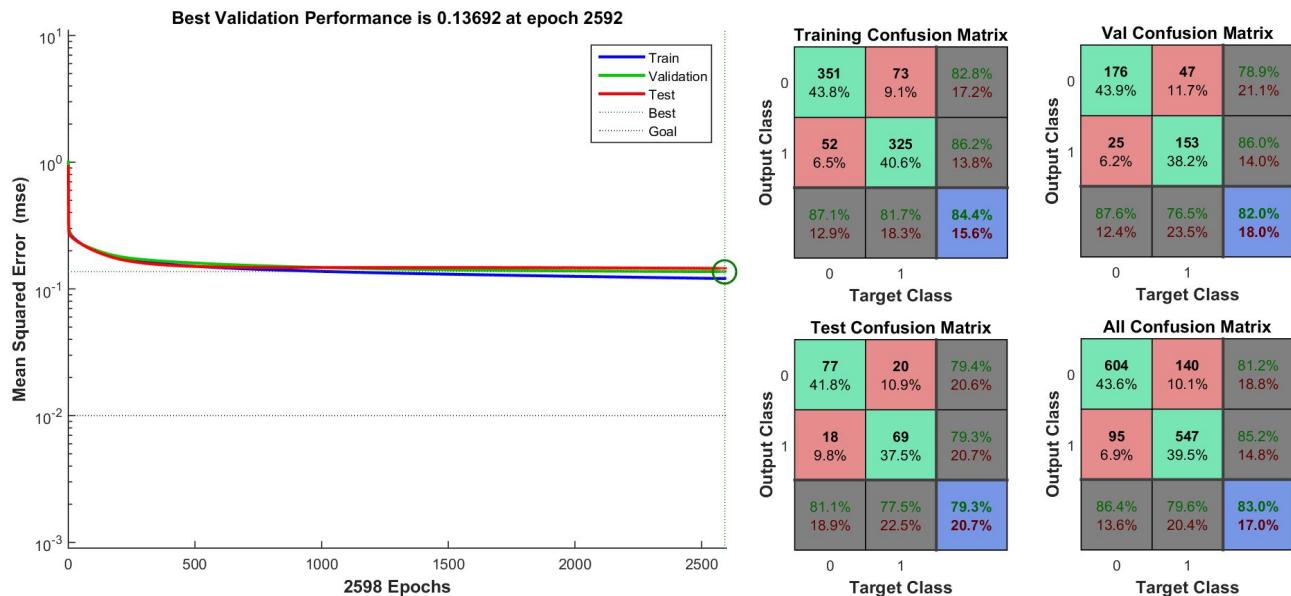


Imagen 3.1.1.1: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (4) para momentum 0.0 y tasa de aprendizaje 0.1.

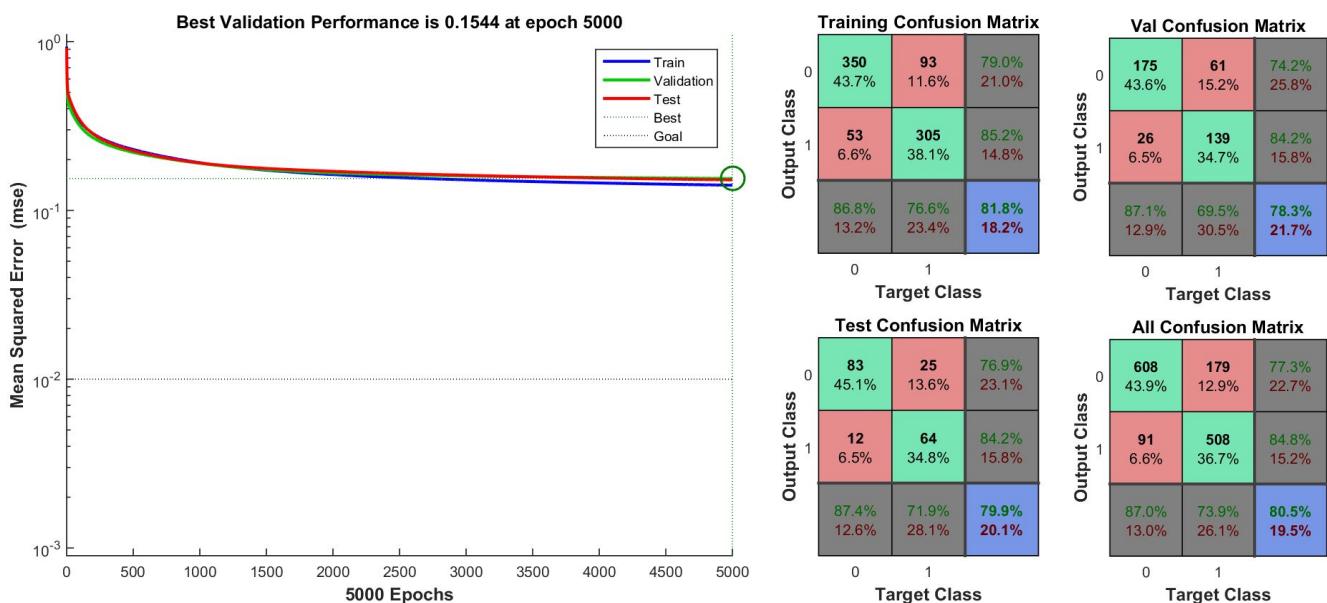


Imagen 3.1.1.2: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (6) para momentum 0.0 y tasa de aprendizaje 0.01.



Imagen 3.1.1.3: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (1) para momentum 0.0 y tasa de aprendizaje 0.001.

- Mejor learning rate para Momentum 0.0:

Learning_rate	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
0.1	0.797007	0.018295	0.147186	0.006866
0.01	0.766334	0.014109	0.163417	0.005765
0.001	0.637406	0.047290	0.219169	0.014260

Tabla 3.1.2: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE, para cada tasa de aprendizaje con momentum 0.0. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla la mejor tasa de aprendizaje 0.1.

- **Momentum 0.9:**

La Tabla 3.1.3 contiene los resultados de los 30 experimentos realizados para momentum 0.9. resaltando las mejores simulaciones, se observa que los porcentajes de clasificación correcta son de por sí inferiores a los obtenidos con momentum 0.0, pero cabe destacar que al disminuir la tasa de aprendizaje de 0.1 a 0.01 se nota una pequeña alza de este, que luego disminuye al pasar a 0.001 siendo de igual manera superior a los resultados de 0.1. Con respecto al MSE ocurre lo contrario, es decir, decrece y luego aumenta su valor, a pesar que en la Tabla 3.1.4, la cual contiene los promedios de las diez simulaciones de la Tabla 3.1.3 y la mejor tasa de aprendizaje 0.01 para momentum 0.9, el MSE solo disminuye.

En el caso de las épocas, estas tienden a aumentar mientras la tasa de aprendizaje decrece, de manera similar a momentum 0, pero con la particularidad de que en un inicio (0.1) las épocas son extremadamente pequeñas y que para 0.01 a menor cantidad de épocas, peor es el desempeño de la red neuronal.

Esto lleva a concluir que el uso de momentum permite llegar al mejor performance en una menor cantidad de épocas, pero sus resultados son inferiores a los que se obtienen sin utilizarlo.

Simulation	Momentum: 0.9											
	learning_rate: 0.1				learning_rate: 0.01				learning_rate: 0.001			
	val (correct)	val (mse)	Total Epochs	Best Epoch	val (correct)	val (mse)	Total Epochs	Best Epoch	val (correct)	val (mse)	Total Epochs	Best Epoch
1	0.5262	0.4570	7	1	0.7631	0.1604	5000	5000	0.6534	0.2257	5000	5000
2	0.4489	0.4048	8	2	0.7581	0.1663	5000	5000	0.6259	0.2316	5000	5000
3	0.4414	0.4371	7	1	0.5362	0.2999	14	8	0.6584	0.2161	5000	5000
4	0.5835	0.3230	7	1	0.7631	0.1617	5000	5000	0.6908	0.1895	5000	5000
5	0.6035	0.2446	7	1	0.3691	0.4123	15	9	0.5761	0.2415	5000	5000
6	0.4539	0.3572	7	1	0.7531	0.1609	5000	5000	0.6708	0.2067	5000	5000
7	0.5486	0.3720	7	1	0.7257	0.1759	5000	5000	0.6160	0.2351	5000	5000
8	0.4813	0.3143	8	2	0.6209	0.2935	20	14	0.6035	0.2325	5000	5000
9	0.3990	0.3939	7	1	0.7656	0.1642	5000	5000	0.6658	0.2181	5000	5000
10	0.4439	0.3075	7	1	0.4065	0.3801	17	11	0.6858	0.1955	5000	5000

Tabla 3.1.3: Porcentaje de clasificaciones correctas, performance (error cuadrático medio), cantidad de épocas y mejor época para el conjunto de validación por cada una de las 10 simulaciones, cada tasa de aprendizaje y momentum 0.9. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Las imágenes (Imagen 3.1.1.4 - 3.1.1.6) poseen los gráficos de performance (MSE) vs épocas y matrices de confusión para los conjuntos de train, validación y testing de los tres mejores resultados para momentum 0.9. En ellas se observa un comportamiento similar, a excepción de la Imagen 3.1.1.1 en la que se llega a la mejor performance en un 0.0002% del máximo de épocas, obteniendo el peor valor de los tres.

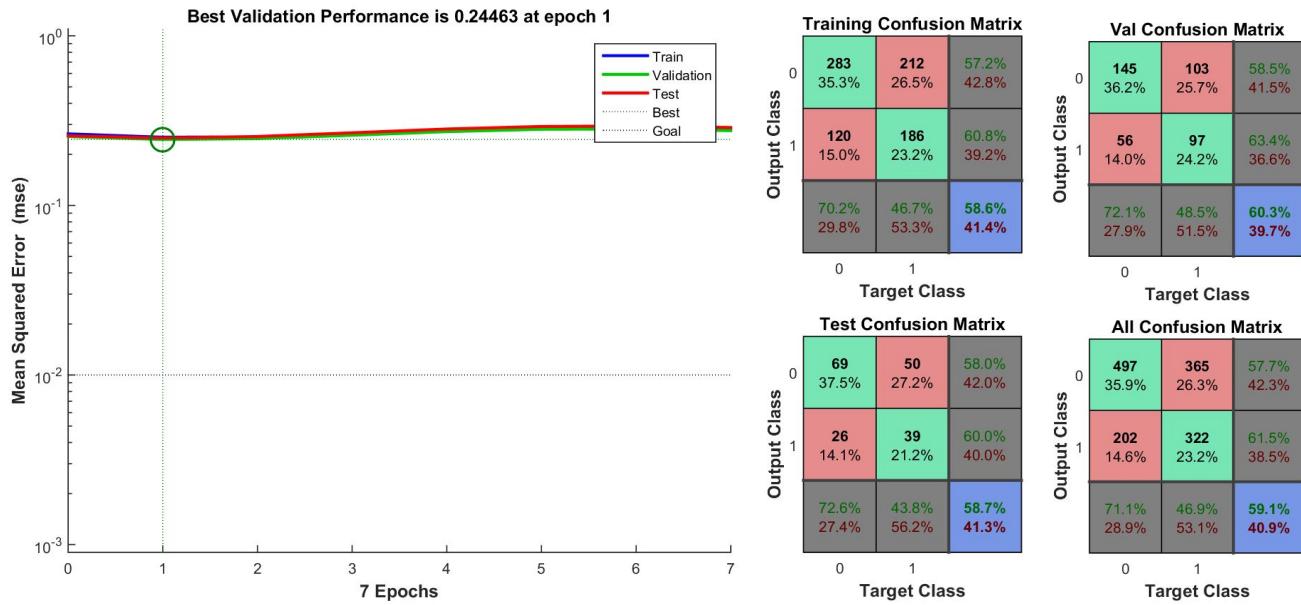


Imagen 3.1.1.4: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (5) para momentum 0.9 y tasa de aprendizaje 0.1.

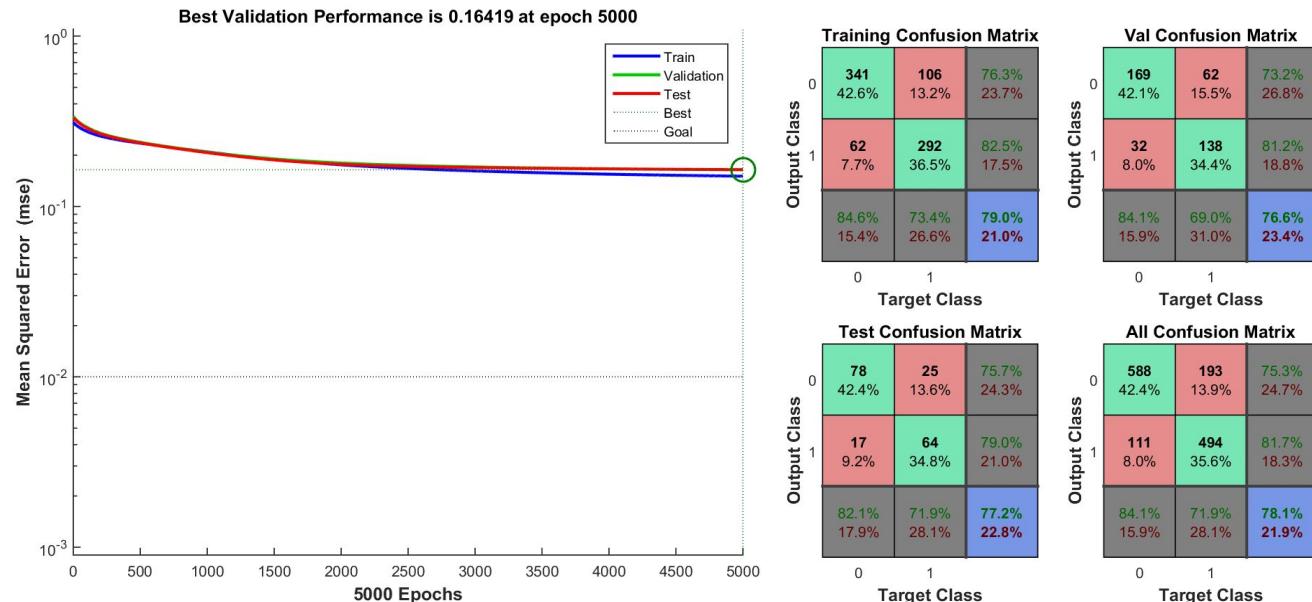


Imagen 3.1.1.5: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (9) para momentum 0.9 y tasa de aprendizaje 0.01.



Imagen 3.1.1.5: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (4) para momentum 0.9 y tasa de aprendizaje 0.001.

- Mejor learning rate para Momentum 0.9:

Learning_rate	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
0.1	0.493017	0.068416	0.361140	0.065131
0.01	0.646135	0.155942	0.237516	0.099849
0.001	0.644638	0.037696	0.219229	0.017438

Tabla 3.1.4: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE, para cada tasa de aprendizaje con momentum 0.9. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla la mejor tasa de aprendizaje 0.01.

- Momentum 0.0 vs 0.9 (Mejores Parámetros):

Momentum	Learning rate	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
0	0.1	0.7970	0.0183	0.1472	0.0069
0.9	0.01	0.6461	0.1559	0.2375	0.0998

Tabla 3.1.5: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE, para las mejores tasas de aprendizaje y los mejores valores de momentum. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla la mejor tasa de aprendizaje 0 y momentum 0.1.

- **Porcentajes de clasificación:**

Momentum	Learning_rate	ctr_mean	ctr_std	eval_mean	eval_std	cts_mean	cts_std
0	0.1	0.831710	0.015243	0.797007	0.018295	0.801087	0.013605
0	0.01	0.788015	0.017237	0.766334	0.014109	0.785326	0.021165
0	0.001	0.654307	0.051493	0.637406	0.047290	0.658152	0.049708
0.9	0.1	0.492385	0.062873	0.493017	0.068416	0.464674	0.065481
0.9	0.01	0.664794	0.167237	0.646135	0.155942	0.652717	0.171547
0.9	0.001	0.651061	0.046698	0.644638	0.037696	0.646739	0.060411

Tabla 3.1.6: Promedio y desviación estándar de porcentajes de clasificación correctos para los conjuntos de entrenamiento, validación y prueba de cada valor de momentum y tasa de aprendizaje. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Clasificaciones correctas por configuración

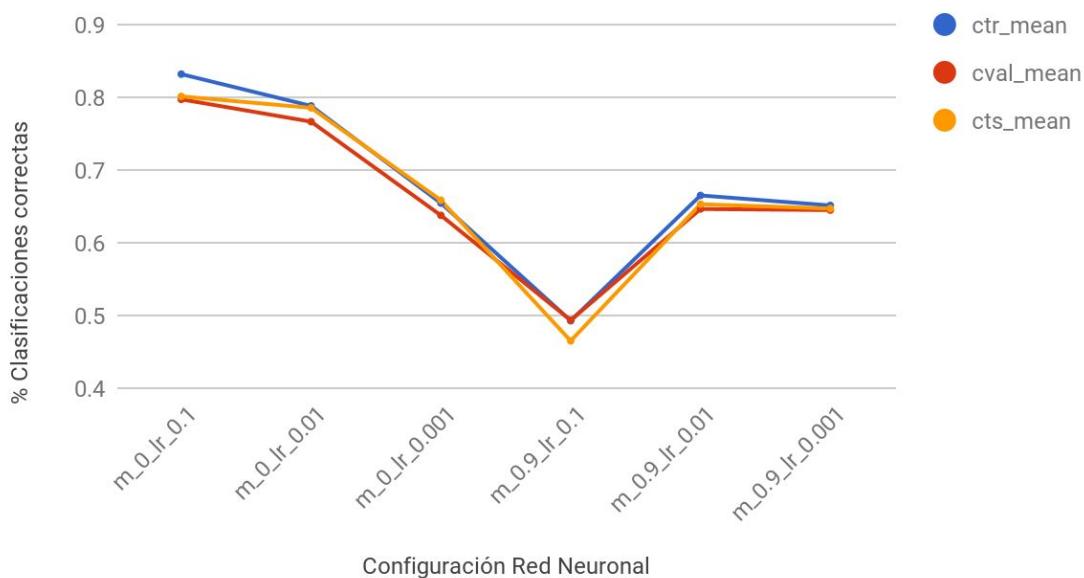


Gráfico 3.1.1: Porcentaje de clasificaciones correctas para los conjuntos de entrenamiento, validación y prueba, los tres presentan un comportamiento similar tal como se explica en la sección 3.1.

3.2.- Cantidad óptima de unidades de capa oculta - Gradiente Descendiente.

Utilizando una red neuronal con entrenamiento de gradiente descendente y utilizando la mejor tasa de aprendizaje y momentum de la Sección 3.1 (0.1 y 0 respectivamente), se determinó experimentalmente el número óptimo de unidades (N) de la capa oculta, con $N = 0$ (sin capa oculta), 6, 12 y 16, para redes con arquitectura 29-N-1.

La Tabla 3.1.1 contiene los resultados de los 40 experimentos realizados para los distintos valores de N resaltando las mejores simulaciones, se observa que a mayor cantidad de unidades disminuye la cantidad de épocas necesarias para encontrar el mejor performance, por ejemplo, para 0 unidades llega hasta el máximo de épocas permitidas, mientras que para 16 unidades solo necesita un 47% del total de épocas, llegando incluso a un 25% para la primera simulación.

Respecto a los porcentajes de clasificación, estos tienen su mayor valor para un total de 6 unidades en la capa oculta, este valor disminuye un poco en 12 unidades y vuelve a subir con 16 (siendo menor al de 6 unidades), pero tal como se aprecia en el Gráfico 3.2.1 el porcentaje de clasificación no aumenta para $N = 16$, sino que disminuye, por lo que se podría pensar en una capa oculta con 6 unidades como arquitectura óptima para este conjunto de datos en particular.

	Momentum: 0.0 Learning rate: 0.1											
	units: 0			units: 6			units: 12			units: 16		
Simulation	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch
1	0.7756	0.1653	5000	0.8105	0.1423	3106	0.7980	0.1465	2347	0.7830	0.1565	1250
2	0.7805	0.1652	5000	0.7930	0.1500	4293	0.7905	0.1494	3252	0.7880	0.1459	1887
3	0.7756	0.1654	5000	0.7905	0.1442	2353	0.7905	0.1542	2101	0.7980	0.1553	2138
4	0.7830	0.1645	5000	0.8204	0.1411	4416	0.8080	0.1504	2389	0.7830	0.1577	1280
5	0.7731	0.1661	5000	0.8030	0.1458	2027	0.7955	0.1544	2143	0.7980	0.1505	2337
6	0.7781	0.1650	5000	0.8130	0.1352	5000	0.7955	0.1505	2302	0.8155	0.1404	2367
7	0.7781	0.1645	5000	0.8155	0.1367	2415	0.8005	0.1460	2590	0.7855	0.1581	2935
8	0.7756	0.1657	5000	0.7706	0.1634	2264	0.7955	0.1443	1466	0.7855	0.1546	1743
9	0.7731	0.1654	5000	0.8180	0.1393	2332	0.8005	0.1430	2780	0.7781	0.1615	1443
10	0.7830	0.1647	5000	0.8130	0.1375	2288	0.8080	0.1436	4168	0.7880	0.1587	2442

Tabla 3.2.1: Porcentaje de clasificaciones correctas, performance (error cuadrático medio) y mejor época para el conjunto de validación por cada una de las 10 simulaciones y unidades 0, 6, 12 y 16 para momentum 0.0 y tasa de aprendizaje 0.1. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Las imágenes (Imagen 3.2.1.1 - 3.2.1.4) poseen los gráficos de performance (MSE) vs épocas y matrices de confusión para los conjuntos de train, validación y testing de los cuatro mejores resultados para unidades 0, 6, 12 y 16. En ellas se observa que a excepción del grafico de la Figura 3.2.1.1 cada uno de los mejores resultados tienen un comportamiento similar, pero con un valor de error para prueba y validación mayores a los de entrenamiento (lo que implica un sobreajuste o sobreentrenamiento del modelo). En el caso del menor error, coincide con el que tiene menor cantidad de épocas, que es 0.14044 para 16 unidades en la Imagen 3.2.1.4.



Imagen 3.2.1.1: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (4) para 0 unidades en la capa oculta.

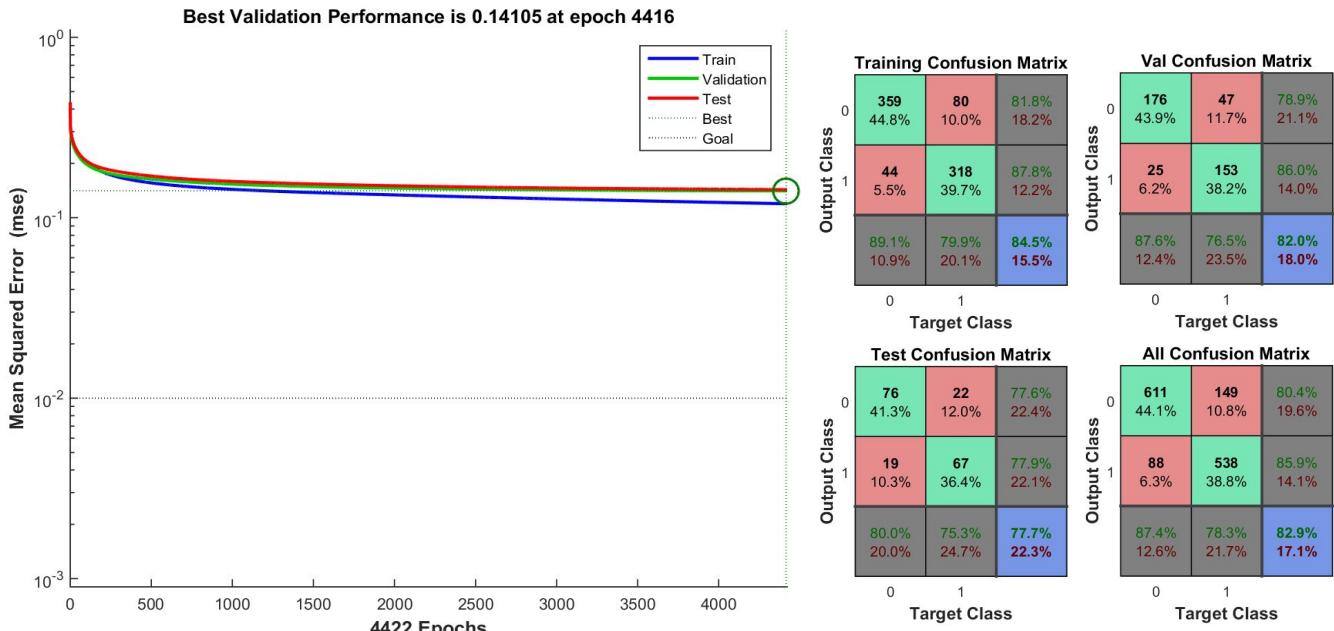


Imagen 3.2.1.2: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (4) para 6 unidades en la capa oculta.



Imagen 3.2.1.3: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (10) para 12 unidades en la capa oculta.

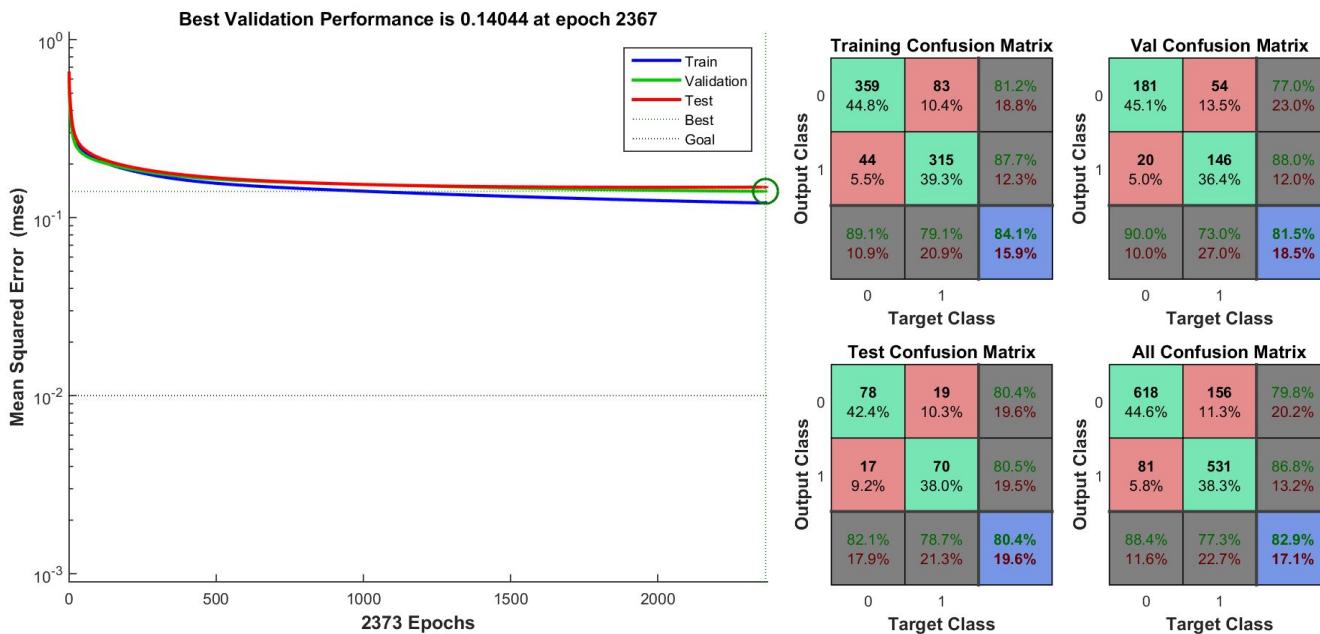


Imagen 3.2.1.4: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (6) para 16 unidades en la capa oculta.

- **N-óptimo:**

Units	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
0	0.777556	0.003680	0.165175	0.000510
6	0.804738	0.015686	0.143535	0.008296
12	0.798254	0.006159	0.148239	0.004182
16	0.790274	0.010867	0.153896	0.006473

Tabla 3.2.2: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE del conjunto de validación, para cada valor de N. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla el mejor valor de N que en este caso es 6 unidades.

val_corr (main) v/s val_mse (mean)

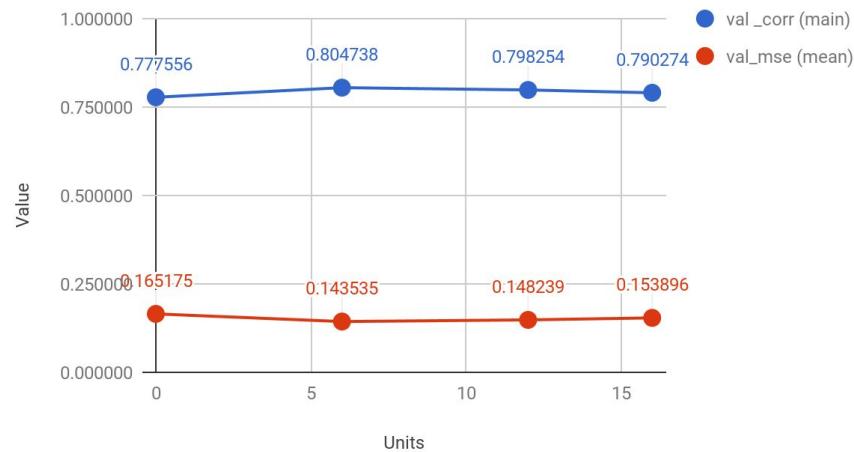


Gráfico 3.2.1: Representación gráfica de Tabla 3.2.2, donde se observa un alza en porcentaje promedio de clasificaciones correctas para 6 unidades, pero a diferencia de lo descrito en la Tabla 3.2.1 sigue disminuyendo para N = 12 y 16.

- **Porcentajes de clasificación:**

Units	ctr_mean	ctr_std	eval_mean	eval_std	cts_mean	cts_std
0	0.792385	0.001323	0.777556	0.003680	0.789130	0.004287
6	0.831960	0.008871	0.804738	0.015686	0.805435	0.014673
12	0.844320	0.005585	0.798254	0.006159	0.800000	0.015542
16	0.839576	0.010744	0.790274	0.010867	0.807609	0.005250

Tabla 3.2.3: Promedio y desviación estándar de porcentajes de clasificación correctos para los conjuntos de entrenamiento, validación y prueba, para N = 0, 6, 12 y 16. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

3.3.- Cantidad óptima de unidades de capa oculta - Levenberg-Marquardt.

Utilizando una red neuronal con algoritmo de *backpropagation* de segundo orden con tasa de aprendizaje adaptativa de Levenberg-Marquardt (LM) y al igual que la Sección 3.2, se determinó experimentalmente el número óptimo de unidades (N) de la capa oculta, con $N = 0$ (sin capa oculta), 6, 12 y 16, para redes con arquitectura 29-N-1.

La Tabla 3.3.1 contiene los resultados de los 40 experimentos realizados para los distintos valores de N resaltando las mejores simulaciones, se observa un comportamiento similar al utilizar momentum 0.9 en la Sección 3.1, debido a la minúscula cantidad de épocas necesarias para obtener el mejor performance del conjunto de validación, siendo los mejores resultados los de $N = 6$. Por otro lado los mejores resultados de porcentajes de clasificación se observan nuevamente en una capa oculta con 6 unidades, a pesar de poseer la mayor cantidad de épocas.

Esto supone una mejora considerable con respecto al resto de configuraciones de red neuronal realizados en los experimentos anteriores, puesto que se pueden obtener buenos resultados (bajo error y alto porcentaje de clasificación) con una pequeña cantidad de épocas.

	units: 0			units: 6			units: 12			units: 16		
Simulation	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch	val (correct)	val (mse)	Best Epoch
1	0.7706	0.1668	*2	0.8204	0.1343	5	0.8005	0.1355	4	0.8030	0.1442	4
2	0.7830	0.1655	**1	0.8130	0.1380	9	0.8080	0.1450	6	0.7830	0.1596	3
3	0.7731	0.1643	**1	0.8229	0.1258	6	0.8304	0.1511	8	0.7855	0.1636	6
4	0.7656	0.1662	**1	0.7955	0.1479	10	0.8055	0.1517	9	0.7830	0.1472	5
5	0.7606	0.1640	**1	0.8080	0.1202	9	0.8155	0.1371	6	0.8180	0.1364	8
6	0.7706	0.1668	*2	0.8254	0.1359	8	0.7756	0.1563	3	0.8080	0.1427	7
7	0.7731	0.1674	*2	0.8229	0.1327	5	0.8304	0.1421	7	0.8055	0.1572	4
8	0.7681	0.1651	**1	0.8304	0.1177	14	0.8204	0.1334	10	0.8030	0.1507	8
9	0.7706	0.1667	*2	0.8130	0.1346	10	0.7905	0.1604	5	0.7905	0.1519	3
10	0.7706	0.1669	**1	0.8254	0.1250	11	0.8155	0.1473	5	0.7930	0.1552	6

Tabla 3.3.1: Porcentaje de clasificaciones correctas, performance (error cuadrático medio) y mejor época para el conjunto de validación por cada una de las 10 simulaciones y unidades 0, 6, 12 y 16. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado. * implica que se realizaron 2 épocas de *validation check*, mientras que ** significa que se realizaron 3 épocas de *validation check*, el resto realizó 6.

Las imágenes (Imagen 3.3.1.1 - 3.3.1.4) poseen los gráficos de performance (MSE) vs épocas y matrices de confusión para los conjuntos de train, validación y testing de los cuatro mejores resultados para unidades 0, 6, 12 y 16 para algoritmo LM. En ellas se observa que al aumentar la cantidad de unidades, el performance de los conjuntos de datos de validación y prueba aumentan

considerablemente, alejándose de los resultados entregados por el conjunto de entrenamiento (lo que implica un sobreajuste o sobreentrenamiento del modelo).

Esto se ve luego de la cuarta época para 6, 12 y 16. El mejor performance se encuentra en $N = 6$ y época 14, con un comportamiento de prueba peor que en $N = 0$, pero paralelo al de entrenamiento, por lo que no resulta tan malo como el resto y al tener a la vez el menor error de todos los valores de N , lo que implica que es la mejor arquitectura para el conjunto de datos, al igual que en la Tabla 3.2.2.

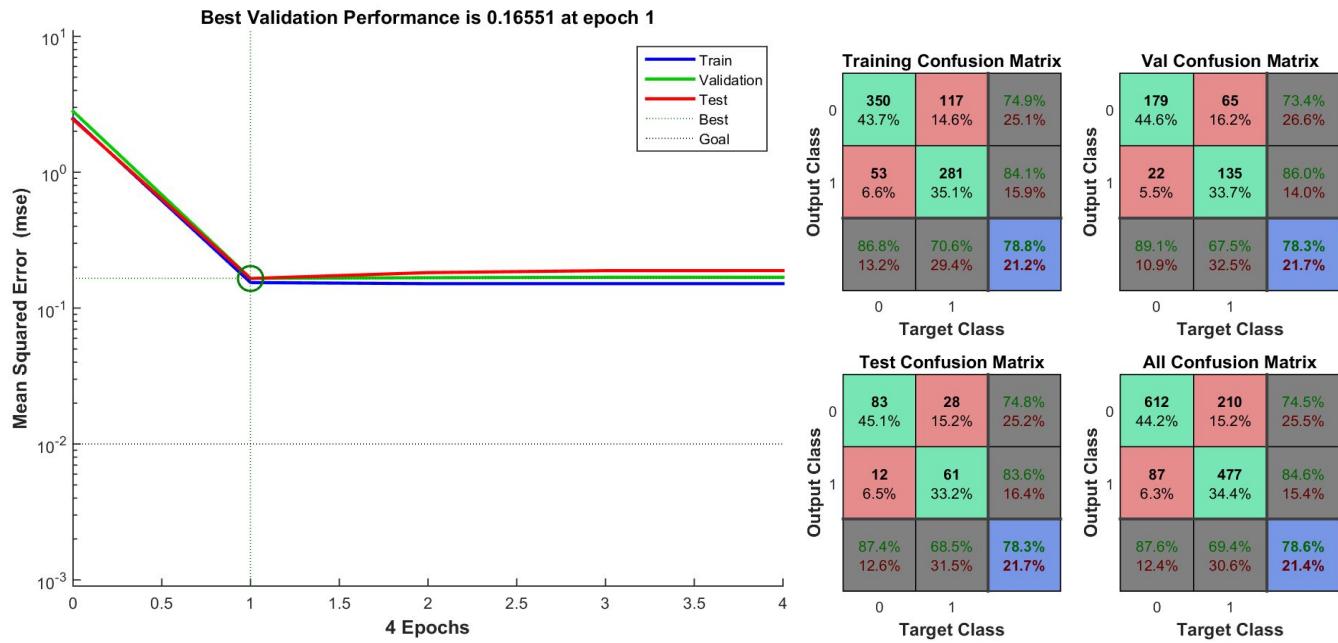


Imagen 3.3.1.1: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (2) para 0 unidades en la capa oculta (Algoritmo LM).

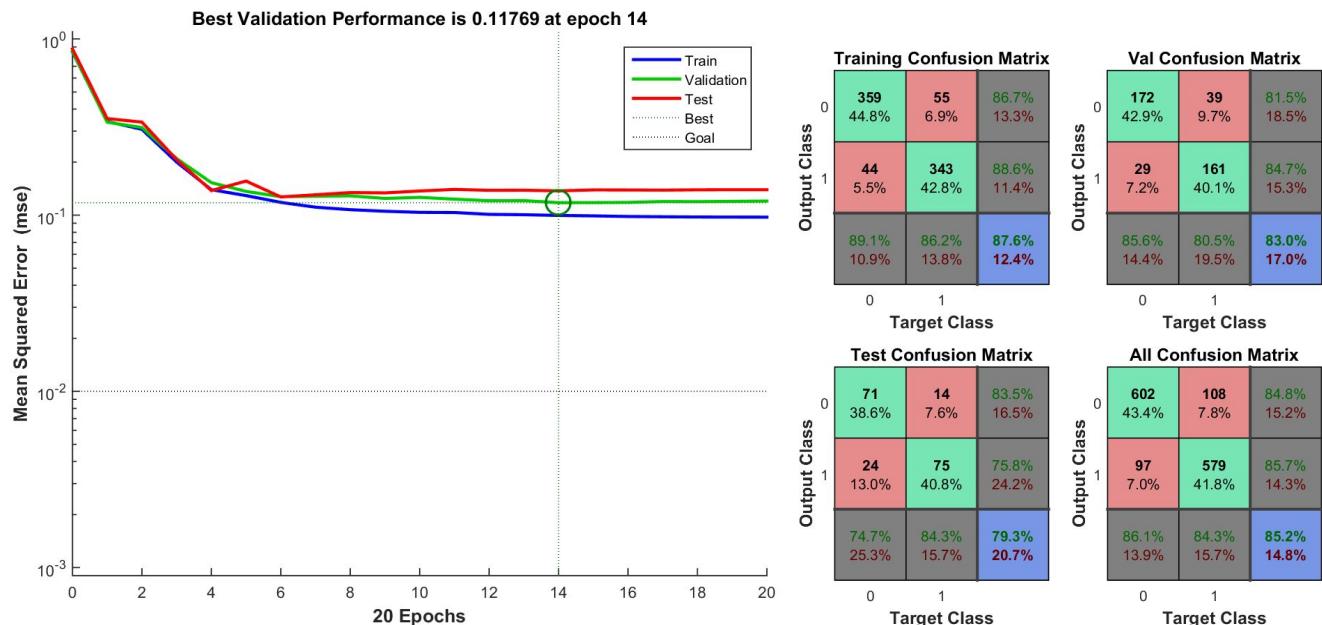


Imagen 3.3.1.2: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (8) para 6 unidades en la capa oculta (Algoritmo LM).

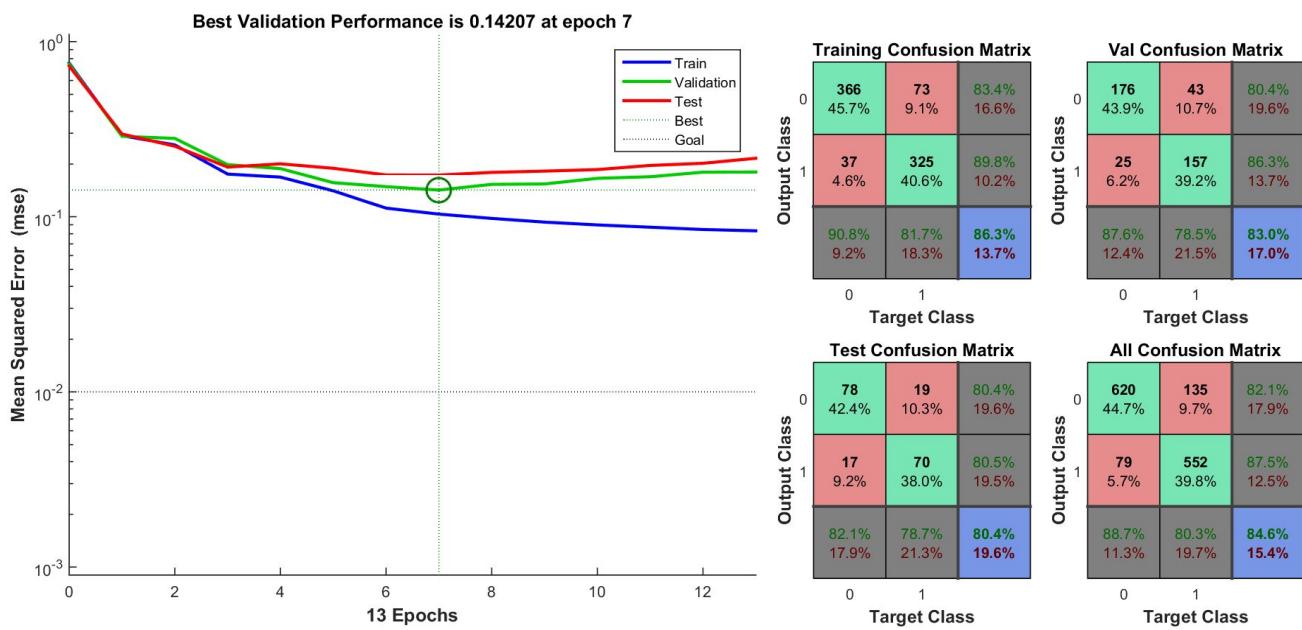


Imagen 3.3.1.3: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (7) para 12 unidades en la capa oculta (Algoritmo LM).

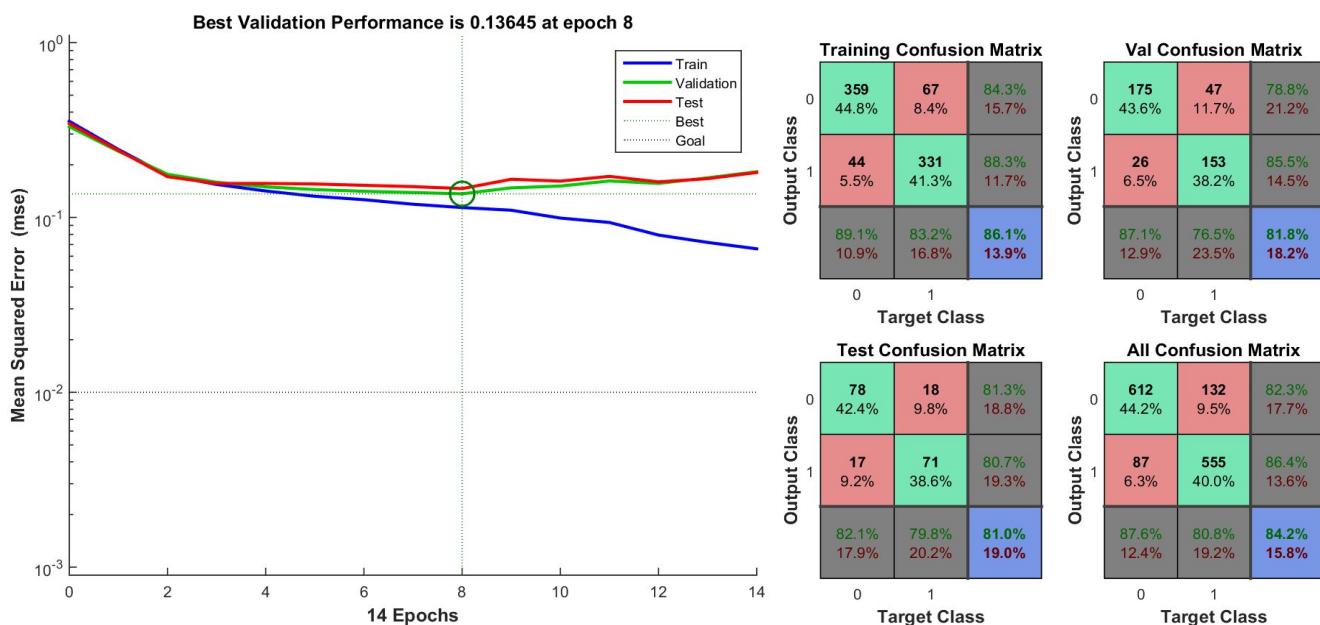


Imagen 3.3.1.4: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (5) para 16 unidades en la capa oculta (Algoritmo LM).

- **N-óptimo:**

Units	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
0	0.770574	0.005759	0.165983	0.001180
6	0.817706	0.010412	0.131205	0.009074
12	0.809227	0.017247	0.145980	0.009050
16	0.797257	0.011934	0.150894	0.008371

Tabla 3.3.2: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE del conjunto de validación, para cada valor de N (Algoritmo LM). La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla el mejor valor de N que en este caso es 6 unidades.

- **Porcentajes de clasificación:**

Units	ctr_mean	ctr_std	eval_mean	eval_std	cts_mean	cts_std
0	0.786642	0.007073	0.770574	0.005759	0.786413	0.005156
6	0.857303	0.015627	0.817706	0.010412	0.815761	0.017461
12	0.863920	0.016698	0.809227	0.017247	0.802717	0.013569
16	0.857553	0.012309	0.797257	0.011934	0.798370	0.018905

Tabla 3.3.3: Promedio y desviación estándar de porcentajes de clasificación correctos para los conjuntos de entrenamiento, validación y prueba, para N = 0, 6, 12 y 16 (Algoritmo LM). Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Comparación configuraciones anteriores con % clasificación de ts.txt

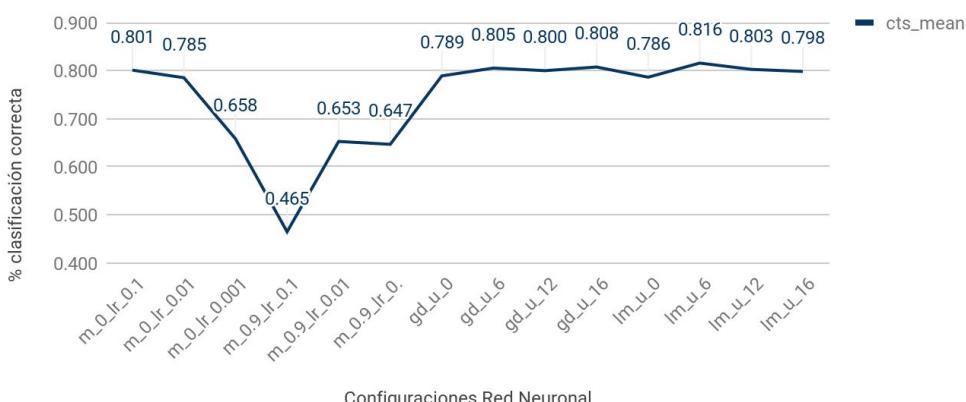


Gráfico 3.3.1: Porcentaje de clasificaciones correctas del conjunto de prueba, para cada una de las configuraciones anteriores, los lm corresponden a las redes LM y que tienen un comportamiento mucho mayor a todos los demás para N = 6 con un porcentaje de 0.816.

3.4.- Gradiente Descendiente vs Levenberg-Marquardt.

Luego de obtener los resultados de redes con Gradiente Descendiente (GD) y con Levenberg-Marquardt (LM), se concluyó que para ambas la cantidad de unidades óptima de la capa oculta es 6. En la Tabla 3.4.1 se aprecia cuál de los dos tipos de redes neuronales es la mejor (basado en porcentaje de clasificación para conjunto de validación).

Type	Units	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
GD	6	0.8047	0.0157	0.1435	0.0083
LM	6	0.8177	0.0104	0.1312	0.0091

Tabla 3.4.1: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE del conjunto de validación, para cada Gradiente Descendiente y Levenberg-Marquardt con 6 unidades. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla la mejor configuración que en este caso es LM con 6 unidades.

3.5.- Arquitectura ([29+5]-N-1) vs ([29+5+5]-N-1).

Utilizando una red neuronal con algoritmo de *backpropagation* de segundo orden con tasa de aprendizaje adaptativa de Levenberg-Marquardt (LM), con la mejor arquitectura obtenida de la Sección 3.3 (29-6-1), pero modificando la capa de entrada.

Esta modificación consiste en agregar cinco variables obtenidas de un conjunto de candidatos con el formato de la Figura 3.5.1 (a), por lo que la arquitectura en vez de ser (29-6-1), será (34-6-1) ó ([29+5]-6-1). Luego se realizó otra modificación agregando otras cinco características del segundo conjunto candidatos de la Figura 3.5.1 (b), de esta manera la nueva arquitectura pasó a ser (39-6-1) ó ([29+5+5]-6-1)

$$\begin{array}{llllll} \text{(a)} & X13-X17 & X14-X18 & X14-X17 & X14-X24 & X14-X20 \\ \text{(b)} & X13-X24 & X13-X12 & X13-X20 & X13-X23 & X13-X13 \end{array}$$

Imagen 3.5.1: **(a)** Primer conjunto candidatos, X13-X17 se refiere a que la característica 13 del conjunto de datos original debe restarse a la característica 17. **(b)** Segundo conjunto candidatos, X13-X24 se refiere a que la característica 13 del conjunto de datos original debe restarse a la característica 24. Se calcula el valor absoluto para que todas sean positivas.

La Tabla 3.3.1 contiene los resultados de los 20 experimentos realizados para capa de entrada [29+5] y [29+5+5] resaltando las mejores simulaciones, se observa que al emplear el algoritmo LM la cantidad de épocas es baja, pero un poco más altas que las registradas en la configuración (29-6-1) con LM, por otro lado los mejores porcentajes de clasificación se encuentran en [29+5] al igual que los valores de performance.

Simulation	29+5				29+5+5			
	val (correct)	val (mse)	Total Epochs	Best Epoch	val (correct)	val (mse)	Total Epochs	Best Epoch
1	0.8379	0.1290	12	6	0.8229	0.1318	13	7
2	0.7980	0.1305	15	9	0.8229	0.1279	13	7
3	0.8229	0.1254	14	8	0.8155	0.1271	12	6
4	0.8254	0.1343	14	8	0.7955	0.1419	12	6
5	0.8254	0.1220	14	8	0.8005	0.1341	12	6
6	0.8204	0.1328	25	19	0.8254	0.1319	12	6
7	0.8279	0.1298	10	4	0.7955	0.1436	13	7
8	0.8304	0.1218	17	11	0.8105	0.1384	16	10
9	0.7980	0.1466	14	8	0.7955	0.1505	13	7
10	0.7955	0.1545	14	8	0.8279	0.1330	14	8

Tabla 3.5.1: Porcentaje de clasificaciones correctas, performance (error cuadrático medio), cantidad total de épocas y mejor época para el conjunto de validación por cada una de las 10 simulaciones y capa de entradas 29+5 y 29+5+5. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado.

Las imágenes 3.5.1.1 y 3.5.1.2 poseen los gráficos de performance (MSE) vs épocas y matrices de confusión para los conjuntos de train, validación y testing de los cuatro mejores resultados para configuraciones [29+5] y [29+5+5] con redes neuronales con algoritmo LM. En ellas se observa que el performance de [29+5] para los conjuntos de validación y prueba es mejor que el de [29+5+5], los cuales se alejan de los valores del conjunto de entrenamiento siendo peores (lo que implica un sobreajuste o sobreentrenamiento del modelo). Por otro lado el mejor performance lo tiene [29+5] con un MSE de 0.129, un 3% mejor que el de [29+5+5] por lo que no se puede definir con claridad cuál es el mejor, para ello se obtienen los resultados promedios para las 10 simulaciones tal como se hace en la Tabla 3.5.2.

La Tabla 3.5.2 compara los resultados promedio entre [29+5] y [29+5+5] obteniendo los mejores resultados con la arquitectura ([29+5]-6-1) con un porcentaje de clasificación de 0.818 (mayor al 0.817 al utilizar solo 29 unidades en la capa de entrada), por lo que se considera que agregar

las primeras 5 características es mucho mejor que agregar las 10 (que genera resultados peores que al utilizar solo las 29), puesto que además de obtener un mejor performance el error es menor, a costa eso sí, de una pequeña alza en la cantidad de épocas necesarias.

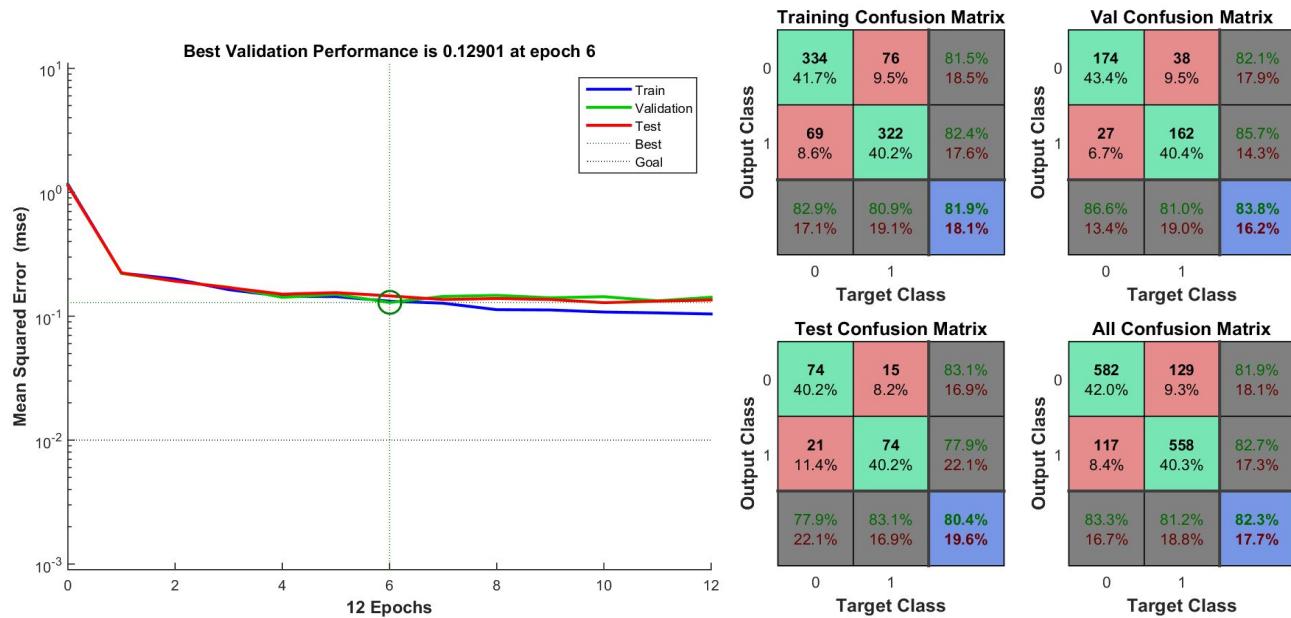


Imagen 3.5.1.1: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (1) para configuración (25+5) de red con Algoritmo LM.

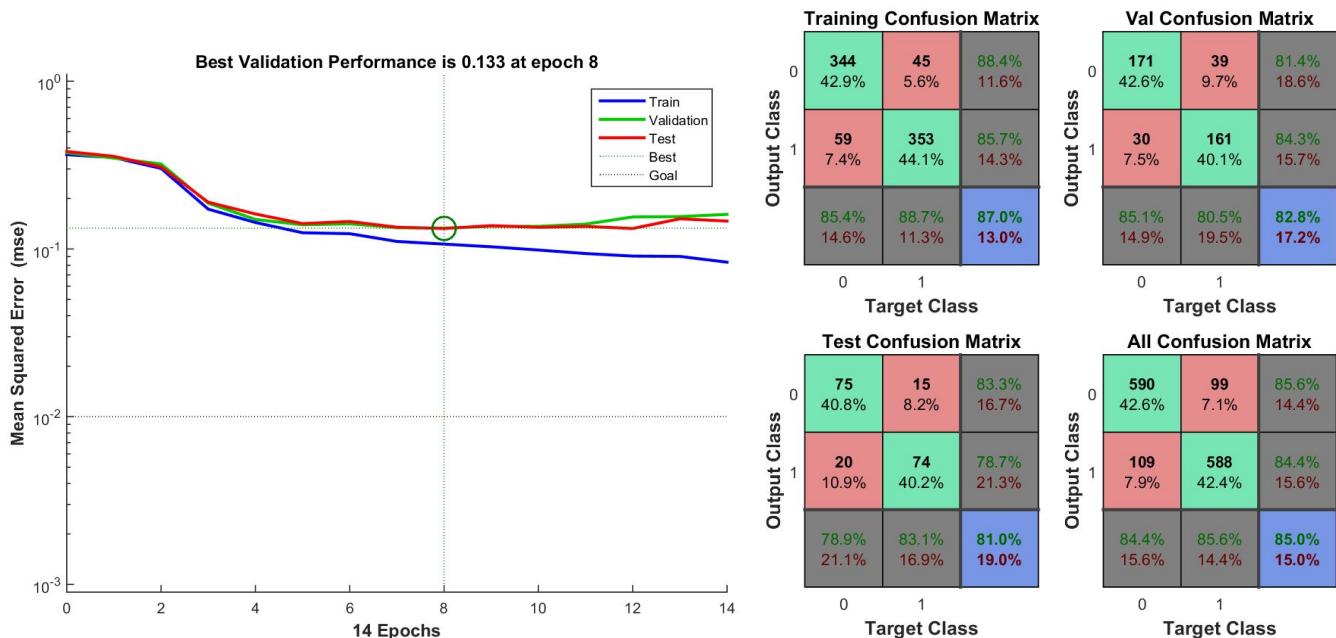


Imagen 3.5.1.1: Gráfico de performance (MSE) vs épocas y matrices de confusión para cada uno de los conjuntos de datos de la mejor simulación (10) para configuración (25+5+5) de red con Algoritmo LM.

- Mejor Arquitectura:

Input	val_corr (main)	val_corr (std)	val_mse (mean)	val_mse (std)
25+5	0.818204	0.015258	0.132651	0.010456
25+5+5	0.811222	0.013458	0.136018	0.007468

Tabla 3.5.2: Promedio y desviación estándar de porcentajes de clasificación correctos y de MSE del conjunto de validación, para configuraciones [29+5] y [29+5+5] de red con Algoritmo LM. La columna naranja indica el criterio de mejor solución (clasificación correcta) y la fila amarilla la mejor arquitectura que en este caso es ([29+5]-6-1).

- Porcentajes de clasificación:

Input	ctr_mean	ctr_std	cval_mean	cval_std	cts_mean	cts_std
25+5	0.856180	0.018375	0.818204	0.015258	0.817391	0.009988
25+5+5	0.854682	0.018770	0.811222	0.013458	0.815217	0.007686

Tabla 3.5.3: Promedio y desviación estándar de porcentajes de clasificación correctos para los conjuntos de entrenamiento, validación y prueba, para configuraciones [29+5] y [29+5+5] de red con Algoritmo LM. Las columnas naranjas identifican el criterio de mejor resultado (clasificaciones correctas) y las filas amarillas las simulación con el mejor resultado, que para este caso es [25+5].

3.6.- Comparación porcentajes de clasificación.

Porcentajes de clasificación (tr, va y ts)

Para cada configuración

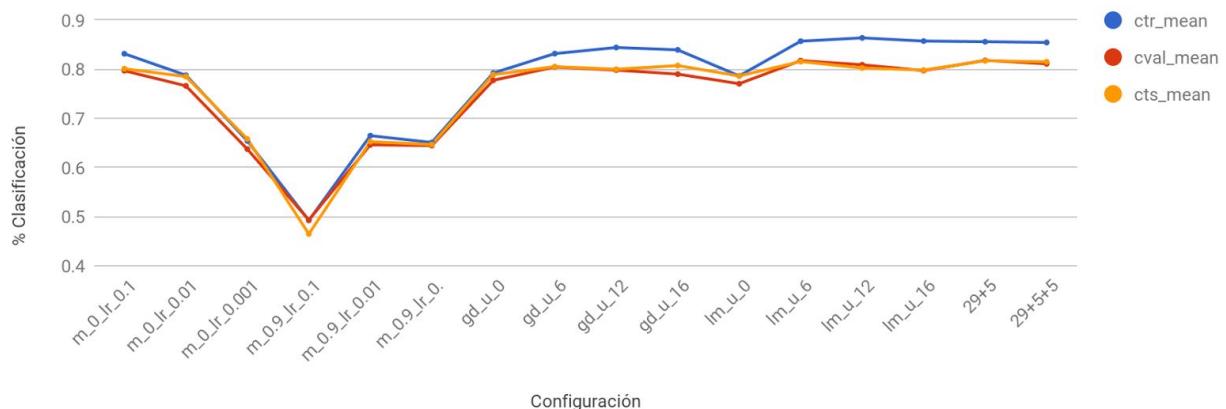


Gráfico 3.6.1: Porcentaje de clasificaciones correctas del conjunto de prueba, para cada una de las configuraciones, donde el mejor resultado para el conjunto de entrenamiento es 0.863920 con lm_u_12, para el conjunto de

validación es 0.818204 con [29+5] y para el conjunto prueba es 0.817 con [29+5], por lo que la mejor red es la con arquitectura ([29+5]-6-1).

3.7.- Clasificación conjunto de prueba *tst.txt*.

Utilizando la mejor red neuronal utilizada hasta el momento, correspondiente a una arquitectura ([29+5]-6-1) con algoritmo LM, se clasificó el conjunto de prueba (*tst.txt*) con un total de 147 individuos, los que carecen de etiqueta. Los resultados de este procedimiento se observan en la Tabla 3.7.1, donde 1: cambio de institución y 0: permanencia en la institución actual.

Nº	C	Nº	C	Nº	C	Nº	C	Nº	C	Nº	C	Nº	C	Nº	C
1	0	21	1	41	1	61	1	81	0	101	0	121	1	141	1
2	0	22	0	42	0	62	1	82	0	102	0	122	0	142	0
3	1	23	1	43	1	63	0	83	0	103	0	123	1	143	1
4	1	24	0	44	1	64	1	84	0	104	0	124	1	144	0
5	0	25	0	45	1	65	0	85	1	105	1	125	1	145	0
6	1	26	1	46	1	66	1	86	0	106	0	126	0	146	0
7	0	27	1	47	0	67	1	87	0	107	0	127	0	147	1
8	1	28	1	48	0	68	1	88	1	108	0	128	0		
9	1	29	1	49	0	69	0	89	1	109	1	129	1		
10	1	30	0	50	1	70	1	90	1	110	1	130	1		
11	0	31	1	51	1	71	1	91	1	111	0	131	1		
12	0	32	1	52	0	72	1	92	1	112	0	132	0		
13	0	33	0	53	1	73	1	93	0	113	1	133	0		
14	1	34	0	54	1	74	0	94	0	114	1	134	0		
15	1	35	1	55	1	75	1	95	0	115	1	135	0		
16	0	36	0	56	0	76	0	96	1	116	1	136	0		
17	1	37	1	57	0	77	1	97	1	117	1	137	0		
18	1	38	1	58	1	78	0	98	0	118	0	138	0		
19	1	39	1	59	0	79	1	99	1	119	1	139	0		
20	0	40	1	60	1	80	0	100	0	120	1	140	0		

Tabla 3.7.1: Resultados de clasificación para cada uno de los 147 ejemplos del conjunto de prueba *tst.txt*.

Distribución Clases

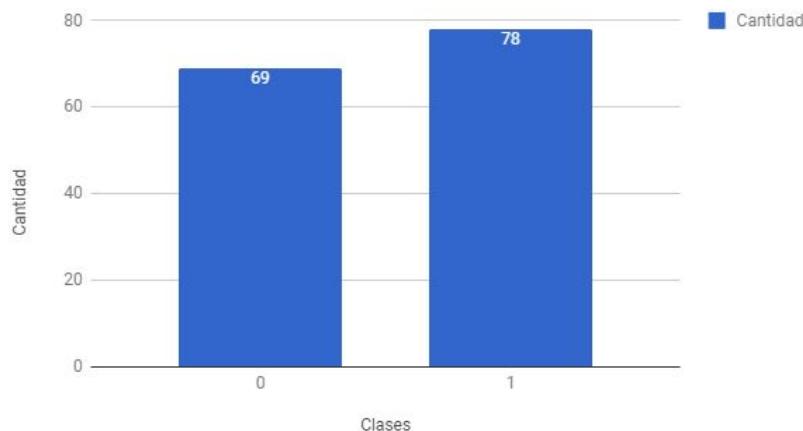


Gráfico 3.7.1: Distribución de clases de resultados de clasificación de conjunto de prueba tst.txt.

4.- Código.

El código (comentado) utilizado para obtener los resultados de clasificación para cada una de las configuraciones de redes neuronales se encuentra en las imágenes (Imagen 4.1 - 4.8)

```
1 function main()
2
3 %Lectura archivos
4 [m_data,m_target,m_data1,m_data2,data_tst,data_tst1,data_tst2] = lectura();
5
6 fclose('all');
7 mkdir 'Plots/'
8 rmdir 'Plots/' s
9
10 %-----Problem 1 (Momentum 0.0 vs 0.9)-----
11 [best_vp1,best_lr1,best_net1] = problem_1(1,'traingdm',m_data,m_target,0); %Momentum 0
12 [best_vp2,best_lr2,best_net2] = problem_1(2,'traingdm',m_data,m_target,0.9); %Momentum 0.9
13
14 %Calculo mejor tasa de aprendizaje y momentum
15 best_lr = best_lr1;
16 best_momentum = 0;
17 if(best_vp2 > best_vp1)
18     best_lr = best_lr2;
19     best_momentum = 0.9;
20 end
21
22 fileID = fopen('Plots/Problem_1/prom_c.csv','a');
23 fprintf(fileID,'\\nbest_lr;best_momentum\\n');
24 fprintf(fileID,'%f;%f \\n',best_lr,best_momentum);
25 fclose(fileID);
26
27 %-----Problem 2 (Unidades Gradiente Descendiente)-----
28 [best_vp3,best_unit1,best_net3] = problem_2_3(2,'trainlm',m_data,m_target,best_momentum,best_lr);
29
30 %-----Problem 3 (Unidades Levenberg-Marquardt)-----
31 [best_vp4,best_unit2,best_net4] = problem_2_3(3,'trainlm',m_data,m_target,0,0);
32
33 %-----Mejor entre 2 y 3-----
34 best_unit = best_unit1;
35 if(best_vp4 > best_vp3)
36     best_unit = best_unit2;
37 end
38
39 %-----Problem 4 (Arquitectura [29+5] vs [29+5+5])-----
40 [best_vp5,best_net5] = problem_4(1,'trainlm',m_data1,m_target,best_unit);
41 [best_vp6,best_net6] = problem_4(2,'trainlm',m_data2,m_target,best_unit);
42
43 %Calculo mejor capa de entrada
44 best_input = 34;
45 if(best_vp6 > best_vp5)
46     best_input = 39;
47 end
48
49 fileID = fopen('Plots/Problem_4/prom_c.csv','a');
50 fprintf(fileID,'\\nbest imput: %d\\n',best_input);
51 fclose(fileID);
52
53
54 %-----Problem 5 (Clasificacion tst.txt con mejor red)-----
55
56 best = [best_vp1 best_vp2 best_vp3 best_vp4 best_vp5 best_vp6];
57 best_n = max(best);
58
59 if(best_n == best_vp1)
60     out = sim(best_net1, data_tst');
61 elseif(best_n == best_vp2)
62     out = sim(best_net2, data_tst');
63 elseif(best_n == best_vp3)
64     out = sim(best_net3, data_tst');
65 elseif(best_n == best_vp4)
66     out = sim(best_net4, data_tst');
67 elseif(best_n == best_vp5)
68     out = sim(best_net5, data_tst1');
69 elseif(best_n == best_vp6)
70     out = sim(best_net6, data_tst2');
71 end
72
73 %Umbral de decision 0.5
74 for i = 1:length(out)
75     num = abs(out(i:i));
76     if(num >= 0.5)
77         out(i:i) = 1;
78     else
79         out(i:i) = 0;
80     end
81 end
82
83 mkdir('Plots/Problem_5/');
84 dlmwrite('Plots/Problem_5/tst_labels.txt',out');
85
86 end
--
```

Imagen 4.1: Código comentado de función main.

```
1 function [m_data,m_target,m_data1,m_data2,data_tst,data_tst1,data_tst2] = lectura()
2
3 %-----
4 list = {'./tr.txt', './va.txt', './ts.txt'};
5 data_tst = importdata('./tst.txt','\t',0);
6
7 m_data = []; m_data1 = []; m_data2 = []; data_tst1 = []; data_tst2 = [];
8 m_target = [];
9
10 %-----Concatenación conjuntos de datos-----
11 for c = list
12     file = c{:};
13     matrix = importdata(file,'\t',0);
14
15     data = matrix(:,1:end-1);
16     target = matrix(:,end);
17
18     m_data = vertcat(m_data, data);
19     m_target = vertcat(m_target, target);
20 end
21
22 %----- (29+5, 29+5+5) -----
23 [m_data1,m_data2] = new_matrix(m_data);
24
25 %-----tst(29+5, 29+5+5) -----
26 [data_tst1,data_tst2] = new_matrix(data_tst);
27
28 end
29
30 function [m1,m2] = new_matrix(m)
31 m1 = [m abs(m(:,13)-m(:,17))];
32 m1 = [m1 abs(m(:,14)-m(:,18))];
33 m1 = [m1 abs(m(:,14)-m(:,17))];
34 m1 = [m1 abs(m(:,14)-m(:,24))];
35 m1 = [m1 abs(m(:,14)-m(:,20))];
36 %-----
37 m2 = [m1 abs(m(:,13)-m(:,24))];
38 m2 = [m2 abs(m(:,13)-m(:,12))];
39 m2 = [m2 abs(m(:,13)-m(:,20))];
40 m2 = [m2 abs(m(:,13)-m(:,23))];
41 m2 = [m2 abs(m(:,13)-m(:,13))];
42 end
```

Imagen 4.2: Código comentado de función lectura.

```

1 function [best_vp,best_lr,best_net] = problem_1(num,net_type,m_data,m_target,momentum)
2
3 mkdir('Plots/Problem_1');
4 fileID = fopen('Plots/Problem_1/results.csv','a');
5 fileID2 = fopen('Plots/Problem_1/prom_mse.csv','a');
6 fileID3 = fopen('Plots/Problem_1/prom_c.csv','a');
7
8 fprintf(fileID, '\nProblem 1.%d: \n',num);
9 fprintf(fileID2, '\nProblem 1.%d: \n',num);
10 fprintf(fileID3, '\nProblem 1.%d: \n',num);
11
12 fprintf(fileID,'Iteration;learning_rate;momentum;p;c_p;trp;c_tr;vp;c_val;tsp;c_ts;ne;be\n');
13 fprintf(fileID2,'Learning_rate;momentum;p_mean;p_std;trp_mean;trp_std;vp_mean;vp_std;tsp_mean;
14 | tsp_std\n');
15 fprintf(fileID3,'Learning_rate;momentum;cp_mean;cp_std;ctr_mean;ctr_std;cval_mean;cval_std;
16 | cts_mean;cts_std\n');
17
18 lr = [0.1 0.01 0.001]; %Tasa de aprendizaje
19
20 best_lr = 0; best_vp = -Inf; best_vp1 = -Inf; %Inicialización variables
21 best_net = patternnet(1); best_net0 = patternnet(1); %Inicialización redes
22
23 for lrate = lr
24 p_p = []; trp_p = []; vp_p = []; tsp_p = []; %Performance por conjunto
25 c_pp = []; c_trp = []; c_valp = []; c_tsp = []; %Porcentaje de clasificación por conjunto
26 best_vp0 = -Inf;
27 for c = 1:10
28 %Simulación red neuronal
29 [p,c_p,trp,c_tr,vp,c_val,tsp,c_ts,ne,be,net] = Neural_Network(net_type,m_data,m_target, ...
30 | momentum,lrate,10);
31
32 %Resultados
33 fprintf(fileID,'%d;%f;%f;%f;%f;%f;%f;%f;%f;%d;%d\n',c,lrate,momentum,p,c_p,
34 | trp,c_tr,vp,c_val,tsp,c_ts,ne,be);
35
36 %Agrupamiento 10 simulaciones
37 p_p = [p_p p]; trp_p = [trp_p trp]; vp_p = [vp_p vp]; tsp_p = [tsp_p tsp];
38 c_pp = [c_pp c_p]; c_trp = [c_trp c_tr]; c_valp = [c_valp c_val]; c_tsp = [c_tsp c_ts];
39
40 folder = sprintf('Plots/Problem_1/Momentum_.1f/Lrate_.3f/Iteration_%d',momentum,lrate,c);
41 file_maker(folder);
42
43 [best_vp0,best_net0] = min(c_val,net,best_vp0,best_net0); %Mejor red
44 end
45
46 %Promedio y desviación estandar de 10 simulaciones (Performance)
47 fprintf(fileID2,'%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f\n',lrate,momentum,
48 | mean(p_p),std(p_p),mean(trp_p),std(trp_p),mean(vp_p),std(vp_p),mean(tsp_p),std(tsp_p));
49
50 %Promedio y desviación estandar de 10 simulaciones (Clasificación)
51 fprintf(fileID3,'%f;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f\n',lrate,momentum,
52 | mean(c_pp),std(c_pp),mean(c_trp),std(c_trp),mean(c_valp),std(c_valp),mean(c_tsp),std(c_tsp));
53
54 [best_vp,best_lr] = min(mean(c_valp),lrate,best_vp,best_lr); %Mejor tasa de aprendizaje
55 [best_vp1,best_net] = min(mean(c_valp),best_net0,best_vp1,best_net); %Mejor red
56 end
57
58 fprintf(fileID3,'\nbest_lr;best_vp\n');
59 fprintf(fileID3,'%f;%f\n',best_lr,best_vp);
60
61 fclose('all');
62
63 end
64
65 %Genera Gráficos y Tablas
66 function file_maker(folder_name)
67 name = {'roc','confusion','error','training','performance'};
68 mkdir(sprintf('%s',folder_name));
69 h = findobj('Type', 'figure');
70 for k = 1:numel(h)
71 if(k<6)
72 print(sprintf('%s/%s',folder_name,name{k}),h(k),'-dpng')
73 end
74 end;
75 close all
76 end
77
78 %Obtiene mejor porcentaje de clasificación.
79 function [a,b] = min(first,value,best,best_value)
80 if(first > best)
81 best = first;
82 best_value = value;
83 end
84 a = best; b = best_value;
85 end

```

Imagen 4.3: Código comentado de función problem_1.

```

1 function [best_vp,best_unit,best_net] = problem_2_3(num,net_type,m_data,m_target,best_momentum,best_lr)
2
3 mkdir (sprintf('Plots/Problem_%d',num));
4 fileID = fopen(sprintf('Plots/Problem_%d/results.csv',num),'w');
5 fileID2 = fopen(sprintf('Plots/Problem_%d/prom_mse.csv',num),'w');
6 fileID3 = fopen(sprintf('Plots/Problem_%d/prom_c.csv',num),'w');
7
8 fprintf(fileID,'nProblem %d: \n',num);
9 fprintf(fileID2,'nProblem %d: \n',num);
10 fprintf(fileID3,'nProblem %d: \n',num);
11
12 fprintf(fileID,'iteration;units;learning_rate;momentum;p;c_p;trp;c_tr;vp;c_val;tsp;c_ts;ne;be\n');
13 fprintf(fileID2,'units;learning_rate;momentum;p_mean;p_std;trp_mean;trp_std;vp_mean;vp_std;tsp_mean;
14 tsp_std\n');
15 fprintf(fileID3,'units;learning_rate;momentum;cp_mean;cp_std;ctr_mean;ctr_std;cval_mean;cval_std;
16 cts_mean;cts_std\n');
17
18 lunit = [0 6 12 16]; %Unidades de capa oculta
19
20 best_unit = 0; best_vp = -Inf; best_vp1 = -Inf; %Inicialización variables
21 best_net = patternnet(1); best_net0 = patternnet(1); %Inicialización redes
22
23 for units = lunit
24     p_p = []; trp_p = []; vp_p = []; tsp_p = []; %Performance por conjunto
25     c_pp = []; c_trp = []; c_valp = []; c_tsp = []; %Porcentaje de clasificación por conjunto
26     best_vp0 = -Inf;
27     for c = 1:10
28         %Simulación red neuronal
29         [p,c_p,trp,c_tr,vp,c_val,tsp,c_ts,ne,be,net] = Neural_Network(net_type,m_data,m_target, ...
30             best_momentum,best_lr,units);
31
32         %Resultados
33         fprintf(fileID,'%d;%d;%f;%f;%f;%f;%f;%f;%f;%d;%d\n',c,units, ...
34             best_lr,best_momentum,p,c_p,trp,c_tr,vp,c_val,tsp,c_ts,ne,be);
35
36         %Agrupamiento 10 simulaciones
37         p_p = [p_p p]; trp_p = [trp_p trp]; vp_p = [vp_p vp]; tsp_p = [tsp_p tsp];
38         c_pp = [c_pp c_p]; c_trp = [c_trp c_tr]; c_valp = [c_valp c_val]; c_tsp = [c_tsp c_ts];
39
40         folder = sprintf('Plots/Problem_%d/m_%1f_lr_%3f_unit_%d/Iteration_%d',num,best_momentum,
41             best_lr,units,c);
42
43         file_maker(folder);
44
45         [best_vp0,best_net0] = min(c_val,net,best_vp0,best_net0); %Mejor red
46     end
47
48     %Promedio y desviación estandar de 10 simulaciones (Performance)
49     fprintf(fileID2,'%d;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f\n',units,best_lr,best_momentum, ...
50         mean(p_p),std(p_p),mean(trp_p),std(trp_p),mean(vp_p),std(vp_p),mean(tsp_p),std(tsp_p));
51
52     %Promedio y desviación estandar de 10 simulaciones (Clasificación)
53     fprintf(fileID3,'%d;%f;%f;%f;%f;%f;%f;%f;%f;%f;%f\n',units,best_lr,best_momentum, ...
54         mean(c_pp),std(c_pp),mean(c_trp),std(c_trp),mean(c_valp),std(c_valp),mean(c_tsp),std(c_tsp));
55
56     [best_vp,best_unit] = min(mean(c_valp),units,best_vp,best_unit); %Mejor cantidad unidades
57     [best_vp1,best_net] = min(mean(c_valp),best_net0,best_vp1,best_net); %Mejor red
58 end
59
60 fprintf(fileID3,'\nbest_unit;best_vp\n');
61 fprintf(fileID3,'%f;%f\n',best_unit,best_vp);
62
63 fclose('all');
64
65 end
66
67 %Genera Gráficos y Tablas
68 function file_maker(folder_name)
69     name = {'roc','confusion','error','training','performance'};
70     mkdir(sprintf('%s',folder_name));
71     h = findobj('Type', 'figure');
72     for k = 1:numel(h)
73         if(k<6)
74             print(sprintf('%s/%s',folder_name,name{k}),h(k),'-dpng')
75         end
76     end;
77     close all
78 end
79
80 %Obtiene mejor porcentaje de clasificación.
81 function [a,b] = min(first,value,best,best_value)
82     if(first > best)
83         best = first;
84         best_value = value;
85     end
86     a = best; b = best_value;
87 end

```

Imagen 4.4: Código comentado de función problem_2_3.

```

1 function [best_vp,best_net] = problem_4(num,net_type,m_data,m_target,units)
2
3 mkdir('Plots/Problem_4');
4 fileID = fopen('Plots/Problem_4/results.csv','a');
5 fileID2 = fopen('Plots/Problem_4/prom_mse.csv','a');
6 fileID3 = fopen('Plots/Problem_4/prom_c.csv','a');
7
8 fprintf(fileID,'\\nProblem 4.%d: \\n',num);
9 fprintf(fileID2,'\\nProblem 4.%d: \\n',num);
10 fprintf(fileID3,'\\nProblem 4.%d: \\n',num);
11
12 fprintf(fileID,'iteration;p;c_p;trp;c_tr;vp;c_val;tsp;c_ts;ne;be\\n');
13 fprintf(fileID2,'p_mean;p_std;trp_mean;trp_std;vp_mean;vp_std;tsp_mean;tsp_std\\n');
14 fprintf(fileID3,'cp_mean;cp_std;ctr_mean;ctr_std;cval_mean;cval_std;cts_mean;cts_std\\n');
15
16 p_p = []; trp_p = []; vp_p = []; tsp_p = []; %Performance por conjunto
17 c_pp = []; c_trp = []; c_valp = []; c_tsp = []; %Porcentaje de clasificación por conjunto
18
19 best_vp0 = -Inf; best_net = patternnet(1); %Inicialización variables y red
20
21 for c = 1:10
22   %Simulación red neuronal
23   [p,c_p,trp,c_tr,vp,c_val,tsp,c_ts,ne,be,net] = Neural_Network(net_type,m_data,m_target,0,0,units);
24
25   %Resultados
26   fprintf(fileID,'%d;%f;%f;%f;%f;%f;%f;%d;%d\\n',c,p,c_p,trp,c_tr,vp,c_val,tsp,c_ts,ne,be);
27
28   %Agrupamiento 10 simulaciones
29   p_p = [p_p p]; trp_p = [trp_p trp]; vp_p = [vp_p vp]; tsp_p = [tsp_p tsp];
30   c_pp = [c_pp c_p]; c_trp = [c_trp c_tr]; c_valp = [c_valp c_val]; c_tsp = [c_tsp c_ts];
31
32   folder = sprintf('Plots/Problem_4/Problem_4.%d/Iteration_%d',num,c);
33   file_maker(folder);
34
35   [best_vp0,best_net] = min(c_val,net,best_vp0,best_net); %Mejor red
36 end
37
38 %Promedio y desviación estandar de 10 simulaciones (Performance)
39 fprintf(fileID2,'%f;%f;%f;%f;%f;%f;%f\\n',...
40   mean(p_p),std(p_p),mean(trp_p),std(trp_p),mean(vp_p),std(vp_p),mean(tsp_p),std(tsp_p));
41
42 %Promedio y desviación estandar de 10 simulaciones (Clasificación)
43 fprintf(fileID3,'%f;%f;%f;%f;%f;%f;%f\\n',...
44   mean(c_pp),std(c_pp),mean(c_trp),std(c_trp),mean(c_valp),std(c_valp),mean(c_tsp),std(c_tsp));
45
46 best_vp = mean(c_valp); %Mejor tasa de clasificación
47
48 fclose('all');
49
50 end
51
52 %Genera Gráficos y Tablas
53 function file_maker(folder_name)
54   name = {'roc','confusion','error','training','performance'};
55   mkdir(sprintf('%s',folder_name));
56   h = findobj('Type','figure');
57   for k = 1:numel(h)
58     if(k<6)
59       print(sprintf('%s/%s',folder_name,name{k}),h(k),'-dpng')
60     end
61   end;
62   close all
63 end
64
65 %Obtiene mejor porcentaje de clasificación.
66 function [a,b] = min(first,value,best,best_value)
67   if(first > best)
68     best = first;
69     best_value = value;
70   end
71   a = best; b = best_value;
72 end

```

Imagen 4.5: Código comentado de función problem_4.

```

1 function [p,corr_p,trp,corr_tr,vp,corr_val,tsp,corr_ts,ne,be,net] = Neural_Network(net_type,
2     train_data, train_target, momentum, lr, layers)
3
4 % Solve a Pattern Recognition Problem with a Neural Network
5 % Script generated by Neural Pattern Recognition app
6 % Created 27-Oct-2017 19:34:32
7 %
8 % This script assumes these variables are defined:
9 %
10 %   train_data - input data.
11 %   train_target - target data.
12
13 x = train_data';
14 t = train_target';
15
16 % Choose a Training Function
17 % For a list of all training functions type: help nntrain
18 % 'trainlm' is usually fastest.
19 % 'trainbr' takes longer but may be better for challenging problems.
20 % 'trainscg' uses less memory. Suitable in low memory situations.
21 %trainFcn = 'trainlm'; % Scaled conjugate gradient backpropagation.
22
23 % Create a Pattern Recognition Network
24 if(layers > 0)
25     hiddenLayerSize = layers;
26 else
27     hiddenLayerSize = [];
28 end
29 net = patternnet(hiddenLayerSize);
30 net.trainFcn = net_type;
31
32 %Params
33 net.trainParam.epochs = 5000;
34 net.trainParam.mc = momentum;
35 net.trainParam.lr = lr;
36 net.trainParam.show = 100;
37 net.trainParam.goal = 0.01;
38
39 % Hiden and Output functions
40 if(layers > 0)
41     net.biases{1}.learnFcn = 'learngdm';
42     net.layers{1}.transferFcn = 'tansig'; % Hiden Layers tranfer function
43     net.layers{2}.transferFcn = 'purelin'; % Output Layer transfer function
44 else
45     net.layers{1}.transferFcn = 'purelin'; % Output Layer transfer function
46 end
47
48 % Choose Input and Output Pre/Post-Processing Functions
49 % For a list of all processing functions type: help nnprocess
50 net.input.processFcns = {'removeconstantrows','mapminmax'};
51 net.output.processFcns = {'removeconstantrows','mapminmax'};
52
53 % Setup Division of Data for Training, Validation, Testing
54 % For a list of all data division functions type: help nndivide
55 net.divideFcn = 'divideblock'; % Divide data by block
56 net.divideMode = 'sample'; % Divide up every sample
57 net.divideParam.trainRatio = 0.5772005772;
58 net.divideParam.valRatio = 0.28932178932;
59 net.divideParam.testRatio = 0.13347763347;
60
61 % Choose a Performance Function
62 % For a list of all performance functions type: help nnperformance
63 net.performFcn = 'mse'; % Mean Squared Error
64
65 % Choose Plot Functions
66 % For a list of all plot functions type: help nnplot
67 net.plotFcns = {'plotperform','plottrainstate','ploterrhist',...
    'plotconfusion', 'plotroc'};
```

Imagen 4.6: Código comentado de función Neural Network (parte1).

```
70 % Train the Network
71 [net,tr] = train(net,x,t);
72
73 % Test the Network
74 y = net(x);
75 e = gsubtract(t,y);
76 performance = perform(net,t,y);
77 tind = vec2ind(t);
78 yind = vec2ind(y);
79 percentErrors = sum(tind == yind)/numel(tind);
80
81 % Recalculate Training, Validation and Test Performance
82 trainTargets = t .* tr.trainMask{1};
83 valTargets = t .* tr.valMask{1};
84 testTargets = t .* tr.testMask{1};
85 trainPerformance = perform(net,trainTargets,y);
86 valPerformance = perform(net,valTargets,y);
87 testPerformance = perform(net,testTargets,y);
88
89 %Tasa de acierto
90 [c_p,cm,ind,per] = confusion(t,y);
91 corr_p = 1-c_p;
92
93 [c_tr,cm,ind,per] = confusion(trainTargets,y);
94 corr_tr = 1-c_tr;
95
96 [c_val,cm,ind,per] = confusion(valTargets,y);
97 corr_val = 1-c_val;
98
99 [c_ts,cm,ind,per] = confusion(testTargets,y);
100 corr_ts = 1-c_ts;
101
102 %Error
103 e1 = gsubtract(trainTargets,y);
104 e2 = gsubtract(valTargets,y);
105 e3 = gsubtract(testTargets,y);
106
107 p = performance;
108 trp = trainPerformance;
109 vp = valPerformance;
110 tsp = testPerformance;
111 tr = tr;
112 ne = tr.num_epochs;
113 be = tr.best_epoch;
114
115 % View the Network
116 %view(net)
117
118 %Plots
119 h = figure;
120 h; plotperform(tr);
121 set(h, 'Visible', 'off');
122
123 h = figure;
124 h; plottrainstate(tr);
125 set(h, 'Visible', 'off');
126
127 h = figure;
128 h; ploterrhist(e1,'Training',e2,'Validation',e3,'Test');
129 set(h, 'Visible', 'off');
130
131 h = figure;
132 h; plotconfusion(trainTargets,y,'Training',valTargets,y,'Val',testTargets,y,'Test',t,y,'All');
133 set(h, 'Visible', 'off');
134
135 h = figure;
136 h; plotroc(trainTargets,y,'Training',valTargets,y,'Val',testTargets,y,'Test',t,y,'All');
137 set(h, 'Visible', 'off');
```

Imagen 4.7: Código comentado de función Neural Network (parte2).

```
138
139 % Deployment
140 % Change the (false) values to (true) to enable the following code blocks.
141 % See the help for each generation function for more information.
142 if (false)
143     % Generate MATLAB function for neural network for application
144     % deployment in MATLAB scripts or with MATLAB Compiler and Builder
145     % tools, or simply to examine the calculations your trained neural
146     % network performs.
147     genFunction(net,'myNeuralNetworkFunction');
148     y = myNeuralNetworkFunction(x);
149 end
150 if (false)
151     % Generate a matrix-only MATLAB function for neural network code
152     % generation with MATLAB Coder tools.
153     genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
154     y = myNeuralNetworkFunction(x);
155 end
156 if (false)
157     % Generate a Simulink diagram for simulation or deployment with
158     % Simulink Coder tools.
159     gensim(net);
160 end
```

Imagen 4.8: Código comentado de función Neural Network (parte3).

5.- Conclusiones.

Según los datos obtenidos en cada uno de los experimentos realizados anteriormente, se concluye que para una red neuronal con gradiente descendente la tasa de aprendizaje y momentum que entrega un mayor porcentaje de clasificación correcta 0.797007 y menor mse 0.1472 son (0.1 y 0.0) por lo que el uso de valores de momentum mayores al igual que tasas menores, empeoran el desempeño de los mismos.

Respecto a los resultados de redes neuronales implementadas con Gradiente Descendiente (GD) y Levenberg-Marquardt (LM), utilizando tasa de aprendizaje 0.1 y momentum 0.0 (los mejores), se concluyó que el valor óptimo de unidades dentro de la capa oculta de la red para el conjunto de datos es 6. Donde para GD el porcentaje de aciertos es 0.80473 y su mse es 0.143535, mientras que para LM (considerado el mejor entre ambos) el porcentaje de aciertos es 0.817706 y su mse es 0.131205.

Los resultados de redes neuronales implementadas Levenberg-Marquardt (LM), arquitecturas ([29+5]-N-1) y ([29+5+5]-N-1), donde N=6 (mejor cantidad de unidades obtenida de los experimentos anteriores), se concluyó que la mejor arquitectura es ([29+5]-6-1) con un porcentaje de aciertos de 0.818204 y un mse de 0.132651.

Por último, al comparar el porcentaje de clasificación para cada uno de las configuraciones de redes neuronales, para el conjunto de entrenamiento, validación y prueba, el mejor resultado para el conjunto de entrenamiento es 0.863920 con lm_u_12, para el conjunto de validación es 0.818204 con [29+5] y para el conjunto prueba es 0.817 con [29+5], por lo que la mejor red la de arquitectura ([29+5]-6-1) para el conjunto de datos.

Levenberg-Marquardt supone una mejora considerable con respecto al resto de configuraciones de red neuronal realizados en los experimentos anteriores, puesto que se pueden obtener buenos resultados (bajo error y alto porcentaje de clasificación) con una pequeña cantidad de épocas. Cabe destacar que a nivel de desempeño en términos de mse de los conjuntos de validación y prueba aumentaron con la implementación progresiva de cada una de las configuraciones, empeoraron respecto al comportamiento del conjunto de entrada, sin embargo, los mejores resultados en cada uno de los problemas corresponden a los que tienen un performance mucho más equilibrado que el resto (a pesar de ser peores que los primeros de gradiente descendiente).

5.- Referencias.

- McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics*. 5 (4): 115–133. doi:10.1007/BF02478259.
- Gradient descent with momentum (traingd). (n.d.). Retrieved November 07, 2017, from <https://www.mathworks.com/help/nnet/ref/traingdm.html>.
- Levenberg-Marquardt backpropagation. (n.d.). Retrieved November 07, 2017, from https://www.mathworks.com/help/nnet/ref/trainlm.html?searchHighlight=Levenberg-Marquardt&s_tid=doc_srchtile.