Trabajo de Aplicación Redes Neuronales

Yorguin José Mantilla cc. 1127617499 y Alejandra Zuleta González cc.1037664084

31 de agosto de 2021

1 Conjunto de datos

La base de datos corresponde a la Wisconsin Breast Cancer Database. El objetivo es identificar, a partir de una red neuronal, dos clases: no-cáncer (1) y cáncer (-1) que corresponden a la presencia de la enfermedad en una persona. Se busca predecir la etiqueta a partir de 9 características por caso. Las características de cada caso son numéricas de valor entero. No están normalizadas y no tienen nombre en sus etiquetas. Por ende, el proceso de normalización se realizó dentro de los algoritmos de clasificación implementados.

2 Arquitecturas Implementadas

Se exploraron dos arquitecturas de redes neuronales; una primera de una sola neurona, y una segunda multicapa. Tanto la representación de las redes como los algoritmos de entrenamiento se hicieron en Matlab, con el código desarrollado por los integrantes. A continuación podemos ver estas arquitecturas.

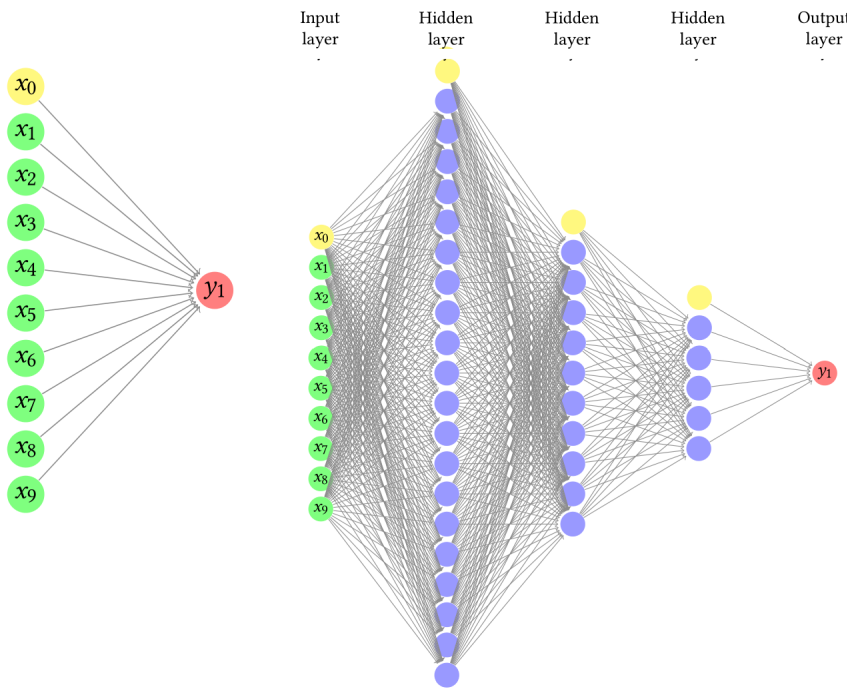


Figura 1: Una sola neurona y multicapa

3 Redes de una neurona

Para la red de una neurona se utilizan 9 pesos para cada una de las características de entrada y un peso adicional para el bias (x_0). La salida es una clasificación binaria.

La red se entrena tanto con un algoritmo de entrenamiento de tipo **LMS** (Least Mean Squares) como de tipo **Perceptron**.

La función de activación de la neurona de salida es la función identidad. Sin embargo, como el objetivo es clasificar, resulta necesario además utilizar la función de signo que discrimina entre 1 y -1.

3.1 Parámetros de entrada para el entrenamiento

El algoritmo de entrenamiento desarrollado para las redes de una sola neurona recibe los siguientes parámetros de entrada.

Parámetro	Significado
X	Matriz de datos de forma número de casos × número de características.
Y	Etiquetas de los casos. Deben ser 2 etiquetas numéricas de signo contrario: Forma: num. casos × 1.
μ	Tamaño del paso o tasa de aprendizaje.
E_{min}	Porcentaje de error máximo aceptable.
IterationsMax	Numero de iteraciones maximas de entrenamiento para tratar de cumplir con E_{min}
tipo	Para escoger entre algoritmo LMS y algoritmo Perceptron.
$NumSetsCrossValidation$	Numero de grupos a utilizar en la validación cruzada. Si es 1 no hay validación cruzada.
t_{max}	Iteraciones minimas sin equivocacion para actualizar los pesos de bolsillo (solo perceptron).
$seed$	Semilla del generador de numeros aleatorios para mantener reproducibilidad del algoritmo.
$(Weight_{min}, Weight_{max})$	Rango de los pesos iniciales generados de forma aleatoria.
$testingDataPercent$	En caso de no usar validacion-cruzada indica el % correspondiente al conjunto de prueba.

3.2 Condición de convergencia

El entrenamiento se realiza mientras se alcanza el número máximo de iteraciones o se alcanza un valor de porcentaje de error menor a E_{min}.

Para el calculo del porcentaje de error sobre N casos en la iteración k del algoritmo se implementó el error cuadrático medio $E_k = \frac{1}{2}\sum_i^N (y(i) - z_k(i))^2$.

En particular, como las salidas esperadas son variables numéricas (1, -1), por cada error ocurrido en la sumatoria anterior se tendrá una diferencia de 2 (y un cuadrado de 4). Finalmente se dividirá por 2 y E_k será igual al doble de la cantidad de errores cometidos en la iteración k. El numero máximo de errores por iteración corresponde al caso en que para todos los datos de entrada de un resultado erróneo, de forma que: $E_{max} = 2 \cdot$ # de casos usados

Así, al realizar el cociente entre los errores cometidos por iteración y el número máximo de errores posibles se obtiene la porción de errores del total, es decir, se puede interpretar como el porcentaje de errores: $E\,\% = E/E_{max}$.

3.3 Umbral

Las clases o etiquetas de salida deben ser dos y deben estar asociadas a dos números de diferente signo. Esto es debido a que la función con la cual se realizó el umbral fue la función signo que umbraliza sobre 0.

3.4 Perceptron de bolsillo

En particular, para el caso del algoritmo **Perceptron** se implementó la técnica del bolsillo. Se evidencia que es útil para hacer una exploración mas inteligente del espacio de búsqueda de los pesos. Lo anterior para obtener un menor error en los resultados. Usar el perceptron de bolsillo da la libertad de tener un tamaño de paso mas grande para saltar fácilmente de mínimos locales sabiendo que tendremos una copia de la mejor versión de los pesos (y bias) guardada en memoria. En el caso de que la exploración con un una tasa de aprendizaje grande no converja, es decir, que no se alcance el valor mínimo esperado o que se alcance el número máximo de iteraciones, se podrá retornar al mejor mínimo local hallado durante el proceso de entrenamiento.

3.5 Resultados y conclusiones sobre dependencia sobre de los datos

Se implementó adicionalmente la validación cruzada para observar la dependencia entre la partición de datos y el entrenamiento de la red. Esto se realizó mediante la función crossvalind de Matlab.

Para estas ejecuciones se obtienen los siguientes resultados:

Con LMS:

Fold	Iterations	ErrorTraining	ErrorTesting
1	200k	0.1328	0.0906
2	200k	0.0723	0.0146
3	200k	0.0624	0.0265
4	200k	0.0918	0.0556
-	Promedio	0.0898	0.0486
-	Desv.Std.	0.0311	0.0339

Con perceptron:

Fold	Iterations	ErrorTraining	ErrorTesting
1	200k	0.0273	0.0088
2	200k	0.0254	0.0146
3	200k	0.0273	0.0088
4	200k	0.0195	0.0175
-	Promedio	0.0249	0.0124
-	Desv.Std.	0.0037	0.0044

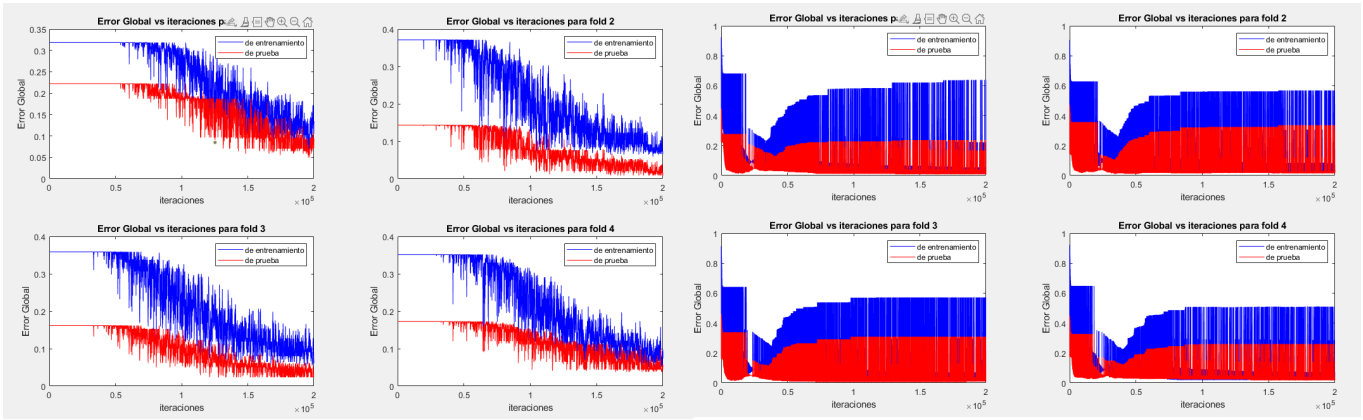


Figura 2: A la izquierda resultados por fold para LMS y a la derecha con perceptron ambos con $\mu = 0{,}005$, $E_{min} = 0{,}003$

El maximo de iteraciones se coloco en 200000 y el error maximo aceptable en 0.003 (0.3 %) ya que queriamos explorar a fondo las capacidades del algoritmo de entrenamiento. Pensamos que quizas llegariamos a un overfitting sin embargo obtuvimos en cambio un buen valor de error de entrenamiento y prueba pesar de que el algoritmo no pudo cumplir con el error aceptable en las iteraciones dispuestas. Es de notar que el comportamiento de las curvas de aprendizaje del perceptron es mucho mas inestable. Esto se discutira mas adelante. Asi mismo, cabe aclarar que errores comparables en el perceptron se pueden lograr con solo 65k iteraciones y $E_{min} = 0{,}03$, es decir, en realidad aquí estamos exprimiendo al máximo las iteraciones debido a la rapidez del entrenamiento; pero no es necesario para tener buenos errores.

Según los resultados anteriores, se puede concluir que los datos de entrada están uniformemente escogidos por fold, debido a que el valor de error de prueba no varía notablemente por fold y la desviación estándar del error en ambos casos es baja. Es de notar que la estabilidad del error es mas fuerte en el perceptron (para este caso particular).

3.6 Conclusiones sobre en torno a la generalización

Como se observa en los resultados de la sección anterior, los errores de prueba son menores a los errores de entrenamiento, por lo que no se evidencia un caso de underfitting (es decir, la red fue entrenada lo suficiente como para obtener buen performance en los casos de prueba). Además tampoco es mucho menor el error de entrenamiento al error de prueba por fold por lo que tampoco se evidencia que haya overfitting. La red entrenada clasifica suficientemente bien los casos de prueba como los de entrenamiento.

Es particularmente curioso que en la mayoria de los casos el error de prueba fue menor que el de entrenamiento. En cierta manera, lo anterior es anti-intuitivo; no tenemos un suposicion clara acerca de porque esto podria haber pasado.

3.7 Conclusiones sobre la Tasa de Aprendizaje

3.7.1 Interpretación

La tasa de aprendizaje determina la cantidad de cambio que se introduce a los pesos al actualizarlos. Como tal representa un porcentaje que multiplica a otra cantidad que varia según el algoritmo de entrenamiento. Por ejemplo, para el **Perceptron** la actualización de los pesos se calcula sobre el producto entre X y Y, mientras que en el **LMS** la cantidad en la que se actualizan los pesos es una fracción del error. Esta diferencia conlleva a que un valor de μ de 0.1 pueda representar cambios muy grandes o muy pequeños dependiendo del rango de los valores que den las cantidades por las que se multiplica. En la figura 3 se puede ver como $\mu = 0{,}2$ origina comportamientos distintos entre el algoritmo **LMS** y el algoritmo **Perceptron**.

3.7.2 Los saltos en las curvas de aprendizaje

Una tasa de aprendizaje que represente cambios bruscos en los pesos puede conllevar a un comportamiento aparentemente caótico en las gráficas del error de entrenamiento y prueba a medida que avanzan las iteraciones. Esto se ve debido a los numerosos saltos del error, a veces de una iteración a otra. Esto quiere decir que los pesos cambiaron tanto que la red cambio su comportamiento (o mejor dicho, nuestra observación de este a través del error). Lo anterior puede dificultar un aprendizaje de convergencia suave pero puede permitir explorar de forma mas diversificada el espacio de los pesos (Fig. 4 Izquierda.). Por el contrario, a medida que la taza de aprendizaje es menor el algoritmo puede explorar de forma mas localizada y los saltos disminuyen a medida de que se aproxima a algún punto de silla de la función que se optimiza (Fig. 4 Centro.). Por ultimo, si la tasa de aprendizaje es suficientemente pequeña, la convergencia es mas suave y la búsqueda muy localizada (lo que puede desaprovechar otros mínimos mejores),y además la cantidad de iteraciones necesarias aumenta (Fig. 4 Derecha.).

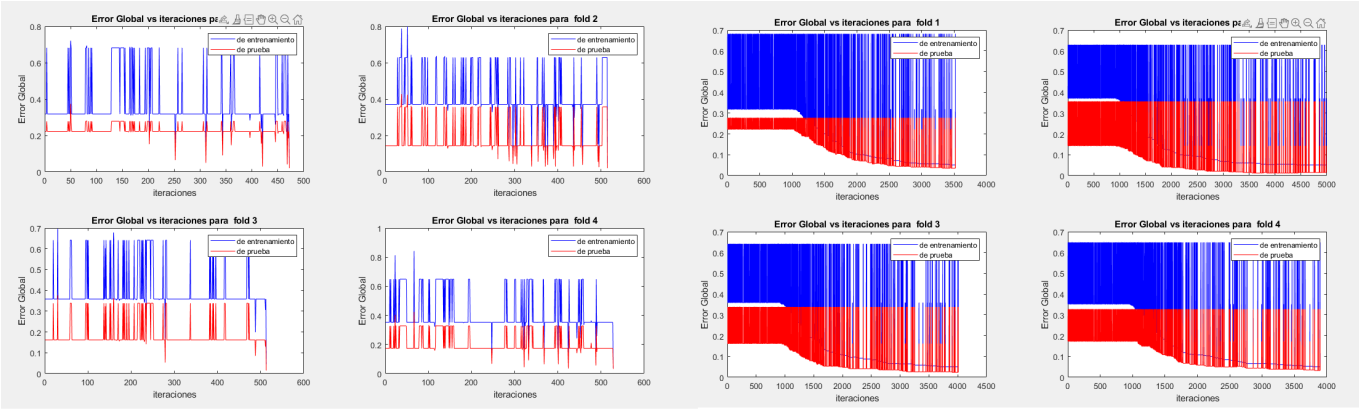


Figura 3: Con $\mu = 0{,}2$ se tiene un comportamiento con menos saltos en el algoritmo LMS, en comparación con el algoritmo Perceptron. Esto debido a lo que representan las respectivas cantidades que multiplican en cada uno de los algoritmos.

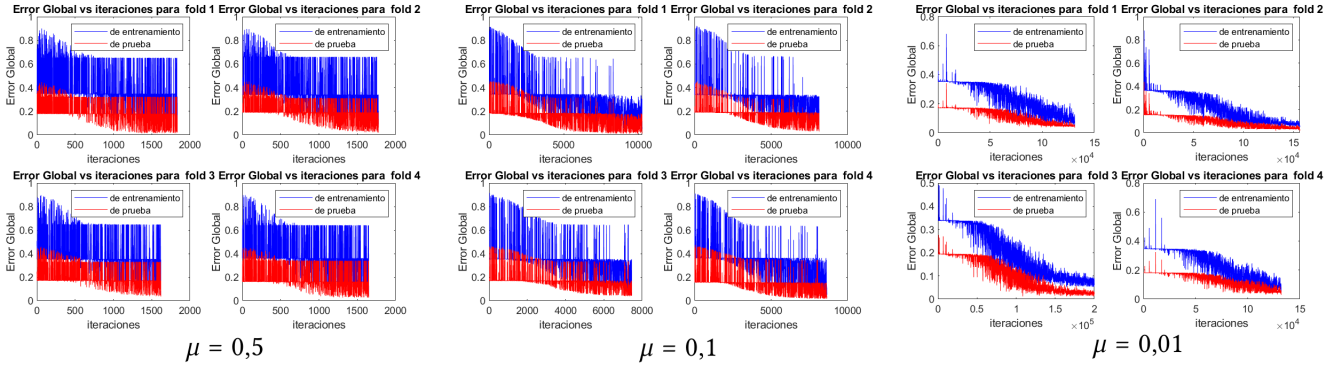
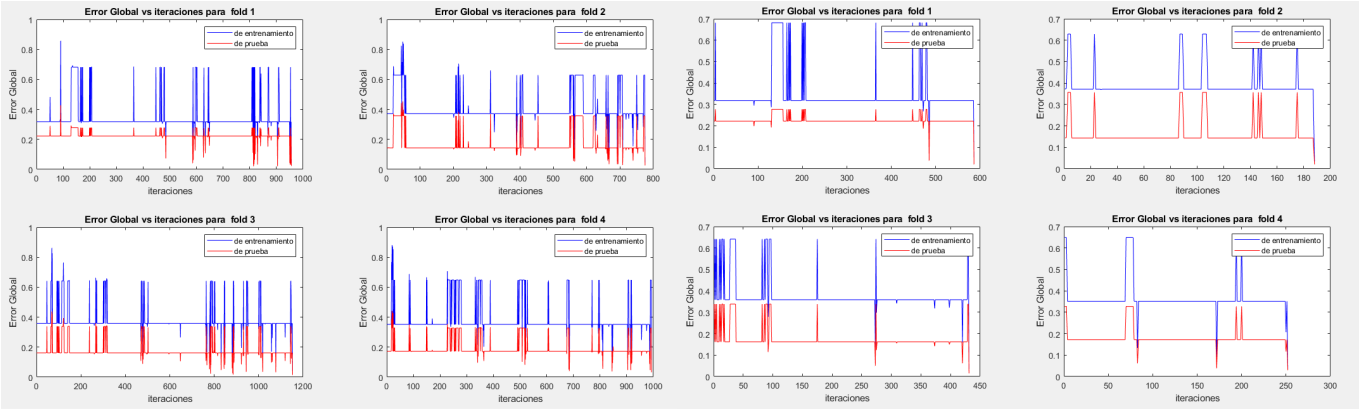


Figura 4: Influencia de la tasa de aprendizaje sobre los saltos de la función de error a lo largo de las iteraciones. Los 3 casos se dan a partir del algoritmo LMS con pesos iniciales entre -10 y 10.

3.8 Conclusiones sobre la influencia de la Inicialización

El valor inicial dado a los pesos puede ser determinante en el resultado final. En este caso vimos que si inicializamos los pesos entre -1 y 1 (nuestra intuición inicial para los pesos) la red se entrena mas lento y generalmente con peores resultados en comparación con la inicialización con pesos entre -0.1 y 0.1 . Es claro pues, que inicializar cerca de algún (buen) mínimo es ventajoso; pero dicha eventualidad es prácticamente suerte.



A la izquierda resultados con pesos iniciales entre -1 y 1 y a la derecha entre -0.1 y 0.1. Ambos son LMS con $\mu = 0{,}1$, $E_{min} = 0{,}05$

4 Red neuronal multicapa

Utilizamos el algoritmo de gradiente descendiente, donde el gradiente es calculado mediante la técnica de backpropagation. Seguimos el modelo matemático del procedimiento descrito en el libro Neural Networks and Deep Learning (Michael Nielsen). Implementamos sobre esto la validación cruzada y un peso de bolsillo donde guardamos continuamente la mejor versión de los pesos (y biases) hasta el momento; para así retornar el mejor resultado al finalizar. La condición de convergencia sigue siendo la obtención de al menos el error máximo aceptable o una cantidad máxima de iteraciones.

4.1 Parámetros de entrada

A diferencia de los algoritmos de las redes de una sola neurona en este caso no se requiere entrar el $tipo$ ni la variable t_{max}. Además del resto de parámetros de entrada utilizados en las redes de neurona única, se agregó el parametro $NeuronsPerLayer$ que es un vector fila con la cantidad de neuronas en cada capa oculta. Si este parámetro es un vector vació ([]) se tiene una sola neurona de salida conectada a las entradas. Para la función de activación se utilizó la función sigmoide, excepto en la capa de salida donde es la función identidad; la cual es posteriormente umbralizada.

4.2 Selección número de capas y neuronas por capa

Este proceso no se encuentra estandarizado por lo que se realizaron diferentes pruebas con distinto número de capas y de neuronas por capa, teniendo en cuenta algunas recomendaciones encontradas, como por ejemplo utilizar para la primera capa una cantidad de neuronas igual al doble de la cantidad de entradas. En general, se pueden encontrar muchas heurísticas alrededor de la selección de capas y número de neuronas; pero el ensayo y error es predominante. Nos decidimos por una arquitecturas de capas ocultas de 20,10 y 5 neuronas como se muestra en la figura 1. Intentamos entrenar con arquitecturas mas grandes para experimentar el caso de overfitting pero no lo logramos.

4.3 Resultados y conclusiones acerca de la Generalización

Análogamente al proceso realizado en los algoritmos de neurona única, se implementó la validación cruzada para analizar la diferencia de los errores de prueba y entrenamiento y la relación de los resultados del algoritmo con los datos de entrada. En las ejecuciones no se envidencia overfitting ni underfitting como en el caso de las redes anteriores. Se evidencia sin embargo cierta dependencia en los datos ya que el error de training estuvo entre el 21 % y el 2 %. Para entrenar se utilizó μ = 0,1 y E_{min} = 0,03.

Fold	Iterations	ErrorTraining	ErrorTesting
1	747	0.0293	0.0293
2	1000	0.1365	0.1294
3	1000	0.0449	0.0585
4	1000	0.2090	0.2339
-	Promedio	0.1049	0.1128
-	Desv.Std.	0.0839	0.0910

4.4 Conclusiones acerca de la inicialización

Al ejecutar el algoritmo con una arquitectura multicapa se notó lo influyente que es el proceso aleatorio de la inicialización en el esfuerzo computacional que puede verse reflejado en el número de iteraciones. Se evidenció al trabajar con 4 capas con 10, 20, 10 y 5 neuronas por capa e inicializando el proceso con diferentes semillas que el número de iteraciones podría variar hasta en dos ordenes de magnitud. Por otra parte, a diferencia del perceptron se encontró que una inicialización de los pesos en un rango más amplio (-10 y 10) acelera la convergencia del algoritmo. Esto sin embargo, es simplemente algo particular de estos datos mas que una tendencia general. Se puede concluir sin embargo que no hay una relación entre los pesos que podrían necesitar distintas arquitecturas de red, son simplemente distintos.

5 Conclusiones generales

- Para el conjunto de datos trabajado se ve que la implementación de red de una sola neurona es suficiente para realizar una buena clasificación. Lo que indica que la mayoría de los datos son linealmente separables en la dimensión del espacio de sus características.
- A nivel general nuestro modelo preferido para este dataset es el perceptron ya que dió buenos resultados con independencia de los datos y su entrenamiento fue lo suficientemente rapido como para explorar 200k iteraciones. Lo anterior no sorprende ya que el perceptron esta optimizado para la clasificacion binaria.
- Una ventaja que tiene la red de una sola neurona es que su baja complejidad la hace mas interpretable; lo cual en contextos clínicos puede ser un factor decisivo para escoger la arquitectura de la red.
- No se justifica el esfuerzo computacional que provee una red multicapa para este ejercicio de clasificación debido a que errores de la misma magnitud se pueden obtener en con menos complejidad de red. Mas grande no es siempre mejor.