Análisis de una MLP (Multi-Layer Perceptron)

Broggi Cesar, Campuzano Matias, Carrizo Juan, Cerutti Lucia, Ocampo Alejandro, Zoloaga Gonzalo

Inteligencia Artificial
Universidad Tecnológica Nacional
Resistencia (Chaco), Argentina
{cesar.broggi, matiasmartincampuzano16, juancarrizoast, lucia.cerutti27, aleocampoiza97, gonzalozoloaga44}@gmail.com

Abstract. El reconocimiento de patrones es una actividad que permite clasificar elementos en determinadas clases. El desarrollo exponencial de la tecnología permite ir desplazando la actividad de las personas a través de la automatización de tareas. Es aquí donde entra la Inteligencia Artificial, que busca la replicación de la consciencia y actividades humanas. El perceptrón multicapas es una arquitectura que permite automatizar este proceso. Este artículo presenta esta arquitectura para el reconocimiento de patrones sencillos y comprobar su funcionamiento con el reconocimiento de las letras 'b', 'd' y 'f'.

Keywords: Entrenamiento de una MLP, Reconocimiento de Patrones, Clasificación de caracteres, Accuracy.

1 Introducción

Los problemas de clasificación fomentaron la creación de un procedimiento que permite analizar las características de algo para buscar patrones, y así poder catalogarlos en alguna clase específica.

Se pueden encontrar diversos patrones en el día a día. Analizando diferentes estilos de natación, por ejemplo, se pueden diferenciar los tipos de nado según los movimientos que realice la persona. Otro ejemplo podría ser el correo electrónico, en el cual se analizan las características del correo recibido para determinar si este es spam o no.

Para poder lograr esto, se puede recurrir a métodos de aprendizajes supervisados o no supervisados; donde en el primero se debe considerar según una entrada cuál es la salida esperada y en el segundo se debe representar lo mejor posible las entradas para que refleje la estructura y poder buscar patrones en esa misma entrada [1].

Este artículo plantea una solución para el reconocimiento y la clasificación de las letras 'b', 'd' y 'f' (en minúsculas) con un método de aprendizaje supervisado. Para esto se destinaron las secciones Métodos y Solución Propuesta, en donde se explica el

procedimiento desarrollado. Luego se muestran los resultados y una simulación, y por último se presenta una conclusión del trabajo.

2 Métodos

Las redes neuronales están inspiradas en el funcionamiento del cerebro humano, en donde se manejan las conexiones de las neuronas de la red para conseguir el resultado esperado.

Esto se puede lograr gracias a que cada conexión entre las neuronas tiene asociado un peso, el cual se va ajustando en su aprendizaje. Las redes van a estar compuestas por varias capas, en donde cada una permite construir nuevas características teniendo en cuenta las que se trataron en la capa anterior [2].

En resumidas cuentas, funcionan mediante una primera capa que recibe los valores de entrada (inputs) y el resto de capas reciben los valores de la capa anterior y realizan una suma ponderada de todos los valores de entrada teniendo en cuenta un bias, o también llamado sesgo.

Para evitar que toda la red neuronal se pueda reducir a una simple regresión lineal, se aplica una función conocida como función de activación. El resultado de esta función es el resultado de la neurona.

2.2 El Perceptrón y su Modelo Multi-capa

El perceptrón [1, 4] es un modelo de red neuronal, en donde lo más básico que se puede representar sólo está formado por dos capas: una capa de entrada y una capa de salida. Pero ese tipo de redes presentan grandes limitaciones, debido a que realizan las proyecciones separando linealmente a los patrones.

Esa separación lineal solo permite reconocer entre dos clases diferentes, pero esto cambia si se ocupan más capas (capas ocultas). Al tener más de dos capas se logra un modelo más complejo pero más potente, realizando proyecciones que permiten resolver problemas que no son linealmente separables.

Para poder simular el funcionamiento de la neurona, se necesita evaluar las conexiones con su peso asociado, por lo cual se utiliza el cálculo de net [1,4]:

$$net_{pj}^{h} = \sum_{i=0}^{N} w_{ji}^{h} * x_{pi} \quad (1)$$

Donde h es la magnitud de capa oculta, p es el patrón n y j la j-ésima neurona. x se refiere a la entrada en el caso de que sea la primera capa oculta o a la salida de la anterior capa en el resto de capas. Esta salida está dada por [4]:

$$y_{pj} = f(net_{pj}) \quad (2)$$

2.3 Backpropagation

Backpropagation [1, 3] es un algoritmo de aprendizaje supervisado que busca ajustar los pesos de las conexiones entre las neuronas y el bias, para ello utiliza el método del descenso del gradiente.

Este ajuste permite analizar cómo va cambiando el comportamiento de la red y se realiza desde la salida hasta las entradas, por eso el nombre de backpropagation. El proceso consiste en dos pasos que se repiten hasta obtener los resultados deseados.

Paso 1: se aplica un patrón de entrada y se calcula capa a capa hasta obtener un conjunto salida. Para calcular esta salida se aplica la fórmula (1).

Paso 2: Se recalculan los pesos a partir de una regla de ajuste de error. Esto significa calcular el valor de una función error, la cual compara el valor obtenido con el deseado, y luego lo propaga hacia atrás [1].

El valor del error en la salida va a estar dado por [4]:

$$\delta_{pk} = (d_{pk} - y_{pk}) \cdot f^{\cdot\cdot}(net_{pj}) \tag{3}$$

Donde d_{nk} es la salida deseada para la neurona k e y_{nk} es la salida obtenida. Luego para calcular el error el error de las neuronas de las capas ocultas se utiliza [4]:

$$\delta_{pj} = (\sum_{k} \delta_{pk} w_{kj}) \cdot f'(net_{pj}) \tag{4}$$

Donde k cubre el número de neuronas con las que está conectado j. Para recalcular los pesos se utiliza [4]:

$$w_{ji}(t+1) = w_{ji}(t) + \alpha\delta_{pj} y_{pi} + \beta\Delta w_{ji}(t) \tag{5}$$

En el cual δ_{pj} está definido según si es una capa de salida o una capa oculta, respectivamente utilizando la ecuación (3) o la ecuación (4), siendo α el coeficiente de aprendizaje.

El último segmento de la ecuación (5) define el momento, donde β representa el coeficiente de momento, tomando valores de 0 a 1, y $\Delta w_{ji}(t)$ representa la variación del peso en las últimas dos iteraciones. Por lo que, el término momento sirve para poder acelerar la convergencia mirando las variaciones de los pesos [4].

Para poder determinar el progreso del aprendizaje se utiliza ECM (error cuadrado medio) a partir de (6) [4]:

$$Ep = \frac{1}{2} \sum_{j=1}^{N} \delta_{pj}^2 \tag{6}$$

Pero para entrenar a la red se usa el error global, que da un promedio de los errores cometidos para cada patrón, este se utiliza para determinar el progreso del

entrenamiento, donde si es menor a un mínimo aceptable propuesto, se deja de realizar más iteraciones [4].

$$Eg = \frac{1}{L}\sum Ep \tag{7}$$

3 Solución Propuesta

El modelo propuesto tiene la intención de reconocer las letras ‘b’, ‘d’ y ‘f’ con hasta un 30% de distorsión en las mismas. Por lo que utiliza una matriz de 10x10 para describirlas, la cual tienen dos valores posibles en su celdas: un valor de 1 para formar la letra y un valor de 0 en donde no corresponda. Un ejemplo de esto se encuentra en la Figura 1.

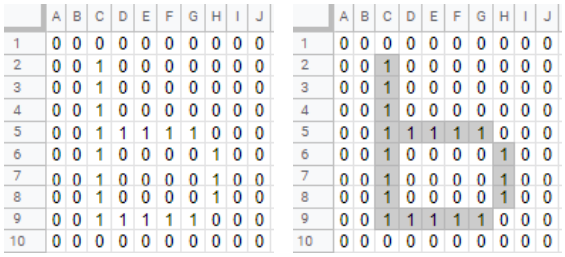

Figura 1. Letra b representada en una matriz.

Pero para poder manejar los datos se debe pasar esta matriz a un arreglo lineal, en el cual se agregan tres elementos más al final del mismo para indicar la clase a la cual pertenece la letra; como es aprendizaje supervisado, en la entrada se proporciona tanto el dato de entrada como la salida esperada del mismo. Tomando la Figura 1 como ejemplo, en la Figura 2 se presenta el arreglo lineal de la letra ‘b’.

Figura 2. Letra ‘b’ representada en un arreglo lineal.

La salida se considera que puede ser de tres tipos, por lo que se trabaja con el siguiente orden para identificarlas: ‘b’, ‘d’ y luego ‘f’.

El esquema simplificado del modelo propuesto se encuentra en la Figura 3. Para poder inferir correctamente, el modelo debe realizar un entrenamiento y testeo del mismo, en donde el dataset con el cual trabaja está divido en tres y están formados por arreglos lineales que representan las letras; en la sección Simulación y Resultados hay mayor detalle sobre estos.

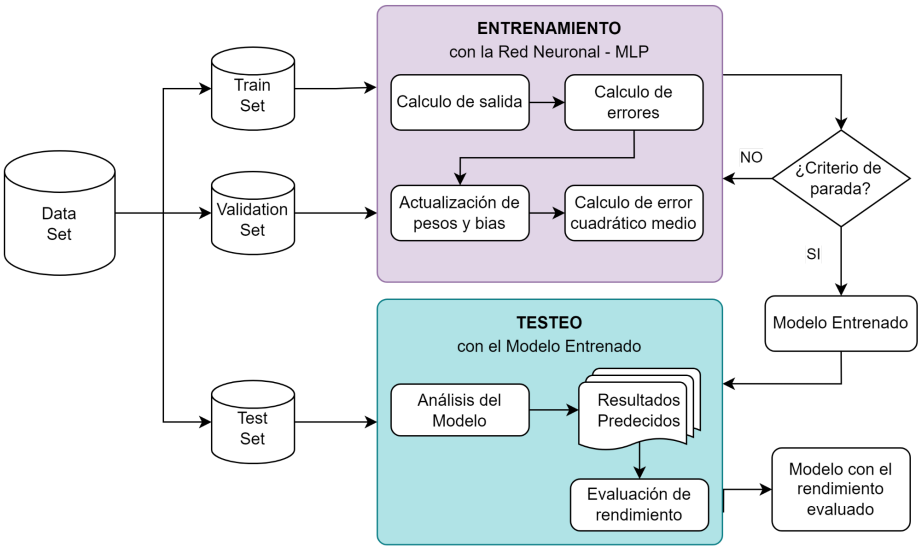

Figura 3. Descripción del modelo propuesto.

Esta red neuronal presenta 100 neuronas en la capa de entrada para poder tratar el dato ingresado junto con su salida esperada, y una capa de salida con 3 neuronas para reconocer a qué letra pertenece el dato ingresado. Esto se puede ver ilustrado en la Figura 4.

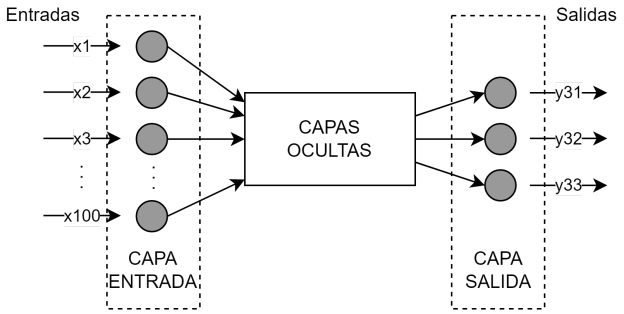

Figura 4. Representación de la red neuronal con los datos de entrada y salida.

El modelo propuesto cuenta con dos capas ocultas, en donde se puede seleccionar la cantidad de neuronas que va a tener, teniendo un límite de hasta 10 neuronas por capa oculta. Estas capas son lineales, esto significa que:

$$y_{pj} = f(net_{pj}) = net_{pj} \tag{8}$$

$$f'(net_{pj}) = 1 \tag{9}$$

Y la capa de salida es del tipo sigmoidal, es decir, la salida no es mutuamente excluyente, cuenta con una probabilidad de ser cualquier de las clases: [5]

$$y_{pj} = f(net_{pj}) = \frac{1}{1+e^{-net_{pj}}} \tag{10}$$

$$f'(net_{pj}) = f(net_{pj}) * (1 - f(net_{pj})) \tag{11}$$

Para que el modelo se entrene, se debe calcular los errores y actualizar los pesos en consecuencia de los mismos. Las dimensiones de la primera matriz de pesos está condicionada por la cantidad de neuronas de la primera capa oculta (m1) y la cantidad de neuronas en la capa de entrada (100); esta matriz de peso tiene un tamaño de (m1,100). En cambio, la matriz de bias depende de la cantidad de neuronas de la primera capa oculta, por lo que la matriz de bias para la primera capa sería de (m1,1). A partir de esto, se calcula la salida con la expresión (1), lo cual da como resultado una matriz de tamaño (m1,1).

En la segunda capa, la matriz de peso tiene el tamaño (m2,m1), donde m2 es la cantidad de neuronas de la segunda capa. La matriz bias para esta capa es de (m2,1) y la salida es una matriz de dimensiones (m2,1).

Por último, hay 3 neuronas en la capa de salida por lo que la matriz de peso tiene un tamaño de (3, m2) y su matriz de bies es de (3,1).

El criterio de parada definido para el modelo (momento en el cual deja de entrenar) es la cantidad de épocas que se desea entrenar el modelo. Por cada época, además de actualizar la matriz de pesos y bias, se calcula el error del patrón para luego calcular el error global, el cual se ocupa para analizar la precisión del modelo. De manera conjunta al entrenamiento se realiza el cálculo del error con el set de validación, el cual posee un conjunto menor de letras distorsionadas las cuales son ingresadas para la predicción según el entrenamiento actual y obtener un error global para su comparación.

4 Simulación y Resultados

A fin de poner a prueba el modelo se consideran 3 set de datos, conteniendo 100, 500 y 1000 ejemplos y se los analiza bajo diferentes parámetros para evaluar la predicción del modelo.

Los dataset están divididos en tres partes, donde el 80% es dedicado al set de entrenamiento, el 15% al set de test y el último 5% al set de validación (pero estos porcentajes van a ir variando a la hora de realizar diferentes simulaciones, para evaluar diferentes situaciones). El dataset contiene variedad de casos y distribuidos apropiadamente para que a la hora de entrenar el modelo no suceda el underfitting.

Se consideran 4 tipos de arquitecturas para las simulaciones:

1) 1 capa oculta con 5 neuronas.
2) 1 capa oculta con 10 neuronas.
3) 2 capas ocultas, con 5 neuronas por cada capa oculta.
4) 2 capas ocultas, con 10 neuronas por cada capa oculta.

Para un momento de 0,5, un coeficiente de aprendizaje de 0,5 y 30 épocas se simularon las situaciones planteadas en la Tabla 1.

Tabla 1. Simulación 1

Arq	Set de Validación (%)	Data Set	Predicción (%)	Data Set	Predicción (%)	Data Set	Predicción (%)
1	10	100	100	500	100	1000	100
	20		100		100		100
	30		100		100		100
2	10	100	100	500	100	1000	100
	20		100		100		100
	30		100		100		100
3	10	100	98,75	500	100	1000	100
	20		71,6		100		100
	30		40		100		100
4	10	100	100	500	100	1000	100
	20		100		100		100
	30		72,5		100		100

Bajo las mismas condiciones pero para un momento de 0,9 se simularon las situaciones planteadas en la Tabla 2.

Se puede observar que en casos donde tenemos más de una capa oculta y un dataset pequeño, si el porcentaje del set de validación es muy alto este termina ocupando mucho de los casos por lo que no tiene suficientes ejemplos para el que set de entrenamiento sea lo más representativo posible, por lo que el modelo se deteriora. En cambio en sets más grandes, aunque el set de validación ocupe un alto porcentaje, se logra tener una mayor distribución de casos diferentes a lo hora de entrenar.

Tabla 2. Simulación 2

Arq	Set de Validación (%)	Data Set	Predicción (%)	Data Set	Predicción (%)	Data Set	Predicción (%)
1	10	100	100	500	100	1000	100
	20		100		100		100
	30		100		100		100
2	10	100	100	500	100	1000	100
	20		100		100		100
	30		100		100		100
3	10	100	67,5	500	100	1000	100
	20		61,6		100		100
	30		40		100		100
4	10	100	100	500	100	1000	100
	20		100		100		100
	30		67,5		100		100

Se presenta en la Figura 5 y en la Figura 6 los errores de entrenamiento y de validación de la arquitectura 3 con un dataset de 100 ejemplos según la Tabla 1, donde los valores rojos muestran los errores por época de los ejemplos de entrenamiento, y los valores verdes los errores por época de los ejemplos de validación.

En la Figura 5 y la Figura 6 se puede ver dos situaciones diferentes, pero ambas muestran una deficiencia en la precisión del modelo. Para la Figura 5 el modelo tiene un periodo de épocas en las cuales deja de aprender. En cambio en la Figura 6 lo que ocurre es que el modelo comienza a “memorizar” los patrones en lugar de aprender a identificarlos y diferenciarlos, es decir, ocurre un overfitting.

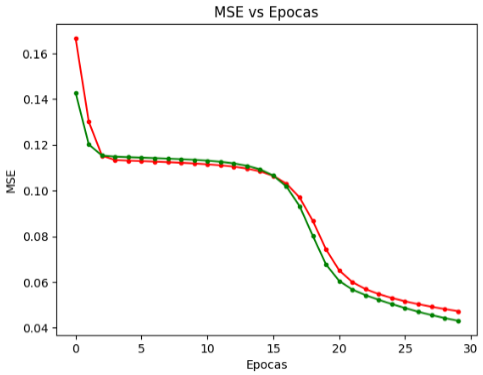

Figura 5. Entrenamiento con un set de validación de 10%.

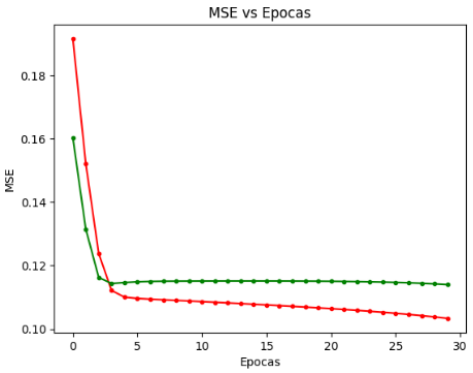

Figura 6. Entrenamiento con un set de validación de 30%.

5 Conclusión

En este informe se describe la implementación de un Perceptrón Multicapa, utilizando diferentes cantidad de neuronas y capas ocultas, variando la cantidad de elementos del dataset y los porcentajes asignados al set de entrenamiento, validación y test.

Luego de su implementación, se evalúa la precisión que tiene cada modelo luego de entrenarlo en base a los distintos parámetros. De esta manera, mediante los datos observados en la Tabla 1 y en la Tabla 2 de la sección de simulación y resultados, se concluye que el porcentaje del dataset destinado al set de validación y el tamaño del dataset influye en la precisión del modelo. Está relacionado con la cantidad de elementos correspondientes al set de entrenamiento.

Teniendo en cuenta la cantidad de ejemplos totales del dataset, esta disminución en el set de entrenamiento afecta decrementando la distribución de casos diferentes a la hora de entrenar. Esto impacta negativamente en la precisión del modelo (que se

puede observar como afecta en la Figura 5 y 6 de la sección de Simulación y Resultado).

Referencias

1. Morales, R. M.; Palma Méndez J.S.: *Inteligencia artificial. Técnicas, métodos y aplicaciones*. McGraw-Hill Interamericana de España S.L. (2008)

2. Berzal, F.: *Redes Neuronales y Deep* Learning. Fernando Berzal (2018)

3. Nielsen, M.: Neural Networks and Deep Learning. *Neural Networks and Deep Learning Web* http://neuralnetworksanddeeplearning.com/ (2019). Accedido el 04 de noviembre de 2022

4. Hilera, J. R. Martinez, V. J. *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones*. Alfaomega. (2000)

5. Draelos, R. "Multi-label vs. Multi-class Classification: Sigmoid vs. Softmax." *Glass Box*, (2019), https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/. Accedido el 04 de noviembre de 2022.