

# Diseño e implementación de una red neuronal para la detección de potenciales evocados visuales

Martínez, J. N ; Pose, F. ; González, N.

Universidad Tecnológica Nacional – Facultad Regional Buenos Aires - Argentina

ngonzalez@frba.utn.edu.ar

**Resumen** — Los productos de apoyo permiten disminuir las barreras que enfrentan las personas con discapacidad y adultos mayores y así favorecer la accesibilidad, comunicación y autonomía. En trabajos previos, se desarrolló un sistema no invasivo basado en una interfaz cerebro-computadora utilizando potenciales evocados visuales. Al ser éste un sistema no invasivo, el nivel de señal respecto del ruido es de varios órdenes de magnitud inferior, razón por lo cual es necesario realizar una adquisición de gran cantidad de muestras de la señal de electroencefalografía, a fin de mejorar la relación señal a ruido, conllevando de esta forma, a reducir la tasa de acciones a ser realizadas por el usuario. En el presente trabajo se propuso realizar la implementación de una red neuronal multicapa y su respectiva evaluación, mediante el algoritmo de retropropagación, para la detección de ondas cerebrales. Se desarrolló un software en C#, el cual permite al usuario configurar los parámetros internos de la red, dándole la posibilidad al usuario de poder seleccionar, de forma empírica, el mejor set de valores para el funcionamiento de la red neuronal. En este trabajo fueron evaluados cuatro sets de parámetros. Como resultado del trabajo, se observó que los mejores resultados se dan con una red neuronal de 2 capas, de 3 neuronas en cada una, y un coeficiente de aprendizaje de 0,5. Bajo estas condiciones, y con 7.000 iteraciones para su entrenamiento, se logró un porcentaje de 71% de aciertos en la validación del modelo. El trabajo realizado permite avanzar con el desarrollo de una herramienta, alternativa, la cual permita a los profesionales de la salud un medio efectivo de trabajo en su quehacer diario.

**Palabras claves:** BCI, rehabilitación, EEG, Redes neuronales.

## I. INTRODUCCIÓN

Una interfaz cerebro computadora -ICC- (BCI por sus siglas en inglés - brain computer interface-) es un sistema que analiza la actividad eléctrica del cerebro con el objetivo de controlar dispositivos externos (fig.1). Los sistemas ICC/BCI se basan en los potenciales corticales lentos, oscilaciones de las ondas alfa y beta, respuesta de la onda P300 o potenciales evocados visuales [1]. El propósito de la interfaz cerebro computadora es establecer un canal de interacción entre el cerebro del usuario y el mundo exterior.



Figura 1: Interfaz cerebro computadora

Los potenciales evocados visuales son la respuesta obtenida al estimular la vía visual. La forma habitual de realizar este procedimiento es presentarle al usuario diversas opciones, las cuales se estimulan a través de contrastes luminicos. Dentro del set de posibilidades, sólo unas pocas serán de su interés, siendo éstas traducidas en estímulos infrecuentes, las cuales deben ser detectadas entre todas las respuestas [2].

El potencial evocado más utilizado es el P300, siendo éste uno de los paradigmas más empleados en la implementación de los sistemas BCI [3]. En general, se obtiene con una latencia de aproximadamente de 300 ms, que representa una media de la rapidez del procesamiento cognitivo. Este potencial se observa principalmente en las zonas central y parietal de la corteza cerebral [4].

En trabajos previos [6,7], se llevó a cabo la implementación de una interfaz BCI para la detección del potencial evocado visual P300. Se empleó un hardware de licencia libre, basado en OpenBCI [5] y un casco realizado con técnicas de impresión 3D. Para realizar la adquisición de las señales y el procesamiento de las mismas, se desarrolló una aplicación en lenguaje C# [6]. Como resultado, se obtuvo un algoritmo capaz de detectar eficientemente el potencial P300.

Al ser un sistema no invasivo, el nivel de la señal respecto del ruido es de varios órdenes de magnitud inferior motivo por el cual es necesario obtener gran cantidad de muestras para mejorar esta relación mediante un promediado coherente implicando un tiempo de concentración prolongado para los usuarios.

El objetivo de este trabajo es implementar una red neuronal que permita reducir los tiempos necesarios para la detección de la onda de interés, sin deteriorar los porcentajes de acierto.

## II. MARCO TEÓRICO

### A. Definición de una Red Neuronal

Una red neuronal puede describirse como un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas o también como un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.

Podemos considerar que la red neuronal es un algoritmo matemático, que recibe la información a través de sus entradas externas, y se espera que la misma sea capaz de aprender a generar cierto tipo de salidas en función del estado de dichas entradas y su historia previa.

### B. Clasificación de las redes neuronales

Podemos clasificar las redes neuronales en redes neuronales biológicas, las cuales no son de interés en este trabajo, y redes neuronales para discriminar señales. Dentro de estas últimas puede realizarse una clasificación más extensa. Las categorías dependen, principalmente, de la naturaleza de la información a procesar (números reales, enteros, conjuntos acotados o infinitos, datos lógicos, etc.), tanto en las entradas como en las salidas. Es necesario tener en cuenta si hay información sobre los pares entradas-salidas o si la red sólo recibirá un conjunto de datos y deberá aprender el patrón de clasificación.

En el presente trabajo, se pretende utilizar un set de señales previamente obtenido en [6], las cuales poseen la onda P300 de interés. Mediante el algoritmo de clasificación se procede a etiquetar cada una de estas señales permitiendo entrenar la red de forma asistida o lo que es lo mismo, para cada valor de entrada, se le indicará cómo debe ser la salida.

En este trabajo se buscó un porcentaje alto de aciertos en la aplicación sin ser considerado el tiempo de entrenamiento de la red.

### C. Perceptrón multicapa

Su funcionamiento se basa en la partícula fundamental, denominada “Neurona” (fig. 2), siendo éste un nodo de procesamiento. Cada neurona recibe señales provenientes de neuronas anteriores, ponderadas por un peso específico para cada una, más una señal de umbral. Se realiza la sumatoria de estos datos y, mediante una función matemática específica, se calcula la salida de la neurona.

A través del entrenamiento de la red, se buscan modificar los pesos de ponderación y los umbrales para cada entrada logrando las salidas deseadas para determinados conjuntos de entradas.

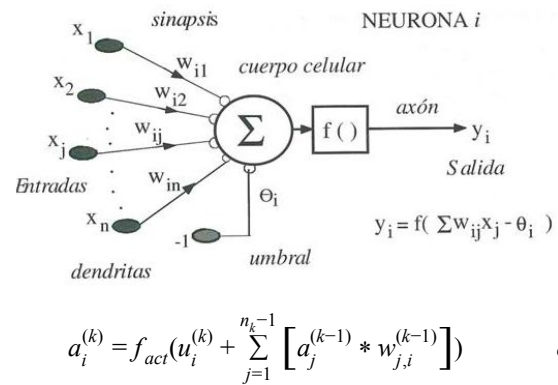


Figura 2: Unidad básica - Neurona

El siguiente nivel de procesamiento, consiste en la agrupación de un conjunto de neuronas para formar las capas de la red.

Una capa es un arreglo de unidades de procesamiento del mismo nivel jerárquico, que toman como entradas a las señales de salida de todas las neuronas de la capa anterior, las procesan y las envían a las de la capa siguiente. El primer nivel se corresponde con la entrada, siendo la única capa que no realiza procesamiento, ya que sólo replica el valor de las entradas externas a la red. La última capa es la que entrega los valores de salida de la red hacia el exterior. Las capas intermedias, se denominan capas ocultas, ya que éstas no interactúan con el exterior.

La capa de entrada deberá tener tantas neuronas como entradas externas deba procesar la red. Mientras que en la de salida, el número de neuronas, se deberá corresponder con la cantidad de datos que la red debe entregar. Para todas las capas ocultas, no existe imposición alguna, tanto de la cantidad de neuronas por capa, como de la cantidad de capas ocultas. Así mismo, no es necesario que éstas tengan la misma cantidad de neuronas entre sí motivo por el cual pueden generarse infinitas combinaciones de valores a fin de obtener una red neuronal.

En principio no es posible conocer cuál será la configuración óptima de la red para la aplicación deseada. Se deberá tomar un punto de partida aleatorio y comparar el rendimiento ante cada cambio en la estructura.

Esta topología presenta dos grandes inconvenientes al momento de definir sus parámetros internos:

- Si se utiliza una red con una cantidad baja tanto de capas ocultas como de neuronas por capa, se puede obtener un entrenamiento deficiente. Lo que provocará que la red no sea capaz de aprender a través de los datos usados y no tendrá un buen desempeño al ser utilizada.
- Si se utiliza una cantidad elevada de capas ocultas o neuronas, podría generarse un sobre entrenamiento de la red, lo que provocará que la red sólo aprenda a reconocer los datos utilizados y no generalice las reglas de detección.

#### D. Backpropagation

Consiste en evaluar las salidas de la red para un determinado set de valores de entrada, y obtener la diferencia entre la salida esperada y la salida real. Con esta diferencia, se comienza desde la última capa, la de salida, corrigiendo los pesos y los umbrales de cada nivel, hasta llegar al primer nivel, el de las entradas. La complejidad de esta técnica se encuentra en que cada nivel depende de todos los niveles anteriores, por lo que la complejidad varía en función de la cantidad de capas ocultas de la red, y la cantidad de neuronas por cada capa elevando la complejidad del algoritmo si la red debe re configurarse con nuevos parámetros en tiempo de ejecución.

### III. ELEMENTOS DE TRABAJO Y METODOLOGÍA

#### A. Procedimiento

A partir del análisis de los datos de partida, se concluye como requisitos que:

- Se deberá utilizar una red para clasificar señales dentro de un conjunto finito de opciones.
- Se podrá etiquetar los pares de entradas-salidas, con lo que será un entrenamiento asistido.
- La salida será binaria, o lógica, ya que deberá informar si está la onda buscada o no dentro de los valores de entrada.
- Deberá tener flexibilidad para poder modificar su forma de funcionamiento, a fin de encontrar el modo óptimo de trabajo.
- Deberá realizarse mediante funciones sencillas y no privativas, permitiéndole ser adaptada a otras plataformas, en futuras aplicaciones.

En base a los criterios mencionados, se optó por diseñar un perceptrón multicapa. En [6], se desarrolló una aplicación gráfica en lenguaje C# la cual permite la configuración del Hardware en tiempo de ejecución, la adquisición y el procesamiento de muestras. Se propuso implementar la nueva solución como una clase en C#, para luego integrar la misma a la interfaz previamente desarrollada.

Las variables de la red son almacenadas de manera matricial (fig. 3). Estas serán los valores de entradas y salidas de la red, las señales internas, los umbrales de cada neurona y los pesos de ponderación de cada conexión. De esta forma, se simplificó el desarrollo de los métodos de la clase, debido a que toda la información se encuentra en un número reducido de matrices.

La estructura de la red puede ser modificada sin realizar cambios en el código de procesamiento debido a que sólo se alteran los límites de los loops para recorrer dichos datos.

```
private double[, ,] pesos;  
private double[, ] seniales;  
private double[, ] umbrales;  
private int[] neuronasPorCapa;  
private int totalDeCapas;  
private int totalDeEntradasExternas;  
private int totalDeSalidasExternas;  
funcionesActivacion tipoDeActivacion =  
funcionesActivacion.FUNCION_SIGMOIDEA;
```

Figura 3: Variables matriciales del kernel de la red

En la ecuación 1, se observa que la salida de cada neurona es una función dependiente de la totalidad de las señales que llegan a ésta desde la capa anterior, más el umbral. Esta función, denominada “Función de activación”, presenta una cantidad de variantes que otorgan ventajas, según la naturaleza de los datos utilizados.

Se tomó como premisa de diseño, tener la posibilidad de cambiar la función empleada en tiempo de ejecución, lo que permite comparar el desempeño de las variantes, para elegir la que mejor se adapte.

#### B. Entrenamiento de la red neuronal

Se basa en la corrección de los parámetros internos de la red para que las salidas obtenidas sean próximas a las esperadas. A fin de lograr el objetivo planteado, se alimenta la red con un set de entradas que provoquen un conjunto de salidas conocidas. Tanto el valor de los umbrales como el de los pesos de ponderación son modificados proporcionalmente a la diferencia entre los valores de las salidas reales y las esperadas. Para esto, es utilizado el método del descenso del gradiente.

$$\gamma_{n+1} = \gamma_n - \alpha * f'(\gamma_n) \quad ec. 2$$

Donde  $\alpha$  es el coeficiente de aprendizaje de la red,  $\gamma_n$  es el valor actual del parámetro a modificar,  $\gamma_{n+1}$  es el nuevo valor y  $f'(\gamma_n)$  es la derivada de la función error. Esta función se define como la suma cuadrática de la diferencia de las N salidas respecto de sus valores esperados, y es lo que se busca minimizar mediante el entrenamiento de la red.

Partiendo de:

$$Error = \sqrt[2]{(S_1 - Y_1)^2 + (S_2 - Y_2)^2 + \dots + (S_N - Y_N)^2} \quad ec. 3$$

Derivando la función de error a fin de hallar el mínimo, se llega a la condición de que es necesario simplemente igualar a cero la derivada del argumento de la raíz.

Afectando los términos por una constante, que no cambiará el resultado, se obtienen las expresiones reducidas.

$$Error = \frac{1}{2} [(S_1 - Y_1)^2 + (S_2 - Y_2)^2 + \dots + (S_N - Y_N)^2] \quad ec. 4$$

En todos los términos, las únicas variables son  $Y_i$ , ya que son las salidas reales de la red, que cambian en cada iteración.

Mientras que las salidas esperadas  $S_i$ , son valores constantes. Al tener una ecuación con  $N$  variables independientes, se deben aplicar las derivadas parciales para obtener la ecuación de la derivada:

$$\frac{dError}{dm} = \frac{dError}{dY_1} * \frac{dY_1}{dm} + \dots + \frac{dError}{dY_N} * \frac{dY_N}{dm} \quad ec. 5$$

En resumen:

$$\frac{dError}{dm} = \sum_{i=1}^N \frac{dError}{dY_i} * \frac{dY_i}{dm} \quad ec. 6$$

Trabajando sobre el primer término de la sumatoria:

$$\frac{dError}{dY_i} = \frac{d[\frac{1}{2} * \{(S_1 - Y_1)^2 + \dots + (S_N - Y_N)^2\}]}{dY_i} \quad ec. 7$$

Donde el único término no nulo será el de cada salida en cuestión se tiene:

$$\frac{dError}{dY_i} = Y_i - S_i \quad ec. 8$$

Reemplazando esto en la ecuación anterior, queda:

$$\frac{dError}{dm} = \sum_{i=1}^N (Y_i - S_i) \frac{dY_i}{dm} \quad ec. 9$$

Como conclusión, se tiene que, obtener la derivada del error respecto del parámetro  $m$ , es equivalente a realizar la sumatoria de la diferencia entre las salidas reales y sus valores esperados, multiplicadas por la derivada de dicha salida real respecto del parámetro  $m$ . Esta fórmula es válida tanto para los pesos como para los umbrales.

A continuación se obtendrá la derivada de cada salida real respecto de los parámetros de la red. Utilizando la expresión de cada salida (ec.1), se reemplaza:

$$\frac{dY_i}{dm} = \frac{d}{dm} \left\{ f_{act}(u_i^{(k)} + \sum_{j=1}^{n_k-1} [Y_j^{(k-1)} * w_{j,i}^{(k-1)}]) \right\} \quad ec. 10$$

Siendo ésta una función compuesta la cual debe ser derivada. Para simplificar el procedimiento se define:

$$g(m) = u_i^{(k)} + \sum_{j=1}^{n_k-1} (Y_j^{(k-1)} * w_{j,i}^{(k-1)}) \quad ec. 11$$

$$Y_i = f_{act}(g(m)) \quad ec. 12$$

Obteniendo como resultado:

$$\frac{dY_i}{dm} = \frac{df_{act}(g(m))}{dg(m)} * \frac{dg(m)}{dm} \quad ec. 13$$

El primer término indica que se debe obtener la derivada de la función de activación de las neuronas. Debido a que se tomó como premisa la posibilidad de tener varias funciones de activación distintas, se contempla también, en el algoritmo, la definición de sus derivadas. A modo de ejemplo, se utiliza la derivada de la función Sigmoidea por

ser ésta la más utilizada. La función sigmoidea es definida como:

$$derivada \text{ de la Sigmoidea} = Y_i * (1 - Y_i) \quad ec. 15$$

Desarrollando el segundo término de la ecuación, se tiene que:

$$\frac{dg(m)}{dm} = \frac{d}{dm} \left\{ u_i^{(k)} + \sum_{j=1}^{n_k-1} (Y_j^{(k-1)} * w_{j,i}^{(k-1)}) \right\} \quad ec. 16$$

Donde  $m$  representa el parámetro respecto del cual se está derivando, y puede ser tanto un umbral como un peso. Operando dicha expresión se obtiene:

$$\frac{dg(m)}{dm} = \frac{d\{u_i^{(k)}\}}{dm} + \frac{d}{dm} \left\{ \sum_{j=1}^{n_k-1} (Y_j^{(k-1)} * w_{j,i}^{(k-1)}) \right\} \quad ec. 16$$

Como última simplificación y teniendo en cuenta que, en el segundo término de la ecuación, el valor de los pesos es constante, se puede reescribir la ecuación como:

$$\frac{dg(m)}{dm} = \frac{d\{u_i^{(k)}\}}{dm} + \sum_{j=1}^{n_k-1} \left( w_{j,i}^{(k-1)} * \frac{d\{Y_j^{(k-1)}\}}{dm} \right) \quad ec. 17$$

Lo que significa que la derivada de  $g(z)$  es igual a la derivada de la función umbral respecto del parámetro en cuestión, más la productoria de los pesos de ponderación por la derivada de cada salida respecto del mismo parámetro.

Finalmente, reescribiendo la ecuación, se obtiene:

$$\frac{dY_i}{dm} = Y_i (1 - Y_i) \left\{ \frac{d\{u_i^{(k)}\}}{dm} + \sum_{j=1}^{n_k-1} \left( w_{j,i}^{(k-1)} * \frac{d\{Y_j^{(k-1)}\}}{dm} \right) \right\} \quad ec. 18$$

Se observa en la derivada dentro de la sumatoria, que es la misma derivada que se está calculando, pero aplicada a las salidas de la capa anterior. Por lo que esta ecuación permite implementar un método recursivo. La función operará de forma recursiva hasta alcanzar el último nivel donde la señal depende del parámetro  $m$ .

El código de procesamiento, no dependerá de la estructura que posea la red, en cuanto a la cantidad de neuronas por capa o de las capas ocultas que posea. Reescribiendo las ecuaciones y reemplazando el parámetro  $m$ :

$$\frac{dY_i}{dw_{a,\beta}} = Y_i (1 - Y_i) \left\{ \frac{d\{u_i^{(k)}\}}{dw_{a,\beta}} + \sum_{j=1}^{n_k-1} \left( w_{j,i}^{(k-1)} * \frac{d\{Y_j^{(k-1)}\}}{dw_{a,\beta}} \right) \right\} \frac{dY_i}{du_i^{(k)}} \quad ec. 19$$

Lo que es lo mismo, operando la expresión:

$$\frac{dY_i}{dw_{a,\beta}} = Y_i (1 - Y_i) \left\{ \frac{d\{u_i^{(k)}\}}{du_i^{(k)}} + \sum_{j=1}^{n_k-1} \left( w_{j,i}^{(k-1)} * \frac{d\{Y_j^{(k-1)}\}}{du_j^{(k-1)}} \right) \right\} \quad ec. 20$$

A continuación se analiza el final del bucle de recursividad considerando la derivada de una salida respecto de un peso de la capa anterior.

$$\frac{da_{\sigma}^{\gamma}}{dw_{\alpha,\beta}^{\gamma-1}} = a_{\sigma}^{\gamma} (1 - a_{\sigma}^{\gamma}) \left\{ \frac{d\{u_{\alpha}^{\gamma}\}}{dw_{\alpha,\beta}^{\gamma-1}} + \sum_{j=1}^{n_{\gamma}-1} \frac{d}{dw_{\alpha,\beta}^{\gamma-1}} \left( w_{j,\beta}^{(\gamma-1)} * a_j^{\gamma-1} \right) \right\} \quad \text{ec. 21}$$

La derivada del umbral es nula, ya que no depende de ningún peso. Y al desarrollar la derivada dentro de la sumatoria, se obtiene que:

$$\frac{d}{dw_{\alpha,\beta}^{\gamma-1}} \left( w_{j,\beta}^{(\gamma-1)} * a_j^{\gamma-1} \right) = a_j^{\gamma-1} \quad \text{si } \beta = \sigma$$

ec. 22

$$\frac{d}{dw_{\alpha,\beta}^{\gamma-1}} \left( w_{j,\beta}^{(\gamma-1)} * a_j^{\gamma-1} \right) = 0 \quad \text{si } \beta \neq \sigma$$

Esto se debe a que cuando el índice de la salida analizada,  $\sigma$ , no coincide con el índice de llegada del peso,  $\beta$ , éste no conecta a ninguna neurona de la capa anterior con la neurona que se está contemplando de la siguiente capa. En el último nivel de recursividad, el algoritmo debe comparar estos índices, y devolver la señal de la neurona de la capa anterior, o cero, según coincidan o no.

Mediante el mismo análisis, se puede estudiar la condición para los umbrales:

$$\frac{d}{du_{\beta}^{\gamma}} \left( u_{\beta}^{\gamma} \right) = 1 \quad \text{si } \beta = \sigma$$

ec. 23

$$\frac{d}{du_{\beta}^{\gamma}} \left( u_{\beta}^{\gamma} \right) = 0 \quad \text{si } \beta \neq \sigma$$

Como conclusión del análisis, se obtuvo un algoritmo de procesamiento que utiliza la recursividad para calcular las derivadas de las salidas respecto de cualquier parámetro. En el último nivel, sólo se compararán dos índices y devolverá un valor de la matriz de señales o cero.

Se da la posibilidad de poder incluir más funciones de activación neuronal agregando el código correspondiente a la función y su derivada.

### C. Algoritmo para el entrenamiento de la red

A continuación se presenta el algoritmo desarrollado para el entrenamiento de la red neuronal.

**MIENTRAS** iteraciones < total de iteraciones

**MIENTRAS** entrada-salidas < total de entradas-salidas

Copiar las entradas [número de par entradas-salidas]

Obtener salidas reales de la red

Calcular diferencia entre salidas reales y esperadas

Obtener las derivadas de los pesos y umbrales

Corregir valor de pesos y umbrales

El algoritmo desarrollado permite definir un porcentaje de aciertos en las salidas, a fin de finalizar el entrenamiento de la red de manera anticipada evitando un sobre entrenamiento de la red neuronal.

### D. Sub algoritmo de obtención de las derivadas de pesos y umbrales

A continuación se presenta el algoritmo desarrollado para el cálculo de las derivadas de los pesos, siendo el mismo utilizado, además, para el cálculo de las derivadas de los umbrales.

**MIENTRAS** haya pesos por derivar

**MIENTRAS** salidas < total de salidas

Primer término = Diferencia salida real y esperada

Segundo término = Derivada de salida respecto al peso

Derivada = Derivada + primer término \* segundo término

**RESULTADO** = Derivada

### E. Sub algoritmo de obtención de la derivada de salida respecto al peso

El algoritmo desarrollado para la obtención de las derivadas respecto al peso, siendo el mismo utilizado para el cálculo de umbrales es el siguiente:

**SI** capa del peso es previa a capa de la salida

**SI** índice partida del peso es igual al número de salida

Primer término = Derivada función de activación

Segundo término = Señal salida de neurona anterior

**RESULTADO** = Primer término \* Segundo término

**CASO CONTRARIO**

**RESULTADO** = 0

**CASO CONTRARIO**

Primer término = Derivada función de activación

**MIENTRAS** salida capa anterior < total salidas capa anterior

Segundo término = Derivada de salida respecto del peso

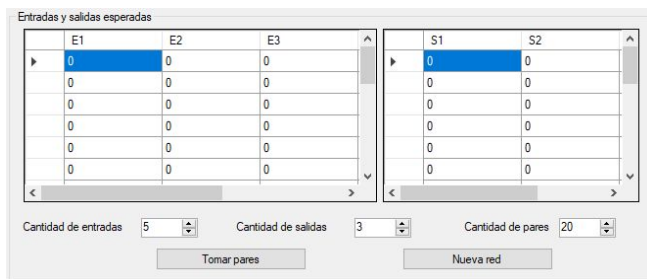
Sumatoria = Sumatoria + Segundo término

**RESULTADO** = Primer término \* sumatoria

## F. Software desarrollado

La interfaz gráfica implementada se encuentra subdivida en secciones para el ingreso de los datos, entrenamiento y verificación de la red.

En la figura 4 se observa el panel de ingreso de datos de entrada y salidas esperada.



El panel 'Entradas y salidas esperadas' contiene dos tablas de configuración. La primera tabla, con encabezados E1, E2, E3, tiene 7 filas de datos, todas con el valor 0. La segunda tabla, con encabezados S1, S2, también tiene 7 filas de datos, todas con el valor 0. Debajo de las tablas hay tres controles numéricos: 'Cantidad de entradas' (5), 'Cantidad de salidas' (3) y 'Cantidad de pares' (20). En la parte inferior hay dos botones: 'Tomar pares' y 'Nueva red'.

Figura 4: Panel de configuración del ingreso de datos

Mediante los controles numéricos se puede configurar la cantidad de entradas y de salidas, así como los pares de datos entradas-salidas que se utilizarán. Los botones sirven para tomar como válidos los datos de las tablas o para crear una nueva red con los parámetros actuales. Los datos deben ser volcados en las tablas manualmente.

En la figura 5 se observa el panel de configuración de la estructura interna de la red. Este permite configurar el Kernel de la red mediante cuatro controles numéricos.



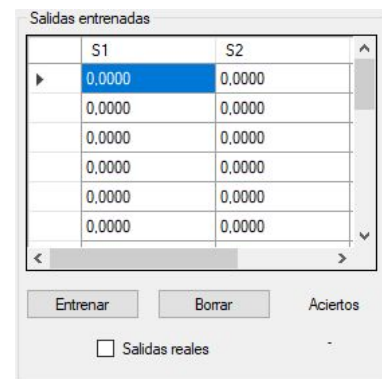
El panel 'Parámetros de la red' contiene cuatro controles numéricos: 'Cantidad de iteraciones' (1500), 'Cantidad de capas ocultas' (2), 'Coeficiente de aprendizaje' (0,4) y 'Cantidad de neuronas' (3).

Figura 5: Configuración del Kernel de la red

La cantidad de neuronas hace referencia a las contenidas en cada capa oculta. En esta instancia, se tomó el mismo valor para todas.

La cantidad de iteraciones (pasadas de todos los pares de datos entradas-salidas) y el coeficiente de aprendizaje ( $\alpha$ ), se deben ajustar de forma empírica, junto a los dos anteriores, hasta obtener un rendimiento aceptable.

En la figura 6 se observa el panel de entrenamiento de la red neuronal. Una vez configurada la red e ingresado los datos, se habilita esta sección para el entrenamiento de la misma.



El panel 'Salidas entrenadas' muestra una tabla con dos columnas, S1 y S2. Ambas columnas contienen 7 filas de datos, todas con el valor 0,0000. Debajo de la tabla hay tres botones: 'Entrenar', 'Borrar' y 'Aciertos'. En la parte inferior hay un checkbox 'Salidas reales' que está desactivado.

Figura 6: Presentación de las salidas obtenidas

Se cuenta con la posibilidad de presentar el valor real de las salidas, o de convertirlas a binario.

## IV. RESULTADOS

A fin de corroborar el funcionamiento de los algoritmos desarrollados, se lo evaluaron los mismos mediante el uso de tablas binarias de 5 entradas. Se observaron los inconvenientes del sub y sobre entrenamiento, al utilizar redes muy básicas o excesivamente complejas, respectivamente.

Se verificó la independencia del algoritmo de aprendizaje respecto a la cantidad de capas ocultas y la cantidad de neuronas en cada una.

Para la verificación del algoritmo fueron utilizadas las siguientes condiciones:

- Cinco entradas binarias las cuales permitieron 32 casos distintos.
- Una salida para detectar cuando, la cantidad de 1 en las entradas, sea impar.
- Una salida para detectar cuando, la cantidad de 1, sea mayor a los 0.
- Una salida, para detectar las 2 condiciones anteriores.

De las 32 combinaciones posibles, se tomaron al azar 20, para ser utilizadas a modo de entrenamiento, y 8 para verificación. Fueron utilizadas cuatro redes con estructuras distintas.

En la tabla 1, puede observarse que la mejor respuesta en la verificación se obtuvo con un coeficiente de entrenamiento moderado y una baja cantidad de neuronas por capa. Se observó que un mejor rendimiento en el entrenamiento puede llevar a una incorrecta generalización de las reglas.

Configuración	A	B	C	D
---------------	---	---	---	---

Alpha	0,1	0,5	0,5	1,0
Iteraciones	10.000	7.000	10.000	4.000
Capas ocultas	2	2	3	3
Neuronas por capa	3	3	2	2
Aciertos entrenamiento	100%	100%	90%	90%
Aciertos verificación	63%	63%	71%	67%

Tabla 1: Resultados obtenidos

## V. CONCLUSIONES

Se consiguió establecer la red neuronal como una sola clase, que opera y almacena los datos y parámetros de funcionamiento internos de manera matricial.

La cantidad de métodos o funciones para su uso son reducidos, simples y de código abierto. Su forma de entrenamiento se basa en algoritmos recursivos, lo que reduce su extensión, y logra fácilmente independizarla de la configuración que se le solicite en tiempo de ejecución.

Mediante un apartado sencillo se pueden elegir las funciones de activación de las neuronas, pudiendo realizarse métricas de comparación y así elegir la mejor para la aplicación deseada.

## VI. TRABAJO FUTURO

A fin de poder emplear la red neuronal diseñada en la detección de la onda P300, se analizarán las características principales de la señal lo que permitirá elaborar un algoritmo de procesamiento que extraiga valores representativos de las muestras obtenidas. Con estos datos, se entrenará la red, indicando qué tipos de muestras contienen la onda y cuáles no, de forma que la red podrá detectar rápidamente las características correspondientes a la onda P300 mientras son recolectadas las muestras del estudio, una vez que haya sido correctamente entrenada. Dicho trabajo permitirá reducir el tiempo necesario para realizar la detección de la onda buscada.

## AGRADECIMIENTOS

Este trabajo se realizó en el marco del proyecto "Diseño y evaluación de un sistema de estimulación y control basado en una interfaz cerebro-computadora". Así mismo, los autores, agradecen a la Universidad Tecnológica Nacional –

Facultad Regional Buenos Aires, institución en la cual se lleva a cabo el desarrollo del proyecto.

## REFERENCIAS

- [1] Álvaro Morán García. "Diseño de interfaces cerebro-máquina controlados mediante registros de EEG". Grupo de neurocomputación biológica. Escuela Politécnica Superior Universidad Autónoma de Madrid, 2015.
- [2] Claudia Nureibis Henríquez Muños. "Estudio de técnicas de análisis y clasificación de señales EEG en el contexto de sistemas BCS (Brain Computer Interface)". Universidad Autónoma de Madrid. Escuela Politécnica Superior – Departamento de Ingeniería Informática, 2014.
- [3] C. Descalls-Moll, J. Burcet.Dardé. "Potenciales evocados y su aplicación en epilepsia". Revista neurología 2002.
- [4] Klem, G.H., Luders, H., Jasper, H. and Elger, C. "The ten-twenty electrode system of international federation. The international federation of clinical neurophysiology". Electroencephalography and clinical neurophysiology. Supplement 52 (1999), 3.
- [5] <http://openbci.com/>
- [6] Pose, F., Geria, G.M., Martínez, N., González, N. "Interfaz Cerebro-Computadora basada en OpenBCI e Impresión 3D". Argencon. Argentina, 2018.
- [7] M. Araujo, N. González, F. Pose. Evaluación y detección de potenciales evocados sobre una interfaz cerebro computadora. Aranducon. Paraguay, 2016.