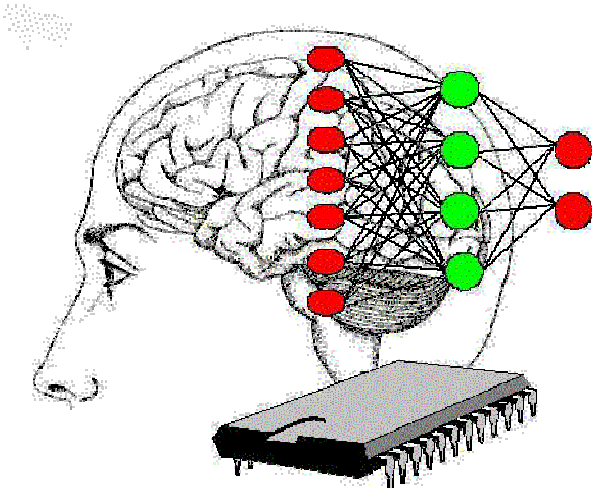
UNIVERSIDAD DE ZARAGOZA

CENTRO POLITÉCNICO SUPERIOR

Proyecto Fin de Carrera

Ingeniería de Telecomunicación

REDES DE CREENCIA PROFUNDA PARA EL RECONOCIMIENTO DE ERPS EN SEÑALES DE EEG



Autor: David Pérez Arbués

Director: Luis Montesano del Campo
Departamento de Informática e Ingeniería de Sistemas

Zaragoza, Septiembre 2010

Apéndice A

Variaciones al modelo original de RBM

Hasta ahora se ha venido hablando de los distintos algoritmos que se pueden utilizar para el entrenamiento de RBM, sin embargo, poco se ha comentado acerca de los parámetros que regulan la actualización de los pesos de la red. Esto es así porque en la mayoría de los algoritmos y artículos, la actualización de los pesos queda en cierta medida condicionada a los gustos del autor o del grupo de investigación en concreto. Es en estos parámetros y en su ajuste preciso, donde recae la mayor culpa del éxito o fracaso de un algoritmo determinado y es por ello que hay que tener muy en cuenta la utilidad de los distintos parámetros y su importancia, así como los valores más comúnmente utilizados en cada uno de los algoritmos.

De igual forma, a lo largo del tiempo, se han ido realizando pequeños cambios en los algoritmos originales que pretenden mejorar el algoritmo y encontrar mejores redes a una velocidad de entrenamiento mayor. Resumimos a continuación los parámetros y técnicas más utilizadas y que mayor influencia parecen tener en el entrenamiento de las RBM ordenados de mayor a menor importancia o uso.

A.1. Rao-Blackwellisation

Para reducir la varianza de la estimación de $< v_i h_j >_{model}$, que en nuestro caso se convierte en $v_i^K h_j^K$, es posible sustituir la muestra h_j^K por su esperanza: $E[h_j^K = 1|\mathbf{v}^K]$. De hecho, esta técnica, es muy utilizada en la mayoría de los estudios y experimentos que se han venido realizando en los últimos años tanto con RBM como con DBN.

De esta forma, las ecuaciones de actualización de pesos definidas en (A.1), (A.2) y (A.3) quedarían de la siguiente forma:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon(v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} \tag{A.1}$$

$$b_j^t = b_j^{t-1} + \epsilon(h_j^0 - E[h_j^K = 1|\mathbf{v}^K])^{t-1} \tag{A.2}$$

$$c_i^t = c_i^{t-1} + \epsilon(v_i^0 - v_i^K)^{t-1} \tag{A.3}$$

A.2. Tasa de aprendizaje

Podemos definir la tasa de aprendizaje ϵ como el tamaño del paso que vamos a dar en el gradiente de la distribución de probabilidad, a mayor tasa de aprendizaje, mayor velocidad de convergencia tendremos hacia el mínimo, sin embargo, la varianza aumentará y la solución será más imprecisa.

Suele ser común utilizar una tasa de aprendizaje grande en las primeras etapas del entrenamiento, de esta forma conseguimos una convergencia rápida, con la posibilidad de escapar de los mínimos de energía locales. Esta tasa, debería ir disminuyendo con el tiempo hasta convertirse en una tasa de aprendizaje muy pequeña al final del entrenamiento, con la idea de lograr acercarnos a la solución más óptima posible, intentando hacer lo menor posible la distancia a este óptimo. En la mayoría de los artículos, se propone una tasa de aprendizaje que disminuya con el tiempo de una forma $1/t^\alpha$, con un $\alpha \in (1/2, 1)$, es lo que se denomina recocido simulado o simulated annealing. En nuestro caso, como posteriormente veremos a la hora de estudiar el entrenamiento de DBN, no se realiza una medida de la convergencia como condición de salida, sino el haber utilizado todos los datos de entrenamiento, por lo que la variación de la tasa de aprendizaje quedaría reducida a $1/N^\alpha$ siendo N el número de iteraciones total en el entrenamiento.

Otra propuesta muy válida es utilizar una aproximación más o menos ruda a esta variación, utilizando 2 o 3 valores de tasa de aprendizaje que van disminuyendo a lo largo de las iteraciones.

Los valores óptimos de la tasa de aprendizaje ϵ suelen variar entre 0.1 y 0.001, dependiendo de las características del algoritmo. Como ya hemos comentado, PCD y SML, requieren unas tasas de aprendizaje relativamente inferiores a las utilizadas en CD-K.

A.3. Momento

El uso de métodos de momento es una forma de hacer dependiente la variación del gradiente de cada iteración con el gradiente de las iteraciones anteriores, evitando variaciones demasiado grandes que podrían desestabilizar el algoritmo. La idea es tan sencilla como calcular la variación de los pesos en el instante actual teniendo en cuenta la variación en el instante anterior, que, a su vez, depende del instante anterior a él, y así sucesivamente.

Traducido a fórmulas, incluimos una nueva tasa β que nos da idea de la importancia que se concede a la variación en el instante anterior y definimos I_{ij}^t como el incremento que se va a realizar en los pesos en el instante t de forma que modificamos la ecuación (A.1) hasta convertirla en:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon(v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} + I_{ij}^{t-1} \tag{A.4}$$

siendo

$$I_{ij}^t = \epsilon(v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} \tag{A.5}$$

Se omite la modificación de las ecuaciones (A.2) y (A.3) por considerarse triviales.

Los valores del momento más comúnmente utilizados se encuentran alrededor de 0.7 y 0.8, aunque, de igual forma que en la tasa de aprendizaje, se pueden utilizar técnicas de annealing, partiendo de valores muy bajos de momento (comúnmente 0), elevándolos poco a poco hasta alcanzar los 0.8 o incluso 0.9 en las últimas etapas del entrenamiento. Esto es así porque al comenzar el entrenamiento conviene tener una mayor varianza que nos permita eliminar rápidamente la respuesta intrínseca de la red, para, al estar cerca de concluir el entrenamiento, reducirlo para alcanzar una solución lo más cercana al óptimo posible.

A.4. Agrupación en grupos de datos

A la hora de calcular las variaciones de los pesos, es muy común que en lugar de evaluar uno a uno todos los datos de entrada, lo que llevaría un gran consumo de tiempo, estos datos se agrupen en los que se ha denominado grupos (o batches). De esta forma, se calcula la variación de los pesos como el promedio de las variaciones individuales. Con esto conseguimos por un lado una mayor velocidad de entrenamiento, al reducir el número iteraciones del algoritmo y simultáneamente, movernos en la media de la variación de los pesos, lo que nos asegura acercarnos a una mayor velocidad al óptimo.

Añadiendo el uso de minibatches al entrenamiento de una RBM, la ecuación (**??**) pasaría a ser:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon\frac{1}{B}\sum_{b=1}^{B}(v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} + I_{ij}^{t-1} \tag{A.6}$$

con

$$I_{ij}^t = \epsilon\frac{1}{B}\sum_{b=1}^{B}(v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} \tag{A.7}$$

siendo B el tamaño de batch. Normalmente se suele utilizar un tamaño de batch de entre 20 y 100 muestras, aunque en el caso de PCD puede ser recomendable aumentar este número de muestras hasta las 200.

A.5. Valores reales en entrada

Mientras que hasta el momento hemos hablado de neuronas binarias, en muchos artículos se utilizan numerosas técnicas que tratan de violar este principio, obteniendo sin embargo muy buenos resultados. La idea general es normalizar los datos de forma que cada pixel de entrada tenga un valor entre 0 y 1. Este nuevo valor normalizado puede considerarse como la probabilidad de que un pixel de una imagen esté encendido o no. De igual forma, a la hora de calcular un nuevo valor reconstruido en la capa visible, deberemos tomar $P(v_i^K = 1|\mathbf{h}^{K-1})$ omitiendo el muestrear sobre esta distribución de probabilidad.

A.6. Decaimiento de los pesos

Para evitar un crecimiento desmesurado de los pesos y tender a una situación en la que los pesos sean los menores posibles, se plantea normalmente el uso de un decaimiento de pesos o weight decay. El procedimiento es tan sencillo como disminuir en un pequeño porcentaje el peso de cada una de las conexiones para obligarlas a tender a cero. Normalmente, el weight decay se utiliza como un factor λ que varía entre los 0.1 y los 0.0001. De igual forma que con el momento y con la tasa de aprendizaje, se puede realizar un recocido simulado que disminuya el valor del decaimiento conforme avanza el entrenamiento.

La fórmula de actualización de pesos (A.6), una vez tenido en cuenta este factor, quedaría de la siguiente forma:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon \frac{1}{B} \sum_{b=1}^{B} (v_i^0 h_j^0 - v_i^K E[h_j^K = 1|\mathbf{v}^K])^{t-1} + I_{ij}^{t-1} - \lambda w_{ij}^{t-1} \quad \text{(A.8)}$$

donde I_{ij}^{t-1} se mantiene inalterado.

Apéndice B

Tratamiento previo de los datos de EEG

B.1. Saturación

La grabación de señales de EEG presenta numerosos inconvenientes. Uno de los principales es discernir la actividad cerebral propia del sujeto o sus capacidades motoras de la que propiamente corresponde a la tarea que estamos intentando clasificar, en este caso, el correcto o incorrecto funcionamiento del robot.

Uno de los efectos más nocivos y difícilmente evitable es el parpadeo de los ojos. Cuando el sujeto a examen parpadea, se producen unos picos de señal en la grabación que superan en varios ordenes de nivel la verdadera señal que estamos intentando clasificar.

Una tarea previa y absolutamente necesaria previa a la normalización de las señales de ERP es detectar estos parpadeos y actuar de una forma consecuente, bien eliminando la señal que contiene el parpadeo, bien atenuando su efecto. Debido a la escasez de muestras de las que disponemos, se decidió atenuar su efecto mediante una saturación de la señal. De esta forma, el parpadeo no toma valores varios ordenes superiores a los datos, lo que con la normalización podría anular el resto de la señal, sino que se satura a un nivel de señal parecido al del resto de los datos.

Para lograr este objetivo y sabiendo que los parpadeos tienen valores de amplitud muy separados de los de la señal, se decide saturar el 0.0005 % de los valores de la señal, como se puede ver en la figura B.1

B.2. Normalización

Una vez que se han eliminado los parpadeos de la señal, podemos pasar a trabajar con una señal un poco más limpia, sin embargo, los requisitos de las DBN nos obligan a utilizar señales que varíen entre 0 y 1, así que debemos de

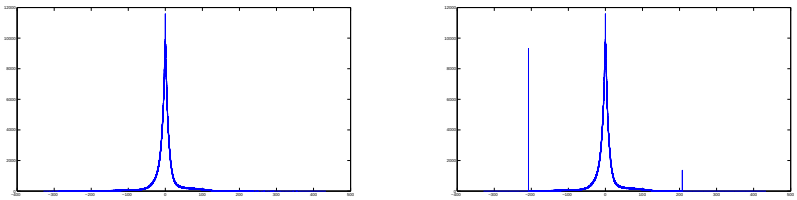


Figura B.1: Histograma de la señal antes de la saturación (izda) y después de la saturación (dcha)

normalizar la señal para que se sitúe entre esos valores. Para conseguir este objetivo, hemos seguido varias estrategias distintas con el afán de encontrar cual de ellas producía mejores resultados al intentar clasificarlas con las redes de creencia profunda.

Normalización global

La primera idea que se nos vendría a la cabeza a la hora de normalizar la señal sería tomar el valor máximo y el mínimo del conjunto de señales (en este caso serían los valores de saturación) y convertir el mínimo en 0 y el máximo en 1 usando la siguiente fórmula:

$$d_i = \frac{d_i - minimo_{global}}{maximo_{global} - minimo_{global}}; \tag{B.1}$$

El problema que presenta este tipo de normalización es que, aunque hayamos efectuado una saturación previa, esta saturación es aproximada y hay muestras del conjunto de datos cuya amplitud queda muy reducida, dificultando el proceso de aprendizaje de nuestra red. En la figura B.2 se puede ver la media de los canales 28 y 29 de cada una de las clases a clasificar una vez aplicada la normalización global. Como se puede ver, la amplitud de la señal es relativamente pequeña, apenas supera el 0.35. En rojo se puede ver la clase 1, en verde la clase 2, en azul la clase 3, que corresponde a la señal correcta, la 4 en cyan y por último la clase 5 en amarillo.

Normalización por señal

Otra posible aproximación al problema es la normalización por señal o por muestra, de forma que el máximo y el mínimo ya no son globales, sino que pertenecen a cada una de las muestras, incluyendo todos los canales. Al utilizar esta normalización, perdemos la relación de amplitud entre una muestra y otra, pero seguimos manteniendo la relación de amplitud entre canales.

Como se puede observar en la figura B.3, se ha conseguido aumentar la amplitud y que los valores estén más próximos al 0 y al 1 de la señal.

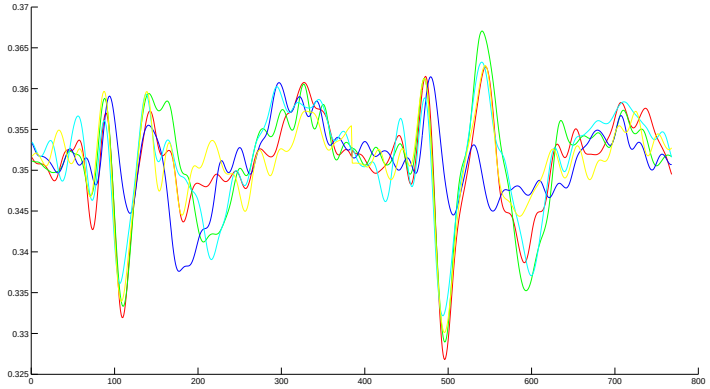


Figura B.2: Media de las distintas clases de ERP para el sujeto 1 y normalización global

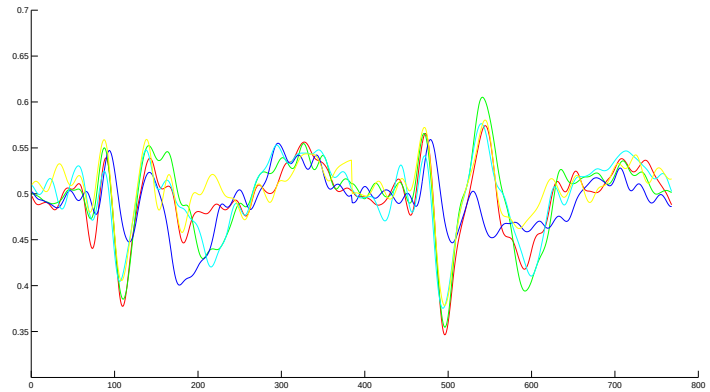


Figura B.3: Media de las distintas clases de ERP para el sujeto 1 y normalización por señal

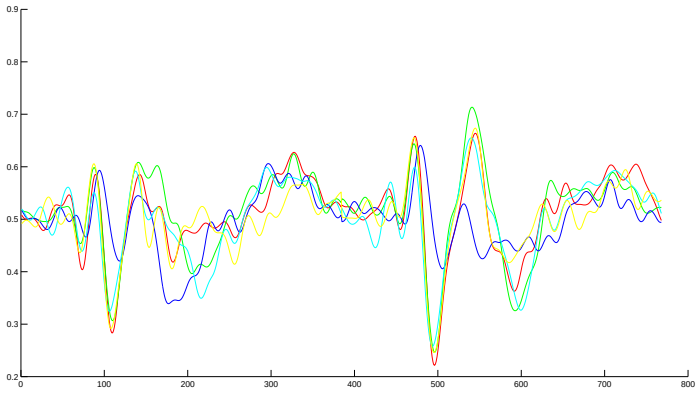


Figura B.4: Media de las distintas clases de ERP para el sujeto 1 y normalización por canal

Normalización por canal

Siguiendo un paso más adelante, otra posibilidad es normalizar cada uno de los canales de cada una de las muestras por separado, esto nos hace perder la relación de amplitud entre canales, pero nos permite una mayor separación de cada una de las clases, como se puede ver en la figura B.4

Normalización por valor

Repasando las normalizaciones que hemos venido haciendo hasta ahora, en todas ellas hemos tratado las muestras como una señal eléctrica, sin embargo, una red neuronal no trata las muestras como una señal, sino como un conjunto de datos, pudiendo desordenar cada uno de los valores a su antojo.

Para que quede más claro pondremos el ejemplo de una señal sinusoidal muestreada, ver figura B.5. Al no tener las DBN conexiones intracapa, no es capaz de relacionar una muestra con la siguiente y con la anterior, así que da exactamente igual que demos como entrada a la red una señal como la de la izquierda o como la de la derecha, siempre que se respete el mismo orden de seleccionar las componentes de los datos para todas las muestras.

Al hilo de esta particularidad, la amplitud de cada una de las variables de una muestra respecto de las vecinas no debería tener apenas importancia, sin embargo, sí que debería tener importancia la amplitud de una variable con respecto a la de las otras clases. Es por ello que decidimos probar a normalizar por variables en lugar de por señales. En este tipo de normalización, el máximo y el mínimo vienen dados por el valor máximo y mínimo en cada variable de cada muestra, siendo este valor el que utilizamos para normalizar.

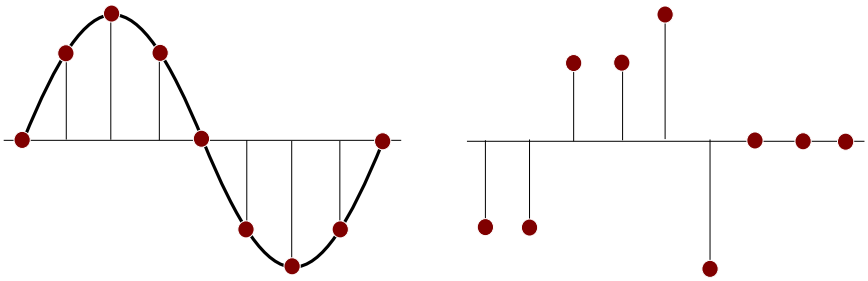


Figura B.5: El orden de los valores de cada muestra no es importante, mientras que se mantenga el mismo para todas las muestras

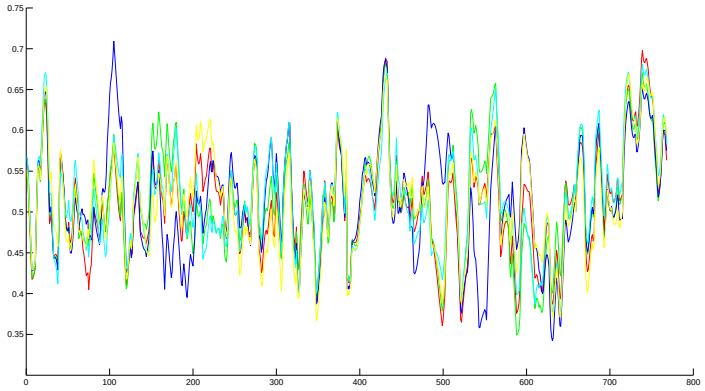


Figura B.6: Media de las distintas clases de ERP para el sujeto 1 y normalización por componente

El resultado de esta normalización, como cabría esperar, rompe la estructura de la señal, pero es igual de válido para las pruebas en DBN (ver en figura B.6).

Normalización combinada

Por último, sólo añadir que estos tipos de normalización pueden combinarse entre sí para dar lugar a otros tipos de normalización, en concreto y a pesar de que los resultados han sido muy similares para todas ellas, se han probado 8 tipos distintos de combinaciones.

B.3. Reducción del tamaño de las muestras

Una vez solucionada la normalización, otro de los problemas importantes que nos plantean las señales de EEG es su gran tamaño, si consideramos

la opción de tomar todos los canales para el análisis, necesitaríamos una primera capa en la DBN de más de 12.000 neuronas, lo que supone una complicación extrema a la hora del cálculo y del tiempo de entrenamiento. Por este motivo se han buscado soluciones que traten de reducir este tamaño al mínimo posible tratando a la vez de mantener todas las características de la señal.

B.3.1. Subsampleado

La forma más rápida de reducir el tamaño de las muestras de entrada es eliminar componentes de las muestras, para ello, nuestra primera idea pasó por reducir la tasa de muestreo. Dado que la señal está muestreada a 256Hz, una posibilidad es reducir la frecuencia de muestreo, tomando menos muestras por segundo. De esta forma se han hecho pruebas tomando 1 de cada 3 muestras e incluso 1 de cada 4, esto equivaldría a reducir la frecuencia de muestreo a 85Hz y a 64Hz respectivamente. Esto conseguiría reducir las muestras a 4.096 en el caso de 85Hz y 3.072 en el de 64Hz.

Como se puede ver, la reducción de componentes de entrada es notable, pero también es notable la pérdida de información que ello conlleva y las tareas de clasificación a estas tasas comienzan a perder exactitud. Es por ello que se descartó esta solución y se decidió hacer una selección más inteligente de los canales a introducir al clasificador.

B.3.2. Selección inteligente de canales

Dada la poca reducción que se puede obtener con el subsampleado y que muchos de los canales apenas presentan actividad, parece una muy buena idea descartar algunos canales que aporten poca información o, todavía mejor, seleccionar aquellos que contengan la mayor información útil posible. Según los estudios del departamento de informática e ingeniería de sistemas, los canales más indicados para intentar la clasificación son los canales 28 y 29, lo que reduce el tamaño de las muestras de entrada a apenas 768 componentes, un valor muy similar al utilizado con la base de datos del MNIST.

A pesar de que estos canales fueran los recomendados, se decidió realizar un estudio comparativo con el resto de canales utilizando la distancia euclídea para seleccionar aquellos canales que cumplieran simultáneamente estas dos características:

1. Distancia euclidea mínima entre las muestras de la clase
2. Distancia euclidea máxima entre las muestras de distinta clase o, al menos, distancia euclidea máxima entre las muestras correctas e incorrectas.

A partir de este estudio se obtuvo que los canales que mejor discriminaban el canal 3 del resto (correcto frente a incorrecto) eran el 19 y el 32 y que

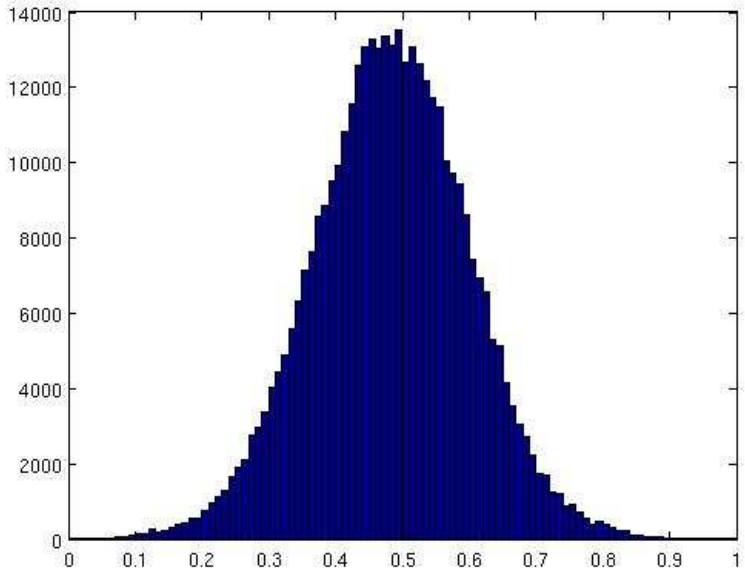


Figura B.7: Distribución de probabilidad de los datos de EEG sin normalizar ni saturar

para discriminar las distintas clases entre sí, lo más sencillo era utilizar los canales 9, 11, 17, 28 y 32.

B.4. Transformaciones de la señal

Además de las transformaciones en amplitud, pueden hacerse multitud de transformaciones distintas sobre la señal para intentar adecuarla lo más posible a lo que puede esperar una DBN, presentamos a continuación algunos ejemplos de los que se han utilizado en este proyecto.

B.5. Escalado exponencial

Considerando los datos de entrada como una distribución de probabilidad, podemos analizar que la distribución de los datos de EEG sin ningún tipo de normalización ni saturación se podría aproximar a una gaussiana centrada en 0.5 (Ver figura B.7). Esta distribución de probabilidad es muy diferente a la que hemos visto tanto en el caso de nuestro conjunto de datos de test como de la base de datos del MNIST.

Una buena forma de superar a la vez los problemas de normalización, de saturación y simultáneamente acercar la distribución de probabilidad a la

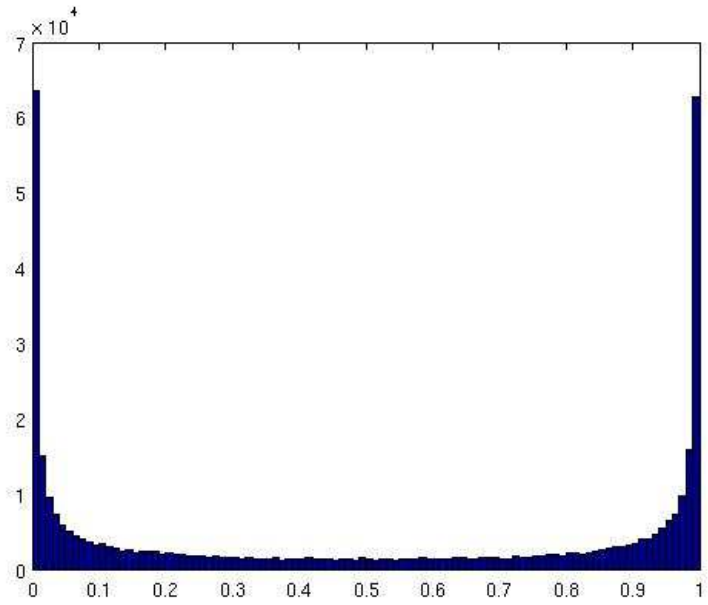


Figura B.8: Distribución de probabilidad de los datos de EEG tras normalización exponencial

utilizada en el resto de casos es utilizar una normalización exponencial. Para ello, utilizaríamos una ecuación similar a la siguiente:

$$d_i = \frac{1}{1 + e^{d_i}}; \tag{B.2}$$

Con ello logramos comprimir los valores muy elevados de la señal como 1 y los valores muy negativos como 0, como se muestra en la figura B.8).

B.6. PCA

El análisis de componentes principales o PCA es una técnica que se suele utilizar para reducir la dimensionalidad de un conjunto de datos. El PCA construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje (llamado el Primer Componente Principal), la segunda varianza más grande es el segundo eje, y así sucesivamente. De esta forma conseguimos una lista ordenada de componentes según influyan más o menos en la varianza de los datos, pudiendo de esta forma descartar aquellas componentes que apenas aporten variación a los datos.

Al realizar un análisis PCA a los datos de EEG descubrimos que las

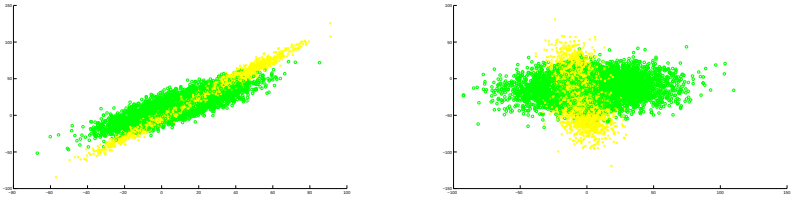


Figura B.9: Efecto de aplicar CSP sobre dos conjunto de datos, antes (izda) y después (dcha)

variables originales están altamente correladas y que es muy fácil retener el 100 % de la información presente en los datos originales únicamente tomando alrededor de 500 componentes del nuevo sistema de coordenadas.

Por supuesto, una vez obtenidas las nuevas componentes y antes de introducirlas en nuestro clasificador, tenemos que volver a hacer un normalizado que sitúe sus valores entre 0 y 1.

B.7. CSP

El algoritmo de patrones espaciales comunes o CSP (Common Spatial Patterns) [6], [35] resulta muy útil cuando se dispone de dos distribuciones de datos en un espacio de alta dimensionalidad ya que es capaz de transformar los datos de manera que encuentra las direcciones donde se maximiza la varianza de una distribución a la vez que se minimiza la varianza de la otra distribución.

Al lograr encontrar simultáneamente zonas de alta varianza en un conjunto de datos y baja varianza en el otro, CSP consigue separar las señales de forma que sea mucho más sencilla su clasificación. Un ejemplo sencillo de la aplicación de CSP a dos conjuntos de datos gaussianos se puede ver en la gráfica B.9.

Como no existe ninguna implementación Matlab del algoritmo, tuvimos que implementarlo nosotros mismos a partir de la teoría del algoritmo. Esta teoría dice que partiendo de la covarianza espacial normalizada de cada uno de los conjuntos de datos podemos calcular la covarianza compuesta como la suma de las individuales de la siguiente forma:

$$\mathbf{C}_c = \overline{\mathbf{C}_l} + \overline{\mathbf{C}_r} \tag{B.3}$$

siendo $\overline{\mathbf{C}_l}$ y $\overline{\mathbf{C}_r}$ la covarianza espacial normalizada de cada uno de los dos conjuntos de datos a separar, quedando definida esta covarianza como:

$$\overline{\mathbf{C}_l} = \frac{1}{N_l}\mathbf{C}_l = \frac{1}{N_l}\frac{\mathbf{E}_l\mathbf{E'}_l}{trace(\mathbf{E}_l\mathbf{E'}_l)'} \tag{B.4}$$

Siendo N_l el número de muestras del conjunto l y $\mathbf{E}_l$ el conjunto de muestras.

Obtenida la covarianza compuesta, podemos factorizarla como $\mathbf{C}_c = \mathbf{U}_c\lambda_c\mathbf{U'}_c$, siendo $\mathbf{U}_c$ la matriz de autovectores y λ_c la matriz de autovalores. Una vez ordenados los autovalores de mayor a menor y reorganizadas por tanto las matrices de autovectores, podemos realizar un blanqueamiento de la matriz de covarianza compuesta usando una matriz $\mathbf{P} = \sqrt{\lambda_c^{-1}}\mathbf{U'}_c$ de forma que todos los autovalores de $\mathbf{PC}_c\mathbf{P'}$ sean igual a uno.

Si a partir de la ecuación B.3 multiplicamos a derecha por $\mathbf{P'}$ y a izquierda por $\mathbf{P}$, obtenemos que:

$$\mathbf{PC}_c\mathbf{P'} = \mathbf{P}\overline{\mathbf{C}_l}\mathbf{P'} + \mathbf{P}\overline{\mathbf{C}_r}\mathbf{P'} = \overline{\mathbf{S}_l} + \overline{\mathbf{S}_r} \tag{B.5}$$

De donde se deduce que $\overline{\mathbf{S}_l}$ y $\overline{\mathbf{S}_r}$ comparten autovectores comunes, por lo que nos sería posible encontrar una matriz B de forma que si $\mathbf{S}_l = \mathbf{B}\lambda_l\mathbf{B'}$, también $\mathbf{S}_r = \mathbf{B}\lambda_r\mathbf{B'}$. De esta forma se puede llegar a la conclusión de que $\lambda_l + \lambda_r = \mathbf{I}$. O lo que es lo mismo, que para autovalores altos de $\mathbf{S}_l$, corresponderían autovalores bajos de $\mathbf{S}_c$.

Es por ello que utilizando una matriz de proyección tal como $\mathbf{W} = \mathbf{B'P}$, podemos transformar cada una de las muestras del espacio original al espacio nuevo de características como $\mathbf{Z} = \mathbf{WE}$. Que sería donde realizaríamos la clasificación. Denominamos a las columnas de $\mathbf{W}^{-1}$ como los patrones espaciales comunes de ambas señales.

Apéndice C

Programas Matlab y su manejo

C.1. Introducción

Además de toda la teoría y de los resultados obtenidos, es conveniente también presentar el modelo Matlab empleado, así como todos los programas y scripts utilizados a la hora de preprocesar las señales de EEG, por si se desean emplear en desarrollos futuros. Podríamos dividir los múltiples programas escritos en dos categorías principalmente, entrenamiento y testeo de una DBN y preprocesado de los datos de EEG. Esta diferenciación viene determinada por la metodología de trabajo empleada a lo largo del proyecto, donde queda claramente separada la preparación de los datos, del entrenamiento de la red con esos datos.

El objetivo del grupo de programas de preprocesado es generar uno o varios archivos de datos de Matlab (.mat) que contengan únicamente dos variables, una variable *datos* con cada una de las muestras a aprender por la DBN y otra variable *labels* que define la clase a la que pertenecen cada uno de las muestras. Una vez generados estos archivos de datos, el grupo de programas de entrenamiento se encarga de leerlos y comenzar el entrenamiento, teniendo en cuenta los parámetros definidos en el archivo "Belief.conf".

Es muy importante mantener la estructura de carpetas de la solución informática, ya que los programas se ocupan de buscar automáticamente los datos de la carpeta datos, leer la configuración de la carpeta BeliefNet y depositar los resultados en la carpeta Resultados. Un cambio en esta estructura, debería llevar parejo un cambio en la programación interna de las funciones descritas a continuación.

C.2. Entrenamiento y testeo de una DBN

Definimos en esta sección todos los programas utilizados para el entrenamiento de una red de creencia profunda, principalmente se trata de un único programa, que a su vez hacen llamadas a multitud de funciones o programas

secundarios.

C.2.1. General_v5

La quinta versión de la función *General* es la que aglutina la lectura de datos, su preparación, la creación y entrenamiento de la red, así como realizar el repesado final, la comparación con los métodos SVM si se necesita y el testeo de la red. Por supuesto también se ocupa de informar al usuario de como se encuentra el proceso de entrenamiento y darle una pequeña estimación del tiempo restante hasta que se concluya el entrenamiento.

Este programa toma como datos de entrada el nombre de un fichero de datos, que debería encontrarse en la carpeta de datos de la estructura de carpetas del proyecto y el nombre de un fichero de configuración, que debería encontrarse en la carpeta *BeliefNet*, junto al resto de funciones y métodos a los que llama la función a lo largo de su funcionamiento.

Aunque esta función no devuelve ninguna salida de forma implícita, sí que crea un nuevo fichero de datos con el nombre del fichero de entrada analizado seguido de un número secuencial que muestra el número de análisis que se han hecho sobre ese mismo conjunto de datos. Por supuesto, este fichero queda guardado en la carpeta *Resultados* de la estructura del programa.

La primera tarea a la que se dedica *General* es a leer la configuración de la red, de los datos y de entrenamiento, creando tres estructuras de configuración bien diferenciadas. Para ello, se hace uso de la función *leeconf_v1* que no es más que una versión modificada de la función *textread* de Matlab, para facilitar su integración con *General*. Una explicación más en detalle de la función *leeconf_v1* puede verse en C.2.2.

Una vez leída la configuración (ver sección dedicada al fichero de configuración,C.2.3), se crea la estructura que contendrá la DBN, para ello se hace uso de la función *creaRed*, que analizaremos en el apartado C.2.4.

El siguiente paso es la lectura del fichero de datos y su preparación para el entrenamiento de la red, esto incluye el proceso de desordenar los datos, agruparlos en batches y definir qué datos formarán parte del conjunto de entrenamiento y cuáles del conjunto de test. En ningún caso se realiza ninguna modificación de las muestras en sí. Para todo ello, hacemos uso de la función *preparaDatos*, que explicaremos posteriormente en la sección C.2.5

Lista la red sin entrenar y listos los datos, comienza el verdadero proceso de entrenamiento de la red, que es el que más tiempo consume. El entrenamiento lo realiza la función *pretrainingBatch_v3* (sección C.2.6), que es una versión evolucionada de las primeras funciones de entrenamiento, que no permitían el entrenamiento mediante el uso de batchs.

Una vez concluida la llamada a la función *pretrainingBatch_v3*, la mayor parte del programa es opcional, y puede comentarse sin influir en los resultados obtenidos. Esto es así porque a partir de ahora, con la red entrenada, se realiza el fine-tunning gracias a la función *finetuning_v3* (sección C.2.8), el

test de reconocimiento de la red mediante el método *Porcentaje_v3* (sección C.2.10) y la comparación con las redes SVM, mediante las funciones propias de Matlab para ello.

C.2.2. leeconf_v1

Esta función supone añadir cierta funcionalidad a la función *textread* de Matlab para que nuestra nueva función sea capaz de leer de manera correcta el número de capas de la red y su tamaño sin tener que complicar en exceso las llamadas a la función. Como mínimo, debe de recibir como datos de entrada el nombre del archivo de configuración y el nombre del proceso que ejecuta la llamada a la función, para etiquetar los posibles errores y warnings. Aparte de ello, se pueden especificar una serie de parámetros auxiliares que corresponden a cada uno de los valores que se quieren leer y su valor por defecto, siempre en parejas. Si no se especifica un valor por defecto y no se encuentra el parámetro especificado en el fichero de configuración, el programa lanzará una excepción que acabará la ejecución del programa y de los programas que lo llamaron. Más información puede encontrarse ejecutando *"help leeconf_v1"* en Matlab. Como datos de salida, se devuelven los parámetros solicitados en el mismo orden en que se pidió su lectura.

El objetivo principal es usar la función estándar *textread* en todos los casos salvo en aquellos en los cuales la palabra clave del fichero de configuración empiece por la palabra clave *"Layer"*. De encontrarnos en este caso, el programa se ocupa de leer consecutivamente todas las claves del fichero de configuración formadas por la palabra *"Layer"* seguida de un número consecutivo comenzando en 1. De esta forma podemos agrupar en una única variable de salida los tamaños de cada una de las capas de la DBN.

De igual forma, este programa se ocupa de realizar el control de las excepciones que la función *textread* puede generar, mostrando al usuario información sobre los posibles errores que se han podido dar y, en su caso, explicando que se han tomado los valores por defecto introducidos en la llamada a la función.

C.2.3. Fichero de configuración

Para el entrenamiento de una DBN, hacemos uso de un fichero de configuración que contiene todos los parámetros y variables que son necesarios a lo largo del entrenamiento de la DBN. Las características que debe de tener este fichero son las siguientes:

- Debe de estar escrito en formato Matlab, de forma que los comentarios comienzan por el símbolo %.
- Debe de constar el tamaño de las capas, en un formato "LayerX=N", siendo X el número de la capa y N el tamaño de dicha capa.

- Los números que indican el número de capa (X), deben de ser correlativos.
- Es obligatorio indicar el número de clases a clasificar.
- Los parámetros y sus valores se escriben como Parámetro=Valor.
- No se puede escribir nada más después del Valor del parámetro, salvo comentarios.
- Todos los valores deben de ser numéricos.
- Los valores booleanos toman el valor 0 o 1.

Entre los parámetros a los que se les puede dar valor en el fichero de configuración podemos encontrar:

- LayerX: Tamaño de las capas, obligatorio.
- T: Número de clases a clasificar, obligatorio.
- Test: Si se le da valor 1, se realiza el análisis de la variación de pesos y de energía a lo largo del entrenamiento. Usar con cautela porque ralentiza mucho el entrenamiento.
- Pasadas: Número de veces que se va a entrenar cada RBM con todos los datos de entrada.
- epsilon: Tasa de aprendizaje.
- k: Número de Sampleados de Gibbs a realizar en el algoritmo CD-K.
- lambda: Tasa de decay.
- PCD: Booleano, indica si se utiliza el algoritmo PCD o no.
- resetPCD: Número de iteraciones tras las que se resetea el algoritmo PCD (si resetPCD=0, no se resetea la cadena de Markov).
- inicb: Varianza del ruido gaussiano con el que se realiza la inicialización de los biases.
- inicW: Varianza del ruido gaussiano con el que se realiza la inicialización de los pesos.
- Rin: Valor inicial del momento.
- Rfin: Valor final del momento.
- fin: Porcentaje del entrenamiento a partir del cual se comienza a utilizar Rfin como momento.

- Batch: Booleano, indica si se utiliza el procesamiento por lotes o no.
- Tambatch: Tamaño de los batchs.
- Porc_test: Porcentaje de los datos de entrada usados para crear el conjunto de datos de test.
- IterFine: Iteraciones del algoritmo de fine-tunning.
- PasadasFine: Número de veces que se va a entrenar la RBM superior durante el fine-tunning.
- epsilonFine: Tasa de aprendizaje en el fine-tunning.

C.2.4. creaRed

Esta función se encarga de devolver una estructura de datos que se asemeje a una DBN, en esta estructura tenemos agrupada toda la información que necesitamos acerca de las DBN, a saber:

- Pesos de las conexiones entre cada una de las capas W (Por simplicidad y velocidad en los cálculos, los biases se integran dentro de esta misma matriz de pesos, añadiendo una nueva fila y una nueva columna. Esto fuerza a añadir una columna extra en los datos, que toma siempre valor 1).
- Tamaño de las capas.
- Número de capas.
- Número de etiquetas.
- Tipo y número de neuronas en cada una de las capas (Gaussiana para entrada, Soft-max para etiquetas o sigmoidea para el resto).

De igual forma, se ocupa de iniciar los pesos de la red según los parámetros leídos en el fichero de configuración. Para ello, toma como entradas las estructuras de configuración de la red y la estructura de configuración general, donde se indica la forma de inicializar de los pesos.

C.2.5. preparaDatos

Tomando como entrada los datos a analizar, sus correspondientes etiquetas y la configuración que se desea aplicar a estos datos, el programa es capaz de devolver dos conjuntos de datos. El primero de ellos, denominado conjunto de datos de entrenamiento es el conjunto que servirá para entrenar la DBN junto con sus correspondientes etiquetas y el segundo denominado

conjunto de datos de validación, también con sus correspondientes etiquetas, servirá para comprobar el buen funcionamiento de la red en reconocimiento.

A partir de los datos iniciales, se selecciona un tanto por cien de los datos como conjunto de test, dejando el resto como conjunto de entrenamiento. Este conjunto de entrenamiento, se desordena y, si la configuración así lo indica, se agrupa en batches, quedando así listos para el entrenamiento de la red.

Por supuesto, a pesar de desordenar los datos, las etiquetas siempre quedan ligadas a los datos a los que acompañan.

C.2.6. pretrainingBatch_v3

Esta función es la que se encarga de realizar el entrenamiento de la DBN a partir del conjunto de datos de entrenamiento, para ello, necesita como entradas el conjunto de datos de entrenamiento, sus correspondientes etiquetas, la red neuronal sin entrenar (o previamente entrenada si se quiere continuar un entrenamiento previo) y la configuración que contiene todos los parámetros necesarios para el entrenamiento. Como salida, la función devuelve la red entrenada, incluyendo, si el usuario se lo indica en la parte correspondiente del fichero de configuración, la variación de los pesos y de la energía de red a lo largo del proceso de entrenamiento.

El método seguido para el entrenamiento es el descrito en la sección 2.3.1, esto es, para cada capa, se entrena una RBM realizando un número determinado de pasadas de cada uno de los batchs en los que se han agrupado los datos, actualizando para cada batch de datos, los pesos de la red. Una vez analizada la primera RBM, se toman como datos de la siguiente RBM las probabilidades condicionales de activación de la capa oculta de la anterior RBM y se repite el proceso.

La actualización de pesos de la RBM se realiza mediante una llamada a la función *updateRBM_v6*

C.2.7. updateRBM_v6

Esta función es la encargada de implementar el algoritmo de actualización de pesos de una RBM dados unos datos de entrada, esto es, implementar bien contrastive divergence, bien PCD o bien SML, según los parámetros de entrenamiento recibidos. Para ello, recibe como entrada una estructura denominada entrada que contiene los pesos de la iteración anterior, los datos de entrada a la RBM, el incremento en el instante anterior para su uso con el momento, una estructura con el tipo y número de neuronas por capa y algunos otros parámetros como las fantasías de PCD o SML. Como segundo parámetro recibe la configuración para la actualización de pesos en la RBM, incluido el algoritmo que se debe de utilizar. Por último, devuelve en una

estructura similar a la de entrada, los pesos actualizados, el nuevo incremento y el estado de los parámetros de PCD o SML, incluyendo las nuevas fantasías.

No vamos a entrar en más detalle acerca del funcionamiento de esta función ya que únicamente se limita a implementar los algoritmos de entrenamiento de RBM descritos en la sección 2.2.1

C.2.8. finetuning_v3

Esta función tiene la única función de, a partir de la red entrenada con el pretraining, realizar un repesado final de los pesos mediante el uso del algoritmo UP-DOWN explicado en la sección 2.3.2.

Toma como entradas la red que ha devuelto el pretraining y la configuración necesaria para el entrenamiento, devolviendo una nueva red con los pesos ajustados.

C.2.9. Tester

Este pequeño script se ocupa de realizar el test de generación de la red, una vez escogida una etiqueta, el programa se ocupa de generar, tras varias iteraciones, muestras similares a las que se introdujeron como entrenamiento.

C.2.10. Porcentaje_v3

Esta función se encarga de comprobar el correcto funcionamiento de la red entrenada y de medir su tasa de error tanto a la hora de clasificar el conjunto de datos de entrenamiento como el conjunto de datos de test, de forma que pueda presentarse un porcentaje de acierto de la red en cada uno de los dos casos y unas matrices de confusión como las mostradas en la sección 5.4. Para conseguir calcular la salida de la red utilizamos el algoritmo descrito en 5.3.

Esta función recibe como entradas los conjuntos de datos de entrenamiento y de test, así como sus correspondientes etiquetas, para devolver el porcentaje de acierto en cada uno de los dos conjuntos de datos y las matrices de confusión.

C.3. Preprocesado de los datos de EEG

Los programas que a continuación se presentan, aunque fácilmente generalizables a otro tipo de señales, son más específicos a los conjuntos de datos de EEG, concretamente en la línea de adaptar los datos a lo que espera recibir como entrada las DBN.

A diferencia de lo que ocurría en el entrenamiento de DBN, no existe ningún programa que aglutine todos los procesos a realizar con la señal, porque el orden o la necesidad de aplicar uno u otros, corresponde al usuario

final, que puede escoger así más fácilmente entre las múltiples opciones que se le presentan, dependiendo de los datos de entrada. Sin embargo y a modo de ejemplo del orden más común en el que se suelen realizar las operaciones, presentaremos en primer lugar un script denominado *Posproc*, que nos ayudará a dar un orden lógico a la presentación de los distintos métodos a utilizar.

C.3.1. Posproc

El script *Posproc*, no podemos llegar a considerarlo como un método en sí, sino como una simple composición de comandos situados en el orden en que deberían ser ejecutados en Matlab. Sin embargo, la necesidad de decidir qué comandos se desean ejecutar o no, de cambiar el tamaño de los datos o la agrupación en clases, requiere del usuario un ajuste previo de este script a lo que realmente desee obtener como salida.

De forma general, podemos decir que usando este script, podemos generar, a partir de las señales proporcionadas desde el departamento de robótica de la Universidad de Zaragoza, un conjunto de datos de entrada que posteriormente podremos utilizar en el entrenamiento de redes DBN.

Antes de ejecutar este script deben de estar cargadas en memoria uno de los dos conjuntos de datos proporcionados por el grupo de robótica. El primero de ellos está compuesto por 5 variables llamadas *Respuesta1*, *Respuesta2* y así sucesivamente hasta *Respuesta5*, que contienen cada una un conjunto de 100 muestras de cada una de las clases a estudiar. El segundo, por dos señales denominadas *signal* y *estimuloMostrado*, que corresponden, respectivamente a la señal de EEG en crudo y de los estímulos mostrados al sujeto de test, grabados como se muestra en el apartado 4.4.

En el caso de encontrarnos en el segundo caso, el primer paso necesario sería realizar el corte de la señal, de forma que separásemos las distintas reacciones del sujeto ante los mismos estímulos. Para ello, se utiliza el método *cortasignal* (sección C.3.2)

Cuando disponemos de los datos bien organizados por clases, el siguiente paso es realizar la saturación, para evitar los efectos indeseados de los parpadeos, esta función la realiza el programa *Saturacion*. (sección C.3.3)

Concluida la saturación de la señal, podríamos comenzar con las transformaciones a la señal, mostramos en este anexo el algoritmo CSP B.7, ya que es el único de todos los procesos de transformación de la señal del que no existía una versión en Matlab, aunque se podría aplicar también PCA o un escalado exponencial.

Posteriormente vendría la normalización de la señal, de ello se encarga el método *Normalizacion*, que nos devuelve los datos normalizados según todos los procedimientos descritos en la sección B.2.

Llegados a este punto, sólo nos quedaría clasificar los datos por clases, crear las etiquetas correspondientes y guardarlos en ficheros de datos que

serán leídos posteriormente para entrenar la red.

C.3.2. cortasignal

Los datos provenientes de la grabación de las señales de EEG son una cadena de datos continua de una duración de varios minutos, en la que se mezclan muestras de errores leves, graves y casos correctos, para que nuestra DBN sea capaz de clasificarlos, antes de nada debemos separar cada uno de los casos por separado, aplicándoles una etiqueta. De igual forma, aprovechamos que hay muchos espacios entre las señales, para obtener muestras de la no presencia de estímulo, con la esperanza de que esto nos ayude en la clasificación.

A partir de la grabación de los datos, guardada en la variable *signal* y la señal que contiene los estímulos mostrados al sujeto, guardada en *estimuloMostrado*, vamos recortando la señal, creando una celda de 6 posiciones, cada una conteniendo una de las clases posibles de error, siendo 1 y 5 los errores graves a izquierda y derecha, 2 y 4 los errores leves a izquierda y derecha y 3 los casos correctos. El caso 6 agrupa las muestras de no presencia de ningún estímulo.

Como salida de este método se presenta una celda D de 6 posiciones, con todas las muestras tomadas de la señal.

C.3.3. Saturación

Este método, toma como entrada la señal producida por cortasignal (o producida manualmente por el usuario a través de las variables *Respuestas*, y devuelve una nueva celda con el mismo nombre que la de entrada habiendo realizado una saturación de la señal como la explicada en el apartado C.3.3.

Esta función permite la configuración del punto de recorte a través de su parámetro *maxporcentaje*

C.3.4. CSP

La función CSP seguramente sea la más compleja de todas las creadas para analizar la señal de EEG. Dado que no existía ninguna versión Matlab del algoritmo CSP que funcionara correctamente, se decidió implementar nuestra propia versión a partir de la teoría mostrada en el apartado B.7.

Esta función toma como entrada dos conjuntos de datos a separar (en forma de matriz $Muestras \times Tiempo \times Canales$ y devuelve los dos conjuntos de datos transformados en el mismo formato y la matriz de transformación. Hay que destacar que los valores de salida pueden contener valores reales e imaginarios, fundamentalmente debido a la precisión limitada que tiene Matlab, así que es recomendable tomar únicamente la parte real de los datos de salida.

Para evitar errores, a lo largo de la función se van realizando distintos tests, para verificar todas y cada una de las condiciones que deberían cumplirse.

C.3.5. Normalización

La última parte del proceso de preparación de datos es la normalización. Esta función, toma como entrada una celda D, con cada una de las clases en las que se han dividido las muestras y realiza todas y cada una de las normalizaciones descritas en B.2. Como salida, obtenemos una serie de celdas cuyo nombre viene dado según la normalización realizada, a modo de resumen diremos que N corresponde a normalización global, I a normalización por señal, L a normalización por valor y D a normalización por canal. La distinta combinación de estas letras da lugar a las distintas combinaciones posibles de Normalización (por ejemplo NIL corresponde a realizar primero una normalización por señal y posteriormente una por valor)

C.4. Otras funciones y scripts

C.4.1. Análisis

Este script toma como entrada una variable datos y devuelve una matriz con las distancia euclídeas entre cada uno de los canales para cada una de las clases. A partir de este análisis pueden escogerse los canales que presentan la mayor distancia euclídea entre sí.

C.4.2. Script

Este script automatiza el entrenamiento simultaneo de varias redes DBN con distintos conjuntos de datos de entrada, permitiendo decidir qué ficheros de datos se van a analizar y con qué fichero de configuración.