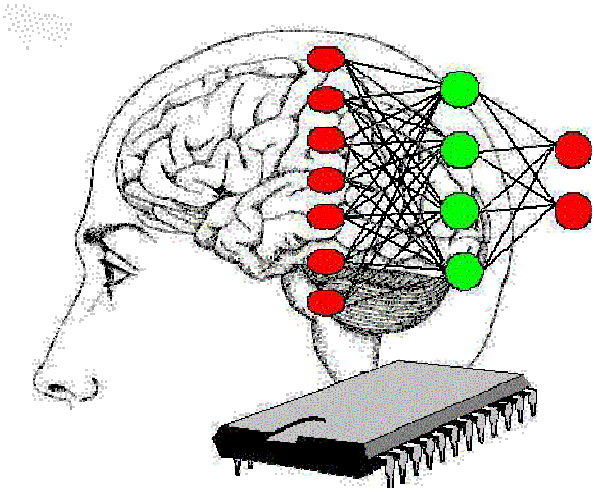
UNIVERSIDAD DE ZARAGOZA

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

Índice general

Capítulo 1

Introducción

1.1. Objetivos

Dentro del Grupo de Robótica de la universidad de Zaragoza, se viene trabajando en los últimos años en la utilización de técnicas de aprendizaje y de interfaces cerebro computador para el control de robots inteligentes. El objetivo último de estos trabajos es el de crear una simbiosis entre el usuario y el robot que permita un aprendizaje continuo y mutuo entre ambos. Uno de los retos esenciales planteados en este escenario es la posibilidad de utilizar la actividad cerebral del usuario para evaluar el comportamiento del robot. En particular, el Grupo de Robótica ha estado estudiando los potenciales de error, un tipo de actividad cerebral que se genera espontáneamente como resultado de una acción ejecutada u observada. Si la acción o su resultado no han sido los esperados, se genera un potencial particular en el cerebro. Para trabajar con estos potenciales, se utilizan lo que comúnmente se denomina electroencefalograma (EEG en adelante), o lecturas de la actividad eléctrica del cerebro a lo largo del tiempo tomadas en determinadas áreas de la superficie de la cabeza.

Uno de los retos más importantes en este contexto, y en el campo de los interfaces cerebro-computador en general, es el desarrollo de algoritmos de detección y clasificación fiables, ya que las medidas de los potenciales de error obtenidas con EEG suelen ser altamente ruidosas y no estacionarias. En la literatura, se pueden encontrar distintos métodos de clasificación con diferentes grados de complejidad y con mayor o menor tasa de acierto. El método que se había venido utilizando en el grupo había sido las máquinas de soporte vectorial o SVM. Esta solución ha venido siendo la más utilizada en la literatura por ser la que menor tasa de error cometía en una amplia variedad de aplicaciones de EEG, incluyendo la clasificación de potenciales de error.

Recientemente, se han desarrollado un nuevo tipo de redes neuronales llamadas redes de creencia profunda o deep belief networks (en adelante

DBN), introducidas por Geoffrey Hinton en 2002 [15]. Diversos trabajos han ido desarrollando algoritmos de entrenamiento, estudios de convergencia y aplicando este tipo de modelos a varios problemas de aprendizaje como la clasificación de imágenes, la generación de movimiento a partir de datos capturados o las series temporales, entre otros. Las DBN han demostrado ser una solución muy efectiva en una amplia variedad de problemas, superando en la mayoría de los casos a la mayor parte de las soluciones propuestas hasta el momento. Además de su gran versatilidad, (son capaces de afrontar problemas de aprendizaje supervisado como no supervisado), las DBN pueden llegar a ser entrenadas de una manera especialmente rápida y producir redes de gran eficacia.

El objetivo de este proyecto es estudiar el comportamiento de estas redes de nueva aparición sobre los datos de EEG correspondientes a potenciales de error, para evaluar su tasa de error y comparar sus prestaciones con el método de clasificación basado en SVM utilizado hasta el momento.

La realización del proyecto ha consistido en tres tareas secuenciales. En primer lugar, la documentación disponible está basada principalmente en artículos publicados en conferencias internacionales. Ésta contiene multitud de versiones de redes y variaciones de los algoritmos de entrenamiento originales, siendo difícil evaluar sus diferencias y compararlas entre sí. En otras palabras, todavía no existe un consenso definitivo en cuanto a qué algoritmos funcionan mejor en cada caso. Obviamente, tampoco existe una librería de código que permita utilizar las DBN como una herramienta estándar y simplemente ajustar los parámetros del método. Por lo tanto, la primera tarea ha consistido en realizar un estudio en profundidad del estado del arte tratando de identificar los distintos métodos para seleccionar aquellos que fuesen mas adecuados a nuestro problema.

Una vez realizado el estudio de las DBN y de sus distintos algoritmos de entrenamiento, el segundo paso es realizar una implementación de los modelos de redes de creencia profunda y los métodos de entrenamiento correspondientes. Para ello se comenzó con un modelo básico al que incrementalmente se le añadieron las modificaciones y mejoras seleccionadas. Este proceso permite tomar contacto con las DBNs y, sobre todo, comparar los resultados de la implementación con los resultados obtenidos en los diferentes artículos científicos publicados hasta la fecha en la literatura. En particular, nos centraremos en el estudio de la base de datos del MNIST, compuesta por cifras escritas a mano que deben de clasificarse correctamente en 10 clases entre el 0 y el 9.

Una vez desarrollado el software de entrenamiento y clasificación y verificado su correcto funcionamiento, la última tarea fue estudiar la aplicación del modelo de red seleccionado a los datos de EEG. Para ello se utilizó un protocolo de potenciales de error (en adelante, ERPS) desarrollado en el Grupo de Robótica que proporciono los datos correspondientes. Como se ha dicho anteriormente, los datos de EEG son altamente ruidosos y normalmente re-

quieren un pre-procesamiento que incluye la aplicación de diferentes filtros antes de ser utilizados para clasificación. Los filtros implementados incluyen técnicas estándar como reducción de dimensionalidad basado en componentes principales y/o independientes y filtrado en frecuencias así como filtros espaciales especialmente diseñados para señales EEG, los Common Spatial Patterns. Los datos procesados fueron finalmente usados para clasificación utilizando tanto las DBNs como SVM.

1.2. Descripción del documento

Se ha intentado con el documento agrupar de la mejor manera posible las distintas áreas que se han venido tratando a lo largo de la realización del proyecto, intentando crear un documento claro que permita al lector familiarizarse progresivamente con todas las tecnologías utilizadas. Sin embargo y debido a restricciones de espacio, se presentan numerosas referencias que pueden ampliar los datos que aquí se presentan.

En el primer capítulo, se intenta presentar un resumen de los objetivos del proyecto, de la metodología utilizada y de los resultados obtenidos, así como dar una pequeña introducción a lo que se puede encontrar en el resto del documento.

El segundo capítulo introduce las redes de creencia profunda, desde sus orígenes como máquinas restringidas de Boltzmann hasta las tendencias más actuales. Puede encontrarse más información a este respecto en los anexos, donde se presentan las modificaciones más comunes y que han sido utilizadas en este proyecto y en las numerosas referencias que se van citando a lo largo del texto.

El tercer capítulo trata, de una manera ligera ya que no es el tema central del proyecto, la teoría que sostiene las máquinas de vectores soporte o SVM y las distintas implementaciones que de él se dan en la literatura.

En el cuarto capítulo presentamos los distintos conjuntos de datos que se han utilizado a lo largo del proyecto, presentando sus características más especiales y prestando un especial interés al análisis de los datos de EEG, su pre-procesado y las distintas manipulaciones realizadas sobre estos datos antes de alimentar las redes de creencia profunda.

En el quinto capítulo, se presentan los resultados obtenidos por las DBN en cada uno de los conjuntos de datos, así como los análisis de convergencia que se han realizado sobre estas redes para demostrar su correcto funcionamiento.

Por último, el sexto y séptimo capítulo quedan reservados, respectivamente, a la presentación de la evolución que ha seguido el proyecto a lo largo del tiempo y a las conclusiones finales.

En los anexos puede encontrarse información detallada acerca de las variaciones más comúnmente citadas por los distintos artículos sobre las DBN,

algunas de las cuales producen mejoras evidentes en los resultados, como hemos podido comprobar a lo largo del proyecto. También puede encontrarse una pequeña introducción y tutorial de los distintos programas escritos a lo largo del proyecto, por si se desea ampliar el estudio y se quieren utilizar como base para futuros desarrollos.

Capítulo 2

Redes neuronales y DBN

2.1. Introducción

Las redes neuronales, aunque han demostrado su alta capacidad de representar funciones de alta no linealidad y gran variabilidad, han presentado tradicionalmente un problema de entrenamiento que crecía conforme aumentaban el número de capas y el tamaño de estas capas. Aunque se ha podido demostrar que en multitud de aplicaciones, al aumentar el número de capas, las soluciones aportadas en términos de parámetros y resultados son mucho más eficientes [5], la dificultad de entrenamiento en redes muy profundas, suele causar que, al partir de pesos iniciales aleatorios, las redes queden atascadas en mínimos locales muy alejados del mínimo real de la función de energía de la red. Esto ha provocado que en problemas de gran complejidad, las soluciones aportadas por redes muy simples (una o dos capas ocultas) hayan dado iguales o mejores resultados que las redes más complejas [30], a pesar de que las redes de mayor profundidad pudieran, en teoría, aproximar mejor la distribución de probabilidad de los datos.

Las redes de creencia profunda o deep belief networks (de ahora en adelante DBN), son un caso concreto de redes neuronales multicapa que, dadas sus características que posteriormente explicaremos, se han aplicado recientemente y con mucho éxito en ámbitos como la clasificación, la reducción de dimensionalidad [12], el filtrado colaborativo [24] o comparación de documentos [26]... Este éxito radica principalmente en la forma de entrenamiento, ya que, a diferencia de lo que se venía haciendo con las redes multicapas, el entrenamiento deja de hacerse por retropropagación (backpropagation), que causaba que las redes se estabilizaran en torno a mínimos de energía locales, para pasar a utilizar el método descrito por Geoffrey Hinton [15], que restringe la conectividad de las redes multicapa y asimila de esta forma una red multicapa con una pila de máquinas restringidas de Boltzman (en adelante Restricted Boltzmann Machine o RBM), para las que existe una forma extremadamente rápida y a la vez precisa de entrenamiento.

A continuación, se va a comenzar a desgranar todos los detalles técnicos que sustentan las redes de creencia profunda y cómo estos modelos han ido poco a poco evolucionando en el tiempo para llegar a lo que hoy en día conocemos como DBN. Comenzaremos nuestro estudio con las máquinas de Boltzmann (Sección 2.2), haciendo breves referencias a las redes de Hopfield de las que derivan. Posteriormente, restringiremos la conectividad de este tipo de redes para definir las máquinas de Boltzmann restringidas. Podemos considerar a las máquinas restringidas de Boltzmann como las precursoras de las DBN, es por ello que nos centraremos en detalle en las diferentes formas de entrenarlas (Sección 2.2.1), ya que estos algoritmos serán los mismos que posteriormente utilizaremos en las redes de creencia profunda.

Comprendidas las máquinas de Boltzmann, pasaremos a complicar nuestro diseño para incluir varias capas ocultas, dando lugar a lo que se conoce como redes de creencia profunda (Sección 2.3), estudiando su forma de entrenamiento (Sección 2.3.1) e introduciendo métodos complementarios para el ajuste fino de los pesos (Sección 2.3.2). Por último, y dado que las redes de Boltzmann y sus derivadas son redes no supervisadas y nosotros necesitamos redes supervisadas para la clasificación de patrones de entrada, plantearemos una forma de convertir una red no supervisada en otra supervisada (Sección 2.3.3).

2.2. Máquinas de Boltzmann

Hasta la aparición de las máquinas de Boltzmann, la mayor parte de las redes neuronales que se habían utilizado presentaban un comportamiento puramente determinista, esto es, ante una determinada entrada, y dadas unas mismas condiciones iniciales y unos parámetros de la red fijos, siempre describían el mismo comportamiento, reduciendo el nivel de energía de la red hasta alcanzar un punto estable considerado como óptimo. Sin embargo, cuando lo que se pretende es que una red sea capaz de generar datos que sean similares a un conjunto dado, lo que es necesario es que esta red sea capaz de aprender la distribución de probabilidad subyacente en los datos y, una vez aprendida y asimilada, sea capaz de generar nuevos datos creíbles a partir de ellos. Dicho de otra forma, dada una entrada x, en las redes tradicionales y alcanzado el equilibrio, la salida siempre es una $y = f(x)$, mientras que si lo que buscamos es modelar una distribución de un conjunto de datos determinado, necesitamos que para un dato x obtengamos una muestra aleatoria de la distribución de probabilidad $P(y|x)$.

Uno de los métodos que se propusieron para solucionar este problema son las máquinas de Boltzmann, en las que el estado de cada una de las neuronas que forman la red se decide de una forma estocástica en función de sus entradas. En realidad, la máquina de Boltzmann no es más que una simple generalización estocástica de una red Hopfield [16].

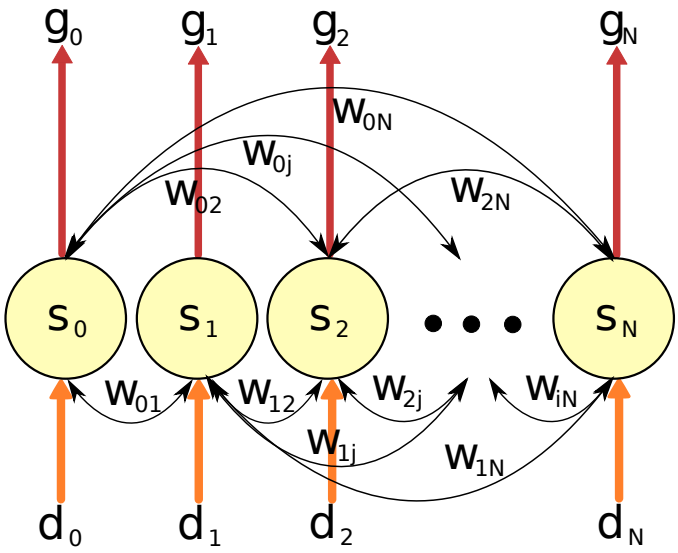


Figura 2.1: Red Boltzmann con N neuronas y una única capa

En una red Hopfield de una única capa, como la mostrada en la figura 2.1, con conexiones simétricas, la regla de actualización es muy sencilla, en un momento dado, una neurona calcula la energía de la red en sus dos posibles estados, encendida y apagada (0 o 1), y toma el estado que tenga una menor energía, disminuyendo así la energía global de la red y acercándose al óptimo. La energía de la red, tanto en el caso de una red Hopfield como en una máquina de Boltzmann, puede calcularse como:

$$E(\mathbf{s}) = \sum_{j,i<j} w_{ij} s_i s_j + \sum_i s_i b_i \tag{2.1}$$

De donde se deriva que la variación de energía viene dada por:

$$\Delta E_i = \sum_{j,i\neq j} s_j w_{ij} + b_i \tag{2.2}$$

siendo d_i los datos de entrada a la red, g_i los datos de salida de la red, s_i y s_j el estado de cualquiera de los nodos de la red, w_{ij} el peso dado a la sinapsis que une ambos nodos y b_i el bias del nodo i. Dicho de otra forma, la variación de la energía al encender la neurona i viene dada por la suma de los pesos provenientes del resto de neuronas j en estado encendido más el bias de la propia neurona i.

Al igual que en la red de Hopfield, pero de forma estocástica, una red de Boltzmann [3], [2] realiza el cálculo de la variación de energía entre los dos posibles estados que puede tomar, pero la neurona tomará el estado activo de acuerdo a la siguiente probabilidad:

$$p(s_i = 1) = \frac{1}{1 + e^{-\Delta E_i / T}} = \frac{1}{1 + e^{-(\sum_j s_j w_{ij} + b_i)/T}} \tag{2.3}$$

siendo T un parámetro que describe la "Temperatura" de la red y que normalmente toma valor 1.

Observando la ecuación 2.3, resulta sencillo entender que para valores muy positivos en la variación de la energía, es decir, cuando al encender la neurona i aumentamos mucho la energía global de la red, $p(s_i = 1)$ tiende a 0 y por tanto es muy probable que la neurona se quede en estado apagado. Si por el contrario, la variación de la energía de red es muy negativa, al encender la neurona i disminuimos la energía global de la red y $p(s_i = 1)$ tiende a la unidad, siendo muy probable que la neurona se encienda. El ruido que se produce para valores de variación de energía próximos a 0, es el que nos permite escapar de mínimos de energía locales.

Una vez escogidos unos pesos y unos biases correctos para la red, proceso que detallaremos en el apartado 2.2.1, y dado un dato de entrada $\mathbf{d}$, el proceso a seguir para obtener una salida $\mathbf{r}$ es el siguiente:

1. Damos un valor inicial al vector de estados $\mathbf{s}$ haciendo $\mathbf{s} = \mathbf{d}$

2. Dejamos libre la red, iniciando una cadena de Markov, tomando en cada iteración de la cadena una muestra de la distribución de probabilidad de los estados $\mathbf{s}$, proceso denominado sampleado de Gibbs o Gibbs sampling.

3. Alcanzado el equilibrio, bien tras un número determinado de iteraciones o porque la variación de la energía de la red entre iteraciones es muy escasa, damos valor al vector de salidas $\mathbf{r}$ como $\mathbf{g} = \mathbf{s}$

Cuando la red se encuentra en equilibrio, es muy sencillo calcular la probabilidad de un determinado estado $\mathbf{s}$ de la red a partir de su energía y de la energía del resto de estados $\mathbf{u}$ como:

$$P(\mathbf{s}) = \frac{e^{-E_\mathbf{s}}}{\sum_\mathbf{u} e^{-E_\mathbf{u}}} \tag{2.4}$$

El problema del aprendizaje es mucho más interesante cuando se añaden capas ocultas que no dependen directamente de los datos, ya que éstas nuevas neuronas extraen las características subyacentes de los datos y permiten modelar distribuciones de probabilidad que no podrían reconstruirse únicamente mediante la conexión directa de las unidades visibles. Aunque se ha planteado el estudio de BM de mayor orden [27], la máquina de Boltzmann más utilizada consta de dos capas binarias, una capa visible, formada por las variables observables $\mathbf{v} \in \{0,1\}^n$ y una capa de variables ocultas o latentes $\mathbf{h} \in \{0,1\}^m$.

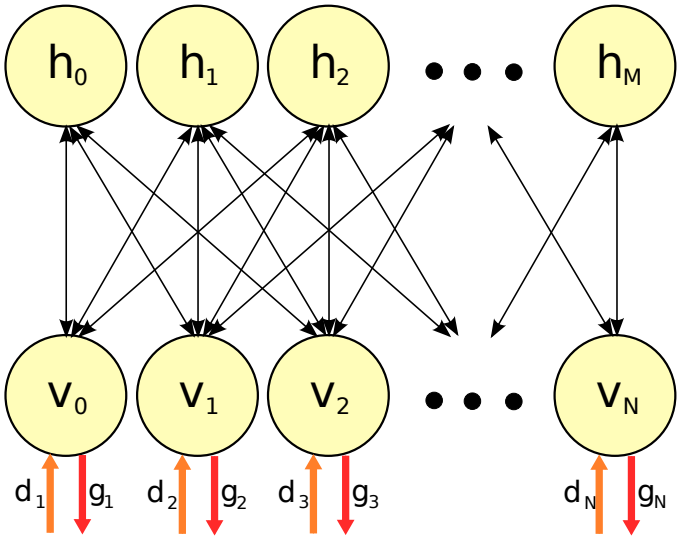


Figura 2.2: Máquina Restringida de Boltzmann con N neuronas en capa visible y M en capa oculta

Generalizando la ecuación (2.1), la energía de red de una BM vendría dada por:

$$E(\mathbf{v}, \mathbf{h}, \theta) = -\mathbf{v}^T W \mathbf{h} - \mathbf{v}^T I \mathbf{v} - \mathbf{h}^T L \mathbf{h} - \mathbf{c}^T \mathbf{v} - \mathbf{b}^T \mathbf{h} \tag{2.5}$$

donde $\theta = (W, I, L, \mathbf{b}, \mathbf{c})$ son los parámetros de la red, siendo:

- W los pesos de las conexiones entre cada uno de los nodos de la capa visible y de la oculta
- I los pesos de las conexiones entre los nodos de la capa visible
- L los pesos de las conexiones entre los nodos de la capa oculta
- $\mathbf{c}$ el término de bias de la capa visible
- $\mathbf{b}$ el término de bias de la capa oculta

Para evitar realimentación, los valores de la diagonal de las matrices I y L son igual a cero.

Durante muchos años, las Boltzmann machine se consideraron intratables computacionalmente por su alto coste de entrenamiento. Por ello surgieron determinadas restricciones a la conectividad y al número de capas ocultas en estas máquinas, lo que permitía acelerar de una manera extrema la velocidad de entrenamiento.

Smolenski y Hinton, [29] y [13] presentaron una nueva modalidad de máquinas de Boltzmann, en las que no se permitían conexiones entre unidades

en la misma capa y en las que el número de capas quedaba restringido a dos únicas capas, una visible, en la que se podían introducir vectores de entrada y obtener valores de salida, y una oculta, que se ocupaba de buscar y memorizar las características subyacentes en los datos introducidos (ver figura 2.2). Al eliminar las conexiones intracapa, podemos eliminar I y L de la ecuación (2.5), por lo que la energía quedaría reducida a:

$$E(\mathbf{v}, \mathbf{h}, \theta) = -\mathbf{v}^T W \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h} \tag{2.6}$$

El funcionamiento de la red mostrada en la figura 2.2 es relativamente similar al de las máquinas de Boltzmann de una única capa, tras cargar un vector de datos en la capa visible $\mathbf{v} = \mathbf{d}$, se calcula las probabilidades en la capa oculta de la red hasta obtener un vector $\mathbf{h}$. A partir de este vector $\mathbf{h}$, se podría calcular un nuevo vector $\mathbf{v}'$. Este proceso de obtener un $\mathbf{v}'$ a partir de un vector $\mathbf{v}$ se denomina sampleado alterno de Gibbs o alternating Gibbs sampling y nos permite crear una cadena de Markov que lleve a la red al equilibrio. De nuevo, llegados al equilibrio, podemos extraer una muestra de la respuesta de la red como $\mathbf{g} = \mathbf{s}$. Esta muestra, si el entrenamiento ha sido el correcto, debería ser muy similar al dato de entrada $\mathbf{d}$ al modelar la red la distribución de probabilidad de los datos.

Existen muchas otras variaciones de las máquinas de Boltzmann que pretenden dar solución a los problemas que presentan su aplicación a determinados problemas, como pueden ser maquinas semirestringidas de Boltzmann (SRBM) [23] al anular únicamente L, de las que derivan las RBM de impacto local o LIRBM [11]. También se podrían citar las RBM convolucionales o CRBM [22] [20] o las máquinas de Boltzmann de tercer o mayor orden [28].

2.2.1. Métodos de entrenamiento

Hasta ahora hemos explicado el funcionamiento de las redes de Boltzmann, sin embargo nada se ha dicho de cómo establecer los parámetros de las redes, los pesos w_{ij} y los biases b_i. El objetivo es lograr que dado un conjunto de datos, los parámetros de la red sean capaz de reconstruirlos de una manera lo más acertada posible, esto es, generar, con los parámetros de la red, una distribución en la que los datos tengan alta probabilidad. Para ello, tenemos que realizar un entrenamiento previo de la red, que trate de encontrar los parámetros w_{ij} y b_i que maximicen la probabilidad de los datos.

En una red Hopfield, la regla para actualizar los pesos es $\Delta w_{ij} = \frac{1}{N}(s_i sj)$, (aprendizaje Hebbs o hebbian learning), en el caso de una máquina Boltzmann sin unidades ocultas (ver figura 2.1), al tratar de maximizar la probabilidad, derivando la ecuación (2.4) y teniendo en cuenta que $\frac{\partial E(\mathbf{s})}{\partial w_{ij}} = -s_i s_j$ se puede demostrar que (ver [1]):

$$\frac{\partial \log P(\mathbf{s})}{\partial w_{ij}} =< s_j(s_i - s'_i) > \tag{2.7}$$

Donde <> es una media de los datos utilizados (puede ser un único vector de datos o, como se explica en la sección A.4, un conjunto de vectores de datos) y s'_i es la probabilidad de que la variable i se encuentre encendida dado el estado de s_j. Dicho de otra forma, dado un vector de entrada $\mathbf{s}$, podemos calcular, una vez eliminado este estímulo, la probabilidad de que la unidad i esté encendida como s'_i. Con esto lo tratamos de minimizar la diferencia entre los datos $\mathbf{s}$ y la versión reconstruida de esos mismos datos por la red $\mathbf{s}'$

Generalizando este método y suponiendo que $w_{ij} = w_{ji}$ podemos llegar a la regla de actualización de una RBM (ver [15]):

$$\frac{\partial \log P(\mathbf{s})}{\partial w_{ij}} =< v_i h_j >_{data} - < v_i h_j >_{model} \tag{2.8}$$

donde $< v_i h_j >_{data}$ es el valor esperado en la distribución de entrada y $< v_i h_j >_{model}$ el valor esperado cuando la máquina de Boltzmann está tomando muestras de vectores de estado en equilibrio. Para llegar a este estado de equilibrio, iniciamos una cadena de Markov haciendo uso del sampleado alterno de Gibbs hasta alcanzar el equilibrio.

Como ya se ha comentado, el objetivo es encontrar aquellos pesos que maximizan la probabilidad de cada uno de los estados, o, lo que es lo mismo, los pesos que minimizan la energía de la red para cada uno de los estados de la red, para ello, lo que hacemos es aumentar los pesos ω_{ij} una pequeña fracción ϵ denominada tasa de aprendizaje, del resultado de la ecuación anterior. Lo mismo ocurre con los términos de bias, salvo que no se tienen en cuenta los h_j:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon(< v_i h_j >_{data} - < v_i h_j >_{model})^{t-1} \tag{2.9}$$

$$b_i^t = b_i^{t-1} + \epsilon(< v_i >_{data} - < v_i >_{model})^{t-1} \tag{2.10}$$

Donde t es un indice que da idea del número de datos leídos del conjunto de entrenamiento, que corresponde a otras tantas actualizaciones de los parámetros de la red.

Para que los pesos y los biases se ajusten de manera correcta a la distribución que deseamos modelar, es necesario que todos los datos presentes en el conjunto de entrenamiento sean introducidos al menos una vez en la red y que los pesos sean actualizados con respecto a ellos. La opción más común es introducir de uno en uno cada uno de los datos de entrada, actualizando para cada uno de ellos los pesos de la red. Es normalmente de gran utilidad

que cada una de las muestras del conjunto de entrenamiento sean introducidas varias veces en la red, ya que las primeras muestras que entran a la red encuentran una red muy poco ajustada y apenas son recordadas por ella.

Como puede verse, estimar la formula (2.8) y por tanto las reglas de actualización (2.9) y (2.10) es intratable en el caso de utilizar una o más capas ocultas, ya que esto implicaría conocer la distribución de probabilidad en las capas ocultas y poder extraer muestras de ella, y, aun en este último caso, implicaría esperar a que la red alcanzara el equilibrio y tomar muestras de este equilibrio, muestras que, debido a tratarse de variables estocásticas, pueden llegar a ser muy ruidosas. Por ello, se han propuesto determinados algoritmos que buscan soluciones al entrenamiento de las máquinas de Boltzmann. Por supuesto, los algoritmos presentados de ahora en adelante únicamente explican el proceso a seguir para un caso concreto de dato de entrada, siendo necesario para un correcto entrenamiento repetir el mismo algoritmo para cada uno de los datos de entrada.

La aproximación más utilizada para aprender en una BM [1] consta de dos fases, en la primera fase (fase positiva) damos un valor a las entradas y una vez alcanzado el equilibrio, aumentamos el peso entre las conexiones entre dos neuronas que se encuentren encendidas. En la segunda fase (fase negativa), no se da ningún valor en las entradas y se deja a la red libre, una vez alcanzado el equilibrio, se decrementa el peso de las conexiones entre dos neuronas que se encuentren activadas.

El principal problema de este tipo de entrenamiento es la lentitud, debemos repetir muchas veces la fase positiva y la fase negativa para conseguir lograr acercar la red a un estado óptimo. Es por este motivo que, a pesar de sus buenas características como aproximador, las máquinas de Boltzmann han sido reemplazadas por modelos más sencillos y limitados.

Las RBM, al tener limitada su conectividad, consiguen que las unidades ocultas sean condicionalmente independientes dado un valor de entrada en la capa visible, por lo que es posible obtener una muestra no sesgada de $< vih_j >_{data}$ en un único paso. Sin embargo, sigue resultando complicado obtener una muestra de $< v_i h_j >_{model}$, ya que implicaría esperar hasta que la red se estabilizara.

Aunque simplificada enormemente, el entrenamiento de estas redes siguió siendo relativamente lento hasta la introducción en 2002 de lo que Geoffrey Hinton denominó Contrastive Divergence learning algorithm, un método que, a pesar de no seguir estrictamente el gradiente de la variación del logaritmo de la probabilidad, sí que, al menos, es capaz, en media, de indicar el sentido de variación adecuado, aunque el valor no sea el correcto. Este algoritmo, que detallaremos posteriormente con detalle, comienza una cadena de Markov con uno de los datos que utilizamos para calcular ($< v_i h_j >_{data}$), como punto de partida para, tras realizar varios procesos de alternating Gibbs sampling completos, tomar la configuración resultante como una estimación del estado del modelo, $< v_i h_j >_{model}$.

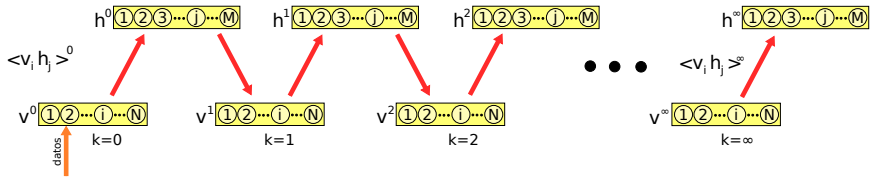


Figura 2.3: Entrenamiento de una RBM tradicional

Contrastive divergence

Como ya habíamos dicho, el cálculo del segundo término de la ecuación 2.8 es intratable computacionalmente. Así que se propuso una nueva técnica denominada Contrastive Divergence [15] para mejorar la velocidad de cálculo. No vamos a entrar demasiado a fondo en las justificaciones matemáticas y la convergencia del método, porque este es un campo que todavía se mantiene abierto y sobre el que se plantean estudios tanto de convergencia como de análisis, podemos citar entre ellos [9], [34] o [4]. Por este motivo, vamos a limitarnos a explicar de una manera lo más sencilla posible el algoritmo a seguir para entrenar la red, ya que es la pieza fundamental para el entrenamiento de las DBN.

Como ya habíamos explicado anteriormente, el objetivo es encontrar aquellos pesos y biases que maximicen la probabilidad de que la red reproduzca los datos de entrenamiento, por lo tanto, nuestro objetivo no es otro que reducir el logaritmo de la probabilidad (ver ecuación 2.8).

La forma tradicional de estimar la segunda parte de la ecuación (2.8) consistía en lanzar una cadena de Markov a partir de cada uno de los datos en la que se van tomando muestras utilizando un sampleado alterno de Gibbs hasta llegar al equilibrio (Ver Figura 2.3). Esta muestra en equilibrio se consideraba una aproximación a la respuesta del modelo de la red $< v_i h_j >_{model} = < v_i h_j >^{\infty}$.

La idea es que podemos tener una buena aproximación a $< v_i h_j >_{model}$ truncando la cadena de Markov tras K iteraciones (con K normalmente menor a 15) y utilizando en su lugar $< v_i h_j >^{K}$. El algoritmo es el siguiente: (Ver Figura 2.4)

1. Tomamos un dato de entrada y forzamos las unidades visibles a los valores dados por el dato de entrada, obteniendo un vector $\mathbf{v}^0$

2. Calculamos la probabilidad de activación de cada una de las unidades ocultas dado el dato de entrada $\mathbf{v}^0$ ($P(h_j^0 = 1|\mathbf{v}^0)$) con la ecuación (2.3)

3. Una vez calculada la distribución de probabilidad en capa oculta, tomamos una muestra de esta distribución, dando un valor a cada una de las neuronas de la capa oculta, la denominamos $\mathbf{h}^0$

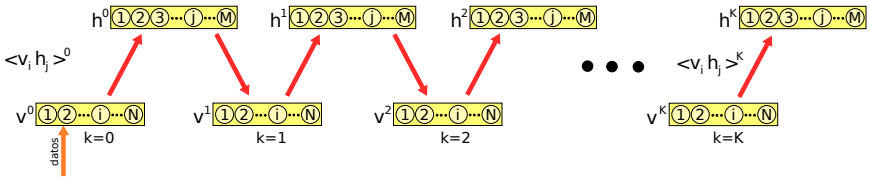


Figura 2.4: Algoritmo CD-K aplicado a una RBM

4. Con $\mathbf{v}^0$ y $\mathbf{h}^0$ podemos calcular $< v_i h_j >_{data}$ (ver ecuación 2.8), tomando $v_i = v_i^0$ y $h_j = h_i^0$

5. A partir de ahora dejamos a la red que se estabilice, para ello, liberamos la capa visible de datos, y tratamos de que la red llegue a un equilibrio. A partir de ahora, repetiremos durante K veces el siguiente proceso (empezamos con $k = 1$):

   - Tomando $\mathbf{h}^{k-1}$ como dato de partida, calculamos la distribución de probabilidad en capa visible $P(v_i^k = 1|\mathbf{h}^{k-1})$ con la ecuación 2.3
   - Generamos una "fantasía" en la capa visible, que denominaremos $\mathbf{v}^k$ tras tomar una muestra de la distribución calculada.
   - A partir de esta muestra, repetimos el proceso para calcular un $\mathbf{h}^k$ al muestrear la distribución de probabilidad $P(h_j^k = 1|\mathbf{v}^k)$, calculada a partir de $\mathbf{v}^k$

6. Terminado el proceso, estimamos $< v_i h_j >_{model}$, segundo término de la ecuación (2.8), como $v_i^K h_i^K$

7. Actualizamos los pesos entre las capas (w_{ij}) y los biases de cada una de las capas (b_i para capa oculta y c_i para capa visible) de acuerdo a las ecuaciones (2.9) y (2.10), que quedan convertidas en:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon(v_i^0 h_j^0 - v_i^K h_j^K)^{t-1} \tag{2.11}$$

$$b_j^t = b_j^{t-1} + \epsilon(h_j^0 - h_j^K)^{t-1} \tag{2.12}$$

$$c_i^t = c_i^{t-1} + \epsilon(v_i^0 - v_i^K)^{t-1} \tag{2.13}$$

8. Repetimos el proceso tomando otro dato de entrada distinto y aumentando el valor de t.

Sobre este modelo, se pueden aplicar numerosas variaciones que traten de modificar y mejorar el comportamiento del entrenamiento, las utilizadas en este proyecto quedan descritas en el anexo A.

En la mayoría de los casos, la estimación del segundo término de la ecuación (2.8) es suficientemente buena tomando únicamente un $K = 1$ (denominado CD-1) (ver figura 2.5), lo que únicamente nos indicaría la dirección

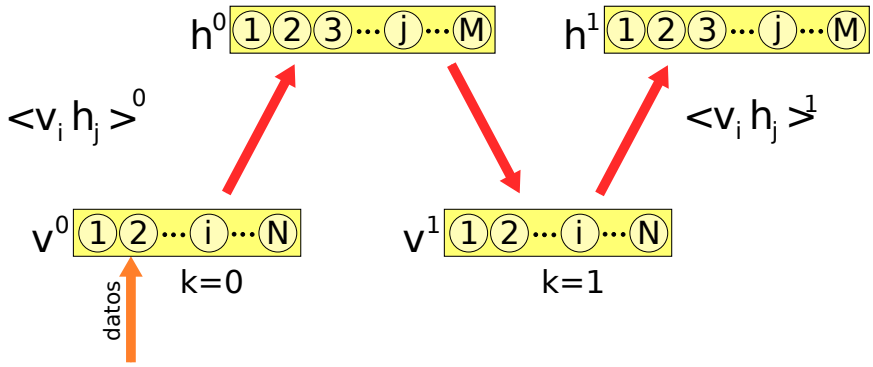


Figura 2.5: Algoritmo CD-1 aplicado a una RBM

en la que deberíamos movernos para disminuir la energía. Valores mayores de K son también comunes, lo que vendría a dar la nomenclatura general CD-K para referirse al algoritmo contrastive divergence.

La idea detrás de todo este algoritmo es exactamente la misma que utilizábamos en las redes de Hopfield, esto es, aumentar el peso de una conexión cuando una característica j se activa a la vez que un pixel i, rebajándolo cuando esta característica se activa sobre una fantasía i.

PCD

El algoritmo Contrastive Divergence, en concreto CD-1, es muy rápido, presenta una varianza baja y es una relativa buena aproximación al gradiente, sin embargo, en algunos casos puede no ser suficiente. Normalmente y si el tiempo de entrenamiento lo permitiera, sería recomendable utilizar CD-K con K lo suficientemente elevado como para obtener una aproximación lo suficientemente buena de la respuesta de la red neuronal en ausencia de estímulos externos. Sin embargo, el tiempo de entrenamiento de una RBM y por extensión, como posteriormente veremos, de una DBN, es un parámetro crucial, por lo que, siguiendo las directrices marcadas en [21] para redes de creencia (Belief nets o BN de ahora en adelante), se propuso un nuevo algoritmo basado en CD-K denominado Persistent Contrastive Divergence o PCD [31].

La idea básica sobre la que se asienta este nuevo algoritmo es que, si las variaciones de los pesos son muy escasas entre las distintas iteraciones del algoritmo PCD, el modelo de la red neuronal entre una iteración y la siguiente es prácticamente estable, por lo que, partiendo del estado estable anterior, en unos pocos muestreos alternos de Gibbs, podemos volver a alcanzar la distribución propia de la red. Si la red apenas ha cambiado, $< v_i h_j >_{model}$ en una iteración, será muy parecido al valor en la siguiente iteración. Por lo que, para variaciones de los pesos tendiendo a cero y usando un algoritmo

CD-K con K tendiendo a infinito, esta suposición es correcta.

Como posteriormente veremos cuando hablemos de los parámetros de entrenamiento, este algoritmo necesita una tasa de aprendizaje muy pequeña, para que las variaciones entre iteraciones sean escasas a la vez que presenta muy buenos resultados con tamaños de batch grandes (ver Anexo A.4), ya que permite realizar una mejor aproximación al gradiente.

Como ya vimos, en el algoritmo CD-K estamos truncando una cadena de Markov tras K iteraciones, esto nos hace perder información de la respuesta original de la red. En PCD se pretende no realizar ese truncado, sino continuarlo en la siguiente iteración del algoritmo. A diferencia de lo que haríamos en CD-K, no usamos $\mathbf{h}^0$ para calcular una "fantasía" en la capa visible $\mathbf{v}^K$ sino que, si nos encontramos en la primera iteración, hacemos un paso de Gibbs sampling para obtener una "fantasia" a partir de unos datos aleatorios gaussianos, o si no, usamos directamente la "fantasía" $\mathbf{v}^K$ de la iteración anterior. (Ver Figura 2.6)

De esta forma, el nuevo algoritmo a utilizar para cada vector de datos quedaría de la siguiente forma:

1. Tomamos un dato de entrada $\mathbf{v}^0$ y calculamos la probabilidad de activación de cada una de las unidades ocultas dado el dato de entrada $P(h_j^0 = 1|\mathbf{v}^0)$ y tomamos una muestra $\mathbf{h}^0$ de esta distribución.

2. A partir de $\mathbf{v}^0$ y $\mathbf{h}^0$ obtenemos $< v_i h_j >_{datos}$, tomando $v_i = v_i^0$ y $h_j = h_j^0$

3. Si nos encontramos en la primera iteración, hacemos un paso de alternating Gibbs sampling para obtener una "fantasía" a partir de unos datos aleatorios gaussianos. Si no es la primera iteración usamos directamente la "fantasía" $\mathbf{v}^K$ de la iteración anterior.

4. A partir de esta "fantasía", iniciamos una cadena de Markov de K pasos, para calcular un nuevo $\mathbf{v}^K$, que guardamos como "fantasía" de inicio para la próxima iteración, y un $\mathbf{h}^K$ muestreando $P(h_j^K = 1|\mathbf{v}^K)$

5. Terminado el proceso, estimamos el segundo término de la ecuación (2.8) como $v_i^K h_j^K$ (Donde $\mathbf{v}^K$ es el resultante de la última iteración de la cadena de Markov)

6. Por último, al igual que en CD-K actualizamos los pesos entre las capas (w_{ij}) y los biases de acuerdo a las ecuaciones (A.1), (A.2) y (A.3)

7. Repetimos el proceso con un nuevo dato de entrada y aumentamos el valor de t

Como puede verse, estamos haciendo de nuevo una aproximación, ya que el modelo de red está cambiando poco a poco mientras se realiza el proceso

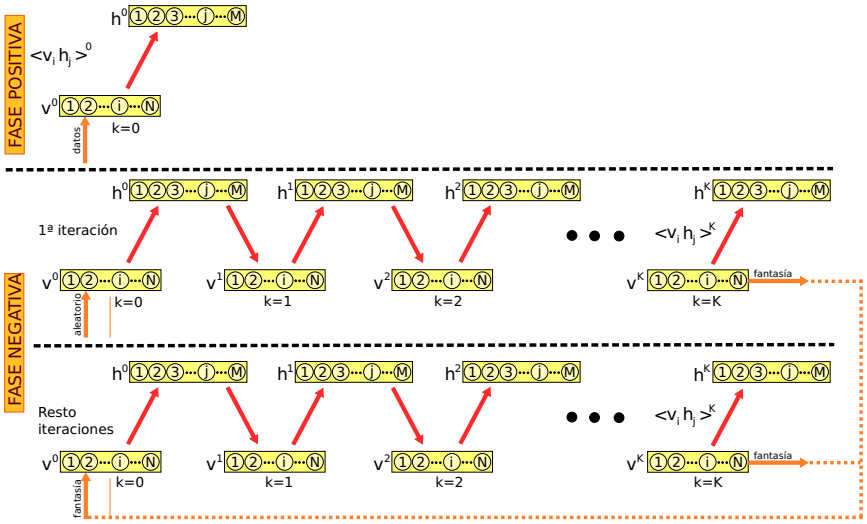


Figura 2.6: Algoritmo PCD sobre una RBM

de aprendizaje. Sin embargo, en el límite, cuando la tasa de aprendizaje es infinitesimalmente pequeña, la aproximación se convierte en exacta, por lo tanto, la aproximación realizada por este algoritmo requiere utilizar tasas de aprendizaje muy pequeñas.

La cadena de Markov puede truncarse pasado un cierto tiempo, (por ejemplo 10 iteraciones), aunque los estudios demuestran que los mejores resultados se obtienen sin truncar la cadena de Markov en ningún momento.

Máxima verosimilitud estocástica (SML)

Antes de la publicación del algoritmo CD-K, se había desarrollado otro método para el entrenamiento de redes de Boltzmann [33], con un cierto parecido con el ya descrito PCD, denominado Stochastic Maximum Likelihood o SML. En la mayoría de los estudios realizados [19], [7], los resultados obtenidos con SML son mejores que los obtenidos con CD-k, aunque no se ha planteado ningún estudio comparativo con PCD, con el que comparten muchas características. En nuestro caso y dadas las escasas diferencias entre ambos algoritmos, hemos utilizado ambos en nuestro estudio.

La idea subyacente en SML es prácticamente la misma que en PCD, utilizar una cadena de Markov persistente que intentará seguir las evoluciones de la respuesta intrínseca del modelo de red para realizar un des-aprendizaje lo más correcto posible, sin embargo, a diferencia de en PCD, la fantasía no se crea a partir de unos datos aleatorios gaussianos, sino que procede de los datos utilizados en la primera iteración del algoritmo. De esta forma, el $\mathbf{v}^K$ calculado en la primera iteración, pasa a ser la fantasía para las siguientes iteraciones. (Ver Figura 2.7)

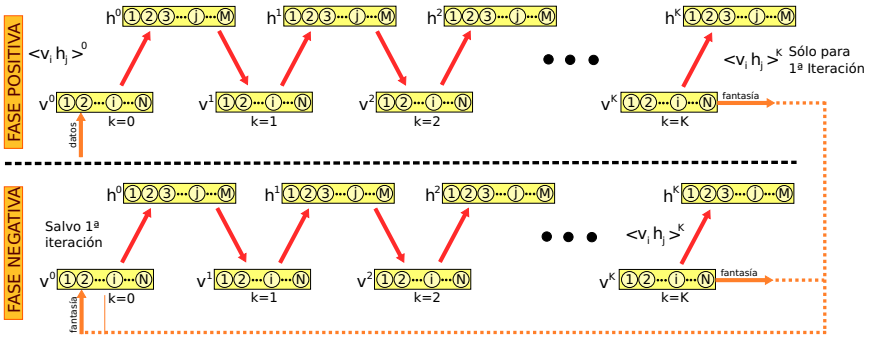


Figura 2.7: Algoritmo SML sobre una RBM

2.3. Belief Networks

La principal limitación que tiene el trabajar con RBM, solucionado el tiempo de entrenamiento, deriva de la complejidad de las distribuciones que son capaces de modelar, al utilizar una única capa oculta, lo que es lo mismo, una única capa de detectores de características, el modelo es demasiado rígido para adaptarse a no linealidades de alto orden, lo que nos plantea la necesidad de utilizar un mayor número de capas ocultas.

Sin embargo, la utilización de un mayor número de capas ocultas (Ver Figura 2.8), nos devuelve de lleno al problema del entrenamiento. Para ello, Geoffrey Hinton en [15] plantea considerar una red neuronal profunda (con varias capas ocultas) como un apilamiento de RBM en la que la primera máquina sea entrenada con los datos de entrada a modelar, mientras que las subsiguientes, toman como datos de entrada la probabilidad de activación de la capa oculta de la máquina anterior frente a los datos de entrenamiento (Ver Figura 2.8). Este esquema, fue denominado por Hinton como red de creencia profunda o Deep Belief Network (DBN)

La convergencia de este tipo de redes queda demostrada al ser convergentes las máquinas de Boltzmann que la componen, pero la aplicación del algoritmo Contrastive Divergence, al tratarse de una aproximación, hace necesario, en algunos casos, un repesado final de la red, utilizando métodos tradicionales como backpropagation, u otros diseñados específicamente para DBN como Wake-Sleep [14], o su variante Up-Down.

2.3.1. Entrenamiento de DBN

El entrenamiento de una DBN se puede dividir en dos fases, la segunda de ellas opcional, pero recomendable. En la primera de ellas, la red se entrena por capas, agrupando las capas de dos en dos formando RBMs, y utilizando el algoritmo de Contrastive Divergence mencionado en el capítulo anterior. Posteriormente a este entrenamiento, que nos acerca a una solución óptima, deberemos realizar un entrenamiento tradicional y más costoso

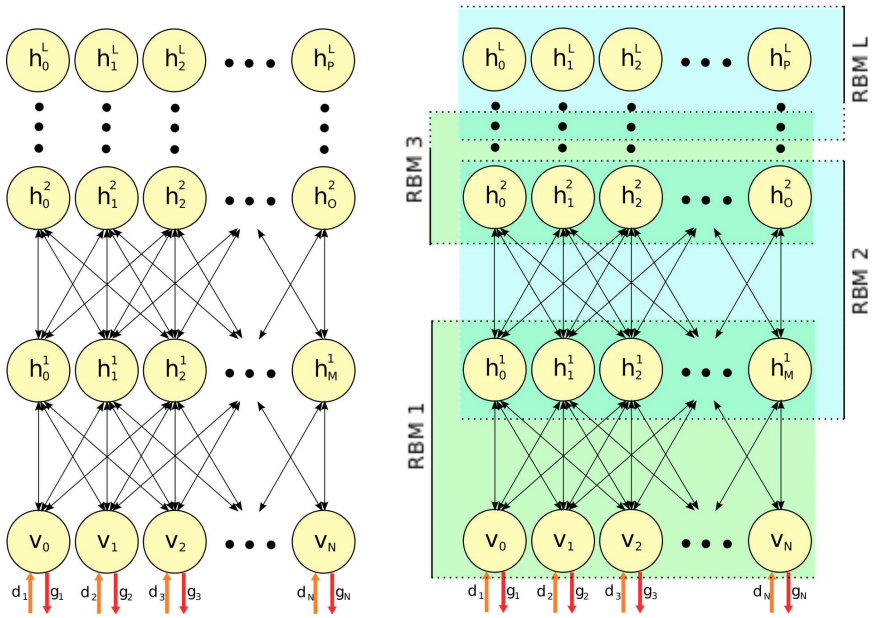


Figura 2.8: Red de creencia profunda o DBN (derecha) y su división en RBMs (izquierda)

computacionalmente, como Up-Down, que explicaremos más adelante.

En la primera parte, los pesos de reconocimiento (pesos que generan una distribución de probabilidad en una capa superior a partir de datos en una inferior) entre v y h, se mantienen iguales a los de generación (los que realizan el proceso contrario, infiriendo un estado para las capas inferiores a partir del estado de las superiores), por lo que $W_{rec}^{T} = W_{gen}$ para todas los pares de capas. El algoritmo básico de entrenamiento (para un modelo más completo ver sección A) se podría resumir de la siguiente forma:

1. Al principio, se comienza entrenando una RBM formada por la capa 1 y la 2 a partir de los datos de entrada

2. Entrenada la primera RBM, se introducen de nuevo los datos, para calcular la distribución de probabilidad generada en la capa oculta.

3. Esta distribución se toma como dato de entrada para el entrenamiento de la RBM formada por la capa 2 y 3.

4. El proceso se repetiría hasta entrenar todas las capas.

Una vez repetido este proceso para todo el conjunto de entrenamiento, tenemos una red profunda entrenada al conjunto de los datos de entrada, pero de una manera no óptima. Para completar el proceso de entrenamiento, tenemos que hacer uso del algoritmo Wake-Sleep o su variante para DBN, Up-Down.

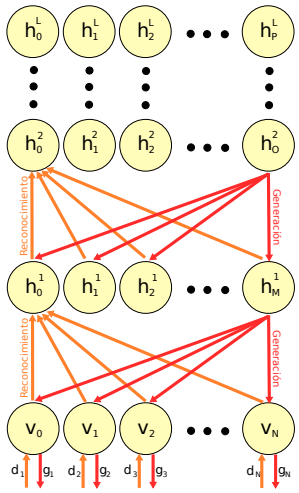


Figura 2.9: Separación entre pesos de reconocimiento (naranja) y generación (rojo)

2.3.2. Algoritmo Up-Down

Una vez que se han aprendido todos los pesos en todas las capas de una DBN como un apilamiento de RBM, tenemos una buena red, pero no la óptima, esto se debe a que ni los pesos, ni el procedimiento utilizado son óptimos para las capas bajas. El no utilizar una solución óptima puede ser no demasiado grave en casos como la clasificación, donde un cierto grado de error puede ser más beneficioso que un sobreajuste de la red. Pero en casos como la generación de datos a partir de un modelo, puede dar lugar a verdaderos problemas.

Para llegar a una solución todavía más ajustada, se utiliza una variante del algoritmo wake-sleep [14] denominada por G. Hinton como el algoritmo Up-Down. Tras haber obtenido unos buenos parámetros de inicialización, el primer paso es separar los pesos utilizados en reconocimiento de los utilizados en generación para todas las RBM que componen la DBN salvo para la última, (Ver Figura 2.9) puesto que es necesario actualizarlos por separado. Dado que $w_{ij} \neq w_{ji}$ ya no podemos utilizar la ecuación (2.8) para la actualización de pesos, sino que debemos de usar directamente (2.7) que, aplicada a nuestro caso, quedaría de la siguiente forma:

$$\frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} =< h_j^0(v_i^0 - v_i^1) > \tag{2.14}$$

que, aplicando una tasa de aprendizaje, vendría a convertirse en:

$$w_{ij}^t = w_{ij}^{t-1} + \epsilon(h_j^0(v_i^0 - v_i^1))^{t-1} \tag{2.15}$$

A esta ecuación se le pueden aplicar y de hecho en la mayoría de casos se aplican, de igual modo, todos los métodos descritos en el anexo A.

Teniendo en cuenta todo lo dicho hasta el momento, para cada uno de los datos de entrenamiento, el proceso a seguir para la actualización de los pesos es muy sencillo:

1. Separamos los pesos de generación de los de reconocimiento.
2. Tomando un dato de entrada $\mathbf{v}^0$, calculamos los estados en cada una de las capas de la DBN usando los pesos de reconocimiento. Obteniendo así $\mathbf{h}^0$ para cada una de las capas.
3. A partir de cada una de los estados $\mathbf{h}^0$ de cada una de las capas, calculamos, usando los pesos de generación, una fantasía en la capa inmediatamente inferior $\mathbf{v}^1$.
4. Actualizamos los pesos de generación en cada RBM excepto en la última usando la ecuación (2.15).
5. Realizamos un entrenamiento RBM convencional en las dos últimas capas de la DBN.
6. Tomando el estado de la última iteración del alternating Gibbs sampling, propagamos el estado de la última capa hasta la primera, usando los pesos de generación y creando fantasías en todas las capas. Conseguimos así $\mathbf{v}^2$ y un $\mathbf{h}^2$ para cada una de las capas ocultas.
7. Partiendo de la fantasía generada en cada una de las capas, calculamos, usando los pesos de reconocimiento, una fantasía en la capa inmediatamente superior $\mathbf{h}^3$.
8. De forma similar y modificando la ecuación (2.15), actualizamos los pesos de reconocimiento como $w_{ji}^t = w_{ji}^{t-1} + \epsilon v_i^0(h_j^0 - h_j^1)$.
9. Repetimos de nuevo el proceso tomando un nuevo dato de entrada.

Como puede observarse, el proceso puede dividirse claramente en tres fases, en la primera se hace una pasada de abajo hacia arriba en la DBN para actualizar los pesos de generación, en la segunda se entrena una RBM en la capa superior y en la última se hace una pasada de arriba a abajo, actualizando los pesos de reconocimiento. Todo este proceso puede verse de una forma más clara en la figura (2.10)

2.3.3. Modelo no supervisado frente a supervisado

Hasta el momento hemos venido hablando de métodos de aprendizaje de datos, pero en ningún momento nos hemos aproximado al problema de la clasificación. Las máquinas de Boltzmann, así como las DBN, fueron diseñadas en sus orígenes como generadores de datos, es decir, tras haber recibido una

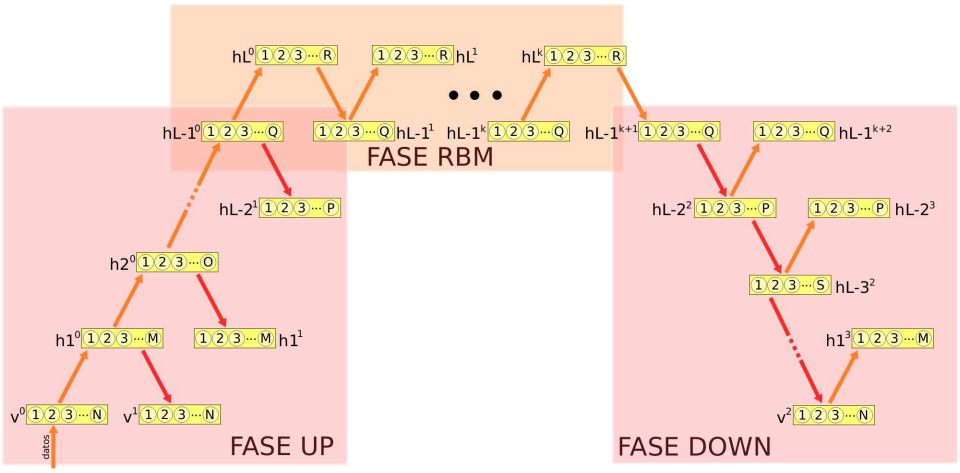


Figura 2.10: Algoritmo UP-DOWN

suficiente cantidad de datos, estas redes son capaces de generar unas nuevos datos coherentes con los datos originales. A este tipo de redes se les denomina redes no supervisadas, ya que no requieren para su entrenamiento de ninguna señal externa que clasifique los datos de entrada como pertenecientes a un grupo o a otro.

Para realizar tareas de clasificación propiamente dichas, las DBN deben de modificarse ligeramente de forma que permitan la entrada de etiquetas asociadas a los datos. Según el método descrito en [15], lo más aconsejable es modificar la penúltima capa de la DBN, añadiendo las etiquetas a la distribución generada por los datos en esa capa, como se puede ver en la figura (2.11). La idea detrás de este método es poder modificar el estado de las dos últimas capas de la red, que vendrían a ser como la memoria del sistema, logrando de esta forma seleccionar con las etiquetas, las activaciones de la memoria que corresponden a cada uno de los dígitos.

Es importante remarcar que las neuronas que se añaden no son el mismo tipo de neuronas que hemos venido utilizando (sigmoideas), sino que, al contrario, se trata de neuronas soft-max. Este tipo de neuronas presenta un comportamiento tendente a que una sola de las neuronas sea la que presente el estado encendido.

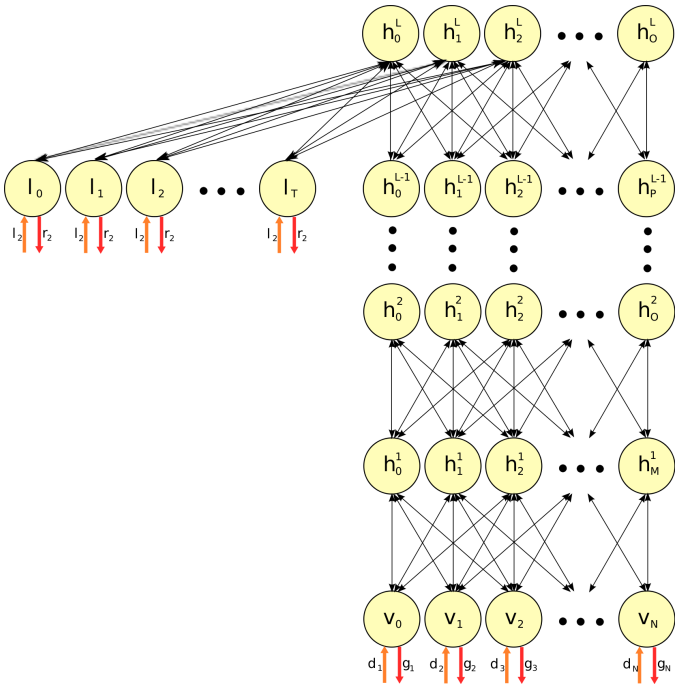


Figura 2.11: Red de creencia profunda supervisada

Capítulo 3

Modelos kernel y máquinas de vectores soporte (SVM)

3.1. Introducción

Previamente al nacimiento de las redes de creencia profunda o DBN, los métodos que más interes suscitaron en el ámbito de la inteligencia artificial fueron los denominados métodos kernel, sobretodo por los buenos resultados conseguidos por el que se ha considerado como el primer método kernel, las máquinas de vectores soporte o support vector machines (SVM).

3.2. Principios teóricos

El elemento primordial de todo método kernel es lo que se denomina la función kernel, que sirve como mecanismo de representación de la información de entrada al algoritmo. Obviando las condiciones matemáticas que tienen que cumplir este tipo de funciones y que se pueden encontrar fácilmente en la bibliografía, diremos que la función kernel se ocupa de transformar los datos de entrada a un nuevo espacio de características donde es más eficaz la clasificación. Esto nos permite una gran variedad de aplicaciones, ya que diseñando una función kernel adecuada, podemos utilizar el mismo algoritmo de clasificación para aplicaciones tan dispares como el tratamiento de imágenes a la clasificación de riesgos en inversiones.

Los métodos basados en funciones kernel proponen una forma de representar la información radicalmente diferente a los métodos tradicionales. En lugar de asociar a cada elemento del dominio de entrada una representación en el espacio de características, se utiliza una función kernel que calcula la similitud de cada pareja de objetos del conjunto de datos de entrada. Al aplicar la función kernel obtenemos es una matriz K de $n \times n$ valores reales correspondientes a aplicar la función k a cada una de las posibles parejas de datos de entrada, de forma que $K_{i,j} = k(x_i, x_j)$.

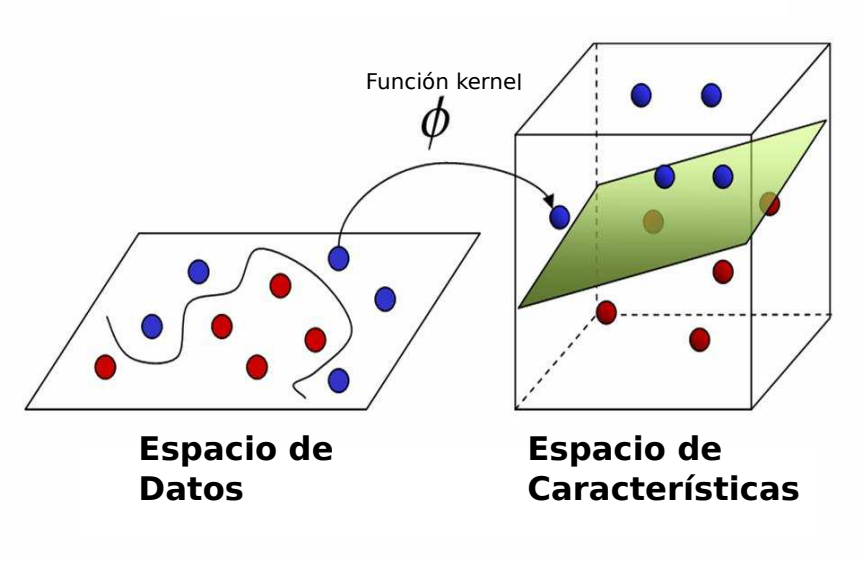


Figura 3.1: Ejemplo de transformación del espacio de datos al de características

Esta nueva representación de los datos permite separar de alguna forma la implementación del algoritmo de análisis de la representación de los datos y, lo que es más importante, permite que algoritmos de análisis relativamente sencillos encuentren relaciones complejas entre objetos de una manera extremadamente eficiente. Como podemos ver en la figura (3.1), una función kernel adecuada puede transformar una clasificación complicada como es la que tenemos en el espacio de datos a una extremadamente simple en el espacio de características.

Existen multitud de funciones kernel aplicables a distintos tipos de datos y no vamos a entrar a valorar cada una de las posibles variantes, tan sólo diremos que en nuestro estudio de las señales de EEG hemos utilizado un kernel lineal, que únicamente calcula el producto escalar de los datos de entrada. Este kernel ha mostrado buenos resultados en clasificación y por ello y por su simplicidad ha sido el escogido en nuestro estudio.

3.3. Máquinas de vectores soporte

El método kernel más utilizado son las máquinas de vectores soporte o SVM (support vector machine). Fueron introducidas por Vapnik a principios de los noventa [10], [8] como una nueva forma de afrontar las tareas de aprendizaje. En lugar de tratar de construir hipótesis que cometan pocos

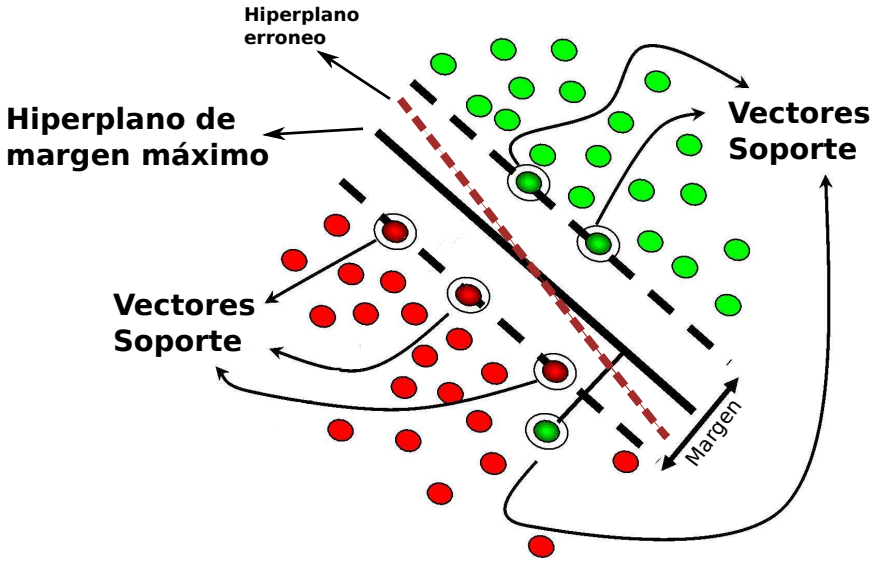


Figura 3.2: Ejemplo de transformación del espacio de datos al de características

errores al evaluar los datos, este método pretende producir predicciones en las que se puedan tener un alto grado de confianza a pesar de producir algunos errores. Es la diferencia entre lo que se denomina la Minimización del Riesgo Empírico (ERM), frente al enfoque de las SVM, que siguen el principio de Minimización del Riesgo Estructural (SRM). Dicho de otra forma, se trata de potenciar un modelo que tenga poca probabilidad de producir errores ante datos futuros, sacrificando el que se puedan producir algunos errores en los datos de entrenamiento actuales.

Originalmente, las máquinas de vectores soporte fueron diseñadas para ser capaces de distinguir entre únicamente dos clases de datos de entrada. Aunque esto ha ido cambiando a lo largo de los años, vamos a centrarnos en éste caso para comprender mejor la lógica de las SVM.

Tomando el caso sencillo de dos conjuntos de datos linealmente separables, existen muchos hiperplanos capaces de separar las clases, sin embargo, las SVM tratan de encontrar aquel que presente el menor riesgo de que en el futuro y con nuevas muestras, haya confusión entre los datos. Para ello, trata de buscar el hiperplano que tenga el margen máximo a las muestras más cercanas de las dos clases. Estas muestras que dan idea del margen de separación de las clases, son las denominadas vectores soporte (Ver Figura (3.2))

Capítulo 4

Conjuntos de datos a Analizar

4.1. Introducción

En esta sección vamos a detallar las características y diferencias entre los grupos de datos que se han utilizado durante la realización del proyecto. Fundamentalmente se han utilizado tres conjuntos de datos para probar la efectividad de las redes DBN y compararlas con los resultados ofrecidos tanto en los artículos como los logrados por otros algoritmos como las máquinas de vectores soporte. El primero de ellos, se ha utilizado para los tests de programación, así como para analizar la convergencia del método. El segundo se utilizó para comparar los resultados obtenidos con los presentes en los artículos de revistas científicas que hemos analizado, mientras el tercero, que ha centrado gran parte de nuestro trabajo, se utilizó para comparar los resultados de las DBN con los obtenidos con las SVM.

4.2. Datos de test

Para realizar los primeros ajustes en la red así como para comprobar la convergencia del método empíricamente se creó un pequeño conjunto de datos de 150 imágenes de 6x6 pixeles clasificadas en 3 tipos distintos. Estos tipos pueden verse en la parte superior de la figura 4.1. A estas imágenes, se le añadió un ruido gaussiano de varianza 0.1 para diferenciar un dato particular de otro y convertir los valores en reales. Al añadirle este ruido, se hizo necesaria una normalización que devolviera los valores que eran mayores que 1 y menores que 0 al rango (0,1). Para ello, se realizó una traslación y una compresión de los datos, de forma que el mínimo pasara a ser 0 y el máximo 1. No se realizó ningún tipo de saturación, por lo que la distribución de probabilidad global de los datos pasó a ser la de la figura 4.2. Un ejemplo de cada uno de los datos de entrada finales puede verse en la parte inferior de la figura 4.1.

Para introducir estos datos en la red neuronal, es necesario convertir estas

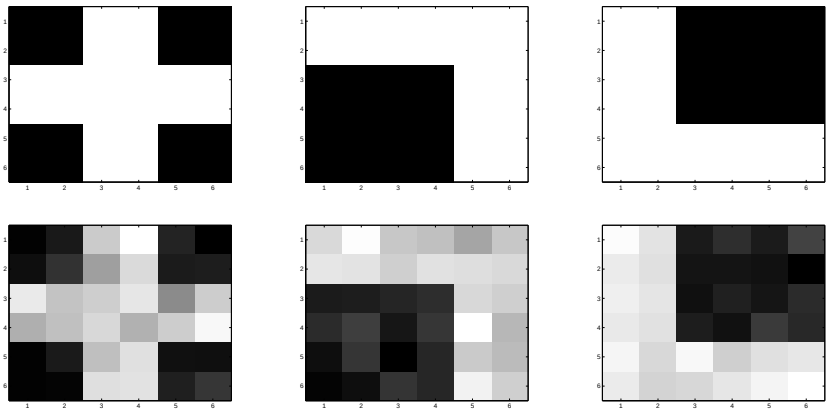


Figura 4.1: Clases del conjunto de test sin ruido y con ruido

imágenes en un vector de datos. El orden en el que se lea la imagen para dar lugar al vector es totalmente indiferente, siempre que todos los datos se lean en el mismo orden. Es por ello que se pierde completamente la información de proximidad entre los pixeles.

4.3. Base de datos del MNIST

La base de datos del MNIST [32] contiene 60.000 imágenes de entrenamiento y 10.000 imágenes de test de dígitos escritos a mano del 0 al 9 por alrededor de 250 autores distintos. Los dígitos han sido previamente normalizados en tamaño y centrados en un cuadrado de 28x28 pixeles. La intensidad de cada uno de los pixeles también ha sido normalizada de forma que se sitúe entre 0 y 255. Para utilizarlos en nuestra red, únicamente tenemos que normalizar los valores de los dígitos entre 0 y 1. Podemos observar un ejemplo de los datos de la base de datos MNIST en la figura 4.3

La importancia de esta base de datos radica en los continuos experimentos que se han venido realizando con distintos algoritmos, lo que permite una muy fácil comparación entre las distintas tendencias en reconocimiento de imágenes. Nuestro principal objetivo es comprobar que los resultados descritos en las publicaciones científicas son acordes con los que obtenemos en nuestra red. Esto nos permite ajustar los parámetros y familiarizarnos con el entorno para estar preparados para analizar los datos de EEG.

4.4. Datos de EEG

Los datos provenientes de electroencefalogramas (EEG) son un método no invasivo para medir la actividad cerebral durante el proceso cognitivo.

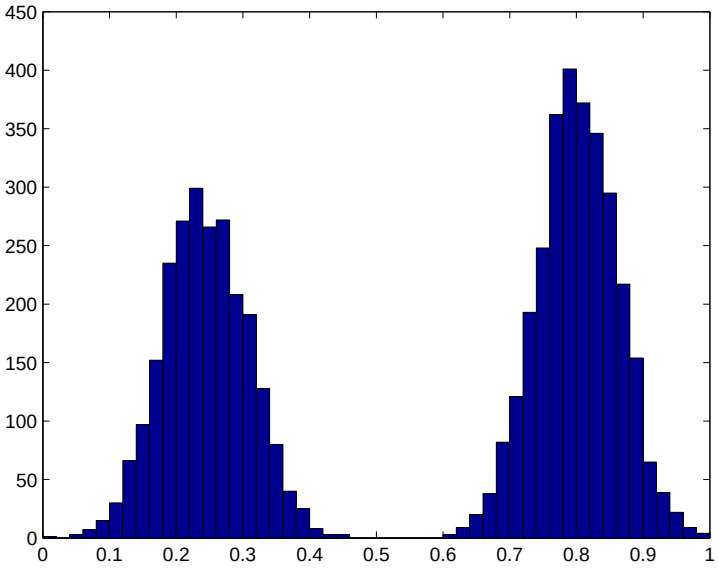


Figura 4.2: Histograma del conjunto de datos de test

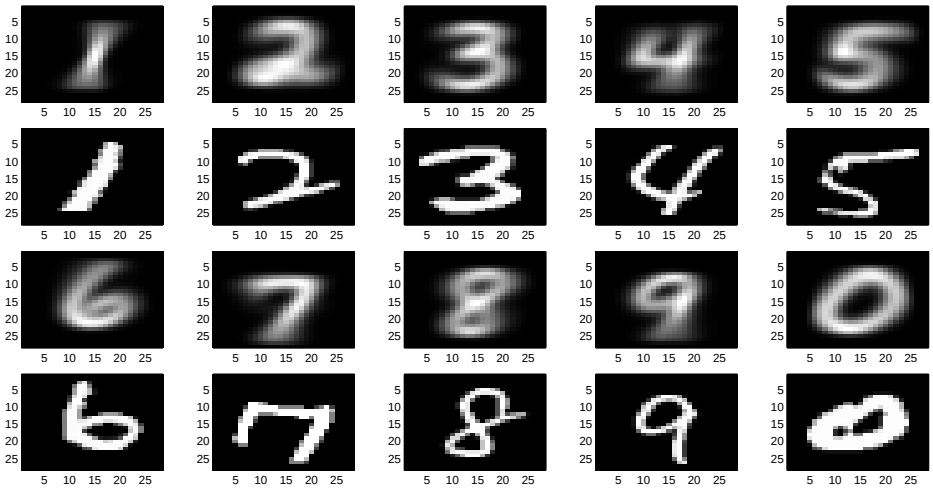


Figura 4.3: Media y un ejemplo concreto de cada uno de los dígitos de la base de datos del MNIST

Figura 4.4: Montaje para la grabación de las señales de EEG

Los potenciales eléctricos están ligados en tiempo a unos estímulos presentados al sujeto, sujeto que activa inconscientemente una o varias partes de su cerebro en respuesta al estímulo. Estas respuestas, quedan grabadas en 32 receptores situados alrededor de la cabeza del sujeto, para su posterior estudio y clasificación.

Entre los distintos tipos de potenciales que se pueden medir, son de particular interés los potenciales de error (ErrP). Estos potenciales se generan cuando se produce una disconformidad entre lo que el usuario espera y lo que que realmente ocurre. Existen diversos tipos, por ejemplo, cuando un sujeto se da cuenta de haber cometido un error al realizar una tarea bajo un tipo de presión externa (ErrP de respuesta), al dársele al sujeto un estímulo que le dice que ha cometido un error (ErrP de realimentación), cuando el sujeto observa un error cometido por otra persona (ErrP de observación) o cuando el sujeto manda una orden y una máquina ejecuta otra (ErrP de interacción).

El grupo de robótica de la Universidad de Zaragoza, como se puede ver en [17], se encuentra trabajando en clasificadores de esta clase de potenciales de error para su aplicación al control y aprendizaje en robótica. En este caso concreto (ver figura 4.4) el sujeto es presentado frente a una pantalla en la que se muestra las 5 posibles posiciones hacia las que es posible que el robot se mueva. Se considera que la central (posición 3) es la correcta, mientras que las posiciones más cercanas al centro (posición 2 y 4) se consideran errores leves y las posiciones más alejadas (posición 1 y 5) son consideradas errores graves.

El conjunto de datos está formado por las señales ERP de 32 canales de 3 sujetos distintos enfrentados a 500 movimientos del robot, distribuidos

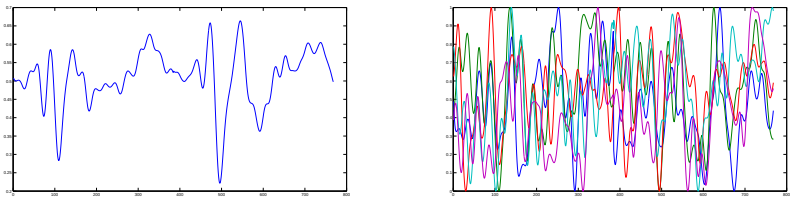


Figura 4.5: Media de la señal de EEG y 5 ejemplos de señal

equitativamente entre las 5 posiciones finales y por una señal de control, sincronizada a los datos, que indica el caso mostrado al sujeto. De esta forma, podríamos decir que tenemos 100 casos correctos, 200 casos pertenecientes a errores leves (100 error leve a derecha, 100 error leve a izquierda) y 200 casos pertenecientes a errores graves (100 a derecha y 100 a izquierda). Estas señales son grabadas durante el segundo y medio posterior a la presentación del estímulo, muestreadas a 256Hz y posteriormente filtradas para eliminar la señal entre 0.5 y 10Hz. Es posible también conseguir otras 1.236 muestras pertenecientes a la no actividad del sujeto, grabadas durante los intervalos de descanso entre muestras.

Las posibilidades de clasificación son varias, aunque nos hemos centrado en el estudio de cada sujeto por separado, ya que tratar de clasificar simultáneamente las señales cerebrales de varios sujetos, se presenta excesivamente complicado. De entre todas las posibilidades restantes, nos hemos centrado en tres principalmente:

1. Distinguir, error frente a no error (100 muestras correctas frente a 400 incorrectas)

2. Separar error frente a no error y frente a no actividad (100 muestras correctas, 400 incorrectas y 1.236 de no actividad)

3. Clasificar cada una de las cinco señales de movimiento del robot

El hecho de estar tratando, aun con un único sujeto, un tamaño importante en los datos de entrada, concretamente 32 canales de 384 muestras cada uno, nos obliga a reducir el número de datos de entrada. De no ser así, una DBN tendría que contener en su primera capa 12.288 neuronas, (el MNIST usaba 784), y el tamaño de las matrices de pesos y el tiempo de entrenamiento para lograr una solución correcta sería excesivo.

De igual forma, la gran complejidad de la señal, tanto por su alto nivel de ruido, como por su gran variabilidad (ver figura 4.5, nos obliga a realizar un complejo preprocesado de la señal.

Teniendo en cuenta las numerosas posibilidades a la hora de reducir la dimensionalidad como a la hora de escoger la normalización o el tratamiento

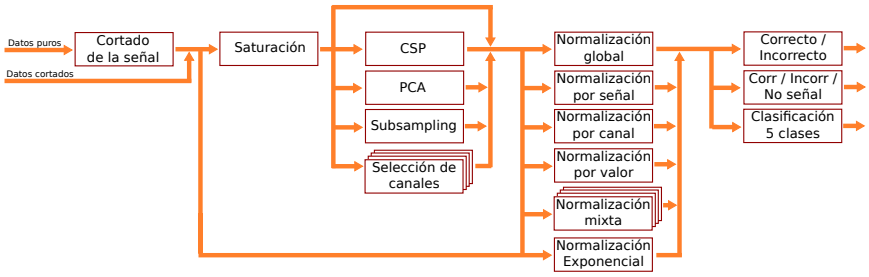


Figura 4.6: Esquema de las diferentes opciones para preprocesar la señal

de los datos, podemos decir que estamos trabajando con un set de test muy extenso, como se puede ver en la figura 4.6. Más información sobre los distintos procesos de normalización, reducción del tamaño de datos y selección inteligente de los canales a utilizar puede encontrarse en el anexo B.

Capítulo 5

Tests

5.1. Introducción

En esta sección vamos a tratar de presentar de la manera más resumida posible todos los resultados que se han obtenido en el análisis de los distintos tipos de conjuntos de datos. Aunque se presentan los análisis para los tres conjuntos de datos mencionados en el capítulo anterior, hay que reseñar que el análisis de EEG presenta a su vez un conjunto casi infinito de subconjuntos de datos, ya que cada una de las tres posibles tareas de clasificación mencionadas anteriormente, presenta, como se comenta en los anexos, numerosas posibilidades de normalización y de transformación de los datos. Es por ello que se presentan los mejores resultados obtenidos para cada uno de los tres problemas de clasificación mencionados anteriormente.

5.2. Test en generación

Las redes de creencia profunda o DBN, como ya se comentó en su momento, nacieron como un tipo de redes capaces de aprender la distribución de probabilidad subyacente en los datos y generar nuevas muestras a partir de esta distribución.

Una vez entrenada una DBN resulta relativamente sencillo extraer muestras de ella, basta con dar valores aleatorios a la penúltima capa de la DBN y fijar la etiqueta que representa a la clase de la que queremos generar nuevas muestras. Tras varios procesos de muestreado alternativo en las últimas capas (dejando siempre fijas las etiquetas), la red irá tendiendo hacia un equilibrio en las últimas capas (Ver figura 5.1). A partir de ese equilibrio podemos empezar a utilizar los pesos de generación para dar valor a las capas inferiores, hasta generar una fantasía en la capa visible. Si la red está convenientemente entrenada, la salida debería ser muy similar al conjunto de datos de entrada.

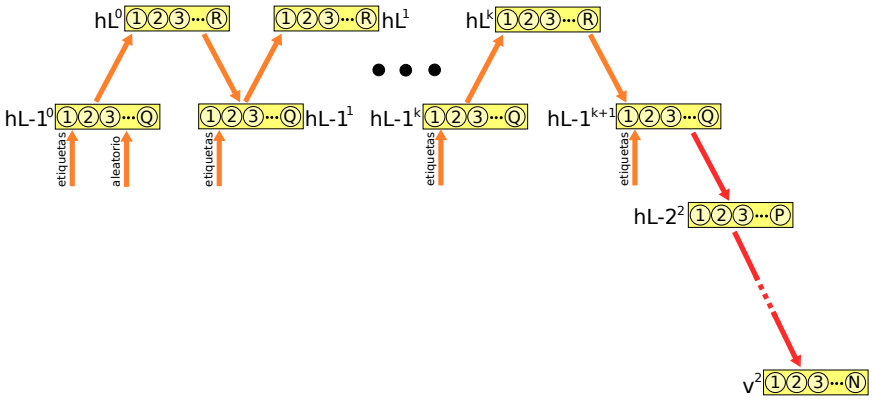


Figura 5.1: Funcionamiento de una DBN en generación

Una vez que la red ha alcanzado el equilibrio en las capas superiores, la red, al ser estocástica, irá generando muestras diferentes de la misma clase.

5.3. Test en reconocimiento

El test en reconocimiento es el más importante y el que más ha centrado nuestros esfuerzos, ya que es en él donde realmente se prueba la capacidad clasificadora de la red. A lo largo de la literatura se presentan varios métodos de probar la red en reconocimiento, el más utilizado comienza tomando la muestra a analizar y alimentando con ella la red. Posteriormente, podemos muestrear una a una las capas utilizando los pesos de reconocimiento hasta llegar a la penúltima capa. Llegados a este punto e inicializando las etiquetas a un valor aleatorio, se realiza un sampleado alternativo de Gibbs hasta que se considere que la red ha alcanzado la estabilidad. Durante este proceso, únicamente se permite cambiar de valor en la penúltima capa a las neuronas que representan las etiquetas, el resto quedan fijas con la señal proveniente de las capas inferiores. Una vez estable la red, podemos extraer el valor de las etiquetas de las neuronas destinadas a ello. Todo este proceso puede verse en la figura 5.2.

Los resultados se validan con dos subconjuntos de datos, el primero extraído del conjunto de datos de entrenamiento, y el segundo de ellos es un conjunto de datos que no ha sido utilizado para el entrenamiento y que por lo tanto la red no conoce, denominado conjunto de datos de test.

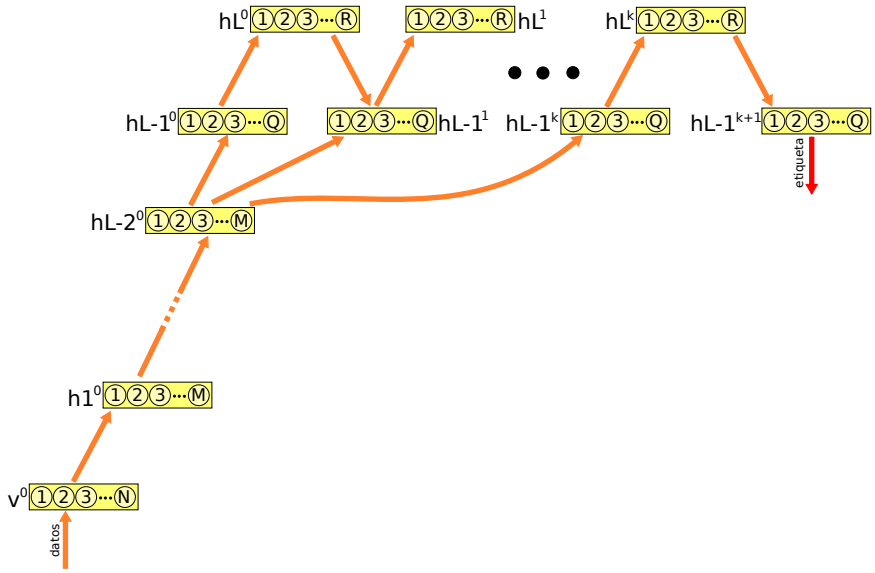


Figura 5.2: Funcionamiento de una DBN en Reconocimiento

5.4. Resultados

5.4.1. Datos de test

Para los datos de test hemos entrenado una red de creencia profunda de 4 capas con la distribución mostrada en la figura 5.3. La primera capa tiene tantas neuronas como la dimensión de los datos, 36, la segunda y tercera tienen 20 neuronas y la última capa tiene 100. La capa lateral, en la que se introducen las etiquetas, tiene 3 neuronas. En su entrenamiento hemos usado PCD, utilizando 40 pasadas de los datos para el entrenamiento de cada RBM. La tasa de aprendizaje se ha fijado en 0.1, el decaimiento en 0.001, el momento en 0.3 para el primer 70 % del entrenamiento y de 0.8 para el último 30 %. El tamaño de batch, teniendo en cuenta el pequeño número de muestras del que disponemos, se ha fijado en 10 muestras. Dado que los resultados obtenidos han sido inmejorables, no hemos utilizado ningún tipo de repesado final.

En el conjunto de datos de entrenamiento, el porcentaje de acierto ha sido del 96,15 % obteniendo la siguiente matriz de confusión (en tanto por ciento):

$$\begin{pmatrix} 95{,}122 & 2{,}2727 & 2{,}2222 \\ 0 & 97{,}727 & 2{,}2222 \\ 4{,}878 & 0 & 95{,}556 \end{pmatrix}$$

En el conjunto de datos de test el porcentaje de acierto es un poco inferior, del 94,67 %, con una matriz de confusión dada por:

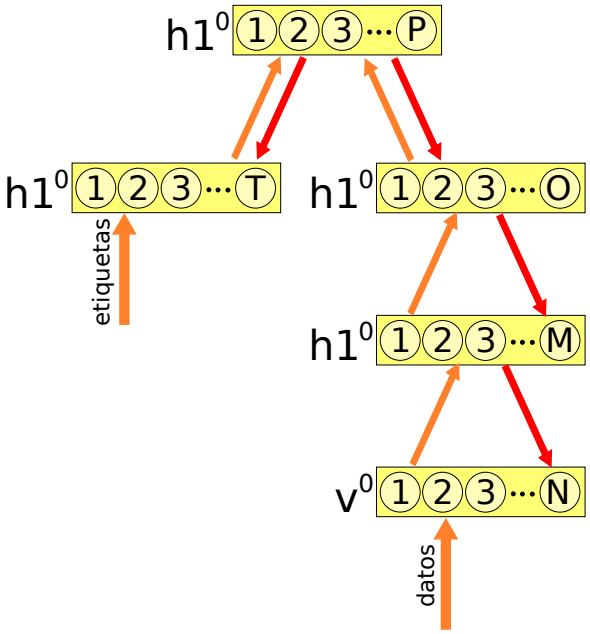


Figura 5.3: Esquema general de la DBN usada para las pruebas

$$\begin{pmatrix} 96 & 4 & 4 \\ 0 & 92 & 0 \\ 4 & 4 & 96 \end{pmatrix}$$

La generación de datos en el conjunto de datos de test funciona de manera fluida, dando muestras prácticamente iguales a las que se utilizaron para el entrenamiento (Ver figura 5.4)

Figura 5.4: Ejemplo de datos generados por la DBN

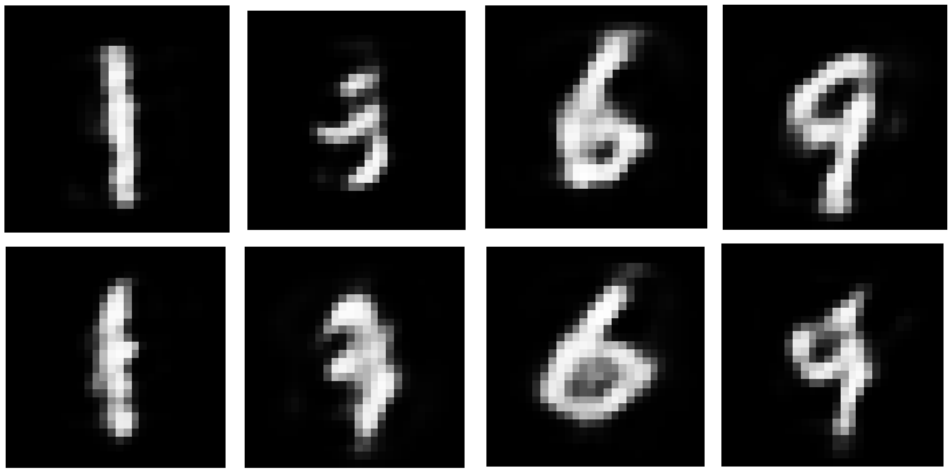


Figura 5.5: Ejemplo de dígitos generados por la DBN (1,3,6 y 9)

5.4.2. MNIST

El modelo de red utilizado para los datos del MNIST es el mismo mostrado en la figura 5.3, salvo que el tamaño de las capas varía. La primera capa tiene en este caso 784 neuronas, la segunda y tercera 500 y la última 2.000, mientras que la capa de etiquetas tiene 10 neuronas.

En cuanto a los parámetros de entrenamiento, realizamos 10 pasadas de los datos en el entrenamiento de cada RBM. Estos datos han sido agrupados en conjuntos de 100 muestras. Hemos utilizado PCD con k=3, una tasa de aprendizaje de 0.1, un decaimiento de 0.001 y un momento al inicio de 0.3, que cambia a 0.8 superado el 70 % del entrenamiento.

La generación de datos para el MNIST necesita de un mayor tiempo de estabilización en las capas superiores de la red, pero aun así, es posible generar muestras de la red sin ningún problema. En la figura 5.5 se muestran algunos ejemplos de números generados con la red DBN.

A la hora de reconocer, las DBN se comportan de una manera más que aceptable, obteniendo un 90,22 % de aciertos en el reconocimiento de los datos de entrenamiento. La matriz de confusión se muestra a continuación:

$$\begin{pmatrix}
95{,}65 & 0 & 1{,}14 & 0{,}4 & 0{,}1 & 1{,}41 & 0{,}77 & 0{,}39 & 0{,}4 & 0{,}49 \\
0{,}64 & 99{,}31 & 12{,}75 & 9{,}33 & 3{,}56 & 5{,}57 & 2{,}31 & 5{,}57 & 34{,}31 & 4{,}41 \\
0{,}08 & 0{,}28 & 76{,}56 & 1{,}41 & 0{,}13 & 0{,}18 & 0{,}17 & 0{,}34 & 1{,}08 & 0{,}3 \\
0{,}23 & 0{,}03 & 1{,}27 & 0{,}32 & 0{,}02 & 3{,}71 & 0{,}04 & 0{,}02 & 3{,}4 & 1{,}03 \\
0{,}3 & 0{,}1 & 2{,}35 & 0{,}27 & 90{,}24 & 1{,}23 & 0{,}86 & 1{,}67 & 2{,}28 & 9{,}43 \\
0{,}53 & 0{,}07 & 0{,}06 & 3{,}43 & 0{,}08 & 83{,}05 & 1{,}69 & 0{,}04 & 5{,}53 & 0{,}62 \\
2{,}08 & 0{,}13 & 1{,}73 & 0{,}74 & 0{,}91 & 2{,}28 & 94{,}08 & 0{,}02 & 1{,}39 & 0{,}04 \\
0{,}26 & 0{,}08 & 2{,}96 & 2{,}05 & 0{,}23 & 0{,}68 & 0{,}04 & 89{,}66 & 0{,}82 & 4{,}59 \\
0{,}06 & 0 & 0{,}34 & 0{,}72 & 0{,}02 & 0{,}16 & 0{,}02 & 0 & 45{,}44 & 0{,}07 \\
0{,}19 & 0 & 0{,}84 & 1{,}32 & 4{,}72 & 1{,}72 & 0{,}02 & 2{,}31 & 5{,}36 & 79{,}03
\end{pmatrix}$$

A la hora de reconocer los datos de test, los resultados son bastante similares, obtenemos un 89,49 % de tasa de acierto, con la matriz de confusión siguiente:

$$\begin{pmatrix}
95{,}68 & 0 & 1{,}34 & 0{,}88 & 0 & 0{,}59 & 0{,}47 & 0{,}16 & 0{,}34 & 0{,}27 \\
0{,}37 & 99{,}08 & 14{,}89 & 9{,}14 & 2{,}84 & 6{,}61 & 3{,}25 & 4{,}92 & 31{,}76 & 3{,}54 \\
0{,}07 & 0{,}27 & 71{,}09 & 0{,}98 & 0{,}21 & 0 & 0{,}03 & 0{,}06 & 0{,}48 & 0{,}3 \\
0{,}03 & 0{,}09 & 2{,}05 & 81{,}73 & 0{,}03 & 4{,}32 & 0 & 0{,}1 & 4{,}1 & 0{,}88 \\
0{,}14 & 0{,}06 & 2{,}18 & 0{,}46 & 90{,}96 & 2{,}77 & 1{,}15 & 1{,}98 & 3{,}28 & 11{,}35 \\
0{,}41 & 0 & 0{,}2 & 2{,}15 & 0{,}1 & 80{,}41 & 0{,}54 & 0{,}1 & 4{,}68 & 0{,}17 \\
2{,}36 & 0{,}03 & 1{,}71 & 0{,}95 & 0{,}99 & 3{,}28 & 94{,}45 & 0{,}03 & 2{,}15 & 0{,}07 \\
0{,}24 & 0{,}45 & 4{,}47 & 2{,}22 & 0{,}31 & 0{,}33 & 0{,}1 & 89{,}78 & 0{,}48 & 3{,}3 \\
0{,}1 & 0 & 0{,}64 & 0{,}23 & 0 & 0{,}18 & 0 & 0 & 47{,}66 & 0{,}07 \\
0{,}61 & 0{,}03 & 1{,}41 & 1{,}27 & 4{,}55 & 1{,}51 & 0 & 2{,}88 & 5{,}06 & 80{,}07
\end{pmatrix}$$

En los artículos que hemos encontrado los resultados eran algo mejores, en torno al 95-97 % de acierto, sin embargo, en todos ellos, se entrenaban varias redes con un tiempo de entrenamiento cercano a la semana. Mientras tanto, por limitaciones de hardware, nuestro tiempo de entrenamiento apenas ha superado las 10-12 horas.

5.4.3. EEG

A la hora de trabajar con las señales de EEG las DBN se han mostrado en general muy poco efectivas, como se puede comprobar en los anexos, se ha intentado realizar muchas variaciones en los datos que llevaran a una mejor clasificación. Sin embargo, tristemente, los resultados son malos, más aún si, como vamos a mostrar a continuación, los comparamos con los resultados obtenidos con SVM.

Reconocimiento de señales correctas e incorrectas

Se han probado numerosas configuraciones de red y muchas variaciones de parámetros, sin embargo, en ningún caso se han conseguido resultados dignos de destacar. En el mejor de los casos, usando una normalización por señal seguida de una normalización por valor, con 30 pasadas de los datos, un $k = 3$, una tasa de aprendizaje de 0.01, un momento entre 0.3 y 0.8 y un decay de 0.001 obtenemos en el análisis de los datos de entrenamiento apenas un 56,11 %, con una matriz de confusión dada por:

$$\begin{pmatrix} 66{,}389 & 54{,}167 \\ 33{,}611 & 45{,}833 \end{pmatrix}$$

Los resultados en el análisis de los datos de test son ligeramente mejores, un 65 % con la siguiente matriz de confusión:

$$\begin{pmatrix} 72{,}5 & 42{,}5 \\ 27{,}5 & 57{,}5 \end{pmatrix}$$

Simultáneamente, en la misma tarea de clasificación y con los mismos datos de entrada, la máquina de vectores soporte con un kernel lineal, consigue separar el 94,16 % de los datos de entrenamiento, con una matriz de confusión dada por:

$$\begin{pmatrix} 96{,}491 & 7{,}9365 \\ 3{,}5088 & 92{,}063 \end{pmatrix}$$

En cuanto al conjunto de datos de test, los resultados son peores, pero mucho mejores que los obtenidos con las DBN, manteniendo una tasa de acierto del 85 %, con la siguiente matriz de confusión:

$$\begin{pmatrix} 88{,}889 & 18{,}182 \\ 11{,}111 & 81{,}818 \end{pmatrix}$$

Reconocimiento de señales correctas, incorrectas y no señal

La idea detrás de incluir en el análisis la no presencia de ningún tipo de señal era permitir a la DBN descartar aquellas entradas que fueran aleatorias y que no aportaran nada al problema de clasificación. Sin embargo, los resultados no son tampoco los esperados.

Usando los mismos parámetros que en el apartado anterior, los mejores resultados se obtienen usando una normalización por valor seguida de una normalización por canal. En el caso de analizar el conjunto de dato de entrenamiento, obtenemos una tasa de acierto del 51,44 %, siendo la matriz de confusión:

$$\begin{pmatrix} 60 & 53{,}056 & 25{,}463 \\ 35{,}556 & 37{,}5 & 21{,}296 \\ 4{,}4444 & 9{,}4444 & 53{,}241 \end{pmatrix}$$

Simultáneamente, la tasa de acierto en el caso del conjunto de test es del 49,5 %, dada por esta matriz de confusión:

$$\begin{pmatrix} 47{,}5 & 47{,}5 & 25 \\ 45 & 47{,}5 & 24{,}167 \\ 7{,}5 & 5 & 50{,}833 \end{pmatrix}$$

Reconocimiento de cada señal por separado

En el caso del reconocimiento de las cinco señales por separado, presentamos los mismos problemas que anteriormente, la red es capaz de distinguir con cierta facilidad las señales correctas, pero comete numerosos errores en la distinción del resto de señales entre sí. Poniéndolo en números, los mejores resultados, tomando los parámetros anteriormente citados, los obtiene la combinación de una normalización por canal seguida de una normalización por valor.

En este caso, el porcentaje de acierto en los datos de entrenamiento es únicamente del 24,66 %, siendo la matriz de confusión:

$$\begin{pmatrix} 22{,}222 & 25{,}556 & 8{,}8889 & 24{,}444 & 15{,}556 \\ 25{,}556 & 14{,}444 & 11{,}111 & 10 & 18{,}889 \\ 28{,}889 & 26{,}667 & 45{,}556 & 30 & 33{,}333 \\ 15{,}556 & 15{,}556 & 14{,}444 & 17{,}778 & 8{,}8889 \\ 7{,}7778 & 17{,}778 & 20 & 17{,}778 & 23{,}333 \end{pmatrix}$$

En el caso del conjunto de datos de test los resultados son muy similares, una tasa de acierto del 26 % con la matriz de confusión siguiente:

$$\begin{pmatrix} 24 & 12 & 16 & 36 & 14 \\ 10 & 24 & 6 & 18 & 14 \\ 24 & 26 & 50 & 10 & 40 \\ 22 & 14 & 18 & 14 & 14 \\ 20 & 24 & 10 & 22 & 18 \end{pmatrix}$$

5.5. Test de convergencia

Vistos los malos resultados obtenidos en el análisis de EEG, nos planteamos estudiar los procesos internos que estaban teniendo lugar en las DBN, para intentar tener más pistas de lo que podía estar sucediendo.

El primer paso fue comprobar la convergencia de los pesos de la red, esto nos puede dar idea de si la red ha alcanzado un régimen estable o no. En este

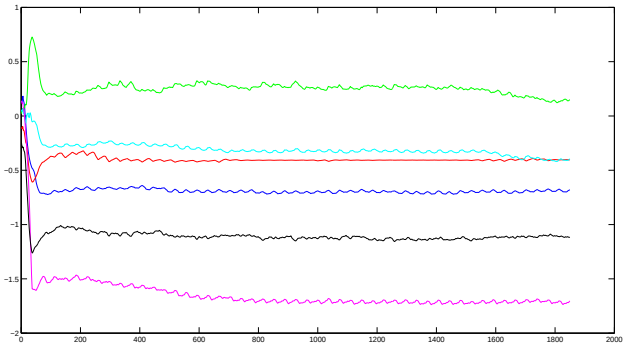


Figura 5.6: Variación de varios de los pesos de una DBN entrenándose con datos de EEG a lo largo del tiempo

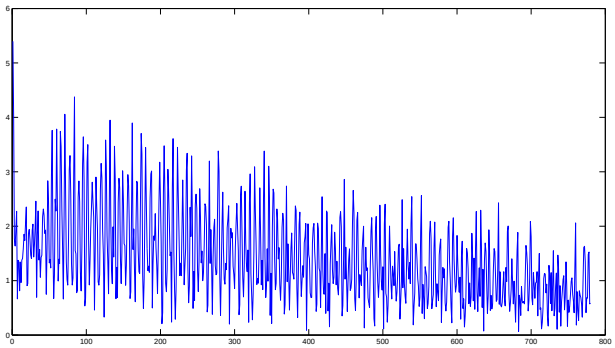


Figura 5.7: Variación de la energía de red a lo largo del tiempo

caso, tras comprobar la variación de varios de los pesos de la red durante 1.800 actualizaciones, pudimos ver, como se muestra en la figura 5.6, que los pesos habían convergido a una solución estable, aunque, vistos los resultados anteriores, no lo suficientemente óptima.

Comprobada la convergencia de los pesos, pasamos a analizar la energía de red, que, en media, debería disminuir conforme avanza el proceso de entrenamiento. Como se puede ver en la figura 5.7, el entrenamiento hace que la red se vaya moviendo hacia estados de menor energía, cumpliendo por tanto lo que era de esperar.

Tras desechar un mal funcionamiento de la red, los malos resultados nos pueden estar indicando que la red ha quedado atrapada en un mínimo de energía local. Algo que puede ser muy plausible al disponer de tan pocos datos de entrada en comparación con la dificultad de los datos. Por poner un ejemplo, en los dígitos del MNIST se dispone de 60.000 imágenes, mientras

que en nuestro caso apenas se disponen de 500, problema que se agrava al tener en cuenta la gran variabilidad que presentan las señales de EEG.

Capítulo 6

Fases del proyecto

Siguiendo los objetivos descritos en el primer capítulo, las primeras semanas de trabajo se dedicaron exclusivamente a la recopilación de información en forma de artículos de revista, videos, conferencias y presentaciones que trataban el tema de las redes de creencia profunda o DBN. A partir de ellos, se comenzaron a estudiar los antecedentes y los distintos algoritmos previos, para poner estas redes en su debido contexto. Por este motivo se estudiaron las redes de Boltzmann, las redes de Boltzmann restringidas y las redes de creencia.

Comprendidos los fundamentos de este nuevo tipo de redes, se comenzaron a intentar extraer de entre los numerosos artículos, los escasos detalles referentes a la implementación de las DBN en la realidad y se comenzaron a entender los fundamentos de los algoritmos contrastive divergence y los algoritmos de repesado final como el Up-Down. Conforme se iba avanzando en este estudio, se iban programando las primeras lineas de código para modelar las DBN en Matlab, así como se creaban un conjunto básico de datos test para comprobar su correcto funcionamiento.

Una vez validado este primer algoritmo, se pasó a probar con la base de dígitos escritos del MNIST, obteniendo resultados positivos y con las primeras señales de EEG, donde no se obtuvieron los resultados deseados. Para mejorar el reconocimiento de las señales de EEG, se decidió utilizar una nueva colección de datos de EEG y se comenzó a estudiar los posibles efectos de la normalización, la correcta selección de los canales de entrada, PCA, efectos como los parpadeos... etc. Paralelamente y visto que los resultados seguían sin ser positivos en EEG, se decidió implementar una DBN de mayor complejidad haciendo uso de PCD y las modificaciones descritas en los anexos. Esto permitía un entrenamiento más rápido y un ajuste más fino, con la esperanza de ser capaces de modelar con una mayor precisión la distribución de probabilidad de los datos.

Las pruebas con esta nueva red fueron correctas tanto en el conjunto de datos de prueba como en el MNIST, sin embargo los resultados obtenidos en

EEG distaban mucho de los obtenidos con otras técnicas como las máquinas de vectores soporte, por lo que se estudiaron las bases de las máquinas de vectores soporte y se probaron repetidamente sobre las distintas formas de preprocesar la señal, comprobando que los resultados ofrecidos por las máquinas de vectores soporte eran mucho mejores a los obtenidos por las DBN en todas las configuraciones probadas.

Dados los problemas de las DBN en separar incluso las señales correctas de las incorrectas, se decidió probar un método conocido como patrones espaciales comunes (Common Spatial Patterns o CSP) para intentar preprocesar la señal de forma que se facilitase a las DBN el reconocimiento. Este preprocesado mejoró el reconocimiento de la red, pero no llegó en ningún caso a superar los resultados obtenidos con las máquinas de vectores soporte.

Por último, se trató de realizar un estudio más fino de los parámetros de las DBN que obligó a una reestructuración completa del software para lograr analizar en profundidad las variaciones que cada uno de los parámetros podía aportar a la mejora del reconocimiento en EEG. Sin embargo, se comprobó que la red, en el caso de EEG, convergía siempre a unos valores que no eran óptimos, quedando atrapada en mínimos de energía locales.

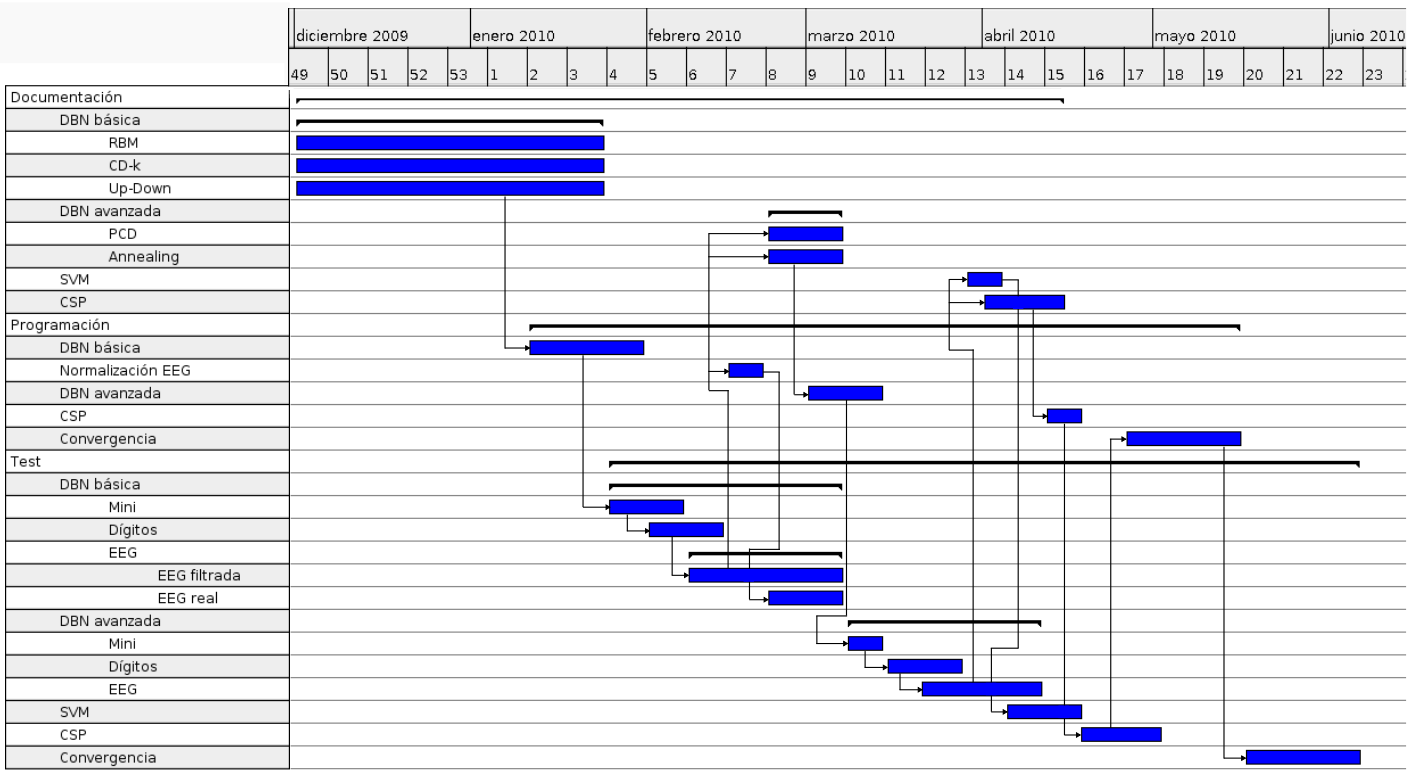


Figura 6.1: Diagrama de Gantt con las fases del proyecto y su distribución en el tiempo

Capítulo 7

Conclusiones y trabajo futuro

El objetivo del proyecto era desarrollar los algoritmos necesarios para evaluar el potencial de las DBNs en tareas de clasificación de potenciales de error medidos con EEG y su comparación con los clasificadores utilizados hasta el momento.

Para lograrlo, se ha realizado un estudio detallado del estado del arte, a partir de una recopilación de artículos publicados en conferencias internacionales acerca de las DBN y de la teoría que las sustentaban, como las RBM o las Boltzmann Machine, así como de los distintos métodos de entrenamiento y de sus diversas variantes. Esta recopilación y su estudio detallado, además de dar una explicación actualizada del estado del arte actual de las DBN y de sus futuras tendencias, ha permitido desarrollar una completa guía tutorial prácticamente inédita en el mundo de las DBN. En estos momentos, únicamente existe un tutorial, publicado durante la realización de este proyecto, acerca de las DBN, de similares características al que aquí se presenta [18].

El estudio en profundidad de las redes de creencia profunda y sus distintos algoritmos de entrenamiento nos ha permitido lograr una implementación de las DBN en Matlab que contiene un conjunto amplio de las distintas variantes presentadas en la literatura. Se ha desarrollado una librería propia (ver Anexo C que permite automatizar el proceso de entrenamiento de la red y su posterior funcionamiento y testeo, permitiendo la carga de los parámetros necesarios cómodamente desde ficheros de configuración. Este software permite además seleccionar de manera externa el algoritmo de entrenamiento que se desea utilizar de entre los varios descritos en los artículos científicos, siendo por tanto muy sencillo comparar los resultados obtenidos con cada uno de ellos. Esta librería es única por el momento ya que previamente a este trabajo únicamente había una versión de entrenamiento de las DBN muy rudimentaria proporcionada por Hinton en su página web [25]. Sin embargo, esta versión requería una modificación de los parámetros directamente en el código fuente y no implementaba algoritmos de entrenamiento como CD-K, PCD o SML ni permitía un correcto testeo de la red.

Para comprobar el correcto funcionamiento de las soluciones implementadas, se han creado conjuntos de datos de test y se han evaluado los resultados obtenidos en distintas publicaciones científicas sobre la base de datos del MNIST, siendo los resultados obtenidos con nuestro software comparables con los obtenidos por la comunidad científica.

De la misma forma que con el entrenamiento y testeo de las redes de creencia profunda, se han desarrollado programas para automatizar todos los procesos de preprocesado de la señal de EEG, desde los más simples como la normalización o saturación, hasta los más complejos como los Common Spatial Patterns. También se han validado los resultados obtenidos en el grupo de robótica de la Universidad de Zaragoza con los clasificadores SVM, permitiéndonos comprobar que el preprocesado realizado a la señal era el adecuado.

Una vez que la implementación de los algoritmos estaba completa y validada, se utilizo para clasificar los datos de EEG correspondientes a los potenciales de error y su comparación con el clasificador SVM. A pesar de los esfuerzos realizados a nivel de pre-procesamiento y de ajuste de parámetros de la DBN, los resultados no han sido sueriores a los conseguidos por SVM. Los motivos que hacen que las DBN no funcionen para este tipo de señal pueden ser varios. Entre ellos el más destacable es el escaso número de muestras de cada una de las clases, la base de datos del MNIST por ejemplo cuenta con 60.000 dígitos, mientras que en EEG apenas contábamos con 500. Otros factores como la alta variabilidad de las señales de EEG, la normalización, la diferente distribución de probabilidad de los datos del MNIST y de EEG o incluso la necesidad de modelos más complejos de DBN que tengan en cuenta la evolución temporal de los datos, pueden ser otros factores a tener en cuenta.

En conclusión y a pesar de no obtener los resultados esperados con EEG, se presenta un trabajo pionero en la investigación de las DBN, ofreciendo simultáneamente un tutorial de las DBN y una completa plataforma de pruebas, que puede ser utilizado en el futuro para la creación de nuevos clasificadores sobre distintos tipos de datos, así como ampliarse fácilmente para incorporar las nuevas tendencias que pueden ir surgiendo.

Para terminar, este proyecto no debería ser un punto y aparte, quedan muchas cosas por hacer y que investigar, ya no sólo en la aplicación de las DBN a otros tipos de tareas de clasificación, que sería lo más sencillo, ni siquiera en continuar la investigación de las DBN con las tendencias más actuales. Puede continuarse el estudio de las DBN aplicadas al campo de la clasificación de EEG con modelos más complejos como las redes de creencia profunda convolucionales (CDBN), que tienen en cuenta la dependencia temporal de las señales y que parece que han dado buenos resultados en señales de audio, modificando los algoritmos de entrenamiento de las RBM utilizando pesos rápidos (FPCD) o utilizando campos medios (MFCD). Como se puede ver, estamos hablando de una tecnología muy novedosa, que

está evolucionando muy rápido y que presenta numerosas aplicaciones en el futuro.

Sería también recomendable la grabación de nuevas señales de EEG, cosa que lleva meses de trabajo y que, por salirse fuera del ámbito del proyecto, no se ha podido realizar. Con estas nuevas señales podría continuarse el estudio de la aplicación de las DBN a las señales de EEG, porque, como se ha demostrado en numerosos estudios, en presencia de una gran cantidad de datos de entrenamiento, las DBN son capaces de superar a los mejores métodos de clasificación utilizados hasta el momento.

Bibliografía

[1] D Ackley, G Hinton, and T Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, (9), 1985.

[2] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. Boltzmann machines: Constraint satisfaction networks that learn. Technical report, Carnegie-Mellon University, Dept. of Computer Science, 1984.

[3] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[4] Y Bengio, P Lamblin, D Popovici, and H Larochelle. Greedy layer-wise training of deep networks. In *In NIPS*, 2007.

[5] Y Bengio and LeCun. Y.: Scaling learning algorithms towards ai. In *Large-Scale Kernel Machines*, pages 321–388. MIT Press, 2007.

[6] Steven Lemm Motoaki Kawanabe Klaus-Robert Müller Benjamin Blankertz, Ryota Tomioka. Optimizing spatial filters for robust eeg single-trial analysis. *IEEE Signal Processing magazine*, 2008.

[7] Bo Chen Nando de Freitas Benjamin Marlin, Kevin Swersky. Inductive principles for restricted boltzmann machine learning. *JMLR WCP*, (9):509–516, 2010.

[8] B E Boser, I M Guyon, and V N Vapnik. A training algorithm for optimal margin classifiers. In *In Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[9] M A Carreira-Perpignan and G E Hinton. On contrastive divergence learning. *Artificial Intelligence and Statistics*, 2005.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, 1995.

[11] Andreas M¨ller Hannes Schulz and Sven Behnke. Exploiting local structure in stacked boltzmann machines. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2010.

[12] G Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, (313).

[13] G E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput*, (14), 2002.

[14] G E Hinton, P Dayan, B J Frey, and R M Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, (268):1158–1161, 1995.

[15] G E Hinton, S Osindero, and Y W Teh. A fast learning algorithm for deep belief nets. *Neural Comp*, (18), 2006.

[16] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Nat. Academ. Sciences USA, Vol 79*, pages 2554–2558, 1982.

[17] L.Montesano I.Iturrate and J.Minguez. Robot reinforcement learning using eeg-based reward signals. *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[18] Ben Marlin Kevin Swersky, Bo Chen and Nando de Freitas. A tutorial on stochastic approximation algorithms for training restricted boltzmann machines and deep belief nets. 2010.

[19] Benjamin Marlin Kevin Swersky, Bo Chen and Nando de Freitas. A tutorial on stochastic approximation algorithms for training restricted boltzmann machines and deep belief nets. *Information Theory and Applications (ITA) Workshop*, 2010.

[20] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, 2009.

[21] Radford M Neal. Connectionist learning of belief networks. *Artificial Intelligence*, (56):71–113, 1992.

[22] M Norouzi, M Ranjbar, and G Mori. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. In *In CVPR*, 2009.

[23] S Osindero and G E Hinton. Modeling image patches with a directed hierarchy of markov random fields. In *In NIPS*, 2008.

[24] R Salakhutdinov, A Mnih, and G Hinton. Restricted boltzmann machines for collaborative ltering. In *In Proc. Of the 24th international conference on Machine learning*, pages 791–798, 2007.

[25] Ruslan Salakhutdinov and Geoff Hinton. Training a deep autoencoder or a classifier on mnist digits. http://www.cs.toronto.edu/ hinton/MatlabForSciencePaper.html.

[26] Ruslan Salakhutdinov and Geoffrey E Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, (50):969–978, 2009.

[27] Terrence J. Sejnowski. Higher-order boltzmann machines. In *Neural Networks for Computing*, pages 398–403. American Institute of Physics, 1986.

[28] Terrence J. Sejnowski. Higher-order boltzmann machines. In *Neural Networks for Computing*, pages 398–403. American Institute of Physics, 1986.

[29] Paul Smolensky. Information processing in dynamical systems: foundations of harmony theory. *MIT Press Cambridge*, 1986.

[30] G Tesauro. Practical issues in temporal difference learning. In *Mach Learn 8:257–277*, 1992.

[31] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *In Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008.

[32] Wang. The mnist database of handwritten digits, 2002.

[33] L Younes. Parametric inference for imperfectly observed gibbsian fields," springer-verlag probability theory and related fields 82. *IEEE Trans*, (33):625–645, 1989.

[34] Alan Yuille. The convergence of contrastive divergences. *Advances in Neural Information Processing Systems*, 2004.

[35] Qibin Zhao and Liqing Zhang. Temporal and spatial features of single-trial eeg for brain-computer interface. *Computational Intelligence and Neuroscience*, 2007.