



Escuela  
Politécnica  
Superior

# Optimización de Redes Neuronales mediante métodos bioinspirados



Grado en Ingeniería Informática

## Trabajo Fin de Grado

Autor:

Antonio Molina García-Retamero

Tutor/es:

Daniel Ruiz Fernández



Universitat d'Alacant  
Universidad de Alicante

Noviembre 2014

# Índice general

<b>1. Introducción</b>	<b>5</b>
1.1. La propuesta de Trabajo de Fin de Grado . . . . .	5
1.1.1. Justificación y objetivos . . . . .	5
<b>2. Proceso de investigación</b>	<b>7</b>
2.1. Planteamiento del problema . . . . .	7
2.2. El estado del arte . . . . .	7
2.3. La metaplasticidad en redes biológicas . . . . .	8
2.4. Metaplasticidad en redes neuronales artificiales . . . . .	9
2.4.1. Propuesta de implementación de la AMP en las MLP . . . . .	10
2.5. Metaplasticidad Artificial en RBFN . . . . .	12
2.5.1. Definición y características de las RBFN . . . . .	12
2.5.2. Aplicación directa del estimador de probabilidad . . . . .	13
2.5.3. Problemática . . . . .	14
2.6. Ampliación de la investigación y del estado del arte . . . . .	15
2.6.1. Tonotopía y Retinotopía . . . . .	17
2.7. Propuesta de solución . . . . .	18
2.8. Implicaciones de la propuesta . . . . .	18
2.8.1. Adaptando la metaplasticidad al modelo . . . . .	19
<b>3. Implementación</b>	<b>20</b>
3.1. Metodología de desarrollo . . . . .	21
3.2. Esquema general del diseño . . . . .	22
3.3. Lenguaje y herramientas . . . . .	22
3.3.1. Python[4] . . . . .	23
3.3.2. SciPy[3] . . . . .	24
3.3.3. Scikit-Learn[7] . . . . .	24
3.4. Implementación de la RBFN . . . . .	24
3.5. Implementación del generador de datos . . . . .	25
3.5.1. Generación de datos aleatorios . . . . .	26
3.5.2. Obtención de datos reales . . . . .	27

3.6. Implementación del generador de informes . . . . .	27
<b>4. Conclusiones</b>	<b>29</b>

# Índice de figuras

2.1. Iteración en el entrenamiento de una red neuronal aplicando el mecanismo de meta-plasticidad . . . . .	10
2.2. Comparación de la evolución del aprendizaje del algoritmo ponderado (BPW) frente al tradicional (BP) . . . . .	11
2.3. Arquitectura de las RBFN . . . . .	12
2.4. Distribución de centros neuronales reactivos a distintas frecuencias en la cóclea . . . .	17
2.5. Ilustración de la representación del fenómeno de la tonotopía como suma de gaussianas	18
3.1. Diagrama de clases simplificado . . . . .	22
3.2. Esquema del generador de datos . . . . .	26
3.3. Conjunto de datos con distribución normal . . . . .	26
3.4. Ejemplo de conjunto de datos generados aleatoriamente . . . . .	27
3.5. Esquema del Generador de informes . . . . .	27

# Capítulo 1

## Introducción

### 1.1. La propuesta de Trabajo de Fin de Grado

En este documento se recoge la memoria del trabajo de fin de grado que he realizado y que pretende justificar los valores y facultades adquiridas en el estudio y superación de las competencias recogidas en el Grado en Ingeniería Informática. En estas primeras líneas trataré de exponer el trabajo en términos generales y justificar la elección del mismo y las bondades y problemas del mismo.

#### 1.1.1. Justificación y objetivos

En el proceso de elección del proyecto de fin de grado y de los objetivos a plantear para el desarrollo del mismo, he de indicar que primó mi clara vocación investigadora y es por ello que desde el principio le planteé a mi tutor del proyecto mi deseo de realizar algún tipo de investigación básica, con especial interés es el campo de la inteligencia artificial. Es por eso que este proyecto de fin de grado está generalmente enfocado al desarrollo de una investigación que pueda encontrar una aplicación práctica evidente y que cubra en la medida de lo posible todas las competencias adquiridas durante el grado. Por lo tanto, los objetivos generales que cubren este trabajo serían:

- Búsqueda de un problema relativo al campo de estudio sobre el que realizar un proceso investigador con tal de dar solución al problema desde un punto científico-técnico.
- Estudio del estado del arte del problema en cuestión.
- Planteamiento de soluciones.
- Experimentación.

La redacción de esta memoria está estructurada haciendo una división entre el trabajo puramente teórico e investigador que será la primera parte de la misma y que tratará de clarificar la metodología de investigación, el planteamiento del problema objeto de estudio y el estudio del estado del arte. En la segunda parte se tratará el diseño y la implementación del banco de pruebas sobre el que realizar el proceso experimentador y se justificará la elección de las tecnologías y metodologías utilizadas. En

la tercera parte de esta memoria se detallará la experimentación realizada, se recogerán los resultados y se detallarán las conclusiones del trabajo en su conjunto.

## Capítulo 2

# Proceso de investigación

### 2.1. Planteamiento del problema

Una vez definido el trabajo fin de grado y planteados los objetivos de este, el siguiente paso ha sido determinar el problema que se va a tratar dentro del proceso investigador y así tratar de proponer una solución al problema siguiendo una metodología de investigación. Las redes neuronales artificiales han pretendido, desde su concepción, emular, de alguna manera, el proceso cognitivo que se produce en el cerebro de los mamíferos, siempre a una escala reducida, pero sirviéndose de los mismos principios fundamentales. Sin embargo, el problema del aprendizaje es tradicionalmente objeto de la estadísticas y no de la biología. Las redes neuronales artificiales aplicadas al problema de aprendizaje han sido muy estudiadas como un problema estadístico y existe una muy extensa literatura al respecto, sin embargo, no ha sido hasta años recientes que se está abordando el problema del aprendizaje máquina desde un punto de vista biológico. En los últimos años, la neurocomputación ha adquirido una muy notable importancia dentro del campo del aprendizaje máquina y propone sistemas que realmente pretenden emular a los sistemas biológicos y en los que la posibilidad de llegar a entender quizá un poco más los procesos que dotan a los sistemas biológicos de inteligencia y quizá incluso emular algunas de sus funcionalidades es un campo muy prometedor y cargado de retos ilusionantes.

Partiendo del problema de tratar de emular mecanismos biológicos en los modelos de aprendizaje máquina, mi tutor, Daniel Ruiz, me mostró el trabajo de Diego Andina[2] sobre la aplicación de la metaplasticidad neuronal al perceptrón multicapa (MLP) y me propuso aplicarlo a otro tipo de redes como las RBFN.

### 2.2. El estado del arte

Dado que partimos de un trabajo en curso, que es la implementación de la metaplasticidad artificial en redes neuronales artificiales, el estudio del estado del arte ha sido un trabajo aurduo que ha requerido de un estudio bastante en profundidad de las diferentes disciplinas bajo las que se contempla el problema que hemos definido y que cubren desde la neurobiología a la estadística y el aprendizaje

máquina. Es por esto que me serviré de esta sección primero para demostrar de alguna forma el trabajo realizado en la documentación y adquisición de conocimientos necesarios para comprender el problema y plantear una solución así como para tratar de introducir al lector, de la forma más amena posible, en los conceptos y terminología en que se definen la problemática y la solución propuesta al problema.

El trabajo se inicia con el estudio de lo relativo a la metaplasticidad en las redes neuronales biológicas. Además, he dedicado tiempo a estudiar y entender los modelos estadísticos bajo los que se estudian las redes neuronales artificiales para buscar la forma de aplicar este principio, que es la metaplasticidad artificial, a otros tipos de redes neuronales artificiales más allá, y partiendo, del trabajo que ha desarrollado Diego Andina en las MLP[2].

Una vez adquiridos los conocimientos necesarios para comprender el problema, se realizará un estudio del estado del arte de las diferentes técnicas de entrenamiento de las RBFN con tal de encontrar la forma de aplicar el concepto de metaplasticidad que se da de forma natural en las redes neuronales biológicas.

En las siguientes subsecciones se definirán primero la metaplasticidad en términos biológicos y se expondrá la propuesta de Diego Andina para poder especificar el problema y, partiendo de ahí, plantear la solución propuesta.

En este punto de la investigación se ha recurrido de nuevo a la bioinspiración tratando de encontrar una perspectiva bajo la que contemplar las RBFN y sobre la que plantear una solución al problema.

### 2.3. La metaplasticidad en redes biológicas

Con el término metaplasticidad nos referimos al cambio dependiente de la actividad que modula la plasticidad sináptica en los sistemas neuronales. Estos cambios en la plasticidad pueden ser del tipo LTP<sup>1</sup> y LTD<sup>2</sup>. Abraham and Bear lo definió de forma sencilla como: *plasticity of synaptic plasticity*, es decir, la plasticidad de la plasticidad sináptica. A diferencia de los mecanismos convencionales de neuromodulación de la plasticidad sináptica en los que son neurotransmisores y hormonas los que inducen cierto grado de LTP y LTD, la metaplasticidad se refiere al cambio que es provocado en un momento dado por lo que comunmente denominamos *primar* la actividad. Es decir, la metaplasticidad se refiere a la inducción en los cambios de los LTP y LTD derivados de la actividad. Reforzando aquellas conexiones sinápticas relativas a las *experiencias* más comunes y deprimiendo aquellas relativas a *experiencias* menos comunes.

Funcionalmente, la metaplasticidad dota a las conexiones sinápticas de la capacidad de integrar señales relativas a la plasticidad a través del tiempo. Además, la metaplasticidad tiene un importante papel como regulador de los umbrales de los LTP y LTD para mantener las conexiones sinápticas en un rango funcional dinámico que evite que estas conexiones se vuelvan demasiado fuertes o demasia-

---

<sup>1</sup>**Long-Term Potentiation** es la potenciación de la eficiencia provocada por los eventos más reciente en la señal transmitida entre dos neuronas que se estimulan sincronamente

<sup>2</sup>**Long-Term Depression** es la reducción de la eficiencia provocada por los evento menos reciente en la señal transmitida entre dos neuronas que se estimulan sincronamente



do débiles como consecuencia de los mecanismos de plasticidad sináptica convencionales basados en neurotransmisores y hormonas.

## 2.4. Metaplasticidad en redes neuronales artificiales

El doctor Diego Andina realizó una aproximación a la integración de los mecanismos de metaplasticidad[2], donde se propone un mecanismo de metaplasticidad artificial (AMP<sup>3</sup>) aplicado a las MLP<sup>4</sup>. En las siguientes líneas trataré de explicar la interpretación que Diego Andina propone sobre la metaplasticidad en el cálculo del error y de la modificación del algoritmo de *backpropagation* propone para la implementación de la AMP en los MLP.

El problema del entrenamiento en cualquier problema de aprendizaje es el de minimizar una función de error ajustando los parámetros del modelo en base a un conjunto de vectores de entrada  $x = (x_1, x_2, \dots, x_n), (x \in \mathbb{R}^n)$  que conforman el *training set*. En base al criterio de error  $E(x)$  que elijamos, el error esperado del sistema será por tanto:

$$E_M = \int_{\mathbb{R}^n} E(x) f(x) dx = \int_{\mathbb{R}^n} e(x) dx \quad (2.1)$$

Sobre lo que podemos hacer la siguiente transformación:

$$E_M = \int_{\mathbb{R}^n} \frac{e(x)}{f_X^*(x)} f_X^*(x) dx = \epsilon^* \left\{ \frac{e(x)}{f_X^*(x)} \right\} \quad (2.2)$$

y calcular el error del modelo añadiendo el estimador  $f_X^*(x)$ . El error del sistema será, por tanto,

$$\hat{E}_M = \frac{1}{M} \sum_{k=1}^M \frac{e(x_k^*)}{f_X^*(x_k^*)} \quad (2.3)$$

donde  $x_k^*, k = 1, 2, \dots, M$  son muestras independientes cuya *pdf* es  $f_X^*(x)$ . Como se observa,  $f_X^*(x)$  es la distribución de probabilidad de la función que tratamos de generalizar con nuestro modelo a través de nuestras muestras de entrenamiento. Por lo tanto, idealmente,

$$(f_X^*(x))_{opt} = \frac{1}{E_M} e(x) \quad (2.4)$$

Lo que se pretende conseguir a través de la introducción de la función subóptima  $f_X^*(x)$  de las ecuaciones 2.2 y 2.3 es incluir el factor de probabilidad a priori de la muestra con tal de provocar un efecto de LTP o LTD en los pesos sinápticos consiguiendo así introducir un mecanismo de metaplasticidad en la red neuronal artificial. Como se expresa en la ecuación 2.4, la función  $f_X^*(x)$  óptima sería la *fdp* (funcion de distribucion de probabilidad) de la señal de entrada de nuestro modelo ( $f_X(x)$ ), lo que es desconocido por nosotros, con lo que, en la práctica, se elige una  $f_X^*(x)$  arbitraria tal que  $f_X^*(x) \simeq f_X(x)$ .

La figura 2.1 ilustra como se incorpora el mecanismo de metaplasticidad como un factor de corrección sobre los pesos. Como se observa en la figura, esta corrección del error se hace en base a la salida esperada interpretando esta como  $\hat{y}_L = P(H_L/x)$ .

---

<sup>3</sup>Artificial Metaplasticity

<sup>4</sup>Multilayer Perceptron

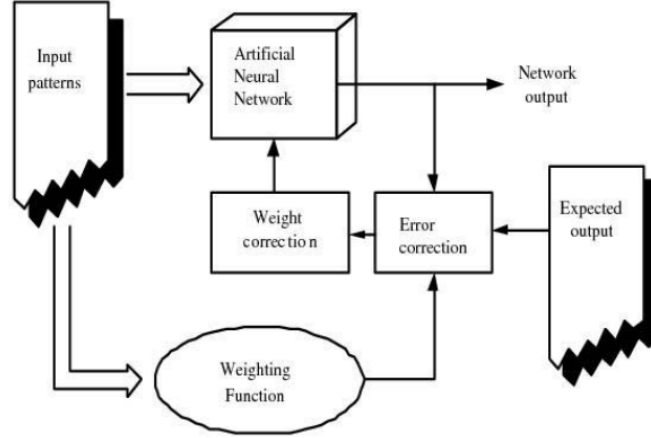


Figura 2.1: Iteración en el entrenamiento de una red neuronal aplicando el mecanismo de metaplasticidad

### 2.4.1. Propuesta de implementación de la AMP en las MLP

En base a esta interpretación que Diego Andina hace de la metaplasticidad en el mismo artículo [2] se propone la siguiente modificación sobre el algoritmo de *backpropagation* para incluir la función subóptima  $f_X(x)$  a los MLP.

$$\frac{\partial \varepsilon(W)}{\partial w_i^{(S)}} = \frac{\partial}{\partial w_i^{(S)}} \left( \frac{1}{2} \frac{(y - \hat{y}^{(S)})^2}{f_X^*(x)} \right) = \frac{1}{f_X^*(x)} \frac{\partial \varepsilon(W)}{\partial w_i^{(S)}} \quad (2.5)$$

$$\delta_j^{(S)} = (y_j - \hat{y}_j^{(S)}) \cdot \frac{f'^{(S)}}{f_X^*(x)} \quad (2.6)$$

siendo  $s$  el contador de capas,  $s = 1, 2, \dots, S$ ,  $j$  e  $i$  son el contador de nodos y entradas y  $\delta_j^{(S)}$  el error de un nodo  $j$  para la capa  $s$ . Por tanto, la ecuación 2.6 define el cálculo del error para los nodos de la última capa. Para el resto de capas, el error se propaga siguiendo la forma convencional del *backpropagation*. En definitiva, lo que se propone es incluir, para cada iteración en el entrenamiento, el siguiente factor:

$$w^*(x) = \frac{1}{f_X^*(x)} \quad (2.7)$$

con lo que se consigue acelerar el entrenamiento dando más importancia a los casos menos frecuentes. Esto además es acorde con la teoría de la información en tanto que *los sucesos menos frecuentes contienen más información*.

De todo lo comentado anteriormente, queda patente que una de las claves en la implementación de la metaplasticidad a las redes neuronales es la elección acertada de la función subóptima  $f_X^*(x)$ . En el caso concreto de las MLP, para clasificar  $L$  clases  $H_l$  para  $l = 0, 1, \dots, L - 1$ , y basándonos en el teorema de Bayes, obtenemos

$$\hat{y}_l \simeq P(H_l/x) = \frac{f_X(x/H_l) \cdot P(H_l)}{f_X(x)} \quad (2.8)$$

si asumimos la salida de nuestra red  $\hat{y}_l$  como  $P(H_l/x)$ , la propia salida de la red neuronal generaliza  $f_X(x)$  con lo que esta puede ser utilizada como función subóptima.

De esta forma, Diego Andina obtiene los resultados de la figura 2.2 y que representa la media del error para redes con la modificación propuesta (BPW) para un número creciente de iteraciones en contraposición con la media del error en redes entrenando con un algoritmo de *backpropagation* tradicional (BP). En este caso, la función de distribución de probabilidad, se ha asumido la siguiente función gaussiana

$$f_X^*(x) = \frac{A}{\sqrt{(2\pi)^n} \cdot e^{\frac{B}{8} \sum_{i=1}^n x_i^2}} \quad (2.9)$$

donde  $A$  y  $B$  son los parámetros de la gaussiana que en este caso representan los parámetros de la metaplasticidad. En el artículo[2] se muestra el siguiente ejemplo de como puede mejorar esta propuesta para una entrada que sigue una distribución de probabilidad como la descrita en la ecuación 2.9.

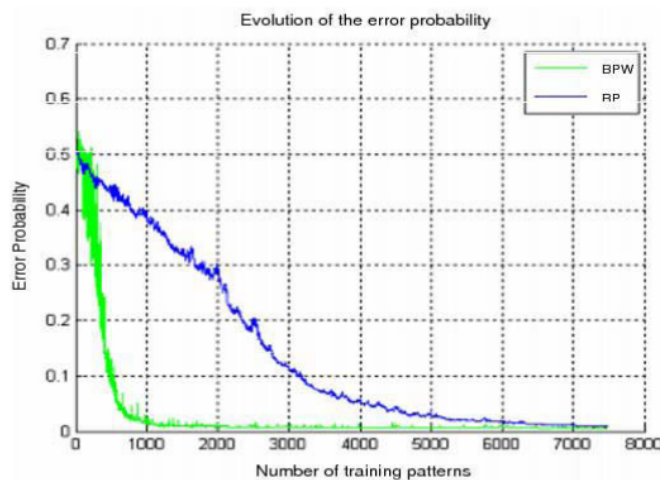


Figura 2.2: Comparación de la evolución del aprendizaje del algoritmo ponderado (BPW) frente al tradicional (BP)

### Implementación propia

Sobre esta propuesta de modificación del algoritmo de *backpropagation* realicé mi propia implementación con tal de poder hacer mi propia experimentación. Esta implementación la basé en el desarrollo de una MLP que realicé para la asignatura de *Razonamiento Automático* en C++. El objetivo de esta implementación era tan solo el de poder experimentar de primera mano, utilizando diferentes funciones subóptimas y diferentes datos de entrenamiento. Así, en la siguiente subsección expondré las conclusiones derivadas del estudio y la experimentación de la técnica propuesta por Diego Andina. Esta implementación está publicada en el siguiente repositorio de GitHub<sup>5</sup>.

### Conclusiones del estudio

Una vez asimilados los conceptos y habiendo realizado una implementación y habiendo experimentado primero sobre el conjunto de datos sobre los que se publica el artículo y después con otros

<sup>5</sup><https://github.com/aydevosotros/AMMLP>

conjuntos de datos públicos y típicos en los problemas de aprendizaje máquina y habiendo experimentado con diferentes funciones subóptimas, he alcanzado las siguientes conclusiones

- Encontrando una función subóptima que se aproxime a la función de distribución de probabilidad de los datos que componen la señal de entrada podemos obtener una gran mejora emulando los procesos LTP y LTD de los sistemas biológicos.
- A priori,  $f_X(x)$  es desconocida en los problemas de aprendizaje máquina, con lo que escoger una  $f_X^*(x)$  subóptima apropiada es un problema muy considerable.
- La posibilidad de aproximar  $f_X(x)$  por la salida de la red neuronal  $\hat{y}_l$  cuando esta está lo suficientemente entrenada como para proveer una aproximación lo suficientemente buena puede ser muy interesante.

## 2.5. Metaplasticidad Artificial en RBFN

En esta sección trataré de expresar la problemática observada en el proceso de encontrar algún mecanismo de aplicar el concepto de metaplasticidad artificial aplicado a las redes neuronales con función de activación de base radial (RBFN). Para ello, haré en primer lugar una breve introducción a las RBFN tratando de introducir al lector en las principales características de este tipo de redes para después exponer una primera solución y la problemática encontrada.

### 2.5.1. Definición y características de las RBFN

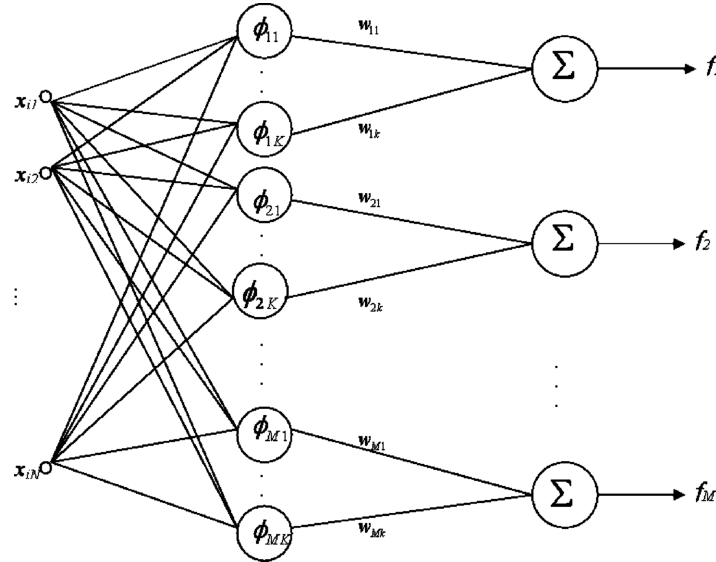


Figura 2.3: Arquitectura de las RBFN

En las siguientes líneas trataré de introducir muy brevemente en definir las RBFN y las metodologías de entrenamiento tradicionales de éstas con tal de facilitar al lector la comprensión de esta sección. Procuraré ser lo más didáctico posible y no entraré en detalle pues no considero que sea

objeto de esta memoria. Sin embargo si me parece adecuado hacer una descripción básica de estas. Me gustaría indicar, además, que para el estudio de las RBFN, fundamentalmente, he estudiado el libro de *Simon Haykin*[5].

Una posible definición de RBFN es considerarla un caso concreto de red neuronal con una arquitectura definida en la que hay una sola capa oculta y cuyos nodos de esta capa tienen como función de activación una función de base radial que tradicionalmente es una función gaussiana. En la figura 2.5.1 se representa esta arquitectura.

Los parámetros del modelo son, por tanto, el valor de los pesos sinápticos  $w_{ij}$  y los parámetros de las funciones gaussianas  $\sigma_i$  y  $\mu_i$ . Por tanto, tradicionalmente se aproxima el entrenamiento de estos parámetros en dos fases:

- **Primera fase:** Se seleccionan los parámetros de las gaussianas. Para ello, lo más convencional es servirse de alguna de estas dos técnicas:
  - Se seleccionan puntos aleatorios del conjunto de datos de entrenamiento
  - Se realiza alguna técnica de *clustering* que infiera estos centroides del conjunto de datos de entrenamiento
- **Segunda fase:** Se procede al entrenamiento de los pesos sinápticos. También existen dos aproximaciones a este problema:
  - Sirviéndose del álgebra lineal para resolver la ecuación:  $W = GY$ . Donde  $W$  es el vector de pesos,  $G$  es la matriz de activación para la salida de la capa oculta para cada centroide y cada muestra del conjunto de datos e  $Y$  es el vector con la salida esperada para cada muestra del conjunto de datos.
  - Una segunda aproximación es el de utilizar métodos de minimización como el descenso por gradiente para minimizar el error variando los pesos en función a la derivada parcial de estos con respecto al error.

### 2.5.2. Aplicación directa del estimador de probabilidad

Una primera implementación de la metaplasticidad en las RBFN podría ser la de incluir la estimación de probabilidad subóptima en la fase de entrenamiento de pesos con tal de ponderarlos de forma similar a, como he explicado en líneas anteriores, Diego Andina propuso para las AMMLP[2].

De la misma forma que explicamos anteriormente,

$$E_M = \int_{\mathbb{R}^n} e(x) dx \quad (2.10)$$

hacemos igualmente la transformación con tal de incluir el estimador  $f_X^*(x)$

$$E_M = \int_{\mathbb{R}^n} \frac{e(x)}{f_X^*(x)} f_X^*(x) dx = \varepsilon^* \left\{ \frac{e(x)}{f_X^*(x)} \right\} \quad (2.11)$$

para obtener de nuevo el error del sistema en función del criterio de error que elijamos  $e(x_k^*)$  y la función de distribución de probabilidad  $f_X^*(x)$

$$\hat{E}_M = \frac{1}{M} \sum_{k=1}^M \frac{e(x_k^*)}{f_X^*(x_k^*)} \quad (2.12)$$

### Propuesta de implementación

Para aplicar este nuevo cálculo del error a una RBFN, tan solo hemos de aplicar el criterio de error a la propia red. Si consideramos el criterio de error más típico que es el del cuadrado de la diferencia e incluyéndolo en 2.12 obtenemos:

$$\hat{E}_M = \frac{1}{M} \sum_{k=1}^N \frac{\overbrace{\left( y_k - \sum_{j=1}^M w_j G(\|x_k^* - t_j\|_{C_j}) \right)^2}^{\text{Salida de la RBFN}}}{f_X^*(x_k^*)} \quad (2.13)$$

que sustituiremos la derivada parcial respecto  $w_i$

$$\frac{\partial \varepsilon(n)}{\partial w_i(n)} = \sum_{j=1}^N \hat{E}_M(n) G(\|x_j - t_i\|_{C_i}) \quad (2.14)$$

Por tanto, y en términos del algoritmo general, el entrenamiento de nuestra RBFN solo varía con respecto a las técnicas de entrenamientos típicas en que utilizaremos la función de coste será la definida en 2.13. Por tanto la actualización de pesos sería:

$$w_i(n+1) = w_i(n) - \eta \frac{\partial \varepsilon(n)}{\partial w_i(n)} \quad (2.15)$$

### 2.5.3. Problemática

Como ya se ha expuesto anteriormente y en esencia, la propuesta de Diego Andina se basa en la emulación de los procesos LTP y LTD añadiendo un factor que pondera la derivada parcial en el descenso por gradiente de la muestra con respecto al peso de la conexión sináptica que representa cuan *rara*, en términos de probabilidad, es la muestra respecto a la clase en que se clasifica a posteriori. Esta probabilidad a posteriori se estima por la propia salida de la red neuronal en ese estado del entrenamiento.

En este punto, queda patente que existe numerosos problemas a la hora de aplicar esta técnica a otro tipo de redes neuronales, en particular en las RBFN. En la figura 2.5.1, se representa la arquitectura típica de una RBFN.

Un problema fundamental se debe a la salida de la última capa de sendas redes neuronales. Para implementar un mecanismo de metaplasticidad similar al descrito en las AMMLP<sup>6</sup>[2], se requiere de una estimación de la probabilidad de pertenencia a la clase (cuán *rara* es la muestra) y en las AMMLP la estimamos como la salida de la propia red neuronal. Ya que la neurona de la última capa es una función logística<sup>7</sup>, asumimos ésta como la probabilidad a posteriori de pertenencia a la clase. Por

---

<sup>6</sup>AMMLP: Artificial Metaplasticity MultiLayer Perceptron

<sup>7</sup>Una función logística es una función continua cuya salida  $\in [0,1]$ . Un ejemplo de función logística típica es

$f_{log}(z) = \frac{1}{1+e^{-z}}$

su parte, la salida de una RBFN para el problema de clasificación se calcula como  $\text{sign}(\sum w\phi(x))$ . Queda patente que la salida de la RBFN no nos sirve para estimar una probabilidad a posteriori de la muestra.

Sin embargo, la propia naturaleza de las RBFN parece resultar idónea a su vez para experimentar con estas técnicas. Este *parecer* es una percepción completamente intuitiva, pero existen ciertas características que trataré de explicar a continuación y que pueden hacer de estas redes una estructura idónea para representar las redes neuronales biológicas la forma en que estas representan la información y que me inspiraron a plantearme el problema en multitud de formas tratando de encontrar la forma de encajar estas características con la naturaleza del aprendizaje y de la metaplasticidad. Estas características son, en líneas generales:

- La capa oculta característica de las RBFN está compuesta por nodos cuya función de activación es típicamente una gaussiana en la forma  $\phi(x)$ . Estas gaussianas tienen dos parámetros básicos y que son fundamentalmente dependientes de la naturaleza de la señal de entrada y que son un centroide y la varianza.
- La capa oculta realiza una transformación no lineal en un espacio basada en la distancia al centroide. Esta característica sirve de nexo para el estudio de las RBFN dentro del conjunto de los *Kernel Methods*.
- La matriz de covarianza determina el campo receptivo de las funciones de activación de base radial gaussianas.

Expuestos los problemas y las características más relevantes en este estudio, continué la investigación buscando técnicas a través de las cuales poder extraer información que pudiera estimar la función de distribución de probabilidad de la señal de entrada. Tal y como hemos visto anteriormente, existen típicamente dos fases de entrenamiento y en la primera de estas se trata de encontrar la posición de los centroides en el espacio de características para las gaussianas de la capa oculta y, en ocasiones, estimar una covarianza para estas gaussianas. Esta primera fase es, por tanto y básicamente, una búsqueda de información sobre la señal de entrada donde tratar de estimar de alguna forma  $f_X(x)$  y utilizarlo en la segunda fase de entrenamiento para ponderar los pesos con la función  $w^*$  en un descenso por gradiente en busca de minimizar la función de coste de nuestra red.

## 2.6. Ampliación de la investigación y del estado del arte

En este punto y en base a la intuición, como ya se ha indicado, de que las propia naturaleza de las RBFN y de la propiedades intrínsecas a las funciones de base radial, así como su característico entrenamiento en dos fases podían ser candidatas ideales para abstraer mecanismos típicos en las redes biológicas como el de la metaplasticidad que es el que nos ocupa en este trabajo.

Ya hemos comentado los problemas que representaba la aplicación directa de la ponderación de pesos propuesta por [2] y la imposibilidad de estimar la  $f_{dp}$  de la señal de entrada siguiendo mecanismos similares a los estudiados para las AMMLP. Sin embargo, desde el principio de este trabajo

tuve la convicción de que, bajo la interpretación adecuada, esta debería poder ser inferida de alguna forma.

Este proceso, que se extendió durante meses en el tiempo y que tenía por objeto encontrar algún mecanismo para estimar esta  $f_{dp}$  en esta arquitectura tan particular de red, me llevó a tratar de experimentar con diferentes ideas, siempre en busca de inferir una estimación de  $f_X(x)$  aprovechando el proceso de estimación de los centroides de las gaussianas.

Existen diferentes aproximaciones al problema, pero tratando de ser consecuente con la investigación en la que se pretende recurrir a la bioinspiración, todo este trabajo creativo de proponer una nueva solución al problema de la metaplasticidad en las RBFN culminó en la propuesta que describiré en las siguientes líneas.

En líneas generales, se propone una nueva interpretación de las RBFN bajo la perspectiva de que, en lugar de que los nodos cuya función de activación es una RBF sean contemplados en términos de neuronas, se propone el contemplarlos como centros neuronales. En las redes neuronales biológicas, se ha observado que en las cócleas de los mamíferos, las neuronas se organizan siguiendo una estructura topológica que responde a las características propias de las señales que procesa el oído, esto es al sonido. Más adelante se trata un poco más en detalle los fenómenos de la tonotopía y la retinotopía.

Esta disposición topológica puede realmente ser representada por una RBFN si consideramos que cada función de base radial abstrae esta topología definida por el centroide de la gaussiana y la varianza para definir la topología en base a las características del espacio de nuestra señal de entrada. Esta aproximación nos invita a intentar representar esta estructura topológica sobre el espacio de características con las RBF. Además, y en lo que se refiere a la aplicación del mecanismo de metaplasticidad, esta nueva interpretación nos invita a aproximar la  $f_X(x)$  en base a la topología que define la propia red neuronal ya que las propias gaussianas definen una  $f_{dp}$  que nos puede dar una idea de la probabilidad a priori de una muestra en base a la topología que define la red.

Además, las RBFN son estudiadas también dentro del conjunto de modelos que se recogen bajo el nombre de *Kernel Methods* y que tienen como características principales que se sirven de *Kernels* para realizar transformaciones no lineales para buscar la separabilidad de los datos en espacio de muchas dimensiones y que estos *kernels* realizan esta transformación siempre en base a una distancia.

Sin intención de entrar demasiado en detalle, trataré de introducir al lector en el estado del arte de estas nuevos campos de investigación en los que sustento la propuesta de solución haciendo una pequeña introducción a la tonotopía y la retinotopía en redes neuronales biológicas. Por último y para acabar esta primera parte de trabajo de investigación, trataré de exponer de forma más concreta esta propuesta.

De forma simplificada, trataré de conducir al lector a través de la línea de pensamiento que me llevó a plantear una nueva propuesta. Ya se han explicado, también de forma muy simplificada, las estrategias típicas de entrenamiento de este tipo de redes. De entre estas, tenemos la opción de escoger los centroides de las gaussianas aleatoriamente de entre los puntos de nuestro conjunto de datos.



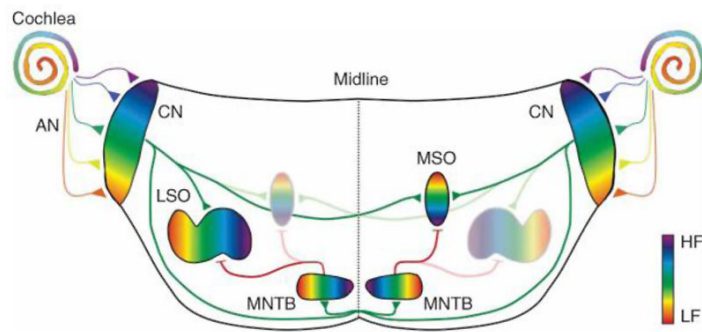


Figura 2.4: Distribución de centros neuronales reactivos a distintas frecuencias en la cóclea

### 2.6.1. Tonotopía y Retinotopía

La tonotopía se define como la organización topológica de las neuronas en el cortex auditivo bajo la que neuronas sensibles a diferentes frecuencias se distribuyen. En una de las primeras investigaciones al respecto, Colin Humphries publicó en *Tonotopic organization of human auditory cortex*[6] los resultados que obtuvo del estudio de este fenómeno en el cerebro humano a través de la observación de este bajo resonancia magnética funcional, como efectivamente, neuronas reactivas a estímulos con una frecuencia cercana se distribuyen formando una mapa de frecuencias como se muestra en la figura 2.6.1. En la misma línea, se hace un análisis detallado de este fenómeno en el siguiente artículo[1]. Considero que queda fuera del alcance de esta memoria el de definir detalladamente el estado del arte de estos estudios y pretendo no abrumar al lector con tal cantidad de información colateral, sin embargo, sí me gustaría hacer constar que en el estudio del fenómeno se han observado detalles que animan al esfuerzo en intentar emular estos mecanismos basándose en la propuesta que planteo y que se basa en representar la topología de estos conjuntos de neuronas que responden a frecuencias de sonido concretas con estructuras matemáticas como los *kernels* que realizan una transformación no lineal en base a una distancia (en este caso concreto, esas distancias son las frecuencias) tal y como se observa que hace el cerebro al distribuirse topológicamente.

De la misma forma que ocurre en el córtex auditivo, se ha observado un fenómeno similar relacionado con la percepción de la luz y el procesamiento de la imagen en el córtex visual y que se estudia bajo la disciplina de la retinotopía. En este campo, N. Mayer ha llegado a proponer en un artículo titulado *Retinotopy and spatial phase in topographic maps*[8] un algoritmo que se sirve del mapeo en subespacios para producir mapas topográficos similares a la contra-parte biológica.

No es objeto de este trabajo entrar en detalles al respecto de los detalles concretos de este fenómeno y es muy importante la investigación que actualmente se está desarrollando al respecto. Así mismo, esta aproximación desde el aprendizaje máquina tal vez pueda arrojar luz a como funcionan ciertos procesos del cerebro que aún no se explican en el estado del arte de ninguna disciplina científica.

## 2.7. Propuesta de solución

En base a los trabajos previos, la idea de servirnos de la naturaleza de los datos de nuestro entrenamiento para hacernos una idea a priori de cuan significativa es una muestra y actuar ponderando esa conexión sináptica concreta parece encajar de forma especialmente buena en el concepto de RBFN debido a la naturaleza de las Funciones de Base Radial.

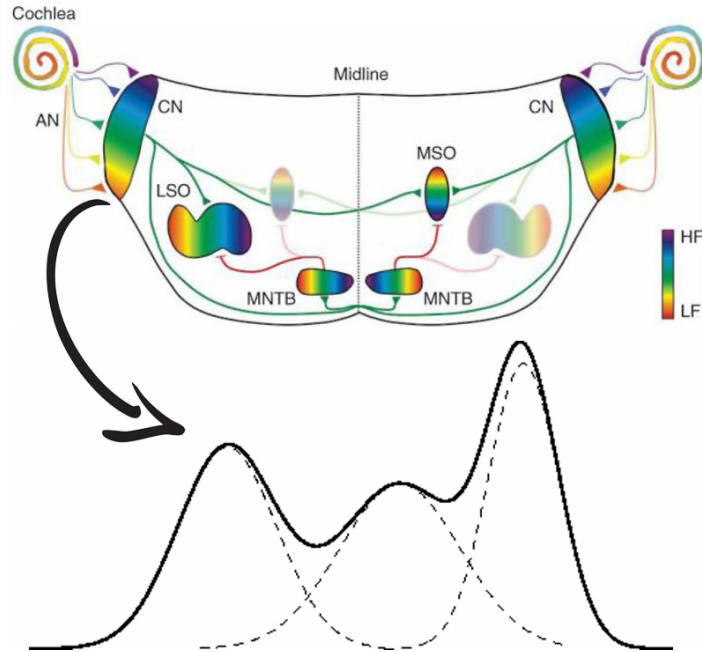


Figura 2.5: Ilustración de la representación del fenómeno de la tonotopía como suma de gaussianas

Si posicionamos los centroides de nuestras funciones de base radial en unas frecuencias concretas tal y como se ha demostrado que ocurre en el cerebro de los mamíferos y como se ilustra en la figura 2.5, en el modelo propuesto, la capa oculta de las RBFN puede representar la mezcla de gaussianas que mapean el espacio de características que son las muestras de la señal de entrada.

En resumen, lo que se propone es un modelo bioinspirado en los fenómenos de la tonotopía y la retinotopía que se basa en los siguientes principios:

- La distribución topológica de las neuronas que se ha demostrado en las redes neuronales biológicas que transforman las señales que perciben nuestros sentidos, puede ser representada como una suma de funciones gaussianas.
- Cada una de estas funciones gaussianas es un kernel de nuestro modelo que mapea la señal de entrada a un espacio de alta dimensionalidad donde los datos serán separables por un hiperplano con lo que el modelo se puede enmarcar conceptualmente entre los *kernel methods*.

## 2.8. Implicaciones de la propuesta

Algunas de las implicaciones que he analizado de la propuesta y que recoge esta memoria son las siguientes:

- La retinotopía y tonotopía vienen a demostrar que la topología del cerebro ha sido diseñada por la evolución para conservar en si misma información acerca de como procesar la información. Esto es, existe un conocimiento a priori en los sistemas neuronales biológicos como es en este caso la distribución de las neuronas que procesan la señal auditiva y se organizan topológicamente en función a las características de la señal procesada como es la frecuencia en el caso de la tonotopía. Por lo tanto podemos diseñar modelos que definan una topología basada en el conocimiento y la experiencia que se tiene de la señal que se va a procesar.
- Así como el cerebro de los mamíferos, la estructura topológica posibilita el aprendizaje de funciones de bajo nivel transformando el espacio de características a otros espacios. Como hacemos tradicionalmente con los *kernel methods*.

### 2.8.1. Adaptando la metaplasticidad al modelo

En conclusión, bajo esta nueva perspectiva que se ha tratado de dilucidar en las secciones anteriores, y asumiendo que las funciones gaussianas de la capa oculta representan la distribución topológica dependiente de alguna característica intrínseca a la señal que se ha de procesar de forma mimética a lo que hacen los conjuntos de neuronas, que se distribuyen según se ha descrito en los fenómenos de tonotopía y retinotopía, la función de distribución de probabilidad de la señal de entrada podría ser aproximada por la suma de las gaussianas tal y como se ha representado en la figura 2.5.

Con esto, y partiendo de la aplicación directa del estimador de probabilidad descrito en la sección 2.5.2, el entrenamiento de los pesos sinápticos de una RBFN con el mecanismo de metaplasticidad propuesto sería:

$$w_i(n+1) = w_i(n) - \eta \frac{\partial \varepsilon(n)}{\partial w_i(n)} \quad (2.16)$$

cuya derivada del error respecto a los pesos sería:

$$\frac{\partial \varepsilon(n)}{\partial w_i(n)} = \sum_{j=1}^N \hat{E}_M(n) G(\|x_j - t_i\|_{C_i}) \quad (2.17)$$

donde

$$\hat{E}_M = \frac{1}{M} \sum_{k=1}^M \frac{\overbrace{\left( y_k - \sum_{j=1}^M w_j G(\|x_k^* - t_j\|_{C_j}) \right)^2}^{\text{Salida de la RBFN}}}{f_X^*(x_k^*)} \quad (2.18)$$

y en dónde definimos el estimador  $f_X^*(x_k^*)$  como:

$$f_X^*(x_k^*) = \sum_{i=1}^K \phi_i(\mu_i, \sigma_i) \quad (2.19)$$

## Capítulo 3

# Implementación

Con tal de dar soporte al planteamiento teórico que ha sido expuesto en el capítulo anterior, he desarrollado un *benchmarks* sobre el que realizar la experimentación que sustente la validez (o pruebe la falta de esta) de los algoritmos propuestos. Pese a tener un objeto muy concreto, he tratado de realizar un diseño tan generalista como me fuera posible, tratando de definir y delegar las responsabilidades entre clases y persiguiendo siempre un bajo acoplamiento y una alta cohesión entre estas. En líneas generales, el software desarrollado ha de cubrir los siguientes objetivos:

- Implementar una **RBFN** configurable tanto en los parámetros propios de la RBFNN (dimensionalidad, parámetros de las gaussianas, etc...) como en los algoritmos de entrenamiento.
- Implementar un mecanismo de **generación/obtención** de los conjuntos de datos:
  - Ha de ofrecer la posibilidad de generar datos aleatorios en base a una *fdp* ( $f_X(x)$ ) conocida con tal de facilitarnos la experimentación en nuestro problema en concreto en el que, como se ha discutido ampliamente en secciones anteriores, esta función tiene un papel fundamental.
  - Además, ha de ofrecer diferentes conjuntos de datos reales con tal de posibilitar la experimentación y comparación de estos algoritmos con el estado del arte del problema.
- Implementar un mecanismo que recopile los resultados obtenidos y **genere un informe** con toda la información necesaria para evaluar estos resultados.

Una vez especificados los objetivos, en las siguientes subsecciones describiré y justificaré las metodologías escogidas para el desarrollo del software. Además, describiré y justificaré la elección del lenguaje de programación escogido y de las herramientas en las que me he apoyado para el desarrollo. Por último, trataré de ir describiendo los diferentes paquetes que he desarrollado para cubrir cada uno de los requerimientos que se han planteado en forma de objetivos (*RBFN*, *Generador de datos*, *Generador de informes*).

El objetivo final del software a desarrollar será el de proveer de un *framework* de desarrollo rápido que fuera lo más general y flexible posible para poder realizar una experimentación lo más rápida y cómoda posible. Es decir, el objetivo último era conseguir un entorno con las siguientes características:

- **Flexible:** Ha de facilitar el proceso de realizar experimentación con diferentes algoritmos y parámetros con el menor número de cambios posible.
- **Extensible:** Ha de ser fácilmente extensible a nuevos algoritmos y que estas extensiones no tengan repercusiones notables en los experimentos ya implementados
- **Desarrollo rápido:** Ha de permitir implementar algoritmos y desarrollar experimentos con poca inversión de esfuerzo y tiempo para no desfocalizar al experimentador con un desarrollo engorroso.

De esta forma, la experimentación será un ejercicio dinámico en la que fácilmente cambiar los diferentes parámetros y algoritmos y experimentar sobre diferentes conjuntos de datos. La experimentación, en multitud de ocasiones, se trata de un ejercicio de ensayo-error guiado por un proceso creativo en donde la simplicidad de las herramientas juega un papel muy importante y por lo tanto, el diseño ha tenido como requisito constante exponer un conjunto de objetos sencillos, claros y fáciles de utilizar y que a su vez sean altamente parametrizables y extensibles en su comportamiento.

Por último, indicar que todo el código relativo a este trabajo, incluyendo el código fuente de esta memoria y de la presentación, está disponible públicamente en el siguiente repositorio de GitHub<sup>1</sup>.

### 3.1. Metodología de desarrollo

En cuanto a las metodologías de desarrollo, en fase de diseño, determiné usar diferentes metodologías para afrontar los diferentes problemas y con tal de aprovechar las ventajas que, sobre cada uno de los problemas, estas ofrecen.

Así pues, para implementar la RBFN me serví de un desarrollo basado en Test Driven Development (TDD). Debido a la complejidad de implementación a la susceptibilidad de errores de implementación en los algoritmos, me aproximé al desarrollo de la red neuronal basándome en diferentes pruebas lo suficientemente sencillas para poder precalcular cada uno de los resultados de los algoritmos y garantizar así la correcta implementación. Así mismo, el diseño de la clase vino determinado por las necesidades que surgían al diseñar las pruebas y, por lo tanto, siguiendo un desarrollo *up-to-bottom*.

Para el desarrollo del generador de datos realicé un Diseño Orientado a Objetos. Esto es, dado que los requisitos para el generador de datos, descritos anteriormente, me permitían hacer un diseño de clases bastante claro ya que los propios requisitos definen la interfaz pública de la clase generadora de datos (*generarDataSet*, *obtenerX*, *obtenerY*, *etc...*) que define el diseño de la clase y su comportamiento.

Por último, para el generador de informes partía de un resultado esperado que era el informe y una interfaz pública más o menos clara (*addGrafico*, *addSection*, *etc*). Sin embargo en tiempo de desarrollo, las pruebas fueron redefiniendo el comportamiento en base a los requisitos que iban imponiendo las pruebas rediseñando así el modelo en cada *iteración* del desarrollo.

---

<sup>1</sup><https://github.com/aydevosotros/TFG-AntonioMolina>

## 3.2. Esquema general del diseño

Ya que el objetivo fundamental del desarrollo es el de proveer de un entorno de experimentación con los requisitos que se han descrito en la introducción de este capítulo, he decidido plantear el desarrollo como una pequeña librería de clases que me ofrezca clases listas para instanciar y con unos pocos parámetros de configuración me provea de diversas configuraciones (tipo de RBFN, conjunto de datos, etc..) para un experimento. Así, con unas pocas líneas de código debería ser capaz de, obtener una RBFN con o sin metaplasticidad y con los parámetros propios de la red deseados, obtener un conjunto de datos con las características deseadas y realizar tantas pruebas como se quiera. Además la recogida de datos deberá ser asistida por una clase que permitirá la construcción de tablas y gráficos y construirá el código LaTeX requerido para la representación de estos. Por lo tanto, nuestra librería constará de 3 clases principales en las que se delegarán las 3 responsabilidades descritas:

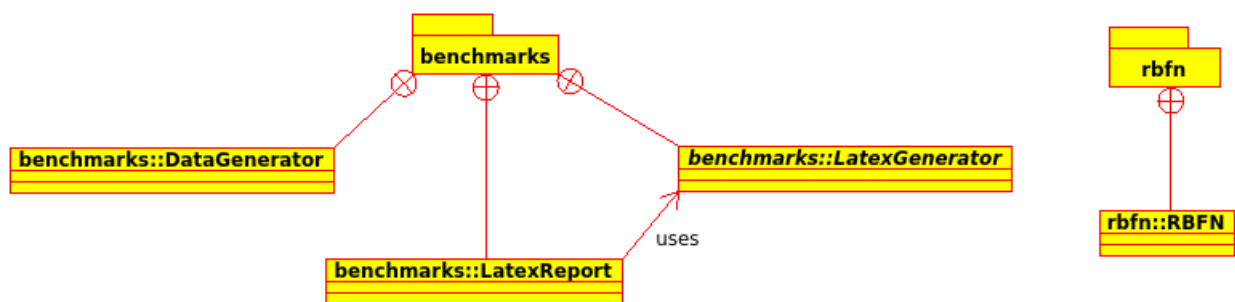


Figura 3.1: Diagrama de clases simplificado

En la figura 3.2 se muestra un diagrama de clases simplificado. En las siguientes subsecciones se detallarán cada una de las clases y se tratará de mostrar al lector las dificultades y bondades de la implementación realizada.

Como se puede observar se trata de un diseño sencillo. Me gustaría dejar constar también que la dificultad de esta implementación ha estribado más en las dificultades propias de implementar algoritmos relativamente complejos, con mucho cálculo matemático y que en ocasiones puede resultar especialmente engorroso de depurar. También me gustaría señalar que en este trabajo de implementación considero que ha sido especialmente acertada la elección de lenguajes y herramientas que describo a continuación y que, en muchos casos, la sintaxis del lenguaje y la potencia de que lo dota la paquetería utilizada me ha facilitado esta tarea de codificación de fórmulas e implementación de algoritmos.

## 3.3. Lenguaje y herramientas

Establecidos los objetivos, metodologías, requisitos y diseño general, me propongo a decidir bajo que entorno construir la librería. Una primera opción muy recurrida en los entornos investigadores es el uso de *Matlab*. Este entorno muy popular y potente ofrece un numero enorme de algoritmos categorizados, estructurados y fáciles de usar se ha ganado su popularidad en entornos de investigación muy merecidamente ya que en un solo entorno integra todo lo requerido para prácticamente cualquier

investigación en cualquier campo. Además cumple perfectamente con los requerimientos que pueda tener para el desarrollo descrito. Sin embargo, este entorno es privativo y de código cerrado. Además, los costes de su licencia son muy considerables, y en tanto me fuera posible, intento siempre utilizar herramientas abiertas.

De entre las alternativas abiertas, existen numerosas (y algunas muy buenas) aproximaciones al entorno *Matlab* como puede ser *octave* para el que existe una gran comunidad y multitud de librerías y módulos para extenderlo a muy diversas áreas. Otra aproximación a este tipo de entornos es la librería para *Python* *SciPy* que aglomera un conjunto de paquetes que abarcan gran parte de los requerimientos comunes a cualquier investigación científica. Complementando este *ecosistema* de paquetería que es *SciPy* con otro conjunto de módulos llamado *scikit-learn* y que recoge una muy completa utilitería para problemas de aprendizaje máquina y que a su vez está construida sobre *SciPy*, será la opción elegida para este proyecto. A continuación dedicaré unas pocas líneas a tratar de dar una visión global de cada uno de estos conjuntos de módulos y herramientas:

#### 3.3.1. Python[4]







*Python* es un lenguaje de programación interpretado muy popular en la comunidad *Linux* desde sus primeras versiones. *Python* ha sido utilizado en innumerables proyectos y existen en enorme ecosistema a su alrededor. alguna de las características que lo hacen idónea para el desarrollo de este proyecto son los siguientes:

- Es un lenguaje de alto nivel de sintaxis clara y concisa que lo hace ideal para tareas de prototipado rápido como es la implementación de los algoritmos objeto de esta investigación.
- Es un lenguaje multiplataforma que permite la ejecución directa en diferentes plataformas y dispositivos, requisito fundamental en cualquier equipo de investigación. Además, cualquiera puede acceder al código desarrollado y ejecutarlo con tal de corroborar los resultados directamente.
- Existe una enorme cantidad de software (y mucho de muy buena calidad) desarrollado en *Python* y que enriquecen la plataforma haciéndola idónea para muchos ámbitos. En concreto a lo que nuestro problema se refiere, he citado anteriormente las librerías que se han utilizado en este desarrollo y cuyas características describiré en las siguientes subsecciones.

#### 3.3.2. SciPy[3]



Tal y como se definen en su web, *SciPy es un ecosistema basado en Python de software abierto para matemáticas, ciencias e ingeniería*. En concreto, estos son los paquetes que componen el núcleo de *SciPy*:

- *NumPy*  : Paquete básico de arrays n-dimensionales
- *SciPy library*  : Librería fundamental para el cómputo científico
- *Matplotlib*  : 2D Plotting
- *Sympy*  : Matemáticas simbólicas

Este conjunto de librerías nos proveerán el entorno básico sobre el que trabajar con toda la paquetería que requerimos para el álgebra lineal, generación aleatoria en base a funciones de densidad de probabilidad, las herramientas para dibujar los gráficos, etc...

#### 3.3.3. Scikit-Learn[7]



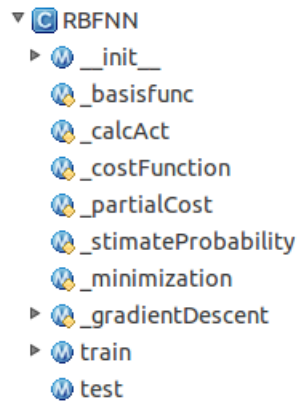
Se trata de un conjunto de herramientas con objeto de hacer simple y eficientemente tareas de *data mining* y *data analysis*. Está construido sobre *SciPy* y está licenciado bajo una licencia de código abierto. Ofrece diferentes implementaciones de algoritmos orientados a tareas de clasificación, regresión, clustering, reducción de dimensionalidad, selección de modelos y preprocesamientos.

Aunque en esta investigación se han definido nuevos modelos, me he servido de estas herramientas para el preprocesamiento, por ejemplo al probar con los datos escalados y sin escalar, o para utilizar técnicas de clustering para encontrar los centroides de las gaussianas de las RBFN.

### 3.4. Implementación de la RBFN

La clase RBFN es básicamente una clase contenedora de algoritmos y sus parámetros. La estructura se muestra en la figura tal. La interfaz pública de la clase es muy sencilla y se basa en un constructor que recibe los parámetros de la red y que permite escoger el algoritmo de entrenamiento. Un método para entrenar el modelo en base a un conjunto de datos de entrenamiento etiquetado y un tercer método de prueba que devuelve el resultado de la red para una entrada dada. Con esta estructura sencilla se





pretende que el mismo código de experimentación sea reutilizable y que con muy pocas modificaciones (los parámetros del constructor) podamos disponer de redes que implementen algoritmos o parámetros distintos.

Como se observa del diseño, todos los tipos de redes comparten la misma interfaz pública (métodos *train* y *test*) y la elección de los algoritmos se hace en el constructor, con lo que un mismo experimento se puede aplicar a diferentes algoritmos y parámetros tan solo cambiando el constructor, lo que es consecuente con los requisitos descritos. Así mismo, la mayoría de parámetros propios de las redes están definidos con unos parámetros por defecto.

También me gustaría señalar que, como se puede observar en el esquema de la clase, para la minimización del coste se ha realizado una implementación propia y sencilla del algoritmo de *descenso por gradiente* con tal de depurar fácilmente las funciones de coste y las derivadas parciales. Para pruebas de rendimiento también se proporciona la posibilidad de utilizar diferentes algoritmos de minimización que proporciona *SciPy*. Estas opciones de minimización son: *Nelder-Mead simplex algorithm*, *BFGS algorithm*, y *Newton-Conjugate-Gradient algorithm*.

Sin encontrar dificultades en la implementación y habiendo indicado las particularidades de la implementación, tan solo añadir que para aplicar el mecanismo de metaplasticidad propuesto tan solo es necesario un parámetro por parámetro en el constructor que así lo indique y en el método *\_partialCost()* se hará uso de una estimación de la probabilidad de la muestra para ponderar el peso tal y como se ha descrito.

### 3.5. Implementación del generador de datos

El generador de datos tiene como objetivos generales generar/obtener los conjuntos de datos que alimentarán nuestros modelos y hacerlos disponibles de forma sencilla. El generador de datos tiene un papel fundamental en la experimentación debido a las particularidades de la solución propuesta ya que es importante tener control sobre la distribución de los datos con los que trabajamos. Además, se delega en este la obtención de los datos de diferentes bases de datos estándar con tal de comparar los resultados con otras propuestas del estado del arte.

La siguiente figura representa el esquema general de la clase generadora de datos implementada. Los dos métodos principales de estas clases y que son los encargados de generar y de obtener los

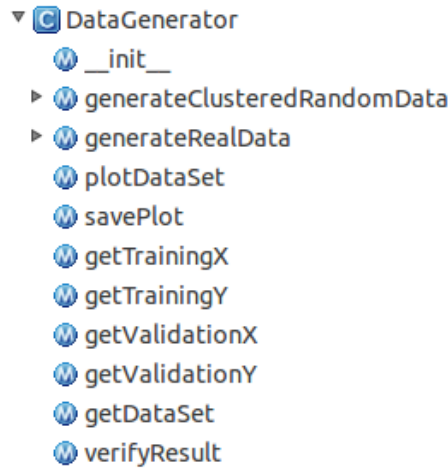


Figura 3.2: Esquema del generador de datos

conjuntos de datos serán explicados en detalle a continuación.

### 3.5.1. Generación de datos aleatorios

Tal como se ha descrito en secciones anteriores, en la implementación de la metaplasticidad propuesta por Diego Andina[2], la  $f_{dp}(f_X(x))$  es fundamental y esto hace muy importante poder tener control sobre la distribución de probabilidad de los conjuntos de datos que se generan. Dado que se ha observado históricamente que las distribuciones gaussianas son las más comunes en los datos de origen natural, he utilizado para la generación de datos aleatorios la implementación que ofrece *numpy* y que genera los datos siguiendo la siguiente función de distribución de probabilidad normal:

$$pdf(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(\mu-x)^2}{2\sigma^2}\right)}$$

El método, por tanto, construye el conjunto de datos en base a dos centroides aleatorios (que representan dos clases) etiquetando cada muestra en su clase. Dado lo compacto del código me decido a introducirlo en esta memoria para el lector interesado.

```

def generateClusteredRandomData(self, nSamples=500, dim=2, sigma=0.1, cf=30, cd=10):
    for i in xrange(2):
        mean = np.random.rand(dim)*cf + (np.random.rand(dim)*cd)
        samples = np.random.normal(size=[nSamples, dim], loc=mean, scale=sigma)
        for n, sample in enumerate(samples):
            sample = np.append(sample, (1 if i==0 else -1))
            if n==0 and i==0:
                self.data = sample
            else:
                self.data = np.vstack((self.data, sample))
    np.random.shuffle(self.data) # Shuffle data to avoid dataset to be ordered by centroids
  
```

Figura 3.3: Conjunto de datos con distribución normal

A continuación podemos ver un ejemplo del funcionamiento de la generación para 3 clases:

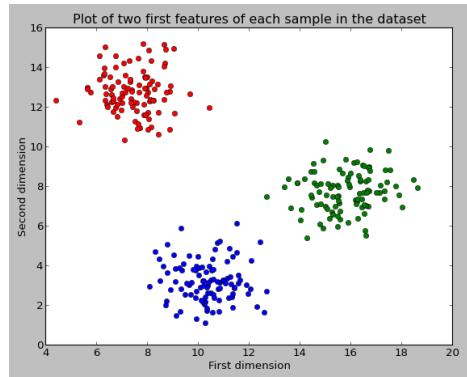


Figura 3.4: Ejemplo de conjunto de datos generados aleatoriamente

### 3.5.2. Obtención de datos reales

Además de la experimentación con datos generados y con objeto de validar las propuestas, he implementado este segundo método en el generador de datos que se sirve de dos conjuntos de datos enormemente utilizados en el entrenamiento y verificación de sistemas de aprendizaje máquina y que son:

- The Wisconsin Breast Cancer Data Set
- Vertebral Column Data Set

Una vez generados los datos, los métodos de obtención de los datos mantienen la misma interfaz independientemente del conjunto de datos con lo que sobre el mismo código de prueba, la elección del conjunto requiere tan solo la llamada de un método a otro.

## 3.6. Implementación del generador de informes

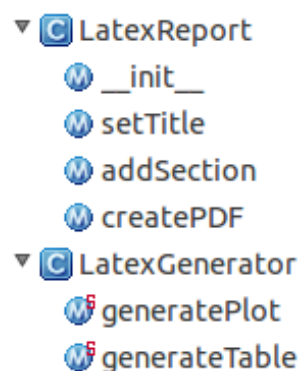


Figura 3.5: Esquema del Generador de informes

El generador de informes es un módulo compuesto de dos clases (*LatexReport* y *LatexGenerator*). Este módulo tiene la responsabilidad de generar el código *l*á*t*ex necesario para producir un informe de los resultados de la experimentación. La clase *LatexReport* permite crear un informe estableciendo un título para este y habilitando la posibilidad de añadir tantas secciones como se desee. Estas secciones

están compuestas por un texto descriptivo, una gráfica que represente los datos y una tabla que contenga estos datos. Además, será la responsable de generar el archivo PDF como resultado de guardar el código en un archivo de texto *.tex* y llamando al comando del sistema pertinente para la compilación del fuente. La clase *LatexGenerator* es un contenedor de métodos estáticos y que son los que proporcionarán el código LaTeX para los gráficos y las tablas. El resultado de esta clase fue una refactorización para una división de responsabilidades y mejorar la mantenibilidad y la legibilidad.

El objetivo de estas clases es el de proporcionar un mecanismo por el cual generar las tablas y los gráficos de los resultados de la experimentación en un informe PDF con tal de simplificar esta tarea. Al mismo tiempo, se ha pretendido ser genérico en cuanto a los datos a representar. Como resultado, se ha obtenido una implementación que pese a ser poco flexible y permitir relativamente pocas opciones, nos permite generar informes que presenten secciones sencillas compuesta de los gráficos generados con *matplotlib* y construir tablas sencillas que recopilen los resultados.

## Capítulo 4

# Conclusiones

A lo largo de estas páginas se ha pretendido exponer al lector el trabajo realizado intentando abarcar en la medida de lo posible la extensión del mismo, las dificultades y soluciones a los problemas encontrados en la andadura, así como demostrar la cobertura de los objetivos propuestos en la introducción a la memoria.

Para cumplir con estos objetivos y en líneas generales:

- Se ha tratado un problema que no es trivial y que se trataba de mejorar el entrenamiento de las RBFN basándose en métodos bioinspirados partiendo del trabajo propuesto por Diego Andina[2] sobre la metaplasticidad en las redes neuronales artificiales.
- Se ha realizado un extenso estudio que incluye el estado del arte tanto de problemas de aprendizaje máquina como de diferentes problemas relativos a las neurociencias.
- Se ha realizado un extenso estudio del problema concreto de encontrar un método para aplicar la metaplasticidad en RBFNs encontrando una solución bioinspirada, original y con potencial para desarrollar nuevas técnicas y algoritmos que emulen los procesos de las redes neuronales biológicas.
- Se ha desarrollado un conjunto de herramientas para poder realizar una experimentación ágil basada en prototipado rápido.

En esta conclusión, y haciendo una visión en retrospectiva de la propuesta, considero que se han abarcado todos los objetivos de la propuesta con éxito y solvencia a la hora de plantear una solución a los problemas descritos y que no son triviales. Así mismo, es de justicia mencionar que el desarrollo del proyecto ha sido probablemente en exceso ambicioso, el trabajo aquí expuesto ha sido el resultado de casi un año en que en multitud de ocasiones me he visto superado por las dificultades que presenta el contexto del problema. He tenido que ampliar sucesivas veces los conocimientos adquiridos durante el grado, así, he tenido que dedicar numerosas horas al estudio de disciplinas como estadística, *machine learning*, etc. Así como el estudio de las neurociencias que son la base de la inspiración biológica de este trabajo.

---

Al tratarse de un problema de investigación, no se pude prever la dimensión de este, pero he de reconocer que la problemática me ha apasionado y probablemente he pecado de inconformista a la hora de buscar una solución *elegante y congruente* en términos matemáticos. He querido entender en profundidad el problema y tratar de ser lo más formal en el procedimiento como me fuera posible. He tratado de buscar siempre la bioinspiración y de acercarme lo máximo posible a la realidad biológica con tal de proponer una solución creativa, original y que realmente pueda suponer un aporte al estado del arte.

Por todo esto, me ha supuesto un problema enorme poner un punto y final a lo que propuse como TFG, es decir, este trabajo empezó a plantearse en noviembre de 2013 y no he sabido cerrarlo en las dos convocatorias que se han celebrado hasta hoy. No ha sido hasta muy recientemente que he sabido plantear una solución que me pareciera lo suficientemente buena para el problema y concluir así este proyecto. Aunque en estas líneas que componen la memoria se ha tratado de ser didáctico y se ha tratado de enlazar todo el trabajo de forma sencilla, el proceso hasta construir la propuesta de solución expuesta en la sección 2.7 ha sido un trabajo de meses que difícilmente se puede expresar aquí. Así, en estas últimas semanas me he atropellado recopilando y redactando esta memoria y es por esto que pido disculpas al lector por posibles erratas o faltas que pueda encontrar en esta.

Por último quisiera disculparme con el lector por no haber podido incluir en esta memoria los resultados de la experimentación, desde el momento en que *dí* con una solución *apropiada* hasta el momento en que escribo estas líneas ha sido una batalla contra el tiempo que, a este respecto, he perdido. Confío en que en el momento de la presentación de este proyecto pueda contar con estos resultados.

# Bibliografía

- [1] Ben M. Clopton, Jeffrey A. Winfield, and Frank J. Flammino. Tonotopic organization: Review and analysis. *Brain Research*, 76(1):1 – 20, 1974.
- [2] J. Aleksandar D. Andina, A. Álvarez-Vellisco and J. Fombellida. Artificial metaplasticity can improve artificial neural networks learning. *Intelligent Automation & Soft Computing*, 26:683–696, 2009.
- [3] SciPy developers. *SciPy documentation*, 2014 (visitado en Noviembre, 2014).
- [4] Python Software Foundation. *Python 2.7.8 documentation*, 2014 (visitado en Noviembre, 2014).
- [5] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [6] Colin Humphries, Einat Liebenthal, and Jeffrey R. Binder. Tonotopic organization of human auditory cortex. *NeuroImage*, 50(3):1202 – 1211, 2010.
- [7] Sci kit developers. *Sci-kit-learn documentation*, 2014 (visitado en Noviembre, 2014).
- [8] N. Mayer, J.M. Herrmann, and T. Geisel. Retinotopy and spatial phase in topographic maps. *Neurocomputing*, 32–33(0):447 – 452, 2000.