

**UNIVERSIDAD DE SANTIAGO DE CHILE  
FACULTAD DE INGENIERIA  
DEPARTAMENTO DE INGENIERIA INFORMÁTICA**



**ANÁLISIS DE LA EFICIENCIA DEL ENTRENAMIENTO BASADO EN  
SIMULATED ANNEALING EN REDES NEURONALES PROFUNDAS**

Felipe Alberto Reyes González

Profesor Guía: Victor Parada

Tesis de grado presentada en conformidad a los  
requisitos para obtener el grado de Magíster en  
Ingeniería Informática

Santiago, Chile

2017

© Felipe Alberto Reyes González- 2017



• Algunos derechos reservados. Esta obra está bajo una Licencia Creative Commons Atribución-Chile 3.0. Sus condiciones de uso pueden ser revisadas en:

<http://creativecommons.org/licenses/by/3.0/cl/>.

# TABLA DE CONTENIDOS

<b>1. Introducción</b>	<b>1</b>
1.1. Antecedentes y motivación . . . . .	1
1.2. Descripción del problema . . . . .	4
1.3. Solución propuesta . . . . .	4
1.3.1. Características de la solución . . . . .	4
1.3.2. Propósito de la solución . . . . .	4
1.4. Objetivos y alcances del proyecto . . . . .	5
1.4.1. Objetivo general . . . . .	5
1.4.2. Objetivos específicos . . . . .	5
1.4.3. Alcances . . . . .	5
1.5. Metodología y herramientas utilizadas . . . . .	5
1.5.1. Metodología de trabajo . . . . .	5
1.5.2. Herramientas de desarrollo . . . . .	6
<b>2. Aspectos teóricos y revisión de la literatura</b>	<b>7</b>
2.1. Aspectos teóricos . . . . .	7
2.1.1. Algoritmo de retropropagación . . . . .	7
2.1.2. El perceptrón multicapa . . . . .	7
2.1.3. El desvanecimiento del gradiente . . . . .	12
2.2. Revisión de la literatura . . . . .	13
2.2.1. Métodos de búsqueda global . . . . .	14
2.2.2. El modelo de memoria a corto y largo plazo . . . . .	15
2.2.3. Preprocesamiento de la señal . . . . .	15
Referencias bibliográficas . . . . .	16

# ÍNDICE DE TABLAS

1.1. Especificaciones del equipo . . . . .	6
--	---

## ÍNDICE DE FIGURAS

2.1. Perceptrón multicapa . . . . .	8
2.2. Funciones de activación mas utilizadas. . . . .	10
2.3. Gradiente descendente . . . . .	13

# INTRODUCCIÓN

## ANTECEDENTES Y MOTIVACIÓN

Las redes neuronales (*Neural Networks*, NN) son sistemas de procesamiento de información que basan su estructura en una analogía de las redes neuronales biológicas. Consisten en un conjunto de elementos de procesamiento simple llamados nodos, estos nodos están dispuestos en una estructura jerárquica y conectadas entre sí por un valor numérico llamado peso que, mediante un proceso de entrenamiento, varía su valor.

La actividad que una neurona realiza en una NN es simple. El proceso consiste en ponderar las entradas de la neurona por los pesos de las conexiones de la neurona para luego ser sumadas y entregadas a la función de activación asociada a la neurona (McCulloch y Pitts, 1943). La salida corresponderá a la respuesta que la neurona genera a la entrada que se presentada.

Las neuronas con una función de activación umbral, fueron estudiadas por Rosenblatt (1962) quién las denominó *perceptrón*. El modelo básico se compone de 3 capas conectadas consecutivamente. La primera capa corresponde a la capa de entrada, que recibe el patrón de entrada a clasificar. La segunda capa contiene las neuronas de asociación o detección de características. Y la tercera capa es la capa de salida, que contiene las neuronas que reconocen los patrones. La limitación de los perceptrones se debe a capacidad de clasificación basado en el umbral, lo que no permite clasificar patrones mas complejos.

El conjunto de  $n$  neuronas se llamará capa, y una NN puede estar compuesta de una o más capas. Cada capa estará compuesta por una cantidad de neuronas que no necesariamente será la misma para todas las capas, y estarán dispuesta en forma consecutiva de tal manera que las capas se conecten unas con otras y siempre hacia adelante. La primera capa, la de entrada, recibirá un patrón que será entregado a las distintas neuronas que la capa posea. Cada neurona de la capa de entrada procesará los datos y generará una salida que servirá de entrada para la capa siguiente, repitiendo el proceso para cada una de las capas de la NN hasta llegar a la capa de salida, en cuyo caso la salida representará la respuesta de la red, concretando así el ciclo.

Las NN han sido utilizadas para la clasificación de entradas, y han sido diseñados diversos métodos para entrenar la red y que los pesos se adapten de tal manera que la salida de la red sea representativa de la salida esperada, a esto método de entrenamiento se le llama supervisado. Dentro de los métodos clásicos de entrenamiento se encuentra el método del gradiente descendente, el método de Newton, el gradiente conjugado, el método quasi-Newton, o el algoritmo Levenberg-Marquardt. El más utilizado es el método del gradiente, que consiste en actualizar los

pesos de las distintas neuronas en función de la dirección contraria al gradiente de la función de activación, logrando minimizar el error.

La técnica de minimización más simple, cuando el gradiente está disponible, es seleccionar la dirección de descenso más empinada y aplicar una búsqueda unidireccional a lo largo de esta dirección. Esta técnica de optimización de descenso más pronunciada se define por

$$\Delta(n)W = -\lambda(n)G(n) \quad (1.1)$$

Donde  $\Delta W(n)$  es la variación de los pesos para una iteración y  $\lambda(n)$  es un coeficiente que minimiza la función en la dirección de descenso. Se ha demostrado que no hay ventaja en encontrar el mínimo exacto en la dirección de búsqueda en cada iteración (Dennis y Schnabel, 1983). El algoritmo de retropropagación propuesto por Werbos (1974) y popularizado por Rumelhart et al (1986) se basa en una variación de la técnica de pendiente más pronunciada. No se utiliza búsqueda unidireccional sino un paso de descenso fijo,  $\eta$ , llamado tasa de aprendizaje, que se añade a una fracción de la última variación,  $\alpha$ , llamada momentum. El último término introduce algunos elementos del método de gradiente conjugado. Este algoritmo se define por:

$$\nabla_p W_{ij}^{(L)}(n) = -\eta G_{p_{ij}}^{(L)}(n) + \alpha \Delta_p W_{ij}^L(n-1) \quad (1.2)$$

Al hacer una actualización continua por:

$$\Delta W(n) = -\eta G(n) + \alpha \Delta W(n-1) \quad (1.3)$$

Cuando se elige un procedimiento de actualización por lotes

Mediante el cálculo de las segundas derivadas de la función objetivo, se obtiene una mejor comprensión de la topología de la función, que conduce a su vez a elegir una dirección decente más eficiente. Dejar:

$$\Delta W(n) = \lambda(n)S(n) \quad (1.4)$$

Donde la dirección de descenso  $S(n)$  está definida por:

$$S(n) = -[H(n)]^{-1}G(n) \quad (1.5)$$

Y donde  $H(n)$  es la matriz Hessiana. La principal dificultad con este enfoque es que encontrar la

solución de este sistema en cada iteración es una tarea muy tediosa. Los métodos métricos variable, también llamados métodos cuasi-Newton, eluden esta dificultad aproximando directamente el inverso de la matriz de Hessiana,  $[H(n)]^{-1}$ , de la primera derivada,  $G(n)$ . Estos métodos son las técnicas de optimización sin restricciones más populares y, entre ellas, BGFS es el método más utilizado.

El gradiente descendente es el mas simple de los métodos. Este requiere información del gradiente del vector, por lo tanto corresponde a un método de primer orden. El método comienza en un punto inicial y, mientras el criterio de parada no se cumpla, la solución inicial se moverá en la dirección contraria al gradiente a razón de una tasa de aprendizaje establecida. La tasa de aprendizaje es quién decide cuan largo será el paso, si éste es muy grande, puede que la convergencia a la solución sea superada por un paso muy grande, en cambio, una tasa de aprendizaje muy pequeña puede demorar la convergencia.

El método de Newton es un método de segundo orden, pues hace uso de la matriz Hessiana. El objetivo de este método es encontrar una mejor dirección de entrenamiento mediante el uso de la segunda derivada de la función de perdida. Considera una aproximación cuadrática de la función en la solución inicial utilizando una expansión de la serie de Taylor. De esta manera, el método de Newton permite mover la solución inicial en función de la inversa de la matriz Hessiana, que pasará a ser la dirección de entrenamiento de Newton.

El método del gradiente del conjugado fue propuesto por Fletcher y Reeves (1964), utiliza sucesivas direcciones conjugadas basadas en el gradiente y el residuo. Si la función objetivo es cuadrática y la dirección de búsqueda es minimizada exactamente en cada iteración, el método converge de forma cuadrática. Leonard y Kramer (1990) han utilizado el método para entrenar redes neuronales como alternativa a la retropropagación. Los pesos de una red se actualizan de acuerdo con una búsqueda unidireccional en la dirección de descenso  $S(n)$  de la siguiente forma:

$$\Delta W(n) = \lambda(n)S(n) \quad (1.6)$$

Y la dirección de descenso  $S(n)$  se calcula a partir del gradiente de iteraciones pasadas y presentes como muestra la ecuación 1.7

$$S(n) = -G(n) + \frac{\|G(n)\|}{\|G(n-1)\|}S((n-1)) \quad (1.7)$$

Para que las direcciones de descenso permanezcan conjugadas, la búsqueda unidireccional, en cada iteración, tiene que ser llevada con alta precisión. Sin embargo, después de muchas iteraciones, las direcciones podrían llegar a ser casi paralelas. Para superar esta dificultad, Fletcher y Reeves (1964) sugieren volver a iniciar el procedimiento igualando la dirección de descenso al



gradiente en cada iteración  $(t + 1)$ , donde  $(t)$  es el número total de pesos en la red.

## DESCRIPCIÓN DEL PROBLEMA

La retropropagación basa su funcionamiento en la regla de la cadena para poder calcular los gradientes, y a medida que el error se propaga hacia la capa de entrada de la red el gradiente comienza a disminuir su valor por cada capa que atraviesa. Esto significa que el gradiente disminuirá de manera exponencial, lo que representa un problema para una red de muchas capas, ya que las capas mas cercanas a la capa de entrada necesitarán más tiempo para ser entrenadas.

## SOLUCIÓN PROPUESTA

### Características de la solución

Mediante el uso de el algoritmo *simulated annealing* se busca analizar la eficiencia que la NN alcanza en una red neuronal profunda frente a otros métodos de aprendizaje.

### Propósito de la solución

El propósito de la solución es aportar en el campo de las redes neuronales y la clasificación de datos, proporcionando un análisis comparativo de la convergencia de distintas redes.

## **OBJETIVOS Y ALCANCES DEL PROYECTO**

### **Objetivo general**

Evaluar el desempeño del algoritmo *simulated annealing* y su efecto sobre entrenamiento de redes neuronales profundas.

### **Objetivos específicos**

Los objetivos establecidos para el presente trabajo son descritos a continuación

1. Definir las reglas de aprendizaje a implementar.
2. Construir los conjuntos de datos de entrada y salida a analizar.
3. Establecer los parámetros de las redes neuronales para la experimentación.
4. Entrenar las redes con los distintos conjuntos de datos.
5. Establecer las conclusiones del trabajo.

### **Alcances**

## **METODOLOGÍA Y HERRAMIENTAS UTILIZADAS**

### **Metodología de trabajo**

Considerando el aspecto investigativo del trabajo, se considera la utilización del método científico. Entre las actividades que componen la metodología, Sampieri (2006) describe los siguientes pasos para desarrollar una investigación:

- **Formulación de la hipótesis:** Las redes neuronales que adolecen del desvanecimiento del gradiente se ven beneficiadas por el uso del algoritmo *simulated annealing* en la convergencia.
- **Marco teórico:** Una revisión de la literatura donde se aborda el problema planteado, para situarse en el contexto actual de los problemas. Se describirán redes neuronales que buscan solucionar el mismo problema.
- **Diseño de la solución:** Se deberá diseñar el experimento para generar los datos que permitan sustentar las comparaciones entre las distintas redes. Diseñar y ejecutar el experimento basado en entradas equivalentes.
- **Análisis y verificación de los resultados:** Los resultados se analizarán considerando los valores de convergencia de los distintos métodos.
- **Presentación de los resultados:** Se presentarán tablas que describan los resultados obtenidos y que se consideren pertinentes.
- **Conclusiones obtenidas en el desarrollo de la investigación.**

## Herramientas de desarrollo

Para el desarrollo y ejecución de los experimentos se utilizará un equipo con las siguientes características

Sistema Operativo	Solus 2017.04.18.0 64-bit
Procesador	Intel® Core™ i5-2450M CPU @ 2.50GHz x 4
RAM	7.7Gb
Gráficos	Intel® Sandybridge Mobile
Almacenamiento	935.6 GB

Tabla 1.1: Especificaciones del equipo

El software que se utilizará es:

- **Plataforma de desarrollo:** Atom.
- **Lenguaje de programación:** Python.
- **Sistema de redes neuronales:** Keras API (Chollet, 2015).
- **Herramienta ofimática:** L<sup>A</sup>T<sub>E</sub>X.

# ASPECTOS TEÓRICOS Y REVISIÓN DE LA LITERATURA

## ASPECTOS TEÓRICOS

### Algoritmo de retropropagación

El elemento básico de las redes neuronales (*Neural Networks*, NN) es el nodo, que recibe un vector de entrada para producir una salida. Cada entrada tiene asociado un vector de pesos  $w$ , que se va modificando durante el proceso de aprendizaje. Cada unidad aplica una función  $f$  sobre la suma ponderada de las entradas ponderada donde el resultado puede servir como entrada de otras unidades.

Existen dos fases importante dentro del modelo

- Fase de entrenamiento: Se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos que definen el modelo de la NN. Se calculan de manera iterativa, de acuerdo con los valores de entrenamiento, con el objeto de minimizar el error cometido entre la salida obtenida por la NN y la salida deseada.
- Fase de prueba: Durante el entrenamiento, el modelo se ajusta al conjunto de entrenamiento, perdiendo la habilidad de generalizar su aprendizaje a casos nuevos, a esta situación se le llama sobreajuste. Para evitar el sobreajuste, se utiliza un segundo grupo de datos diferentes, el conjunto de validación, que permitirá controlar el proceso de aprendizaje.

Los pesos óptimos se obtienen minimizando una función. Uno de los criterios utilizados es la minimización del error cuadrático medio entre el valor de salida y el valor real esperado.

### El perceptrón multicapa

Dentro de las redes neuronales, el perceptrón multicapa es una de las arquitecturas más usadas para resolver problemas. Esto es debido a que poseen la capacidad de ser un aproximador universal (Marvin Minsky, 1987). Esto no implica que sea una de las redes más potentes o con mejores resultados, el perceptrón multicapa posee una serie de limitaciones, como el proceso de aprendizaje para problemas que dependan de un gran número de variables, la dificultad para

realizar un análisis teórico de la red debido a la presencia de componentes no lineales y a la alta conectividad.

### Arquitectura

El perceptrón multicapa posee una estructura de capas compuestas por neuronas. Cada una de las capas está formada por un conjunto de neuronas y se distinguen tres tipos de capas: la capa de entrada, las capas ocultas y la capa de salida.

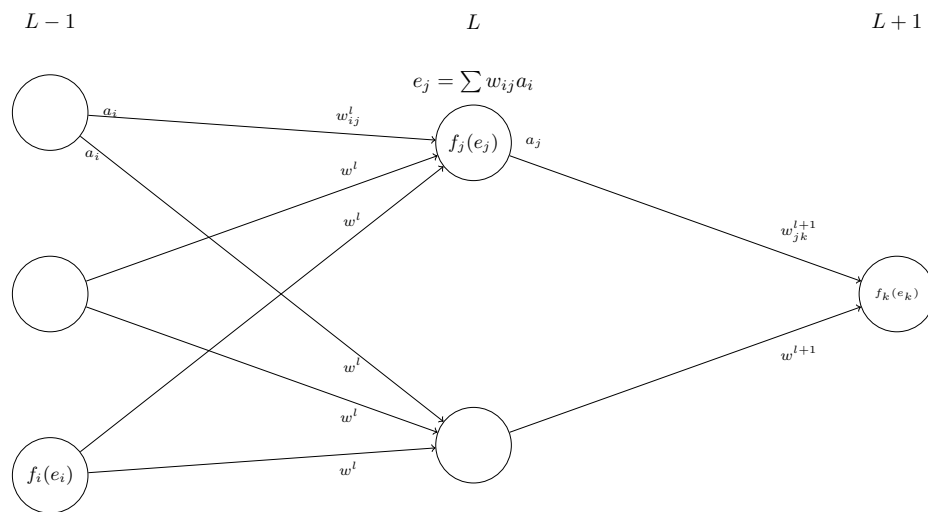


Figura 2.1: Perceptrón multicapa

En la figura 2.1 se observa que las conexiones van siempre hacia adelante. Las neuronas de la capa  $l$  se conectan con las neuronas de la capa  $l + 1$ . Las neuronas de la capa de entrada se encargan de recibir los patrones y propagar dichas señales a las neuronas de la capa siguiente. La última capa, la capa de salida, proporciona la respuesta de la red al patrón presentado. Las neuronas de las capas ocultas realizan el procesamiento de las señales generadas por el patrón de entrada.

### Propagación de la entrada y el algoritmo de retropropagación

El perceptrón multicapa define una relación entre la entrada y la salida. Esta relación se obtiene propagando hacia adelante los valores de las variables de entrada, es por esto que también se

les llama redes *feedforward*. Cada neurona de la red procesa la entrada recibida y produce una respuesta que se propaga, mediante las conexiones, hacia las neuronas de la capa siguiente.

Si un perceptrón multicapa con  $C$  capas y  $n_c$  neuronas en la capa  $c$ , donde  $W_c = (w_{ij}^c)$  es la matriz de pesos,  $w_{ij}^c$  representará el peso de la conexión de la neurona  $i$  de la capa  $c$  hasta la neurona  $j$  de la capa siguiente. Denotaremos  $a_i^c$  a la activación de la neurona  $i$  de la capa  $c$  que se calcula de la siguiente manera:

- **Activación de una neurona de la capa de entrada:** Las neuronas se encargan de transmitir la entrada recibida, por lo tanto

$$a_i^1 = x_i, i = 1, 2, \dots, n$$

donde  $X = (x_1, x_2, \dots, x_n)$  representa el vector de entrada.

- **Activación de una neurona de la capa oculta:** Las neuronas de una capa oculta procesa la información recibida aplicando la función de activación  $f$  a la suma de los productos de la entrada por sus pesos, es decir

$$a_i^c = f \left( \sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} a_j^{c-1} + \theta_i^c \right), i = 1, 2, \dots, n_c; c = 2, 3, \dots, C - 1$$

donde  $a_j^{c-1}$  es la salida de la capa anterior a  $c$ .

- **Activación de una neurona de la capa de salida:** La activación de una neurona de la capa de salida viene dada por la función de activación  $f$  aplicada a la suma de los productos de la entrada por sus pesos, es decir

$$y_i = a_i^c = f \left( \sum_{j=1}^{n_{c-1}} w_{ji}^{C-1} a_j^{C-1} + \theta_i^C \right), i = 1, \dots, n_c$$

donde  $Y = (y_1, y_2, \dots, y_{n_c})$  es el vector de salida.

La función  $f$  es la función de activación de la neurona. Las funciones de activación mas utilizadas son la sigmoideal y la tangente hiperbólica, descritas en las ecuaciones 2.1 y 2.2 respectivamente.

$$f_{sigm}(x) = \frac{1}{1 + \exp(-x)} \quad (2.1)$$

$$f_{tanh}(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)} \quad (2.2)$$

Ambas funciones poseen como imagen intervalo de valores entre  $[0, 1]$  y  $[-1, 1]$  como se observa en la figura 2.2.

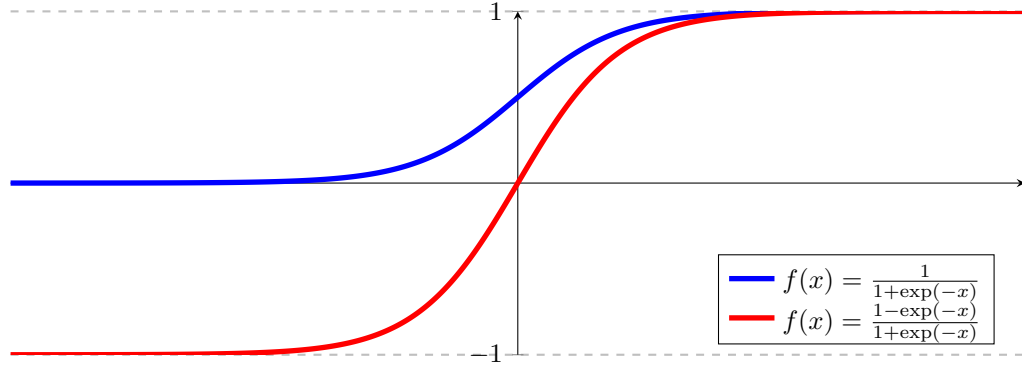


Figura 2.2: Funciones de activación mas utilizadas.

El perceptrón multicapa actualiza sus pesos en función de una regla de aprendizaje, de tal manera que los nuevos pesos permitan reducir el error de salida. Por tanto, para cada patrón de entrada a la red es necesario disponer de un patrón de salida deseada. El objetivo es que la salida de la red sea lo más próxima posible a la salida deseada, debido a esto es que el aprendizaje de la red se describe como un problema de minimización de la siguiente manera

$$\min_W E$$

donde  $W$  es el conjunto de parámetros de la red (pesos y umbrales) y  $E$  es una función de error que evalúa la diferencia entre las salidas de la red y las salidas deseadas. En la mayor parte de los casos, la función de error se define como:

$$E = \frac{1}{N} \sum_{i=1}^N e(i) \quad (2.3)$$

Donde  $N$  es el número de muestras y  $e(n)$  es el error cometido por la red para el patrón  $i$ , definido de la siguiente manera

$$e(i) = \frac{1}{n_C} \sum_{j=1}^{n_C} (s_j(i) - y^j(n))^2 \quad (2.4)$$

Siendo  $Y(i) = (y_1(i), y_2(i), \dots, y_{n_C}(i))$  y  $S(i) = (s_1(i), s_2(i), \dots, s_{n_C}(i))$  los vectores de salida y salidas deseadas para el patrón  $i$  respectivamente.

De esta manera, si  $W^*$  es un mínimo de la función de error  $E$ , en dicho punto el error será cercano a cero, y en consecuencia, la salida de la red será próxima a la salida deseada. La presencia de funciones de activación no lineales hace que la respuesta de la red sea no lineal respecto a los parámetros ajustables, por lo que el problema de minimización es un problema no lineal y se hace

necesario el uso de técnicas de optimización no lineales para su resolución.

Las técnicas utilizadas suelen basarse en la actualización de los parámetros de la red mediante la determinación de una dirección de búsqueda. En el caso de las redes neuronales multicapa, la dirección de búsqueda más utilizada se basa en la dirección contraria del gradiente de la función de error  $E$ , el método de gradiente descendente.

Si bien el aprendizaje de la red busca minimizar el error total de la red, el procedimiento está basado en métodos del gradiente estocástico, que son una sucesión de minimizaciones del error  $e(i)$  por cada patrón, en lugar de minimizar el error total  $E$  de la red. Aplicando el método del gradiente estocástico, cada parámetro  $w$  se modifica para cada patrón de entrada  $n$  según la siguiente regla de aprendizaje

$$w(i) = w(n-1) - \alpha \frac{\partial e(i)}{\partial w} \quad (2.5)$$

donde  $e(i)$  es el error para el patrón de entrada  $i$  dado por la ecuación 2.4, y  $\alpha$  es la tasa de aprendizaje, éste último determina el desplazamiento en la superficie del error.

Como las neuronas están ordenadas por capas y en distintos niveles, es posible aplicar el método del gradiente de forma eficiente, resultando en el *algoritmo de retropropagación* (Rumelhart, Hinton, y Williams, 1986) o *regla delta generalizada*. El término retropropagación es utilizado debido a la forma de implementar el método del gradiente en las redes multicapa, pues el error cometido en la salida de la red es propagado hacia atrás, transformándolo en un error para cada una de las neuronas ocultas de la red.

El algoritmo de retropropagación es el método de entrenamiento más utilizado en redes con conexión hacia adelante. Es un método de aprendizaje supervisado, en el que se distinguen claramente dos fases:

1. Se aplica un patrón de entrada, el cual se propaga por las distintas capas que componen la red hasta producir la salida de la misma. Esta salida se compara con la salida deseada y se calcula el error cometido por cada neurona de salida.
2. Estos errores se transmiten desde la capa de salida, hacia todas neuronas de las capas anteriores (Fritsch, 1996). Cada neurona recibe un error que es proporcional a su contribución sobre el error total de la red. Basándose en el error recibido, se ajustan los errores de los pesos sinápticos de cada neurona.



## El desvanecimiento del gradiente

El problema del gradiente desvaneciente nace en las NN profundas, éstas utilizan funciones cuyo gradiente se encuentran entre 0 y 1. Debido a que estos gradientes pequeños se multiplican durante la retropropagación, tienden a *desvanecerse* a través de las capas, evitando que la red aprenda.

Si se tiene una NN, la activación de una neurona de una capa intermedia  $i$  con función de activación  $f_i$  y con entrada

$$net_i(t) = \sum_j w_{ji} y^j(t-1)$$

es

$$y^i(t) = f_i(net_i(t))$$

Además  $w_{ji}$  es el peso de la conexión desde la unidad  $j$  de la capa anterior hasta la unidad  $i$  de la capa actual,  $d_k(t)$  será la respuesta esperada de la unidad  $k$  de la capa de salida en el tiempo  $t$ . Usando el error cuadrático medio (*Mean square error*, MSE), el error de  $k$  será

$$E_k(t) = (d_k(t) - y^k(t))^2$$

En un tiempo  $\tau \leq t$  cualquiera, el error de una neurona  $j$  que no sea una neurona de entrada es la suma de los errores externos y el error propagado hacia atrás desde la neurona previa será

$$\vartheta_j(\tau) = f'_j(net_j(\tau)) \left( E_j(\tau) + \sum_i w_{ij} \vartheta_i(\tau+1) \right)$$

El peso actualizado en el tiempo  $\tau$  resulta  $w_{jl}^{new} = w_{jl}^{old} + \alpha \vartheta_j(\tau) y^l(\tau-1)$  donde  $\alpha$  es la tasa de aprendizaje, y  $l$  es una unidad arbitraria conectada a la unidad  $j$ .

La propagación hacia atrás de un error que ocurre en una unidad  $u$  en un tiempo  $t$  hacia una unidad  $v$  para  $q$  pasos, escala el error de la siguiente manera

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f'_v(net_v(t-1)) w_{uv} & q = 1 \\ f'_v(net_v(t-q)) \sum_{l=1}^n \frac{\partial \vartheta_l(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q > 1 \end{cases} \quad (2.6)$$

Con  $l_q = v$  y  $l_0 = u$ , el factor de escalamiento es

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^n \cdots \sum_{l_{q-1}=1}^n \prod_{m=1}^q f'_{l_m}(net_{l_m}(t-m)) w_{l_m l_{m-1}} \quad (2.7)$$

La sumatoria de los  $n^{q-1}$  términos  $\prod_{m=1}^q f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}$  escalan el error. Los distintos términos pueden tener signos diferentes, por lo tanto, el aumento del número de unidades  $n$  no implica un incremento del error absoluto. Pero con mas unidades se incrementa la expectativa de que el valor absoluto del error aumente. Si  $\rho(m, l_m, l_{m-1}) := |f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}}| < 1,0$  para todo  $m$ , el producto en (2.7) decrece exponencialmente con  $q$ , es decir, el error se desvanece como muestra la figura 2.1.3. Un error que se desvanece a lo largo del flujo casi no tiene efecto en la actualización de los pesos.

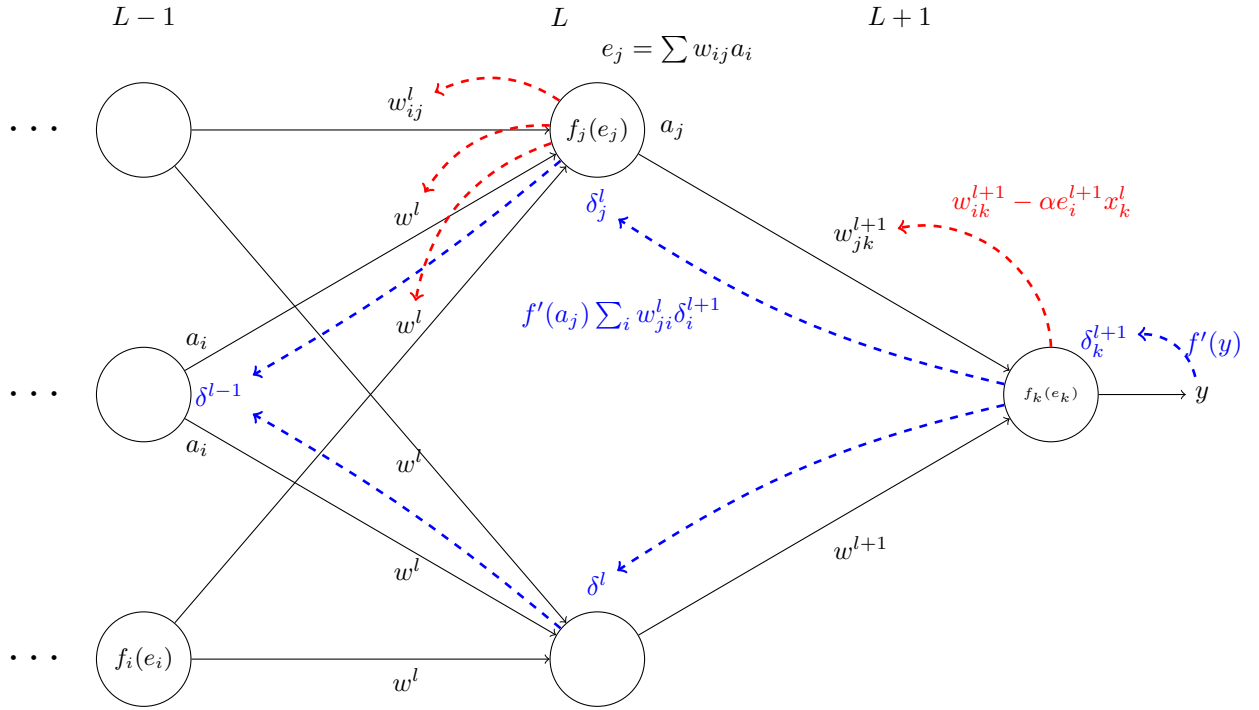


Figura 2.3: Gradiente descendente

## REVISIÓN DE LA LITERATURA

Muchos de los métodos utilizados (Elman, 1990; Schmidhuber, 1992; Pearlmutter, 1989, 1995) sufren del desvanecimiento del gradiente. Para solventar el problema hay diversos métodos que lo evitan.

## Métodos de búsqueda global

Los métodos de búsqueda global no utilizan el gradiente. Métodos como *simulated annealing* (SA), *multi-grid random search* (Bengio, Simard, y Frasconi, 1994) y *random weight guessing* (Schmidhuber y Hochreiter, 1996) han sido investigados. Se ha encontrado que los métodos de búsquedas globales funcionan bien en problemas que involucren dependencias a largo plazo y que además utilizan redes que contienen pocos parámetros y no precisan de alta precisión en sus calculos.

El método SA es un algoritmo de búsqueda meta-heurística para problemas de optimización global. La técnica está basada en el proceso de calentamiento de un metal, vidrio o cristal hasta su punto de fusión, para luego enfriarlo hasta obtener una estructura cristalina. El algoritmo 1 describe el procedimiento de la siguiente manera: dada una solución inicial se genera un conjunto de soluciones que son una perturbación de la solución actual, al evaluar cada perturbación se actualizará la solución actual en caso de que la perturbación sea mejor que la solución actual, y en caso contrario, existirá una probabilidad de que la solución actual sea actualizada de igual manera.

---

**Algoritmo 1:** Simulated annealing

---

**Data:** Temperatura  $T$ , constante de Boltzmann  $k$ , factor de reducción  $c$

---

```
1  Seleccionar el mejor vector solución  $x_0$  a optimizar
2  while El criterio no se cumpla do
3      while Existan soluciones en el conjunto do
4          Seleccionar una solución  $x_0 + \Delta x$ 
5          if  $f(x_0 + \Delta x) < f(x_0)$  then
6               $f_{new} = f(x_0 + \Delta x)$ ;  $x_0 = x_0 + \Delta x$ 
7          else
8               $\Delta f = f(x_0 + \Delta x) - f(x_0)$ 
9              if  $rand(0, 1) > \exp(-\Delta f / kT)$  then
10                  $f_{new} = f(x_0 + \Delta x)$ ;  $x_0 = x_0 + \Delta x$ 
11             else
12                  $f_{new} = f(x_0)$ 
13             end
14         end
15          $f = f_{new}$ ;  $T = cT$ 
16     end
17 end
```

---

## El modelo de memoria a corto y largo plazo

Hochreiter y Schmidhuber (1997) introdujeron el modelo de memoria a corto y largo plazo (*Long short-term memory*, LSTM) como solución al problema del desvanecimiento del gradiente. La red LSTM se basa en el bloque de memoria, que se compone de una o más celdas de memoria, una compuerta de entrada y una compuerta de salida. Las entradas son unidades multiplicativas con activación continua y son compartidas por todas las celdas de un mismo bloque de memoria. Cada celda contiene una unidad lineal con una conexión recurrente local llamada carrusel de error constante (CEC), se conocerá como estado de la celda a la activación del CEC.

Cada celda recibe una entrada ponderada por los pesos correspondientes a la capa anterior. La compuerta de entrada se encarga de permitir o impedir el acceso de estos valores al CEC del interior de la celda. La compuerta de salida realiza una acción similar sobre la salida de la celda, tolerando o reprimiendo la difusión del estado del CEC al resto de la red.

Los bloques de memoria configuran una red LSTM, donde no se indican los sesgos de las distintas neuronas del modelo. La existencia de las conexiones con pesos  $W^{y,u}$  determina la naturaleza de la red. Así, si se permite la existencia de esta conexión, la red LSTM se puede considerar como una máquina neuronal de estados de MEaly, si no se permite, la red LSTM puede considerarse como una máquina de estados de Moore. El estado de la red LSTM está formado por las activaciones de las compuertas, el CEC y las celdas de los bloques de memoria.

## Preprocesamiento de la señal

Squartini, Hussain, y Piazza (2003) propone pre-procesar la señal de entrada a través de una descomposición wavelet, buscando separar la información a corto plazo de la información a largo plazo, y entrenando diferentes NN. Los resultados son combinados para alcanzar el objetivo final. Este enfoque simplifica el proceso de aprendizaje de la NN, evitando cambios relevantes en la arquitectura de la red y las técnicas de aprendizaje.

## REFERENCIAS BIBLIOGRÁFICAS

- Bengio, Y., Simard, P., y Frasconi, P. (1994, Mar). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166. doi: 10.1109/72.279181
- Chollet, F. (2015). *Keras*. <https://github.com/fchollet/keras>. GitHub.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179 - 211. Descargado de <http://www.sciencedirect.com/science/article/pii/036402139090002E> doi: [https://doi.org/10.1016/0364-0213\(90\)90002-E](https://doi.org/10.1016/0364-0213(90)90002-E)
- Fletcher, R., y Reeves, C. M. (1964, 1 de febrero). Function minimization by conjugate gradients. *The Computer Journal*, 7(2), 149–154. Descargado de <http://dx.doi.org/10.1093/comjnl/7.2.149> doi: 10.1093/comjnl/7.2.149
- Fritsch, J. (1996). *Modular neural networks for speech recognition* (Masters Thesis). KIT.
- Hochreiter, S., y Schmidhuber, J. (1997, noviembre). Long short-term memory. *Neural Comput.*, 9(8), 1735–1780. Descargado de <http://dx.doi.org/10.1162/neco.1997.9.8.1735> doi: 10.1162/neco.1997.9.8.1735
- Leonard, J., y Kramer, M. (1990). Improvement of the backpropagation algorithm for training neural networks. *Computers & Chemical Engineering*, 14(3), 337 - 341. Descargado de <http://www.sciencedirect.com/science/article/pii/0098135490870706> doi: [http://dx.doi.org/10.1016/0098-1354\(90\)87070-6](http://dx.doi.org/10.1016/0098-1354(90)87070-6)
- Marvin Minsky, S. A. P. (1987). *Perceptrons: An introduction to computational geometry* (Expanded ed.). The MIT Press.
- McCulloch, W. S., y Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 115–133. doi: 10.1007/BF02478259
- Pearlmutter, B. A. (1989, June). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2), 263-269. doi: 10.1162/neco.1989.1.2.263
- Pearlmutter, B. A. (1995, Sep). Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks*, 6(5), 1212-1228. doi: 10.1109/72.410363
- Rosenblatt, F. (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books.
- Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.
- Sampieri, R. (2006). *Metodología de la investigación*. México: McGraw Hill.
- Schmidhuber, J. (1992). A fixed size storage  $O(n^3)$  time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4, 243-248.

- Schmidhuber, J., y Hochreiter, S. (1996). *Guessing can outperform many long time lag algorithms* (Inf. Téc.).
- Squartini, S., Hussain, A., y Piazza, F. (2003, May). Preprocessing based solution for the vanishing gradient problem in recurrent neural networks. En *Circuits and systems, 2003. iscas '03. proceedings of the 2003 international symposium on* (Vol. 5, p. V-713-V-716 vol.5). doi: 10.1109/ISCAS.2003.1206412