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

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

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

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

1.1 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 jerarquica y conectadas entre si por un valor numérico llamado peso que, mediante un proceso de entrenamiento, varia su valor.

La actividad que una neurona realiza en una NN 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 (McCulloch y Pitts, 1943). La salida corresponderá a la respuesta que la neurona genera a la entrada.

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 la capacidad de clasificación basado en el umbral, lo que no permite clasificar patrones más 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 este 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 contraría 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

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

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. 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, η, llamado tasa de aprendizaje, que se añade a una fracción de la última variación, α, llamada momentum. El último término introduce algunos elementos del método de gradiente conjugado. Así la actualización se realiza como en la ecuación 1.2

$$W(n) \quad = \quad -\eta G(n) + \alpha \Delta W(n-1) \tag{1.2}$$

El método quasi-Newton utiliza 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 decendente más eficiente. Dejar:

$$\Delta W(n) \quad = \quad \lambda(n)S(n) \tag{1.3}$$

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

$$S(n) \quad = \quad .[H(n)]^{-1}G(n) \tag{1.4}$$

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 ariable, 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 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 unidencial en la dirección de descenso $S(n)$ de la siguiente forma:

$$\Delta W(n) = \lambda(n) S(n) \tag{1.5}$$

Y la dirección de descenso $S(n)$ se calcula a partir del gradiente de iteraciones pasadas y presentes como muestra la ecuación 1.6

$$S(n) \quad = \quad G(n) + \frac{||G(n)||}{||G(n-1)||} S((n-1)) \tag{1.6}$$

Para que las direcciones de descenso permanezcan conjugadas, la búsqueda unidirecional, 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.

1.2 DESCRIPCIÓN DEL PROBLEMA

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

1.3 SOLUCIÓN PROPUESTA

1.3.1 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.

1.3.2 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.

1.4 OBJETIVOS Y ALCANCES DEL PROYECTO

1.4.1 Objetivo general

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

1.4.2 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. Establecer los algoritmos de aprendizaje a comparar.

5. Entrenar las redes con los distintos conjuntos de datos.
6. Establecer las conclusiones del trabajo.

1.4.3 Alcances

1. Se analizará la misma arquitectura con diferentes reglas de aprendizaje.
2. Los conjunto de datos para el entrenamiento a utilizar son los propuestos en (Morse y Stanley, 2016).

1.5 METODOLOGÍA Y HERRAMIENTAS UTILIZADAS

1.5.1 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.
- 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 Tla investigación.

1.5.2 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:

- Lenguaje de programación: Python.
- Sistema de redes neuronales: Keras API (Chollet, 2015).
- Herramienta ofimática: LaTeX.

CAPÍTULO 2. ASPECTOS TEÓRICOS Y REVISIÓN DE LA LITERATURA

En esta sección se abarcan los aspectos relacionados al conocimiento general para la comprensión del presente trabajo y la revisión de la literatura asociada al trabajo presentado. Para realizar un análisis de la eficiencia en la convergencia de los algoritmos de aprendizaje es necesario conocer la base teórica de ésta. Para ellos, se describen los conceptos necesarios de las redes neuronales y el desvanecimiento del gradiente. La sección 2.1 se centra en explicar las partes fundamentales para el entendimiento del problema en cuestión.

2.1 ASPECTOS TEÓRICOS

2.1.1 El perceptrón multicapa y su arquitectura

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.

El perceptron 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.

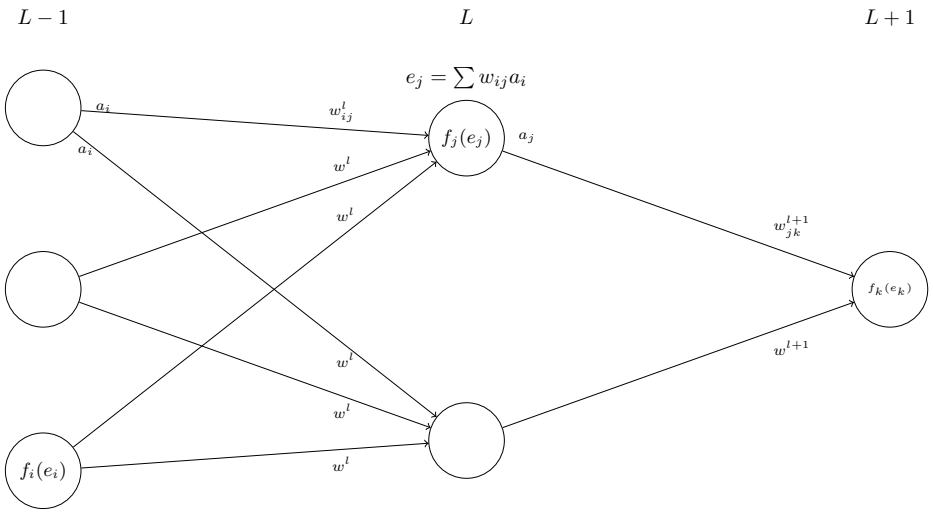

Figura 2.1: Perceptrón multicapa

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 patrónes 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 procesado de las señales generadas por el patrón de entrada.

2.1.2 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.

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.

  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.

- 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.

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 conexion 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

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

  donde $X = (x_1, x_2, \cdots, x_n)$ 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

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

  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

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

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

La función f es la función de activación de la neurona. Las funciones de activación mas utilizadas

son la sigmoidal y la tangente hiperbólica, descritas en las escuaciones 2.1 y 2.2 respectivamente.

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

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

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

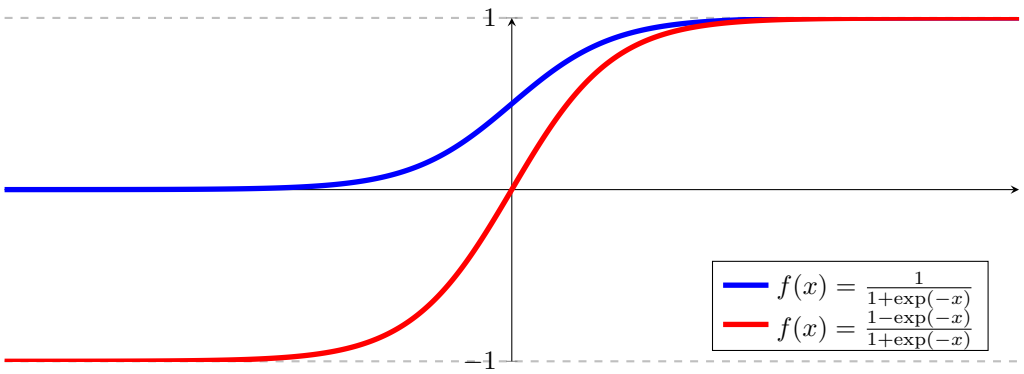

Figura 2.2: Funciones de activación mas utilizadas.

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:

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

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

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

Siendo $Y(i) = (y_1(i), y_2(i), \cdots, y_{n_C}(i))$ y $S(i) = (s_1(i), s_2(i), \cdots, s_{n_C}(i))$ 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

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

donde $e(i)$ es el error para el patrón de entrada i dado por la ecuación 2.4, y α 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.

2.1.3 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 τ resulta $w_{jl}^{new} = w_{jl}^{old} + \alpha \vartheta_j(\tau) y^l(\tau - 1)$ donde α 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

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

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

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

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 2.3. Un error que se desvanece a lo largo del flujo casi no tiene efecto en la actualización de los pesos.

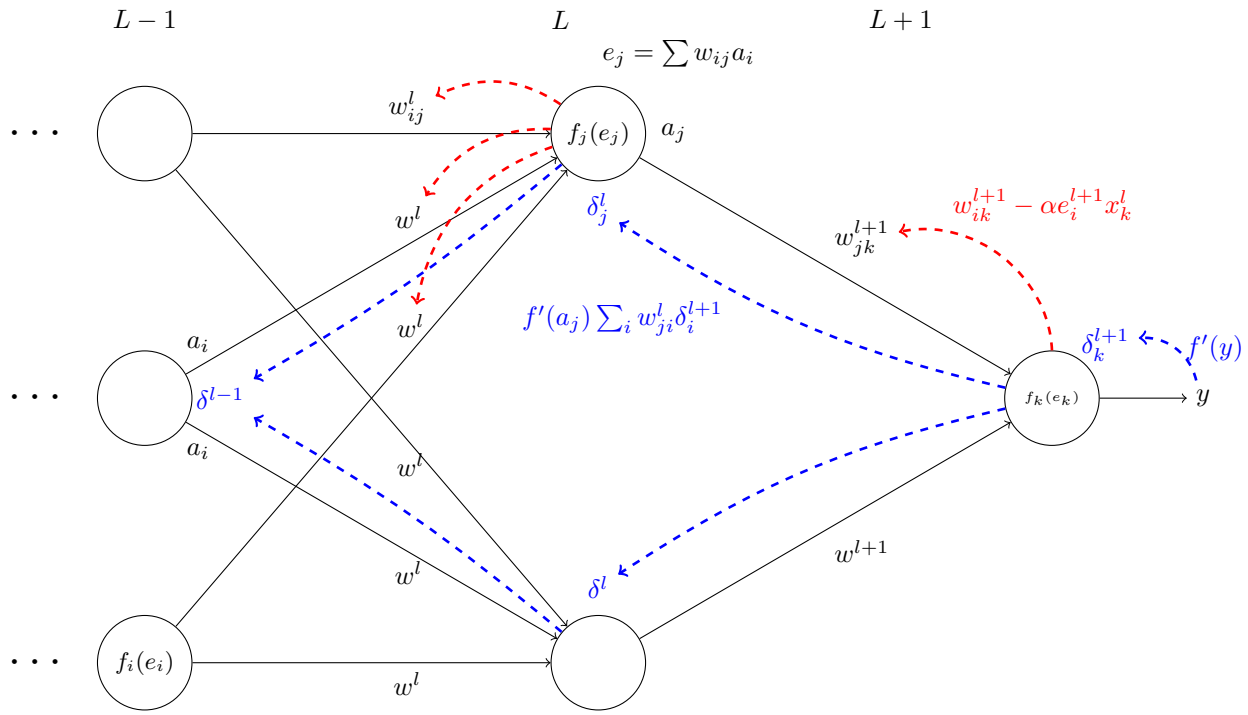

Figura 2.3: Gradiente descendente

2.2 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, como los métodos de búsqueda global, el modelo de memoria a corto y largo plazo o el preprocesamiento de las señales.

2.2.1 Gradiente descendente

2.2.2 Gradiente estocástico descendente

El método del gradiente estocástica descendente (*Stochastic gradient descent*, SGD) es una aproximación estocástica del método del gradiente descendente para la minimización de la función objetivo mediante la actualización de los pesos en cada iteración.

Algoritmo 1: SGD

Data: Vector de pesos W, taza de aprendizaje η, conjunto S de entrenamiento

1 **repeat**

2 Mezclar en forma aleatoria el conjunto S;

3 Evaluar la red con un patrón y calcular el error;

4 Actualizar los pesos $W = W - \eta \nabla Q_i(W)$;

5 **until**;

El algoritmo SGD (ver 1) trabaja como el método del gradiente descendente tradicional, pero acelera su funcionamiento

2.2.3 RMSProp

Tieleman y Hinton (2012) presentan un optimizador que utiliza la magnitud del gradiente para normalizar los gradientes. Siempre mantiene una media móvil sobre la raíz cuadrática media de los gradientes, por el cual se divide el gradiente actual. Sea $f'(\theta_t)$ la derivada de la función de pérdida con respecto a los parámetros en la etapa t del tiempo. En su forma básica, dada una tasa de paso α y un término de decaimiento γ se realizan las siguientes actualizaciones:

$$r_t = (1-\gamma) f'(\theta_t)^2 + \gamma r_{t-1} \tag{2.8}$$

$$v_{t+1} = \frac{\alpha}{\sqrt{r_t}} f'(\theta_t) \tag{2.9}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{2.10}$$

REFERENCIAS

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

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.

Leonard, J., y Kramer, M. (1990). Improvement of the backpropagation algorithm for training neural networks. *Çomputers & 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

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

Morse, G., y Stanley, K. O. (2016). Simple evolutionary optimization can rival stochastic gradient descent in neural networks. En *Proceedings of the genetic and evolutionary computation conference 2016* (pp. 477–484). New York, NY, USA: ACM. Descargado de http://doi.acm.org/10.1145/2908812.2908916 doi: 10.1145/2908812.2908916

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(n3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, *4*, 243-248.

Tieleman, T., y Hinton, G. (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning.