UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Análisis de la convergencia del algoritmo *simulated annealing* para redes neuronales profundas

Propuesta de tesis

Nombre: Felipe Alberto Reyes González

Programa: Magíster en Ingeniería Informática

Profesor patrocinante: Victor Parada

Cel.: 890 26 317

email: felipe.reyesg@usach.cl

16 de mayo de 2017

Tabla de contenido

1 Descripción del problema

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.

1.1 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.

1.1.1 Arquitectura

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.

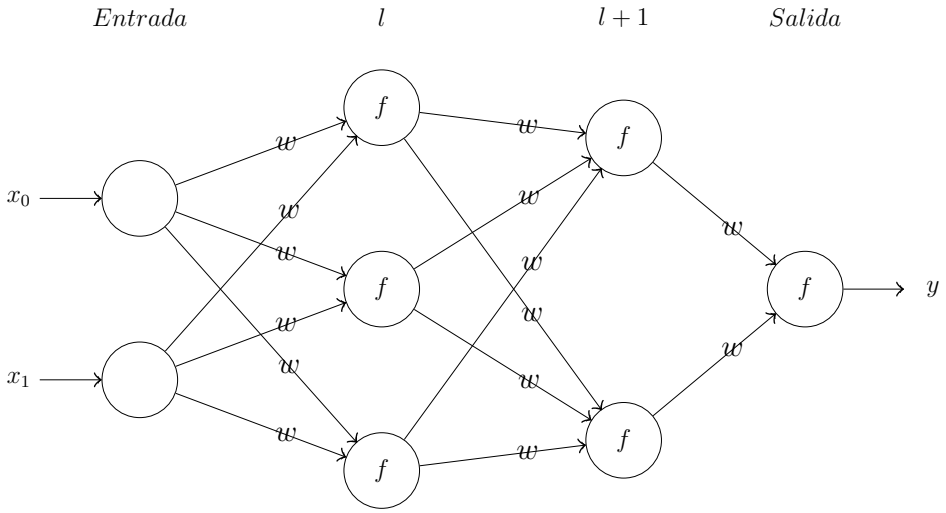

Figura 1: Perceptrón multicapa

En la figura 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.

1.1.2 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.

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 1 y 2 respectivamente.

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

$$f_{tanh}(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)} \tag{2}$$

Ambas funciones poseen como imagen intervalo de valores entre $[0, 1]$ y $[-1, 1]$ como se observa en la figura 2.

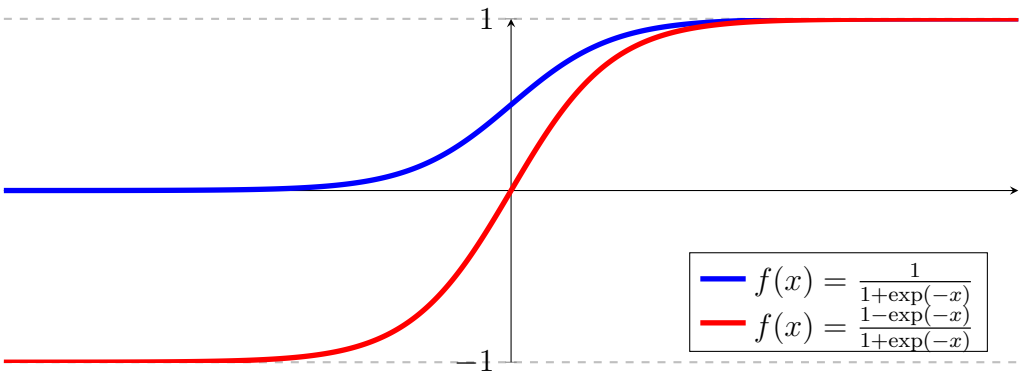

Figura 2: Funciones de activación mas utilizadas.

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:

$$E = \frac{1}{N}\sum_{i=1}^{N} e(i) \tag{3}$$

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

$$e(i) = \frac{1}{n_C}\sum_{j=1}^{n_C}(s_j(i) - y^j(n))^2 \tag{4}$$

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

$$w(i) = w(n-1) - \alpha \frac{\partial e(i)}{\partial w} \tag{5}$$

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

1.2 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

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \begin{cases} f_v'(net_v(t-1))w_{uv} & q=1 \\ f_v'(net_v(t-q))\sum_{l=1}^{n} \frac{\partial \vartheta(t-q+1)}{\partial \vartheta_u(t)} w_{lv} & q>1 \end{cases} \tag{6}$$

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

$$\frac{\partial \vartheta_v(t-q)}{\partial \vartheta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f_{l_m}'(net_{l_m}(t-m))w_{l_m l_{m-1}} \tag{7}$$

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 (7) decrece exponencialmente con q, es decir, el error se desvanece como muestra la figura 1.2. Un error que se desvanece a lo largo del flujo casi no tiene efecto en la actualización de los pesos.

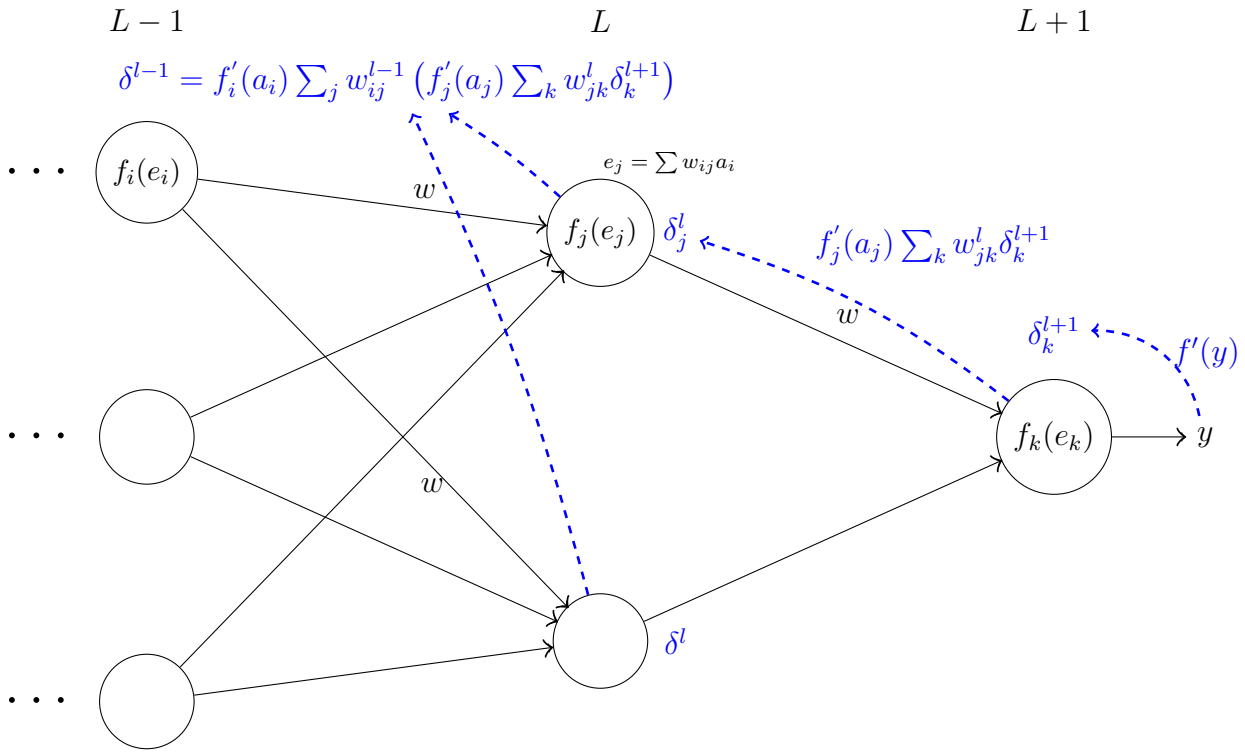

Figura 3: Gradiente descendente

2 Objetivos del proyecto

2.1 Objetivo general

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

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

3 Descripción de la solución

3.1 Estado del arte

Muchos de los métodos utilizados ((Elman, 1990; Schmidhuber, 1992a; Pearlmutter, 1989, 1995) ref-13-14-15-16-17) sufren del desvanecimiento del gradiente. Para solventar el problema hay cuatro tipos de soluciones.

(i) Métodos que no utilizan el gradiente Los métodos de búsqueda global no utilizan el gradiente. Métodos como *simulated annealing*, *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.

(ii) Métodos que obligan el uso de gradiente mas altos Los valores más grandes del gradiente pueden ser reforzados por la optimización pseudo-Newton ponderada en el tiempoy la propagación discreta del errror. Presentan problemas para almacenar información real de gran valor en el tiempo.

(iii) Métodos que operan en niveles mas altos Anteriormente se ha propuesto un enfoque EM para la propagación del objetivo (Bengio y Frasconi, 1993). Este enfoque utiliza un número discreto de estados y, por lo tanto, tendrá problemas con valores continuos.

Las técnicas de filtrado de Kallman se utilizan para el entrenamiento de redes recurrentes (Puskorius y Feldkamp, 1994). Sin embargo, un factor de descuento derivado conduce a problemas de desvanecimiento del gradiente.

Si un problema de retraso a largo plazo contiene regularidades locales, un sistema jerárquico chunket funciona bien (Schmidhuber, 1992b).

(iv) Métodos que utilizan arquitecturas especiales Las redes de segundo orden (utilizando unidades sigma-pi) son, en principio, capaces de aumentar el flujo de error, pero los problemas derivados del desvanecimiento del gradiente difícilmente pueden evitarse ref-22-23.

Con una red neuronal con retardo (*Time-Delay Neural Network*, TDNN ref-24), las activaciones de la red de pasos de tiempo anteriores son devueltas a la red usando líneas de retardo fijo. En las TDNN la disminución de error se ralentiza porque el error utiliza .ªccesos rápidos.ª medida que se propagan de nuevo. Las TDNN tienen que hacer frente a un trade-off: el aumento de la longitud de la línea de retardo aumenta el flujo de error, pero la red tiene más parámetros/unidades. Casos especiales de TDNN son las redes NARX ref-25, y la suma ponderada de los viejos activaciones en lugar de una línea de retardo fijo ref-26. Una versión más compleja de un TDNN, llamada "Memoria Gamma", se propuso red-27, pero su rendimiento en problemas que implican dependencias a largo plazo no parece ser mejor que el rendimiento de TDNNs.

En algunas arquitecturas, las constantes de tiempo determinan el factor de escala del error si se propaga de nuevo para un paso de tiempo en una sola unidad ref-3. Sin embargo, los intervalos de tiempo prolongado no se pueden procesar debido a que una constante de tiempo apropiada es casi imposible.

Horchreiter y Schmidhuber (1997) introdujeron las LSTM como solución al problemas

del devanecimiento del gradiente. La red LSTM se basa en el bloque de memoria, que se componse 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 errro 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 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 nerupna 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.

3.2 Características de la solución

La solución propóne un análisis práctico de la convergencia de las redes neuronales profundas mediante la aplicación de la regla de aprendizaje basada en el algoritmo *simulated annealing*. El análisis se realizará sobre conjuntos de datos de diferente indole, utilizados en otras investigaciones. Se comparará su desempeño frente a otras reglas de aprendizaje definidas en la literatura.

3.3 Propósitos de la solución

El propósito del presente trabajo es analizar la convergencia de las redes neuronales profundas, determinando el comportamiento de diferentes reglas de aprendizaje.

3.4 Alcances o limitaciones de la solución

Los alcances y limitaciones descritos para el trabajo son los siguientes

- El estudio se plantea desde una perspectiva práctica, precisando conjuntos de datos acotados.
- Los datos que se utilizarán son utilizados por Morse y Stanley (2016) en su publicación.
- Se estudiarán las reglas de aprendizaje definidas por el gradiente estocástico y *simulated annealing*.
- Las redes neuronales utilizarán la misma configuración para todos los experimentos salvo por la regla de aprendizaje.

4 Metodología, herramientas y ambiente de desarrollo

A continuación se especifica la metodología, herramientas y ambientes de desarrollo que se utilizarán para llebar a cabo el proyecto.

4.1 Metodología a usar

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: Los modelos lineales y no lineales que aprenden el comportamiento de la autorregulación del flujo sanguíneo cerebral se comportan distinto cuando se utilizan diferentes bandas de frecuencias.
- Marco teórico: Una revisión de la literatura donde se aborda el problema planteado, para situarse en el contexto actual de los problemas. Se describirán los modelos que permiten establecer relaciones entre las componentes de la autorregulación cerebral dinámica.
- Diseño de la solución: Se deberá diseñar el experimento para generar los modelos y preparar las señales para su evaluación. Diseñar y ejecutar el experimento provocando ruído en las señales utilizadas.
- Análisis y verificación de los resultados: Los resultados se analizarán considerando métodos estadísticos con un valor $p < 0{,}05$ que será considerado como diferencia significativa.

- 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.

4.2 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

El software que se utilizará es:

- Software: Entorno para computación y gráficos estadísticos R.
- Herramienta ofimática: LaTeX.

4.3 Ambiente de desarrollo

El desarrollo de la investigación se realizará en el Departamento de Ingeniería Informática de la Universidad de Santiago de Chile, el cual cuenta con una biblioteca y un laboratorio de computación con acceso a internet, para la recopilación de información y para el desarrollo del experimento. Además, se utilizará el hogar del autor, donde se ubica el equipo de desarrollo a utilizar.

5 Plan de trabajo

La planificación del trabajo de investigación, se indica en una carta Gantt que se puede apreciar en la Tabla 1, donde se fija como fecha de inicio el día 12 de marzo de 2016 con fecha de término el día 30 de Junio de 2016. En un regimen de trabajo que considera periodos de 5 horas diarias de trabajo promedio, los 7 días de la semana, lo que equivale a un total de 640 horas de trabajo.

Tabla 1: Carta Gantt

	TAREA	FECHA INICIO	FECHA TÉRMINO	DURACIÓN (DÍAS)
PROYECTO DE TESIS		07-03-16	24-06-16	–
Inicio		07-03-16	08-03-16	2
	Reunión con el profesor	07-03-16	07-03-16	1
	Inscripción del tema	08-03-16	08-03-16	1
Estado del arte		09-03-16	29-03-16	21
	Revisión bibliográfica	09-03-16	18-03-16	10
	Reunión con el profesor	19-03-16	19-03-16	1
	Marco teórico	20-03-16	29-03-16	10
Desarrollo de la solución		30-03-16	24-04-16	26
	Análisis	30-03-16	02-04-16	4
	Diseño	03-04-16	06-04-16	4
	Implementación	07-04-16	16-04-16	10
	Pruebas	17-04-16	19-04-16	3
	Correcciones	20-04-16	22-04-16	3
	Reunión con el profesor	23-04-16	23-04-16	1
	Redacción del análisis, diseño e implementación	24-04-16	24-04-16	1
Experimentos		25-04-16	25-05-16	31
	Calibración de parámetros	25-04-16	29-04-16	5
	Experimentación	30-04-16	14-05-16	15
	Reunión con el profesor	15-05-16	15-05-16	1
	Redacción de la experimentación	16-05-16	25-05-16	10
Análisis de resultados		26-05-16	08-06-16	14
	Análisis detallado	26-05-16	29-05-16	4
	Interpretación de los resultados	30-05-16	02-06-16	4
	Reunión con el profesor	03-06-16	03-06-16	1
	Redación de los resultados	04-06-16	08-06-16	5
Documento final		09-06-16	24-06-16	16
	Redacción de la conclusión	09-06-16	10-06-16	2
	Revisión del documento	11-06-16	17-06-16	7
	Presentación al profesor	18-06-16	18-06-16	1
Correcciones		19-06-16	22-06-16	4
	Versión final	23-06-16	23-06-16	1
	Entrega del documento	24-06-16	24-06-16	1

Bibliografía

Bengio, Y., y Frasconi, P. (1993). Credit assignment through time: Alternatives to backpropagation. En *Advances in neural information processing systems 6, [7th NIPS conference, denver, colorado, usa, 1993]* (pp. 75–82). Descargado de `http://papers.nips.cc/paper/724-credit-assignment-through-time-alternatives-to-backpropagation`

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

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

Fritsch, J. (1996). *Modular neural networks for speech recognition* (Masters Thesis). KIT.

Marvin Minsky, S. A. P. (1987). *Perceptrons: An introduction to computational geometry* (Expanded ed.). The MIT Press.

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

Puskorius, G. V., y Feldkamp, L. A. (1994, Mar). Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, *5*(2), 279-297. doi: 10.1109/72.279191

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. (1992a). A fixed size storage o(n3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, *4*, 243-248.

Schmidhuber, J. (1992b, mar). Learning complex, extended sequences using the principle of history compression. *Neural Comput.*, *4*(2), 234–242. Descargado de `http://dx.doi.org/10.1162/neco.1992.4.2.234` doi: 10.1162/neco.1992.4.2.234

Schmidhuber, J., y Hochreiter, S. (1996). *Guessing can outperform many long time lag algorithms* (Inf. Téc.).