# Optimización de Scaled Conjugate Gradient para Froog Neural Networks

Matías Roodschild[1], Jorge Gotay Sardiñas[1], Adrián E. Will[1], and Sebastián A. Rodriguez[1]

Facultad Regional Tucumán, Universidad Tecnológica Nacional,
Rivadavia 1050, San Miguel de Tucumán, Tucumán, Argentina.
{mroodschild, jgotay, awill, sebastian.rodriguez}@gitia.org
http://www.gitia.org

**Resumen** El Scaled Conjugate Gradient es un algoritmo de aprendizaje iterativo para redes neuronales artificiales, destacándose por su velocidad de convergencia frente al Backpropagation. Esto se debe a que utiliza derivadas de segundo orden, lo que permite una convergencia más veloz. En este artículo, se demuestra lo expuesto por Möller en su trabajo original, respecto a que una adecuada implementación del Scaled Conjugate Gradient puede reducir en un 50 % el tiempo de ejecución. La implementación fue realizada en la herramienta gratuita "Froog Neural Networks"de redes neuronales y probada en 4 base de datos estándar, con diferentes cantidades de neuronas para probar la efectividad de los cambios realizados.

**Keywords:** Scaled Conjugate Gradient, Froog Neural Networks

## 1. Introducción

En el año 1993, Möller introdujo un nuevo algoritmo de aprendizaje para redes neuronales artificiales llamado Scaled Conjugate Gradient (SCG) con tasa de convergencia superlineal [1]. En su artículo destaca el rendimiento del SCG frente al algoritmo Backpropagation (BP) [2], el Conjugate Gradient Backpropagation (CG) [3] y el algoritmo cuasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) [4]. Además, el algoritmo SCG, se caracteriza por no requerir parámetros de configuración (a diferencia de los mencionados) y tiene la virtud de evitar búsquedas lineales para encontrar el tamaño del paso o taza de aprendizaje, a diferencia del CG. Por otro lado, BP es un algoritmo que se caracteriza por utilizar información de primer orden de la función objetivo, haciendo que el proceso iterativo de búsqueda tome una gran cantidad de pasos. Es por esto que se originaron algoritmos que aprovechan la información de segundo orden para acelerar la velocidad de convergencia como el CG, el SCG y el BFGS. Los algoritmos de segundo orden, calculan reiteradas veces el gradiente en cada iteración, el paralelismo conseguido con las tecnologías actuales, como los CPU, GPU y Cloud Computing, hacen que el cálculo del gradiente no sea tan costoso y logren reducir el tiempo global de ejecución, haciéndolos ideales para grandes volúmenes de datos [5].

Se han presentado otras propuestas para acelerar el algoritmo BP, algunas simples como ajustar el learning rate y algunas más complejas como Momentum [6], RMSProp [7], Adam [8], Adagrad [9], y Nesterov [10]. Si bien estas soluciones mejoran considerablemente el tiempo de convergencia de BP, requieren de la configuración de varios hiperparámetros, para lo cual, es necesario conocer en profundidad los efectos de cada uno y de manera empírica seleccionar los mejores parámetros.

Finalmente, es importante destacar que el algoritmo Adam es uno de los más utilizados actualmente y puede acelerar el proceso de convergencia del algoritmo BP, logrando obtener resultados similares al algoritmo SCG con información de primer orden. Sin embargo, algunos trabajos han demostrado que el algoritmo Adam en determinados casos no logra una convergencia adecuada [11]. Una búsqueda en las herramientas más importantes del área como: Keras, Tensorflow, Torch, Caffe, y Deepl4J, el algoritmo SCG no se encuentra en las versiones oficiales hasta la fecha. En este trabajo, implementamos dos versiones del algoritmo SCG, la original sugerida por Möller y otra siguiendo las sugerencias de implementación planteadas en ese mismo trabajo. Los algoritmos fueron implementados en la herramienta gratuita que estamos desarrollando Froog.

Este trabajo se divide en las siguientes secciones: en la sección 2, se describen las estrategias de optimización en los algortimos BP y SCG, en la sección 3, se muestra el algoritmo SCG, sus modificaciones y el proceso de implementación del mismo, para la librería Froog NN, en la sección 4, se describen la configuración experimental, las bases de datos y los resultados de los ensayos que muestra la comparación del rendimiento entre los algoritmos SCG, su versión modificada y el BP. Por último en la sección 5 se exponen las conclusiones del trabajo.

## 2. Estrategia de Optimización

El entrenamiento de una red neuronal es, por lo general, un proceso iterativo que busca el mínimo de una función de error $E(\tilde{w})$. En este caso, $\tilde{w}$ es el vector compuesto por los parámetros de la red neuronal, que será ajustado por medio del vector opuesto del gradiente $\tilde{p}_k$ hasta obtener el error deseado. Podemos resumir los algoritmos de optimización de la siguiente manera:

1. Elegir el vector de parámetro inicial $\tilde{w}_1$ y $k = 1$.
2. Determinar el gradiente $\tilde{p}_k$ y un tamaño de paso $\alpha_k$ tal que $E(\tilde{w}_k + \alpha_k \tilde{p}_k) < E(\tilde{w}_k)$.
3. Actualizar el vector de parámetros: $\tilde{w}_{k+1} = \tilde{w}_k + \alpha_k \tilde{p}_k$.
4. Si $E'(\tilde{w}_k) \neq \tilde{0}$ entonces $k = k+1$ y continuar desde el paso 2. Caso contrario, $\tilde{w}_{k+1}$ es el mínimo deseado (donde $E'(\tilde{w})$ es gradiente de la función error).

Este proceso iterativo involucra dos pasos independientes. Primero se debe determinar una dirección de descenso y una vez encontrada tal dirección, $\alpha_k$ determina cuánto ha de moverse el algoritmo en dicha dirección. El algoritmo descripto es la estrategia utilizada por el BP en todas sus variantes y se vale del cálculo del gradiente para cumplir su cometido. El SCG a diferencia del BP, no

solo calcula el gradiente $\tilde{p}_k$, también explora el tamaño del paso máximo posible $\alpha_k$ en dicha dirección, garantizando una convergencia en menos iteraciones. Para calcular $\alpha_k$, el SCG requiere calcular un segundo gradiente $\tilde{r}_k$ ortogonal a $\tilde{p}_k$ y realizar una exploración lineal en la dirección del gradiente $\tilde{p}_k$.

## 3. Detalles de implementación del algoritmo SCG

En la sección anterior se describió brevemente el algoritmo SCG, si bien éste funciona correctamente, se puede mejorar su rendimiento si aprovechamos correctamente los cálculos de las activaciones, la derivada y la función de costo. Los siguientes diagramas de flujo muestran los cambios propuestos para lograr ahorrar hasta un 50 % el tiempo de ejecución.

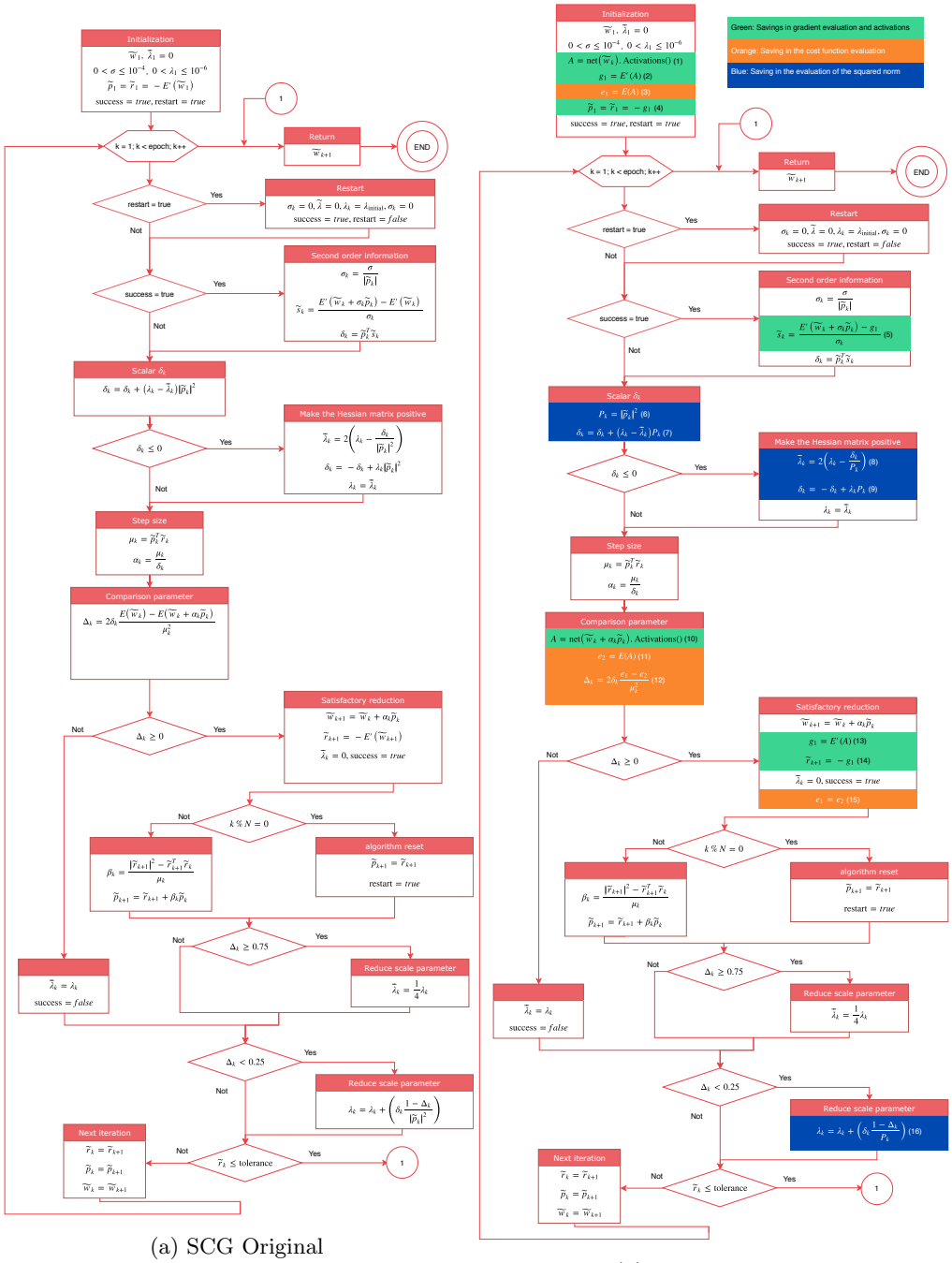
Como puede observarse en el diagrama de la Figura 1, en color verde se indica los pasos donde se ahorra cómputo por el cálculo de las activaciones de la red neuronal, realizados en (1) e influyendo en: (2), (4), (5); y las activaciones realizadas en (10) influirán en (13), (14) y en (5) a partir de la segunda iteración. En anaranjado, el cálculo del error de la red neuronal es calculado en (3) y (11), impactando en el rendimiento de (12) y (15), donde el error (15) solo será actualizado, si y solo si, $\Delta_k >= 0$, evitando cálculos innecesarios. Por último, en azul el cálculo la norma cuadrada del gradiente $|\tilde{p}_k|^2$ se realiza en (6) e impacta en (7), (8), (9) y (16).

En otras palabras, se cambió el orden de las operaciones al inicio del algoritmo y se hizo énfasis en calcular primero todo lo que puede ser reutilizable, para luego ser aprovechado por el cálculo del gradiente $g_1$, la función costo $e_1$ y la norma cuadrada del gradiente $|\tilde{p}_k|^2$, permitiendo hacer un mayor ahorro en el procesamiento del algoritmo, y permitir tener tiempos similares a dos veces el tiempo del BP por iteración, que es el tiempo óptimo para el SCG.

La herramienta utilizada durante los ensayos del algoritmo propuesto, es Froog Neural Network (Froog) [12], este software es Open Source y gratuito. Froog esta preparado, tanto para el diseño y entrenamiento de redes neuronales, como así también, funcionar en Java, R y Android. Froog fue creado por el Grupo de Investigación en Tecnologías Informáticas Avanzadas de la Facultad Regional de Tucumán - Universidad Tecnológica Nacional. Froog se caracteriza por estar implementado en su totalidad en Java con multihilos y tiene un desarrollo vectorizado gracias a la librería EJML (Efficient Java Matrix Library) [13]. Actualmente, Froog se encuentra en continuo desarrollo y es utilizado como proyecto educativo y de investigación en diferentes algoritmos de redes neuronales.

## 4. Configuración Experimental y Bases de Datos

Para los ensayos se seleccionaron 4 base de datos: **MNIST** [14], **Fashion-MNIST** [15], **Letters EMNIST** y **Balanced EMNIST** [16]. Estas bases de datos tienen las características de ser bien conocidas, de acceso público y tener una fácil interpretación de los resultados. Estas 4 bases de datos, constan de un número adecuado de ejemplos para ser utilizados como benchmarks, todas




(a) SCG Original

(b) SCG Propuesto

Figura 1: La figura (1a) representa el diagrama de flujo original propuesto por Möller y la figura (1b) representa el diagrama de flujo propuesto.

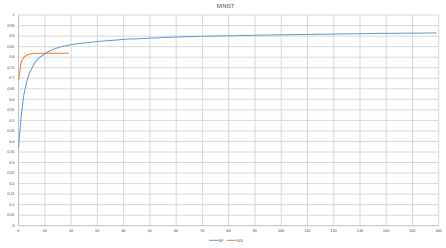
están en escala de grises y tienen una dimensión de 28 x 28 pixeles. MNIST es una de las bases de datos más empleadas en la bibliografía del área, y se caracteriza por constar de dígitos escritos a mano con números del 0 al 9 y posee de un total de 60.000 registros, separados en 50.000 para entrenamiento y 10.000 para testeo. La base de datos Fashion MNIST, consta de 10 clases de vestimentas, aumentando la complejidad en la clasificación de los objetos. Letters EMNIST, es una base de datos similar a MNIST, pero realizada con caracteres, sin distinción entre mayúsculas y minúsculas del diccionario anglosajón, de la "a" a la "z", 26 clases en total. Balanced MNIST, es una mezcla entre la base de datos de Letters y la de Dígitos, pero a su vez garantiza tener clases balanceadas, y agrega la dificultad de distinguir entre las letras minúsculas y mayúsculas que difieren entre sí, constando en un total de 47 clases. Para este ensayo se usó un procesador i7 6700K, la herramienta Froog empleó un solo hilo de procesamiento y para cada de base de datos, se utilizó el conjunto de entrenamiento completo y se evaluaron los resultados de la iteración 10 con el conjunto de testeo, para evaluar la eficiencia del algoritmo SCG propuesto frente al SCG original, también se incluye una comparación con el algoritmo BP para tener una referencia de eficiencia.

| | | BP *lr = 0.1* | | SCG Original | | SCG Propuesto | |
|---|---|---|---|---|---|---|---|
| BD | Neur | *Seg/iter* | Acc. Test | *Seg/iter* | Acc. Test | *Seg/iter* | Acc. Test |
| MNIST | 100 | 4.5 | 0.774 | 18.4 | 0.797 | **9.2** | 0.797 |
| | 200 | 8.9 | 0.791 | 36.7 | 0.819 | **17.9** | 0.819 |
| | 300 | 13.3 | 0.807 | 54.4 | 0.819 | **26.7** | 0.819 |
| | 400 | 17.7 | 0.815 | 72.1 | 0.823 | **35.5** | 0.823 |
| | 500 | 22.1 | 0.825 | 90.1 | 0.824 | **44.2** | 0.824 |
| Letters | 100 | 12.3 | 0.372 | 50.2 | 0.502 | **25.1** | 0.502 |
| | 200 | 25.6 | 0.400 | 100.1 | 0.528 | **48.4** | 0.528 |
| | 300 | 36.5 | 0.451 | 148.9 | 0.560 | **75.4** | 0.560 |
| | 400 | 50.1 | 0.484 | 198.1 | 0.564 | **96.4** | 0.564 |
| | 500 | 61.5 | 0.498 | 254.6 | 0.559 | **124.2** | 0.559 |
| Fashion | 100 | 6.3 | 0.737 | 24.4 | 0.661 | **12.3** | 0.661 |
| | 200 | 11.9 | 0.744 | 47.9 | 0.673 | **24.9** | 0.673 |
| | 300 | 17.7 | 0.756 | 73.9 | 0.670 | **36.6** | 0.670 |
| | 400 | 24.7 | 0.766 | 98.5 | 0.679 | **47.9** | 0.679 |
| | 500 | 29.9 | 0.765 | 126.5 | 0.688 | **58.8** | 0.688 |
| Balanced | 100 | 11.9 | 0.183 | 49.6 | 0.446 | **24.4** | 0.446 |
| | 200 | 23.1 | 0.250 | 96.0 | 0.504 | **46.2** | 0.504 |
| | 300 | 33.6 | 0.274 | 138.1 | 0.497 | **66.7** | 0.497 |
| | 400 | 45.3 | 0.306 | 185.2 | 0.490 | **88.6** | 0.490 |
| | 500 | 56.7 | 0.338 | 232.7 | 0.512 | **114.8** | 0.512 |

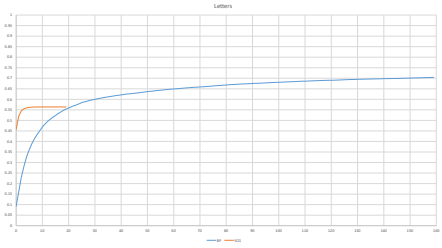
Cuadro 1: Se detalla el tiempo de ejecución por iteración, requerida por los diferentes algoritmos y el accuracy obtenido con los datos de testeo luego de 10 iteraciones.

En el siguiente ensayo, se comparó los algoritmos BP y SCG para evaluar la eficacia en la convergencia de cada uno. Se utilizaron nuevamente las 4 bases

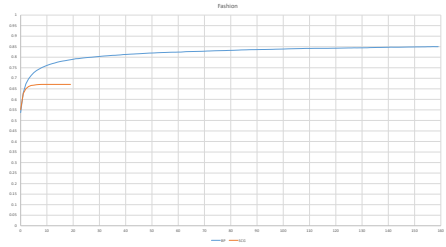
de datos, pero se limitó a 300 neuronas en la capa oculta para cada caso. En el BP, se utilizó un learning rate = 0.1 y se ejecutó por 160 iteraciones. El SCG se ejecutó por 20 iteraciones, donde alcanza su estabilización y se realizaron las comparaciones de cada algoritmo.



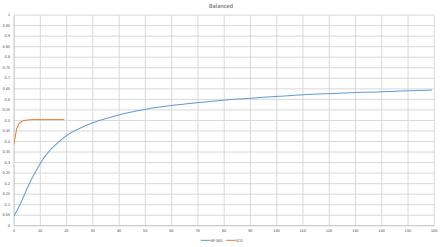
(a) MNIST



(b) Letters



(c) Fashion MNIST



(d) Balanced

Figura 2: Se muestra el accuracy de testeo de cada algoritmo, para cada iteración con las diferentes bases de datos. En Azul el BP y en Rojo el SCG.

## 5. Conclusiones

Como se puede observar en el Cuadro 1, los tiempos de ejecución del algoritmo SCG propuesto, reduce el tiempo a la mitad del algoritmo presentado originalmente, también se advierte, que es aproximadamente el doble del tiempo que el algoritmo de BP requiere por iteración, esto se debe a la doble evaluación del gradiente, que será compensada por el paralelismo del CPU, GPU u otro hardware. En cuanto a los resultados, se puede notar en la Figura 2, que el algoritmo de SCG puede realizar una mejor aproximación en las primeras iteraciones respecto al BP, pero luego de que el SCG se estabiliza, sería recomendable cambiarlo por el BP para terminar de realizar el entrenamiento de la red neuronal.

## Recursos

Códigos fuentes: https://github.com/mroodschild/Jaiio2019

## Referencias


1. Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.
2. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
3. E.M. Johansson, F.U. Dowla, and D.M. Goodman. Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. *Lawrence Livermore National Laborartoru*, 1990.
4. R. Battiti and F. Masulli. Bfgs optimization for faster automated supervised learning. *INCC 90 Paris, International Neural Network Conference*, pages 757–760, 1990.
5. Quoc V Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 265–272. Omnipress, 2011.
6. Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.
7. Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2545–2553. JMLR. org, 2017.
8. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
9. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
10. Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.
11. Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.
12. Matías Roodschild, Jorge Gotay Sardiñas, Adrián E. Will, and Sebastián A. Rodriguez. Froog neural network, 2019.
13. Peter Abeles. Efficent java matrix library, 2019.
14. Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
15. Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
16. Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.