Optimización para Redes Neuronales

Pamela Chirino[a], Mariela Galdámez[a], Germán Bianchini[a], Paola Caymes Scutari[a,b]

[a] Laboratorio de Investigación en Cómputo Paralelo/Distribuido (LICPaD)
Departamento de Ingeniería en Sistemas de Información, Facultad Regional Mendoza,
Universidad Tecnológica Nacional. (M5502AJE) Mendoza, Argentina
pamelaachirino@gmail.com, mariela_galdamez@hotmail.com, gbianchini@frm.utn.edu.ar

[b] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
pcaymesscutari@frm.utn.edu.ar

Resumen

Las Redes neuronales artificiales tratan de reproducir el proceso de solución de problemas del cerebro, una red neuronal toma como ejemplos problemas resueltos para poder construir un sistema de toma de decisiones y realizar calificaciones. Estas características de poder aprender de ejemplos o problemas resueltos las hacen una solución óptima cuando se trabaja con modelos de predicción.

El objetivo a futuro de esta investigación es poder aplicar redes neuronales en el modelo paralelo de predicción de incendios utilizado en el laboratorio LICPaD. En este modelo se trabaja constantemente con la incertidumbre de variables, lo que dificulta una predicción óptima. Al implementar redes neuronales en el modelo, se buscará que este logre una mejor toma de decisiones sobre las variables según qué peso tienen estas en el incendio a predecir.

Para poder aplicarlo a este modelo buscando mejorar el tiempo y calidad de resultados, se busca paralelizarlo de forma que tanto el aprendizaje como el funcionamiento de la red no afecten considerablemente el tiempo de obtención de los resultados.

1. Introducción

En general, los métodos de inteligencia artificial (IA) son una respuesta al deseo de aproximar el comportamiento y el pensamiento humano a diversos sistemas de solución de determinadas problemáticas. Por ello no es extraño ver que en la actualidad existen sistemas muy avanzados que pueden simular algunos de los comportamientos humanos [1].

Una red neuronal toma como ejemplo problemas resueltos para construir un sistema que toma decisiones y realiza clasificaciones. Los problemas adecuados para la solución neuronal son aquellos que no tienen una solución computacional precisa o que requieren algoritmos muy extensos. También son muy útiles para modelos de predicción debido a su forma de aprender que será detallada más adelante en este artículo [2].

Por los motivos mencionados anteriormente se pretende investigar si es posible la paralización de los algoritmos de aprendizaje de redes neuronales con el fin de poder aplicar estas redes de manera eficiente al modelo de predicción de incendios llevado a cabo en el LICPaD [2].

Para poder llevar a cabo este estudio se seleccionó la red neuronal Perceptrón, qué es uno de los

modelos más básicos para poder lograr su correcta comprensión y estudiar en que parte de las mismas sería razonable paralelizarla.

2. Redes neuronales artificiales

Las redes neuronales artificiales son un modelo computacional inspirado en el comportamiento observado en el cerebro humano. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo valores de salida. Con el fin de lograr un mayor entendimiento de una neurona artificial se la comparará con la neurona biológica, como se muestra en la Figura 1 [1] [3].

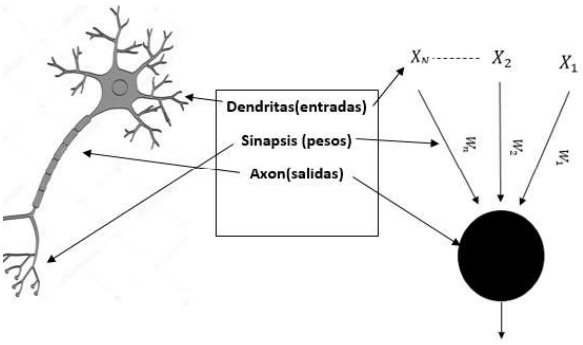

Figura 1: Grafica de comparación entre una neurona biológica y una artificial.

Una neurona biológica es una célula especializada en procesar información. Está compuesta por el cuerpo de la célula (soma) y dos tipos de ramificaciones: el axón y las dendritas. La neurona recibe las señales (impulsos) de otras neuronas a través de sus dendritas y transmite señales generadas por el cuerpo de la célula a través del axón.

Una de las características de las neuronas es su capacidad de comunicarse. En forma concreta, las dendritas y el cuerpo celular reciben señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transmite dichas señales a los terminales axónicos, los cuales distribuyen la información.

Para establecer una similitud directa entre la actividad sináptica y las redes neurales artificiales podemos considerar que las señales que llegan a la sinapsis son las entradas a la neurona artificial; éstas son ponderadas a través de un parámetro denominado peso, asociado a la sinapsis correspondiente. Estas señales de entrada pueden excitar a la neurona (sinapsis con peso positivo) o inhibirla (peso negativo).

3. Elementos básicos que componen una red neuronal

Una red neuronal artificial está constituida por neuronas interconectadas y arregladas en capas, como se muestra en la Figura 2. Los datos ingresan por medio de la "capa de entrada", pasan a través de la "capa oculta" y salen por la "capa de salida". Cabe mencionar que la capa oculta puede estar constituida por varias capas de neuronas [1] [4] [5].

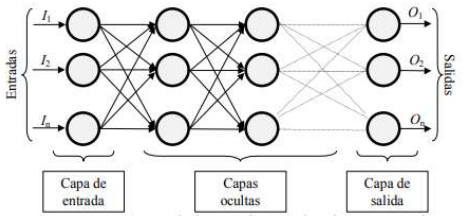

Figura 2: Esquema de una red neuronal

La capa de entrada recibe un input (datos de entrada) desde otras neuronas o de una fuente externa de datos. Cada input tiene un peso asociado *w,* que se va modificando en el proceso de aprendizaje el cual detallaremos en la sección 4.

3.1. Función de entrada (input function).

La neurona trata a muchos valores de entrada como si fueran uno solo; esto recibe el nombre de entrada global. Por lo tanto, ahora nos enfrentamos al problema de cómo se pueden combinar estas simples entradas ($in_{i1}, in_{i2}, \ldots in_{in}$) dentro de la entrada global, g_{ini}. Esto se logra a través de la

función de entrada, la cual se calcula a partir del vector entrada. La función de entrada puede describirse como sigue:

$$input_i = (in_{i1} \bullet w_{i1}) * (in_{i2} \bullet w_{i2}) * \ldots (in_{in} \bullet w_{in})$$

donde: * representa al operador apropiado, n al número de entradas a la neurona n_iy w_i al peso.

Los valores de entrada se multiplican por los pesos anteriormente ingresados a la neurona. Estos pesos de los valores de entrada cambian los pesos de las neuronas haciendo que cambie también la influencia de las mismas, es decir, una neurona con mayor peso tendrá mayor relevancia en el problema que una con un peso inferior a esta.

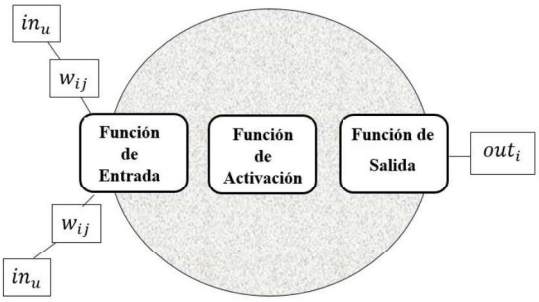

Figura 3: Diagrama los elementos de una neurona.

3.2. Funciones de activación

La función de activación tiene como objetivo acotar los valores de salida de una red neuronal para mantenerlos en ciertos rangos, es decir, calcula el estado de actividad de una neurona; transformando la entrada global en un valor (estado) de activación. Las neuronas artificiales, como las biológicas, tienen diferentes estados de activación, algunas de ellas solamente dos, pero otras pueden tomar cualquier valor dentro de un conjunto determinado.

Las funciones de activación más utilizadas se detallan a continuación:

1. Función lineal

La función lineal o identidad responde a la expresión F(u), es decir, esta función acota los valores obtenidos para mantenerlos en cierto rango. En este caso la función es binaria, es decir, devuelve dos posibles valores de activación 1(neurona activada o excitada) y 0 (neurona no activada).

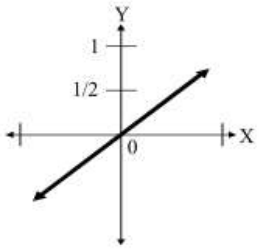

Figura 4: Grafica de la función lineal.

2. Función sigmoidal

En la función sigmoidal el valor dado por la función es cercano a uno de los valores asintóticos. Esto hace que, en la mayoría de los casos, el valor de salida esté comprendido en la zona alta o baja del sigmoide.

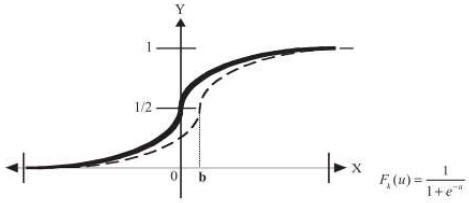

Figura 5: Grafica de la función sigmoidal.

3.3. Función de salida (*output function*).

El último componente que una neurona necesita es la función de salida. El valor resultante de esta función es la salida de la neurona i (out_i); por ende, la función de salida determina que valor se transfiere a las neuronas vinculadas. Si la función de activación está por debajo de un umbral determinado, ninguna salida se pasa a la neurona subsiguiente.

4. Aprendizaje de las redes neurales

Una red neuronal debe aprender a calcular la salida correcta para cada arreglo o vector de entrada en el conjunto de ejemplos. Este proceso de aprendizaje se denomina proceso de entrenamiento o acondicionamiento de la red neuronal. El conjunto de datos (o conjunto de ejemplos) sobre el cual este proceso se basa es llamado conjunto de datos de entrenamiento.

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el mismo se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma manera, una conexión se destruye cuando su peso pasa a ser cero. Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por lo tanto, se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables.

La capacidad de aprendizaje adaptativo es una de las características más atractivas de las redes neuronales. Significa que aprenden a llevar a cabo ciertas tareas mediante un entrenamiento con ejemplos ilustrativos. Como las redes neuronales pueden aprender a diferenciar patrones mediante ejemplos y entrenamientos, no es necesario elaborar modelos ni especificar funciones de distribución o de probabilidad.

Las redes neuronales son sistemas dinámicos autoadaptativos; debido a la capacidad de autoajuste de los elementos procesales (neuronas) que componen el sistema. Son dinámicos, porque son capaces de estar constantemente cambiando para adaptarse a las nuevas condiciones.

Una red neuronal no necesita un algoritmo para resolver un problema, ya que ella puede generar su propia distribución de pesos en los enlaces mediante el aprendizaje. También existen redes que continúan aprendiendo a lo largo de su vida, después de completado su período de entrenamiento.

Las redes neuronales fueron los primeros métodos computacionales con la capacidad de tolerancia a fallos. Comparados con los sistemas computacionales tradicionales, los cuales pierden su funcionalidad cuando sufren un pequeño error de memoria, en las redes neuronales, si se produce un fallo en un número no muy grande de neuronas y aunque el comportamiento del sistema se ve influenciado, no sufre una caída repentina.

Hay dos aspectos distintos respecto a la tolerancia a fallos:

a) Las redes pueden aprender a reconocer patrones con ruido, distorsionados o incompletos. Esta es una tolerancia a fallos respecto a los datos.

b) Las redes pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

La razón por la que las redes neuronales son tolerantes a los fallos es que tienen su información distribuida en las conexiones entre neuronas.

En las redes neuronales hay dos métodos de aprendizaje importantes que pueden distinguirse:

a) Aprendizaje supervisado.
b) Aprendizaje no supervisado.

Otro criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en considerar si la red puede aprender durante su funcionamiento habitual o si el aprendizaje supone la desconexión de la red. En el primer caso, se trataría de un aprendizaje *on line*, mientras que el segundo es lo que se conoce como *off line*. Cuando el aprendizaje es *off line*, se distingue entre una fase de aprendizaje o entrenamiento y una fase de operación o funcionamiento [4] [5] [6].

4.1 Aprendizaje supervisado.

Se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo que podría ser el programador (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada. En este tipo de aprendizaje se suelen considerar, a su vez, tres formas de llevarlo a cabo: Aprendizaje por corrección de error, Aprendizaje por refuerzo y Aprendizaje estocástico. De estas tres formas de aprendizaje supervisado, nos es de interés el Aprendizaje por corrección de error, el cual detallaremos a continuación.

Es de nuestro interés porque en esta investigación nos centraremos en el estudio del perceptrón el cual lo utiliza.

4.1.1. Aprendizaje por corrección de error

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red, es decir, en función del error cometido en la salida. Un ejemplo de esto es el aprendizaje del perceptrón (sección 6) que para cada neurona en la capa de salida se le calcula la desviación a la salida objetivo como el error, δ. El cual luego se utiliza para cambiar los pesos sobre la conexión de la neurona precedente.

Otro ejemplo importante de esta forma de aprendizaje es la regla de aprendizaje de propagación hacia atrás o de *backpropagation,* también conocido como Regla delta generalizada. Estos mismos serán ampliados más adelante en las secciones 5 y 6.

4.2. Aprendizaje no supervisado.

Las redes con aprendizaje no supervisado no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada. En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada y las informaciones que se le han mostrado en el pasado. En otro caso, podría realizar una *clusterización (clustering)* o establecimiento de categorías, indicando la red a la salida a qué categoría pertenece la información presentada a la entrada, siendo la propia red quien debe encontrar las categorías apropiadas a partir de las correlaciones entre las informaciones presentadas. En cuanto a los algoritmos de aprendizaje no supervisado, en general se suelen considerar dos tipos: Aprendizaje hebbiano y Aprendizaje competitivo y comparativo. Estos aprendizajes exceden a esta investigación por el momento.

4.3. Elección del conjunto inicial de pesos

Antes de comenzar el proceso de entrenamiento se debe determinar un estado inicial, lo que significa: escoger un conjunto inicial de pesos para las diversas conexiones entre las neuronas de la red neuronal. Esto puede realizarse por varios criterios; por ejemplo, uno de ellos es otorgar un peso aleatorio a cada conexión, encontrándose los mismos dentro de un cierto intervalo. Generalmente un intervalo del tipo **[-n, n]**, donde **n** es un número natural positivo. Cabe mencionar que durante el transcurso del entrenamiento los pesos no se encuentran restringidos a dicho intervalo.

4.4. Detención del proceso de aprendizaje.

Para determinar cuándo se detendrá el proceso de aprendizaje, es necesario establecer una condición de detención. Normalmente el entrenamiento se detiene cuando el cálculo del error cuadrado sobre todos los ejemplos de entrenamiento ha alcanzado

un mínimo o cuando para cada uno de los ejemplos dados, el error observado está por debajo de un determinado umbral. Otra condición de detención del aprendizaje puede ser cuando un cierto número de ciclos y/o pasos de entrenamiento hayan sido completados.

5. Redes de retropropagación (backpropagation)

Es un método de aprendizaje supervisado con descenso del gradiente. En las redes de retropropagación primero se aplica un patrón de entrada, el cual se propaga por las distintas capas que componen la red hasta producir una salida de la misma. Esta salida se compara con la salida deseada y se calcula el error cometido por cada neurona de salida. Estos errores se transmiten hacia atrás, partiendo de la capa de salida hacia todas las neuronas de las capas intermedias. Cada neurona recibe un error que es proporcional a su contribución sobre el error total de la red. Basándose en este error recibido se ajustan los pesos sinápticos de cada neurona [1].

5.1 Descenso del gradiente

El descenso del gradiente se presenta en las redes con la característica de retropropagación para ayudar a minimizar el error de las predicciones de la Red neuronal.

La idea detrás de este método es encontrar el punto mínimo de la función de forma iterativa, debido a que se requieren múltiples pasos para llegar al punto de convergencia. Por lo general es utilizado en funciones de múltiples variables de las cuales es muy difícil analíticamente encontrar su mínimo.

Las redes neuronales inicializan sus pesos con valores aleatorios, por lo cual sus predicciones en un principio estarán muy alejadas de su valor real, esta distancia entre el valor real y la salida se ira acortando con cada iteración en donde se modifican los pesos de las neuronas. El descenso del gradiente ayuda a converger a la función a su valor óptimo ayudando así a reducir el error en la predicción de la red neuronal [1] [5].

5.2 Principios para entrenar una red multicapa empleando el algoritmo de retropropagación

Si se considera la red de tres capas con dos entradas (x_1, x_2) y una salida (y), como en la figura 6, es posible apreciar que cada neurona está compuesta de dos unidades, donde la primera suma los productos de las entradas por sus respectivos pesos, y la segunda unidad contiene la función de activación.

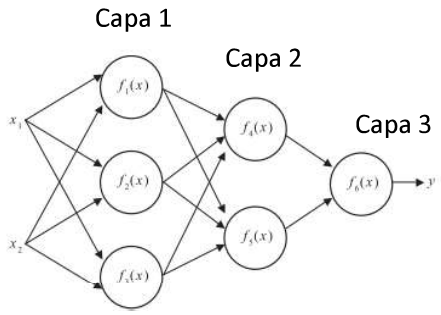

Figura 6:Red neuronal de tres capas.

Para enseñarle a la red neuronal es necesario entrenar un conjunto de datos, el cual consta de señales de entradas x_1 y x_2 asignadas a una denominada "salida deseada" a la cual llamaremos z. En el entrenamiento la salida deseada representa la salida esperada para cierto patrón de entrada, es decir, pasamos a la red un conjunto de entradas con la salida que debería producir ese conjunto.

El entrenamiento es un proceso iterativo. En cada iteración los pesos de los nodos se modifican usando nuevos datos del conjunto para el entrenamiento. Las modificaciones de los pesos se calculan empleando la secuencia de pasos que se explicará a continuación.

La salida de la red (y) es comparada con la salida deseada(z). La diferencia entre la salida de la red y la salida deseada se denomina error de la señal (**δ**). El algoritmo de retropropagación propaga el error de regreso a todas las neuronas, cuya salida fue la entrada de la última neurona. Luego el error se va propagando a las neuronas de capas anteriores, considerando los pesos de las conexiones, como se aprecia en la figura 7.

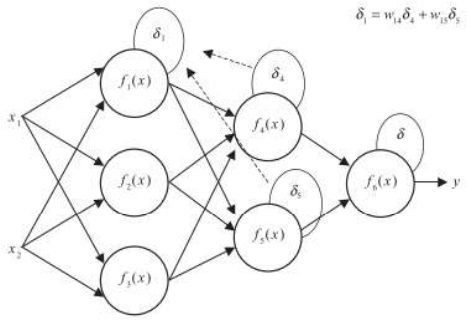

Figura 7: Retropropagación del error

Se considera que la red ha aprendido cuando el error es 0 o un margen próximo al mismo [5] [6].

6. Perceptrón

El perceptrón es la red neuronal más básica que existe de aprendizaje supervisado. El funcionamiento del perceptrón es muy sencillo, simplemente lee los valores de entrada, suma todas las entradas de acuerdo a unos pesos y el resultado lo introduce en una función de activación que genera el resultado final. En la Figura 9 se puede observar un esquema del percetrón.

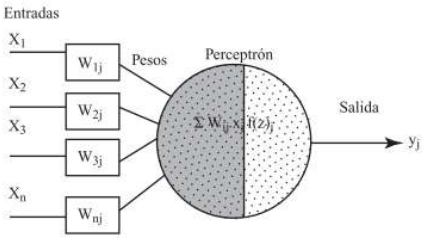

Figura 9:Esquema del perceptrón.

El perceptrón es un tipo de red neuronal que aplica los conceptos vistos en la sección 5(Redes de retropropagación).

Dado que al principio de esta investigación se contaba con poca experiencia en redes neuronales, se decidió comenzar a estudiar el perceptrón multicapa. También se tuvo en cuenta que en el LiCPaD se trabajaba con lenguaje C/C++ y que una de las pocas redes neuronales que se pueden implementar en este lenguaje es el perceptrón, debido a que es más sencillo de implementar cuando no se cuenta con clases.

Luego de determinar que se utilizaría el perceptrón para experimentar, se estudió cómo funcionaba y las formas de pasaje de parámetros, o valores de entrada que podía tener el mismo. Las formas para pasarle parámetros al perceptrón con el que nos encontramos trabajando es a través de archivos o incluyendo el valor de las variables en el mismo código. Dado que se pretende pasarle una gran cantidad de variables para poder aplicarlo a la predicción de incendios decimos trabajar con el pasaje de entradas por archivo para mantener un buen orden y simpleza en el código.

Entre los aspectos a destacar en la experimentación, se descubrió que para obtener simpleza en el código es conveniente trabajar con leguajes que permitan la orientación a objetos para poder separar por clases las neuronas y las capas de la red neuronal.

Para poder pasarle los datos de entrada por archivo a la red neuronal se necesita de un algoritmo que realice la carga. También en el mismo archivo se le puede pasar la topología de la red, el factor de error admitido y el número de iteraciones máximas que le permitimos a la red.

El número de iteraciones máximas viene dado por la retropropagación de error. Cada vez que se calcula el error y vuelve hacia atrás llevando el error a cada neurona se cuenta como una iteración. Actualmente se está analizando el tema de

paralelizar estas iteraciones separando en capas de neuronas.

El factor de error admitido es indicarle a la red que margen de error le permitimos con respecto a los resultados esperados.

Indicarle estos factores en el archivo (topología, factor de error admitido y número de iteraciones máximas) simplifica el código del perceptrón pero alarga, dificulta y demora el código que se encarga de cargar el archivo, dado que hay que crear un caso, no solo para los datos de entrada, sino también para cada uno de estos factores. Actualmente se está estudiando cuál es la forma más eficiente de pasaje y programación de estos factores.

En cuanto a librerías para redes neuronales se encontró que hay mayor disponibilidad y documentación de las mismas para lenguajes como Python. En el lenguaje C/C++ no se cuenta con mucha documentación de las librerías necesarias, pero esto no es un problema dado que el fin de implementar estas librerías sería para simplificar el código o conseguir modelos más gráficos del mismo pero no impide la realización de la red neuronal en C/C++ [4] [5] [6].

7. Paralelismo en Redes Neuronales

Una vez explicadas las bases de las redes neuronales, podemos explicar cómo se planea aplicar la programación paralela a una red neuronal para realizar un aprendizaje eficiente a medida que ésta se vuelve más compleja. Con esto nos referimos a una mejora apreciable en el tiempo de ejecución para la cantidad de recursos que se utilizan, en nuestro caso procesadores.

A continuación se explicarán brevemente algunos conceptos del paradigma paralelo, los cuales serán utilizados más adelante para describir su implementación en una red neuronal.

1. Broadcast. Consiste en enviar el mismo mensaje (dato) a más de un proceso destino [7]. En la figura 10 se muestra un esquema del funcionamiento; cabe aclarar que la acción broadcast requiere sincronismo: no se realiza hasta que todos los procesos hayan ejecutado su rutina `bcast()`.

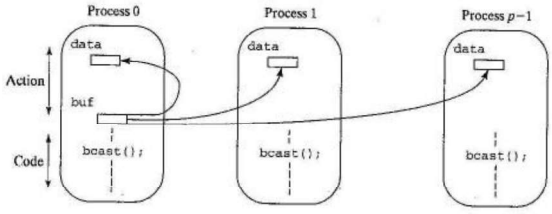

Figura 10. Esquema de comunicación broadcast.

2. Gather. Consta de un proceso principal que recolecta valores individuales desde un conjunto de procesos [7]. En la figura 11 es graficado el procedimiento donde el dato de un proceso i es recibido por el proceso principal y es almacenado en la posición i de un arreglo, preparado para recibir los datos.

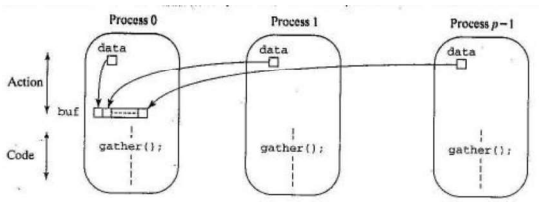

Figura 11. Esquema de recolección de datos mediante `gather`.

3. Barrier. Este término se refiere a un mecanismo de regulación que obliga a cada proceso a esperar hasta que todos los demás hayan llegado a un punto para continuar, indicado este previamente en el programa [7]. En la figura 12 se ilustra el concepto de barrier, donde la flecha señala el último barrier que se debe alcanzar para que todos los procesos continúen su ejecución.

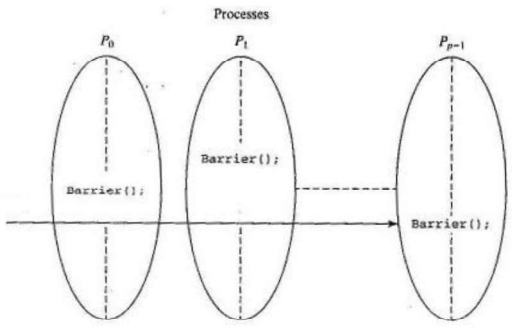

Figura 12. Los procesos esperan hasta que todos alcanzan su llamada `barrier()`.

4. Master – Worker. El proceso principal denominado Master es el proceso encargado de coordinar todo el tratamiento y procesamiento del problema, para lo que genera muchos subprocesos, que son ejecutados como procesos independientes denominados Workers, y en general se ejecutan en procesadores independientes. La interacción que existe entre ellos es que el Master inicia los procesos Worker, les asigna el trabajo a realizar, y estos devuelven el resultado al proceso Master [8]. En la figura 13 se encuentra graficado este concepto.

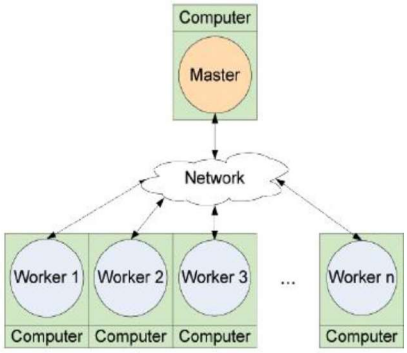

Figura 13. Master-Worker implementado sobre un Cluster de computadoras.

Como se detalló en las secciones anteriores, una red neuronal está compuesta por una capa de entrada, capas ocultas y una capa de salida, esquematizado en la figura 2. Recordamos esto ya que es importante para explicar cada paso de nuestra propuesta hacia la paralelización.

Primero, se propone determinar al Master como el encargado de almacenar datos y de realizar el envío y recepción de estos. Este, además, realizará los cálculos en la capa de entrada, para transformar los datos de entrada en valores utilizables por la red neuronal.

En segundo lugar, proponemos considerar a cada capa oculta y a la capa de salida como instancias donde una o más neuronas, contenidas en las mismas, serán representadas por un proceso. El Master distribuirá la cantidad de neuronas de cada capa -n- por procesador -p-, realizando la división n/p. De esta forma, cada procesador -Worker- quedaría a cargo de cierta cantidad de neuronas y realizaría diversos cálculos -explicados anteriormente- en cada neurona que le fuera asignada obteniendo una salida.

Luego de ingresar los datos de entrada en la capa de entrada, se obtendría sus salidas que serán enviadas a cada proceso Worker utilizando la rutina broadcast. Se propone realizar esta sucesión de pasos ya que las salidas de la capa de entrada serán las entradas de cada neurona de la primera capa oculta, y como el dato es el mismo, se utilizará el broadcast para enviar estos datos en un solo instante. Una vez que cada proceso worker haya recibido correctamente los datos, calculará el valor de salida de cada neurona que tiene a cargo. Estos datos serán enviados al Master con la rutina *gather* y finalmente entrará en un estado de espera con la rutina barrier hasta que todos los procesos hayan completado su envío. Esto se debe a que se necesitan las salidas de cada neurona para pasar a la capa siguiente, por esto mismo, consideraremos cada capa como una barrera -barrier-, como se muestra en la figura 14.

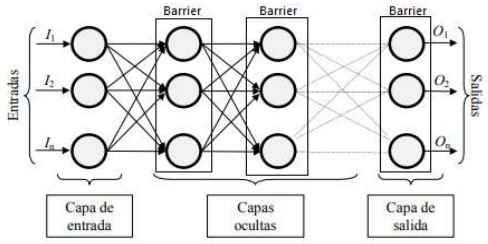

Figura 14. Esquema de aplicación de barrier en una red neuronal.

En el momento en que todos los procesos hayan alcanzado la rutina barrier, se procederá a realizar lo anterior hasta llegar a la última capa, es decir, la capa de salida.

El procedimiento explicado anteriormente representa el primer ajuste de pesos y cálculo de salidas de las neuronas de las capas. Sin embargo, no se ha propuesto aún la paralelización de su aprendizaje a través del *backpropagation*.

El algoritmo de *backpropagation* ajusta los pesos de las entradas de cada neurona hasta que el error de salida, comparado con la salida esperada, sea mínimamente aceptable (como se explica en la Sección 5.2). Por lo tanto, primero se calcula el error de cada salida de las neuronas en la capa de salida y luego, estos se transmitirán a través de la función *broadcast* a cada procesador. Recordemos que cada proceso se sigue manteniendo a cargo de las neuronas asignadas por el Master al principio del algoritmo.

Una vez recibido los errores, cada neurona procederá a realizar sus ajustes de pesos y el cálculo de su error de salida. Este procedimiento se repetirá en cada capa, excluyendo la capa de entrada.

Finalmente, se realizará nuevamente el cálculo de las salidas con los ajustes de pesos aplicados y en el caso que el error no sea aceptable, volverá a aplicar el algoritmo *backpropagation* cuantas veces sea necesario.

Conclusiones

Este trabajo menciona los conceptos estudiados para entender el funcionamiento de una red neuronal y los conceptos que se aplicarán en esta para lograr paralelizar su aprendizaje. Actualmente, se sigue estudiando el algoritmo de la construcción y el aprendizaje de una red neuronal para realizar la implementación y la prueba práctica de los conceptos mencionados al paralelizarla. Nuestro objetivo es aplicar el algoritmo de *backpropagation* paralelizado y estudiar si la eficiencia es aceptable para la cantidad de procesadores utilizados.

Se buscará también mejorar la granularidad y la asignación de tareas del programa, es decir, cómo se divide el programa en los procesadores para aumentar la eficiencia del mismo ya que los conceptos presentados darían una mejora lineal. Se estudiarán funciones para el traspaso de mensajes y sincronización de las partes de un programa más eficientes que brinden, al menos, una mejora logarítmica al paralelizar. Esta eficiencia puede mejorar a costa de mayor complejidad para implementarla,

Referencias

[1] Pedro Ponce Cruz, “Inteligencia Artificial Aplicada a la ingeniería”, Alfaomega, 2001.

[2] Germán Bianchini. Paola Caymes Scutari, Miguel Méndez Garabetti, Evolutionary-Statistical System: a Parallel Method for Improving Forest Fire Spread Prediction. Journal of Computational Science (JOCS) Vol 6 pp. 58-66. ISSN: 1877-7503 doi: 10.1016/j.jocs.2014.12.001 Elsevier.

[3] Carlos Alberto Ruiz, “Redes Neuronales: Conceptos básicos y aplicaciones”, UTN-Facultad Regional de Rosario, 2005.

[4]http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf, accedida el 20/08/19.

[5] Rivera E., “Entrenamiento de redes neuronales en algoritmos evolutivos”, 2005.

[6] Matich, D. J., “Introducción a las Redes Neuronales Artificiales”,2001.

[7] Barry Wilkinson, Michael Allen. “Parallel Programming” (2005). Pearson.

[8] http://charm.cs.uiuc.edu/research/masterSlave. Último acceso: Septiembre 2019.