

GENERACIÓN AUTOMÁTICA DE REDES NEURONALES CON AJUSTE DE PARÁMETROS BASADO EN ALGORITMOS GENÉTICOS

Fiszelew, A.¹ & García-Martínez, R.²

1. Investigador Asistente del Laboratorio de Sistemas Inteligentes. Facultad de Ingeniería. Universidad de Buenos Aires.
2. Director del Centro de Ingeniería del Software e Ingeniería del Conocimiento. Instituto Tecnológico de Buenos Aires.

Resumen

Este trabajo trata con métodos para encontrar arquitecturas óptimas de red neuronal para aprender problemas particulares. Se utiliza un algoritmo genético para encontrar las arquitecturas adecuadas, este algoritmo evolutivo emplea codificación directa y usa el error de la red entrenada como medida de desempeño para guiar la evolución. El entrenamiento de la red se lleva a cabo mediante el algoritmo de retro-propagación o back-propagation; se aplican técnicas como la repetición del entrenamiento, la detención temprana y la regularización de la complejidad para mejorar los resultados del proceso evolutivo. El criterio de evaluación se basa en las habilidades tanto de aprendizaje como de generalización de las arquitecturas generadas específicas de un dominio. Los resultados de la clasificación también se comparan con arquitecturas halladas por otros métodos.

Palabras clave: computación evolutiva, redes neuronales, algoritmos genéticos, métodos de codificación.

1. Introducción

La inteligencia artificial ha sido definida como la forma de diseñar procesos que exhiben características que comúnmente se asocian con el comportamiento humano inteligente [García Martínez, 1997, García-Martínez & Borrajo, 2000]. Sus enfoques abordan el modelado, con base en distintas arquitecturas, de distintos procesos propios del pensamiento humano tales como: la toma de decisiones, el razonamiento o el aprendizaje. Una de las arquitecturas que ha surgido para emular el comportamiento del aprendizaje es la red neuronal, que toma como modelo el cerebro humano [Rich y Knight, 1991].

Las redes neuronales artificiales ofrecen un paradigma atractivo para el diseño y el análisis de sistemas adaptativos inteligentes para un amplio rango de aplicaciones en inteligencia artificial por muchas razones incluyendo: flexibilidad para la adaptación y el aprendizaje

(mediante la modificación de las estructuras computacionales empleadas), robustez ante la presencia de ruido (datos erróneos o incompletos), habilidad para generalizar, capacidad de recuperación ante fallas, potencial para su computación masiva paralela, y semejanza (si bien superficial) con las redes neuronales biológicas [Hinton, 1989; Hertz, Krogh y Palmer, 1991].

A pesar de la gran actividad e investigación en esta área durante los últimos años, que ha llevado al descubrimiento de varios resultados teóricos y empíricos significativos, el diseño de las redes neuronales artificiales para aplicaciones específicas bajo un conjunto dado de restricciones de diseño (por ejemplo, aquellas dictadas por la tecnología) es, por mucho, un proceso de prueba y error, dependiendo principalmente de la experiencia previa con aplicaciones similares [Dow y Sietma, 1991]. La performance (y el costo) de una red neuronal sobre problemas particulares es críticamente dependiente, entre otras cosas, de la elección de los elementos de procesamiento (neuronas), la arquitectura de la red y el algoritmo de aprendizaje utilizado. Por ejemplo, muchos de los algoritmos de aprendizaje utilizados en las redes neuronales esencialmente buscan un ajuste adecuado de los parámetros modificables (también llamados pesos) dentro de una topología de red especificada a priori, bajo la guía de muestras de entrada (ejemplos de entrenamiento) provenientes del ambiente de la tarea. Claramente, para que este enfoque sea exitoso, el ajuste deseado de parámetros debe de hecho existir en el espacio donde se busca (el cual a su vez está restringido por la elección de la topología de red) y el algoritmo de búsqueda usado debe ser capaz de encontrarlo [Honavar y Uhr, 1993]. Aún cuando se pueda encontrar un ajuste adecuado de parámetros usando tal enfoque, la habilidad de la red resultante para generalizar sobre datos no vistos durante el aprendizaje, o el costo de usar la red (medido por su tamaño, consumo de energía, implementación en hardware, etc.) pueden estar lejos del óptimo. Estos factores tornan difícil al proceso de diseño de las redes neuronales. Adicionalmente, la falta de principios de diseño constituye un obstáculo de importancia en el desarrollo de sistemas de redes neuronales a gran escala para una amplia variedad de problemas prácticos. Por consiguiente, son de gran interés las técnicas para automatizar el diseño de arquitecturas neuronales para clases particulares de problemas bajo distintas restricciones de diseño y performance.

Este trabajo se centra en el desarrollo de métodos para el diseño *evolutivo* de arquitecturas de redes neuronales artificiales. Las redes neuronales son vistas comúnmente como un método de implementar mapeos no-lineales complejos (funciones) usando unidades elementales simples que se conectan entre sí mediante conexiones con pesos adaptables [Yao, 1999, Yao y Liu 1998]. Nos concentraremos en la optimización de las estructuras de conexión de las redes.

Los algoritmos evolutivos, inspirados en los principios de la evolución biológica, son otro paradigma en inteligencia artificial que ha recibido mucha atención últimamente. Estos algoritmos, que emplean modelos idealizados del código genético, la recombinación de información genética, las mutaciones y la selección, han producido formas muy genéricas y robustas de encontrar soluciones para problemas de búsqueda computacionalmente difíciles. Uno de esos problemas es la adaptación (o entrenamiento) de las arquitecturas y parámetros de las redes neuronales.

Los algoritmos genéticos, que constituyen una de las formas más importantes de los algoritmos evolutivos, son herramientas de optimización muy poderosas [Goldberg, 1991]. Al

igual que las redes neuronales, los algoritmos genéticos están inspirados en la biología: se basan en la teoría de la evolución genética y en el concepto de la supervivencia del más apto [Holland, 1975; 1980; 1986; Holland *et al.*, 1986]. Entre las características más importantes se destacan su naturaleza estocástica, su capacidad de considerar simultáneamente una población de soluciones, y su adaptabilidad ante un rango amplio de problemas [Wolpert y Macready, 1995].

El entrenamiento de redes neuronales usando algoritmos evolutivos ha sido un tema de interés durante los últimos años, desde distintos puntos de vista. Uno de éstos es el de la inteligencia artificial, donde queremos dejar que las computadoras resuelvan problemas y aprendan cosas por sí mismas sin involucrar a un programador humano. Otro campo cercano, el reconocimiento de patrones estadístico, le ha dado al aprendizaje neuronal muchas herramientas matemáticas fuertemente conceptuales para el análisis de diferentes enfoques. Aunque se han desarrollado métodos bastante eficientes para el entrenamiento de los pesos de las conexiones, no existe un método definitivo para determinar las arquitecturas neuronales más apropiadas para problemas particulares.

Los métodos en que estamos interesados son aquellos que se utilizan específicamente en el diseño arquitectónico de redes. Queremos hacer el entrenamiento de los pesos de la red mediante métodos de aprendizaje neuronal tradicional (no evolutivo), ya que han demostrado ser muy eficientes para ese propósito. El algoritmo evolutivo utiliza el error de la red entrenada como medida de desempeño para guiar la evolución.

El objetivo de este trabajo es combinar dos técnicas de inteligencia artificial, los algoritmos genéticos y las redes neuronales, para crear un sistema híbrido, donde usamos la primera para mejorar la segunda. El algoritmo genético se aplica a la estructura de una red neuronal con el propósito de encontrar las topologías óptimas específicas del dominio que faciliten tanto el aprendizaje como la generalización.

2. Diseño evolutivo de arquitecturas neuronales

El diseño de una arquitectura óptima (o cuasi-óptima) para una red neuronal puede verse como un problema de búsqueda en el espacio de arquitecturas donde cada punto representa una arquitectura distinta. Dado un criterio de performance acerca de las arquitecturas, como por ejemplo el error de entrenamiento más bajo, la complejidad de red más baja, etc., el nivel de performance de todas las arquitecturas formará una superficie discreta en el espacio. Obtener el diseño de arquitectura óptimo es equivalente a encontrar el punto más alto sobre esta superficie. Hay muchas características sobre dicha superficie que hacen que los AG sean buenos candidatos para la búsqueda, como por ejemplo [Yao, 1999]:

- La superficie es infinitamente grande ya que el número de nodos y conexiones posibles es ilimitado.
- La superficie es no diferenciable porque los cambios en el número de nodos o conexiones son discretos y pueden tener un efecto discontinuo sobre la performance de las redes neuronales.

- La superficie es compleja y ruidosa puesto que el mapeo desde una arquitectura hacia su performance es indirecto y dependiente del método de evaluación utilizado.
- La superficie es engañosa ya que arquitecturas similares pueden tener una performance bastante diferente.
- La superficie es multimodal porque diferentes arquitecturas pueden tener performances similares.

El proceso clave en el enfoque evolutivo del diseño de topologías se ve en la figura 1.

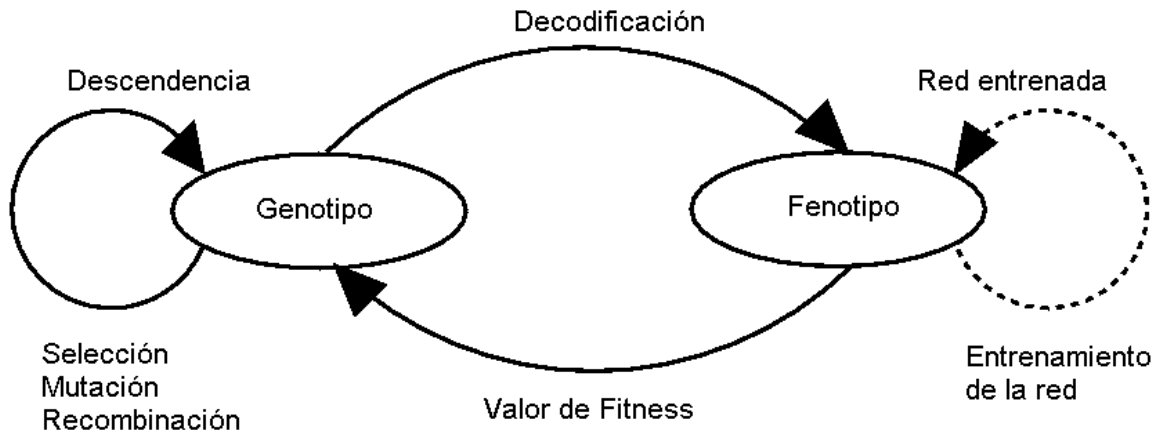


Figura 1 Proceso de diseño evolutivo de arquitecturas neuronales

En el caso más general, un genotipo puede pensarse como un arreglo (una cadena) de genes, donde cada gen toma valores de un dominio definido adecuadamente (también conocidos como alelos). Cada genotipo codifica un fenotipo o solución candidata en el dominio de interés – en nuestro caso una clase de arquitectura neuronal. Tales codificaciones podrían emplear genes que toman valores numéricos para representar unos pocos parámetros o estructuras complejas de símbolos que se transforman en fenotipos (en este caso, redes neuronales) por medio del proceso de *decodificación* apropiado. Este proceso puede ser extremadamente simple o bastante complejo. Las redes neuronales resultantes (los fenotipos) pueden también equiparse con algoritmos de aprendizaje que las entrenen usando el estímulo del entorno o simplemente ser evaluadas en la tarea dada (asumiendo que los pesos de la red se determinan también por el mecanismo de codificación / decodificación). Esta evaluación de un fenotipo determina la aptitud del genotipo correspondiente.

El procedimiento evolutivo trabaja en una población de tales genotipos, preferentemente seleccionando genotipos que codifican a fenotipos de aptitud alta, y reproduciéndolos. Los operadores genéticos tales como la mutación, la cruce, etc., se utilizan para introducir variedad dentro de la población y probar variantes de soluciones candidatas representadas en la población actual. Así, sobre varias generaciones, la población gradualmente evolucionará hacia genotipos que corresponden a fenotipos de aptitud alta.

En este trabajo, el genotipo codifica solo la arquitectura de una red neuronal con conexión hacia delante. El entrenamiento de los pesos de las conexiones se realiza por medio del algoritmo de back-propagation.

3. El problema de la generalización

La topología de una red neuronal, es decir, el número de nodos y la ubicación y el número de conexiones entre ellos, tiene un impacto significativo en la performance de la red y su habilidad para generalizar. La densidad de conexiones en una red neuronal determina su habilidad para almacenar información. Si una red no tiene suficientes conexiones entre nodos, el algoritmo de entrenamiento puede no converger nunca; la red neuronal no es capaz de aproximar la función. Por el otro lado, en una red densamente conectada, puede ocurrir el sobreajuste (overfitting). El sobreajuste es un problema de los modelos estadísticos donde se presentan demasiados parámetros. Esto es una mala situación porque en lugar de aprender a aproximar la función presente en los datos, la red simplemente puede memorizar cada ejemplo de entrenamiento. El ruido en los datos de entrenamiento se aprende entonces como parte de la función, a menudo destruyendo la habilidad de la red para generalizar.

3.1. Validación Cruzada

La esencia del aprendizaje back-propagation es codificar un mapeo entrada-salida (representado por un conjunto de ejemplos etiquetados) en los pesos y umbrales de un Perceptrón multicapa. Lo que queremos es que la red se entrene bien de forma tal que aprenda lo suficiente acerca del pasado para generalizar en el futuro. Desde esta perspectiva el proceso de aprendizaje debe elegir la parametrización de la red más acorde a este conjunto de ejemplos. Más específicamente, podemos ver al problema de selección de la red como una elección, dentro del conjunto de modelos de estructuras candidatas (parametrizaciones), de la mejor estructura de acuerdo a un cierto criterio.

En este contexto, una herramienta estándar en estadística conocida como *validación cruzada* provee una guía interesante [Stone, 1974]. El set de entrenamiento se particiona aleatoriamente en dos subsets disjuntos:

- El *subset de estimación*, usado para seleccionar el modelo.
- El *subset de validación*, usado para evaluar o validar el modelo.

La motivación aquí es validar el modelo sobre un set de datos diferente de aquel utilizado para la estimación de los parámetros. De este modo podemos usar el set de entrenamiento para evaluar la performance de varios modelos candidatos, y así elegir el mejor.

El uso de la validación cruzada resulta particularmente interesante cuando debemos diseñar una red neuronal grande que tenga como objetivo a la generalización.

3.1.1. Entrenamiento con Detención Temprana (Early Stopping)

Comúnmente, un Perceptrón multicapa entrenado con el algoritmo de back-propagation aprende en etapas, moviéndose desde la realización de funciones de mapeo bastante simples a más complejas a medida que progresa la sesión de entrenamiento. Esto se ejemplifica por el hecho de que en una situación típica el error cuadrático medio disminuye con el incremento del número de repeticiones durante el entrenamiento: comienza con un valor grande, decrece rápidamente, y luego continúa disminuyendo lentamente mientras la red hace su camino hacia un mínimo local sobre la superficie de error. Teniendo como meta a una buena generalización, es muy difícil darse cuenta cuándo es el mejor momento de detener el entrenamiento si solamente estamos mirando a la curva de aprendizaje para el entrenamiento. En particular, como mencionamos anteriormente, es posible que la red termine sobreajustándose a los datos de entrenamiento si la sesión de entrenamiento no se detiene en el momento correcto.

Podemos identificar el comienzo del sobreajuste a través del uso de la validación cruzada, para lo cual los ejemplos de entrenamiento se separan en un subconjunto de estimación y un subconjunto de validación. El subconjunto de estimación se utiliza para entrenar a la red en el modo usual, excepto por una modificación menor: la sesión de entrenamiento se detiene periódicamente (cada tantas repeticiones), y se evalúa la red con el set de validación después de cada período de entrenamiento. Más específicamente, el proceso periódico de estimación-seguimiento-de-validación procede de la siguiente manera:

- Después del período de estimación (entrenamiento), se fijan todos los pesos y los umbrales del Perceptrón multicapa, y la red opera en su modo hacia delante. El error de validación se mide así para cada ejemplo en el set de validación.
- Cuando la fase de validación se completa, la estimación (entrenamiento) se reanuda para otro período y el proceso se repite.

Este procedimiento se denomina método de entrenamiento con detención temprana.

La figura 2 muestra las formas conceptualizadas de dos curvas de aprendizaje, una perteneciente a las medidas sobre el subconjunto de estimación y la otra sobre el subconjunto de validación. Normalmente, el modelo no trabaja tan bien sobre el subconjunto de validación como lo hace sobre el set de estimación, en el cual se basó su diseño. La *curva de aprendizaje de estimación* decrece monótonamente para un número creciente de repeticiones en la forma acostumbrada. En contraste, *curva de aprendizaje de validación* decrece monótonamente hasta un mínimo, entonces empieza a incrementarse mientras continúe el entrenamiento.

Cuando miramos a la curva de aprendizaje de estimación puede parecer que podríamos mejorar si vamos más allá del punto mínimo sobre la curva de aprendizaje de validación. En realidad, lo que la red está aprendiendo más allá de ese punto es esencialmente ruido contenido en el set de entrenamiento. Esta heurística sugiere que el punto mínimo sobre la curva de aprendizaje de validación sea utilizado como un criterio para detener la sesión de entrenamiento.

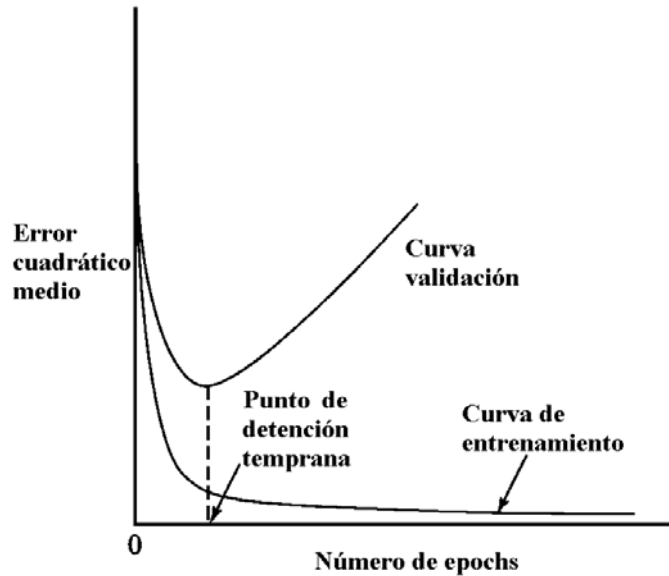


Figura 2 Ilustración de la regla de detención temprana basada en la validación cruzada.

3.2. Poda de la Red (Pruning)

Para resolver problemas del mundo real con redes neuronales usualmente se requiere el uso de redes altamente estructuradas de bastante tamaño. Un asunto práctico que asoma en este contexto es aquel sobre la minimización del tamaño de la red manteniendo la buena performance. Una red neuronal con tamaño mínimo es menos probable que aprenda el ruido en los datos de entrenamiento, y puede así generalizar mejor sobre datos nuevos

La producción de la topología (es decir, el número de capas y neuronas ocultas de la red) la dejaremos en manos del proceso evolutivo; nos concentraremos en un enfoque basado en una forma de *regularización de la complejidad*.

Al diseñar un Perceptrón multicapa por el método que sea, estamos en efecto construyendo un *modelo* no-lineal del fenómeno físico responsable de la generación de ejemplos de entrada-salida usados para entrenar la red. En la medida que el diseño de la red es estadístico por naturaleza, necesitamos un balance adecuado entre la confiabilidad de los datos de entrenamiento y la calidad del modelo. En el contexto del aprendizaje back-propagation o cualquier otro procedimiento de aprendizaje supervisado, podemos producir el balance minimizando el riesgo total expresado como [Haykin, 1999]:

$$R(\mathbf{w}) = \varepsilon_S(\mathbf{W}) + \lambda \varepsilon_C(\mathbf{w})$$

El primer término, $\varepsilon_S(\mathbf{W})$, es la *medida de performance* estándar, la cual depende tanto de la red (modelo) y de los datos de entrada. En el aprendizaje back-propagation se define típicamente como un error cuadrático medio cuya evaluación se extiende sobre las neuronas de salida de la red, y el cual se lleva a cabo para todos los ejemplos de entrenamiento. El segundo término, $\varepsilon_C(\mathbf{w})$, es la *penalidad de complejidad*, la cual depende solamente de la red

(modelo); su inclusión impone sobre la solución un conocimiento a priori que tengamos sobre los modelos siendo considerados. Podemos pensar a λ como un **parámetro de regularización**, representando la importancia relativa del término de la penalidad de complejidad con respecto al término de la medida de performance. Cuando λ es cero, el proceso de aprendizaje no está restringido, y la red se determina completamente con los ejemplos de entrenamiento. Cuando λ se hace infinitamente grande, por el contrario, la implicación es que la restricción impuesta por la penalidad de complejidad es por sí misma suficiente para especificar la red, lo cual es otra manera de decir que los ejemplos de entrenamiento no son confiables. En los casos prácticos donde se utiliza la regularización de la complejidad para mejorar la generalización, al parámetro λ se le asigna un valor entre medio de estos dos casos extremos.

La forma de regularización de complejidad que utilizamos en este trabajo es la **degradación de pesos (weight decay)** [Hinton, 1989]. En el procedimiento weight-decay, el término penalidad de complejidad se define como la norma al cuadrado del vector de pesos \mathbf{w} (es decir, todos los parámetros libres) en la red, como se muestra en:

$$\varepsilon_c(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i \in C_{total}} w_i^2$$

donde el conjunto C_{total} se refiere a todos los pesos de la red. Este procedimiento opera al forzar a algunos de los pesos en la red a tomar valores cercanos a cero, mientras permite que otros pesos retengan sus valores relativamente grandes. En consecuencia, los pesos de la red se agrupan a grandes rasgos en dos categorías: aquellos que tienen una gran influencia sobre la red (modelo), y aquellos que tienen poca o ninguna influencia sobre ella. Los pesos en la última categoría se llaman *pesos excedentes*. En la ausencia de la regularización de complejidad, estos pesos resultan en una generalización pobre en virtud de sus altas probabilidades de tomar valores completamente arbitrarios o causar que la red sobreajuste los datos al tratar de producir una leve reducción en el error de entrenamiento [Hush, 1993]. El uso de la regularización de complejidad fomenta que los pesos excedentes asuman valores cercanos a cero, y por lo tanto mejoran la generalización.

4. El problema de la permutación

Un problema que enfrentan las redes neuronales evolutivas es el problema de la permutación. Este problema no solo hace a la evolución menos eficiente, sino que también dificulta a los operadores de recombinación la producción de hijos con alta aptitud. La causa es el mapeo muchos-a-uno desde la representación codificada de una red neuronal hacia la red neuronal real decodificada, porque dos redes que ordenan sus nodos ocultos en forma diferentes tienen distinta representación pero pueden ser funcionalmente equivalentes. Por ejemplo, las redes neuronales mostradas por las figuras 3(a) y 4(a) son equivalentes, pero tienen distintas representaciones del genotipo, como se ve en las figuras 3(b) y 4(b) usando el esquema de codificación directo. En general, cualquier permutación de los nodos ocultos producirá redes neuronales funcionalmente equivalentes pero con distintas representaciones del genotipo. Esto también es verdad para los esquemas de codificación indirecta.

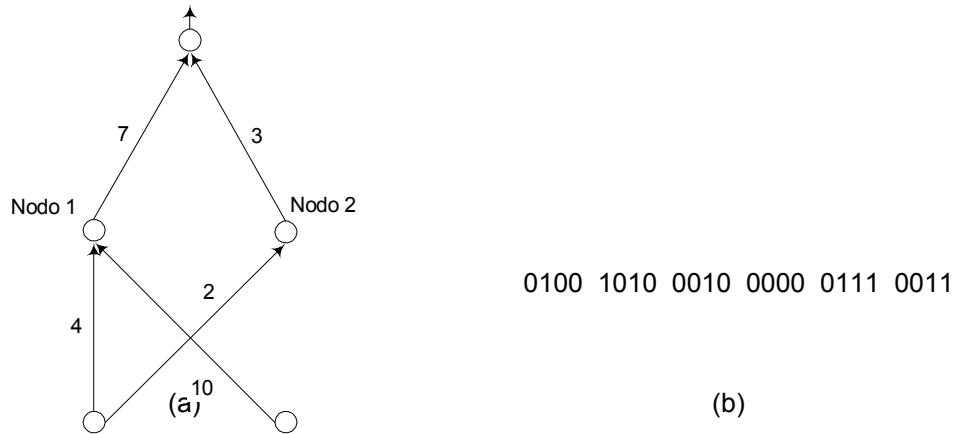


Figura 3 (a) Una red neuronal con sus pesos de conexión; (b) Una representación binaria de los pesos, asumiendo que cada peso se representa con 4 bits. Cero significa sin conexión.

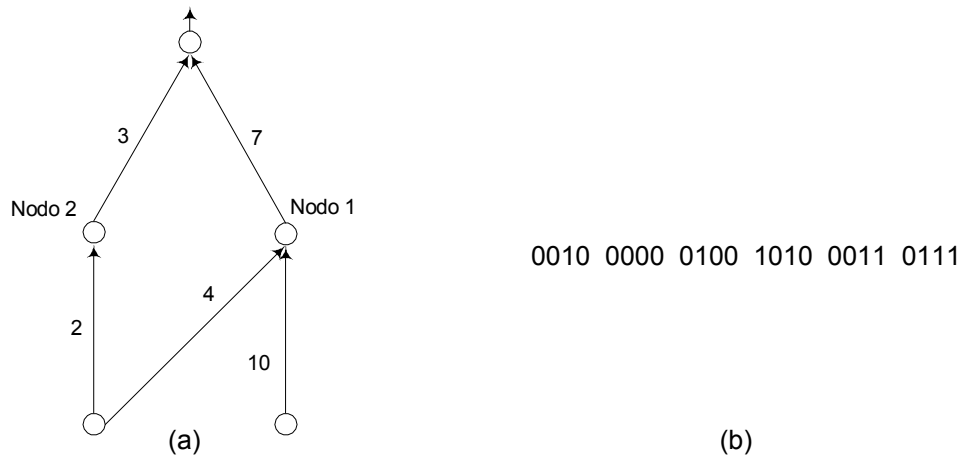


Figura 4 (a) Una red neuronal que es equivalente a la dada en la figura 3(a); (b) Su representación del genotipo bajo el mismo esquema de representación.

Para atenuar los efectos del problema de la permutación, implementaremos una cruce de fenotipos, es decir, una cruce que trabaja sobre redes neuronales en lugar de hacerlo sobre las cadenas de genes que forman la población. En cada padre hacemos un corte simple aleatorio separando los nodos de cada capa en dos partes: una a la izquierda y la otra a la derecha del corte. Primero producimos al primer hijo con la parte izquierda de la madre y la parte derecha del padre. Después se combinan las partes opuestas para hacer al otro hijo.

En la cruce, tal como fue definida, la parte izquierda de una red neuronal se combina siempre con la parte derecha de otra red neuronal. Pero de esta forma si existieran buenas características en las dos partes izquierdas, éstas no se podrían combinar, lo cual no es bueno desde el punto de vista del problema de la permutación. De manera de hacer que este problema sea menos significativo, introducimos un *operador de inversión*, previo a la aplicación de la recombinación. La inversión consiste en intercambiar dos nodos elegidos al azar, con todas sus conexiones, en cada padre.

Otro operador que ayuda ante el problema de la permutación es la mutación. Este operador induce a explorar la totalidad del espacio de búsqueda y permite mantener la diversidad

genética en la población, para que el algoritmo genético sea capaz de encontrar soluciones entre todas las permutaciones posibles de la red.

5. El problema de la evaluación ruidosa de la aptitud

La evaluación de la aptitud de las arquitecturas de redes neuronales siempre será ruidosa si la evolución de las arquitecturas se separa del entrenamiento de los pesos. La evaluación es ruidosa porque lo que se utiliza para evaluar la aptitud del genotipo es la arquitectura real con pesos (es decir, el fenotipo creado a partir del genotipo a través de reglas de desarrollo), y el mapeo entre fenotipo y genotipo no es uno-a-uno. En la práctica, un modo de evaluar genotipos es transformando un genotipo (una arquitectura codificada) en un fenotipo (una arquitectura completamente especificada) a través de reglas de desarrollo, y entonces entrenar al fenotipo a partir de distintos pesos iniciales de conexión generados al azar. El resultado del entrenamiento se utiliza como parte de la función de aptitud para evaluar el genotipo. En otras palabras, se utiliza la aptitud del fenotipo para estimar la aptitud del genotipo. Existen dos fuentes principales de ruido:

1. La primer fuente es la inicialización de los pesos al azar. Diferentes pesos iniciales aleatorios pueden producir diferentes resultados de entrenamiento. Así, el mismo genotipo puede tener una aptitud bastante distinta debido a los distintos pesos iniciales aleatorios utilizados en el entrenamiento.
2. La segunda fuente es el algoritmo de entrenamiento. Diferentes algoritmos de entrenamiento pueden producir diferentes resultados de entrenamiento aún partiendo del mismo conjunto de pesos iniciales. De aquí la importancia de la función de error utilizada y la capacidad de búsqueda global del algoritmo.

Tal ruido puede engañar a la evolución, porque el hecho de que la aptitud de un fenotipo generado a partir del genotipo G1 sea más alto que aquel generado a partir del genotipo G2 no implica que G1 tenga verdaderamente mejor calidad que G2.

De manera de reducir este ruido, se entrenará a cada arquitectura muchas veces partiendo de diferentes pesos iniciales aleatorios. Tomamos entonces el mejor resultado para estimar la aptitud del genotipo, y guardamos la semilla generadora de números aleatorios utilizada para poder reproducir este resultado. Este método incrementa el tiempo de cómputo para la evaluación de la aptitud, así que se analizará en la experimentación en que medida se atenúa el ruido en función de número de repeticiones del entrenamiento.

En general, el esquema de codificación directa es mejor para el ajuste fino y preciso de una arquitectura para una aplicación específica, debido a que a partir de la decodificación del cromosoma obtenemos directamente la arquitectura de red neuronal. La codificación dentro de cromosomas de reglas de desarrollo para generar redes neuronales (lo cual constituye uno de los mejores esquemas de codificación indirecta) puede producir representaciones muy compactas, pero sufre los problemas del ruido causados por la transformación de las

reglas de desarrollo en arquitecturas de red neuronal. Para evitar este ruido adicional introducido por las reglas de desarrollo, se utilizará el esquema de codificación directa en el algoritmo.

6. Diseño experimental

El algoritmo híbrido que empleamos para la generación automática de redes neuronales es el siguiente:

1. Crear una población inicial de individuos (redes neuronales) con topologías aleatorias. Entrenar a cada individuo usando el algoritmo de back-propagation.
2. Seleccionar al padre y a la madre mediante un torneo: para cada uno, elegir tres individuos al azar de la población, quedarse con aquel que tenga el menor error.
3. Recombinar ambos padres para obtener dos hijos.
4. Mutar aleatoriamente a cada hijo.
5. Entrenar a cada hijo usando el algoritmo de back-propagation.
6. Ubicar a los hijos en la población mediante un torneo: para cada uno, elegir tres individuos al azar de la población, reemplazar por el hijo a aquel que tenga el mayor error.
7. Repetir desde el paso 2 durante un número dado de generaciones.

Este algoritmo emplea un esquema de codificación directo, utiliza la selección por torneo (basada en el orden) y el reemplazo consiste en una actualización gradual también aplicando la técnica de torneo. Cada red neuronal se entrena durante 500 repeticiones con back-propagation. Este valor es más alto del que se utiliza generalmente para entrenar redes neuronales, dándole al entrenamiento tiempo suficiente para converger y aprovechando entonces todo el potencial de cada red.

En cada experimento se llevan a cabo 100 generaciones del algoritmo genético. Todos los experimentos utilizan un tamaño de población de 20 individuos. Este es un valor estándar usado en algoritmos genéticos. Aquí debemos hacer un compromiso entre presión selectiva

y tiempo de computación. El empleo de 20 individuos sirve para acelerar el desarrollo de los experimentos sin afectar a los resultados.

El algoritmo back-propagation se basa en el modo de entrenamiento secuencial, la función de activación elegida para cada neurona es la tangente hiperbólica. Utilizamos un número de repeticiones de back-propagation n igual 3 para entrenar a cada red neuronal partiendo de diferentes pesos iniciales aleatorios. Se usa entonces el mejor resultado para estimar la aptitud de la red.

6.1. El set de datos utilizado

El set de entrenamiento elegido para usarse en estos experimentos se obtuvo desde un archivo de data sets en Internet [Blake y Merz, 1998]. Consiste en datos concernientes a 600 solicitudes de tarjeta de crédito. Cada solicitud de tarjeta de crédito constituye un ejemplo de entrenamiento. Para el aprendizaje, la información de la solicitud forma la entrada a la red neuronal, y la salida es un valor verdadero o falso que especifica si la solicitud fue o no aprobada. La idea, por supuesto, es entrenar a la red neuronal sobre estos ejemplos, y luego tener una red neuronal capaz de aprobar solicitudes de tarjeta de crédito.

Toda la información de las solicitudes está cambiada a símbolos sin significado para mantener la confidencialidad. Los atributos de un ejemplo de entrenamiento se ven como:

b,30.83,0,u,g,w,v,1.25,t,t,01,f,g,00202,0,+

Algunos de los parámetros toman la forma de números. Para presentar estos números a la red, se determina el mínimo y el máximo valor para ese atributo dentro del set completo de entrenamiento y luego se escalan los atributos entre -1 y $+1$. Por ejemplo, si un atributo del set de entrenamiento va desde 0 a 100.000, entonces se debe comprimir entre -1 y $+1$ a todos los valores para este atributo, de manera tal que puedan ser usados por la red neuronal. Las entradas no numéricas (las letras en el ejemplo de arriba) son respuestas a preguntas múltiple choice. Por ejemplo, el primer atributo en cada ejemplo de entrenamiento podría ser a o b . Ya que hay dos opciones, este atributo tomará la forma de dos entradas a la red donde una es verdadera y la otra es falsa, dependiendo de si los ejemplos usan a o b . Verdadero se representa en la red como $+1$, y falso se representa como -1 . Utilizando estos dos métodos de transformar los datos de la solicitud de aprobación de tarjetas de crédito en una forma que pueda utilizar la red, llegamos a una red que tiene 47 entradas. El signo $+$ del final del ejemplo de arriba constituye la *clase* del ejemplo. En este caso es un $+$ o un $-$ dependiendo de si la solicitud se aprobó o no, por lo tanto la red neuronal tendrá dos salidas, una que se activa para cada caso.

El set de datos se particiona en tres partes de igual tamaño: A, B y C, cada una con la misma cantidad de solicitudes aprobadas y desaprobadas.

6.2. Significado del error

En el análisis de los experimentos, es necesario ser capaz de asignarle una medida a qué tan bien una red neuronal clasifica un set de datos. En este trabajo, al valor utilizado para reportar este resultado se lo refiere como error. El error es inversamente proporcional a la performance sobre el set de validación, caracterizada por el número de patrones clasificados correctamente. Existe una penalidad sobre el número de nodos que utiliza la red, dividida por el número de capas, de lo contrario las redes con muchas capas tendrían una penalidad muy elevada. Además hay un término de penalidad para el porcentaje de conexiones que tiene la red, dividida (nuevamente por los mismos motivos recién mencionados) por el número de capas.

6.3. Regularización de la complejidad óptima

Utilizamos el parámetro de regularización $\lambda \in \{0.05; 0.1; 0.15; 0.2\}$ para entrenar a la red con la partición A (para que los valores sean útiles deben ser pequeños). Luego observamos sus efectos sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 5. Por último evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 6.

De la figura 5 se ve que la red generada con el parámetro $\lambda=0.1$ es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que directamente no implementaba la regularización ($\lambda=0$). Se comprueba entonces que el uso de la regularización de la complejidad fomenta que los pesos excedentes (aquellos que tienen poca o ninguna influencia sobre la red) asuman valores cercanos a cero, y por lo tanto mejoran la generalización. Esto se refleja en la figura 6, donde la red generada con el parámetro $\lambda=0.1$ posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C).

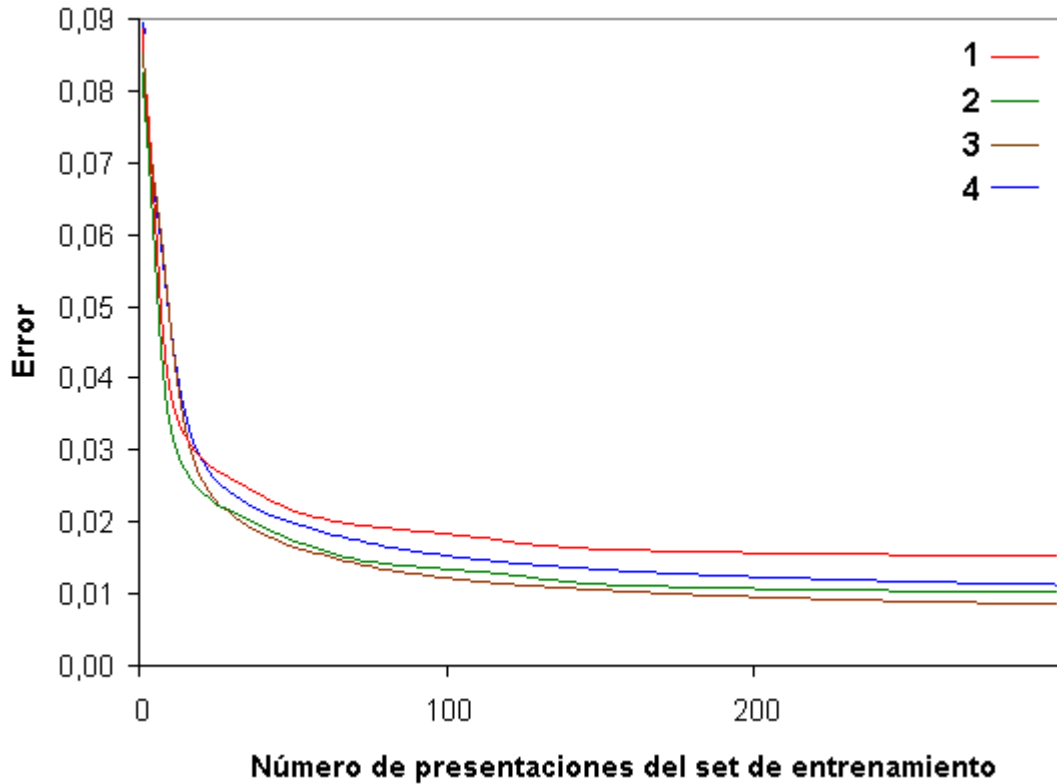


Figura 5 Regularización de la complejidad con parámetro de regularización ajustable: (1) $\lambda=0$ (sin regularización); (2) $\lambda=0.05$; (3) $\lambda=0.01$; (4) $\lambda=0.15$

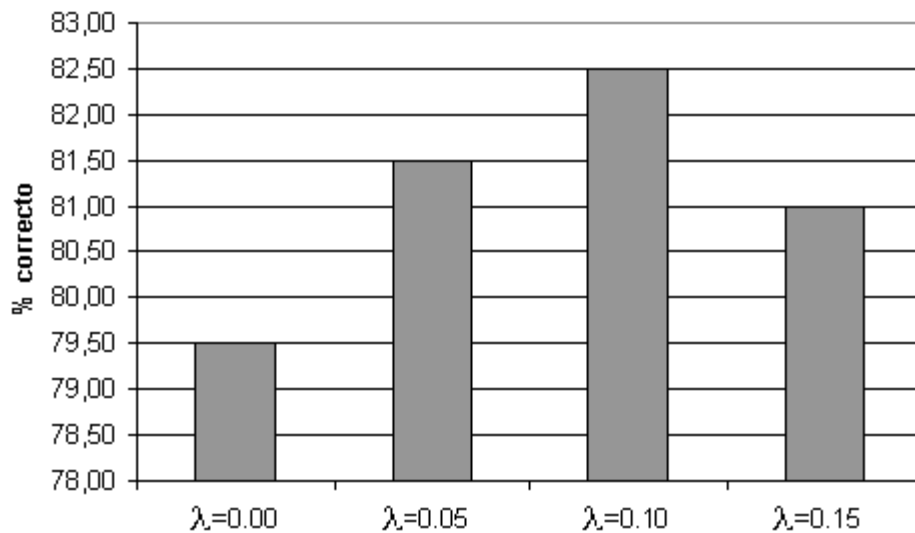


Figura 6 Comparación de los porcentajes de aciertos de las redes neuronales generadas con distintos parámetros de regularización λ .

6.4. Detención temprana óptima

La pregunta que surge aquí es cuántas veces deberíamos permitir que el entrenamiento no mejore sobre el set de validación, antes de detener el entrenamiento. Definimos entonces al parámetro de detención β para representar a este número de entrenamientos.

Utilizamos el parámetro de detención $\beta \in \{2; 5; 10\}$ para entrenar a la red con la partición A. Después observamos sus efectos sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 7. Posteriormente evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 8.

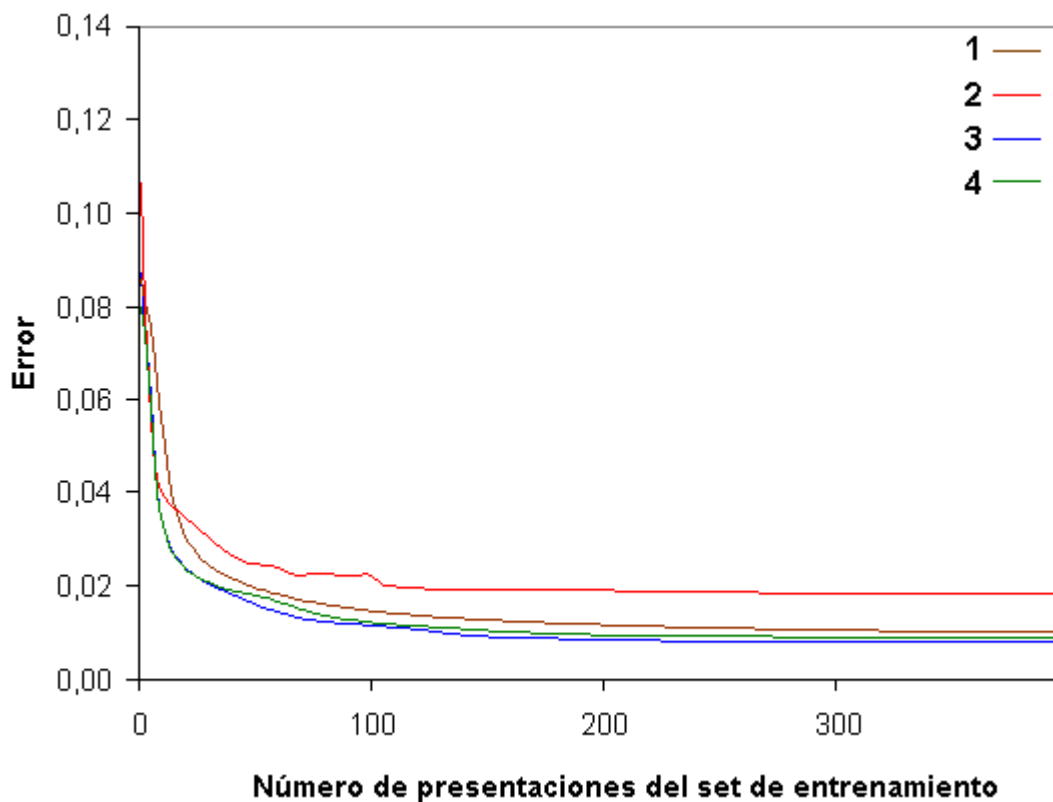


Figura 7 Detención temprana con parámetro de detención ajustable: (1) Sin detención temprana; (2) $\beta=2$; (3) $\beta=5$; (4) $\beta=10$.

De la figura 7 vemos que, si bien la diferencia es pequeña, la red neuronal generada con el parámetro de detención $\beta=5$ es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que directamente no implementaba la detención (lo que equivaldría a $\beta \rightarrow \infty$). Esto verifica que lo que la red aprende más allá del punto de detención óptimo es esencialmente ruido contenido en el set de entrenamiento. En el caso de $\beta=2$, el crecimiento del error posiblemente se deba a que la detención se realiza demasiado pronto, sin dejarle a la red el tiempo suficiente para aprender bien.

Los resultados anteriores se corroboran en la figura 8, donde la red generada con el parámetro $\beta=5$ posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C).

Una variante interesante es aplicar nuevamente el método de detención temprana cuando se entrena con la partición B a la mejor red neuronal generada (en nuestro caso cuando usamos $\beta=5$). En este caso, la habilidad de clasificación de la red sobre la partición C se muestra en la figura 9. Esta red entrenada con $\beta=5$ sigue teniendo el mejor porcentaje.

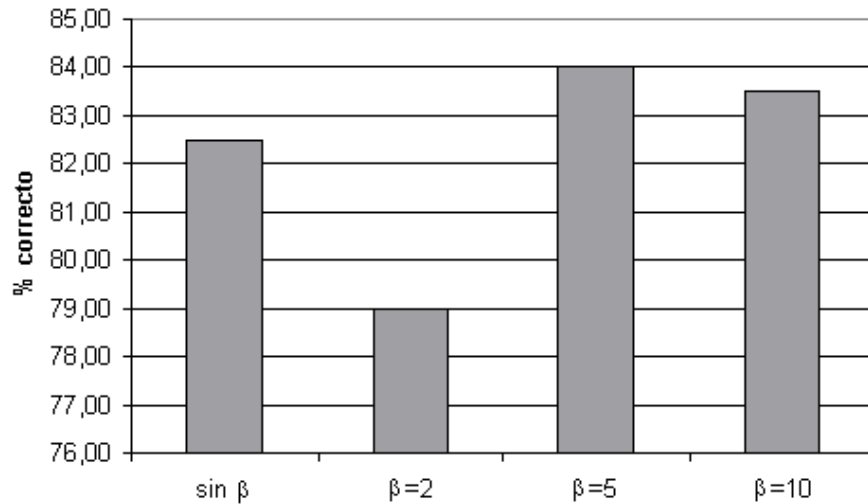


Figura 8 Comparación de los porcentajes de aciertos de las redes neuronales generadas con distintos parámetros de detención β .

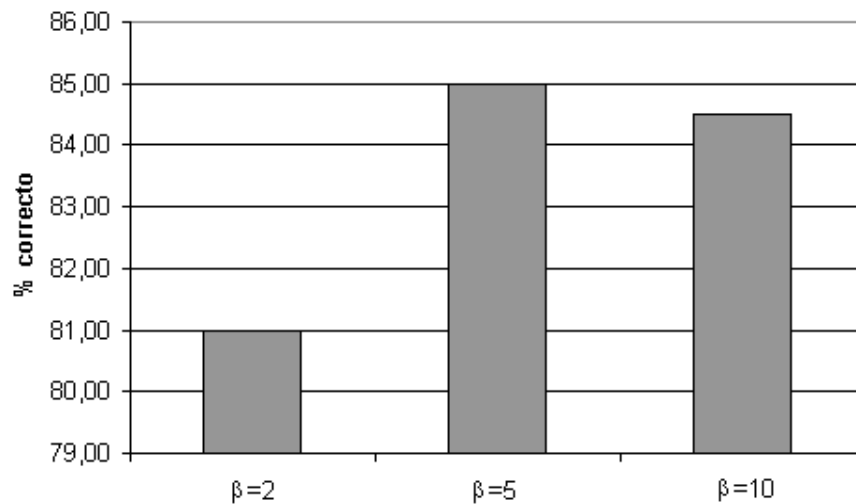


Figura 9 Comparación de los porcentajes de aciertos de la red neuronal generada usando $\beta = 5$, entrenada con la partición B utilizando distintos β

6.5. Comparación de la red neuronal óptima con otras redes

Para determinar si el proceso evolutivo está realmente mejorando o no a las redes neuronales en cuanto a sus topologías específicas del dominio, vamos a comparar a la red generada por el algoritmo híbrido con la mejor topología random (generada en la primera generación del AG). Estas redes también son comparadas con una topología similar a la obtenida por el algoritmo híbrido pero 100% conectada (o conectada completamente).

Observamos los efectos de las distintas topologías sobre la convergencia de la red neuronal cuando se entrena con la partición B, como se aprecia en la figura 10. Posteriormente evaluamos su habilidad de clasificación sobre la partición C, mostrada en la figura 11.

De la figura 10 vemos que la red neuronal generada con el algoritmo híbrido es capaz de aprender mejor un nuevo set de datos que las otras redes, incluyendo aquella que implementaba su misma topología pero conectada completamente.

La red generada con el algoritmo híbrido también posee el mejor porcentaje de clasificación de ejemplos no vistos con anterioridad (los de la partición C), como se aprecia en la figura 11.

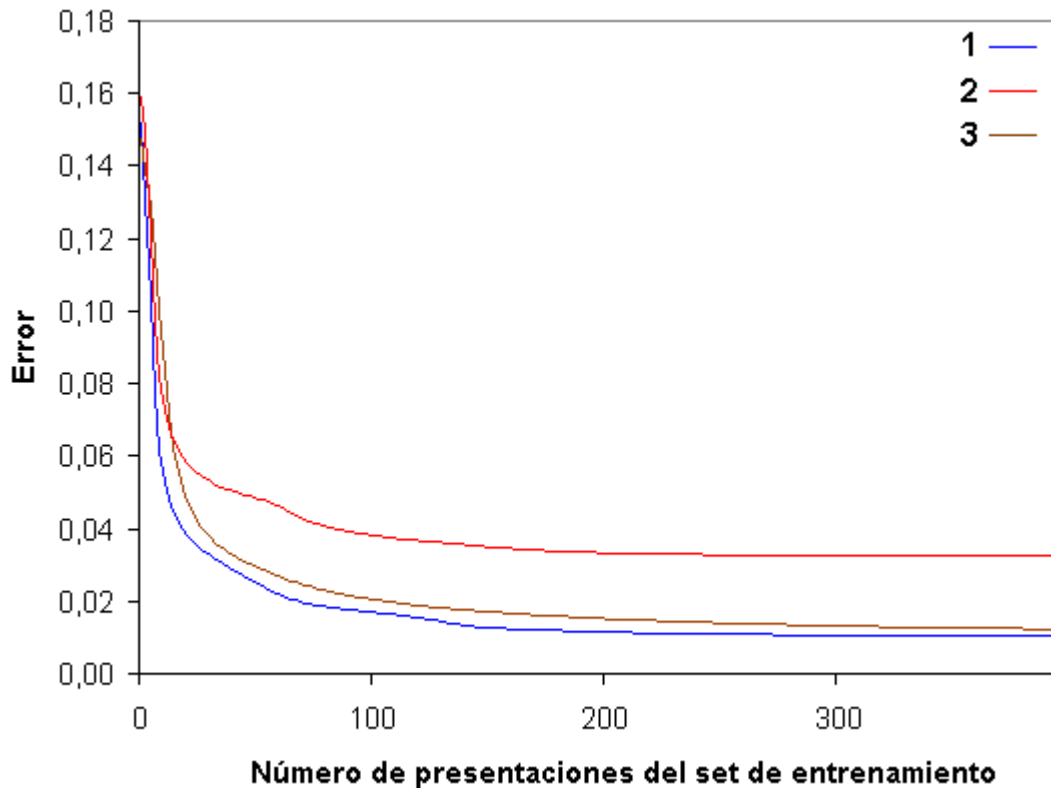


Figura 10 Capacidad de aprender nuevos datos usando: (1) La topología resultante del algoritmo híbrido; (2) La mejor topología random; (3) La topología del algoritmo híbrido pero 100% conectada.

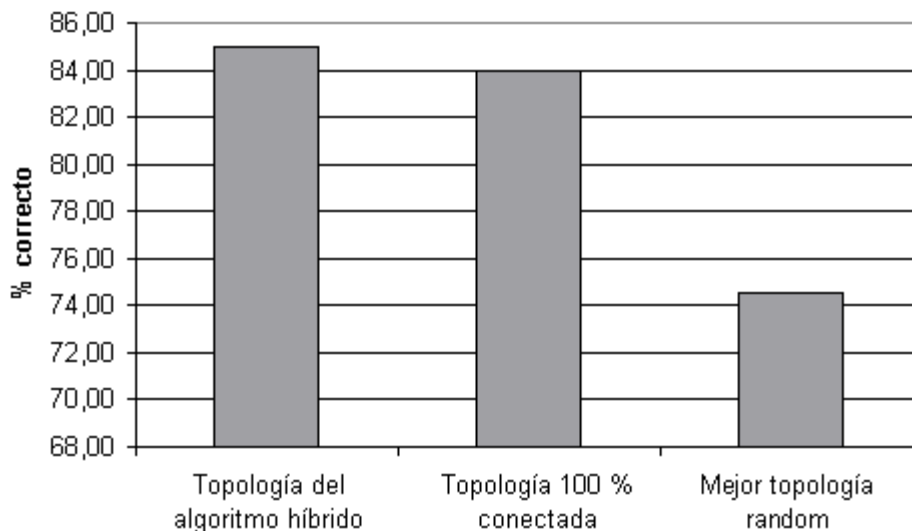


Figura 11 Comparación de los porcentajes de aciertos de las redes neuronales con distintas topologías y conexiones.

7. Conclusiones

El mundo real a menudo posee problemas que no pueden ser resueltos exitosamente por una sola técnica básica; cada técnica tiene sus pros y sus contras. El concepto de sistema híbrido en inteligencia artificial consiste en combinar dos enfoques, de manera de que se compensen sus debilidades y se potencien sus fortalezas.

El objetivo de este trabajo es crear una forma de generar topologías de red neuronal que puedan fácilmente aprender y clasificar una cierta clase de datos. Para lograrlo, el algoritmo genético se usa para encontrar la mejor topología que cumpla con esta tarea. Cuando el proceso termina, el resultado es una población de redes neuronales específicas del dominio, listas para tomar nuevos datos no vistos anteriormente.

Este trabajo presenta la implementación y los experimentos resultantes del sistema híbrido que utiliza el algoritmo genético para buscar en el espacio de estructuras de una red neuronal de manera de encontrar una topología óptima.

El análisis de los resultados de los experimentos presentados demuestran que esta implementación AG/RN es capaz de crear topologías de red neuronal que en general, se desempeñan mejor que topologías random o conectadas completamente cuando se aprende y clasifica nuevos datos de un dominio específico.

Para disminuir los efectos de la permutación, o equivalencia funcional de genotipos, implementamos un operador de cruce directamente sobre los fenotipos, y un operador de inversión, previo a la aplicación de la recombinación.

Considerando que diferentes pesos iniciales aleatorios en las redes neuronales pueden producir diferentes resultados de entrenamiento, la solución propuesta consistió en entrenar varias veces a cada arquitectura partiendo de diferentes pesos iniciales aleatorios, y usar entonces el mejor resultado para estimar la aptitud del genotipo. Un incremento en el número de repeticiones del algoritmo mejora el progreso del algoritmo genético, debido a que la evaluación del genotipo es más precisa. Sin embargo, el aumento del número de repeticiones también conlleva a un crecimiento notorio en los tiempos de computación insumidos por el algoritmo. Teniendo en cuenta que la mejora en el desempeño del algoritmo genético con respecto al número de repeticiones del entrenamiento se hace cada vez menos apreciable, se determinó el mejor valor de esta última variable para lograr un compromiso tiempo-performance.

Se planteó una forma de poda de la red (pruning), la regularización de la complejidad, como un modo de fomentar que los pesos excedentes, que tienen poca o ninguna influencia sobre la red, asuman valores cercanos a cero, y por lo tanto mejoren la generalización. Se empleó el método de degradación de pesos (weight decay) observándose sus efectos al variar el parámetro de regularización. Se pudo apreciar que el uso de este método produjo una mejora en las redes resultantes del proceso evolutivo en cuanto al aprendizaje y a la generalización.

Se planteó también un método de validación cruzada, la detección temprana (early stopping) del entrenamiento back-propagation, como un modo de evitar que la red sobreajuste a los datos de entrenamiento. Se utilizaron distintos valores para el parámetro de detención, que determina el momento en que se para el entrenamiento, determinándose el valor óptimo. También se pudo visualizar una mejora de las redes neuronales generadas, que fueron capaces de aprender mejor y clasificar mejor.

Un aspecto que debería examinarse con más profundidad es cómo se debería determinar el costo de una topología. En la implementación actual, el costo es simplemente el error de entrenamiento de la red neuronal sobre una partición del set de datos. La pregunta es si ésta es la mejor forma de determinar la aptitud de una topología. La virtud más importante del criterio que utilizamos es su manejo matemático sencillo. Sin embargo, en muchas situaciones que se dan en la práctica, la minimización de la función de costo corresponde a optimizar una cantidad intermedia que no es el objetivo final del sistema, y por lo tanto puede conducir a una performance subóptima. Por ejemplo, en sistemas para mercados financieros, la meta final de un inversionista es maximizar la expectativa de retorno a mínimo riesgo [Choey y Wiegand, 1996]. La relación *recompensa-volatilidad* como medida de performance del *retorno con riesgo ajustado* es intuitivamente más atractiva que la aptitud estándar.

Otro paso muy importante sería repetir los experimentos presentados aquí para set de datos diferentes. La escalabilidad es un problema importante en las implementaciones de redes neuronales, por lo tanto sería interesante ver cómo la implementación actual escala a redes grandes que contienen miles de entradas.

Por último, otro punto que debería ser explorado es la paralelización del algoritmo genético, especialmente considerando los tiempos elevados de procesamiento que se manejaron durante los experimentos. Excepto en la fase de selección, donde hay competencia entre los

individuos, las únicas interacciones entre los miembros de la población ocurren durante la fase de reproducción, y solo son necesarios dos padres para engendrar un hijo. Cualquier otra operación de la evolución, en particular la evaluación de cada miembro de la población, puede hacerse separadamente. Por lo tanto, casi todas las operaciones en un AG son implícitamente paralelas. El uso de computadoras que trabajan en paralelo no solo provee más espacio de almacenamiento sino también permite el uso de múltiples procesadores para producir y evaluar más soluciones en menos tiempo. Al paralelizar el algoritmo, es posible incrementar el tamaño de la población, reducir el costo computacional y mejorar entonces la performance del AG. Los algoritmos genéticos paralelos o PGA (Parallel Genetic Algorithms) constituyen un área reciente de investigación, y existen enfoques muy interesantes como el modelo de islas o el modelo de grano fino [Hue, 1997].

Bibliografía

- Blake C. L. y Merz C. J. (1998)** *UCI Repository of machine learning databases*. <http://www.ics.uci.edu/~mllearn/MLRepository.html> Irvine, CA: University of California, Department of Information and Computer Science.
- Choey M. y Weigend A. (1996)** *Nonlinear trading models through sharp ratio maximization*. Decision Technologies for Financial Engineering, pp. 3-22, Singapore.
- Dow R. J. y Sietsma J. (1991)** *Creating Artificial Neural Networks that generalize*. Neural Networks, vol. 4, no. 1, pp. 198-209.
- García Martínez Ramón (1997)** *Sistemas Autónomos. Aprendizaje Automático*. Nueva Librería.
- García Martínez, R. y Borrajo, D. (2000)** *An Integrated Approach of Learning, Planning and Executing*. Journal of Intelligent and Robotic Systems. Volumen 29, Número 1, Páginas 47-78. Kluwer Academic Press. 2000
- Goldberg D. E. (1991)** *A comparative analysis of selection schemes used in genetic algorithms*. In Gregory Rawlins, editor. Foundations of Genetic Algorithms, pages 69-93, San Mateo, CA: Morgan Kaufmann Publishers.
- Haykin Simon (1999)** *Neural Networks. A Comprehensive Foundation*. Second Edition. Prentice Hall.
- Hertz J., A. Krogh y R. Palmer (1991)** *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley.
- Hinton G. E. (1989)** *Connectionist Learning Procedures*. Artificial Intelligence, vol. 40, pp. 185-234
- Holland J. H. (1975)** *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Ann Arbor).

- Holland, J. H. (1980)** *Adaptive algorithms for discovering and using general patterns in growing knowledge-based.* International Journal of Policy Analysis and Information Systems, 4(3), 245-268.
- Holland, J. H. (1986)** *Escaping brittleness: The possibilities of general purpose learning algorithms applied in parallel rule-based systems.* In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), Machine Learning II (pp. 593-623). Los Altos, CA: Morgan Kaufmann.
- Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1987).** *Classifier systems, Q-morphisms, and induction.* In L. Davis (Ed.), Genetic algorithms and simulated annealing (pp. 116-128).
- Honavar V. and L. Uhr. (1993)** *Generative Learning Structures and Processes for Generalized Connectionist Networks.* Information Sciences, 70:75--108.
- Hue Xavier (1997)** *Genetic Algorithms for Optimization.* Edinburgh Parallel Computing Centre. The University of Edinburgh.
- Rich E. y Knight K. (1991)** *Introduction to Artificial Networks.* MacGraw-Hill Publications.
- Stone M. (1974)** *Cross-validatory choice and assessment of statistical predictions.* Journal of the Royal Statistical Society, vol. B36, pp. 111-133.
- Wolpert D. H. y Macready W. G. (1995)** *No Free Lunch Theorems for Search.* Technical Report SFI-TR-95-02-010, Santa Fe Institute.
- Yao Xin (1999)** *Evolving Artificial Neural Networks.* School of Computer Science. The University of Birmingham. B15 2TT.
- Yao X. y Liu Y. (1998)** *Toward Designing Artificial Neural Networks by Evolution.* Applied Mathematics and Computation, 91(1): 83-90.