ISSN 1870-4069

Redes neuronales evolutivas con modelos de Lotka-Volterra

Fernando Javier Aguilar Canto

Universidad Autónoma de Yucatán
Facultad de Matemáticas,
México pherjev@gmail.com

Resumen. El entrenamiento de redes neuronales multicapa supone de dos partes diferenciadas: un ajuste de los pesos y una optimización de los hiperparámetros, tales como el número de capas y el número de neuronas en cada una de las mismas, la cual ha sido tradicionalmente elaborada de forma manual. En el presente artículo proponemos un algoritmo genético basado en el modelo ecológico de Lotka-Volterra, cuyo objetivo es permitir la evolución de la arquitectura de las capas densas (completamente conectadas) de la red para lo cual se concibieron dos variantes principales: aplicando cambios constantes en la estructura de las redes hijas (mutaciones constantes) o mediante cambios aleatorios. De forma experimental se observó que el empleo de aleatoriedad en el modelo arroja mejores resultados. Asimismo, fue posible mejorar una red convolucional dada optimizando sus capas densas y, las pruebas realizadas en la base de datos EMNIST lograron alcanzar 93.99 % de exactitud para el problema de clasificación de caracteres alfabéticos, mejorando a la red originalmente concebida. Finalmente, ofrecemos una prueba de la convergencia local del algoritmo propuesto, el cual además presenta un costo computacional polinomial, haciendo de nuestra propuesta una opción válida para el desarrollo de arquitecturas de forma automatizada.

Palabras clave: Redes neuronales prealimentadas, algoritmo evolutivo, redes neuronales evolutivas, redes neuronales convolucionales, optimización hiperparamétrica.

Evolutionary Neural Networks using Lotka-Volterra Models

Abstract. The training of multilayer neural network involves two different parts: an adjustment of the weights and an optimization of the hyperparameters, such as the number of layers and the number of neurons in each of them, which has been traditionally made manually. In this article, we propose a genetic algorithm based on the ecological Lotka-Volterra model, whose objective is to allow the evolution of the architecture of the dense (completely connected) layers of the network. For this, two main

variants were conceived: applying constant changes in the structure of the daughter-networks (constant mutations) or by means of random changes. Experimentally, it was observed that the use of randomness in the model yields better results. Also, it was possible to improve a given convolutional network by optimizing its dense layers and, the tests carried out in the EMNIST database achieved 93.99 % accuracy for the problem of classification of alphabetic characters, improving the network originally conceived. Finally, we offer a test of the local convergence of the proposed algorithm, which also presents a polynomial computational cost, making our proposal a valid option for the development of architectures in an automated way.

Keywords: Feedforward neural networks, evolutionary algorithm, evolutionary neural networks, convolutional neural networks, hyperparametric optimization.

1. Introducción

Las Redes Neuronales Prealimentadas (mejor conocidas como *Feedforward Neural Networks* o FNN) han sido exitosamente aplicadas en distintos contextos como Visión Computacional y Reconocimiento de Voz [12]. En gran medida, el éxito de la red depende de la arquitectura para abordar el problema específico. No obstante, en general, tanto para la elección de la arquitectura como el resto de los hiperparámetros, se carece de una fórmula explícita para su configuración óptima [22].

El presente trabajo aborda el problema de la optimización hiperparamétrica, en particular para el caso del diseño de las capas completamente conectadas, las cuales se han empleado en importantes redes neuronales profundas como las VGGs propuestas en [20]. El algoritmo presentado en este trabajo sugiere desarrollar un ecosistema artificial de redes neuronales, asignándoles un rol de presas siguiendo el modelo de Lotka-Volterra para efectuar su evolución mediante cambios graduales en su arquitectura. Con los depredadores eliminando a los ejemplares con menor exactitud sobre el conjunto de validación, las redes mejor adaptadas pueden preservar su estructura para las generaciones futuras. Siguiendo esta inspiración biológica, podemos desarrollar comunidades estables de redes neuronales que progresivamente vayan mejorando su respuesta al entorno, es decir, a los datos, para poder, a la postre, obtener una arquitectura adecuada para la resolución de un problema específico.

1.1. Redes Neuronales Prealimentadas

Las Redes Neuronales Prealimentadas consisten en varias capas de neuronas artificiales que realizan una operación de producto punto con sus entradas $\mathbf{x}$ (típicamente de la capa anterior o de la entrada del modelo) con los pesos $\mathbf{w}$ de la neurona, para luego aplicar una función de activación f, de forma que

calculan $y = f(\mathbf{w} \cdot \mathbf{x})$. La disposición en varias capas de neuronas sin conexiones recurrentes forma a las *redes multicapa prealimentadas* [1]. Una red neuronal prealimentada completamente conectada consta de una capa de entrada (*input layer*) y una capa de salida (*output layer*), los cuales dependen del problema modelado. De manera libre aparece tanto el número de capas ocultas (*hidden layers*) como el número de neuronas en cada una, siendo hiperparámetros de la red neuronal [22].

Los Teoremas de Aproximación Universal [10,15] sobre redes neuronales completamente conectadas, básicamente enuncian que una red neuronal de tipo FNN completamente conectada con una capa intermedia con suficientes neuronas pueden aproximar a cualquier función continua si la función de activación empleada no es un polinomio algebraico. Otros resultados matemáticos muestran la importancia de aumentar el número de capas frente al número de neuronas, lo cual, aunado al desarrollo de *Deep Learning* en términos prácticos nos inclina a considerar aumentar tal hiperparámetro [12] y no exclusivamente efectuar un ajuste con una red de una capa intermedia. Por ende, los modelos desarrollados deberán considerar tanto un aumento de neuronas en una capa como de número de capas.

1.2. Antecedentes

Las Redes Neuronales Evolutivas (*Evolutionary Neural Networks*) son un conjunto de estrategias adaptativas inspiradas en los mecanismos de selección natural con el objetivo de optimizar un aspecto de una red neuronal artificial. Reconocemos dos niveles básicos en los que podemos realizar optimización en redes neuronales: un nivel *paramétrico*, propio de los pesos adaptables de la red neuronal y un nivel *hiperparamétrico*, dirigido a las modificaciones no realizadas en los pesos de la red, sino en aspectos meramente externos como la arquitectura (número de capas intermedias con el número de neuronas), las funciones de activación (ReLU contra sigmoide, por ejemplo), los parámetros propios de los optimizadores (como la tasa de aprendizaje o el *momentum* es los gradientes acelerados [8]), entre otros aspectos.

El concepto de evolución ha sido aplicado en redes neuronales tanto a nivel de pesos (paramétrico, véase [18]) con una arquitectura fija, como a nivel hiperparamétrica [16]. Por dos razones centraremos el estudio en el caso de los hiperparámetros. La primera radica en el estado de solución de la optimización hiperparamétrica frente a la paramétrica. Métodos basados en el gradiente (tal como Adam) han demostrado su efectividad en el entrenamiento de los pesos de las redes neuronales [17]. Desde la formulación del Perceptrón en 1958 [19], se disponen de diferentes métodos para entrenar los pesos de la red neuronal. Sin embargo, tradicionalmente, la optimización hiperparamétrica se ha logrado de forma empírica [5,4]. Una segunda razón para centrarse en la optimización hiperparamétrica radica en su motivación biológica. De la Teoría de Hebb, se considera que los pesos de las neuronas reales son adaptables [2], mediante mecanismos de plasticidad neuronal [11], lo cual descarta que se ajusten por generaciones. Lo

que pudiese haber sido moldeado por la evolución es la arquitectura de las redes neuronales.

De acuerdo con Young *et al* [22], en la actualidad, se disponen de tres formas comunes para efectuar la selección de los hiperparámetros, las cuales, junto con otras propuestas, son la *búsqueda manual*, la *búsqueda por rejilla* (consiste en aplicar aumentos y decrementos constantes para explorar el espacio de los hiperparámetros, aunque tiene la desventaja de ser poco eficiente, en especial en espacios multidimensionales), la *búsqueda aleatoria* (propuesta por Bergstra y sus colaboradores en ([5,4]) consistente en modificar a los incrementos constantes por aleatorios, obteniendo mejores resultados) y los *algoritmos genéticos*.

Mención aparte merecen los llamados *algoritmos genéticos*, los cuales constan de tres fases diferenciadas: selección, cruzamiento y mutación. Algunas de estas fases, como selección y mutación son recogidas en el presente artículo. Estos algoritmos han sido implementados en el contexto de la optimización de arquitecturas de redes neuronales, en particular para el contexto de Redes Neuronales Convolucionales en [22,21], optimizando otros hiperparámetros como el número de filtros. Para el caso de reconocimiento de caracteres (que es abordado en este trabajo), existen propuestas de evolución centradas a las capas convolucionales que han sido aplicadas a las bases de datos MNIST y EMNIST. Entre dichas propuestas destacan [6,7] para MNIST y [3] para EMNIST.

2. Metodología

2.1. FNN evolutivas

Como se especificó en anteriormente, una optimización metaparamétrica de una FNN consiste en ajustar el número de capas y neuronas intermedias hasta lograr un mayor rendimiento de la red sobre el conjunto de validación. Las Redes FNNs pueden expresarse de manera abstracta como una sucesión $(s_1, s_2, \ldots, s_n, 0, \ldots)$ donde cada s_i representa el número de neuronas de la capa oculta i. Como el número de capas n es finito, entonces $s_i = 0$ para $i > n$. Por fines representativos, expresaremos la sucesión como $(s_1, \ldots, s_n)$ en lugar de la forma anteriormente presentada.

A la sucesión anteriormente presentada se le definirá como el *código genético* de la FNN. La razón del nombre obedece a que es posible introducir operaciones (*mutaciones*) al código con la finalidad de hallar una arquitectura apropiada, mediante una fisión binaria en dos redes hijas (que pueden ser entendidos como *organismos* en una *ecología artificial*). Tales mutaciones pueden ser de tres tipos:

– α: Agrega k neuronas a una capa aleatoriamente escogida:

$$(r_1, \ldots, r_i, \ldots, r_n) \to (r_1, \ldots, r_i + k, \ldots, r_n). \tag{1}$$

– β: Agrega una capa con k neuronas:

$$(r_1, \ldots, r_n) \to (r_1, \ldots, r_n, k). \tag{2}$$

– γ: Reduce k neuronas en una capa específica o la vuelve 0:

$$(r_1, \ldots, r_i, \ldots, r_n) \to (r_1, \ldots, \text{máx}(r_i - k, 0), \ldots, r_n). \tag{3}$$

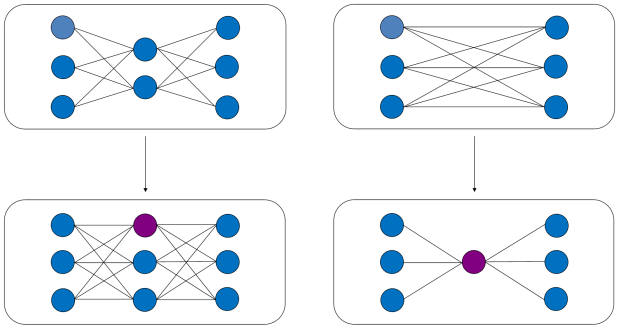

Fig. 1. Mutación de tipo α (izquierda) y β (derecha) con $k = 1$.

La elección de un tipo de mutación u otro será aleatorio, con una probabilidad p_x de elegir una mutación tipo x. Debido a que se tiene una ausencia de deleciones, se aconsejará iniciar una población de redes con el organismo (0).

De esta forma, el organismo inicial se "duplica" para crear a los organismos (0) y (k) hijos, los cuales se vuelven a duplicar para producir una nueva generación de organismos de forma recursiva (véase figura 2).

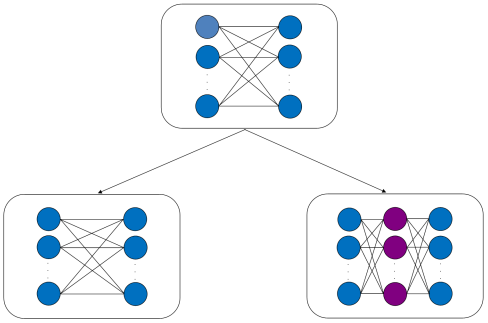

Fig. 2. Duplicación del organismo (0) en (0) y (k).

2.2. Modelo de Lotka-Volterra

Es posible efectuar una optimización simplemente efectuando sucesivas duplicaciones de organismos y tomar el rendimiento máximo de la última generación. Sin embargo, tal algoritmo tiene complejidad exponencial sobre el número m de generaciones ($O(m) = 2^m$). Para ello, introduciremos un *depredador artificial* cuyo fin será estabilizar la población de organismos, mediante la eliminación directa de los organismos más "débiles", *i.e.*, con menor rendimiento [1].

El número de depredadores y presas puede ser controlado mediante el modelo de Lotka-Volterra. Dado que se trata de un modelo abstracto, para fines prácticos únicamente necesitamos regular la población de organismos presa mediante "depredaciones" periódicas.

El clásico modelo de Predador-Presa de Lotka-Volterra es un sistema de dos ecuaciones diferenciales que representan la interacción ecológica entre dos especies, las cuales toman los roles de depredador y presa. Las ecuaciones están dadas por:

$$\frac{dx}{dt} = af(x) - bg(x,y)y, \tag{4}$$

$$\frac{dy}{dt} = cg(x,y)y - hy, \tag{5}$$

donde $x(t)$ denota la densidad de presas sobre el tiempo t, $y(t)$ es la densidad de depredadores sobre el tiempo, $f(x)$ es la razón de crecimiento poblacional de las presas en ausencia de depredadores, $g(x,y)$ representa a la respuesta funcional del depredador frente a su presa, a es la tasa de crecimiento de la población, b es la tasa de competencia, c representa la tasa de crecimiento del depredador y h a la tasa de muerte del depredador [13]. Considerando un modelo de crecimiento exponencial para la población de presas, se tiene que $f(x) = x$ y también podemos considerar una interacción $g(x,y) = x$.

Para nuestro modelo en particular, podemos tomar otras consideraciones que nos permiten fijar algunos parámetros del modelo. En ausencia de depredadores, el crecimiento de la presa está dado por $\frac{dx}{dt} = ax$. Como la reproducción de nuestros organismos presas se da por fisión binaria, entonces una solución propuesta a la ecuación diferencial es $f(t) = 2^t$, de donde observamos que $a = \ln 2$.

De esta forma, el modelo final está dado por las ecuaciones:

$$\frac{dx}{dt} = \ln 2 \; x - bxy, \tag{6}$$

$$\frac{dy}{dt} = cxy - hy. \tag{7}$$

Una simulación del modelo puede observarse en la figura 3.

[1] Una variante estocástica puede consistir en eliminar a los organismos con probabilidad en función de su rendimiento

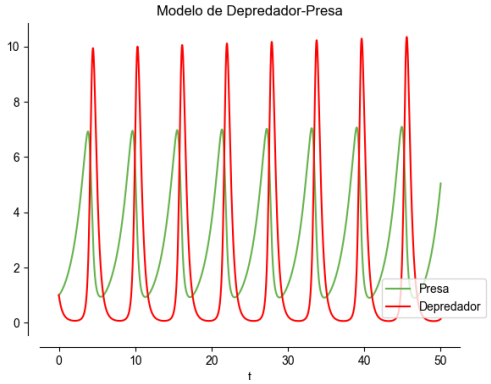

Fig. 3. Simulación del modelo de Lotka-Volterra utilizando método de Euler.

2.3. Discretización del modelo

Los parámetros b, c y h son libres para el sistema de las ecuaciones (5 y 6). Eligiendo adecuadamente los parámetros es posible efectuar una simulación de la dinámica poblacional que implica el aumento de la población presa hasta su declive.

De la simulación podemos observar que las soluciones analíticas del sistema son periódicas y dadas ciertas condiciones iniciales y parámetros predefinidos podemos calcular la amplitud, el periodo T, el máximo M y el mínimo m, dados por $M = \arg\max_{t\in[0,T]} x(t)$ y $m = \arg\min_{t\in[0,T]} x(t)$

La periodicidad de las soluciones es un hecho probado (veáse [14]). Los nuevos parámetros T, m, M son suficientes para realizar una aproximación discreta a la dinámica de la poblacón de presas, que es la que nos interesa. Tal aproximación será dado por los siguientes pasos:

1. Aumentar la población de presas exponencialmente utilizando $f(t) = 2^t$ (fisión binaria) hasta que $t = M$
2. Cuando $t = m$ reducir el número de presas hasta eliminar μ_m.
3. Aplicar $m \leftarrow m + T$ y $M \leftarrow M + T$ y repetir el paso 1.

Aquí $\mu_m = x(m)$, El anterior algoritmo nos permite disponer de una forma aproximada y estable para simular dicha dinámica. Más aún, podemos generalizar los parámetros T, m, M, μ_m, y establecerlos como iniciales en lugar de b, c y h. Una simplificación más extrema, puede fijar $m = M + 1$, de forma que en la siguiente iteración se efectúe la eliminación. De este modo, un algoritmo global para la optimización de los hiperparámetros está dado por el siguiente, siendo T, M, μ_m, t_{max} constantes de entrada:

En el anterior algoritmo, con las constantes de la simplificación del modelo, μ_m controla el número mínimo de presas, M controla el crecimiento inicial y T el crecimiento después de cada reducción. Una menor μ_m implica una mayor

Algoritmo 1 Evolución de redes neuronales con Modelo de Depredador-Presa.

Entradas: Parámetros t_{max}, M, μ_m, Conjuntos de entrenamiento y validación
Salidas: Configuraciones de capas de redes neuronales con mejor rendimiento

Initialization $t \leftarrow 0$
while $t < t_{max}$ **do**
  if $t = M$ **then**
    $t \leftarrow t + 1$
    $\mu \leftarrow$ número de organismos
    Eliminar $\mu - \mu_m$ organismos con menor rendimiento
  else
    Duplicar la población de presas
    Calcular el rendimiento de cada presa
  end if
  $t \leftarrow t + 1$
  $M \leftarrow M + T$
end while

depredación ya que es necesario reducir más presas. Una mayor M permite un mayor crecimiento de presas en el momento inicial y una mayor T permite un mayor crecimiento de presas después de las depredaciones.

2.4. Costo computacional de la solución propuesta

Como se mencionó anteriormente, el costo computacional del modelo (basado en el número de entrenamientos realizados, que es el paso considerablemente más costoso) en ausencia de depredadores está dado por $O(t) = 2^t$, ya que 2^t es el número de entrenamientos. En este contexto, el cálculo de la complejidad está dado por el número de entrenamientos que se realizan por cada generación de organismos. La presencia de depredadores permite reducir notablemente la complejidad del problema.

Las constantes del modelo son T, M, y μ_m, por lo que permanecerán como valores fijos. Si $t \leq M$, $O(t) = 2^t \leq 2^M$, por lo que la complejidad está acotada por una constante. Si $t > M$ se eliminan organismos cada periodo T dejando μ_m. Tras dicha eliminación los organismos, los restantes μ_m se siguen duplicando durante todo el periodo, alcanzando $2^T\mu_m$. Como esto ocurre cada $floor\left(\frac{t-M}{T}\right)$ veces, entonces una cota superior de la complejidad está dada por:

$$O(t) \leq 2^M + 2^T\mu_m\Big(\frac{t-M}{T} + 1\Big). \tag{8}$$

Lo anterior esclarece la importancia de la introducción de depredadores, al acotar de forma polinomial el problema.

3. Resultados

Para verificar la efectividad del algoritmo propuesto[2], se utilizó la base de datos de MNIST, consistente de dígitos manuscritos, dividiendo el dataset en 50000 imágenes para entrenamiento, 10000 para validación y 10000 para prueba. Los ajuste de los pesos se efectuaron minimizando la función de entropía cruzada categórica dispersa utilizando Adam. Todas las neuronas fueron dotadas de activación ReLU, exceptuando la última capa, donde se utilizó softmax. Dadas las características de la base de datos, se utilizó una entrada fija de 28×28 y una salidad de 10 neuronas correspondientes a las 10 clases de dígitos.

Se hicieron pruebas variando las constantes μ_m, M, T del algoritmo, que no son propiamente hiperparámetros (lo cual llevaría a una nueva optimización) sino constantes de calibración que influyen en el número de entrenamientos del modelo y el tiempo de ejecución. Su elección dependerá de las capacidades de los sistemas empleados. Las variantes del algoritmo son las siguientes:

1. $A(k, \mu_m, M, T)$: Algoritmo con k fija.
2. $B(k_{\text{máx}}, \mu_m, M, T)$: Algoritmo con k aleatoria elegida en el intervalo $[1, k_{\text{máx}}]$.
3. $C(k_{\text{máx}}, \mu_m, M, T)$: Algoritmo con k aleatoria elegida en $[1, k_{\text{máx}}]$ y utilizando una red neuronal convolucional de soporte.

La variante de tipo A está relacionada con la búsqueda por rejilla, ya que realiza aumentos graduales pero en múltiples direcciones y de forma controlada por las depredaciones, mientras que la variante de tipo B introduce el concepto de aleatoriedad descrito anteriormente para la búsqueda por rejilla. De acuerdo con la literatura [5,4], esta modificación podría mostrar resultados beneficiosos.

Por otro lado, la variante de tipo C involucra la introducción de una CNN seguida de una FNN. Dado que el trabajo propuesto se centra en la optimización de FNNs, las capas convolucionales y de pooling se mantuvieron constantes. Para optimizar dichas capas, es posible recurrir a las técnicas referenciadas en los Antecedentes. La red propuesta cuenta con la siguiente arquitectura:

1. Una capa convolucional con 169 filtros con kernels de tamaño 4×4 y activación ReLU. La entrada es de $28 \times 28 \times 1$.
2. Una capa convolucional con 36 filtros con kernels de 3×3 y activación ReLU.
3. Una capa de MaxPooling de 2×2.
4. Una capa densa de 30 unidades y activación ReLU.
5. Una capa de salida con 10 neuronas con activación Softmax.

La capa densa conforma a la red completamente conectada cuya arquitectura optimizarse. Dado que iniciamos con la neurona con código (30) entonces utilizaremos mutaciones γ. Para los otros casos, al iniciar en (0), solamente usaremos mutaciones α y β.

Para la red convolucional, se utilizó como optimizador a la entropía cruzada categórica, en lugar de la categórica dispersa. Además, para entrenar, debido al

[2] Los códigos están provistos en `https://github.com/Pherjev/NN-Evolutivas-Lotka-Volterra`

Tabla 1. Resumen de los entrenamientos indicando la configuración del entrenamiento, la arquitectura óptima calculada, la validación obtenida y la exactitud sobre el conjunto de prueba, así como el tiempo de ejecución empleado en segundos.

Algoritmo	Arquitectura	Validación	Prueba	Tiempo
$A(1, 9, 4, 3)$	(24)	0.9506	0.9622	2233.99039006
$A(30, 9, 4, 3)$	$(150, 60)$	0.9732	0.9806	1661.98341298
$A(30, 20, 6, 3)$	$(150, 90)$	0.9738	0.9788	3051.702986
$B(70, 9, 4, 3)$	$(271, 131)$	0.9768	0.9801	3053.07773709
$B(70, 20, 6, 3)$	(362)	0.9767	0.9837	6803.21170092
$C(30, 3, 2, 1)$	$(30, 33)$	0.9837	0.9896	6742.15097094

costo computacional, se empleó una iteración de entrenamiento en los algoritmos tipo C y dos iteraciones para los tipos A y B. De esta forma, se realizaron entrenamientos rápidos para muestrear la exactitud con escasas épocas.

Para probar el rendimiento final, se utilizó un entrenamiento con el conjunto de prueba usando 50 épocas. Todos las evoluciones se efectuaron usando 15 generaciones, excepto para $k = 1$, donde se emplearon 60 debido a que no mostraba una convergencia aún hacia un máximo local y su tiempo de ejecución en general fue menor. Un resumen de los resultados aparece en la tabla 1.

Tal como esperado, el algoritmo tipo C tuvo mejor exactitud tanto en la validación como en la prueba. Igualmente anticipado pero notorio es que el uso de aleatoriedad en B mostró mejoras significativas al rendimiento, aún mejores que aumentando μ_m, M, T, cambio que se reflejó más en el tiempo de ejecución y reportó poca o ninguna mejora. Ejemplos de gráficas de evolución de la exactitud y número de individuos se proporcionan a continuación.

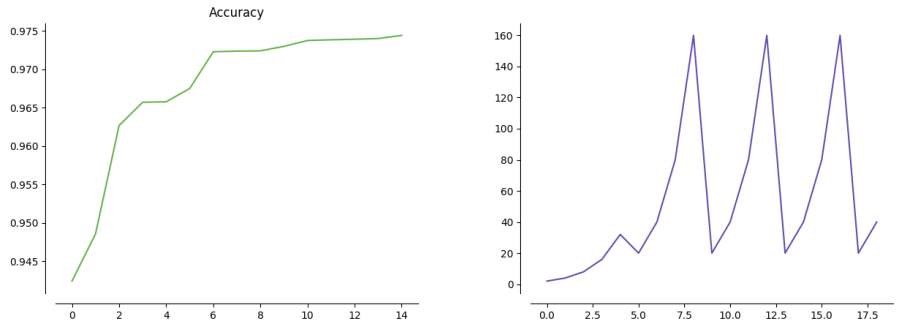

Fig. 4. Mejora de la exactitud (eje y, izquierda) por generación (eje x, izquierda), así como número de individuos en $A(30, 20, 6, 3)$. La diferencia entre los números de individuos de ambas gráficas radica en que se introduce una generación de redución para la gráfica del número de individuos.

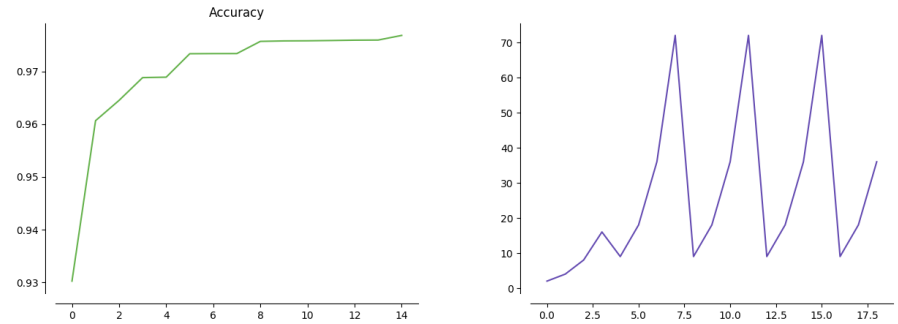

Fig. 5. Mejora de la exactitud (eje y, izquierda) por generación (eje x, izquierda), así como número de individuos en $B(70, 9, 4, 3)$.

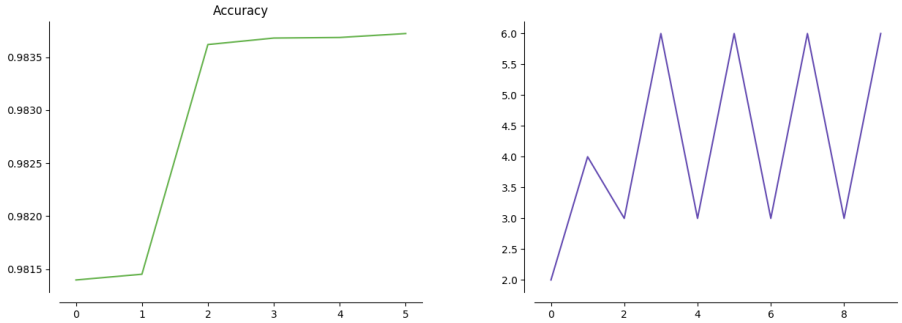

Fig. 6. Mejora de la exactitud (eje y, izquierda) por generación (eje x, izquierda), así como número de individuos en $C(30, 3, 2, 1)$.

3.1. EMNIST

Observando los resultados dados en la clasificación MNIST, se hicieron experimentos utilizando la base de datos de caracteres alfabéticos EMNIST propuesta por [9], utilizando 104000 imágenes para entrenamiento, 20800 para prueba y 20800 para validación. Se empleó la arquitectura convolucional propuesta anteriormente, efectuando el método de evolución $C(30, 3, 2, 1)$, el cual aportó los mejores resultados para MNIST. La evolución arrojó una arquitectura de (104), que amplía la propuesta original de (30) obteniendo una exactitud de validación de 0,9167. Se hizo un entrenamiento con 10 épocas, alcanzando 0,9263 de exactitud sobre el conjunto de prueba.

Un entrenamiento con el modelo originalmente propuesto arrojó 0,9209 de exactitud, que supera al 0,8515 logrado en el artículo original utilizando la técnica de *Online Pseudo-Inverse Update Method*. Una segunda red convolucional, basada en las arquitecturas VGG [20] fue implementada utilizando dos capas convolucionales de 64 filtros de 3×3, seguido de un MaxPooling de 2×2, luego otras dos capas convolucionales de 128 filtros de 3×3 y otro MaxPooling de 2×2 así como una capa densa de 256 neuronas. Esta arquitectura entrenada con 10 épocas logró una exactitud de 0.9379. Una evolución con los parámetros $C(70, 2, 2, 1)$ arrojó una estructura densa de $(361, 85)$, mejorando la exactitud a 0.9399 sobre el conjunto de prueba.

4. Conclusiones

En este proyecto se propuso un algoritmo de optimización hiperparamétrica centrada a la arquitectura de las redes FNN siguiendo una inspiración basada en el modelo de Lotka-Volterra sobre relación ecológica de depredadores y presas, dando a las redes neuronales un rol de presas frente a un depredador latente encargado de controlar su población. En términos de optimización, el rol del depredador se traduce en evitar una explosión combinatoria que supone permitir la reproducción no acotada de las presas.

En este artículo presentamos tres propuestas para realizar la optimización hiperparamétrica en la arquitectura de las capas densas: la primera consiste en realizar k aumentos en las capas para generar redes hijas, la segunda introduce aleatoriedad en la k, y la tercera, de forma separada, involucra incorporar alguno de estos algoritmos para modificar las capas densas presentes en una red neuronal convolucional dada. Para los problemas de clasificación que se abordaron (MNIST y EMNIST para caracteres), el uso de redes convolucionales es una opción más adecuada, lo cual se reflejó en los resultados.

En los casos aplicados fue posible mejorar el rendimiento de las redes, obteniendo arquitecturas optimizadas tanto partiendo de una red sin capas intermedias como desde una serie de capas densas ya definidas. En general, el uso de una k aleatoria arrojó mejores resultados que utilizando una k fija, tal como sugiere la literatura. Asimismo, para los problemas que se abordaron, fue conveniente utilizar una red inicial convolucional que variara las capas densas. Una observación relevante es que el cambio de las capas densas si bien reportó mejorías por medio de la evolución, se notó un aumento más fuerte de la exactitud mediante el empleo de capas convolucionales distintas, lo cual nos inclina a considerar incorporar un algoritmo evolutivo para las capas convolucionales o emplear las sugerencias presentadas en la literatura.

Nuestra propuesta, hasta ahora enfocada en las capas densas, el cual nos ha permitido obtener resultados beneficiosos en ambas bases de datos, tiene la convergencia hacia un mínimo local asegurada (véase Apéndice) para el caso $k = 1$, además de disponer de un costo computacional relativamente bajo que nos permite realizar varias pruebas sin llegar a una explosión combinatoria. Para términos prácticos, reuniendo tanto resultados teóricos como experimentales,

consideramos recomendable la realización de entrenamientos utilizando k aleatorias y realizar un ajuste final utilizando $k = 1$. Esta técnica evolutiva, aunque centrada en el caso de las redes FNNs, podría extrapolarse a otros contextos tanto de redes neuronales como para aspectos aún más generales[3].

Apéndice

Convergencia del algoritmo para k = 1

En esta sección probaremos la convergencia del algoritmo para $k = 1$ hacia un máximo local. Si bien el resultado no se generalizó para $k > 1$, podemos ajustar $k = 1$ en alguna iteración determinada. Para la formalización de la prueba, definamos el espacio ℓ_f de las sucesiones naturales finitas dado por:

$$\ell_f = \{(s_n)_{n\in\mathbb{N}} \mid s_n \in \mathbb{N}, \exists m : s_m = 0\}. \tag{9}$$

Los elementos de ℓ_f son las configuraciones de las redes neuronales. Definamos también una función evaluación que mide la exactitud de la red, dada por $e : \ell_f \to [0, 1]$, por ejemplo, exactitud sobre un conjunto de prueba fijo. Queremos mostrar que nuestro algoritmo converge para el caso $k = 1$ hacia un máximo local de e. En realidad esa situación puede no ocurrir, pero es posible garantizar la convergencia *casi seguramente*. En general, e puede tener no tener un máximo local, en cuyo caso, la sucesión de búsqueda se mantendrá creciente. En cada iteración i del algoritmo, tomamos un subconjunto finito $\mathcal{C}_i$ de ℓ_f y lo devuelve $\mathcal{C}_{i+1} \subset \ell_f$, también finito. Definamos la sucesión dada por:

$$m_i = \text{máx}\{e((s_n))|(s_n) \in \mathcal{C}_i\}. \tag{10}$$

Afirmamos que esta sucesión converge casi seguramente a un máximo local. Primeramente demostraremos que la sucesión (m_i) es no decreciente.

Lema 1 (m_i) *es no decreciente.*

Demostración. Sea $i \in \mathbb{N}$. Entonces como $\{e((s_n))|(s_n) \in \mathcal{C}_i\}$ es un conjunto finito, existe $(s_n) \in \mathcal{C}_i$ tal que $m_i = e((s_n))$. Si en la iteración $i + 1$ se duplican los individuos, (s_n) se conserva. En caso de que reduzcan, dado que (s_n) tiene la mejor exactitud, solamente puede ser eliminado si existe $(r_n) \in \mathcal{C}_i$ tal que $e((r_n)) = m_i$. Por lo tanto, en todos los casos $(s_n) \in \mathcal{C}_{i+1}$, o bien, $(r_n) \in \mathcal{C}_{i+1}$ de donde $m_i \leq m_{i+1}$ □

El Lema 1 se cumple para cualquer $k \in \mathbb{N}$. Para $k = 1$, debemos asegurar en nuestro algoritmo que si $(s_i) \in \ell_f$ cumple que $e((s_i)) = m_i$, entonces (s_i) no será depredado. Dada esta condición particular, podemos probar la convergencia casi segura:

[3] Agradecimientos al Dr. Carlos Francisco Brito-Loeza por el seguimiento de la investigación

Proposición 1 *Supongamos que la función e tiene un máximo global. Entonces* (m_i) *converge casi seguramente a un máximo local.*

Demostración. Por el Lema 1, la sucesión (m_i) es no decreciente. Ahora, veamos que si $m_i = m_j$ para $j > i$, entonces el algoritmo ha convergido a un máximo local. Por contradicción, supongamos que existe $(r_n) = \arg\max_{(s_n)\in\mathcal{C}_i} c((s_n))$, que no es máximo local, tal que $m_i = m_j$ para toda $j > i$. Como (r_n) no es máximo local, entonces existe (q_n) tal que $e((q_n)) > e((r_n))$ y $|(q_n) - (r_n)| = 1$, puesto que se tratan de sucesiones de naturales diferentes, por lo que difieren en solamente un punto. Notemos que $(r_n) \in \mathcal{C}_j$ para toda $j > i$. Por lo tanto, como $(q_n), (r_n) \in \ell_f$, ambas sucesiones son finitas. Si ambas se hacen 0 en el mismo índice, entonces se tiene que difieren en el índice t, de forma que:

$$(r_n) = (r_n^1, \dots, r_n^t, \dots, r_n^d, 0, \dots), \tag{11}$$

$$(q_n) = (r_n^1, \dots, r_n^t + 1, \dots, r_n^d, 0, \dots). \tag{12}$$

Sea p_α la probabilidad de ocurrencia de la mutación tipo α. La probabilidad de que no ocurra el evento r_n muta a q_n es de $p_\alpha \frac{d-1}{d} + p_\beta$. Por lo tanto, la probabilidad de que no ocurra i veces está dada por $(p_\alpha \frac{d-1}{d})^i$, de donde:

$$\lim_{i\to\infty} \left(p_\alpha \frac{d-1}{d} + p_\beta\right)^i = 0, \tag{13}$$

ya que la constante $p_\alpha \frac{d-1}{d} + p_\beta < 1$. Si ambas difieren en el índice en el que se hacen 0, entonces se tiene que:

$$(r_n) = (r_n^1, \dots, r_n^d, 0, 0, \dots), \tag{14}$$

$$(q_n) = (r_n^1, \dots, r_n^d, 1, 0, \dots), \tag{15}$$

de donde la probabilidad evento de la mutación de (r_n) a q_n es p_β, por lo que la probabilidad de que no ocurra es p_α, por lo que la probabilidad de que no ocurra i veces está dada por $(p_\alpha)^i$, de donde $\lim_{i\to\infty}(p_\alpha)^i = 0$.

Por lo tanto, (r_n) muta a (q_n) casi seguramente en alguna iteración $j > i$, de donde $(r_n) \in \mathcal{C}_j$ y por lo tanto $m_j \geq e((r_n)) > m_i$, que contradice nuestra hipótesis original. Por lo tanto, si el algoritmo se estanca, ha llegado a un mínimo local.

En el paso anterior vimos que (m_i) crece casi seguramente si se estaciona en un máximo local. Por lo tanto es de la forma:

$$m_1 = \cdots m_{1+o_1} < m_2 = \cdots . \tag{16}$$

Por lo que tenemos una subsucesión estrictamente creciente. Como existe un máximo global y el dominio es discreto, no puede crecer indefinidamente. Por lo tanto (m_i) converge a un máximo local □.

Referencias

1. Aggarwal, C.C.: Neural networks and deep learning. Springer 10, 978–3 (2018)
2. Aguilar Canto, F.J.: Eficacia de diferentes reglas hebbianas en el aprendizaje supervisado. Tecnología Educativa Revista CONAIC 7(1), 92–97 (2020)
3. Baldominos, A., Saez, Y., Isasi, P.: Model selection in committees of evolved convolutional neural networks using genetic algorithms. In: International Conference on Intelligent Data Engineering and Automated Learning. pp. 364–373. Springer (2018)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research 13(Feb), 281–305 (2012)
5. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. pp. 2546–2554 (2011)
6. Bhandare, A., Kaur, D.: Designing convolutional neural network architecture using genetic algorithms. In: Proceedings on the International Conference on Artificial Intelligence (ICAI). pp. 150–156. The Steering Committee of The World Congress in Computer Science (2018)
7. Bochinski, E., Senst, T., Sikora, T.: Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In: 2017 IEEE International Conference on Image Processing (ICIP). pp. 3924–3928. IEEE (2017)
8. Botev, A., Lever, G., Barber, D.: Nesterov's accelerated gradient and momentum as approximations to regularised update descent. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 1899–1903. IEEE (2017)
9. Cohen, G., Afshar, S., Tapson, J., Van Schaik, A.: Emnist: Extending mnist to handwritten letters. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 2921–2926. IEEE (2017)
10. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems 2(4), 303–314 (1989)
11. Dayan, P., Abbott, L.F.: Theoretical neuroscience: computational and mathematical modeling of neural systems (2001)
12. Eldan, R., Shamir, O.: The power of depth for feedforward neural networks. In: Conference on learning theory. pp. 907–940 (2016)
13. Garain, K., Kumar, U., Mandal, P.S.: Global dynamics in a Beddington–DeAngelis Prey–Predator model with density dependent death rate of predator. Differential Equations and Dynamical Systems pp. 1–19 (2019)
14. Hadžiabdić, V., Mehuljić, M., Bektešević, J.: Lotka-Volterra model with two predators and their prey. Tem Journal 6(1), 132–136 (2017)
15. Hornik, K., Stinchcombe, M., White, H., et al.: Multilayer feedforward networks are universal approximators. Neural networks 2(5), 359–366 (1989)
16. Kavitha, K., Ramakrishnan, K., Singh, M.K.: Modeling and design of evolutionary neural network for heart disease detection. International Journal of Computer Science Issues (IJCSI) 7(5), 272 (2010)
17. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
18. Rocha, M., Cortez, P., Neves, J.: Evolutionary neural network learning. In: Portuguese Conference on Artificial Intelligence. pp. 24–28. Springer (2003)
19. Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65(6), 386 (1958)

20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
21. van Wyk, G.J., Bosman, A.S.: Evolutionary neural architecture search for image restoration. In: 2019 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2019)
22. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments. pp. 1–5 (2015)