Un Primer Acercamiento al Aprendizaje del Código Genético usando Redes Neuronales

Natalia L. Weinbach[1] nlw@cs.uns.edu.ar

Carlos I. Chesñevar[1,2] cic@eps.udl.es

[1]Laboratorio de Investigación en Inteligencia Artificial (LIDIA)
Departamento de Ciencias e Ingeniería de la Computación — Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
Tel: (0291) 459-5135 / Fax: (0291) 459-5136

[2]Grupo de Investigación en Inteligencia Artificial
Departament d'Informàtica — Universitat de Lleida
Campus Cappont – C/Jaume II, 69 – E-25001 Lleida, Catalunya, España
Tel/Fax: (+34) (973) 70 2718 / 2702

Resumen

La bioinformática constituye un área interdisciplinaria que ha tenido un fuerte desarrollo en los últimos años. Dentro de dicha área, el aprendizaje del código genético a través de modelos computacionales apropiados constituye una problemática de gran importancia. El objetivo de este trabajo es presentar un primer acercamiento al aprendizaje del código genético a través de redes neuronales utilizando la herramienta de software Weka. Dentro del aprendizaje del código genético, el problema concreto a resolver es la clasificación de cada uno de los 61 codones como alguno de los 20 aminoácidos. Para esta tarea, las redes con tres y cuatro unidades intermedias son relativamente fáciles de entrenar; es más difícil obtener redes perfectas con sólo dos unidades intermedias. Se observó que para esta tarea el esquema de entrenamiento de Backpropagation convencional no resulta efectivo. En este trabajo se muestran los resultados de los algoritmos de entrenamiento modificados y se los contrasta con resultados anteriores en el área.

Palabras clave: Aprendizaje Automatizado. Redes Neuronales. Bioinformática. Código Genético. Weka.

1. Introducción y motivaciones

La bioinformática constituye un área interdisciplinaria que ha tenido un fuerte desarrollo en los últimos años. Dentro de dicha área, el aprendizaje del código genético a través de modelos computacionales apropiados constituye una problemática de gran importancia. El objetivo de este trabajo es presentar un primer acercamiento al aprendizaje del código genético a través de redes neuronales utilizando la herramienta de software Weka [14, 15].

En las aplicaciones de redes neuronales, es común distinguir entre *regresión* y *clasificación*. En los problemas de regresión, el objetivo es aproximar o ajustar una superficie determinada. En los problemas de clasificación o reconocimiento, el objetivo es clasificar una entrada dada en un número relativamente pequeño de clases. El problema de aprender el código genético es un buen ejemplo de un problema que se encuentra en la frontera de estas dos clases de problemas.

En nuestro caso haremos uso de una red neuronal para intentar que ésta aprenda la estructura del código genético, o más precisamente, va a clasificar un *codón*[1] dentro de un conjunto de 20 aminoácidos posibles. La experimentación se llevará a cabo a través de una implementación en Weka [14], utilizando planillas de cálculo auxiliares para un análisis más enriquecedor de los resultados, incluyendo la evolución de la red. También se plantearán las limitaciones encontradas y se propondrán alternativas para mitigarlas.

El resto de este trabajo está estructurado como sigue. En la Sección 2 se presentan conceptos fundacionales sobre las redes neuronales y su entrenamiento a través de distintos algoritmos alternativos. La Sección 3 presenta conceptos básicos de bioinformática. En la Sección 4 se presenta la contribución principal del artículo, mostrando cómo utilizar una red neuronal para modelar el código genético y el problema presentado. Se contrastan los resultados obtenidos con el trabajo existente en la literatura. Finalmente, la Sección 5 presenta las principales conclusiones obtenidas.

2. Redes Neuronales Artificiales

2.1. Conceptos básicos

Las redes neuronales artificiales (ANNs)[2] [8] fueron desarrolladas originalmente con el objetivo de modelar el procesamiento de información y el aprendizaje en el cerebro humano. Aunque la metáfora cerebral se considera una útil fuente de inspiración, es claro que las neuronas artificiales de las ANNs son algo lejanas a su contraparte biológica. No obstante, el desarrollo de redes neuronales motivó un gran número de aplicaciones prácticas en varios campos de la ciencia, incluyendo la biología molecular. Las redes neuronales se han convertido en una herramienta importante dentro de las técnicas de aprendizaje automatizado que pueden ser aplicadas al análisis de secuencias y problemas de reconocimiento de patrones.

Conceptualmente las ANNs consisten de *neuronas* y *conexiones* entre las entradas y salidas de las mismas. Cada conexión con otras neuronas tiene un peso asociado. La idea es que la red neuronal aprenda a asociar entradas con salidas adaptando estos pesos.

2.2. Mecanismo de entrenamiento para ANNs

A fin de que una red neuronal adapte una configuración determinada de conexiones y pesos asociados a éstas, necesita ser *entrenada* a partir de un cierto conjunto de datos de entrenamiento. Se distinguen distintos acercamientos para entrenamiento de redes neuronales en función de su complejidad. Resumiremos seguidamente algunos de los aspectos principales asociados a esta tarea. El tipo más simple de red neuronal está basado en una unidad llamada *perceptrón*. Un perceptrón toma un vector de entradas de valor real, calcula una combinación lineal de esas entradas, y produce una salida igual a 1 si el resultado es mayor que un umbral, de lo contrario produce un -1. El perceptrón también puede no tener umbral, de modo que la salida o se vuelve continua de la forma:

$$o = w_0 + w_1 x_1 + \ldots + w_n x_n$$

[1] *Biol.* Terna que, en un ARN mensajero, codifica la incorporación de aminoácidos específicos en la biosíntesis de proteínas.

[2] Por su sigla en inglés, *Artificial Neural Networks*.

Existen dos enfoques populares aplicados al entrenamiento de un perceptrón. Uno de ellos es la *regla de entrenamiento* del perceptrón y otro la regla *delta*. Estos dos algoritmos garantizan la convergencia a distintas hipótesis aceptables, bajo ciertas condiciones. Son importantes para las ANNs porque proveen la base para el aprendizaje de redes de muchas unidades. La regla de entrenamiento del perceptrón puede fallar en su convergencia si los ejemplos no son linealmente separables. En consecuencia, existe una segunda regla (la regla Delta) que permite vencer esta limitación. Si los ejemplos no son linealmente separables, la regla Delta converge a la aproximación *best-fit* del concepto meta.

La idea del algoritmo de regla Delta es ir recorriendo el espacio de hipótesis de posibles vectores de peso, buscando el vector que mejor se ajuste a los ejemplos. Esta regla es importante porque sirve como base para el algoritmo de *Backpropagation*, que puede aprender redes con varias unidades interconectadas. Lo que se busca es entrenar la red tal que minimicen el error cuadrado asociado al vector w de pesos, esto es:

$$E[\vec{w}] = E[w_1, \ldots, w_n] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{1}$$

donde D es el conjunto de ejemplos de entrenamiento
t_d es la salida meta para el ejemplo $d \in D$
o_d es la salida de la unidad lineal para el ejemplo d

El algoritmo completo aplicado se basa en el conocido método de *descenso del gradiente*. Mayores detalles pueden encontrarse en [8].

El algoritmo de **Backpropagation** es ampliamente utilizado para entrenar ANNs, dado que aprende los pesos para una red *multicapa*, dada una red con un conjunto fijo de unidades e interconexiones. Emplea el descenso del gradiente para intentar minimizar el error cuadrado entre los valores de salida de la red y los valores meta. Como ahora se consideran redes con múltiples unidades de salida en vez de unidades simples, se redefine el error E para que sume los errores de todas las unidades de salida de la red:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in salidas} (t_{kd} - o_{kd})^2 \tag{2}$$

donde *salidas* es el conjunto de las unidades de salida de la red, y t_{kd} y o_{kd} son los valores meta y de salida asociados con la k-ésima unidad de salida y el ejemplo de entrenamiento d. La diferencia para el caso de redes multicapa es que la superficie de error puede tener múltiples mínimos locales. Desafortunadamente, esto significa que el descenso del gradiente garantiza la convergencia hacia un mínimo local, y no necesariamente al error mínimo global. Hay varias estrategias para alivianar esto:

- Utilizar un *momentum* en la regla de actualización de pesos que nos desvíe de un mínimo local.
- Uso de descenso estocástico.
- Entrenar múltiples redes con los mismos conjuntos de datos, pero inicializarlas con distintos valores. Luego se puede utilizar algún esquema de votación para elegir qué pesos utilizar.

El ciclo de actualización de pesos de Backpropagation puede ser iterado miles de veces en una aplicación típica. Existen varias condiciones de terminación posibles. Se puede elegir un número fijo de iteraciones o pasos, o se puede parar cuando el error en los ejemplos de entrenamiento cae debajo de cierto umbral, o cuando el error en un conjunto de validación separado reúna ciertas condiciones. El criterio de terminación es importante porque pocas iteraciones pueden fallar en reducir el error, y demasiadas pueden llevar a un *overfitting* de los datos de entrenamiento.

2.3. Dominio de problemas apropiados para ANNs

Tal como se señala en [8], los métodos de aprendizaje basados en redes neuronales proveen un enfoque robusto para aproximar funciones reales, discretas o vectorizadas. Para ciertos tipos de problemas, como la interpretación de datos complejos obtenidos por sensores del mundo real, las redes neuronales se ubican entre los métodos de aprendizaje más efectivos actualmente conocidos. Las redes neuronales son adecuadas en problemas donde los datos de entrenamiento provienen de información compleja de sensores que pueden estar afectados por ruido, como una cámara o micrófono. Son también aplicables a problemas para los cuales se suelen utilizar representaciones más simbólicas, como las tareas de aprendizaje de los árboles de decisión.

El algoritmo de entrenamiento más popular es el de *Backpropagation*, que resulta apropiado para problemas con las siguientes características [8]:

- Las instancias o ejemplos están representados como varios pares atributo-valor.
- La salida es un valor discreto o real, o un vector de valores de estos tipos.
- Los ejemplos de entrenamiento pueden contener errores (son robustas).
- Se consideran aceptables largos tiempos de entrenamiento.
- La forma de la función meta es desconocida y la habilidad de los seres humanos para enterderla no es importante.

2.4. Arquitecturas posibles

En muchas de las aplicaciones actuales de redes neuronales en biología molecular, las arquitecturas utilizadas son *layered feed-forward*, como se ilustra en la Figura 1. Las capas pueden contener distinto número de unidades; los patrones de conectividad entre las capas también pueden variar. El comportamiento de cada unidad en el tiempo puede ser descrito utilizando ecuaciones diferenciales o ecuaciones discretas. En una arquitectura layered feed-forward, todas las unidades en una capa son actualizadas simultáneamente, y las capas son actualizadas secuencialmente en el orden natural.

3. Bioinformática. Definición y Desafíos Actuales

La bioinformática consta de los métodos matemáticos, estadísticos y computacionales que tienen la intención de resolver problemas biológicos utilizando secuencias de ADN y aminoácidos e información relacionada. Según el NCBI[3], la bioinformática es el campo de la ciencia en el

[3]National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov/

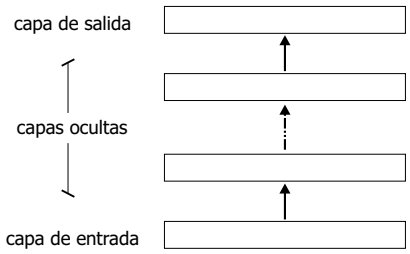

Figura 1: Arquitectura *Layered Feed-Forward* o Perceptrón Multicapa.

cual la biología, las ciencias de la computación y la tecnología de la información se combinan en una única disciplina. Se distinguen tres sub-disciplinas importantes dentro de la bioinformática:

- el desarrollo de nuevos algoritmos y estadísticas con los cuales se evalúan las relaciones entre los miembros de grandes conjuntos de datos;
- el análisis y la interpretación de variados tipos de datos, incluyendo secuencias de aminoácidos y estructuras de proteínas; y
- el desarrollo e implementación de herramientas que permitan un acceso eficiente y un manejo de los distintos tipos de información.

Otras definiciones alternativas coinciden en presentar a la bioinformática como la investigación, desarrollo o aplicación de herramientas y enfoques computacionales para expandir el uso de datos biológicos, médicos, del comportamiento o de la salud; incluyendo los mecanismos para adquirir, almacenar, organizar, archivar, analizar, o visualizar tales datos.

Aminoácidos: su rol en el Código Genético

Dentro de la bioinformática, el estudio y formalización del código genético de los seres vivos ha jugado un papel fundamental. En este marco se encuentra en desarrollo el conocido proyecto Genoma Humano, iniciado en 1986[4]. En un marco más general, desde que el código genético fue descubierto se han hecho numerosos intentos para revelar su simetría potencial subyacente y su historia de evolución. En este contexto, las propiedades de los 20 aminoácidos presentes en dicho código y las similaridades entre ellos han jugado un rol determinante en este tipo de análisis. Los aminoácidos son las unidades elementales constitutivas de las moléculas denominadas proteínas. Se caracterizan por poseer un grupo carboxilo (-COOH) y un grupo amino ($-NH_2$). Son pequeños elementos con los cuales el organismo reconstituye permanentemente sus proteínas específicas. Se sabe que de los 20 aminoácidos proteicos conocidos, 8 resultan indispensables para la vida humana.

Los aminoácidos están formados por 4 tipos distintos de bases nitrogenadas, que en el ARN (ácido ribonucléico) son: adenina (A), guanina (G), uracil (U) y citosina (C). Los aminoácidos se presentarán en forma de *codón*, que es una terna que, en un ARN mensajero, codifica la incorporación de aminoácidos específicos en la biosíntesis de proteínas. Cabe destacar que el ARN es el principal material genético presente en los organismos llamados virus, siendo también importante en la producción de proteínas en otros organismos vivos. El ARN puede moverse

[4]También conocido por su denominación en inglés (*Human Genome Project* o HGP).

alrededor de las células de los organismos vivos y por consiguiente sirve como una suerte de "mensajero genético", transmitiendo la información guardada en el ADN (ácido desoxiribonucléico) de la célula, desde el núcleo hacia otras partes de la célula donde se usa para ayudar a producir proteínas.

4. Modelaje del Código Genético a través de Redes Neuronales

Las asignaciones de codón tienen correlación con las propiedades físicas de los aminoácidos de manera sistemática. Las tres posiciones en las cadenas se asocian a características considerablemente diferentes entre los aminoácidos. La primer posición en la cadena se relaciona con los caminos biosintéticos de los aminoácidos y con su evolución. La segunda posición se relaciona con las propiedades de los aminoácidos asociadas al agua, y la tercera posición se relaciona al peso molecular o tamaño de los aminoácidos [12]. Estas características se utilizan como correctoras de errores de dos maneras. En primer lugar, la degeneración se correlaciona con la abundancia del aminoácido en las proteínas, lo que disminuye la chance de una mutación. En segundo lugar, se observó que aminoácidos parecidos tienen codones similares, y esto disminuye las posibilidades de que una mutación tenga un efecto riesgoso sobre la estructura de la proteína resultante.

Es importante notar que en el enfoque de redes neuronales para el estudio de la estructura del código genético, el análisis se independiza de cualquier tipo de conjetura previa, al ser completamente conducido por los datos de entrenamiento utilizados. La red neuronal infiere la estructura subyacente directamente a partir del mapeo entre los codones y los aminoácidos de igual forma como se presenta en el código genético estándar (ver cuadro 1). En efecto, al utilizar redes neuronales para abordar el problema no se introducen relaciones *a priori* entre los nucleótidos[5] o aminoácidos.

En este marco de referencia, en una red que aprenda el código genético habrá una capa de entrada que recibe una terna de nucleótidos (A, C, G o U) y arroja como salida el aminoácido correspondiente. Luego, las 64 diferentes ternas son posibles como entrada, y en la salida aparecen alguno de los 20 aminoácidos posibles. Las combinaciones UUA, UAG y UGA serán denominados "codones de parada", es decir, no corresponden a ningún aminoácido en especial. La red neuronal construida con dos capas ocultas es la que se muestra en la Figura 2.

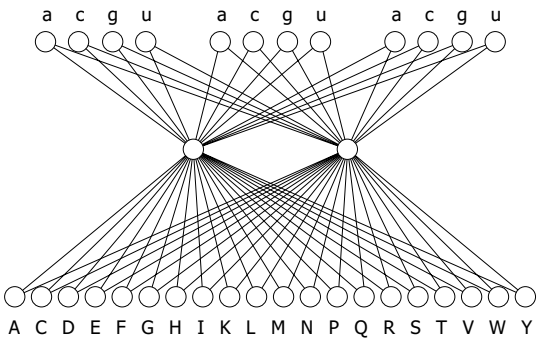

Figura 2: Arquitectura de una ANN entrenada para aprender el código genético.

[5] *Bioquím.* Compuesto orgánico constituido por una base nitrogenada, un azúcar y ácido fosfórico. Según el azúcar sea la ribosa o la desoxirribosa, el nucleótido resultante se denomina ribonucleótido o desoxirribonucleótido.

Aminoácido			Codones
F	(Phe)	Fenilalanina	UUU UUC
M	(Met)	Metionina	AUG
I	(Ile)	Isoleucina	AUU AUC AUA
L	(Leu)	Leucina	UUA UUG CUU CUC CUA CUG
V	(Val)	Valina	GUU GUC GUA GUG
C	(Cys)	Cisteína	UGU UGC
W	(Trp)	Triptófano	UGG
A	(Ala)	Alanina	GCU GCC GCA GCG
T	(Thr)	Treonina	ACU ACC ACA ACG
G	(Gly)	Glicina	GGU GGC GGA GGG
S	(Ser)	Serina	UCU UCC UCA UCG AGU AGC
P	(Pro)	Prolina	CCU CCC CCA CCG
Y	(Tyr)	Tirosina	UAU UAC
H	(His)	Histidina	CAU CAC
Q	(Gln)	Glutamina	CAA CAG
N	(Asn)	Asparagina	AAU AAC
E	(Glu)	Ácido Glutamínico	GAA GAG
K	(Lys)	Lisina	AAA AAG
D	(Asp)	Ácido Aspártico	GAU GAC
R	(Arg)	Arginina	CGU CGC CGA CGG AGA AGG

Cuadro 1: Los 20 aminoácidos

Las redes con tres y cuatro unidades intermedias son relativamente fáciles de entrenar; es más difícil obtener redes perfectas (esto es, con un poder de predicción que tenga exactitud absoluta) con sólo dos unidades intermedias. La única forma en que se pueden encontrar redes mínimas (con dos unidades intermedias) es por medio de un procedimiento de entrenamiento adaptable. Al menos para esta tarea, se observó que el esquema de entrenamiento de Backpropagation convencional (que trata a todos los ejemplos de entrenamiento de la misma manera) falla en encontrar una red mínima cuya existencia es demostrable [1].

En la Figura 3 se puede ver la diferencia en la cantidad de instancias bien clasificadas, en redes de 2, 3 y 4 capas. En el ejemplo se tomó un conjunto de validación con 61 ejemplos, conteniendo todas las combinaciones de A, G, C, U en los aminoácidos. Con una tasa de entrenamiento (*learning rate*) baja, en un principio hay más aciertos, pero se necesitan largos períodos de entrenamiento para obtener mejores predicciones. La figura permite apreciar cómo una red de 4 capas obtiene una predicción perfecta en 5000 pasos de entrenamiento, con $\eta = 0{,}4$. En el ensayo con 5000 pasos en la red de 2 capas se ve cómo se presenta el fenómeno de sobreajuste (*overfitting*); para mejorar esta red se necesita modificar el algoritmo de entrenamiento.

La técnica estándar para entrenamiento de redes feed-forward es Backpropagation, y de hecho es la que implementa Weka en su clasificador `MultilayerPerceptron` [15]. En la literatura se han propuesto distintas estrategias de entrenamiento para obtener bajos errores de clasificación. Una modificación simple pero eficiente consiste en utilizar una tasa de aprendizaje η alta para ejemplos incorrectamente clasificados y una tasa baja para los ejemplos correctamente clasificados. Otro procedimiento efectivo es modificar las frecuencias de presentación de las diferentes categorías para obtener una situación más balanceada. En el caso del código genéti-

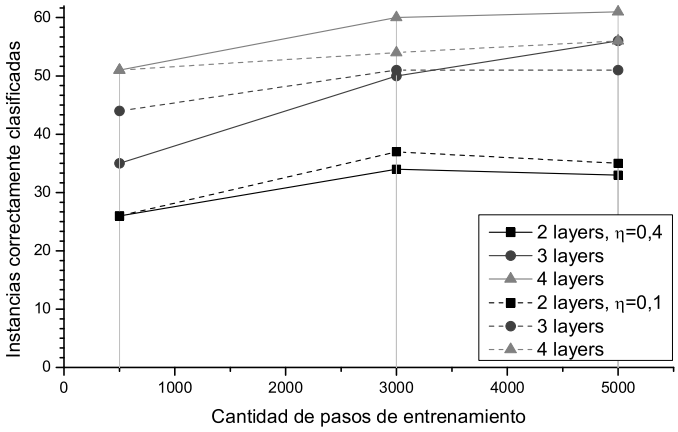

Figura 3: Diferencias en el entrenamiento de redes con 2, 3 y 4 capas.

co, esto implica que para cada aminoácido debe presentarse el mismo número de codones sin importar cuántas veces aparecía en el código original. Así, por ejemplo, el codón de Metionina (Met) debe aparecer en el conjunto seis veces, y cada codón de Cisteína (Cys) debe aparecer tres veces. El conjunto de entrenamiento se extiende de 61 a 186 codones, lo que triplica el tiempo de entrenamiento en cada paso. Una estrategia aún más poderosa para obtener un error de clasificación bajo es tener un conjunto de entrenamiento adaptable, donde los ejemplos de entrenamiento son incluidos o excluidos después de determinar si son clasificados correctamente por la red.

4.1. Alcances de la solución. Consideraciones de implementación

Siguiendo el acercamiento presentado por Tolstrup *et al.* [13] se comprobó que el algoritmo de entrenamiento convencional de Backpropagation, que es el que se incluye en Weka, no resulta efectivo para obtener una predicción perfecta del código genético en una red con dos capas. Es por ello que se modificó el algoritmo en Java, según las soluciones propuestas en [1, 13] y se evaluaron los resultados (ver sección siguiente). El trabajo de Tolstrup [13] fue extendido para incluirlo en el libro de Baldi y Brunak [1], aunque éste último no describe las características hidrofóbicas de los aminoácidos. En los trabajos mencionados, si bien se proponen soluciones para alcanzar mejores predicciones, no se incluye ninguna idea de implementación.

En principio, nuestra red fue entrenada en Weka con un conjunto de datos regular de 186 codones contenido en un archivo `arff`. Se observó que con learning rates bajos mejoraba la cantidad de predicciones correctas a largo plazo, luego se fijó $\eta = 0{,}025$ y $momentum = 0{,}1$. El conjunto de validación consistió de los 61 codones originales del cuadro 1. El alcance de la solución con el algoritmo de Backpropagation tradicional resultó satisfactorio, con un **69 % de aciertos**. Sin embargo, para ampliar la contribución se pretendía también evaluar el impacto en la práctica de las distintas modificaciones antes mencionadas para obtener mejores predicciones.

La primer modificación realizada consistió en utilizar una tasa de aprendizaje η alta para ejemplos incorrectamente clasificados y una tasa baja para los ejemplos correctamente clasificados. Se modificó en consecuencia el procedimiento `buildClassifier(Instances i)` del archivo `MultilayerPerceptron.java`, básicamente los pasos son:

Algoritmo:

Para cada iteración:
$i \leftarrow$ instancia a clasificar
Si la red clasifica correctamente a i
entonces utilizar un valor η en el entrenamiento
sino utilizar un valor $3 * \eta$ para reforzar el aprendizaje.

Con este mecanismo se logró entrenar la red hasta lograr un **87 % de los ejemplos correctamente clasificados**. Se observó que los aminoácidos con menos frecuencia de presentación eran los que resultaban incorrectamente clasificados, como por ejemplo el codón Met o el Trp (1 aparición cada uno dentro de las 61 instancias), el His o el Phe (2 apariciones cada uno).

Luego de estas consideraciones, sería lógico pensar que un algoritmo que automáticamente adapta su conjunto de entrenamiento de acuerdo a las falencias de la clasificación tendría mejores resultados. Una segunda alternativa sería tener un conjunto de entrenamiento *adaptable*, donde los ejemplos de entrenamiento son incluidos o excluidos después de determinar si son clasificados correctamente por la red. En cada paso, un ejemplo de entrenamiento se escoge al azar de un *pool*. Para incrementar la frecuencia de un ejemplo difícil de aprender, cada ejemplo mal clasificado se agrega al pool de ejemplos, reemplazando uno existente. Para asegurar que ningún ejemplo se pierda, sólo parte del pool está abierto para intercambio. En nuestra implementación se parte con 3 conjuntos de entrenamiento idénticos (cada uno conteniendo los 61 codones originales), aunque sólo dos de ellos están abiertos a intercambio.

Datos de entrada: $Intercambio_1$, $Intercambio_2$, $Instancias$ (conjuntos idénticos)
Algoritmo:

Para cada iteración:
$i \leftarrow$ escoger una instancia de $(Intercambio_1 \cup Intercambio_2 \cup Instancias)$
Si la red clasifica correctamente a i
entonces utilizar un valor η en el entrenamiento
sino $j \leftarrow$ escoger una instancia de $(Intercambio_1 \cup Intercambio_2)$
reemplazar a la instancia j por la instancia i
utilizar un valor $3 * \eta$ para reforzar el aprendizaje de i.

Naturalmente, la performance de esta implementación depende mucho de la calidad del generador de números aleatorios, y de allí la justificación de que distintas semillas den resultados muy variables. En la implementación se utilizó la clase `Random` del paquete `java.util`. Esta librería genera números pseudoaleatorios por el denominado método congruencial lineal (ver [7], sección 3.2.1). Para evitar los períodos de repetición de los números, en cada ciclo o iteración del algoritmo de Backpropagation se cambia la semilla. La elección de la semilla inicial condiciona los resultados del algoritmo de entrenamiento, i. e., con la misma cantidad de iteraciones se logra distinta cantidad de ejemplos bien clasificados. Cabe señalar que en los experimentos realizados se logró entrenar de manera perfecta (**100 % de aciertos**) a una red de dos capas con 15200 pasos de entrenamiento. La performance de dicha red a partir de los pesos asignados a sus conexiones serán analizados en la sección siguientes.

4.2. Análisis de Resultados

Una red con dos unidades ocultas fue entrenada exitosamente utilizando el esquema de entrenamiento adaptable. Durante el entrenamiento, la red formó una representación interna de la correspondencia del código genético. La representación interna de la estructura del código está dada por las vinculaciones entre las dos unidades intermedias, las que pueden ser fácilmente visualizadas en dos dimensiones. En los gráficos de la Figura 4, cada codón se corresponde con un punto (x, y) en el plano con la leyenda asociada al aminoácido. Tanto x como y son valores entre 0 y 1, y corresponden a la salida de cada una de las unidades intermedias de la red. Durante el entrenamiento, los 61 puntos siguieron trayectorias que emergen desde posiciones iniciales cercanas al centro de la gráfica, y finalizando mayormente en posiciones cercanas a los bordes. Después de 5 iteraciones del algoritmo, en la red prácticamente sin entrenar, los 61 puntos se ubican cerca del centro del cuadrado, dando sólo 4 instancias bien clasificadas. Luego de 500 pasos de entrenamiento, los puntos comienzan a dispersarse en regiones y se obtienen 22 instancias correctamente clasificadas. En el tercer gráfico se cuadruplicó el tiempo de entrenamiento: en 2000 pasos los puntos sufren un reacomodamiento significativo, y se tienen 43 ejemplos correctamente clasificados. Finalmente, en 15200 pasos los puntos están ubicados en sus posiciones finales, en grupos bastante separados, y la cantidad de predicciones correctas es igual a 61. Las instancias que permanecen agrupadas cerca del centro corresponden a la Arginina, que se conoce como una excepción en el contexto del código genético [12]. En efecto, el número de codones de Arginina está en conflicto con su abundancia en las proteínas, y fue sugerida como un agregado posterior al código genético. A partir de los resultados precedentes se podrían delimitar "regiones"que contengan a los codones de cada aminoácido, corroborando que las clases han sido separadas exitosamente.

Luego de 15200 pasos de entrenamiento, la red clasifica correctamente la totalidad de los ejemplos, con un error promedio absoluto de 0,0698. El hecho de que la red necesite de al menos dos unidades intermedias para aprender el código genético es consecuencia de que el código es inherentemente no lineal. Cabe señalar que el tamaño de los pesos de las conexiones desde las unidades de entrada hacia las intermedias refleja la importancia de cada nucleótido en su posición de codón. En la Figura 5 se ve la suma de los dos pesos que conectan una unidad de entrada con cada unidad intermedia, para cada una de las tres posiciones de codón. En el gráfico de la izquierda se observa que inicialmente (tras 5 iteraciones) la magnitud de los pesos es similar, y siempre negativa. Cuando la red ya está entrenada, los pesos difieren bastante entre los nucleótidos de una misma posición y de las posiciones entre sí. Es interesante notar también que la segunda posición de codón es la que posee los pesos más grandes, seguida por la primera y tercera posición, en concordancia con observaciones similares realizadas en la literatura [1, 9, 13].

5. Conclusiones

En este trabajo se ha explorado un primer acercamiento al aprendizaje del código genético a través de redes neuronales. Como se mencionó al comienzo, el aprendizaje del código genético es un problema central en la Bioinformática, y su resolución involucra necesariamente la aplicación de modelos computacionales suficientemente poderosos y expresivos como para atacarlo. En tal sentido, hemos visto que las redes neuronales constituyen clasificadores robustos para aproximar funciones altamente complejas. Este trabajo estuvo motivado por comenzar a investigar algunos

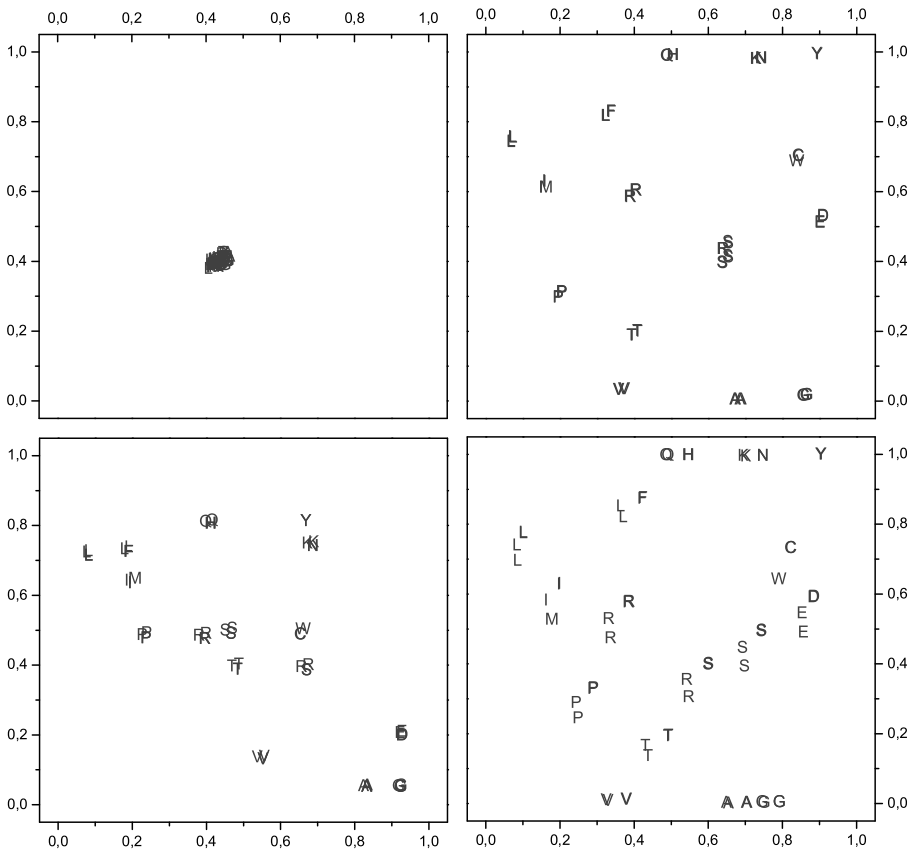

Figura 4: Evolución en el aprendizaje de la red de dos capas.

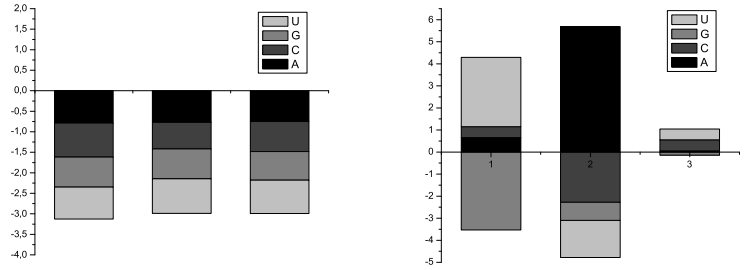

Figura 5: Representación gráfica de los pesos de las unidades de entrada.

de los aspectos sobre cómo aprender código genético a través de redes neuronales, utilizando los diversos elementos de evaluación y de análisis que brinda la plataforma de software WEKA.

En nuestro caso el problema abordado fue relativamente sencillo, dado que la red neuronal tenía que aprender a reconocer los distintos aminoácidos a partir de una secuencia de nucleótidos. El alcance de la solución con el algoritmo de Backpropagation tradicional resultó satisfactorio, con un 69 % de aciertos. No obstante, creemos que es interesante el hecho de que se obtuvo una red mínima perfecta (i. e., 100 % de predicciones correctas) a partir de algoritmos ligeramente modificados para la tarea, siguiendo la línea de los desarrollados en [13]. Para verificar la correctitud, los resultados obtenidos fueron contrastados exitosamente con los de este último artículo, obteniéndose un alto grado de similitud y correlación. Parte del trabajo futuro a desarrollar involucra analizar distintas hipótesis aún no resueltas sobre correspondencias entre

codones y aminoácidos en el ámbito de la bioinformática [1] a partir del uso de redes neuronales como modelo computacional subyacente.

Agradecimientos

Agradecemos a los revisores por los comentarios que nos fueron aportados y las sugerencias para mejorar este trabajo.

Referencias

[1] Pierre Baldi and Søren Brunak. *Bioinformatics: The Machine Learning Approach*, chapter 5, 6. The MIT Press, Cambridge, Massachusetts, 2nd. edition, 2001.

[2] Carlos Chesñevar. Notas de Clase. *Técnicas de Aprendizaje Automatizado y Datamining: Fundamentos y Aplicaciones*, 2004. Universidad Nacional del Sur.

[3] P. S. Churchland and T. J. Sejnowski. *The computational brain.* The MIT Press, Cambridge, MA, 1992.

[4] Real Academia Española. Diccionario de la Real Academia Española. http://www.rae.es.

[5] E. Frank, M. Hall, L. Trigg, G. Holmes, and I. H. Witten. Data mining in bioinformatics using Weka. *Bioinformatics*, 20(15):2479–2481, October 2004.

[6] M. Gabriel and J. Moore. *Learning and computational neuroscience: Foundations of adaptive networks (edited collection).* The MIT Press, Cambridge, MA, 1990.

[7] Donald E. Knuth. *The Art of Computer Programming*, volume Volume 2: Seminumerical Algorithms. Addison-Wesley Professional, 3rd edition, 1997.

[8] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[9] M. D. Perlwitz, C. Burks, , and M. S. Waterman. Pattern analysis of the genetic code. *Advances in Applied Mathematics*, 9:7–21, 1988.

[10] N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[11] Sun Microsystems, Inc. Java 2 Platform Std. Ed. v1.4.2 - API Reference, 2003.

[12] F. J. R. Taylor and D. Coates. The code within the codons. *Biosystems*, 22:177–187, 1989.

[13] N. Tolstrup, J. Toftgård, J. Engelbrecht, and S. Brunak. Neural Network Model of the Genetic Code is Strongly Correlated to the GES Scale of Amino Acid Transfer Free Energies. *Journal of Molecular Biology*, 243(5):816–820, Nov. 1994.

[14] Weka webpage. http://www.cs.waikato.ac.nz/~ml/weka/.

[15] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, chapter 8 "Nuts and bolts: Machine learning algorithms in Java". Morgan Kaufmann, 2000.

[16] S. F. Zornetzer, J. L. Davis, and C. Lau. *An introduction to neural and electronic networks (edited collection).* Academic Press, New York, 2nd. edition, 1994.