Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Depto. de Ciencias de la Computación
CC5114-1 Redes Neuronales y Programación Genética

Tarea 3

Aprendizaje No Supervisado en Topología Fija

Alumno : Belisario Panay S.
R.U.T. : 18.026.101-k
Profesor : Alexandre Bergel

Introducción

En esta tarea se implemento un algoritmo genético sobre una red neuronal con topología fija, en el cual se buscaban los pesos y bias de las neuronas de la red, para así poder clasificar el dataset de iris. Para lograr esto se implemento un método de selección por torneo, en el que se escogía un subconjunto de individuos y se elegía al mejor de este subconjunto para ser reproducido, a continuación se presentan los detalles de la implementación y sus resultados.

Descripción del Dataset

Para esta tarea se utilizó el Iris data set uno de los dataset más conocidos en el reconocimiento de patrones, este dataset contiene datos de 3 clases de plantas Iris, se tienen 50 instancias cada una (150 filas en total). Cada instancia posee 4 atributos que permiten identificar el tipo de la planta, estos son la longitud de su sépalo en centímetros, la anchura de su sépalo, la longitud de su pétalo y la anchura de su pétalo. Cabe recalcar que no existe ni un dato duplicado en este dataset.

Estructura del Proyecto

Todo el código fuente de esta tarea se encuentra en la carpeta **src**, los datasets (el de iris y el de prueba) se encuentran en la carpeta **Datasets** y el unittest en la carpeta **Tests**. Para hacer correr el código se debe ejecutar el archivo **main.py** con **Python 3.6**, main ejecutará una red definida sobre el data set **iris.data** y entregará 2 gráficos, uno del índice de aciertos de la red y otro de la cantidad de errores en función de los epochs transcurridos, la cantidad de epochs que son realizados para el entrenamiento pueden ser elegidos al cambiar la variable $number\_of\_epochs$, se puede notar que existen otras tres líneas comentadas en la función main, las cuales permitieron hacer algunos análisis sobre la red, pero fueron comentados por el tiempo que se demoran en ejecutar.
Este código solo soporta el dataset de las plantas Iris, si se desea procesar otro dataset se debe agregar un método a la clase Parser y agregar la cantidad de clases de output al momento de inicializar el objeto *NeuralNetwork*.

Algoritmo Genético

En la implementación del algoritmo genético se utilizó una topología fija, por lo que para no crear una red por cada individuo, el algoritmo poseía una única red neuronal en la cual se cargaban los valores de los individuos, así cada individuo era un arreglo de valores que luego eran ingresados a la red neuronal para luego ser evaluados. Para la selección de los individuos que producirían la siguiente generación se utilizó el método de selección de torneo, en el cual se elegía al mejor individuo de un subconjunto aleatorio.

Configuración de la Red Neuronal

La red fija del algoritmo se creó con 3 capas, una de input, otra capa escondida y una última de salida. La capa de entrada poseía 4 neuronas (por cada variable del dataset), en la intermedia se tenían otras 4 neuronas y 3 en la última (debido a que existen tres clases en el dataset). La red se puede observar en la siguiente figura.

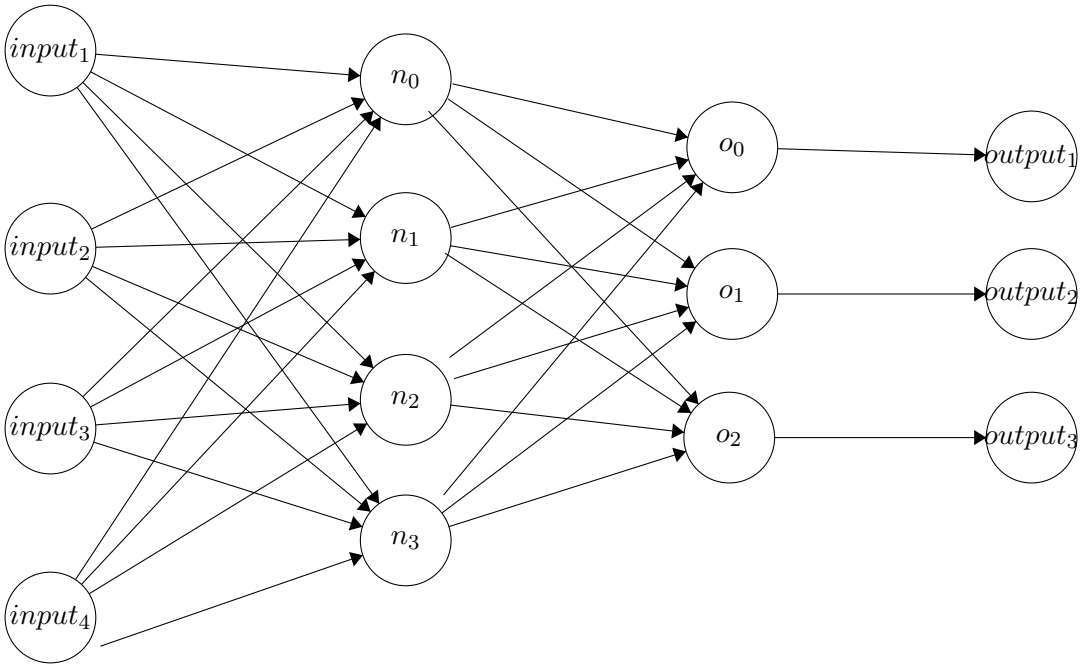

Figura 1: Estructura red neuronal

Individuos

Cada uno de los individuos es una red neuronal pero estos son representados como un arreglo de valores, donde cada valor puede ser un peso o un bias de la red, para la red fija utilizada se tenían 11 neuronas en las cuales se tenia un total de 32 pesos y 11 bias. Los valores de pesos y bias para la red son elegidos al azar uniformemente en un rango entre $-15{,}0$ y $15{,}0$.

Estructura algoritmo

- `initializa_population()`: Se crea una población con sujetos random.
- `evaluate_fitness()`: Se retorna el valor de aciertos al probar el dataset en la red
- `selection()`: Se elige un subconjunto de individuos y se retorna el con mayor fitness de este (tournament selection).
- `cross_over()`: Se crean dos hijos, mezclando dos individuos
- `mutate()`: A un individuo se le cambian sus neuronas (al azar) en función de una proporción (mutation ratio).

Resultados

Para poder evaluar el *fitness* en el algoritmo se calcularon los aciertos de los individuos con un subset de los datos (20 % de lo datos, tomados de manera aleatoria), se probaron distintas configuraciones para poder encontrar una red con la mayor precisión posible. Se tiene una población de 100 individuos por generación, los resultados fueron los siguientes.

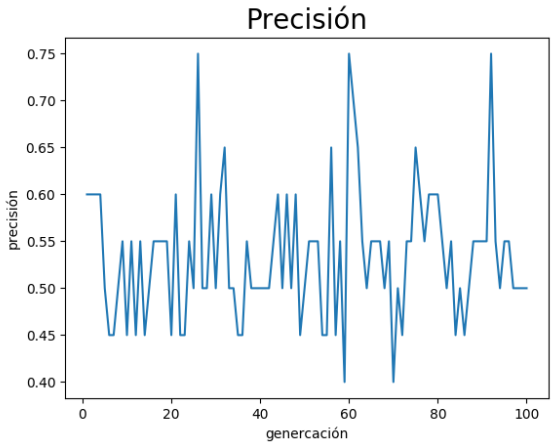

Figura 2: Precisión por generación a través de una selección de torneo con un 20 % de los individuos (truncada en 100 generaciones).

Del gráfico anterior se puede observar que la red aprende y olvida fácilmente, moviéndose al rededor de un 55 % de precisión, alcanzando máximos de 75 % de precisión. Esto se puede deber a dos métodos de la implementación, el primero es la selección de torneo la cual solo escoge el mejor individuo de un subconjunto de la población por lo que el individuo con mejor *fitness* puede quedar fuera de este subconjunto, empeorando la nueva generación. Otro factor que puede haber influido es el descarte de los padres a la hora de crear la siguiente generación, a que una vez que los padres se reproducen estos son descartados, por lo que se cambio la implementación, comenzando por tomar un subconjunto de mayor tamaño, el resultado fue el siguiente.

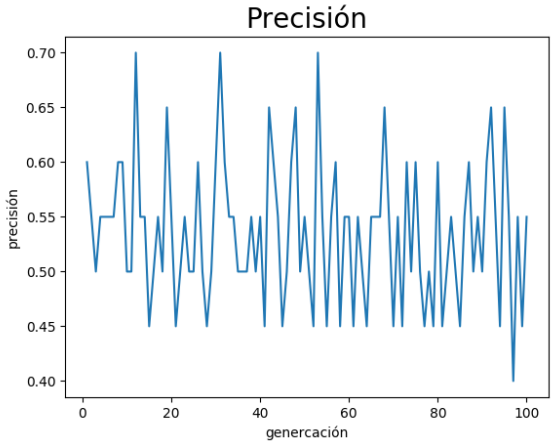

Figura 3: Precisión por generación a través de una selección de torneo con un 100 % de los individuos (truncada en 100 generaciones).

Se puede observar que los resultados no cambian en gran medida, por lo que se intentara cambiar el otro factor que se cree es el causante de la gran fluctuación de la precisión de los individuos, haciendo que al momento de reproducir a dos individuos, si los nuevos individuos tienen un *fitness* inferior a sus padres, los padres tomaran el lugar de sus hijos.

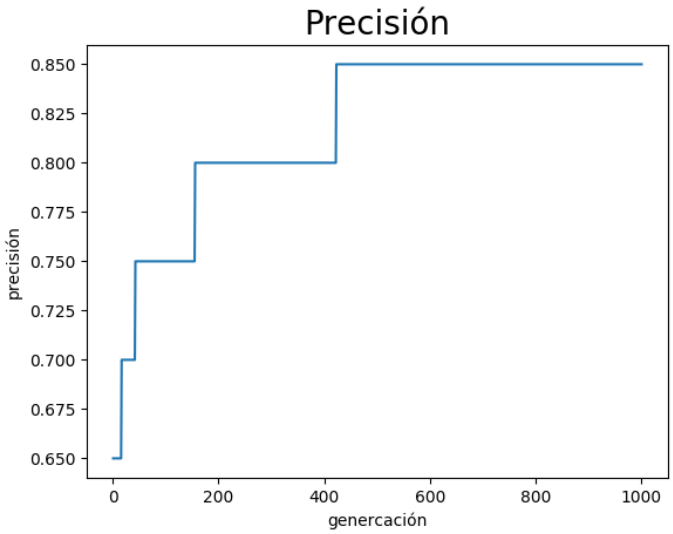

Figura 4: Precisión tomando en cuenta a los padres para la siguientes generaciones.

En este gráfico se puede observar datos mucho mejores y más consistentes, esto demuestra que es mejor tomar en cuenta a los padres para las siguientes generaciones y no reemplazar a la población en su totalidad (*generational replacement*).

Tiempo Procesamiento

Para la tarea 1 se obtuvo que la red no demora un tiempo considerable en realizar 1000 *epochs*. Se pueden observar los resultados a continuación.

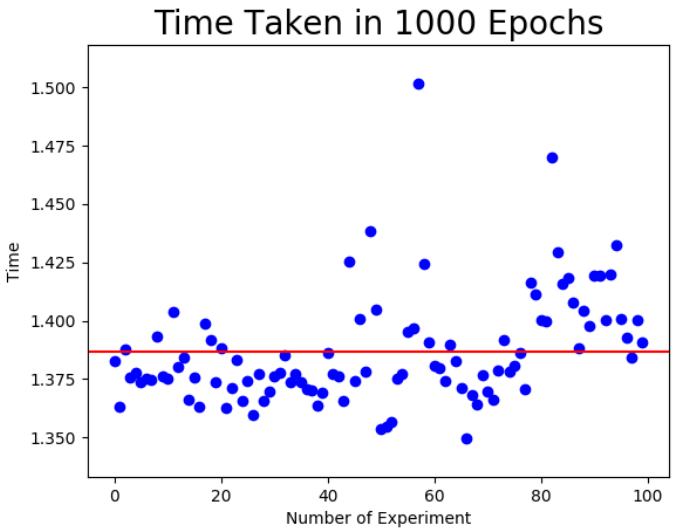

Figura 5: Tiempo promedio clasificación del 1000 epochs con backpropagation, en segundos.

En cambio para poder obtener porcentajes similares de aprendizaje en una red creada con el algoritmo genetico se necesitan al meno 1000 generaciones, para alcanzar un acierto comparable al que se tiene con bakcpropagation, los tiempos de ejecución de estas 1000 generaciones, se muestra a continuación

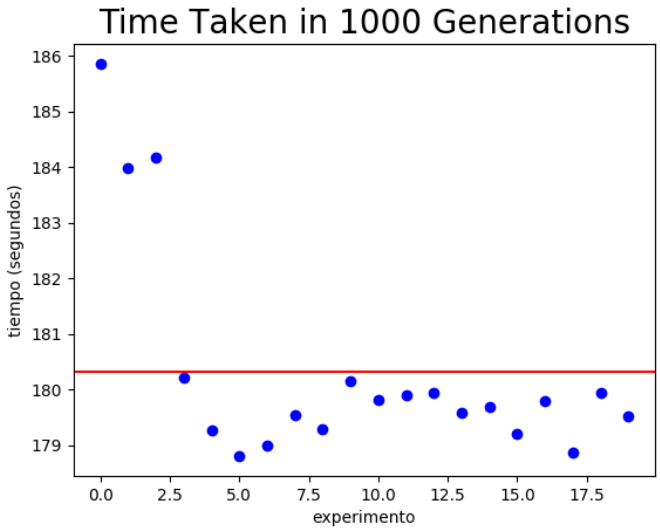

Figura 6: Tiempo promedio ejecución de 1000 generaciones, con población de 100 individuos.

El tiempo que se demora el algoritmo en avanzar 1000 generaciones toma un tiempo mucho mayor que el backpropagation toma en ralizar 1000 epochs.

Repositorio

Todo el código fuente, *tests* y datasets pueden ser encontrados en GitHub.