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

Tarea 1

Clasificación Iris Data Set

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

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 codigo 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*.

Configuración de la Red Neuronal

Para poder clasificar el dataset, se comenzó por crear una red fija, con 2 capas. En la primera se crearon 4 neuronas y 3 en la segunda (debido a que existen tres clases en el dataset). La red se puede observar en la siguiente figura.

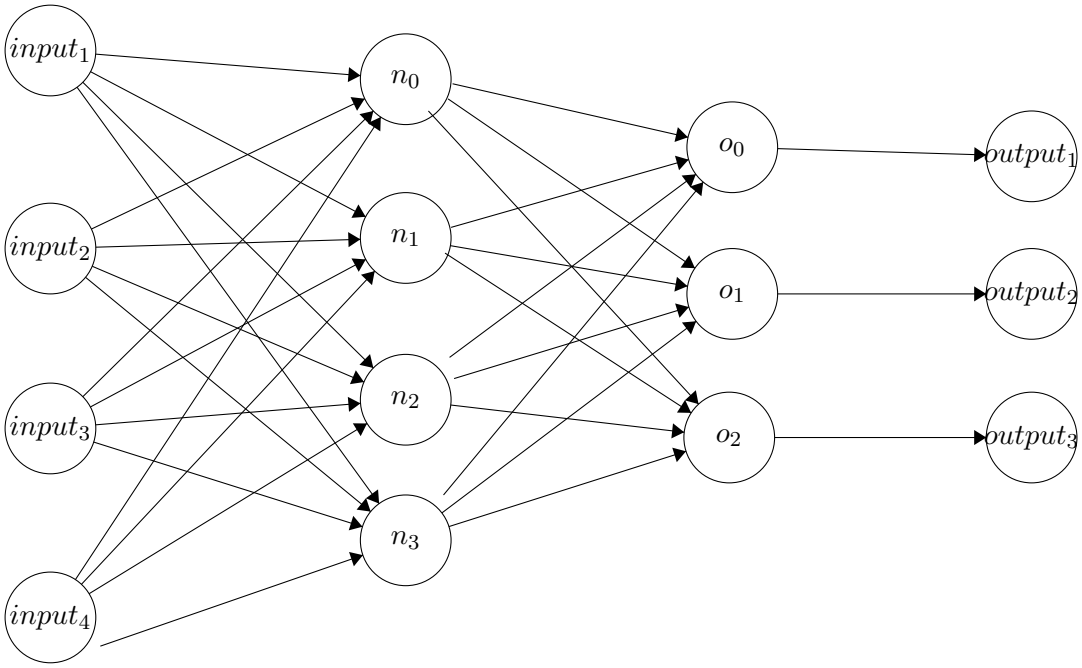

Figura 1: Estructura red neuronal

Los valores de pesos y bias para la red son elegidos al azar uniformemente, para el caso de los pesos, estos son elegidos en un rango que va desde -1 hasta 2 y para los bias se tiene un rango desde 1 a 3. Para los *learning rate* de la red son fijados en un valor de 0,05, los gráficos son demasiados para mostrarlos en un informe corto, pero se observó durante las pruebas que los mejores resultados se obtenían con esta taza de aprendizaje.

Resultados Clasificación Red Neuronal

Para poder cuantificar el progreso de la red se utilizaron dos métricas, la primera fue la precisión de la red, para esto se necesitaban datos de prueba por lo que el dataset original se separó en dos partes, una para poder entrenar a la red y otra para probar los resultados de la red sin reajustar los valores de esta. Con este dataset de prueba se pudo obtener el porcentaje de precisión de la red al clasificar (aciertos/total) en cada uno de los *epochs*. El mejor resultado se muestra en la siguiente gráfica.

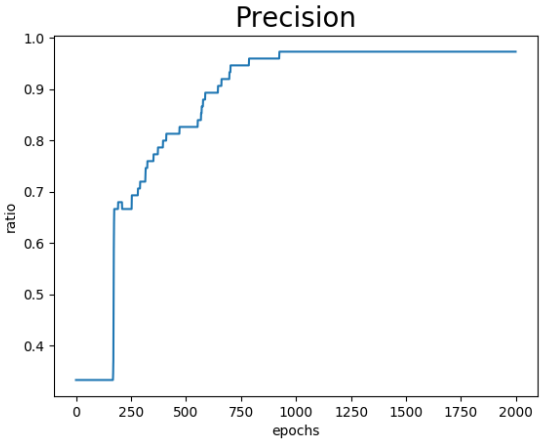

Figura 2: Precisión en cada epoch

Del gráfico anterior se puede observar que cerca de los 250 *epochs* se tiene un gran salto en la precisión del clasificador; desde un 33,3 % hasta 66,6 %, y luego la red aprende de manera escalonada hasta alcanzar un valor fijo de 97,3 % en precisión. La otra métrica utilizada fue el error cuadrático medio el cual fue calculado en cada *epoch*, el resultado obtenido para esta métrica fue el siguiente.

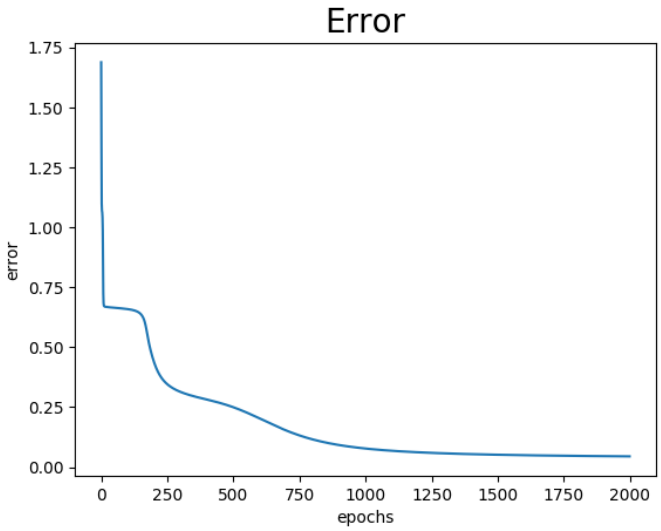

Figura 3: Error cuadrático medio por epoch

Acá se puede observar el brusco descenso en el error al mismo momento en que se tiene un gran diferencia en la precisión (250 *epochs*), finalizando en un valor cercano a cero.

Variación Capas Internas

Se creó una función auxiliar en el archivo *main* para poder comparar el aprendizaje de la red en función de la cantidad de capas que poseía esta. Se decidió mantener constante la cantidad de neuronas por capa e ir aumentando la cantidad de capas. Los resultados se pueden ver a continuación.

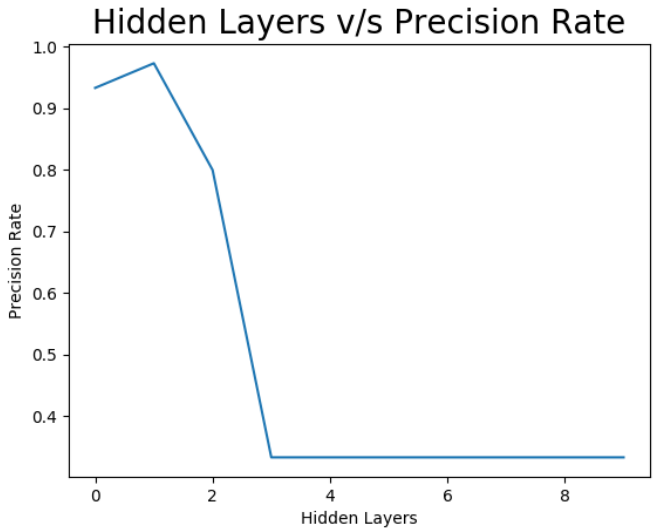

Figura 4: Precisión en función al número de capas

Se observa que al aumentar el número de capas de la red produce un deterioro en sus niveles de precisión, esto puede deberse a que al tener la misma cantidad de neuronas por capa hace que se produzca un sobre ajuste de los valores provocando que la red olvide lo que había aprendido hasta ese momento.

Tiempo Procesamiento

La red no demora un tiempo considerable en realizar 1 *epoch* por lo que al calcular el tiempo de ejecución se podría obtener un porcentaje de error muy grande en el tiempo calculado, por lo que se optó por medir el tiempo de ejecución de 1000 *epochs*. Se pueden observar los resultados a continuación.

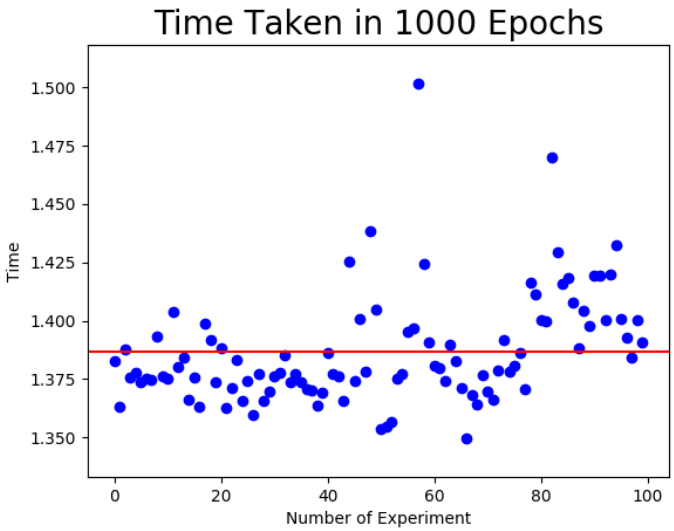

Figura 5: Tiempo promedio clasificación del 1000 epochs

El tiempo que se toma la red para clasificar 1000 *epochs* no supera los 2 segundos en ningún caso, esto era esperado por la poca cantidad de líneas que posee el dataset y la poca complejidad de la red.

Diferentes Tasas de Aprendizaje

ara comparar las diferentes tasas de aprendizaje se utilizó un rango que iba desde los 0,05 hasta los 2,0 con 18 valores intermedios (20 en total), los resultados en la precisión fueron los siguientes.

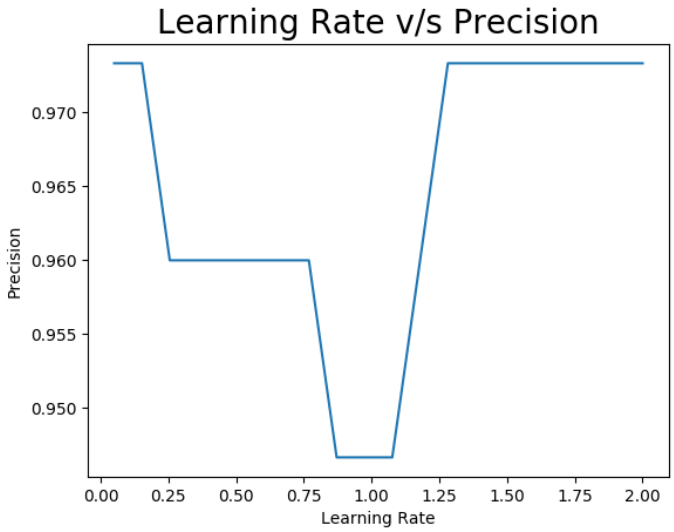

Figura 6: Precisión de distintos factores de aprendizaje

Se puede ver que cerca del valor de aprendizaje de 1,0 existe un mínimo local y que el valor escogido para las pruebas de precisión iniciales de 0,05 corresponden a un máximo local, pero también existen máximos locales con valores superiores a 1,25.

Orden de Datos

Para saber si es que el orden de los datos de entrenamiento afectan en algo al aprendizaje de la red, se comparan sus precisiones cuando los datos vienen directamente desde el dataset y otro en donde los datos fueron desordenados con la ayuda de la función *shuffle* de la biblioteca *random* de Python.

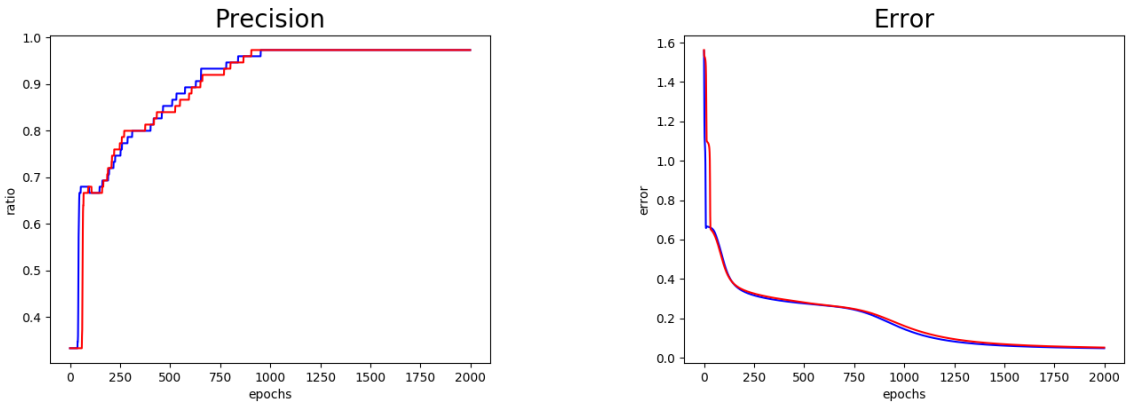

Figura 7: Precisión y error con datos ordenados y desordenados

En el gráfico la línea azul representa a los datos sin modificar y la roja con los datos desordenados, se puede observar claramente que el ordenamiento de los datos no afecta de ninguna manera a la red.

Repositorio

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