CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

APRENDIZAJE INTELIGENTE

6° "A"

PRÁCTICA 3: RED NEURONAL CON BACKTRACKING

Profesor: Francisco Javier Luna Rosas

Alumnos:

Espinoza Sánchez Joel Alejandro

Gómez Garza Dariana

González Arenas Fernando Francisco

Fecha de Entrega: Aguascalientes, Ags.**, 13** de febrero de 2021

Práctica 3: Red Neuronal con Backtracking

Objetivo:

Mediante el desarrollo de esta práctica, implementar la propagación hacia atrás de una red neuronal.

Introducción:

Las redes neuronales artificiales (también conocidas como sistemas conexionistas) son un modelo computacional el que fue evolucionando a partir de diversas aportaciones científicas que están registradas en la historia. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.

Cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Esta función se conoce como función de activación.

Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, y sobresalen en áreas donde la detección de soluciones o características es difícil de expresar con la programación convencional. Para realizar este aprendizaje automático, normalmente, se intenta minimizar una función de pérdida que evalúa la red en su total. Los valores de los pesos de las neuronas se van actualizando buscando reducir el valor de la función de pérdida. Este proceso se realiza mediante la propagación hacia atrás.

El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano, aunque las redes neuronales son más abstractas. Las redes

neuronales actuales suelen contener desde unos miles a unos pocos millones de unidades neuronales.

La propagación hacia atrás permite que la red neuronal ajuste los pesos por cada neurona y que se elaboró previamente en el lenguaje de programación R y ahora se tratará de implementar en Python

Pregunta de Investigación:

¿Cómo se puede implementar una red neuronal con propagación hacia atrás en Python?

Predicción:

Creemos que la implementación consistirá en buscar funciones equivalentes a las existentes en R para trabajar ahora en Python.

Materiales:

Una computadora con Python y el entorno Anaconda.

Método (Variables):

Dependiente: La predicción que se realizará.
Independiente: Los datos a tratar con el conjunto de datos de flores.
Controlada: El algoritmo de red neuronal con backtracking a implementar.

Seguridad:

Realmente no se trabajó en campo, por lo que no se corren riesgos con la práctica.

Procedimiento:

1.- Haciendo uso del lenguaje Python, se cargó al programa un conjunto de datos de flores:

```
#%% Extraer los datos del CSV
datos = pandas.read_csv('iris.csv', sep = ';', decimal = '.')
print("Datos:")
print(datos)
```

```
s.largo;s.ancho;p.largo;p.ancho;tipo
5.1;3.5;1.4;0.2;setosa
4.9;3.0;1.4;0.2;setosa
4.7;3.2;1.3;0.2;setosa
4.6;3.1;1.5;0.2;setosa
5.0;3.6;1.4;0.2;setosa
5.4;3.9;1.7;0.4;setosa
4.6;3.4;1.4;0.3;setosa
5.0;3.4;1.5;0.2;setosa
4.4;2.9;1.4;0.2;setosa
4.9;3.1;1.5;0.1;setosa
5.4;3.7;1.5;0.2;setosa
4.8;3.4;1.6;0.2;setosa
4.8;3.0;1.4;0.1;setosa
4.3;3.0;1.1;0.1;setosa
5.8;4.0;1.2;0.2;setosa
5.7;4.4;1.5;0.4;setosa
5.4;3.9;1.3;0.4;setosa
5.1;3.5;1.4;0.3;setosa
5.7;3.8;1.7;0.3;setosa
5.1;3.8;1.5;0.3;setosa
5.4;3.4;1.7;0.2;setosa
5.1;3.7;1.5;0.4;setosa
4.6;3.6;1.0;0.2;setosa
5.1;3.3;1.7;0.5;setosa
4.8;3.4;1.9;0.2;setosa
5.0;3.0;1.6;0.2;setosa
5.0;3.4;1.6;0.4;setosa
5.2;3.5;1.5;0.2;setosa
5.2;3.4;1.4;0.2;setosa
4.7;3.2;1.6;0.2;setosa
4.8;3.1;1.6;0.2;setosa
5.4;3.4;1.5;0.4;setosa
5.2;4.1;1.5;0.1;setosa
5.5;4.2;1.4;0.2;setosa
4.9;3.1;1.5;0.1;setosa
5.0;3.2;1.2;0.2;setosa
5.5;3.5;1.3;0.2;setosa
4.9;3.1;1.5;0.1;setosa
4.4;3.0;1.3;0.2;setosa
5.1;3.4;1.5;0.2;setosa
5.0;3.5;1.3;0.3;setosa
4.5;2.3;1.3;0.3;setosa
4.4;3.2;1.3;0.2;setosa
5.0;3.5;1.6;0.6;setosa
5.1;3.8;1.9;0.4;setosa
4.8;3.0;1.4;0.3;setosa
5.1;3.8;1.6;0.2;setosa
4.6;3.2;1.4;0.2;setosa
5.3;3.7;1.5;0.2;setosa
5.0;3.3;1.4;0.2;setosa
```

2.- En el código, se transformaron estos datos en dos conjuntos, uno de aprendizaje y otro de pruebas:

```
#%% Se segmentan los datos en dos tablas
muestra = random.sample(range(150),50)
ttesting = datos.iloc[muestra]
print("Tabla de pruebas:")
print(ttesting)
taprendizaje = datos.drop(muestra)
print("Tabla de aprendizaje:")
print(taprendizaje)
```

3.- Se asignaron posteriormente los datos de entrada y el rango de las salidas:

```
#%% Se asignan las entradas y salidas
x = datos.iloc[:,:4]
print("Tabla de datos de x:")
print(x)
y = datos.iloc[:,4:5]
print("Tabla del rango de y:")
print(y)
```

4.- Después la población de entrada y el rango de salida de clasifica en el entrenamiento y la prueba en una proporción de 70% y 30%.

```
#%% Se separa la población de entrenamiento y de aprendizaje
x_train,x_test,y_train,y_test = sklearn.model_selection.train_test_split(x, y, train_size = 0.7, random_state = 0)
instancia_Knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors = 7)
instancia_Knn.fit(x_train,y_train.iloc[:,0].values)
sklearn.neighbors.KNeighborsClassifier(n_neighbors = 7)
```

5.- Luego se realiza la predicción:

```
#%% Comienza la etapa de predicción
regression = instancia_Knn.predict(x_test)
```

6.- Finalmente se hace el cálculo de la matriz de confusión:

```
#%% Se calcula la matriz de confusión
confussion = sklearn.metrics.confusion_matrix(y_test,regression)

def indexing(confussion,flores = None):
    precision = numpy.sum(confussion.diagonal())/numpy.sum(confussion)
    #error = 1 - precision
    precisionI = pandas.DataFrame(confussion.diagonal()/numpy.sum(confussion,axis=1)).T
    if flores != None:
        precisionI.colums = flores
        return [precision, precisionI, 1 - precision, confussion]
```

7.- Al haber terminado de realizar cálculos, se imprimen los valores técnicos de estos cálculos:

```
index = indexing(confussion,list(numpy.unique(y)))

print("Precisión Global")
print(index[0])
print("Precisión Individual de cada aspecto")
print(index[1])
print("Error Global")
print(index[2])
print("Matriz de Confusión")
print(index[3])
```

Obtención y Procesamiento de Datos:

Primero se imprime la tabla de datos completa para asegurarse a nivel de desarrollo que todo se haga correctamente:

Datos:

	s.largo	s.ancho	p.largo	p.ancho	tipo
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
22	4.6	3.6	1.0	0.2	setosa
23	5.1	3.3	1.7	0.5	setosa
24	4.8	3.4	1.9	0.2	setosa
25	5.0	3.0	1.6	0.2	setosa
26	5.0	3.4	1.6	0.4	setosa
27	5.2	3.5	1.5	0.2	setosa
28	5.2	3.4	1.4	0.2	setosa
29	4.7	3.2	1.6	0.2	setosa
..	...	...	...	...	...
120	6.9	3.2	5.7	2.3	virginica
121	5.6	2.8	4.9	2.0	virginica

Después se imprime la tabla de pruebas seleccionada al azar y la tabla de aprendizaje que es el complemento de la anterior:

Tabla de pruebas:

	s.largo	s.ancho	p.largo	p.ancho	tipo
2	4.7	3.2	1.3	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
78	6.0	2.9	4.5	1.5	versicolor
138	6.0	3.0	4.8	1.8	virginica
63	6.1	2.9	4.7	1.4	versicolor
62	6.0	2.2	4.0	1.0	versicolor
25	5.0	3.0	1.6	0.2	setosa
26	5.0	3.4	1.6	0.4	setosa
109	7.2	3.6	6.1	2.5	virginica
12	4.8	3.0	1.4	0.1	setosa
126	6.2	2.8	4.8	1.8	virginica
29	4.7	3.2	1.6	0.2	setosa
82	5.8	2.7	3.9	1.2	versicolor
35	5.0	3.2	1.2	0.2	setosa
68	6.2	2.2	4.5	1.5	versicolor
15	5.7	4.4	1.5	0.4	setosa
79	5.7	2.6	3.5	1.0	versicolor
142	5.8	2.7	5.1	1.9	virginica
69	5.6	2.5	3.9	1.1	versicolor
141	6.9	3.1	5.1	2.3	virginica
103	6.3	2.9	5.6	1.8	virginica
22	4.6	3.6	1.0	0.2	setosa
19	5.1	3.8	1.5	0.3	setosa
73	6.1	2.8	4.7	1.2	versicolor
49	5.0	3.3	1.4	0.2	setosa
36	5.5	3.5	1.3	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
83	6.0	2.7	5.1	1.6	versicolor
122	7.7	2.8	6.7	2.0	virginica
133	6.3	2.8	5.1	1.5	virginica
60	5.0	2.0	3.5	1.0	versicolor
117	7.7	3.8	6.7	2.2	virginica
91	6.1	3.0	4.6	1.4	versicolor

Tabla de aprendizaje:

	s.largo	s.ancho	p.largo	p.ancho	tipo
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
13	4.3	3.0	1.1	0.1	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
23	5.1	3.3	1.7	0.5	setosa
24	4.8	3.4	1.9	0.2	setosa
28	5.2	3.4	1.4	0.2	setosa
30	4.8	3.1	1.6	0.2	setosa
31	5.4	3.4	1.5	0.4	setosa
32	5.2	4.1	1.5	0.1	setosa
33	5.5	4.2	1.4	0.2	setosa
39	5.1	3.4	1.5	0.2	setosa
40	5.0	3.5	1.3	0.3	setosa
41	4.5	2.3	1.3	0.3	setosa
42	4.4	3.2	1.3	0.2	setosa
43	5.0	3.5	1.6	0.6	setosa
44	5.1	3.8	1.9	0.4	setosa
45	4.8	3.0	1.4	0.3	setosa
46	5.1	3.8	1.6	0.2	setosa
..	...	...	...	...	...
108	6.7	2.5	5.8	1.8	virginica

También se imprimieron los datos de entrada de la red y el rango de salida:

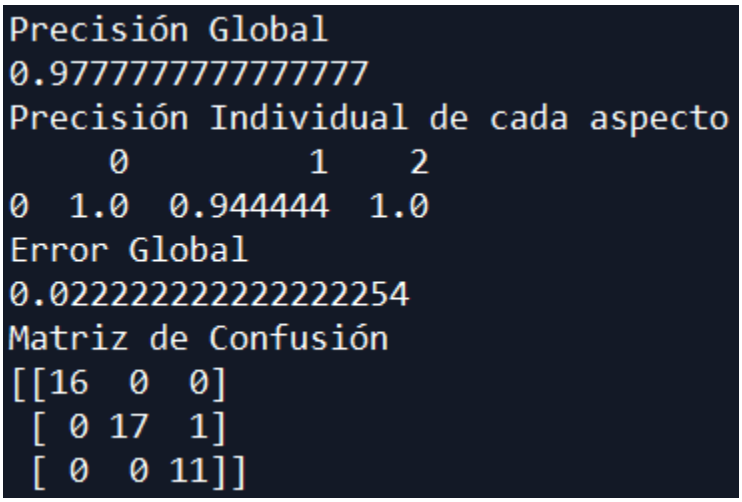
Tabla de datos de x:

	s.largo	s.ancho	p.largo	p.ancho
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
120	6.9	3.2	5.7	2.3
121	5.6	2.8	4.9	2.0
122	7.7	2.8	6.7	2.0

Tabla del rango de y:

	tipo
0	setosa
1	setosa
2	setosa
3	setosa
4	setosa
5	setosa
6	setosa
7	setosa
8	setosa
9	setosa
10	setosa
11	setosa
12	setosa
13	setosa
14	setosa
15	setosa
16	setosa
17	setosa
18	setosa
19	setosa
20	setosa
21	setosa
22	setosa
23	setosa
24	setosa
25	setosa
26	setosa
27	setosa
28	setosa
29	setosa
..	...
120	virginica

Finalmente se imprime la precisión global, la precisión individual, el error y la matriz generada de confusión:



Y puede observarse que la red neuronal consigue aprender para que el error se disminuya en una gran mayoría al punto de tender a 1.

Conclusiones:

Joel Alejandro Espinoza Sánchez: Las redes neuronales nos ayudan en la predicción de datos que es muy útil, sin embargo, como se busca la independencia de la máquina en algoritmos de este tipo, la aplicación del Backtracking es importante en este algoritmo.

En este trabajo, nos habían encargado adaptar un algoritmo de redes neuronales con backtracking de R a Python que es muy útil para observar cómo se pueden tratar los mismos problemas con diferentes herramientas, ya que parece ser que en R dicho algoritmo es más sencillo, sin embargo es perfectamente replicable en Python aunque sea ligeramente más complicado.

Dariana Gómez Garza: Esta práctica nos ayudó a conocer un poco más el código que el maestro nos enseñó en R pero aplicado en Python.

En la realización de la práctica nos dimos cuenta de muchas funciones que tiene R para ponerse "ahorrar" ciertos pasos que en otros lenguajes.

Al aplicar el NNBP en este ejercicio también nos ayudó a reforzar un poco más el tema y los temas pasados, ya que para mi parecer fue un complemento de lo ya visto de las redes neuronales, así como el funcionamiento del backtrancking en un

ejercicio más específico y gracias a eso podemos tener una visión más amplia de la aplicación de una red neuronal y que al mismo tiempo nos da sentido de porque es importante su aplicación.

Fernando Francisco González Arenas: La implementación de las redes neuronales con técnica de Backtracking es un método interesante para utilizar en el aprendizaje de estas, ya que va buscando la solución y vuelve atrás si así es conveniente, es una técnica muy útil porque además al usar recursividad es una técnica muy eficiente. es muy interesante todas las técnicas que se pueden usar en la implementación de los algoritmos de inteligencia artificial, más específicamente con las redes neuronales.

Referencias:

McCulloch, Warren; Walter Pitts. (1943). *A Logical Calculus of Ideas Immanent in Nervous Activity.* Inglaterra: Bulletin of Mathematical Biophysics

Farley, B.G.; W.A. Clark. (1954). *Simulation of Self-Organizing Systems by Digital Computer.* Chicago: IRE Transactions on Information Theory.

Anexos:

Anexo 1: Código del programa en lenguaje Python:

```
#%% Presentación
'''
   Universidad Autónoma de Aguascalientes

         Centro de Ciencias Básicas
 Departamento de Ciencias de la Computación
          Aprendizaje Inteligente
                   6° "A"

                 Práctica 3

   Profesor: Francisco Javier Luna Rosas
  Alumnos:
      Espinoza Sánchez Joel Alejandro
      Gómez Garza Dariana
      González Arenas Fernando Francisco

  Fecha de Entrega: 15 de marzo del 2021

Descripción: Red neuroonal con Backtracking
'''


#%% Librerías
import pandas
import numpy
import random
import sklearn

#%% Extraer los datos del CSV
datos = pandas.read_csv('iris.csv', sep = ';', decimal = '.')
print("Datos:")
print(datos)

#%% Se segmentan los datos en dos tablas
muestra = random.sample(range(150),50)
ttesting = datos.iloc[muestra]
print("Tabla de pruebas:")
print(ttesting)
taprendizaje = datos.drop(muestra)
print("Tabla de aprendizaje:")
print(taprendizaje)

#%% Se asignan las entradas y salidas
x = datos.iloc[:,:4]
print("Tabla de datos de x:")
print(x)
y = datos.iloc[:,4:5]
```

```
print("Tabla del rango de y:")
print(y)

#%% Se separa la población de entrenamiento y de aprendizaje
x_train,x_test,y_train,y_test = sklearn.model_selection.train_test_split(x, y,
train_size = 0.7, random_state = 0)
instancia_Knn = sklearn.neighbors.KNeighborsClassifier(n_neighbors = 7)
instancia_Knn.fit(x_train,y_train.iloc[:,0].values)
sklearn.neighbors.KNeighborsClassifier(n_neighbors = 7)

#%% Comienza la etapa de predicción
regression = instancia_Knn.predict(x_test)

#%% Se calcula la matriz de confusión
confussion = sklearn.metrics.confusion_matrix(y_test,regression)

def indexing(confussion,flores = None):
    precision = numpy.sum(confussion.diagonal())/numpy.sum(confussion)
    #error = 1 - precision
    precisionI =
pandas.DataFrame(confussion.diagonal()/numpy.sum(confussion,axis=1)).T
    if flores != None:
        precisionI.colums = flores
        return [precision, precisionI, 1 - precision, confussion]

index = indexing(confussion,list(numpy.unique(y)))

print("Precisión Global")
print(index[0])
print("Precisión Individual de cada aspecto")
print(index[1])
print("Error Global")
print(index[2])
print("Matriz de Confusión")
print(index[3])
```