UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE
CIENCIA DE LA COMPUTACIÓN

INTELIGENCIA ARTIFICIAL

Redes Neuronales

Docente:
Cristian lopes

Integrantes:
Quispe Totocayo, Raul Edgar

8 de junio de 2019

Índice general

0.1. Redes Neuronales

0.1.1. BackPropagation

Sea la siguiente composición de funciones:

$$C(a(z^L))$$

donde C es la funcion conste definida como:

$$C(a_j^L) = \frac{1}{2}\sum_j (y_i - a_j^L)^2$$

a la funcion de activacion:

$$a^L(z^L) = \frac{1}{1+e^{-z^L}}$$

y z la suma ponderada:

$$z^L = \sum_i a_i^{L-1} w_i^L + b^L$$

Como varia el *coste* ante un cambio del parametro W?

el parametro W esta conformado por w y el b por lo cual tendremos que derivar con respecto a cada uno

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial w^L}$$
$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L} * \frac{\partial z^L}{\partial b^L}$$

Ahora resolvemos esas derivadas parciales: derivada del coste con respecto al la funcion de activacion:

$$\frac{\partial C}{\partial a^L} = (a_j^L - y_j)$$

derivada de la funcion de activacion con respecto a la suma ponderada:

$$\frac{\partial a^L}{\partial z^L} = a^L(z^L) * (1 - a^L(z^L))$$

derivada de la suma ponderada con respecto a w:

$$\frac{\partial z^L}{\partial w^L} = a_i^{L-1}$$

derivada de la suma ponderada con respecto a b:

$$\frac{\partial z^L}{\partial b^L} = 1$$

Algoritmo

1. Computo del error de la ultima capa

$$\delta^L = \frac{\partial C}{\partial a^L} * \frac{\partial a^L}{\partial z^L}$$

2. Retropropagamos el error a la capa anterior

$$\delta^{L-1} = W^L * \delta^L * \frac{\partial a^{L-1}}{\partial z^{L-1}}$$

3. calculamos las derivadas de la capa usando el error

$$\frac{\partial C}{\partial b^{L-1}} = \delta^{L-1}$$

$$\frac{\partial C}{\partial w^{L-1}} = \delta^{L-1} * a^{L-2}$$

0.1.2. Implementacion de la Red Neuronal

```
1
2  class NeuralLayer(object):#clase capa neuronal
3      def __init__(self,numberConections,numberNeurons,activationFunction):
4          self.numberConections=numberConections
5          self.numberNeurons=numberNeurons
6          self.activationFunction=activationFunction
7          self.bayas=np.random.rand(1,numberNeurons)*2-1#inicializacion con r
8          self.W=np.random.rand(numberConections,numberNeurons)*2-1#inicializ
9  class NeuralNetwork:
10     def __init__(self,learningRatio=0.01,train=True,numIterations=1000,topo
11         self.learningRatio=learningRatio
12         self.train=train
13         self.numIterations=numIterations
14         self.topology=topology
15         self.neuralNetwork=self.createNeuralNetwork()
16     def createNeuralNetwork(self):
17         nn=[]
18         for i,layer in enumerate(self.topology[:-1]):#itera hasta len(topol
19             nn.append(NeuralLayer(self.topology[i],self.topology[i+1],self.
20         return nn
21     sigmoide=(lambda x:1/(1+np.e**(-x)),lambda x:x*(1-x)) #funcion de activ
22     costFunction=(lambda yp,yr:np.mean((yp-yr)**2),
23                   lambda yp,yr:(yp-yr))#funcion de costo mas su rerivada
24     def forwardPropagation(self,X,Y):
25         out=[(None,X)]#tupla None,X
26         for i,layer in enumerate(self.neuralNetwork):
27             z=out[-1][1]@self.neuralNetwork[i].W+self.neuralNetwork[i].baya
28             a=self.neuralNetwork[i].activationFunction[0](z)
29             out.append((z,a))#se agrega una nueva tupla confotmado de (z,a)
30                              #y a es resultado de pasar z como parametro po
31         return out
32     def backPropagation(self,X,Y):
33         out=self.forwardPropagation(X,Y)
34         if self.train:
35             deltas=[]
36             for i in reversed(range(0, len(self.neuralNetwork))):
37                 a=out[i+1][1]
```

```
38                  z=out[i+1][0]
39                  if i==len(self.neuralNetwork)-1:#para la ultima capa
40                      deltas.insert(0,self.costFunction[1](a,Y)*self.neuralNetwo
41                  else:#para las demas capas
42                      deltas.insert(0, deltas[0] @ _W.T * self.neuralNetwork[i].
43                  _W=self.neuralNetwork[i].W
44                  ##desenso del gradiente
45                  self.neuralNetwork[i].bayas=self.neuralNetwork[i].bayas-np.mea
46                  self.neuralNetwork[i].W=self.neuralNetwork[i].W-out[i][1].T@d
47          return out[-1][1]
48      def fit(self,X,Y):
49          loss=[]
50          for i in range(self.numIterations):
51              out=self.backPropagation(X,Y)
52              loss.append(self.costFunction[0](out,Y))
53              clear_output(wait=True)
54          #plt.plot(range(len(loss)), loss)
55          #plt.show()
56          return loss
57      def predict(self,X,Y):
58          confusionMatrix=[[0,0,0],[0,0,0],[0,0,0]]
59          outPut=[]
60          for i in range(X.shape[0]):
61              out=self.forwardPropagation(X[i:i+1,:],Y[i])
62              #outPut.append(out[-1][1])
63              #outPut[i]=outPut[i].flatten()
64              #outPut[i]=np.asscalar(outPut[i])
65              #print("salida ","i=",i,out[-1][1],"salida deseada",Y[i])
66              if np.argmax(out[-1][1])== np.argmax(Y[i]):
67                  confusionMatrix[np.argmax(Y[i])][np.argmax(Y[i])]=confusionMat
68              elif np.argmax(out[-1][1])!=np.argmax(Y[i]):
69                  confusionMatrix[np.argmax(Y[i])][np.argmax(out[-1][1])]=confus
70          return confusionMatrix
71  def plotError(topology,lr,name):
72      losts=[]
73      for i in range(len(topology)):
74          nn=NeuralNetwork(learningRatio=lr,topology=topology[i],numIterations=1
75          losts.append(nn.fit(X,Y))
76      labels=['topologia[4,4,3]','topologia[4,6,3]','topologia[4,8,3]','topologi
77      for i in range(len(losts)):
78          plt.plot(range(len(losts[i])), losts[i], label=labels[i])
79      plt.xlabel('Iteraciones')
80      plt.ylabel('Error')
81      plt.title("learning ratio="+str(lr))
82      plt.legend()
83      #plt.show()
84      plt.savefig(name+".png")
85  def plotConfusinMatrix(topology,lr,name):
86      labels=['topologia[4,4,3]','topologia[4,6,3]','topologia[4,8,3]','topologi
87
88      for i in range(len(topology)):
```

```
89          ax = plt.axes()
90          nn=NeuralNetwork(learningRatio=lr,topology=topology[i],numIteration
91          nn.fit(X,Y)
92          confusionMatrix=nn.predict(forTestX,forTestY)
93          cm_df = pd.DataFrame(confusionMatrix,
94                               index = ['setosa','versicolor','Iris-virginica'],
95                               columns = ['setosa','versicolor','Iris-virginica'])
96          sns.heatmap(cm_df, annot=True,cbar=False)
97          ax.set_title(labels[i]+"learningRatio="+str(lr))
98          plt.savefig(name+str(i)+".png")
99          plt.clf()
100         #plt.show()
101 if __name__=='__main__':
102     topologys=[[4,4,3],[4,6,3],[4,8,3],[4,10,3],[4,12,3]]
103     plotError(topologys,0.04,"errorlr04")
104     #plotConfusinMatrix(topologys,0.01,"CMmatrixForLr01-")
```

0.1.3. Clase Utilitaria para Leer el dataset

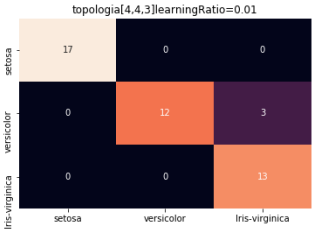
```
1  import pandas as pd
2  import numpy as np
3  import plotly.plotly as py
4  import plotly.tools as tls
5  import matplotlib.pyplot as plt
6  import seaborn as sns
7  from sklearn.utils import shuffle
8  import seaborn as sns
9  import time
10 from IPython.display import clear_output
11 %matplotlib inline
12 class ReadData(object):
13     def __init__(self,datasetName='Iris.csv'):
14         self.datasetName=datasetName
15     def readData(self):
16         df = pd.read_csv('Iris.csv')
17         df = df.drop(['Id'],axis=1)
18         #rows = list(range(100,150))
19         #df = df.drop(df.index[rows])
20         Y = []
21         target = df['Species']
22         for val in target:
23             if(val == 'Iris-setosa'):
24                 Y.append(0)
25             elif(val=='Iris-versicolor'):
26                 Y.append(1)
27             else:
28                 Y.append(2)
29         df = df.drop(['Species'],axis=1)
30         X = df.values.tolist()
31         datafeatureSize=50
```

```
32          labels = np.array([0]*datafeatureSize + [1]*datafeatureSize + [2]*data
33          Y = np.zeros((datafeatureSize*3, 3))
34          for i in range(datafeatureSize*3):
35              Y[i, labels[i]] = 1
36          X, Y = shuffle(X,Y)
37          X=np.array(X)
38          Y=np.array(Y)
39          forTestY=Y[105:]
40          forTestX=X[105:,:]
41          X=X[0:105,:]
42          Y=Y[0:105,:]
43          return X,Y,forTestX,forTestY
44  r=ReadData()
45  r.readData()
46  [X,Y,forTestX,forTestY]=r.readData()
```

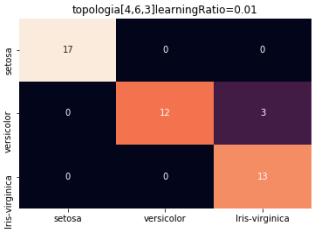
0.2. Pruebas cambiando la topologia y el parametro de aprendizaje

Se pide probar la red neuronal cambiando la topologia específicamente solo la capara hidden por [4,6,8,10,12] y los parámetros de aprendizaje [0.01,0.04,0.07,0.105]

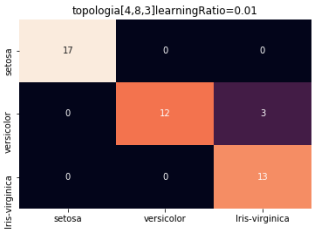
0.2.1. Resultados al entrenar la red neuronal con 1000



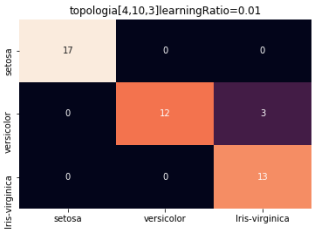
(A) topología=[4,4,3]



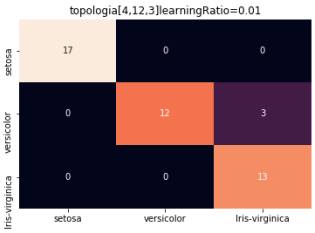
(B) topología=[4,6,3]



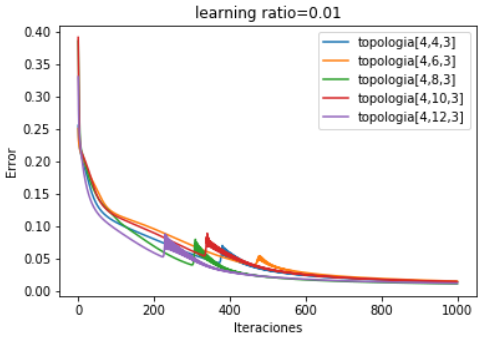
(C) topología=[4,8,3]



(D) topología=[4,10,3]

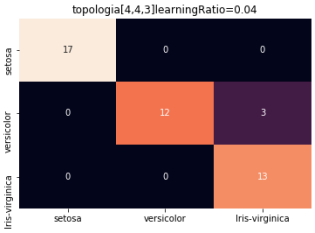


(E) topología=[4,12,3]

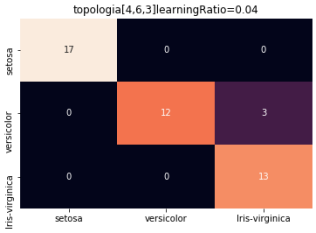


(F) error-numIteraciones

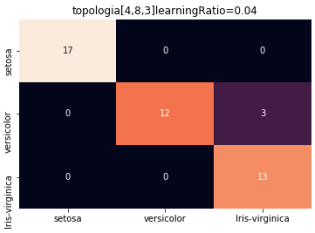
FIGURA 1: Matrices de confusión para cada topología con el parámetro de aprendizaje=0.01 mas la gráfica del error con respecto al numero de iteraciones



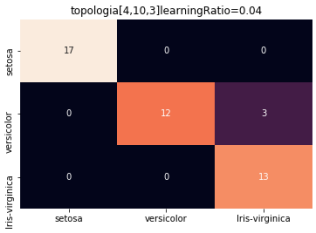
(A) topología=[4,4,3]



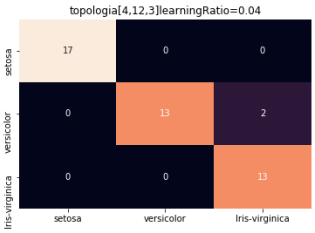
(B) topología=[4,6,3]



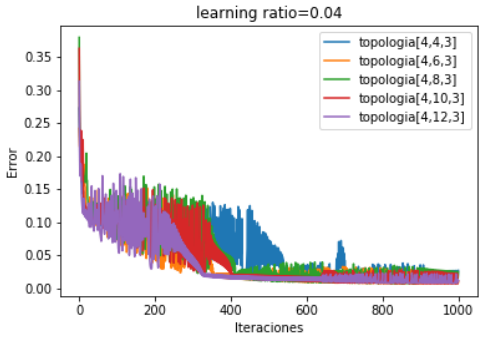
(C) topología=[4,8,3]



(D) topología=[4,10,3]

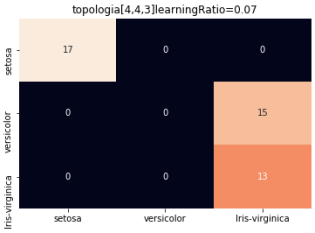


(E) topología=[4,12,3]

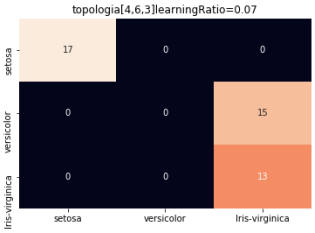


(F) error-numIteraciones

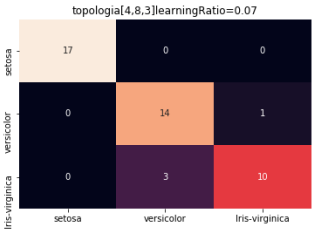
FIGURA 2: Matrices de confusión para cada topología con el parámetro de aprendizaje=0.04 mas la gráfica del error con respecto al numero de iteraciones



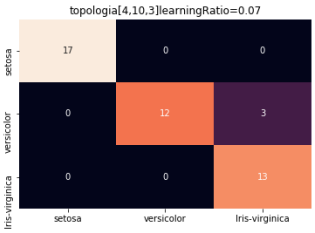
(A) topología=[4,4,3]



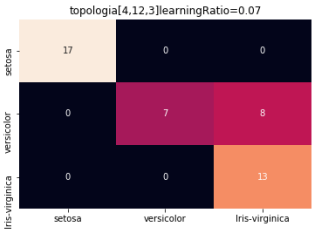
(B) topología=[4,6,3]



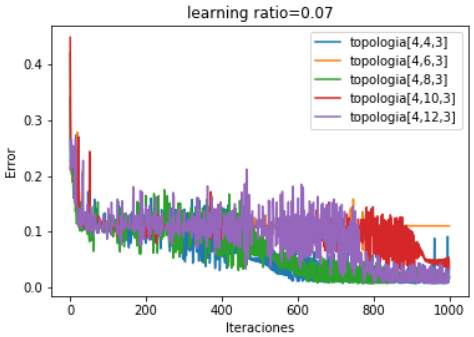
(C) topología=[4,8,3]



(D) topología=[4,10,3]

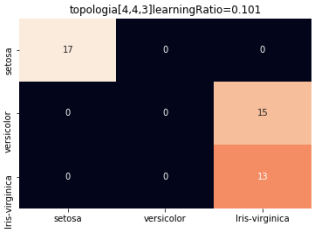


(E) topología=[4,12,3]

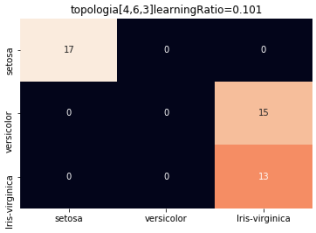


(F) error-numIteraciones

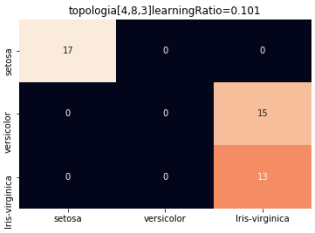
FIGURA 3: Matrices de confusión para cada topología con el parámetro de aprendizaje=0.07 mas la gráfica del error con respecto al numero de iteraciones



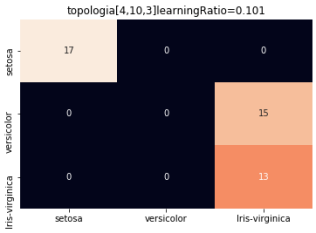
(A) topología=[4,4,3]



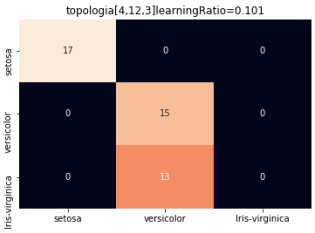
(B) topología=[4,6,3]



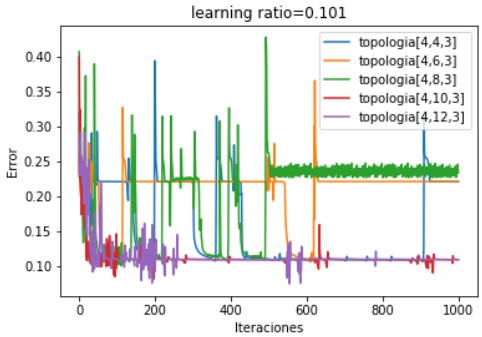
(C) topología=[4,8,3]



(D) topología=[4,10,3]



(E) topología=[4,12,3]



(F) error-numIteraciones

FIGURA 4: Matrices de confusión para cada topología con el parámetro de aprendizaje=0.101 mas la gráfica del error con respecto al numero de iteraciones

0.3. Conclusiones

El numero de iteraciones con el que se hizo todas la pruebas es 1000, cuando el parámetro de aprendizaje es muy bajo como 0,01 o 0,04 se puede podrían lograr predecir de una manera aceptable tal como se representa en las matrices de confusión 1 y 2: sin embargo cuando el parámetro de aprendizaje es mayor se observa una manera mas brusca de entrenamiento tal como se representa en las gráfica de error con respecto al numero de iteraciones 3 y en el ultimo caso cuando el parámetro de aprendizaje es 0.101 4 es donde se observa que el error no converge a 0 esto seria un underfitting esto se puede solucionar de alguna manera haciendo mas grande el numero de iteraciones como se observa en la siguiente imagen 5 en la que algunas topologías con el parámetro de aprendizaje 0.101 logran converger sin embargo hay otros que no convergen como el [4,6,3] y [4,12,3] Por ultimo donde se obtienen mejo-

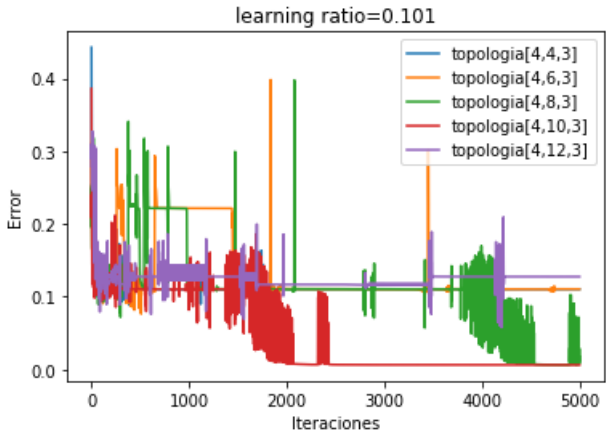


FIGURA 5: 5000 iteraciones

res resultado es cuando el parámetro de aprendizaje es pequeño como el 0.01 o 0.04 sin embargo cuando el parámetro de aprendizaje sea muy pequeño la red neuronal necesitaría muchas iteraciones para poder entrenar y si la red esta demasiado sobre entrenada podría sufrir de overfiting