

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE
CIENCIA DE LA COMPUTACIÓN



UNSA
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

INTELIGENCIA ARTIFICIAL

Redes Neuronales

Docente:
Cristian lopes

Integrantes:
Quispe Totocayo, Raul Edgar

8 de junio de 2019

Índice general

0.1. Redes Neuronales	1
0.1.1. BackPropagation	1
Como varia el <i>coste</i> ante un cambio del parametro W ?	1
Algoritmo	1
0.1.2. Implementacion de la Red Neuronal	2
0.1.3. Clase Utilitaria para Leer el dataset	4
0.2. Pruebas cambiando la topologia y el parametro de aprendizaje	5
0.2.1. Error	5
0.3. Conclusiones	7

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 :

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

derivada de la suma ponderada con respecto a b :

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

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

```

```

38         if i==len(self.neuralNetwork)-1:#para la ultima capa
39             deltas.insert(0,self.costFunction[1](a,Y)*self.neuralNetwork
40         else:#para las demas capas
41             deltas.insert(0,deltas[0] @_W.T * self.neuralNetwork[i].
42         _W=self.neuralNetwork[i].W
43         ##desenso del gradiente
44         self.neuralNetwork[i].bayas=self.neuralNetwork[i].bayas-np.me
45         self.neuralNetwork[i].W=self.neuralNetwork[i].W-out[i][1].T@d
46     return out[-1][1]
47 def fit(self,X,Y):
48     loss=[]
49     for i in range(self.numIterations):
50         out=self.backPropagation(X,Y)
51         loss.append(self.costFunction[0](out,Y))
52         clear_output(wait=True)
53         plt.plot(range(len(loss)),loss)
54         plt.show()
55
56 def predict(self,X,Y):
57     confusionMatrix=[[0,0],[0,0]]
58     outPut=[]
59     for i in range(X.shape[0]):
60         out=self.forwardPropagation(X[i:i+1,:],Y[i])
61         outPut.append(out[-1][1])
62         outPut[i]=outPut[i].flatten()
63         outPut[i]=np.asscalar(outPut[i])
64         if outPut[i]>0.5 and Y[i]==1:
65             confusionMatrix[0][0]=confusionMatrix[0][0]+1
66         elif outPut[i]<=0.5 and Y[i]==1:
67             confusionMatrix[0][1]=confusionMatrix[0][1]+1
68         elif outPut[i]<=0.5 and Y[i]==0:
69             confusionMatrix[1][1]=confusionMatrix[1][1]+1
70         elif outPut[i]>0.5 and Y[i]==0:
71             confusionMatrix[1][0]=confusionMatrix[1][0]+1
72         #print("salida ",outPut[i],"salida deseada",Y[i])
73     print(confusionMatrix)
74     cm_df = pd.DataFrame(confusionMatrix,
75                           index = ['setosa','versicolor'],
76                           columns = ['setosa','versicolor'])
77     sns.heatmap(cm_df,annot=True)
78     plt.show()
79     N = len(Y)
80     x = range(N)
81     xx=np.array(x)
82     xx=xx+0.35
83     width = 1/1.5
84     plt.bar(x,Y,width=0.35,color="blue")
85     plt.bar(xx,outPut,width=0.35,color="red")
86     plt.legend(["Y","Y predicho"])
87 if __name__=='__main__':
88     nn1=NeuralNetwork(learningRatio=0.04,topology=[4,8,1],numIterations=300)

```

```

89     nn1.fit(X,Y)
90     nn1.predict(forTestX ,forTestY)

```

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
21         Y = []
22         target = df[' Species ' ]
23         for val in target:
24             if (val == ' Iris -setosa '):
25                 Y.append(0)
26             else:
27                 Y.append(1)
28         df = df.drop([' Species '],axis=1)
29         X = df.values.tolist()
30         X, Y = shuffle(X,Y)
31         X=np.array(X)
32         Y=np.array(Y)
33         forTestY=Y[70:]
34         Y=Y[: ,np.newaxis]
35         forTestX=X[70:,:]
36         X=X[0:70,:]
37         Y=Y[0:70,:]
38         return X,Y,forTestX ,forTestY
39 r=ReadData()
40 [X,Y,forTestX ,forTestY]=r.readData()

```

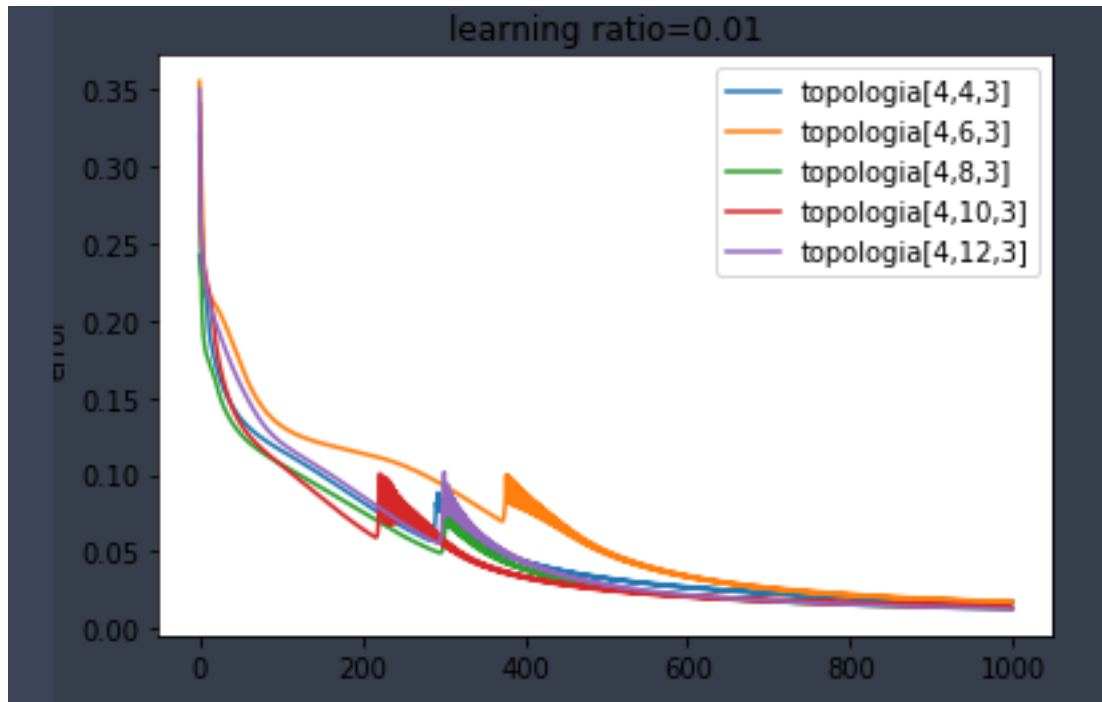


FIGURA 1: con lr =0.01

0.2. Pruebas cambiando la topologia y el parametro de aprendizaje

Se pide probar la red neuronal cambiando la topologia especificamente solo la capara hidden por [4,6,8,10,12] y los parametros de aprendizaje [0,01,0,04,0,07,0,105]

0.2.1. Error

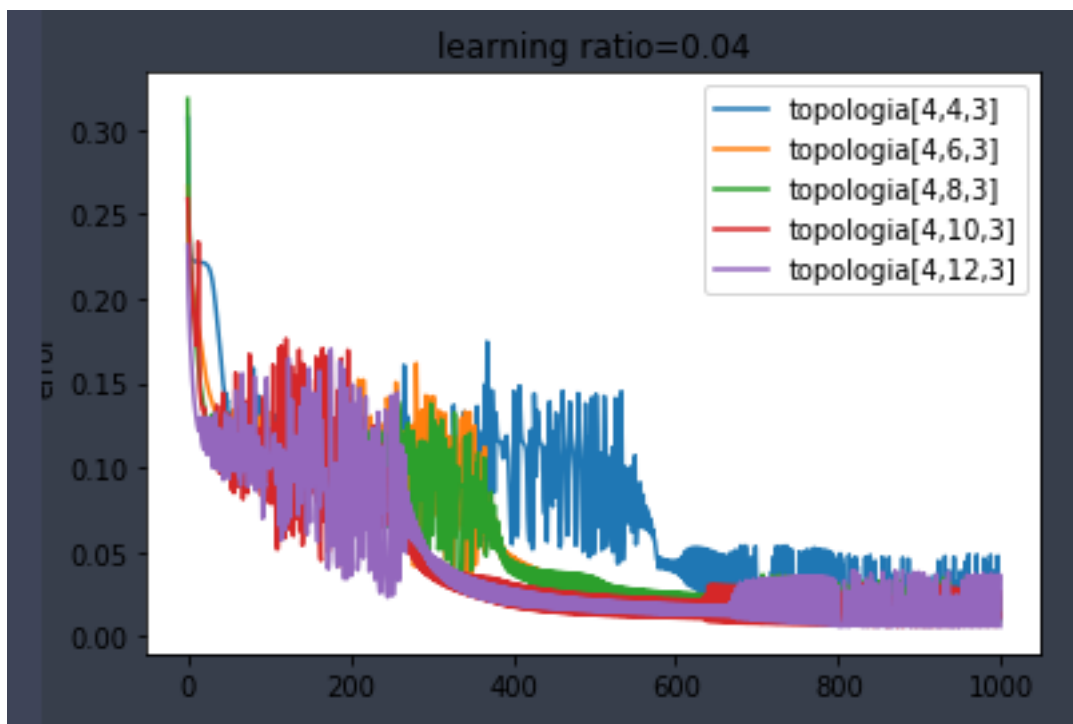


FIGURA 2: lr=0.04

FIGURA 3: lr=0.07

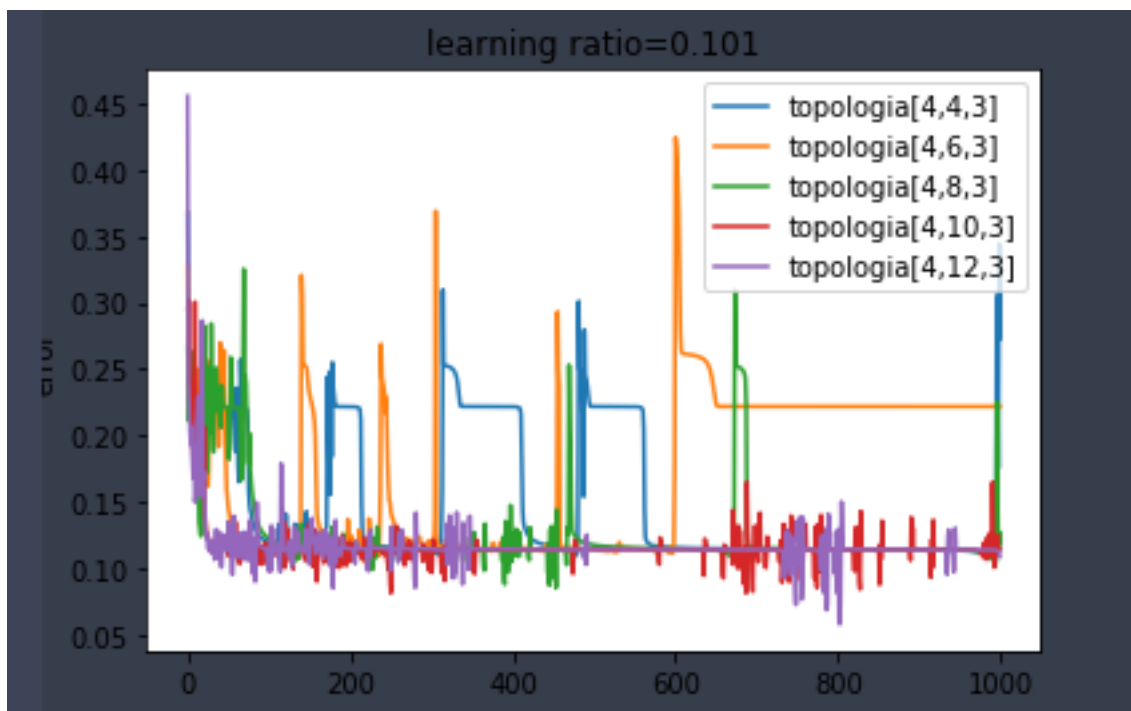


FIGURA 4: lr=0.101

0.3. Conclusiones

El numero de iteraciones con el que se hizo todas la pruebas es 300, cuando el parametro de aprendizaje es muy bajo como 0,01 o 0,04 se puede observar en el grafico de resultado obtenido vs esperado que hay mas error que cuando los parametros de aprendizaje son mas altos como 0.07 o 0.105 en los cuales casi no hay error .esto se debe a que con los parámetros de aprendizaje mas altos se puede entrenar mas rápido , y la red esta tendiendo mas al overfitting para el caso de los parámetros de aprendizaje 0.01 y 0.04 se tendría que hacer mas iteraciones para que el error se reduzca .

La matriz de confusión resulta igual para todas las pruebas por que por que se esta de la siguiente manera :

- para que pertenezca a la clase 1 el resultado obtenido debe ser mayor que 0.5
- para el caso contrario osea que pertenezca a la clase 0 tiene que se menor o igual a 0.5

es por eso que la red esta prediciendo correctamente con las condiciones antes mencionadas

Con respecto a la topología se observa que mientras mas capas hidden hay también el error de predicción disminuye.