Tarea 7: Aplicación perceptrón multicapa

En la tabla adjunta se muestran los datos de 1000 clientes que solicitaron créditos a un banco dado. La última columna muestra la información de los clientes que cayeron en mora en algún momento del período del crédito. El monto máximo de crédito que puede asignarse son $300,000 y la antigüedad laboral máxima que se toma en cuenta para asignar el crédito es de 15 años (es decir, antigüedades mayores ya no generan más posibilidad de ser aprobado).

Se busca una relación entre la información presentada (que se obtiene al contratar el crédito) y la posibilidad de que el cliente caiga en mora en algún momento del plazo.

Entrene un perceptrón multicapa para encontrar una relación tomando como entradas el monto solicitado (normalizado), la carga que implica al salario el pago de la mensualidad y la antigüedad laboral al contratar (normalizada).

Utilice 70-30 de relación entrenamiento-prueba y calcule el accuracy.

## **Desarrollo de actividad**

El número de neuronas ocultas es a su criterio.

Para el proyecto se utilizarán las variables de monto, antigüedad, mensualidad e ingreso mensual para predecir si un cliente caerá en mora o no.

Se presentan los histogramas a continuación

Chart

Description automatically generated

Estas variables serán normalizadas de la siguiente manera:

* Al monto se le aplicará la raíz cuadrada y a esos valores se le restará la media y dividirá la desviación estándar
* La variable de antigüedad se dividirá entre el valor máximo de la antigüedad laboral
* Finalmente se creara una variable de carga que considerara la mensualidad de pago y el ingreso mensual

Tras el tratamiento de los datos se presentan las variables transformadas donde monto presenta una distribución cercana a la normal. Antigüedad y carga tienen un comportamiento similar a una distribución uniforme

Chart

Description automatically generated

Una vez transformados los datos se inicializan las variables de la siguiente manera para la red neuronal:

|  |  |
| --- | --- |
| **Variables** | |
| Proporción train/test | 70/30 |
| Alpha | 0.01 |
| Neuronas | 5 |
| Épocas | 1500 |

Se hace la corrida con los datos de entrenamiento obteniendo la siguiente información

|  |  |
| --- | --- |
| **Corrida de entrenamiento** | |
| Tiempo de entrenamiento | 35.21 segundos |
| Accuracy Train | 98.85% |

Con esa información se hace la validación para el los datos de test obteniendo la siguiente información

|  |  |
| --- | --- |
| **Corrida de prueba** | |
| Accuracy Test | 98% |

## **Anexo 1 Codigo empleado**

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
import numpy as np  
from pytictoc import TicToc  
t = TicToc()  
df = pd.read\_excel("PercMultAplicado.xlsx")  
#%%  
df.head()  
#%%  
df[["Monto","Antigüedad laboral (meses)","Mensualidad","Ingreso mensual"]].hist(figsize=(12,6))  
  
#%%  
df["Carga"] = df["Mensualidad"]/df["Ingreso mensual"]  
df["Monto"] = df["Monto"]\*\*(1/2)  
df["Monto"] = (df["Monto"]-df["Monto"].mean())/df["Monto"].std()  
df["Antigüedad laboral (meses)"] = df["Antigüedad laboral (meses)"]/max(df["Antigüedad laboral (meses)"])  
df["Mora"] = df["Mora"]=="SI"  
  
  
#%%  
dffin = df[["Monto","Antigüedad laboral (meses)","Carga","Mora"]]  
dffin.head()  
dffin.hist(figsize=(12,6))  
  
  
#%%  
Xt,xt,Yt,yt=train\_test\_split(dffin[["Monto","Antigüedad laboral (meses)","Carga"]],dffin['Mora'],train\_size=0.7)  
  
#%%  
#Inicializacion de variables  
a= 1  
alpha = 0.01  
N = Xt.shape[1]#inputs  
M = 1 #Salidas conocidas  
Q = len(Xt)#Patrones de aprendizaje  
L = 4 #Neuronas  
epocas = 1500  
wh = np.random.uniform(-1, 1, (L, N))#Vector de pesos  
wo = np.random.uniform(-1, 1, (M, L))  
x = np.float64(Xt.to\_numpy())  
d = np.float64(Yt.to\_numpy())  
y = np.zeros([Q,M])  
  
#%%  
c = 0  
t.tic()  
for i in range(epocas):  
 #Forward  
 for i in range(Q):  
 net\_h = wh @ x[i].transpose()  
 y\_h = np.reshape(1/(1+np.exp(-a\*net\_h)),(L,1))  
 net\_o = wo @ y\_h  
 y = 1 / (1 + np.exp(-a\*net\_o))  
 #Backward  
 d\_o= ( np.reshape(d[i],(M,1)) - y)\*y\* (1 - y)  
 d\_h = y\_h \* (1 - y\_h) \* (np.transpose(wo) @ d\_o)  
  
 wo += alpha \* d\_o @ y\_h.transpose()  
 wh += alpha \* d\_h @ np.reshape(x[i], (1, N))  
 c +=1  
 if np.linalg.norm(d\_o) < 0.0001:  
 break  
  
t.toc()  
#%%  
#Originales  
res\_o = []  
for i in range(Q):  
 net\_h = wh @ x[i].transpose()  
 y\_h = np.reshape(1/(1+np.exp(-a\*net\_h)),(L,1))  
 net\_o = wo @ y\_h  
 y = 1 / (1 + np.exp(-a\*net\_o))  
 res\_o = np.append(res\_o,np.round(y))  
  
res\_o = np.reshape(res\_o,(Q,M))  
  
  
#%%  
accuracy = accuracy\_score(Yt,res\_o)  
accuracy  
  
#%%  
testlen = len(xt)  
xtest = np.float64(xt.to\_numpy())  
ytest = np.float64(yt.to\_numpy())  
restest = []  
for i in range(testlen):  
 net\_h = wh @ xtest[i].transpose()  
 y\_h = np.reshape(1/(1+np.exp(-a\*net\_h)),(L,1))  
 net\_o = wo @ y\_h  
 y = 1 / (1 + np.exp(-a\*net\_o))  
 restest = np.append(restest,np.round(y))  
  
restest = np.reshape(restest,(testlen,M))  
accuracy\_score(ytest,restest)