```
#-----------------------------------------------------
#-----------------------------------------------------
# Ejemplo de Uso de Redes Neuronales
# En estos ejemplos se muestra cómo hacer una red neuornal desde cero y cómo entrenar una red neuronal usando sklearn.
#-----------------------------------------------------

#-----------------------------------------------------
#0. Setup inicial
#
# Liberías necesarias para que funcione el algoritmo.
#-----------------------------------------------------
#=====================================================
# Setup de librerías
#-----------------------------------------------------
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from   sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
from   IPython.display import clear_output
#=====================================================


#-----------------------------------------------------
# Carga de base de datos
#-----------------------------------------------------
#=====================================================
# Carga de base de datos
!wget https://www.dropbox.com/. . ./G4.zip
!unzip -qq G4
#=====================================================


#-----------------------------------------------------
# Funciones necesarias
#-----------------------------------------------------
#=====================================================
def load_features(prefix):
    Xtrain  = np.load(prefix+'_Xtrain.npy')            # training samples
    Xtest = np.load(prefix+'_Xtest.npy')             # testing samples
    ytrain  = np.ravel(np.load(prefix+'_dtrain.npy'))  # training labels
    ytest = np.ravel(np.load(prefix+'_dtest.npy'))   # testing labels
    print('Training data: '+str(Xtrain.shape[0]) +' samples with '+str(Xtrain.shape[1]) +' features')
    print(' Testing data: '+str(Xtest.shape[0])+' samples with '+str(Xtest.shape[1])+' features')
    print('      Classes: '+str(int(np.min(ytrain)))+'...'+str(int(np.max(ytrain))))
    return Xtrain,ytrain,Xtest,ytest

def print_confusion(dt,ds,show_heatmap=0,Cnorm=1):
    # dt - true, ds - predicted
    C   = confusion_matrix(dt,ds)
    print('Confusion Matrix:')
    print(C)
    acc = accuracy_score(dt,ds)
    acc_st = "{:.2f}".format(acc*100)
    print('Accuracy = '+str(acc_st))
    if show_heatmap:
      sns.heatmap(C/Cnorm, annot=True, cbar=None, cmap="Blues")
      plt.title("Confusion Matrix"), plt.tight_layout()
      plt.ylabel("True Class"), plt.xlabel("Predicted Class")
      plt.show()

def plot_features(X,d,st,show=1):
    dmin = int(np.min(d))
    dmax = int(np.max(d))
    #colors =
```

```
    np.array(["red","green","blue","yellow","pink","black","orange","purple","beige","brow
    n","gray","cyan","magenta"])
67      #colors = 'Greens'
68      for j in range(dmin,dmax+1):
69          plt.scatter(X[d==j,0],X[d==j,1],label=str(j),s=27)
70      plt.grid(True)
71      plt.legend()
72      plt.xlabel('$x_1$',fontsize=14)
73      plt.ylabel('$x_2$',fontsize=14)
74      plt.title('Feature Space - '+st,fontsize=14)
75      if show==1:
76          plt.show()
77
78  def plot_decision_lines(clf,X,show=0,decisionline=1):
79      # based on example of https://scikit-learn.org
80      h = 0.075
81      x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
82      y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
83      xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
84                           np.arange(y_min, y_max, h))
85      if decisionline == 1:
86          Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
87          Z = Z.reshape(xx.shape)
88          plt.contourf(xx, yy, Z, cmap=plt.cm.Reds, alpha=0.8)
89      plt.xlim(x_min, x_max)
90      plt.ylim(y_min, y_max)
91      if show==1:
92          plt.show()
93
94  def show_clf_results(clf,X,d,Xt,dt,d0,ds,st,decisionline=1):
95      gs = gridspec.GridSpec(1, 2)
96      fig = plt.figure(figsize=(18, 6))
97      print('Training:')
98      acc = accuracy_score(d,d0)
99      accst = f'Acc = {acc:.4f}'
100     ax = plt.subplot(gs[0,0])
101     print_confusion(d,d0)                                   # confusion matrix in training
102     plot_decision_lines(clf,X,0,decisionline)               # decision lines
103     plot_features(X,d,st+' - Training: '+accst,0)           # feature space in training
104     ax = plt.subplot(gs[0,1])
105     print('Testing:')
106     acc = accuracy_score(ds,dt)
107     accst = f'Acc = {acc:.4f}'
108     print_confusion(dt,ds)                                  # confusion matrix in testing
109     plot_decision_lines(clf,X,0,decisionline)               # decision lines
110     plot_features(Xt,dt,st+' - Testing: '+accst,1)          # feature space in testing
111
112
113 def plot_loss(loss_train):
114     plt.figure(figsize=[8,6])
115     plt.plot(loss_train,'r',linewidth=1.5)
116     # plt.plot(loss_val,'b',linewidth=1.5)
117     # plt.legend(['Training loss', 'Validation Loss'],fontsize=14)
118     plt.xlabel('Epochs ',fontsize=14)
119     plt.ylabel('Loss',fontsize=14)
120     plt.title('Training Loss',fontsize=14)
121     xmin, xmax = plt.xlim()
122     ymin, ymax = plt.ylim()
123     plt.xlim(0,xmax)
124     plt.ylim(0,ymax)
125     plt.grid(True)
126     plt.show()
127 #=====================================================
128
129
130 #-----------------------------------------------------
131 # Training/Testing Data
132 #-----------------------------------------------------
133 #=====================================================
```

```
data = 'G4' # <= puede ser 'G2','G3','G4','P2','F2' (ver explicación más arriba)
print('Cargando '+data+'...')

Xtrain,ytrain,Xtest,ytest   = load_features(data + '/' + data)   # load training and testing data

plot_features(Xtrain,ytrain,'Training')
plot_features(Xtest,ytest,'Testing')
#=====================================================


#-----------------------------------------------------
# Logistic Regression (sklearn)
#-----------------------------------------------------
#=====================================================
from sklearn.linear_model import LogisticRegression
# Training
clf = LogisticRegression(C=0.1,solver="lbfgs")

clf.fit(Xtrain, ytrain)

# Evaluation on training subset
y0  = clf.predict(Xtrain)

# Evaluation on testing subset
ypred  = clf.predict(Xtest)

# Display results
show_clf_results(clf,Xtrain,ytrain,Xtest,ytest,y0,ypred,'LR',decisionline=1)
#=====================================================


#-----------------------------------------------------
# Redes Neuronales
#-----------------------------------------------------
# En este ejemplo, entrenaremos una red neuoronal como la que se muestra en la figura: 2 características de entrada ( x1,x2 ), 3 capas (2 ocultas  a1  de 6 nodos, y  a2  de 12 nodos), y una salida de 4 elementos, es decir 4 clases, tal cual como tiene la base de datos G4.
#-----------------------------------------------------
#=====================================================
# Definición de Redes Neuoronales "from Scratch"
# (solo con fines pedagógicos, en la práctica se usa sklearn, como
# el ejemplo mostrado en la siguiente sección)

def nn_definition(n,N):
    W  = [None]
    b  = [None]
    m1 = len(n)
    for k in range(1,m1):
        W    = W  + [np.random.rand(n[k],n[k-1])]
        b    = b  + [np.random.rand(n[k],1)]
    return W,b

def nn_forward_propagation(X,W,b):
    a = [None]
    a[0]  = X
    m1 = len(W)
    for k in range(1,m1):
        zk = W[k].dot(a[k-1])+b[k]
        a = a + [1/(1+np.exp(-zk))]
    return a

def nn_backward_propagation(Y,a,W,b):
    m = len(W)-1
    N = Y.shape[1]
    dam = a[m]-Y
    dW = [None]
    db = [None]
```

```
199
200         # Derivatives
201         for k in range(1,m+1):
202             dW   = dW + [np.zeros([W[k].shape[0],W[k].shape[1]])]
203             db   = db + [np.zeros([b[k].shape[0],1])]
204         for k in range(m,0,-1):
205             if k == m:
206                 dak = dam
207             ds    = np.multiply(a[k], 1-a[k])
208             Gk    = np.multiply(dak,ds)
209             dW[k] = np.matmul(Gk,a[k-1].transpose())/N
210             db[k] = (np.sum(Gk,axis=1,keepdims=True))/N
211             dak   = np.matmul(W[k].transpose(),Gk)
212 
213         return dW,db
214 
215     def nn_parameters_update(W,b,dW,db,alpha):
216         m1 = len(W)
217 
218         # Updates
219         for k in range(1,m1):
220             b[k] = b[k] - alpha*db[k]
221             W[k] = W[k] - alpha*dW[k]
222 
223         return W,b
224 
225     def nn_loss_function(a,Y):
226         m = len(a)-1
227         N = Y.shape[1]
228         Ys = a[m]
229         dam = Ys-Y
230         d2 = np.multiply(dam,dam)
231         ds = np.sqrt(np.sum(d2,axis=0,keepdims=True))/N
232         loss = np.sum(ds)
233         return loss
234 
235     # Conversion of labels: class <-> hot
236     # yclass : vector nx1 with m classes
237     # yhot   : matrix mxn where yhot(i,j)=c if y(j)==i
238 
239     def class2hot(yclass):
240       ymin = (np.min(yclass)).astype(int)
241       ymax = (np.max(yclass)).astype(int)
242       y = (yclass - ymin).astype(int)
243       m = ymax-ymin+1
244       n = y.shape[0]
245       yhot = np.zeros((m,n))
246       for i in range(n):
247         yhot[y[i],i] = 1
248       return yhot
249 
250     def hot2class(yhot):
251       yclass  = np.argmax(yhot,axis = 0)
252       return yclass
253     #=====================================================
254 
255 
256     #-----------------------------------------------------
257     # Training
258     #-----------------------------------------------------
259     #=====================================================
260     # Entrenamiento de una Red Neuoronal "from Scratch"
261     # (solo con fines pedagógicos, en la práctica se usa sklearn, como
262     # el ejemplo mostrado en la siguiente sección)
263 
264 
265     # Conversión de datos
266     Ytrain      = class2hot(ytrain)
267     Ytest       = class2hot(ytest)
```

```
XtrainT     = np.transpose(Xtrain)
XtestT      = np.transpose(Xtest)

# Definiciones
N           = XtrainT.shape[1]   # training samples
n_0         = XtrainT.shape[0]   # number of inputs (X)
n_m         = Ytrain.shape[0]    # number of outputs (Y)
tmax        = 1000               # max number of iterations
alpha       = 10                 # learning rate
loss_eps    = 0.01               # stop if loss<loss_eps
nh          = [6,12]             # nodes of hidden layers
n           = [n_0]+nh+[n_m]     # nodes of each layer
m           = len(n)-1
ltrain      = np.zeros([tmax,1]) # training loss

# Training
t     = -1
train =  1
W,b   = nn_definition(n,N)                              # (step 1)
while train:
    t         = t+1
    a         = nn_forward_propagation(XtrainT,W,b)    # (step 2)
    dW,db     = nn_backward_propagation(Ytrain,a,W,b)  # (step 3)
    W,b       = nn_parameters_update(W,b,dW,db,alpha)  # (step 4)
    ltrain[t] = nn_loss_function(a,Ytrain)             # (step 5)
    train     = ltrain[t]>=loss_eps and t<tmax-1

# Loss function on training and validation subsets
plot_loss(ltrain)

# Evaluation on training subset
print('Training:')
a = nn_forward_propagation(XtrainT,W,b)    # output layer is a[m]
y0 = hot2class(a[m])+np.min(ytrain)
print_confusion(ytrain,y0,show_heatmap=1,Cnorm=2)

# Evaluation on testing subset
print('Testing:')
a = nn_forward_propagation(XtestT,W,b)     # output layer is a[m]
ypred = hot2class(a[m])+np.min(ytest)
print_confusion(ytest,ypred,show_heatmap=1,Cnorm=1)
#=====================================================


#-----------------------------------------------------
# Redes Neuronales (sklearn)
#-----------------------------------------------------
# En la práctica se usa esta implementación de Redes Neuoronales
#-----------------------------------------------------
#=====================================================
from sklearn.neural_network import MLPClassifier

# Definitions
alpha       = 1e-5     # learning rate
nh          = (6,12)   # nodes of hidden layers
tmax        = 2000     # max number of iterations
solver      = 'adam'   # optimization approach ('lbfgs','sgd', 'adam')
nn_st       = 'NN-'+str(nh)

# Training
clf = MLPClassifier(solver=solver, alpha=alpha,hidden_layer_sizes=nh,
                    random_state=1,max_iter=tmax)
clf.fit(Xtrain, ytrain)

# Evaluation on training subset
y0  = clf.predict(Xtrain)

# Evaluation on testing subset
ypred  = clf.predict(Xtest)
```

```
337
338 # Display results
339 show_clf_results(clf,Xtrain,ytrain,Xtest,ytest,y0,ypred,nn_st,decisionline=1)
340 #====================================================
341
342
343 #----------------------------------------------------
344 # Redes Neuronales (sklearn)
345 #----------------------------------------------------
346 # En la práctica se usa esta implementación de Redes Neuoronales
347 #----------------------------------------------------
348 #====================================================
349 from sklearn.neural_network import MLPClassifier
350
351 # Definitions
352 alpha       = 1e-5     # learning rate
353 nh          = (6,12)   # nodes of hidden layers
354 tmax        = 2000     # max number of iterations
355 solver      = 'adam'   # optimization approach ('lbfgs','sgd', 'adam')
356 nn_st       = 'NN-'+str(nh)
357
358 # Training
359 clf = MLPClassifier(solver=solver, alpha=alpha,hidden_layer_sizes=nh,
360                     random_state=1,max_iter=tmax)
361 clf.fit(Xtrain, ytrain)
362
363 # Evaluation on training subset
364 y0  = clf.predict(Xtrain)
365
366 # Evaluation on testing subset
367 ypred  = clf.predict(Xtest)
368
369 # Display results
370 show_clf_results(clf,Xtrain,ytrain,Xtest,ytest,y0,ypred,nn_st,decisionline=1)
371 #====================================================
372
373 #====================================================
374 #====================================================
375 #====================================================
```