LABORATORY EXPERIENCE #7
(Classification through Neural Networks)

Introduction

This time we want to use neural networks to solve a classification problem on arrhythmia data. The experiment has to be repeated twice: the first time collapsing the 16 classes into two classes: negative and positive cases; the second time using all the 16 classes to divide the dataset.

Data preparation

Also in this case, we have to normalize the dataset after having isolated the last column that contains the classification provided by the medical doctors, that will be used to train the neural network.

Algorithms implementation and results

The neural network for the **binary classification** has been developed with 2 hidden layers, using the sigmoid function as non-linearity, as it gives at output a continuous value in a range with extremes 0 and 1, that can be the representatives of the two classes.

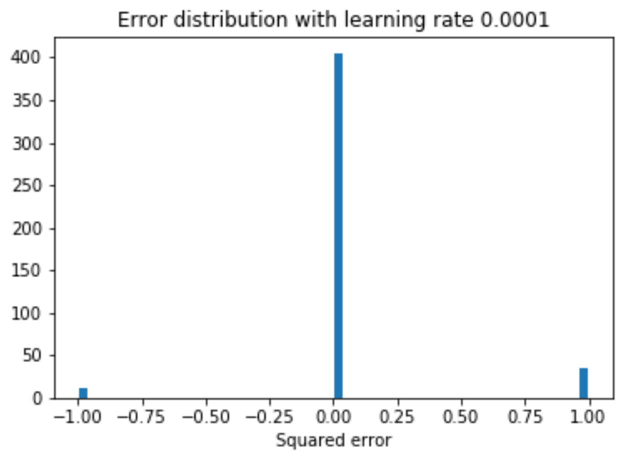
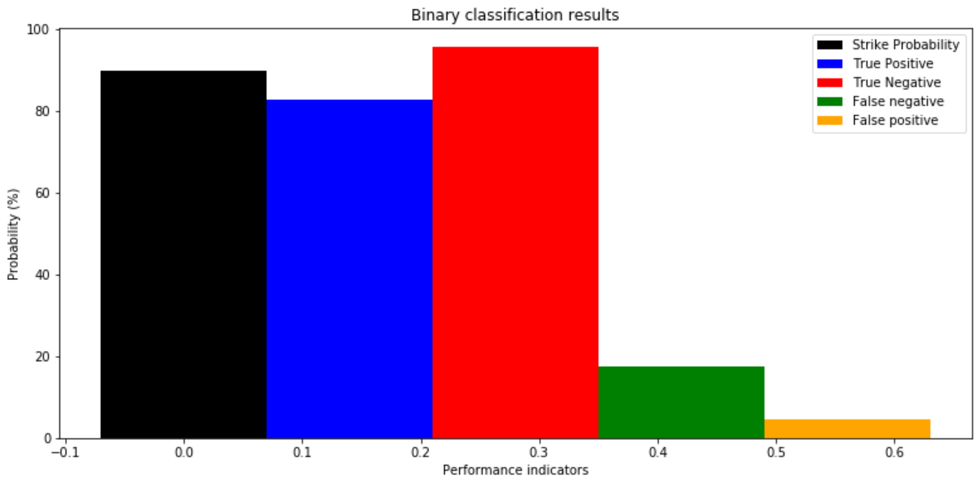


Figure 1: Histogram of the error distribution for binary classification

Figure 2: Evaluation of the classification performances



Using learning rate equal to 10^{-4} we obtain very good results in a reasonable time, with a strike probability over 90% and excellent performances in specificity.

In the case of **16 classes**, as the neural network is built, it gives as output a row of 16 elements for each patients. This row vector represents the probabilities for a patient to belong to one of these 16 classes. The assignment of the patient in a class is done finding the index for which we have the maximum probability.
In this case the classification is more difficult, but extending the execution of the optimization algorithm to 10000 steps instead of 1000 and setting the learning rate to 10^{-2}, we can obtain very good results, with a strike rate over 91%.

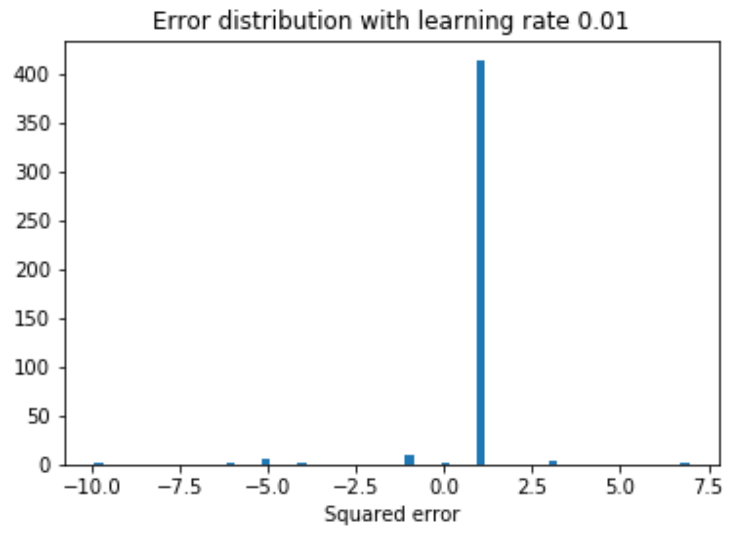


Figura 2: Histogram of the error distribution for 16-classes problem

LAB 07 – 2 CLASSES

```
# we import the .mat file of arrhythmia
import scipy.io as scio
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

mat_file=scio.loadmat('arrhythmia.mat')
data=mat_file.get('arrhythmia')

data = data[~np.all(data==0, axis=1)] # deleting eventual zero columns
class_id=data[:,-1]
class_id[np.where(class_id > 1)]=2
class_id=class_id-1

data=data[:,:-1]
(N,F)=np.shape(data)

mean=np.mean(data)
std=np.std(data)
x_norm=(data-mean)/std

mean = np.mean(x_norm,0)
var = np.var(x_norm,0)

n_healthy=sum(class_id==0)
n_ill=sum(class_id==1)

# initializing the neural network graph
tf.set_random_seed(1234)
learning_rate = 1e-4
n_hidden_nodes_1=F
n_hidden_nodes_2=128

x = tf.placeholder(tf.float64, [N, F])
t = tf.placeholder(tf.float64, [N, 1])

# first layer
w1 = tf.Variable(tf.random_normal(shape=[F, n_hidden_nodes_1], mean=0.0, stddev=1.0,
dtype=tf.float64, name="weights"))
b1 = tf.Variable(tf.random_normal(shape=[1, n_hidden_nodes_1], mean=0.0, stddev=1.0,
dtype=tf.float64, name="biases"))
a1 = tf.matmul(x, w1) + b1
z1 = tf.nn.sigmoid(a1)

# second layer
```

```
w2 = tf.Variable(tf.random_normal(shape=[n_hidden_nodes_1, n_hidden_nodes_2], mean=0.0,
stddev=1.0, dtype=tf.float64, name="weights2"))
b2 = tf.Variable(tf.random_normal(shape=[1, n_hidden_nodes_2], mean=0.0, stddev=1.0,
dtype=tf.float64, name="biases2"))
a2 = tf.matmul(z1, w2) + b2
z2 = tf.nn.sigmoid(a2)

# second layer
w3 = tf.Variable(tf.random_normal(shape=[n_hidden_nodes_2, 1], mean=0.0, stddev=1.0,
dtype=tf.float64, name="weights3"))
b3 = tf.Variable(tf.random_normal(shape=[1, 1], mean=0.0, stddev=1.0, dtype=tf.float64,
name="biases3"))
y = tf.nn.sigmoid(tf.matmul(z2, w3) + b3)

#implementation of gradient algorithm
cost = tf.reduce_sum(tf.squared_difference(y, t, name="objective_function"))
optim = tf.train.GradientDescentOptimizer(learning_rate, name="GradientDescent")
optim_op = optim.minimize(cost, var_list=[w1,b1,w2,b2,w2,b3])

init = tf.global_variables_initializer()

#--- run the learning machine
sess = tf.Session()
sess.run(init)

xval = x_norm.reshape(N,F)
tval = class_id.reshape(N, 1)
for i in range(1000):
    # generate the data
    # train
    input_data = {x: xval, t: tval}
    sess.run(optim_op, feed_dict = input_data)
    if i % 100 == 0:# print the intermediate result
        print(i,cost.eval(feed_dict=input_data,session=sess))


#--- print the final results
print(sess.run(w1),sess.run(b1))
print(sess.run(w2),sess.run(b2))
print(sess.run(w3),sess.run(b3))

a = sess.run(w1)
yval = np.round(y.eval(feed_dict = input_data, session=sess))
yval= np.array(yval, dtype=np.int32).reshape(len(yval),)


hist, bins = np.histogram((class_id-yval), bins=50)
width = 0.7 * (bins[1] - bins[0])
```

```
center = (bins[:-1] + bins[1:]) / 2
plt.bar(center, hist, align='center', width=width)
plt.xlabel('Squared error')
plt.title('Error distribution with learning rate '+str(learning_rate))
plt.savefig('squared_error_2classes'+ str(learning_rate)+'.png',format='png')
plt.show()


n_strike = float((yval == class_id).sum())
p_strike = 100.0*n_strike/N
p_true_positive = float(100*((yval >= 1) & (class_id >= 1)).sum())/n_ill
p_true_negative = float(100*((yval == 0) & (class_id == 0)).sum())/n_healthy
p_false_positive = float(100*((yval >= 1) & (class_id == 0)).sum())/n_healthy
p_false_negative = float(100*((yval == 0) & (class_id >= 1)).sum())/n_ill



plt.figure(figsize=(13,6))
index = np.arange(0,1,1)

bars0 = [p_strike]
bars1 = [p_true_positive]
bars2 = [p_true_negative]
bars3 = [p_false_negative]
bars4 = [p_false_positive]


plt.bar(index, bars0, 0.14, label="Strike Probability", color="black")
plt.bar(index + 0.14, bars1, 0.14, label="True Positive", color="blue")
plt.bar(index + 0.28, bars2, 0.14, label="True Negative", color="red")
plt.bar(index + 0.42, bars3, 0.14, label="False negative", color="green")
plt.bar(index + 0.56, bars4, 0.14, label="False positive", color="orange")
plt.title("Binary classification results")
plt.xlabel("Performance indicators")
plt.ylabel("Probability (%)")
plt.legend(loc=0, framealpha=0.7)
```

LAB 07 – 16 CLASSES

```
# we import the .mat file of arrhythmia
import scipy.io as scio
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

mat_file=scio.loadmat('arrhythmia.mat')
data=mat_file.get('arrhythmia')

data = data[~np.all(data==0, axis=1)] # deleting eventual zero columns
class_id=data[:,-1]
n_classes=int(max(class_id))

(N,F)=np.shape(data)

mx_classes=np.zeros((N,n_classes))
for i in range(0,N-1):
    mx_classes[i][int(class_id[i])-1] = 1


data=data[:,:-1]
(N,F)=np.shape(data)

mean=np.mean(data)
std=np.std(data)
x_norm=(data-mean)/std

mean = np.mean(x_norm,0)
var = np.var(x_norm,0)

n_healthy=sum(class_id==0)
n_ill=sum(class_id==1)

# initializing the neural network graph
tf.set_random_seed(1234)
learning_rate = 1e-2
n_hidden_nodes_1=64
n_hidden_nodes_2=32


x = tf.placeholder(tf.float64, [N, F])
t = tf.placeholder(tf.float64, [N, n_classes])

# first layer
w1 = tf.Variable(tf.random_normal(shape=[F, n_hidden_nodes_1], mean=0.0, stddev=1.0,
dtype=tf.float64, name="weights"))
```

```
b1 = tf.Variable(tf.random_normal(shape=[1, n_hidden_nodes_1], mean=0.0, stddev=1.0,
dtype=tf.float64, name="biases"))
a1 = tf.matmul(x, w1) + b1
z1 = tf.nn.sigmoid(a1)

# second layer
w2 = tf.Variable(tf.random_normal(shape=[n_hidden_nodes_1, n_hidden_nodes_2], mean=0.0,
stddev=1.0, dtype=tf.float64, name="weights2"))
b2 = tf.Variable(tf.random_normal(shape=[1, n_hidden_nodes_2], mean=0.0, stddev=1.0,
dtype=tf.float64, name="biases2"))
a2 = tf.matmul(z1, w2) + b2
z2 = tf.nn.sigmoid(a2)

# second layer
w3 = tf.Variable(tf.random_normal(shape=[n_hidden_nodes_2, n_classes], mean=0.0, stddev=1.0,
dtype=tf.float64, name="weights3"))
b3 = tf.Variable(tf.random_normal(shape=[1, 1], mean=0.0, stddev=1.0, dtype=tf.float64,
name="biases3"))
y = tf.nn.softmax(tf.matmul(z2, w3) + b3)

#implementation of gradient algorithm
cost = tf.reduce_sum(tf.squared_difference(y, t, name="objective_function"))
optim = tf.train.GradientDescentOptimizer(learning_rate, name="GradientDescent")
optim_op = optim.minimize(cost, var_list=[w1,b1,w2,b2,w2,b3])

init = tf.global_variables_initializer()

#--- run the learning machine
sess = tf.Session()
sess.run(init)

xval = x_norm.reshape(N,F)
tval = mx_classes.reshape(N, n_classes)
for i in range(10000):
   # generate the data
   # train
   input_data = {x: xval, t: tval}
   sess.run(optim_op, feed_dict = input_data)
   if i % 1000 == 0:# print the intermediate result
      print(i,cost.eval(feed_dict=input_data,session=sess))


#--- print the final results
print(sess.run(w1),sess.run(b1))
print(sess.run(w2),sess.run(b2))
print(sess.run(w3),sess.run(b3))
```

```
decisions = np.zeros(N)
yval = y.eval(feed_dict=input_data,session=sess)
for i in range (0,N):
   decisions[i]=np.argmax(yval[i])

hist, bins = np.histogram((class_id-decisions), bins=50)
width = 0.7 * (bins[1] - bins[0])
center = (bins[:-1] + bins[1:]) / 2
plt.bar(center, hist, align='center', width=width)
plt.xlabel('Squared error')
plt.title('Error distribution with learning rate '+str(learning_rate))
plt.savefig('squared_error_16classes'+ str(learning_rate)+'.png',format='png')
plt.show()

n_strike = float((decisions == class_id-1).sum())
p_strike = 100.0*n_strike/N
```