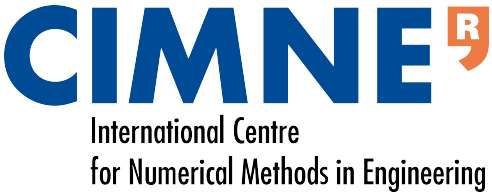
Practical Session 3

Small Neural Network

Alex Ferrer Ferre
Antonio Darder Bennassar

July 2022



Contents

1 Introduction

In this session we will increase the difficulty for the supervised classification task. The multi-layer perceptrons are one of the most well known classification algorithms which serve this purpose. MLPs are complex structures made by simple functions, they only use the same equations used for logistic regression.

$$h = X\theta \tag{1}$$

$$a = \frac{1}{1 + e^{-h}} \tag{2}$$

$$J = \frac{1}{m} \sum_{i}^{m} \sum_{j}^{n} y_i^j \cdot ln(\hat{y}_i^j) + (1 - y_i^j) \cdot ln(1 - \hat{y}_i^j) \tag{3}$$

The idea of the neural network is that now there will be several layers increasing the capacity of the model. Forward propagation is the function that performs the concatenation of hypothesis function and sigmoid through all the layers until reaching the last layer where the cost function is applied.

The other important concept in neural networks is backward propagation and it can be a little more confusing. The aim of backprop is to compute the gradient of the function by applying the chain rule from outer layers to inner layers. Quick recap of the formulas for backprop are:

$$\frac{\partial a}{\partial h} = a \cdot (1 - a) \tag{4}$$

$$\nabla J_{\hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \tag{5}$$

$$\delta^{(L+1)} = \nabla J_{\hat{y}} \cdot \frac{\partial a^{(L)}}{\partial h} \tag{6}$$

$$\delta^{(l)} = \delta^{(l+1)} \times {\theta^{(l)}}^T \cdot \frac{\partial a^{(l)}}{\partial h} \tag{7}$$

$$\nabla J_{\theta^{(l)}} = \frac{1}{m} {a^{(l)}}^T \times \delta^{(l+1)} \tag{8}$$

A pseudo code of backprop is given for clarification.

Algorithm 1 Backward propagation

Require:

θ parameters
y matrix of labels related to X
a activations at each layer
topology the network architecture

Ensure:

grad gradient of the cost function

1: $i \leftarrow length(topology)$
2: **while** $i \geq 2$ **do**
3: $a' \leftarrow \frac{\partial a^{(i)}}{\partial h}$
4: **if** i = length(topology) **then**
5: $\delta^{(i)} \leftarrow \nabla J_{\hat{y}} \cdot a'$
6: **else**
7: $\delta^{(i)} \leftarrow \delta^{(i+1)} \times {\theta^{(i)}}^T \cdot a'$
8: $grad^{(i-1)} \leftarrow \frac{1}{m} {a^{(i-1)}}^T \times \delta^{(i)}$
9: $i \leftarrow i - 1$

1.1 The datasets

First we will use the same dataset as the one used for logistic regression, the microchip dataset. Afterwards we will move to multi classification with the iris dataset.

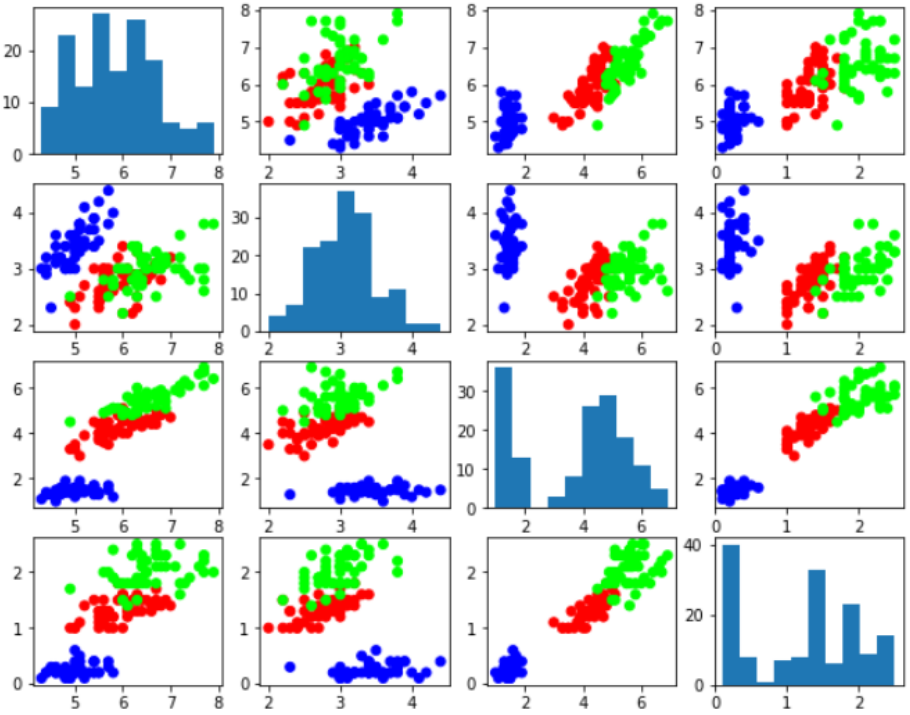


Figure 1 Iris dataset

2 Code

2.1 Step 1 & 2: Run the first 4 cells

Like in the other lessons we will import the libraries, add the class data, and we will also add the plot functions from the previous lesson.

2.2 Step 3: Computing the derivatives

So as we know we will use the same functions as the ones used in logistic regression. Yesterday we computed the gradient directly but now we will need to apply backward propagation so we need the partial derivatives.

```
def hypothesisFunction(X,theta):
  h = X@theta
  return h

def actFCN(z):
  g = 1/(1+math.e**(-z))

  return g,g_der

def crossentroy(P,T):
  logP0 = np.where(P > 10**(-16), P, -16)
  logP1 = np.where(1-P > 10**(-16), 1-P, -16)
  L     = -np.mean(np.sum(T*np.log(logP0,out=logP0,where=logP0>0)+(1-T)
     *np.log(logP1,out=logP1,where=logP1>0),axis=1))

  return L,L_der

def regularization(l,theta):
  r  = l*0.5*np.sum(theta**2)

  return r,r_der
```

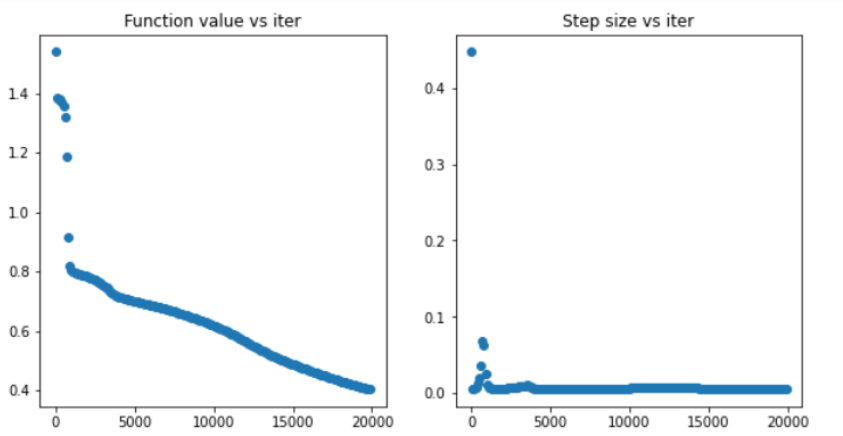
2.3 Step 4: Forward and backward propagation

Next step will be to programm the forward propagation and backward propagation for the neural network. These two have to pass through all the layers to obtain its results.

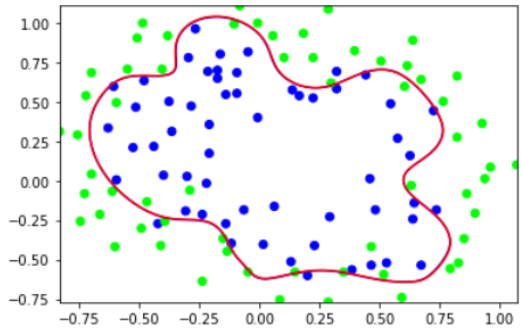
```
    def forwardprop(theta,topology,X,Y,l):
      a = [X]
      cont = 0
      for k in range(len(topology)-1):
        theta_l = np.reshape(theta[cont:cont+topology[k]*topology[k+1]],(
            topology[k],topology[k+1]))
        cont = cont + topology[k]*topology[k+1]

      return J,a

  def backprop(theta,topology,a,Y,l):
      delta = []
      m     = Y.shape[0]
      cont  = len(theta)
      grad  = np.zeros((theta.shape[0],1))
      for k in range(len(topology)-1,0,-1):

      grad = grad + r_der
      return grad
```

Once we have these formulas we can start playing around with the hyperparameters running the next cell:

```
d = 2
l = 0.0
idx = [0,1]
hiddenlayers = [4,8]
lr = 0.5
theta0,Xtr,Ytr,topology = buildmodel(data.Xtrain,data.Ytrain,d,idx,hiddenlayers)
_,Xte,Yte,_ = buildmodel(data.Xtest,data.Ytest,d,idx,hiddenlayers)

F = lambda theta,X,Y: forwardprop(theta,topology,X,Y,l)
f = lambda theta,a,Y: backprop(theta,topology,a,Y,l)
gd = SGD(lr=lr,epochs=20000,batch=Xtr.shape[0],plot=True)
thetaOPT = gd.train(F,f,Xtr,Ytr,theta0)
PlotBoundary(data,thetaOPT,d,idx,topology)

tr = testAccuracy(Xtr,Ytr,thetaOPT,topology,l)
te = testAccuracy(Xte,Yte,thetaOPT,topology,l)
print('\nTrain accuracy: ',tr,'\nTest accuracy: ',te)
plt.show()
```



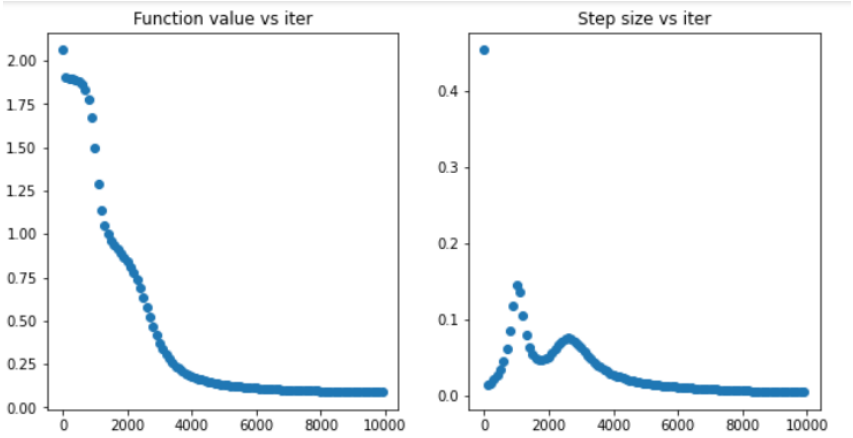

```
Train accuracy:  0.9032258064516129
Test accuracy:  0.8333333333333334
```



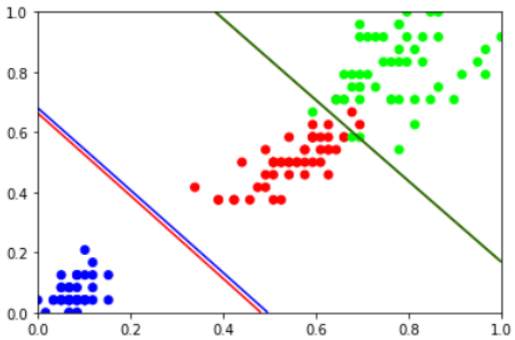
2.4 Step 5: Multi-classification

For multi-classification we will use the iris dataset, with the network that we already programmed we can play with the parameters again

```
d = 1
l = 0.0
idx = [2,3]
hiddenlayers = [4,8]
lr = 0.1
theta0,Xtr,Ytr,topology = buildmodel(iris.Xtrain,iris.Ytrain,d,idx,hiddenlayers)
_,Xte,Yte,_ = buildmodel(iris.Xtest,iris.Ytest,d,idx,hiddenlayers)

F = lambda theta,X,Y: forwardprop(theta,topology,X,Y,l)
f = lambda theta,a,Y: backprop(theta,topology,a,Y,l)
gd = SGD(lr=lr,epochs=10000,batch=Xtr.shape[0],plot=True)
thetaOPT = gd.train(F,f,Xtr,Ytr,theta0)
PlotBoundary(iris,thetaOPT,d,idx,topology)

tr = testAccuracy(Xtr,Ytr,thetaOPT,topology,l)
te = testAccuracy(Xte,Yte,thetaOPT,topology,l)
print('\nTrain accuracy: ',tr,'\nTest accuracy: ',te)
plt.show()
```



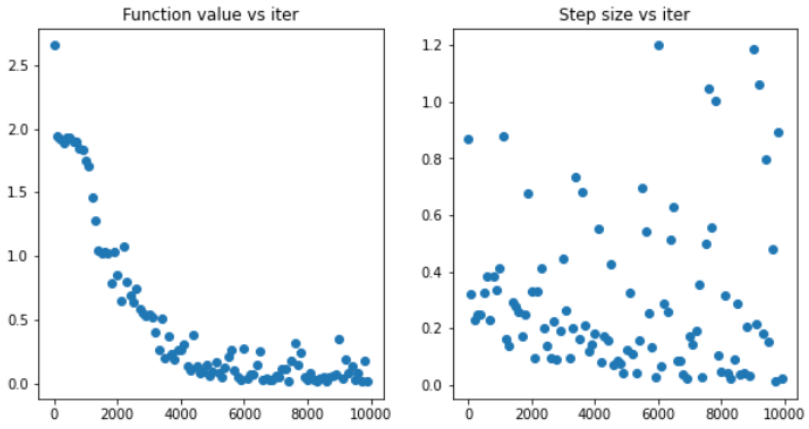

```
rain accuracy:  0.9831932773109243
est accuracy:  0.9
```

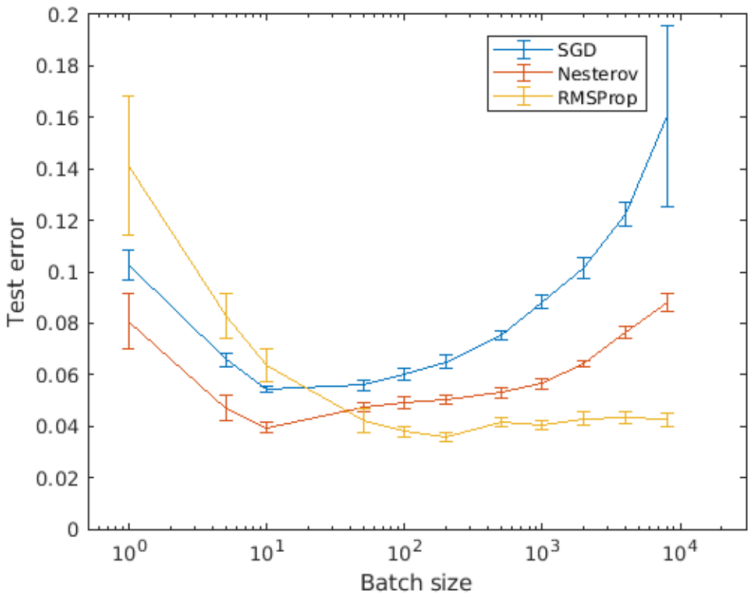


2.5 Step 6: Batch Size

Lastly we can see the influence of the batch size in the minimization and in the norm of the gradient.

```
d = 1
l = 0.0
idx = [2,3]
hiddenlayers = [4,8]
BS = Xtr.shape[0]//10
lr = 0.1
theta0,Xtr,Ytr,topology = buildmodel(iris.Xtrain,iris.Ytrain,d,idx,hiddenlayers)
_,Xte,Yte,_ = buildmodel(iris.Xtest,iris.Ytest,d,idx,hiddenlayers)

F = lambda theta,X,Y: forwardprop(theta,topology,X,Y,l)
f = lambda theta,a,Y: backprop(theta,topology,a,Y,l)
gd = SGD(lr=lr,epochs=10000,batch=BS,plot=True)
thetaOPT = gd.train(F,f,Xtr,Ytr,theta0)
PlotBoundary(iris,thetaOPT,d,idx,topology)

tr = testAccuracy(Xtr,Ytr,thetaOPT,topology,l)
te = testAccuracy(Xte,Yte,thetaOPT,topology,l)
print('\nTrain accuracy: ',tr,'\nTest accuracy: ',te)
plt.show()
```




SGD
Nesterov
RMSProp
Test error
Batch size
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
10^0
10^1
10^2
10^3
10^4