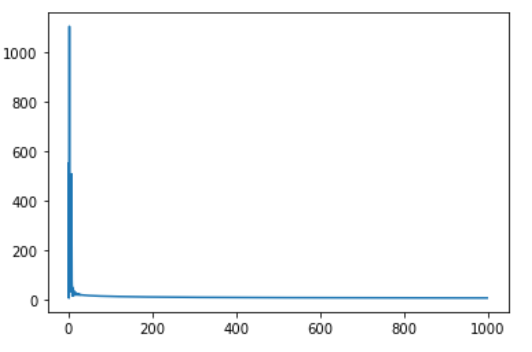
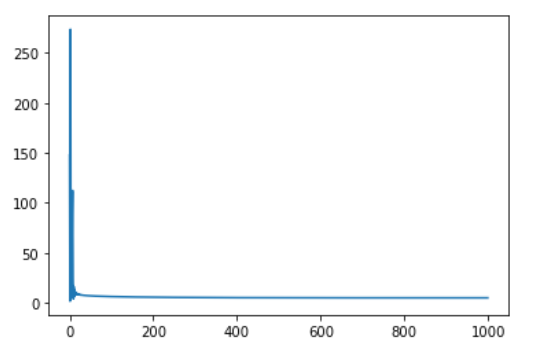
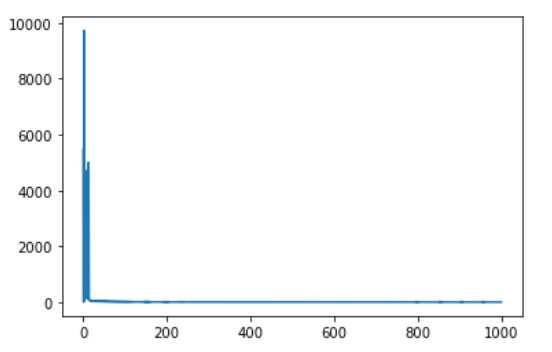
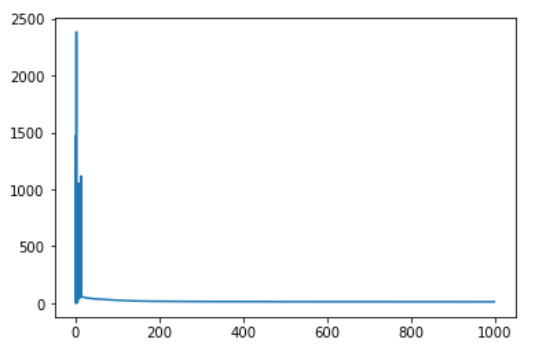
WINE DATASET

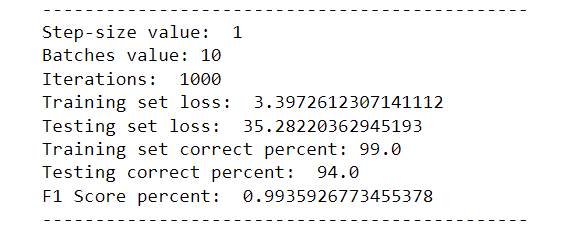
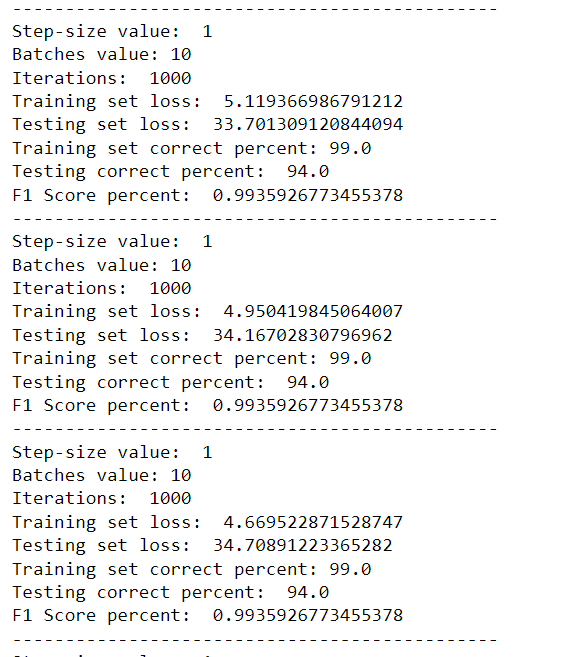
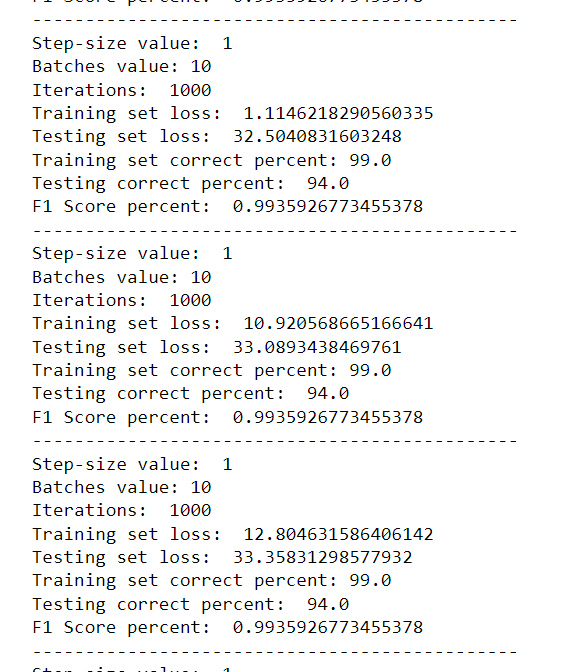
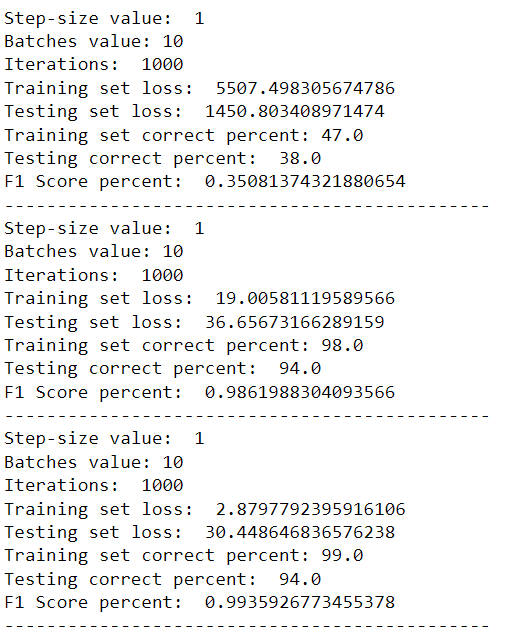
Number of batches vs Training Loss(X-Axis/Y-Axis): **Step-size 0.1**  


Number of batches vs Testing Loss(X-Axis/Y-Axis): **Step-size=0.1**  


Number of batches vs Training Loss(X-Axis/Y-Axis): **Step-size = 1**  


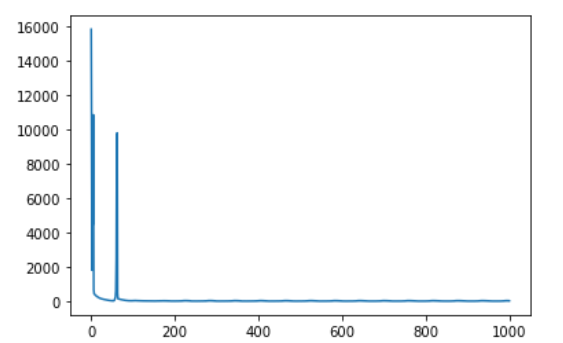
Number of batches vs Testing Loss(X-Axis/Y-Axis): **Step-size=1**

****

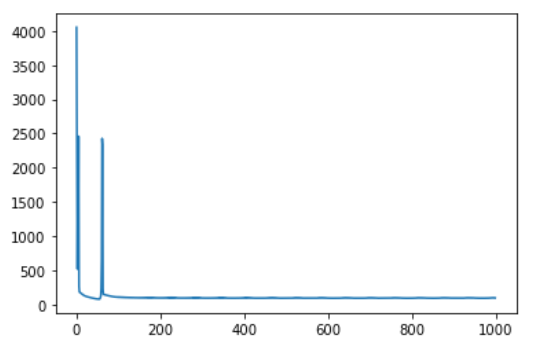
**We can see that training loss reduces greatly with lower stepsize value.  
Stepsize greatly affects the training loss. We can see max loss =10000 for stepsize=1, under 2500 for stepsize=0.1  
After a certain point there is no point in increasing iterations or the number of neurons as the training loss reduces by an insignificant margin.**

HOUSE VOTES DATASET  
  
Reducing stepsize significantly reduces the training loss even at the earlier stages. This is what impacts the algorithm the most

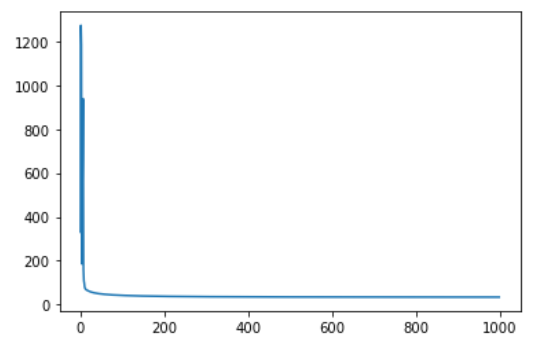
Number of batches vs Training Loss(X-Axis/Y-Axis): **Step-size 0.1**



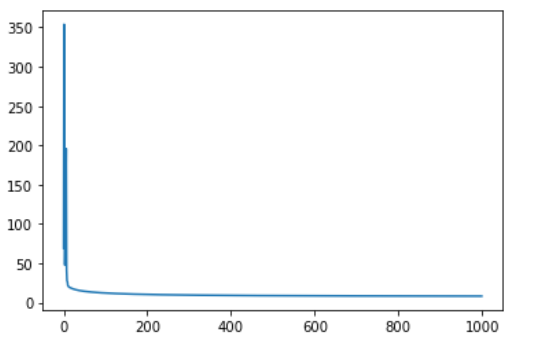
Number of batches vs Testing Loss(X-Axis/Y-Axis): **Step-size 0.1**



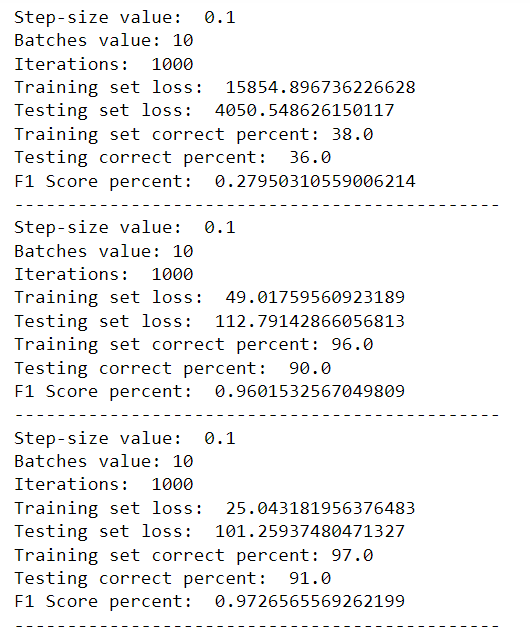
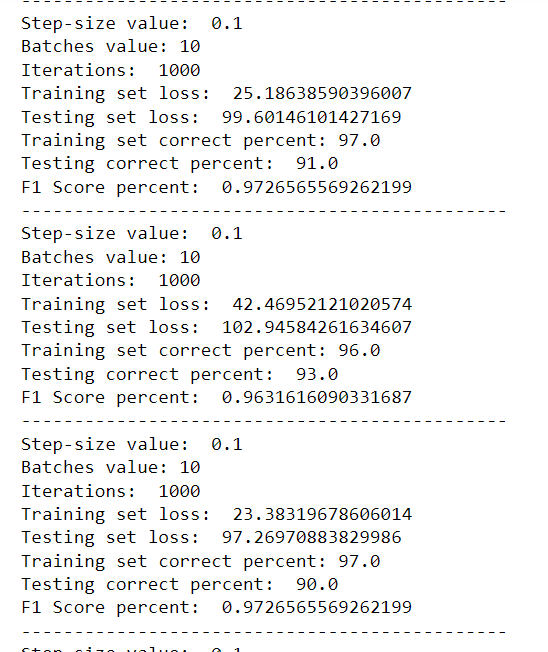
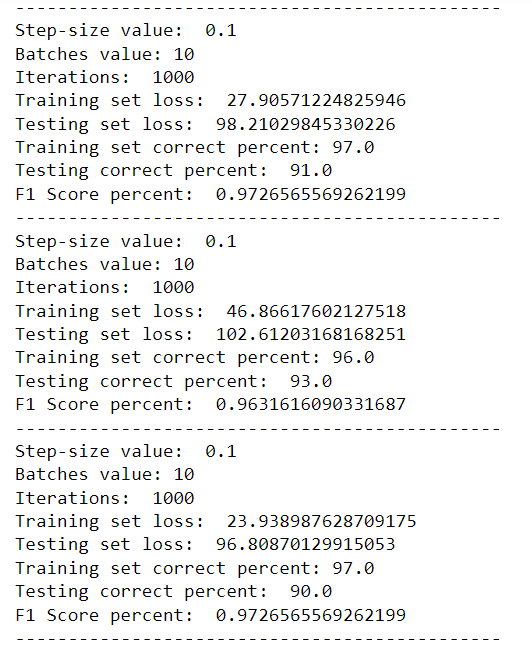
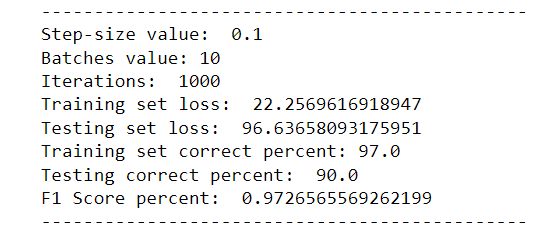
Number of batches vs Training Loss(X-Axis/Y-Axis): **Step-size 0.01**

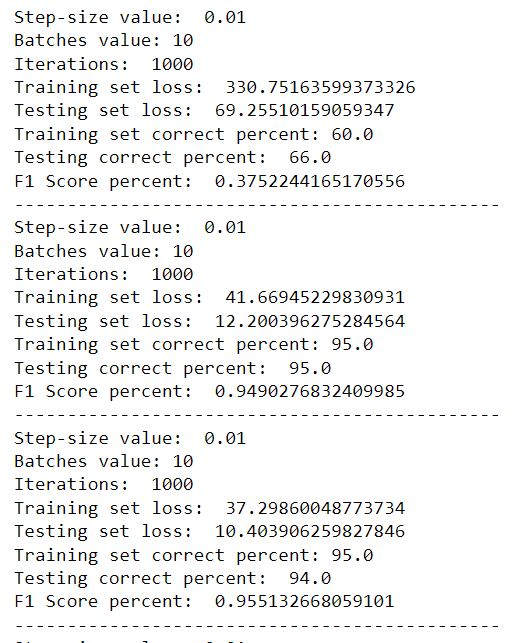


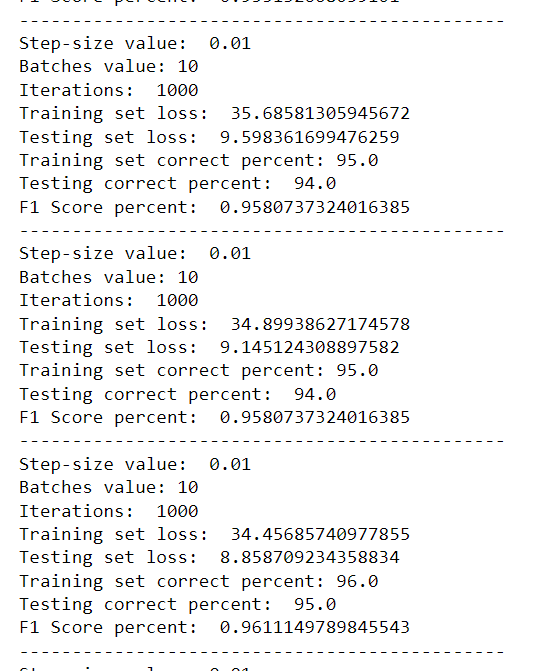
Number of batches vs Testing Loss(X-Axis/Y-Axis): **Step-size 0.01**

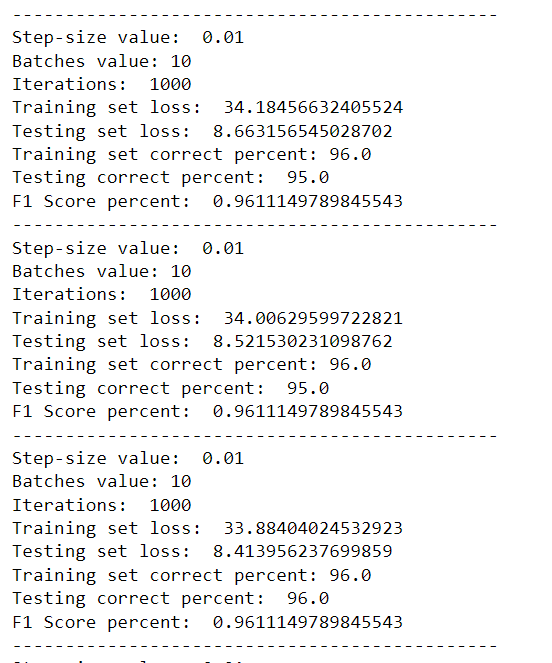


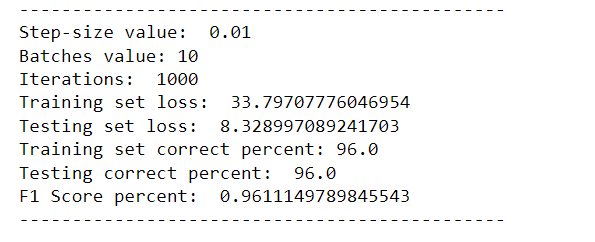
As we reduce Step-Size, the model converges significantly faster to higher accuracy.









def loss(WT,X,y):

nimages = X.shape[1]

c = WT.shape[0]

S = np.dot(WT,X)

P = logistic\_regression(S)

Pyi = P[ y, np.arange(nimages) ] # select the prob of the true class

li = -np.log(Pyi) # cross-entropy

L = li.sum() # this is the loss

# back-prop of the gradient of the loss

dLdli = np.ones\_like(li)

dLdP = np.zeros\_like(P)

dLdP[ y, np.arange(nimages) ] = dLdli \* (-1/Pyi)

dLdS = np.zeros\_like(S)

for m in range(c):

dLdS += dLdP[m]\*(-P[m]\*P)

dLdS += dLdP\*P

dLdWT = np.dot(dLdS,X.T) # finally, this is the gradient of the loss

ypred = np.argmax(P,axis=0)

return L,dLdWT,ypred

I would implement Network with lower number of network layers along with a smaller step-size value(0.1,0.01) as this will help with quickly reaching the minimum. A