CA2 Review

from SG 1

From the loss-iteration plot in page 11, which is not supposed to decrease linearly, we think there are bugs in the implementation, maybe in the computation of gradient or implementation of the solvers. Also the loss is already extremely small at iteration 0, compared to the loss at iteration 50, which indicates there is something wrong in the code.

For example:

a. in your “SGD_Batch” implementation, you use both “i” in the inner and outer loops, which is wrong.

```
elif (optimizer == "SGD_Batch"):
    #computes the SGD with respect to the whole batch
    for i in range(num_iters):
        D,N=x.shape
        #Random shuffling the batch
        for i in shuffle(np.arange(N)):
            g_i= function_gradient(x[:,i:i+1], y[i:i+1], w, lambda_)
            w=w-alpha*g_i
        if (np.linalg.norm(g_i) <= epsilon):
            break
```

b. It’s unclear what “ num_iters” represent in your plot. It is supposed to be “number of update iterations”, however, in your SGD, it seems like the number of epochs.

c. In the SVRG implementation. You preset T=100, when num_iter is below 50, K will be always set as 0. This would cause the strange result of your SGD,SVRG,SAG.

```
elif (optimizer == "SVRG"):
    T  = 100
    K  = math.floor(num_iters/T)
    N  = x.shape[1]
    wk_= w
    for k in range(K):
        ga_ = function_gradient(x, y, wk_, lambda_) #the average

        for t in range(T):
            index = np.random.randint(N, size=1)
            g1 = function_gradient(x[:,index], y[index,], w, lambda_)
            g2 = function_gradient(x[:,index], y[index,], wk_, lambda_)
            g  = g1-g2+ga_

            cost1 = cost(X_test,Y_test,w,lambda_)
            cost2 = cost(X_test,Y_test,w-alpha*g,lambda_)
            if ((cost2-cost1)/cost1>0.5):
                break

            w  = w-alpha*g

            if (np.linalg.norm(g) <= epsilon):
                break
        wk_ = w
```

[113]:
```
## Executing the iterations and plot the cost function here:

ti= np.zeros((50,4))
cost_= np.zeros((50,4))
for i in range(50):
    print("......",i,".......")
    #--------------GD------------------
    start = time.time()
    gde =␣
↪solver(X_train,Y_train,w,alpha,num_iters=i,lambda_=lambda_,epsilon=epsilon,optimizer="GD",m
    end = time.time()
```



```
    cost_[i,0] = cost(X_test,Y_test,gde,lambda_)

    ti[i,0] = end-start

    #---------------SGD------------------
    start = time.time()
    sgde =␣
↪solver(X_train,Y_train,w,alpha,num_iters=i,lambda_=lambda_,epsilon=epsilon,optimizer="SGD",
    end = time.time()

    cost_[i,1] = cost(X_test,Y_test,sgde,lambda_)

    ti[i,1] = end-start


    #---------------SVRG----------------
    start = time.time()
    sgde =␣
↪solver(X_train,Y_train,w,alpha,num_iters=i,lambda_=lambda_,epsilon=epsilon,optimizer="SVRG"
    end = time.time()

    cost_[i,2] = cost(X_test,Y_test,sgde,lambda_)

    ti[i,2] = end-start

    #---------------SAG------------------
    start = time.time()
    sgde =␣
↪solver(X_train,Y_train,w,alpha,num_iters=i,lambda_=lambda_,epsilon=epsilon,optimizer="SAG",
    end = time.time()

    cost_[i,3] = cost(X_test,Y_test,sgde,lambda_)

    ti[i,3] = end-start

    #-----------------------------------

    ## Pl the results:


l0 = plt.plot(cost_[:,0],color="red")
l0 = plt.plot(cost_[:,1],color="blue")
l0 = plt.plot(cost_[:,2],color="yellow")
```

All in all, because of the problems in your solver implementations, the results from experiments lack credibility to draw much interesting conclusions. We think implementations need to be improved to conduct the experiments to further complete the assignment.