```
import os,gc
import numpy as np
import scipy.optimize as opt
import scipy.io as sio
import torch
from torch.autograd import Variable, grad

# ---- Reconstruct marks. At initiation, every point has the average value of
the marks.----#
#---- Trying scipy L-BFGS ----#
def obj_fun(x,wph_ops,factr_ops,Sims,op_id):
    if x.grad is not None:
        x.grad.data.zero_()
    #global wph_ops
    wph_op = wph_ops[op_id]
    p = wph_op(x)
    diff = p-Sims[op_id]
    diff = diff * factr_ops[op_id]
    loss = torch.mul(diff,diff).sum()
    return loss

def grad_obj_fun(x_gpu,grad_err,wph_ops,factr_ops,Sims):
    loss = 0
    #global grad_err
    grad_err[:] = 0
    for op_id in range(len(wph_ops)):
        x_t = x_gpu.clone().requires_grad_(True)
        loss_t = obj_fun(x_t,wph_ops,factr_ops,Sims,op_id)
        grad_err_t, = grad([loss_t],[x_t], retain_graph=False)
        loss = loss + loss_t
        grad_err = grad_err + grad_err_t

    return loss, grad_err

from time import time
def fun_and_grad_conv(x,grad_err,wph_ops,factr_ops,Sims,size):
    x_float = torch.reshape(torch.tensor(x,dtype=torch.float),(1,1,size,size))
    x_gpu = x_float.cuda()
    loss, grad_err = grad_obj_fun(x_gpu,grad_err,wph_ops,factr_ops,Sims)
    return loss.cpu().item(),
np.asarray(grad_err.reshape(size**2).cpu().numpy(), dtype=np.float64)

def callback_print(x):
    return

def
call_lbfgs_routine(FOLOUT,labelname,im,wph_ops,Sims,N,Krec,nb_restarts,maxite,
factr,factr_ops,\
                       maxcor=20,gtol=1e-14,ftol=1e-
14,init='normal',toskip=True):
    grad_err = im.clone()
    size = N
    for krec in range(Krec):
        if init=='normal':
```

```
50              print('init normal')
51              x = torch.Tensor(1, 1, N, N).normal_()
52          elif init=='normal00105':
53              print('init normal00105')
54              x = torch.Tensor(1, 1, N, N).normal_(std=0.01)+0.5
55          elif "maxent" in init:
56              print('load init from ' + init)
57              xinit = sio.loadmat('./data/maxent/' + init + '.mat')
58              x = torch.from_numpy(xinit['imgs'][:,:,krec]) #  .shape)
59              #assert(false)
60          elif init=='normalstdbarx':
61              stdbarx = im.std()
62              print('init normal with std barx ' + str(stdbarx))
63              x = torch.Tensor(1, 1, N, N).normal_(std=stdbarx)
64          else:
65              assert(false)
66          x0 = x.reshape(size**2).numpy()
67          x0 = np.asarray(x0, dtype=np.float64)
68          x_opt = None
69          for start in range(nb_restarts+1):
70              time0 = time()
71              datname =  FOLOUT + '/' + labelname + '_krec' + str(krec) +
    '_start' + str(start) + '.pt'
72              if os.path.isfile(datname) and toskip:
73                  print('skip', datname)
74                  continue
75              else:
76                  print('save to',datname)
77
78              if start==0:
79                  x_opt = x0
80              elif x_opt is None:
81                  # load from previous saved file
82                  prename = FOLOUT + '/' + labelname + '_krec' + str(krec) +
    '_start' + str(start-1) + '.pt'
83                  print('load x_opt from',prename)
84                  saved_result = torch.load(prename)
85                  im_opt = saved_result['tensor_opt'].numpy()
86                  x_opt = im_opt.reshape(size**2)
87                  x_opt = np.asarray(x_opt,dtype=np.float64)
88
89              res = opt.minimize(fun_and_grad_conv, x_opt,args=
    (grad_err,wph_ops,factr_ops,Sims,size),\
90                                 method='L-BFGS-B', jac=True, tol=None,\
91                                 callback=callback_print,\
92                                 options={'maxiter': maxite, 'gtol': gtol,
    'ftol': ftol, 'maxcor': maxcor})
93              final_loss, x_opt, niter, msg = res['fun'], res['x'], res['nit'],
    res['message']
94              print('OPT fini avec:', final_loss,niter,msg)
95
96              im_opt = np.reshape(x_opt, (size,size))
97              tensor_opt = torch.tensor(im_opt,
    dtype=torch.float).unsqueeze(0).unsqueeze(0)
98
```

```
 99             ret = dict()
100             ret['tensor_opt'] = tensor_opt
101             ret['normalized_loss'] = final_loss/(factr**2)
102             torch.save(ret, datname)
103 
104             print('krec',krec,'strat', start, 'using time (sec):' , time()-
    time0)
105             time0 = time()
106 
```