Algorithm sL-BFGS-TR

This document provides step-by-step training and testing instructions

- for **LeNet-like**, **ResNet-20** or **ConvNet3FC2** without Batch Normalization Layers
- on **MNIST**, **Fashion-MNIST**, or **CIFAR10**.

Defaults:

`epoch = 10`

`lim_m = 20 (limited memory parameter)`

`os    = 500 (batch size = 2*os)`

```
clear
clc
close all
rng default
```

Load the data

Train and Test (images): 4D double

Train and Test (label): categorical

1. MNIST

```
[XTrain, yTrain, XTest, yTest] = loadDataMnist;
input_image_size               = [28 28 1];
```

2. Fashion-MNIST

```

% [XTrain, yTrain, XTest, yTest] = loadData_FashionMnist;
% input_image_size               = [28 28 1];
```

3. CIFAR10

```
% [XTrain, yTrain, XTest, yTest] = loadDataCifar10;
% input_image_size               = [32 32 3];
```

```
classes                      = categories(yTrain);
numClasses                   = numel(classes);  % 10
num_of_Train_Images          = size(XTrain,4);  % 50,000 (Cifar10) or 60,000 (Mnist and F.Mnis
num_of_Test_Images           = size(XTest,4);   % 10,000
```

One-hot label

```
% One-hot labels for train set:

YTrain = zeros(numClasses, num_of_Train_Images, 'single');
for c  = 2:10
    YTrain( c-1, yTrain == classes(c)) = 1;
end
YTrain( 10, yTrain == classes(1)) = 1;


% One-hot labels for test set:

YTest = zeros(numClasses, num_of_Test_Images, 'single');
for c = 2:10
    YTest( c-1, yTest == classes(c) ) = 1;
end
YTest( 10, yTest == classes(1) ) = 1;
```

Convert Test set from 4D double to 4D single > dlArray > gpuArray

```
executionEnvironment = "auto";

XTest   = dlarray(single(XTest),'SSCB');
if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
    XTest = gpuArray(XTest);
end
```

NOTE: This conversion is also required for the training set; (see `get_Jk.m`)

Architecture (without BN)

1. LeNet-like

```
% layers = [
%     imageInputLayer(input_image_size,'Normalization','none','Name','input')
%     convolution2dLayer(5,20,'Name', 'conv1')
%     reluLayer('Name','relu1')
%     maxPooling2dLayer(2,'Stride',2, 'Name', 'maxpool.1')
%     convolution2dLayer(5,50,'Name', 'conv2')
%     reluLayer('Name','relu2')
%     maxPooling2dLayer(2,'Stride',2, 'Name', 'maxpool.2')
%     fullyConnectedLayer(500,'Name','fc1')
%     reluLayer('Name','relu3')
%     fullyConnectedLayer(numClasses,'Name','fc2')
%     softmaxLayer('Name','softmax')];
```

2. ConvNet3FC2(no BN)

```
% layers = [
%     imageInputLayer(input_image_size, 'Name', 'input', 'Normalization','zscore',...
%     'Mean', mean(XTrain,4), 'StandardDeviation', std(XTrain, 0, 4))
```

```
%
%     % ConvNet
%
%     convolution2dLayer(5, 32, 'Stride', 1,'Padding', 2,'Name','conv1')
%     reluLayer('Name','relu1')
%     maxPooling2dLayer(2,'Stride', 1, 'Name', 'pooling_max1')
%
%     % ConvNet
%
%     convolution2dLayer(5, 32,'Stride', 1, 'Padding', 2,'Name','conv2')
%     reluLayer('Name','relu2')
%     maxPooling2dLayer(2,'Stride', 1, 'Name', 'pooling_max2')
%
%     % ConvNet
%
%     convolution2dLayer(5, 64,'Stride', 1,'Padding', 2,'Name','conv3')
%     reluLayer('Name','relu3')
%     maxPooling2dLayer(2,'Stride', 1, 'Name', 'pooling_max3')
%
%     % FC
%
%     fullyConnectedLayer(64,'Name','fc1')
%     reluLayer('Name','relu4')
%
%     % FC
%
%     fullyConnectedLayer(numClasses,'Name','fc2')
%     softmaxLayer('Name','softmax')];
%
%
% lgraph = layerGraph(layers);
```

3. ResNet-20(no BN)

```
lgraph = layerGraph();

tempLayers = [
    imageInputLayer(input_image_size, 'Name', 'imageinput', 'Normalization','zscore',...
    'Mean', mean(XTrain,4), 'StandardDeviation', std(XTrain, 0, 4))
    convolution2dLayer([3 3],16,"Name","conv_1","Padding",[1 1 1 1])
    reluLayer("Name","relu_1")];
lgraph = addLayers(lgraph,tempLayers);

% B1:

tempLayers = [
    convolution2dLayer([3 3],16,"Name","conv_2","Padding",[1 1 1 1])
    reluLayer("Name","relu_2")
    convolution2dLayer([3 3],16,"Name","conv_3","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_1")
    reluLayer("Name","relu_3")];
```

```
lgraph = addLayers(lgraph,tempLayers);


% B2:

tempLayers = [
    convolution2dLayer([3 3],16,"Name","conv_4","Padding",[1 1 1 1])
    reluLayer("Name","relu_4")
    convolution2dLayer([3 3],16,"Name","conv_5","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_2")
    reluLayer("Name","relu_5")];
lgraph = addLayers(lgraph,tempLayers);


% B3:

tempLayers = [
    convolution2dLayer([3 3],16,"Name","conv_6","Padding",[1 1 1 1])
    reluLayer("Name","relu_6")
    convolution2dLayer([3 3],16,"Name","conv_7","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_3")
    reluLayer("Name","relu_7")];
lgraph = addLayers(lgraph,tempLayers);

%===========================================================

% B1:

tempLayers = [
    convolution2dLayer([3 3],32,"Name","conv_8","Padding",[1 1 1 1],"Stride",[2 2])
    reluLayer("Name","relu_8")
    convolution2dLayer([3 3],32,"Name","conv_9","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([1 1],32,"Name","conv_10","Stride",[2 2])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_4")
    reluLayer("Name","relu_9")];
lgraph = addLayers(lgraph,tempLayers);


% B2:

tempLayers = [
    convolution2dLayer([3 3],32,"Name","conv_11","Padding",[1 1 1 1])
```

```
    reluLayer("Name","relu_10")
    convolution2dLayer([3 3],32,"Name","conv_12","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_5")
    reluLayer("Name","relu_11")];
lgraph = addLayers(lgraph,tempLayers);


% B3:

tempLayers = [
    convolution2dLayer([3 3],32,"Name","conv_13","Padding",[1 1 1 1])
    reluLayer("Name","relu_12")
    convolution2dLayer([3 3],32,"Name","conv_14","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_6")
    reluLayer("Name","relu_13")];
lgraph = addLayers(lgraph,tempLayers);

%===========================================================

% B1:

tempLayers = [
    convolution2dLayer([3 3],64,"Name","conv_15","Padding",[1 1 1 1],"Stride",[2 2])

    reluLayer("Name","relu_14")
    convolution2dLayer([3 3],64,"Name","conv_16","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);


tempLayers = [
    convolution2dLayer([1 1],64,"Name","conv_17","Stride",[2 2])];
lgraph = addLayers(lgraph,tempLayers);



tempLayers = [
    additionLayer(2,"Name","addition_7")
    reluLayer("Name","relu_15")];
lgraph = addLayers(lgraph,tempLayers);

% B2:


tempLayers = [
    convolution2dLayer([3 3],64,"Name","conv_18","Padding",[1 1 1 1])
    reluLayer("Name","relu_16")
    convolution2dLayer([3 3],64,"Name","conv_19","Padding",[1 1 1 1])];
```

```
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_8")
    reluLayer("Name","relu_17")];
lgraph = addLayers(lgraph,tempLayers);


% B3:

tempLayers = [
    convolution2dLayer([3 3],64,"Name","conv_20","Padding",[1 1 1 1])
    reluLayer("Name","relu_18")
    convolution2dLayer([3 3],64,"Name","conv_21","Padding",[1 1 1 1])];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    additionLayer(2,"Name","addition_9")
    reluLayer("Name","relu_19")
    globalAveragePooling2dLayer("Name","gapool")
    fullyConnectedLayer(10,"Name","fc")
    softmaxLayer("Name","softmax")];
lgraph = addLayers(lgraph,tempLayers);

% clean up helper variable
clear tempLayers;

lgraph = connectLayers(lgraph,"relu_1","conv_2");
lgraph = connectLayers(lgraph,"relu_1","addition_1/in2");
lgraph = connectLayers(lgraph,"conv_3","addition_1/in1");


lgraph = connectLayers(lgraph,"relu_3","conv_4");
lgraph = connectLayers(lgraph,"relu_3","addition_2/in2");
lgraph = connectLayers(lgraph,"conv_5","addition_2/in1");


lgraph = connectLayers(lgraph,"relu_5","conv_6");
lgraph = connectLayers(lgraph,"relu_5","addition_3/in2");
lgraph = connectLayers(lgraph,"conv_7","addition_3/in1");


lgraph = connectLayers(lgraph,"relu_7","conv_8");
lgraph = connectLayers(lgraph,"relu_7","conv_10");
lgraph = connectLayers(lgraph,"conv_10","addition_4/in2");
lgraph = connectLayers(lgraph,"conv_9","addition_4/in1");


lgraph = connectLayers(lgraph,"relu_9","conv_11");
lgraph = connectLayers(lgraph,"relu_9","addition_5/in2");
lgraph = connectLayers(lgraph,"conv_12","addition_5/in1");


lgraph = connectLayers(lgraph,"relu_11","conv_13");
```

```
lgraph = connectLayers(lgraph,"relu_11","addition_6/in2");
lgraph = connectLayers(lgraph,"conv_14","addition_6/in1");


lgraph = connectLayers(lgraph,"relu_13","conv_17");
lgraph = connectLayers(lgraph,"relu_13","conv_15");
lgraph = connectLayers(lgraph,"conv_16","addition_7/in1");
lgraph = connectLayers(lgraph,"conv_17","addition_7/in2");


lgraph = connectLayers(lgraph,"relu_15","conv_18");
lgraph = connectLayers(lgraph,"relu_15","addition_8/in2");
lgraph = connectLayers(lgraph,"conv_19","addition_8/in1");


lgraph = connectLayers(lgraph,"relu_17","conv_20");
lgraph = connectLayers(lgraph,"relu_17","addition_9/in2");
lgraph = connectLayers(lgraph,"conv_21","addition_9/in1");
```

Create and initialize the network

```
dlNet  = dlnetwork(lgraph);  % Returns an initialized network; see convolution2dLayer & batchNo
```

Show important properties of the dlNet

```
dlNet.Layers
dlNet.Learnables
dlNet.State
```

See the graphical architecture

```
figure(1),
plot(lgraph),
title("Architecture of DNN")
```

Compute the number of parameters

```
[total, detailes] = find_num_parameters(dlNet);
fprintf('\n The number of learnable parameters: '), disp(total)
```

Training options

REMARK: In this program `os` is set such that `num_of_Train_Images` is a multiple of it. Therefore, "remain size" (`rs`) is zero.

```
global show
```

```
show                    = 0; % see TRsubproblem_solver_OBB

S                       = [];
Y                       = [];

tol                     = 1e-5;
delta                   = 1;
gamma                   = 1;

epoch                   = 10;
lim_m                   = 20;                                 % limited memory
os                      = 50;                                 % overlap size
Nb                      = floor( num_of_Train_Images /os ) - 1; % number of multi-batch
rs                      = mod( num_of_Train_Images, os );     % remain size
or                      = os / (2*os + rs);                   % overlap ratio

skip                    = 0;
k                       = 0;
epoch_k                 = 0;
```

Train the network

```
fprintf("\n====================================================================\n")
fprintf("Start Training...")
fprintf("\n====================================================================\n")


start = tic;
while (1)

    fprintf('\n ===================>>> Iteration k : %d \n', k)

    %-----------------------------> Computations at current (duplex) multi-batch

    if k == 0 % Initial shuffling + initial duplex mini-batch:

        shuffel_index                                   = randperm( num_of_Train_Images );
        [X_set1, Y_set1, X_set2, Y_set2, ~, ~, ~]     = get_Jk(k, Nb, rs, os, num_of_Train_Image
        [grad_set1, loss_set1, acc_set1]              = dlfeval(@model_Forward_Backward, dlNet,
        [grad_set2, loss_set2, acc_set2]              = dlfeval(@model_Forward_Backward, dlNet,


        f_set1              = double(gather(extractdata( loss_set1 )));
        g_set1              = table_2_vec( grad_set1 );


        f_set2              = double(gather(extractdata( loss_set2 )));
        g_set2              = table_2_vec( grad_set2 );


    else % k ~= 0

        if mod(k+1,Nb) ~=0  % Duplex mini-batches within the epoch:
```

```
        [X_set1, Y_set1, X_set2, Y_set2, ~, ~, ~]             = get_Jk(k, Nb, rs, os, num_c
    else % Last duplex mini-batch + new shuffling:

        [X_set1, Y_set1, X_set2, Y_set2, ~, ~, shuffel_index] = get_Jk(k, Nb, rs, os, num_c
        epoch_k                                               = epoch_k + 1;
    end

    [grad_set2, loss_set2, acc_set2]                          = dlfeval(@model_Forward_Back

    f_set2         = double(gather(extractdata( loss_set2 )));
    g_set2         = table_2_vec( grad_set2 );

end

acc            = (acc_set1 + acc_set2)./2;
f              = (f_set1 + f_set2)./2;
g              = (g_set1 + g_set2)./2;
llgll          = norm(g);

%-----------------------------> TR subproblem:
if k == 0 || size(S,2) == 0
    p                  = -delta*(g/llgll); % llpll <= delta
    Bp                 = gamma*p;         % B0*p for Q(p) = p'*g + 1/2 p'*B0*p
else
    p                  = TRsubproblem_solver_OBB(delta, gamma, g, Psi, Minv);
    Bp                 = gamma*p + Psi*(Minv\(Psi'*p));   % Bk*p for Q(p) = p'*g * 1/2 p'*(
end

Q_p                = p'*(g + 0.5*Bp);
llpll              = norm(p);

%-----------------------------> Computation at trial point (See Figure.1):

dlDir            = vec_2_table(dlNet, p);         % a table whose values are the elemnts of
dlNet_trial      = update_dlNet(dlNet, dlDir);    % dlNet.Learnables +  dlDir:  w + p


[grad_new_set1, loss_new_set1]                = dlfeval(@model_Forward_Backward, dlNet_tr
[grad_new_set2, loss_new_set2, acc_new_set2]  = dlfeval(@model_Forward_Backward, dlNet_tr

g_new_set1       = table_2_vec( grad_new_set1 );
g_new_set2       = table_2_vec( grad_new_set2 );
g_new            = (g_new_set1 + g_new_set2)./2;


f_new_set1       = double(gather(extractdata( loss_new_set1 )));
f_new_set2       = double(gather(extractdata( loss_new_set2 )));
f_new            = (f_new_set1 + f_new_set2)./2;

%----------------------------->  Curvature pair (s,y):
s                           = p;
y                           = g_new - g;
```

```
    %----------------------------> Rho
    rho                        = ( f_new - f ) / Q_p;

    %----------------------------> Acceptance Condition:
    if rho > 1e-4

        dlNet       = dlNet_trial;

        g_set1      = g_new_set2;
        f_set1      = f_new_set2;
        acc_set1    = acc_new_set2;
    else
        skip        = skip + 1;

        g_set1      = g_set2;
        f_set1      = f_set2;
        acc_set1    = acc_set2;
    end

    %----------------------------> Evaluate Network:

    dlNet_optimal   = dlNet;
    YPred           = predict(dlNet_optimal, XTest);
    loss_test       = crossentropy(YPred, YTest);
    f_test          = double(gather(extractdata(loss_test)));
    [~,idx_pred]    = max( (extractdata(YPred)), [], 1 );
    [~,idx_true]    = max(YTest, [], 1);
    acc_test        = mean(idx_pred == idx_true)*100;


    %----------------------------> Collect Info:

    Time_(k+1,1) = toc(start);
    F_(k+1, 1) = f;
    Acc_(k+1, 1) = acc;
    F_t(k+1, 1) = f_test;
    Acc_t(k+1, 1) = acc_test;

    %----------------------------> Exit conditions:

    if llgll < tol || acc >= 100 || epoch_k == epoch
        fprintf('\n Training Stopped! \n ')

        return
    end

    %----------------------------> TR Radius:

    if rho > 0.75
        if norm(p)<= 0.8*delta
            delta_new = delta;
        else
            delta_new = 2*delta;
        end
```

```
    elseif (0.1 <= rho && rho <= 0.75)
        delta_new = delta;
    else
        delta_new = 0.5*delta;
    end
    delta  = delta_new;

    %----------------------------->  Bk = gamma*I + psi*M*Psi':

    % -- updating condition:

    sty    = s'*y;

    if sty > 1e-2*llpll^2
        %-- S, Y:
        S = [S, s];
        Y = [Y, y];
        if ( size(S,2) > lim_m )
            S  = S(:, 2:end);
            Y  = Y(:, 2:end);
        end

        if size(S,2) == 0
            warning('S is empty!')
        end

        while ( size(S, 2) > 0 )

            % -- gamma, Minv, Psi:
            SY      = S'*Y;
            SS      = S'*S;

            Lt = tril(SY,-1)';
            LD = tril(SY);
            LDLt    = LD +  Lt;

            eig_val       = eig(LDLt,SS);
            lambdaHat_min = min(eig_val);

            if lambdaHat_min > 0
                gamma = max( 0.5*lambdaHat_min, 1);
            else
                gamma = max( 1, (y'*y)/sty );
            end

            minv{1,1} = -gamma*SS;
            minv{1,2} = -Lt';
            minv{2,1} = -Lt;
            minv{2,2} = diag(diag(SY));

            Minv      = cell2mat(minv);
            Psi       = [gamma*S, Y];
```

```
            if  size(Psi,2) == rank(Psi) && rank(Minv) == size(Minv,2)
                break
            else

                S  = S(:, 2:end);  % Starting removing columns so that Psi and Minv be full c.
                Y  = Y(:, 2:end);
            end
        end % while

    end % if

    k = k + 1;
end % loop

fprintf("\n End of Training!")
fprintf("\n===================================================================\n")
```

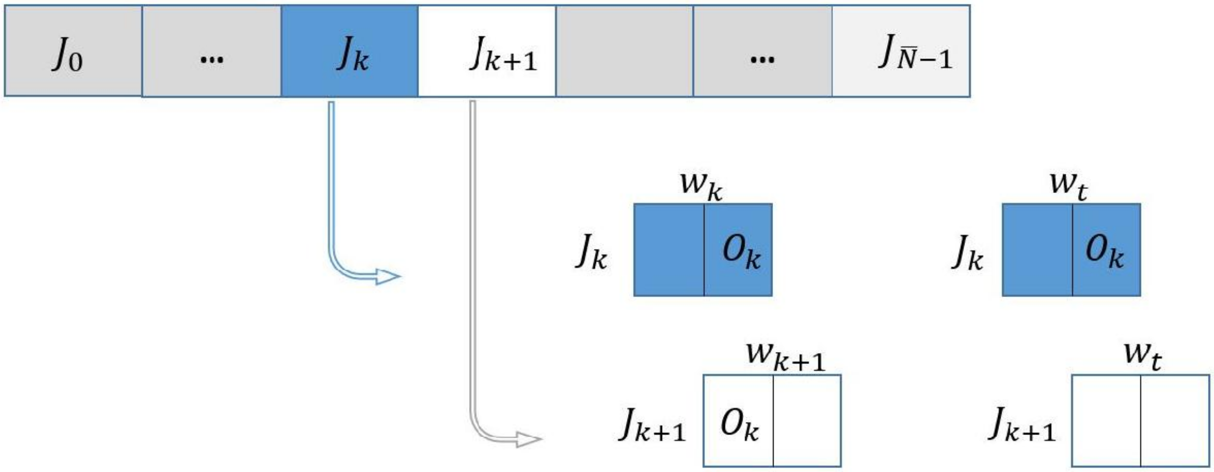
Models

```
function [gradients, loss, acc, state] = model_Forward_Backward(dlNet, dlX, Y)
if nargout == 2
    Yhat              = forward(dlNet, dlX);
    loss              = crossentropy(Yhat, Y);
    gradients         = dlgradient(loss, dlNet.Learnables);
elseif nargout == 3
    Yhat              = forward(dlNet, dlX);
    loss              = crossentropy(Yhat, Y);
    gradients         = dlgradient(loss, dlNet.Learnables);
    acc               = accuracy_fun(Yhat, Y);
elseif nargout == 4
    [Yhat, state]     = forward(dlNet, dlX);
    loss              = crossentropy(Yhat, Y);
    gradients         = dlgradient(loss, dlNet.Learnables);
    acc               = accuracy_fun(Yhat, Y);
end
end


function acc = accuracy_fun(Y_pre, Y)
Y_predict    = extractdata(Y_pre);
[~,idx_pre]  = max(Y_predict,[],1);
[~,idx_true] = max(Y,[],1);
acc          = mean(idx_pre == idx_true)*100;
end
```

Sampling strategy (Batch formation)

Figure.1


J_0
...
J_k
J_{k+1}
...
$J_{\bar{N}-1}$
w_k
J_k
O_k
w_t
J_k
O_k
w_{k+1}
J_{k+1}
O_k
w_t
J_{k+1}