Assignment 08

Table of Contents

- Zhankun Luo:(1)Write the functions, implement Quasi-Newton algorithm
- Wen Ou: (2)Give advice on the improvement
- Andres Jara: (3)Give advice on the improvement

```
clear; clc; close all;
x = csvread('input_data.csv'); d = csvread('output_data.csv');
alpha = -0.5; th = 5e-4;
layer = [10, 5];% nodes of hidden layers
act_type = 'sigmoid';
act_out = true;     % activatetion on output layer
percent_val = 10; seed = 0; % percent of val.: 10%;
num_trial = 10; batch_size = 2; % batch_size accelerate training speed
[list_error_train_qn, list_error_val_qn, ...
 list_error_train_gd, list_error_val_gd] = deal(zeros(1, num_trial));
for seed = 0:num_trial-1 % seed for rng() -> different train, val
 dataset
    train_method = 'qn';
    [list_error_train_qn(seed+1),list_error_val_qn(seed
+1),~,~,~,~,~,~]=...
    run_ann(x,d,alpha,th,layer,act_type,act_out,percent_val,...
            seed,train_method,batch_size);
    train_method = 'gd';
    [list_error_train_gd(seed+1),list_error_val_gd(seed
+1),~,~,~,~,~,~]= ...
    run_ann(x,d,alpha,th,layer,act_type,act_out,percent_val,...
            seed,train_method,batch_size);
end
error_train_qn = mean(list_error_train_qn);
error_val_qn = mean(list_error_val_qn);
error_train_gd = mean(list_error_train_gd);
error_val_gd = mean(list_error_val_gd);
csvwrite('error_train_qn', error_train_qn');
csvwrite('error_val_qn', error_val_qn');
csvwrite('error_train_gd', error_train_gd');
csvwrite('error_val_gd', error_val_gd');
fprintf('error train QN: %.6f%%', 100*error_train_qn);
fprintf('error val   QN: %.6f%%', 100*error_val_qn);
fprintf('error train GD: %.6f%%', 100*error_train_gd);
fprintf('error val   GD: %.6f%%', 100*error_val_gd);
```

```
Trial No.: 1   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.16  3.7e-13
 0.47    7e-14
 0.34  5.5e-14
 0.34  5.4e-13
 0.51    2e-13

   y1       y2
 0.16  3.7e-13
 0.46  1.1e-13
 0.35  8.8e-14
 0.35  4.9e-13
  0.5  2.1e-13
After 2424 epochs
train percent: 90%, MSE: 0.0058%
val   percent: 10%, MSE: 0.0038%


Trial No.: 1   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.16  3.7e-13
 0.47    7e-14
 0.34  5.5e-14
 0.34  5.4e-13
 0.51    2e-13

   y1       y2
 0.16  3.5e-13
 0.47  1.1e-13
 0.34  8.2e-14
 0.36  4.8e-13
 0.51  1.9e-13
After 1726 epochs
train percent: 90%, MSE: 0.0037%
val   percent: 10%, MSE: 0.0019%


Trial No.: 2   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.15  2.6e-12
 0.47    7e-14
 0.27    1e-13
 0.35  2.5e-14
 0.24  2.6e-13

   y1       y2
 0.15  2.2e-12
 0.46  1.1e-13
 0.27  1.1e-13
 0.34  6.7e-14
```

```
 0.23  2.3e-13
After 2376 epochs
train percent: 90%, MSE: 0.0059%
val   percent: 10%, MSE: 0.0038%


Trial No.: 2   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.15  2.6e-12
 0.47    7e-14
 0.27    1e-13
 0.35  2.5e-14
 0.24  2.6e-13

   y1       y2
 0.15  2.3e-12
 0.46  1.2e-13
 0.26  1.2e-13
 0.34  5.4e-14
 0.22  2.7e-13
After 1457 epochs
train percent: 90%, MSE: 0.0070%
val   percent: 10%, MSE: 0.0049%


Trial No.: 3   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.23  6.8e-14
 0.24  2.2e-13
  0.4  7.5e-14
 0.28  8.1e-14
 0.13  4.4e-13

   y1       y2
 0.22  8.6e-14
 0.23    2e-13
  0.4  1.1e-13
 0.28    1e-13
 0.15  4.3e-13
After 2412 epochs
train percent: 90%, MSE: 0.0056%
val   percent: 10%, MSE: 0.0046%


Trial No.: 3   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.23  6.8e-14
 0.24  2.2e-13
  0.4  7.5e-14
 0.28  8.1e-14
 0.13  4.4e-13
```

```
   y1       y2
 0.23  6.8e-14
 0.23  2.4e-13
 0.41  1.1e-13
 0.27  7.8e-14
 0.15  4.8e-13
After 2308 epochs
train percent: 90%, MSE: 0.0044%
val   percent: 10%, MSE: 0.0038%


Trial No.: 4   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.31  2.5e-13
 0.29  9.5e-13
 0.24  1.6e-13
 0.54  6.6e-14
 0.21  6.8e-13

   y1       y2
 0.32  2.1e-13
 0.28  9.7e-13
 0.23  1.7e-13
  0.5  1.1e-13
  0.2  5.6e-13
After 2168 epochs
train percent: 90%, MSE: 0.0065%
val   percent: 10%, MSE: 0.0104%


Trial No.: 4   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.31  2.5e-13
 0.29  9.5e-13
 0.24  1.6e-13
 0.54  6.6e-14
 0.21  6.8e-13

   y1       y2
  0.3  1.8e-13
 0.28  9.6e-13
 0.22  1.5e-13
 0.51  1.2e-13
  0.2  5.1e-13
After 1589 epochs
train percent: 90%, MSE: 0.0070%
val   percent: 10%, MSE: 0.0129%


Trial No.: 5   Train method: qn
First 5 val. samples of d & y
```

```
   d1       d2
 0.25  1.8e-12
 0.34  3.6e-14
 0.31  2.5e-13
 0.24  2.2e-13
 0.16  2.6e-13

   y1       y2
 0.24  1.9e-12
 0.34    8e-14
 0.32  2.1e-13
 0.23  2.1e-13
 0.16  2.6e-13
After 2317 epochs
train percent: 90%, MSE: 0.0051%
val   percent: 10%, MSE: 0.0041%


Trial No.: 5   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.25  1.8e-12
 0.34  3.6e-14
 0.31  2.5e-13
 0.24  2.2e-13
 0.16  2.6e-13

   y1       y2
 0.23  1.7e-12
 0.34  7.6e-14
 0.31  2.3e-13
 0.23  2.2e-13
 0.16  2.6e-13
After 1807 epochs
train percent: 90%, MSE: 0.0033%
val   percent: 10%, MSE: 0.0035%


Trial No.: 6   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.32  1.2e-13
 0.16  3.7e-13
 0.27  1.6e-13
  0.4    6e-14
 0.37  2.9e-13

   y1       y2
 0.33  1.2e-13
 0.16  3.3e-13
 0.27  1.6e-13
  0.4  8.9e-14
 0.38  2.7e-13
After 4246 epochs
```

```
train percent: 90%, MSE: 0.0047%
val   percent: 10%, MSE: 0.0049%


Trial No.: 6   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.32  1.2e-13
 0.16  3.7e-13
 0.27  1.6e-13
  0.4    6e-14
 0.37  2.9e-13

   y1       y2
 0.32  1.3e-13
 0.16  3.1e-13
 0.26  1.7e-13
  0.4  9.3e-14
 0.38  2.7e-13
After 2885 epochs
train percent: 90%, MSE: 0.0043%
val   percent: 10%, MSE: 0.0045%


Trial No.: 7   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.35  2.5e-14
 0.27  1.2e-13
 0.47  5.8e-14
 0.24  1.6e-13
 0.18  2.9e-13

   y1       y2
 0.35  6.7e-14
 0.27  1.2e-13
 0.47    1e-13
 0.23  1.5e-13
 0.18  3.1e-13
After 2372 epochs
train percent: 90%, MSE: 0.0051%
val   percent: 10%, MSE: 0.0030%


Trial No.: 7   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.35  2.5e-14
 0.27  1.2e-13
 0.47  5.8e-14
 0.24  1.6e-13
 0.18  2.9e-13

   y1       y2
```

```
 0.35  4.9e-14
 0.27  1.2e-13
 0.47  9.3e-14
 0.24  1.4e-13
 0.18  2.7e-13
After 1921 epochs
train percent: 90%, MSE: 0.0052%
val   percent: 10%, MSE: 0.0037%


Trial No.: 8   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.44  2.2e-13
 0.23  1.1e-13
 0.24    2e-13
 0.11    2e-13
 0.32  1.2e-13

   y1       y2
 0.45    2e-13
 0.22  1.1e-13
 0.23  1.8e-13
 0.12  1.8e-13
 0.32  1.2e-13
After 4904 epochs
train percent: 90%, MSE: 0.0020%
val   percent: 10%, MSE: 0.0020%


Trial No.: 8   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.44  2.2e-13
 0.23  1.1e-13
 0.24    2e-13
 0.11    2e-13
 0.32  1.2e-13

   y1       y2
 0.46  2.1e-13
 0.23  1.2e-13
 0.24  1.9e-13
 0.13  1.9e-13
 0.33  1.3e-13
After 2097 epochs
train percent: 90%, MSE: 0.0072%
val   percent: 10%, MSE: 0.0061%


Trial No.: 9   Train method: qn
First 5 val. samples of d & y
   d1       d2
 0.34  8.2e-14
```

```
 0.29  5.1e-14
 0.47    7e-14
 0.17  2.8e-13
 0.19  1.3e-12

   y1       y2
 0.33    1e-13
 0.28  7.5e-14
 0.46  1.1e-13
 0.17  2.4e-13
 0.18  1.4e-12
After 3292 epochs
train percent: 90%, MSE: 0.0044%
val   percent: 10%, MSE: 0.0035%


Trial No.: 9   Train method: gd
First 5 val. samples of d & y
   d1       d2
 0.34  8.2e-14
 0.29  5.1e-14
 0.47    7e-14
 0.17  2.8e-13
 0.19  1.3e-12

   y1       y2
 0.33  9.5e-14
 0.29  7.1e-14
 0.47  9.2e-14
 0.16  2.8e-13
 0.18  1.4e-12
After 8205 epochs
train percent: 90%, MSE: 0.0011%
val   percent: 10%, MSE: 0.0014%


Trial No.: 10   Train method: qn
First 5 val. samples of d & y
   d1       d2
  0.1  1.8e-12
  0.2  3.3e-13
 0.18  3.3e-13
 0.48  3.5e-14
 0.17  1.6e-13

   y1       y2
 0.13  1.9e-12
  0.2  3.1e-13
 0.18  3.4e-13
 0.47  7.1e-14
 0.17  1.4e-13
After 2021 epochs
train percent: 90%, MSE: 0.0059%
val   percent: 10%, MSE: 0.0049%
```

```
Trial No.: 10   Train method: gd
First 5 val. samples of d & y
   d1       d2
  0.1  1.8e-12
  0.2  3.3e-13
 0.18  3.3e-13
 0.48  3.5e-14
 0.17  1.6e-13

   y1       y2
 0.13  1.9e-12
 0.19  3.2e-13
 0.18  3.3e-13
 0.47  5.8e-14
 0.17  1.4e-13
After 1377 epochs
train percent: 90%, MSE: 0.0058%
val   percent: 10%, MSE: 0.0042%


error train QN: 0.005100%error val   QN: 0.004492%error train GD:
 0.004902%error val   GD: 0.004695%
```

new functions

```
function [u_new, v_new] = update_gd(u, v, x, d, alpha,
 act_type,act_out,batch_size)
%UPDATE_GD Update Weights with Gradient Descendent
% update_gd(u, v, x, d, alpha, act_type,act_out) updates weights with
 GD
% Inputs:
%   u: weight of the input layer
%   v: weight of the hidden layers
%   x: input matrix
%   d: actual output matrix
%   alpha: negative learning rate
%   act_type: type of activation function: 'sigmoid', 'tanh' or 'relu'
%   act_out: option of activation function on the output layer
%   train_method: gradient descendent: 'gd', quasi-Newton: 'qn'
%   batch_size: batch size of data while training
% Outputs:
%   u_new: weight of the input layer after a poch
%   v_new: weight of the hidden layers after a epoch
len = ceil(length(d) / batch_size);
for ii = 1:len
    start = (ii-1) * batch_size+1;
    if ii == len
        [x_ss, d_ss] = deal(x(start:end, :), d(start:end, :));
    else
        [x_ss, d_ss] = deal(x(start:start+batch_size-1, :), ...
                            d(start:start+batch_size-1, :));
```

```
    end
    [x_ss, d_ss] = deal(x(ii, :), d(ii, :));
    [y_ss, ~, active_z] = calc_ann(x_ss, u, v,act_type,act_out);
    grad_u = get_der_u(v, active_z, x_ss,y_ss,d_ss,act_type,act_out);
    grad_v = get_der_v(v, active_z, y_ss, d_ss,act_type,act_out);
    u = u + alpha * grad_u;
    v = add(v, multiply(alpha, grad_v));
end
u_new = u; v_new = v;
end


function [u_new, v_new] = update_qn(u, v, x, d, alpha,
 act_type,act_out,batch_size)
%UPDATE_GD Update Weights with Gradient Descendent
% update_gd(u, v, x, d, alpha, act_type,act_out) updates weights with
 GD
% Inputs:
%   u: weight of the input layer
%   v: weight of the hidden layers
%   x: input matrix
%   d: actual output matrix
%   alpha: negative learning rate
%   act_type: type of activation function: 'sigmoid', 'tanh' or 'relu'
%   act_out: option of activation function on the output layer
%   train_method: gradient descendent: 'gd', quasi-Newton: 'qn'
%   batch_size: batch size of data while training
% Outputs:
%   u_new: weight of the input layer after a poch
%   v_new: weight of the hidden layers after a epoch
s = []; y =[]; rho = []; n_corrs = 10; % default memory for s, y = 10
function output = calc_ann_simple(x, u, v,act_type,act_out)
[output, ~, ~] = calc_ann(x, u, v,act_type,act_out);
end
len = ceil(length(d) / batch_size);
for ii = 1:len
    start = (ii-1) * batch_size+1;
    if ii == len
        [x_ss, d_ss] = deal(x(start:end, :), d(start:end, :));
    else
        [x_ss, d_ss] = deal(x(start:start+batch_size-1, :), ...
                            d(start:start+batch_size-1, :));
    end
    [y_ss, ~, active_z] = calc_ann(x_ss, u, v,act_type,act_out);
    grad_u = get_der_u(v, active_z, x_ss,y_ss,d_ss,act_type,act_out);
    grad_v = get_der_v(v, active_z, y_ss, d_ss,act_type,act_out);
    func= @(u,
 v)calc_mse(calc_ann_simple(x_ss,u,v,act_type,act_out),d_ss);
    [weight, node_input, node_output, layer] = combine_weights(u, v);
    [grad_weight, ~, ~, ~] = combine_weights(grad_u, grad_v);
    if size(s, 2) == n_corrs
        s(:, 1) = []; y(:, 1) = []; rho(1) = [];
    end
    if ii == 1
```

```
        s = weight;
        y = grad_weight;
        rho = 1 / (weight'* grad_weight);
    else
        s = [s weight - weight_prev];
        y = [y grad_weight - grad_weight_prev];
        rho = [rho 1 / (s(:, end)'* y(:, end))];
    end
    weight_prev = weight;
    grad_weight_prev = grad_weight;
    direction = search_direction(s, y, rho, grad_weight);
    product = grad_weight'* direction;% product: gradient, search
 direction
    [du, dv] = separate_weights(direction,node_input, node_output,
 layer);
    alpha_adapt = line_search(func, u, v, du, dv, -alpha, product);
    alpha_ada = - alpha_adapt;
    u = u + alpha_ada * grad_u;
    v = add(v, multiply(alpha_ada, grad_v));
end
u_new = u; v_new = v;
end

function [weight, node_input, node_output, layer] = combine_weights(u,
 v)
%COMBINE WEIGHTS Combine Weights to a Vector
% combine_weights(u, v) combines u and v to a vector: weight
% Inputs:
%   u: weight of the input layer
%   v: weight of the hidden layers
% Outputs:
%   weight: conbined vector of weights, shape=(:, 1)
%   node_input: dimension of input
%   node_output: dimension of output
%   layer: number of nodes for hidden layers
node_input = size(u, 2)-1;
node_output = size(v{end}, 1);
layer = zeros(1, length(v));
for ii = 1:length(v)
    layer(ii) = size(v{ii}, 2) - 1;
end
len = sum( ([node_input layer]+1).*[layer node_output] );
weight = zeros(len, 1);
len_chunk = (node_input+1)*layer(1);
weight(1: len_chunk) = u(:);
start = len_chunk+1;
for ii = 1:length(layer)-1
    len_chunk = (layer(ii)+1)*layer(ii+1);
    weight(start: start+len_chunk-1) = v{ii}(:);
    start = start + len_chunk;
end
weight(start:end) = v{end}(:);
end
```

```
function [u, v] = separate_weights(weight, node_input, node_output,
 layer)
%SEPARATE WEIGHTS Separate Weight Vector to u, v
% combine_weights(u, v) separates weight vector to u, v
% Intputs:
%   weight: conbined vector of weights, shape=(:, 1)
%   node_input: dimension of input
%   node_output: dimension of output
%   layer: number of nodes for hidden layers
% Outputs:
%   u: weight of the input layer
%   v: weight of the hidden layers
assert(sum(([node_input layer]+1).*[layer
 node_output])==length(weight),...
      'node_input, node_output, layer not match length of weight');
u = zeros(layer(1), node_input+1);
v = cell(1, length(layer));
len_chunk = (node_input+1)*layer(1);
u = reshape(weight(1: len_chunk), size(u));
start = len_chunk+1;
for ii = 1:length(layer)-1
    len_chunk = (layer(ii)+1)*layer(ii+1);
    v{ii} = reshape(weight(start: start+len_chunk-1), ...
            [layer(ii+1), layer(ii)+1]);
    start = start + len_chunk;
end
v{end} = reshape(weight(start: end), ...
         [node_output, layer(end)+1]);
end
```

functions used in Quasi-Newton method

```
function d = search_direction(s, y, rho, grad_weight)
% SEARCH DIRECTION Search Direction of Quasi-Newton: limit BFGS
% search_direction(s, y) find the search direction with history:
 s_k,y_k
% reference: SCG algorithm
% 1. https://github.com/scipy/scipy/blob/master/scipy/optimize/
lbfgsb.py
% 2. https://courses.engr.illinois.edu/ece544na/fa2014/nocedal80.pdf
% Input:
%   s: s(:, k) = s_k = weight_{k+1}-weight_k, shape=(:, n_corrs)
%   y: y(:, k) = y_k = gradient_{k+1}-gradient_k, shape=(:, n_corrs)
%   rho: rho(k) = rho_k = 1 / (s_k'*y_k), shape=(1, n_corrs)
%   grad_weight: gradient at weight_k, shape=(:, 1)
% Output:
%   d: search deirection -Hessian^{-1} * gradient
if isempty(s)
    d = - grad_weight;
    return
end
assert(all(size(s)==size(y)), 'size of s, y not equal');
```

```
n_corrs = size(s, 2);
alpha = zeros(1, n_corrs);
q = grad_weight;
for ii = n_corrs:-1:1
    alpha(ii) = rho(ii) * (s(:,ii)'* q);
 q = q - alpha(ii) * y(:,ii);
end
r = q;
for ii = 1:n_corrs
    beta = rho(ii) * (y(:,ii)'* r);
    r = r + s(:,ii) * (alpha(ii) - beta);
end
d = -r; % r = Hessian^{-1} * gradient
end


function alpha = line_search(func, u, v, du, dv, alpha_0, product, r,
 c)
% LINE SEARCH Line Search with Backtracking Method
% line_search(func, u, v, du, dv, r, c) backtracking line search
% reference: https://sites.math.washington.edu/~burke/crs/408/
lectures/L7-line-search.pdf
% Input:
%   func: function
%   u: current u
%   v: current v
%   du: the search direction for u
%   dv: the search direction for v
%   alpha_0: initial step length (>=0)
%   r: backtrack step between (0,1) usually 1/2
%   c: (0,1) usually 10^{-4}
% Output:
%   alpha: adaptive step length (>=0)
if nargin < 8
    r = 1 / 2;
    c = 1e-4;
end
y = func(u, v);
alpha = alpha_0;
u_new = u + alpha * du;
v_new = add(v, multiply(alpha, dv));
y_new = func(u_new, v_new);
while y_new > y + c * alpha * product % Armijo condition
    alpha = r * alpha;
    u_new = u + alpha * du;
    v_new = add(v, multiply(alpha, dv));
    y_new = func(u_new, v_new);
end
end
```

helper functions: operators of cell array

```
function list_sum = add(list_cell1, list_cell2)
```

```
%Add Cell Arrays: v = v1 + v2
assert(length(list_cell1)==length(list_cell2),'cell size not equal')
list_sum = cell(size(list_cell1));
for ii = 1:length(list_cell1)
    list_sum{ii} = list_cell1{ii} + list_cell2{ii};
end
end

function list_multiply = multiply(scalar, list_cell)
%Multiply Cell Array with a Scalar: v = k * v1
list_multiply = cell(size(list_cell));
for ii = 1:length(list_cell)
    list_multiply{ii} = scalar * list_cell{ii};
end
end
```

Published with MATLAB® R2018a