BITS F312

Neural Networks and Fuzzy Logic

Assignment - 3

Name and ID :

Ravi Bharadwaj C

2016AAPS0244H

Question 1 :

Editor - E:\Studies\3-1\NNFL\assignment3\cnn_2.m

cnn_2.m

```
199 -            z2t(h) = relu(sum(w2(h, :).*z1t) + b2);
200 -        end
201 -        for i = 1:c
202 -            y_p(p, i) = sigmoid(sum(w3(i, :).*z2t) + b3);
203 -            if y_p(p,i) > 0.5
204 -                y_p(p,i) = 1;
205 -            else
206 -                y_p(p,i) = 0;
207 -            end
208 -        end
209 -    end
210 -    [cm,~] = confusionmat(test_y,y_p);
211 -    diagonal = 0;
212 -    for i = 1:size(cm,1)
213 -        diagonal = diagonal + cm(i,i);
214 -    end
215 -    accuracy = diagonal/sum(sum(cm));
216 -    disp('The accuracy is : ');
217 -    disp(accuracy);
218
219
```

Command Window

```
The accuracy is :
    0.5000

fx >>
```

Accuracy achieved was around 0.5 every time it was run.

CODE:

```m
clear;
clc;
close all;

load('data_for_cnn.mat') % ecg_in_window
load('class_label.mat') % label
x = ecg_in_window;
y = label;
dataset = [x y];
% Randomizing and splitting into testing and training set
% 70-30 cross validation
dataset = dataset(randperm(size(dataset,1)),:);
train = dataset(1:700,:);
test = dataset(701:1000,:);
train_x = train(:,1:1000);
train_y = train(:,1001);
tr_y = zeros(length(train_y),2);
test_x= test(:,1:1000);
test_y = test(:,1001);
for i = 1:length(train_y)
    if (train_y(i) == 0)
        tr_y(i,:) = [1,0];
    elseif (train_y(i) == 1)
        tr_y(i,:) = [0,1];
    end
end

[M, N] = size(train_x);
[P, Q] = size(test_x);

% Kernel init
rmin = -0.01;
rmax = 0.01;
% K = rand(1,3);
K = [1/3 1/3 1/3];

% ReLU function definition
relu = @(x) x*(x>=0);

% ReLU function derivative definition :
del_relu = @(x) (x>0);

% Sigmoid function definition
sigmoid = @(x) 1/(1 + exp(-x));

% Parameters init
iter = 5;
b = rand();
conv_op = zeros(M,1);
pool_output = zeros(size(conv_op,1)/2,1);
Nc = N-2;
Np = (N-2)/2;
```

```
% Dense layer parameters
H1 = 10;
H2 = 20;
c = 1;
alpha = 0.5;

% weights init
w1 = rmin + (rmax - rmin)*rand(H1,Np);
w2 = rmin + (rmax - rmin)*rand(H2, H1);
w3 = rmin + (rmax - rmin)*rand(c, H2);
b1 = 1; %rmin + (rmax-rmin)*rand();
b2 = 1; %rmin + (rmax-rmin)*rand();
b3 = 1; %rmin + (rmax-rmin)*rand();

z1 = zeros(1,H1);
z2 = zeros(1,H2);
cost = zeros(iter,1);

% ------Training-------
for k = 1:iter
    for m = 1:M
       cost(k) = cost(k) + (y(m) - train_y(m)).^2;
    end
    cost(k) = 0.5*sqrt(cost(k));
    % Forward Propogation
    for m = 1:M
        f = train_x(m, :);
        conved = zeros([Nc 1]);
        for i = 1:Nc
            conved(i) = relu(K*f(i:i+2)' + b);
        end
        % Average Pooling (downsampled by 2)
        pooled = zeros([Np 1]);
        for i = 1:Np
            pooled(i) = mean(conved(i:i+1));
        end

        % Dense layers
        for h = 1:H1
            z1(h) = relu(w1(h, :)*pooled + b1);
        end
        for h = 1:H2
            z2(h) = relu(sum(w2(h, :).*z1) + b2);
        end
        for i = 1:c
            y(m, i) = sigmoid(sum(w3(i, :).*z2) + b3);
        end
    % --- Back Propogation --- %
        % Update Weights and Biases
        for i = 1:c
            for h = 1:H2
```

```
        for i = 1:c
            for h = 1:H2
                del_w3(i, h) = -alpha*(train_y(m, i)-y(m, i))*y(m, i)*(1-y(m,i))*z2(h);
            end
            del_b3 = -alpha*(train_y(m, i)-y(m, i))*y(m, i)*(1-y(m,i));
        end

        for h2 = 1:H2
            for h1 = 1:H1
               sigma = 0;
               for i = 1:c
                   sigma = sigma + (train_y(m, i)-y(m,i))*w3(i,h2);
               end
               del_w2(h2, h1) = -alpha*sigma*z1(1,h1)*del_relu(z2(h2));
            end
            del_b2 = -alpha*sigma*del_relu(z2(h2));
        end

        del_p = zeros([Np 1]);
        for h1 = 1:H1
            for j = 1:Np
               sigma = 0;
               for h2 = 1:H2
                   for i = 1:c
                       sigma = sigma + (train_y(m,i)-y(m,i))*w3(i,h2)*w2(h2,h1);
                   end
               end
               del_w1(h1,j) = -alpha*sigma*del_relu(z1(h1))*pooled(j);
               del_p(j) = sigma*del_relu(z1(h1));
            end
            del_b1 = -alpha*sigma*del_relu(z1(h1));
        end

        for i = 1:c
            for h = 1:H2
                w3(i,h) = w3(i,h) - del_w3(i,h);
            end
            b3 = b3 - del_b3;
        end

        for h2 = 1:H2
            for h1 = 1:H1
                w2(h2,h1) = w2(h2,h1) - del_w2(h2,h1);
            end
            b2 = b2 - del_b2;
        end

        for h = 1:H1
            for j = 1:Np
                w1(h,j) = w1(h,j) - del_w1(h,j);
            end
            b1 = b1 - del_b1;
        end
```

```
        end


        % Upsampling
        upsampled = zeros([N 1]);
        for i = 1:2:Nc
            upsampled(i:i+1) = pooled((i+1)/2);
        end

        % Update Kernel and Bias
        del_g = 0;
        del_b = 0;
        for i = 1:3
            delta_g = 0;
            delta_b = 0;
            for j = 1:Nc
                delta_g = delta_g + del_relu(K*f(j:j+2)' + b)*upsampled(j:j+2);
                delta_b = delta_b + del_relu(K*f(j:j+2)' + b);
            end
            del_g = del_g + delta_g;
            del_b = del_b + delta_b;
        end
        K = K - alpha*del_g(1);
        b = b - alpha*del_b;
    end
end

% --- Testing --- %
z1t = zeros([1 H1]);
z2t = zeros([1 H2]);
for p = 1:P
    ft = test_x(p, :);
    convedt = zeros([Nc 1]);
    for i = 1:Nc
        convedt(i) = relu(K*ft(i:i+2)');
    end
    pooledt = zeros([Np 1]);
    for i = 1:Np
        pooledt(i) = mean(convedt(i:i+1));
    end
    for h = 1:H1
        z1t(h) = relu(sum(w1(h, :).*pooledt(p, :)) + b1);
    end
    for h = 1:H2
        z2t(h) = relu(sum(w2(h, :).*z1t) + b2);
    end
    for i = 1:c
        y_p(p, i) = sigmoid(sum(w3(i, :).*z2t) + b3);
        if y_p(p,i) > 0.5
            y_p(p,i) = 1;
        else
            y_p(p,i) = 0;
```

```
209    end
210    [cm,~] = confusionmat(test_y,y_p);
211    diagonal = 0;
212    for i = 1:size(cm,1)
213        diagonal = diagonal + cm(i,i);
214    end
215    accuracy = diagonal/sum(sum(cm));
216    disp('The accuracy is : ');
217    disp(accuracy);
218
219
```