MATLAB Deep Learning Notes VI

Key Words : Momentum

1. Momentum

1. In fact, there are various weight adjustment forms available. The simplest adjustment form such as equation (9) and (19) doesn't have higher stability or faster speed. So, we should find some more effective weight adjustment forms. In this notes we introduce **Momentum**. The basic concept of Momentum is (20) as follows

$$\begin{aligned}
&\Delta w = \alpha \delta x \\
&m = \Delta w + \beta m^- \\
&w = w + m \\
&m^- = m
\end{aligned}$$

where m^- is the previous momentum and β is a positive constant that is less than 1.

2. The following steps (21) show how the momentum changes over time.

$$\begin{aligned}
&m(0) = 0 \\
&m(1) = \Delta w(1) + \beta m(0) = \Delta w(1) \\
&m(2) = \Delta w(2) + \beta m(1) = \Delta w(2) + \beta \Delta w(1) \\
&m(3) = \Delta w(3) + \beta m(2) = \Delta w(3) + \beta \Delta w(2) + \beta^2 \Delta w(1) \\
&m(4) = \cdots
\end{aligned}$$

3. It is noticeable from these steps that the previous weight update, i.e., $\Delta w(1)$, $\Delta w(2)$, $\Delta w(3)$, etc., is added to each momentum over the process. Since β is less than 1, the older weight updates exerts a lesser influence on the momentum. Although the influence diminishes over time, the old weight updates remain in the momentum. Therefore, the weight is not solely affected by a particular weight update value, and the learning stability improves accordingly. In addition, since the momentum grows, the weight update becomes greater and greater as well.

Program 12: BackPropMmt

Listing 1: BackPropMmt.m

```matlab
function [weight1, weight2] = BackPropMmt(weight1, weight2, data_input,...
correct_output)

alpha = 0.9;
beta = 0.9;

mmt1 = zeros(size(weight1));
mmt2 = zeros(size(weight2));

N = 4;
for k = 1 : N
    x = data_input(k, :)';
    d = correct_output(k);

    v1 = weight1 * x;
    y1 = sigmoid(v1);
    v = weight2 * y1;
    y = sigmoid(v);

    e = d - y; % calculate the error of the output to the correct output
    delta = y .* (1 - y) .* e;

    e1 = weight2' * delta; % calculate the error of the hidden layer
    delta1 = y1 .* (1 - y1) .* e1;

    % Adjust the weights use momentum
    dw1 = alpha * delta1 * x';
    mmt1 = dw1 + beta * mmt1;
    weight1 = weight1 + mmt1;

    dw2 = alpha * delta * y1';
    mmt2 = dw2 + beta * mmt2;
    weight2 = weight2 + mmt2;
end
```

```
end
```

Program 13: testBackPropMmt

Listing 2: testBackPropMmt.m

```
clear
data_input = [ 0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1]; % training data
correct_output = [0; 1; 1; 0]; % correct outputs(i.e., labels)
weight1 = 2 * rand(4, 3) - 1;
weight2 = 2 * rand(1, 4) - 1;


for epoch = 1:10000 % train
[weight1, weight2] = BackPropMmt(weight1, weight2, data_input,...
correct_output);
end

N = 4; % inference
for k = 1 : N
x = data_input(k, :)';
v1 = weight1 * x;
y1 = sigmoid(v1);
v = weight2 * y1;
y = sigmoid(v)
end
```

Output 13:

```
>> testBackPropMmt

y =

    0.0046

y =

    0.9940

y =

    0.9924

y =
   0.0119
```

2. Comparison between Simple Adjustment and Momentum

4. In this section we compare the average training error performance between Simple Adjustment and Momentum. We use SGD and aim to solve the linear separable problem we talked before.

Program 14: SimpleVsMmt

Listing 3: SimpleVsMmt.m

```
clear

data_input = [ 0, 0, 1; 0, 1, 1; 1, 0, 1; 1, 1, 1 ]; % training data
correct_output = [ 0; 0; 1; 1 ]; % correct outputs
weight11 = 2 * rand(4, 3) - 1;
weight12 = 2 * rand(1, 4) - 1;
weight21 = weight11;
weight22 = weight12;

E1 = zeros(1000, 1);
E2 = E1;

for epoch = 1 : 1000 % the number of epoch = 1000
    [weight11, weight12] =  BackPropXOR(weight11, weight12, data_input,
       correct_output);
    [weight21, weight22] =  BackPropMmt(weight21, weight22, data_input,
       correct_output);

    es1 = 0; es2 = 0;
    N = 4;
    for k = 1 : N
        x = data_input(k, :)';
        d = correct_output(k);
        % Simple(i.e., Mmt)
        v1_simple = weight11 * x;
        y1_simple = sigmoid(v1_simple);
        v_simple = weight12 * y1_simple;
        y_simple = sigmoid(v_simple)
        es1 = es1 + (d - y_simple)^2;
        % Mmt
        v1_mmt = weight21 * x;
        y1_mmt = sigmoid(v1_mmt);
        v_mmt = weight22 * y1_mmt;
        y_mmt = sigmoid(v_mmt)
        es2 = es2 + (d - y_mmt)^2;
    end
    E1(epoch) = es1 / N;
    E2(epoch) = es2 / N;

end

plot(E1, '--b', 'Linewidth',1);
hold on
plot(E2, '-.r', 'Linewidth',1);
xlabel('the number of Epoch')
ylabel('Average of Training error')
legend('Simple', 'Mmt')
```

Output 14:

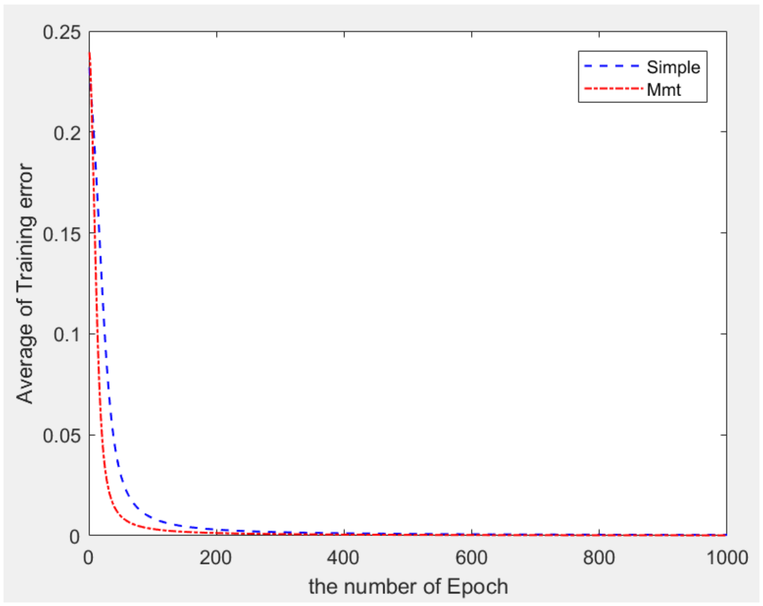

Figure 1: Mmt has better performance!.