MATLAB Deep Learning Notes III

Key Words : Batch; Mini Batch.

1. Batch

1. In the batch method, each weight update is calculated for all errors of the training data, and the average of the weight updates is used for adjusting the weights. This method uses all of the training data and updates only once.

$$\Delta w_{ij} = \frac{1}{N} \sum_{k=1}^{N} \Delta w_{ij}(k) \tag{11}$$

where $\Delta w_{ij}(k) = \alpha \delta_i(k) x_j(k)$ is the weight update for the k-th training data, and N is the total number of the training data.

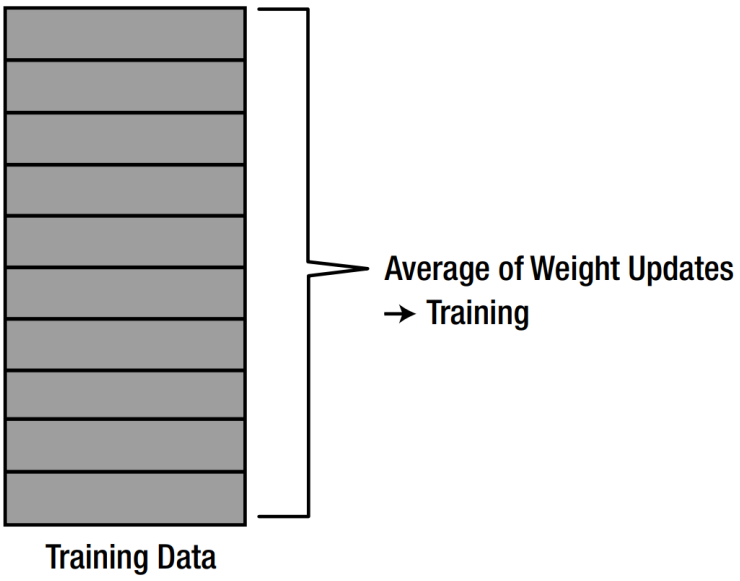

Figure 1: The batch method's weight update calculation and training process

2. Because of the averaged weight update calculation, the batch method consumes a significant amount of time for training

Program 4: DeltaSGD

Listing 1: DeltaBatch.m

```
function [weight] = DeltaBatch(weight, data_input, correct_output)

alpha = 0.9; % learning rate
dwsum = zeros(3, 1); % initialize the sum of weights
N = 4;

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

    v = weight * x; % {1X3} * {3X1}
    y = sigmoid(v);

    e = d - y; % error = correct output - actual output

    delta = y * (1-y) * e; % equation (8)

    dw = alpha * delta * x; % delta rule

% adds the individual weight updates of the entire training data to dwsum
    dwsum = dwsum + dw;

end
dwavg = dwsum / N; % average weight update(i.e., equation (11))

    weight(1) = weight(1) + dwavg(1); % weight updating
    weight(2) = weight(2) + dwavg(2); % weight updating
    weight(3) = weight(3) + dwavg(3); % weight updating
end
```

Output 4:

```
>> DeltaBatch(2 * rand(1, 3) - 1,  [ 0, 0, 1; 0, 1, 1; 1, 0, 1;...
1, 1, 1], [0; 0; 1; 1])

ans =

   -0.4037    0.0803    0.8892
```

Program 5: testDeltaSGD

Listing 2: testDeltaBatch.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(i.e., labels)
weight = 2 * rand(1, 3) - 1; % initializes the weights  with random real
    numbers between [-1, 1]

for epoch = 1 : 40000 % 4 * 10000(the epoch number of testDeltaSGD) = 40000
    weight =  DeltaBatch(weight, data_input, correct_output)
end

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

Output 5:

```
...
weight =

    9.5648   -0.2096   -4.5744

weight =

    9.5649   -0.2096   -4.5744

y =

    0.0102

y =

    0.0083

y =
```

```
    0.9932

y =

    0.9917
```

4. The output is very similar to the correct output

$$\begin{bmatrix} 0.0102 \\ 0.0083 \\ 0.9932 \\ 0.9917 \end{bmatrix} \quad \Leftrightarrow \quad D = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Figure 2: The output is very similar to the correct output

5. An interesting point about this method is that it trained the neural network 40,000 times. Recall that the SGD method performed only 10,000 trainings. This indicates that the batch method requires more time to train the neural network to yield a similar level of accuracy of that of the SGD method. In other words, the batch method learns slowly.

2. Mini Batch

6. The mini batch method is a blend of the SGD and batch methods. It selects a part of the training dataset and uses them for training in the batch method. Therefore, it calculates the weight updates of the selected data and trains the neural network with the averaged weight update.

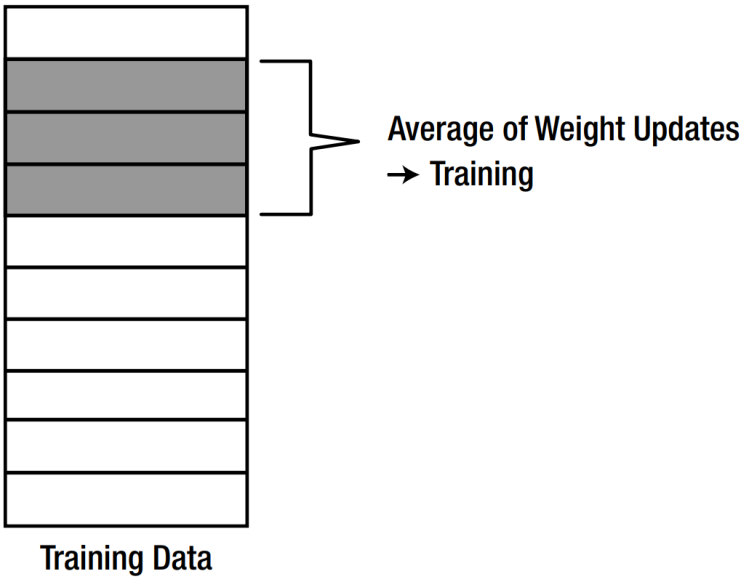

Figure 3: How the mini batch scheme selects training data and calculates the weight update

Program 6: DeltaMiniBatch

Listing 3: DeltaMiniBatch.m

```
function [weight] = DeltaMiniBatch(weight, data_input, correct_output)

alpha = 0.9; % learning rate

% initialize the sum of dalta weight of the first mini batch
dwsum1 = zeros(3, 1);
% initialize the sum of dalta weight of the second mini batch
dwsum2 = zeros(3, 1);
N = 4; % the number of data points in training data
M = 2; % the number of mini batch = 2

for k = 1 : N/M % the first mini batch
    x = data_input(k, :)';
    d = correct_output(k);

```

```
    v = weight * x; % {1X3} * {3X1}
    y = sigmoid(v);

    e = d - y; % error = correct output - actual output

    delta = y * (1-y) * e; % equation (8)

    dw = alpha * delta * x; % delta rule

    dwsum1 = dwsum1 + dw; % the sum of dalta weight of the first mini batch

end
dwavg2 = dwsum1/ (N/M); % average weight update(i.e., equation (11))

    weight(1) = weight(1) + dwavg2(1); % weight updating
    weight(2) = weight(2) + dwavg2(2); % weight updating
    weight(3) = weight(3) + dwavg2(3); % weight updating

for k = (N/M) +1 : N % the second mini batch
    x = data_input(k, :)';
    d = correct_output(k);

    v = weight * x; % {1X3} * {3X1}
    y = sigmoid(v);

    e = d - y; % error = correct output - actual output

    delta = y * (1-y) * e; % equation (8)

    dw = alpha * delta * x; % delta rule

    dwsum2 = dwsum2 + dw; % the sum of dalta weight of the second mini batch

end
dwavg2 = dwsum2 / (N/M); % average weight update(i.e., equation (11))

    weight(1) = weight(1) + dwavg2(1); % weight updating
    weight(2) = weight(2) + dwavg2(2); % weight updating
    weight(3) = weight(3) + dwavg2(3); % weight updating

end
```

Output 6:

```
>> DeltaMiniBatch(2 * rand(1, 3) - 1,  [ 0, 0, 1; 0, 1, 1; 1, 0, 1;...
1, 1, 1], [0; 0; 1; 1])

ans =

    0.6740   -0.5464    0.7740
```

Program 7: testDeltaMiniBatch

Listing 4: testDeltaMiniBatch.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(i.e., labels)
weight = 2 * rand(1, 3) - 1; % initializes the weights with random real
    numbers between [-1, 1]

for epoch = 1 : 20000
% 40000 *1(Batch) = 20000 * 2(Mini Batch) = 10000 * 4(SGD)

    weight =  DeltaMiniBatch(weight, data_input, correct_output)
end

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

Output 7:

```
...
weight =

    9.5651   -0.2090   -4.5749

weight =

    9.5651   -0.2090   -4.5749

y =

    0.0102

y =

    0.0083

y =

    0.9932

y =

    0.9917
```

7. From the programs above we can find that SGD, Batch and Mini Batch could get the same result, but the number of epoch of each scheme is different.