# MATLAB Deep Learning Notes II

**Key Words : Generalized Delta Rule, SGD**

## 1. Generalized Delta Rule

1. For an arbitrary activation function, the delta rule is expressed as the following equation.

$$w_{ij} \leftarrow w_{ij} + \alpha \delta_i x_j \tag{5}$$

2. Note that the difference between (4) and (5) is that $e_i$ in (4) is replaced with $\delta_i$, which defined as

$$\delta_i = \varphi'(v_i)e_i \tag{6}$$

where $v_i$ is the weighted sum of the output node $i$, $\varphi(\cdot)$ is activation function. In (4) we actually use $\varphi(x) = x$, so $\varphi'(x) = 1$, but now we use a more complicated one called *sigmoid function*, as shown in Fig.1

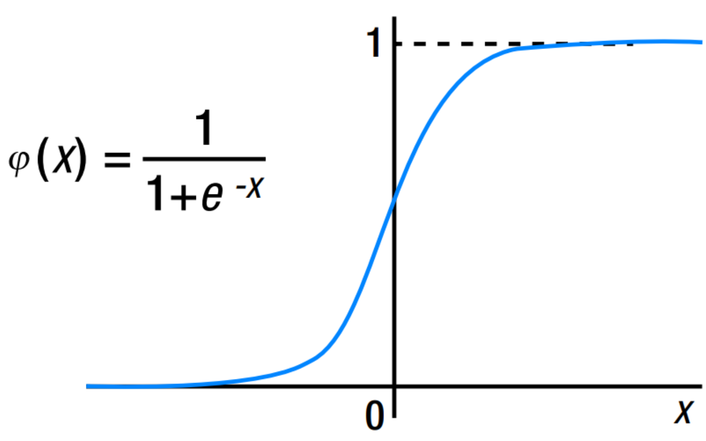


Figure 1: Sigmoid Function $\varphi(x) = 1/(1 + e^{-x})$

**Program 1**: Sigmoid Function $\varphi(x) = 1/(1+e^{-x})$

Listing 1: sigmoid.m

```
function [varphi] = sigmoid(x)

varphi = 1 ./ (1 + exp(-x));

end
```

**Output 1**:

```
>>  x = [-3 : 1 : 3 ]; sigmoid(x)

ans =

    0.0474    0.1192    0.2689    0.5000    0.7311    0.8808    0.9526
```

3. The derivative of sigmoid function is given blew

$$\varphi'(x) = \left(\frac{1}{1+e^{-x}}\right)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{(1+e^{-x})} = \varphi(x)[1-\varphi(x)] \tag{7}$$

so (6) is

$$\delta_i = \varphi(v_i)[1-\varphi(v_i)]e_i \tag{8}$$

then (5) can be rewritten as

$$w_{ij} \leftarrow w_{ij} + \alpha\varphi(v_i)[1-\varphi(v_i)]e_i x_j \tag{9}$$

4. Although the weight update formula is rather complicated, it maintains the identical fundamental concept where the weight is determined in proportion to the output node error, $e_i$ and the input node value, $x_j$.

# 2. Stochastic Gradient Descent

1. The Stochastic Gradient Descent (SGD) calculates the error for each training data and adjust the weights **immediately**. For example, if we have 100 training data points, the SGD adjusts the weights 100 times.

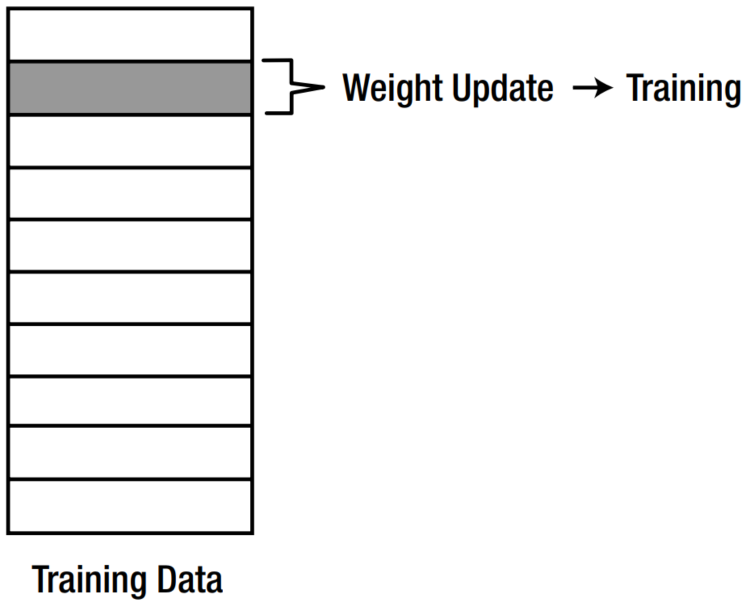


Figure 2: How the weight update of the SGD is related to the entire training data

2. It is easy to obtain how the SGD calculate the weight updates

$$\Delta w_{ij} = \alpha \delta_i x_j \tag{10}$$

3. As the SGD adjusts the weight for each data point, the performance of the neural network is crooked while the undergoing the training process. The name `stochastic` implies the random behavior of the training process.

**Program 2**: DeltaSGD

Listing 2: DeltaSGD.m

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

alpha = 0.9; % learning rate
N = 4;

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

```

    v = weight * x; % {1X3} * {3X1}
    y = sigmoid(v); % use sigmoid function and the output is between [0, 1]

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

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

    dw = alpha * delta * x; % delta rule, equation (10)

    weight(1) = weight(1) + dw(1); % equation (9)
    weight(2) = weight(2) + dw(2); % equation (9)
    weight(3) = weight(3) + dw(3); % equation (9)
end
end
```

**Output 2**:

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

ans =

    0.1163    0.0533    0.2357
```

4. So far we already have sigmoid.m and DeltaSGD.m, then we write a test program test-DeltaSGD.m

**Program 3**: testDeltaSGD

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

for epoch = 1 : 1000
    weight =  DeltaSGD(weight, data_input, correct_output)
end

N = 4;  % inference
for k = 1:N
```

```
    x = data_input(k, :)';
    v = weight * x;
    y = sigmoid(v)
end
```

**Output 3**:

```
...
weight =

    7.1473   -0.2259   -3.3521

weight =

    7.1484   -0.2258   -3.3526

weight =

    7.1495   -0.2258   -3.3532

y =

    0.0338

y =

    0.0271

y =

    0.9780

y =

    0.9726
```

5. These output values are very close to the correct outputs. Therefore, we can conclude that the neural network has been properly trained.

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

Figure 3: Output values are very close to the correct outputs