MATLAB Deep Learning Notes IX

Key Words : Deep Learning; ReLU; Dropout;

1. Basic Concept of Deep Learning

1. Deep Learning is a Machine Learning technique that employs the deep neural network, and the deep neural network is the multi-layer neural network that contains two or more hidden layers. The reason that the neural network with deeper layers yielded poorer performance was that the network was not properly trained. The back-propagation algorithm experiences the following three primary difficulties in the training process of the deep neural network, namely, **Vanishing Gradient**, **Overfitting** and **Computational Load**.

2. Vanishing Gradient

2. The gradient in neural network can be thought as a similar concept to the **delta** of the back-propagation algorithm. The *vanishing gradient* in the training process with back-propagation algorithm occurs when the output error is more likely to fail to reach the father nodes. In other words, if the error not reaches the first hidden layer, the weight is not be well adjusted then. There is no point of adding hidden layers if they cannot be trained.

3. One of solutions is using the **Re**ctified **L**inear **U**nit (**ReLU**) function as the activation function. It is known to better transmit the error than the sigmoid function. The ReLU function is defined as follows:

$$\varphi(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{28}$$

4. So the derivative of the ReLU function is

$$\varphi'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{29}$$

By the way, (29) in MATLAB can be implemented as *varphi > 0*

Program 22: ReLU

Listing 1: ReLU.m

```
function [y] = ReLU(x)
y = max(0, x);
end
```

Output 22:

```
>>  ReLU([-2, 2])

ans =

     0     2
```

3. Overfitting

5. The reason that the deep neural network is especially vulnerable to overfitting is that the model becomes more complicated as it includes more hidden layers, and hence more weight. The most representative solution is the **dropout**, which trains only some of the randomly selected nodes rather than the entire network. Other solutions include **adding regularization** and using **massive training data**. Let's focus on dropout.

Program 23: dropout

Listing 2: dropout.m

```
function [ym] = dropout(y, ratio)
[m, n] = size(y);
ym = zeros(m, n);

num = round(m * n * (1 - ratio)) % round([0.49, 0.5, 0.51]) =  0 1 1
idx = randperm(m * n, num) % randperm(10, 5) =  3 2 10 9 5
ym(idx) = 1 / (1 - ratio);

end
```

Output 23:

```
>> y1 = rand(6, 1)
   ym = dropout(y1, 0.5)
   y1 = y1 .* ym

y1 =
```

```
    0.5369
    0.1723
    0.8799
    0.2481
    0.4845
    0.9334

num =

     3

idx =

     3     4     1

ym =

     2
     0
     2
     2
     0
     0

y1 =

    1.0738
         0
    1.7599
    0.4961
         0
         0
```

6. The reason that we multiply the other element by $1/(1-ratio)$ is to compensate for the loss of output due to the dropped elements. In the previous example, once half of the vector $y1$ has been dropped out, the magnitude of the layer's output significantly diminishes. Therefore, the outputs of the survived nodes are amplified by the proper proportion.

4. Computational Load

7. The number of weights increases geometrically with the number of hidden layers, thus requiring more training data. This ultimately requires more calculations to be made. The more computations the neural network performs, the longer the training takes. The solution of this challenge is **using high-performance hardware** such as GPU, and some **algorithms** like batch normalization.