1.5em 0pt

Chapter 2

Adam p19@ipm.edu.mo
20200429

Keywords: *Stochastic Gradient Descent; SGD; Batch; Mini Batch*

The neural network node consists of three parts that are input node, output node, activate function. A variety of neural networks can be created depending on how the nodes are connected. The structure of neural network has three layers which are input layer, hidden layer and output layer.(Phil Kim, 2017)

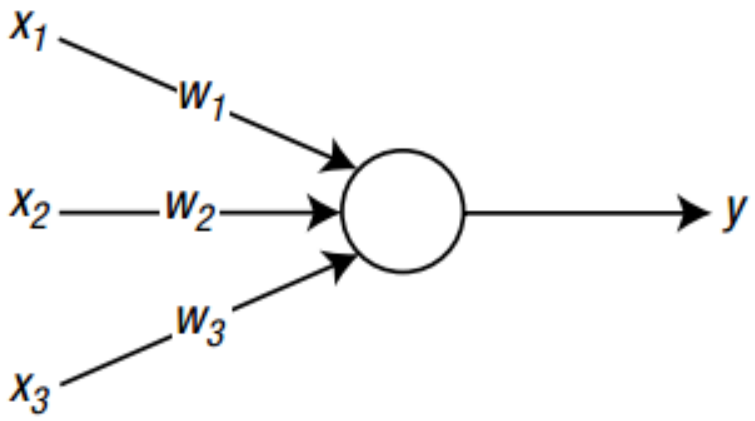

Figure 1: Neural network that consists of three input nodes and one output node

1. Stochastic Gradient Descent

In this book, we just talk about the supervised learning. The supervised learning of the neural network proceeds in the following steps: article

Algorithm 1 Supervised learning of the neural network proceeds

1: Initialize the weights with adequate values.
2: Take the "input" from the training data, which is formatted as (input, correct output), and enter it into the neural network. Obtain the output from the neural network and calculate the error from the correct output.
3: Adjust the weights to reduce the error.
4: Repeat Steps 2-3 for all training data.

Let's train the neural network which can regards as a step-by-step process, as follows

$$\begin{aligned} \delta_i &= \varphi(v_i)(1-\varphi(v_i))e_i \\ \Delta w_{ij} &= \alpha \delta_i x_j \\ w_y &\leftarrow w_{ij} + \Delta w_{y_j} \end{aligned} \tag{1}$$

You can visit the website to search for the code (https://github.com/weizhang1993/MATLAB-Deep-Learning-Notes) . This summary just show the compare of this three method code.

```
correct_output = [0; 0; 1; 1]; % correct outputs(i.e., labels)
EPOCH1 = 1000;
E1 = zeros(EPOCH1,1);
E2 = zeros(EPOCH1,1);
E3 = zeros(EPOCH1,1);
weight1 = 2 * rand(1, 3) - 1; % initializes the weights  with random real
    numbers between [-1, 1]
weight2 = weight1;
weight = weight1;
%% catulate the weight
for epoch = 1 : 1000
    weight1 =  DeltaSGD(weight1, data_input, correct_output);
    weight2 =  DeltaBatch(weight2, data_input, correct_output);
    weight =  DeltaMiniBatch(weight, data_input, correct_output);
    es1 = 0;      es2 = 0;     es3 = 0;
   % inference
    N = 4;
for k = 1:N
    x = data_input(k, :)';
    d = correct_output(k);
    %SGD
    v1 = weight1 * x;
    y1 = sigmoid(v1);
    es1 = es1 + (d - y1)^2;
    %Batch
    v2 = weight2 * x;
    y2 = sigmoid(v2);
    es2 = es2 + (d - y2)^2;
    %MinBatch
    v3 = weight * x;
    y3 = sigmoid(v3);
    es3 = es3 + (d - y3)^2;
end
    E1(epoch) = es1 / N;
    E2(epoch) = es2 / N;
    E3(epoch) = es3 / N;
end
plot(E1,'r')
hold on
plot(E2,'b')
hold on
plot(E3,'y')
xlabel('Epoch')
ylabel('Average of Training error')
legend('SGD','Batch','MiniBatch')
```

References

Phil Phil Kim, Kim. *MATLAB Deep Learning.* Apress, 2017.