**DEEP LEARNING**

Deep learning is an artificial intelligence approach that imitates human brain to process data. Also, deep learning is a subset of machine learning method based on artificial neural network. By means of learning deeply, algorithms can extract higher level features from the dataset and it does this with its hidden layers. Learning can be supervised, unsupervised or semi-supervised. Deep learning is used in solution of many problems in the machine learning field. Also, many different methods have been tried to develop algorithms. These learning methods such as deep neural networks, deep belief networks, recurrent neural networks, long-short term memory (LSTM) and convolutional neural networks are performed in computer vision, speech recognition and more fields. To mention briefly what these terms are:

Multi-layer neural network (MLP) consists at least one input layer, one hidden layer and one output layer. The part that provides deep learning is the hidden layer. In this context, MLP is the basis for deep learning.

Convolutional Neural Network (CNN) is most commonly used in computer vision. It generally uses images or videos by nature. It does the learning by examining the input in depth.

Recurrent Neural Network (RNN) is another deep learning method that is commonly used in text and speech recognition field. It differs from multi-layer perceptron with feedback network term. By means of not only feedforward network but also its feedback network, it can use the past response of the hidden or output layer as an input.

Consequently, learning the MLP that underlies all of them means understanding the deep learning logic. Therefore, in the second part, an example of deep learning with MLP will be explained.

**EXAMPLE: SIMPLE DEEP NEURAL NETWORK**

In this part, multi-layer neural network with backpropagation will be examined as a deep learning approach. Also, pseudo-code, block diagram and results will be shown with figures. As known single layer neural network approach solve only linear problems but real-life problems are generally non-linear. Representing non-linear problem, XOR gate problem can be solved using deep neural network approach. The truth table of XOR gate is shown in Table 1.



Table 1: XOR gate truth table.

Additionally, it is shown in Figure 1 that the XOR problem is not linearly separable, so it is a non-linear problem. As shown in Figure 1, 0 and 1 values are not linearly separable. In order to separate 0 and 1 classes, the parabolic approach must be used.



Figure 1: XOR gate problem. (linearly inseparable)

In this example, in order to solve the XOR gate problem, simple multi-layer neural network approach was proposed. Since it is not a complex problem, one input layer, two hidden layer and one output layer will be used. In hidden layers, two neurons were added. The block diagram of the MLP is shown in the following figure.



Figure 2: Block diagram of deep neural network

The steps of multi-layer network will be explained as follows:

1. Input dataset is prepared as shown in Table 1.
2. Label the desired output or assign a class for each observation
3. Specify the number of layers and neurons.
4. Specify the activation function for each neuron.
5. Start the training using observations
6. Calculate the error for output layer using gradient descent
7. Backpropagate the error for hidden layer and calculate the error
8. Repeat the step 7 for each node after each observation.
9. Update the unknown parameters
10. Store the terminal estimated output, error and weights
11. Plot the error history and estimated output with respect to iteration

Pseudo-code of the multi-layer neural network is shown as follows:

**Pseudo Code: Delta Learning Rule and Backpropagation for  Inputs and  Outputs NNs**

1. **Input: Training input** **, where**  **is the length of the each**  **number of the input data, labelled output** **, where**  **is the number of the output, randomly initialized unknown parameter matrix/vector** **, estimated output** **, the training error**  **, the parameter learning rate** , the number of the iteration **, store error** **, store estimated output** **.**
2. **Output:** The terminal value of the trained parameters  store learning error  , store output 
3. for   to 
4. for  to 
5. 1. Calculate the estimated current output 
6. 
7. 2. Apply a threshold  to the output (if necessary) and repeat this
8. part for other neurons in hidden layer
9. 
10. 3. Determine the error
11. 
12. 4. Calculate the gradient of neurons using delta rule
13. and backpropagate the error
14. 
15. 5. Update and store the unknown parameter
16. 
17. end 
18. Store the error and the output
19. 
20. 
21. end  

The corresponding code proceeds as follows:

Firstly, training parameters are defined as following codes using trainParameters function:

function trainPar = trainParameters()

trainPar.x = [1 0 0 ; 1 0 1 ; 1 1 0 ; 1 1 1]; % Training input data where the first value of 1 represents the bias term

trainPar.y = [0 ; 1 ; 1 ; 0]; % Labelled / Desired output data

trainPar.noi = size(trainPar.x , 1); % Number of input

trainPar.w = rand(3 , 3); % Unknown / Weight parameters which randomly initialized

trainPar.y\_hat = zeros(size(trainPar.x , 1) , 1); % Initialize estimated output

trainPar.mu = 0.2; % Initialize the learning rate

trainPar.it = 1000000; % Initialize the iteration number

trainPar.e = zeros(size(trainPar.x , 1) , 1); % Initialize the instant error

trainPar.es = zeros(size(trainPar.x , 1) , trainPar.it);

% Initialize the estimated error

trainPar.ys\_hat = zeros(size(trainPar.x , 1) , trainPar.it);

% Initialize the estimated output

end

Secondly, the sigmoid activation function is automatically run in sigmoid function block as follows:

function y\_sigmoid = sigmoid(activation)

y\_sigmoid = 1 / (1 + exp(-activation));

% sigmoid activation function

end

Thirdly, the main program is used for learning process. These codes includes the multi-layer neural network approach steps. Delta rule with backpropagation is used to train the networks in this code block.

% This m.file trains a multi layer NN (MLP) for the XOR problem

clear , close all;

clc;

trainPar = trainParameters(); % Upload the training parameters

w = trainPar.w; % Upload the unknown parameters

mu = trainPar.mu; % Upload the learning coefficient

y\_hat = trainPar.y\_hat; % Upload the output

es = trainPar.es; % Upload the allocated estimated error

e = trainPar.e; % Upload the instant error

ys\_hat = trainPar.ys\_hat; % Upload the allocated estimated output

for i = 1 : trainPar.it

for j = 1 : trainPar.noi

% Calculate the output of the first hidden layer neuron

H1 = w(1,:) \* trainPar.x(j,:)';

% Apply a threshold for estimated output of first hidden layer neuron

X1(2) = sigmoid(H1);

% Calculate the output of the second hidden layer neuron

H2 = w(2,:)\* trainPar.x(j,:)';

% Apply a threshold for estimated output of second hidden layer neuron

X1(3) = sigmoid(H2);

% Assign a bias term for output layer

X1(1) = 1;

H2\_1 = w(3,:) \* X1';

X2(2) = sigmoid(H2\_1);

H2\_2 = w(4,:) \* X1';

X2(3) = sigmoid(H2\_2);

X2(1) = 1;

% Calculate the output of the output layer neuron

X3 = w(5,:) \* X2';

% Apply a threshold for output layer neuron

y\_hat(j) = sigmoid(X3);

% Calculate the error of the output and store it

e(j,:) = trainPar.y(j) - y\_hat(j);

% Calculate the error for each neuron using delta rule

delta3\_1 = y\_hat(j) \* (1 - y\_hat(j)) \* (trainPar.y(j) - y\_hat(j));

delta2\_1 = X2(2) \* (1 - X2(2)) \* w(5,2) \* delta3\_1;

delta2\_2 = X2(3) \* (1 - X2(3)) \* w(5,3) \* delta3\_1;

delta1\_1 = X1(2) \* (1 - X1(2)) \* w(3,2) \* delta2\_1;

delta1\_2 = X1(2) \* (1 - X1(2)) \* w(4,2) \* delta2\_2;

delta1\_3 = X1(3) \* (1 - X1(3)) \* w(3,3) \* delta2\_1;

delta1\_4 = X1(3) \* (1 - X1(3)) \* w(4,3) \* delta2\_2;

% Update the unknown parameters

w(1,:) = w(1,:) + mu \* trainPar.x(j,:) \* (delta1\_1 + delta1\_2);

w(2,:) = w(2,:) + mu \* trainPar.x(j,:) \* (delta1\_3 + delta1\_4);

w(3,:) = w(3,:) + mu \* X1 \* delta2\_1;

w(4,:) = w(4,:) + mu \* X1 \* delta2\_2;

w(5,:) = w(5,:) + mu \* X2 \* delta3\_1;

end

% Store the estimated output

ys\_hat(:,i) = y\_hat;

% Store the error history

es(:,i) = e;

end

% Plot the estimated output for the XOR gate

figure(1),plot(1:length(ys\_hat),ys\_hat(1,:),'r','LineWidth',2),hold on,

plot(1:length(ys\_hat),ys\_hat(2,:),'b','LineWidth',2),

plot(1:length(ys\_hat),ys\_hat(3,:),'g','LineWidth',2),

plot(1:length(ys\_hat),ys\_hat(4,:),'y','LineWidth',2), hold off;

xlabel('iteration'),ylabel('output');

title('Estimated Outputs for the XOR Gate');

% Plot the error history for the XOR gate

figure(),plot(1:length(es),es(1,:),'r','LineWidth',2),hold on,

plot(1:length(es),es(2,:),'b','LineWidth',2),

plot(1:length(es),es(3,:),'g','LineWidth',2),

plot(1:length(es),es(4,:),'y','LineWidth',2), hold off;

xlabel('iteration'),ylabel('error');

title('Training Error for the XOR Gate');

Lastly, after training process, the estimated outputs and error with respect to iteration are plotted as stated in following code block.

% Plot the estimated output for the XOR gate

figure(1),plot(1:length(ys\_hat),ys\_hat(1,:),'r','LineWidth',2),hold on,

plot(1:length(ys\_hat),ys\_hat(2,:),'b','LineWidth',2),

plot(1:length(ys\_hat),ys\_hat(3,:),'g','LineWidth',2),

plot(1:length(ys\_hat),ys\_hat(4,:),'y','LineWidth',2), hold off;

xlabel('iteration'),ylabel('output');

title('Estimated Outputs for the XOR Gate');

% Plot the error history for the XOR gate

figure(),plot(1:length(es),es(1,:),'r','LineWidth',2),hold on,

plot(1:length(es),es(2,:),'b','LineWidth',2),

plot(1:length(es),es(3,:),'g','LineWidth',2),

plot(1:length(es),es(4,:),'y','LineWidth',2), hold off;

xlabel('iteration'),ylabel('error');

title('Training Error for the XOR Gate');

Consequently, the error history and estimated output are as shown in Figure 3 and Figure 4.



Figure 3: The error history with respect to iteration.



Figure 4: The estimated output with respect to iteration.

As shown in Figure 3 and 4, output values converge the true values which shown in truth table in Table 1. Also, error term converges to 0. That means XOR problem can solve with this algorithm. In addition, the output values shown in the following table:



Table 2: Estimated output after training and its classes.