# Neural Networks: Error Back Propagation

William L. and Andrea T.
Data Mining and Machine Learning

## Structure

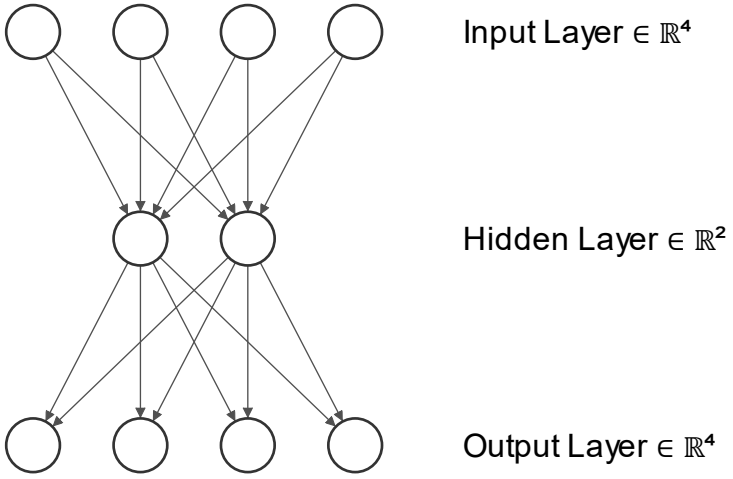



***Fig. 1.*** *MLP structure for 4-2-4 encoder*

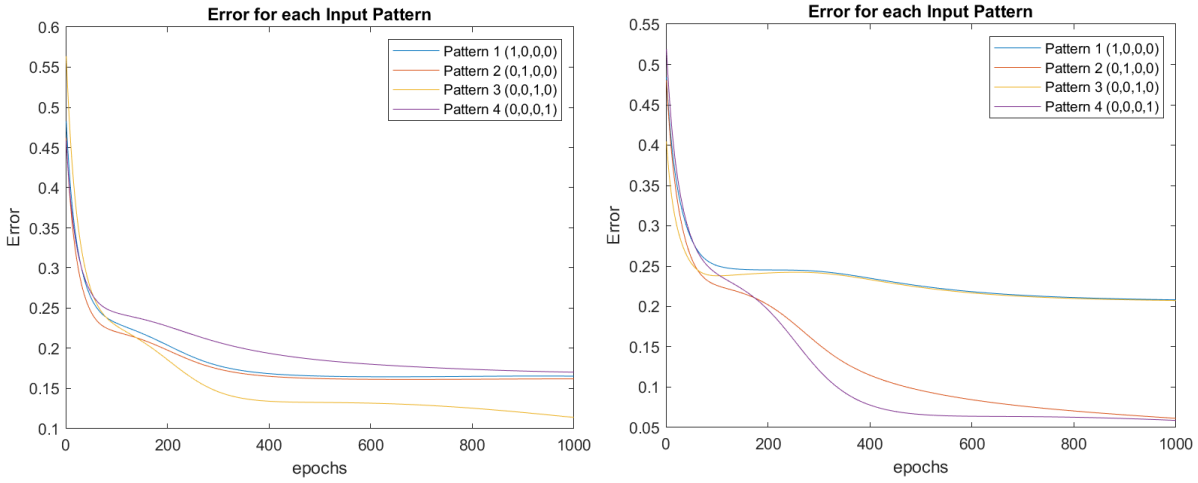
| Input Pattern | Output Pattern |
|---|---|
| 1,0,0,0 | 1,0,0,0 |
| 0,1,0,0 | 0,1,0,0 |
| 0,0,1,0 | 0,0,1,0 |
| 0,0,0,1 | 0,0,0,1 |

***Table 1****: Input-output pairs for 4-2-4 encoder*

A multi-layer perceptron (MLP) has the structure of a 4-2-4 encoder as shown in Fig. 1. It has four input units, two hidden layer units and four output units, with a sigmoid activation function for each unit. The encoder should map the inputs into their corresponding outputs and going into the input units are four different patterns, equal to the target outputs, as shown in Table 1. The training set of the error back propagation algorithm (EBP) consists of the previously mentioned four input-output pairs. But because of the sigmoid activation function in each unit, the values of 0 and 1 will never be realized so for practical purposes the values of 1 and 0 will be replaced by 0.9 and 0.1, respectively.

The training algorithm and the structure of the encoder was coded using MATLAB, iterated for 1000 times, cycling through the four input patterns while accumulating the weights so that the weights are updated only after all four input patterns have gone through a single iteration.

# Results and Discussion




*(a)* *(b)*

***Fig. 2.*** *Error for each of the four input patterns after 1000 iterations of the EBP algorithm for 2 different initial random weights (a) and (b)*

```
out_out =

  4×4 dlarray

    0.3075    0.2909    0.2924    0.3618
    0.1563    0.7462    0.2419    0.0227
    0.3022    0.3929    0.3107    0.2667
    0.2280    0.0231    0.1303    0.7807
```

*(a)*

```
final_hid_out =

    0.6178    0.1212    0.6291    0.9999
    0.5638    0.9998    0.5608    0.0062
```

*(b)*

***Fig. 3.*** *Final result for all input patterns corresponding to Fig. 2b. (a) Final output. (b) Hidden layer output*

Fig. 2 shows that having different initial weight values affect the error and output for different input patterns. Every time the MATLAB script was executed, new weights were generated, leading to vastly different results. From the current algorithm and structure of the MLP, using the code presented in Appendix 1, it was concluded that only a maximum of two input patterns can be roughly mapped to the desired outputs. This is evident as seen in Fig. 3a, where input patterns 2 and 4 are closely mapped to the desired results, at 0.7462 and 0.7807 respectively, while the other two are widely imprecise. This is not always the case as seen in Fig. 2a, where sometimes only one input pattern was mapped correctly, with the other three being no where near the desired results. Other times, different combinations of two input patterns were approximately mapped. Despite what random values the program may choose for the weight matrices, the error cannot reach zero unless a bias is implemented in the input and hidden layer.

The 4-2-4 bottleneck encoder was supposed to learn binary encoding as it is iterated, but this algorithm is struggling to carry out this learning effectively, as seen in Fig. 3b. Each corresponding input-output pair should have their own binary code (00, 01, 10, 11) so that the inputs could be correctly mapped to the output. The hidden layers are not able to learn binary coding adequately as the activation

function is being activated either too soon or too late, so not all inputs are being mapped correctly to their target outputs. To combat this fault with the activation function, bias is introduced to the first two layers, as shown in Fig. 4. The code for the algorithm with added bias is presented in Appendix 2.

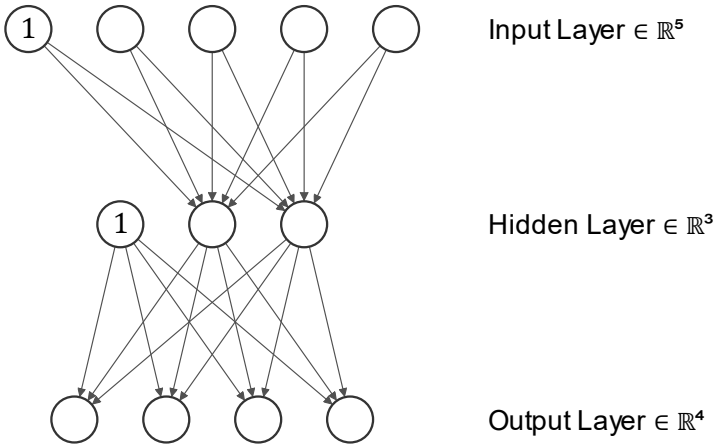
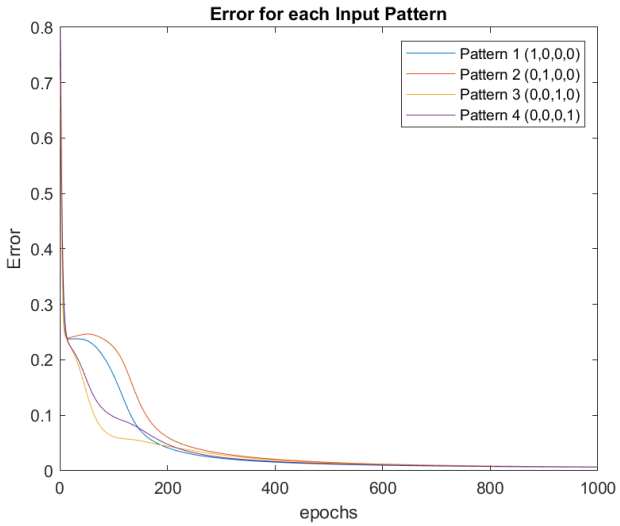



***Fig. 4.*** *Final MLP structure with bias (nodes with 1)*




*(a)*

```
out_out =

  4×4 dlarray

    0.8516    0.0086    0.1374    0.1379
    0.0017    0.8451    0.1312    0.1255
    0.1216    0.1384    0.8535    0.0032
    0.1261    0.1317    0.0045    0.8556
```

*(b)*

```
final_hid_out =

    0.9225    0.0000    0.0000    0.8508
    0.8346    0.0000    0.7299    0.0000
```

*(c)*

***Fig. 5.*** *Results after adding bias to the MLP structure. (a) Error value for each input patterns after 1000 iterations. (b) Final result after inputting all input patterns. (c) Hidden layer output*

Introducing bias provides the results in Fig. 5, decreasing the error nearly to zero for all input patterns. With bias, the activation function was shifted either left or right depending on what the algorithm learns when its propagating the inputs through the MLP. Comparing the results from Fig. 3a to Fig. 5b, it can be seen that all the final outputs in Fig. 5b are closer to the target output values of 0.9 and 0.1. The results confirm that bias was critical for successful learning, and does help the algorithm

in mapping the inputs correctly to the outputs. To further confirm this, we can take a look at the output of the hidden layer shown in Fig. 5c, which is outputting a binary code – 11, 00, 01, 10 – for each corresponding input-output pair. By adding the bias and iterating its weight, the hidden layer was able to learn binary encoding to map the inputs to the outputs correctly, signifying that bias is important for neural networks.

# Appendix

1. MATLAB Code: EBP without bias

```
clc
clear
%Global values
Error = {}  ;
lr    = 0.5 ;

%Initializing Inputs and Weight Matrix's
input = [0.9, 0.1, 0.1, 0.1; 0.1, 0.9, 0.1, 0.1  ;
         0.1, 0.1, 0.9, 0.1; 0.1, 0.1, 0.1, 0.9];
target = input;
W_1 = dlarray(rand(4,2));
W_2 = dlarray(rand(2,4));
Delta_W1 = zeros(4,2)   ;
Delta_W2 = zeros(2,4)   ;

%Start Iteration
for epoch = 1:1:1000
    %Using different input patterns
    for i = 1:1:4
        %Input-Hidden Layer
        y_in   = dlarray(W_1'*input(:,i));
        y_out  = sigmoid(y_in)           ;
        %Hidden-Output Layer
        o_in   = W_2'*y_out              ;
        o_out = sigmoid(o_in)            ;

        %EBP
        %Output Layer
        deltaj_out   = (o_out.*(o_out - target(:,i))).*(1-o_out)        ;
        Delta_W2     = Delta_W2 + (-lr * (y_out * deltaj_out'))         ;
        %Hidden Layer
        deltaj_hid   = W_2*deltaj_out                                   ;
        Delta_W1     = Delta_W1 + (-lr * (input(:,i) * deltaj_hid'))    ;

        %Appending Error to a cell
        Error{epoch, i} = extractdata((1/2)*sum((o_out-target(:,i)).^2));
    end
    %Updating weights after doing each input pattern
    W_1 = W_1 + Delta_W1./4;
    W_2 = W_2 + Delta_W2./4;
    Delta_W1 = 0           ;
    Delta_W2 = 0           ;
end

%Testing if output is correct
for i = 1:1:4
    hid_in  = W_1'*input(:,i)     ;
    hid_out = sigmoid(hid_in)     ;
    out_in  = W_2'*hid_out        ;
    out_out(:,i) = sigmoid(out_in);
end
%Output to command window
out_out

%Plotting Error vs Epoch
Error = cell2mat(Error);
plot(1:1000, Error)
xlabel('epochs')
ylabel('Error')
title('Error for each Input Pattern')
legend({'Pattern 1 (1,0,0,0)','Pattern 2 (0,1,0,0)',...
    'Pattern 3 (0,0,1,0)', 'Pattern 4 (0,0,0,1)'},'Location','northeast')
```

**2.** MATLAB Code: EBP with bias

```
clc
clear
%Global values
Error = {}  ;
lr    = 0.5 ;
%Initializing Inputs and Weight Matrix's + Bias
input = [0.9, 0.1, 0.1, 0.1; 0.1, 0.9, 0.1, 0.1  ;
          0.1, 0.1, 0.9, 0.1; 0.1, 0.1, 0.1, 0.9 ;
          1.0, 1.0, 1.0, 1.0];
target = input;
W_1 = dlarray(rand(5,2));
W_2 = dlarray(rand(3,4));
Delta_W1 = zeros(5,2)   ;
Delta_W2 = zeros(3,4)   ;

%Start Iteration
for epoch = 1:1:1000
    %Using different input patterns
    for i = 1:1:4
        %Input-Hidden Layer
        y_in   = dlarray(W_1'*input(:,i));
        y_out  = sigmoid(y_in)           ;
        y_out(3,1) = 1.0                 ; % bias at hidden layer
        %Hidden-Output Layer
        o_in   = W_2'*y_out              ;
        o_out = sigmoid(o_in)            ;

        %EBP
        %Output Layer
        deltaj_out   = (o_out.*(o_out - target((1:4),i))).*(1-o_out)     ;
        Delta_W2     = Delta_W2 + (-lr * (y_out * deltaj_out'))          ;
        %Hidden Layer
        deltaj_hid   = W_2*deltaj_out                                    ;
        Delta_W1     = Delta_W1 + (-lr * (input(:,i)) * deltaj_hid(1:2)');

        %Appending Error to a cell
        Error{epoch, i} = extractdata((1/2)*sum((o_out-target((1:4),i)).^2));
    end
    %Updating weights after doing each input pattern
    W_1 = W_1 + Delta_W1   ;
    W_2 = W_2 + Delta_W2   ;
    Delta_W1 = 0           ;
    Delta_W2 = 0           ;
end

%Testing if output is correct
final_hid_out = zeros(2,4);
for i = 1:1:4
    hid_in  = W_1'*input(:,i)     ;
    hid_out = sigmoid(hid_in)     ;
    final_hid_out((1:2), i) = hid_out;
    hid_out(3,1) = 1.0            ;
    out_in  = W_2'*hid_out        ;
    out_out(:,i) = sigmoid(out_in);
end
%Output to command window
final_hid_out
out_out

%Plotting Error vs Epoch
Error = cell2mat(Error);
plot(1:1000, Error)
xlabel('epochs')
ylabel('Error')
title('Error for each Input Pattern')
legend({'Pattern 1 (1,0,0,0)','Pattern 2 (0,1,0,0)',...
    'Pattern 3 (0,0,1,0)', 'Pattern 4 (0,0,0,1)'},'Location','northeast')
```