**EXPERIMENT 4** Date:

**Problem Definition:** To implement the classifier using Error Back Propagation Algorithm in C

**Packages Used**: None

**Dataset Used:** None

**Theory:**

Error backpropagation, often simply referred to as backpropagation, is a widely used algorithm in the training of feedforward neural networks for supervised learning. Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network. This gradient is then used by an optimization algorithm, such as stochastic gradient descent, to adjust the weights to minimize the loss.

The process involves two main phases: a forward pass, where the input data is passed through the network to compute the output, and a backward pass, where the gradients are computed by propagating the error backward through the network.

Forward Pass:

In the forward pass, the input data is fed into the network, and operations defined by the network architecture are performed layer by layer to compute the output. This output is then used to calculate the loss, which measures the difference between the network's prediction and the true target values.

Backward Pass:

The backward pass starts with the computation of the gradient of the loss function with respect to the output of the network. Then, by applying the chain rule, the algorithm calculates the gradient of the loss with respect to each weight by propagating the error information back from the output layer to the input layer. This process involves the following steps:

1. Compute the derivative of the loss function with respect to the activations of the output layer.
2. For each layer, starting from the last hidden layer and moving to the first, compute the gradients of the loss with respect to the layer's inputs, which are the activations of the previous layer.
3. Compute the gradients of the loss with respect to the weights by considering the gradients with respect to the layer's inputs and the derivatives of the layer's activations with respect to its weights.

These gradients tell us how much a change in each weight would affect the loss, allowing the optimization algorithm to adjust the weights in a direction that minimizes the loss.

Updating Weights:

Once the gradients are computed, the weights are updated typically using a gradient descent optimization algorithm. The weight update rule is generally of the form:

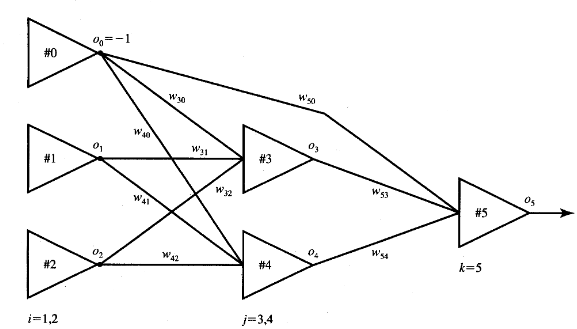
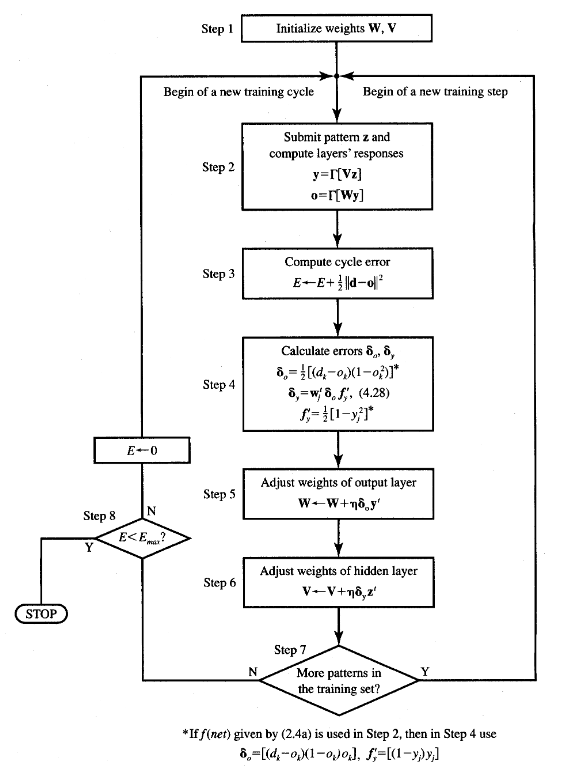
new\_weight = old\_weight - learning\_rate \* gradient

where the learning rate is a hyperparameter that controls the size of the weight updates.

Challenges with Backpropagation:

While backpropagation is a powerful tool for training neural networks, it comes with several challenges:

1. Vanishing Gradients: In deep networks, the gradients can become very small, effectively preventing the weights in the earlier layers from changing significantly. This can slow down or even halt training.
2. Exploding Gradients: Conversely, gradients can also grow exponentially, causing large weight updates that can destabilize the learning process.
3. Non-Convex Loss Functions: Neural networks typically have non-convex loss functions, which means there can be multiple local minima. Gradient descent methods can get stuck in these local minima instead of finding the global minimum.



**Code:**

#include <stdio.h>

#include <math.h>

float sigmoid(float net)

{ return 1 / (1 + exp(-1.0 \* net));}

void print\_outputs(int layer, int nodes, float matrix[layer][nodes])

{ for (int i = 1; i < layer; i++)

{ printf("Layer %d: ", i);

for (int j = 0; j < nodes && i + j < 3; j++)

{ printf("%f ", matrix[i][j]); }

printf("\n"); }

printf("\n");}

void print\_weights(int layer, int nodes, int prev\_nodes, float matrix[layer][nodes][prev\_nodes])

{ for (int i = 1; i < layer; i++)

{ printf("Layer %d: ", i);

for (int j = 0; j < nodes; j++)

{ for (int k = 0; k < prev\_nodes && i + j < 3; k++)

printf("%f ", matrix[i][j][k]); } printf("\n"); }

printf("\n");}

void forward\_pass(float weights[3][2][2], float outputs[3][2])

{ for (int layer = 1; layer < 3; layer++)

{ for (int node = 0; node < 2; node++) {

if (outputs[layer][node] == -1)

continue;

float sum = 0.0;

for (int prev\_node = 0; prev\_node < 2; prev\_node++)

{ sum += outputs[layer - 1][prev\_node] \* weights[layer][node][prev\_node]; }

outputs[layer][node] = sigmoid(sum); } }}

void backpropagation(float weights[3][2][2], float outputs[3][2], float diff, float lr){

float gradients[3][2] = {

{},

{0, 0},

{0, -1}};

for (int layer = 2; layer > 0; layer--)

{ for (int node = 0; node < 2; node++) {

if (layer == 2 && gradients[layer][node] != -1)

{ gradients[layer][node] = outputs[layer][node] \* (1 - outputs[layer][node]) \* diff; }

else if (layer < 2) {

// grad = output \* (1 - output) \* summation(next\_layer\_node\_grad \* weight)

for (int next\_node = 0; next\_node < 2; next\_node++)

{ if (gradients[layer + 1][next\_node] != -1) {

gradients[layer][node] += outputs[layer][node] \* (1 - outputs[layer][node]) \* gradients[layer + 1][next\_node] \* weights[layer + 1][next\_node][node]; } } }

if (gradients[layer][node] != -1) {

for (int prev\_node = 0; prev\_node < 2; prev\_node++) {

weights[layer][node][prev\_node] += lr \* gradients[layer][node] \* outputs[layer - 1][prev\_node];

} } } }}

int main()

{ // weights[layer\_num][current\_node\_num][node\_num\_in\_prev\_layer]

float weights[3][2][2] = {

{},

{{0.2, 0.2}, {0.3, 0.3}},

{{0.3, 0.9}}};

// outputs[layer\_num][current\_node\_num]

float outputs[3][2] = {

{0.35, 0.7},

{0, 0},

{0, -1}};

float target\_output = 0.5;

float lr = 1;

for (int i = 0;; i++)

{ forward\_pass(weights, outputs);

printf("Output Matrix for Forward Pass %d:\n", i);

print\_outputs(3, 2, outputs);

float diff = target\_output - outputs[2][0];

float error = 0.5 \* pow(diff, 2);

printf("Error: %f\n\n", error);

if (error < 0.01)

break;

backpropagation(weights, outputs, diff, lr);

printf("Weight Matrix for Backward Pass %d:\n", i);

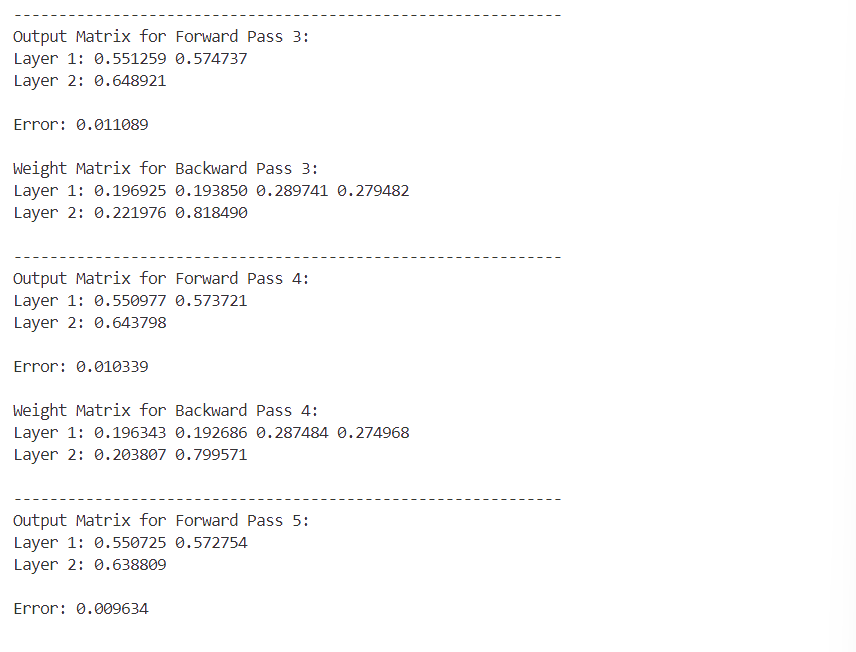
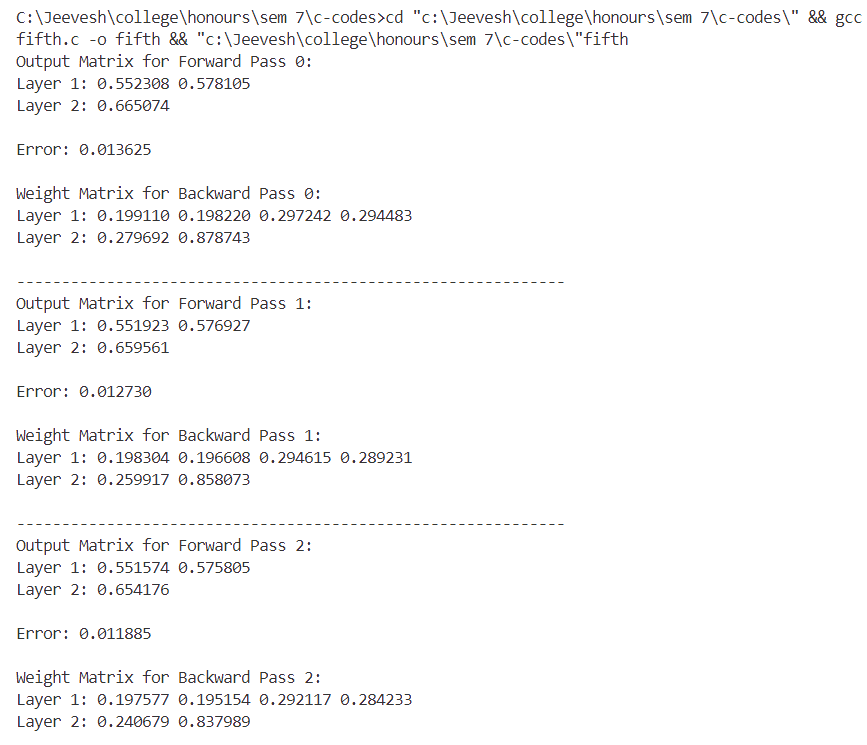
print\_weights(3, 2, 2, weights);

printf("-------------------------------------------------------------\n");

} return 0;

}

**Output:**

****

**Conclusion:**

Implemented the classifier using Error Back Propagation Algorithm successfully.