```
import numpy as np


N_INPUTS = 3
N_HIDDEN = 5
N_OUTPUTS = 1
N_WOLVES = 30
MAX_ITER = 100
LB = -10.0
UB = 10.0

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)


def forward_pass(input_data, weights):
    hidden_layer = np.dot(input_data, weights[:N_INPUTS *
N_HIDDEN].reshape(N_INPUTS, N_HIDDEN))
    hidden_layer = sigmoid(hidden_layer)
    output = np.dot(hidden_layer, weights[N_INPUTS *
N_HIDDEN:].reshape(N_HIDDEN, N_OUTPUTS))
    return output, hidden_layer

def fitness_function(weights, inputs, targets, n_samples):
    total_error = 0.0
    for i in range(n_samples):
        output, _ = forward_pass(inputs[i], weights)
        total_error += (output - targets[i]) ** 2
    return total_error / n_samples


def rand_range(min_val, max_val):
    return min_val + (max_val - min_val) * np.random.random()

def update_position(positions, alpha_pos, beta_pos, delta_pos, i, t):
    A = 2 - t * (2.0 / MAX_ITER)
    C = 2 * np.random.random()

    for j in range(len(positions[i])):

        D_alpha = np.abs(C * alpha_pos[j] - positions[i][j])
        D_beta = np.abs(C * beta_pos[j] - positions[i][j])
        D_delta = np.abs(C * delta_pos[j] - positions[i][j])

        new_position = alpha_pos[j] - A * D_alpha if np.random.random()
> 0.5 else beta_pos[j] - A * D_beta
```

```
        positions[i][j] = np.clip(new_position, LB, UB)


def gwo_optimization(inputs, targets, n_samples):
    positions = np.random.uniform(LB, UB, (N_WOLVES, N_INPUTS *
N_HIDDEN + N_HIDDEN))  # Wolves' positions (weights)
    fitness = np.zeros(N_WOLVES)  # Fitness of wolves
    alpha_pos = np.zeros(N_INPUTS * N_HIDDEN + N_HIDDEN)  # Best
position (alpha wolf)
    beta_pos = np.zeros(N_INPUTS * N_HIDDEN + N_HIDDEN)   # Second best
position (beta wolf)
    delta_pos = np.zeros(N_INPUTS * N_HIDDEN + N_HIDDEN)  # Third best
position (delta wolf)
    alpha_score = float('inf')  # Best fitness value (alpha wolf)
    beta_score = float('inf')   # Second best fitness value (beta wolf)
    delta_score = float('inf')  # Third best fitness value (delta wolf)


    for i in range(N_WOLVES):
        fitness[i] = fitness_function(positions[i], inputs, targets,
n_samples)


        if fitness[i] < alpha_score:
            alpha_score = fitness[i]
            alpha_pos = positions[i]
        elif fitness[i] < beta_score:
            beta_score = fitness[i]
            beta_pos = positions[i]
        elif fitness[i] < delta_score:
            delta_score = fitness[i]
            delta_pos = positions[i]

    for t in range(MAX_ITER):
        for i in range(N_WOLVES):
            update_position(positions, alpha_pos, beta_pos, delta_pos,
i, t)
            fitness[i] = fitness_function(positions[i], inputs,
targets, n_samples)


            if fitness[i] < alpha_score:
                alpha_score = fitness[i]
                alpha_pos = positions[i]
            elif fitness[i] < beta_score:
                beta_score = fitness[i]
                beta_pos = positions[i]
```

```
            elif fitness[i] < delta_score:
                delta_score = fitness[i]
                delta_pos = positions[i]


        if t % 10 == 0:
            print(f"Iteration {t}/{MAX_ITER}, Best Fitness =
{alpha_score}")

    print(f"\nBest Fitness: {alpha_score}")
    return alpha_pos


inputs = np.array([
    [0.0, 0.0, 1.0],
    [1.0, 0.0, 1.0],
    [0.0, 1.0, 1.0],
    [1.0, 1.0, 1.0]
])

targets = np.array([0.0, 1.0, 1.0, 0.0])


best_weights = gwo_optimization(inputs, targets, len(inputs))

print("\nEvaluating the final model with the best weights...")
for i in range(len(inputs)):
    output, _ = forward_pass(inputs[i], best_weights)
    print(f"Input: {inputs[i]}, Predicted Output: {output[0]}, Actual
Target: {targets[i]}")
```

Iteration 0/100, Best Fitness = 0.5000013911617378

Iteration 10/100, Best Fitness = 0.5000013911617378

Iteration 20/100, Best Fitness = 0.5000013911617378

Iteration 30/100, Best Fitness = 0.5000013911617378

Iteration 40/100, Best Fitness = 0.5000013911617378

Iteration 50/100, Best Fitness = 0.5000013911617378

Iteration 60/100, Best Fitness = 0.5000013911617378

Iteration 70/100, Best Fitness = 0.5000013911617378

Iteration 80/100, Best Fitness = 0.5000013911617378

Iteration 90/100, Best Fitness = 0.5000013911617378

Best Fitness: 0.5000013911617378

Evaluating the final model with the best weights...

Input: [0. 0. 1.], Predicted Output: -0.0022698934351217197, Actual Target: 0.0

Input: [1. 0. 1.], Predicted Output: -1.0305768090951018e-07, Actual Target: 1.0

Input: [0. 1. 1.], Predicted Output: -1.0305768090951018e-07, Actual Target: 1.0

Input: [1. 1. 1.], Predicted Output: -4.678811484419649e-12, Actual Target: 0.0

=== Code Execution Successful ===