Implementation of Neural Network Training using Forward and Backward Propagation in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to implement neural network training using forward and backward propagation in Python. Neural network is used to process the input data and provide accurate predictions. The training of neural network is performed in two stages: forward and backward propagation. During the training process, the cost function is computed and the weights and biases are updated to reach the optimal solution.

The basic steps of neural network training using forward and backward propagation are explained: defining neural network (input, target output, and weights and biases), performing forward propagation, computing cost function, performing backward propagation, updating weights and biases, printing predicted output, and plotting charts.

The developed program was tested on an experimental data. The program has successfully performed the basic steps of neural network training using forward and backward propagation and provided the required results.

Keywords
Computer Science, Artificial Intelligence, Machine Learning, Neural Network, Training, Forward, Backward, Propagation, Python, Programming.

1. INTRODUCTION
In the recent years, machine learning has played a major role in the development of computer systems. Machine learning (ML) is a branch of Artificial Intelligence (AI) which is focused on the development of methods and algorithms to improve the performance and efficiency of computer programs [1-10].

Neural networks is an important area in the field of machine learning. It is sharing knowledge with many other fields like: programming, data science, mathematics, statistics, and numerical methods [11-14, 15-18].

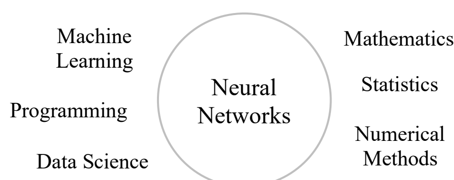

Fig 1: Area of Neural Networks

The neural networks are designed and trained to process input data and produce accurate predictions. They are intensively used in a wide range of applications. For example: prediction, classification, recognition, searching, filtering, and more.

2. LITERATURE REVIEW
The literature was reviewed to explore the fundamental concepts, methods, and applications of neural network training using forward and backward propagation [19-25, 26-32].

Neural networks are used to process input data using weights and biases and provide accurate predictions. Now, they are widely applied in different fields like: technology, science, medicine, biology, business, languages, media, etc.

The research in neural networks started in the early 1940s. The artificial neuron was first designed to simulate the function of the biological neuron in the human brain. In 1943, McCulloch and Pitts proposed a mathematical model for the neuron [33]. Later, in 1958, Rosenblatt implemented the Perceptron [34]. Then, Widrow and Hoff developed Adaline and Madaline models [35].

After that, in 1986, the "Back-propagation" algorithm was developed by Rumelhart, Hinton, and Williams [36]. It was based on the Chain Rule method. The back-propagation algorithm allowed neural networks to solve complex problems.

The modern "deep learning" revolution began in the 2000s. New technologies have emerged, for example: AlexNet, Transformer, Large Language Model (LLM), and generative models like ChatGPT.

The fundamental concepts of neural network training using forward and backward propagation are explained in the following section.

Neural Network:
Neural network is a set of connected neurons. It is used to process the input data using the given weights and biases to produce the predicted output. The neural network is trained in two stages: forward and backward propagation.

The concept of neural network is illustrated in the following diagram:

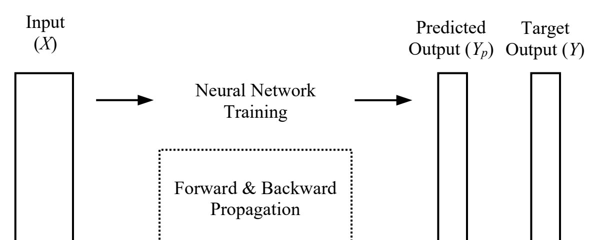

Fig 2: Concept of Neural Network

Neuron:

Neuron is the building unit of neural network. Simply, it takes the input values and process them using the given weights and bias. Then, it calculates the predicted output.

The neuron structure is explained in the following diagram:

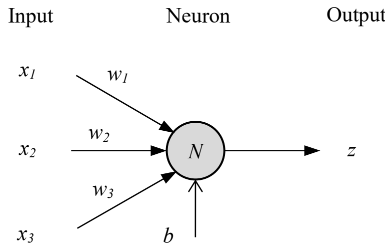

Fig 3: Concept of Neuron

Where:

(N) is the neuron,
(x) is the input value,
(w) is the weight,
(b) is the bias, and
(z) is the computed output,

The neuron output (z) is calculated by the following formula:

$$z = \sum x_i \, . \, w_i + b$$
$$= x_1 \, . \, w_1 + x_2 \, . \, w_2 + x_3 \, . \, w_3 + b$$

Example:

Assume the following neuron with the given input, weights, and bias.

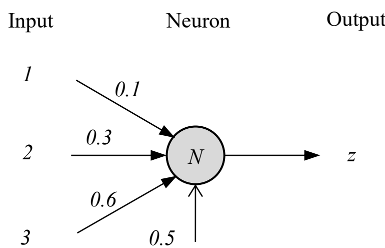

Fig 4: Example of Neuron

The neuron output (z) is calculated as shown here:

$$z = 1\ (0.1) + 2\ (0.3) + 3\ (0.6) + 0.5 = 3$$

Activation Function:

The neuron output is transformed using an activation function, like sigmoid, ReLU, and softmax. The sigmoid function is computed by the following formula:

$$sigmoid(z) = \frac{1}{(1 + e^{-z})}$$

Now, the activated output for the previous example is calculated as shown here:

$$sigmoid(3) = \frac{1}{(1 + e^{-3})} = 0.95257413$$

The sigmoid function is used to convert the output values into probabilities, between (0) and (1).

The graph of sigmoid function is shown in the following chart:

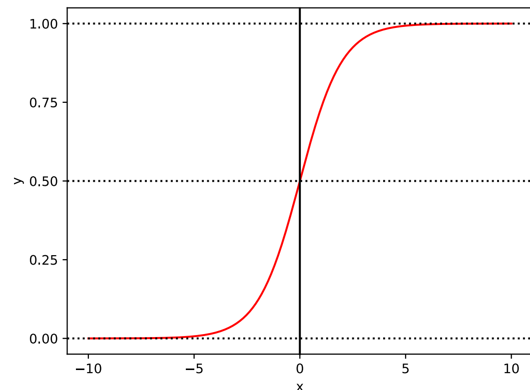

Fig 5: Sigmoid Function

Neural Network Model:

In general, the neural network model is composed of three layers: input layer, hidden layer, and output layer.

The neural network model is shown in the following diagram:

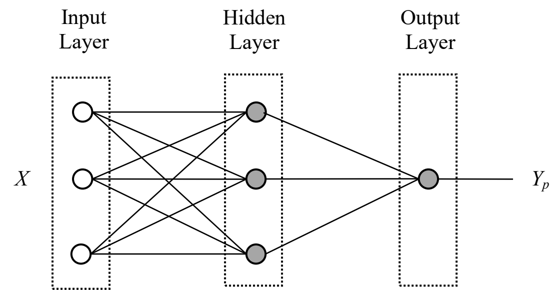

Fig 6: Neural Network Model

The input layer receives the input values. The hidden layer processes the input values using the given weights and biases. The output layer calculates the predicted output.

Note: The neural network can have multiple hidden layers.

Neural Network Training:

The neural network is trained by changing the weights and biases to reach the optimal solution. The training of neural network is performed in two stages: forward and backward propagation.

The forward propagation is done in the following order:

$$Input \rightarrow Hidden \rightarrow Output$$

And, the backward propagation is done in the opposite order:

$$Input \leftarrow Hidden \leftarrow Output$$

The training process is illustrated in the following diagram:

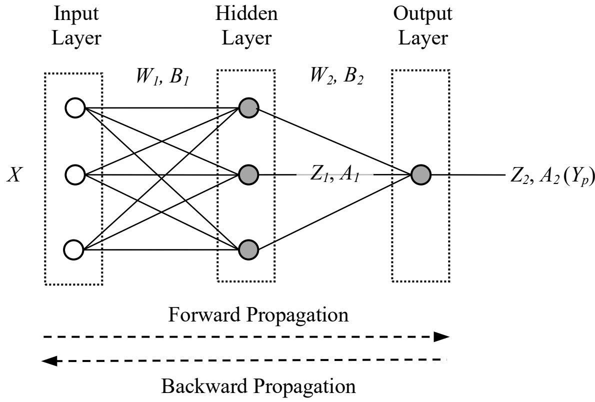

Fig 7: Forward and Backward Propagation

Forward Propagation:

The forward propagation is performed using the following formulas:

$$Z_1 = (X \, . \, W_1) + B_1$$

$$A_1 = Act(Z_1)$$

$$Z_2 = (A_1 \, . \, W_2) + B_2$$

$$A_2 = Act(Z_2)$$

Where:

(X) is the input,
(W_1) is the weights of layer (1),
(B_1) is the bias of layer (1),
(Z_1) is the computed output of layer (1),
(A_1) is the activated output of layer (1),
(W_2) is the weights of layer (2),
(B_2) is the bias of layer (2),
(Z_2) is the computed output of layer (2), and
(A_2) is the activated output of layer (2).

Cost Function:

The cost function (C) is calculated using the Mean Squared Error (MSE) formula as shown here:

$$C(MSE) = (\frac{1}{n}) \sum (Y_p - Y)^2$$

Where:

(Y_p) is the predicted output,
(Y) is the target output, and
(n) is the number of output values.

Backward Propagation:

The backward propagation is performed using the partial derivatives of the cost function (C) with respect to weights (W) and biases (B) in the output and hidden layers (2 and 1). The partial derivatives are calculated using the Chain Rule method. They are explained in the following section.

The partial derivative of the cost function (C) with respect to weight (W_2):

$$\frac{\partial C}{\partial W_2} = \frac{\partial C}{\partial A_2} \, . \, \frac{\partial A_2}{\partial Z_2} \, . \, \frac{\partial Z_2}{\partial W_2}$$

$$= (A_2 - Y) \, . \, Act'(Z_2) \, . \, A_1$$

The partial derivative of the cost function (C) with respect to weight (B_2):

$$\frac{\partial C}{\partial B_2} = \frac{\partial C}{\partial A_2} \, . \, \frac{\partial A_2}{\partial Z_2} \, . \, \frac{\partial Z_2}{\partial B_2}$$

$$= (A_2 - Y) \, . \, Act'(Z_2) \, . \, 1$$

The partial derivative of the cost function (C) with respect to weight (W_1):

$$\frac{\partial C}{\partial W_1} = \frac{\partial C}{\partial A_2} \, . \, \frac{\partial A_2}{\partial Z_2} \, . \, \frac{\partial Z_2}{\partial A_1} \, . \, \frac{\partial A_1}{\partial Z_1} \, . \, \frac{\partial Z_1}{\partial W_1}$$

$$= (A_2 - Y) \, . \, Act'(Z_2) \, . \, W_2 \, . \, Act'(Z_1) \, . \, X$$

The partial derivative of the cost function (C) with respect to weight (B_1):

$$\frac{\partial C}{\partial B_1} = \frac{\partial C}{\partial A_2} \, . \, \frac{\partial A_2}{\partial Z_2} \, . \, \frac{\partial Z_2}{\partial A_1} \, . \, \frac{\partial A_1}{\partial Z_1} \, . \, \frac{\partial Z_1}{\partial B_1}$$

$$= (A_2 - Y) \, . \, Act'(Z_2) \, . \, W_2 \, . \, Act'(Z_1) \, . \, 1$$

Gradient Descent:

The weights and biases are updated using the Gradient Descent method. It is a famous mathematical method used to find the optimal solution by taking the derivative of the cost function.

The concept of gradient descent method is illustrated in the following diagram:

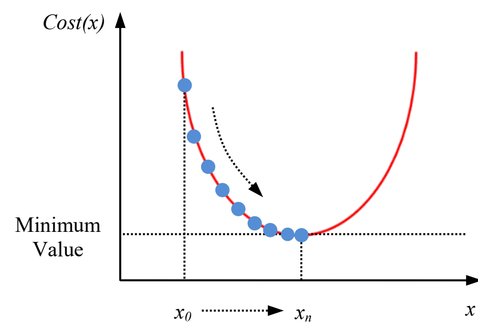

Fig 8: Concept of Gradient Descent

The optimal solution is reached when the cost function is minimum. The formula of gradient descent is shown here:

$$x_{new} = x_{old} - \alpha \, (\frac{\partial Cost}{\partial x})$$

The weights (W) and biases (B) are updated using the gradient descent formula. It is explained in the following section.

The weights (W) are updated by the following formula:

$$W = W - \alpha \, (\frac{\partial Cost}{\partial W})$$

Where:

(W) are the weights,
(α) is the learning rate, and

$(\frac{\partial Cost}{\partial W})$ is the partial derivative of the cost function with respect to weights.

And, the biases (B) are updated by the formula:

$$B = B - \alpha \left(\frac{\partial Cost}{\partial B}\right)$$

Where:

(B) are the biases,
(α) is the learning rate, and
$(\frac{\partial Cost}{\partial B})$ is the partial derivative of the cost function with respect to biases.

Training Algorithm:

The training of neural network is performed in two stages: forward and backward propagation. The training process is explained step by step in the following algorithm.

Algorithm 1: Neural Network Training

```
# define input
X = [[…],
      … ,
     […]]
# define target output
Y = [[…],
      … ,
     […]]
# initialize weights and biases
W1 = rand(n1, n)
B1 = zero(n1, 1)
W2 = rand(n2, n1)
B2 = zero(n2, 1)
# initialize cost list
cost = []
# learning rate
alpha = 0.5
# number of iterations
Nt = 10^4
for t = 1 to Nt do
    # --------------------------------------------
    # forward propagation:
    # --------------------------------------------
    # hidden layer (1)
    Z1 = (X * W1) + B1
    A1 = Act(Z1)
    # output layer (2)
    Z2 = (A1 * W2) + B2
    A2 = Act(Z2)
    # --------------------------------------------
    # cost function:
    # --------------------------------------------
    mse = compute_mse(A2, Y)
    cost.append(mse)
    # --------------------------------------------
    # backward propagation:
    # --------------------------------------------
    # output layer (2)
    dC/dA2 = (A2 – Y)
    dA2/dZ2 = Act_der(Z2)
    dC/dZ2 = dC/dA2 * dA2/dZ2
    dZ2/dW2 = A1
    dC/dW2 = dC/dZ2 * dZ2/dW2
    dC/dB2 = sum_col(dC/dZ2)
    # hidden layer (1)
    dZ2/dA1 = W2
    dA1/dZ1 = Act_der(Z1)
    dC/dZ1 = (dC/dZ2 * dZ2/dA1) * dA1/dZ1
    dZ1/dW1 = X
    dC/dW1 = dC/dZ1 * dZ1/dW1
    dC/dB1 = sum_col(dC/dZ1)
    # --------------------------------------------
    # update weights and biases:
    # --------------------------------------------
    # hidden layer (1)
    W1 = W1 – alpha * dC/dW1
    B1 = B1 – alpha * dC/dB1
    # output layer (2)
    W2 = W2 – alpha * dC/dW2
    B2 = B2 – alpha * dC/dB2
end for
print(A2)
```

Neural Network Training System:

The neural network training system is briefly described in the following summary.

Input: Input, Target Output, Weights, and Biases.
Output: Predicted Output.
Processing: First, the neural network is defined (input, output, and weights and biases). Then, the forward propagation is performed and the cost function is computed. Next, the backward propagation is performed and the weights and biases are updated. After that, the predicted output is printed and the charts are plotted.

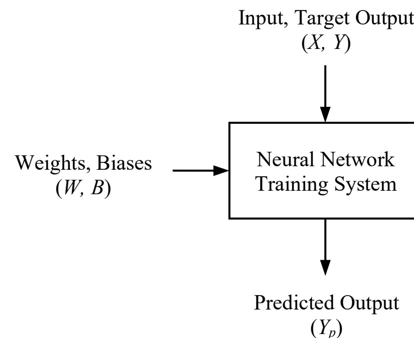

Fig 9: Neural Network Training System

Python:

Python [37] is an open source, object-oriented, and general-purpose programming language. It is easy to learn, simple to code, and powerful. It is the most popular programming language, especially in the field of machine learning.

Python provides many additional libraries for different purposes. For example: Numpy [38], Pandas [39], Matplotlib [40], Seaborn [41], SciPy [42], NLTK [43], and SK Learn [44].

3. RESEARCH METHODOLOGY

The basic steps of neural network training using forward and backward propagation are: (1) defining neural network (input, target output, and weights and biases), (2) performing forward propagation, (3) computing cost function, (4) performing backward propagation, (5) updating weights and biases (6) printing predicted output, and (7) plotting charts.

- Defining Neural Network:
  - Defining Input
  - Defining Target Output
  - Initializing Weights and Biases
- Performing Forward Propagation
- Computing Cost Function
- Performing Backward Propagation
- Updating Weights and Biases
- Printing Predicted Output
- Plotting Charts

Fig 10: Steps of Neural Network Training

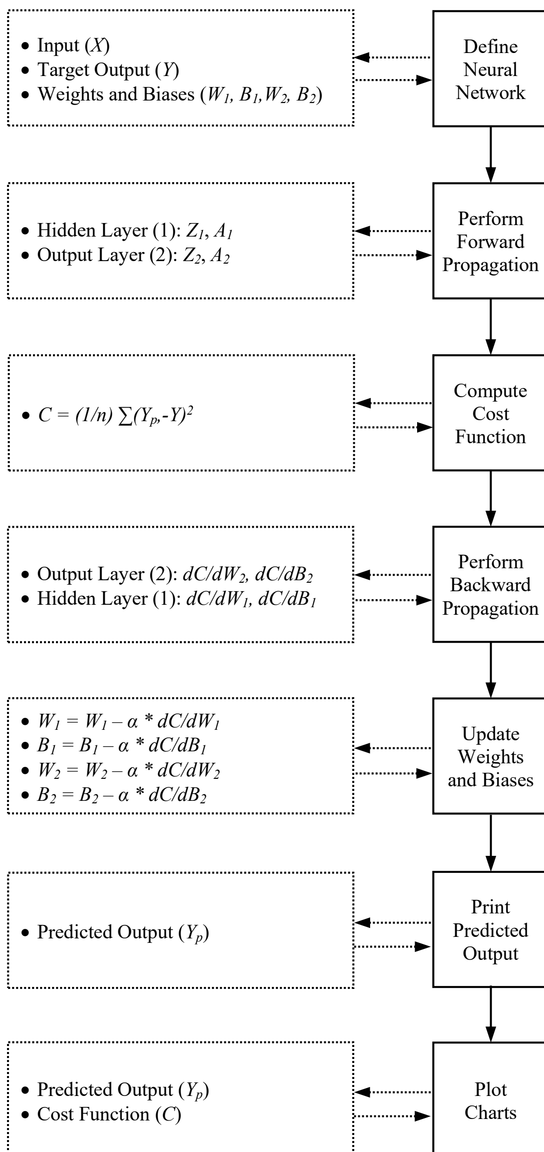

Fig 11: Flowchart of Neural Network Training

The basic steps of neural network training using forward and backward propagation are explained in details in the following section.

Note: The program is developed using only the standard functions of Python without any additional library.

1. Defining Neural Network:

The neural network is defined by the following steps:

1.1. Defining Input:

The input (*X*) is defined by the following code:

```
X = [[x0,0, x0,1, ..., x0,n-1],
     [x1,0, x1,1, ..., x1,n-1],
      ...
     [xm-1,0, xm-1,1, ..., xm-1,n-1]]
```

1.2. Defining Target Output:

The target output (*Y*) is defined by the following code:

```
Y = [[y0],
     [y1],
      ...
     [ym-1]]
```

1.3. Initializing Weights and Biases:

The weights (*W*) and biases (*B*) of the hidden and output layers (1 and 2) are initialized by the following code:

```
import numpy as np

# hidden layer (1)
W1 = rand(n1, n)
B1 = zero(n1, 1)

# output layer (2)
W2= rand(n2, n1)
B2 = zero(n2, 1)
```

2. Performing Forward Propagation:

The forward propagation is performed to calculate the outputs of the hidden and output layers (1 and 2). It is done by the following code:

```
# hidden layer (1)
Z1 = add(dot(X, T(W1)), B1)
A1 = sigmoid(Z1)

# output layer (2)
Z2 = add(dot(A1, T(W2)), B2)
A2 = sigmoid(Z2)
```

The function (*T*) is used to compute the transpose of a matrix. It is done by the following code:

```
def T(m):
    r, c = len(m), len(m[0])
    t = []
    for i in range(c):
        row = []
        for j in range(r):
            row.append(m[j][i])
        t.append(row)
    return t
```

The function (*dot*) is used to multiply two matrices (row by column). It is done by the following code:

```
def dot(m1, m2):
    r1, c1 = len(m1), len(m1[0])
    r2, c2 = len(m2), len(m2[0])
```

```
    t = []
    for i in range(r1):
        row = []
        for j in range(c2):
            sum = 0
            for k in range(c1):
                sum += m1[i][k] * m2[k][j]
            row.append(sum)
        t.append(row)
    return t
```

The function (*add*) is used to add the bias (*B*) to a matrix. It is done by the following code:

```
def add(m, b):
    r, c = len(m), len(m[0])
    t = []
    for i in range(r):
        row = []
        for j in range(c):
            row.append(m[i][j] + b[j][0])
        t.append(row)
    return t
```

The function (*sigmoid*) is used to compute the sigmoid values of a matrix. It is done by the following code:

```
import math

def sigmoid(m):
    r, c = len(m), len(m[0])
    t = []
    for i in range(r):
        row = []
        for j in range(c):
            row.append(1 / (1 + math.exp(
                                    -m[i][j])))
        t.append(row)
    return t
```

3. Computing Cost Function:

The cost function is computed using the MSE formula. It is done by the following code:

```
def compute_mse(yp, y):
    r, c = len(y), len(y[0])
    sum = 0
    for i in range(r):
        for j in range(c):
            sum += (yp[i][j] - y[i][j])**2
    return sum/r
```

4. Performing Backward Propagation:

The backward propagation is performed to calculate the partial derivatives of the cost function with respect to weights and biases of the output and hidden layers (2 and 1). It is done by the following code:

```
# output layer (2)
dC_dA2 = sub(A2, Y)
dA2_dZ2 = sigmoid_der(A2)
dC_dZ2 = mul(dC_dA2, dA2_dZ2)
dZ2_dW2 = A1
dC_dW2 = dot(T(dC_dZ2), dZ2_dW2)
dC_dB2 = sum_col(dC_dZ2)

# hidden layer (1)
dZ2_dA1 = W2
dA1_dZ1 = sigmoid_der(A1)
dC_dZ1 = mul(dot(dC_dZ2, dZ2_dA1), dA1_dZ1)
dZ1_dW1 = X
dC_dW1 = dot(T(dC_dZ1, dZ1_dW1)
dC_dB1 = sum_col(dC_dZ1)
```

The function (*sub*) is used to subtract two matrices. It is done by the following code:

```
def sub(m1, m2):
    r, c = len(m1), len(m1[0])
    t = []
    for i in range(r):
        row = []
        for j in range(c):
            row.append(m1[i][j] - m2[i][j])
        t.append(row)
    return t
```

The function (*mul*) is used to multiply two matrices (element by element). It is done by the following code:

```
def mul(m1, m2):
    r, c = len(m1), len(m1[0])
    t = []
    for i in range(r):
        row = []
        for j in range(c):
            row.append(m1[i][j] * m2[i][j])
        t.append(row)
    return t
```

The function (*sigmoid_der*) is used to calculate the sigmoid derivative of a matrix. It is done by the following code:

```
def sigmoid_der(m):
    r, c = len(m), len(m[0])
    t = []
    for i in range(r):
        row = []
        for j in range(c)):
            row.append(m[i][j] * (1 - m[i][j]))
        t.append(row)
    return t
```

The function (*sum_col*) is used to calculate the sum of columns in a matrix. It is done by the following code:

```
def sum_col(m):
    mt = T(m)
    t = []
    for row in mt:
        t.append([sum(row)])
    return t
```

5. Updating Weights and Biases:

The weights (*W*) and biases (*B*) of the hidden and output layers (1 and 2) are updated by the following code:

```
# hidden layer (1)
W1 = sub(W1, mulk(alpha, dC_dW1))
B1 = sub(B1, mulk(alpha, dC_dB1)

# output layer (2)
W2 = sub(W2, mulk(alpha, dC_dW2))
B2 = sub(B2, mulk(alpha, dC_dB2)
```

The function (*mulk*) is used to multiply a constant (*k*) by a matrix. It is done by the following code:

```
def mulk(k, m):
    r, c = len(m), len(m[0])
    t = []
    for i in range(r):
        row = []
```

```
        for j in range(c):
            row.append(k * m[i][j])
        t.append(row)
    return t
```

6. Printing Predicted Output:

The final predicted output (Y_p) is printed by the following code:

```
print("Predicted Output (Yp):")
print(A2)
```

7. Plotting Charts:

The predicted output (Y_p) is plotted by the following code:

```
import matplotlib.pyplot as plt

plt.bar(range(len(A2)), A2, color='red')
plt.title("Predicted Output (Yp)")
plt.xlabel("Output")
plt.ylabel("Value")
plt.show()
```

The cost function (*cost*) is plotted by the following code:

```
plt.plot(range(len(cost), cost)
plt.title("Cost Function")
plt.xlabel("Iterations")
plt.ylabel("MSE")
plt.show()
```

4. RESULTS AND DISCUSSION

The developed program was tested on an experimental data. The program has successfully performed the basic steps of neural network training using forward and backward propagation and provided the required results. The program output is explained in details in the following section.

Defining Network:

The neural network is defined by the following steps:

1. Defining Input:

The input (X) is defined as shown in the following view:

```
X = [[0,0], [0,1], [1,0], [1,1]]
```

2. Defining Output:

The target output (Y) is defined as shown in the following view:

```
Y = [[0], [1], [1], [0]]
```

3. Initializing Weights and Biases:

The weights (W) and biases (B) of the hidden and output layers (1 and 2) are initialized as shown in the following view:

```
W1 =
[[0.548813503927, 0.715189366372],
[0.602763376072, 0.544883182997],
[0.423654799339, 0.645894113067],
[0.437587211263, 0.891773000782]]
B1 =
[[0], [0], [0], [0]]
W2 =
[[0.963662760501, 0.383441518826,
0.791725038083, 0.528894919753]]
B2 =
[[0]]
```

Performing Forward Propagation:

The forward propagation is performed to calculate the outputs of the hidden and output layers (1 and 2) for each iteration. The final values are shown in the following view:

```
Z1 =
[[-3.086808958780, -5.714160914295,
-1.541133245897, -3.893202025029],
[3.669308378316, -2.071321268248,
-0.540251221062, -1.468084078638],
[3.654230542142, -2.059714730152,
-0.527205042535, -1.497264995082],
[10.410347879239, 1.583124915895,
0.473676982300, 0.927852951310]]
A1 =
[[0.043654663693, 0.003288070362,
0.176370594559, 0.019972936106],
[0.975139694888, 0.111915649644,
0.368129143778, 0.187233999257],
[0.974771543416, 0.113074436230,
0.371169003063, 0.182833794442],
[0.999969881712, 0.829646627097,
0.616253678732, 0.716639492918]]
Z2 =
[[-3.929993137788], [4.147227900538],
[4.146570386955], [-4.164021815706]]
A2 =
[[0.019265362287], [0.984437831876],
[0.984427755567], [0.015306967769]]
```

Computing Cost Function:

The cost function (*cost*) is computed using the MSE formula for each iteration and printed as shown in the following view:

```
t:     0       mse = 0.364403845639
t:  1000       mse = 0.016851789467
t:  2000       mse = 0.002733952277
t:  3000       mse = 0.001363793880
t:  4000       mse = 0.000889025569
t:  5000       mse = 0.000653314956
t:  6000       mse = 0.000513794160
t:  7000       mse = 0.000422063612
t:  8000       mse = 0.000357381934
t:  9000       mse = 0.000309434622
t: 10000       mse = 0.000272533330
```

Performing Backward Propagation:

The backward propagation is performed to calculate the partial derivatives of the cost function with respect to weights and biases for each iteration. The final values are shown in the following view:

```
dC_dW2 =
[[-0.000218581017, 0.000138935247,
0.000030008199, 0.000084326286]]
dC_dB2 =
[[0.000117589732]]
dC_dW1 =
[[-0.000064612597, -0.000063610835],
[-0.000063810771, -0.000065613364],
[0.000003513949, 0.000002722548],
[-0.000053806645, -0.000050847865]]
dC_dB1 =
[[0.000039156887], [0.000101864297],
[0.000011354718], [0.000086449314]]
```

Updating Weights and Biases:

The weights (W) and biases (B) of the hidden and output layers (1 and 2) are updated for each iteration. The final values are shown in the following view:

```
W1 =
[[6.741071807221, 6.756149142514],
[3.654478089529, 3.642872452729],
[1.013926446387, 1.000880663561],
[2.395963933270, 2.425143370324]]
B1 =
[[-3.086828537224], [-5.714211846444],
[-1.541138923256], [-3.893245249686]]
W2 =
[[11.019352015293, -7.362429605447,
-3.038557198956, -4.810586812074]]
B2 =
[[-3.754895115381]]
```

Predicted Output:

The final predicted output (Y_p) is computed and printed as shown in the following view:

```
Predicted Output (Yp):
[[0.019265362287], [0.984437831876],
[0.984427755567], [0.015306967769]]
```

Plotting Charts:

The predicted output (Y_p) of the neural network is plotted as shown in the following chart:

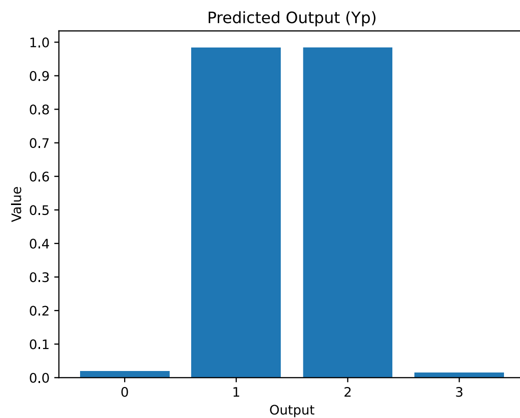

Fig 12: Predicted Output Plot

The plot shows that the predicted output (Y_p) is [0, 1, 1, 0], which is equal to the target output (Y).

The cost function (C) of the neural network is plotted as shown in the following chart:

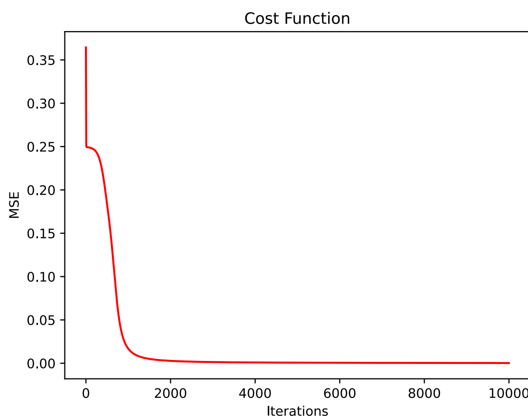

Fig 13: Cost Function Plot

The plot shows that the error is decreasing with iterations towards zero. This indicates that the neural network is converging to the optimal solution.

In addition, the weights and bias of the output layer (2) are plotted as shown in the following chart:

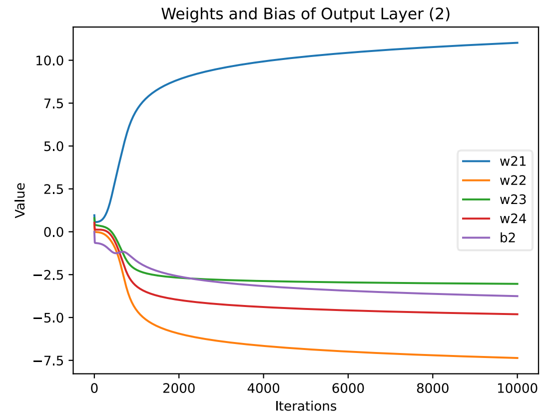

Fig 14: Weights and Bias Plot

The plot shows that the weights and bias are updated during the training process. First, they started with initial values. Then, they are changed using the forward and backward propagation. At last, they stabilize and reach their final values.

Note: The developed program was tested on more different datasets. For example, the target output (Y) = [1, 0, 0, 1], [0, 1, 1, 1], and [0, 0, 0, 1]. The program always provided the correct predictions.

In summary, it is clear that the developed program has successfully performed the basic steps of neural network training using forward and backward propagation and provided the required results.

5. CONCLUSION

In this research, the goal was to implement neural network training using forward and backward propagation in Python. The literature was reviewed to explore the fundamental concepts of neural network training: neural network, neuron, activation function, neural network model, neural network training, forward propagation, cost function, backward propagation, gradient descent, and training algorithm.

The author developed a program in Python to perform the basic steps of neural network training using forward and backward propagation: defining neural network (input, target output, and weights and biases), performing forward propagation, computing cost function, performing backward propagation, updating weights and biases, printing predicted output, and plotting charts.

The developed program was tested on an experimental data. The program has successfully performed the basic steps of neural network training using forward and backward propagation and provided the required results.

In the future, more work is needed to improve the methods of neural network training using forward and backward propagation. In addition, they should be more investigated on different domains and fields.

6. REFERENCES

[1] Sammut, C., & Webb, G. I. (2011). "Encyclopedia of Machine Learning". Springer.

[2] Jung, A. (2022). "Machine Learning: The Basics". Springer.

[3] Kubat, M. (2021). "An Introduction to Machine Learning". Springer.

[4] Li, H. (2023). "Machine Learning Methods". Springer.

[5] Zollanvari, A. (2023). " Machine Learning with Python". Springer.

[6] Chopra, D., & Khurana, R. (2023). "Introduction to Machine Learning with Python". Bentham Science Publishers.

[7] Müller, A. C., & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly Media.

[8] Raschka, S. (2015). "Python Machine Learning". Packt Publishing.

[9] Forsyth, D. (2019). "Applied Machine Learning". Springer.

[10] Sarkar, D., Bali, R., & Sharma, T. (2018). "Practical Machine Learning with Python". Apress.

[11] Igual, L., & Segui, S. (2017). "Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications". Springer.

[12] VanderPlas, J. (2017). "Python Data Science Handbook: Essential Tools for Working with Data". O'Reilly Media.

[13] Muddana, A., & Vinayakam, S. (2024). "Python for Data Science". Springer.

[14] Unpingco, J. (2021). "Python Programming for Data Analysis". Springer.

[15] Zelle, J. (2017). "Python Programming: An Introduction to Computer Science". Franklin, Beedle & Associates.

[16] Chun, W. (2001). "Core Python Programming". Prentice Hall Professional.

[17] Padmanabhan, T. (2016). "Programming with Python". Springer.

[18] Beazley, D., & Jones, B. K. (2013). "Python Cookbook: Recipes for Mastering Python 3". O'Reilly Media.

[19] Gurney, K. (1997). "An Introduction to Neural Networks". UCL Press.

[20] Krose, B., & Smagt, P. (1996). "An Introduction to Neural Networks". University of Amsterdam.

[21] Haykin, S. (2009). "Neural Networks and Learning Machines". Pearson.

[22] Silva, I., Spatti, D., Flauzino, R., Liboni, L., & Alves, S. (2017). "Artificial Neural Networks: A Practical Course". Springer.

[23] Kinsley, H., & Kukieła, D. (2020). "Neural Networks from Scratch in Python".

[24] Rajput, V. (2023). "Ultimate Neural Network Programming with Python". Orange Education.

[25] De Marchi, L., & Mitchell, L. (2019). "Hands-On Neural Networks". Packt Publishing.

[26] Aggarwal, C. (2018). "Neural Networks and Deep Learning: A Textbook". Springer.

[27] Nielsen, M. (2015). "Neural Networks and Deep Learning". Determination Press.

[28] Goodfellow, I., Bengio, Y., & Courville, A. (2016). "Deep Learning". MIT Press.

[29] Prince, S. (2023). "Understanding Deep Learning". MIT Press.

[30] Menshawy, A. (2018). "Deep Learning by Example". Packt Publishing.

[31] Zhang, A., Lipton, Z., Li, M., & Smola, A. (2023). "Dive into Deep Learning". Cambridge University Press.

[32] Chollet, F. (2018). "Deep Learning with Python". Manning Publications.

[33] McCulloch W., & Pitts W. (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". The Bulletin of Mathematical Biophysics. 5 (4): 115–133.

[34] Rosenblatt F (1958). "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain". Psychological Review. 65 (6): 386–408.

[35] Widrow, B., & Hoff, M. (1960). "Adaptive Switching Circuits". Stanford University Labs.

[36] Rumelhart D., Hinton G., & Williams R. (1986). "Learning Representations by Back-Propagating Errors". Nature. 323 (6088): 533–536.

[37] Python: http://www.python.org

[38] Numpy: http://www.numpy.org

[39] Pandas: http://pandas.pydata.org

[40] Matplotlib: http://www. matplotlib.org

[41] Seaborn: http://seaborn.pydata.org

[42] SciPy: http://scipy.org

[43] NLTK: http://www.nltk.org

[44] SK Learn: http://scikit-learn.org