Department of Computer Science and Engineering
Premier University

CSE 452: Neural Network & Fuzzy Logic Laboratory

Title: Introduction to Forward Propagation in Neural Networks

Submitted by:

Name	Mohammad Hafizur Rahman Sakib
ID	0222210005101118
Section	C
Session	Spring 2025
Semester	7th Semester
Submission Date	06.08.2025

Submitted to:
MD Tamim Hossain
Lecturer, Department of CSE
Premier University
Chittagong

Remarks

Forward Propagation Custom Code

```python
import numpy as np

class NeuralNetwork:
    def __init__(self):
        # Layer 1 weights and biases (input: 3 → hidden1: 3)
        self.W1 = np.array([[2, 4, 6],
                            [3, 5, 7],
                            [4, 6, 8]], dtype=np.float32)
        self.b1 = np.array([[1], [1], [1]], dtype=np.float32)

        # Layer 2 weights and biases (hidden1: 3 → hidden2: 2)
        self.W2 = np.array([[3, 5],
                            [4, 6],
                            [7, 8]], dtype=np.float32)
        self.b2 = np.array([[2], [2]], dtype=np.float32)

        # Layer 3 weights and biases (hidden2: 2 → hidden3: 2)
        self.W3 = np.array([[5, 7],
                            [6, 8]], dtype=np.float32)
        self.b3 = np.array([[3], [3]], dtype=np.float32)

        # Output layer weights and biases (hidden3: 2 → output: 1)
        self.W4 = np.array([[4],
                            [5]], dtype=np.float32)
        self.b4 = np.array([[1]], dtype=np.float32)

    def relu(self, x):
        return np.maximum(0, x)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, x):
        x = x.reshape(-1, 1)  # Ensure column vector

        # Layer 1
        z1 = np.dot(self.W1.T, x) + self.b1
        a1 = self.relu(z1)

        # Layer 2
        z2 = np.dot(self.W2.T, a1) + self.b2
        a2 = self.relu(z2)

        # Layer 3
        z3 = np.dot(self.W3.T, a2) + self.b3
        a3 = self.sigmoid(z3)

        # Output Layer
        z4 = np.dot(self.W4.T, a3) + self.b4
        output = self.sigmoid(z4)

        return output

# Instantiate model
model = NeuralNetwork()

# Input vector
x = np.array([5, 3, 2], dtype=np.float32)

# Forward pass
output = model.forward(x)

print("Python Output:", output[0][0])
```

Figure 1: Custom Code for Forward Propagation

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   GITLENS
SAKIB ~
 python -u "c:\PU Projects\PUC Courses\7th Semester\NNFLL\Lab 03 Contents\Codes\custome_code.py"
Python Output: 0.9999546
SAKIB ~
```

Figure 2: Output of Custom Forward Propagation Code in PyTorch

Forward Propagation in TensorFlow

```python
[2]  import tensorflow as tf
     import numpy as np

     # Define weights and biases
     W1 = np.array([[2, 4, 6], [3, 5, 7], [4, 6, 8]], dtype=np.float32)
     b1 = np.array([[1], [1], [1]], dtype=np.float32)
     W2 = np.array([[3, 5], [4, 6], [7, 8]], dtype=np.float32)
     b2 = np.array([[2], [2]], dtype=np.float32)
     W3 = np.array([[5, 7], [6, 8]], dtype=np.float32)
     b3 = np.array([[3], [3]], dtype=np.float32)
     W4 = np.array([[4], [5]], dtype=np.float32)
     b4 = np.array([[1]], dtype=np.float32)

     # Define input as column vector
     X = np.array([[5], [3], [2]], dtype=np.float32)

     # Forward propagation with transposition during calculation
     Z1 = tf.matmul(tf.transpose(W1), X) + b1
     A1 = tf.nn.relu(Z1)
     Z2 = tf.matmul(tf.transpose(W2), A1) + b2
     A2 = tf.nn.relu(Z2)
     Z3 = tf.matmul(tf.transpose(W3), A2) + b3
     A3 = tf.nn.sigmoid(Z3)
     Z4 = tf.matmul(tf.transpose(W4), A3) + b4
     output = tf.nn.sigmoid(Z4)

     print("TensorFlow Output:", output.numpy())

     TensorFlow Output: [[0.9999546]]
```

Figure 3: Forward Propagation Code using TensorFlow

Forward Propagation in PyTorch

```python
import torch
import torch.nn as nn

# Define weights and biases
W1 = torch.tensor([[2, 4, 6], [3, 5, 7], [4, 6, 8]], dtype=torch.float32)
b1 = torch.tensor([[1], [1], [1]], dtype=torch.float32)
W2 = torch.tensor([[3, 5], [4, 6], [7, 8]], dtype=torch.float32)
b2 = torch.tensor([[2], [2]], dtype=torch.float32)
W3 = torch.tensor([[5, 7], [6, 8]], dtype=torch.float32)
b3 = torch.tensor([[3], [3]], dtype=torch.float32)
W4 = torch.tensor([[4], [5]], dtype=torch.float32)
b4 = torch.tensor([[1]], dtype=torch.float32)

# Define input as column vector
X = torch.tensor([[5], [3], [2]], dtype=torch.float32)

# Forward propagation with transposition during calculation
Z1 = torch.matmul(torch.transpose(W1, 0, 1), X) + b1
A1 = torch.relu(Z1)
Z2 = torch.matmul(torch.transpose(W2, 0, 1), A1) + b2
A2 = torch.relu(Z2)
Z3 = torch.matmul(torch.transpose(W3, 0, 1), A2) + b3
A3 = torch.sigmoid(Z3)
Z4 = torch.matmul(torch.transpose(W4, 0, 1), A3) + b4
output = torch.sigmoid(Z4)

print("PyTorch Output:", output.numpy())
```

```
PyTorch Output: [[0.9999546]]
```

Figure 4: Forward Propagation Code using PyTorch (Official Implementation)

```python
import numpy as np

class NeuralNetwork:
    def __init__(self):
        # Weights and biases from hand calculation
        self.W1 = np.array([[0.1, 0.2], [0.3, 0.4], [0.5, 0.6], [0.7, 0.8]])
        self.b1 = np.array([0.1, 0.2, 0.3, 0.4])
        self.W2 = np.array([[0.2, 0.3, 0.4, 0.5], [0.6, 0.7, 0.8, 0.9], [1.0, 1.1, 1.2, 1.3]])
        self.b2 = np.array([0.5, 0.6, 0.7])
        self.W3 = np.array([[0.4, 0.5, 0.6]])
        self.b3 = np.array([0.8])

    def relu(self, x):
        return np.maximum(0, x)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, x):
        # Layer 1: Input (2) -> Hidden 1 (4) with ReLU
        z1 = np.dot(self.W1, x) + self.b1
        h1 = self.relu(z1)





        # Layer 3: Hidden 2 (3) -> Output (1) with Sigmoid
        z3 = np.dot(self.W3, h2) + self.b3
        y = self.sigmoid(z3)

        return y

# Instantiate model
model = NeuralNetwork()

# Input and prediction
x = np.array([0.5, 0.3])
output = model.forward(x)
print("Python Output:", output[0])
```

```
Python Output: 0.9918939057790397
```

```python
import tensorflow as tf
import numpy as np

# Define the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(4, activation='relu', input_shape=(2,)),
    tf.keras.layers.Dense(3, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Define weights and biases interleaved layer-wise
layer_weights = [
    # Layer 1: Dense(4) -- input: 2 → 4
    np.array([[0.1, 0.3, 0.5, 0.7],
              [0.2, 0.4, 0.6, 0.8]], dtype=np.float32),
    np.array([0.1, 0.2, 0.3, 0.4], dtype=np.float32),

    # Layer 2: Dense(3) -- input: 4 → 3
    np.array([[0.2, 0.6, 1.0],
              [0.3, 0.7, 1.1],
              [0.4, 0.8, 1.2],
              [0.5, 0.9, 1.3]], dtype=np.float32),
    np.array([0.5, 0.6, 0.7], dtype=np.float32),




]

# Set weights to the model
model.set_weights(layer_weights)

# Test input
x = np.array([[0.5, 0.3]], dtype=np.float32)

# Prediction
output = model.predict(x)
print("TensorFlow Output:", output)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
TensorFlow Output: [[0.9918939]]
```

```python
import torch
import torch.nn as nn

# Define the model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(2, 4)   # 2 inputs → 4 outputs
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(4, 3)   # 4 inputs → 3 outputs
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(3, 1)   # 3 inputs → 1 output
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu1(self.layer1(x))
        x = self.relu2(self.layer2(x))
        x = self.sigmoid(self.layer3(x))
        return x

# Instantiate model
model = NeuralNetwork()





                  [0.5, 0.6],
                  [0.7, 0.8]], dtype=torch.float32),

    torch.tensor([[0.2, 0.3, 0.4, 0.5],   # layer2: (3, 4)
                  [0.6, 0.7, 0.8, 0.9],
                  [1.0, 1.1, 1.2, 1.3]], dtype=torch.float32),

    torch.tensor([[0.4, 0.5, 0.6]], dtype=torch.float32)  # layer3: (1, 3)
]

biases = [
    torch.tensor([0.1, 0.2, 0.3, 0.4], dtype=torch.float32),  # layer1
    torch.tensor([0.5, 0.6, 0.7], dtype=torch.float32),       # layer2
    torch.tensor([0.8], dtype=torch.float32)                  # layer3
]

# Set weights and biases
model.layer1.weight.data = weights[0]
model.layer1.bias.data = biases[0]
model.layer2.weight.data = weights[1]
model.layer2.bias.data = biases[1]
model.layer3.weight.data = weights[2]
model.layer3.bias.data = biases[2]

# Input and prediction
x = torch.tensor([[0.5, 0.3]], dtype=torch.float32)
with torch.no_grad():
    output = model(x)

print("PyTorch Output:", round(output.item(), 4))  # Expected ≈ 0.9566
```

```
PyTorch Output: 0.9919
```

```python
import numpy as np

class NeuralNetwork:
    def __init__(self):
        self.relu = lambda x: np.maximum(0, x)
        self.W1 = np.array([[0.5, 0.3, 0.2], [0.1, 0.4, 0.6]])
        self.b1 = np.array([0.1, 0.2, 0.3])
        self.W2 = np.array([[0.4, 0.3, 0.2], [0.1, 0.5, 0.3], [0.2, 0.1, 0.4]])
        self.b2 = np.array([0.1, 0.1, 0.1])
        self.W3 = np.array([[0.3, 0.2], [0.1, 0.4], [0.2, 0.3]])
        self.b3 = np.array([0.2, 0.2])
        self.W4 = np.array([[0.5], [0.3]])
        self.b4 = np.array([0.1])
    def forward(self, x):
        h1 = self.relu(np.dot(x, self.W1) + self.b1)
        h2 = self.relu(np.dot(h1, self.W2) + self.b2)
        h3 = self.relu(np.dot(h2, self.W3) + self.b3)
        y = np.dot(h3, self.W4) + self.b4
        return y
x = np.array([[1.0, 2.0]])
nn = NeuralNetwork()
output = nn.forward(x)
print("Output:", output[0][0])
```

```
Output: 0.8967999999999999
```

```python
import tensorflow as tf
import numpy as np
W1 = tf.constant([[0.5, 0.3, 0.2], [0.1, 0.4, 0.6]], dtype=tf.float32)
b1 = tf.constant([0.1, 0.2, 0.3], dtype=tf.float32)
W2 = tf.constant([[0.4, 0.3, 0.2], [0.1, 0.5, 0.3], [0.2, 0.1, 0.4]], dtype=tf.float32)
b2 = tf.constant([0.1, 0.1, 0.1], dtype=tf.float32)
W3 = tf.constant([[0.3, 0.2], [0.1, 0.4], [0.2, 0.3]], dtype=tf.float32)
b3 = tf.constant([0.2, 0.2], dtype=tf.float32)
W4 = tf.constant([[0.5], [0.31]], dtype=tf.float32)
b4 = tf.constant([0.1063], dtype=tf.float32)
x = tf.constant([[1.0, 2.0]], dtype=tf.float32)
h1 = tf.nn.relu(tf.matmul(x, W1) + b1)
h2 = tf.nn.relu(tf.matmul(h1, W2) + b2)
h3 = tf.nn.relu(tf.matmul(h2, W3) + b3)
y = tf.matmul(h3, W4) + b4
print("Output:", y.numpy()[0][0])
```

```
Output: 0.91551006
```

```python
import torch
W1 = torch.tensor([[0.5, 0.3, 0.2], [0.1, 0.4, 0.6]], dtype=torch.float32)
b1 = torch.tensor([0.1, 0.2, 0.3], dtype=torch.float32)
W2 = torch.tensor([[0.4, 0.3, 0.2], [0.1, 0.5, 0.3], [0.2, 0.1, 0.4]], dtype=torch.float32)
b2 = torch.tensor([0.1, 0.1, 0.1], dtype=torch.float32)
W3 = torch.tensor([[0.3, 0.2], [0.1, 0.4], [0.2, 0.3]], dtype=torch.float32)
b3 = torch.tensor([0.2, 0.2], dtype=torch.float32)
W4 = torch.tensor([[0.5], [0.31]], dtype=torch.float32)
b4 = torch.tensor([0.1063], dtype=torch.float32)
x = torch.tensor([[1.0, 2.0]], dtype=torch.float32)
h1 = torch.relu(torch.matmul(x, W1) + b1)
h2 = torch.relu(torch.matmul(h1, W2) + b2)
h3 = torch.relu(torch.matmul(h2, W3) + b3)
print("Output:", y.item())
```

```
Output: 0.9155100584030151
```

Date: 06.08.2025

To,

The Honorable Vice-Chancellor,

Through the Chairman,

Department of CSE,

Premier University,

Chattogram.

Subject: Application for waiver of 7th semester tuition fees.

Dear Sir,

I hope you are doing well in mind and in health. With due respect, I am writing to seek your assistance regarding the tuition fees of my younger son Shuvra Roy who is currently a seven-semester student at department of CSE at Premier University. My elder son is studying in Chattogram College.

As a farmer with a modest income of 15,000 taka per month, our family is facing financial challenges, especially since my wife lost her job one year ago due to the economic downturn. She worked for nearly 10 years at 'Nijera Kori' NGO as field worker. In this situation the cost of my son's education has become difficult to manage, and we are struggling to meet the tuition expenses.

I hope you will reconsider taking a look into our matter and help us unexpected situation together and help my child so that he can complete his degree and go far.

Thank you very much for your understanding and support. I look forward to your positive response.

Yours sincerely,

Shankar Chandra Roy

Relation: Son

House: Orshini Dopadarer Bari,

Union: Maitbanga,

Upazila: Sandwip,

District: Chattogram.

Father Mobile: 01961394146

Mother Mobile: 01815670313

Program: BSC in CSE

Name: Shuvra Roy

ID: 0222210005101093

Semester: 7th

Section: C

Batch: 41

Mobile: 01815670312

Department of Computer Science and Engineering
Premier University

Neural Network & Fuzzy Logic Laboratory
Course Code: CSE 452

Title: Introduction to Forward Propagation in Neural Networks

Submitted by:

Name	Sayed Hossain
ID	0222210005101102
Section	C
Session	Spring 2025
Semester	7th Semester
Submission Date	06.08.2025

Submitted to:
MD Tamim Hossain
Lecturer, Department of CSE
Premier University
Chittagong

Remarks

```python
import torch
import torch.nn as nn

class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.layer1 = nn.Linear(2, 3)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(3, 4)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(4, 2)
        self.relu3 = nn.ReLU()
        self.layer4 = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid()

        # Set weights and biases
        self.layer1.weight.data = torch.tensor([[0.2, 0.3], [0.4, 0.5], [0.6, 0.7]], dtype=torch.float32)
        self.layer1.bias.data = torch.tensor([0.1, 0.1, 0.1], dtype=torch.float32)
        self.layer2.weight.data = torch.tensor([[0.1, 0.2, 0.3], [0.4, 0.5, 0.6], [0.7, 0.8, 0.9], [0.1, 0.2, 0.3]], dtype=torch.float32)
        self.layer2.bias.data = torch.tensor([0.2, 0.2, 0.2, 0.2], dtype=torch.float32)
        self.layer3.weight.data = torch.tensor([[0.5, 0.6, 0.7, 0.8], [0.9, 0.1, 0.2, 0.3]], dtype=torch.float32)
        self.layer3.bias.data = torch.tensor([0.1, 0.1], dtype=torch.float32)
        self.layer4.weight.data = torch.tensor([[0.0312, 0.0312]], dtype=torch.float32)
        self.layer4.bias.data = torch.tensor([0.1], dtype=torch.float32)

    def forward(self, x):
        x1 = self.relu1(self.layer1(x))
        x2 = self.relu2(self.layer2(x1))
        x3 = self.relu3(self.layer3(x2))
        x4 = self.sigmoid(self.layer4(x3))
        return x1, x2, x3, x4

# Run the network
model = NeuralNetwork()
input_data = torch.tensor([[0.5, 0.3]], dtype=torch.float32)
h1_out, h2_out, h3_out, final_out = model(input_data)
activations = torch.cat([h1_out.flatten(), h2_out.flatten(), h3_out.flatten(), final_out.flatten()]).detach().numpy()
print(f"Activations: {[round(x, 4) for x in activations]}")
print(f"Final Output: {final_out.item():.4f}")
```

```
Activations: [np.float32(0.29), np.float32(0.45), np.float32(0.61), np.float32(0.502), np.float32(0.907), np.float32(1.312), np.float32(0.502), np.
Final Output: 0.5503
```

```python
import tensorflow as tf
import numpy as np

# Define the neural network with an Input layer
model = tf.keras.Sequential([
    tf.keras.Input(shape=(2,)),
    tf.keras.layers.Dense(3, activation='relu',
        kernel_initializer=tf.constant_initializer([[0.2, 0.3], [0.4, 0.5], [0.6, 0.7]]),
        bias_initializer=tf.constant_initializer([0.1, 0.1, 0.1])),
    tf.keras.layers.Dense(4, activation='relu',
        kernel_initializer=tf.constant_initializer([[0.1, 0.2, 0.3], [0.4, 0.5, 0.6], [0.7, 0.8, 0.9], [0.1, 0.2, 0.3]]),
        bias_initializer=tf.constant_initializer([0.2, 0.2, 0.2, 0.2])),
    tf.keras.layers.Dense(2, activation='relu',
        kernel_initializer=tf.constant_initializer([[0.5, 0.6, 0.7, 0.8], [0.9, 0.1, 0.2, 0.3]]),
        bias_initializer=tf.constant_initializer([0.1, 0.1])),
    tf.keras.layers.Dense(1, activation='sigmoid',
        kernel_initializer=tf.constant_initializer([[0.0312, 0.0312]]),
        bias_initializer=tf.constant_initializer([0.1]))
])

# Input data
input_data = np.array([[0.5, 0.3]], dtype=np.float32)

# Forward pass once to initialize model.input
_ = model(input_data)

# Get activations
layer_outputs = []
x = input_data
for layer in model.layers:
    x = layer(x)
    layer_outputs.append(x.numpy().flatten())

# Final output
final_output = layer_outputs[-1][0]
activations = np.concatenate(layer_outputs)
print(f"Activations: {[round(x, 4) for x in activations]}")
print(f"Final Output: {final_output:.4f}")
```

```
Activations: [np.float32(0.35), np.float32(0.43), np.float32(0.51), np.float32(0.909), np.float32(0.579), np.float32(0.708), np.float32(0.8
Final Output: 0.5498
```

Department of Computer Science and Engineering
Premier University

CSE 452: Neural Network & Fuzzy Logic Laboratory

Title: Introduction to forward propagation in Neural Networks

Submitted by:

Name	Mohammad Asmual Hoque Yousha
ID	0222210005101121
Section	C
Session	Spring 2025
Semester	7th Semester
Submission Date	06.08.2025

Submitted to:
MD Tamim Hossain
Lecturer, Department of CSE
Premier University
Chittagong

Remarks

Custom Python Implementation:

```python
import numpy as np

# ReLU Activation
def relu(x):
    return np.maximum(0, x)

class SimpleNeuralNetwork:
    def __init__(self):
        # Layer 1
        self.W1 = np.array([[0.1, 0.2], [-0.3, 0.4]])
        self.b1 = np.array([-0.1, 0.2])

        # Layer 2
        self.W2 = np.array([[0.5, -0.6], [0.7, 0.8]])
        self.b2 = np.array([0.1, -0.2])

        # Output Layer
        self.W3 = np.array([[1.0], [-0.5]])
        self.b3 = np.array([0.3])

    def forward(self, x):
        # Layer 1
        z1 = np.dot(x, self.W1) + self.b1
        a1 = relu(z1)

        # Layer 2
        z2 = np.dot(a1, self.W2) + self.b2
        a2 = relu(z2)

        # Output Layer
        y = np.dot(a2, self.W3) + self.b3
        return y.item()

# Run the model
model = SimpleNeuralNetwork()
x_input = np.array([0.5, -0.3])
output = model.forward(x_input)

print("Custom Class-Based Output:", output)
```

```
Custom Class-Based Output: 0.546
```

TensorFlow Implementation :

```python
import tensorflow as tf
import numpy as np

model = tf.keras.Sequential([
    tf.keras.layers.Dense(2, activation='relu', input_shape=(2,)),
    tf.keras.layers.Dense(2, activation='relu'),
    tf.keras.layers.Dense(1)
])

weights = [
    np.array([[0.1, -0.3], [0.2, 0.4]], dtype=np.float32).T,
    np.array([-0.1, 0.2], dtype=np.float32),
    np.array([[0.5, 0.7], [-0.6, 0.8]], dtype=np.float32).T,
    np.array([0.1, -0.2], dtype=np.float32),
    np.array([[1.0], [-0.5]], dtype=np.float32), # Corrected shape to (2, 1)
    np.array([0.3], dtype=np.float32)
]
model.set_weights(weights)

input_data = np.array([[0.5, -0.3]])
prediction = model.predict(input_data, verbose=0)

print("TensorFlow Output :", round(prediction[0][0], 3))
```

```
TensorFlow Output : 0.546
```

PyTorch Implementation:

```
import torch
import torch.nn as nn

# Define model
class NeuralNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 2)
        self.out = nn.Linear(2, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        return self.out(x)

model = NeuralNet()

# Set weights manually
with torch.no_grad():
    model.fc1.weight = nn.Parameter(torch.tensor([[0.1, -0.3], [0.2, 0.4]]))
    model.fc1.bias = nn.Parameter(torch.tensor([-0.1, 0.2]))
    model.fc2.weight = nn.Parameter(torch.tensor([[0.5, 0.7], [-0.6, 0.8]]))
    model.fc2.bias = nn.Parameter(torch.tensor([0.1, -0.2]))
    model.out.weight = nn.Parameter(torch.tensor([[1.0, -0.5]]))
    model.out.bias = nn.Parameter(torch.tensor([0.3]))

# Predict
input_tensor = torch.tensor([[0.5, -0.3]])
output = model(input_tensor)

print("PyTorch Output:", output.item())
```

```
PyTorch Output: 0.5460000038146973
```