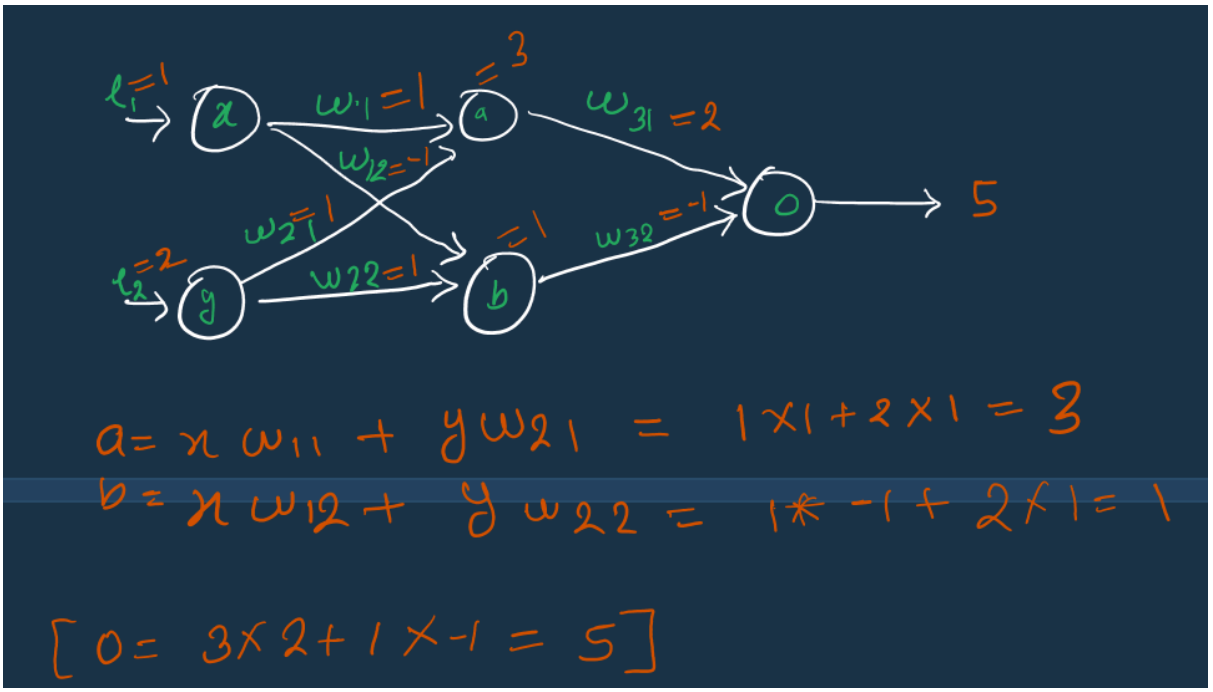
Assignment - 1

Forward Propagation

```python
import numpy as np

input_data = np.array([1, 2])

weights = {
    'node0': np.array([1, 1]),
    'node1': np.array([-1, 1]),
    'output': np.array([2, -1])
}

# Calculate the node0, node1 value with corsponding weights
node_0_value = (input_data * weights['node0']).sum()
node_1_value = (input_data * weights['node1']).sum()

hidden_layer_result = np.array([node_0_value, node_1_value])
print(hidden_layer_result) # [3 1]

output = (hidden_layer_result * weights['output']).sum()
print(output) # 5
```

Output of Forward Propagation



Backward Propagation

```python
import numpy as np
import time

# Learning Rate
LEARNING_RATE = 0.6

# Debug Code
DEBUG = False

# Acceptance Tharsold
THARSOLD = 0.23

# Time expention Log | Start
start = time.time()

# Activation Function: Sigmoid
def sigmoid(h):
```

```
    return (1 / (1 + np.exp(-h)))


# Error Total => E(t) = E_o1 + E_o2
def error_total(targets, outputs):
    error = 0
    for i in range(len(outputs)):
        error += ((targets[i] - outputs[i]) ** 2) / 2
    return error


# Update Weights
def weights_update(outputs, targets, hidden, weights,
weights_node):
    chain1 = (outputs - targets)
    chain2 = (outputs * (1 - outputs))
    chain3 = (hidden)

    retrive_weight = chain1 * chain2 * chain3

    new_weight = (weights[weights_node] - (LEARNING_RATE *
retrive_weight))
    weights[weights_node] = new_weight


# Input Feature data.
input_data = np.array([0.05, 0.10])

# Weights of individual edges.
weights = {
    "node0": np.array([0.15, 0.20]),
    "node1": np.array([0.25, 0.30]),
    "output0": np.array([0.40, 0.45]),
    "output1": np.array([0.50, 0.55])
}

# Considring some bias for prision
bias = {
    "b1": 0.35,
    "b2": 0.60
}
```

```
targets = np.array([0.01, 0.99])
epoch = 0

while True:
    epoch += 1
    h1_in = (input_data * weights["node0"]).sum() + bias["b1"]
    h1_out = sigmoid(h1_in)
    print(f"h1(in): {h1_in}\nh1(out): {h1_out}") if DEBUG else
None

    h2_in = (input_data * weights["node1"]).sum() + bias["b1"]
    h2_out = sigmoid(h2_in)
    print(f"h2(in): {h2_in}\nh2(out): {h2_out}") if DEBUG else
None

    hidden_layer_data = np.array([h1_out, h2_out])

    o1_in = (hidden_layer_data * weights["output0"]).sum() +
bias["b2"]
    o1_out = sigmoid(o1_in)
    print(f"o1(in): {o1_in}\no1(out): {o1_out}") if DEBUG else
None

    o2_in = (hidden_layer_data * weights["output1"]).sum() +
bias["b2"]
    o2_out = sigmoid(o2_in)
    print(f"o2(in): {o2_in}\no2(out): {o2_out}") if DEBUG else
None

    outputs = np.array([o1_out, o2_out])

    total_error = error_total(targets, outputs)
    print(total_error) if DEBUG else None # Total Error

    # Break when get optimal weights.
    if total_error < THARSOLD:
        break

    # Minimizing error by back propogating => Adjesting weights.
```

```
    weights_update(outputs, targets, hidden_layer_data, weights,
weights_node="output0")
    weights_update(outputs, targets, hidden_layer_data, weights,
weights_node="output1")
    weights_update(hidden_layer_data, targets, input_data,
weights, weights_node="node0")
    weights_update(hidden_layer_data, targets, input_data,
weights, weights_node="node1")
    print(weights) if DEBUG else None

# Time expention Log | End
end = time.time()

print(
f'''
Final Weights: {weights}
Total Epoch: {epoch}
Total Time: {(end-start) * 10**3}
'''
)

# Turn DEBUG = True for geting iteration information.
```

Output of Backward Propagation

```
Final Weights: {'node0': array([-0.11665147,  0.55439267]), 'node1': array([-0.01665147,  0.65439267]), 'output0': array([-2.78980305,  2.09814895]), 'output1': array([-2.68980305,  2.19814895])}
Total Epoch: 64
Total Time: 1.5053749084472656
```