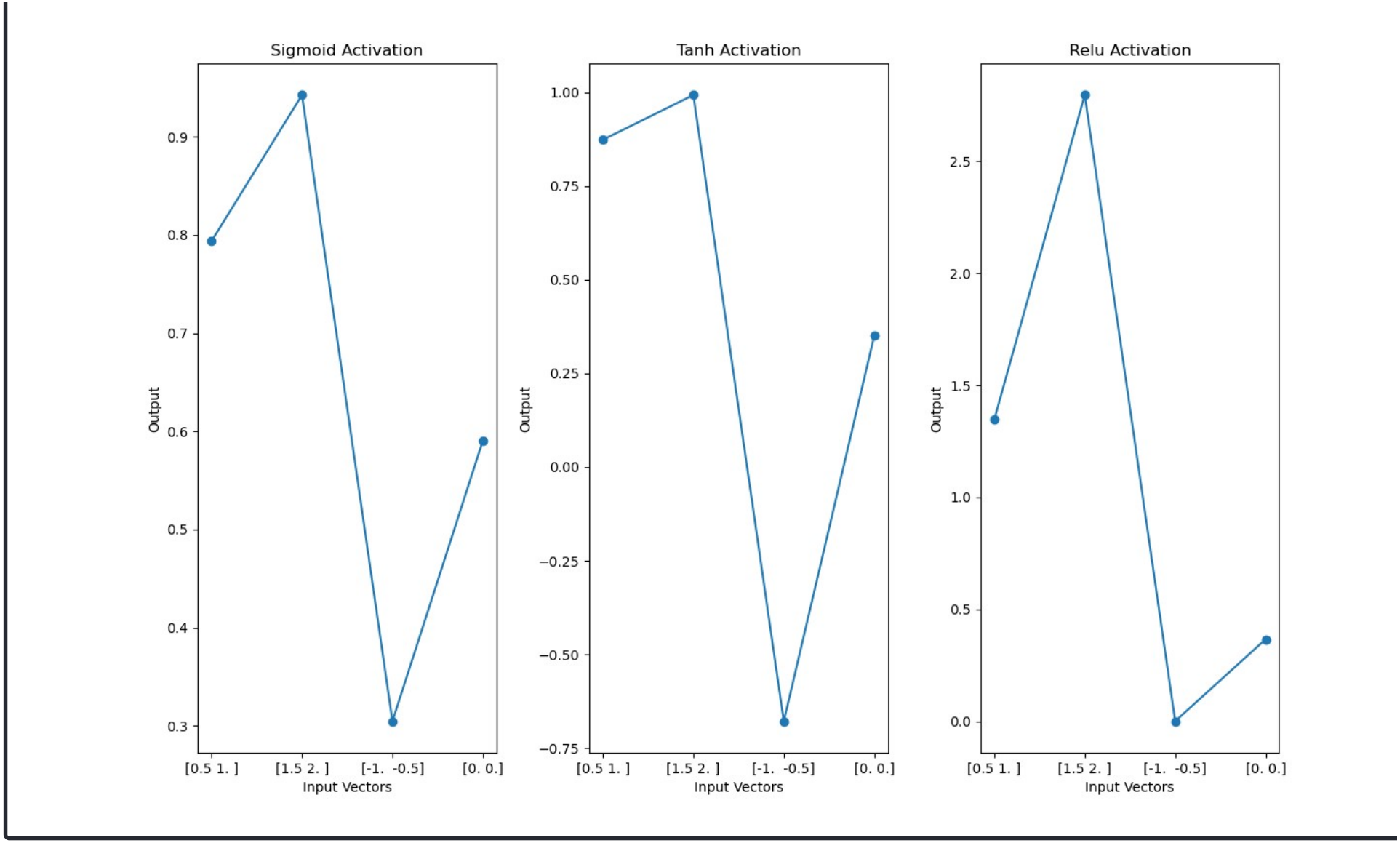
In [6]:

```
#program 1
#create  and implement a basic neuron model with the computational framework integrate essential elements
#like input node,weight parameters,bias and an activation function(included but not limited to sigmoid
#hyperbolic,tangent and relu)
#to construct a comprehensive representation of neuron then evaluate and observe how each activation
#function influences the network behavior and effectiveness in handling different types of data
import numpy as np
import matplotlib.pyplot as plt
class Neuron:
    def __init__(self, n_inputs):
        self.weights = np.random.rand(n_inputs)
        self.bias = np.random.rand(1)
    def activate(self, x, activation_function='sigmoid'):
        if activation_function == 'sigmoid':
            return self.sigmoid(x)
        elif activation_function == 'tanh':
            return self.tanh(x)
        elif activation_function == 'relu':
            return self.relu(x)
        else:
            raise ValueError("Unsupported activation function.")
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
    def tanh(self, x):
        return np.tanh(x)
    def relu(self, x):
```

```
            return np.maximum(0, x)
        def forward(self, inputs, activation_function='sigmoid'):
            linear_combination = np.dot(inputs, self.weights) + self.bias
            return self.activate(linear_combination, activation_function)
data = np.array([[0.5, 1.0], [1.5, 2.0], [-1.0, -0.5], [0.0, 0.0]])
neuron = Neuron(n_inputs=2)
activation_functions = ['sigmoid', 'tanh', 'relu']
results = {}
for activation in activation_functions:
    results[activation] = []
    for input_vector in data:
        output = neuron.forward(input_vector, activation_function=activation)
        results[activation].append(output)
for activation, outputs in results.items():
    print(f"Outputs with {activation} activation function:")
    print(outputs)
    print()
plt.figure(figsize=(12, 8))
for i, activation in enumerate(activation_functions):
    plt.subplot(1, 3, i + 1)
    plt.plot(range(len(data)), results[activation], marker='o')
    plt.title(f'{activation.capitalize()} Activation')
    plt.xticks(range(len(data)), [str(vec) for vec in data])
    plt.ylabel('Output')
    plt.xlabel('Input Vectors')
plt.tight_layout()
plt.show()
```

```
Outputs with sigmoid activation function:
[array([0.79367663]), array([0.94243877]), array([0.30462835]), array([0.59051676])]

Outputs with tanh activation function:
[array([0.87339823]), array([0.99256695]), array([-0.6779734]), array([0.35057752])]

Outputs with relu activation function:
[array([1.34723142]), array([2.79562167]), array([0.]), array([0.36610205])]
```


Sigmoid Activation
Output
0.9
0.8
0.7
0.6
0.5
0.4
0.3
[0.5 1.]
[1.5 2.]
[-1. -0.5]
[0. 0.]
Input Vectors
Tanh Activation
Output
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
[0.5 1.]
[1.5 2.]
[-1. -0.5]
[0. 0.]
Input Vectors
Relu Activation
Output
2.5
2.0
1.5
1.0
0.5
0.0
[0.5 1.]
[1.5 2.]
[-1. -0.5]
[0. 0.]
Input Vectors

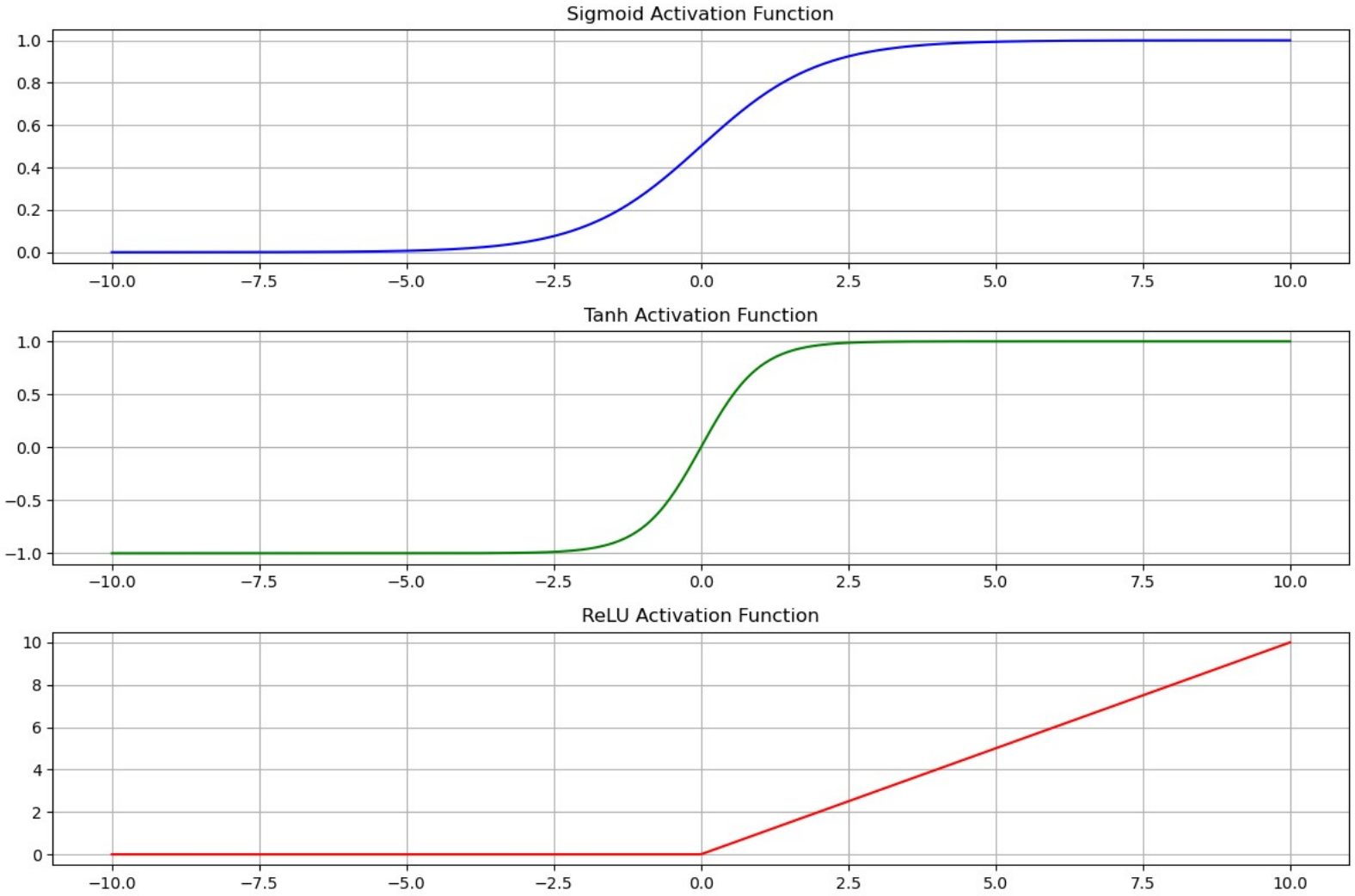
In [7]:

```
import numpy as np
import matplotlib.pyplot as plt

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

def tanh(x):
    return np.tanh(x)

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

class Neuron:
    def __init__(self, num_inputs, activation_function='sigmoid'):
        self.weights = np.random.randn(num_inputs)
        self.bias = np.random.randn()

        if activation_function == 'sigmoid':
            self.activation_function = sigmoid
        elif activation_function == 'tanh':
            self.activation_function = tanh
        elif activation_function == 'relu':
            self.activation_function = relu
        else:
            raise ValueError("Unknown activation function. Use 'sigmoid', 'tanh', or 'relu'.")

    def forward(self, inputs):
        z = np.dot(inputs, self.weights) + self.bias
```

```
        output = self.activation_function(z)
        return output

xor_inputs = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

xor_outputs = np.array([0, 1, 1, 0])

neuron_sigmoid = Neuron(num_inputs=2, activation_function='sigmoid')
neuron_tanh = Neuron(num_inputs=2, activation_function='tanh')
neuron_relu = Neuron(num_inputs=2, activation_function='relu')

print("Evaluating XOR Gate Behavior:")
for i, xor_input in enumerate(xor_inputs):
    output_sigmoid = neuron_sigmoid.forward(xor_input)
    output_tanh = neuron_tanh.forward(xor_input)
    output_relu = neuron_relu.forward(xor_input)

    print(f"Input: {xor_input} | Expected: {xor_outputs[i]} | "
          f"Sigmoid Output: {output_sigmoid:.4f}, "
          f"Tanh Output: {output_tanh:.4f}, "
          f"ReLU Output: {output_relu:.4f}")

x = np.linspace(-10, 10, 400)
```

```
y_sigmoid = sigmoid(x)
y_tanh = tanh(x)
y_relu = relu(x)

plt.figure(figsize=(12, 8))

plt.subplot(3, 1, 1)
plt.plot(x, y_sigmoid, label='Sigmoid', color='blue')
plt.title('Sigmoid Activation Function')
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(x, y_tanh, label='Tanh', color='green')
plt.title('Tanh Activation Function')
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(x, y_relu, label='ReLU', color='red')
plt.title('ReLU Activation Function')
plt.grid(True)

plt.tight_layout()
plt.show()
```

```
Evaluating XOR Gate Behavior:
Input: [0 0] | Expected: 0 | Sigmoid Output: 0.3533, Tanh Output: 0.6843, ReLU Output: 2.6981
Input: [0 1] | Expected: 1 | Sigmoid Output: 0.1756, Tanh Output: 0.7792, ReLU Output: 3.5241
Input: [1 0] | Expected: 1 | Sigmoid Output: 0.4109, Tanh Output: 0.8218, ReLU Output: 3.0782
Input: [1 1] | Expected: 0 | Sigmoid Output: 0.2139, Tanh Output: 0.8783, ReLU Output: 3.9043
```



In [ ]: