Task 1:

Implementation of Softmax Function:

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def loss_softmax(y_true, y_pred):
    epsilon = 1e-10
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.sum(y_true * np.log(y_pred))

z = np.array([[1.0, 2.0, 3.0]])
probs = softmax(z)

print("Input:", z[0])
print("Softmax:", probs[0])
print("Sum:", np.sum(probs))

y_true = np.array([0, 1, 0])
y_pred = probs[0]
loss = loss_softmax(y_true, y_pred)

print("\nTrue label:", y_true)
print("Prediction:", y_pred)
print("Loss:", loss)
```

```
Input: [1. 2. 3.]
Softmax: [0.09003057 0.24472847 0.66524096]
Sum: 0.9999999999999999

True label: [0 1 0]
Prediction: [0.09003057 0.24472847 0.66524096]
Loss: 1.4076059644443804
```

Test Cases for Softmax Function:

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def loss_softmax(y_true, y_pred):
    epsilon = 1e-10
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return -np.sum(y_true * np.log(y_pred))

def test_softmax():
    test_cases = [
        (np.array([[0, 0, 0]]), "All zeros"),
        (np.array([[1, 2, 3]]), "Simple case"),
        (np.array([[1000, 1000, 1000]]), "Large identical values"),
        (np.array([[-1000, -1000, -1000]]), "Small identical values"),
        (np.array([[1, 0, -1]]), "Mixed positive and negative")
    ]

    for i, (z, description) in enumerate(test_cases):
        print(f"Test {i + 1}: {description}")
        result = softmax(z)

        assert np.allclose(result.sum(axis=1), 1), f"Failed: Probabilities do not sum to 1 in {description}"
        assert np.all(result >= 0), f"Failed: Negative probabilities in {description}"

        print(f"  Input: {z[0]}")
        print(f"  Output: {result[0]}")
        print(f"  Sum: {result.sum():.10f}")
        print("  Passed.\n")

    print("All tests passed for softmax function.")
```

```
test_softmax()
```

```
Test 1: All zeros
  Input: [0 0 0]
  Output: [0.33333333 0.33333333 0.33333333]
  Sum: 1.0000000000
  Passed.

Test 2: Simple case
  Input: [1 2 3]
  Output: [0.09003057 0.24472847 0.66524096]
  Sum: 1.0000000000
  Passed.

Test 3: Large identical values
  Input: [1000 1000 1000]
  Output: [0.33333333 0.33333333 0.33333333]
  Sum: 1.0000000000
  Passed.

Test 4: Small identical values
  Input: [-1000 -1000 -1000]
  Output: [0.33333333 0.33333333 0.33333333]
  Sum: 1.0000000000
  Passed.

Test 5: Mixed positive and negative
  Input: [ 1  0 -1]
  Output: [0.66524096 0.24472847 0.09003057]
  Sum: 1.0000000000
  Passed.

All tests passed for softmax function.
```

Task 2:

Implementation of Categorical Log-Loss Function:

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def loss_softmax(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred + 1e-10))

y_true = np.array([0, 1, 0])
y_pred = np.array([0.1, 0.8, 0.1])
loss = loss_softmax(y_true, y_pred)
expected = -np.log(0.8)

print("True:", y_true)
print("Pred:", y_pred)
print("Loss:", loss)
print("Expected:", expected)
print("Passed:", np.isclose(loss, expected))
```

```
True: [0 1 0]
Pred: [0.1 0.8 0.1]
Loss: 0.2231435511892097
Expected: 0.2231435513142097
Passed: True
```

Implemenetation of Cost Function:

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def loss_softmax(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred + 1e-10))

def cost_softmax(X, y, W, b):
```

```
    n, d = X.shape
    z = np.dot(X, W) + b
    y_pred = softmax(z)
    return -np.sum(y * np.log(y_pred + 1e-10)) / n

X = np.array([[1, 2], [3, 4], [5, 6]])
y = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
W = np.random.randn(2, 3)
b = np.zeros(3)

cost = cost_softmax(X, y, W, b)
print("Cost:", cost)
```

```
Cost: 4.172775108476507
```

Task 3:

Testing the Cost Function:

```
import numpy as np

def softmax(z):
    exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)

def loss_softmax(y_true, y_pred):
    return -np.sum(y_true * np.log(y_pred + 1e-10))

def cost_softmax(X, y, W, b):
    n, d = X.shape
    z = np.dot(X, W) + b
    y_pred = softmax(z)
    return -np.sum(y * np.log(y_pred + 1e-10)) / n

X = np.array([[1, 2], [2, 3], [3, 4]])
y = np.array([[1, 0], [0, 1], [1, 0]])
W = np.array([[1, -1], [-1, 1]])
b = np.array([0, 0])

z = np.dot(X, W) + b
y_pred = softmax(z)
expected_cost = -np.sum(y * np.log(y_pred + 1e-10)) / X.shape[0]
calculated_cost = cost_softmax(X, y, W, b)

print("Expected:", expected_cost)
print("Calculated:", calculated_cost)
print("Passed:", np.isclose(calculated_cost, expected_cost))
```

```
Expected: 1.460261343779191
Calculated: 1.460261343779191
Passed: True
```