```python
"""
This code was originally written for CS 231n at Stanford University
(cs231n.stanford.edu).  It has been modified in various areas for use in the
ECE 239AS class at UCLA.  This includes the descriptions of what code to
implement as well as some slight potential changes in variable names to be
consistent with class nomenclature.  We thank Justin Johnson & Serena Yeung for
permission to use this code.  To see the original version, please visit
cs231n.stanford.edu.
"""

from nndl.layers import *
from utils.gradient_check import eval_numerical_gradient, eval_numerical_gradient_array
from nndl.layer_utils import affine_relu_forward, affine_relu_backward
from nndl.fc_net import FullyConnectedNet

def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))

def affine_forward_test():
    # Test the affine_forward function

    num_inputs = 2
    input_shape = (4, 5, 6)
    output_dim = 3

    input_size = num_inputs * np.prod(input_shape)
    weight_size = output_dim * np.prod(input_shape)

    x = np.linspace(-0.1, 0.5, num=input_size).reshape(num_inputs, *input_shape)
    w = np.linspace(-0.2, 0.3, num=weight_size).reshape(np.prod(input_shape), output_dim)
    b = np.linspace(-0.3, 0.1, num=output_dim)

    out, _ = affine_forward(x, w, b)
    correct_out = np.array([[ 1.49834967,  1.70660132,  1.91485297],
                            [ 3.25553199,  3.5141327,   3.77273342]])

    # Compare your output with ours. The error should be around 1e-9.
    print('If affine_forward function is working, difference should be less than 1e-9:')
    print('difference: {}'.format(rel_error(out, correct_out)))

def affine_backward_test():
    # Test the affine_backward function

    x = np.random.randn(10, 2, 3)
    w = np.random.randn(6, 5)
    b = np.random.randn(5)
    dout = np.random.randn(10, 5)

    dx_num = eval_numerical_gradient_array(lambda x: affine_forward(x, w, b)[0], x, dout)
    dw_num = eval_numerical_gradient_array(lambda w: affine_forward(x, w, b)[0], w, dout)
    db_num = eval_numerical_gradient_array(lambda b: affine_forward(x, w, b)[0], b, dout)

    _, cache = affine_forward(x, w, b)
    dx, dw, db = affine_backward(dout, cache)

    # The error should be around 1e-10
    print('If affine_backward is working, error should be less than 1e-9::')
    print('dx error: {}'.format(rel_error(dx_num, dx)))
    print('dw error: {}'.format(rel_error(dw_num, dw)))
    print('db error: {}'.format(rel_error(db_num, db)))

def relu_forward_test():
    # Test the relu_forward function

    x = np.linspace(-0.5, 0.5, num=12).reshape(3, 4)
```

```python
    out, _ = relu_forward(x)
    correct_out = np.array([[ 0.,          0.,          0.,          0.,         ],
                            [ 0.,          0.,          0.04545455,  0.13636364,],
                            [ 0.22727273,  0.31818182,  0.40909091,  0.5,       ]])

    # Compare your output with ours. The error should be around 1e-8
    print('If relu_forward function is working, difference should be around 1e-8:')
    print('difference: {}'.format(rel_error(out, correct_out)))

def relu_backward_test():
    x = np.random.randn(10, 10)
    dout = np.random.randn(*x.shape)

    dx_num = eval_numerical_gradient_array(lambda x: relu_forward(x)[0], x, dout)

    _, cache = relu_forward(x)
    dx = relu_backward(dout, cache)

    # The error should be around 1e-12
    print('If relu_forward function is working, error should be less than 1e-9:')
    print('dx error: {}'.format(rel_error(dx_num, dx)))

def affine_relu_test():

    x = np.random.randn(2, 3, 4)
    w = np.random.randn(12, 10)
    b = np.random.randn(10)
    dout = np.random.randn(2, 10)

    out, cache = affine_relu_forward(x, w, b)
    dx, dw, db = affine_relu_backward(dout, cache)

    dx_num = eval_numerical_gradient_array(lambda x: affine_relu_forward(x, w, b)[0], x, dout)
    dw_num = eval_numerical_gradient_array(lambda w: affine_relu_forward(x, w, b)[0], w, dout)
    db_num = eval_numerical_gradient_array(lambda b: affine_relu_forward(x, w, b)[0], b, dout)

    print('If affine_relu_forward and affine_relu_backward are working, error should be less
than 1e-9::')
    print('dx error: {}'.format(rel_error(dx_num, dx)))
    print('dw error: {}'.format(rel_error(dw_num, dw)))
    print('db error: {}'.format(rel_error(db_num, db)))

def fc_net_test():
    N, D, H1, H2, C = 2, 15, 20, 30, 10
    X = np.random.randn(N, D)
    y = np.random.randint(C, size=(N,))

    for reg in [0, 3.14]:
      print('Running check with reg = {}'.format(reg))
      model = FullyConnectedNet([H1, H2], input_dim=D, num_classes=C,
                                reg=reg, weight_scale=5e-2, dtype=np.float64)

      loss, grads = model.loss(X, y)
      print('Initial loss: {}'.format(loss))

      for name in sorted(grads):
        f = lambda _: model.loss(X, y)[0]
        grad_num = eval_numerical_gradient(f, model.params[name], verbose=False, h=1e-5)
        print('{} relative error: {}'.format(name, rel_error(grad_num, grads[name])))
```