```
import numpy as np
import numpy.random as npr
from test_util import *
from funkyyak import grad
npr.seed(1)

def test_dot():
    def fun(x, y): return to_scalar(np.dot(x, y))

    mat1 = npr.randn(10, 11)
    mat2 = npr.randn(10, 11)
    vect1 = npr.randn(10)
    vect2 = npr.randn(11)
    vect3 = npr.randn(11)

    check_grads(fun, mat1, vect2)
    check_grads(fun, mat1, mat2.T)
    check_grads(fun, vect1, mat1)
    check_grads(fun, vect2, vect3)

def test_max():
    def fun(x): return to_scalar(np.max(x))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_sum_1():
    def fun(x): return to_scalar(np.sum(x))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)
```

```python

def test_sum_2():
    def fun(x): return to_scalar(np.sum(x, axis=0))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_sum_3():
    def fun(x): return to_scalar(np.sum(x, axis=0, keepdims=True))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_mean_1():
    def fun(x): return to_scalar(np.mean(x))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_mean_2():
    def fun(x): return to_scalar(np.mean(x, axis=0))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_mean_3():
    def fun(x): return to_scalar(np.mean(x, axis=0, keepdims=True))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    mat = npr.randn(10, 11)
    check_grads(fun, mat)
    check_grads(d_fun, mat)

def test_index_ints():
    A = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(x[3, 0, 1])
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_index_slice():
    A = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(x[::-1, 2:4, :])
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_index_lists():
    A = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(x[[0, 1, 2], :, :])
```

```python
        def fun(x): return to_scalar(x[[0, 1, 2], :, :])
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_index_mixed():
        A = npr.randn(5, 6, 4)
        def fun(x): return to_scalar(x[3, 2:, [1, 3]])
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_vector_slice():
        A = npr.randn(5)
        def fun(x): return to_scalar(x[2:4])
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_index_slice_fanout():
        A = npr.randn(5, 6, 4)
        def fun(x):
            y = x[::-1, 2:4, :]
            z = x[::-1, 3:5, :]
            return to_scalar(y + z)
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_index_multiple_slices():
        A = npr.randn(7)
        def fun(x):
            y = x[2:6]
            z = y[1:3]
            return to_scalar(z)
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_reshape_method():
        A = npr.randn(5, 6, 4)
        def fun(x): return to_scalar(x.reshape((5 * 4, 6)))
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

    def test_reshape_call():
        A = npr.randn(5, 6, 4)
        def fun(x): return to_scalar(np.reshape(x, (5 * 4, 6)))
        d_fun = lambda x : to_scalar(grad(fun)(x))
        check_grads(fun, A)
        check_grads(d_fun, A)

```

```python
def test_ravel_method():
    A = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(x.ravel())
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_ravel_call():
    A = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(np.ravel(x))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_concatenate_axis_0():
    A = npr.randn(5, 6, 4)
    B = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(np.concatenate((B, x, B)))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_concatenate_axis_1():
    A = npr.randn(5, 6, 4)
    B = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(np.concatenate((B, x, B), axis=1))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

def test_concatenate_axis_1_unnamed():
    """Tests whether you can specify the axis without saying "axis=1"."""
    A = npr.randn(5, 6, 4)
    B = npr.randn(5, 6, 4)
    def fun(x): return to_scalar(np.concatenate((B, x, B), 1))
    d_fun = lambda x : to_scalar(grad(fun)(x))
    check_grads(fun, A)
    check_grads(d_fun, A)

# TODO:
# squeeze, transpose, getitem
```