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

def arg_pairs():
    scalar = 2.0
    vector = npr.randn(6)
    mat = npr.randn(7, 6)
    mat2 = npr.randn(1, 6)
    allargs = [scalar, vector, mat, mat2]
    for arg1, arg2 in it.product(allargs, allargs):
        yield arg1, arg2

def test_mul():
    fun = lambda x, y : to_scalar(x * y)
    d_fun_0 = lambda x, y : to_scalar(grad(fun, 0)(x, y))
    d_fun_1 = lambda x, y : to_scalar(grad(fun, 1)(x, y))
    for arg1, arg2 in arg_pairs():
        check_grads(fun, arg1, arg2)
        check_grads(d_fun_0, arg1, arg2)
        check_grads(d_fun_1, arg1, arg2)

def test_add():
    fun = lambda x, y : to_scalar(x + y)
    d_fun_0 = lambda x, y : to_scalar(grad(fun, 0)(x, y))
    d_fun_1 = lambda x, y : to_scalar(grad(fun, 1)(x, y))
    for arg1, arg2 in arg_pairs():
        check_grads(fun, arg1, arg2)
        check_grads(d_fun_0, arg1, arg2)
        check_grads(d_fun_1, arg1, arg2)

```

```
34    def test_sub():
35        fun = lambda x, y : to_scalar(x - y)
36        d_fun_0 = lambda x, y : to_scalar(grad(fun, 0)(x, y))
37        d_fun_1 = lambda x, y : to_scalar(grad(fun, 1)(x, y))
38        for arg1, arg2 in arg_pairs():
39            check_grads(fun, arg1, arg2)
40            check_grads(d_fun_0, arg1, arg2)
41            check_grads(d_fun_1, arg1, arg2)
42
43    def test_div():
44        fun = lambda x, y : to_scalar(x / y)
45        d_fun_0 = lambda x, y : to_scalar(grad(fun, 0)(x, y))
46        d_fun_1 = lambda x, y : to_scalar(grad(fun, 1)(x, y))
47        make_gap_from_zero = lambda x : np.sqrt(x **2 + 0.5)
48        for arg1, arg2 in arg_pairs():
49            arg1 = make_gap_from_zero(arg1)
50            arg2 = make_gap_from_zero(arg2)
51            check_grads(fun, arg1, arg2)
52            check_grads(d_fun_0, arg1, arg2)
53            check_grads(d_fun_1, arg1, arg2)
54
55    def test_pow():
56        fun = lambda x, y : to_scalar(x ** y)
57        d_fun_0 = lambda x, y : to_scalar(grad(fun, 0)(x, y))
58        d_fun_1 = lambda x, y : to_scalar(grad(fun, 1)(x, y))
59        make_positive = lambda x : np.abs(x) + 1.1 # Numeric derivatives fail near
60    zerofor arg1, arg2 in arg_pairs():
61            arg1 = make_positive(arg1)
62            arg2 = np.round(arg2)
63            check_grads(fun, arg1, arg2)
64            check_grads(d_fun_0, arg1, arg2)
65            check_grads(d_fun_1, arg1, arg2)
```