凸分析与优化方法 HW3 代码报告

2100011025 王奕博

在逻辑上，第三题写在第二题之前，因为第三题搭起了一个DAG自动微分的框架，而第二题将这个框架应用到了一个ResNet模型中。因此，先说明第三题的代码。

此外，**这里的代码为了方便展示全部放在一个代码块里，而压缩包里的作业按照要求是用多文件的方式写出来的。**

第三题 自动微分

代码展示如下：

In [6]:

```
# Yibo Wang,2100011025,coe@pku,convex opt 23 spring.
import numpy as np
import math


class Add:
    def forward(a, b):
        return a + b

    def diff_a(a, b):
        return 1

    def diff_b(a, b):
        return 1


class Sub:
    def forward(a, b):
        return a - b

    def diff_a(a, b):
        return 1

    def diff_b(a, b):
        return -1


class Mul:
    def forward(a, b):
        return a * b

    def diff_a(a, b):
        return b.forward()

    def diff_b(a, b):
        return a.forward()


class Div:
    def forward(a, b):
        assert b != 0
        return a / b

    def diff_a(a, b):
        return 1 / b.forward()
```

```
    def diff_b(a, b):
        return -a.forward() / (b.forward() * b.forward())


class Pow:
    def forward(a, b):
        assert a >= 0
        return a ** b

    def diff_a(a, b):
        return b.forward() * (a.forward() ** (b.forward() - 1))

    def diff_b(a, b):
        return (a.forward() ** b.forward()) * math.log(a.forward())


class Log:
    def forward(a):
        return math.log(a)

    def diff(a):
        return 1 / a.forward()


def log(a):
    if isinstance(a, float) or isinstance(a, int):
        return math.log(a)
    return Node(a, 0, Log, False)


class Exp:
    def forward(a):
        return math.exp(a)

    def diff(a):
        return math.exp(a.forward())


def exp(a):
    if isinstance(a, float) or isinstance(a, int):
        return math.exp(a)
    return Node(a, 0, Exp, False)


class Sin:
    def forward(a):
        return math.sin(a)

    def diff(a):
        return math.cos(a.forward())


def sin(a):
    if isinstance(a, float) or isinstance(a, int):
        return math.sin(a)
    return Node(a, 0, Sin, False)


class Cos:
    def forward(a):
        return math.cos(a)

    def diff(a):
```

```
        return -math.sin(a.forward())


def cos(a):
    if isinstance(a, float) or isinstance(a, int):
        return math.cos(a)
    return Node(a, 0, Cos, False)


class Tan:
    def forward(a):
        return math.tan(a)

    def diff(a):
        return 1 / (math.cos(a.forward())) ** 2


def tan(a):
    if isinstance(a, float) or isinstance(a, int):
        return math.tan(a)
    return Node(a, 0, Tan, False)
```

这一部分定义了一些运算符类，它们并不是计算图上的节点，而是节点的一部分。这里，很多函数的名字是相同的，这在事后的向前、向后传播中很方便（相当于是多态了）

In [7]:

```
class Node:
    def __init__(self, a, b=0, op=None, binary=True):
        self.a = a
        self.b = b
        self.op = op
        self.result = None
        self.binary = binary

    def __add__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(self, Variable(x), Add)
        return Node(self, x, Add)

    def __radd__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(Variable(x), self, Add)
        return Node(x, self, Add)

    def __sub__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(self, Variable(x), Sub)
        return Node(self, x, Sub)

    def __rsub__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(Variable(x), self, Sub)
        return Node(x, self, Sub)

    def __mul__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(self, Variable(x), Mul)
        return Node(self, x, Mul)

    def __rmul__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(Variable(x), self, Mul)
        return Node(x, self, Mul)
```

```
    def __truediv__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(self, Variable(x), Div)
        return Node(self, x, Div)

    def __pow__(self, x):
        if isinstance(x, float) or isinstance(x, int):
            return Node(self, Variable(x), Pow)
        return Node(self, x, Pow)

    def forward(self):
        if self.result is not None:
            return self.result
        if self.binary:
            ans = self.op.forward(self.a.forward(), self.b.forward())
        else:
            ans = self.op.forward(self.a.forward())
        self.result = ans
        return ans

    def backward(self, chain=1):
        if self.binary:
            self.a.backward(chain * self.op.diff_a(self.a, self.b))
            self.b.backward(chain * self.op.diff_b(self.a, self.b))
        else:
            self.a.backward(chain * self.op.diff(self.a))
```

这个部分定义了node类，这是计算图DAG上的结点。前半部分重载了加减乘除的运算符（log等函数的重载放在上一部分了），这样，每当两个node被作用了一个operator时，它相当于在图上建立了新的结点，同时连接了两个运算的node。于是，只要写完运算的式子，运算图就自动的建好了。

后半部分有backward()和forward()两个函数，分别通过递归的方式实现了计算值和计算偏导数的操作。forward()存在回溯的操作，backward()则没有，它把偏导数的值存在"叶子结点"里了。

In [8]:

```
class Variable(Node):
    def __init__(self, value):
        self.value = value
        self.diff = 0

    def forward(self):
        return self.value

    def diff(self):
        return self.diff

    def backward(self, chain):
        self.diff += chain
```

这个部分定义了node的派生Variable类，加入了value功能。

In [9]:

```
def expression(x1, x2, x3):
    return (sin(x1 + 1.0) + cos(2.0 * x2)) * tan(log(x3)) + (
        sin(x2 + 1.0) + cos(2.0 * x1)
    ) * exp(1.0 + sin(x3))


def expression_x1(x1, x2, x3):
    return cos(x1 + 1.0) * tan(log(x3)) - 2.0 * sin(2.0 * x1) * exp(1.0 + sin(x3))
```

```python
def expression_x2(x1, x2, x3):
    return -2.0 * sin(2.0 * x2) * tan(log(x3)) + cos(x2 + 1.0) * exp(1.0 + sin(x3))


def expression_x3(x1, x2, x3):
    return (sin(x1 + 1.0) + cos(2.0 * x2)) / (x3 * cos(log(x3)) ** 2) + (
        sin(x2 + 1.0) + cos(2.0 * x1)
    ) * exp(1.0 + sin(x3)) * cos(x3)


if __name__ == "__main__":
    v1 = 5
    v2 = 2.6
    v3 = 3.0

    x1 = Variable(v1)
    x2 = Variable(v2)
    x3 = Variable(v3)

    f = expression(x1, x2, x3)
    f_x1 = expression_x1(x1, x2, x3)
    f_x2 = expression_x2(x1, x2, x3)
    f_x3 = expression_x3(x1, x2, x3)
    f_value = f.forward()
    print(f"f = {f_value}")
    f.backward()
    print("-------automatic differential test------------")
    print(f"df/dx1 = {x1.diff}")
    print(f"df/dx2 = {x2.diff}")
    print(f"df/dx3 = {x3.diff}")
    print("-------mathematically derivation test---------")
    print(f"df/dx1 = {f_x1.forward()}")
    print(f"df/dx2 = {f_x2.forward()}")
    print(f"df/dx3 = {f_x3.forward()}")

    t = 1e-7

    print("----------test:numerically derivation----------")
    x1_test = Variable(v1 + t)
    x2_test = Variable(v2)
    x3_test = Variable(v3)
    f_test_1 = expression(x1_test, x2_test, x3_test)
    ans = (f_test_1.forward() - f_value) / t
    print(f"df/dx1 = {ans}")

    x1_test = Variable(v1)
    x2_test = Variable(v2 + t)
    x3_test = Variable(v3)
    f_test_2 = expression(x1_test, x2_test, x3_test)
    ans = (f_test_2.forward() - f_value) / t
    print(f"df/dx2 = {ans}")

    x1_test = Variable(v1)
    x2_test = Variable(v2)
    x3_test = Variable(v3 + t)
    f_test_3 = expression(x1_test, x2_test, x3_test)
    ans = (f_test_3.forward() - f_value) / t
    print(f"df/dx2 = {ans}")

    print(
        "The correctness can be verified,",
        "if the values calculated from the three ways above are all close.",
    )
```

```
f = -3.6414654941561286
-------automatic differential test------------
df/dx1 = 5.285913883736087
df/dx2 = 0.6525692560484719
df/dx3 = 4.276282846436922
-------mathematically derivation test---------
df/dx1 = 5.285913883736087
df/dx2 = 0.6525692560484719
df/dx3 = 4.276282846436922
----------test:numerically derivation----------
df/dx1 = 5.28591445281279
df/dx2 = 0.6525691453873605
df/dx2 = 4.276282687953881
The correctness can be verified, if the values calculated from the three ways above ar
e all close.
```

这个部分是验证的部分，分别用**计算图方式、求出偏导数表达式并带入方式、数值微分方式**三种办法求出了偏导数值，并用这种方式来验证了程序的正确性。

可以看出，三种方法计算的值是差不多的。