凸分析与优化方法 HW3 代码报告

2100011025 王奕博

第二题 resnet自动微分

代码展示：这里应用了第三题的框架，将其应用在一个简单的resnet cnn上。这里，求导的对象是一个卷积filter，于是求导的结果是一个小矩阵。

In [1]:

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

# to define all classes needed in automatic differential.
# any operator between two nodes will form an edge of DAG, and that's how the whole gr


class Add:
    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 Pow:
    def forward(a, b):
        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 Relu:
    def forward(a):
        if a >= 0:
            return a
        else:
            return 0

    def diff(a):
        if a.forward() >= 0:
```

```python
            return 1
        else:
            return 0


def relu(a):
    if isinstance(a, float) or isinstance(a, int):
        if a >= 0:
            return a
        else:
            return 0
    return Node(a, 0, Relu, False)


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 __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 __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))
```

```
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
```

这一部分是第三题的框架，详见第三题的代码报告。

In [2]:

```
# to define the RESNET. That is,
#        ____________________________
#       |                            |
#       |                            |add
#       |      conv      relu        |
# input_layer ------>layer----->layer----->output_layer
# my loss function V is MSE loss ||sample-output_layer||^2_F
# then calculate the derivation of MSE-loss with respect to filter(conv) matrix
```

这个注释展示了resnet的简单模型。之后的损失函数V采用了与sample矩阵的差的frobenius范数，即所有元素平方和。

In [3]:

```
class inChannel:
    def __init__(self, width, height, value: np.array):
        self.width = width
        self.height = height
        self.value = [
            [Variable(value[i][j]) for i in range(width)] for j in range(height)
        ]


class conv:
    def __init__(self, width, height, value: np.array):
        self.width = width
        self.height = height
        self.value = [
            [Variable(value[i][j]) for i in range(width)] for j in range(height)
        ]


class resNet:
    def __init__(self, width1, height1, width2, height2, value1, value2):
        self.inc = inChannel(width1, height1, value1)
        self.cv = conv(width2, height2, value2)
        self.width1 = width1
        self.width2 = width2
        self.height1 = height1
        self.height2 = height2
        self.padding_width = (width1 - width2) // 2
        self.padding_height = (height1 - height2) // 2
        self.outChannel = None
        self.loss = 0

    def convolution(self, x, y):
        temp = Variable(0)
```

```
        for i in range(self.width2):
            for j in range(self.height2):
                if (
                    0 <= x + i - self.padding_width < self.width1
                    and 0 <= y + j - self.padding_height < self.height1
                ):
                    temp = (
                        temp
                        + self.inc.value[x + i - self.padding_width][
                            y + j - self.padding_height
                        ]
                        * self.cv.value[i][j]
                    )
        return temp

    def connect(self):
        self.outChannel = [
            [
                relu(self.convolution(i, j)) + self.inc.value[i][j]
                for i in range(self.width1)
            ]
            for j in range(self.height1)
        ]

    def MSE(self, sample: np.array):
        temp = Variable(0)
        for i in range(self.width1):
            for j in range(self.height1):
                # print(self.outChannel[i][j].forward())
                temp = temp + (self.outChannel[i][j] + sample[i][j]) * (
                    self.outChannel[i][j] + sample[i][j]
                )
        self.loss = temp
        return temp

    def start(self, sample):
        self.connect()
        self.MSE(sample)

    def gradient(self, i, j):
        return self.cv.value[i][j].diff
```

这个部分展示了小的resnet模型，输入层加入了padding以使得通过卷积运算后还能保持同样的尺寸。卷积层filter是优化的目标，也是V的自变量。之后再经过relu函数的激活得到了输出层，输出层再加上输入层就得到了结果，这也是resnet的feature。

这里的图链接和第三题是相似的，只是简单的连了个图。

In [4]:

```
if __name__ == "__main__":
    np.random.seed(1919810)

    # size setting

    width1 = height1 = 10
    width2 = height2 = 3

    # initial value setting.I just randomize them.

    inputInit = np.random.normal(0, 1, (width1, height1))
    convInit = np.random.normal(0, 1, (width2, height2))
    sampleInit = np.random.normal(0, 1, (width1, height1))

    # BP to calculate the value and the derivation
```

```python
resnet = resNet(width1, height1, width2, height2, inputInit, convInit)
resnet.start(sampleInit)
resnetLoss = resnet.loss.forward()

print("MSE loss =", resnetLoss)

resnet.loss.backward()
resnetGradient = [
    [resnet.gradient(i, j) for i in range(width2)] for j in range(height2)
]
gradientMatrix = np.array(resnetGradient)

print("gradient matrix:")
print(gradientMatrix)

# using [f(X+a*t)-f(X)]/t = <f'(X),a> tp test the program

gradient_test = np.random.normal(5, 1, (width2, height2))
t = 0.000001

resnetTest = resNet(
    width1, height1, width2, height2, inputInit, convInit + t * gradient_test
)
resnetTest.start(sampleInit)
resnetLossTest = resnetTest.loss.forward()
numericalTest = (resnetLossTest - resnetLoss) / t
print("---------------------------------------------")
print("numerically derivation:     ", numericalTest)

approx = np.sum(gradient_test * gradientMatrix)

print("first-order approx:         ", approx)
print("---------------------------------------------")
print(
    "The correctness can be verified,",
    "if the two numbers above are close enough.^_^",
)
```

```
MSE loss = 851.6435013317051
gradient matrix:
[[  31.84542421   27.88594078  177.42633147]
 [  -2.89826774  -47.95362952  -43.04123583]
 [-226.78507827  135.36814192  160.87924851]]
---------------------------------------------
numerically derivation:      1082.2033767681205
first-order approx:          1082.1961242353982
---------------------------------------------
The correctness can be verified, if the two numbers above are close enough.^_^
```

这里是测试，输入层、filter初始值以及sample值都是随机生成的。从后面可以看出计算图的结果和数值一阶近似的结果是相同的，这验证了程序的正确性。