一 实现的功能

基础功能：

1.比较运算（> < >= <= != ==）

结构支持：

Less Greater Leq Geq Equal Ineq类

测试代码：

Node\* x = placeholder("hhh");

Node\* y = placeholder("hahaha");

Node\* z= constant(3);

Node\* t = sub(x,y);

Node\* res =less(z,t);

cout << Run({{x,1},{y,2}}, \*t) <<endl;

cout << Run({{x,1},{y,2}}, \*res) << endl;

cout << Run({{x,1},{y,3}}, \*(res->grad(x))) << endl;

cout << Run({{x,1},{y,3}}, \*(res->grad(y))) << endl;

cout << Run({{x,1},{y,3}}, \*(res->grad(z))) << endl;

cout << Run( \*(res->grad(t))) << endl;

输出：

-1

0

0

0

0

0

2.Assert 和 Bind

测试代码：

auto x = placeholder();

auto y = placeholder();

auto t = add(x,y);

auto t1 = bind(t, assert(greater(y,x)));

auto res = bind(t, assert(less(y,add(x,constant(2)))));

cout << Run({{x,1},{y,2}}, \*res) << endl;//3

cout << Run({{x,1},{y,4}}, \*res) << endl;//fail

cout << Run({{x,1},{y,-1}}, \*t1) << endl;//fail

输出：

3

Assertion failed:0<=0

Assertion failed:0<=0

3.链式求导

结构支持：void getgrad()接口（在各个计算节点有相对应的处理）

测试代码：

Node\* x = placeholder();

Node\* y = placeholder();

Node\* z= constant(3);

Node\* t = add(x, mul(y,constant(0.5)));

Node\* res =cond(sin(x),tan(mul(t,z)),cos(mul(t,z)));//cond(a,b,c):return a>0?b:c

cout << Run({{x,pi/2},{y,pi}}, \*res) << endl;//0

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(x))) << endl;//3

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(y))) << endl;//1.5

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(x)->grad(x))) << endl;//0

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(x)->grad(y))) << endl;//0

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(y)->grad(x))) << endl;//0

cout << Run({{x,pi/2},{y,pi}}, \*(res->grad(y)->grad(y))) << endl;//0

cout << Run({{x,-pi/2},{y,pi}}, \*res) << endl;//1

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(x))) << endl;//0

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(y))) << endl;//0

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(x)->grad(x))) << endl;//-9

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(x)->grad(y))) << endl;//-4.5

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(y)->grad(x))) << endl;//-4.5

cout << Run({{x,-pi/2},{y,pi}}, \*(res->grad(y)->grad(y))) << endl;//-2.25

输出：

-3.67394e-016

3

1.5

-6.61287e-015

-3.30644e-015

-3.30644e-015

-1.65322e-015

1

0

0

-9

-4.5

-4.5

-2.25

4.牛顿迭代法（调用后执行的详细操作见func文件）

测试代码：

int n;

int a[101];

int i,j,k;

std::cin>>n;

auto x = placeholder();

Node \*y=Zero;

for(i=0;i<=n;i++)std::cin>>a[i];

for(i=n;i>=0;i--)y=add(mul(y,x),constant(a[i]));

float ans=solve(y,x,-1);

cout<<ans<<'\n';

二 拓展需求：

1. 实现了多维矩阵Tensor的运算图，支持所有操作节点，（由于已经包装在Node中，可以支持所有节点），并支持已经完成的各种运算。

注意矩阵运算可以匹配大小，不匹配时有报错的实现。有两种Reshape操作，可以自定义或按某个节点的形式reshape，可以在各个维度进行concat，改变矩阵大小。但是没有实现对于matmul和softmax的求导。（其他运算的求导应该都正常）

改变形状测试代码：

Node\* x=constant({{1,1,1,1,1,1,1,2},{2,4}});

Node\* y=transpose(x);

cout<<Run(\*y)<<endl;

Node\* x1=constant({{1,1,1,1,1,1,1,1},{2,4}});

Node\* y1=constant({{2,2,2,2,2,2,2,2},{2,4}});

Node\* z1=concat(x1,y1,2);

cout<<Run(\*z1)<<endl;

Node\* x2=constant({{1,1,1,1,1,1,1,1},{2,4}});

Node\* y2=reshape(x2, {1,2,1,1,2,1,1,2});

cout<<Run(\*y2)<<endl;

输出：

{{1,1}{1,1}{1,1}{1,2}}

{{{1,2}{1,2}{1,2}{1,2}}{{1,2}{1,2}{1,2}{1,2}}}

{{{{{{{{1,1}}}{{{1,1}}}}}}{{{{{{1,1}}}{{{1,1}}}}}}}}

1. 能够适应标量以及延伸到Tensor的数学运算：

（见MathsOp）

（Cos、Sin、Tan、Acos、Asin、Atan、Cosh、Sinh、Tanh、Acosh、Asinh、Atanh、Exp、Log、Log10、Exp2、Expm1、Log1p、Log2、Sqrt、Sqr、Erf、Erfc、Ceil、Floor、Abs、Sgn、Sigmoid、Relu、Softmax）

以Sigmoid为例：

测试代码：

Node\* x=constant({{1,1,1,1,1,1,1,2},{2,4}});

cout<<Run(\*sigmoid(x))<<'\n';

输出：

{{0.731059,0.731059,0.731059,0.731059}{0.731059,0.731059,0.731059,0.880797}}

1. 实现给parameter赋值的Assign运算

测试代码：

auto x = parameter(1);

auto y = parameter(2);

auto t1 = assign(x, y);

auto t2 = assign(y, mul(t1,y));

auto res = mul(mul(t2,x),y);

cout<<Run(\*res)<<'\n';

cout<<Run(\*x)<<'\n';

cout<<Run(\*y)<<'\n';

输出：

8

2

4

4.完整的求导功能

（对于没有意义的，用One，Zero，MinusOne等）

5.基于神经网络的手写数字识别

见learn.cpp, exam.cpp

由于没有矩阵自动求导，是手动求导后构造的运算图。由于程序基本没有优化，运算效率较低。另外由于没有掌握正确的训练方法（或者是隐藏的bug？），收敛速度慢，运行一晚上也只能达到92%正确率。

三 分析

实现的优点：

功能较全面，对原先的基础代码有了比较好的拓展，对增加功能有较好的开放性，没有写死，Tensor相较第一阶段几个组的尝试比较灵活

缺点：

一些显式构造比较长（因为不能对指针重载+-\*/和关系运算符，只能用函数代替）

文件名的形式比较奇怪，有用下划线的也有用驼峰式的，有首字母大写的也有首字母小写的（当然这和拿到代码时原来的文件名形式就很奇怪也有关系）