文件结构

\*.h是各种类型和函数声明的头文件，定义位于名称相同的\*.cpp中

stdops.h是把其他\*.h都包含进来的头文件集合

main.cpp中是示例程序，各示例需要单独取消注释才能编译运行

main\_old.cpp是第一阶段的main.cpp

learn.cpp是神经网络的训练程序

exam.cpp是神经网络的测试程序

train.tmp是保存训练的暂时结果的文件

rand.tmp是训练时打乱数据集，保存打乱后的标号的文件

train\_final.tmp是我们达到过的正确率最高的参数文件

t10k-images.idx3-ubyte

t10k-labels.idx1-ubyte

train-images.idx3-ubyte

train-labels.idx1-ubyte是mnist的手写数字识别数据集，以此为训练和测试数据

基类 Node.h

（0）特殊的编译预处理命令

#pragma GCC optimize("O2")

#pragma GCC optimize("fast-math") 开启编译器优化

#define float double 提高精度而无需改变程序的奇技淫巧

（1）Tensor类-对数据内容的包装，将数据拓宽成张量

私有成员：

vector<float> data; 以一维的形式存储每个元素

vector<int> shape; 每一维的大小

vector <int> num; 到当前维的一块中的元素个数 num[0]为元素总数（简化计算）

int size; 维数

公有成员：

Tensor(); 构造一个空的Tensor

Tensor(const float& list); 用float构造一个0维Tensor

Tensor(const std::vector<float>& list, const std::vector<int>& dims); 用list里的内容和dims中指定的各维大小构造Tensor

~Tensor(); 析构函数

void \_reshape(const std::vector<int>& list); 输入维度列表，重新成形

Tensor \_concat(const Tensor& r,int dim=0); 在给定的维度上对两个张量进行拼接

Tensor \_matmul(const Tensor& r); 矩阵乘法

void \_transpose(); 转置

float& at(const std::vector<int>& dims); 获取特定位置的元素

Tensor operator+(const Tensor& tr); 张量加法，支持Broadcast

Tensor operator-(const Tensor& tr); 张量减法，支持Broadcast

Tensor operator\*(const Tensor& tr); 张量逐项乘法，支持Broadcast

Tensor operator/ (const Tensor& tr); 张量逐项除法，支持Broadcast

const std::vector<int>& getshape()const; 返回Tensor::shape

void fill(float x); 用一个float填充整个张量

友元类和友元函数：tensor内部数据是隐私的，因此只开放给Node类和一些相关函数

（2）Node类-代表所有结点的抽象类

私有成员：

Tensor value; 当前结点的计算结果

map<Node \*,Node \*> grads; 当前结点对各结点求导所得的导函数结点

virtual Tensor calc(set<Node\*>& calced) = 0; 计算当前结点的值

公有成员：

void setvalue(const Tensor& v); 改变value

Tensor getvalue(); 获取value

float getfloat(const int& seq=0); 获取value中的第seq项

int get\_num(); 获取value.num[0]

void reshape(const std::vector<int>& list); 对Tensor中接口的调用

void transpose();对Tensor中接口的调用

virtual Node\* eval(set<Node\*>& calced) = 0; 对当前结点进行求值，根据calced中有没有当前结点决定是否重新计算当前结点的值，并且检查求值是否会出错。如果出错则返回nullptr，没出错则返回this

virtual void getgrad()=0; 计算当前结点对其它各结点的导数

const map<Node \*,Node \*>& grad(); 返回当前结点对其它各结点的导数表

Node \* grad(Node \*p); 返回当前结点对p的导数

（3）Nodeptr类-存储所有结点的地址，以便程序结束后释放，使用单例模式定义

私有成员：

set<Node\*> s; 存储地址的集合

~Nodeptr(); 析构函数，析构时将集合中所有地址delete掉

公有成员：

void add(Node \*p); 将地址加入集合中

static Nodeptr \_ins; 实例化

（4）其他声明

Tensor cos(Tensor ts);等 将数学函数重载为Tensor的版本

extern Constant\* const Zero;

extern Constant\* const One;

extern Constant\* const MinusOne; 声明外部常量，定义（在nodes.cpp中）为常量0，1和-1，用于防止求导时出现大量内容相同的常量结点

extern Nodeptr &ptrs; 声明外部变量ptrs，定义（在Node.cpp中）为\_ins，这样不用每次都写Nodeptr::\_ins这么长还难打的一串

nodes.h-继承自Node的基本结点

（1）Constant-常数结点

私有成员：

Tensor calc(set<Node\*>& calced); 计算，详细说明见Node.h，由于每个结点都重写了此函数，因此以后的结点不再列出

公有成员：

Constant (float number)：构造函数

Tensor calc (set<Node\*>& calced)

Node\* eval(set<Node\*>& calced)

Constant(const Tensor& value); 构造函数，以value为此常量的值

void getgrad(); 求导，详细说明见Node.h，由于每个结点都重写了此函数，因此以后的结点不再列出

Node\* eval(set<Node\*>& calced); 求值，详细说明见Node.h，由于每个结点都重写了此函数，因此以后的结点不再列出

（2）Placeholder-占位符结点

私有成员：

string name; 在Error情况下确定出错对象名称

公有成员：

Placeholder(string myname=""); 构造函数，初始化name

（3）Print-输出结点

私有成员：

Node\* dest; 要输出的结点

string info; 输出时额外输出的信息

公有成员：

Print(Node\* node,const string &\_info="");

（4）Parameter-参数

公有成员：

Parameter(const Tensor& number); 构造函数，用number赋初值

void set(const Tensor& number); 赋值

void add(const Tensor& number); 加上一个值

void multiply(const Tensor& number); 减去一个值

void divide(const Tensor& number); 乘上一个值

void minus(const Tensor& number); 除以一个值

（5）Assign-parameter赋值

私有成员：

Node\* src; 值的来源

Parameter \*tar; 赋值的对象

公有成员：

Assign(Parameter\* para,Node\* node); 构造函数

（6）其他声明

extern map<Parameter\*,Tensor> assign\_map;　外部变量，用于保存一次求值过程中需要赋值的parameter及应赋的值。

Constant\* constant(const Tensor &a); 以a为构造函数参数新建一个Constant结点，将地址加入ptrs（见Node.h），并返回地址。每个非抽象结点类都有相应的名字是类名首字母小写的名字的函数。以下将这些函数略去不写。

Operator.h-继承自Node的运算符基类

（1）Operator-所有二元运算结点的基类（抽象类）

私有成员：

Node\* const left;

Node\* const right; 二元运算的两个参数结点，一旦建立不得改变

公有成员：

Operator(Node\* node1, Node\* node2); 构造函数

Node\* getleft();

Node\* getright(); 获取left，right的接口

（2）Operator\_1-所有一元运算节点的基类（抽象类）

私有成员：

Node\* const op; 一元运算的参数，一旦建立不得改变

公有成员：

Operator\_1(Node\* \_op); 构造函数

Node\* getop(); 获取输入节点

Ops.h-基本运算

（1）Add加法结点

（2）Multiply乘法结点

（3）Minus减法结点

（4）Divide除法结点

（5）Power幂运算结点

（以上5个二元运算继承自Operator，相对基类无新增或改动的接口）

（6）Transpose转置节点

（一元运算，继承自Operator\_1，相对基类无新增或改动的接口）

（7）Concat拼接（继承自Operator）

私有成员：

const int dim; 表示在哪个维度拼接

公有成员：

Concat(Node\* node1, Node\* node2, int \_dim=0); 构造函数

int get\_dim(); 获取dim

（8）Reshape改变张量形态（继承自Operator\_1）

私有成员：

const std::vector<int> list; 表示新的形状

Tensor calc(set<Node\*>& calced) override

void getgrad()

（9）Reshape2 将left的张量形态变为right的形态

(继承自Operator，相对基类无新增或改动的接口）

（10）Matmul矩阵乘法节点

(继承自Operator，相对基类无新增或改动的接口）

Ex\_ops.h-基于一些拓展功能的节点

（1）Less 输入两个节点，判断是否一个小于另一个

(继承自Operator，相对基类无新增或改动的接口）

（2）Greater 同（1）>

（3）Leq <=

（4）Geq >=

（5）Equal ==

（6）Ineq !=

（7）Assert

(继承自Operator\_1，相对基类无新增或改动的接口）

（8）Bind 捆绑运算

(继承自Operator，相对基类无新增或改动的接口）

（9）Cond

Tensorflow中的一个功能，三个输入节点，co，ans1，ans2，根据co内情况的不同，控制数据的流向，是执行ans1还是执行ans2

私有成员：

Node \*const co,\*const ans1,\*const ans2;

公有成员：

Cond(Node\* node1, Node\* node2,Node\* node3); 构造函数

MathsOp.h 基于一些数学函数的节点

都是一元运算符，继承自Operator\_1，且没有新增或改动接口。所表示的函数都是能在cmath中找到或者在cmath中没有但我们觉得常用的。

Cos

Sin

Tan

Acos

Asin

Atan

Cosh

Sinh

Tanh

Acosh

Asinh

Atanh

Exp

Log

Log10

Exp2

Expm1

Log1p

Log2

Sqrt

Erf

Erfc

Ceil

Floor

Abs

（以上是在cmath中有的）

Sqr

Sgn

Sigmoid

Relu

Softmax：只对形状为n\*1的Tensor有效

（这5个是我们觉得常用的）

func.h 基本函数与运算符重载

用户层主要需要参考的：

Node\* Run(Node& des); 没有placeholder的赋值，进行des结点的运算

Node\* Run(map <Node\*, float>& initmap, Node& des); 执行placeholder的赋值，并进行des结点的运算（为什么别的地方都是Node\*就这里是Node&？虽然很奇怪但为了不改变接口只能保持原样）

ostream& operator<< (ostream& out, Node\* const nodeptr); 输出流重载：通过“输出Node指针”指令输出其指向结点的计算值（那我要输出Node\*的地址怎么办？显然第一阶段的开发者没有考虑这个问题，然而为了不改变接口只能保持原样）

ostream& operator<< (ostream& out, const Tensor& tensor); 输出流重载，用来输出Tensor

float solve(Node \*y, Node \*x, float x0, float eps); 牛顿法解方程y(x)=0，提供初始值为x0，返回令abs(y(x))<eps的x

stdops.h 头文件集合

就是把所有的头文件都包一遍，这样在.cpp中只要包这一个头文件就可以了