计算图大作业说明文档

如果源文件中的注释无法正常阅读，请使用utf-8或者与之兼容的编码。

0.文件结构

根目录下文件：

- tt.h：公用的头文件
- main.cpp：第一阶段代码
- main2.cpp：实现第二阶段的基础需求
- main3.cpp：实现基础需求3，牛顿迭代
- tensor_test.cpp：测试一些张量/session/矩阵乘法的基础功能
- MLP_train.cpp：用于训练MLP，并导出训练好的参数
- MLP_test.cpp：用于测试已有的参数，并做出一定的成果展示
- makefile：用于编译和测试
- Instruction.md/pdf：说明文档
- ExperimentReport.md/pdf：实验报告

include目录下文件：

- Graph.h：Graph类对应的头文件
- MLP.h：MLP类对应的头文件
- Opt.h：所有的结点类对应的头文件
- Session.h：Session类对应的头文件
- Tensor.h：Tensor类对应的头文件

graph目录下文件：

- Basic.cpp：实现计算图Graph类的基础操作
- NewOpt.cpp：实现向计算图中加入新节点的操作

mlp目录下文件：

- MLP.cpp：实现MLP类中需要的所有方法

opt目录下文件：

- Basic.cpp：实现所有结点类的公有操作
- Binary.cpp：实现四则运算对应的结点
- Cmp.cpp：实现比较运算符
- Unary.cpp：实现print运算符和数学函数运算符
- Special.cpp：实现其他的运算符结点，如矩阵乘法、损失函数等

session目录下文件：

- Basic.cpp：实现Session类中与计算相关的方法
- File.cpp：实现Session类中与文件和参数相关的方法
- Input.cpp：实现对oj测试中的输入格式的解析

tensor目录下文件：

- Basic.cpp：实现Tensor类的基础操作
- Calc0.cpp：实现张量的比较运算符和数学函数，如sigmoid,sin,cos,tanh,cosh,log,inverse等
- Calc1.cpp：实现张量的四则运算
- Calc2.cpp：实现矩阵转置、矩阵乘法、求交叉熵损失函数及其梯度

mnist目录下文件：

- mnist_digits_test_images.zip：存放mnist的测试集，使用前解压
- mnist_digits_training_images.zip：存放mnist的数据集，使用前解压
    - 注意MLP_train.cpp和MLP_test.cpp需要解压好的这两个数据集

parameter目录下文件：

- 以Parameter开头的文件：用于读取的参数文件

tests目录下文件：

- tensortest1.txt：用于暂存tensor_test.cpp结果的文件

picture目录以下文件：

- 三张图片：用于实验报告的图片

1.对象

- Tensor：计算图中的张量，实现了Tensor类所需的基本操作和张量的运算
- 计算图的结点：继承自Opt基类，同时Placeholder、Constant、Variable，以及其他各种运算符都视为计算图的结点
- Graph：记录结点名称和地址的对应关系，用于创建和维护各类结点，管理计算图使用的内存
- Session：实现Session和存储功能，同时由于一阶段代码将基础功能的实现放在Session中，第二阶段维持了这种做法
- MLP：实现神经网络的测试、学习，并计算正确率

2.运行环境

在Windows 10系统中经过测试，可以运行。

g++需要支持c++11，测试时用的是mingw的g++ 4.9.2

在Linux系统下没有测试。理论上可以编译运行。

makefile中的clean需要系统支持del命令。

3.对第一阶段代码的修改和修改的原因

首先，第一阶段代码的所有注释和调试用语句被删除。

出于开发方便性的考虑，将第一阶段的代码拆成了很多独立的文件。

修改了计算的基础逻辑，从递归计算改为正向传播。是为了将计算的过程从计算图中取出到session中，使其更加符合逻辑。

将结点中存储的float全部改为Tensor，是为了使用统一的方式应对oj的测试和MLP的需求。

4.接口和使用指南

- Graph

创建不同类型的运算符

```
Opt* creat_BinaryOpt(string, string, string);
//创建双目运算符
Opt* creat_UnaryOpt(string, string);
//创建单目运算符
Opt* creat_CondOpt(string, string, string);
//创建Cond条件判断结点
```

创建不同类型的结点

```
Opt* creat_PlaceHolderOpt(const vector<int>&);
//创建占位符结点
Opt* creat_ConstantOpt(Tensor);
//创建常量结点
Opt* creat_VariableOpt(const vector<int>&);
//创建变量结点
```

通常使用的接口

```
void new_Binary(string, string, string, string);
//新建双目运算符
void new_Unary(string, string, string);
//新建单目运算符
void new_Cond(string, string, string, string);
//新建Cond条件判断运算符
void new_PlaceHolder(string, const vector<int>&);
//新建占位符
void new_Constant(string, Tensor);
//新建常量
void new_Variable(string, const vector<int>&);
//新建变量
```

- Session 用于完成基础功能的函数（类似第一阶段）

```
void declare(const string&);
//一阶段，描述变量
void code(const string&);
//二阶段，定义运算结点
void run(const string&);
//三阶段，计算操作
```

正向传播和反向传播

```
bool BackPropagation(Opt*);
//反向传播
bool ForwardPropagation(string, const map<string,Tensor>&);
//正向传播
```

提供对于参数和结点值的接口

```
Tensor GetCurVal(string);
//获取该结点当前的值
void SetParameter(string,Tensor);
//修改参数为一个新的值
void AddParameter(string,Tensor);
//修改参数为现有值加一个新的值
void RandomInit();
//随机初始化参数
```

文件操作

```
void Export(string);
//向文件读入session中的参数的值
void Import(string);
//向文件输出session中的参数的值
```

- MLP

```
void MiniBatch(const vector<vector<float> >&, const vector<vector<float> >&);
//minibatch训练
int TestBatch(const vector<vector<float> >&, const vector<vector<float> >&);
//批量测试
bool Judge(const vector<float>&,const vector<float>&);
//判断两个one-hot是否表达同一个数
void SetLearningRate(float x);
//设置学习率
void SampleSum();
//将当前数据对应的梯度加入batchsum
void Init();
//初始化参数
void ParameterUpdate(int);
//用存储的梯度更新参数
int SingleTest(const vector<float>&, const vector<float>&);
//单独测试一组数据
```

- 举例：
  - 向图中添加结点，参数顺序为：结点名称，类型，前置结点的名称（可能有多个）

    ```
    graph->new_Unary("G","GRAD",cn);
    graph->new_Binary("gx","AT","G","X");
    graph->new_Binary("div","/",cn,"gx");
    graph->new_Binary("minus","-","X","div");
    graph->new_Binary("Asn","ASSIGN","X","minus");
    ```

  - 为Graph指定Session

    ```
    sess=new Session(graph);
    ```

- 另：MLP的使用依赖于结点的名称

- 必须有名为“Input”，“Answer”，“Grad”，“Loss”，“Output”的结点
- 对应的功能应当为：输入数据，输入数据的答案，梯度，损失函数，输出层