计算图大作业

本组工作介绍

于剑
周恩贤
张思源

2019 年 6 月 9 日

第一阶段

代码结构

- Graph：管理一张计算图，分配节点唯一ID

代码结构

- Graph：管理一张计算图，分配节点唯一ID
- Session：保存计算图当前变量取值状态并进行求值运算（即第二阶段拓展需求4）

代码结构

- Graph：管理一张计算图，分配节点唯一ID
- Session：保存计算图当前变量取值状态并进行求值运算（即第二阶段拓展需求4）
- Node：节点基类，依照节点类型由子类继承实现

代码结构

- Graph：管理一张计算图，分配节点唯一ID
- Session：保存计算图当前变量取值状态并进行求值运算（即第二阶段拓展需求4）
- Node：节点基类，依照节点类型由子类继承实现
- Data：计算图中使用的数据基类，由子类继承实现不同的数据类型。

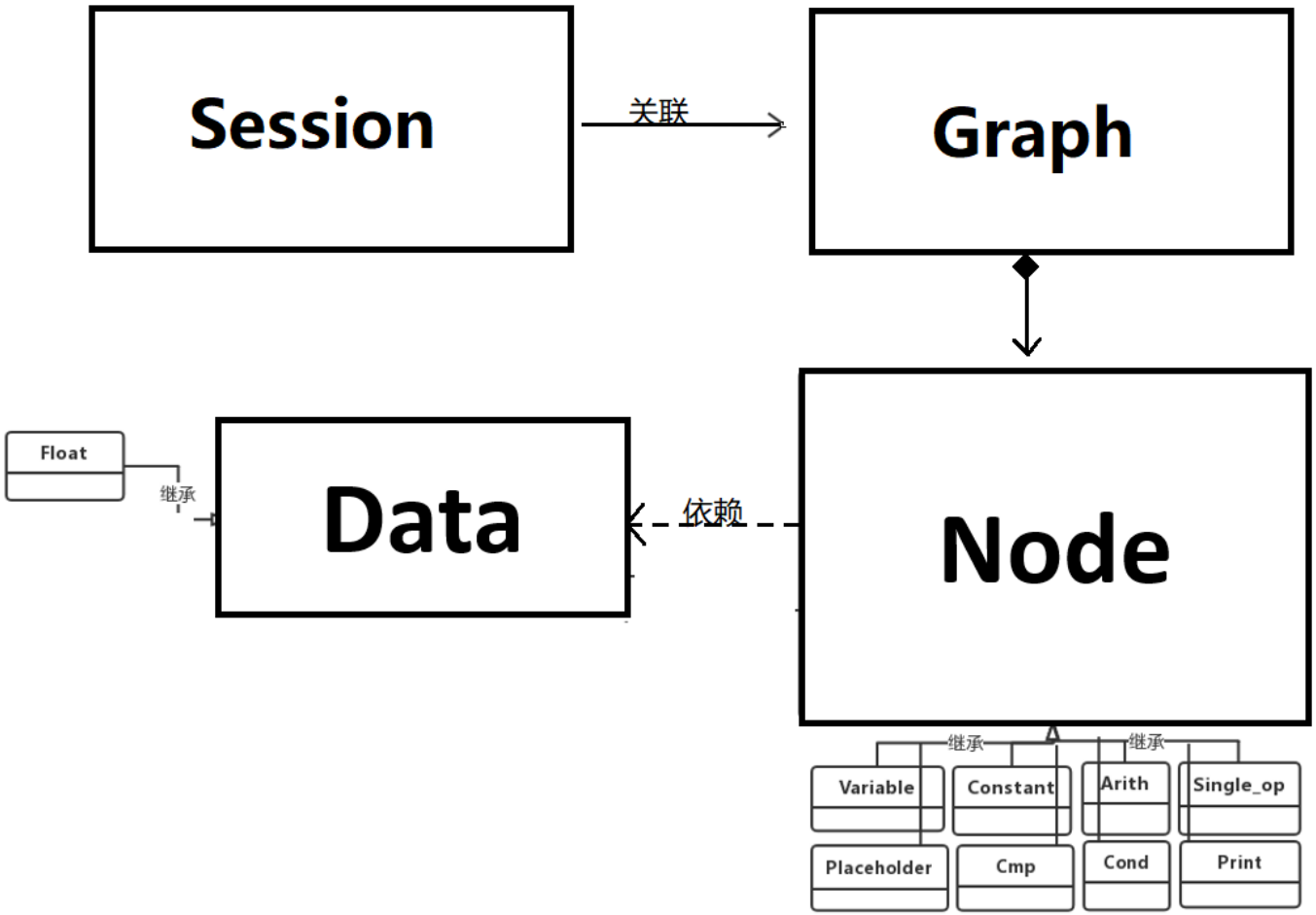
代码结构

- Graph：管理一张计算图，分配节点唯一ID
- Session：保存计算图当前变量取值状态并进行求值运算（即第二阶段拓展需求4）
- Node：节点基类，依照节点类型由子类继承实现
- Data：计算图中使用的数据基类，由子类继承实现不同的数据类型。
- Parser类：额外提供的工具，用于处理一个简单的计算图描述语言（即OJ测试的输入）

代码结构

- Graph：管理一张计算图，分配节点唯一ID
- Session：保存计算图当前变量取值状态并进行求值运算（即第二阶段拓展需求4）
- Node：节点基类，依照节点类型由子类继承实现
- Data：计算图中使用的数据基类，由子类继承实现不同的数据类型。
- Parser类：额外提供的工具，用于处理一个简单的计算图描述语言（即OJ测试的输入）
- Message类：为日志功能提供输出接口

UML图


Session
关联
Graph
Float
继承
Data
依赖
Node
继承
继承
Variable
Constant
Arith
Single_op
Placeholder
Cmp
Cond
Print

Data及其子类

- 派生出Float类，并提供各类运算接口

Data及其子类

- 派生出Float类，并提供各类运算接口
- 上层代码使用基类Data的智能指针管理运算数据

Data及其子类

- 派生出Float类，并提供各类运算接口
- 上层代码使用基类Data的智能指针管理运算数据
- 方便第二阶段扩展出其他数据类型，上层代码无需关心具体的数据类型。添加新类型、新运算时只需添加子类和运算接口

Node及其子类

- 混合使用模板方法模式与策略模式

Node及其子类

- 混合使用模板方法模式与策略模式
- 模板方法模式：将节点分为几大类，分别对应Node的派生类。

Node及其子类

- 混合使用模板方法模式与策略模式
- 模板方法模式：将节点分为几大类，分别对应Node的派生类。
- 双目算术运算(Arith类)，双目比较运算(Cmp类)，单目运算(Single_op)类等

Node及其子类

- 混合使用模板方法模式与策略模式
- 模板方法模式：将节点分为几大类，分别对应Node的派生类。
- 双目算术运算(Arith类)，双目比较运算(Cmp类)，单目运算(Single_op)类等
- 方便分辨不同逻辑的节点。

Node及其子类

- 子类中使用策略模式（以Arith节点为例）

Node及其子类

- 子类中使用策略模式（以Arith节点为例）
- 建立std::string到std::function对象的std::map，由”+”, ”-”, ”*”, ”/”映射到相应运算接口

Node及其子类

- 子类中使用策略模式（以Arith节点为例）
- 建立std::string到std::function对象的std::map，由”+”, ”-”, ”*”, ”/”映射到相应运算接口
- 创建Arith节点时：通过传入的运算符找到相应运算接口，并将std::function对象保存在节点中

Node及其子类

- 节点创建时需要较复杂的逻辑处理：

Node及其子类

- 节点创建时需要较复杂的逻辑处理：
- 判断前驱节点是否在同一张计算图中

Node及其子类

- 节点创建时需要较复杂的逻辑处理：
- 判断前驱节点是否在同一张计算图中
- 调用Graph类的接口转移节点所有权、分配ID

Node及其子类

- 节点创建时需要较复杂的逻辑处理：
- 判断前驱节点是否在同一张计算图中
- 调用Graph类的接口转移节点所有权、分配ID
- 使用工厂模式，Graph的相关接口以及节点类的构造函数对外界隐藏

额外功能-日志

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态
- 日志等级：debug, info, warning, error
  分别对应Message类的不同接口

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态
- 日志等级：debug, info, warning, error
  分别对应Message类的不同接口
- 日志输出：默认关联std::cerr，可通过Message类的接口调整

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态
- 日志等级：debug, info, warning, error
  分别对应Message类的不同接口
- 日志输出：默认关联std::cerr，可通过Message类的接口调整
- 可通过Message类的接口调整日志等级

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态
- 日志等级：debug, info, warning, error
  分别对应Message类的不同接口
- 日志输出：默认关联std::cerr，可通过Message类的接口调整
- 可通过Message类的接口调整日志等级
- 如：等级设为warning时，只有warning与error日志获得输出

日志功能与Message类

- Message类没有实例，全部方法和成员均为静态
- 日志等级：debug, info, warning, error
  分别对应Message类的不同接口
- 日志输出：默认关联std::cerr，可通过Message类的接口调整
- 可通过Message类的接口调整日志等级
- 如：等级设为warning时，只有warning与error日志获得输出
- 计算图的所有输出均通过Message类进行，方便统一管理

日志功能示例

这里给出了日志等级设定为3(warning)时的一个测试实例

```
@ubuntu:~/bigwork$ ./main < data5.input > data5.output
[error] Division by zero
[error] Division by zero
```

日志功能示例

这里给出了日志等级设定为1(debug)，并将输出流调整到文件输出流的一个测试实例

error.txt (~/bigwork) - gedit

Open

```
[debug] Parser::start() called, input string is f P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is ef V 4.3229
[debug] Variable::create() called
[debug] Parser::start() called, input string is l C -1.1788
[debug] Constant::create() called
[debug] Parser::start() called, input string is j P
[debug] Placeholder::create() called
[debug] Parser::start() called, input string is te C -4.6442
[debug] Constant::create() called
```

第二阶段开发

数据类变动

- 新增Data的派生类Tensor

数据类变动

- 新增Data的派生类Tensor
- 将Float视为shape=(1,)的Tensor，作为Tensor的派生类

数据类变动

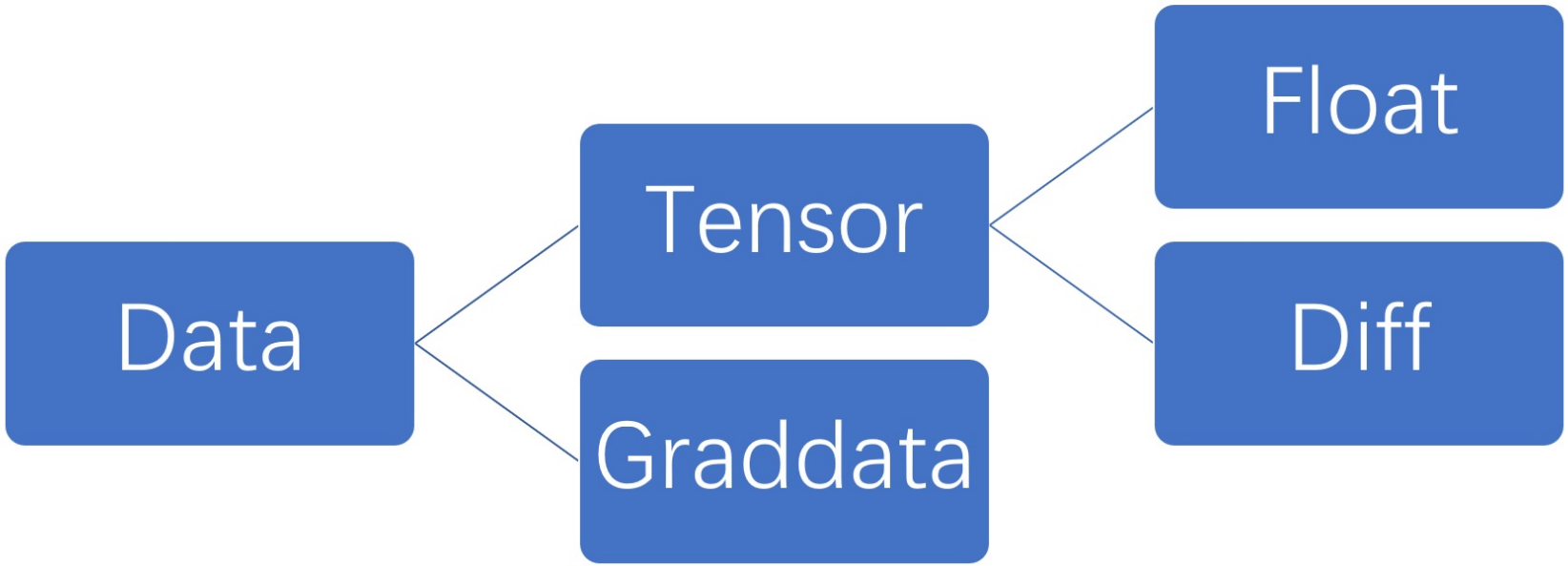
- 新增Data的派生类Tensor
- 将Float视为shape=(1,)的Tensor，作为Tensor的派生类
- 新增Tensor的派生类Diff，将shape划分为两部分，用于存储偏导数

数据类变动

- 新增Data的派生类Tensor
- 将Float视为shape=(1,)的Tensor，作为Tensor的派生类
- 新增Tensor的派生类Diff，将shape划分为两部分，用于存储偏导数
- 新增Data的派生类Graddata，作为Grad节点的求值结果，内含一组Diff数据

继承结构

更改后的数据类继承结构如下图所示



由于数据与结构分离，上层代码只关心Data基类的interface，因此上层逻辑无需修改

其他修改

- 新增若干节点派生类

其他修改

- 新增若干节点派生类
- Session的求值算法由记忆化dfs改为按拓扑序求值，方便实现求导（接口无需变动）

其他修改

- 新增若干节点派生类
- Session的求值算法由记忆化dfs改为按拓扑序求值，方便实现求导（接口无需变动）
- 实现自动求导后，调用框架容易实现其他功能

不足与未来的改进方向

调用不方便

- 使用数据时必须调用工厂方法
- 不能自动创建Constant节点

```
1 x = tf.Variable(1)
2 y = x + 1
```

Listing 1: TensorFlow示例

```
1 x = cg::Variable::create(&default_graph, Float::create(1));
2 y = x + cg::Constant::create(&default_graph, Float::create(1));
```

Listing 2: 我们的框架

其他不足之处

- 目前底层使用的标量类型固定为C++的double类型，不能自定义为不同浮点类型
- 许多需要用到的节点没有提供，如optimizer, reduce_sum等
- 大部分实现并未考虑线程安全，难以扩展出并行功能