计算图大作业实验报告

1.设计思路

- 基础逻辑
  - 计算图所需要有的功能是正向传播和反向传播
  - Graph不应当具备计算的逻辑，计算的逻辑应当归于Session
  - 使用结点的名字作为主要的控制手段
- Graph类
  - 作用是存储计算图的结构，并暂存计算图运行过程中的值和梯度
- Session类
  - 作用是提供正向反向传播的计算逻辑，提供参数对文件的接口
- MLP类
  - 作用是提供计算图的训练方法和测试方法，但是需要有对应名称的结点
- Tensor类
  - 取代第一阶段中的float类，让MLP和基础需求可以在相同的结构中运行

2.实现细节

- 使用类似拓扑排序的手段获取正向和反向传播所需要的顺序，使用时间戳来记录是否访问过
- MLP控制结点的手段是结点的名字，比如必须要有名为“Input”和“Answer”的结点来表示输入，必须要有名为“Grad”的结点来求梯度，必须要有名为“Loss”的结点来提供对损失函数的访问等。
- Session中添加了一个innate关键字，用于兼容第一阶段的代码和后续实验的代码。

3.功能综述

- 基础需求1：
  - 发现了第一阶段代码对于Print操作符的处理不恰当，并作出了对应的修改
- 基础需求2：
  - 实现了Assert和Bind运算符，可以通过oj的测试
- 基础需求3：
  - 实现了对于加法和乘法的求导，可以通过oj的测试
- 基础需求4：
  - 实现了题目要求的牛顿迭代，可以通过oj的测试
- 拓展需求1：
  - 实现了Assign运算符，并在牛顿迭代中有所运用
- 拓展需求2：
  - 完成了对-,/,sin,exp,log,tanh,Print,Assert,Bind等操作的求导
  - 对比较运算符的处理方式是直接报错
- 拓展需求3：
  - 实现了Tensor类，支持大多数的运算
  - 对二维矩阵实现了矩阵乘法和转置，并实现了矩阵乘法的反向传播
  - 不支持broadcast，不支持concat

    - 实现了矩阵乘法的结点和对应的求导功能
- 拓展需求4：
    - 支持session和parameter的绑定
    - 支持向文件导出parameter和从文件读入parameter
- 拓展需求6：
    - 实现了可以使用的MLP，并在mnist数据集上达到了90%+的正确率

4.测试和实验

- 基础功能

  在通过了提供的所有样例之后，很快就通过了oj上的所有测试，可以认为没有问题

  在这一过程中发现了原先第一阶段代码对print运算符处理不当，并做出了对应的修改

- Session的文件功能和Tensor的基础功能

  根目录下的tensor_test.cpp测试了session的文件功能和Tensor的一些基础功能，用来暂存的临时文件在tests文件夹下

- 其他的功能

  大部分的功能都和MLP直接相关，当MLP能给出理想结果的时候，可以认为这些功能都是正确的。

  MLP不需要的功能，比如concat、broadcast都被放弃了。

5.MLP的测试

- 数据集选用的是mnist，来自tensorflow 1.13.1，导出之后总共是55000张图片作为训练集，10000张图片作为测试集
- 选取的MLP模型具有一个300个结点的隐藏层，最终建图的方式如下
    - 首先建立输入节点，一个[784,1]的矩阵负责输入标准化之后的灰度，一个[10,1]的矩阵负责输入表示结果的one-hot向量

      ```
      graph->new_PlaceHolder("Input",{784,1});
      graph->new_PlaceHolder("Answer",{10,1});
      ```

    - 然后建立参数结点，包括输入层到隐藏层的权重矩阵和偏置向量，隐藏层到输出层的权重矩阵和偏置向量，这些也是神经网络的参数

      ```
      graph->new_Variable("Weight1",{300,784});
      graph->new_Variable("Bias1",{300,1});
      graph->new_Variable("Weight2",{10,300});
      graph->new_Variable("Bias2",{10,1});
      ```

    - 然后建立从输入层到隐藏层的计算节点，将输入层和隐藏层联系起来

      ```
      graph->new_Binary("t11","MATMUL","Weight1","Input");
      graph->new_Binary("t12","+","Bias1","t11");
      graph->new_Unary("x1","TANH","t12");
      ```

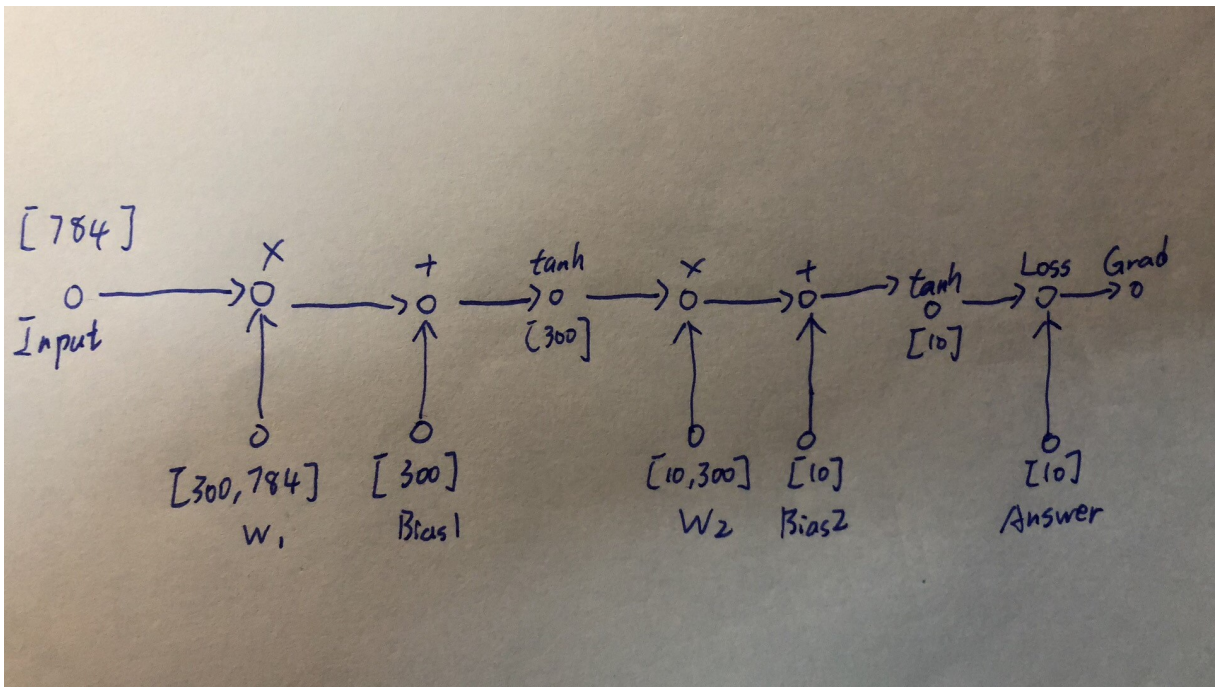
    - 然后建立从隐藏层到输出层的计算节点，将隐藏层和输出层联系起来

```
graph->new_Binary("t21","MATMUL","Weight2","x1");
graph->new_Binary("t22","+","Bias2","t21");
graph->new_Unary("Output","TANH","t22");
```

- 最后建立损失函数和梯度结点

```
graph->new_Binary("Loss","LOSS","Output","Answer");
graph->new_Unary("Grad","GRAD","Loss");
```

- 最终的结构可以用下图表示



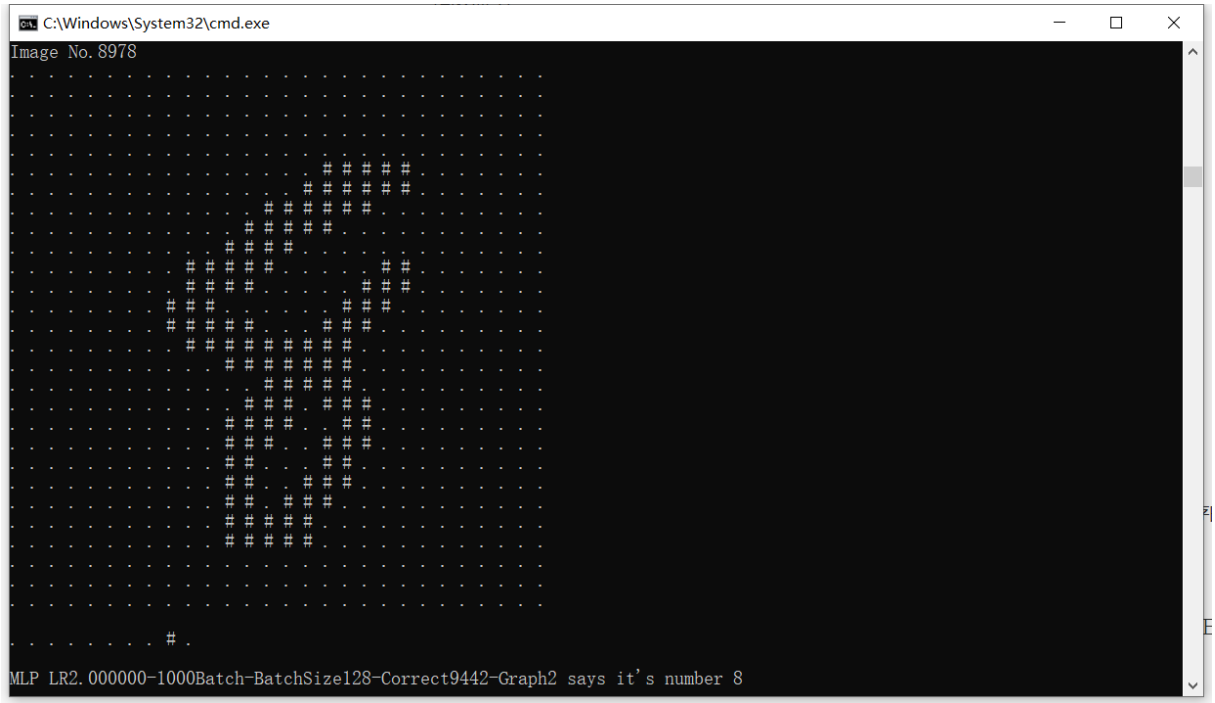
- 初始化的方式选用了均值为0的高斯分布，使用Xavier方法确定方差
- 损失函数选择了softmax之后的交叉熵，具体实现可以看代码
- 训练方法选用minibatch，一个batch包含了128组数据。总共训练1000组batch
- 在实验过程中只修改学习率这一参数，最终实验结果如下

学习率	训练结果在测试集上的正确率
0.01	0.8769
0.05	0.9062
0.35	0.9302
2	0.9442
10	0.9014
25	0.3191
50	0.2099

- 可以看出学习率为2时的效果最好，过大的学习率会导致损失函数在大范围上震荡，过小的学习率则无法在1000个batch中得到理想的结果；如果batch数量增加一至两个数量级，相信更小的学习率可以得到更好的结果
- 由于训练没有初始化随机种子，在随机函数机制相同的环境下，实验结果应当可以复现
- 对于训练出的参数，存放在parameter目录下，文件名说明了相应的参数
- 对于测试代码（MLP_train.cpp和MLP_test.cpp）还有一些解释
    - 运行这两个之前，需要将mnist文件夹下的数据集解压，解压的格式可以参考两个cpp文件中的文件读入部分
    - 这程序中包含了两个图的建图过程，一个采用了tanh作为激活函数，另一个采用了sigmoid作为激活函数，最终出于值域的考虑选择了tanh作为激活函数的图来测试
    - MLP_train.cpp开头的几个常数是可以调整的参数，在实验过程中只调整了学习率
    - MLP_train.cpp运行之后会在parameter目录下创造一个存储着参数的文件，MLP_test.cpp可以从中直接读取参数，并进行测试和展示结果。注意MLP_test.cpp读取的参数由文件名控制，比如

```
const string filename="LR2.000000-1000Batch-BatchSize128-Correct9442-Graph2";
...
nn->sess->Import("parameter/Parameter-"+filename+".txt");
```

    - MLP_test.cpp会读入参数，将对应的MLP在测试集上运行，并随机展示50个结果。本机测试结果举例如下：



6.优点

- 选择了合理的初始化方法，因而在较少的总训练量下，MLP就可以达到相当理想的正确率
- 用符合逻辑的方式实现了正向传播和反向传播，将计算过程与Graph分离
- 正向传播和反向传播的过程规避了map的大规模使用，从而做到了题目要求中的线性时间复杂度

7.缺点

- 矩阵乘法没有做优化，导致效率不足，训练组数不多也是对效率的妥协
- 大多数的操作都依赖结点的名字，对于使用者提出了一定的要求，同时也会有一定的不便
- 对于类内成员的封装直接继承了第一阶段代码的风格，严谨性略有不足
- Tensor的很多操作是对MLP特化的，功能还不够全面
- parameter的存取和读入依赖map的存储顺序，如果出现变化，可能导致参数不可用
- 为了实现真正线性的求导，利用了Graph类内的空间作为暂存，导致无法对高阶求导和多次求导做出拓展