作业说明

这是人工智能中的编程第二次作业的代码。

项目结构

项目有四个目录，一个 `main.cpp` 文件，一个 `CMakeLists.txt` ，和一个 `README.md` 。`include` 目录下包含项目需要的头文件， `src` 目录下包含源文件， `test` 目录下包含测试文件， `third_patrs` 包含第三方库。项目使用 `GoogleTest` 框架编写单元测试， `GoogleTest` 静态连接库和头文件在 `third_parts` 目录下，可以直接使用。项目使用 `cmake` 进行编译构建， `CMakeListst.txt` 在项目根目录下。

环境

项目在以下环境已经得到验证，可以正常运行：

操作系统	**Ubuntu24.04.1 LTS**
CUDA Toolkit	12.4
gcc/g++	13.2
CMake	3.28.3
Make	4.3

一般地，使用其他 Linux 发行版或者使用 WSL2 ，CMake 版本高于3.20，gcc/g++ 和 nvcc 版本支持 C++20或以上，是可以完成项目的编译构建的。如果使用 Windows 操作系统，一般使用 MinGW 也可以，MSVC 不确定能否编译构建本项目。

编译运行

进入项目根目录，依次执行下面的命令：

```
mkdir build && cd build
cmake .. # 默认 Debug 模式，如果需要使用 Release 模式，可以执行 cmake -DCMAKE_BUILD_TYPE=Release ..
make -j4 # 如果CPU核心数小于4,应该修改-j选项
./my_tensor # 将会打印 Hello, from my_tensor!
./tensor_test # 将会测试 Tensor 类的方法
./relu_test # 将会测试 Relu 类的正向传播和反向传播
./sigmoid_test # 将会测试 Sigmoid 类的正向传播和反响传播
./blas_test # 将会定义的若干个数学函数
./linear_test # 将会测试全连接层的正向传播和反向传播
./im2col_test # 将会测试 im2col 和 col2im 方法
./conv_test # 将会测试卷积层的正向传播和反向传播
./pooling_test # 将会测试池化层的正向传播和反向传播
./softmax_test # 将会测试 Softmax 类的正向传播和反向传播方法
./loss_with_softmax_test # 将会测试带有损失函数的 Softmax 层的正向传播和反向传播
```

项目代码解析

下面简单介绍一下项目的代码。

`Tensor` 类

`Tensor` 类定义在 `include/tensor.cuh` 文件中，在命名空间 `my_tensor` 下，包含属性 `data_`，`diff_`，`shape_` 和 `size_`。`Tensor` 类的对象可以从 `const std::vector<int>& shape` 构造，也可以进行复制构造和移动构造。可以进行拷贝和移动，其中 `data_` 和 `diff_` 是 `SyncedVector` 类型，用来同步 CPU 和 GPU 上的数据。

`Layer` 类

`Layer` 类是一个抽象类，不能拷贝和移动，有 `Forward` 和 `Backward` 两个纯虚函数。 `Layer` 从 `LayerParameter` 对象构造，在 `SetUp` 方法中设置一些相关的参数。其他的网络层都是继承这个抽象类。在本项目中，我们定义了 `Relu` 和 `Sigmoid` 作为激活层， `Linear` 作为全连接层， `Convolution` 作为卷积层， `Pooling` 作为池化层， `Softmax` 作为分类层， `LossWithSoftmax` 作为损失层。

作业相关函数的实现思路

下面来介绍作业相关函数的实现思路。

全连接层的正向和反向传播函数

参照讲义和课堂讲授的内容，我们在 `include/linear.cuh` 声明了全连接层的正向和反向传播函数，实现了 CPU 和 GPU 两个版本，这里我们主要介绍 GPU 上的实现，实现的细节在 `src/linear.cu` 中。在介绍这两个函数之前，我们先来看 `include/blas.cuh` 中声明的数学方法。

基于 `cuBLAS` 实现的一些数学方法

为了实现全连接层的正向和方向传播的方便，我们对一些数学方法进行了封装，具体实现基于 `cuBLAS` 库，代码在 `include/blas.cuh` 和 `src/blas.cu` 。

1. 首先我们封装了矩阵乘法，主要是 `matmul` , `transpose_matmul` , `matmul_transpose` , `transpose_matmul_transpose` ，分别表示两个矩阵相乘，左矩阵的转置乘以右矩阵，左矩阵乘以右矩阵的转置，左矩阵的转置乘以右矩阵的转置。在本项目中，我们只实现了单精度浮点数的相关方法，如果你使用其他模板参数，将会抛出异常。考虑到在卷积层的时候我们需要使用批量的矩阵乘法，并可能有广播机制的需要，我们在这里封装的是 `cuBLAS` 中的 `cublasSgemmBatched` 方法，用 `batch_count` 参数控制批量，默认为 1 ，也就是跟普通的矩阵乘法一样；同时用 `broadcast` 控制广播，为 1 的时候对第一个变量进行广播，为 2 的时候对第二个变量进行广播。
2. 然后我们封装了矩阵加向量的方法，这主要是为了全连接层中添加偏置的方便，主要的函数是 `add_row_vector` , `add_col_vector` 。
3. 我们还封装了矩阵和向量的求和方法，这主要是为了全连接层中偏置的梯度求解的方便，主要的函数是 `tensor_sum` , `row_sum` , `row_sum` 。

正向传播

全连接层的正向传播方法定义为 `Linear<T>::ForwardGPU` ，输入两个 `Tensor` 指针的数组，一个作为全连接层的输入，一个作为输出。由于我们已经封装了数学方法，正向传播的实现是比较简单的，我们首先用输入左乘权重矩阵，然后加上偏置向量，也就是如下的代码：

```
matmul(bottom[0]->GetGPUDataPtr(), weight_->GetGPUDataPtr(),
       top[0]->GetGPUDataPtr(), m, k, n);
add_col_vector(top[0]->GetGPUDataPtr(), bias_->GetGPUDataPtr(), m, n);
```

反向传播

全连接层的反向传播方法定义为 `Linear<T>::BackwardGPU` ，输入两个 `Tensor` 指针的数组，一个作为全连接层的输出，另一个作为输入。我们需要分别求解输入的梯度，权重矩阵的梯度和偏置的梯度。

对于输入的梯度，我们用输出梯度乘以权重矩阵的梯度，代码如下

```
matmul_transpose(top[0]->GetGPUDiffPtr(), this->GetWeight()->GetGPUDataPtr(),
                 bottom[0]->GetGPUDiffPtr(), m, n, k);
```

对于权重矩阵的梯度，我们用输入矩阵的转置乘以输出梯度，代码如下

```
transpose_matmul(bottom[0]->GetGPUDataPtr(), top[0]->GetGPUDiffPtr(),
                 this->GetWeight()->GetGPUDiffPtr(), k, m, n);
```

对于偏置的梯度，我们对输出梯度每一列求和，代码如下

```
col_sum(top[0]->GetGPUDiffPtr(), this->GetBias()->GetGPUDiffPtr(), m, n);
```

卷积层的正向和反向传播

参照讲义和课堂讲授的内容，我们在 `include/conv.cuh` 声明了全连接层的正向和反向传播函数，实现了 CPU 和 GPU 两个版本，这里我们主要介绍 GPU 上的实现，实现的细节在 `src/conv.cu` 中。这里我们使用 `im2col` 方法来实现卷积层。

`im2col` 和 `col2im` 的实现

为了加速卷积层的正向和反向传播，我们使用 `im2col` 和 `col2im` ，相关函数的声明在 `include/im2col.cuh` 中，实现在 `src/im2col.cu` 中。我们实现了 CPU 和 GPU 两个版本，同样地，这里我们只介绍 GPU 上的实现。

首先是 `im2col` 方法的实现，相关的函数为 `Im2col_GPU` ，我们定义了核函数 `Im2col_kernel` ，对于每一个卷积窗口进行转换，在 `Im2col_GPU` 中启动 `n * channels * width * height` 个核进行运算。

然后是 `col2im` 方法的实现，相关的函数为 `Col2im_GPU` ，我们定义了核函数 `Col2im_kernel` ，每一个线程负责输出的一个元素的修改，从而避免了原子操作，我们在 `Col2im_GPU` 中启动 `n * channels * im_size` 个核进行运算。

正向传播

卷积层的正向传播定义为 `Convolution<T>::ForwardGPU` 函数，我们首先调用 `Im2col_GPU` 方法将输入转换并写入 `col_cache_` ，然后用核左乘之，并注意设置批量为 `N` ，以及对第一个参数，也就是核进行广播，然后添加偏置。具体的代码如下

```
Im2col_GPU(batch_size_, bottom[0]->GetGPUDataPtr(), input_channels_, height_,
           width_, kernel_height_, kernel_width_,
           col_cache_->GetGPUDataPtr());
matmul(kernel_->GetGPUDataPtr(), col_cache_->GetGPUDataPtr(),
       top[0]->GetGPUDataPtr(), output_channels_,
       input_channels_ * kernel_size, im_size, batch_size_, 1);
add_row_vector(top[0]->GetGPUDataPtr(), bias_->GetGPUDataPtr(),
               output_channels_, im_size, batch_size_);
```

反向传播

卷积层的反向传播定义为 `Convolution<T>::BackwardGPU` 函数， 我们需要求解输入的梯度、核的梯度和偏置的梯度。

对于输入的梯度，我们先求 `col_cache_` 的梯度。我们用核的转置左乘输出的梯度，结果作为，注意设置批量为 `N` ，并对核进行广播，得到 `col_cache_` 的梯度。然后我们调用 `Col2im_GPU` 方法得到输入的梯度。主要的代码如下

```
transpose_matmul(kernel_->GetGPUDataPtr(), top[0]->GetGPUDiffPtr(),
                 col_cache_->GetGPUDiffPtr(), input_channels_ * kernel_size,
                 output_channels_, im_size, batch_size_, 1);
Col2im_GPU(batch_size_, col_cache_->GetGPUDiffPtr(), input_channels_, height_,
           width_, kernel_height_, kernel_width_, bottom[0]->GetGPUDiffPtr());
```

对于核的梯度，我们用输出梯度左乘 `col_cache_` 的转置，然后对列求和。主要的代码如下

```
matmul_transpose(top[0]->GetGPUDiffPtr(), col_cache_->GetGPUDataPtr(),
                 batch_kernel->GetGPUDiffPtr(), output_channels_, im_size,
                 input_channels_ * kernel_size, batch_size_);
col_sum(batch_kernel->GetGPUDiffPtr(), kernel_->GetGPUDiffPtr(), batch_size_,
        output_channels_ * input_channels_ * kernel_height_ * kernel_width_);
```

偏置的梯度求解有些复杂，我们首先对输出梯度批量按行求和，然后整体按列求和，得到偏置的梯度。主要的代码如下

```
row_sum(top[0]->GetGPUDiffPtr(), temp_diff, output_channels_, im_size,
        batch_size_);
col_sum(temp_diff, bias_->GetGPUDiffPtr(), batch_size_, output_channels_);
```

全连接层和卷积层的参数初始化

按照作业的要求和课堂的讲授，我们定义一个类 `Filler` ，负责全连接层和卷积层的参数初始化。项目只支持 GPU 上的初始化，随机数使用 `cuRAND` 生成。项目支持四种初始化方式，零初始化，常数初始化， `xavier` 初始化和 `He` 初始化。具体的代码在 `include/filler.cuh` 和 `src/filler.cu` 中。具体地，每一个 `Filler` 的子类定义一种初始化方式，重载 `Filler` 方法进行初始化。下面是 `cuRAND` 生成 `xavier` 初始化的随机数的核函数

```
__global__ static void XavierFillerKernel(float *data, float limit, int n) {
  CUDA_KERNEL_LOOP(i, n) {
    curandState state;
    curand_init(1234 + i, i, 0, &state);
    data[i] = curand_uniform(&state) * 2 * limit - limit;
  }
}
```

池化层的正向和反向传播

参照讲义和课堂讲授的内容，我们在 `include/pooling.cuh` 声明了全连接层的正向和反向传播函数，实现了 CPU 和 GPU 两个版本，这里我们主要介绍 GPU 上的实现，实现的细节在 `src/pooling.cu` 中。

正向传播

对每一个池化窗口，我们启动一个线程计算其最大值，输出到输出数据中，并记录其在输入数据的索引到 `mask` 张量中。

反向传播

先将输出梯度全部置为 0 ，然后根据 `mask` 数组的索引将对应位置的梯度置为 1 。这里我们直接调用 `thrust` 库的 `scatter` 方法，代码如下：

```
thrust::fill(bottom_diff.begin(), bottom_diff.end(), 0);
thrust::scatter(top_diff.begin(), top_diff.end(), mask_data.begin(),
                bottom_diff.begin());
```

`Softmax` 层的正向传播

按照作业的要求，我们只实现了正向传播，主要的代码思路与讲义的一致。首先，我们使用 `thrust::reduce_by_key` 进行分段归约，求出每一行的最大值，然后用 `thrust::transform` 将每一行减去最大值，并取对数，再进行一次分段归约，求出每一行的和，再对每一行除以这一行的和，得到最终结果。代码如下

```
thrust::device_vector<int> keys(batch_size_ * channels_);
int channels = channels_;
// generate key
thrust::transform(
    thrust::counting_iterator(0),
    thrust::counting_iterator(batch_size_ * channels_), keys.begin(),
    [channels] __device__(int i) -> int { return (i / channels) + 1; });
  thrust::device_vector<int> output_keys(batch_size_);
  thrust::device_vector<T> max_values(batch_size_);
T* max_ptr = RAW_PTR(max_values);
// compute row max element
thrust::reduce_by_key(keys.begin(), keys.end(), bottom_data.begin(),
                      output_keys.begin(), max_values.begin(),
                      thrust::equal_to<int>(), thrust::maximum<T>());
// substract the max element
thrust::transform(
    thrust::counting_iterator(0),
    thrust::counting_iterator(batch_size_ * channels_), bottom_data.begin(),
    top_data.begin(), [max_ptr, channels] __device__(int i, T val) -> T {
      return static_cast<T>(std::exp(val - max_ptr[i / channels]));
    });
// compute normalization factor
thrust::reduce_by_key(keys.begin(), keys.end(), top_data.begin(),
                      output_keys.begin(), max_values.begin(),
                      thrust::equal_to<int>(), thrust::plus<T>());
// noramlization
thrust::transform(thrust::counting_iterator(0),
                  thrust::counting_iterator(batch_size_ * channels_),
                  top_data.begin(), top_data.begin(),
                  [max_ptr, channels] __device__(int i, T val) -> T {
                    return static_cast<T>(val / max_ptr[i / channels]);
                  });
```

`Cross Entropy Loss` 的正向传播和反向传播

我们定义了一个类 `LossWithSoftmax` 实现基于 `Softmax` 的交叉熵损失。类 `LossWithSoftmax` 继承自 `Layer` ，重载了正向传播和反向传播，有一个 `Softmax` 实例。

正向传播

实现 `Cross Entropy Loss` 的函数是 `LossWithSoftmax<T>::ForwardGPU` ，输入是两个数组，第一个数组包括输入数据和标签，另一个数组为损失。首先调用 `LossWithSoftmax` 中 `Softmax` 实例的正向传播函数，得到 `softmax_top_` ，然后对每一个 `label` 对应的概率取对数再取相反数，得到一个数组，再进行 `reduce` 并取平均，得到损失。代码如下

```
thrust::transform(
    thrust::counting_iterator(0), thrust::counting_iterator(batch_size_),
    temp_data.begin(),
    [softmax_top_data, label_data, channels] __device__(int i) -> T {
      return -std::log(
          softmax_top_data[i * channels + static_cast<int>(label_data[i])]);
    });
top_data[0] =
    thrust::reduce(temp_data.begin(), temp_data.end()) / batch_size_;
```

反向传播

根据讲义和课堂的讲授，我们将输入数据的梯度设置为概率值，也就是上面求出的 `softmax_top_` ，然后每一行对应 `label` 的元素减去 1 ，得到输入数据的梯度值。代码如下

```
thrust::copy(softmax_top_data.begin(), softmax_top_data.end(),
             bottom_diff.begin());
thrust::for_each(
    thrust::counting_iterator(0), thrust::counting_iterator(batch_size_),
    [label_ptr, bottom_ptr, channels] __device__(int i) -> void {
      bottom_ptr[i * channels + static_cast<int>(label_ptr[i])] -= 1;
    });
```

测试

本项目使用 `GoogleTest` 框架进行单元测试，暂时未进行集成测试。测试代码在 `test` 目录下。跟本次作业相关的测试主要在 `test/linear-test.cu` ， `test/conv-test.cu` ， `test/pooling-test.cu` ， `test/softmax-test.cu` 和 `test/loss-with-softmax-test.cu` 中，目录下还有其他与作业不相关的测试文件，主要是开发过程中的单元测试以及上一次作业的测试代码，考虑项目的完整性，一并提交，亦可作为整体作业要求完成正确性的测试。

本次作业相关的测试方法，基本的思路是一致的，就是通过 CPU 上未经过优化的串行运算直接求得结果，与我们项目经过优化的方法或者 GPU 上的方法求得的结果进行比较，以检验正确性。测试的数据通过随机数生成，考虑到我们需要在 CPU 上进行检验运算，我们直接在 CPU 上生成随机数据，而没有使用 `cuRAND` 。测试样例的网络定义在 `test/json-test` 目录下。

运行项目与作业相关的测试，可以在构建好项目后，进入 `build` 目录并执行下列命令：

```
./linear_test # 将会测试全连接层的正向传播和反向传播
./conv_test # 将会测试卷积层的正向传播和反向传播
./pooling_test # 将会测试池化层的正向传播和反向传播
./softmax_test # 将会测试 Softmax 类的正向传播和反向传播方法
./loss_with_softmax_test # 将会测试带有损失函数的 Softmax 层的正向传播和反向传播
```

测试结果全部通过。 ~~（这是自然的，否则我也不会提交上去）~~

需要注意的是，浮点数运算可能会有误差，所以一些地方我设置的允许的偏差比较大，但还是有可能出现不匹配的可能。考虑到我们输入的数据量比较大，即使测试报错，显示误差超过了一定范围，但如果观察到真实值和期待值是接近的，应该认为还是正确的，因为错误的方法导致相近的结果的概率是较小的。以下是我运行相关测试的结果

```
(base) linyibo@linyibo-ThinkBook-14-G5-IRH:~/Code/my-tensor/build$ ./linear_test
[==========] Running 8 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from LinearCPUTest
[ RUN      ] LinearCPUTest.Linear_ForwardCPUTest
[       OK ] LinearCPUTest.Linear_ForwardCPUTest (2813 ms)
[ RUN      ] LinearCPUTest.Linear_BackwardBottomCPUTest
[       OK ] LinearCPUTest.Linear_BackwardBottomCPUTest (2751 ms)
[ RUN      ] LinearCPUTest.Linear_BackwardWeightCPUTest
[       OK ] LinearCPUTest.Linear_BackwardWeightCPUTest (2775 ms)
[ RUN      ] LinearCPUTest.Linear_BackwardBiasCPUTest
[       OK ] LinearCPUTest.Linear_BackwardBiasCPUTest (2684 ms)
[----------] 4 tests from LinearCPUTest (11024 ms total)

[----------] 4 tests from LinearGPUTest
[ RUN      ] LinearGPUTest.Linear_ForwardGPUTest
[       OK ] LinearGPUTest.Linear_ForwardGPUTest (885 ms)
[ RUN      ] LinearGPUTest.Linear_BackwardBottomGPUTest
[       OK ] LinearGPUTest.Linear_BackwardBottomGPUTest (465 ms)
[ RUN      ] LinearGPUTest.Linear_BackwardWeightGPUTest
[       OK ] LinearGPUTest.Linear_BackwardWeightGPUTest (558 ms)
[ RUN      ] LinearGPUTest.Linear_BackwardBiasGPUTest
[       OK ] LinearGPUTest.Linear_BackwardBiasGPUTest (85 ms)
[----------] 4 tests from LinearGPUTest (1994 ms total)

[----------] Global test environment tear-down
[==========] 8 tests from 2 test suites ran. (13019 ms total)
[  PASSED  ] 8 tests.
```

```
(base) linyibo@linyibo-ThinkBook-14-G5-IRH:~/Code/my-tensor/build$ ./conv_test
[==========] Running 8 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from ConvolutionCPUTest
[ RUN      ] ConvolutionCPUTest.ForwardTest
[       OK ] ConvolutionCPUTest.ForwardTest (467 ms)
[ RUN      ] ConvolutionCPUTest.BackwardBottomTest
[       OK ] ConvolutionCPUTest.BackwardBottomTest (1042 ms)
[ RUN      ] ConvolutionCPUTest.BackwardKernelTest
[       OK ] ConvolutionCPUTest.BackwardKernelTest (1035 ms)
[ RUN      ] ConvolutionCPUTest.BackwardBiasTest
[       OK ] ConvolutionCPUTest.BackwardBiasTest (998 ms)
[----------] 4 tests from ConvolutionCPUTest (3544 ms total)

[----------] 4 tests from ConvolutionGPUTest
[ RUN      ] ConvolutionGPUTest.ForwardTest
[       OK ] ConvolutionGPUTest.ForwardTest (1115 ms)
[ RUN      ] ConvolutionGPUTest.BackwardBottomTest
[       OK ] ConvolutionGPUTest.BackwardBottomTest (643 ms)
[ RUN      ] ConvolutionGPUTest.BackwardKernelTest
[       OK ] ConvolutionGPUTest.BackwardKernelTest (88 ms)
[ RUN      ] ConvolutionGPUTest.BackwardBiasTest
[       OK ] ConvolutionGPUTest.BackwardBiasTest (50 ms)
[----------] 4 tests from ConvolutionGPUTest (1897 ms total)

[----------] Global test environment tear-down
[==========] 8 tests from 2 test suites ran. (5442 ms total)
[  PASSED  ] 8 tests.
```

```
(base) linyibo@linyibo-ThinkBook-14-G5-IRH:~/Code/my-tensor/build$ ./pooling_test
[==========] Running 6 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 3 tests from PoolingCPUTest
[ RUN      ] PoolingCPUTest.ForwardTop
[       OK ] PoolingCPUTest.ForwardTop (10 ms)
[ RUN      ] PoolingCPUTest.ForwardMask
[       OK ] PoolingCPUTest.ForwardMask (11 ms)
[ RUN      ] PoolingCPUTest.BackwardBottom
[       OK ] PoolingCPUTest.BackwardBottom (17 ms)
[----------] 3 tests from PoolingCPUTest (39 ms total)

[----------] 3 tests from PoolingGPUTest
[ RUN      ] PoolingGPUTest.ForwardTop
[       OK ] PoolingGPUTest.ForwardTop (137 ms)
[ RUN      ] PoolingGPUTest.ForwardMask
[       OK ] PoolingGPUTest.ForwardMask (82 ms)
[ RUN      ] PoolingGPUTest.BackwardBottom
[       OK ] PoolingGPUTest.BackwardBottom (320 ms)
[----------] 3 tests from PoolingGPUTest (540 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 2 test suites ran. (579 ms total)
[  PASSED  ] 6 tests.
```

```
(base) linyibo@linyibo-ThinkBook-14-G5-IRH:~/Code/my-tensor/build$ ./softmax_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from SoftmaxTest
[ RUN      ] SoftmaxTest.ForwardCPUTest
[       OK ] SoftmaxTest.ForwardCPUTest (4 ms)
[ RUN      ] SoftmaxTest.ForwardGPUTest
[       OK ] SoftmaxTest.ForwardGPUTest (119 ms)
[----------] 2 tests from SoftmaxTest (123 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (123 ms total)
[  PASSED  ] 2 tests.
```

```
(base) linyibo@linyibo-ThinkBook-14-G5-IRH:~/Code/my-tensor/build$ ./loss_with_softmax_test
[==========] Running 4 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from LossWithSoftmaxCPUTest
[ RUN      ] LossWithSoftmaxCPUTest.ForwardLoss
[       OK ] LossWithSoftmaxCPUTest.ForwardLoss (3 ms)
[ RUN      ] LossWithSoftmaxCPUTest.BackwardBottom
[       OK ] LossWithSoftmaxCPUTest.BackwardBottom (4 ms)
[----------] 2 tests from LossWithSoftmaxCPUTest (7 ms total)

[----------] 2 tests from LossWithSoftmaxGPUTest
[ RUN      ] LossWithSoftmaxGPUTest.ForwardLoss
[       OK ] LossWithSoftmaxGPUTest.ForwardLoss (66 ms)
[ RUN      ] LossWithSoftmaxGPUTest.BackwardBottom
[       OK ] LossWithSoftmaxGPUTest.BackwardBottom (59 ms)
[----------] 2 tests from LossWithSoftmaxGPUTest (127 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 2 test suites ran. (135 ms total)
[  PASSED  ] 4 tests.
```

确实是全部通过了测试。