编译技术project2报告

小组编号：7

小组成员

郭宇琪 1700012785

顾怿洋 1700012788

周博文 1700013028

袁皓晨 1700012959

组内分工

求导原理描述

在本部分，我们使用以下这个简单的矩阵乘法运算例子，解释我们的导数计算方法。

> A<32, 16>[i, j] = C<32, 16>[i, k] * B<32, 16>[k, j] + 1.0

根据我们在project1中的代码实现，我们为这样的一个运算表达式生成了如下的计算代码。

```
void kernel_example(float (&B)[32][16], float (&C)[32][16], float (&A)[32][16]) {
  float temp1[32][16];
  for (int i = 0;i < 32;++i){
    for (int j = 0;j < 16;++j){
      temp1[i][j] = 0;
      for (int k = 0;k < 16;++k){
        if (0 <= i && i < 32) {
          if (0 <= k && k < 16) {
            if (0 <= k && k < 32) {
              if (0 <= j && j < 16) {
                temp1[i][j] += C[i][k] * B[k][j];
              }
            }
          }
        }
      }
      temp1[i][j] += 1;
    }
  }
  for (int i = 0;i < 32;++i){
    for (int j = 0;j < 16;++j){
      A[i][j] = temp1[i][j];
    }
  }
}
```

容易看出，我们为每一条计算表达式生成了三个语句块。

第一个语句块是临时数组的声明。

```
float temp1[32][16];
```

第二个语句块是一系列的循环，按照计算表达式中每一个项(item)出现的次序，进行计算，并将结果累加到临时数组中。

```
for (int i = 0;i < 32;++i){
    for (int j = 0;j < 16;++j){
      temp1[i][j] = 0;

      //item-1
      for (int k = 0;k < 16;++k){
        if (0 <= i && i < 32) {
          if (0 <= k && k < 16) {
            if (0 <= k && k < 32) {
              if (0 <= j && j < 16) {
                temp1[i][j] += C[i][k] * B[k][j];
              }
            }
          }
        }
      }

      //item-2
      temp1[i][j] += 1;
    }
  }
```

由于要求支持爱因斯坦求和约定，因此在每一项的计算过程中，可能出现更多的内层循环。在这个例子中，item-1的计算过程中就出现了关于k的内层循环。为了叙述方便，我们在此约定，将内层循环的循环体称为子项(subitem)。在本例中，item-1的子项为

> temp1[i][j] += C[i][k] * B[k][j]

第三个语句块也是一系列的循环。这部分则将临时数组中的结果复制到输出数组中，完成计算结果的输出。

```
for (int i = 0;i < 32;++i){
    for (int j = 0;j < 16;++j){
      A[i][j] = temp1[i][j];
    }
  }
```

基于以上的分析，我们可以给出我们在计算过程时遵循的数学表达式：

$$A=\sum_{i,j}(item1+item2)=\sum_{i,j}(\sum_k subitem1+item2)$$

现在，假设我们需要求对B矩阵的导数。由链式求导法则可知

$$\frac{\partial Loss}{\partial B}=\sum_{m,n}(\frac{\partial Loss}{\partial A(m,n)}\times\frac{\partial A(m,n)}{\partial B})$$

其中

$$\frac{\partial Loss}{\partial A(m,n)}=dA(m,n)$$

已经给出，因此我们只需求

$$\frac{\partial A(m,n)}{\partial B}$$

将前面给出的A的计算表达式代入这个式子，得出

$$\frac{\partial A(m,n)}{\partial B}=\frac{\partial(\sum_{i,j}(\sum_k subitem1+item2))}{\partial B}$$

化简可得

$$\frac{\partial A(m,n)}{\partial B}=\sum_{i,j}(\sum_k\frac{\partial subitem1}{\partial B}+\frac{\partial item2}{\partial B})$$

对比A的计算表达式，可以发现，A对B矩阵的导数表达式和A矩阵的计算表达式具有相似的结构，这意味着两者的代码结构也应高度相似，我们可以充分利用原有代码来构造求导计算代码。经过仔细分析，我们发现，只需将原来代码等式右边的部分对B求导，即可在不对代码结构做大幅度修改的情况下，计算出A对B的矩阵导数。

在将等式右边部分对B求导后，我们得到了如下的代码结构。其中index1和index2是B矩阵的下标，这段代码计算的是A矩阵对B[index1,index2]的导数。

```
for (int i = 0;i < 4;++i){
  for (int j = 0;j < 16;++j){
    temp1[i][j] = 0;

      //item-1
      if (0 <= i && i < 4) {
        if (0 <= j && j < 16) {
          if (0 <= i && i < 4) {
            if (0 <= j && j < 16) {
                temp1[i][j] += (( index1 == i && index2 == j ) ? ( 1 ) : ( 0 )) *
B[i][j];
            }
          }
        }
      }

      //item-2
      temp1[i][j] += 0;
    }
  }
```

随后，将这一结果与dA矩阵相乘累加，即可完成dB[index1,index2]的计算。最后，在最外层增加关于index1和index2的循环，即可完成整个dB矩阵的计算。完整版本的代码如下。

```
void grad_example(float (&C)[32][16], float (&dA)[32][16], float (&dB)[32][16]) {
  for (int index1 = 0;index1 < 32;++index1){
    for (int index2 = 0;index2 < 16;++index2){
      float temp1[32][16];
      dB[index1][index2] = 0.0;
      for (int i = 0;i < 32;++i){
        for (int j = 0;j < 16;++j){
          temp1[i][j] = 0;
          for (int k = 0;k < 16;++k){
            if (0 <= i && i < 32) {
              if (0 <= k && k < 16) {
                if (0 <= k && k < 32) {
                  if (0 <= j && j < 16) {
                    temp1[i][j] += C[i][k] * (( index1 == k && index2 == j ) ? ( 1
) : ( 0 ));
                  }
                }
              }
            }
          }
          temp1[i][j] += 0;
        }
      }
      for (int i = 0;i < 32;++i){
        for (int j = 0;j < 16;++j){
          dB[index1][index2] += dA[i][j] * temp1[i][j];
        }
      }
    }
  }
}
```