实验题目 - logsoftmax

在上一节中，我们实现了 实现自动求导算子基类Function，现在，你需要在此基础上，实现一个 `logsoftmax` 函数，其输入是一个矩阵，输出是对矩阵的每一行进行 `logsoftmax` 操作后的结果。

`logsoftmax` 的定义如下（每一行的操作）：

$$\text{logsoftmax}(x_{i,j}) = log\frac{e^{x_{i,j}}}{\sum_1^j e^{x_{i,j}}}$$

其中，提供测试基类的 `main()` 函数如下

```
int main() {
    using T=float;
    // Create a GradTensor<T> object A with dimensions [m, n]
    // vector<int>  shape = {1,2,3};
    int m, n, k;
    cin>>m>>n>>k;
    vector<int>  shape = {n,k};
    GradTensor<T> A(2, shape.data());
    std::cout<<"Tensor A:"<<std::endl;
    A.printTensor();


    // // Create instances of Add and Mul classes
    // Add<T> add;
    // Mul<T> mul;

    // // Calculate A + A and A * A using the add and mul objects
    // GradTensor<T> resultMul = mul.forward(A,A);
    // std::cout<<"\nTensor A*A:"<<std::endl;
    // resultMul.printTensor();
    // GradTensor<T> resultAdd = add.forward(resultMul,A);
    // std::cout<<"\nTensor A*A + A:"<<std::endl;
    // resultAdd.printTensor();

    // // Tensor<T> gradA;
    // // // Calculate gradients of A*A + A with respect to A
    // // A.grad=Tensor<T>();
    // add.backward(A);
```

```
    // std::cout<<"\nA's grad after add backward:"<<std::endl;
    // A.grad.printTensor();
    // std::cout<<"\n(A*A)'s grad after add backward:"<<std::endl;
    // resultMul.grad.printTensor();
    // // A.grad=Tensor<T>();
    // mul.backward(resultMul.grad);
    // std::cout<<"\nA's grad after (A*A)+A backward:"<<std::endl;
    // A.grad.printTensor();


    Add<T> add1;
    Mul<T> mul1;
    LogSoftmax<T> logsoftmax;
    A.grad_zero_();
    // GradTensor<T> resultMulAddLogSoftmax =
logsoftmax.forward(add1.forward(mul1.forward(A,A),A));
    GradTensor<T> resultMulAddLogSoftmax =
logsoftmax(add1(mul1(A,A),A));
    std::cout<<"\nresultMulAddLogSoftmax:"<<std::endl;
    resultMulAddLogSoftmax.printTensor();

    logsoftmax.backward(A);
    std::cout<<"\nOnePass: A's grad after LogSoftmax backward:"
<<std::endl;
    A.grad.printTensor();
    add1.backward();
    std::cout<<"\nOnePass: A's grad after add backward:"
<<std::endl;
    A.grad.printTensor();
    // A.grad=Tensor<T>();
    mul1.backward();
    std::cout<<"\nOnePass: A's grad after (A*A)+A backward:"
<<std::endl;
    A.grad.printTensor();

    // resultMulAddLogSoftmax.backward(A);
    // std::cout<<"\nOnePass: A's grad after (A*A)+A backward:"
<<std::endl;
    // A.grad.printTensor();

    return 0;
}
```

前向传播

`logsoftmax` 的计算过程 `forward` 中，需要对矩阵进行遍历，计算每一行的 `exp` 和 `sum`

假设输入向量为 $x = (x_1, x_2, \ldots, x_n)$， `logsoftmax` 的前向计算公式为：

$$\text{logsoftmax}(x_i) = \ln\left(\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}\right)$$

反向求导

`logsoftmax` 的反向传播过程 `backward` 中，需要计算 `softmax` 的导数，其计算公式（以 $N = 1$ 为例）如下：

对于 `logsoftmax` 的反向求导，我们需要计算出其梯度，对于梯度的计算，我们需要分两种情况来讨论：

1. 当 $i = j$ 时：

$$\frac{\partial \text{logsoftmax}(x_i)}{\partial x_i} = 1 - e^{\text{logsoftmax}(x_j)}$$

2. 当 $i \neq j$ 时：

$$\frac{\partial \text{logsoftmax}(x_i)}{\partial x_j} = -e^{\text{logsoftmax}(x_j)}$$

   这里，$\text{logsoftmax}(x_i)$ 和 $\text{logsoftmax}(x_j)$ 分别表示 `logsoftmax` 函数在 x_i 和 x_j 处的值。 最后，将所得结果加和，即为以 z_i 对 x_i 求偏导数的结果，即对于梯度的每一项，都有

$$\frac{\partial z_i}{\partial x_i} = (1 - e^{\text{logsoftmax}(x_i)}) \times a_i + \sum_{j=1}^{j \leq n, j \neq i} (-e^{\text{logsoftmax}(x_i)} \times a_j)$$

   你也可以使用 `log` 函数和 `softmax` 函数来实现 `logsoftmax` 函数，但是注意数值稳定性问题，就是注意数值不要溢出到 `0` 或者无穷。

样例输入

```
2 3
```

样例输出

```
Tensor A:
0.548814 0.592845 0.715189
0.844266 0.602763 0.857946

resultMulAddLogSoftmax:
-1.26873 -1.17443 -0.892051
-0.952218 -1.54318 -0.915252
LogSoftmax backward

OnePass: A's grad after LogSoftmax backward:
0 0 0
0 0 0

OnePass: A's grad after add backward:
0.0266874 0.019086 -0.0457734
-0.0451876 0.11019 -0.0650024

OnePass: A's grad after (A*A)+A backward:
0.0559802 0.0417161 -0.111247
-0.121488 0.243027 -0.176539
```

样例解释

以输入的 `0.548814 0.592845 0.715189` 的第一项 `0.548814` 为例

$$\text{logsoftmax}(a_0) = \ln\left(\frac{e^{0.548814}}{e^{0.548814}+e^{0.592845}+e^{0.715189}}\right) = -1.26872$$

然后通过 logsoftmax 的 `backward` 函数进行反向传播，可得 `logsoftmax(add1(mul1(A, A), A))` 的梯度（记 $x_i = a_i \times a_i + a_i$）

$$\begin{aligned}\frac{\partial z_1}{\partial x_1} &= (1 - e^{\text{logsoftmax}(x_1)}) \times a_1 + (-e^{\text{logsoftmax}(x_1)} \times a_2) + (-e^{\text{logsoftmax}(x_1)} \times a_3) \\ &= 0.0266874\end{aligned}$$

将 `logsoftmax(add1(mul1(A, A), A))` 的梯度传播到 `add(mul1(A, A), A)`，此时梯度尚未传递到 `A`，所以此时执行 `A.grad.printTensor` 输出的是每一项均为 `0` 的矩阵

在此之后，通过 `add1.backward()` 将 `logsoftmax` 传入的梯度再传播给 `mul1(A, A)` 和 `A`，故此时执行 `A.grad.printTensor` 输出才是刚刚计算得到的梯度结果

最后，通过 `mul1.backward()` 将 `add1` 传入的梯度再次传播给 `A`，此时

$$
\begin{aligned}
\frac{\partial z_1}{\partial a_1} &= \frac{\partial z_1}{\partial x_1} \times \frac{\mathrm{d}x_1}{\mathrm{d}a_1} \\
&= \left[(1 - e^{\mathrm{logsoftmax}(x_1)}) \times a_i + \sum_{j=1}^{j\leq,j\neq i} (-e^{\mathrm{logsoftmax}(x_1)} \times a_j)\right] \times (2 \times a_1 + 1) \\
&= 0.0266874 \times (2 \times 0.548814 + 1) \\
&= 0.0559802
\end{aligned}
$$