Reduce 层

- 初始示例代码
- op
- axes
- keep_dims

初始示例代码

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 3, 4, 5  # 输入张量 NCHW
data = np.ones([nIn, cIn, hIn, wIn], dtype=np.float32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))  # 单输入示例代码
#------------------------------------------------------------ -------------------# 替换部分
reduceLayer = network.add_reduce(inputT0, trt.ReduceOperation.SUM, 1 << 1, False)
#------------------------------------------------------------ -------------------# 替换部分
network.mark_output(reduceLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输入张量形状 (1,3,4,5)

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix} \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix} \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量形状 (1,4,5)，在次高维上进行了求和

$$\begin{bmatrix}\begin{bmatrix}3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\end{bmatrix}\end{bmatrix}$$

---

op

```
reduce = network.add_reduce(inputTensor, trt.ReduceOperation.PROD, 1 << 0, False)
reduce.op = trt.ReduceOperation.SUM  # 重设规约运算种类
```

- 输出张量形状 (1,4,5)，结果与初始示例代码相同
- 可用的规约计算方法

trt.ReduceOperation	函数
PROD	求积
AVG	求平均值
MAX	取最大值
MIN	取最小值
SUM	求和

---

axes

```
axesIndex = 0
reduceLayer = network.add_reduce(inputT0, trt.ReduceOperation.SUM, 1 << 1, False)
reduceLayer.axes = 1 << axesIndex  # 规约计算的轴号
```

- 指定 axes=1<<0，输出张量形状 (3,4,5)，在最高维上进行规约，相当于什么也没做

$$\begin{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix} \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix} \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1.\end{bmatrix}\end{bmatrix}$$

- 指定 axes=1<<1，输出张量形状 (1,4,5)，在次高维上进行规约，结果与初始示例代码相同
- 指定 axes=1<<2，输出张量形状 (1,3,5)，在季高维上进行规约

$$\begin{bmatrix}\begin{bmatrix}4. & 4. & 4. & 4. & 4.\\ 4. & 4. & 4. & 4. & 4.\\ 4. & 4. & 4. & 4. & 4.\end{bmatrix}\end{bmatrix}$$

- 指定 axes=(1<<2)+(1<<3)，输出张量形状 (1,3)，同时在第二和第三维度上进行规约，注意 << 优先级低于 +，要加括号

$$[20.\quad 20.\quad 20.]$$

keep_dims

```
reduceLayer = network.add_reduce(inputT0, trt.ReduceOperation.SUM, 1 << 1, False)
reduceLayer.keep_dims = True  # 重设是否保留被规约维度
```

- 指定 keep_dims=True，输出张量形状 (1,1,4,5)，保留了发生规约的维度

$$\left[\left[\begin{bmatrix}3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\\ 3. & 3. & 3. & 3. & 3.\end{bmatrix}\right]\right]$$

- 指定 keep_dims=False，输出张量形状 (1,4,5)，结果与初始示例代码相同