Element Wise 层

- 初始示例代码
- op
- 广播操作

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
data0 = np.full([nIn, cIn, hIn, wIn], 1, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)  # 输入数据
data1 = np.full([nIn, cIn, hIn, wIn], 2, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)

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()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#---------------------------------------------------------- --------------------# 替换部分
elementwiseLayer = network.add_elementwise(inputT0, inputT1, trt.ElementWiseOperation.SUM)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(elementwiseLayer.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(data0.reshape(-1))
inputH1 = np.ascontiguousarray(data1.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
```

```python
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)

$$\left[\left[\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\end{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\end{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\\1. & 1. & 1. & 1. & 1.\end{bmatrix}\right]\right]$$

$$\left[\left[\begin{bmatrix}2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\end{bmatrix}\begin{bmatrix}2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\end{bmatrix}\begin{bmatrix}2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\\2. & 2. & 2. & 2. & 2.\end{bmatrix}\right]\right]$$

- 输出张量形状 (1,3,4,5)，两个输入张量做逐元素加法

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

---

op

```python
elementwiseLayer = network.add_elementwise(inputT0, inputT1, trt.ElementWiseOperation.SUM)
elementwiseLayer.op = trt.ElementWiseOperation.SUB  # 重设运算种类
```

- 输出张量形状 (1,3,4,5)，每个元素变成 0

$$\left[\left[\begin{bmatrix}0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\end{bmatrix}\begin{bmatrix}0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\end{bmatrix}\begin{bmatrix}0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\\0. & 0. & 0. & 0. & 0.\end{bmatrix}\right]\right]$$

- 可用的计算类型

trt.ElementWiseOperation 名	$f(a,b)$	**备注**
SUM	a + b	
PROD	a * b	
MAX	$\max(a,b)$	
MIN	$\min(a,b)$	
SUB	a - b	
DIV	a / b	
POW	a ** b (a^b)	输入 float32/float16 型
FLOOR_DIV	a // b	
AND	a and b	输入输出都是 Bool 型
OR	a or b	输入输出都是 Bool 型
XOR	a ^ b (a xor b)	输入输出都是 Bool 型
EQUAL	a == b	输出是 Bool 型
GREATER	a > b	输出是 Bool 型
LESS	a < b	输出是 Bool 型

- 需要 BOOL 型输入的算子在输入其他数据类型时报错：

```
[TensorRT] ERROR: 4: [layers.cpp::validate::2304] Error Code 4: Internal Error ((Unnamed Layer* 0)
[ElementWise]: operation AND requires boolean inputs.)
```

- 添加 float/int 与 bool 类型转换的 Idenetity 层时注意设置 config.max_workspace_size，否则报错：

```
[TensorRT] ERROR: 1: [codeGenerator.cpp::createMyelinGraph::314] Error Code 1: Myelin
(myelinTargetSetPropertyMemorySize called with invalid memory size (0).)
```

- POW 的两个输入必须是 activation type，否则报错：

```
[TensorRT] ERROR: 4: [layers.cpp::validate::2322] Error Code 4: Internal Error ((Unnamed Layer* 0)
[ElementWise]: operation POW requires inputs with activation type.)
# 或者
[TensorRT] ERROR: 4: [layers.cpp::validate::2291] Error Code 4: Internal Error ((Unnamed Layer* 0)
[ElementWise]: operation POW has incompatible input types Float and Int32)
```

广播操作

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

nIn,cIn,hIn,wIn = 1,3,4,5
data0    = np.full([nIn,cIn,1,wIn],1,dtype=np.float32).reshape(nIn,cIn,1,wIn)
data1    = np.full([nIn,1,hIn,1],2,dtype=np.float32).reshape(nIn,1,hIn,1)

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()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn,cIn,1,wIn))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn,1,hIn,1))
elementwiseLayer = network.add_elementwise(inputT0,inputT1,trt.ElementWiseOperation.SUM)
network.mark_output(elementwiseLayer.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(data0.reshape(-1))
inputH1     = np.ascontiguousarray(data1.reshape(-1))
outputH0    = np.empty(context.get_binding_shape(2),dtype = trt.nptype(engine.get_binding_dtype(2)))
_,inputD0   = cudart.cudaMallocAsync(inputH0.nbytes,stream)
_,inputD1   = cudart.cudaMallocAsync(inputH1.nbytes,stream)
_,outputD0  = cudart.cudaMallocAsync(outputH0.nbytes,stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
cudart.cudaMemcpyAsync(inputD1, inputH1.ctypes.data, inputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data0.shape)
print(data0)
print("inputH0 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

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

- 输入张量形状 (1,3,1,5) 和 (1,1,4,1)

$$[[[1. \quad 1. \quad 1. \quad 1. \quad 1.][1. \quad 1. \quad 1. \quad 1. \quad 1.][1. \quad 1. \quad 1. \quad 1. \quad 1.]]]$$

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}2.\\2.\\2.\\2.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量形状 (1,3,4,5)，将两加数广播后进行计算，结果与初始示例代码相同