Scale 层

- 初始示例代码
- mode & scale & shift & power
- CHANNEL 和 ELEMENTWISE 级的 scale
- add_scale_nd 及其参数 channel_axis

初始示例代码

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

nIn, cIn, hIn, wIn = 1, 3, 3, 3  # 输入张量 NCHW
data = np.arange(1, 1 + nIn * cIn * wIn * wIn, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)  # 输入数据

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))
#---------------------------------------------------------- --------------------# 替换部分
scale = np.array([0.5], dtype=np.float32)
shift = np.array([-7.0], dtype=np.float32)
power = np.array([1.0], dtype=np.float32)
scaleLayer = network.add_scale(inputT0, trt.ScaleMode.UNIFORM, shift, scale, power)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(scaleLayer.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 (1,3,3,3)

$$\left[\left[\begin{bmatrix}1. & 2. & 3.\\ 4. & 5. & 6.\\ 7. & 8. & 9.\end{bmatrix}\begin{bmatrix}10. & 11. & 12.\\ 13. & 14. & 15.\\ 16. & 17. & 18.\end{bmatrix}\begin{bmatrix}19. & 20. & 21.\\ 22. & 23. & 24.\\ 25. & 26. & 27.\end{bmatrix}\right]\right]$$

- 输出张量形状 (1,3,3,3)，所有元素都做了变换 $y = (x \cdot scale + shift)^{power}$

$$\left[\left[\begin{bmatrix}-6.5 & -6. & -5.5\\ -5. & -4.5 & -4.\\ -3.5 & -3. & -2.5\end{bmatrix}\begin{bmatrix}-2. & -1.5 & -1.\\ -0.5 & 0. & 0.5\\ 1. & 1.5 & 2.\end{bmatrix}\begin{bmatrix}2.5 & 3. & 3.5\\ 4. & 4.5 & 5.\\ 5.5 & 6. & 6.5\end{bmatrix}\right]\right]$$

- TensorRT6 及之前版本需要对参数 scale，shift，power 的 np 数组作拷贝，防止其被后续同名变量覆盖，因为 TensorRT 中这些参数的定义和使用是异步的。TensorRT7 及之后该问题被修正，不再需要额外的拷贝工作

```
# TensorRT 6
bag = []
scale = np.array([0.5], dtype=np.float32)
shift = np.array([-7.0], dtype=np.float32)
power = np.array([1.0], dtype=np.float32)
bag += [scale, shift, power]
scaleLaer = network.add_scale(...)
```

mode & scale & shift & power

```
one = np.array([1], dtype=np.float32)
scaleLayer = network.add_scale(inputT0, trt.ScaleMode.UNIFORM, one, one, one)
scaleLayer.scale = np.array([0.5], dtype=np.float32)  # 乘法参数
scaleLayer.shift = np.array([-7.0], dtype=np.float32)  # 加法参数
scaleLayer.power = np.array([1.0], dtype=np.float32)  # 指数参数
```

- 输出张量形状 (3,3,3)，与初始示例代码相同
- 可用的模式

trt.ScaleMode 名	说明
ELEMENTWISE	每个元素使用一套参数
UNIFORM	所有元素使用一套参数
CHANNEL	同个 C 维的元素使用一套参数

CHANNEL 和 ELEMENTWISE 级的 scale

```
shift = np.array([-2.5, -7.0, -11.5], dtype=np.float32)  # 参数元素数等于通道数
scale = np.full(3, 0.5, dtype=np.float32)
power = np.ones(3, dtype=np.float32)
sc = network.add_scale(inputTensor, trt.ScaleMode.CHANNEL, shift, scale, power)
print("sc->", sc.get_output(0).shape)
```

- 输出张量形状 (1,3,3,3)，每个通道依不同参数进行 scale

$$\left[\left[\begin{bmatrix} -2. & -1.5 & -1. \\ -0.5 & 0. & 0.5 \\ 1. & 1.5 & 2. \end{bmatrix} \begin{bmatrix} -2. & -1.5 & -1. \\ -0.5 & 0. & 0.5 \\ 1. & 1.5 & 2. \end{bmatrix} \begin{bmatrix} -2. & -1.5 & -1. \\ -0.5 & 0. & 0.5 \\ 1. & 1.5 & 2. \end{bmatrix}\right]\right]$$

```
shift = np.full([cIn, hIn, wIn], -7.0, dtype=np.float32)  # 参数元素数等于输入张量元素数
scale = np.full([cIn, hIn, wIn], 0.5, dtype=np.float32)
power = np.ones([cIn, hIn, wIn], dtype=np.float32)
scaleLayer = network.add_scale(inputT0, trt.ScaleMode.ELEMENTWISE, shift, scale, power)
```

- 输出张量形状 (1,3,3,3)，每个元素依不同参数进行 scale，结果与初始示例代码相同

使用 add_scale_nd 和 channel_axis

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

nIn, cIn, hIn, wIn = 2, 2, 3, 4
data = np.zeros([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()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))

scale = np.array([1.0, 1.0], dtype=np.float32)
shift = np.array([2.0, 3.0], dtype=np.float32)
power = np.array([1.0, 1.0], dtype=np.float32)
scaleLayer = network.add_scale_nd(inputT0, trt.ScaleMode.CHANNEL, shift, scale, power, 0)
scaleLayer.channel_axis = 0  # 设置 scale 的轴号，0（N 轴）或 1（C 轴）

network.mark_output(scaleLayer.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 (2,2,3,4)

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

- 指定 channel_axis=0（在 N 维上进行 scale）， 输出张量形状 (2, 2, 3, 4)，每个 batch 内 2 个通道共用一套参数

$$\left[\begin{matrix}\left[\begin{matrix}\left[\begin{matrix}2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2.\end{matrix}\right] \left[\begin{matrix}2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2.\end{matrix}\right]\end{matrix}\right] \\ \left[\begin{matrix}\left[\begin{matrix}3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3.\end{matrix}\right] \left[\begin{matrix}3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3.\end{matrix}\right]\end{matrix}\right]\end{matrix}\right]$$

- 指定 channel_axis=1（在 C 维上进行 scale）， 输出张量形状 (2, 2, 3, 4)，各 batch 内同一通道共用一套参数

$$\left[\begin{matrix}\left[\begin{matrix}\left[\begin{matrix}2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2.\end{matrix}\right] \left[\begin{matrix}3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3.\end{matrix}\right]\end{matrix}\right] \\ \left[\begin{matrix}\left[\begin{matrix}2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2. \\ 2. & 2. & 2. & 2.\end{matrix}\right] \left[\begin{matrix}3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3. \\ 3. & 3. & 3. & 3.\end{matrix}\right]\end{matrix}\right]\end{matrix}\right]$$