Fill 层

- 初始示例代码
- set_input 与 buildtime 线性填充
- set_input 与 buildtime 均匀随机填充
- buildtime 指定形状 runtime 指定范围的均匀随机填充
- runtime 指定形状和范围的均匀随机填充

初始示例代码

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

nOut, cOut, hOut, wOut = 1, 3, 4, 5  # 输出张量形状 NCHW

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()
#---------------------------------------------------------- --------------------# 替换部分
fillLayer = network.add_fill((nOut, cOut, hOut, wOut), trt.FillOperation.LINSPACE)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(fillLayer.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()

outputH0 = np.empty(context.get_binding_shape(0), dtype=trt.nptype(engine.get_binding_dtype(0)))
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

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

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

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

- TensorRT7 输出张量形状 (1,3,4,5)，TensorRT8 建立网络失败，没有输出

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

- 包含错误，因为指定 LINSPACE 模式填充，但是没有指定起点张量（α）和增量张量（β）

```
#TensorRT 7:
[TensorRT] ERROR: 2: [fillRunner.cpp::executeLinSpace::46] Error Code 2: Internal Error (Assertion
dims.nbDims == 1 failed.Alpha and beta tensor should be set when output an ND tensor)
[TensorRT] INTERNAL ERROR: Assertion failed: dims.nbDims == 1 && "Alpha and beta tensor should be set
when output an ND tensor"
#TensorRT 8:
[TRT] [E] 2: [fillRunner.cpp::executeLinSpace::46] Error Code 2: Internal Error (Assertion dims.nbDims
== 1 failed. Alpha and beta tensor should be set when output an ND tensor)
```

set_input 与线性填充

```
fillLayer = network.add_fill((1, 1, 1), trt.FillOperation.LINSPACE)
constantLayer0 = network.add_constant((4, ), np.array([nOut, cOut, hOut, wOut], dtype=np.int32))  # 形状张量
constantLayer1 = network.add_constant((), np.array([1000], dtype=np.float32))  # 初值标量
constantLayer2 = network.add_constant((4, ), np.array([0, 100, 10, 1], dtype=np.float32))  # 增量张量
fillLayer.set_input(0, constantLayer0.get_output(0))
fillLayer.set_input(1, constantLayer1.get_output(0))
fillLayer.set_input(2, constantLayer2.get_output(0))
```

- 输出张量形状 (1,3,4,5)

$$\left[\left[\begin{bmatrix}1000. & 1001. & 1002. & 1003. & 1004.\\ 1010. & 1011. & 1012. & 1013. & 1014.\\ 1020. & 1021. & 1022. & 1023. & 1024.\\ 1030. & 1031. & 1032. & 1033. & 1034.\end{bmatrix}\begin{bmatrix}1100. & 1101. & 1102. & 1103. & 1104.\\ 1110. & 1111. & 1112. & 1113. & 1114.\\ 1120. & 1121. & 1122. & 1123. & 1124.\\ 1130. & 1131. & 1132. & 1133. & 1134.\end{bmatrix}\begin{bmatrix}1200. & 1201. & 1202. & 1203. & 1204.\\ 1210. & 1211. & 1212. & 1213. & 1214.\\ 1220. & 1221. & 1222. & 1223. & 1224.\\ 1230. & 1231. & 1232. & 1233. & 1234.\end{bmatrix}\right]\right]$$

set_input 与均匀随机填充

```
fillLayer = network.add_fill((1, 1, 1), trt.FillOperation.RANDOM_UNIFORM)
constantLayer0 = network.add_constant((4, ), np.array([nOut, cOut, hOut, wOut], dtype=np.int32))  # 形状张量
constantLayer1 = network.add_constant((), np.array([-10], dtype=np.float32))  # 最小值标量
constantLayer2 = network.add_constant((), np.array([10], dtype=np.float32))  # 最大指标量
fillLayer.set_input(0, constantLayer0.get_output(0))
fillLayer.set_input(1, constantLayer1.get_output(0))
fillLayer.set_input(2, constantLayer2.get_output(0))
```

- 输出张量形状 (1,3,4,5)

$$\left[\left[\begin{bmatrix}-9.167 & 0.371 & -7.195 & 2.565 & 9.814\\ -9.129 & -2.469 & 4.144 & 0.099 & -6.012\\ 9.422 & -9.963 & -2.179 & 8.372 & -9.639\\ 6.106 & 6.965 & -0.493 & 6.938 & -7.576\end{bmatrix}\begin{bmatrix}6.567 & 5.466 & -6.148 & -7.764 & -5.719\\ 4.527 & 1.752 & -7.469 & 1.24 & -6.424\\ -9.2 & 3.142 & 9.268 & 9.176 & -6.118\\ -1.818 & 5.001 & -3.764 & 9.836 & -9.384\end{bmatrix}\begin{bmatrix}-6.398 & 7.669 & -6.942 & -7.131 & 8.463\\ -0.08 & -7.027 & 9.608 & 2.046 & -7.655\\ 1.096 & -4.69 & 7.327 & -6.187 & 3.415\\ -5.887 & -2.402 & -6.263 & -1.868 & -4.79\end{bmatrix}\right]\right]$$

- 注意：
  - 随机数种子固定，按相同形状和数值范围生成的随机数是相同的
  - 建立 engine 后其数值为常数，多次运行数值均相同

buildtime 指定形状 runtime 指定范围的均匀随机填充

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

```
import tensorrt as trt

nOut, cOut, hOut, wOut = 1, 3, 4, 5

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  # 设置空间给 TensoRT 尝试优化，单位 Byte
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, ())
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, ())
#------------------------------------------------------- --------------------# 替换部分
fillLayer = network.add_fill([nOut, cOut, hOut, wOut], trt.FillOperation.RANDOM_UNIFORM)
fillLayer.set_input(1, inputT0)
fillLayer.set_input(2, inputT1)
#------------------------------------------------------- --------------------# 替换部分
network.mark_output(fillLayer.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(np.array([-10], dtype=np.float32).reshape(-1))
inputH1 = np.ascontiguousarray(np.array([10], dtype=np.float32).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("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(inputD1)
cudart.cudaFree(outputD0)
```

- 输出张量形状 (1,3,4,5)，结果与“set_input 与均匀随机填充”示例相同

runtime 指定形状和范围的均匀随机填充

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

nOut, cOut, hOut, wOut = 1, 3, 4, 5
```

```
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))
profile = builder.create_optimization_profile()  # 需要使用 profile
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30  # 设置空间给 TensoRT 尝试优化，单位 Byte
inputT0 = network.add_input('inputT0', trt.DataType.INT32, (4, ))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, ())
inputT2 = network.add_input('inputT2', trt.DataType.FLOAT, ())
profile.set_shape_input(inputT0.name, (1, 1, 1, 1), (nOut, cOut, hOut, wOut), (5, 5, 5, 5))  # 这里设置的不是 shape input 的形状而是值，范围覆盖住之后需要的值就好
config.add_optimization_profile(profile)
#----------------------------------------------------------- --------------------# 替换部分
fillLayer = network.add_fill([1, 1, 1, 1], trt.FillOperation.RANDOM_UNIFORM)
fillLayer.set_input(0, inputT0)
fillLayer.set_input(1, inputT1)
fillLayer.set_input(2, inputT2)
#----------------------------------------------------------- --------------------# 替换部分
network.mark_output(fillLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
context.set_shape_input(0, [nOut, cOut, hOut, wOut])  # 运行时绑定真实形状张量值
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(np.array([0, 0, 0, 0], dtype=np.int32).reshape(-1))  # 传形状张量数据可用垃圾值
inputH1 = np.ascontiguousarray(np.array([-10], dtype=np.float32).reshape(-1))
inputH2 = np.ascontiguousarray(np.array([10], dtype=np.float32).reshape(-1))
outputH0 = np.empty(context.get_binding_shape(3), dtype=trt.nptype(engine.get_binding_dtype(3)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, inputD1 = cudart.cudaMallocAsync(inputH1.nbytes, stream)
_, inputD2 = cudart.cudaMallocAsync(inputH2.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)
cudart.cudaMemcpyAsync(inputD2, inputH2.ctypes.data, inputH2.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(inputD1), int(inputD2), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

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

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(inputD1)
cudart.cudaFree(inputD2)
cudart.cudaFree(outputD0)
```

- 输出张量形状 (1,3,4,5)，结果与“set_input 与均匀随机填充”示例相同