MatrixMultiply 层

- 初始示例代码
- op0 & op1
- 乘数广播
- 矩阵乘向量
- add_matrix_multiply_deprecated 及其参数 transpose0 & transpose1（要求 TensorRT<8）

初始示例代码

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

np.random.seed(97)
nIn, cIn, hIn, wIn = 1, 3, 4, 5  # 输入张量 NCHW
data = np.arange(nIn * cIn * hIn * wIn, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)  # 输入张量 HWC

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))
#---------------------------------------------------------- --------------------# 替换部分
factorShape = data.transpose(0, 1, 3, 2).shape
constantLayer = network.add_constant(factorShape, np.ones(factorShape, dtype=np.float32))
matrixMultiplyLayer = network.add_matrix_multiply(inputT0, trt.MatrixOperation.NONE,
constantLayer.get_output(0), trt.MatrixOperation.NONE)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(matrixMultiplyLayer.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)

$$\left[\left[\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 5. & 6. & 7. & 8. & 9. \\ 10. & 11. & 12. & 13. & 14. \\ 15. & 16. & 17. & 18. & 19. \end{bmatrix}\begin{bmatrix} 20. & 21. & 22. & 23. & 24. \\ 25. & 26. & 27. & 28. & 29. \\ 30. & 31. & 32. & 33. & 34. \\ 35. & 36. & 37. & 38. & 39. \end{bmatrix}\begin{bmatrix} 40. & 41. & 42. & 43. & 44. \\ 45. & 46. & 47. & 48. & 49. \\ 50. & 51. & 52. & 53. & 54. \\ 55. & 56. & 57. & 58. & 59. \end{bmatrix}\right]\right]$$

- 输出张量形状 (1,3,4,4)，各通道上进行矩阵乘法

$$\left[\begin{bmatrix} \color{green}{10.} & \color{green}{10.} & \color{green}{10.} & \color{green}{10.} \\ 35. & 35. & 35. & 35. \\ 60. & 60. & 60. & 60. \\ 85. & 85. & 85. & 85. \end{bmatrix}\begin{bmatrix} 110. & 110. & 110. & 110. \\ 135. & 135. & 135. & 135. \\ 160. & 160. & 160. & 160. \\ 185. & 185. & 185. & 185. \end{bmatrix}\begin{bmatrix} 210. & 210. & 210. & 210. \\ 235. & 235. & 235. & 235. \\ 260. & 260. & 260. & 260. \\ 285. & 285. & 285. & 285. \end{bmatrix}\right]$$

- 计算过程：

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \color{green}{10} & \color{green}{10} & \color{green}{10} & \color{green}{10} \\ 35 & 35 & 35 & 35 \\ 60 & 60 & 60 & 60 \\ 85 & 85 & 85 & 85 \end{bmatrix}$$

---

op0 & op1

```
factorShape = data.shape
constantLayer = network.add_constant(factorShape, np.ones(factorShape, dtype=np.float32))  # 这里的 constantayer.get_output(0) 是初始示例代码的转置版本，在 matrixMultiplyLayer 中再转置一次恢复
matrixMultiplyLayer = network.add_matrix_multiply(inputT0, trt.MatrixOperation.NONE, constantLayer.get_output(0), trt.MatrixOperation.NONE)
matrixMultiplyLayer.op0 = trt.MatrixOperation.NONE  # 重设第 0 乘数的预处理，默认值 trt.MatrixOperation.NONE
matrixMultiplyLayer.op1 = trt.MatrixOperation.TRANSPOSE  # 重设第 1 乘数的预处理，默认值 trt.MatrixOperation.NONE
```

- 输出张量形状 (1,3,4,4)，结果与初始示例代码相同。第 1 乘数在进行转置操作后再计算矩阵乘法
- 可用的选项

trt.MatrixOperation 名	说明
NONE	默认行为，不作限制
VECTOR	指明该参数为向量，不进行元素广播（见"矩阵乘向量"示例）
TRANSPOSE	计算乘法前对该矩阵进行转置

乘数广播

```
factorShape = (1, 1) + data.transpose(0, 1, 3, 2).shape[-2:]
constantLayer = network.add_constant(factorShape, np.ones(factorShape, dtype=np.float32))
matrixMultiplyLayer = network.add_matrix_multiply(inputT0, trt.MatrixOperation.NONE,
constantLayer.get_output(0), trt.MatrixOperation.NONE)
```

- 输出张量形状 (1,3,4,4)，结果与初始示例代码相同。乘数的形状由 (1,1,5,4) 广播为 (1,3,5,4) 进行计算

矩阵乘向量

```
factorShape = data.transpose(0, 1, 3, 2).shape[:-1]  # 向量比矩阵少一维
constantLayer = network.add_constant(factorShape, np.ones(factorShape, dtype=np.float32))
matrixMultiplyLayer = network.add_matrix_multiply(inputT0, trt.MatrixOperation.NONE,
constantLayer.get_output(0), trt.MatrixOperation.NONE)
matrixMultiplyLayer.op0 = trt.MatrixOperation.NONE
matrixMultiplyLayer.op1 = trt.MatrixOperation.VECTOR
```

- 输出张量形状 (1,3,4)，输入张量右乘向量

$$\left[\begin{bmatrix} 10. & 35. & 60. & 85. \\ 110. & 135. & 160. & 185. \\ 210. & 235. & 260. & 285. \end{bmatrix}\right]$$

- 计算过程：

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 35 \\ 60 \\ 85 \end{bmatrix}$$

```
factorShape = data.shape[:-1]
constantLayer = network.add_constant(factorShape, np.ones(factorShape, dtype=np.float32))
matrixMultiplyLayer = network.add_matrix_multiply(constantLayer.get_output(0), trt.MatrixOperation.NONE,
inputT0, trt.MatrixOperation.NONE)
matrixMultiplyLayer.op0 = trt.MatrixOperation.VECTOR
matrixMultiplyLayer.op1 = trt.MatrixOperation.NONE
```

- 输出张量形状 (1,3,5)，输入张量左乘向量

$$\left[\begin{bmatrix} 30. & 34. & 38. & 42. & 46. \\ 110. & 114. & 118. & 122. & 126. \\ 190. & 194. & 198. & 202. & 206. \end{bmatrix}\right]$$

- 计算过程：

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \end{bmatrix} = \begin{bmatrix} 30 & 34 & 38 & 42 & 46 \end{bmatrix}$$

---

add_matrix_multiply_deprecated 及其参数 transpose0 & transpose1（TensorRT <8）

```
factorShape = data.transpose(0,1,3,2).shape
constantLayer = network.add_constant(factorShape,np.ones(factorShape,dtype=np.float32))
matrixMultiplyLayer = network.add_matrix_multiply_deprecated(inputT0, True, constantLayer.get_output(0),
True)
matrixMultiplyLayer.transpose0 = False                                                               # 重
设乘数是否转置
matrixMultiplyLayer.transpose1 = False
```

- 输出张量形状 (3, 4, 4)，结果与初始示例代码相同
- transpose0 与 transpose1 作用与前面的 op0 与 op1 作用类似，取值为 True 或 False