

Einsum 层

- **Einsum** 层从 **TensorRT 8.2** 开始支持，但是功能尚不完善
- 初始示例代码（用 **Einsum** 层做双张量单缩并）
- equatuiouon
- 用 **Einsum** 层做转置
- 用 **Einsum** 层做求和规约
- 用 **Einsum** 层做点积
- 用 **Einsum** 层做矩阵乘法
- 用 **Einsum** 层做多张量缩并（尚不可用）[TODO]
- 用 **Einsum** 层取对角元素（尚不可用）[TODO]
- 省略号 (...) 用法（尚不可用）[TODO]

初始示例代码（用 **Einsum** 层做双张量单缩并）

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

nIn0, hIn0, wIn0 = 1, 3, 4 # 输入张量 NCHW
nIn1, hIn1, wIn1 = 2, 3, 5
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0) # 输入数据
data1 = np.arange(nIn1 * hIn1 * wIn1, dtype=np.float32).reshape(nIn1, hIn1, wIn1)

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 TensorRT 尝试优化, 单位 Byte
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn0, hIn0, wIn0)) # 双输入网络
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn1, hIn1, wIn1))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0, inputT1], "ijk,pjr->ikpr")
#-----# 替换部分
network.mark_output(einsumLayer.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("inputH1 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

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

```

- 输入张量形状 (1,3,4) 和 (2,3,5)

$$\left[\begin{bmatrix} 0. & 1. & 2. & 3. \\ 4. & 5. & 6. & 7. \\ 8. & 9. & 10. & 11. \end{bmatrix} \right], \left[\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 5. & 6. & 7. & 8. & 9. \\ 10. & 11. & 12. & 13. & 14. \end{bmatrix} \begin{bmatrix} 15. & 16. & 17. & 18. & 19. \\ 20. & 21. & 22. & 23. & 24. \\ 25. & 26. & 27. & 28. & 29. \end{bmatrix} \right]$$

- 输出张量形状 (1,4,2,5)

$$\left[\begin{bmatrix} 100. & 112. & 124. & 136. & 148. \\ 280. & 292. & 304. & 316. & 328. \\ 115. & 130. & 145. & 160. & 175. \\ 340. & 355. & 370. & 385. & 400. \\ 130. & 148. & 166. & 184. & 202. \\ 400. & 418. & 436. & 454. & 472. \\ 145. & 166. & 187. & 208. & 229. \\ 460. & 481. & 502. & 523. & 544. \end{bmatrix} \right]$$

- 计算过程: $A_{1 \times 3 \times 4}, B_{2 \times 3 \times 5}$ 关于长度为 3 的那一维度作缩并, 输出为 $E_{1 \times 4 \times 2 \times 5}$

$$\begin{aligned}
C_{1 \times 4 \times 3} &= A^{T(0,2,1)} \\
D_{1 \times 2 \times 4 \times 5} &= CB \\
E_{1 \times 4 \times 2 \times 5} &= D^{T(0,2,1,3)}
\end{aligned}$$

equation

```

einsumLayer = network.add_einsum([inputT0, inputT1], "ijk,ijq->ijq")
einsumLayer.equation = "ijk,pjr->ikpr"
设计算表达式

```

重

- 输出张量形状 (1,4,2,5), 结果与初始示例代码相同

用 Einsum 层做转置

```

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

```

```

nIn0, hIn0, wIn0 = 1, 3, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)

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, (nIn0, hIn0, wIn0)) # 单输入网络
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0], "ijk->jki")
#-----# 替换部分
network.mark_output(einsumLayer.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))
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 :", data0.shape)
print(data0)
print("outputH0:", outputH0.shape)
print(outputH0)

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

```

- 输入张量形状 (1,3,4)

$$\begin{bmatrix} \begin{bmatrix} 0. & 1. & 2. & 3. \end{bmatrix} \\ \begin{bmatrix} 4. & 5. & 6. & 7. \end{bmatrix} \\ \begin{bmatrix} 8. & 9. & 10. & 11. \end{bmatrix} \end{bmatrix}$$

- 输出张量形状 (3,4,1)，结果等价于 `inputH0.transpose(1,2,0)`

$$\begin{bmatrix} \begin{bmatrix} 0. \\ 1. \\ 2. \\ 3. \end{bmatrix} \\ \begin{bmatrix} 4. \\ 5. \\ 6. \\ 7. \end{bmatrix} \\ \begin{bmatrix} 8. \\ 9. \\ 10. \\ 11. \end{bmatrix} \end{bmatrix}$$

用 Einsum 层做求和规约

```
import numpy as np
```

```

from cuda import cudart
import tensorrt as trt

nIn0, hIn0, wIn0 = 1, 3, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)

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, (nIn0, hIn0, wIn0))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0], "ijk->ij")
#-----# 替换部分
network.mark_output(einsumLayer.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))
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 :", data0.shape)
print(data0)
print("outputH0:", outputH0.shape)
print(outputH0)

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

```

- 输入张量形状 (1,3,4)

$$\left[\begin{bmatrix} 0. & 1. & 2. & 3. \\ 4. & 5. & 6. & 7. \\ 8. & 9. & 10. & 11. \end{bmatrix} \right]$$

- 指定求和表达式 "ijk->ij", 输出张量形状 (1,3), "-" 右侧最后一维 k 消失, 即按该维求规约, 结果等价于 `np.sum(ipnutH0,axis=2)`

$$[6. \quad 22. \quad 38.]$$

- 指定求和表达式 "ijk->ik", 输出张量形状 (1,4), 结果等价于 `np.sum(ipnutH0,axis=1)`

$$[12. \quad 15. \quad 18. \quad 21.]$$

用 Einsum 层做点积

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

nIn0, hIn0, wIn0 = 1, 1, 4
nIn1, hIn1, wIn1 = 1, 1, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)
data1 = np.ones(nIn1 * hIn1 * wIn1, dtype=np.float32).reshape(nIn1, hIn1, wIn1) # 使用 ones 方便观察结果

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, (nIn0, hIn0, wIn0))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn1, hIn1, wIn1))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0, inputT1], "ijk,pqk->")
#-----# 替换部分
network.mark_output(einsumLayer.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("inputH1 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

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

- 输入张量形状 (1,1,4) 和 (1,1,4)

$\begin{bmatrix} 0. & 1. & 2. & 3. \end{bmatrix}, \begin{bmatrix} 1. & 1. & 1. & 1. \end{bmatrix}$

- 输出张量形状 ()

6.0

- 计算过程，“->”左侧相同下标（k）对应的维参与缩并运算（同维度内乘加），右侧消失的下标（i、j、p、q）都参与求和运算（跨维度求和）

$$0.0 * 1.0 + 0.1 * 1.0 + 0.2 * 1.0 + 0.3 * 1.0 = 6.0$$

修改输入数据形状

nIn0, hIn0, wIn0 = 1, 2, 4

nIn1, hIn1, wIn1 = 1, 3, 4

- 输入张量形状 (1,2,4) 和 (1,3,4)

$$\begin{bmatrix} \begin{bmatrix} 0. & 1. & 2. & 3. \end{bmatrix} \\ \begin{bmatrix} 4. & 5. & 6. & 7. \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \begin{bmatrix} 1. & 1. & 1. & 1. \end{bmatrix} \\ \begin{bmatrix} 1. & 1. & 1. & 1. \end{bmatrix} \\ \begin{bmatrix} 1. & 1. & 1. & 1. \end{bmatrix} \end{bmatrix}$$

- 输出张量形状 ()

84.0

- 计算过程，每个张量最后一维参与内积计算，一共有 2 * 3 组

$$6 + 6 + 6 + 22 + 22 + 22 = 84$$

同时修改输入数据形状和计算表达式

nIn0, hIn0, wIn0 = 1, 2, 4

nIn1, hIn1, wIn1 = 1, 3, 4

einsumLayer = network.add_einsum([inputT0, inputT1], "ijk,pqk->j")

- 输出张量形状 (2,)

[18. 66.]

- 计算过程，保留了 j = 2 这一下标不做加和，其他同上一个示例

$$6 + 6 + 6 = 18, 22 + 22 + 22 = 66$$

用 Einsum 层做矩阵乘法

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

nIn0, hIn0, wIn0 = 2, 2, 3
nIn1, hIn1, wIn1 = 2, 3, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)
data1 = np.ones(nIn1 * hIn1 * wIn1, dtype=np.float32).reshape(nIn1, hIn1, wIn1)

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, (nIn0, hIn0, wIn0))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn1, hIn1, wIn1))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0, inputT1], "ijk,ikl->ijl")
#-----# 替换部分
network.mark_output(einsumLayer.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("inputH1 :", data1.shape)
print(data1)
print("outputH0:", outputH0.shape)
print(outputH0)

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

```

- 输入矩阵形状 (2,2,3) 和 (2,3,4)

$$\left[\begin{bmatrix} 0. & 1. & 2. \\ 3. & 4. & 5. \end{bmatrix} \begin{bmatrix} 6. & 7. & 8. \\ 9. & 10. & 11. \end{bmatrix} \right], \left[\begin{bmatrix} 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \end{bmatrix} \begin{bmatrix} 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \end{bmatrix} \right]$$

- 输出矩阵形状 (2,2,4)

$$\left[\begin{bmatrix} 3. & 3. & 3. & 3. \\ 12. & 12. & 12. & 12. \end{bmatrix} \begin{bmatrix} 21. & 21. & 21. & 21. \\ 30. & 30. & 30. & 30. \end{bmatrix} \right]$$

- 计算过程，在最高 i 维（batch 维）保留，每个 batch 内做矩阵乘法

$$\begin{bmatrix} 0. & 1. & 2. \\ 3. & 4. & 5. \end{bmatrix} \begin{bmatrix} 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \end{bmatrix} = \begin{bmatrix} 3. & 3. & 3. & 3. \\ 12. & 12. & 12. & 12. \end{bmatrix}$$

用 Einsum 层做多张量缩并（尚不可用）

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

nIn0, hIn0, wIn0 = 1, 2, 3
nIn1, hIn1, wIn1 = 4, 3, 2
hIn2, wIn2 = 4, 5
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)
data1 = np.ones(nIn1 * hIn1 * wIn1, dtype=np.float32).reshape(nIn1, hIn1, wIn1)
data2 = np.ones(hIn2 * wIn2, dtype=np.float32).reshape(hIn2, wIn2)

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, (nIn0, hIn0, wIn0))
inputT1 = network.add_input('inputT1', trt.DataType.FLOAT, (nIn1, hIn1, wIn1))
inputT2 = network.add_input('inputT2', trt.DataType.FLOAT, (hIn2, wIn2))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0, inputT1, inputT2], "abc,dcb,de->ae")
#-----# 替换部分
network.mark_output(einsumLayer.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))
inputH2 = np.ascontiguousarray(data2.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("inputH0 :", data0.shape)
print(data0)
print("inputH1 :", data1.shape)
print(data1)
print("inputH2 :", data2.shape)
print(data2)
```



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

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

- TensorRT 8.2 中收到报错:

```
[TRT] [E] 3: [layers.cpp::EinsumLayer::5525] Error Code 3: API Usage Error (Parameter check failed at:
optimizer/api/layers.cpp::EinsumLayer::5525, condition: nbInputs > 0 && nbInputs <= MAX_EINSUM_NB_INPUTS
```

用 Einsum 层取对角元素（尚不可用）[TODO]

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

nIn0, hIn0, wIn0 = 1, 4, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)

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, (nIn0, hIn0, wIn0))
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0], "ijj->ij")
#-----# 替换部分
network.mark_output(einsumLayer.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))
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 :", data0.shape)
print(data0)
print("outputH0:", outputH0.shape)
print(outputH0)
```

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

- TensorRT 8.2 中收到报错:

```
[TRT] [E] 3: [layers.cpp::validateEquation::5611] Error Code 3: Internal Error ((Unnamed Layer* 0)
[Einsum]: Diagonal operations are not permitted in Einsum equation)
```

省略号 (...) 用法 (尚不可用) [TODO]

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

nIn0, hIn0, wIn0 = 1, 3, 4
data0 = np.arange(nIn0 * hIn0 * wIn0, dtype=np.float32).reshape(nIn0, hIn0, wIn0)

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, (nIn0, hIn0, wIn0)) # 单输入网络
#-----# 替换部分
einsumLayer = network.add_einsum([inputT0], "...j->...j")
#-----# 替换部分
network.mark_output(einsumLayer.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))
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 :", data0.shape)
print(data0)
print("outputH0:", outputH0.shape)
print(outputH0)

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

- TensorRT 8.2 中收到报错:

```
[TRT] [E] 3: [layers.cpp::validateEquation::5589] Error Code 3: Internal Error ((Unnamed Layer* 0)
[Einsum]: ellipsis is not permitted in Einsum equation)
```