Fully Connected 层

- 初始示例代码
- num_output_channels & kernel& bias
- set_input 用法

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
cOut = 2  # 输出张量 C
data = np.arange(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weight = np.ones(cIn * hIn * wIn, dtype=np.float32)  # 全连接权值
weight = np.concatenate([weight, -weight], 0).reshape(cOut, cIn, hIn, wIn)
bias = np.zeros(cOut, dtype=np.float32)  # 全连接偏置

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
fullyConnectedLayer = network.add_fully_connected(inputT0, cOut, weight, bias)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(fullyConnectedLayer.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 (1,2,1,1)

$$\left[\left[\begin{matrix} [[1770.]] \\ [[-1770.]] \end{matrix}\right]\right]$$

- 计算过程：$output = X \cdot W^T + bias$

$$\begin{aligned} &= X.reshape(nIn, cIn \cdot hIn \cdot wIn) * W.reshape(cOut, cIn \cdot hIn \cdot wIn).transpose() \\ &= \begin{bmatrix} 0 & 1 & 2 & \cdots & 59 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 1 & -1 \\ \cdots & \cdots \\ 1 & -1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1770 & -1770 \end{bmatrix} \end{aligned}$$

- Dynamic Shape 模式下，最低 3 维尺寸必须是构建期常量，不可为 -1

num_output_channels & kernel& bias

```
placeHolder = np.zeros(1, dtype=np.float32)
fullyConnectedLayer = network.add_fully_connected(inputT0, 1, placeHolder, placeHolder)
fullyConnectedLayer.num_output_channels = cOut  # 重设输出通道数
fullyConnectedLayer.kernel = weight  # 重设全连接权值
fullyConnectedLayer.bias = bias  # 重设全连接偏置, bias 为可选参数, 默认值 None
```

- 输出张量形状 (1,2,1,1)，结果与初始示例代码相同

set_input 用法

- 参考 link

```
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
cOut = 2  # 输出张量 C
data = np.arange(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weight = np.ones(cIn * hIn * wIn, dtype=np.float32)  # 全连接权值
weight = np.concatenate([weight, -weight], 0).reshape(cOut, cIn, hIn, wIn)
bias = np.zeros(cOut, dtype=np.float32)  # 全连接偏置

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
```

```python
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.flags = 1 << int(trt.BuilderFlag.INT8)  # 需要打开 int8 模式
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#----------------------------------------------------------- --------------------# 替换部分
constantLayer0 = network.add_constant([], np.array([1], dtype=np.float32))
constantLayer1 = network.add_constant([], np.array([1], dtype=np.float32))
quantizeLayer0 = network.add_quantize(inputT0, constantLayer0.get_output(0))
quantizeLayer0.axis = 0
dequantizeLayer0 = network.add_dequantize(quantizeLayer0.get_output(0), constantLayer1.get_output(0))
dequantizeLayer0.axis = 0

fullyConnectedLayer = network.add_fully_connected(dequantizeLayer0.get_output(0), cOut, weight, bias)
#----------------------------------------------------------- --------------------# 替换部分
network.mark_output(fullyConnectedLayer.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)
```

- 输出张量形状 (1,2,1,1)，结果与初始示例代码相同