RNN 层

- 初始示例代码
- seq_lengths
- num_layers & hidden_size & max_seq_length & data_length & op
- input_mode
- direction
- hidden_state
- cell_state（单层单向 LSTM 的例子）
- 单层双向 LSTM 的例子
- add_rnn（已废弃）及其参数 weight & bias
- 多层数的 RNN 的例子

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, 7  # 输入张量 NCHW
lenH = 5  # 隐藏层宽度
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weightX = np.ones((lenH, wIn), dtype=np.float32)  # 权重矩阵 (X->H)
weightH = np.ones((lenH, lenH), dtype=np.float32)  # 权重矩阵 (H->H)
biasX = np.zeros(lenH, dtype=np.float32)  # 偏置 (X->H)
biasH = np.zeros(lenH, dtype=np.float32)  # 偏置 (H->H)

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))  # 单输入示例代码
#------------------------------------------------------------ --------------------# 替换部分
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.RELU)  # 1 层 ReLU 型 RNN，隐藏层元素宽 lenH，序列长度 hIn，单词编码宽度 wIn，batchSize 为 cIn
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightX)  # 0 层 INPUT 门，输入元 X 变换阵，wX.shape=(lenH,wIn)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)  # 0 层 INPUT 门，隐藏元 H 变换阵，wH.shape=(lenH,lenH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)  # 0 层 INPUT 门，输入元 X 偏置，bX.shape=(lenH,)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)  # 0 层 INPUT 门，隐藏元 H 偏置，bH.shape=(lenH,)
#------------------------------------------------------------ --------------------# 替换部分
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
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)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)

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

- 输入张量形状 (1,3,4,7)，3 个独立输入，每个输入 4 个单词，每个单词 7 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\end{bmatrix}\\ \begin{bmatrix}1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\end{bmatrix}\\ \begin{bmatrix}1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\\ 1. & 1. & 1. & 1. & 1. & 1. & 1.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量 0 形状 (1,3,4,5)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标

$$\left[\left[\left[\begin{matrix}7. & 7. & 7. & 7. & 7.\\ 42. & 42. & 42. & 42. & 42.\\ 217. & 217. & 217. & 217. & 217.\\ 1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right.\right.$$
$$\left.\left.\left[\begin{matrix}7. & 7. & 7. & 7. & 7.\\ 42. & 42. & 42. & 42. & 42.\\ 217. & 217. & 217. & 217. & 217.\\ 1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right.\right.$$
$$\left.\left.\left[\begin{matrix}7. & 7. & 7. & 7. & 7.\\ 42. & 42. & 42. & 42. & 42.\\ 217. & 217. & 217. & 217. & 217.\\ 1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right]\right]$$

- 输出张量 1 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标

$$\left[\left[\left[\begin{matrix}1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right.\right.$$
$$\left.\left.\left[\begin{matrix}1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right.\right.$$
$$\left.\left.\left[\begin{matrix}1092. & 1092. & 1092. & 1092. & 1092.\end{matrix}\right]\right]\right]$$

- 计算过程：默认使用单向 RNN，隐藏状态全 0，并且要对输入张量作线性变换

$$\begin{aligned}
h_1 &= \mathbf{ReLU}\left(W_{i,X}\cdot x_1 + W_{i,H}\cdot h_0 + b_{i,X} + b_{i,H}\right)\\
&= \mathbf{ReLU}\left(\begin{bmatrix}1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\right)\\
&= \mathbf{ReLU}\left((7,7,7,7,7)^{\mathrm{T}}\right)\\
&= (7,7,7,7,7)^{\mathrm{T}}
\end{aligned}$$

$$\begin{aligned}
h_2 &= \mathbf{ReLU}\left(W_{i,X}\cdot x_2 + W_{i,H}\cdot h_1 + b_{i,X} + b_{i,H}\right)\\
&= \mathbf{ReLU}\left(\begin{bmatrix}1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\begin{bmatrix}7\\7\\7\\7\\7\\7\\7\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\right)\\
&= \mathbf{ReLU}\left((42,42,42,42,42)^{\mathrm{T}}\right)\\
&= (42,42,42,42,42)^{\mathrm{T}}
\end{aligned}$$

- 注意设置 config.max_workspace_size，否则可能报错：

```
[TensorRT] ERROR: 10: [optimizer.cpp::computeCosts::1855] Error Code 10: Internal Error (Could not find any implementation for node (Unnamed Layer* 0) [RNN].)
```

- 收到警告，以后 RNNV2 层可能被废弃

```
DeprecationWarning: Use addLoop instead.
```

seq_lengths

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

nIn, cIn, hIn, wIn = 1, 3, 4, 7
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)
seqLen = np.array([4, 3, 2], dtype=np.int32).reshape(nIn, cIn)  # 每个输入的真实长度
weightX = np.ones((lenH, wIn), dtype=np.float32)
weightH = np.ones((lenH, lenH), dtype=np.float32)
biasX = np.zeros(lenH, dtype=np.float32)
biasH = np.zeros(lenH, 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()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))  # 两输入示例代码
inputT1 = network.add_input('inputT1', trt.DataType.INT32, (nIn, cIn))
#------------------------------------------------------------ -------------------# 替换部分
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.RELU)
rnnV2Layer.seq_lengths = inputT1  # 设置每个独立输入的真实长度，默认均为 hIn
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightX)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)
#------------------------------------------------------------ -------------------# 替换部分
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
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))
inputH1 = np.ascontiguousarray(seqLen.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
outputH1 = 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)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.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), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)
```

```
print("inputH0 :", data.shape)
print(data)
print("inputH1 :", data.shape)
print(seqLen)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)

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

- 输出张量 0 形状 (1,3,4,5)，三个独立输入分别迭代 4 次、3 次和 2 次，长度不足的部分计算结果为 0

$$\left[\left[\begin{bmatrix} 7. & 7. & 7. & 7. & 7. \\ 42. & 42. & 42. & 42. & 42. \\ 217. & 217. & 217. & 217. & 217. \\ 1092. & 1092. & 1092. & 1092. & 1092. \end{bmatrix} \begin{bmatrix} 7. & 7. & 7. & 7. & 7. \\ 42. & 42. & 42. & 42. & 42. \\ 217. & 217. & 217. & 217. & 217. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} 7. & 7. & 7. & 7. & 7. \\ 42. & 42. & 42. & 42. & 42. \\ 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix}\right]\right]$$

- 输出张量 1 形状 (1,3,1,5)，3 个独立输出，记录每个独立输入的末状态

$$\left[\left[\begin{bmatrix} 1092. & 1092. & 1092. & 1092. & 1092. \end{bmatrix} \begin{bmatrix} 217. & 217. & 217. & 217. & 217. \end{bmatrix} \begin{bmatrix} 42. & 42. & 42. & 42. & 42. \end{bmatrix}\right]\right]$$

---

num_layers & hidden_size & max_seq_length & data_length & op

```
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.LSTM)  # 基于单输入示例代码
rnnV2Layer.op = trt.RNNOperation.RELU  # 重设 RNN 类型
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightX)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)
print("num_layers=%d\nhidden_size=%d\nmax_seq_length=%d\ndata_length=%d\n"% \
    (rnnV2Layer.num_layers,rnnV2Layer.hidden_size,rnnV2Layer.max_seq_length,rnnV2Layer.data_length))# 仅供输出，不能更改
```

- 结果与初始示例代码相同，多了 RNN 层的相关信息的输出：

```
num_layers=1
hidden_size=5
max_seq_length=4
data_length=7
```

- 可用的 RNN 模式

trt.RNNOperation 名	说明
RELU	单门 ReLU 激活 RNN
TANH	单门 tanh 激活 RNN
LSTM	4 门 LSTM
GRU	3 门 GRU

input_mode

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

nIn, cIn, hIn, wIn = 1, 3, 4, 5  #7
    # 输入 wIn 与 lenH 相等
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weightH = np.ones((lenH, lenH), dtype=np.float32)  # RNN 权重矩阵只剩 H->H 部分
biasX = np.zeros(lenH, dtype=np.float32)  # RNN 偏置仍然两个都要
biasH = np.zeros(lenH, 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()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))  # 单输入示例代码
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.RELU)  # 基于单输入示例代码
rnnV2Layer.input_mode = trt.RNNInputMode.SKIP  # 是否对输入张量线性变换，默认值 trt.RNNInputMode.LINEAR（需要线性变换）
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
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)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
```

```
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)

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

- 输出张量 0 形状 (1,3,4,5)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 6. & 6. & 6. & 6. & 6.\\ 31. & 31. & 31. & 31. & 31.\\ 156. & 156. & 156. & 156. & 156.\end{bmatrix}\\ \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 6. & 6. & 6. & 6. & 6.\\ 31. & 31. & 31. & 31. & 31.\\ 156. & 156. & 156. & 156. & 156.\end{bmatrix}\\ \begin{bmatrix}1. & 1. & 1. & 1. & 1.\\ 6. & 6. & 6. & 6. & 6.\\ 31. & 31. & 31. & 31. & 31.\\ 156. & 156. & 156. & 156. & 156.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量 1 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}156. & 156. & 156. & 156. & 156.\end{bmatrix}\\ \begin{bmatrix}156. & 156. & 156. & 156. & 156.\end{bmatrix}\\ \begin{bmatrix}156. & 156. & 156. & 156. & 156.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 计算过程：

$$
\begin{aligned}
h_1 &= \mathbf{ReLU}\left(x_1 + W_{i,H} \cdot h_0 + b_{i,X} + b_{i,H}\right) \\
&= \mathbf{ReLU}\left(\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\right) \\
&= \mathbf{ReLU}\left((1,1,1,1,1)^{\mathrm{T}}\right) \\
&= (1,1,1,1,1,)^{\mathrm{T}}
\end{aligned}
$$

$$
\begin{aligned}
h_2 &= \mathbf{ReLU}\left(x_2 + W_{i,H} \cdot h_1 + b_{i,X} + b_{i,H}\right) \\
&= \mathbf{ReLU}\left(\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}\right) \\
&= \mathbf{ReLU}\left((6,6,6,6,6)^{\mathrm{T}}\right) \\
&= (6,6,6,6,6)^{\mathrm{T}}
\end{aligned}
$$

- 注意，可用的输入张量处理方法

trt.RNNInputMode 名	说明
LINEAR	对输入张量 x 做线性变换
SKIP	不对输入张量 x 做线性变换（要求wIn == lenH）

- 在 SKIP 模式（不做输入张量线性变换）的情况下，仍然需要设置 biasX，否则报错：

```
[TensorRT] ERROR: Missing weights/bias: (0, INPUT, 1)
[TensorRT] ERROR: 4: [network.cpp::validate::2639] Error Code 4: Internal Error (Layer (Unnamed Layer*
0) [RNN] failed validation)
```

---

direction

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

nIn, cIn, hIn, wIn = 1, 3, 4, 7
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weightFX = np.ones((wIn, lenH), dtype=np.float32)  # 正向权重矩阵 (X->H)
weightFH = np.ones((lenH, lenH), dtype=np.float32)  # 正向权重矩阵 (H->H)
weightBX = np.ones((wIn, lenH), dtype=np.float32)  # 反向权重矩阵 (X->H)
weightBH = np.ones((lenH, lenH), dtype=np.float32)  # 反向权重矩阵 (H->H)
biasFX = np.zeros(lenH, dtype=np.float32)  # 正向偏置 (X->H)
biasFH = np.zeros(lenH, dtype=np.float32)  # 正向偏置 (H->H)
biasBX = np.zeros(lenH, dtype=np.float32)  # 反向偏置 (X->H)
biasBH = np.zeros(lenH, dtype=np.float32)  # 反向偏置 (H->H)
```

```
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))  # 单输入示例代码
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.RELU)  # 基于单输入示例代码
rnnV2Layer.direction = trt.RNNDirection.BIDIRECTION  # RNN 方向，默认值 trt.RNNDirection.UNIDIRECTION 为单向
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightFX)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightFH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasFX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasFH)
rnnV2Layer.set_weights_for_gate(1, trt.RNNGateType.INPUT, True, weightBX)  # 反向为第 1 层
rnnV2Layer.set_weights_for_gate(1, trt.RNNGateType.INPUT, False, weightBH)
rnnV2Layer.set_bias_for_gate(1, trt.RNNGateType.INPUT, True, biasBX)
rnnV2Layer.set_bias_for_gate(1, trt.RNNGateType.INPUT, False, biasBH)
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
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)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)

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

- 输出张量 0 形状 (1,3,4,10)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标，2 个方向在同一行并排放置

$$\left[\left[\begin{bmatrix} 7. & 7. & 7. & 7. & 7. & 1092. & 1092. & 1092. & 1092. & 1092. \\ 42. & 42. & 42. & 42. & 42. & 217. & 217. & 217. & 217. & 217. \\ 217. & 217. & 217. & 217. & 217. & 42. & 42. & 42. & 42. & 42. \\ 1092. & 1092. & 1092. & 1092. & 1092. & 7. & 7. & 7. & 7. & 7. \end{bmatrix} \begin{bmatrix} 7. & 7. & 7. & 7. & 7. & 1092. & 1092. & 1092. & 1092. & 1092. \\ 42. & 42. & 42. & 42. & 42. & 217. & 217. & 217. & 217. & 217. \\ 217. & 217. & 217. & 217. & 217. & 42. & 42. & 42. & 42. & 42. \\ 1092. & 1092. & 1092. & 1092. & 1092. & 7. & 7. & 7. & 7. & 7. \end{bmatrix} \begin{bmatrix} 7. & 7. & 7. & 7. & 7. & 1092. & 1092. & 1092. & 1092. & 1092. \\ 42. & 42. & 42. & 42. & 42. & 217. & 217. & 217. & 217. & 217. \\ 217. & 217. & 217. & 217. & 217. & 42. & 42. & 42. & 42. & 42. \\ 1092. & 1092. & 1092. & 1092. & 1092. & 7. & 7. & 7. & 7. & 7. \end{bmatrix}\right]\right]$$

- 输出张量 1 形状 (1,3,2,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标，2 个方向分行放置

$$\left[\left[\begin{bmatrix} 1092. & 1092. & 1092. & 1092. & 1092. \\ 1092. & 1092. & 1092. & 1092. & 1092. \end{bmatrix} \begin{bmatrix} 1092. & 1092. & 1092. & 1092. & 1092. \\ 1092. & 1092. & 1092. & 1092. & 1092. \end{bmatrix} \begin{bmatrix} 1092. & 1092. & 1092. & 1092. & 1092. \\ 1092. & 1092. & 1092. & 1092. & 1092. \end{bmatrix}\right]\right]$$

- 可用方向参数

trt.RNNDirection 名	说明
UNIDIRECTION	单向 RNN
BIDIRECTION	双向 RNN

---

hidden_state

```python
h0 = network.add_constant((cIn, 1, lenH), np.ones((cIn, 1, lenH), dtype=np.float32))  # 初始隐藏状态
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.RELU)  # 基于单输入初始示例代码
rnnV2Layer.hidden_state = h0.get_output(0)  # 设置初始隐藏状态，默认为全 0
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightX)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)
```

- 输出张量 0 形状 (1,3,4,5)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标

$$\left[\left[\begin{bmatrix} 12. & 12. & 12. & 12. & 12. \\ 67. & 12. & 12. & 12. & 12. \\ 342. & 342. & 342. & 342. & 342. \\ 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix} \begin{bmatrix} 12. & 12. & 12. & 12. & 12. \\ 67. & 67. & 67. & 67. & 67. \\ 342. & 342. & 342. & 342. & 342. \\ 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix} \begin{bmatrix} 12. & 12. & 12. & 12. & 12. \\ 67. & 67. & 67. & 67. & 67. \\ 342. & 342. & 342. & 342. & 342. \\ 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix}\right]\right]$$

- 输出张量 1 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标

$$\left[\left[\begin{bmatrix} 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix} \begin{bmatrix} 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix} \begin{bmatrix} 1717. & 1717. & 1717. & 1717. & 1717. \end{bmatrix}\right]\right]$$

- 计算过程：

$$
\begin{aligned}
h_1 &= \mathbf{ReLU}\left(W_{i,X}\cdot x_1 + W_{i,H}\cdot h_0 + b_{i,X} + b_{i,H}\right) \\
&= \mathbf{ReLU}\left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right) \\
&= \mathbf{ReLU}\left((12, 12, 12, 12, 12)^{\mathrm{T}}\right) \\
&= (12, 12, 12, 12, 12)^{\mathrm{T}}
\end{aligned}
$$

$$
\begin{aligned}
h_2 &= \mathbf{ReLU}\left(W_{i,X}\cdot x_2 + W_{i,H}\cdot h_1 + b_{i,X} + b_{i,H}\right) \\
&= \mathbf{ReLU}\left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}\begin{bmatrix} 12 \\ 12 \\ 12 \\ 12 \\ 12 \\ 12 \\ 12 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right) \\
&= \mathbf{ReLU}\left((67, 67, 67, 67, 67)^{\mathrm{T}}\right) \\
&= (67, 67, 67, 67, 67)^{\mathrm{T}}
\end{aligned}
$$

---

cell_state（单层单向 LSTM 的例子）

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

nIn, cIn, hIn, wIn = 1, 3, 4, 7  # 输入张量 NCHW
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weightAllX = np.ones((lenH, wIn), dtype=np.float32)  # 权重矩阵 (X->H)
weightAllH = np.ones((lenH, lenH), dtype=np.float32)  # 权重矩阵 (H->H)
biasAllX = np.zeros(lenH, dtype=np.float32)  # 偏置 (X->H)
biasAllH = np.zeros(lenH, dtype=np.float32)  # 偏置 (H->H)

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))
h0 = network.add_constant((cIn, 1, lenH), np.ones((cIn, 1, lenH), dtype=np.float32))  # 初始隐藏状态
c0 = network.add_constant((cIn, 1, lenH), np.zeros((cIn, 1, lenH), dtype=np.float32))  # 初始细胞状态
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.LSTM)  # 基于单输入初始示例代码
rnnV2Layer.hidden_state = h0.get_output(0)
rnnV2Layer.cell_state = c0.get_output(0)  # 设置初始细胞状态，默认为全 0

for kind in [trt.RNNGateType.INPUT, trt.RNNGateType.CELL, trt.RNNGateType.FORGET,
trt.RNNGateType.OUTPUT]:
```

```
    rnnV2Layer.set_weights_for_gate(0, kind, True, weightAllX)
    rnnV2Layer.set_weights_for_gate(0, kind, False, weightAllH)
    rnnV2Layer.set_bias_for_gate(0, kind, True, biasAllX)
    rnnV2Layer.set_bias_for_gate(0, kind, False, biasAllH)
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
network.mark_output(rnnV2Layer.get_output(2))  # 多了一个最终细胞状态可以输出
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)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
outputH2 = np.empty(context.get_binding_shape(3), dtype=trt.nptype(engine.get_binding_dtype(3)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)
_, outputD2 = cudart.cudaMallocAsync(outputH2.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1), int(outputD2)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH2.ctypes.data, outputD2, outputH2.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)
print("outputH2:", outputH2.shape)
print(outputH2)

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

- 输出张量 0 形状 (1,3,4,5)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517\\ 0.963940 & 0.963940 & 0.963940 & 0.963940 & 0.963940\\ 0.995037 & 0.995037 & 0.995037 & 0.995037 & 0.995037\\ 0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321\end{bmatrix}\\ \begin{bmatrix}0.760517 & 0.760517 & 0.760517 & 0.7605171 & 0.7605171\\ 0.963940 & 0.963940 & 0.963940 & 0.963940 & 0.963940\\ 0.995037 & 0.995037 & 0.995037 & 0.995037 & 0.995037\\ 0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321\end{bmatrix}\\ \begin{bmatrix}0.760517 & 0.760517 & 0.760517 & 0.7605171 & 0.7605171\\ 0.963940 & 0.963940 & 0.963940 & 0.963940 & 0.963940\\ 0.995037 & 0.995037 & 0.995037 & 0.995037 & 0.995037\\ 0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量 1 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}[0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321]\\ [0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321]\\ [0.999321 & 0.999321 & 0.999321 & 0.999321 & 0.999321]\end{bmatrix}\end{bmatrix}$$

- 输出张量 2 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终细胞状态，每个细胞状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}[3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999]\\ [3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999]\\ [3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999]\end{bmatrix}\end{bmatrix}$$

- 计算过程：

$$\begin{aligned}
I_1 = F_1 = O_1 &= \mathbf{sigmoid}\left(W_{?,X}\cdot x_1 + W_{?,H}\cdot h_0 + b_{i,X} + b_{i,H}\right) = (0.999088, 0.999088, 0.999088, 0.999088, 0.999088)^{\mathrm{T}}\\
C_1 &= \mathbf{tanh}\left(W_{C,X}\cdot x_1 + W_{C,H}\cdot h_0 + b_{C,X} + b_{C,H}\right) = (0.999998, 0.999998, 0.999998, 0.999998, 0.999998)^{\mathrm{T}}\\
c_1 &= F_1 \cdot c_0 + I_1 \cdot C_1 = (0.999087, 0.999087, 0.999087, 0.999087, 0.999087)^{\mathrm{T}}\\
h_1 &= O_1 \cdot \mathbf{tanh}\left(c_1\right) = (0.760517, 0.760517, 0.760517, 0.760517, 0.760517)^{\mathrm{T}}
\end{aligned}$$

$$\begin{aligned}
I_2 = F_2 = O_2 &= \mathbf{sigmoid}\left(W_{?,X}\cdot x_2 + W_{?,H}\cdot h_1 + b_{i,X} + b_{i,H}\right) = (0.999979, 0.999979, 0.999979, 0.999979, 0.999979)^{\mathrm{T}}\\
C_2 &= \mathbf{tanh}\left(W_{C,X}\cdot x_2 + W_{C,H}\cdot h_1 + b_{C,X} + b_{C,H}\right) = (0.999999, 0.999999, 0.999999, 0.999999, 0.999999)^{\mathrm{T}}\\
c_2 &= F_2 \cdot c_1 + I_2 \cdot C_2 = (1.999046, 1.999046, 1.999046, 1.999046, 1.999046)^{\mathrm{T}}\\
h_2 &= O_2 \cdot \mathbf{tanh}\left(c_2\right) = (0.963940, 0.963940, 0.963940, 0.963940, 0.963940)^{\mathrm{T}}
\end{aligned}$$

单层双向 LSTM 的例子

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

nIn, cIn, hIn, wIn = 1, 3, 4, 7  # 输入张量 NCHW
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)  # 输入数据
weightAllX = np.ones((lenH, wIn), dtype=np.float32)  # 权重矩阵 (X->H)
weightAllH = np.ones((lenH, lenH), dtype=np.float32)  # 权重矩阵 (H->H)
biasAllX = np.zeros(lenH, dtype=np.float32)  # 偏置 (X->H)
biasAllH = np.zeros(lenH, dtype=np.float32)  # 偏置 (H->H)

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))
h0 = network.add_constant((cIn, 2, lenH), np.ones((cIn, 2, lenH), dtype=np.float32))  # 初始隐藏状态变成 2 行
c0 = network.add_constant((cIn, 2, lenH), np.zeros((cIn, 2, lenH), dtype=np.float32))  # 初始细胞状态变成 2 行
rnnV2Layer = network.add_rnn_v2(inputT0, 1, lenH, hIn, trt.RNNOperation.LSTM)  # 基于单输入初始示例代码
rnnV2Layer.direction = trt.RNNDirection.BIDIRECTION
rnnV2Layer.hidden_state = h0.get_output(0)
rnnV2Layer.cell_state = c0.get_output(0)
for layer in range(2):
    for kind in [trt.RNNGateType.INPUT, trt.RNNGateType.CELL, trt.RNNGateType.FORGET,
trt.RNNGateType.OUTPUT]:
        rnnV2Layer.set_weights_for_gate(layer, kind, True, weightAllX)
        rnnV2Layer.set_weights_for_gate(layer, kind, False, weightAllH)
        rnnV2Layer.set_bias_for_gate(layer, kind, True, biasAllX)
        rnnV2Layer.set_bias_for_gate(layer, kind, False, biasAllH)
network.mark_output(rnnV2Layer.get_output(0))
network.mark_output(rnnV2Layer.get_output(1))
network.mark_output(rnnV2Layer.get_output(2))  # 多了一个最终细胞状态可以输出
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)))
outputH1 = np.empty(context.get_binding_shape(2), dtype=trt.nptype(engine.get_binding_dtype(2)))
outputH2 = np.empty(context.get_binding_shape(3), dtype=trt.nptype(engine.get_binding_dtype(3)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)
_, outputD2 = cudart.cudaMallocAsync(outputH2.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0), int(outputD1), int(outputD2)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH2.ctypes.data, outputD2, outputH2.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)
print("outputH2:", outputH2.shape)
print(outputH2)

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

- 输出张量 0 形状 (1,3,4,10)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标，2 个方向在同一行并排放置

$$\left[\left[\begin{bmatrix} 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 \\ 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 \end{bmatrix}\right.\right.$$
$$\left.\left.\begin{bmatrix} 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 \\ 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 \end{bmatrix}\right.\right.$$
$$\left.\left.\begin{bmatrix} 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 \\ 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.995038 & 0.963941 & 0.963941 & 0.963941 & 0.963941 & 0.963941 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.760517 & 0.760517 & 0.760517 & 0.760517 & 0.760517 \end{bmatrix}\right]\right]$$

- 输出张量 1 形状 (1,3,2,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标，2 个方向分行放置

$$\left[\left[\begin{bmatrix} 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \end{bmatrix}\right.\right.$$
$$\left.\left.\begin{bmatrix} 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \end{bmatrix}\right.\right.$$
$$\left.\left.\begin{bmatrix} 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \\ 0.999322 & 0.999322 & 0.999322 & 0.999322 & 0.999322 \end{bmatrix}\right]\right]$$

- 输出张量 2 形状 (1,3,1,5)，3 个独立输出，每个包含 1 个最终细胞状态，每个细胞状态 5 维坐标，2 个方向分行放置

$$\left[\begin{bmatrix} 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \\ 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \end{bmatrix}\right.$$
$$\left.\begin{bmatrix} 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \\ 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \end{bmatrix}\right.$$
$$\left.\begin{bmatrix} 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \\ 3.998999 & 3.998999 & 3.998999 & 3.998999 & 3.998999 \end{bmatrix}\right]$$

---

add_rnn（已废弃）及其参数 weight & bias

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

nIn, cIn, hIn, wIn = 1, 3, 4, 7
lenH = 5
data = np.ones(cIn * hIn * wIn, dtype=np.float32).reshape(cIn, hIn, wIn)
weight = np.ones((lenH, wIn + lenH), dtype=np.float32)  # 权重矩阵，X 和 H 连接在一起
bias = np.zeros(lenH * 2, dtype=np.float32)  # 偏置，bX 和 bH 连接在一起

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()  # 必须使用 implicit batch 模式
config = builder.create_builder_config()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (cIn, hIn, wIn))
shuffleLayer = network.add_shuffle(inputT0)  # 先 shuffle 成 (hIn,cIn,wIn)
shuffleLayer.first_transpose = (1, 0, 2)
fakeWeight = np.random.rand(lenH, wIn + lenH).astype(np.float32)
```

```
fakeBias = np.random.rand(lenH * 2).astype(np.float32)
rnnLayer    = network.add_rnn(shuffleLayer.get_output(0), 1, lenH, hIn, trt.RNNOperation.RELU,\
                              trt.RNNInputMode.LINEAR, trt.RNNDirection.UNIDIRECTION, fakeWeight,
fakeBias)
rnnLayer.weights = weight  # 重设 RNN 权重
rnnLayer.bias = bias  # 重设 RNN 偏置
network.mark_output(rnnLayer.get_output(0))
network.mark_output(rnnLayer.get_output(1))
engine = builder.build_engine(network, config)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty((nIn, ) + tuple(context.get_binding_shape(1)),
dtype=trt.nptype(engine.get_binding_dtype(1)))
outputH1 = np.empty((nIn, ) + tuple(context.get_binding_shape(2)),
dtype=trt.nptype(engine.get_binding_dtype(2)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)
_, outputD1 = cudart.cudaMallocAsync(outputH1.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async(nIn, [int(inputD0), int(outputD0), int(outputD1)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaMemcpyAsync(outputH1.ctypes.data, outputD1, outputH1.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)
print("outputH1:", outputH1.shape)
print(outputH1)

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

- 输出张量 0 形状 (1,4,3,5)，3 个独立输出，每个输出 4 个隐藏状态，每个隐藏状态 5 维坐标，结果与初始示例代码相同，只是结果放置方式不同

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}7. & 7. & 7. & 7. & 7.\\ 7. & 7. & 7. & 7. & 7.\\ 7. & 7. & 7. & 7. & 7.\end{bmatrix}\\ \begin{bmatrix}42. & 42. & 42. & 42. & 42.\\ 42. & 42. & 42. & 42. & 42.\\ 42. & 42. & 42. & 42. & 42.\end{bmatrix}\\ \begin{bmatrix}217. & 217. & 217. & 217. & 217.\\ 217. & 217. & 217. & 217. & 217.\\ 217. & 217. & 217. & 217. & 217.\end{bmatrix}\\ \begin{bmatrix}1092. & 1092. & 1092. & 1092. & 1092.\\ 1092. & 1092. & 1092. & 1092. & 1092.\\ 1092. & 1092. & 1092. & 1092. & 1092.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量 1 形状 (1,1,3,5)，3 个独立输出，每个包含 1 个最终隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}1092. & 1092. & 1092. & 1092. & 1092.\\ 1092. & 1092. & 1092. & 1092. & 1092.\\ 1092. & 1092. & 1092. & 1092. & 1092.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

多层数的 RNN 的例子

```
rnnV2Layer = network.add_rnn_v2(inputT0, 2, lenH, hIn, trt.RNNOperation.RELU)  # 2 层 ReLU 型 RNN
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, True, weightX)
rnnV2Layer.set_weights_for_gate(0, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, True, biasX)
rnnV2Layer.set_bias_for_gate(0, trt.RNNGateType.INPUT, False, biasH)

rnnV2Layer.set_weights_for_gate(1, trt.RNNGateType.INPUT, True, weightH)  # 第二层的权重，注意尺寸与隐藏层相等
rnnV2Layer.set_weights_for_gate(1, trt.RNNGateType.INPUT, False, weightH)
rnnV2Layer.set_bias_for_gate(1, trt.RNNGateType.INPUT, True, biasH)
rnnV2Layer.set_bias_for_gate(1, trt.RNNGateType.INPUT, False, biasH)
```

- 输出张量 0 形状 (1,3,4,5)，3 个独立输出，每个包含 4 个隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}35. & 35. & 35. & 35. & 35.\\ 385. & 385. & 385. & 385. & 385.\\ 3010. & 3010. & 3010. & 3010. & 3010.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\\ \begin{bmatrix}35. & 35. & 35. & 35. & 35.\\ 385. & 385. & 385. & 385. & 385.\\ 3010. & 3010. & 3010. & 3010. & 3010.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\\ \begin{bmatrix}35. & 35. & 35. & 35. & 35.\\ 385. & 385. & 385. & 385. & 385.\\ 3010. & 3010. & 3010. & 3010. & 3010.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- 输出张量 1 形状 (1,3,2,5)，3 个独立输出，每个包含各层的最终隐藏状态，每个隐藏状态 5 维坐标

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}1092. & 1092. & 1092. & 1092. & 1092.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\\ \begin{bmatrix}1092. & 1092. & 1092. & 1092. & 1092.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\\ \begin{bmatrix}1092. & 1092. & 1092. & 1092. & 1092.\\ 20510. & 20510. & 20510. & 20510. & 20510.\end{bmatrix}\end{bmatrix}\end{bmatrix}$$