MEMORY MANAGEMENT IN NVIDIA :

TensorRT stores weights and activations on GPUs. The size of each engine stored in the cache of `TRTEngineOp` is about the same as the size of weights. TensorRT allocates memory through TensorFlow allocators, therefore, all TensorFlow memory configurations also apply to TensorRT. For example, if the TensorFlow session configuration `config.gpu_options.per_process_gpu_memory_fraction` is set to `0.3`, it means 30% of the GPU memory is allocated by TensorFlow to be used for all of its internal usage including TF-TRT and TensorRT. That means if TensorRT asks TensorFlow to allocate memory with the amount more than what is available to TensorFlow, then it will run out of memory.

On top of the memory used for weights and activations, certain TensorRT algorithms also require temporary workspace. The argument `max_workspace_size_bytes` limits the maximum size that TensorRT can use for the workspace. The default value is 1GB. If the value is too small, TensorRT will not be able to use certain algorithms that need that much workspace and that may lead to poor performance. The workspace is also allocated through TensorFlow allocators.

Although TensorRT is allowed to use algorithms that require at most `max_workspace_size_bytes` amount of workspace, but the maximum workspace requirement of all the TensorRT algorithms may still be smaller than `max_workspace_size_bytes` (meaning, TensorRT may not have any algorithm that needs that much workspace). In such cases, TensorRT only allocates the needed workspace instead of allocating how much the user specifies.

If you observe your inference running out of memory or you want to experiment with whether you can get better performance by using more memory, then try to increase `config.gpu_options.per_process_gpu_memory_fraction` and `max_workspace_size_bytes`. The memory usage highly depends on your model and it’s hard to predict a suitable default for `max_workspace_size_bytes.`

2.9. INT8 Quantization

In order to use INT8 precision, the weights and activations of the model need to be quantized so that floating point values can be converted into integers using appropriate ranges. There are different calibration algorithms which can be used to perform the quantization after the model is trained. You can use such algorithms to compute the quantization ranges and then feed those custom ranges into TF-TRT.

TensorRT also provides a quantization algorithm which is integrated into TF-TRT and can be used without worrying about feeding ranges into TF-TRT. It’s also possible to quantize the model during training (quantization-aware training) and then feed the ranges into TF-TRT. Since quantization-aware training requires many considerations, we recommend that you use the TensorRT calibration algorithm instead.

2.9.1. Post-Training Quantization Using TensorRT Calibration

TensorRT provides a calibration algorithm which can quantize the weights and activations after training. In the first step, TF-TRT runs the calibration algorithm which results in a calibration table, and in the second step the calibrated model is converted to a graph that is ready to be used by inference. The following code snippet shows how to use this method:

```
from tensorflow.python.compiler.tensorrt import trt_convert as trt

converter = trt.TrtGraphConverter(

       input_graph_def=frozen_graph,

       nodes_blacklist=['logits', 'classes'],

      precision_mode='INT8',

      use_calibration=True)

frozen_graph = converter.convert()

frozen_graph = converter.calibrate(

     fetch_names=['logits', 'classes'],

     num_runs=num_calib_inputs // batch_size,

     input_map_fn=input_map_fn)
```

The function `input_map_fn` should return a dictionary mapping input names (as strings) in the GraphDef to be calibrated to Tensor objects. The values of the named input tensors in the GraphDef to be calibrated will be re-mapped to the respective `Tensor` values during calibration. The following is an example of such a function:

```
dataset = tf.data.TFRecordDataset(data_files)

iterator = dataset.make_one_shot_iterator()

features = iterator.get_next()

def input_map_fn():

     return {'input:0': features}
```

The following code snippet shows how to extract the TensorRT calibration table after the calibration is done:

```
for n in trt_graph.node:

  if n.op == "TRTEngineOp":

    print("Node: %s, %s" % (n.op, n.name.replace("/", "_")))

    with tf.gfile.GFile("%s.calib_table" % (n.name.replace("/", "_")),
'wb') as f:

      f.write(n.attr["calibration_data"].s)
```

2.9.2. Post-Training Quantization Using Custom Quantization Ranges

TF-TRT also allows you to supply your own quantization ranges in case you do not want to use TensorRT's built-in calibrator. To do so, augment your TensorFlow model with quantization nodes to provide the converter with the floating point range for each tensor.

You can use any of the following TensorFlow ops to provide quantization ranges:

- `QuantizeAndDequantizeV2`
- `QuantizeAndDequantizeV3`
- `FakeQuantWithMinMaxVars`
- `FakeQuantWithMinMaxArgs`

You should then call TF-TRT in the following way:

```
from tensorflow.python.compiler.tensorrt import trt_convert as trt

converter = trt.TrtGraphConverter(

      input_graph_def=frozen_graph,

      nodes_blacklist=['logits', 'classes'],

     precision_mode='INT8',

     use_calibration=False)

frozen_graph = converter.convert()
```

The following code snippet shows a simple hypothetical TensorFlow graph which has been augmented using `QuantizeAndDequantizeV2` ops to include quantization ranges which can be read by TF-TRT.

This particular graph has inputs which range from -1 to 1, so we set the quantization range for the input tensor to [-1. 1].

The output of this particular matmul op has been measured to fit mostly between -9 to 9, so the quantization range for that tensor is set accordingly.

Finally, the output of this bias_add op has been measured to range from -3 to 3, therefore quantization range of the output tensor is set to [-3, 3].

Note: TensorRT only supports symmetric quantization ranges.

```
def my_graph(x):

  x = tf.quantize_and_dequantize_v2(x, input_min=-1.0, input_max=1.0)

  x = tf.matmul(x, kernel)

  x = tf.quantize_and_dequantize_v2(x, input_min=-9.0, input_max=9.0)

  x = tf.nn.bias_add(x, bias)

  x = tf.quantize_and_dequantize_v2(x, input_min=-3.0, input_max=3.0)

  return x
```

TensorRT may decide to fuse some operations in your graph. If you have provided a quantization range for a tensor which is removed due to fusion, your unnecessary range will be ignored.

You may also provide custom quantization ranges for *some* tensors and still use the TensorRT calibration to determine the rest of the ranges. To do this, provide quantization ranges in your TensorFlow model as described above using the supported quantization ops and perform the TensorRT calibration procedure as described in Post-Training Quantization Using TensorRT Calibration (with use_calibration=True).