使用Python训练和部署低精度模型

（TensorFlow版）

张校捷

2019/9/21

目录 CONTENTS

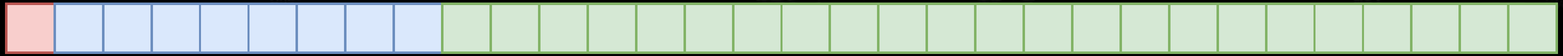
1 低精度的概念和意义

实数的16-bit半精度浮点数和8-bit定点数表示

使用低精度的意义

深度学习模型中实数的表示

FP32: E8M23

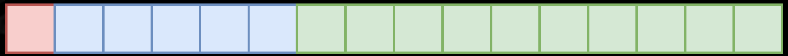


(tf.float32)

FP16: E8M7

(TPU, tf.bfloat16)

FP16: E5M10



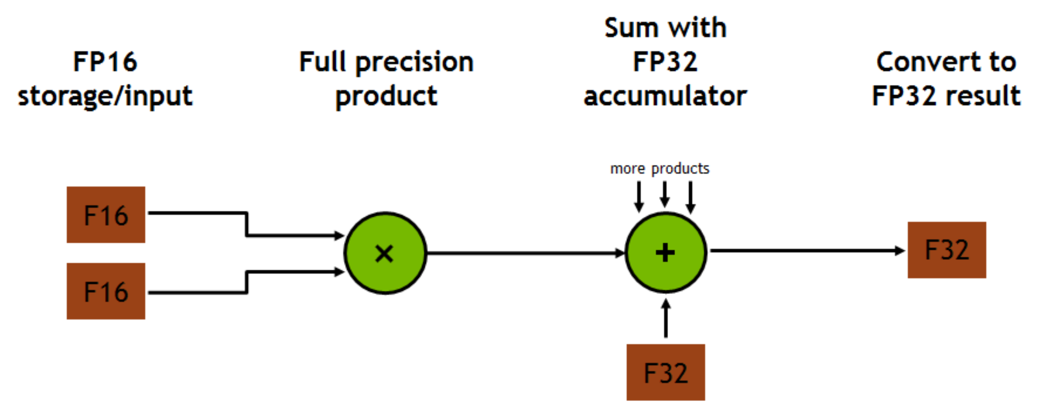
(GPU, tf.float16)

Int8

低精度浮点数的优点

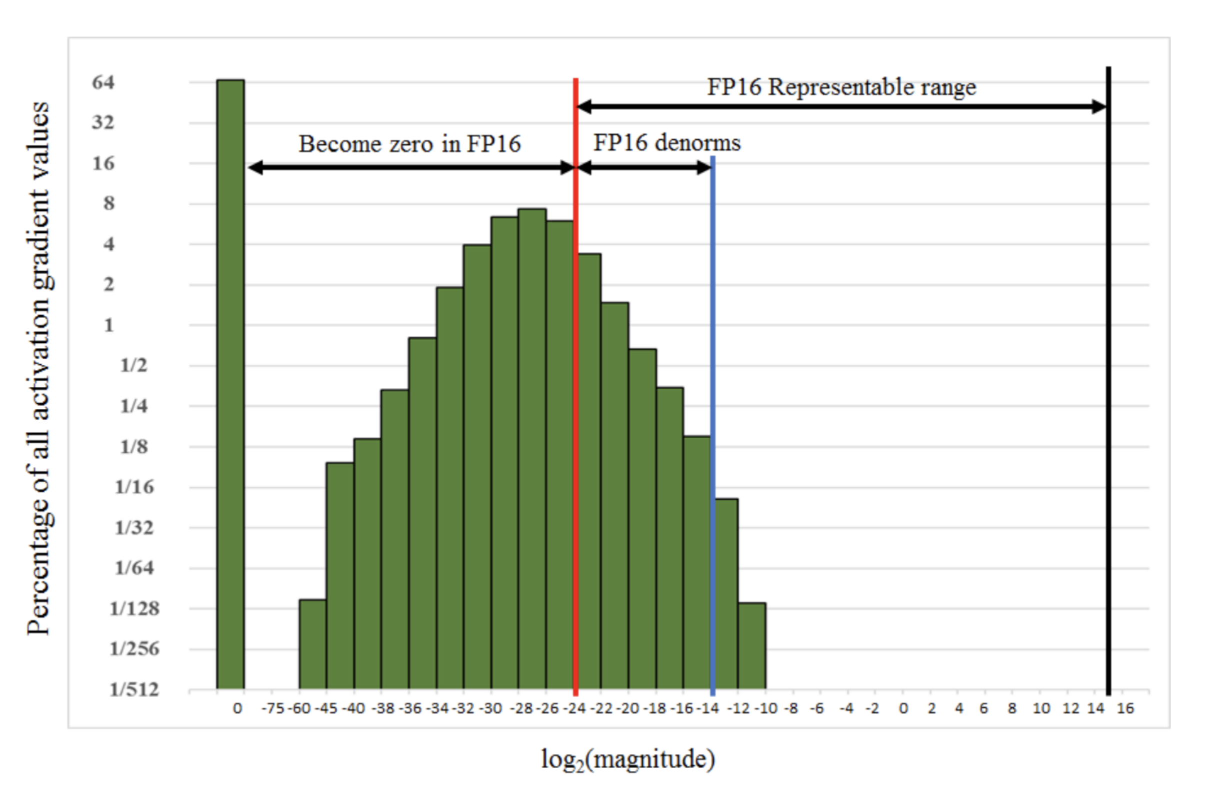
1.节约内存/显存的使用（FP16为原来的1/2，int8为原来的1/4）

2.特殊的硬件专门用于低精度浮点数的计算加速（TensorCore）

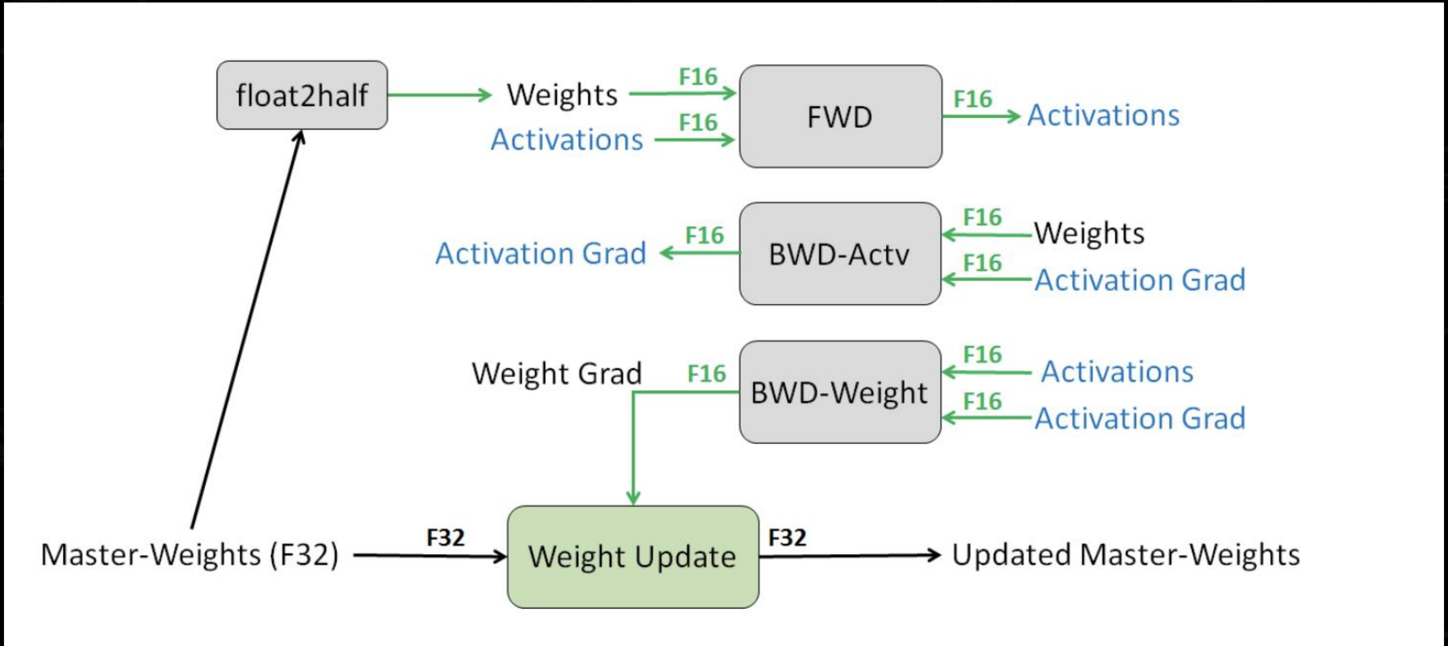


Model	Speedup
BERT Q&A	3.3X speedup
GNMT	1.7X speedup
NCF	2.6X speedup
ResNet-50-v1.5	3.3X speedup
SSD-RN50-FPN-640	2.5X speedup

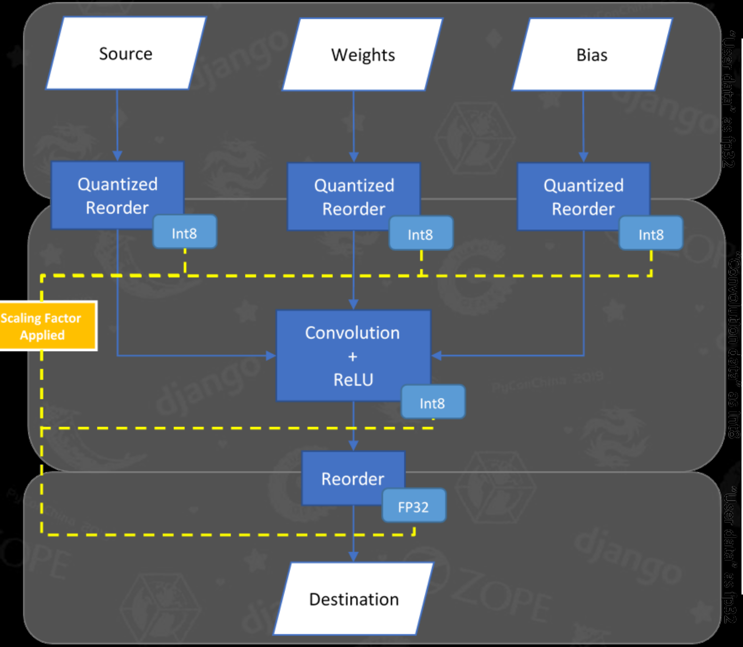
FP16浮点数（E5M10）的表示范围

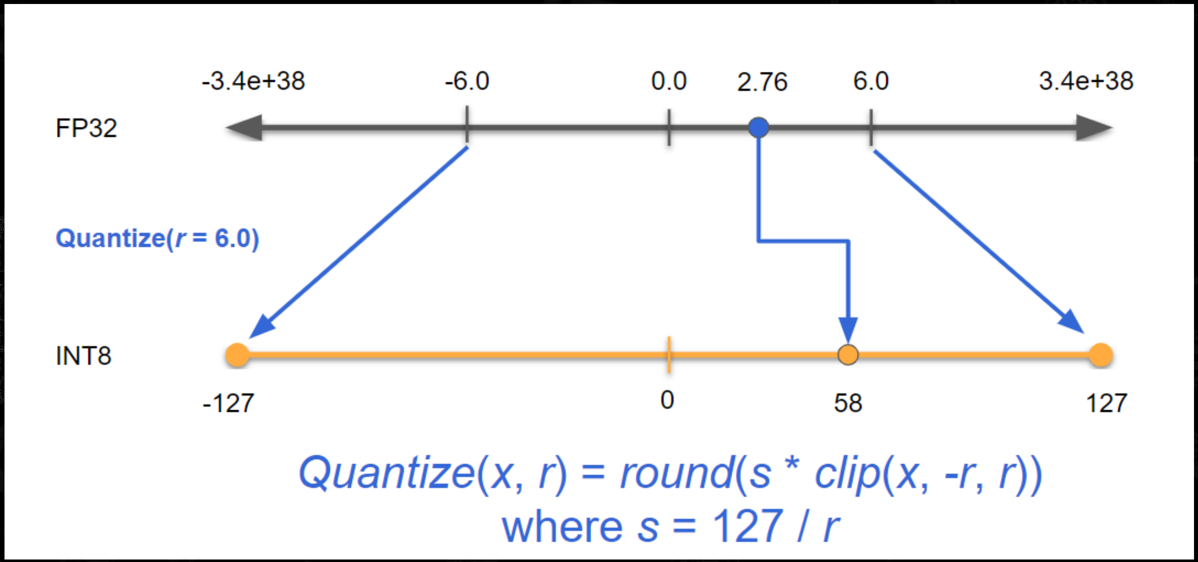


FP16模型的训练方法



Int8模型的推断过程





2 TensorFlow的FP16模型

实数的16-bit半精度浮点数和8-bit定点数表示

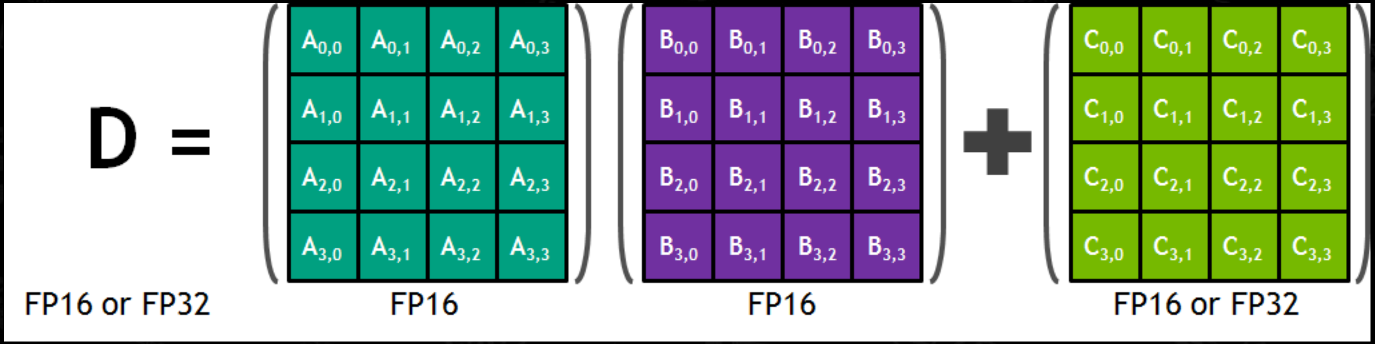
使用低精度的意义

TensorCores适用条件

1. 卷积：K（输入通道），C（输出通道）
2. 通用矩阵乘法（GEMM）：MxK，KxN，（M，N，K）

FP16: 大小为8x
Int8: 大小为16x

如果FP32要使用，可以设置（内部转为FP16）：
TF_ENABLE_CUBLAS_TENSOR_OP_MATH_FP32=1
TF_ENABLE_CUDNN_TENSOR_OP_MATH_FP32=1
TF_ENABLE_CUDNN_RNN_TENSOR_OP_MATH_FP32=1



TensorFlow手动转换模型


```
import tensorflow as tf
import numpy as numpy

input = tf.placeholder(dtype=tf.float32, shape=[None, 128, 128, 16])
input_fp16 = tf.cast(input, dtype=tf.float16)

# Force the layer use tf.float16
conv1 = tf.keras.layers.Conv2D(32, (3,3), 1, "same", dtype=tf.float16)
ret_fp16 = conv1(input_fp16)
ret = tf.cast(ret_fp16, dtype=tf.float32)

s = tf.Session()
s.run(tf.global_variables_initializer())
r1, r2 = s.run([ret, ret_fp16], feed_dict={
input: np.random.randn(4, 128, 128, 16).astype(np.float32)})
# Output: float32 float16
print(r1.dtype, r2.dtype)
```

TensorFlow优化器

1. 使用 tf.train.experimental.MixedPrecisionLossScaleOptimizer

2. 损失函数缩放：FixedLossScale和DynamicLossScale

```
# Define optimizer
opt = tf.train.AdamOptimizer()

# Define static loss scale
loss_scale_value = 1024
loss_scale = tf.train.experimental.FixedLossScale(loss_scale_value)
opt = tf.train.experimental.MixedPrecisionLossScaleOptimizer(opt, loss_scale)

# Define dynamic loss scale
loss_scale = tf.train.experimental.DynamicLossScale(
    initial_loss_scale = loss_scale_value,
    increment_period = 2000,
    multiplier = 2.0,
)
opt = tf.train.experimental.MixedPrecisionLossScaleOptimizer(opt, loss_scale)
```

PYTHON 30th

TensorFlow自动混合精度（Automatic Mixed Precision，AMP）

1.设置环境变量 TF_ENABLE_AUTO_MIXED_PRECISION=“1”

2.在代码中手动开启（1和2选项互相冲突，同时设置会报错）

```
# Define optimizer
opt = tf.train.AdamOptimizer()

# Modify optimizer in graph, copy between fp32 weight and fp16 weight
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)

train_op = opt.miminize(loss)
```

AMP的白名单/灰名单/黑名单

1. 黑名单Op（不使用FP16）：Exp, Log, Softmax…
2. 灰名单Op（不在黑名单Op之后使用FP16）：AvgPool，Sqrt…
3. 白名单Op（可以安全使用FP16）：MaxPool，ReLu…

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/grappler/optimizers/auto_mixed_precision_lists.h

FP16训练模型精度

Table 1: ILSVRC12 classification top-1 accuracy.

Model	Baseline	Mixed Precision	Reference
AlexNet	56.77%	56.93%	(Krizhevsky et al., 2012)
VGG-D	65.40%	65.43%	(Simonyan and Zisserman, 2014)
GoogLeNet (Inception v1)	68.33%	68.43%	(Szegedy et al., 2015)
Inception v2	70.03%	70.02%	(Ioffe and Szegedy, 2015)
Inception v3	73.85%	74.13%	(Szegedy et al., 2016)
Resnet50	75.92%	76.04%	(He et al., 2016b)

Model	Baseline	MP without loss-scale	MP with loss-scale
Faster R-CNN	69.1%	68.6%	69.7%
Multibox SSD	76.9%	diverges	77.1%

https://arxiv.org/pdf/1710.03740.pdf

3 TensorRT的FP16/Int8模型

TensorFlow中使用TensorRT

在TensorRT中使用FP16/Int8

TensorFlow + TensorRT环境的构建

TensorRT的安装（https://docs.nvidia.com/deeplearning/sdk/tensorrt-install-guide/index.html）：

1. TensorRT 安装包：https://developer.nvidia.com/tensorrt

2. 从.deb文件安装libnvinfer.so
   同时安装Python wheel文件tensorrt-6.0.1.5-cp37-none-linux_x86_64.whl

3. 安装TensorFlow 1.14 （GPU版本）

或者直接使用 Docker镜像：

docker pull nvcr.io/nvidia/tensorflow:19.07-py3

TensorFlow中使用TensorRT

PyConChina 2019

1. SavedModel使用TensorRT

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

converter = trt.TrtGraphConverter(
    input_saved_model_dir=input_saved_model_dir)
converter.convert()
converter.save(output_saved_model_dir)

with tf.Session() as sess:
    # First load the SavedModel into the session
    tf.saved_model.loader.load(
        sess, [tf.saved_model.tag_constants.SERVING],
        output_saved_model_dir)
    output = sess.run([output_tensor],
        feed_dict={input_tensor: input_data})
```

PYTHON 30th

TensorFlow模型使用TensorRT

2. Frozen Graph使用TensorRT

```
import tensorflow as tf
from tensorflow.python.compiler.tensorrt import trt_convert as trt
with tf.Session() as sess:
# First deserialize your frozen graph:
with tf.gfile.GFile(“/path/to/your/frozen/graph.pb”, ‘rb’) as f:
    frozen_graph = tf.GraphDef()
    frozen_graph.ParseFromString(f.read())
    # Now you can create a TensorRT inference graph from your
    # frozen graph:
    converter = trt.TrtGraphConverter(
        input_graph_def=frozen_graph,
        nodes_blacklist=['logits', 'classes']) #output nodes
    trt_graph = converter.convert()
    # Import the TensorRT graph into a new graph and run:
    output_node = tf.import_graph_def(
        trt_graph,
        return_elements=['logits', 'classes‘])
    sess.run(output_node)
```

PYTHON 30th

PyConChina 2019

TensorFlow模型使用TensorRT

3. MetaGraph和Checkpoint使用TensorRT

```
with tf.Session() as sess:
    # First create a `Saver` object (for saving and rebuilding a
    # model) and import your `MetaGraphDef` protocol buffer into it:
    saver = tf.train.import_meta_graph(“/path/to/your/model.ckpt.meta”)
    # Then restore your training data from checkpoint files:
    saver.restore(sess, “/path/to/your/model.ckpt”)
    # Finally, freeze the graph:
    frozen_graph = tf.graph_util.convert_variables_to_constants(
        sess,
        tf.get_default_graph().as_graph_def(),
        output_node_names=['logits', 'classes'])
```

PYTHON 30th

TensorFlow导出低精度模型


```
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)
```

TensorRT下Int8模型的校正


```
dataset = tf.data.TFRecordDataset(data_files)
iterator = dataset.make_one_shot_iterator()
features = iterator.get_next()
def input_map_fn():
    return {'input:0': features}

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)
```

TensorRT导出TensorRT plan


```
for n in trt_graph.node:
    if n.op == "TRTEngineOp":
        print("Node: %s, %s" % (n.op, n.name.replace("/", "_")))
        with tf.gfile.GFile("%s.plan" \
            % (n.name.replace("/", "_")), 'wb') as f:
            f.write(n.attr["serialized_segment"].s)
    else:
        print("Exclude Node: %s, %s" \
            % (n.op, n.name.replace("/", "_")))
```

THANK YOU

PYTHON 30th

https://github.com/zxjzxj9

zxjzxj9@gmail.com