


GPU监控

nvidia-smi

nvidia-smi是 Nvidia 显卡命令行管理套件，基于 NVML 库，旨在管理和监控 Nvidia GPU 设备。

```
1 | nvidia-smi
```

 ~ `nvidia-smi` 16:45:24

Fri Nov 10 16:45:26 2017

NVIDIA-SMI 384.90				Driver Version: 384.90			
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
0	GeForce GTX 108...	Off	00000000:02:00.0	Off		N/A	
55%	69C	P2	106W / 280W	8427MiB / 11172MiB	0%	Default	
1	TITAN X (Pascal)	Off	00000000:81:00.0	Off		N/A	
91%	89C	P2	154W / 250W	7061MiB / 12189MiB	38%	Default	

这是 nvidia-smi 命令的输出，其中最重要的两个指标：

- 显存占用
- GPU 利用率

显存占用和 GPU 利用率是两个不一样的东西，显卡是由 GPU 计算单元和显存等组成的，显存和 GPU 的关系有点类似于内存和 CPU 的关系。

显示的表格中：

Fan：风扇转速（0%--100%），N/A表示没有风扇

Temp：GPU温度（GPU温度过高会导致GPU频率下降）

Perf：性能状态，从P0（最大性能）到P12（最小性能）

Pwr：GPU功耗

Persistence-M：持续模式的状态（持续模式耗能大，但在新的GPU应用启动时花费时间更少）

Bus-Id：GPU总线，`domain:bus:device.function`

Disp.A：Display Active，表示GPU的显示是否初始化

Memory-Usage：显存使用率

Volatile GPU-Util：GPU使用率

ECC：是否开启错误检查和纠正技术，0/DISABLED, 1/ENABLED

Compute M.：计算模式，0/DEFAULT,1/EXCLUSIVE_PROCESS,2/PROHIBITED

附加选项：

`nvidia-smi -i xxx`

指定某个GPU

`nvidia-smi -l xxx`

动态刷新信息（默认5s刷新一次），按Ctrl+C停止，可指定刷新频率，以秒为单位

`nvidia-smi -f xxx`

将查询的信息输出到具体的文件中，不在终端显示

gpustat

gpustat 基于nvidia-smi，可以提供更美观简洁的展示，结合 watch 命令，可以动态实时监控GPU 的使用情况。

```
1 pip3 install gpustat
2 watch --color -n1 gpustat -cpu
```

Every 1.0s: gpustat -cpu

```
gpu2  Sun Nov 12 18:31:55 2017
[0] GeForce GTX 1080 Ti | 57°C | 0 % | 2687 / 11172 MB | x:python/4817(2675M)
[1] TITAN X (Pascal) | 89°C | 97 % | 11705 / 12189 MB | j:ipython/35256(3835M)
```

温度 利用率 占用显存 / 总显存 user:process/pid(memory)

显存可以看成是空间，类似于内存。

- 显存用于存放模型，数据
- 显存越大，所能运行的网络也就越大

GPU 计算单元类似于 CPU 中的核，用来进行数值计算。衡量计算量的单位是 flop：the number of floating-point multiplication-adds，浮点数先乘后加算一个 flop。计算能力越强大，速度越快。衡量计算能力的单位是 flops：每秒能执行的 flop 数量

```
1 1*2+3 # 1个flop
2 12+34+4*5 # 3个flop
```

在深度学习中会用到各种各样的数值类型，数值类型命名规范一般为TypeNum，比如 Int64、Float32、Double64。

- Type：有 Int，Float，Double 等
- Num：一般是 8，16，32，64，128，表示该类型所占据的比特数目

常用的数值类型如下图所示：

类型	大小	备注
int8	1个字节	又名Byte
Int16	2个字节	又名Short
Int32	4个字节	又名Int
int64	8个字节	又名Long
Float32	4个字节	单精度浮点数, 又名float
float16	2个字节	半精度浮点数

其中 Float32 是在深度学习中最常用的数值类型，称为单精度浮点数，每一个单精度浮点数占用 4Byte 的显存。

举例来说：有一个 1000x1000 的 矩阵，float32，那么占用的显存差不多就是

1000x1000x4 Byte = 4MB

32x3x256x256 的四维数组（BxCxHxW）占用显存为：24M

GPU加速

使用方法：tf.device('/cpu:0')或tf.device('/gpu:0')。

```
1 import tensorflow as tf
2
3 with tf.device('/cpu:0'):
4     a = tf.constant([1.,2.,3.],shape=[3],name='a')
5     b = tf.constant([2.,3.,4.],shape=[3],name='b')
6 with tf.device('/gpu:0'):
7     c = a + b
8
9 sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
10 sess.run(c)
```

运行结果：

```
1 2019-06-13 17:37:10.031128: I tensorflow/compiler/xla/service/service.cc:150] XLA
  service 0x5c21980 executing computations on platform CUDA. Devices:
2 2019-06-13 17:37:10.031202: I tensorflow/compiler/xla/service/service.cc:158]
  StreamExecutor device (0): Tesla T4, Compute Capability 7.5
3 2019-06-13 17:37:10.031226: I tensorflow/compiler/xla/service/service.cc:158]
  StreamExecutor device (1): Tesla T4, Compute Capability 7.5
4 2019-06-13 17:37:10.036025: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94]
  CPU Frequency: 2200205000 Hz
5 2019-06-13 17:37:10.044162: I tensorflow/compiler/xla/service/service.cc:150] XLA
  service 0x5d5fdb0 executing computations on platform Host. Devices:
```

```
6 2019-06-13 17:37:10.044208: I tensorflow/compiler/xla/service/service.cc:158]
StreamExecutor device (0): <undefined>, <undefined>
7 2019-06-13 17:37:10.044904: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1433]
Found device 0 with properties:
8 name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
9 pciBusID: 0000:05:00.0
10 totalMemory: 14.73GiB freeMemory: 14.60GiB
11 2019-06-13 17:37:10.045397: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1433]
Found device 1 with properties:
12 name: Tesla T4 major: 7 minor: 5 memoryClockRate(GHz): 1.59
13 pciBusID: 0000:84:00.0
14 totalMemory: 14.73GiB freeMemory: 14.60GiB
15 2019-06-13 17:37:10.045563: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512]
Adding visible gpu devices: 0, 1
16 2019-06-13 17:37:10.048288: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984]
Device interconnect StreamExecutor with strength 1 edge matrix:
17 2019-06-13 17:37:10.048320: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990]
0 1
18 2019-06-13 17:37:10.048334: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1003]
0: N N
19 2019-06-13 17:37:10.048346: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1003]
1: N N
20 2019-06-13 17:37:10.049316: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 14202 MB
memory) -> physical GPU (device: 0, name: Tesla T4, pci bus id: 0000:05:00.0, compute
capability: 7.5)
21 2019-06-13 17:37:10.049881: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:1 with 14202 MB
memory) -> physical GPU (device: 1, name: Tesla T4, pci bus id: 0000:84:00.0, compute
capability: 7.5)
22 Device mapping:
23 /job:localhost/replica:0/task:0/device:XLA_GPU:0 -> device: XLA_GPU device
24 /job:localhost/replica:0/task:0/device:XLA_GPU:1 -> device: XLA_GPU device
25 /job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
26 /job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pci bus id:
0000:05:00.0, compute capability: 7.5
27 /job:localhost/replica:0/task:0/device:GPU:1 -> device: 1, name: Tesla T4, pci bus id:
0000:84:00.0, compute capability: 7.5
28 2019-06-13 17:37:10.058346: I tensorflow/core/common_runtime/direct_session.cc:317]
Device mapping:
29 /job:localhost/replica:0/task:0/device:XLA_GPU:0 -> device: XLA_GPU device
30 /job:localhost/replica:0/task:0/device:XLA_GPU:1 -> device: XLA_GPU device
31 /job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
32 /job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pci bus id:
0000:05:00.0, compute capability: 7.5
33 /job:localhost/replica:0/task:0/device:GPU:1 -> device: 1, name: Tesla T4, pci bus id:
0000:84:00.0, compute capability: 7.5
34
35 add: (Add): /job:localhost/replica:0/task:0/device:GPU:0
36 2019-06-13 17:37:10.061066: I tensorflow/core/common_runtime/placer.cc:1059] add:
(Add)/job:localhost/replica:0/task:0/device:GPU:0
37 a: (Const): /job:localhost/replica:0/task:0/device:CPU:0
```

```
38 2019-06-13 17:37:10.061099: I tensorflow/core/common_runtime/placer.cc:1059] a:
   (Const)/job:localhost/replica:0/task:0/device:CPU:0
39 b: (Const): /job:localhost/replica:0/task:0/device:CPU:0
40 2019-06-13 17:37:10.061118: I tensorflow/core/common_runtime/placer.cc:1059] b:
   (Const)/job:localhost/replica:0/task:0/device:CPU:0
```

该日志中信息如下：

- 发现服务器中有两个GPU可用，打印GPU信息；
- 将所有可见GPU都加入运算；
- 创建TensorFlow逻辑GPU和真实物理GPU的映射关系；
- 显示TensorFlow中所有的逻辑GPU和CPU；
- 开始运算：a和b的赋值操作在CPU上完成，add操作在GPU上完成。

在默认情况下，即使机器有多个cpu，Tensorflow也不会去区分它们，统一使用/cpu:0。

而同一台机器上不同GPU的名称是不同的，如/gpu:0，/gpu:1等。

默认情况下，Tensorflow优先使用GPU。

需要注意的是，在Tensorflow上，不是所有的操作都可以放在GPU上的，如：

```
1 import tensorflow as tf
2
3 a_cpu = tf.Variable(0,name='a_cpu')
4 with tf.device('/gpu:0'):
5     a_gpu = tf.Variable(0,name='a_gpu')
6
7 with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
8     sess.run(tf.initialize_all_variables())
```

则会报错如下：

```
1 InvalidArgumentError (see above for traceback): Cannot assign a device for operation
   a_gpu: Could not satisfy explicit device specification '/device:GPU:0' because no
   supported kernel for GPU devices is available.
2 Colocation Debug Info:
3 Colocation group had the following types and devices:
4 Assign: CPU
5 Identity: GPU CPU XLA_CPU XLA_GPU
6 VariableV2: CPU
7
8 Colocation members and user-requested devices:
9   a_gpu (VariableV2) /device:GPU:0
10  a_gpu/Assign (Assign) /device:GPU:0
11  a_gpu/read (Identity) /device:GPU:0
12
13 Registered kernels:
14   device='GPU'; dtype in [DT_INT64]
15   device='GPU'; dtype in [DT_DOUBLE]
16   device='GPU'; dtype in [DT_FLOAT]
17   device='GPU'; dtype in [DT_HALF]
```

```
18 | device='CPU'
19 |
20 | [[node a_gpu (defined at demo.py:5) ]]
```

为了避免这个问题，可以在生成Session时指定allow_soft_placement=True，当运算无法在GPU上执行时，会自动将运算放到CPU上。用法：

```
se 1 |
tf _ | sion(config=tf.ConfigProto(log_device_placement=True,allow_soft_placement=True))
```

Tensorflow会默认占用设备所有GPU以及每个GPU上的显存，如果只使用部分GPU可以：

(注：虽然占用所有GPU，但是会优先使用/GPU:0)

```
1 | #命令行用法
2 | CUDA_VISIBLE_DEVICES=0,1 python demo.py
```

或者

```
1 | #在代码中使用
2 | import os
3 | os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
```

TensorFlow默认一次性占用GPU的所有显存，但是也支持动态分配GPU的显存，使得不会一开始就占满所有显存。

```
1 | config = tf.ConfigProto()
2 | config.gpu_options.allow_growth = True
3 | #也可以直接按固定的比例分配
4 | #config.gpu_options.per_process_gpu_memory_fraction = 0.4
5 | sess = tf.Session(config=config)
```

总结几个参数

- log_device_placement：将运行每一个操作的设备输出到屏幕上。
- allow_soft_placement：将GPU上不能运行的运算自动放到CPU上运行。
- allow_growth：动态分配GPU显存。
- per_process_gpu_memory_fraction：按比例分配GPU显存。

使用多GPU

也可以称为多塔式方式，其中每个塔都会分配给不同 GPU。

指定GPU的方式

如果你想让 TensorFlow 在多个 GPU 上运行, 你可以建立 multi-tower 结构, 在这个结构 里每个 tower 分别被指配给不同的 GPU 运行. 比如:

```
1 | import tensorflow as tf
```

```

2 # 新建一个 graph.
3 c = []
4 for d in ['/gpu:0', '/gpu:1']:
5     with tf.device(d):
6         a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
7         b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
8         c.append(tf.matmul(a, b))
9 with tf.device('/cpu:0'):
10     sum = tf.add_n(c)
11 # 新建session with log_device_placement并设置为True.
12 sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
13 # 运行这个op.
14 print(sess.run(sum))

```

在运行结果的日志中，找到关于各个操作的说明：

```

1 MatMul: (MatMul): /job:localhost/replica:0/task:0/device:GPU:0
2 2019-06-14 09:50:44.365093: I tensorflow/core/common_runtime/placer.cc:1059] MatMul:
  (MatMul)/job:localhost/replica:0/task:0/device:GPU:0
3 MatMul_1: (MatMul): /job:localhost/replica:0/task:0/device:GPU:1
4 2019-06-14 09:50:44.365131: I tensorflow/core/common_runtime/placer.cc:1059] MatMul_1:
  (MatMul)/job:localhost/replica:0/task:0/device:GPU:1
5 AddN: (AddN): /job:localhost/replica:0/task:0/device:CPU:0
6 2019-06-14 09:50:44.365158: I tensorflow/core/common_runtime/placer.cc:1059] AddN:
  (AddN)/job:localhost/replica:0/task:0/device:CPU:0
7 Const: (Const): /job:localhost/replica:0/task:0/device:GPU:0
8 2019-06-14 09:50:44.365178: I tensorflow/core/common_runtime/placer.cc:1059] Const:
  (Const)/job:localhost/replica:0/task:0/device:GPU:0
9 Const_1: (Const): /job:localhost/replica:0/task:0/device:GPU:0
10 2019-06-14 09:50:44.365196: I tensorflow/core/common_runtime/placer.cc:1059] Const_1:
  (Const)/job:localhost/replica:0/task:0/device:GPU:0
11 Const_2: (Const): /job:localhost/replica:0/task:0/device:GPU:1
12 2019-06-14 09:50:44.365214: I tensorflow/core/common_runtime/placer.cc:1059] Const_2:
  (Const)/job:localhost/replica:0/task:0/device:GPU:1
13 Const_3: (Const): /job:localhost/replica:0/task:0/device:GPU:1
14 2019-06-14 09:50:44.365231: I tensorflow/core/common_runtime/placer.cc:1059] Const_3:
  (Const)/job:localhost/replica:0/task:0/device:GPU:1
15 [[ 44.  56.]
16  [ 98. 128.]]

```

日志和代码效果一致，a、b和c操作都在GPU上运行，sum操作在CPU上运行。

非指定GPU的方式

大多数时候，我们只能确定要用几个GPU，而不想一直在代码中修改GPU的名称，因此下面给出例子，讲解如何仅指定GPU数量来运行多GPU算法。


```

1 import tensorflow as tf
2 c = []
3 for i in range(num_gpus):
4     with tf.device("/gpu:%d" % i):
5         with tf.name_scope("tower_%d" % i):
6             a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
7             b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
8             c.append(tf.matmul(a, b))
9 with tf.device("cpu:0"):
10     sum = tf.add_n(c)
11 sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
12 print(sess.run(sum))

```

输出结果为：

```

1 tower_0/MatMul: (MatMul): /job:localhost/replica:0/task:0/device:GPU:0
2 2019-06-14 10:53:24.509237: I tensorflow/core/common_runtime/placer.cc:1059]
tower_0/MatMul: (MatMul)/job:localhost/replica:0/task:0/device:GPU:0
3 tower_1/MatMul: (MatMul): /job:localhost/replica:0/task:0/device:GPU:1
4 2019-06-14 10:53:24.509265: I tensorflow/core/common_runtime/placer.cc:1059]
tower_1/MatMul: (MatMul)/job:localhost/replica:0/task:0/device:GPU:1
5 AddN: (AddN): /job:localhost/replica:0/task:0/device:CPU:0
6 2019-06-14 10:53:24.509286: I tensorflow/core/common_runtime/placer.cc:1059] AddN:
(AddN)/job:localhost/replica:0/task:0/device:CPU:0
7 tower_0/Const: (Const): /job:localhost/replica:0/task:0/device:GPU:0
8 2019-06-14 10:53:24.509305: I tensorflow/core/common_runtime/placer.cc:1059]
tower_0/Const: (Const)/job:localhost/replica:0/task:0/device:GPU:0
9 tower_0/Const_1: (Const): /job:localhost/replica:0/task:0/device:GPU:0
10 2019-06-14 10:53:24.509323: I tensorflow/core/common_runtime/placer.cc:1059]
tower_0/Const_1: (Const)/job:localhost/replica:0/task:0/device:GPU:0
11 tower_1/Const: (Const): /job:localhost/replica:0/task:0/device:GPU:1
12 2019-06-14 10:53:24.509341: I tensorflow/core/common_runtime/placer.cc:1059]
tower_1/Const: (Const)/job:localhost/replica:0/task:0/device:GPU:1
13 tower_1/Const_1: (Const): /job:localhost/replica:0/task:0/device:GPU:1
14 2019-06-14 10:53:24.509358: I tensorflow/core/common_runtime/placer.cc:1059]
tower_1/Const_1: (Const)/job:localhost/replica:0/task:0/device:GPU:1
15 [[ 44.  56.]
16  [ 98. 128.]]

```

在日志中我们能够看到操作都是以tower作用域划分，在不同作用域下的变量名称不同。

作用域

tf.Variable与tf.get_variable()

使用tf.Variable()时，如果检测到命名冲突，系统会自己处理。使用tf.get_variable()时，系统不会处理冲突，而会报错

tf.Variable


```

1 import tensorflow as tf
2 w_1 = tf.Variable(3,name="w_1")
3 w_2 = tf.Variable(1,name="w_1")
4 print w_1.name
5 print w_2.name
6 #输出
7 #w_1:0
8 #w_1_1:0

```

tf.get_variable()

```

1 import tensorflow as tf
2
3 w_1 = tf.get_variable(name="w_1",initializer=1)
4 w_2 = tf.get_variable(name="w_1",initializer=2)
5 #错误信息
6 #ValueError: Variable w_1 already exists, disallowed. Did
7 #you mean to set reuse=True in VarScope?

```

基于这两个函数的特性，当我们需要共享变量的时候，需要使用tf.get_variable()。在其他情况下，这两个的用法是一样的

```

1 import tensorflow as tf
2
3 with tf.variable_scope("scope1"):
4     w1 = tf.get_variable("w1", shape=[])
5     w2 = tf.Variable(0.0, name="w2")
6 with tf.variable_scope("scope1", reuse=True):
7     w1_p = tf.get_variable("w1", shape=[])
8     w2_p = tf.Variable(1.0, name="w2")
9
10 print(w1 is w1_p, w2 is w2_p)
11 #输出
12 #True False

```

总结：由于tf.Variable() 每次都在创建新对象，所有reuse=True 和它并没有什么关系。对于get_variable()，来说，如果已经创建的变量对象，就把那个对象返回，如果没有创建变量对象的话，就创建一个新的。

tf.name_scope()和tf.variable_scope()

TF中有两种作用域类型：命名域 (name scope)，通过tf.name_scope 或 tf.op_scope创建；变量域 (variable scope)，通过tf.variable_scope 或 tf.variable_op_scope创建；这两种作用域，对于使用tf.Variable()方式创建的变量，具有相同的效果，都会在变量名称前面，加上域名称。对于通过tf.get_variable()方式创建的变量，只有variable scope名称会加到变量名称前面，而name scope不会作为前缀。

1. tf.variable_scope有一个参数reuse默认None，当reuse为None或False时，表示在该作用域下tf.get_variable函数只能创建变量，如果创建两个相同名字的变量就会报错
2. 将reuse参数设置为True时，作用域可以共享变量，两个变量相等

```

1 with tf.variable_scope("foo", reuse=True):
2
3     v1 = tf.get_variable("v", [1])
4
5     v2 = tf.get_variable("v", [1])
6
7 print(v1.name, v2.name, v1 == v2)
8
9 #foo/v:0 foo/v:0 True

```

总结：

1. name_scope不会作为tf.get_variable变量的前缀，但是会作为tf.Variable的前缀。
2. 在variable_scope的作用域下，tf.get_variable()和tf.Variable()都加了scope_name前缀。因此，在tf.variable_scope的作用域下，通过get_variable()可以使用已经创建的变量，实现了变量的共享，即可以通过get_variable()在tf.variable_scope设定的作用域范围内进行变量共享。
3. 在重复使用的时候，一定要在代码中强调 scope.reuse_variables()

实战

随机森林

在各大材料中提到随机森林使用GPU加速不明显，这个我们具体看看其原理。

现在有个数据集data_per200.csv，其中共298列，前297列都是特征列，最后一列是类别列，共11个类别。现在将数据分为训练集和测试集，用训练集训练的模型来预测测试集的类别。代码如下：

```

1 import time
2
3 import pandas
4 import tensorflow as tf
5 from tensorflow.contrib.tensor_forest.python import tensor_forest
6 from tensorflow.python.ops import resources
7 from sklearn.model_selection import train_test_split
8
9 # 1. 导入数据
10 start = time.time()
11 data = pandas.read_csv('./data_per200.csv')
12 input_x = data.iloc[:, 0:-1]
13 input_y = data.iloc[:, -1]
14 # 2. 数据切分为训练集和测试集
15 X_train, X_test, y_train_label, y_test_label = train_test_split(input_x, input_y,
16     test_size=0.25, random_state=0)
17 class_name = list(set(y_train_label))
18 class_nums = len(class_name)
19 class_dict = {}
20 for idx, name in enumerate(class_name):
21     class_dict[name] = idx
22 y_train = y_train_label.map(class_dict)
23 y_test = y_test_label.map(class_dict)
24 print("数据准备完毕")

```

```

24 # 3. 准备算法参数
25 num_steps = 300
26 num_classes = len(set(input_y))
27 num_features = len(data.columns) - 1
28 num_trees = 500
29 max_nodes = 1000
30 # 4. 在指定的GPU上设置数据流图
31 with tf.device("/gpu:0"):
32     # Input and Target placeholders
33     x = tf.placeholder(tf.float32, shape=[None, num_features])
34     y = tf.placeholder(tf.int64, shape=[None])
35     # Random Forest Parameters
36     hparams = tensor_forest.ForestHParams(num_classes=num_classes,
37                                           num_features=num_features, num_trees=num_trees,
38                                           max_nodes=max_nodes).fill()
39     # Build the Random Forest
40     forest_graph = tensor_forest.RandomForestGraphs(hparams)
41     # Get training graph and loss
42     train_op = forest_graph.training_graph(x, y)
43     loss_op = forest_graph.training_loss(x, y)
44     # Measure the accuracy
45     infer_op, _, _ = forest_graph.inference_graph(x)
46     correct_prediction = tf.equal(tf.argmax(infer_op, 1), tf.cast(y, tf.int64))
47     accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
48     # 初始化参数
49     init_vars = tf.group(tf.global_variables_initializer(),
50                           resources.initialize_resources(resources.shared_resources()))
51 # 5. 创建数据流图的运行时会话，并配置打印详细的日志信息
52 sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
53 sess.run(init_vars)
54 # 6. 开始训练
55 for i in range(1, num_steps + 1):
56     _, loss = sess.run([train_op, loss_op], feed_dict={x: x_train, y: y_train})
57     if i % 50 == 0 or i == 1:
58         acc = sess.run(accuracy_op, feed_dict={x: x_train, y: y_train})
59         print('Step %i, Loss: %f, Acc: %f' % (i, loss, acc))
60 # 7. 模型测试
61 print("Test Accuracy:", sess.run(accuracy_op, feed_dict={x: x_test, y: y_test}))
62 end = time.time()
63 print("time: ", end - start)
64

```

执行命令为：

```
1 | python3 tf_randomforest.py > tf_randomforest.log
```

使用gpustat的监控命令查看，有一瞬间GPU使用率飙升到4%。现在查看tf_randomforest.log日志。其中有如下关键信息：

发现有两个GPU

```

1 Device mapping:
2 /job:localhost/replica:0/task:0/device:XLA_GPU:0 -> device: XLA_GPU device
3 /job:localhost/replica:0/task:0/device:XLA_GPU:1 -> device: XLA_GPU device
4 /job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
5 /job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pci bus id:
  0000:05:00.0, compute capability: 7.5
6 /job:localhost/replica:0/task:0/device:GPU:1 -> device: 1, name: Tesla T4, pci bus id:
  0000:84:00.0, compute capability: 7.5

```

随机森林的核心功能在GPU上进行

```

1 device_dummy_0/Initializer/random_uniform/RandomUniform: (RandomUniform):
  /job:localhost/replica:0/task:0/device:CPU:0
2 device_dummy_0/Initializer/random_uniform/sub: (Sub):
  /job:localhost/replica:0/task:0/device:CPU:0
3 device_dummy_0/Initializer/random_uniform/mul: (Mul):
  /job:localhost/replica:0/task:0/device:CPU:0
4 device_dummy_0/Initializer/random_uniform: (Add):
  /job:localhost/replica:0/task:0/device:CPU:0
5 .....

```

存在少量的GPU计算

```

1 ToFloat: (Cast): /job:localhost/replica:0/task:0/device:GPU:0
2
3 stack: (Pack): /job:localhost/replica:0/task:0/device:GPU:0
4 ToFloat_1: (Cast): /job:localhost/replica:0/task:0/device:GPU:0
5 Mean: (Mean): /job:localhost/replica:0/task:0/device:GPU:0
6 training_loss: (Neg): /job:localhost/replica:0/task:0/device:GPU:0
7
8 ArgMax: (ArgMax): /job:localhost/replica:0/task:0/device:GPU:0
9 Equal: (Equal): /job:localhost/replica:0/task:0/device:GPU:0
10 Cast: (Cast): /job:localhost/replica:0/task:0/device:GPU:0
11 Mean_1: (Mean): /job:localhost/replica:0/task:0/device:GPU:0
12
13 Placeholder: (Placeholder): /job:localhost/replica:0/task:0/device:GPU:0
14 Placeholder_1: (Placeholder): /job:localhost/replica:0/task:0/device:GPU:0
15
16 Const: (Const): /job:localhost/replica:0/task:0/device:GPU:0
17
18 ArgMax/dimension: (Const): /job:localhost/replica:0/task:0/device:GPU:0
19 Const_1: (Const): /job:localhost/replica:0/task:0/device:GPU:0

```

其中空行表示中间有大量的操作是CPU完成的。从行数统计来看，大概有1.9w行都是CPU的操作，而GPU操作仅14行。从GPU的操作来看，也是和随机森林训练无关的操作。因此才会说随机森林使用GPU加速不明显。

如果将两份相同的代码都在后台执行，命令如下：

```

1 python3 tf_randomforest.py > tf_randomforest1.log &
2 python3 tf_randomforest.py > tf_randomforest2.log &

```

那么会发现在日志文件中显示两个都能正常完成。但是在控制台日志中却出现了OOM的异常。

```
1 2019-06-14 11:54:36.616992: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 13.76G (14773796864 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
2 2019-06-14 11:54:36.618294: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 12.38G (13296416768 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
3 2019-06-14 11:54:36.621591: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 11.14G (11966774272 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
4 2019-06-14 11:54:36.624942: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 10.03G (10770096128 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
5 2019-06-14 11:54:36.628375: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 9.03G (9693086720 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
6 2019-06-14 11:54:36.639852: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 8.12G (8723777536 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
7 2019-06-14 11:54:36.642044: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 7.31G (7851399680 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
8 2019-06-14 11:54:36.644030: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 6.58G (7066259456 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
9 2019-06-14 11:54:36.646217: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 5.92G (6359633408 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
10 2019-06-14 11:54:36.647268: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 5.33G (5723670016 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
11 2019-06-14 11:54:36.648343: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 4.80G (5151302656 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
12 2019-06-14 11:54:36.649380: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 4.32G (4636172288 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
13 2019-06-14 11:54:36.650417: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 3.89G (4172555008 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
14 2019-06-14 11:54:36.651460: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 3.50G (3755299328 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
15 2019-06-14 11:54:36.652493: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 3.15G (3379769344 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
16 2019-06-14 11:54:36.653525: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 2.83G (3041792256 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
17 2019-06-14 11:54:36.654559: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 2.55G (2737613056 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
```

```
18 2019-06-14 11:54:36.655600: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 2.29G (2463851776 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
19 2019-06-14 11:54:36.656648: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 2.06G (2217466624 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
20 2019-06-14 11:54:36.657680: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.86G (1995719936 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
21 2019-06-14 11:54:36.658711: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.67G (1796147968 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
22 2019-06-14 11:54:36.659752: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.50G (1616533248 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
23 2019-06-14 11:54:36.660790: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.35G (1454880000 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
24 2019-06-14 11:54:36.661836: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.22G (1309392128 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
25 2019-06-14 11:54:36.662891: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1.10G (1178452992 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out
of memory
26 2019-06-14 11:54:36.663929: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 1011.47M (1060607744 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
27 2019-06-14 11:54:36.664989: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 910.33M (954546944 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
28 2019-06-14 11:54:36.666038: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 819.29M (859092224 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
29 2019-06-14 11:54:36.667086: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 737.36M (773182976 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
30 2019-06-14 11:54:36.668152: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 663.63M (695864832 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
31 2019-06-14 11:54:36.669192: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 597.27M (626278400 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
32 2019-06-14 11:54:36.670226: E tensorflow/stream_executor/cuda/cuda_driver.cc:806]
failed to allocate 537.54M (563650560 bytes) from device: CUDA_ERROR_OUT_OF_MEMORY:
out of memory
```

发现申请的空间逐渐缩小。这是因为如果没有设定GPU使用的显存大小，那么TensorFlow任务默认占用一个GPU的全部显存。而实际上该任务却没有使用那么多的显存，因此第二个任务也能够完成，这里的解决方法是限定每个任务对GPU的显存使用量。方法如下：

```

1 options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
2 sess = tf.Session(config=tf.ConfigProto(log_device_placement=True,
    gpu_options=gpu_options))

```

结果如下：

```

Every 1.0s: gpustat -cpu
rgibns1 Fri Jun 14 12:36:28 2019
[0] Tesla T4          | 53'C,  1 % | 10500 / 15079 MB | root:python3/35701(5245M) root:python3/35751(5245M)
[1] Tesla T4          | 52'C,  0 % |  248 / 15079 MB | root:python3/35701(119M) root:python3/35751(119M)

```

在该图中可以发现GPU使用量就比较低，而且不会出现OOM异常。另外，设定的显存也会出现不够用的情况，还可以指定动态显存占用量。方法如下：

```

1 config = tf.ConfigProto()
2 config.gpu_options.allow_growth=True
3 sess = tf.Session(config=config)

```

当allow_growth设置为True时，分配器将不会指定所有的GPU内存，而是根据需求增长。可以发现下图中实际使用的GPU比预期的还要少。

```

Every 1.0s: gpustat -cpu
rgibns1 Fri Jun 14 12:43:42 2019
[0] Tesla T4          | 54'C,  4 % |  460 / 15079 MB | root:python3/44607(225M) root:python3/44552(225M)
[1] Tesla T4          | 53'C,  0 % |  248 / 15079 MB | root:python3/44607(119M) root:python3/44552(119M)

```

但是在任务运行要结束的时候，并不会主动释放显存，官方表示不释放显存是为了防止显存出现碎片化导致其他正在运行的任务无法合理利用。

```

Every 1.0s: gpustat -cpu
rgibns1 Fri Jun 14 12:44:19 2019
[0] Tesla T4          | 57'C,  1 % |  460 / 15079 MB | root:python3/44607(225M) root:python3/44552(225M)
[1] Tesla T4          | 56'C,  0 % |  248 / 15079 MB | root:python3/44607(119M) root:python3/44552(119M)

```

当限定GPU时，例如限定使用ID为1的GPU，方法是：

```

1 os.environ["CUDA_VISIBLE_DEVICES"] = "1"

```

代码中指定使用gpu:1，输出的控制台日志为：

```

1 InvalidArgumentError (see above for traceback): Cannot assign a device for operation
  ToFloat: node ToFloat (defined at tf_randomforest.py:51) was explicitly assigned to
  /device:GPU:1 but available devices are [ /job:localhost/replica:0/task:0/device:CPU:0,
  /job:localhost/replica:0/task:0/device:GPU:0,
  /job:localhost/replica:0/task:0/device:XLA_CPU:0,
  /job:localhost/replica:0/task:0/device:XLA_GPU:0 ]. Make sure the device specification
  refers to a valid device.
2 [[node ToFloat (defined at tf_randomforest.py:51) ]]

```

但是在代码中指定使用gpu:0，就会出现以下情况：

输出的控制台日志为：

```
1 2019-06-14 12:50:58.717355: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512]
Adding visible gpu devices: 0
2 2019-06-14 12:50:58.718891: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984]
Device interconnect StreamExecutor with strength 1 edge matrix:
3 2019-06-14 12:50:58.718919: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990]
0
4 2019-06-14 12:50:58.718941: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1003] 0:
N
5 2019-06-14 12:50:58.719444: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 14202 MB
memory) -> physical GPU (device: 0, name: Tesla T4, pci bus id: 0000:84:00.0, compute
capability: 7.5)
```

日志中表示使用了ID为0的GPU，而从GPU的监控来看，却是使用了ID为1的GPU

```
Every 1.0s: gpustat -cpu

rgibns1 Fri Jun 14 12:51:13 2019
[0] Tesla T4      | 55'C,  0 % |   10 / 15079 MB |
[1] Tesla T4      | 56'C,  0 % |  235 / 15079 MB | root:python3/5339(225M)
```

因此这里的gpu会仅使用一个，然后会映射到TensorFlow的逻辑ID=0

这里再进行第二个实验，限定使用ID为0的GPU，方法是：

```
1 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
```

代码中指定使用gpu:0，输出的控制台日志为：

```
totalMemory: 14.73GiB freeMemory: 14.60GiB
2019-06-14 12:59:45.356607: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0
2019-06-14 12:59:45.357810: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] Device interconnect StreamExecu
2019-06-14 12:59:45.357830: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0
2019-06-14 12:59:45.357842: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1003] 0:  N
2019-06-14 12:59:45.358218: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job
.5)
```

```
1172.18.130.34 x +
Every 1.0s: gpustat -cpu

rgibns1 Fri Jun 14 12:59:52 2019
[0] Tesla T4      | 53'C,  0 % |  129 / 15079 MB | root:python3/16027(119M)
[1] Tesla T4      | 54'C,  0 % |   10 / 15079 MB |
```

代码中指定使用gpu:1，输出的控制台日志为：

```
1 | InvalidArgumentError (see above for traceback): Cannot assign a device for operation
   | ToFloat: node ToFloat (defined at tf_randomforest.py:51) was explicitly assigned to
   | /device:GPU:1 but available devices are [ /job:localhost/replica:0/task:0/device:CPU:0,
   | /job:localhost/replica:0/task:0/device:GPU:0,
   | /job:localhost/replica:0/task:0/device:XLA_CPU:0,
   | /job:localhost/replica:0/task:0/device:XLA_GPU:0 ]. Make sure the device specification
   | refers to a valid device.
2 | [[node ToFloat (defined at tf_randomforest.py:51) ]]
```

可以发现同样报错。因此限定GPU语句：

```
1 | os.environ["CUDA_VISIBLE_DEVICES"] = "0"
```

规则如下：

- 指定使用物理的GPU的ID作为候选GPU集合；
- 物理GPU的ID会映射为TensorFlow的逻辑ID，且逻辑ID从0开始。