

记录一下在服务器上配环境的经过

12月6日

首先是装了anaconda, 创建了虚拟环境, 输入nvcc -V并没有反应, 因此怀疑没装cuda, 便准备装cuda和cudnn, 首先是使用官网的wget方法, 发现连不上网站, 经过查询之后便选择在自己电脑下载完之后再通过scp传到服务器上。之后发现没有sudo的权限, 办个后来在usr/local里面发现了cuda, 只不过是没加到环境变量。

之后安装pytorch, 使用pycharm远程运行程序, 发现运行速度很慢, 在经过验证之后确实是在gpu上跑的。运行时间如下图: 基本上在150秒左右

```
/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/test.py
epoch1
-----
train_loss: 0.6941      train_acc: 0.5106
val_loss: 0.6884      val_acc: 0.5150

149.99994564056396
epoch2
-----
train_loss: 0.6769      train_acc: 0.5656
val_loss: 0.6671      val_acc: 0.6175

149.38256549835205
epoch3
-----
train_loss: 0.6563      train_acc: 0.5962
val_loss: 0.6450      val_acc: 0.6300

149.29409956932068
epoch4
-----
```

而我在自己3060笔记本上用gpu跑的运行时间如下: 基本上只要5秒

```
F:\program\anaconda\envs\pytorch\python.exe D:\code\Pycharm\猫狗大战\test.py
epoch1
-----
train_loss: 0.6956      train_acc: 0.4969
val_loss: 0.6899      val_acc: 0.5350

5.191365480422974
epoch2
-----
train_loss: 0.6791      train_acc: 0.5763
val_loss: 0.6689      val_acc: 0.6000

4.775965690612793
epoch3
-----
train_loss: 0.6505      train_acc: 0.6206
val_loss: 0.6390      val_acc: 0.6425

5.325186252593994
epoch4
-----
```

再看gpu利用率为0

NVIDIA-SMI 545.23.08			Driver Version: 545.23.08			CUDA Version: 12.3		
GPU	Name		Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute M.	
							MIG M.	
0	NVIDIA GeForce RTX 3090		On	00000000:B1:00:0	Off		N/A	
37%	51C	P2	111W / 350W	8703MiB / 24576MiB		0%	Default	
							N/A	
Processes:								
GPU	GI	CI	PID	Type	Process name	GPU Memory		
	ID	ID				Usage		
0	N/A	N/A	1609	G	/usr/lib/xorg/Xorg	4MiB		
0	N/A	N/A	2307	G	/usr/lib/xorg/Xorg	4MiB		
0	N/A	N/A	1503697	C	python	376MiB		
0	N/A	N/A	3098038	C	...4/anaconda3/envs/pytorch/bin/python	470MiB		
0	N/A	N/A	3871272	C	python	7810MiB		

于是第一反应是服务器上其实并没有进行cuda加速，联想到之前的nvcc问题，经过网上搜索后发现是由于没有设置cuda的环境变量导致的，于是按照教程设置了，此时可以显示nvcc版本，但是程序运行结果仍然和之前一样。

```
watney1024@HiFEM:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Fri_Nov__3_17:16:49_PDT_2023
Cuda compilation tools, release 12.3, V12.3.103
Build cuda_12.3.r12.3/compiler.33492891_0
```

之后从网上找了一段测试pytorch的代码，

```
import torch

# 打印 CUDA 版本
print('CUDA版本:', torch.version.cuda)

# 打印 PyTorch 版本
print('PyTorch版本:', torch.__version__)

# 检查 CUDA 是否可用
print('显卡是否可用:', '可用' if torch.cuda.is_available() else '不可用')

# 检查显卡数量
print('显卡数量:', torch.cuda.device_count())

# 检查是否支持 BF16 数字格式
print('是否支持BF16数字格式:', '支持' if torch.cuda.is_bf16_supported() else '不支持')
```

```

# 获取当前显卡型号
print('当前显卡型号:', torch.cuda.get_device_name())

# 获取当前显卡的 CUDA 算力
print('当前显卡的CUDA算力:', torch.cuda.get_device_capability())

# 获取当前显卡的总显存
print('当前显卡的总显存:', torch.cuda.get_device_properties(0).total_memory / 1024 /
1024 / 1024, 'GB')

# 检查是否支持 TensorCore
print('是否支持TensorCore:', '支持' if (torch.cuda.get_device_properties(0).major
>= 7) else '不支持')

# 获取当前显卡的显存使用率
print('当前显卡的显存使用率:', torch.cuda.memory_allocated(0) /
torch.cuda.get_device_properties(0).total_memory * 100, '%')

# 检查 cuDNN 是否可用
print('cudnn是否可用:', torch.backends.cudnn.is_available())

```

结果如下：

```

/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/test_cuda.py
CUDA版本： 12.1
PyTorch版本： 2.1.2+cu121
显卡是否可用： 可用
显卡数量： 1
是否支持BF16数字格式： 支持
当前显卡型号： NVIDIA GeForce RTX 3090
当前显卡的CUDA算力： (8, 6)
当前显卡的总显存： 23.48492431640625 GB
是否支持TensorCore： 支持
当前显卡的显存使用率： 0.0 %
cudnn是否可用： True

进程已结束,退出代码0

```

12月7号

测试了一下如果device='cpu'的话程序的运行速度：平均在225秒左右

```

/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/test.py
epoch1
-----
train_loss: 0.6907      train_acc: 0.5294
val_loss: 0.6858      val_acc: 0.5550

225.75041341781616
epoch2
-----
train_loss: 0.6738      train_acc: 0.5900
val_loss: 0.6883      val_acc: 0.5200

225.51720929145813
epoch3
-----
train_loss: 0.6473      train_acc: 0.6212
val_loss: 0.6388      val_acc: 0.6725

225.55458617210388
epoch4
-----

```

感觉上来说使用gpu确实比用cpu快了很多，只不过gpu的运行速度仍然不知道为什么这么慢。所以gpu确实是有加速的，只不过本身就很慢。

目前能得到的结论就是cuda应该是好的，pytorch应该也没问题，唯一的一点问题就在于cuda的include库中找不到cudnn，但是pytorch显示cudnn可用。问了管服务器的师兄，他说cudnn是装好了的。。。

昨天发现没有cudnn，但今天又上去看了下，cudnn又出现了，照道理来说这么醒目的颜色我昨天肯定不会漏掉，可惜昨天忘截图了。

```

watney1024@H1FEN: /usr/local/cuda-12.3/include$ ls
builtin_types.h      cuda_pipeline_helpers.h      cufftXt.h      curand_uniform.h      ngpi_arithmetic_and_logical_operations.h  nvToolsExtSync.h
channel_descriptor.h  cuda_pipeline_primitives.h   curfile.h      cusolver_common.h     ngpi_color_conversion.h                  nvtx3
CL                   cuda_profiler_api.h         cupti_activity_deprecated.h  cusolverDn.h          ngpi_data_exchange_and_initialization.h  Openacc
common_functions.h   cudaProfiler.h              cupti_activity.h  cusolverMg.h          ngpi_filtering_functions.h               Openmp
cooperative_groups   cudaProfilerTypeDefs.h      cupti_callbacks.h  cusolverRf.h          ngpi_geometry_transforms.h              sm_20_atomic_functions.h
cooperative_groups.h cuda_runtime_api.h          cupti_checkpoint.h  cusolverSp.h          ngpi_linear_transforms.h                 sm_20_atomic_functions.hpp
crt                  cuda_runtime.h              cupti_driver_cbid.h  cusolverSp.LOWLEVEL_PREVIEW.h  ngpi_morphological_operations.h          sm_20_intrinsics.h
cub                   cuda_staint.h               cupti_events.h      cusparse.h            ngpi_statistics_functions.h              sm_20_intrinsics.hpp
cublas_api.h         cuda_surface_types.h        cupti.h           cusparse_v2.h         ngpi_support_functions.h                 sm_30_intrinsics.h
cublas.h             cuda_texture_types.h        cupti_metrics.h    device_atomic_functions.h  ngpi_threshold_and_compare_operations.h  sm_30_intrinsics.hpp
cublasLt.h           cudaTypedefs.h              cupti_nvtx_cbid.h  device_double_functions.hpp  ngps_arithmetic_and_logical_operations.h  sm_32_atomic_functions.h
cublas_v2.h          cudaTypeDefs.h              cupti_pcsampling.h  device_functions.h       ngps_conversion_functions.h              sm_32_intrinsics.h
cublasXt.h           cudaVDPAU.h                 cupti_pcsampling_util.h  device_launch_parameters.h  ngps_filtering_functions.h              sm_32_intrinsics.hpp
cuComplex.h          cuda_vdpau_interop.h        cupti_profiler_target.h  device_types.h          ngps_initialization.h                   sm_35_atomic_functions.h
cuda                 cudaVDPAUTypeDefs.h         cupti_result.h     driver_functions.h       ngps_statistics_functions.h              sm_35_intrinsics.h
cuda_ambarrier.h     cudnn_adv_infer.h           cupti_runtime_cbid.h  driver_types.h          ngps_support_functions.h                 sm_60_atomic_functions.h
cuda_ambarrier_helpers.h  cudnn_adv_infer_v8.h       cupti_sass_metrics.h  fatbinary_section.h     nv                                       nv
cuda_ambarrier_primitives.h  cudnn_adv_train_v8.h      cupti_target.h       generated_cuda_gl_interop_meta.h  nvblas.h                                nv_decode.h
cuda_bf16.hpp         cudnn_backend_v8.h          curand_discrete2.h    generated_cuda_meta.h     nvfunctional                             nvJitLink.h
cuda_device_runtime_api.h  cudnn_cnn_infer.h          curand_discrete.h    generated_cuda_runtime_api_meta.h  nvjpeg.h                                nvml.h
cudaEGL.h             cudnn_cnn_train_v8.h       curand_globals.h     generated_cuda_vdpau_interop_meta.h  nvperfc_common.h                        nvperf_cuda_host.h
cuda_egl_interop.h      cudnn_cnn_train_v8.h       curand.h              generated_nvtx_meta.h       nvperf_host.h                           nvperf_target.h
cudaEGLTypeDefs.h      cudnn_ops_infer.h           curand_kernel.h      host_config.h             nvPTXCompiler.h                         nvrtc.h
cuda_fp16.h           cudnn_ops_infer_v8.h        curand_lognormal.h   host_defines.h            nvToolsExtCuda.h                       nvToolsExtCudaRt.h
cuda_fp16.hpp          cudnn_ops_train.h           curand_atgp32dc_p_11213.h  math_constants.h          nvToolsExt.h                           nvToolsExtOpenCL.h
cuda_fp8.h            cudnn_ops_train_v8.h        curand_atgp32_kernel.h  math_functions.h          nvToolsExt.h
cudaGL.h              cudnn_v8.h                  curand_normal.h       mma.h                     nvToolsExt.h
cudaGL_interop.h      cudnn_version.h             curand_normal_static.h  mma.h                     nvToolsExt.h
cudaGLTypeDefs.h      cudnn_version_v8.h          curand_poisson.h      nppcore.h                 nvToolsExt.h
cuda.h                cufft.h                     curand_precalc.h      nppdefs.h                 nvToolsExt.h
cudalibxt.h           cufftw.h                                                             npp.h
cuda_occupancy.h
cuda_pipeline.h

```

而且cpu也是21年的，照道理来说光用cpu跑也不可能比我笔记本的cpu差吧

```

watney1024@HiFEM:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         46 bits physical, 57 bits virtual
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    1
Core(s) per socket:    28
Socket(s):              2
NUMA node(s):          2
Vendor ID:              GenuineIntel
CPU family:             6
Model:                 106
Model name:             Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz
Stepping:               6
CPU MHz:                2599.998
BogoMIPS:               4000.00
Virtualization:         VT-x
L1d cache:              2.6 MiB
L1i cache:              1.8 MiB
L2 cache:               70 MiB
L3 cache:               84 MiB
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55
Vulnerability Gather data sampling: Mitigation; Microcode
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:     Not affected
Vulnerability Mds:      Not affected
Vulnerability Meltdown: Not affected
Vulnerability Mmio stale data: Mitigation; Clear CPU buffers; SMT disabled
Vulnerability Retbleed: Not affected
Vulnerability Spec rstack overflow: Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence
Vulnerability Srbds:     Not affected

```

12月8日

后面又跑了下单个算子和单个数据经过cnn网络，发现服务器的用时都是小于自己笔记本的用时，之后经过多次输出用时，发现问题出在批次这里

```

def train(dataloader, model, loss_fn, optimizer):
    model.train()
    loss, current, n = 0.0, 0.0, 0
    #star_train = time.time()
    sum = 0
    i = 1
    start = time.time()
    for batch, (x, y) in enumerate(dataloader):
        star_train = time.time()

        image, y = x.to(device), y.to(device).float() # Ensure y is float
        output = model(image)

        cur_loss = loss_fn(output, y)
        cur_acc = torch.sum((y == output.round()).int()).float() / output.shape[0]

        # 反向传播
        optimizer.zero_grad()
        cur_loss.backward()

        optimizer.step()
        loss += cur_loss.item()

        n = n+1
        current += cur_acc.item()
        end_train = time.time()
        cost = end_train-star_train
        sum += cost
    end= time.time()
    print(sum)
    print(end-start)
    train_loss = loss / n
    train_acc = current / n

```

先输出的sum是每个batch用时总和，batch_size=16,1600张图片，因此sum是100个batch用时总和，end-start可以近似看做一个epoch训练用时，这里就出现了很大的差距

这是服务器上的

```

/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/test_all.py
epoch1
-----|
0.7805473804473877
120.12421941757202

```

这是笔记本上的

```

F:\program\anaconda\envs\pytorch\python.exe D:\code\Pycharm\猫狗大战\test_all.py
epoch1
-----
1.2281160354614258
4.1362175941467285

```

可以发现100个batch用时上，服务器快于笔记本，说明cuda确实是好的，但是总的时间不知道为什么变成了120。

12月10日

用cprofile看了下用时较多的部分，发现用时长的大多执行了1600次，细看了一下是dataloader这块用了大部分时间，目前就完全可以确定了，不是计算的问题。

名称	调用计数	时间 (毫秒)	自用时间(毫秒)
cpl.py	1	124459 100.0%	0 0.0%
train	1	120911 97.1%	29 0.0%
next_	101	120059 96.5%	2 0.0%
next_data	101	120047 96.5%	3 0.0%
fetch	100	120042 96.5%	0 0.0%
<listcomp>	100	102892 82.7%	6 0.0%
__getitem__	1600	102886 82.7%	40 0.0%
__call__	1600	100822 81.0%	20 0.0%
wrapped_call_impl	5100	53485 43.0%	13 0.0%
__call__	5100	53473 43.0%	23 0.0%
forward	1600	51443 41.3%	4 0.0%
normalize	1600	51439 41.3%	22 0.0%
normalize	1600	51397 41.3%	87 0.1%
__call__	1600	47631 38.3%	4 0.0%
to_tensor	1600	47626 38.3%	52 0.0%
<method 'to' of 'torch._C_TensorBase' objects>	1810	17388 14.0%	17341 13.9%
<method 'div_' of 'torch._C_TensorBase' objects>	1600	17149 13.8%	17149 13.8%
collate	300	17148 13.8%	3 0.0%
default_collate	100	17148 13.8%	0 0.0%
collate_tensor_fn	100	17134 13.8%	1 0.0%
<built-in method torch.stack>	100	17133 13.8%	17133 13.8%
<method 'div' of 'torch._C_TensorBase' objects>	1600	17130 13.8%	17130 13.8%
<method 'clone' of 'torch._C_TensorBase' objects>	1600	17086 13.7%	17086 13.7%
<method 'sub_' of 'torch._C_TensorBase' objects>	1600	16975 13.6%	16975 13.6%
<method 'contiguous' of 'torch._C_TensorBase' objects>	1600	13089 10.5%	13089 10.5%
find_and_load_unlocked	1673	3284 2.6%	7 0.0%
find_and_load	1674	3284 2.6%	8 0.0%
exec_module	1592	3282 2.6%	4 0.0%
load_unlocked	1607	3282 2.6%	7 0.0%
call_with_frames_removed	2016	3279 2.6%	1 0.0%
<built-in method builtins.__import__>	402	2575 2.1%	1 0.0%
handle_fromlist	3420	2538 2.0%	8 0.0%

问了学长之后，他是感觉是内存的问题，于是就想办法看看能不能做一些针对性的测试。看到to_tensor和normalize这些操作都花费了很多时间，为了调试快点，我把normalize删去了，运行时间也变成了70秒左右。

由于在linux上干很多事情都需要sudo权限，所以能够debug的工具还是比较有限的。

于是在网上搜索一下看看有没有类似的问题，在知乎上找了一个类似的，并且尝试了其中几种方法

[pytorch dataloader数据加载占用了大部分时间，各位大佬都是怎么解决的？](#)

- 查看了下自己硬盘的类型，发现确实是固态的，所以并不是由于文件存在机械硬盘上，导致硬盘的传输速度限制了运行时间。
- 尝试把图片都放到内存上面，这样就可以判断是不是从硬盘传输到内存的过程中出了问题，但是运行时间也并没有快多少

```
class InMemoryDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.images = []
        self.labels = []
        self.load_data()

    def load_data(self):
        # 使用 ImageFolder 获取数据和标签
        dataset = ImageFolder(self.root_dir)
        for img, label in dataset:
            self.images.append(img)
            self.labels.append(label)

    def __len__(self):
        return len(self.images)
```

```
def __getitem__(self, idx):
    image = self.images[idx]
    label = self.labels[idx]
    if self.transform:
        image = self.transform(image)
    return image, label
```

```
/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/fangneicunshang.py
train_loss: 0.6927      train_acc: 0.5062
70.64359617233276
Done!
```

- 设置pin_memory, 尝试了pin_memory=True但依旧没什么用。
- 在看到[pytorch性能瓶颈检查](#)之后便去搜索了torch.utils.bottleneck的用法, 然后就发现了最终解决问题的文章[pytorch备忘录](#)

博主和我一样都是gpu利用率很低, profile了之后发现大多数时间用在了读数据上, 他是把num_workers从8调成了0, 于是我进行了实验, 默认是0, 所以我设置成8跑了一下, 发现成功解决了问题

```
/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/cpf.py
train_loss: 0.6927      train_acc: 0.5131
1.5847134590148926

进程已结束,退出代码0
```

后来尝试了一下发现不同的num, 用时也不一样, 但除了设置成0以外并没有很夸张的差别。至此debug算是结束了, 但是我还是不太清楚具体的原因, 期待以后等我学了更多的东西之后能搞清楚其中原理吧。

```
/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/cpf.py
num_workers = 0 用时 68.91664934158325
num_workers = 1 用时 3.3042519092559814
num_workers = 2 用时 1.7168376445770264
num_workers = 3 用时 1.2563226222991943
num_workers = 4 用时 1.3598997592926025
num_workers = 5 用时 0.9273545742034912
num_workers = 6 用时 0.8057889938354492
num_workers = 7 用时 0.9113552570343018
num_workers = 8 用时 0.914792537689209
num_workers = 9 用时 0.8375527858734131
num_workers = 10 用时 0.8662154674530029
num_workers = 11 用时 0.8991374969482422
num_workers = 12 用时 0.9144086837768555
num_workers = 13 用时 1.013092279434204
num_workers = 14 用时 0.9335317611694336
num_workers = 15 用时 0.9164307117462158
num_workers = 16 用时 1.0003573894500732
num_workers = 17 用时 1.0188877582550049
num_workers = 18 用时 1.0037341117858887
num_workers = 19 用时 1.0560393333435059
num_workers = 20 用时 1.0690147876739502
best_num 6
```


再贴一张gpu利用率

```
+-----+
| NVIDIA-SMI 545.23.08                Driver Version: 545.23.08   CUDA Version: 12.3   |
+-----+-----+-----+-----+-----+-----+
| GPU  Name                Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|                                           | MIG M.         |
+-----+-----+-----+-----+-----+-----+
|   0   NVIDIA GeForce RTX 3090        On   | 00000000:B1:00.0 Off |           N/A        |
| 38%    52C    P2              138W / 350W |  8843MiB / 24576MiB |      36%      Default |
|                                           |           N/A        |
+-----+-----+-----+-----+-----+-----+

+-----+
| Processes:                         |
| GPU   GI    CI          PID    Type   Process name                      GPU Memory |
|      ID     ID                                 Usage      |
+-----+-----+-----+-----+-----+-----+
|   0   N/A   N/A         1609      G   /usr/lib/xorg/Xorg                  4MiB |
|   0   N/A   N/A         2307      G   /usr/lib/xorg/Xorg                  4MiB |
|   0   N/A   N/A       1503697      C   python                             376MiB |
|   0   N/A   N/A     3374439      C   ...4/anaconda3/envs/pytorch/bin/python 610MiB |
|   0   N/A   N/A     3871272      C   python                             7810MiB |
+-----+-----+-----+-----+-----+-----+
```

12月13日

人工智能的课的作业是编写个网络跑CIFAR10，我放到服务器上跑了，然后特地尝试了一下 num_workers=0，发现并没有之前那么夸张。

```
/home/watney1024/anaconda3/envs/pytorch/bin/python /home/watney1024/dl/mgdz/test_numworkers.py
num_workers = 0 用时 8.601812362670898
num_workers = 1 用时 8.036286115646362
num_workers = 2 用时 4.379769325256348
num_workers = 3 用时 3.934762716293335
num_workers = 4 用时 3.0078234672546387
num_workers = 5 用时 2.5217885971069336
num_workers = 6 用时 2.2830684185028076
num_workers = 7 用时 2.362724542617798
num_workers = 8 用时 2.3443613052368164
num_workers = 9 用时 2.215522527694702
num_workers = 10 用时 2.3180582523345947
num_workers = 11 用时 2.442931652069092
num_workers = 12 用时 2.284935474395752
num_workers = 13 用时 2.3842484951019287
num_workers = 14 用时 2.3823812007904053
num_workers = 15 用时 2.423767328262329
num_workers = 16 用时 2.5078678131103516
num_workers = 17 用时 2.5180580615997314
num_workers = 18 用时 2.537062644958496
num_workers = 19 用时 2.4300918579101562
num_workers = 20 用时 2.546705961227417
best_num 9
```