Task 2：了解 TensorFlow

> 1) Tensorflow 安装实验
>
> 2) 学习 Tensorflow 基本操作及使用方法

1) Tensorflow 安装实验

在第一次的实验中已经使用 anaconda 配置 python 3.6 的虚拟环境，并根据官网安装了 tensorflow cpu 最新版，此次实验是在实验室虚拟机上根据 cuda 相关环境安装其支持的 tensorflow gpu 版本，并使用 gpu 测试运行。

一、远程连接虚拟机

由于之前使用过云服务器，一般使用 ssh 远程连接 bash，服务器的默认的端口为 22，但由于每个虚拟机分配了 10 个连续端口，其中第一个端口为 ssh 端口，第二个端口为 vnc 远程桌面端口，第三个端口映射到 8080，可用于一些其他应用。所以使用如下命令进行连接

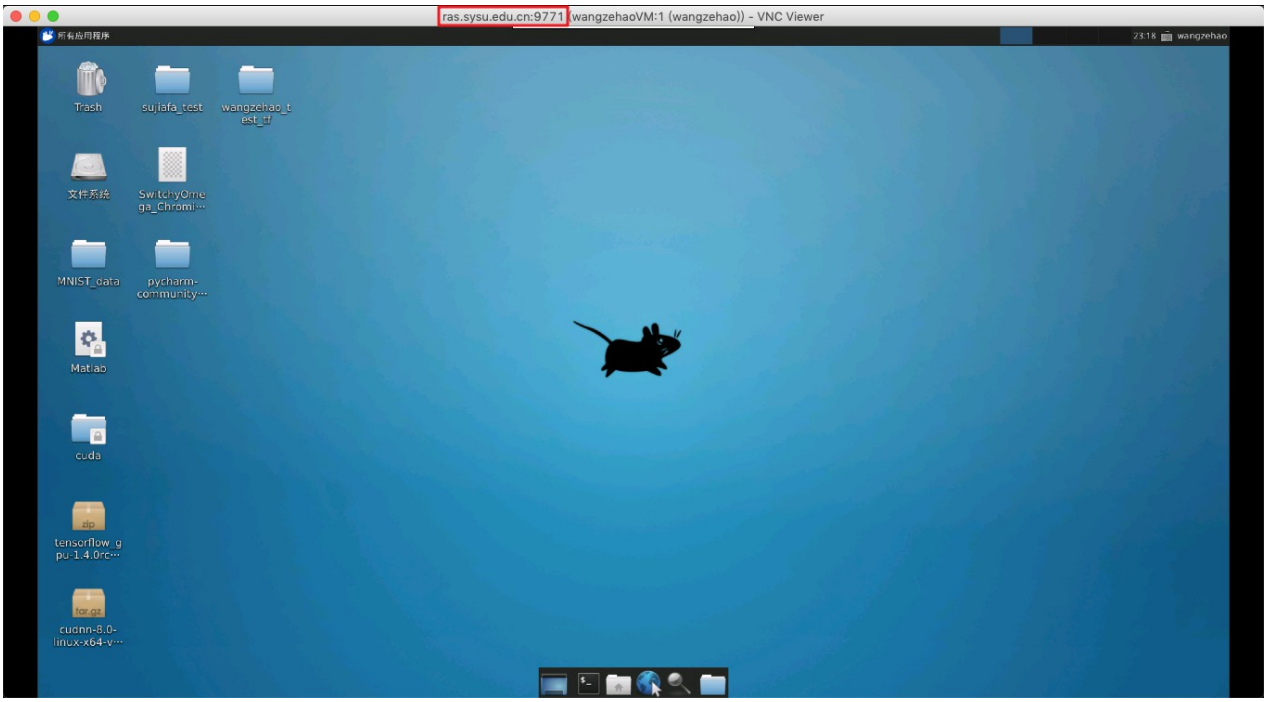
```
ssh -p 9770 wangzehao@ras.sysu.edu.cn
```

第一次登录会提示 "无法确认 host 主机的真实性，只知道它的公钥指纹，问你还想继续连接吗？"，选择 yes 后表示 host 主机已经得到认可。

但每次登录需要输入密码，比较麻烦，所以可以使用 SSH 的公钥登录，可以省去输入密码的步骤。所谓"公钥登录"，原理很简单，就是用户将自己的公钥储存在远程主机上。登录的时候，远程主机会向用户发送一段随机字符串，用户用自己的私钥加密后，再发回来。远程主机用事先储存的公钥进行解密，如果成功，就证明用户是可信的，直接允许登录 shell，不再要求密码。使用如下命令将自己的公钥传送到远程主机 host 上即可。

```
ssh-copy-id -p 9770 wangzehao@ras.sysu.edu.cn
```

另一种可以使用虚拟机图形化的桌面环境的连接方法是使用 VNC，下载客户端后并登录后如下



由于没有配置虚拟机的科学上网环境，关于文件传输，可以使用 scp 命令或者直接使用 filezilla 的 sftp 协议进行上传和下载等文件管理。

二、Amax 虚拟机深度学习框架安装

> *因为某块显卡的原因，Amax 的 Cuda 仅支持 Cuda 8.0 版本，以下的环境 & DL 框架均针对 Cuda 8.0*

因为虚拟机已经配置好 cuda 8.0 了（最新为 9.0，但由于某张显卡不支持 Cuda 9 的原因，所以虚拟机暂时配置Cuda 8），所以只需要安装配套的 cuDNN v6.0，即安装头文件和库，具体按照文档就可以。

之后安装 anaconda，然后新建一个虚拟环境，使用 pip 安装 *TensorFlow r1.4* 版本（当前 TensorFlow 最高版本是r1.9，但是只能兼容Cuda 9，所以建议使用能兼容 Cuda 8 的TensorFlow r1.4）

这里注意安装完 anaconda 后并配置环境变量后，使用 python 命令启动的不是虚拟机之前安装在 /usr/bin 中的 3.5.2 版本的 python 了，而是 /home/wangzehao/anaconda3/bin 中的 3.6.3 版本的 python。由于之后避免遇到 python 版本不同的问题，直接 `conda install python=3.5.2`，然后进入 anaconda 的 base 虚拟环境，其中包括了 conda, numpy, scipy, ipython notebook 等科学包及其依赖项的发行版本。安装需要的环境：

```
sudo apt-get install libcupti-dev
sudo apt-get install python3-pip python3-dev
```

按照教程使用校园网下载安装包`https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.4.0rc1-cp35-cp35m-linux_x86_64.whl`（速度特别快），接着使用 FileZilla 通过 SFTP 上传到服务器

最后直接命令行进入存放目录，使用国内镜像（清华）加速安装（还会下载其他一些依赖）：

```
# 在 conda 中使用 pip，而不是 pip3
pip upgrade
pip install --upgrade Downloads/tensorflow_gpu-1.4.0rc1-cp35-cp35m-linux_x86_64.whl -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

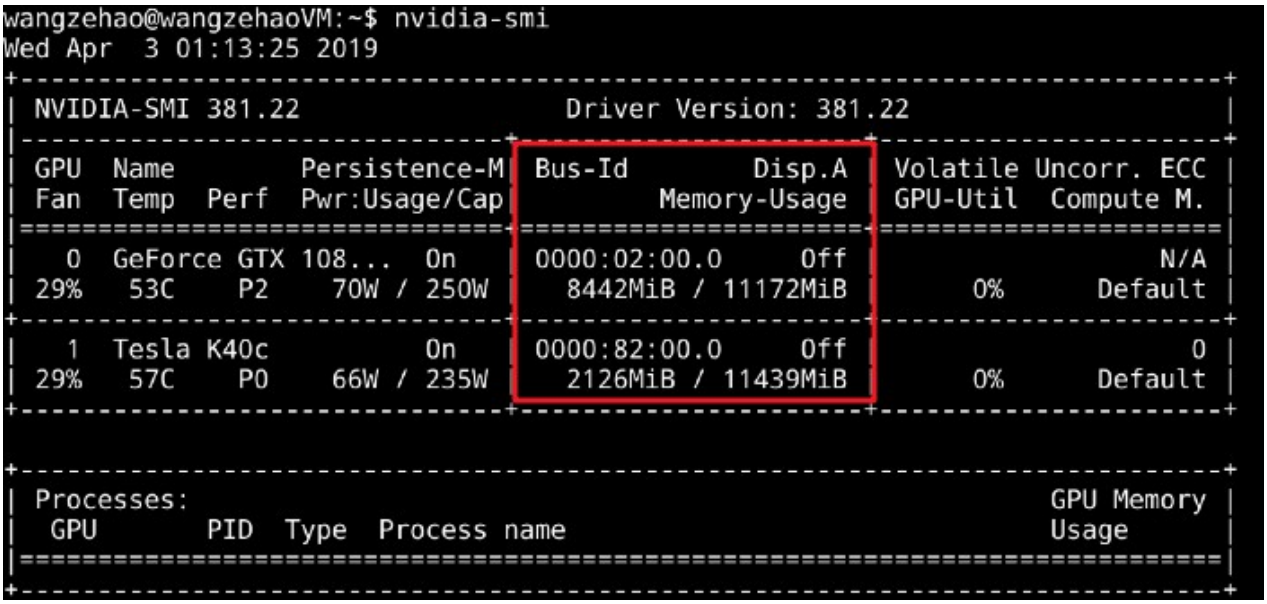
```
wangzehao@wangzehaoVM:~$ conda list | grep -E 'numpy|python|pandas|jupyter|pip|tensorflow'
bleach                    1.5.0                     <pip>
enum34                    1.1.6                     <pip>
html5lib                  0.9999999                 <pip>
ipython                   6.5.0                    py35_0
ipython_genutils          0.2.0              py35hc9e07d0_0
jupyter                   1.0.0                    py35_7
jupyter_client            5.2.3                    py35_0
jupyter_console           5.2.0              py35h4044a63_1
jupyter_core              4.4.0                    py35_0
jupyterlab                0.34.9                   py35_0
jupyterlab_launcher       0.13.1                   py35_0
Markdown                  3.1                       <pip>
msgpack-python            0.5.6              py35h6bb024c_1
numpy                     1.16.2                    <pip>
numpy                     1.13.3             py35ha266831_3
numpydoc                  0.8.0                    py35_0
pandas                    0.20.3             py36h842e28d_2
pip                       10.0.1                   py35_0
pip                       19.0.3                    <pip>
protobuf                  3.7.1                     <pip>
python                    3.5.2                        0
python-dateutil           2.7.3                    py35_0
tensorflow-gpu            1.4.0rc1                  <pip>
tensorflow-tensorboard    0.4.0                     <pip>
```

最后验证是否安装成功，运行一下代码`python test.py`，由于安装的 tf 是支持 gpu 的，所以 GPU 设备有优先权。例如，如果`matmul`同时存在 CPU 和 GPU 核函数，在同时有`cpu:0`和`gpu:0`设备的系统中，`gpu:0`会被选来运行`matmul`。

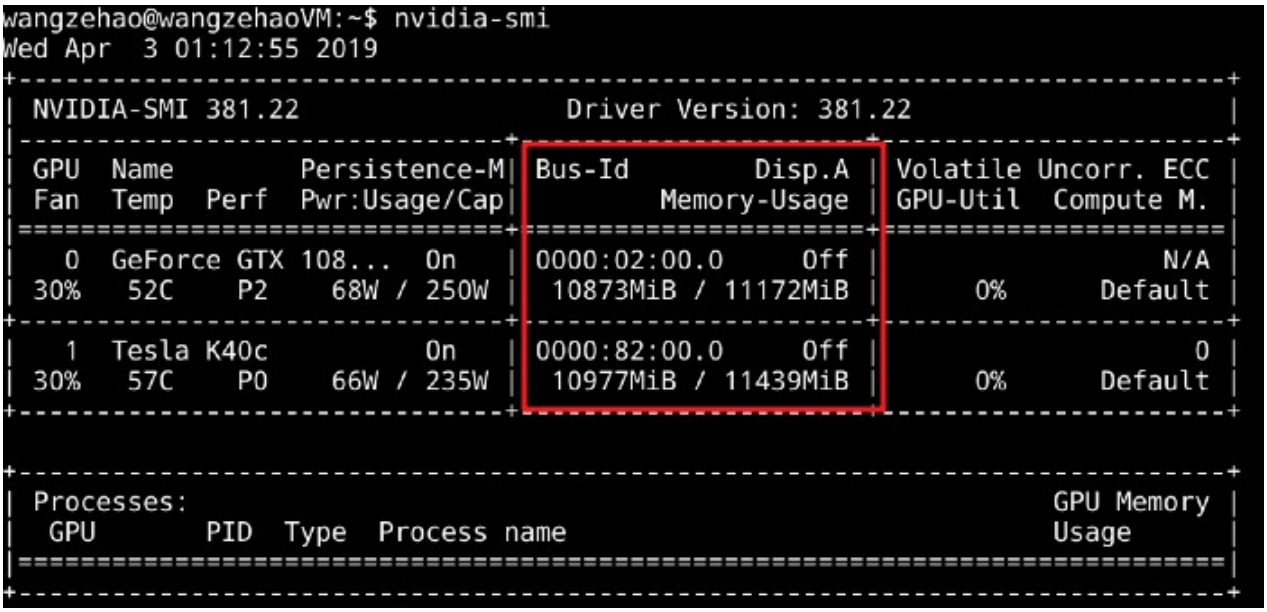
```
import tensorflow as tf
a = tf.constant(6, dtype=tf.uint8)
with tf.Session() as sess:
    while True:
        a_val = sess.run(a)
        print(a_val)
```

命令行输入`watch nvidia-smi`查看当前显卡状态，参数含义参考

运行之前



运行之后



可以看到两块显存基本全部占满，实际使用时，需要按需调用 GPU，因为默认情况下 TensorFlow 会映射进程可见的所有 GPU 的几乎所有 GPU 内存（取决于 `CUDA_VISIBLE_DEVICES`）。通过减少内存碎片，可以更有效地使用设备上相对宝贵的 GPU 内存资源。

在某些情况下，最理想的是进程只分配可用内存的一个子集，或者仅根据进程需要增加内存使用量。TensorFlow 在 Session 上提供两个 Config 选项来进行控制。

第一个是 `allow_growth` 选项，它试图根据运行时的需要来分配 GPU 内存：它刚开始分配很少的内存，随着 Session 开始运行并需要更多 GPU 内存，我们会扩展 TensorFlow 进程所需的 GPU 内存区域。请注意，我们不会释放内存，因为这可能导致出现更严重的内存碎片情况。要开启此选项，请通过以下方式在 ConfigProto 中设置选项：

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

第二个是 `per_process_gpu_memory_fraction` 选项，它可以决定每个可见 GPU 应分配到的内存占总内存量的比例。例如，您可以通过以下方式指定 TensorFlow 仅分配每个 GPU 总内存的 40%：

```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
session = tf.Session(config=config, ...)
```

当我们想知道指令和张量被分配到哪个设备，可以使用如下配置记录设备分配方式

```
config = tf.ConfigProto()
config.log_device_placement=True
session = tf.Session(config=config, ...)
```

当需要强制使用 CPU 进行计算时

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
```

2) 学习 Tensorflow 基本操作及使用方法