

# 彻底搞懂深度学习环境

## 1. 基于Anaconda的深度学习环境配置

首先安装Anaconda，然后创建虚拟环境，在虚拟环境里面安装需要的库。针对不同的实际项目，其依赖的工具库是不一样的，要采用虚拟环境进行隔离，这样你可以安装多个虚拟环境，各个项目互不相干。

### (1) 下载Anaconda&安装

<https://www.anaconda.com/download/success>

下载安装完之后，配置下.condarc文件，采用清华源，主要用于安装其库时加速。**注意是修改channels中内容**

文件地址：

window系统：C:\Users\your\_usrername\.condarc

Ubuntu系统：~/.condarc

```
1 show_channel_urls: true
2 channels:
3   - https://mirrors.ustc.edu.cn/anaconda/cloud/bioconda/
4   - https://mirrors.ustc.edu.cn/anaconda/cloud/bioconda/linux-64
5   - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorch/
6   - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/menpo/
7   - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda/
8   - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
9   - https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
10  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkg/free/
11  - defaults
```

### (2) conda命令操作

#### ① 创建python虚拟环境

```
1 conda create --name your_env_name python=3.10
```

## ② 激活python虚拟环境

```
1 conda activate your_env_name
```

## ③ 虚拟环境内安装相关包：

```
1 conda install pytorch=1.12.1 torchaudio=0.12.1 cudatoolkit=11.6 -c pytorch -c  
conda-forge （依据需要安装相关版本）
```

或者

```
1 pip install torch==2.0.1+cu117 torchvision==0.15.2+cu117  
torchaudio==2.0.2+cu117 -f https://download.pytorch.org/whl/torch_stable.html
```

## ④ 查看虚拟环境列表

```
1 conda env list
```

## ⑤ 如果有些虚拟环境不想要了，可以删除

```
1 conda remove --name your_env_name --all
```

## (3) IDE工具

建议使用VS Code，比Pycharm更方便。

做到这里之后，你就可以从github上下载代码或者自己写代码，虽然anaconda会帮我们解决很多问题，但是，或多或少会遇到一些奇奇怪怪的问题，cuda的问题、库的问题、需要手动配置环境，网上信息是个大杂烩，很难快速定位和解决问题，如果你掌握了一些基础知识会大有裨益。

## 2.深度学习环境进阶

# (1) GPU、cuda、cudnn、nvcc、cuda driver、cudatoolkit到底是什么？

## GPU型号含义

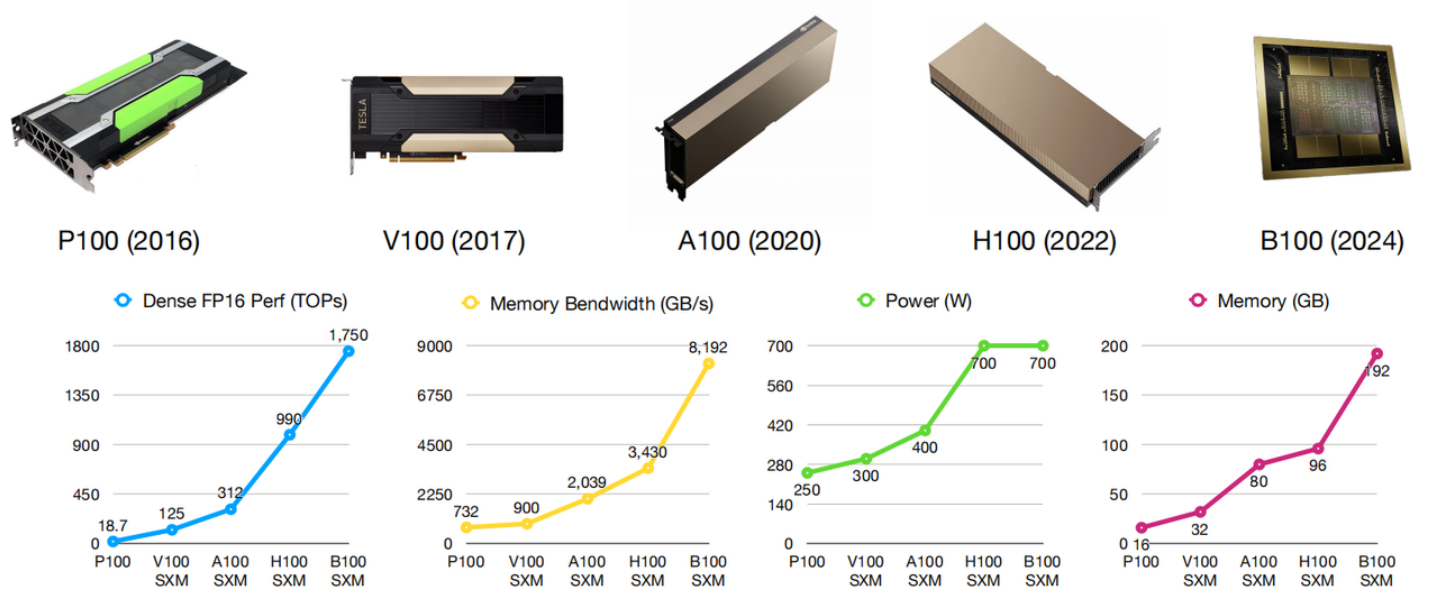
参考<https://chenrudan.github.io/blog/2015/12/20/introductionofgpuhardware.html>

- 显卡：简单理解这个就是我们前面说的GPU，尤其指NVIDIA公司生产的GPU系列，因为后面介绍的cuda,cudnn都是NVIDIA公司针对自身的GPU独家设计的。
- 显卡驱动：很明显就是字面意思，通常指NVIDIA Driver，其实它就是一个驱动软件，而前面的显卡就是硬件。
- gpu架构：Fermi、Kepler、Maxwell、Pascal、Ampere、Hopper
- 显卡系列：GeForce、Quadro、Tesla

gpu架构指的是硬件的设计方式，例如流处理器簇中有多少个core、是否有L1 or L2缓存、是否有双精度计算单元等等。每一代的架构是一种思想，如何去更好完成并行的思想。

显卡系列在本质上并没有什么区别，只是NVIDIA希望区分成三种选择，GeFore用于家庭娱乐，Quadro用于工作站，而Tesla系列用于服务器。

## NVIDIA



## CUDA名称含义

有人说CUDA就是一门编程语言，像C,C++,python 一样，也有人说CUDA是API。CUDA英文全称是Compute Unified Device Architecture，是显卡厂商NVIDIA推出的运算平台。CUDA™是一种由NVIDIA推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题。按照官方的说法是，CUDA是一个并行计算平台和编程模型，能够使得使用GPU进行通用计算变得简单和优雅。

GPU Computing Applications						
Libraries and Middleware						
cuDNN TensorRT	cuFFT, cuBLAS, cuRAND, cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL, SVM, OpenCurrent	PhysX, OptiX, iRay	MATLAB Mathematica
Programming Languages						
C	C++	Fortran	Java, Python, Wrappers	DirectCompute	Directives (e.g., OpenACC)	
CUDA-enabled NVIDIA GPUs						
Turing Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier	GeForce 2000 Series	Quadro RTX Series	Tesla T Series		
Volta Architecture (Compute capabilities 7.x)	DRIVE/JETSON AGX Xavier			Tesla V Series		
Pascal Architecture (Compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series		
Maxwell Architecture (Compute capabilities 5.x)	Tegra X1	GeForce 900 Series	Quadro M Series	Tesla M Series		
Kepler Architecture (Compute capabilities 3.x)	Tegra K1	GeForce 700 Series GeForce 600 Series	Quadro K Series	Tesla K Series		
	EMBEDDED	CONSUMER DESKTOP, LAPTOP	PROFESSIONAL WORKSTATION	DATA CENTER		

## cuda

这个其实就是一个专门为深度学习计算设计的软件库，里面提供了很多专门的计算函数，如卷积等。从上图也可以看到，还有很多其他的软件库和中间件，包括实现c++ STL的thrust、实现gpu版本blas的cublas、实现快速傅里叶变换的cuFFT、实现稀疏矩阵运算操作的cuSparse以及实现深度学习网络加速的cuDNN等等，具体细节可参阅<https://developer.nvidia.com/gpu-accelerated-libraries>

## CUDA Toolkit

参考<https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#major-components>

CUDA Toolkit由以下组件组成：

- Compiler: CUDA-C和CUDA-C++编译器 `NVCC` 位于 `bin/` 目录中。它建立在 `NVVM` 优化器之上，而 `NVVM` 优化器本身构建在 `LLVM` 编译器基础结构之上。因此开发人员可以使用 `nvm/` 目录下的 Compiler SDK来直接针对NVVM进行开发。
- Tools: 提供一些像 `profiler` , `debuggers` 等工具，这些工具可以从 `bin/` 目录中获取
- Libraries: 下面列出的部分科学库和实用程序库可以在 `lib/` 目录中使用(Windows上的DLL位于 `bin/` 中)，它们的接口在 `include/` 目录中可获取。
  - cudart: CUDA Runtime
  - cudadevrt: CUDA device runtime
  - cupti: CUDA profiling tools interface
  - nvml: NVIDIA management library
  - nvrtc: CUDA runtime compilation

- cublas: BLAS (Basic Linear Algebra Subprograms, 基础线性代数程序集)
- cublas\_device: BLAS kernel interface
- ...
- CUDA Samples: 演示如何使用各种CUDA和library API的代码示例。可在Linux和Mac上的 `samples/` 目录中获得, Windows上的路径是 `C:\ProgramData\NVIDIA Corporation\CUDA Samples` 中。在Linux和Mac上, `samples/` 目录是只读的, 如果要对它们进行修改, 则必须将这些示例复制到另一个位置。
- CUDA Driver: 运行CUDA应用程序需要系统至少有一个具有CUDA功能的GPU和与CUDA工具包兼容的驱动程序。每个版本的CUDA工具包都对应一个最低版本的CUDA Driver, 也就是说如果你安装的CUDA Driver版本比官方推荐的还低, 那么很可能会无法正常运行。CUDA Driver是向后兼容的, 这意味着根据CUDA的特定版本编译的应用程序将继续在后续发布的Driver上也能继续工作。通常为了方便, 在安装CUDA Toolkit的时候会默认安装CUDA Driver。在开发阶段可以选择默认安装Driver, 但是对于像Tesla GPU这样的商用情况时, 建议在[官方](#)安装最新版本的Driver。目前(2019年10月)的CUDA Toolkit和CUDA Driver版本的对应情况如下:

CUDA Toolkit	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 10.1 (10.1.105 general release, and updates)	>= 418.39	>= 418.96
CUDA 10.0.130	>= 410.48	>= 411.31
CUDA 9.2 (9.2.148 Update 1)	>= 396.37	>= 398.26
CUDA 9.2 (9.2.88)	>= 396.26	>= 397.44
CUDA 9.1 (9.1.85)	>= 390.46	>= 391.29
CUDA 9.0 (9.0.76)	>= 384.81	>= 385.54
CUDA 8.0 (8.0.61 GA2)	>= 375.26	>= 376.51
CUDA 8.0 (8.0.44)	>= 367.48	>= 369.30
CUDA 7.5 (7.5.16)	>= 352.31	>= 353.66
CUDA 7.0 (7.0.28)	>= 346.46	>= 347.62

`nvcc` & `nvidia-smi`

`nvcc` 其实就是CUDA的编译器, 可以从CUDA Toolkit的 `/bin` 目录中获取, 类似于 `gcc` 就是c语言的编译器。由于程序是要经过编译器编程成可执行的二进制文件, 而cuda程序有两种代码, 一种是运行在cpu上的host代码, 一种是运行在gpu上的device代码, 所以 `nvcc` 编译器要保证两部分代码能够编译成二进制文件在不同的机器上执行。nvcc涉及到的文件后缀及相关意义如下表

文件后缀	意义
.cu	cuda源文件，包括host和device代码
.cup	经过预处理的cuda源文件，编译选项--preprocess/-E
.c	c源文件
.cc/.cxx/.cpp	c++源文件
.gpu	gpu中间文件，编译选项--gpu
.ptx	类似汇编代码，编译选项--ptx
.o/.obj	目标文件，编译选项--compile/-c
.a/.lib	库文件，编译选项--lib/-lib
.res	资源文件
.so	共享目标文件，编译选项--shared/-shared
.cubin	cuda的二进制文件，编译选项--cubin

`nvidia-smi` 全程是NVIDIA System Management Interface，它是一个基于前面介绍过的NVIDIA Management Library(NVML) 构建的命令行实用工具，旨在帮助管理和监控NVIDIA GPU设备。

`nvcc` 和 `nvidia-smi` 显示的CUDA版本不同？

在我们实验室的服务器上 `nvcc --version` 显示的结果如下：

```
1 nvcc: NVIDIA (R) Cuda compiler driver
2 Copyright (c) 2005-2022 NVIDIA Corporation
3 Built on Wed_Sep_21_10:41:10_Pacific_Daylight_Time_2022
4 Cuda compilation tools, release 11.8, V11.8.89
5 Build cuda_11.8.r11.8/compiler.31833905_0
```

而 `nvidia-smi` 显示结果如下：

```
1 Fri Nov 8 11:38:42 2024
2 +-----+
3 | NVIDIA-SMI 561.03                Driver Version: 561.03          CUDA
  Version: 12.6                |
```

---

可以看到 `nvcc` 的CUDA 版本是11.8，而 `nvidia-smi` 的CUDA版本是12.6。

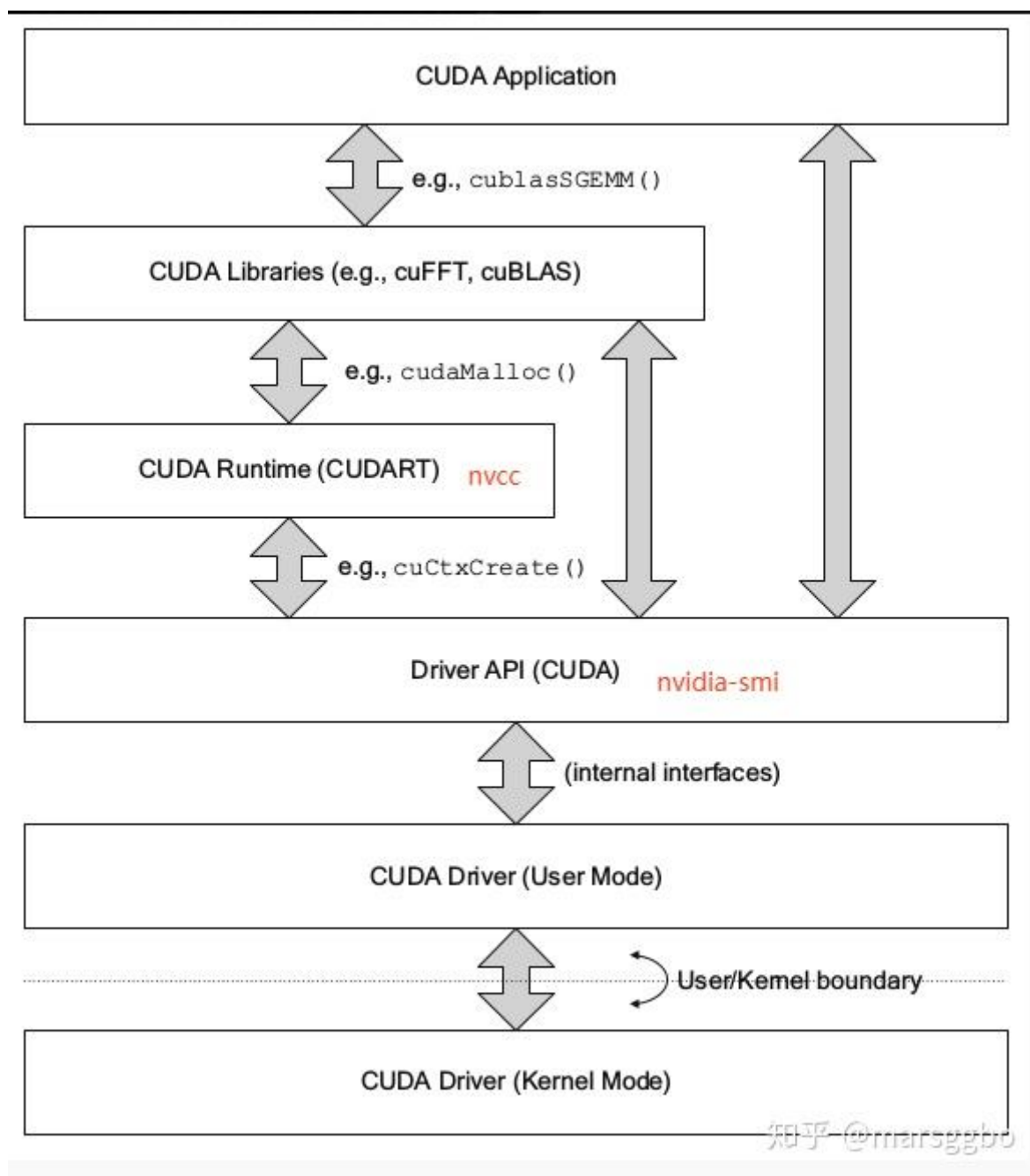
CUDA有两个主要的API：runtime(运行时) API和driver API。这两个API都有对应的CUDA版本（如11.8和12.6等）。

- 用于支持driver API的必要文件(如 `libcudart.so` )是由GPU driver installer安装的。 `nvidia-smi` 就属于这一类API。 [（安装cuda驱动里对应的版本）](#)
- 用于支持runtime API的必要文件(如 `libcudart.so` 以及 `nvcc` )是由CUDA Toolkit installer安装的。 [（在虚拟环境里面安装的版本）](#)

## runtime和driver API区别

下图很清楚的展示前面提到的各种概念之间的关系，其中runtime和driver API在很多情况非常相似，也就是说用起来的效果是等价的，但是你不能混合使用这两个API，因为二者是互斥的。也就是说在开发过程中，你只能选择其中一种API。简单理解二者的区别就是：runtime是更高级的封装，开发人员用起来更方便，而driver API更接近底层，速度可能会更快。





两种API详细的区别如下：

- 复杂性

- runtime API通过提供隐式初始化、上下文管理和模块管理来简化设备代码管理。这使得代码更简单，但也缺乏驱动程序API所具有的控制级别。
- 相比之下，driver API提供了更细粒度的控制，特别是在上下文和模块加载方面。实现内核启动要复杂得多，因为执行配置和内核参数必须用显式函数调用指定。

- 控制

- 对于runtime API，其在运行时，所有内核都在初始化期间自动加载，并在程序运行期间保持加载状态。
- 而使用driver API，可以只加载当前需要的模块，甚至动态地重新加载模块。driver API也是语言独立的，因为它只处理 `cubin` 对象。



- 上下文管理 上下文管理可以通过driver API完成，但是在runtime API中不公开。相反，runtime API自己决定为线程使用哪个上下文：
  - 如果一个上下文通过driver API成为调用线程的当前上下文，runtime将使用它，
  - 如果没有这样的上下文，它将使用“主上下文(primary context)”。

主上下文会根据需要创建，每个设备每个进程一个上下文，并进行引用计数，然后在没有更多的引用时销毁它们。在一个进程中，所有runtime API的用户都将共享主上下文，除非上下文已成为每个线程的当前上下文。runtime使用的上下文，即当前上下文或主上下文，可以用

`cudaDeviceSynchronize()` 同步，也可以用 `cudaDeviceReset()` 销毁。但是，将runtime API与主上下文一起使用会有tradeoff。例如，对于那些需要给较大的软件包写插件的开发者来说会带来不少麻烦，因为如果所有的插件都在同一个进程中运行，它们将共享一个上下文，但可能无法相互通信。也就是说，如果其中一个在完成所有CUDA工作后调用 `cudaDeviceReset()`，其他插件将失败，因为它们使用的上下文在它们不知情的情况下被破坏。为了避免这个问题，CUDA clients可以使用driver API来创建和设置当前上下文，然后使用runtime API来处理它。但是，上下文可能会消耗大量的资源，比如设备内存、额外的主机线程和设备上上下文切换的性能成本。当将driver API与基于runtime API(如cuBLAS或cuFFT)构建的库一起使用时，这种runtime-driver上下文共享非常重要。

## (2) Linux中PATH、LIBRARY\_PATH、LD\_LIBRARY\_PATH的区别

参考<https://link.zhihu.com/?target=https%3A//www.imooc.com/article/43747>

### PATH

PATH是可执行文件路径，是我们最常接触到的，因为命令行中的每句能运行的命令，如ls、top、ps等，都是系统通过PATH找到了这个命令执行文件的所在位置，再run这个命令（可执行文件）。比如说，在用户的目录 `~/mycode/` 下有一个bin文件夹，里面放了有可执行的二进制文件、shell脚本等。如果想要在任意目录下都能运行上述bin文件夹的可执行文件，那么只需要把这个bin的路径添加到PATH即可，方法如下：

```
1 vim ~/.bashrc
2 PATH=$PATH:~/mycode/bin
```

### LIBRARY\_PATH和LD\_LIBRARY\_PATH

- `LIBRARY_PATH` 是**程序编译期间**查找动态链接库时指定查找共享库的路径
- `LD_LIBRARY_PATH` 是**程序加载运行期间**查找动态链接库时指定除了系统默认路径之外的其他路径

两者的共同点是库，库是这两个路径和PATH路径的区别，PATH是可执行文件。

两者的差异点是使用时间不一样。一个是编译期，对应的是开发阶段，如gcc编译；一个是加载运行期，对应的是程序已交付的使用阶段。

配置方法也是类似：

```
1 export LD_LIBRARY_PATH=LD_LIBRARY_PATH:XXXX
2 export LIBRARY_PATH=LIBRARY_PATH:XXXX
```

### (3) 多版本CUDA切换

参考<https://blog.csdn.net/Maple2014/article/details/78574275>

基于前面几个小节介绍，现在再来介绍如何管理多版本CUDA就会好懂很多了。

### cuda 的下载与安装方法选择

到 <https://developer.nvidia.com/cuda-downloads> 下载所需版本，以 cuda\_9.0.176\_384.81\_linux.run 为例：

#### Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX			
Architecture	x86_64	ppc64le				
Distribution	Fedora	OpenSUSE	RHEL	CentOS	SLES	Ubuntu
Version	17.04	16.04				
Installer Type	runfile (local)	deb (local)	deb (network)	cluster (local)		

#### Download Installer for Linux Ubuntu 16.04 x86\_64

The base installer is available for download below.

> Base Installer

Download [1.6 GB]

Installation Instructions:

1. Run `sudo sh cuda_9.0.176_384.81_linux.run`
2. Follow the command-line prompts

The CUDA Toolkit contains Open-Source Software. The source code can be found [here](#).

The checksums for the installer and patches can be found in [Installer Checksums](#).

For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

知乎 @marsggbo  
<http://blog.csdn.net/Maple2014>

建议选择使用 `.run` 文件安装，因为使用 `.deb` 可能会将已经安装的较新的显卡驱动替换。

## cuda 安装

进入到放置 `cuda_9.0.176_384.81_linux.run` 的目录：

- 1 `sudo chmod +x cuda_9.0.176_384.81_linux.run` (为 `cuda_9.0.176_384.81_linux.run` 添加可执行权限)
- 2 `./cuda_9.0.176_384.81_linux.run` (安装 `cuda_9.0.176_384.81_linux.run`)

在安装过程中截取其中比较重要的几个选择：

- 1 Do you accept the previously read EULA?
- 2 accept/decline/quit: accept
- 3 Install NVIDIA Accelerated Graphics Driver for Linux-x86\_64 384.81?
- 4 (y)es/(n)o/(q)uit: n (如果在这之前已经安装好更高版本的显卡驱动就不需要再重复安装, 如果需要重复安装就选择 yes, 此外还需要关闭图形界面。)
- 5 Install the CUDA 9.0 Toolkit?
- 6 (y)es/(n)o/(q)uit: y
- 7 Enter Toolkit Location[ default is /usr/local/cuda-9.0 ]: (一般选择默认即可, 也可以选择安装在其他目录, 在需要用的时候指向该目录或者使用软连接 link 到 /usr/local/cuda。)
- 8 /usr/local/cuda-9.0 is not writable.
- 9 Do you wish to run the installation with 'sudo'?
- 10 (y)es/(n)o: y
- 11 Please enter your password:
- 12 Do you want to install a symbolic link at /usr/local/cuda? (是否将安装目录通过软连接的方式 link 到 /usr/local/cuda, yes or no 都可以, 取决于你是否使用 /usr/local/cuda 为默认的 cuda 目录。)(y)es/(n)o/(q)uit: n
- 13 Install the CUDA 9.0 Samples?
- 14 (y)es/(n)o/(q)uit: n

前面选择的一些汇总：

- 1 Driver: Not Selected
- 2 Toolkit: Installed in /usr/local/cuda-9.0
- 3 Samples: Not Selected
- 4 Please make sure that
- 5 PATH includes /usr/local/cuda-9.0/bin
- 6 LD\_LIBRARY\_PATH includes /usr/local/cuda-9.0/lib64, or, add /usr/local/cuda-9.0/lib64 to /etc/ld.so.conf and run ldconfig as root
- 7 To uninstall the CUDA Toolkit, run the uninstall script in /usr/local/cuda-9.0/bin

```
8 Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-9.0/doc/pdf for
   detailed information on setting up CUDA.
9 ***WARNING: Incomplete installation! This installation did not install the
   CUDA Driver. A driver of version at least 384.00 is required for CUDA 9.0
   functionality to work.
10 To install the driver using this installer, run the following command,
   replacing <CudaInstaller> with the name of this run file:
11     sudo <CudaInstaller>.run -silent -driver
```

安装完成后可以在 /usr/local 目录下看到：

```
1 cuda-8.0 之前安装的cuda-8.0
2 cuda-9.0 刚刚安装的cuda-9.0
3 cuda cuda-8.0 的软连接
```

## 多个 cuda 版本之间进行切换

将 `~/.bashrc` 下与cuda相关的路径都改为 `/usr/local/cuda/` 而不使用 `/usr/local/cuda-8.0/` 或 `/usr/local/cuda-9.0/`。

```
1 #在切换cuda版本时
2 rm -rf /usr/local/cuda#删除之前创建的软链接
3 sudo ln -s /usr/local/cuda-8.0/ /usr/local/cuda/
4 nvcc --version #查看当前 cuda 版本
5 nvcc: NVIDIA (R) Cuda compiler driver
6 Copyright (c) 2005-2016 NVIDIA Corporation
7 Built on Mon_Jan_23_12:24:11_CST_2017
8 Cuda compilation tools, release 8.0, V8.0.62
9
10 #cuda8.0 切换到 cuda9.0
11 rm -rf /usr/local/cuda
12 sudo ln -s /usr/local/cuda-9.0/ /usr/local/cuda/
13 nvcc --version
```

## 参考资料

1. [https://www.bilibili.com/video/BV1W1D1YqERC/?spm\\_id\\_from=333.999.0.0&vd\\_source=102ea859b1641807888984fa32fea341](https://www.bilibili.com/video/BV1W1D1YqERC/?spm_id_from=333.999.0.0&vd_source=102ea859b1641807888984fa32fea341)
2. <https://zhuanlan.zhihu.com/p/91334380>

