Facial Key Points Detection

Setup & Configuration Document

**Tingwen Bao & Sanjay Dorairaj**

**W207-5 - Spring 2017**

# Document Overview

This document captures the setup and execution instructions for the Facial Keypoints Detection project. It lists the steps required to configure AWS EC2 for GPU execution, installation of Theano, Lasagne, Jupyter and related software and instructions to execute and monitor GPU usage.

The Facial Keypoints Detection project is a Kaggle competition project that challenges folks to improve the ability to accurately detect the location of keypoints on face images. It can be found here - <https://www.kaggle.com/c/facial-keypoints-detection>

[**Document Overview**](#_tizkyb9b6dcw) **2**

[**AWS EC2 Initial Setup**](#_91j8mwau5bf7) **2**

[**Theano/Lasagne Configuration**](#_vomolapzsne3) **3**

[**Adding cuDNN support**](#_33shrycbmqqw) **4**

[**Jupyter Notebook configuration**](#_25l4woeztyel) **4**

[**Validating GPU support**](#_yg8ysgczsakc) **5**

[./deviceQuery Check](#_meebn7umj0vi) 5

[Using nvidia-smi](#_scht5w68mws6) 6

[Using glances](#_hls7bnjspnix) 7

[**Basic Troubleshooting**](#_bacql6p3ohju) **8**

[NVIDIA Driver Issues](#_k75qjo4lnzrs) 8

[MNIST data access issues](#_y0vt0rcmn5xs) 8

# AWS EC2 Initial Setup

This section captures step required to instantiate an AWS EC2 server with GPU support and install the NVIDIA CUDA package.

* 1. Select the AWS g2.2xlarge instance on AWS
  2. Followed instructions here <http://markus.com/install-theano-on-aws/> with the below exceptions
  3. Did not use the AMI provided.
  4. Used Ubuntu 16.04 and ran through the instructions.
  5. The documentation refers to a 14.04 deb...I used the one for 16.04
     1. sudo wget <http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cuda-repo-ubuntu1604_8.0.61-1_amd64.deb>
  6. Uninstalled the nvidia-375 driver that installs by default and replaced with nvidia-367 driver.
     1. sudo apt-get remove nvidia-375
     2. sudo apt-get install nvidia-367
  7. Once that was done, I was able to do the usual reboot and run deviceQuery with no issues.

Listed below are bash instructions for the above setup

|  |
| --- |
| sudo apt-get update  sudo apt-get -y dist-upgrade  screen -S “theano”  sudo apt-get install -y gcc g++ gfortran build-essential git wget linux-image-generic libopenblas-dev python-dev python-pip python-nose python-numpy python-scipy  sudo wget [http://developer.download.n...](http://disq.us/url?url=http%3A%2F%2Fdeveloper.download.nvidia.com%2Fcompute%2Fcuda%2Frepos%2Fubuntu1604%2Fx86_64%2Fcuda-repo-ubuntu1604_8.0.61-1_amd64.deb%3AoqZP-jNHSUD0_e2SMnyQzMkRM1I&cuid=2622654)  sudo dpkg -i cuda-repo-ubuntu1604\_8.0.61-1\_amd64.deb  sudo apt-get update  sudo apt-get install -y cuda  echo -e "\nexport PATH=/usr/local/cuda/bin:$PATH\n\nexport LD\_LIBRARY\_PATH=/usr/local/cuda/lib64" >> .bashrc  sudo reboot  sudo apt-get remove nvidia-375  sudo apt-get install nvidia-367  cd NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery  make  ./deviceQuery |

The above steps also installs Theano and Lasagne packages on the server.

# Theano/Lasagne Configuration

Theano configuration parameters are present in the .theanorc file present in the **HOME directory**.

In this step, **~/.theanorc** is modified with the following parameters. In addition **~/.bashrc** is modified to include the lines shown below.

|  |
| --- |
| **ubuntu@ip-172-31-8-76:~/dev$ cat ~/.theanorc**  [global]  floatX = float64  device = gpu  [lib]  cnmem = 1  [cuda]  root = /usr/local/cuda-8.0  **ubuntu@ip-172-31-8-76:~/dev$ tail -2 ~/.bashrc (last 2 lines of ~/.bashrc file)**  export CUDA\_ROOT=/usr/local/cuda/bin  export THEANO\_FLAGS='cuda.root=/usr/local/cuda/bin,device=gpu,floatX=float32' |

# Adding cuDNN support

The NVIDIA CUDA Deep Neural Network Library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks (sic - <https://developer.nvidia.com/cudnn>)

Follow the instructions below to install cuDNN for Ubuntu 16.04

<http://askubuntu.com/questions/799184/how-can-i-install-cuda-on-ubuntu-16-04>

Listed below are instructions used to get cuDNN support added. **Use the above link to Download the appropriate cuDNN image.**

|  |
| --- |
| tar xvzf cudnn-8.0-linux-x64-v5.1.tgz sudo cp cuda/include/cudnn.h /usr/local/cuda/include sudo cp cuda/lib64/libcudnn\* /usr/local/cuda/lib64 sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn\* |

# Jupyter Notebook configuration

Jupyter is a web-based interactive python development environment. It is installed as part of the Anaconda installation.

The link below contains installation to install Anaconda.

<http://jupyter.readthedocs.io/en/latest/install.html>

Jupyter is executed via the below command

jupyter notebook --ip='\*'

The reason for “--ip” parameter is to ensure that Jupyter binds to the network interface as opposed. The above command provides a token that you will need to use when accessing Jupyter Notebook remotely.

http://<public ip address>:8888

# Monitoring GPU usage

## Using nvidia-smi

GPU support can also be verified using the nvidia-smi command. The output of nvidia-smi shows any active processes currently using the GPU and the GPU processor and memory utilization.

|  |
| --- |
| ubuntu@ip-172-31-8-76:/mnt/dev/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ nvidia-smi  Wed Mar 22 05:56:00 2017  +-----------------------------------------------------------------------------+  | NVIDIA-SMI 367.57 Driver Version: 367.57 |  |-------------------------------+----------------------+----------------------+  | GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |  | Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |  |===============================+======================+======================|  | 0 GRID K520 Off | 0000:00:03.0 Off | N/A |  | N/A 43C P8 17W / 125W | 3922MiB / 4036MiB | 0% Default |  +-------------------------------+----------------------+----------------------+  +-----------------------------------------------------------------------------+  | Processes: GPU Memory |  | GPU PID Type Process name Usage |  |=============================================================================|  | 0 25380 C /mnt/anaconda2/bin/python 3920MiB |  +-----------------------------------------------------------------------------+ |

## Using glances

A command that we found very useful was **glances.** It can be installed using the below command

sudo apt-get install glances

The output of glances is as follows. Interesting sections below are in bold.

|  |
| --- |
| p-172-31-8-76 (Ubuntu 16.04 64bit / Linux 4.4.0-67-generic) Uptime: 1 day, 2:36:10  CPU [ 0.4%] CPU 0.4% **GPU GRID K520** MEM 9.9% SWAP 0.0% LOAD 8-core  MEM [ 9.9%] user: 0.3% **proc: 0%**  total: 14.7G total: 0 1 min: 0.00  SWAP [ 0%] system: 0.2% **mem: 97%**  used: 1.45G used: 0 5 min: 0.02  idle: 99.6% free: 13.2G free: 0 15 min: 0.16  NETWORK Rx/s Tx/s TASKS 181 (236 thr), 1 run, 180 slp, 0 oth sorted automatically by cpu\_percent, flat view  eth0 400b 2Kb  lo 256b 256b CPU% MEM% VIRT RES PID USER NI S TIME+ R/s W/s Command  3.5 0.2 387M 29.7M 29614 root 0 R 1:46.51 0 0 /usr/bin/python /usr/lo  DISK I/O R/s W/s 0.3 0.0 24.2M 3.82M 25212 ubuntu 0 S 0:21.18 0 0 htop  xvda1 0 0 0.0 0.0 0 0 37 root 0 S 0:00.18 0 0 migration/6  xvdb 0 0 0.0 0.0 21.0M 5.19M 25610 ubuntu 0 S 0:00.40 0 0 -bash  0.0 0.7 138M 98.6M 479 root 0 S 9:53.25 0 0 /lib/systemd/systemd-ud  FILE SYS Used Total 0.0 0.0 0 0 25 root -20 S 0:00.00 0 0 kworker/3:0H  / (xvda1) 6.86G 7.74G 0.0 0.0 0 0 177 root -20 S 0:00.00 0 0 kpsmoused  /mnt 3.73G 63.8G 0.0 0.0 0 0 164 root -20 S 0:00.00 0 0 bioset  0.0 0.0 0 0 45 root -20 S 0:00.00 0 0 kworker/7:0H  0.0 0.0 43.7M 344K 6691 avahi 0 S 0:00.00 0 0 avahi-daemon: chroot he  0.0 0.0 61.9M 1.96M 6954 lightdm 0 S 0:00.00 0 0 (sd-pam) |

# Validating GPU support

This section captures steps to validate that GPU works.

NVIDIA provides several tools to validate GPU configuration works.

## ./deviceQuery Check

The first step is already captured in the initial set up of instructions and is part of NVIDIA CUDA samples.

The steps to execute the ./deviceQuery check are as follows

cd NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery

make

./deviceQuery

The resulting output of the ./deviceQuery command will be as follows

|  |
| --- |
| ubuntu@ip-172-31-8-76:/mnt/dev/NVIDIA\_CUDA-8.0\_Samples/1\_Utilities/deviceQuery$ ./deviceQuery  ./deviceQuery Starting...  CUDA Device Query (Runtime API) version (CUDART static linking)  Detected 1 CUDA Capable device(s)  Device 0: "GRID K520"  CUDA Driver Version / Runtime Version 8.0 / 8.0  CUDA Capability Major/Minor version number: 3.0  Total amount of global memory: 4036 MBytes (4232052736 bytes)  ( 8) Multiprocessors, (192) CUDA Cores/MP: 1536 CUDA Cores  GPU Max Clock rate: 797 MHz (0.80 GHz)  Memory Clock rate: 2500 Mhz  Memory Bus Width: 256-bit  L2 Cache Size: 524288 bytes  Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers  Total amount of constant memory: 65536 bytes  Total amount of shared memory per block: 49152 bytes  Total number of registers available per block: 65536  Warp size: 32  Maximum number of threads per multiprocessor: 2048  Maximum number of threads per block: 1024  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)  Maximum memory pitch: 2147483647 bytes  Texture alignment: 512 bytes  Concurrent copy and kernel execution: Yes with 2 copy engine(s)  Run time limit on kernels: No  Integrated GPU sharing Host Memory: No  Support host page-locked memory mapping: Yes  Alignment requirement for Surfaces: Yes  Device has ECC support: Disabled  Device supports Unified Addressing (UVA): Yes  Device PCI Domain ID / Bus ID / location ID: 0 / 0 / 3  Compute Mode:  < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >  deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GRID K520  Result = PASS |

# Test Drive GPU

The program gputest.py is a simple program to test GPU usage. It can be downloaded at

<https://github.com/tajo/deeplearning/blob/master/libs/gputest.py>

The command can be executed using the below commands and the output looks as follows

|  |
| --- |
| ubuntu@ip-172-31-8-76:/mnt/dev/gputest$ python gputest.py  Using gpu device 0: GRID K520 (CNMeM is disabled, cuDNN 5005)  [GpuElemwise{exp,no\_inplace}(<CudaNdarrayType(float32, vector)>), HostFromGpu(GpuElemwise{exp,no\_inplace}.0)]  Looping 1000 times took 0.837495 seconds  Result is [ 1.23178029 1.61879349 1.52278066 ..., 2.20771813 2.29967761  1.62323296]  Used the gpu  ubuntu@ip-172-31-8-76:/mnt/dev/gputest$ |

# Basic Troubleshooting

## NVIDIA Driver Issues

During the course of working, for some unknown reason, GPU driver support became unavailable. It turned out that somehow an incompatible driver was activated.

This can be determined by looking Linux kernel using the **dmesg** command. The tail end of the dmesg command displays information on the GPU driver loaded.

|  |
| --- |
| [ 1908.123010] vgaarb: device changed decodes: PCI:0000:00:03.0,olddecodes=io+mem,decodes=none:owns=io+mem  [ 1908.123142] nvidia-nvlink: Nvlink Core is being initialized, major device number 247  [ 1908.123155] NVRM: loading NVIDIA UNIX x86\_64 Kernel Module 367.27 Thu Jun 9 18:53:27 PDT 2016  [ 1908.124626] nvidia-modeset: Loading NVIDIA Kernel Mode Setting Driver for UNIX platforms 367.27 Thu Jun 9 18:24:10 PDT 2016  [ 1908.125820] [drm] [nvidia-drm] [GPU ID 0x00000003] Loading driver  [ 1957.429230] nvidia-uvm: Loaded the UVM driver in 8 mode, major device number 246  [ 2311.224180] nvidia-uvm: Unloaded the UVM driver in 8 mode  [ 2311.251841] [drm] [nvidia-drm] [GPU ID 0x00000003] Unloading driver  [ 2311.283850] nvidia-modeset: Unloading  [ 2311.307844] nvidia-nvlink: Unregistered the Nvlink Core, major device number 247  [ 2383.322312] vgaarb: device changed decodes: PCI:0000:00:03.0,olddecodes=none,decodes=none:owns=io+mem  [ 2383.322425] nvidia-nvlink: Nvlink Core is being initialized, major device number 247  **[ 2383.322438] NVRM: loading NVIDIA UNIX x86\_64 Kernel Module 367.57 Mon Oct 3 20:37:01 PDT 2016**  [ 2383.323744] nvidia-modeset: Loading NVIDIA Kernel Mode Setting Driver for UNIX platforms 367.57 Mon Oct 3 20:32:57 PDT 2016  [ 2383.324965] [drm] [nvidia-drm] [GPU ID 0x00000003] Loading driver  [ 2406.301815] nvidia-uvm: Loaded the UVM driver in 8 mode, major device number 246 |

In our case, we had noticed that the 375 driver was loaded and had to ununstalled nvidia-375 driver and reinstall the nvidia-367.57 driver whcih was the approriate driver for our GPU.

## MNIST data access issues

Over the past few weeks (current date: March 20th 2017), Yann Le Cunn’s website has been unreachable and therefore the MNIST database could not be retrieved. In order to overcome this issue, the MNIST data was downloaded from a different mirror and then copied over to the sklearn cache using the below instructions.

|  |
| --- |
| cd ~/scikit\_learn\_data/  wget <https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat> |