SEBASTIAN RASCHKA

Introduction to Artificial Neural Networks and Deep Learning

with Applications in Python

Introduction to Artificial Neural Networks

with Applications in Python

Sebastian Raschka

DRAFT

Last updated: May 31, 2018

This book will be available at http://leanpub.com/ann-and-deeplearning.

Please visit https://github.com/rasbt/deep-learning-book for more information, supporting material, and code examples.

Contents

Website

Please visit the GitHub repository to download the code examples accompanying this book and other supplementary material.

If you like the content, please consider supporting the work by buying a copy of the book on Leanpub. Also, I would appreciate hearing your opinion and feedback about the book, and if you have any questions about the contents, please don't hesitate to get in touch with me via mail@sebastianraschka.com. Happy learning!

Sebastian Raschka

About the Author

Sebastian Raschka received his doctorate from Michigan State University developing novel computational methods in the field of computational biology. In summer 2018, he joined the University of Wisconsin–Madison as Assistant Professor of Statistics. Among others, his research activities include the development of new deep learning architectures to solve problems in the field of biometrics. Among his other works is his book "Python Machine Learning," a bestselling title at Packt and on Amazon.com, which received the ACM Best of Computing award in 2016 and was translated into many different languages, including German, Korean, Italian, traditional Chinese, simplified Chinese, Russian, Polish, and Japanese.

Sebastian is also an avid open-source contributor and likes to contribute to the scientific Python ecosystem in his free-time. If you like to find more about what Sebastian is currently up to or like to get in touch, you can find his personal website at https://sebastianraschka.com.

Acknowledgements

I would like to give my special thanks to the readers, who provided feedback, caught various typos and errors, and offered suggestions for clarifying my writing.

- Appendix A: Artem Sobolev, Ryan Sun
- Appendix B: Brett Miller, Ryan Sun
- Appendix D: Marcel Blattner, Ignacio Campabadal, Ryan Sun, Denis Parra Santander
- Appendix F: Guillermo Monecchi, Ged Ridgway, Ryan Sun, Patric Hindenberger
- Appendix H: Brett Miller, Ryan Sun, Nicolas Palopoli, Kevin Zakka

Appendix H

Cloud Computing

In machine learning and deep learning applications, we are working with large matrix multiplications that can be parallelized. So taking advantage of GPU architectures with thousands of cores usually makes sense if we want to train neural network models efficiently.

If you do not have an NVIDIA GPU – as of this writing, the GPU versions of TensorFlow and PyTorch only run via CUDA and cuDNN[1] – you will still be able to run all the code in this book on your machine via CPU, but it may execute slower. Again, using a dedicated GPU to run the code examples in this book is by no means required, but running your code on a GPU will likely become necessary when you are tackling real-world problems – if you want to train neural networks in a timely manner.

In this appendix, you are going to learn how we can use Amazon Web Services (AWS), an on-demand computing platform, for deep learning on GPU computing instances. This tutorial is meant for people who do not have a GPU and want to use Amazon's cloud services to get a GPU instance up and running. However, keep in mind that AWS is *not* free, but in my experience, it is a convenient and affordable service (usually less than a dollar per hour for "small" computing instances). I have no affiliation to AWS though, but I found it a fairly-priced, reliable, and fast platform for training deep neural networks[2].

[1]There is now experimental support for OpenCL.

[2]For example, Microsoft and Google are now offering similar cloud computing services.

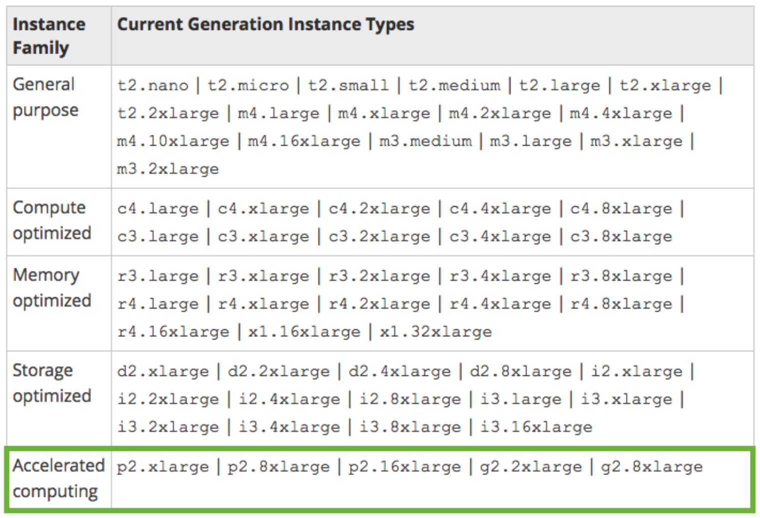
H.1 Setting Up an AWS Account

If you do not have an AWS account, yet, go to https://aws.amazon.com/ and click on the button "Create an AWS Account" in the top right corner. If you are already an Amazon customer, just provide your login credentials in the respective form fields; otherwise, select the radio button "I am a new user, " and follow the sign-up instructions to create a new account.

H.2 About AWS GPU Instances

Assuming that you have successfully created an AWS account and signed in, let us launch a new AWS instance. Since we are mainly interested in machine learning and deep learning applications, we want to take advantage of GPUs that are extremely efficient for parallelized computations. So, let us check out the list of current GPU instances Amazon's Elastic Compute Cloud (EC2) provides by visiting the documentation page at http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html.

Scrolling down to the section "Current Generation Instances," we find a table that list all the available instances; for deep learning, we are mostly interested in GPU instances, such as `g2.2xlarge`, `g2.8xlarge`, `p2.xlarge`, `p2.8xlarge`, and `p2.16xlarge` (Figure H.1).



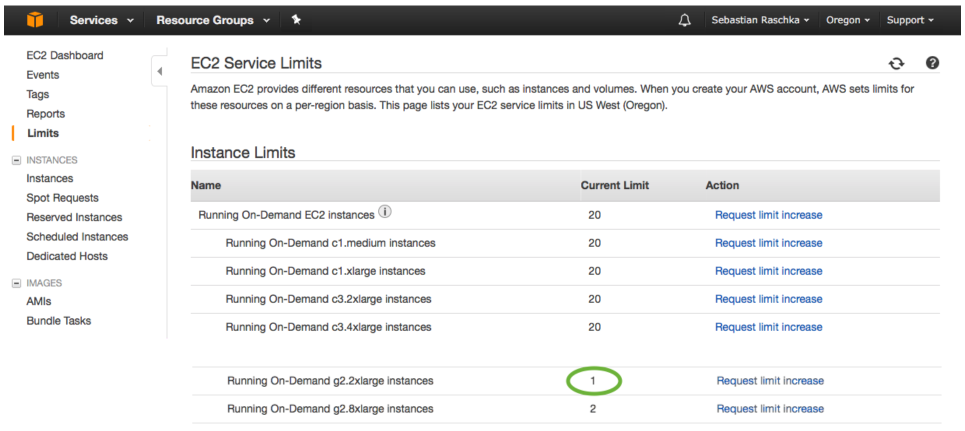
Instance Family	Current Generation Instance Types
General purpose	t2.nano | t2.micro | t2.small | t2.medium | t2.large | t2.xlarge | t2.2xlarge | m4.large | m4.xlarge | m4.2xlarge | m4.4xlarge | m4.10xlarge | m4.16xlarge | m3.medium | m3.large | m3.xlarge | m3.2xlarge
Compute optimized	c4.large | c4.xlarge | c4.2xlarge | c4.4xlarge | c4.8xlarge | c3.large | c3.xlarge | c3.2xlarge | c3.4xlarge | c3.8xlarge
Memory optimized	r3.large | r3.xlarge | r3.2xlarge | r3.4xlarge | r3.8xlarge | r4.large | r4.xlarge | r4.2xlarge | r4.4xlarge | r4.8xlarge | r4.16xlarge | x1.16xlarge | x1.32xlarge
Storage optimized	d2.xlarge | d2.2xlarge | d2.4xlarge | d2.8xlarge | i2.xlarge | i2.2xlarge | i2.4xlarge | i2.8xlarge | i3.large | i3.xlarge | i3.2xlarge | i3.4xlarge | i3.8xlarge | i3.16xlarge
Accelerated computing	p2.xlarge | p2.8xlarge | p2.16xlarge | g2.2xlarge | g2.8xlarge

Figure H.1: GPU instances on AWS.

For this tutorial, we are going to choose the smallest GPU instance type, `g2.2xlarge` which are backed by an NVIDIA GRID GPU and eight hardware hyperthreads from an Intel Xeon CPU as of this writing. The `g2.2xlarge` instance is also the cheapest GPU instance type, which suffices to run the code in this book relatively efficiently. However, please feel free to choose one of the larger GPU instances instead – just be aware that those instances are a bit more expensive.

After we have settled on an instance type that suits our needs, we need to check if we can launch them. So, let us check "EC2 Service Limit report" by heading to https://console.aws.amazon.com/ec2/v2/home?#Limits.

Scroll down the list entry for the `g2.2xlarge` instances (Figure H.2). If you see a "0" in the "Current Limit" column, you need to request a limit increase by clicking the respective hyperlink in the right column and following the instructions. Please note that it may take 1-2 days until Amazon approves your limit increase.



EC2 Service Limits

Amazon EC2 provides different resources that you can use, such as instances and volumes. When you create your AWS account, AWS sets limits for these resources on a per-region basis. This page lists your EC2 service limits in US West (Oregon).

Instance Limits

Name	Current Limit	Action
Running On-Demand EC2 instances	20	Request limit increase
Running On-Demand c1.medium instances	20	Request limit increase
Running On-Demand c1.xlarge instances	20	Request limit increase
Running On-Demand c3.2xlarge instances	20	Request limit increase
Running On-Demand c3.4xlarge instances	20	Request limit increase
Running On-Demand g2.2xlarge instances	1	Request limit increase
Running On-Demand g2.8xlarge instances	2	Request limit increase

Figure H.2: Instance Limits.

H.3 Launching an AWS GPU Instance

After we checked that `g2.2xlarge` instances are available to use, we can proceed to launch our first EC2 instance. Head over to your AWS dashboard and select "EC2" from the panel (Figure H.3).

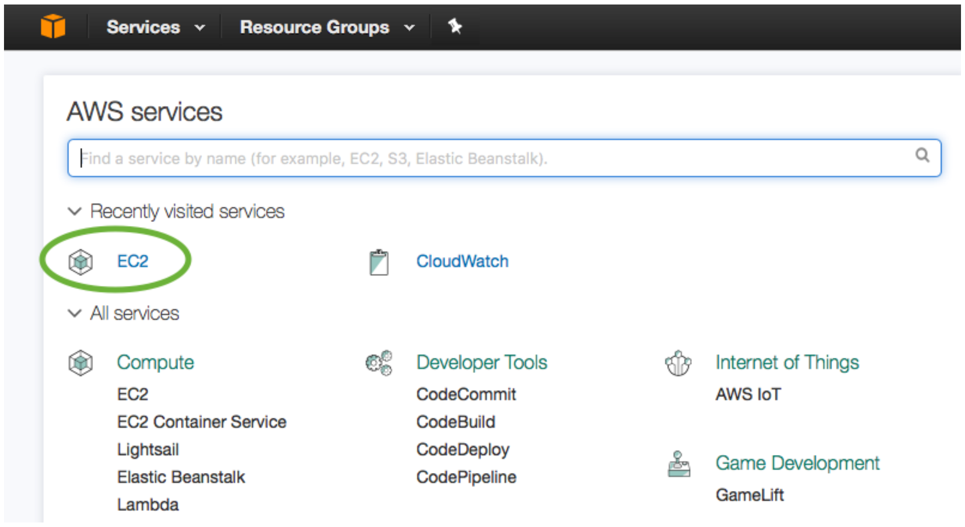
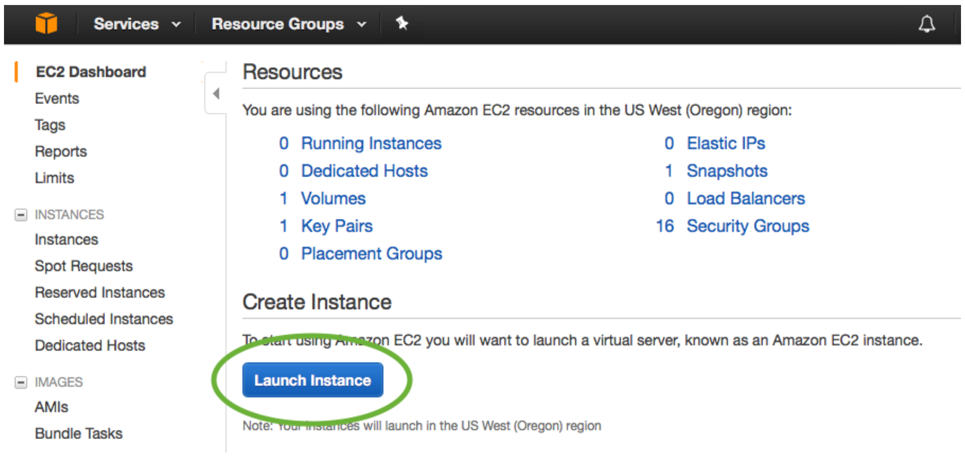


Figure H.3: Selecting the EC2 Panel.

After clicking the link shown in the figure above, you should see the EC2 dashboard. Here, just click the button that reads "Launch Instance" (Figure H.4).



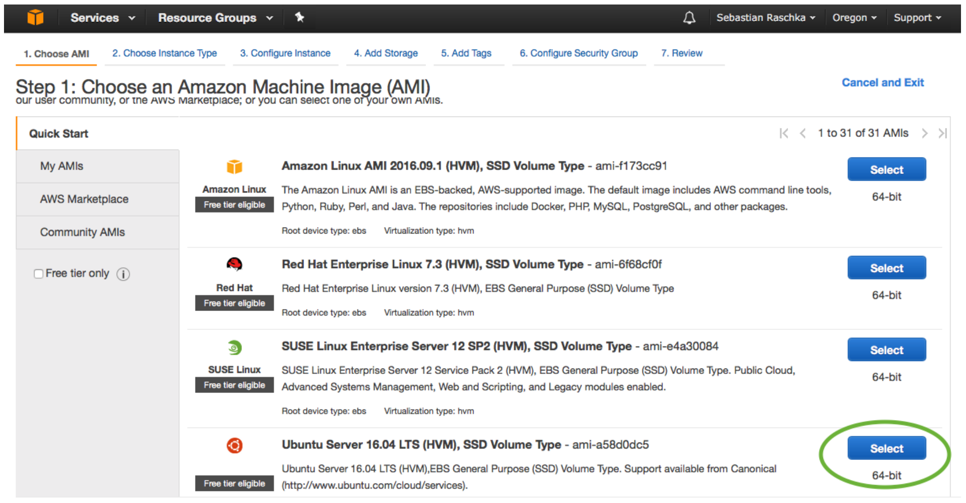
You are using the following Amazon EC2 resources in the US West (Oregon) region:

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US West (Oregon) region

Figure H.4: Launching an EC2 instance.

Next, we are going to select an Amazon Machine Image (AMI) that comes with a pre-installed Linux distribution to save us some hassle setting it up from scratch. You can choose whatever distribution you are most comfortable with, but the rest of this tutorial is based on "Ubuntu Server 16.04 LTS" (Figure H.5).



Step 1: Choose an Amazon Machine Image (AMI)

our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Amazon Linux — Free tier eligible
Amazon Linux AMI 2016.09.1 (HVM), SSD Volume Type - ami-f173cc91
The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm

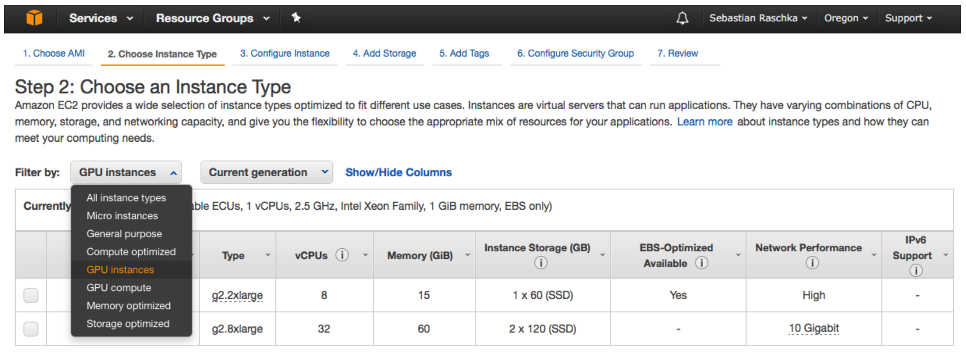
Red Hat — Free tier eligible
Red Hat Enterprise Linux 7.3 (HVM), SSD Volume Type - ami-6f68cf0f
Red Hat Enterprise Linux version 7.3 (HVM), EBS General Purpose (SSD) Volume Type
Root device type: ebs Virtualization type: hvm

SUSE Linux — Free tier eligible
SUSE Linux Enterprise Server 12 SP2 (HVM), SSD Volume Type - ami-e4a30084
SUSE Linux Enterprise Server 12 Service Pack 2 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs Virtualization type: hvm

Free tier eligible
Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-a58d0dc5
Ubuntu Server 16.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Figure H.5: Selecting a Ubuntu AMI.

In the next window, select "GPU instances" in the "Filter by" drop-down menu to select "GPU instances;" then, select `g2.2xlarge` (Figure H.6).



Step 2: Choose an Instance Type

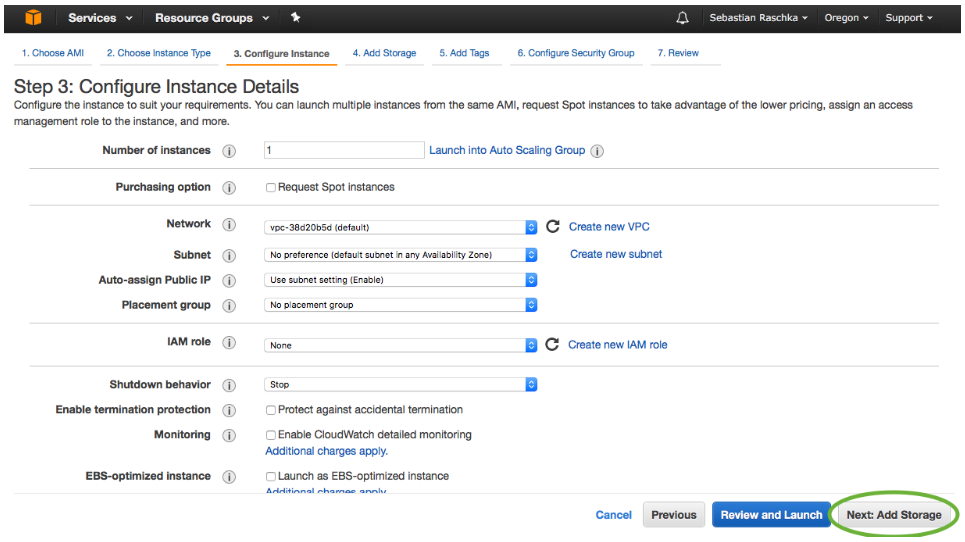
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Currently ...ble ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
	g2.2xlarge	8	15	1 x 60 (SSD)	Yes	High	-
	g2.8xlarge	32	60	2 x 120 (SSD)	-	10 Gigabit	-

Figure H.6: Filtering AMIs by type.

Then, click on the button "Next: Configure Instance Details" in the lower right. In the following menu, "3. Configure Instance," you can leave the default settings unchanged and click on "Next: Add Storage" in the lower right corner (Figure H.7).



Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1
Purchasing option	☐ Request Spot instances
Network	vpc-38d20b5d (default)
Subnet	No preference (default subnet in any Availability Zone)
Auto-assign Public IP	Use subnet setting (Enable)
Placement group	No placement group
IAM role	None
Shutdown behavior	Stop
Enable termination protection	☐ Protect against accidental termination
Monitoring	☐ Enable CloudWatch detailed monitoring
EBS-optimized instance	☐ Launch as EBS-optimized instance

Figure H.7: Adding instance storage.

To set up our first instance, 16 Gb[3] of storage should do just fine; enter the value in the corresponding form field.

Unless you want to provide custom tags, you can head directly to the section "6. Configure Security Group" via the panel at the top of this page. Once you are on the "Configure Security Group" page, I recommend selecting "My IP" from the drop-down menu, unless you want other computers to be able to log into your instance (Figure H.8). Be aware though that you may run into troubles logging in to the AWS instance with this setting if your local machine's IP gets reassigned dynamically, for example, by rebooting your system. However, keep in mind that you can always change the security group settings later on when your instance is running.

[3]We will save an image of this instance for re-use later, which allows you to instantiate an instance with more storage if needed.

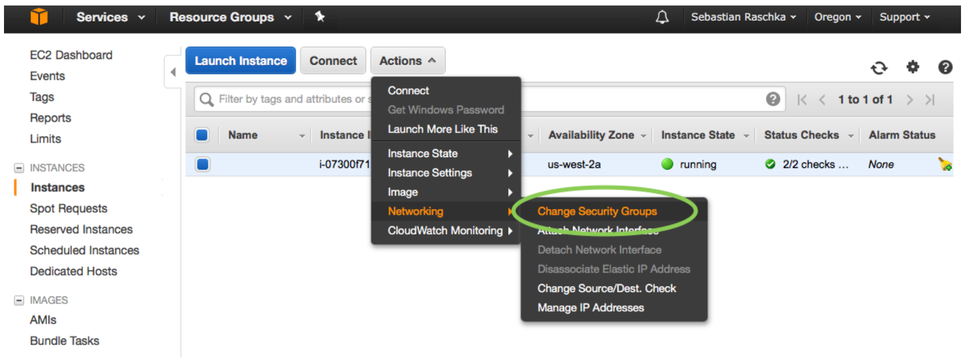
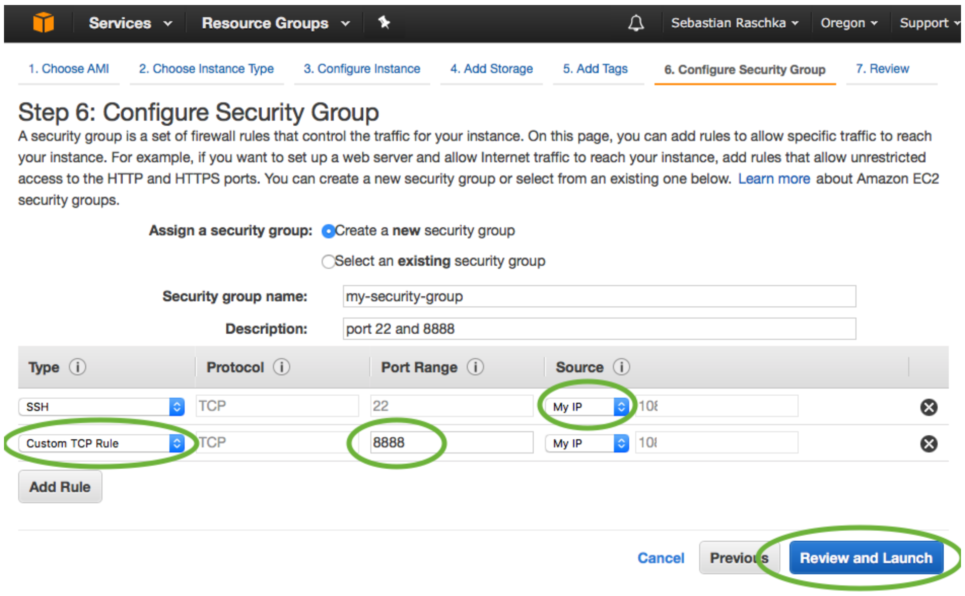


Figure H.8: Accessing security settings.

If you'd like to launch Jupyter notebooks in your instance later and be able to access them from your local machine, click on "Add Rule," select "Custom TCP Rule" from the drop-down menu, and type `8888` into the "Port Range" form field. Then, click the "Review and Launch" button (Figure H.9).



Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: [✓ Create a **new** security group] Select an **existing** security group

Security group name: my-security-group

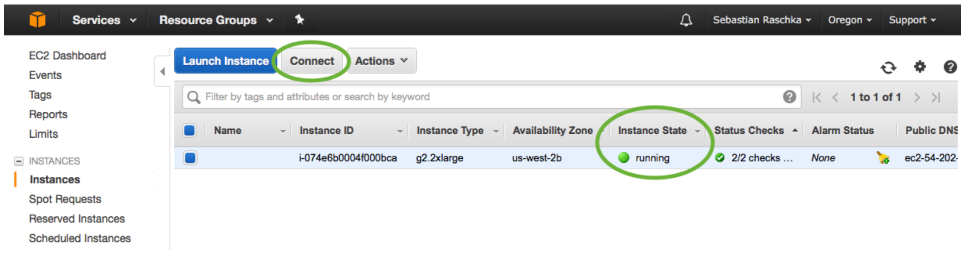
Description: port 22 and 8888

Type	Protocol	Port Range	Source
SSH	TCP	22	My IP 10
Custom TCP Rule	TCP	8888	My IP 10

Figure H.9: Configuring security settings.

Next, a pop-up window will prompt you to select or create a public "key pair." If you already have a key pair, select it from the drop-down menu; otherwise, select "Create a new key pair" and provide a key pair name and click the button "Download Key Pair." After you downloaded the `.pem` key file to your local machine, keep it someplace safe! Then proceed by clicking on "Launch Instances."

It usually takes a few moments after launching an instance until you can connect to it. Go to "Instances" dashboard from the left panel and check if your instance is online, yet (Figure H.10).



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
	i-074e6b0004f000bca	g2.2xlarge	us-west-2b	running	2/2 checks ...	None	ec2-54-202-

Figure H.10: Connecting to a running instance.

If your instance status shows "running," you can click on the "connect" button as shown in Figure H.10, and a new menu with login instructions will appear. Here, look out for the example section that lists a command similar to the following:

```
ssh -i "aws-key.pem"
ubuntu@ec2-XX-XXX-XXX-XX.us-west-2.compute.amazonaws.com
```

Copy this command to your command line terminal, and make sure that you provide the correct path to your `.pem` file (you may need to change `aws-key.pem` to `/path/to/your/aws-key/your.pem`). After executing the previous command, you should be logged in to the AWS instance.

H.4 Installing Basic Packages, Conda, and Python

Now that you are logged in to an AWS GPU instance, you can install Continuum Analytic's Miniconda Python distribution and the scientific Python packages we will be mainly working with. But first, it is recommended update Ubuntu's package installer and the already pre-installed packages by executing the following two commands:

```
sudo apt-get update
sudo apt-get upgrade
```

Also, we will need some additional packages for installing the NVIDIA CUDA and cuDNN libraries later, which are required for running TensorFlow or PyTorch via a GPU. Execute the following command to install those:

```
sudo apt-get install -y build-essential \
g++ gfortran git libfreetype6-dev libxft-dev \
libncurses-dev libopenblas-dev libblas-dev \
liblapack-dev libatlas-base-dev \
linux-headers-generic linux-image-extra-virtual \
zlib1g-dev libcurl3-dev
```

Next, we are going to install the Miniconda Python distribution. First, download it via:

```
wget https://repo.continuum.io/miniconda/\
Miniconda3-latest-Linux-x86_64.sh
```

After the download finished, invoke the installer by executing

```
bash Miniconda3-latest-Linux-x86_64.sh
```

and follow the instructions and use the default settings; however, if the installer asks you `to prepend the Miniconda3 install location to PATH in your /home/ubuntu/.bashrc`, type `yes`. If you accidentally typed `no` you would want to edit your `/.bashrc` file and append the line `export PATH="/home/ubuntu/miniconda3/bin:$PATH"`, to ensure that conda-Python will be the default Python when you log into the instance.

Now that we have successfully installed Miniconda, let us `source` our current configuration by executing

```
source ~/.bashrc
```

To check that the installation was indeed successful, execute `which python`, which should print the following output to your terminal:

```
/home/ubuntu/miniconda3/bin/python
```

Finally, we can use the `conda` package manager to install our favorite packages for scientific computing in Python via:

```
conda install numpy scipy
```

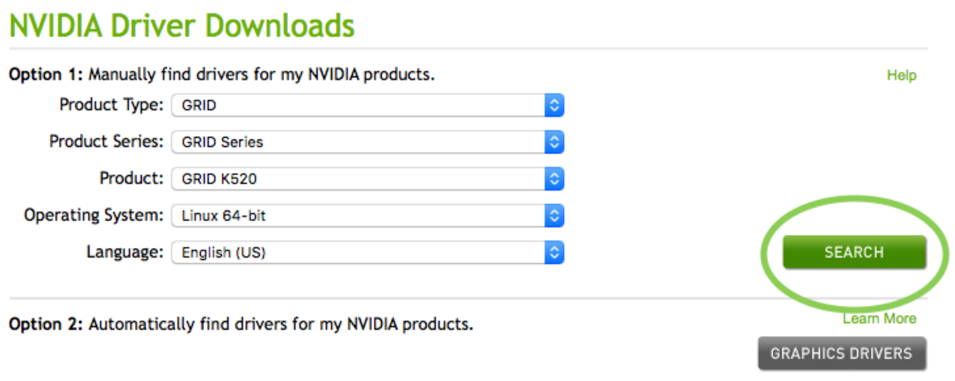
H.5 Installing a Recent NVIDIA Driver

In this section, we are going to install the latest NVIDIA drivers for the graphics card in the AWS EC2 GPU instance. Ubuntu already comes with a non-proprietary driver, `nvidiafb`, and a reverse-engineered, open-source version of the NVIDIA driver called `nouveau`. However, I found that these two are not playing nicely with NVIDIA's CUDA and cuDNN libraries, which we require for PyTorch or TensorFlow. Note if we are installing PyTorch via its recommended binary installers, we do not need to install CUDA and cuDNN separately as the binaries come with the necessary library files. However, installing an NVIDIA display driver is still required. Luckily, installing NVIDIA drivers is much simpler than it used to be a few years ago.

Before we start, let us make sure that graphics cards are available in the current EC2 instance. Execute the command `lspci -nnk | grep -i nvidia` and you should see an output similar to the following:

```
00:03.0 VGA compatible controller [0300]:
    NVIDIA Corporation GK104GL [GRID K520]
    [10de:118a] (rev a1)
        Subsystem: NVIDIA Corporation GK104GL
                   [GRID K520] [10de:1014]
        Kernel modules: nvidiafb, nouveau
```

Next, we need to visit http://www.nvidia.com/Download/index.aspx and select the instance's graphic card from the dropdown menus. Based on the output shown in previous code example, the graphics card would be `GRID K520` (Figure H.11).


NVIDIA Driver Downloads

Option 1: Manually find drivers for my NVIDIA products.

Product Type: GRID
Product Series: GRID Series
Product: GRID K520
Operating System: Linux 64-bit
Language: English (US)

Option 2: Automatically find drivers for my NVIDIA products.

Figure H.11: Selecting and downloading NVIDIA drivers.

After clicking on the "Search" button on that page (as shown in Figure H.11), click on "Download" and copy the link from "Agree & Download" button if you agree with the "License For Customer Use of NVIDIA Software." Then, copy the download link into the command line terminal that connects to your AWS instance and download the driver *runfile* via `wget`, for example,

```
wget http://us.download.nvidia.com/XFree86/Linux-x86_64/\
367.57/NVIDIA-Linux-x86_64-367.57.run
```

Infobox H.5.1 Use Up to Date Links

Please note that links to specific NVIDIA driver versions as well as the links to specific TensorFlow, PyTorch, cuDNN, and CUDA versions are likely not up to date when you are reading this. Thus, I generally recommend you to visit the actual download pages and select the most up to date versions, which usually contain bug fixes and other improvements.

Then, start the installer by executing the following command:

```
sudo bash NVIDIA-Linux-x86_64-367.57.run
```

and follow the on-screen instructions. After the installation has completed, execute `lspci -nnk | grep -i nvidia` again, which should now show `nvidia` as the kernel driver:

```
00:03.0 VGA compatible controller [0300]:
   NVIDIA Corporation GK104GL
   [GRID K520] [10de:118a] (rev a1)
        Subsystem: NVIDIA Corporation GK104GL [GRID K520] [10de:1014]
        Kernel driver in use: nvidia
        Kernel modules: nvidiafb, nouveau, nvidia_drm, nvidia
```

To prevent that `nouveau` is causing conflicts if we need to reboot our instance, it is recommended to blacklist it. First, create a new blacklist file by executing the following command:

```
sudo vi /etc/modprobe.d/blacklist-nouveau.conf
```

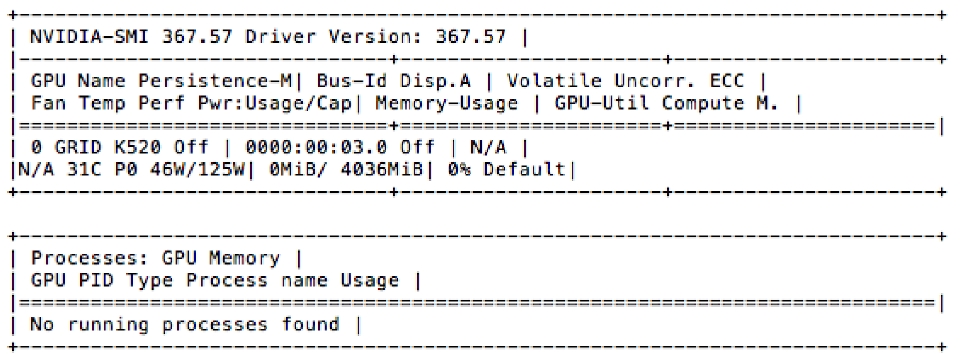
Then, add the following two lines to this blacklist file:

```
blacklist nouveau
options nouveau modeset=0
```

Finally, we need to update the "inital RAM file system" by executing the following:

```
sudo update-initramfs -u
```

You should also be able to run the command `nvidia-smi` now, which provides you with more detailed information about the EC2 graphics card (Figure H.12).



```
+-----------------------------------------------------------------------------+
| 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 31C P0 46W/125W| 0MiB/ 4036MiB| 0% Default|
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes: GPU Memory |
| GPU PID Type Process name Usage |
|=============================================================================|
| No running processes found |
+-----------------------------------------------------------------------------+
```

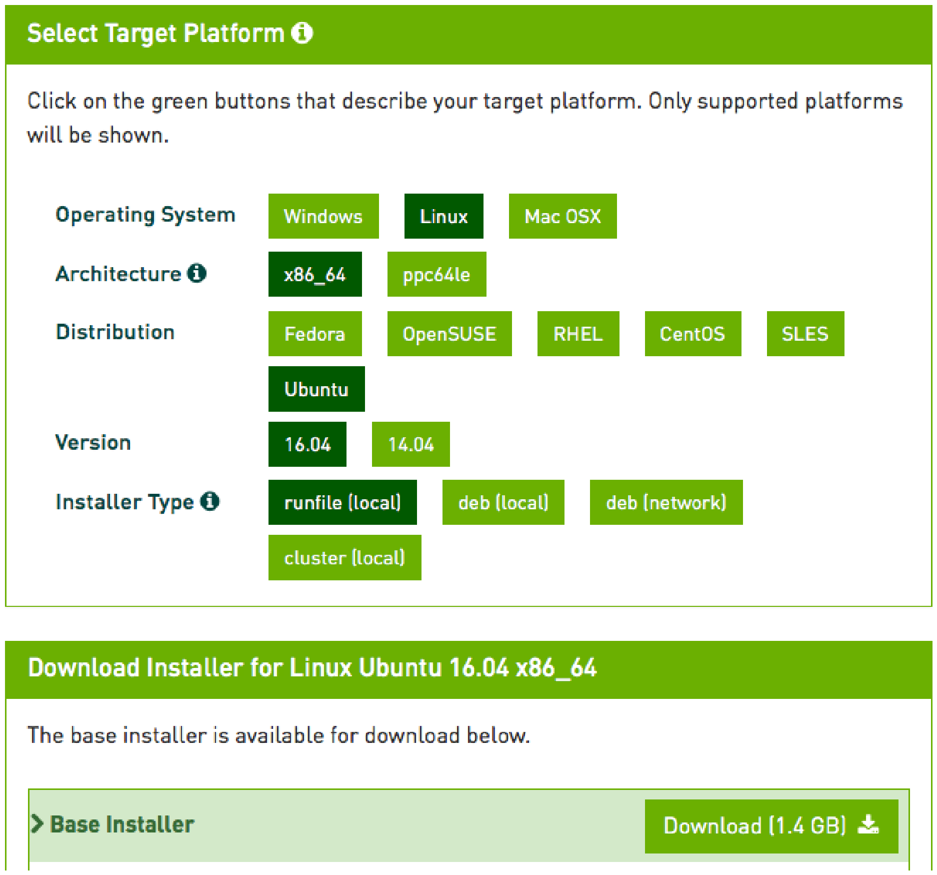
Figure H.12: Example `nvidia-smi` output.

H.6 Installing CUDA

Infobox H.6.1 PyTorch and CUDA/cuDNN

If you are planning to use PyTorch and not TensorFlow, you can skip this section, since PyTorch, unless we want to compile it from source, comes pre-packed with all the necessary CUDA/cuDNN files via the `conda` or `pip` binary installers as explained in Section H.9.

To install CUDA, the parallel computing platform and application programming interface (API) model by NVIDIA, visit the website `https://developer.nvidia.com/cuda-downloads`. On this website, select "Linux," click on "Architecture x86_64," select "Distribution Ubuntu," and finally click on "16.04." Then, copy the download link provided via the "Installer Type" button that says "runfile (local)" on it (Figure H.13).



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	[✓ 16.04] 14.04
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.

Figure H.13: Example `nvidia-smi` output.

Infobox H.6.2 Installing CUDA via the "deb (local)" File

Your mileage may vary, but I do not recommend using the "deb (local)" file since it comes packaged with NVIDIA drivers that may override the drivers we installed in the previous section, which can result in incompatible graphics card drivers on certain systems. However, note that while it is generally recommended to use the latest version of CUDA, it is also important to make sure that this version is also supported by the graphics card, TensorFlow, and PyTorch (if you want to install one of these packages later). For more information about the current CUDA support, please see the following websites:

- https://developer.nvidia.com/cuda-gpus
- https://www.tensorflow.org/install/install_linux
- https://pytorch.org

Now, copy the CUDA download link to the terminal window of your AWS instance and put a `wget` command in front of it to download the installer file, for example:

```
wget https://developer.nvidia.com/compute/cuda/8.0/Prod2/\
local_installers/cuda_8.0.61_375.26_linux-run
```

Next, we can launch the CUDA installer by executing the following command:

```
sudo bash cuda_8.0.61_375.26_linux-run
```

When you follow the instructions in your terminal, make sure that you do **not** install the "NVIDIA Accelerated Graphics Driver" (we have done this previously):

```
    Do you accept the previously read EULA?
    accept/decline/quit: accept

    Install NVIDIA Accelerated Graphics Driver
    for Linux-x86_64 375.26?
    (y)es/(n)o/(q)uit: n

    Install the CUDA 8.0 Toolkit?
    (y)es/(n)o/(q)uit: y

    Enter Toolkit Location
     [ default is /usr/local/cuda-8.0 ]:

    Do you want to install a symbolic link at /usr/local/cuda?
    (y)es/(n)o/(q)uit: y

    Install the CUDA 8.0 Samples?
    (y)es/(n)o/(q)uit: n
```

Next, run the command

```
lspci -nnk | grep -i nvidia
```

again and check that is still lists `nvidia` as the "Kernel driver in use:"

```
00:03.0 VGA compatible controller [0300]:
     NVIDIA Corporation GK104GL
     [GRID K520] [10de:118a] (rev a1)
Subsystem: NVIDIA Corporation GK104GL
           [GRID K520] [10de:1014]
Kernel driver in use: nvidia
Kernel modules: nvidiafb, nouveau, nvidia_drm, nvidia
```

After installing CUDA, I recommend to reboot your system (`sudo reboot`). Then, if you log into your AWS EC2 instance again (notice that the API might have changed after the reboot) and execute `lspci -nnk | grep -i nvidia` one more time. If you see any deviation from the output prior to the reboot, please make sure that you blacklisted "nouveau" as described in the previous Section H.5.

H.7 Installing cuDNN

Infobox H.7.1 PyTorch and CUDA/cuDNN

If you are planning to use PyTorch and not TensorFlow, you can skip this section, since PyTorch, unless we want to compile it from source, comes pre-packed with all the necessary CUDA/cuDNN files via the `conda` or `pip` binary installers as explained in Section H.9.

Now that we have installed CUDA, let us move to the next step and install cuDNN, which is a GPU-accelerated library of primitives for deep neural networks. Unfortunately, installing cuDNN is a bit of a hassle and we need to register for an account at https://developer.nvidia.com/cudnn first. Once we have done that, we can head over to https://developer.nvidia.com /rdp/cudnn-download and download the runtime and developer libraries of the most recent version of cuDNN. Note that as of this writing, there are no runtime and developer libraries for Ubuntu 16.04 LTS available via NVIDIA, but the respective installers for Ubuntu 14.04 work just fine:

- cuDNN v5.1 Runtime Library for Ubuntu14.04 (Deb)
- cuDNN v5.1 Developer Library for Ubuntu14.04 (Deb)

Since the download of cuDNN requires a sign in with our developer credentials, we need to download these two to our local machine first. Then, we can upload those download files to our AWS instance via the `rsync` program available in Linux/Unix shells, for example, by executing the following command:

```
rsync -rave "ssh -i aws-key.pem" ~/Downloads/libcudnn* \
ubuntu@ec2-XX-XXX-XXX-XX.us-west-2\
.compute.amazonaws.com:/home/ubuntu
```

Once the files are uploaded to our AWS instance, we can install the cuDNN packages using `dpkg`:

```
sudo dpkg -i libcudnn5_5.1.10-1+cuda8.0_amd64.deb
sudo dpkg -i libcudnn5-dev_5.1.10-1+cuda8.0_amd64.deb
```

Finally, we edit our `~/.bashrc` file on the AWS instance once more and add the following lines (for example, by using `vi ~/.bashrc`):

```
export CUDA_HOME=/usr/local/cuda
export CUDA_ROOT=/usr/local/cuda
export PATH=$PATH:$CUDA_ROOT/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUDA_ROOT/lib64
```

Then we `source` the `.bashrc` file one more time so that these changes take effect in our current login terminal:

```
source ~/.bashrc
```

H.8 Installing TensorFlow

After we installed CUDA and cuDNN, we are all set and ready to install TensorFlow via `pip`, which is as simple as executing the following command on your AWS terminal:

```
pip install tensorflow-gpu
```

Next, let us just double-check that everything was installed okay by invoking a new Python session from the terminal and import TensorFlow and checking the device mapping:

```
ubuntu@ip-XXX-XX-XX-XXX:~$ python

Python 3.6.0 |Continuum Analytics, Inc.|
  (default, Dec 23 2016, 12:22:00)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits"
  or "license" for more information.
```

```
>>> import tensorflow as tf
I tensorflow/stream_executor/dso_loader.cc:135]
  successfully opened CUDA
library libcublas.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135]
  successfully opened CUDA
library libcudnn.so.5 locally
I tensorflow/stream_executor/dso_loader.cc:135]
  successfully opened CUDA
library libcufft.so.8.0 locally
I tensorflow/stream_executor/dso_loader.cc:135]
  successfully opened CUDA
library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:135]
  successfully opened CUDA
library libcurand.so.8.0 locally
>>>
>>>
>>> tf.__version__
'1.0.0'
>>>
>>>
>>> tf.test.gpu_device_name()
I tensorflow/core/common_runtime/gpu/gpu_device.cc:885]
  Found device 0 with
properties:
name: GRID K520
major: 3 minor: 0 memoryClockRate (GHz) 0.797
```

```
pciBusID 0000:00:03.0
Total memory: 3.94GiB
Free memory: 3.91GiB
I tensorflow/core/common_runtime/gpu/
  gpu_device.cc:906] DMA: 0
I tensorflow/core/common_runtime/gpu/
  gpu_device.cc:916] 0:   Y
I tensorflow/core/common_runtime/gpu/
  gpu_device.cc:975] Creating TensorFlow
device (/gpu:0)
-> (device: 0, name: GRID K520, pci bus id: 0000:00:03.0)
Device mapping:
/job:localhost/replica:0/task:0/gpu:0
  -> device: 0, name: GRID K520, pci
bus id: 0000:00:03.0
I tensorflow/core/common_runtime/direct_session.cc:257]
 Device mapping:
/job:localhost/replica:0/task:0/gpu:0
  -> device: 0, name: GRID K520, pci
bus id: 0000:00:03.0
>>> quit()
```

If the output looks similar to the one shown above, you are all set and can run your TensorFlow computations on AWS now!

H.9 Installing PyTorch

Unlike TensorFlow, PyTorch does not come with separate installers for the CPU and GPU versions. By default, PyTorch executes computations on the CPU, and we have to specify for which parts of the code the GPU is going to be used – if a GPU is available.

For example, if we use the `torch.nn.Module` class,

class MyConvNet(torch.nn.Module): ...

a convenient way to specify that the GPU should be used, if it is available, could be as follows:

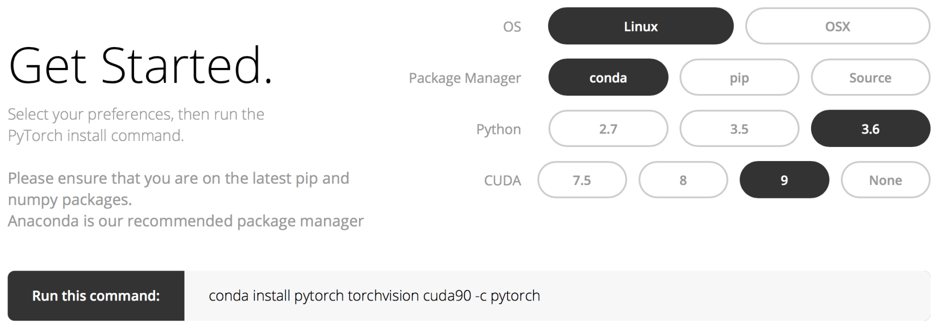
```
device = torch.device("cuda" if
                      torch.cuda.is_available()
                      else "cpu")
model = ConvNet(num_features=num_features,
                num_classes=num_classes)
model.to(device)
```

Then, during a training loop, we can apply the same concept to the input features and target variables:

```
for batch_idx, (features, targets) in enumerate(train_loader):

    features.to(device)
    targets.to(device)
    ...
    predictions = model(features, targets)
    ...
```

Now, to install PyTorch, we just need to head over to the official website (http://pytorch.org) and use the convenient installer selection menu on the front page (Figure H.14).


Get Started.

Select your preferences, then run the PyTorch install command.

Please ensure that you are on the latest pip and numpy packages.
Anaconda is our recommended package manager

OS: [✓ Linux] OSX
Package Manager: [✓ conda] pip Source
Python: 2.7 3.5 [✓ 3.6]
CUDA: 7.5 8 [✓ 9] None

Run this command:

```
conda install pytorch torchvision cuda90 -c pytorch
```

Figure H.14: PyTorch Installer Mendu

Infobox H.9.1 PyTorch and CUDA/cuDNN

If we install PyTorch from the binary packages via `pip` or `conda` (which will be explained later in this section), we do not need to install CUDA or cuDNN separately, since PyTorch comes with all the necessary files pre-packaged. All it requires is having NVIDIA graphics card drivers installed.

Then, we just need to copy the appropriate command shown at the bottom of the installer selection menu into our AWS EC2 command line terminal and execute it. Note that both conda and pip installers work equally smoothly. Finally, we can doublecheck that PyTorch detects the current CUDA installation using `torch.cuda.is_available()`:

```
ubuntu@ip-XXX-XX-XX-XXX:~$ python
Python 3.6.3 |Anaconda, Inc.| (default, Nov 20 2017, 20:41:42)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import torch
>>> torch.cuda.is_available()
True

In addition, we can check which version of CUDA PyTorch is using by execut
the following command in the python interpreter:

>>> torch.version.cuda
'9.0.176'
```

Infobox H.9.2 PyTorch and CUDA Troubleshooting Tip

Sometimes, it may happen that PyTorch may not detect the CUDA version when executing `torch.cuda.is_available()`. Possible reasons are

- the graphics card is incompatible with the CUDA version that came pre-bundled with the PyTorch binary installers
- NVIDIA graphics card driver is incompatible with the CUDA version

To check the graphic card compatibility, visit https://developer.nvidia.com/cuda-gpus. If the graphics card seem to support the CUDA version that you selected via the PyTorch installer menu, I found that in practice, it can happen that the NVIDIA driver causes issues. In this case, a workaround is to download CUDA (see Section **??**) and install the NVIDIA driver from the installer menu.

(Note that we are not installing CUDA, only the NVIDIA display driver.) For instance, use the following settings:

```
Do you accept the previously read EULA?
accept/decline/quit:       accept

Install NVIDIA Accelerated Graphics Driver
for Linux-x86_64 384.81?
(y)es/(n)o/(q)uit: y

Do you want to install the OpenGL libraries?
(y)es/(n)o/(q)uit [ default is yes ]: n

Do you want to run nvidia-xconfig?
This will update the system X configuration
file so that the NVIDIA X
driver
is used. The pre-existing X configuration
file will be backed up.
This option should not be used on systems
that require a custom
X configuration, such as systems
with multiple GPU vendors.
(y)es/(n)o/(q)uit [ default is no ]: y

Install the CUDA 9.0 Toolkit?
(y)es/(n)o/(q)uit: n

Install the CUDA 9.0 Samples?
(y)es/(n)o/(q)uit: n

Installing the NVIDIA display driver...
```

H.10 Accessing Jupyter Notebooks on AWS Instances

This is an optional session in case you are interested in running Jupyter Notebooks on AWS instances and want to be able to access it from your local machine. First, we need to install IPython, which we can do by running the following conda command:

```
conda install ipython
```

Next, we need to start a new IPython session by typing the `ipython` command into the terminal. In the IPython shell, we import `passwd` from `IPython.lib` and call the `passwd` function:

```
In [I]: from IPython.lib import passwd
In [2]: passwd(algorithm='sha512')
```

When it prompts for a password, select a strong password and generate the password hash:

```
Enter password:
Verify password:
Out[2]: 'sha1:4b3...'
```

Save the output string, since we are going to need it later, then type `exit` to return to the bash shell.
Next, we need to install Jupyter Notebook:

```
conda install jupyter notebook
```

Then, we can generate a configuration file by executing the following commands:

```
jupyter notebook --generate-config
mkdir certificates
```

(By default, the configuration file should be written to `/home/ubuntu/.jupyter/jupyter_notebook_config.py`.)
In the next step, we are going to generate a new SSL certificate via the following command:

```
sudo openssl req -days 365 -nodes \
-x509 -newkey rsa:1024 \
-keyout certificates/mycertificate.pem \
-out certificates/mycertificate.pem
```

Next, open the file `~/.jupyter/jupyter_notebook_config.py` in a text editor (for example, `vi`) and add the following lines:

```
c.IPKernelApp.pylab = 'inline'
c.NotebookApp.open_browser = True
c.NotebookApp.certfile = u'/home/ubuntu/certifi
cates/mycertificate.pem'
c.NotebookApp.ip = '*'
c.NotebookApp.password = u'<insert the sha1...
string here>'
c.NotebookApp.port = 8888
```

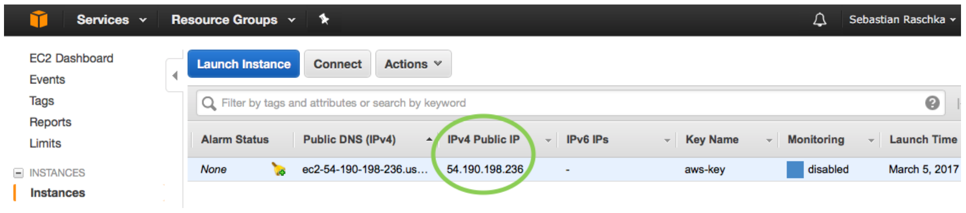
We should be all set now! You can now start a new Jupyter Notebook session by typing the following command in your AWS instance:

```
jupyter notebook
```

Then, head over to your favorite web browser on your local machine and enter the following line into the address bar:

```
https://[all ip addresses on your system]:8888/
```

where `[all ip addresses on your system]` needs to be replaced by your instance's IP address – if you forgot the address, you can get it from the EC2 dashboard (Figure H.15).



Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name	Monitoring	Launch Time
None	ec2-54-190-198-236.us…	54.190.198.236	-	aws-key	disabled	March 5, 2017

Figure H.15: EC2 instance IP address.

H.11 Creating a Custom Amazon Machine Image

Now that we have gone through all this work, it would be a good idea to save the Amazon Machine Image (AMI) so that we can re-use it at any point in future without going through the set-up steps again. But first, let us clean-up our home directory by removing the files that we do not need anymore:

```
cd ~
rm Miniconda3-latest-Linux-x86_64.sh
rm NVIDIA-Linux-x86_64-367.57.run
rm cuda_8.0.61_375.26_linux-run
rm libcudnn5_5.1.10-1+cuda8.0_amd64.deb
rm libcudnn5-dev_5.1.10-1+cuda8.0_amd64.deb
```

After we did this spring cleaning to save storage space, we can create a new image. Head over to the EC2 panel in your web browser, click on the "Actions" button and select "Image > Create Image" (Figure H.16).

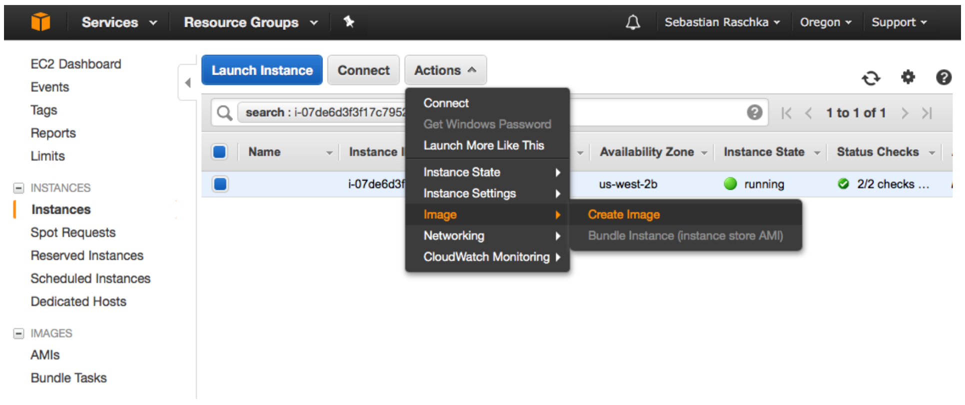
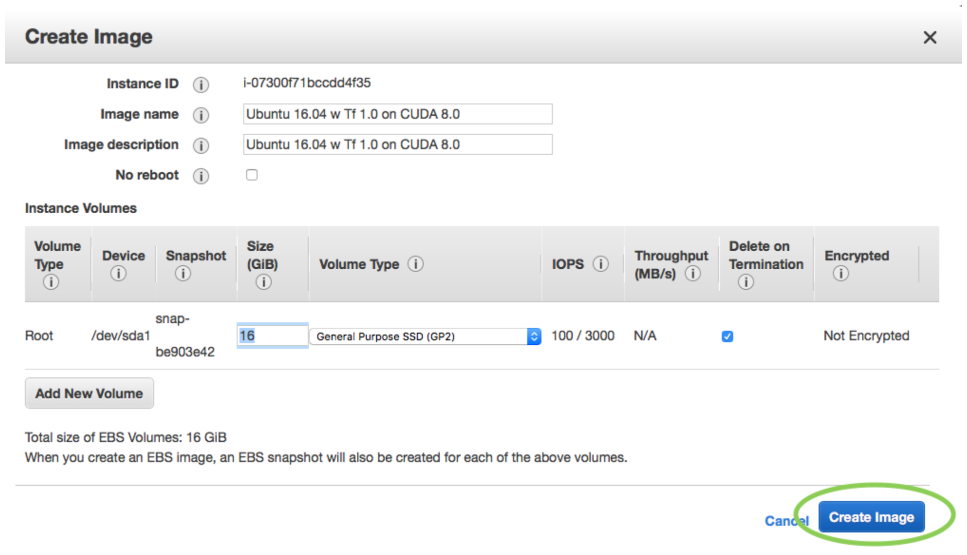


Figure H.16: Creating a new machine image.

In the pop-up window, provide your desired "Image name" and "Image description" and click on the "Create Image" button on the lower right to create your image (Figure H.17).



Create Image

Instance ID	i-07300f71bccdd4f35
Image name	Ubuntu 16.04 w Tf 1.0 on CUDA 8.0
Image description	Ubuntu 16.04 w Tf 1.0 on CUDA 8.0
No reboot	☐

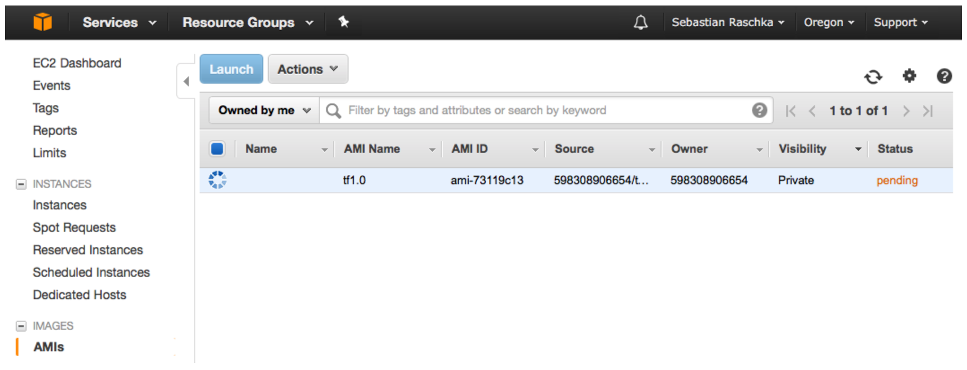
Instance Volumes

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-be903e42	16	General Purpose SSD (GP2)	100 / 3000	N/A	✓	Not Encrypted

Total size of EBS Volumes: 16 GiB
When you create an EBS image, an EBS snapshot will also be created for each of the above volumes.

Figure H.17: Adding a description and name for the new machine image.

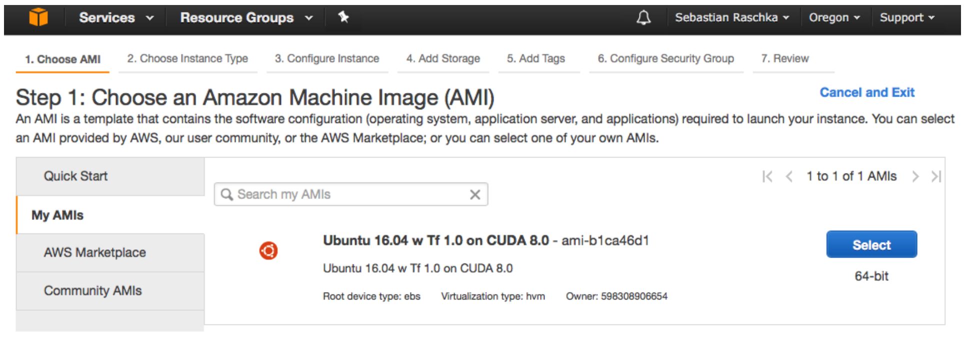
Usually, it takes a couple of minutes until your image has been created. You can check the status by clicking "AMIs" in the left sub-panel on AWS (Figure H.18).



Name	AMI Name	AMI ID	Source	Owner	Visibility	Status
	tf1.0	ami-73119c13	598308906654/t...	598308906654	Private	pending

Figure H.18: Checking the machine image creation process.

Once it says "available" in the status column, our newly created AMI is available for our future AWS instances. So, the next time you are starting an AWS instance, you can simply select the instance you have just created from the "My AMIs" panel (Figure H.19).



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Ubuntu 16.04 w Tf 1.0 on CUDA 8.0 - ami-b1ca46d1
Ubuntu 16.04 w Tf 1.0 on CUDA 8.0
Root device type: ebs Virtualization type: hvm Owner: 598308906654

Figure H.19: Choosing an existing machine image.

H.12 Wrap Up

As mentioned in the very beginning of this appendix, running AWS GPU instances is *not* free. So, if you are done with your computations, please make sure that you shut down your AWS instance to avoid unnecessary costs. On AWS, there are two ways to stop your instance, `stop` or `terminate` (Figure H.20).

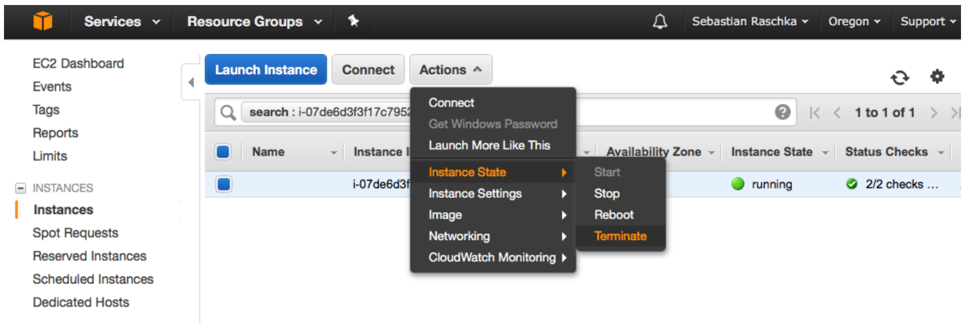


Figure H.20: Shutting instances down.

Choosing the `terminate` option has consequences ... it means that everything gets wiped off your instances' storage. Only if you are sure that you have either

1. saved all data to a local/permanent storage
2. have no need of this data in future

feel free to hit the `terminate` button. The other option to shut down an AWS instance is the `stop` option, which will power down the instance, however, the data will be saved and you can power up the instance in any time in future – keep in mind that you may be charged with storage costs over time, though.

Bibliography

Abbreviations and Terms

AMI	[Amazon Machine Image]
API	[Application Programming Interface]
CNN	[Convolutional Neural Network]

Index