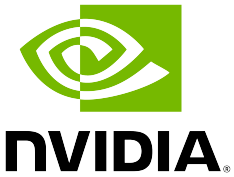

NVIDIA Base Command Manager 10

Machine Learning Manual

Revision: a360e32d4

Date: Fri Apr 10 2026

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of SUSE LLC. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, and NVLink are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. NVIDIA Corporation shall not be liable for technical or editorial errors or omissions which may occur in this document. NVIDIA Corporation shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to NVIDIA Corporation

The NVIDIA Base Command Manager product principally consists of free software that is licensed by the Linux authors free of charge. NVIDIA Corporation shall have no liability nor will NVIDIA Corporation provide any warranty for the NVIDIA Base Command Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the NVIDIA Base Command Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the NVIDIA Base Command Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

Table of Contents

Preface

Welcome to the *Machine Learning Manual* for NVIDIA Base Command Manager 10.

0.1 About This Manual

This manual is aimed at helping cluster administrators install, understand, configure, and manage basic machine learning capabilities easily using NVIDIA Base Command Manager. The administrator is expected to be reasonably familiar with the *Administrator Manual*.

0.2 About The Manuals In General

Name Changes From Version 9.2 To 10

The cluster manager software was originally developed by Bright Computing and the name "Bright" featured previously in the product, repositories, websites, and manuals.

Bright Computing was acquired by NVIDIA in 2022. The corresponding name changes, to be consistent with NVIDIA branding and products, are a work in progress. There is some catching up to do in places. For now, some parts of the manual still refer to Bright Computing and Bright Cluster Manager. These remnants will eventually disappear during updates.

BCM in particular is a convenient abbreviation that happens to have the same letters as the former Bright Cluster Manager. With the branding change in version 10, Base Command Manager is the official full name for the product formerly known as Bright Cluster Manager, and BCM is the official abbreviation for Base Command Manager.

Regularly updated versions of the NVIDIA Base Command Manager 10 manuals are available on updated clusters by default at `/cm/shared/docs/cm`. The latest updates are always online at `https://docs.nvidia.com/base-command-manager`.

- The *Installation Manual* describes installation procedures for a basic cluster.
- The *Administrator Manual* describes the general management of the cluster.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with BCM.
- The *Edge Manual* describes how to deploy BCM Edge with BCM.
- The *Machine Learning Manual*—this manual—describes how to install and configure machine learning capabilities with BCM.
- The *Containerization Manual* describes how to manage containers with BCM.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: `<Alt>-<Backarrow>` in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the BCM environment and the addition of new hardware and/or applications. The manuals also regularly incorporate feedback from administrators and users, and any comments, suggestions or corrections will be very gratefully accepted at `manuals@brightcomputing.com`.

There is also a feedback form available via Base View, via the menu icon,  , following the navigation path:

 `>Help>Feedback`

0.3 Getting Administrator-Level Support

Support for BCM subscriptions from version 10 onwards is available via the NVIDIA Enterprise Support page at:

`https://www.nvidia.com/en-us/support/enterprise/`

Section 16.2 of the *Administrator Manual* has more details on working with support.

0.4 Getting Professional Services

The BCM support team normally differentiates between

- regular support (customer has a question or problem that requires an answer or resolution), and
- professional services (customer asks for the team to do something or asks the team to provide some service).

Professional services can be provided via the NVIDIA Enterprise Services page at:
`https://www.nvidia.com/en-us/support/enterprise/services/`

1

Introduction And Machine Learning Packages Installation

1.1 Introduction

A number of machine learning and deep learning library and framework packages can be used with BCM The packages provided make it faster and easier for organizations to install the latest state-of-the-art libraries, and gain insights from rich, complex data.

1.1.1 Associated Repository, And Support For Machine Learning Packages

The packages are distributed via a separate BCM `cm-ml` repository. The repository is automatically available and enabled.

The support team supports the integration of packages distributed in the dedicated `cm-ml` repository with BCM only if the cluster is licensed for the Data Science Add-on package.

An administrator who is upgrading a cluster that has machine learning and deep learning packages installed on it should always make sure that the dedicated BCM `cm-ml` repository is accessible, if required by the new NVIDIA Base Command Manager version.

1.1.2 NVIDIA Base Command Manager Versions—Supported Distributions And Architecture For Machine Learning Packages

At the time of writing of this section, October 2024, a number of different Linux distribution versions and architectures are supported, depending on the NVIDIA Base Command Manager version. For convenience, a support matrix for this is shown in table 1.1:

Table 1.1: *Supported Linux distributions and architectures for BCM Machine Learning packages*

BCM version	Architectures	Linux distributions
8.0	x86_64	CentOS 7, RHEL 7
8.1	x86_64	CentOS 7, RHEL 7
8.2	x86_64	CentOS 7, RHEL 7, Ubuntu 18.04
9.0	x86_64	CentOS 7, Rocky 8, RHEL 7, RHEL 8, SLES 15, Ubuntu 18.04

9.1	x86_64	CentOS 7, RHEL 7, RHEL 8, Rocky 8, SLES 15, Ubuntu 18.04, Ubuntu 20.04
9.2	x86_64	CentOS 7, RHEL 7, RHEL 8, Rocky 8, RHEL 9, Rocky 9, SLES 15, Ubuntu 18.04, Ubuntu 20.04, Ubuntu 22.04
10.0	x86_64	RHEL 8, RHEL9, Rocky 8, Rocky 9, SLES 15, Ubuntu 20.04, Ubuntu 22.04

An updated list of the supported Linux distributions and architectures available for the various NVIDIA Base Command Manager versions can be found at `https://support.brightcomputing.com/feature-matrix/`, in the section of the `Feature` column dedicated to machine learning.

1.2 Available Packages

An updated list of the machine learning packages available for the various NVIDIA Base Command Manager versions can be found at `https://support.brightcomputing.com/packages-dashboard/`. Most of the machine learning packages are to be found within the `ml` group. However, some of them are found within the `cm` group for legacy reasons.

At the time of writing, December 2021, the following packages were available:

Table 1.2: *Machine Learning packages provided by the NVIDIA Base Command Manager repositories*

Package name	Description
`cm-cub-cuda11.8`	A flexible library of cooperative threadblock primitives and other utilities for CUDA kernel programming.
`cm-fastai2-py39-cuda11.8-gcc11`	A library that simplifies training fast and accurate neural nets using modern best practices.
`cm-gpytorch-py39-cuda11.8-gcc11`	A Gaussian process library implemented using PyTorch.
`cm-ml-distdeps-cuda11.8`	Meta-package containing distribution-specific dependencies for machine learning frameworks.
`cm-ml-pythondeps-py39-cuda11.8-gcc11`	Meta-package containing Python dependencies for machine learning frameworks. It includes, for example, NumPy, PyCUDA, pandas, NetworkX, pydot and Matplotlib.
`cm-nccl2-cuda11.8-gcc11`	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.

...continues

Table 1.2: Machine Learning Packages Included...continued

Package name	Description
cm-nccl2-cuda12.0-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.1-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.2-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.3-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.4-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.5-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-nccl2-cuda12.6-gcc11	Version 2 of NCCL, an implementation of multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs.
cm-onnx-pytorch-py39-cuda11.8-gcc11	An open format built to represent machine learning models.
cm-opencv4-py39-cuda11.8-gcc11	An open-source BSD-licensed library that includes several hundreds of computer vision algorithms.
cm-protobuf3-gcc11	Version 3 of Protocol Buffers, a language-neutral, platform-neutral extensible mechanism for serializing structured data.
cm-pytorch-extra-py39-cuda11.8-gcc11	A collection of models, libraries, dataset and useful extra functionality for PyTorch.
cm-pytorch-py39-cuda11.8-gcc11	An optimized tensor library for deep learning using GPUs and CPUs. It provides tensor computation (like NumPy) with strong GPU acceleration, and deep neural networks built on a tape-based autograd system. It now includes Caffe2.

...continues

Table 1.2: Machine Learning Packages Included...continued

Package name	Description
cm-tensorflow2-py39-cuda11.8-gcc11	An open source software library for numerical computation using data flow graphs originally developed by researchers and engineers working on the Google Brain team.

Legend:

Package are available for every distribution unless otherwise tagged.

1.2.1 Considerations

There are some considerations that the cluster administrator should be aware of with the packages.

- Some packages may be labelled in the table 1.2 as deprecated. "Deprecated" in the software industry is not a well-defined term. Here it is used by BCM to mean a package that may no longer be offered in a future release, or for which a newer existing version is preferred.
- Several different packages may be provided for the same machine learning library or framework. For example, TensorFlow may be provided by:
  - `cm-tensorflow2-py39-cuda11.2-gcc9`
  - `cm-tensorflow2-py39-cuda11.7-gcc9`

  As is the norm with other package management systems in the software industry, the name given to a BCM package includes the most relevant dependencies required to build and use it. The dependencies commonly highlighted in this manner are:
  - Python interpreter version used (e.g. `*-py36*`, `*-py37*` and `*-py39*`)
  - accelerator library used (e.g. `*-cuda10.1*`, `*-cuda10.2*`, `*-cuda11.2*`, `*-cuda11.7*`, and `*-mkl*`)
  - compiler used (e.g. `*-gcc*`, `*-gcc8*` and `*-gcc9*`)

  The availability of different variants of the same package makes it easier for administrators to set up a working environment that is suited to their needs.
- Machine learning packages are designed to coexist, and can therefore all be installed at the same time. This also applies to different variants of the same library or framework.

  This means that administrators can install different versions of the same machine learning library or framework, simply by using different variants. For example: an older `*-py36*` version of TensorFlow, as well as a more recent `*-py37*` version of TensorFlow, can both be installed at the same time.
- As is done with other packages provided by the BCM developers, the updates released for machine learning libraries and frameworks generally leave their major versions unchanged.

  Whenever a major version for a third party machine learning library or a framework is publicly released, a new package or a set of packages is typically placed in the repository.

  Such package(s) imply or contain a reference to the major version in the name. For example:
  - `cm-tensorflow-*` is the name used for TensorFlow major version 1
  - `cm-tensorflow2-*` is the name for TensorFlow major version 2

  As a result, administrators can safely upgrade cluster packages without breaking backward compatibility with users' applications.

- MPI modules and libraries should not be blindly added by the cluster administrator. During module loading, warnings are typically given to suggest an MPI library (Open MPI, or an MPICH or MVAPICH implementation of Open MPI) is required. However, the exact implementation of the MPI library that can be used depends upon the hardware (GPU, interface, architecture) used and requires judgment of suitability by the cluster administrator. BCM uses the `cm-openmpi4-cuda11.2-ofed51-gcc9` package in this manual as the reference MPI library implementation. This driver package corresponds with using Open MPI with Gigabit Ethernet networking, InfiniBand networking, and NVIDIA GPUs.

1.3 Requirements

The following requirements must be met before installing the preceding machine learning packages.

- RHEL users must have access to the YUM repositories and EPEL repository.
- There must be enough free space for the packages that are installed on the head node and compute nodes. The actual amount depends on the packages installed.
- 8 GB of RAM on the nodes is the minimum recommended amount.
- In order to use packages built with the CUDA toolkit accelerator library version 10.2 and below (e.g. `*-cuda10.2*`), the NVIDIA GPUs must be Maxwell or more recent, with compute capability 3.5 or later. CUDA compute capability 6.0 or later is recommended.
- In order to use packages built with the CUDA toolkit accelerator library version 11.0 and above (e.g. `*-cuda11.2*`), the NVIDIA GPUs must be Maxwell or more recent, with compute capability 5.2 or later. CUDA compute capability 6.0 or later is recommended.
- In order to use packages built using CUDA as the accelerator (i.e. `*-cuda*`), the CPU must support the AVX/AVX2, FMA, and SSE4.2 instructions. This can be checked by inspecting the CPU flags:

  Example

  ```
  [root@node ~]# egrep -m1 -o '(avx|avx2|fma|sse4_2)' /proc/cpuinfo
  fma
  sse4_2
  avx
  avx2
  ```

- In order to use packages built using MKL as the accelerator (i.e. `*-mkl`), the CPU must support the AVX-512 Vector Neural Network Instructions (VNNI). Examples of such CPUs are Intel Xeon Scalable processors with Deep Learning Boost.

1.4 Machine Learning Packages Installation

Head Node Installation

BCM machine learning packages are installed in the `/cm/shared` directory, which is by default exported over NFS. Packages installed on the head node are therefore also available to all the compute nodes by default.

The `.rpm` and `.deb` files have proper dependencies defined. This means that the cluster administrator does not need to spend time figuring out what needs to be installed to set up a working environment. Whenever a package is installed or updated, the required dependencies will be also automatically fetched, if necessary. As a result, packages can be installed with the usual package manager that is provided by the Linux distribution in the usual way (page 549 of the *Administrator Manual*).

For example, the administrator can install `cm-pytorch-py39-cuda11.8-gcc11` as follows:

Example

```
[root@basecm10 ~]# yum install cm-pytorch-py39-cuda11.8-gcc11       #on RHEL 8 and 9

[root@basecm10 ~]# zypper install cm-pytorch-py39-cuda11.8-gcc11    #on SLES 15

[root@basecm10 ~]# apt-get install cm-pytorch-py39-cuda11.8-gcc11   #on Ubuntu 22.04
```

The package managers also automatically install the corresponding package dependencies, such as

- `cm-cudnn8.6-cuda11.8`
- `cm-ml-pythondeps-py39-cuda11.8-gcc11`
- `cm-nccl2-cuda11.8-gcc11`
- `cm-protobuf3-gcc11`
- `cuda11.8-toolkit`

Machine learning packages share several dependencies, usually providing useful Python or system libraries. For convenience, these dependencies are grouped in the `cm-ml-pythondeps-*` and `cm-ml-distdeps-*` meta-packages.

- `cm-ml-pythondeps-*`: This meta-package provides the application libraries such as `numba`, `numpy`, `scikit-learn`, and `scipy`.
  - If RHEL8 is being run, and `cm-ml-pythondeps-py39-cuda11.8-gcc11` is being installed, then the `codeready-builder-for-rhel-8-x86_64-rpms` repository needs to be enabled. For example using:
    `subscription-manager repos --enable=codeready-builder-for-rhel-8-x86_64-rpms`
- `cm-ml-distdeps-*`: This meta-package, on the other hand, provides development libraries such as `blas-devel`, `libjpeg-devel` and `libpng-devel`, and the utility library `gnuplot`.

The appropriate meta-packages are automatically installed whenever a package installation requires it.

Administrators only need to make sure that their clusters meet the preceding hardware requirements listed at the start of section 1.3. If that is not done, then unexpected failures may occur during run time, such as segmentation faults.

Examples of common mistakes are

- using packages requiring CUDA (e.g. `cm-pytorch-py39-cuda11.8-gcc11`) on clusters without GPUs
- using packages requiring VNNI (e.g. `cm-tensorflow2-py39-cuda11.8-gcc11`) on CPUs not supporting the instruction set

Compute Nodes Installation

The `cm-ml-distdeps-*` meta-packages must be also installed onto all compute nodes that are to run machine learning applications.

For example, if the name of the software image is `gpu-image`, then the administrator can install `cm-ml-distdeps-cuda11.8` on RHEL 9 as follows:

Example

```
[root@basecm10 ~]# yum install --installroot=/cm/images/gpu-image cm-ml-distdeps-cuda11.8
```

The preceding command must be applied to all software images that are used to run machine learning applications.

There are equivalents to the `--installroot` option of `yum` for the other distribution package managers.

For SLES the installation command equivalent is:

```
[root@basecm10 ~]# zypper --root /cm/images/gpu-image install cm-ml-distdeps-cuda11.8
```

For Ubuntu the installation command equivalent is:

```
[root@basecm10 ~]# cm-chroot-sw-img /cm/images/gpu-image
[root@basecm10 ~]# apt install cm-ml-distdeps-cuda11.8
[root@basecm10 ~]# exit     #get out of chroot
```

Details on using `zypper` and `apt` commands for installation to software images are given on page 549 of the *Administrator Manual*.

The preceding command must be applied to all software images that are used to run machine learning applications.

No automatic install of `cuda-driver` and `cuda-dcgm` since `cm-ml-distdeps-cuda-*` v3.0.0: The `cuda-driver` and `cuda-dcgm` packages used to be automatically installed during installation of earlier versions of the `cm-ml-distdeps-cuda-*` meta-package. This behavior has changed.

Version 3.0.0 onward of the package requires a manual install of the `cuda-driver` and `cuda-dcgm` packages. Installation of the `cuda-driver` and `cuda-dcgm` packages is covered in section 9.1 of the *Installation Manual*.

The version number of the available or installed `cm-ml-distdeps-cuda*` package can be found with the package manager. For example using `yum info`:

Example

```
[root@basecm10 ~]# yum info cm-ml-distdeps-cuda*
...
Available Packages
Name         : cm-ml-distdeps-cuda11.7
Version      : 3.0.1
...
```

A distribution-agnostic way to find the version is with the `packages` command (section 11.1.1 of the *Administrator Manual*):

Example

```
[basecm10->device[basecm10]]% packages -f cm-ml-distdeps-cuda11.2 -n basecm10
Node     Type     Name                      Version  Release                  Arch     Size
-------- -------- ------------------------ -------- ------------------------ -------- --------
basecm10 rpm      cm-ml-distdeps-cuda11.2  3.0.1    100092_cm9.2_db74c57a10  x86_64   0B
```

The manual installation of the `cuda-driver` and `cuda-dcgm` packages allows a wider range of NVIDIA drivers and cluster configurations to be installed. Version 3.0.0 onward now allows the administrator to install custom NVIDIA drivers, for example for special hardware such as DGX machines. It also allows the administrator to install different versions of NVIDIA drivers for different groups of compute nodes.

Just as for the `cm-ml-distdeps-*` meta-packages, the custom NVIDIA drivers must be installed onto all the compute nodes that are to run machine learning applications.

1.4.1 Module Loading

BCM provides environment module definitions for all the machine learning packages. The environment module files are also compatible with the Lmod software introduced in NVIDIA Base Command Manager 7.3. They can be listed once the `shared` module is loaded, if it has not already been loaded:

```
[root@basecm10 ~]# module purge; module available
------------------------------------ /cm/local/modulefiles -------------------------------------
boost/1.77.0              cm-setup/9.2       dot              luajit        openldap
cluster-tools/9.2         cmake-gcc9/3.21.3  freeipmi/1.6.8   mariadb-libs  python3
cm-bios-tools             cmd                gcc/11.2.0       module-git    python39
cm-image/9.2              cmjob              ipmitool/1.8.18  module-info   shared
cm-scale/cm-scale.module  cmsh               lua/5.4.4        null

[root@basecm10 ~]# module load shared; module available
------------------------------------ /cm/local/modulefiles -------------------------------------
boost/1.77.0              cm-setup/9.2       dot              luajit        openldap
cluster-tools/9.2         cmake-gcc9/3.21.3  freeipmi/1.6.8   mariadb-libs  python3
cm-bios-tools             cmd                gcc/11.2.0       module-git    python39
cm-image/9.2              cmjob              ipmitool/1.8.18  module-info   shared
cm-scale/cm-scale.module  cmsh               lua/5.4.4        null

------------------------------------ /cm/shared/modulefiles ------------------------------------
blacs/openmpi/gcc/64/1.1patch03  intel-tbb-oss/ia32/2021.4.0                ucx/1.10.1
blas/gcc/64/3.10.0               intel-tbb-oss/intel64/2021.4.0
bonnie++/2.00a                   iozone/3_492
cm-pmix3/3.1.4                   lapack/gcc/64/3.10.0
cm-pmix4/4.1.1                   ml-pythondeps-py39-cuda11.2-gcc9/4.8.1
cuda11.2/blas/11.2.2             mpich/ge/gcc/64/3.4.2
cuda11.2/fft/11.2.2              mvapich2/gcc/64/2.3.7
cuda11.2/toolkit/11.2.2          nccl2-cuda11.2-gcc9/2.14.3
cudnn8.1-cuda11.2/8.1.1.33       netcdf/gcc/64/gcc/64/4.8.1
default-environment              netperf/2.7.0
fftw3/openmpi/gcc/64/3.3.10      openblas/dynamic/(default)
gcc9/9.5.0                       openblas/dynamic/0.3.18
gdb/10.2                         opencv4-py39-cuda11.2-gcc9/4.5.4
globalarrays/openmpi/gcc/64/5.8  openmpi/gcc/64/4.1.2
hdf5/1.12.1                      openmpi4-cuda11.2-ofed51-gcc9/4.1.4
hdf5_18/1.8.21                   openmpi4/gcc/4.1.2
hpcx/mlnx-ofed51/2.7.4           protobuf3-gcc9/3.9.2
hwloc/1.11.11                    tensorflow2-extra-py39-cuda11.2-gcc9/2.7.0
hwloc2/2.7.1                     tensorflow2-py39-cuda11.2-gcc9/2.7.0
```

For example, after having installed the `cm-tensorflow2-py39-cuda11.8-gcc11` package, the associated TensorFlow module can be loaded with:

```
[root@basecm10 ~]# module load tensorflow2-py39-cuda11.8-gcc11
Loading tensorflow2-py39-cuda11.8-gcc11/2.11.0
Loading requirement: openblas/dynamic/0.3.18 hdf5_18/1.8.21 gcc11/11.3.0 python39\
cuda11.8/toolkit/11.8.0 cudnn8.6-cuda11.8/8.6.0.163 ml-pythondeps-py39-cuda11.8-gcc11/4.12.0\
protobuf3-gcc11/3.9.2 nccl2-cuda11.8-gcc11/2.16.5
```

The machine learning environment modules automatically load additional environment modules as dependencies, with the notable exception of Open MPI implementations for the reasons given in section 1.2.1.

The module dependencies are achieved via the module definition files:

Example

```
[root@basecm10 ~]#  module show tensorflow2-py39-cuda11.8-gcc11/
-------------------------------------------------------------------
/cm/shared/modulefiles/tensorflow2-py39-cuda11.8-gcc11/2.11.0:

module-whatis   {adds TensorFlow2 to your environment variables}
module          load ml-pythondeps-py39-cuda11.8-gcc11
module          load protobuf3-gcc11
module          load cudnn8.1-cuda11.8
module          load nccl2-cuda11.8-gcc11
prepend-path    PYTHONPATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib/python3.9/\
site-packages/
prepend-path    PYTHONPATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib64/python3.9/\
site-packages/
prepend-path    LD_LIBRARY_PATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib/python3.9/\
site-packages/tensorflow
prepend-path    LD_LIBRARY_PATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib/python3.9/\
site-packages/tensorflow_core
prepend-path    LD_LIBRARY_PATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib64/python3.9/\
site-packages/tensorflow
prepend-path    LD_LIBRARY_PATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/lib64/python3.9/\
site-packages/tensorflow_core
prepend-path    PATH /cm/shared/apps/tensorflow2-py39-cuda11.8-gcc11/2.11.0/bin
-------------------------------------------------------------------
```

1.5 Machine Learning Packages Removal

Machine learning packages can be removed in the usual way with the package manager commands used by the Linux distribution. For example, the administrator can remove the `cm-pytorch-py37-cuda10.2-gcc8` package with:

Example

```
[root@basecm10 ~]# yum remove cm-pytorch-py39-cuda11.8-gcc11       #on RHEL 8 and 9
```

or

```
[root@basecm10 ~]# zypper remove cm-pytorch-py39-cuda11.8-gcc11    #on SLES 15
```

or

```
[root@basecm10 ~]# apt-get remove cm-pytorch-py39-cuda11.8-gcc11   #on Ubuntu 22.04
```

BCM machine learning packages are installed in the `/cm/shared` directory, which is by default exported over NFS. Packages removed from the head node are therefore also removed from all the compute nodes by default.

The `cm-ml-distdeps-*` meta-packages must be also removed from all compute nodes that are to run machine learning applications. For example, if the name of the software image is `gpu-image`, then the images directory is `/cm/images/gpu-image`, and then the administrator can remove `cm-ml-distdeps-cuda11.8` from the image as follows in RHEL 9:

Example

```
[root@basecm10 ~]# yum remove --installroot=/cm/images/gpu-image cm-ml-distdeps-cuda11.8
```

The preceding command must be applied to all software images that are used to run the machine learning applications.

The equivalents to the `--installroot` option of `yum` for the other distribution package managers are described in section 1.4.

2

Running TensorFlow

This chapter goes through some example runs with TensorFlow. Some output messages have been removed or simplified in the runs for readability.

The sample runs assume that the GPU drivers are in place (section 9.1 of the *Installation Manual*). The runs also assume that TensorFlow package and the TensorFlow extras package have both been installed from the NVIDIA Base Command Manager repository with a package manager.

TensorFlow can be installed, for example with

```
yum install cm-tensorflow2-py39-cuda11.2-gcc9 cm-tensorflow2-extra-py39-cuda11.2-gcc9
```

In addition to requiring the extras package, TensorFlow requires an OpenMPI implementation to work. Chapter 3 of the *User Manual* describes the different OpenMPI packages that the BCM repositories provide. The different OpenMPI packages allow the user to choose which one to use. For example, depending on which interconnect is being used, or depending on if CUDA support is required.

In this chapter, the `cm-openmpi4-cuda11.2-ofed51-gcc9` package is used. It can be installed with, for example:

```
yum install cm-openmpi4-cuda11.2-ofed51-gcc9
```

More information on the examples can be found at `https://github.com/tensorflow/examples`.

2.1 Hello World

A "`Hello World`" interactive example, that just shows that the software is in place for TensorFlow, can be run as follows:

Example

```
[root@basecm10 ~]# module load shared
[root@basecm10 ~]# module load tensorflow2-py39-cuda11.2-gcc9
Loading tensorflow2-py39-cuda11.2-gcc9/2.7.0
Loading requirement: openblas/dynamic/0.3.18 hdf5_18/1.8.21 gcc9/9.5.0 python39 \
cuda11.2/toolkit/11.2.2 cudnn8.1-cuda11.2/8.1.1.33 ml-pythondeps-py39-cuda11.2-gcc9/4.8.1 \
protobuf3-gcc9/3.9.2 nccl2-cuda11.2-gcc9/2.14.3
[root@basecm10 ~]# module load openmpi4-cuda11.2-ofed51-gcc9/
Loading openmpi4-cuda11.2-ofed51-gcc9/4.1.4
  Loading requirement: hpcx/mlnx-ofed51/2.7.4 ucx/1.10.1 cm-pmix3/3.1.4 hwloc/1.11.11
[root@basecm10 ~]# python3.9
Python 3.9.16 (main, Mar 16 2023, 14:03:52)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import tensorflow as tf
...
>>> hello = tf.constant('TensorFlow 2 Hello World')
...
2023-03-22 17:20:10.514379: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created \
device /job:localhost/replica:0/task:0/device:GPU:0 with 31002 MB memory:  -> device: 0, name: \
Tesla V100-SXM3-32GB, pci bus id: 0000:00:06.0, compute capability: 7.0
>>>

>>> hello = tf.constant('TensorFlow 2 Hello World')
>>> tf.print(hello)
TensorFlow 2 Hello World
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> tf.print(a+b)
42
>>>
```

2.2 Deep Convolutional Generative Adversarial Network

The following trains a deep convolutional generative adversarial network (DCGAN).

The example code is included in the TensorFlow extras package. Once its module has been loaded, the example directory is defined with the `CM_TENSORFLOW2_EXTRA` environment variable.

The example picks up training images and labels from the MNIST site, and places them in a directory `tensorflow_datasets/` if it needs to. The images are then used to train the model. End users would be expected to train the neural network within their home directories.

Example

```
[root@basecm10 ~]# module load tensorflow2-extra-py39-cuda11.2-gcc9
Loading tensorflow2-extra-py39-cuda11.2-gcc9/2.7.0
  Loading requirement: openblas/dynamic/0.3.18 hdf5_18/1.8.21 gcc9/9.5.0 python39 \
  cuda11.2/toolkit/11.2.2 cudnn8.1-cuda11.2/8.1.1.33 ml-pythondeps-py39-cuda11.2-gcc9/4.8.1\
  protobuf3-gcc9/3.9.2 nccl2-cuda11.2-gcc9/2.14.3
  tensorflow2-py39-cuda11.2-gcc9/2.7.0 opencv4-py39-cuda11.2-gcc9/4.5.4
[root@basecm10 ~]# module load openmpi4-cuda11.2-ofed51-gcc9/
  Loading requirement: hpcx/mlnx-ofed51/2.7.4 ucx/1.10.1 cm-pmix3/3.1.4 hwloc/1.11.11
[root@basecm10 ~]# cd ${CM_TENSORFLOW2_EXTRA}/tensorflow_examples/models/dcgan/
[root@basecm10 dcgan]# python3.9 dcgan.py --epochs 5
...
I0322 18:24:37.097312 23456247932736 dataset_builder.py:400] Generating dataset mnist \
(/root/tensorflow_datasets/mnist/3.0.1)
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown \
size, total: Unknown size) to /root/tensorflow_datasets/mnist/3.0.1...
Dl Completed...: 0 url [00:00, ? url/s]          I0322 18:24:37.484689 23456247932736 \
download_manager.py:354] Downloading https://storage.googleapis.com/cvdf-datasets/mnist/...
...
Generating splits...:  ...
...
2023-03-22 18:24:39.178613: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created \
device /job:localhost/replica:0/task:0/device:GPU:0 with 31002 MB memory:  -> device: 0, name: \
Tesla V100-SXM3-32GB, pci bus id: 0000:00:06.0, compute capability: 7.0
       ... Done writing mnist-train.tfrecord. Number of examples: 60000 (shards: [60000])
Generating splits...: ...
Dataset mnist downloaded and prepared to /root/tensorflow_datasets/mnist/3.0.1. Subsequent calls will \
reuse this data.
```

```
I0322 18:25:39.912966 23456247932736 logging_logger.py:35] Constructing tf.data.Dataset mnist for split \
train, from /root/tensorflow_datasets/mnist/3.0.1
Training ...
2023-03-22 18:25:42.881613: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8101
Epoch 0, Generator loss 1.0063459873199463, Discriminator Loss 1.0740859508514404
Epoch 1, Generator loss 0.9040709733963013, Discriminator Loss 1.0049669742584229
Epoch 2, Generator loss 0.88487708568573, Discriminator Loss 1.3763940334320068
Epoch 3, Generator loss 0.8470083475112915, Discriminator Loss 1.4865632057189941
Epoch 4, Generator loss 1.2192224264144897, Discriminator Loss 1.2187243700027466
[root@basecm10 dcgan]#
```

Datasets generated by the preceding run can be purged with:

Example

```
[root@basecm10 dcgan]# rm -rf /root/tensorflow_datasets/
```

2.3 Image-to-Image Translation with Conditional Adversarial Nets

The following trains a conditional adversarial networks as a general-purpose solution to image-to-image translation problems. The trained model is capable of completing different tasks, such as colorizing black and white photos.

The example code is included in the TensorFlow extras package. Once its module has been loaded, the example directory is defined with the `CM_TENSORFLOW2_EXTRA` environment variable.

The example uses a preprocessed copy of the CMP Facade Database, helpfully provided by the Center for Machine Perception at the Czech Technical University in Prague. The original dataset includes 606 rectified images of facades from various sources, which have been manually annotated. The facades are from different cities around the world and diverse architectural styles.

The example conveniently downloads the dataset in a temporary directory and then trains the model. End users would be expected to train the neural network within their home directories.

Example

```
[root@basecm10 ~]# module load tensorflow2-extra-py39-cuda11.2-gcc9
[root@basecm10 ~]# module load openmpi4-cuda11.2-ofed51-gcc9/
[root@basecm10 ~]# cd ${CM_TENSORFLOW2_EXTRA}/tensorflow_examples/models/pix2pix
[root@basecm10 pix2pix]# python3.9 data_download.py --download_path /tmp
...
Downloading data from http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/facades.tar.gz
...
[root@basecm10 pix2pix]# python3.9 pix2pix.py --path /tmp/facades --epochs 5
...
2023-03-22 20:11:12.173915: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created \
device /job:localhost/replica:0/task:0/device:GPU:0 with 31002 MB memory:  -> device: 0,\
name: Tesla V100-SXM3-32GB, pci bus id: 0000:00:06.0, compute capability: 7.0
Training ...
2023-03-22 20:11:16.660300: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8101
Epoch 0, Generator loss 39.03266906738281, Discriminator Loss 0.23912914097309113
Epoch 1, Generator loss 41.69317626953125, Discriminator Loss 0.07581206411123276
Epoch 2, Generator loss 33.53696823120117, Discriminator Loss 0.13100799918174744
Epoch 3, Generator loss 30.571992874145508, Discriminator Loss 1.0946425199508667
Epoch 4, Generator loss 29.1583309173584, Discriminator Loss 1.0569590330123901
[root@basecm10 pix2pix]#
```

2.4 Neural Machine Translation With Attention

The following trains a sequence to sequence model for neural machine translation with attention using gated recurrent units (GRUs).

The example code is included in the TensorFlow extra package. Once its module has been loaded, the example directory is defined with the `CM_TENSORFLOW2_EXTRA` environment variable.

The example conveniently downloads a dataset for Spanish to English translation in a temporary directory and then trains the model. End users would be expected to train the neural network within their home directories.

Example

```
[root@basecm10 ~]# module load tensorflow2-extra-py39-cuda11.2-gcc9
[root@basecm10 ~]# module load openmpi4-cuda11.2-ofed51-gcc9
[root@basecm10 ~]# cd ${CM_TENSORFLOW2_EXTRA}/tensorflow_examples/models/nmt_with_attention/
[root@basecm10 nmt_with_attention]# python3.9 train.py --epochs 5 --download_path /tmp
...
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip
...
2023-03-22 21:18:11.717144: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created \
device /job:localhost/replica:0/task:0/device:GPU:0 with 31002 MB memory:  -> device: 0,\
name: Tesla V100-SXM3-32GB, pci bus id: 0000:00:06.0, compute capability: 7.0
Training ...
2023-03-22 21:18:27.574448: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8101
Epoch: 0, Train Loss: 1.814919114112854, Test Loss: 2.885890483856201
Epoch: 1, Train Loss: 1.085754156112671, Test Loss: 2.093322515487671
Epoch: 2, Train Loss: 0.6579596996307373, Test Loss: 1.7698687314987183
Epoch: 3, Train Loss: 0.43058785796165466, Test Loss: 1.7104995250701904
Epoch: 4, Train Loss: 0.29615747928619385, Test Loss: 1.7149256467819214
[root@basecm10 nmt_with_attention]#
```

3

Running PyTorch

This chapter goes through some example runs with PyTorch. Some output messages have been removed or simplified in the runs for readability.

The sample runs assume that PyTorch and its extra libraries have been installed from the NVIDIA Base Command Manager repository with a package manager. For example with:

```
yum install cm-pytorch-py37-cuda10.2-gcc8 cm-pytorch-extra-py37-cuda10.2-gcc8
```

In addition to requiring the extra libraries, PyTorch requires an OpenMPI implementation to work. Chapter 3 of the *User Manual* describes the different OpenMPI packages that the BCM repositories provide. The different OpenMPI packages allow the user to choose which one to use. For example, depending on which interconnect is being used, or depending on if CUDA support is required.

In this chapter, the `cm-openmpi4-cuda10.2-ofed50-gcc8` package is used.

More information on the examples can be found at `https://github.com/pytorch/examples`.

3.1 Variational Autoencoders

The following example shows how to train *Variational Autoencoders*[1], powerful generative models that can be used for a wide variety of applications.

The example code is included in the PyTorch extra (`cm-pytorch-extra-*`) package. Once its module has been loaded, the example directory is defined with the `CM_PYTORCH_EXTRA` environment variable.

Example

```
[root@basecm10 ~]# module load shared
[root@basecm10 ~]# module load pytorch-extra-py37-cuda10.2-gcc8
[root@basecm10 ~]# module load openmpi4-cuda10.2-ofed50-gcc8
[root@basecm10 ~]# cd ${CM_PYTORCH_EXTRA}/vae
```

The Variational Autoencoders network is trained by default for 10 epochs. The required dataset can automatically be downloaded and extracted with:

Example

```
[root@basecm10 vae]# python3.7 main.py
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
../data/MNIST/raw/train-images-idx3-ubyte.gz
9920512it [00:04, 2041116.37it/s]
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/raw
```

[1] Original paper: "`Auto-Encoding Variational Bayes`" by Diederik P Kingma and Max Welling: `https://arxiv.org/abs/1312.6114`.

```
...
Processing...
Done!
Train Epoch: 1 [0/60000 (0%)] Loss: 550.187805
Train Epoch: 1 [1280/60000 (2%)] Loss: 323.104736
Train Epoch: 1 [2560/60000 (4%)] Loss: 237.460938
...
Train Epoch: 1 [58880/60000 (98%)] Loss: 130.540909
====> Epoch: 1 Average loss: 164.1742
====> Test set loss: 127.8219
Train Epoch: 2 [0/60000 (0%)] Loss: 127.949753
...
Train Epoch: 10 [58880/60000 (98%)] Loss: 107.980888
====> Epoch: 10 Average loss: 106.1472
====> Test set loss: 105.8715
```

The output sampled digits can be found in the `results` directory:

```
[root@basecm10 vae]# ls results/
reconstruction_10.png  reconstruction_4.png  reconstruction_8.png  sample_2.png  sample_6.png
reconstruction_1.png   reconstruction_5.png  reconstruction_9.png  sample_3.png  sample_7.png
reconstruction_2.png   reconstruction_6.png  sample_10.png         sample_4.png  sample_8.png
reconstruction_3.png   reconstruction_7.png  sample_1.png          sample_5.png  sample_9.png
[root@basecm10 vae]#
```