# GETTING STARTED

## 1 Building LBANN
1.1 Download ......................................................... 3
1.2 Building with Spack ........................................... 3
1.3 Advanced build methods .................................... 6

## 2 Running LBANN
2.1 Anatomy of an LBANN experiment .......................... 19
2.2 Python frontend ................................................ 21
2.3 Protobuf frontend (advanced) ............................... 24

## 3 Papers, Presentations, and Posters

## 4 LBANN Software Architecture and Class Overview
4.1 Trainers (i.e. execution environment) ....................... 27
4.2 Execution Context ............................................. 27
4.3 Termination Criteria (Pending) .............................. 27
4.4 Training Algorithms .......................................... 28
4.5 Model .......................................................... 28

## 5 LBANN API
5.1 Documentation of lbann::Al::dummy_backend ............. 29
5.2 Documentation of lbann::lbann_comm ...................... 29
5.3 Callback Interface .......................................... 36
5.4 Data Readers Interface ..................................... 63
5.5 Data Store Interface ....................................... 90
5.6 Execution Context Interface ................................. 92
5.7 I/O Utilities .................................................. 94
5.8 Layer Interface .............................................. 95
5.9 Metrics Interface ............................................. 143
5.10 Models Interface .......................................... 145
5.11 Objective Function Interface ................................. 149
5.12 Optimizer Interface ........................................ 153
5.13 Trainer Interface ........................................... 160
5.14 Training Algorithm Interface .............................. 161
5.15 Transform Interface ....................................... 162
5.16 General Utilities ........................................... 176
5.17 Weights Interface ........................................... 185

## 6 LBANN Style Guide
6.1 In-Source Documentation .................................... 191
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 LBANN CI</td>
<td></td>
</tr>
<tr>
<td>7.1 Plan Configuration</td>
<td>193</td>
</tr>
<tr>
<td>7.2 Directory Structure</td>
<td>194</td>
</tr>
<tr>
<td>7.3 Writing Your Own Tests</td>
<td>194</td>
</tr>
<tr>
<td>7.4 Running Tests On Your Individual Plan</td>
<td>194</td>
</tr>
<tr>
<td>7.5 Navigating Bamboo</td>
<td>194</td>
</tr>
<tr>
<td>7.6 Bamboo Agent Properties</td>
<td>194</td>
</tr>
<tr>
<td>7.7 Running Tests From The Command Line</td>
<td>195</td>
</tr>
<tr>
<td>7.8 Helpful Files</td>
<td>195</td>
</tr>
<tr>
<td>8 LBANN Documentation Building</td>
<td></td>
</tr>
<tr>
<td>8.1 Adding Documentation Outside Code</td>
<td>197</td>
</tr>
<tr>
<td>8.2 Making The Build Work</td>
<td>197</td>
</tr>
<tr>
<td>Index</td>
<td>199</td>
</tr>
</tbody>
</table>
The Livermore Big Artificial Neural Network toolkit (LBANN) is an open-source, HPC-centric, deep learning training framework that is optimized to compose multiple levels of parallelism.

LBANN provides model-parallel acceleration through domain decomposition to optimize for strong scaling of network training. It also allows for composition of model-parallelism with both data parallelism and ensemble training methods for training large neural networks with massive amounts of data. LBANN is able to take advantage of tightly-coupled accelerators, low-latency high-bandwidth networking, and high-bandwidth parallel file systems.

LBANN supports state-of-the-art training algorithms such as unsupervised, self-supervised, and adversarial (GAN) training methods in addition to traditional supervised learning. It also supports recurrent neural networks via back propagation through time (BPTT) training, transfer learning, and multi-model and ensemble training methods.

Users are advised to view the Doxygen API Documentation for API information.
1.1 Download

LBANN source code can be obtained from the Github repo.

1.2 Building with Spack

1.2.1 Setup Spack and local base tools

1. Download and install Spack. Additionally setup shell support as discussed here.

   ```bash
   source ${SPACK_ROOT}/share/spack/setup-env.sh
   ```

2. Setup your compiler and external software environment. For example, on LLNL’s LC machines, one might load the following modules:

   - ml gcc/7.3.0 mvapich2/2.3 cuda/10.0.130 # Pascal
   - ml gcc/7.3.1 cuda/9.2.148 spectrum-mpi/rolling-release # Lassen / Sierra

   • Note to unload unwanted modules you can execute `ml` with package names prepended with a dash, e.g.:
     `ml -intel`. To unload all currently loaded modules, use `ml purge`.

3. Optionally, setup your spack environment to take advantage of locally installed tools. Note that unless your spack environment is explicitly told about tools such as cmake, python, mpi, etc. it will install everything that LBANN and all of its dependencies require. This can take quite a long time, but only has to be done once for a given spack repository. Once all of the standard tools are installed, rebuilding LBANN with spack is quite fast.

   • Advice on setting up paths to external installations is beyond the scope of this document, but is covered in the Spack Documentation.

1.2.2 Building & Installing LBANN as a user

**Warning:** This section is still under development and being tested. It contains known issues. This warning will be removed when it is believed to be generally usable.
With Spack setup and installed into your path, it can be used to install the LBANN executables. This approach is appropriate for users that want to train new or existing models using the python front-end.

Note: If your model requires custom layers or data readers, you may need to install LBANN as a developer, which would allow you to modify and recompile the source code.

Here are three easy ways to install LBANN:

- Using the Spack environment method, (e.g., for an x86_64 LLNL LC system with GPU support):

  Note: This method provides a consistent set of dependencies during installation.

```
  cd <path to LBANN repo>/spack_environments/users/llnl_lc/<arch>_cuda/ # where
  →<arch> = x86_64 | ppc64le
  spack install
  spack env loads
  source ./.loads
```

- Building with the latest released versions and GPU support (use the user’s defaults for specifying the compiler, MPI library, etc.):

```
  spack install lbann +gpu +nccl
  ml load lbann
```

- Building with the head of develop branch for lbann, hydrogen and aluminum with GPU support (use the user’s defaults for specifying the compiler, MPI library, etc.):

```
  spack install lbann@develop +gpu +nccl ^hydrogen@develop ^aluminum@master
  ml load lbann
```

There are numerous options for all of these packages. These options can be viewed via commands such as `spack info lbann`. To specify the compiler, one can add options such as `%gcc@7.3.0`. For further information about specifying dependencies, such as the MPI library, please consult the Spack documentation.

### 1.2.3 Building & Installing LBANN as a developer

Developers of LBANN will often need to interact with the source code and/or advanced configuration options for Aluminum, Hydrogen, and LBANN while the other dependencies remain constant. The Spack installation instructions below set up a Spack environment with the remaining dependencies, requiring the developer to build Aluminum, Hydrogen, and LBANN separately, by whatever means they choose.

1. Establish a Spack environment and install software dependencies. Note that there are four environments to pick from along two axes:

   Note: This spack environment has to be setup once each time you create a new build directory.

   1. developers or users
   2. x86_64 and ppc64le

   For example if you are a developer and want to build the inside of the git repo use the following instructions:
1.2. Building with Spack

Note that the environments provided here have a set of external packages and compilers that are installed on an LLNL LC CZ system. Please update these for your system environment. Alternatively, you can create baseline versions of the user-level Spack configuration files and remove the externals and compilers from the spack.yaml file. More details are provided [here](#).

Note that the initial build of all of the standard packages in Spack will take a while.

Note that the Spack module files set the LIBRARY_PATH environment variable. This behavior allows autotools-based builds to pickup the correct libraries but interferes with the way that CMake sets up RPATHs. To correctly establish the RPATH, please unset the variable as noted above, or you can explicitly pass the RPATH fields to CMake using a command such as:

```
cmake -DCMAKE_INSTALL_RPATH=$(sed 's/:/;/g' <<< "$LIBRARY_PATH") \
-DCMAKE_BUILD_RPATH=$(sed 's/:/;/g' <<< "$LIBRARY_PATH") \
...
```

2. Build LBANN locally from source and build Hydrogen and Aluminum using the superbuild. See [here](#) for a list and descriptions of all CMake flags known to LBANN’s “Superbuild” build system. A representative CMake command line that expects LBANN_HOME, LBANN_BUILD_DIR, LBANN_INSTALL_DIR environment variables might be:

```
cd ${LBANN_BUILD_DIR}
cmake \
  -G Ninja \
  -D CMAKE_BUILD_TYPE:STRING=Release \
  -D CMAKE_INSTALL_PREFIX:PATH=${LBANN_INSTALL_DIR} \
  \
  -D LBANN_SB_BUILD_ALUMINUM=ON \
  -D ALUMINUM_ENABLE_MPI_CUDA=OFF \
  -D ALUMINUM_ENABLE_NCCL=ON \
  \
  -D LBANN_SB_BUILD_HYDROGEN=ON \
  -D Hydrogen_ENABLE_ALUMINUM=ON \
  -D Hydrogen_ENABLE_CUDA=ON \
  -D Hydrogen_ENABLE_CUB=ON \
  \
  -D LBANN_SB_BUILD_LBANN=ON \
  -D LBANN_DATATYPE:STRING=float \
  -D LBANN_SEQUENTIAL_INITIALIZATION:BOOL=OFF \
  -D LBANN_WITH_ALUMINUM:BOOL=ON \
  -D LBANN_WITH_CONDUIT:BOOL=ON \
```

(continues on next page)
-D LBANN_WITH_CUDA:BOOL=ON \\ 
-D LBANN_WITH_CUDNN:BOOL=ON \ 
-D LBANN_WITH_NCCL:BOOL=ON \ 
-D LBANN_WITH_NVPROF:BOOL=ON \ 
-D LBANN_WITH_SOFTMAX_CUDA:BOOL=ON \ 
-D LBANN_WITH_TOPO_AWARE:BOOL=ON \ 
-D LBANN_WITH_TBINF=OFF \ 
-D LBANN_WITH_VTUNE=OFF \ 
${LBANN_HOME}/superbuild 

ninja 
ml use ${LBANN_INSTALL_DIR}/etc/modulefiles/
ml load lbann-0.99.0

The complete documentation for building LBANN directly with CMake can be found here.

1.3 Advanced build methods

1.3.1 Building LBANN on OS X

**Warning:** This section is still under development and being tested. It contains known issues. This warning will be removed when it is believed to be generally usable.

Getting Started

**Warning:** If using OSX 10.14 or newer, be sure that /usr/include has been restored. In version 10.14, this may be accomplished by installing /Library/Developer/CommandLineTools/Packages/macOS_SDK_headers_for_macOS_10.14.pkg. If this package is not available, it’s possible command line tools have not been installed; do so by executing xcode-select --install.

Setup Spack and local base tools

To get started follow the general directions on building LBANN to setup spack.

Setup Homebrew

**Note:** Setting up Homebrew only needs to be done once per system.

1. Download and install Homebrew. Setup base development packages. Note that at the moment we use brew to install llvm, open-mpi, scalapack, and cmake.

```
brew install llvm
brew install open-mpi
brew install scalapack
```
Put the brew based clang in your path:

```
export PATH="/usr/local/opt/llvm/bin:$PATH";
```

Install lmmod so that we can use modules to put spack built packages into your path.

```
brew install lmod
brew install luarocks
```

Update your .profile to enable use of modules via lmod

```
source $(brew --prefix lmod)/init/$(basename $SHELL)
```

### Building & Installing LBANN as a developer

1. Establish a Spack environment and install software dependencies.

   **Note:** This spack environment has to be setup once each time you create a new build directory.

   ```
   export LBANN_HOME=/path/to/lbann/git/repo
   export LBANN_BUILD_DIR=/path/to/a/build/directory
   export LBANN_INSTALL_DIR=/path/to/an/install/directory
   cd $({LBANN_BUILD_DIR})
   spack env create -d . $(LBANN_HOME)/spack_environments/developer_release_osx_→spack.yaml
   spack install
   spack env loads # Spack creates a file named loads that has all of the correct←modules
   source loads
   unset LIBRARY_PATH
   ```

2. Build LBANN locally from source and build Hydrogen and Aluminum using the superbuild. See [here](#) for a list and descriptions of all CMake flags known to LBANN’s “Superbuild” build system. A representative CMake command line that expects LBANN_HOME, LBANN_BUILD_DIR, LBANN_INSTALL_DIR environment variables might be:

   ```
   cd ${LBANN_BUILD_DIR}
   ```

(continues on next page)
\-D LBANN_SB_BUILD_HYDROGEN=ON \n-\-D Hydrogen\_ENABLE\_ALUMINUM=ON \n-\-D Hydrogen\_ENABLE\_CUB=OFF \n-\-D Hydrogen\_ENABLE\_CUDA=OFF \n-\-D LBANN_SB_FWD_HYDROGEN\_OpenMP\_CXX\_LIB\_NAMES=omp \n-\-D LBANN_SB_FWD_HYDROGEN\_OpenMP\_CXX\_FLAGS="-fopenmp\=libomp" \n-\-D LBANN_SB_FWD_HYDROGEN\_OpenMP\_omp\_LIBRARY=/usr/local/opt/llvm/lib/libomp. \n→dylib \n\/ \n-\-D LBANN_SB_BUILD\_LBANN=ON \n-\-D LBANN\_DATATYPE\_STRING=float \n-\-D LBANN\_SEQUENTIAL\_INITIALIZATION\_BOOL=OFF \n-\-D LBANN\_WITH\_ALUMINUM\_BOOL=ON \n-\-D LBANN\_WITH\_CONDUIT\_BOOL=ON \n-\-D LBANN\_WITH\_CUDA\_BOOL=OFF \n-\-D LBANN\_WITH\_CUDNN\_BOOL=OFF \n-\-D LBANN\_WITH\_NCCL\_BOOL=OFF \n-\-D LBANN\_WITH\_MVPROF\_BOOL=OFF \n-\-D LBANN\_WITH\_SOFTMAX\_CUDA\_BOOL=OFF \n-\-D LBANN\_WITH\_TOPO\_AWARE\_BOOL=ON \n-\-D LBANN\_WITH\_TBINF=OFF \n-\-D LBANN\_WITH\_VTUNE\_BOOL=OFF \n-\-D LBANN\_SB\_FWD\_LBANN\_HWLOC\_DIR=/usr/local/opt/hwloc \n-\-D LBANN\_SB\_FWD\_LBANN\_OpenMP\_CXX\_LIB\_NAMES=omp \n-\-D LBANN\_SB\_FWD\_LBANN\_OpenMP\_CXX\_FLAGS="-fopenmp\=libomp" \n-\-D LBANN\_SB\_FWD\_LBANN\_OpenMP\_omp\_LIBRARY=/usr/local/opt/llvm/lib/libomp.d\n\n\n\n\n-ninja

1.3.2 Building LBANN with CMake

LBANN uses CMake for its build system and a version newer than or equal to 3.12.0 is required. LBANN development is done primarily on UNIX-based platforms. As such, the build is tested regularly on Linux-based machines, occasionally on OSX, and never on Windows machines.

The CMake build system is available to any users or developers who need a more fine-grained level of control over dependency resolution and/or features of LBANN. The LBANN team has made an effort to expose as many knobs as possible through the Spack package but if something is missing, please open an issue.

It is required that LBANN be built out-of-source. That is, CMake must not be invoked in a directory containing a CMakeLists.

Dependencies

The following packages and tools are required to build LBANN. All packages listed below may be installed using Spack. See the Spack installation instructions for more details on using Spack to build a complete LBANN environment.

The following basic tools are required.
• A C++11-compliant compiler.
• OpenMP, version 3.0 or newer.
• An MPI-3.0 implementation.
• CEREAL is used to handle complex serialization tasks.
• CMake, version 3.12 or newer.

The following LLNL-maintained packages are **required**.

• **Hydrogen** is a fork of the Elemental distributed dense linear-algebra library and it may be installed via Spack using the package name “hydrogen”. If CUDA support is enabled in Hydrogen, LBANN will inherit this support.

The following third-party packages are **required**.

• **CNPY** is used to ingest data in NumPy format. In principle this should be optional, but at time of writing, LBANN will not build without it.
• **OpenCV** is used to preprocess image data. For performance reasons, it is recommend to build OpenCV with JPEG-turbo for JPEG format support.
• **ProtoBuf** is used to express models in a portable format.

The following LLNL-maintained packages are **optional**.

• **Aluminum** is a communication library optimized for machine learning and interaction with GPUs. We cannot recommend its use strongly enough. It can be built using Spack.
• **CONDUIT** is used to ingest structured data produced by scientific simulations.

The following third-party packages are **optional**.

• **CUDA**. The development team currently uses CUDA version 9.2. Building with CUDA support requires that Hydrogen has been built with CUDA support (see below).
• **cuDNN** is required if building LBANN with CUDA support. It is freely available as a binary distribution from NVIDIA.
• **HWLOC**. HWLOC enables LBANN to make certain optimizations based on the hardware topology. Its use is strongly recommended.
• **NVTX**. LBANN supports some improved annotations for NVPROF using NVTX. NVTX is provided as part of the CUDA toolkit.
• **VTune**. LBANN supports some improved annotations for VTune.

**LBANN CMake options**

The following options are exposed in the CMake build system.

• **LBANN_WITH_ALUMINUM** (Default: OFF): Use the Aluminum communication package. This will be set to ON automatically if Hydrogen was built with Aluminum.
• **LBANN_WITH_CNPY** (Default: ON): Build with support for CNPY for reading Numpy data.
• **LBANN_WITH_CONDUIT** (Default: OFF): Build with support for CONDUIT.
• **LBANN_WITH_NVPROF** (Default: OFF): Build with extra annotations for NVPROF.
• **LBANN_WITH_TOPO_AWARE** (Default: ON): Use HWLOC for topology-aware choices.
• **LBANN_WITH_TBINF** (Default: ON): Enable the Tensorboard interface.
• **LBANN_WITH_VTUNE** (Default: OFF): Build with extra annotations for VTune.
• **LBANN_DETERMINISTIC** *(Default: OFF)*: Force as much of the code as possible to be deterministic. This is not a guarantee as certain operations in third-party libraries cannot be forced into a deterministic mode, especially for CUDA-enabled builds.

• **LBANN_SEQUENTIAL_INITIALIZATION** *(Default: OFF)*: Force sequentially consistent initialization of data structures.

• **LBANN_WARNINGS_AS_ERRORS** *(Default: OFF)*: Promote compiler warnings to errors. This should be used by developers only. Developers are encouraged to build with this **ON** prior to merging any code into the repository.

• **LBANN_USE_PROTOBUF_MODULE** *(Default: OFF)*: Search for Protobuf using CMake’s FindProtobuf. cmake module instead of the Protobuf config file. This is useful on platforms with differently architected compute nodes or when the config method is inexplicably failing.

The following variables may also be set:

• **LBANN_DATATYPE** *(Default: float)*: The datatype to use for training. Currently this must be **float** or **double**.

The following variable has been deprecated and removed:

• **LBANN_WITH_CUDA**. The “CUDA-ness” of LBANN is now tied 1:1 with the “CUDA-ness” of Hydrogen. At present, it seems like unnecessary overhead to support the situation in which Hydrogen has CUDA support but LBANN doesn’t want to use it until a compelling use-case reveals itself.

### Controlling dependency resolution

The following variables may be set with CMake to identify dependencies that are not installed into the “typical” locations that CMake searches by default. They may be either exported into the environment used by CMake using whatever mechanisms are allowed by the shell or passed to CMake as a cache variable (e.g., `cmake -DPKG_DIR=/path/to/pkg`). The latter option is recommended.

• **Aluminum_DIR** or **ALUMINUM_DIR** or **AL_DIR**: The path to *either* the Aluminum installation prefix or the AluminumConfig.cmake file. If Hydrogen has not been built with Aluminum support, set **LBANN_WITH_ALUMINUM=ON** to enable Aluminum support.

• **CEREAL_DIR**: The path to *either* the CEREAL installation prefix or the cereal-config.cmake file.

• **CNPY_DIR**: The path to the CNPY installation prefix. Must set **LBANN_WITH_CNPY=ON** to enable CNPY support.

• **CONDUIT_DIR** or **CONDUIT_DIR**: The path to *either* the CONDUIT installation prefix or the ConduitConfig.cmake file. Must set **LBANN_WITH_CONDUIT=ON** to enable CONDUIT support.

• **HDF5_DIR**: The path to *either* the HDF5 installation prefix or the hdf5_config.cmake file. There is a known issue with CONDUIT that it may link to HDF5 but not properly export that dependency.

• **HWLOC_DIR**: The path to the HWLOC installation prefix. Must set **LBANN_WITH_HWLOC=ON** to enable HWLOC support.

• **Hydrogen_DIR** or **HYDROGEN_DIR**: The path to *either* the Hydrogen installation prefix or the HydrogenConfig.cmake file.

• **NVTX_DIR**: The path the the prefix of NVTX. This should not be used except in circumstances in which one might want to link to a different NVTX installation than the CUDA toolkit. Under normal circumstances, if CUDA was found without issue, NVTX should be as well.

• **OpenCV_DIR** or **OPENCV_DIR**: The path to *either* the OpenCV installation prefix or the OpenCVConfig.cmake file.
LBANN Documentation

- **Protobuf_DIR** or **PROTOBUF_DIR**: The path to either the Protobuf installation prefix or the protobuf-config.cmake file.
- **VTUNE_DIR**: The path to the prefix of the VTune (or Intel compiler suite) installation.

Compilers, include CUDA compilers, are found using the default CMake mechanisms, as are OpenMP and MPI. Thus, the process of finding these tools can be manipulated using the usual CMake mechanisms and/or cache variables as documented by CMake.

Except where otherwise noted, this list attempts to address the first level of dependencies of LBANN, that is, those that are one edge away in the DAG. If deeper dependency issues appear, please consult the documentation of the packages that are causing the issues as they may require additional CMake/environment flags to be set before properly resolving.

### Building JAG utilities

The JAG utility executables are not part of the all target. In order to use or install them, they must be built using the jag-utils target. In order to install them, this must be done before installing.

```bash
# Configure LBANN
cmake <see below... or above> /path/to/lbann

# Build main LBANN library and front-ends
cmake --build .

# If JAG utilities are required, build them
cmake --build . --target jag-utils

# Install all (built) targets
cmake --build . --target install
```

### Example CMake invocation

A sample CMake build for LBANN might look like the following.

```bash
cmake \
-D LBANN_WITH_CUDA:BOOL=ON \
-D LBANN_WITH_NVPROF:BOOL=ON \
-D LBANN_DATATYPE:STRING=float \
-D Hydrogen_DIR:PATH=/path/to/hydrogen \
-D HWLOC_DIR:PATH=/path/to/hwloc \
/path/to/lbann
```

### 1.3.3 Building an entire ecosystem with the “Superbuild”

**Warning**: This is primarily for developer convenience and is not meant to be robust to all possible use-cases for LBANN.

LBANN includes CMake `ExternalProject` definitions for a large portion of its dependency graph. The following dependencies are supported. These are one or two edges from LBANN in the dependency DAG.

- Aluminum
- CEREAL

---

1.3. Advanced build methods
• CNPY
• CONDUIT
• CUB. This is used by Hydrogen for efficiently managing GPU memory.
• HDF5. This is a dependency of CONDUIT.
• Hydrogen
• JPEG-turbo. This is a dependency of OpenCV.
• OpenBLAS. This is an optional dependency of Hydrogen. It is recommended if your system does not have a system-optimized BLAS distribution (e.g., Intel’s MKL).
• OpenCV
• Protobuf

The following dependencies are known to exist but for some reason or another are not supported by the superbuild framework.

• cuDNN is a freely available binary package available from NVIDIA.
• NCCL is a freely available binary package available from NVIDIA. Inspired users may also build it from source from its github repository.
• HWLOC is often installed by default, especially on large supercomputers. Certain components may require superuser access to configure, but these features are not used by LBANN. If it is not available, ask the system administrators, consult the package manager, install using Spack, or build from source.

The superbuild system is itself a CMake project rooted in $LBANN_HOME/superbuild (distinct from the LBANN CMake project rooted in $LBANN_HOME). Options that control the superbuild system are prefixed with LBANN_SB_; other options that appear in a CMake invocation for the superbuild are either interpreted on a sub-project basis or forwarded to certain sub-projects.

Choosing packages to build in the Superbuild

The superbuild system is constructive or additive; that is, it will only build the packages that it is asked to build. Any required package that is not requested is assumed to exist on the system by the time it is needed by whichever package requires it. For example, if HDF5 is provided by the system administrators on a system, it does not need to be built and CONDUIT can be built by pointing its build to the system HDF5.

Packages are included in a superbuild by passing LBANN_SB_BUILD_<PKG> options to CMake for each package that it should build, including LBANN itself. E.g.,

```
cmake \
  -DLBANN_SB_BUILD_ALUMINUM=ON \n  -DLBANN_SB_BUILD_HYDROGEN=ON \n  -DLBANN_SB_BUILD_LBANN=ON \n /path/to/lbann/superbuild
```

will invoke the superbuild to build Aluminum, Hydrogen, and LBANN only. Acceptable values for <PKG> are ALUMINUM, CEREAL, CNPY, CONDUIT, CUB, HDF5, HYDROGEN, JPEG_TURBO, OPENCV, PROTOBUF and LBANN.

Forwarding options to sub-projects

The subprojects are largely pre-configured to “do the right thing” for building LBANN. However, there are some variables that users of the superbuild system may need to control. These are exposed as regular CMake options in the
individual projects’ CMakeLists and can be viewed by running, e.g.,

cmake -L superbuild/<pkg>/CMakeLists.txt

Several significant CMake flags are automatically forwarded from the superbuild CMake to subprojects. These are generally “typical” CMake flags (but not all; if something is missing, open please an issue. Some examples are

- `CMAKE_INSTALL_PREFIX`
- `CMAKE_BUILD_TYPE`
- `CMAKE_<LANG>_COMPILER`
- `CMAKE_<LANG>_FLAGS`

To accommodate developers working on edge-cases with these dependencies, any flag may be forwarded to any CMake-built package using the following syntax: `LBANN_SB_FWD_<PKG>_<OPTION>=<VALUE>`. This will result in a cache variable being sent to the CMake command for `<PKG>` with the form

```
-D<OPTION>=<VALUE>
```

The `<OPTION>` may be something specific to `<PKG>` or it may be a CMake flag that is not automatically forwarded. For example, the following CMake invocation would send `CMAKE_INTERPROCEDURAL_OPTIMIZATION` to the `HYDROGEN` package and `SPHINX_DIR` to LBANN:

```
cmake -D LBANN_SB_BUILD_HYDROGEN=ON \
-D LBANN_SB_BUILD_LBANN=ON \
-D LBANN_SB_FWD_HYDROGEN_CMAKE_INTERPROCEDURAL_OPTIMIZATION=ON \
-D LBANN_SB_FWD_LBANN_SPHINX_DIR=/path/to/sphinx \
/path/to/superbuild
```

### Special targets in the Superbuild

Modern shells should be able to tab-complete the names of targets in Makefiles or Ninja files, and IDEs should display all targets interactively. The superbuild should create project-level targets for all of the subprojects; these match the `<PKG>` values noted above. For example, after a successful CMake configuration of the superbuild using the Ninja generator, the command

```
ninja HYDROGEN
```

will build the sub-DAG ending with Hydrogen. If `LBANN_SB_BUILD_LBANN=ON`, `ninja LBANN` is equivalent to `ninja` since `LBANN` depends on all other targets built by the superbuild.

When building on UNIX platforms, the “Unix Makefiles” and “Ninja” generators will have special targets defined for debugging superbuild issues. These targets are `gather-build` and `gather-log`. These create tarballs of the build system files and the execution logs generated for the superbuild or during the superbuild build phase, respectively. The target `gather-all` depends on both of these targets and may be used to generate both tarballs at once. The resulting tarballs are helpful to the build system maintainers for debugging build issues if using the superbuild system.

### A full superbuild example

A full invocation to the superbuild that builds all dependencies might look like the following. This example will use a CUDA-enabled build with Aluminum and CONDUIT support using the currently-load GCC toolset. It assumes that desired flags are stored in `<LANG>_FLAGS` in the environment.

---

1.3. Advanced build methods 13


cmake -GNinja \\
-DCMAKE_BUILD_TYPE=Release \\
-DCMAKE_INSTALL_PREFIX=${PWD}/install \\
-DCMAKE_C_COMPILER=$(which gcc) \\
-DCMAKE_C_FLAGS="$(C_FLAGS)" \\
-DCMAKE_CXX_COMPILER=$(which g++) \\
-DCMAKE_CXX_FLAGS="$(CXX_FLAGS)" \\
-DCMAKE_Fortran_COMPILER=$(which gfortran) \\
-DCMAKE_Fortran_FLAGS="$(Fortran_FLAGS)" \\
-DCMAKE_CUDA_COMPILER=$(which nvcc) \\
-DCMAKE_CUDA_FLAGS="$(CUDA_FLAGS)" \\
-DBUILD_CEREAL=ON \\
-DBUILD_CNPY=ON \\
-DBUILD_CONDUIT=ON \\
-DBUILD_CUDA=ON \\
-DBUILD_HDF5=ON \\
-DBUILD_JPEG_TURBO=ON \\
-DBUILD_OPENBLAS=ON \\
-DBUILD_OPENCV=ON \\
-DBUILD_PROTOBUF=ON \\
-DBUILD_ALUMINUM=ON \\
-DBUILD_HYDROGEN=ON \\
-DBUILD_LBANN=ON \\
-DBUILD_OPENMP=ON \\
-DBUILD_CUDA=ON \\
-DBUILD_NVPROF=ON \\
-DBUILD_TBINF=ON \\
-DBUILD_TOPO_AWARE=ON \\
-DBUILDSEQUENTIAL_INITIALIZATION=OFF \\
-DBUILD_WARNINGS_AS_ERRORS=OFF \\
/path/to/superbuild

Please report any issues with the superbuild on Github, but note that they will be evaluated on a case-by-case basis and may not be fixed in a timely manner or at all if they do not affect the development team. To repeat, the superbuild exists for developer convenience and is not meant to supplant a legitimate package manager.

### 1.3.4 Building LBANN in containers

We currently support Docker and Singularity.

**Warning:** The container builds are not regularly tested. If problems are encountered, please open an issue.
Singularity

First build a Singularity container with the lbann.def file:

```
sudo singularity build --writable lbann.img lbann.def
```

**Note:** Building the image requires root access.

**Note:** `--writable` allows users to make changes inside the container. This is required for LLNL’s LC systems.

This will create a container called lbann.img which can be used to invoke lbann on any system with Singularity and Open-MPI installed.

Customizing Configuration in lbann.def

Singularity is designed to take advantage of underlying HPC resources. The lbann.def file in this directory specifically installs packages necessary for infiniband interconnects (lines 15-19). It builds openmpi outside of the spack step to ensure it is built with infiniband support (lines 37-55). Experienced users should modify these sections to match with the underlying resources they intend to run on. This definition file also builds GCC version 4.9.3, and uses it to build Open-MPI and LBANN (lines 33-35). This is also customized to run on specific LLNL LC resources, and can be modified depending on the users system.

Running LBANN with Singualrity

To run LBANN use mpirun and singularity’s execute command:

```
salloc -N2
mpirun -np 4 singularity exec -B /p:/p lbann.img \
   /lbann/spack_builds/singularity/model_zoo/lbann \
   --model=/lbann/model_zoo/models/lenet_mnist/model_lenet_mnist.prototext \
   --reader=/lbann/model_zoo/data_readers/data_reader_mnist.prototext \
   --optimizer=/lbann/model_zoo/optimizers/opt_adagrad.prototext
```

**Note:** The `--B` Singularity command binds directories from the surrounding filesystem to the container. Be sure to include any necessary files using this command (e.g., model prototext files, datasets, etc). Alternatively, system administrators are capable of allowing a Singularity container to utilize the host’s filesystem. This is done by changing MOUNT HOSTFS in the Singularity config file.

Docker

First build a Docker image with the Dockerfile. From whichever directory contains the Dockerfile:

```
docker build -t dockban .
```

**Note:** The `-t` flag specifies an identifying tag for this image. “dockban” can be changed to any desired tag.
Customizing Configuration in Dockerfile

The Dockerfile container definition is less complicated than its Singularity counterpart. GCC 7.1.0 is built and registered with spack in lines 19-21. Users can change this, as well as LBANN-specific build options in spack (line 22). For example, to add gpu support, a user can add +gpu to this line.

Running LBANN with Docker

This LBANN build also uses Open-MPI. Thus, LBANN can be launched with mpirun here as well. However, this example will just show the single-process invocation.

Start a docker container from the previously created image, and attach to it. Make sure to bind any necessary directories using -v:

```
docker run -it -v $HOME/MNIST:/MNIST dockban
```

Run LBANN as you would outside of a container:

```
./spack_build/docker_build/model_zoo/lbann \
--model=model_zoo/models/lenet_mnist/model_lenet_mnist.prototext \
--reader=model_zoo/data_readers/data_reader_mnist.prototext \
--optimizer=model_zoo/optimizers/opt_sgd.prototext
```

1.3.5 Specific information for Livermore Computing (LC) systems

**Warning:** Many features below make assumptions that users belong to certain groups on LC systems. Any information contained here should not be considered general-purpose and any examples are not expected to work except for certain users on LLNL’s LC systems.

**The build_lbann_lc.sh script**

The build_lbann_lc.sh script in the scripts/ directory is a historical script with logic to choose the “right” compilers and grab all the LC-installed dependencies. It is updated on an ad-hoc basis by the subset of developers who use it and it should not be relied upon as a replacement for other methods described in this guide.

**Warning:** Certain paths through this script require access to a certain linux group on the Livermore Computing machines (LC) at LLNL.

Pre-installed Binary Packages

The LC machines have many instances of cuDNN and NCCL installed in locations shared by the brain group. These may be consistently detected by CMake by export-ing their locations into the shell:

```
export CUDNN_DIR=/usr/WS2/brain/cudnn/cudnn-7.4.2/cuda-10.0_x86_64
export NCCL_DIR=/usr/WS2/brain/nccl2/nccl_2.4.2-1+cuda10.0_x86_64
```

Notice that this is specific to using CUDA 10.0 on an x86_64 LC machine. This is a shortcut around formally passing this location as a cache variable to all relevant CMake projects. The cache method for passing these to the LBANN CMake is:
LBANN will detect NCCL automatically from the Aluminum import; there should be no need to pass `NCCL_DIR` to the LBANN CMake.

The Superbuild, on the other hand, may require both `CUDNN_DIR` and `NCCL_DIR` if building both Aluminum and LBANN. Such an invocation might be:

```bash
cmake -DLBANN_SB_BUILD_ALUMINUM=ON -DALUMINUM_ENABLE_NCCL=ON
  -DLBANN_SB_FWD_ALUMINUM_NCCL_DIR=/usr/WS2/brain/nccl2/nccl_2.4.2-1+cuda10.0_x86_64
  -DLBANN_SB_BUILD_HYDROGEN=ON
  -DLBANN_SB_BUILD_LBANN=ON
  -DLBANN_SB_FWD_CUDNN_DIR=/usr/WS2/brain/cudnn/cudnn-7.4.2/cuda-10.0_x86_64
  <other options>
/path/to/lbann/superbuild
```

### 1.3.6 Setting up Basic Spack Environment

**Note:** These instructions are specific to LLNL’s Livermore Computing (LC) machines x86_64 machines. External users will likely have to modify these paths to be specific to their build platform.

- Copy the following text into `~/.spack/linux/compilers.yaml`:

  ```yaml
  compilers:
  - compiler:
      environment: {}
      extra_rpaths: []
      flags: {}
      modules: []
      operating_system: rhel7
      paths:
        cc: /usr/tce/packages/gcc/gcc-7.3.0/bin/gcc
        cxx: /usr/tce/packages/gcc/gcc-7.3.0/bin/g++
        f77: /usr/tce/packages/gcc/gcc-7.3.0/bin/gfortran
        fc: /usr/tce/packages/gcc/gcc-7.3.0/bin/gfortran
      spec: gcc@7.3.0
      target: x86_64
  - compiler:
      environment: {}
      extra_rpaths: []
      flags: {}
      modules: []
      operating_system: rhel7
      paths:
        cc: /usr/tce/packages/gcc/gcc-7.3.1/bin/gcc
        cxx: /usr/tce/packages/gcc/gcc-7.3.1/bin/g++
        f77: /usr/tce/packages/gcc/gcc-7.3.1/bin/gfortran
        fc: /usr/tce/packages/gcc/gcc-7.3.1/bin/gfortran
      spec: gcc@7.3.1
      target: ppc64le
  ``

- Copy the following text into `~/.spack/linux/packages.yaml`:
packages:
  all:
    compiler: [gcc]

cmake:
  variants: -openssl -ncurses
  paths:
    cmake@3.12.1 arch=linux-rhel7-x86_64: /usr/tce/packages/cmake/cmake-3.12.1

mvapich2:
  buildable: True
  version: [2.3]
  paths:
    mvapich2@2.3@gcc7.3.0 arch=linux-rhel7-x86_64: /usr/tce/packages/mvapich2/
                        →mvapich2-2.3-gcc-7.3.0/

hwloc:
  buildable: False
  version: [2.0.2]
  paths:
    hwloc@2.0.2 arch=linux-rhel7-x86_64: /usr/lib64/libhwloc.so

cuda:
  buildable: False
  version: [9.2.88, 10.0.130]
  paths:
    cuda@10.0.130 arch=linux-rhel7-x86_64: /usr/tce/packages/cuda/cuda-10.0.130
    cuda@9.2.88 arch=linux-rhel-ppc64le: /usr/tce/packages/cuda/cuda-9.2.88/

cudnn:
  buildable: False
  version: [7.4.2]
  paths:
    cudnn@7.4.2 arch=linux-rhel7-x84_64: /usr/workspace/bsb brain/cudnn/cudnn-7.4.2/cuda-10.0_x86_64
    cudnn@7.4.2 arch=linux-rhel-ppc64le: /usr/workspace/bsb brain/cudnn/cudnn-7.4.2/cuda-9.2_ppcle

nccl:
  buildable: False
  version: [2.3]
  paths:
    nccl@2.3 arch=linux-rhel7-x84_64: /usr/workspace/bsb brain/nccl2/nccl_2.3.7-1+cuda10.0_x86_64

spectrum-mpi:
  buildable: False
  version: [rolling-release]
  paths:
    spectrum-mpi@rolling-release %gcc@7.3.1 arch=linux-rhel-ppc64le: /usr/tce/packages/spectrum-mpi/spectrum-mpi-rolling-release-gcc-7.3.1
2.1 Anatomy of an LBANN experiment

2.1.1 Parallelism

LBANN is run under MPI, i.e. with multiple processes that communicate with message passing. This set of processes is subdivided into one or more “trainers.” Conceptually, a trainer owns parallel objects, like models and data readers, and generally operates independently of other trainers.

Comments:

- LBANN targets HPC systems with homogeneous compute nodes and GPU accelerators, which motivates some simplifying assumptions:
  - All trainers have the same number of processes.
  - If GPU acceleration is enabled, each MPI process corresponds to one GPU.
- Processors are block assigned to trainers based on MPI rank.
  - In order to minimize the cost of intra-trainer communication, make sure to map processes to the hardware and network topologies. Typically, this just means choosing a sensible number of processes per trainer, e.g. a multiple of the number of GPUs per compute node.
- Generally, increasing the number of processes per trainer will accelerate computation but require more intra-trainer communication. There is typically a sweet spot where run time is minimized, but it is complicated and sensitive to the nature of the computation, the mini-batch size, the data partitioning scheme, hardware and network properties, the communication algorithms, and myriad other factors.
  - Rule-of-thumb: Configure experiments so that the bulk of run time is taken by compute-bound operations (e.g. convolution or matrix multiplication) and so that each process has enough work to achieve a large fraction of peak performance (e.g. by making the mini-batch size sufficiently large).
- Most HPC systems are managed with job schedulers like Slurm. Typically, users can not immediately access compute nodes but must request them from login nodes. The login nodes can be accessed directly (e.g. via `ssh`), but users are discouraged from doing heavy computation on them.
  - For debugging and quick testing, it’s convenient to request an interactive session (`salloc` or `sxterm` with Slurm).
  - If you need to run multiple experiments or if experiments are not time-sensitive, it’s best to submit a batch job (`sbatch` with Slurm).
  - When running an experiment, make sure you know what scheduler account to charge (used by the scheduler for billing and determining priority) and what scheduler partition to run on (compute nodes on a system are typically subdivided into multiple groups, e.g. for batch jobs and for debugging).
With `salloc`, specify the partition using the `--partition` command-line argument and specify the account using `--account`.

Familiarize yourself with the rules for the systems you use (e.g. the expected work for each partition, time limits, job submission limits) and be a good neighbor.

## 2.1.2 Model components

- **Layer**: A tensor operation, arranged within a directed acyclic graph.
  - During evaluation ("forward prop"), a layer receives input tensors from its parents and sends an output tensor to each child.
  - During automatic differentiation ("backprop"), a layer receives “input error signals” (objective function gradients w.r.t. output tensors) from its children and sends “output error signals” (objective function gradients w.r.t. input tensors) to its parents. If the layer has any associated weight tensors, it will also compute objective function gradients w.r.t. the weight tensors.
  - Most layers require a specific number of parents and children, but LBANN will insert layers into the graph if there is a mismatch and the intention is obvious. For example, if a layer expects one child but has multiple, then a split layer (with multiple output tensors all identical to the input tensor) is inserted. Similarly, if a layer has fewer children than expected, dummy layers will be inserted. However, this does not work if there is any ambiguity. In such cases (common with input and slice layers), it is recommended to manually insert identity layers so that the parent/child relationships are absolutely unambiguous.
  - See `lbann/src/proto/layers.proto` for a full list of supported layers.

- **Weights**: A tensor consisting of trainable parameters, typically associated with one or more layers. A weight tensor owns an initializer to initially populate its values and an optimizer to find values that minimize the objective function.
  - A weight tensor without a specified initializer will use a zero initializer.
  - A weight tensor without a specified optimizer will use the model’s default optimizer.
  - If a layer requires weight tensors and none are specified, it will create the needed weight tensors. The layer will pick sensible initializers and optimizers for the weight tensors. For example, a convolution layer will initialize its kernel tensor with He normal initialization and with the model’s default optimizer.
  - The dimensions of a weight tensor is determined by their associated layers. The user can not set it directly.

- **Objective function**: Mathematical expression that the optimizers will attempt to minimize. It is made up of multiple terms that are added together (possibly with scaling factors).
  - An objective function term can get its value from a scalar-valued layer, i.e. a layer with an output tensor with one entry.

- **Metric**: Mathematical expression that will be reported to the user. This typically does not affect training, but is helpful for evaluating the progress of training. A canonical example for classification problems is classification accuracy.

- **Callback**: Function that is performed at various points during an experiment. Callbacks are helpful for reporting, debugging, and performing advanced training techniques.
  - This is the natural home for experimental training techniques.
  - A common use-case is to export values with the “dump outputs” callback so that the user can perform data post-processing or visualization.
2.1.3 Data readers

**Warning:** The core infrastructure for data readers is slated for significant refactoring, so expect major changes in the future.

Data readers are responsible for managing a data set and providing data samples to models. A data set is comprised of independent data samples, each of which is made up of multiple tensors. For example, a data sample for a labeled image classification problem consists of an image tensor and a one-hot label vector.

**Note:** The data readers are currently hard-coded to assume this simple classification paradigm. Hacks are needed if your data does not match it exactly, e.g. if a data sample is comprised of more than two tensors. The most basic approach is to flatten all tensors and concatenate them into one large vector. The model is then responsible for slicing this vector into the appropriate chunks and resizing the chunks into the appropriate dimensions. Done correctly, this should not impose any additional overhead.

Specifically, data readers and models interact via input layers. Each model must have exactly one input layer and its output tensors are populated by a data reader every mini-batch step. This is typically performed by a background thread pool, so data ingestion will efficiently overlap with other computation, especially if the data reader’s work is IO-bound or if the computation is largely on GPUs.

**Note:** An input layer has an output tensor for each data sample tensor. Since each data sample has two tensors (one for the data and one for the label), it follows that every input layer should have two child layers. To make parent/child relationships unambiguous, we recommend manually creating identity layers as children of the input layer.

Note that layers within a model treat the data for a mini-batch as a single tensor where the leading dimension is the mini-batch size. Thus, corresponding tensors in all data samples must have the same dimensions. The data dimensions must be known from the beginning of the experiment and can not change. However, real data is rarely so consistent and some preprocessing is typically required. See `lbann/src/proto/transforms.proto` for a list of available preprocessing transforms.

2.2 Python frontend

LBANN provides a Python frontend with syntax reminiscent of PyTorch. See a simple implementation of LeNet.

Comments:

- Under-the-hood, the Python frontend is actually a convenience wrapper around the Protobuf frontend. The core infrastructure allows users to configure an experiment and “compiles” it to a Prototext text file.
- The Python interface can only configure and launch experiments. It is not active during an experiment and it does not allow for any dynamic control flow.
- Only Python 3 is supported.

2.2.1 Setup

The `lbann` Python package is installed as part of the LBANN build process. However, it is necessary to update the `PYTHONPATH` environment variable to make sure Python detect it. There are several ways to do this:
2.2.2 Basic usage

A typical workflow involves the following steps:

1. Configuring a Trainer.
2. Configuring LBANN model components (like the graph of Layers) and creating a Model.
   • Classes for model components are automatically generated from the LBANN Protobuf specifications in lbann/src/proto. These files are currently the best source of documentation. Message fields in the Protobuf specification are optional keyword arguments for the corresponding Python class constructor. If a keyword argument is not provided, it is logically zero (e.g. false for Boolean fields and empty for string fields)
3. Configuring the default Optimizer to be used by the Weights objects.
4. Loading in a Protobuf text file describing the data reader.
   • The Python frontend currently does not have good support for specifying data readers. If any data reader properties need to be set programmatically, the user must do it directly via the Protobuf Python API.
5. Launching LBANN by calling run.
   • lbann.run will detect whether the user is currently on a login node or a compute node. If on a login node, a batch job will be submitted to the job scheduler. If on a compute node, LBANN will be run directly on the allocated nodes.
   • A timestamped work directory will be created each time LBANN is run. The default location of these work directories can be set with the environment variable LBANN_EXPERIMENT_DIR.
   • Supported job managers are Slurm and LSF.
   • LLNL users may prefer to use lbann.contrib.lc.launcher.run. This is a wrapper around lbann.run, with defaults and optimizations specifically for LC systems.
2.2.3 A simple example

```python
import lbann

# Construct layer graph
# ----------------------------------

# Input data
input = lbann.Input()
image = lbann.Identity(input)
label = lbann.Identity(input)

# Softmax classifier
y = lbann.FullyConnected(image, num_neurons = 10, has_bias = True)
pred = lbann.Softmax(y)

# Loss function and accuracy
loss = lbann.CrossEntropy([pred, label])
acc = lbann.CrossEntropy([pred, label])

# ----------------------------------

# Setup experiment
# ----------------------------------

# Setup trainer
trainer = lbann.Trainer()

# Setup model
mini_batch_size = 64
num_epochs = 5
model = lbann.Model(mini_batch_size,
    num_epochs,
    layers=lbann.traverse_layer_graph(input),
    objective_function=loss,
    metrics=[lbann.Metric(acc, name='accuracy', unit='%')],
    callbacks=[lbann.CallbackPrint(), lbann.CallbackTimer()])

# Setup optimizer
opt = lbann.SGD(learn_rate=0.01, momentum=0.9)

# Load data reader from prototext
import google.protobuf.text_format
data_reader_proto = lbann.lbann_pb2.LbannPB()
with open('path/to/lbann/model_zoo/data_readers/data_reader_mnist.prototext', 'r') as f:
    google.protobuf.text_format.Merge(f.read(), data_reader_proto)
data_reader_proto = data_reader_proto.data_reader

# Run experiment
lbann.run(trainer, model, data_reader_proto, opt)
```

2.2.4 Useful submodules
lbann.modules

A Module is a pattern of layers that can be applied multiple times in a neural network. Once created, a Module is callable, taking a layer as input and returning a layer as output. They will create and manage Weights internally, so they are convenient for weight sharing between different layers. They are also useful for complicated patterns like RNN cells.

_A possible note of confusion:_ “Modules” in LBANN are similar to “layers” in PyTorch, TensorFlow, and Keras. LBANN uses “layer” to refer to tensor operations, in a similar manner as Caffe.

lbann.models

Several common and influential neural network models are implemented as Modules. They can be used as building blocks within more complicated models.

lbann.proto

The _save_prototext_ function will export a Protobuf text file, which can be fed into the Protobuf frontend.

lbann.onnx

This contains functionality to convert between LBANN and ONNX models. See _python/docs/onnx/README.md_ for full documentation.

2.3 Protobuf frontend (advanced)

The main LBANN driver uses Protobuf text files (sometimes called prototext files) to specify experiments. The Python frontend operates by “compiling” an experiment configuration into a Protobuf text file and passing it into the LBANN driver. Aside from quick debugging, there are very few situations where directly manipulating Protobuf text files is superior to using the Python frontend. In fact, it is possible to use Protobuf’s Python API to programmatically manipulate Protobuf messages, if such fine control is necessary.

In order to fully specify an experiment, the user must provide Protobuf text files for the model, default optimizer, and data reader. These can be provided as three separate files or one unified file. The basic template for running LBANN is

```
<mpi-launcher> <mpi-options> \\
    lbann --prototext=experiment.prototext
```

The LBANN Protobuf format is defined in _src/proto/lbann.proto_. It is important to remember that the default value of a Protobuf field is logically zero (e.g. false for Boolean fields and empty for string fields).
Publications about or related to using LBANN:


  
  – IPDPS’19

  
  – MLHPC’18

  
  – MLHPC’17

  
  – MLHPC’16

  
  – MLHPC’15
Presentations highlighting LBANN and its impact on science applications:

**Note:** Presentations and links to be added

Posters about LBANN and its core algorithms and features:

**Note:** Posters and links to be added

LBANN SOFTWARE ARCHITECTURE AND CLASS OVERVIEW

4.1 Trainers (i.e. execution environment)

A trainer is a collection of compute resources and defines a explicit communication domain. It provides the execution for both the training and inference of a trained model. Once constructed a trainer owns an LBANN comm object that defines both intra- and inter-trainer communication domains. Additionally, a trainer will contain an I/O thread pool that is used to fetch and pre-process data that will be provided to the trainer’s models.

A trainer owns:
- comm object
- I/O thread pool
- One or more models
- Execution context for each model
- In the future, it will also contain the data readers.

4.2 Execution Context

When a model is attached to a trainer the execution context of the training algorithm is stored in an execution_context class (or sub-class) per execution mode. Thus there is one execution context per model and mode that contains all of the state with respect to the training algorithm being applied to the model.

For example it tracks the current:
- step
- execution mode
- epoch
- and a pointer back to the trainer

4.3 Termination Criteria (Pending)

(Pending feature) When a model is going to be trained or evaluated, the termination criteria is specified in an object that is passed into the training algorithm. (Note that this feature is under development, currently the termination criteria is dictated by when the training algorithm executes a fixed number of epochs.)
4.4 Training Algorithms

The training algorithm defines the optimization that is to be applied to the model(s) being trained. Additionally, it can specify how to evaluate the model.

4.5 Model

A model is a collection of operations with dependencies encoded as a directed acyclic graph (DAG). In a typical formulation, these operations form a neural network that will be either trained or used for inference. Each operation in the model is an instance of the layer class. The model is then a collection of layers that perform transformations and mathematical operations on data that is passed between layers. The model’s DAG is executed in topological order. Inside of some layer types are weight matrices that define a trained model. (Note that LBANN should be able to support non-DNN models, but this is a subject for future work.)

Each layer in the graph contains a set of tensors that holds the inputs, computed outputs, gradients with respect to the outputs, and gradients with respect to the inputs. Furthermore, for each layer in the graph with learnable parameters, there is an associated weight tensor that form the learned weights of the model. The model also owns the objective function, since that is integrally tied into the model’s computational graph. Additionally, the model owns both the default optimizer that is used to provide a standard optimizer for the model’s weight tensors. Once each weight tensor is instantiated, it will owns an instance of an optimizer.

The model also owns the max_mini_batch_size that is supported by the model. This is due to the fact that it changes the size and shape of input, output, and gradient tensors. Additionally, the model owns a field that controls if background I/O is allowed for this model and associated data reader.
CHAPTER
FIVE

LBANN API

The LBANN API documentation is almost entirely generated by Doxygen. We encourage developers to view the Doxygen-generated documentation. The API documentation is largely reproduced here (using Breathe) for those who prefer the Sphinx/RTD style. It is laid out following a similar structure to the source code to aid in navigation.

5.1 Documentation of lbann::Al::dummy_backend

class dummy_backend
    Dummy Aluminum backend.

5.2 Documentation of lbann::lbann_comm

class lbann_comm
    Manage communication. This supports separate trainers, each of which are split over potentially several processes. Every trainer is split over the same number of processes. The corresponding processes between trainers are on the “inter-trainer communicator”. You can also do point-to-point or broadcast communication to arbitrary sets of processes.

Public Functions

lbann_comm(int procs_per_trainer = 0, El::mpi::Comm world = El::mpi::COMM_WORLD.GetMPIComm())
    Init communicators for trainers each with procs_per_trainer processes, defaulting to every process in one trainer.

lbann_comm(const lbann_comm&)
    Don’t allow copying; it doesn’t make sense for the communicator.

lbann_comm& operator=(const lbann_comm&)
    Don’t allow assignment; it doesn’t make sense for the communicator.

void split_trainers(int procs_per_trainer)
    Split communicators so each trainer has procs_per_trainer processes. If you call this multiple times, it will invalidate existing grids and communicators.

int get_trainer_rank() const
    Get which trainer this process is in.
int get_rank_in_trainer() const
    Get the rank of this process in its trainer.

int get_rank_in_world() const
    Get my rank in COMM_WORLD.

int get_world_rank(int trainer, int rank) const
    Return the COMM_WORLD rank of the rank'th processor in trainer.

int get_trainer_master() const
    Return the rank of the master process in this trainer.

int get_intertrainer_master() const
    Return the rank of the inter-trainer master process.

int get_world_master() const
    Return the rank of the world master process.

bool am_trainer_master() const
    Return true if this process is the master process in its trainer.

bool am_world_master() const
    Return true if this process is the world master process.

Grid & get_trainer_grid()
    Return a grid to use for this trainer.

const Grid & get_trainer_grid() const
    Return a read-only grid to use for this trainer.

int get_num_trainers() const
    Return the total number of trainers.

int get_procs_per_node() const
    Return the number of processes in a compute node.

int get_procs_in_world() const
    Return the total number of ranks.

int get_rank_in_node() const
    Return the rank of this process within its compute node.

bool is_world_rank_on_node(int rank) const
    Return true if rank (in COMM_WORLD) is on this compute node.

int get_default_threads_per_proc() const
    Get default number of threads per process. This is the number of OpenMP threads to use for parallel
    regions, provided omp_set_num_threads has not been called or the num_threads directive has not been
    provided.

void reset_threads()
    Reset the number of threads per process to the default.

void intertrainer_sum_matrix(AbsMat &mat)
    Perform a sum reduction of mat over the inter-trainer communicator.

void intertrainer_broadcast_matrix(AbsMat &mat, int root)
    Broadcast mat over the inter-trainer communicator starting from root.
template<typename T, bool S = is_instantiated_t::value>
void broadcast (int root, T &val, const El::mpi::Comm &c)
    Broadcast a scalar value over an arbitrary communicator.

template<typename T>
void world_broadcast (int root, T &val)
    World broadcast of a scalar.

template<typename T>
void intertrainer_broadcast (int root, T &val)
    Inter-trainer broadcast of a scalar.

template<typename T>
void trainer_broadcast (int root, T &val)
    Within-trainer broadcast of a scalar.

template<typename T>
void broadcast (const int root, T *data, const int count, const El::mpi::Comm &c)
    Broadcast a buffer over an arbitrary communicator assuming that the buffer space is already allocated.

template<typename T>
void world_broadcast (const int root, T *data, const int count)
    World broadcast of a buffer.

template<typename T>
void intertrainer_broadcast (const int root, T *data, const int count)
    Inter-trainer broadcast of a buffer.

template<typename T>
void trainer_broadcast (const int root, T *data, const int count)
    Within-trainer broadcast of a buffer.

template<typename T>
size_t resize (const int root, std::vector<T> &data, const El::mpi::Comm &c)
    Resize vector<> over an arbitrary communicator to match the one on root.

template<typename T>
void broadcast (const int root, std::vector<T> &data, const El::mpi::Comm &c)
    Broadcast vector<> over an arbitrary communicator; vector<> for non-root processes will be resized as needed.

template<typename T>
void world_broadcast (int root, std::vector<T> &data)
    Broadcast vector<> to world.

template<typename T>
void intertrainer_broadcast (int root, std::vector<T> &data)
    Broadcast vector<> across trainers.

    Broadcast vector<> within trainer; vector<> for non-root processes will be resized as needed.

template<typename T>
void trainer_broadcast (int root, std::vector<T> &data)
    Broadcast vector<> within trainer.

void count_bytes_broadcast (const size_t bytes, const int rank, const int root)
    Keep track of the number of broadcast bytes transmitted and received
template<typename T>
void all_gather (const T *src, int src_count, T *rcv, int rcv_count, const El::mpi::Comm &c)
    Allgather over an arbitrary communicator

template<typename T>
void all_gather (std::vector<T> &src, std::vector<T> &rcv, std::vector<int> &rcv_counts, std::vector<int> &rcv_disp, const El::mpi::Comm &c)
    Allgatherv over an arbitrary communicator; all vectors must be correctly sized prior to entry.

template<typename T>
void trainer_all_gather (std::vector<T> &src, std::vector<T> &rcv, std::vector<int> &rcv_counts, std::vector<int> &rcv_disp)
    Allgatherv over a trainer communicator; all vectors must be correctly sized prior to entry.

template<typename T>
void all_gather (T &src, std::vector<T> &data, const El::mpi::Comm &c)
    Allgather for a single element over an arbitrary communicator; std::vector<T> &data must be correctly sized prior to entry.

template<typename T>
void trainer_all_gather (T &src, std::vector<T> &data)
    Allgather for a single element over the trainer communicator; std::vector<T> &data must be correctly sized prior to entry.

template<typename T>
void trainer_gather (T snd, int root)
    Within-trainer scalar gather (for non-root processes).

template<typename T>
void trainer_gather (T snd, T *rcv)
    Within-trainer scalar gather (for root processes).

template<typename T>
void trainer_gather (T *snd, int count, int root)
    Within-trainer scalar-array gather (for non-root processes).

template<typename T>
void trainer_gather (T *snd, int count, T *rcv)
    Within-trainer scalar-array gather (for root processes).

template<typename T>
void trainer_gatherv (T *snd, int count, int root)
    Within-trainer variable-length-array gather (for non-root processes).

template<typename T>
void intertrainer_gather (T snd, int root)
    Inter-trainer gather (for non-root processes).

template<typename T>
void intertrainer_gather (T snd, std::vector<T> &rcv)
    Inter-trainer gather (for root processes).

template<typename T>
void intertrainer_gather (T *snd, int count, int root)
    Inter-trainer scalar-array gather (for non-root processes).
void intertrainer_gather (T *snd, int count, T *rcv)
    Inter-trainer scalar-array gather (for root processes).

template<typename T>
void gather (T snd, int root, const El::mpi::Comm &c)
    Scalar gather (for non-root processes).

template<typename T>
void gather (T snd, T *rcv, const El::mpi::Comm &c)
    Scalar gather (for root processes).

template<typename T>
void gather (T snd, std::vector<T> &rcv, const El::mpi::Comm &c)
    Scalar gather (for root processes).

template<typename T>
void gather (T *snd, int count, int root, const El::mpi::Comm &c)
    Scalar-array gather (for non-root processes).

template<typename T>
void gather (T *snd, int count, T *rcv, const El::mpi::Comm &c)
    Scalar-array gather (for root processes).

T scatter (int root, const El::mpi::Comm &c)
    Scalar scatter (for non-root processes).

template<typename T>
T scatter (T *snd, const El::mpi::Comm &c)
    Scalar scatter (for root processes).

void intertrainer_reduce (T snd, int root, El::mpi::Op op = El::mpi::SUM)
    Inter-trainer reduce (for non-root processes).

template<typename T>
T intertrainer_reduce (T snd, El::mpi::Op op = El::mpi::SUM)
    Inter-trainer reduce (for root processes).

void trainer_reduce (T snd, int root, El::mpi::Op op = El::mpi::SUM)
    Within-trainer reduce (for non-root processes).

template<typename T>
T trainer_reduce (T snd, El::mpi::Op op = El::mpi::SUM)
    Within-trainer reduce (for root processes).

void trainer_reduce (T *snd, int count, int root, El::mpi::Op op = El::mpi::SUM)
    Within-trainer scalar array reduce (for non-root processes).

void trainer_reduce (T *snd, int count, T *rcv, El::mpi::Op op = El::mpi::SUM)
    Within-trainer scalar array reduce (for root processes).

void reduce (T snd, int root, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
    Scalar reduce (for non-root processes).
template<typename T>
T reduce (T snd, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
Scalar reduce (for root processes).

template<typename T>
void reduce (T *snd, int count, const El::mpi::Comm &c)
Scalar-array reduce (for non-root processes).

template<typename T, El::Device D>
void reduce (T *snd, int count, T *rcv, const El::mpi::Comm &c, El::SyncInfo<D> const &sync_info)
Scalar-array reduce (for root processes).

template<typename T>
T intertrainer_allreduce (T snd, El::mpi::Op op = El::mpi::SUM)
Inter-trainer all-reduce.

template<typename T>
T trainer_allreduce (T snd, El::mpi::Op op = El::mpi::SUM)
Within-trainer all-reduce.

template<typename T>
void trainer_allreduce (T *snd, int count, T *rcv, El::mpi::Op op = El::mpi::SUM)
Scalar array within-trainer all-reduce.

template<typename T>
T allreduce (T snd, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
Scalar allreduce.

template<typename T>
void allreduce (T *data, int count, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
Scalar-array allreduce.

template<typename T>
void allreduce (AbsMat &m, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
Matrix allreduce.

template<typename T>
void allreduce (AbsDistMat &m, const El::mpi::Comm &c, El::mpi::Op op = El::mpi::SUM)
Matrix allreduce.

void nb_allreduce (AbsMat &m, const El::mpi::Comm &c, Al::request &req, El::mpi::Op op = El::mpi::SUM)
Non-blocking matrix allreduce. If LBANN has not been built with Aluminum, then this calls a blocking
matrix allreduce.

void nb_allreduce (AbsDistMat &m, const El::mpi::Comm &c, Al::request &req, El::mpi::Op op = El::mpi::SUM)
Non-blocking matrix allreduce. If LBANN has not been built with Aluminum, then this calls a blocking
matrix allreduce.

template<typename T>
void nb_allreduce (T *data, int count, const El::mpi::Comm &c, Al::request &req, El::mpi::Op op = El::mpi::SUM)
Non-blocking in-place scalar-array allreduce. If LBANN has not been built with Aluminum, then this calls
a blocking allreduce. This currently only supports host pointers (i.e. the MPI backend).
template<typename T>
void wait_all (std::vector<El::mpi::Request<T>> &req)
    Wait for all non-blocking requests to complete.

template<typename T>
void wait (El::mpi::Request<T> &req)
    Wait for a non-blocking request to complete.

void wait (Al::request &req)
    Wait for a non-blocking request to complete.

bool test (Al::request &req)
    Test whether a non-blocking request has completed; true if it has.

void intertrainer_barrier()
    Barrier among the inter-trainer processes.

void trainer_barrier()
    Barrier among processes in this trainer.

global_barrier()
    Barrier among all processes.

void barrier (const El::mpi::Comm &c)
    Barrier on an arbitrary communicator.

template<typename T>
void send (const T *data, int count, int trainer, int rank)
    Send a buffer to rank in trainer.

void nb_send (const T *data, int count, int trainer, int rank, El::mpi::Request<T> &req)
    Corresponding non-blocking sends.

template<typename T>
void recv (T *data, int count, int trainer, int rank)
    Corresponding receive to send.

template<typename T, El::Device D>
void recv (T *data, int count, El::SyncInfo<D> const &syncInfo)
    As above, but receive from anyone.

template<typename T>
void nb_recv (T *data, int count, int trainer, int rank, El::mpi::Request<T> &req)
    Corresponding non-blocking receives.

template<typename T, El::Device D>
void sendrecv (const T *snd, int send_count, int send_trainer, int send_rank, T *rcv, int recv_count,
    int recv_trainer, int recv_rank)
    Send/recv to/from ranks.

template<typename T>
int get_count (int trainer, int rank)
    Determine the size (count) of an incoming message.

size_t get_num_trainer_barriers () const
    Return the number of trainer barriers performed.
size_t get_num_intertrainer_barriers() const
Return the number of inter-trainer barriers performed.

size_t get_num_global_barriers() const
Return the number of global barriers performed.

size_t get_bytes_sent() const
Return the number of bytes sent.

size_t get_bytes_received() const
Return the number of bytes received.

const El::mpi::Comm & get_intertrainer_comm() const
Return the intertrainer communicator.

const El::mpi::Comm & get_trainer_comm() const
Return the trainer communicator.

const El::mpi::Comm & get_world_comm() const
Return the world communicator.

const El::mpi::Comm & get_node_comm() const
Return the communicator for this node.

const El::mpi::Comm & get_packed_group_comm(int num_per_group) const
Return a communicator containing num_per_group processors.

This will attempt to pack processes so that the processes in each group are physically close together on the system.

num_per_group must evenly divide the number of processors in the world.

bool is_rank_node_local(int rank, const El::mpi::Comm & comm) const
Return true if rank (in comm) is on the local node.

void lbann_comm_abort(std::string msg)
throws an lbann_exception

Public Static Functions

static bool is_sendable(const AbsMat & mat)
Return true if mat can be transmitted.

static bool is_sendable(const AbsDistMat & dist_mat)
Return true if the local portion of dist_mat can be transmitted.

5.3 Callback Interface

Callbacks give users information about their model as it is trained. Users can select which callbacks to use during training in their model prototext file.
5.3.1 Documentation of lbann::callback_base

class callback_base
Base class for callbacks during training/testing.

The method of each callback is called at a given point during training or testing by the model. Implement whichever ones you care about. Callbacks may be passed a lbann_summary instance, which they can use to log any relevant information.

Subclassed by lbann::callback::check_dataset, lbann::callback::check_gradients, lbann::callback::check_init, lbann::callback::check_metric, lbann::callback::check_nan, lbann::callback::check_small, lbann::callback::checkpoint, lbann::callback::confusion_matrix, lbann::callback::debug, lbann::callback::debug_io, lbann::callback::dump_error_signals, lbann::callback::dump_gradients, lbann::callback::dump_minibatch_sample_indices, lbann::callback::dump_outputs, lbann::callback::dump_weights, lbann::callback::early_stopping, lbann::callback::gpu_memory_usage, lbann::callback::hang, lbann::callback::imcomm, lbann::callback::learning_rate, lbann::callback::ltfb, lbann::callback::mixup, lbann::callback::monitor_io, lbann::callback::perturb_adam, lbann::callback::perturb_dropout, lbann::callback::print_model_description, lbann::callback::print_statistics, lbann::callback::profiler, lbann::callback::replace_weights, lbann::callback::save_images, lbann::callback::save_model, lbann::callback::summary, lbann::callback::sync_layers, lbann::callback::timeline, lbann::callback::timer, lbann::callback::variable_minibatch

Constructors and destructor

callback_base (int batch_interval = 1)
Initialize a callback with an optional batch interval.

Modifiers

virtual void setup (model *m)
Called once to set up the callback (after all layers are set up).

Callback hooks

virtual void on_setup_end (model *m)
Called at the end of setup.

virtual void on_train_begin (model *m)
Called at the beginning of training.

virtual void on_train_end (model *m)
Called at the end of training.

virtual void on_phase_end (model *m)
Called at the end of every phase (multiple epochs) in a layer-wise model training.

virtual void on_epoch_begin (model *m)
Called at the beginning of each epoch.

virtual void on_epoch_end (model *m)
Called immediate after the end of each epoch.
virtual void on_batch_begin (model *m)
   Called at the beginning of a (mini-)batch.

virtual void on_batch_end (model *m)
   Called immediately after the end of a (mini-)batch.

virtual void on_test_begin (model *m)
   Called at the beginning of testing.

virtual void on_test_end (model *m)
   Called immediately after the end of testing.

virtual void on_validation_begin (model *m)
   Called at the beginning of validation.

virtual void on_validation_end (model *m)
   Called immediately after the end of validation.

virtual void on_forward_prop_begin (model *m)
   Called when a model begins forward propagation.

virtual void on_forward_prop_begin (model *m, Layer *l)
   Called when a layer begins forward propagation.

virtual void on_forward_prop_end (model *m)
   Called when a model ends forward propagation.

virtual void on_forward_prop_end (model *m, Layer *l)
   Called when a layer ends forward propagation.

virtual void on_backward_prop_begin (model *m)
   Called when a model begins backward propagation.

virtual void on_backward_prop_begin (model *m, Layer *l)
   Called when a layer begins backward propagation.

virtual void on_backward_prop_end (model *m)
   Called when a model ends backward propagation.

virtual void on_backward_prop_end (model *m, Layer *l)
   Called when a layer ends backward propagation.

virtual void on_optimize_begin (model *m)
   Called when a model begins optimization.

virtual void on_optimize_begin (model *m, weights *w)
   Called when weights begins optimization.

virtual void on_optimize_end (model *m)
   Called when a model ends optimization.

virtual void on_optimize_end (model *m, weights *w)
   Called when weights ends optimization.

virtual void on_batch_evaluate_begin (model *m)
   Called at the beginning of a (mini-)batch evaluation (validation / testing).

virtual void on_batch_evaluate_end (model *m)
   Called at the end of a (mini-)batch evaluation (validation / testing).
virtual void on_evaluate_forward_prop_begin (model *m)
   Called when a model begins forward propagation for evaluation (validation / testing).

virtual void on_evaluate_forward_prop_begin (model *m, Layer *l)
   Called when a layer begins forward propagation for evaluation (validation / testing).

virtual void on_evaluate_forward_prop_end (model *m)
   Called when a model ends forward propagation for evaluation (validation / testing).

virtual void on_evaluate_forward_prop_end (model *m, Layer *l)
   Called when a layer ends forward propagation for evaluation (validation / testing).

Queries

int get_batch_interval () const
   Return the batch interval.

virtual std::string name () const = 0
   Return this callback’s name.

virtual std::string get_description () const
   Human-readable description.

Documentation of lbann::callback::check_dataset

class check_dataset : public lbann::callback_base
   Save the sample indices for each mini-batch to ordered set. Check to make sure that all samples were properly processed.

Public Functions

void on_forward_prop_end (model *m, Layer *l)
   Called when a layer ends forward propagation.

void on_evaluate_forward_prop_end (model *m, Layer *l)
   Called when a layer ends forward propagation for evaluation (validation / testing).

void on_epoch_end (model *m)
   Called immediate after the end of each epoch.

void on_validation_end (model *m)
   Called immediately after the end of validation.

void on_test_end (model *m)
   Called immediately after the end of testing.

std::string name () const
   Return this callback’s name.
Documentation of lbann::callback::check_gradients

class check_gradients : public lbann::callback_base
Gradient checking callback.

Gradient checking is performed at the end of each execution mode phase. Using a fourth-order finite difference scheme, a numerical partial derivative is computed for every weight parameter. If the numerical derivative differs significantly from the analytical derivative computed during backprop, the gradient check has failed.

Public Functions

check_gradients (std::set<execution_mode> modes = {}, DataType step_size = DataType(0), bool verbose = false, bool error_on_failure = false)

Parameters

• modes: Execution modes with gradient checks. If none are provided, gradient checking is performed for every execution mode.

• step_size: Step size for numerical differentiation (with a step size of zero, the step size is estimated to minimize the numerical error).

• verbose: Whether to print results for each parameter.

• error_on_failure: Whether to throw an exception for large gradient errors.

std::string name () const
Return this callback’s name.

void on_train_end (model *m)
Called at the end of training.

void on_validation_end (model *m)
Called immediately after the end of validation.

void on_test_end (model *m)
Called immediately after the end of testing.

Documentation of lbann::callback::check_init

class check_init : public lbann::callback_base
Verify that every model uses the same initialization.

Public Functions

void on_train_begin (model *m)
Check initializations.

std::string name () const
Return this callback’s name.

Documentation of lbann::callback::check_metric

class check_metric : public lbann::callback_base
Metric checking callback. Checks if a metric value falls within an expected range.
Public Functions

std::string name() const
   Return this callback’s name.

void on_epoch_end(model *m)
   Called immediately after the end of each epoch.

void on_validation_end(model *m)
   Called immediately after the end of validation.

void on_test_end(model *m)
   Called immediately after the end of testing.

Documentation of lbann::callback::check_nan

class check_nan : public lbann::callback_base
   Check matrices for whether they include any NaNs or infs to help debugging. This will kill the rank if such values are discovered.

Public Functions

void on_forward_prop_end(model *m, Layer *l)
   Check that activations are good.

void on_backward_prop_end(model *m, Layer *l)
   Check that error signals are good.

void on_backward_prop_end(model *m)
   Check that gradients are good.

void on_batch_end(model *m)
   Check that weights are good.

std::string name() const
   Return this callback’s name.

Documentation of lbann::callback::check_small

class check_small : public lbann::callback_base
   Check matrices for whether they include any very small values to avoid getting denormalized values. Denormalized values can significantly slow floating point computations. Since we often square values, the check is based on the square root of the smallest floating point value. This will kill the rank if such values are discovered.

Public Functions

void on_forward_prop_end(model *m, Layer *l)
   Check that activations are good.

void on_backward_prop_end(model *m)
   Check that gradients are good.
void on_batch_end (model *m)
Check that weights are good.

std::string name () const
Return this callback’s name.

Documentation of lbann::callback::checkpoint

class checkpoint : public lbann::callback_base
Checkpoint at given interval in given directory.

Public Functions

checkpoint (std::string checkpoint_dir, int checkpoint_epochs, int checkpoint_steps, int checkpoint_secs, std::string per_rank_dir, int ckpt_dist_epochs, int ckpt_dist_steps)
Construct the checkpoint callback.

It may be beneficial to the distributed checkpoints at a higher tempo than the shared checkpoints because they are less expensive.

Parameters

• checkpoint_dir: directory to save checkpoint files
• checkpoint_epochs: interval to checkpoint
• checkpoint_steps: interval to checkpoint
• checkpoint_secs: interval to checkpoint
• per_rank_dir: The directory into which to dump distributed checkpoints
• ckpt_dist_epochs: The frequency of distributed checkpoints in epochs
• ckpt_dist_steps: The frequency of distributed checkpoints in steps

void setup (model *m)
Called once to set up the callback (after all layers are set up).

void on_train_begin (model *m)
Called at the beginning of training.

void on_epoch_end (model *m)
Called immediate after the end of each epoch.

void on_batch_end (model *m)
Called immediately after the end of a (mini-)batch.

void on_validation_end (model *m)
Called immediately after the end of validation.

std::string name () const
Return this callback’s name.
Documentation of \texttt{lbann::callback::confusion\_matrix}

\textbf{class confusion\_matrix : public lbann::callback\_base}

Compute confusion matrix. Confusion matrices are saved in CSV files of the form “\texttt{<prefix><mode>.csv}”. The (i,j)-entry is the proportion of samples with prediction i and label j. The prediction and label layers are assumed to output one-hot vectors for each mini-batch sample.

\textbf{Public Functions}

\begin{verbatim}
std::string name () const
    Return this callback’s name.

void setup (model *m)
    Called once to set up the callback (after all layers are set up).

void on_epoch_begin (model *m)
    Called at the beginning of each epoch.

void on_epoch_end (model *m)
    Called immediate after the end of each epoch.

void on_validation_begin (model *m)
    Called at the beginning of validation.

void on_validation_end (model *m)
    Called immediately after the end of validation.

void on_test_begin (model *m)
    Called at the beginning of testing.

void on_test_end (model *m)
    Called immediately after the end of testing.

void on_batch_end (model *m)
    Called immediately after the end of a (mini-)batch.

void on_batch_evaluate_end (model *m)
    Called at the end of a (mini-)batch evaluation (validation / testing).
\end{verbatim}

Documentation of \texttt{lbann::callback::debug}

\textbf{class debug : public lbann::callback\_base}

Phase specific “printf debugging”.

Print verbose status updates to standard error stream. This callback is useful for “printf debugging.”

Takes a prototext parameter \texttt{phase}: \texttt{train | validate | test | <empty>} if \texttt{<empty>} will print messages for all phases

\textbf{Public Functions}

\begin{verbatim}
dbg (std::set<execution\_mode> modes)
    Constructor.

    If modes is empty, status updates will be printed for all execution modes.
\end{verbatim}

5.3. Callback Interface
std::string name() const
    Return this callback’s name.

void on_batch_begin(model *m)
    Print that a batch is beginning.

void on_batch_end(model *m)
    Print that a batch is ending.

void on_batch_evaluate_begin(model *m)
    Print that a layer’s forward prop is beginning.

void on_batch_evaluate_end(model *m)
    Print that a layer’s forward prop is ending.

void on_forward_prop_begin(model *m, Layer *l)
    Print that a layer’s forward prop is beginning.

void on_forward_prop_end(model *m, Layer *l)
    Print that a layer’s forward prop is ending.

void on_backward_prop_begin(model *m, Layer *l)
    Print that a layer’s backward prop is beginning.

void on_backward_prop_end(model *m, Layer *l)
    Print that a layer’s backward prop is ending.

void on_evaluate_forward_prop_begin(model *m, Layer *l)
    Print that a layer’s backward prop is beginning.

void on_evaluate_forward_prop_end(model *m, Layer *l)
    Print that a layer’s backward prop is ending.

void on_optimize_begin(model *m, weights *w)
    Print that a weights’ optimization step is beginning.

void on_optimize_end(model *m, weights *w)
    Print that a weights’ optimization step is ending.

Documentation of lbann::callback::debug_io

class debug_io : public lbann::callback_base
    Print status updates on where training is.

Public Functions

default constructor (execution_mode phase = execution_mode::invalid, int debug_lvl = 0)
    Debug a particular phase; use invalid to debug every phase.

void on_epoch_begin(model *m)
    Print that a training epoch is being started.

void on_forward_prop_begin(model *m, Layer *l)
    Print that forward prop for a layer is beginning.
void on_validation_begin(model *m)
    Print I/O details at the beginning of validation.

void on_evaluate_forward_prop_begin(model *m, Layer *l)
    Print that an evaluation forward prop is beginning.

void on_test_begin(model *m)
    Print I/O details at the beginning of testing.

void print_fp_start(model *m, generic_input_layer *input)
    Common format for printing I/O stats at the start of a mini-batch

void print_phase_start(model *m, execution_mode mode)
    Common format for printing I/O stats at the start of a phase

std::string name() const
    Return this callback’s name.

**Documentation of lbann::callback::dump_error_signals**

class dump_error_signals : public lbann::callback_base
    Dump gradients w.r.t. inputs to file. After each layer performs a backward prop step, this callback will dump the gradients w.r.t. inputs (the “error signals”) to a human-readable ASCII file. This is slow and produces a lot of output.

**Public Functions**

dump_error_signals(std::string basename = "")
    Constructor.

Parameters
    • basename: The basename for output files.

std::string name() const
    Return this callback’s name.

void on_backward_prop_end(model *m, Layer *l)
    Write error signals to file after each backward prop step.

**Documentation of lbann::callback::dump_gradients**

class dump_gradients : public lbann::callback_base
    Dump gradient matrices to files.

This will dump each hidden layer’s gradient matrix after each minibatch. The matrices are written to files using Elemental’s simple ASCII format. This is not meant for checkpointing, but for exporting gradient matrices for analysis that isn’t easily done in LBANN. Note this dumps matrices during each mini-batch. This will be slow and produce a lot of output.
Public Functions

\textbf{dump\_gradients} \texttt{(std::string basename, int batch\_interval = 1)}

\textbf{Parameters}

\begin{itemize}
  \item \texttt{basename}: The basename for writing files.
  \item \texttt{batch\_interval}: The frequency at which to dump the gradients
\end{itemize}

\textbf{on\_backward\_prop\_end} \texttt{(model *m)}

Called when a model ends backward propagation.

\textbf{name} \texttt{() const}

Return this callback’s name.

Documentation of \texttt{lbann::callback::dump\_minibatch\_sample\_indices}

\textbf{class dump\_minibatch\_sample\_indices: public lbann::callback\_base}

Dump sample indices for each minibatch to files.

This will dump the list of indices from the training / validation / testing data that was processed. Note this dumps vectors during each mini-batch. This will be slow and produce a lot of output.

Public Functions

\textbf{dump\_minibatch\_sample\_indices} \texttt{(std::string basename, int batch\_interval = 1)}

\textbf{Parameters}

\begin{itemize}
  \item \texttt{basename}: The basename for writing files.
  \item \texttt{batch\_interval}: The frequency at which to dump sample indices
\end{itemize}

\textbf{on\_forward\_prop\_end} \texttt{(model *m, Layer *l)}

Called when a layer ends forward propagation.

\textbf{on\_evaluate\_forward\_prop\_end} \texttt{(model *m, Layer *l)}

Called when a layer ends forward propagation for evaluation (validation / testing).

\textbf{name} \texttt{() const}

Return this callback’s name.

Documentation of \texttt{lbann::callback::dump\_outputs}

\textbf{class dump\_outputs: public lbann::callback\_base}

Dump layer output tensors to files.

Saves a file for each output tensor of each selected layer, computed at each mini-batch step. Output files have the form “<model>-<mode>-epoch<$\#$>-step<$\#$>-layer-output<$\#$>.<format>”. This is primarily intended as a debugging tool, although it can be used for inference when performance is not critical.

For NumPy file formats (npy and npz), tensor dimensions are recorded. For text file formats (CSV and TSV), each line contains flattened tensor data corresponding to one mini-batch sample (which is the transpose of the column-major matrix representation we use internally).

CNPY is required to export to NumPy file formats (npy and npz).
**Public Functions**

```cpp
dump_outputs (std::set<std::string> layer_names, std::set<execution_mode> modes, El::Int
batch_interval = 0, std::string directory = "", std::string file_format = "")
```

Construct a callback to dump outputs.

**Parameters**

- `layer_names`: Names of layers with output dumps (default: dump outputs for all layers).
- `modes`: Execution modes with output dumps (default: dump outputs for all modes).
- `batch_interval`: Frequency of output dumps (default: dump outputs at each mini-batch step).
- `directory`: Directory for output files (default: current working directory).
- `file_format`: Output file format. Options are csv, tsv, npy, npz (default: csv).

```cpp
std::string name () const
```

Return this callback’s name.

```cpp
void on_forward_prop_end (model *m, Layer *)
```

Called when a layer ends forward propagation.

```cpp
void on_evaluate_forward_prop_end (model *m, Layer *)
```

Called when a layer ends forward propagation for evaluation (validation / testing).

**Documentation of lbann::callback::dump_weights**

```cpp
class dump_weights : public lbann::callback_base
```

Dump weight matrices to files. This will dump each hidden layer’s weight/bias matrix after specified epoch interval. The matrices are written to files using Elemental’s simple ASCII format. This is not meant for check-pointing, but for exporting weight matrices for analysis that isn’t easily done in LBANN.

**Public Functions**

```cpp
dump_weights (std::string basename, El::Int epoch_interval = 1)
```

**Parameters**

- `basename`: The basename for writing files.

```cpp
void on_train_begin (model *m)
```

Called at the beginning of training.

```cpp
void on_epoch_end (model *m)
```

Called immediately after the end of each epoch.

```cpp
std::string name () const
```

Return this callback’s name.

**Documentation of lbann::callback::early_stopping**

```cpp
class early_stopping : public lbann::callback_base
```

Stop training after validation error stops improving.
Public Functions

**early_stopping** (int64_t *patience)
Continue training until score has not improved for patience epochs.

**on_validation_end**(model *m)
Update validation score and check for early stopping.

std::string **name**() const
Return this callback’s name.

Documentation of lbann::callback::gpu_memory_usage

class gpu_memory_usage : public lbann::callback_base
Callback hooks for printing GPU memory usage.

Public Functions

gpu_memory_usage()
Constructor.

void **on_epoch_begin**(model *m)
Called at the beginning of each epoch.

std::string **name**() const
Return this callback’s name.

Documentation of lbann::callback::hang

class hang : public lbann::callback_base
Hang LBANN as training starts so debuggers can attach. This will cause either a specific rank (in COMM_WORLD) or every rank to hang. Attach to the hung ranks and set the hang flag to false with a debugger to proceed.

Public Functions

**hang**(int rank_to_hang = -1)

Parameters

• rank_to_hang: The rank to hang; -1 for every rank (default).

void **setup**(model *m)
Called once to set up the callback (after all layers are set up).

void **on_train_begin**(model *m)
Hang on train begin.

std::string **name**() const
Return this callback’s name.
Documentation of `lbann::callback::imcomm`

```cpp
class imcomm : public lbann::callback_base
```
Support inter-model communication after each mini-batch to synchronize gradient updates.

**Public Functions**

```cpp
imcomm (comm_type ct = NORMAL, const std::shared_ptr<lbann_summary> &summarizer = nullptr)
Initialize with ct being used for all weights.

imcomm (comm_type ct, std::unordered_set<weights *> weights_list, const std::shared_ptr<lbann_summary> &summarizer = nullptr)
Convenience initialization to do one update type for specific weights. Implies no inter-model updates for other weights.

void set_weights_comm (weights *w, comm_type ct)
Choose comm type ct for weights.

void setup (model *m)
Do initialization for this model.

void on_train_begin (model *m)
Make sure all models have the same weights.

void on_backward_prop_end (model *m)
Do inter-model gradient updates.

std::string name () const
Return this callback’s name.
```

Documentation of `lbann::callback::learning_rate`

```cpp
class learning_rate : public lbann::callback_base
```
Base class for learning rate schedules. Child classes should implement the schedule method to make changes.

**Subclassed by**

- `lbann::callback::adaptive_learning_rate`
- `lbann::callback::drop_fixed_learning_rate`
- `lbann::callback::linear_growth_learning_rate`
- `lbann::callback::optimizerwise_adaptive_learning_rate`
- `lbann::callback::poly_learning_rate`
- `lbann::callback::step_learning_rate`

**Public Functions**

```cpp
learning_rate (std::vector<std::string> weights_names)
Only apply to specific weights.

void setup (model *m)
Do some initialization.

void on_epoch_end (model *m)
Apply global learning rate schedules.

void on_backward_prop_end (model *m)
Apply local/per-optimizer learning rate schedules.
```
Documentation of lbann::callback::adaptive_learning_rate

class adaptive_learning_rate : public lbann::callback::learning_rate

Decrease the learning rate by a fixed proportion when validation error stops improving.

Public Functions

adaptive_learning_rate (size_t patience, float amt)

Decrease the learning rate by amt if accuracy does not improve for patience epochs.

std::string name () const

Return this callback’s name.

Documentation of lbann::callback::drop_fixed_learning_rate

class drop_fixed_learning_rate : public lbann::callback::learning_rate

Decrease learning rate by a fixed amount at fixed times.

Public Functions

drop_fixed_learning_rate (std::vector<size_t> drop_epochs, float amt)

Decrease the learning rate by amt when each epoch in drop_epochs is reached.

std::string name () const

Return this callback’s name.

Documentation of lbann::callback::linear_growth_learning_rate

class linear_growth_learning_rate : public lbann::callback::learning_rate

Linearly increase the learning rate to reach a target value over a fixed number of epochs.

Note This currently assumes every optimizer begins with the same learning rate. This also forces its schedule and will stomp over other changes.

Public Functions

linear_growth_learning_rate (float target, size_t num_epochs)

Linearly increase the learning rate to reach target after num_epochs.

void setup (model *m)

Do some initialization.

std::string name () const

Return this callback’s name.
Documentation of lbann::callback::optimizerwise_adaptive_learning_rate

class optimizerwise_adaptive_learning_rate : public lbann::callback::learning_rate

This implements an adaptive scheme for adjust each optimizer’s learning rate based on the ratio of the norms of its weights and its gradients. See: You et al. “Scaling SGD Batch Size to 32K for ImageNet Training”, 2017.

Public Functions

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::poly_learning_rate

class poly_learning_rate : public lbann::callback::learning_rate

Decrease the learning rate by polynomial policy base_lr*(1 - i_cur/i_max)^p, where base_lr is the initial learning rate, i_cur is the current iteration, i_max is the maximum iteration, and p is a parameter.

Public Functions

void setup (model *m)
    Do some initialization.

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::step_learning_rate

class step_learning_rate : public lbann::callback::learning_rate

Decrease the learning rate by a fixed proportion every X epochs.

Public Functions

step_learning_rate (size_t step, float amt)
    Decrease the learning rate by amt every step epochs.

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::ltfb

class ltfb : public lbann::callback_base

Tournament training.

This is intended to support research into the LTFB algorithm. An outline:

• Divide the computational resources into multiple “trainers” that can operate in parallel.
• Setup a model on each trainer and begin training independently.

5.3. Callback Interface 51
• Periodically launch tournaments to select “good” models. More specifically, trainers partner up and exchange their models. Each trainer evaluates a metric for its local and partner models, using its validation data set. The model with the better score is retained and the other one is discarded.

There are many algorithmic variations to be explored:

• How is data is divvied up amongst the trainers. Is it strictly partitioned, partially shared, or completely replicated?
• What model components are exchanged? Just the trainable weights, or a subset of the weights? Hyperparameters?
• Can this be used to explore model architectures?

Public Types

def communication_algorithm

Inter-trainer communication scheme for LTFB.

The specifics of these algorithms are experimental and will be in flux.

Values:

sendrecv_weights

Directly exchange weights values with sendrecv.

<table>
<thead>
<tr>
<th>Corresponding ranks in partner trainers will iterate through their weights and exchange values with sendrecvs.</th>
</tr>
</thead>
</table>

Notes:
- Requires all models to be identical aside from their weights values, so this is not suitable for hyperparameter or model architecture exploration.
- Optimizer state is not exchanged, so there may be wonky learning behavior immediately after a tournament.
- Optimal if communication performance between ranks is uniform and independent. If intra-trainer communication is fast or if communication performance is sensitive to network traffic, it may be advantageous to gather model data on the trainer master ranks and only perform inter-trainer communication between them.

checkpoint_file

Save and load model data with checkpoint files.

<table>
<thead>
<tr>
<th>Notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports hyperparameter exploration.</td>
</tr>
<tr>
<td>Checkpoint files currently do not store model architecture information, so this is not suitable for model architecture exploration.</td>
</tr>
<tr>
<td>This approach is temporary and experimental, since going through the file system is very suboptimal. When a wire format for model checkpoints is developed, it should be used instead.</td>
</tr>
</tbody>
</table>
Public Functions

`ltfb(EI::Int batch_interval, std::string metric_name, std::set<std::string> weights_names = std::set<std::string>(), bool low_score_wins = false, communication_algorithm comm_algo = communication_algorithm::sendrecv_weights, bool exchange_hyperparameters = false)`

Construct the LTFB callback.

Parameters

- `batch_interval`: Number of training mini-batch steps between tournaments.
- `metric_name`: Metric for tournament evaluation.
- `weights_names`: List of weights to exchange with partner. If empty, then all weights are exchanged.
- `low_score_wins`: Whether low-scoring or high-scoring models survive a tournament.
- `comm_algo`: Inter-trainer communication scheme.
- `summarizer`: The summarizer to use for this callback.

std::string `name()` const

Return this callback’s name.

void `setup(model *)m)`

Called once to set up the callback (after all layers are set up).

void `on_train_begin(model *)m)`

Called at the beginning of training.

void `on_batch_begin(model *)m)`

Called at the beginning of a (mini-)batch.

Public Static Functions

static `communication_algorithm string_to_comm_algo(const std::string &str)`

Convert string to LTFB communication algorithm.

If an empty string is provided, returns `communication_algorithm::sendrecv_weights`.

Documentation of lbann::callback::mixup

class mixup: public lbann::callback_base

Apply mixup to named input layers.

See:


This implementation does mixup within a single batch, per the recommendation within the paper. This approach may create duplicate images, and so uses

```
lambda = max(lambda, 1 - lambda)
```
for the mixing value.

This recommendation comes from https://docs.fast.ai/callbacks.mixup.html

The recommended default alpha (from the paper) is 0.4.

Public Functions

mixup (std::unordered_set<std::string> layers, float alpha)
    Apply mixup to layers named in layers with mixup parameter alpha.

std::string name () const
    Return this callback’s name.

void on_forward_prop_end (model *m, Layer *l)
    Called when a layer ends forward propagation.

Documentation of lbann::callback::monitor_io

class monitor_io : public lbann::callback_base
    Print information on the amount of IO that layers do.

Public Functions

monitor_io (std::vector<std::string> const &layers)
    Only apply to specific layers.

void on_epoch_end (model *m)
    Report how much I/O has occurred per data reader

void on_test_end (model *m)
    Called immediately after the end of testing.

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::perturb_adam

class perturb_adam : public lbann::callback_base
    Hyperparameter exploration with Adam optimizers.

    Goes through the Adam optimizers in a model and perturbs four hyperparameters: the learning rate, $\beta_1$, $\beta_2$, and $\epsilon$. Since these hyperparameters can range over orders of magnitude, the perturbations are performed in log space. More precisely, random values are drawn from normal distributions (with user-provided standard deviations) and added to $\log(\text{learning rate})$, $\log(1 - \beta_1)$, $\log(1 - \beta_2)$, and $\log \epsilon$.

Public Functions

perturb_adam (DataType learning_rate_factor, DataType beta1_factor, DataType beta2_factor, DataType eps_factor = 0, bool perturb_during_training = false, El::Int batch_interval = 1, std::set<std::string> weights_names = std::set<std::string>())

Parameters
• **learning_rate_factor**: Standard deviation of learning rate perturbation (in log space).

• **beta1_factor**: Standard deviation of $\beta_1$ perturbation (in log space).

• **beta2_factor**: Standard deviation of $\beta_2$ perturbation (in log space).

• **eps_factor**: Standard deviation of $\epsilon$ perturbation (in log space).

• **perturb_during_training**: Whether to periodically perturb hyperparameters during training or to only perturb once during setup.

• **batch_interval**: Number of training mini-batch steps between perturbations. Only used if **perturb_during_training** is true.

• **weights_names**: Names of weights with Adam optimizers. If empty, all Adam optimizers in the model are perturbed.

```cpp
std::string name() const
    Return this callback’s name.

void setup(model *m)
    Called once to set up the callback (after all layers are set up).

void on_batch_begin(model *m)
    Called at the beginning of a (mini-)batch.
```

**Documentation of lbann::callback::perturb_dropout**

```cpp
class perturb_dropout : public lbann::callback_base
    Hyperparameter exploration with dropouts.

    Goes through the dropout layers in a model and perturbs keep probability

Public Functions

perturb_dropout(EvalType keep_prob_factor, std::set<std::string> layer_names = std::set<std::string>())

Parameters

• **keep_prob_factor**: Standard deviation of learning rate perturbation (in log space).

• **layer_names**: Names of layers with dropout keep prob to perturb. If empty, all dropout layers in the model are perturbed.

```cpp
std::string name() const
    Return this callback’s name.

void setup(model *m)
    Called once to set up the callback (after all layers are set up).
```

**Documentation of lbann::callback::print_model_description**

```cpp
class print_model_description : public lbann::callback_base
    Print human-readable description of model to standard input.

    Message is printed when the model has finished setup. The description includes information on the model’s layers, weights, and callbacks.
```
Public Functions

void on_setup_end (model *m)
   Called at the end of setup.

std::string name () const
   Return this callback’s name.

Documentation of lbann::callback::print_statistics

class print_statistics : public lbann::callback_base
   Periodically print computational results. Prints average objective function value and metric scores after each training epoch and evaluation.

Public Functions

void setup (model *m)
   Called once to set up the callback (after all layers are set up).

void on_epoch_begin (model *m)
   Called at the beginning of each epoch.

void on_epoch_end (model *m)
   Called immediate after the end of each epoch.

void on_validation_end (model *m)
   Called immediately after the end of validation.

void on_test_end (model *m)
   Called immediately after the end of testing.

std::string name () const
   Return this callback’s name.

Documentation of lbann::callback::profiler

class profiler : public lbann::callback_base

Public Functions

void on_epoch_begin (model *m)
   Called at the beginning of each epoch.

void on_epoch_end (model *m)
   Called immediate after the end of each epoch.

void on_validation_begin (model *m)
   Called at the beginning of validation.

void on_validation_end (model *m)
   Called immediately after the end of validation.
void **on_test_begin** (*model *m*)
Called at the beginning of testing.

void **on_test_end** (*model *m*)
Called immediately after the end of testing.

void **on_batch_begin** (*model *m*)
Called at the beginning of a (mini-)batch.

void **on_batch_end** (*model *m*)
Called immediately after the end of a (mini-)batch.

void **on_batch_evaluate_begin** (*model *m*)
Called at the beginning of a (mini-)batch evaluation (validation / testing).

void **on_batch_evaluate_end** (*model *m*)
Called at the end of a (mini-)batch evaluation (validation / testing).

void **on_forward_prop_begin** (*model *m*)
Called when a model begins forward propagation.

void **on_forward_prop_end** (*model *m*)
Called when a model ends forward propagation.

void **on_evaluate_forward_prop_begin** (*model *m*)
Called when a model begins forward propagation for evaluation (validation / testing).

void **on_evaluate_forward_prop_end** (*model *m*)
Called when a model ends forward propagation for evaluation (validation / testing).

void **on_backward_prop_begin** (*model *m*)
Called when a model begins backward propagation.

void **on_backward_prop_end** (*model *m*)
Called when a model ends backward propagation.

void **on_forward_prop_begin** (*model *m*, **Layer** *l*)
Called when a layer begins forward propagation.

void **on_forward_prop_end** (*model *m*, **Layer** *l*)
Called when a layer ends forward propagation.

void **on_evaluate_forward_prop_begin** (*model *m*, **Layer** *l*)
Called when a layer begins forward propagation for evaluation (validation / testing).

void **on_evaluate_forward_prop_end** (*model *m*, **Layer** *l*)
Called when a layer ends forward propagation for evaluation (validation / testing).

void **on_backward_prop_begin** (*model *m*, **Layer** *l*)
Called when a layer begins backward propagation.

void **on_backward_prop_end** (*model *m*, **Layer** *l*)
Called when a layer ends backward propagation.

void **on_optimize_begin** (*model *m*)
Called when a model begins optimization.

void **on_optimize_end** (*model *m*)
Called when a model ends optimization.
void on_optimize_begin (model *m, weights *w)
    Called when weights begins optimization.

void on_optimize_end (model *m, weights *w)
    Called when weights ends optimization.

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::replace_weights

class replace_weights : public lbann::callback_base
    Weights/parameters replacement on k-batch end. Currently support replacing weights/parameters using layer names. Can easily be extended to support replacement by weights name. Given two layers specified in prototext, weights are copied from source layer to destination layer.

Public Functions

void setup (model *m)
    Called once to set up the callback (after all layers are set up).

void on_batch_end (model *m)
    Called immediately after the end of a (mini-)batch.

std::string name () const
    Return this callback’s name.

Documentation of lbann::callback::save_images

class save_images : public lbann::callback_base
    Save layer outputs as image files. Image files are in the form “<prefix><tag>-<layer name>.<format>”.

Public Functions

save_images (std::vector<std::string> layer_names, std::string image_format = "jpg", std::string image_prefix = "")
    Constructor.

Parameters

    • layer_names: List of layer names to save as images.
    • image_format: Image file format (e.g. jpg, png, pgm).
    • image_prefix: Prefix for image file names.

void on_epoch_end (model *m)
    Called immediately after the end of each epoch.

void on_test_end (model *m)
    Called immediately after the end of testing.

std::string name () const
    Return this callback’s name.
Documentation of lbann::callback::save_model

class save_model : public lbann::callback_base
Save model to as protobuf file and set of weights
Subclassed by lbann::callback::save_topk_models

Public Functions

save_model (std::string dir, bool disable_save_after_training, std::string extension = "prototext")
Parameters
• dir: directory to save model
• disable_save_after_training: Don’t save after training
• extension: file extension e.g., model, state ……

void on_train_end (model *m)
Called at the end of training.

std::string name () const
Return this callback’s name.

Documentation of lbann::callback::save_topk_models

class save_topk_models : public lbann::callback::save_model
Save_topk_models for (e.g., inference and other analysis).

Parameters
• dir: directory to save model
• k: number of models to save, should be less than number of trainers
• metric_nameevaluation: metric for the topk, descending order is default Note: may end up
saving more than k models if multiple models (trainers) have the same metric score

Public Functions

void on_test_end (model *m)
Called immediately after the end of testing.

std::string name () const
Return this callback’s name.

Documentation of lbann::callback::summary

class summary : public lbann::callback_base
Summarize information to Tensorboard using LBANN’s summary interface.
Public Functions

**summary**

```cpp
summary(const std::shared_ptr<lbann_summary> &summarizer, int batch_interval = 1, int mat_interval = 25)
```

**Parameters**

- **summarizer**: The summary object to write to; this callback takes ownership of it.
- **batch_interval**: The frequency with which to summarize
- **mat_interval**: FIXME

```cpp
void on_train_begin(model *m)
```

Called at the beginning of training.

```cpp
void on_batch_end(model *m)
```

Called immediately after the end of a (mini-)batch.

```cpp
void on_epoch_end(model *m)
```

Called immediately after the end of each epoch.

```cpp
void on_test_end(model *m)
```

Called immediately after the end of testing.

```cpp
std::string name() const
```

Return this callback’s name.

**Documentation of lbann::callback::sync_layers**

```cpp
class sync_layers : public lbann::callback_base
```

Synchronize layers after forward and backward prop. Additionally updates layer timing information to account for this. Note that this callback should come before the summarizer callback to report time correctly (otherwise it will be shifted by one mini-batch).

**Public Functions**

```cpp
sync_layers (bool sync_gpus = true, bool sync_mpi = true, bool only_input = false)
```

**Parameters**

- **sync_gpus**: The GPU stream will be synchronized.
- **sync_mpi**: A global barrier will synchronize processes.
- **only_input**: The only synchronization will be after the input layer in forward prop.

```cpp
std::string name() const
```

Return this callback’s name.

```cpp
void on_forward_prop_end(model *m, Layer *l)
```

Called when a layer ends forward propagation.

```cpp
void on_backward_prop_end(model *m, Layer *l)
```

Called when a layer ends backward propagation.
Documentation of lbann::callback::timeline

class timeline : public lbann::callback_base

Record a timeline of training runtime on each rank and output it to a logfile for external processing. The logfile is named timeline.m-model-rank>.<rank>.txt. Each line is a separate event, written as name:start-time:end-time. Times are relative to the beginning of training.

Public Functions

std::string name () const

Return this callback’s name.

void on_train_begin (model *m)

Called at the beginning of training.

void on_train_end (model *m)

Called at the end of training.

void on_forward_prop_begin (model *m, Layer *l)

Called when a layer begins forward propagation.

void on_forward_prop_end (model *m, Layer *l)

Called when a layer ends forward propagation.

void on_backward_prop_begin (model *m, Layer *l)

Called when a layer begins backward propagation.

void on_backward_prop_end (model *m, Layer *l)

Called when a layer ends backward propagation.

void on_optimize_begin (model *m, weights *w)

Called when weights begins optimization.

void on_optimize_end (model *m, weights *w)

Called when weights ends optimization.

Documentation of lbann::callback::timer

class timer : public lbann::callback_base

Record and report model timing results. Reports the total time and mini-batch time statistics for training epochs and for model evaluations. This reports times for the master process in each model.

Public Functions

void on_epoch_begin (model *m)

Start timing for a training epoch.

void on_epoch_end (model *m)

Report timing for a training epoch.

void on_validation_begin (model *m)

Start timing for validation.
void on_validation_end (model *m)
    Report timing for validation.

void on_test_begin (model *m)
    Start timing for testing.

void on_test_end (model *m)
    Report timing for testing.

void on_batch_begin (model *m)
    Record training mini-batch start time.

void on_batch_end (model *m)
    Record training mini-batch run time.

void on_batch_evaluate_begin (model *m)
    Record evaluation mini-batch start time.

void on_batch_evaluate_end (model *m)
    Record evaluation mini-batch run time.

std::string name () const
    Callback name.

Documentation of lbann::callback::variable_minibatch

class variable_minibatch : public lbann::callback_base
    Support changing the mini-batch size on different schedules. Implementations should override implement the abstract methods to define concrete schedules.

    Subclassed by lbann::callback::minibatch_schedule, lbann::callback::step_minibatch

Public Functions

void on_train_begin (model *m)
    Set the initial mini-batch size.

void on_epoch_end (model *m)
    Potentially change the mini-batch size.

Documentation of lbann::callback::minibatch_schedule

class minibatch_schedule : public lbann::callback::variable_minibatch

Public Functions

std::string name () const
    Return this callback’s name.

struct minibatch_step
    Represents a step in a schedule of mini-batch sizes.
**Public Members**

size_t *epoch  
Epoch for this schedule to start.

size_t *mbsize  
Mini-batch size to use.

float *lr  
Learning rate to use.

size_t *ramp_time  
Number of epochs to ramp the learning rate over.

**Documentation of lbann::callback::step_minibatch**

class step_minibatch : public lbann::callback::variable_minibatch  
Double the mini-batch size every set number of epochs. Also doubles the learning rate.

**Public Functions**

std::string name() const  
Return this callback’s name.

---

**5.4 Data Readers Interface**

Data readers provide a mechanism for ingesting data into LBANN. This is typically where a user may have to interact with the LBANN source code.

**5.4.1 Documentation of lbann::generic_compound_data_reader**

class generic_compound_data_reader : public lbann::generic_data_reader  
Data reader for merging the samples from multiple data readers into a single dataset.

Subclassed by lbann::data_reader_merge_features, lbann::data_reader_merge_samples

**Public Functions**

void set_validation_percent (double s)  
Apply operations to subsidiary data readers.

void set_role (std::string role)  
Set an identifier for the dataset. The role should be one of “train”, “test”, or “validate”.

void set_master (bool m)  
only the master may write to cerr or cout; primarily for use in debugging during development

void set_rank (int rank)  
Allow the reader to know where it is in the model hierarchy.

std::vector<generic_data_reader *> & get_data_readers ()  
needed to support data_store_merge_samples
Documentation of \texttt{lbann::data_reader_merge_features}

class \texttt{data_reader_merge_features} : public \texttt{lbann::generic\_compound\_data\_reader}

Data reader for merging multiple data readers. This can take any positive number of data readers, which will be concatenated in the order provided to provide the data, and a single data reader to provide the label. This data reader uses the \texttt{fetch\_datum} method of its subsidiary data readers to fetch all data, including the labels. Label data reader is optional

\textbf{Public Functions}

\begin{verbatim}
std::string \texttt{get\_type} \ () \ \texttt{const} \\
\hspace{1em} Return this data\_reader's type

void \texttt{load} () \\
\hspace{1em} Call load on the subsidiary data readers.

int \texttt{get\_num\_labels} () \ \texttt{const} \\
\hspace{1em} Return the number of labels (classes) in this dataset.

\hspace{1em} This is called at the end of update; it permits data readers to perform actions that are specific to their data sets, for example, \texttt{data\_reader\_jag\_conduit\_hdf5} has the 'primary' data reader broadcast its shuffled indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.

int \texttt{get\_linearized\_data\_size} () \ \texttt{const} \\
\hspace{1em} Get the linearized size (i.e. number of elements) in a sample.

int \texttt{get\_linearized\_label\_size} () \ \texttt{const} \\
\hspace{1em} Get the linearized size (i.e. number of elements) in a label.

const std::vector<int> \texttt{get\_data\_dims} () \ \texttt{const} \\
\hspace{1em} Get the dimensions of the data.
\end{verbatim}

Documentation of \texttt{lbann::data_reader_merge_samples}

class \texttt{data_reader_merge_samples} : public \texttt{lbann::generic\_compound\_data\_reader}

Data reader for merging the samples from multiple data readers into a single dataset.

\textbf{Public Functions}

\begin{verbatim}
std::string \texttt{get\_type} \ () \ \texttt{const} \\
\hspace{1em} Return this data\_reader's type

void \texttt{load} () \\
\hspace{1em} Load subsidiary data readers.

int \texttt{get\_num\_labels} () \ \texttt{const} \\
\hspace{1em} Return the number of labels (classes) in this dataset.

\hspace{1em} This is called at the end of update; it permits data readers to perform actions that are specific to their data sets, for example, \texttt{data\_reader\_jag\_conduit\_hdf5} has the 'primary' data reader broadcast its shuffled indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.
\end{verbatim}
int get_num_responses() const
Return the number of responses in this dataset.

int get_linearized_data_size() const
Get the linearized size (i.e. number of elements) in a sample.

int get_linearized_label_size() const
Get the linearized size (i.e. number of elements) in a label.

int get_linearized_response_size() const
Get the linearized size (i.e. number of elements) in a response.

const std::vector<int> get_data_dims() const
Get the dimensions of the data.

const std::vector<int>& get_num_samples_psum()
support for data store functionality

5.4.2 Documentation of lbann::generic_data_reader

class generic_data_reader
A data reader manages reading in data in a particular format. This abstract base class manages common functionality. In particular, child classes should implement load and the appropriate subset of fetch_datum, fetch_label, and fetch_response.

Subclassed by lbann::csv_reader, lbann::data_reader_jag, lbann::data_reader_jag_conduit, lbann::data_reader_synthetic, lbann::generic_compound_data_reader, lbann::image_data_reader, lbann::mesh_reader, lbann::numpy_npy_conduit_reader, lbann::numpy_npy_reader, lbann::numpy_reader, lbann::pilot2_molecular_reader

Public Functions

generic_data_reader (bool shuffle = true)
cTOR

template<class Archive>
void serialize (Archive &ar)
Archive for checkpoint and restart

void set_comm (lbann_comm *comm)
set the comm object

lbann_comm *get_comm () const
returns a (possibly nullptr) to comm

void set_file_dir (std::string s)
Set base directory for your data.

void set_local_file_dir (std::string s)
Set base directory for your locally cached (e.g, on ssd) data.

void set_max_files_to_load (size_t n)
for some data readers (jag_conduit) we load from multiple files; for testing we want to be able to restrict that number
std::string get_file_dir() const
    Returns the base directory for your data. If set_file_dir was not called, returns the empty string

std::string get_local_file_dir() const
    Returns the base directory for caching files in local ssd If set_local_file_dir was not called, returns the
    empty string

void set_data_index_list (std::string s)
    Set the index list for your data (images, etc). The index lists contains an enumeration of all samples in the
data set.

std::string get_data_index_list () const
    Returns the complete index list for your data set.

void set_data_filename (std::string s)
    Set the filename for your data (images, etc). This may either be a complete filepath, or a subdirectory; see
    note for set_file_dir(). Also, use this method for cases where the file contains a list of files (e.g, imagenet)

std::string get_data_filename () const
    Returns the complete filepath to you data file. See not for set_file_dir()

void set_label_filename (std::string s)
    Set the filename for your data (images, etc). This may either be a complete filepath, or a subdirectory; see
    note for set_file_dir()

std::string get_label_filename () const
    Returns the complete filepath to you data file. See not for set_file_dir(). Note: some pipelines (autoen-
coders) will not make use of this method.

void set_shuffle (bool b)
    If set to false, indices (data samples) are not shuffled default (in ctor) is true.

bool is_shuffled () const
    Returns true if data samples are shuffled.

void set_shuffled_indices (const std::vector<int> &indices)
    Set shuffled indices; primary use is for testing and reproducibility

const std::vector<int> &get_shuffled_indices () const
    Returns the shuffled indices; primary use is for testing.

void set_first_n (int n)
    Read the first 'n' samples. If nonzero, this over-rides set_absolute_sample_count, set_use_percent. The
    intent is to use this for testing. A problem with set_absolute_sample_count and set_use_percent is that the
    entire data set is read in, then a subset is selected

void set_absolute_sample_count (size_t s)
    Sets the absolute number of data samples that will be used for training or testing.

void set_use_percent (double s)
    Set the percentage of the data set to use for training and validation or testing.

Parameters
    • s: The percentage used, in the range [0, 1].

virtual void set_validation_percent (double s)
    Sets the percentage of the dataset to be used for validation.
Parameters

- $s$: The percentage used, in the range $[0, 1]$.

**virtual void** `set_role` (std::string `role`)  
Set an identifier for the dataset. The role should be one of “train”, “test”, or “validate”.

```cpp
std::string get_role () const  
Get the role for this dataset.
```

**virtual void** `load` () = 0  
Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.

**virtual void** `setup` (int `num_io_threads`, observer_ptr<thread_pool> `io_thread_pool`)  
Prepare to start processing an epoch of data. If shuffle is true, then shuffle the indices of the data set If the base offset is not specified set it to 0 If the stride is not specified set it to batch size.

```cpp
virtual std::string get_type () const = 0  
Return this data_reader’s type.
```

**virtual int** `fetch_data` (CPUMat &X, El::Matrix<El::Int> &indices_fetched)  
Fetch this mini-batch’s samples into X.

**virtual int** `fetch_labels` (CPUMat &Y)  
Fetch this mini-batch’s labels into Y.

**virtual int** `fetch_responses` (CPUMat &Y)  
Fetch this mini-batch’s responses into Y.

**virtual bool** `update` (bool `is_active_reader`)  
During the network’s update phase, the data reader will advanced the current position pointer. If the pointer wraps around, then reshuﬄe the data indicies.

**virtual int** `get_num_labels` () const  
Return the number of labels (classes) in this dataset.

This is called at the end of update; it permits data readers to perform actions that are speciﬁc to their data sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader bcast its shuﬄed indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.

**virtual int** `get_num_responses` () const  
Return the number of responses in this dataset.

**virtual int** `get_linearized_data_size` () const  
Get the linearized size (i.e. number of elements) in a sample.

**virtual int** `get_linearized_label_size` () const  
Get the linearized size (i.e. number of elements) in a label.

**virtual int** `get_linearized_response_size` () const  
Get the linearized size (i.e. number of elements) in a response.

**virtual int** `get_linearized_size` (const std::string &`desc`) const  
Get the linearized size of what is identiﬁed by desc.
virtual const std::vector<int> get_data_dims() const
Get the dimensions of the data.

virtual bool position_valid() const
True if the data reader’s current position is valid.

virtual bool position_is_overrun() const
True if the data reader’s current position is not valid but within # ranks per model of the end of the data set
(e.g. it is a rank with no valid data on the last iteration)

bool at_new_epoch() const
True if the data reader is at the start of an epoch.

void set_mini_batch_size(const int s)
Set the mini batch size.

int get_mini_batch_size() const
Get the mini batch size.

int get_loaded_mini_batch_size() const
Get the loaded mini-batch size.

int get_current_mini_batch_size() const
Get the current mini-batch size.

int get_current_global_mini_batch_size() const
Get the current global mini-batch size.

int get_current_world_master_mini_batch_adjustment(int model_rank) const
Get the current mini-batch size.

int get_mini_batch_max() const
Return the full mini_batch_size.

void set_global_mini_batch_size(const int s)
Set the mini batch size across all models (global)

int get_global_mini_batch_size() const
Return the mini_batch_size across all models (global)

void set_stride_to_next_mini_batch(const int s)
Set the mini batch stride.

int get_stride_to_next_mini_batch() const
Return the mini batch stride.

void set_sample_stride(const int s)
Set the sample stride.

int get_sample_stride() const
Return the sample stride.

void set_iteration_stride(const int s)
Set the iteration stride.

int get_iteration_stride() const
Return the iteration stride.
virtual void set_base_offset(const int s)
   Return the base offset.

int get_base_offset() const
   Return the base offset.

void set_model_offset(const int s)
   Set the model offset.

int get_model_offset() const
   Return the model offset.

void set_last_mini_batch_size(const int s)
   Set the last mini batch size.

int get_last_mini_batch_size() const
   Return the last mini batch size.

void set_global_last_mini_batch_size(const int s)
   Set the last mini batch size across all models (global)

int get_global_last_mini_batch_size() const
   Return the last mini batch size across all models (global)

void set_world_master_mini_batch_adjustment(const int s)
   Set the world master mini batch adjustment (global)

int get_world_master_mini_batch_adjustment() const
   Return the world master mini batch adjustment (global)

void set_stride_to_last_mini_batch(const int s)
   Set the last mini batch stride.

int get_stride_to_last_mini_batch() const
   Return the last mini batch stride.

void set_num_parallel_readers(const int s)
   Set the number of parallel readers per model.

int get_num_parallel_readers() const
   Return the number of parallel readers per model.

virtual void set_reset_mini_batch_index(const int s)
   Set the starting mini-batch index for the epoch.

int get_reset_mini_batch_index() const
   Return the starting mini-batch index for the epoch.

int get_loaded_mini_batch_index() const
   Return the current mini-batch index for the epoch.

int get_current_mini_batch_index() const
   Return the current mini-batch index for the epoch.

void set_initial_position()
   Set the current position based on the base and model offsets.

int get_position() const
   Get the current position in the data reader.
int \texttt{get\_next\_position\()\ \textbf{const}\)
Get the next position in the data reader.

int \*\texttt{get\_indices\()\)
Get a pointer to the start of the shuffled indices.

\textbf{virtual} int \texttt{get\_num\_data\()\ \textbf{const}\)
Get the number of samples in this dataset.

int \texttt{get\_num\_unused\_data\()\ \textbf{const}\)
Get the number of unused samples in this dataset.

int \*\texttt{get\_unused\_data\()\)
Get a pointer to the start of the unused sample indices.

void \texttt{set\_num\_iterations\_per\_epoch\(\text{int num\_iterations\_per\_epoch}\\))\}
Set the number of iterations in each epoch.

int \texttt{get\_num\_iterations\_per\_epoch\()\ \textbf{const}\)
Get the number of iterations in each epoch.

int \texttt{get\_current\_step\_in\_epoch\()\ \textbf{const}\)
Return the index of the current iteration step in the epoch (also the mini-batch index)

\textbf{virtual} void \texttt{set\_master\(\text{bool m}\\))\}
only the master may write to cerr or cout; primarily for use in debugging during development

bool \texttt{is\_master\()\ \textbf{const}\)
only the master may write to cerr or cout; primarily for use in debugging during development

\textbf{virtual} void \texttt{set\_rank\(\text{int rank}\\))\}
Allow the reader to know where it is in the model hierarchy.

int \texttt{get\_rank\()\ \textbf{const}\)
Allow the reader to know where it is in the model hierarchy.

void \texttt{resize\_shuffled\_indices\()\)
Optionally resizes the shuffled indices based on the data reader prototext settings: absolute_sample_count,
percent_of_data_to_use. (dah - this was formerly part of select_subset_of_data)

void \texttt{select\_subset\_of\_data\()\)
Select the appropriate subset of data for the validation set based on the data reader prototext setting:
validation_percent

void \texttt{select\_subset\_of\_data\_partitioned\()\)
called by select_subset_of_data() if data set is partitioned

\textbf{virtual} void \texttt{use\_unused\_index\_set\()\)
Replaced the shuffled index set with the unused index set, empying the unused set.

void \texttt{set\_partitioned\(\text{bool is\_partitioned = true, double overlap = 0.0, int mode = 0}\\))\}
partition the dataset amongst the models

bool \texttt{is\_partitioned\()\ \textbf{const}\)
returns true if the data set is partitioned

\textbf{virtual} bool \texttt{has\_list\_per\_model\()\ \textbf{const}\)
Does the data reader have a unique index list per model.
virtual bool has_list_per_trainer() const
  Does the data reader have a unique index list per trainer.

bool save_to_checkpoint_shared(persist &p, execution_mode mode)
  Given directory to store checkpoint files, write state to file and add to number of bytes written.

bool load_from_checkpoint_shared(persist &p, execution_mode mode)
  Given directory to store checkpoint files, read state from file and add to number of bytes read.

bool load_from_checkpoint_distributed(persist &p, execution_mode mode)
  Given directory to store checkpoint files, read state from file and add to number of bytes read.

virtual const data_store_conduit &get_data_store() const
  returns a const ref to the data store

void setup_data_store(int mini_batch_size)
  sets up a data_store; this is called from build_model_from_prototext() in utils/lbann_library.cpp. This is a bit awkward: would like to call it when we instantiate the data_store, but we don't know the mini_batch_size until later.

void set_data_store(data_store_conduit *g)
  support of data store functionality

virtual void post_update()
  experimental; used to ensure all readers for jag_conduit_hdf5 have identical shuffled indices

void set_transform_pipeline(transform::transform_pipeline &&tp)
  Set the transform pipeline this data reader will use.

Documentation of lbann::csv_reader

class csv_reader : public lbann::generic_data_reader
  Data reader for CSV (and similar) files. This will parse a header to determine how many columns of data there are, and will return each row split based on a separator. This does not handle quotes or escape sequences. The label column is by default converted to an integer.

  Note This does not currently support comments or blank lines.

Subclassed by lbann::data_reader_nci

Public Functions

csv_reader (bool shuffle = true)
  This defaults to using the last column for the label/response.

std::string get_type () const
  Return this data_reader's type

void set_label_col (int col)
  Set the label column.

void set_response_col (int col)
  Set the response column.

void disable_labels (bool b = true)
  Disable fetching labels.
void **enable_responses** (bool \( b \) = false)
   Enable fetching responses (disabled by default).

void **set_separator** (char \( sep \))
   Set the column separator (default is ",").

void **set_skip_cols** (int \( cols \))
   Set the number of columns (from the left) to skip; default 0.

void **set_skip_rows** (int \( rows \))
   Set the number of rows (from the top) to skip; default 0.

void **set_has_header** (bool \( b \))
   Set whether the CSV file has a header; default true.

void **set_column_transform** (int \( col \), std::function<DataType const std::string&>
   \( f \))
   Supply a custom transform to convert an input string to a numerical value.

Parameters

- \( col \): The column to apply this transform to; do not account for skipped columns.
- \( f \): The transform to apply.

void **set_label_transform** (std::function<int const std::string&>
   \( f \))
   Supply a custom transform to convert the label column to an integer. Note that the label should be an integer starting from 0.

void **set_response_transform** (std::function<DataType const std::string&>
   \( f \))
   Supply a custom transform to convert the response column to a DataType.

void **load** ()
   This parses the header of the CSV to determine column information.

int **get_num_labels** () const
   Return the number of labels (classes) in this dataset.

   This is called at the end of update; it permits data readers to perform actions that are specific to their data sets, for example, `data_reader_jag_conduit_hdf5` has the ‘primary’ data reader bcast its shuffled indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.

int **get_linearized_data_size** () const
   Get the linearized size (i.e. number of elements) in a sample.

int **get_linearized_label_size** () const
   Get the linearized size (i.e. number of elements) in a label.

const std::vector<int> **get_data_dims** () const
   Get the dimensions of the data.

std::vector<DataType> **fetch_line_label_response** (int \( data_id \))
   Return the parsed CSV line and store the label and response in the m_labels and m_responses vectors, respectively. The label and response are not present in the vector. (Made public to support data store functionality)

**Documentation of lbann::data_reader_nci**

class **data_reader_nci** : public lbann::csv_reader
Public Functions

std::string get_type() const
Return this data_reader’s type

Documentation of lbann::data_reader_jag

class data_reader_jag : public lbann::generic_data_reader
Loads the pairs of JAG simulation inputs and results

Public Types

enum variable_t
Dependent/independent variable types

• JAG_Image: simulation output images
• JAG_Scalar: simulation output scalars
• JAG_Input: simulation input parameters
• Undefined: the default

Values:

Undefined = 0
JAG_Image
JAG_Scalar
JAG_Input

Public Functions

std::string get_type() const
Return this data_reader’s type

void set_independent_variable_type(const std::vector<std::vector<variable_t>> &independent)
Choose which data to use for independent variable.

void set_dependent_variable_type(const std::vector<std::vector<variable_t>> &dependent)
Choose which data to use for dependent variable.

std::vector<variable_t> get_independent_variable_type() const
Tell which data to use for independent variable.

std::vector<variable_t> get_dependent_variable_type() const
Tell which data to use for dependent variable.

void set_normalization_mode(int mode)
Set normalization mode: 0 = none, 1 = dataset-wise, 2 = image-wise.

void set_image_dims(const int width, const int height)
Set the image dimension.
void load()
    // Load data and do data reader’s chores.

std::string get_description() const
    // Show the description.

size_t get_num_samples() const
    // Return the number of samples.

size_t get_linearized_image_size() const
    // Return the linearized size of an image.

size_t get_linearized_scalar_size() const
    // Return the linearized size of scalar outputs.

size_t get_linearized_input_size() const
    // Return the linearized size of inputs.

int get_linearized_data_size() const
    // Get the linearized size (i.e. number of elements) in a sample.

int get_linearized_response_size() const
    // Get the linearized size (i.e. number of elements) in a response.

const std::vector<int> get_data_dims() const
    // Get the dimensions of the data.

data_t* get_image_ptr(const size_t i) const
    // Return the pointer to the raw image data.

scalar_t* get_scalar_ptr(const size_t i) const
    // Return the pointer to the raw scalar data.

std::vector<DataType> get_scalar(const size_t i) const
    // Return the scalar values of the i-th sample.

input_t* get_input_ptr(const size_t i) const
    // Return the pointer to the raw input data.

std::vector<DataType> get_input(const size_t i) const
    // Return the input values of the simulation corresponding to the i-th sample.

**Documentation of lbann::data_reader_jag_conduit**

class data_reader_jag_conduit : public lbann::generic_data_reader
    Loads JAG simulation parameters and results from hdf5 files using conduit interfaces

**Public Types**

enum variable_t
    // Dependent/independent variable types
    
    • JAG_Image: simulation output images
    • JAG_Scalar: simulation output scalars
    • JAG_Input: simulation input parameters
• Undefined: the default

Values:

Undefined = 0
JAG_Image
JAG_Scalar
JAG_Input

using ch_t = float
    jag output image channel type

using conduit_ch_t = conduit::float32_array
    conduit type for ch_t array wrapper

using scalar_t = double
    jag scalar output type

using input_t = double
    jag input parameter type

using sample_locator_t = std::pair<std::string, hid_t>
    Type for the pair of the key string of a sample and the handle of the file that contains it.

using sample_map_t = std::vector<sample_locator_t>
    valid sample map type

using linear_transform_t = std::pair<double, double>
    linear transform on X defined as: first * X + second => X'

using prefix_t = std::pair<std::string, size_t>
    Type to define a prefix string and the minimum length requirement to filter out a key.

Public Functions

void setup (int num_io_threads, observer_ptr<thread_pool> io_thread_pool)
Prepare to start processing an epoch of data. If shuffle is true, then shuffle the indices of the data set If the base offset is not specified set it to 0 If the stride is not specified set it to batch size

std::string get_type () const
    Return this data_reader’s type

void set_independent_variable_type (const std::vector<std::vector<variable_t>>& independent)
    Choose which data to use for independent variable.

void set_dependent_variable_type (const std::vector<std::vector<variable_t>>& dependent)
    Choose which data to use for dependent variable.

std::vector<variable_t> get_independent_variable_type () const
    Tell which data to use for independent variable.

std::vector<variable_t> get_dependent_variable_type () const
    Tell which data to use for dependent variable.

void set_output_scalar_prefix (const std::string &prefix)
    Set the common prefix path for any output scalar fields stored.
void set_output_image_prefix (const std::string &prefix)
    Set the common prefix path for any output images stored.

void set_input_prefix (const std::string &prefix)
    Set the common prefix path for any input variables stored.

void set_image_dims (const int width, const int height, const int ch = 1)
    Set the image dimension.

void set_image_choices (const std::vector<std::string> image_keys)
    Choose images to use. e.g. by measurement views and time indices.

const std::vector<std::string> &get_image_choices () const
    Report the image choices.

void add_scalar_filter (const std::string &key)
    Add a scalar key to filter out.

void add_scalar_prefix_filter (const prefix_t &p)
    Add a scalar key prefix to filter out.

void add_input_filter (const std::string &key)
    Add an input key to filter out.

void add_input_prefix_filter (const prefix_t &p)
    Add an input key prefix to filter out.

void set_scalar_choices (const std::vector<std::string> &keys)
    Select the set of scalar output variables to use.

void set_all_scalar_choices ()
    Set to use the entire set of scalar outputs.

const std::vector<std::string> &get_scalar_choices () const
    Report the selected scalar outputs.

void set_input_choices (const std::vector<std::string> &keys)
    Select the set of simulation input parameters to use.

void set_all_input_choices ()
    Set to use the entire set of simulation input parameters.

const std::vector<std::string> &get_input_choices () const
    Report the selected simulation input parameters.

void load ()
    Load data and do data reader’s chores.

void set_io_buffer_type (const std::string io_buffer)
    Set the type of io_buffer that will rely on this reader.

void set_local_id (const std::string role)
    Set the id of this local instance.

int get_local_id (const std::string role) const
    Get the id of this local instance.
void set_leading_reader(data_reader_jag_conduit *r)
Set the set of open hdf5 data files.
Set the leader of local data reader group

data_reader_jag_conduit *get_leading_reader()
Get the leader of local data reader group.

void set_list_per_trainer(bool flag)
Set every reader instances in a trainer to have an independent index list.

void set_list_per_model(bool flag)
Set every reader instances in a model to have an independent index list.

bool has_list_per_model() const
Does the data reader have a unique index list per model.

bool has_list_per_trainer() const
Does the data reader have a unique index list per trainer.

int fetch_data(CPUMat &X, El::Matrix<El::Int> &indices_fetched)
Fetch data of a mini-batch or reuse it from the cache of the leading reader.

int fetch_responses(CPUMat &Y)
Fetch responses of a mini-batch or reuse it from the cache of the leading reader.

int fetch_labels(CPUMat &Y)
Fetch labels of a mini-batch or reuse it from the cache of the leading reader.

unsigned int get_num_img_srcs() const
Return the number of measurement views.

size_t get_linearized_image_size() const
Return the linearized size of an image.

size_t get_linearized_1ch_image_size() const
Return the linearized size of a single channel image.

size_t get_linearized_scalar_size() const
Return the linearized size of scalar outputs.

size_t get_linearized_input_size() const
Return the linearized size of inputs.

int get_linearized_data_size() const
Return the total linearized size of data.

int get_linearized_response_size() const
Return the total linearized size of response.

std::vector<size_t> get_linearized_data_sizes() const
Return the per-source linearized sizes of composite data.

std::vector<size_t> get_linearized_response_sizes() const
Return the per-source linearized sizes of composite response.

const std::vector<int> get_data_dims() const
Return the dimension of data.
std::vector<El::Int> get_slice_points_independent() const
Return the slice points for linearized independent variables.

std::vector<El::Int> get_slice_points_dependent() const
Return the slice points for linearized dependent variables.

int get_num_data() const
Get the number of samples in this dataset.

int get_num_labels() const
Return the number of labels (classes) in this dataset.

This is called at the end of update; it permits data readers to perform actions that are specific to their data sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader broadcast its shuffled indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.

int get_linearized_label_size() const
Get the linearized size (i.e. number of elements) in a label.

int get_linearized_size(const std::string &desc) const
get the linearized size of what is identified by desc.

std::string get_description() const
Show the description.

std::vector<scalar_t> get_scalars(const size_t i, conduit::Node &sample) const
Return the scalar simulation output data of the i-th sample.

std::vector<input_t> get_inputs(const size_t i, conduit::Node &sample) const
Return the simulation input parameters of the i-th sample.

void print_schema(const size_t i) const
print the schema of the specific sample identified by a given id

Public Static Functions

template<typename S>
size_t add_val(const std::string key, const conduit::Node &n, std::vector<S> &vals)
Retrieve a value from the given node n, and add it to the vector of type S, vals. The first argument key is the name of the current node (i.e. the name reported by the node iterator to the node).

static std::string to_string(const variable_t t)
A utility function to convert a JAG variable type to name string.

Documentation of lbann::data_reader_synthetic

class data_reader_synthetic : public lbann::generic_data_reader
Data reader for generating random samples. Samples are different every time.

Public Functions

std::string get_type() const
Return this data_reader’s type
void load()
    Load the dataset. Each data reader implementation should implement this to initialize its internal data
structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle
samples.

int get_linearized_data_size() const
    Get the linearized size (i.e. number of elements) in a sample.

int get_linearized_label_size() const
    Get the linearized size (i.e. number of elements) in a label.

int get_linearized_response_size() const
    Get the linearized size (i.e. number of elements) in a response.

const std::vector<int> get_dataDims() const
    Get the dimensions of the data.

int get_num_labels() const
    Return the number of labels (classes) in this dataset.

    This is called at the end of update; it permits data readers to perform actions that are specific to their data
sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader bcast its shuffled indices to
the other data readers. In general most data readers will probably not override this method. It may also be
called outside of update.

int get_num_responses() const
    Return the number of responses in this dataset.

Documentation of lbann::image_data_reader

class image_data_reader : public lbann::generic_data_reader
    Subclassed by lbann::cifar10_reader, lbann::imagenet_reader, lbann::mnist_reader

Public Functions

virtual void set_input_params(const int width = 0, const int height = 0, const int num_ch
    = 0, const int num_labels = 0)
    Set up imagenet specific input parameters If argument is set to 0, then this method does not change the
value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

void load()
    Load the dataset. Each data reader implementation should implement this to initialize its internal data
structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle
samples.

void setup(int num_io_threads, observer_ptr<thread_pool> io_thread_pool)
    Prepare to start processing an epoch of data. If shuffle is true, then shuffle the indices of the data set If the
base offset is not specified set it to 0 If the stride is not specified set it to batch size

int get_num_labels() const
    Return the number of labels (classes) in this dataset.

    This is called at the end of update; it permits data readers to perform actions that are specific to their data
sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader bcast its shuffled indices to
the other data readers. In general most data readers will probably not override this method. It may also be
called outside of update.

5.4. Data Readers Interface
```cpp
int get_linearized_data_size() const
    Get the total number of channel values in a sample of image(s).

int get_linearized_label_size() const
    Get the linearized size (i.e. number of elements) in a label.

const std::vector<int> get_data_dims() const
    Get the dimensions of the data.

std::vector<sample_t> get_image_list_of_current_mb() const
    Return the sample list of current minibatch.

const std::vector<sample_t> & get_image_list() const
    Allow read-only access to the entire sample list.

sample_t get_sample(const size_t idx) const
    Returns idx-th sample in the initial loading order. The second argument is only to facilitate overloading, and not to be used by users.
```

### Documentation of lbann::cifar10_reader

**class cifar10_reader : public lbann::image_data_reader**

A data reader for the CIFAR-10/100 datasets.

This requires the binary distributions of the datasets, which must retain their original filenames. CIFAR-10 vs CIFAR-100 is inferred by the number of labels set. See: [https://www.cs.toronto.edu/~kriz/cifar.html](https://www.cs.toronto.edu/~kriz/cifar.html)

**Note** This does not store the coarse labels from CIFAR-100.

### Public Functions

```cpp
std::string get_type() const
    Return this data_reader’s type

void set_input_params(const int width, const int height, const int num_ch, const int num_labels)
    Set up imagenet specific input parameters. If argument is set to 0, then this method does not change the value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

void load()
    Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.
```

### Documentation of lbann::imagenet_reader

**class imagenet_reader : public lbann::image_data_reader**

Subclassed by lbann::data_reader_multi_images

### Public Functions

```cpp
std::string get_type() const
    Return this data_reader’s type
```
Documentation of `lbann::data_reader_multi_images`

```cpp
class data_reader_multi_images : public lbann::imagenet_reader

Subclassed by lbann::data_reader_multihead_siamese, lbann::data_reader_triplet
```

### Public Functions

- **std::string get_type() const**
  Return this data_reader's type

- **void set_input_params(const int width, const int height, const int num_ch, const int num_labels, const int num_img_srcs)**
  Set up imagenet specific input parameters. If argument is set to 0, then this method does not change the value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

- **void set_input_params(const int width, const int height, const int num_ch, const int num_labels)**
  Set up imagenet specific input parameters. If argument is set to 0, then this method does not change the value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

- **void load()**
  Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.

- **int get_linearized_data_size() const**
  Get the total number of channel values in a sample of image(s).

- **const std::vector<int> get_data_dims() const**
  Get the dimensions of the data.

- **std::vector<sample_t> get_image_list_of_current_mb() const**
  Return the sample list of current minibatch.

- **const std::vector<sample_t> &get_image_list() const**
  Allow read-only access to the entire sample list.

- **unsigned int get_num_img_srcs() const**
  The number of image sources or the number of siamese heads. e.g., 2; this method is added to support data_store functionality

Documentation of `lbann::data_reader_multihead_siamese`

```cpp
class data_reader_multihead_siamese : public lbann::data_reader_multi_images
```

### Public Functions

- **std::string get_type() const**
  Return this data_reader’s type

- **void set_input_params(const int width, const int height, const int num_ch, const int num_labels)***
  Set up imagenet specific input parameters. If argument is set to 0, then this method does not change the value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

5.4. Data Readers Interface
void load()
Load the dataset. Each data reader implementation should implement this to initialize its internal data
structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle
samples.

std::vector<sample_t> get_image_list_of_current_mb() const
Return the sample list of current minibatch.

std::vector<sample_t> get_image_list() const
Allow read-only access to the entire sample list.

Documentation of lbann::data_reader_triplet

class data_reader_triplet : public lbann::data_reader_multi_images

Public Functions

std::string get_type() const
Return this data_reader’s type

void set_input_params(const int width, const int height, const int num_ch, const int num_labels)
Set up imagenet specific input parameters If argument is set to 0, then this method does not change the
value of the corresponding parameter. However, width and height can only be both zero or both non-zero.

void load()
Load the dataset. Each data reader implementation should implement this to initialize its internal data
structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle
samples.

std::vector<sample_t> get_image_list_of_current_mb() const
Return the sample list of current minibatch.

std::vector<sample_t> get_image_list() const
Allow read-only access to the entire sample list.

Documentation of lbann::mnist_reader

class mnist_reader : public lbann::image_data_reader

Public Functions

std::string get_type() const
Return this data_reader’s type

void set_input_params(const int width, const int height, const int num_ch, const int num_labels)
Set up imagenet specific input parameters If argument is set to 0, then this method does not change the
value of the corresponding parameter. However, width and height can only be both zero or both non-zero.
void load()
Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.

Documentation of lbann::mesh_reader

class mesh_reader : public lbann::generic_data_reader
Data reader for reading dumped mesh images. Provide the directory containing all the channel subdirectories. This assumes the data is stored as floats in row-major order. The channels to load are currently hardcoded. This only supports regression.

Public Functions

std::string get_type() const
Return this data_reader’s type

void set_suffix(const std::string suffix)
Set a suffix to append to the channel directories.

void set_data_shape(int height, int width)
Set the shape (height and width) of the data.

void set_index_length(int l)
Set the index length for filenames.

void set_random_flips(bool b)
Set whether to do random horizontal and vertical flips.

void load()
Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.

int get_linearized_data_size() const
Get the linearized size (i.e. number of elements) in a sample.

int get_linearized_response_size() const
Get the linearized size (i.e. number of elements) in a response.

const std::vector<int> get_data_dims() const
Get the dimensions of the data.

Documentation of lbann::numpy_npz_conduit_reader

class numpy_npz_conduit_reader : public lbann::generic_data_reader
Data reader for data stored in numpy (.npz) files that are encapsulated in conduit::Nodes.

Public Functions

std::string get_type() const
Return this data_reader’s type
void \texttt{set\_has\_labels} (bool \textcolor{black}{b})
\begin{itemize}
\item Set whether to fetch labels.
\end{itemize}

void \texttt{set\_has\_responses} (bool \textcolor{black}{b})
\begin{itemize}
\item Set whether to fetch responses.
\end{itemize}

void \texttt{set\_scaling\_factor\_int16} (\texttt{DataType \textcolor{black}{s}})
\begin{itemize}
\item Set a scaling factor for int16 data.
\end{itemize}

void \texttt{load} ()
\begin{itemize}
\item Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.
\end{itemize}

\begin{Verbatim}
\textbf{Public Functions}
\end{Verbatim}

\begin{itemize}
\item \texttt{std::string \texttt{get\_type} () \textcolor{black}{const}}
\begin{itemize}
\item Return this data_reader’s type
\end{itemize}
\item \texttt{void \texttt{set\_has\_labels} (bool \textcolor{black}{b})}
\begin{itemize}
\item Set whether to fetch labels.
\end{itemize}
\item \texttt{void \texttt{set\_has\_responses} (bool \textcolor{black}{b})}
\begin{itemize}
\item Set whether to fetch responses.
\end{itemize}
\end{itemize}

\textbf{Documentation of lbann::numpy\_npz\_reader}

class \texttt{numpy\_npz\_reader : public lbann::generic\_data\_reader}
\begin{itemize}
\item Data reader for data stored in numpy (.npz) files. This assumes that the file contains “data”, “labels” (optional), and “responses” (optional) whose the zero’th axis is the sample axis. float, double, int16 data-types is accepted for “data”.
\end{itemize}
void `set_scaling_factor_int16` (DataType `s`)
Set a scaling factor for int16 data.

void `load`()
Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.

int `get_num_labels`() const
Return the number of labels (classes) in this dataset.

This is called at the end of update; it permits data readers to perform actions that are specific to their data sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader broadcast its shuffled indices to the other data readers. In general most data readers will probably not override this method. It may also be called outside of update.

int `get_num_responses`() const
Return the number of responses in this dataset.

int `get_linearized_data_size`() const
Get the linearized size (i.e. number of elements) in a sample.

int `get_linearized_label_size`() const
Get the linearized size (i.e. number of elements) in a label.

int `get_linearized_response_size`() const
Get the linearized size (i.e. number of elements) in a response.

const std::vector<int> `get_data_dims`() const
Get the dimensions of the data.

### Documentation of `lbann::numpy_reader`

**class numpy_reader : public lbann::generic_data_reader**

Data reader for data stored in numpy (.npy) files. This assumes that the zero’th axis is the sample axis and that all subsequent axes can be flattened to form a sample. This supports fetching labels, but only from the last column. (This can be relaxed if necessary.) Ditto responses.

### Public Functions

std::string `get_type`() const
Return this data_reader’s type

void `set_has_labels` (bool `b`)
Set whether to fetch labels.

void `set_has_responses` (bool `b`)
Set whether to fetch responses.

void `load`()
Load the dataset. Each data reader implementation should implement this to initialize its internal data structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle samples.
int get_num_labels() const
    Return the number of labels (classes) in this dataset.

    This is called at the end of update; it permits data readers to perform actions that are specific to their data
    sets, for example, data_reader_jag_conduit_hdf5 has the ‘primary’ data reader broadcast its shuffled indices to
    the other data readers. In general most data readers will probably not override this method. It may also be
    called outside of update.

int get_linearized_data_size() const
    Get the linearized size (i.e. number of elements) in a sample.

int get_linearized_label_size() const
    Get the linearized size (i.e. number of elements) in a label.

const std::vector<int> get_data_dims() const
    Get the dimensions of the data.

---

Documentation of lbann::pilot2_molecular_reader

class pilot2_molecular_reader : public lbann::generic_data_reader
    Data reader for loading Pilot 2 molecular data.

Public Functions

std::string get_type() const
    Return this data_reader’s type

void load()
    Load the dataset. Each data reader implementation should implement this to initialize its internal data
    structures, determine the number of samples and their dimensionality (if needed), and set up and shuffle
    samples.

int get_linearized_data_size() const
    Get the linearized size (i.e. number of elements) in a sample.

const std::vector<int> get_data_dims() const
    Get the dimensions of the data.

template<class T>
    T scale_data(int idx, T datum)
        Data format is: [Frames (2900), Molecules (3040), Beads (12), ['x', 'y', 'z', 'CHOL', 'DPPC', 'DIPC',
        'Head', 'Tail', 'BL1', 'BL2', 'BL3', 'BL4', 'BL5', 'BL6', 'BL7', 'BL8', 'BL9', 'BL10', 'BL11', 'BL12']
        (20)]

float *get_features_4()
    support for data_store_pilot2_molecular

int get_word_size() const
    support for data_store_pilot2_molecular

int get_num_neighbors() const
    support for data_store_pilot2_molecular

int get_frame(int data_id) const
    Return the frame data_id is in. (made public to support data_store_pilot2_molecular)
int get_num_samples_per_frame() const
    support for data_store_pilot2_molecular

int get_max_neighborhood() const
    support for data_store_pilot2_molecular

int get_num_features() const
    support for data_store_pilot2_molecular

int get_neighbors_data_size()
    support for data_store_pilot2_molecular

5.4.3 Documentation of lbann::sample_list_open_files

template<typename sample_name_t, typename file_handle_t>
class sample_list_open_files : public lbann::sample_list<sample_name_t>

Public Types

template<>
using sample_file_id_t = std::size_t
    The type for the index assigned to each sample file.

template<>
using sample_t = std::pair<sample_file_id_t, sample_name_t>
    To describe a sample as a pair of the file to which it belongs and its name Each file may contain multiple
    samples.

template<>
using file_id_stats_t = std::tuple<std::string, file_handle_t, std::deque<std::pair<int, int>>>>
    Information for each file used by the sample list: includes the file name, file descriptor, and and a queue of
    each step and substep when data will be loaded from the file

template<>
using samples_t = std::template vector<sample_t>
    Type for the list of samples.

template<>
using file_id_stats_v_t = std::vector<file_id_stats_t>
    Mapping of the file index to the statistics for each file.

template<>
using fd_use_map_t = std::template pair<sample_file_id_t, std::pair<int, int>>>
    Type for the map of file descriptors to usage step and substep.

Public Functions

sample_list_open_files(const sample_list_open_files &rhs)
    Copy constructor replicates all the member variables as they are except the file information vector, for
    which only the file name is copied.

sample_list_open_files<sample_name_t, file_handle_t> &operator= (const sample_list_open_files &rhs)
    Assignment operation replicates all the member variables as they are except the file information vector, for
    which only the file name is copied.
size_t size() const
    Tells how many samples in the list.

size_t get_num_files() const
    Tells how many sample files are there.

template<class Archive>
void save(Archive & ar) const
    Emit a serialized archive using the cereal library.

template<class Archive>
void load(Archive & ar)
    Restore the member variables from a given archive serialized by the cereal library.

bool to_string(std::string & sstr) const
    Serialize this sample list into an std::string object.

const sample_list_open_files<sample_name_t, file_handle_t>::samples_t &
    get_list() const
    Allow read-only access to the internal list data.

const sample_list_open_files<sample_name_t, file_handle_t>::sample_t &
    operator[](size_t idx) const
    Allow read-only access to the metadata of the idx-th sample in the list.

5.4.4 Documentation of lbann::offline_patches_npz

class offline_patches_npz
    Loads the list of patche files, generated off-line, and the label per sample. As the list is quite large itself in the ASCII text format, it is packed and loaded as a compressed NumPy file (*.npz). Each image file name is compressed further by representing it as a sequence of indices to common substring dictionaries. There are two types of substring dictionaries, root and variant. There is an array of index sequences and an array of dictionary substrings per type, and a label array. For example, a file path train/n000111/abc.tag1.tag2.jpg would be represented as ‘r[i][j][k]’, ‘v[i][j][x]’, ‘v[i][j][y]’, ‘v[i][j][z]’ for the j-th patch of the i-th sample where ‘r[i][j][k]’ is “train/n000111”, and ‘v[i][j][x]’, ‘v[i][j][y]’ and ‘v[i][j][z]’ is “abc”, “tag1”, and “tag2” respectively. ‘r’ is the root dictionary and ‘v’ is the variant dictionary. The list is kept in a compressed form, and uncompressed on-demand during execution. Each index sequence array is kept as a CNPY data structure, and each dictionary array is loaded into a vector of strings. The label array is loaded into a vector of uint8_t.

Public Functions

bool load(const std::string filename, size_t first_n = 0u, bool keep_file_lists = false)
    Load the data in the compressed numpy format file. Use only first_n available samples if specified. keep_file_lists indicates whether to remove file lists loaded once converting them to vector of strings. Need to keep it for selecting a range of samples afterwards.

std::string get_description() const
    Show the description.

size_t get_num_samples() const
    Return the number of samples.

size_t get_num_patches() const
    Return the number of patches per sample (the number of image data sources)
void set_num_patches (size_t npatches)
    Set the number of patches per sample (the number of image data sources)

sample_t get_sample (const size_t idx) const
    Reconstruct and return the meta-data (patch file names and the label) of idx-th sample.

label_t get_label (const size_t idx) const
    Return the label of idx-th sample.

5.4.5 Documentation of lbann::sample_list

template< typename sample_name_t>
class sample_list
    Subclassed by lbann::sample_list_open_files< sample_name_t, file_handle_t >,
    lbann::sample_list_open_files< sample_name_t, conduit::relay::io::IOHandle* >,
    lbann::sample_list_open_files< sample_name_t, hid_t >

Public Types

template<>
using sample_file_id_t = std::size_t
    The type for the index assigned to each sample file.

template<>
using sample_t = std::template pair<sample_file_id_t, sample_name_t>
    To describe a sample as the id of the file to which it belongs. Each file contains only one sample.

template<>
using samples_t = std::template vector<sample_t>
    Type for the list of samples.

template<>
using file_id_stats_v_t = std::vector<std::string>
    Mapping of the file index to the filename.

Public Functions

void load (const std::string &samplelist_file, size_t stride = 1, size_t offset = 0)
    Load a sample list file.

sample_list_header load_header (const std::string &samplelist_file) const
    Load the header of a sample list file.

void load_from_string (const std::string &samplelist)
    Restore a sample list from a serialized string.

size_t size () const
    Tells how many samples in the list.

size_t get_num_files () const
    Tells how many sample files are there.

bool empty () const
    Tells if the internal list is empty.
template<class Archive>
void serialize(Archive &ar)
    Serialize to and from an archive using the cereal library.

bool to_string (std::string &sstr) const
    Serialize sample list.

void write (const std::string filename) const
    Write the sample list.

const sample_list<sample_name_t>::samples_t &get_list () const
    Allow read-only access to the internal list data.

const sample_list_header &get_header () const
    Allow the read-only access to the list header.

const sample_list<sample_name_t>::sample_t &operator[](size_t idx) const
    Allow read-only access to the metadata of the idx-th sample in the list.

5.5 Data Store Interface

The data store provides in-memory caching of the data set and inter-epoch data shuffling.

5.5.1 Documentation of lbann::data_store_conduit

class data_store_conduit

    Public Functions

data_store_conduit (generic_data_reader *reader)
    ctor

data_store_conduit (const data_store_conduit&)
    copy ctor

data_store_conduit (const data_store_conduit&,
                    const std::vector<int>&)
    copy / split ctor

data_store_conduit &operator= (const data_store_conduit&)
    operator=

~data_store_conduit ()
    dtor

void set_data_reader_ptr (generic_data_reader *reader)
    required when the copy ctor is used to construct a validation set

void set_shuffled_indices (const std::vector<int> *indices)
    convenience handle

size_t get_num_global_indices () const
    Returns the number of samples summed over all ranks.
const conduit::Node &get_conduit_node(int data_id) const
returns the conduit node.

void set_conduit_node(int data_id, conduit::Node &node, bool already_have = false)
if 'already_have = true' then the passed 'node' was obtained by a call to get_empty_node(). In some
operating modes this saves us from copying the node.

c conduit::Node &get_empty_node(int data_id)
returns an empty node.

void build_preloaded_owner_map(const std::vector<int> &per_rank_list_sizes)
fills in m_owner, which maps index -> owning processor.

void compact_nodes()
Recompact the nodes because they are not copied properly when instantiating using the copy constructor.

int get_index_owner(int idx)
returns the processor that owns the data associated with the index.

bool is_local_cache() const
Returns "true" is running in local cache mode.
In local cache mode, each node contains a complete copy of the data set. This is stored in a shared memory
segment, but part of the set may be spilled to disk if memory is insufficient. Local cache mode is activated
via the cmd line flag: data_store_cache.

void preload_local_cache()
Read the data set into memory.
Each rank reads a portion of the data set, then bcasts to all other ranks.

int get_data_size()
for use during development and debugging.

void copy_members(const data_store_conduit &rhs)
made public for debugging during development.

void flush_debug_file()
Closes then reopens the debug logging file.
Debug logging is enabled on all ranks via the cmd line flag: data_store_debug.

void flush_profile_file()
Closes then reopens the profile logging file.
Profile logging is enabled on P_0 via the cmd line flag: data_store_profile.

void write_checkpoint(std::string dir_name)
Writes object's state to file.

void load_checkpoint(std::string dir_name, generic_data_reader *reader = nullptr)
Loads object's state from file.

Public Members

std::ofstream *m_debug = nullptr
only used for debugging; pass debug on cmd line to get each data store to print to a different file. This is
made public so data readers can also print to the file.
5.6 Execution Context Interface

When a model is attached to a trainer, the execution context of the training algorithm is stored in an `execution_context` (or sub-class) object per execution mode. Thus there is one execution context per model and mode that contains all of the state with respect to the training algorithm being applied to the model.

For example it tracks the current:

- step
- execution mode
- epoch
- and a pointer back to the trainer.

5.6.1 Documentation of `lbann::execution_context`

```cpp
class execution_context
  Subclassed by `lbann::sgd_execution_context`

Public Functions

  execution_context(observer_ptr<trainer> trainer, lbann_comm *comm, execution_mode mode)
                  Constructor.

  ~execution_context()
                   Destructor.

  std::unique_ptr<execution_context> copy_execution_context() const
                  Copy execution_context.

  template<class Archive>
  void serialize(Archive &ar)
                  Archive for checkpoint and restart.

size_t get_step() const
                  Current step in the training algorithm.

void inc_step()
                  Increment the current step in the training algorithm.

void set_execution_mode(execution_mode mode)
                  Get the mode that the trainer is currenting executing.

  execution_mode get_execution_mode() const
                  Get the mode that the trainer is currenting executing.

bool get_terminate_training() const
                  Return true if the flag to stop training is set.

void set_terminate_training(bool f)
                  Set the terminate training flag (on or off).
```
trainer & get_trainer()
Get the execution environment

bool background_io_activity_allowed()
Are background I/O activities enabled by the input layers

virtual void save_to_checkpoint_shared(persist & p)
Checkpoint training_algorithm to given file descriptor

virtual void load_from_checkpoint_shared(persist & p)
Restore training_algorithm by reading checkpoint from given file descriptor

Documentation of lbann::sgd_execution_context

class sgd_execution_context : public lbann::execution_context

  SGD Uses the step to track the Current mini-batch step for execution mode.
  Step counts are not reset after each epoch.

Public Functions

sgd_execution_context (observer_ptr<trainer> trainer, lbann_comm * comm, execution_mode mode, size_t mini_batch_size)
Constructor.

virtual ~sgd_execution_context ()
Destructor.

sgd_execution_context (const sgd_execution_context & other)
Copy constructor.

sgd_execution_context & operator= (const sgd_execution_context & other)
Copy assignment operator.

sgd_execution_context (sgd_execution_context && other)
Move constructor.

sgd_execution_context & operator= (sgd_execution_context && other)
Move assignment operator.

virtual std::unique_ptr<execution_context> copy_execution_context () const
Copy sgd_execution_context.

template<class Archive>
void serialize(Archive & ar)
Archive for checkpoint and restart

size_t get_epoch () const
Number of times the training set has been traversed.

void inc_epoch ()
Increment the current epoch in the execution context.
Increment the counter tracking the number of times that the data set has been traversed.

void set_current_mini_batch_size (size_t mini_batch_size)
Set the trainer’s current mini-batch size.
size_t get_current_mini_batch_size() const
    Get the trainer’s current mini-batch size.

size_t get_effective_mini_batch_size() const
    Get the trainer’s effective mini-batch size.

void set_effective_mini_batch_size(size_t mini_batch_size)
    Set the trainer’s effective mini-batch size.

virtual void save_to_checkpoint_shared(persist &p)
    Checkpoint training_algorithm to given file descriptor

virtual void load_from_checkpoint_shared(persist &p)
    Restore training_algorithm by reading checkpoint from given file descriptor

5.6.2 Documentation of lbann::sgd_termination_criteria

class sgd_termination_criteria : public lbann::termination_criteria

5.6.3 Documentation of lbann::termination_criteria

class termination_criteria
    Subclassed by lbann::sgd_termination_criteria

5.7 I/O Utilities

Classes for persisting the state of LBANN (checkpoint and restart), file I/O and data buffers.

5.7.1 Documentation of lbann::persist

class persist

5.7.2 Data Buffers for Data Ingestion

The data buffer classes describe how data is distributed across the input layer. Note that this part of the class hierarchy is scheduled to be deprecated and folded into the existing input layer class.

Documentation of lbann::generic_io_buffer

class generic_io_buffer
    Subclassed by lbann::partitioned_io_buffer

    Public Functions

    virtual std::string get_type() const = 0
        Return this buffer’s type, e.g: “partitioned_io_buffer,” etc.
5.8 Layer Interface

LBANN models are defined in model prototext files. The bulk of these definitions will be the series of layers which make up the model itself. LBANN layers all inherit from the common base `lbann::Layer`. The concrete layers belong to one of several categories.
5.8.1 Documentation of lbann::Layer

class Layer
Neural network tensor operation.

A layer takes input tensors ("previous activations") and applies a mathematical operation to obtain output tensors ("activations"). This operation often has trainable parameters called "weights." The previous activations are retrieved from "parent layers" and the activations are sent to "child layers," making each layer a node in a directed graph. The layer graph and the weights are managed by a neural network model class. A layer should also be able to take objective function gradients w.r.t. the outputs ("previous error signals") and compute the objective function gradients w.r.t. the inputs ("error signals") and w.r.t. the weights. This allows the model to perform automatic differentiation and to apply first-order optimization methods to the weights.

Subclassed by lbann::argmax_layer< Layout, Device >, lbann::argmin_layer< Layout, Device >, lbann::base_convolution_layer< Device >, lbann::bilinear_resize_layer< Layout, Device >, lbann::categorical_accuracy_layer< T_layout, Dev >, lbann::channelwise_mean_layer< Layout, Device >, lbann::channelwise_scale_bias_layer< Layout, Device >, lbann::clamp_layer< Layout, Device >, lbann::cross_entropy_layer< T_layout, Dev >, lbann::elu_layer< Layout, Device >, lbann::embedding_layer< Layout, Device >, lbann::entrywise_batch_normalization_layer< Layout, Device >, lbann::entrywise_scale_bias_layer< Layout, Device >, lbann::identity_layer< Layout, Device >, lbann::io_layer, lbann::l1_norm_layer< T_layout, Dev >, lbann::l2_norm2_layer< T_layout, Dev >, lbann::leaky_relu_layer< Layout, Device >, lbann::learning_layer, lbann::log_softmax_layer< Layout, Device >, lbann::mean_absolute_error_layer< T_layout, Dev >, lbann::mean_squared_error_layer< T_layout, Dev >, lbann::mini_batch_index_layer< Layout, Device >, lbann::mini_batch_size_layer< Layout, Device >, lbann::one_hot_layer< Layout, Device >, lbann::regularizer_layer, lbann::softmax_layer< Layout, Device >, lbann::tessellate_layer< Layout, Device >, lbann::top_k_categorical_accuracy_layer< T_layout, Dev >, lbann::transform_layer, lbann::variance_layer< Layout, Device >

Public Functions

virtual Layer *copy() const = 0
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

virtual std::string get_type() const = 0
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

std::string get_name() const
Get the layer instance’s name. Each layer in a model should have a unique, preferably human-readable, name.

void set_name(const std::string name)
Set the layer instance’s name. Each layer in a model should have a unique, preferably human-readable, name.

virtual description get_description() const
Human-readable description.

virtual void forward_prop()
Forward propagation step. Apply a mathematical operation to input tensors to obtain output tensors.

virtual void back_prop()
Backward propagation step. Given the objective function gradients w.r.t. the output tensors, compute the gradients w.r.t. the input tensors and w.r.t. the weights. This is essentially an application of the chain rule.
virtual bool update()
Update step. Update the layer’s internal members. Note that the optimization step for the weights happens elsewhere.

virtual void setup()
Setup layer members. This calls the ‘setup_pointers’, ‘setup_dims’, ‘setup_matrices’, ‘setup_data’, and ‘setup_gpu’ (if needed) functions. It is assumed that pointers to parent/child layers have already been initialized.

virtual void check_setup()
Check that the setup is reasonable.

virtual data_layout get_data_layout() const = 0
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

virtual El::Device get_device_allocation() const = 0
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

std::string get_data_layout_string(data_layout d) const
Get a human-readable description of the data_layout

std::string get_device_allocation_string (El::Device dev) const
Get a human-readable description of the device allocation

std::string get_device_allocation_string_short (El::Device dev) const
Get a short human-readable description of the device allocation

virtual void reset_counters()
Reset layer stat counters.

bool using_gpus() const
Whether the layer is using a GPU implementation.

int get_expected_num_parent_layers() const
Get expected number of parent layers. A negative value indicates no limit.

int get_expected_num_child_layers() const
Get expected number of child layers. A negative value indicates no limit.

model *get_model() const
Return the model that manages this layer.

void set_model(model *m)
Set the model that manages this layer.

virtual void write_proto(lbann_data::Layer *proto) const
Write layer to proto file

std::vector< const Layer * > & get_parent_layers()
Get parent layers.

const std::vector< const Layer * > & get_parent_layers() const
Get parent layers. (const)
std::vector<const Layer *> &get_child_layers()
   Get child layers.

const std::vector<const Layer *> &get_child_layers() const
   Get child layers. (const)

int get_num_parents() const
   Get number of parent layers.

int get_num_children() const
   Get number of child layers.

void add_parent_layer(const Layer *parent)
   Add a parent layer. Does nothing if parent is a null pointer, the same layer, or already a parent.

void add_child_layer(const Layer *child)
   Add a child layer. Does nothing if child is a null pointer, the same layer, or already a child.

void clear_parent_layers()
   Remove all parent layers. Parent layers are not deallocated.

void clear_child_layers()
   Remove all child layers. Child layers are not deallocated.

virtual std::vector<Layer *> get_layer_pointers()
   Get list of pointers to other layers.

virtual void set_layer_pointers(std::vector<Layer *> layers)
   Set list of pointers to other layers.

std::vector<weights *> &get_weights()
   Get references to weights.

const std::vector<weights *> &get_weights() const
   Get references to weights. (const)

void set_weights(std::vector<weights *> w)
   Set list of pointers to weights.

void replace_weights(Layer *other_layer)
   Replace weights with another Layer’s weights

std::vector<int> get_input_dims(int input_index = 0) const
   Get input tensor dimensions.

int get_input_size(int input_index = 0) const
   Get input tensor size.

std::vector<int> get_output_dims(int output_index = 0) const
   Get output tensor dimensions.

int get_output_size(int output_index = 0) const
   Get output tensor size.

void set_output_dims(std::vector<int> dims, int output_index = 0)
   Set output tensor dimensions.

AbsDistMat &get_activations(int child_index = 0)
   Get activation tensor.
AbsDistMat &get_error_signals (int parent_index = 0)
   Get error signal tensor.

const AbsDistMat &get_prev_activations (int parent_index = 0) const
   Get previous activation tensor.

const AbsDistMat &get_activations (int child_index = 0) const
   Get activation tensor.

const AbsDistMat &get_prev_error_signals (int child_index = 0) const
   Get previous error signal tensor.

const AbsDistMat &get_error_signals (int parent_index = 0) const
   Get error signal tensor.

AbsMat &get_local_activations (int child_index = 0)
   Get local portion of activation tensor.

AbsMat &get_local_error_signals (int parent_index = 0)
   Get local portion of error signal tensor.

const AbsMat &get_local_prev_activations (int parent_index = 0) const
   Get local portion of previous activation tensor.

const AbsMat &get_local_activations (int child_index = 0) const
   Get local portion of activation tensor.

const AbsMat &get_local_prev_error_signals (int child_index = 0) const
   Get local portion of previous error signal tensor.

const AbsMat &get_local_error_signals (int parent_index = 0) const
   Get local portion of error signal tensor.

lbann_comm *get_comm () const
   Get reference to LBANN communicator.

void set_hint_layer (const Layer *)
   Set hint layer. Properties of the hint layer are used during the setup phase. For instance, the output tensor dimensions are set to match the hint layer’s first output tensor.

const Layer *get_hint_layer () const
   Get hint layer.

Public Static Functions

static std::string get_layer_names (const std::vector<const Layer *> &list)
   Get names in a particular list of layers

Documentation of lbann::argmax_layer

template<data_layout Layout, El::Device Device>
class argmax_layer : public lbann::Layer
   Get index of maximum-value tensor entry.

   Expects a 1-D input tensor. If multiple entries have the same maximum value, outputs the index of the first one.
Public Functions

argmax_layer *copy() const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const

Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::argmin_layer

template<
    data_layout Layout,
    El::Device Device>

class argmin_layer : public lbann::Layer

Get index of minimum-value tensor entry.

Expects a 1-D input tensor. If multiple entries have the same minimum value, outputs the index of the first one.

Public Functions

argmin_layer *copy() const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const

Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::base_convolution_layer

template<
    El::Device Device>

class base_convolution_layer : public lbann::Layer

Computation kernels for convolution and deconvolution layers.

Subclassed by lbann::convolution_layer< Layout, Device >, lbann::deconvolution_layer< Layout, Device >
Public Functions

base_convolution_layer (lbann_comm *comm, int num_data_dims, int output_channels,
std::vector<int> conv_dims, std::vector<int> pads, std::vector<int> strides, std::vector<int> dilations, int groups, bool has_bias)

description get_description () const
   Human-readable description.

void setup_dims ()
   Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_data ()
   Setup layer data. The kernel weights are setup in the convolution and deconvolution classes.

void setup_gpu ()
   Initialize GPU objects.

Documentation of lbann::convolution_layer

template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class convolution_layer : public lbann::base_convolution_layer<Device>
   Standard deep learning convolution.

   Applies convolution (more precisely, cross-correlation) to input tensors. This is primarily optimized for image data in NCHW format.

Public Functions

convolution_layer *copy () const
   Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
   Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
   Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
   Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::deconvolution_layer

template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class deconvolution_layer : public lbann::base_convolution_layer<Device>
   Transpose of the convolution layer.
Public Functions

deconvolution_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

Documentation of lbann::bilinear_resize_layer

template<data_layout Layout, El::Device Device>
class bilinear_resize_layer : public lbann::Layer
Resize image with bilinear interpolation.

Tensors are assumed to be image data in CHW format. Gradients are not propagated during backprop.

Public Functions

bilinear_resize_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void fp_compute()
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.
Documentation of `lbann::categorical_accuracy_layer`

```
template<data_layout T_layout, El::Device Dev>
class categorical_accuracy_layer : public lbann::Layer
```

0-1 loss function.

Requires two inputs, which are respectively interpreted as prediction scores and as a one-hot label vector. The output is one if the top entries in both inputs are in the same position and is otherwise zero. Ties are broken in favor of entries with smaller indices.

This is primarily intended for use as a metric since it is not differentiable.

**Public Functions**

- `categorical_accuracy_layer * copy () const`
  - Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`
  - Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

- `data_layout get_data_layout () const`
  - Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

- `El::Device get_device_allocation () const`
  - Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

- `void setup_dims ()`
  - Setup tensor dimensions Called by the `setup` function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

- `void fp_compute ()`
  - Apply layer operation. Called by the `forward_prop` function. Given the input tensors, the output tensors are populated with computed values.

Documentation of `lbann::channelwise_mean_layer`

```
template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class channelwise_mean_layer : public lbann::Layer
```

**Public Functions**

- `channelwise_mean_layer * copy () const`
  - Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`
  - Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.
data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::channelwise_scale_bias_layer

template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class channelwise_scale_bias_layer : public lbann::Layer
Apply scale and bias to tensor channels.

The input tensor is sliced along the first tensor dimension (the “channel” dimension, assuming image data in CHW format) and scale and bias terms are applied independently to each slice. More precisely, given input and output tensors $X, Y \in \mathbb{R}^{d_1 \times \cdots \times d_n}$ and scale and bias vectors $a, b \in \mathbb{R}^{d_1}$:

$$Y_{i,j,\ldots} = a_i X_{i,j,\ldots} + b_i$$

The scale and bias vectors are fused into a single weights tensor to reduce the number of gradient allreduces during backprop. In particular, the weights tensor is a num_channels \times 2 matrix, where the first column correspond to scale terms and the second column to bias terms.

Public Functions

channelwise_scale_bias_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_matrices(const El::Grid &grid)
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstatiated.

void setup_data()
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.
Documentation of lbann::clamp_layer

template<data_layout Layout, El::Device Device>
class clamp_layer : public lbann::Layer

Constrain values to a range.

\[
\text{clamp}(x; \text{min}, \text{max}) = \begin{cases} 
\min x & \text{min} \leq x < \text{min} \\
 x & \min < x < \text{max} \\
\max x & x \geq \text{max}
\end{cases}
\]

Public Functions

clamp_layer *\textbf{copy} () const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string \textbf{get_type} () const

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout \textbf{get_data_layout} () const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device \textbf{get_device_allocation} () const

Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description \textbf{get_description} () const

Human-readable description.

Documentation of lbann::covariance_layer

template<data_layout Layout, El::Device Device>
class covariance_layer : public lbann::Layer

Estimate covariance.

Given inputs \(x\) and \(y\) with empirical means \(\bar{x}\) and \(\bar{y}\), an unbiased estimator for the covariance is given by

\[
\sigma^2_{xy} \approx \frac{1}{n-1} \sum_{i=1}^{n} (x - \bar{x})(y - \bar{y})
\]

Scaling by \(1/n\) instead of \(1/(n-1)\) is a biased estimator.

Public Functions

covariance_layer *\textbf{copy} () const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.
std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const
Human-readable description.

Documentation of lbann::cross_entropy_layer

template<data_layout T_layout, El::Device Dev>
class cross_entropy_layer : public lbann::Layer
Cross entropy loss function.
Given a predicted distribution $y$ and ground truth distribution $\hat{y}$,

$$CE(y, \hat{y}) = - \sum_i \hat{y}_i \log y_i$$

Public Functions

cross_entropy_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims ()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_data ()
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.
void fp_compute()  
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void bp_compute()  
Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.

Documentation of lbann::elu_layer

template<
    data_layout Layout,
    El::Device Device>

class elu_layer : public lbann::Layer
    Exponential linear unit.

ELU\left(x; \alpha\right) = \begin{cases}  
x & x > 0 \\
\alpha(e^x - 1) & x \leq 0
\end{cases}

\alpha should be non-negative. See:

Public Functions

eLU_layer *copy() const  
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const  
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const  
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const  
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description() const  
Human-readable description.

Documentation of lbann::embedding_layer

template<
    data_layout Layout,
    El::Device Device>

5.8. Layer Interface
class embedding_layer : public lbann::Layer

Lookup table to vectors of fixed size.

Takes a scalar input, interprets it as an index, and outputs the corresponding vector. The number of embedding vectors and the size of vectors are fixed. If the index is out-of-range, then the output is a vector of zeros.

The embedding vectors are stored in an embedding_dim × num_embeddings weights matrix. Note that this is the transpose of the weights in the PyTorch embedding layer.

Public Functions

eMBEDDING_LAYERSHAPE (lbann_comm *comm, size_t num_embeddings, size_t embedding_dim, El::Int padding_idx = -1)

Parameters

• comm: LBANN communicator.
• num_embeddings: Size of dictionary of embeddings.
• embedding_dim: Size of embedding vectors.
• padding_idx: If set, then the corresponding embedding vector is initialized with zeros. The objective function gradient w.r.t. this embedding vector is always zero.

eMBEDDING_LAYERSHAPE *copy () const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const

Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

data_layout get_data_layout () const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const

Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const

Human-readable description.

Documentation of lbann::entrywise_batch_normalization_layer

template<data_layout Layout, El::Device Device>

class entrywise_batch_normalization_layer : public lbann::Layer

Each input entry is normalized across the mini-batch to have zero mean and unit standard deviation. This uses the standard approach of maintaining the running mean and standard deviation (with exponential decay) for use at test time. See:

**Public Functions**

entrywise_batch_normalization_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const
Human-readable description.

**Documentation of lbann::entrywise_scale_bias_layer**

template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class entrywise_scale_bias_layer : public lbann::Layer
Apply scale and bias to tensor entries.
Scale and bias terms are applied independently to each tensor entry. More precisely, given input, output, scale, and bias tensors \( X, Y, A, B \in \mathbb{R}^{d_1 \times \cdots \times d_n} \):

\[
Y = A \circ X + B
\]

The scale and bias terms are fused into a single weights tensor to reduce the number of gradient allreduces during backprop. In particular, the weights tensor is a size \( \times 2 \) matrix, where the first column correspond to scale terms and the second column to bias terms.

**Public Functions**

entrywise_scale_bias_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
El::Device \texttt{get\_device\_allocation()} \texttt{const} \\
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

\textbf{void} \texttt{setup\_matrices(const El::Grid &grid)} \\
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct\_matrix’ function. If any matrices have already been setup, they are destroyed and reinstantiated.

\textbf{void} \texttt{setup\_data()} \\
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.

\textbf{void} \texttt{fp\_setup\_outputs(El::Int mini\_batch\_size)} \\
Setup output tensors. Called by the ‘forward\_prop’ function. Each output tensor is resized to match the mini-batch size.

\textbf{void} \texttt{bp\_setup\_gradient\_wrt\_inputs(El::Int mini\_batch\_size)} \\
Setup gradient w.r.t. input tensors. Called by the ‘back\_prop’ function. Each gradient w.r.t. input tensor is resized to match the mini-batch size.

\textbf{Documentation of lbann::identity\_layer}

\texttt{template<\texttt{data\_layout Layout, El::Device Device}>}

\texttt{class identity\_layer: public lbann::Layer} \\
Output a tensor view.

Forward and backward prop simply involve setting up tensor views, and hence are very cheap.

\textbf{Public Functions}

identity\_layer \* \texttt{copy()} \texttt{const} \\
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

\texttt{std::string get\_type()} \texttt{const} \\
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data\_layout \texttt{get\_data\_layout()} \texttt{const} \\
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device \texttt{get\_device\_allocation()} \texttt{const} \\
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

\textbf{Documentation of lbann::io\_layer}

\texttt{class io\_layer: public lbann::Layer} \\
Subclassed by \texttt{lbann::generic\_input\_layer}
Public Functions

\texttt{virtual dataset \& get\_dataset (execution\_mode m) = 0}

Return the dataset for the given execution mode.

\texttt{virtual dataset \& select\_dataset () = 0}

Return the dataset associated with the current execution mode.

\texttt{virtual dataset \* select\_first\_valid\_dataset () = 0}

Return the first dataset with a valid (non-null) datareader. Returns null if none are valid.

\texttt{virtual generic\_data\_reader \* select\_data\_reader () const = 0}

Return the data reader associated with the current execution mode.

\texttt{virtual long update\_num\_samples\_processed (long num\_samples) = 0}

Update the number of samples processed for the current execution mode.

\texttt{El::Matrix<El::Int> \* get\_sample\_indices\_per\_mb () = 0}

Return the sample indices fetched in the current mini-batch.

\texttt{virtual const std::vector<int> get\_data\_dims (int child\_index = 0) const = 0}

Get the dimensions of the underlying data.

\texttt{virtual long get\_linearized\_data\_size () const = 0}

Get the linearized size of the underlying data.

\texttt{virtual long get\_linearized\_label\_size () const = 0}

Get the linearized size of the labels for the underlying data.

Documentation of \texttt{lbann::generic\_input\_layer}

\texttt{class generic\_input\_layer : public lbann::io\_layer}

Subclassed by \texttt{lbann::input\_layer< T\_io\_buffer, T\_layout, Dev >}

Public Functions

\texttt{template<class Archive> void serialize (Archive \& ar)\n}

Archive for checkpoint and restart

\texttt{std::string get\_type () const\n}

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

\texttt{description get\_description () const\n}

Human-readable description.

\texttt{void setup\_dims ()\n}

Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

\texttt{void setup\_data ()\n}

Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.
void **fp_setup_outputs** (El::Int *mini_batch_size)

Setup output tensors. Sets up the effective (global) mini-batch size.

void **collect_background_data_fetch** (execution_mode mode)

Check for each buffer if there is an outstanding fetch request.

void **fp_compute** ()

Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

bool **update_compute** ()

Once a mini-batch is processed, resuffle the data for the next batch if necessary.

void **calculate_num_iterations_per_epoch_training_spans_models** (int mini_batch_size)

Calculate how many iterations are required for training, testing, and validation given a specified mini-batch size and that the training data set is spanning all of the models.

dataset &**get_dataset** (execution_mode m)

Return the dataset for the given execution mode.

dataset &**select_dataset** ()

Return the dataset associated with the current execution mode.

dataset *select_first_valid_dataset ()

Return the first dataset with a valid (non-null) datareader. Returns null if none are valid.

generic_data_reader *select_data_reader () const

Return the data reader associated with the current execution mode.

long **update_num_samples_processed** (long num_samples)

Update the number of samples processed for the current execution mode.

El::Matrix<El::Int> *get_sample_indices_per_mb ()

Return the sample indices fetched in the current mini-batch.

const std::vector<int> get_data_dims (int child_index = 0) const

Get the dimensions of the underlying data.

long **get_linearized_data_size** () const

Get the linearized size of the underlying data.

long **get_linearized_label_size** () const

Get the linearized size of the labels for the underlying data.

Documentation of lbann::input_layer

template<ctypename _T_io_buffer , data_layout _T_layout = data_layout::DATA_PARALLEL, El::Device _Dev = El::Device::CPU> class input_layer : public lbann::generic_input_layer

Interface with data reader.

Public Functions

**input_layer** (lbann_comm *comm , int num_parallel_readers , std::map<execution_mode, generic_data_reader *> data_readers , bool data_set_spans_models = true, data_reader_target_mode target_mode = data_reader_target_mode::CLASSIFICATION)
input_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the decice allocation of the previous ac-
tivations, activations, previous error signals, and error signals are the same. Each concrete layer that is
templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::L1_norm_layer

template<data_layout T_layout, El::Device Dev>
class L1_norm_layer: public lbann::Layer
L1 vector norm.

\[ \|x\|_1 = \sum_i |x_i| \]

Public Functions

L1_norm_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the decice allocation of the previous ac-
tivations, activations, previous error signals, and error signals are the same. Each concrete layer that is
templated on its device allocation should override this function to return its template parameter.

void setup_dims ()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_data ()
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.
void **fp_compute** ()
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void **bp_compute** ()
Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.

**Documentation of lbann::l2_norm2_layer**

```cpp
template<data_layout T_layout, El::Device Dev>
class l2_norm2_layer : public lbann::Layer

Square of L2 vector norm.

\[ \|x\|_2^2 = \sum_i x_i^2 \]
```

**Public Functions**

l2_norm2_layer * **copy** () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string **get_type** () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout **get_data_layout** () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device **get_device_allocation** () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void **setup_dims** ()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void **setup_data** ()
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.

void **fp_compute** ()
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void **bp_compute** ()
Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.
Documentation of lbann::leaky_relu_layer

template<data_layout Layout, El::Device Device>
class leaky_relu_layer : public lbann::Layer

    LeakyReLU(x; \alpha) = \begin{cases} 
    x & x > 0 \\
    \alpha x & x \leq 0
    \end{cases}

See:

Public Functions

leaky_relu_layer *copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description() const
    Human-readable description.

Documentation of lbann::learning_layer

class learning_layer : public lbann::Layer

    Subclassed by lbann::fully_connected_layer<T_layout, Dev>

Documentation of lbann::fully_connected_layer

template<data_layout T_layout, El::Device Dev>
class fully_connected_layer : public lbann::learning_layer

    Affine transformation.
    Flattens the input tensor, multiplies with a weights matrix, and optionally applies an entry-wise bias. Following the column-vector convention:

    \[ y = W \ast \text{vec}(x) + b \]
Two weights are required if bias is applied: the linearity and the bias. Only the linearity weights are required if bias is not applied. If weights aren’t provided, the linearity weights are initialized with He normal initialization and the bias weights are initialized to zero.

Public Functions

`fully_connected_layer(lbann_comm *comm, int output_size, bool transpose = false, weights *weight = nullptr, bool has_bias = true)`

`fully_connected_layer *copy () const`

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

`std::string get_type () const`

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

`data_layout get_data_layout () const`

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

`El::Device get_device_allocation () const`

Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

`description get_description () const`

Human-readable description.

Documentation of `lbann::log_softmax_layer`

`template<data_layout Layout, El::Device Device>
class log_softmax_layer : public lbann::Layer`

Logarithm of softmax function.

\[
\log \text{softmax}(x)_i = x_i - \log \sum_j e^{x_j}
\]

Public Functions

`log_softmax_layer *copy () const`

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

`std::string get_type () const`

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

`data_layout get_data_layout () const`

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_matrices(const El::Grid &grid)
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstantiated.

void fp_setup_outputs(El::Int mini_batch_size)
Setup output tensors. Called by the ‘forward_prop’ function. Each output tensor is resized to match the mini-batch size.

void fp_compute()
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void bp_compute()
Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.

Documentation of lbann::mean_absolute_error_layer

```
template<data_layout T_layout, El::Device Dev>
class mean_absolute_error_layer : public lbann::Layer
```

Given a prediction $y$ and ground truth $\hat{y}$,

$$MAE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Public Functions

mean_absolute_error_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_data()
Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.

void fp_compute()
Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void bp_compute()
Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.

Documentation of lbann::mean_squared_error_layer

template<data_layout T_layout, El::Device Dev>
class mean_squared_error_layer : public lbann::Layer
Given a prediction $y$ and ground truth $\hat{y}$,

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Public Functions

mean_squared_error_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims()
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.
void setup_data()
    Setup layer data. Called by the ‘setup’ function. Memory is allocated for distributed matrices.

void fp_compute()
    Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors
    are populated with computed values.

void bp_compute()
    Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and
    gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and
    the gradients w.r.t. the weights are sent to the appropriate optimizers.

Documentation of lbann::mini_batch_index_layer

template<
    data_layout Layout = data_layout::DATA_PARALLEL,
    El::Device Device = El::Device::CPU>

class mini_batch_index_layer : public lbann::Layer

    Mini-batch index.

    Output tensor is a 1D tensor with a single entry containing the mini-batch sample. Each sample in a model’s
    mini-batch has a unique index in [0, mini_batch_size).

Public Functions

mini_batch_index_layer * copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.

std::string get_type() const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s
    mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations,
    previous error signals, and error signals are the same. Each concrete layer that is templated on its data
    layout should override this function to return its template parameter.

El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous ac-
    tivations, activations, previous error signals, and error signals are the same. Each concrete layer that is
    templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::mini_batch_size_layer

template<
    data_layout Layout = data_layout::DATA_PARALLEL,
    El::Device Device = El::Device::CPU>

class mini_batch_size_layer : public lbann::Layer

    Mini-batch size.

    Output tensor is a 1D tensor with a single entry containing the model’s current mini-batch size.

Public Functions

mini_batch_size_layer * copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.
std::string get_type() const
Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::one_hot_layer

template<data_layout Layout, El::Device Device>
class one_hot_layer : public lbann::Layer
Convert index to a one-hot vector.

Expects a scalar input tensor and outputs a 1-D output tensor with size entries. The input is interpreted as an index, and output entries are one if they correspond to that index and zero otherwise. If the input is outside [0, size), then the output is all zeros.

Public Functions

one_hot_layer* copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::regularizer_layer

class regularizer_layer : public lbann::Layer
Subclassed by lbann::batch_normalization_layer< T_layout, Dev >, lbann::dropout< T_layout, Dev >, lbann::local_response_normalization_layer< T_layout, Dev >, lbann::selu_dropout< T_layout, Dev >
Documentation of lbann::batch_normalization_layer

template<data_layout T_layout, El::Device Dev>
class batch_normalization_layer : public lbann::regularizer_layer

Each input channel is normalized across the mini-batch to have zero mean and unit standard deviation. Learned scaling factors and biases are then applied. This uses the standard approach of maintaining the running mean and standard deviation (with exponential decay) for use at test time. See:


Public Functions

batch_normalization_layer (lbann_comm *comm, DataType decay = 0.9, DataType epsilon = 1e-5, int statistics_group_size = 1)

Set up batch normalization.

Parameters

• comm: The communication context for this layer
• decay: Controls the momentum of the running mean/standard deviation averages.
• epsilon: A small number to avoid division by zero.
• statistics_group_size: Number of processors to aggregate statistics over. Defaults to 1 (i.e. local aggregation).

batch_normalization_layer *copy () const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const

Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const

Human-readable description.

Documentation of lbann::dropout

template<data_layout T_layout, El::Device Dev>
class dropout : public lbann::regularizer_layer

Probabilistically drop layer outputs.

5.8. Layer Interface
The weights are multiplied by 1/(keep probability) at training time. Keep probabilities of 0.5 for fully-connected layers and 0.8 for input layers are good starting points. See:


Public Functions

dropout (lbann_comm *comm, EvalType keep_prob = EvalType(0.5))
Keep units with probability keep_prob.

dropout *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const
Human-readable description.

EvalType get_keep_prob () const
get prob for keep each unit.

void set_keep_prob (EvalType keep_prob)
set prob for keep each unit.

Documentation of lbann::local_response_normalization_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class local_response_normalization_layer : public lbann::regularizer_layer
See:
Public Functions

local_response_normalization_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const
Human-readable description.

Documentation of lbann::selu_dropout

template<data_layout T_layout, El::Device Dev>
class selu_dropout : public lbann::regularizer_layer
Scaled dropout for use with SELU activations.

A default keep probability of 0.95 is recommended. See:

Public Functions

selu_dropout (lbann_comm *comm, float keep_prob = 0.95f, DataType alpha =
Datatype(1.6732632423543772848170429916717),
Datatype scale =
Datatype(1.0507009873554804934193349852946))
Keep units with probability keep_prob.

selu_dropout *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
El::Device get_device_allocation ( ) const  
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims ( )  
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_matrices ( const El::Grid & grid)  
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstantiated.

Documentation of lbann::softmax_layer

template<data_layout Layout, El::Device Device>  
class softmax_layer : public lbann::Layer

softmax(x)ᵢ = \frac{e^{xᵢ}}{\sum_j e^{xⱼ}}

Public Functions

softmax_layer * copy ( ) const  
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type ( ) const  
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout ( ) const  
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation ( ) const  
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims ( )  
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

void setup_matrices ( const El::Grid & grid)  
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstantiated.
void **fp_setup_outputs** ([El::Int mini_batch_size])**

Setup output tensors. Called by the ‘forward_prop’ function. Each output tensor is resized to match the mini-batch size.

void **fp_compute** ()

Apply layer operation. Called by the ‘forward_prop’ function. Given the input tensors, the output tensors are populated with computed values.

void **bp_compute** ()

Compute objective function gradients. Called by the ‘back_prop’ function. Given the input, output, and gradient w.r.t. output tensors, the gradient w.r.t. input tensors are populated with the computed values and the gradients w.r.t. the weights are sent to the appropriate optimizers.

**Documentation of lbann::tessellate_layer**

template<data_layout Layout = data_layout::DATA_PARALLEL, El::Device Device = El::Device::CPU>
class tessellate_layer : public lbann::Layer

Repeat a tensor until it matches specified dimensions.

The output dimensions do not need to be integer multiples of the input dimensions.

As an example, tessellating a $2 \times 2$ matrix into a $3 \times 4$ matrix looks like the following:

$\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix} \rightarrow \begin{bmatrix}
1 & 2 & 1 & 2 \\
3 & 4 & 3 & 4 \\
1 & 2 & 1 & 2
\end{bmatrix}$

Formally, suppose we tessellate an input tensor $X$ with dimensions $d_1 \times \cdots \times d_n$ to obtain an output tensor $Y$ with dimensions $e_1 \times \cdots \times e_n$. Then, denoting the modulo operator with $\%$,

$Y_{i_1, \cdots, i_n} = X_{i_1 \% d_1, \cdots, i_n \% d_n}$

**Public Functions**

tessellate_layer *copy () const

Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const

Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_dims ()

Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.
void setup_matrices(const El::Grid &grid)
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstantiated.

Documentation of lbann::top_k_categorical_accuracy_layer

template<data_layout T_layout, El::Device Dev>
class top_k_categorical_accuracy_layer : public lbann::Layer
Requires two inputs, which are respectively interpreted as prediction scores and as a one-hot label vector. The output is one if the corresponding label matches one of the top-k prediction scores and is otherwise zero. Ties in the top-k prediction scores are broken in favor of entries with smaller indices.

Public Functions

top_k_categorical_accuracy_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the decice allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description () const
Human-readable description.

Documentation of lbann::transform_layer

class transform_layer : public lbann::Layer
Subclassed by lbann::abstract_evaluation_layer, lbann::bernoulli_layer< T_layout, Dev >, lbann::categorical_random_layer< T_layout, Dev >, lbann::concatenation_layer< T_layout, Dev >, lbann::crop_layer< T_layout, Dev >, lbann::discrete_random_layer< T_layout, Dev >, lbann::dummy_layer< T_layout, Dev >, lbann::gaussian_layer< T_layout, Dev >, lbann::hadamard_layer< T_layout, Dev >, lbann::in_top_k_layer< T_layout, Dev >, lbann::pooling_layer< T_layout, Dev >, lbann::reduction_layer< T_layout, Dev >, lbann::reshape_layer< T_layout, Dev >, lbann::slice_layer< T_layout, Dev >, lbann::sort_layer< T_layout, Dev >, lbann::split_layer< T_layout, Dev >, lbann::stop_gradient_layer< T_layout, Dev >, lbann::sum_layer< T_layout, Dev >, lbann::uniform_layer< T_layout, Dev >, lbann::unpooling_layer< T_layout, Dev >, lbann::weighted_sum_layer< T_layout, Dev >, lbann::weights_layer< T_layout, Dev >
Documentation of lbann::abstract_evaluation_layer

class abstract_evaluation_layer : public lbann::transform_layer

Interface with objective function and metrics.
Subclassed by lbann::evaluation_layer< T_layout, Dev >

Public Functions

EvalType get_scale() const
Get scaling factor.

void set_scale(EvalType scale)
Set scaling factor.

EvalType get_value(bool scaled = true)
Get evaluated value.

Public Static Functions

static abstract_evaluation_layer * construct(lbann_comm * comm, data_layout layout, El::Device device)
Construct an evaluation layer. The caller is responsible for deallocating the layer.

Documentation of lbann::evaluation_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class evaluation_layer : public lbann::abstract_evaluation_layer

Evaluation layer. Computes the average value across a mini-batch. If the input tensor has multiple neurons, their values are added together.

Public Functions

evaluation_layer * copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.
Documentation of lbann::bernoulli_layer

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class bernoulli_layer : public lbann::transform_layer
    Random values with Bernoulli distribution.

    During validation and testing, outputs are all zero.

Public Functions

bernoulli_layer *copy () const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.

std::string get_type () const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s
    mathematical operation.

data_layout get_data_layout () const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations,
    previous error signals, and error signals are the same. Each concrete layer that is templated on its data
    layout should override this function to return its template parameter.

El::Device get_device_allocation () const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous ac-
    tivations, activations, previous error signals, and error signals are the same. Each concrete layer that is
    templated on its device allocation should override this function to return its template parameter.

description get_description () const
    Human-readable description.
```

Documentation of lbann::categorical_random_layer

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class categorical_random_layer : public lbann::transform_layer
    Random categorical outputs.

    Inputs are probability distributions and outputs are one-hot vectors. An input entry is the probability that the
    corresponding output entry is one.

Public Functions

categorical_random_layer *copy () const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.

std::string get_type () const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s
    mathematical operation.

data_layout get_data_layout () const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations,
    previous error signals, and error signals are the same. Each concrete layer that is templated on its data
    layout should override this function to return its template parameter.
```
El::Device \texttt{get\_device\_allocation()} const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.


documentation of lbann::concatenation\_layer

template<data\_layout T\_layout = data\_layout::DATA\_PARALLEL, El::Device Dev = El::Device::CPU>
class concatenation\_layer : public lbann::transform\_layer
Concatenate tensors along specified dimension.

Public Functions

concatenation\_layer *\texttt{copy()} const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string \texttt{get\_type()} const
Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.

data\_layout \texttt{get\_data\_layout()} const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device \texttt{get\_device\_allocation()} const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description \texttt{get\_description()} const
Human-readable description.

Documentation of lbann::constant\_layer

template<data\_layout T\_layout = data\_layout::DATA\_PARALLEL, El::Device Dev = El::Device::CPU>
class constant\_layer : public lbann::transform\_layer
Constant output.

Public Functions

constant\_layer *\texttt{copy()} const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string \texttt{get\_type()} const
Get the layer type's name. A layer type name should be brief, human-readable description of the layer's mathematical operation.
data_layout `get_data_layout()` const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device `get_device_allocation()` const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description `get_description()` const
Human-readable description.

**Documentation of lbann::crop_layer**

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class crop_layer : public lbann::transform_layer
Crop tensor.

Extract a crop from an $N$-D tensor. The second input tensor is interpreted as a normalized crop position in $[0, 1]^N$. For images in CHW format, a position of $(0,0,0)$ corresponds to the red-top-left corner and $(1,1,1)$ to the blue-bottom-right corner. The crop size is determined at setup.

**Public Functions**

crop_layer * `copy()` const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string `get_type()` const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout `get_data_layout()` const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device `get_device_allocation()` const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void `setup_matrices(const El::Grid &grid)`
Setup distributed matrices. Called by the ‘setup’ function. Each column of these distributed matrices is interpreted as the flattened tensor for a mini-batch sample. The matrices themselves are constructed by calling the ‘construct_matrix’ function. If any matrices have already been setup, they are destroyed and reinstated.

void `setup_dims()`
Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.
Documentation of lbann::discrete_random_layer

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class discrete_random_layer : public lbann::transform_layer
```

Random output from discrete distribution.

Inputs are interpreted as the probability of choosing each distribution value.

**Public Functions**

- `discrete_random_layer *copy () const`  
  Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`  
  Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

- `data_layout get_data_layout () const`  
  Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

- `El::Device get_device_allocation () const`  
  Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::dummy_layer

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class dummy_layer : public lbann::transform_layer
```

Placeholder layer.

Does no computation and is primarily intended as a placeholder for unused layer outputs.

**Public Functions**

- `dummy_layer *copy () const`  
  Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`  
  Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

- `data_layout get_data_layout () const`  
  Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::gaussian_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class gaussian_layer : public lbann::transform_layer
Random values with Gaussian distribution.
During validation and testing, outputs are all equal to the distribution mean.

Public Functions

gaussian_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description() const
Human-readable description.

Documentation of lbann::hadamard_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class hadamard_layer : public lbann::transform_layer
Entry-wise tensor product.

Public Functions

hadamard_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.
data_layout\texttt{get\_data\_layout()} \texttt{const}
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

\texttt{El::Device get\_device\_allocation()} \texttt{const}
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

\textbf{Documentation of lbann::in\_top\_k\_layer}

\texttt{template\langle data\_layout T\_layout = data\_layout::DATA\_PARALLEL, El::Device Dev = El::Device::CPU\rangle class in\_top\_k\_layer: public lbann::transform\_layer}
Indicate top-k entries.

Output entries corresponding to the top-k input entries are set to one and the rest to zero. Ties are broken in favor of entries with smaller indices.

\textbf{Public Functions}

\texttt{in\_top\_k\_layer *copy()} \texttt{const}
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

\texttt{std::string get\_type()} \texttt{const}
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout\texttt{get\_data\_layout()} \texttt{const}
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

\texttt{El::Device get\_device\_allocation()} \texttt{const}
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

\texttt{description get\_description()} \texttt{const}
Human-readable description.

\textbf{Documentation of lbann::pooling\_layer}

\texttt{template\langle data\_layout T\_layout = data\_layout::DATA\_PARALLEL, El::Device Dev = El::Device::CPU\rangle class pooling\_layer: public lbann::transform\_layer}

\textbf{Public Functions}

\texttt{pooling\_layer *copy()} \texttt{const}
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.
**std::string get_type() const**

Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

**data_layout get_data_layout() const**

Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

**El::Device get_device_allocation() const**

Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

**description get_description() const**

Human-readable description.

**Documentation of lbann::reduction_layer**

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class reduction_layer : public lbann::transform_layer
    Reduce tensor to scalar.
```

### Public Functions

```cpp
reduction_layer *copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.
```

```cpp
std::string get_type() const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.
```

```cpp
data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.
```

```cpp
El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.
```

```cpp
description get_description() const
    Human-readable description.
```

**Documentation of lbann::reshape_layer**

```cpp
template<data_layout T_layout, El::Device Dev>
class reshape_layer : public lbann::transform_layer
    Reshape tensor.
```

Forward and backward prop simply involve setting up tensor views, and hence are very cheap.
Public Functions

Reshape layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::slice_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class slice_layer : public lbann::transform_layer
Slice tensor along a specified dimension.

Suppose we slice a $D_1 \times \cdots \times D_n$ input tensor along the dimension $k$. We specify slice points $s_1, \ldots, s_\ell$, which are strictly increasing and have $s_1 = 0$ and $s_\ell = D_k$. The $i$th output tensor is then a $D_1 \times \cdots \times D_{i-1} \times (s_i - s_{i-1}) \times D_{i+1} \times \cdots \times D_n$ tensor.

Public Functions

slice_layer *copy () const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type () const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout () const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation () const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

std::vector<El::Int> &get_slice_points ()
Get slice points.

std::vector<El::Int> get_slice_points () const
Get slice points (const).
**Description**

```
description get_description() const
    Human-readable description.
```

**Documentation of lbann::sort_layer**

template<
data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class sort_layer : public lbann::transform_layer
    Sort tensor entries.

**Public Functions**

```
sort_layer *copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.

std::string get_type() const
    Get the layer type's name. A layer type name should be brief, human-readable description of the layer's
    mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations,
    previous error signals, and error signals are the same. Each concrete layer that is templated on its data
    layout should override this function to return its template parameter.

El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous activations,
    activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its
device allocation should override this function to return its template parameter.

description get_description() const
    Human-readable description.
```

**Documentation of lbann::split_layer**

template<
data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class split_layer : public lbann::transform_layer
    Present input tensor to multiple outputs.

**Public Functions**

```
split_layer *copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy.
    The caller is responsible for deallocating the instance.

std::string get_type() const
    Get the layer type's name. A layer type name should be brief, human-readable description of the layer's
    mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations,
    previous error signals, and error signals are the same. Each concrete layer that is templated on its data
    layout should override this function to return its template parameter.
```
El::Device `get_device_allocation()` const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

**Documentation of lbann::stop_gradient_layer**

template<
data_layout T_layout, El::Device Dev>
class stop_gradient_layer : public lbann::transform_layer
Block back propagation.

The output is identical to the input, but the back propagation output (i.e. the error signal) is always zero. Compare with the stop_gradient operation in TensorFlow and Keras. Note that this means that computed gradients in preceding layers are not exact gradients of the objective function.

**Public Functions**

`stop_gradient_layer *copy() const`
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

`std::string get_type() const`
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

`data_layout get_data_layout() const`
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device `get_device_allocation()` const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

**Documentation of lbann::sum_layer**

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class sum_layer : public lbann::transform_layer

**Public Functions**

`sum_layer *copy() const`
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

`std::string get_type() const`
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.
data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

Documentation of lbann::uniform_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class uniform_layer : public lbann::transform_layer
Random values with uniform distribution.
During validation and testing, outputs are all equal to the distribution mean.

Public Functions

uniform_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description get_description() const
Human-readable description.

Documentation of lbann::unpooling_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class unpooling_layer : public lbann::transform_layer
Transpose of pooling layer.

Public Functions

unpooling_layer *copy() const
Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.
std::string get_type() const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

void setup_pointers()
    Setup layer pointers. Called by the ‘setup’ function. Pointers to parent/child layers are assumed to be already initialized.

void setup_dims()
    Setup tensor dimensions Called by the ‘setup’ function. If there are any input tensors, the base method sets all uninitialized output tensor dimensions equal to the first input tensor dimensions.

std::vector<Layer*> get_layer_pointers() const
    Get list of pointers to other layers.

void set_layer_pointers(std::vector<Layer*> layers)
    Set list of pointers to other layers.

Documentation of lbann::weighted_sum_layer

template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class weighted_sum_layer : public lbann::transform_layer
    Add tensors with specified scaling factors.

Public Functions

weighted_sum_layer* copy() const
    Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

std::string get_type() const
    Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

data_layout get_data_layout() const
    Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

El::Device get_device_allocation() const
    Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.
**Description**

`get_description()` const

Human-readable description.

### Documentation of `lbann::weights_layer`

```cpp
template<data_layout T_layout = data_layout::DATA_PARALLEL, El::Device Dev = El::Device::CPU>
class weights_layer: public lbann::transform_layer
```

Output a weights tensor.

Interfaces with a `weights` object and outputs its tensor.

#### Public Functions

- `weights_layer *copy () const`  
  Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`  
  Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.

- `data_layout get_data_layout () const`  
  Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

- `El::Device get_device_allocation () const`  
  Get the device allocation for the data tensors. We assume that the decice allocation of the previous ac-tivations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

### Documentation of `lbann::variance_layer`

```cpp
template<data_layout Layout, El::Device Device>
class variance_layer: public lbann::Layer
```

Estimate variance.

Given an input $x$ with empirical mean $\bar{x}$, an unbiased estimator for the variance is given by

$$\sigma_x^2 \approx \frac{1}{n-1} \sum_{i=1}^{n} (x - \bar{x})^2$$

Scaling by $1/n$ instead of $1/(n - 1)$ is a biased estimator.

#### Public Functions

- `variance_layer *copy () const`  
  Copy function. This function dynamically allocates memory for a layer instance and instantiates a copy. The caller is responsible for deallocating the instance.

- `std::string get_type () const`  
  Get the layer type’s name. A layer type name should be brief, human-readable description of the layer’s mathematical operation.
data_layout \texttt{get\_data\_layout()} \texttt{const}
Get data layout of the data tensors. We assume that the data layouts of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its data layout should override this function to return its template parameter.

\texttt{El::Device get\_device\_allocation()} \texttt{const}
Get the device allocation for the data tensors. We assume that the device allocation of the previous activations, activations, previous error signals, and error signals are the same. Each concrete layer that is templated on its device allocation should override this function to return its template parameter.

description \texttt{get\_description()} \texttt{const}
Human-readable description.

5.8.2 Activation Layers

Documentation of \texttt{lbann::log\_sigmoid\_layer}

\texttt{class log\_sigmoid\_layer}
Logarithm of sigmoid function.

\[
\log(\sigma(x)) = -\log(1 + e^{-x})
\]


Documentation of \texttt{lbann::relu\_layer}

\texttt{class relu\_layer}
Rectified linear unit.

\[
\text{ReLU}(x) = \max(x, 0)
\]


Documentation of \texttt{lbann::selu\_layer}

\texttt{class selu\_layer}
Scaled exponential rectified linear unit.

\[
\text{SELU}(x) = \begin{cases} \alpha x & x > 0 \\ s(\alpha e^x - 1) & x \leq 0 \end{cases}
\]

with \(\alpha \approx 1.67\) and \(s \approx 1.05\). Note that SELU is equivalent to \(s\text{ELU}(x; \alpha)\). See:
Documentation of lbann::sigmoid_layer

class sigmoid_layer
   Special case of logistic function.

   \[ \sigma(x) = \frac{1}{1 + e^{-x}} \]


Documentation of lbann::softplus_layer

class softplus_layer
   Smooth approximation to ReLU function.

   \[ \text{softplus}(x) = \log(e^x + 1) \]


Documentation of lbann::softsign_layer

class softsign_layer
   Smooth approximation to sign function.

   \[ \text{softsign}(x) = \frac{x}{1 + |x|} \]

5.8.3 Image Layers

5.8.4 I/O Layers
5.9 Metrics Interface

A metric function can be used to evaluate the performance of a model without affecting the training process. Users define the metric with which to test their model in their model prototext file. The available metric functions in LBANN are found below.

5.9.1 Documentation of lbann::metric

class metric

Abstract base class for metric functions. A metric function can be used to evaluate the performance of a model without affecting the training process.

Subclassed by lbann::layer_metric

Public Functions

metric(lbann_comm *comm)

Constructor.

metric(const metric &other)

Copy constructor.

metric &operator=(const metric &other)

Copy assignment operator.

virtual ~metric()

Destructor.

virtual metric *copy() const = 0

Copy function.

template<class Archive>
void serialize(Archive &ar)

Archive for checkpoint and restart

virtual std::string name() const = 0

Return a string name for this metric.
virtual std::string get_unit () const
   Return a display unit for this metric. Default is an empty string. This is overridden if the metric has units,
   e.g. “%” or “sec”.

t::virtual void setup (model &m)
   Setup metric.

virtual EvalType evaluate (execution_mode mode, int mini_batch_size) = 0
   Evaluate the metric value. This function takes the model’s current mini-batch size. If multiple models are
   being trained, the current mini-batch size may be different from the effective mini-batch size. The result is
   stored in history.

void reset_statistics ()
   Clear all statistics.

void reset_statistics (execution_mode mode)
   Clear statistics for an execution mode.

EvalType get_mean_value (execution_mode mode) const
   Get mean metric value. If mini-batch sizes are not identical, the mean is over the sample values rather than
   over the mini-batch mean values.

int get_statistics_num_samples (execution_mode mode) const
   Get number of samples for statistics.

virtual std::vector<Layer *> get_layer_pointers () const
   Get list of pointers to layers.

virtual void set_layer_pointers (std::vector<Layer *> layers)
   Set list of pointers to layers.

EvalType get_evaluate_time () const
   Get the time spent in evaluation for this metric (const).

EvalType &get_evaluate_time ()
   Get the time spent in evaluation for this metric.

void reset_counters ()
   Reset timing counters.

virtual bool save_to_checkpoint_shared (persist &p)
   Save metric state to checkpoint.

virtual bool load_from_checkpoint_shared (persist &p)
   Load metric state from checkpoint.

Documentation of lbann::layer_metric

class layer_metric : public lbann::metric

Public Functions

layer_metric *copy () const
   Copy function.
std::string name() const
    Return a string name for this metric.

std::string get_unit() const
    Return a display unit for this metric. Default is an empty string. This is overridden if the metric has units,
    e.g. “%” or “sec”.

void set_layer(Layer &l)
    Set corresponding layer.

Layer &get_layer()
    Get corresponding layer.

const Layer &get_layer() const
    Get corresponding layer (const).

std::vector<Layer *> get_layer_pointers() const
    Get list of pointers to layers.

void set_layer_pointers(std::vector<Layer *> layers)
    Set list of pointers to layers.

5.10 Models Interface

A model is a collection of layers that are composed into a computational graph. The model also holds the weight
matrices for each learning layer. During training the weight matrices are the free parameters. For a trained network
during inference the weight matrices are preloaded from saved matrices. The model also contains the objective function
and optimizer classes for the weights.

5.10.1 Documentation of lbann::model

class model
    Abstract base class for neural network models.

    Subclassed by lbann::directed_acyclic_graph_model

Public Functions

virtual std::string get_type() const = 0
    Model type’s name.
    Should be a brief, human-readable description of the model’s architecture.

std::string get_name() const
    Model instance name.
    Each model in a trainer should have a unique, and preferably human-readable, name.

void set_name(std::string name)
    Model instance name.
    Each model in a trainer should have a unique, and preferably human-readable, name.

virtual description get_description() const
    Human-readable description.
**objective_function** *get_objective_function() const*
Mathematical function to be minimized during training.

**virtual const std::vector<metric*>& get_metrics() const**
Return the model’s metrics.

El::Int *get_num_layers() const*
Size of model’s list of layers.

**Layer & get_layer (El::Int pos)**
Parameters
- pos: Position in model’s list of layers.

**const Layer & get_layer (El::Int pos) const**
Parameters
- pos: Position in model’s list of layers.

**std::vector<Layer*>& get_layers () const**
Return list of layers in model.
The list is in execution order for forward propagation.

**const std::vector<Layer*>& get_layers () const**
Return list of layers in model.
The list is in execution order for forward propagation.

**virtual std::vector<callback_base*>&& getCallbacks()**
Get the list of callbacks for the model.

**lbann_comm *get_comm() const**
Get the model’s comm.

**bool has_valid_execution_context() const**
Check to see if there is a valid training context for the model.

**const execution_context& get_execution_context() const**
Grab the training context of the model.

**execution_context& get_execution_context()**
Grab the training context of the model.

**virtual void add_layer (std::unique_ptr<Layer> l)**
Add layer to model.

**void add_weights (std::unique_ptr<weights> w)**
Add weights to model.

**void add_callback (callback_base *cb)**
Register a new callback for the model.

**void add_metric (metric *m)**
Register a new metric for the model.

**void replace_weights (std::vector<weights*> &w)**
Replace the model’s weights.
void copy_trained_weights_from (std::vector<weights *> &w)
    Copy trained weights from input parameter w.
    Only weight values are placed, pointers and layer structure are in place. Weights to be copied are of the same name

optimizer *create_optimizer() const
    Construct an instance of the default optimizer.
    If there is no default optimizer, a null pointer is returned.

size_t get_max_mini_batch_size() const
    Get the trainer’s maximum mini-batch size.

void allow_background_io_activity (bool enable)
    Set a flag that can be used to enable / disable the background I/O activities.

bool background_io_activity_allowed ()
    Are background I/O activities enabled by the input layers.

virtual void setup ()
    Must be called after model specification and before execution.

virtual void summarize_stats (lbann_summary &summarizer)
    Summarize statistics (e.g. timers, counters); these should be computable quickly.

virtual void summarize_matrices (lbann_summary &summarizer)
    Summarize matrices (e.g. means); these are called less frequently and can be more expensive.

virtual bool save_to_checkpoint_shared (persist &p)
    Checkpoint model to given file descriptor, return number of bytes written.

virtual bool load_from_checkpoint_shared (persist &p)
    Restore model by reading checkpoint from given file descriptor, return number of bytes read.

virtual bool save_weights (persist &p)
    Save the model’s weight to file.

virtual bool reload_weights (const std::string latest, const std::vector<std::string> &weight_list)
    Reload the model’s weights from a file.

virtual bool save_model ()
    Saves the model explicitly if the save_model callback is present.

virtual void write_proto (lbann_data::Model *proto)
    Write model to proto file.

virtual void reset_mode (execution_context &context, execution_mode mode)
    Reset model pointer and execution mode.

virtual void reset_epoch_statistics (execution_mode mode)
    Reset model statistics for an epoch.

virtual bool is_execution_mode_valid (execution_mode mode) const
    Check if the trainer execution mode is valid for this model.

virtual void collect_background_data_fetch (execution_mode mode)
    Complete any background I/O data fetch for the execution mode requested.
virtual void forward_prop (execution_mode mode)
    Forward propagation step.

virtual void backward_prop ()
    Backward propagation step.

virtual void evaluate_metrics (execution_mode mode, size_t current_mini_batch_size)
    Evaluate any metrics in the model

virtual void clear_gradients ()
    Clear each optimizer’s gradient.

    This must be called before training forward prop since layers set an optimizer flag during forward prop.

virtual void update_weights ()
    Update weights step.

virtual bool update_layers ()
    Update layers step.

virtual void reconcile_weight_values ()
    Reconcile weight values.

    If weight values are duplicated across multiple processes, they are set to the average across the processes.

virtual void do_setup_end_cbs ()
    Execute callbacks at end of setup.

virtual void do_model_forward_prop_begin_cbs (execution_mode mode)
    Execute callbacks at start of model forward propagation.

virtual void do_model_forward_prop_end_cbs (execution_mode mode)
    Execute callbacks at end of model forward propagation.

virtual void do_layer_forward_prop_begin_cbs (execution_mode mode, Layer *l)
    Execute callbacks at start of layer forward propagation.

virtual void do_layer_forward_prop_end_cbs (execution_mode mode, Layer *l)
    Execute callbacks at end of layer forward propagation.

virtual void do_model_backward_prop_begin_cbs ()
    Execute callbacks at start of model backward propagation.

virtual void do_model_backward_prop_end_cbs ()
    Execute callbacks at end of model backward propagation.

virtual void do_layer_backward_prop_begin_cbs (Layer *l)
    Execute callbacks at start of layer backward propagation.

virtual void do_layer_backward_prop_end_cbs (Layer *l)
    Execute callbacks at end of layer backward propagation.

virtual void do_model_optimize_begin_cbs ()
    Execute callbacks at start of model optimization.

virtual void do_model_optimize_end_cbs ()
    Execute callbacks at end of model optimization.

virtual void do_weight_optimize_begin_cbs (weights *w)
    Execute callbacks at the start of weight optimization.
virtual void do_weight_optimize_end_cbs (weights *w)
    Execute callbacks at the end of weight optimization.

Documentation of lbann::directed_acyclic_graph_model

class directed_acyclic_graph_model : public lbann::model
    Neural network model with a DAG layer graph.

Public Functions

std::string get_type () const
    Model type's name.
    Should be a brief, human-readable description of the model’s architecture.

5.11 Objective Function Interface

An objective function is the measure that training attempts to optimize. Objective functions are defined in a user’s model definition prototext file. Available objective functions can be found below.

5.11.1 Documentation of lbann::objective_function_term

class objective_function_term
    Abstract class for objective function terms.
    Subclassed by lbann::l2_weight_regularization, lbann::layer_term

Public Functions

objective_function_term (EvalType scale_factor = EvalType(1))
    Default constructor.

objective_function_term (const objective_function_term &other)
    Copy constructor.

objective_function_term &operator=(const objective_function_term &other)
    Copy assignment operator.

virtual ~objective_function_term ()
    Destructor.

virtual objective_function_term *copy () const = 0
    Copy function.

virtual std::string name () const = 0
    Get the name of the objective function term.

virtual void setup (model &m)
    Setup objective function term.
virtual void start_evaluation() = 0
Start evaluation of the objective function term. This should include the scaling factor. The result is not available until finish_evaluation has been called.

virtual EvalType finish_evaluation() = 0
Complete evaluation of the objective function term.

virtual void differentiate() = 0
Compute the gradient of the objective function term. The gradient is computed w.r.t. the objective function term inputs. This should include the scaling factor.

virtual void compute_weight_regularization() = 0
Compute the gradient of the weight regularization term. The gradient is computed w.r.t. the weights.

std::vector<Layer*> get_layer_pointers() const
Get list of pointers to layers.

void set_layer_pointers(std::vector<Layer*> layers)
Set list of pointers to layers.

std::vector<weights*> get_weights_pointers() const
Get list of pointers to weights.

void set_weights_pointers(std::vector<weights*> w)
Set list of pointers to weights.

Documentation of lbann::l2_weight_regularization

class l2_weight_regularization : public lbann::objective_function_term
Apply L2 regularization to a set of weights.
Given a weights tensor \( w \),
\[
L2(w) = \frac{1}{2} \sum_i w(i)^2
\]
Note the \( 1/2 \) scaling factor.

Public Functions

l2_weight_regularization(EvalType scale_factor = 1)
Parameters
- scale_factor: The objective function term is scale_factor \( \times \sum L2(w_i) \)

l2_weight_regularization *copy() const
Copy function.

std::string name() const
Get the name of the objective function term.

void setup(model &m)
Setup objective function term.

void start_evaluation()
Start evaluation of the objective function term. This should include the scaling factor. The result is not available until finish_evaluation has been called.
EvalType **finish_evaluation**()
Complete evaluation of the objective function term.

```cpp
void **differentiate**()
Compute the gradient w.r.t. the activations.

L2 regularization is independent of forward prop output, so nothing needs to be done here.

```cpp
void **compute_weight_regularization**()
Compute the gradient w.r.t. the weights.
```

\[ \nabla_w L2(w) = w \n\]

**Documentation of lbann::layer_term**

class **layer_term** : public lbann::**objective_function_term**

**Public Functions**

```cpp
layer_term * **copy** () const
Copy function.
```

```cpp
std::string **name** () const
Get the name of the objective function term.
```

```cpp
void **set_layer** (Layer &l)
Set corresponding layer.
```

```cpp
Layer & **get_layer** ()
Get corresponding layer.
```

```cpp
const Layer & **get_layer** () const
Get corresponding layer (const).
```

```cpp
void **setup** (model &m)
Setup objective function term.
```

```cpp
void **start_evaluation**()
Start evaluation of the objective function term. This should include the scaling factor. The result is not available until finish_evaluation has been called.
```

EvalType **finish_evaluation**()
Complete evaluation of the objective function term.

```cpp
void **differentiate**()
Compute the gradient of the objective function term. The gradient is computed w.r.t. the objective function term inputs. This should include the scaling factor.
```

```cpp
void **compute_weight_regularization**()
Compute the gradient of the weight regularization term. The gradient is computed w.r.t. the weights.
```
5.11.2 Documentation of lbann::objective_function

```cpp
class objective_function
Objective function class.

Public Functions

objective_function()
Default constructor.

objective_function(const objective_function &other)
Copy constructor.

objective_function &operator=(const objective_function &other)
Copy assignment operator.

~objective_function()
Destructor.

objective_function *copy() const
Copy function.

void add_term(objective_function_term *term)
Add a term to the objective function. The objective function takes ownership of the objective function term and deallocates it during destruction.

std::vector<objective_function_term *> get_terms()
Get list of objective function terms.

void setup(model &m)
Setup objective function.

void start_evaluation(execution_mode mode, int mini_batch_size)
Start evaluating the objective function. This function takes the model’s current mini-batch size. If multiple models are being trained, the current mini-batch size may be different from the effective mini-batch size. The result is not guaranteed to be available until finish_evaluation is called.

EvalType finish_evaluation(execution_mode mode, int mini_batch_size)
Complete evaluation of the objective function. The result is stored in history.

void differentiate()
Compute the objective function gradient. The gradient is with respect to the objective function inputs.

void compute_weight_regularization()
Compute the gradient of the weight regularization term. The gradient is computed w.r.t. the weights.

void reset_statistics()
Clear all statistics.

void reset_statistics(execution_mode mode)
Clear statistics for an execution mode.

EvalType get_mean_value(execution_mode mode) const
Get mean objective function value. This is a weighted average such that each mini-batch sample makes an equal contribution.
```
int get_statistics_num_samples (execution_mode mode) const
  Get number of samples for statistics.

std::vector<Layer *> get_layer_pointers () const
  Get list of pointers to layers.

void set_layer_pointers (std::vector<Layer *> layers)
  Set list of pointers to layers.

std::vector<weights *> get_weights_pointers () const
  Get list of pointers to weights.

void set_weights_pointers (std::vector<weights *> w)
  Set list of pointers to weights.

EvalType get_evaluation_time () const
  Get the time spent evaluating the objective function.

EvalType get_differentiation_time () const
  Get the time spent computing the objective function gradient.

void reset_counters ()
  Reset time counters.

5.11.3 Objective Functions for Weight Regularization

TODO: Something about objective_functions/weight_regularization

5.12 Optimizer Interface

Optimizer algorithms attempt to optimize model weights. Optimizers are selected when invoking LBANN via a command line argument (--optimizer=<path_top_opt_proto>). Available optimizers are found below.

5.12.1 Documentation of lbann::optimizer

class optimizer
  Abstract base class for gradient-based optimization algorithms.

  Uses a variant of stochastic gradient descent to optimize the values in a weights instance. The weights values are iteratively adjusted to minimize an objective function. Each optimization step requires the objective function gradient w.r.t. the weights.

  Subclassed by lbann::adagrad, lbann::adam, lbann::hypergradient_adam, lbann::rmsprop, lbann::sgd

Public Functions

virtual optimizer *copy () const = 0
  Create a copy of the class instance.

  The caller is responsible for deallocating the returned object.

virtual std::string get_type () const = 0
  Human-readable type name.
virtual description get_description() const
    Human-readable description.

weights &get_weights()
    Weights being optimized.

const weights &get_weights() const
    Weights being optimized.

void set_weights(weights *w)
    Weights being optimized.

AbsDistMat &get_gradient()
    Objective function gradient w.r.t. the weights.
    An allreduce may be launched and/or synchronized if needed.

void add_to_gradient(const AbsDistMat &gradient, DataType scale = DataType(1), bool allreduce_needed = false)
    Add to the objective function gradient w.r.t. the weights.

Parameters
• gradient: Contribution to gradient.
• scale: Scaling factor for gradient contribution.
• allreduce_needed: Whether the gradient contribution requires an allreduce over its redundant communicator. If false, duplicated data (over the redundant communicator) is assumed to be identical. If true, an allreduce is performed lazily when the gradient is accessed.

void clear_gradient()
    Zero out the objective function gradient w.r.t. the weights.

AbsDistMat &get_gradient_buffer(DataType &buf_scale, DataType &in_scale, bool allreduce_needed = false)
    Get the gradient buffer.
    This provides access to the underlying gradient buffer, which may be directly summed into. This buffer should be considered ephemeral and not stored. The caller must also ensure the buffer has an appropriate distribution. buf_scale provides the caller with a scale factor that must be applied to the gradient buffer before writing to it, and in_scale provides a scaling factor that must be applied to the user’s data. Essentially, this enables computations of the form gradient = buf_scale*gradient + in_scale*new_gradient. This is an expert-mode function and is intended to help eliminate copies and facilitate kernel fusion.

Parameters
• buf_scale: A scale factor provided to the caller to scale the returned buffer by.
• in_scale: A scale factor provided to the caller to scale their gradient contributions by.
• allreduce_needed: Whether this gradient contribution will need to be allreduced.

El::Int get_num_gradient_sources() const
    Objects that are expected to contribute to the gradient.

void add_gradient_source(const void *source)
    Register a gradient source.
    Any object that uses the weights and influences the objective function is expected to contribute to the objective function gradient. These objects should register themselves during forward prop.
void remove_gradient_source(const void *source)
Unregister a gradient source.

When an object adds its contribution to the objective function gradient during back prop, it should unreg-
ister itself. If there are no more gradient sources remaining, a non-blocking allreduce will be launched on
the gradient, if needed.

virtual void setup(weights *w = nullptr)
Must be called before training.

Parameters

• w: Weights being optimized. If null, no change is made to the weights.

void step()
Optimization step.

lbann_comm &get_comm()
LBANN communicator.

const lbann_comm &get_comm() const
LBANN communicator.

DataType get_learning_rate() const
Scaling factor for optimization step sizes.

void set_learning_rate(DataType learning_rate)
Scaling factor for optimization step sizes.

EvalType get_step_time() const
Time spent in optimization step.

virtual void reset_counters()
Reset stats counters.

Documentation of lbann::adagrad

class adagrad: public lbann::optimizer
AdaGrad optimizer.

Reference:
John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient
methods for online learning and stochastic optimization.” Journal of Machine Learning Research 12, no. Jul

Public Functions

adagrad *copy() const
Create a copy of the class instance.

The caller is responsible for deallocating the returned object.

std::string get_type() const
Human-readable type name.
**LBANN Documentation**

```cpp
description get_description() const
    Human-readable description.

void setup (weights *w = nullptr)
    Must be called before training.

Parameters
    • w: Weights being optimized. If null, no change is made to the weights.
```

**Documentation of lbann::adam**

```cpp
class adam : public lbann::optimizer
    Adam optimizer.

Reference:
```

**Life cycle functions**

```cpp
adam *copy() const
    Create a copy of the class instance.
    The caller is responsible for deallocating the returned object.
```

**Descriptions**

```cpp
std::string get_type() const
    Human-readable type name.

description get_description() const
    Human-readable description.
```

**Access functions**

```cpp
DataType get_beta1() const
    Update factor for first moment estimate.

void set_beta1 (DataType beta1)
    Update factor for first moment estimate.

DataType get_beta2() const
    Update factor for second moment estimate.

void set_beta2 (DataType beta2)
    Update factor for second moment estimate.

DataType get_eps() const
    Small factor to avoid division by zero.

void set_eps (DataType eps)
    Small factor to avoid division by zero.
```
const AbsDistMat &get_moment1 () const
    First moment estimates.

AbsDistMat &get_moment1 ()
    First moment estimates.

const AbsDistMat &get_moment2 () const
    Second moment estimates.

AbsDistMat &get_moment2 ()
    Second moment estimates.

DataType get_current_beta1 () const
    beta1 ^ iteration.

void set_current_beta1 (DataType current_beta1)
    beta1 ^ iteration.

DataType get_current_beta2 () const
    beta2 ^ iteration.

void set_current_beta2 (DataType current_beta2)
    beta2 ^ iteration.

Setup

void setup (weights *w = nullptr)
    Must be called before training.

Parameters
    • w: Weights being optimized. If null, no change is made to the weights.

Friends

friend lbann::adam::callback::perturb_adam
    Hyperparameter exploration.

Documentation of lbann::hypergradient_adam

class hypergradient_adam : public lbann::optimizer
    Hypergradient Adam optimizer.

Reference:

Public Functions

hypergradient_adam (DataType init_learning_rate = 1e-3, DataType hyper_learning_rate = 1e-7,
    DataType beta1 = 0.9, DataType beta2 = 0.99, DataType eps = 1e-8)
    Construct a Hypergradient Adam optimizer object.
Parameters

- `comm`: Communication context for this object
- `init_learning_rate`: Initial Adam learning rate (0.001 is reasonable).
- `hyper_learning_rate`: Hypergradient learning rate.
- `beta1`: Decay rate for the first moment moving average.
- `beta2`: Decay rate for the second moment moving average.
- `eps`: Small factor to avoid division by zero.

`hypergradient_adam *copy () const`

Create a copy of the class instance.

The caller is responsible for deallocating the returned object.

`std::string get_type () const`

Human-readable type name.

`description get_description () const`

Human-readable description.

`void setup (weights *w = nullptr)`

Must be called before training.

Parameters

- `w`: Weights being optimized. If null, no change is made to the weights.

Documentation of `lbann::rmsprop`

class `rmsprop`: public `lbann::optimizer`

RMSprop optimizer.


Public Functions

`rmsprop *copy () const`

Create a copy of the class instance.

The caller is responsible for deallocating the returned object.

`std::string get_type () const`

Human-readable type name.

`description get_description () const`

Human-readable description.

`void setup (weights *w = nullptr)`

Must be called before training.

Parameters

- `w`: Weights being optimized. If null, no change is made to the weights.
Documentation of lbann::sgd

class sgd : public lbann::optimizer
Stochastic gradient descent optimizer.
Supports momentum and Nesterov acceleration.

Life cycle functions

sgd *copy() const
Create a copy of the class instance.
The caller is responsible for deallocating the returned object.

Descriptions

std::string get_type() const
Human-readable type name.

description get_description() const
Human-readable description.

Access functions

DataType get_momentum() const
Decay rate for gradient accumulation.
A momentum of zero corresponds to vanilla SGD.

void set_momentum(DataType momentum)
Decay rate for gradient accumulation.
A momentum of zero corresponds to vanilla SGD.

bool using_nesterov() const
Whether Nesterov acceleration is applied.

void set_nesterov(bool nesterov)
Whether Nesterov acceleration is applied.

const AbsDistMat &get_velocity() const
Accumulated gradients for momentum optimizer.

AbsDistMat &get_velocity()
Accumulated gradients for momentum optimizer.

Setup

void setup(weights *w = nullptr)
Must be called before training.

Parameters

• w: Weights being optimized. If null, no change is made to the weights.
5.13 Trainer Interface

A trainer is a collection of compute resources and defines an explicit communication domain. It manages the execution for both the training and inference of a trained model. Once constructed, a trainer owns an `lbann_comm` object that defines both intra- and inter-trainer communication domains. Additionally, a trainer will contain an I/O thread pool that is used to fetch and preprocess data that will be provided to the trainer’s models.

A trainer owns:

- `lbann_comm` object,
- I/O thread pool,
- One or more models, and
- Execution context for each model.

In the future, it will also contain the data readers.

5.13.1 Documentation of `lbann::trainer`

```cpp
class trainer
  Represents an LBANN trainer and its context.

Public Functions

trainer (lbann_comm *comm)
  Constructor.

trainer (const trainer &other)
  Copy constructor.

trainer &operator= (const trainer &other)
  Copy assignment operator.

~trainer ()
  Destructor.

void set_name (std::string const &name)
  Set the trainer’s name; this is an arbitrary string that may be useful in multi-trainer scenarios, e.g. LTFB, jag

std::string get_name () const
  Return the trainer’s name; this is an arbitrary string that may be useful in multi-trainer scenarios, e.g. LTFB, jag

description get_description () const
  Human-readable description.

void setup (std::unique_ptr<thread_pool> io_thread_pool)
  Set up the trainer.

thread_pool &get_io_thread_pool () const
  Return the I/O thread pool

lbann_comm *get_comm () const
  Get the trainer’s comm.
```
void allow_background_io_activity (bool enable)
    Set a flag that can be used to enable / disable the background I/O activities

bool background_io_activity_allowed ()
    Are background I/O activities enabled by the input layers

5.14 Training Algorithm Interface

The training algorithm defines the optimization that is to be applied to the model(s) being trained. Additionally, it can specify how to evaluate the model.

5.14.1 Documentation of lbann::training_algorithm

class training_algorithm
    Base class for LBANN training_algorithms.

    Subclassed by lbann::sgd_training_algorithm

Public Functions

training_algorithm ()
    Constructor.

training_algorithm (const training_algorithm &other)
    Copy constructor.

training_algorithm &operator= (const training_algorithm &other)
    Copy assignment operator.

training_algorithm (training_algorithm &&other)
    Move constructor.

training_algorithm &operator= (training_algorithm &&other)
    Move assignment operator.

virtual ~training_algorithm ()
    Destructor.

virtual void apply (execution_context &context, model &model, execution_mode mode, termination_criteria const &term_criteria) = 0
    Copy training_algorithm.

Documentation of lbann::sgd_training_algorithm

class sgd_training_algorithm : public lbann::training_algorithm
    Base class for LBANN SGD-family training algorithms.
Public Functions

`sgd_training_algorithm()`
Constructor.

`sgd_training_algorithm(const sgd_training_algorithm &other)`
Copy constructor.

`sgd_training_algorithm &operator=(const sgd_training_algorithm &other)`
Copy assignment operator.

`sgd_training_algorithm(sgd_training_algorithm &&other)`
Move constructor.

`sgd_training_algorithm &operator=(sgd_training_algorithm &&other)`
Move assignment operator.

`virtual ~sgd_training_algorithm()`
Destructor.

`void apply(execution_context &c, model &model, execution_mode mode, termination_criteria const &term_criteria)`
Copy `training_algorithm`. Apply the training algorithm to the model with the provided context and execution mode.

`void train(sgd_execution_context &c, model &model, size_t num_epochs, size_t num_batches = 0)`
Train a model using an iterative SGD solver.

`void evaluate(sgd_execution_context &c, model &model, execution_mode mode, size_t num_batches = 0)`
Evaluate a model using the forward pass of an SGD solver.

5.15 Transform Interface

5.15.1 Documentation of lbann::transform::transform

class transform
Abstract base class for transforms on data.

A transform takes a CPU Mat and modifies it in-place. Transforms should be thread-safe, as one instance of a transform may be called concurrently within multiple threads.

Because transforms may switch between underlying data types throughout the pipeline, everything is done in terms of a `type_erased_matrix`, which can swap between underlying data types.

Subclassed by `lbann::transform::adjust_brightness`, `lbann::transform::adjust_contrast`, `lbann::transform::adjust_saturation`, `lbann::transform::center_crop`, `lbann::transform::color_jitter`, `lbann::transform::colorize`, `lbann::transform::cutout`, `lbann::transform::grayscale`, `lbann::transform::horizontal_flip`, `lbann::transform::normalize`, `lbann::transform::normalize_to_lbann_layout`, `lbann::transform::random_affine`, `lbann::transform::random_crop`, `lbann::transform::random_resized_crop`, `lbann::transform::random_resized_crop_with_fixed_aspect_ratio`, `lbann::transform::repack_HWC_to_CHW_layout`, `lbann::transform::resize`, `lbann::transform::resized_center_crop`, `lbann::transform::sample_normalize`, `lbann::transform::scale`, `lbann::transform::scale_and_translate`, `lbann::transform::to_lbann_layout`, `lbann::transform::vertical_flip`
Public Functions

\texttt{virtual transform *\textit{copy}() const = 0}
Create a copy of the transform instance.

\texttt{virtual std::string \textit{get\_type}() const = 0}
Human-readable type name.

\texttt{virtual description \textit{get\_description}() const}
Human-readable description.

\texttt{virtual bool \textit{supports\_non\_inplace}() const}
True if the transform supports non-in-place apply.

\texttt{virtual void \textit{apply}(\texttt{utils::type\_erased\_matrix \&data, std::vector<\_t \&dims}) = 0}
Apply the transform to data.

\texttt{Note} dims is a hack until we have proper tensors.

Parameters

\begin{itemize}
  \item \texttt{data}: The input data to transform, which is modified in-place. The matrix should be contiguous.
  \item \texttt{dims}: The dimensions of the data tensor. For "plain data", dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
\end{itemize}

\texttt{virtual void \textit{apply}(\texttt{utils::type\_erased\_matrix \&data, CPUMat \&out, std::vector<\_t \&dims})}
Apply the transform to data. This does not modify data in-place but places its output in out.

Documentation of \texttt{lbann::transform::adjust\_brightness}

class \texttt{adjust\_brightness} : public \texttt{lbann::transform::transform}
Adjust the brightness of an image.

Public Functions

\texttt{adjust\_brightness(float \textit{factor})}
Adjust brightness with given factor.

Parameters

\begin{itemize}
  \item \texttt{factor}: A non-negative factor. 0 gives a black image, 1 the original.
\end{itemize}

\texttt{transform *\textit{copy}() const}
Create a copy of the transform instance.

\texttt{std::string \textit{get\_type}() const}
Human-readable type name.

\texttt{void \textit{apply}(\texttt{utils::type\_erased\_matrix \&data, std::vector<\_t \&dims})}
Apply the transform to data.

\texttt{Note} dims is a hack until we have proper tensors.

Parameters

\begin{itemize}
  \item \texttt{data}: The input data to transform, which is modified in-place. The matrix should be contiguous.
  \item \texttt{dims}: The dimensions of the data tensor. For "plain data", dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
\end{itemize}
Documentation of \texttt{lbann::transform::adjust\_contrast}

\textbf{class} \texttt{adjust\_contrast : public lbann::transform::\texttt{transform}}

Adjust the contrast of an image.

This operates similarly to the contrast control on a television.

\textbf{Public Functions}

\texttt{adjust\_contrast (float factor)}

Adjust contrast with given factor.

\textbf{Parameters}

- \texttt{factor}: A non-negative factor. 0 gives a solid grey image, 1 the original.

\texttt{transform \texttt{*copy () const}}

Create a copy of the transform instance.

\texttt{std::string \texttt{get\_type () const}}

Human-readable type name.

\texttt{void apply (utils::type\_erased\_matrix \&data, std::vector<size\_t> \&dims)}

Apply the transform to data.

\textbf{Note} dims is a hack until we have proper tensors.

\textbf{Parameters}

- \texttt{data}: The input data to transform, which is modified in-place. The matrix should be contiguous.
- \texttt{dims}: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

Documentation of \texttt{lbann::transform::adjust\_saturation}

\textbf{class} \texttt{adjust\_saturation : public lbann::transform::\texttt{transform}}

Adjust the saturation of an image.

This operates similarly to the controls on a color television (as opposed to a direct adjustment of saturation) by interpolating between the original value and its grayscale value.

\textbf{Public Functions}

\texttt{adjust\_saturation (float factor)}

Adjust saturation with given factor.

\textbf{Parameters}

- \texttt{factor}: A non-negative factor. 0 gives a grayscale image, 1 the original.

\texttt{transform \texttt{*copy () const}}

Create a copy of the transform instance.

\texttt{std::string \texttt{get\_type () const}}

Human-readable type name.
void **apply** (utils::**type_erased_matrix** &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note**  dims is a hack until we have proper tensors.

**Parameters**

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

**Documentation of lbann::transform::center_crop**

class  **center_crop** : public  lbann::transform::transform

Crop an image at the center.

**Public Functions**

center_crop(size_t h, size_t w)

Crop to an h x w image.

transform *copy() const

Create a copy of the transform instance.

std::string get_type() const

Human-readable type name.

void apply (utils::**type_erased_matrix** &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note**  dims is a hack until we have proper tensors.

**Parameters**

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

**Documentation of lbann::transform::color_jitter**

class  **color_jitter** : public  lbann::transform::transform

Randomly change brightness, contrast, and saturation. This randomly adjusts brightness, contrast, and saturation, in a random order.

**Public Functions**

color_jitter(float min_brightness_factor, float max_brightness_factor, float min_contrast_factor,

float max_contrast_factor, float min_saturation_factor, float max_saturation_factor)

Randomly adjust brightness, contrast, and saturation within given ranges. Set both min and max to 0 to disable that adjustment.

**Parameters**

- **min_brightness_factor**: Minimum brightness adjustment (>= 0).
• **max_brightness_factor**: Maximum brightness adjustment.
• **min_contrast_factor**: Minimum contrast adjustment (>= 0).
• **max_contrast_factor**: Maximum contrast adjustment.
• **min_saturation_factor**: Minimum saturation adjustment (>= 0).
• **max_saturation_factor**: Maximum saturation adjustment.

```cpp
transform *copy() const
    Create a copy of the transform instance.
```

```cpp
std::string get_type() const
    Human-readable type name.
```

```cpp
void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
    Apply the transform to data.
    
    **Note**  dims is a hack until we have proper tensors.

    **Parameters**
    
    • **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
    • **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
```

**Documentation of lbann::transform::colorize**

```cpp
class colorize : public lbann::transform::transform
    Convert an image from grayscale to color.
```

**Public Functions**

```cpp
transform *copy() const
    Create a copy of the transform instance.
```

```cpp
std::string get_type() const
    Human-readable type name.
```

```cpp
void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
    Apply the transform to data.
    
    **Note**  dims is a hack until we have proper tensors.

    **Parameters**
    
    • **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
    • **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
```

**Documentation of lbann::transform::cutout**

```cpp
class cutout : public lbann::transform::transform
    Cutout data augmentation which randomly masks out square regions of input.
```

**See:**

This will randomly select a center pixel for each square and set all pixels within that square to 0. It is permissible for portions of the masks to lie outside of the image.

Normalization about 0 should be applied after applying cutout.

**Public Functions**

**cutout** (size_t num_holes, size_t length)
Cutout with a given number of squares of a given size.

**Parameters**
- num_holes: Number of squares to mask out (must be positive).
- length: Length of a side of the square (must be positive).

transform *copy() const
Create a copy of the transform instance.

std::string get_type() const
Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
Apply the transform to data.

*Note* dims is a hack until we have proper tensors.

**Public Functions**

**grayscale**
Convert an image to grayscale.

**Public Functions**

transform *copy() const
Create a copy of the transform instance.

std::string get_type() const
Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
Apply the transform to data.

*Note* dims is a hack until we have proper tensors.
• **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

### Documentation of `lbann::transform::horizontal_flip`

**class horizontal_flip : public lbann::transform::transform**

Horizontally flip image data with given probability.

**Public Functions**

```
horizontal_flip (float p)
   Flip image with probability p.
```

```
transform *copy () const
   Create a copy of the transform instance.
```

```
std::string get_type () const
   Human-readable type name.
```

```
void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
   Apply the transform to data.
```

**Note** dims is a hack until we have proper tensors.

**Parameters**

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

### Documentation of `lbann::transform::normalize`

**class normalize : public lbann::transform::transform**

Normalize with mean and standard deviation. This is done channel-wise for images. If the input does not have channels, (e.g. it is not an image), it is treated as having one “channel”. This is only applicable after conversion to an LBANN CPUMat.

**Public Functions**

```
normalize (std::vector<float> means, std::vector<float> stds)
   Apply channel-wise means and standard deviations.
```

```
transform *copy () const
   Create a copy of the transform instance.
```

```
std::string get_type () const
   Human-readable type name.
```

```
bool supports_non_inplace () const
   True if the transform supports non-in-place apply.
```

```
void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
   Apply the transform to data.
```
Note dims is a hack until we have proper tensors.

Parameters

• data: The input data to transform, which is modified in-place. The matrix should be contiguous.
• dims: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

void apply (utils::type_erased_matrix &data, CPUMat &out, std::vector<size_t> &dims)
Apply the transform to data. This does not modify data in-place but places its output in out.

Documentation of lbann::transform::normalize_to_lbann_layout

class normalize_to_lbann_layout : public lbann::transform::transform
Normalize and convert data to LBANN’s native data layout. Currently only supports converting from OpenCV layouts. This normalizes with provided channel-wise means and standard deviations, scales from [0, 255] to [0, 1], and converts to LBANN’s data layout. Normalization is applied after the scaling to [0, 1]. This essentially fuses the to_lbann_layout and normalize transforms.

Public Functions

normalize_to_lbann_layout (std::vector<float> means, std::vector<float> stds)
Apply channel-wise means and standard deviations.

transform *copy () const
Create a copy of the transform instance.

std::string get_type () const
Human-readable type name.

bool supports_non_inplace () const
True if the transform supports non-in-place apply.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
Apply the transform to data.

Note dims is a hack until we have proper tensors.

Parameters

• data: The input data to transform, which is modified in-place. The matrix should be contiguous.
• dims: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

void apply (utils::type_erased_matrix &data, CPUMat &out, std::vector<size_t> &dims)
Apply the transform to data. This does not modify data in-place but places its output in out.

Documentation of lbann::transform::random_affine

class random_affine : public lbann::transform::transform
Apply a random affine transform to an image.
Public Functions

**random_affine** (float *rotate_min*, float *rotate_max*, float *translate_h*, float *translate_w*, float *scale_min*, float *scale_max*, float *shear_min*, float *shear_max*)

Set up the affine transform. Rotate a random number of degrees selected in [rotate_min, rotate_max]. Translate the vertical dimension in a random amount in [-h*translate_h, h*translate_h], and the horizontal dimension in [-w*translate_w, w*translate_w]. Scale by a random amount in [scale_min, scale_max]. Shear by a random number of degrees in [shear_min, shear_max]. Set arguments to 0 to disable that transform.

transform *copy () const

Create a copy of the transform instance.

std::string get_type () const

Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note** dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

Documentation of lbann::transform::random_crop

class random_crop : public lbann::transform::transform

Crop an image at a random location.

Public Functions

**random_crop** (size_t *h*, size_t *w*)

Crop to an h x w image.

transform *copy () const

Create a copy of the transform instance.

std::string get_type () const

Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note** dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
Documentation of lbann::transform::random_resized_crop

class random_resized_crop : public lbann::transform::transform
Extract a crop of random size and aspect ratio, then crop to a size. This is commonly used for Inception-style networks and some other image classification networks.

Public Functions

random_resized_crop (size_t h, size_t w, float scale_min = 0.08, float scale_max = 1.0, float ar_min = 0.75, float ar_max = 4.0f / 3.0f)
Crop to a random size and aspect ratio, then resize to h x w. The random crop has area in [scale_min, scale_max] of the original image area, and aspect ratio in [ar_min, ar_max] of the original. This random crop is then resized to be h x w. These default to (0.08, 1.0) and (3/4, 4/3), respectively, which are the standard.

transform *copy () const
Create a copy of the transform instance.

std::string get_type () const
Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
Apply the transform to data.

Note dims is a hack until we have proper tensors.

Parameters

• data: The input data to transform, which is modified in-place. The matrix should be contiguous.
• dims: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

Documentation of lbann::transform::random_resized_crop_with_fixed_aspect_ratio

class random_resized_crop_with_fixed_aspect_ratio : public lbann::transform::transform
Resize an image then extract a random crop.

Public Functions

random_resized_crop_with_fixed_aspect_ratio (size_t h, size_t w, size_t crop_h, size_t crop_w)
Resize to h x w, then extract a random crop_h x crop_w crop.

transform *copy () const
Create a copy of the transform instance.

std::string get_type () const
Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)
Apply the transform to data.

Note dims is a hack until we have proper tensors.

Parameters
• **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.

• **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

Documentation of `lbann::transform::repack_HWC_to_CHW_layout`

class `repack_HWC_to_CHW_layout` : public `lbann::transform::transform`

Convert data to LBANN’s native data layout. Currently only supports converting from and interleaved channel format.

**Public Functions**

transform *`copy` () const

Create a copy of the transform instance.

std::string `get_type` () const

Human-readable type name.

bool `supports_non_inplace` () const

True if the transform supports non-in-place apply.

void `apply` (utils::type_erased_matrix &`data`, std::vector<size_t> &`dims`)

Apply the transform to data.

**Parameters**

• **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.

• **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

void `apply` (utils::type_erased_matrix &`data`, CPUMat &`out`, std::vector<size_t> &`dims`)

Apply the transform to data. This does not modify data in-place but places its output in `out`.

Documentation of `lbann::transform::resize`

class `resize` : public `lbann::transform::transform`

Resize an image.

**Public Functions**

`resize` (size_t `h`, size_t `w`)

Resize to `h` x `w`.

transform *`copy` () const

Create a copy of the transform instance.

std::string `get_type` () const

Human-readable type name.

void `apply` (utils::type_erased_matrix &`data`, CPUMat &`out`, std::vector<size_t> &`dims`)

Apply the transform to data.
Note dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

Documentation of lbann::transform::resized_center_crop

**class resized_center_crop : public** lbann::transform::transform

Resize an image and then crop its center.

**Public Functions**

```cpp
class resized_center_crop { public: lbann::transform::transform;

resized_center_crop(size_t h, size_t w, size_t crop_h, size_t crop_w) 
    Resize to h x w, then extract a crop_h x crop_w crop from the center.

transform *copy() const
    Create a copy of the transform instance.

std::string get_type() const
    Human-readable type name.

void apply(util::type_erased_matrix &data, std::vector<size_t> &dims)
    Apply the transform to data.

    Note dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
```

Documentation of lbann::transform::sample_normalize

**class sample_normalize : public** lbann::transform::transform

Normalize to have mean 0, standard deviation 1. This only works after conversion to an LBANN CPUMat.

**Public Functions**

```cpp
class sample_normalize { public: lbann::transform::transform;

sample_normalize() { }

sample_normalize(const sample_normalize &) = default;

sample_normalize(sample_normalize &&) = default;

sample_normalize & operator=(const sample_normalize &);

sample_normalize & operator=(sample_normalize &&);

```

```cpp
transform *copy() const
    Create a copy of the transform instance.

std::string get_type() const
    Human-readable type name.

void apply(util::type_erased_matrix &data, std::vector<size_t> &dims)
    Apply the transform to data.

    Note dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.
- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

### Documentation of lbann::transform::scale

class scale : public lbann::transform::transform

Scale data by a constant.

**Public Functions**

scale (float scale_val)

Scale all data by scale_val.

transform *copy () const

Create a copy of the transform instance.

std::string get_type () const

Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note** dims is a hack until we have proper tensors.

**Parameters**

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

### Documentation of lbann::transform::scale_and_translate

class scale_and_translate : public lbann::transform::transform

Scale and Translate data by a constant pair of constants.

**Public Functions**

scale_and_translate (float scale_val, float translate_val)

Scale_And_Translate all data by scale_and_translate_val.

transform *copy () const

Create a copy of the transform instance.

std::string get_type () const

Human-readable type name.

void apply (utils::type_erased_matrix &data, std::vector<size_t> &dims)

Apply the transform to data.

**Note** dims is a hack until we have proper tensors.

**Parameters**
• **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.

• **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

### Documentation of lbann::transform::to_lbann_layout

**class to_lbann_layout : public lbann::transform::transform**

Convert data to LBANN’s native data layout. Currently only supports converting from OpenCV layouts. This will also rescale data from [0, 255] to [0, 1].

#### Public Functions

transform *`copy` () const
Create a copy of the transform instance.

std::string `get_type` () const
Human-readable type name.

bool `supports_non_inplace` () const
True if the transform supports non-in-place apply.

void `apply` (utils::type_erased_matrix &`data`, std::vector<size_t> &`dims`
Apply the transform to data.

**Note**: dims is a hack until we have proper tensors.

**Parameters**

• **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.

• **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

void `apply` (utils::type_erased_matrix &`data`, CPUMat &`out`, std::vector<size_t> &`dims`
Apply the transform to data. This does not modify data in-place but places its output in out.

### Documentation of lbann::transform::vertical_flip

**class vertical_flip : public lbann::transform::transform**

Vertically flip image data with given probability.

#### Public Functions

`vertical_flip` (float `p`)
Flip image with probability p.

transform *`copy` () const
Create a copy of the transform instance.

std::string `get_type` () const
Human-readable type name.

void `apply` (utils::type_erased_matrix &`data`, std::vector<size_t> &`dims`
Apply the transform to data.
Note  dims is a hack until we have proper tensors.

Parameters

- **data**: The input data to transform, which is modified in-place. The matrix should be contiguous.
- **dims**: The dimensions of the data tensor. For “plain data”, dims should have one entry, giving its size. For images, dims should have three entries: channels, height, width.

5.15.2 Documentation of lbann::transform::transform_pipeline

**class transform_pipeline**  
Applies a sequence of transforms to input data.

**Public Functions**

- `void add_transform(std::unique_ptr<transform> &trans)`  
  Add trans as the next transform to apply.

- `void set_expected_out_dims(std::vector<size_t> expected_out_dims)`  
  Set the expected dimensions of the data after applying the transforms. This is primarily meant as a debugging aid/sanity check.

- `void apply(utils::type_erased_matrix &data, std::vector<size_t> &dims)`  
  Apply the transforms to data.

  **Parameters**

  - **data**: The data to transform. data will be modified in-place.
  - **dims**: Dimensions of data. Will be modified in-place.

- `void apply(CPUMat &data, std::vector<size_t> &dims)`  
  Apply to CPU Mat data, which will be modified in-place.

- `void apply(El::Matrix<uint8_t> &data, CPUMat &out_data, std::vector<size_t> &dims)`  
  Apply the transforms to data.

  **Parameters**

  - **data**: The data to transform. Will be modified in-place.
  - **out_data**: Output will be placed here. It will not be reallocated.
  - **dims**: Dimensions of data. Will be modified in-place.

5.15.3 vision

5.16 General Utilities

Utility classes and functions.
5.16.1 Documentation of lbann::utils::any

class any

Type-erasure class to store any object of copy-constructible type.

This class is (mostly) API-compatible with std::any. The most notable omission is the std::in_place_type_t
overloads of the constructor (std::in_place_type_t is also C++17, and I don’t want to implement the whole
standard). For best results, do not attempt to use those in this code. For even better results (yes, better than best.
English is overrated), incessantly remind your friends, colleagues, and, most importantly, vendors that it’s 2019,
that 2019 > 2017, and that there are excellent free compilers in the world until they concede to updating to a
modern compiler and this implementation can be banished to the depths.

Constructors and destructor

any ()

Default construct an empty “any”.

template<typename T>
any (T &&obj)

Construct an object holding a T.

any(any const &other)

Copy construct from another container.

Makes a copy of the held object.

any(any &&other)

Move construct from another container.

~any ()

Default destructor.

Assignment operator

any &operator= (any const &other)

Copy assign from another container.

Makes a deep copy of the held object.

any &operator= (any &&other)

Move assign from another container.

Modifiers

template<typename T, typename ... Args>
auto emplace (Args&... args)

Change the contained object to one of type T.

Any held object is destroyed and the new object is emplace-constructed from the arguments given.

Return A reference to the newly constructed object

Template Parameters

• T: The type of the new held object
• **Args:** (Deduced) types of arguments to the T constructor

**Parameters**
- **args:** The arguments to the T constructor

```cpp
void reset() 
    Reset the container to an empty state, destroying the held object.

void swap(any &other)
    Swap the contents of this container with another.
```

**Observers**

```cpp
bool has_value() const
    Test whether the container holds a value.

std::type_info const &type() const
    Get the type_info object for the held type.
```

### 5.16.2 Documentation of lbann::beta_distribution

```cpp
template<typename RealType = double>
class beta_distribution
    Produces random floating point values drawn from a Beta distribution with parameters a > 0 and b > 0.
```

See:

```url
https://en.wikipedia.org/wiki/Beta_distribution
```

for more details.

### 5.16.3 Documentation of lbann::dataset

```cpp
class dataset
```

### 5.16.4 Documentation of lbann::default_key_error_policy

```cpp
template<typename KeyT, class ObjectT>
class default_key_error_policy
    Default policy describing how to handle unknown keys.
    The policy must define “handle_unknown_key(KeyT const&)”.
    The default behavior is to throw an exception.
```

**Template Parameters**
- **KeyT:** The type of key.
- **ObjectT:** The type of the object being constructed by the factory.
5.16.5 Documentation of lbann::description

class description
Generates nicely formatted description messages.

Messages have hanging indentation and can be output to an output stream like `std::cout`. For example:

```
Title
Some numerical field: 12.3
A boolean parameter: true
Miscellaneous statement
```

Public Functions

description (std::string title = "")

Parameters

- title: First line in description message.

void set_title (std::string title)

Set first line in description message.

void add (std::string line)

Add new line.

template<typename T>
void add (std::string field, T value)

Add new line describing a field value.

The line is formatted:

```
<field> : <value>
```

void add (const description & desc)

Insert a nested description.

The indentation in `desc` is combined with the current indentation. For instance:

```
Outer description
Some numerical field: 12.3
Nested description
This: abc
That: 123
```

Friends

std::ostream & operator<< (std::ostream & os, const description & desc)

Print description to stream.

5.16.6 Documentation of lbann::enum_iterator

template<
typename C, 
C beginVal, C endVal>

class enum_iterator
Create an iterator that goes over a contiguous (unit-step) enum class.
5.16.7 Documentation of lbann::lbann_summary

class lbann_summary
  Dummy class when TBinf is not present.

5.16.8 Documentation of lbann::nullptr_key_error_policy

template<typename KeyT, class ObjectT>
class nullptr_key_error_policy
  Policy returning a nullptr if the key is unknown.

  This class just returns “nullptr”. Use of this class is not recommended as it probably indicates bad design that
  would better utilize exception handling. But it felt awkward to not at least provide it.

  Template Parameters
  • KeyT: The type of key.
  • ObjectT: The type of the object being constructed by the factory.

5.16.9 Documentation of lbann::options

class options
  Singleton, globally accessible class, for setting and retrieving options (key/value pairs).

  Unnamed Group

  bool has_int (std::string option)
    Returns true if the database contains the option

  Unnamed Group

  void set_option (std::string name, int value)
    insert option in database; if option already exists it’s value will changed

  Unnamed Group

  int get_int (std::string option)
    returns the value of the option; throws exception if option doesn’t exist, or if it’s associated value (which
    is stored internally as a string) cannot be cast to the specified return type

  Unnamed Group

  int get_int (std::string option, int the_default)
    returns the value of the option; if option isn’t found, inserts the option in the internal db, with the specified
    default value, and returns the the_default
Public Functions

void init (int argc, char **argv)
   if cmd line contains “–loadme=<string>” then initialize options database from that file. Next, intialize from all other options on cmd line.

void print (std::ostream &out = std::cout)
   prints all registered options to ‘out’

Public Static Functions

static options *get ()
   returns a pointer to the Options singleton

5.16.10 Documentation of lbann::beta_distribution::param_type

class param_type

5.16.11 Documentation of lbann::rng

template<typename DistType, typename DType = DataType>
class rng

5.16.12 Documentation of lbann::stack_profiler

class stack_profiler
   This is a singleton, globally accessible class, for recording stack traces and timing

   Public Functions

   void activate (int thread_id)
      Turns on stack profiling if you have passed the cmd line option: st_on or st_on=1

5.16.13 Documentation of lbann::utils::type_erased_matrix

class type_erased_matrix
   A type-erased wrapper around an El::Matrix<T,Device::CPU>

   Warning  This class is an implementation detail of the preprocessing pipeline and should not be used in general LBANN code.

   Public Functions

   template<typename Field>
   type_erased_matrix (El::Matrix<Field> const &in_matrix)
      Construct from a copy of a given matrix.
      Deep-copy the input matrix into the held matrix.
Warning  This performs a deep copy of the matrix.

**Template Parameters**

- **Field**: The data type of the input matrix

**Parameters**

- **in_matrix**: The input matrix.

```cpp
template<typename Field>
type_erased_matrix(El::Matrix<Field> &&in_matrix)
    Construct by moving the given matrix into type-erased storage.
    Move the input matrix into the held matrix.
```

**Template Parameters**

- **Field**: The data type of the input matrix

**Parameters**

- **in_matrix**: The input matrix.

```cpp
template<typename Field>
El::Matrix<Field> &get()
    Access the underlying matrix.
    Provides read/write access to the underlying matrix if the input Field matches the data type of the held matrix.
```

**Template Parameters**

- **Field**: The data type of the held matrix

**Exceptions**

- **bad_any_cast**: If the datatype of the held matrix does not match the input Field.

```cpp
template<typename Field>
El::Matrix<Field> const &get() const
    Access the underlying matrix.
    Provides read-only access to the underlying matrix if the input Field matches the data type of the held matrix.
```

**Return**  Reference to the underlying matrix

**Template Parameters**

- **Field**: The data type of the held matrix

**Exceptions**

- **bad_any_cast**: If the datatype of the held matrix does not match the input Field.

```cpp
template<typename Field, typename ...Args>
El::Matrix<Field> &emplace(Args&&... args)
    Replace the held matrix with a new one constructed in-place from the arguments.
```

**Return**  Reference to the new underlying matrix
Template Parameters

- **Field**: The data type of the newly held matrix

Parameters

- **args**: The arguments with which to construct the new matrix.

### 5.16.14 Multithreading Utilities

TODO: Something about utils/threads

**Documentation of lbann::thread_pool**

```cpp
class thread_pool

Public Functions

thread_pool ()
    Construct an empty threadpool. Size must be set with launch().

thread_pool (size_type max_threads)
    Construct a threadpool of a given size.

Parameters

- **max_threads**: Total threads available. max_threads-1 worker threads will be launched.

~thread_pool ()
    Destroy the threadpool.

void launch_threads (size_type num_threads)
    Launch the threads.

void launch_pinned_threads (size_type num_threads, int cpu_offset)
    Launch the threads and pin them to the Hyperthreaded cores.

void reap_threads ()
    Wake and terminate all threads in the pool

void relaunch_pinned_threads (size_type num_threads)
    Reap all threads in the pool and relaunch pinned threads

template<typename FunctionT>
std::future<typename std::result_of<FunctionT()>::type> submit_job
    Submit a job to the pool’s queue.

template<typename FunctionT>
void submit_job_to_work_group (FunctionT func)
    Submit a job to the pool’s queue and place the future into a work group.

bool finish_work_group ()
    Wait for all of the jobs in a work group to finish.

size_type get_num_threads () const
    Query the number of worker threads actually present.
```
int get_local_thread_id()
   Convert the C++ thread id into a local thread pool id.

int get_threads_offset()
   Convert the C++ thread id into a local thread pool id.

Documentation of lbann::thread_safe_queue

template<typename T>
class thread_safe_queue
   A queue that is safe for multiple threads to push to or pull from “simultaneously”.
   This version uses locks.
   This is essentially a fancy linked-list implementation that enables finer-grained locks than simply wrapping an
   std::queue. The trade-off is two locks, one for the front and one for the back of the list.

Template Parameters
   • T: A move- or copy-constructible type

Public Functions

thread_safe_queue()
   Default constructor; creates an empty queue.

void push (T value)
   Adds a value to back of the queue.

void set_stop_threads (bool flag)
   Allow the thread pool to set / reset the flags.

std::unique_ptr<T> try_pop ()
   Try to remove the first value from the queue.
   Return nullptr if empty(); otherwise return a value

std::unique_ptr<T> wait_and_pop ()
   Wait for data and then return it.

bool empty () const
   Check if queue is empty.

Documentation of lbann::type_erased_function

class type_erased_function
   A move-only callable type for wrapping functions.

Deleted functions

type_erased_function()
   Deleted constructor.
**type_erased_function** (const type_erased_function &other)
Deleted copy constructor.

**type_erased_function &operator= (const type_erased_function &other)**
Deleted copy assignment.

**Public Functions**

template<typename FunctionT>
**type_erased_function** (FunctionT &&F)
Erase the type of input function F.

type_erased_function (type_erased_function &&other)
Move constructor.

type_erased_function &operator= (type_erased_function &&other)
Move assignment.

void operator() ()
Make the function callable.

5.17 Weights Interface

The weight class is the representation of the trainable parameters in the neural network. Learning layers each have an independent weight class. During stochastic gradient descent training the weight matrices are updated after each forward and backward propagation step.

5.17.1 Documentation of lbann::variance_scaling_initializer

class variance_scaling_initializer: public lbann::weights_initializer
Generalization of “Xavier” initialization.

Weights values are randomly sampled from a probability distribution with a variance determined by a “fan-in” and a “fan-out” parameter.

Weights with variance scaling initialization are only compatible with layers that set fan-in and fan-out parameters, e.g. the convolution and fully-connected layers.

Subclassed by lbann::glorot_initializer, lbann::he_initializer, lbann::lecun_initializer

**Public Functions**

description get_description () const
Human-readable description of class instance.

void fill (AbsDistMat &matrix)
Initialize entries in a weights matrix.

void set_fan_in (El::Int fan_in)
Set fan-in parameter.

void set_fan_out (El::Int fan_out)
Set fan-out parameter.
Documentation of `lbann::glorot_initializer`

```cpp
class glorot_initializer : public lbann::variance_scaling_initializer

Fill weights with variance of 2 / (fan-in + fan-out).
Also called Xavier initialization.
```

**Public Functions**

```cpp
glorot_initializer *copy() const
Create a copy.
```

```cpp
std::string get_type() const
Human-readable string describing concrete class.
```

Documentation of `lbann::he_initializer`

```cpp
class he_initializer : public lbann::variance_scaling_initializer

Fill weights with variance of 2 / fan-in.
```

**Public Functions**

```cpp
he_initializer *copy() const
Create a copy.
```

```cpp
std::string get_type() const
Human-readable string describing concrete class.
```

Documentation of `lbann::lecun_initializer`

```cpp
class lecun_initializer : public lbann::variance_scaling_initializer

Fill weights with variance of 1 / fan-in.
```

**Public Functions**

```cpp
lecun_initializer *copy() const
Create a copy.
```

```cpp
std::string get_type() const
Human-readable string describing concrete class.
```

5.17.2 Documentation of `lbann::weights_initializer`

```cpp
class weights_initializer
Scheme for initializing weight values.
Subclassed by `lbann::constant_initializer`, `lbann::normal_initializer`, `lbann::uniform_initializer`, `lbann::value_initializer`, `lbann::variance_scaling_initializer`
```
Public Functions

```cpp
virtual std::string get_type() const = 0
    Human-readable string describing concrete class.

virtual std::string get_description() const
    Human-readable description of class instance.

virtual weights_initializer* copy() const = 0
    Create a copy.

virtual std::string get_type() const
    Human-readable string describing concrete class.

description get_description() const
    Human-readable description of class instance.

void fill(AbsDistMat &matrix)
    Initialize entries in a weights matrix.
```

Documentation of lbann::constant_initializer

```cpp
class constant_initializer : public lbann::weights_initializer
    Fill weights with a constant value.

Public Functions

constant_initializer* copy() const
    Create a copy.

std::string get_type() const
    Human-readable string describing concrete class.

description get_description() const
    Human-readable description of class instance.

void fill(AbsDistMat &matrix)
    Initialize entries in a weights matrix.
```

Documentation of lbann::normal_initializer

```cpp
class normal_initializer : public lbann::weights_initializer
    Draw weights values from a normal random distribution.

Public Functions

normal_initializer* copy() const
    Create a copy.

std::string get_type() const
    Human-readable string describing concrete class.

description get_description() const
    Human-readable description of class instance.

void fill(AbsDistMat &matrix)
    Initialize entries in a weights matrix.
```
Documentation of `lbann::uniform_initializer`

class uniform_initializer : public lbann::weights_initializer

Draw weights values from a uniform random distribution.

Public Functions

uniform_initializer *copy() const
    Create a copy.

std::string get_type() const
    Human-readable string describing concrete class.

description get_description() const
    Human-readable description of class instance.

void fill(AbsDistMat &matrix)
    Initialize entries in a weights matrix.

Documentation of `lbann::value_initializer`

class value_initializer : public lbann::weights_initializer

Fill weights with values from a list.

The number of weight entries must exactly match the number of provided values.

Public Functions

value_initializer *copy() const
    Create a copy.

std::string get_type() const
    Human-readable string describing concrete class.

void fill(AbsDistMat &matrix)
    Initialize entries in a weights matrix.

5.17.3 Documentation of `lbann::weights`

class weights

Neural network weights. Weights are tensors that act as trainable parameters for a neural network. The values can be initialized with a weights initializer and are optimized with first-order methods (e.g. stochastic gradient descent).

Internally, the weight values are stored in a 2D distributed matrix. The “matrix height dimensions” are tensor dimensions that correspond to the matrix height. The remaining dimensions, the “matrix width dimensions,” correspond to the matrix width.

Note that LBANN weights are similar to Tensorflow variables and Caffe parameters.
Public Functions

void set_name (std::string name)
    Set weights name. Each set of weights in a model should have a unique, human-readable name.

std::string get_name () const
    Get weights name.

weights *copy () const
    Create a copy of the weights. This function dynamically allocates memory for a weights instance and
    instantiates a copy. The caller is responsible for deallocating the instance.

description get_description () const
    Human-readable description.

std::vector<int> get_dims () const
    Get weight tensor dimensions. The dimensions are sorted in decreasing order of the data strides. This is a
generalization of the “NCHW/NHWC” notation commonly used to describe image data.

    These dimensions are obtained by concatenating the matrix width dimensions with the matrix height di-
mensions (in that order). If the weight matrix is duplicated on all processes (i.e. in STAR,STAR layout),
the tensor data is packed w.r.t. the matrix height dimensions. If the local matrices are also fully packed,
the tensor data is fully packed.

int get_size () const
    Get number of entries in weight tensor.

std::vector<int> get_matrix_height_dims () const
    Get tensor dimensions corresponding to weight matrix height. The dimensions are sorted in decreasing
order of strides. Matrix rows are fully-packed w.r.t. the matrix height dimensions.

std::vector<int> get_matrix_width_dims () const
    Get tensor dimensions corresponding to weight matrix width. The dimensions are sorted in decreasing
order of strides. Matrix columns are fully-packed w.r.t. the matrix width dimensions.

int get_matrix_height () const
    Get weight matrix height. If there are no matrix height dimensions, the height is one.

int get_matrix_width () const
    Get weight matrix width. If there are no matrix width dimensions, the width is one.

void set_dims (std::vector<int> matrix_height_dims, std::vector<int> matrix_width_dims =
    std::vector<int>())
    Set weight tensor dimensions. See the ‘get_dims’ function for an explanation of the notation.

void set_dims (int size)
    Set weight tensor dimensions as a 1D tensor.

weights_initializer *get_initializer()
    Get weights initializer.

const weights_initializer *get_initializer () const
    Get weights initializer (const).

void set_initializer (std::unique_ptr<weights_initializer> &init)
    Set weights initializer. The contents of ‘init’ are moved to a class member.

optimizer *get_optimizer ()
    Get weights optimizer. Returns a null pointer if the weights are frozen.
const optimizer *get_optimizer() const
    Get weights optimizer. Returns a null pointer if the weights are frozen.

void set_optimizer(std::unique_ptr<optimizer> &opt)
    Set weights optimizer. The contents of opt are moved to a class member.

AbsDistMat &get_values() const
    Get the weight matrix.

const AbsDistMat &get_values() const
    Get the weight matrix.

void set_values(const AbsDistMat &values)
    Set the weight matrix.

void set_value(DataType value, int index)
    Set a weight value.

void set_value(DataType value, std::vector<int> pos)
    Set an entry in the weight tensor.

void set_value(DataType value, int row, int col)
    Set an entry in the weight matrix.

void reconcile_values()
    Reconcile weight values. If weight values are duplicated across multiple processes, they are set to the average across the processes.

void reconcile_values(Al::request &req)
    Asynchronously reconcile weight values. If weight values are duplicated across multiple processes, they are set to the average across the processes.

void freeze()
    Disable weight optimization.

void unfreeze()
    Enable weight optimization.

bool is_frozen() const
    Whether weight optimization is enabled.

void write_proto(lbann_data::WeightsData *proto) const
    Write weights to proto file
6.1 In-Source Documentation

In-source documentation should be written using Doxygen. LBANN will use C-style Doxygen comments (/** @brief A short comment */) and with ampersats (@details) instead of backslashes to denote directives. The aim is maximal grep-ability and readability of the source code. Using C-style comments for Doxygen helps differentiate quickly between C-style Doxygen and C++-style source-only documentation.

Note: C-style comments on classes and functions default to @details, not @brief, even for one-line comments. Be sure to add @brief when appropriate.

6.1.1 Documentation of Functions

Every function should be decorated with the maximally applicable set of the following:

- @brief: A short description of the class. May span multiple lines if necessary for maintaining line character limits.

- @details: Begin a detailed description of the function. This is not explicitly needed if a blank line is inserted between the @brief and the body of the @details section.

- @param <name> Describe a parameter to the function. It may be helpful to annotate with @param[in], @param[out], or @param[in,out] if not clear from the types. Repeat this directive for each applicable parameter.

- @tparam <name> Describe a template parameter. This can be useful for explaining any assumptions (or even better, static assertions) about satisfied concepts/predicates. Repeat this directive for each applicable template parameter.

- @returns Describe the return value of the function. This is not needed for trivial “getters”.

- @throws <exception> Indicate an exception that may be thrown. It is not expected that every possible exception (e.g., those coming from corner-cases of the STL) be documented. However, if a function’s implementation has an explicit throw statement, the exception should be noted with this directive. Repeat this directive for each applicable exception.

- @pre Description of preconditions. This is most useful for functions that use in/out parameters or those that require various conditions on objects in the case of member functions. Repeat this directive for each precondition.

- @post Description of postconditions. This is most useful for functions that use in/out parameters or those that require various conditions on objects in the case of member functions. Repeat this directive for each postcondition.
Some hypothetical examples of appropriately marked up functions are:

```cpp
/** @brief Does a foo.
 *  * These are details.
 *  * @tparam T The type of parameter. Must implement `operator+=`
 *  * @param param This is a parameter. It says how to foo.
 *  * @throws crazy_error If a crazy error occurs.
 *  * @pre param is not foo'd yet
 *  * @post param has been foo'd
 */
template<typename T>
void foo(T& param);

/** @brief Computes a result.
 *  * @details The algorithm is simple $A+B$.
 *  * @param A the first value
 *  * @param B the second value
 *  * @returns The output of the complicated algorithm
 */
int compute_result(int A, int B) noexcept;
```

### 6.1.2 Documentation of Classes

Every class should be decorated with the maximally applicable set of the following:

- **@brief**: A short description of the class. May span multiple lines if necessary for maintaining line character limits.

- **@details**: Begin a detailed description of the function. This is not explicitly needed if a blank line is inserted between the @brief and the body of the @details section.

- **@tparam <name>**: Describe a template parameter. This can be useful for explaining any assumptions (or even better, static assertions) about satisfied concepts/predicates. Repeat this directive for each applicable template parameter.

- **@name <name>,@{,*}**: Group members into named sections.

Member functions are functions and should be documented as above. An example of a completely marked up file is `include/lbann/utils/any.hpp`. 
Bamboo is the continuous integration (CI) framework we use. A Bamboo plan consists of stages (which run sequentially), which consist of jobs (which run in parallel), which consist of tasks (which run sequentially).

The LBANN build project has many plans. Two plans run off of LLNL/lbann/develop - Nightly Develop and Weekly Develop. Nightly Develop runs every night (except Saturday) at midnight. Weekly Develop runs every Saturday at midnight. The other plans in the build project are for each individual LBANN developer’s fork of LBANN.

All plans run off the latest pushed commits to the repository. That means if you have local commits that you have not pushed to your fork, these commits will not be tested by Bamboo. If you have pushed commits to your fork but have not merged your branch into the main repository’s “develop”, your commits will be tested on your individual plan, but not on Nightly Develop or Weekly Develop.

### 7.1 Plan Configuration

Each plan is identical (except Weekly Develop, which will be explained below). The plans consist of a single stage “Tests”. The stage consists of two jobs - “x86_cpu” (Catalyst), and “x86_gpu” (Pascal). Each of these jobs can run in parallel. They consist of an identical list of tasks:

1. Checkout Default Repository (checkout the repository)
2. Run `./allocate_and_run.sh`; Weekly Develop adds the `--weekly` option. This script allocates nodes and then runs “run.sh” which does the following:
   a. Remove Generated Files (each build creates a large number of files. We may look at these files between builds, so we cannot delete them at the end of a build. So, instead we delete them before doing any real work in the next build. This also ensures the generated files came from the latest build and not a previous build).
   b. Compiler Tests (run tests in “bamboo/compiler_tests”)
   c. Integration Tests (run tests in “bamboo/integration_tests”)
   d. Unit Tests (run tests in “bamboo/unit_tests”)
3. JUnit Parser (this allows Bamboo to render test results in a nice UI)

The tests in Task 2 run `$PYTHON -m pytest -s -vv --durations=0 [--weekly] --junitxml=results.xml`, which will run all the pytests in the job’s associated directory. Note that `$PYTHON` refers to the Python build to use. Also note that only Weekly Develop adds the `--weekly` option. Many (mostly longer-running) tests are set to not run unless this option is on. Weekly Develop runs a superset of the tests that Nightly Develop runs.
7.2 Directory Structure

“bamboo/compiler_tests”, “bamboo/integration_tests”, “bamboo/unit_tests” each have a “conftest.py” that pytest requires. They also contain one or more python files. Each of these files have a number of tests to run.

7.3 Writing Your Own Tests

A side effect of our Bamboo setup is that tests must be written using pytest. Test files must begin with test_ to be recognized by pytest. Individual test methods must also begin with test_. Test methods should use the assert keyword or raise an AssertionError. A test will only fail if the assertion turns out to be false. Not putting an assertion will automatically cause the test to pass.

How then to test non-Python code? You can just wrap your test with Python. A test can be as simple as asserting the output code of a shell command is 0. The output code of a command can be found using Python’s os.system().

7.4 Running Tests On Your Individual Plan

Unlike Nightly Develop, the individual plans are triggered to run by polling your fork for commits. They do not run nightly. If you push new commits to your fork, a new build should start automatically. You can also manually start a build by navigating to your individual plan and clicking Run > Run plan (this will say “Run branch” if you have plan branches set up). Once again, keep in mind that the tests will run off what has been pushed to your GitHub fork of LBANN and not your local copy of the LBANN repository.

Plan branches allow you to test multiple branches simultaneously instead of simply testing “<fork-name>/develop”. You can create plan branches by navigating to your individual plan, clicking Actions > Configure plan > Branches > Create plan branch.

7.5 Navigating Bamboo

From the LBANN Project Summary, click on a plan. From there, click on a build (builds are listed under “Recent History” and can also be accessed from the pass/fail marks in the top right, to the left of the “Run” button). This will bring you to a certain build’s page. The most relevant tabs are “Tests” and “Logs”. It is recommended to look at failures first in the “Tests” tab, as the build logs can be difficult to parse through. The build’s “Tests” tab shows “New test failures”, “Existing test failures”, “Fixed tests”, and “Skipped Tests”.

From the build’s page, you can also click on individual jobs, which have the same tabs. The “Tests” tabs of the individual jobs have two sub-tabs, “Failed tests” and “Successful tests”. They do not display skipped tests. The Bamboo agent that ran the job can be found by looking at the “Agent” field under the “Job Summary” tab. Alternatively, you can determine the agent from one of the first lines in the build logs: “Build working directory is /usr/workspace/wsb/lbannusr/bamboo/<bamboo-agent-name>/xml-data/build-dir/<build-plan-and-job>”.

7.6 Bamboo Agent Properties

Bamboo agent properties are used to specify requirements for each job.
Currently, “agent_owner”, “architecture”, and “gpu_architecture” are used to determine agents to run a job.

### 7.7 Running Tests From The Command Line

Navigate to “bamboo/compiler_tests”, “bamboo/integration_tests”, or “bamboo/unit_tests”.

To run all the tests in a subdirectory: python -m pytest -s --weekly. Note that running all tests can take a substantial amount of time.

To run the tests that Nightly Develop or the individual plans run in a subdirectory: python -m pytest -s.

To run a specific test file: python -m pytest -s <test_file>.py.

To run a specific test: python -m pytest -s <test_file>.py -k ‘<test_name>’.

Most integration and unit tests allow for running a test with a different executable. The convention is to have a similarly structured test replacing `<compiler_name>` with `_exe`. These tests are set to be skipped in Bamboo, but can be run locally. There should be a line above the test that gives the command to run the test locally, likely in the following form: python -m pytest -s <test_file>.py -k ‘<test_name>’ --exe=<executable>.

If you have an executable, you can run the _exe tests with local_test.sh. Use local_test.cmd as a template for writing a batch script. You can run only integration tests, only unit tests, or both.

### 7.8 Helpful Files

First, run sudo lbannusr.

To look at output and error from previous builds: cd /usr/workspace/wsb/lbannusr/bamboo/<bamboo-agent-name>/xml-data/build-dir/<build-plan-and-job>/bamboo/<compiler_tests, integration_tests, or unit_tests>/error_or_output.

To look at archived results from previous builds: cd /usr/workspace/wsb/lbannusr/archives/<build-plan>

To look at Bamboo agent properties: cat /usr/global/tools/bamboo/agents/lbannusr/<bamboo-agent-name>/bin/bamboo-capabilities.properties

You can copy these files over to your own machine as follows:

- sudo lbannusr

---

**Agents (jobs)** | **agent_owner** | **architecture** | **cluster** | **gpu_architecture** | **sys_type**
--- | --- | --- | --- | --- | ---
Catalyst Agents (x86_cpu) | lbannusr | x86_64 | catalyst | none | toss_3_x86_64_ib
Corona Agents (x86_cpu_corona) | lbannusr | x86_64 | corona | none | toss_3_x86_64_ib
Lassen Agents (ppc64le_gpu) | lbannusr | ppc64le |lassen | volta | blueos_3_ppc64le_ib_p9
Pascal Agents (x86_gpu_pascal) | lbannusr | x86_64 | pascal |pascal | chaos_6_x86_64_ib
Ray Agents (ppc64le_gpu) | lbannusr | ppc64le | ray | pascal | blueos_3_ppc64le_ib
• give <lc-username> <absolute-path>

• exit - to go back to your own LC account, not lbannusr's.

• take lbannusr - now the file exists on your LC account, but not yet on your own machine.

From your own machine, not a ssh terminal:

• scp <lc-username>@<cluster>.llnl.gov:<absolute-path> .
CHAPTER

EIGHT

LBANN DOCUMENTATION BUILDING

**Warning:** Some of the directions in this section are Mac-specific.

### 8.1 Adding Documentation Outside Code

1. Create a file such as “new_docs.rst” in “lbann/docs”.
2. Add “new_docs” (no “.rst”) to the appropriate documentation block in “lbann/docs/index.rst”.
3. Look at the other “.rst” files in “lbann/docs” to see how to get certain formatting.
4. When you want to see how your code looks, you have a couple options:
   a. Push your docs to your fork/branch on GitHub and look at how the text renders. This is a very simplified look compared to Read-the-Docs.
   b. From “lbann/docs” run `make html` and then `open -a <preferred web browser> _build/html/index.html`. This is exactly how the docs will look.
5. Merge your code into “lbann/develop” and then have someone with correct permissions on Read-the-Docs update the official docs.

### 8.2 Making The Build Work

In order to make `make html` work, you may need to do a few steps:

1. Run `pip3 install sphinx breathe sphinx-rtd-theme`.
2. Download Doxygen by going to the Doxygen downloads page, downloading “Doxygen-1.8.15.dmg”, and dragging the app to the “Applications” folder.
3. Determine the directory Doxygen is in by running `which Doxygen`. If nothing is returned, see if `doxygen` is in “/Applications/Doxygen.app/Contents/Resources” or “/Applications/Doxygen.app/Contents/MacOS”.
4. Add Doxygen to your path with `PATH=“<doxygen directory>:${PATH}”`. You may want to add this to your “~/.bash_profile” so your PATH is always correct. Run `source ~/.bash_profile` to run that code.
5. Try running `make html` again.

• genindex
INDEX

lbann::abstract_evaluation_layer (C++ class), 127
lbann::abstract_evaluation_layer::construct (C++ function), 127
lbann::abstract_evaluation_layer::get_scale (C++ function), 127
lbann::abstract_evaluation_layer::get_value (C++ function), 127
lbann::abstract_evaluation_layer::set_scale (C++ function), 127
lbann::adagrad (C++ class), 155
lbann::adagrad::copy (C++ function), 155
lbann::adagrad::get_description (C++ function), 155
lbann::adagrad::get_type (C++ function), 155
lbann::adagrad::setup (C++ function), 156
lbann::adam (C++ class), 156
lbann::adam::copy (C++ function), 156
lbann::adam::get_beta1 (C++ function), 156
lbann::adam::get_beta2 (C++ function), 156
lbann::adam::get_current_beta1 (C++ function), 157
lbann::adam::get_current_beta2 (C++ function), 157
lbann::adam::get_description (C++ function), 156
lbann::adam::get_eps (C++ function), 156
lbann::adam::get_moment1 (C++ function), 157
lbann::adam::get_moment2 (C++ function), 157
lbann::adam::get_type (C++ function), 156
lbann::adam::set_beta1 (C++ function), 156
lbann::adam::set_beta2 (C++ function), 156
lbann::adam::set_current_beta1 (C++ function), 157
lbann::adam::set_current_beta2 (C++ function), 157
lbann::adam::set_eps (C++ function), 156
lbann::adam::setup (C++ function), 157
lbann::Al::dummy_backend (C++ class), 29
lbann::argmax_layer (C++ class), 99
lbann::argmax_layer::copy (C++ function), 100
lbann::argmax_layer::get_data_layout (C++ function), 100
lbann::argmax_layer::get_device_allocation (C++ function), 100
lbann::argmax_layer::get_type (C++ function), 100
lbann::argmin_layer (C++ class), 100
lbann::argmin_layer::copy (C++ function), 100
lbann::argmin_layer::get_data_layout (C++ function), 100
lbann::argmin_layer::get_device_allocation (C++ function), 100
lbann::argmin_layer::get_type (C++ function), 100
lbann::base_convolution_layer (C++ class), 100
lbann::base_convolution_layer::base_convolution_layer (C++ function), 101
lbann::base_convolution_layer::get_description (C++ function), 101
lbann::base_convolution_layer::setup_data (C++ function), 101
lbann::base_convolution_layer::setup_dims (C++ function), 101
lbann::base_convolution_layer::setup_gpu (C++ function), 101
lbann::batch_normalization_layer (C++ class), 121
lbann::batch_normalization_layer::batch_normalization (C++ function), 121
lbann::batch_normalization_layer::copy (C++ function), 121
lbann::batch_normalization_layer::get_data_layout (C++ function), 121
lbann::batch_normalization_layer::get_description (C++ function), 121
lbann::batch_normalization_layer::get_device_allocation (C++ function), 121
lbann::batch_normalization_layer::get_type (C++ function), 121
class lbann::callback::confusion_matrix::name (C++ function), 43
lbann::callback::confusion_matrix::on_batch_begin (C++ function), 43
lbann::callback::confusion_matrix::on_batch_evaluate_begin (C++ function), 43
lbann::callback::confusion_matrix::on_batch_evaluate_end (C++ function), 43
lbann::callback::confusion_matrix::on_epoch_begin (C++ function), 43
lbann::callback::confusion_matrix::on_epoch_end (C++ function), 43
lbann::callback::confusion_matrix::on_test_begin (C++ function), 43
lbann::callback::confusion_matrix::on_test_end (C++ function), 43
lbann::callback::confusion_matrix::on_validation_begin (C++ function), 43
lbann::callback::confusion_matrix::on_validation_end (C++ function), 43
lbann::callback::confusion_matrix::setup (C++ function), 43
lbann::callback::debug (C++ class), 43
lbann::callback::debug::debug (C++ function), 43
lbann::callback::debug::name (C++ function), 43
lbann::callback::debug::on_backward_prop_begin (C++ function), 43
lbann::callback::debug::on_backward_prop_end (C++ function), 43
lbann::callback::debug::on_batch_begin (C++ function), 43
lbann::callback::debug::on_batch_end (C++ function), 43
lbann::callback::debug::on_batch_evaluate_begin (C++ function), 43
lbann::callback::debug::on_batch_evaluate_end (C++ function), 43
lbann::callback::debug::on_evaluate_forward_prop_begin (C++ function), 43
lbann::callback::debug::on_evaluate_forward_prop_end (C++ function), 43
lbann::callback::debug::on_optimize_begin (C++ function), 43
lbann::callback::debug::on_optimize_end (C++ function), 43
lbann::callback::debug::on_test_begin (C++ function), 43
lbann::callback::debug::on_test_end (C++ function), 43
lbann::callback::debug::on_validation_begin (C++ function), 43
lbann::callback::debug::on_validation_end (C++ function), 43
lbann::callback::debug_io (C++ class), 44
lbann::callback::debug_io::debug_io (C++ function), 44
lbann::callback::debug_io::name (C++ function), 44
lbann::callback::debug_io::on_epoch_begin (C++ function), 44
lbann::callback::debug_io::on_evaluate_forward_prop_begin (C++ function), 44
lbann::callback::debug_io::on_forward_prop_begin (C++ function), 44
lbann::callback::debug_io::on_forward_prop_end (C++ function), 44
lbann::callback::debug_io::on_optimize_begin (C++ function), 44
lbann::callback::debug_io::on_optimize_end (C++ function), 44
lbann::callback::debug_io::on_test_begin (C++ function), 44
lbann::callback::debug_io::on_test_end (C++ function), 44
lbann::callback::debug_io::on_validation_begin (C++ function), 44
lbann::callback::debug_io::on_validation_end (C++ function), 44
lbann::callback::dump_error_signals (C++ class), 45
lbann::callback::dump_error_signals::dump_error_signals (C++ function), 45
lbann::callback::dump_error_signals::name (C++ function), 45
lbann::callback::dump_error_signals::on_backward_prop_end (C++ function), 45
lbann::callback::dump_gradients (C++ class), 46
lbann::callback::dump_gradients::dump_gradients (C++ function), 46
lbann::callback::dump_gradients::name (C++ function), 46
lbann::callback::dump_gradients::on_backward_prop_end (C++ function), 46
lbann::callback::dump_minibatch_sample_indices (C++ class), 47
lbann::callback::dump_minibatch_sample_indices::dump_minibatch_sample_indices (C++ function), 47
lbann::callback::dump_minibatch_sample_indices::name (C++ function), 47
lbann::callback::dump_outputs (C++ class), 47
lbann::callback::dump_outputs::dump_outputs (C++ function), 47
lbann::callback::dump_outputs::name (C++ function), 47
lbann::callback::dump_outputs::on_evaluate_forward_prop_begin (C++ function), 47
lbann::callback::dump_outputs::on_forward_prop_end
(C++ function), 47
lbann::callback::dump_outputs::on_backward_prop_end
(C++ function), 47
lbann::callback::dump_weights (C++ class), 47
lbann::callback::dump_weights::dump_weights
(C++ function), 47
lbann::callback::dump_weights::name
(C++ function), 47
lbann::callback::dump_weights::on_epoch_end
(C++ function), 47
lbann::callback::dump_weights::on_train_begin
(C++ function), 47
lbann::callback::early_stopping (C++ class), 47
lbann::callback::early_stopping::early_stopping
(C++ function), 47
lbann::callback::early_stopping::name
(C++ function), 47
lbann::callback::early_stopping::on_validation_end
(C++ function), 47
lbann::callback::early_stopping::on_train_begin
(C++ function), 47
lbann::callback::early_stopping (C++ function), 50
lbann::callback::early_stopping::linear_growth_learning_rate
(C++ function), 50
lbann::callback::early_stopping::name
(C++ function), 50
lbann::callback::early_stopping::on_backward_prop_end
(C++ function), 50
lbann::callback::early_stopping::on_epoch_end
(C++ function), 50
lbann::callback::early_stopping::on_train_begin
(C++ function), 50
lbann::callback::linear_growth_learning_rate
(C++ class), 50
lbann::callback::linear_growth_learning_rate::linear_growth_learning_rate
(C++ function), 50
lbann::callback::linear_growth_learning_rate::name
(C++ function), 50
lbann::callback::linear_growth_learning_rate::setup
(C++ function), 50
lbann::callback::ltfb (C++ class), 51
lbann::callback::ltfb::checkpoint_file
(C++ enumerator), 52
lbann::callback::ltfb::communication_algorithm
(C++ enum), 52
lbann::callback::ltfb::ltfb
(C++ function), 53
lbann::callback::ltfb::name
(C++ function), 53
lbann::callback::ltfb::on_batch_begin
(C++ function), 53
lbann::callback::ltfb::on_train_begin
(C++ function), 53
lbann::callback::ltfb::sendrecv_weights
(C++ enumerator), 52
lbann::callback::ltfb::setup
(C++ function), 53
lbann::callback::ltfb::string_to_comm_algo
(C++ function), 53
lbann::callback::minibatch_schedule
(C++ class), 62
lbann::callback::minibatch_schedule::minibatch_step
(C++ member), 63
lbann::callback::minibatch_schedule::minibatch_step
(C++ member), 63
lbann::callback::minibatch_schedule::minibatch_step
(C++ member), 63
lbann::callback::minibatch_schedule::setup
(C++ function), 63
lbann::callback::minibatch_schedule::minibatch_step
(C++ member), 63
lbann::callback::mixup (C++ class), 53
lbann::callback::mixup::mixup
(C++ function), 54
lbann::callback::mixup::setup
(54)
(C++ function), 54
lbann::callback::learning_rate
(C++ class), 49
lbann::callback::learning_rate::learniing_rate
(C++ function), 49
lbann::callback::learning_rate::on_backward_prop_end
(C++ function), 49
lbann::callback::learning_rate::on_epoch_end
(C++ function), 49
lbann::callback::learning_rate::on_train_begin
(C++ function), 49
lbann::callback::learning_rate::setup
(C++ function), 49
lbann::callback::learning_rate::mixup
(C++ function), 49
lbann::callback::learning_rate::name
(C++ function), 49
lbann::callback::learning_rate::on_forward_prop_end
(C++ function), 49
lbann::callback::monitor_io (C++ class), 54
lbann::callback::monitor_io::monitor_io (C++ function), 54
lbann::callback::monitor_io::on_epoch_end (C++ function), 54
lbann::callback::monitor_io::on_test_end (C++ function), 54
lbann::callback::optimizerwise_adaptive_learning_rate (C++ class), 51
lbann::callback::optimizerwise_adaptive_learning_rate::name (C++ function), 51
lbann::callback::perturb_adam (C++ class), 54
lbann::callback::perturb_adam::name (C++ function), 55
lbann::callback::perturb_adam::on_batch_begin (C++ function), 54
lbann::callback::perturb_adam::perturb_adam (C++ function), 54
lbann::callback::perturb_adam::setup (C++ function), 55
lbann::callback::perturb_dropout (C++ class), 55
lbann::callback::perturb_dropout::name (C++ function), 55
lbann::callback::perturb_dropout::perturb_dropout (C++ function), 55
lbann::callback::perturb_dropout::setup (C++ function), 55
lbann::callback::poly_learning_rate (C++ class), 51
lbann::callback::poly_learning_rate::name (C++ function), 51
lbann::callback::poly_learning_rate::setup (C++ function), 51
lbann::callback::print_model_description (C++ class), 55
lbann::callback::print_model_description::name (C++ function), 55
lbann::callback::print_model_description::on_setup_end (C++ function), 56
lbann::callback::print_statistics (C++ class), 56
lbann::callback::print_statistics::name (C++ function), 56
lbann::callback::print_statistics::setup (C++ function), 56
lbann::callback::profiler (C++ class), 56
lbann::callback::profiler::name (C++ function), 58
lbann::callback::profiler::on_backward_prop_begin (C++ function), 57
lbann::callback::profiler::on_backward_prop_end (C++ function), 57
lbann::callback::profiler::on_batch_begin (C++ function), 57
lbann::callback::profiler::on_batch_end (C++ function), 57
lbann::callback::profiler::on_evaluate_forward_prop_begin (C++ function), 57
lbann::callback::profiler::on_evaluate_forward_prop_end (C++ function), 57
lbann::callback::profiler::on_forward_prop_begin (C++ function), 57
lbann::callback::profiler::on_forward_prop_end (C++ function), 57
lbann::callback::profiler::on_optimize_begin (C++ function), 57
lbann::callback::profiler::on_optimize_end (C++ function), 57
lbann::callback::profiler::on_test_begin (C++ function), 56
lbann::callback::profiler::on_test_end (C++ function), 57
lbann::callback::profiler::on_validation_begin (C++ function), 56
lbann::callback::profiler::on_validation_end (C++ function), 56
lbann::callback::replace_weights (C++ class), 58
lbann::callback::replace_weights::name (C++ function), 58
lbann::callback::replace_weights::on_batch_end (C++ function), 58
lbann::callback::save_images (C++ class), 58
lbann::callback::save_images::name (C++ function), 58
lbann::callback_base::name (C++ function), 39
lbann::callback_base::on_backward_prop_begin (C++ function), 38
lbann::callback_base::on_backward_prop_end (C++ function), 38
lbann::callback_base::on_batch_begin (C++ function), 37
lbann::callback_base::on_batch_end (C++ function), 38
lbann::callback_base::on_batch_evaluate_begin (C++ function), 38
lbann::callback_base::on_batch_evaluate_end (C++ function), 38
lbann::callback_base::on_epoch_begin (C++ function), 37
lbann::callback_base::on_epoch_end (C++ function), 37
lbann::callback_base::on_evaluate_forward_prop_begin (C++ function), 39
lbann::callback_base::on_evaluate_forward_prop_end (C++ function), 39
lbann::callback_base::on_forward_prop_begin (C++ function), 38
lbann::callback_base::on_forward_prop_end (C++ function), 38
lbann::callback_base::on_optimize_begin (C++ function), 38
lbann::callback_base::on_optimize_end (C++ function), 38
lbann::callback_base::on_phase_end (C++ function), 37
lbann::callback_base::on_setup_end (C++ function), 37
lbann::callback_base::on_test_begin (C++ function), 38
lbann::callback_base::on_test_end (C++ function), 38
lbann::callback_base::on_train_begin (C++ function), 37
lbann::callback_base::on_train_end (C++ function), 37
lbann::callback_base::on_validation_begin (C++ function), 38
lbann::callback_base::on_validation_end (C++ function), 38
lbann::callback_base::setup (C++ function), 37
lbann::categorical_accuracy_layer (C++ class), 103
lbann::categorical_accuracy_layer::get_data_layout (C++ function), 103
lbann::categorical_accuracy_layer::get_device_allocation (C++ function), 103
lbann::categorical_accuracy_layer::get_type (C++ function), 103
lbann::categorical_random_layer (C++ class), 128
lbann::categorical_random_layer::copy (C++ function), 128
lbann::categorical_random_layer::setup_dims (C++ function), 128
lbann::categorical_random_layer::copy (C++ function), 128
lbann::channelwise_mean_layer (C++ class), 103
lbann::channelwise_mean_layer::copy (C++ function), 103
lbann::channelwise_mean_layer::get_data_layout (C++ function), 104
lbann::channelwise_mean_layer::get_device_allocation (C++ function), 104
lbann::channelwise_mean_layer::get_type (C++ function), 104
lbann::channelwise_scale_bias_layer (C++ class), 104
lbann::channelwise_scale_bias_layer::copy (C++ function), 104
lbann::channelwise_scale_bias_layer::get_data_layout (C++ function), 104
lbann::channelwise_scale_bias_layer::get_device_allocation (C++ function), 104
lbann::channelwise_scale_bias_layer::get_type (C++ function), 104
lbann::channelwise_scale_bias_layer::setup_data (C++ function), 104
lbann::channelwise_scale_bias_layer::setup_matrices (C++ function), 104
lbann::cifar10_reader (C++ class), 80
lbann::cifar10_reader::get_type (C++ function), 80
lbann::cifar10_reader::load (C++ function), 80
lbann::cifar10_reader::set_input_params (C++ function), 80
lbann::clamp_layer (C++ class), 105
lbann::clamp_layer::copy (C++ function), 105
lbann::clamp_layer::get_data_layout (C++ function), 105
lbann:: clamp_layer::get_description (C++ function), 105
lbann::categorical_accuracy_layer::fp_compute (C++ function), 103
lbann::csv_reader::set_has_header (C++ function), 72
lbann::csv_reader::set_label_col (C++ function), 71
lbann::csv_reader::set_label_transform (C++ function), 72
lbann::csv_reader::set_response_col (C++ function), 71
lbann::csv_reader::set_response_transform (C++ function), 72
lbann::csv_reader::set_separator (C++ function), 72
lbann::csv_reader::set_skip_cols (C++ function), 72
lbann::csv_reader::set_skip_rows (C++ function), 72
lbann::data_buffer (C++ class), 95
lbann::data_buffer::m_indices_fetched_per_mb (C++ member), 95
lbann::data_buffer::m_input_buffers (C++ member), 95
lbann::data_buffer::m_num_samples_fetched (C++ member), 95
lbann::data_reader_jag (C++ class), 73
lbann::data_reader_jag::get_data_dims (C++ function), 74
lbann::data_reader_jag::get_dependent_variable_type (C++ function), 73
lbann::data_reader_jag::get_description (C++ function), 74
lbann::data_reader_jag::get_image_ptr (C++ function), 74
lbann::data_reader_jag::get_independent_variable_type (C++ function), 73
lbann::data_reader_jag::get_input (C++ function), 74
lbann::data_reader_jag::get_input_ptr (C++ function), 74
lbann::data_reader_jag::get_linearized_data_size (C++ function), 74
lbann::data_reader_jag::get_linearized_image_size (C++ function), 74
lbann::data_reader_jag::get_linearized_input_size (C++ function), 74
lbann::data_reader_jag::get_linearized_response_size (C++ function), 74
lbann::data_reader_jag::get_linearized_scalar_size (C++ function), 74
lbann::data_reader_jag::get_num_samples (C++ function), 74
lbann::data_reader_jag::get_scalar (C++ function), 74
lbann::data_reader_jag::get_scalar_ptr (C++ function), 74
lbann::data_reader_jag::get_type (C++ function), 73
lbann::data_reader_jag::JAG_Image (C++ enumerator), 73
lbann::data_reader_jag::JAG_Input (C++ enumerator), 73
lbann::data_reader_jag::JAG_Scalar (C++ enumerator), 73
lbann::data_reader_jag_conduit (C++ class), 74
lbann::data_reader_jag_conduit::add_input_filter (C++ function), 76
lbann::data_reader_jag_conduit::add_input_prefix_filter (C++ function), 76
lbann::data_reader_jag_conduit::add_scalar_filter (C++ function), 76
lbann::data_reader_jag_conduit::add_scalar_prefix_filter (C++ function), 76
lbann::data_reader_jag_conduit::ch_t (C++ type), 75
lbann::data_reader_jag_conduit::conduit_ch_t (C++ type), 75
lbann::data_reader_jag_conduit::fetch_data (C++ function), 77
lbann::data_reader_jag_conduit::fetch_labels (C++ function), 77
lbann::data_reader_jag_conduit::fetch_responses (C++ function), 77
lbann::data_reader_jag_conduit::get_data_dims (C++ function), 77
lbann::data_reader_jag_conduit::get_dependent_variable_type (C++ function), 75
lbann::data_reader_jag_conduit::get_description (C++ function), 78
lbann::data_reader_jag_conduit::get_image_choices (C++ function), 76
lbann::data_reader_jag_conduit::get_independent_variable_type (C++ function), 75
lbann::data_reader_jag_conduit::get_num_samples (C++ function), 78
lbann::data_reader_jag_conduit::get_image_choices (C++ function), 76
lbann::data_reader_jag_conduit::get_independent_variable_type (C++ function), 75
Index 207
lbann::data_reader_jag_conduit::variable_t
(C++ enum), 74
lbann::data_reader_merge_features (C++ class), 64
lbann::data_reader_merge_features::get_data_dims
(C++ function), 64
lbann::data_reader_merge_features::get_linearized_data_dims
(C++ function), 64
lbann::data_reader_merge_features::get_num_labels
(C++ function), 64
lbann::data_reader_merge_features::get_type
(C++ function), 64
lbann::data_reader_merge_features::load
(C++ function), 64
lbann::data_reader_merge_samples (C++ class), 64
lbann::data_reader_merge_samples::get_data_dims
(C++ function), 65
lbann::data_reader_merge_samples::get_linearized_data_dims
(C++ function), 65
lbann::data_reader_merge_samples::get_num_labels
(C++ function), 65
lbann::data_reader_merge_samples::get_num_responses
(C++ function), 65
lbann::data_reader_merge_samples::get_num_samples_psum
(C++ function), 65
lbann::data_reader_merge_samples::get_type
(C++ function), 64
lbann::data_reader_merge_samples::load
(C++ function), 64
lbann::data_reader_multi_images (C++ class), 81
lbann::data_reader_multi_images::get_data_dims
(C++ function), 81
lbann::data_reader_multi_images::get_image_list
(C++ function), 81
lbann::data_reader_multi_images::get_image_list_of_current_mb
(C++ function), 81
lbann::data_reader_multi_images::get_type
(C++ function), 81
lbann::data_reader_multi_images::load
(C++ function), 81
lbann::data_reader_multihead_siamese (C++ class), 81
lbann::data_reader_multihead_siamese::get_image_list
(C++ function), 82
lbann::data_reader_multihead_siamese::get_image_list_of_current_mb
(C++ function), 82
lbann::data_reader_multihead_siamese::get_type
(C++ function), 81
lbann::data_reader_multihead_siamese::load
(C++ function), 82
lbann::data_reader_multihead_siamese::set_input_params
(C++ function), 81
lbann::data_reader_nci (C++ class), 72
lbann::data_reader_synthetic (C++ class), 78
lbann::data_reader_synthetic::get_data_dims
(C++ function), 79
lbann::data_reader_synthetic::get_linearized_data_dims
(C++ function), 79
lbann::data_reader_synthetic::get_linearized_label_size
(C++ function), 79
lbann::data_reader_synthetic::get_num_labels
(C++ function), 79
lbann::data_reader_synthetic::get_num_responses
(C++ function), 79
lbann::data_reader_synthetic::get_type
(C++ function), 78
lbann::data_reader_synthetic::load
(C++ function), 78
lbann::data_reader_triplet (C++ class), 82
lbann::data_reader_triplet::get_image_list
(C++ function), 82
lbann::data_reader_triplet::get_image_list_of_current_mb
(C++ function), 82
lbann::data_reader_triplet::set_input_params
(C++ function), 82
lbann::data_store_conduit::~data_store_conduit
(C++ function), 90
lbann::data_store_conduit::build_preloaded_owner_map
(C++ function), 91
lbann::data_store_conduit::compact_nodes
(C++ function), 91
lbann::data_store_conduit::copy_members
(C++ function), 91
lbann::data_store_conduit::data_store_conduit
(C++ function), 91
(C++ function), 90
lbann::data_store_conduit::flush_debug_file (C++ function), 91
lbann::data_store_conduit::flush_profile (C++ function), 90
lbann::data_store_conduit::get_conduit_node (C++ function), 90
lbann::data_store_conduit::get_data_size (C++ function), 91
lbann::data_store_conduit::get_empty_node (C++ function), 91
lbann::data_store_conduit::get_index_owner (C++ function), 91
lbann::data_store_conduit::get_num_global_indices (C++ function), 90
lbann::data_store_conduit::is_local_cache (C++ function), 91
lbann::data_store_conduit::load_checkpoint (C++ function), 91
lbann::data_store_conduit::m_debug (C++ member), 91
lbann::data_store_conduit::operator= (C++ function), 90
lbann::data_store_conduit::preload_local_cache (C++ function), 91
lbann::data_store_conduit::set_conduit_node (C++ function), 91
lbann::data_store_conduit::set_data_reader_ptr (C++ function), 90
lbann::data_store_conduit::set_shuffled_indices (C++ function), 90
lbann::data_store_conduit::write_checkpoint (C++ function), 91
lbann::dataset (C++ class), 178
lbann::deconvolution_layer (C++ class), 101
lbann::deconvolution_layer::copy (C++ function), 102
lbann::deconvolution_layer::get_data_layout (C++ function), 102
lbann::deconvolution_layer::get_device_allocation (C++ function), 102
lbann::deconvolution_layer::get_type (C++ function), 102
lbann::deconvolution_layer::setup_dims (C++ function), 102
lbann::default_key_error_policy (C++ class), 178
lbann::description (C++ class), 179
lbann::description::add (C++ function), 179
lbann::description::description (C++ function), 179
lbann::description::set_title (C++ function), 179
lbann::directed_acyclic_graph_model (C++ class), 149
lbann::directed_acyclic_graph_model::get_type (C++ function), 149
lbann::discrete_random_layer (C++ class), 131
lbann::discrete_random_layer::copy (C++ function), 131
lbann::discrete_random_layer::get_data_layout (C++ function), 131
lbann::discrete_random_layer::get_device_allocation (C++ function), 131
lbann::dropout (C++ class), 121
lbann::dropout::copy (C++ function), 122
lbann::dropout::get_data_layout (C++ function), 122
lbann::dropout::get_device_allocation (C++ function), 122
lbann::dropout::get_keep_prob (C++ function), 122
lbann::dropout::get_type (C++ function), 122
lbann::dummy_layer (C++ class), 131
lbann::dummy_layer::copy (C++ function), 131
lbann::dummy_layer::get_data_layout (C++ function), 131
lbann::dummy_layer::get_device_allocation (C++ function), 131
lbann::elu_layer (C++ class), 107
lbann::elu_layer::copy (C++ function), 107
lbann::elu_layer::get_data_layout (C++ function), 107
lbann::elu_layer::get_device_allocation (C++ function), 107
lbann::embedding_layer (C++ class), 107
lbann::embedding_layer::copy (C++ function), 108
lbann::embedding_layer::get_data_layout (C++ function), 108
lbann::embedding_layer::get_device_allocation (C++ function), 108
lbann::generic_data_reader::has_list_per_trainer
(C++ function), 70

lbann::generic_data_reader::is_master
(C++ function), 70

lbann::generic_data_reader::is_partitioned
(C++ function), 70

lbann::generic_data_reader::is_shuffled
(C++ function), 66

lbann::generic_data_reader::load
(C++ function), 67

lbann::generic_data_reader::load_from_checkpoint_distributed
(C++ function), 71

lbann::generic_data_reader::load_from_checkpoint_shared
(C++ function), 71

lbann::generic_data_reader::position_is_overrun
(C++ function), 68

lbann::generic_data_reader::position_valid
(C++ function), 68

lbann::generic_data_reader::post_update
(C++ function), 71

lbann::generic_data_reader::resize_shuffled_indices
(C++ function), 70

lbann::generic_data_reader::save_to_checkpoint_shared
(C++ function), 71

lbann::generic_data_reader::select_subset_of_data
(C++ function), 70

lbann::generic_data_reader::select_subset_of_data_partitioned
(C++ function), 70

lbann::generic_data_reader::serialize
(C++ function), 65

lbann::generic_data_reader::set_absolute_sample_count
(C++ function), 66

lbann::generic_data_reader::set_base_offset
(C++ function), 68

lbann::generic_data_reader::set_comm
(C++ function), 65

lbann::generic_data_reader::set_data_filename
(C++ function), 66

lbann::generic_data_reader::set_data_index_list
(C++ function), 66

lbann::generic_data_reader::set_data_store
(C++ function), 71

lbann::generic_data_reader::set_file_dir
(C++ function), 65

lbann::generic_data_reader::set_first_n
(C++ function), 66

lbann::generic_data_reader::set_global_last_mini_batch_size
(C++ function), 69

lbann::generic_data_reader::set_global_mini_batch_size
(C++ function), 68

lbann::generic_data_reader::set_initial_position
(C++ function), 69

lbann::generic_data_reader::set_iteration_stride
(C++ function), 68

lbann::generic_data_reader::set_label_filename
(C++ function), 66

lbann::generic_data_reader::set_last_mini_batch_size
(C++ function), 69

lbann::generic_data_reader::set_local_file_dir
(C++ function), 65

lbann::generic_data_reader::set_model_offset
(C++ function), 69

lbann::generic_data_reader::set_num_iterations_per_epoch
(C++ function), 70

lbann::generic_data_reader::set_num_parallel_readers
(C++ function), 69

lbann::generic_data_reader::set_partitioned
(C++ function), 70

lbann::generic_data_reader::set_rank
(C++ function), 70

lbann::generic_data_reader::set_reset_mini_batch_index
(C++ function), 69

lbann::generic_data_reader::set_sample_stride
(C++ function), 68

lbann::generic_data_reader::set_shuffle
(C++ function), 66

lbann::generic_data_reader::set_use_percent
(C++ function), 66

lbann::generic_data_reader::set_validation_percent
(C++ function), 66

lbann::generic_data_reader::set_world_master_mini_batch_adjustment
(C++ function), 69

lbann::generic_data_reader::setup
(C++ function), 67

lbann::generic_data_reader::setup_data_store
(C++ function), 71

lbann::generic_data_reader::update
(C++ function), 67

lbann::generic_data_reader::use_unused_index_set
(C++ function), 70

lbann::generic_input_layer
(C++ class), 111

lbann::generic_input_layer::calculate_num_iteration
(C++ function), 68
<table>
<thead>
<tr>
<th>Function/Method Name</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbann::in_top_k_layer::get_description</td>
<td>114</td>
</tr>
<tr>
<td>lbann::in_top_k_layer::get_device_allocation</td>
<td>114</td>
</tr>
<tr>
<td>lbann::in_top_k_layer::get_type</td>
<td>114</td>
</tr>
<tr>
<td>lbann::input_layer::copy</td>
<td>113</td>
</tr>
<tr>
<td>lbann::input_layer::get_data_layout</td>
<td>113</td>
</tr>
<tr>
<td>lbann::input_layer::get_device_allocation</td>
<td>113</td>
</tr>
<tr>
<td>lbann::input_layer::get_type</td>
<td>113</td>
</tr>
<tr>
<td>lbann::io_layer::get_data_dims</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::get_dataset</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::get_linearized_data_size</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::get_linearized_label_size</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::get_sample_indices_per_mb</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::select_data_reader</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::select_dataset</td>
<td>112</td>
</tr>
<tr>
<td>lbann::io_layer::select_first_valid_dataset</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::add_child_layer</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::add_parent_layer</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::back_prop</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::check_setup</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::clear_child_layers</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::clear_parent_layers</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::copy</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::forward_prop</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_activations</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_child_layers</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_comm</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_device_allocation</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_data_layout</td>
<td>112</td>
</tr>
<tr>
<td>lbann::layer::get_type</td>
<td>112</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::bp_compute</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::copy</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::fp_compute</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::get_data_layout</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::get_device_allocation</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l1_norm_layer::get_type</td>
<td>113</td>
</tr>
<tr>
<td>lbann::l2_norm2_layer::copy</td>
<td>114</td>
</tr>
<tr>
<td>lbann::l2_norm2_layer::fp_compute</td>
<td>114</td>
</tr>
<tr>
<td>lbann::l2_norm2_layer::get_data_layout</td>
<td>114</td>
</tr>
<tr>
<td>lbann::l2_norm2_layer::get_device_allocation</td>
<td>114</td>
</tr>
<tr>
<td>lbann::l2_norm2_layer::get_type</td>
<td>114</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::compute_weight_regularization</td>
<td>151</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::copy</td>
<td>150</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::differentiate</td>
<td>151</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::finish_evaluation</td>
<td>151</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::name</td>
<td>150</td>
</tr>
<tr>
<td>lbann::l2_weight_regularization::setup</td>
<td>150</td>
</tr>
<tr>
<td>Index 215</td>
<td></td>
</tr>
</tbody>
</table>
lbann::Layer::get_data_layout_string (C++ function), 97
lbann::Layer::get_description (C++ function), 96
lbann::Layer::get_device_allocation (C++ function), 97
lbann::Layer::get_device_allocation_string (C++ function), 97
lbann::Layer::get_device_allocation_string_short (C++ function), 97
lbann::Layer::get_error_signals (C++ function), 98, 99
lbann::Layer::get_expected_num_child_layers (C++ function), 97
lbann::Layer::get_expected_num_parent_layers (C++ function), 98
lbann::Layer::get_hint_layer (C++ function), 99
lbann::Layer::get_input_dims (C++ function), 98
lbann::Layer::get_input_size (C++ function), 98
lbann::Layer::get_layer_names (C++ function), 99
lbann::Layer::get_layer_pointers (C++ function), 98
lbann::Layer::get_local_activations (C++ function), 99
lbann::Layer::get_local_error_signals (C++ function), 99
lbann::Layer::get_local_prev_activations (C++ function), 99
lbann::Layer::get_local_prev_error_signals (C++ function), 99
lbann::Layer::get_model (C++ function), 97
lbann::Layer::get_name (C++ function), 96
lbann::Layer::get_num_children (C++ function), 98
lbann::Layer::get_num_parents (C++ function), 98
lbann::Layer::get_output_dims (C++ function), 98
lbann::Layer::get_output_size (C++ function), 98
lbann::Layer::get_parent_layers (C++ function), 97
lbann::Layer::get_prev_activations (C++ function), 99
lbann::Layer::get_prev_error_signals (C++ function), 99
lbann::Layer::get_type (C++ function), 96
lbann::Layer::get_weights (C++ function), 98
lbann::Layer::replace_weights (C++ function), 98
lbann::Layer::reset_counters (C++ function), 97
lbann::Layer::set_hint_layer (C++ function), 99
lbann::Layer::set_layer_pointers (C++ function), 98
lbann::Layer::set_model (C++ function), 97
lbann::Layer::set_name (C++ function), 96
lbann::Layer::set_output_dims (C++ function), 98
lbann::Layer::set_weights (C++ function), 98
lbann::Layer::set_model (C++ function), 97
lbann::Layer::set_up (C++ function), 97
lbann::Layer::update (C++ function), 96
lbann::Layer::using_gpus (C++ function), 97
lbann::Layer::write_proto (C++ function), 97
lbann::layer_metric (C++ class), 144
lbann::layer_metric::copy (C++ function), 144
lbann::layer_metric::get_layer (C++ function), 145
lbann::layer_metric::get_layer_pointers (C++ function), 145
lbann::layer_term (C++ class), 151
lbann::layer_term::compute_weight_regularization (C++ function), 151
lbann::layer_term::copy (C++ function), 151
lbann::layer_term::differentiate (C++ function), 151
lbann::layer_term::finish_evaluation (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::layer_term::get_layer (C++ function), 151
lbann::lbann_comm (C++ class), 29
lbann::lbann_comm::all_gather (C++ function), 31, 32
lbann::lbann_comm::allreduce (C++ function), 34
lbann::lbann_comm::am_trainer_master (C++ function), 30
lbann::lbann_comm::am_world_master (C++ function), 31
lbann::lbann_comm::all_gather (C++ function), 31, 32
function), 30
lbann::lbann_comm::barrier (C++ function), 35
lbann::lbann_comm::broadcast (C++ function), 30, 31
lbann::lbann_comm::count_bytes_broadcast (C++ function), 31
lbann::lbann_comm::gather (C++ function), 33
lbann::lbann_comm::get_bytes_received (C++ function), 36
lbann::lbann_comm::get_bytes_sent (C++ function), 36
lbann::lbann_comm::get_count (C++ function), 35
lbann::lbann_comm::get_default_threads_per_proc (C++ function), 30
lbann::lbann_comm::get_intertrainer_comm (C++ function), 36
lbann::lbann_comm::get_intertrainer_master (C++ function), 30
lbann::lbann_comm::get_node_comm (C++ function), 36
lbann::lbann_comm::get_num_global_barriers (C++ function), 36
lbann::lbann_comm::get_num_intertrainer_barriers (C++ function), 35
lbann::lbann_comm::get_num_trainer_barriers (C++ function), 35
lbann::lbann_comm::get_num_trainers (C++ function), 30
lbann::lbann_comm::get_packed_group_comm (C++ function), 36
lbann::lbann_comm::get_procs_in_world (C++ function), 36
lbann::lbann_comm::get_procs_per_node (C++ function), 30
lbann::lbann_comm::get_rank_in_node (C++ function), 30
lbann::lbann_comm::get_rank_in_trainer (C++ function), 29
lbann::lbann_comm::get_rank_in_world (C++ function), 30
lbann::lbann_comm::get_trainer_comm (C++ function), 36
lbann::lbann_comm::get_trainer_grid (C++ function), 30
lbann::lbann_comm::get_trainer_master (C++ function), 30
lbann::lbann_comm::get_trainer_rank (C++ function), 29
lbann::lbann_comm::get_world_comm (C++ function), 36
lbann::lbann_comm::get_world_master (C++ function), 30
lbann::lbann_comm::get_world_rank (C++ function), 30
lbann::lbann_comm::global_barrier (C++ function), 35
lbann::lbann_comm::intertrainer_allreduce (C++ function), 34
lbann::lbann_comm::intertrainer_barrier (C++ function), 35
lbann::lbann_comm::intertrainer_broadcast (C++ function), 31
lbann::lbann_comm::intertrainer_broadcast_matrix (C++ function), 30
lbann::lbann_comm::intertrainer_gather (C++ function), 32
lbann::lbann_comm::intertrainer_reduce (C++ function), 33
lbann::lbann_comm::is_rank_node_local (C++ function), 36
lbann::lbann_comm::is_sendable (C++ function), 36
lbann::lbann_comm::is_world_rank_on_node (C++ function), 30
lbann::lbann_comm::intertrainer_barrier (C++ function), 35
lbann::lbann_comm::intertrainer_master (C++ function), 30
lbann::lbann_comm::intertrainer_reduce (C++ function), 33
lbann::lbann_comm::split_trainers (C++ function), 29
lbann::lbann_comm::test (C++ function), 35
lbann::lbann_comm::trainer_all_gather (C++ function), 32
lbann::lbann_comm::trainer_allreduce (C++ function), 34
lbann::lbann_comm::nb_allreduce (C++ function), 34
lbann::lbann_comm::nb_recv (C++ function), 35
lbann::lbann_comm::nb_send (C++ function), 35
lbann::lbann_comm::operator= (C++ function), 29
lbann::lbann_comm::recv (C++ function), 35
lbann::lbann_comm::reduce (C++ function), 33, 34
lbann::lbann_comm::reset_threads (C++ function), 30
lbann::lbann_comm::resize (C++ function), 31
lbann::lbann_comm::scatter (C++ function), 33
lbann::lbann_comm::send (C++ function), 35
lbann::lbann_comm::sendrecv (C++ function), 35
lbann::lbann_comm::sendrecv (C++ function), 35
lbann::lbann_comm::sendrecv (C++ function), 35
lbann::lbann_comm::split_trainers (C++ function), 29
lbann::lbann_comm::test (C++ function), 35
lbann::lbann_comm::trainer_all_gather (C++ function), 32
lbann::lbann_comm::trainer_allreduce (C++ function), 34
lbann::lbann_comm::trainer_barrier (C++ function), 35
lbann::lbann_comm::trainer_broadcast (C++ function), 31
lbann::lbann_comm::trainer_gather (C++ function), 32
lbann::lbann_comm::trainer_gatherv (C++ function), 32
lbann::lbann_comm::trainer_reduce (C++ function), 33
lbann::lbann_comm::wait (C++ function), 35
lbann::lbann_comm::wait_all (C++ function), 35
lbann::lbann_comm::world_broadcast (C++ function), 31
lbann::lbann_summary (C++ class), 180
lbann::leaky_relu_layer (C++ class), 115
lbann::leaky_relu_layer::copy (C++ function), 115
lbann::leaky_relu_layer::get_data_layout (C++ function), 115
lbann::leaky_relu_layer::get_description (C++ function), 115
lbann::leaky_relu_layer::get_device_allocation (C++ function), 115
lbann::leaky_relu_layer::get_type (C++ function), 115
lbann::learning_layer (C++ class), 115
lbann::lecun_initializer (C++ class), 186
lbann::lecun_initializer::copy (C++ function), 186
lbann::lecun_initializer::get_type (C++ function), 186
lbann::local_response_normalization_layer (C++ class), 122
lbann::local_response_normalization_layer::copy (C++ function), 123
lbann::local_response_normalization_layer::get_data_layout (C++ function), 123
lbann::local_response_normalization_layer::get_device_allocation (C++ function), 123
lbann::local_response_normalization_layer::get_type (C++ function), 123
lbann::log_sigmoid_layer (C++ class), 141
lbann::log_softmax_layer (C++ class), 116
lbann::log_softmax_layer::bp_compute (C++ function), 117
lbann::log_softmax_layer::copy (C++ function), 116
lbann::log_softmax_layer::fp_compute (C++ function), 117
lbann::log_softmax_layer::fp_setup_outputs (C++ function), 117
lbann::log_softmax_layer::get_data_layout (C++ function), 117
lbann::log_softmax_layer::get_device_allocation (C++ function), 116
lbann::log_softmax_layer::get_type (C++ function), 117
lbann::log_softmax_layer::setup_dims (C++ function), 117
lbann::log_softmax_layer::setup_matrices (C++ function), 117
lbann::mean_absolute_error_layer (C++ class), 117
lbann::mean_absolute_error_layer::bp_compute (C++ function), 118
lbann::mean_absolute_error_layer::copy (C++ function), 117
lbann::mean_absolute_error_layer::fp_compute (C++ function), 118
lbann::mean_squared_error_layer (C++ class), 118
lbann::mean_squared_error_layer::bp_compute (C++ function), 119
lbann::mean_squared_error_layer::copy (C++ function), 118
lbann::mean_squared_error_layer::fp_compute (C++ function), 119
lbann::mean_squared_error_layer::get_data_layout (C++ function), 118
lbann::mean_squared_error_layer::get_device_allocation (C++ function), 118
lbann::mean_squared_error_layer::setup_data (C++ function), 118
lbann::mean_squared_error_layer::setup_dims (C++ function), 118
lbann::mesh_reader (C++ class), 83
lbann::mesh_reader::get_data_dims (C++ function), 83
lbann::mesh_reader::get_linearized_data_size (C++ function), 83
lbann::mesh_reader::get_linearized_response_size (C++ function), 83
lbann::log_sigmoid_layer::bp_compute (C++ function), 117
lbann::log_softmax_layer::bp_compute (C++ function), 116
lbann::mesh_reader::get_type (C++ function), 83
lbann::mesh_reader::load (C++ function), 83
lbann::mesh_reader::set_data_shape (C++ function), 83
lbann::mesh_reader::set_index_length (C++ function), 83
lbann::mesh_reader::set_random_flips (C++ function), 83
lbann::mesh_reader::set_suffix (C++ function), 83
lbann::metric (C++ class), 143
lbann::metric::~metric (C++ function), 143
lbann::metric::copy (C++ function), 143
lbann::metric::evaluate (C++ function), 144
lbann::metric::get_evaluate_time (C++ function), 144
lbann::metric::get_layer_pointers (C++ function), 144
lbann::metric::get_mean_value (C++ function), 144
lbann::metric::get_statistics_num_samples (C++ function), 144
lbann::metric::get_unit (C++ function), 143
lbann::metric::load_from_checkpoint_shared (C++ function), 144
lbann::metric::metric (C++ function), 143
lbann::metric::operator= (C++ function), 143
lbann::metric::reset_counters (C++ function), 144
lbann::metric::reset_statistics (C++ function), 144
lbann::metric::save_to_checkpoint_shared (C++ function), 144
lbann::metric::serialize (C++ function), 143
lbann::metric::setup (C++ function), 144
lbann::model (C++ class), 145
lbann::model::add_callback (C++ function), 146
lbann::model::add_layer (C++ function), 146
lbann::model::add_metric (C++ function), 146
lbann::model::add_weights (C++ function), 146
lbann::model::allow_background_io_activity (C++ function), 147
lbann::model::background_io_activity_allowed (C++ function), 147
lbann::model::backward_prop (C++ function), 148
lbann::model::clear_gradients (C++ function), 148
lbann::model::collect_background_data_fetch (C++ function), 147
lbann::model::copy_trained_weights_from (C++ function), 146
lbann::model::create_optimizer (C++ function), 147
lbann::model::do_layer_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_backward_prop_end_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_end_cbs (C++ function), 148
lbann::model::do_model_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_model_backward_prop_end_cbs (C++ function), 148
lbann::model::do_model_optimize_begin_cbs (C++ function), 148
lbann::model::do_model_optimize_end_cbs (C++ function), 148
lbann::model::do_setup_end_cbs (C++ function), 148
lbann::mini_batch_size_layer (C++ class), 119
lbann::mini_batch_size_layer::copy (C++ function), 119
lbann::mini_batch_size_layer::get_data_layout (C++ function), 120
lbann::mini_batch_size_layer::get_device_allocation (C++ function), 120
lbann::mini_batch_size_layer::get_type (C++ function), 120
lbann::mnist_reader (C++ class), 82
lbann::mnist_reader::get_type (C++ function), 82
lbann::mnist_reader::load (C++ function), 82
lbann::mnist_reader::set_input_params (C++ function), 82
lbann::model::add_callback (C++ function), 146
lbann::model::add_layer (C++ function), 146
lbann::model::add_metric (C++ function), 146
lbann::model::add_weights (C++ function), 146
lbann::model::allow_background_io_activity (C++ function), 147
lbann::model::background_io_activity_allowed (C++ function), 147
lbann::model::backward_prop (C++ function), 148
lbann::model::clear_gradients (C++ function), 148
lbann::model::collect_background_data_fetch (C++ function), 147
lbann::model::copy_trained_weights_from (C++ function), 146
lbann::model::create_optimizer (C++ function), 147
lbann::model::do_layer_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_backward_prop_end_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_end_cbs (C++ function), 148
lbann::model::do_model_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_model_backward_prop_end_cbs (C++ function), 148
lbann::model::do_model_optimize_begin_cbs (C++ function), 148
lbann::model::do_model_optimize_end_cbs (C++ function), 148
lbann::model::do_setup_end_cbs (C++ function), 148
lbann::mini_batch_index_layer (C++ class), 119
lbann::mini_batch_index_layer::copy (C++ function), 119
lbann::mini_batch_index_layer::get_data_layout (C++ function), 120
lbann::mini_batch_index_layer::get_device_allocation (C++ function), 120
lbann::mini_batch_index_layer::get_type (C++ function), 120
lbann::mnist_reader (C++ class), 82
lbann::mnist_reader::get_type (C++ function), 82
lbann::mnist_reader::load (C++ function), 82
lbann::mnist_reader::set_input_params (C++ function), 82
lbann::model (C++ class), 145
lbann::model::add_callback (C++ function), 146
lbann::model::add_layer (C++ function), 146
lbann::model::add_metric (C++ function), 146
lbann::model::add_weights (C++ function), 146
lbann::model::allow_background_io_activity (C++ function), 147
lbann::model::background_io_activity_allowed (C++ function), 147
lbann::model::backward_prop (C++ function), 148
lbann::model::clear_gradients (C++ function), 148
lbann::model::collect_background_data_fetch (C++ function), 147
lbann::model::copy_trained_weights_from (C++ function), 146
lbann::model::create_optimizer (C++ function), 147
lbann::model::do_layer_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_backward_prop_end_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_begin_cbs (C++ function), 148
lbann::model::do_layer_forward_prop_end_cbs (C++ function), 148
lbann::model::do_model_backward_prop_begin_cbs (C++ function), 148
lbann::model::do_model_backward_prop_end_cbs (C++ function), 148
lbann::model::do_model_optimize_begin_cbs (C++ function), 148
lbann::model::do_model_optimize_end_cbs (C++ function), 148
lbann::model::do_setup_end_cbs (C++ function), 148
lbann::model::do_weight_optimize_begin_cbs (C++ function), 148
lbann::model::do_weight_optimize_end_cbs (C++ function), 149
lbann::model::evaluate_metrics (C++ function), 148
lbann::model::forward_prop (C++ function), 147
lbann::model::get_callbacks (C++ function), 146
lbann::model::get_comm (C++ function), 146
lbann::model::get_description (C++ function), 145
lbann::model::get_execution_context (C++ function), 146
lbann::model::get_layer (C++ function), 146
lbann::model::get_layers (C++ function), 146
lbann::model::get_max_mini_batch_size (C++ function), 147
lbann::model::get_metrics (C++ function), 146
lbann::model::get_name (C++ function), 145
lbann::model::get_num_layers (C++ function), 146
lbann::model::get_objective_function (C++ function), 146
lbann::model::get_type (C++ function), 145
lbann::model::has_valid_execution_context (C++ function), 146
lbann::model::is_execution_mode_valid (C++ function), 147
lbann::model::load_from_checkpoint_shared (C++ function), 147
lbann::model::reconcile_weight_values (C++ function), 148
lbann::model::reload_weights (C++ function), 147
lbann::model::reset_epoch_statistics (C++ function), 147
lbann::model::reset_mode (C++ function), 147
lbann::model::save_model (C++ function), 147
lbann::model::save_to_checkpoint_shared (C++ function), 147
lbann::model::save_weights (C++ function), 147
lbann::model::set_name (C++ function), 145
lbann::model::setup (C++ function), 147
lbann::model::summarize_matrices (C++ function), 147
lbann::model::summarize_stats (C++ function), 147
lbann::model::update_layers (C++ function), 148
lbann::model::update_weights (C++ function), 148
lbann::model::write_proto (C++ function), 147
lbann::normal_initializer (C++ class), 187
lbann::normal_initializer::copy (C++ function), 187
lbann::normal_initializer::fill (C++ function), 187
lbann::normal_initializer::get_description (C++ function), 187
lbann::normal_initializer::get_type (C++ function), 187
lbann::nullptr_key_error_policy (C++ class), 180
lbann::numpy_npz_conduit_reader (C++ class), 83
lbann::numpy_npz_conduit_reader::get_data_dims (C++ function), 84
lbann::numpy_npz_conduit_reader::get_linearized_data_size (C++ function), 84
lbann::numpy_npz_conduit_reader::get_linearized_label_size (C++ function), 84
lbann::numpy_npz_conduit_reader::get_linearized_response_size (C++ function), 84
lbann::numpy_npz_conduit_reader::get_num_labels (C++ function), 84
lbann::numpy_npz_conduit_reader::get_num_responses (C++ function), 84
lbann::numpy_npz_conduit_reader::get_type (C++ function), 83
lbann::numpy_npz_conduit_reader::load (C++ function), 84
lbann::numpy_npz_conduit_reader::set_has_labels (C++ function), 83
lbann::numpy_npz_conduit_reader::set_has_responses (C++ function), 84
lbann::numpy_npz_conduit_reader::set_scaling_factor (C++ function), 84
lbann::numpy_npz_reader (C++ class), 84
lbann::numpy_npz_reader::get_data_dims (C++ function), 85
lbann::numpy_npz_reader::get_linearized_data_size (C++ function), 85
lbann::numpy_npz_reader::get_linearized_label_size (C++ function), 85
lbann::numpy_npz_reader::get_linearized_response_size (C++ function), 85
lbann::numpy_npz_reader::get_num_labels (C++ function), 85
lbann::numpy_npz_reader::get_num_responses (C++ function), 85
lbann::numpy_npz_reader::get_type (C++ function), 85
lbann::numpy_npz_reader::load (C++ function), 84
lbann::numpy_npz_reader::set_has_labels (C++ function), 84
lbann::numpy_npz_reader::set_has_responses (C++ function), 84
lbann::numpy_npz_reader::set_scaling_factor_int16 (C++ function), 84
lbann::numpy_reader (C++ class), 85
lbann::numpy_reader::get_data_dims (C++ function), 86
lbann::numpy_reader::get_linearized_data_size (C++ function), 86
lbann::numpy_reader::get_linearized_label_size (C++ function), 86
lbann::numpy_reader::get_num_labels (C++ function), 85
lbann::numpy_reader::get_type (C++ function), 85
lbann::numpy_reader::load (C++ function), 85
lbann::numpy_reader::set_has_labels (C++ function), 85
lbann::numpy_reader::set_has_responses (C++ function), 85
lbann::objective_function (C++ class), 152
lbann::objective_function::~objective_function (C++ function), 152
lbann::objective_function::add_term (C++ function), 152
lbann::objective_function::compute_weight_regularization (C++ function), 152
lbann::objective_function::copy (C++ function), 152
lbann::objective_function::differentiate (C++ function), 152
lbann::objective_function::finish_evaluation (C++ function), 152
lbann::objective_function::get_layer_pointers (C++ function), 152
lbann::objective_function::get_mean_value (C++ function), 152
lbann::objective_function::get_statistics_num_samples (C++ function), 152
lbann::objective_function::get_terms (C++ function), 152
lbann::objective_function::get_weights_pointers (C++ function), 152
lbann::objective_function::objective_function (C++ function), 152
lbann::objective_function::operator= (C++ function), 152
lbann::objective_function::reset_counters (C++ function), 153
lbann::objective_function::reset_statistics (C++ function), 152
lbann::objective_function::set_layer_pointers (C++ function), 153
lbann::objective_function::set_weights_pointers (C++ function), 153
lbann::objective_function::setup (C++ function), 152
lbann::objective_function::start_evaluation (C++ function), 152
lbann::objective_function_term (C++ class), 149
lbann::objective_function_term::~objective_function_term (C++ function), 149
lbann::objective_function_term::compute_weight_regularization (C++ function), 150
lbann::objective_function_term::copy (C++ function), 149
lbann::objective_function_term::differentiate (C++ function), 150
lbann::objective_function_term::finish_evaluation (C++ function), 150
lbann::objective_function_term::get_layer_pointers (C++ function), 150
lbann::objective_function_term::get_weights_pointers (C++ function), 150
lbann::objective_function_term::name (C++ function), 149
lbann::objective_function_term::objective_function (C++ function), 152
lbann::objective_function_term::operator= (C++ function), 149
lbann::objective_function_term::reset (C++ function), 152
lbann::objective_function_term::set_layer_pointers (C++ function), 152
lbann::objective_function_term::set_weights_pointers (C++ function), 152
lbann::objective_function_term::setup (C++ function), 149
lbann::objective_function_term::start_evaluation (C++ function), 149
lbann::offline_patches_npz (C++ class), 88
lbann::offline_patches_npz::get_description (C++ function), 88
lbann::offline_patches_npz::get_label (C++ function), 89
lbann::offline_patches_npz::get_num_patches (C++ function), 88
lbann::offline_patches_npz::get_num_samples (C++ function), 88
lbann::offline_patches_npz::get_sample (C++ function), 88
lbann::offline_patches_npz::objective_function (C++ function), 88
lbann::objective_function::objective_function (C++ function), 88
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbann::offline_patches_npz::load</td>
<td>88</td>
</tr>
<tr>
<td>lbann::offline_patches_npz::set_num_patches</td>
<td>88</td>
</tr>
<tr>
<td>lbann::one_hot_layer</td>
<td>120</td>
</tr>
<tr>
<td>lbann::one_hot_layer::copy</td>
<td>120</td>
</tr>
<tr>
<td>lbann::one_hot_layer::get_data_layout</td>
<td>120</td>
</tr>
<tr>
<td>lbann::one_hot_layer::get_device_allocation</td>
<td>120</td>
</tr>
<tr>
<td>lbann::one_hot_layer::get_type</td>
<td>120</td>
</tr>
<tr>
<td>lbann::optimizer</td>
<td>153</td>
</tr>
<tr>
<td>lbann::optimizer::add_gradient_source</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::add_to_gradient</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::clear_gradient</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::copy</td>
<td>153</td>
</tr>
<tr>
<td>lbann::optimizer::get_comm</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::get_description</td>
<td>153</td>
</tr>
<tr>
<td>lbann::optimizer::get_gradient</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::get_gradient_buffer</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::get_learning_rate</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::get_num_gradient_sources</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::get_step_time</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::get_type</td>
<td>153</td>
</tr>
<tr>
<td>lbann::optimizer::get_weights</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::remove_gradient_source</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::reset_counters</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::set_learning_rate</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::set_weights</td>
<td>154</td>
</tr>
<tr>
<td>lbann::optimizer::setup</td>
<td>155</td>
</tr>
<tr>
<td>lbann::optimizer::step</td>
<td>155</td>
</tr>
<tr>
<td>lbann::options::init</td>
<td>181</td>
</tr>
<tr>
<td>lbann::options::print</td>
<td>181</td>
</tr>
<tr>
<td>lbann::options::set_option</td>
<td>181</td>
</tr>
<tr>
<td>lbann::partitioned_io_buffer</td>
<td>95</td>
</tr>
<tr>
<td>lbann::partitioned_io_buffer::get_type</td>
<td>95</td>
</tr>
<tr>
<td>lbann::partitioned_io_buffer::m_data_buffers</td>
<td>95</td>
</tr>
<tr>
<td>lbann::persist</td>
<td>94</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_data_dims</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_features_4</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_feature</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_linearized_data</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_max_neighborhood</td>
<td>87</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_neighbors_data</td>
<td>87</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_num_neighbors</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_num_samples_per_frame</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_type</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::get_word_size</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::load</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pilot2_molecular_reader::scale_data</td>
<td>86</td>
</tr>
<tr>
<td>lbann::pooling_layer</td>
<td>133</td>
</tr>
<tr>
<td>lbann::pooling_layer::copy</td>
<td>133</td>
</tr>
<tr>
<td>lbann::pooling_layer::get_data_layout</td>
<td>134</td>
</tr>
<tr>
<td>lbann::pooling_layer::get_description</td>
<td>134</td>
</tr>
<tr>
<td>lbann::pooling_layer::get_device_allocation</td>
<td>134</td>
</tr>
<tr>
<td>lbann::pooling_layer::get_type</td>
<td>134</td>
</tr>
<tr>
<td>lbann::reduction_layer</td>
<td>134</td>
</tr>
<tr>
<td>lbann::reduction_layer::copy</td>
<td>134</td>
</tr>
<tr>
<td>lbann::reduction_layer::get_data_layout</td>
<td>134</td>
</tr>
<tr>
<td>lbann::reduction_layer::get_data_layout</td>
<td>134</td>
</tr>
</tbody>
</table>

Index
lbann::sgd (C++ class), 159
lbann::sgd::copy (C++ function), 159
lbann::sgd::get_description (C++ function), 159
lbann::sgd::get_momentum (C++ function), 159
lbann::sgd::get_type (C++ function), 159
lbann::sgd::get_velocity (C++ function), 159
lbann::sgd::get_nesterov (C++ function), 159
lbann::sgd::set_momentum (C++ function), 159
lbann::sgd::set_nesterov (C++ function), 159
lbann::sgd::setup (C++ function), 159
lbann::sgd::using_nesterov (C++ function), 159
lbann::sgd_execution_context (C++ class), 93
lbann::sgd_execution_context::~sgd_execution_context (C++ function), 93
lbann::sgd_execution_context::copy_execution_context (C++ function), 93
lbann::sgd_execution_context::get_current_mini_batch_size (C++ function), 93
lbann::sgd_execution_context::get_effective_mini_batch_size (C++ function), 94
lbann::sgd_execution_context::get_epoch (C++ function), 93
lbann::sgd_execution_context::inc_epoch (C++ function), 93
lbann::sgd_execution_context::load_from_checkpoint_shared (C++ function), 94
lbann::sgd_execution_context::operator= (C++ function), 93
lbann::sgd_execution_context::save_to_checkpoint_shared (C++ function), 94
lbann::sgd_execution_context::serialize (C++ function), 93
lbann::sgd_execution_context::set_current_mini_batch_size (C++ function), 93
lbann::sgd_execution_context::set_effective_mini_batch_size (C++ function), 94
lbann::sgd_execution_context::sgd_execution_context (C++ function), 93
lbann::sgd_termination_criteria (C++ class), 94
lbann::sgd_training_algorithm (C++ class), 161
lbann::sgd_training_algorithm::train (C++ function), 162
lbann::sigmoid_layer (C++ class), 142
lbann::slice_layer (C++ class), 135
lbann::slice_layer::copy (C++ function), 135
lbann::slice_layer::get_data_layout (C++ function), 135
lbann::slice_layer::get_description (C++ function), 136
lbann::slice_layer::get_device_allocation (C++ function), 135
lbann::slice_layer::get_slice_points (C++ function), 135
lbann::slice_layer::set_type (C++ function), 135
lbann::softmax_layer (C++ class), 124
lbann::softmax_layer::bp_compute (C++ function), 125
lbann::softmax_layer::copy (C++ function), 124
lbann::softmax_layer::fp_compute (C++ function), 125
lbann::softmax_layer::fp_setup_outputs (C++ function), 124
lbann::softmax_layer::get_data_layout (C++ function), 124
lbann::softmax_layer::get_device_allocation (C++ function), 124
lbann::softmax_layer::get_type (C++ function), 124
lbann::softmax_layer::setup_dims (C++ function), 124
lbann::softmax_layer::setup_matrices (C++ function), 124
lbann::softmax_layer::set_current (C++ function), 124
lbann::softsign_layer (C++ class), 142
lbann::softsign_layer::copy (C++ function), 142
lbann::softsign_layer::get_type (C++ function), 136
lbann::softsign_layer::setup (C++ function), 136
lbann::sort_layer (C++ class), 136
lbann::sort_layer::copy (C++ function), 136
lbann::sort_layer::get_data_layout (C++ function), 136
lbann::sort_layer::get_device_allocation (C++ function), 136
lbann::sort_layer::get_type (C++ function), 136
lbann::sort_layer::setup (C++ function), 136
lbann::split_layer (C++ class), 136
lbann::split_layer::copy (C++ function), 136
lbann::split_layer::get_data_layout (C++ function), 136
lbann::split_layer::get_device_allocation (C++ function), 137
lbann::split_layer::get_type (C++ function), 136
lbann::training_algorithm::training_algorithm (C++ function), 161
lbann::transform::adjust_brightness (C++ class), 163
lbann::transform::adjust_brightness::adjust_brightness (C++ function), 163
lbann::transform::adjust_brightness::apply (C++ function), 163
lbann::transform::adjust_brightness::copy (C++ function), 163
lbann::transform::adjust_brightness::get_type (C++ function), 163
lbann::transform::adjust_contrast (C++ class), 164
lbann::transform::adjust_contrast::adjust_contrast (C++ function), 164
lbann::transform::adjust_contrast::apply (C++ function), 164
lbann::transform::adjust_contrast::copy (C++ function), 164
lbann::transform::adjust_contrast::get_type (C++ function), 164
lbann::transform::center_crop (C++ class), 165
lbann::transform::center_crop::apply (C++ function), 165
lbann::transform::center_crop::center_crop (C++ function), 165
lbann::transform::center_crop::copy (C++ function), 165
lbann::transform::center_crop::get_type (C++ function), 165
lbann::transform::color_jitter (C++ class), 166
lbann::transform::color_jitter::apply (C++ function), 166
lbann::transform::color_jitter::color_jitter (C++ function), 166
lbann::transform::color_jitter::copy (C++ function), 166
lbann::transform::color_jitter::get_type (C++ function), 166
lbann::transform::colorize (C++ class), 166
lbann::transform::colorize::apply (C++ function), 166
lbann::transform::colorize::copy (C++ function), 166
lbann::transform::colorize::get_type (C++ function), 166
lbann::transform::cutout (C++ class), 166
lbann::transform::cutout::apply (C++ function), 166
lbann::transform::cutout::copy (C++ function), 166
lbann::transform::cutout::get_type (C++ function), 166
lbann::transform::grayscale (C++ class), 167
lbann::transform::grayscale::apply (C++ function), 167
lbann::transform::grayscale::copy (C++ function), 167
lbann::transform::grayscale::get_type (C++ function), 167
lbann::transform::horizontal_flip (C++ class), 168
lbann::transform::horizontal_flip::apply (C++ function), 168
lbann::transform::horizontal_flip::copy (C++ function), 168
lbann::transform::horizontal_flip::get_type (C++ function), 168
lbann::transform::normalize (C++ class), 168
lbann::transform::normalize::apply (C++ function), 168
lbann::transform::normalize::copy (C++ function), 168
lbann::transform::normalize::get_type (C++ function), 168
lbann::transform::normalize::supports_non_inplace (C++ function), 168
lbann::transform::normalize_to_lbann_layout (C++ class), 169
lbann::transform::normalize_to_lbann_layout::apply (C++ function), 169
lbann::transform::normalize_to_lbann_layout::copy (C++ function), 169
lbann::transform::normalize_to_lbann_layout::get_type (C++ function), 169
lbann::transform::normalize_to_lbann_layout::supports_non_inplace (C++ function), 169
lbann::transform::random_affine (C++ class), 169
lbann::transform::random_affine::apply (C++ function), 170
lbann::transform::random_affine::copy (C++ function), 170
lbann::transform::random_affine::get_type (C++ function), 170
lbann::transform::random_crop (C++ class), 170
lbann::transform::random_crop::apply (C++ function), 170
lbann::transform::random_crop::copy (C++ function), 170
lbann::transform::random_crop::get_type (C++ function), 170
lbann::transform::random_crop::random_crop (C++ function), 170
lbann::transform::random_resized_crop (C++ class), 171
lbann::transform::random_resized_crop::apply (C++ function), 171
lbann::transform::random_resized_crop::copy (C++ function), 171
lbann::transform::random_resized_crop::get_type (C++ function), 171
lbann::transform::random_resized_crop::random_resized_crop (C++ function), 171
lbann::transform::random_resized_crop::scale::get_type (C++ function), 174
lbann::transform::random_resized_crop::with_fixed_aspect_ratio::scale::apply (C++ function), 174
lbann::transform::random_resized_crop::with_fixed_aspect_ratio::scale::copy (C++ function), 174
lbann::transform::random_resized_crop::with_fixed_aspect_ratio::scale::translate (C++ function), 174
lbann::transform::random_resized_crop::with_fixed_aspect_ratio::scale::translate::copy (C++ function), 174
lbann::transform::random_resized_crop::with_fixed_aspect_ratio::scale::translate::supports_non_inplace (C++ function), 174
lbann::transform::repack_HWC_to_CHW_layout::supports_non_inplace (C++ function), 175
lbann::transform::resize (C++ class), 172
lbann::transform::resize::copy (C++ function), 172
lbann::transform::resize::get_type (C++ function), 172
lbann::transform::resize::resize (C++ function), 172
lbann::transform::resized_center_crop (C++ class), 173
lbann::transform::resized_center_crop::apply (C++ function), 173
lbann::transform::resized_center_crop::copy (C++ function), 173
lbann::transform::resized_center_crop::get_type (C++ function), 173
lbann::transform::resized_center_crop::resized_center_crop (C++ function), 173
lbann::transform::sample_normalize (C++ class), 173
lbann::transform::sample_normalize::apply (C++ function), 173
lbann::transform::sample_normalize::copy (C++ function), 173
lbann::transform::sample_normalize::get_type (C++ function), 173
lbann::transform::sample_normalize::sample_normalize (C++ function), 173
lbann::transform::scale (C++ class), 174
lbann::transform::scale::copy (C++ function), 174
lbann::transform::scale::get_type (C++ function), 174
lbann::transform::scale::scale (C++ function), 174
lbann::transform::scale_and_translate (C++ class), 174
lbann::transform::scale_and_translate::apply (C++ function), 174
lbann::transform::scale_and_translate::copy (C++ function), 174
lbann::transform::scale_and_translate::scale_and_translate (C++ function), 174
lbann::transform::to_lbann_layout (C++ class), 175
lbann::transform::to_lbann_layout::apply (C++ function), 175
lbann::transform::to_lbann_layout::copy (C++ function), 175
lbann::transform::to_lbann_layout::get_type (C++ function), 175
lbann::transform::to_lbann_layout::supports_non_inplace (C++ function), 175
lbann::transform::resize::apply (C++ function), 175
lbann::transform::transform (C++ class), 162
lbann::transform::transform::apply (C++ function), 163
lbann::transform::transform::copy (C++ function), 163
lbann::transform::transform::get_description (C++ function), 163
lbann::transform::transform::get_type (C++ function), 163
lbann::transform::transform::supports_non_inplace (C++ function), 163
lbann::transform::transform_pipeline (C++ class), 176
lbann::transform::transform_pipeline::add_transform (C++ function), 176
lbann::transform::transform_pipeline::apply (C++ function), 176
lbann::transform::transform_pipeline::set_expected_out_dims (C++ function), 176
lbann::transform::vertical_flip (C++ class), 175
lbann::transform::vertical_flip::apply (C++ function), 175
lbann::transform::vertical_flip::copy (C++ function), 175
lbann::transform::vertical_flip::get_type (C++ function), 175
lbann::transform::vertical_flip::vertical_flip (C++ function), 175
lbann::transform_layer (C++ class), 126
lbann::type_erased_function (C++ class), 184
lbann::type_erased_function::operator() (C++ function), 185
lbann::type_erased_function::operator= (C++ function), 185
lbann::type_erased_function::type_erased_function (C++ function), 184, 185
lbann::uniform_initializer (C++ class), 188
lbann::uniform_initializer::copy (C++ function), 188
lbann::uniform_initializer::fill (C++ function), 188
lbann::uniform_initializer::get_description (C++ function), 188
lbann::uniform_initializer::get_type (C++ function), 188
lbann::uniform_layer (C++ class), 138
lbann::uniform_layer::copy (C++ function), 138
lbann::uniform_layer::get_data_layout (C++ function), 138
lbann::uniform_layer::get_description (C++ function), 138
lbann::uniform_initializer::get_device_allocation (C++ function), 138
lbann::uniform_layer::get_type (C++ function), 138
lbann::unpooling_layer (C++ class), 138
lbann::unpooling_layer::copy (C++ function), 138
lbann::unpooling_layer::get_data_layout (C++ function), 139
lbann::unpooling_layer::get_device_allocation (C++ function), 139
lbann::unpooling_layer::get_layer_pointers (C++ function), 139
lbann::unpooling_layer::set_layer_pointers (C++ function), 139
lbann::unpooling_layer::setup_dims (C++ function), 139
lbann::unpooling_layer::setup_pointers (C++ function), 139
lbann::update_data_reader_functor (C++ class), 95
lbann::utils::any (C++ class), 177
lbann::utils::any::any (C++ function), 177
lbann::utils::any::~any (C++ function), 177
lbann::utils::any::any (C++ function), 177
lbann::utils::any::any::emplace (C++ function), 177
lbann::utils::any::has_value (C++ function), 178
lbann::utils::any::operator== (C++ function), 177
lbann::utils::any::reset (C++ function), 178
lbann::utils::any::swap (C++ function), 178
lbann::utils::any::type (C++ function), 178
lbann::utils::type_erased_matrix (C++ class), 181
lbann::utils::type_erased_matrix::emplace (C++ function), 182
lbann::utils::type_erased_matrix::get (C++ function), 182
lbann::utils::type_erased_matrix::type_erased_matrix (C++ function), 181, 182
lbann::value_initializer (C++ class), 188
lbann::value_initializer::copy (C++ function), 188
lbann::value_initializer::fill (C++ function), 188
lbann::value_initializer::get_description (C++ function), 188
lbann::value_initializer::get_type (C++ function), 188
lbann::variance_layer (C++ class), 140
lbann::variance_layer::copy (C++ function), 140