Blocks and Fuel: GPU-accelerated neural networks for large data

Bart van Merriënboer BART.VAN.VANMERRIENBOER@UMONTREAL.CA
Montreal Institute for Learning Algorithms, University of Montreal, Montreal, Canada

Dzmitry Bahdanau D.BAHDANAU@JACOBS-UNIVERSITY.DE
Jacobs University, Bremen, Germany

Jan Chorowski JAN.CHOROWSKI@II.UNI.WROC.PL
University of Wrocław, Wrocław, Poland

Vincent Dumoulin DUMOULIV@IRO.UMONTREAL.CA
Dmitriy Serdyuk SERDYUK.DMITRIY@GMAIL.COM
David Warde-Farley WARDEFAR@IRO.UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Montreal Institute for Learning Algorithms, University of Montreal, Montreal, Canada

Editor: ?

Abstract

We introduce two Python frameworks to train neural networks on large datasets: *Blocks* and *Fuel*. *Blocks* is based on the Theano, a linear algebra compiler with CUDA-support (**??**). It facilitates the training of complex neural network models by providing parametrized Theano operations, attaching metadata to Theano's symbolic computation graph, and providing an extensive set of utilities to assist training the networks, e.g. training algorithms, logging, monitoring, visualization, and serialization. *Fuel* provides a standard format for machine learning datasets. It allows the user to easily iterate over large datasets, performing many types of pre-processing on the fly.

Keywords: Neural networks, GPGPU, large-scale machine learning

1. Introduction

Blocks is a framework that builds on Theano to facilitate the training of neural networks. It is being developed by the Montreal Institute of Learning Algorithms (MILA) at the University of Montreal. Its focus lies on quick prototyping of complex neural network models. *Fuel*

2. History

- Pylearn2: More general (machine learning vs. neural networks), abstracts away Theano
- Lasagne, Keras: Stronger focus on easy implementation of existing models than the development of new models
- scikit-learn: More plug-and-play, less research-oriented

3. Blocks

Theano is a popular choice for the implementation of neural networks (see e.g. **??**). Blocks and many other libraries, such as Pylearn2 **?**, build on Theano by providing reusable components that are common in neural networks, such as linear transformations followed by non-linear activations, or more complicated components such as LSTM units. In Blocks these components are referred ot as *bricks* or "parametrized Theano operations".

Bricks take the form of a set of parameters in the form of Theano shared variables, for example the weight matrix of a linear transformation or the filters of a convolutional layer. Bricks use these parameters to transform symbolic Theano variables, potentially in multiple ways.

Bricks can contain other bricks within them. This effectively introduces a hierarchy of structures on top of the flat computation graph defined by Theano, which makes it easier to address and configure complex models programmatically.

The parameters of bricks can be initialized using a variety of schemes that are popular in the neural network literature, such as sparse initialization, orthogonal initialization for recurrent weights, etc.

Large neural networks can often result in Theano computation graphs containing hundreds of variables and operations. Blocks does not attempt to abstract away this complex graph, but to make it manageable by annotating variables in the graph. Each input, output, and parameter of a brick is annotated as such. Variables can also be annotated with the role they play in a model, such as *weights*, *biases*, *filters*, etc. A series of convenience tools were written that allow users to filter the symbolic computation graph based on these annotations, resulting in an approach that allows for powerful queries such as "Apply weight noise to all weights that belong to an LSTM unit whose parent is a brick with the name ..."

4. Fuel

Fuel's goal is to provide a common interface to a variety of data formats and published datasets such as MNIST, CIFAR-10, ImageNet, etc. while making it easy for users to write an interface to new datasets. Fuel allows for efficient ways of iterating over these datasets (sequentially, shuffled, minibatches, etc.). It also provides a variety of on-the-fly preprocessing methods such as random cropping of images, creating n-grams from text files, and the ability to implement many other methods easily.

Blocks relies on Fuel for its data interface, but Fuel can easily be used by other machine learning frameworks that interface with datasets.

4.1 Standardized data format

Fuel concentrates most of its built-in datasets around one common interface: HDF5. In addition to being an efficient format for large datasets that don't fit into memory, HDF5 is very easy to organize and document. This allows to simplify dataset storage by putting everything into one file and documenting how this file is organized with metadata.

The main class for interfacing with HDF5 datasets is the `H5PYDataset` class. Given a series of assumptions about how data is organized within an HDF5 file, it is able to parse:

- What are the data sources available (e.g. features, targets, etc.)?
- How are these data sources officially split (e.g. training, test sets)?
- Are some data sources unavailable for some splits (e.g. test set only offers unlabeled examples)?
- What are the axes semantics for a given data source (e.g. batch, feature, width, height, channel, time, etc.)?

TODO: explain these assumptions in more details?

This class is flexible enough that most built-in datasets are little more than a wrapper that defines the default location of the dataset file and some sensible transformations to apply to the data by default. Integrating user data into Fuel is also very straightforward if these assumptions are respected.

For small datasets that fit into memory, reading off disk is inefficient. `H5PYDataset` supports loading data into memory with the use of a `load_in_memory` constructor argument.

4.2 Automated data management

Fuel offers built-in scripts that automate the task of downloading and converting raw data files.

The `fuel-download` script is used to download raw data files. Downloading the raw MNIST data files is as easy as typing `fuel-download mnist`. Under the hood, `fuel-download` uses the `argparse` module along with subparsers so that each built-in dataset can define its own specific command-line arguments. For instance, datasets whose download URL is not publicly available could require that the URL is passed as argument to `fuel-download`.

The `fuel-convert` script is used to convert raw data files into a format that Fuel understands, which for most cases is the aforementioned HDF5 format. Converting the raw MNIST data files into an HDF5 file translates to the the `fuel-convert mnist` command. Like for `fuel-convert`, `fuel-download` uses `argparse` along with subparsers for flexibility. This feature is especially handy for large datasets with no standardized preprocessing, such as ImageNet.

Reproducibility being a very important feature in research, the `fuel-convert` script automatically tags all files it creates with relevant module and interface versions and the exact command that was used to generate these files. Inspection of this metadata is done with the `fuel-info` script.

5. Serialization and checkpointing

The training of large, deep neural networks can often take days or even weeks. Hence, regular checkpointing of training progress is important. Blocks aims to make the resumption of experiments entirely transparent, even across platforms, while ensuring the reproducibility of these experiments.

This goal is complicated by shortcomings in Python's Pickle serialization module, which is unable to serialize many iterators, which Fuel heavily depends on in order to iterate over large datasets efficiently. To circumvent this we reimplemented the itertools from the Python standard library to be serializable.

As a result, Blocks experiments are able to be interrupted in the middle of a pass over the dataset, serialized, and resumed later, without affecting the final training results.

6. Documentation and community

Blocks and Fuel are well documented, with both API documentation and tutorials available online. Two active mailing lists[1] support users of the libraries. A separate repository[2] is maintained for users to contribute non-trivial examples of the use of Blocks. Implementations of neural machine translation models (NMT, **?**) and the Deep Recurrent Attentive Writer (DRAW, **?**) model are publicly available examples of state-of-the-art models successfully implemented using Blocks.

7. NOTES

Some topics to discuss?

- Serialization/resumability
- "Bricks" i.e. parametrized Theano operations
- Graph management through annotation, hierarchy
- Implementations of GRU, LSTM, attention mechanism, convolutional networks
- Monitoring non-Theano variables, aggregation, plotting results
- Composing custom algorithms from bits and pieces called step rules
- Automatic downloading and converting of files
- Standard HDF5 format that conveys axis semantics, splits, etc.

Acknowledgments

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. We would also like to thank the developers of Theano.

1. `https://groups.google.com/d/forum/blocks-users` and `https://groups.google.com/d/forum/fuel-users`
2. `https://github.com/mila-udem/blocks-examples`