INNOVATIVE COMPUTING LABORATORY

RAPYDLI

RAPYDLI GOALS

Environment supporting broad base of deep learning

Our approach is based on a general framework that includes various dimensions of the deep network layers to allow us to provide high performance kernels, not just for one particular network type or configuration, which might be of great interest at the moment, but for broader applications that utilize a variety of neural network layers, dimensionality, and connectivity. The goal is to first target the popular classifier networks of interest to the community and then broaden the scope further to accommodate other uses, e.g., object detection, etc.

Data management

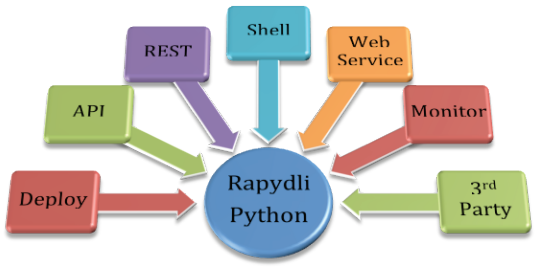
Deep Learning researchers will benefit from the ease of use and highly productive RaPyDLI environment. It will allow users rapid prototyping or interactive experimentation with new algorithms and apply to vast datasets with billions or trillions of parameters. Our data storage and access interface is associated with multiple stages in the deep learning process such as input training data records composed of feature sets, metadata shared by parallel workers during the learning process, and output records describing the trained models. The primary computation is performed on N-dimensional arrays of numerical data. We define a generic high level interface for data load and save operations. They take a set of input data files, a subset extraction policy, a partition policy, a distribution policy, and a list of computation nodes as input parameters. We support heterogeneous data storage backends from parallel or distributed file systems (e.g., Lustre, HDFS) and NoSQL databases (e.g., HBase) with uniform I/O interfaces.

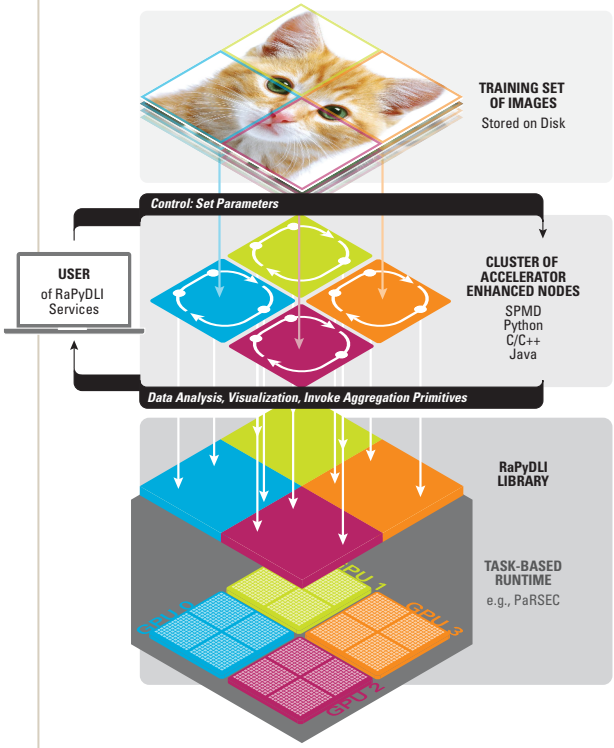
Rapid Prototyping

RaPyDLI can be used as an on-ramp to deep learning. It serves as an aggregate for modules related to deep learning while simplifying access via a service interface and hosted services that provide access to fast computing resources. To achieve this we need to address a multitude of issues, including deployment, access, and integration of other frameworks. We offer simple interfaces to the deployment framework but also the actual algorithmic functionality through a variety of interfaces. This includes a Python API, REST, a command shell, and Web Services. In addition, we will provide an experiment management framework that allows us to monitor the services and resources.

Interoperability/ portability with very good performance

Our convolution kernel, at the heart of deep learning neural networks for some of the network layers, achieves almost the same performance as the cuDNN library provided by NVIDIA. Both of these implementations are competitive replacements of the most time consuming kernels of the University of California at Berkeley's Caffe project for deep neural networks. Almost the same percent of the peak performance is achieved by our kernels on both Kepler and the newly released Maxwell GPU cards. The competing implementations are either closed source (NVIDIA's cuDNN) and do not contribute to the broader impact that our optimization techniques have, or they are platform specific (maxDNN is a convolutional network implementation from Ebay based on a Maxwell-only assembler from Nervana).





ICL INNOVATIVE COMPUTING LABORATORY

INDIANA UNIVERSITY

STANFORD UNIVERSITY

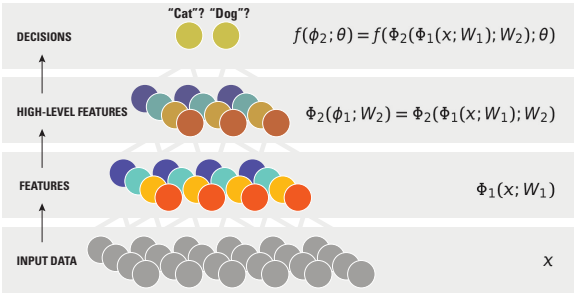
THE UNIVERSITY OF TENNESSEE KNOXVILLE

SPONSORED BY National Science Foundation

INNOVATIVE COMPUTING LABORATORY

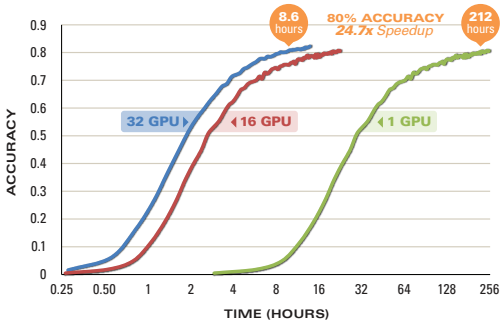
RaPyDLI

DEEP LEARNING PROBLEM



Learning Networks take input data, such as a set of pixels in images, and map them into decisions, such as labelling objects in an image. The inputs and outputs are linked by multiple nodes in a layered network. A (very) deep learning (DL) network has multiple layers (intermediate nodes) - perhaps 10 or more. DL networks have the advantage of being very general: no need for application-specific models, but rather general structures suffice, such as in convolutional neural networks that generate translation and rotation invariant mappings. DLs are now used in all major commercial speech recognition systems. The quality of a DL depends on the number of layers and nature of nodes in the network. These choices require extensive prototyping with rapid turnaround on trial networks. RaPyDLI aims to enable this prototyping with an attractive Python interface and a high performance DL network computation engine for the training stage.

DEEP LEARNING STATE OF THE ART



The Image-net Large Scale Visual Recognition Challenge (ILSVRC) has become the most significant large scale benchmark for computer vision. Since the success of AlexNet in 2012, Convolutional Neural Networks (CNNs) have been essential to obtaining state-of-the-art performance on ILSVRC. The trend in CNNs since AlexNet has been to increase network size and data set size (through augmentation). Both of these require substantially more computational power which motivates a need for software that can utilize multiple GPUs while still allowing for rapid prototyping. In the current state-of-the-art model for ILSVRC, Baidu showed how they were able to train their network using 32 GPUs in 8.6 hours to 80% accuracy, something that would've taken 212 hours on a single GPU (a speedup of 24.7x). Similar speedups have now been achieved throughout the AI industry.

RAPYDLI APPROACH

Python Frontend

Multiple frontends are provided to make it possible to utilize RaPyDLI as a developer and user platform. The RaPyDLI Frontend will therefore become an OnRamp to deep learning and the deployment of hosted RaPyDLI services. A prototype hosted web service is used to demonstrate the capabilities of the RaPyDLI framework. These services can then be choreographed to deploy and stage services that users may need. Our RaPyDLI framework can leverage HPC, Clouds, or Grids through service orchestration.

Runtime

We are investigating two parallelization potentials offered by the deep neural networks: data-parallel and model-parallel. The latter allows Bulk Synchronous Parallel execution with great potential for locality-aware computing with sporadic data interchanges of the border neurons and synapses. The former gives rise to independent calculation with concurrent access to the global state. Our work focuses on combining both and improving on Stanford's FastLab code base for deep neural networks that expose both types of parallelism. We are using our dataflow runtime systems that support both multicore processors and accelerators.

Autotuning

Our kernel work is based on an autotuning framework that allows generic expression of a computational stencil. The simplest cases are templates and/or stencils based on multiple loop nests–a perfect case for deep neural network research. Given a generic code and the parameter space, our framework discovers a subspace of the parametric space that is allowed by the hardware (hard constraint pruning) and has not been excluded by the user (soft constraint pruning). After the extensive pruning, we enumerate all possible instantiations of the kernel with all parameters replaced by constants, which gives the compiler the opportunity to generate a very specific–and often superior–code when compared with variable-length loop nests. The execution harness tests the produced binaries and selects the best performer, sometimes with a surprising set of parameter values due to complex interaction between the kernel template, the compiler toolchain, and the hardware configuration.

Hbase

To effectively support parallel computation of the deep learning library, we investigate the storage I/O interface that provides operations in three stages: input data loading, snapshot of intermediate parameters, and the resulting collection. For example, various optimizations will be applied for the partition and distribution steps. We've compared existing database approaches in Caffe using an AlexNet model on ImageNet Data. LMDB and LevelDB outperform direct image file loading by caching nearby image data using table and block cache. Beyond the parallel file I/O operations, RaPyDLI will also provide fine-grained record-level I/O operations using HBase and further compose higher-level abstractions, including N-dimensional array-based indexing.

ICL INNOVATIVE COMPUTING LABORATORY

INDIANA UNIVERSITY

STANFORD UNIVERSITY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

SPONSORED BY National Science Foundation