Comparing the costs of abstraction for DL frameworks

Maksim Levental and Elena Orlova
University of Chicago, Chicago IL

Introduction

Deep Learning (DL) frameworks represent neural network models as dataflow and computation graphs (where nodes correspond to functional units and edges correspond to composition). In recent years, there has been a proliferation of DL frameworks implemented as domain-specific languages (DSLs) embedded in "high-level" languages such as Python, Java, and C#. These DSLs serve as abstractions that aim to map the DL graphs to hardware pipelines. That is to say, they hide details of DL models that are judged to be either irrelevant or too onerous to consider. Thus, our intent here is to investigate the costs of some of the abstractions employed by framework developers. In particular we focus on the PyTorch framework and ecosystem deployed to GPUs.

CUDA semantics and ResNet-50

```
template <typename dtype>
__global__ void softmax_loss_kernel(
    dtype *reduced_loss,
    dtype *predict,
    dtype *target,
    dtype *workspace,
    int batch_size,
    int num_outputs) {

    int thread_id = blockDim.x * blockIdx.x + threadIdx.x;
    extern __shared__ dtype s_data[];
    dtype loss = 0.f;

    for (int c = 0; c < num_outputs; c++)
        loss -= (
            target[thread_id * num_outputs + c] *
            logf(predict[thread_id * num_outputs + c]
        );
    workspace[thread_id] = loss;
    // ...
}
```

Figure 1: Softmax loss CUDA kernel

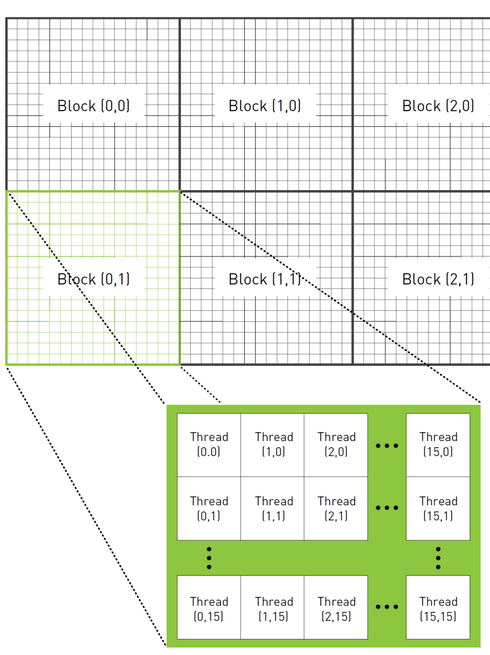


Figure 2: Mapping from thread and block to matrix element [1].

stage	output	ResNet-50
conv1	112×112	7×7, 64, stride 2
		3×3 max pool, stride 2
conv2	56×56	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3	28×28	[1×1, 128; 3×3, 128; 1×1, 512] ×4
conv4	14×14	[1×1, 256; 3×3, 256; 1×1, 1024] ×6
conv5	7×7	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	global average pool
		1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$

Figure 3: ResNet-50 network architecture [2]. Note that each convolution is followed by a batch normalization unit and each "stage" is followed by a ReLU (residual connections omitted).

References

[1] Sanders, J. and Kandrot, E., *CUDA by Example: An Introduction to General-Purpose GPU Programming* (Addison-Wesley Professional, 2010), first edn.

[2] He, K. et al, Deep residual learning for image recognition (2015).

Methods

We implement the popular object detection deep neural network ResNet-50 [2] at four levels of abstraction (PyTorch, TorchScript[a], LibTorch, and cuDNN) in order to investigate the differences amongst them. We measure accuracy, execution time, GPU utilization, and memory efficiency of each implementation on four image datasets (MNIST, CIFAR10, STL10, PASCAL). The source for the implementations is available on GitHub[b]. The datasets were chosen because they span the spectrum of image complexity (from small single-channel images to large multi-channel images). The reasons for choosing ResNet-50 are two fold. Firstly, it serves as a benchmark architecture in the research community. Secondly, it includes functional units included in many other network architectures (residual units, convolutions of various sizes, batch normalizations, ReLU activations, and pooling layers) and is therefore representative of typical neural network compute workloads.

[a] TorchScript models are serializations of PyTorch models but can run in inference mode in C++, i.e. sans Python runtime.
[b] https://github.com/makslevental/pytorch_abstraction_comparison

Experimental Results

The PyTorch implementation compares **favorably** with both LibTorch and the cuDNN implementations in terms of accuracy. On MNIST and CIFAR10 all three implementations perform reasonably well; LibTorch and PyTorch attain maximum accuracy at around the same time while cuDNN lags behind. In terms of execution time and memory usage PyTorch compares **unfavorably** with each of the other implementations. We measure execution time, memory usage, and GPU utilization during evaluation on PASCAL for various batch sizes and resolution.

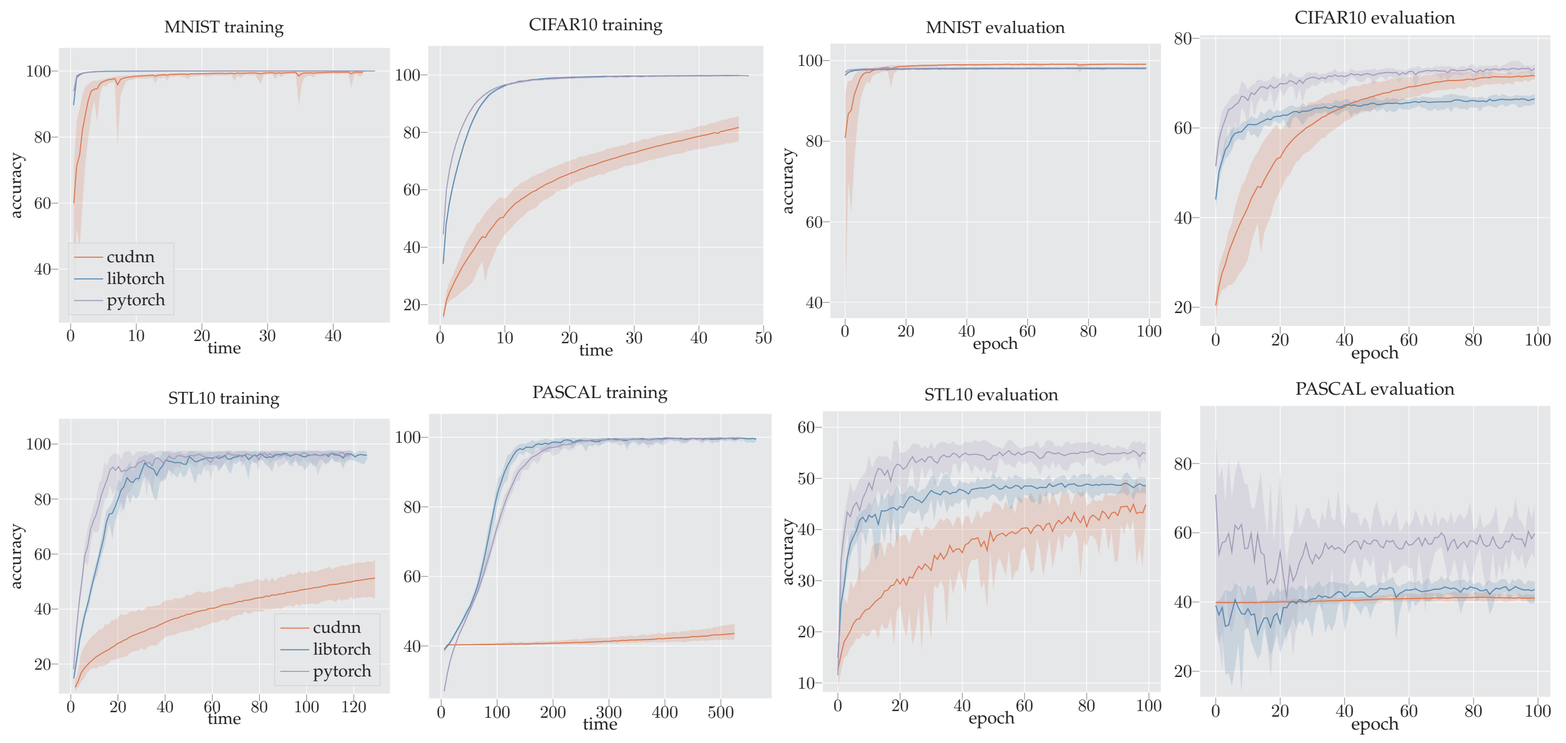
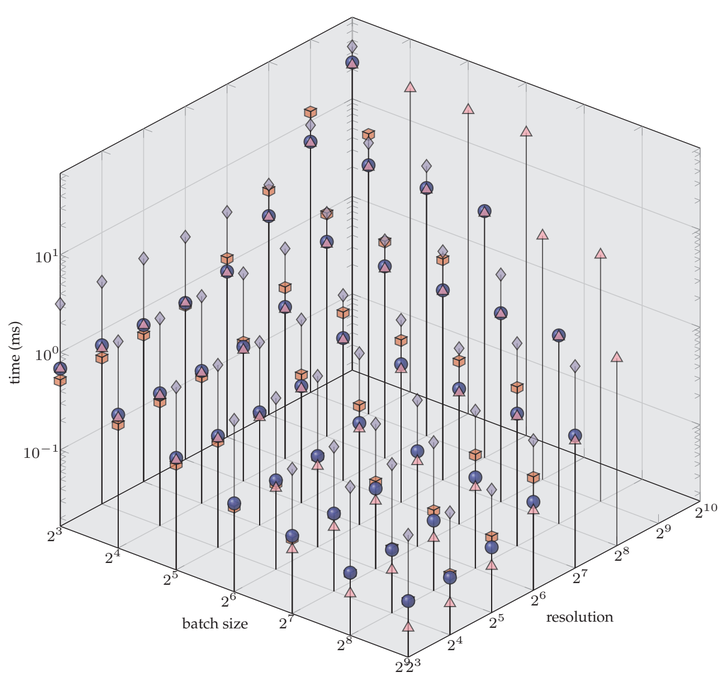
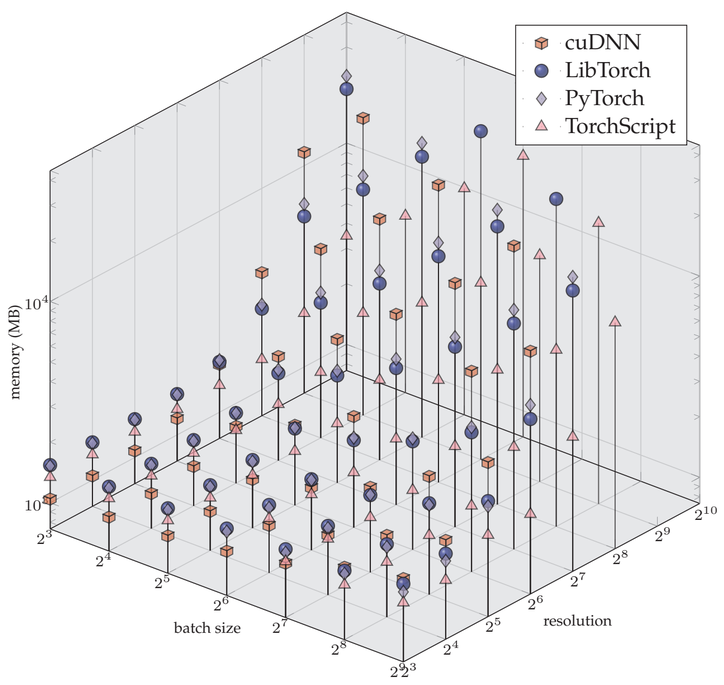


Figure 4: Comparison of accuracy for PyTorch, LibTorch, and cuDNN implementations during training and evaluation. Solid line corresponds to mean while shaded regions correspond to min and max. Time is measured in units of *epoch* × *average epoch time*.



(a) Average sample time in evaluation



(b) Average memory used in evaluation

(c) Average sample time in training

(d) Average memory used in training

Figure 5: Execution time and memory efficiency comparison of accuracy for PyTorch, TorchScript, LibTorch, cuDNN implementations during training and evaluation on the PASCAL dataset.