Deep Learning Frameworks Portability Survey: Post-K Perspective

Aleksandr Drozd[2], Mohamed Wahib[3], Yosuke Oyama[2], Artur Podobas[2], Jens Domke[2], Satoshi Matsuoka[1,2,3]

1: RIKEN-CCS, Kobe, Japan
2: Tokyo Institute Of Technology, Tokyo. Japan
3: AIST/TokyoTech Open Innovation Laboratory, Tokyo, Japan

13th January 2019

Why This Report?

Prof. Matsuoka
R-CCS Director

What is the path forward, and effort, required to qualify Post-K for supporting DL software stacks?

Fundamental Conditions:

- Post-K MUST support training DNNs, not just inference
- No branching/custom versions of DL frameworks

Summary

- The issue is not just a low-level primitive library
  - How to interface it with a DL framework?
- Different DL frameworks are designed differently
  - Different tradeoffs WRT supporting new hardware
- TensorFlow (TF) is modular
  - Defining and interfacing a new backend is relatively straightforward
- Chainer does not provide separation of concerns
  - Yet codebase is small: support could be added
- Model exchange formats (ONNX and NNEF)
  - Might support training in the future

Survey Methodology

- We investigate DL software solutions
  - At different abstraction levels of DL domains

- Automated approach
  - Use basic DNN primitives (eg: activation func, dense layer)
  - Trace execution paths and call stacks
  - Collecting code fragments
    - At which GPU/CPU take different execution paths

Scope of Survey

- Objective of survey is ARM CPU platform

- Focus on x86 CPUs, not GPU
  - However, frameworks not thoroughly optimized for CPU
  - For some frameworks not optimized at all
  - So we focus heavily on studying GPU backends

Overview of DL Software Stacks

- DL frameworks include the following layers of abs:
  - User interface
    - API
    - Import/export formats
  - Intermediate representation / graph-level optimizations
  - Tensor computation backend
    - Hard-coded
    - Code generation
    - DL Primitives library
    - BLAS libraries

Model Exchange Formats

- Allows moving between frameworks
- ONNX (Open Neural Network eXchange)
  - Extensible graph computation model
  - Built-in operators and data types
  - Limitations:
    - No cycles allowed
    - Must adhere to SSA
    - Only NCHW layout supported
    - No backprob

ONNX support, as of OCT 2018

Framework	Exporting	Importing
Caffe2	✓	✓
PyTorch	✓	
CNTK	✓	✓
MXNet	✓	✓
Chainer	✓	
TensorFlow	✓	(✓)

- NNEF (Neural Network Exchange Format)
  - Not a graph engine: can not be executed

DL Libraries for Performance Primitives

- Most computations in DNN can use BLAS primitives
- However, more efficient and specialized algo. Used
- Algorithms are common to DL frameworks
  - Implementation is HW specific
  - Lib. typically developed/maintained by HW vendors

cuDNN (Nvidia)

- Closed source
- Provides primitives of DNNs (including backprob):
  - N-dim convolution || Pooling || softmax || Activiation
  - Tensor transformations || Normalization || RNN
- Includes up to eight different convolution algorithms
- cuDNN is evolving to allow fusing operators
  - ex: fusing batchnorm with add and Relu operations

MKL-DNN (Intel)

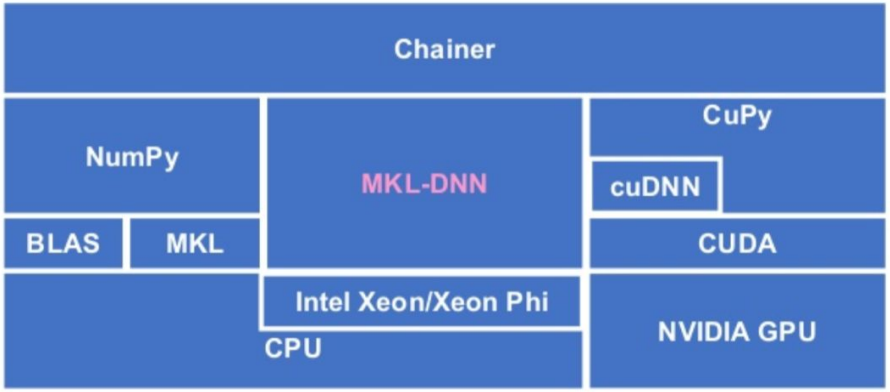
- Open source
  - Yet relies heavily on MKL (closed source)
- Support many primitives
  - For both x86 and Xeon Phi
  - Most primitives in fp32 (some also in int8)
  - No mention of reduced precision training
- MKL-DDN is a rough proxy for the effort required
  - ~81k lines of code
  - Supports only fp32, for a range of x86 processors

ARM NN SDK (ARM)

- Open source
  - Current support is for Caffee and TF (not complete)
- Supports ARM Cortex CPUs and Mali GPUs
- Forward propagation only
  - No means to derive backwards computation
  - Models are read from prototxt format
  - prototxt presents the computation graph
- ~21k lines of code using C++ templating

Chainer (1 of 2)

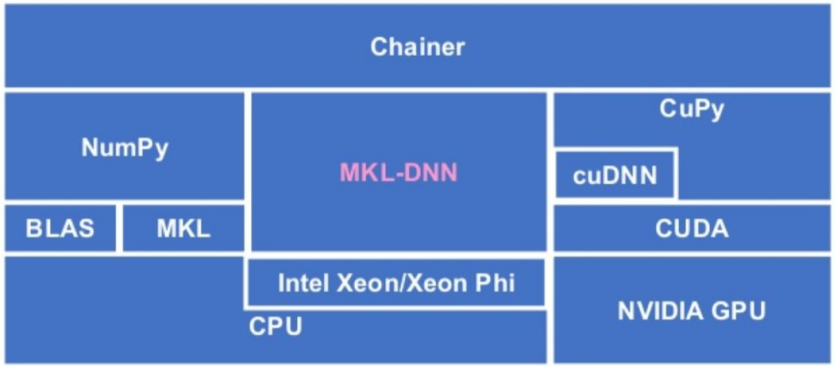
- A Python-based framework widely used in Japan
  - Heavy geared towards GPUs
  - Define-by-run approach (AKA dynamic comp. graph)

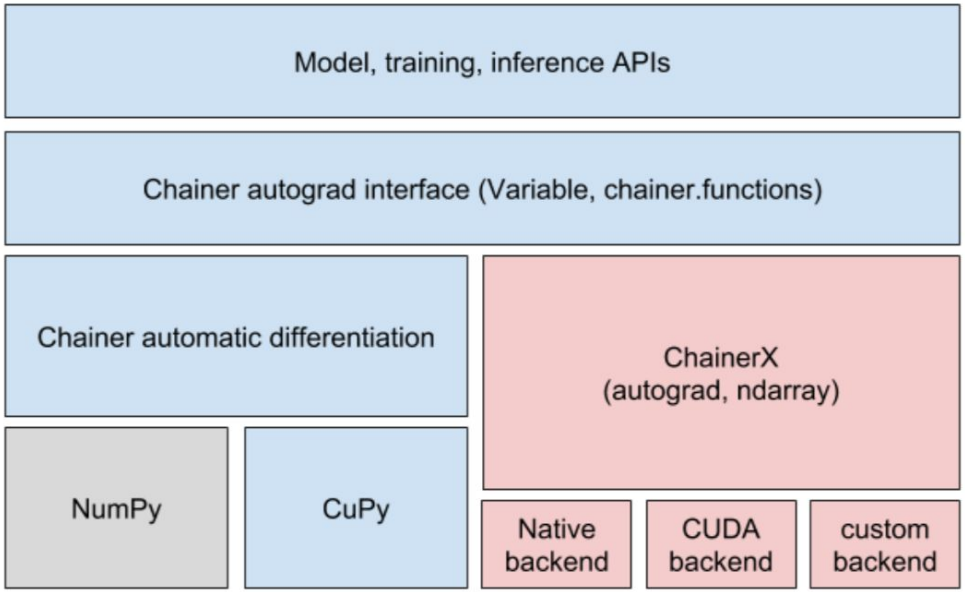


  - Not modular: separate implementations hardcoded
    - Think #if #def style programming
  - Uses NumPy as containers for Tensors
  - Distributed training uses MPI (and NCCL)

Chainer (2 of 2)

- ChainerX is a new variant that is modular
  - Not clear if it will replace Chainer or not

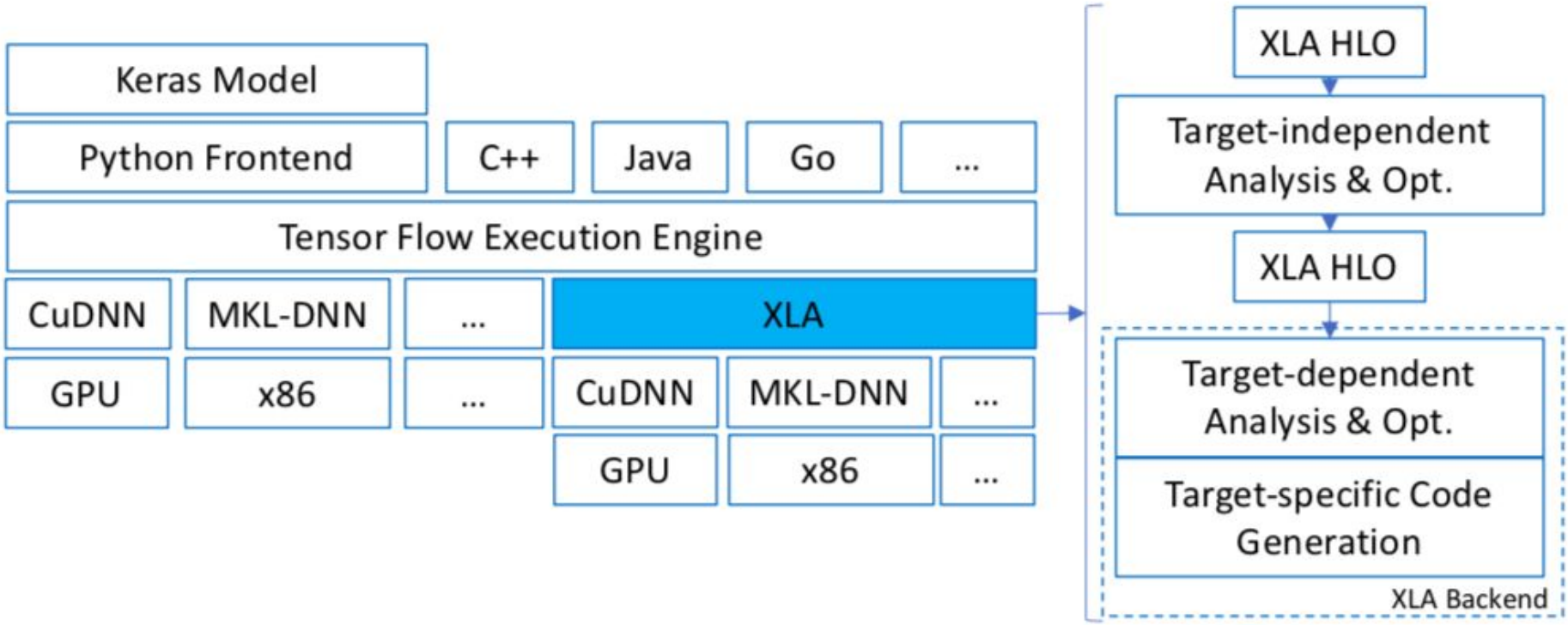




Chainer → ChainerX

TensorFlow (1 of 2)

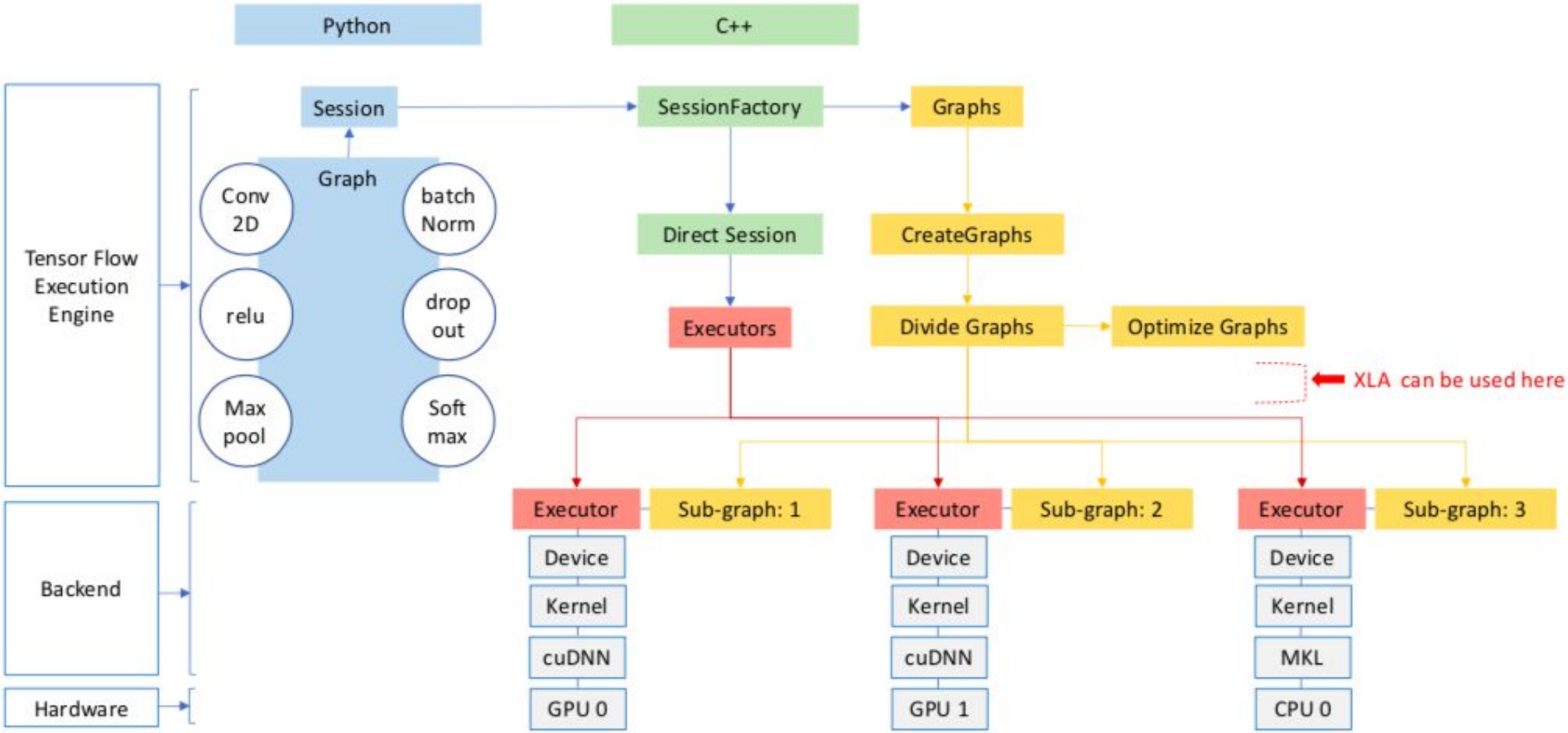
- A symbolic math library for tensor computation
  - Different frontends and backends
  - Current uses Keras as official Python frontend
  - Includes a compiler for linear algebra (XLA)



TensorFlow (2 of 2)

- Extending the frontend
  - One can define and add “operators”
  - TF can then a graph with the new “operators”
  - Relatively easy
    - Requires some knowledge of internal C++ code used
- Extending the backend
  - The programmer has to define how the graph is exec.
    - ex: Fujitsu A64FX has four CMGs to split the graph to
- Distributed training
  - Google’s gRPC (and MPI version also exists)
    - Scaling issues, still remains commonly used
    - Mesh-TensorFlow (model-parallelism)

A detailed look at TensorFlow



Summary

- The issue is not just a low-level primitive library
  - How to interface it with a DL framework?
- Different DL frameworks are designed differently
  - Different tradeoffs WRT supporting new hardware
- TensorFlow (TF) is modular
  - Defining and interfacing a new backend is relatively straightforward
- Chainer does not provide separation of concerns
  - Yet codebase is small: support could be added
- Model exchange formats (ONNX and NNEF)
  - Might support training in the future

Forward Convolution with cuDNN

```
int n, c, c_, h, h_, p, s, d,ws; // Layer parameters
half *x, *y, *w, *ws; // Tensors
int wsLimit = 64*1024*1024;
cudnnConvolutionFwdAlgo_t algo;

// Create cuDNN handle/descriptors
cudnnHandle_t handle;
cudnnTensorDescriptor_t xDesc, yDesc;
cudnnFilterDescriptor_t wDesc;
cudnnConvolutionDescriptor_t convDesc;
cudnnCreateTensorDescriptor(&xDesc);
cudnnCreateTensorDescriptor(&yDesc);
cudnnCreateFilterDescriptor(&wDesc);
cudnnCreateConvolutionDescriptor(&convDesc);
cudnnCreate(&handle);

// Initialize descriptors
cudnnSetTensor4dDescriptorEx(xDesc, CUDNN_DATA_HALF,
                             n, c, h, h, h*h*c, 1, h*c, c);
cudnnSetTensor4dDescriptorEx(yDesc, CUDNN_DATA_HALF,
                             n, c_, h_, h_, h_*h_*c_, 1, h_*c_, c_);
cudnnSetFilter4dDescriptor(wDesc, CUDNN_DATA_HALF, wFormat,
                           c_, c, k, k);
cudnnSetConvolution2dDescriptor(convDesc,
                                p, p, s, s, d, d,
                                CUDNN_CROSS_CORRELATION, CUDNN_DATA_HALF);
cudnnSetConvolutionMathType(convDesc, CUDNN_TENSOR_OP_MATH);

// Get a convolution algorithm
cudnnGetConvolutionForwardAlgorithm(handle,
                                    xDesc, wDesc, convDesc, yDesc,
                                    CUDNN_CONVOLUTION_FWD_SPECIFY_WORKSPACE_LIMIT,
                                    wsLimit,
                                    &algo);

// Perform convolution: Y = alpha (X * W) + beta Y
const float alpha = 1.0, beta = 0.0;
cudnnConvolutionForward(handle,
                        &alpha,
                        xDesc, x,
                        wDesc, w,
                        convDesc,
                        algo,
                        ws,
                        wsLimit,
                        &beta,
                        yDesc, y);
```

Using MKL-DNN by DL Frameworks

```
auto relu_desc = eltwise_forward::desc(prop_kind::forward,
        algorithm::eltwise_relu, conv_pd.dst_primitive_desc().desc(),
        negative_slope);
auto relu_pd = eltwise_forward::primitive_desc(relu_desc, cpu_engine);
auto relu_dst_memory = memory(relu_pd.dst_primitive_desc());
auto relu = eltwise_forward(relu_pd, conv_dst_memory, relu_dst_memory);

net_fwd.push_back(relu);

/* Backward */
auto relu_diff_dst_md = lrn_diff_src_memory.get_primitive_desc().desc();
auto relu_src_md = conv_pd.dst_primitive_desc().desc();

      relu_diff_dst_md, relu_src_md, negative_slope);
       relu_diff_dst_md, relu_src_md, negative_slope);
auto relu_bwd_pd
       = eltwise_backward::primitive_desc(relu_bwd_desc, cpu_engine, relu_pd);
auto relu_diff_src_memory = memory(relu_bwd_pd.diff_src_primitive_desc());
auto relu_bwd = eltwise_backward(relu_bwd_pd, conv_dst_memory,
    lrn_diff_src_memory, relu_diff_src_memory);

net_bwd.push_back(relu_bwd);
```