DEEP LEARNING ON THE INTEL® NERVANA™ PLATFORM

Ravi Panchumarthy, PhD
Machine Learning Engineer, Intel®
Outline

• **Intel® Nervana™ Overview**

• Intel’s Deep Learning Hardware

• Intel’s Deep Learning Software
  - MKL-DNN
  - Neon and Nervana Cloud
  - Nervana Graph
Intel® Nervana™ Deep Learning Platform

Compressing the “innovation cycle” to deliver custom, high-accuracy, enterprise-grade DL solutions in record time

Images

Video

Text

Speech

Tabular

Time series

Hosted Nervana Cloud -or-
On-premise Nervana DL Appliance -or-
Single-user version

Frameworks:
Neon, Caffe2*,
TensorFlow*, MXNet*,
Torch*

*Multiple framework support coming soon.
Outline

• Intel® Nervana™ Overview

• **Intel's Deep Learning Hardware**

• Intel’s Deep Learning Software
  - MKL-DNN
  - Neon and Nervana Cloud
  - Nervana Graph
End-to-End AI Compute

**DATACENTER**
Many-to-many hyperscale for stream and massive batch data processing

**GATEWAY**
1-to-many with majority streaming data from devices

**EDGE**
1-to-1 devices with lower power and often UX requirements

- **Ethernet & Wireless**
- **Wireless and non-IP wired protocols**
  - Secure
  - High throughput
  - Real-time

**Intel® Xeon® Processors**

**Intel® Xeon Phi™ Processors***

**Intel® Core™ & Atom™ Processors**

**Intel® Processor Graphics**

**Intel® FPGA**

**Crest Family (Nervana ASIC)***

**Movidius Myriad (VPU)**

**Intel® GNA (IP)***

**Intel® GNA (IP)**
Deep learning
By design

Custom Hardware
- Unprecedented compute density

Blazing Data Access
- 32 GB of in package memory via HBM2 technology
- 8 Tera-bits/s of memory access speed

High-Speed Scalability
- 12 bi-directional high-bandwidth links
- Seamless data transfer via interconnects

Scalable acceleration with best performance for intensive deep learning training & inference, period

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit: http://www.intel.com/performance Source: Intel measured as of November 2016

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804
Outline

• Intel® Nervana™ Overview
• Intel’s Deep Learning Hardware
• **Intel’s Deep Learning Software**
  - MKL-DNN
  - Neon and Nervana Cloud
  - Nervana Graph
Intel® MKL and Intel® MKL-DNN for Deep Learning

Deep Learning Frameworks

- Caffe
- theano
- TensorFlow
- torch
- dmld
- mxnet

Intel® Math Kernel Library (Intel® MKL)

Intel® MKL-DNN

Xeon  Xeon Phi  FPGA

Intel® MKL

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel® MKL-DNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN primitives + wide variety of other math functions</td>
<td>DNN primitives</td>
</tr>
<tr>
<td>C DNN APIs (C++ future)</td>
<td>C/C++ DNN APIs</td>
</tr>
<tr>
<td>Binary distribution</td>
<td>Open source DNN code*</td>
</tr>
<tr>
<td>Free community license. Premium support available as part of Parallel Studio XE</td>
<td>Apache 2.0 license</td>
</tr>
<tr>
<td>Broad usage DNN primitives; not specific to individual frameworks</td>
<td>Multiple variants of DNN primitives as required for framework integrations</td>
</tr>
<tr>
<td>Quarterly update releases</td>
<td>Rapid development ahead of Intel MKL releases</td>
</tr>
</tbody>
</table>

https://github.com/01org/mkl-dnn

* GEMM matrix multiply building blocks are binary
Outline

• Intel® Nervana™ Overview
• Intel’s Deep Learning Hardware
• Intel’s Deep Learning Software
  ▪ MKL-DNN
  ▪ Neon and Nervana Cloud
  ▪ Nervana Graph
Neon Framework

![GitHub screenshot of Neon Framework repository](image)

Intel® Nervana™ reference deep learning framework committed to best performance on all hardware

http://neon.nervanasys.com/docs/latest

- **1,022 commits**
- **4 branches**
- **29 releases**
- **63 contributors**
- Apache-2.0 license

**Branch: master**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>MKL backend (#783)</td>
<td>2 months ago</td>
</tr>
<tr>
<td>doc</td>
<td>Update for v2.1.0 (#917)</td>
<td>12 days ago</td>
</tr>
<tr>
<td>examples</td>
<td>Adding &quot;numactl --interleave=all&quot; Performance Best Practice Notes (#919)</td>
<td>12 days ago</td>
</tr>
<tr>
<td>loader</td>
<td>Makefiles now check for virtualenv and pkg-config, (#872)</td>
<td>24 days ago</td>
</tr>
<tr>
<td>neon</td>
<td>add mkl backend for neon tests (#881)</td>
<td>13 days ago</td>
</tr>
<tr>
<td>tests</td>
<td>add mkl backend for neon tests (#881)</td>
<td>13 days ago</td>
</tr>
</tbody>
</table>
# Neon: Enterprise-class Deep Learning Library

<table>
<thead>
<tr>
<th>Library</th>
<th>Class</th>
<th>Time (ms)</th>
<th>forward (ms)</th>
<th>backward (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nervana-neon-fp16</td>
<td>ConvLayer</td>
<td>254</td>
<td>82</td>
<td>171</td>
</tr>
<tr>
<td>Nervana-neon-fp32</td>
<td>ConvLayer</td>
<td>320</td>
<td>103</td>
<td>217</td>
</tr>
<tr>
<td>CuDNN[R4]-fp16 (Torch)</td>
<td>cudnn.SpatialConvolution</td>
<td>471</td>
<td>140</td>
<td>331</td>
</tr>
<tr>
<td>CuDNN[R4]-fp32 (Torch)</td>
<td>cudnn.SpatialConvolution</td>
<td>529</td>
<td>162</td>
<td>366</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>conv2d</td>
<td>540</td>
<td>158</td>
<td>382</td>
</tr>
<tr>
<td>Chainer</td>
<td>Convolution2D</td>
<td>885</td>
<td>251</td>
<td>632</td>
</tr>
</tbody>
</table>

Third-party (Facebook) benchmarking
Why Python?

• Popular, well established, developer familiarity
• Fast to prototype
• Rich ecosystem of existing packages.
• Data Science: pandas, pycuda, ipython, matplotlib, h5py, ...
• Good “glue” language: scriptable plus functional and OO support, plays well with other languages
Neon Workflow

Dataset

Model/Layers

$C(y, t)$

Cost

Optimizer
# Neon Overview

<table>
<thead>
<tr>
<th>Backend</th>
<th>NervanaGPU, NervanaCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datasets</td>
<td>MNIST, CIFAR-10, Imagenet 1K, PASCAL VOC, Mini-Places2, IMDB, Penn Treebank, Shakespeare Text, bAbI, Hutter-prize, UCF101, flickr8k, flickr30k, COCO</td>
</tr>
<tr>
<td>Initializers</td>
<td>Constant, Uniform, Gaussian, Glorot Uniform, Xavier, Kaiming, IdentityInit, Orthonormal</td>
</tr>
<tr>
<td>Optimizers</td>
<td>Gradient Descent with Momentum, RMSProp, AdaDelta, Adam, Adagrad, MultiOptimizer</td>
</tr>
<tr>
<td>Activations</td>
<td>Rectified Linear, Softmax, Tanh, Logistic, Identity, ExpLin</td>
</tr>
<tr>
<td>Layers</td>
<td>Linear, Convolution, Pooling, Deconvolution, Dropout, Recurrent, Long Short-Term Memory, Gated Recurrent Unit, BatchNorm, LookupTable, Local Response Normalization, Bidirectional-RNN, Bidirectional-LSTM</td>
</tr>
<tr>
<td>Costs</td>
<td>Binary Cross Entropy, Multiclass Cross Entropy, Sum of Squares Error</td>
</tr>
<tr>
<td>Metrics</td>
<td>Misclassification (Top1, TopK), LogLoss, Accuracy, PrecisionRecall, ObjectDetection</td>
</tr>
</tbody>
</table>
Example: recognition of handwritten digits

MNIST dataset
70,000 images (28x28 pixels)
Goal: classify images into a digit 0-9

N = 28 x 28 pixels
= 784 input units

N = 100 hidden units
(user-defined parameter)

N = 10 output units
(one for each digit)

Each unit \(i\) encodes the probability of the input image of being of the digit \(i\)
MNIST

be = gen_backend(backend='cpu')

(X_train, y_train), (X_test, y_test), nclass = load_mnist()
train_set = ArrayIterator(X_train, y_train, nclass=nclass)
test_set = ArrayIterator(X_test, y_test, nclass=nclass)

init_norm = Gaussian(loc=0.0, scale=0.01)

layers = []
layers.append(Affine(nout=100, init=init_norm, activation=Rectlin()))
layers.append(Affine(nout=10, init=init_norm, activation=Softmax()))
mlp = Model(layers=layers)

cost = GeneralizedCost(costfunc=CrossEntropyMulti())

optimizer = GradientDescentMomentum(0.1, momentum_coef=0.9)
callbacks = Callbacks(mlp, eval_set=valid_set)

mlp.fit(train_set, optimizer=optimizer, num_epochs=10, cost=cost, callbacks=callbacks)
MNIST

```python
mlp.fit(train_set, optimizer=optimizer, num_epochs=10, cost=cost, callbacks=callbacks)
```

customizable callbacks
Neon: Example operations - command line arguments

```bash
./examples/mnist_mlp.yaml -e 10 -s "mymodel.pkl" -r 0 -i 4
```

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w, --data_dir</td>
<td>Path to data directory (default: nervana/data )</td>
</tr>
<tr>
<td>-e, --epochs</td>
<td>Number of epochs to run during training (default: 10 )</td>
</tr>
<tr>
<td>-s, --save_path</td>
<td>Path to save the model snapshots (default: None )</td>
</tr>
<tr>
<td>-o, --output_file</td>
<td>Path to save the metrics and callback data generated during training. Can be used by nvis for visualization (default: None )</td>
</tr>
<tr>
<td>-b, --backend {cpu,gpu}</td>
<td>Which backend to use (default: cpu )</td>
</tr>
<tr>
<td>-z, --batch_size</td>
<td>Batch size for training (default: 128 )</td>
</tr>
</tbody>
</table>

http://neon.nervanasys.com/index.html/
Neon: Example operations – Visualizing Results

```
./examples/mnist_mlp.yaml -o data.h5 nvis -i data.h5 -o viz/
```

http://neon.nervanasys.com/index.html/tools.html#visualization
Curated Models: Segnet

Badrinarayanan et al., 2015

<table>
<thead>
<tr>
<th>Number of GPUs</th>
<th>Iteration time per image</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66 ms</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>35 ms</td>
<td>1.9 x</td>
</tr>
<tr>
<td>4</td>
<td>18 ms</td>
<td>3.7 x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Neon (ms)</th>
<th>Caffe (ms)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>101</td>
<td>719</td>
<td>7.1x</td>
</tr>
<tr>
<td>Backward</td>
<td>164</td>
<td>746</td>
<td>4.5x</td>
</tr>
<tr>
<td>Total</td>
<td>265</td>
<td>1455</td>
<td>5.5x</td>
</tr>
</tbody>
</table>
Model Zoo

- [https://github.com/NervanaSystems/ModelZoo](https://github.com/NervanaSystems/ModelZoo)

- Pre-trained weights and models
Neon...

- Easy-to-use deep learning framework
- Populated model zoo with pre-trained models
- Integration within nervana cloud for easy deployment and experiment-tracking
- 10-20x speed-ups with forthcoming custom processor
Nervana Cloud

Frameworks:
Neon, Caffe2*, TensorFlow*, MXNet*, Torch*
*Multiple framework support coming soon.

2-3x speedup on Titan X GPUs

http://docs.cloud.nervanasys.com/index.html/
Nervana Cloud: Two Powerful Interfaces

Nervana ncloud CLI

Nervana Cloud Web UX

http://docs.cloud.nervanasys.com/index.html/
Nervana Cloud

CLI

Web UI

https

web services

scheduler

gpu VM

neon

Storage

DB
Nervana Cloud: Interactive Mode

- Interactive Python environment within container
- Access Nervana Cloud GPUs & software
- Attach data sets and code repository
- Extend neon with custom callbacks (e.g., BokehJS)
- Get root access to virtual environment
- Install packages, change system settings
Outline

• Intel® Nervana™ Overview
• Intel’s Deep Learning Hardware

• **Intel’s Deep Learning Software**
  ▪ MKL-DNN
  ▪ Neon and Nervana Cloud
  ▪ **Nervana Graph**
The Nervana Graph Project

• An intermediate representation (IR) for deep learning

• A compiler for Nervana graph IR

• Frontends for other frameworks to Nervana graph

• Many-to-many problem

• A reference frontend (neon)

https://github.com/NervanaSystems/ngraph
Learn More... intelnervana.com