Intel® AI Analytics Toolkit
Deep Learning Optimizations

Yuning Qiu, AI Software Solutions Engineer
Aaryan Kothapalli, AI Software Solutions Engineer
Deep Learning Framework (Optimizations by Intel)

SCALING
- Improve load balancing
- Reduce synchronization events, all-to-all comms

UTILIZE ALL THE CORES
- OpenMP, MPI
- Reduce synchronization events, serial code
- Improve load balancing

VECTORIZE / SIMD
- Unit strided access per SIMD lane
- High vector efficiency
- Data alignment

EFFICIENT MEMORY / CACHE USE
- Blocking
- Data reuse
- Prefetching
- Memory allocation

Graph optimizations
- fuse
- Batch normalization
- Memory allocation

More framework optimizations underway
(e.g., PaddlePaddle*, CNTK* & more)

See installation guides at
ai.intel.com/framework-optimizations/

SEE ALSO: Machine Learning Libraries for Python (Scikit-learn, Pandas, NumPy), R (Cart, randomForest, e1071), Distributed (MLib on Spark, Mahout)
*Limited availability today
Other names and brands may be claimed as the property of others.
Optimization Notice
Agenda

- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Speedup DL inference via Intel® Neural Compressor
- Exercises
Intel Optimization for TensorFlow
What is TensorFlow?

An end-to-end open source machine learning platform

TensorFlow

The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.

Get started with TensorFlow
Intel Contributions in TensorFlow

4+ years of close collaboration between Intel and Google

- oneDNN library integrated (2017)
- Quantization Introduced (2018)
- Data type optimizations (2019)
- AI accelerator support (Intel® AMX) (2020)
- Co-architected pluggable device for new AI devices (2021)
- TensorFlow 1.0 released (2022)

2017 2018 2019 2020 2021 2022
# oneAPI Deep Neural Network Library (oneDNN)

## Features
- Supports FP32, FP16, Bfloat16, and int8.
- Leverages Intel® DL Boost, AVX512 instructions and processor capabilities
- Fused operations for optimized performance

## Support Matrix
- Compilers: Intel® oneAPI DPC++ / C++ Compilers
- OS: Linux, Windows, macOS
- CPU: Intel Atom, Intel® Core™, Intel® Xeon®, Intel® Xeon® Scalable processors
- GPU: Intel® Processor Graphics Gen9, Intel® Processor Graphics Gen12

## Category
<table>
<thead>
<tr>
<th>Category</th>
<th>Functions</th>
</tr>
</thead>
</table>
| Compute intensive operations                 | - (De-)Convolution
- Inner Product
- RNN (Vanilla, LSTM, GRU)
- GEMM |
| Memory bandwidth limited operations          | - Pooling
- Batch Normalization
- Local Response Normalization
- Layer Normalization
- Elementwise
- Binary elementwise
- Softmax
- Sum
- Concat
- Shuffle |
| Data manipulation                            | - Reorder                                                                 |

Software and Advanced Technologies Group | SATG/AIA
oneDNN Integration with TensorFlow

- Replaces compute-intensive standard TF ops with highly optimized custom oneDNN ops
- Aggressive op fusions to improve performance of Convolutions and Matrix Multiplications
- Primitive caching to reduce overhead of calling oneDNN
oneDNN Optimizations in TensorFlow

- Turn on oneDNN optimizations at runtime in official TensorFlow distributions by setting an environment variable `TF_ENABLE_ONEDNN_OPTS=1`.
- Up to 3X performance improvement
- Supported in TensorFlow-based packages as well (TensorFlow serving, TensorFlow Extended - TFX)

* https://github.com/tensorflow/community/pull/400

Intel Optimization for PyTorch
What is Intel Optimization for PyTorch?

- Intel regularly upstreams most of the CPU optimizations to standard PyTorch
- Intel releases additional features and performance improvements through the Intel Extension for PyTorch
  - As the extension presents more aggressive optimizations, it offers bigger speed-up for training and inference
Intel® Extension for PyTorch* (IPEX)

- Buffer the PRs for stock Pytorch
- Provide users with the up-to-date Intel software/hardware features
- Streamline the work to integrate oneDNN
- Unify user experiences on Intel CPU and GPU

- **Operator Optimization**
  - Customized operators
  - Auto graph optimization
- **Mix Precision**
  - Accelerate PyTorch operator by LP
  - Simplify the data type conversion
- **Optimal Optimizer**
  - Split Optimizer (e.g., split-sgd)
  - Fused Optimizer
Ease-of-Use User-Facing API (v1.10.x~)

**fp32**

```python
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

# code changes
import intel_extension_for_pytorch as irepid
model = irepid.optimize(model)

with torch.no_grad():
    model(data)
```

**bfloat16**

```python
import torch
import torchvision.models as models

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

model = model.to(memory_format=torch.channels_last)
data = data.to(memory_format=torch.channels_last)

# code changes
import intel_extension_for_pytorch as irepid
model = irepid.optimize(model, dtype=torch.bfloat16)

with torch.no_grad():
    with torch.cpu.amp.autocast():
        model(data)
```
Intel Extension for PyTorch benchmark for fp32

Speed-up compared to stock PyTorch for Float32

Model architecture

 throughput Inference    Realtime Inference
Intel Extension for PyTorch benchmark for BFloat16

Speed-up compared to stock PyTorch for BFloat16

Throughput Inference
Realtime Inference

BERT-Large
Bert-Base

MODEL ARCHITECTURE
Use case with Intel-optimized PyTorch

- Mycobacterium Tuberculosis Detection with Intel-optimized PyTorch

![Detectron2 Throughput Performance](chart)

- Baseline PyTorch 1.4 with 1 instance: 1.0
- Intel optimized PyTorch 1.6 with 1 instance: 2.17
- Intel optimized PyTorch 1.6 with 24 instances: 11.4

(Higher is Better)
Speedup DL inference via Intel Neural Compressor
Low-Precision (8-bit Integer) Inference Optimization

- Quantized models using 8-bit integers gaining adoption
  - Improved performance
  - Trade off accuracy for performance
- Additional post-training quantization step needed
- Intel Neural Compressor
  - Automatically quantizes pre-trained model
  - Picks quantization scheme to meet specific performance and accuracy needs

---

**Quantization**

```
FP32  FP32  FP32  INT8  INT8  INT8  INT8
```

**Graphic Optimization**

```
Conv2D
BatchNorm
Relu
Conv2D BatchNorm Relu
```
Performance

Stock TF on ICX (FP32)  oneDNN Enabled (FP32)  INC Quantization (INT8)

Baseline  1.5x  3.9x

13.06  20.54  81.66

6.25x Total Perf Gain

Customer/LZ model: SSD-ResNet-34 (BS=1)

Model Accuracy & Performance

Performance Speedup (Higher is Better)

Accuracy Loss (Lower is Better)

-1.00%  0.00%  1.00%

4.00x  3.50x  3.00x  2.50x  2.00x  1.50x  1.00x

-1.00%  0.00%  1.00%

OOB Random Models (w/ VNNI example)

2.2x geomean and up to 4x performance
Architecture

- Technology
  - Quantization
  - Pruning
  - Knowledge distillation
  - Graphic Optimization
  - Mix Precision

- Platform: Intel CPU and GPU (ongoing)

- Framework
  - TensorFlow*, including 1.15.0 UP3, 2.5.0, 2.7.0, Official TensorFlow 2.6.2, Official TensorFlow 2.7.0
  - PyTorch*, including 1.8.0+cpu, 1.9.0+cpu, 1.10.0+cpu
  - Apache* MXNet, including 1.6.0, 1.7.0, 1.8.0
  - ONNX* Runtime, including 1.7.0, 1.8.0, 1.9.0
  - Execution Engine, a reference bare metal solution(./engine) for domain-specific NLP models.

---

Model | TensorFlow | PyTorch | ONNX | MXNet
--- | --- | --- | --- | ---

<table>
<thead>
<tr>
<th>User-Facing APIs</th>
<th>Quantization, Pruning, Knowledge Distillation, Graph Optimization, ...</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Compressions</th>
<th>Quantization, Pruning, Knowledge Distillation, Graph Optimization, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post training static quantization</td>
<td>Pruning</td>
</tr>
<tr>
<td>Post training dynamic quantization</td>
<td>magnitude pruning</td>
</tr>
<tr>
<td>Quantization-aware training</td>
<td>Gradient sensitivity pruning</td>
</tr>
<tr>
<td>Knowledge Distillation</td>
<td>Knowledge Distillation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auto Tuning</th>
<th>Tuning Strategies</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Backends</th>
<th>TensorFlow, PyTorch, ONNX Runtime, MXNet, Engine</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Hardware platforms</th>
<th>Intel CPU</th>
<th>Intel GPU</th>
</tr>
</thead>
</table>

Adoption

“happy to see out of the box performance by quantizing NLP models on PyTorch using LPOT/INC”

“deliver in an easy and accessible way through Optimum, as we did with Intel and Neural Compressor”

“support first INT8 model quantized by Intel Neural Compressor in ONNX model zoo”
Exercises

- Intel Optimization for PyTorch
- Intel Optimization for TensorFlow
- Intel Neural Compressor
Notices and Disclaimers

• Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

• Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

• Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

• Your costs and results may vary.

• Intel technologies may require enabled hardware, software or service activation.

• © Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Workloads and Configurations

See all benchmarks and configurations: https://software.intel.com/content/www/us/en/develop/articles/blazing-fast-python-data-science-ai-performance.html. Each performance claim and configuration data is available in the body of the article listed under sections 1, 2, 3, 4, and 5. Please also visit this page for more details on all scores, and measurements derived.

Testing Date: Performance results are based on testing by Intel® as of October 16, 2020 and may not reflect all publicly available updates. Configurations details and Workload Setup: 2 x Intel® Xeon® Platinum 8280 @ 28 cores, OS: Ubuntu 19.10.5.3.0-64-generic Mitigated 384GB RAM (192 GB RAM (12x 32GB 2933). SW: Modin 0.8.1. Scikit-learn 0.22.2. Pandas 1.01, Python 3.8.5, DAL (DAAL4Py) 2020.2, Census Data, (21721922.45) Dataset is from IPUMS USA, University of Minnesota, www.ipums.org [Steven Ruggles, Sarah Flood, Ronald Goeken, Josiah Grover, Erin Meyer, Jose Pacas and Matthew Sobek. IPUMS USA: Version 10.0 [dataset], Minneapolis, MN. IPUMS, 2020. https//doc.org/10.18128/D010.V10.0]

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. Configuration Details and Workload Setup: Intel® oneAPI Data Analytics Library 2021.1 (oneDAL). Scikit-learn 0.23.1, Intel® Distribution for Python 3.8; Intel® Xeon® Platinum 8280L CPU @ 270GHz, 2 sockets, 28 cores per socket, 10M samples, 10 features, 100 clusters, 100 iterations, float32.

Testing Date: Performance results are based on testing by Intel® as of October 23, 2020 and may not reflect all publicly available updates. Configuration Details and Workload Setup: Intel® oneAPI Data Analytics Toolkit v2021.1; Intel® oneAPI Data Analytics Library (oneDAL) beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7, Intel® Xeon® Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x4003003, total available memory 376 GB, 12X32GB modules, DDR4. AMD Configuration: AMD Rome 7742 @2.25 GHz, 2 sockets, 64 cores per socket, microcode: 0x8301038, total available memory 512 GB, 16X32GB modules, DDR4, oneDAL beta10, Scikit-learn 0.23.1, Intel® Distribution for Python 3.7. NVIDIA Configuration: NVIDIA Tesla V100 – 16 Gb, total available memory 376 GB, 12X32GB modules, DDR4, Intel® Xeon Platinum 8280 CPU @ 2.70GHz, 2 sockets, 28 cores per socket, microcode: 0x5003003, cuDF 0.15, cuML 0.15, CUDA 10.2.89, driver 440.33.01, Operation System: CentOS Linux 7 (Core), Linux 4.19.36 kernel.

Testing Date: Performance results are based on testing by Intel® as of October 13, 2020 and may not reflect all publicly available updates. Configurations details and Workload Setup: CPU: c5.18xlarge AWS Instance (2 x Intel® Xeon® Platinum 8124M @ 18 cores. OS: Ubuntu 20.04.2 LTS, 193 GB RAM. GPU: p3.2xlarge AWS Instance (GPU: NVIDIA Tesla V100 16GB, 8 vCPUs, OS: Ubuntu 18.04.2 LTS, 61 GB RAM. SW: XGBoost 1.1: build from sources compiler – G++ 7.4, nvcc 9.1 Intel® DAAL: 2019.4 version: Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25 Scikit-learn 0.21.2.
Workloads and Configurations

Testing Date: Performance results are based on testing by Intel® as of October 26, 2020 and may not reflect all publicly available updates. **Configuration Details and Workload Setup:** Intel® Optimization for Tensorflow v2.2.0; oneDNN v1.2.0; Intel® Low Precision Optimization Tool v1.0; Platform; Intel® Xeon® Platinum 8280 CPU; #Nodes 1; #Sockets: 2; Cores/socket: 28; Threads/socket: 56; HT: On; Turbo: On; BIOS version:SE5C620.86B.02.01.0010.010620200716; System DDR Mem Config: 12 slots/16GB/2933; OS: CentOS Linux 7.8; Kernel: 4.4.240-1.el7.elrepo x86_64.

Configuration Details

NYCTaxi Workload performance:
For 20 million rows: Dual socket Intel(R) Xeon(R) Platinum 8280L CPUs (S2600WFT platform), 28 cores per socket, hyperthreading enabled, turbo mode enabled, NUMA nodes per socket=2, BIOS: SE5C620.86B.02.01.0013.121520200651, kernel: 5.4.0-65-generic, microcode: 0x4003003, OS: Ubuntu 20.04.1 LTS, CPU governor: performance, transparent huge pages: enabled, System DDR Mem Config: slots / cap / speed: 12 slots / 32GB / 2933MHz, total memory per node: 384 GB DDR RAM, boot drive: INTEL SSDSC2BB800G7. For 1 billion rows: Dual socket Intel Xeon Platinum 8260M CPU, 24 cores per socket, 2.40GHz base frequency, DRAM memory: 384 GB 12x32GB DDR4 Samsung @ 2666 MT/s 1.2V, Optane memory: 3TB 12x256GB Intel Optane @ 2666MT/s, kernel: 4.15.0-91-generic, OS: Ubuntu 20.04.4

End-to-End Census Workload performance (Stock):

End-to-End Census Workload performance (Optimized):