4TH CERN OPENLAB/INTEL HANDS-ON WORKSHOP ON CODE OPTIMISATION

INTRODUCTION TO TENSORFLOW WITH INTEL® OPTIMIZATIONS

Geneva, November 2018

LEGAL DISCLAIMERS

- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to: Learn About Intel® Processor Numbers http://www.intel.com/products/processor_number
- Some results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect 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.
- Intel does not control or audit the design or implementation of third party benchmarks or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmarks are reported and confirm whether the referenced benchmarks are accurate and reflect performance of systems available for purchase.
- Relative performance is calculated by assigning a baseline value of 1.0 to one benchmark result, and then dividing the actual benchmark result for the baseline platform into each of the specific benchmark results of each of the other platforms, and assigning them a relative performance number that correlates with the performance improvements reported.
- SPEC, SPECint, SPECfp, SPECrate, SPECpower, SPECjbb, SPECompG, SPEC MPI, and SPECjEnterprise* are trademarks of the Standard Performance Evaluation Corporation. See http://www.spec.org for more information.
- TPC Benchmark, TPC-C, TPC-H, and TPC-E are trademarks of the Transaction Processing Council. See http://www.tpc.org for more information.
- No computer system can provide absolute reliability, availability or serviceability. Requires an Intel® Xeon® processor E7-8800/4800/2800 v2 product families or Intel® Itanium® 9500 series-based system (or follow-on generations of either.) Built-in reliability features available on select Intel® processors may require additional software, hardware, services and/or an internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.
  For systems also featuring Resilient System Technologies: No computer system can provide absolute reliability, availability or serviceability. Requires an Intel® Run Sure Technology-enabled system, including an enabled Intel processor and enabled technology(ies). Built-in reliability features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.
  For systems also featuring Resilient Memory Technologies: No computer system can provide absolute reliability, availability or serviceability. Requires an Intel® Run Sure Technology-enabled system, including an enabled Intel® processor and enabled technology(ies). built-in reliability features available on select Intel® processors may require additional software, hardware, services and/or an Internet connection. Results may vary depending upon configuration. Consult your system manufacturer for more details.

AGENDA

- Introduction to TensorFlow
- Neural Networks with TensorFlow
- Convolutional Neural Networks (image classification)
- Optimizations and performance comparisons
- Install or build Intel® optimized TensorFlow

TENSORFLOW HISTORY

- **2nd gen. open source ML framework from Google***
  - Widely used by Google's: search, Gmail, photos, translate, etc.
  - Open source implementation released in November 2015
- **Core in C++, frontend wrapper is in Python**
  - Core: key computational kernel, extensible per user-ops
  - Python script to specify/drive computation
- **Runtime**
  - Multi-node originally per GRPC protocol, MPI added later
  - Own threading runtime (not OpenMP, TBB, etc.)

Milestones

02'16: TensorFlow Serving

02'16: TensorFlow Serving

01'17: Accelerated Linear Algebra (XLA)

02'17: TensorFlow Fold

INTEL AI FRAMEWORKS

Popular DL Frameworks are now optimized for CPU!

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

More under optimization: Caffe2* PYTORCH* Microsoft CNTK* PaddlePaddle* *and others to be enabled via Intel® nGraph™ Library*

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

WHY DO WE NEED OPTIMIZATIONS FOR CPU?

TensorFlow* on CPU has been very slow

With optimization: up to 14x Speedup in Training and 3.2x Speedup in Inference. Up-streamed and Ready to Use.

MAIN TENSORFLOW API CLASSES

Graph

Container for operations and tensors

Operation

Nodes in the graph

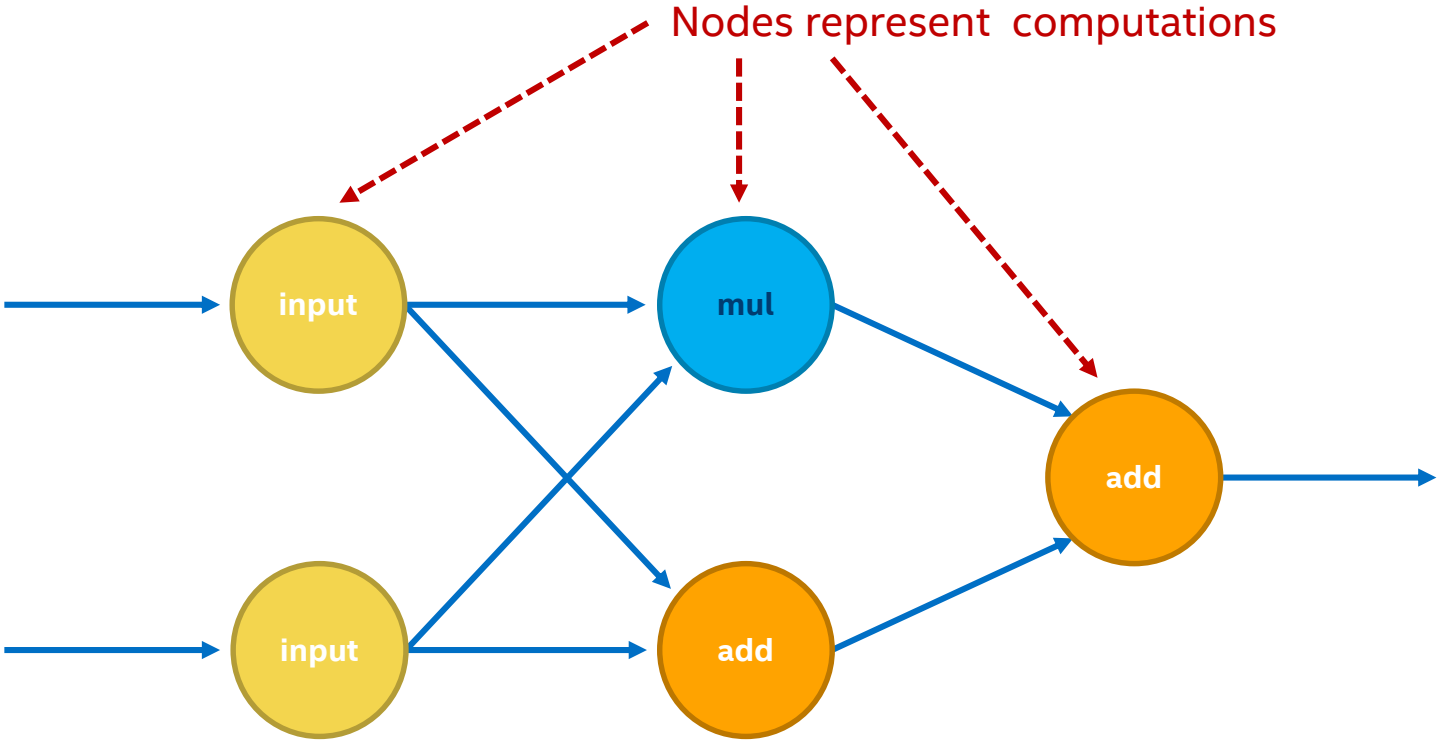
Represent computations

Tensor

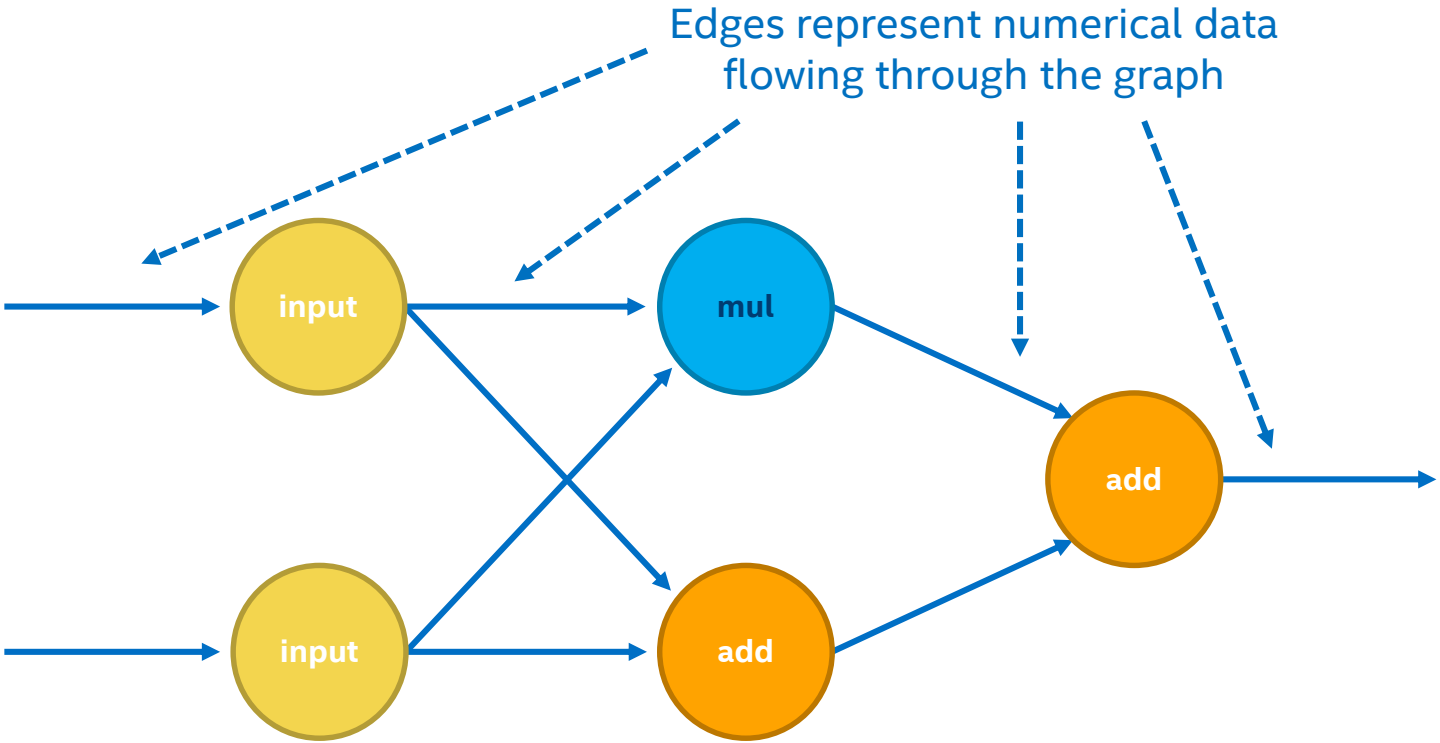
Edges in the graph

Represent data

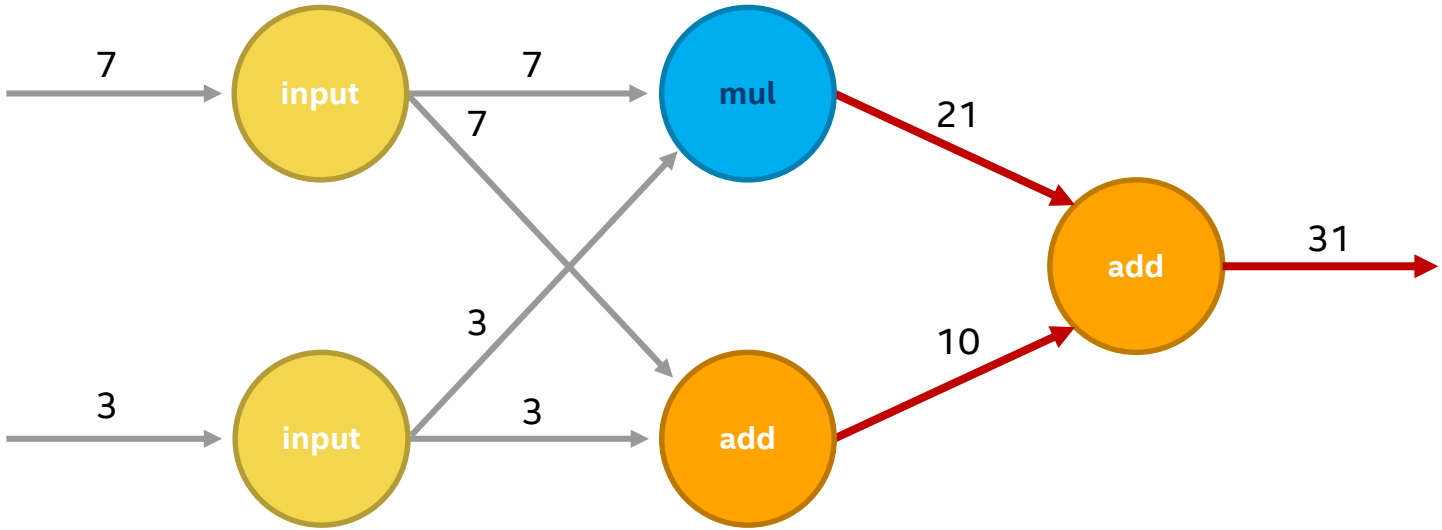
COMPUTATION GRAPH



COMPUTATION GRAPH

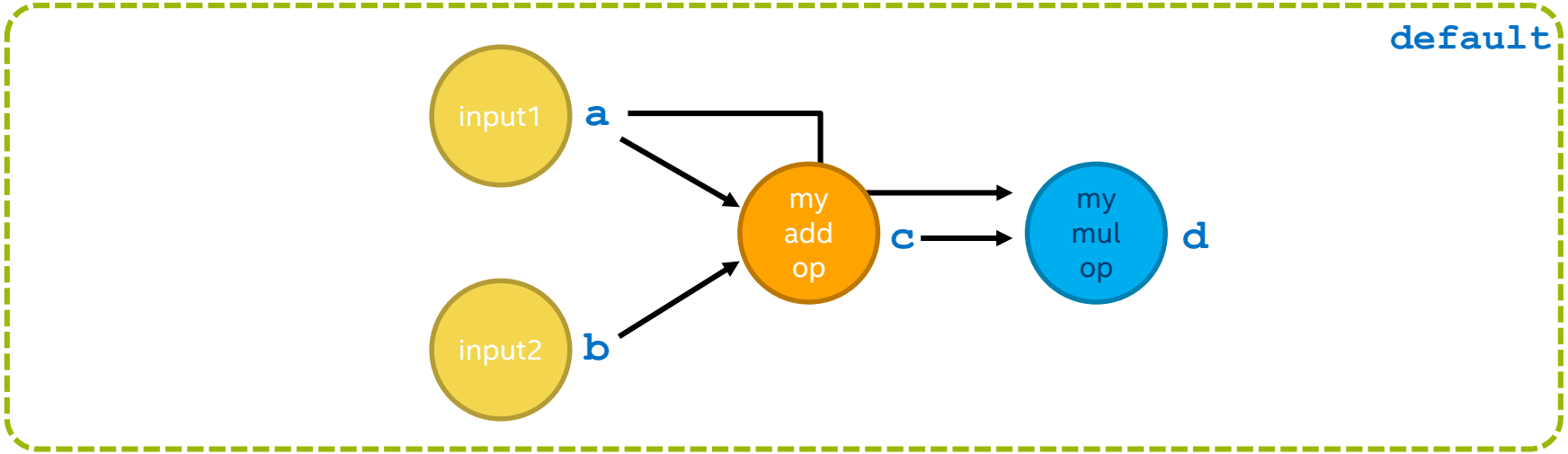


DATA FLOW



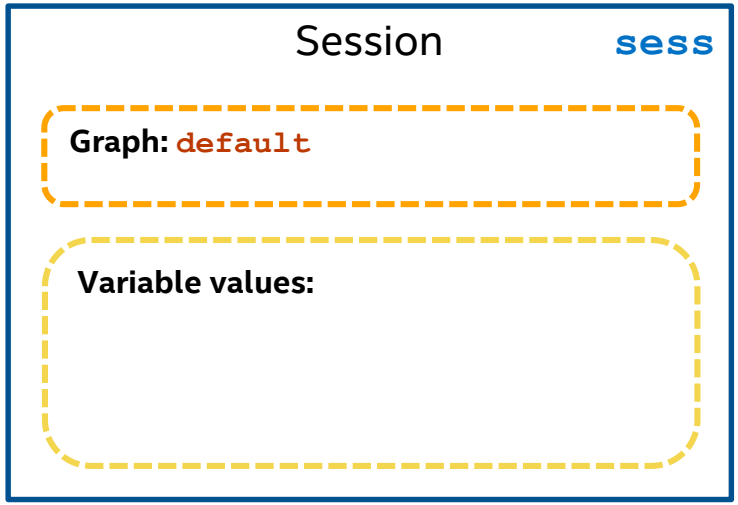
`tf.constant()` creates an `Operation` that returns a fixed value
`tf.placeholder()` defines explicit input that vary run-to-run

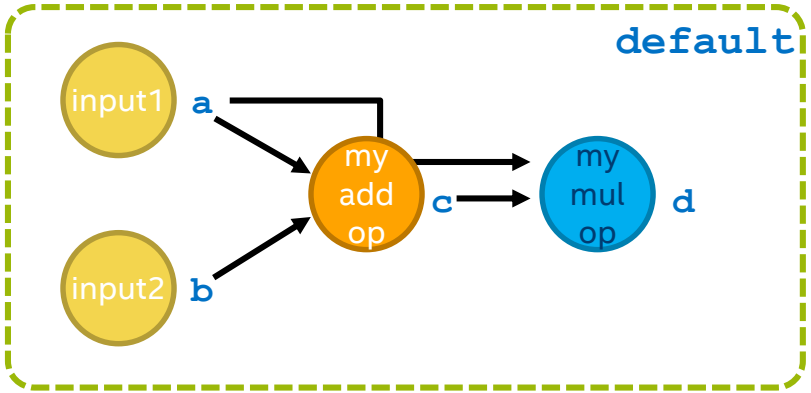
```
>>> a = tf.placeholder(tf.float32, name="input1")
>>> c = tf.add(a, b, name="my_add_op")
```



We use a `Session` object to execute graphs.
Each `Session` is dedicated to a single graph.

```
>>> sess = tf.Session()
```

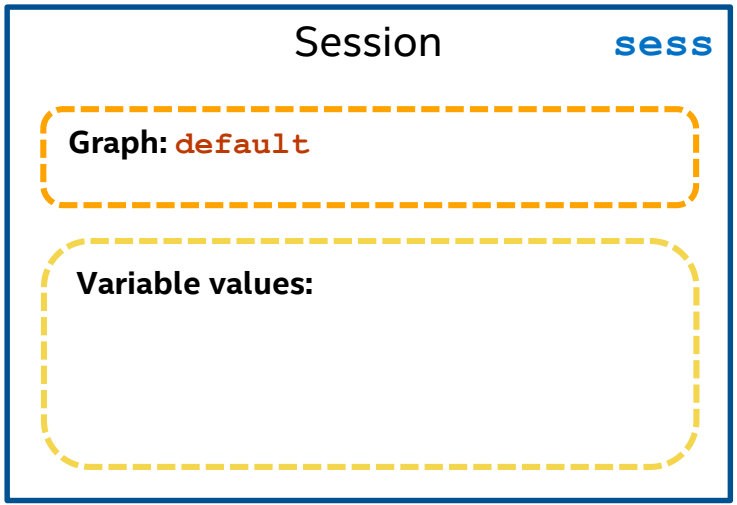


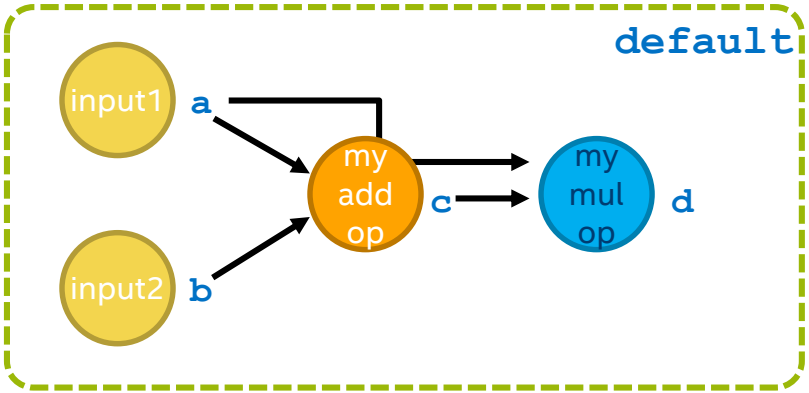


`ConfigProto` is used to set configurations of the `Session` object.

```
>>> config = tf.ConfigProto(
            inter_op_parallelism_threads=2,
            intra_op_parallelism_threads=24)

>>> tf.Session(config=config)
```

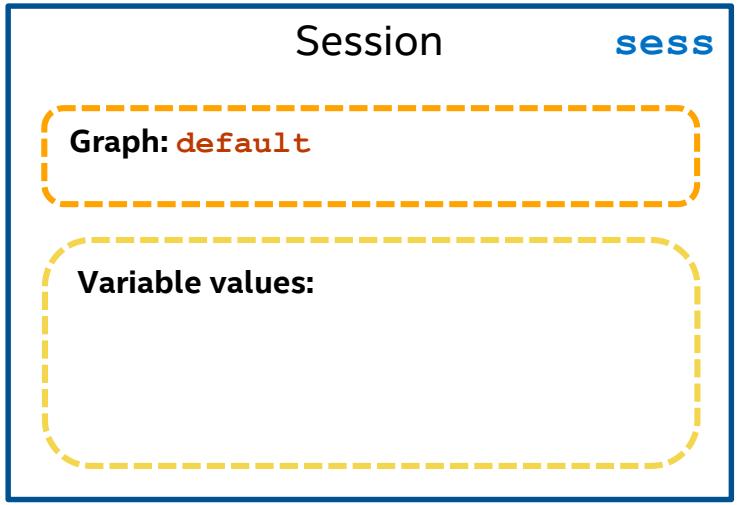


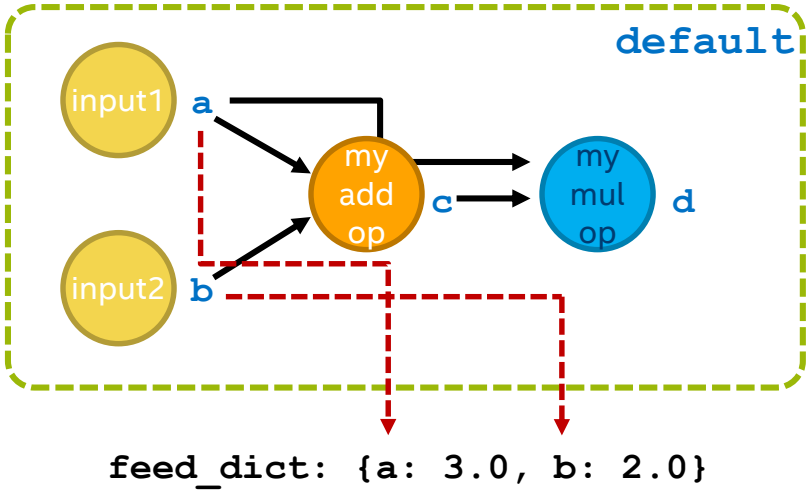


`placeholders` require data to fill them in when the graph is run

We do this by creating a dictionary mapping `Tensor`-keys to numeric values

```
>>> feed_dict = {a: 3.0, b: 2.0}
```

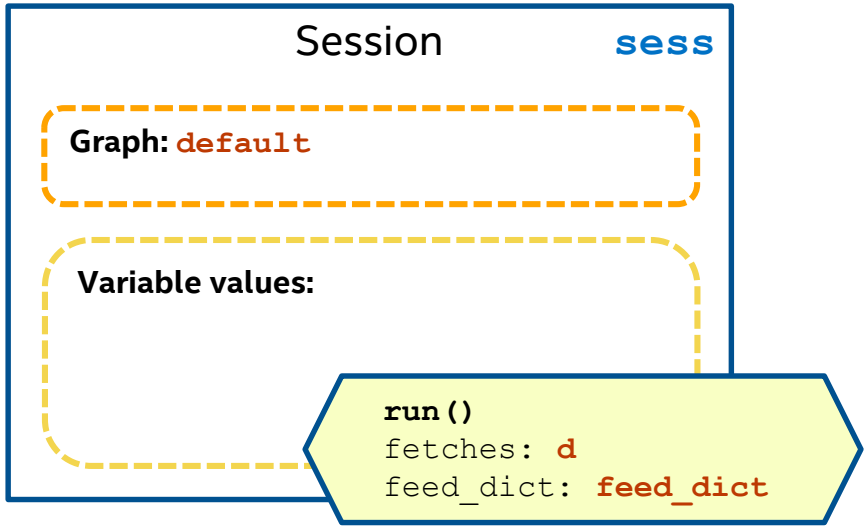


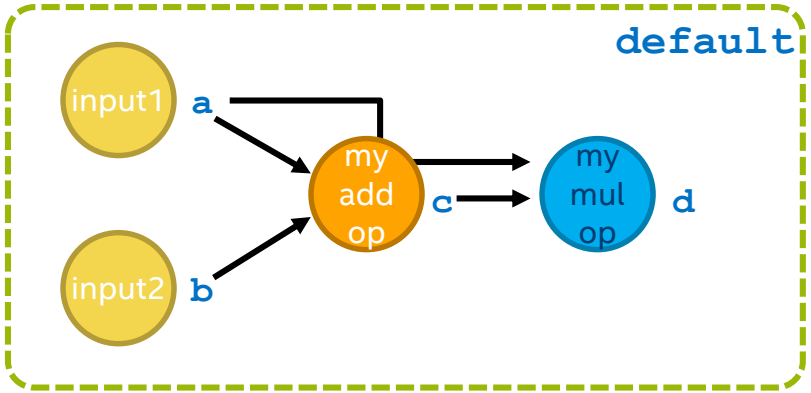


We execute the graph with `sess.run(fetches, feed_dict)`

`sess.run` returns the fetched values as a NumPy array

```
>>> out = sess.run(d, feed_dict=feed_dict)
```

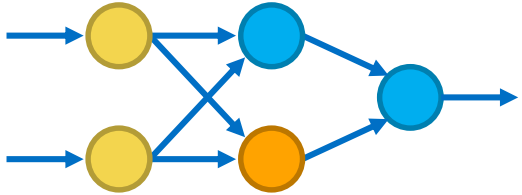




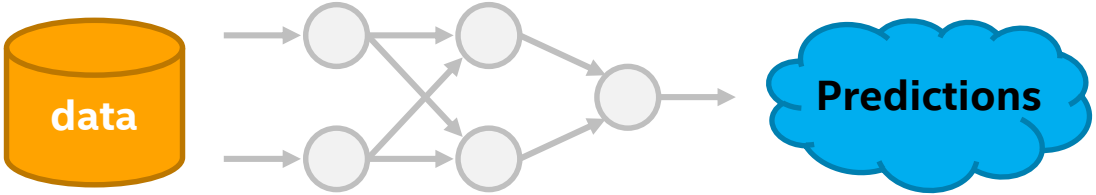
`feed_dict: {a: 3.0, b: 2.0}`

TWO-STEP PROGRAMMING PATTERN

1. Define a computation graph ("topology")



2. Run the graph



NEURAL NETWORKS WITH TENSORFLOW

NEURAL NETWORKS

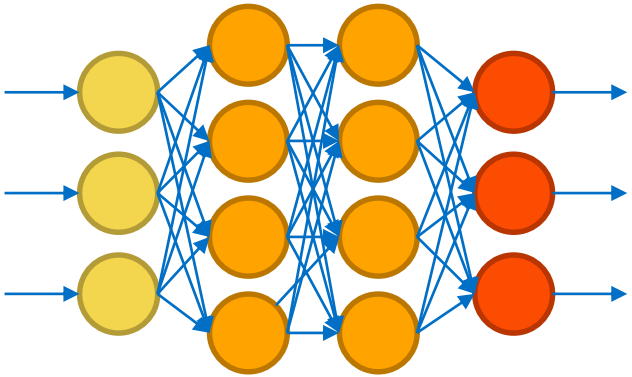
Use biology as inspiration for math model

Neurons:

Get signals from previous neurons

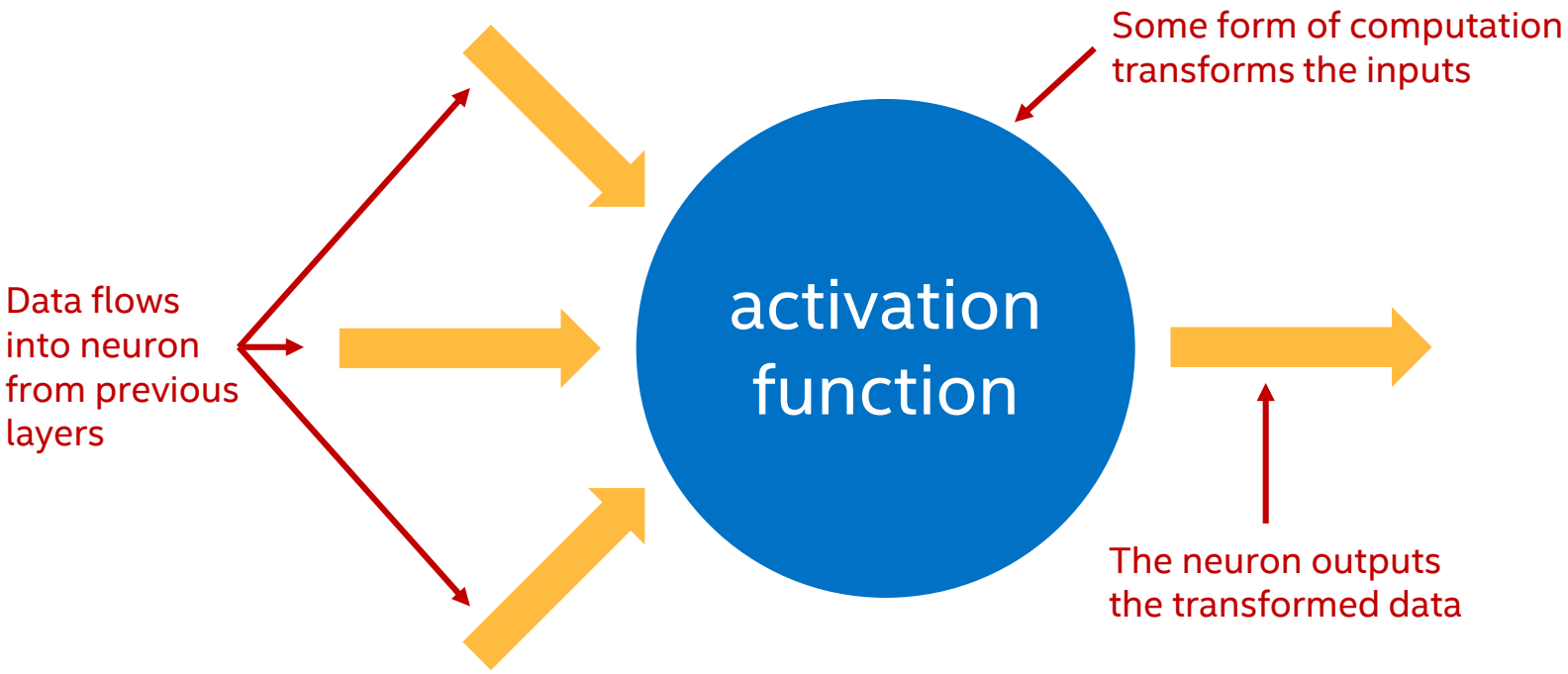
Generate signal (or not) according to inputs

Pass that signal on to future neurons



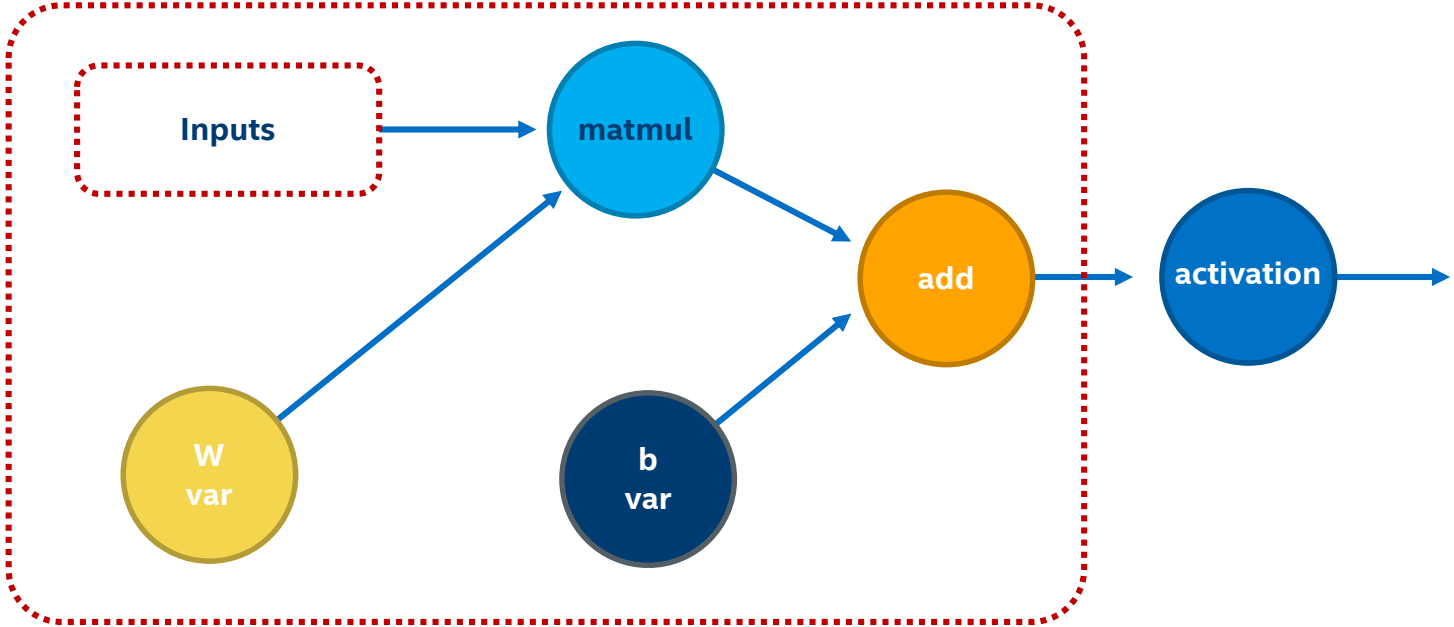
By layering many neurons, can create complex model

READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH

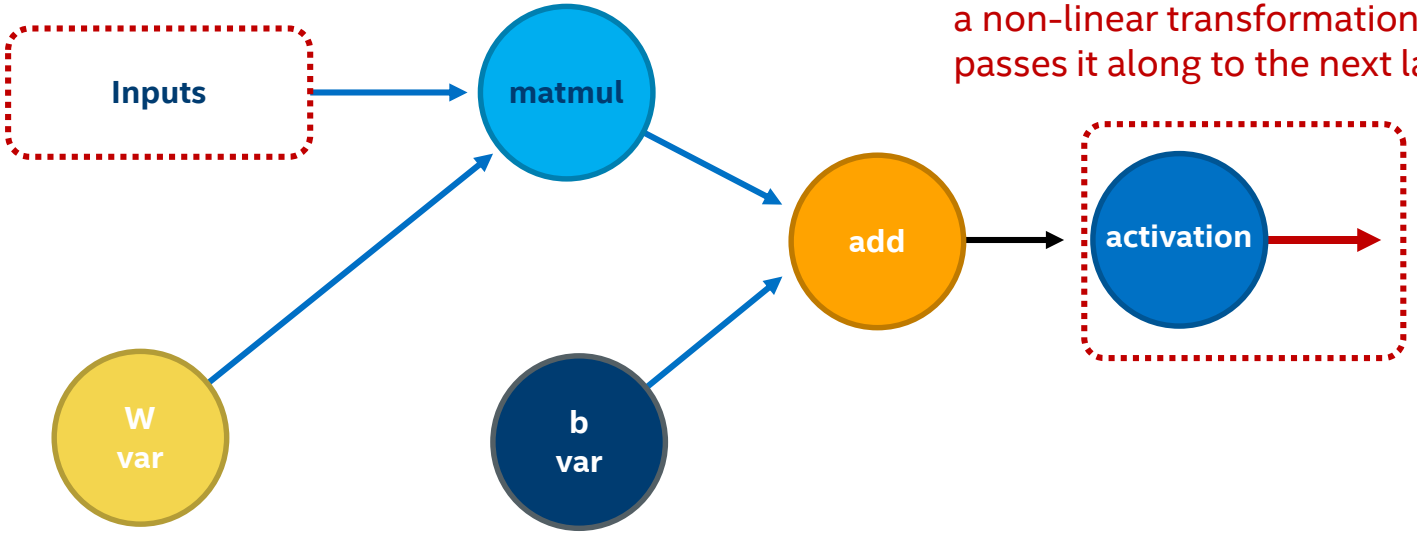


INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)

Represents the function $z = W^t X + b$

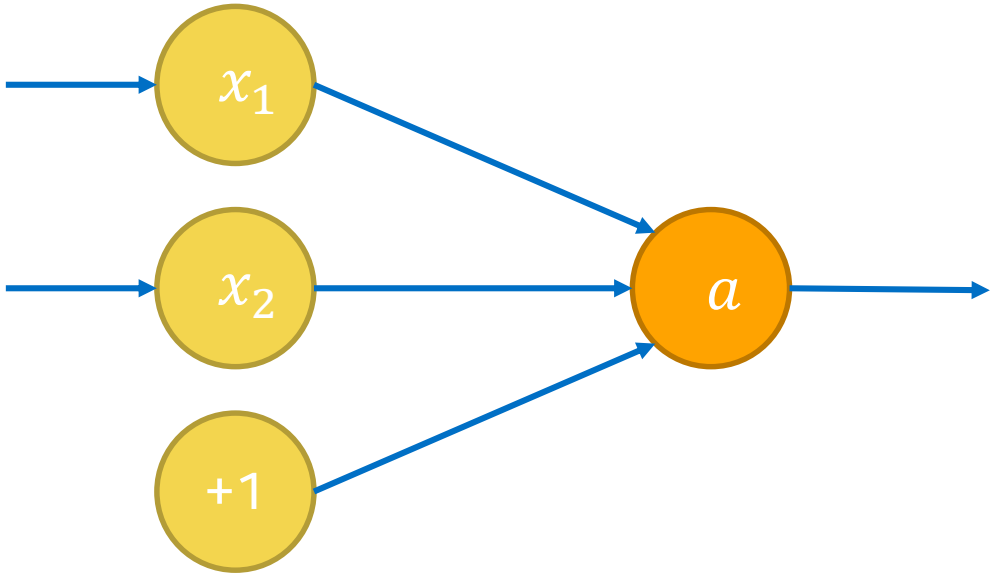


INSIDE A SINGLE NEURON (TENSORFLOW GRAPH)



The activation function applies a non-linear transformation and passes it along to the next layer

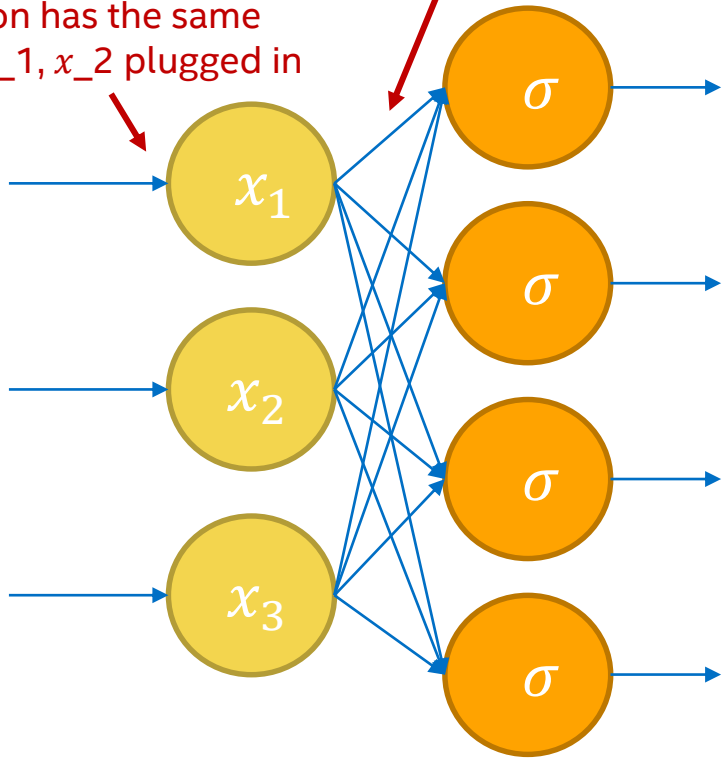
To keep visual noise down, we'll use this notation for now



A SINGLE NEURAL LAYER

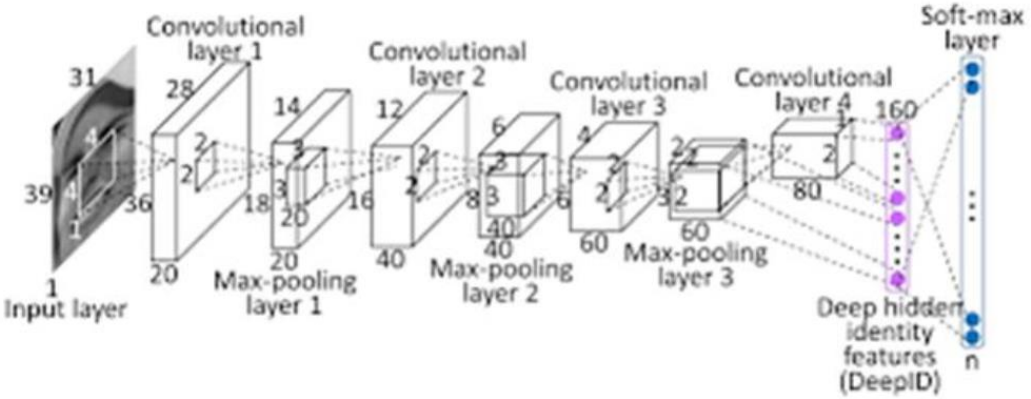
But having different weights means neurons respond to inputs differently

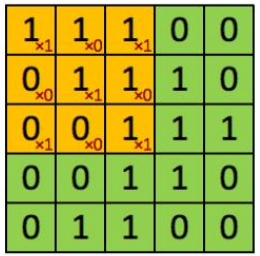
Each neuron has the same value for x_1, x_2 plugged in



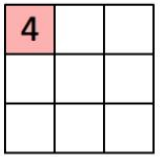
CONVOLUTIONAL NEURAL NETWORK WITH TENSORFLOW

CONVOLUTIONAL NEURAL NETS



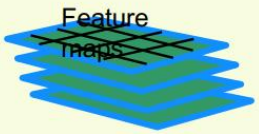


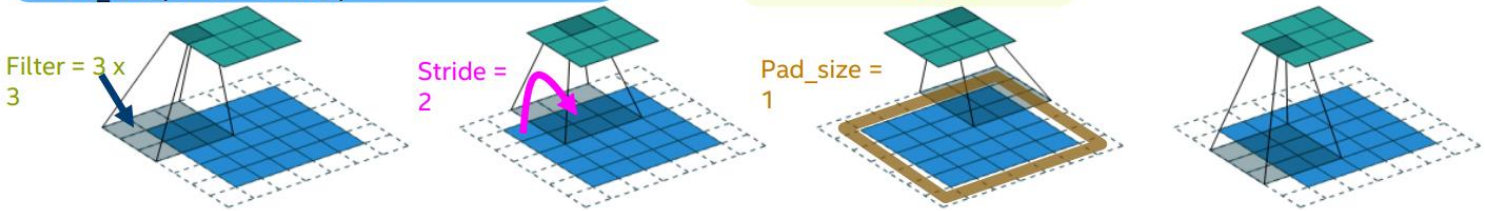
Image



Convolved Feature

Convolution Parameters:
Number of outputs/feature-maps: < 4 >
Filter size: < 3 x 3 >
Stride: < 2 >
Pad_size (for corner case): <1>





CONVOLUTION IN TENSORFLOW

`tf.nn.conv2d(input, filter, strides, padding)`

`input`: 4d tensor [batch_size, height, width, channels]

`filter`: 4d: [height, width, channels_in, channels_out]

Generally a Variable

`strides`: 4d: [1, vert_stride, horiz_strid, 1]

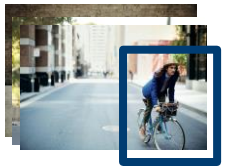
First and last dimensions must be 1 (helps with under-the-hood math)

`padding`: string: 'SAME' or 'VALID'

TRAINING AND INFERENCE

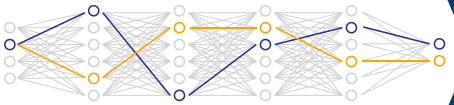
Step 1: Training
(Over Hours/Days/Weeks)

Input data



Person

Create Deep network



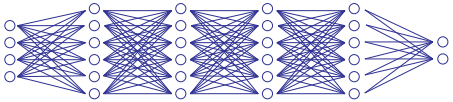
Trained Model

Output Classification

90% person
8% traffic light

Step 2: Inference
(Real Time)

New input from camera and sensors



Trained neural network model

97% person

Output Classification

CNN'S: TRAINING VS. INFERENCE

Training (N epochs)

1. Convolution (forward)
2. Convolution (backward)
3. Weight Update

Inference

1. Convolution (forward)

Training vs. Inference

$T_{train} \approx 3 * N * T_{infer}$

Assumption ("rule of thumb")

$T_{fwd} + T_{bwd} + T_{upd} \approx 3 * T_{fwd}$

Same type-width for training and inference
e.g., single-precision.

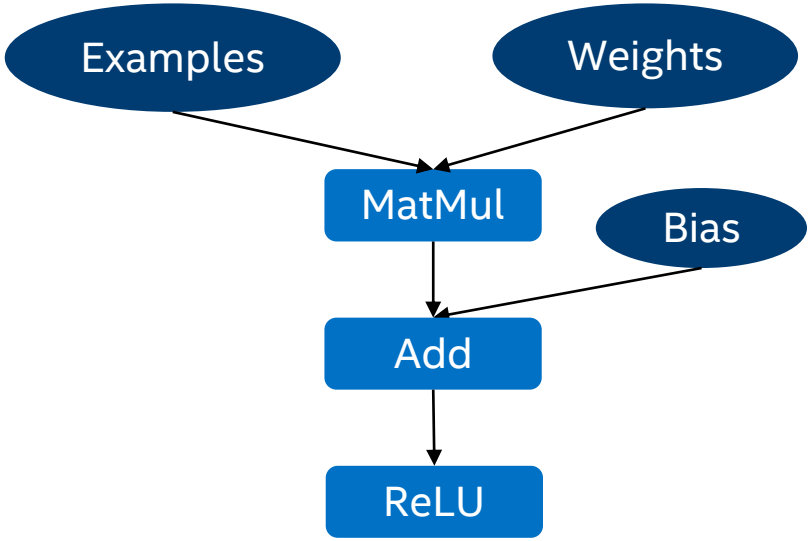
INTEL® TENSORFLOW OPTIMIZATIONS

INTEL-TENSORFLOW OPTIMIZATIONS

1. Operator optimizations
2. Graph optimizations
3. System optimizations

OPERATOR OPTIMIZATIONS

In TensorFlow, computation graph is a data-flow graph.



OPERATOR OPTIMIZATIONS

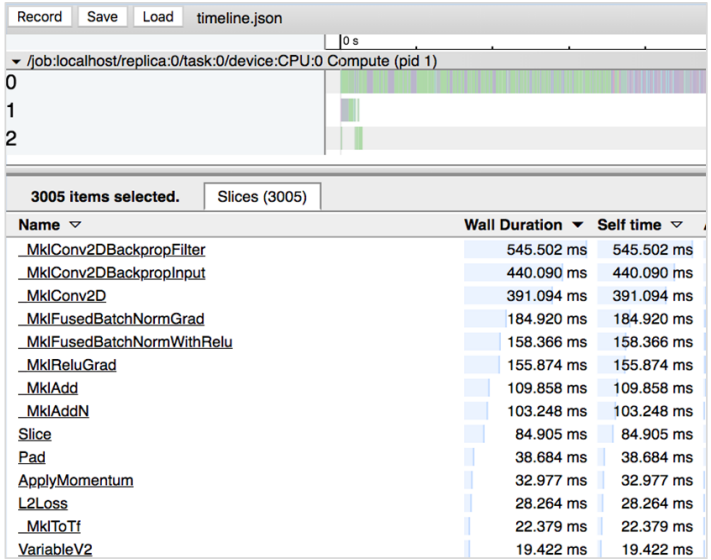
Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

Intel® MKL-DNN has optimized a set of TensorFlow operations.

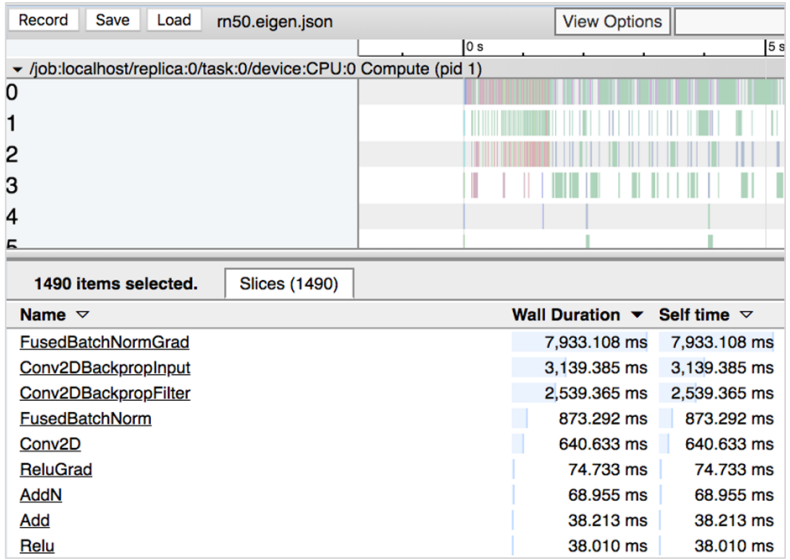
Library is open-source (https://github.com/intel/mkl-dnn) and downloaded automatically when building TensorFlow.

Forward	Backward
Conv2D	Conv2DGrad
Relu, TanH, ELU	ReLUGrad, TanHGrad, ELUGrad
MaxPooling	MaxPoolingGrad
AvgPooling	AvgPoolingGrad
BatchNorm	BatchNormGrad
LRN	LRNGrad
MatMul, Concat	

OPERATOR OPTIMIZATIONS IN RESNET50

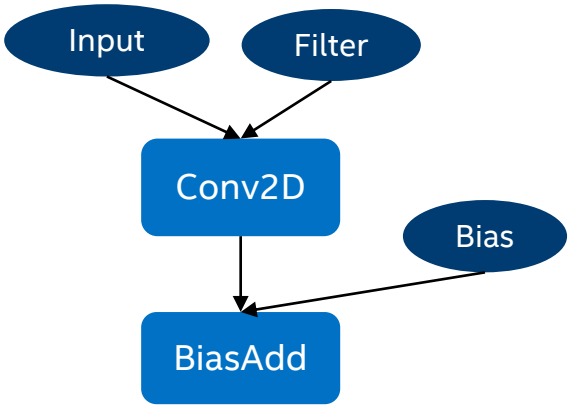


Intel-optimized TensorFlow timeline

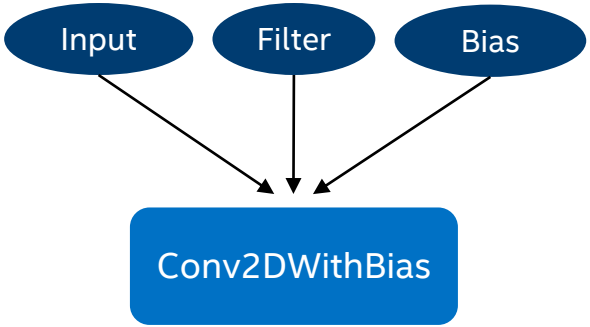


Default TensorFlow timeline

GRAPH OPTIMIZATIONS: FUSION

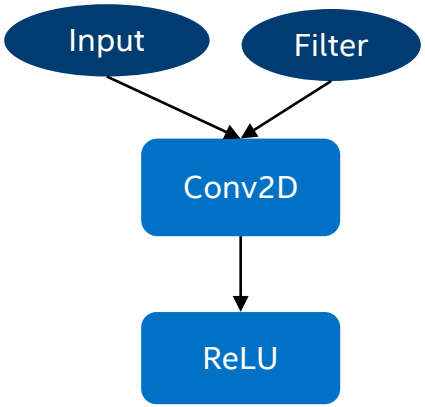


Before Merge

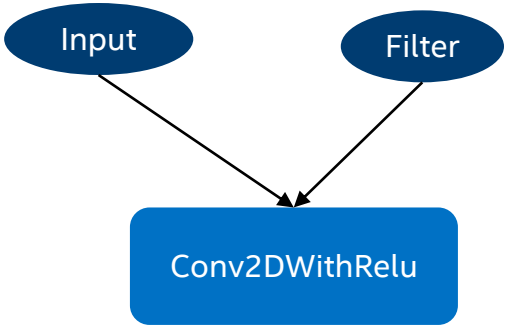


After Merge

GRAPH OPTIMIZATIONS: FUSION



Before Merge

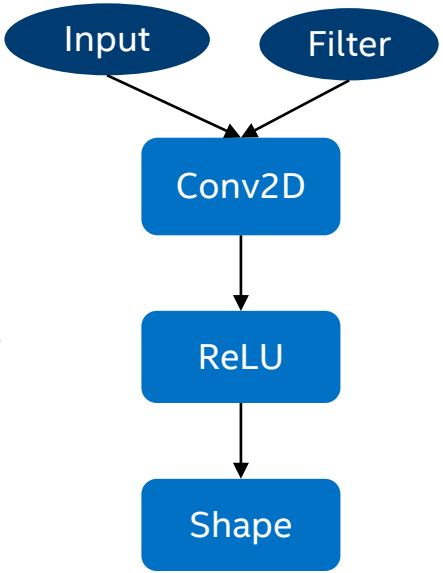


After Merge

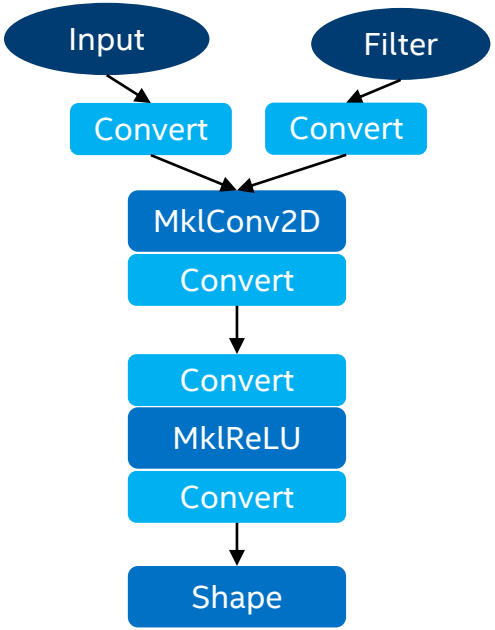
GRAPH OPTIMIZATIONS: LAYOUT PROPAGATION

Converting to/from optimized layout can be less expensive than operating on un-optimized layout.

All MKL-DNN operators use highly-optimized layouts for TensorFlow tensors.



Initial Graph

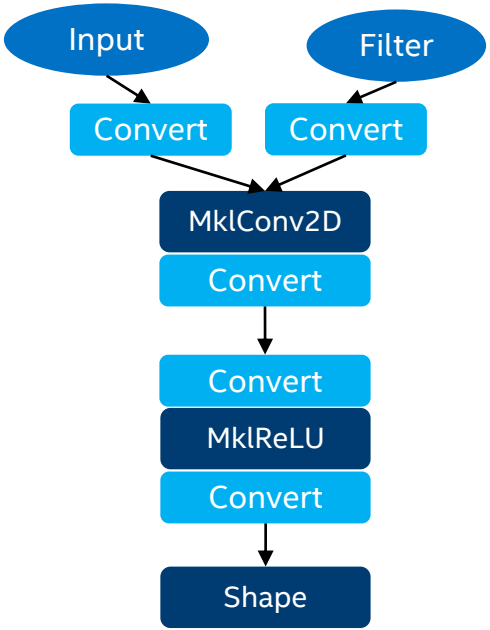


After Layout Conversions

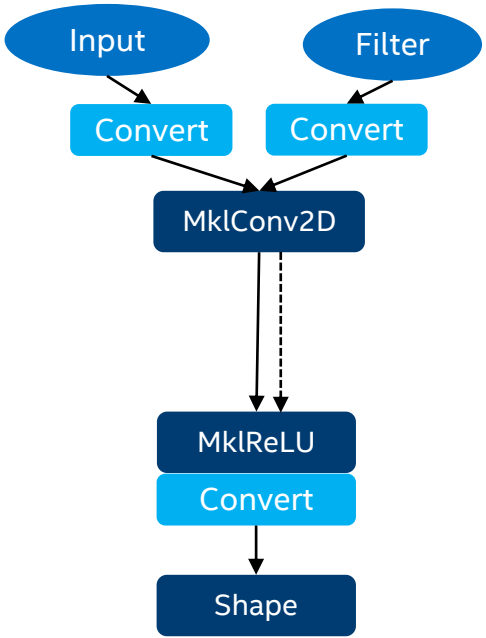
GRAPH OPTIMIZATIONS: LAYOUT PROPAGATION

Did you notice anything wrong with previous graph?

Problem: redundant conversions



After Layout Conversion



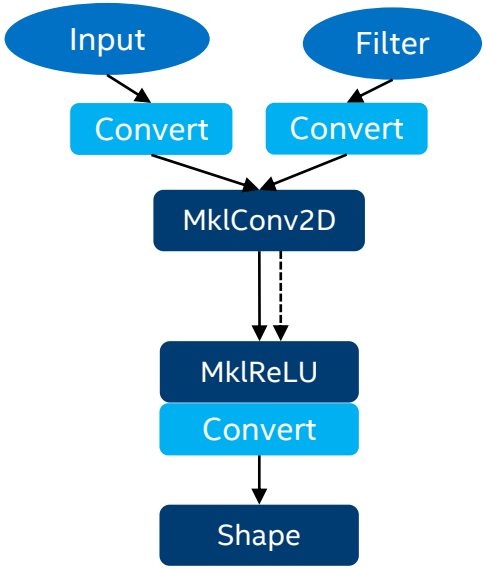
After Layout Propagation

SYSTEM OPTIMIZATIONS: LOAD BALANCING

TensorFlow graphs offer opportunities for parallel execution.

Threading model

1. `inter_op_parallelism_threads` = max number of operators that can be executed in parallel
2. `intra_op_parallelism_threads` = max number of threads to use for executing an operator
3. `OMP_NUM_THREADS` = MKL-DNN equivalent of `intra_op_parallelism_threads`



PERFORMANCE GUIDE

`tf.ConfigProto` is used to set the `inter_op_parallelism_threads` and `intra_op_parallelism_threads` configurations of the `Session` object.

```
>>> config = tf.ConfigProto()
>>> config.intra_op_parallelism_threads = 24
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn

SYSTEM OPTIMIZATIONS: LOAD BALANCING

Incorrect setting of threading model parameters can lead to over- or under-subscription, leading to poor performance.

```
OMP: Error #34: System unable
to allocate necessary resources
for OMP thread:

OMP: System error #11: Resource
temporarily unavailable

OMP: Hint: Try decreasing the
value of OMP_NUM_THREADS.
```

Solution:

Set these parameters for your model manually.

Guidelines on TensorFlow webpage

PERFORMANCE GUIDE

Setting the threading model correctly

We provide best settings for popular CNN models. (https://ai.intel.com/tensorflow-optimizations-intel-xeon-scalable-processor)

Example setting MKL variables with python **`os.environ`** :

```
os.environ["KMP_BLOCKTIME"] = "1"
os.environ["KMP_AFFINITY"] = "granularity=fine,compact,1,0"
os.environ["KMP_SETTINGS"] = "0"
os.environ["OMP_NUM_THREADS"] = "24"
```

Tuning MKL for the best performance

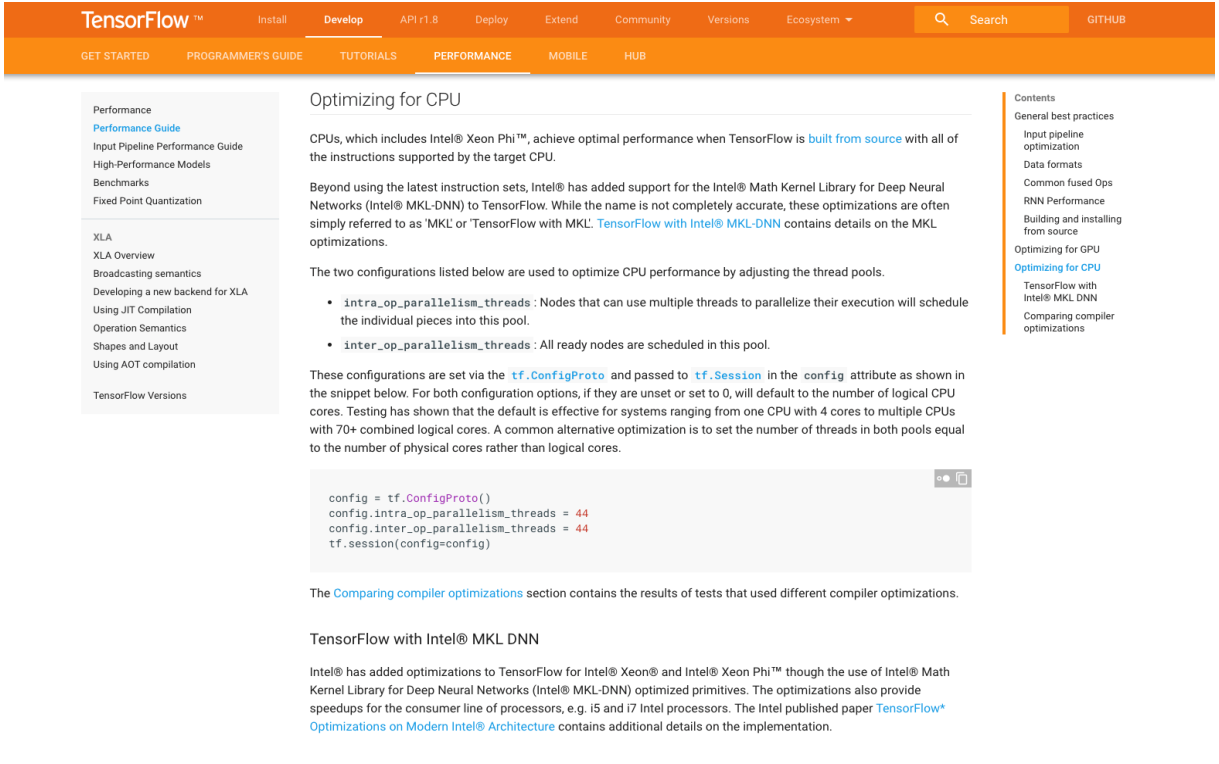
This section details the different configurations and environment variables that can be used to tune the MKL to get optimal performance. Before tweaking various environment variables make sure the model is using the `NCHW` (`channels_first`) data format. The MKL is optimized for `NCHW` and Intel is working to get near performance parity when using `NHWC`.

MKL uses the following environment variables to tune performance:

- KMP_BLOCKTIME - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.
- KMP_AFFINITY - Enables the run-time library to bind threads to physical processing units.
- KMP_SETTINGS - Enables (true) or disables (false) the printing of OpenMP* run-time library environment variables during program execution.
- OMP_NUM_THREADS - Specifies the number of threads to use.

https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn

PERFORMANCE GUIDE



TensorFlow ™ Install Develop API r1.8 Deploy Extend Community Versions Ecosystem Search GITHUB

GET STARTED PROGRAMMER'S GUIDE TUTORIALS PERFORMANCE MOBILE HUB

Performance
Performance Guide
Input Pipeline Performance Guide
High-Performance Models
Benchmarks
Fixed Point Quantization

XLA
XLA Overview
Broadcasting semantics
Developing a new backend for XLA
Using JIT Compilation
Operation Semantics
Shapes and Layout
Using AOT compilation

TensorFlow Versions

Optimizing for CPU

CPUs, which includes Intel® Xeon Phi™, achieve optimal performance when TensorFlow is built from source with all of the instructions supported by the target CPU.

Beyond using the latest instruction sets, Intel® has added support for the Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) to TensorFlow. While the name is not completely accurate, these optimizations are often simply referred to as 'MKL' or 'TensorFlow with MKL'. TensorFlow with Intel® MKL-DNN contains details on the MKL optimizations.

The two configurations listed below are used to optimize CPU performance by adjusting the thread pools.

- `intra_op_parallelism_threads`: Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.
- `inter_op_parallelism_threads`: All ready nodes are scheduled in this pool.

These configurations are set via the `tf.ConfigProto` and passed to `tf.Session` in the `config` attribute as shown in the snippet below. For both configuration options, if they are unset or set to 0, will default to the number of logical CPU cores. Testing has shown that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative optimization is to set the number of threads in both pools equal to the number of physical cores rather than logical cores.

```
config = tf.ConfigProto()
config.intra_op_parallelism_threads = 44
config.inter_op_parallelism_threads = 44
tf.session(config=config)
```

The Comparing compiler optimizations section contains the results of tests that used different compiler optimizations.

TensorFlow with Intel® MKL DNN

Intel® has added optimizations to TensorFlow for Intel® Xeon® and Intel® Xeon Phi™ though the use of Intel® Math Kernel Library for Deep Neural Networks (Intel® MKL-DNN) optimized primitives. The optimizations also provide speedups for the consumer line of processors, e.g. i5 and i7 Intel processors. The Intel published paper TensorFlow* Optimizations on Modern Intel® Architecture contains additional details on the implementation.

Contents
General best practices
Input pipeline optimization
Data formats
Common fused Ops
RNN Performance
Building and installing from source
Optimizing for GPU
Optimizing for CPU
TensorFlow with Intel® MKL DNN
Comparing compiler optimizations

PERFORMANCE GUIDE

References

- https://www.tensorflow.org/performance/performance_guide#tensorflow_with_intel_mkl_dnn
- https://ai.intel.com/tensorflow

Other hints

- Batch-size tuning (if feasible): use multiples of the CPU/core-count
- Multi-node scaling of TF (Horovod)

INTEL-OPTIMIZED TENSORFLOW PERFORMANCE AT A GLANCE

TRAINING THROUGHPUT

Intel-optimized TensorFlow ResNet50 training performance compared to default TensorFlow for CPU

INFERENCE THROUGHPUT

Intel-optimized TensorFlow InceptionV3 inference throughput compared to Default TensorFlow for CPU

Inference and training throughput uses FP32 instructions

System configuration:
CPU Thread(s) per core: 2 **Core(s) per socket**: 28 **Socket(s)**: 2 **NUMA node(s)**: 2 **CPU family**: 6 **Model**: 85 **Model name**: Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz Stepping: 4 **HyperThreading**: ON **Turbo**: ON **Memory** 376GB (12 x 32GB) 24 slots, 12 occupied 2666 MHz Disks Intel RS3WC080 x 3 (800GB, 1.6TB, 6TB) **BIOS** SE5C620.86B.00.01.0004.071220170215 **OS** Centos Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

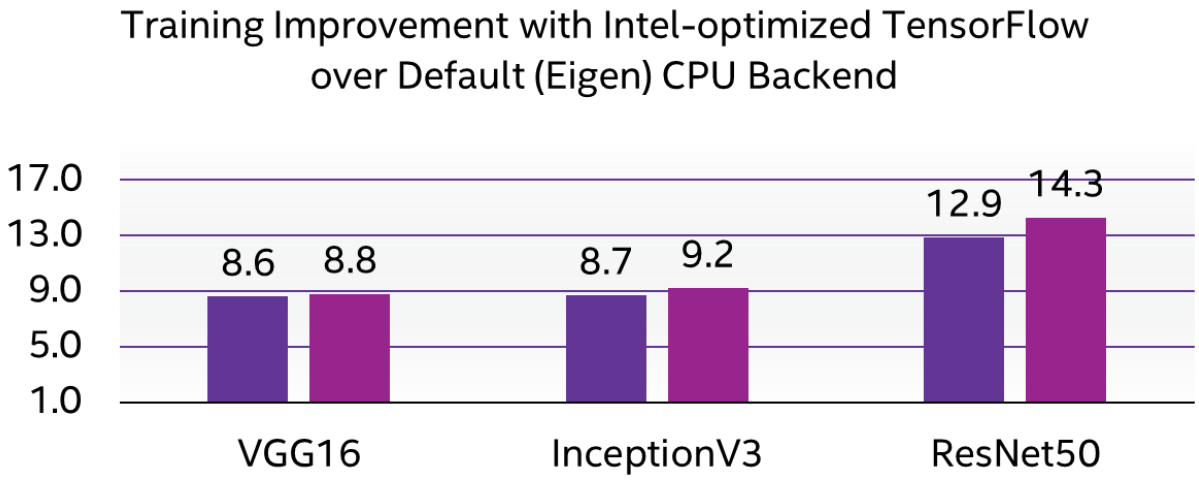
TensorFlow Source:
https://github.com/tensorflow/tensorflow
TensorFlow Commit ID:
926fc13f7378d14fa7980963c4fe774e5922e336.

TensorFlow benchmarks:
https://github.com/tensorflow/benchmarks

Unoptimized TensorFlow may not exploit the best performance from Intel CPUs.

Model	Data_format	Intra_op	Inter_op	OMP_NUM_THREADS	KMP_BLOCKTIME
VGG16	NCHW	56	1	56	1
InceptionV3	NCHW	56	2	56	1
ResNet50	NCHW	56	2	56	1

INTEL-OPTIMIZED TENSORFLOW TRAINING PERFORMANCE



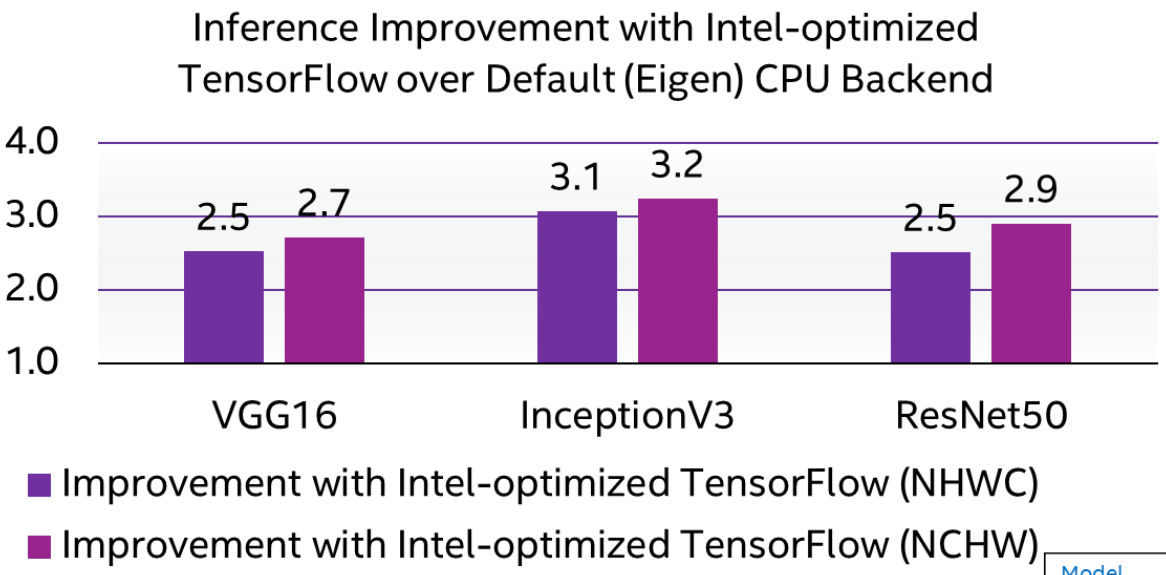
System configuration:
CPU Thread(s) per core: 2 **Core(s) per socket**: 28 **Socket(s)**: 2 **NUMA node(s)**: 2 **CPU family**: 6 **Model**: 85 **Model name**: Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz Stepping: 4 **HyperThreading**: ON **Turbo**: ON **Memory** 376GB (12 x 32GB) 24 slots, 12 occupied 2666 MHz Disks Intel RS3WC080 x 3 (800GB, 1.6TB, 6TB) **BIOS** SE5C620.86B.00.01.0004.071220170215 **OS** Centos Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

TensorFlowSource:
https://github.com/tensorflow/tensorflow
TensorFlow Commit ID:
926fc13f7378d14fa7980963c4fe774e5922e336.

TensorFlow benchmarks:
https://github.com/tensorflow/benchmarks

Model	Data_format	Intra_op	Inter_op	OMP_NUM_THREADS	KMP_BLOCKTIME
VGG16	NCHW	56	1	56	1
InceptionV3	NCHW	56	2	56	1
ResNet50	NCHW	56	2	56	1

INTEL-OPTIMIZED TENSORFLOW INFERENCE PERFORMANCE



System configuration:
CPU Thread(s) per core: 2 **Core(s) per socket**: 28 **Socket(s)**: 2 **NUMA node(s)**: 2 **CPU family**: 6 **Model**: 85 **Model name**: Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz Stepping: 4 **HyperThreading**: ON **Turbo**: ON **Memory** 376GB (12 x 32GB) 24 slots, 12 occupied 2666 MHz Disks Intel RS3WC080 x 3 (800GB, 1.6TB, 6TB) **BIOS** SE5C620.86B.00.01.0004.071220170215 **OS** Centos Linux 7.4.1708 (Core) Kernel 3.10.0-693.11.6.el7.x86_64

TensorFlowSource:
https://github.com/tensorflow/tensorflow
TensorFlow Commit ID:
926fc13f7378d14fa7980963c4fe774e5922e336.

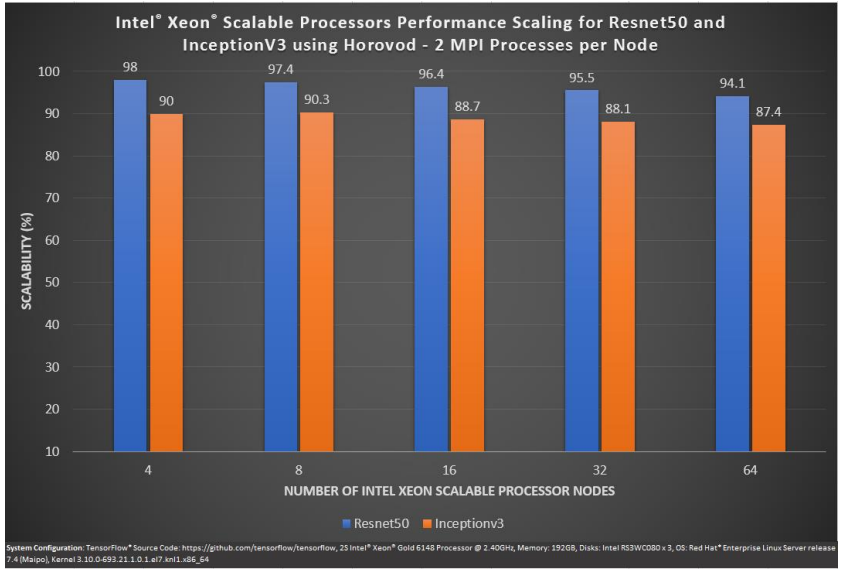
TensorFlow benchmarks:
https://github.com/tensorflow/benchmarks

Model	Data_format	Intra_op	Inter_op	OMP_NUM_THREADS	KMP_BLOCKTIME
VGG16	NCHW	56	1	56	1
InceptionV3	NCHW	56	2	56	1
ResNet50	NCHW	56	2	56	1

MULTI-NODE SCALING: TENSORFLOW + HOROVOD

https://software.intel.com/en-us/articles/using-intel-xeon-processors-for-multi-node-scaling-of-tensorflow-with-horovod

- Developed by Uber for TF, Keras, and PyTorch
- Embarrassing parallel
  - Partitions dataset, trains partitions separately
  - Exchanges weights every Nth epoch
  - Averaging weights (or other strategy)
- Python based
  - Directly from TensorFlow, or
  - TF+Keras
- In contrast: TF/Serving is only for inference



INTEL® OPTIMIZED TENSORFLOW

GETTING INTEL-OPTIMIZED TENSORFLOW

PIP – Python Package Installer (plain "pip install tensorflow" is without MKL/DNN!)

```
$ Python 3.6> pip --user install https://anaconda.org/intel/tensorflow/1.11.0/download/tensorflow-1.11.0-cp36-cp36m-linux_x86_64.whl

$ Python 2.7> pip install --user https://anaconda.org/intel/tensorflow/1.11.0/download/tensorflow-1.11.0-cp27-cp27mu-linux_x86_64.whl
```

Anaconda or Miniconda ("conda“ command is already there if Intel tools are source’d)

```
$ Python 3.7> wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

$ Python 2.7> wget https://repo.anaconda.com/miniconda/Miniconda2-latest-Linux-x86_64.sh

$ bash Miniconda2-latest-Linux-x86_64.sh  # install miniconda (no need if Intel tools are source’d)

$ export PATH=$HOME/miniconda2/bin:$PATH

$ conda create --name mytensorflow tensorflow

$ source activate mytensorflow
```

* To install as non-root user: **pip install --user ...**

BUILD TENSORFLOW WITH INTEL MKL-DNN

```
$  git clone https://github.com/tensorflow/tensorflow.git
     (or rely on https://github.com/tensorflow/tensorflow/releases/latest)
$  cd tensorflow; ./configure
#  AVX-512: --copt=-mfma --copt=-mavx512f --copt=-mavx512cd
                --copt=-mavx512bw --copt=-mavx512vl --copt=-mavx512dq
$  bazel build --config=mkl -c opt --copt=-O2 \
     --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 \
     --copt=-mfma --copt=-mavx2 \
    //tensorflow/tools/pip_package:build_pip_package
$  bazel-bin/tensorflow/tools/pip_package/build_pip_package \
     /tmp/tensorflow_pkg
```

* Some more details: http://libxsmm.readthedocs.io/tensorflow/

BUILD TENSORFLOW (CONT.)

Package the TensorFlow Wheel file

```
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package \
  /tmp/tensorflow_pkg
```

Optional (save Wheel file for future installation):

```
$ cp /tmp/tensorflow_pkg/tensorflow-1.11.0-cp27-cp27mu-linux_x86_64.whl \
  /path/to/mysafeplace
```

Install the TensorFlow Wheel

```
[user] $  pip install --user --upgrade -I \
/tmp/tensorflow_pkg/tensorflow-1.11.0-cp27-cp27mu-linux_x86_64.whl
```

```
[root] $  sudo -H pip install --upgrade -I \
/tmp/tensorflow_pkg/tensorflow-1.11.0-cp27-cp27mu-linux_x86_64.whl
```

TENSORFLOW SERVING

Build TF Serving*

```
$ git clone https://github.com/hfp/serving.git

    (or rely on https://github.com/tensorflow/serving/releases/latest)

$ bazel build --verbose_failures -c opt --copt=-O2 \
    --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0 \
    --copt=-mfma --copt=-mavx2 \
    --action_env TF_REVISION="master" \
  tensorflow_serving/...
```

* Some more details: http://libxsmm.readthedocs.io/tfserving/

TENSORFLOW: QUICK TEST-DRIVE AFTER INSTALLATION

Try optimized TF (pip, conda, or own build)

```
$ git clone https://github.com/tensorflow/models.git tensorflow-models
$ cd tensorflow-models/tutorials/image/alexnet
$ KMP_BLOCKTIME=1 python ./alexnet_benchmark.py
```

Try TF without MKL-DNN

```
$ rm -rf $HOME/.local
$ pip install --target $HOME/.local/lib/python2.7/site-packages numpy
$ pip list --outdated --format=freeze | grep -v '^\-e' | cut -d = -f 1  | xargs -n1 pip install -U --user
$ pip install --user tensorflow
```

TENSORFLOW SCALING

Estimate perf. gain for multiple workers per single system

$ numactl -H # shows CPU/memory numa-nodes

$ KMP_BLOCKTIME=1 numactl **-C 0-23,48-71** python ./alexnet_benchmark.py

Speedup: [duration with full system] * [number of partitions to get full system] / [duration with limited CPU-partition]

Example: above, Alexnet runs on 1 socket of a dual-socket system (2x24 cores).

Training: [0.180 s] * [2] / [0.267 s] → **1.35x**

Typically, the performance peaks with 4 partitions per dual-socket system.

TENSORFLOW SCALING (MULTINODE)

References

- https://ai.intel.com/white-papers/best-known-methods-for-scaling-deep-learning-with-tensorflow-on-intel-xeon-processor-based-clusters/
- https://builders.intel.com/docs/aibuilders/best-practices-for-scaling-deep-learning-training-and-inference-with-tensorflow-on-intel-xeon-processor-based-hpc-infrastructures.pdf

Points of Interests

- Above reference(s) documentation provides scripts to automate setting up TF+Horovod

  Attempt to "read ahead" (above document) as certain cases ("your case") is perhaps covered later

- Since MVAPICH2 is supported as well, Intel MPI is just ready to use (MVAPICH2 compatible)
- If OpenMPI is built, one should take "openmpi-x.y.z-hfi" sources in case of Intel Omnipath
- Currently, Keras may induce (significant) overhead compared to plain TF (data handling, etc.)


intel®
AI Academy