INTEL® OPTIMIZED AI FRAMEWORKS: TENSORFLOW*

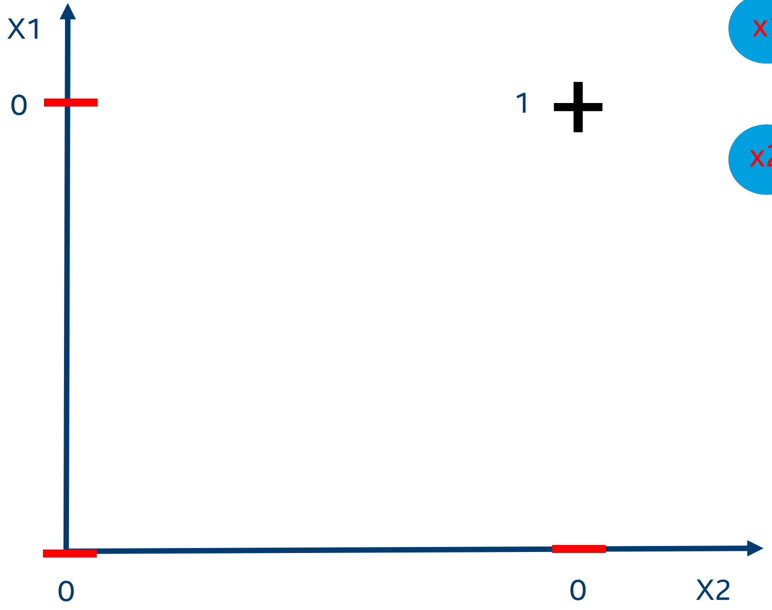
Dr. Fabio Baruffa & Shailen Sobhee
Technical Consulting Engineers, Intel IAGS

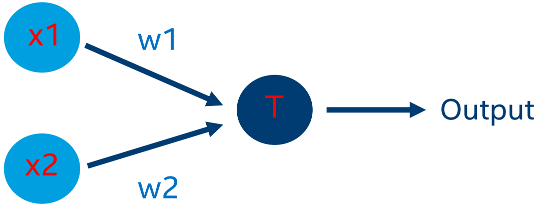
INTRODUCTION TO NEURAL NETWORK

LINEAR CLASSIFIER: SINGLE PERCEPTRON

Linear classifier can solve the **AND** problem.

X1	X2	y
0	0	0
0	1	0
1	0	0
1	1	1



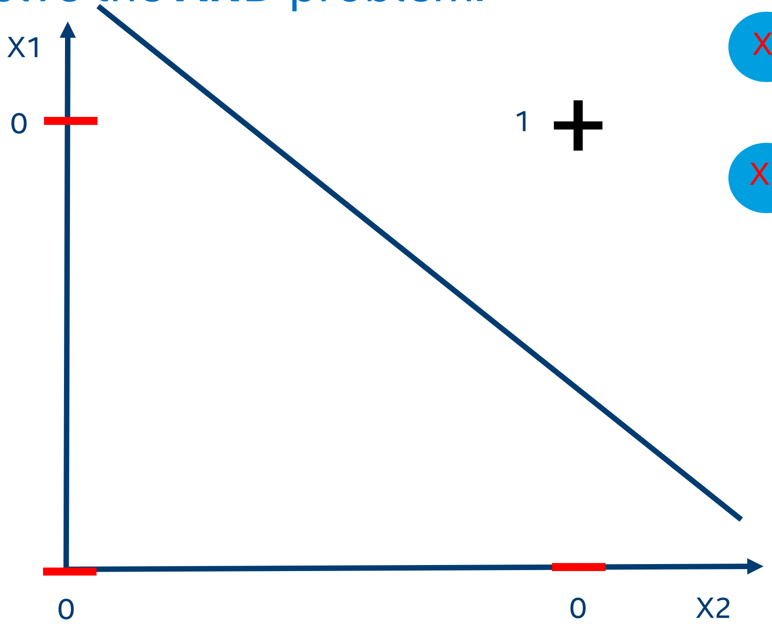


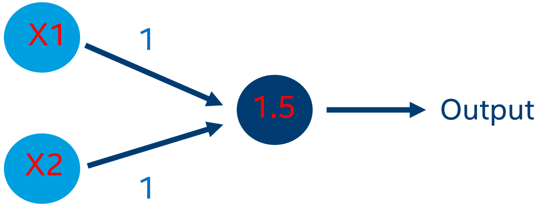
We need to train the network to compute the ‘**unknown**’ weights and threshold

LINEAR CLASSIFIER: SINGLE PERCEPTRON

Linear classifier can solve the **AND** problem.

X1	X2	y
0	0	0
0	1	0
1	0	0
1	1	1





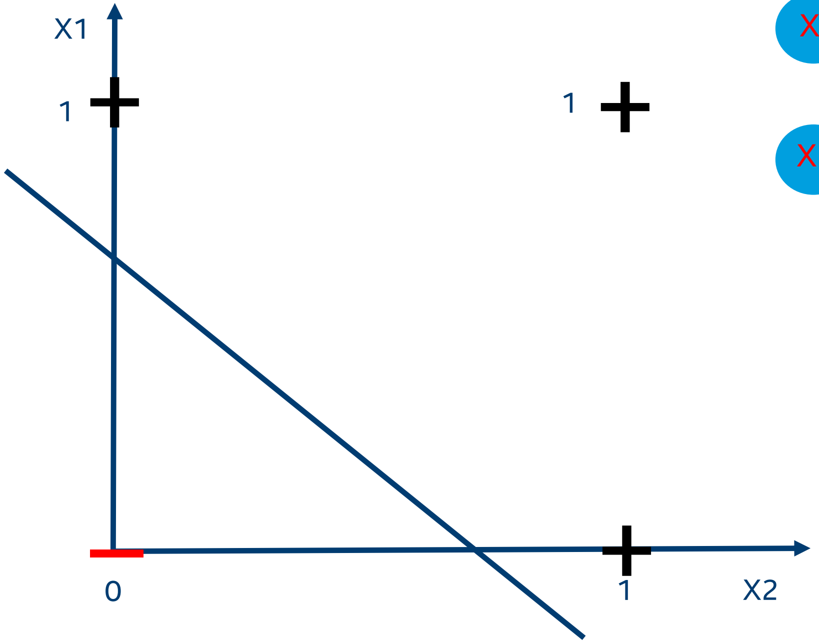
X1 x 1 + X2 x 1 = Z

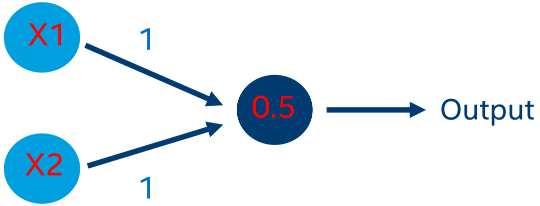
if (Z > 1.5) Output = 1

LINEAR CLASSIFIER: SINGLE PERCEPTRON

Linear classifier can solve the **OR** problem.

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	1





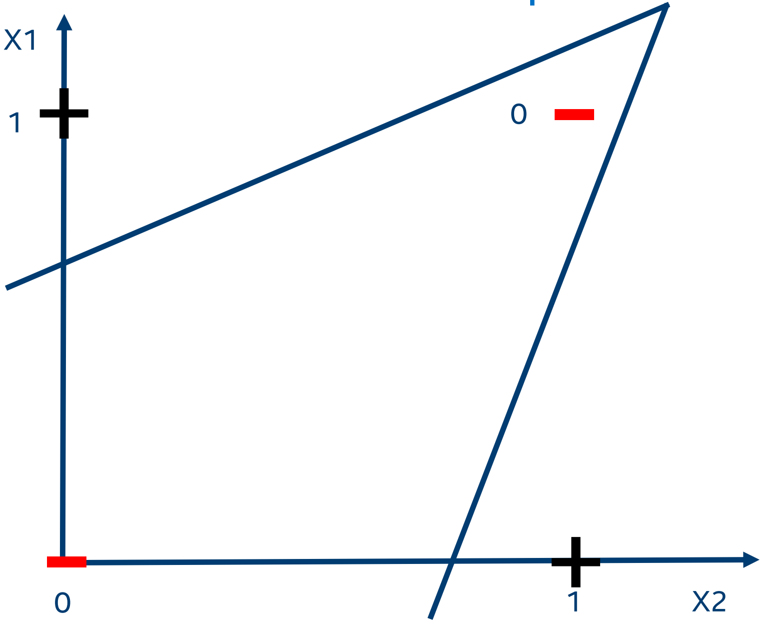
X1 x 1 + X2 x 1 = Z

if (Z > 0.5) Output = 1

WHAT IS WRONG WITH LINEAR CLASSIFIERS ON XOR GATE?

A single linear classifier cannot solve the **XOR** problem.

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	0



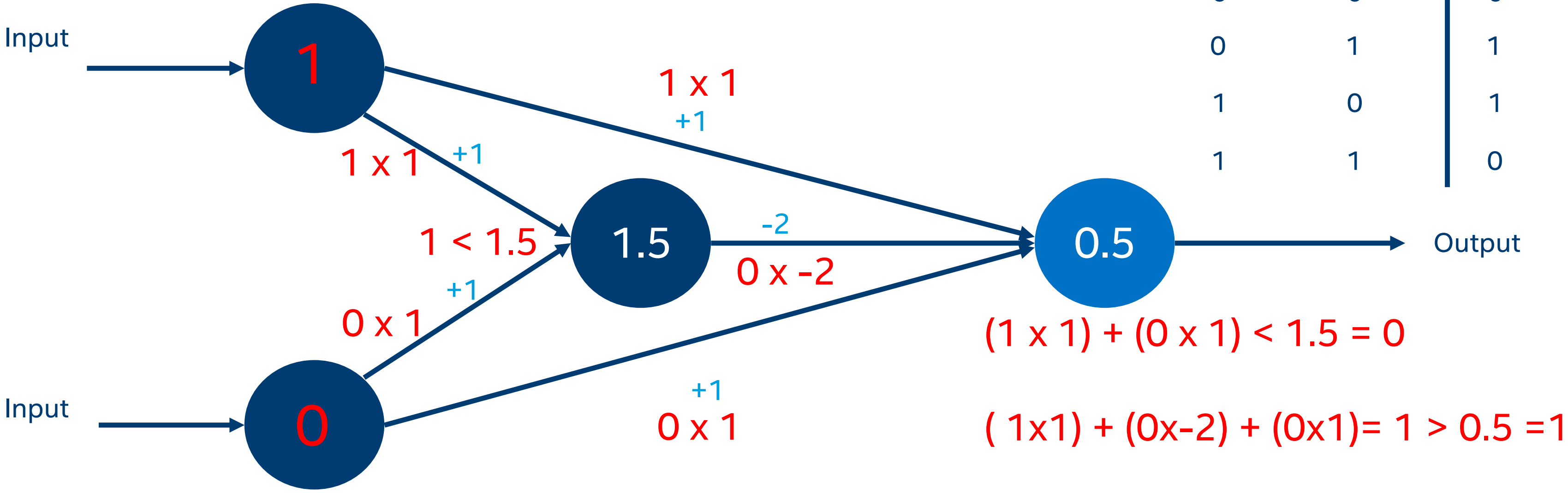
XOR
The counter example to all models

We need two straight line for separation

MULTILAYER PERCEPTRON

XOR = (X1 and not X2) OR (Not X1 and X2)

X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	0

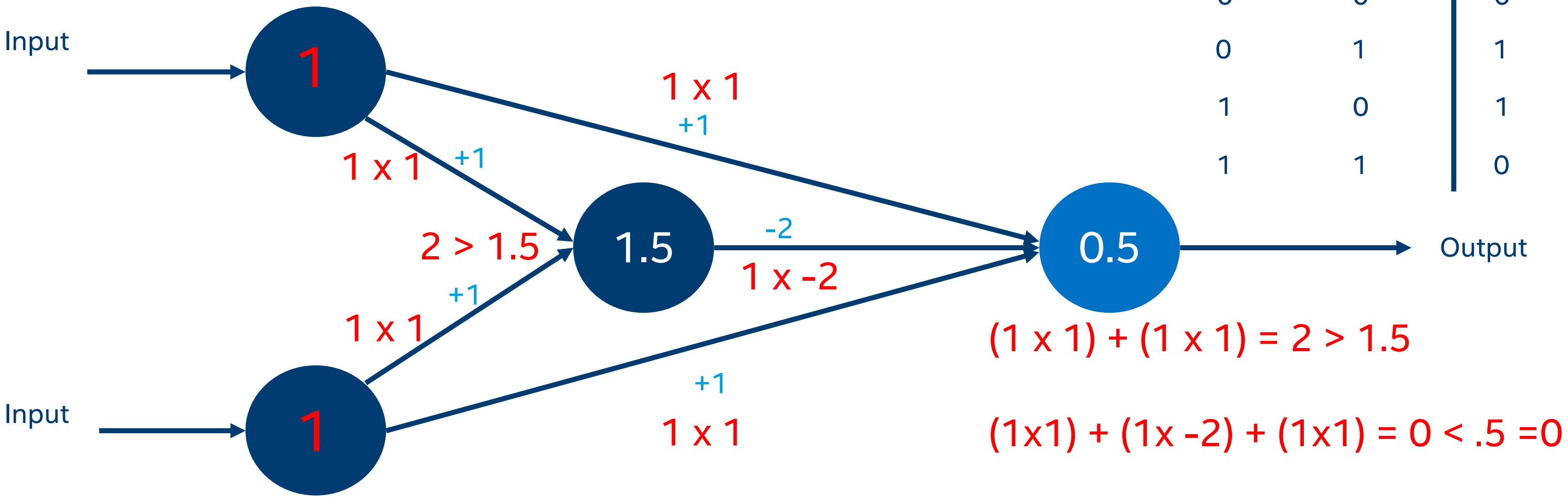


Threshold to 0 or 1

MULTILAYER PERCEPTRON

XOR = (X1 and not X2) OR (Not X1 and X2)

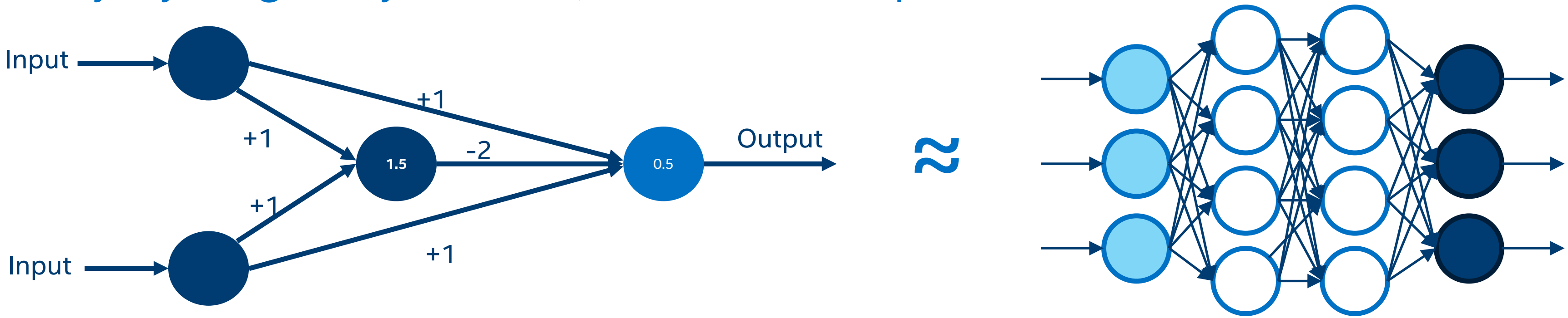
X1	X2	y
0	0	0
0	1	1
1	0	1
1	1	0



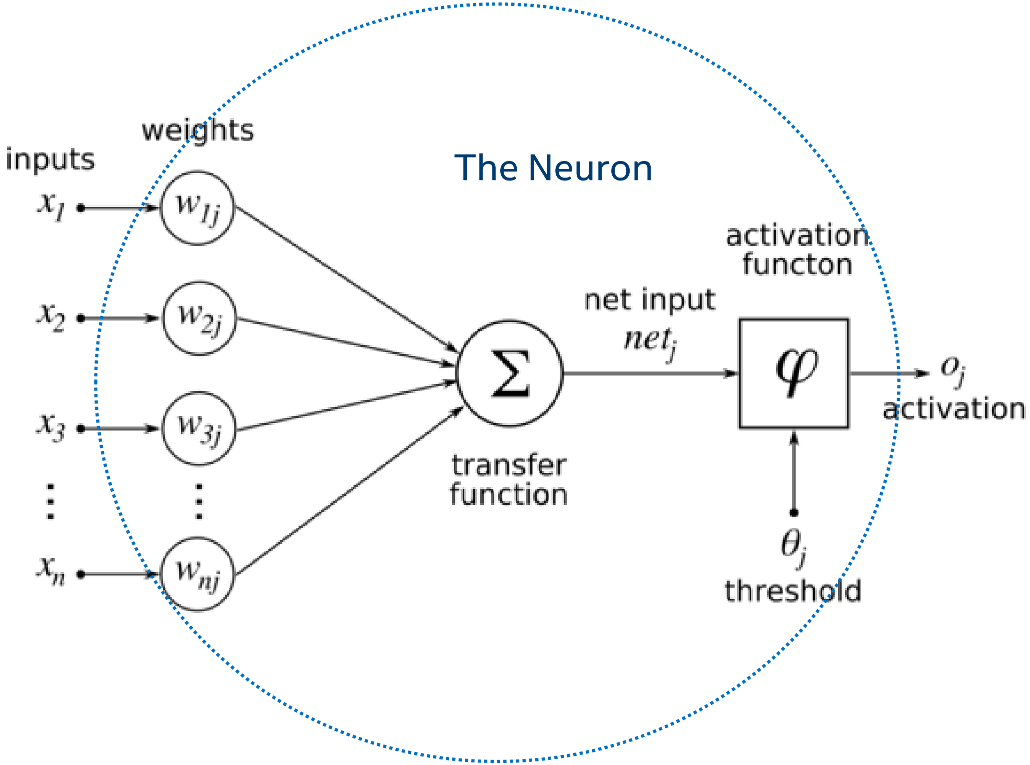
Threshold to 0 or 1

MOTIVATION FOR NEURAL NETS

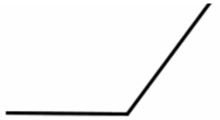
- Use biology as inspiration for mathematical model
- Get signals from previous neurons
- Generate signals (or not) according to inputs
- Pass signals on to next neurons≈
- By layering many neurons, can create complex model



DEEP LEARNING NEURAL NETWORK

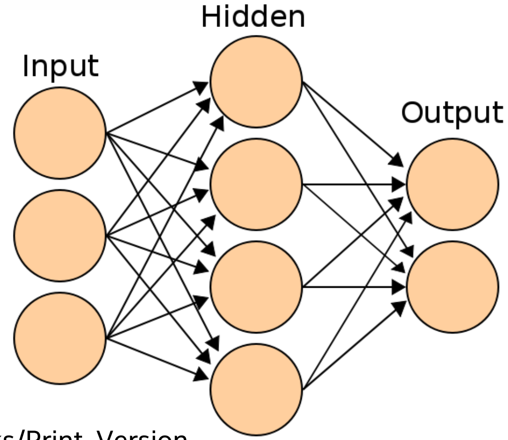


Activation Function



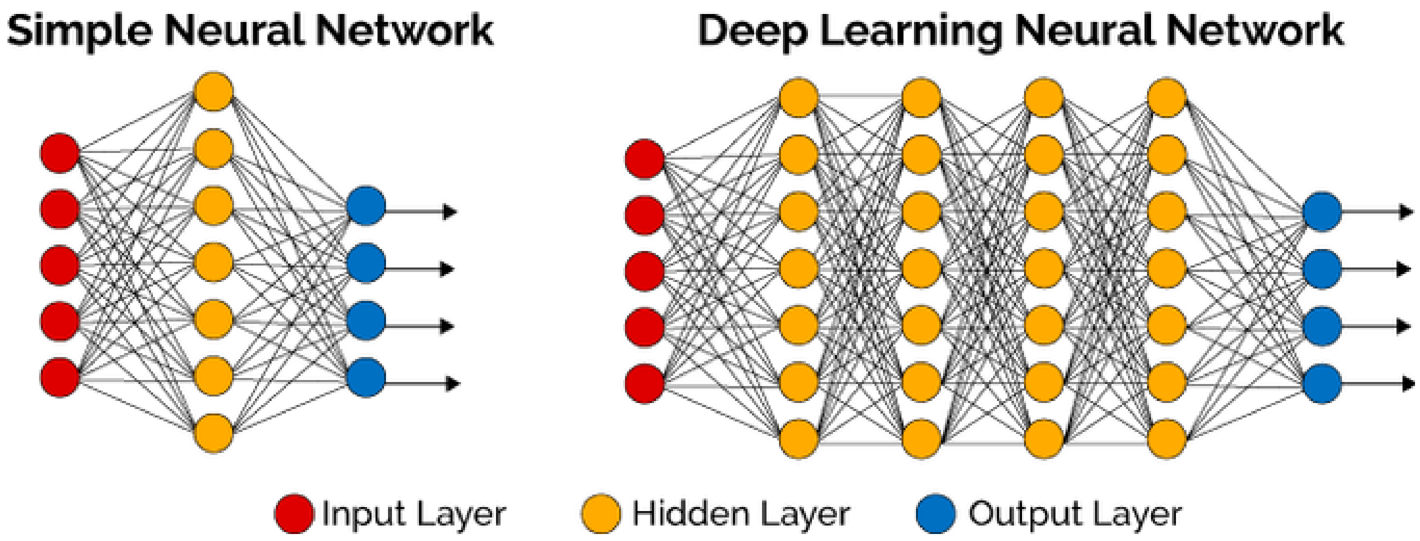
$F(x) = \max(0, x)$

F: a non-linear differentiable function



https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

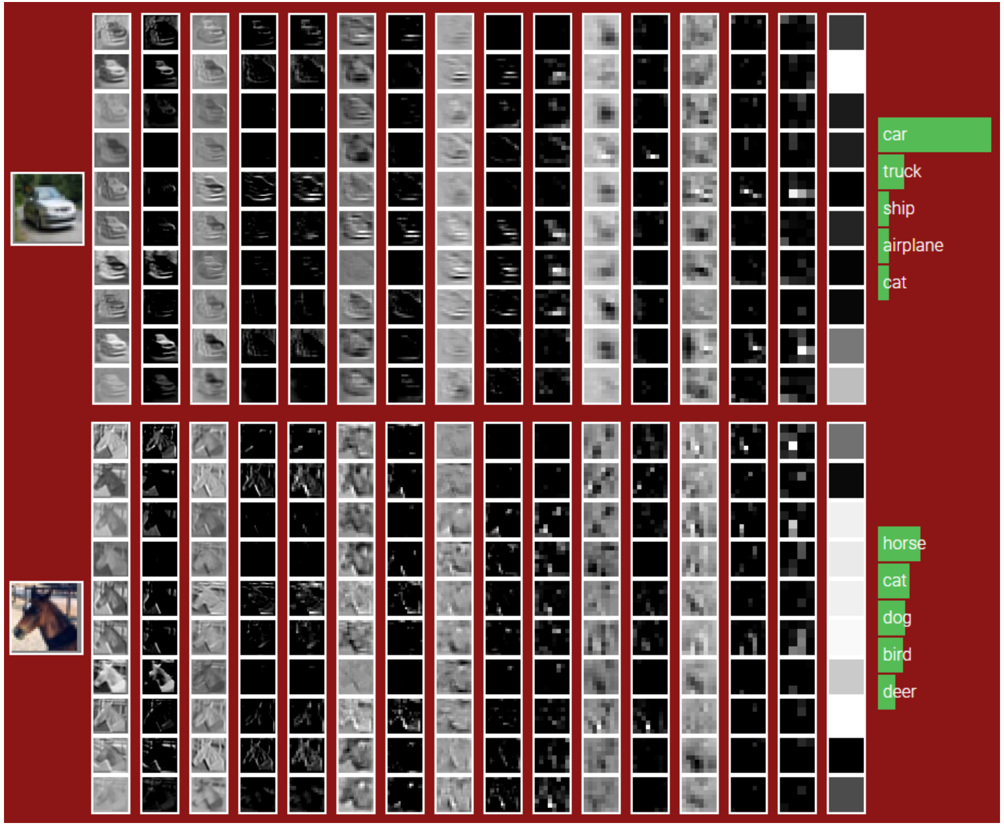
DEEP LEARNING NEURAL NETWORK



IMAGES RECOGNITION

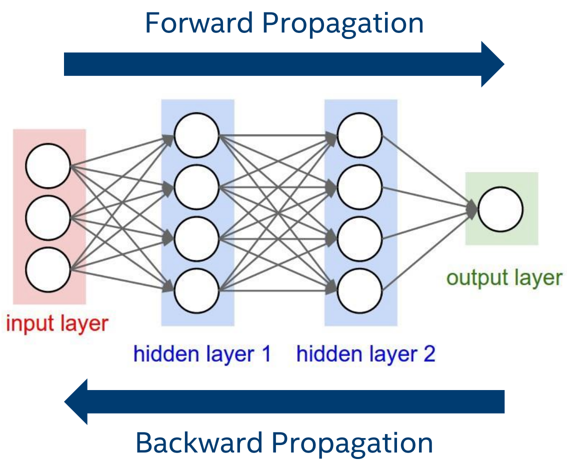
http://cs231n.stanford.edu/

- Training:
  - Use neural network techniques to gather common information among one category objects. ➔ Model
- Inference:
  - Use the gathered common information (model) for prediction or/and generation.

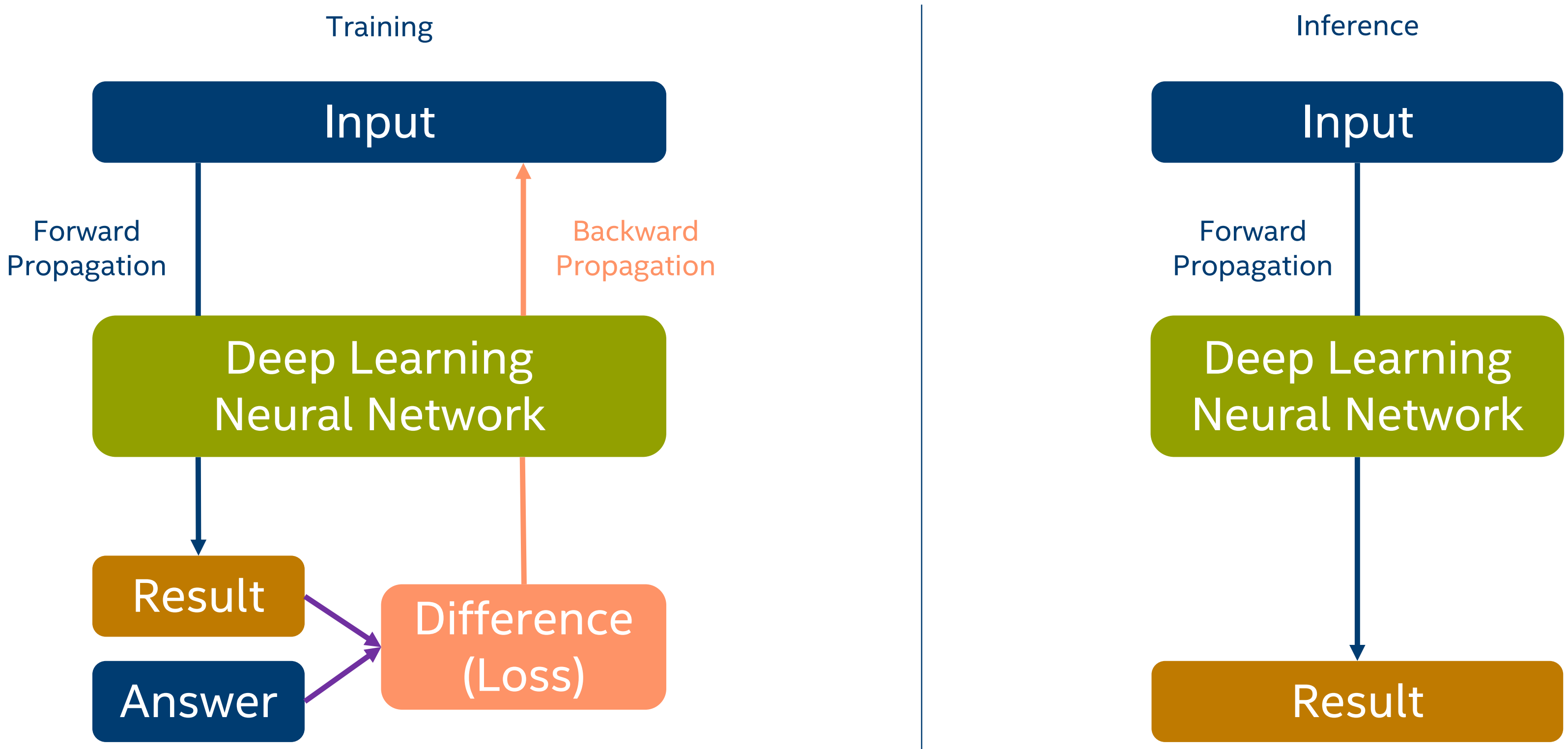


TRAINING TECHNIQUE

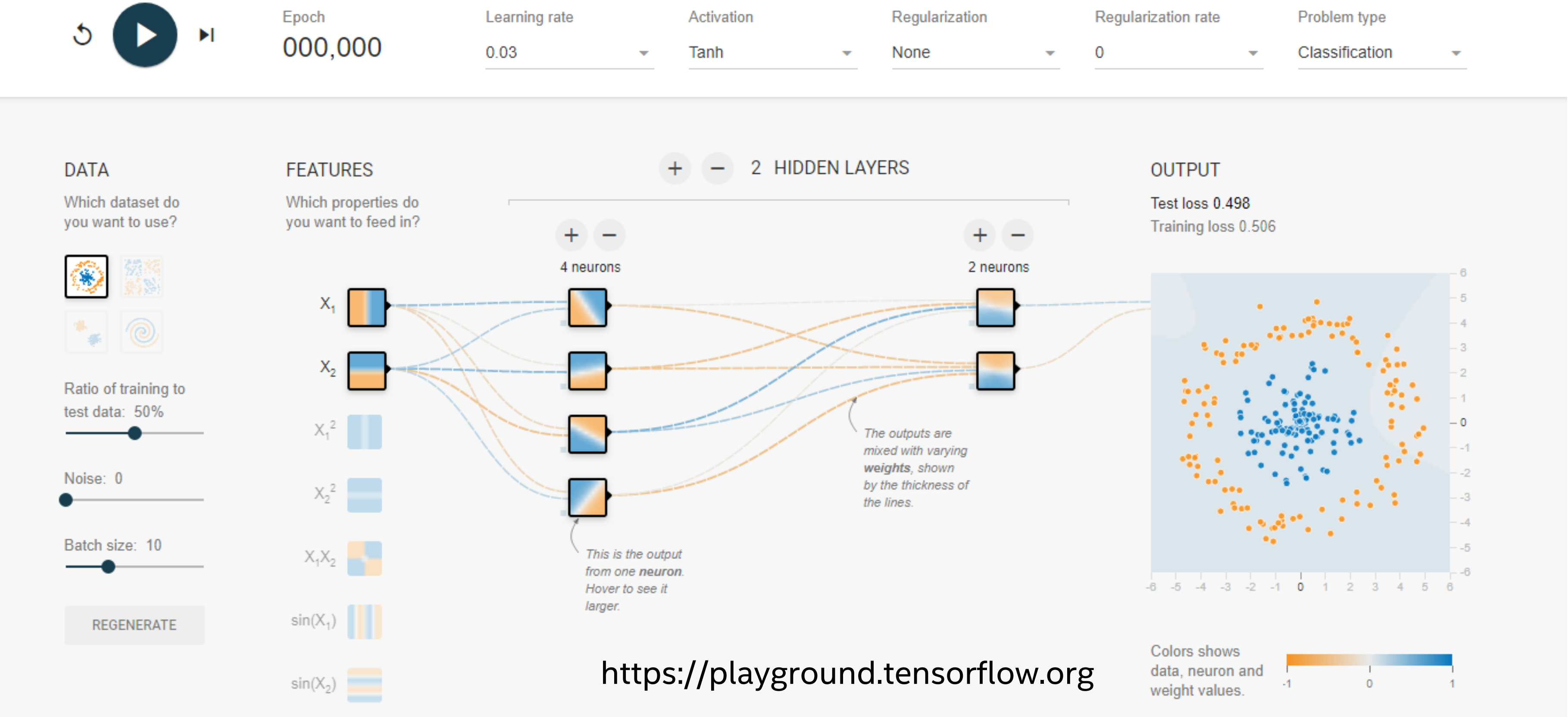
- Techniques used for training and inference
  - Forward propagation
    - Go through the network in forward direction
    - Yield result of the model
  - Backward propagation
    - Go through the network in backward direction
    - Check how different results of the model is against correct answers
    - Update model parameters based on the differences



TRAINING AND INFERENCE

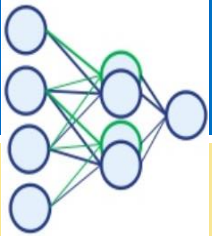


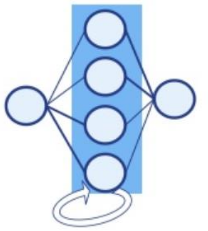
NEURAL NETWORK PLAYGROUND



https://playground.tensorflow.org

Deep Neural Network (DNN)





Convolution Neural Network (CNN)	Recurrent Neural Network (RNN)
Image Recognition, Outline of object in an image, Video processing	Image captioning, Text generation, Language translation
Hierarchical	Sequential
extract position invariant features	model units in sequence
Learns to recognize patterns across space	Learns to recognize patterns across time
The lines and curves I saw will help me recognize the faces and objects	What I spoke last will impact what I will speak next

The number of neurons in a layer (hidden size) and batch size can make DNN performance vary dramatically. This suggests that optimization of these two parameters is crucial to good performance of both CNNs and RNNs [1]

Types of RNN	
Long Short-Term Memory (LSTM)	Gated Recurrent Unit (GRU)

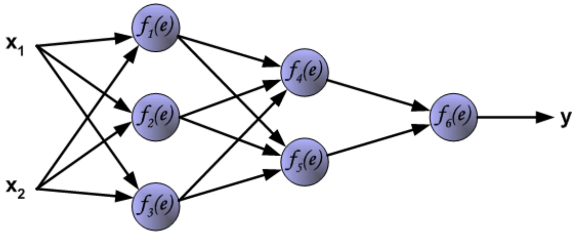
In empirical evaluations, Chung et al. (2014) and Jozefowicz et al. (2015) found there is no clear winner between GRU and LSTM. In many tasks, they yield comparable performance and tuning hyperparameters like layer size is often more important than picking the ideal architecture [1]

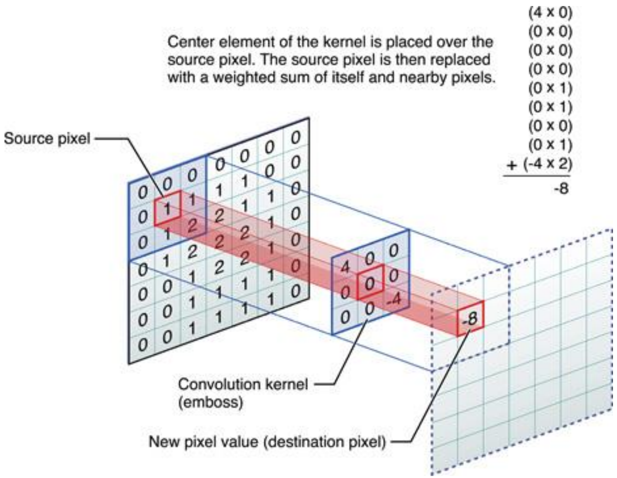
(1) Comparative Study of CNN and RNN for Natural Language Processing : https://arxiv.org/pdf/1702.01923.pdf

Image Source: https://www.slideshare.net/sheemap/convolutional-neural-netwoks

DEEP LEARNING OPERATIONS

- Fully Connected
- Convolution 2D
- Convolution 3D
- Batch Normalization
- ReLU
- Dropout
- RNN
- …





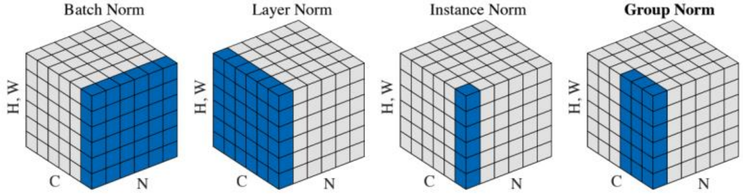


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

WHY DO WE NEED DEEP LEARNING FRAMEWORKS?

- Questions:
  - How long will it take to write code to implement those processes?
  - Are you willing to write code to implement those processes every time when you would like to develop a new topology?
- Deep learning frameworks implemented these complicated operations for you
  - Easy to use
  - High efficiency for development of topologies
  - Even if you don’t understand mathematic theories underneath

INTEL® AI OPTIMIZED FRAMEWORKS

Popular DL Frameworks are now optimized for CPU!

CHOOSE YOUR FAVORITE FRAMEWORK

TensorFlow™ * | Caffe * | mxnet * | BigDL FOR APACHE Spark™ * | PYTORCH

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

More under optimization: Caffe2* PYTORCH* PaddlePaddle*

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.

AI (ML & DL) SOFTWARE STACK FOR INTEL® PROCESSORS

Deep learning and AI ecosystem includes edge and datacenter applications.

- Open source frameworks (Tensorflow*, MXNet*, PyTorch*, PaddlePaddle*)
- Intel deep learning products (, BigDL, OpenVINO™ toolkit)
- In-house user applications

Intel® MKL and Intel® MKL-DNN optimize deep learning and machine learning applications for Intel® processors :

- Through the collaboration with framework maintainers to upstream changes (Tensorflow*, MXNet*, PyTorch, PaddlePaddle*)
- Through Intel-optimized forks (Caffe*)
- By partnering to enable proprietary solutions

Intel® MKL-DNN is an open source performance library for deep learning applications (available at https://github.com/intel/mkl-dnn)

- Fast open source implementations for wide range of DNN functions
- Early access to new and experimental functionality
- Open for community contributions

Intel® MKL is a proprietary performance library for wide range of math and science applications

Distribution: Intel Registration Center, package repositories (apt, yum, conda, pip), Intel® Parallel Studio XE, Intel® System Studio

INTRODUCTION TO TENSORFLOW*

WHAT IS TENSORFLOW?

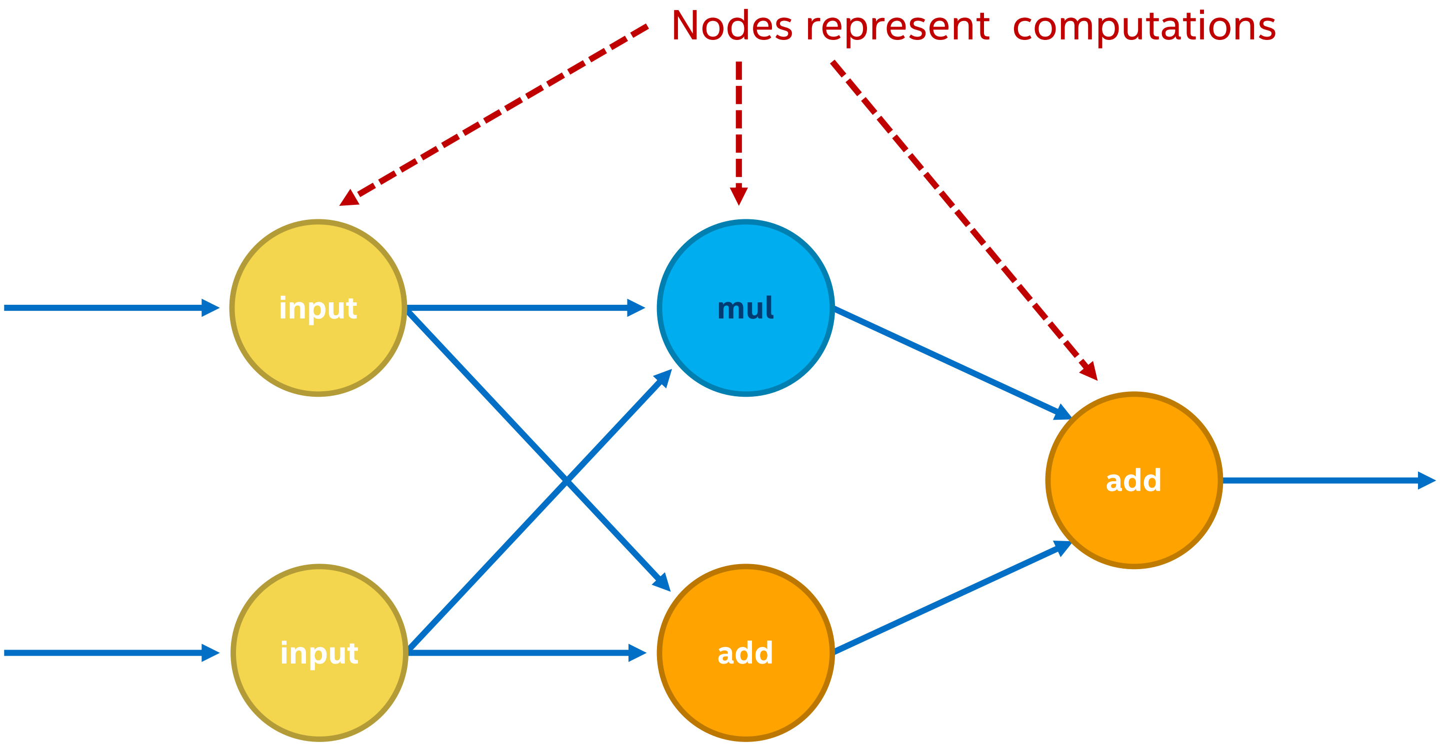
- Framework for math using Computation Graphs
- Has features specifically for machine learning
- Primary interface is Python, integrates NumPy
- Designed to be flexible, scalable and deployable

TensorFlow

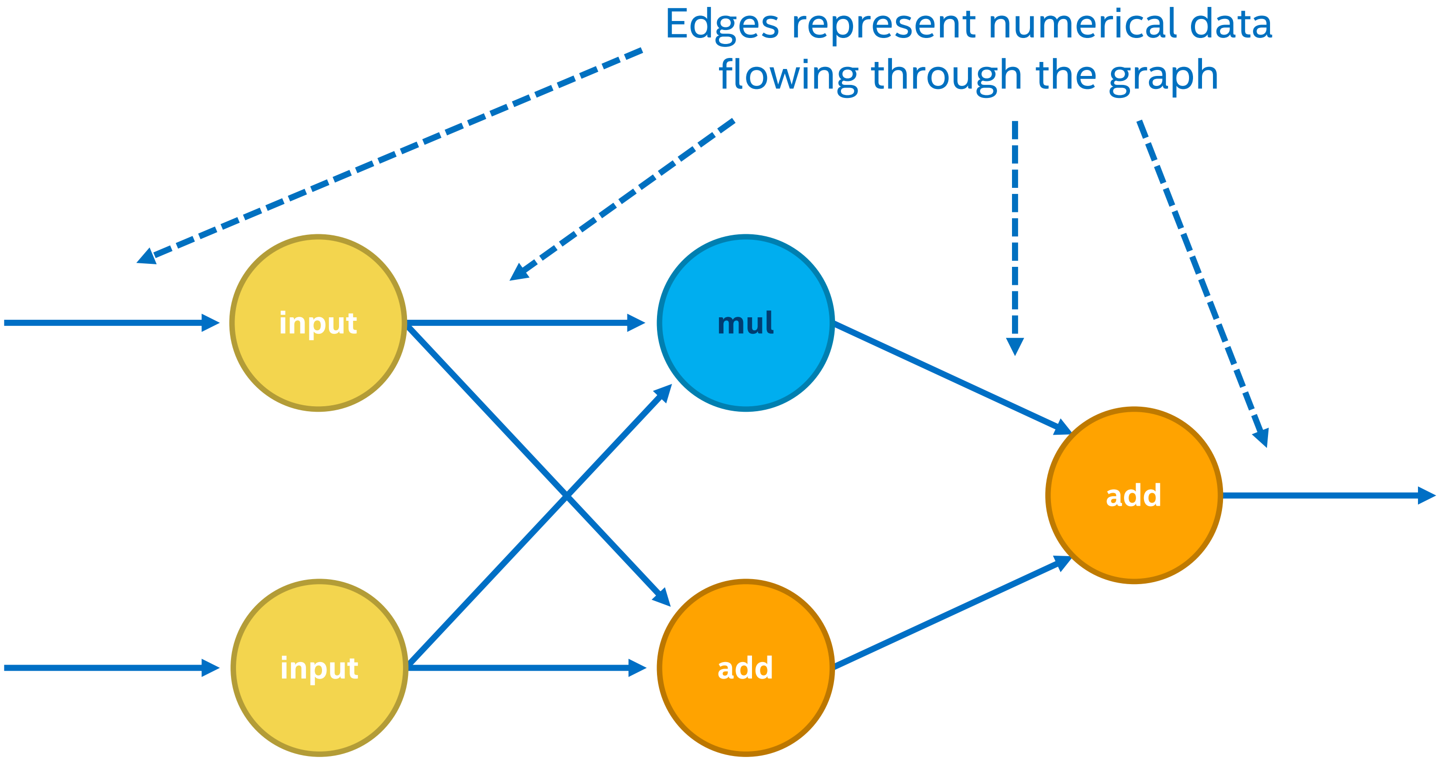
- Easy to install including the Intel® optimization via using conda
  - `conda install tensorflow -c intel`

https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide

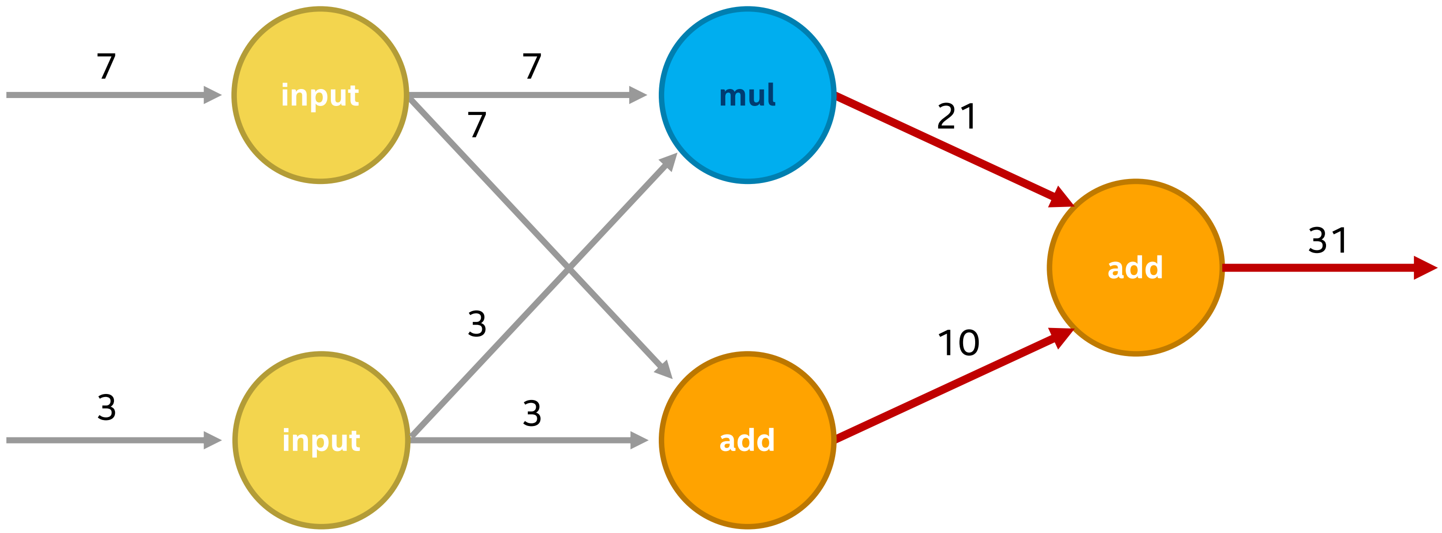
COMPUTATION GRAPH



COMPUTATION GRAPH

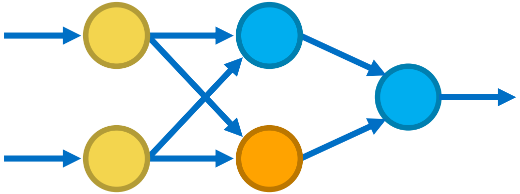


DATA FLOW

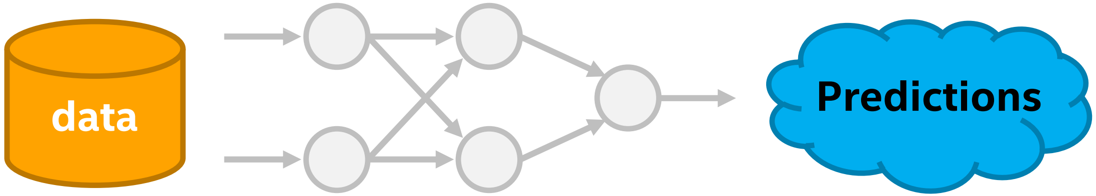


TWO-STEP PROGRAMMING PATTERN

1. Define a computation graph

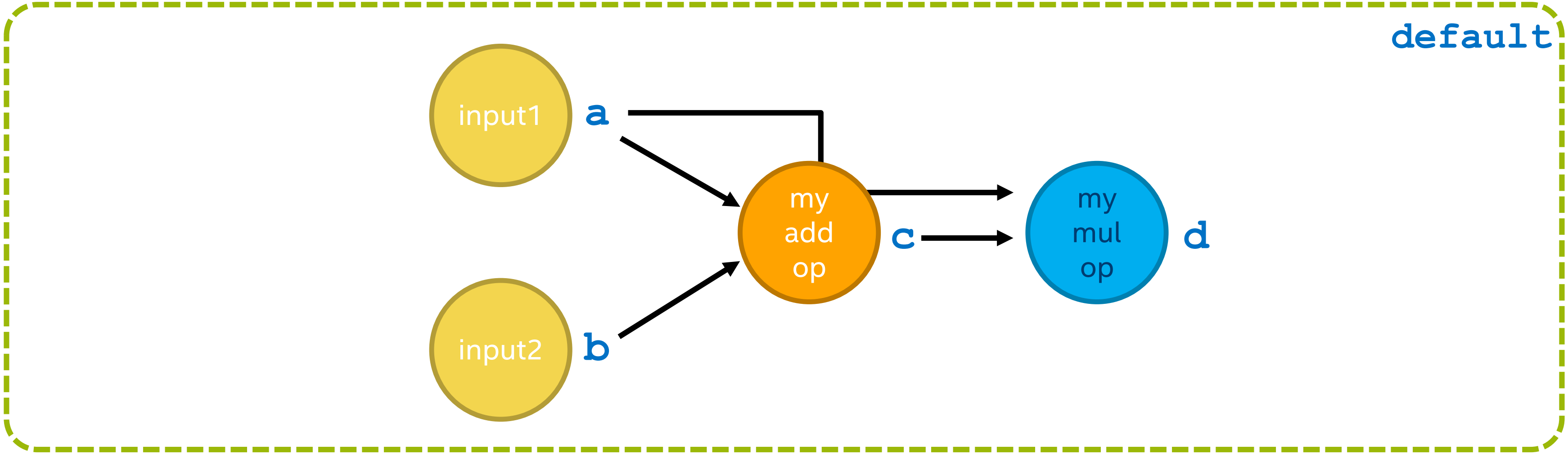


2. Run the graph



DEFINE A COMPUTATIONAL GRAPH

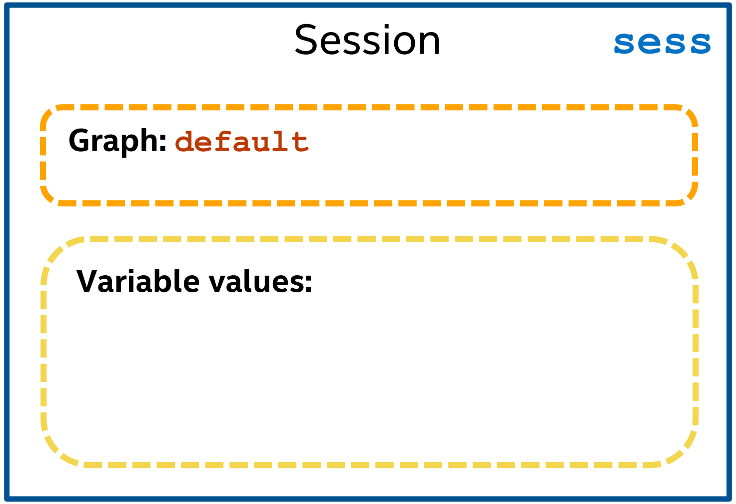
```
>>> a = tf.placeholder(tf.float32, name="input1")
>>> b = tf.placeholder(tf.float32, name="input2")
>>> c = tf.add(a, b, name="my_add_op")
>>> d = tf.multiply(a, c, name="my_mul_op")
```

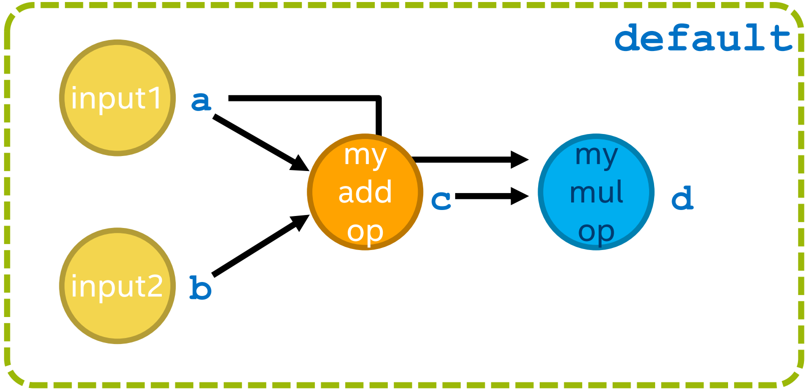


RUN THE GRAPH

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

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

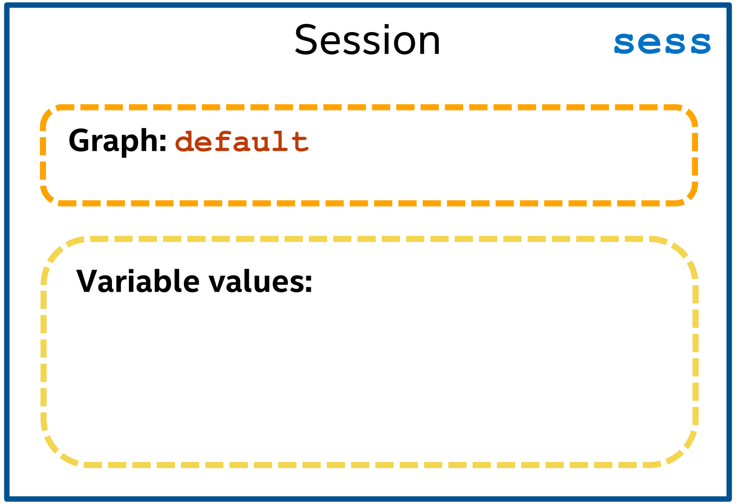


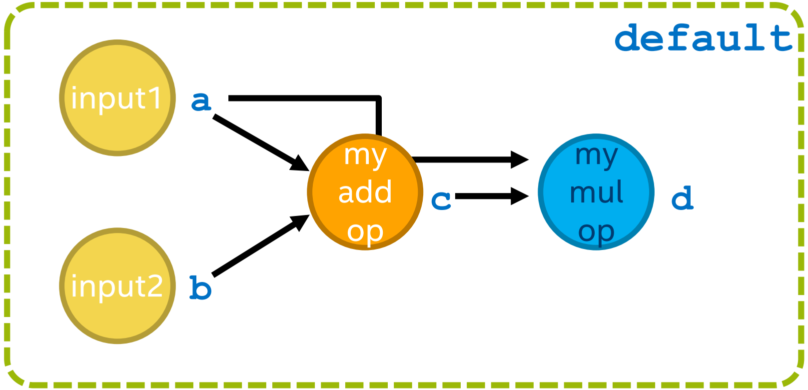


SETUP THE CONFIGURATION (OPTIONAL)

```
>>> config = tf.ConfigProto(inter_op_parallelism_threads=2,
                            intra_op_parallelism_threads=44)

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



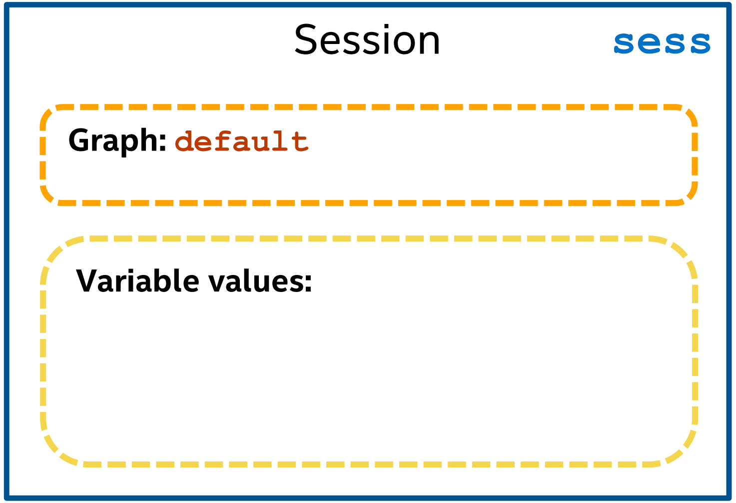


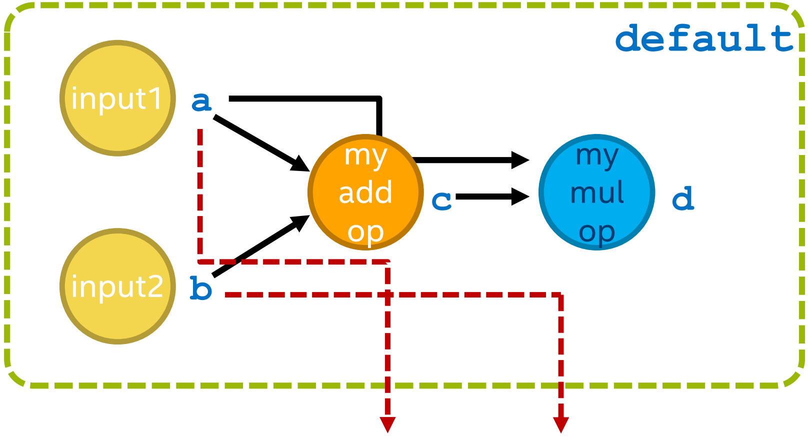
INITIALIZE THE VALUES

`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}
```





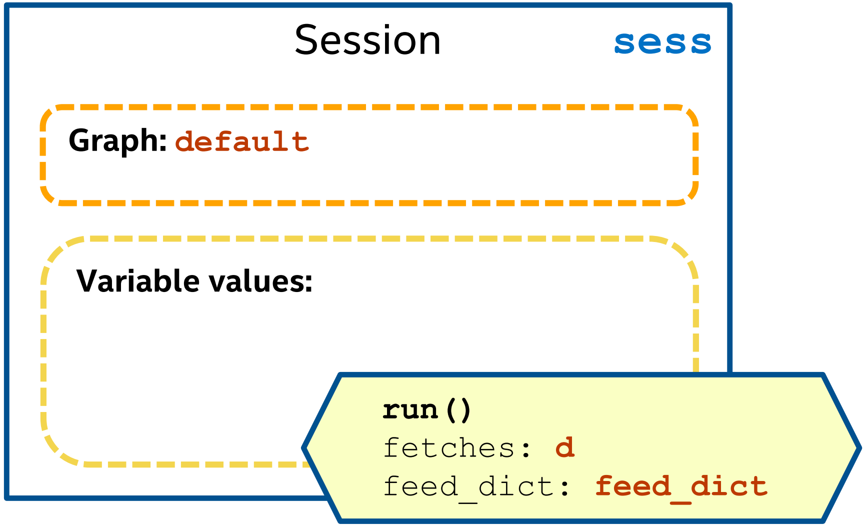
`feed_dict: {a: 3.0, b: 2.0}`

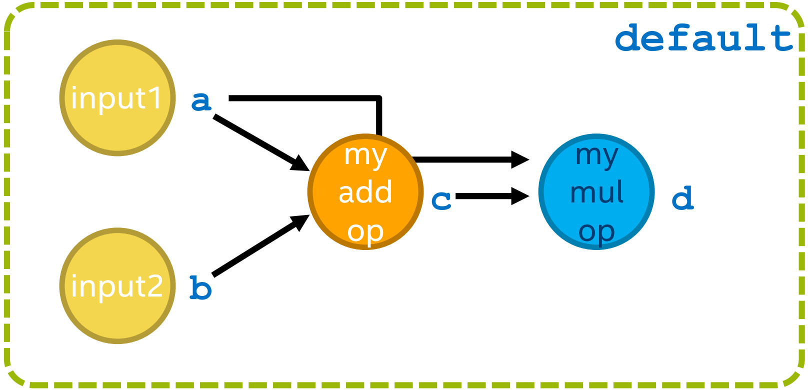
EXECUTE THE GRAPH

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}`

MAIN TENSORFLOW API

Graph

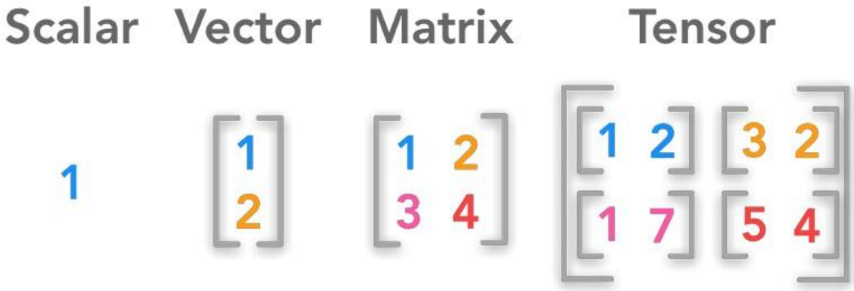
- Container for operations and tensors

Operation

- Nodes in the graph
- Represent computations

Tensor

- Edges in the graph
- Represent data



Source: https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/

TENSORFLOW* GRAPH EXAMPLE:

`TensorFlowBasic.ipynb`

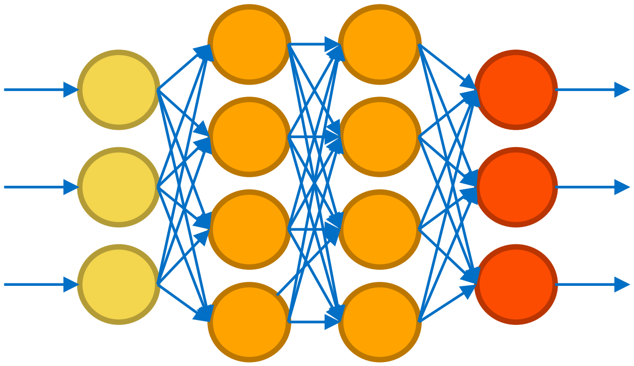
NEURAL NETWORK WITH TENSORFLOW*

NEURAL NETWORK

Use biology as inspiration for math model

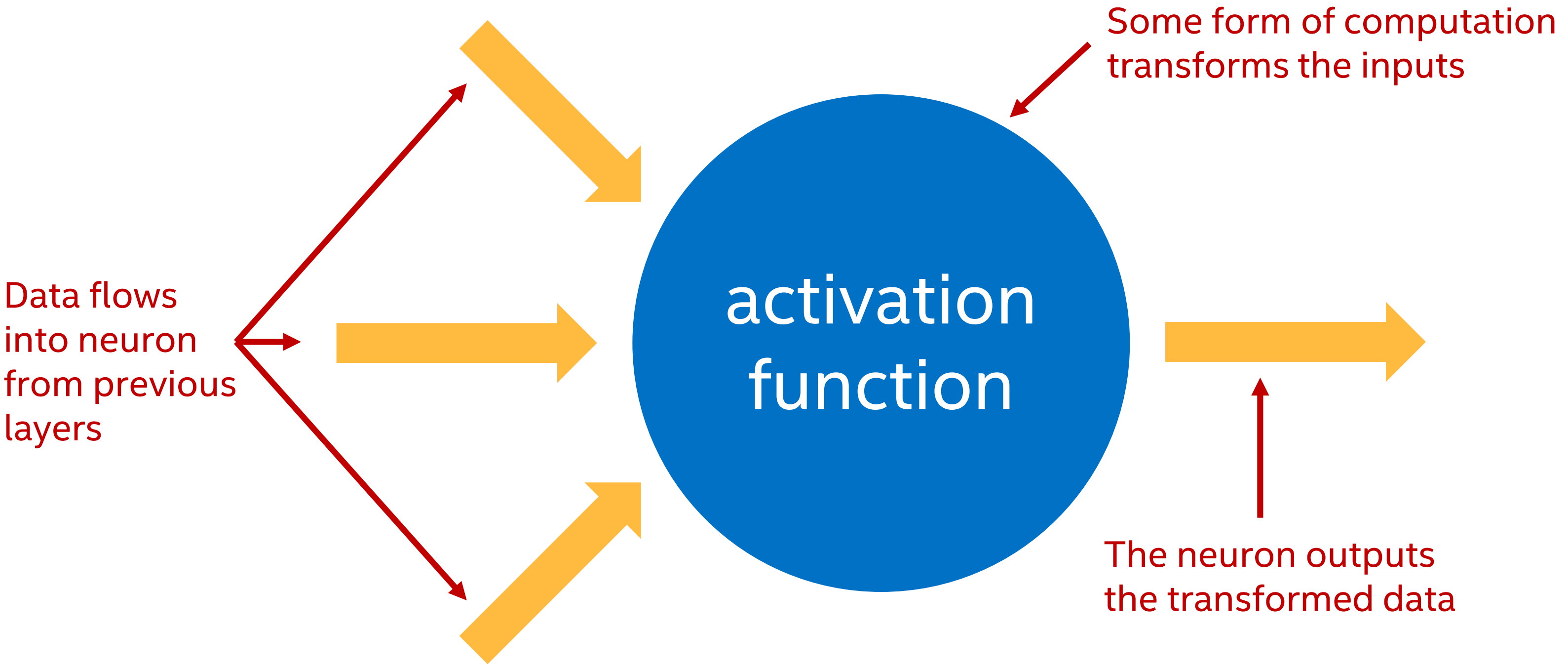
Neurons:

- Get signals from previous neurons
- Generate signal (or not) according to inputs
- Pass that signal on to future neurons



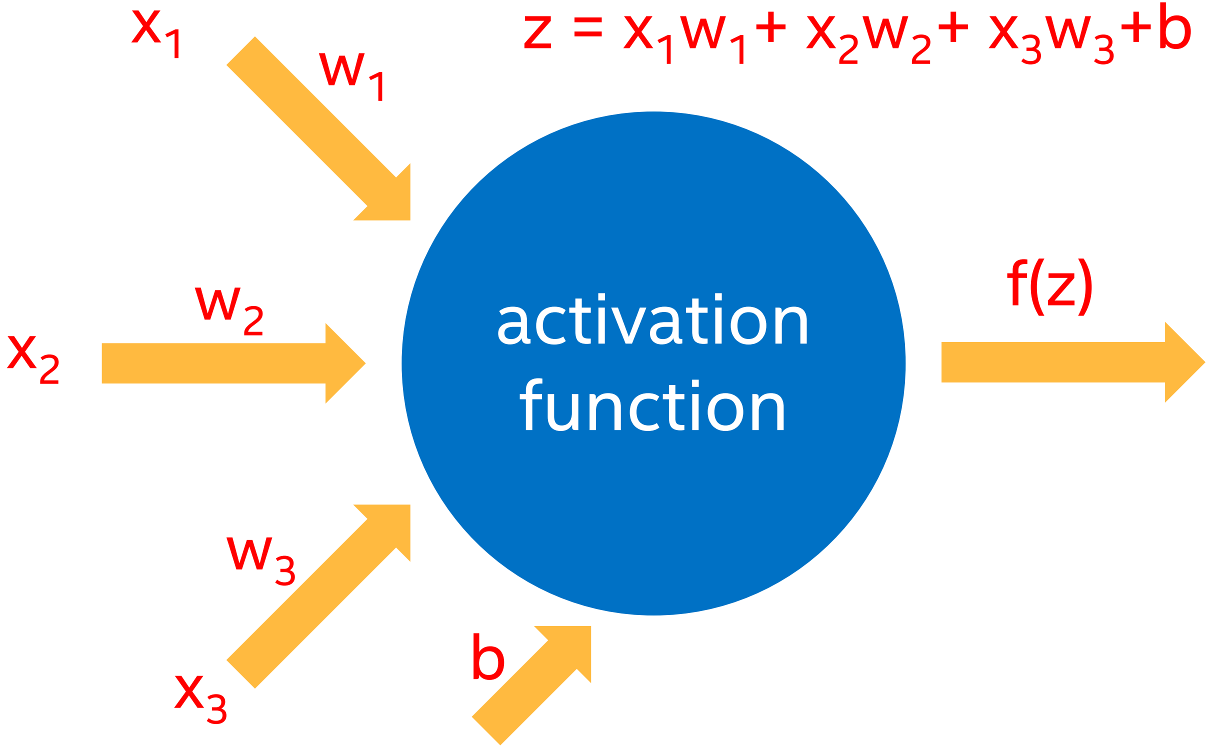
By layering many neurons, can create complex model

READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH

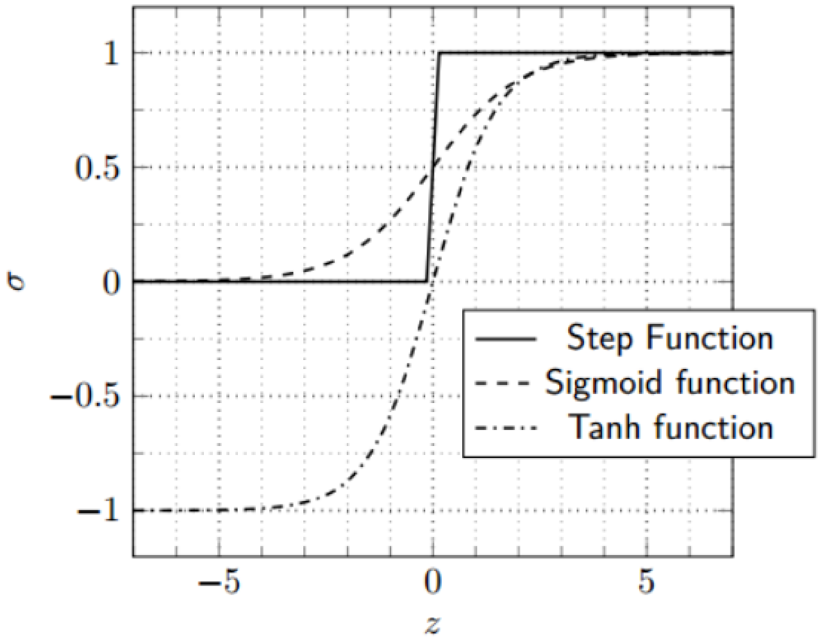


READS ROUGHLY THE SAME AS A TENSORFLOW GRAPH

$$z = x_1w_1 + x_2w_2 + x_3w_3 + b$$



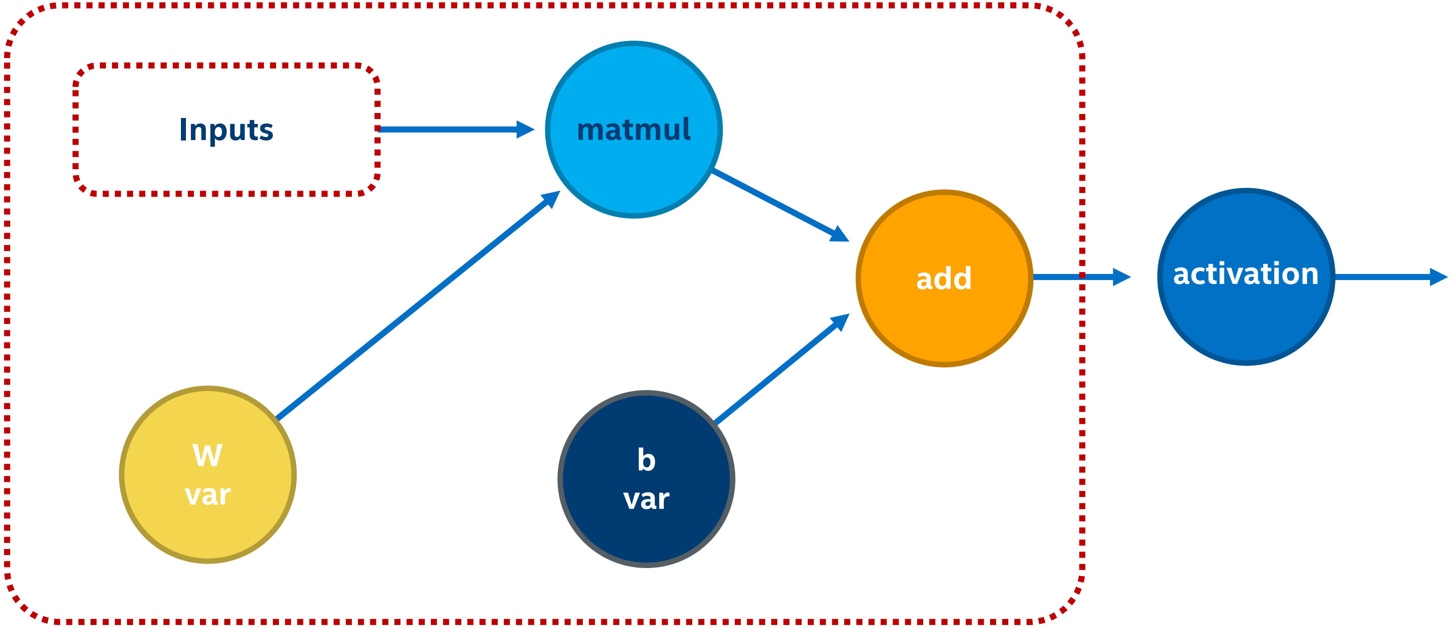
TYPES OF ACTIVATION FUNCTIONS



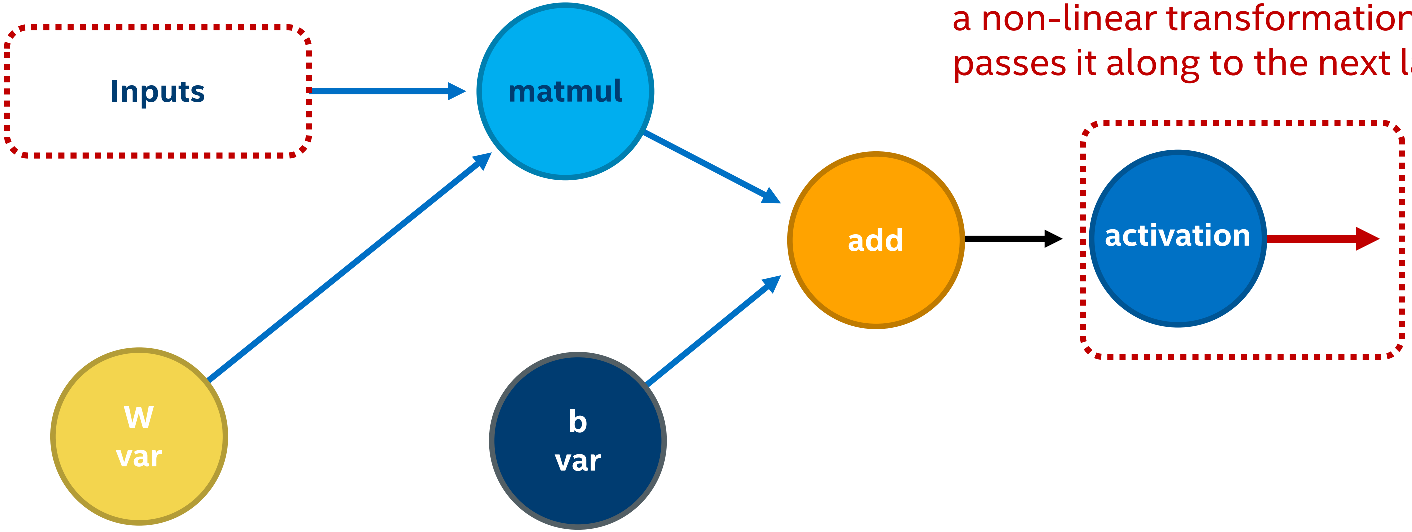
- Sigmoid function
  - Smooth transition in output between (0,1)
- Tanh function
  - Smooth transition in output between (-1,1)
- ReLU function
  - f(x) = max(x,0)
- Step function
  - f(x) = (0,1)

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

TENSORFLOW* NEURON EXAMPLE:

`TensorNeuronBasic.ipynb`

INTEL® TENSORFLOW* OPTIMIZATION

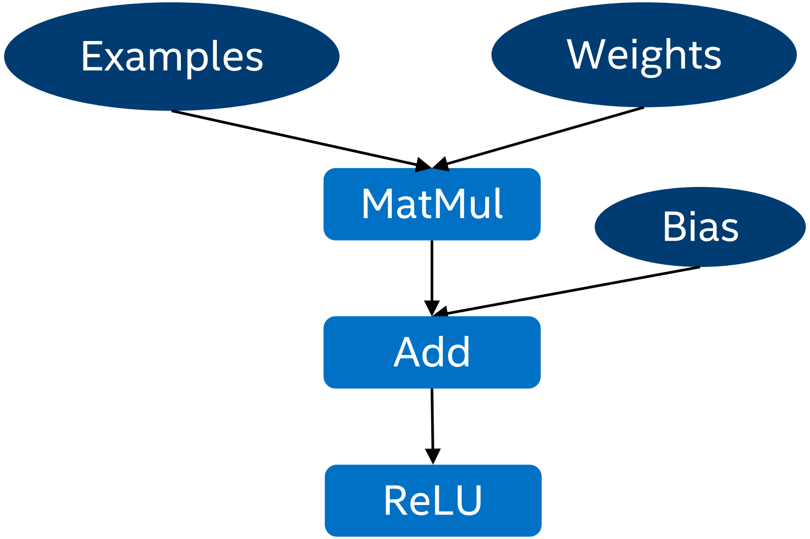
INTEL® TENSORFLOW* OPTIMIZATIONS

1. Operator optimizations: Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)
2. Graph optimizations: Fusion, Layout Propagation
3. System optimizations: Threading model

BEFORE GIVING MORE DETAILS RUN: TENSORFLOW* BENCHMARK

OPERATOR OPTIMIZATIONS

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



OPERATOR OPTIMIZATIONS

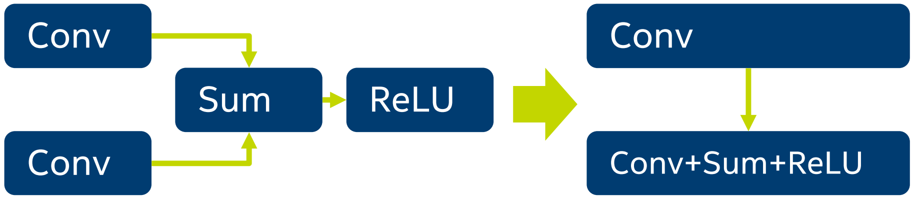
Replace default (Eigen) kernels by highly-optimized kernels (using Intel® MKL-DNN)

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	

FUSING COMPUTATIONS



On Intel processors a high % of time is typically spent in BW-limited ops

- ~40% of ResNet-50, even higher for inference

The solution is to fuse BW-limited ops with convolutions or one with another to reduce the # of memory accesses

- Conv+ReLU+Sum, BatchNorm+ReLU, etc
- Done for inference, WIP for training

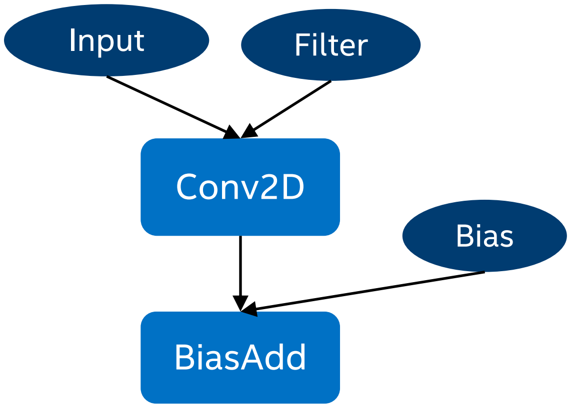
The FWKs are expected to be able to detect fusion opportunities

- IntelCaffe already supports this

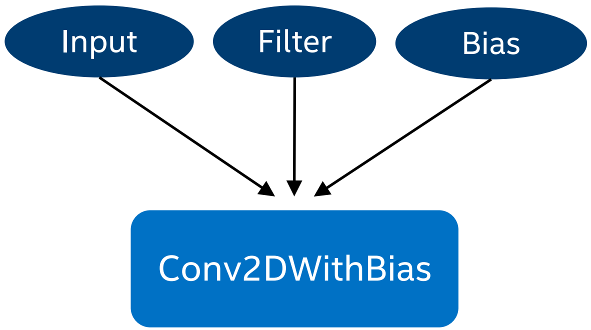
Major impact on implementation

- All the impls. must be made aware of the fusion to get max performance
- Intel MKL-DNN team is looking for scalable solutions to this problem

GRAPH OPTIMIZATIONS: FUSION

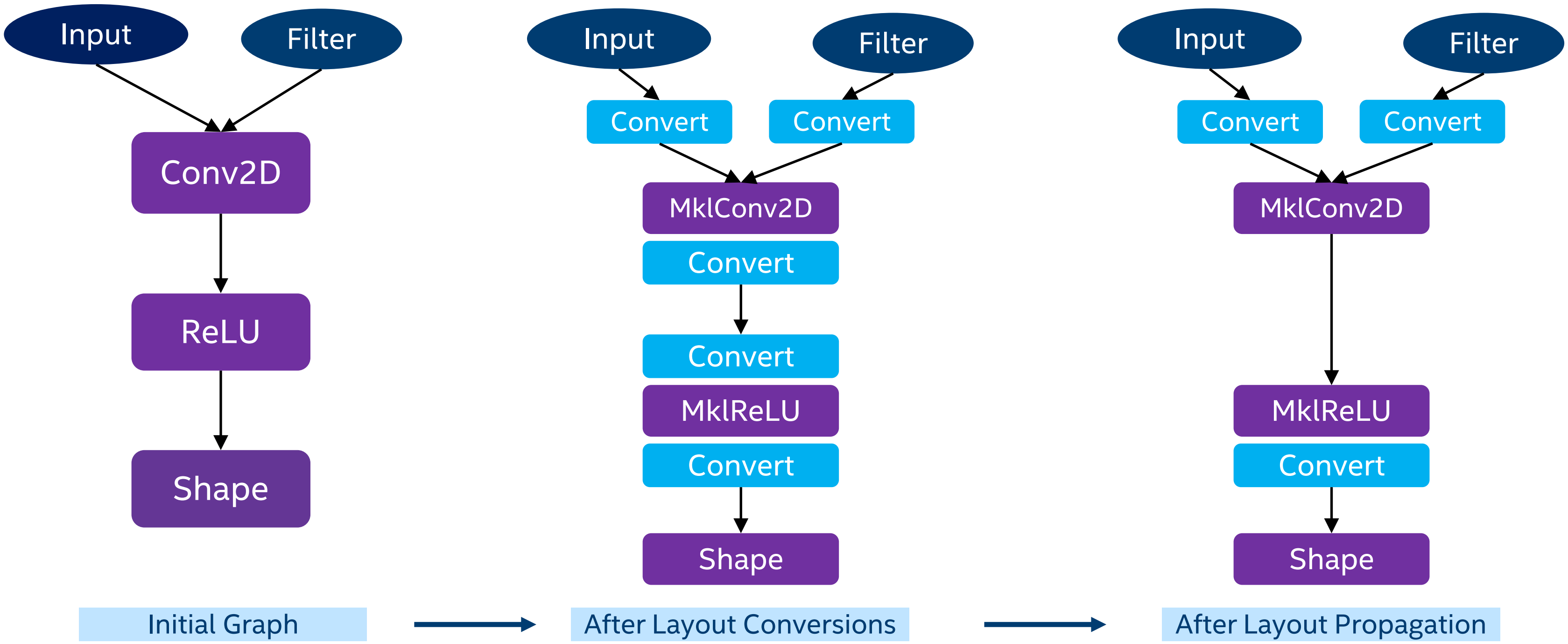


Before Merge



After Merge

GRAPH OPTIMIZATIONS: LAYOUT PROPAGATION



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

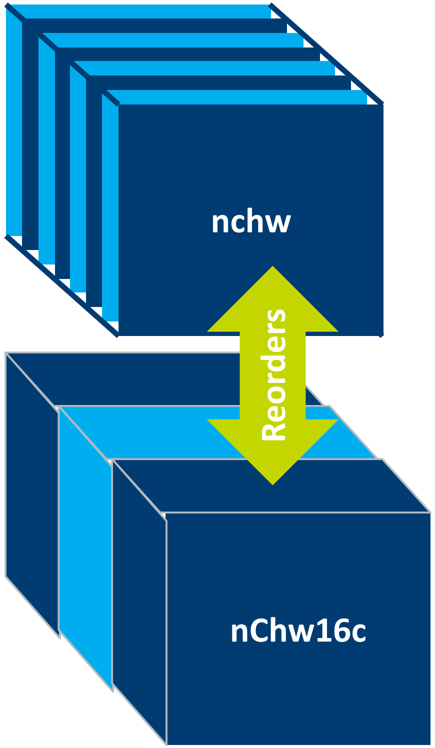
MORE ON MEMORY CHANNELS: MEMORY LAYOUTS

Most popular memory layouts for image recognition are **nhwc** and **nchw**

- Challenging for Intel processors either for vectorization or for memory accesses (cache thrashing)

Intel MKL-DNN convolutions use blocked layouts

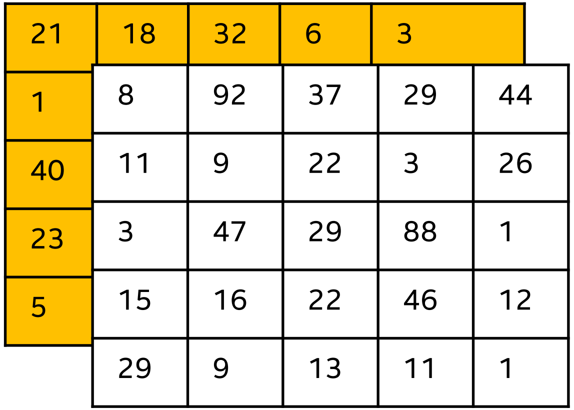
- Example: **nhwc** with channels blocked by 16 – **nChw16c**
- Convolutions define which layouts are to be used by other primitives
- Optimized frameworks track memory layouts and perform reorders **only** when necessary



DATA LAYOUT HAS A BIG IMPACT

- Continuous access to avoid gather/scatter
- Have iterations in inner most loop to ensure high vector utilization
- Maximize data reuse; e.g. weights in a convolution layer

Overhead of layout conversion is sometimes negligible, compared with operating on unoptimized layout



21	18	…	1	..	8	92	..

Channel based (NCHW)

21	8	18	92	..	1	11	..

Pixel based (NHWC)

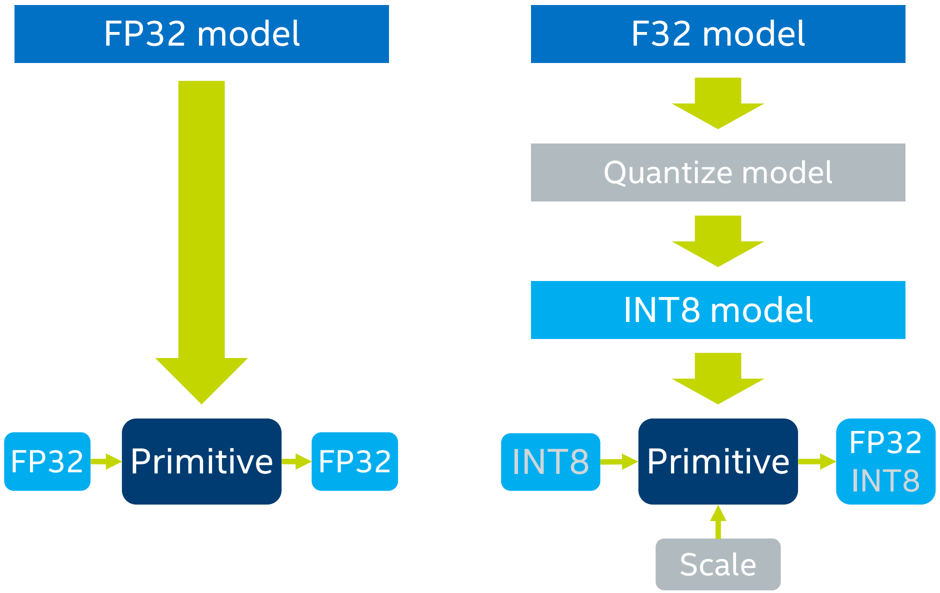
```
for i= 1 to N # batch size
    for j = 1 to C # number of channels, image RGB = 3 channels
        for k  = 1 to H  # height
            for l = 1 to W # width
                dot_product( …)
```

LOW-PRECISION INFERENCE

Proven only for certain CNNs by IntelCaffe at the moment

A trained float32 model quantized to int8

Some operations still run in float32 to preserve accuracy

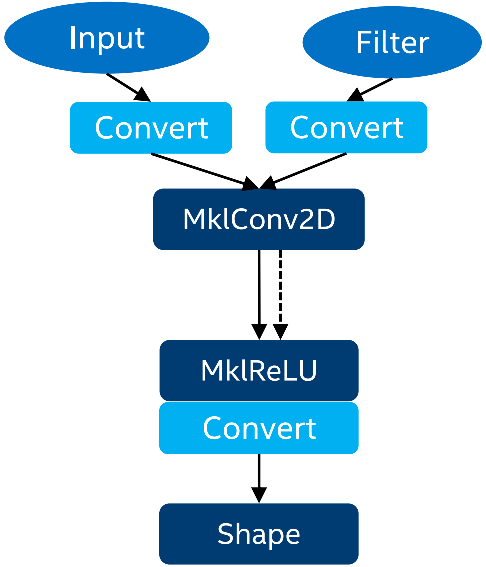


SYSTEM OPTIMIZATIONS: LOAD BALANCING

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`**

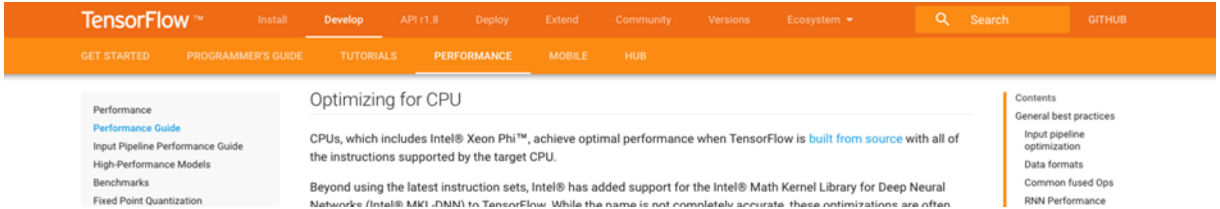


PERFORMANCE GUIDE

`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 = 56
>>> config.inter_op_parallelism_threads = 2
>>> tf.Session(config=config)
```

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



PERFORMANCE GUIDE

Maximize TensorFlow* Performance on CPU: Considerations and Recommendations for Inference Workloads: https://software.intel.com/en-us/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference

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"] = “56"
```

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.

Intel Tensorflow* install guide is available → https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide

BENCHMARK RESULT

BENCHMARKS/TF_BENCH.SH

```
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: 3.7 +/- 0.0 (jitter = 0.0)  7.780
10      images/sec: 3.8 +/- 0.0 (jitter = 0.1)  7.877
20      images/sec: 3.9 +/- 0.0 (jitter = 0.1)  7.744
30      images/sec: 3.8 +/- 0.0 (jitter = 0.1)  7.672
----------------------------------------------------------------
total images/sec: 3.84
----------------------------------------------------------------
```

Tensorflow*

```
Generating training model
Initializing graph
Running warm up
Done warm up
Step    Img/sec total_loss
1       images/sec: 17.3 +/- 0.0 (jitter = 0.0) 7.993
10      images/sec: 17.6 +/- 0.1 (jitter = 0.4) 7.854
20      images/sec: 17.6 +/- 0.1 (jitter = 0.5) 7.726
30      images/sec: 17.7 +/- 0.1 (jitter = 0.4) 7.360
----------------------------------------------------------------
total images/sec: 17.69
----------------------------------------------------------------
```

Tensorflow* with Intel® MKL-DNN

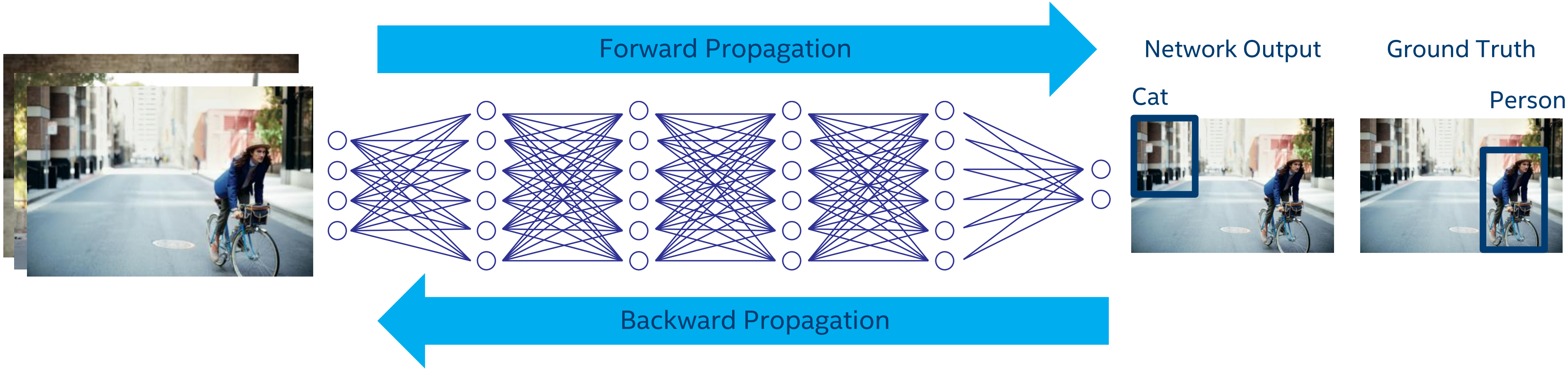
```
######### Executive Summary #########


Environment |  Network   | Batch Size | Images/Second
--------------------------------------------------------
Default     | resnet50 |     16     | 3.84
Optimized   | resnet50 |     16     | 17.69


#############################################
Average Intel Optimized speedup = 5X
#############################################
```

CONVOLUTIONAL NEURAL NETWORK (CNN)

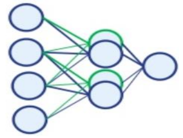
DEEP LEARNING TRAINING

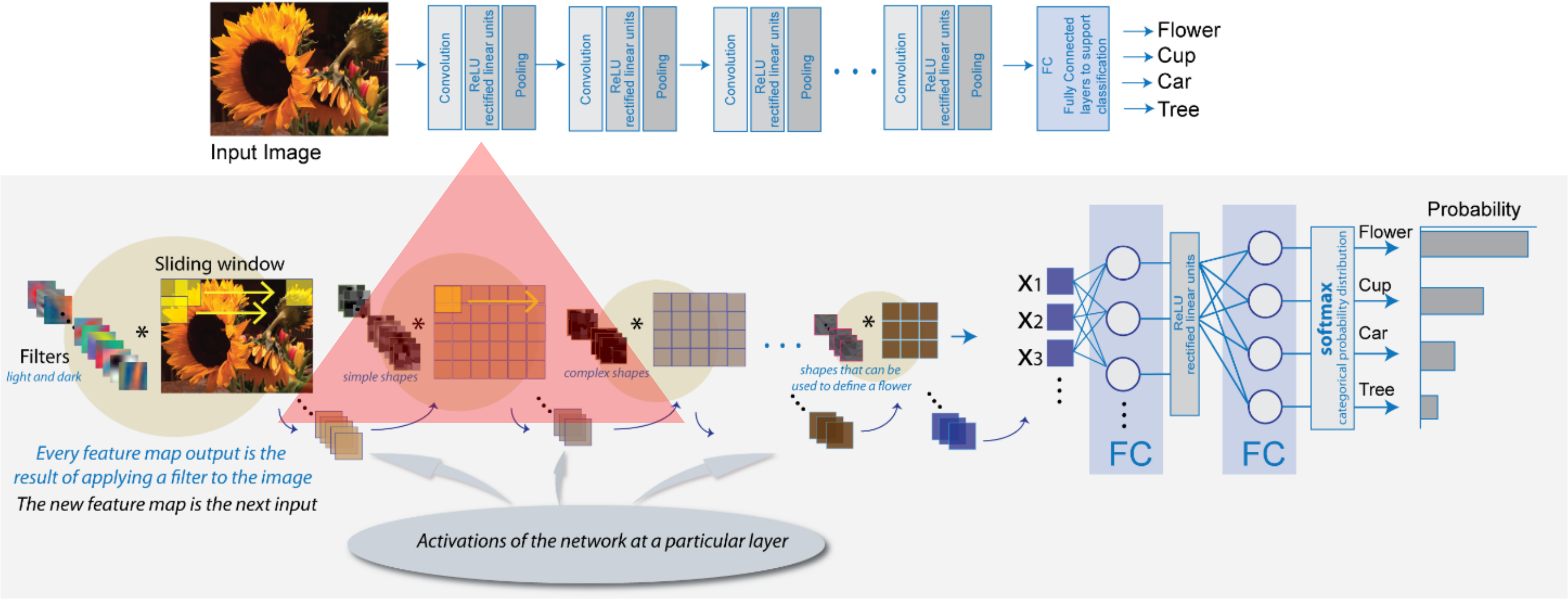


Complex Networks with billions of parameters can take days to train on a modern processor

Hence, the need to reduce time-to-train using a cluster of processing nodes

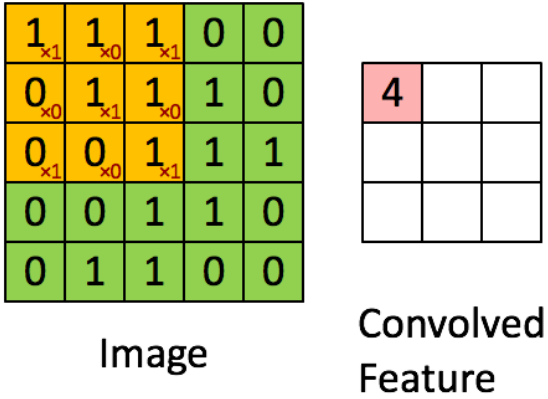
CONVOLUTION NEURAL NETWORK LAYERS

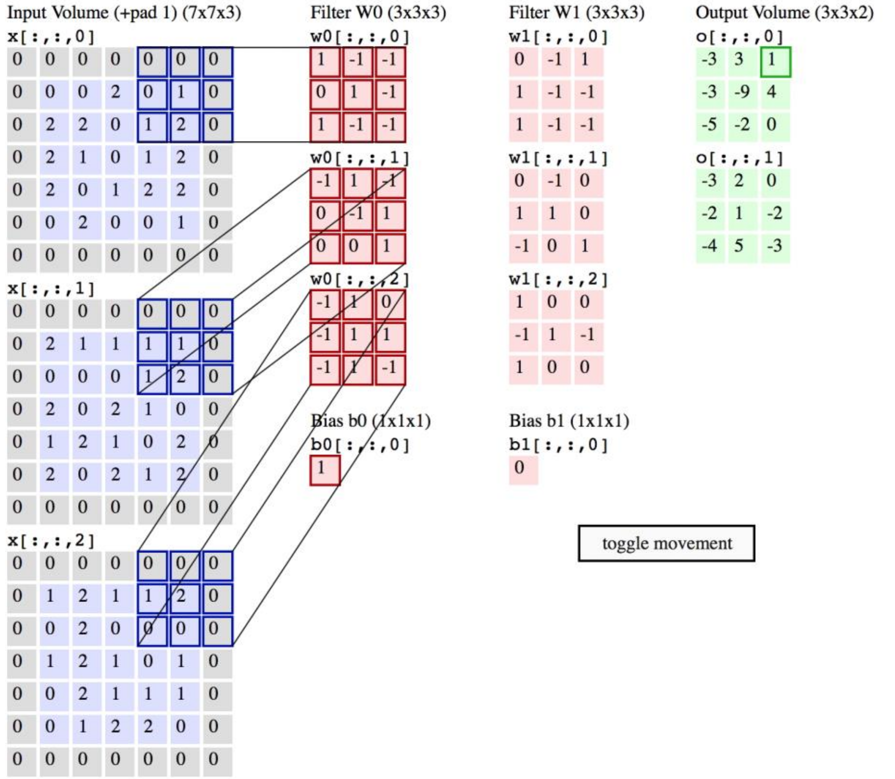




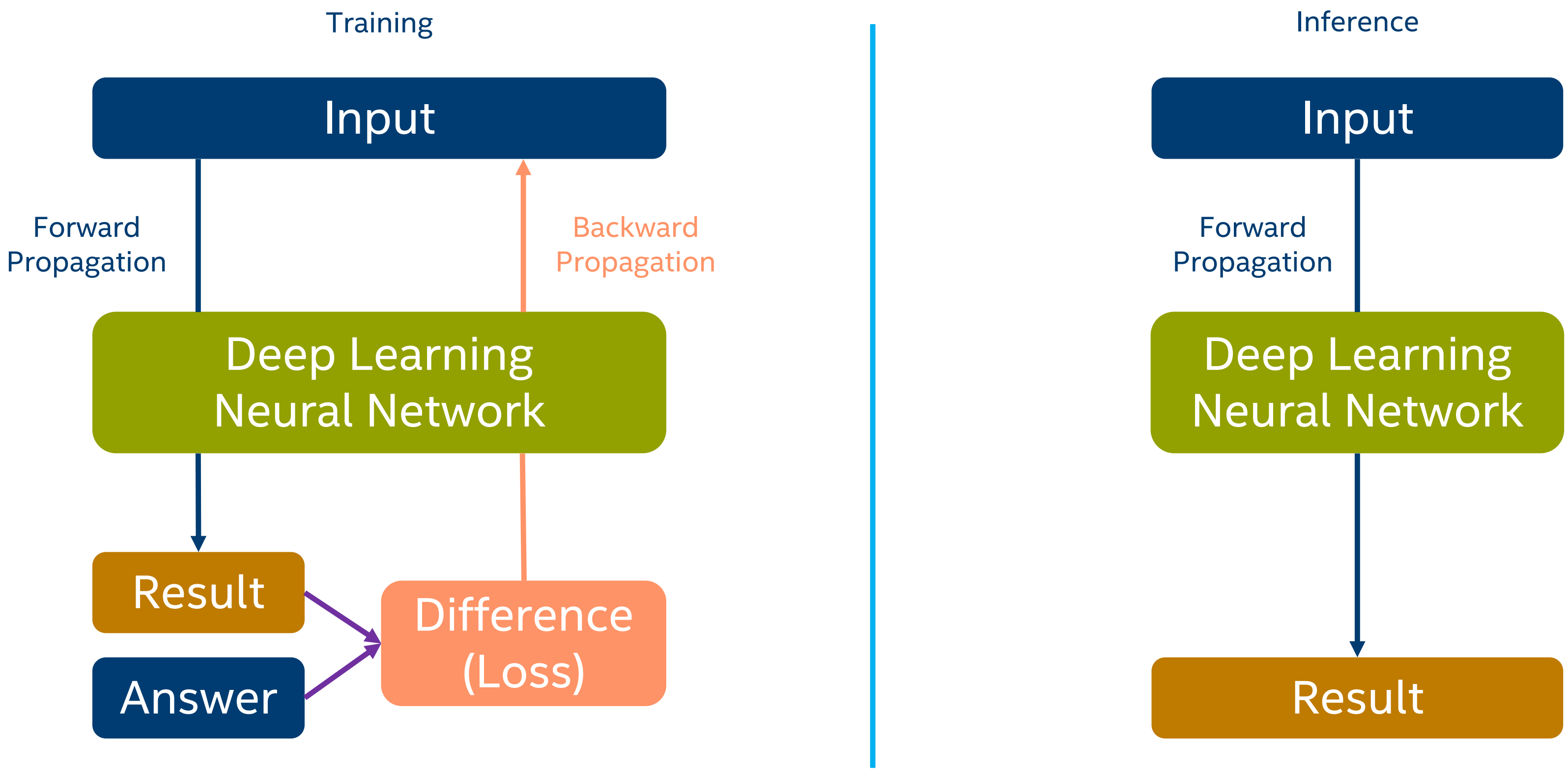
https://www.mathworks.com/help/nnet/ug/introduction-to-convolutional-neural-networks.html

CONVOLUTION = MULTIPLY - ADD OP.



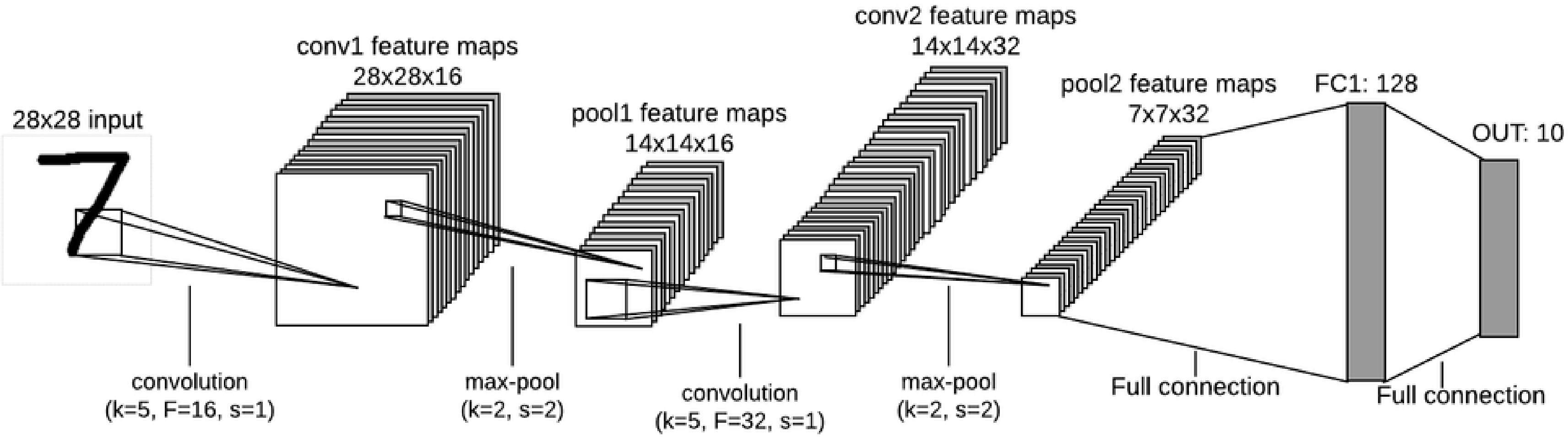


DEEP LEARNING TRAINING & INFERENCE



HANDS-ON

IMAGE CLASSIFICATION OF THE MNIST DATASET



Source:https://www.easy-tensorflow.com/tf-tutorials/convolutional-neural-nets-cnns

- Implementation of a simple Convolutional Neural Network in TensorFlow with two convolutional layers, followed by two fully-connected layers at the end

TENSORFLOW: CNN_MNIST.IPYNB

Let's try to run this example a observe the performance

Standard Python and Tensorflow installation

- `source activate python-3.6`
- `pip show tensorflow | grep Location`
  - `useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,`
- `rm -rf mnist_convnet_model/*`
- `Run the sample: time python cnn_mnist.py`

Intel Python and Optimized Tensorflow

- `source activate intel-py`
- `pip show tensorflow | grep Location`
  - `useful to locate the TF installation for see the library linked: ldd $Location/tensorflow,`
- `rm-rf mnist_convnet_model/*`
- `export export MKLDNN VERBOSE=1`

Legal Disclaimer & Optimization Notice

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 www.intel.com/benchmarks.

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

BACKUP

intel®

Software