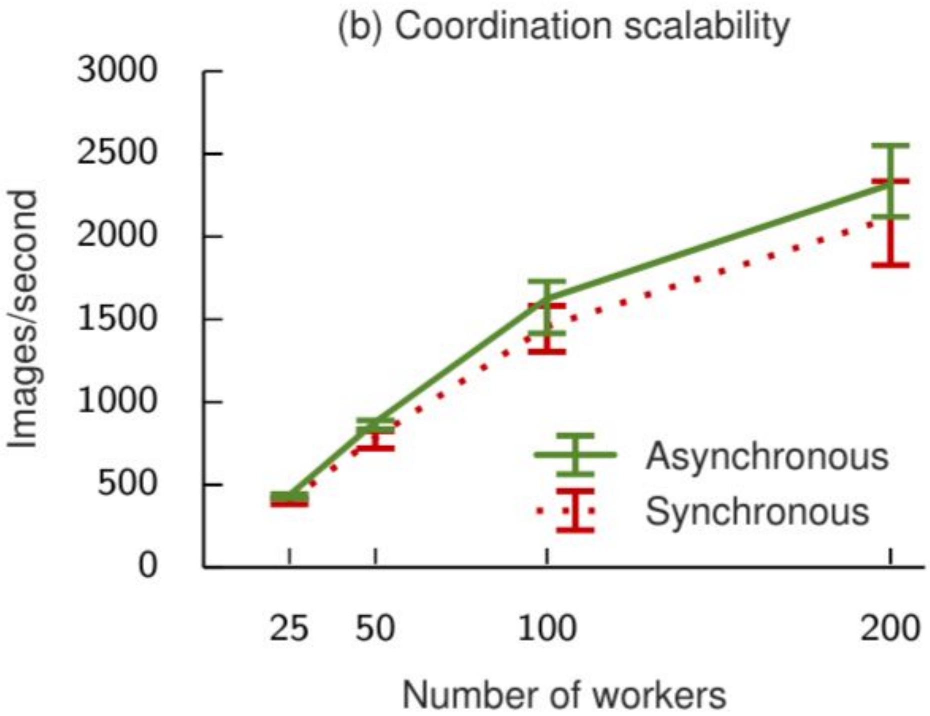
GPU & distributed computing with Tensorflow

David Parks

Distributed machine learning – challenges and approaches

Params Raman

Goal



Scaling processing up to hundreds of GPUs in tensorflow relatively easily

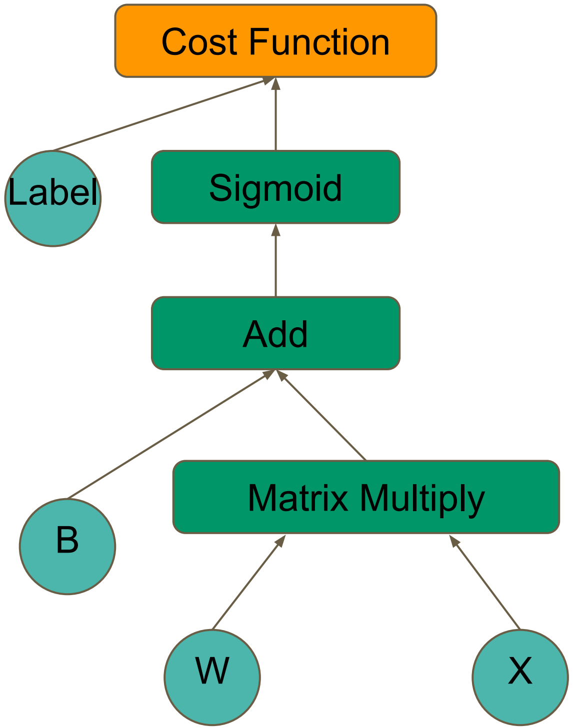
https://www.youtube.com/watch?v=la_M6bCV91M

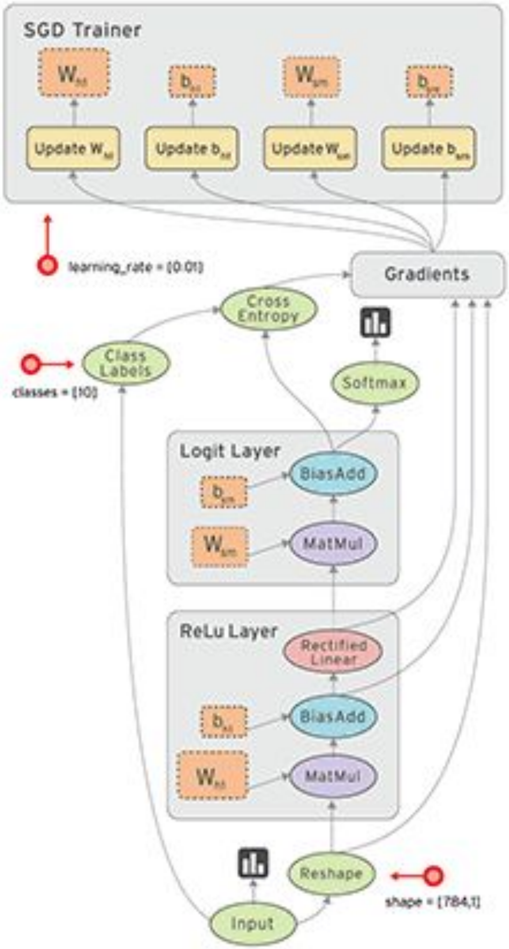
Introducing Tensorflow

- Open sourced, Nov 9th 2015
- Supports optimized multicore CPU and GPU computation (MKL, AVX Extensions)
- Built using C/C++ using OpenMP for parallelism
- Cross platform, cross language, cross device
- Based on data flow graphs

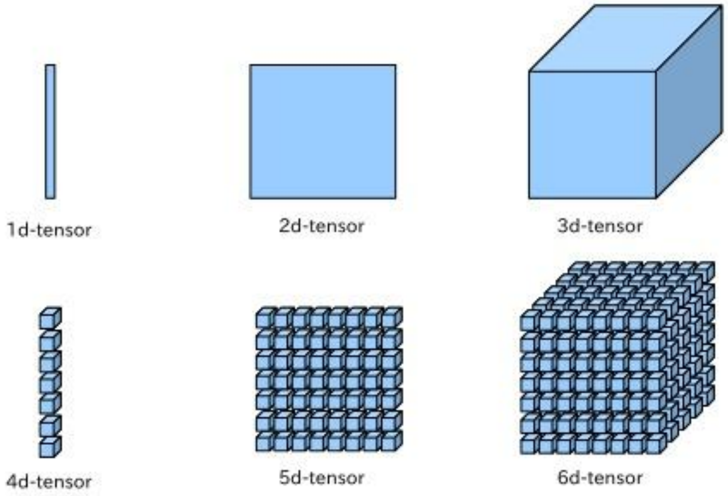
https://www.tensorflow.org

Data flow graphs





What is a “Tensor”



Rank of a tensor	Math Entity	Example
0	Scalar	`x = 42`
1	Vector	`z = [10, 15, 20]`
2	Matrix	`a = [[1 0 2 3], [2 1 0 4], [0 2 1 1]]`
3	3-Tensor (a cube of numbers)	A single image of shape `[height, width, color_channels]` example: `[1080, 1920, 3]`
4	4-Tensor (a set of cubes)	A batch of n images with shape `[batch_size, height, width, channels]` example: `[10, 1080, 1920, 3]`
n	n-dimensional Tensor	You get the idea...

The most common error in tensorflow:

```
ValueError: Shape must be rank 2 but is rank 3 for 'MatMul'
(op: 'MatMul') with input shapes: [10,512,1], [512,2]
```

https://www.slideshare.net/yokotatsuya/principal-component-analysis-for-tensor-analysis-and-eeg-classification

Basic workflow for Tensorflow

1 Build a computation Graph

```
import tensorflow as tf

a = tf.constant(2.0, tf.float32, name='a')
b = tf.constant(3.0, tf.float32, name='b')

c = tf.multiply(a, b)
```

```
<tf.Tensor 'a:0' shape=() dtype=float32>
<tf.Tensor 'b:0' shape=() dtype=float32>

<tf.Tensor 'Mul_2:0' shape=() dtype=float32>
```

2 Run computations on the graph

```
sess = tf.Session()
result = sess.run([b, c])
```

```
print(result[0])
3.0

print(result[1])
6.0
```

Passing parameters to tensorflow

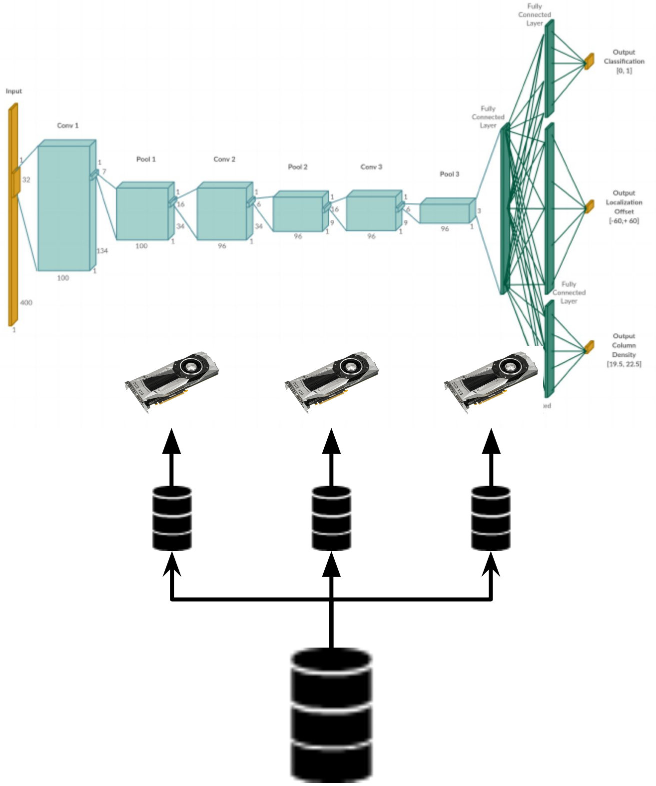
```
import tensorflow as tf

# Build your graph
a = tf.placeholder(tf.float32, shape=(),  name='a')   # A scalar
b = tf.placeholder(tf.float32, shape=(2), name='b')   # A vector
c = tf.multiply(a, b, name='c')

# Launch a session & evaluate tensor c
with tf.Session() as sess:
  params = {a: 10, b: [1, 2]}   # a & b are tensor objects
  result = sess.run([c], feed_dict=params)
```

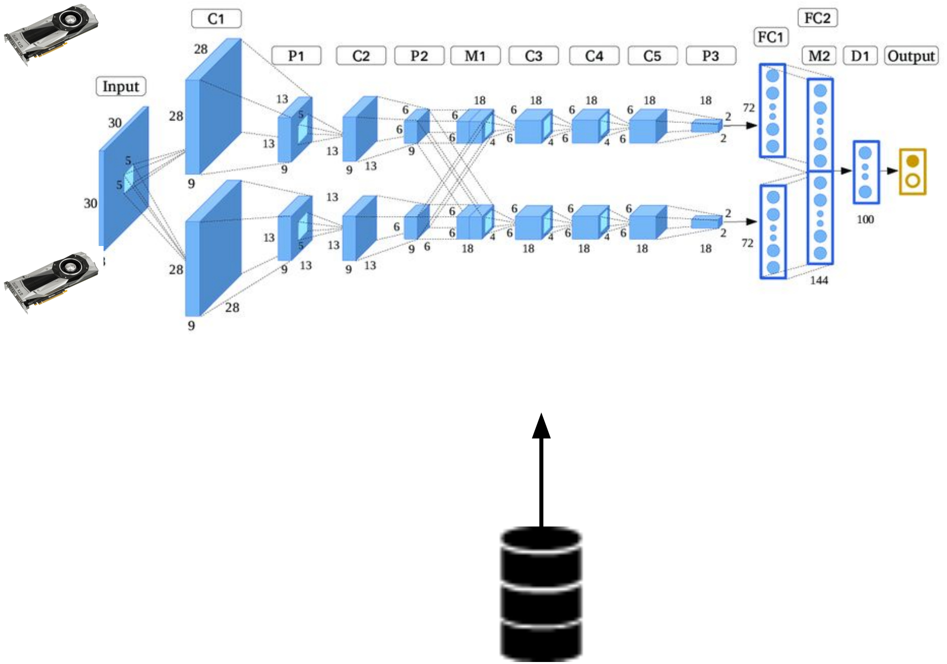
```
>>> print result
[array([ 10.,  20.], dtype=float32)]
```

Best practices note:

Structure your code with a `build_graph(...)` function which separates tensorflow graph operations from iterating through a dataset with calls to `sess.run(...)`

Data and model parallelism



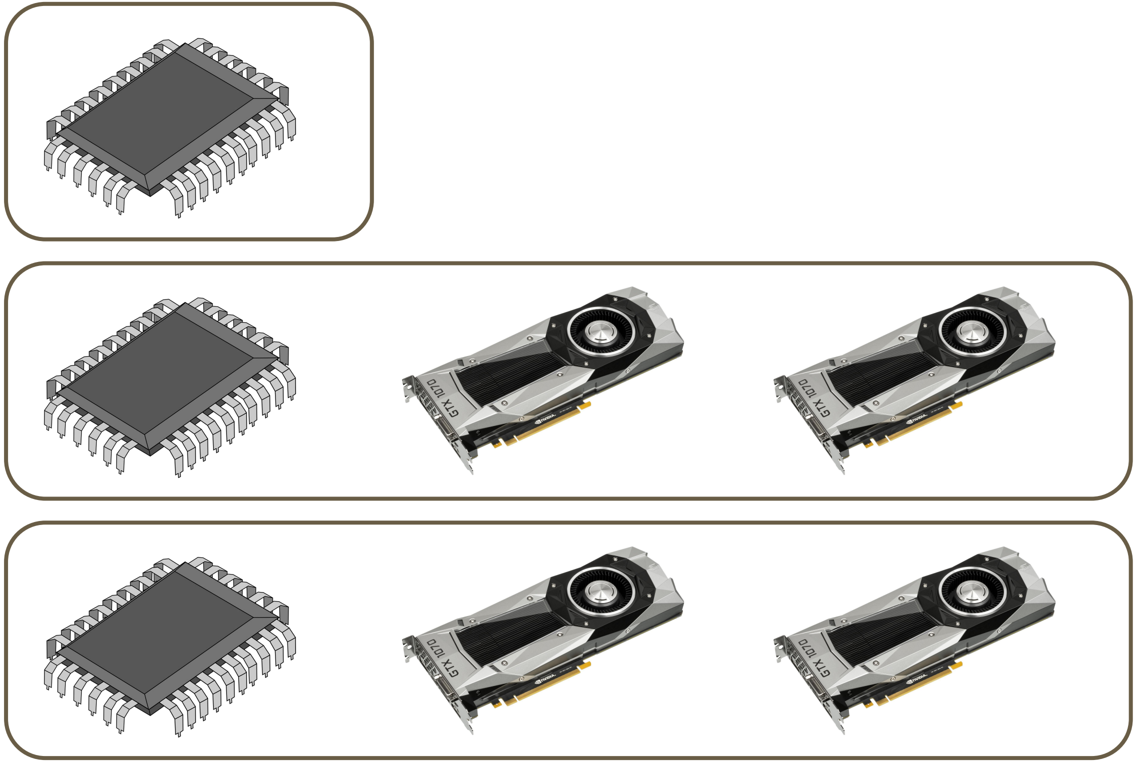


https://www.researchgate.net/figure/AlexNet-like-architecture_fig4_320723863
Icon made by Freepik from www.flaticon.com

Parameter server and worker

Parameter Server(s)
Typical a CPU only node that
maintains cluster wide variables

Worker(s)
CPU or GPU nodes which perform the
primary computation



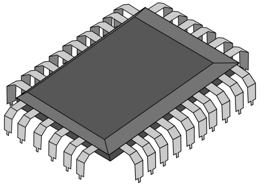
Distributed tensorflow single server multiple devices

```python
with tf.device('/cpu:0'):
  W = tf.Variable(...)
  B = tf.Variable(...)

with tf.device('/gpu:0'):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Tensorflow handles the DMA transfer between devices

/job:worker/task:0/

cpu:0



gpu:0

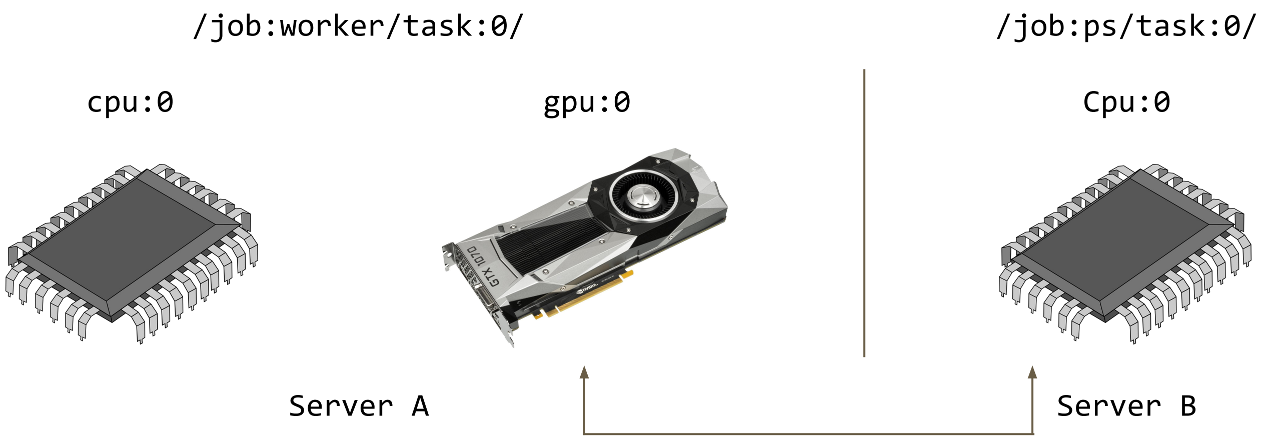
gpu:1

Server A

Distributed tensorflow multiple servers

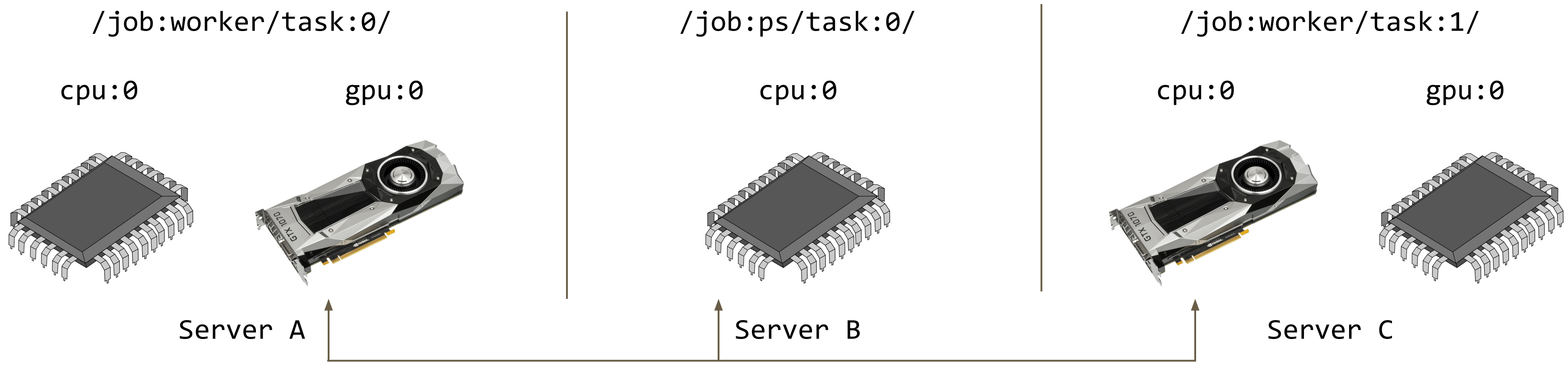
```python
with tf.device('/job:ps/task:0/cpu:0'):
  W = tf.Variable(...)
  B = tf.Variable(...)

with tf.device('/job:worker/task:0/gpu:0'):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

Tensorflow uses GRPC to communicate between servers



In-graph replication easier, smaller scale (single server)

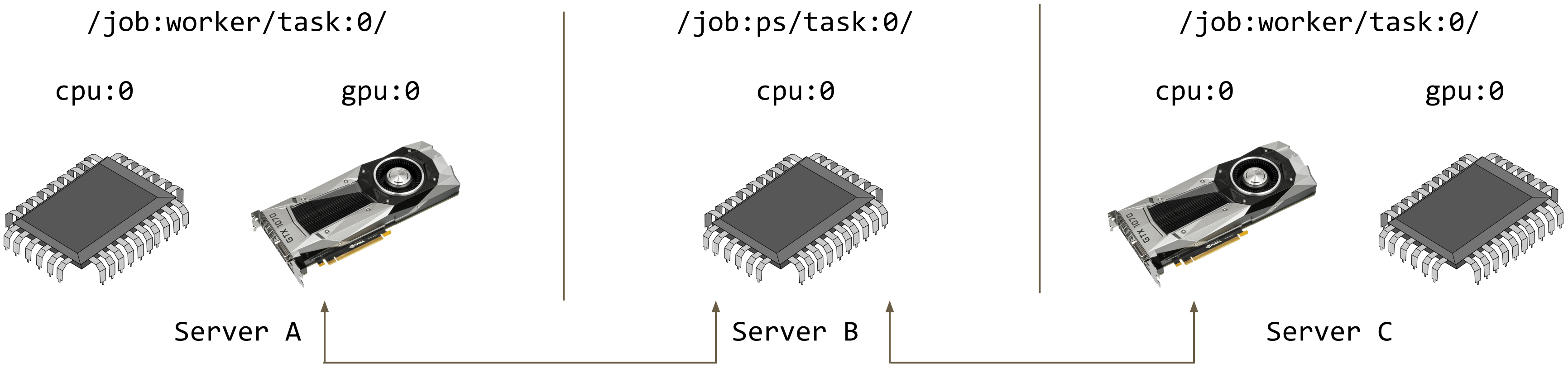
```
with tf.device('/job:ps/task:0/cpu:0'):
  W = tf.Variable(...)
  B = tf.Variable(...)

inputs = tf.split(0, num_workers, input)
Outputs = []

for i in range(num_worker):
  with tf.device('/job:worker/task:%d/gpu:0' % i):
    outputs.append(tf.matmul(input[i], W) + b)
loss = f(output)
```



Between-graph replication

```python
with tf.device('/job:ps/task:0/cpu:0'):
  W = tf.Variable(...)
  B = tf.Variable(...)

with tf.device('/job:worker/task:0/gpu:0'):
  output = tf.matmul(input, W) + b
  loss = f(output)
```

```python
with tf.device('/job:ps/task:0/cpu:0'):
  W = tf.Variable(...)
  B = tf.Variable(...)

with tf.device('/job:worker/task:1/gpu:0'):
  output = tf.matmul(input, W) + b
  loss = f(output)
```



Creating a tensorflow cluster

```
# Distributed code for a worker task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="worker", task_index=0)

with tf.Session(server.target) as sess:
  # …
```

Creating the parameter server

```
# Distributed code for a PS task.
cluster = tf.train.ClusterSpec({"worker": ["192.168.0.1:2222", …],
                                "ps": ["192.168.1.1:2222", …]})

server = tf.train.Server(cluster, job_name="ps", task_index=0)

# Wait for incoming connections forever.
server.join()
```

CPU Optimizations

Build Tensorflow from sources

Supports MKL and MKL-DNN libraries, requires compiling from sources

Batch Size: 1

Command executed for the MKL test:

```
python tf_cnn_benchmarks.py --forward_only=True --device=cpu --mkl=True \
--kmp_blocktime=0 --nodistortions --model=inception3 --data_format=NCHW \
--batch_size=1 --num_inter_threads=1 --num_intra_threads=4 \
--data_dir=<path to ImageNet TFRecords>
```

Optimization	Data Format	Images/Sec (step time)	Intra threads	Inter Threads
AVX2	NHWC	7.0 (142ms)	4	0
MKL	NCHW	6.6 (152ms)	4	1
AVX	NHWC	5.0 (202ms)	4	0
SSE3	NHWC	2.8 (361ms)	4	0

Batch Size: 32

Command executed for the MKL test:

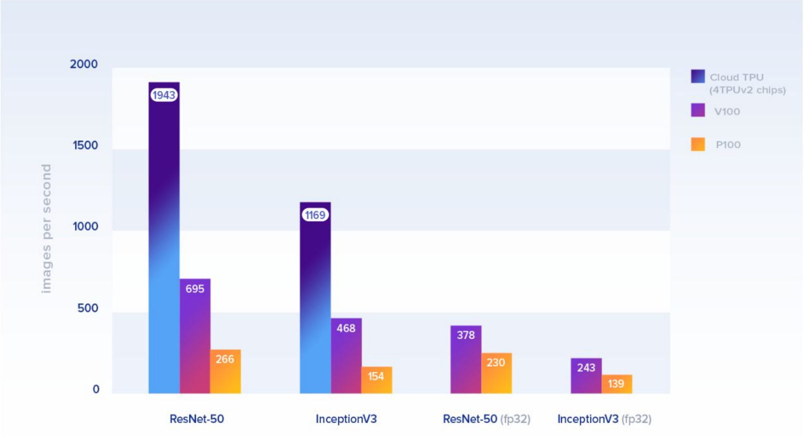
```
python tf_cnn_benchmarks.py --forward_only=True --device=cpu --mkl=True \
--kmp_blocktime=0 --nodistortions --model=inception3 --data_format=NCHW \
--batch_size=32 --num_inter_threads=1 --num_intra_threads=4 \
--data_dir=<path to ImageNet TFRecords>
```

Optimization	Data Format	Images/Sec (step time)	Intra threads	Inter Threads
MKL	NCHW	10.3 (3,104ms)	4	1
AVX2	NHWC	7.5 (4,255ms)	4	0
AVX	NHWC	5.1 (6,275ms)	4	0
SSE3	NHWC	2.8 (11,428ms)	4	0

GPUs and TPUs

- CUDA & CuDNN
- Use large matrix operations
- Use 32 bit floating point operations if possible

RiseML Blog



ResNet-50 Performance

	TPUv2	V100 fp16	P100	V100
Cloud	Google	AWS	Google	AWS
Price *$ per hour*	7.26	3.06	1.58	3.06
images / second	1943	695	230	378
Performance *images/s per $*	268	227	146	124

https://blog.riseml.com/benchmarking-googles-new-tpuv2-121c03b71384

Working with GPUs

Get the state of the GPU(s)

nvidia-smi

Specify GPU(s) to use:

Use GPU 0:

export CUDA_VISIBLE_DEVICES=0

Use GPU 0 and 1:

export CUDA_VISIBLE_DEVICES=0,1

CPU only:

export CUDA_VISIBLE_DEVICES=-1

```
[dfparksucscedu@patternlab ~]$ nvidia-smi
Sat Oct 28 16:45:12 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 367.55                 Driver Version: 367.55                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M40 24GB      Off  | 0000:85:00.0     Off |                    0 |
| N/A   31C    P0    59W / 250W |      0MiB / 22939MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M40 24GB      Off  | 0000:8D:00.0     Off |                    0 |
| N/A   29C    P0    58W / 250W |      0MiB / 22939MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
[dfparksucscedu@patternlab ~]$
```

Installing Tensorflow

Start with **Anaconda**, both python 2 and 3 are supported

All major python libraries are included: `Numpy, Matplotlib, Jupyter Notebook, etc.`

Then:

```
pip install tensorflow
```

or

```
pip install tensorflow-gpu
```

University GPU resources:

- **`citrisdance.soe.ucsc.edu`**
    - 2 older K20 GPUs and 32 cores storage is an issue
- **`https://patternlab.calit2.optiputer.net/`**
    - 2 fast M40 GPUs, 40 cores, shared with UCSD, 60+TB of storage
- More coming...

Demos

https://github.com/aymericdamien/TensorFlow-Examples

Tutorial index

0 - Prerequisite

- Introduction to Machine Learning.
- Introduction to MNIST Dataset.

1 - Introduction

- **Hello World** (notebook) (code). Very simple example to learn how to print "hello world" using TensorFlow.
- **Basic Operations** (notebook) (code). A simple example that cover TensorFlow basic operations.

2 - Basic Models

- **Linear Regression** (notebook) (code). Implement a Linear Regression with TensorFlow.
- **Logistic Regression** (notebook) (code). Implement a Logistic Regression with TensorFlow.
- **Nearest Neighbor** (notebook) (code). Implement Nearest Neighbor algorithm with TensorFlow.
- **K-Means** (notebook) (code). Build a K-Means classifier with TensorFlow.
- **Random Forest** (notebook) (code). Build a Random Forest classifier with TensorFlow.

3 - Neural Networks

Supervised

- **Simple Neural Network** (notebook) (code). Build a simple neural network (a.k.a Multi-layer Perceptron) to classify MNIST digits dataset. Raw TensorFlow implementation.
- **Simple Neural Network (tf.layers/estimator api)** (notebook) (code). Use TensorFlow 'layers' and 'estimator' API to build a simple neural network (a.k.a Multi-layer Perceptron) to classify MNIST digits dataset.
- **Convolutional Neural Network** (notebook) (code). Build a convolutional neural network to classify MNIST digits dataset. Raw TensorFlow implementation.
- **Convolutional Neural Network (tf.layers/estimator api)** (notebook) (code). Use TensorFlow 'layers' and 'estimator' API to build a convolutional neural network to classify MNIST digits dataset.
- **Recurrent Neural Network (LSTM)** (notebook) (code). Build a recurrent neural network (LSTM) to classify MNIST digits dataset.
- **Bi-directional Recurrent Neural Network (LSTM)** (notebook) (code). Build a bi-directional recurrent neural network (LSTM) to classify MNIST digits dataset.
- **Dynamic Recurrent Neural Network (LSTM)** (notebook) (code). Build a recurrent neural network (LSTM) that performs dynamic calculation to classify sequences of different length.

Unsupervised

- **Auto-Encoder** (notebook) (code). Build an auto-encoder to encode an image to a lower dimension and re-construct it.
- **Variational Auto-Encoder** (notebook) (code). Build a variational auto-encoder (VAE), to encode and generate images from noise.
- **GAN (Generative Adversarial Networks)** (notebook) (code). Build a Generative Adversarial Network (GAN) to generate images from noise.
- **DCGAN (Deep Convolutional Generative Adversarial Networks)** (notebook) (code). Build a Deep Convolutional Generative Adversarial Network (DCGAN) to generate images from noise.

4 - Utilities

- **Save and Restore a model** (notebook) (code). Save and Restore a model with TensorFlow.
- **Tensorboard - Graph and loss visualization** (notebook) (code). Use Tensorboard to visualize the computation Graph and plot the loss.
- **Tensorboard - Advanced visualization** (notebook) (code). Going deeper into Tensorboard; visualize the variables, gradients, and more...

5 - Data Management

- **Build an image dataset** (notebook) (code). Build your own images dataset with TensorFlow data queues, from image folders or a dataset file.
- **TensorFlow Dataset API** (notebook) (code). Introducing TensorFlow Dataset API for optimizing the input data pipeline.

6 - Multi GPU

- **Basic Operations on multi-GPU** (notebook) (code). A simple example to introduce multi-GPU in TensorFlow.
- **Train a Neural Network on multi-GPU** (notebook) (code). A clear and simple TensorFlow implementation to train a convolutional neural network on multiple GPUs.

Other resources

- Distributed Tensorflow (TF Dev Summit 2017) video presentation by Mrry
  - https://www.youtube.com/watch?v=la_M6bCV91M
- TensorFlow: A System for Large-Scale Machine Learning
  - https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf
- Tensorflow performance guide
  - https://www.tensorflow.org/performance/performance_guide
- Tensorflow high performance models
  - https://www.tensorflow.org/performance/performance_models