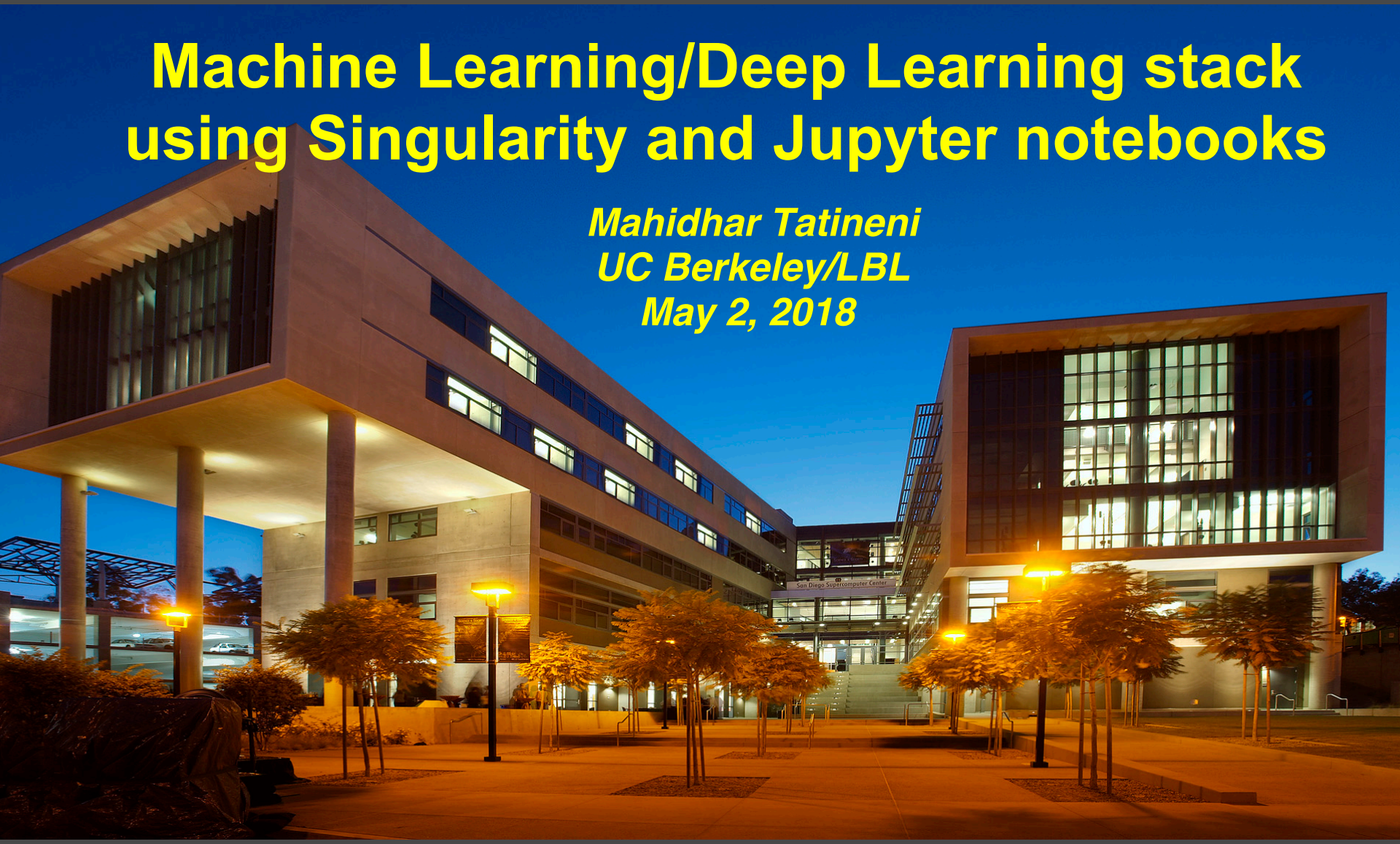
Machine Learning/Deep Learning stack using Singularity and Jupyter notebooks

Mahidhar Tatineni
UC Berkeley/LBL
May 2, 2018



Machine Learning/Deep Learning on Comet via Singularity

- Bulk of the Singularity usage on Comet is for machine learning/deep learning applications.
- Lot of these packages are constantly upgraded and the dependency list is difficult to update in the standard Comet environment.
- Install options
  - Singularity image provides dependencies and user can compile actual application from source. e.g. Torch
  - Entire dependency stack and the application is in the image. e.g Keras with backend to TensorFlow and some additional python libraries
- Run options
  - Most cases are run on single GPU nodes (4 GPUs at most)
  - Can access this via Jupyter notebooks
  - Multi-node options are possible but difficult to set up via singularity.

Tensorflow via Singularity (Single Node)

```
#!/bin/bash
#SBATCH --job-name="TensorFlow"
#SBATCH --output="TensorFlow.%j.%N.out"
#SBATCH --partition=gpu-shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH -t 01:00:00

#Run the job
#

module load singularity
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img lsb_relea
se -a
singularity exec /share/apps/gpu/singularity/sdsc_ubuntu_gpu_tflow.img python -m
 tensorflow.models.image.mnist.convolutional
```

Tensorflow via Singularity

- **Change to the examples directory:**
  cd /home/$USER/UCB2018/TensorFlow

- **Submit the job:**
  sbatch TensorFlow.sb

Tensorflow Example: Output

```
Distributor ID: Ubuntu
Description: Ubuntu 16.04 LTS
Release: 16.04
Codename: xenial
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcublas.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcudnn.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcufft.so locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:108] successfully opened CUDA library libcurand.so locally
I tensorflow/core/common_runtime/gpu/gpu_init.cc:102] Found device 0 with properties:
name: Tesla K80
major: 3 minor: 7 memoryClockRate (GHz) 0.8235
pciBusID 0000:85:00.0
Total memory: 11.17GiB
Free memory: 11.11GiB
I tensorflow/core/common_runtime/gpu/gpu_init.cc:126] DMA: 0
I tensorflow/core/common_runtime/gpu/gpu_init.cc:136] 0:   Y
I tensorflow/core/common_runtime/gpu/gpu_device.cc:838] Creating TensorFlow device (/gpu:0) -> (device: 0, name:
Tesla K80, pci bus id: 0000:85:00.0)
Extracting data/train-images-idx3-ubyte.gz
…
Step 8500 (epoch 9.89), 11.6 ms
Minibatch loss: 1.601, learning rate: 0.006302
Minibatch error: 0.0%
Validation error: 0.9%
Test error: 0.9%
```

Accessing via Jupyter Notebook

[1] Get an interactive node:

```
srun --pty --nodes=1 --ntasks-per-node=24 -p compute --reservation=UCB2018Res -t 02:00:00 --wait 0 /bin/bash
```

[2] Load the singularity module and get an interactive shell

```
module load singularity
singularity shell /share/apps/gpu/singularity/sdsc_ubuntu_tf1.1_keras_R.img
```

[3] Launch the notebook

```
ipython notebook --no-browser --ip="*" &
```

This will give you an address which has localhost in it and a token. Something like:

http://localhost:8888/?token=389587c9d1b69f8f595e7d8bfdd83c9961ed26b8b3f1bb3e

You can replace localhost with comet-XX-YY.sdsc.edu and then paste it into your browser. That should get you into the running notebook. From there everything should be working as a regular notebook. Note: This token is your auth so don't email/send it around (I already stopped the above link).

MNIST Example

- **Keras, deep learning library in python - backend to Tensorflow or Theano. Today we will use Tensorflow.**

- **Deep learning refers to neural networks with multiple hidden layers that can learn increasingly abstract representations of the input data. For example in a image classification case**
  - first layers might learn local edge pattern
  - each subsequent layer (or filter) gets more complex reps
  - eventually the last layer can classify images - car, tree etc.

- **Convolutional Neural Networks - images are the input data - reduce the parametric space for tuning and effectively handle the high dimensionality of raw images.**

- **References:**
  - https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html

MNIST Example via Jupyter Notebook

- **Step 1: Follow the instructions for spinning up the Jupyter notebook**

- **Step 2: From the browser window, open the following notebook:**

  - LabMNIST_Final.ipynb

Multi-Node runs via Singularity

- **Easy for cases with MPI backends - we already saw this in the first talk.**
- **ML/DL frameworks can be a bit more complicated with process launches on remote nodes (need to be done via image)**
- **Two examples:**
  - Tensorflow - processes are launched once => just need to wrap the launch tasks.
  - MXNET : Lot of remote commands => need to patch the script that does this part.

Script snippet from multi-node TF

```
#Start the parameter server
cd /scratch/$USER/$SLURM_JOBID
cp $SLURM_SUBMIT_DIR/example.sh .
cp $SLURM_SUBMIT_DIR/example.py .
./example.sh "ps" 0 2>&1 > $SLURM_SUBMIT_DIR/ps.$SLURM_JOBID.log &


#Run the worker processes remotely
ssh $H2 $SLURM_SUBMIT_DIR/run_worker.sh $SLURM_SUBMIT_DIR
/scratch/$USER/$SLURM_JOBID 0 > $SLURM_SUBMIT_DIR/worker0_$SLURM_JOBID.log
&
sleep 10s

ssh $H3 $SLURM_SUBMIT_DIR/run_worker.sh $SLURM_SUBMIT_DIR
/scratch/$USER/$SLURM_JOBID 1 >
$SLURM_SUBMIT_DIR/worker1_$SLURM_JOBID.log
```

*** For the multinode case, you need to untar the tensorflow.tar file in your home directory.

MXNET

- **Remote launch controlled by:**
  - incubator-mxnet/dmlc-core/tracker/dmlc_tracker/ssh.py

- **Change made:**

```
prog = get_env(pass_envs) + ' cd ' + working_dir + ';
/opt/singularity/bin/singularity exec
/oasis/scratch/comet/username/temp_project/singularity/mxn
et-gpu.img ' + (' '.join(args.command))
```

MXNET interactive snippet

```
scontrol show hostname > hosts

for NODE in `cat hosts`; do ssh $NODE cp -r /home/username/mxnet/incubator-mxnet/example/image-classification /scratch/$USER/$SLURM_JOBID; done

cd /scratch/$USER/$SLURM_JOBID/image-classification

singularity shell /oasis/scratch/comet/username/temp_project/singularity/mxnet-gpu.img

python /home/username/mxnet/incubator-mxnet/tools/launch.py -n 2 --launcher ssh -H hosts python train_mnist.py --gpus 0,1,2,3 --network lenet --kv-store dist_sync

for NODE in `cat hosts`; do ssh $NODE killall -s 9 python ; done
exit
```

Summary

- Most ML/DL packages on Comet are enabled via Singularity
- Single node runs are easy to wrap with “singularity exec” commands
- Can launch and use Jupyter notebooks from within Singularity containers
- Multi-node runs need some work but feasible in most cases.