Speech Recognition with PyTorch and TensorFlow

Charlie, Zhiwei,Vibhu, Angelo, Giacomo

Chosen Task & Dataset

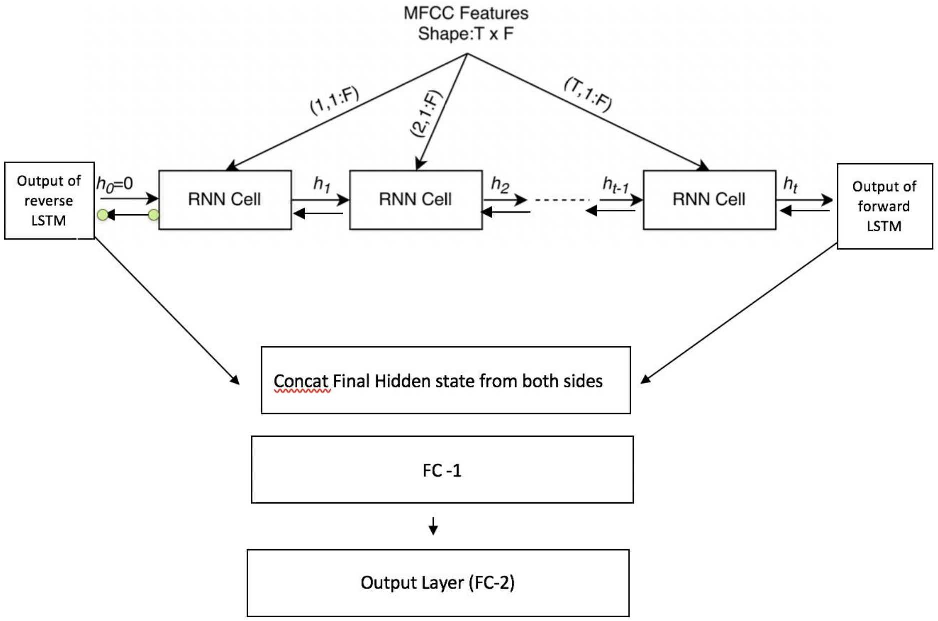
Task

- Identify verbal commands in voice data from multiple people and of varying quality

Dataset

- 65,000 1 second long utterances
- Each utterance is classified as 1 of 30 short words
- Each utterance is from 1 of thousands of different people
- Utterances include “yes”, “no”, “up”, “down”, “left”, “right”, “on”, “off”, “stop”, “go”.

Network Architecture



Number of Parameters: 8504 trainable parameters

Model memory requirement: 406 KB (Single Precision)

Keras vs PyTorch - Overview

- Both organize parameters as tensors, interpret models as DAGs

TensorFlow:

- models are static, must be defined before running (or incur serialization penalty)
- limited opportunities to "communicate" with model at runtime
- Keras

PyTorch:

- more dynamic: can add/alter nodes on the go
- highly integrated with Python

Keras vs PyTorch - Overview

Keras:

- optimized to run on multiple GPUs, TPUs
- automatically determines calculations necessary
- defaults to NHWC format, can be difficult to change

PyTorch:

- difficult/hacky to run across multiple GPUs, performance drops
- can perform necessary calculations if not manually tweaked; difficult in complex networks (ACGAN, WGAN-GP, etc.)
- NCHW, usually faster out of the box

Keras vs PyTorch - Overview

- PyTorch’s dynamic nature offers top-down support
  - seamlessly integrated, easier for rapid prototyping

- Keras/TF’s is “tacked on”
  - usually requires more hacking around to get working
  - still known limitations:
    - RNN max size, padding; other recursive neural net issues
    - no conditional branching during evaluation, cannot backpropagate across multiple runs

Keras vs PyTorch - Overview

Keras:

- Can be optimized to be faster/more efficient given enough time
- Offers high-performance deployment options (i.e. TF Serving)

PyTorch:

- Faster for rapid prototyping
- More flexible
- Easier setup out of the box

Hardware Benchmarking Experiments

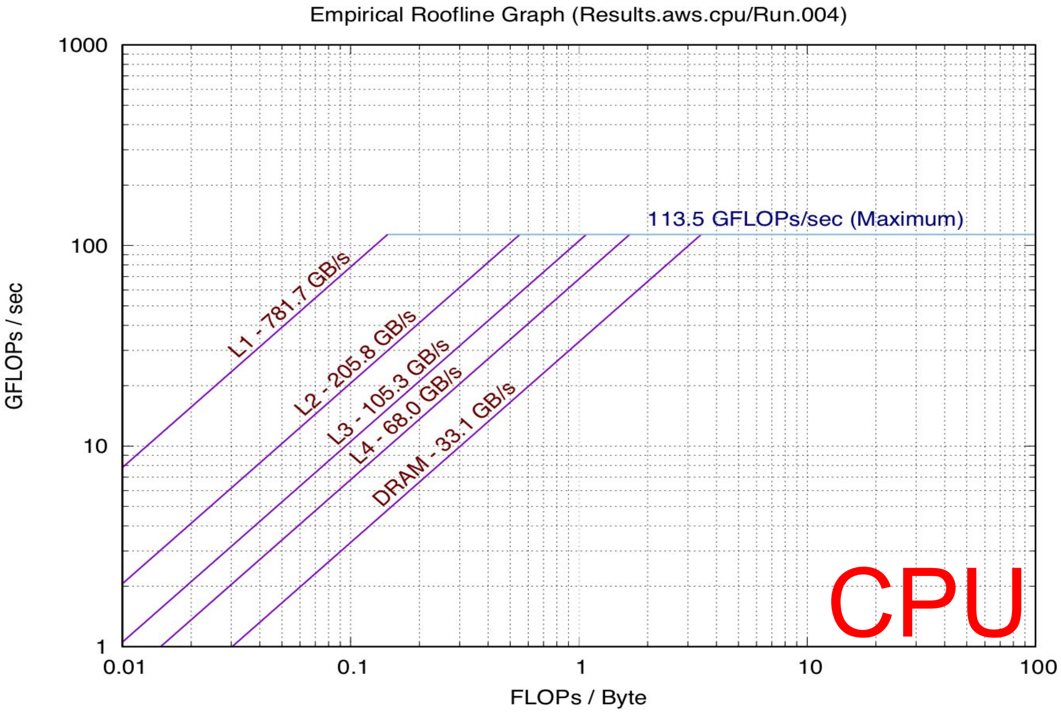
We tried out different configurations on AWS as due to virtualization

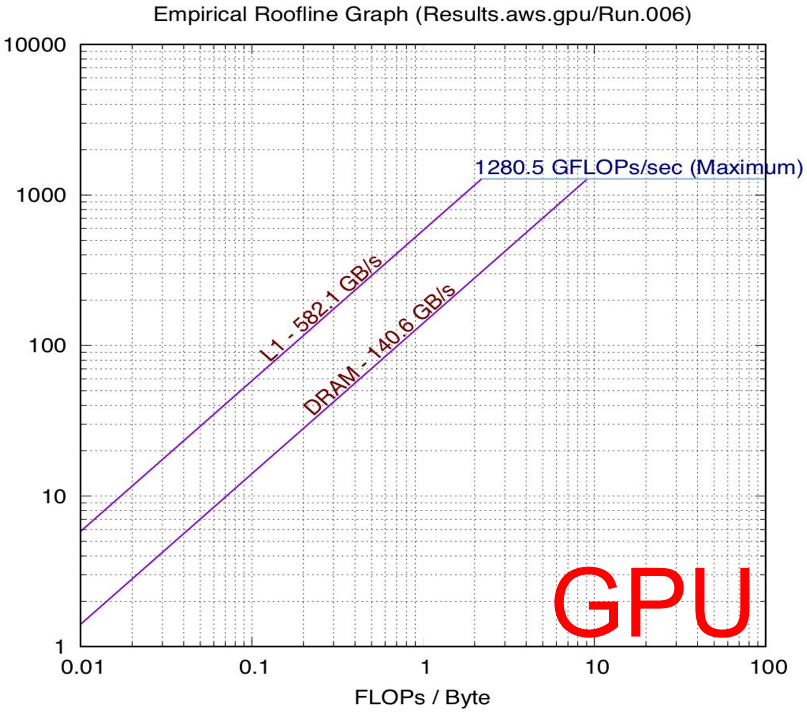
```
ERT_SPEC_GBYTES_DRAM   60

ERT_SPEC_GFLOPS        260.40

ERT_DRIVER  driver1
ERT_KERNEL  kernel1

ERT_OPENMP         True
ERT_OPENMP_CFLAGS  -openmp
ERT_OPENMP_LDFLAGS -openmp

ERT_FLOPS   1,2,4,8,16
ERT_ALIGN   64

ERT_CC      gcc
ERT_CFLAGS  -O3 -mavx2 -march=native -Wno-abi -fno-inline -fopenmp

ERT_LD      gcc
ERT_LDFLAGS   -fopenmp
ERT_LDLIBS

ERT_RUN     export KMP_AFFINITY=scatter; export OMP_NUM_THREADS=ERT_OPENMP_THREADS; ERT_CODE

ERT_OPENMP_THREADS 1,2,4,8

ERT_NUM_EXPERIMENTS 5

ERT_MEMORY_MAX 1073741824

ERT_WORKING_SET_MIN 1
```

```
ERT_GPU         True
ERT_GPU_CFLAGS  -x cu
ERT_GPU_LDFLAGS

ERT_FLOPS   1,2,4,8,16,32,64,128,256
ERT_ALIGN   32

ERT_CC      nvcc
ERT_CFLAGS  -O3

ERT_LD      nvcc
ERT_LDFLAGS
ERT_LDLIBS

ERT_RUN     ERT_CODE

# ERT_BLOCKS_THREADS 28672
# ERT_GPU_BLOCKS     28,56,112,224,448
# ERT_GPU_THREADS    64,128,256,512,1024

ERT_BLOCKS_THREADS 39936
ERT_GPU_BLOCKS 52,104,208
ERT_GPU_THREADS 192,384,768

ERT_NUM_EXPERIMENTS 1

ERT_MEMORY_MAX 1073741824

ERT_WORKING_SET_MIN 128

ERT_TRIALS_MIN 1
```

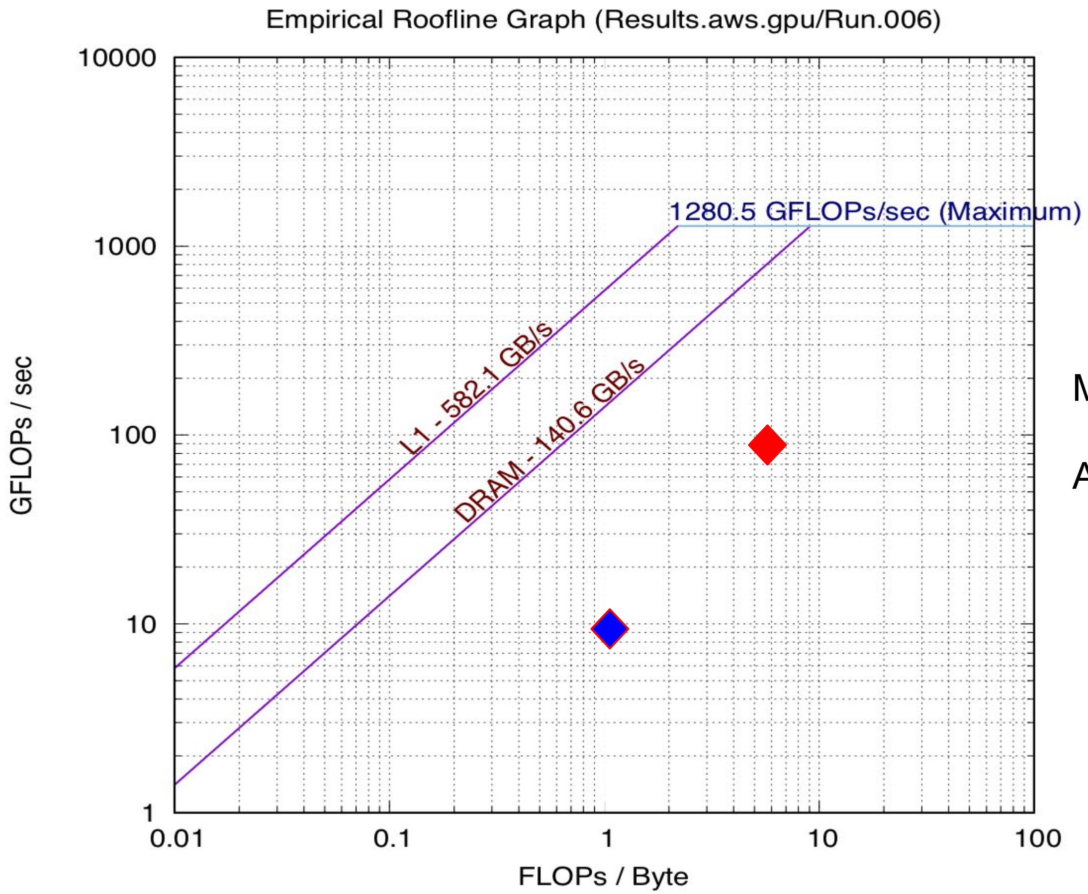
Hardware Benchmarking

Processor	Cores	CUDA Cores	Frequency	GFLOPs (double)[1]
NVIDIA Tesla K80 GPU (Kepler)	2 x 13 (SMX)	2 x 2,496	562 MHz	2 x 1,455





Tensorflow Kernel Benchmarking



Most Utilized Kernel: ◆

Avg Utilized Kernel: ◆

PyTorch Kernel Benchmarking (???)