Deep Learning Ecosystem

Tools of the trade
October 2018

Outline

1. Shifting Mindsets - using APIs
2. GPU vs CPU
3. Nvidia’s contribution
4. APIs
5. Installation process
6. Building your own rig

Shifting mindsets - Using APIs

So far we've programmed our neural networks

Why should we use an API?

- APIs abstract much of the complicated math
- There are memory optimized (i.e. XLA)
- Provide greater flexibility with best coding practices supporting their methods

APIs offer:

- An interface for symbolic computation
- Community support
- Easy of iteration and prototyping
- Open-source, i.e. can be used in commercial products

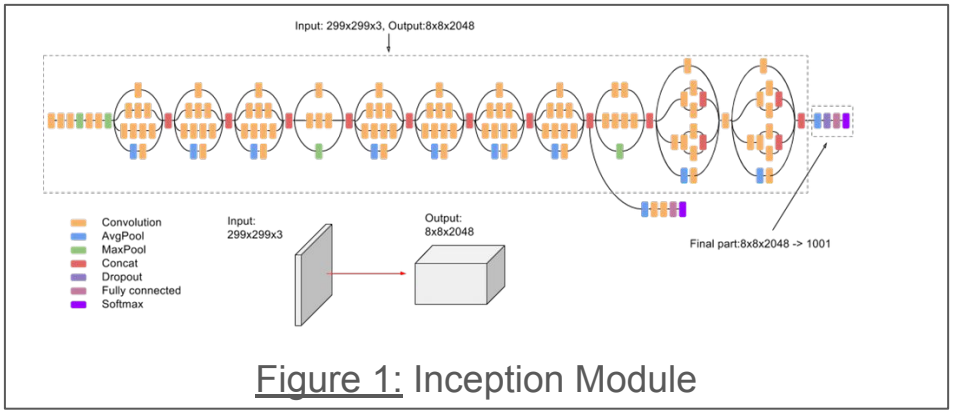


Figure 1: Inception Module

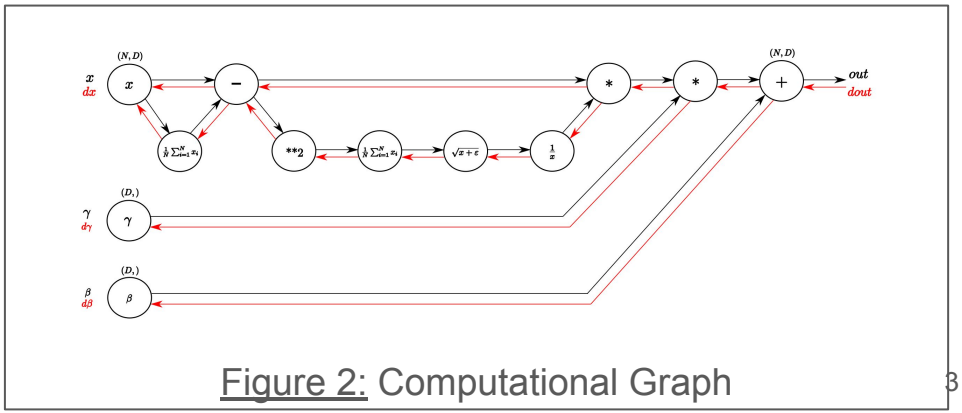


Figure 2: Computational Graph

GPUs rather than CPUs are used in DL

CPUs are fast calculators:

- They process information serially
- Benefit from multi-core distribution of computation
- Speedup of non-serial operations will be ~linear however are limited by the number of cores

Deep learning benefits from being able to parallelize operations, in particular matrix, batching and gradient calculations

CPUs while fast are limited by the number of cores/threads that distribute matrix operations

In contrast to CPUs, GPUs have thousands of cores (that are much slower) but can distribute and collect data more efficiently

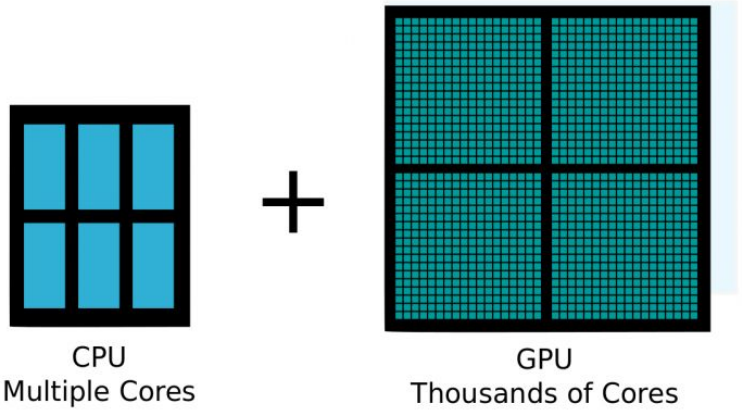


Figure 3: Core comparison CPU vs GPU

* Before Google had TPUs (tensor processing units) and NVidia's Infiniband technology, large early models were trained on clusters of CPUs rather than GPUs

Bottlenecks between CPU/GPUs limit training speed

- GPU bandwidth is quicker than CPU
- PCIe communication is important for sending data to the CPU and writing to disk (checkpointing)

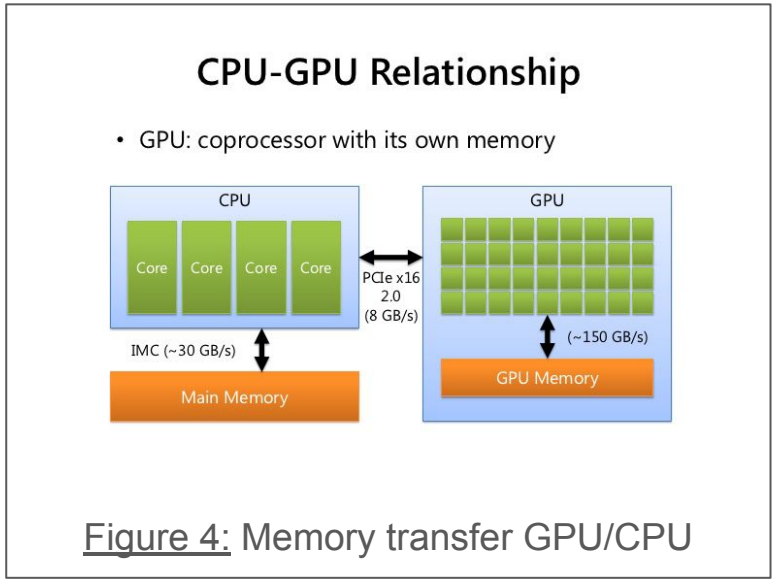


Figure 4: Memory transfer GPU/CPU

- GPUs are bottlenecked by PCIe bandwitdth with CPU for “reduce”-type operations
- GPU clusters to talk directly with one another through NVLink

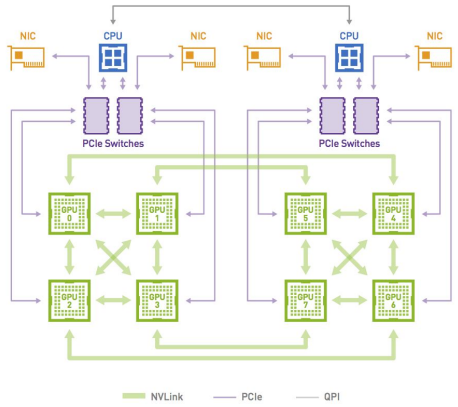


Figure 5: NVLink Technology for GPU Clusters

NVIDIA’s contribution

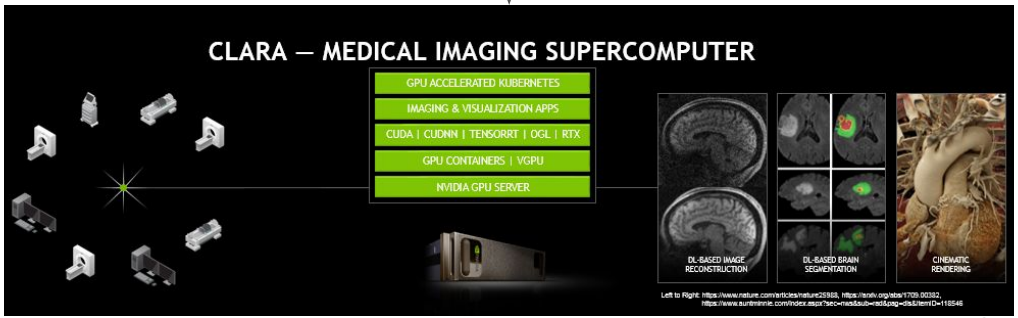
- In early years NVidia's hardware was used for gaming and video editing

- The hardware was designed to deal with heavy parallel workloads such as image manipulation

- What lead to NVidia's early advantage was the distribution of a proprietary set of drivers NVidia Cuda in 2006

- These drivers transformed GPUs for gaming/rendering to general purpose GPUS of __GPGPUs__ that enabled large matrix computation for machine learning

Figure 6: *Early gaming GPUs such as the GTX 900 series (2014) helped subsidize Deep Learning hardware*



Theano changed the game for Deep learning

Theano, developed by the MILA lab at L'Universite de Montreal circa 2008 helped establish NVidia's dominance in AI over competing AMD (and Intel) GPUs which did not have the same type of software-hardware support

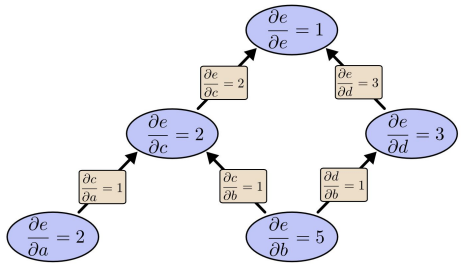
By adapting graphics cards to machine learning workloads similar boosts in performance could be achieved to early CPU clusters more affordably (for the amount of data that was being used)

Figure 7: *Theano was created by the MILA lab*

theano

MILA

Figure 8: *Among the innovations introduced by Theano include the symbolic/computational graph*



A future look at deep learning hardware

Nvidia will likely remain the dominant provider of DL hardware and low level-software for the upcoming years

Although several incumbents in the fields of hardware and software are beginning to emerge

Hardware challengers include Google (TPU), Facebook and Amazon

Software challengers include OpenGL and PlaidML

In addition FPGA (Field Programmable Gate Arrays) are entering the space as alternate DL accelerators

Other existential considerations include whether performance a considered

Figure 9: *Google's Tensor Processing Unit (TPU)*

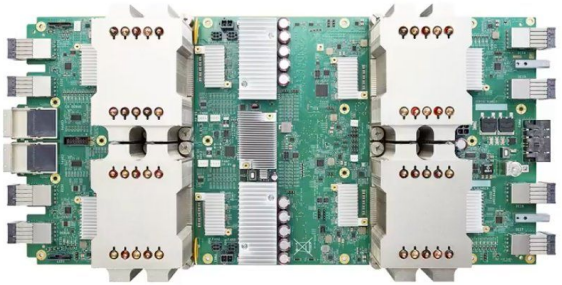


Figure 10: *PlaidML is DL software that is agnostic to GPU hardware*

APIs

History

Theano was the first widely distributed DL framework

It introduced several novel approaches to deep learning

Theano has a steep learning curve, some elements of Theano remain today in other DL frameworks however many frameworks have simplified

In November 2017, after release 1.0, major development stopped as offerings by industrial players offered similar strong

Why use a DL framework?

Simples interface for building neural networks

Open source

Supported by a broad community

Rapid prototyping, development and productionization

Abstracted symbolic computation, GPU acceleration and gradient checking

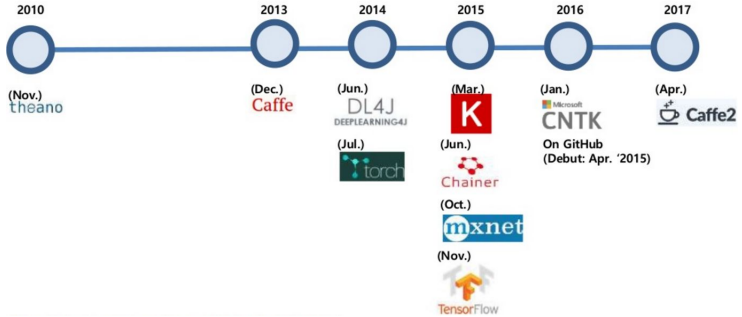
Tensforflow is the most popular framework

Several industrial Deep Learning Frameworks (DLFs) have been open-sourced to advance DL research

Despite being released after several of it's peers Google's Tensorflow has established itself as one of the most popular DL frameworks by focusing on:

- Ease of use in python
- Portability across different platforms
- Easy visualization of trained models through it's Tensorboard plugin
- Community support
- Excellent documentation and tutorials
- Continuously expanding it's offerings

Figure 11: *Deep Learning Framework timeline*



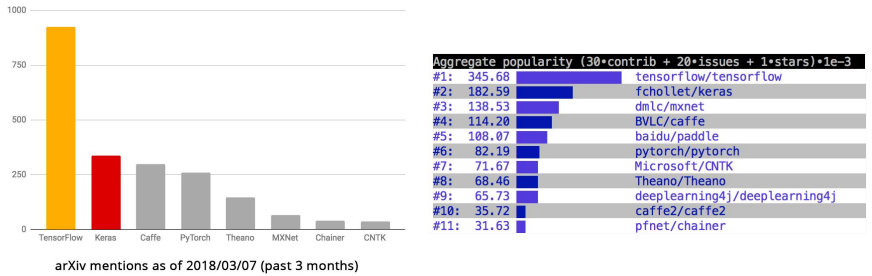


Figure 12: *Mentions arxiv and github (tweet here)*

We're Using Keras why isn't it on the DL list?

We'll be using Keras because

- Can be used with several frameworks
- Simplifies design and implementation of networks
- Good documentation
- Easy to learn and build models

An important reason to use Keras is that while backends change over time, Keras is a wrapper and is unlikely to be supplanted given it's ease of use

That said, it does have some drawbacks for building more customized models but at that point it is probably a good time to switch over to learning a DL backend

Figure 7: *Keras was conceived by Francis Chollet. In early 2018 it was integrated with Tensforflow*

Installing Tensorflow GPU version is convoluted

Tensorflow comes in 2 flavors, vanilla Tensorflow (CPU) and Tensorflow-GPU

To install Tensforflow-GPU you need to install the underlying NVidia drivers

It's easier to install a Docker image or spin up and AWS EC2 AMI with these drivers installed

In most cases however to get optimal performance you will need to compile and install them from source, these include: `tensorRT`, `cuBLAS`, `cuSPARSE`, `cuDNN`

*Take benchmarks for various frameworks with a grain of salt since most of them cannot guarantee that they've been optimized

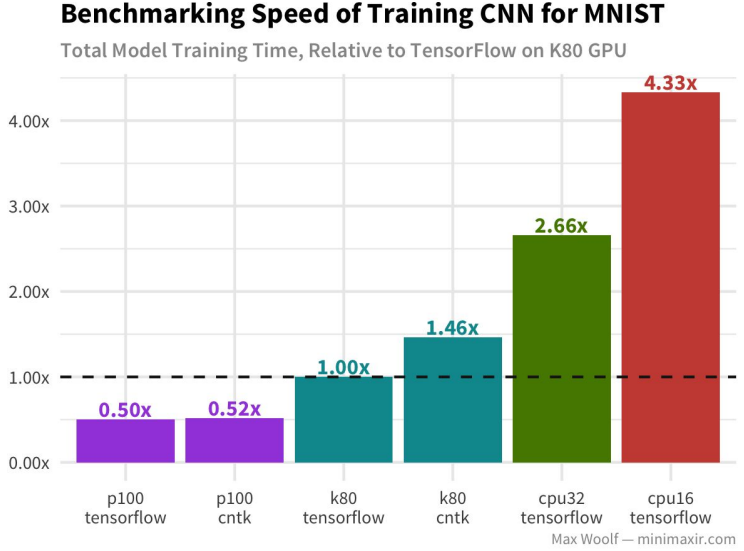


Figure 13: *Benchmarks for GPU to CPU tensorflow by Max Woolf (11/2017 - see full post here)*

Building your own workstation

Why build your own system?

Spinning up an instance on an EC2 is expensive

Allows rapid prototyping

Creates an appreciation of the software/hardware integration

Greater control over system and easier to build locally then deploy at scale when ready to productionize

Are there alternatives?

For experimentation Google CoLab provide free GPU support

Lambda Labs builds custom rigs

GPUs on the cloud:

- AWS, Azure, Google Cloud
- NVidia also offers cloud support
- Floydhub

Figure 14: *Lambda Labs custom build*

What components do I need?

1. CPU: AMD Threadripper with at least 40 PCIe Lanes (more if you are planning to use >2 GPUs)
2. GPU: NVIDIA 1000 or 2000 series
3. RAM: Double amount of GPU memory
4. Motherboard: One that supports your CPU, gaming motherboards are usually best
5. Hard-drive: NVME SSD (not SATA or HD)
6. Power supply: 500W + #GPUs x 150W
7. Cooling: Liquid

Total cost: ~$2000(custom build) - $15000 (Lambda Labs)
*See notes for more details

Figure 15: NVidia *GPUs*

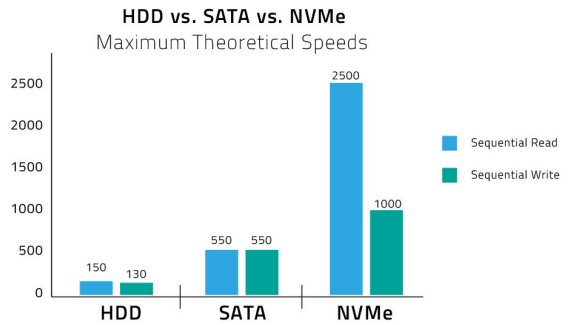


Figure 16: *Mentions arxiv and github (tweet here)*