Inference, Low-Cost Models, and Compression

CS6787 Lecture 11 — Fall 2018

Review: Inference

- Suppose that our training loss function looks like

$$f(w) = \frac{1}{N} \sum_{i=1}^{n} l(\hat{y}(w; x_i), y_i)$$

- Inference is the problem of computing the prediction

$$\hat{y}(w; x_i)$$

Why should we care about inference?

- **Train once, infer many times**
  - Many production machine learning systems just do inference
- Often want to run inference on **low-power edge devices**
  - Such as cell phones, security cameras
  - **Limited memory** on these devices to store models
- Need to get **responses to users quickly**
  - On the web, users won't wait more than a second

Inference on neural networks

- Just need to run the forward pass of the network.
  - A bunch of matrix multiplies and non-linear units.

- Unlike backpropagation for learning, here we do not need to keep the activations around for later processing.

- This makes inference a much simpler task than learning.
  - Although it can still be costly — it's a lot of linear algebra to do.

Metrics for Inference

- Important metric: **throughput**
  - **How many examples** can we classify in some amount of time
- Important metric: **latency**
  - **How long** does it take to get a prediction for a single example
- Important metric: **model size**
  - **How much memory** do we need to store/transmit the model for prediction
- Important metric: **energy use**
  - **How much energy** do we use to produce each prediction
- **What are examples where we might care about each metric?**

Improving the performance of inference

Altering the batch size

- Just like with learning, we can **make predictions in batches**
- Increasing the batch size helps **improve parallelism**
  - Provides more work to parallelize and an additional dimension for parallelization
  - This improves **throughput**
- But increasing the batch size can make us do more work before we can return an answer for any individual example
  - Can negatively affect **latency**

Demo

Compression

- Find an **easier-to-compute network** with similar accuracy
  - Or find a network with **smaller model size**, depending on the goal

- **Many techniques** for doing this

- Usually involve some sort of **fine-tuning**
  - Apply a lossy compression operation, then retrain the model to improve accuracy

- The subject of this week's paper

Low-precision arithmetic for inference

- Very simple technique: just use low-precision arithmetic in inference
- Can make any signals in the model low-precision
- Simple **heuristic for compression**: keep lowering the precision of signals until the accuracy decreases
  - Can often get down below 16 bit numbers with this method alone
- **Binarization/ternarization** is low-precision arithmetic in the extreme

Pruning

- **Remove activations** that are usually zero
  - Or that don't seem to be contributing much to the model
- Effectively creates **a smaller model**
  - This makes it easy to retrain, since we're just training a smaller network
- There's always the question of whether training a smaller model in the first place would have been as good or better.
  - But usually pruning is observed to produce benefits.

Knowledge distillation

- Idea: take a large/complex model and **train a smaller network to match its output**
  - Often used for distilling **ensemble models** into a single network
  - E.g. Hinton et. al. "Distilling the Knowledge in a Neural Network."

- Can also improve the accuracy in some cases.

Efficient architectures

- Some neural network architectures are **designed to be efficient at inference time**
  - Examples: MobileNet, ShuffleNet, CirCNN
- These networks are often based on sparsely connected neurons
  - This limits the number of weights which makes models smaller and easier to run inference on
- To be efficient, we can just **train one of these networks in the first place** for our application.

Re-use of computation

- For video and time-series data, there is a lot of **redundant information** from one frame to the next.

- We can try to **re-use some of the computation** from previous frames.

- This is less popular than some of the other approaches here, because it is not really general.

The last resort for speeding up DNN inference

- **Train another, faster type of model** that is not a deep neural network
  - For some real-time applications, you can't always use a DNN

- If you can get away with **a linear model**, that's almost always much faster.

- Also, **decision trees** tend to be quite fast for inference.

Where do we run inference?

Inference in the cloud

- Most inference today is run on **cloud platforms**
- The cloud supports **large amounts of compute**
  - And makes it easy to access it and make it reliable
- This is a good place to put AI that needs to think about data
- For interactive models, **latency** is critical

Inference on edge devices

- Inference can run on your **laptop or smartphone**
  - Here, the size of the model becomes more of an issue
  - Limited smartphone memory
- This is good for **user privacy and security**
  - But not as good for companies that want to keep their models private
- Also can be used to deploy **personalized models**

Inference on sensors

- Sometimes we want **inference right at the source**
  - On the sensor where data is collected

- Example: a surveillance camera taking video
  - Don't want to stream the video to the cloud, especially if most of it is not interesting.

- **Energy use** is very important here.

Questions?

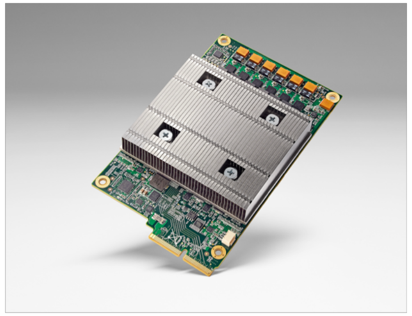
- Upcoming things
  - Paper Review #8a or #8b — **due today**
  - Paper Presentation #9a and #9b **on Wednesday**

Hardware for Machine Learning

CS6787 Lecture 11 — Fall 2018

Recap: modern ML hardware

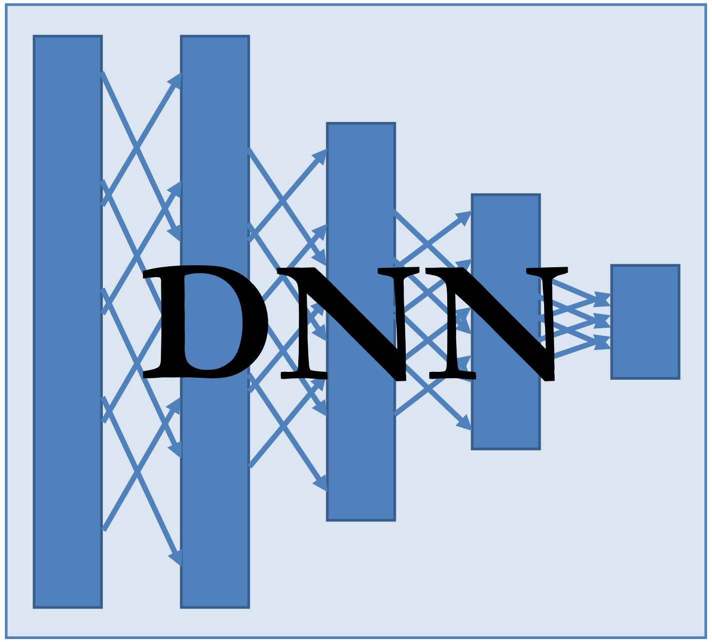
- **Lots of different types**
  - CPUs
  - GPUs
  - FPGAs
  - Specialized accelerators



- Right now, **GPUs are dominant**…we'll get to why later

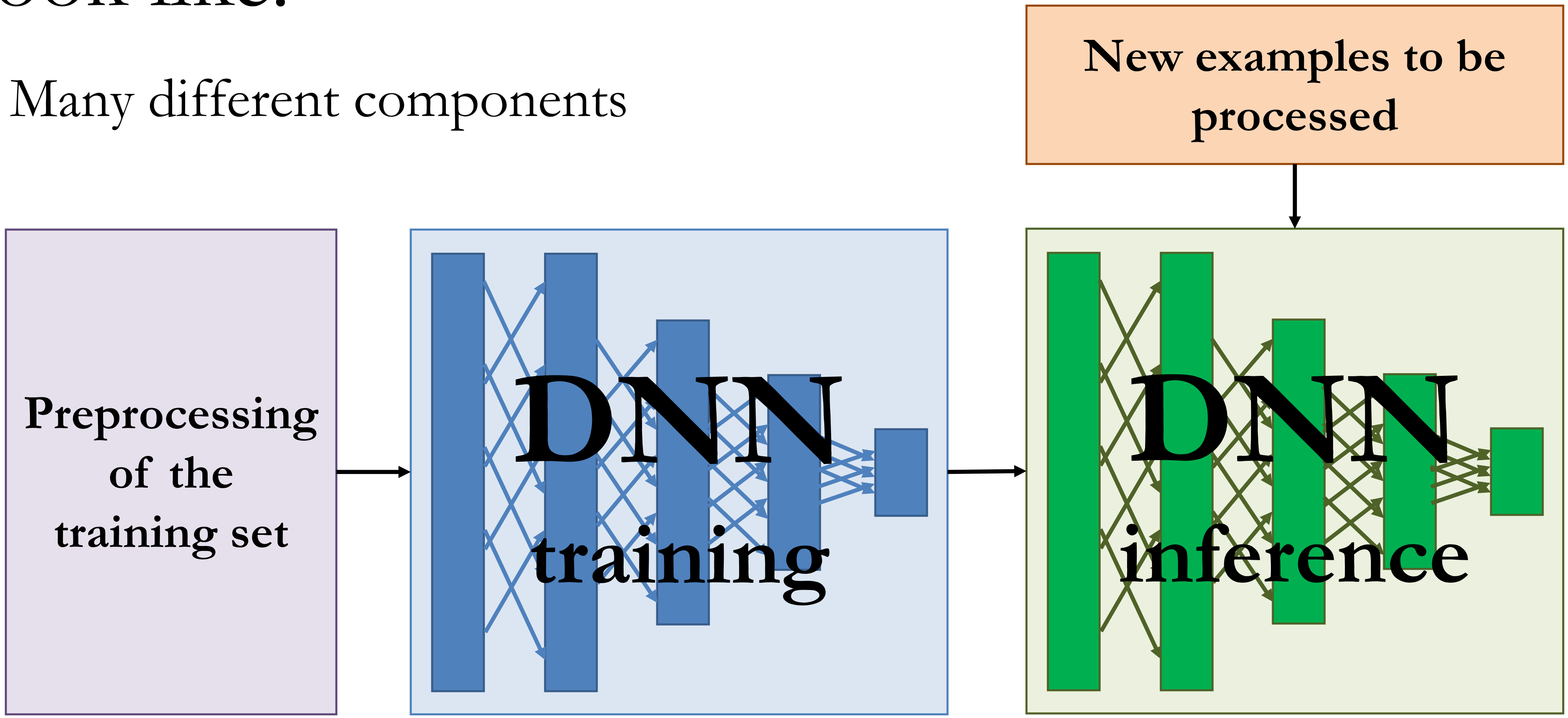
What does a modern machine learning pipeline look like?

- Not just a neural network:



What does a modern machine learning pipeline look like?

- Many different components



Where can hardware help?

- **Everywhere!**

- There's interest in using hardware everywhere in the pipeline
  - both **adapting existing hardware architectures**, and
  - **developing new ones**

- What improvements can we get?
  - **Lower latency inference**
  - **Higher throughput training**
  - **Lower power cost**

How can hardware help? Three ways

- Speed up the **basic building blocks** of machine learning computation
  - Major building block: **matrix-matrix multiply**
  - Another major building block: **convolution**
- Add **data/memory paths specialized** to machine learning workloads
  - Example: having a local cache to store network weights
- Create **application-specific functional units**
  - Not for general ML, but for a specific domain or application

Why are GPUs so popular for machine learning?

Why are GPUs so popular for *training deep neural networks*?

GPU vs CPU

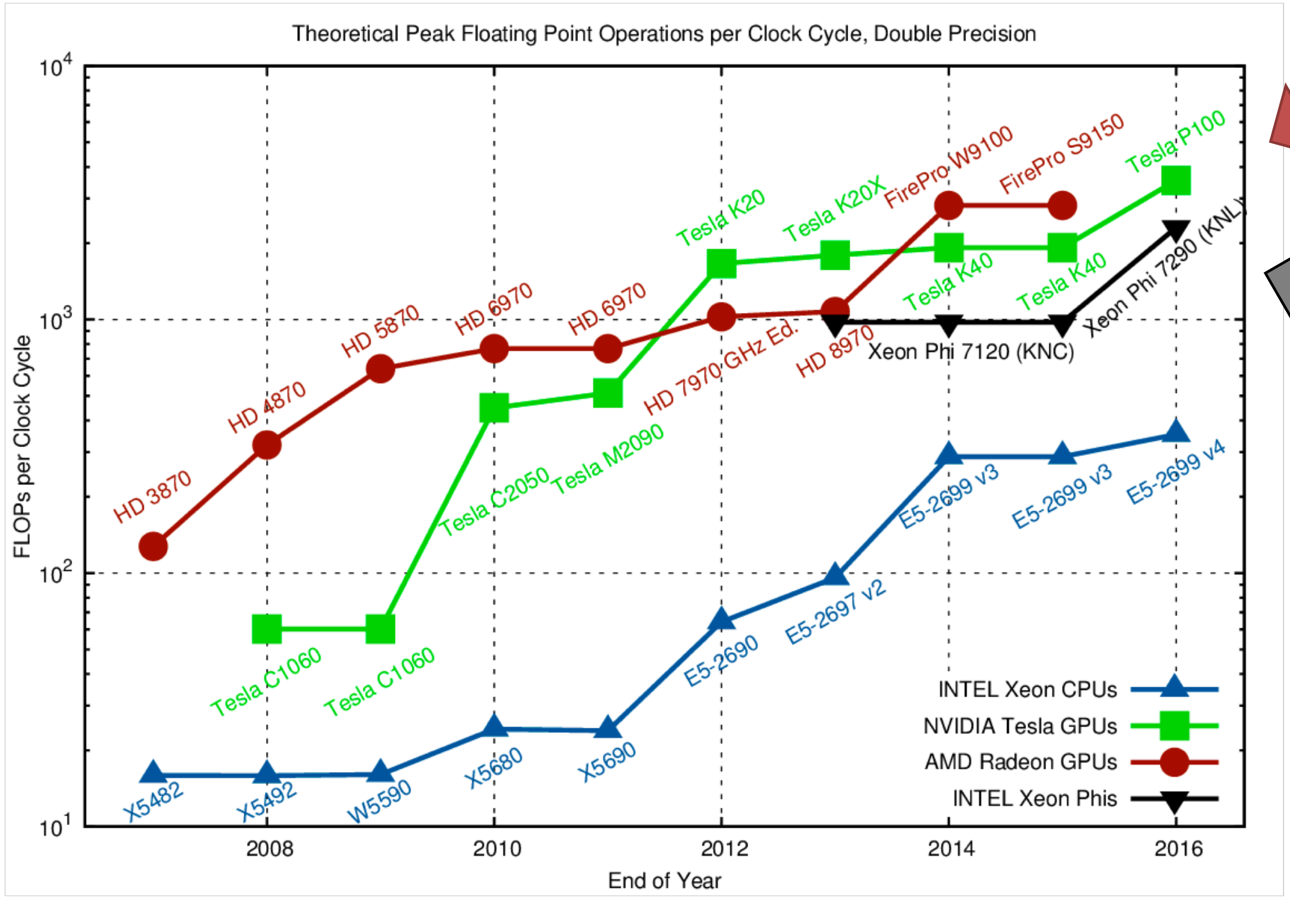
- **CPU is a general purpose processor**
  - Modern CPUs spend most of their area on deep caches
  - This makes the CPU a great choice for applications with random or non-uniform memory accesses

- **GPU is optimized for**
  - more compute intensive workloads
  - streaming memory models

Machine learning applications look more like this

FLOPS: GPU vs CPU

- FLOPS: **f**loating **p**oint **o**perations **p**er **s**econd



GPU FLOPS consistently exceed CPU FLOPS

Intel Xeon Phi chips are compute-heavy manycore processors that compete with GPUs

From Karl Rupp's blog https://www.karlrupp.net/2016/08/flops-per-cycle-for-cpus-gpus-and-xeon-phis/

This was the best diagram I could find that shows trends over time.

Memory bandwidth: CPU vs GPU

- GPUs have **higher memory bandwidths** than CPUs
  - E.g. new NVIDIA Tesla V100 has a claimed **900 GB/s memory bandwidth**
  - Wheras Intel Xeon E7 has only about **100 GB/s memory bandwidth**

- But, this **comparison is unfair!**
  - GPU memory bandwidth is the bandwidth to GPU memory
  - E.g. on a PCIE2, bandwidth is only **32 GB/s for a GPU**

What limits deep learning?

- **Is it compute bound or memory bound?**

- Ideally: it's **compute bound**
  - Why? Matrix-matrix multiply takes $\mathbf{O(n^2)}$ memory but $\mathbf{O(n^3)}$ compute

- Sometimes it is memory/communication bound
  - Especially when we are running at **large scale on a cluster**

Challengers to the GPU

- More **compute-intensive CPUs**
  - Like Intel's Phi line — promise same level of compute performance and better handling of sparsity
- **Low-power devices**
  - Like mobile-device-targeted chips
  - Configurable hardware like FPGAs and CGRAs
- Accelerators that **speed up matrix-matrix multiply**
  - Like Google's TPU

Will all computation become dense matrix-matrix multiply?

Deep learning and matrix-matrix multiply

- Traditionally, the most costly operation for deep learning for both training and inference is dense **matrix-matrix multiply**

- Matrix-matrix multiply at $\mathbf{O(n^3)}$ scales worse than other operations
  - So should expect it to **become even more of a bottleneck** as problems scale

- Deep learning is **still exploding** and capturing more compute cycles
  - Motivates the question: **will most computation in the future become dense matrix-matrix multiply?**

What if dense matrix multiply takes over?

- Great opportunities for **new highly specialized hardware**
  - The TPU is already an example of this
  - It's a glorified matrix-matrix multiply engine

- **Significant power savings** from specialized hardware
  - But not as much as if we could use something like sparsity

- It might put us all out of work
  - Who cares about researching algorithms when there's **only one algorithm anyone cares about?**

What if matrix multiply doesn't take over?

- Great opportunities for designing new **heterogeneous, application-specific hardware**
  - We might want one chip for SVRG, one chip for low-precision

- Interesting systems/framework opportunities to give users **suggestions for which chips to use**
  - Or even to **automatically dispatch work** within a heterogeneous datacenter

- **Community might fragment**
  - Into smaller subgroups working on particular problems

The truth is somewhere in the middle

- We'll probably see **both**
  - a lot of dense matrix-matrix multiply compute
  - a lot of opportunities for faster more specialized compute
- New models are being developed every day
  - And we **shouldn't read too much into** the current trends
- But this means we get the **best of both worlds**
  - We can do research on either side and still have impact

Questions?

- Upcoming things
  - Paper Review #8a or #8b — **due today**
  - Paper Presentation #9a and #9b **on Wednesday**