performance & deployment

- Check if a GPU is underutilized by running `nvidia-smi -l 2`. If GPU utilization is not approaching 80-100%, then the input pipeline may be the bottleneck. Check CPU usage. It is possible to have an optimized input pipeline and lack the CPU cycles to process the pipeline.
- Reading large numbers of small files significantly impacts I/O performance. One approach to get maximum I/O throughput is to preprocess input data into larger (~100MB) `TFRecord` files. For smaller data sets (200MB-1GB), the best approach is often to load the entire data set into memory.
- To use multiple GPUs to train, we run the same copy of the model with different batches on each GPU, compute gradient for each GPU, send that gradient to CPU for overall gradient update step (this sync is called the parameter server approach). The a synced copy is sent back to each GPU to run the next batch.
- These days, we actually have a lot of models being deployed in commercial applications. The computation demands of training grow with the number of researchers, but the cycles needed for inference expand in proportion to users. That means pure inference efficiency has become a burning issue for a lot of teams.
  That is where quantization comes in. It's an umbrella term that covers a lot of different techniques to store numbers and perform calculations on them in more compact formats than 32-bit floating point. I am going to focus on eight-bit fixed point, for reasons I'll go into more detail on later.
- Training neural networks is done by applying many tiny nudges to the weights, and these small increments typically need floating point precision to work (though there are research efforts to use quantized representations here too).
- Taking a pre-trained model and running inference is very different. One of the magical qualities of deep networks is that they tend to cope very well with high levels of noise in their inputs. If you think about recognizing an object in a photo you've just taken, the network has to ignore all the CCD noise, lighting changes, and other non-essential differences between it and the training examples it's seen before, and focus on the important similarities instead. This ability means that they seem to treat low-precision calculations as just another source of noise, and still produce accurate results even with numerical formats that hold less information.
- Quantization helps take up lesser space and do faster math which helps to do inference faster (use a 32 bit gpu op to do 4 8-bit ops!)
- TensorFlow has production-grade support for eight-bit calculations built in. It also has a process for converting many models trained in floating-point over to equivalent graphs using quantized calculations for inference.
- We approach converting floating-point arrays of numbers into eight-bit representations as a compression problem. We know that the weights and activation tensors in trained neural network models tend to have values that are distributed across comparatively small ranges (for example you might have -15 to +15 for weights, -500 to 1000 for activations on an image model, though the exact numbers will vary). We also know from experiment that neural nets tend to be very robust in the face of noise, and so the noise-like error produced by quantizing down to a small set of values will not hurt the precision of the overall results very much. We also want to pick a representation that's easy to perform calculations on, especially the large matrix multiplications that form the bulk of the work that's needed to run a model.

- We've found that we can get extremely good performance on mobile and embedded devices by using eight-bit arithmetic rather than floating-point.
- XLA:
  - *Improve execution speed.* Compile subgraphs to reduce the execution time of short-lived Ops to eliminate overhead from the TensorFlow runtime, fuse pipelined operations to reduce memory overhead, and specialize to known tensor shapes to allow for more aggressive constant propagation.
  - *Improve memory usage.* Analyze and schedule memory usage, in principle eliminating many intermediate storage buffers.
  - *Reduce reliance on custom Ops.* Remove the need for many custom Ops by improving the performance of automatically fused low-level Ops to match the performance of custom Ops that were fused by hand.
  - *Reduce mobile footprint.* Eliminate the TensorFlow runtime by ahead-of-time compiling the subgraph and emitting an object/header file pair that can be linked directly into another application. The results can reduce the footprint for mobile inference by several orders of magnitude.
  - *Improve portability.* Make it relatively easy to write a new backend for novel hardware, at which point a large fraction of TensorFlow programs will run unmodified on that hardware. This is in contrast with the approach of specializing individual monolithic Ops for new hardware, which requires TensorFlow programs to be rewritten to make use of those Ops.
- TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs.
- Remember, TF has a client server architecture of defining computational graph and then running graph.
- TF has hooks to add new ops, write new data readers, create new estimators etc.