Practical issues in training deep neural nets for genomics

Chris Probert & CS Foo
Kundaje Lab

We typically focus on the math and theory behind deep learning

Today: let’s consider the systems problems in applying deep learning

Outline

- Training neural networks with online optimization
  - I/O considerations
  - Data augmentation
- Multi-threaded I/O
  - Keras - prefetching with fit_generator
  - TensorFlow - multi-threaded input pipelines

Training neural networks with online optimization

- Estimate gradient on full dataset using a small subset of examples (minibatch)
- Take small, locally optimal step w.r.t each minibatch → converge faster
- Minibatch needs to be “representative” - take **random sample** from full dataset
- Minibatch size typically ranges from **16 - 512** examples

I/O Considerations: IOPS

- Need to **randomly sample** 16 - 512 examples per training iteration
- IOPS: **I**/O **O**perations **P**er **S**econd
- Typical numbers for 4KB random reads
  - Hard drive: 100-500 IOPS
  - SSD: 10k - 100k IOPS **(100x of HDD)**
  - On cloud platforms these numbers are likely lower due to hardware sharing
- SSDs are necessary when streaming from disk

I/O Considerations: Decompression

- Data is typically compressed (e.g. JPEGs) - need to decompress before feeding into model
- Decompression for common formats is slow
  - JPEG - 20-50 MB/s
  - GZIP (zlib) - 80-100 MB/s
    - used in many genomics file formats, including BAM, bigWig, bigBed, BCF
- Example: 256 images from ImageNet (~200kb each) would take about 1s to decompress. One forward + backward pass through the network on a K40 for AlexNet only takes 0.5s.

Aside: Data Augmentation

- Add synthetic data to increase size of training set
- Synthetic data generated by perturbing training data
  - Reverse complements of sequence inputs
  - Jittering peaks/windows
  - Resampling/subsampling reads
  - Adding noise
- Can pick perturbations to make model robust to those
- This is also related to I/O because you can’t possibly store and generate all perturbations beforehand

Latency hiding with concurrency

- Time taken for random data sampling & decompression (or other processing such as augmentation) can be as much as for a forward/backward pass
  - Especially when there are large number of inputs and/or a small model, as is typical in genomics applications
- Usual systems trick: hide this I/O time by running other processes simultaneously
- We will discuss threading based solutions that are implemented in popular deep learning packages

Simple approach: load data in memory

Before training, load data in memory.

Pros: fast data access (just copy from memory as needed)

Cons: requires data to fit in memory; data must be loaded before training

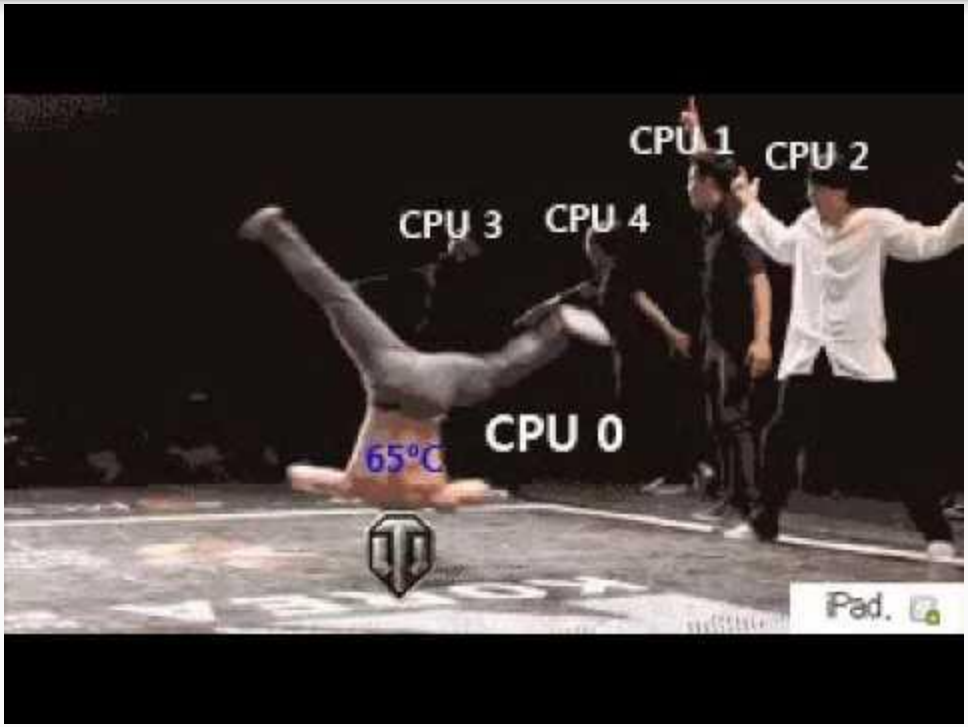
Streaming data - reduced memory footprint

If the data doesn’t fit in memory, need to stream it from disk.

One option: pre-process data, write in an easy-to-stream format (like hdf5)

Pros: Can easily stream batches from disk by reading file blocks

Cons: Requires pre-processing; doesn’t scale in number of data configurations; can significantly slow down training while reading from disk


CPU 1
CPU 2
CPU 3
CPU 4
CPU 0
65°C
Pad.

Multi-threaded streaming I/O from disk

- Key idea: use separate threads to extract from files on disk
- **Pros**: de-couples training loop from data loading (can run in parallel)
  - Even true in Python, since GIL can be released during system calls
- **Cons**: requires work to set up, not as fast as in-memory

Keras - fit_generator

fit_generator

```
fit_generator(self, generator, samples_per_epoch, nb_epoch, verbose=1, callbacks=[], validation_data=Nc
```

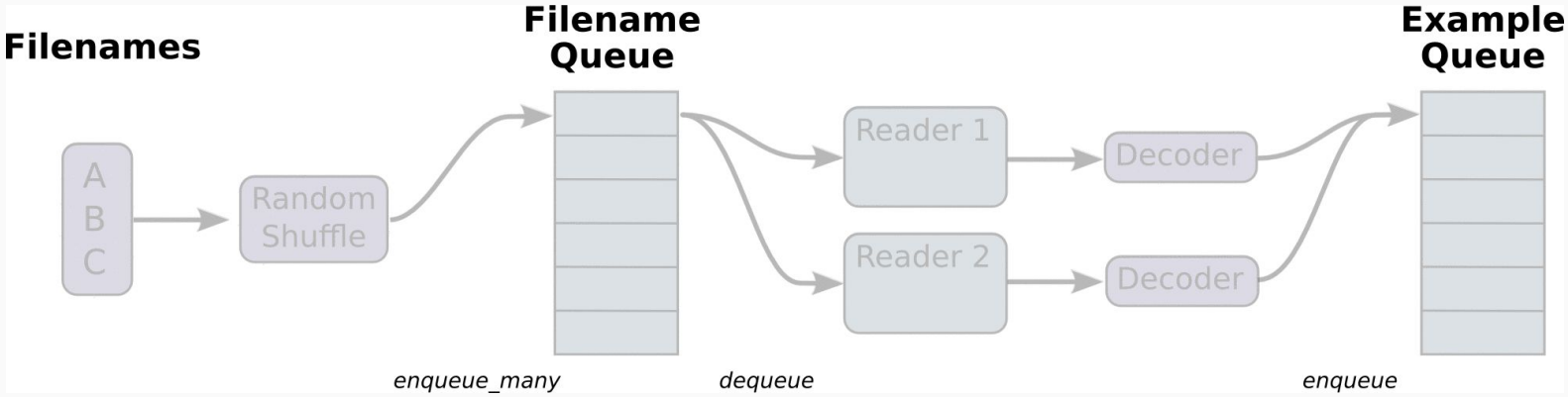
Fits the model on data generated batch-by-batch by a Python generator. The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.

Arguments

- **generator**: a generator. The output of the generator must be either
  - a tuple (inputs, targets)
  - a tuple (inputs, targets, sample_weights). All arrays should contain the same number of samples. The generator is expected to loop over its data indefinitely. An epoch finishes when `samples_per_epoch` samples have been seen by the model.

TensorFlow - streaming I/O

- Queues are very useful for parallel algorithms
- Central to many TensorFlow I/O routines (and other ops)



Interactive example: TensorFlow streaming

Batch sizes and algorithms

- What's a good batch size?
  - Minimum: 4-16
  - Maximum: often, the size that can fit in GPU memory
  - Often: 64 - 256
    - Tradeoff between speed of convergence and proximity to stationary point
    - Power of 2 can have memory allocation + vectorization benefits
- What's a good algorithm?
  - ConvNets: Adam is a good default choice
  - Recurrent nets: RMSprop or SGD can be good default choices
  - Worth experimenting with as a hyperparameter