Deploying Neural Models for Theorem Proving

Or, a bag of tricks to make Stuff Go Faster and thereby hasten the arrival of the semantic AI paradise of computer-understandable math and science

Michael Rawson

TECHNISCHE UNIVERSITÄT WIEN

What's a Neural Model?

- Blob of linear algebra
- Architecture + **learned weights**
- *Relatively* simple to evaluate
- Harder to train
- Slow! If it's not slow, make it bigger.
  - Faster on GPU
- Libraries: PyTorch, TensorFlow etc
- Input: "tensors"
- Output: **policy** or **value**

https://xkcd.com/1838/

What's a Theorem Prover?

- Why, it's a system that proves theorems
  - By search, we're not magicians
  - Classical first-order. Fight me.
- Fast exploration of search space critical for performance
- Want to have a good heuristic for where to go
  - But we don't know many good ones
- This is what we use the neural model for!
- Two flavours:
  - **Saturation**: E, Vampire, SPASS, Prover9, OTTER, iProver (sorry), SMT solvers (kinda), …
  - **Backtracking**: leanCoP, SETHEO, rlCoP, plCoP, (Fe)MaLeCoP, other CoPs…

The Aim of the Game

- Need to plug the big heavy NN into the ATP somehow
- Performance is completely destroyed, even if the ATP is “smart” now
- **Fundamental problem**
- Many ways to Not Do That:
  - Smaller, lighter, possibly non-neural models
  - Use the NN only occasionally
  - Use the NN at first, then stop at some point (DNGPS)
  - Staged evaluation with increasingly-complex models (ENIGMA)
  - …
- Circa 2019, I wanted to do this anyway, ignoring good advice.

Performance Indicators

- Problems solved!
- Inferences/sec
- NN state evaluation **throughput**
- NN state evaluation **latency**
- Startup time
- NN accuracy if approximated
- ???

Asynchronous Evaluation

Very Briefly

- On the CPU, do inference work
- On the GPU, do evaluation work
- GPU typically slower than CPU, plays catch-up
- CPU assigns uniform policy to unevaluated stuff
- CPU never “stalls” waiting for GPU
- **Only really works for backtracking systems**
- **Only really works for policy: what value do you choose?**

Sketch Asynchronous Evaluation

Integrating NNs Into ATPs

A Dirty Hack

- Have trained model in e.g. TensorFlow, PyTorch
- Conventional:
  - Save model (weights + arch) to disk somehow
  - Load model again
  - Do inference using library routines
- **Dirty hack**:
  - Save **only weights** to disk as a **header file**
  - Implement your **own forward pass** (!)
  - **Compile weights** into your ATP from header file
  - ???
  - Profit!

Dirty Hack: pros…

- No dependence on third-party library for final binary
- Seriously improved startup time: several seconds to 0.
  - Matters for experiments with 1000s of problems
- No encoding/decoding at interface of ATP/NN: use your own data
  - And DIY parallelism!
- Compiler can do constant folding, loop unrolling, loop fusion etc.
- Cute domain/model-specific tricks (more later)
- Can use Geoff's computers without installing PyTorch on StarExec
- **But…**

…and cons

- Have to implement your own forward pass
- Not so trivial to do hardware acceleration
- Lost library optimisations

Fused and Reused

- Memory reuse, e.g. in-place ReLU
- Tensor allocation reuse
- Fusion: BatchNorm + preceding convolution
- Fun with combination of above and e.g. residual networks

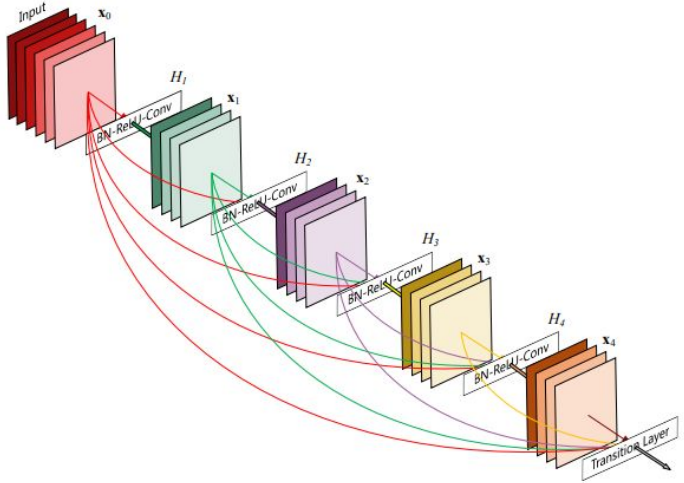


Image from “Densely Connected Convolutional Networks”, Huang *et al*.

Hardware Acceleration

Hardware Acceleration Primer

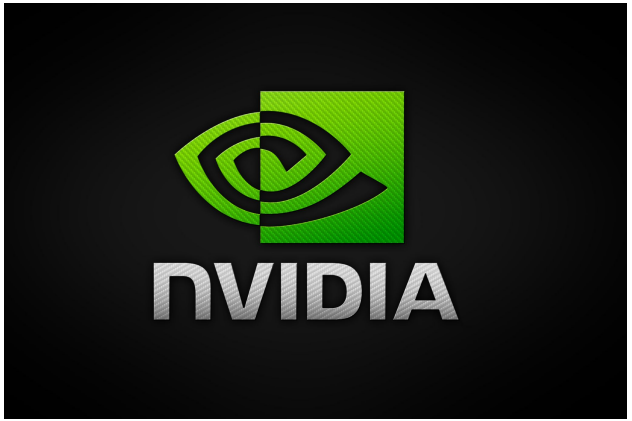
- As models/data get bigger, typical CPUs start to suffer
- If you have a GPU, can abuse it for faster inference
  - Even cheap models! Mine is ancient GT730.
- Put suitable tasks on GPU, do other stuff with CPU
- GPUs:
  - Have huge numbers of “cores”
  - Actually nested (sub-)processors with shared resources
  - Surprising cache effects
  - Low clock speeds
  - Deep instruction pipelines
  - Does not enjoy (conditional) branching **at all**

Hardware Acceleration Landscape

There's a competing ecosystem of accelerator cards at knock-down prices, but you can program for any card or CPU using the OpenCL API, a widely-supported and efficient cross-platform abstraction.

Lies. There is only CUDA, devourer of ~~hopes and dreams~~ large matrices.

CUDA: the briefest of introductions

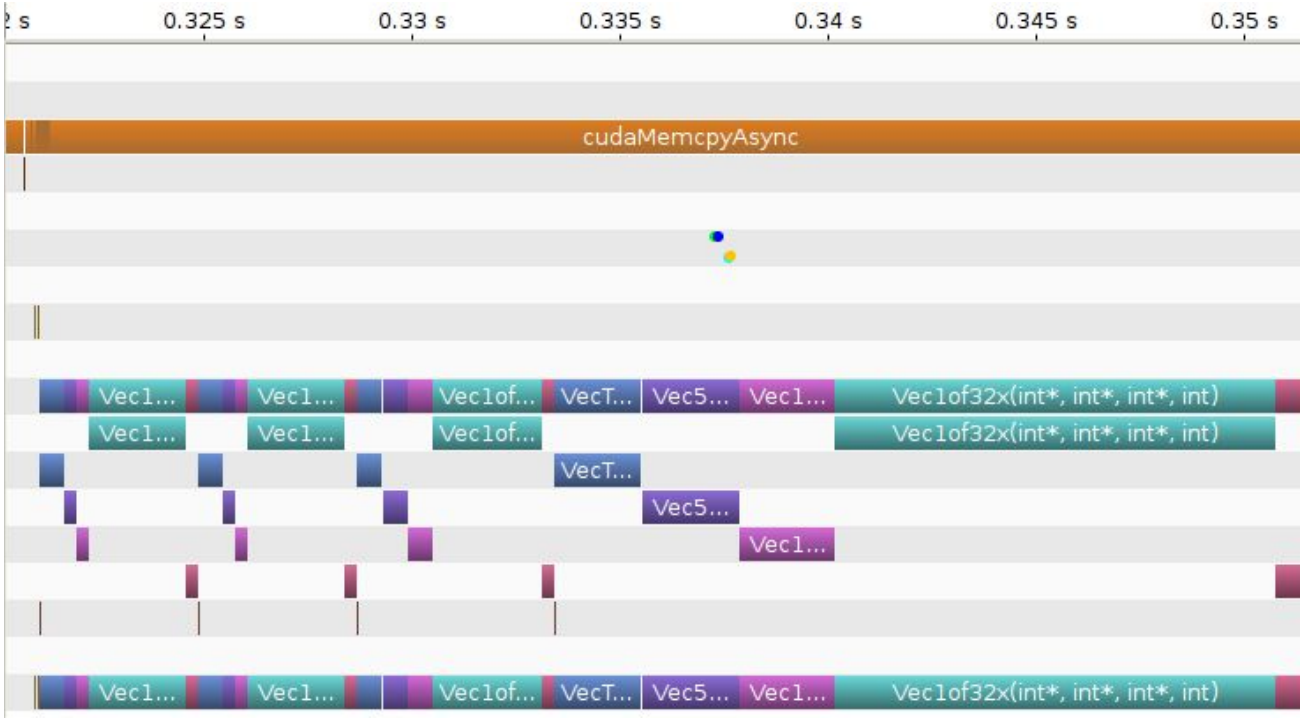


CUDA

- If you have an NVIDIA card, you can use CUDA.
- Once you get past this unpleasantness, can be quite nice:
  - Generally well-supported (by NVIDIA) on various platforms
  - Relatively stable API - if you have an old card, don’t use the fancy stuff
  - Libraries for e.g. BLAS, sparse matrices
  - Nice tooling e.g. profilers
  - Ridiculous performance if you get it right
- Write something like C++, launch *kernels* on the GPU.
- *In practice, need to tune some magic constants for maximum performance.*

CUDA Programming Model

- CUDA mini-programs called *kernels*
- Executing kernel (with parameters etc) called a *thread*
- Grouped into *thread blocks*
- Thread blocks distributed over *streaming multiprocessors*
- GPUs may have several such SMs
- There may be performance interactions at **every** level
- RTFM, it's actually OK.
- Interaction with CPU:
  - Data must be explicitly uploaded/downloaded
  - Operations can be **asynchronous** wrt CPU - **very useful**
- But wait, there's more! Atomics, streams, cooperative groups, dynamic parallelism, textures, compute graphs, …


0.325 s
0.33 s
0.335 s
0.34 s
0.345 s
0.35 s
cudaMemcpyAsync
Vec1...
Vec1...
Vec1of...
VecT...
Vec5...
Vec1...
Vec1of32x(int*, int*, int*, int)
Vec1...
Vec1...
Vec1of...
Vec1of32x(int*, int*, int*, int)
VecT...
Vec5...
Vec1...
Vec1...
Vec1...
Vec1of...
VecT...
Vec5...
Vec1...
Vec1of32x(int*, int*, int*, int)

https://github.com/MichaelRawson/lazycop/blob/master/nn/cuda/model.cu

Results

- System (2020) with trained model makes *more* inferences/sec
  - Still a little weirded out by this tbh
- Hundreds to thousands of policy evaluations per second
- Fully end-to-end GNN, mid-size model
- Low GPU memory usage
- Near 100% utilisation of compute - hit thermal cutout once in summer!
- …the numbers for the associated publication were a bit better?
- Kept my sanity by not trying to use something like TensorFlow Serving
  - Not that such things are bad…square peg, round hole

Other Things I Haven’t Done

- NNUE - only update stuff that changed, from computer Shogi
- Similar: Martin’s “historical” technique for Deepire
- Use off-the-shelf software to apply further NN optimisations
- For example, quantisation
  - But my card doesn’t have this…
- Persistent server architecture: can’t figure out how to do IPC fast enough

Recap

- Worth paying attention to performance, can be an easy win
- Asynchronous evaluation of NN avoids stalling inference
- Compiling weights into ATP is horrible but has many advantages
- Hardware acceleration worthwhile
  - Significant effort to do manually
  - But mostly learning how to do a new kind of programming
  - Probably easier the more you do it?

Questions