Software R&D for Next Generation of HEP Experiments, Inspired by Theano

Amir Farbin
University of Texas at Arlington
Goal

• How can Data Science ideas and tools address software problems in HEP?
  • I’ll describe some of these problems.

• Paint a picture of what HEP software can look like in 10 years (for HL-LHC, LBNF/DUNE, ILC).
  • Highly speculative…

• 3 R&D directions with physics deliverables
  • LHC General Search with MEM/DNN
  • DNN Event Classification in LArTPC
  • Tomographic Reconstruction in LArTPC
Physics Landscape

- **Europe:** *LHC at Energy Frontier:* World’s most energetic proton-proton machine.
  - Found the Higgs in Run 1…
  - Next goals:
    - Test naturalness (Was the Universe and accident?) by searching for New Physics like Supersymmetry.
    - Find Dark Matter (reasons to think related to 1)
    - Study the SM Higgs find new Higgses
  - Run 2 at higher energy now.
  - Run 3 at higher luminosity by end of decade.
  - High Luminosity- LHC by 2025.
  - 100 TeV Machine later in the century? (In China?)
Physics Landscape

- **US:** Long Baseline Neutrino Facility (LBNF)/Deep Underground Neutrino Experiment (DUNE) at Intensity Frontier

  - Shoot intense neutrino beam through earth at a Near and Far (1300km) detector.

- Physics Goals:
  - Study Neutrinos, especially Charge Matter Violation (Why is there Matter in the Universe?)
  - Supernova
  - Proton Decay
  - Dark Matter

- Liquid Argon Time Projection Chambers (LArTPC) detector technology.

- Short Base Line program and LArTPC R&D until ~2020. (Many experiments ~ 100 Ton)

- Beam to 10 kiloton DUNE in 2025…

- Gradually expand to 40 kilotons and run for 30 years.
Physics Landscape

- **Europe:** *LHC at Energy Frontier*
- **US:** *LBNF/DUNE at Intensity Frontier*
- **Japan:** *International Linear Collider (ILC):* Most energetic $e^+e^-$ machine.
  - Japanese will hopefully build this in 2020s.
  - Precision studies of Higgs and hopefully new particles found at LHC.
  - High granularity Silicon Tracking and Digital Calorimeters.
Processor Landscape

• Parallelism taking off:
  • Multi-core CPUs, with built in GPUs.
  • Many-core GPU/MiCs
  • OpenCL Programmable FPGAs (and ASIC?)
    • Potential to easily move algorithms from software to hardware.
• On the horizon:
  • CPU/GPU/RAM Stacking
  • CPU+FPGA on same Dye
  • Neuromorphic Chips
HEP Software Landscape

• Shift from Fortan to C++ in late 1990s.

• The biggest datasets until mid 2000s… industry now leads.

• Problems:
  
  • Extremely complicated C++ frameworks, data structures, …
  
  • Difficulty utilizing Multi-core CPUs and massively parallel GPU/MiC co-processors… or any future emerging technology.
  
  • Expensive: ATLAS software cost ~O(250 Million) CHF to build over 15 years…
    
    • starting from scratch (to deal with Parallelization problems) for Run 3 wasn't an option for ATLAS or CMS.
  
  • We cannot find developers to fill mission critical posts. Critical people get stuck in jobs…
  
  • We do not educate HEP PhDs in software… rely on talented people training themselves.
  
  • There is a culture that software isn’t physics… but electronics and hardware are!
    
    • e.g. we do not support software R&D.
Parallelization Problem

- **Multi-core CPUs**: we are quickly approaching 100’s of cores/CPU.
  - Currently relying on Embarrassingly Parallel nature of HEP data.
    - Filling CPU cores with independent instances of software.
    - Not practical to have 4 GB/core for 100’s of cores…
    - Not enough bandwidth to memory if every core needs to access different 4 GB.
    - C++ data structures make it difficult to take advantage of vectorization.

- **Many-core Co-processors (possibly within CPU dye):** GPUs/MiCs, FPGA, (ASICs?)
  - Requires *Data Parallelization* where (for example) many events are simultaneously processed in each algorithm. HEP frameworks designed to see 1 event at a time.
  - Difficult to code. Highly sensitive to optimization and hardware. Difficult to efficiently integrate with current software. Rapidly evolving ecosystem.
  - Mostly used in specialized systems like DAQ and Trigger. No good solution for offline.
Looking Ahead...

• Concurrency (simultaneously processing many events) is a hot topic. 2 types
  
  • *Task Parallel*: Many threads, each processing one event.
  
  • *Data Parallel*: Algorithms processing many events at once.

• The LHC experiments are confronting this issue. Current focus on Task Parallelism:
  
  • CMS already has multi-threaded ART.
  
  • ATLAS using plans to build on Gaudi-Hive for Run 3.
  
  • There are schemes to push some algs to co-processors… but not ideal.

• Experiments will have lifetime of decades (e.g. 30 years for DUNE). We need to insulate ourselves from architecture transitions.

• *My opinion*: We need new frameworks on the time-scale of HL-LHC, DUNE, ILC.

General Search

and

Matrix Element Method (MEM)

Define: Process and Final State
<table>
<thead>
<tr>
<th>Method</th>
<th>Physics</th>
<th>Optimization/Training</th>
<th>Calibration</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut and count</td>
<td>Encapsulated in handcrafted features, based on intuition or simulation studies of signal and background topologies</td>
<td>Find regions in feature space that maximize significance or reach in Monte Carlo, in exchange for efficiency.</td>
<td>Parameterize/assess performance in Simulation or Validation regions in Data (with known physics),</td>
<td>Negligible Computation</td>
</tr>
<tr>
<td>ML (e.g. shallow NN or BDT)</td>
<td>Use same features as input. Output is usually a new feature used in another method.</td>
<td>Typically on smallish samples.</td>
<td>Same.</td>
<td>Negligible Computation</td>
</tr>
<tr>
<td>Maximum Likelihood Fit/Probability Density Estimators</td>
<td>Estimate prob of observing features for given hypothesis.</td>
<td>Analytically Parameterize, histogram, or use Kernel Estimators. No cuts/efficiency loss.</td>
<td>Same + fit nuisance parameters, with additional toy MC studies to assess bias in method.</td>
<td>Addressed in early 2000’s by smartly caching partial computations (RooFit). GPU Parallelized more recently (GooFit).</td>
</tr>
<tr>
<td>MEM</td>
<td>No features. Physics is baked in. Misses some effects (NLO/NNLO, Jet/parton match).</td>
<td>Detector effects encapsulated in parameterized response function models. No cuts.</td>
<td>Must calibrate away imperfections in response functions and NLO/NNLO.</td>
<td>Integration over phase space and convolutions are extremely computationally intensive.</td>
</tr>
</tbody>
</table>
General Search

• After the Higgs discovery, the primary mission of the LHC is to test Naturalness and search for Dark Matter and new Higgs(es). Basically look for new particles.

• Targeted Searches: Pick a specific New Physics (NP) model or sets of related NP models, look at one or few final state(s) (e.g. an interpretation of observed particles in event, 2 jets + lepton + MET) and search exclusively/inclusively.

• General Searches (aka Model Independent/Unspecified/Signature-based searches)
  • Cast a big net: Look at large number of final states (697 ATLAS/250 CMS).
  • Usually just search for deviation from SM and do not target specific NP models.
  • To be practical, General Searches small set of test statistic(s)/features are chosen that are computable for any final state.
    • Detail attention to each final state/topology would be analogous to performing all targeted analyses in the experiment and more…
    • Not optimized for any final state. Less sensitive than targeted search… but:
      • A small excess can prompt a targeted search.
      • Can discover models with small excesses in many final states.
    • Both ATLAS and CMS used 3 generic “features” (\(H_T\), MET, [Transverse] Invariant Mass, \(M_{\text{eff}}\))
General Search Features

- *Recursive Jig-saw*: New set of features that are final state/topology specific, but still easy to generalize/compute.
  - A basis of highly discriminating uncorrelated observables.

- MEM and Deep Neural Nets (DNN) with 4-vectors as input
  - Estimate probability of an observed final state belonging to all relevant processes.
  - MEM is maximally sensitive (caveat: detector effects, NLO, jet/parton match, …)
  - But MEM and Supervised-trained DNNs need some signal models.
    - *Simplified Models* can serve as a way to get back some model independence.
      - Minimalistic models of signal processes on to which more sophisticated models can mapped.
      - Small model parameter space small (2D or 3D) but larger than SM top and Higgs MEM analyses.
    - Of course, we already produce a huge number of NP models… just use them.

- Perhaps high-level features from unsupervised DNN can be mapped to SM simulation. Then search for a cluster of high-level features not associated with any SM process.
General Search with DNN or MEM

• DNN

• Training on huge sample will require a lot of CPU/GPU. Good use of Supercomputers (HPCs).

• Requires large independent training samples of full simulated events, which may be prohibitively expensive. No help from HPCs.

• MEM- Already running on GPUs (see backup), so can run on Supercomputers (HPCs).

• A Very Bad Estimate: \(10^{10}\) Events/Year \(*\ O(1)\) Final State/Event \(*\ O(10^3\ or\ 4)\) Processes/Final State \(/\ 10^3\) Possible Final States = \(10^{12\ or\ 13}\) MEM Computations.

• \(10^{12\ or\ 13}\) MEM Computations \(*\ 1\) Week \(/\ 18,688\) GPUs in Titan = \(O(10^{-2\ or\ -3})\) seconds per MEM calculation.

• MEM Calc time very is process dependent… hard to estimate.

• Example MEM on GPU times \~0.1\ sec/event for 4 param integration, \~5\ sec/event for 8 parameter.

• Possible if less ambitious, i.e. cut events out or consider less processes.
Can MEM go faster?

- We’ll get some speed up as GPUs get faster.

- Hardware matters:
  
  - AMD GPU : Nvidia GPU : 2x 12 Core Xeon : Phi (Knights Corner) = 1 : 3 : 6 : 10. (See backup)

- Currently GPU utilization for MEM ~ 10-20%. (See backup)

- If we parallelize the integration and optimize carefully, we may get to 100% GPU utilization, resulting in 5-10 times speed up.

- My idea inspired by Theano: symbolic simplification, optimization, and code generation…

- Reduce the number of operations.
Math in Python

- **numpy**: Matrix manipulation like matlab
  - C=A*B performs a computation on numbers in A and B matrixes

- **sympy**: Symbolic manipulation like mathematica
  - C=A*B ; D=A^{-1} \ C ==> D=B

- **Theano**:
  - Symbolic representation and operations (e.g. derivatives)
  - Based on Tensors with numpy-like functionality
  - Computation tree optimization
  - Transparently compiles into CPU, OpenMP, CUDA, and OpenCL.
    - Many missing/non-optimal features in GPU implementation
  - Provides a framework for implementing new operations, optimizations, and backends.
    - Provides an environment built for optimizing calculations on CPUs and GPUs.
  - Why? instead of writing code to perform your calculation, use these systems to write down the mathematical expressions... and they will generate *optimized* code for any hardware.
MEM with Theano

• Numeric problems can be accelerated through

  • *symbolic preprocessing*: *e.g.* instead of numerical computation, first simplify the expression, take symbolic derivatives, analytically solve for minimum.

  • *computation optimization*: compute things only once, memory coalesce, loop re-ordering, vectorization, use optimized libraries …

• Acceleration of Matrix elements:

  • Since they are built from Feynman rules, they can be simplified if amplitude expression is built symbolically.

    • MadGraph calculates amplitudes by combining HELAS/HEGET/ALOHA's numerical computation of the wave-function components for each piece of diagrams, missing simplifications (e.g. multiply by $p$ then divide by it a bit later).

  • Diagrams for processes are related by permutations and leg replacements:

    • Naive approach calculates same thing many times… Madgraph is a bit smarter.

    • Theano detects these and optimizes the calculation automatically.
Demonstration


- Note interesting LHC processes have \( O(100) \) diagrams.

- Focusing on ME evaluations only... no integration, change of vars, etc...

- I count \( \sim 500 \) real and 500 complex numerical operations in the ME calculation (\( \sim 1500 \) total).

  - c-code → python → Theano reduces operations to \( \sim 1000 \).

  - c-code → python → sympy → Theano reduces the operations to 321 (116 on GPU, but I can’t compile!)

- \( 5.1x \ (4.6x) \) faster per event single thread computation time float (double).
Issues to Address

- Sympy hangs on amplitude simplification.
- Theano was optimized for Machine/Deep Learning, not MEM.
- Theano's Temporarily Complex variables are always Complex128.
- No Complex support in CUDA or OpenCL backends, so I can't generate GPU code.
  - Was able to run on GPU by computing real and complex parts separately… but very inefficient.
- OpenCL Backend very minimal.
- Looking at code, GPU code generation is optimized for matrix operations… not so much for element-wise operations.
  - Clearly GPU code generation needs work.
- Theano can use some skilled hands under the hood.
Other Speed Ups

• Full spinor/helicity symbolic representation may be more efficient and easier to simplify.

• Small numbers and unnecessary operations sometimes necessitate use of doubles instead of floats:
  • Theano removes unnecessary operations

• Change of phase-space variables to optimize Integration is currently a bottleneck.
  • Can be done symbolically and ME expressed in terms of new variables.

• If integration is also done with Theano, loops can be easily reordered to optimize GPU memory use.
  • Phase space can be smartly reused.
  • MEM for some processes can be computed from cached parts of other MEM computations.

• O(100) times speed up wrt current GPU implementations seems attainable. Already have O(10).

• Interested? Come talk to me… I have a full roadmap, starting with some easy low hanging fruit like Event Generation using GPUs on HPCs.
GS with MEM Schematic

Acceptance, Efficiency, PID E, Fake Rate, Resolution, ...

Response Function Models

Final State

Particle ID/Overlap Removal

Interpretation 1

Interpretation 2

Interpretation 3

Interpretation 4

Event 501

Tag Data Base
Event 501: \{P_4(), P_{21}(), P_{45}(), \ldots\}
Event 502: \{P_2(), P_{18}(), P_{95}(), \ldots\}

Similar Diagram for DNN
General Search Summary

- Apply experimental PID, cleaning, overlap removal recipes and classify events all LHC events into final state categories.

- Ship 4-vectors and other relevant quantities to your favorite HPC.

- Compute MEM Likelihood and/or DNN Probability of every LHC event belonging to every hypothesis (i.e. process) generated in the Experiment's Simulation Samples.

- Store results in a [Tag] Database which references back to the original events.
  - Give the whole experiment access.
  - Suddenly, every analyzer has new super powerful observables for their targeted analysis. Doesn’t have to be perfect…
  - Physics Monitoring: Quick scans can ID excesses to follow up with new targeted searches.
  - Use the usual full simulation sample and standard systematic procedures to calibrate and estimate errors.
  - Develop General Search that can make statements about New Physics using full ensemble of LHC events.
LArTPCs
DNN Reconstruction
In neutrino experiments, try to determine flavor and estimate energy of incoming neutrino by looking at outgoing products of the interaction.

**Typical neutrino event**

**Incoming neutrino:**
- Flavor unknown
- Energy unknown

**Outgoing lepton:**
- Flavor: CC vs. NC, $\mu^+$ vs. $\mu$, e vs. $\gamma$
- Energy: measure

**Mesons:**
- Final State Interactions
- Energy? Identity?

**Target nucleus:**
- Nucleus remains intact for low $Q^2$
- N-N correlations

**Outgoing nucleons:**
- Visible? Energy?
Liquid Argon is an excellent choice for neutrino detectors:

- **LArTPC**
- **Neutrino interaction in LAr produces ionization and scintillation light**
- **Drift the ionization charge in a uniform electric field**
- **Read out charge and light produced using precision wires and PMTs**

**Future prospects of electron/photon separation & Neutral Current measurements with Liquid Argon**

**Jonathan Asaadi**

Tracking, Calorimetry, and Particle ID in same detector.
Image Processing

Jessica Esquivel
Projections

Liquid Argon Time Projection Chamber

ArgoNeuT Data

Pixel size: 4mm x 0.3mm

2 Wire Planes

1.6 GeV nue CC in FD

DUNE

3 Wire Planes

μν +/ - γγ γ γ

νe appearance searches

→ Notoriously difficult topology to reconstruct

μν

→ Particularly insidious background for oscillation searches and cross-section measurements comes from neutral current π0 production.

→ The ArgoNeuT detector is too small to contain the majority of photon showers produced from π0's.

→ However, it may still be possible to utilize this data and look for NC π0 production.

→ Select a sample of events likely to be neutral current

→ Require no track matched to MINOS ND

→ Require at least to clusters of energy found in each view

→ Require a reconstructed vertex in the detector

→ ...
Ambiguous Data

Is this cat going up or down?

Each eye collects 2D data (as a function of time) but we interpret in 3D

Tom Junk
• Correlating Projections leads to full 3D Reconstruction.

http://www.phy.bnl.gov/wire-cell/bee/set/7/event/0

http://www.phy.bnl.gov/wire-cell/bee/set/6/event/24

Chao Zhang, Xin Qian, Brett Viren
LArTPC Reconstruction

- Neutrino Physics has a long history of *hand scans*.
  - QScan: ICARUS user assisted reconstruction.
  - Full automatic reconstruction has yet to be demonstrated.
  - LArSoft project: art framework + LArTPC reconstruction algorithm, started in ArgoNeuT and contributed to/used by many experiments.
  - Ideally suited for DNN-based reconstruction
  - Just need to know what type of event (classification) and the energy of the neutrino (regression).

![Flowchart](image)

**Selection of $\nu_e$ events**
- Reference points and vertices can be defined to mark interesting features of the event in a 2D view (primary interaction, delta rays, decay point of tracks, shower features, muon begin/end point for the momentum measurement via MCS); they can be selected manually in Qscan and can be associated to clusters and matched between different views providing additional input to 3D reconstruction;
- An automatic tool for the primary vertex identification is available;
- Reference points and vertices can be saved in root files.

![Event Display](image)
DNN “Reco” Proof of Principle

- The DUNE-35 ton test experiment will soon begin collecting cosmic ray data.

- High cosmic rate, we are recording 1/2 of the time.
  - More discriminating trigger would be nice, but not necessary.

- Stopped muons, radiated photons, horizontal cosmics, … are small but very useful subset of data.

- How do we make samples of these w/o full reconstruction?

- I faked some events passing/stopped muon events, fed it to NVidias DIGITS, which is a DNN image classification tool.
  - Really just a very simple proof of principle test.
DNN Classification of “Raw” LArTPC Data

GoogleLeNet 256x256

1-4 Tracks With or without noise, DNN correctly classifies ~90-99%
DNN LArTPC Event Classification

• Very instructive first exercise for me… Observations:

  • Training with noisy events failed. But training with no noise events, then noisy events worked great.

  • Training on 4 track sample better after train on 1 track sample first.

• Expect fully simulate events to work… stay tuned.

• Just a toy for now, obviously a more serious effort necessary.

  • Is it necessary to try different approaches/optimize hyper parameters if it already works really well?

• Can be applied to 35 ton data within next few months.
DIGITS

- Great way to play.
- Essentially a web server interface to a batch system. Multi-GPU support.
- Only image classification (for now?).
- Great potential of evolving to more a general tool that will also make DNN accessible to everyone.
  
  - A graphical model editor would be awesome.
DNN Reco

- Motivations?

  - Hopefully DNN-based feature extractors *out perform* had crafted reconstruction algorithms.

  - After training, DNN-based will likely to be much *faster* than algorithmic reconstruction. And it’s already running on GPUs.

  - There is incredible value (CHF/dollars) in the fact that DNN may allow performing reconstruction *without physicists writing algorithms*.

- Maybe instead of writing reconstruction software, new workflow:

  - Train DNN on Simulated Data, perhaps starting with simplified training samples and work towards full complexity. Try different NNs, search hyper-parameters, …

  - Calibrate with the full standard simulation samples of the experiment.

  - If some sub-class of events not are well “reco’d”, add addition training sample. Iterate.

  - Apply to data, perhaps compare to hand scan and use re-enforcement training.

- LArTPC data and Neutrino physics is extremely well suited for this paradigm and a good stepping stone to DNN Reco for LHC.
LArTPC Tomography
Tomography

• Probably most familiar with CAT scans
  • Take 360 degrees of X-rays
    • Measures attenuation
  • Reconstruct voxels.
• Can be applied to TPCs, measuring charge instead of attenuation.

Fig.1: Basic principle of tomography: superposition free tomographic cross sections S1 and S2 compared with the projected image P

https://en.wikipedia.org/wiki/Tomography
WireCell

• Developed by Chao Zhang, Xin Qian, and Brett Viren at BNL.
  • http://www.phy.bnl.gov/wire-cell/

• The challenge in TPCs comes from the wire readout (compared with the pad pixel readout)
  • Wire readout is necessary to reduce cost
  • However the measured information is reduced from $N^2$ (pixels) to $3N$ (wires)
    • Information lost -> exponential degeneracy
  • Use fact that same charge seen every wire to help resolve.

New Input From Charge Information

Assumption: good charge calibrations on all three wire planes

Write down the charge matrix equations

If equations can be solved, fake charge will be close to zero
**Example: a 1.5 GeV electron**

- **Truth**
- **rec_simple** (Use only geometry information)
- **rec_charge** (Use geometry and charge information)
Procedure

1. Form time slices
2. Construct Wire-Cell association
3. Merge adjacent cells into “blobs”
4. Construct $\chi^2$ through matrix equations
5. Obtain best matched 3D space points through $\chi^2$ minimization
Computation

\[ \chi^2 = (B \cdot W - G \cdot C)^T V_{BW}^{-1} (B \cdot W - G \cdot C) \]

\[ \frac{\partial \chi^2}{\partial C} = 0 \rightarrow G^T V_{BW}^{-1} (BW - GC) + (BW - GC)^T V_{BW}^{-1} G = 0 \]

\[ G^T V_{BW}^{-1} BW + W^T B^T V_{BW}^{-1} G = G^T V_{BW}^{-1} GC + C^T G^T V_{BW}^{-1} G \]

\[ C = (G^T V_{BW}^{-1} G)^{-1} G^T V_{BW}^{-1} BW \]

- \( C \): charge in each (merged) cell (to be solved)
- \( G \): Geometry matrix connecting cells and wires
- \( W \): charge in each single wire
- \( B \): Geometry matrix connecting merged wires and single wires
- \( V_{BW} \): Covariance matrix describing uncertainty in wire charge

- Prototype implementation takes 1 hour to 1 day / event to perform \( \chi^2 \) minimization to get \( G \cdot C \), and Markov Chain MC to get \( C \).

- Obvious candidate for parallelization on GPUs.

- Theano implementation would simply write down the \( \chi^2 \)
  - Provide easy means to get a parallelized GPU version.
  - Provide environment to easily further develop the technique.
  - New idea? Just change the equation. Focus on the method not implementation.
Theano

• Might be trivial to implement some algorithms with Theano.

• Anything you can write as a formula can be easily expressed in Theano and automatically optimized.

• Many things are already implemented.

• For example, Kalman Filter (from: http://matthewrocklin.com/blog/work/2013/04/05/SymPy-Theano-part-3/)

```python
from sympy import MatrixSymbol, latex
n = 1000  # Number of variables in our system/current state
k = 500   # Number of variables in the observation
mu = MatrixSymbol('mu', n, 1)  # Mean of current state
Sigma = MatrixSymbol('Sigma', n, n)  # Covariance of current state
H = MatrixSymbol('H', k, n)  # A measurement operator on current state
R = MatrixSymbol('R', k, k)  # Covariance of measurement noise
data = MatrixSymbol('data', k, 1)  # Observed measurement data

ewmu = mu + Sigma*H.T * (R + H*Sigma*H.T).I * (H*mu - data)  # Updated mean
newSigma= Sigma - Sigma*H.T * (R + H*Sigma*H.T).I * H * Sigma  # Updated covariance

inputs = [mu, Sigma, H, R, data]
outputs = [newmu, newSigma]
dtypes = {inp: 'float64' for inp in inputs}

from sympy.printing.theanocode import theano_function
f = theano_function(inputs, outputs, dtypes=dtypes)
import numpy
ninputs = [numpy.random.rand(*i.shape).astype('float64') for i in inputs]
mu, Sigma = f(*ninputs)
```
A Vision of Future HEP Software
Wish List

- Reconstruction closely integrating:
  - Traditional Algorithms like ones in HEP SW today.
  - Deep Neural Networks (and other ML Techniques)
    - Automatic training/monitoring (e.g. for reproducing training in every release)
    - NN visualization (structure and weights), Hyper-parameter scans.
  - Image processing algorithms
  - Event Display / Hand Scan (e.g. for re-enforcement training)
- Data structures optimized for architecture and computation, with automatic data transformations.
- Algorithms can process many events at once.
- Automatically optimized for all/any CPU or GPU architectures. Future proof.
- Allow physicists to focus on the method and performance not implementation.
  - Easier to hand off problem to professional programmers.
Theano-like Framework?

• Today with Theano

  • A Physicists can write down math expression for their computation or algorithm. Theano auto optimizes…
    • no computing expertise necessary.

  • Can pass expression to professionals who tune optimization/code generation in Theano
    • No physics understanding necessary.

  • Code generation can be optimized for each architecture.

• Naively, we should consider completely different approach to writing software:

  • High level description of algorithms/data by physicists (new language?)
    • The representation of the data and the implementation of the computation is changeable.

    • May require a functional language.

  • Automatic analysis of the computation graph and targeted code generation, developed/optimized by experts
Theano vs Compiler

• Isn’t this what a compiler does? Maybe… I’m out of my league here…

• For MEM, compiler sees function calls, not that the result of those calls is an expression which can be greatly simplified and optimized.

• Can a complier realize that it is more efficient to first run muons reco for next 1000 events because magnetic field and geometry are in GPU memory, and then move to jets?

• RooFit/GooFit Example: Theano can reproduce the RooFit caching mechanism from the Likelihood Expression and generate GPU code.
Weaving-in DNN Reco

Feature List = \{Hit_1, Hit_2, \ldots\}

Feature Map = DNN Combined Reconstruction

Simultaneously Train All DNNs
Summary

- HEP Software Frameworks are expensive and will soon be obsolete.
  - Inspiration from Theano:
    - Let the physicist worry about the problem and solutions, not the implementation or architecture.
    - Auto optimize and code generation by professionals.
- ME on GPUs is low hanging fruit with huge PR value, allowing a real utilization of HPCs for HEP event generation.
- We should plan on General Searches with full Run 2 data.
  - Providing MEM results to collaboration computed on HPCs allows physics monitoring and more sensitive targeted searches.
  - Using MEM Likelihood ratios can provide nearly maximal sensitive general search. DNN also good?
- Deep Learning can help alleviate our software problems. Instead of laboriously developing some algorithms, we can train networks that are fast to apply and maybe even more performant.
- Present and imminent LArTPC experiments need help with automatic reconstruction and are a great stepping stone for:
  - Developing DNN-based Reconstruction for LHC.
  - Trying out imaging techniques, for example from Medical Physics.
• For a small experiment like 35 ton, where the manpower is small and software/hardware not quiet ready, DNN can be useful today.
HPCs for LHC

• Now possible to send ATLAS jobs to HPCs. Great PR, but none of our software uses GPUs!

• Low Hanging Fruit: GPUs for Event Generation
  
  • O(10%) of our CPU usage can be shifted to opportunistic use of HPCs
  
  • NLO and NNLO becoming important. Taking increasingly longer to generate.
  
  • Easier to explore large parameter space of new physics models (e.g. MSSM)
  
  • HELAS authors trivially converted functions to GPU… (starting 2009- http://arxiv.org/abs/0908.4403)
    
    • Got O(100) faster computation wrt 1 CPU thread.
    
    • Fully demonstrated and validated in with MadGraph 4, but never release and never incorporated in MadGraph 5.
  
• Easy to do, with big pay off… and a stepping stone for Matrix Element Method.
<table>
<thead>
<tr>
<th>Method</th>
<th>Physics</th>
<th>Optimization/Training</th>
<th>Scale Dependence</th>
<th>Calibration</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut and count</td>
<td>Encapsulated in features, based on intuition or simulation studies of signal and background topologies</td>
<td>Find regions in feature space that maximize significance or reach in Monte Carlo, in exchange for efficiency.</td>
<td>High LHC SM rates necessitate Jet/lepton $p_T$ cuts sacrificing sensitivity to some models. ($\alpha_T$, Razor, Jig-saw)</td>
<td>Parameterize performance in Simulation or Validation regions in Data (with known physics),</td>
<td>Negligible Computation</td>
</tr>
<tr>
<td>ML (e.g. shallow NN or BDT)</td>
<td>Use same features as input. Output is usually a new feature used in another method.</td>
<td>Typically on smallish samples.</td>
<td>Same.</td>
<td>Same.</td>
<td>Same.</td>
</tr>
<tr>
<td>Maximum Likelihood Fit/Probability Density Estimators</td>
<td>Estimate prob of observing features for given hypothesis.</td>
<td>Analytically Parameterize, histogram, or use Kernel Estimators. No efficiency loss.</td>
<td>In principle, looser cuts means less scale dependence. Every event helps, even when signal prob low.</td>
<td>Same + nuisance parameters, with additional toy MC studies to assess bias in method.</td>
<td>Addressed in early 2000's by smartly caching partial computations (RooFit). GPU Parallelized (GooFit).</td>
</tr>
<tr>
<td>Deep NN</td>
<td>Learns features. Enhanced by engineered features.</td>
<td>Very computationally intensive to produce training samples and to train.</td>
<td>No cuts?</td>
<td>Must be calibrated, same as above.</td>
<td>Negligible Computation</td>
</tr>
<tr>
<td>MEM</td>
<td>No features. Physics is baked in. Usually only LO. Jet/Parton matching pitfalls?</td>
<td>Detector effects encapsulated in response functions.</td>
<td>No cuts.</td>
<td>Must calibrate away imperfections in response functions and NLO/NNLO.</td>
<td>Integration over phase space and convolutions are extremely computationally intensive.</td>
</tr>
</tbody>
</table>
MEM vs DNN

• ME encapsulates all physics features so most features are unnecessary.
  • Usually only LO
  • Parton/Jet Matching?
  • No need for hand crafted features.

• Detector effects are encapsulated in response functions.

• No Training, but it is time consuming to calculate.

• NLO/NNLO and detector effects can be calibrated away.
  • What if MEs were easy and quick to calculate?
  • Feed MEs into DNN?

• In principle, a DNN can be trained to reproduce an ME, thereby encapsulating the physics.
  • Can be trained with NLO/NNLO, so better than a LO MEM.
  • Can be augmented with hand crafted features.

• It will simultaneously capture physics and detector, therefore also optimizing on detector.
  • It is time consuming to train, needs huge (independent?) simulation sample which is expensive to produce, but quick to apply.
  • Must also be calibrated.
Accelerating MEMs with GPUs


- Wrote Madgraph5 plug-in to auto-generate GPU code.

- Showed O(50)x improvement over single CPU thread.

- But for real calculation had 10-20% GPU usage, because
  - Vegas MC Integration on CPU.
  - Solving Phase-space equations was inefficient.
  - Lots of temporarily variables depleted the number of available registers.
    - Optimization here has significant architecture dependencies
  - Need to use doubles at times for numerical stability.
ME Computation on GPUs

- About 1 year ago I hand converted $u\bar{u} \rightarrow 3$ gamma matrix element
- Performed first comparison on multi platform/hardware.
- Timing just evaluation of ME, no integration, change vars, etc.

\[ \text{Uux3a} \]

Requirements careful block right sizing to get best performance and 100% compute device usage.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Platform</th>
<th>ns per event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x Intel Xeon E5-2695</td>
<td>Intel OpenCL</td>
<td>6.3</td>
</tr>
<tr>
<td>2 x Intel Xeon E5-2695</td>
<td>AMD OpenCL</td>
<td>29.2</td>
</tr>
<tr>
<td>Nvidia Tesla K20Xm</td>
<td>Nvidia OpenCL</td>
<td>3.2</td>
</tr>
<tr>
<td>Nvidia Tesla K20Xm</td>
<td>Nvidia CUDA</td>
<td>3.0</td>
</tr>
<tr>
<td>Nvidia GTX780</td>
<td>Nvidia OpenCL</td>
<td>2.8</td>
</tr>
<tr>
<td>Nvidia GTX780</td>
<td>Nvidia CUDA</td>
<td>2.7</td>
</tr>
<tr>
<td>Intel Xeon Phi 7120</td>
<td>Intel OpenCL</td>
<td>10.6</td>
</tr>
<tr>
<td>AMD Firepro W9100</td>
<td>AMD OpenCL</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- Surprise, AMD ~3x faster than Nvidia!
- Similar performance in CUDA and OpenCL.
- 2x 12 core CPUs are only 2x (6x) slower than Nvidia (AMD) GPU.
  - Note each CPU ~$2400.
  - Tesla or Firepro ~$4500
  - GTX780 ~$500
Profiling/Optimization Framework

- Work by undergraduate (Zubair Bhatti)...
  - https://github.com/zbhatti/dptm/tree/master/kernelProfiler
- Wraps OpenCL/CUDA kernels
  - Scans block size to find optimal configuration (for every hardware, kernel, and input data size).
    - Sometimes using less cores better… e.g. kernels using too many registers.
    - Compare optimal performance between hardware.
- Comparison of 19 algorithms on GIT page.
- Surprising observations:
  - AMD GPUs ~3-4x faster and NVidia on certain tasks (e.g. Monte Carlo)
  - 2x 12 core Xeon with OpenCL is sometimes just ~ 2x slower than Nvidia GPUs.
  - OpenCL on Xeon much faster than Single Thread time / # threads
  - Architecture dependent optimization is critical
## Matrix Element Time/Event

<table>
<thead>
<tr>
<th>Compulation</th>
<th>Events</th>
<th>Time (s)</th>
<th>Time (ns/event)</th>
<th>Time (ns/event)/48</th>
<th>GFOLPs</th>
<th>Max Time (ns/event)</th>
<th>NOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone Xeon1 32 bit</td>
<td>1048676</td>
<td>6.845</td>
<td>6527.28</td>
<td>135.98</td>
<td>19.1875</td>
<td>78.18</td>
<td>1500</td>
</tr>
<tr>
<td>Standalone Xeon1 64 bit</td>
<td>1048676</td>
<td>8.67</td>
<td>8267.57</td>
<td>172.24</td>
<td>9.59375</td>
<td>156.35</td>
<td>1500</td>
</tr>
<tr>
<td>Standalone CLXeon</td>
<td>16777216</td>
<td>0.198</td>
<td>11.80</td>
<td>19.1875</td>
<td>921</td>
<td>1.63</td>
<td>1500</td>
</tr>
<tr>
<td>S2T Xeon1 (32 bit)</td>
<td>30</td>
<td>1.35</td>
<td>1287.46</td>
<td>26.82</td>
<td>19.1875</td>
<td>78.18</td>
<td>1500</td>
</tr>
<tr>
<td>S2T Xeon1 (64 bit)</td>
<td>30</td>
<td>1.89</td>
<td>1802.44</td>
<td>37.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC Xeon1</td>
<td>897</td>
<td>3.544</td>
<td>3379.82</td>
<td>70.41</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CLXeon gpusarray</td>
<td>954.05</td>
<td>80</td>
<td>19073.49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CLXeon gpusarray (Real)</td>
<td>954.05</td>
<td>58.3</td>
<td>13899.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CLXeon gpusarray (Imaginary)</td>
<td>954.05</td>
<td>56.3</td>
<td>13422.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standalone CL780</td>
<td>16777216</td>
<td>0.049</td>
<td>2.92</td>
<td>4000</td>
<td>0.38</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>S2T AC CU780</td>
<td>914</td>
<td>0.2</td>
<td>47.68</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CU780 gpusarray</td>
<td>930.5</td>
<td>0.2</td>
<td>47.68</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CL780 gpusarray</td>
<td>992</td>
<td>0.64</td>
<td>152.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CU780 gpusarray Real</td>
<td>930.5</td>
<td>0.15</td>
<td>35.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2T AC CU780 gpusarray Imaginary</td>
<td>930.5</td>
<td>0.13</td>
<td>30.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standalone CLK20X</td>
<td>33554432</td>
<td>0.103</td>
<td>3.07</td>
<td>4000</td>
<td>0.38</td>
<td>1500</td>
<td></td>
</tr>
</tbody>
</table>

Open CL on CPU is fast!
Working with Theano

• Easy to switch from C to Python to Theano to Sympy, etc…
  • Just build your expression with python functions and feed the different objects for different versions.
    • consider: def f(x): x*x
      • python: y=f(2) -> y=4 (regular python float)
      • sympy: y=f(x) -> y = symbolic rep
      • theano: y=f(x) -> y = symbolic rep
        • compute_y=function([x],y) optimizes/compiles
        • compute_y(2) -> 4
  • Various ways to convert sympy -> theano:
    • theano_function: takes a sympy expressions and translates it into Theano expression.
    • SymCFunc: creates efficient c-code for scaler expression which Theano can wrap…
      • The c-function can be faster.
      • But then Theano can’t optimize it.
  • Parallelization: tensor representation… numpy broadcasting for scaling.
  • Loops = contraction of indexes… makes reordering loops easy
  • Iteration = shared variables (keep state) and update mechanism.
General Search Features

• For simplicity Features that can be computed for every event are not necessary best for specific final state/topology: $H_T$, MET, Invariant Mass, $M_{\text{eff}}$, Sphericity, $\alpha_T$, Razor

• Some features are very final state/topology specific,
  • e.g. intermediate state masses and angles between particles in specific frames.
    • Difficult to use in General Search. Requires detail attention to each final state/topology... analogous to performing lots of targeted searches.
  • Jig-saw: Evolution of Razor by Chris Rogan and Paul Jackson.
    • A basis of highly discriminating uncorrelated observables.
    • In principle easy to generalize calculation for any topology/final state.

• MEM and DNN (with 4-vectors as input)
  • No features. Give probability of any observed final state belonging to any topology.
  • Maximally sensitive.
  • But they both need some signal models.
    • Simplified models can serve as a way to get some model independence.
      • Note signal model parameter space much larger than SM top and Higgs MEM analyses.
      • Of course, we already produce a huge number of NP models... just use them.
  • Perhaps unsupervised DNN training can identify a cluster of high-level features not associated with any SM process?
General Search with MEM

• Is it feasible to calculate ME-based prob for every LHC event?

• Very hard to make a good estimate, so I'll just make some wild guesses:
  
  • N events/year in Run 2 ~ 10 Billion = $10^{10}$
  
  • Each event assigned to one or two (i.e. O(1)) of $10^3$ possible final states. So $\sim 10^7$ events/final state.
  
  • N different hypothesis generated/simulated in ATLAS~ $10^5$. Each relevant $\sim 10$ final states (total guess). So $10^7$ processes/final state.
  
  • So that's $10^{13}$ MEM computations to be done on 18,688 GPUs in Titan.

• So if I want to do this in one week, the average time to compute ME prob for one process for one event $\sim 10^{-3}$ sec.

  • Bernd et al got 5.9-7 sec per event for 8 parameter integration
    
    • 4 jets + 2 lepton + 2 neutrino final state
    
    • 4 Jet resolution + 2 sharing of MET among neutrino + 2 neutrino z momentum (or struck parton momentum fractions)
    
    • I get 0.1 second/event for 4 parameter integration.

• So we are $\sim 3$ orders of magnitude slower than needed for EVERY hypothesis on EVERY event.

  • Or we can evaluate just 100 hypothesis per week… or make some loose cuts and reduce sample.

• I made a bunch of bad assumptions, for example: not all final states have invisible particles. Higher jet multiplicity…
Reconstructed cosmic muons in 35t 1.6 GeV nue CC in FD Kaon from simulated proton decay.

High resolution detector A lot of information!

**FD Simulation/Reconstruction Status**

- We have produced a FD neutrino sample, including beam events, nue events and nutau events.
  - `/pnfs/lbne/persistent/dunepro/v04_20_00/mergeana/prodgenie_nu_dune10kt_workspace/anahist.root`
  - Replace nu with nue and nutau to get other files.

- We ran reconstruction chain on those samples:
  - Signal processing, hit finder, cluster finder, track finder.

- We encourage people to start looking at those files and developing nue identification algorithms.
  - e.g. remove hadron/muon tracks and examine shower topology using hit information.

**2.3 GeV/c nue CC tracks**

- **vertex**
- **tracks**
- **π**
- **e**

**14 m.i.p.**

**The Electronic Bubble Chamber**

Alberto Guglielmi (ICARUS), Neutrino 2010: [http://indico.cern.ch/conferenceDisplay.py?confId=73981](http://indico.cern.ch/conferenceDisplay.py?confId=73981)
Typical $\nu_\mu$ CC event (Collection view)

muon is $\sim$13m long

Typical MIP signal in Coll.

Charged Current (CC)

$\nu_\mu$ $\rightarrow$ $W^-$ $\rightarrow$ $\mu^-$

$\nu_\mu$ $\rightarrow$ $Z^0$ $\rightarrow$ $\mu^-$

$\nu_\mu$ $\rightarrow$ Ar $\rightarrow$ $\pi^0$ $\rightarrow$ $\gamma$ $\gamma$

$1\mu + 1n$

$1\mu + 2p$

$\sim 7$ GeV deposited energy

ArgoNeuT: Neutral Current

$\pi^0$ analysis

J. Asaadi

The MicroBooNE & ArgoNeuT Experiments

νμ νμ Ar π0 γ γ

An interesting, and particularly important, channel for both oscillation searches and cross-section measurements comes from neutral current $\pi^0$ production.

→ Particularly insidious background for $\nu_\mu$ appearance searches

→ Notoriously difficult topology to reconstruct

The ArgoNeuT detector is too small to contain the majority of photon showers produced from $\pi^0$'s

→ However, it may still be possible to utilize the this data and look for NC $\pi^0$ production

→ Select a sample of events likely to be neutral current

→ Require no track matched to MINOS ND

→ Require at least to clusters of energy found

In each view

→ Require a reconstructed vertex in the detector

→ ...

ArgoNeuT Data
- Amplitude/Matrix Element (encodes all the physics) is a complex function of four-vectors

- Sum amplitude for many processes and square, you get the total probability of event originating from that process.

- MadGraph + HEGET/ALOHA are a code generator of these functions.

- Complex floating point calculations of tens of thousands or more operations per computation.

<table>
<thead>
<tr>
<th>Source</th>
<th>Event Rate</th>
<th>Event Size</th>
<th>Data Rate</th>
<th>Annual Data Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^{39}$Ar (ZS)</td>
<td>11.2 MHz</td>
<td>150 B</td>
<td>1.7 GB s$^{-1}$</td>
<td>53 PB</td>
</tr>
<tr>
<td>all in-spill</td>
<td></td>
<td></td>
<td></td>
<td>159 TB</td>
</tr>
<tr>
<td>with-beam-$\nu$</td>
<td></td>
<td></td>
<td></td>
<td>79 GB</td>
</tr>
<tr>
<td>cosmic-$\mu$ (ZS)</td>
<td>0.259 Hz</td>
<td>2.5 MB</td>
<td>647.4 KB s$^{-1}$</td>
<td>20 TB</td>
</tr>
<tr>
<td>beam-$\nu$ (ZS)</td>
<td>8770 year$^{-1}$</td>
<td>2.5 MB</td>
<td>0.164 MB s$^{-1}$</td>
<td>22 GB</td>
</tr>
<tr>
<td>beam-$\nu$ (FS)</td>
<td>8770 year$^{-1}$</td>
<td>24.9 GB</td>
<td>7 MB s$^{-1}$</td>
<td>218 TB</td>
</tr>
<tr>
<td>SNB cand. (ZS)</td>
<td>12 year$^{-1}$</td>
<td>16.7 GB</td>
<td>6366 B s$^{-1}$</td>
<td>201 GB</td>
</tr>
<tr>
<td>SNB cand. (FS)</td>
<td>12 year$^{-1}$</td>
<td>46.1 TB</td>
<td>17.5 MB s$^{-1}$</td>
<td>553 TB</td>
</tr>
</tbody>
</table>