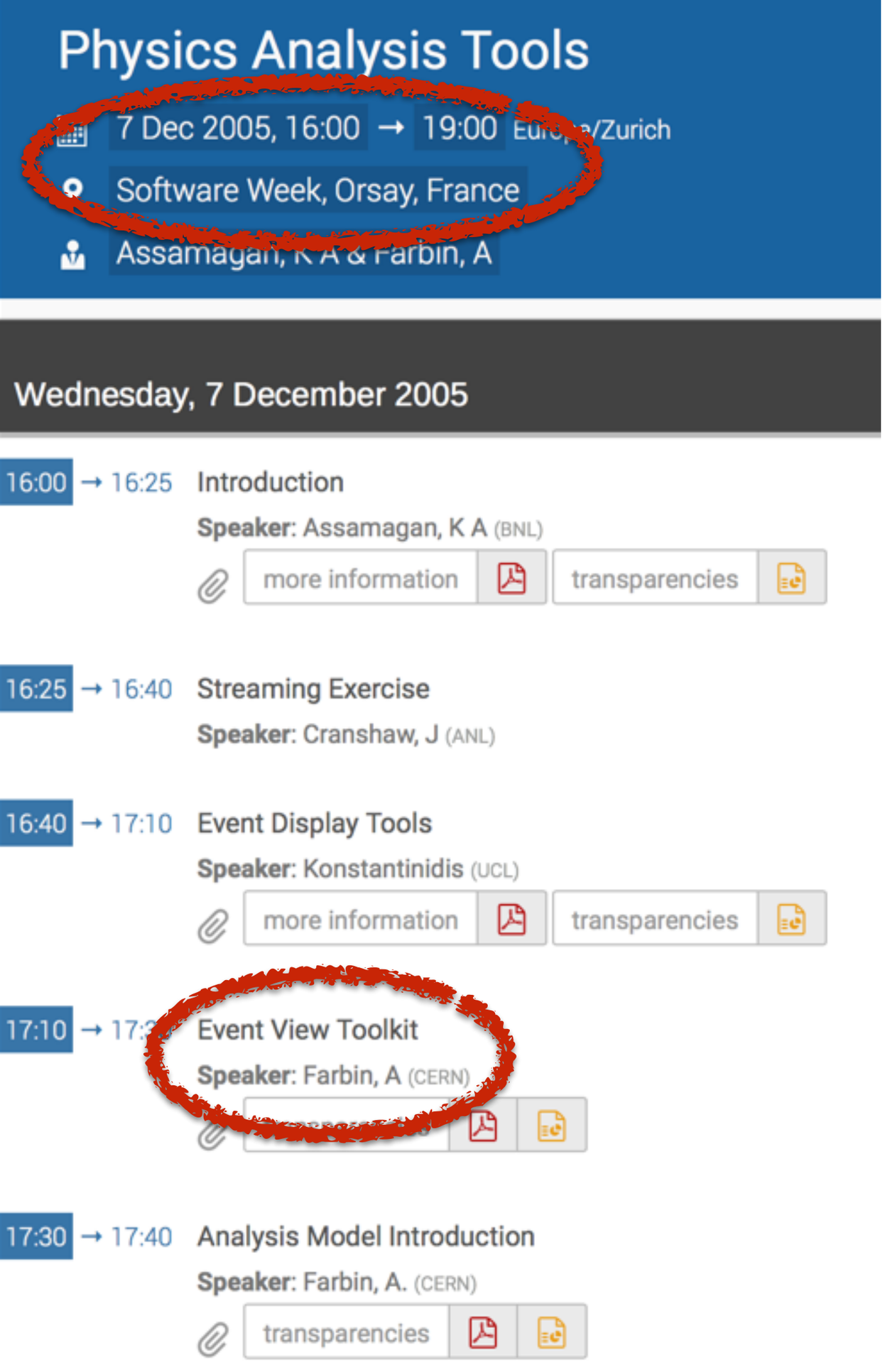
Deep Learning Implications for HEP Software & Computing

Amir Farbin

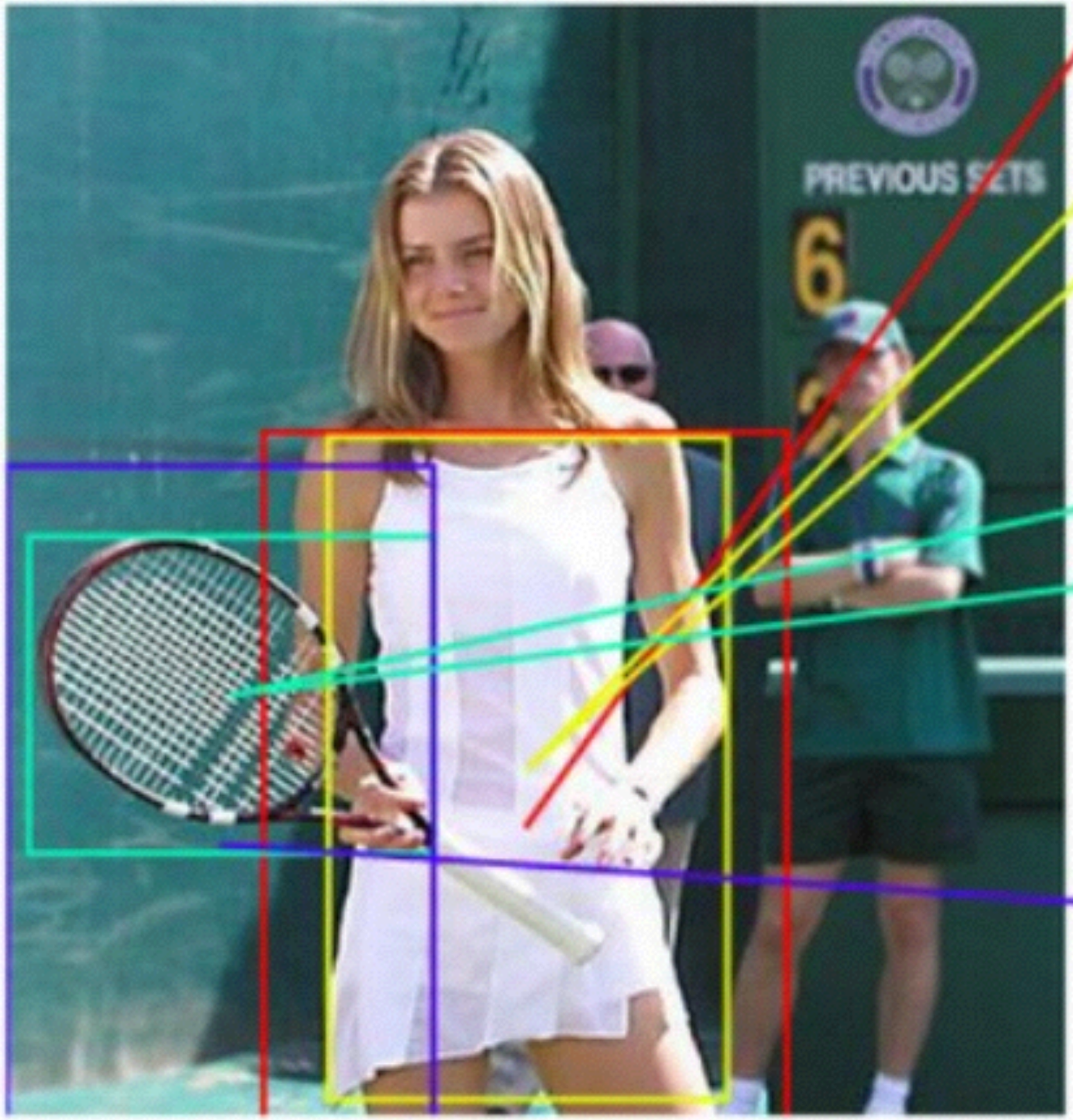
UNIVERSITY OF TEXAS ARLINGTON



- Last time I was at LAL Orsay was 10.5 years ago (ATLAS Software Workshop)
- I showed off something(s) that
  - *now* people say was “ahead of it’s time”,
  - but *then* solicited immediate negative reaction,
  - years of controversy and wasted time,
  - and eventual redemption?
- I’ll present a vision of DL in HEP that is also ahead of it’s time…
  - Please don’t take it too seriously…

Deep Learning

- *What is it?*
  - Many layer Neural Networks with large number of parameters.
- *Why now?*
  - Difficulty training such big networks in the past…
  - Convergence:
    - Solutions to *vanishing gradient problem*
    - *Big Data* provides the necessary large datasets for training
    - *GPUs* provide the computation power.
- *What's the fuss?*
  - Amazing feats like discovering cats in unlabeled raw images, playing Atari, and beating world's go champion.



1.12 woman

-0.28 in

1.23 white

1.45 dress

0.06 standing

-0.13 with

3.58 tennis

1.81 racket

0.06 two

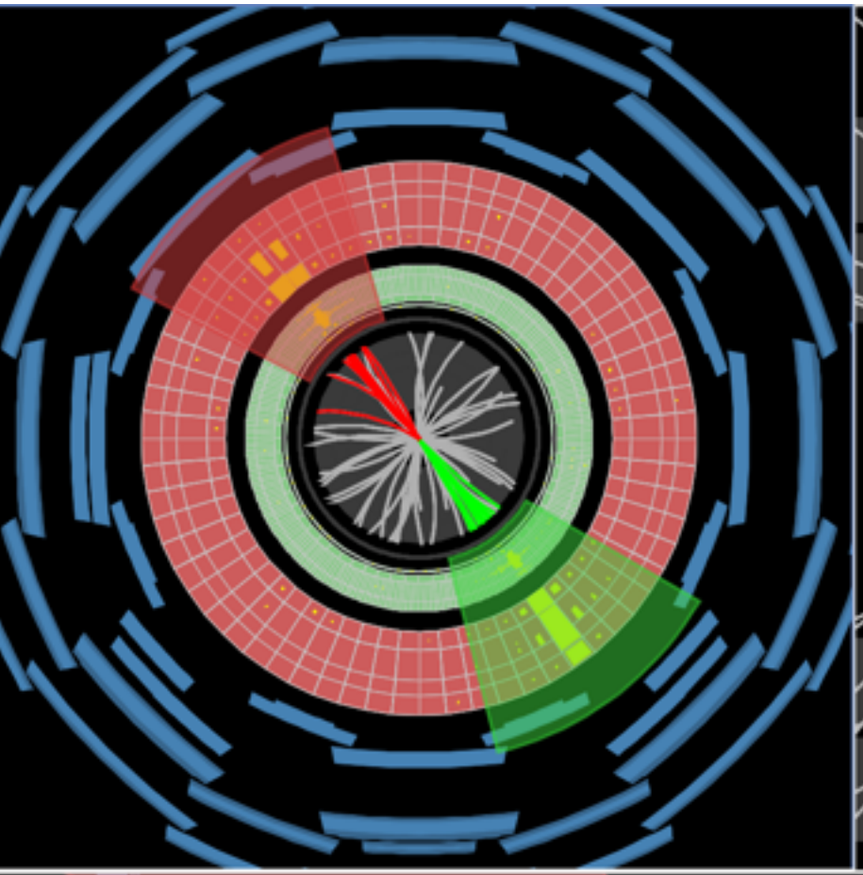
0.05 people

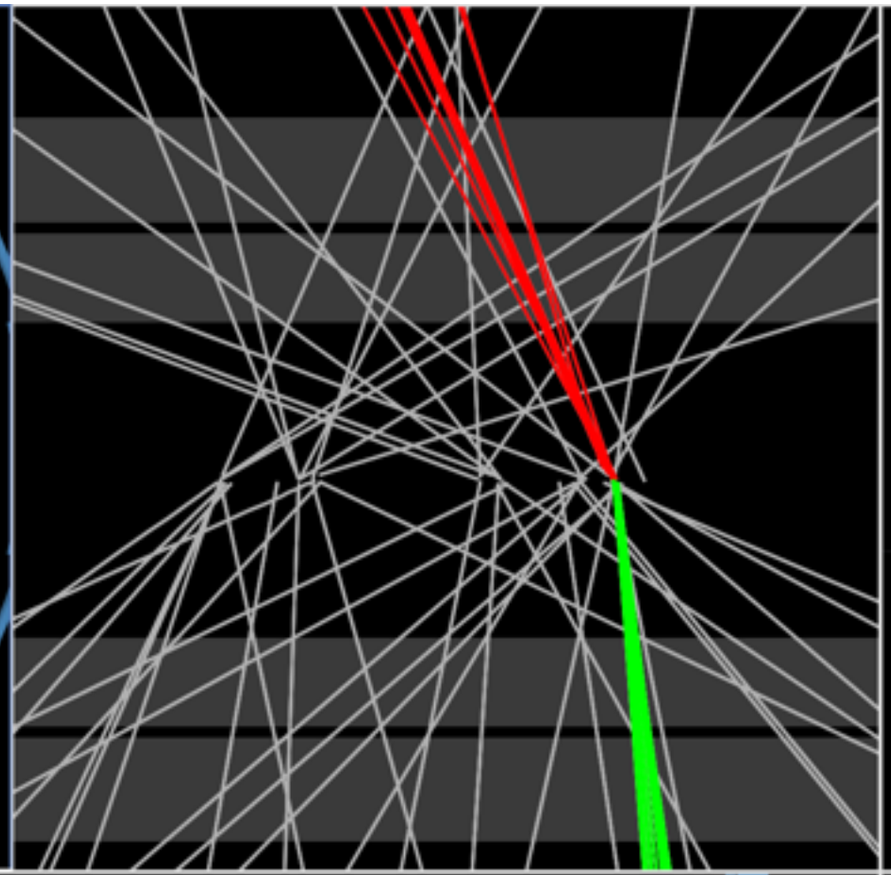
-0.14 in

0.30 green

-0.09 behind

-0.14 her

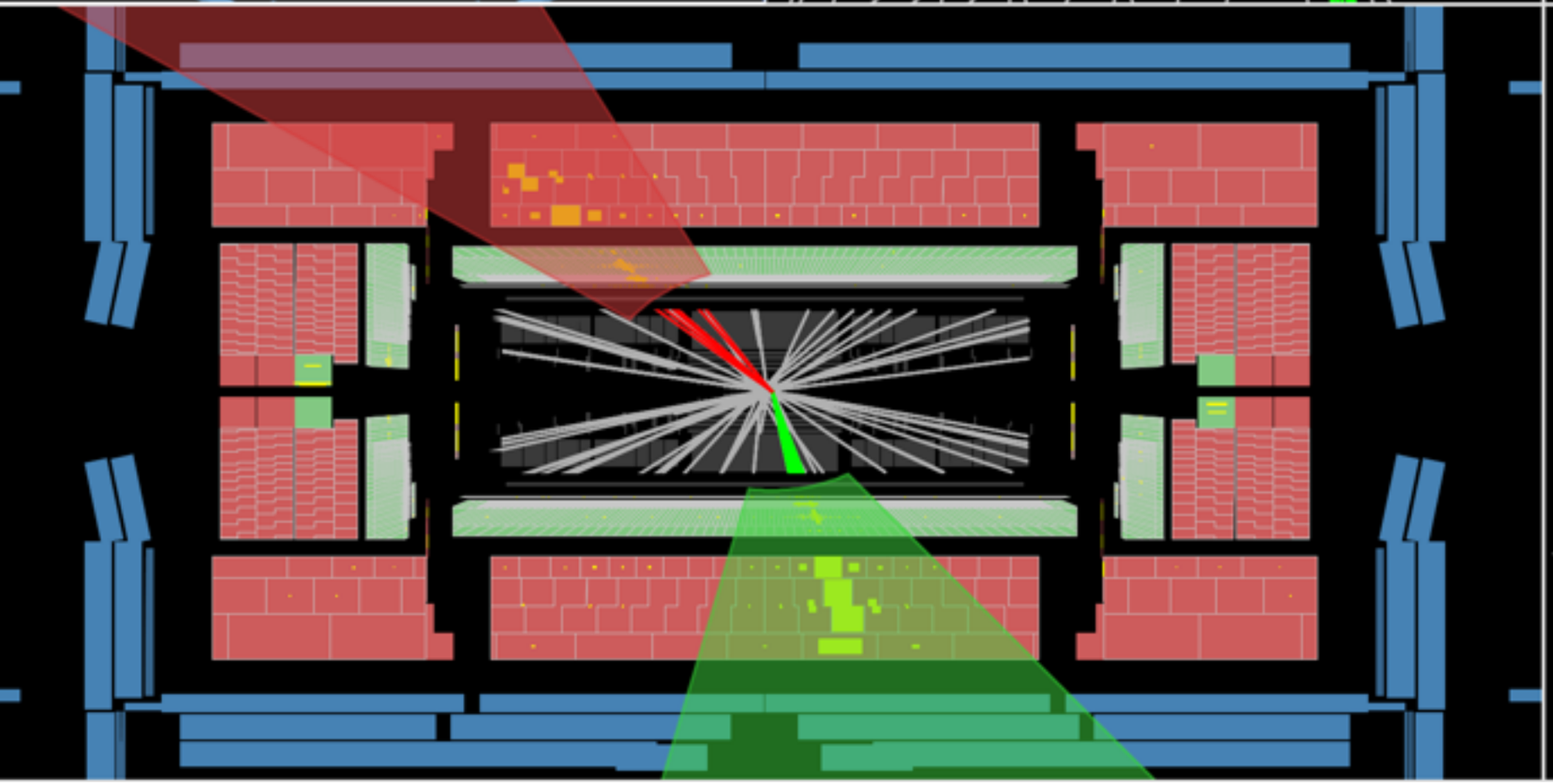


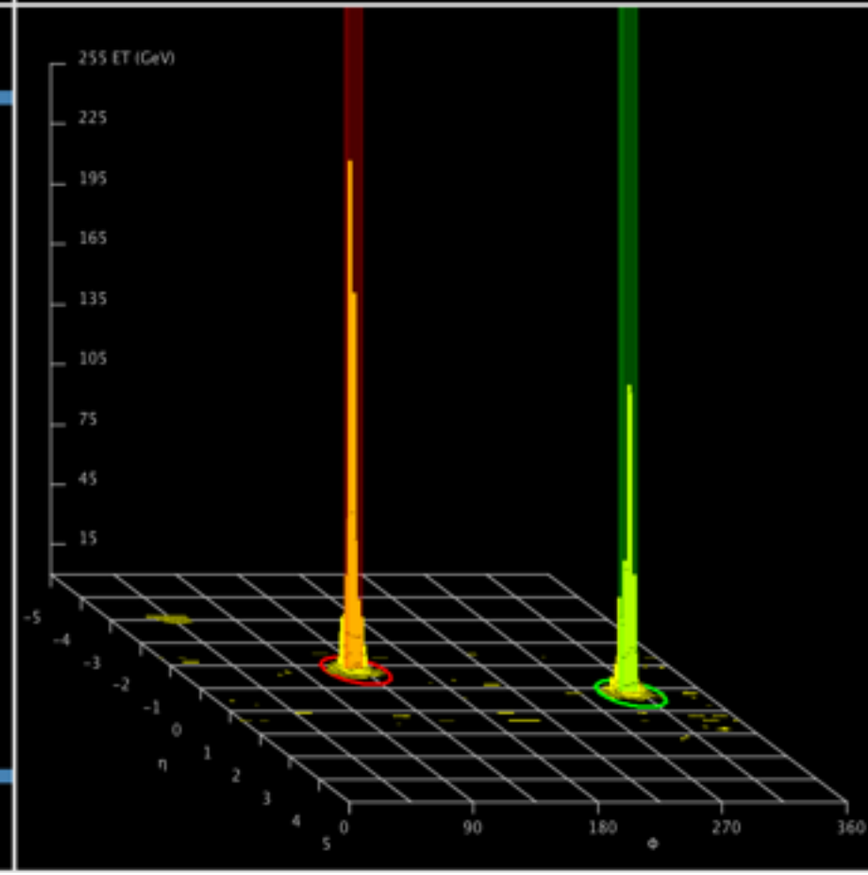


ATLAS
EXPERIMENT

Run Number: 271298, Event Number: 403602858

Date: 2015-07-11 02:09:14 CEST





Why go Deep?

- ***Better*** Algorithms
  - Hopefully DNN-based classification/regression *out performs* hand crafted algorithms.
    - For LArTPC, it may be able to do something we cannot do well algorithmically.
  - *Unsupervised learning*: DNNs classify without being told what are the classes.
    - The hope is that DNNs could make sense of complicated data that we don't understand or expect (e.g. anomaly detection).
- ***Faster*** Algorithms
  - After training, DNN inference is sometimes *faster* than algorithmic approach. e.g. Playing go.
  - Already *parallelized* and optimized for GPUs/HPCs. First broadly applicable and low threshold use of GPUs.
    - Industry building highly optimized software, chips, systems (HPCs), and cloud services.
  - DNN can *encapsulate expensive computations*, e.g. Matrix Element Method or simulation.
- ***Easier*** Algorithm Development: *Feature Learning* instead of *Feature Engineering*
  - Reduce time physicists spend writing developing algorithms that process raw data into the inputs features (e.g. Reconstruction) to traditional analysis or Machine Learning.
  - Save on development time and costs.

Why go Deep?

- ***Better*** Algorithms
  - Hopefully DNN-based classification/regression *out performs* hand crafted algorithms.
    - For LArTPC, it may be able to do something we cannot do well algorithmically.
  - *Unsupervised learning*: DNNs classify without being told what are the classes.
    - The hope is that DNNs could make sense of complicated data that we don't understand or expect (e.g. anomaly detection).
- ***Faster*** Algorithms
  - After training, DNN inference is sometimes *faster* than algorithmic approach. e.g. Playing go.
  - Already *parallelized* and optimized for GPUs/HPCs. First broadly applicable and low threshold use of GPUs.
    - Industry building highly optimized software, chips, systems (HPCs), and cloud services.
  - DNN can *encapsulate expensive computations*, e.g. Matrix Element Method or simulation.
- ***Easier*** Algorithm Development: *Feature Learning* instead of *Feature Engineering*
  - Reduce time physicists spend writing developing algorithms that process raw data into the inputs features (e.g. Reconstruction) to traditional analysis or Machine Learning.
  - Save on development time and costs.

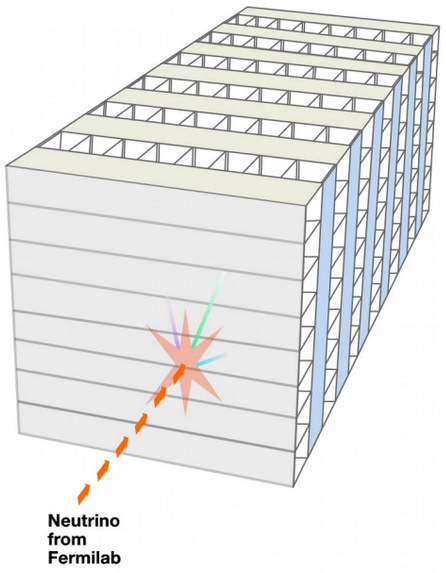
Moore's law?

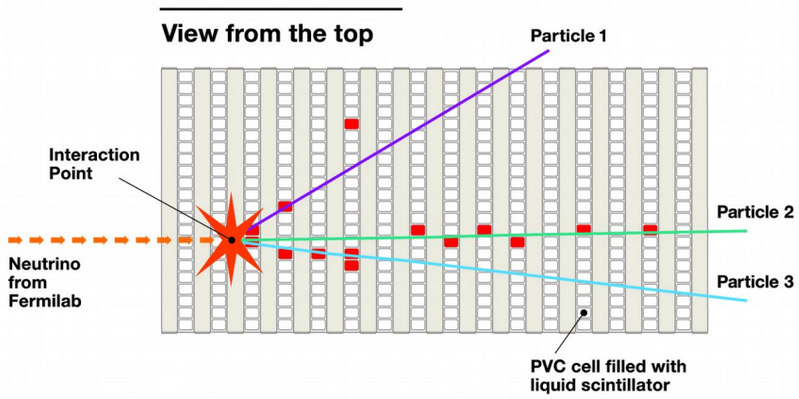
- For the first time, the cost of adding more transistors/silicon area has increased recently.
- HL-LHC computing requirements will outpace Moore's Law.
    - We cannot assume that we will easily get 10x the computing power for same price in 10 years.
- Solutions:
    - Quantum computers are no good for us…
    - Highly parallel processors (e.g. GPUs) are already > 10x CPUs for certain computations.
        - Unfortunately parallelization (i.e. Multi-core/GPU) has been difficult.
    - Trend is away from x86 towards custom hardware (e.g. GPUs, Mics, FPGAs, Custom DL Chips)
    - Deep Learning and Neuromorphic chips are a possible solution.
        - Think of the DL "seeing" tracks in silicon detectors like how DeepMinds's AI sees moves on the go board.
        - Neuromorphic chips are incredibly power efficient.

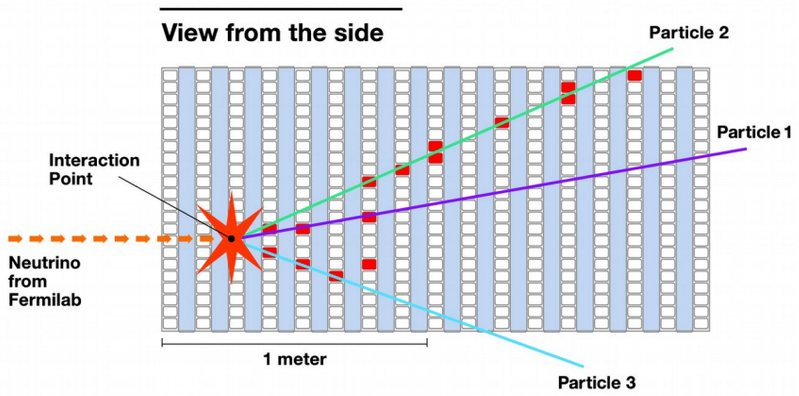
DL in HEP Today

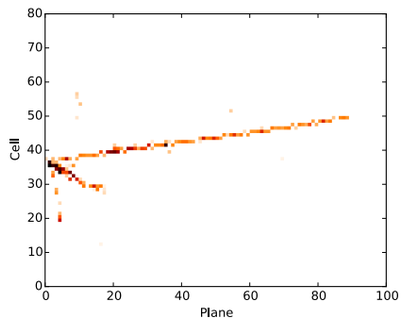
- *Classification*:
  - *Features*: Typically fully connected DNNs
    - signal/background, PID, b-tagging, boosted heavy particle tagging (W/Z/t/H)
  - "*Raw*": Typically Convolutional DNNs
    - Jet tagging (e.g. quark vs gluon), PID in TPCs, Calo, and other "imaging" detectors.
    - Tracking?
- *Regression*:
  - Energy fits/calibration in TPCs, Calorimeters, …
  - Matrix Element Method- Efficiently store the MEM computation (e.g. on a HPC) in a fast DNN.
- *Anomaly Detection/Unsupervised Learning*: Monitoring (Detector, GRID, Analytics), General Searches, …
- *Reinforcement Learning*: Learn through trial/error, likely coupled with above.
  - Hand scans, Anomaly Detection, …
- I'll concentrate on Triggering, Reconstruction, and Simulation… biggest impact on computing.

Nova

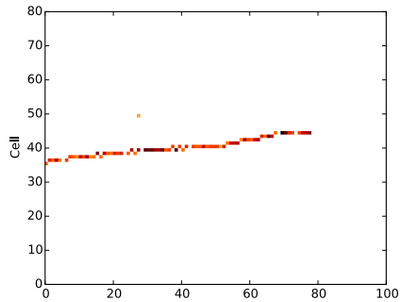




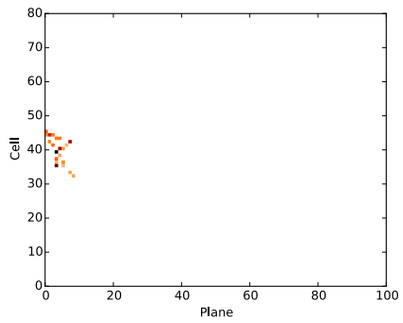




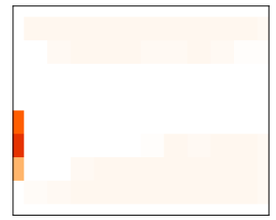
Muon Neutrino
DIS CC



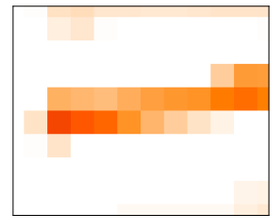
Muon Neutrino
QE CC



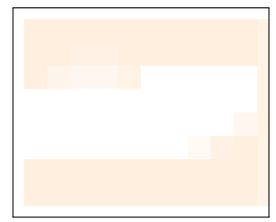
Muon Neutrino
NC

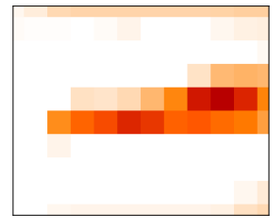


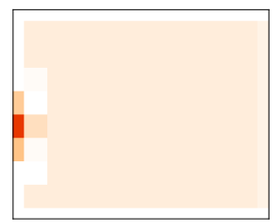
Hadronic
Feature
Map

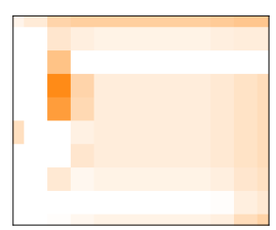


Muon
Feature
Map









	CVN Selection Value	ν_e sig	Tot bkg	NC	ν_μ CC	Beam ν_e	Signal Efficiency	Purity
Contained Events	–	88.4	509.0	344.8	132.1	32.1	–	14.8%
$s/\sqrt{b}$ opt	0.94	43.4	6.7	2.1	0.4	4.3	49.1%	86.6%
$s/\sqrt{s+b}$ opt	0.72	58.8	18.6	10.3	2.1	6.1	66.4%	76.0%

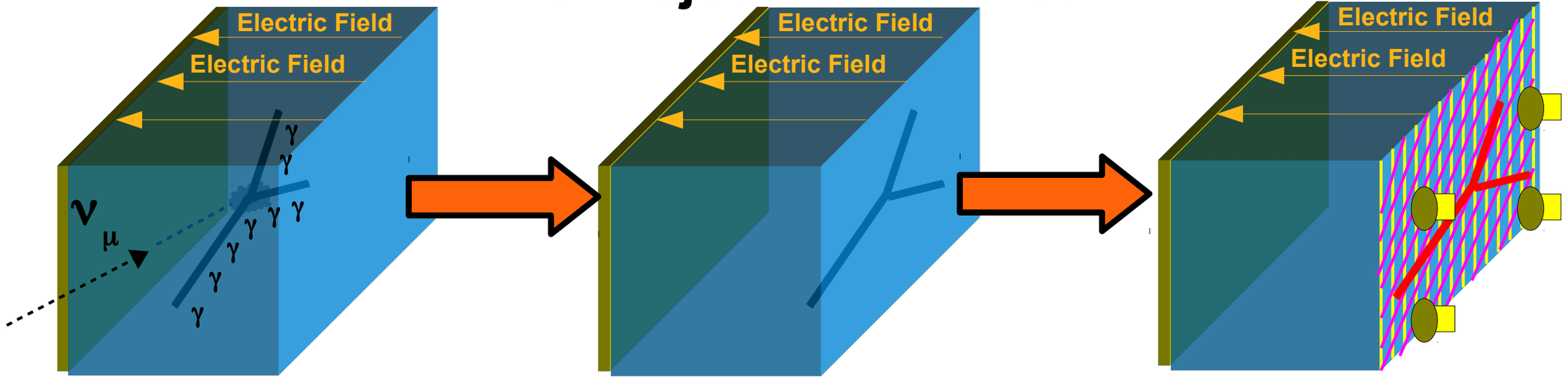
	CVN Selection Value	ν_μ sig	Tot bkg	NC	Appeared ν_e	Beam ν_e	Signal Efficiency	Purity
Contained Events	–	355.5	1269.8	1099.7	135.7	34.4	–	21.9%
$s/\sqrt{b}$ opt	0.99	61.8	0.1	0.1	0.0	0.0	17.4%	99.9%
$s/\sqrt{s+b}$ opt	0.45	206.8	7.6	6.8	0.7	0.1	58.2%	96.4%

- 40% Better Electron Efficiency for same background.

http://arxiv.org/pdf/1604.01444.pdf

LArTPC

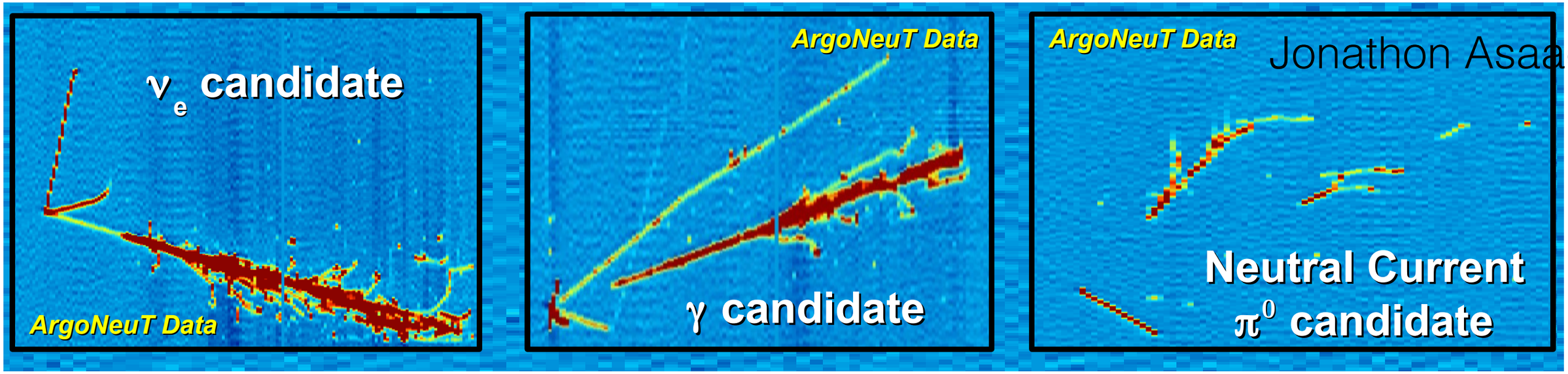
Time Projection Chamber



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 PMT's



Jonathon Asaadi

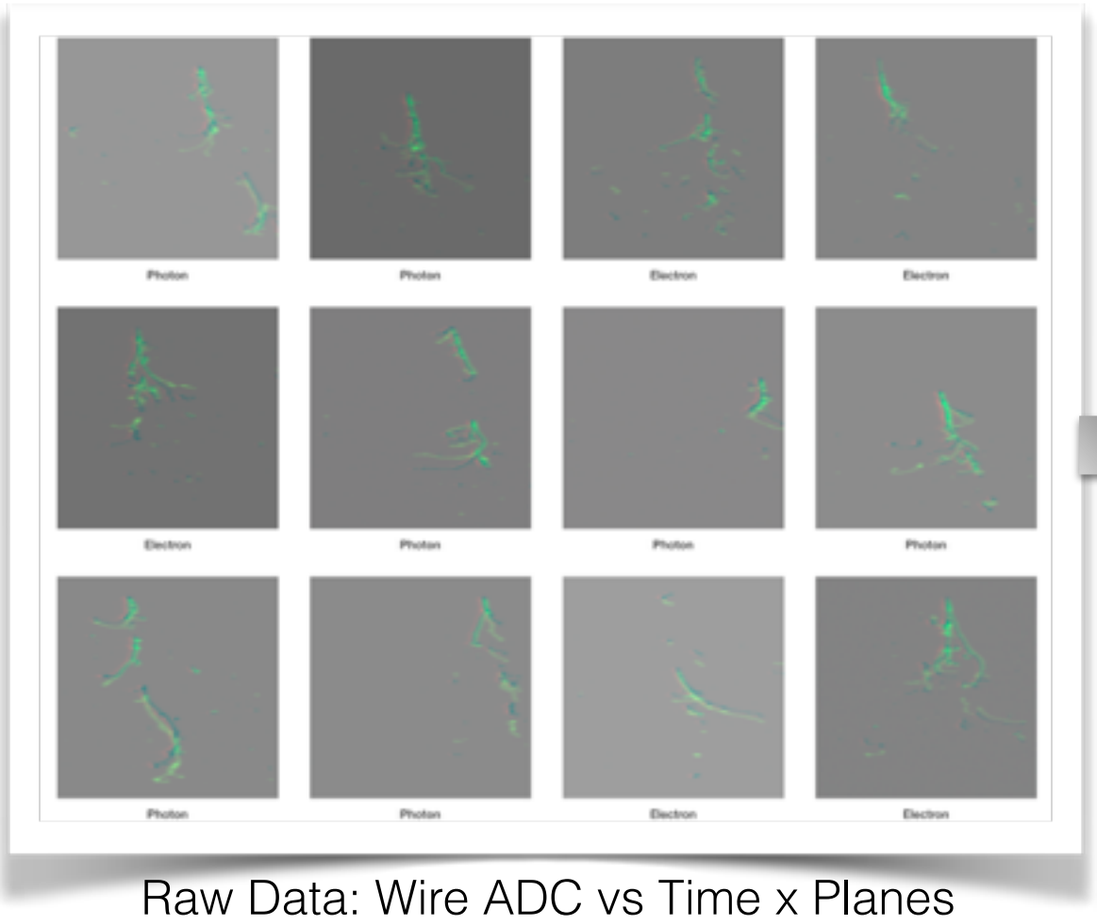
Tracking, Calorimetry, and Particle ID in same detector.

Goal ~80% Neutrino Efficiency.

All you need for Physics is neutrino flavor and energy.

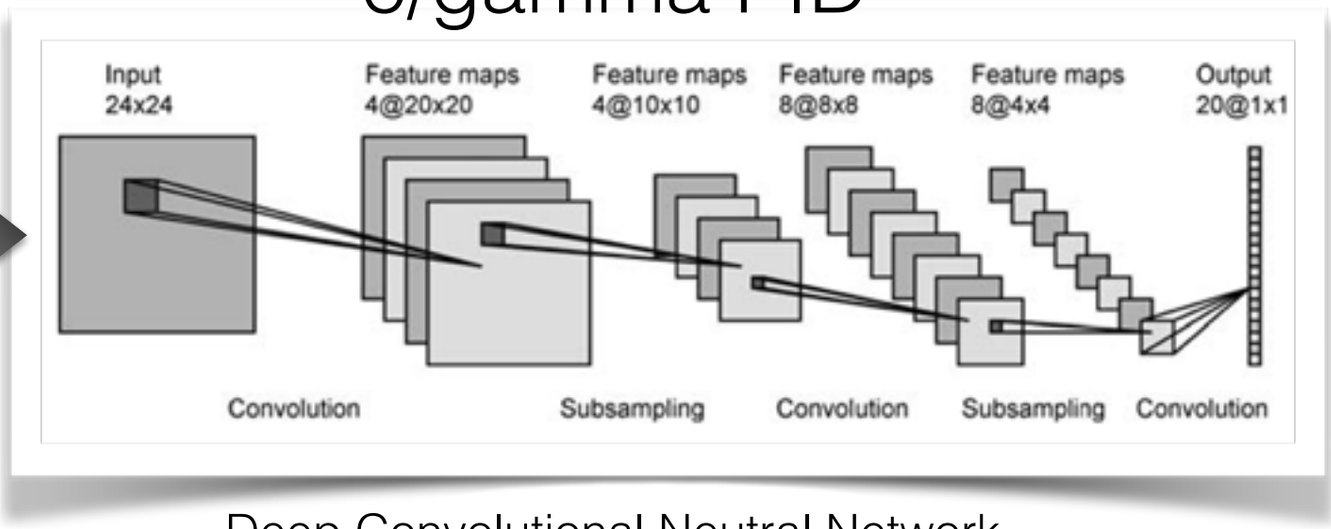
LArIAT

e/gamma PID



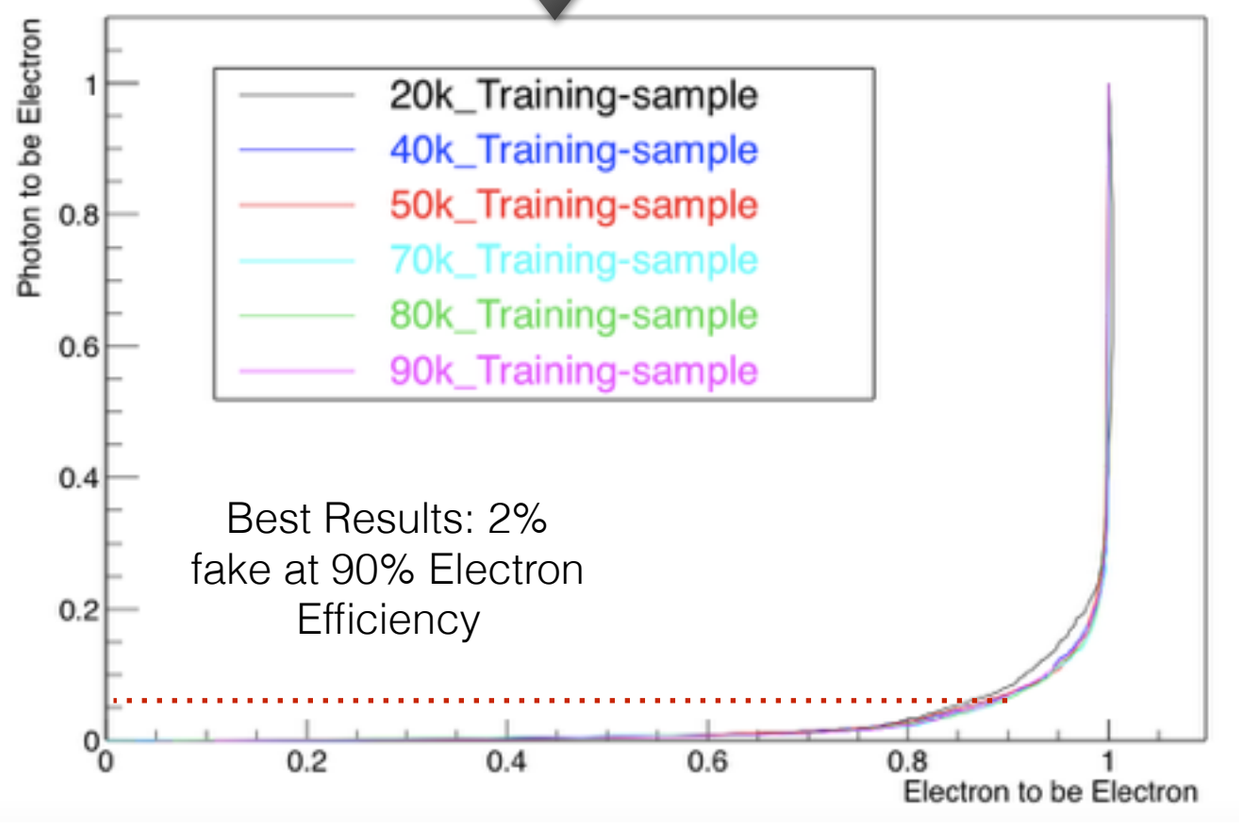
Raw Data: Wire ADC vs Time x Planes
(LArIAT Simulation)

Train



Deep Convolutional Neutral Network
(GoogLeNet)

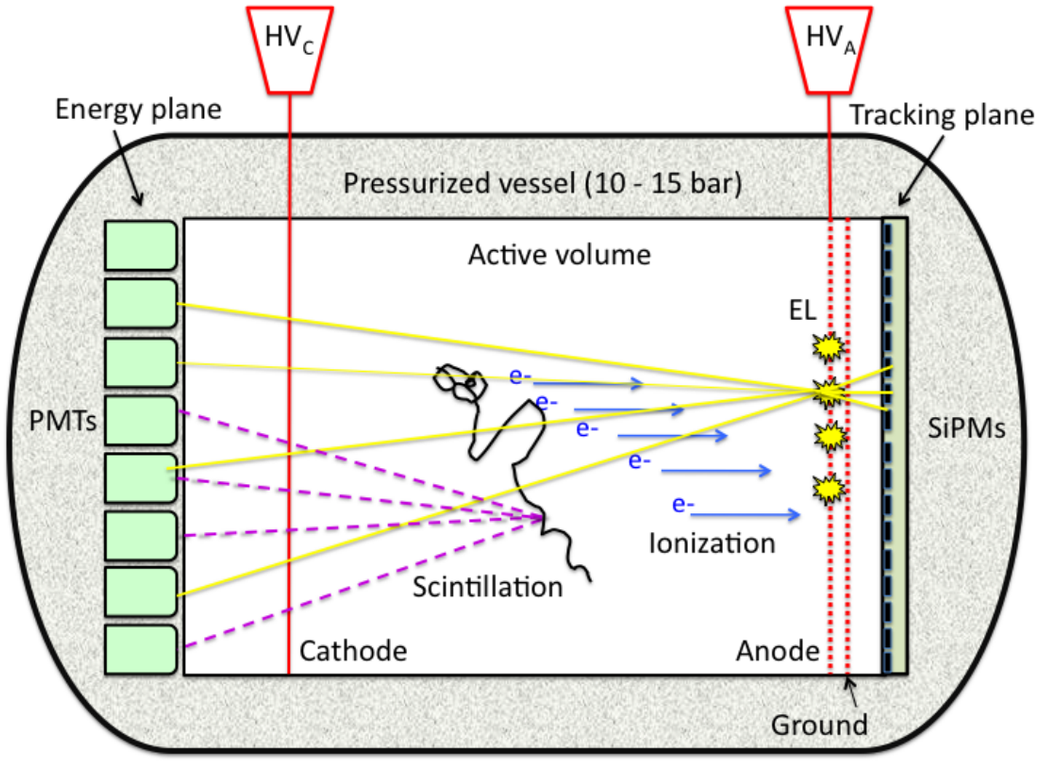
Out of the box *Feasibility Study* with No attempt at optimization.

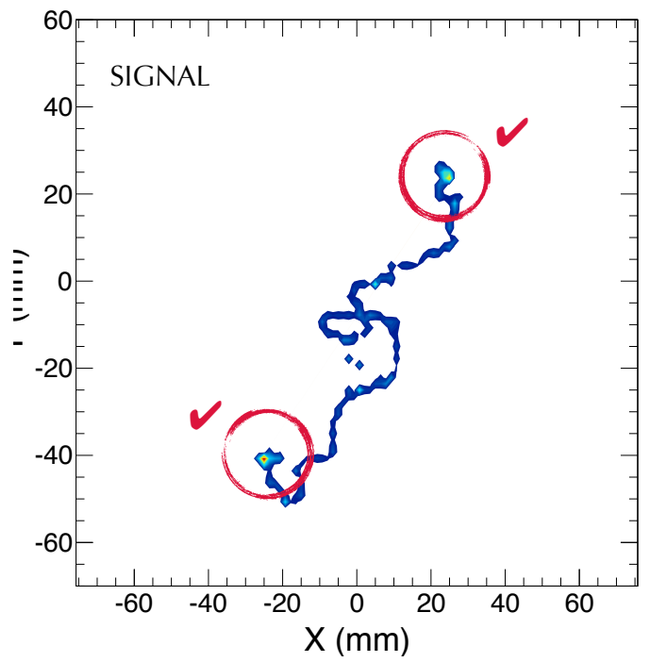


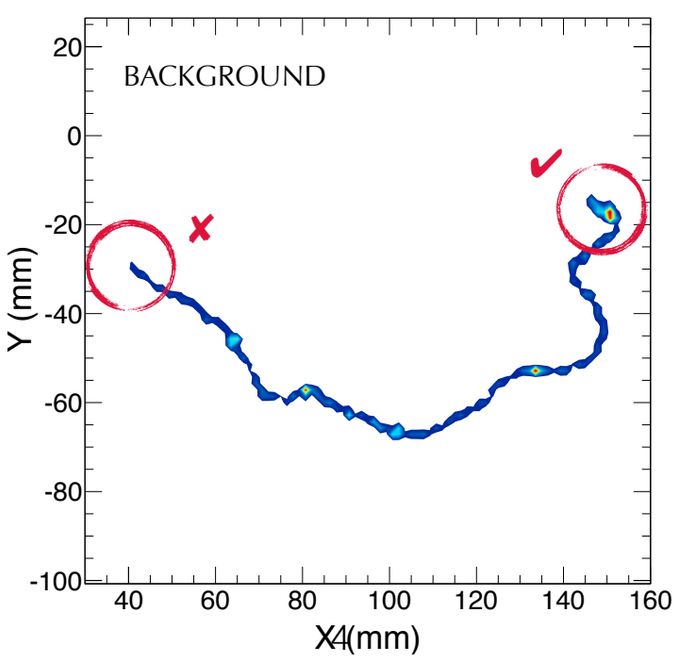
- First results with neutrinos:
  - 5% NC at 80% CC
  - 15% Muon CC at 80% Electron CC
- Regression working on Neutrino Energy
- DL efforts present also in other LArTPC experiments (not yet public).
- May be easy and ideal tool for Detector Optimization.

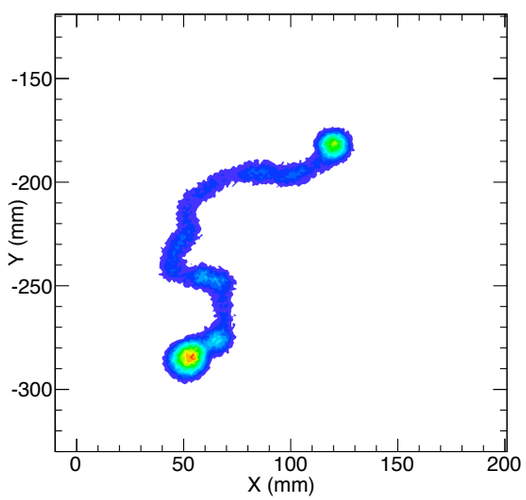
NEXT Experiment

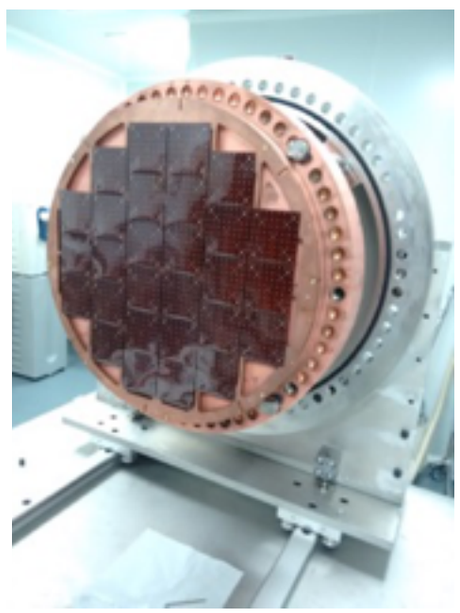
- Neutrinoless Double Beta Decay using Gas TPC/SiPMs
- Signal: 2 Electrons. Bkg: 1 Electron.
- First DL Study: what is the ultimate performance? Are we limited by the physics? (1 mm resolution… design 1 cm)
  - 99.96% Signal Efficiency for 0.2% Background.
- Serious effort in experiment to use DL for at least detector optimization.
- Would you believe a discovery of such a rare process purely based on feeding raw data to DNN?





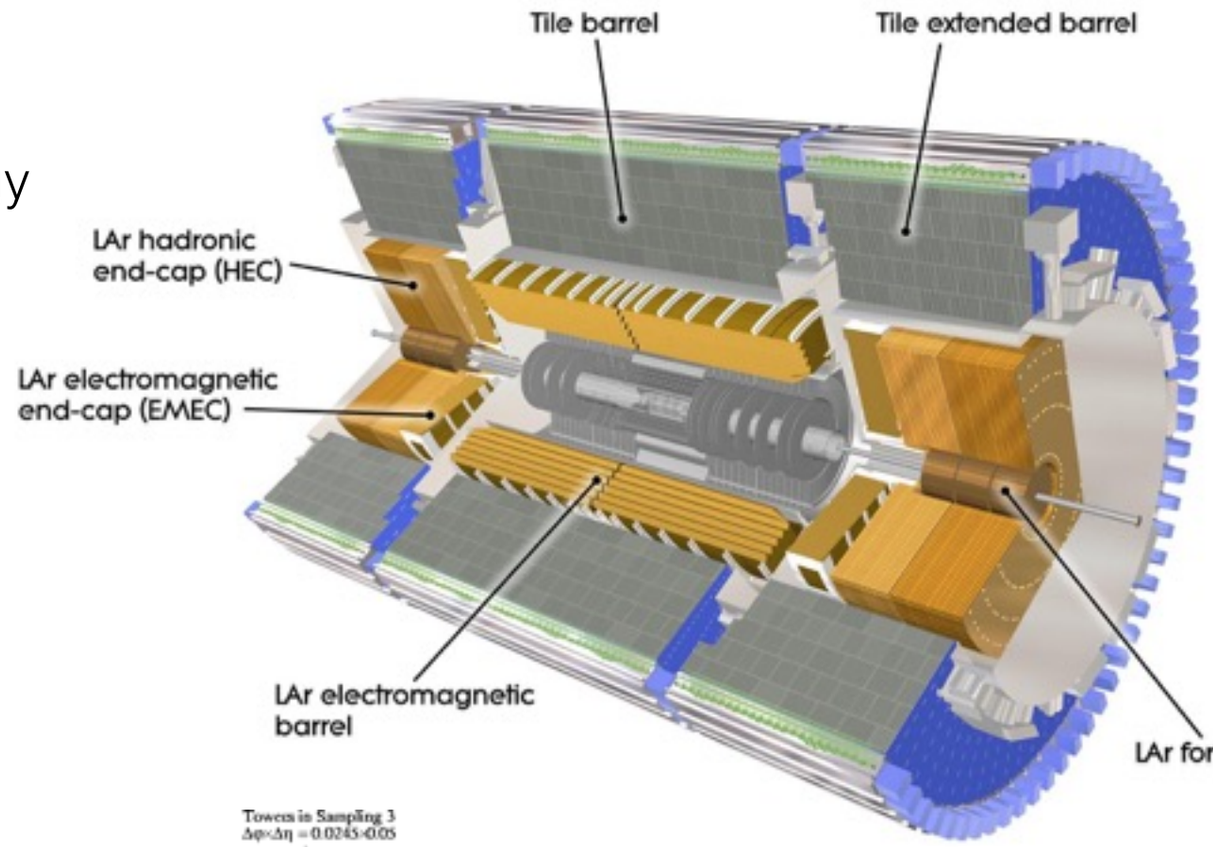


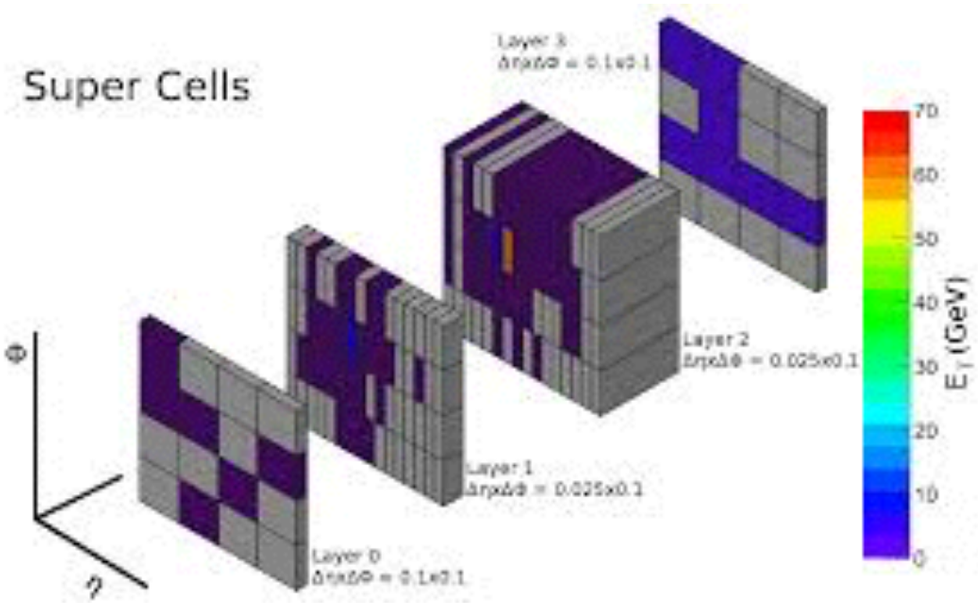


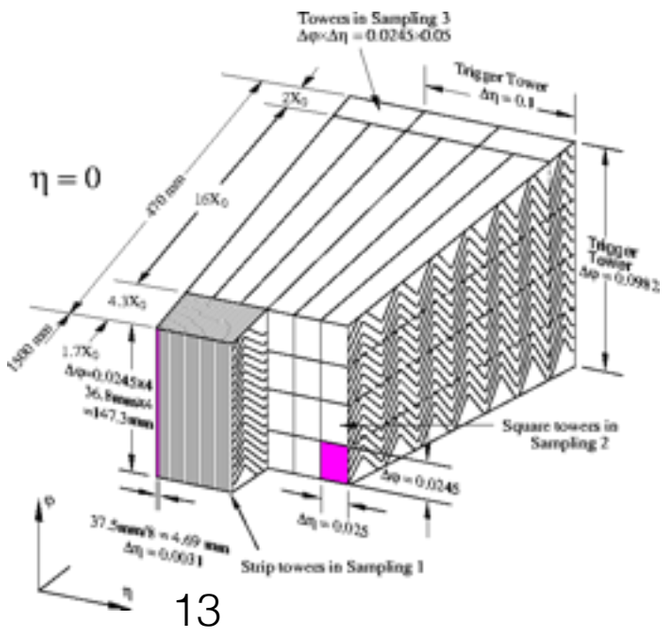


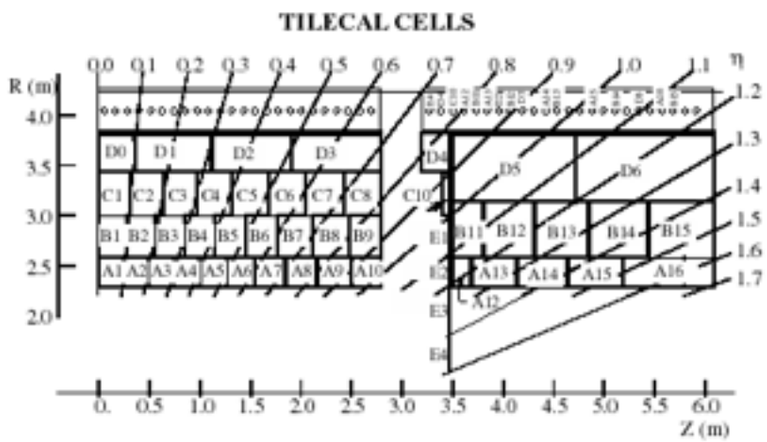
ATLAS Calorimeter

- Ideally suited for “imaging”
  - Electromagnetic- Highly transverse and longitudinal segmented.
  - Hadronic- Longitudinal sampling
  - 200K Calorimeter cells measure energy deposits.
  - ~ 64 x 36 x 7 3D Image
  - Interesting Challenges: non-uniform granularity, cylindrical geometry.









DL Based Reco

- Immediate uses:
    - “Imaging” detectors likely path:
        1. Improved classification/regression with Convolutional NNs.
        2. Fast Showers with Generative models.
        3. Feature (particle) extraction with Regional NN and semantic segmentation.
        4. Full event classification
            - Recurrent networks
            - Reinforcement training… turn reconstruction into a board game.
- Help with detector optimization:
    - DL provides easily obtainable, consistent, and probably optimal metrics.
    - Just simulate… no need to build reco tuned to every possibility.
    - Understand the fundamental limits but turning on physics / detector effects one by one in simulation.

Towards LHC DNNs Reco

- *Calorimetry* most natural starting point.
  - *High likelihood of success:* 3D Image (~ 64 x 36 x 7) classification + Energy Regression, similar to LArTPC.
  - *High impact:*
    - Improve ID *and* energy resolution make the peaks stand out.
    - Turn DNN into generative model for fast shower simulation.
  - *High potential:* we don't use all information so room for improvement
    - *e/gamma:* take full advantage of the high granularity and accordion structure
    - *hadronic calibration:* take full advantage of longitudinal sampling and other handles
    - *particle flow:* correlate with tracks (and vertex) for hadronic calibration, taus, jet-tagging, boosted objects…
      - Imagine Inner Detector and Muon Spectrometer as additional high resolution longitudinal layers.
- *Tracking*
  - Primarily addresses a computing problem: Tracking in trigger for HL-LHC
  - Much more difficult.
  - You'll hear more about tracking later this session…

DNN+HEP Software Needs (1/4)

1. *Inference in HEP Frameworks:*

    - Need optimized and validated inference implementation.

        - Nova uses Caffe in art.

        - ATLAS moving from handwritten C++ to TensorFlow.

    - DNN weights can be Gigabytes, likely need

        - Condition DB-like systems storage.

        - Memory sharing between processes/threads.

    - I can imagine a DL service similar to ATLAS APE GPU service:

        - Processes are client of server(s) that talk to backends/accelerators.

        - No reason for every experiment to reinvent the wheel here…

DNN+HEP Software Needs (2/4)

2. *Training systems*:

- Training DNNs efficiently generally requires GPUs (or other future accelerators).
- Hyper-parameter scans critical part of DNN development workflow.
  - Great use of GPUs on HPCs.
  - Google and other clouds specifically target DL.
- Today's training samples can already be 10s of Terabytes, requiring massive parallelism.
  - *Data Parallelism*: Bottlenecked by gradient syncing between GPUs or systems. Lots of Engineering in Industry already.
  - *Model Parallelism*: Less sync'ing but only makes sense for large enough model.
  - No more embarrassingly parallel. Must provision large number of machines.
- As DNNs become essential, training them becomes part of software releases, simulation, reco,… cycle.
  - New simulation/reco can require regenerating large training sets (various conditions) and running long training before using reco.
  - Somewhat analogous to calibration on express streams.
- I can imagine Workflow and Data Management systems designed for DL training workflows on any available resource.

DNN+HEP Software Needs (3/4)

3. Opportunistic Data Generation/Processing:

- DL generally requires huge independent training samples.
  - Probably need to resort to Data Augmentation, Fast MC, etc… when possible.
  - But the data is private, making collaboration and rapid publication difficult.
- Collaboration with Machine Learning experts and among experiments require public data sets.
  - Publicly available simulation and reconstruction (for base-line). (see: Journal of Brief Ideas.)
  - Reconstruction DNNs will likely require Geant4. (i.e. CPU intensive)
  - No dedicated resources, so rely on opportunistic CPU.
  - Need to store and distribute large data-sets.
- I can image WMS/DDM systems allowing users to opportunistically run docker containers on any system, and centrally collecting samples for everyone.

DNN+HEP Software Needs (4/4)

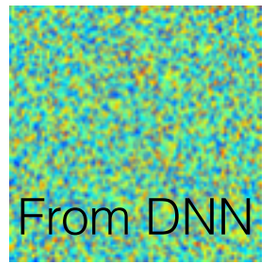
4. *Event Processing within Deep Learning Frameworks*

- DL will potentially become integral to our software and trigger
  - We may replace code with weights.
  - DL integrated into HEP frameworks. Not just an external. (example next slide)
- Many-core/FPGA/neuro-morphic accelerators may prolong Moore's law
  - Experiments like DUNE will run for 30 years and must keep up with emerging tech.
  - Frameworks must [automatically] optimize and place computations on a variety of hardware.
  - May need to distribute processing of individual events across cluster (like HEP trigger)
    - Use network hardware for primitive operations during transfers.
    - Partially process on specialized machines (specific accelerators, HPC, massive memory, …)
- Threading in GaudiHive, CMS FW, art, … use data flow programming model (graphs), like many DL systems.
- Industry will highly optimize DL systems and provide services around them.

Weaving-in DNN Reco

Feature List = {Hit_1, Hit_2, …}

Feature Map =



Raw Data

Sub-detector 1 Feature Extraction DNN

Sub-detector 2 Feature Extraction Alg

Feature List

Feature Map

Feature List

Feature Map

Pattern Recognition Alg 1

DNN Pattern Recognition 1

Pattern Recognition Alg 2

DNN Pattern Recognition 2

Fitting Alg 1

Fitting Alg 2

Simultaneously Train All DNNs

Combined Reconstruction Alg

DNN Combined Reconstruction

R&D Proposal

- *Premise*: We need new frameworks to take advantage of DL and emerging architectures.

    ➡ Build HEP Framework on top of a DL Framework.

- If we envision new frameworks need to do R&D now, ver 1.0 by 2020, deployed by 2025.
- R&D Proposal (can we do traditional HEP Reco in DL Framework?):
    - Build HEP Reco on top of Google's OpenSource TensorFlow
        - General computation system, based on Directed Acyclic Graphs.
            - Framework for Automatic optimizations (like Theano), though currently primitive.
        - Supports all architectures and distributes computation across GPUs and clusters.
        - Build a HEP Framework in python (like Keras) with C++ wrapped in TF ops.
- 3 project ideas:
    - First steps of LArTPC reco: deconvolution, hit finding, …
    - Online Sparsification and compression of LArTPC data for protoDUNEs.
    - ATLAS GPU Trigger Demonstrator: Wrap the existing GPU/CPU kernels in TensorFlow Ops

Science Fiction?

- Imagine in next 10 years DNN lives up to the hype…
  - We've proven DNNs gets us better, faster, easier software… and hardware.
  - Industry investment in DNNs has yielded significant gain over Moore's Law
    - Custom DL/neuromorphic chips and HPCs
    - Software Frameworks
    - Cloud Services
    - Consultants:
      - Data Scientists: DL reduces need for domain-specific expertise (e.g. in biology now).
      - Data Engineers: low level optimization, deployment, operation…
- Actually, all of these already exist!
- Large portions of HEP code replaced by deep neutral network architecture and weights.
  - HEP Software Frameworks built on top of DL Frameworks.
  - To DL systems, our computing looks like everyone else's… e.g. other sciences.
- Optimization, deployment, operations handled by professional Data Engineers.
- Trigger implemented in custom inference systems built from commodity hardware.
- Computation performed on DL Clouds and scientific HPCs.
- DNNs designed and trained in collaboration with professional Data Scientists.
- HEP PhDs trained/funded by industry to apply DL to HEP and then transition to industry.

Final Thoughts

- Deep Learning can change how science is done.
  - Improve performance. Save time and money.
  - Mitigate stalling of Moore's law.
  - Use most recent hardware.
  - Allow scientists to focus on concepts rather than implementation.
- Hopefully by next year, we have proven that DL Reco is a good idea…
- If we want to be ready for the DL revolution in 10 years, we need to do R&D now.
- We can't forget that DL can complicated things:
  - Systematics. Data/MC agreement.
  - Generate large independent training and calibration samples.
  - New complicated "release", production, and analysis cycles/work-flows.

Random Related Slides

HEP Software Landscape

- Shift from Fortan to C++ in late 1990s.
- *The* biggest datasets until mid 2000s… industry now leads.
- Problems:
  - ***Complexity***: C++ frameworks, data structures, …
  - ***Future Architectures***: Difficulty utilizing Multi-core CPUs and *new/future [co-]processors* (e.g. GPU/MiC).
  - ***Computation***: HEP computing requirements outpacing Moore's Law. Trend is away from x86 towards custom hardware (e.g. FPGAs).
    - HL-LHC will present higher rates/pileup (combinatorics in pattern recognition tracking/vertexing in trigger) and higher granularity (data size).
- ***Expensive*: ATLAS software cost ~O(250 Million) CHF to build over 15 years…**
- ***Development and maintenance***:
  - 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.

HEP SW 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 and one event in parallel threads.
- 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.

Looking Ahead…

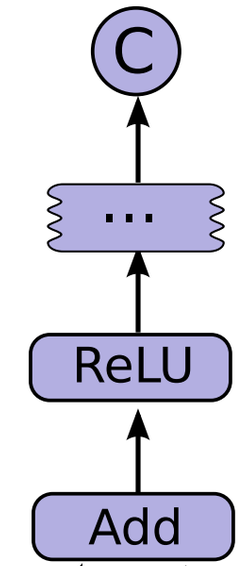
- Concurrency (simultaneously processing many events) is a hot topic. 2 types
  - *Task Parallel*: Many threads, each processing one event.
  - *Data Parallel*: Algorithms processing many events at once.
- The LHC experiments are confronting this issue. Current focus on Task Parallelism:
    - CMS already 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.
    - R&D Now. Framework in early 2020s. Reimplement software chain by 2025.

Obstacles to DNNs

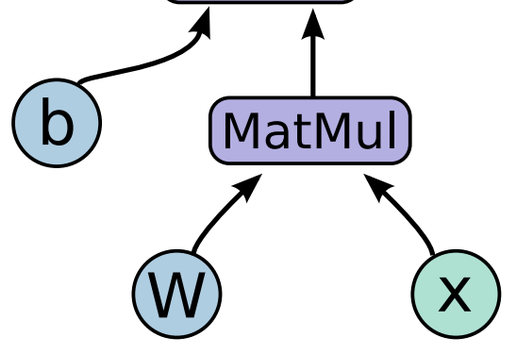
- Generally speaking, DNN software is not really a problem.
    - python tools for conversion between ROOT and numpy/hdf5
    - Fundamentally there are two approaches (DAG vs C++/steering), but at high level, very HEP framework like.
- GPUs are a must (at least for CNNs)
    - Training takes time. Difference between waiting a day vs a month.
    - Hyper-parameter scanning requires lots of experiments.
    - Large models or large data sets (100s TBs) require HPCs and finally tuned software (as in companies).
        - Generally speaking, for complicated problems, deeper networks have been shown to do better.
- Training Data
    - Requires large training sets of nearly raw data, potentially independent of existing MCs.
        - Maybe we don’t solve our computation problem!
    - Experiment data access rules make it difficult to collaborate or publish incremental steps.
    - Private generation requires large computing resources.

DNN Software

- Basic steps
  - Prepare data
  - Build Model
  - Define Cost/Loss Function
  - Run training (most commonly Gradient Decent)
  - Assess performance.
  - Run lots of experiments…



- 2 Classes of DNN Software: (Both build everything at runtime)
  - Hep-Framework-Like: e.g. Torch, Caffe, …
    - C++ Layers (i.e. Algorithms) steered/configured via interpreted script:
  - General Computation Frameworks: Theano and TensorFlow
    - Everything build by building mathematical expression for Model, Loss, Training from primitive ops on Tensors
    - Symbolic derivatives for the Gradient Decent
    - Builds Directed Acyclic Graph of the computation, performs optimizations
    - Theano-based High-level tools make this look like HEP Frameworks (e.g. pylearn2, Lasagna, Keras, …)



DNN Software

- Data is typically represented as numpy-like Tensors (N dim arrays).
    - Easily change between CPU and GPU implementations of tensor.
    - Usually automatically transferred between CPU/GPU memory.
    - Usually persistified in hdf5… sometimes stored in databases.
        - Supposedly very optimized.
- GPUs?
    - Most eventually call the same libraries (e.g. cuCNN) for optimized implementation.
    - Most have some library of algorithms with both CPU and GPU implementations.
    - Theano generates code for kernels and uses kernel libraries when appropriate.
    - Almost all on NVidia/CUDA… seeing OpenCL now.
        - I’ve seen significantly better performance from AMD over NVidia for some kernels.

DAG 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.
  - Automatic analysis of the computation graph and targeted code generation, developed/optimized by experts

TensorFlow

- Google Deep Learning tool, many similarities to Theano, recently open sourced. C++ and python API.
    - Computation is done by building a DAG in a *Session*. Performs DAG optimizations.
    - Library of *operations* which wrap CPU and GPU *kernels*
        - Extendable.
        - Ops can provide gradient implementation.
        - Doesn’t generate code like Theano.
        - Provides control flow operations which allows implementing loops.
    - *Variables* provide mutable data.
    - *Containers* allow sharing mutable data between disjoint computation graphs.
    - *Queues* automatically provide asynchronous computation when possible.
    - Designed for heterogeneous and distributed computing.
        - Point is that the same code generate same computation on any system.
        - From mobile phones, to multi-gpu systems, to heterogeneous distributed clusters.
        - Optimizes kernel placement using cost models, simulations, measurements of performance of kernels on devices and data transfer times.
        - Has Fault Tolerance.
    - *TensorBoard* provides graph and data visualization.

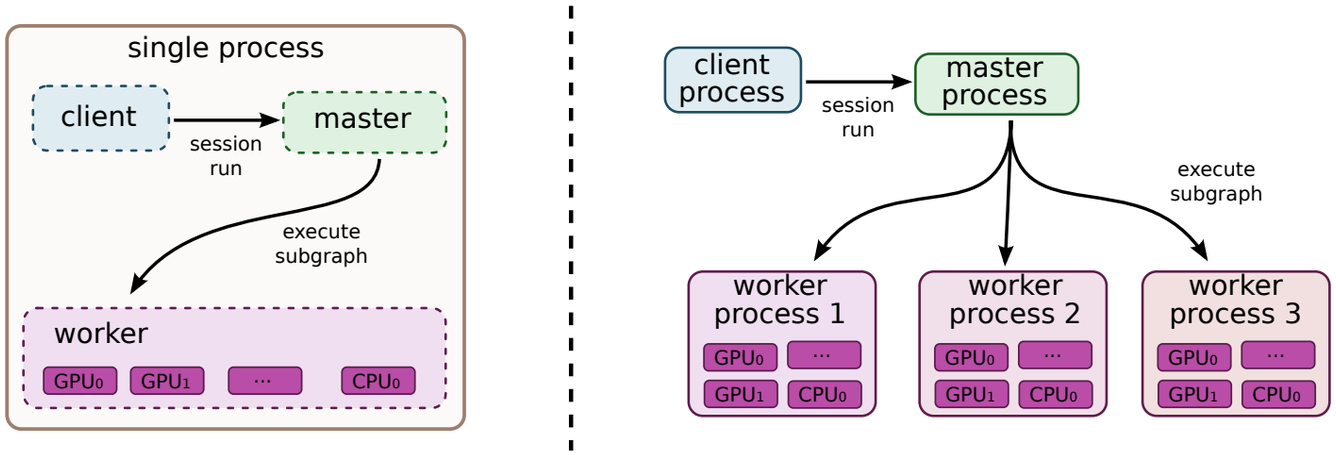


Figure 3: Single machine and distributed system structure

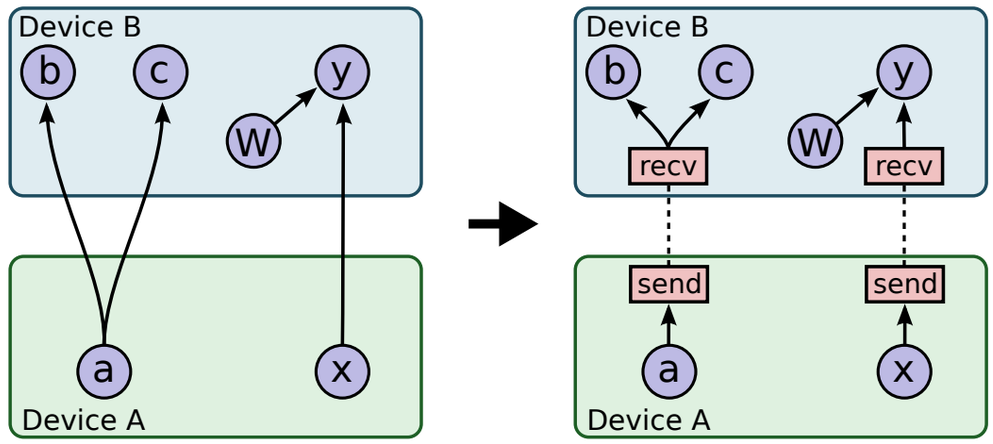


Figure 4: Before & after insertion of Send/Receive nodes

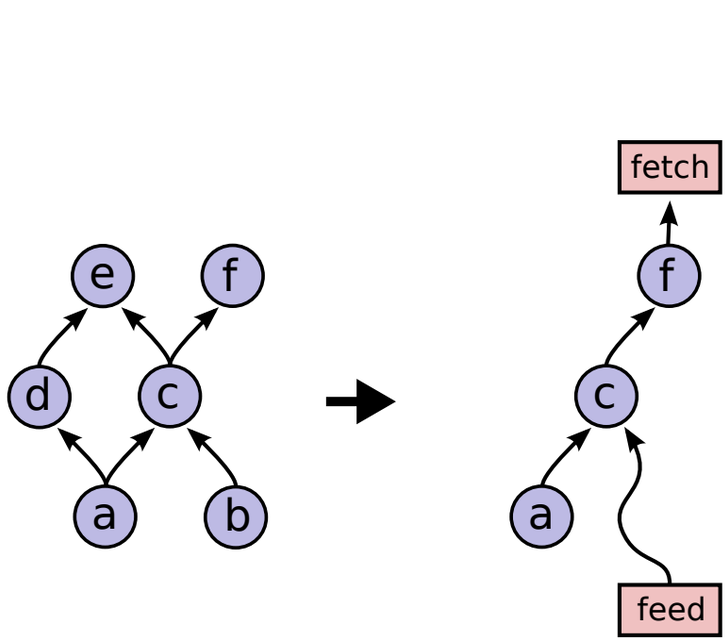


Figure 6: Before and after graph transformation for partial execution

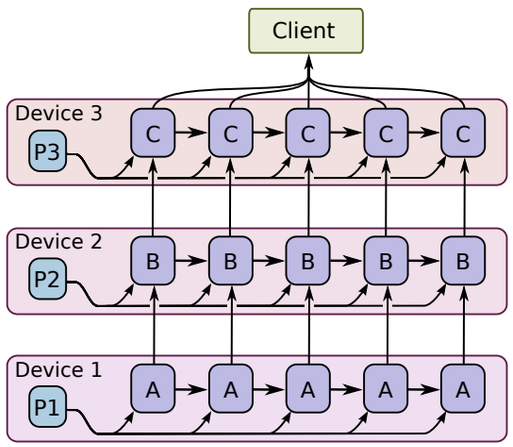


Figure 8: Model parallel training

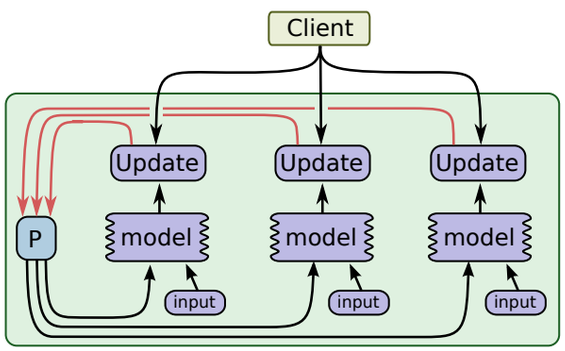


Figure 9: Concurrent steps

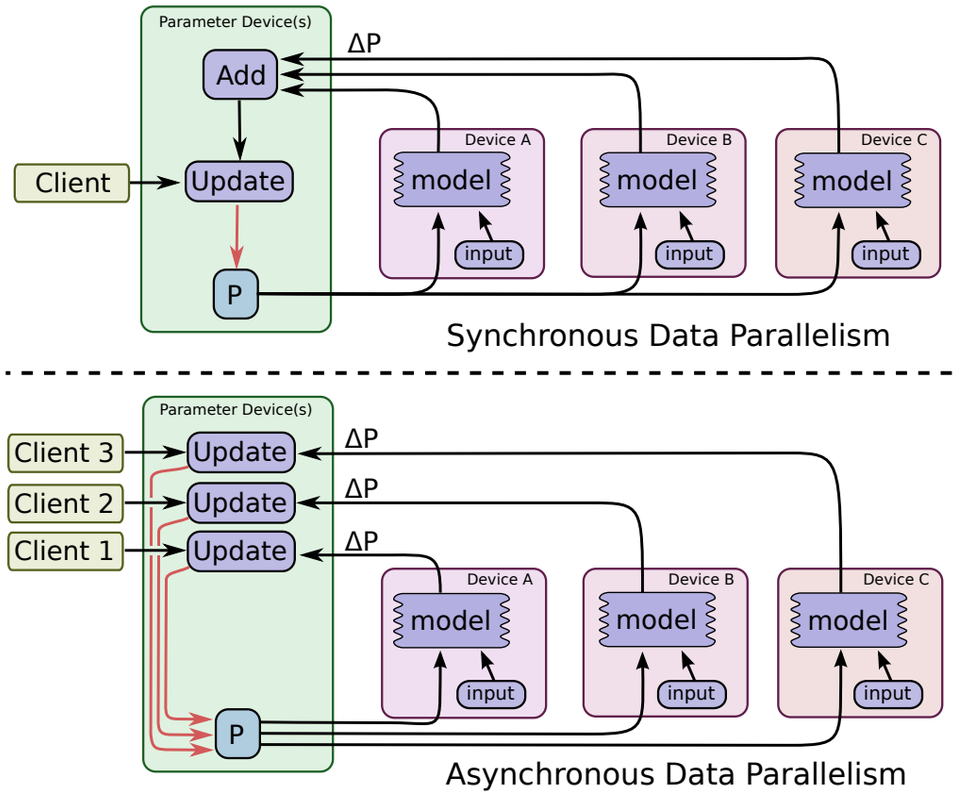


Figure 7: Synchronous and asynchronous data parallel training

From TensorFlow Whitepaper

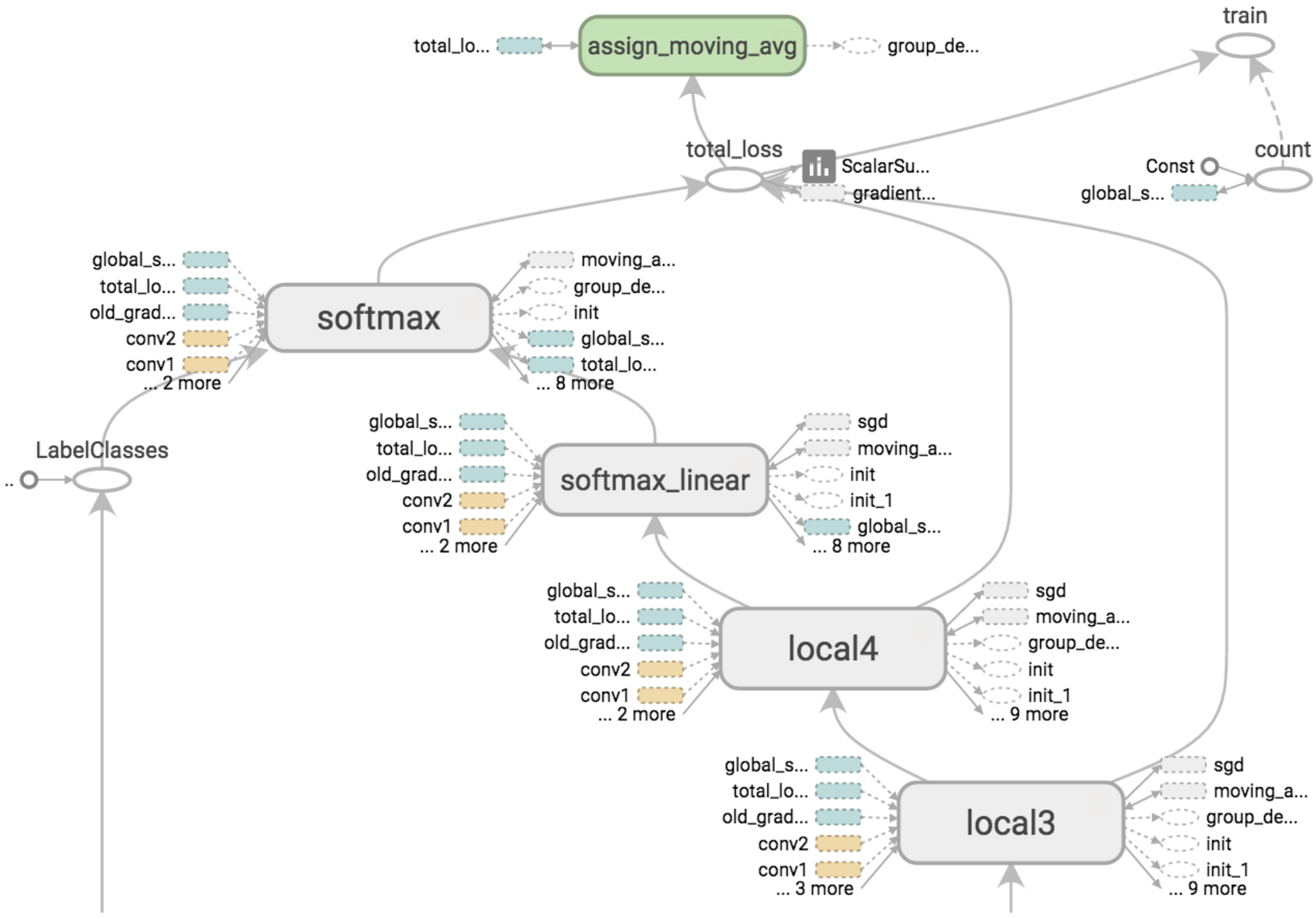


Figure 10: TensorBoard graph visualization of a convolutional neural network model

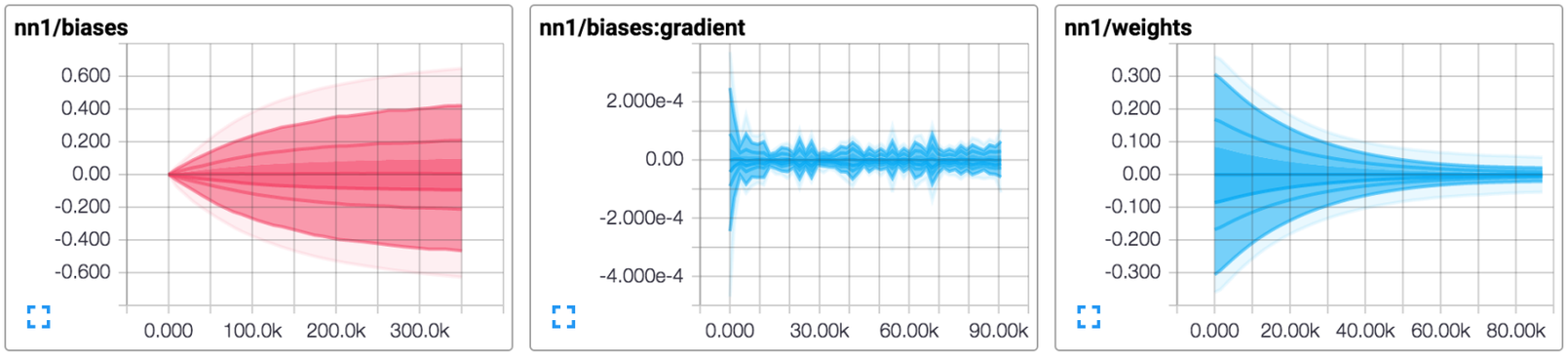


Figure 11: TensorBoard graphical display of model summary statistics time series data

DAG/Tensor HEP Framework?

- I can imagine building Reconstruction, DAQ, or Trigger based on this system.
- Reconstruction is extremely complicated…
    - Some steps (e.g. FFT deconvolution) could be implement in a few lines using existing ops.
    - Complex algorithms can be written into Ops/Kernels.
    - These can be wrapped by an abstractions analogous to ART Modules/Gaudi Algorithms.
        - Similar to high level DNN SW build on top of Theano.
- Our Data is complicated
    - Representation as tensors is efficient for vectorization, GPUs, and easily segmenting data.
    - High Level Objects can wrap these objects.
    - Not sure Event Store is necessary in Data Flow approach… could be faked (e.g. simple python dictionary)
        - Data isn't fetched at run-time.
- Not sure how to handle Conditions Database, Geometry, etc…
- I think implementing WireCell in TensorFlow or Theano is perfect R&D for such an idea.

DNN Reco Discussion

- DNNs replace algorithmic reconstruction?
    - Perhaps for Neutrino physics since classifying events and measure neutrino energy is sufficient.
- Really 4 different reconstruction options: Automatic Reco, Hand Scan, Assisted Reco, or DNN.
- Is it easier?
    - Optimal network and training requires careful craftsmanship and lots of “experiments”.
        - This work is analogous to writing algorithms and performing studies…
            - Arguably *less technically challenging*. No code necessary.
    - Systematics are still the same. Analysts will mostly still have to do all of the same work.
        - Assemble necessary signal and background samples. Make estimates.
        - Optimize cuts: Choose a point on the roc curve. (1-D)
        - Estimate systematics. New Systematics due the DNN?

DNN Calorimetry Subtleties

- Requires some thinking about DNN structure
    - Full Calorimeter or Cluster?
        - Clusters: Address eta/phi dependence with some linear at low resolution and then use regional CNNs.
    - *Non-uniform segmentation* in longitudinal sampling layers
        - Presents challenge to CNNs.
            - Consider example solution: layer wise 2D Convolutions/Pooling into 3D feature maps
                - How do you align features?
        - Fully connected? 200K cells is relatively small. Clusters are even smaller.
    - *Cylindrical boundary conditions*: simple solution is to duplicate phi strip
- Need cells (some clustered cells and Tower grid in xAOD)
- Pileup?
- Need large training samples… e.g. Photon Filtered Jet Samples