Machine learning and real-time analysis

Steven Schramm

1^{st} real-time analysis workshop
Institut Pascal, Université Paris-Saclay
July 16, 2019

- Machine Learning (ML) is a topic of increasing prominence in physics
- While ML can be useful, I would not suggest using it as a fix-all tool
  - ML is not always an improvement over what we have now
  - Even when it is an improvement, sometimes the benefits are outweighed by drawbacks
  - ML also has its own costs, especially in terms of training time and interpretability
- That said, in the right situation, ML can be extremely powerful
- ML can be used both within and external to real-time environments
  - Most "offline" ML developments can be used in real-time, given enough computing power
  - I will be focusing on online-specific ML developments, and ways to adapt offline to online

Sources/biases

- I am part of the ATLAS collaboration
  - I will thus be showing mostly ATLAS examples due to familiarity and lack of time
    - These are only examples - they are not meant to claim that ATLAS did task X first
  - The concepts from the examples generally apply to other HEP experiments
  - I have less experience in extrapolating these beyond HEP, but I am happy to discuss
- For more diverse results, I encourage you to check out talks from the CERN IML events
  - IML = the inter-experimental machine learning group, primarily but not only LHC
  - All standard IML meetings can be found here: indico category link
  - IML annual workshops can be found here: 2017, 2018, 2019

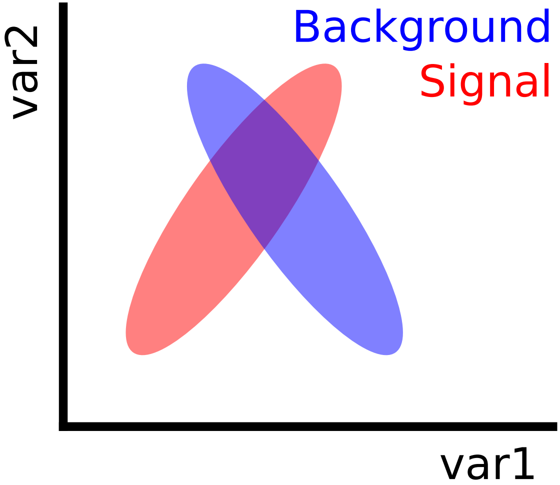
Overview

UNIVERSITÉ DE GENÈVE

- **Introduction to machine learning**
- Real-time analysis strategies and constraints
- Real-time analysis machine learning applications
- Summary

Classification example

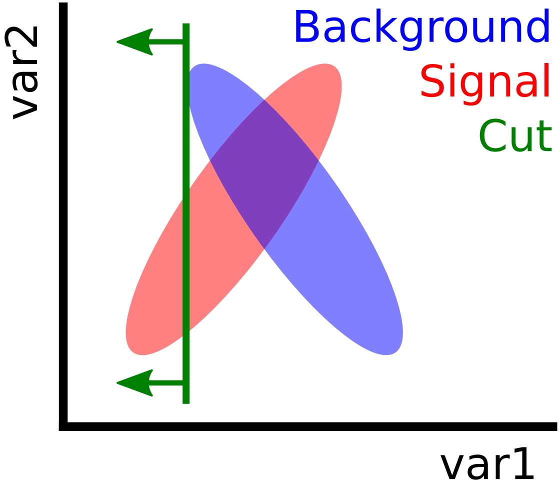
- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?



Classification example

UNIVERSITÉ DE GENÈVE

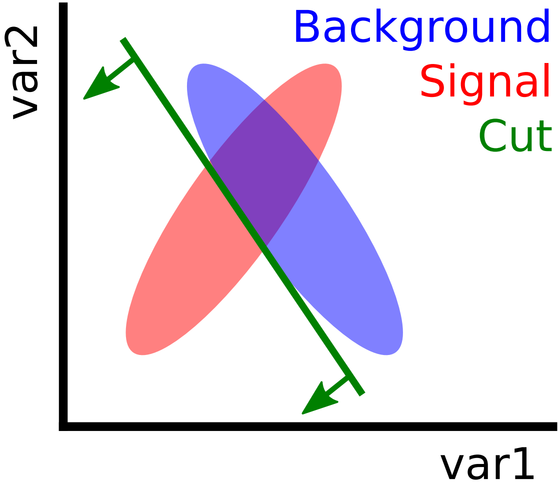
- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?
  1. Cut on variable(s) independently



Classification example

UNIVERSITÉ DE GENÈVE

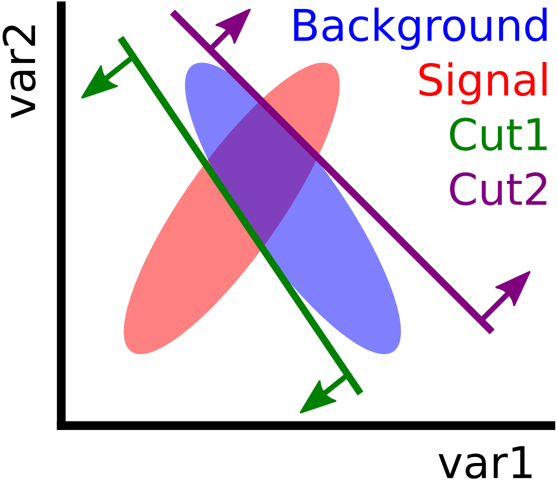
- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?
  1. Cut on variable(s) independently
  2. Cut on var1 and var2 simultaneously



Classification example

UNIVERSITÉ DE GENÈVE

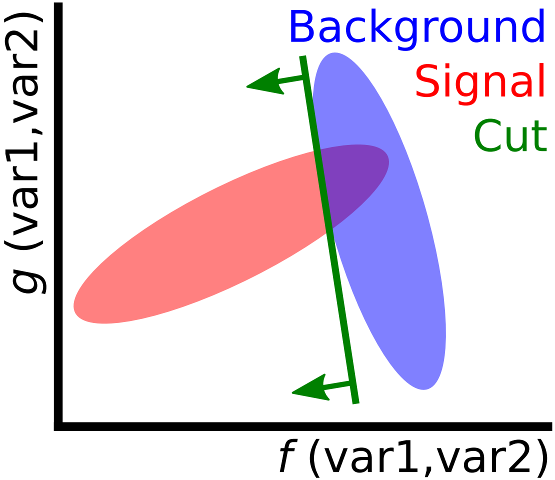
- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?
  1. Cut on variable(s) independently
  2. Cut on var1 and var2 simultaneously
  3. Partition the parameter space and simultaneously cut multiple times



Classification example

UNIVERSITÉ DE GENÈVE

- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?
  1. Cut on variable(s) independently
  2. Cut on var1 and var2 simultaneously
  3. Partition the parameter space and simultaneously cut multiple times
  4. Calculate new properties $f()$ and $g()$; cut simultaneously on them instead



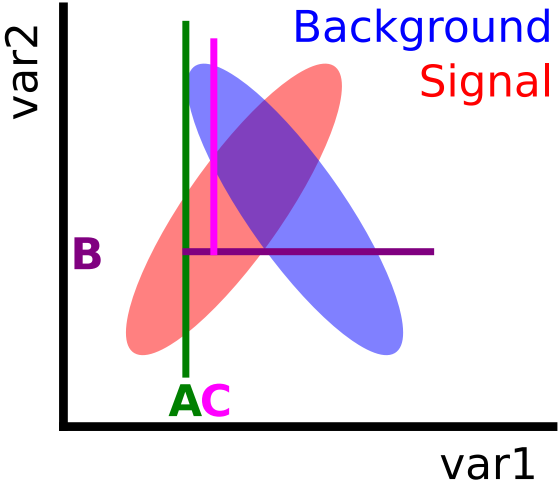
Classification example

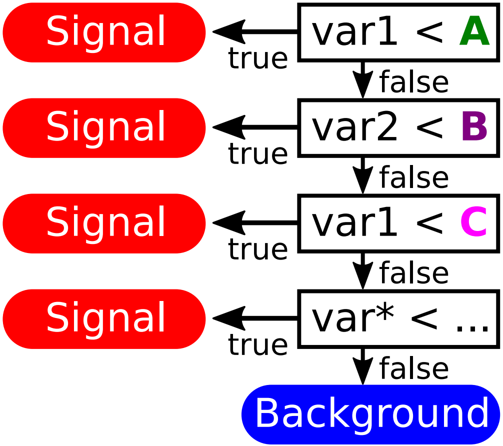
UNIVERSITÉ DE GENÈVE

- Typical HEP use case:
  separating signal and background events
- Two discriminating variables available
- What can we do?
  1. Cut on variable(s) independently
  2. Cut on var1 and var2 simultaneously
  3. Partition the parameter space and simultaneously cut multiple times
  4. Calculate new properties $f()$ and $g()$; cut simultaneously on them instead

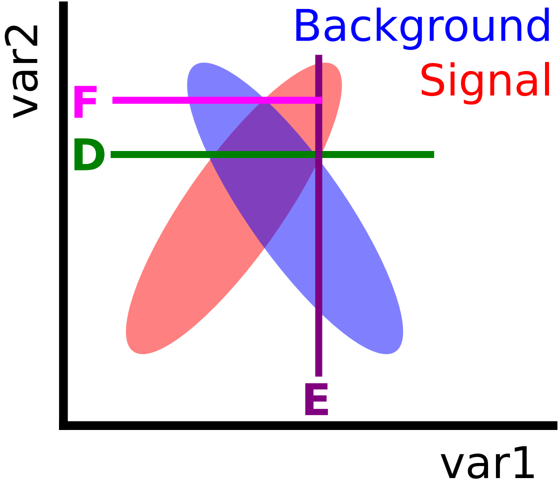
- As a rough conceptual analogy:
  - #3 $\sim$ Boosted Decision Trees (BDTs)
  - #4 $\sim$ Neural Networks (NNs)

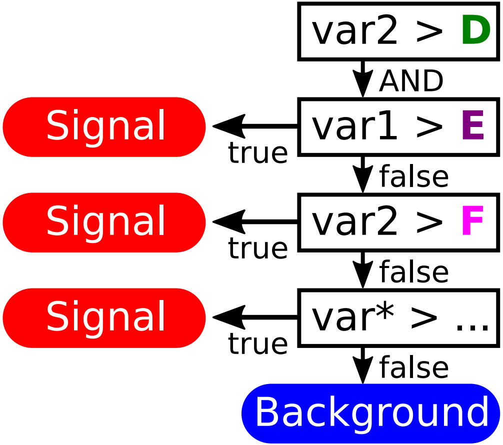
Classification example, BDT perspective





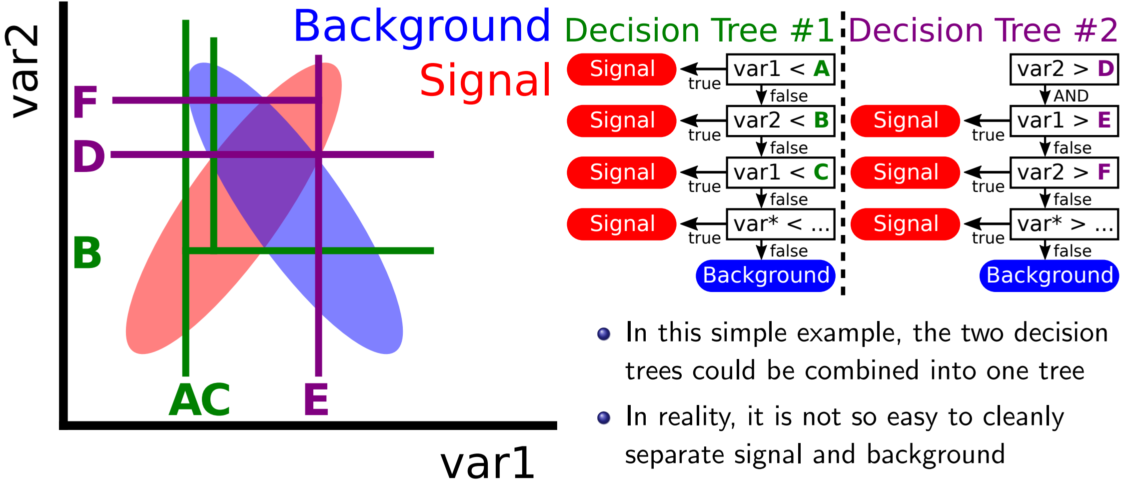
Classification example, BDT perspective





Classification example, BDT perspective

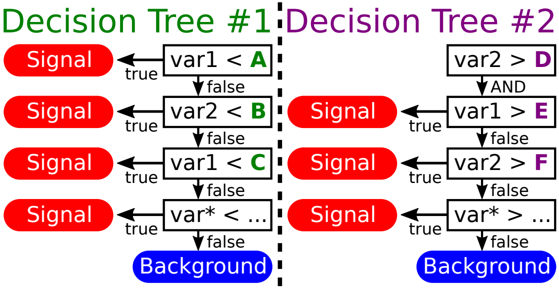
UNIVERSITÉ DE GENÈVE



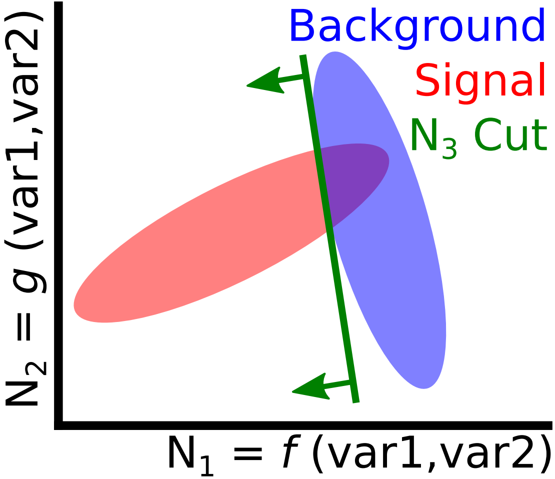
- In this simple example, the two decision trees could be combined into one tree
- In reality, it is not so easy to cleanly separate signal and background

Classification example, BDT perspective

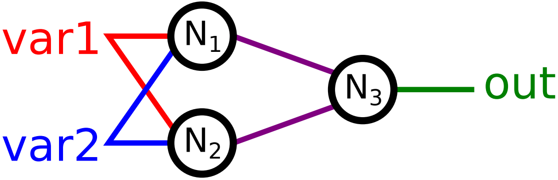
- In reality, partitions are not perfect
  - Background is non-zero in signal region
  - Want a non-negligible amount of signal
    $\Longrightarrow$ need to keep some background
- Each tree has a misclassification rate
  - Define final discriminant as combination of individual trees, weighted by misclassification rates
  - Not actual fraction of events, rather the "loss function" used in training
- Discriminant $= c_1 \cdot \mathrm{DT1} + c_2 \cdot \mathrm{DT2} + \ldots$



Classification example, NN perspective



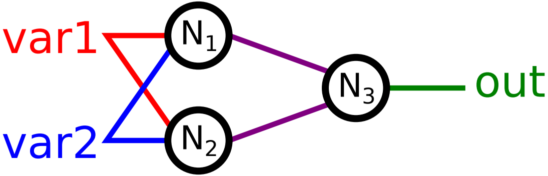
- Neural networks are a bit more complex
  - Cut on convolved input combinations
- In analogy to the past example:
  - Nodes N_1 and N_2 are $f()$ and $g()$
  - Cut on N_3; convolution of $f()$ and $g()$



Classification example, NN perspective

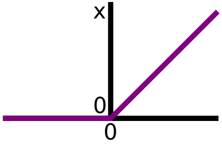
How this works (in very brief):

- Nodes 1 and 2:
  - Inputs: {var1, var2}, {var1, var2}
  - Parameters: $\{c_1, c_2, b_1\}$, $\{d_1, d_2, b_2\}$
  - Activation: ReLU, for non-linearity
  - $\mathrm{N}_1 = \max(0, c_1 \cdot \text{var1} + c_2 \cdot \text{var2} + b_1)$
  - $\mathrm{N}_2 = \max(0, d_1 \cdot \text{var1} + d_2 \cdot \text{var2} + b_2)$
- Node 3 (the final discriminant):
  - Inputs: N_1 and N_2
  - Parameters: a_1, a_2, b_3
  - Activation: Sigmoid, for a probability
  - $\mathrm{N}_3 = 1 \Big/ \left[1 + e^{-(a_1 \cdot \mathrm{N}_1 + a_2 \cdot \mathrm{N}_2 + b_3)}\right]$



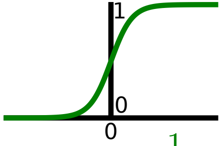
ReLU

(Rectified Linear Unit)



$$f(x) = \max(0, x)$$

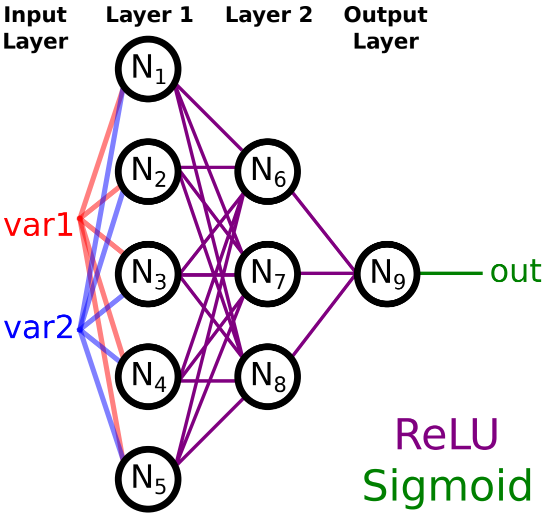
Sigmoid



$$f(x) = \frac{1}{1 + e^{-x}}$$

Classification example, NN perspective

- The last slide was a bit simplistic
  - This is slightly more realistic
- Layers 1 and 2 are "hidden layers"
  - Hidden = neither inputs nor outputs
  - If there are at least two hidden layers, then the network is "deep"; a DNN
- Output layer could have multiple nodes
  - One output = binary discriminant (signal vs background)
  - 2+ outputs = multi-class discriminant (signal vs BG1 vs BG2 or similar)



Classification example, summary

- This example gives a very rough idea of how BDT and NN classifiers work
  - BDTs construct a "forest" of decision trees to partition the parameter space into signal- and background-dominated regions, where the trees are weighted for optimal discrimination
  - NNs form non-linear combinations of the inputs and convolutions thereof to define a cut in a new parameter space, which it finds to provide optimal discrimination
- BDTs and NNs are nowhere near the only ML classifiers that are available
  - They are just the most prominently used types in HEP
- Classifiers are also not the only application of ML
  - They are just the easiest to motivate in such a context

What all is ML used for in HEP?

UNIVERSITÉ DE GENÈVE

- The most prominent examples of ML usage in HEP are:
  - Anomaly detection: identifying outliers which are not consistent with the bulk of the data
  - Classification: telling signal and background apart
  - Clustering: determine related groups within the data with similar features
  - Generation: creating artificial data which is ideally consistent with real data
  - Regression: calibration and similar uses to correct the value of a quantity of interest
  - Apologies to anyone whose favourite use case is not listed
- We will be going through examples of many of these in the context of real-time analysis

Overview

- Introduction to machine learning
- **Real-time analysis strategies and constraints**
- Real-time analysis machine learning applications
- Summary

Real-time analysis strategies

- Traditionally speaking, real-time computing implies two constraints
  1. The response to inputs is **guaranteed** to occur within a given time period
  2. The response to inputs is sufficiently fast to react to them **before the inputs change**
- Strictly speaking, only hardware programs (FPGAs or similar) can fulfill constraint #1
  - However, software programs can be effectively real-time if the typical response time is sufficiently low for #2 and any outliers can be handled in a non-disruptive manner

- There are two primary HEP use cases that I will discuss, and which I may differentiate
  1. Triggering: reacting = deciding whether to keep or reject events
     - Must occur before any buffer overfills, otherwise write it out anyway (non-disruptive failure)
  2. PEB: reacting = calculating the information that we want to use later in analysis
     - Must occur before any buffer overfills, otherwise potentially disruptive failure (?)
     - PEB = Partial Event Building, used for DataScouting / TriggerLevelAnalysis / TurboStream

Triggering

- A typical trigger workflow (some steps may be skipped depending on the trigger):
  - Step 1: read-out the detector in very coarse granularity for every LHC collision
  - Step 2: identify potentially interesting regions using simple hardware algorithms
  - Step 3: if the simple algorithms say the event is interesting, pass to the software
  - Step 4: read out the region(s) previously indicated as interesting with fine granularity
  - Step 5: reconstruct the object(s) with higher precision and decide if the event is interesting
  - Step 6: read out a larger portion of the detector or full detector with fine granularity
  - Step 7: reconstruct the object(s) with very high precision and decide if the event is interesting
  - Step 8: write the interesting event to disk for long-term storage
- Note that LHCb is changing this workflow for Run 3 as the hardware level is disappearing
- Triggering by definition is a form of **classification** - identify interesting events
  - ML therefore has some obvious applications to potentially improving this classification
  - Beyond classification, the above list also has clear links to clustering, regression, and more

Triggering vs partial event building

- Triggering is by its nature a choice of what is interesting
  - Ask a room full of physicists which events are interesting and you'll get very different answers
  - If we could, we would just record **every single event**, and most of them would be used
- The reality is that we can't record everything due to computing limitations
  - We can only write out so much data per second (bandwidth)
  - We can only store a given amount of data for a long period of time (disk resources)
- What we store is typically limited by the bandwidth, where bandwidth = size $\times$ rate

$\Longrightarrow$ If we can reduce the event size, we can increase the rate and thus store more events

- This is the principle of PEB: only reconstruct and record part of the event

Partial event building for analysis

UNIVERSITÉ DE GENÈVE

- There are different levels of information storage, including three major steps
  1. Store the full event (normal "offline" strategy)
  2. Store the information needed to reconstruct all objects of interest (regional read-out)
  3. Store only the objects and key information needed for the final analysis
- PEB can also be run either parasitically or actively
  - Parasitic = read-out and reconstruction already done for other triggers ($\sim$no CPU cost)
  - Active = using extra CPU to read-out the detector and reconstruct events
- Very important to keep this in mind if considering PEB-specific ML uses
  - PEB analyses are typically very high rate $\Longrightarrow$ calling the code **a lot**
  - This may result in very high CPU cost, which may make it so the PEB analysis is not possible
- For active PEB analysis, need to be careful about what to do if time runs out
  - If you didn't calculate all desired information, by definition you can't write it out
  - Perhaps fall-back to writing the full event to a different output stream and recover it offline?

Real-time analysis constraints

UNIVERSITÉ DE GENÈVE

- Before discussing where ML can help, need to understand real-time analysis constraints
- When talking about a given task, considerations include:
  - Dependencies: what has to happen before the task can run
  - Environment: what environment the task must run in
    - Hardware environment: FPGAs, GPUs, CPUs, etc
    - If CPU-based, execution environment: single-threaded, MP, MT, etc
    - If software-based: Whether the computing farm is homogeneous or heterogeneous
    - If software-based: CPU power, memory limits, CPU↔GPU transfer speed, etc
  - Latency: amount of time it takes to complete the task
  - Performance: how close the result is to the optimal result given "infinite" time
  - Reactivity: ability to quickly identify when something has gone wrong and fix it
  - Storage: amount of space it takes to "permanently" store the result

Overview

UNIVERSITÉ DE GENÈVE

- Introduction to machine learning
- Real-time analysis strategies and constraints
- **Real-time analysis machine learning applications**
- Summary

Potential ML applications

UNIVERSITÉ DE GENÈVE

- As discussed in the last section, there are many constraints in real-time systems
  - Whether or not a given approach is possible will depend on these constraints
- For the most part, I will be focusing on software real-time analysis
  - There will be a tutorial on hls4ml later today which will cover hardware ML applications
- I will also be focusing on concepts rather than specifics

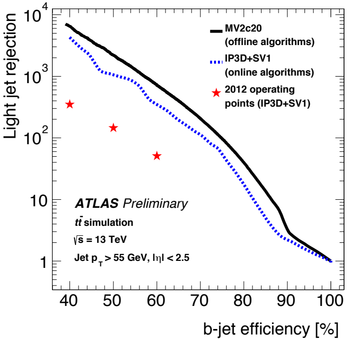
- Topics will generally fall into four categories:
  1. Classical uses of ML (very brief)
  2. Addressing differences between real-time and offline environments
  3. Anomaly and novelty detection
  4. Reducing computing costs

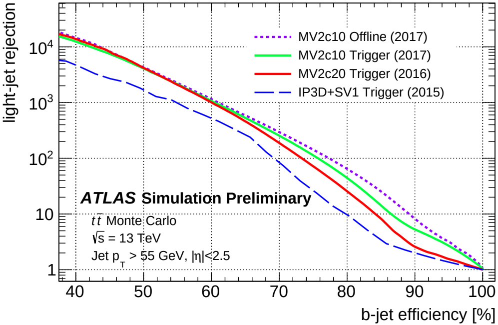
Classical uses of ML

Both: *b*-jet trigger plots

UNIVERSITÉ DE GENÈVE

- The primary use of ML in real-time environments so far has been classification
  - *b*-jet, electron, and photon identification are common examples
- This works well, but offline identification is a constantly moving target
  - This means that trigger ID is always out of date
  - Offline changes impact PEB less, but still impact uncertainties if adapting from offline





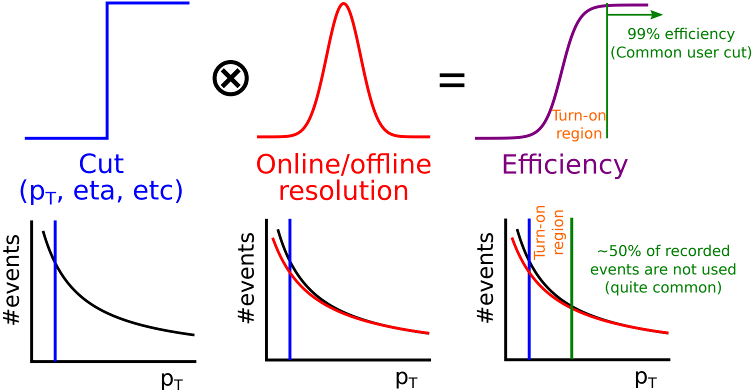
Implications of a moving reference

- ML is a powerful tool for identification, and is used both online and offline
  - Lots of developments in ML classification, can be used in both environments
  - CNNs, RNNs, and much more are increasingly used for improved identification
  - Not the focus of this talk, which is about adapting ML for trigger
- However, we just saw an example of how the offline target is constantly evolving
- In the context of identification for triggering, implications include:
  - Wasted rate, as some of the events recorded will not be used offline
  - Missed events, as some of the events desired offline were not recorded
  - Increased work-load, to calculate trigger-vs-offline efficiency scale factors
  - Duplicated work, as the trigger tries to replicate offline studies to catch up for the next year
- Moreover, identification is not the only stage in triggering
  - After identifying an object, typically you apply a kinematic selection

Moving references and kinematic selections

UNIVERSITÉ DE GENÈVE

- Differences in object kinematics are typically shown as "turn-on curves"
  - Ideally, online = offline, then we have a step function
  - In reality, there is a resolution difference, which leads to the characteristic shape
- As before, these differences can waste a lot of rate and increase the work-load

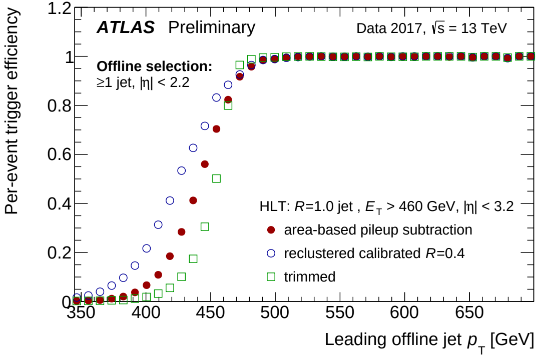


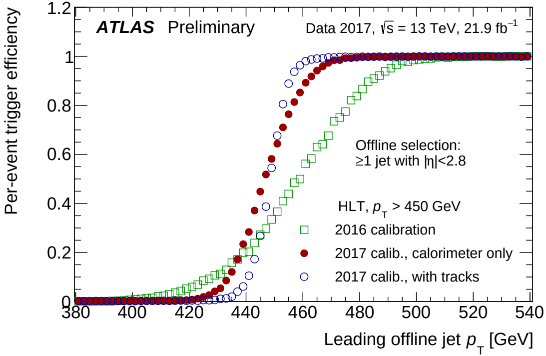
Kinematic selections example

Both: Jet trigger plots

UNIVERSITÉ DE GENÈVE

- This is clear in jet trigger turn-ons (note: no ML here, just as an example)
  - Bringing jet definition and calibration closer to offline leads to sharper turn-ons
  - However, offline target is constantly moving, as for *b*-jet identification
- Can ML help us reduce the impact of online/offline differences?





Addressing online/offline differences

UNIVERSITÉ DE GENÈVE

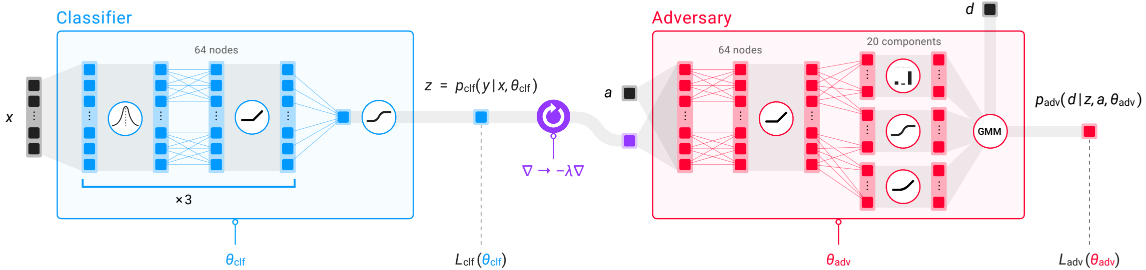
- Can ML help us to address such online/offline differences?
  - Yes, and in fact a lot of work is in this area in a different context
  - Major critique of ML in physics: potential to learn simulation features
- Large effort has thus been invested in ensuring ML does not learn specific features
  - Trigger can benefit: online vs offline instead of simulation vs data
- There are also other areas that are not as relevant to offline analysis

Adversarial training

Schematic: PUB-2018-014

UNIVERSITÉ DE GENÈVE

- Adversarial NNs are one way to intentionally not learn a given set of properties
  - Idea: put two networks in competition, the classifier and adversary
  - Classifier tries to reject background, while adversary tries to predict the properties in question
  - If the adversary can predict the properties from the classifier output, there is a correlation
  - Continue to train until balanced: classification without learning the specific properties
- Choosing values of λ allows for prioritizing classification, decorrelation, or balance

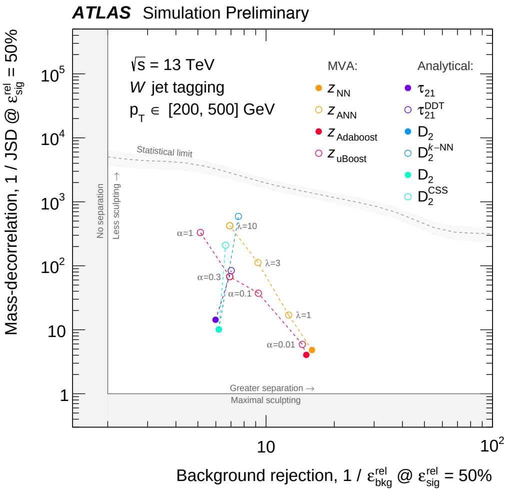


Comparison of decorrelation strategies

Plot: PUB-2018-014

UNIVERSITÉ DE GENÈVE

- Adversarial networks are only one decorrelation strategy
- Study of single-variable decorrelations
  - Adversarial networks (NN)
  - Uniform boosting (BDT)
  - k-nearest neighbours
  - Two other application-specific options
- ANN worked the best overall



Decorrelation strategies and the trigger

UNIVERSITÉ DE GENÈVE

- Why is such a decorrelation strategy useful in the trigger?
  1. Designing triggers decorrelated with respect to given properties (resonance mass, etc)
  2. You know a given variable does not agree well online/offline and is correlated to variables you want to use in your trigger classifier, so train the classifier to not learn that feature
  3. Train the classifier on a mixture of online and offline events and use an adversary to ensure the classifier cannot tell the difference between online and offline events
     - In other words, decorrelate with respect to the input type label
     - Similar can be done for a data/simulation mixture control region and not learn differences
- Note: this will degrade performance (you are intentionally discarding information)
  - Make sure to balance decorrelation with performance as needed
- Note: similar can be done by training a classifier on the latent space of an autoencoder

Regressions and online/offline differences

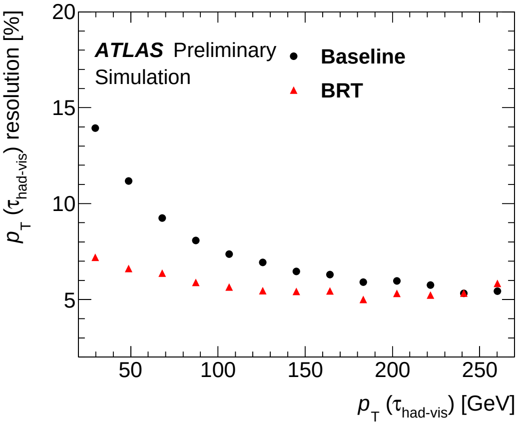
- Adversaries and similar are helpful for identification tasks in the trigger
- Kinematic selections are also very important, where resolution differences are crucial
  - As before, $\sim$50% of recorded events may not be used due to resolution differences
- How can we reduce online/offline resolution differences?
  - Regressions are one key ML tool for such cases

Regressions for resolution

Plot: CONF-2017-029

UNIVERSITÉ DE GENÈVE

- Regressions are essentially advanced ML calibration strategies
  - Rather than calibrating an object with a few inputs, use many variables
  - Exploit variable correlations
  - Train the regression to predict the desired quantity (four-vector)
  - Loss function: can focus on mean (scale), variance (resolution), or other
- Offline example: huge resolution gains when using a regression for τ calibration



Regressions for online/offline resolution

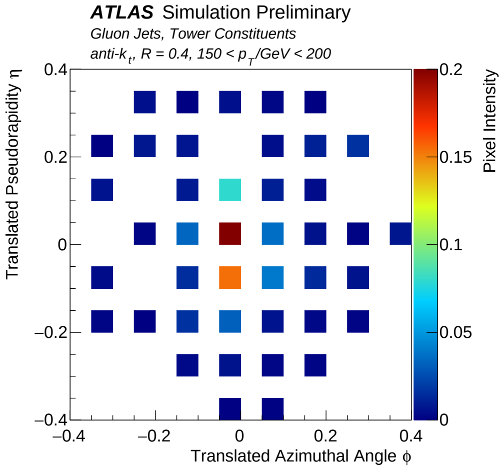
- Regressions are excellent for reducing online/offline differences
  - Can fully exploit the correlations between many variables
  - Can be given online inputs and trained to predict offline four-vector (not truth)
  - Can be trained to prioritize scale, resolution, or somewhere in between
    - For trigger, resolution is generally more important
    - For PEB, scale may be more important, and reference (truth or offline) depends on use case
- Lots of opportunities to use regressions in real-time analysis
  - However, still very susceptible to the changing offline reference challenge
  - Can try to combine with ANN/similar to train one regression that works for online and offline

Regressions for calibration at Level-1

Plot: PUB-2017-017

UNIVERSITÉ DE GENÈVE

- Regressions can be run on a variety of inputs, not just object properties
  - For example, they can be run on images (convolutional NNs)
- This can be very useful at Level-1
  - Insufficient time to calculate features
  - Coarse readout is fine for CNN (readout size defines pixel size)
  - Designed well, CNNs can run on L1 (I expect hls4ml will show L1 CNNs)
- Result: hopefully much improved L1 resolution (and much faster L1 turn-ons)
  - Maybe supports 40 MHz PEB analysis?



Anomaly and novelty detection

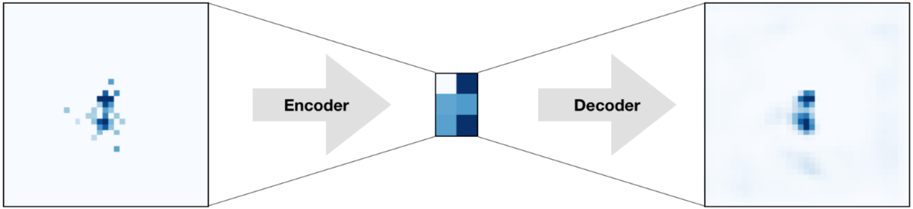
UNIVERSITÉ DE GENÈVE

- We're now moving to a more exotic topic, which has recently exploded in popularity
- Premise: what if our analyses and triggers are looking in the wrong place for new physics?
  - Entirely possible - we are generally looking for what we expect, not something crazy
- Potential solution: use ML to directly search for the unexpected
  - This is traditionally known as anomaly detection, as you are looking for outliers
  - Sometimes also referred to as novelty detection - the detection of something new
- There are lots of ways to do this, with differing levels of generality
  - We had to hold two back-to-back IML meetings on this topic to cover all requests: #1, #2

A more specific strategy

Schematic: arXiv:1808.08992

- Let's focus on jets as a generic object leaving energy in the calorimeter
- Train an autoencoder on a sample of background (QCD) jets
  - An autoencoder "learns" the identity matrix, with some noise
  - The output of an autoencoder should thus be the same object
  - If the autoencoder gets something it doesn't expect, the object will change (it hasn't learned the identity matrix for that type of object)

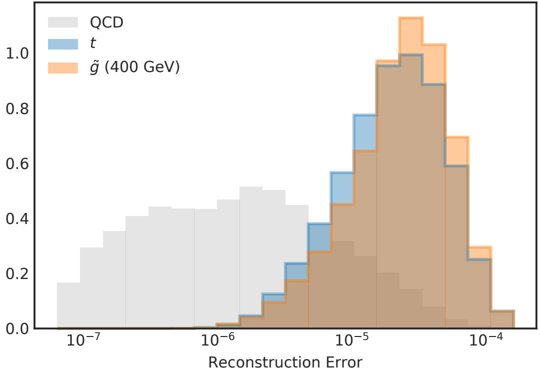


Searching for anomalous jets

Plot: arXiv:1808.08992

UNIVERSITÉ DE GENÈVE

- Autoencoder was trained on QCD jets
  - QCD jets are thus reconstructed similarly (low reconstruction error)
  - Both hadronic top quark decays and gluinos show up as anomolous jet types
- This principle can be expanded
  - Train on all known jet types
  - Will encounter beam background, etc
- Continue to expand autoencoder to reach non-background anomalous jets
  - Then start triggering on anomalies
  - Could identify long-lived particles, etc

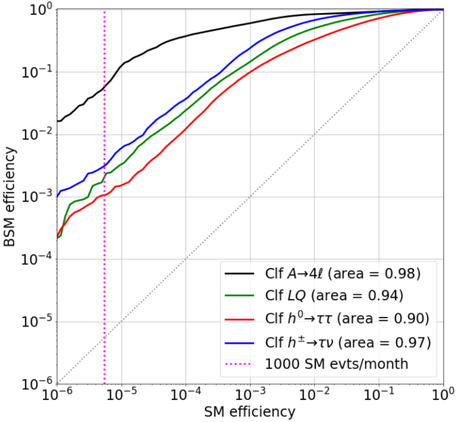


A more general strategy

Plot: arXiv:1811.10276

UNIVERSITÉ DE GENÈVE

- Again, use a variational autoencoder
  - Uses 21 high-level features as inputs
  - Train on a mixture of dominant SM processes: QCD, W, Z, and $t\bar{t}$
  - Test on a few signal samples
- Significant SM rejection with reasonable signal acceptance, despite never having seen the signals during training
  - This is going in the direction of a generic anomaly detection trigger
  - Would be curious to see how this looks in data (detector features, etc)



General thoughts on anomaly and novelty detection

UNIVERSITÉ DE GENÈVE

- These strategies are very interesting for the future
  - Give us the chance to look for the unexpected, without restricting to a specific model
  - Triggers based around such algorithms may indicate the presence of new physics
- However, these are also long-term endeavours
  - You will start by discovering detector features, beam background, etc
  - Only after a lot of work will you get to actual exotic events (SM-generated or otherwise)
- Note that such strategies are also useful for monitoring
  - In this case, the detector features and similar would be what you are looking for
  - Watch for changes that would indicate new detector problems or similar

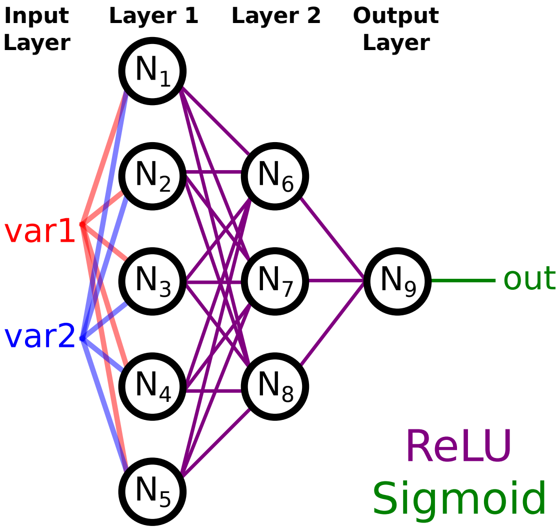
Reducing the computing costs of ML at run-time

UNIVERSITÉ DE GENÈVE

- After deciding that ML is useful for a given application, that is just the start
  - You then have to implement the ML technique and demonstrate it works
  - You also need to make sure that it will run within the computing resource constraints
- ML algorithms are notoriously expensive to train, but they can also be expensive to run
  - BDTs and NNs both require very large amounts of floating point operations
  - CPUs have some FPUs, but really they excel at integer arithmetic
- Large ML models can also take up a sizable amount of memory
  - Training an ML algorithm is essentially fitting a large number of floating point parameters
  - Depending on the size of the model, it may start to hit memory limitations
- How can we address these challenges to make ML more real-time friendly?

Reminder of NN structure

- Number of parameters scales very quickly with the model size
  - N_{par} controls both memory and speed
- How many parameters is this model?
  - Nodes: one per input, +1 for offset
  - Layer 1: 3 parameters per node
  - Layer 2: 6 parameters per node
  - Output layer: four parameters
  - Total: 37 floating-point parameters
  - Generic: $\sum_{i=1..L}^{layers} N^{i}_{\text{nodes}} \cdot (N^{i-1}_{\text{nodes}} + 1)$
  - $N_{\text{par}} \in \mathcal{O}\left(N_{\text{layers}} \cdot \left[\max_{\text{layers}}\left(N_{\text{nodes}}\right)\right]^2\right)$



So how can we reduce computing requirements?

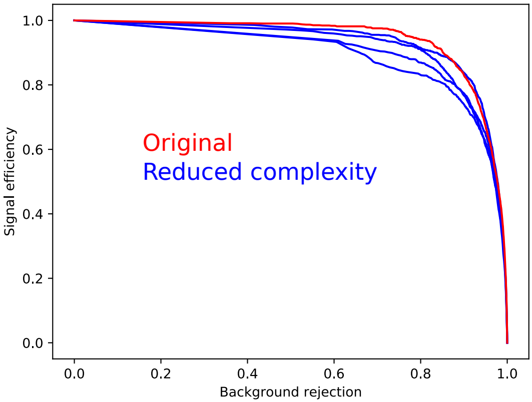
- Reminder: model evaluation speed and size are related to the number of parameters
- How can we make this more efficient?
  1. Run on hardware that is designed for floating point operations
  2. Reduce the number of layers or the number of nodes (less parameters)
  3. Compress the parameters themselves - move away from floating-point values
- Option #1 may or may not be feasible, depending on the environment
  - Not everyone has GPUs, and if they exist they may or may not be sufficient
  - Note that GPUs are not a magic solution - still requires CPU$\leftrightarrow$GPU transfer
- Options #2 and #3 may be needed in some cases
  - These are both powerful approaches with their own benefits and consequences
  - In both cases, the precision will be impacted, but typically it is a small loss

Reducing the model complexity

Plot: S. Benson @ IML2018

UNIVERSITÉ DE GENÈVE

- One option: train a compressed network to learn the output of a large network
  - Compressed network doesn't see truth labels, just sees what original network says for a given set of inputs
- Compressed network has fewer nodes
  - Faster to evaluate
  - Less to keep in memory
- Performance depends on many factors
  - However, often can reduce complexity quite a bit, keeping similar performance

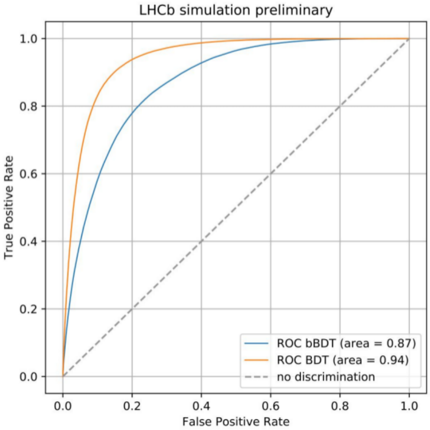


Discretizing the inputs

Plot: LHCb

UNIVERSITÉ DE GENÈVE

- Another option: discretize the inputs
  - Normally inputs are floats/doubles
  - Discretize to integers (bins)
  - Large size reduction
  - Also converts BDT into a lookup table
- Comes with a loss of performance
  - However, in real use, not as bad as plot
- Used extensively by LHCb in Run 2
  - This is the "Bonsai BDT"

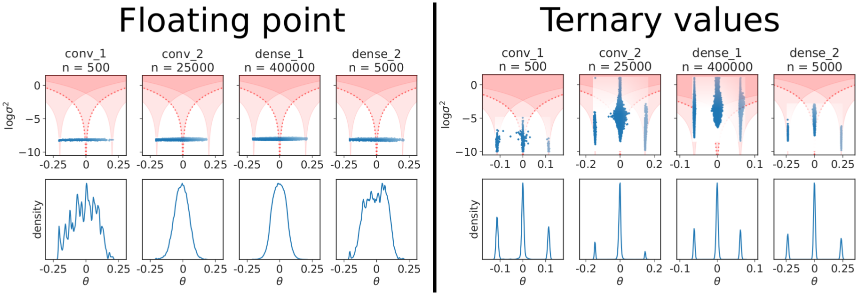


Ternary weights

Talk: T. Genewein @ IML2019

UNIVERSITÉ DE GENÈVE

- We can directly compress weights from floats (32 bit) to ternary (2+ bit)
  - Ternary weights $(-r, 0, +r)$ reduce space considerably
  - With combined ternary weights and pruning (dropping low-importance nodes, not shown) found identical performance to full floating point, but with significantly smaller sizes
- Generally a powerful means of reducing network size and parameters



ML for faster reconstruction

- Of course, ML algorithms are not always slower than non-ML code
  - Huge effort into tracking with ML, especially for the TrackML challenge
  - Recent two-day summary event: indico link
- As such, ML can also be considered to speed up very combinatorically complex tasks
  - Recall that ML is actually an approximate solution, not analytic
  - It can therefore be much faster - just need to make sure it is suitably performant
- Could be used in future trigger for reconstruction
  - Tracking at HL-LHC likely to benefit from ML-inspired designs

Overview

Summary

UNIVERSITÉ DE GENÈVE

- This was my perspective of ML topics that may be relevant to real-time analysis
- I covered four main categories:
  - Traditional uses of ML (object identification and classification)
  - Addressing differences between real-time and offline environments
  - Anomaly and novelty detection
  - Reducing computing costs and constraints
- This is by no means an exhaustive list, and I certainly missed topics
- I would say that we are currently in a very fortunate position
  - There has been **a lot** of ML development for offline HEP usage
  - It is now up to us to think about how we can benefit in the real-time environment