Use of Machine Learning Techniques for improved Monte Carlo Integration

Josh Bendavid (Caltech)

Caltech

September 23, 2016
Beyond Leading Order Calculations on HPCs
FNAL

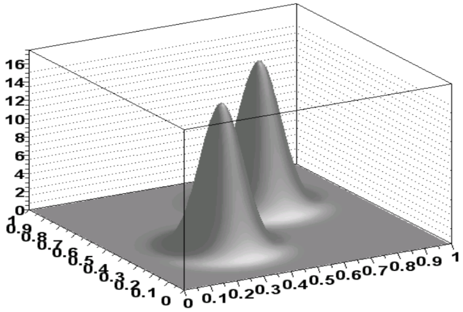
Introduction

- R&D Project Growing out of work on Multivariate Regression: New algorithm for Monte Carlo integration and event generation
- Outline
    - Brief intro to Monte Carlo integration/generation and (non-exhaustive) look at existing algorithms (VEGAS and FOAM)
    - Brief intro on Boosted Decision Trees and their conventional classification/regression applications
    - Adaptation to Monte Carlo integration/generation
    - Some very preliminary comparisons with existing algorithms
    - Discussion of alternative approaches with deep neural networks

Monte Carlo Integration and Generation

- **Monte Carlo integration:** Given an arbitrary/black box multidimensional function $f(\bar{x})$, find the integral $\int f(\bar{x})d\bar{x}$
- **Monte Carlo generation:** Given an arbitrary/black box multidimensional function $f(\bar{x})$, generate an unweighted set of vectors $\bar{x}$ with a probability density $p(\bar{x}) = f(\bar{x})/\int f(\bar{x})d\bar{x}$
- Typical HEP use case: Given a numerical implementation for a matrix element fully differential in incoming/outgoing four-vectors, compute the total cross section (integral), and generate a set of unweighted events

Monte Carlo Integration and Generation: Example Function



S. Jadach, physics/0203033

- This is the "camel" function from the original VEGAS paper, which can be generalized to N dimensions
- Factorized approach will not work well
- Significant low-density regions which cannot be easily excluded a-priori

Monte Carlo Integration: Brute Force Approach

- Simplest possible algorithm:
    - Randomly sample from a (multidimensional) **Uniform** distribution
    - Integration weight $w = Vf(\bar{x})$ (where V is the total volume of the space)
    - Integral $I = (1/N)\sum w$, $\sigma_I = \sigma_w/\sqrt{N}$
    - Generation: Use simple accept-reject sampling ($\epsilon = w_{max}/ < w >$)
- End result: Huge variance for weights $\rightarrow$ need a huge number of samples to get reasonable numerical precision (and very low unweighting efficiency for generation)

Monte Carlo Integration: Importance Sampling

- General idea, sample from some generating probability density $g(\bar{x})$ instead of uniformly:
  - Integration weight $w = f(\bar{x})/g(\bar{x})$
  - Integral $I = (1/N)\sum w$, $\sigma_I = \sigma_w/\sqrt{N}$
  - Generation: Use accept-reject sampling ($\epsilon = w_{max}/ < w >$)
- Ideal case: $g(\bar{x}) = f(\bar{x})/\int f(\bar{x})d\bar{x} \rightarrow$ try to construct some $g(\bar{x})$ that is easy to sample from **and** well approximates $f(\bar{x})$
- Different considerations for integration (minimize variance) vs generation (balance between variance and maximum weight)

Monte Carlo Integration: Importance Sampling

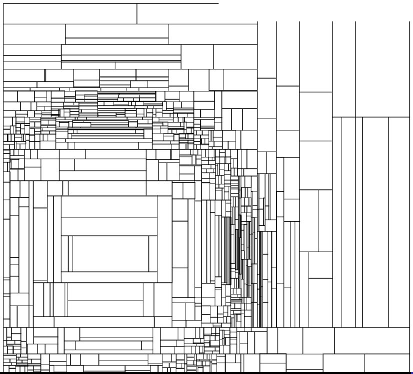
- Typical algorithm divided in two stages
    1. Construct appropriate sampling function $g(\bar{x})$ which approximates $f(\bar{x})$
    2. Generate a large number of events to evaluate the integral with maximum precision (or unweight with maximum efficiency)

VEGAS

- `G.P. Lepage, A New Algorithm for Adaptive Multidimensional Integration, Journal of Computational Physics 27, 192-203, (1978)`
- Iterative algorithm
- Start from uniform sampling distribution
- At each iteration, build an adaptive-binned histogram to approximate $f(\bar{x})$
- Multidimensional functions are handled as a simple product of one-dimensional histograms
- Building histograms is fast and relatively simple, but for higher-dimensional functions with non-trivial correlations there is a hard limit to the achievable weight variance/unweighting efficiency
- Carefully choosing/transforming integration basis can help (but not always possible)

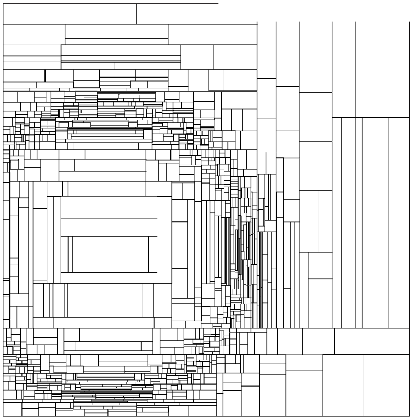
Foam

- S. Jadach, physics/0203033
- Improving on limitations of VEGAS requires true multi-dimensional sampling function
- Foam algorithm based on a single decision tree $\rightarrow$ divide up phase space into hyper-rectangles with optimized boundaries
- Phase-space is sampled uniformly **within each hyper-rectangle** to determine the next binary split until the stopping condition is reached

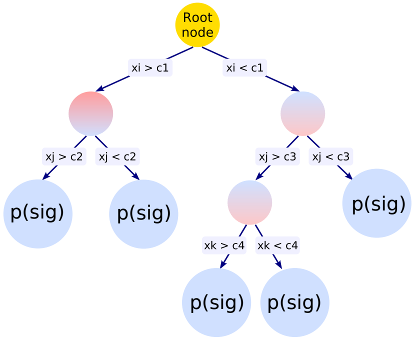


Foam

- Assign a weight to each hyper-rectangle (proportional to estimated integral inside)
- For each event: randomly choose a hyper-rectangle (probability proportional to its weight) then randomly sample within hyper-rectangle



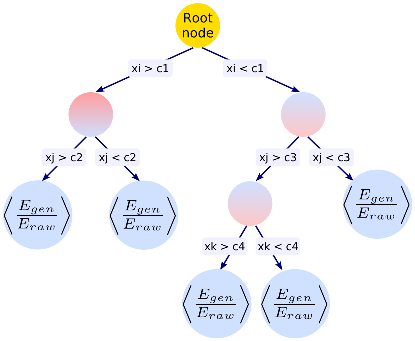
Boosted Decision Trees for Classification



- Decision Tree is a simple structure consisting of a set of connected "nodes"

- Intermediate nodes where a variable and cut value is selected to split events into two subsets
- Terminal nodes are assigned a response, in this case the relative signal probability $\frac{\mathcal{L}_s(\bar{x})}{\mathcal{L}_b(\bar{x})}$
- Multidimensional likelihood ratio is therefore approximated by a piecewise-continuous function over the multivariate input space
- **Boosting**: Construct a series of decision trees to improve the overall response

Boosted Decision Trees for Regression


Root node
xi > c1
xi < c1
xj > c2
xj < c2
xj > c3
xj < c3
xk > c4
xk < c4
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$
$\left\langle \frac{E_{gen}}{E_{raw}} \right\rangle$

- Boosted Decision Trees can also be used for multivariate regression problem
- Replace log likelihood ratio with generic function $f(\bar{x})$
- Minimize deviation between training sample and regression function
- Decision trees form a series of piecewise continuous approximations for the function $f(\bar{x})$ in the multidimensional input space

Boosted Decision Trees for Monte Carlo Integration (aka GBRIntegration)

- Evaluating amplitudes is the critical computational step for phase space integration or unweighting in MC generators
- Number of required phase space points depends on weights variance and/or unweighting efficiency
- **Insight:** Foam is based on a single decision tree, performance of MC integration can be improved by boosting as for classification and regression
- Basic limitation of Foam is that huge number of hyper-rectangles are needed for good performance
- Boosting allows to exploit combinatorics of terminal nodes between different decision trees
- Initial implementation based on GBRLikelihood tool developed for CMS photon energy regression

Boosted Decision Trees for Monte Carlo Integration (aka GBRIntegration)

- Basic Principle: Use a boosted decision tree to **directly** estimate function value $f(\bar{x})$ such that

$$g(\bar{x}) = \sum g_i(\bar{x}) \approx f(\bar{x}) \qquad (1)$$

- Where each $g_i(\bar{x})$ is an individual decision tree with some value assigned to each terminal node
- Trivial to compute integral for each tree (and for sum): $\int g_i(\bar{x}) = \sum V_{ij} g_{ij}$, where V_{ij} and g_{ij} are the volume and value assigned to each hyper-rectangle/terminal-node)

Sampling from a Boosted Decision Tree

- Sampling from a Boosted Decision tree is straightforward/efficient:
  - Randomly choose a tree from the series with probability proportional to its integral
  - Randomly choose a terminal node on the tree with probability proportional to its integral
  - Uniformly sample within the hyper-rectangle of the chosen terminal node
- Critical limitation: Any transformation breaks the above logic (ie cannot efficiently sample from $f(\bar{x})$ if $g(\bar{x}) = \sum g_i(\bar{x}) \approx \ln f(\bar{x})$)
- Critical limitation: Only works for **positive-definite** tree values

Constructing the Boosted Decision Tree

- Positive-definite limitation means that slow convergence is required (later trees cannot correct with negative values)
- Train two BDT's in parallel, one for sampling, and one to aid convergence (with transformation $h(\bar{x}) \sim \ln f(\bar{x})$ and no positive-definite constraint)
- BDTs for MC integration constructed iteratively (start with uniform sampling distribution in first iteration with primary and secondary BDT's initialized to a common small value)
    - Sample N events from current secondary (sampling) BDT series $g(\bar{x})$
    - Train tree for primary BDT $h(\bar{x})$
    - Train tree for secondary BDT $g(\bar{x})$
    - Repeat

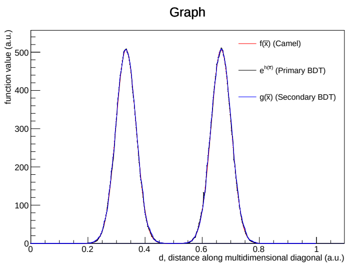
Some results - 4D Camel Function Integration

- Comparing Vegas, GBRIntegrator, Foam for 4-dimensional camel function (since this appears in both VEGAS and Foam papers).
- Given relative weight variance $\sigma_w / < w >$ after training/grid building, relative uncertainty on integral evaluated with N additional events is $\sigma_I / I = \frac{1}{\sqrt{N}} \sigma_w / < w >$

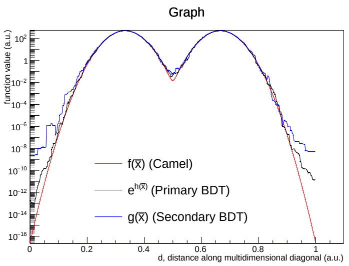
Algorithm	# of Func. Evals	$\sigma_w / < w >$	σ_I / I (2e6 add. evts)
VEGAS	300,000	2.820	$\pm 2.0 \times 10^{-3}$
Foam	3,855,289	0.319	$\pm 2.3 \times 10^{-4}$
GBRIntegrator	300,000	0.0819	$\pm 5.8 \times 10^{-5}$
GBRIntegrator (staged)	300,000	0.077	$\pm 5.4 \times 10^{-5}$

- 3x smaller weight variance to foam with 10x less function evaluations
- Substantially improved performance with respect to initial version of GBRIntegrator algorithm (lacking primary/secondary BDT paradigm)
- For this particular function VEGAS performance saturates at relatively poor weight variance

Diagnostic Plots - 4D Camel Function



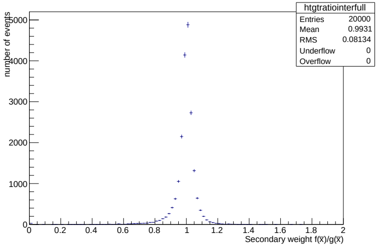
(a) linear



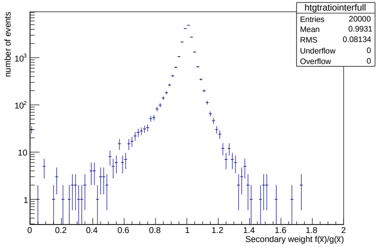
(b) log

- Secondary sampling BDT approximates function slightly worse in very low probability regions (related to initialization values, positive definite constraint during training, and lack of transformation). For this particular case, effect is small. (but this is the reason staged variation achieves slightly better precision)

Diagnostic Plots - 4D Camel Function - Integration Weights



(a) linear



(b) log

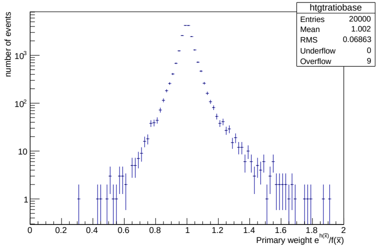
- Excellent weight distribution for integration purposes (symmetric, small variance)

Conclusions/Todo

- Very promising performance/potential for speedup of MC integration (and eventually generation) with BDT-based algorithm
- Also exploring alternatives based on generative neural networks (sample trivial input distribution, train appropriate transformation based on Jacobian)
- Todo:
  - More systematic tests at higher dimensions
  - Tests with real physics examples
  - Further optimization or improvements to tree growth and fitting procedure (consider reoptimizing previous trees, alternate loss function, stochastic gradient descent, exploit existing widely used packages like XGBoost, etc)
  - Implementation for unweighting (modified loss function)
  - Understand if/how to best combine with multi-channelling and related techniques
- Paper and standalone implementation coming soon

Backup

Diagnostic Plots - 4D Camel Function - Integration Weights - Staged case



(a) Primary Weight

- Primary weight for integral evaluation, intermediate weight is for intermediate unweighting (primary vs secondary bdt)

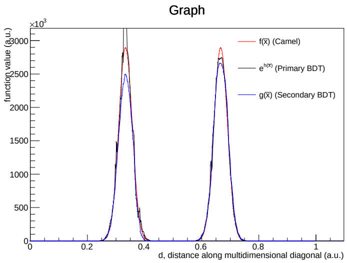
Some results - 9D Camel Function Integration

- Comparing Vegas, GBRIntegrator, Vegas for 9-dimensional camel function

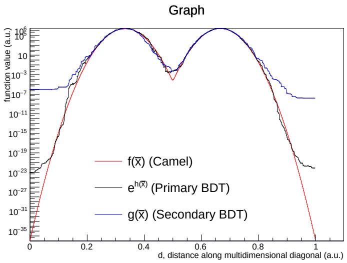
Algorithm	# of Func. Evals	$\sigma_w / < w >$	σ_I / I (2e6 add. evts)
VEGAS	1,500,000	19	$\pm 1.3 \times 10^{-2}$
GBRIntegrator	3,200,000	0.63	$\pm 4.5 \times 10^{-4}$
GBRIntegrator (staged)	3,200,000	0.31	$\pm 2.2 \times 10^{-4}$

- 50x smaller weight variance to Vegas with 2x function evaluations
- Larger performance difference between staged and non-staged variations in this case
- For this particular function VEGAS performance saturates at relatively poor weight variance

Diagnostic Plots - 9D Camel Function



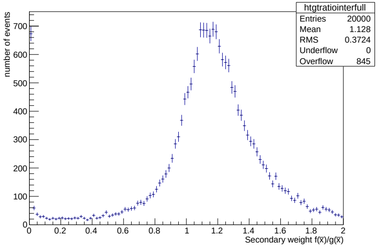
(a) linear



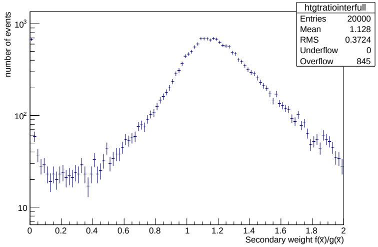
(b) log

- Deficiencies of secondary sampling BDT with respect to primary BDT are larger in higher dimensional case

Diagnostic Plots - 9D Camel Function - Integration Weights



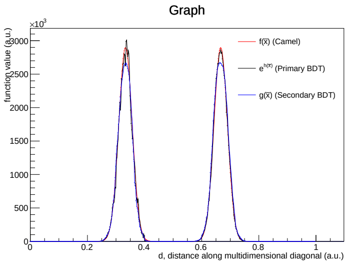
(a) linear



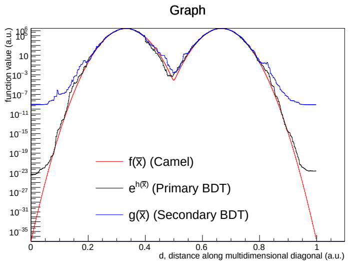
(b) log

- Visible also in the weight distribution
- (Checking staged case now)

Diagnostic Plots - 9D Camel Function - Staged Case



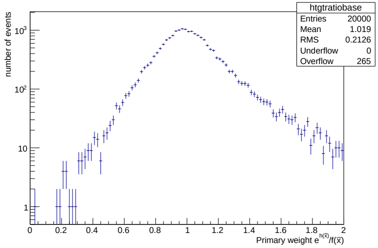
(a) linear



(b) log

- Deficiencies of secondary sampling BDT with respect to primary BDT are larger in higher dimensional case

Diagnostic Plots - 9D Camel Function - Integration Weights - Staged case



(a) Primary Weight

- Staged approach corrects obvious issues with weights in 9D case