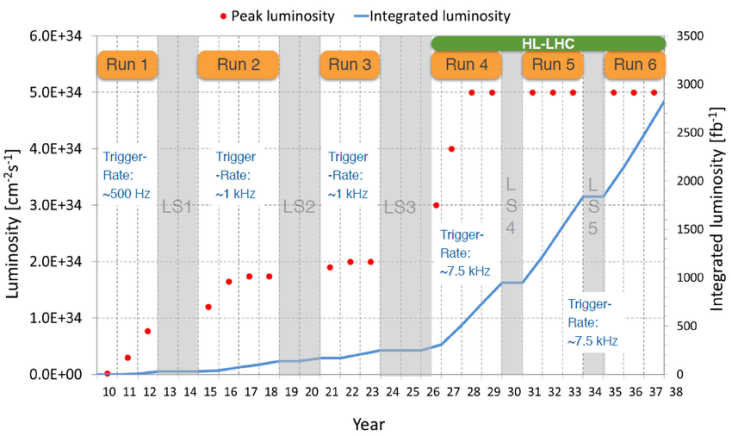
Neural Importance Sampling: Accelerating Phase Space Integrals in High-Energy Physics with Generative AI

Based on "Accelerating HEP simulations with Neural Importance Sampling"
(JHEP 03 (2024) 083)

Niklas Götz

Entering the LHC High Luminosity Era

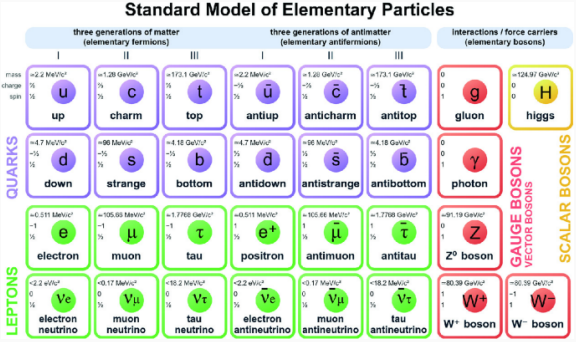


Hufnagel et al.: EPJ 214 (2019)

LHC is about to enter a phase of unprecedented luminosity (= collision rates)!

Why such high luminosity?

- The Standard Model (SM) is inherently incomplete, as it fails to explain many phenomena (gravity, dark matter, dark energy, neutrino oscillations, matter-antimatter asymmetry...)
- At the same time SM is extremely precise in predicting experimental observations as no result contradicts SM by 5 σ
- In order eliminate possible extensions of the SM, rare processes with so far high uncertainties have to be measured with higher precision, which could lead to a breaking of the 5 σ bound

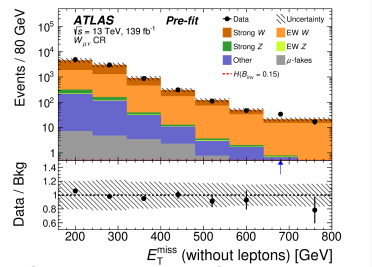


Mohamed: Spinger (2020)

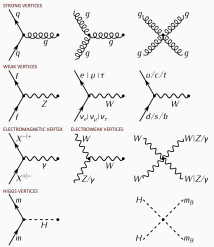
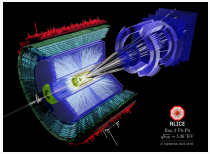
Role of theoretical predictions

In the high luminosity era, theory predcitions also need to be of increasing precision.
ATLAS: JHEP 08 (2022)

- Signals of small magnitude can be only identified when the background is precisely known
- The plurality of extensions of the SM leads to the necessity to discreminate between their predictions
- Precise theoretical predictions are necessary to guide experimentalists to observables which are most likely to show deviations from the SM



From Theory Models to Cross Sections

$\mathcal{L}$ →  → $d\sigma \sim f(\mathbf{x})|M(\mathbf{x})|^2 d\Phi$ → $\sigma \sim \int d\sigma$ → 

Lagrangian

Interactions

Matrix element

Phase Space Integration for total cross section

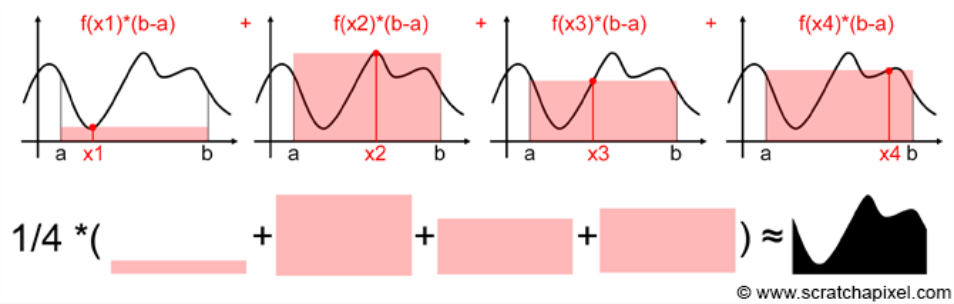
Hadronization
Comparison with Experiment
ALICE

Additionally, many detector simulations need unweighting, which requires the ability to create unbiased samples of the phase space, e.g. i.i.d samples.

Challenges of Phase Space Integration

- Integration of the matrix element is often too complex to be performed analytically, especially at higher order
- For processes with a high number of particles in the final state, the integrals become quickly high-dimensional
- Phase space cuts for regularization and peak structures create a high variance in the value of the integrand
- Parton distribution functions (PDFs) create correlations in the phase space
- Evaluation of the integrand can become very costly, around $\mathcal{O}(1s)$!

Numerical Integration



Numerical Integration

Monte Carlo integration: Integral is the average over randomly sampled integrand points:

$$\hat{I}_X^{(N)} = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \quad \xrightarrow[N\to\infty]{} I = \int_\Omega d\mathbf{x} f(\mathbf{x})$$

For uniform sampling, this is equivalent to the expectation value of the integrand.

$$I = \mathbb{E}_{X\sim U(\Omega)}(f(X)) = \mathbb{E}_{X\sim U(\Omega)}(\hat{I}_X^{(N)})$$

The variance of the estimator for the integral given by the MC method becomes then

$$\mathrm{Var}(\hat{I}_X^{(N)}) = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}(f) = \frac{\mathrm{Var}(f)}{N}$$

Numerical Integration

$\mathrm{Var}(f)$ may be determined by the unbiased sampling variance

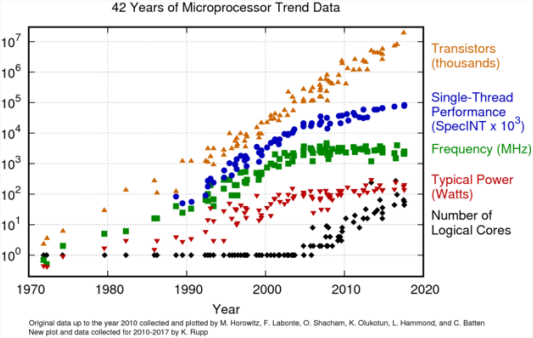
$$\mathrm{Var}(f) \approx s_N^2 = \frac{1}{N-1} \sum_{i=1}^{N} (f(\mathbf{x_i}) - I_X^{(N)})^2.$$

If the sequence converges, the uncertainty estimate of the MC approximation becomes

$$\sigma(\hat{I}_X^{(N)}) = \frac{\sigma(f)}{\sqrt{N}} = \frac{s_N}{\sqrt{N}}.$$

Error scales with $N^{-1/2}$. Other numerical methods scale even worse (e.g. $N^{-2/D}$) How would one naively increase precision of the prediction? → Naive solution: Increase number of samples!

The end of Moore's Law

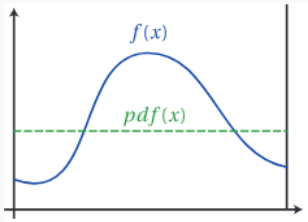


https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Free lunch is over! The increase in precision from the LHC High Luminosity Era outperforms the increase of thread speed from better CPUs.

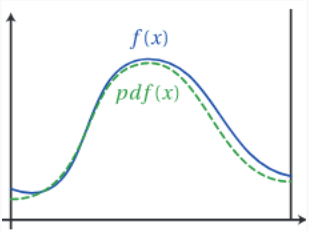
Importance sampling

Two orthogonal ways to improve precision while limiting cost: Integrate on GPU (MC integration is embarrassingly parallelizable) or sample cleverer than uniformly → Importance Sampling (Jarosz (2008))



$$I = \mathbb{E}_{X\sim U(\Omega)}(f(X))$$

Uniform Sampling



$$I = \int_\Omega p(x)dx\frac{f(x)}{p(x)} = \mathbb{E}_{X\sim p(U(\Omega))}\left(\frac{f(X)}{p(X)}\right)$$

Importance sampling

$$\lim_{p\to f}\mathrm{Var}\left(\mathbb{E}_{X\sim p(U(\Omega))}\left(\frac{f(X)}{p(X)}\right)\right) = 0$$

Importance Sampling

The problem of MC integration has been reduced to predicting the integrand for sampling from it!

Generally two starting points exist:

1. Using prior knowledge of the integrand, e.g. Multichanneling
   (Kleiss&Pittau Comput.Phys.Commun. 83 (1994))

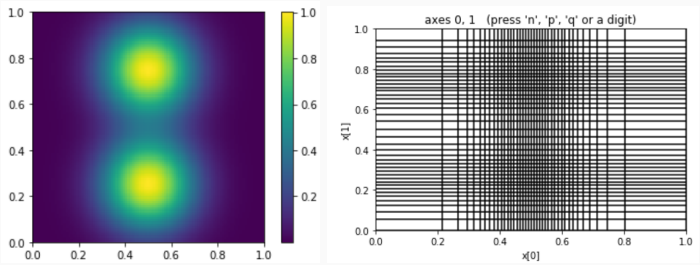
$$g(x) = \sum_i \alpha_i g_i(x), \quad \sum_i \alpha_i = 1$$

2. Using adaptive approaches without prior knowledge.

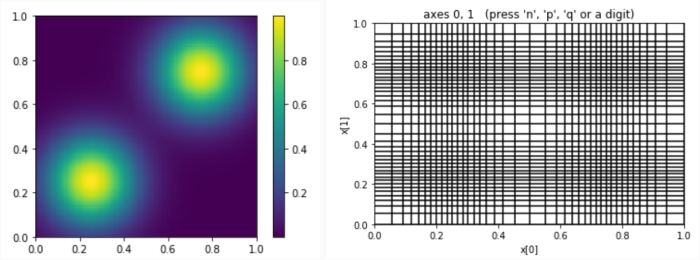
We focus on the latter, as this can perform black box integration and needs to only perform one integral.

The Benchmark: VEGAS

- Most common used importance sampling algorithm, in use since late 70s
  Lepage : JCP 27 (1978)
- Fast, modern extensions exist Sakitotis et al.: 2202.01753
- Factorises along each dimension $p(\mathbf{x}) = p(x_1)p(x_2)...p(x_n)$
- Approximates the integrand by a step function with M steps with likelihood of $1/M$ to draw a sample, and learns stepwidth



The issue with VEGAS



Factorization limits the performance of VEGAS! This is especially relevant when correlations exists, for example due to PDFs. VEGAS works only well if dimensions are independent. Additionally, many peaks in high dimensions also reduce convergence speed.

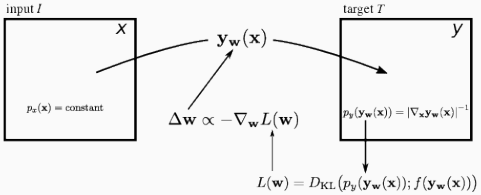
Improvements of VEGAS

- **VAMP:** (Ohl: CPC 120 (1999)) approximates integrand not only with product, but also sum of functions, implementing the mulitchannel approach
- **BASES:** (Kawabata, CPC 88(1995)) treats some of the dimensions as "wild" with strong variations and other as less important
- **PARNI:** (van Hameren, APP B40 (2009)) Local multichannel approach, together with sub-regions where the density is constant
- **VEGAS+:** (Lepage, JCC 430 (2021)) Use stratified sampling, which partitions the integration volume into regions of similiar values. Reduces effects of having structures and multiple peaks. Used as benchmark in the following.

Although each of these represent an improvement, they cannot overcome the core limitations of this approach

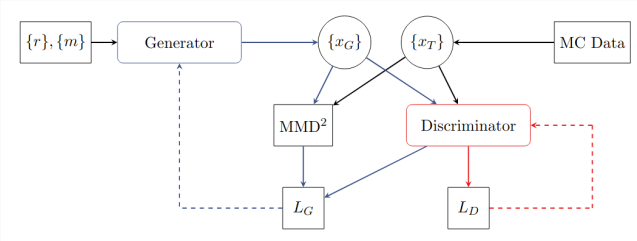
Neural Networks (NN) for the rescue?

- **Advantage**: NN are universal function approximators. One could learn the integrand directly and sample from it. (1707.00028, Klimek et al.: SciPost P. 9 (2020), Chen et al.: SciPost P. 10 (2021))
- The NN learns the whole event generation
- Input random numbers, output $3N - 4$ variables in [0,1) which are mapped into the phase space
- minimise KL divergence



Neural Networks (NN) for the rescue?

- Other approach: use generative adversarial networks
- Two NN compete: one produces samples, one tries to determine if they are from the training set or generated
- Works well for event generation (Butter et al.: SPP 7 (2019))and unweighting (Backes et al.: SPP 7 (2021))
- Struggles with regions poorly populated in the phase space



Neural Networks (NN) for the rescue?

Problem: We still have to sample from the learned function!

This is very problematic!

For sampling from the learned distribution, we need the Jacobian, which also has to be invertible. In general, this is not guaranteed for a NN and is very costly and unstable. The model might ignore tails of the phase space which makes unweighting difficult.

Reason

If $T(u) = x \sim q(x)$ and $u \sim p(u)$, with T the transformation of the probabilty distribution, then:

$$q\left(x = T(u)\right) = p(u)\,|J_T(\,u)|^{-1},$$

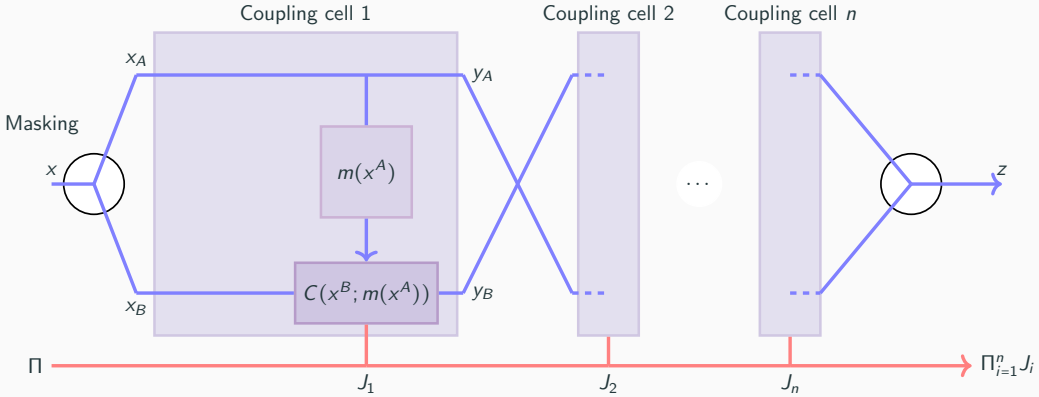
where J_T is the Jacobian determinant of T:

$$J_T(u) = \det \frac{\partial T_i}{\partial u_j}(u).$$

Normalizing Flows (NF)

- Idea: Instead of learning the integrand directly, we learn the parameters of a bijective mapping! (Bothmann et al.: SciPost P. 8 (2020), 2001.05486, Gao et al.: PRD 101 (2020), Winterhalder et al.: SciPost P. 12 (2022)).
- Normalizing flows are such a parametric diffeomorphism (Papamakarios et al.: JMLR 22 (2021))
- $T(\cdot, \theta)$ has a tractable Jacobian which can be calculated without backpropagation, which is composable and where θ is optimized during training.
- We realise NF with coupling cells. (1410.8516, 1605.08803,1808.03856)
- Other than VEGAS, we do not suffer from grids or binning.
- We learn the correct change of variable, just like VEGAS, but better!

Coupling Cells



Schematic structure of a coupling layer. m is a NN, which determines the parameters of separable, invertible functions $C(x^B, \theta)$. The masking has to performed in such a way that every coordinate influences the transformation of all other coordinates.

Coupling Cells

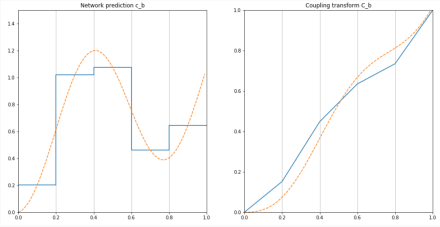
Due to the masking, the determinant of the Jacobian of each cell becomes especially simple to calculate:

$$\left(\begin{array}{ccc|ccc} 1 & & 0 & & & \\ & \ddots & & & \mathbf{0} & \\ 0 & & 1 & & & \\ \hline & & & \frac{\partial C^{b_1}(\boldsymbol{\theta}_{b_1}, x_{b_1})}{\partial x_{b_1}} & & 0 \\ \frac{\partial C^B(\boldsymbol{\theta}^B, \mathbf{x}^B)}{\partial x^A} & & & & \ddots & \\ & & & 0 & & \frac{\partial C^{b_B}(\boldsymbol{\theta}_{b_B}, x_{b_B})}{\partial x_{b_B}} \end{array}\right)$$

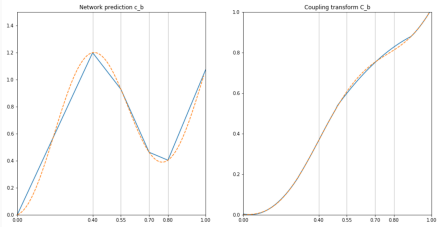
Coupling Transform

Commonly, there are three different forms for the coupling transform:

- affine: $\mathbf{y}^B = C^B(\mathbf{x}^B, \boldsymbol{\theta}^B) = \mathbf{x}^B \odot e^{\mathbf{s}^B} + \mathbf{t}^B, \quad \boldsymbol{\theta}^B = (\mathbf{t}^B, \mathbf{s}^B) \in \mathbb{R}^{2,|B|}$
- piecewise-linear $\mathbf{y}^B = C^B(\mathbf{x}^B, \boldsymbol{\theta}^B)$, with each component $C_b(x_b, \mathbf{Q}) = \int_0^{x_b} c_b(t)dt = \alpha Q_{bk} + \sum_{s=1}^{k-1} Q_{bs}$, which is basically a step function
- piecewise-quadratic $\mathbf{y}^B = C^B(\mathbf{x}^B, \boldsymbol{\theta}^B)$, with each component $C_b(x_b, \mathbf{W}, \mathbf{V}) = \frac{\alpha^2}{2}(V_{bz+1} - V_{bz})W_{bz} + \alpha V_{bz} W_{bz} + \sum_{k=1}^{z-1} \frac{V_{bk} - V_{bk+1}}{2} W_{bk}$, which is a step function with varying bin width → needs less bins to achieve good resolution



Piecewise-linear approximation



Piecewise-quadratic approximation

Training Strategy

Want to minimise: $$\sigma^2(f,p) = \mathop{\mathbb{E}}_{x\sim p}\left(\left(\frac{f(x)}{p(x)}\right)^2\right) - I^2.$$

Forward variance training:

$$\mathcal{L}_{\text{forward}}(\theta) = \frac{1}{N}\sum \frac{f^2(x)}{p^2(x,\theta)}.$$

Two sources of dependence on p: p itself and sampled points - strong dependence on initialisation, constant resampling needed. Good for late stages of training, when p is a good estimator.

Training Strategy

Backward variance training

Let us consider a second PDF q:

$$\mathcal{L}_{\text{backward}}(\theta) = \frac{1}{N} \sum \frac{f(x_i)^2}{q(x_i) p(x_i, \theta)}$$

Sample is now independent of p, so we can ensure good coverage (for example, with choosing q uniform), or run multiple gradient steps with the same batch of points by a frozen copy of the NF (similiar to buffered training in MadNIS Heimel et al.: SciPost P. 15 (2023))

Training Strategy

Adaptive Training

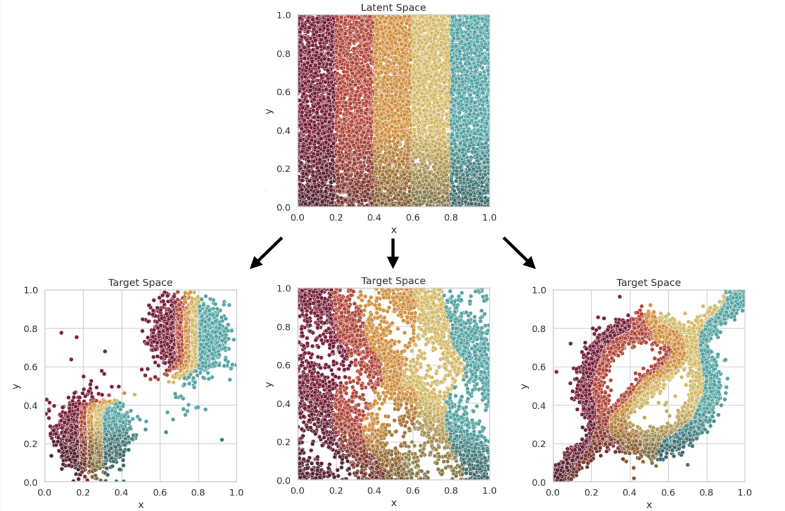
- Uniform q optimises coverage in the beginning, and immunizes against random initial model
- Frozen NF as q optimises θ better
- Switch between both approaches once loss with frozen NF is smaller than with uniform sampling
- Guarantees training on whole phase space, improves stability and convergence

Kullback-Leibler-Distance training:
Alternatively, one could also maximise similarity between distributions:

$$D_{KL}(q|p) = \int dx q(x) \log \frac{q(x)}{p(x)} = \int dx \frac{f(x)}{I} \log \frac{f(x)}{I} - \frac{f(x)}{I} \log q(x) \propto - \int dx f(x) \log q(x)$$

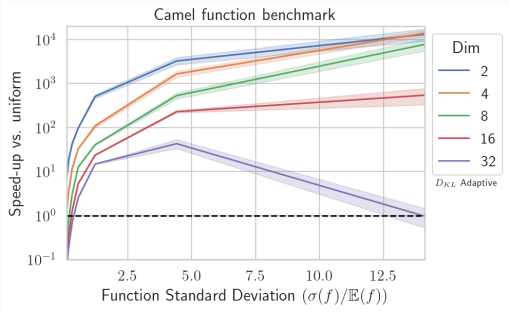
This however trains also regions where the integrand is small and less relevant

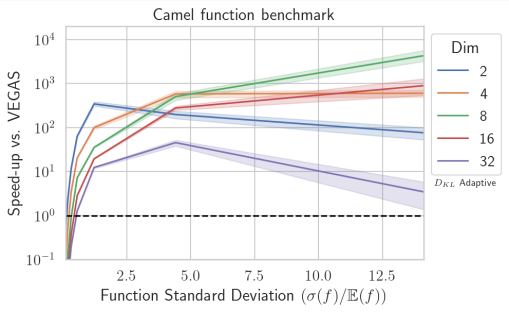
Simple Examples in 2D with ZüNIS



Trained Mapping between latent space and target distribution for 2D functions

Simple Examples in higher dimensions with ZüNIS

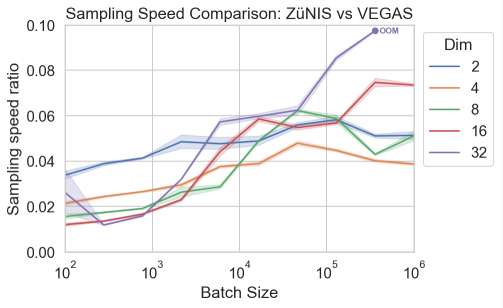


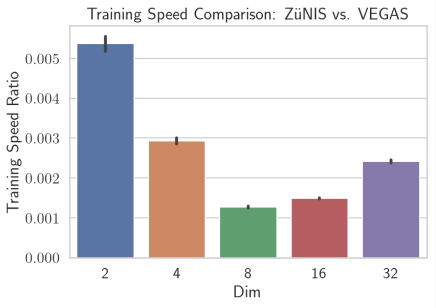


Without any hyperparameter tuning, NIS can reduce the number of needed sample points by orders of magnitude in comparison to uniform sampling and VEGAS. High-variance high-dimension situations need tuning.

Training on 1M points.

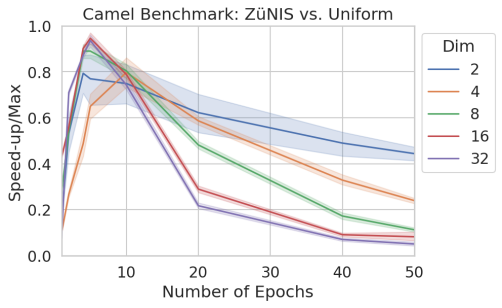
Simple Examples in higher dimensions

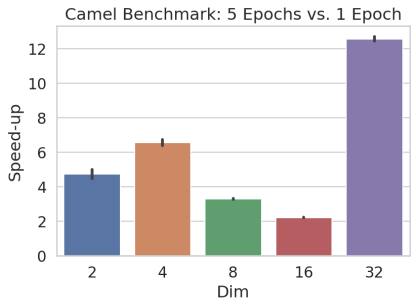




Even though boosted by GPU usage, sampling and training are considerably slowed down using NIS in comparison to VEGAS. This effect is negligble if integrands are expensive to evaluate.

Simple Examples in higher dimensions



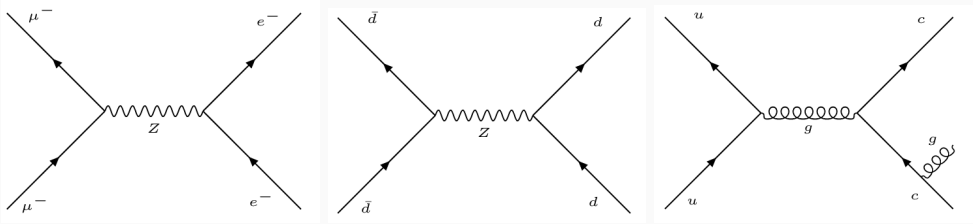


Keeping the model frozen for 5-10 epochs greatly improves performance!

Towards Cross Section Integrals: Phase Space Sampling

- For evaluating differential cross sections, arguments need to be valid phase space points
- Phase space sampling should be done on GPU to minimise data transfer cost and maximise parallelization
- Number of sampled random numbers has to be kept minimal → ideally same as degrees of freedom
- Use "RAMBO on diet" algorithm (1308.2922) in the PyTorch library TorchPS (10.5281/zenodo.4639109)
- Inverse is straightforward
- Creates 1.2M massive 4-particle final states per second on GeForce RTX 2080

Integration of MadGraph cross sections

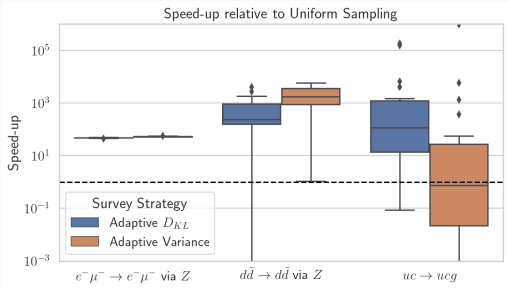


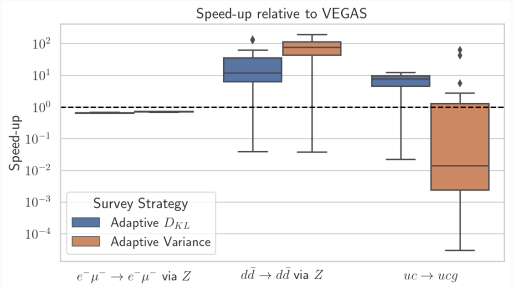
MadGraph is a software tool to calculate (differential) cross sections Alwall et al.: JHEP 07 (2014)
Three benchmark examples:

- trivial, uncorrelated and peak-free process $e^+e^- \to \mu^+\mu^-$
- $d\bar{d} \to d\bar{d}$ with correlations due to PDFs
- $uc \to ucg$ three jet cross section for higher-dimensional integration

All processes with cuts which lead to discontinuities!

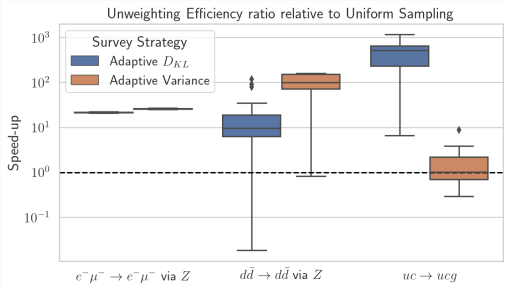
Integration of MadGraph cross sections - speed-up

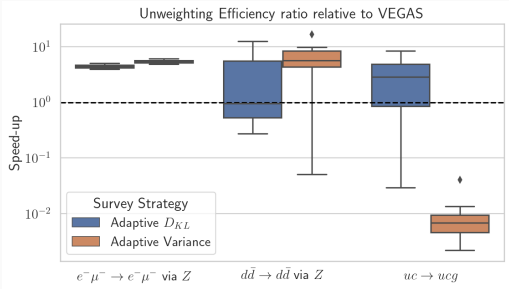




- Typical challenge of cross section integration: high variance of training outcome, different choice of loss function favourable. Training on 500K evaluations
- Improvements increase for complex integrands
- VEGAS performs well for *uc* → *ucg* due to phase space sampling aligning enhancements and reducing correlations

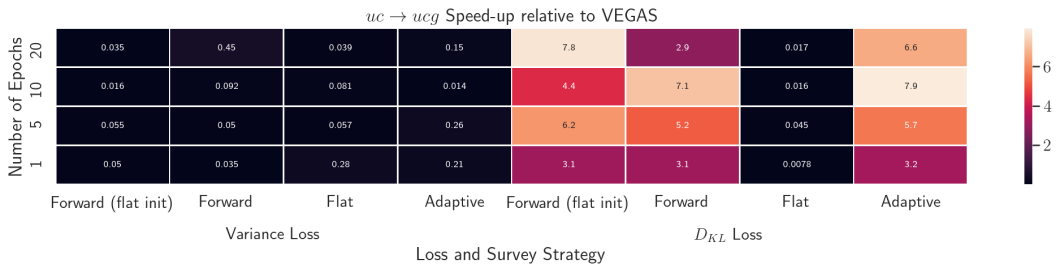
Integration of MadGraph cross sections - Unweighting Efficiency





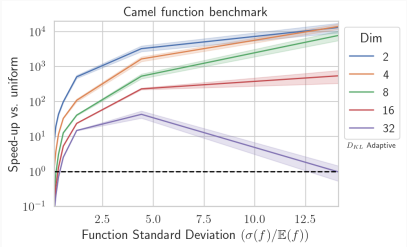
- Unweighting efficiency: how much data of an original sample is kept when employing a veto algorithm during sample generation
- For simple integrals, unweighting benefits from NIS
- Similar trends as for speed-up

Impact of adaptive training



Using adaptive training reaches maximum training performance faster and more stable due to better coverage of integration space!

Limits of the black box approach



unweighting efficiency $\langle w\rangle/w_{\max}$		LO QCD $n=0$	$n=1$	$n=2$	$n=3$	$n=4$	NLO QCD (RS) $n=0$	$n=1$
$W^+ + n$ jets	Sherpa	$2.8 \cdot 10^{-1}$	$3.8 \cdot 10^{-2}$	$7.5 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	$8.3 \cdot 10^{-4}$	$9.5 \cdot 10^{-2}$	$4.5 \cdot 10^{-3}$
	NN+NF	$6.1 \cdot 10^{-1}$	$1.2 \cdot 10^{-1}$	$1.0 \cdot 10^{-2}$	$1.8 \cdot 10^{-3}$	$8.9 \cdot 10^{-4}$	$1.6 \cdot 10^{-1}$	$4.1 \cdot 10^{-3}$
	Gain	2.2	3.3	1.4	1.2	1.1	1.6	0.91
$W^- + n$ jets	Sherpa	$2.9 \cdot 10^{-1}$	$4.0 \cdot 10^{-2}$	$7.7 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$	$9.7 \cdot 10^{-4}$	$1.0 \cdot 10^{-1}$	$4.5 \cdot 10^{-3}$
	NN+NF	$7.0 \cdot 10^{-1}$	$1.5 \cdot 10^{-1}$	$1.1 \cdot 10^{-2}$	$2.2 \cdot 10^{-3}$	$7.9 \cdot 10^{-4}$	$1.5 \cdot 10^{-1}$	$4.2 \cdot 10^{-3}$
	Gain	2.4	3.3	1.4	1.1	0.82	1.5	0.91
$Z + n$ jets	Sherpa	$3.1 \cdot 10^{-1}$	$3.6 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$4.7 \cdot 10^{-3}$		$1.2 \cdot 10^{-1}$	$5.3 \cdot 10^{-3}$
	NN+NF	$3.8 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$	$1.4 \cdot 10^{-2}$	$2.4 \cdot 10^{-3}$		$1.8 \cdot 10^{-3}$	$5.7 \cdot 10^{-3}$
	Gain	1.2	2.9	0.91	0.51		1.5	1.1

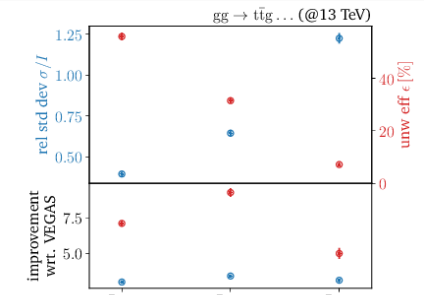
i-flow: Gao et al.: PRD 101 (2020)

- We observed a decay of improvements in high dimensional spaces
- If most of the phase space does not contribute to the integrand, black box approaches are likely to fail
- Similar approaches observed issues at high-multiplicity events
- Here prior knowledge is necessary in order to stabilise training

Limits of the black box approach

Other refinements Heimel et al.: SPP 15 (2023) and 2311.01548

- MadNIS combines Neural Importance Sampling with multichanneling
- It uses initialization with classical importance sampling algorithms to speed up and stabilise training
- However, high-dimensional multi-jet cases stay challenging even for the improved MadNIS approach



The ZüNIS library

- Fully open-source PyTorch-based NIS tool which can be trivially run on GPU
- Wide range of automatic benchmarks included
- Extensivly documented, aimed at being used by non-experts by providing reasonable defaults
- Installation is as easy as pip install zunis
- Incorporates all presented strategies and refinement of NIS

Basic example:

```python
import torch
from zunis.integration import Integrator

device = torch.device("cuda")

d = 2

def f(x):
    return x[:,0]**2 + x[:,1]**2

integrator = Integrator(d=d,f=f,device=device)
result, uncertainty, history = integrator.integrate()
```

Survey and refine stages

```
# Setting values at instantiation time
integrator = Integrator(d=d, f=f, n_iter_survey=3, n_iter_refine=5)
# Override at integration time
integral_data = integrator.integrate(n_survey=10, n_refine=10)
```

- Start by performing a survey phase, in which it optimizes the way it samples points
- In the refine phase, the integral is calculated using the learned sampler
- Steps are useful to observe convergence and for resampling point batches
- Samples of survey phase are by default not used for integration

Customising coupling cells

To implement a new invertible coupling cell, provide an `InvertibleTransform` object, which provides both forward and inverse mapping as well as Jacobian calculation. For example, consider a very simple linear coupling cell over $\mathbb{R}^d$

$$y = Q(x) : \begin{cases} y^A = x^A \\ y^B = \exp\big(T(x^A)\big) \times x^B, \end{cases}$$

```
import torch
from zunis.models.flows.sampling import FactorizedGaussianSampler
from zunis.training.weighted_dataset.stateful_trainer import StatefulTrainer
from zunis.models.flows.coupling_cells.general_coupling import InvertibleCouplingCell
from zunis.models.flows.coupling_cells.transforms import InvertibleTransform
from zunis.models.layers.trainable import ArbitraryShapeRectangularDNN

class LinearTransform(InvertibleTransform):
  def forward(self,x,T):
      alpha = torch.exp(T)
      logj = T*x.shape[-1]
      return x*alpha, logj.squeeze()
  def backward(self,x,T):
      alpha = torch.exp(-T)
      logj = -T*x.shape[-1]
      return x*alpha, logj.squeeze()
```

Customising coupling cells

Now we can construct the coupling cell training the transformation parameters, and specify the architecture of the NN.

```
class LinearCouplingCell(InvertibleCouplingCell):
  def __init__(self, d, mask, nn_width, nn_depth):
      transform = LinearTransform()
      super(LinearCouplingCell, self).__init__(d=d, mask=mask,transform=transform)
      d_in = sum(mask)
      self.T = ArbitraryShapeRectangularDNN(d_in=d_in,out_shape=(1,),d_hidden=nn_width,n_hidden=nn_depth)
      self.inverse=False
```

At this point, we are ready to go!

```
d = 2
device = torch.device("cpu")

mask=[True,False]
nn_width=8
nn_depth=256

sampler=FactorizedGaussianSampler(d=d)
linear_coupling=LinearCouplingCell(d,mask,nn_width,nn_depth)
trainer = StatefulTrainer(d=d, loss="variance", flow_prior=sampler,flow=linear_coupling, device=device)
```

Conclusion

- NIS beats VEGAS for most cases both for convergence rate and unweighting efficiency
- Due to high cost of training in comparison to VEGAS, it fills the gap for high-cost integrands with correlations
- Careful choice of survey strategy is necessary to unleash full potential of NIS
- With ZüNIS, a NIS library is available both trivial to use in an arbitrary context and flexible enough to modify at wish, suitable to explore this method and expand it
- With MadNIS, a full NIS-powered event-generator is in work to be integrated in MG5AMC, including many additional features