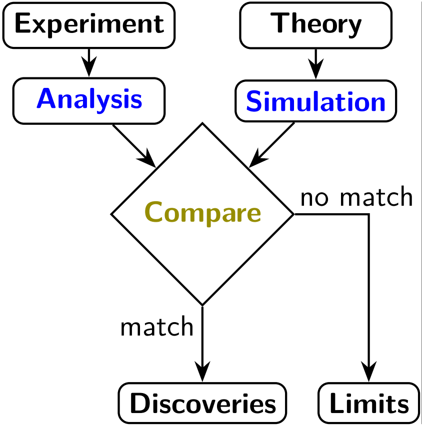
Classifying importance regions in Monte Carlo simulations with machine learning

Raymundo Ramos
Korea Institute for Advanced Study
(based on work with: M. Park (SEOULTECH) and K. Ban (KIAS))

ML4Jets2024
Paris, France, November 5, 2024

From theory to discovery (or limits)

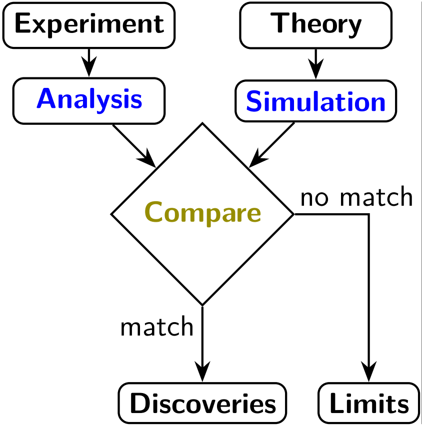


More diverse and **more precise** experimental results.

BSM physics may be hiding in ever shrinking error bars.

Simulations have to keep up with the **complexity** of experiments and provide **accurate** predictions.

From theory to discovery (or limits)



More diverse and **more precise** experimental results.

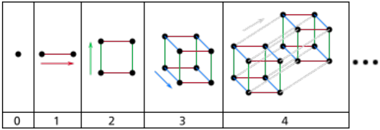
BSM physics may be hiding in ever shrinking error bars.

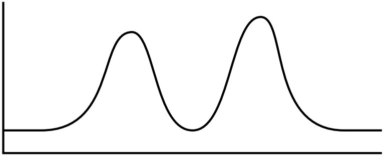
Simulations have to keep up with the **complexity** of experiments and provide **accurate** predictions.

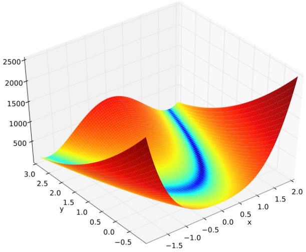
We need improved techniques for data analysis

Complications along the way

- Several dimensions
- Multimodality
- Curved degeneracy
- ...







Monte Carlo: brief review

$f(x)$: Output of a comprehensive calculation with d-dimensional input x

- May become time consuming
- Likely to require lots of computational resources

To extract answers: Interpret $f(x)$ in relation to a probability density and use Monte Carlo simulations.

- **Monte Carlo (MC) integration in space Φ**

$$I[f] = \int_\Phi dx\, f(x) = V_\Phi \langle f \rangle_\Phi, \quad \text{with} \quad V_\Phi = \int_\Phi dx\,,$$

MC estimate (N events): $E(I) = V_\Phi E(\langle f \rangle_\Phi)$, $E(\langle f \rangle_\Phi) = \frac{1}{N} \sum_n^N f(x_n)$

Variance: $\sigma^2(E(I)) = V_\Phi^2 \sigma_\Phi^2(f(x))/N$

Monte Carlo: brief review

- **Variance reduction: stratified sampling**

Reduce variance by partitioning the space:

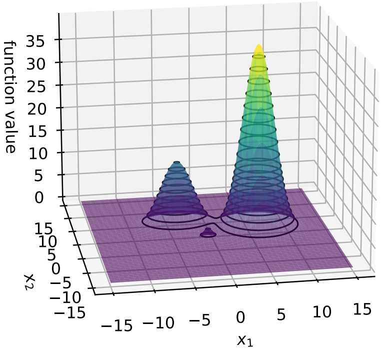
$$\Phi = \sum_j \Phi_j, \quad V_\Phi = \sum_j V_{\Phi_j}$$

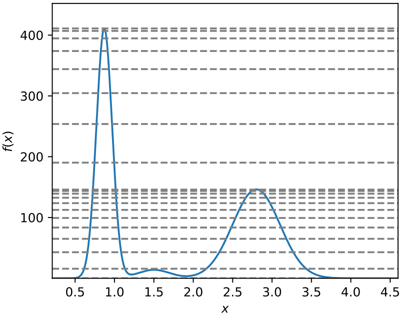
Usually, volumes of partitions are known and

$$E(I) = \sum_j V_{\Phi_j} E(\langle f \rangle_{\Phi_j}), \quad \sigma^2(E(I)) = \sum_j V^2_{\Phi_j} \sigma^2_{\Phi_j}(f(x))/N_j$$

Oversampling needed only in partitions with large variance

Remixing stratified sampling, Lebesgue style





$$\Phi_j = \left\{ x \mid l_j < f(x) \leq l_{j+1} \right\} \quad \rightarrow \quad E(I) = \sum_j E(V_{\Phi_j}) E(\langle f \rangle_{\Phi_j})$$

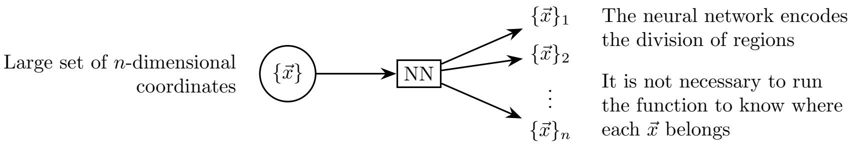
V_{Φ_j} and $\langle f \rangle_{\Phi_j}$ are independent:

$$\sigma^2(E(I)) = E^2(V_{\Phi_j})\sigma^2_{\Phi}(\langle f \rangle_{\Phi_j}) + E^2(\langle f \rangle_{\Phi_j})\sigma^2_{\Phi}(V_{\Phi_j}) + \sigma^2_{\Phi}(\langle f \rangle_{\Phi_j})\sigma^2_{\Phi}(V_{\Phi_j})$$

Remixing stratified sampling with a neural network

- Neural networks (NN) as generic function approximators
- Useful when training a NN **could be more efficient** than passing every single point through a heavy calculation

Main idea: train the NN to classify points according to contours



Evaluations of the neural network $\rightarrow$ determine $E(V_{\Phi_j})$, reduce $\sigma^2_{\Phi}(V_{\Phi_j})$

Remixing stratified sampling with a neural network

- $E^2(V_{\Phi_j})\sigma^2_{\Phi}(\langle f\rangle_{\Phi_j})$

$\sigma^2_{\Phi}(\langle f\rangle_{\Phi_j})$: **reduced** by partitioning, **limited** by contours of $f(x)$.
Only part where the number of evaluations of $f(x)$ is important
Inaccurate network increases variance.

- $E^2(\langle f\rangle_{\Phi_j})\sigma^2_{\Phi}(V_{\Phi_j})$

$\sigma^2_{\Phi}(V_{\Phi_j})$ reduced by evaluations of neural network.
Many techniques can be used to reduce this.
Evaluating the network is fast.

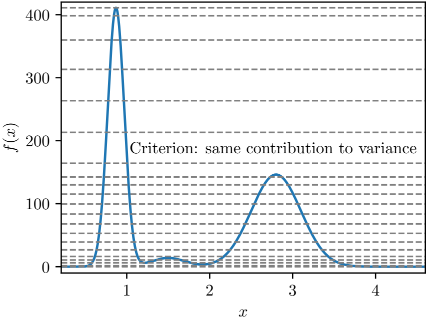
- $\sigma^2_{\Phi}(\langle f\rangle_{\Phi_j})\sigma^2_{\Phi}(V_{\Phi_j})$

This shrinks faster than the other two

Remixing stratified sampling with a neural network

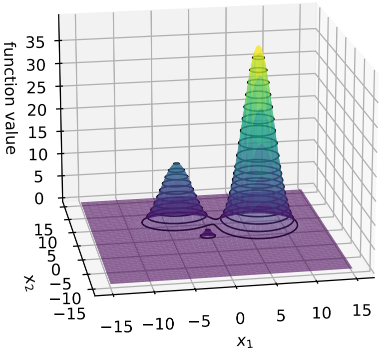
Next question: How to divide the range of $f(x)$?

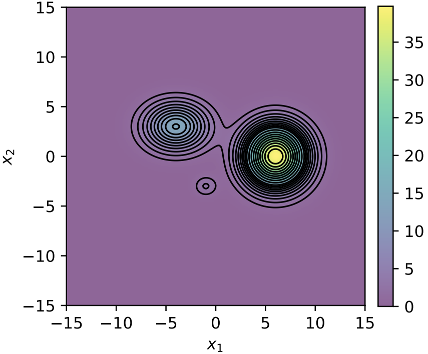
- **Infinite possibilities**
- a few simple examples, choose limits on $f(\vec{x})$ such that:
  - ➜ Φ_j with similar lengths V_{Φ_j}
  - ▼ Φ_j with similar $E^2(V_{\Phi_j})\sigma^2(f(x \in \Phi_j))$



Learn divisions of a function with multiple peaks

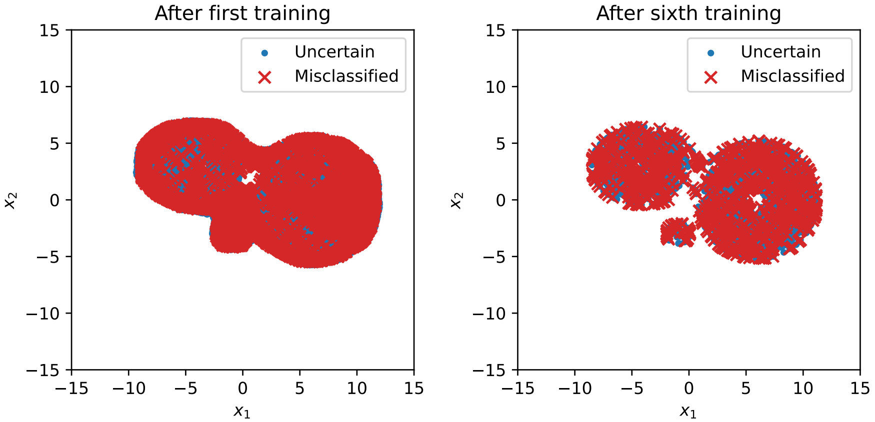
20 regions: Regions can have up to three subregions.





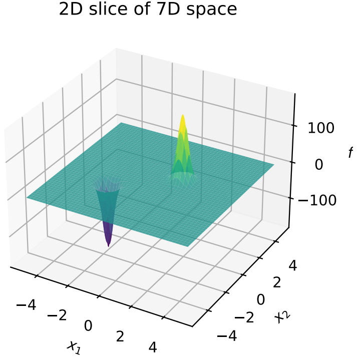
Learn divisions of a function with multiple peaks

20 regions with similar contribution to value of integral



Step: (0.1) train → (0.2) predict → (0.3) add uncertain and errors → (1.1) train ... [See Hammad, Park, RR, Saha, arXiv:2207.09959]

Toy example: 7D function with large cancellation



$$x \in [-5, 5]^7$$

$$f(x) = 100[f_+(x) - f_-(x)] + 0.1 f_{\text{bg}}(x)$$

$$\int f(x)dx = \int 0.1 f_{\text{bg}}(x) \approx 0.1$$

Multilayer perceptron: 2 hidden layers, 7D input

- Hidden: nodes $2 \times n_{\text{reg}} \times 7$, $n_{\text{reg}} \times 7$, activation ReLU
- Output: $n_{\text{reg}} - 1$, activation: *tanh*

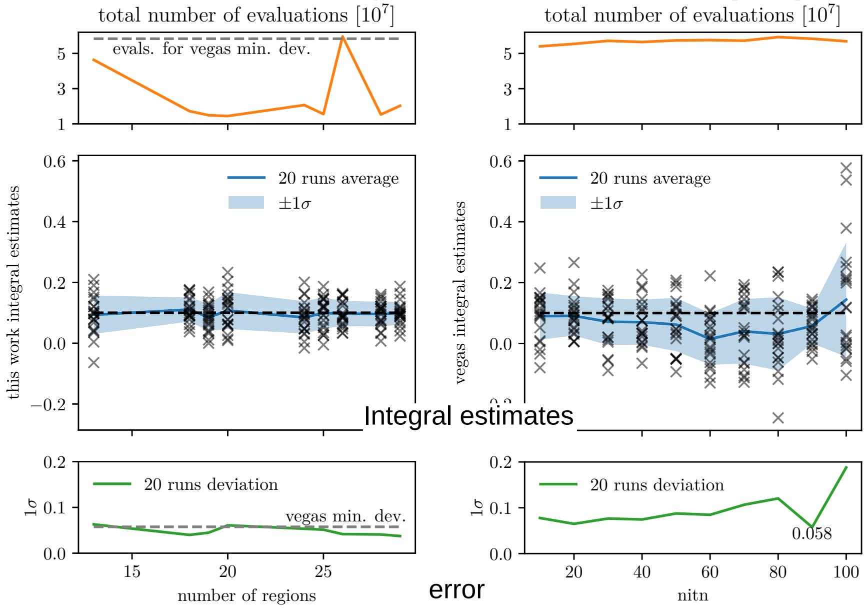
Toy example: 7D function with large cancellation

Compare with vegas: Using `python` vegas module [G. P. Lepage, arxiv:2009.05112]

Simple approach:

- Target an estimated value of total evaluations of same order.
- Vegas: Distribute total evaluations among different iterations.
- Use 20 runs to calculate average and error.
- Our total number of evaluations include points used for training.
- NOTE: expect Vegas+ to be **much faster** for fast $f(x)$.

total number of points evaluated with function [10^7]

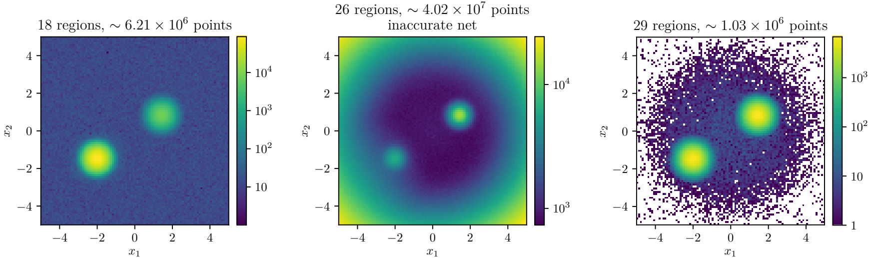


Our result

vegas

Toy example: 7D function with large cancellation

2D distribution of points for different amount of regions



18 regions
6.21×10^6 points

26 regions
4.02×10^7 points
inaccurate net

29 regions
1.03×10^6 points

Event generation: Quark pair to electron + positron

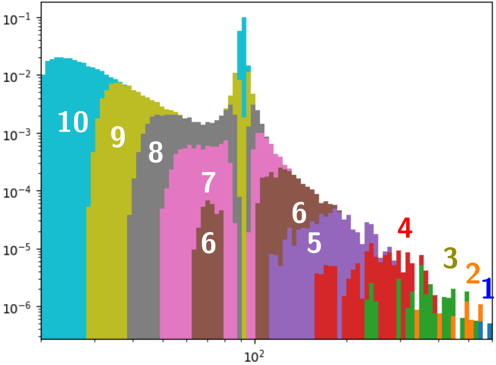
Very simple example:

$$u\bar{u} \rightarrow e^- e^+$$

- `ROOT` - `TGenPhaseSpace`: phase space generator.
- `Madgraph` (standalone mode): matrix element.
- `NNPDF23`: parton density function.
- cuts: leptons: $p_T > 10$ GeV, $|\eta| < 2.5$

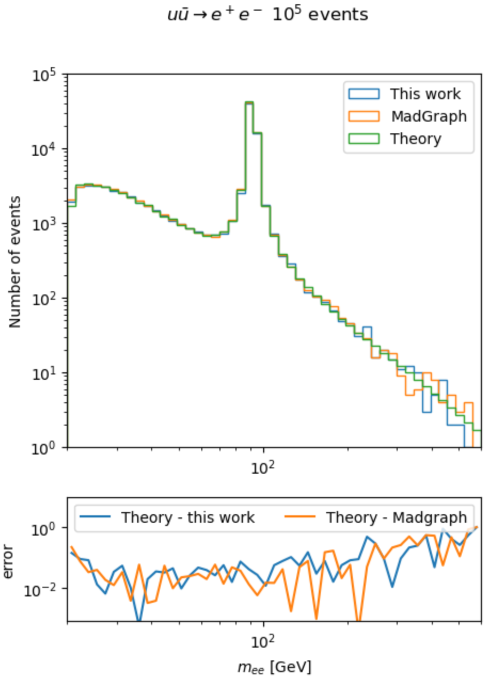
Generate events: 10 usable regions

$e^{-}e^{+}$ invariant mass projection



m_{ee} [GeV]

- Sample each region until enough events are accumulated. **NN can tell which regions points belong to**.
- Select points using correct result.



- 10^5 unweighted events
- High m_{ee} error expected from thinning of sample.
- Invariant mass around Z resonance is similar when comparing to MadGraph
- Efficiency of selection of unweighted events increases with more regions. But more regions requires more points for training

Summary

- Monte Carlo simulations could be challenging due to
  - $$ Time consuming costly operations
  - ❃ Complicated characteristics of the problem
- Machine learning can improve the situation, but many options exist.

➔ We presented a process to accelerate sampling of points for slow functions in a parameter space using a neural network.

➔ The main idea is to **separate** regions according to importance.

- Concentrate on high importance regions
- Reduce work in regions that contribute less to results

➔ Division process based and applied only on value of $f(x)$.

➔ Considerable bike-shedding left out of this talk

Thanks for listening!