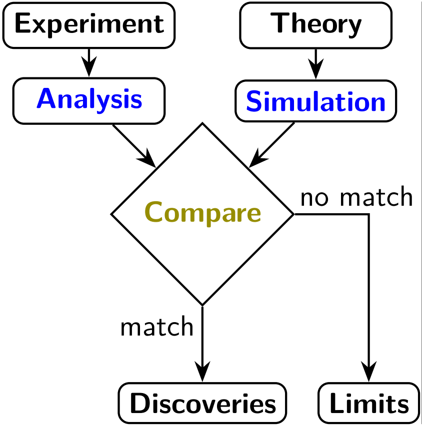
Classifying importance regions in Monte Carlo simulations with machine learning

Raymundo Ramos
Quantum Universe Center, KIAS
(based on work with: M. Park (SEOULTECH) and K. Ban (KIAS))

International Workshop on Future Linear Colliders
Tokyo, Japan, July 10, 2024

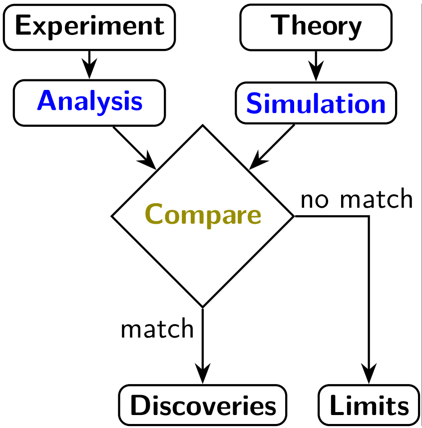
From theory to discovery (or limits)



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

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

From theory to discovery (or limits)

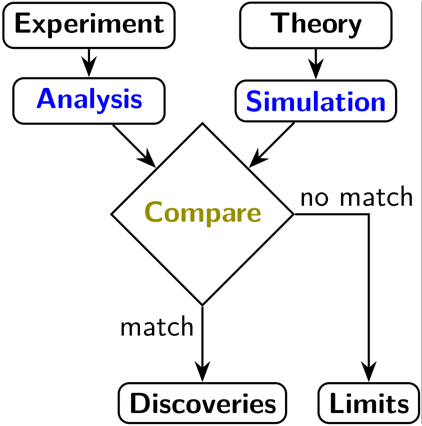


More diverse and **more precise** experimental results.

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

We need more powerful and expensive computers!

From theory to discovery (or limits)



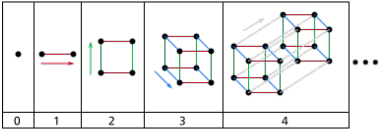
More diverse and **more precise** experimental results.

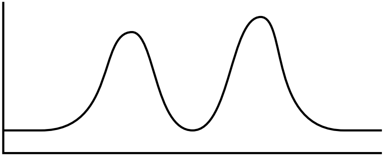
Simulations have to keep up with the **complexity** of experiments and provide **accurate** predictions.

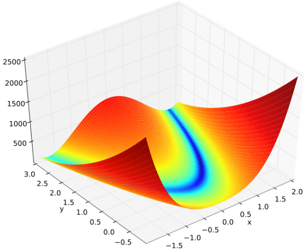
We need ~~more powerful and expensive computers!~~ **improved techniques for data analysis!**

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

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

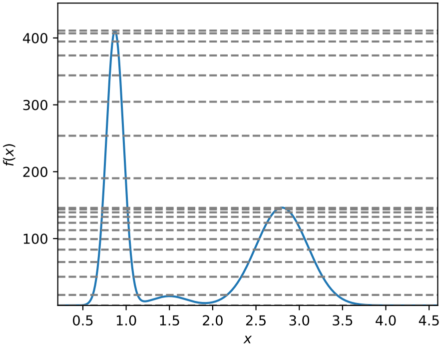
Oversampling needed only in partitions with large variance

Remixing stratified sampling

Divide Φ according to contours of $f(x)$ (**Lebesgue integration**):

$$\Phi_j = \{x \mid l_j < f(x) \leq l_{j+1}\}.$$

Now V_{Φ_j} depend on the contours l_j and l_{j+1} and, in general, are subject to estimation



$$E(I) = \sum_j E(V_{\Phi_j}) E(\langle f \rangle_{\Phi_j})$$

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

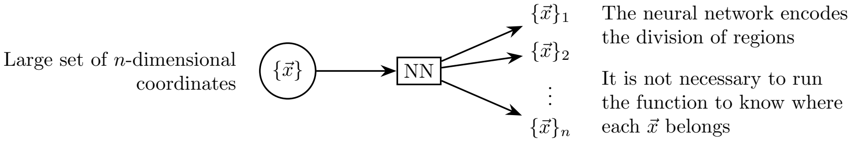
$$\sigma^2(E(I)) = E^2(V_{\Phi_j})\sigma^2_\Phi(\langle f \rangle_{\Phi_j}) + E^2(\langle f \rangle_{\Phi_j})\sigma^2_\Phi(V_{\Phi_j}) + \sigma^2_\Phi(\langle f \rangle_{\Phi_j})\sigma^2_\Phi(V_{\Phi_j})$$

(this is starting to look ugly)

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
<u>Inaccuracy to predict contours by NN can increase variance.</u>

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

$\sigma^2_{\Phi}(V_{\Phi_j})$ reduced by evaluations of neural network.

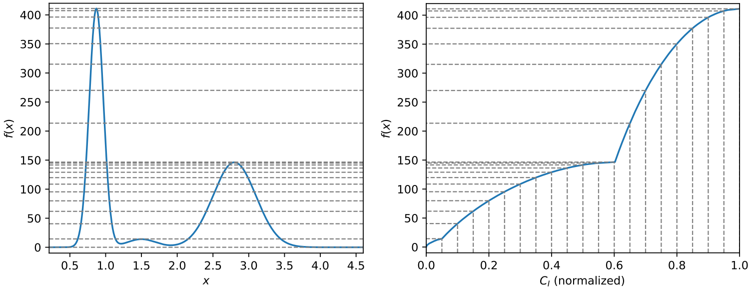
- $\sigma^2_{\Phi}(\langle f \rangle_{\Phi_j})\sigma^2_{\Phi}(V_{\Phi_j})$

Clearly the least important, reduced by reducing the other two.

Remixing stratified sampling with a neural network

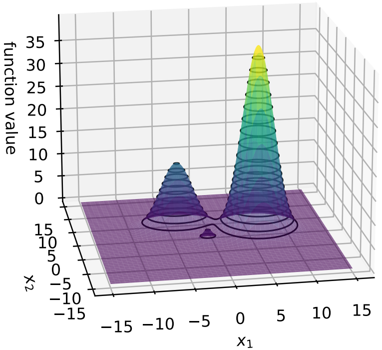
Next question: How to divide the range of $f(x)$?

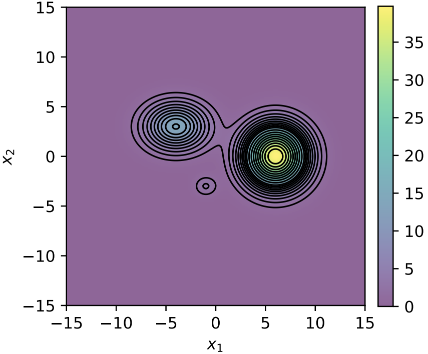
- **Infinite possibilities**
- a few simple examples, choose limits on $f(\vec{x})$ such that:
  - ➜ Φ_j with similar lengths V_{Φ_j}
  - ▼ Φ_j with similar contributions to $I_\Phi[f(x)]$



Learn divisions of a function with multiple peaks

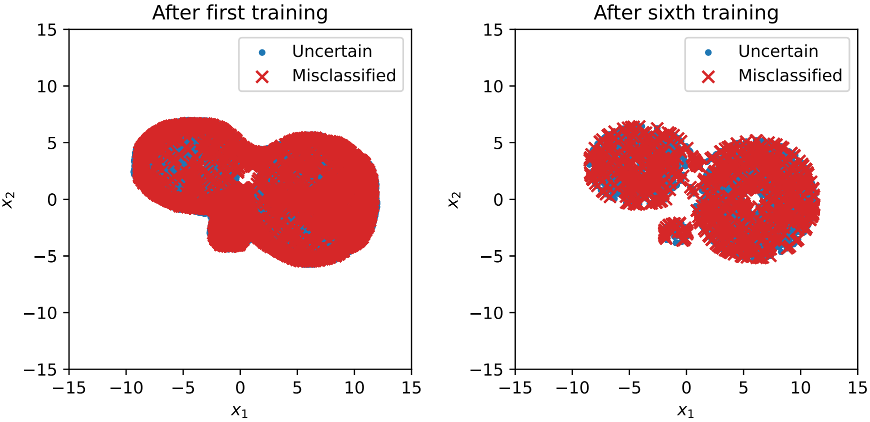
20 regions with similar contribution to value of integral





Learn divisions of a function with multiple peaks

20 regions with similar contribution to value of integral



After sixth training step: above 99% accuracy (100 000 test points).

Toy example: 7D function with large cancellation

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

$$\int f(x)dx = \int 0.1 f_{\text{bg}}(x) \approx 0.1, \text{ on the cube } [-5,5]^7$$

$f_+(x),\ f_-(x)$: two normalized gaussians with $\sigma = 0.3 \times I_7$
f_+ randomly centered in positive 7-hyperoctant.
f_- randomly centered in negative 7-hyperoctant.
$f_{\text{bg}}(x)$: normalized gaussian with $\sigma = 1.0 \times I_7$, $\vec{0}$ centered

NN: Multilayer perceptron: 2 hidden layers, 7D input

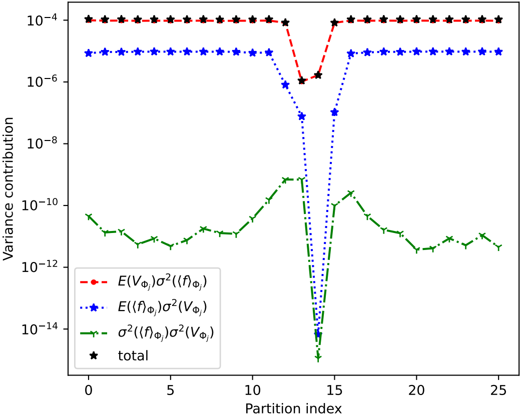
- nodes in 1st hidden layer: 2 $\times$ number of partitions $\times$ 7
- nodes in 2nd hidden layer: number of partitions $\times$ 7
- nodes in output layer: number of partitions
- activation function in last layer is *tanh*, one label per contour.

Toy example: Partitioning and training

- **Objective**: create partitions with similar contributions to variance.
- **Adjust partitions** according the their contribution to variance
- Stop when a **target variance** per region has been reached
- When new partitions are created, a new networks is trained
- Points classified by previous networks are used in repartitioning

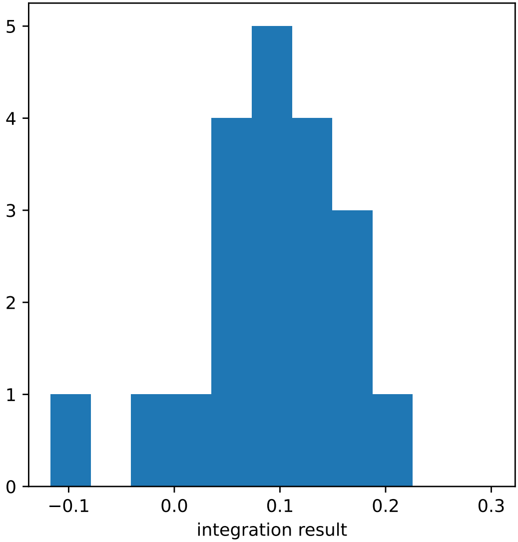
After 8 iterations: 1.4×10^7 evaluations and 26 partitions

Toy example: result for final partitioning



- 2×10^6 $f(x)$ evaluations
- Target: total variance of 0.05^2
- Second variance (reduced by NN) contribution adjusted to be 0.1 of first (reduced by $f(x)$).
- Total number of evaluations 1.6×10^7

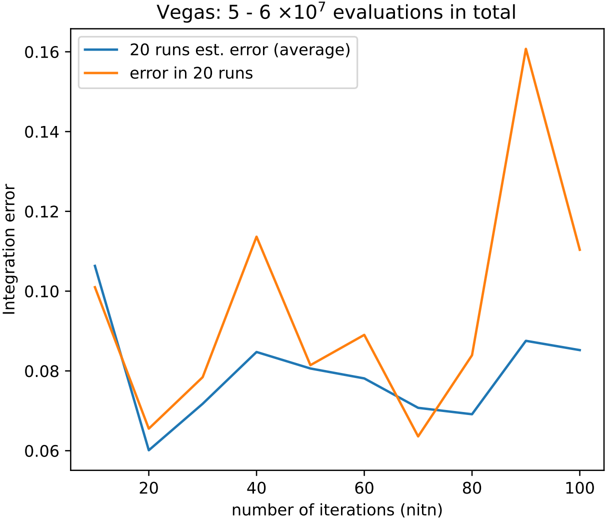
Toy example: Distribution of 20 repetitions



- mean: 0.0924
- σ: 0.0698
- σ is slightly from target (0.05)

Toy example: Simple comparison with Vegas+

Using `python` vegas module [`https://vegas.readthedocs.io/en/latest/index.html`] [G. P. Lepage, arxiv:2009.05112]



- Target estimated error of similar size
- Needed total number of evaluations of $f(x)$ is larger
- Resulting error is not consistent
- Vegas+ is still **much faster** for fast $f(x)$

Quark pair to electron + positron, event generation

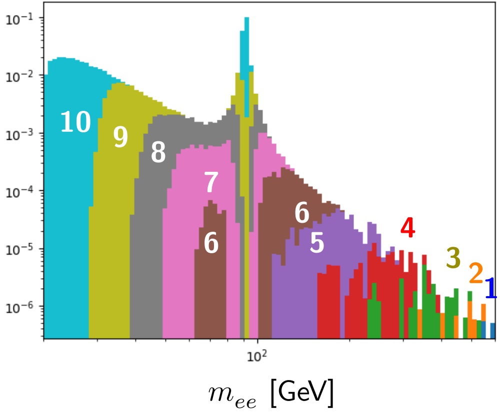
Very simple example:

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

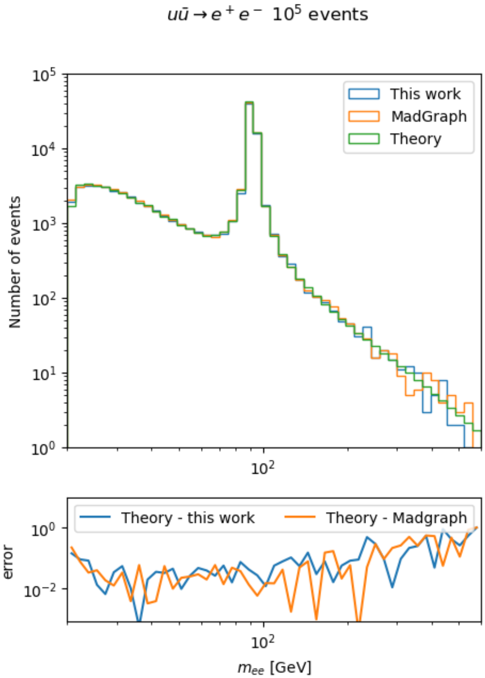
- `ROOT` - `TGenPhaseSpace`: phase space generator.
- `Madgraph` (standalon mode): matrix element.
- `NNPDF23`: parton density function.
- cuts: leptons: $p_T > 10$ GeV, $|\eta| < 2.5$

Generate events: 10 usable regions

e^-e^+ invariant mass projection



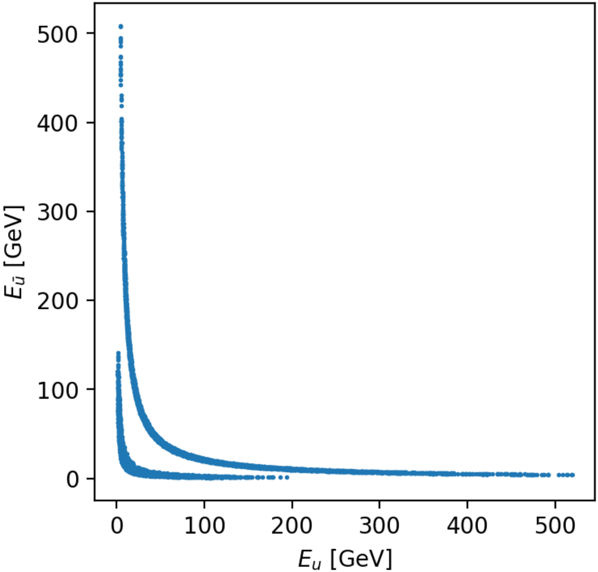
- Sample each region until enough events are accumulated. **NN can tell which regions points belong to**.
- Select points using correct result.



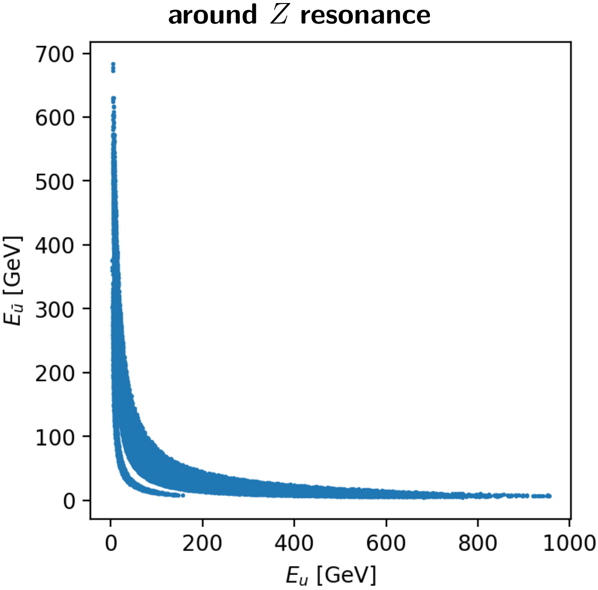
- 10^5 unweighted events
- High m_{ee} error expected from thinning of sample.
- Invariant mass around Z resonance is similar when comparing to MadGraph
- Efficiency of selection of unweighted events increases with more regions. But more regions requires more points for training

Vanity plots: Region 10 as seen by the NN

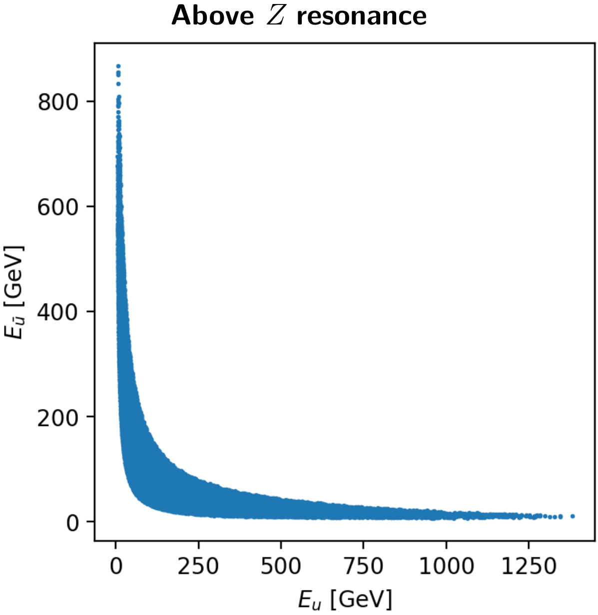
Z **resonance and low** m_{ee}



Vanity plots: Region 6 as seen by the NN



Vanity plots: Region 5 as seen by the NN



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!