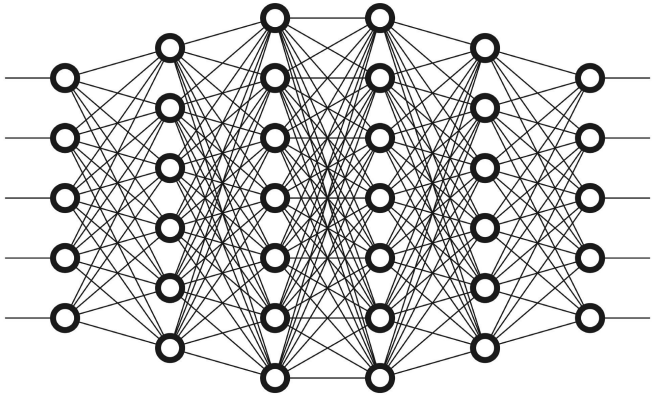
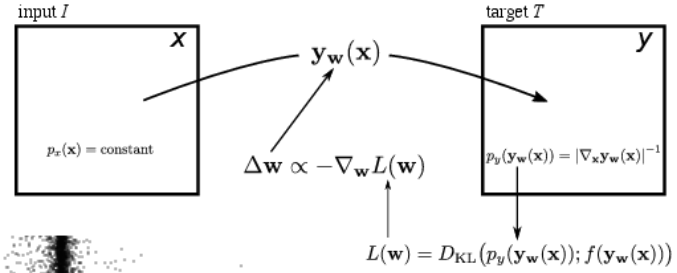
Event Generation with Neural Nets

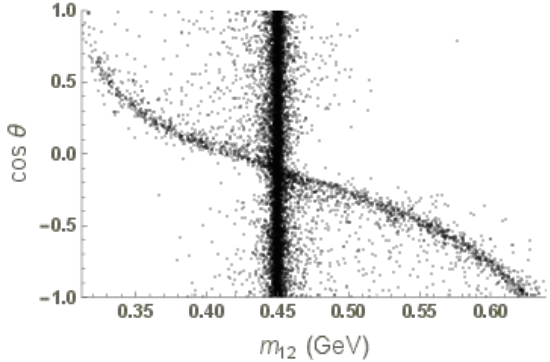
Matthew D. Klimek
Cornell Univ.
Korea Univ.

with M. Perelstein
1810.11509 + wip

HKUST Mini-workshop
10 Jan 2020



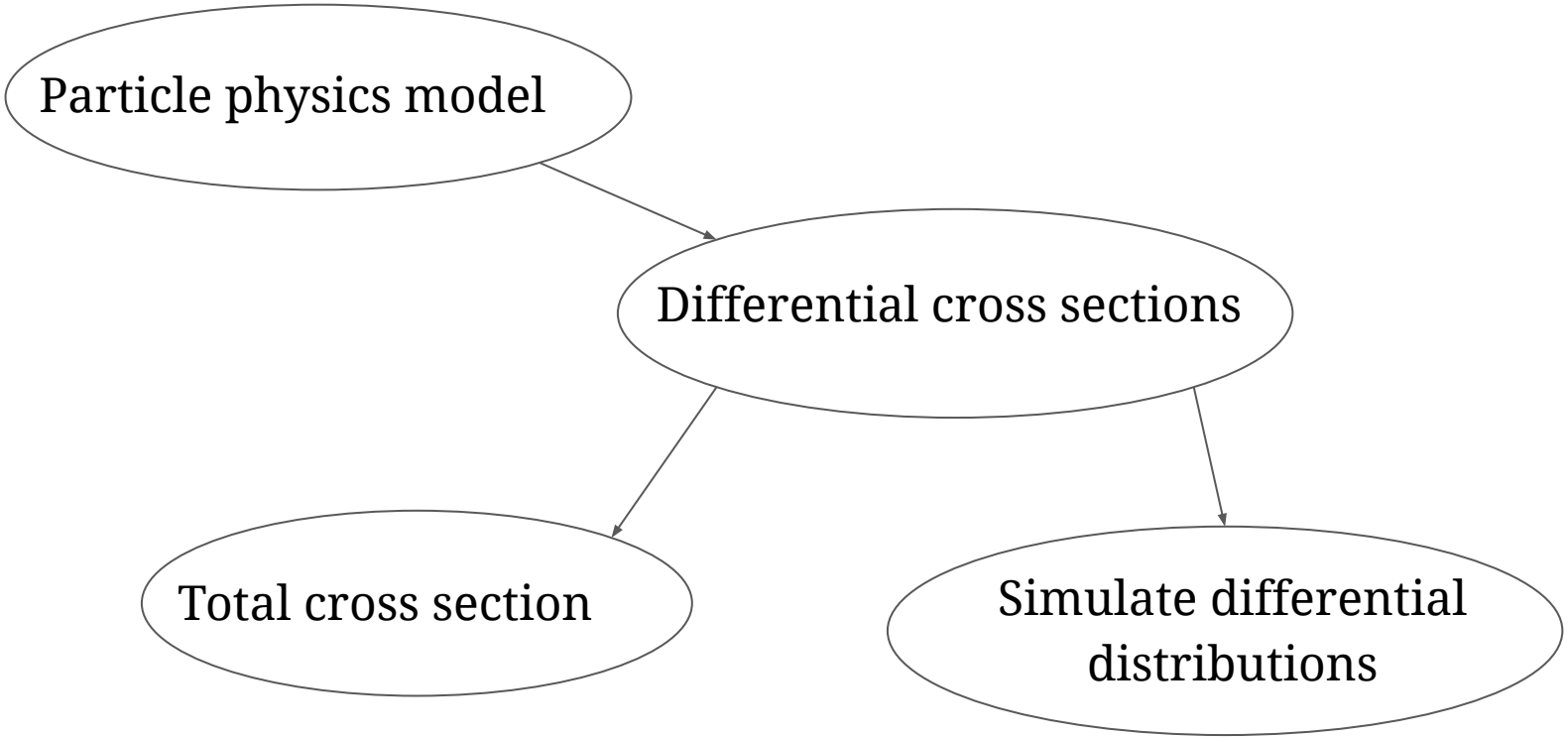




Goals for this talk

- **Using NNs for MC event generator is not fancy or exotic.**
  It is the most natural way to improve current techniques.
- Details of implementation.
- **Results for typical particle physics problems & improvements in efficiency.**

MC integration/generation: general problem



MC: the basic technique

Simplest case:

- ➢ Sample the domain of the function uniformly.
- ➢ Sum the function values at the N random points.

An estimate of the integral is then obtained as: $$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

What is the area of a circle?

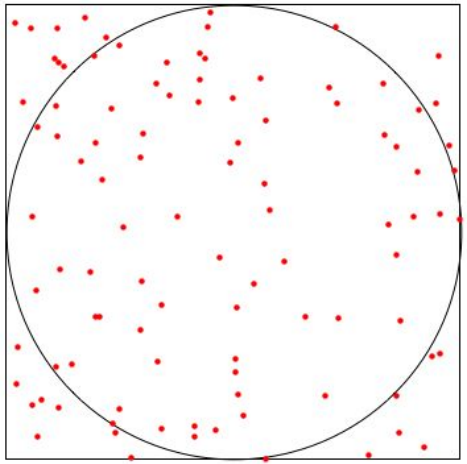
MC: the basic technique

Simplest case:

- ➢ Sample the domain of the function uniformly.
- ➢ Sum the function values at the N random points.

An estimate of the integral is then obtained as: $$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

What is the area of a circle?



100 points, 71 fell inside

A = 2.84

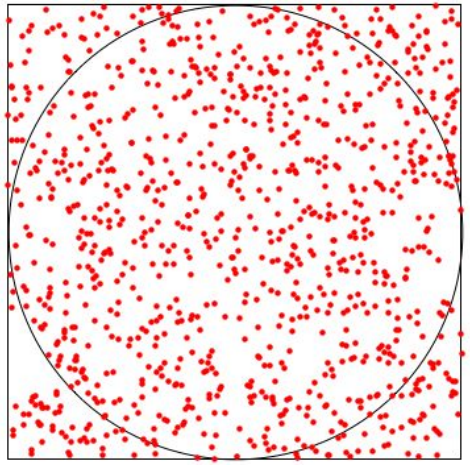
MC: the basic technique

Simplest case:

- ➢ Sample the domain of the function uniformly.
- ➢ Sum the function values at the N random points.

An estimate of the integral is then obtained as: $$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

What is the area of a circle?



1000 points, 772 fell inside

A = 3.088

MC: the basic technique

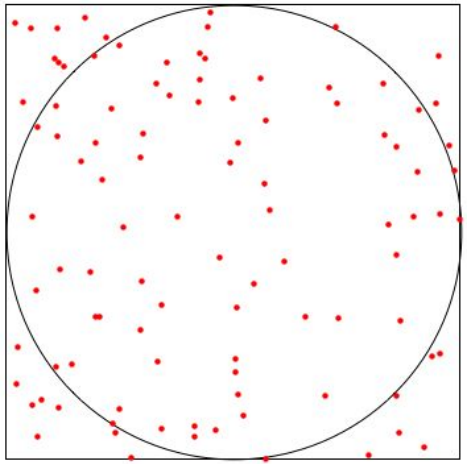
Simplest case:

- ➢ Sample the domain of the function uniformly.
- ➢ Sum the function values at the *N* random points.

An estimate of the integral is then obtained as: $$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

$$f(r) = (1 - r)\Theta(1 - r)$$

I = π/2 = 1.57



100 points, 71 fell inside

I = 1.21

MC: the basic technique

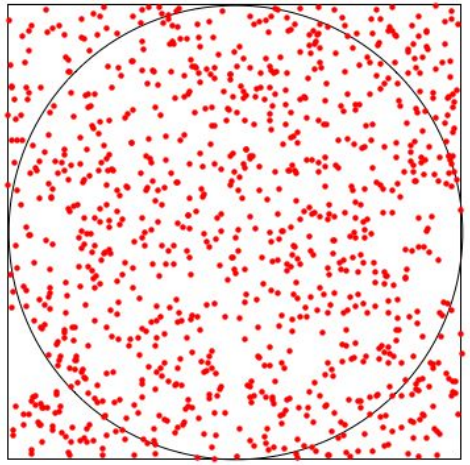
Simplest case:

- ➢ Sample the domain of the function uniformly.
- ➢ Sum the function values at the *N* random points.

An estimate of the integral is then obtained as: $$I \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

$$f(r) = (1 - r)\Theta(1 - r)$$

I = π/2 = 1.57

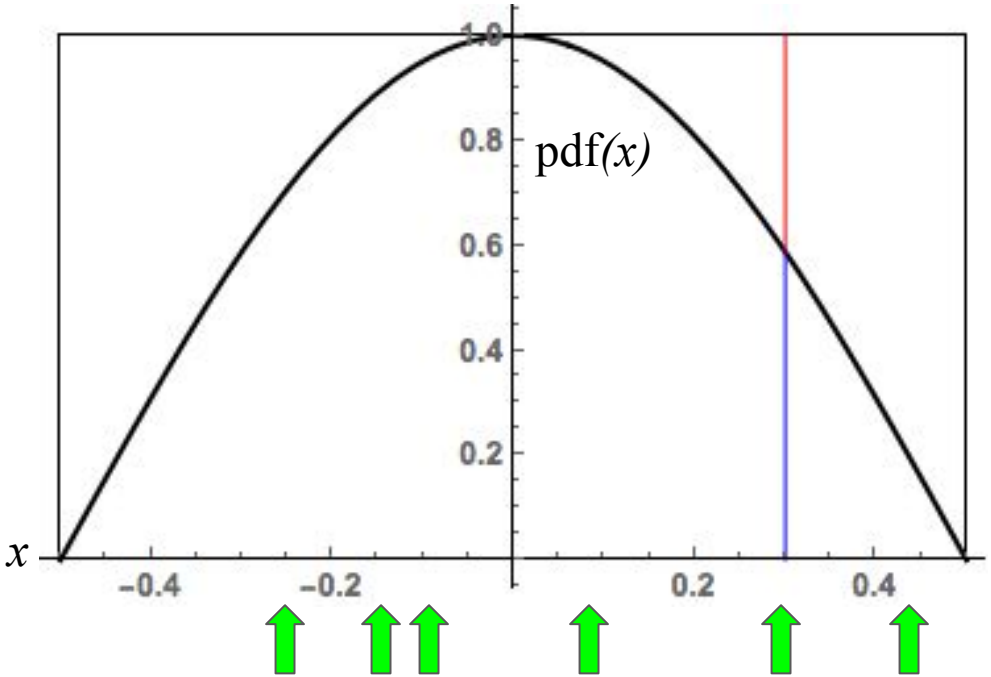


1000 points, 772 fell inside

I = 1.63

MC: Event generation

Generate events according to this 1-d p.d.f.:



Draw points randomly

Discard

Keep

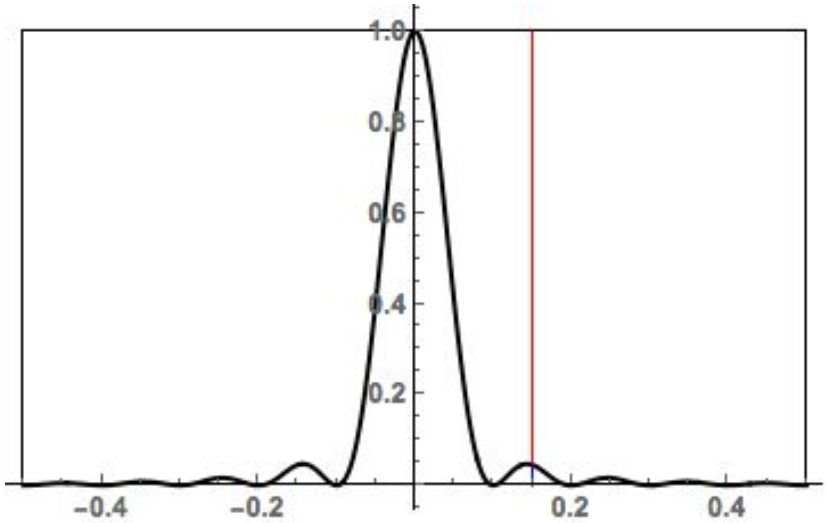
Keep randomly drawn point with probability:

$$\epsilon(x) = \frac{f(x)}{f_{\max}}$$

The basic technique

What about a function with a sharp peak?

$$\epsilon(x) = \frac{f(x)}{f_{\max}} \ll 1$$



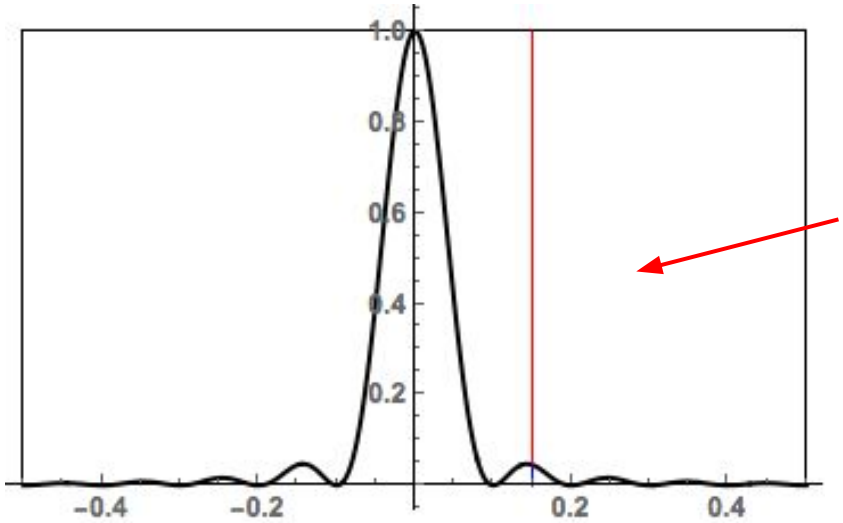
Most points will be discarded

Physical cross sections are often highly variable/peaked in some regions of phase space: on-shell resonances, collinear singularities, etc.

The basic technique

What about a function with a sharp peak?

$$\epsilon(x) = \frac{f(x)}{f_{\max}} \ll 1$$



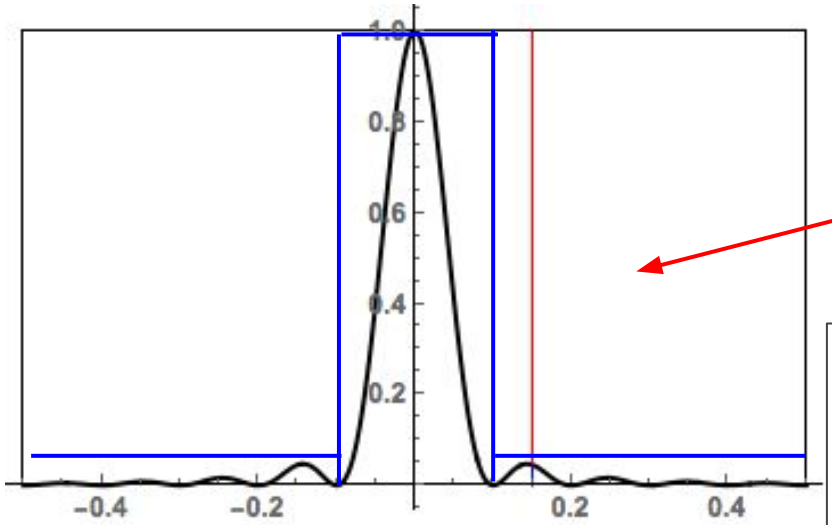
Obvious fix:
Don't sample much out here

Physical cross sections are often highly variable/peaked in some regions of phase space: on-shell resonances, collinear singularities, etc.

The basic technique

What about a function with a sharp peak?

$$\epsilon(x) = \frac{f(x)}{f_{\max}} \ll 1$$



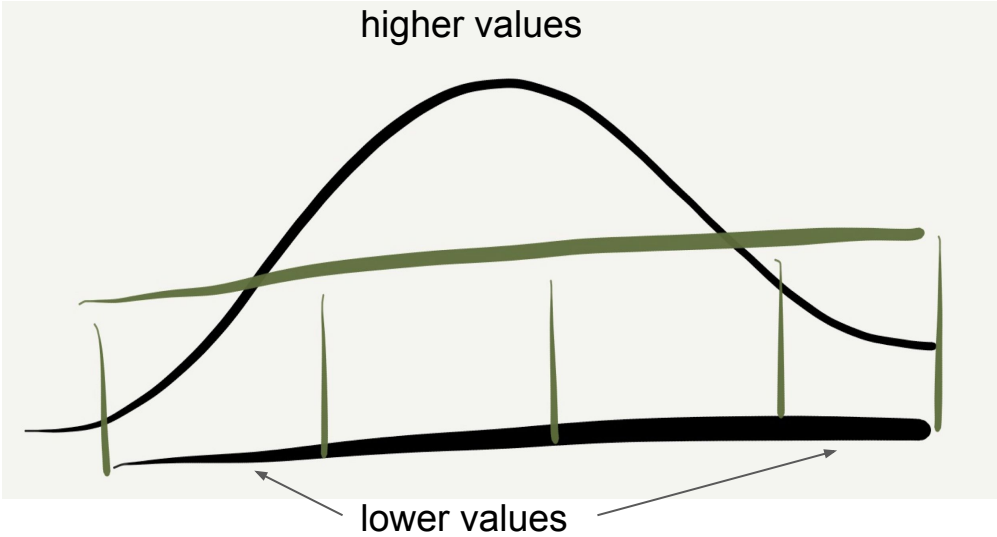
Obvious fix:
Don't sample much out here

Needed: a way to build a function that:
- ➢ is easily sampled, and
- ➢ approximates the target function.

Physical cross sections are often highly variable/peaked in some regions of phase space: on-shell resonances, collinear singularities, etc.

VEGAS (G.P. Lepage, *J.Comp.Phys.* 1978)

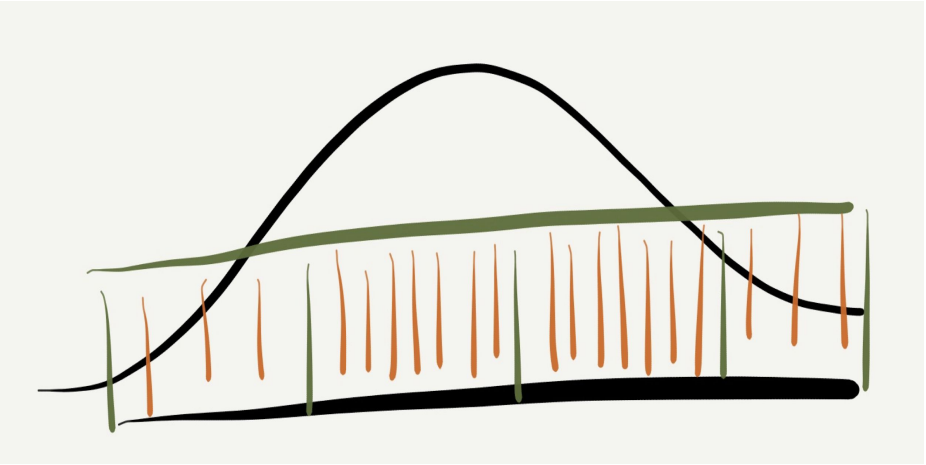
- Approximate the function by a *set of bins* of containing *equal amounts* of the integral of the function.
- To sample the function: simply *choose a random bin*; then *sample uniformly* within that bin.
- Adapt bin edges to better match the function.



Sample the function values

VEGAS (G.P. Lepage, *J.Comp.Phys.* 1978)

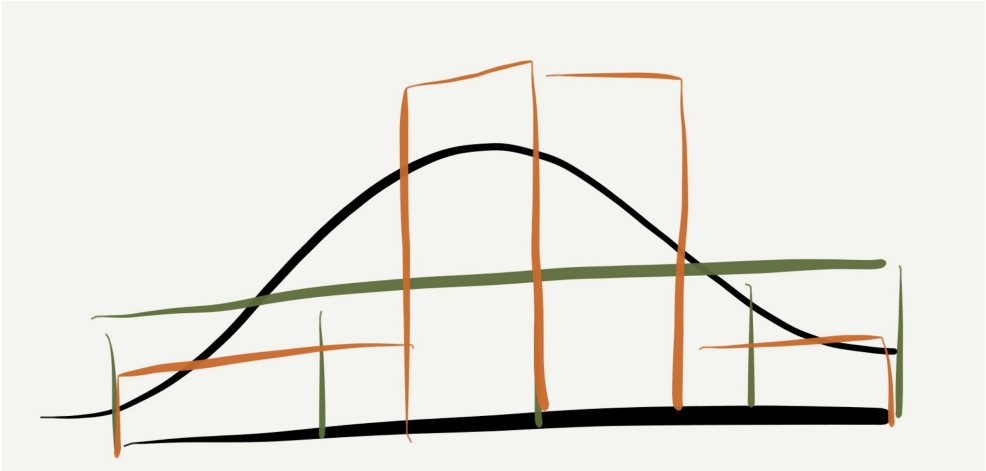
- Approximate the function by a *set of bins* of containing *equal amounts* of the integral of the function.
- To sample the function: simply *choose a random bin*; then *sample uniformly* within that bin.
- Adapt bin edges to better match the function.



Subdivide bins according to average function value in that bin

VEGAS (G.P. Lepage, *J.Comp.Phys.* 1978)

- Approximate the function by a *set of bins* of containing *equal amounts* of the integral of the function.
- To sample the function: simply *choose a random bin*; then *sample uniformly* within that bin.
- Adapt bin edges to better match the function.

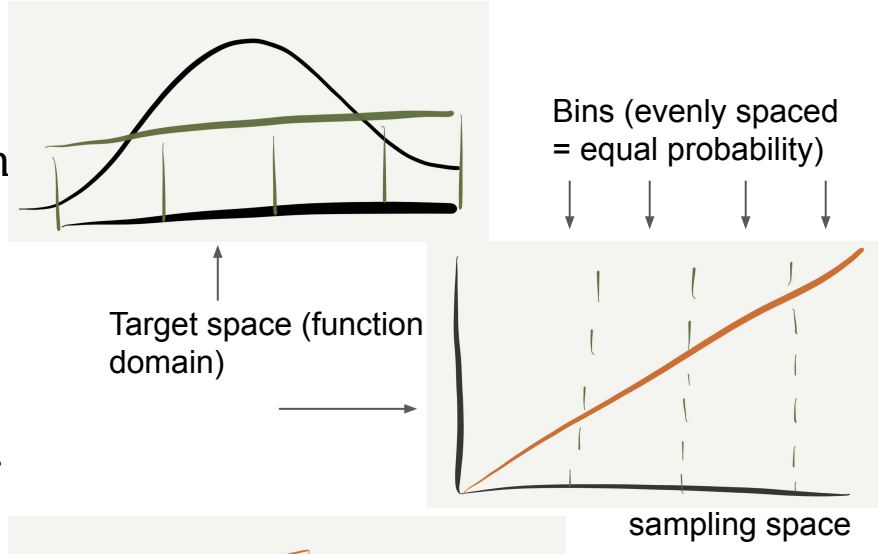


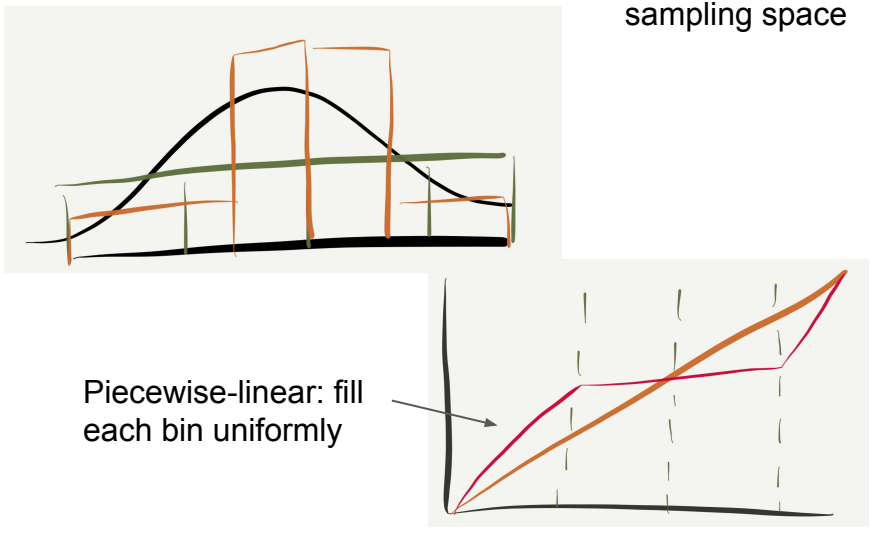
Merge bins back to original number

VEGAS

VEGAS can be viewed as building a **map** from a *sampling space*
(over which points are drawn uniformly)
onto the *target space* (where the density of points approximates the desired function)
in a *piecewise-linear way* (fill bins uniformly).

The training adjusts the values of the map at fixed discrete points.





VEGAS is ML

- *VEGAS is a form of machine learning.*

- The algorithm goes through a *training process:*
  It *adjusts* the location of the bin edges
  Metric: *decreases the variance* in the function values in each bin.

What if we could extend this to adjust the map at **every** point?

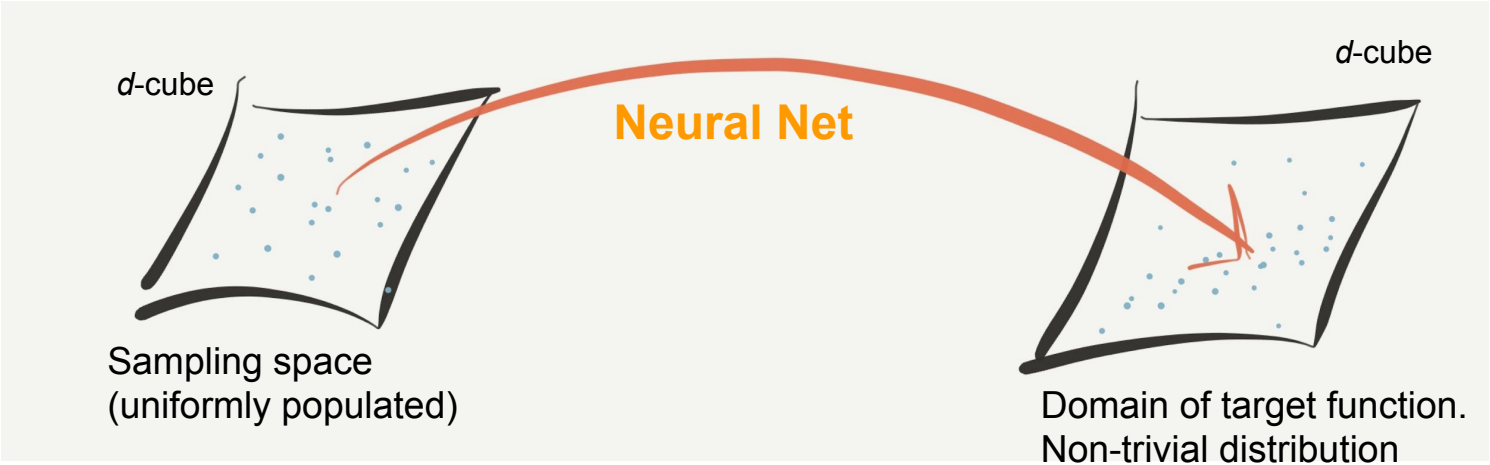
We would need a map that is:

- defined in terms of some *adjustable parameters*
- capable of approximating **any** smooth map.

This is the basis of artificial neural nets.

General Approach

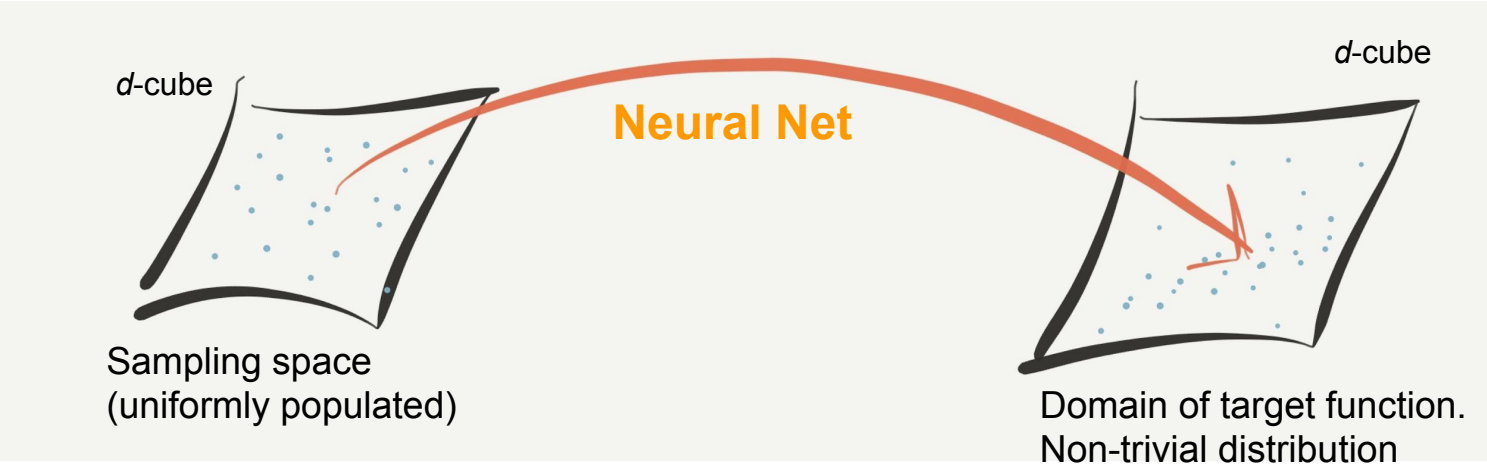
First suggested by Bendavid 1707.00028, applied only to Gaussian functions



$$\mathcal{J}^{-1}(N)|_{x=x(y)} = \left|\frac{\partial N}{\partial x(y)}\right|^{-1}$$

General Approach

First suggested by Bendavid 1707.00028, applied only to Gaussian functions



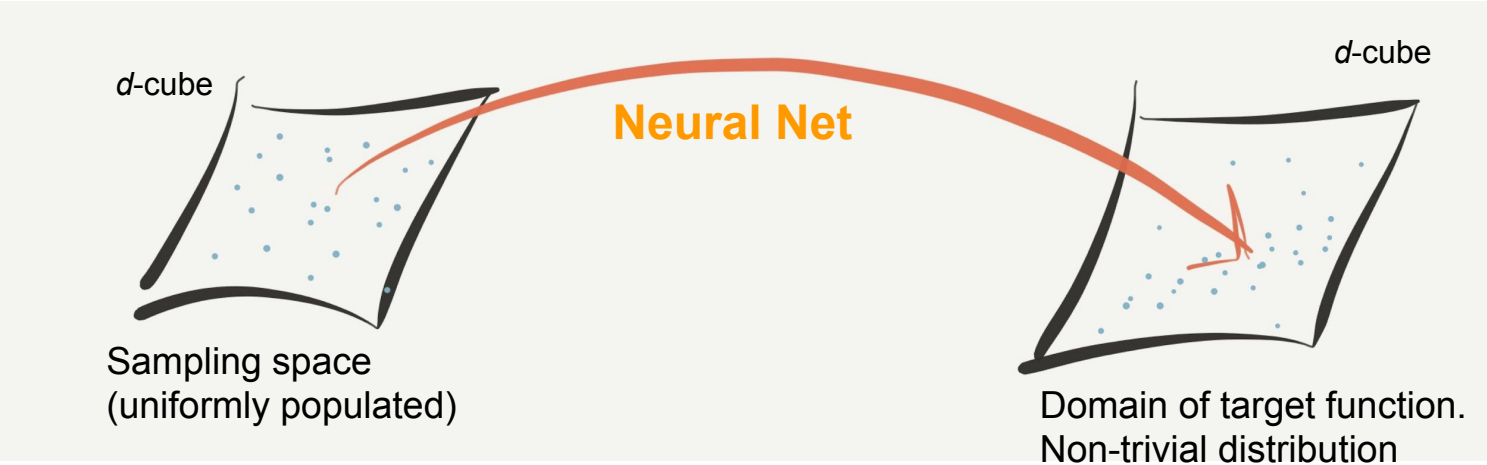
$$\mathcal{J}^{-1}(N)|_{x=x(y)} = \left|\frac{\partial N}{\partial x(y)}\right|^{-1}$$

Statistical distance between the true and induced pdfs:
Kullbeck-Leibler divergence.

$$D_{KL}(f;g) = \int f(x)\log\left(\frac{f(x)}{g(x)}\right)dx$$

General Approach

First suggested by Bendavid 1707.00028, applied only to Gaussian functions

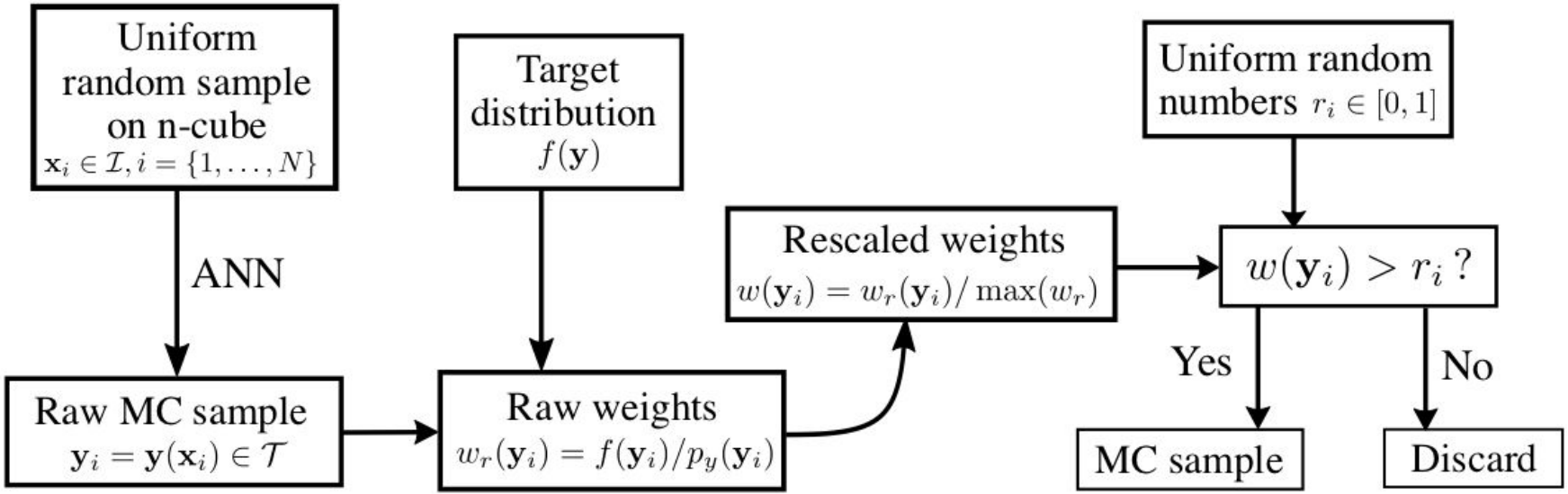


$$\mathcal{J}^{-1}(N)|_{x=x(y)} = \left|\frac{\partial N}{\partial x(y)}\right|^{-1}$$

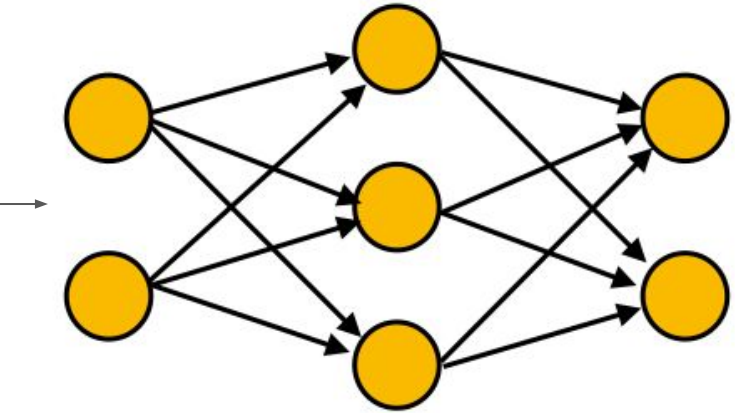
Statistical distance between the true and induced pdfs:
Kullbeck-Leibler divergence.

$$D_{KL}(f;g) = \int f(x)\log\left(\frac{f(x)}{g(x)}\right)dx$$

Adjust the *NN* so that D_{KL} between *1*/Jac(*NN*) and the differential cross section is minimized.


Uniform random sample on n-cube
$\mathbf{x}_i \in \mathcal{I}, i = \{1, \ldots, N\}$
ANN
Raw MC sample
$\mathbf{y}_i = \mathbf{y}(\mathbf{x}_i) \in \mathcal{T}$
Target distribution
$f(\mathbf{y})$
Raw weights
$w_r(\mathbf{y}_i) = f(\mathbf{y}_i)/p_y(\mathbf{y}_i)$
Rescaled weights
$w(\mathbf{y}_i) = w_r(\mathbf{y}_i)/\max(w_r)$
Uniform random numbers $r_i \in [0, 1]$
$w(\mathbf{y}_i) > r_i$?
Yes
No
MC sample
Discard

Our implementation



Same number of input & output nodes = phase space dimension

Uniform distribution on hypercube

Output function maps onto hypercube

Induced distribution on hypercube

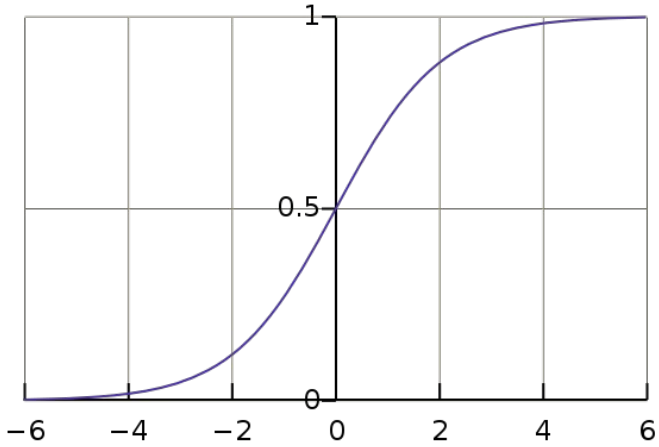
Design questions:

- How many hidden layers/nodes?
- What to use as activation/output functions?

Our basic implementation

- A common choice of output function is the sigmoid:

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



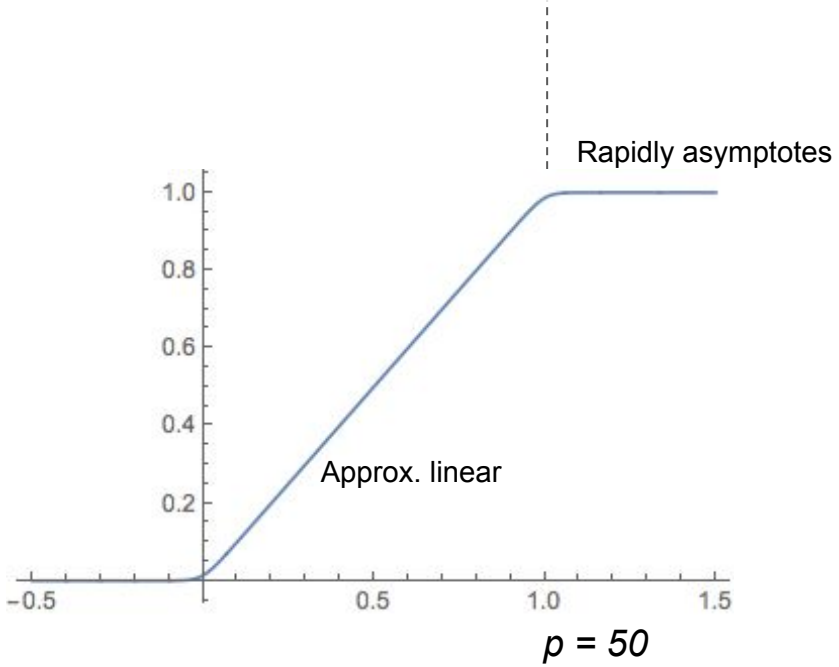
Approaches asymptotic values slowly → hard to populate the edges of phase space

Output layer & Activation function

The NN can generate any real value, but must be mapped onto a hypercube:

$$SC(x) = \frac{1}{p} \log\left(\frac{1 + e^{px}}{1 + e^{p(x-1)}}\right)$$

"Soft clipping" function

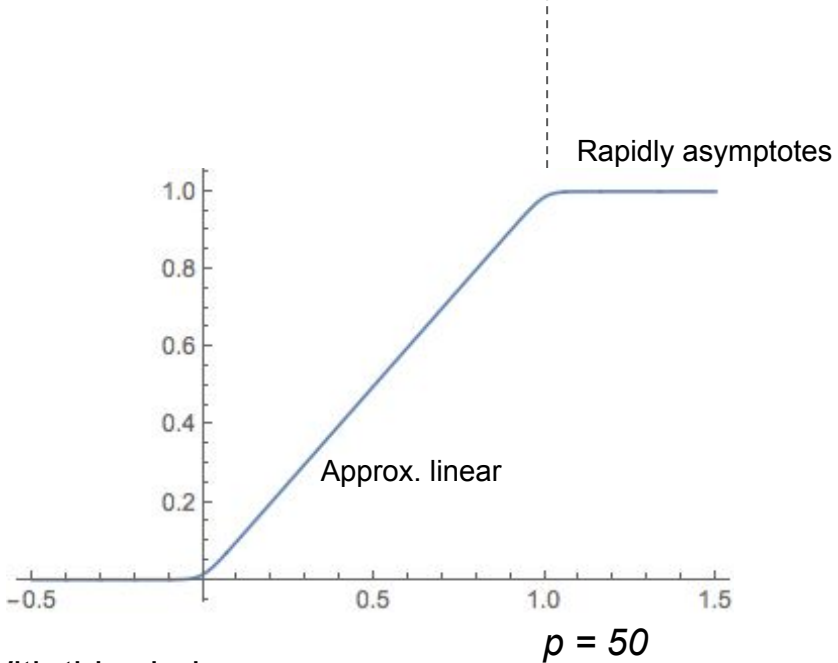


Output layer & Activation function

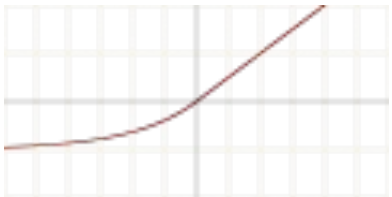
The NN can generate any real value, but must be mapped onto a hypercube:

$$SC(x) = \frac{1}{p} \log\left(\frac{1 + e^{px}}{1 + e^{p(x-1)}}\right)$$

"Soft clipping" function



p = 50



$$ELU(x) = \begin{cases} x & x > 0 \\ e^x - 1 & x < 0 \end{cases}$$

With this choice, a traditional ELU activation function is sufficient.

Choice of coordinates

A generic prescription that maps any *n*-body phase space onto a hypercube.

Define $q_i \in [0,1]$: rescaled range of invariant mass of the system composed of particles {i+1, …, n}.

There are *N-2* of these: {*n-1*, *n*}, …, {*2*, …, *n*}.

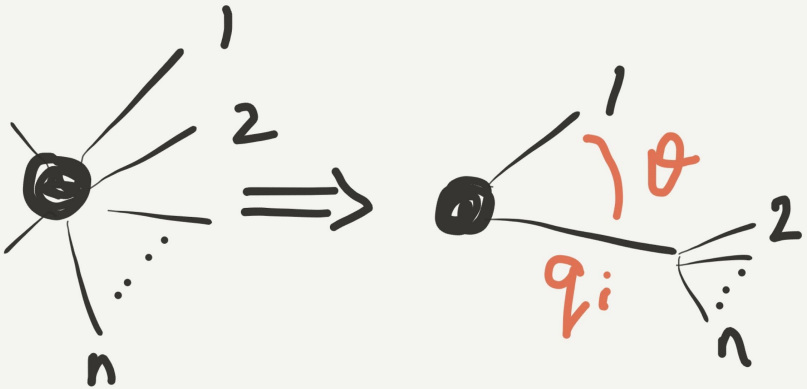
$$q_1 \leftrightarrow \left[\sum_{i=2}^{n} m_i, \sqrt{s} - m_1\right]$$

Rescale *2N-5* relative angles to the range [0,1].

Choose overall rotation (3 Euler angles).

Phase space dimensionality:

(N - 2) + (2N - 5) + 3 = 3N - 4

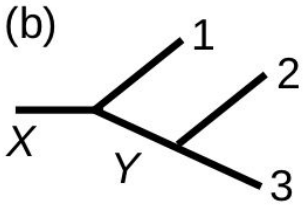


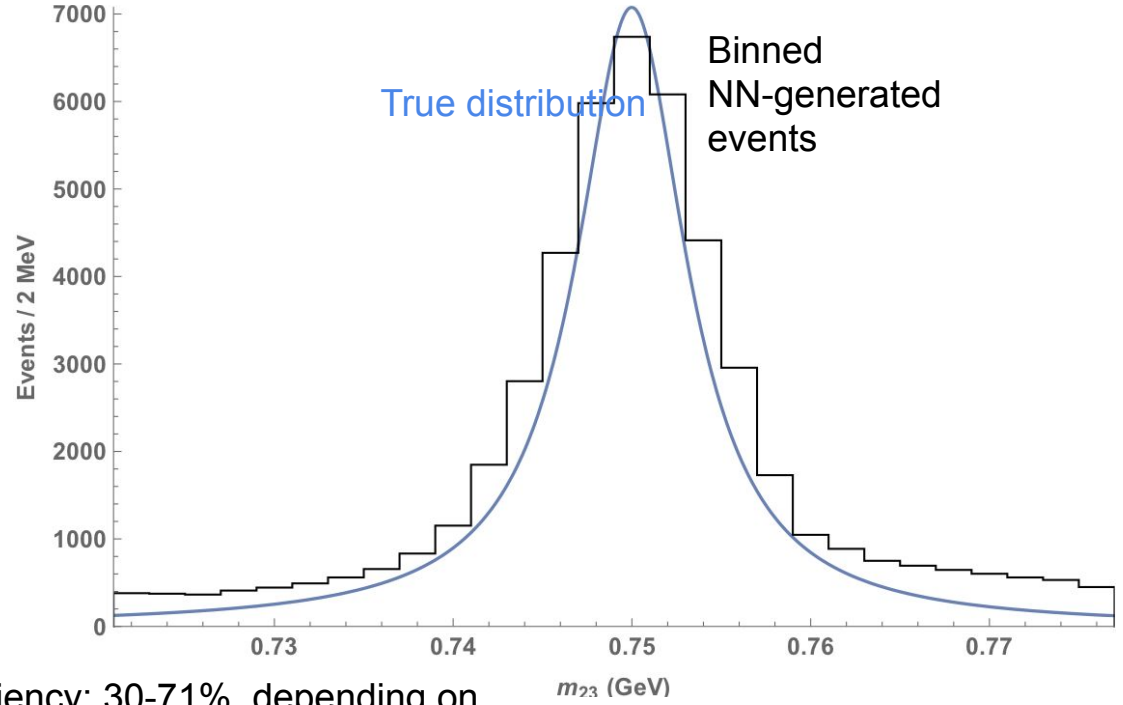
Complete setup

- Physics input: simply type analytical expression for the differential cross section into the code, or provide a function that code can call. (Easy to interface with Feynman diagram calculator.)
- Training:
  - Draw a sample of 100 uniform random points → feed through NN.
  - Compute Jacobian of NN at each point (depends on current NN parameters).
  - Compute the KL divergence the Jacobian and the target diff. cross section.
  - Try to minimize KL divergence by adjusting the NN (gradient descent).
- Save and use NN parameters which gave the lowest KL divergence.

Example: 3-body decay through resonance

3-body decay via an intermediate resonance with mass 0.75 GeV.





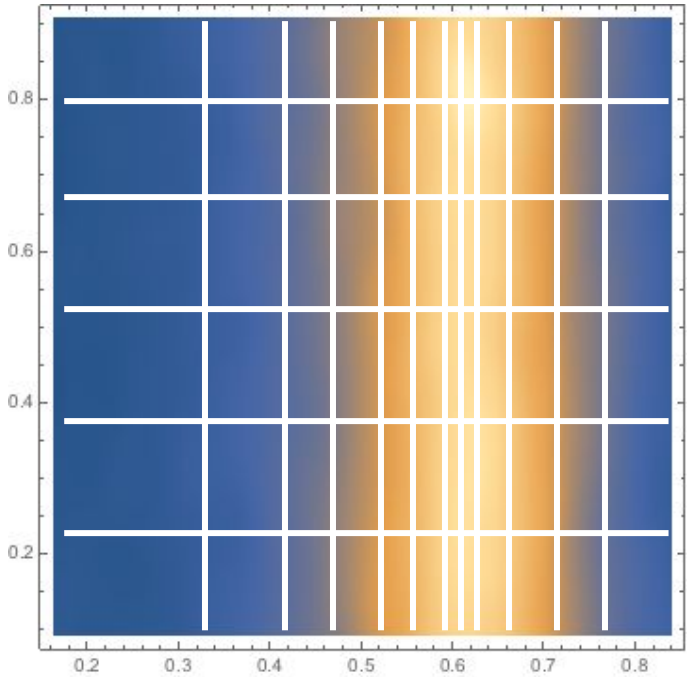
Our efficiency: 30-71%, depending on resonance width

MadGraph's efficiency (VEGAS-based): 6%

Multi-dimensional case: VEGAS

VEGAS uses a rectangular grid.

For multi-dimensional integrals, VEGAS needs any sharp feature to be aligned with a grid axis.



Multi-dimensional case: VEGAS

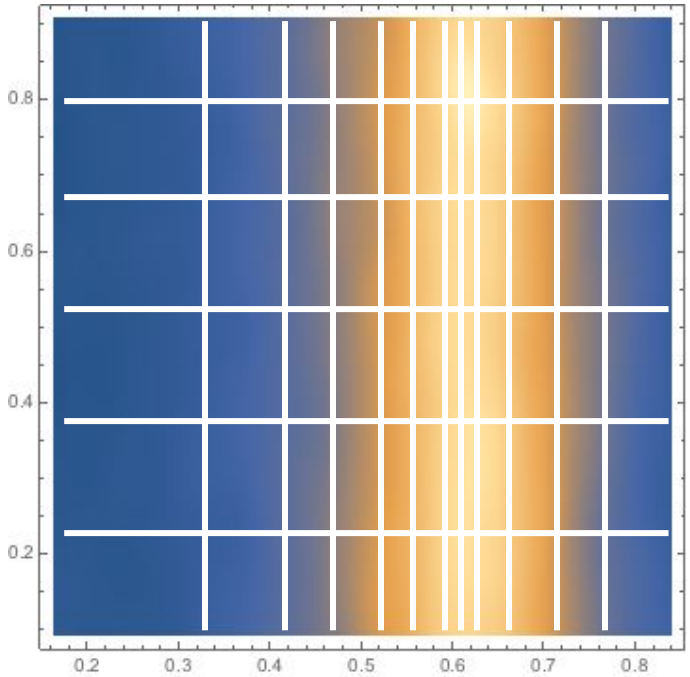
What about matrix elements that have multiple sharp features?

This is currently handled with *multi-channel integration*:

> Define multiple grids, one for each feature

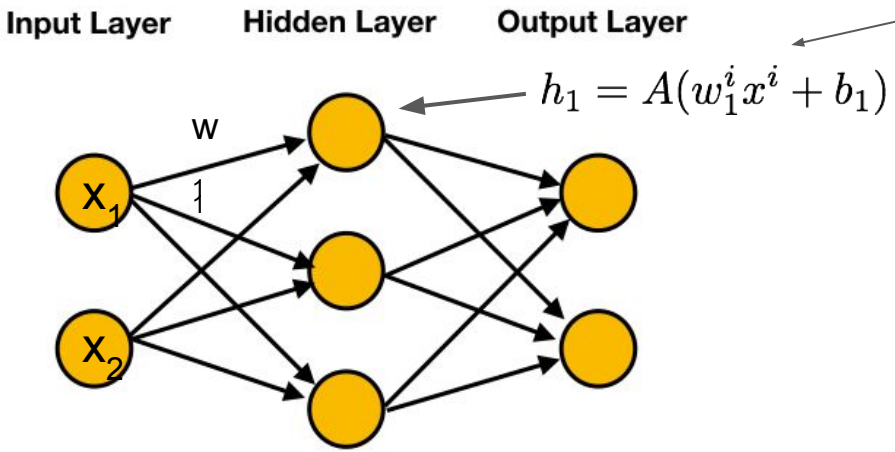
> Tune and sample from each grid separately

> Relative weights among grids must also be tuned.



Multi-dimensional case: NN

The NN approach has no intrinsic axes.

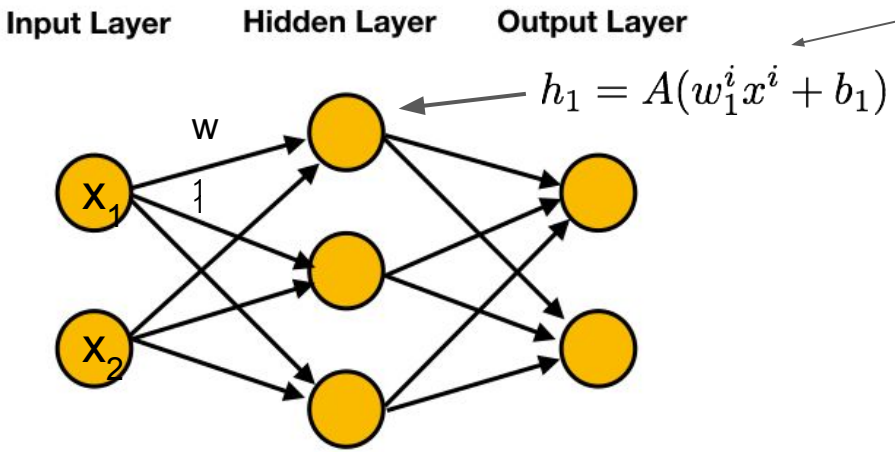


Each node is free to rotate to a new coordinate system.

The NN *learns* what the most interesting coordinates are.

Multi-dimensional case: NN

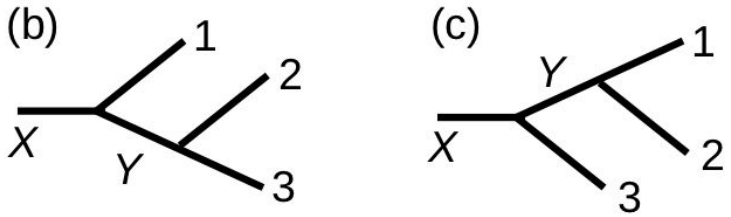
The NN approach has no intrinsic axes.



Each node is free to rotate to a new coordinate system.

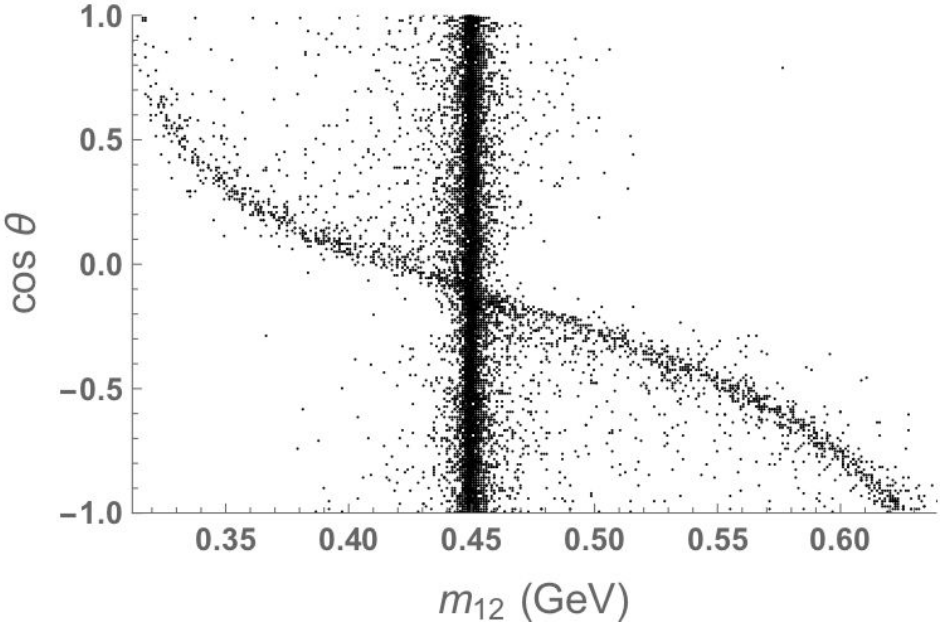
The NN *learns* what the most interesting coordinates are.

We trained the NN on a decay that can proceed through two different diagrams:



In our coordinate system, these cannot be orthogonal!

Multi-dimensional case: NN

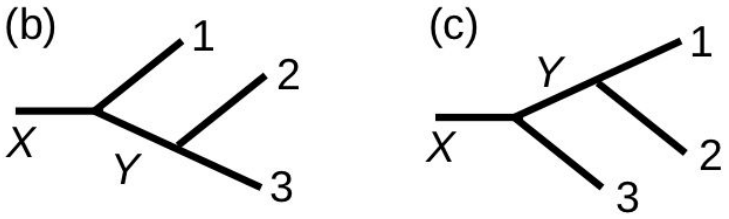


The NN automatically finds and maps onto both features!

Our efficiency: 54%

MadGraph's efficiency (VEGAS-based): 6%

We trained the NN on a decay that can proceed through two different diagrams:



In our coordinate system, these cannot be orthogonal!

qqg: phase space singularities?

Singularities at quark momentum fraction $x = 1$

$$\frac{d\sigma}{dx_1 dx_2} \propto \frac{x_1^2 + x_2^2}{(1-x_1)(1-x_2)}$$

An appropriate kinematic cut can be specified in the code.

To cut on a quantity $x > x_{cut} > 0$, multiply the cross section during training by

$$\kappa(x) = \begin{cases} 1 & x > x_{\text{cut}} \\ (x/x_{\text{cut}})^n & x < x_{\text{cut}} \end{cases}$$

qqg: phase space singularities?

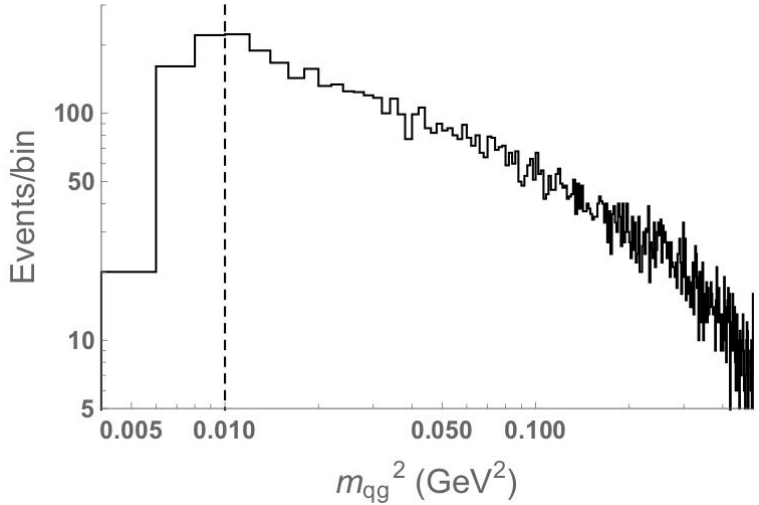
Singularities at quark momentum fraction $x = 1$

$$\frac{d\sigma}{dx_1 dx_2} \propto \frac{x_1^2 + x_2^2}{(1-x_1)(1-x_2)}$$

An appropriate kinematic cut can be specified in the code.

To cut on a quantity $x > x_{cut} > 0$, multiply the cross section during training by

$$\kappa(x) = \begin{cases} 1 & x > x_{\text{cut}} \\ (x/x_{\text{cut}})^n & x < x_{\text{cut}} \end{cases}$$



Our efficiency: 65-75% depending on cuts and *n*

MadGraph's efficiency (VEGAS-based): 4%

sqrt(s) = 1 GeV, $m_{qg} < 0.1$ GeV, n=8

Future Directions: New results

See forthcoming work with Perelstein & I-Kai (Calvin) Chen.

Increased number of final state particles → Performance still good!

Introduce gradient clipping → Solves issues of numerical stability when Jacobian becomes small

More physical examples: $H \to 4l$, *etc.*

Future Directions: MadGraph

We have begun to integrate our NN code with the MadGraph diagram calculator.

MadGraph provides a piece of code that represents the target differential cross section.

```
my_integrator = SimpleMonteCarloIntegrator(my_integrand,
    n_iterations=50, n_points_per_iterations=10000, accuracy_target=None)

print '\nFinal result: %.4e +/- %.2e'%my_integrator.integrate()
```

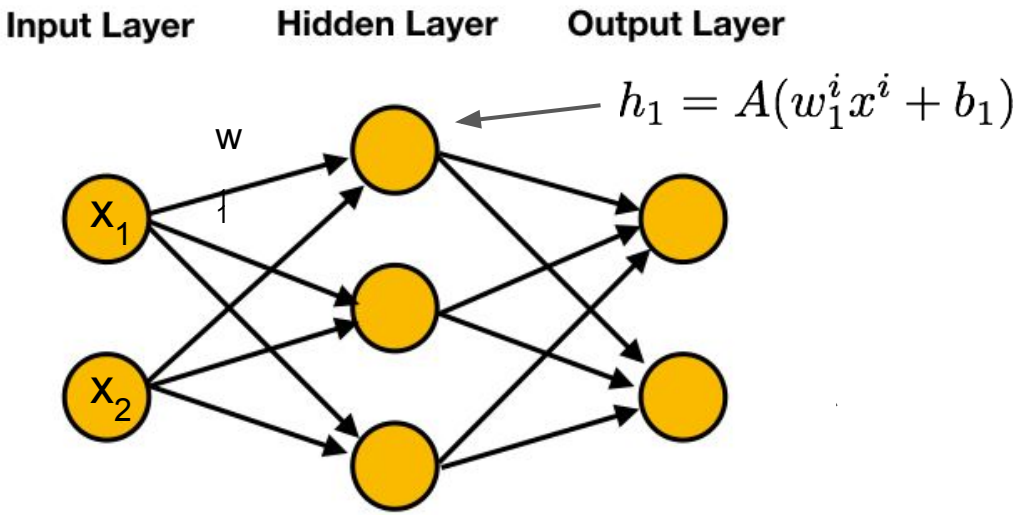
Hope for a publicly usable code soon.

Summary

- Neural Networks allow a continuum implementation of the classic VEGAS phase space integrator/generator.
- We have presented several typical physical scenarios that can be handled by this method.
- We achieve good unweighting efficiency compared to classic methods, and without the need to choose special coordinate systems.
- We are currently integrating this technique with MadGraph and improving performance further.

Backup Slides

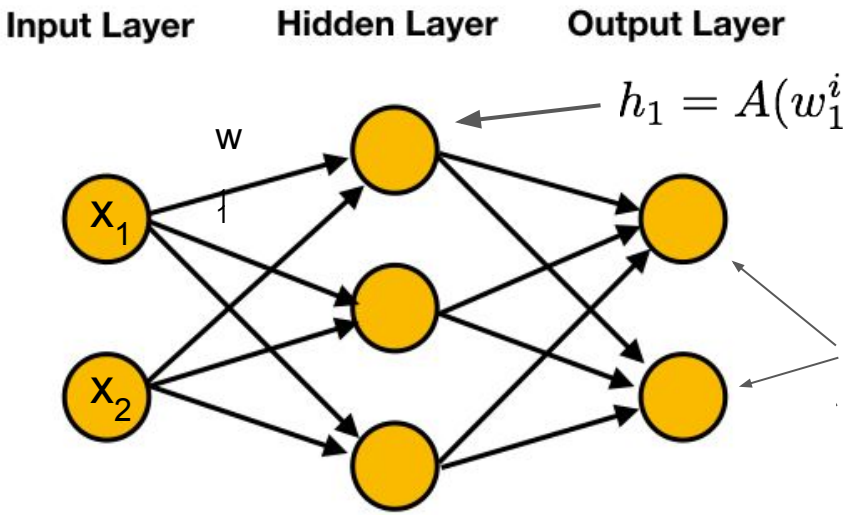
NN Basics



Each hidden node takes a linear combination of the inputs, specified by the *weights* w_1^i plus a constant *bias* b_1, and transforms it by some non-linear *activation function A.*

The weights and biases together comprise the **parameters** of the net.

NN Basics

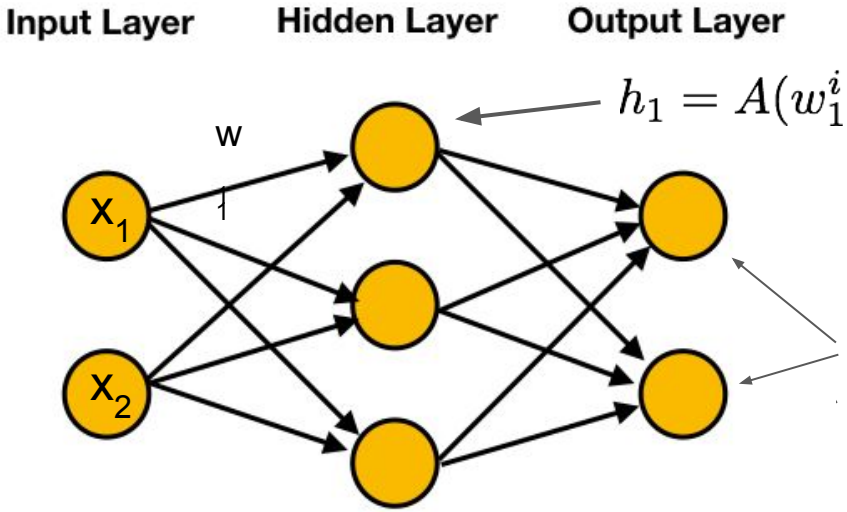


Each hidden node takes a linear combination of the inputs, specified by the *weights* w_1^i plus a constant *bias* b_1, and transforms it by some non-linear *activation function A.*

The weights and biases together comprise the **parameters** of the net.

The output layer is similar, but its activation function should be chosen to map any real number onto the desired output range.

NN Basics



Each hidden node takes a linear combination of the inputs, specified by the *weights* w_1^i plus a constant *bias* b_1, and transforms it by some non-linear *activation function A.*

The weights and biases together comprise the **parameters** of the net.

The output layer is similar, but its activation function should be chosen to map any real number onto the desired output range.

Loss function: a measure of how far the NN is from the desired behavior (KL divergence).

Training:

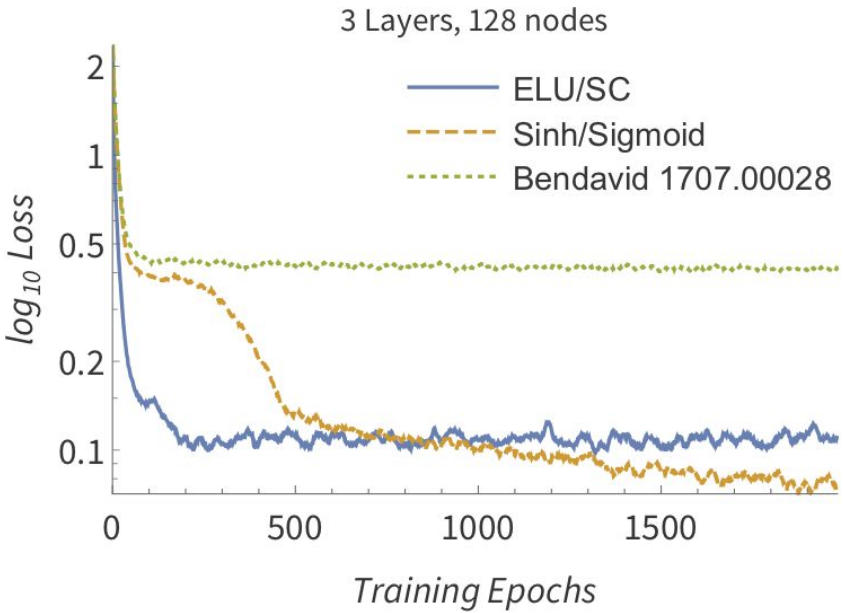
Compute gradient of the loss function w.r.t. all parameters.

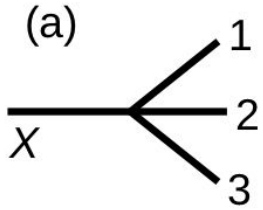
Adjust parameters proportionally (gradient descent).

3-body Dalitz, constant matrix element

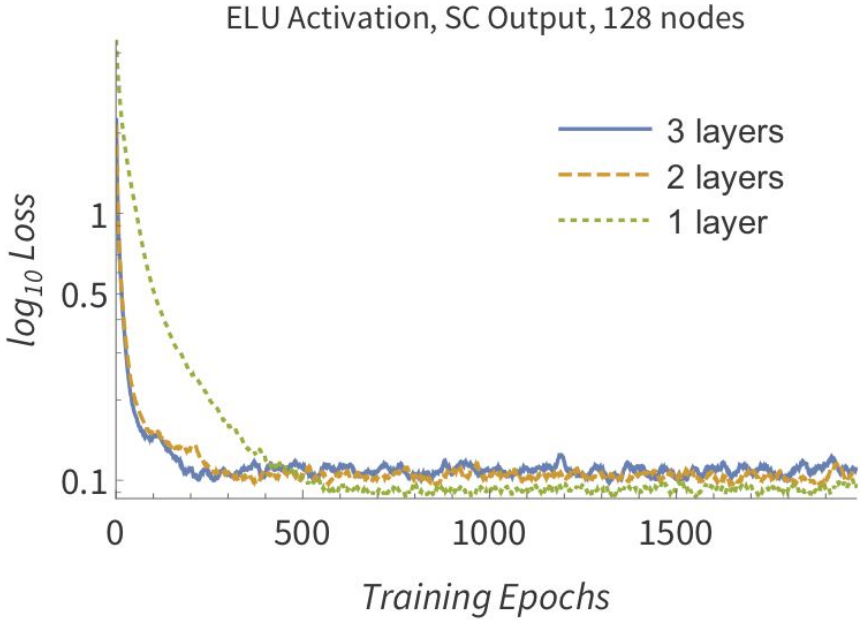
- 2-dimensional phase space. Parametrize with:
  - m_{23} and θ, the angle between p_2 and p_1 in the m_{23} rest frame.
  - Phase space is flat in θ.
  - Both variables can be shifted/scaled to lie in a unit square.

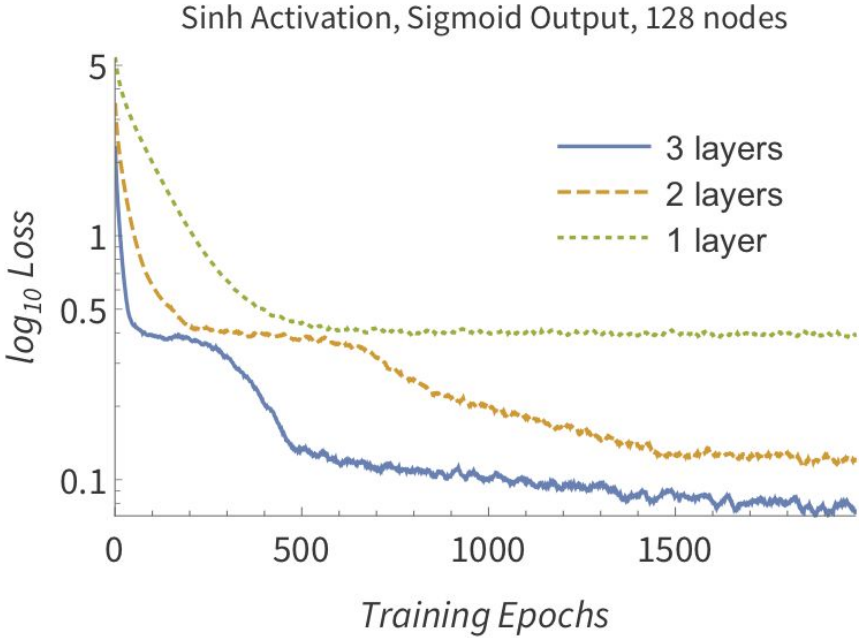
M = sqrt(s) = 1 GeV
m_1 = 0.1 GeV
m_2 = 0.2 GeV
m_3 = 0.3 GeV



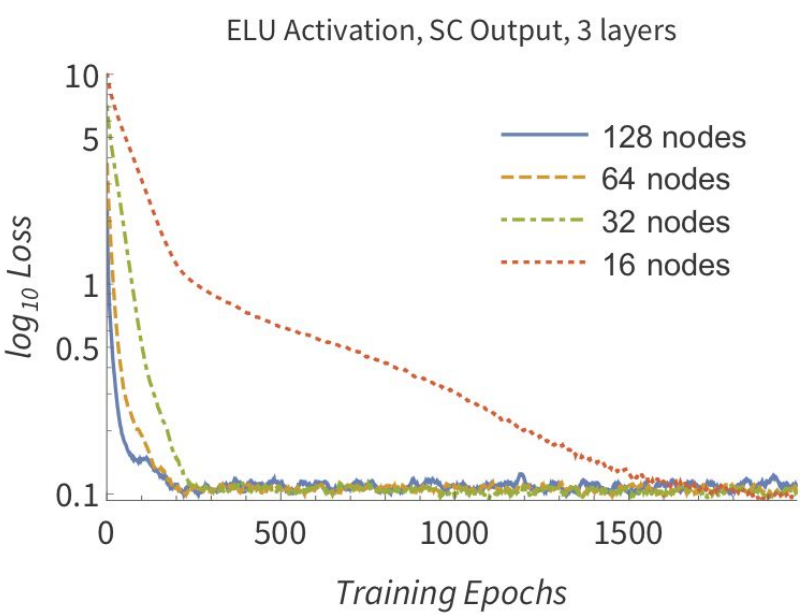


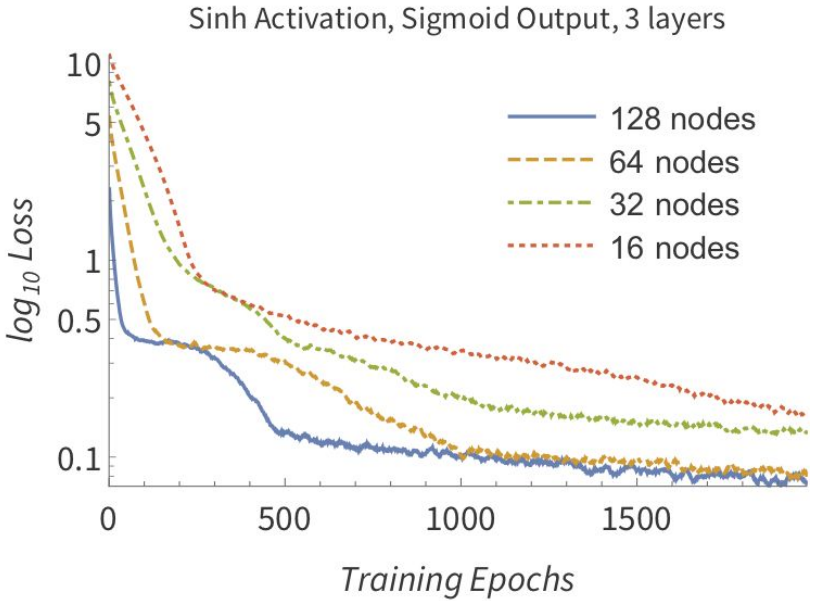
3-body Dalitz





3-body Dalitz





Based on these findings, we choose the ELU/SC architecture for all final results.

Our efficiency: 75%

MadGraph's efficiency (VEGAS-based): 6%