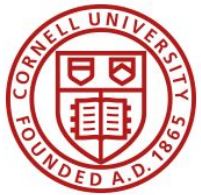
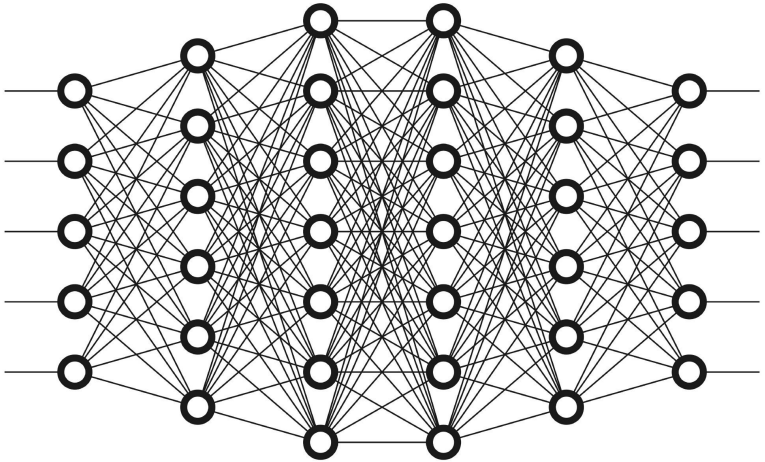
Event Generation with Neural Nets



Matthew D. Klimek
Cornell Univ.
Korea Univ.

Work with M. Perelstein
180?.xxxxx
Cambridge Seminar
27 April 2018



The general problem of MC integration/generation

Particle physics model → predictions of (differential) cross sections.

We can compare models to data by generating simulated data sets.

Simulated data:

- a set of points in phase space (events)
- distributed according to the probability density function (pdf) given by the (normalized) differential cross section.

Since the differential cross sections are typically complicated functions, Monte Carlo techniques are the only feasible way of handling these operations.

The basic technique

In the simplest case:

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

Estimates of the integral and the error are then obtained as:

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \qquad \sigma^2 \approx \frac{1}{N^2}\left(\sum_{i=1}^{N} f^2(x_i) - \left(\sum_{i=1}^{N} f(x_i)\right)^2\right)$$

For event generation, random points are drawn and the function is used to decide the probability of keeping the event (unweighting).

Probability of keeping event: $\epsilon(x) = \frac{f(x)}{f_{\max}}$

The basic technique

The error is large if the function is highly variable:

$$\sigma^2 \approx \frac{1}{N^2}\left(\sum_{i=1}^{N} f^2(x_i) - \left(\sum_{i=1}^{N} f(x_i)\right)^2\right)$$

If the function is highly peaked and variable, the unweighting efficiency will also be low:

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

Accurate integration/generation becomes computationally expensive.

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

Solution: Importance Sampling

Importance Sampling

Sample the function most often in the places where its values are largest.

- For integration: Concentrate samples on the regions that contribute most to the integral
- For generation: Draw points preferentially from regions which should have the most points.

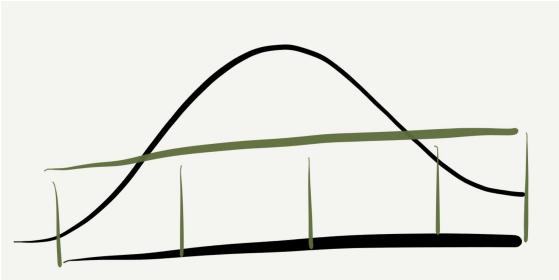
Implemented as an algorithm to build a function which:

- is easily sampled, and
- approximates the target function.

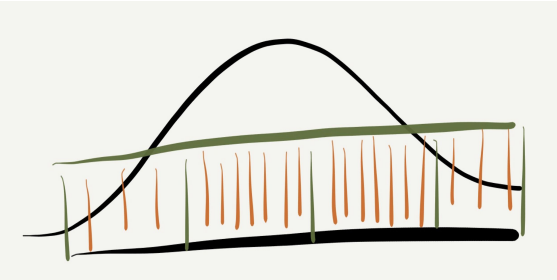
VEGAS

VEGAS (G.P. Lepage, *J.Comp.Phys.* 1978), is an importance sampling technique still in use today:

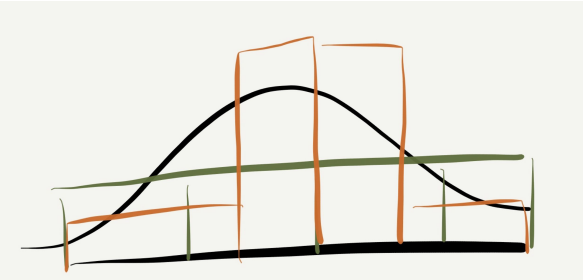
- Approximate the function by a *set of bins* of containing *equal amounts* of the integral of the function.
- To sample the function: simply *choose a random bin*; then *sample uniformly* within that bin.
- Adaptively choose bin edges to best match the function:



Sample the function as prescribed



Sub-divide bins proportional to function weight contained



Merge to obtain original number of bins

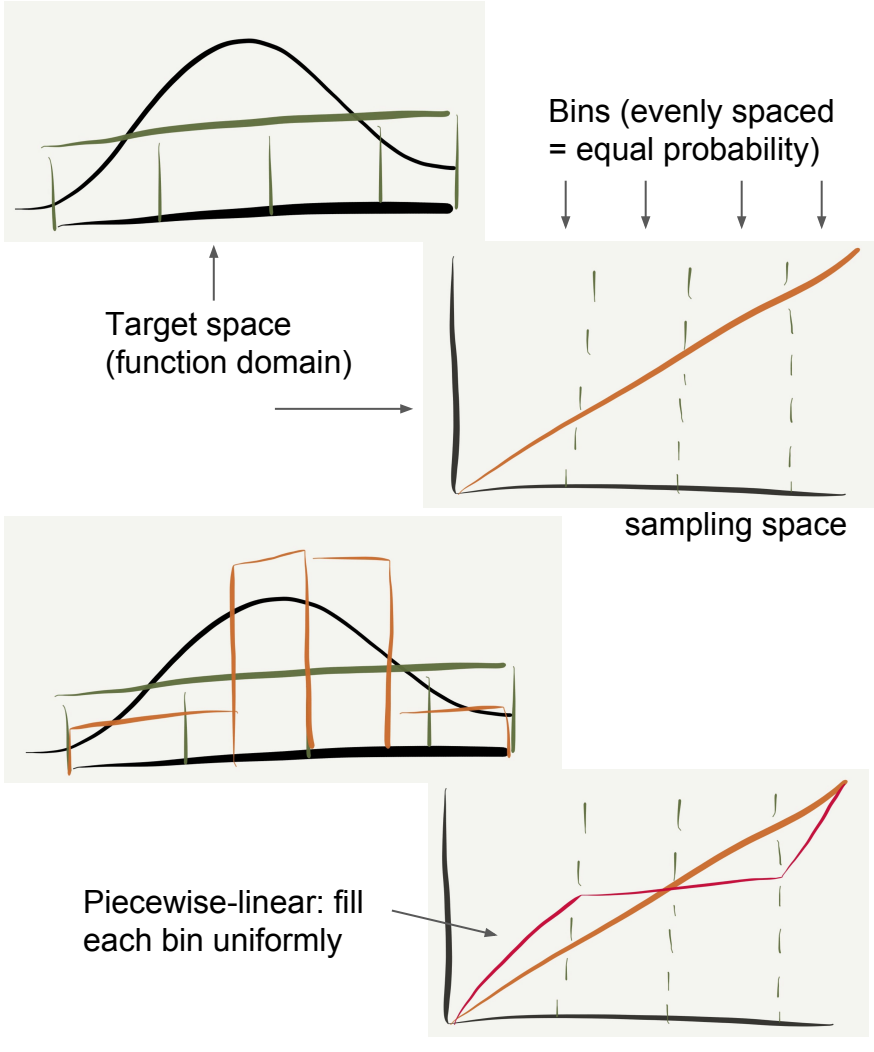
VEGAS as ML

VEGAS is a form of machine learning.

The algorithm goes through a *training process* where it is allowed to *adjust* the location of the bin edges, in order to *decrease the variance* in the number of points that land in each bin.

It builds 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 as ML

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.

Universal approximation theorem: Given any continuous function *f(x)* on the *N*-cube, and any $\epsilon > 0$, *f(x)* can be approximated by a function *F(x)*

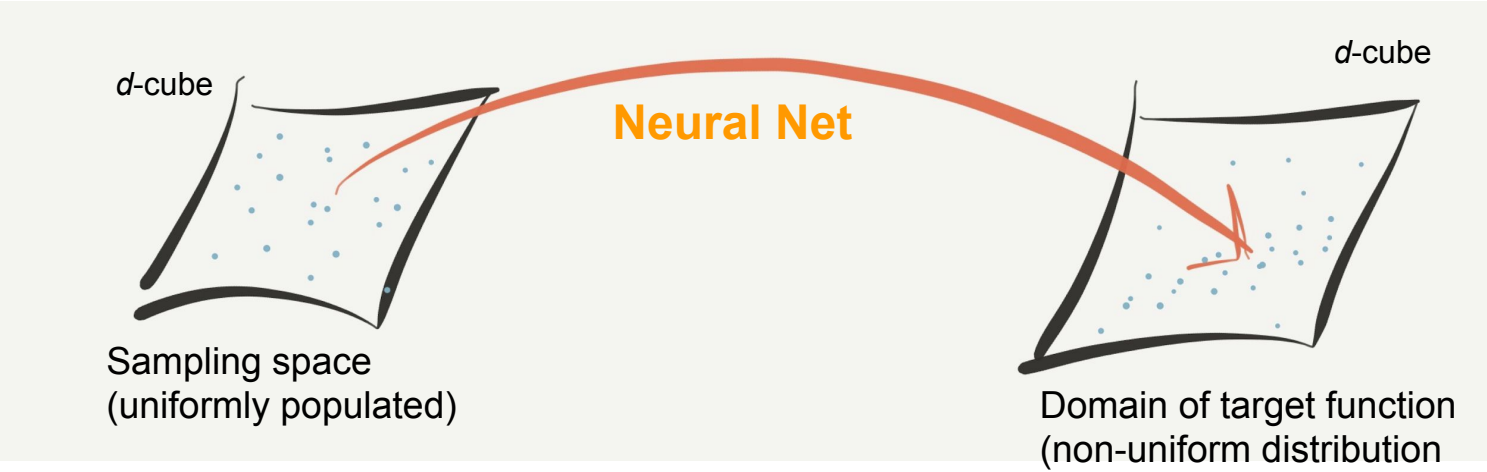
$|F(x) - f(x)| < \epsilon$ where $F(x) = \sum_i v_i A(w_{ij} x_j + b_i)$

and *A* is a ~non-linear function.

This is the basis of artificial neural nets.

General Approach

First suggested by Bendavid 1707.00028, applied only to Gaussian functions



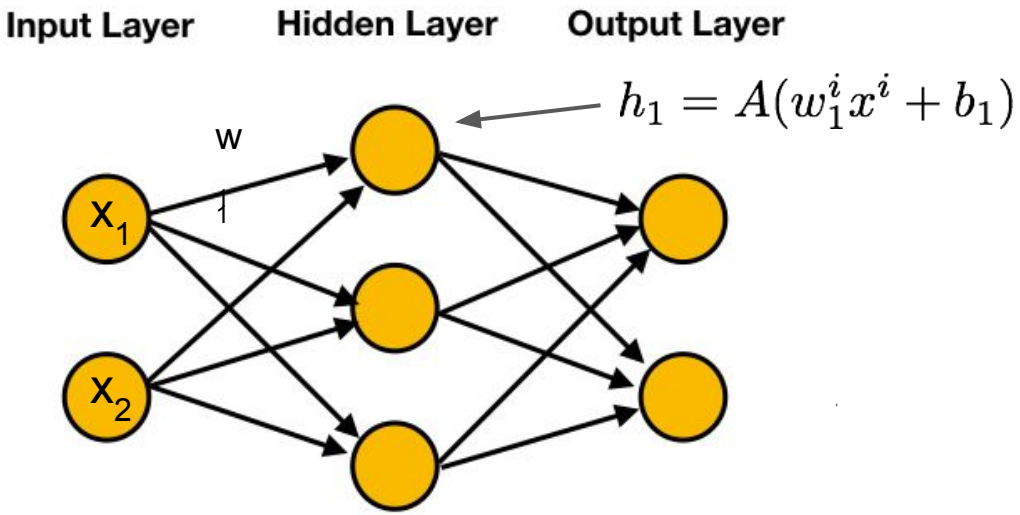
Choose a measure of statistical distance between the true and induced: *Kullbeck-Leibler divergence,* which is zero for two distributions *f* and *g* only when *f* = *g* and is positive otherwise.

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

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

The algorithm should adjust *NN* so that the D_{KL} between *1*/Jac(*NN*) and the differential cross section is minimized (ideally to zero).

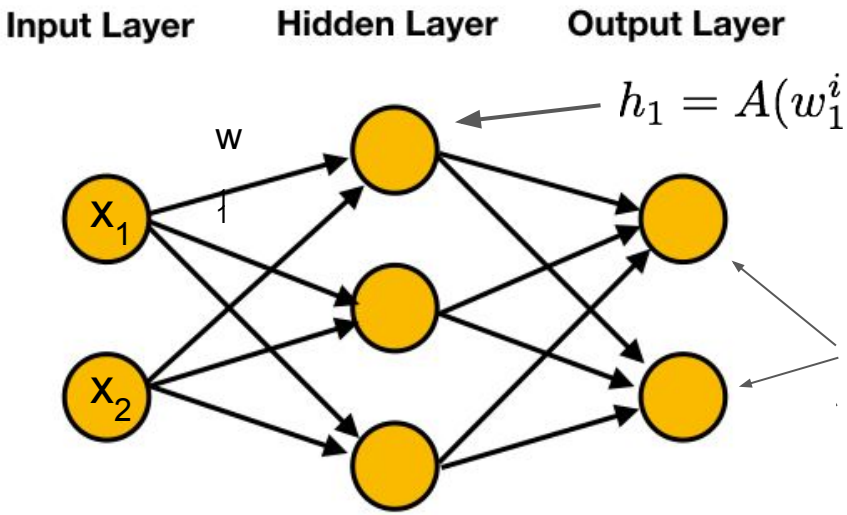
NN Basics



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

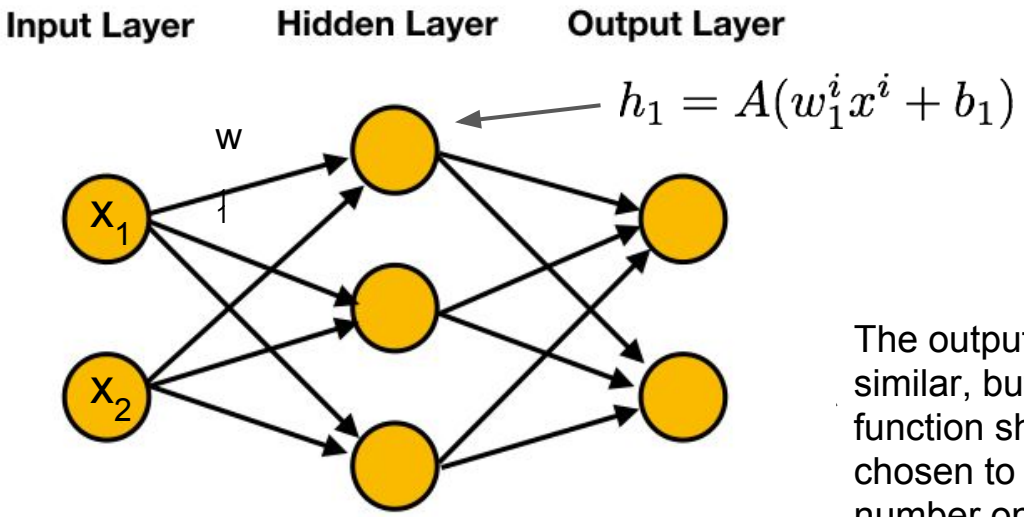


Each hidden node takes a linear combination of the inputs, specified by the *weights* w_1^i plus a constant *bias* b_1, and transforms it by some non-linear *activation function A.*

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 the net is from the desired behavior (KL divergence for us).

Train by adjusting each parameter proportionally to the gradient of the loss function w.r.t. that parameter (gradient descent).

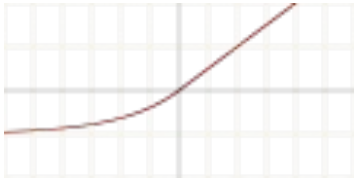
Our basic implementation

- Same number of input and output nodes ($\mathbf{R}^d \rightarrow \mathbf{R}^d$).
- Number of hidden nodes and hidden layers to be determined by studying performance.
- Loss function will be K-L divergence of the net's Jacobian with respect to the target differential cross section, as described earlier.

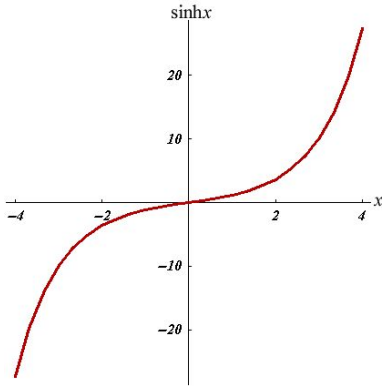
We consider two candidates for the activation functions:

"Exponential Linear Unit" with α = 1:

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



or Sinh:



Choice of coordinates

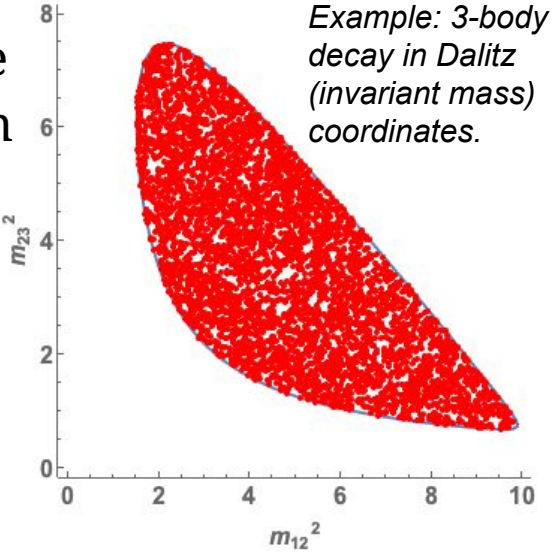
In principle, any choice of coordinates on phase space could be used, but many choices present difficulties in practice.

In many coordinate systems, the physical region of phase space has some non-trivial shape.

The net would need to learn:

- ➢ the correct distribution within the physical region
- ➢ but also where the boundary is, and to not populate anything outside (unphysical events).

However, the net is made of smooth functions so such behavior is not possible.



Example: 3-body decay in Dalitz (invariant mass) coordinates.

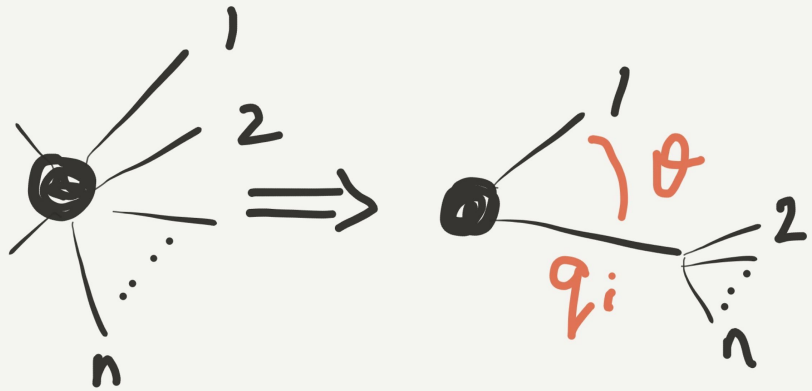
Choice of coordinates

Solution: use coordinates in which the physical region is a unit hypercube, and an output function that maps $\mathbf{R}^n$ onto it.
Then all possible outputs of the net are guaranteed to be physical.

Example: Define $q_i \in [0,1]$ to interpolate between the minimum and maximum possible invariant masses of the system composed of particles {i+1, …, n}.

Rescale all relative angles to the range [0,1].

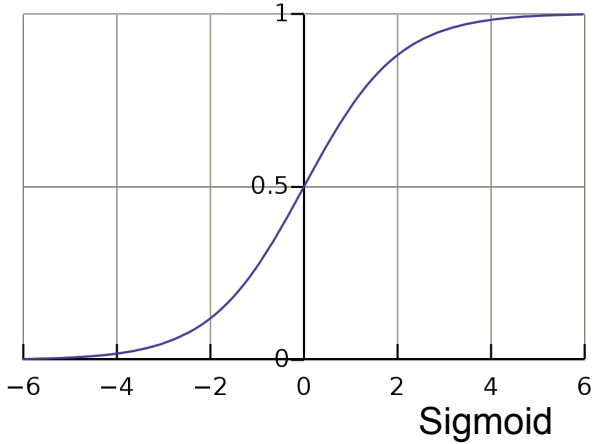
$$q_1 \leftrightarrow \left[\sum_{i=2}^{n} m_i, \sqrt{s} - m_1\right]$$

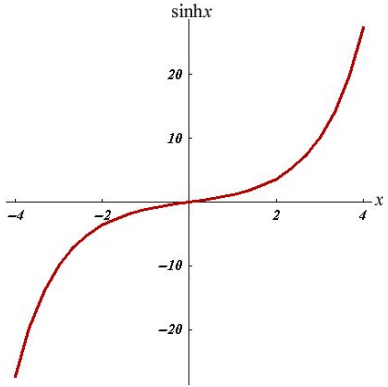


Output layer considerations

The internal layers of the NN may return any real values, so the output layer should contain a final function that maps onto the unit interval.

A common choice of output function is the Sigmoid, but it approaches 0 and 1 exponentially slowly, making it very hard to populate the edges of phase space.





For the sigmoid output function, we use a sinh activation function.

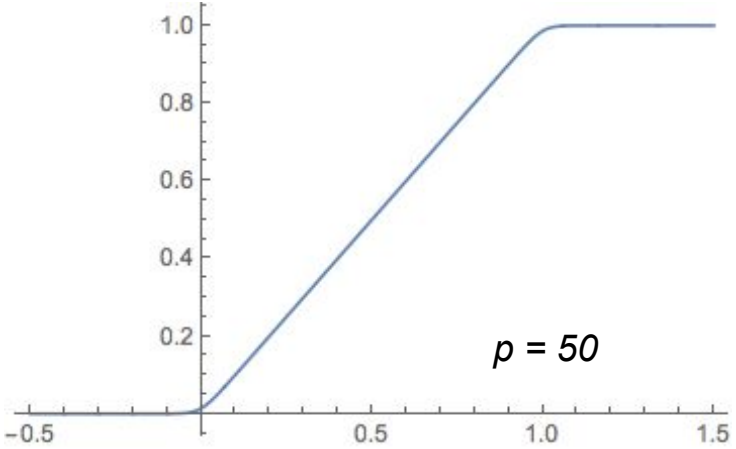
The asymptotically exponential behavior of these functions cancels and allows good reach to the edges 0 and 1.

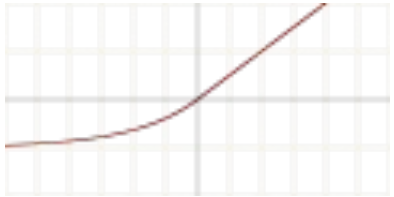
Output layer considerations

We also investigated a function with faster asymptotic behavior:

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

- Always takes values in [0,1]
- Approaches limiting values rapidly
- Approximately linear between [0,1]
- *p* controls how sharp the edges are





For this “custom” activation function which approaches 0 and 1 rapidly, a traditional ELU activation function is sufficient.

Complete setup

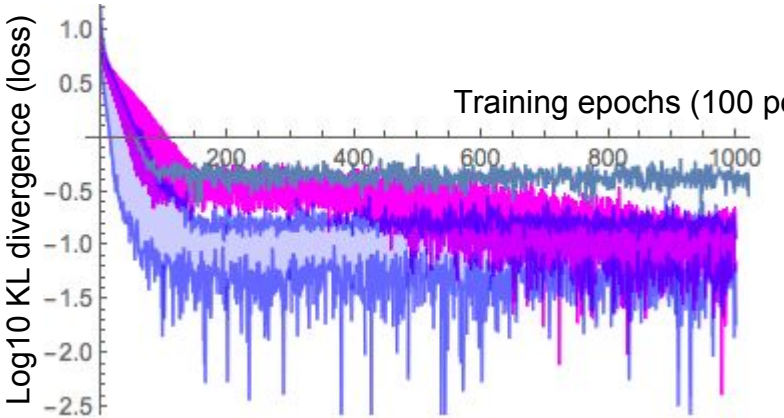
- Implemented in MXNet 1.1.0 with Python 2.7 interface.
- Various numbers of hidden nodes and layers tested, as well as different activation/output functions.
- Physics input: simply type analytical expression for the differential cross section into the code, or provide a function that python can call. (Easy to interface with Feynman diagram calculator.)
- Training:
  - Draw a sample of 100 uniform random points.
  - Feed through the NN.
  - Compute the KL divergence between the NN output and the target diff. cross section.
  - Try to minimize KL divergence by adjusting the NN parameters according to the gradient of the KL divergence (gradient descent).
- Train until value of KL divergence stabilizes.
  - (How close to 0 does it get?)

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

Training results with 3 layers of 64 nodes. Training to stability takes ~1 minute on a very old laptop.

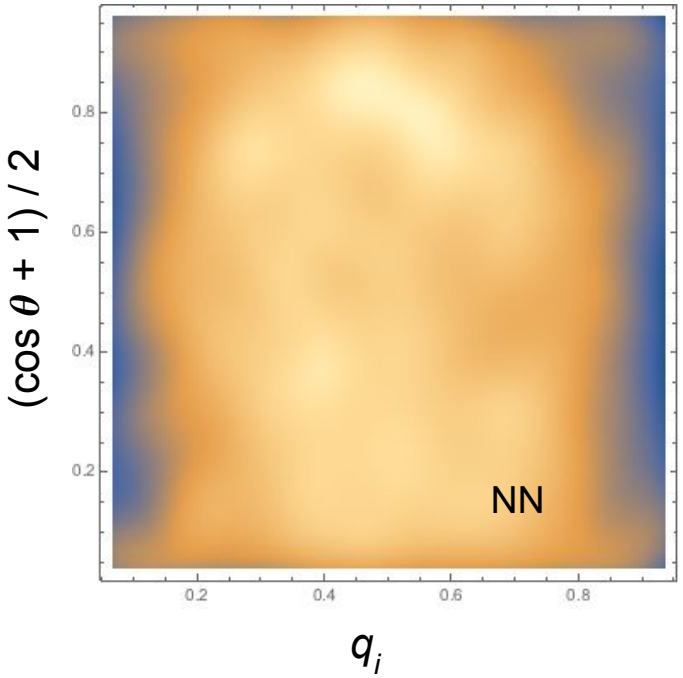


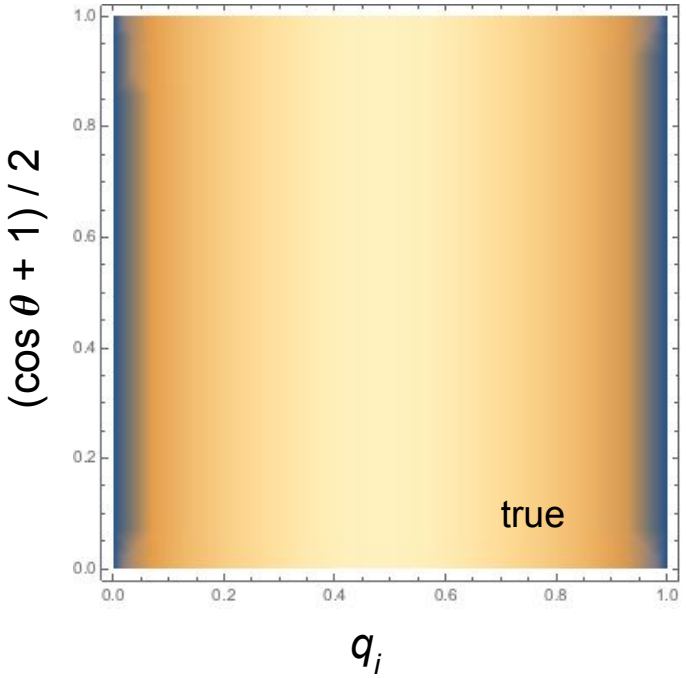
Sinh/Sigmoid
ELU/Custom activation
Bendavid 1707.00028

This is in fact more than is needed for this simple example. 3 layers of 4 nodes, or one layer of 16 nodes is sufficient and training takes ~seconds. See forthcoming paper.

3-body Dalitz

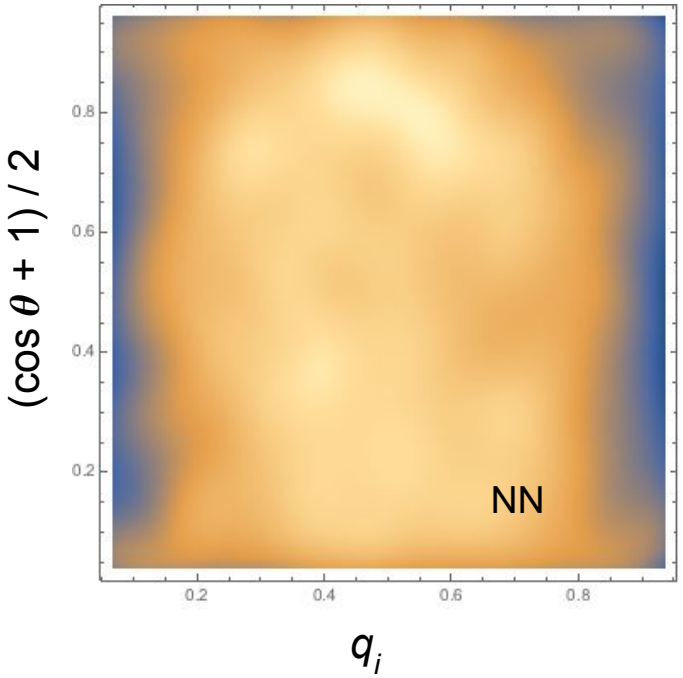
Density of events generated by trained network vs. true distribution:





3-body Dalitz

Is the NN output by itself consistent with the desired true distribution?



No: *p*-value ~ 10^{-4}

But a NN is a universal approximator. What happened?

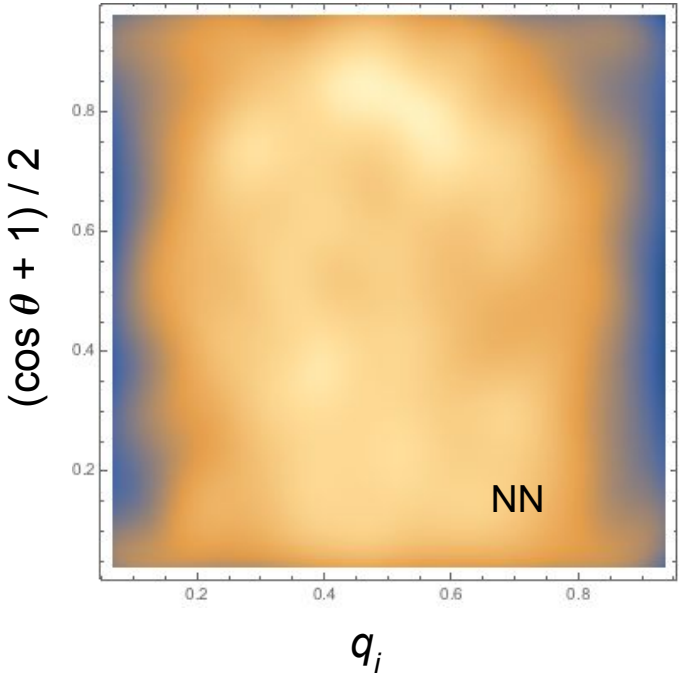
The parameter space of the NN is very high dimensional, and there are many local minima of the loss function (KL divergence.)

In general, one lands in a local minimum which is a good, but not great, approximation to the desired function.

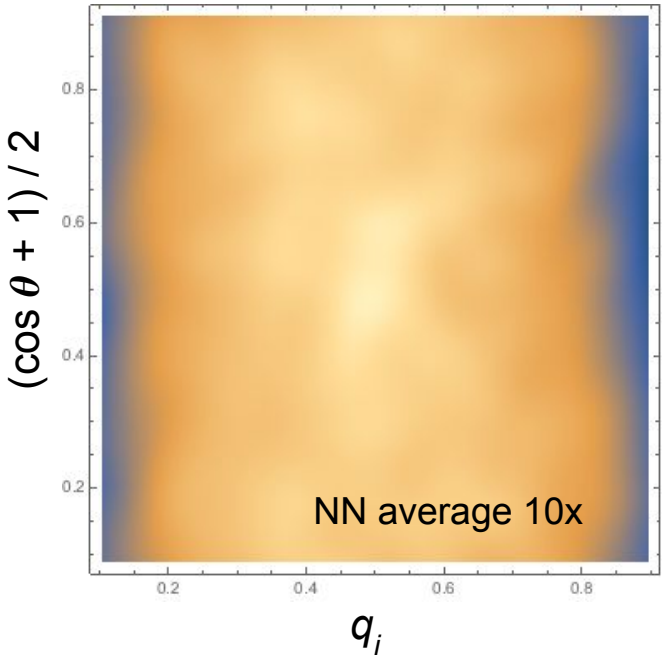
Not a result of over-training: new samples are drawn for each training epoch.

3-body Dalitz

What to do about errors from "pretty good" false minima?



- Unweighting: NN is already pretty good so unweighting is quite efficient
- Average: each false minimum is a ~random deviation from the right answer. Train multiple times with different random seeds and combine results:

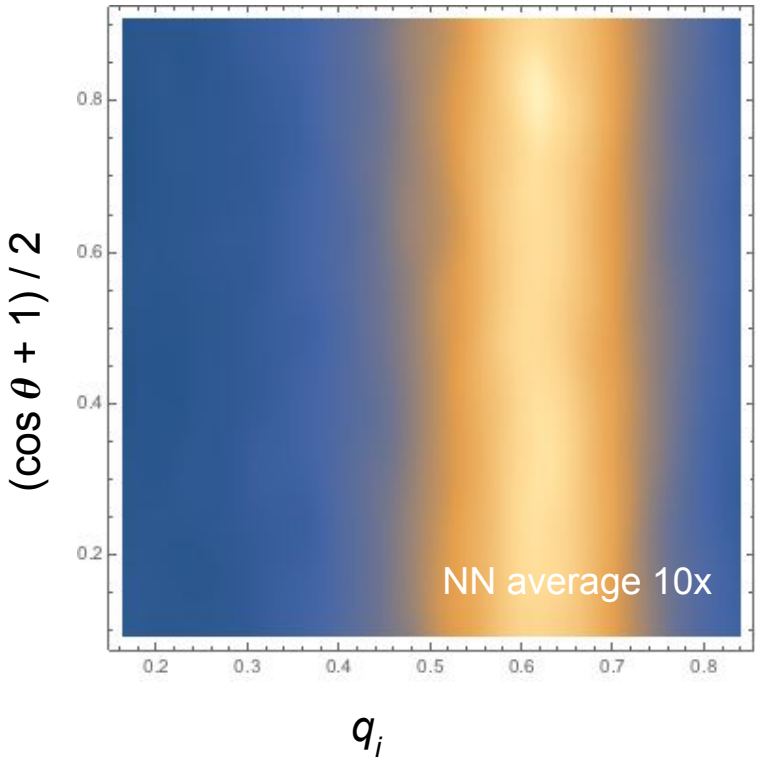


p = O(1) with no unweighting (100% efficiency)

3-body Dalitz with intermediate resonance

We can include a matrix element along with phase space:

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



p = O(1)

Contrast with VEGAS (rectangular grid)

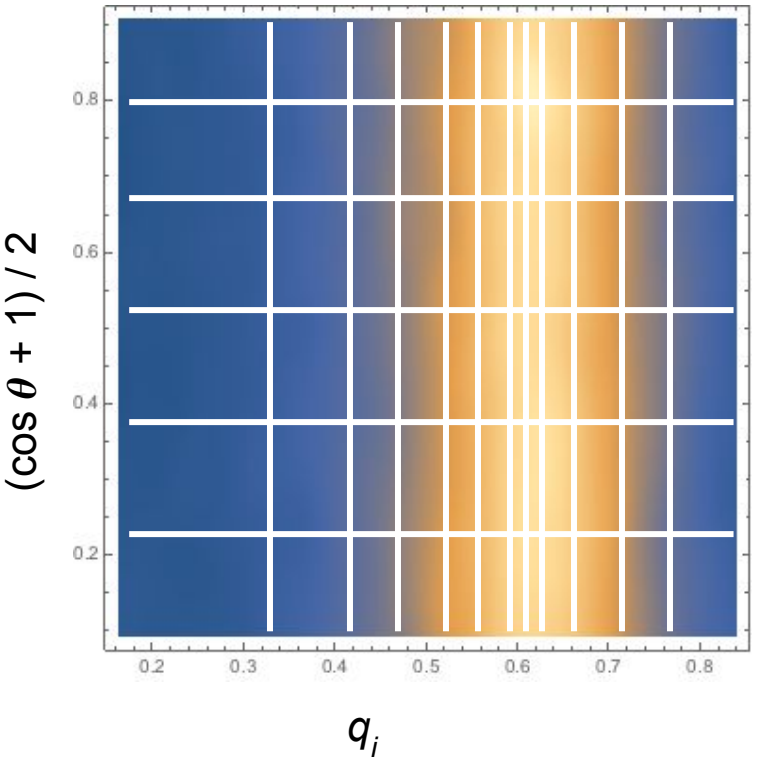
For multi-dimensional integrals, VEGAS needs any sharp feature to be aligned with a grid axis.

What about matrix elements that have multiple, non-orthogonal sharp features? (*E.g.* multiple resonances)

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

Define multiple grids, each aligned with one feature, and sample from all.

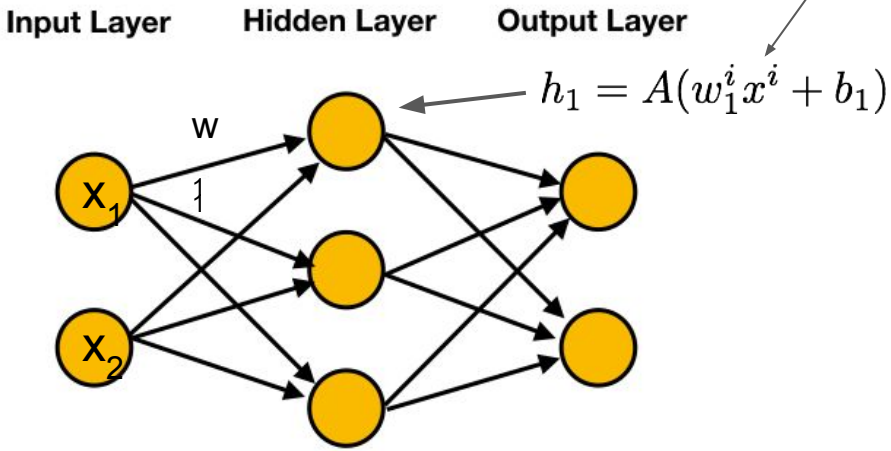
Potential slow, and relative weights among grids must be tuned.

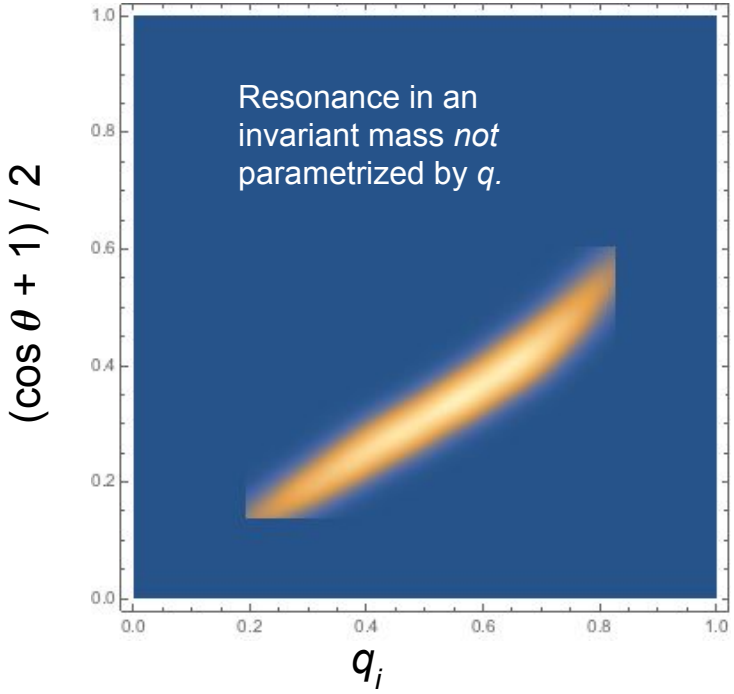


Contrast with VEGAS (rectangular grid)

Our Neural Net approach has no intrinsic axes.

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





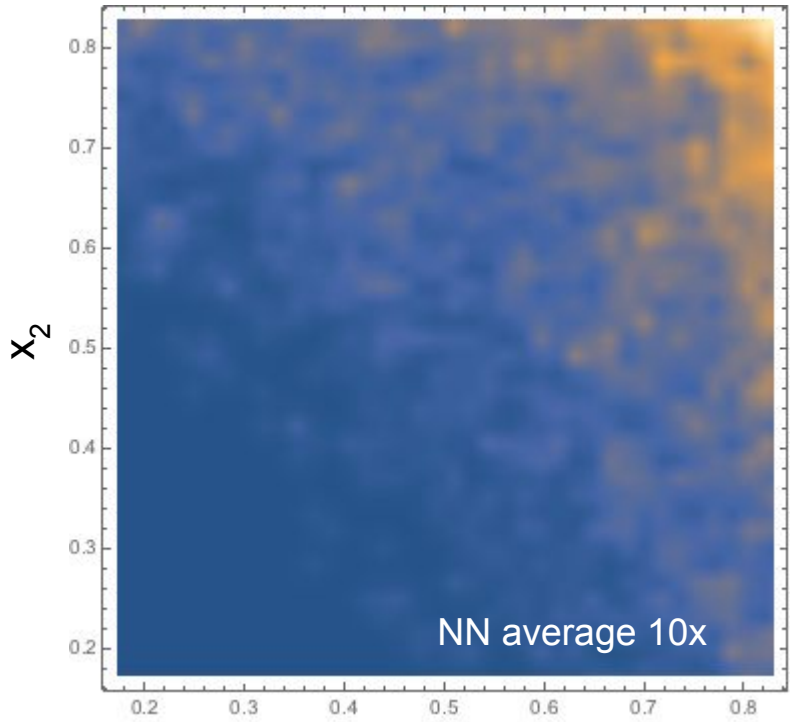
The Neural Net handles features well in any coordinate system.
Indeed it *learns* what the most interesting coordinates are.

qqg

The NN also has no trouble handling singularities such as are present in gluon emission from quarks. (Singularities at quark momentum fraction *x = 1.)*

An appropriate kinematic cut can be specified in the code.

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



Cut on quark momentum fraction (minimum gluon energy):
$x < 0.999$

Summary

- Neural Networks allow a continuum implementation of the classic VEGAS phase space integrator/generator.

- Proper choices of network architecture allow typical physical scenarios to be handled accurately and quickly.

- We have presented several prototypical physical scenarios that can be handled by this method.

- Averaging a set of neural nets gives a MC generator that can provide unweighted events with O(1) efficiency.