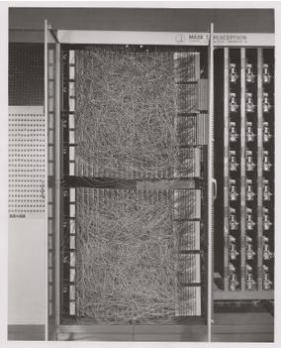
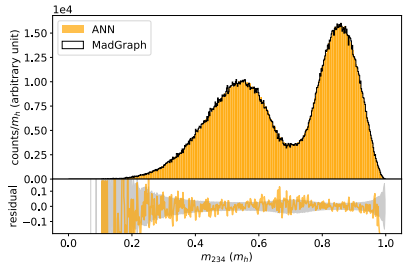


# Monte Carlo Simulations with Neural Networks




Part 1: Maxim Perelstein, Cornell
Part 2: Christina Gao, Fermilab

KITP "Precision" Workshop, May 13 2021

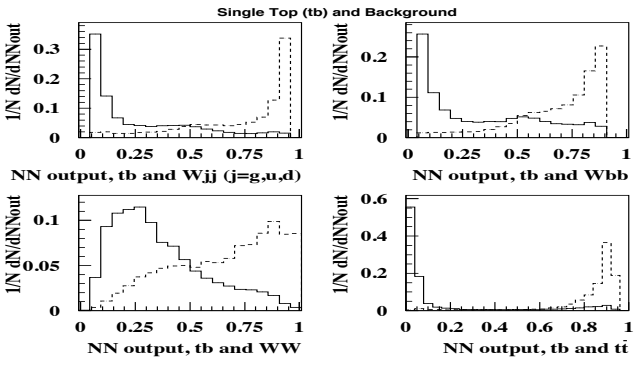
CORNELL UNIVERSITY LEPP
LABORATORY FOR ELEMENTARY-PARTICLE PHYSICS

NSF

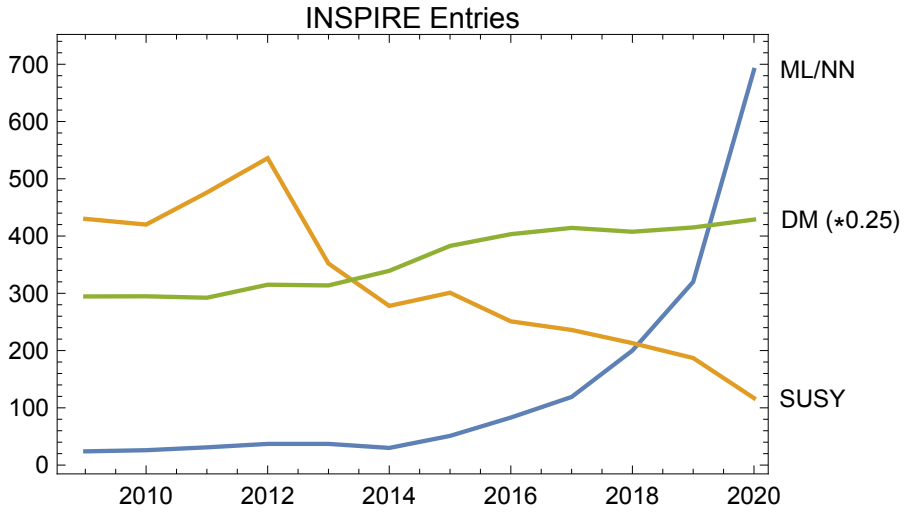
CORNELL UNIVERSITY FOUNDED A.D. 1865

# ML/NN in Particle Physics

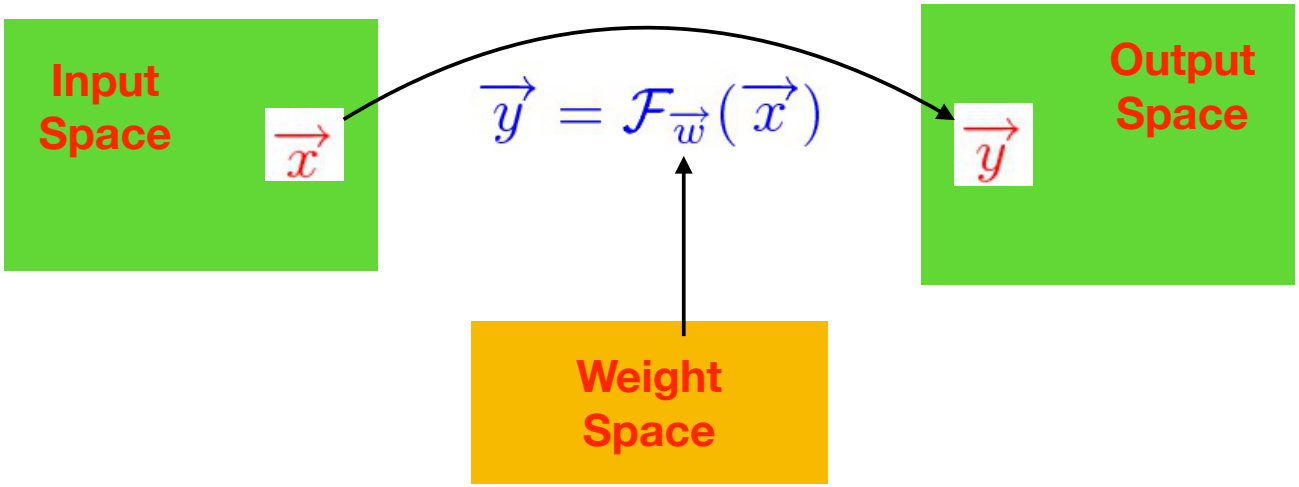
- Long history of NN uses in particle physics, e.g. track reconstruction and combining observables in early top-physics discoveries (1990's)
- Exponential explosion of interest in machine learning and neural networks since 2015, driven by advances in computing and algorithms
- Many applications to classification problems (e.g. jet flavor tagging) and anomaly detection
- We will discuss another application: improving efficiency of Monte Carlo simulations




Multi-variate analyses, using high-level inputs
e.g. D0 single-top search + discovery, 1999-2007



# ML in a Nutshell

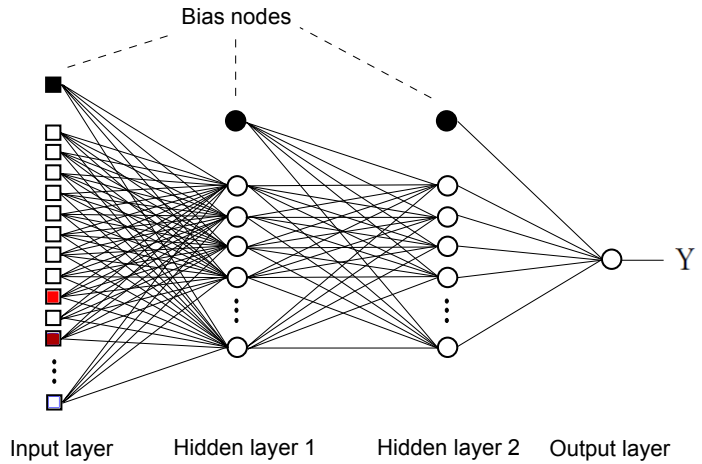



- Start with a “complete” set of maps $\mathcal{F}$, parametrized by “weights” $\vec{w}$
- Set the goal: Define “loss functional” (LF) $L[\mathcal{F}]$
- “Training”: Find the best among all possible maps w.r.t. chosen LF

$$\min_{\vec{w}} L[\mathcal{F}]$$

- “Machine Learning” = numerical algorithms that solve this problem

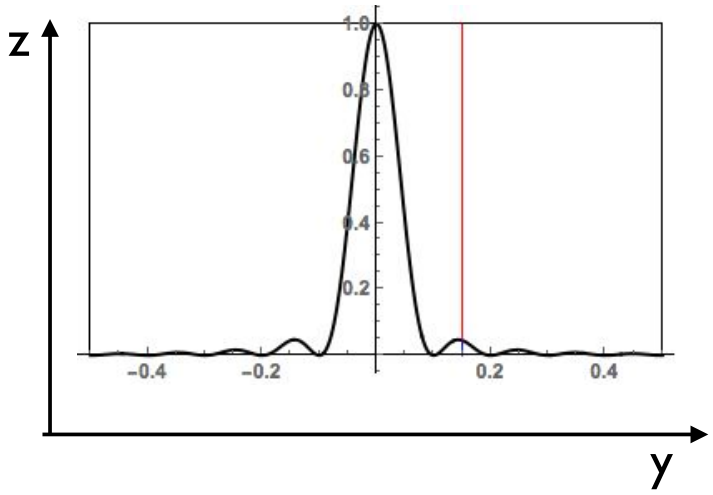
# Neural Networks




- NNs are a set of functions defined recursively: $h_i^{(l)} = f(w_{ij}^{(l)} h_j^{(l-1)})$
- Any map can be approximated by a sufficiently large NN ("universal approximation theorems") → completeness
- Efficient training algorithms make large networks computationally practical, user-friendly packages (TensorFlow, MXNet) make it fun
- Many physics problems can benefit from this technology!

# MC Simulation/Integration

- Monte Carlo Problem: Given a function f(y), such that $f(y) \geq 0$, generate a set of "random" points {y_i} with density proportional to f(y).
- In particle physics, typically y=phase space points, f(y)=differential cross section or decay rate, {y_i}=Monte Carlo sample ("pseudo-experiment")
- Most Naive MC algorithm: randomly select points in 2D box, discard the points with z > f(y).
- Fraction of points that are actually used = "unweighting efficiency": $\epsilon(y) = \dfrac{f(y)}{f_{\max}}$




integration: $\int f(y)dy = f_{\max} \int \epsilon(y)dy$

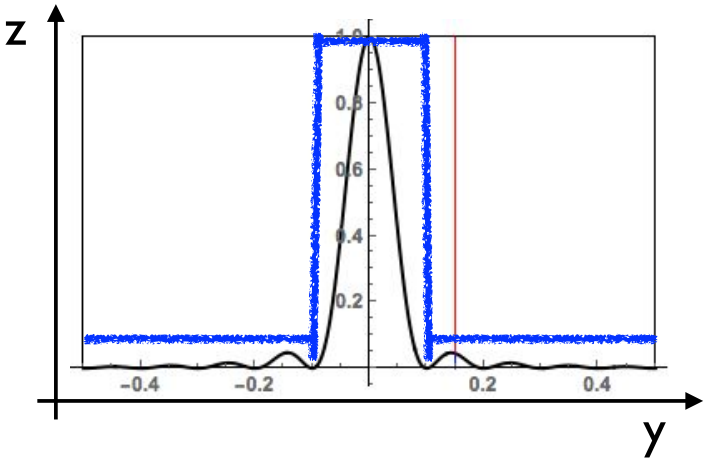
Problem: Resonances, Collinear/Infrared Singularities

$\epsilon \ll 1$

In modern applications, $f(y)$ is often numerically expensive to evaluate (e.g. NNLO - may require numerical integrations)

# Importance Sampling

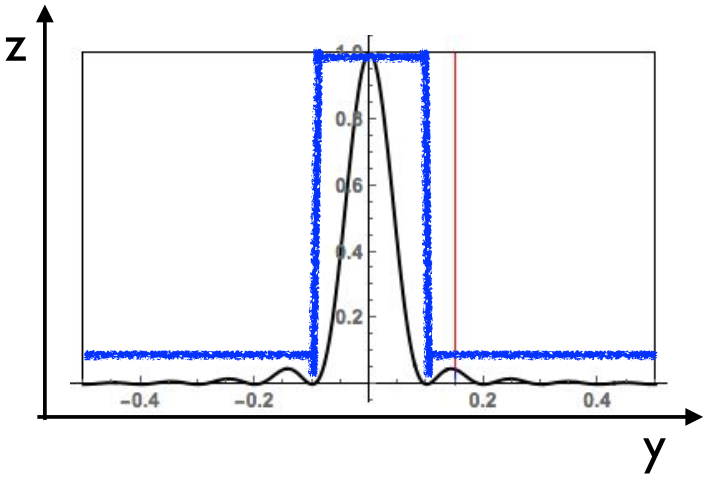
- Classic solution: construct a number of "bounding boxes" in yz plane, covering the function's domain, with heights adjusted to correspond to local values of f(y)
- Classic implementation: VEGAS [Lepage, 1978] (inside MadGraph, etc.)
- Divide the domain into N bins, roughly compute "weight" = $\int_{\text{bin}} \epsilon(y)dy$ in each bin
- Iteratively adjust bin boundaries until each bin contains the same weight
- Simulation: choose a bin at random (equal probabilities), then follow Naive algorithm in that bin. Repeat.

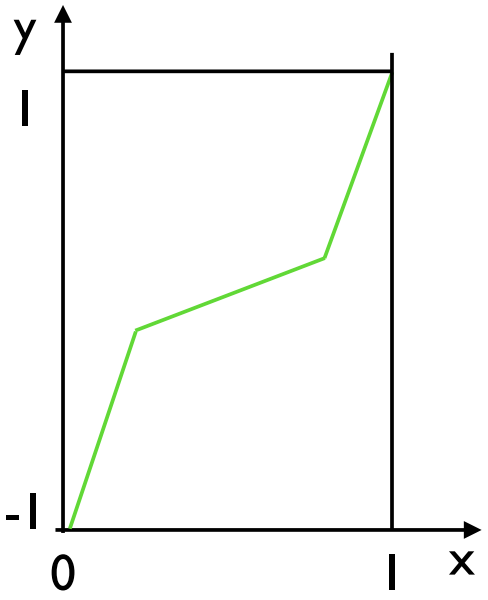



Construct a piecewise-constant approximation to f(y), then sample from that distribution

# Importance Sampling as a Map

- Importance sampling can also be described as a map from "input space" x to "target space" y
- Randomly choose $x \in [0, 1]$ (uniform distribution)
- Deterministic, piecewise-linear map $x \to y(x)$
- Equivalent to "pick a box + random point within the box"
- Unweighting: keep the point with probability $P(y) = f(y) \left| \frac{dy}{dx} \right|$



# MC with Neural Networks

- Idea: Generalize importance sampling from piecewise-linear to nonlinear maps
- Simulation would be 100% efficient if we found a nonlinear map such that

$$\left|\frac{dy}{dx}\right|^{-1} = f(y)$$

- Generalization to functions in N dimensions (same dimensionality for input and target spaces, =dimensionality of phase space)

$$J = \det \frac{\partial y_i}{\partial x_j}, \qquad |J|^{-1} = f(y)$$

- Universal Approximation Theorem: under mild assumptions, a neural network can approximate any continuous functional map $\mathcal{I}_N \to \mathcal{I}_N$ (where $\mathcal{I}_N$ is an N-dimensional hypercube)

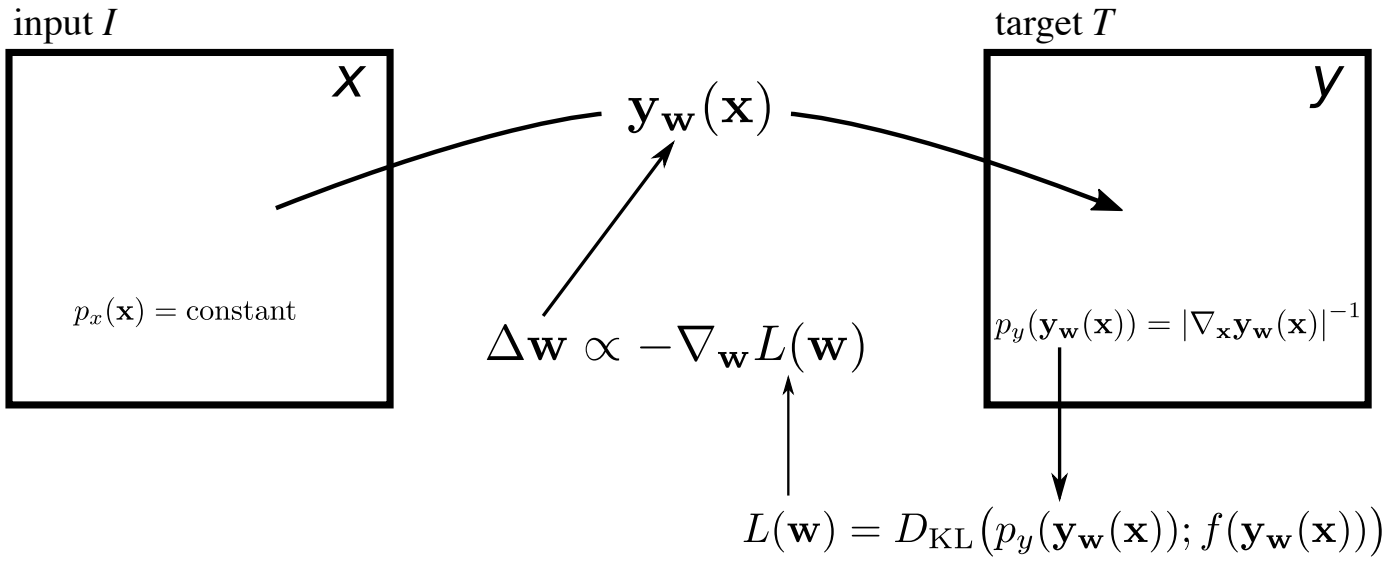
[Cybenko, '89; Hornik, '91]

- This makes a NN a natural choice to implement nonlinear importance sampling

[J. Bendavid, '17; M. Kilmek and MP, '18]

# MC with Neural Networks

[M. Kilmek and MP, 1810.11509, SciPost Phys]



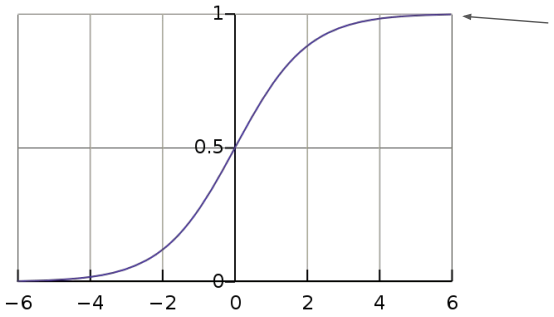

- T = N-particle phase space; can choose coordinates so that T = a unit hypercube for any N (map from 4-momenta to these coordinates is in our paper)
- $f(y)$ = matrix element-squared (computed separately)
- Our goal is a "first principles" simulation, as opposed to bootstrapping with e.g. GAN approach
- Hope that ultimately NN-based algorithm replaces VEGAS inside standard tools

# MC with Neural Networks

- Use classic fully-connected NN (fancier architectures left for future study)
- 3*128 or 6*64 hidden nodes
- An important subtlety is the choice of output function (=activation function for the last layer)

sigmoid:

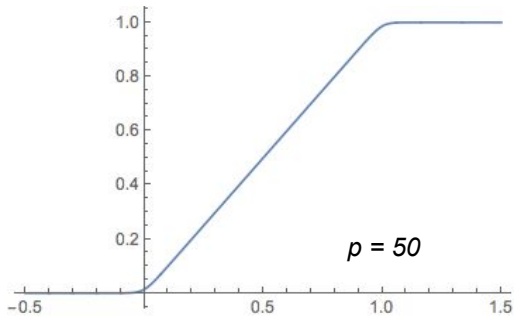
$$S(x) = \frac{1}{1 + e^{-x}}$$




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

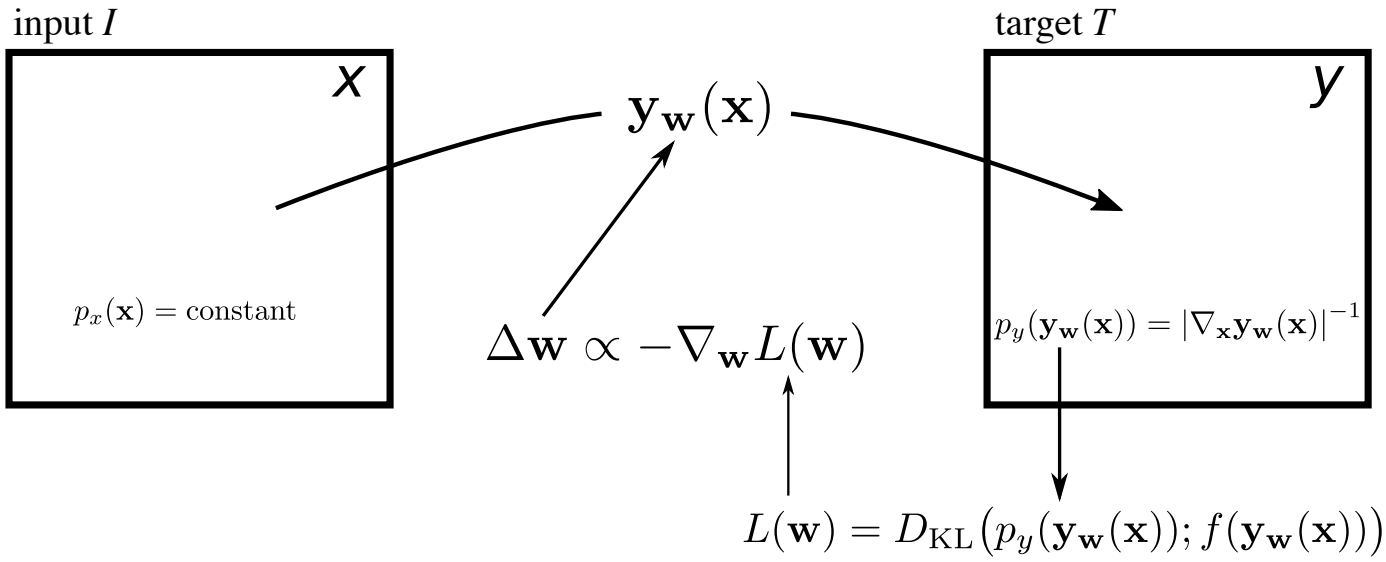
"soft clipping function":

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



# MC with Neural Networks

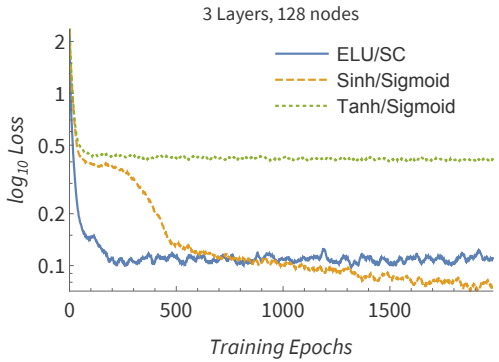
[M. Kilmek and MP, 1810.11509, SciPost Phys]




- Error function: Kullbeck-Leibler divergence between $|J|^{-1}$ and $f(y)$ :

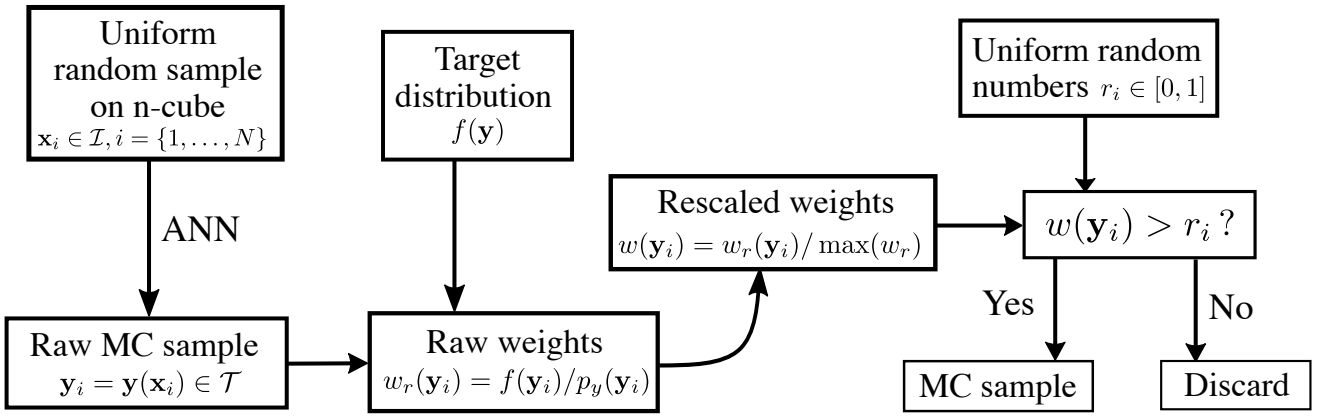
$$D_{\mathrm{KL}}[p_y(\mathbf{y}); f(\mathbf{y})] \equiv \int p_y(\mathbf{y}) \log \frac{p_y(\mathbf{y})}{f(\mathbf{y})}\, d\mathbf{y}$$

- Training: generate a batch of 100 points, compute $D_{KL}$ , adjust weights, iterate



# MCNN Event Generator

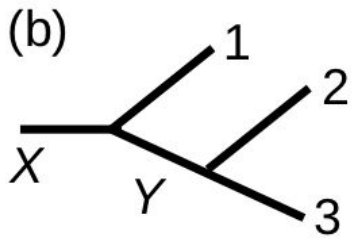
[M. Kilmek and MP, 1810.11509, SciPost Phys]

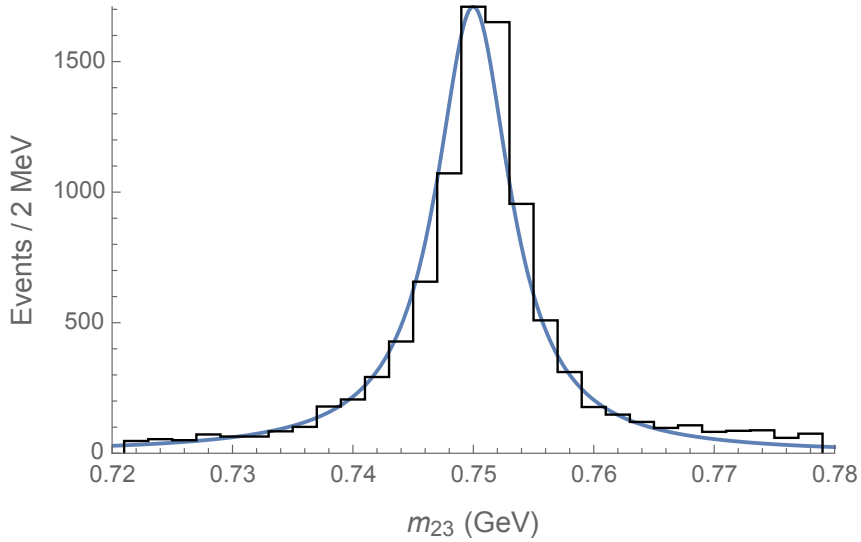



- Unweighting procedure: start with a raw sample produced by trained NN and discard events to obtain a "perfect" distribution, at the expense of reduced sample size
- Unweighting efficiency is a measure of "wasted" events; 100% if NN map is already perfect
- We use unweighting efficiency as a measure of success

# Sample Applications

- Simulate 3-body decay of a scalar X, with a resonance Y

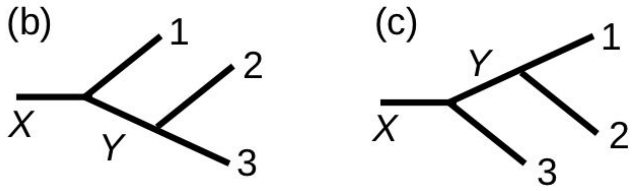



- Choose phase-space coordinates $m_{23}, \theta_{1(23)}$
- Simulated with $\Gamma_Y/m_Y = 10^{-2}, 10^{-3}, 10^{-4}$
- Achieved unweighting efficiency 30-70%, depending on resonance width
- MadGraph (off-the-shelf) efficiency: 6%

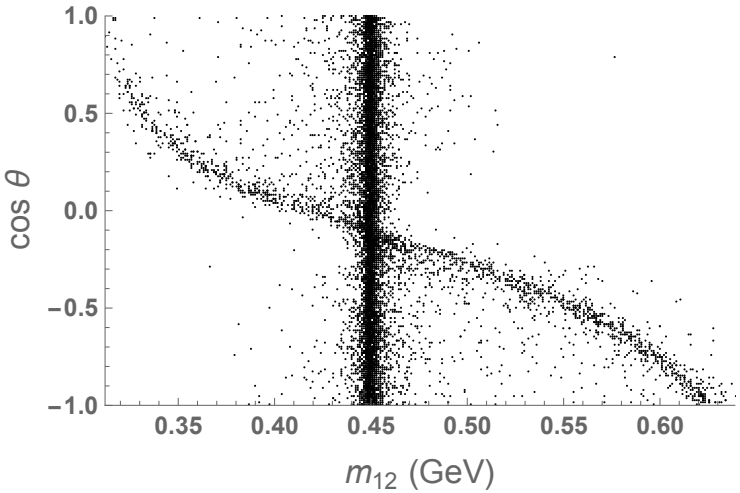
# Sample Applications

- Simulate 3-body decay of a scalar X, with resonances in two channels

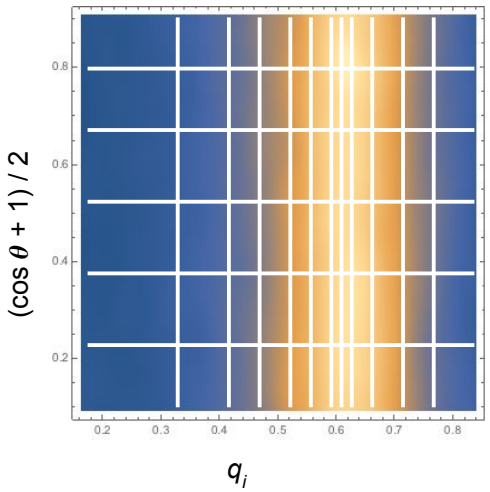



- NN was able to learn both the feature aligned with coordinate axis, and the feature with complicated shape in these coordinates

- In contrast, VEGAS needs each feature to be aligned with a coordinate axis (coordinate choice handled separately by "multi-channeling")




NN output




VEGAS grid/output

# Sample Applications

- A more realistic example: $e^+e^- \to q\bar{q}g$

$$\frac{d\sigma}{dm^2_{qg}dm^2_{\bar{q}g}} \propto \frac{(s-m^2_{qg})^2+(s-m^2_{\bar{q}g})^2}{m^2_{qg}m^2_{\bar{q}g}},$$

- Soft/collinear singularities → need to impose kinematic cuts
- Simple rectangular cuts aligned with target-space coordinates can be simply handled by redefining the target space boundaries
- In practice we need to be able to handle more general cuts:

$$Y \geq Y_{cut} \quad \text{where} \quad Y = Y(y_1, \ldots, y_N)$$

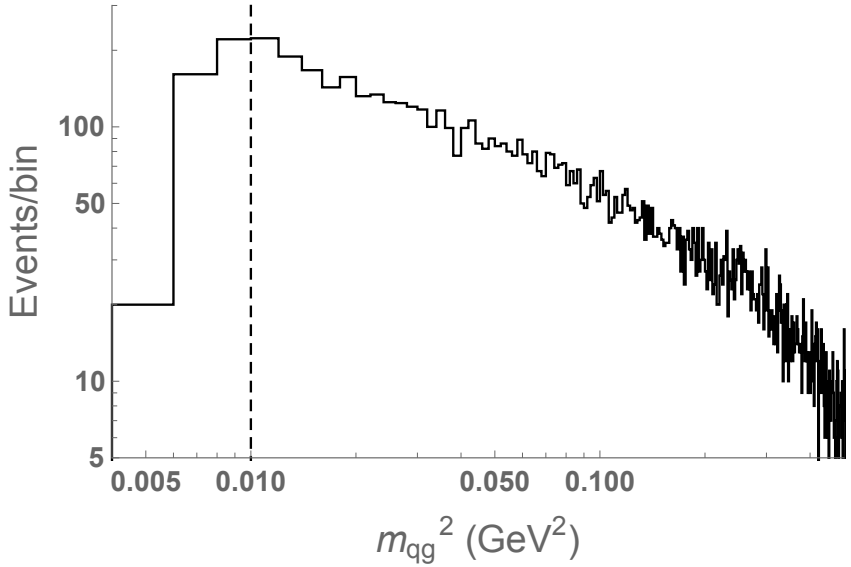
- Naively, we could just replace $f(\mathbf{y}) \to \theta(Y(\mathbf{y}) - Y_{cut})\, f(\mathbf{y})$
- However NN target function must be differentiable! So we opt for

$$f(\mathbf{y}) \to \kappa(Y(\mathbf{y}) - Y_{cut})\, f(\mathbf{y}) \qquad \text{with} \qquad \kappa(x) = \begin{cases} 1 & x > x_{\text{cut}} \\ (x/x_{\text{cut}})^n & x < x_{\text{cut}} \end{cases}$$

# Sample Applications

- A more realistic example: $e^+e^- \to q\bar{q}g$

$$\frac{d\sigma}{dm_{qg}^2 dm_{\bar{q}g}^2} \propto \frac{(s-m_{qg}^2)^2 + (s-m_{\bar{q}g}^2)^2}{m_{qg}^2 m_{\bar{q}g}^2},$$

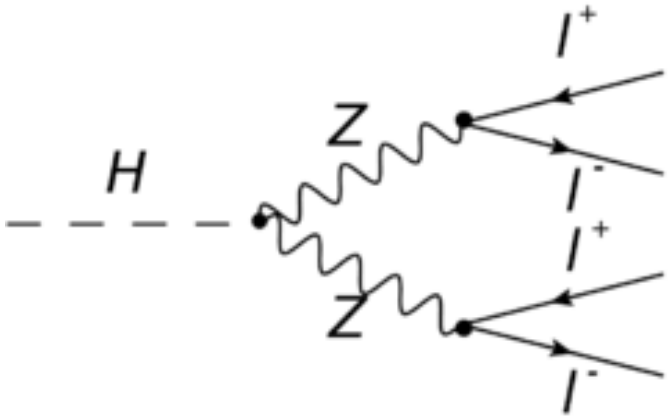


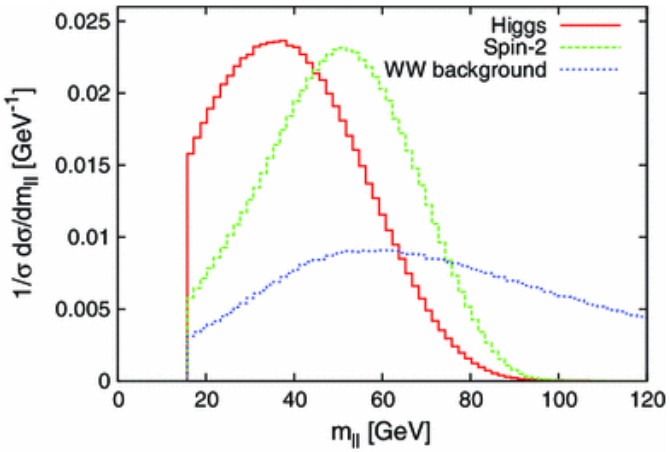

- In this example, we used n=8.
- Unweighting efficiency is 70% (vs. 4% for off-the-shelf MadGraph)

# Leptonic Higgs Decay

[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Most interesting parton-level processes involve large # of final-state particles → # of phase-space coordinates → size of input/output spaces
- We want to explore how the NN approach can handle larger phase spaces
- Picked an example of great interest at the LHC, Higgs decay to 4 leptons
- Non-trivial resonance structure: typically 1 on-shell and 1 off-shell Z/W in each event
- Distributions carry information about Higgs spin/CP



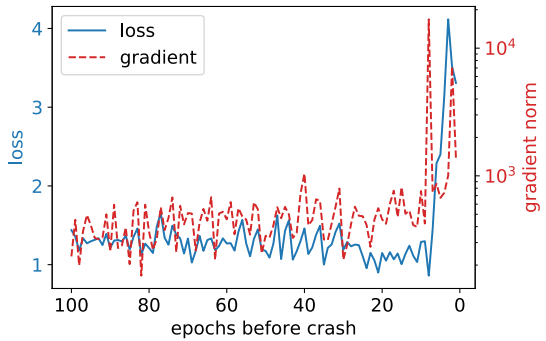



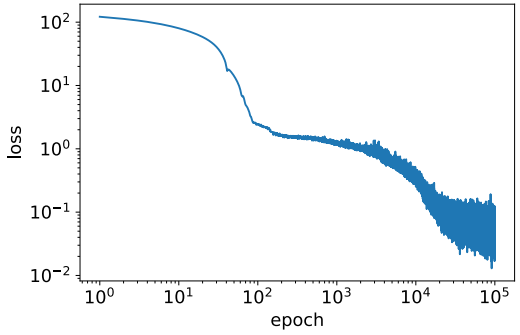

[Frank, Rauch, Zeppenfeld, '14]

# Leptonic Higgs Decay

[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Construct a fully-connected ANN as before (5 input nodes, 6*64 hidden nodes, 5 output nodes)
- Use tree-level $|\overline{\mathcal{M}}|^2$ (including all angular correlations) as the target function
- Train with batches of 1,000 events each (larger batches needed as phase space grows)
- A new complication arises during training due to vanishing of target function on a phase space boundary, making the loss function log-singular there
- Could be solved by a judicious choice of phase-space coordinates, but that's precisely what we want to avoid!
- Opted for a brute-force solution: "gradient clipping". It worked.

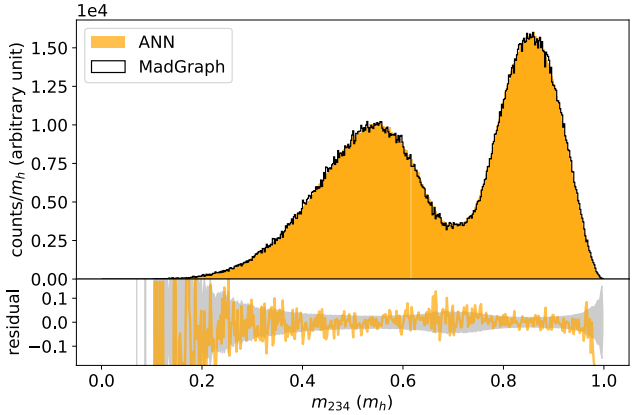


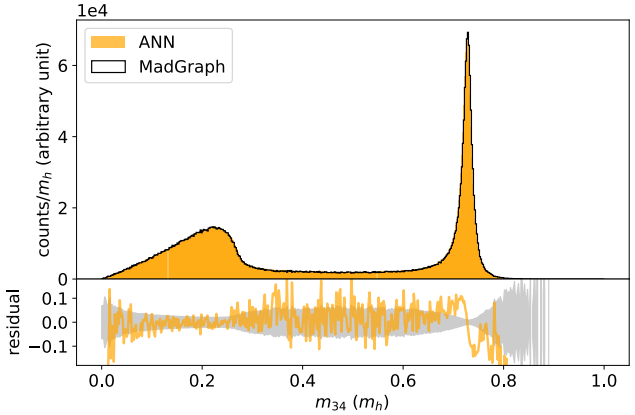



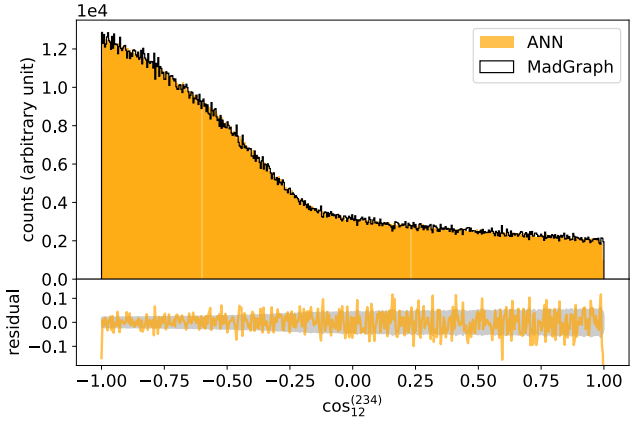
# Leptonic Higgs Decay

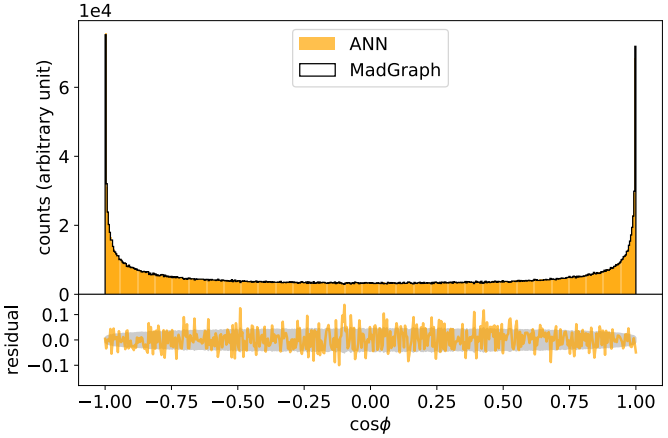
[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Unweighting efficiency of 26% achieved (compared to 8% for MadGraph)
- Generated distributions in perfect agreement with MadGraph



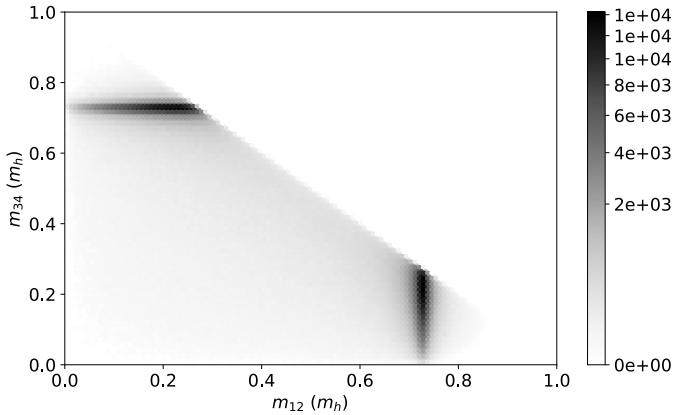



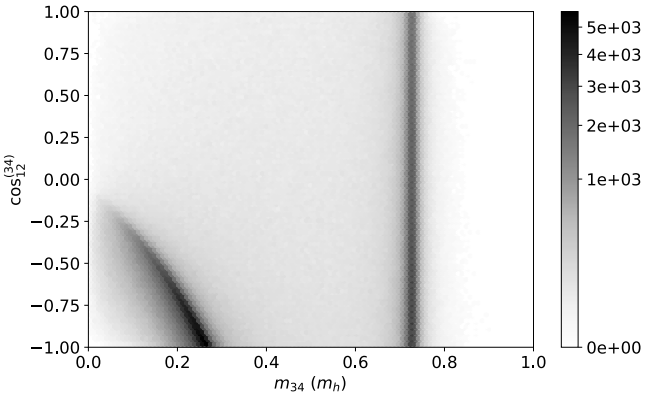



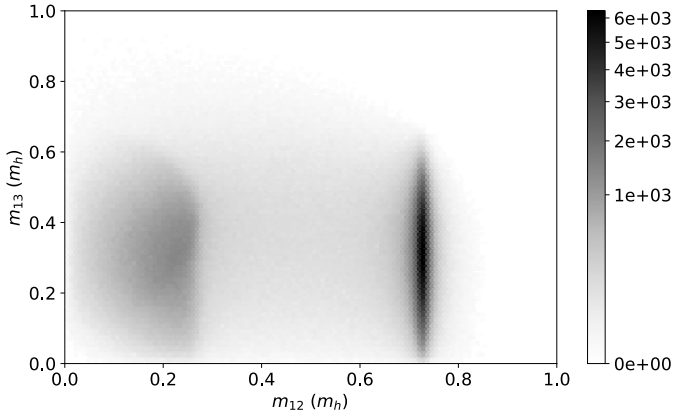
# Leptonic Higgs Decay

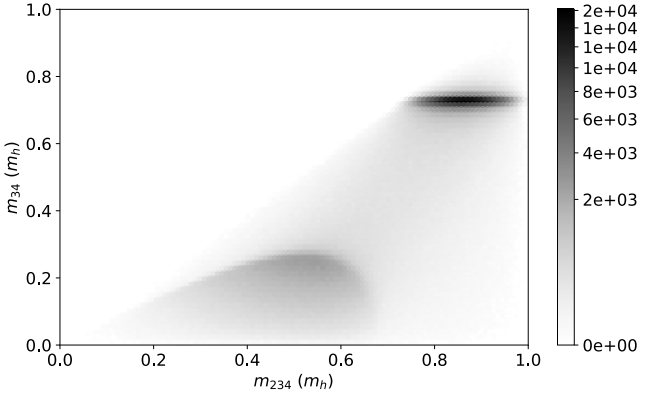
[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Resonant structure correctly reproduced in various coordinate slices



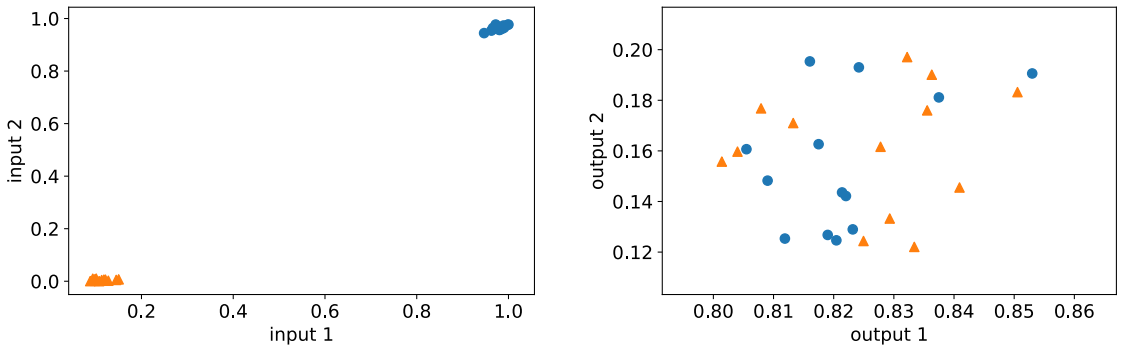



# Bijectivity of the NN Map

[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- The map defined by NN should be bijective (one-to-one) for the MC generation procedure to work correctly
- Non-surjective map would lead to empty regions in phase space, regardless of sample size
- Non-injective map would lead to incorrect evaluation of phase space density, invalidating both our training algorithm and unweighting procedure

$$p_y(\mathbf{y}) \equiv p_y(\mathbf{y}_\mathbf{w}(\mathbf{x})) = \left|\frac{\partial y_i}{\partial x_j}\right|^{-1}$$

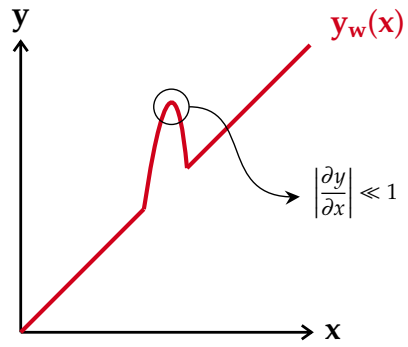



- Fully-commented ANN is not necessarily bijective by construction

# Bijectivity of the NN Map

[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Fortunately, the training procedure favors bijective maps
- Any continuous non-injective map would contain sub manifolds with small Jacobean

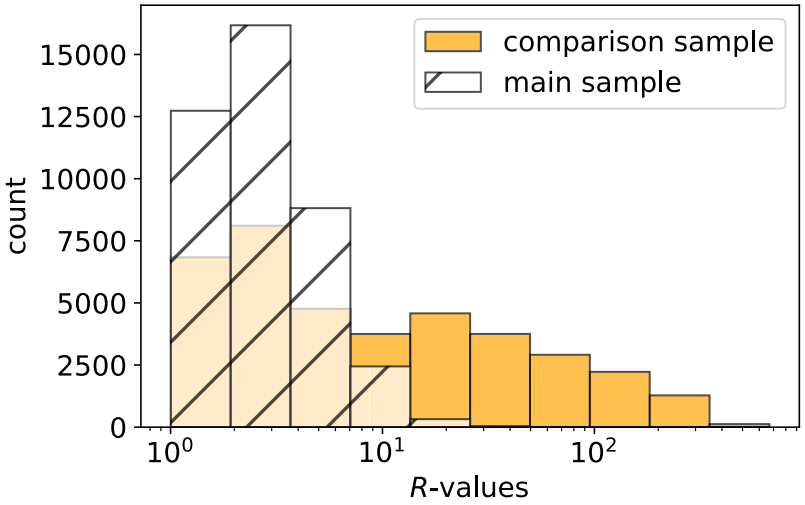


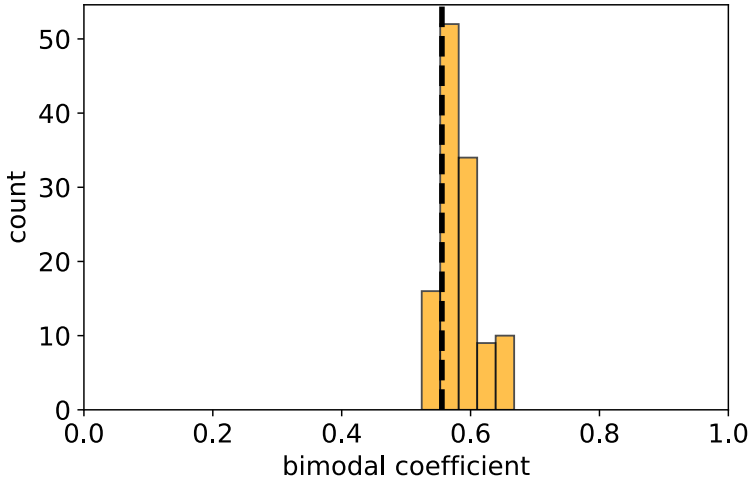

- This results in large local values of the loss function: $D_{\rm KL}[p_y(\mathbf{y}); f(\mathbf{y})] \equiv \int p_y(\mathbf{y}) \log \frac{p_y(\mathbf{y})}{f(\mathbf{y})}\, d\mathbf{y}$
- Training would adjust the map to eliminate such "foldings"
- This "unfolding" feature works very efficiently in practice, but does place a constraint on the form of the loss function (as we discovered the hard way)

# Bijectivity of the NN Map

[I. Chen, M. Kilmek and MP, 2009.07819, SciPost Phys]

- Instead of "built in" surjectivity, we rely on training to create a surjective map, and check surjectivity post-factum
- To check: Divide target space into small cubes, examine the input-space coordinates of points that map into each cube. Do they form a single cluster?




- Conclusion: deviations from surjectivity, if any, are small in our simulation
- Likewise, the trained map is injective to an excellent approximation

# Conclusions

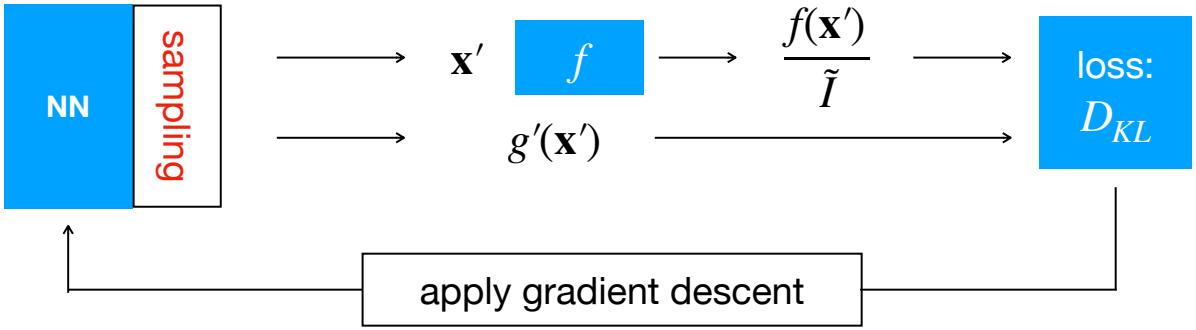
- Neural Network seems a natural candidate to realize "nonlinear importance sampling" in Monte Carlo simulations
- With a bit of tweaking (e.g. proprietary "soft clipping" output function), we got simple fully-connected NNs to work in realistic parton-level simulations with up to 4 final-state particles
- Can handle resonances, in a nicely coordinate-choice-independent way
- Can handle soft/collinear enhancements, generic kinematic cuts
- High unweighting efficiency achieved in all examples
- This may be a crucial advantage in situations when matrix element is computationally expensive to evaluate, e.g. N^kLO simulations
- Bijective (one-to-one) mapping is not built in, but is naturally imposed by training
- The approach seems quite promising, and applications to more challenging examples should be explored


# Monte Carlo Simulations with Neural Networks II: Normalizing Flows

C. Gao, J. Isaacson, and C. Krause (2020), 2001.05486
C. Gao, S. Hoche, J. Isaacson, C. Krause, and H. Schulz (2020), 2001.10028
PRECISION21 - KITP May 13th 2021

# NN based MC Integrator/Event Generator

Bendavid [1707.00028]
Klimek/Perelstein [1810.11509]




- $\mathbf{x}' = C(\mathbf{x})$, where $\mathbf{x} \sim g_0(\mathbf{x})$ ← **sampling**
- $\mathbf{x}' \sim g'(\mathbf{x}') = g_0\left(C^{-1}(\mathbf{x}')\right)\left|\frac{\partial C^{-1}}{\partial \mathbf{x}'}\right|$ ← **density evaluation**
- can model $C$ or $C^{-1}$ as NN
- requires inverting NN (i.e. computing determinant of Jacobian of a matrix) $\sim \mathcal{O}(D^3)$

# Normalizing Flows

Rezende/Mohamed [1505.05770]
Dinh et al. [1410.8516,1605.08803]

- $\mathbf{x}_K = C_K \circ C_{K-1} \cdots C_2 \circ C_1(\mathbf{x})$, where $C_k$ is bijective, invertible, differentiable
- If $\mathbf{x} \sim g_0(\mathbf{x})$, then $\mathbf{x}_K \sim g_K(\mathbf{x}_K) = g_0\left(C_1^{-1} \cdots C_K^{-1}(\mathbf{x}_K)\right) \prod_{k=1}^{K} \left| \frac{\partial C_k^{-1}}{\partial \mathbf{x}_k} \right|$
- $C$ or $C^{-1}$ can be designed such that the Jacobian-determinant computation $\sim \mathcal{O}(D)$

# Normalizing Flows

1912.02762 [stat.ML]

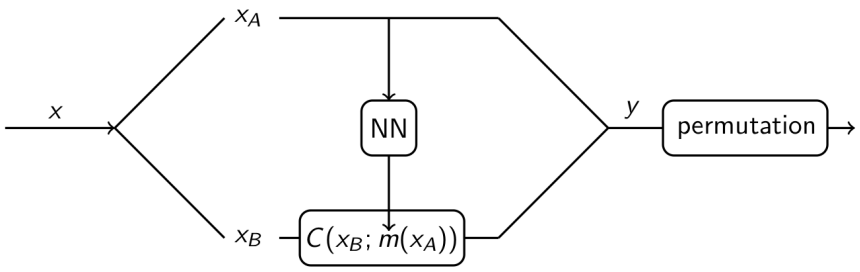
- $\mathbf{x}_K = C_K \circ C_{K-1} \cdots C_2 \circ C_1(\mathbf{x})$, wher
- If $\mathbf{x} \sim g_0(\mathbf{x})$, then $\mathbf{x}_K \sim g_K(\mathbf{x}_K) =$
- $C$ or $C^{-1}$ can be designed such that

| | | |
|---|---|---|
| Autoregressive flows | Transformer type:<br>– Affine<br>– Combination-based<br>– Integration-based<br>– Spline-based | Conditioner type:<br>– Recurrent<br>– Masked<br>– Coupling layer |
| Linear flows | Permutations<br>Decomposition-based:<br>– PLU<br>– QR<br>Orthogonal:<br>– Exponential map<br>– Cayley map<br>– Householder | |
| Residual flows | Contractive residual<br>Based on matrix determinant lemma:<br>– Planar<br>– Sylvester<br>– Radial | |

Table 1: Overview of methods for constructing flows based on finite compositions.

# Coupling Layer

Dinh et al. [1410.8516,1605.08803]




**Forward**

$$y_A = x_A$$
$$y_B = C(x_B; m(x_A))$$

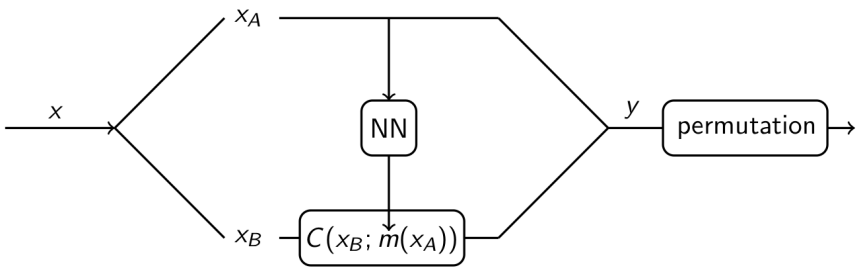
**Inverse**

$$x_A = y_A$$
$$x_B = C^{-1}(y_B; m(x_A))$$

- $C$ is an easy, invertible Coupling Transform function or a transformer
- $$g_y = \left|\partial y/\partial x\right|^{-1} g_x\,, \quad \left|\frac{\partial y}{\partial x}\right|^{-1} = \left|\begin{pmatrix} \overrightarrow{1} & 0 \\ \frac{\partial C}{\partial m}\frac{\partial m}{\partial x_A} & \frac{\partial C}{\partial x_B} \end{pmatrix}\right|^{-1} = \left|\frac{\partial C(x_B; m(x_A))}{\partial x_B}\right|^{-1}$$
- e.g. Affine CT: $C(x_B; s, t) = x_B \odot e^s + t \quad s, t \in \mathbb{R}^{|B|} \quad |\partial C/\partial x_B| = e^{\sum s_i}$

# Coupling Layer

Muller et al. [1808.03856]




**Forward**

$$y_A = x_A$$
$$y_B = C(x_B; m(\overrightarrow{x}_A))$$
$$g_y = g_x \left| \frac{\partial C(x_B; m(x_A))}{\partial x_B} \right|^{-1}$$

- domain and co-domain are restricted to unit hypercube
- separability: $C(x_B; m(x_A)) = \left( C_1(x_{B_1}; m), C_2(x_{B_2}; m), \cdots, C_{|B|}(x_{B_{|B|}}; m) \right)^{\mathrm{T}}$
- if $y \sim g_y$ is uniform, then $C_i$ acts as the cumulative distribution function (CDF) of $x_{B_i}$: $g_y \, dC_i = g_x \, dx_{B_i}$
- each CDF/transformer can be modeled by a piecewise monotonically increasing polynomial

# Example of Transformer

Muller et al. [1808.03856]

- Piecewise linear: Given fixed bin width $w$, NN predicts pdf bin heights $\sim Q_i$

$$C_i(x_{B_i}; Q) = \alpha Q_{ib} + \sum_{k=1}^{b-1} Q_{ik}$$

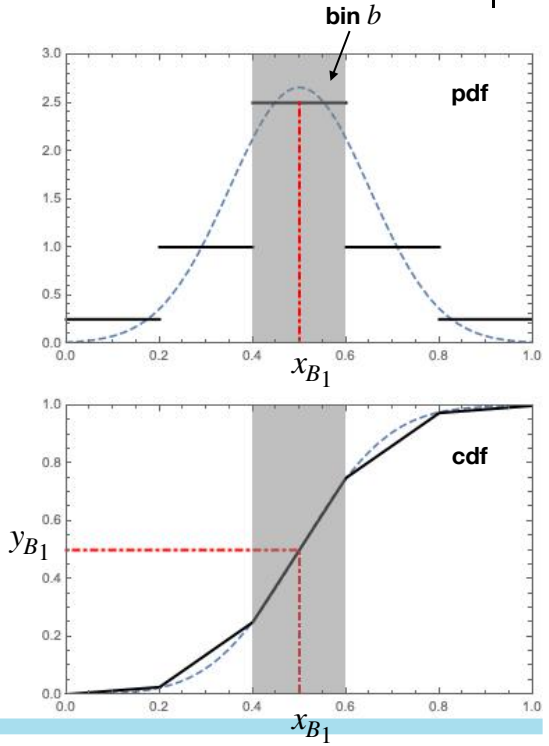
$$b = \lfloor \frac{x_{B_i}}{w} \rfloor \qquad \alpha = \frac{x_{B_i} - (b-1)w}{w}$$

$$\left| \frac{\partial C(x_B; Q)}{\partial x_B} \right| = \prod_i \left| \frac{\partial C_i(x_{B_i}; Q)}{\partial x_{B_i}} \right| = \prod_i \frac{Q_{ib}}{w}$$

- Piecewise quadratic: NN predicts both bin heights and bin widths for the pdf

**Forward**

$$y_A = x_A$$
$$y_B = C(x_B; m(\vec{x}_A)) \qquad g_y = g_x \left| \frac{\partial C(x_B; m(x_A))}{\partial x_B} \right|^{-1}$$






Fermilab

# Example of Transformer

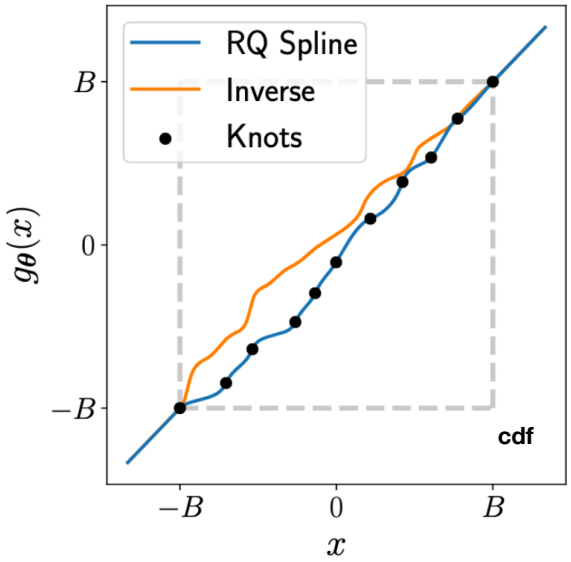
Durkan et al. [1906.04032]

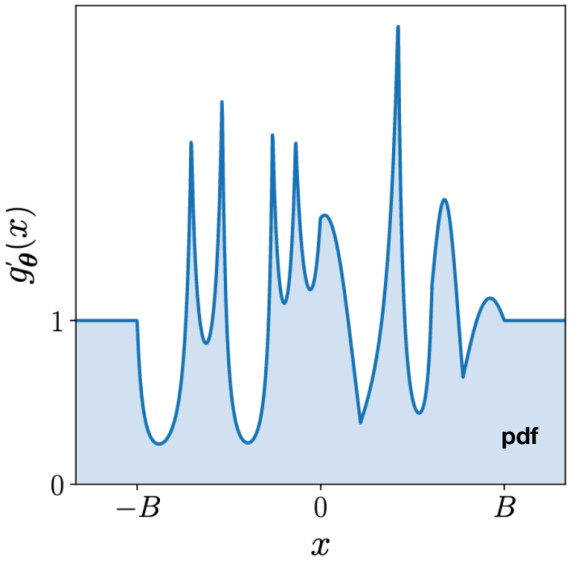
- Rational quadratic spline: NN predicts widths, heights, and derivatives of each knot of the spline.

**Forward**

$$y_A = x_A$$
$$y_B = C(x_B; m(\overrightarrow{x}_A))$$

$$g_y = g_x \left| \frac{\partial C(x_B; m(x_A))}{\partial x_B} \right|^{-1}$$



# How Many Coupling Layers are needed?

- $\mathbf{x}_K = C_K \circ C_{K-1} \cdots C_2 \circ C_1(\mathbf{x})$, where $C_k$ = NN based CL that transforms roughly half of $\mathbf{x}$
- capture all the correlations between every dimension of $\mathbf{x}$
- transform (or train) each dimension equal number of times
- $D$ layers for $D \leq 5$, $2\lceil \log_2 D \rceil$ for $D > 5$
- e.g. D=12

| Dimension | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Transformation 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Transformation 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Transformation 3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Transformation 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

# i-flow: Integration and Sampling with Normalizing Flows

2001.05486 [physics.comp-ph] https://gitlab.com/i-flow/i-flow

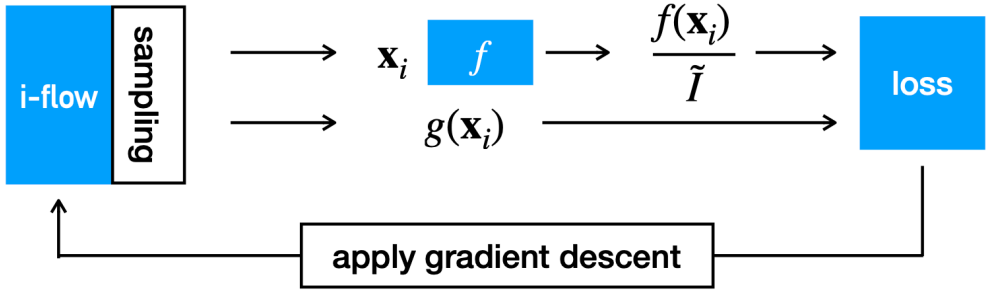
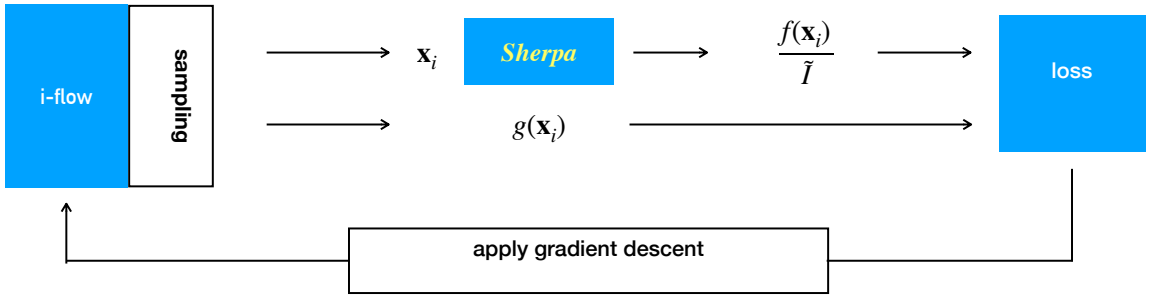



FIG. 2: Illustration of one step in the training of `i-flow`. Users need to provide a normalizing flow network, a function $f$ to integrate, and a loss function. $\tilde{I}$ stands for the Monte-Carlo estimate of the integral using the sample of points $\vec{x}_i$, and $g(\vec{x}_i)$ is the probability of a given point occurring in the `i-flow` sampling.

- $I = \int d^D x\, g(\mathbf{x})\, \frac{f(\mathbf{x})}{g(\mathbf{x})} = V\langle f/g\rangle_G$ , where $g$ resembles the shape of $f$ ( ideally $g \to f/I$ )
- Now can sample uniformly in $d^D G = g(\mathbf{x})\, d^D x$, with uncertainty: $\Delta I = V\sqrt{\frac{\langle (f/g)^2\rangle_G - \langle f/g\rangle_G^2}{N-1}}$

# i-flow + Sherpa: Phase Space Integration

2001.05478 [hep-ph], 2001.10028 [hep-ph]




- Sherpa computes matrix element squared with color sampling
- recursive multi-channel algorithm maps the integration domain in **i-flow** (a unit hypercube) to physical variables: $n_{dim} = (3n_f - 4) + (n_f - 1) + n_{ihadrons}$

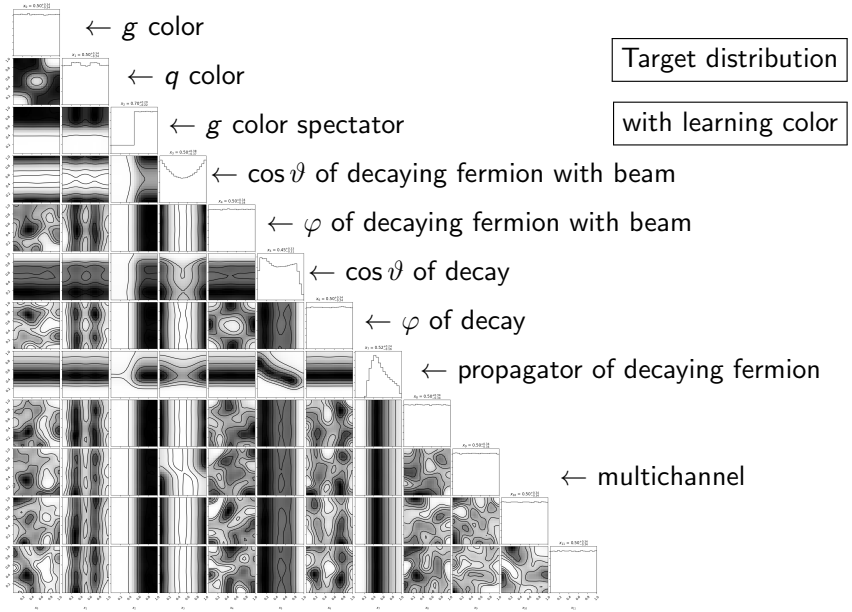
**kinematics** **multi-channel** **proton pdf**

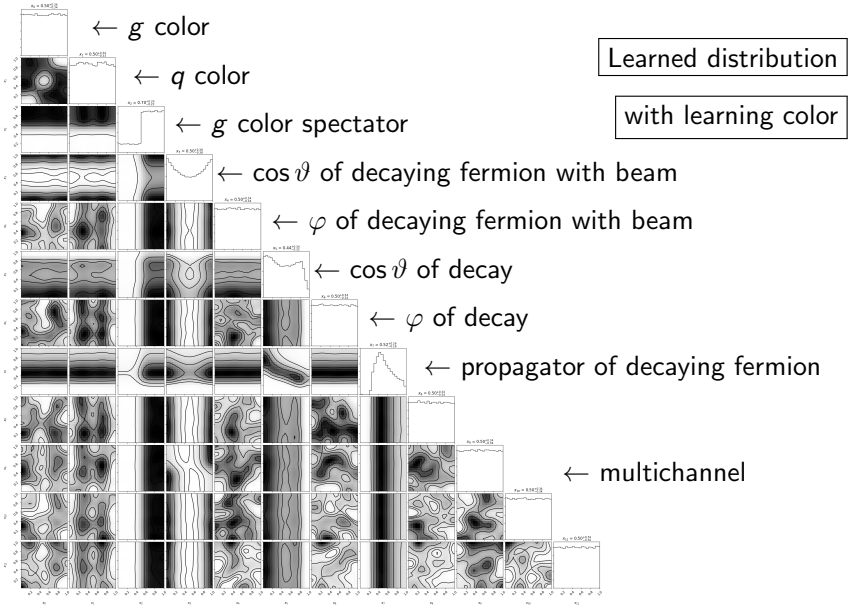
- integrating over final color configurations adds $2n_c - 1$ more variables

https://sherpa-team.gitlab.io

# Example: $e^+e^- \to q\bar{q}g$

Borrowed from Claudius Krause




- $\sigma_{NN} = 4887.1 \pm 4.6pb$
- $\sigma_{Sherpa} = 4887.0 \pm 17.7pb$



Fermilab

# Example: $pp \to V + \text{jets}$

| 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 | $6.1 \cdot 10^{-1}$ | $1.2 \cdot 10^{-1}$ | $1.0 \cdot 10^{-3}$ | $1.8 \cdot 10^{-3}$ | $8.9 \cdot 10^{-4}$ | $1.6 \cdot 10^{-1}$ | $4.1 \cdot 10^{-3}$ |
| | 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 |

TABLE II: Unweighting efficiencies at the LHC at $\sqrt{s} = 14$ TeV using the NNPDF 3.0 NNLO PDF set and a correspondingly defined strong coupling. Jets are identified using the $k_T$ clustering algorithm with $R = 0.4$, $p_{T,j} > 20$ GeV and $|\eta_j| < 6$. In the case of $Z/\gamma^*$ production, we also apply the invariant mass cut $66 < m_{ll} < 116$GeV.

# Conclusion

- Discrete variables like multi-channel or color may not be modeled well by a spline, which could be the reason why it does not work so well for $n \geq 2$ jets.
- **i-flow** takes significantly longer to achieve optimal performance compared to VEGAS.
- After all, it is a MC technique, to get the corners/tails right requires some luck or a very large number of samples to train, which then runs into memory problem.
- New developments in Normalizing Flows could potentially improve the prospect of NN based MC integrator/event generator.
- Thank you!