

Machine Learning for Event Generation and Fast Simulation

— Prospecting for New Physics through Flavor, Dark Matter, and Machine Learning —
— Aspen, CO —

Claudius Krause

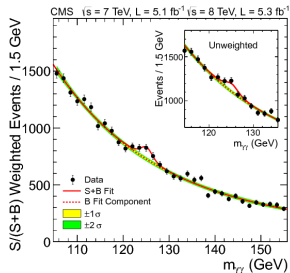
Institute for Theoretical Physics, University of Heidelberg

March 28, 2023

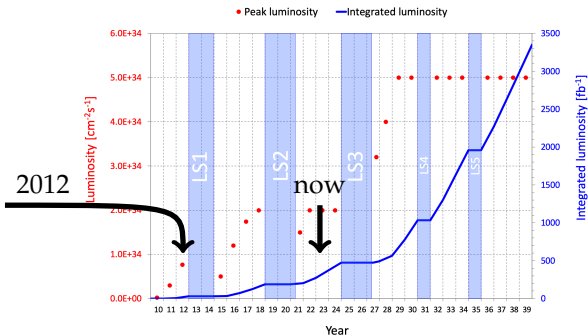


UNIVERSITÄT
HEIDELBERG
Zukunft. Seit 1386.

We will have a lot more data in the near future.



CMS Collaboration [arXiv:1207.7235, Phys.Lett.B]

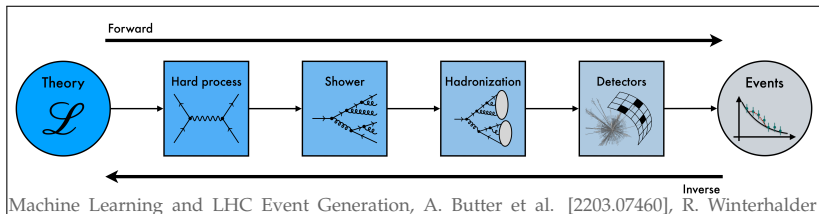


<https://lhc-commissioning.web.cern.ch/schedule/HL-LHC-plots.htm>

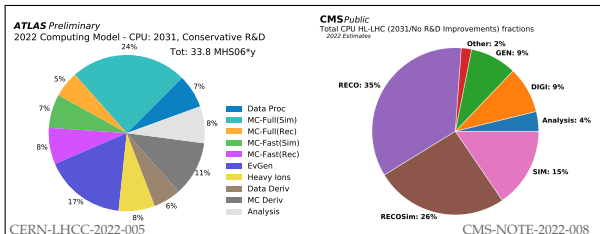
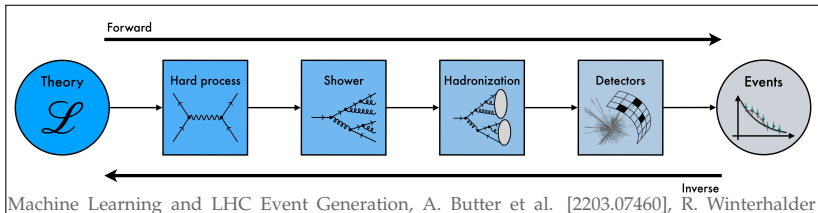
- We will have $20\text{--}25\times$ more data.

⇒ We want to understand every aspect of it based on 1st principles!
(and find New Physics)

Simulation bridges Theory and Experiment.



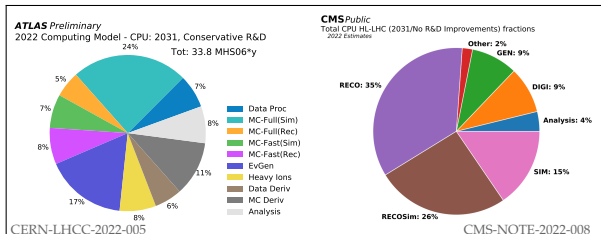
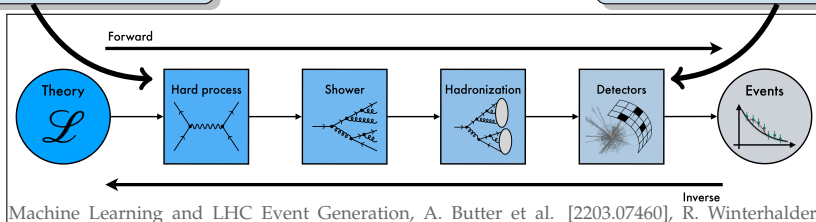
Simulation bridges Theory and Experiment.



Simulation bridges Theory and Experiment.

Phase Space Sampling
 \Rightarrow sample according to $d\sigma$

Detector Simulation
 \Rightarrow sample from $p(\text{showers}|E)$



Deep Generative Models can be Fast Surrogates for Expensive Simulations.



- Deep Generative Models learn to sample from complicated $p(x)$.
- They can generate impressive results for text, speech, images, ...

However, we in HEP have different requirements for quality:

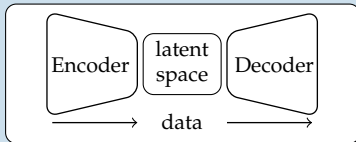
⇒ We want to correctly cover $p(x)$ of the entire phase space.

"Calorimeter Simulation" via [midjourney.com](https://www.midjourney.com)

The Landscape of Generative Models.

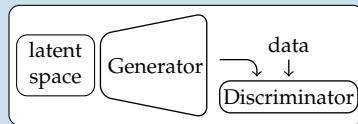
Variational Autoencoder (VAE)

⇒ Compressing data through a bottleneck.



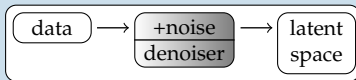
Generative Adversarial Network (GAN)

⇒ Generator and Discriminator play a game against each other.



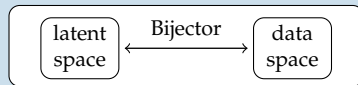
Diffusion Models

⇒ Gradually add noise and revert.

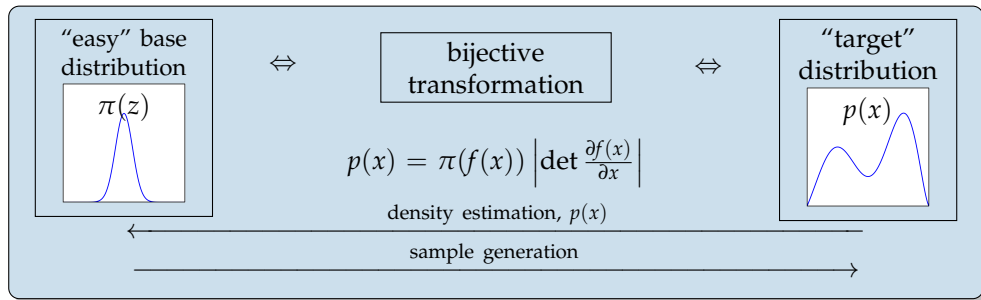


Normalizing Flows

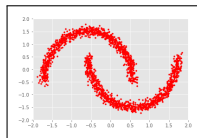
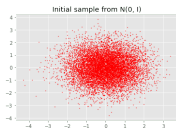
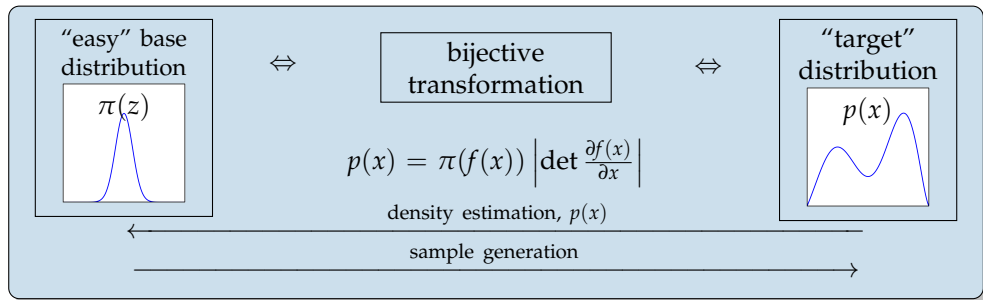
⇒ Bijective map to a known distribution.



Normalizing Flows learn a coordinate transformation.

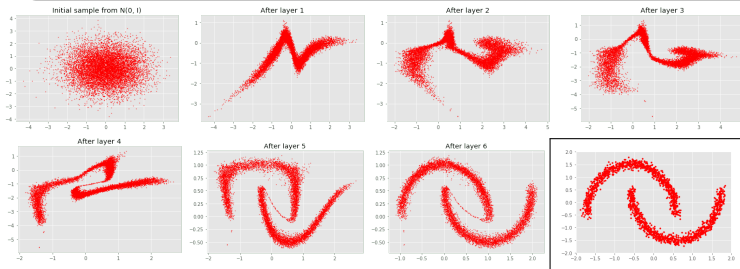
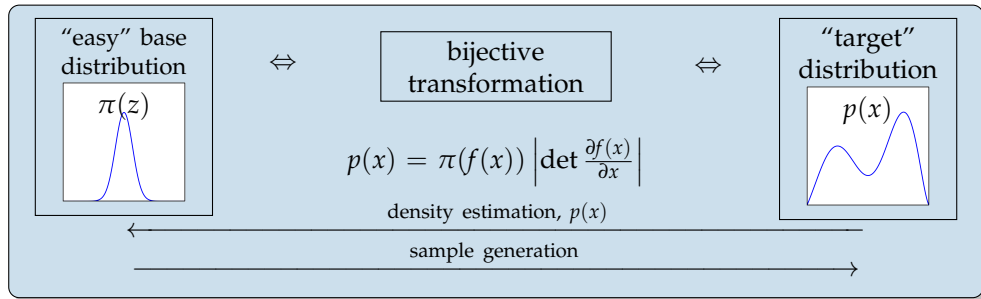


Normalizing Flows learn a coordinate transformation.



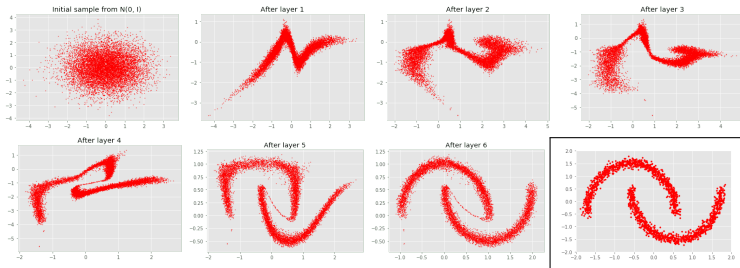
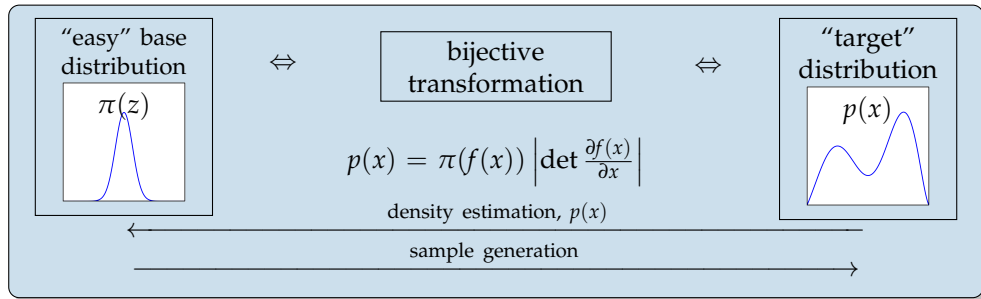
<https://engineering.papercup.com/posts/normalizing-flows-part-2/>

Normalizing Flows learn a coordinate transformation.



<https://engineering.papercup.com/posts/normalizing-flows-part-2/>

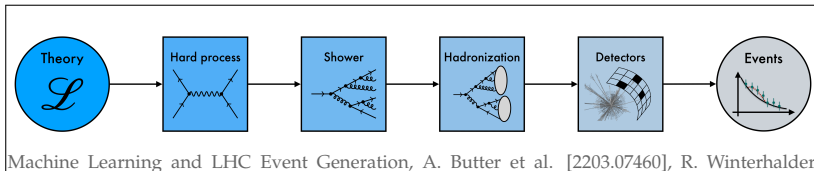
Normalizing Flows learn a coordinate transformation.



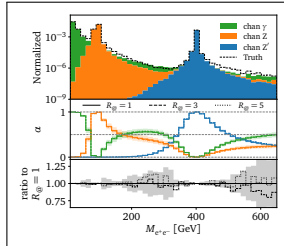
- ✓ Optimization via LL.
- ✓ Stable results.
- ✓ Versatile architecture.

<https://engineering.papercup.com/posts/normalizing-flows-part-2/>

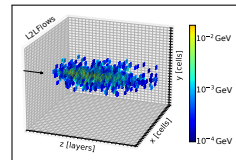
Machine Learning for Event Generation and Fast Simulation



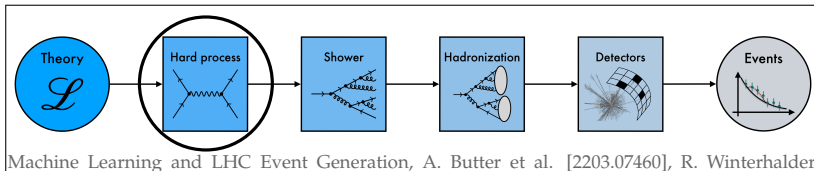
1: Phase Space Integration



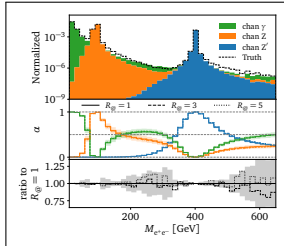
2: Calorimeter Simulation



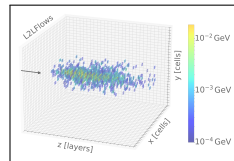
Machine Learning for Event Generation and Fast Simulation



1: Phase Space Integration

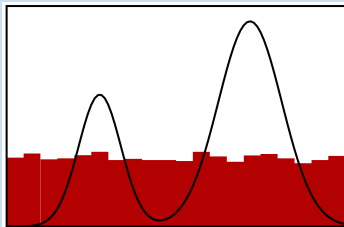


2: Calorimeter Simulation



Phase Space integration uses Importance Sampling.

$$I = \int_0^1 f(\vec{x}) d\vec{x}$$

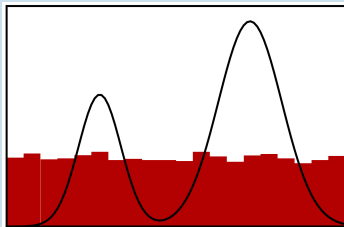


flat sampling:
inefficient.

$$I = \langle f(\vec{x}) \rangle_{x \sim \text{uniform}}$$

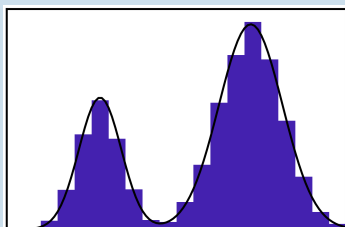
Phase Space integration uses Importance Sampling.

$$I = \int_0^1 f(\vec{x}) d\vec{x}$$



flat sampling:
inefficient.

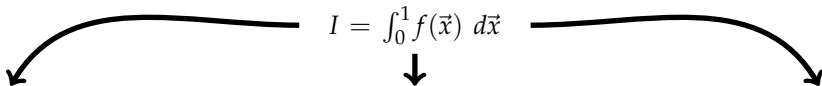
$$I = \langle f(\vec{x}) \rangle_{x \sim \text{uniform}}$$

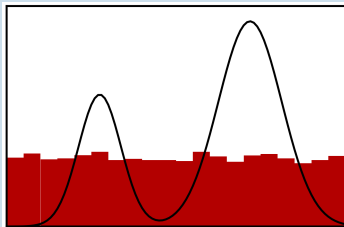


importance sampling:
find g close to f

$$I = \left\langle \frac{f(\vec{x})}{g(\vec{x})} \right\rangle_{x \sim g(x)}$$

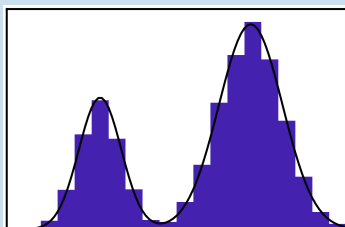
Phase Space integration uses Importance Sampling.

$$I = \int_0^1 f(\vec{x}) d\vec{x}$$




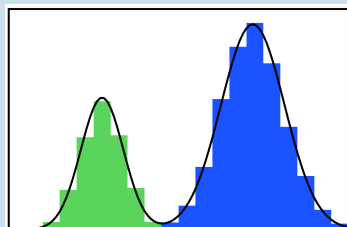
flat sampling:
inefficient.

$$I = \langle f(\vec{x}) \rangle_{x \sim \text{uniform}}$$



importance sampling:
find g close to f

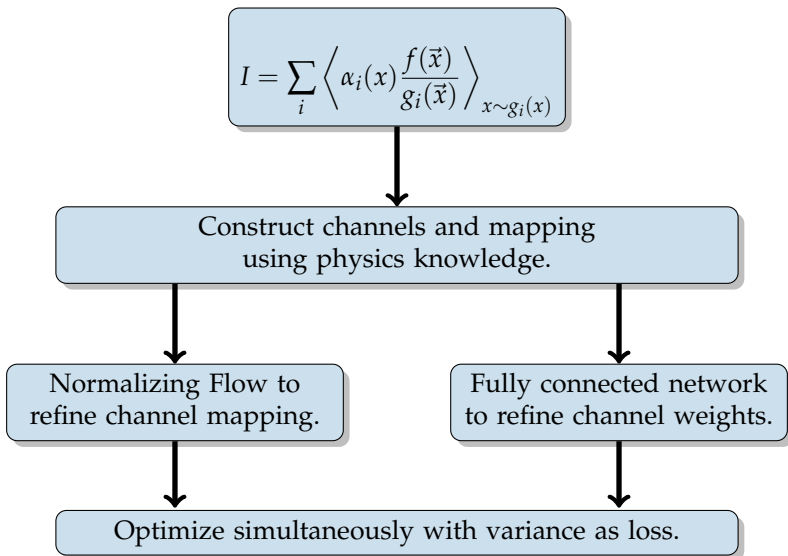
$$I = \left\langle \frac{f(\vec{x})}{g(\vec{x})} \right\rangle_{x \sim g(x)}$$



multichannel: one
map per channel

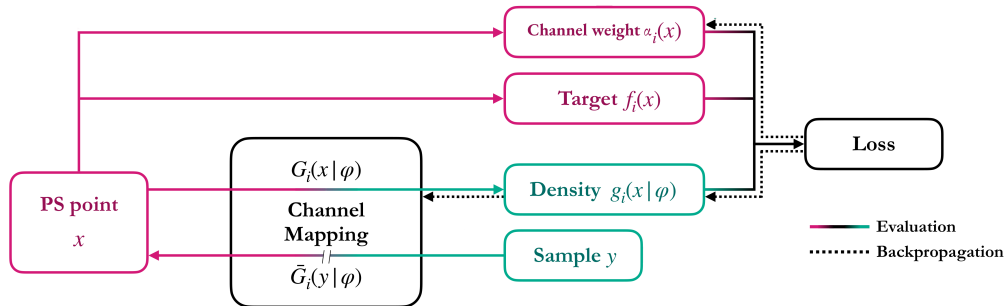
$$I = \sum_i \left\langle \alpha_i(x) \frac{f(\vec{x})}{g_i(\vec{x})} \right\rangle_{x \sim g_i(x)}$$

MadNIS — Neural Importance Sampling



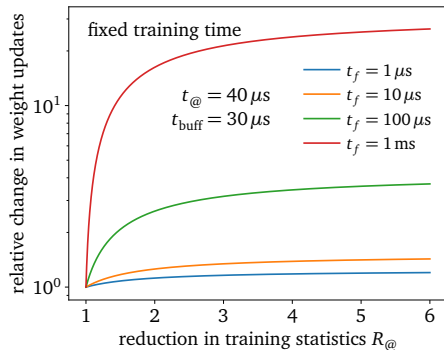
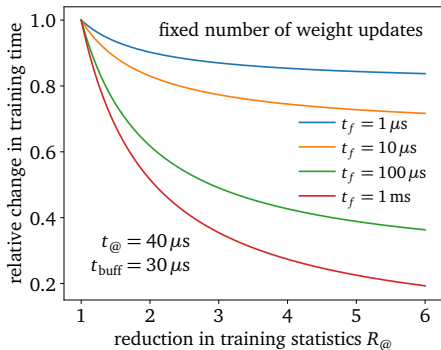
A. Butter, T. Heime1, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

MadNIS — Neural Importance Sampling



A. Butter, T. Heimel, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

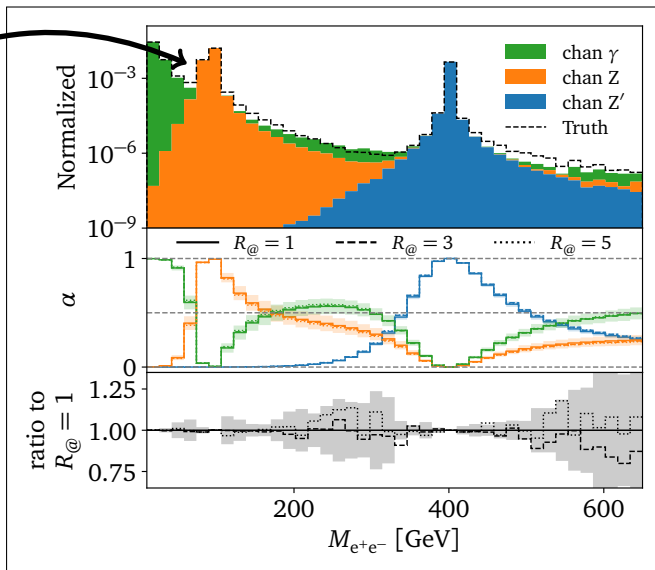
MadNIS re-uses expensive matrix elements



A. Butter, T. Heimes, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

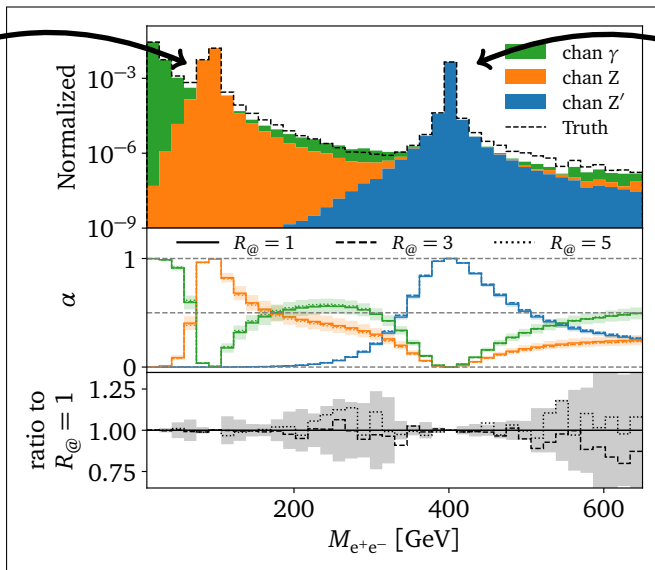
MadNIS — Results for Drell-Yan + Z'

Learned
distribution
matches truth.



MadNIS — Results for Drell-Yan + Z'

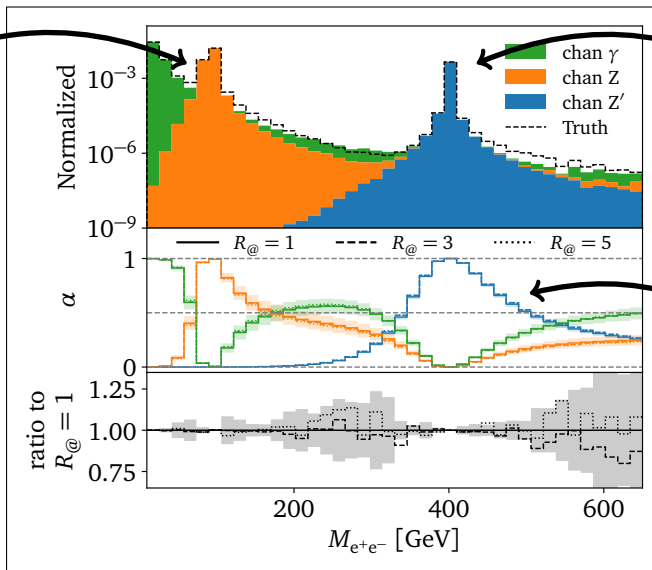
Learned distribution matches truth.



Peaks are learned by different channels.

MadNIS — Results for Drell-Yan + Z'

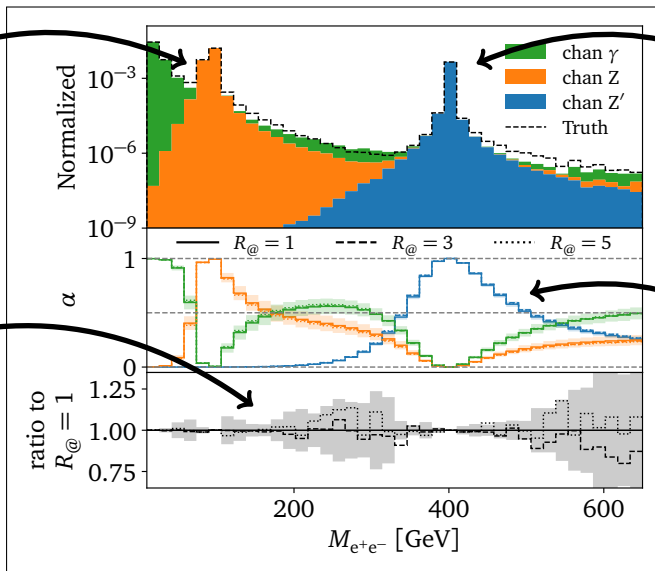
Learned distribution matches truth.



Peaks are learned by different channels.

Channel weights are learned by the network

MadNIS — Results for Drell-Yan + Z'



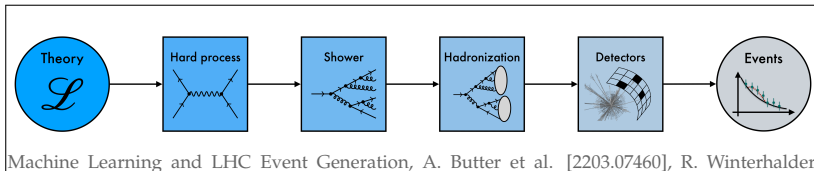
Learned distribution matches truth.

Peaks are learned by different channels.

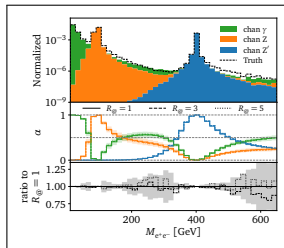
Re-uses samples to make training faster.

Channel weights are learned by the network

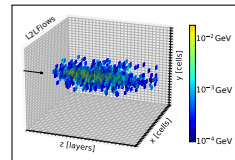
Machine Learning for Event Generation and Fast Simulation



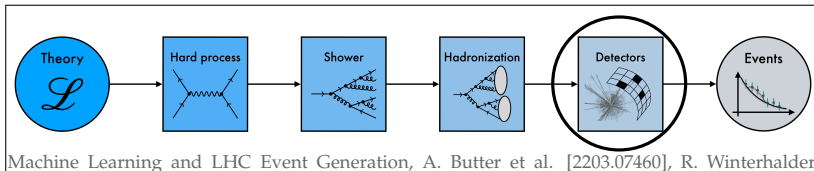
1: Phase Space Integration



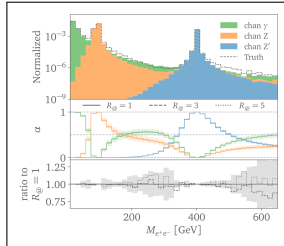
2: Calorimeter Simulation



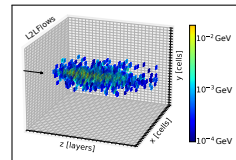
Machine Learning for Event Generation and Fast Simulation



1: Phase Space Integration

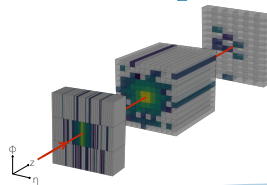


2: Calorimeter Simulation



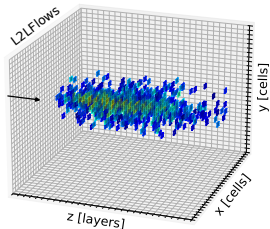
Different Datasets have been explored.

The CALOGAN Dataset.
 \Rightarrow CALOFLOW

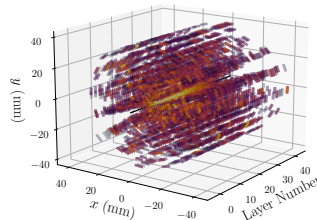


No time today, check out
[2302.11594].

The ILD Dataset
 \Rightarrow L2LFlows

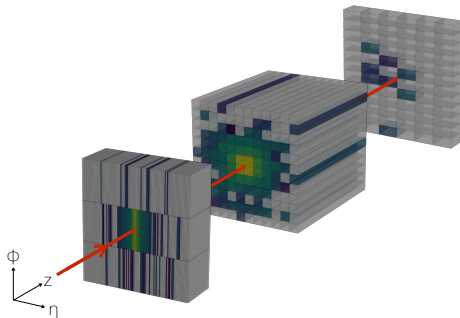
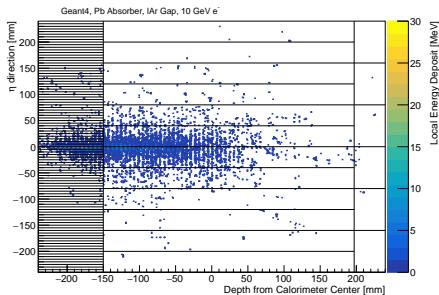


The CaloChallenge 2022
 \Rightarrow iCALOFLOW, ...



CALOFLOW uses the same calorimeter geometry as CALOGAN

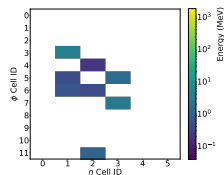
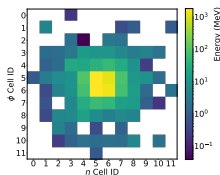
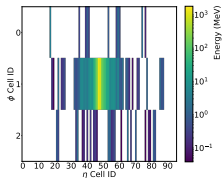
- We consider a toy calorimeter inspired by the ATLAS ECal: flat alternating layers of lead and LAr
- They form three instrumented layers of dimension 3×96 , 12×12 , and 12×6



CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]

CALOFLOW uses the same calorimeter geometry as CALOGAN

- The GEANT4 configuration of CALOGAN is available at <https://github.com/hep-lbdl/CaloGAN>
- We produce our own dataset: available at [DOI: 10.5281/zenodo.5904188]
- Showers of e^+ , γ , and π^+ (100k each)
- All are centered and perpendicular
- E_{inc} is uniform in $[1, 100]$ GeV and given in addition to the energy deposits per voxel:



CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]

CALOFLOW uses a 2-step approach to learn $p(\vec{\mathcal{I}}|E_{\text{inc}})$.

Flow I

- learns $p_1(E_0, E_1, E_2|E_{\text{inc}})$
- is a Masked Autoregressive Flow, optimized using the log-likelihood.

Flow II

- learns $p_2(\hat{\vec{\mathcal{I}}}|E_0, E_1, E_2, E_{\text{inc}})$ of normalized showers
- in CALOFLOW v1 (2106.05285 — called “teacher”):
 - Masked Autoregressive Flow trained with log-likelihood
 - Slow in sampling ($\approx 500\times$ slower than CALOGAN)
- in CALOFLOW v2 (2110.11377 — called “student”):
 - Inverse Autoregressive Flow trained with Probability Density Distillation from teacher (log-likelihood prohibitive),
i.e. matching IAF parameters to frozen MAF van den Oord et al.[1711.10433]
 - Fast in sampling ($\approx 500\times$ faster than CALOFLOW v1)

CALOFLOW passes the “ultimate metric” test.

According to the Neyman-Pearson Lemma we have: $p_{\text{GEANT4}}(x) = p_{\text{generated}}(x)$ if a classifier cannot distinguish data from generated samples.

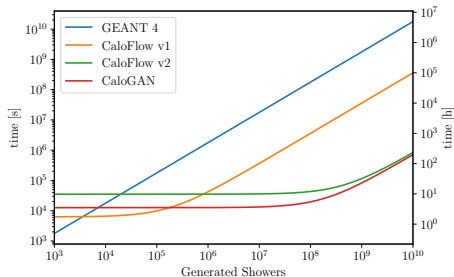
AUC		GEANT4 vs. CALOGAN	GEANT4 vs. (teacher) CALOFLOW v1	GEANT4 vs. (student) CALOFLOW v2
e^+	low-level	1.000(0)	0.870(2)	0.824(4)
	high-level	1.000(0)	0.795(1)	0.762(3)
γ	low-level	1.000(0)	0.796(2)	0.760(3)
	high-level	1.000(0)	0.727(2)	0.739(2)
π^+	low-level	1.000(0)	0.755(3)	0.807(1)
	high-level	1.000(0)	0.888(1)	0.893(2)

AUC ($\in [0.5, 1]$): Area Under the ROC Curve, smaller is better, i.e. more confused

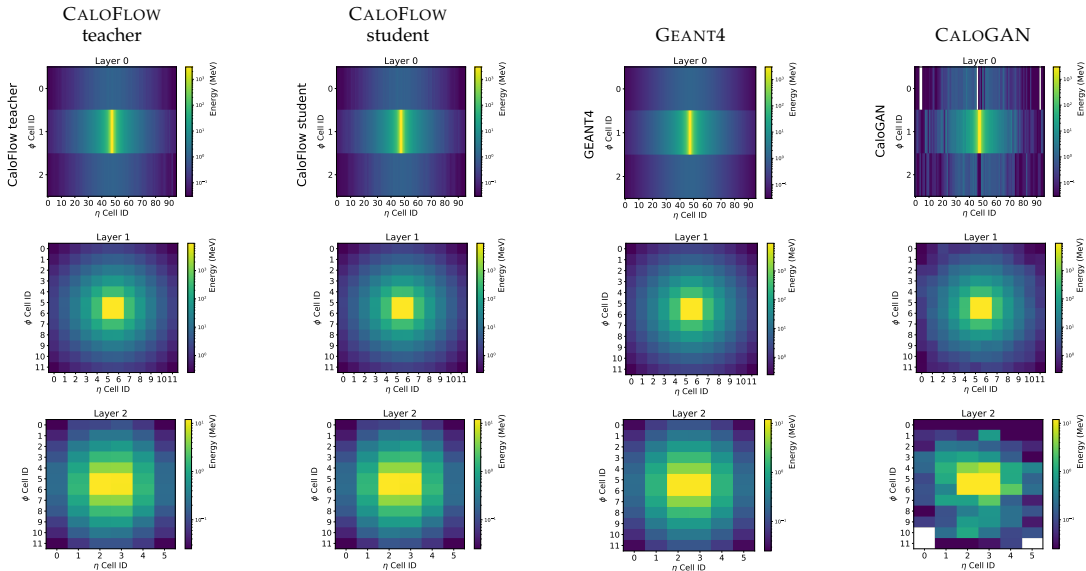
Sampling Speed: The Student beats the Teacher!

	CALOFlow*		CALOGAN*	GEANT4 [†]
	teacher	student		
training	22+82 min	+ 480 min	210 min	0 min
generation time per shower	36.2 ms	0.08 ms	0.07 ms	1772 ms

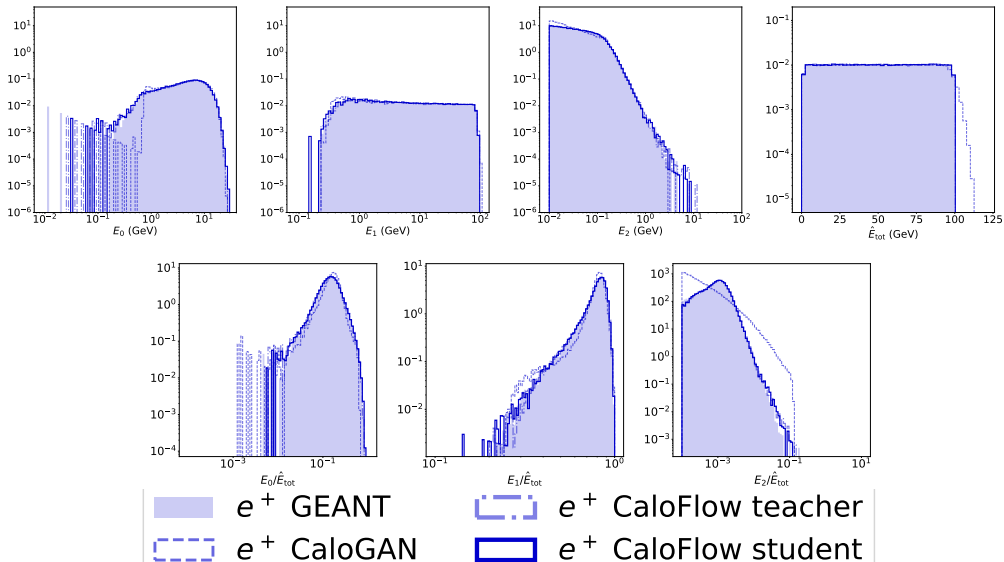
*: on our TITAN V GPU, [†]: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]



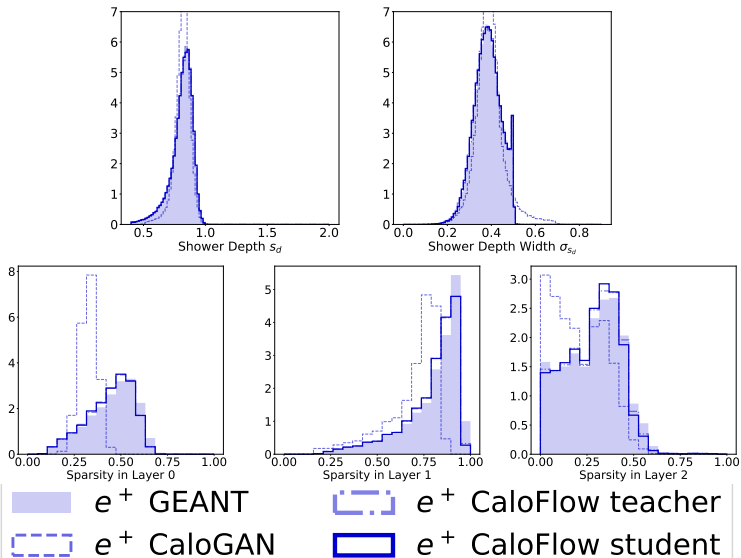
CALOFLOW: Comparing Shower Averages: e^+



CALOFLOW: histograms: e^+



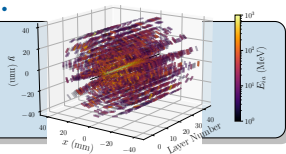
CALOFLOW: histograms: e^+



CaloChallenge datasets 2 and 3 are huge.

CaloChallenge datasets 2 and 3 are much bigger:

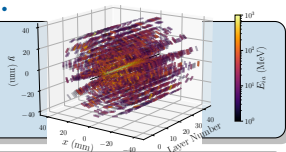
- Dataset 2: 144 voxels in 45 layers \rightarrow 6480 total.
- Dataset 3: 900 voxels in 45 layers \rightarrow 40500 total.



CaloChallenge datasets 2 and 3 are huge.

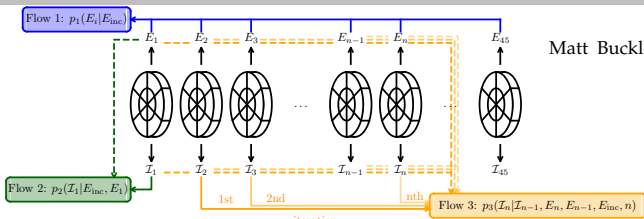
CaloChallenge datasets 2 and 3 are much bigger:

- Dataset 2: 144 voxels in 45 layers \rightarrow 6480 total.
- Dataset 3: 900 voxels in 45 layers \rightarrow 40500 total.



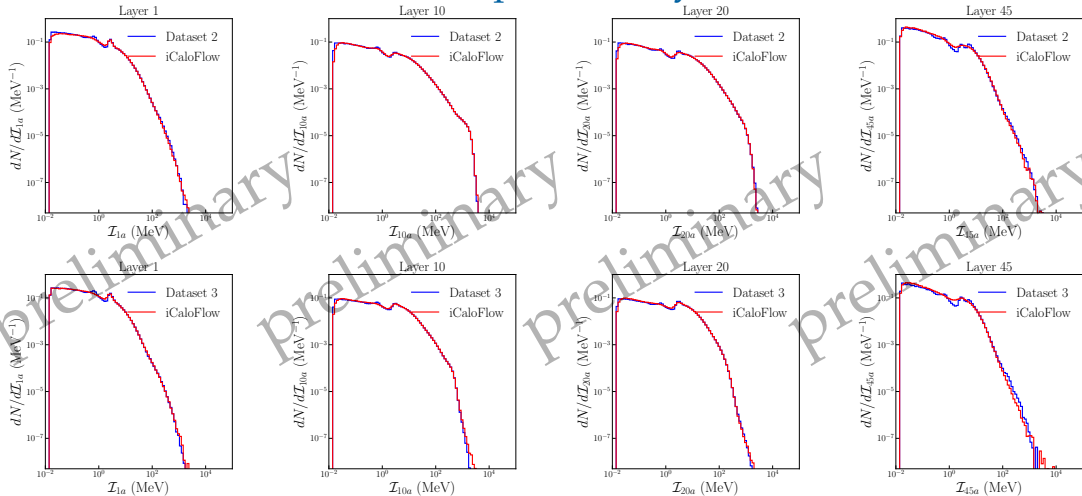
iCALOFlow: Split learning $p(\vec{\mathcal{I}}|E_{\text{inc}})$ into 3 steps, leveraging the regular detector geometry.

- 1 learns $p_1(E_1, E_2, E_3, \dots, E_{45}|E_{\text{inc}})$ \rightarrow how energy is distributed among layers.
- 2 learns $p_2(\mathcal{I}_1|E_1, E_{\text{inc}})$ \rightarrow how the shower in the first layer looks like.
- 3 learns $p_3(\mathcal{I}_n|\mathcal{I}_{n-1}, n, E_n, E_{n-1}, E_{\text{inc}})$ \rightarrow how the shower in layer n looks like, given layer $n - 1$



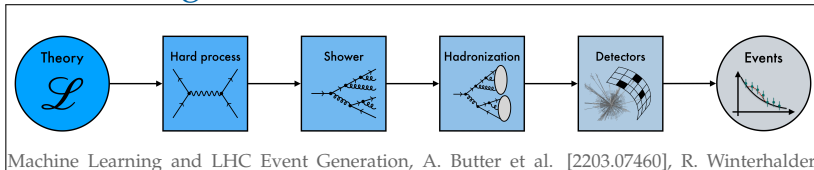
Work in progress with
Matt Buckley, Ian Pang, David Shih

iCALOFlow: preliminary results

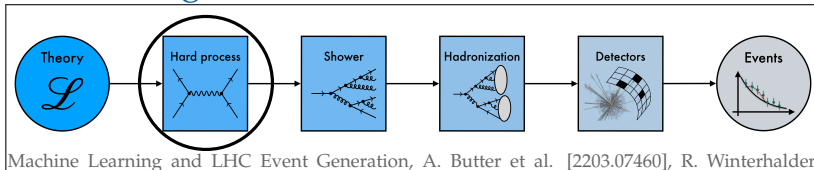


Classifier AUCs:	dataset 2, low:	0.797(5)	dataset 2, high:	0.798(3)
	dataset 3, low:	0.911(3)	dataset 3, high:	0.941(1)

Machine Learning for Event Generation and Fast Simulation

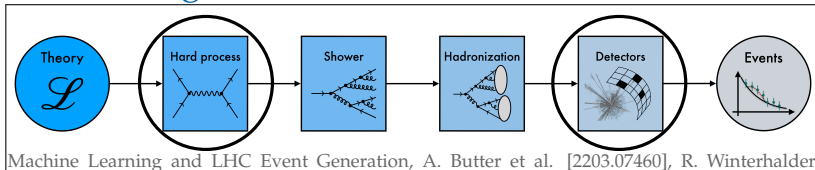


Machine Learning for Event Generation and Fast Simulation



- ⇒ Normalizing Flows are perfect for Importance Sampling.
- ⇒ They don't introduce a bias in the result, only increase the uncertainty if not converged.
- ⇒ They can be combined with other parts of MadGraph / Sherpa.

Machine Learning for Event Generation and Fast Simulation



- ⇒ Normalizing Flows are perfect for Importance Sampling.
- ⇒ They don't introduce a bias in the result, only increase the uncertainty if not converged.
- ⇒ They can be combined with other parts of MadGraph / Sherpa.

- ⇒ Normalizing Flows are able to generate high-quality showers, outperforming other generative models.
- ⇒ Training and model-selection is usually more stable.
- ⇒ The naive scaling to higher dimensions requires a lot of compute. But some assumptions on the underlying physics can help reduce the needed resources.

Backup

How do Normalizing Flows tame Jacobians?

- NFs learn the parameters κ of a series of easy transformations.

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

- Each transformation is 1d & has an analytic Jacobian and inverse.

⇒ We use Rational Quadratic Splines

Durkan et al. [arXiv:1906.04032], Gregory/Delbourgo [IMA J. of Num. An., '82]

- Require a triangular Jacobian for faster evaluation.

⇒ The parameters κ depend only on a subset of all other coordinates.

Having access to the log-likelihood (LL) allows several training options:

⇒ Based on samples: via maximizing $\text{LL}(\text{samples})$.

⇒ Based on target function $f(x)$: via matching $p(x)$ to $f(x)$.

The Bijector is a chain of “easy” transformations.

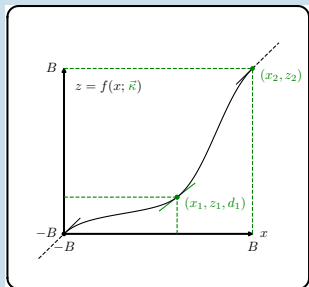
Each transformation

- must be invertible and have analytical Jacobian
- is chosen to factorize:

$$\vec{C}(\vec{x}; \vec{\kappa}) = (C_1(x_1; \kappa_1), C_2(x_2; \kappa_2), \dots, C_n(x_n; \kappa_n))^T,$$

where \vec{x} are the coordinates to be transformed and $\vec{\kappa}$ the parameters of the transformation.

Rational Quadratic Splines:



Durkan et al. [arXiv:1906.04032]

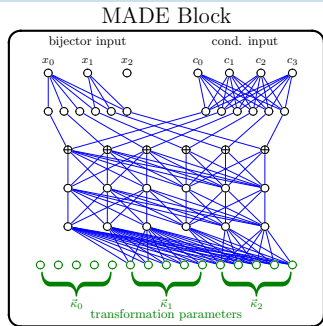
Gregory/Delbourgo [IMA Journal
of Numerical Analysis, '82]

$$C = \frac{a_2 \alpha^2 + a_1 \alpha + a_0}{b_2 \alpha^2 + b_1 \alpha + b_0}$$

- numerically easy
- expressive

The NN predicts the bin widths, heights, and derivatives that go in a_i & b_i .

Triangular Jacobians 1: with Autoregressive Blocks



$$\kappa_{x_i}(x_{j < i})$$

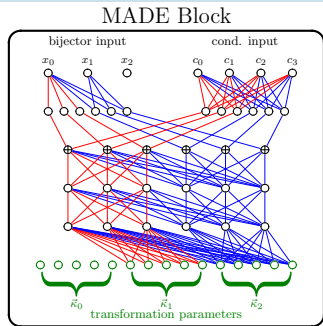
Implementation via masking:

- a single “forward” pass gives all $\kappa_{x_i}(x_{i-1} \dots x_1)$.
 \Rightarrow very fast
- its “inverse” needs to loop through all dimensions.
 \Rightarrow very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Masked Autoregressive Flow (MAF) is slow in sampling and fast in inference.
Papamakarios et al. [arXiv:1705.07057]
- Inverse Autoregressive Flow (IAF) is fast in sampling and slow in inference.
Kingma et al. [arXiv:1606.04934]

Triangular Jacobians 1: with Autoregressive Blocks



$$\kappa_{x_i}(x_{j < i})$$

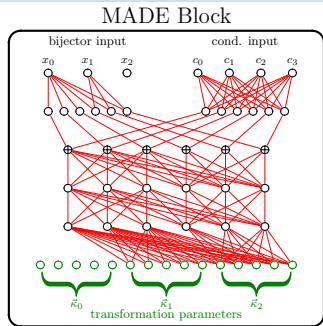
Implementation via masking:

- a single “forward” pass gives all $\kappa_{x_i}(x_{i-1} \dots x_1)$.
 \Rightarrow very fast
- its “inverse” needs to loop through all dimensions.
 \Rightarrow very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Masked Autoregressive Flow (MAF) is slow in sampling and fast in inference.
Papamakarios et al. [arXiv:1705.07057]
- Inverse Autoregressive Flow (IAF) is fast in sampling and slow in inference.
Kingma et al. [arXiv:1606.04934]

Triangular Jacobians 1: with Autoregressive Blocks



$$\kappa_{x_i}(x_{j < i})$$

Implementation via masking:

- a single “forward” pass gives all $\kappa_{x_i}(x_{i-1} \dots x_1)$.
 \Rightarrow very fast
- its “inverse” needs to loop through all dimensions.
 \Rightarrow very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Masked Autoregressive Flow (MAF) is slow in sampling and fast in inference.
Papamakarios et al. [arXiv:1705.07057]
- Inverse Autoregressive Flow (IAF) is fast in sampling and slow in inference.
Kingma et al. [arXiv:1606.04934]

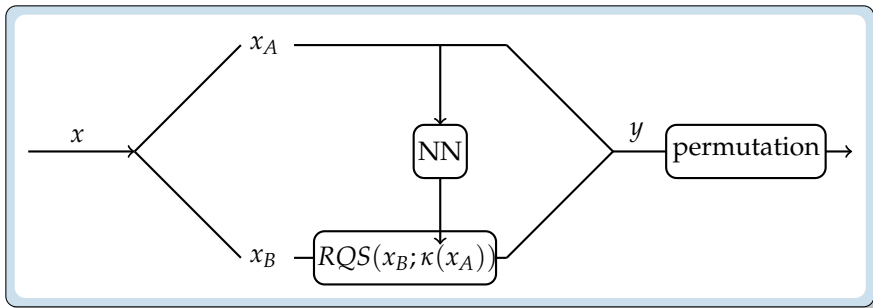
Triangular Jacobians 2: with Bipartite Blocks

$$\kappa_{x \in A}(x \in B) \quad \& \quad \kappa_{x \in B}(x \in A)$$

- Coordinates are split in 2 sets, transforming each other.

+ Forward and inverse pass are equally fast. - Said to be not as expressive.

Dinh et al. [arXiv:1410.8516]



A Classifier provides the “ultimate metric”.

According to the Neyman-Pearson Lemma we have:

- The likelihood ratio is the most powerful test statistic to distinguish the two samples.
- A powerful classifier trained to distinguish the samples should therefore learn (something monotonically related to) **this**.
- If this classifier is confused, we conclude $p_{\text{GEANT4}}(x) = p_{\text{generated}}(x)$

⇒ This captures the full phase space incl. correlations.

⇒ However, it is sufficient, but not necessary.

? But why wasn't this used before?

⇒ Previous deep generative models were separable to almost 100%!

DCTRGAN: Diefenbacher et al. [2009.03796, JINST]