Normalizing Flows for Physics Data Analysis

F9 Seminar
March 22, 2024

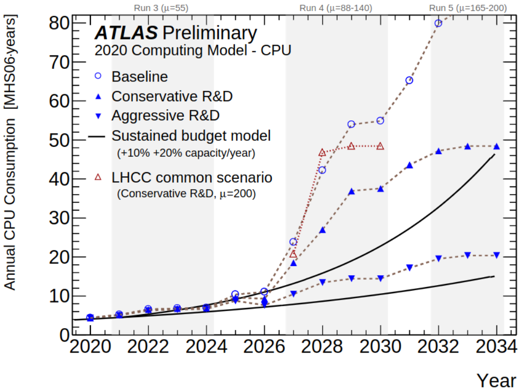
author: Jan Gavranovič

Jožef Stefan Institute and University of Ljubljana

jan.gavranovic@cern.ch

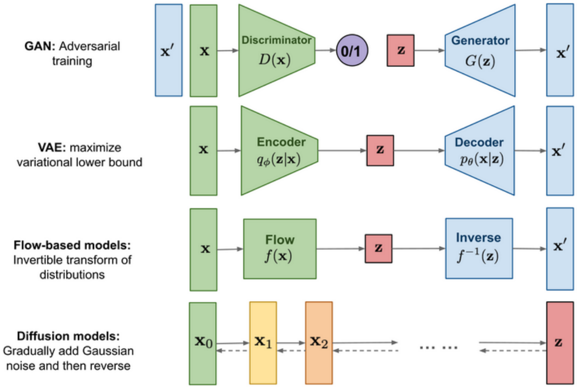
Introduction

- LHC produces **big data**
- MC and analysis need to follow
- Can generative models be used to support physics modeling?
- This talk: developing new analysis ideas with generative ML
- Focus on LHC *final* event simulation with **normalizing flows**:
  1. fast and precise once trained
  2. can be trained on combination of MC and actual data
  3. constructed to be easily invertible



Generative models

- Learn true $p_{\text{data}}(\boldsymbol{x})$ from $\boldsymbol{x} \in \mathbb{R}^D$ with approximate $p_{\text{model},\theta}(\boldsymbol{x}) \approx p_{\text{data}}(\boldsymbol{x})$



- **Problem**: do not know the true generating data distribution
- But have access to an empirical distribution through a finite amount of observations $\boldsymbol{x}$ (*events*)
- **Objective**: approximate $p_{\text{data}}(\boldsymbol{x})$ to enable infinite sampling

Normalizing flows (invertible neural networks)

- Two pieces:
  1. base distribution $p_u(\boldsymbol{u})$, typically something simple like $\mathcal{N}(\boldsymbol{u}|\mathbf{0}, \mathbf{I})$
  2. differentiable transformation $\boldsymbol{x} = T(\boldsymbol{u})$ with an inverse $\boldsymbol{u} = T^{-1}(\boldsymbol{x})$
- Construct a flow by composing together many transformations:

$$T = T_K \circ \ldots \circ T_1 \quad \text{and} \quad T^{-1} = T_1^{-1} \circ \ldots \circ T_K^{-1}$$

- Transformations T are (invertible) neural networks with parameters $\boldsymbol{\phi}$
- Generative process:

$$\boldsymbol{x} = T(\boldsymbol{u}) \approx p_x(\boldsymbol{x}) \quad \text{with sampling} \quad \boldsymbol{u} \sim p_u(\boldsymbol{u})$$

- Density evaluation (using change of variables formula):

$$p_x(\boldsymbol{x}) = p_u(T^{-1}(\boldsymbol{x})) \left| \det \frac{\partial T^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|$$

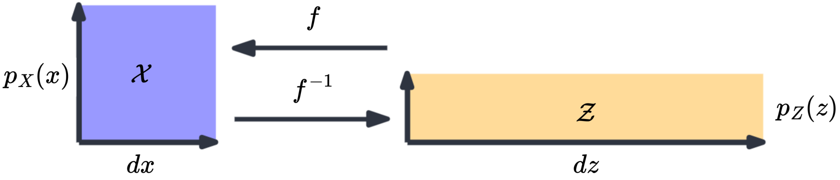
Change of variables trick

- Transformations expand the support of the distribution $\Rightarrow$ we need to scale densities to preserve the volume of probability mass
- For a 1D random variable $X = f(Z)$ with $Z = f^{-1}(X)$ we have:

$$p_X(x) = p_Z(f^{-1}(x)) \left| \frac{d}{dx} f^{-1}(x) \right|$$

- This comes from volume preservation constraint:

$$\int p_Z(z)dz = \int p_Z(z)\frac{dx}{dx}dz = \int p_Z(z)\left|\frac{dz}{dx}\right| dx = \int p_Z(f^{-1}(x)) \left| \frac{d}{dx} f^{-1}(x) \right| dx = 1$$



Jacobians and determinants

- For non-linear transformations $\boldsymbol{f}$, the linearized *change in volume* is given by the determinant of the Jacobian of $\boldsymbol{f}$
- For $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ and $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})]$ the Jacobian is

$$J_{\boldsymbol{f}}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad \text{or} \quad J_{ij} = \frac{\partial f_i}{\partial x_j}$$

- This generalizes the gradient to multi-variate functions
- Change of variables in a general case for $X = \boldsymbol{f}(Z)$ with $Z = \boldsymbol{f}^{-1}(X)$:

$$p_X(\boldsymbol{x}) = p_Z(\boldsymbol{f}^{-1}(\boldsymbol{x})) \left| \det \frac{\partial \boldsymbol{f}^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right|$$

- Computational complexity for determinant of $n \times n$ matrix is $\mathcal{O}(n^3)$
- Flows are designed to have **triangular Jacobians** to simplify this

Summary of the ingredients

- What do we need?

1. Base distribution tha we know how to sample from $\boldsymbol{u} \sim p_u(\boldsymbol{u})$
2. NN invertible transformation $\boldsymbol{x} = T(\boldsymbol{u})$ with $\boldsymbol{u} = T^{-1}(\boldsymbol{x})$ with parameters $\boldsymbol{\phi}$
3. Triangular Jacobian matrix for efficient determinant computation

$$J_{ij} = \frac{\partial T_i}{\partial x_j} = \begin{cases} \frac{\partial T_i}{\partial x_j} & ; \; i \geqslant j \\ 0 & ; \; i < j \end{cases}$$

- What can we do with this?
  1. Generation of new events

$$\boldsymbol{u} \sim p_u(\boldsymbol{u}) \to \boldsymbol{x} = T(\boldsymbol{u})$$

  2. Density estimation

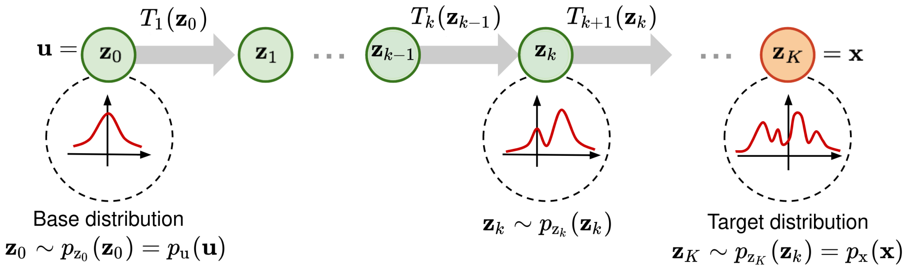
$$p_{\mathrm{x}}(\boldsymbol{x}) = p_u(T^{-1}(\boldsymbol{x}))\, |\det J_{T^{-1}}(\boldsymbol{x})|$$

- Idea: *learn* a transformation T that maps a simple distribution to a complex one

Forward and inverse directions

- **Forward direction:** $\boldsymbol{z}_k = T_k(\boldsymbol{z}_{k-1})$ for $k = 1, \ldots, K$ with $\boldsymbol{z}_0 = \boldsymbol{u}$ (infer)
- **Inverse direction:** $\boldsymbol{z}_{k-1} = T_k^{-1}(\boldsymbol{z}_k)$ for $k = K, \ldots, 1$ with $\boldsymbol{z}_K = \boldsymbol{x}$ (train)
- The log-determinant of a flow is

$$\log |\det J_T(\boldsymbol{z}_0)| = \log \left| \prod_{k=1}^{K} \det J_{T_k}(\boldsymbol{z}_{k-1}) \right| = \sum_{k=1}^{K} \log |\det J_{T_k}(\boldsymbol{z}_{k-1})|$$



- Similar to autoencoder: forward mode $\Leftrightarrow$ decoder, backward mode $\Leftrightarrow$ encoder

Loss function

- Use maximum likelihood estimation
- Fit a parametric flow model $T = p_x(\boldsymbol{x};\theta)$ to a target distribution $p_x(\boldsymbol{x})$
- Use average log-likelihood over N data points

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \log p_x(\boldsymbol{x}_n;\theta)\ .$$

- Density evaluation gives us log-likelihood of input data!
- Loss function has two terms (**log-likelihood + log-determinant**):

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \left[\log p_u\left(T^{-1}(\boldsymbol{x}_n;\phi);\psi\right) + \log\left|\det J_{T^{-1}}(\boldsymbol{x}_n;\phi)\right|\right]$$

- Use gradient descent to get best parameters

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ \mathcal{L}(\theta)\ , \quad \theta \equiv \{\phi,\psi\}$$

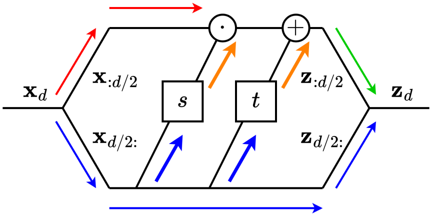
Coupling layer

- A coupling layer splits input vector $\boldsymbol{x} \in \mathbb{R}^D$ into two (*usually equal*) parts
- Transforms the second part as a function of the first part
  - Affine transformation: $\tau(z_i; \boldsymbol{h}_i) = s_i z_i + t_i$, $\boldsymbol{h}_i = \{s_i, t_i\}$
- Active upper lane and passive lower lane

- Forward direction:

$$\boldsymbol{z}_{\leqslant d} = \boldsymbol{x}_{\leqslant d}$$

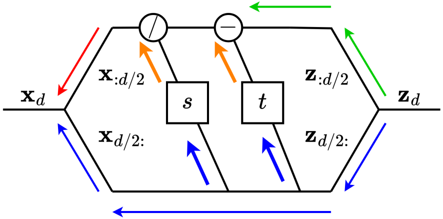
$$\boldsymbol{z}_{>d} = \boldsymbol{x}_{>d} \cdot \exp\left(s(\boldsymbol{x}_{\leqslant d})\right) + t(\boldsymbol{x}_{\leqslant d})$$



- Inverse direction:

$$\boldsymbol{x}_{\leqslant d} = \boldsymbol{z}_{\leqslant d}$$

$$\boldsymbol{x}_{>d} = \left(\boldsymbol{z}_{>d} - t(\boldsymbol{z}_{\leqslant d})\right) \cdot \exp\left(-s(\boldsymbol{z}_{\leqslant d})\right)$$



- Does not require computing inverses of s and t $\Rightarrow$ arbitrarily complex NN!

Coupling flow

- Jacobian is lower triangular with block like structure

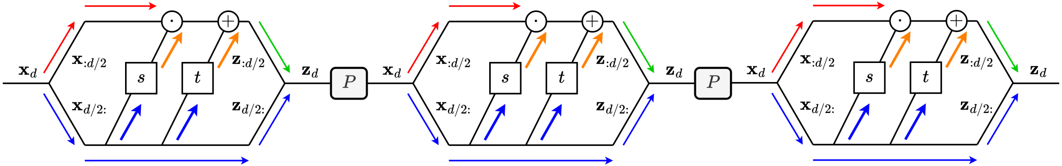
$$J = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L} & \mathbf{D} \end{bmatrix}$$

- Only relevant part is $\mathbf{D} \Rightarrow \mathcal{O}(d)$ time complexity for determinant!

$$\mathbf{D} = \operatorname{diag}\left[\exp(s(\boldsymbol{z}_{\leqslant d}))\right] \quad \text{with} \quad \log|\det J(\boldsymbol{z})| = \sum_j s(\boldsymbol{z}_{\leqslant d})_j$$

- Binary masks $\boldsymbol{b}$ for splitting and joining (*permutations*):

$$\boldsymbol{z} = \boldsymbol{b} \cdot \boldsymbol{x} + (1 - \boldsymbol{b}) \cdot (\boldsymbol{x} \cdot \exp(s(\boldsymbol{b} \cdot \boldsymbol{x})) + t(\boldsymbol{b} \cdot \boldsymbol{x}))$$
$$\boldsymbol{x} = \boldsymbol{b} \cdot \boldsymbol{z} + (1 - \boldsymbol{b}) \cdot (\boldsymbol{z} - t(\boldsymbol{b} \cdot \boldsymbol{z})) \cdot \exp(-s(\boldsymbol{b} \cdot \boldsymbol{z}))$$

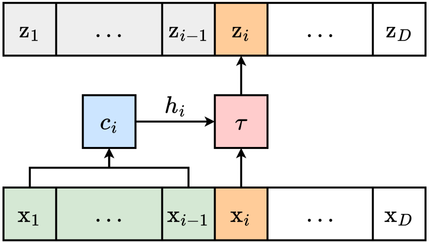


Autoregressive models

- Output at *time-step i* is conditioned on all the previous outputs
- Autoregressive model: $p_x(\boldsymbol{x}) = \prod_{i=1}^{D} p_x(x_i|\boldsymbol{x}_{<i}) \Rightarrow$ chain rule of probability

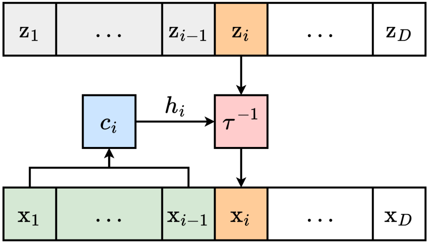
- Forward direction:

$$\boldsymbol{z}_i = \tau(\boldsymbol{z}_i; \boldsymbol{h}_i) \quad \text{with} \quad \boldsymbol{h}_i = c_i(\boldsymbol{x}_{<i}; \boldsymbol{\phi})$$



- Inverse direction:

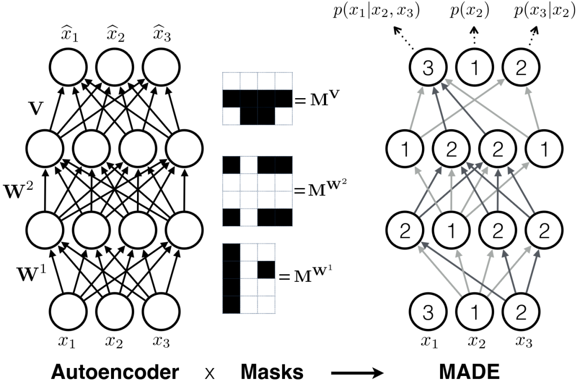
$$\boldsymbol{x}_i = \tau^{-1}(\boldsymbol{z}_i; \boldsymbol{h}_i) \quad \text{with} \quad \boldsymbol{h}_i = c_i(\boldsymbol{x}_{<i}; \boldsymbol{\phi})$$



- Each $\boldsymbol{z}_i$ does not depend on $\boldsymbol{x}_{>i} \Rightarrow \frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{x}_j} = 0$ for $j > i \Rightarrow$ triangular Jacobian

Masked conditioners

- The most popular technique for implementing autoregressive flows
- Output $\hat{\boldsymbol{x}}_i$ only depends on the previous inputs $\boldsymbol{x}_{<i}$ and not on the $\boldsymbol{x}_{\geqslant i}$
- Multiply each weight matrix with a binary matrix $\Rightarrow$ remove connections



$$p(x) = p^{(1)}(x_2)p^{(2)}(x_3|x_2)p^{(3)}(x_1|x_2,x_3)$$

1. Assign each unit in each hidden layer an *integer degree* d_k^l
2. Connect a unit to previous units whose degrees do not exceed its own
3. Do this with *masking* matrices:

$$\mathbf{W}_{ij}^l = \begin{cases} 1 & \text{if } d_i^l \geqslant d_j^{l-1} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{V}_{ij}^L = \begin{cases} 1 & \text{if } d_i^L > d_j^{L-1} \\ 0 & \text{otherwise} \end{cases}$$

Masked autoregressive flow

- Autoregressive model with Gaussian conditionals
- The i-th conditional is given by

$$p(\boldsymbol{z}_i|\boldsymbol{z}_{<i}) = \mathcal{N}(\boldsymbol{z}_i; \boldsymbol{\mu}_i, (\exp \boldsymbol{\alpha}_i)^2) \quad \text{with} \quad \boldsymbol{\mu}_i = f_{\mu}(\boldsymbol{z}_{<i}) \quad \text{and} \quad \boldsymbol{\alpha}_i = f_{\alpha}(\boldsymbol{z}_{<i})$$

- Forward direction:

$$\boldsymbol{z}_i = \boldsymbol{u}_i \cdot \exp \boldsymbol{\alpha}_i + \boldsymbol{\mu}_i$$
$$\text{with} \quad \boldsymbol{u}_i \sim \mathcal{N}(0, 1)$$

- Inverse direction:

$$\boldsymbol{u}_i = (\boldsymbol{z}_i - \boldsymbol{\mu}_i) \cdot \exp(-\boldsymbol{\alpha}_i)$$

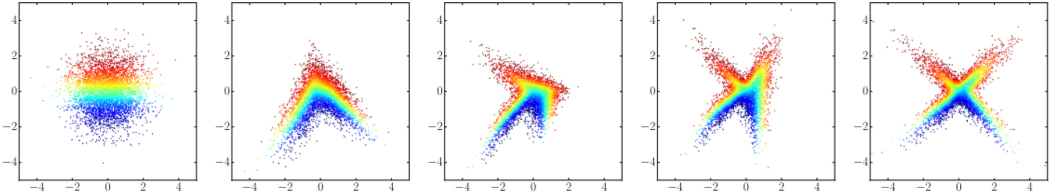
- Due to the autoregressive structure, the Jacobian is lower triangular

$$\log \left| \det \frac{\partial T^{-1}}{\partial \boldsymbol{z}} \right| = -\sum_{i=1}^{D} \boldsymbol{\alpha}_i$$

- f_{μ} and f_{α} are implemented as *masked neural networks*

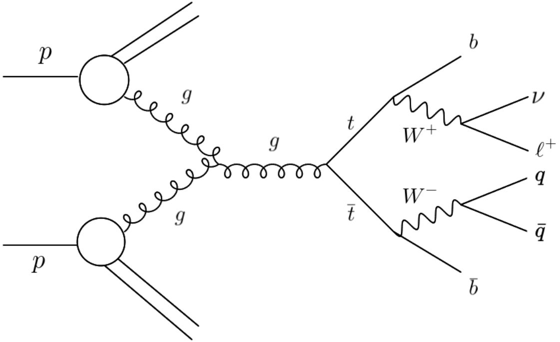
Summary of normalizing flows

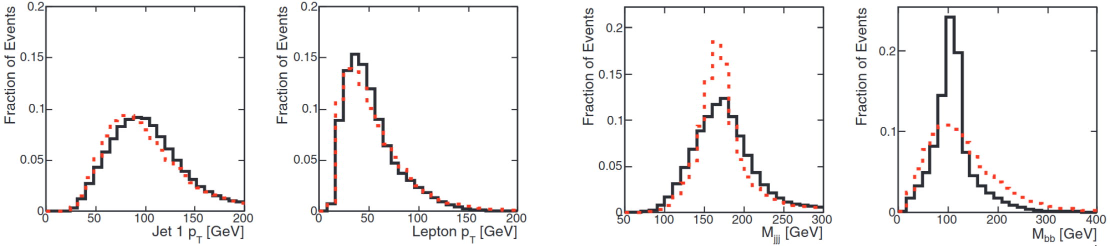
- *Normalizing* flow $T_K^{-1} \circ \ldots \circ T_1^{-1}$ takes samples from $p_x(\boldsymbol{x})$ and transforms (*normalizes*) them into samples from the prescribed base distribution $p_u(\boldsymbol{u})$
- Loss function has two terms (log-likelihood + log-determinant)
- Main goal: build efficient and expressive transformations using neural networks
- Examples: coupling layers (RealNVP) and masked autoregressive flows (MAF)



HIGGS dataset benchmark

- Publicly available dataset with 11M events and 28 variables
- Binary classification problem: signal (BSM) vs. background ($t\bar{t}$)
- 21 low-level and 7 high-level variables
- **Task**: train ML model to generate *new background events*





Feature scaling

- Reduce the modeling complexity that is required by the flow
- Gradient descent converges much faster with feature scaling

- Continuous features $\boldsymbol{x} \in \mathbb{R}^N$
  1. **min-max normalization**

$$\boldsymbol{x} = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})} \in [0, 1]$$

- Discrete features $\boldsymbol{x} \in \mathbb{N}^M$
  1. **add noise** $\epsilon \sim \mathcal{U}(0, 1)$

$$\boldsymbol{x} = \frac{\boldsymbol{x} + \epsilon}{\max(\boldsymbol{x}) + 1} \in [0, 1]$$

2. **clip values for numerical stability**

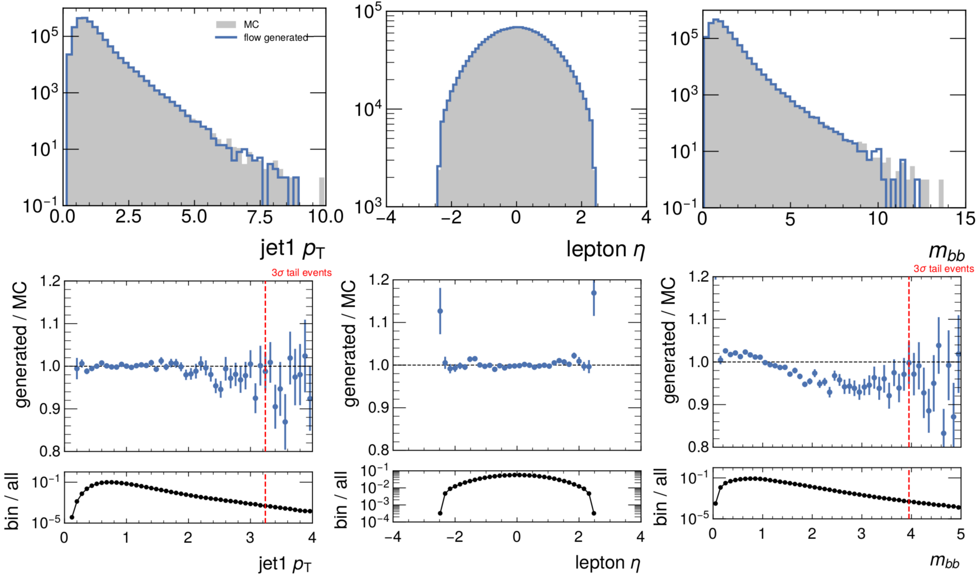
$$\boldsymbol{x} = \boldsymbol{x}(1 - \beta) + \frac{1}{2}\beta \in (0, 1) \quad \text{where} \quad \beta = 10^{-6}$$

3. **logit transformation with standardization**

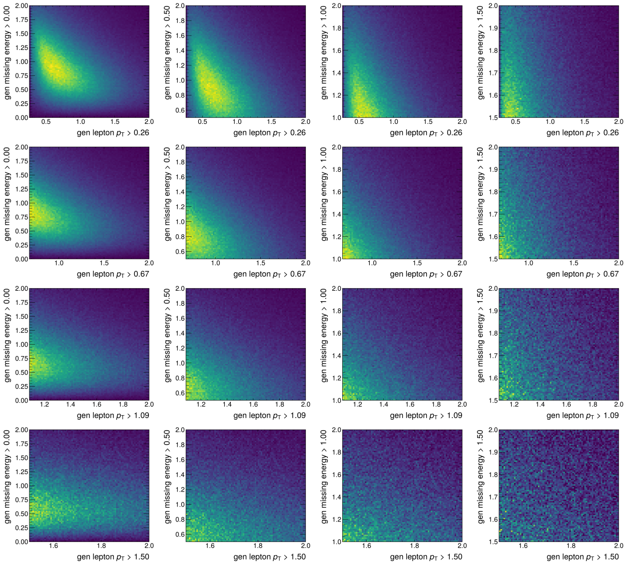
$$\boldsymbol{x} = \log \frac{\boldsymbol{x}}{1 - \boldsymbol{x}} \in (-\infty, \infty) \quad \text{and} \quad \boldsymbol{x} = \frac{\boldsymbol{x} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$$

- Can get back to the original feature space with inverse functions

Learning event distributions



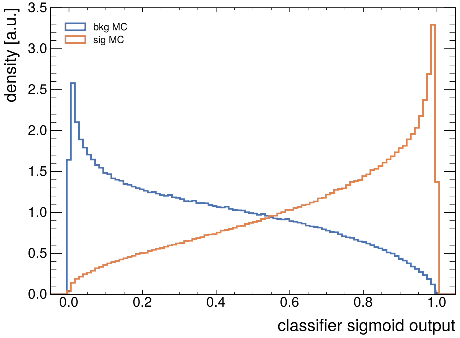
Learning variable correlations

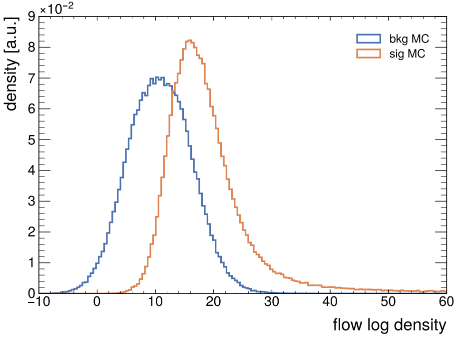


- Correlations for two variables in leptonic *W* decay
- Check generated event invariance to variable cuts

Classification with density estimation

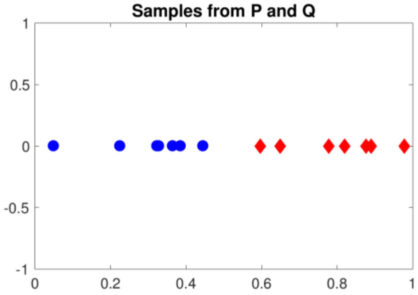
- **Idea:** train flow on background events and estimate density for signal events
- Unsuperivsed learning (need only background) ⇒ anomaly detection
- Can be used as a classifier with *density score* as the output

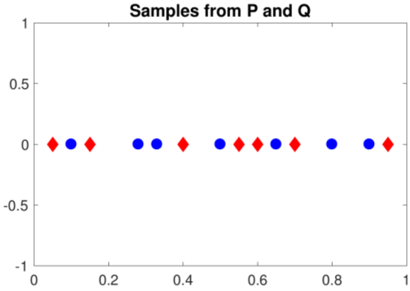




Two-sample testing

- How to tell if the generative model is any good?
- **Have**: two sets of samples X and Y from unknown distributions P and Q
- **Goal**: answer the question *are P (MC) and Q (ML) the same?*
- Two-sample test: determining if the samples come from the same distribution





f-divergence

- Compare distributions with density ratios $r(\boldsymbol{x}) = p(\boldsymbol{x})/q(\boldsymbol{x})$ using

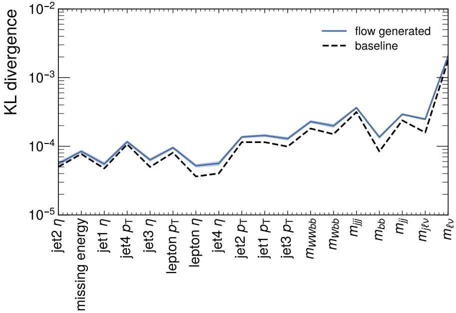
$$D_f(p\|q) = \int p(\boldsymbol{x}) f\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x}$$

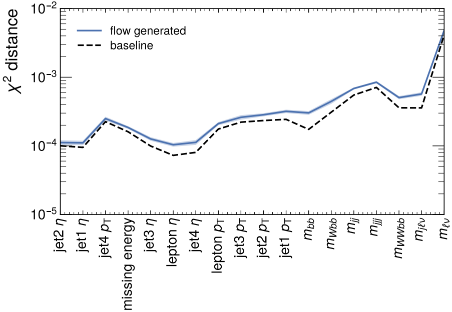
- $D_f(p\|q) \geqslant 0$ and $D_f(p\|p) = 0$
- KL divergence:

$$D_{KL}(p\|q) = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}$$

- χ^2 distance:

$$\chi^2(p, q) = \frac{1}{2} \int \frac{(p(\boldsymbol{x}) - q(\boldsymbol{x}))^2}{q(\boldsymbol{x})} d\boldsymbol{x}$$





Classifier two sample test

- Idea: accuracy of a binary classifier will be 50:50 if we train it on two samples coming from the same distribution

1. Construct a dataset with binary labels from two samples $X \sim P$ and $Y \sim Q$

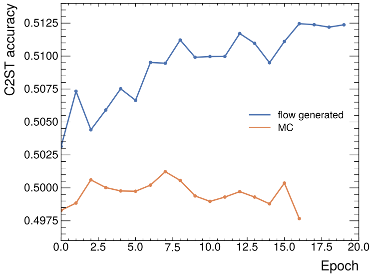
$$\mathcal{D} = \{(\boldsymbol{x}_i, 0)\}_{i=1}^{n} \cup \{(\boldsymbol{y}_i, 1)\}_{i=1}^{n} = \{\boldsymbol{z}_i, \boldsymbol{l}_i\}_{i=1}^{2n}$$

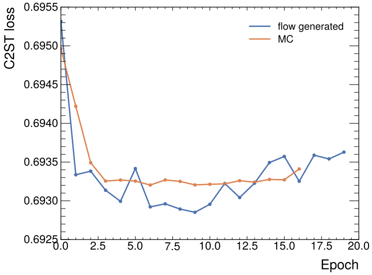
2. Shuffle $\mathcal{D}$ and split it into training and holdout sets $\mathcal{D} = \mathcal{D}_t \cup \mathcal{D}_h$
3. Train a binary classifier $D_\theta(\boldsymbol{z}_i) \approx p(\boldsymbol{l}_i = 1|\boldsymbol{z}_i)$ on $\mathcal{D}_t$ to predict $\boldsymbol{l}_i$ from $\boldsymbol{z}_i$
4. Return classification accuracy on $\mathcal{D}_h$

$$\hat{t} = \frac{1}{n_h} \sum_{(\boldsymbol{z}_i, \boldsymbol{l}_i) \in \mathcal{D}_h} \mathbb{I}\left[\mathbb{I}\left(D_\theta(\boldsymbol{z}_i) > \frac{1}{2}\right) = \boldsymbol{l}_i\right]$$

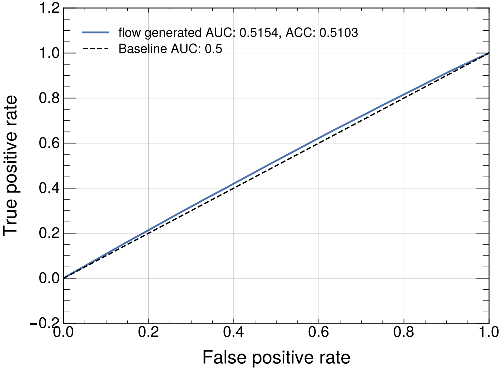
5. Use $\hat{t}$ as a test statistic for testing the null hypothesis $H_0 : P = Q$

Training a confused classifier





- Train a small NN binary classifier on the two-sample dataset
- Use binary cross-entropy loss with sigmoid output activation



Density ratio estimation trick

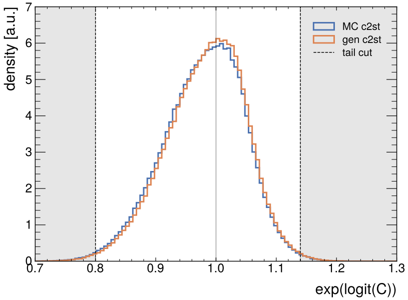
- Where does the generative model fail?
- We can extract *density ratio* $r(\boldsymbol{x}) = P(\boldsymbol{x})/Q(\boldsymbol{x})$ from the binary classifier
- Look at events where $r(\boldsymbol{x})$ is large/small $\Rightarrow$ generative model failure modes
- By Bayes' rule with a prior $p(\boldsymbol{y}=1) = \pi = \frac{1}{2}$ we have:

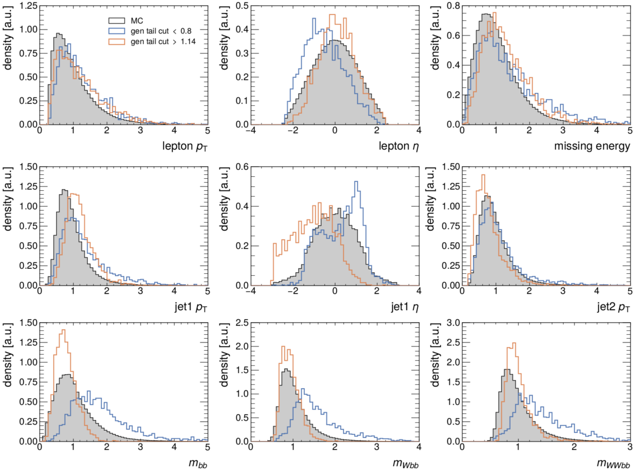
$$
\begin{aligned}
r(\boldsymbol{x}) &= \frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\boldsymbol{y}=1)}{p(\boldsymbol{x}|\boldsymbol{y}=0)} = \frac{p(\boldsymbol{y}=1|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y}=1)} \Big/ \frac{p(\boldsymbol{y}=0|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y}=0)} \\
&= \frac{p(\boldsymbol{y}=1|\boldsymbol{x})}{p(\boldsymbol{y}=0|\boldsymbol{x})} \frac{\pi}{1-\pi} = \frac{p(\boldsymbol{y}=1|\boldsymbol{x})}{1-p(\boldsymbol{y}=1|\boldsymbol{x})} = \exp\left[\log \frac{p(\boldsymbol{y}=1|\boldsymbol{x})}{1-p(\boldsymbol{y}=1|\boldsymbol{x})}\right] \\
&= \exp\{\sigma^{-1}[p(\boldsymbol{y}=1|\boldsymbol{x})]\} \approx \exp\{\sigma^{-1}[D_\theta(\boldsymbol{x})]\}
\end{aligned}
$$

- **What does this mean?** Density ratio is given by the exponential of the inverse sigmoid function of the classifier output trained on the two-sample dataset using binary cross-entropy loss

Generative model failure modes

- Cut on the density ratio distribution $r(\boldsymbol{x})$ tails
- Look at the coresponding events
- *Anomaly detection* on the generative model





Conclusion

- HEP data is complex and high-dimensional
- Machine learning is a huge field with many applications that can be used in HEP
- Talked only about a very small part of generative modeling: *normalizing flows*
- Flows are a powerful tool for the type of data we have:
    1. Precise with both sampling and density estimation
    2. Fast to train and evaluate on MC or data
    3. Interpretable with likelihoods and Jacobians
- Need to be very careful with generative models and not use them as a black box

Thank you!