Use of Normalizing Flows in Particle Physics Simulations

Seminar 2

author: Jan Gavranovič
advisor: prof. dr. Borut Paul Kerševan

University of Ljubljana
Faculty of Mathematics and Physics

April 6, 2022

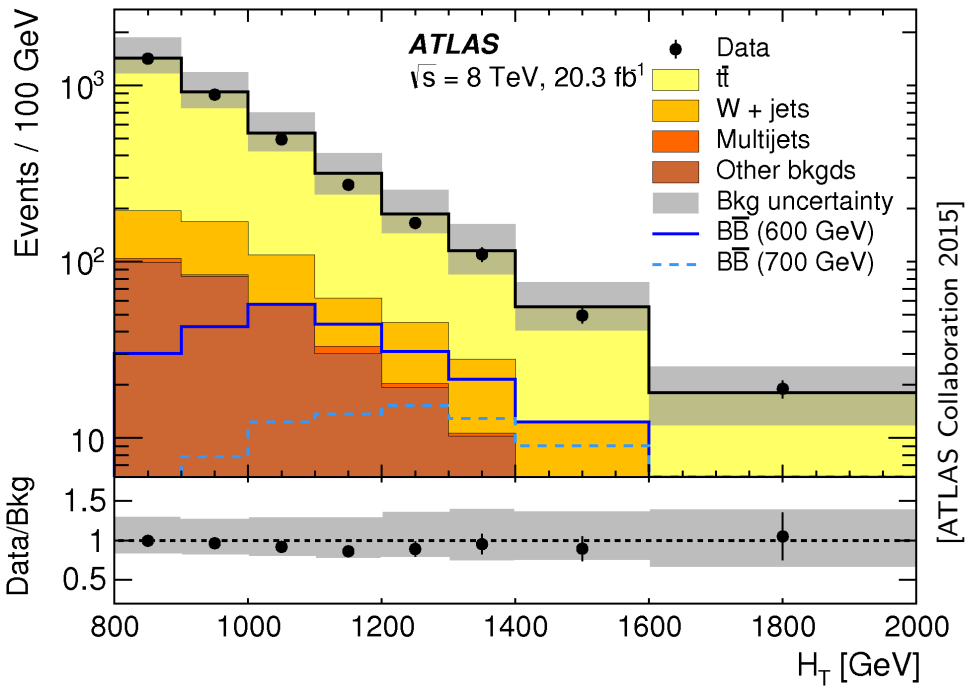
Overview

1. Computing at the LHC
2. Simulated Higgs boson production
3. Normalizing flows
4. Affine coupling layers
5. ML event generation
6. Conclusion and discussion

Computing at the LHC

- LHC: proton-proton collisions with center of mass energy of 14 TeV.
- Use of Monte Carlo (MC) simulations.
- Characterize signal hypotheses and study background processes.
- The simulation chain:
  1. **event generation** - simulate *pp* interactions using theoretical predictions,
  2. **detector simulation** - particle interactions with the material,
  3. **digitization** - conversion of hits into detector responses,
  4. **reconstruction/analysis**.
- Need large MC statistics for precise predictions ⇒ large computational resources.

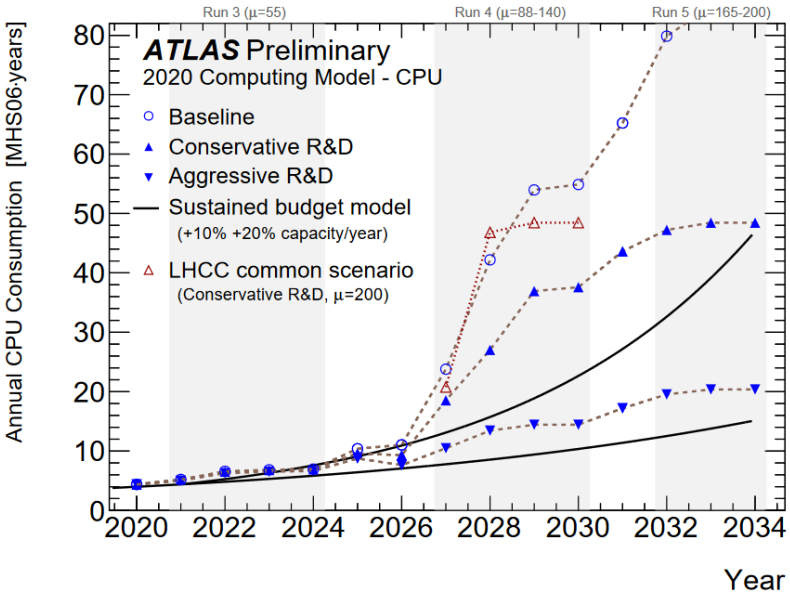
- Example: vector-like B quarks search.

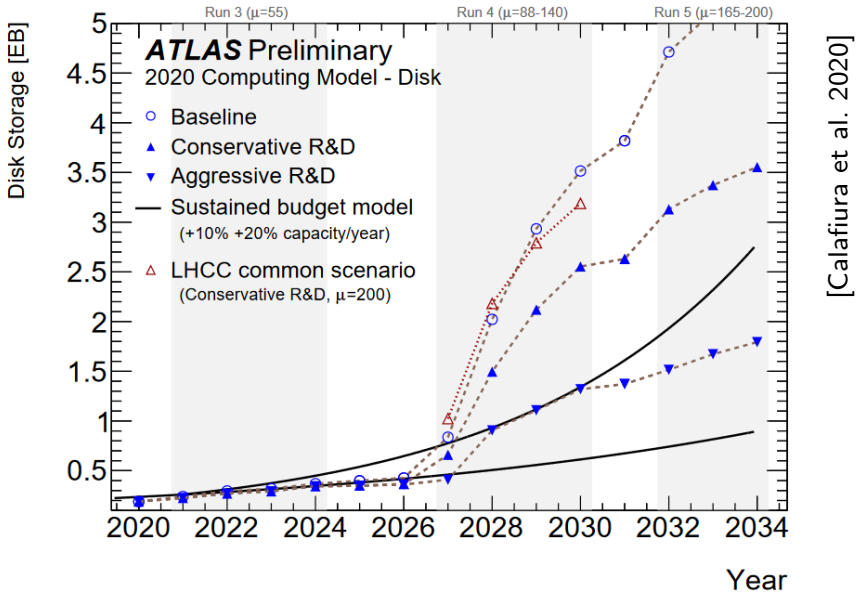


- High cost of CPU time and disk space ⇒ restricts the discovery process.

Computing at the LHC

- MC event generation takes up to 10 minutes *per event*.
- Produces $\sim$ 1 MB of data (only $\sim$ 1 kB useful in the final analysis).
- Computing resources provided by the Worldwide LHC Computing Grid.
- Upcoming Run 3 and in the future HL-LHC $\Rightarrow$ need more computing power.





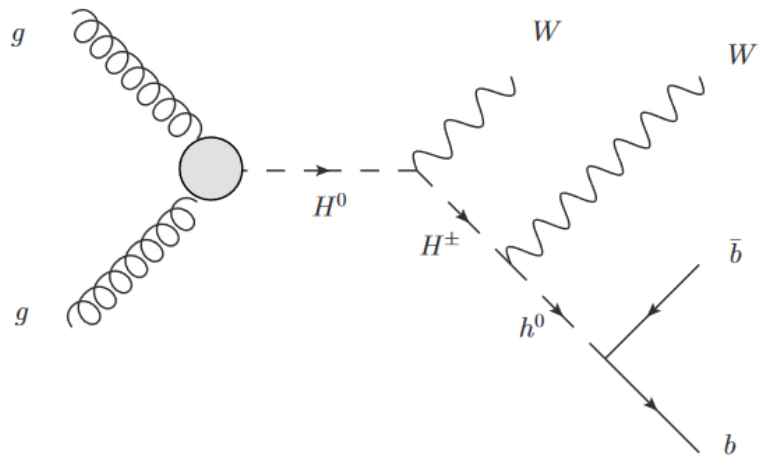
- Promising solution: use Machine Learning (ML).
- **Idea**: create large amounts of events at limited computing cost using a learning algorithm trained on a small set of MC simulated events.

Simulated Higgs boson production

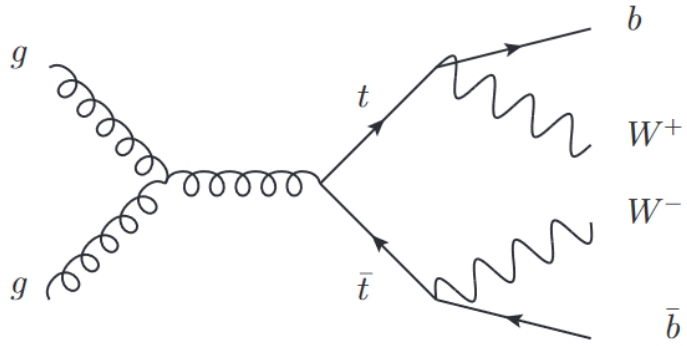
- **Signal**: theoretical Beyond-Standard-Model Higgs boson production

$$gg \to H^0 \to W^{\mp} H^{\pm} \to W^{\mp} W^{\pm} h^0 \to W^{\mp} W^{\pm} b\bar{b}\,.$$

- **Background**: identical decay products but distinct kinematic features.



(a) Signal process H^0 and $H^{\pm}$.



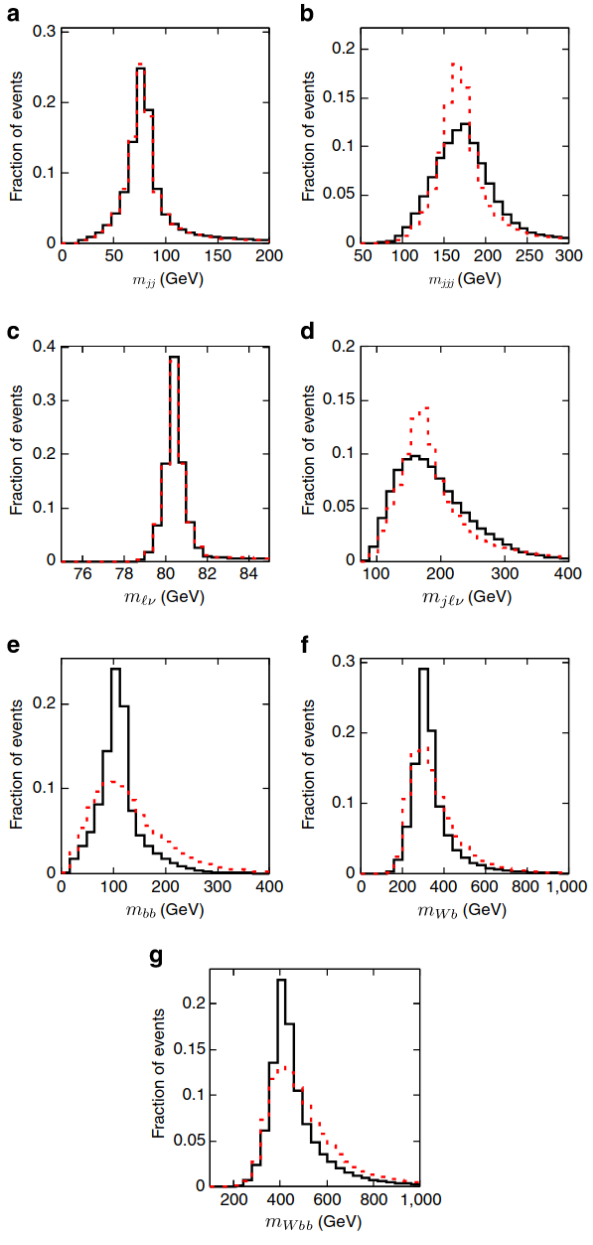
(b) Background top quark process.

[Baldi et al. 2014]

- Generated events assuming 8 TeV pp collisions at the LHC.

Simulated Higgs boson production

- **Idea:** Teach ML to simulate the final kinematic quantities used in the analysis.
- Observable decay products:
  1. leptons ℓ (electrons or muons),
  2. particle jets j (streams of particles from quarks or gluons).
- One W boson $\to \ell\nu$, other W boson $\to jj$
- Decay products: $\ell\nu b$ and jjb.
- Low-level features:
  1. jet p_T, η and azimuthal angle ϕ,
  2. jet b-tag,
  3. lepton p_T, η and ϕ,
  4. missing energy.
- High-level features:
  m_{jj}, m_{jjj}, $m_{l\nu}$, $m_{jl\nu}$, $m_{b\bar{b}}$, $m_{Wb\bar{b}}$, $m_{WWb\bar{b}}$.
- 18 dimensional feature space for the ML model to learn.



[Baldi et al. 2014]

Normalizing flows: basics

- Model probability distributions using samples.
- This is **generative modeling** and is a part of unsupervised learning.
- Distribution $p_{\mathbf{x}}(\mathbf{x})$ from which our data $\mathbf{x} \in \mathbb{R}^D$ was drawn from?

Base distribution

Let $\mathbf{u} \in \mathbb{R}^D$ be a random vector with a *known* probability density function (PDF) $p_{\mathbf{u}}(\mathbf{u}) : \mathbb{R}^D \to \mathbb{R}$. Distribution $p_{\mathbf{u}}(\mathbf{u})$ is called a *base distribution* and is usually taken to be normal $\mathcal{N}(\mathbf{0}, \mathbf{I}_{D\times D})$.

- Express $\mathbf{x}$ as a transformation T of a random variable $\mathbf{u}$:

$$\mathbf{x} = T(\mathbf{u}), \quad \mathbf{u} \sim p_{\mathrm{u}}(\mathbf{u}).$$

- T must be a *diffeomorphism* $\Rightarrow$ invertible with differentiable T and T^{-1}.
- $p_{\mathrm{x}}(\mathbf{x})$ can then be calculated using the **change of variables formula**:

$$\begin{aligned} p_{\mathrm{x}}(\mathbf{x}) &= p_{\mathrm{u}}\left(T^{-1}(\mathbf{x})\right) \left|\det J_T\left(T^{-1}(\mathbf{x})\right)\right|^{-1} \\ &= p_{\mathrm{u}}\left(T^{-1}(\mathbf{x})\right) \left|\det J_{T^{-1}}(\mathbf{x})\right| . \end{aligned}$$

Normalizing flows: basics

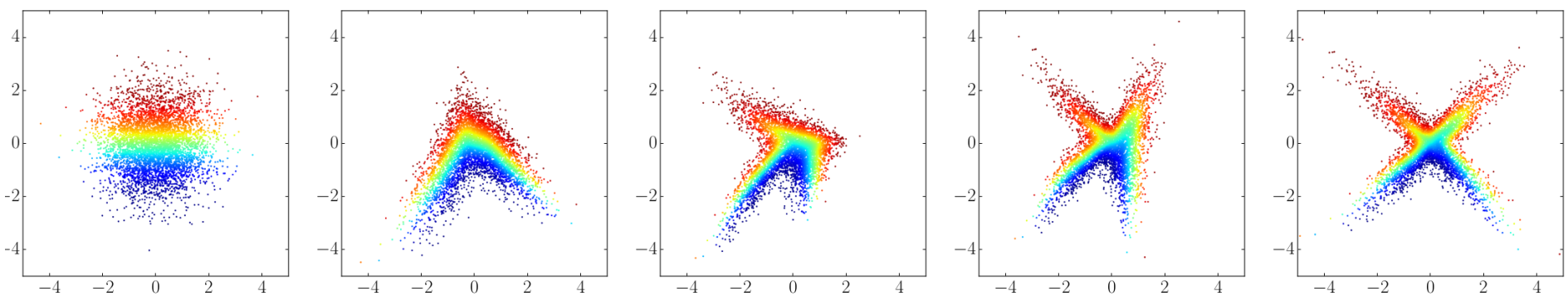
- Invertible and differentiable transformations T are composible:

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

- Construct a **flow** by chaining together many transformations.
- Build a complicated transformation T from many simpler transformations.

Normalizing flow

A trajectory of samples from the base (normal) distribution $p_{\mathbf{u}}(\mathbf{u})$ as they get gradually transformed by each transformation to the target distribution $p_{\mathbf{x}}(\mathbf{x})$.



[Papamakarios et al. 2021]

Normalizing flows: basics

- Forward direction:

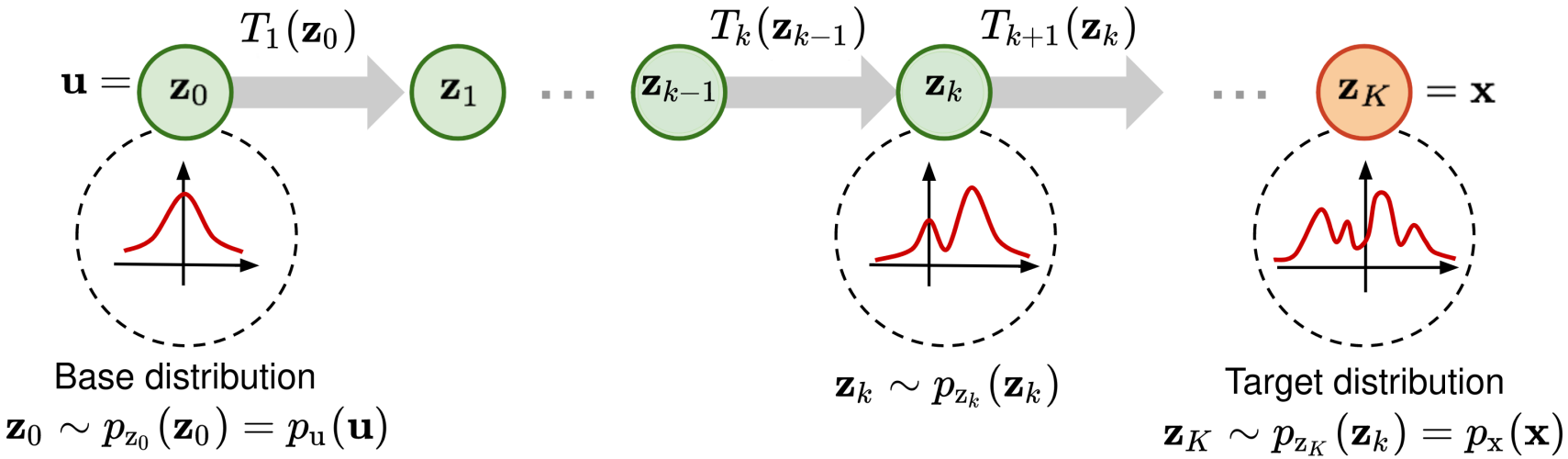
$$\mathbf{z}_k = T_k(\mathbf{z}_{k-1}) \quad \text{for} \quad k = 1, \ldots, K \quad \text{with} \quad \mathbf{z}_0 = \mathbf{u}\,.$$

- Inverse direction:

$$\mathbf{z}_{k-1} = T_k^{-1}(\mathbf{z}_k) \quad \text{for} \quad k = K, \ldots, 1 \quad \text{with} \quad \mathbf{z}_K = \mathbf{x}\,.$$

- The log-determinant of a flow is given by

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



[Weng 2018]

Normalizing flows: learning objective

- Use maximum likelihood estimation.
- Fit a parametric flow model $p_{\mathrm{x}}(\mathbf{x}; \boldsymbol{\theta})$ to a target distribution $p^*_{\mathrm{x}}(\mathbf{x})$.
- Use average log-likelihood over N data points

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

- Exact log-likelihood of input data is avaliable to us!

Loss function

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

Optimized over batches using gradient based methods during training.

Flow design

Machine learning flow model f_ϕ design

1. is invertible and differentiable,
2. has efficiently computable inverse and Jacobian determinant $\Rightarrow \mathcal{O}(D)$.

- General flow implementation form:

$$z'_i = \tau(z_i; \boldsymbol{h}_i)\,, \quad z_i = \tau^{-1}(z'_i; \boldsymbol{h}_i)\,, \quad \boldsymbol{h}_i = c_i(\mathbf{z}_{<i})\,.$$

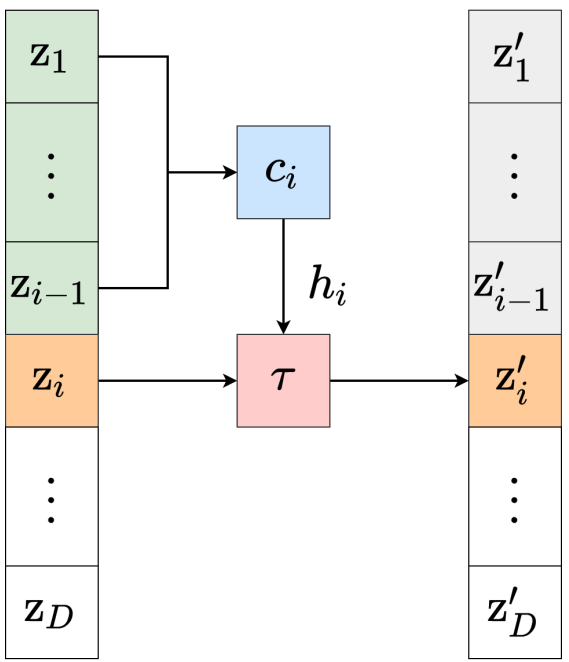
- Autoregressive flow type.

- This makes Jacobian of f_ϕ triangular

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial \tau}{\partial z_1}(z_1; \boldsymbol{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial \tau}{\partial z_D}(z_D; \boldsymbol{h}_D) \end{bmatrix}.$$

- Log-determinant uses only diagonal terms

$$\log\left|\det J_{f_\phi}(\mathbf{z})\right| = \sum_{i=1}^{D} \log\left|\frac{\partial \tau}{\partial z_i}(z_i; \boldsymbol{h}_i)\right|\,.$$



[Papamakarios et al. 2021]

Affine transformer

- Use a simple transformation

$$\tau(z_i; \boldsymbol{h}_i) = \alpha_i z_i + \beta_i \,, \quad \boldsymbol{h}_i = \{\alpha_i, \beta_i\} \,,$$

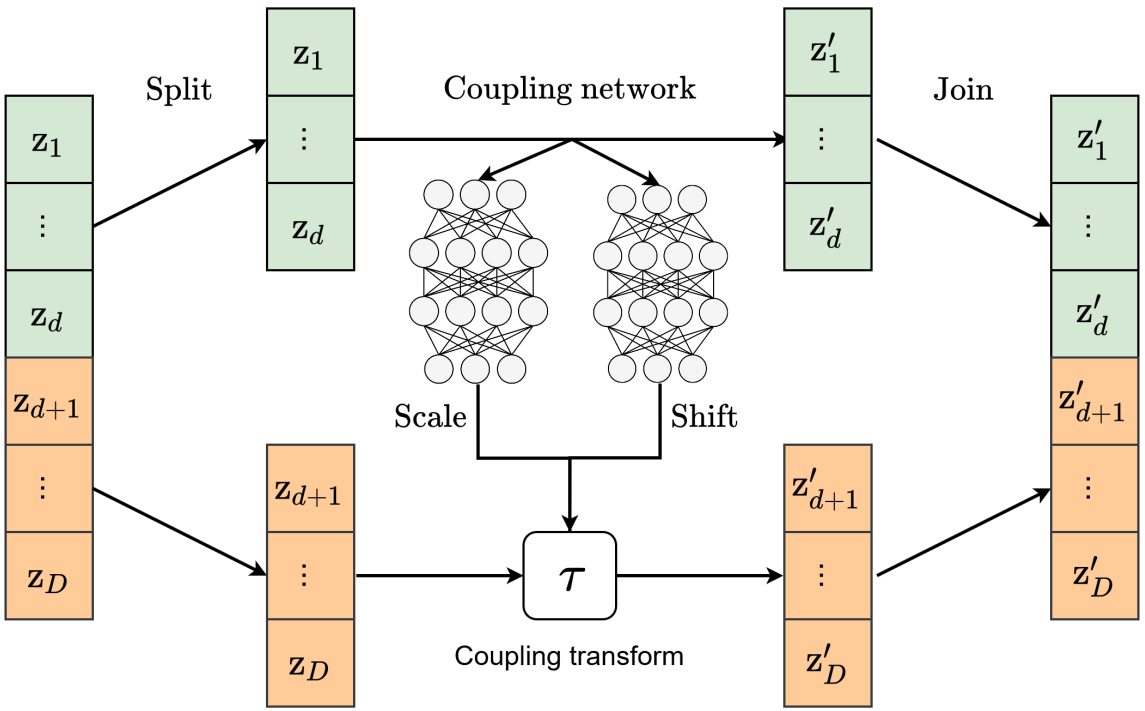
  which scales and shifts the input $\mathbf{z}_{<i}$.
- Invertibility enforced by $\alpha_i = \exp \tilde{\alpha}_i \Rightarrow \boldsymbol{h}_i = \{\tilde{\alpha}_i, \beta_i\}$.
- Jacobian determinant is

$$\log \left| \det J_{f_\phi}(\mathbf{z}) \right| = \sum_{i=1}^{D} \log |\alpha_i| = \sum_{i=1}^{D} \tilde{\alpha}_i \,.$$

- Scale and shift transformations implemented by two neural networks.
- **Downside**: has limited expressiveness $\Rightarrow$ stacking multiple affine layers.

Coupling layer

- A coupling layer splits $\mathbf{z}$ into two parts at index d and transforms the second part as a function of the first part.
- Use *real-valued non-volume preserving (realNVP)* model.



[Dinh et al. 2016]

- Forward direction:

$$\mathbf{z}'_{\leq d} = \mathbf{z}_{\leq d}$$
$$\boldsymbol{h}_{>d} = \{s(\mathbf{z}_{\leq d}),\ t(\mathbf{z}_{\leq d})\}$$
$$\mathbf{z}'_{>d} = \mathbf{z}_{>d} \odot \exp\left(s(\mathbf{z}_{\leq d})\right) + t(\mathbf{z}_{\leq d})$$

- Jacobian is lower triangular

$$J_{f_\phi} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{D} \end{bmatrix}$$

with

$$\mathbf{D} = \operatorname{diag}\left[\exp(s(\mathbf{z}_{\leq d}))\right] .$$

Permutation layer

- $\mathbf{z}_{<d}$ remains unchanged when stacking coupling layers into a flow.
- **Solution**: reverse the ordering of output features.
- We can do better! Try to learn feature permutations.
- **Idea**: learn a generalization of a permutation matrix $\mathbf{z}' = \mathbf{W}\mathbf{z}$.
- **Problem**: need to calculate $\log|\det\mathbf{W}|$ in a flow $\Rightarrow \mathcal{O}(D^3)$.
- Use LU to reduce this to $\mathcal{O}(D)$:
  1. initialize $\mathbf{W}$ as a random rotation matrix
  $$\mathbf{W}, \mathbf{R} = \mathrm{QR}\left[\mathcal{N}(0, \mathbf{I}_{D\times D})\right] ,$$
  2. parameterize $\mathbf{W}$ by its LU decomposition
  $$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s})) ,$$
  3. the determinant is then
  $$\log|\det\mathbf{W}| = \sum_{i=1}^{D} \log|\mathbf{s}_i| ,$$
  4. learning is performed on $\mathbf{L}$, $\mathbf{U}$ and $\mathbf{s}$ with $\mathbf{P}$ fixed.

Data preprocessing and flow blocks

- Features have very different scales $\Rightarrow$ poor performance.
- Perform data normalization (feature scaling) before training.
- This was done in three steps:
  1. min-max normalization

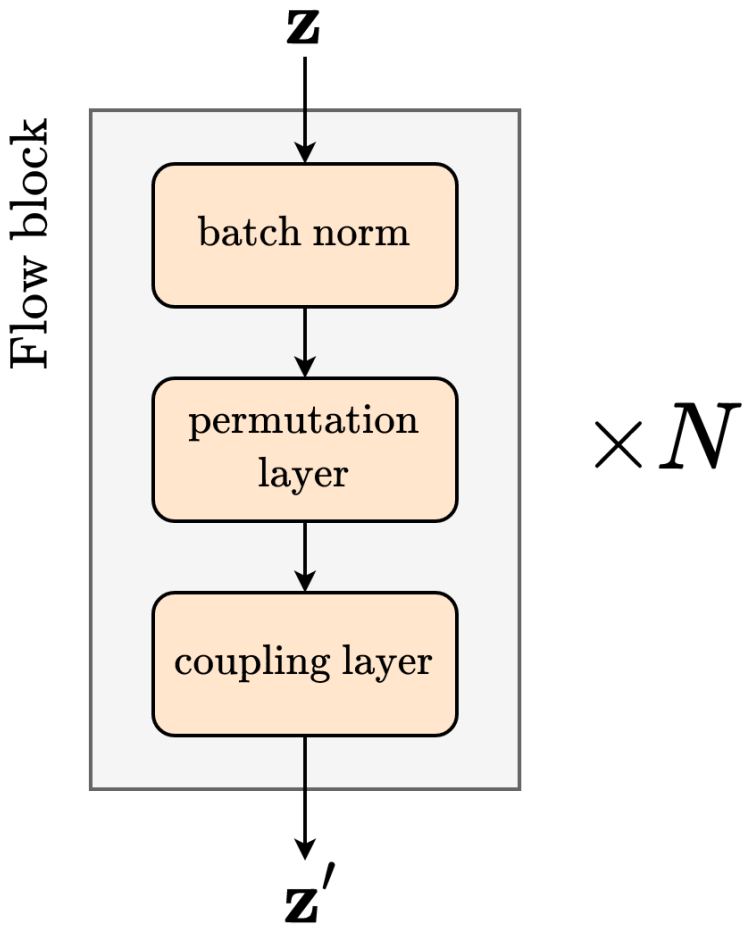
$$\mathbf{x}_1 = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} ,$$

  2. logit transformation

$$\mathbf{x}_2 = \log \frac{\mathbf{x}_1}{1 - \mathbf{x}_1} ,$$

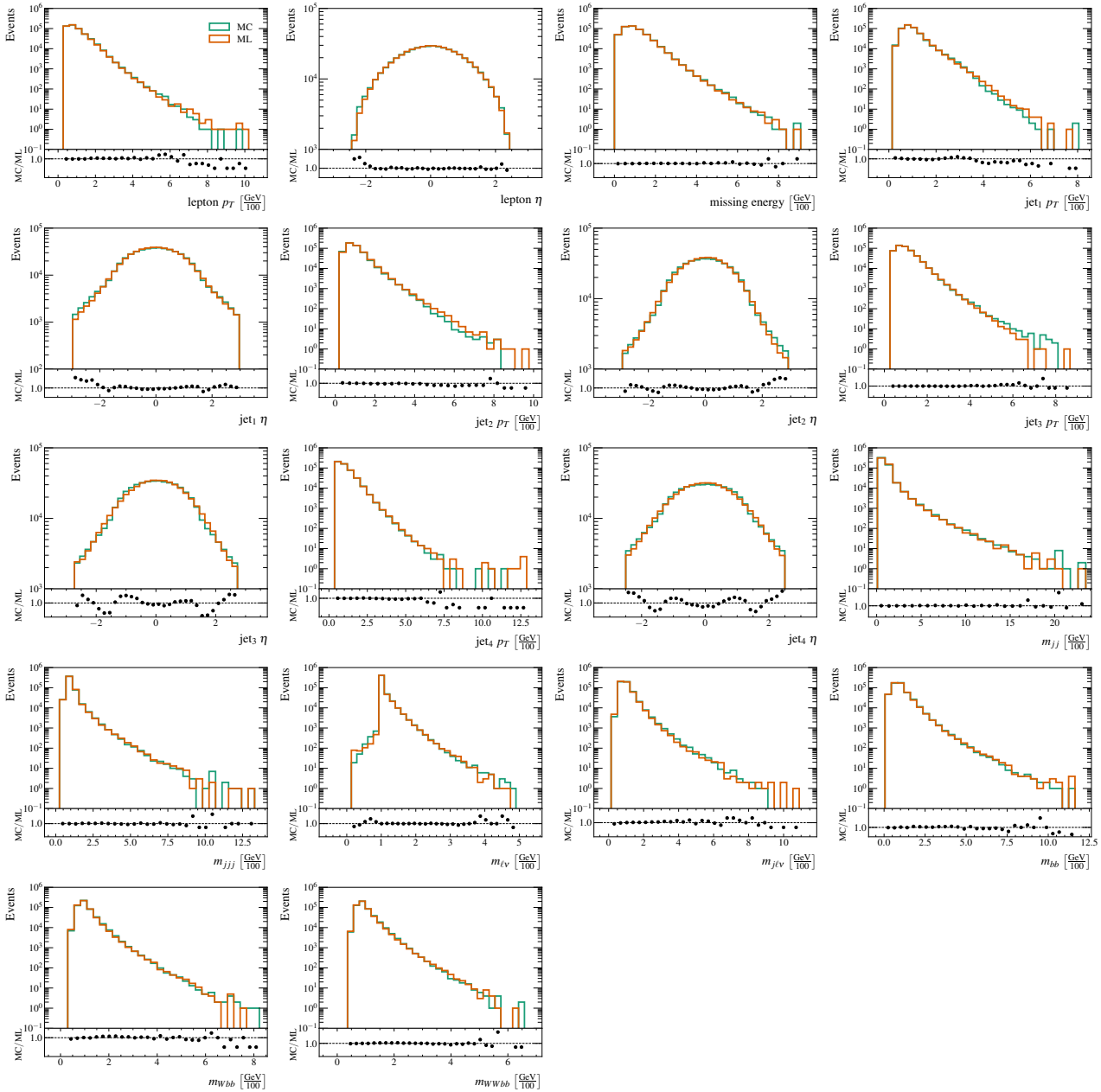
  3. standardization

$$\mathbf{x}_3 = \frac{\mathbf{x}_2 - \boldsymbol{\mu}(\mathbf{x}_2)}{\boldsymbol{\sigma}(\mathbf{x}_2)} .$$



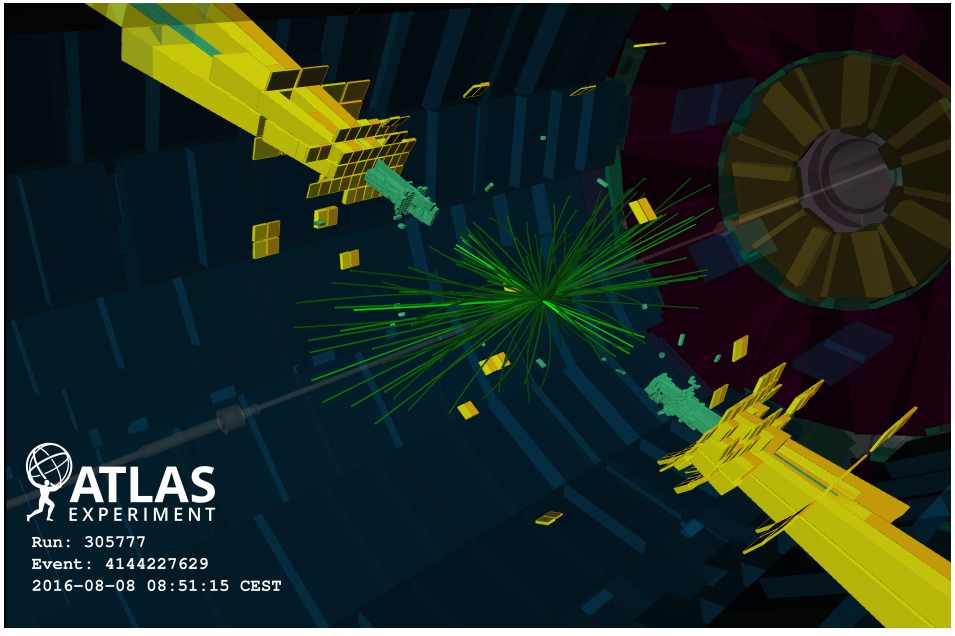
[Kingma et al. 2018]

ML event generation



Conclusion

- Machine learning is a rapidly advancing field of study.
- Methods are potentially applicable in particle physics tasks.
- One such method are generative models.
- We have looked at an exact log-likelihood generative approach to event generation called a normalizing flow.
- Increase in the rate of collision at the LHC will result in much higher data amounts and even more complex events to analyze.



- Machine learning will most certainly play an important role in searching for new physics beyond the Standard Model in the future.

References

ATLAS Collaboration (June 2015). "Search for vector-like B quarks in events with one isolated lepton, missing transverse momentum, and jets at $\sqrt{s} = 8$ TeV with the ATLAS detector". In: *Phys. Rev. D* 91 (11), p. 112011. DOI: 10.1103/PhysRevD.91.112011.

Calafiura, P et al. (Sept. 2020). *ATLAS HL-LHC Computing Conceptual Design Report*. Tech. rep. Geneva: CERN. URL: https://cds.cern.ch/record/2729668.

Baldi, P., P. Sadowski, and D. Whiteson (July 2014). "Searching for exotic particles in high-energy physics with deep learning". In: *Nature Communications* 5.1. ISSN: 2041-1723. URL: http://dx.doi.org/10.1038/ncomms5308.

Papamakarios, George et al. (2021). "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57, pp. 1–64.

Weng, Lilian (2018). "Flow-based Deep Generative Models". In: Accessed February 28 2022. URL: https://lilianweng.github.io/posts/2018-10-13-flow-models/.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2016). *Density estimation using Real NVP*. arXiv: 1605.08803 [cs.LG].

Kingma, Durk P and Prafulla Dhariwal (2018). "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* 31.

Thank you!

https://xkcd.com/1838/

Backup slides
and some other stuff...

Experimental cross section and luminosity

Fixed target experiment

- Flux:

$$\Phi_a = \frac{\dot{N}_a}{A} = n_a v_a$$

n_a . . . density of particle beam

v_a . . . velocity of beam particles

A . . . beam area

- Reaction rate:

$\dot{N} = \Phi_a \cdot N_b \cdot \sigma_b$

N_b . . . target particles within beam area

σ_a . . . effective area of single scattering center

- Luminosity: $L = \Phi_a \cdot N_b$

For the LHC:

$N_{a,b} \sim 10^{11}$, $A \sim 0.0005$ mm^2 ,

$U = 27$ km , $n \sim 2800$, $f \sim 11$ kHz ,

$L \sim 10^{34}$ cm^{-2}s^{-1} , $L_{\text{int}} = 189.3$ fb^{-1}

Collider experiment:

- Flux:

$$\Phi_a = \frac{\dot{N}_a}{A} = \frac{N_a \cdot n \cdot \frac{v}{U}}{A} = \frac{N_a \cdot n \cdot f}{A}$$

N_a . . . # particles per bunch (beam A)

N_b . . . # of particles per bunch (beam B)

U . . . circumeference of the ring

n . . . number of bunches per beam

v . . . velocity of beam particles

f . . . revolution frequency

- Luminosity:

$$L = f\frac{nN_aN_b}{A} = f\frac{nN_aN_b}{4\pi\sigma_x\sigma_y}$$

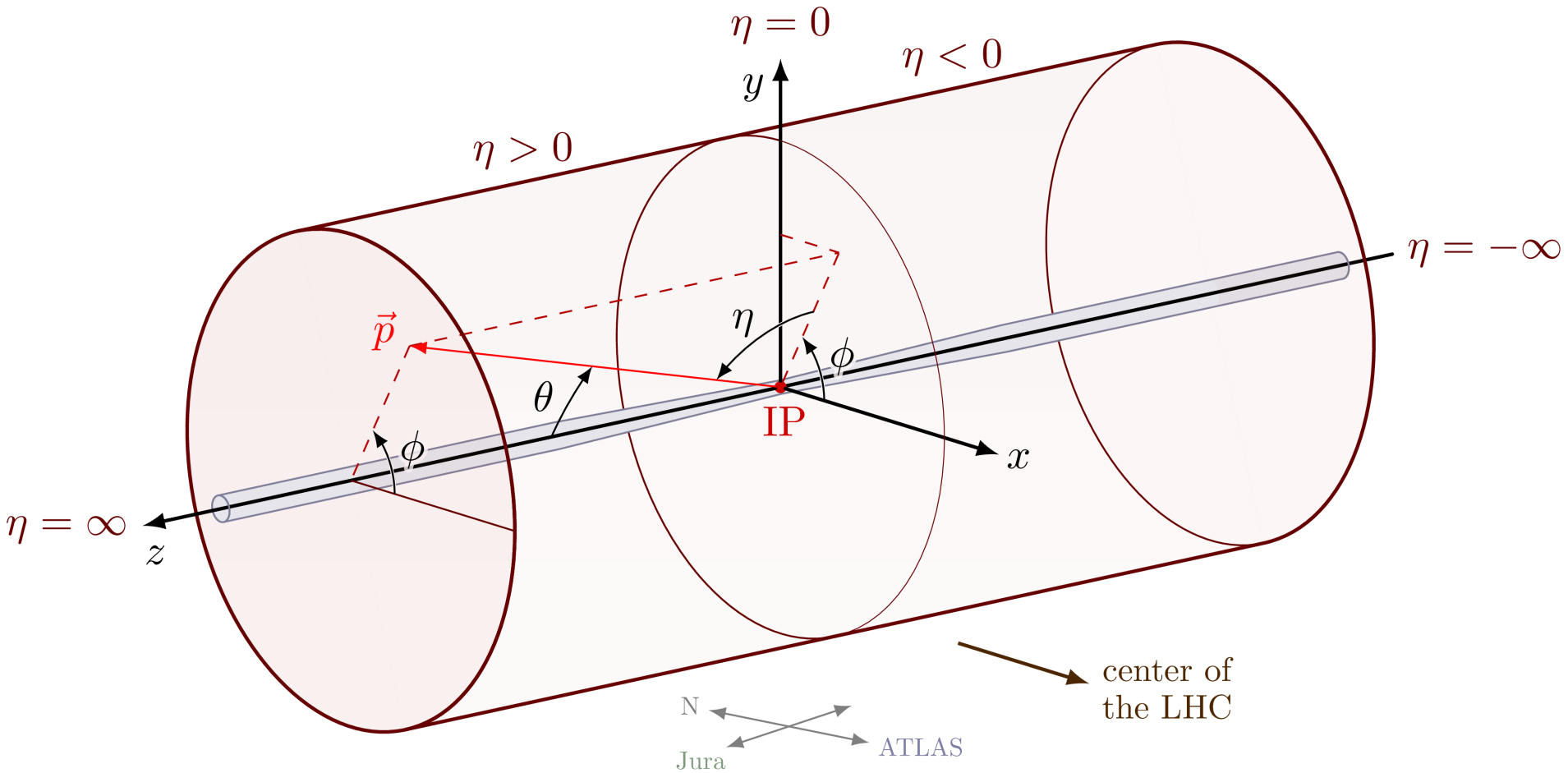
σ_x . . . std of beam profile in x

σ_y . . . std of beam profile in y

- Integrated luminosity:

$$\dot{N} \equiv L \cdot \sigma \ , \quad N = \sigma \cdot \int L dt \ , \quad \sigma = \frac{N}{L_{\text{int}}}$$

CMS and ATLAS coordinate system



Maximum likelihood estimation (MLE)

- Pick the parameters that assign the highest probability to the training data $\mathcal{D}$.
- MLE definition:

$$\hat{\boldsymbol{\theta}} \equiv \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}|\boldsymbol{\theta}) \,.$$

- Assume independent and identically distributed (**iid**) samples

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}) \,,$$

  where $p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta})$ is a (conditional) likelihood.
- We usually work with the **log likelihood** defined as

$$\mathcal{L}(\boldsymbol{\theta}) \equiv \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{y}_n|\mathbf{x}_n, \boldsymbol{\theta}) \,.$$

- At the heart of all model fitting/machine learning (in one way or another):

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) \,.$$

- $\mathcal{L}(\boldsymbol{\theta})$ is generally some kind of loss function or objective function.

- MLE is thus given by

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) \, .$$

- Optimization algorithms designed for cost function minimization $\Rightarrow$ redefine to **negative log-likelihood**:

$$\mathcal{L}(\boldsymbol{\theta}) = \text{NLL}(\boldsymbol{\theta}) = -\log p(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(\mathbf{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) \, .$$

- For conditional (unsupervised) MLE we finally have

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[-\sum_{n=1}^{N} \log p(\mathbf{y}_n | \boldsymbol{\theta}) \right]$$

or written as an expected value (assuming the law of large numbers)

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[-\mathbb{E}_{\mathbf{y} \sim p_{\mathcal{D}}} \log p(\mathbf{y}|\boldsymbol{\theta}) \right] \, ,$$

where $p_{\mathcal{D}}$ is an empirical data distribution.

Kullback-Leibler (KL) divergence and MLE

- KL divergence for continuous distributions:

$$D_{\mathrm{KL}}(p\|q) \equiv \int p(x) \log \frac{p(x)}{q(x)} dx \, .$$

- KL divergence for discrete distributions:

$$D_{\mathrm{KL}}(p\|q) \equiv \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}$$

  and can be rewritten as

$$D_{\mathrm{KL}}(p\|q) = \sum_{k=1}^{K} p_k \log p_k - \sum_{k=1}^{K} p_k \log q_k = -H(p) + H(p,q) \, ,$$

  where $H(p)$ is the entropy of p and $H(p,q)$ is the cross-entropy of p and q.
- We want to find q that is as close as possible to p, measured by KL:

$$q^* = \underset{q}{\operatorname{argmin}} D_{\mathrm{KL}}(p\|q) = \underset{q}{\operatorname{argmin}} \left[\int p(x) \log p(x) dx - \int p(x) \log q(x) dx \right] \, .$$

- Let p be the empirical distribution defined as

$$p_{\mathcal{D}}(x) = \frac{1}{N} \sum_{n=1}^{N} \delta(x - x_n) \, .$$

- We get

$$\begin{aligned} D_{\mathrm{KL}}(p_{\mathcal{D}} \| q) &= -\int p_{\mathcal{D}}(x) \log q(x) dx + C \\ &= -\int \left[\frac{1}{N} \sum_{n=1}^{N} \delta(x - x_n) \right] \log q(x) dx + C \\ &= -\frac{1}{N} \sum_{n=1}^{N} \log q(x_n) + C = \mathrm{NLL}(q) + C \, , \end{aligned}$$

  where C is a constant independent of q (entropy H).

- Thus minimizing the KL is equivalent to minimizing the NLL which is equivalent to computing the MLE.
- This is the loss function used throughout all of flow training.

RealNVP coupling layer construction

- $c_i(\mathbf{z}_{<i})$ can be any model with input $\mathbf{z}_{<i}$ and output $\mathbf{h}$.
- A coupling layer splits $\mathbf{z}$ into two parts at index d and transforms the second part as a function of the first part:

$$\begin{aligned}\mathbf{z}'_{\leq d} &= \mathbf{z}_{\leq d} \\ \boldsymbol{h}_{>d} &= F(\mathbf{z}_{\leq d}) \\ \mathbf{z}'_{>d} &= \tau(\mathbf{z}_{>d}; \boldsymbol{h}_{>d})\end{aligned}$$

- Use *real-valued non-volume preserving (realNVP)* model:

$$\begin{aligned}\mathbf{z}'_{\leq d} &= \mathbf{z}_{\leq d} \\ \boldsymbol{h}_{>d} &= \{s(\mathbf{z}_{\leq d}),\ t(\mathbf{z}_{\leq d})\} \\ \mathbf{z}'_{>d} &= \mathbf{z}_{>d} \odot \exp\left(s(\mathbf{z}_{\leq d})\right) + t(\mathbf{z}_{\leq d})\end{aligned}$$

- The inverse

$$\begin{aligned}\mathbf{z}_{\leq d} &= \mathbf{z}'_{\leq d} \\ \mathbf{z}_{>d} &= \left(\mathbf{z}'_{>d} - t(\mathbf{z}'_{\leq d})\right) \odot \exp\left(-s(\mathbf{z}'_{\leq d})\right)\end{aligned}$$

  does not require computing inverses of s and t $\Rightarrow$ arbitrarily complex!
- The Jacobian is lower triangular with block like structure.

RealNVP Jacobian and masking

- Jacobian is lower triangular with block like structure

$$J_{f_\phi} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{A} & \mathbf{D} \end{bmatrix} .$$

- Only relevant part is

$$\mathbf{D} = \text{diag}\left[\exp(s(\mathbf{z}_{\leq d}))\right] .$$

- Determinant easy to calculate:

$$\log \left|\det J_{f_\phi}(\mathbf{z})\right| = \sum_j s(\mathbf{z}_{\leq d})_j .$$

- Splitting and joining can be implemented using binary masks $\mathbf{b}$:

$$\mathbf{z}' = \mathbf{b} \odot \mathbf{z} + (1 - \mathbf{b}) \odot (\mathbf{z} \odot \exp(s(\mathbf{b} \odot \mathbf{z})) + t(\mathbf{b} \odot \mathbf{z})) ,$$
$$\mathbf{z} = \mathbf{b} \odot \mathbf{z}' + (1 - \mathbf{b}) \odot (\mathbf{z}' - t(\mathbf{b} \odot \mathbf{z}')) \odot \exp(-s(\mathbf{b} \odot \mathbf{z}')) ,$$

  where partitioning is handled implicitly by binary mask construction.
- An example of $\mathbf{b}$ would be a checkerboard mask

$$b_d = d \bmod 2 \quad \text{where} \quad d = 0, \ldots, D - 1 .$$

Invertible 1×1 convolution

- Cross correlation $\Leftrightarrow$ convolution:

$$[\mathbf{W} \star \mathbf{X}](k, i, j) = \sum_{u=1}^{H} \sum_{v=1}^{W} \sum_{c=1}^{C_{\text{in}}} w_{k,u,v,c}\, x_{i+u,j+v,c} \,.$$

- For images $\mathbf{X}$ with height H, width W and number of input channels C_{in}.
- A 1×1 convolution, i.e. $H = W = 1$, is given by

$$[\mathbf{W} \star \mathbf{X}](k, i, j) = \sum_{c=1}^{C_{\text{in}}} w_{k,c,1,1}\, x_{i,j,c}$$

  and changes the number of channels from C_{in} to C_{out}.
- This can be written as matrix multiplication

$$[\mathbf{W} \star \mathbf{X}](k, l) = \sum_{c=1}^{C_{\text{in}}} w_{k,c} x_{c,l} = \mathbf{W}\mathbf{X} \,.$$

- Replacing channels with features (images $\mathbf{X} \rightarrow$ tabular data $\mathbf{x}$) gives us

$$[\mathbf{W} \star \mathbf{x}](k) = \sum_{d=1}^{D_{\text{in}}} w_{k,d} x_d = \mathbf{W}\mathbf{x} \,.$$

Batch normalization

- Use batch normalization to further speed up and stabilize flow training.
- Batch norm and its inverse consist of two affine transformations:

$$\mathrm{BN}(\mathbf{z}) = \boldsymbol{\alpha} \odot \frac{\mathbf{z} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \boldsymbol{\epsilon}}} + \boldsymbol{\beta} \,,$$

$$\mathrm{BN}^{-1}(\mathbf{z}') = \boldsymbol{\mu} + \frac{\mathbf{z}' - \boldsymbol{\beta}}{\boldsymbol{\alpha}} \odot \sqrt{\boldsymbol{\sigma}^2 + \boldsymbol{\epsilon}} \,.$$

- Updates of batch parameters are done as (b is the index of the current batch)

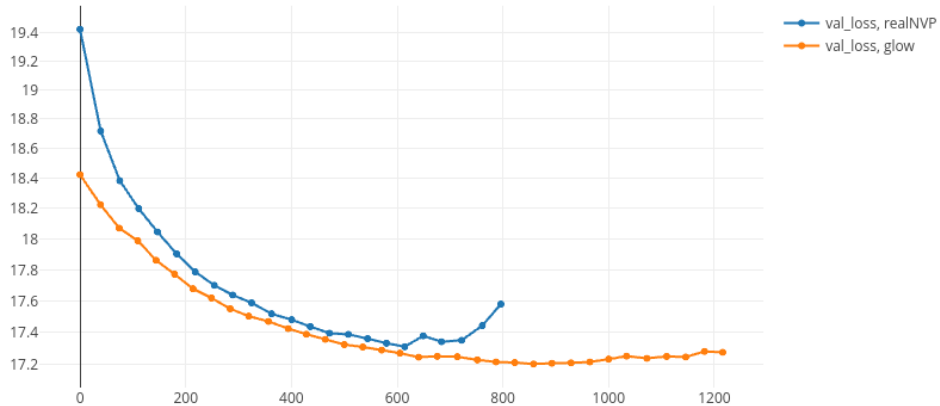
$$\boldsymbol{\mu}_{b+1} = \rho \boldsymbol{\mu}_{b-1} + (1 - \rho)\boldsymbol{\mu}_b \,,$$
$$\boldsymbol{\sigma}^2_{b+1} = \rho \boldsymbol{\sigma}^2_{b-1} + (1 - \rho)\boldsymbol{\sigma}^2_b \,,$$

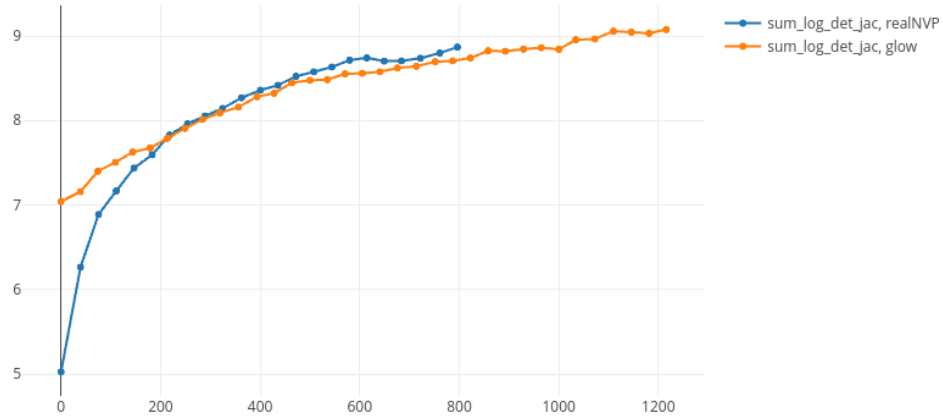
  where ρ is a hyperparameter known as momentum.
- Batch norm is an elementwise operation $\Rightarrow$ easy to compute determinant:

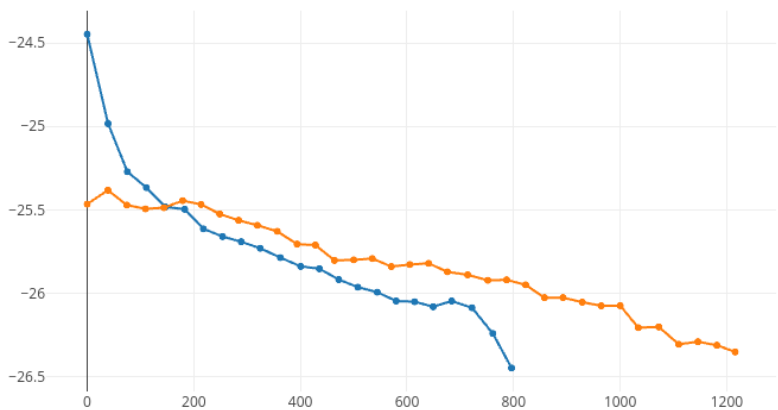
$$\log|\det J_{\mathrm{BN}}(\mathbf{z})| = \sum_{i=1}^{D} \log \frac{\alpha_i}{\sqrt{\sigma_i^2 + \epsilon_i}} \,.$$

- Used as another layer after/before permutation: $\ldots \circ T_k \circ \mathrm{BN} \circ T_{k-1} \circ \ldots$

RealNVP and Glow training curves







Piecewise-quadratic coupling transform

- Piecewise-polynomial invertible maps as coupling transforms.
- Choose to operate in a unit hypercube: $[0, 1]^D$.
- Partitioning (splitting) of input into two parts $\mathbf{z} = (\mathbf{z}^A, \mathbf{z}^B)$ at d to ensure computationally tractable Jacobians via separability.
- Each dimension transforms independently:

$$\mathbf{z}'^B = \tau\left[\mathbf{z}^B; \boldsymbol{h}(\mathbf{z}^A)\right] = \left(\tau_1[\mathbf{z}_1^B; \boldsymbol{h}(\mathbf{z}^A)], \ldots, \tau_d[\mathbf{z}_d^B; \boldsymbol{h}(\mathbf{z}^A)]\right) \ .$$

- Interpret the transformation function τ as a CDF and denoting it as C.
- This can be done due to operation on unit intervals.
- The neural network outputs the unnormalized PDF $q \Rightarrow$ get C by integration.
- Piecewise-quadratic coupling transforms $\Rightarrow$ piecewise-linear PDF.
- Model this PDF by $K + 1$ vertices and K bin widths.
- NN outputs a vector of shape $d(2K + 1)$ that we intepret as a $d \times (K + 1)$ matrix V of vertices and $d \times K$ matrix of bin widths W.

- The PDF q_i of i-th dimension for a data point z_i is defined as:

$$q_i(z) = \begin{cases} \frac{1}{W_{i1}}\left(V_{i2} - V_{i1}\right) z + V_{i1}\ ; & z < W_{i1} \\ \frac{1}{W_{i2}}\left(V_{i3} - W_{i2}\right)\left(z - W_{i1}\right) + V_{i2}\ ; & W_{i1} \le z < W_{i1} + W_{i2} \\ \vdots & \vdots \\ \underbrace{\frac{1}{W_{iK}}\left(V_{i(K+1)} - V_{iK}\right)}_{\text{slope}} \underbrace{\left(z - \sum_{k=1}^{K-1} W_{ik}\right)}_{\text{shift to origin}} + \underbrace{V_{iK}}_{\text{offset}}\ ; & \sum_{k=1}^{K-1} W_{ik} \le z < 1 \end{cases}$$

and is a type of linear interpolation function given by

$$\text{lerp}(a, b, t) = a + (b - a)t\ .$$

- Integrating we get the CDF (trivial once drawn on a piece of paper)

$$C_i(z_i^B, W, V) = \int_0^{z_i^B} q_i(z)dz = \frac{(\alpha_i^B)^2}{2}(V_{ib+1} - V_{ib})W_{ib} + \alpha_i^B V_{ib} W_{ib} + \sum_{k=1}^{b-1} \frac{1}{2}(V_{ik+1} + V_{ik})W_{ik}\ , \quad \alpha_i^B = \frac{1}{W_{ib}}(z_i^B - \sum_{k=1}^{b-1} W_{ik})\ ,$$

where $b \in [1, K]$ is the index of z_i^B data point bin and α_i^B is the relative position of z_i^B in b.

- Note that W is normalized by the softmax function and V by the area $\sum_{k=1}^{b-1} \frac{1}{2}(V_{ik+1} + V_{ik})W_{ik}$ where by V we mean $V = \text{softplus}(V) + \epsilon$.
- Solving the quadratic CDF equation for α_i^B we get the inverse

$$\alpha_i^B = \frac{-V_{ib}}{V_{ib+1} - V_{ib}} + \sqrt{\frac{V_{ib}^2}{(V_{ib+1} - V_{ib})^2} - 2\beta_i^B}$$

where

$$\beta_i^B = -\frac{1}{W_{ib}(V_{ib+1} - V_{ib})} \left[C_i(\mathsf{z}_i^B, W, V) - \sum_{k=1}^{b-1} \frac{1}{2}(V_{ik+1} + V_{ik})W_{ik} \right] .$$

- z_i^B can thus be calculated from

$$\mathsf{z}_i^B = \alpha_i^B W_{ib} + \sum_{k=1}^{b-1} W_{ik} .$$

- The log-determinant (for the forward direction) is given by

$$\log \left| \det \left(\frac{\partial C_i(\mathsf{z}_i^B, W, V)}{\partial \mathsf{z}_i^B} \right) \right| = \log \left[\alpha_i^B (V_{ib+1} - V_{ib}) W_{ib} + V_{ib} W_{ib} \right] .$$

Shift & scale vs piecewise-quadratic