Moving beyond dense networks

IDPASC school 2025

Anja Butter, LPNHE/ITP

LPNHE PARIS

How to train better networks

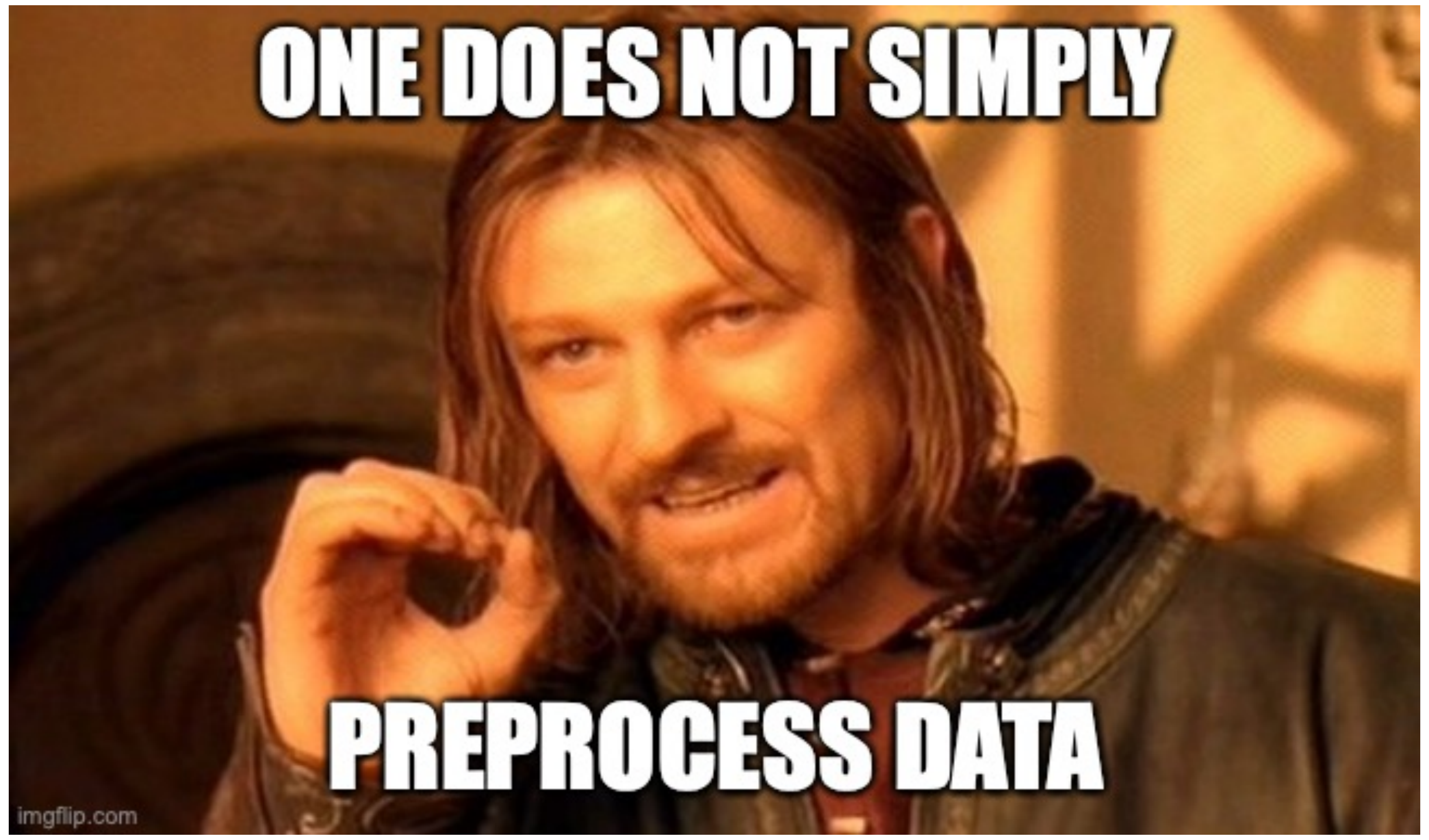
Some pointers

1. Preprocessing
2. Network initialisation
3. Optimisation of the training
4. Hyperparameter scans

Data Preprocessing

Why preporcessing?

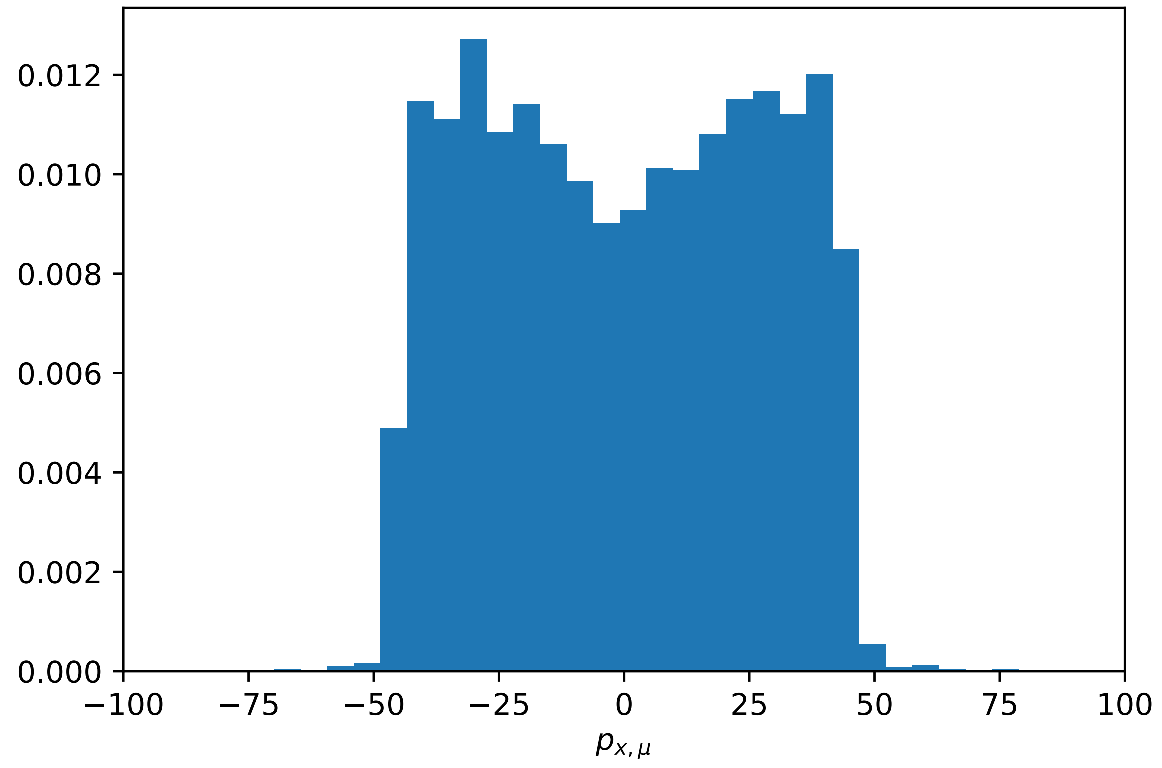
- input features with different scales
  eg. jet = (charge, $n_{particles}, p_T, M, \eta, \phi$)
- large value with small spread
  eg. $pp \to Z \to ll, m_{ll} \in [80 \text{ GeV} - 100 \text{ GeV}]$
- weights usually initialized to be sensitive in range [-1,+1]
- classification output in range [0,1]
- training more efficient/stable if features are also in range [-1, +1]



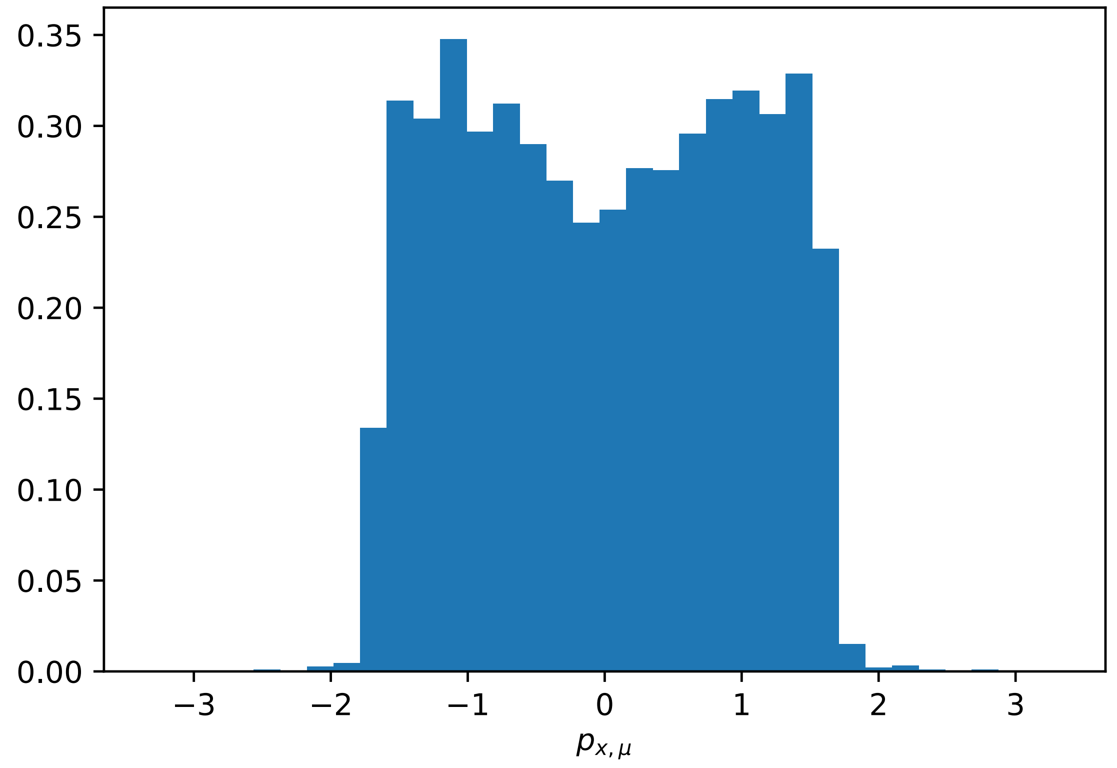
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$

Rule of thumb: rescale to $\mu = 0, \sigma = 1$



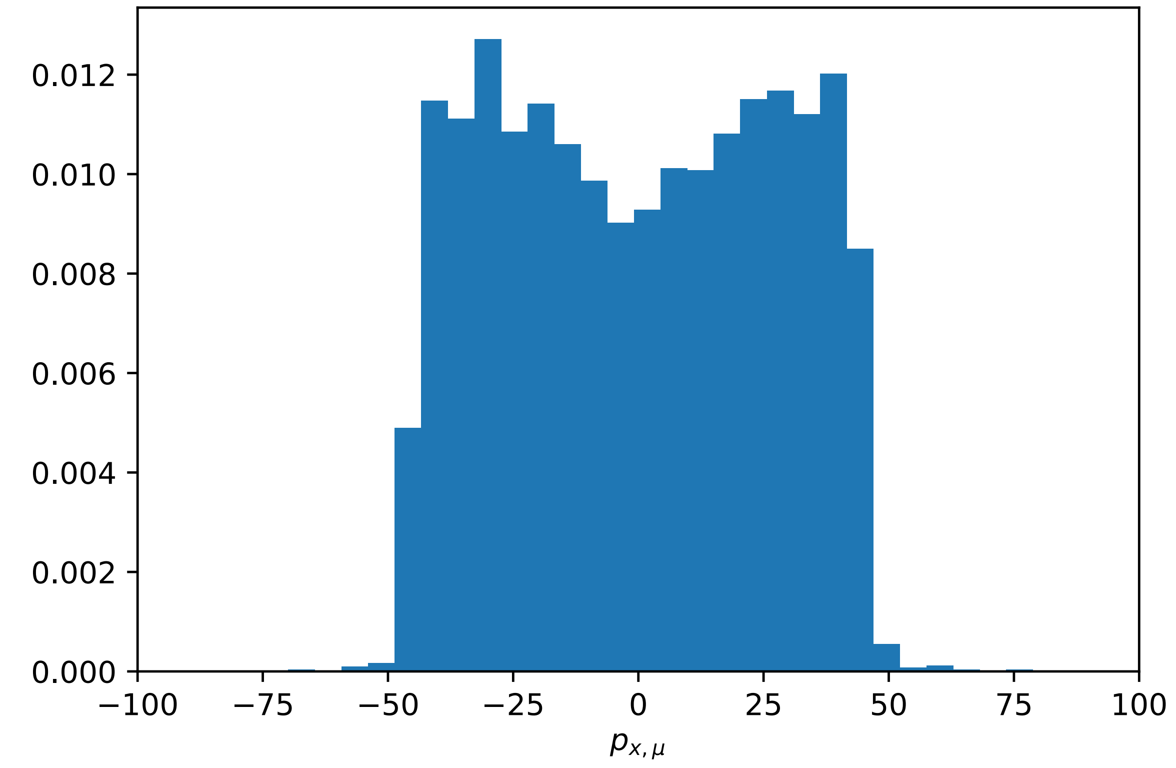
$$\frac{p_x - \bar{p}_x}{\sigma(p_x)}$$



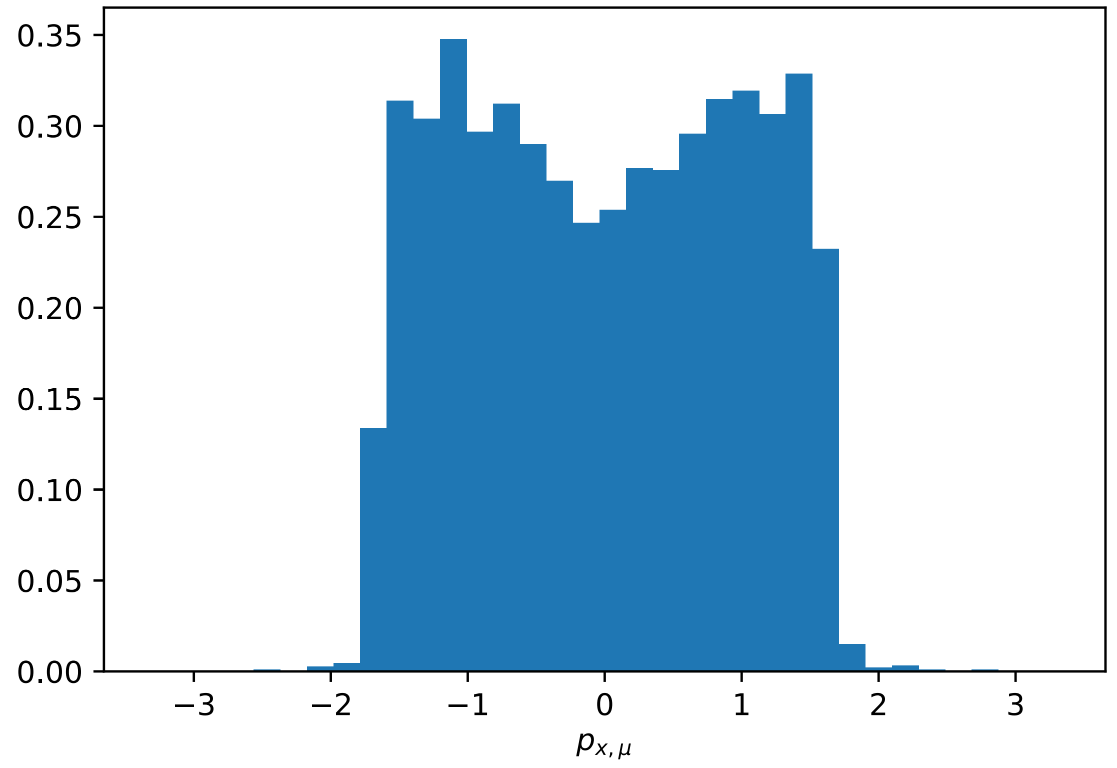
Rescaling

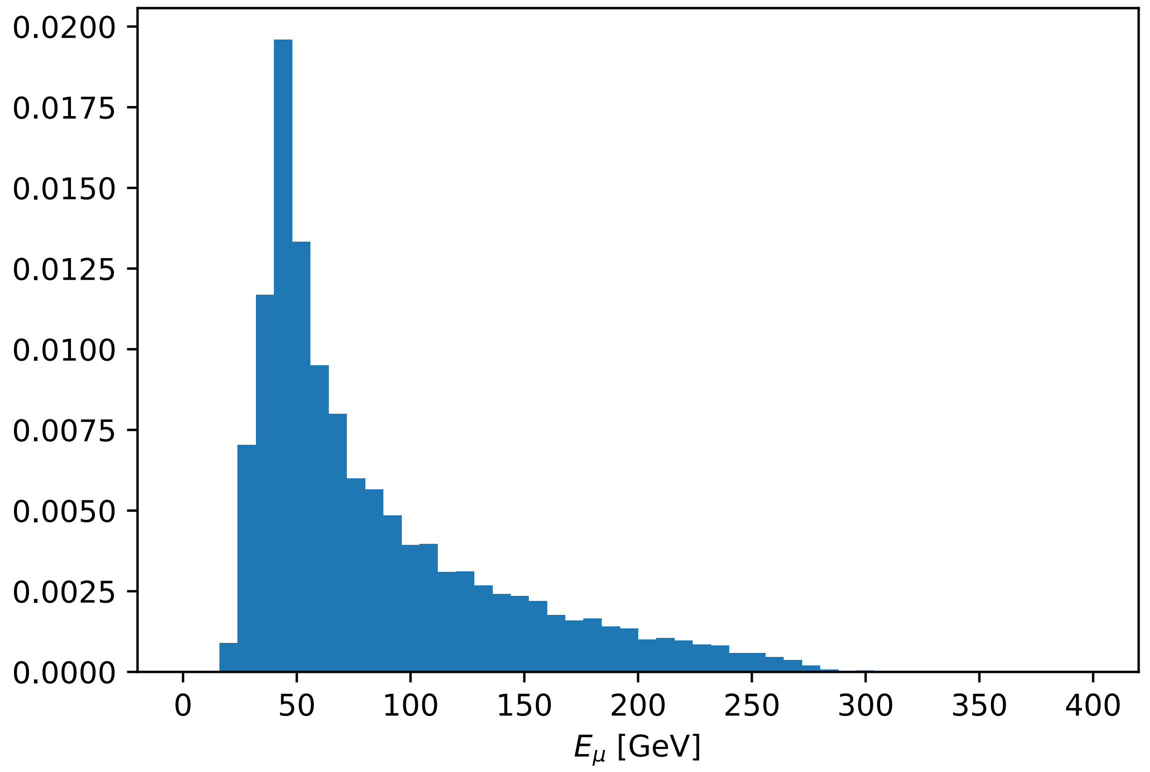
Example: $pp \to Z \to \mu^+\mu^-$

Rule of thumb: rescale to $\mu = 0, \sigma = 1$

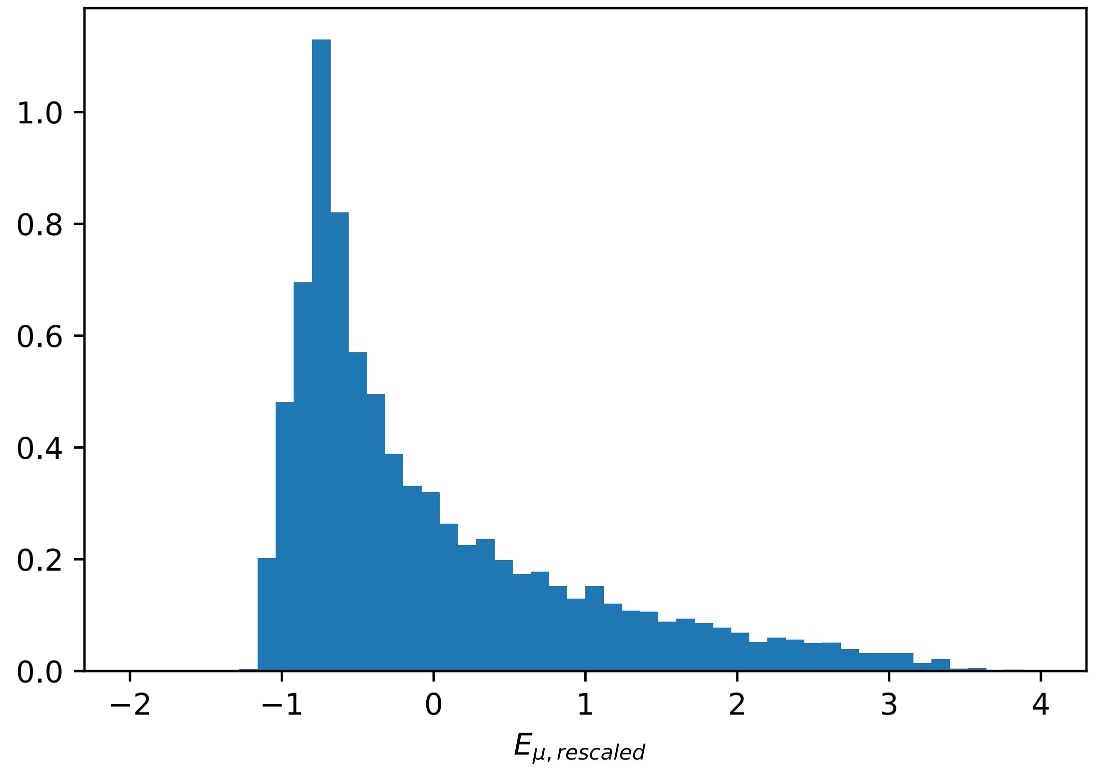


$$\xrightarrow{\frac{p_x - \bar{p}_x}{\sigma(p_x)}}$$





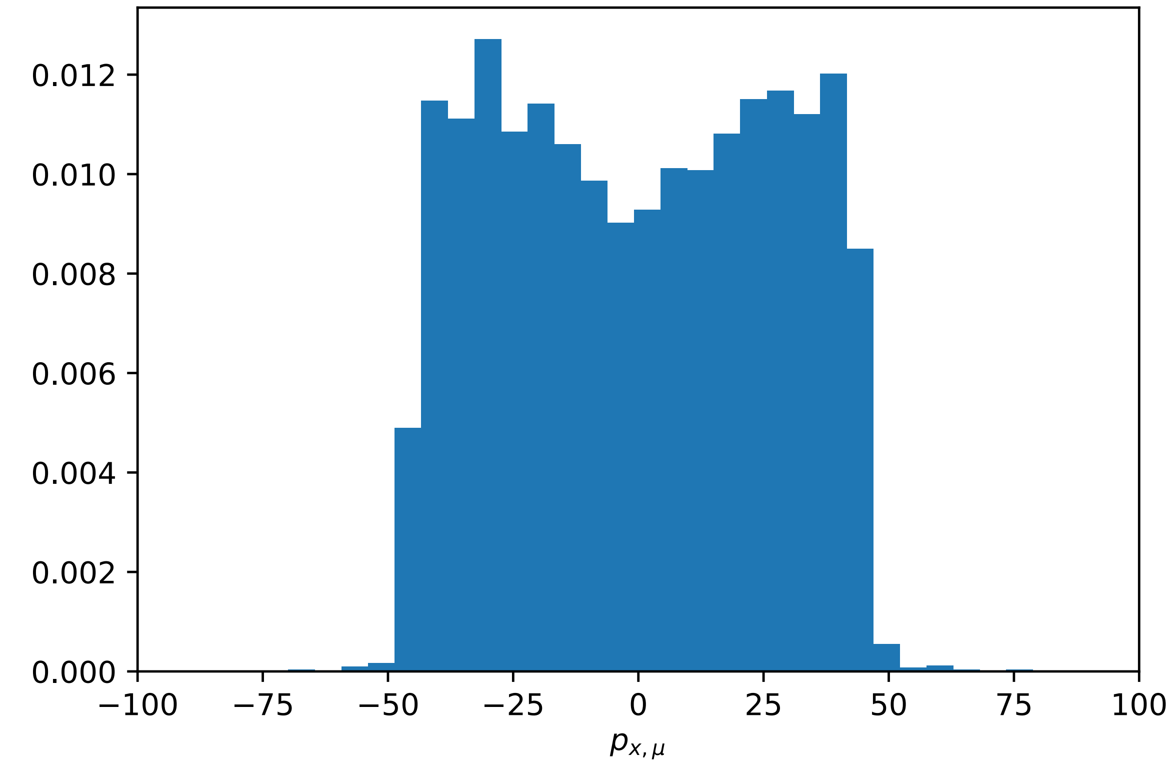
$$\xrightarrow{\frac{E - \bar{E}}{\sigma(E)}}$$



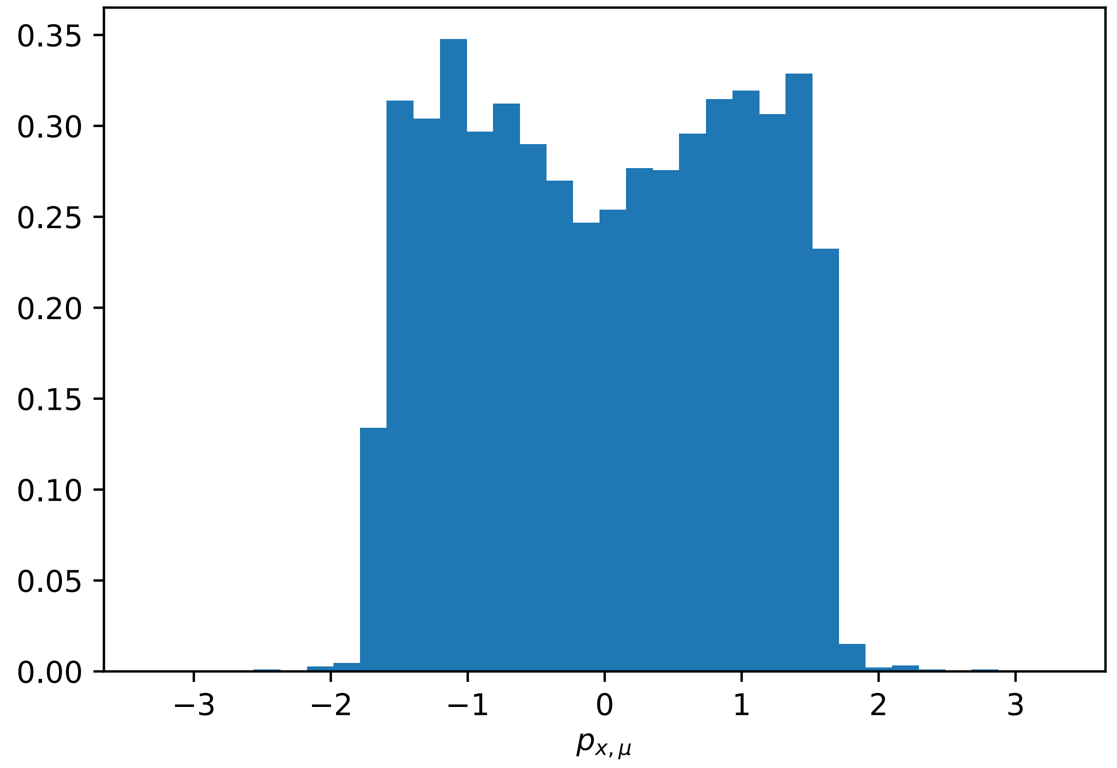
Rescaling

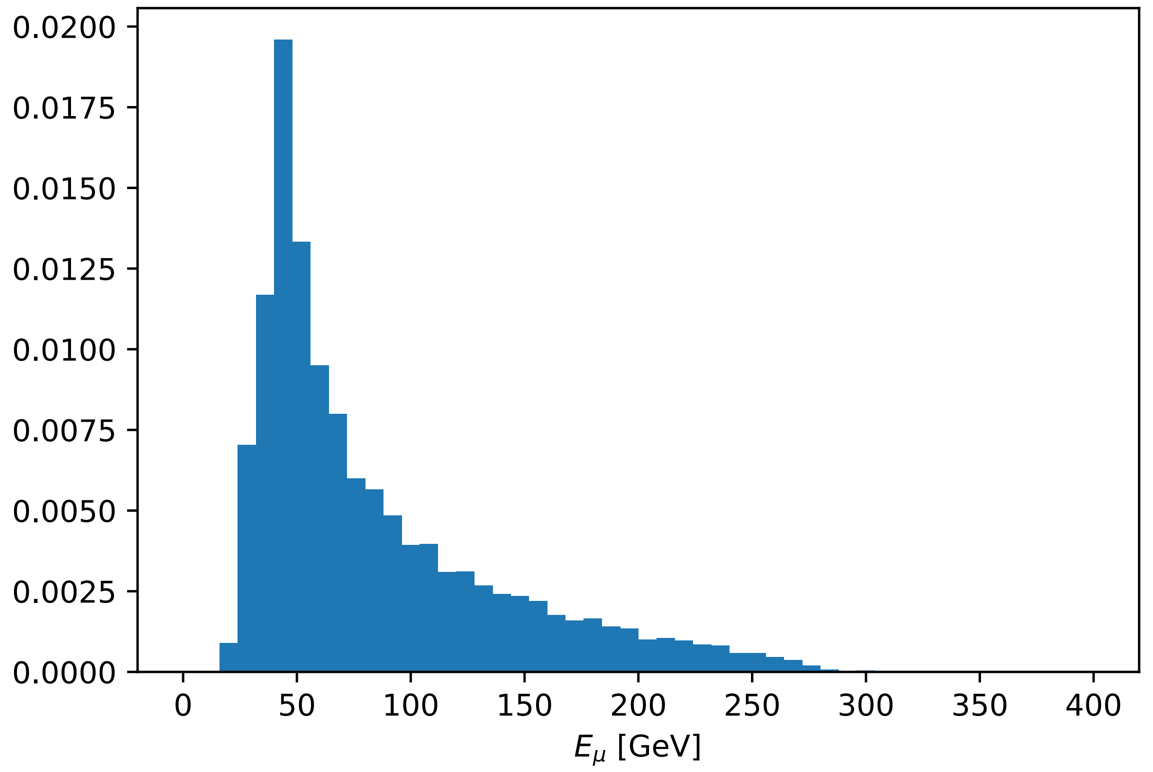
Example: $pp \rightarrow Z \rightarrow \mu^+\mu^-$

Rule of thumb: rescale to $\mu = 0, \sigma = 1$



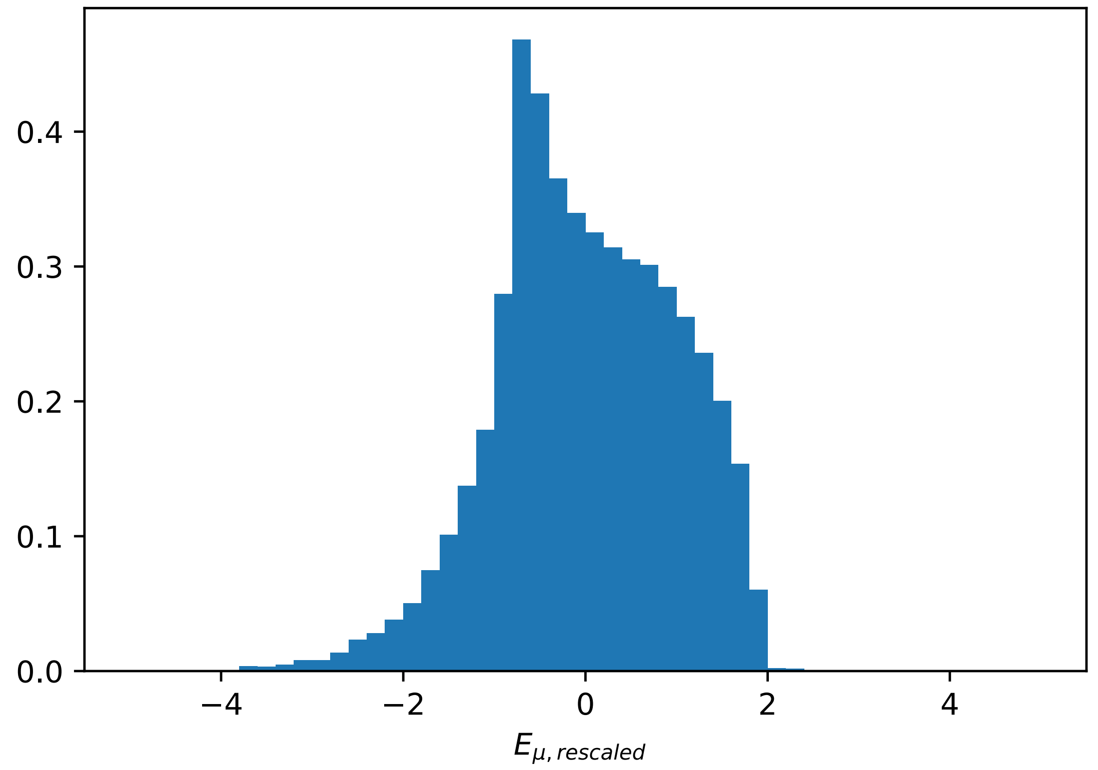
$$\xrightarrow{\frac{p_x - \bar{p}_x}{\sigma(p_x)}}$$





$$\xrightarrow{\frac{E' - \bar{E}'}{\sigma(E')}}$$

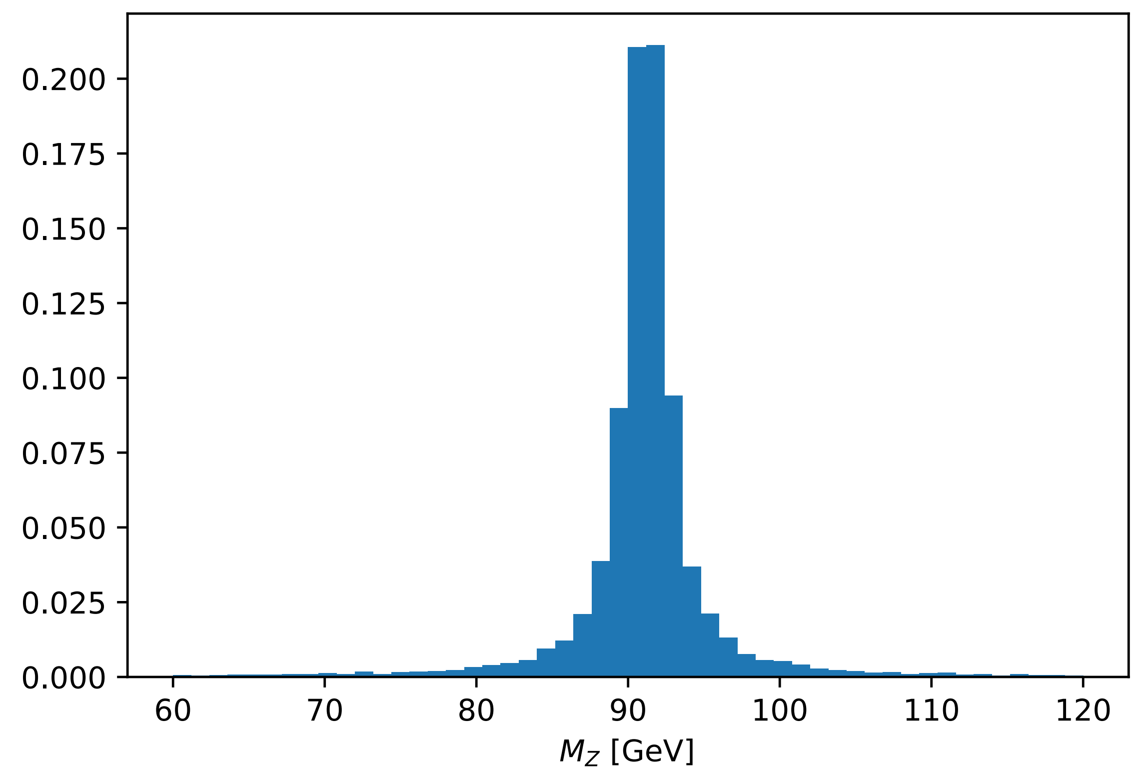
$E' = \log(E - 20)$



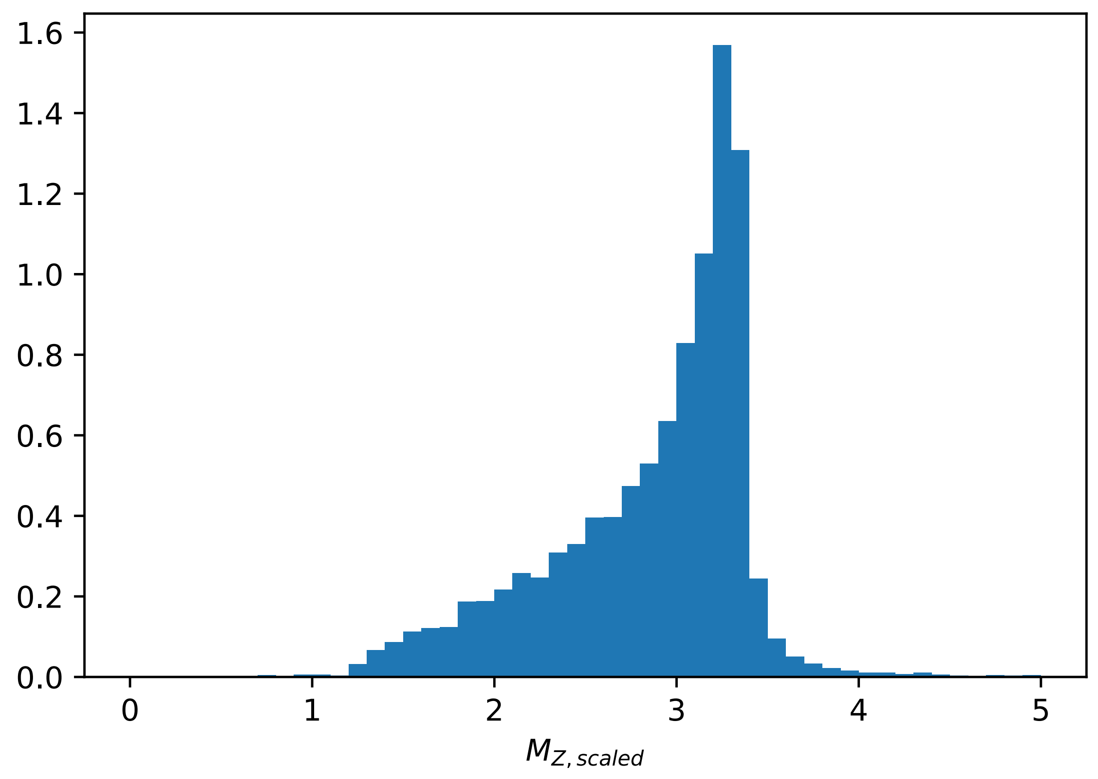
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$

Exception: Correlated observables

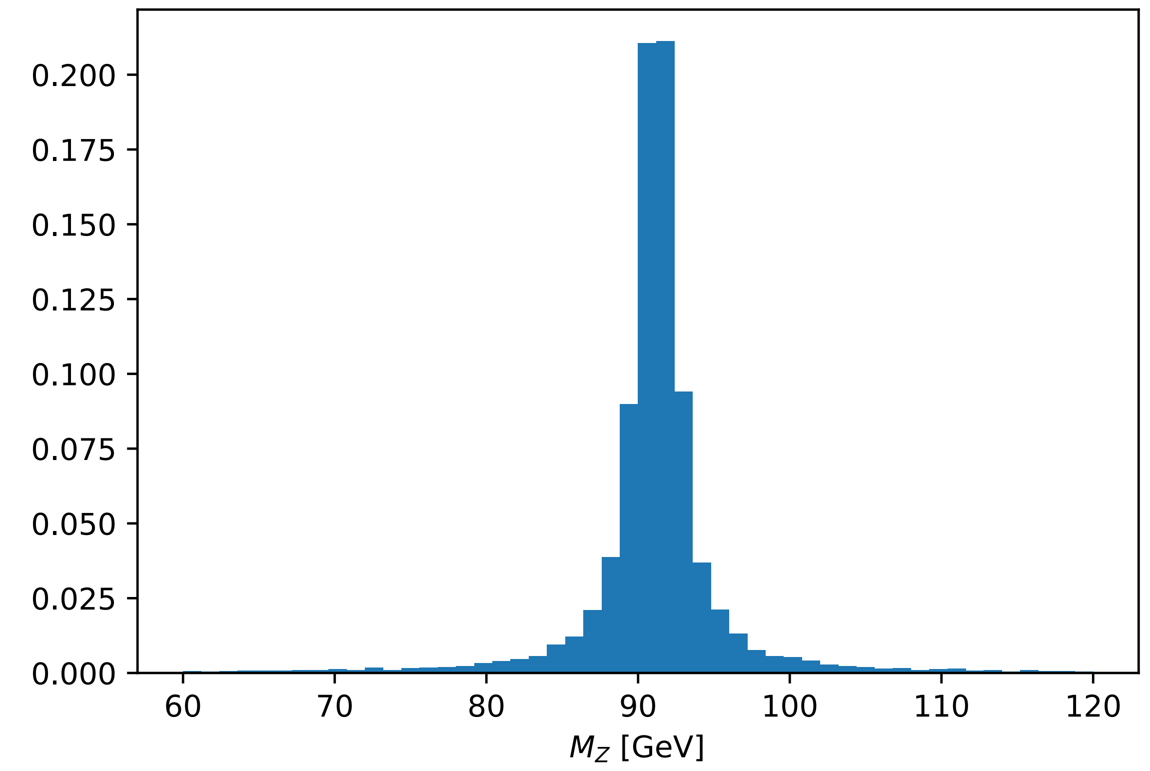


$$\frac{p_{i,\mu} - \bar{p}_{i,\mu}}{\sigma(p_{i,\mu})} \longrightarrow$$

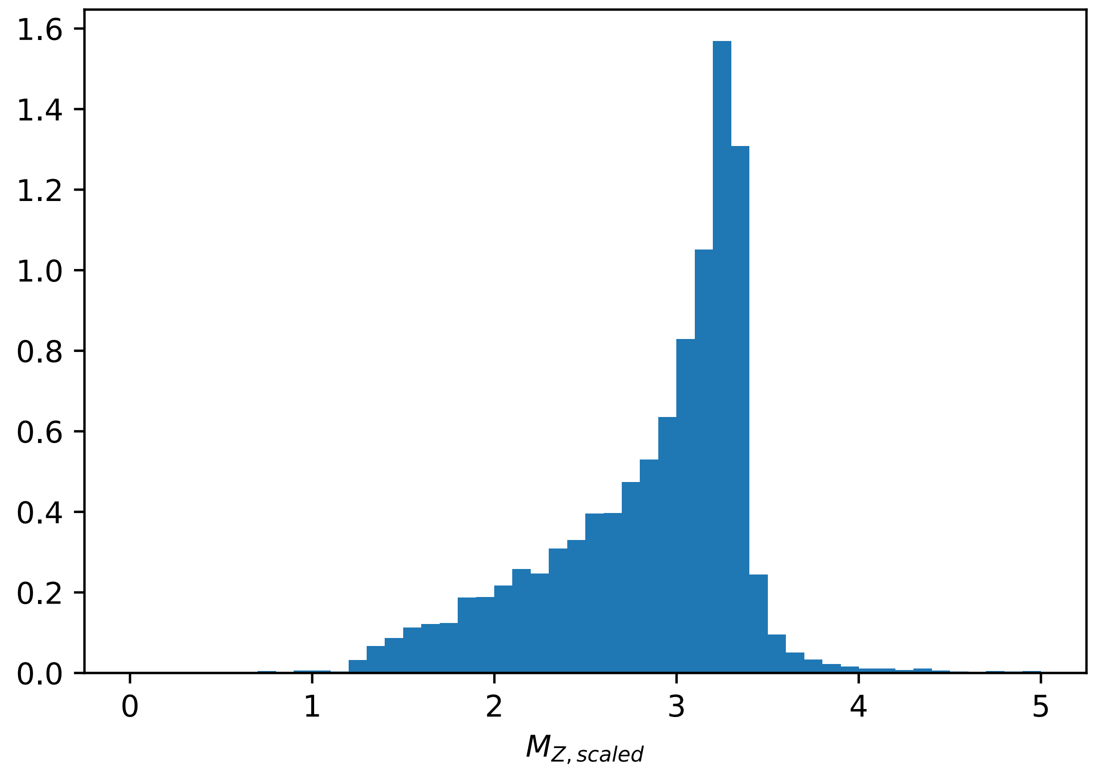


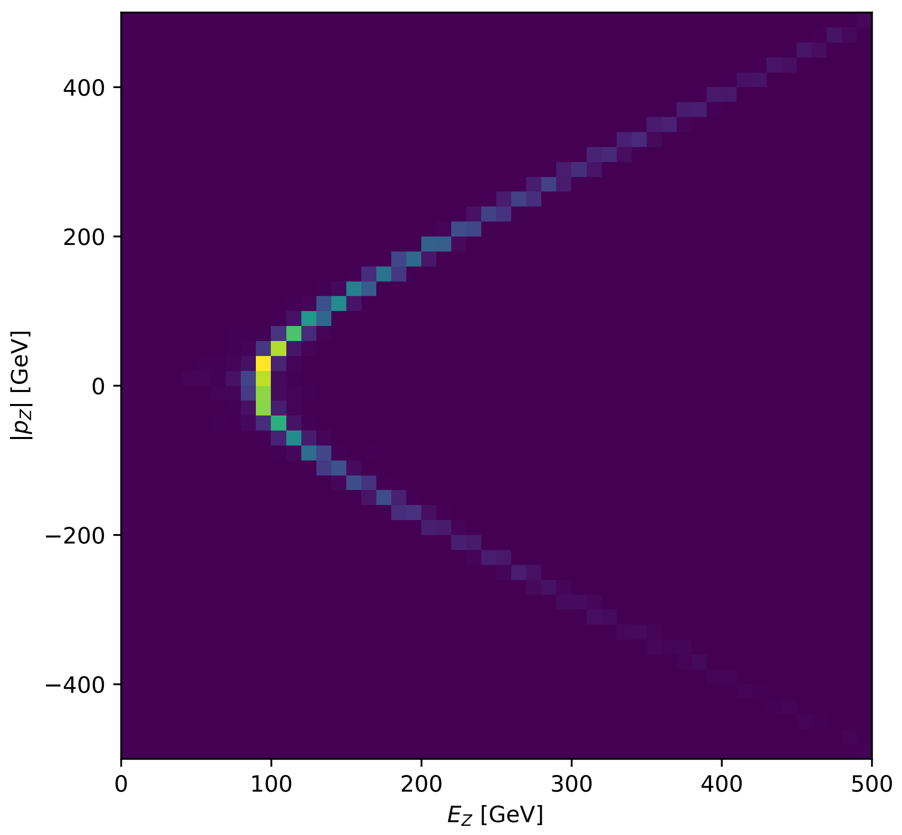
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Exception: Correlated observables

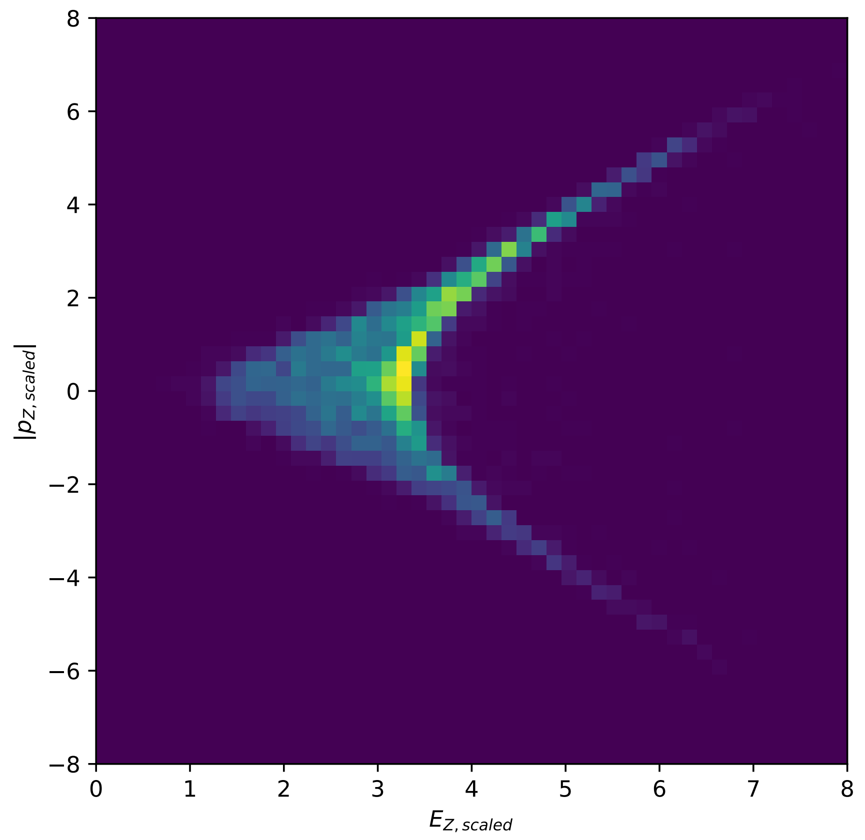


$$\xrightarrow{\frac{p_{i,\mu}-\bar{p}_{i,\mu}}{\sigma(p_{i,\mu})}}$$





$$\xrightarrow{\frac{p_{i,\mu}-\bar{p}_{i,\mu}}{\sigma(p_{i,\mu})}}$$

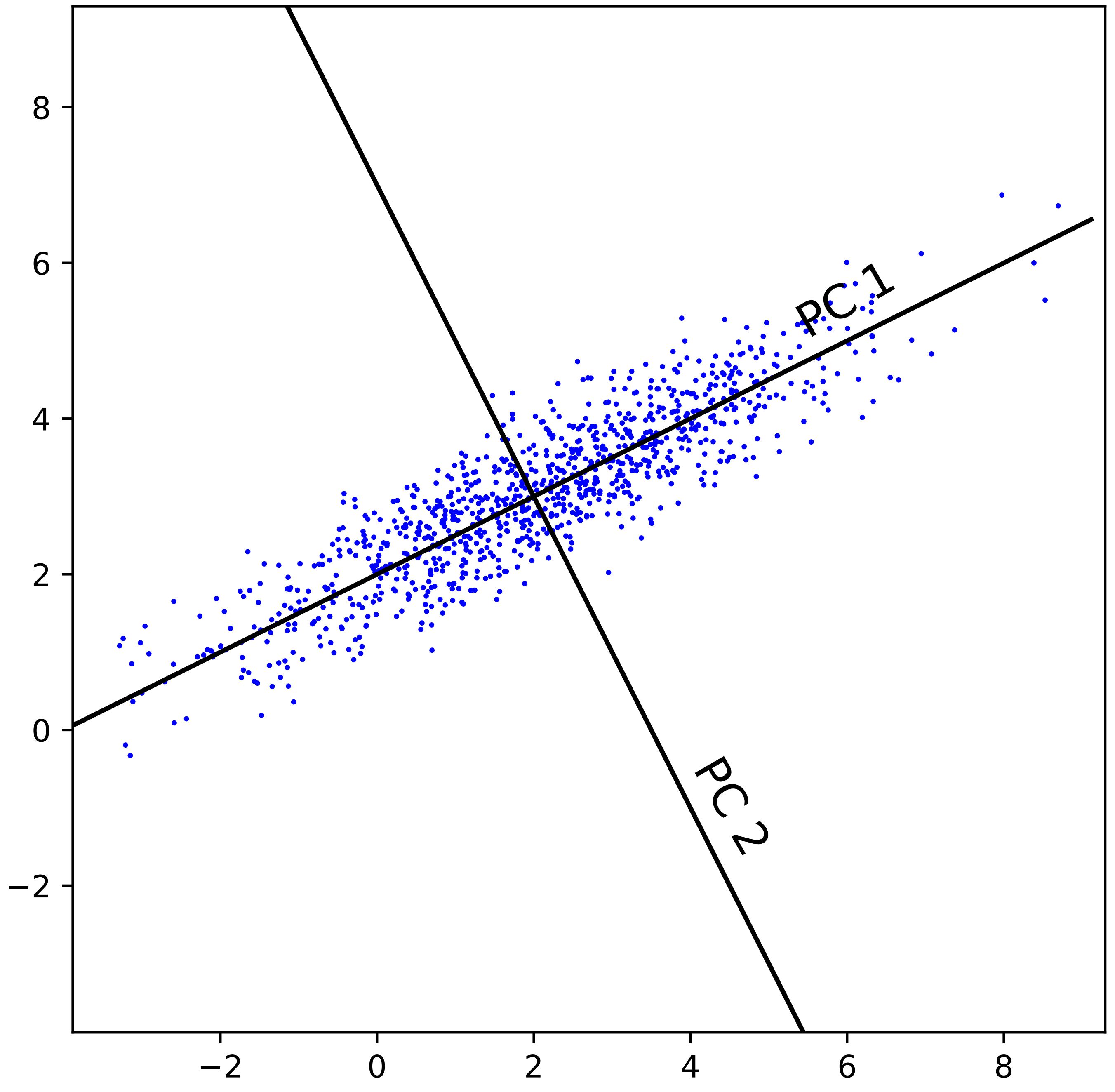


$\Rightarrow$ Use same scale for $p_{i,\mu}$

PCA

Principal component analysis

- directions maximizing variance
- eigenvector of covariance matrix
- $\text{cov}(\boldsymbol{X}) = \boldsymbol{X}^T \boldsymbol{X}$



\+ facilitates training
\+ useful for interpretation
\+ can reduce data dimension

2. Network initialization

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$?

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$?

   symmetric initialization $\Rightarrow$ symmetric updates $\Rightarrow$ identical weights ↯

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1?$ ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)?$

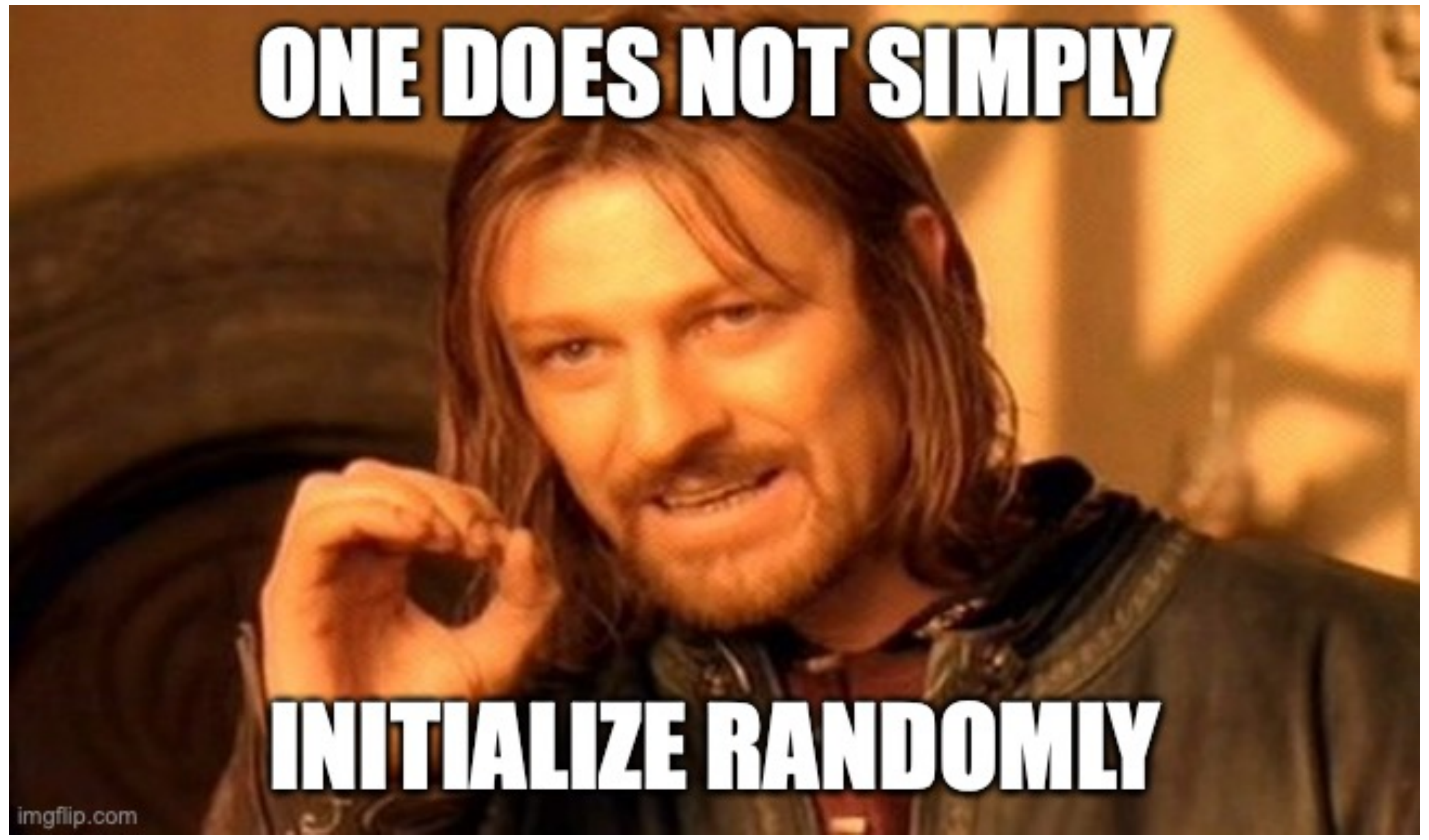
Check for single neuron $y = w_i x_i$ with w_i, x_i independent:

$$
\begin{aligned}
< y^2 > &= \sum_i < w_i^2 x_i^2 > \\
&= \sum_i < w_i >^2 < x_i^2 > + < x_i >^2 < w_i^2 > + < w_i^2 >< x_i^2 > \\
&= \sum_i < w_i^2 >< x_i^2 > \quad \leftarrow < w_i > = < x_i > = 0 \\
&= n_{incoming} < w_i^2 >< x_i^2 > \ \text{diverges!}
\end{aligned}
$$

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1?$ ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)?$ ↯



$$\rightarrow < w_i^2 > = \frac{1}{n_{incoming}}$$ to preserve variance through network

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$? ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$? ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
   - caveat 1: Same argument for backpropagation → average $(n_{in} + n_{out})/2$
   - caveat 2: only for ≈ linear activation function eg. tanh

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$? ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$? ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
4. ReLU $\rightarrow$ 50% of outputs $= 0 \rightarrow$ additional factor 2
   $\Rightarrow$ He initialization $\sigma = \sqrt{2/n_{in}}$

Network initialization

We know how to update weights. But how do we start?

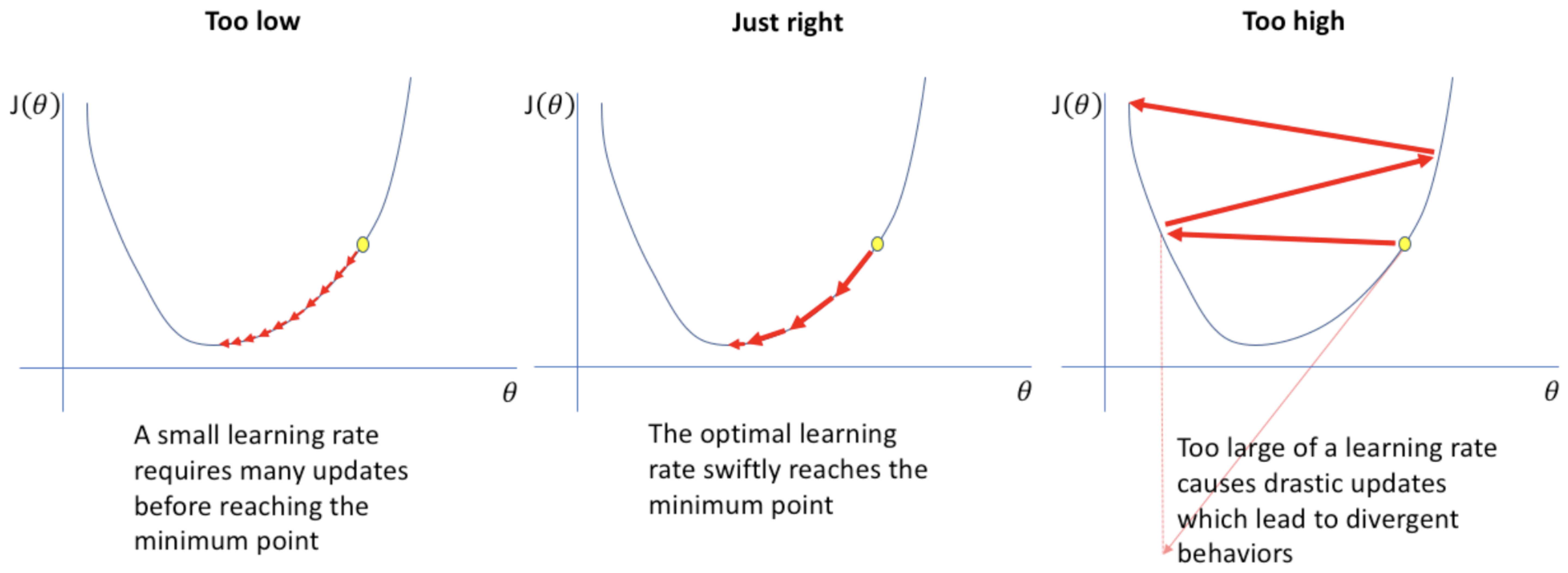
1. $w_i = 1$? ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$? ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
4. ReLU $\rightarrow$ 50% of outputs $= 0 \rightarrow$ additional factor 2
   $\Rightarrow$ He initialization $\sigma = \sqrt{2/n_{in}}$
5. Glorot & He initialization also available for uniform distributions

3. Optimizing the training procedure

Optimizing the training procedure

Reminder

Convergence depends on learning rate



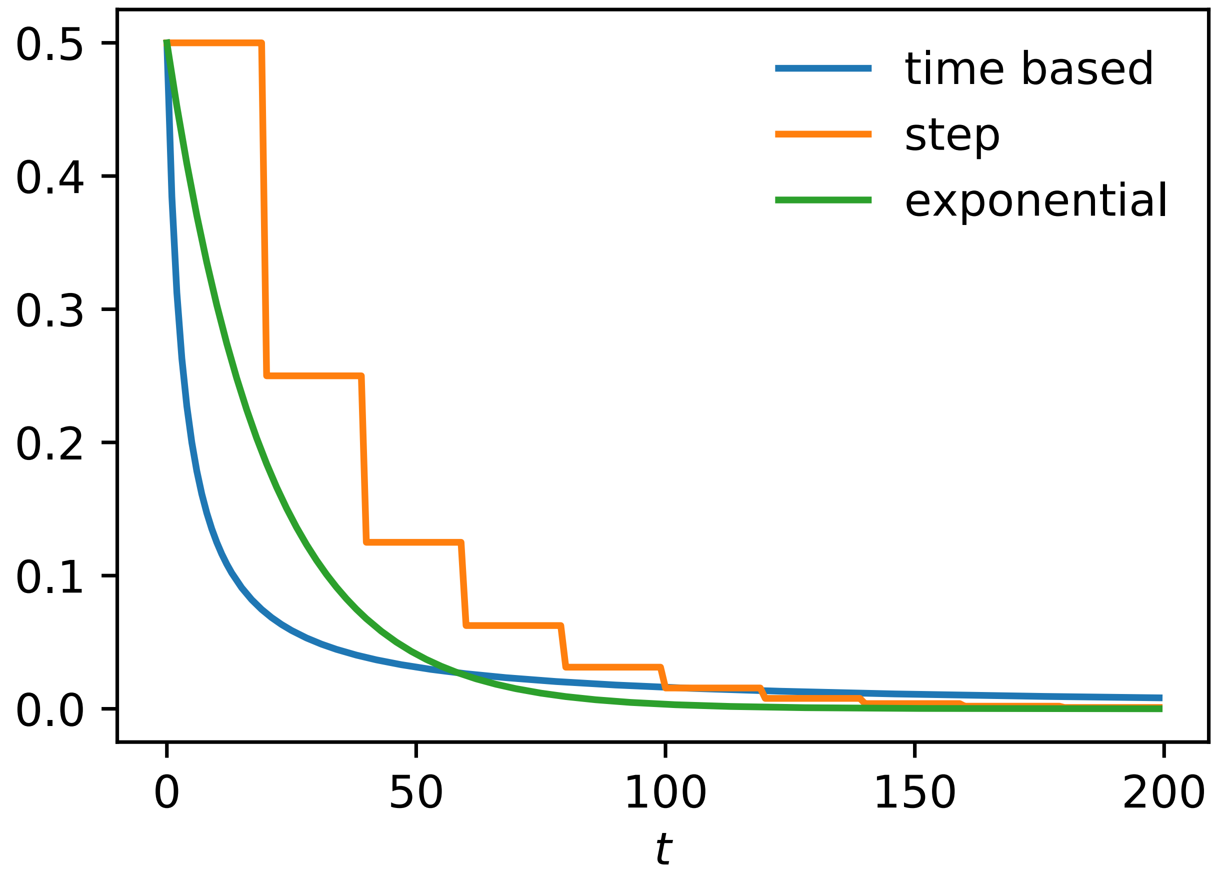
https://www.jeremyjordan.me/nn-learning-rate/

$\rightarrow$ Experiment with different orders of magnitude eg. $10^{-1} \dots 10^{-6}$

Learn rate decay

Reduce learning rate over time to improve convergence

Time-Based Decay	$l(t) = \dfrac{l_0}{1 + k * t}$	
Step Decay	$l(t) = l_0 * \lambda^{int(t/\tau)}$	with $0 < \lambda < 1$
Exponential Decay	$l(t) = l_0 * e^{-t/\tau}$	

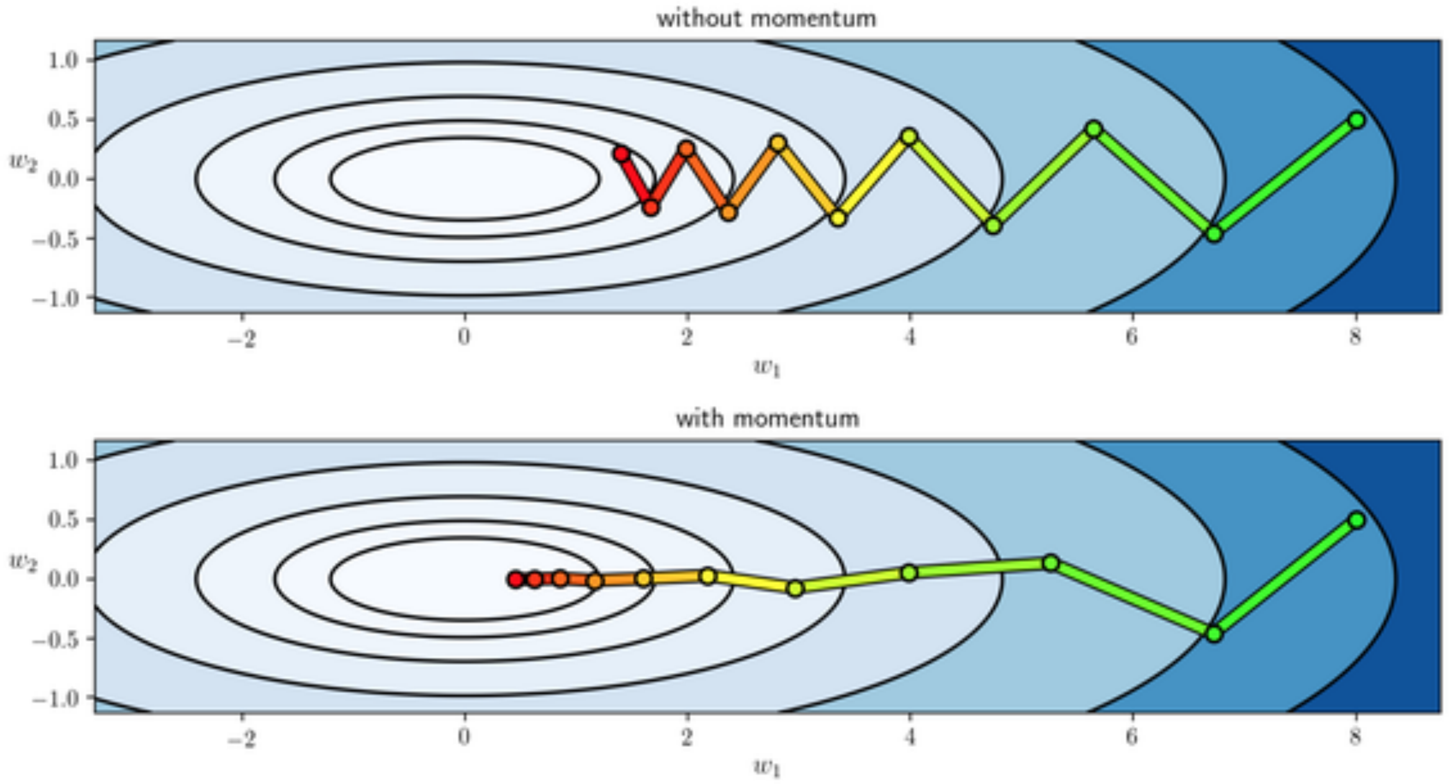


Momentum

Problem: One dimension much steeper than the other

gradient descent	$\boldsymbol{w}_t \to \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \nabla_{\boldsymbol{w}_t} \mathcal{L}$
GD + momentum	$\boldsymbol{w}_t \to \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha v_{dw}$
	$v_{dw} = \beta v_{dw} + (1-\beta) \nabla_{\boldsymbol{w}_t} \mathcal{L}$

Intuition: ball picks up momentum



jermwatt.github.io/machine_learning_refined

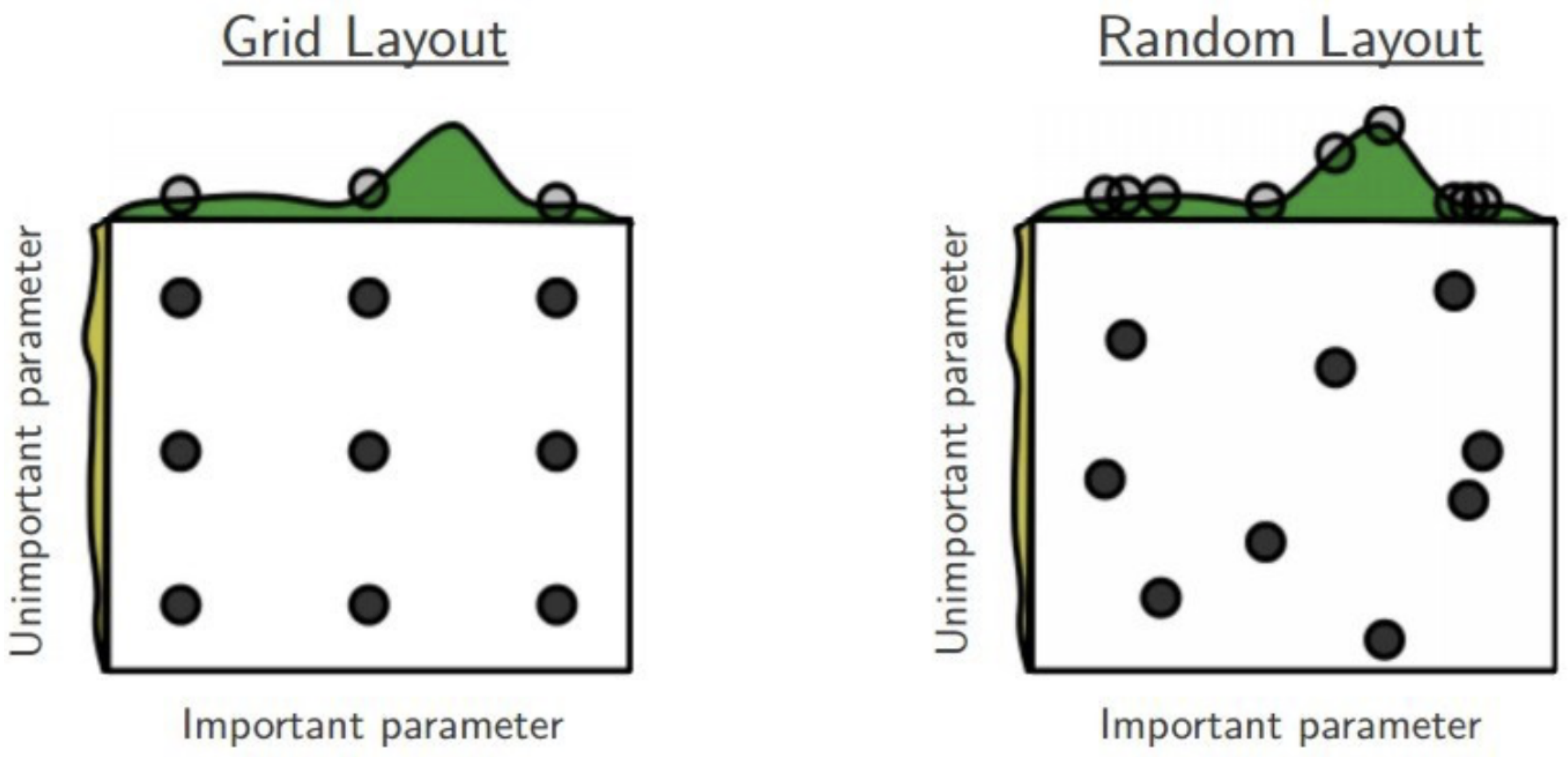
enforces dimensions where gradient points in same direction
+ reduces oscillation

Hyperparameter tuning

How can we find the best settings for the training?
Problem: We can not compute a gradient!

1. by hand $\rightarrow$ underrated, helps to build experience
2. Grid search
3. Random (blind)
4. Bayesian optimization (educated guess, advanced)

Advantage of random vs grid search



Advantages: easy to code, run parallel
Disadvantage: no use of information from previous iterations, curse of dimensionality

A physicist's network

ML for big data in particle physics

Classification

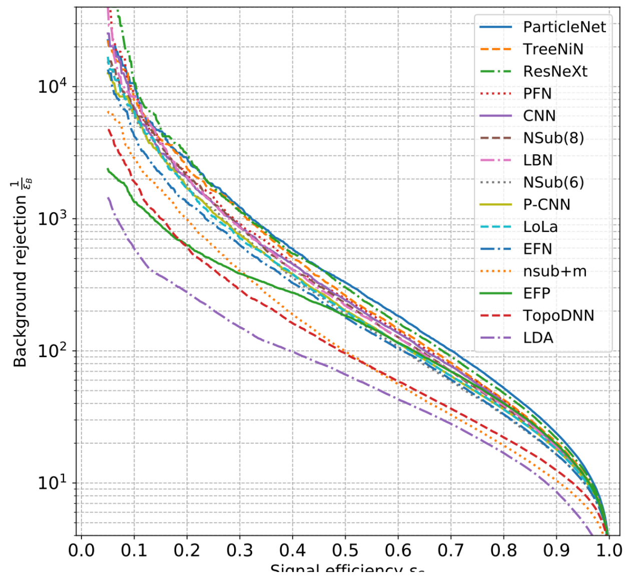
Generative models

Graph networks

Bayesian networks

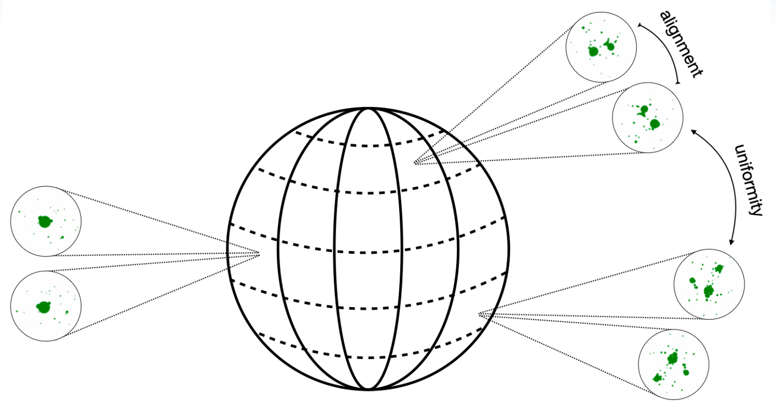
Regression

Top tagging



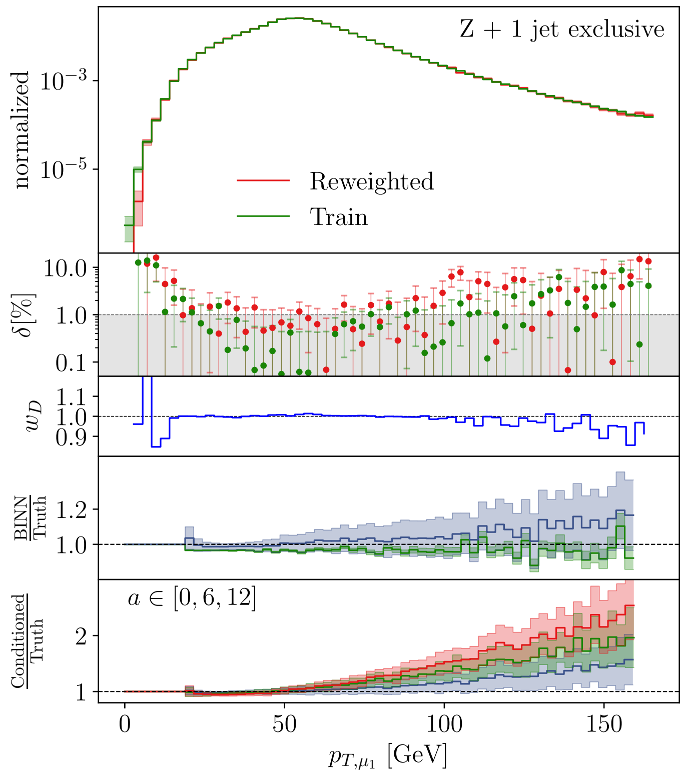
G. Kasieczka et al. [1902.09914]

Anomaly detection



B. Dillon et al. [2108.04253]

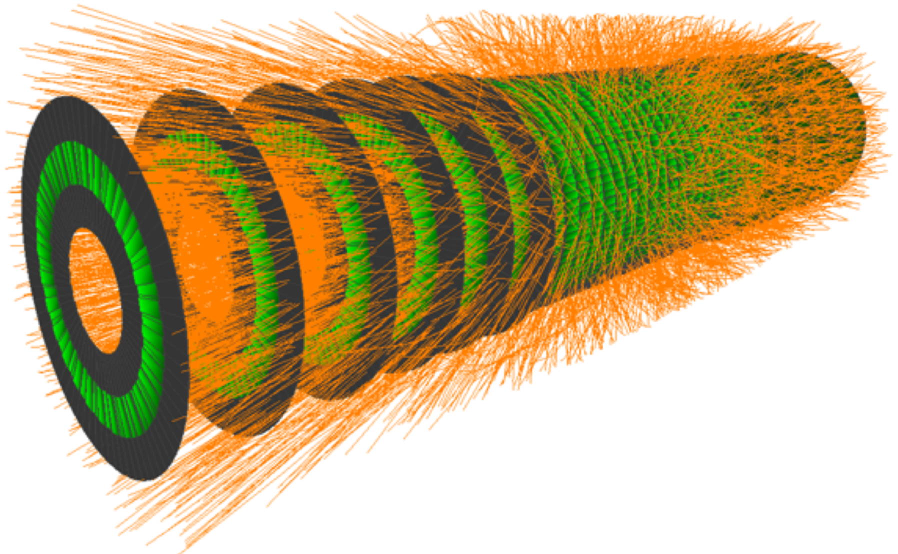
Event generation



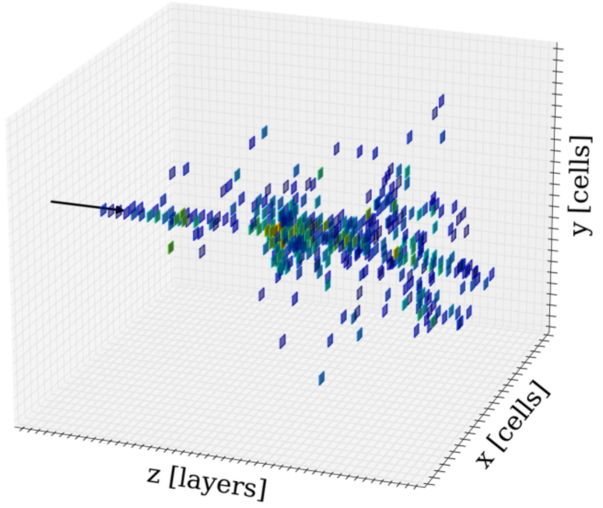
A. Butter et al. [2110.13632]

Track reconstruction

Kaggle challenge

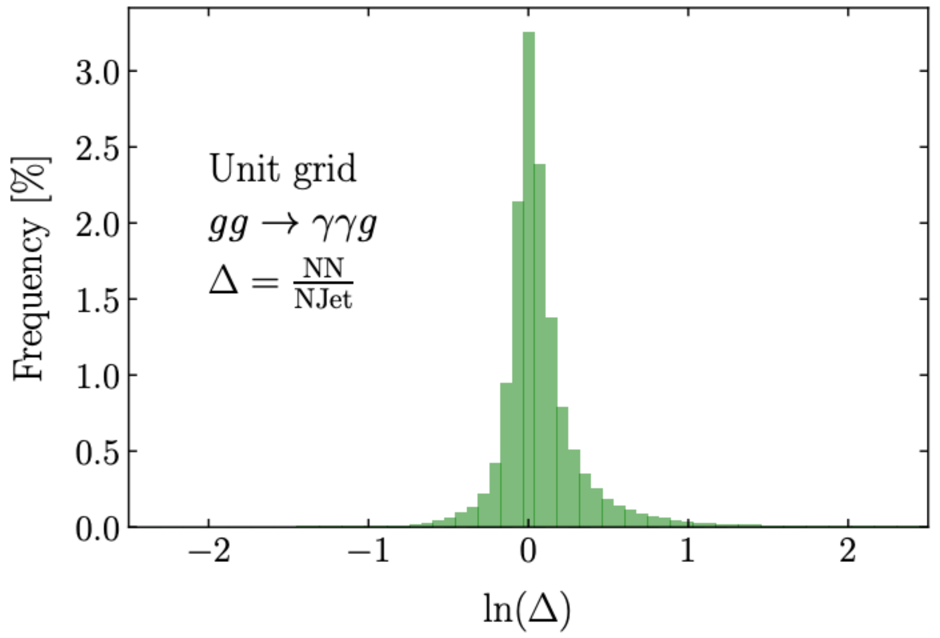


Detector simulation



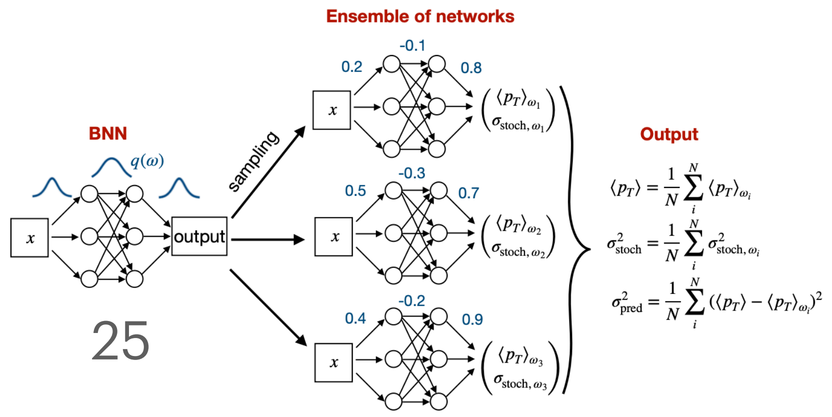
E. Buhmann et al. [2112.09709]

Amplitude estimation



J. Aylett-Bullock, et al. [2106.09474]

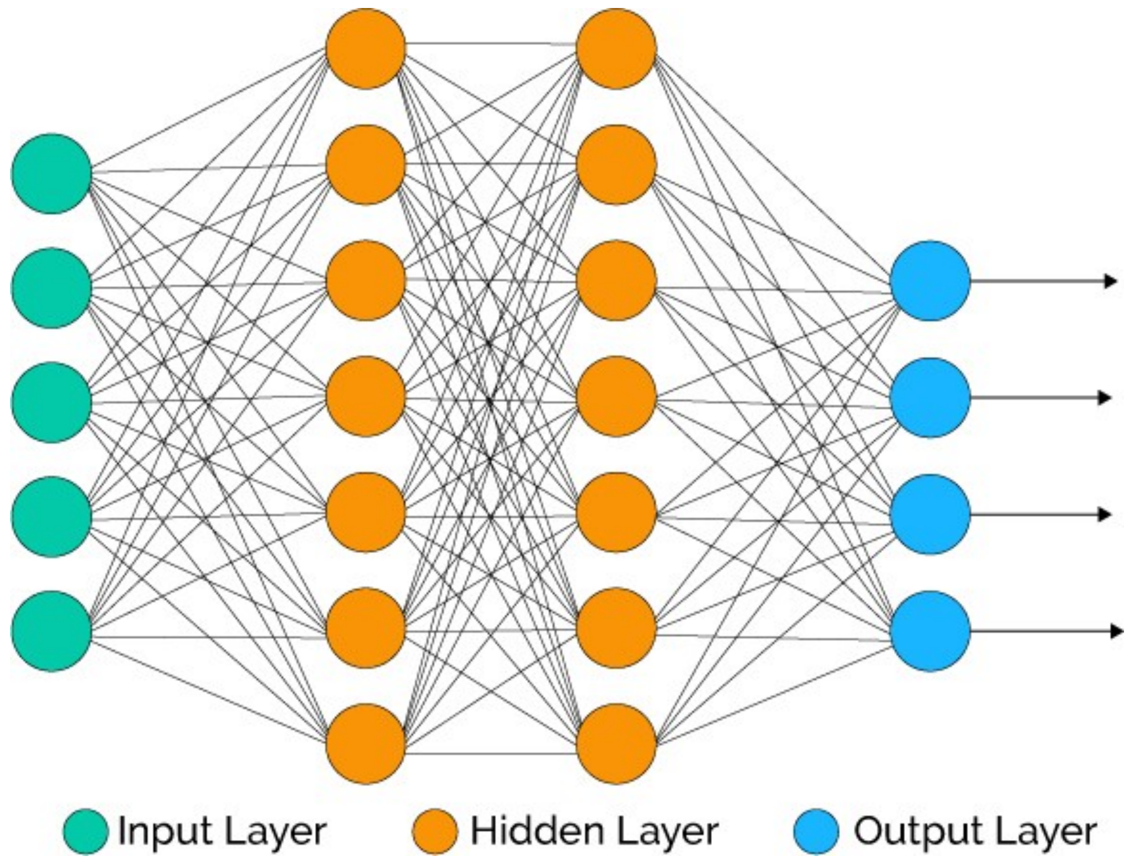
Jet calibration & uncertainties



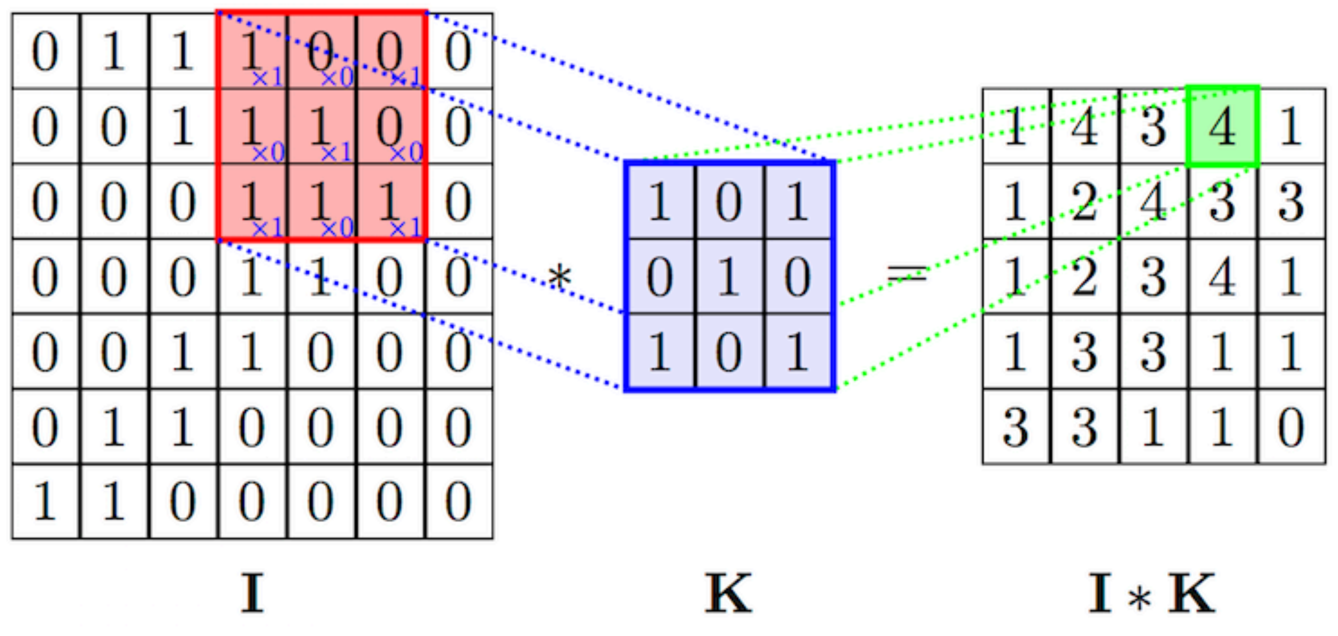
G. Kasieczka et al. [2003.11099]

Complete citations $\mathcal{O}(800)$
https://iml-wg.github.io/HEPML-LivingReview/

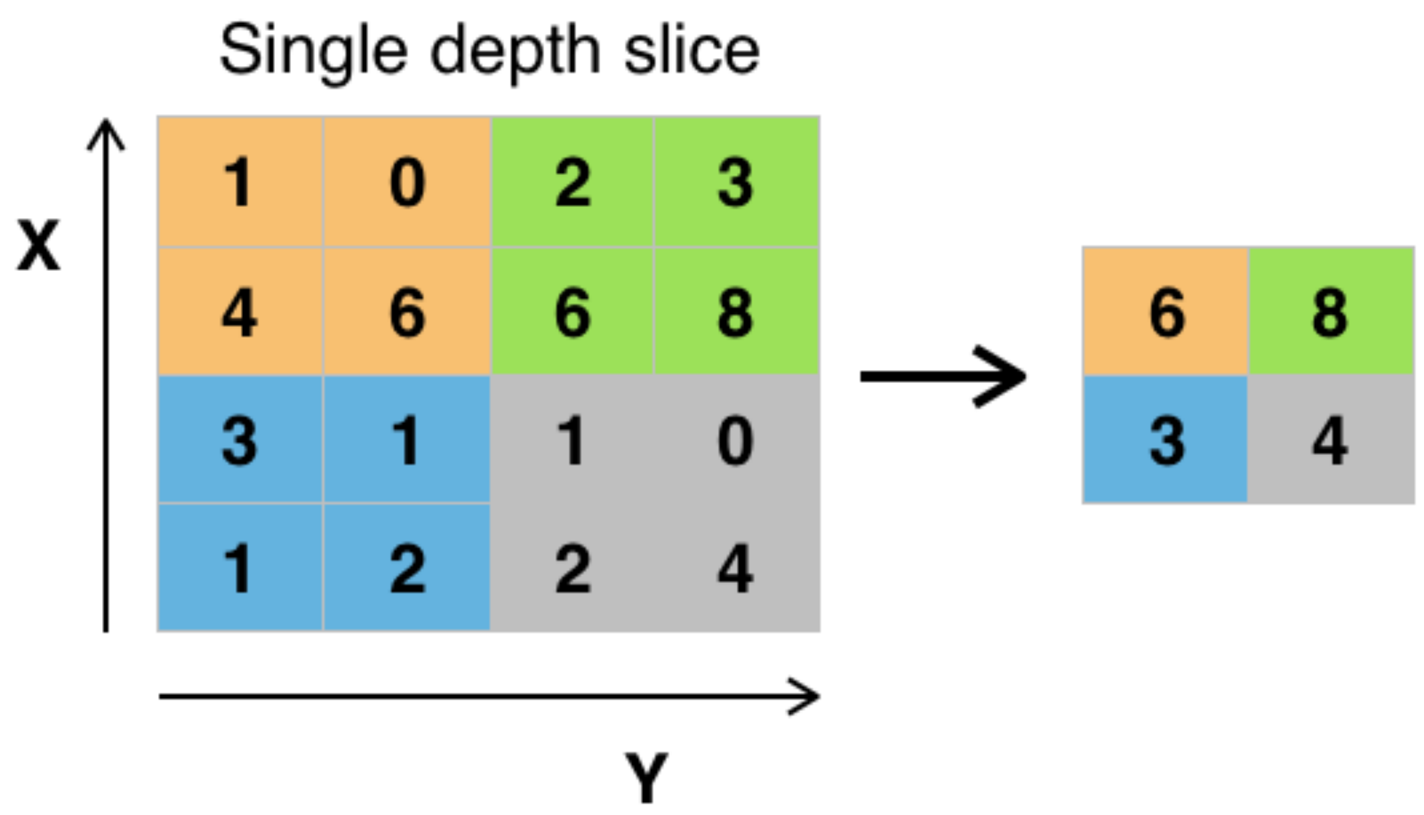
Different types of networks



Dense networks
Standard network



Convolutional neural network (CNN)
Implement **equivariance**

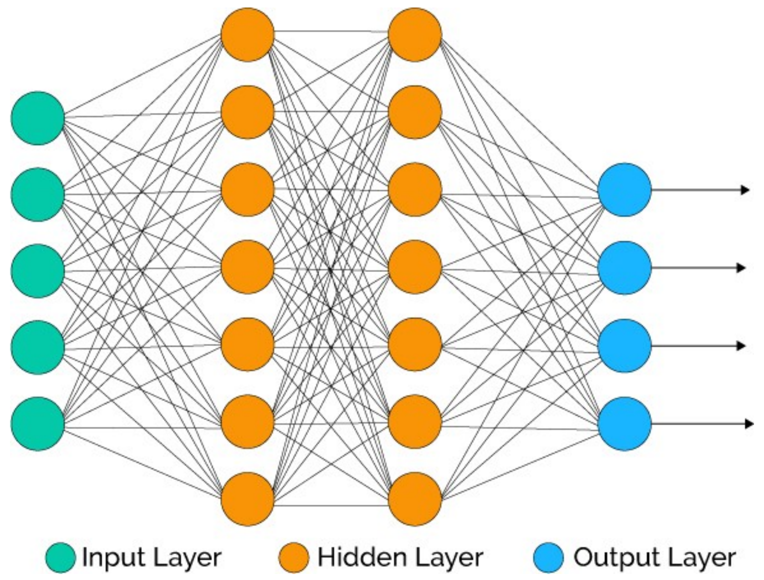


Pooling layer (max/min/mean/std)
Implement **invariance**

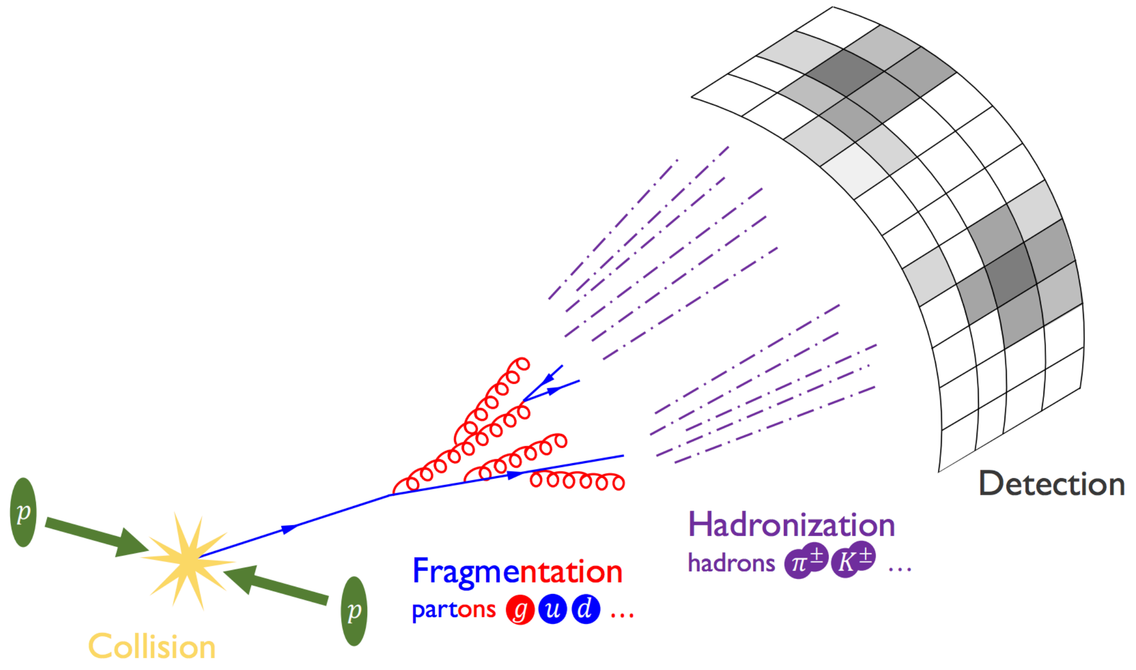
Data determine the network

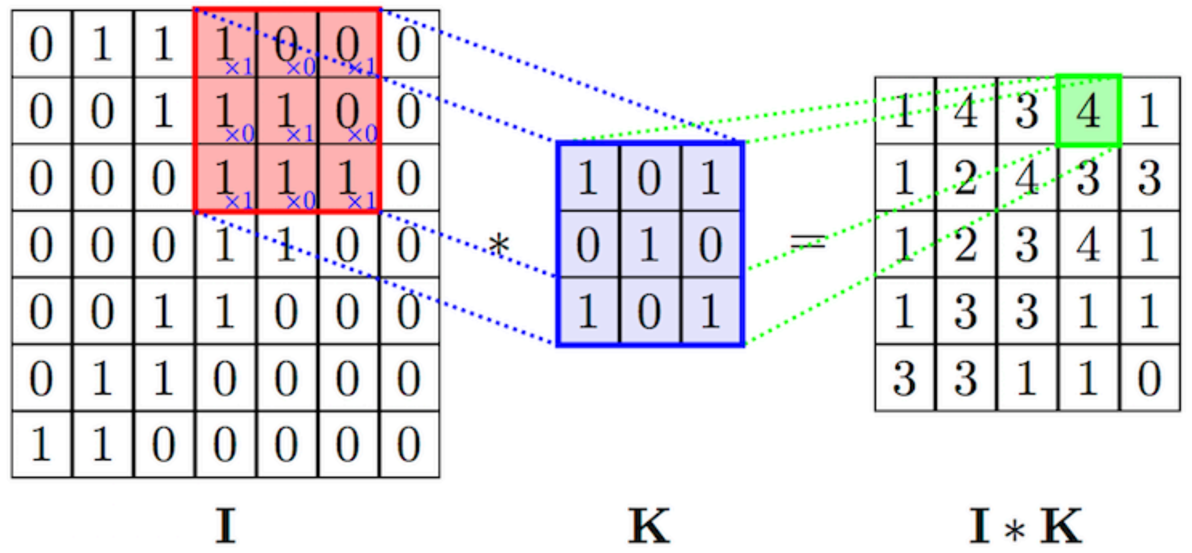
Data with intrinsic order
Example: events with structure

$$event = [p_{T,e^+}, p_{T,e^-}, \eta_{e^+}, \eta_{e^-}, p_{T,j}]$$

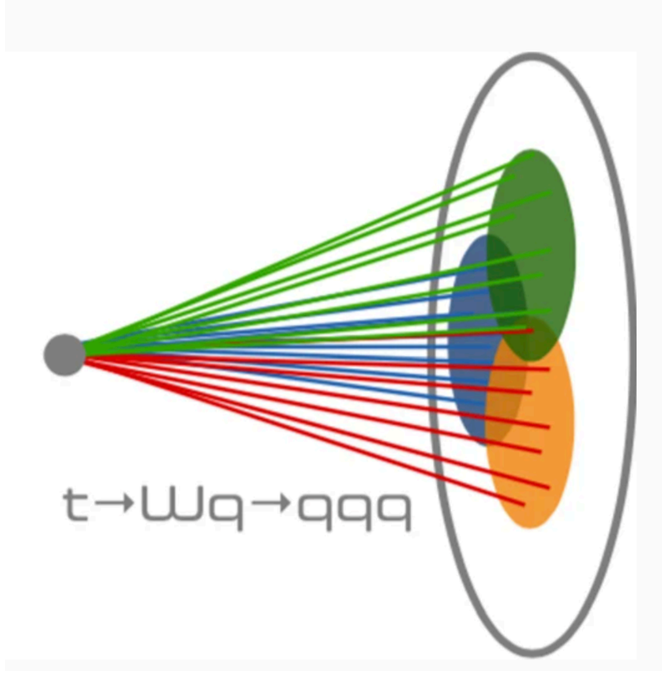


Images
Example: Calorimeter cells





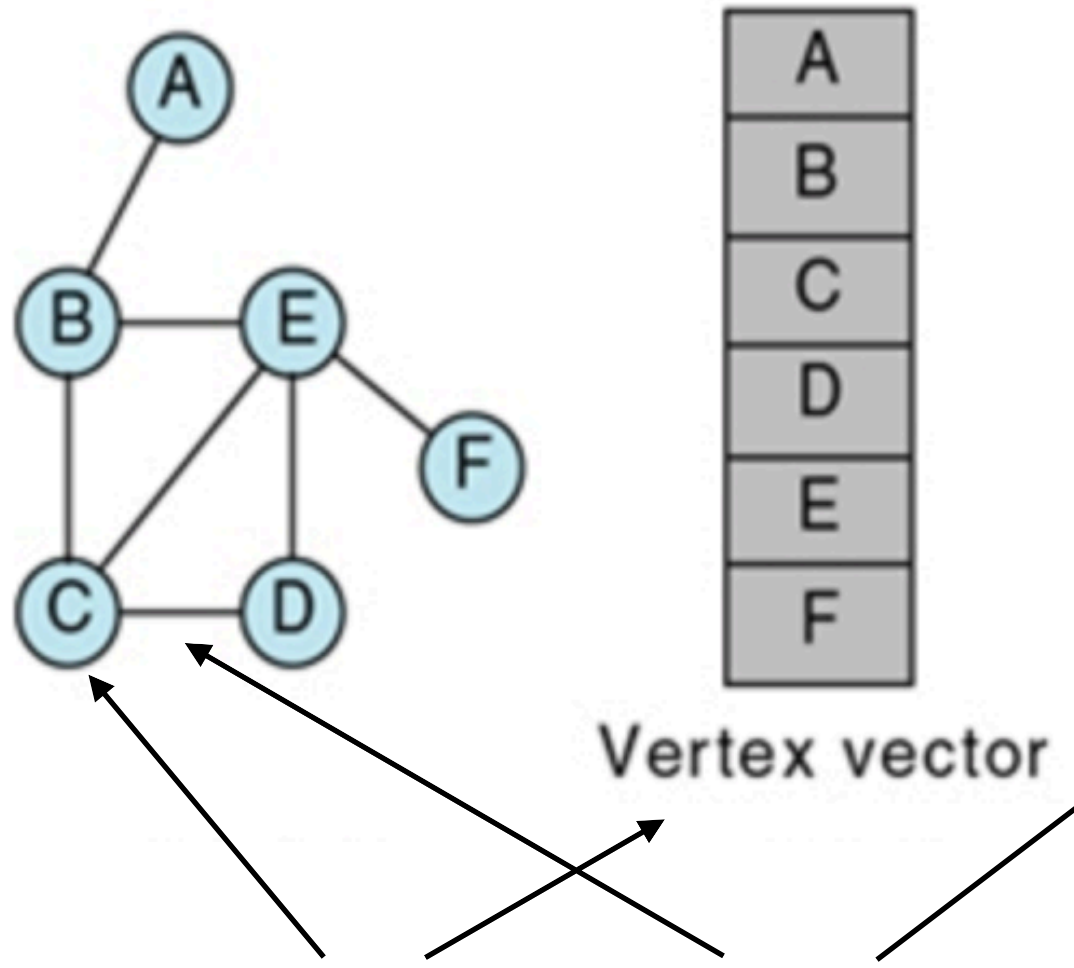
Unordered sets
Example: Jet constituents



?

Graph networks

How to represent a graph

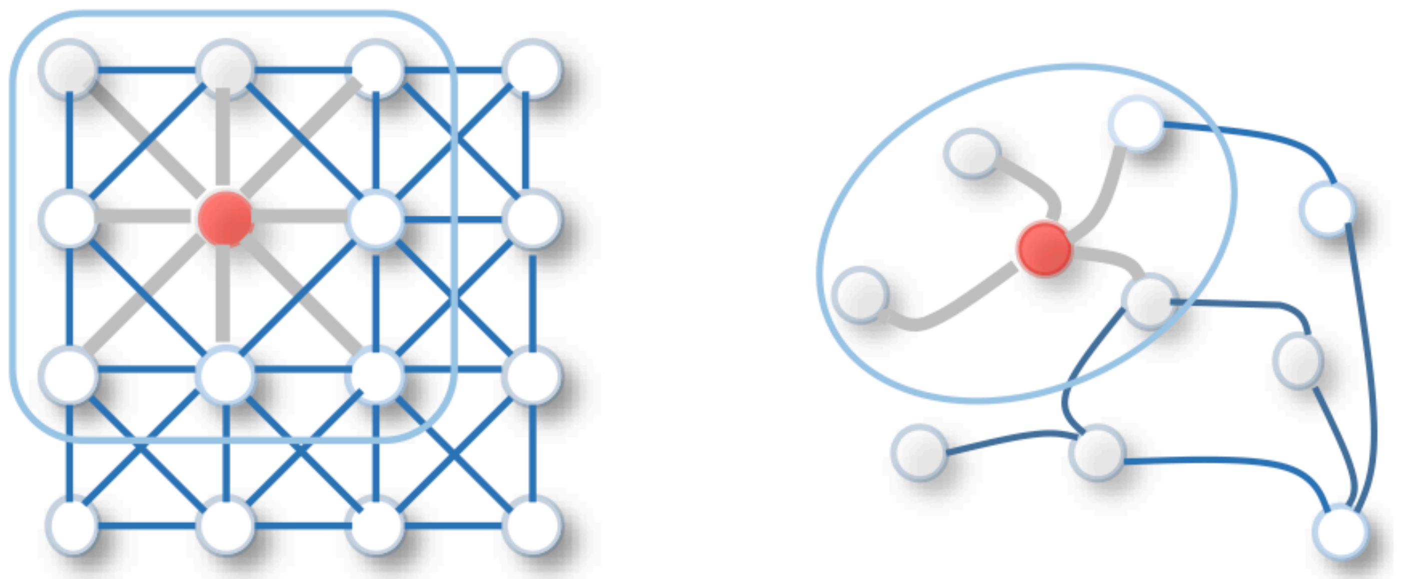


	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	1	0
C	0	1	0	1	1	0
D	0	0	1	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

Adjacency matrix

Node Edge

Image vs Graph

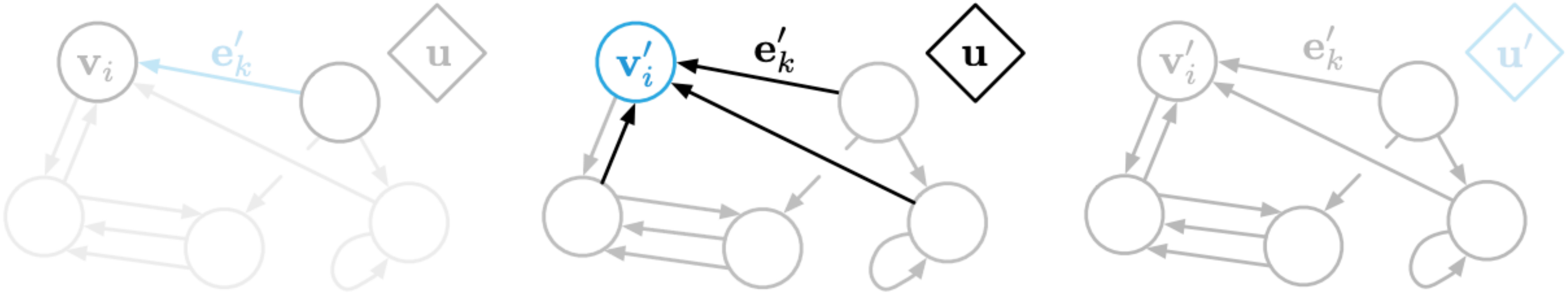


pixels → node

neighbouring pixel → neighbouring node (graph edges)

Graph networks

1806.01261



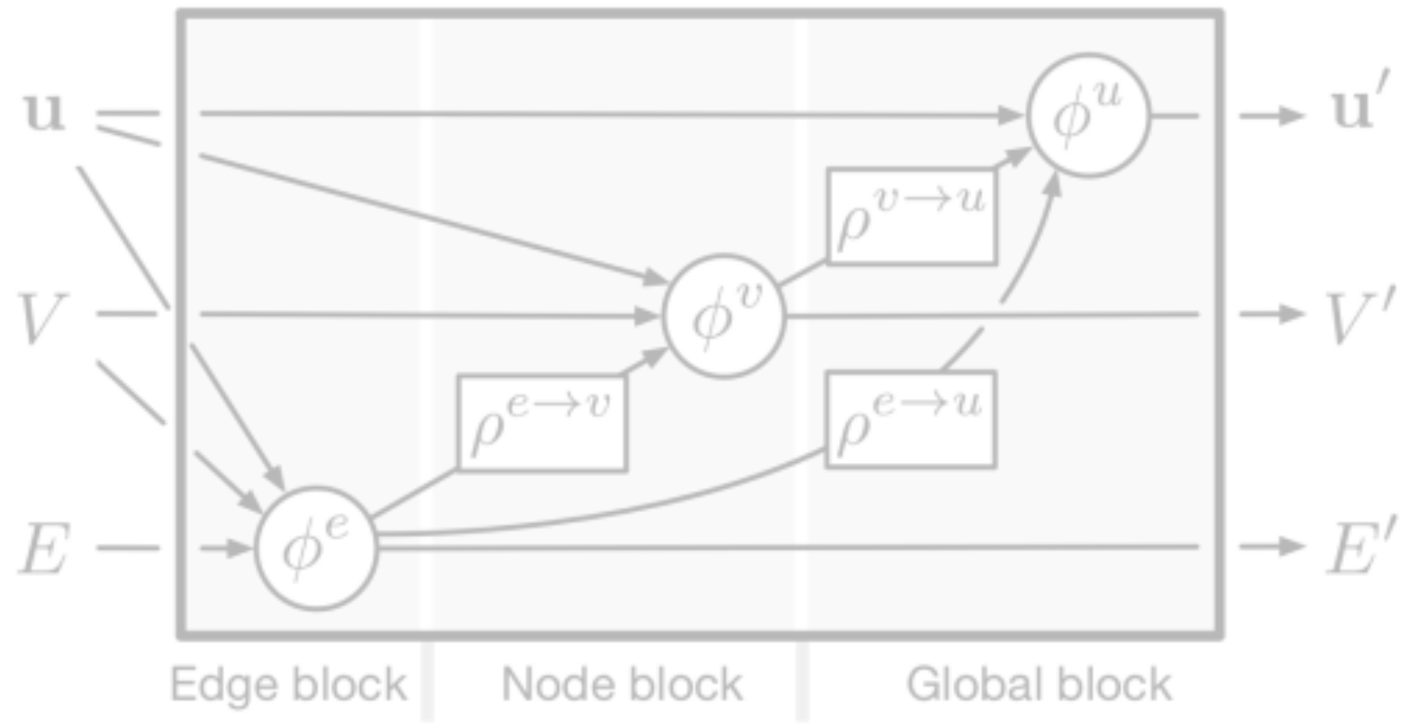
(a) Edge update (b) Node update (c) Global update

→ edge convolution

$$\vec{v}_i' = \frac{1}{k} \sum_{j=1}^{k} h_\Theta(\vec{v}_i, \vec{v}_{i_j} - \vec{v}_i)$$

Aggregation function

h* is independent of *i*, *j



(a) Full GN block

What can we do with graph networks?

Examples

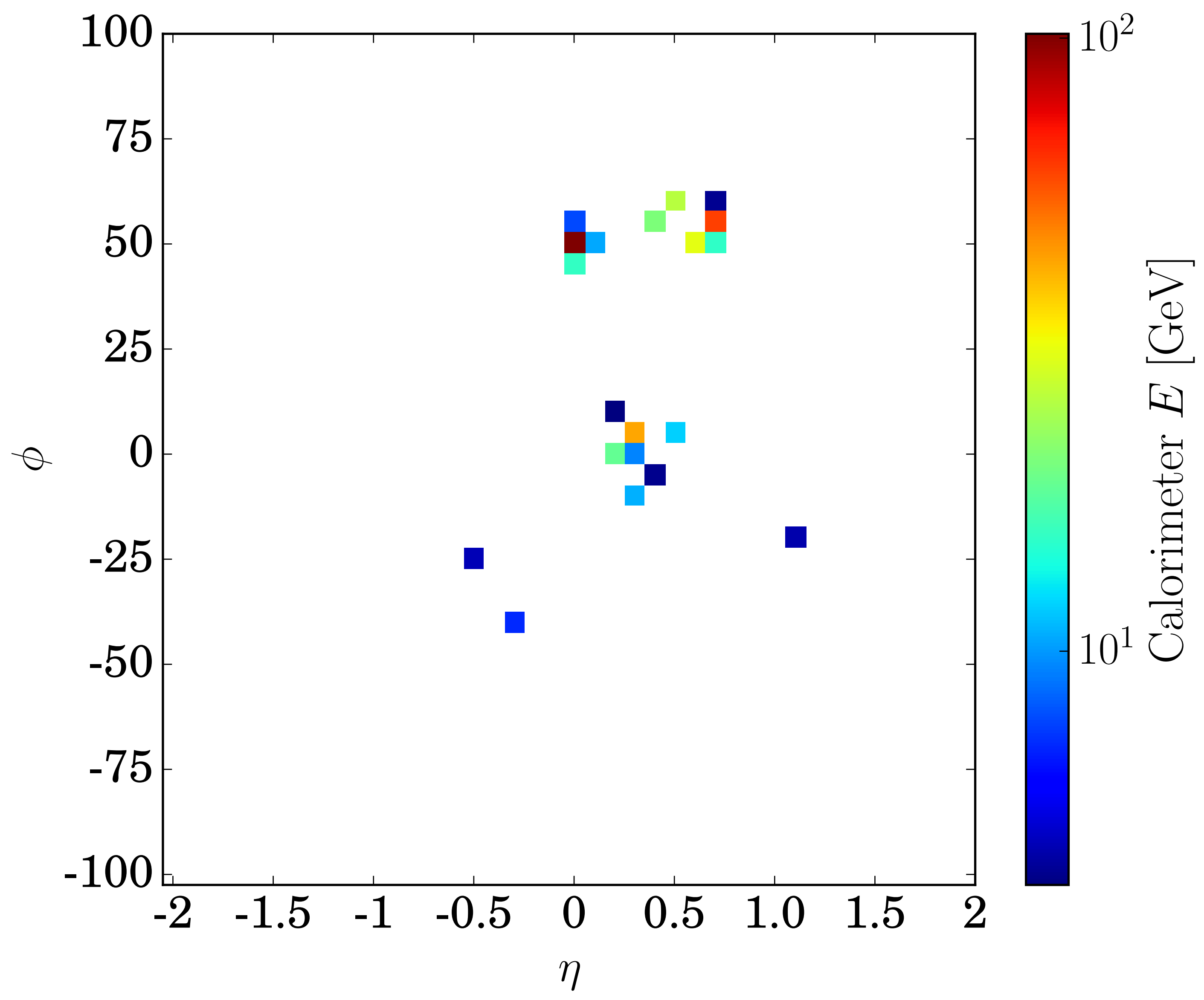
- Node classification (assign label to a node)
  - *Does this hit belong to my track?*
- Graph classification (assign label to graph)
  - *Top vs QCD jet*
  - *B-jet identification*
  - *Event classification (Signal vs Background)*
- Graph generation
  - *Generate new jet*
- Embedding into alternative space for better interpretation

Top jet classification

1707.08966

Data set

- Top vs QCD
- Calorimeter image & Particle Flow objects
- Pythia8 + Delphes 3
- FastJet3 anti-kt with R = 1.5
- $|\eta_{fat}| < 1.0$, $p_{T,jet} = 350 \ldots 450$ GeV



Calorimeter image:
Mostly empty & No tracking information

→ CNN not suited

Instead:
→ Set of particle flow objects
→ They become set of nodes

Optional: Build graph for instance from nearest neighbors

Lorentz Layer

Physics inspired layer that acts on nodes [1707.08966]

Transform Lorentz vectors into physics motivated objects.

$$\tilde{k}_j \xrightarrow{\text{LoLa}} \hat{k}_j = \begin{pmatrix} m^2(\tilde{k}_j) \\ p_T(\tilde{k}_j) \\ w^{(E)}_{jm}\, E(\tilde{k}_m) \\ w^{(d)}_{jm}\, d^2_{jm} \end{pmatrix}$$

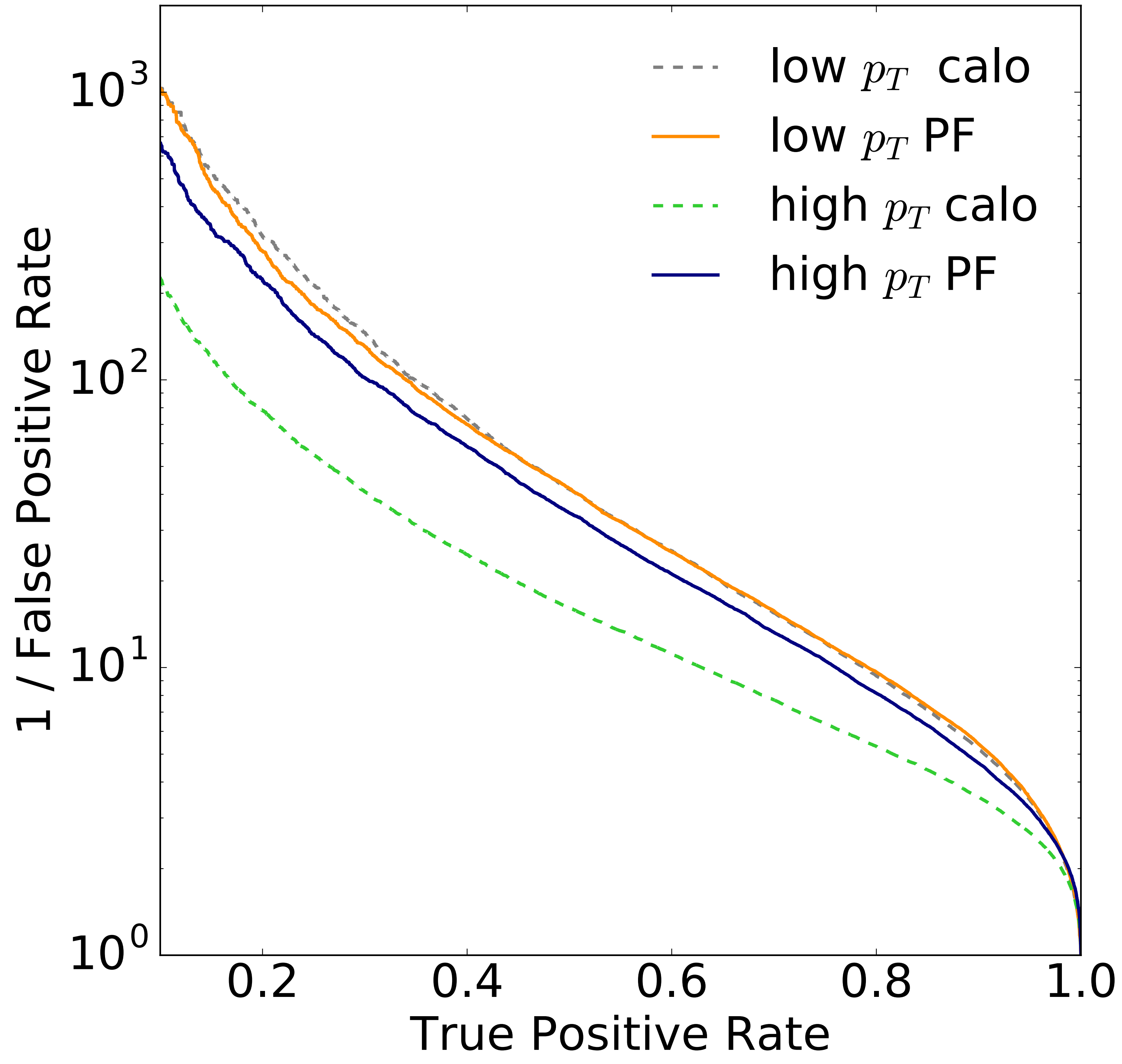
$$d^2_{jm} = (\tilde{k}_j - \tilde{k}_m)_\mu \, g^{\mu\nu} \, (\tilde{k}_j - \tilde{k}_m)_\nu$$

Transformation in place

Aggregation over other objects

Distance d_{jm} encodes edge information

Not exactly graph concept, as weights are index dependent



At high p_T:

PF based network outperforms CNN

→ tracking information is crucial !

ParticleNet

[1902.08570, H. Qu, L. Gouskos]

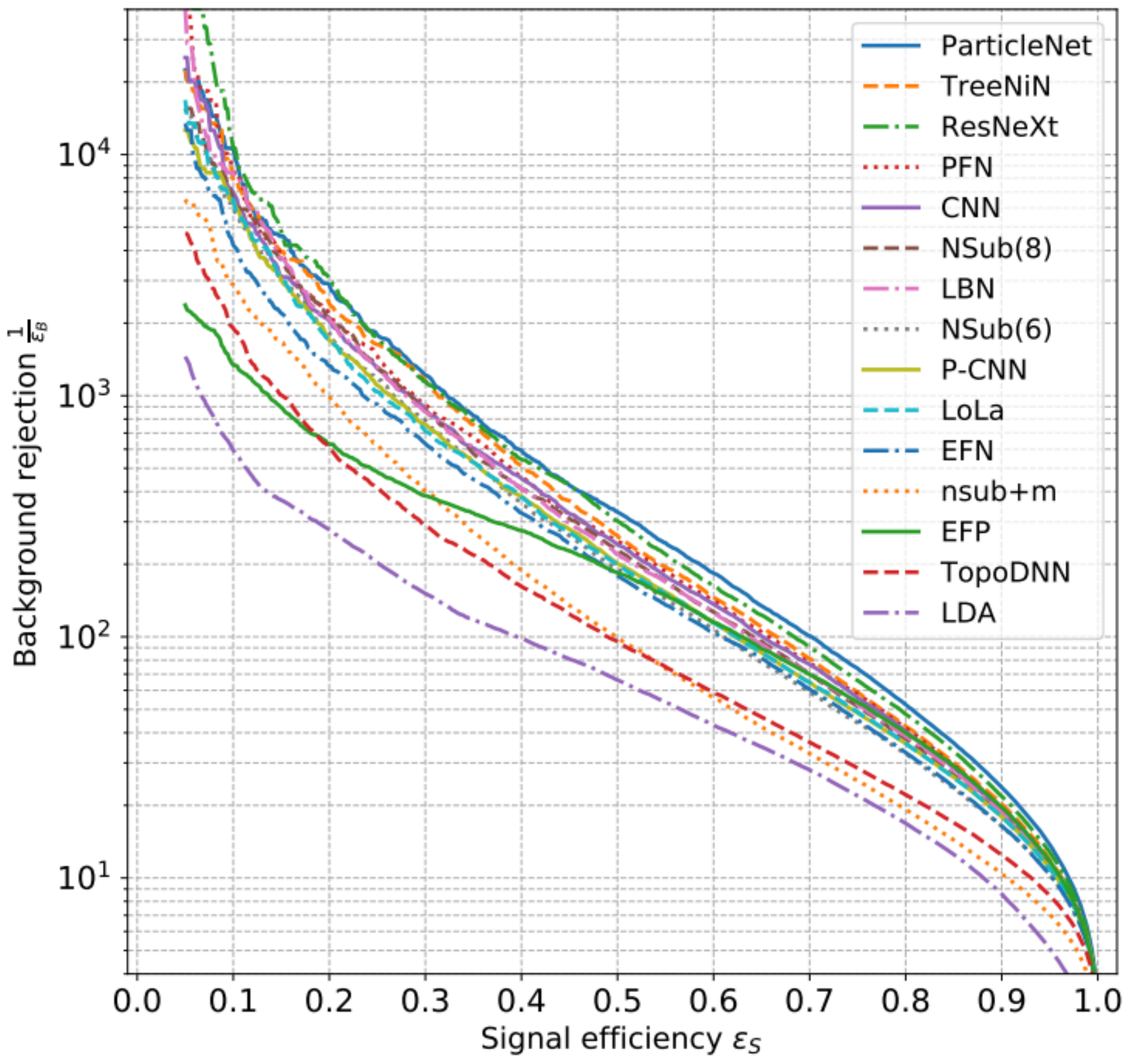
- Jet = unordered set of particles
- Particle cloud (permutation invariant)
- Translational symmetry

- K-nearest neighbours define local patch
  $x_i' = \boxdot_{j=1}^{k} \phi_\theta(x_i, x_{i_j} - x_i)$
  - $\boxdot$ indicates an aggregation function (max, **mean**, sum, ...)
  - ϕ_θ is a 3 layer MLP

- Dynamically update edges for each layer

- Hyperparameter:
  - # neighbors, latent dim, dropout, batchnorm, learning rate, ...

Variable
$\Delta\eta$
$\Delta\phi$
$\log p_T$
$\log E$
$\log \frac{p_T}{p_T(\text{jet})}$
$\log \frac{E}{E(\text{jet})}$
ΔR
q
isElectron
isMuon
isChargedHadron
isNeutralHadron
isPhoton



Lorentz Net

2201.08187, S. Gong et al.

Combination of graph network and physics knowledge

Lorentz Net encodes Lorentz **equivariance**

$$x_i^{l+1} = x_i^l + c \sum_{j \in [N]} \phi_x(m_{ij}^l) \cdot x_j^l$$

$$m_{ij}^l = \phi_e \left(h_i^l, h_j^l, \psi(\|x_i^l - x_j^l\|^2), \psi(\langle x_i^l, x_j^l \rangle) \right)$$

x^0 are the 4-momenta
h^0 embedds charge, PID, etc.
$\langle \cdot , \cdot \rangle$ Minkowski product
$\psi(\cdot) = \mathrm{sgn}(\cdot)\ \log(| \cdot | + 1)$
ϕ_x are neural networks

Top tagging dataset

Training Fraction	Model	Accuracy	AUC	$1/\varepsilon_B$ ($\varepsilon_S = 0.5$)	$1/\varepsilon_B$ ($\varepsilon_S = 0.3$)
0.5%	ParticleNet	0.913	0.9687	77 ± 4	199 ± 14
	LorentzNet	**0.929**	**0.9793**	$\mathbf{176 \pm 14}$	$\mathbf{562 \pm 72}$
1%	ParticleNet	0.919	0.9734	103 ± 5	287 ± 19
	LorentzNet	**0.932**	**0.9812**	$\mathbf{209 \pm 5}$	$\mathbf{697 \pm 58}$
5%	ParticleNet	0.931	0.9807	195 ± 4	609 ± 35
	LorentzNet	**0.937**	**0.9839**	$\mathbf{293 \pm 12}$	$\mathbf{1108 \pm 84}$

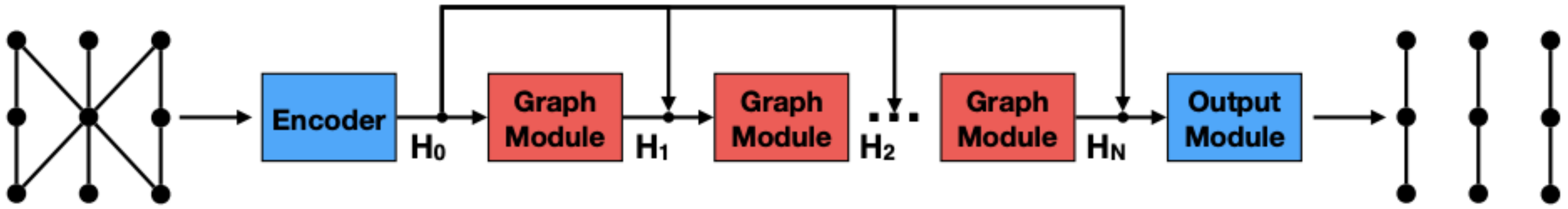
→ Physics layers enable better performance for smaller datasets

Tracking

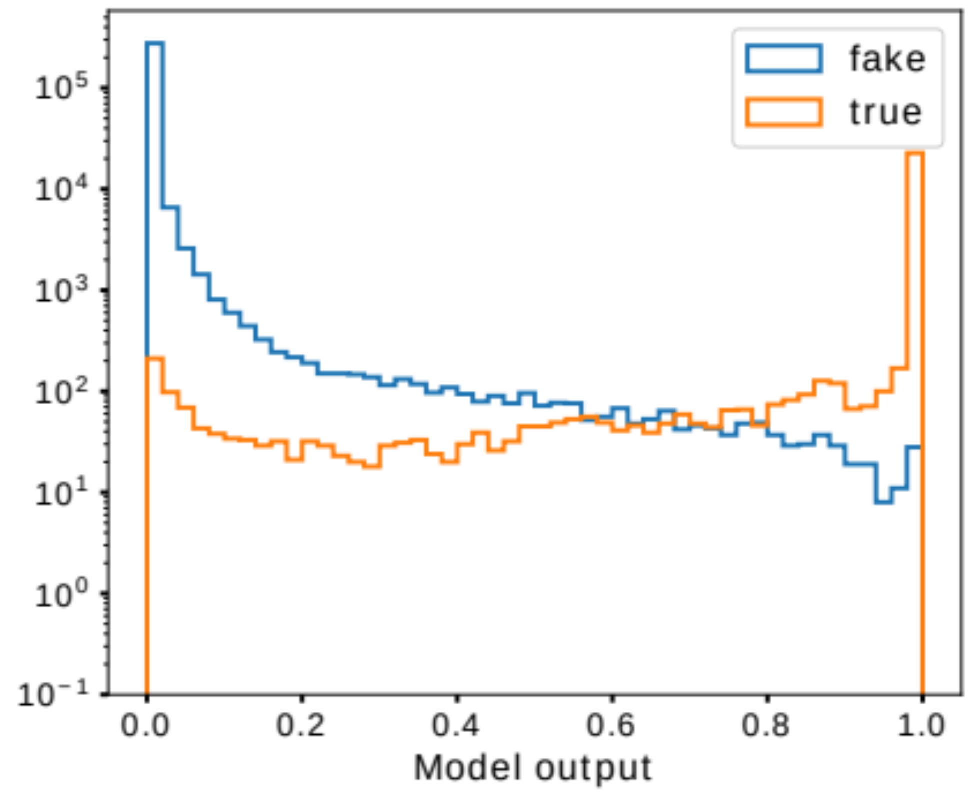
2012.01249 Review by J. Duarte & J.-R. Vlimant

Physics task:
reconstruct tracks from hits in tracker

Graph task:
Edge classification



Which edges truly connect hits from same track?



95.7% purity @ 95.9% efficiency

Summary

- Choose network architecture according to data structure
- Graph networks particularly suitable for unordered sets of objects
- Very efficient training thanks to convolution
- Various applications from top tagging to track reconstruction

Including physics based layers makes networks more efficient!