UNIVERSITY OF WATERLOO | FACULTY OF SCIENCE

Compression techniques 2

Marco Bonici

MOPED Compression

Massively Optimised Parameter Estimation and Data compression

Method to compress the information that preserves the Fisher Information Matrix

Quick intro to Fisher Matrix

Let us assume we have a likelihood with a single peak at θ_0. We can then Taylor expand the likelihood as

$$\ln L(\mathbf{x};\boldsymbol{\theta}) = \ln L\left(\mathbf{x};\boldsymbol{\theta}_0\right) + \tfrac{1}{2}(\boldsymbol{\theta}_\alpha - \boldsymbol{\theta}_{0\alpha})\frac{\partial^2 \ln L}{\partial\boldsymbol{\theta}_\alpha \partial\boldsymbol{\theta}_\beta}(\boldsymbol{\theta}_\beta - \boldsymbol{\theta}_{0\beta}) + \ldots$$

and the Fisher Matrix is defined as

$$\mathrm{F}_{\alpha\beta} \equiv \langle \mathrm{H}_{\alpha\beta} \rangle = \left\langle -\frac{\partial^2 \ln L}{\partial\boldsymbol{\theta}_\alpha \partial\boldsymbol{\theta}_\beta} \right\rangle$$

then for any unbiased estimator $\Delta\theta_\alpha \geq 1/\sqrt{F_{\alpha\alpha}}$ (Cramer-Rao Bound).

MOPED Compression

Let us consider a data vector with N elements and a theoretical model with m parameters. Is there a way to compress the data vector in such a way that we waste as little as possible information?

Let us assume we are considering data distributed according to a MvNormal

$$2\mathcal{L} = \ln\det \boldsymbol{C} + (\boldsymbol{x}-\boldsymbol{\mu})\boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})^t$$

Then the Fisher matrix is given by

$$\boldsymbol{F}_{ij} = \tfrac{1}{2}\mathrm{Tr}\left[\boldsymbol{A}_i\boldsymbol{A}_j + \boldsymbol{C}^{-1}\boldsymbol{M}_{ij}\right]$$

with

$$\boldsymbol{A}_i \equiv \boldsymbol{C}^{-1}\boldsymbol{C}_{,i} = (\ln \boldsymbol{C})_i \qquad \boldsymbol{M}_{ij} \equiv \langle \boldsymbol{D}_{,ij}\rangle = \boldsymbol{\mu}_{,i}\,\boldsymbol{\mu}^t_{,j} + \boldsymbol{\mu}_{,j}\,\boldsymbol{\mu}^t_{,i}$$

The most general linear compression is given by

Then the Fisher Matrix changes according to

$$\tilde{\boldsymbol{F}}_{ij} = \frac{1}{2}\mathrm{Tr}\left[\left(\boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^t\right)^{-1}\left(\boldsymbol{B}\boldsymbol{C}_{,i}\boldsymbol{B}^t\right)\left(\boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^t\right)^{-1}\left(\boldsymbol{B}\boldsymbol{C}_{,j}\boldsymbol{B}^t\right) + \left(\boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^t\right)^{-1}\left(\boldsymbol{B}\boldsymbol{M}_{ij}\boldsymbol{B}^t\right)\right]$$

If $\boldsymbol{B}$ is an invertible matrix, then

$$\tilde{\boldsymbol{F}}_{ij} = \tfrac{1}{2}\mathrm{Tr}\left[\boldsymbol{B}^{-t}\left(\boldsymbol{A}_i\boldsymbol{A}_j + \boldsymbol{C}^{-1}\boldsymbol{M}_{ij}\right)\boldsymbol{B}^t\right] = \boldsymbol{F}_{ij}$$

The Fisher information matrix is exactly conserved!

This is a result we might have expected. We have applied an invertible transformation, $\boldsymbol{B}$, without changing the dimensionality of our target space.

This is not the most interesting case, as we want to *compress* the data vector!

Let us assume we have a covariance matrix that does not depend on the model parameters and that we start our compression procedure with a single parameter, then $\boldsymbol{B} = \boldsymbol{b}^t$ and the Fisher matrix becomes

$$\tilde{\boldsymbol{F}}_{ii} = \frac{1}{2}\left(\frac{\boldsymbol{b}^t \boldsymbol{C}_{,i} \boldsymbol{b}}{\boldsymbol{b}^t \boldsymbol{C} \boldsymbol{b}}\right)^2 + \frac{\left(\boldsymbol{b}^t \boldsymbol{\mu}_{,i}\right)^2}{\left(\boldsymbol{b}^t \boldsymbol{C} \boldsymbol{b}\right)}$$

How do we compute the compression vector?

Let us start choosing a normalization, recalling that

$$\begin{cases} \langle \boldsymbol{y} \rangle & = \boldsymbol{B}\boldsymbol{\mu}, \\ \langle \boldsymbol{y}\boldsymbol{y}^t \rangle - \langle \boldsymbol{y} \rangle \langle \boldsymbol{y} \rangle^t & = \boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^t \end{cases}$$

the common choice is

$$\boldsymbol{b}^t \boldsymbol{C} \boldsymbol{b} = 1$$

How do we compute the compression vector?

Let us start choosing a normalization, recalling that

$$\begin{cases} \langle \boldsymbol{y} \rangle & = \boldsymbol{B}\boldsymbol{\mu}, \\ \langle \boldsymbol{y}\boldsymbol{y}^t \rangle - \langle \boldsymbol{y} \rangle \langle \boldsymbol{y} \rangle^t & = \boldsymbol{B}\boldsymbol{C}\boldsymbol{B}^t \end{cases}$$

the common choice is

$$\boldsymbol{b}^t \boldsymbol{C} \boldsymbol{b} = 1$$

This just means that the variance of the new compressed data vector is going to be 1.

We now have a problem we can solve again with Lagrange multipliers (constrained optimization)

$$\frac{\partial}{\partial b_i}\left(b_j \mu_{,1j} b_k \mu_{,1k} - \lambda b_j C_{jk} b_k\right) = 0$$

Which leads to

$$\boldsymbol{\mu}_{,1}\left(\mathbf{b}^t \boldsymbol{\mu}_{,1}\right) = \lambda \mathbf{C}\mathbf{b}$$

and the solution

$$\mathbf{b}_1 = \frac{\mathbf{C}^{-1}\boldsymbol{\mu}_{,1}}{\sqrt{\boldsymbol{\mu}_{,1}^t \mathbf{C}^{-1}\boldsymbol{\mu}_{,1}}}$$

What about the Fisher matrix? Putting our solution in the initial $\boldsymbol{F}$

$$\tilde{\boldsymbol{F}}_{ii} = \left(\boldsymbol{b}^t \boldsymbol{\mu}_{,i}\right)^2$$

gives

$$\tilde{\boldsymbol{F}}_{ii} = \boldsymbol{\mu}_{,i}^t \boldsymbol{C}^{-1} \boldsymbol{\mu}, i = \boldsymbol{F}_{ii}$$

which is *exactly* the original Fisher Matrix!

We have been able to compress the information about our parameter in a single number! This procedure, in case we have a higher number of parameters, leads to

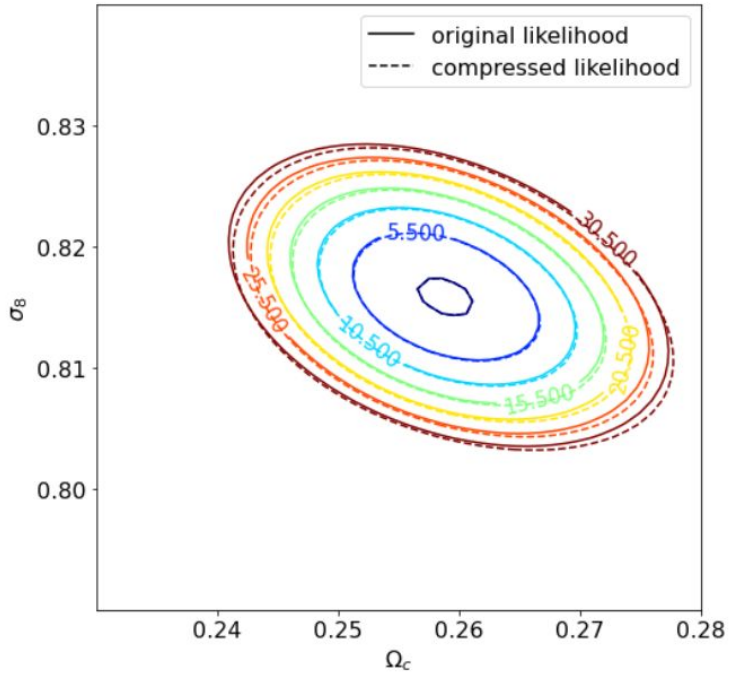
$$b_i = \frac{C^{-1}\mu_{,i} - \sum_{j=1}^{i-1}\left(\mu_{,i}^T b_j\right) b_j}{\sqrt{F_{i,i} - \sum_{j=1}^{i-1}\left(\mu_{,i}^T b_j\right)^2}}$$

MOPED case 1: JAXCosmo (Campagne et al. 2023)

Application to photometric survey (3x2 pt analysis), with a DES-like scenario (datavector with a few hundreds elements)

```
# Orthogonal vectors
b0 = jc.sparse.dot(C_inv,dmu[:,0])/jax.numpy.sqrt(F[0,0])
a = dmu[:,1].T @ b0
b1 = (jc.sparse.dot(C_inv,dmu[:,1]) - a * b0)/jax.numpy.
     sqrt(F[1,1]-a*a)
# MOPED vectors
y0 = b0.T @ data
y1 = b1.T @ data
```

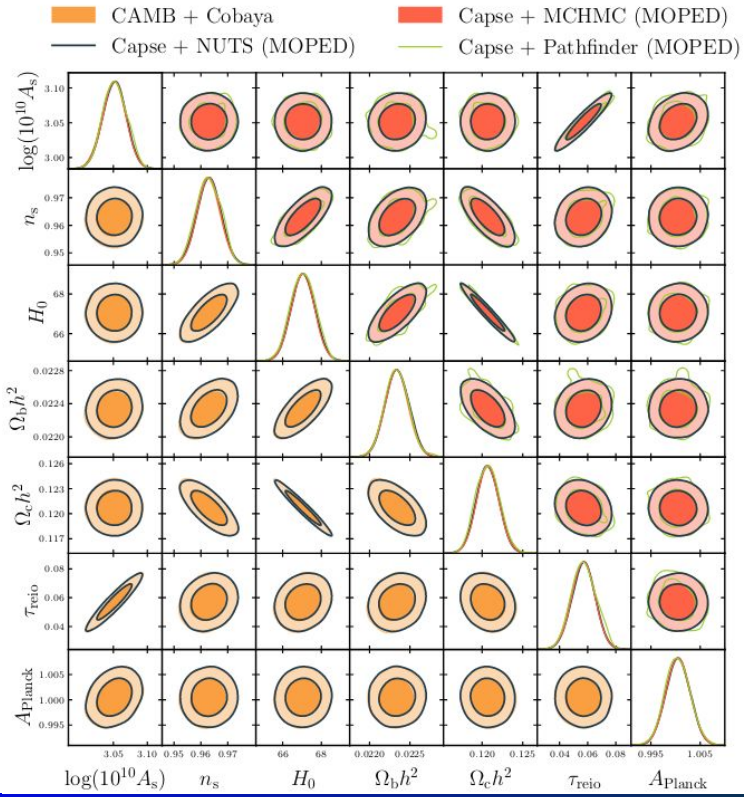
```
@jax.jit
def compressed_likelihood(p):
    # Create a new cosmology at these parameters
    cosmo = jc.Planck15(Omega_c=p[0], sigma8=p[1])
    # Compute mean Cl
    mu = jc.angular_cl.angular_cl(cosmo, ell, tracers).
     flatten()
    # likelihood using the MOPED vector
    return -0.5 * ((y0 - b0.T @ mu)**2 + (y1 - b1.T @ mu)
     **2)
```



MOPED case 2: Capse.jl (Bonici et al. 2024)

Application to the Planck lite likelihood (data vectors with a few houndreds elements!)

Also in this case, the differences on recovered parameters is around 0.1 σ.

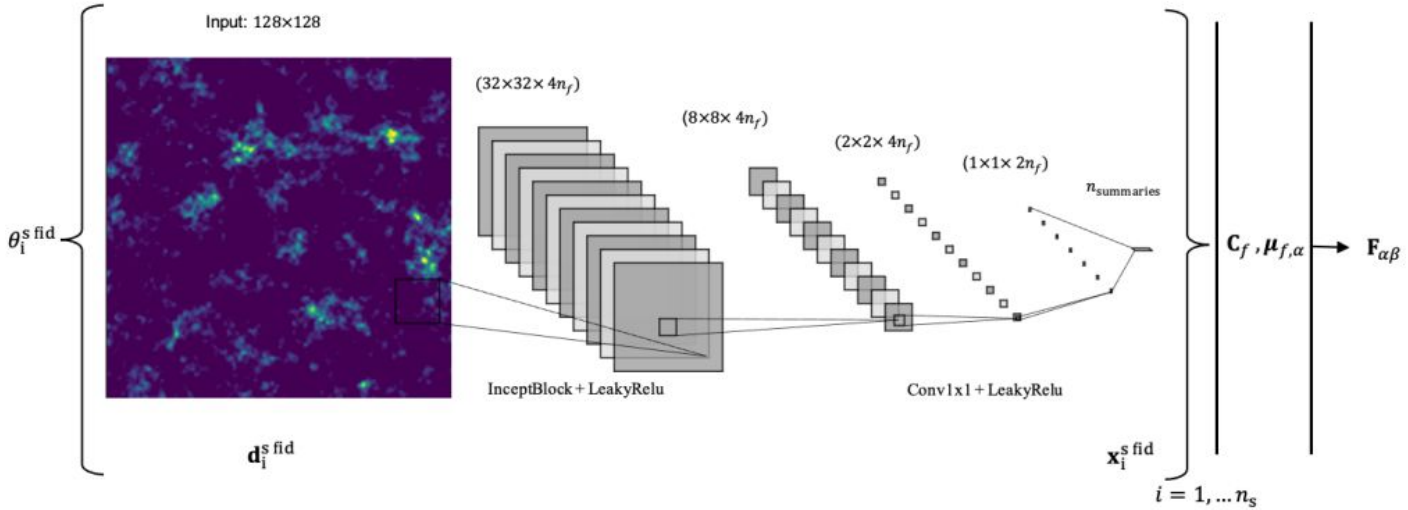


Information Maximizing neural networks

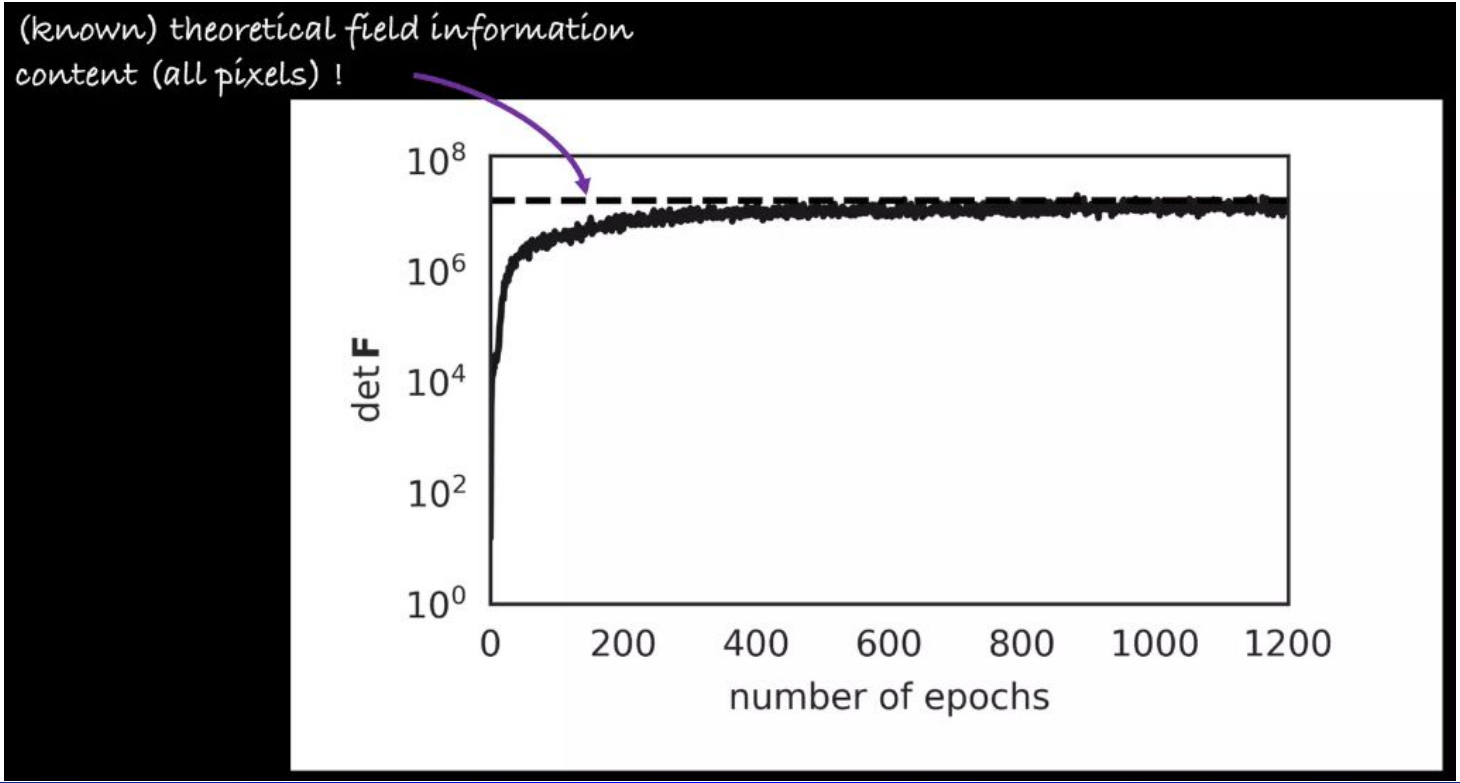
The proposed techniques work nicely in case of *linear* dependence on our parameters. What if we have more complicated, *non-linear*, relationships among our data?

Information Maximizing neural networks

The proposed techniques work nicely in case of *linear* dependence on our parameters. What if we have more complicated, *non-linear*, relationships among our data? We can use a Neural Network!

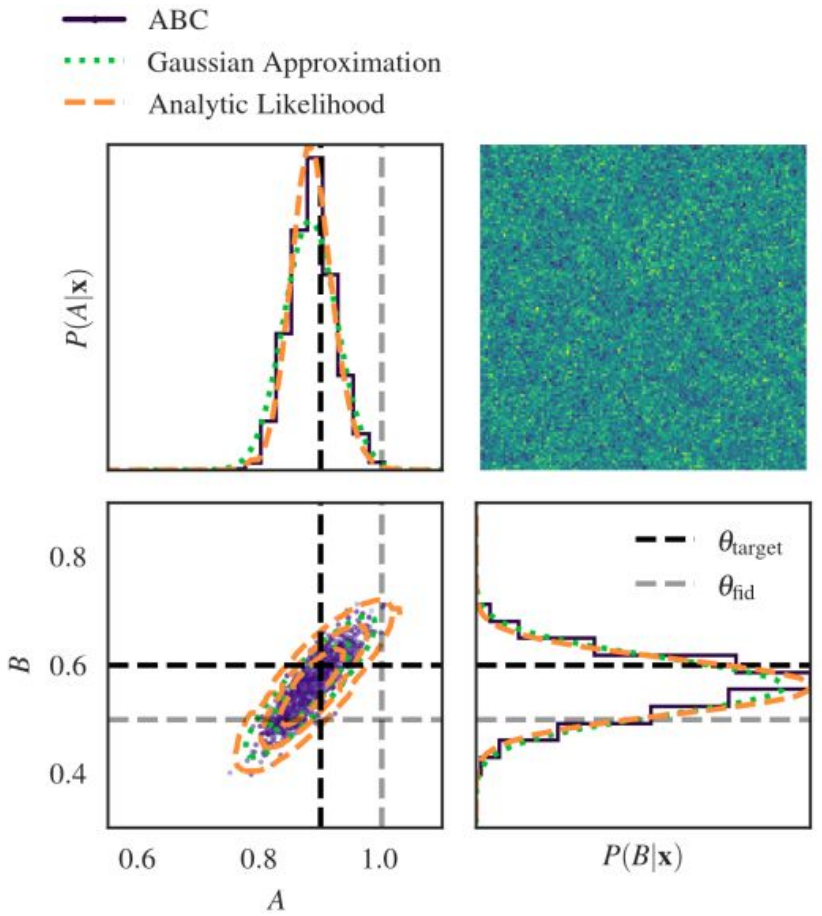


With backpropagation we can train our NN to maximize the Fisher Matrix!



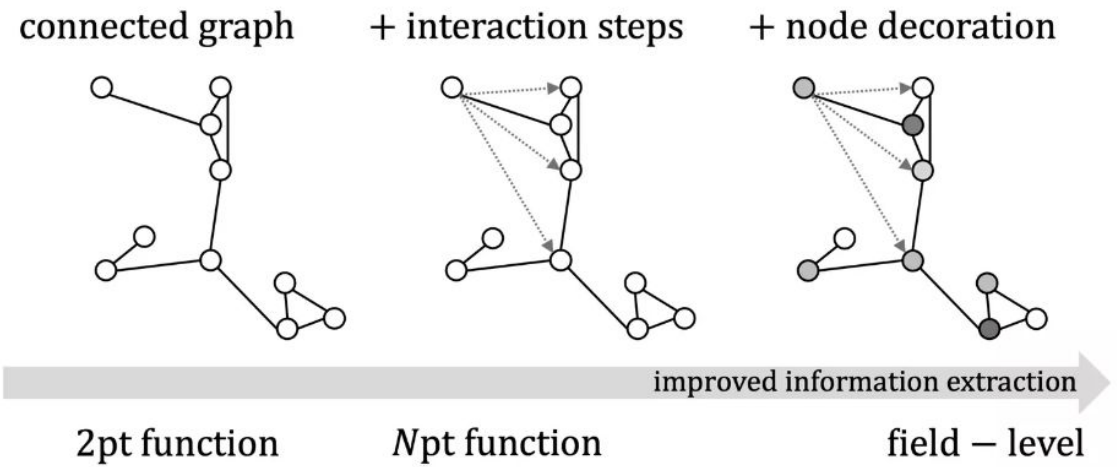
When applied to a Gaussian Random Field, it was able to show the same posterior as to the (full) analytical likelihood! The IMNN procedure almost perfectly conserved the information!

The crucial step of this procedure is given by the choice of the neural network employed as encoder!



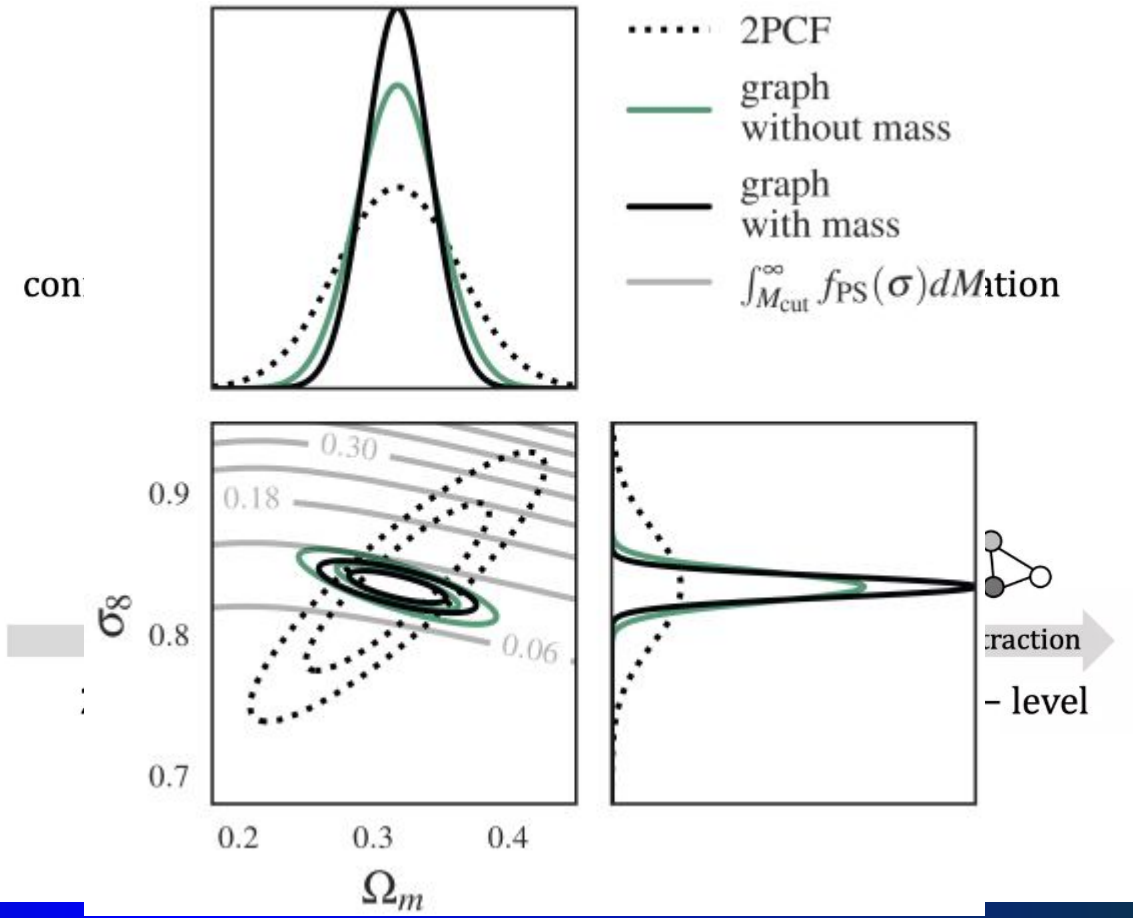
A popular choice when dealing with unstructured data (like eg a galaxy survey) is represented by Graph Neural Network.

Nicely deals with unstructured data, deals with additional information (such as the mass of the particles considered).



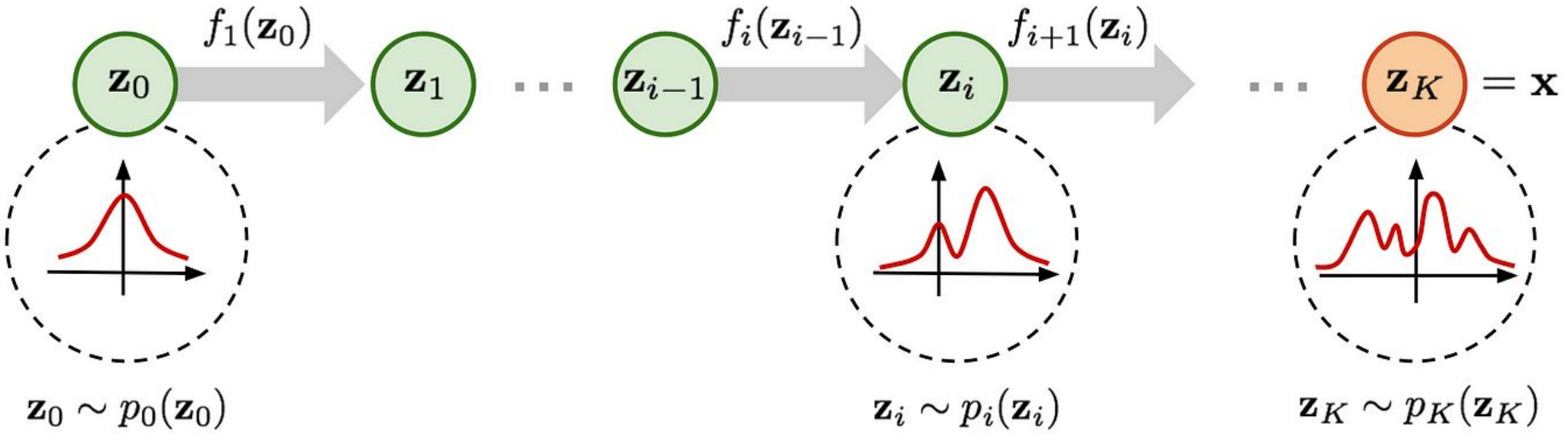
A popular choice when dealing with unstructured data (like eg a galaxy survey) is represented by Graph Neural Network.

Nicely deals with unstructured data, deals with additional information (such as the mass of the particles considered).

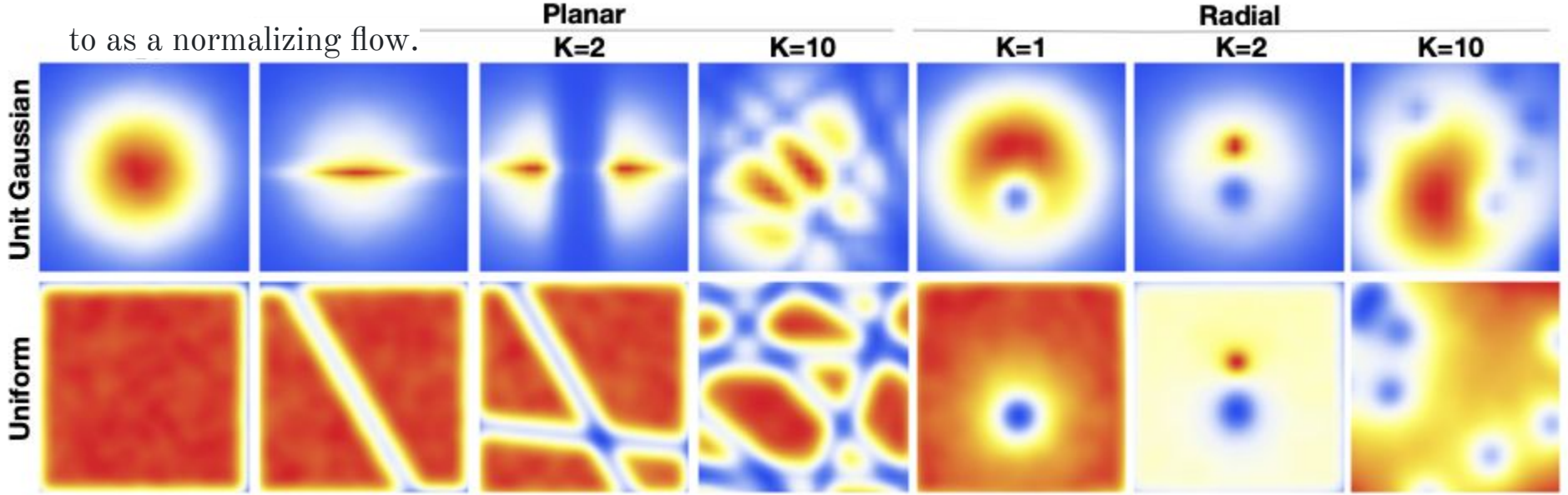


Normalizing Flows

Another popular class of object used to encode the information of a generative process is represented by Normalizing Flows



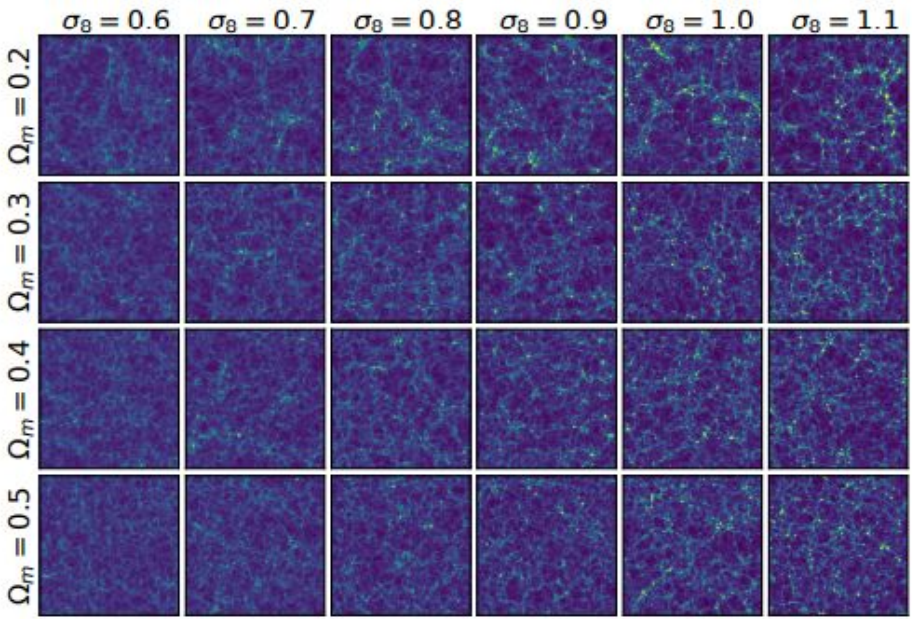
Normalizing Flows are a method for constructing complex distributions by transforming a probability density through a series of invertible mappings. By repeatedly applying the rule for change of variables, the initial density ‘flows’ through the sequence of invertible mappings. At the end of this sequence we obtain a valid probability distribution and hence this type of flow is referred to as a normalizing flow.



So far so good! Where is the catch?

- The more complicated the NF employed, the higher its expressiveness
- The more complicated the NF employed, the more complicated it is to train it!

Dai&Seljak 2023 proposed Translation and Rotation Equivariant Normalizing Flow (TRENF), a powerful class of NF that is cheap to use thanks to the symmetries of the problem.



Once trained, the Normalizing Flow can be used both to create random realizations (generative direction) and to conditionally sample the likelihood (normalizing direction)