Debriefing on
LIKELIHOOD-FREE INFERENCE IN COSMOLOGY

Likelihood-Free Inference Workshop

18-22 March 2019, @Flatiron Institute, NYC

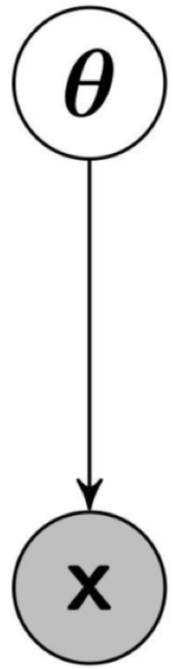
Bayesian inference

Observe data $\mathbf{x}_o$

Infer parameters θ

$$p(\theta \,|\, \mathbf{x} = \mathbf{x}_o) \propto p(\mathbf{x} = \mathbf{x}_o \,|\, \theta)\, p(\theta)$$

posterior likelihood prior



Slide credits:

George Papamakarios

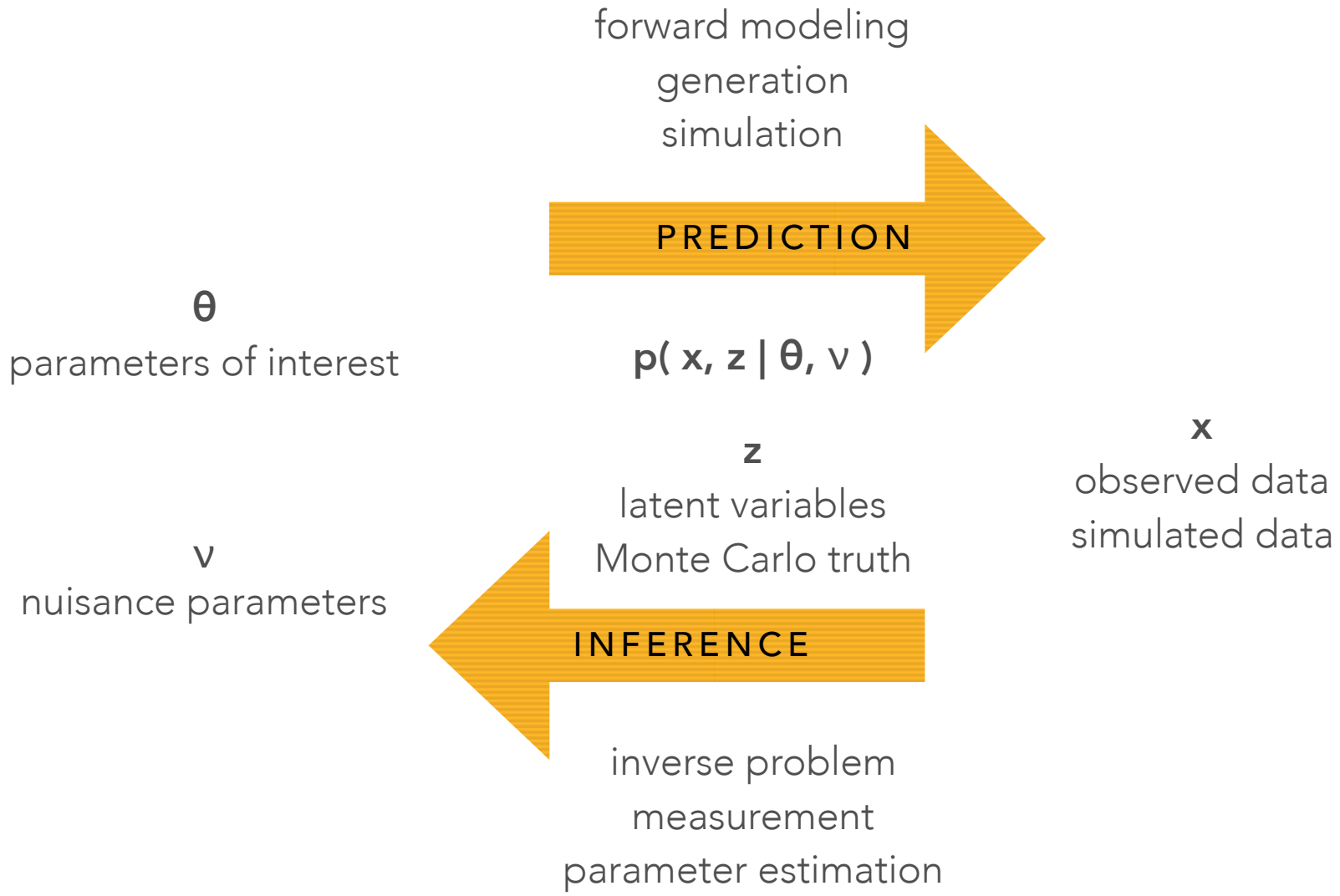
THE UNIVERSITY of EDINBURGH

Motivations

Likelihood not available analytically

- Physical model too complex or unknown
- Theory not fully understood
- Strong dependency in data
- Observational limitations

Statistical Framing



Slide credits:

@KyleCranmer
New York University

Likelihood-free inference

Observe data $\mathbf{x}_o$

Infer parameters θ

Likelihood $p(\mathbf{x} = \mathbf{x}_o \mid \theta)$ is not available

Can simulate $\mathbf{x}_n \sim p(\mathbf{x} \mid \theta)$ for any θ

Slide credits:

George Papamakarios

THE UNIVERSITY of EDINBURGH

Approximate Bayesian Computation

Simulate $\theta_n \sim p(\theta)$

$\mathbf{x}_n \sim p(\mathbf{x} \mid \theta_n)$

If $\mathbf{x}_n = \mathbf{x}_o$ then θ_n is a sample from $p(\theta \mid \mathbf{x} = \mathbf{x}_o)$

In practice:

- Reduce data to summary statistics
- Accept whenever $\|\mathbf{x}_n - \mathbf{x}_o\| < \epsilon$
- Propose new parameters by 'perturbing' accepted parameters
  - MCMC-ABC, SMC-ABC, many others …

Slide credits:

George Papamakarios

THE UNIVERSITY of EDINBURGH

ABC in astronomy and cosmology

Estimating Exoplanet Occurrence Rates Using Approximate Bayesian Computation

Danley Hsu

Collaborators: Eric Ford, Darin Ragozzine, Keir Ashby, Robert Morehead

Likelihood-Free Inference Workshop – March 19, 2019

PennState

Approximate Bayesian Computation (ABC) can be applied to Kepler data

Borucki (2016)

$$\frac{N_{r,p}}{N_{\mathrm{targ}}}$$

$$\left|\left(\frac{n_{r,p}}{N_\star}\right)_i - \frac{N_{r,p}}{N_{\mathrm{targ}}}\right| > \epsilon_{\mathrm{goal}}$$

$$\left|\left(\frac{n_{r,p}}{N_\star}\right)_i - \frac{N_{r,p}}{N_{\mathrm{targ}}}\right| \leq \epsilon_{\mathrm{goal}}$$

$\theta \in [0, 1)$

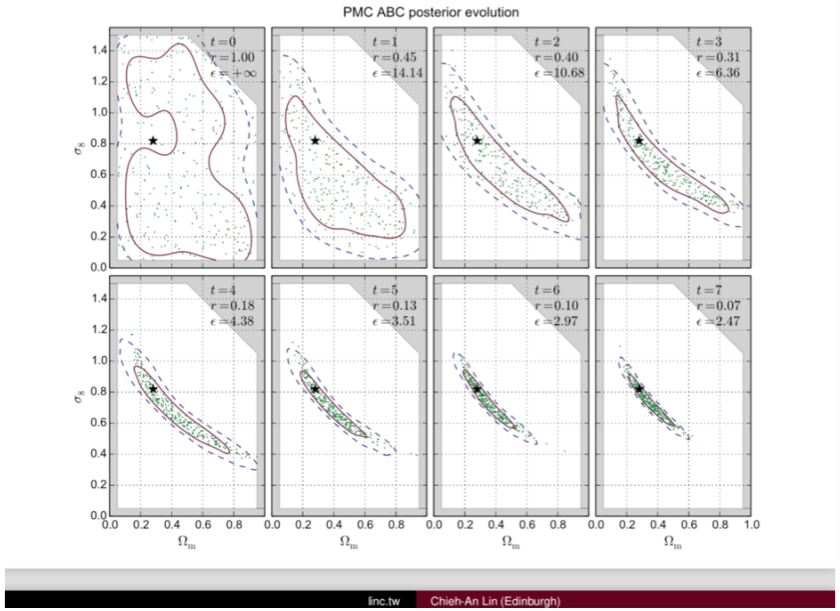
$$\left(\frac{n_{r,p}}{N_\star}\right)_1 \quad \cdots \quad \left(\frac{n_{r,p}}{N_\star}\right)_{N_{\mathrm{part}}}$$

θ

PennState

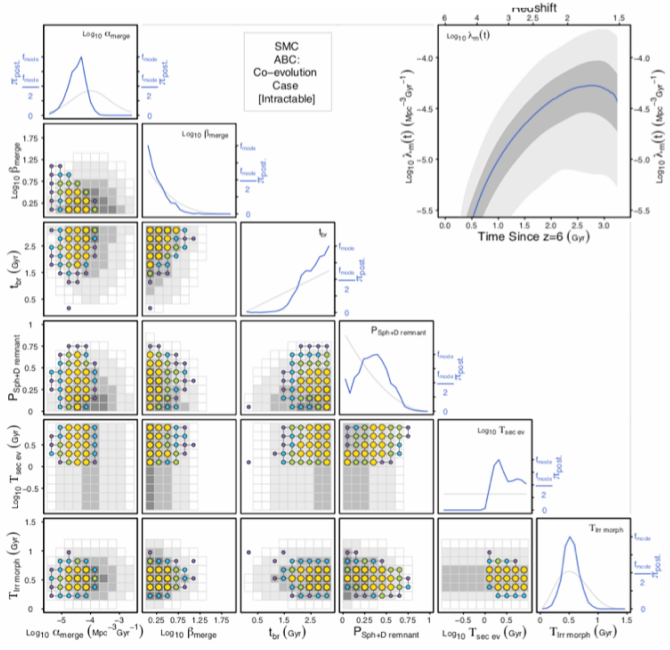
Lin & Kilbinger (2015b)

Result from ABC



Weak Lensing Peak Counts

E. Cameron and A. N. Pettitt



Galaxy Morphology

ABC applications to astronomy

- Galaxy morphology (Cameron & Pettitt 2012)
- Type Ia supernovae (Weyant et al. 2013; Jennings et al. 2016)
- Milky Way disk formation (Robin et al. 2014)
- Weak-lensing peak counts (Lin & Kilbinger 2015b)
- Strong lensing (Killedar et al. 2015)
- Halo occupation distribution (Hahn et al. 2016)
- and so on

Slide credits:

Chieh-An Lin (Linc)
University of Edinburgh

Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation

George Papamakarios
School of Informatics
University of Edinburgh
g.papamakarios@ed.ac.uk

Iain Murray
School of Informatics
University of Edinburgh
i.murray@ed.ac.uk

1

2

3

Conventional ABC algorithms such as the above suffer from three drawbacks. First, they only represent the parameter posterior as a set of (possibly weighted or correlated) samples. A sample-based representation easily gives estimates and error bars of individual parameters, and model predictions. However these computations are noisy, and it is not obvious how to perform some other computations using samples, such as combining posteriors from two separate analyses. Second, the parameter samples do not come from the correct Bayesian posterior, but from an approximation based on assuming a pseudo-observation that the data is within an ϵ-ball centred on the data actually observed. Third, as the ϵ-tolerance is reduced, it can become impractical to simulate the model enough times to match the observed data even once. When simulations are expensive to perform, good quality inference becomes impractical.

Conditional density estimation

- New parametri approach to LFI
- Bayesian conditional density estimation
- Learn a parametric approx to the **exact** posterior
- Preliminary fits to the posterior to guide future simulations

DELFI: Density-Estimation-Likelihood-Free Inference

- INFERENCE ⟶ DENSITY ESTIMATION TASK on sim. $\{\theta, d\}$
- rigorous Bayesian inference

ABSTRACT

Likelihood-free inference provides a framework for performing rigorous Bayesian inference using only forward simulations, properly accounting for all physical and observational effects that can be successfully included in the simulations. The key challenge for likelihood-free applications in cosmology, where simulation is typically expensive, is developing methods that can achieve high-fidelity posterior inference with as few simulations as possible. Density-estimation likelihood-free inference (DELFI) methods turn inference into a density estimation task on a set of simulated data-parameter pairs, and give orders of magnitude improvements over traditional Approximate Bayesian Computation approaches to likelihood-free inference. In this paper we use neural

Alsing, Charnock, Feeney, Wandelt 2019

Papamakarios 2016

DELFI: Density-Estimation-Likelihood-Free Inference

- INFERENCE $\longrightarrow$ DENSITY ESTIMATION TASK on $\{\theta, d\}_{sim}$
- **3 ways:**

$$1) \quad p(\boldsymbol{\theta}, \boldsymbol{t}) \longrightarrow p(\boldsymbol{\theta}|\boldsymbol{t}) \propto p(\boldsymbol{\theta}, \boldsymbol{t} = \boldsymbol{t_0})$$

$$2) \quad p(\boldsymbol{\theta}|\boldsymbol{t}) \longrightarrow p(\boldsymbol{\theta}|\boldsymbol{t} = \boldsymbol{t_0})$$

$$3) \quad p(\boldsymbol{t}|\boldsymbol{\theta}) \longrightarrow p(\boldsymbol{\theta}|\boldsymbol{t}) \propto p(\boldsymbol{\theta})p(\boldsymbol{t}|\boldsymbol{\theta})$$

Learning the likelihood function

- complete freedom as to how the simulations are acquired
- no need to re-weight the target density
- to explore different priors *a posteriori* (without relying on similar importance re-weighting issues)
- small number summaries will often tend to be asymptotically Gaussian
- Gaussian mixture with a modest number of component

DELFI: Density-Estimation-Likelihood-Free Inference

- INFERENCE $\longrightarrow$ DENSITY ESTIMATION TASK on $\{\theta, d\}_{sim}$
- **3 ways:**

$$1) \quad p(\boldsymbol{\theta}, \boldsymbol{t}) \longrightarrow p(\boldsymbol{\theta}|\boldsymbol{t}) \propto p(\boldsymbol{\theta}, \boldsymbol{t} = \boldsymbol{t_0})$$

$$2) \quad p(\boldsymbol{\theta}|\boldsymbol{t}) \longrightarrow p(\boldsymbol{\theta}, \boldsymbol{t} = \boldsymbol{t_0})$$

$$3) \quad p(\boldsymbol{t}|\boldsymbol{\theta}) \longrightarrow p(\boldsymbol{\theta}|\boldsymbol{t}) \propto p(\boldsymbol{\theta})p(\boldsymbol{t}|\boldsymbol{\theta})$$

DELFI: Density-Estimation-Likelihood-Free Inference

- Run simulations to obtain $\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Fit a parametric conditional density estimator $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ to the sim.$\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Evaluate $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ at $\boldsymbol{t_0}$ to get the learned likelihood

---

1) How to parametrize $p(t|\theta; w)$ in a sensible way

2) How to run sim in the most relevant part of parameter space

3) How to compress the data to informative summaries: $d \rightarrow t$

DELFI: Density-Estimation-Likelihood-Free Inference

- Run simulations to obtain $\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Fit a parametric conditional density estimator $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ to the sim.$\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Evaluate $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ at $\boldsymbol{t_0}$ to get the learned likelihood

---

1) How to parametrize $p(t|\theta; w)$ in a sensible way

2) How to run sim in the most relevant part of parameter space

3) How to compress the data to informative summaries: $d \rightarrow t$

Parametrize $p(t|\theta; w)$ **: NEURAL DENSITY ESTIMATORS**

- Neural networks with weights $\boldsymbol{w}$
- Trained on $\{\boldsymbol{\theta}, \boldsymbol{t}\}$

- **Mixture Density Networks (MDN)**

- **Masked Autoregressive flows (MAF)**

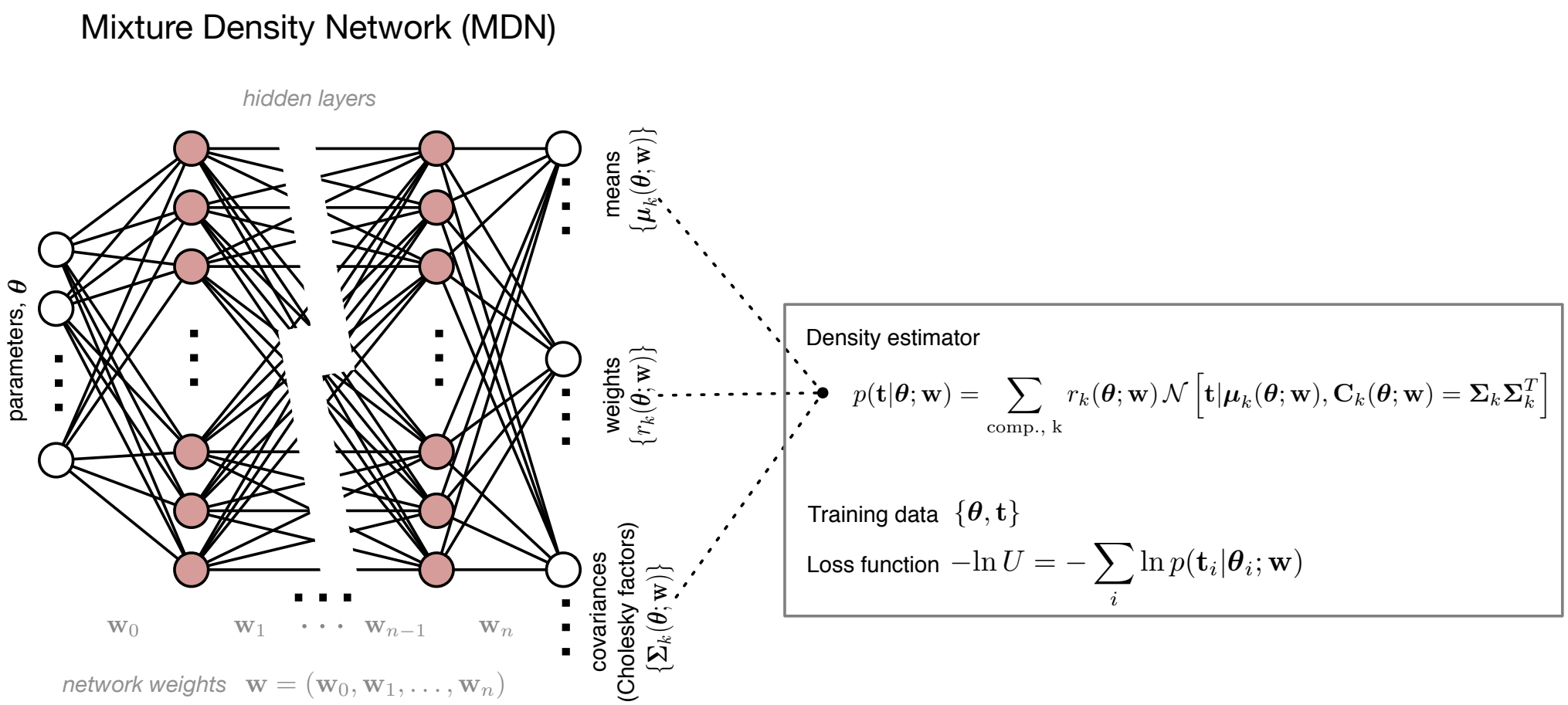


Figure 2. Schematic of the mixture density network parameterization of the conditional density $p(\mathbf{t}|\boldsymbol{\theta})$. The means, weights and covariances of a Gaussian mixture model for $p(\mathbf{t}|\boldsymbol{\theta})$ are free functions of the parameters $\boldsymbol{\theta}$, parameterized by the weights, $\mathbf{w}$, of the neural network. The neural network takes $\boldsymbol{\theta}$ as input and outputs the parameters of the mixture model for those parameters.

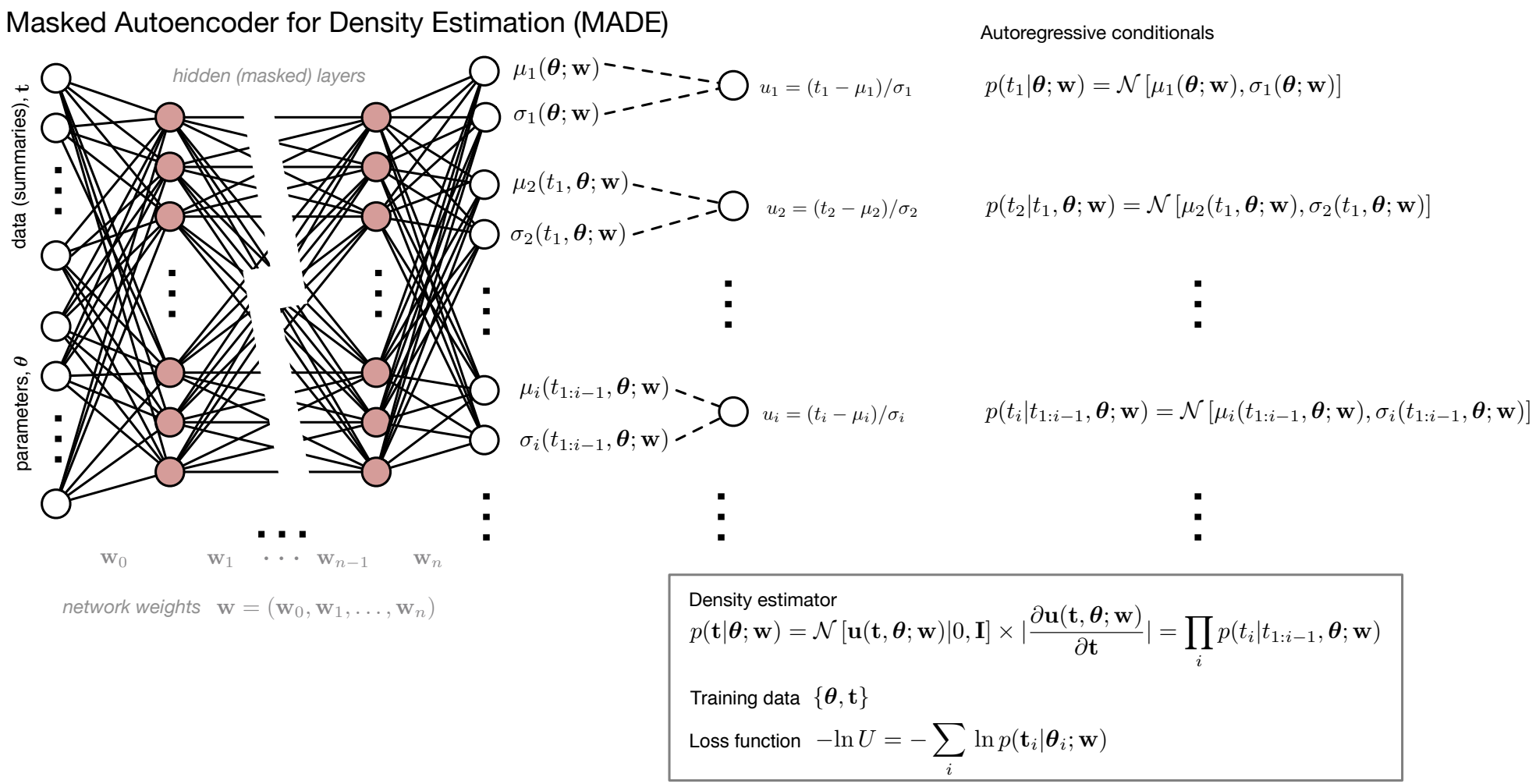


Figure 3. Schematic of the conditional masked autoencoder for density estimation (MADE) parameterization of the conditional density $p(\mathbf{t}|\boldsymbol{\theta})$. The means and variances of the autoregressive conditionals are parameterized by the neural network, with the hidden layers carefully masked to ensure the autoregressive properties are satisfied. A masked autoregressive flow (MAF) is a stack of MADEs, where the output of each MADE is fed as input to the next, and the order of the autoregressive factorization is changed between MADEs.

TRAINING NDEs

- $\boldsymbol{w}$ *s.t. minimize*

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta}, \mathbf{t}\}) = -\sum_{i=1}^{N_{\text{samples}}} \ln p(\mathbf{t}_i|\boldsymbol{\theta}_i; \mathbf{w}),$$

MDN

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta}, \mathbf{t}\}) = -\sum_i \sum_{k=1}^{n_c} \pi_k(\boldsymbol{\theta}_i; \mathbf{w})\, \mathcal{N}\left[\mathbf{t}_i \,|\, \boldsymbol{\mu}_k(\boldsymbol{\theta}_i; \mathbf{w}), \boldsymbol{\Sigma}_k(\boldsymbol{\theta}_i; \mathbf{w})\right]$$

MAF

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta}, \mathbf{t}\}) = -\sum_i \ln\Big[\mathcal{N}\big(\mathbf{u}(\mathbf{t}_i, \boldsymbol{\theta}_i; \mathbf{w}) \,|\, \mathbf{0}, \mathbf{I}\big) + \sum_{n=1}^{N_{\text{mades}}} \sum_{m=1}^{\dim(\mathbf{t})} \ln \sigma_m^n(\mathbf{t}_i, \boldsymbol{\theta}_i; \mathbf{w})\Big]$$

DELFI: Density-Estimation-Likelihood-Free Inference

- Run simulations to obtain $\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Fit a parametric conditional density estimator $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ to the sim.$\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Evaluate $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ at $\boldsymbol{t_0}$ to get the learned likelihood

---

1) How to parametrize $p(t|\theta; w)$ in a sensible way

2) How to run sim in the most relevant part of parameter space

3) How to compress the data to informative summaries: $d \rightarrow t$

Adaptive acquisition of simulations with active learning

Learn the likelihood function

Goals:

-highest fidelity posterior with as few simulation as possible
-run simulation most interesting regions of parameter space

Active learning:

-NDE calls the simulator independently during training
-deciding on-the-fly the best parameters to run new sim

- **Sequential Neural Likelihood**
- **Bayesian Optimization (BO)**

Adaptive acquisition of simulations with active learning

- **Sequential Neural Likelihood (SNL)**

-simulations run in batches

-parameters for new sim drawn from a proposal density based on posterior approx.

-NDEs re-trained after each sim batch

- **Bayesian Optimization (BO)**

-next sim run at the parameters that maximise some deterministic acquisition function

-uncertainty in current learned posterior-density-surface

-define acquisition rule

DELFI: Density-Estimation-Likelihood-Free Inference

- Run simulations to obtain $\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Fit a parametric conditional density estimator $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ to the sim.$\{\boldsymbol{\theta}, \boldsymbol{t}\}$
- Evaluate $p(\boldsymbol{t}|\boldsymbol{\theta}; \boldsymbol{w})$ at $\boldsymbol{t_0}$ to get the learned likelihood

---

1) How to parametrize $p(t|\theta; w)$ in a sensible way

2) How to run sum in the most relevant part of parameter space

3) How to compress the data to informative summaries: $d \rightarrow t$

DATA COMPRESSION

see Alsing & Wandelt 2018b

- large dataset, expensive simulations
- enough simulations learn sampling distribution $p(\boldsymbol{t}|\boldsymbol{\theta};\boldsymbol{w})$

$$N \quad \text{data} \longrightarrow p \quad \text{summaries}$$

$dim(p) = \#$ parameters

-Approximate score-compression: $\mathbf{t} = \nabla\mathcal{L}_*$.

-Data compression with deep neural networks: Information Maximising Neural Network

https://github.com/tomcharnock/IMNN

PYDELFI https://github.com/justinalsing/pydelfi

- General implementation DELFI
- Neural Density Estimators (NDEs)
- $p(\boldsymbol{t}|\boldsymbol{\theta})$
- Active learning: run simulations in the most relevant region of parameter space on-the-fly
- High fidelity posteriors
- $\mathcal{O}(10^3)$ forward simulations

PYDELFI

- Specify a **simulator**
- Specify a **compressor** (if required)
- Provide the observed **data** vector
- Specify the architectures for an ensemble of **NDEs**
- **run** PYDELFI

Callable Likelihood Function

PYDELFI with prerun sim

- Suite of **simulations already available**

- Feed the simulations into PYDELFI

- NDEs trained on those simulations

README.md

pydelfi

Density Estimation Likelihood-Free Inference with neural density estimators and adaptive acquisition of simulations. The implemented methods are described in detail in Alsing, Charnock, Feeney and Wandelt 2019, and are based closely on Papamakarios, Sterratt and Murray 2018, Lueckmann et al 2018 and Alsing, Wandelt and Feeney, 2018. Please cite these papers if you use this code!

Dependencies: tensorflow, getdist, emcee, mpi4py.

Usage: Once everything is installed, try out either `cosmic_shear.ipynb` or `jla_sne.ipynb` as example templates for how to use the code; plugging in your own simulator and letting pydelfi do it's thing.

If you have a set of pre-run simulations you'd like to throw in rather than allowing pydelfi to run simulations on-the-fly, look at `cosmic_shear_prerun_sims.ipynb` as a template for how to do this.

If you are interested in using pydelfi with nuisance hardened data compression to project out nuisances (Alsing & Wandelt 2019), take a look at `jla_sne_marginalized.ipynb`.

The code is not documented yet (documentation coming imminently), but if you are interested in applying it to your problem please get in touch with us (at justin.alsing@fysik.su.se) - we welcome collaboration!

MnuLFI: MassiveNuS + PYDELFI

LFI on neutrino mass from cosmology

We have 100 cosmological models (x 10,000 map realizations each) and computed higher-order statistics (power spectrum, bispectrum, PDF, peak counts) on all of them. We would like help to appl LFI on them.

Contacts: Jia Liu
Participants: Jia Liu, Will Coulton

Goals and deliverable

Get a posterior using an (or multiple!) LFI method(s), and compare that with our (already done) Gaussian-likelihood result.

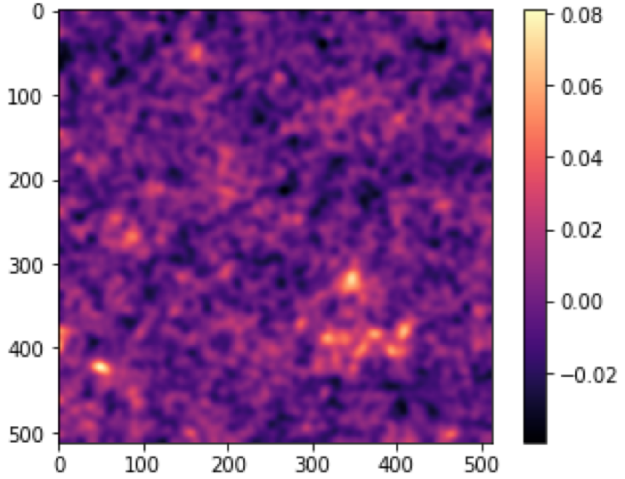
Resources needed

We will use statistics computed already from the MassiveNuS simulation:
http://columbialensing.org/#massivenus

https://github.com/ VMBoehm/MnuLFI

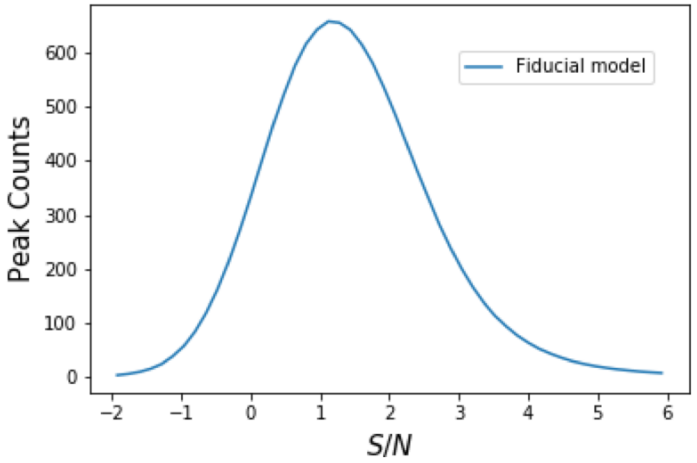
MnuLFI : MassiveNuS + PYDELFI

Lensing
Convergence map



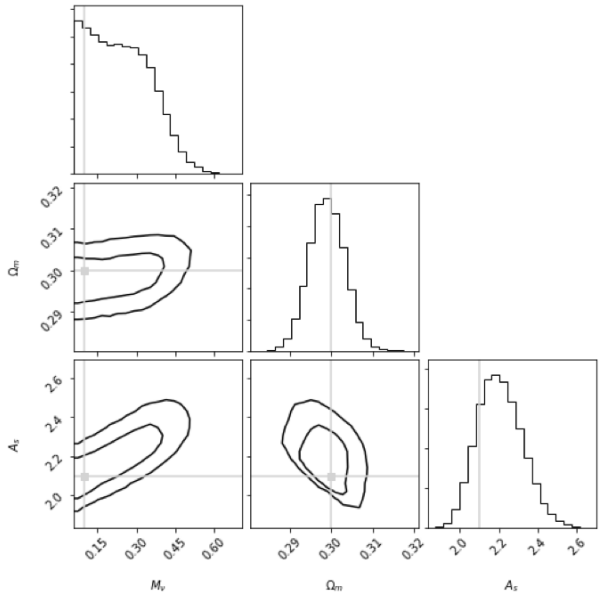
Prerun simulations MassiveNus

Peak Counts



Usual n-point statistics

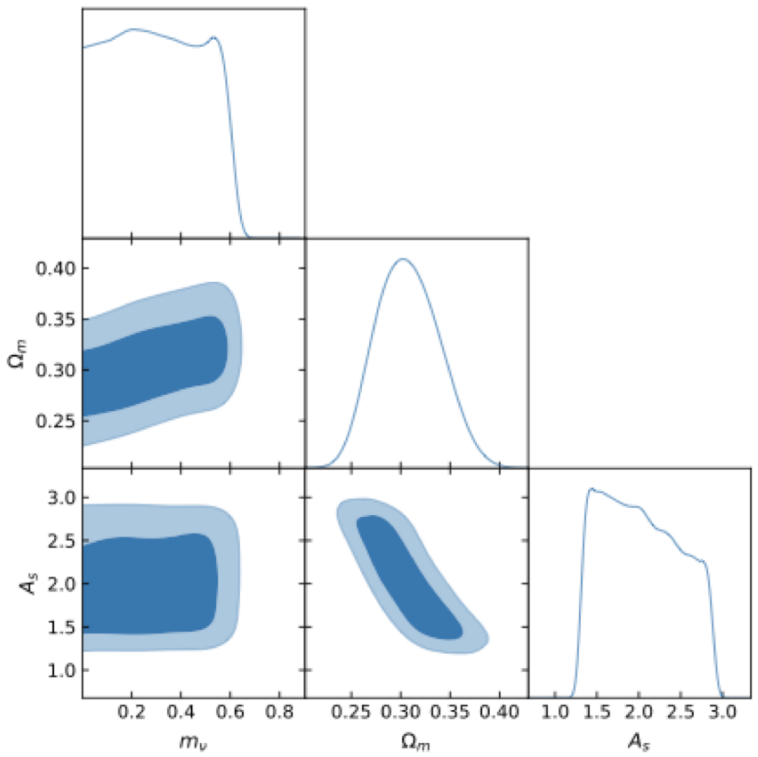
MCMC



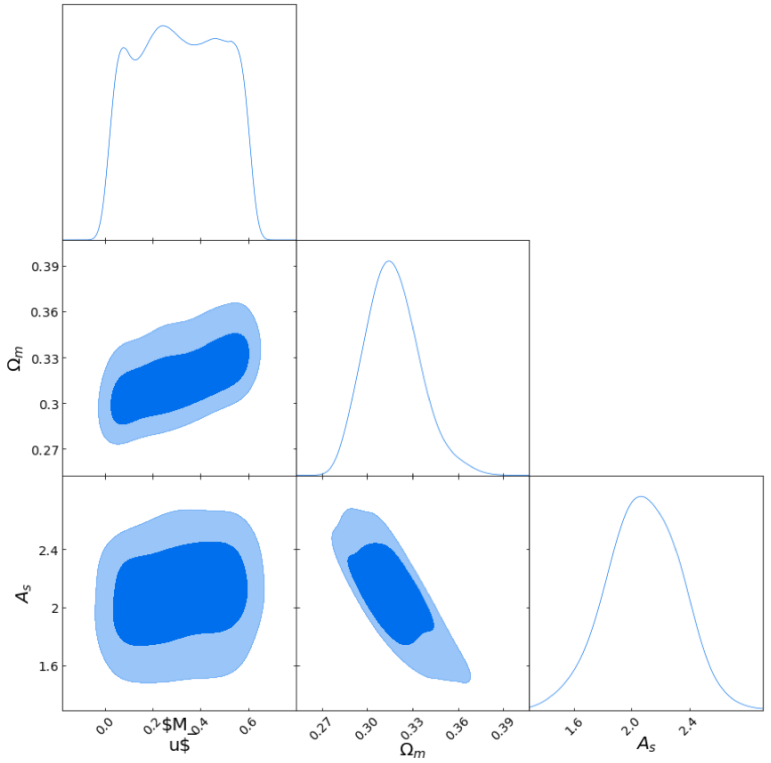
PYDELFI ?

Posterior inference

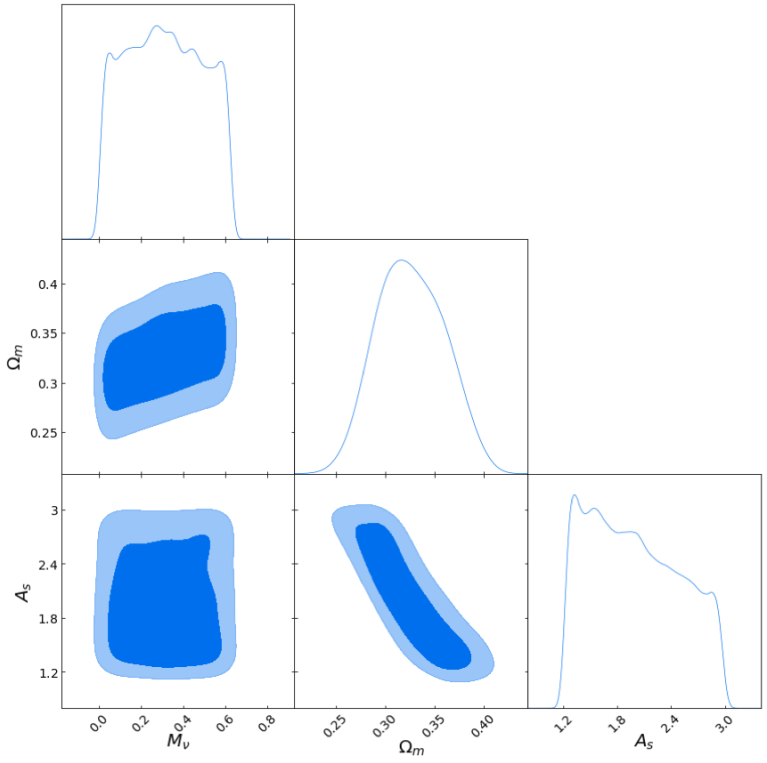
Some attempts



Score compression, no pre-training, 100 samples, no scaling



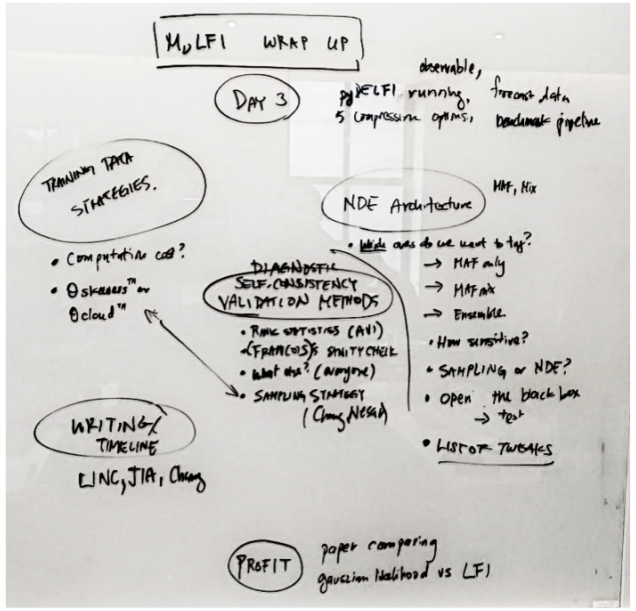
Score compression,6 NDEs, subsamples, scaled data



10 bins (combined each 5 bins), full realizations set, scaled data

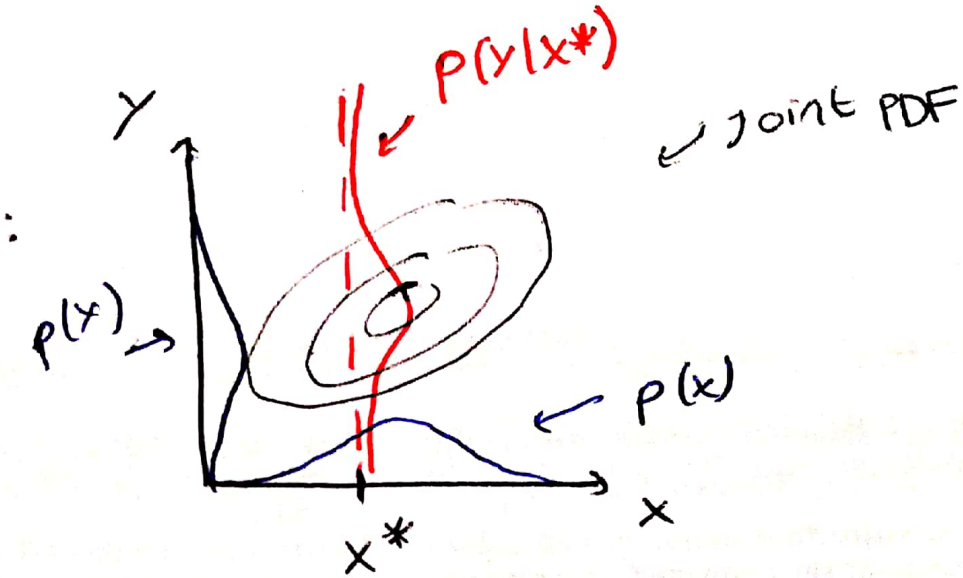
MnuLFI WRAP UP: tests to do

- **Data compression** (Full 50 bins, rebinning, score, IMNN…)
- **Training Strategy** (On the fly sims, with pre-training, w/o pre-training…)
- **NDE Architecture** (MAF only, MDN only, Ensemble…)
- **Diagnostics** (Rank statistics, François…)



Additional Slides

$x, y \longrightarrow p(x, y)$:



- $\int p(x,y)\, dx\, dy = 1$

- $p(x) = \int p(x,y)\, dy$
- $p(y) = \int p(x,y)\, dx$

$$p(y|x^*) = c\, p(x,y) \quad (1)$$

$$\int dy\, p(y|x^*) = c \int p(x,y)\, dy \quad \Rightarrow \quad c = \frac{1}{p(x)}$$

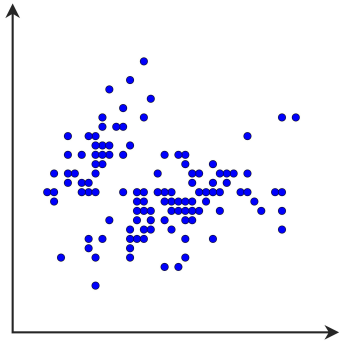
($\int dy\, p(y|x^*) = 1$, $\int p(x,y)\, dy = p(x)$)

$\Rightarrow$ (1)

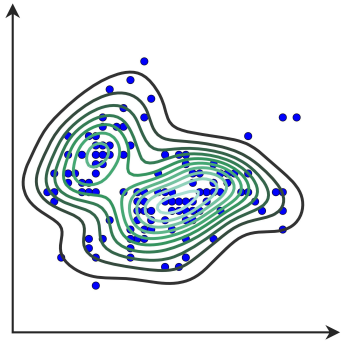
$$\boxed{p(y|x^*) = \frac{1}{p(x)}\, p(x,y) = \frac{p(x|y)\, p(y)}{p(x)}}$$

BAYES IDENTITY

Density estimation

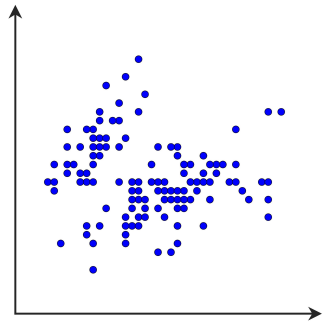


Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

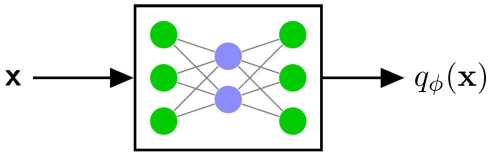


Goal: estimate density function $p(\mathbf{x})$

Neural density estimation



Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



$q_\phi(\mathbf{x})$ must be a density function:

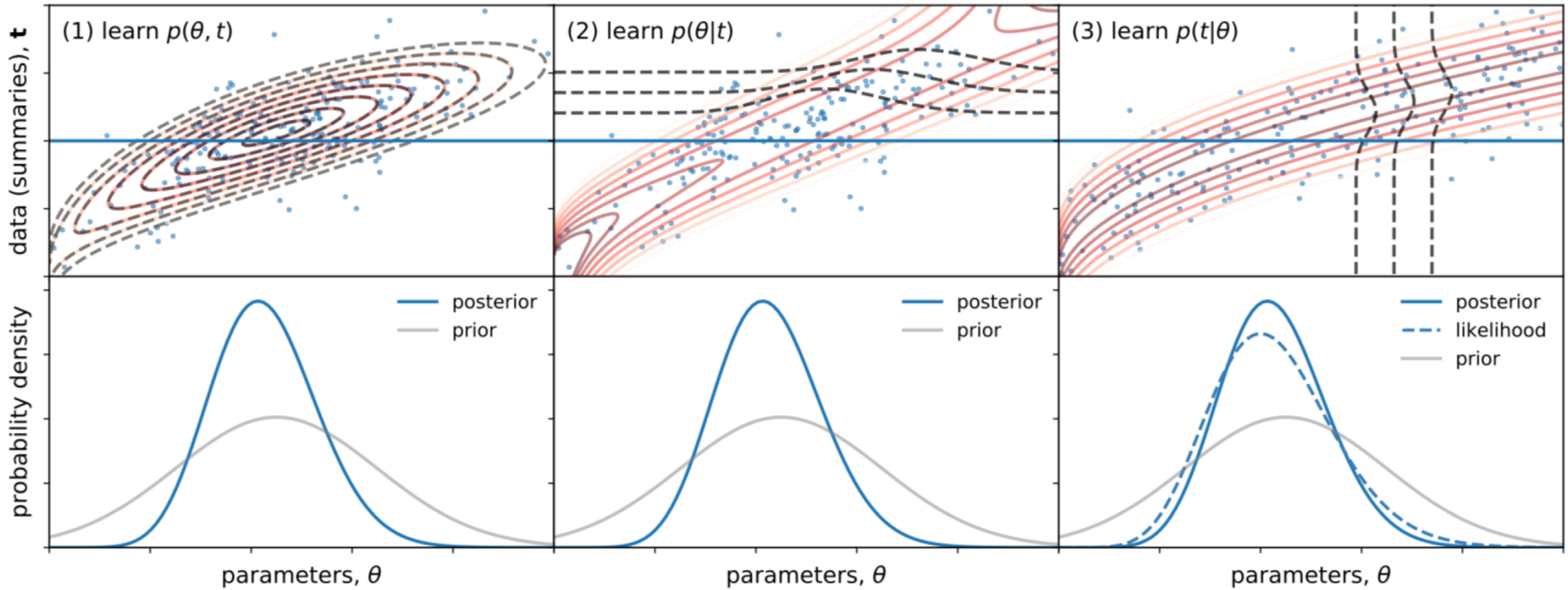
$$\int q_\phi(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1 \qquad q_\phi(\mathbf{x}) \geq 0$$

Slide credits:

George Papamakarios

THE UNIVERSITY of EDINBURGH

Fast likelihood-free cosmology with neural density estimators



Neural Likelihood

Step 1

Simulate $\theta_n \sim p(\theta) \quad \mathbf{x}_n \sim p(\mathbf{x} \mid \theta_n)$

Collect training data $\{(\theta_1, \mathbf{x}_1), \ldots, (\theta_N, \mathbf{x}_N)\}$

Step 2

Train a **conditional** neural density estimator $q_\phi(\mathbf{x} \mid \theta)$ on data

With enough data & capacity: $q_\phi(\mathbf{x} \mid \theta) \approx p(\mathbf{x} \mid \theta)$

Step 3

Sample from $\hat{p}(\theta \mid \mathbf{x} = \mathbf{x}_o) \propto q_\phi(\mathbf{x} = \mathbf{x}_o \mid \theta)\, p(\theta)$ (e.g. with MCMC)

Neural density estimation: Training

Average log likelihood:

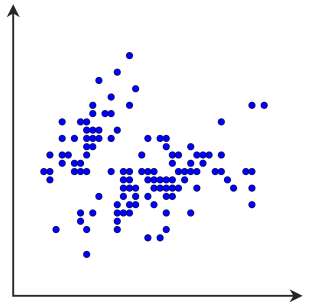
$$L(\phi) = \frac{1}{N} \sum_{n=1}^{N} \log q_\phi(\mathbf{x}_n)$$

Maximize $L(\phi)$ w.r.t. ϕ

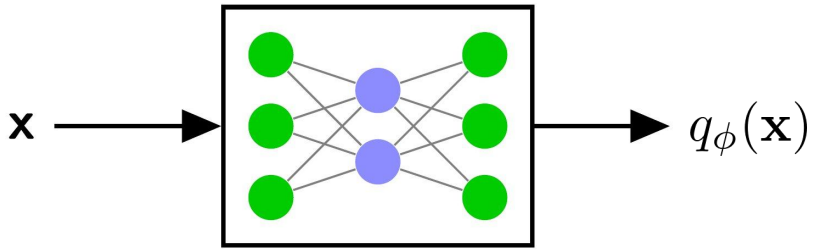
Calculate gradients with backpropagation

Use favourite type of stochastic-gradient ascent (e.g. Adam)

Use favourite machine-learning framework (e.g. TensorFlow, PyTorch)



Training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



Slide credits:

George Papamakarios

THE UNIVERSITY of EDINBURGH

Training as KL-divergence minimization

For $N \to \infty$:

$$L(\phi) = \frac{1}{N} \sum_{n=1}^{N} \log q_\phi(\mathbf{x}_n) \to \mathbb{E}_{\mathbf{x}\sim p}(\log q_\phi(\mathbf{x}))$$

Maximizing average log likelihood is asymptotically equivalent to minimizing KL divergence from the true data density, since:

$$D_{\mathrm{KL}}(p \,\|\, q_\phi) = -\mathbb{E}_{\mathbf{x}\sim p}(\log q_\phi(\mathbf{x})) + \mathrm{const}$$

Slide credits:

George Papamakarios

THE UNIVERSITY of EDINBURGH

Two main ideas for likelihood-free inference

Neural density estimation:

- Estimate likelihood / posterior from simulated data
- Use state-of-the-art neural density estimators

Sequential inference:

- Guide simulations by using so-far posterior as proposal
- Can lead to orders-of-magnitude savings in simulation cost

ABC methods-limitations

- # sim needed scales exponentially with the number of model
- Critically slow down for epsilon ->0
- Draw samples from a broader distribution than the posterior

- parameter posterior as a set of samples
- noisy computations
- not obvious how to perform some other computations using samples

SNL
Sequential Neural Likelihood

-simulations in batches

-parameters from proposal based on posterior approx.

-NDEs re-trained after each new sim. batch

BO
Bayesian Optimization

-parameters maximize acquisition function

-quantify uncertainty in the learned posterior

-acquisition rule

What happens inside the Delfi Object?

3.2. pydelfi.delfi module

class pydelfi.delfi.**Delfi**(*data*, *prior*, *nde*, *Finv=None*, *theta_fiducial=None*, *param_limits=None*, *param_names=None*, *nwalkers=100*, *posterior_chain_length=1000*, *proposal_chain_length=100*, *rank=0*, *n_procs=1*, *comm=None*, *red_op=None*, *show_plot=True*, *results_dir=''*, *progress_bar=True*, *input_normalization=None*, *graph_restore_filename='graph_checkpoint'*, *restore_filename='restore.pkl'*, *restore=False*, *save=True*) [source]

Bases: **object**

acquisition(*theta*) [source]

add_simulations(*xs_batch*, *ps_batch*) [source]

allocate_jobs(*n_jobs*) [source]

bayesian_optimization_training(*simulator*, *compressor*, *n_batch*, *n_populations*, *n_optimizations=10*, *simulator_args=None*, *compressor_args=None*, *plot=False*, *batch_size=100*, *validation_split=0.1*, *epochs=300*, *patience=20*, *seed_generator=None*, *save_intermediate_posteriors=False*, *sub_batch=1*) [source]

complete_array(*target_distrib*) [source]

emcee_sample(*log_likelihood=None*, *x0=None*, *burn_in_chain=100*, *main_chain=1000*) [source]

fisher_pretraining(*n_batch=5000*, *plot=True*, *batch_size=100*, *validation_split=0.1*, *epochs=300*, *patience=20*) [source]

load_simulations(*xs_batch*, *ps_batch*) [source]

log_geometric_mean_proposal_individual(*i*, *x*) [source]

log_geometric_mean_proposal_stacked(*x*) [source]

log_likelihood_individual(*i*, *theta*) [source]

log_likelihood_stacked(*theta*) [source]

log_posterior_individual(*i*, *x*) [source]

log_posterior_stacked(*x*) [source]

run_simulation_batch(*n_batch*, *ps*, *simulator*, *compressor*, *simulator_args*, *compressor_args*, *seed_generator=None*, *sub_batch=1*) [source]

saver() [source]

sequential_training(*simulator*, *compressor*, *n_initial*, *n_batch*, *n_populations*, *proposal=None*, *simulator_args=None*, *compressor_args=None*, *safety=5*, *plot=True*, *batch_size=100*, *validation_split=0.1*, *epochs=300*, *patience=20*, *seed_generator=None*, *save_intermediate_posteriors=True*, *sub_batch=1*) [source]

sequential_training_plot(*savefig=False*, *filename=None*) [source]

train_ndes(*training_data=None*, *batch_size=100*, *validation_split=0.1*, *epochs=500*, *patience=20*, *mode='samples'*) [source]

triangle_plot(*samples=None*, *savefig=False*, *filename=None*) [source]

Generalized massive optimal data compression

Justin Alsing[1,2]★ and Benjamin Wandelt[1,3,4,5]

2.2 Sufficient statistics of linearized likelihoods and compression to the score function

Taylor expanding the log-likelihood to second order in the parameters about some fiducial point $\boldsymbol{\theta}_*$ we have,

$$\mathcal{L} = \mathcal{L}_* + \delta\boldsymbol{\theta}^{\mathrm{T}}\nabla\mathcal{L}_* - \frac{1}{2}\delta\boldsymbol{\theta}^{\mathrm{T}}\mathbf{J}_*\delta\boldsymbol{\theta}, \tag{4}$$

where the derivative of the log-likelihood is commonly referred to as the *score function* $\mathbf{s} \equiv \nabla\mathcal{L}$, the negative second derivative is the *observed information matrix* $\mathbf{J} \equiv -\nabla\nabla^{\mathrm{T}}\mathcal{L}$,

To linear order in the parameters, the parameters only couple to the data through the score function $\nabla\mathcal{L}_*$ – a vector of length n and a function of the data. The score function hence constitutes the sufficient statistics for the parameters for the linearized log-likelihood function. This immediately provides a natural data compression from N data down to n compressed numbers, ie., computing only the n data combinations that appear in the score function:

$$\mathbf{t} = \nabla\mathcal{L}_*. \tag{5}$$

- **Density Estimation LFI**
- **"Mining gold" from implicit models to improve likelihood-free inference**
- **BOLFI and LIFRE Machine Learning**
- **Machine Learning Accelerated Likelihood-Free Event Reconstruction in Dark Matter Direct Detection**
- **IMNN for dimensionality reduction**
- **Compromise-free Bayesian sparse reconstruction**
- **Validation of Emulators and Approximate Likelihood Models**
- **Estimating Exoplanet Occurrence Rates Using Approximate Bayesian Computation**
- **Suite of Gaussian Process emulators for cosmological inference problems**
- **Simulate a universe on your laptop**
- **ELFI - Engine for Likelihood-free Inference**

CASE STUDY: TOMOGRAPHIC COSMIC SHEAR

Import stuff

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pydelfi.priors as priors
import pydelfi.ndes as ndes
import pydelfi.delfi as delfi
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
%matplotlib inline
```

Set up the prior

In [2]:
```
lower = np.array([0, 0.4, 0, 0.4, 0.7])
upper = np.array([1, 1.2, 0.1, 1.0, 1.3])
prior = priors.Uniform(lower, upper)
```

Load in the compressed data vector, sims, Fisher matrix, fiducial params

`compressed_data` is an array of length `n_summaries`
`sim_params` is an array with shape `(n_simulations, n_parameters)`
`sim_compressed_data` is an array with shape `(n_simulations, n_summaries)`

In [3]:
```
compressed_data = np.genfromtxt('simulators/cosmic_shear/pre_ran_sims/compressed_data.dat')

sim_params = np.genfromtxt('simulators/cosmic_shear/pre_ran_sims/simulations_parameters.dat')

sim_compressed_data = np.genfromtxt('simulators/cosmic_shear/pre_ran_sims/simulations_compressed_data.dat')

Finv = np.genfromtxt('simulators/cosmic_shear/pre_ran_sims/Finv.dat')

theta_fiducial = np.array([0.3, 0.8, 0.05, 0.70, 0.96])
```

CASE STUDY: TOMOGRAPHIC COSMIC SHEAR

Create ensemble of NDEs

In [4]:
```
NDEs = [ndes.ConditionalMaskedAutoregressiveFlow(n_parameters=5, n_data=5, n_hiddens=[50,50], n_ma
des=5, act_fun=tf.tanh, index=0),
        ndes.MixtureDensityNetwork(n_parameters=5, n_data=5, n_components=1, n_hidden=[30,30], act
ivations=[tf.tanh, tf.tanh], index=1),
        ndes.MixtureDensityNetwork(n_parameters=5, n_data=5, n_components=2, n_hidden=[30,30], act
ivations=[tf.tanh, tf.tanh], index=2),
        ndes.MixtureDensityNetwork(n_parameters=5, n_data=5, n_components=3, n_hidden=[30,30], act
ivations=[tf.tanh, tf.tanh], index=3),
        ndes.MixtureDensityNetwork(n_parameters=5, n_data=5, n_components=4, n_hidden=[30,30], act
ivations=[tf.tanh, tf.tanh], index=4),
        ndes.MixtureDensityNetwork(n_parameters=5, n_data=5, n_components=5, n_hidden=[30,30], act
ivations=[tf.tanh, tf.tanh], index=5)]
```

Create DELFI object

In [5]:
```
DelfiEnsemble = delfi.Delfi(compressed_data, prior, NDEs,
                            Finv = Finv,
                            theta_fiducial = theta_fiducial,
                            param_limits = [lower, upper],
                            param_names = ['\Omega_m', 'S_8', '\Omega_b', 'h', 'n_s'],
                            results_dir = "simulators/cosmic_shear/results_prerun/",
                            input_normalization="fisher")
```

Load simulations into DELFI object

In [6]:
```
DelfiEnsemble.load_simulations(sim_compressed_data, sim_params)
```

Fisher pre-training to initialize networks

In [7]:
```
DelfiEnsemble.fisher_pretraining(n_batch=5000, batch_size=100)
```

CASE STUDY: TOMOGRAPHIC COSMIC SHEAR

Train the networks

In [8]:
```
DelfiEnsemble.train_ndes()
```

Sample the learned posterior

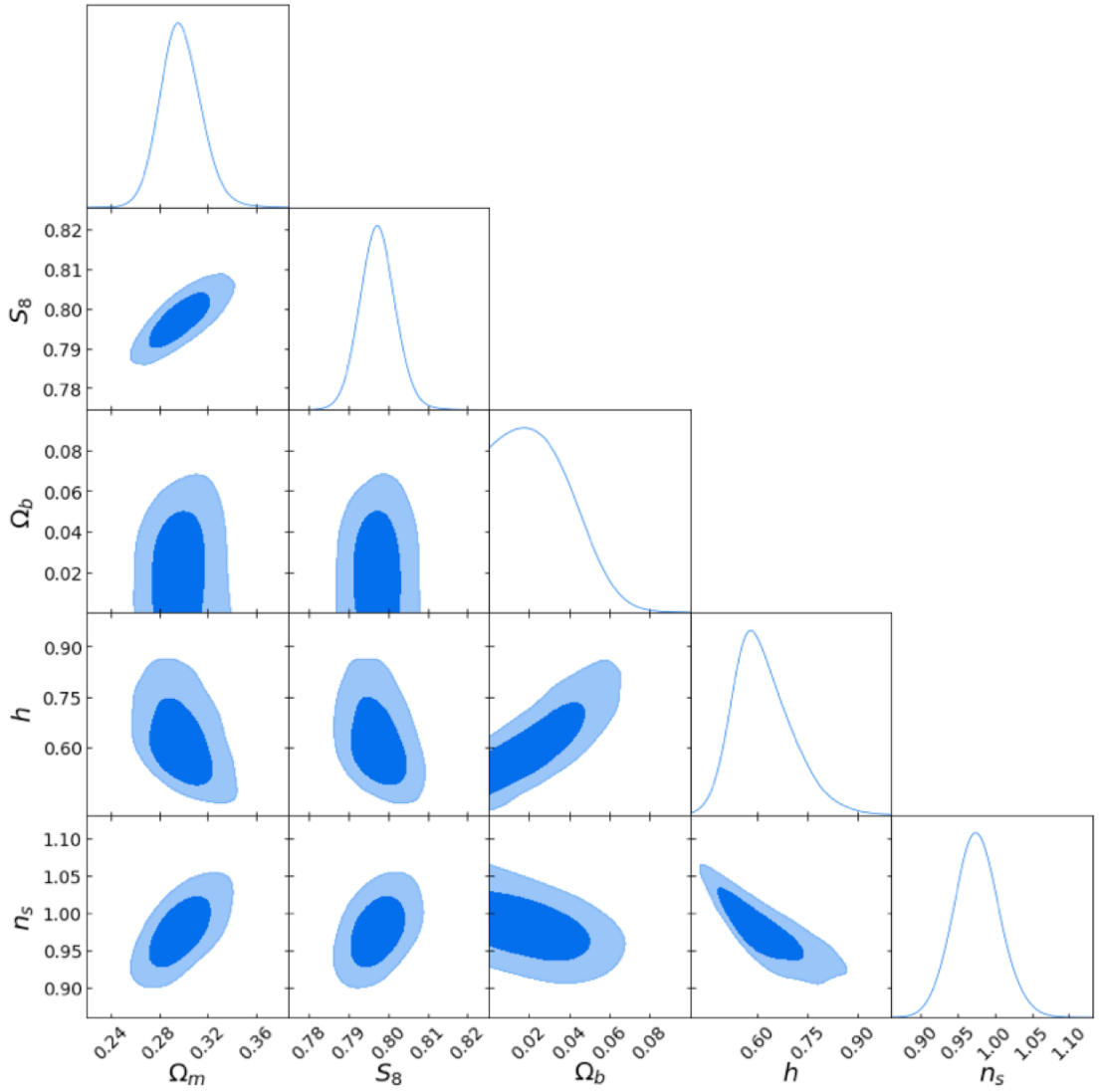
In [9]:
```
posterior_samples = DelfiEnsemble.emcee_sample()
```

Alright let's plot it!

Feed it a list of `(n_samples, n_parameters)` arrays for making a triangle plot; in this case let's just plot the posterior samples.

In [10]:
```
DelfiEnsemble.triangle_plot(samples=[posterior_samples])
```

CASE STUDY: TOMOGRAPHIC COSMIC SHEAR



TRAINING NDEs

- $\boldsymbol{w}$ *s.t. minimize* $D_{\mathrm{KL}}(p^* \mid p) = \int p^*(\mathbf{t}|\boldsymbol{\theta}) \ln\left(\frac{p(\mathbf{t}|\boldsymbol{\theta};\mathbf{w})}{p^*(\mathbf{t}|\boldsymbol{\theta})}\right) d\mathbf{t}$

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta},\mathbf{t}\}) = -\sum_{i=1}^{N_{\mathrm{samples}}} \ln p(\mathbf{t}_i|\boldsymbol{\theta}_i;\mathbf{w}),$$

MDN

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta},\mathbf{t}\}) = -\sum_i \sum_{k=1}^{n_c} \pi_k(\boldsymbol{\theta}_i;\mathbf{w})\, \mathcal{N}\left[\mathbf{t}_i \mid \boldsymbol{\mu}_k(\boldsymbol{\theta}_i;\mathbf{w}), \boldsymbol{\Sigma}_k(\boldsymbol{\theta}_i;\mathbf{w})\right]$$

MAF

$$-\ln U(\mathbf{w}|\{\boldsymbol{\theta},\mathbf{t}\}) = -\sum_i \ln\Big[\mathcal{N}\big(\mathbf{u}(\mathbf{t}_i,\boldsymbol{\theta}_i;\mathbf{w}) \mid \mathbf{0},\mathbf{I}\big) + \sum_{n=1}^{N_{\mathrm{mades}}} \sum_{m=1}^{\dim(\mathbf{t})} \ln \sigma_m^n(\mathbf{t}_i,\boldsymbol{\theta}_i;\mathbf{w})\Big]$$

Ann B. Lee
Department of Statistics & Data Science
Carnegie Mellon University

Validation of Emulators: Open Problems

- Typically, validation is done via ML loss functions or distance measures, or goodness-of-fit tests based on posterior quantiles or rank statistics

- To date, there are no diagnostics or validation techniques in the emulator/LFI literature that are fully consistent (that is, that can distinguish any bad estimator from the ``true'' reference likelihood), and that in addition can answer the if/where/how questions above.

$$\begin{aligned} \text{Test} \quad & H_0 : \widehat{\mathcal{L}}(\mathbf{x};\theta) = \mathcal{L}(\mathbf{x};\theta) \text{ for every } \mathbf{x} \in \mathcal{X} \text{ and } \theta \in \Theta \\ \text{versus} \quad & H_1 : \widehat{\mathcal{L}}(\mathbf{x};\theta) \neq \mathcal{L}(\mathbf{x};\theta) \text{ for some } \mathbf{x} \in \mathcal{X} \text{ and } \theta \in \Theta \end{aligned}$$