Physics 361 - Machine Learning in Physics

Lecture 25 – SBI, Uncertainty, Explicit Inference

April 23rd 2024

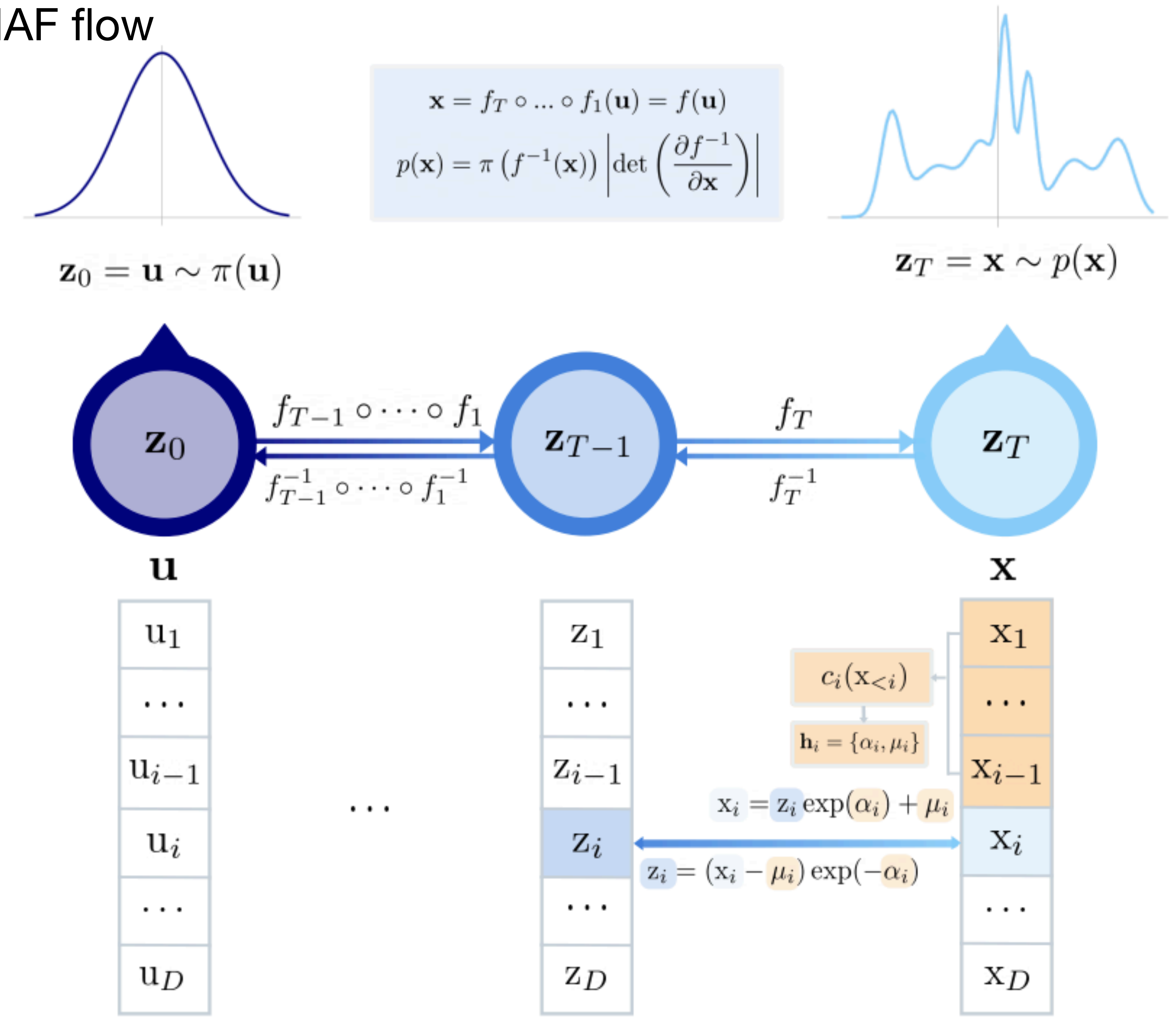
AI ∩ Universe
$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu}$
$+ i\bar{\psi} \not{D} \psi + h.c.$
$+ \bar{\psi}_i y_{ij} \psi_j \phi + h.c.$
$+ |D_\mu \phi|^2 - V(\phi)$

Moritz Münchmeyer

SBI with Normalizing Flows

Illustration of the MAF flow

Source: 2310.03741

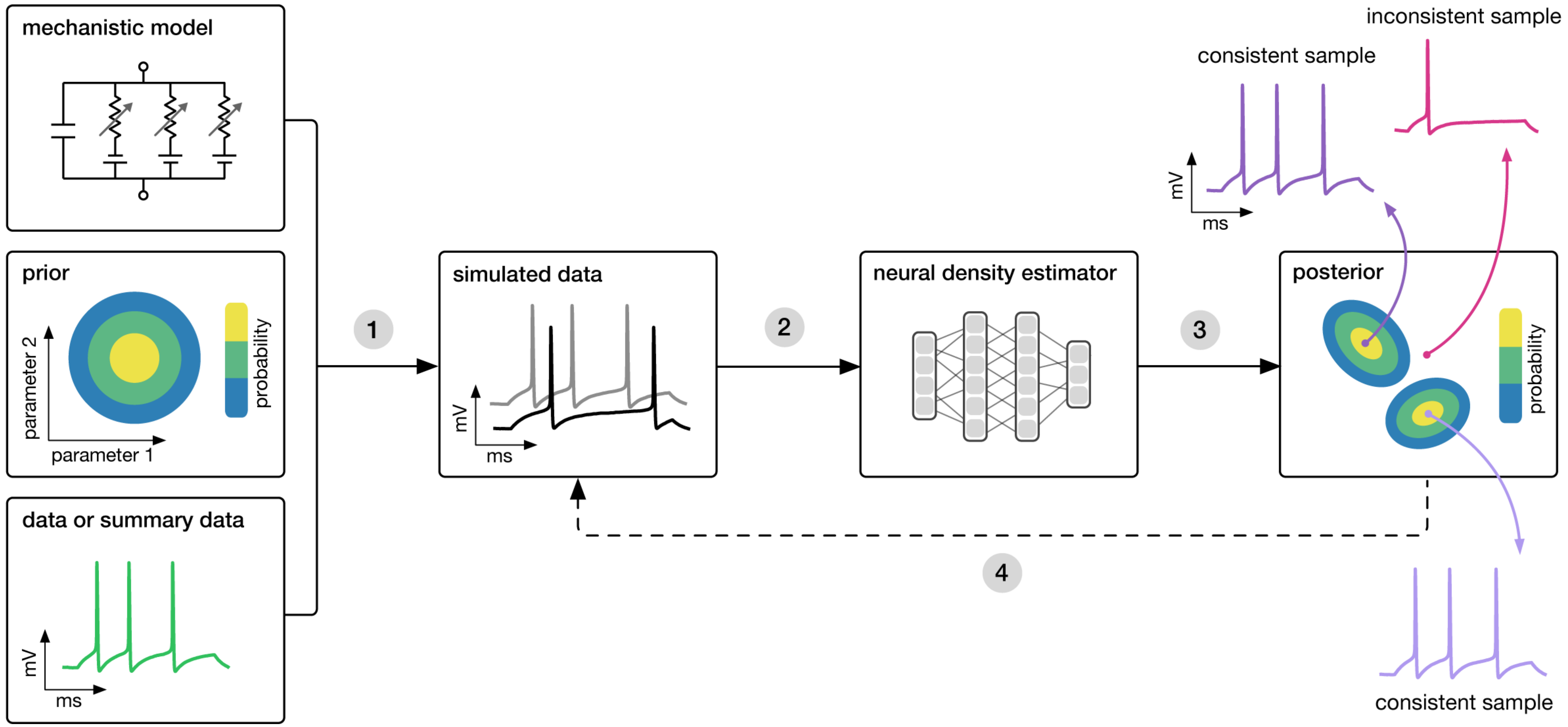


$\mathbf{x} = f_T \circ \ldots \circ f_1(\mathbf{u}) = f(\mathbf{u})$
$p(\mathbf{x}) = \pi\left(f^{-1}(\mathbf{x})\right) \left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right|$
$\mathbf{z}_0 = \mathbf{u} \sim \pi(\mathbf{u})$
$\mathbf{z}_T = \mathbf{x} \sim p(\mathbf{x})$
$\mathbf{z}_0$
$f_{T-1} \circ \cdots \circ f_1$
$f_{T-1}^{-1} \circ \cdots \circ f_1^{-1}$
$\mathbf{z}_{T-1}$
f_T
f_T^{-1}
$\mathbf{z}_T$
$\mathbf{u}$
$\mathbf{x}$
u_1, $\cdots$, u_{i-1}, u_i, $\cdots$, u_D
$\cdots$
z_1, $\cdots$, z_{i-1}, z_i, $\cdots$, z_D
$c_i(x_{<i})$
$\mathbf{h}_i = \{\alpha_i, \mu_i\}$
$x_i = z_i \exp(\alpha_i) + \mu_i$
$z_i = (x_i - \mu_i)\exp(-\alpha_i)$
x_1, $\cdots$, x_{i-1}, x_i, $\cdots$, x_D

Figure 2. Diagram of how normalizing flows work, with the specific example of Masked Autoregressive Flows. The samples from the vector $\mathbf{z}_0 = \mathbf{u}$, sampled from the simple distribution $\pi(\mathbf{u})$, are deformed through the sequence of transformations $f = f_T \circ \cdots \circ f_1$ into those of $\mathbf{z}_T = \mathbf{x}$, which follow a more complex distribution $p(\mathbf{x})$. In the lower panel, we illustrate the conditioner that "masks out" the connections between z_i and $\mathbf{h}_{<i}$, as well as the affine functions applied to the vector components.

Simulation-based inference with NDEs

inputs: A candidate mechanistic model, prior knowledge or constraints on model parameters, and observational data (or summary statistics thereof).



Goal: Algorithmically identify mechanistic models (simulators) which are consistent with data.

Source: https://sbi-dev.github.io/sbi/

Recall: Neural Likelihood Estimation (NLE)

- In neural likelihood estimation we learn the likelihood from simulated pairs of model parameters and data:

$$\{(\theta_n, x_n)\}_{n=1}^{N} \longrightarrow \mathscr{L}(x \,|\, \theta)$$

- On the next slide we will learn the required training objective for the normalizing flow.

- After training the NDE on our simulated data, we can then evaluate the likelihood of observed data from our measurement.

$$\mathscr{L}(x^{obs} \,|\, \theta)$$

- Now we can proceed with normal Bayesian data analysis. That usually means that we sample from the posterior with MCMC:

$$p(\theta \,|\, x^{obs}) \propto \mathscr{L}(x^{obs} \,|\, \theta) p(\theta)$$

Neural Likelihood estimation with NFs

- From a simulated data set $\{(\theta_n, x_n)\}_{n=1}^N$ drawn from a proposal density $\tilde{p}(\theta)$ we want to approximate the target likelihood $p(x \,|\, \theta)$ by a conditional normalizing flow $q_\phi(x \,|\, \theta)$, where $\tilde{p}(\theta, x) \overset{\textcircled{2}}{=} p(x \,|\, \theta)\tilde{p}(\theta)$.

- We thus minimize the expected KL-divergence with respect to the flow parameters ϕ.

Recall from lecture 4: $D_{\text{KL}}(P \| Q) \overset{\textcircled{1}}{=} \mathbb{E}_{X \sim P}\left[\log \frac{P(X)}{Q(X)}\right]$

$$\min_\phi \mathbb{E}_{\tilde{p}(\boldsymbol{\theta})}\left[D_{KL}\left[p(\mathbf{x} \,|\, \boldsymbol{\theta}) \,\|\, q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})\right]\right] \overset{\textcircled{1}}{=} \min_\phi \int d\boldsymbol{\theta}\tilde{p}(\boldsymbol{\theta}) \int d\mathbf{x} p(\mathbf{x} \,|\, \boldsymbol{\theta}) \log\left(\frac{p(\mathbf{x} \,|\, \boldsymbol{\theta})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})}\right)$$

$$\overset{\textcircled{2}}{=} \min_\phi \int d\boldsymbol{\theta} d\mathbf{x}\, \tilde{p}(\boldsymbol{\theta}, \mathbf{x}) \log\left(\frac{p(\mathbf{x} \,|\, \boldsymbol{\theta})}{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})}\right) \quad = \log p - \log q$$

$$= \min_\phi -\mathbb{E}_{\tilde{p}(\boldsymbol{\theta}, \mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})\right] + \text{const.}$$

eval expectation value via sampling

$$\approx \min_\phi -\frac{1}{N_{\text{sim}}} \sum_{n=1}^{N_{\text{sim}}} \log q_{\boldsymbol{\phi}}(\mathbf{x}_n \,|\, \boldsymbol{\theta}_n) + \text{const.},$$

- Minimizing the expected KL is thus the same as maximum likelihood training (see lecture 4).

Recall: Neural Posterior Estimation (NPE)

- You might wonder why why learn the likelihood and not the posterior which is our ultimate goal. Learning the posterior is indeed a possibility.
- From a simulated data set

$$\{(\theta_n, x_n)\}_{n=1}^{N}$$

drawn from a proposal density $\tilde{p}(\theta)$ it is possible to directly learn the posterior

$$p(\theta \,|\, x)$$

- On the next slide we will learn the training objective that makes this possible.
- An advantage of learning the posterior directly is that we do not need to run an MCMC anymore. The model directly outputs the desired posterior, i.e. our parameter measurement. A disadvantage is that it is difficult to explore different prior distributions of the parameters.

Neural posterior estimation with NFs

- From a simulated data set $\{(\theta_n, x_n)\}_{n=1}^{N}$ drawn from a prior $p(\theta)$ we want to approximate the target posterior $p(\theta|x)$ by a conditional normalizing flow $q_\phi(\theta|x)$.
- We minimize the KL-divergence with the joint PDF as follows.

$= p(\theta|x)\,p(x)$

$$\min_\phi D_{KL}\left(p(\theta,x) \| q_\phi(\theta|x)p(x)\right) = \min_\phi \int p(\theta,x) \log \frac{p(\theta,x)}{q_\phi(\theta|x)p(x)} d\theta dx$$

$$= \min_\phi \int p(\theta,x) \log \frac{p(\theta|x)}{q_\phi(\theta|x)} d\theta dx$$

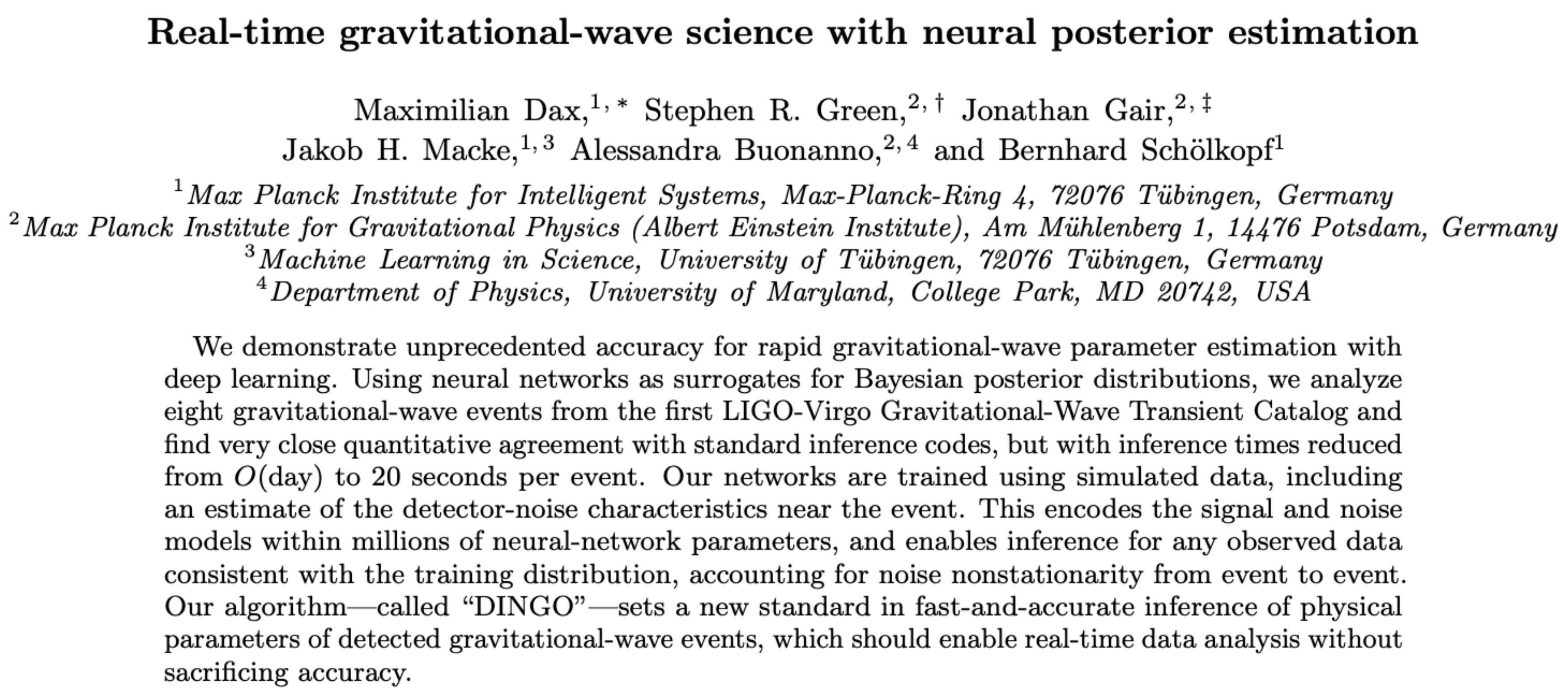
eval expectation value via sampling

$$\approx \min_\phi \sum_i \log p(\theta_i|x_i) - \log q_\phi(\theta_i|x_i)$$

$$\approx \min_\phi \sum_i -\log q_\phi(\theta_i|x_i)$$

- As in the previous case we can thus learn the posterior by sampling from the simulator.

Some examples



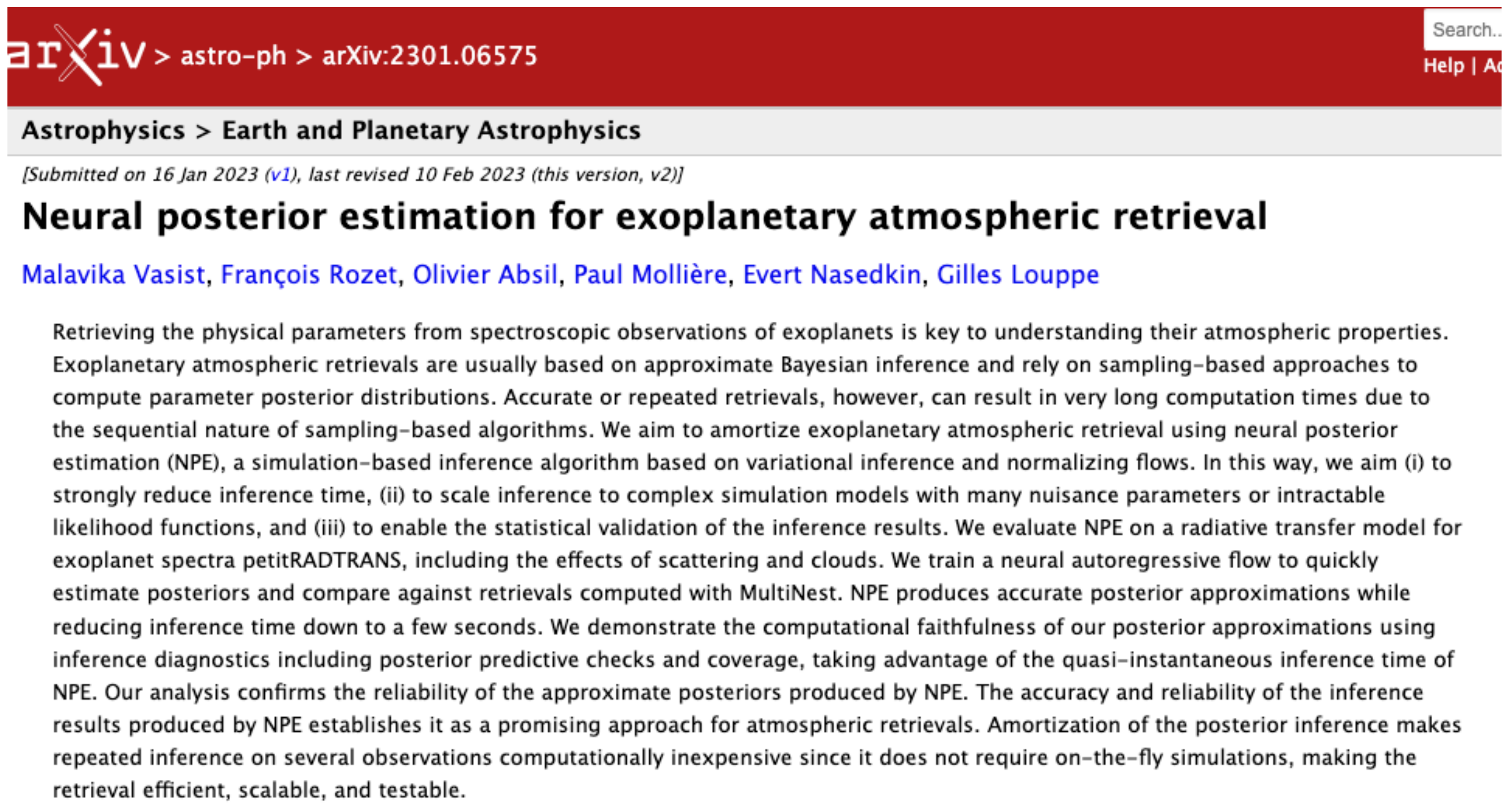
Real-time gravitational-wave science with neural posterior estimation

Maximilian Dax,[1,*] Stephen R. Green,[2,†] Jonathan Gair,[2,‡]
Jakob H. Macke,[1,3] Alessandra Buonanno,[2,4] and Bernhard Schölkopf[1]

[1] *Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, 72076 Tübingen, Germany*
[2] *Max Planck Institute for Gravitational Physics (Albert Einstein Institute), Am Mühlenberg 1, 14476 Potsdam, Germany*
[3] *Machine Learning in Science, University of Tübingen, 72076 Tübingen, Germany*
[4] *Department of Physics, University of Maryland, College Park, MD 20742, USA*

We demonstrate unprecedented accuracy for rapid gravitational-wave parameter estimation with deep learning. Using neural networks as surrogates for Bayesian posterior distributions, we analyze eight gravitational-wave events from the first LIGO-Virgo Gravitational-Wave Transient Catalog and find very close quantitative agreement with standard inference codes, but with inference times reduced from O(day) to 20 seconds per event. Our networks are trained using simulated data, including an estimate of the detector-noise characteristics near the event. This encodes the signal and noise models within millions of neural-network parameters, and enables inference for any observed data consistent with the training distribution, accounting for noise nonstationarity from event to event. Our algorithm—called "DINGO"—sets a new standard in fast-and-accurate inference of physical parameters of detected gravitational-wave events, which should enable real-time data analysis without sacrificing accuracy.

https://arxiv.org/pdf/2106.12594.pdf



Astrophysics > Earth and Planetary Astrophysics

[Submitted on 16 Jan 2023 (v1), last revised 10 Feb 2023 (this version, v2)]

Neural posterior estimation for exoplanetary atmospheric retrieval

Malavika Vasist, François Rozet, Olivier Absil, Paul Mollière, Evert Nasedkin, Gilles Louppe

Retrieving the physical parameters from spectroscopic observations of exoplanets is key to understanding their atmospheric properties. Exoplanetary atmospheric retrievals are usually based on approximate Bayesian inference and rely on sampling-based approaches to compute parameter posterior distributions. Accurate or repeated retrievals, however, can result in very long computation times due to the sequential nature of sampling-based algorithms. We aim to amortize exoplanetary atmospheric retrieval using neural posterior estimation (NPE), a simulation-based inference algorithm based on variational inference and normalizing flows. In this way, we aim (i) to strongly reduce inference time, (ii) to scale inference to complex simulation models with many nuisance parameters or intractable likelihood functions, and (iii) to enable the statistical validation of the inference results. We evaluate NPE on a radiative transfer model for exoplanet spectra petitRADTRANS, including the effects of scattering and clouds. We train a neural autoregressive flow to quickly estimate posteriors and compare against retrievals computed with MultiNest. NPE produces accurate posterior approximations while reducing inference time down to a few seconds. We demonstrate the computational faithfulness of our posterior approximations using inference diagnostics including posterior predictive checks and coverage, taking advantage of the quasi-instantaneous inference time of NPE. Our analysis confirms the reliability of the approximate posteriors produced by NPE. The accuracy and reliability of the inference results produced by NPE establishes it as a promising approach for atmospheric retrievals. Amortization of the posterior inference makes repeated inference on several observations computationally inexpensive since it does not require on-the-fly simulations, making the retrieval efficient, scalable, and testable.

https://arxiv.org/abs/2301.06575

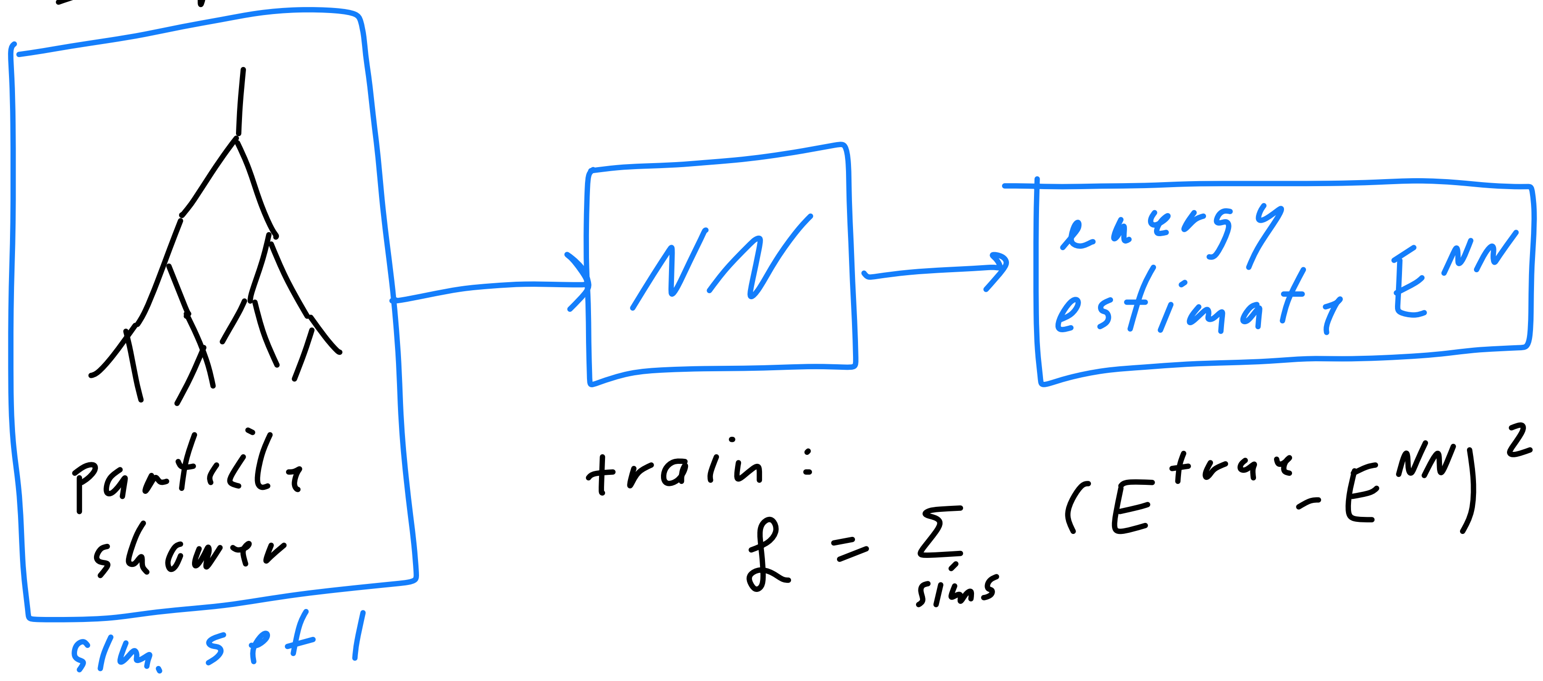
Simulation-Based Inference

More on Uncertainty with Neural Networks

Neural networks as summary statistics

- Above we have developed SBI. Usually the data that goes into SBI is no the raw data but some summary statistic thereof.
- If we simply "train a neural network" to learn a parameter, the neural network can be considered as such a summary statistic.
- We first train the neural network to extract the parameters, e.g. using MSE loss. Then we learn the posterior or the likelihood of the neural network observable with SBI (e.g. using a flow). In this way we get statistical "error bars" for the neural network measurement.
- Example:

Step 1: train summary



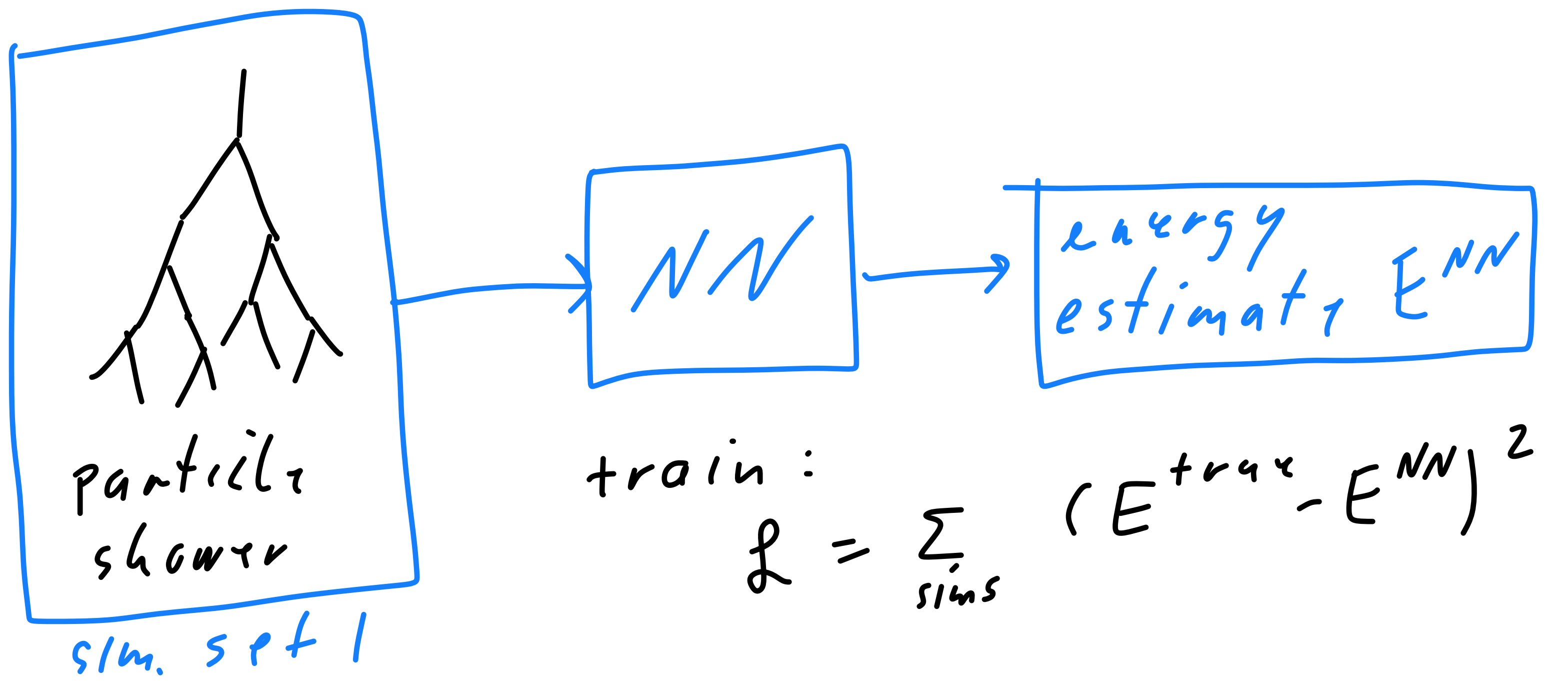
train: $\mathcal{L} = \sum_{\text{sims}} (E^{\text{true}} - E^{NN})^2$

Step 2: SBI

- make new pairs $\{E^{\text{true}}, E^{NN}\}$
- use NLE or NPE to get Likelihood or posterior

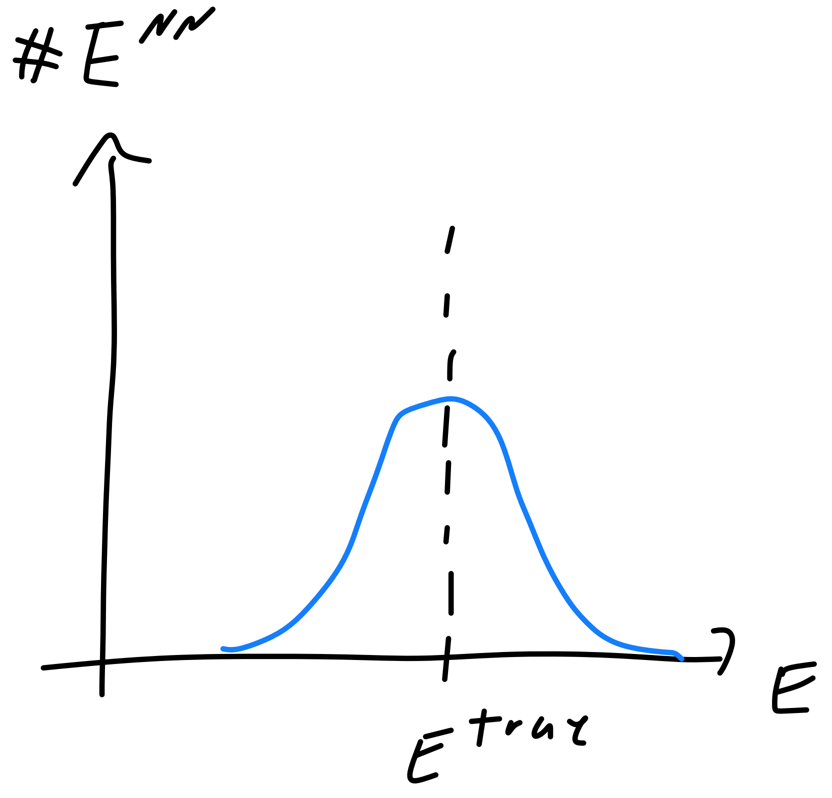
Summary statistics are often Gaussian

- Summary statistics are often Gaussian due to the Central Limit Theorem. This is also true for the output of neural networks (i.e. neural network summary statistics).
- If we can assume Gaussianity (which we can test using simulations), then we can estimate the mean and covariance of the trained neural network output by running many simulations through the trained network. In this case full SBI with a normalizing flow is not necessary.



train:

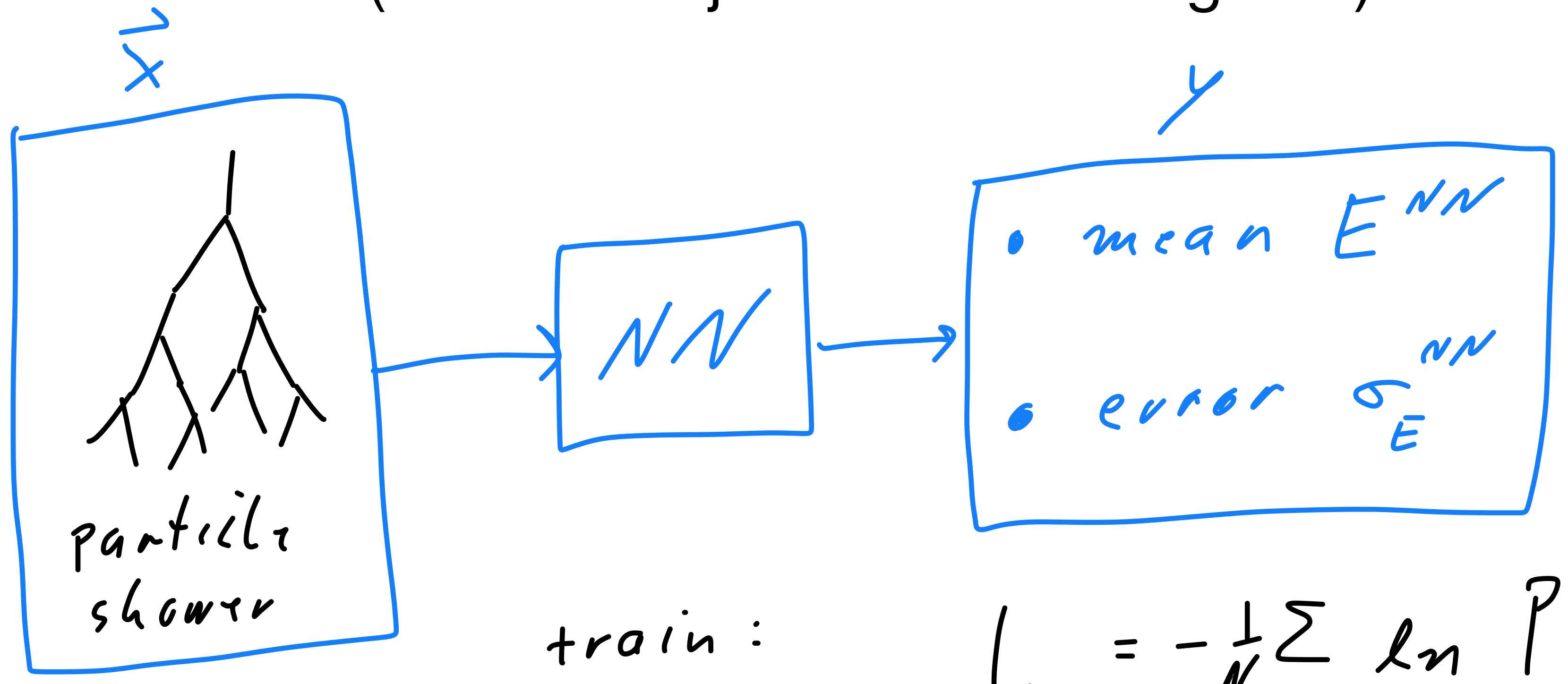
$$\mathcal{L} = \sum_{\text{sims}} (E^{\text{true}} - E^{NN})^2$$



The trained NN gives estimates that are Gaussian around the truth.

Neural networks can be trained to output error bars

- The method we advocated before to assign error bars is to first train the neural network, then keep the weights fixed, and get error bars from SBI.
- However it is also possible to train the neural network to output error bars. For example, for some samples the network may be more confident in the measurement than for other samples. This confidence can be learned from the data.
- A typical strategy is to make the neural network output the mean and the variance of the measurement (rather than just the a 1-shot guess).



assume Gaussian
$\mu = E^{NN}$
$V = (\sigma_E^{NN})^2$

train:

$$L = -\frac{1}{N}\sum_i \ln P^{NN}(y_i \mid x_i)$$

$$\Rightarrow \quad \text{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\log(2\pi\sigma_i^2) + \frac{(y_i-\mu_i)^2}{2\sigma_i^2}\right)$$

- Note however that there is no formal guarantee that the learned error bars will be correct. SBI is a somewhat more systematic approach.

Types of uncertainty

- As you know, there are several types of uncertainty in an analysis:
  - **Statistical uncertainty** (also called **aleatoric uncertainty** in the context of machine learning): this is due to the inherent limitation of the data (e.g. detector noise) and the analysis method (e.g. summary statistics information loss). It can be reduced by getting mode data.
  - **Systematic uncertainty**: this is due to biases and flaws in the measurement. It cannot be reduced by taking more data but only by making better measurements with less bias.
- In machine learning we often use a third concept, **epistemic uncertainty**.
  - Epistemic uncertainty is similar to systematic uncertainty but it is not exactly the same thing.
  - Epistemic uncertainty deals with the uncertainty due to the model's inadequacy while systematic uncertainty usually means flaws in the measurement method.
  - Example of epistemic uncertainty: **Several neural networks trained on the same data give slightly different measurements** and error bars. We do not know which (if any) is correct.

Epistemic uncertainty in neural networks

- The basic idea is to have an ensemble of neural network predictions. If all agree, epistemic uncertainty is low; if they disagree, it is high. There are different versions of that idea:

- **Bayesian Neural Networks**
  - Bayesian Neural Networks (BNNs) are an approach to model epistemic uncertainty by **placing a probability distribution over the weights** of the network. By doing this, BNNs not only provide predictions but also give a measure of uncertainty in those predictions based on the posterior distribution of the weights.
    - **Advantages**: Provides a principled way of quantifying uncertainty.
    - **Challenges**: Computationally expensive and often complex to implement due to the need for approximations like variational inference or Monte Carlo methods to compute the posterior distributions.
    - Typical weight distribution: Gaussian, uncorrelated. But there is no one-size-fits-all answer, and the choice of prior can significantly affect the model's performance and its inferred posterior distributions.

Epistemic uncertainty in neural networks

- **Monte Carlo Dropout**
  - Monte Carlo (MC) Dropout is a practical and widely used method for estimating uncertainty in neural networks. It involves applying dropout not only during training but also at test time, allowing the model to generate different outputs by randomly dropping units during multiple forward passes.
  - **Advantages**: Easy to implement and less computationally intensive than full Bayesian methods.
  - **Challenges**: The estimates of uncertainty may depend significantly on the dropout rate and the number of stochastic forward passes.

- **Deep Ensembles**
  - Deep Ensembles involve training multiple versions of the same model on the same data, but with different initializations or even slightly different model architectures. The variance in the predictions from these models can be used as a measure of epistemic uncertainty.
  - **Advantages**: Often provides improved prediction accuracy and uncertainty estimation over the other methods.
  - **Challenges**: Requires more computational resources since multiple models need to be trained and stored.

Epistemic uncertainty in neural networks

- Some comments on these methods:
  - **https://arxiv.org/abs/2004.10710** **Deeply Uncertain:** Comparing Methods of Uncertainty Quantification in Deep Learning Algorithms
  - Often one reports the statistical and epistemic uncertainty as two different errors.

$$\sigma_{pr} = \sqrt{\sigma_{al}^2 + \sigma_{ep}^2}$$

  - **https://www.sciencedirect.com/science/article/pii/S1566253521001081** A review of uncertainty quantification in deep learning: Techniques, applications and challenges
  - **Epistemic uncertainty in SBI**: By treating a NN as a summary statistic it is not important that the NN is exact. A bias would be corrected for in the likelihood or posterior (though the NN could be sub-optimal if not trained out or too little capacity). However the ML model used in the SBI, i.e. the normalizing flow, can give incorrect probabilities. One could train several of them to test their agreement (as part of the tests of SBI results).
  - If the **data is out of the training data distribution** (i.e. the simulations are "wrong") all bets are off ("unknown unknowns"). So we put a lot of effort into making them reliable in the range where we use them, and also sample over uncertain parameters ("known unknowns").

Simulation-Based Inference

Explicit inference

Recall: Implicit vs Explicit inference

- In most situations a simulation does not provide a probability density (likelihood) $\mathscr{L}(x \,|\, \theta)$ of observations given parameters. Such simulations are sometimes called **implicit models**.
- Implicit means that their **likelihood cannot be computed explicitly**, i.e. it is not computationally tractable. We only get samples of the simulation.
- On the other hand, models or simulations that do provide a likelihood are called **explicit models**. Recall for example Gaussian likelihoods.
- A key problem in **explicit inference is to marginalize over the latent variables**, such as the random initial conditions of a simulation.

$$p(x|\theta) = \int \mathrm{d}z \; p(x, z|\theta)$$

- For comparison to our study of SBI I now want to discuss a specific example of explicit inference. This approach is also sometimes called **“forward modelling” or Bayesian hierarchical modelling.** This is a general approach to many problems that is important to know.

Explicit inference with probabilistic forward modelling

- Assume that we have a simulator, called the **"forward model" f,** depending on **parameters** θ that describes the evolution of a system starting from a **signal s** (e.g. the initial conditions of the forward process):

$$f(\mathbf{s}, \Theta)$$

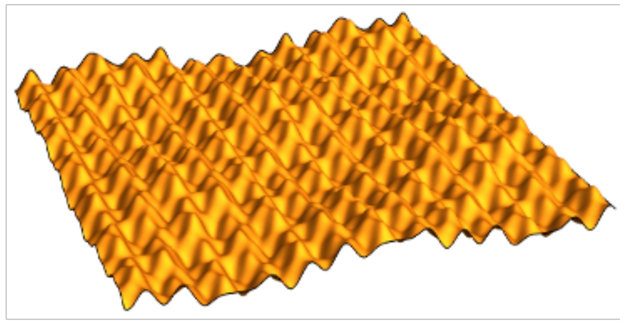
- We want to infer the **parameters** θ and the **signal s** from **data d,** which we take to be a **the forward evolved signal plus some observational noise.**

$$\mathbf{d} = f(\mathbf{s}) + n$$

- This setup is also called an **"inverse problem"**, i.e. we want to reconstruct the signal from the data, assuming that we know the forward model as a function of some parameters. Inverse problems are generally ill-posed (need to regularize).

Example: Cosmology forward model

Initial conditions of the universe s

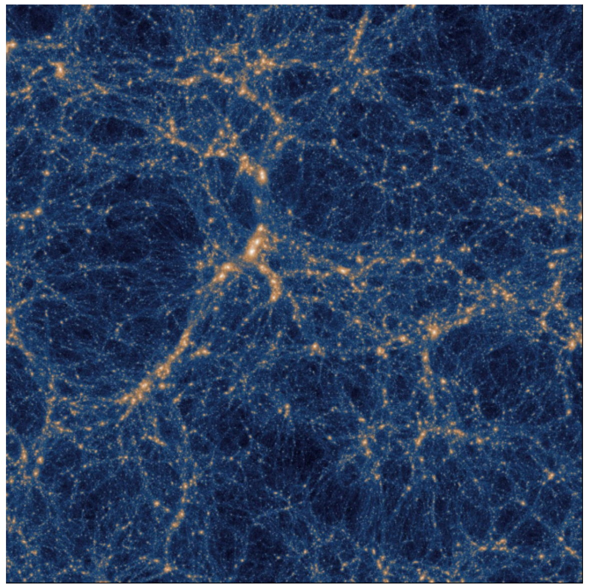


Drawn from prior P(s)

forward

Function of θ

Matter distribution of the universe



forward

Function of θ

Observable galaxies

forward

Function of θ

Telescope imaging model

forward

Function of θ

Data d

Example of a very complicated forward model that maps from the initial conditions of the universe to (simulated) observed data from a telescope. Note that we can easily add new effects to the forward model. Clearly the computational challenge is enormous.

Example: Cosmology forward model

- Assuming the observational noise is Gaussian we can write an explicit likelihood of the form

$$\log \mathcal{L}(\mathbf{d}|\mathbf{s},\Theta) = -\frac{1}{2}(f(\mathbf{s},\Theta) - \mathbf{d}^{obs})^T N^{-1}(f(\mathbf{s},\Theta) - \mathbf{d}^{obs}) + \text{const.}$$

- To complete the posterior we need to add a prior for the parameters parameters θ and s which we want to infer.
- Then we need to sample the posterior. Since **s** is usually very high dimensional, normal MCMC will not work.
- However there are sampling methods that work in very high dimensions, in particular Hamiltonian Monte Carlo (HMC) and versions of Variational Inference (VI). These **require the forward model to be differentiable**.
- Optimization in very high dimensions requires derivatives to find the minimum. For this reason **differentiable simulations** have become an important topic all over physics.
- More details about this approach in cosmology and references can be found in my cosmology lecture notes.

Examples: differentiable cosmology simulations

- https://arxiv.org/abs/2010.11847 FlowPM: Distributed TensorFlow Implementation of the FastPM Cosmological N-body Solver
- https://arxiv.org/abs/2211.09815 Differentiable Cosmological Simulation with Adjoint Method
- https://www.youtube.com/watch?v=Epsgh6vr0qs&ab_channel=ParticleMeshWithDerivatives
- https://arxiv.org/abs/2002.00965 Bayesian de-lensing delight: sampling-based inference of the primordial CMB and gravitational lensing