Introduction to Simulation Based Inference

Enhancing synthetic models with Artificial Intelligence

Alina Bazarova, Jose Robledo, Stefan Kesselheim

Tutorial overview

- What is Simulation Based Inference (SBI) and how it is connected to classical Bayesian Inference

- Basic concepts of the classical Bayesian Inference

- Setting-up an SBI framework and adapting classical Bayesian inference examples to it.

- Hands-on experience with python sbi package within a Jupyter environment

- SBI inference at different levels of model granularity

- Distributing SBI to run on multiple nodes

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

General concepts of Bayesian Inference

- Essentially a type of **statistical inference** (different from classical/frequentist statistical inference)

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

General concepts of Bayesian Inference

- Essentially a type of **statistical inference** (different from classical/frequentist statistical inference)
- Bayesian approach to inference is about **preserving uncertainty**

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

General concepts of Bayesian Inference

- Essentially a type of **statistical inference** (different from classical/frequentist statistical inference)
- Bayesian approach to inference is about **preserving uncertainty**
- The outputs are not point estimates, but rather **probability distributions**.

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

General concepts of Bayesian Inference

- Essentially a type of **statistical inference** (different from classical/frequentist statistical inference)
- Bayesian approach to inference is about **preserving uncertainty**
- The outputs are not point estimates, but rather **probability distributions**.
- Bayesian approach is **based on observed data** and the estimates are updated as more data arrive (usage of conditional probability)

What is the Simulation Based Inference?

Also known as **Likelihood-free Bayesian Inference**

One step back: Let us talk about Classical **Bayesian** Inference! What is it good for?

General concepts of Bayesian Inference

- Essentially a type of **statistical inference** (different from classical/frequentist statistical inference)
- Bayesian approach to inference is about **preserving uncertainty**
- The outputs are not point estimates, but rather **probability distributions**.
- Bayesian approach is **based on observed data** and the estimates are updated as more data arrive (usage of conditional probability)
- Therefore, **more flexibility,** possibly **more information**

Formalism. Bayes rule (theorem).

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}$$

Where usually A is parameters of the model, X is data

$$P(A_j|X) = \frac{P(X|A_j)P(A_j)}{\sum_{i=1}^{k} P(X|A_i)P(A_i)}$$

A full version, where $\{A_1, \dots, A_k\}$ is a partition of A and $j \in \{1, \dots, k\}$

$$posterior = \frac{prior \times likelihood}{evidence}$$

Reformulated in Bayesian language

Continuous space

Let $X, Y \in \mathbb{R}$, $p_X(x)$ is probability density of X (and respective of Y), namely

$p_X(x) > 0$ and $\displaystyle\int_{\mathbb{R}} p_X(x)dx = 1$ $\qquad P(X \in A) = \displaystyle\int_A p_X(x)dx$

Then $p_{X|Y}(x\,|\,y) = \dfrac{p_{X,Y}(x,y)}{p_Y(y)}$, $\qquad p_Y(y) = \displaystyle\int_{\mathbb{R}} p_{X,Y}(x,y)dx$

$$p_{Y|X}(y\,|\,x) = \frac{p_Y(y)p_{X|Y}(x\,|\,y)}{p_X(x)} = \frac{p_Y(y)p_{X|Y}(x\,|\,y)}{\int_{\mathbb{R}} p_X(x)p_{Y|X}(y\,|\,x)dy}, \quad p(y\,|\,x) \propto p(x\,|\,y)p(y)$$

continuous Bayes rule

Possible issues with $\frac{p_Y(y)p_{X|Y}(x|y)}{\int_{\mathbb{R}} p_X(x)p_{Y|X}(y|x)dy}$

- Likelihood $p(x \mid y)$ might be very complicated
- The integral in the denominator is often intractable, hence computational methods (MCMC, EM etc.)
- Computational methods are usually iterative and cannot be efficiently parallelised. Hence, they may take a lot of time, a lot of compute etc.
- Can be a problem when the decision based on the new data has to be made quickly, or when calibrating the model, or when computational resources are limited!

Note:

- $p(x \mid y)$ is our model of the data: data-generating distribution
- $p(y)$ is what we think about the parameters of the model *a priori* (prior)

Bayesian vs Frequentist murder trial

Assume you are (hopefully falsely) accused of a murder and have to face a jury in a misfortunate country where the guilt presumed over innocence (null hypothesis is that one is guilty).

The CCTV footage indicates that you were in the same house as the victim on the night of a murder. There are two types of trial:

1. **Frequentist trial.** The jurors specify a model based on the previous trials: if you commit the murder, 30% of the time you would have been seen by the CCTV. Since the probability $P(security\ camera\ footage \mid guilt) > 0.05$, you are declared guilty.

2. **Bayesian trial.** The jury first are looking at the evidence, such as absence of previous violent conduct etc. and based on that assign a prior probability of $\frac{1}{1000}$. They compute probability according to Bayes rule

$$P(guilt \mid security\ camera\ footage) = \frac{P(security\ camera\ footage \mid guilt) P(guilt)}{P(security\ camera\ footage)} = \frac{0.3 \cdot 0.001}{0.3 \cdot 0.001 + 0.3 \cdot 0.999} = 0.001 < 0.05$$

And therefore you are declared innocent.

Coin-flipping example

Suppose, that you are unsure about the probability of heads in a coin flip (spoiler alert: it's 50%). You believe there is some true underlying ratio, call it p , but have no prior opinion on what p might be.

We begin to flip a coin, and record the observations: either H or T. This is our observed data.

Question to ask: how will our inference change as we observe more and more data?

$P(H = s) = \binom{n}{s} p^s (1-p)^{n-s}$, prior is uniform (constant density)

$$P(p = x \mid s, n) = \frac{P(s, n \mid x) P(x)}{\int P(s, n \mid y) P(y) dy} = \frac{\binom{n}{s} x^s (1-x)^{n-s}}{\binom{n}{s} \int y^s (1-y)^{n-s} dy} = \frac{x^s (1-x)^{n-s}}{B(s, n-s)} \sim Beta(s+1, n-s+1)$$

What is Simulation Based Inference?

Here is where the SBI come in: Why don’t we use an efficient approximation of the posterior with AI?

A simple example: amortized inference

What is Simulation Based Inference?

Here is where the SBI come in: Why don’t we use an efficient approximation of the posterior with AI?

A simple example: amortized inference

- Generate **A LOT** of **simulation outputs**
- Optional: do a **summary statistics** on those

What is Simulation Based Inference?

Here is where the SBI come in: Why don't we use an efficient approximation of the posterior with AI?

A simple example: amortized inference

- Generate **A LOT** of **simulation outputs**
- Optional: do a **summary statistics** on those
- Feed them into a **neural network** -> obtain an **approximate posterior distribution**

What is Simulation Based Inference?

Here is where the SBI come in: Why don’t we use an efficient approximation of the posterior with AI?

A simple example: amortized inference

- Generate **A LOT** of **simulation outputs**
- Optional: do a **summary statistics** on those
- Feed them into a **neural network** -> obtain an **approximate posterior distribution**
- Feed the **actual observations** into the model

How to estimate posterior distribution?

Three main methods:

- **Neural Posterior Estimation**, (NPE) : we estimate the whole thing and can immediately use it!

$$p(y|x) \propto p(x|y)p(y)$$

How to estimate posterior distribution?

Three main methods:

- **Neural Posterior Estimation**, (NPE) : we estimate the whole thing and can immediately use it!

$$p(y|x) \propto p(x|y)p(y)$$

- **Neural Likelihood Estimation** (NLE): we estimate only the likelihood $p(x|y)$

We sample from the posterior distribution using sampling algorithm (e.g. MCMC)

How to estimate posterior distribution?

Three main methods:

- **Neural Posterior Estimation**, (NPE) : we estimate the whole thing and can immediately use it!

$$p(y|x) \propto p(x|y)p(y)$$

- **Neural Likelihood Estimation** (NLE): we estimate only the likelihood $p(x|y)$

We sample from the posterior distribution using sampling algorithm (e.g. MCMC)

- **Neural Likelihood-Ratio Estimation** (NRE): a bit more complicated, we estimate $\frac{p(x|y)}{p(x|y')}$

Requires a deeper insight into sampling algorithm y is the current value of the parameters, and y' is the one proposed by the sampler.

How to estimate posterior distribution?

Three main methods:

- **Neural Posterior Estimation**, (NPE) : we estimate the whole thing and can immediately use it!

$$p(y\,|\,x) \propto p(x\,|\,y)p(y)$$

- **Neural Likelihood Estimation** (NLE): we estimate only the likelihood $p(x\,|\,y)$

We sample from the posterior distribution using sampling algorithm (e.g. MCMC)

- **Neural Likelihood-Ratio Estimation** (NRE): a bit more complicated, we estimate $\dfrac{p(x\,|\,y)}{p(x\,|\,y')}$

Requires a deeper insight into sampling algorithm y is the current value of the parameters, and y' is the one proposed by the sampler.

Can also be used **for frequentist** inference!

Graphical representation

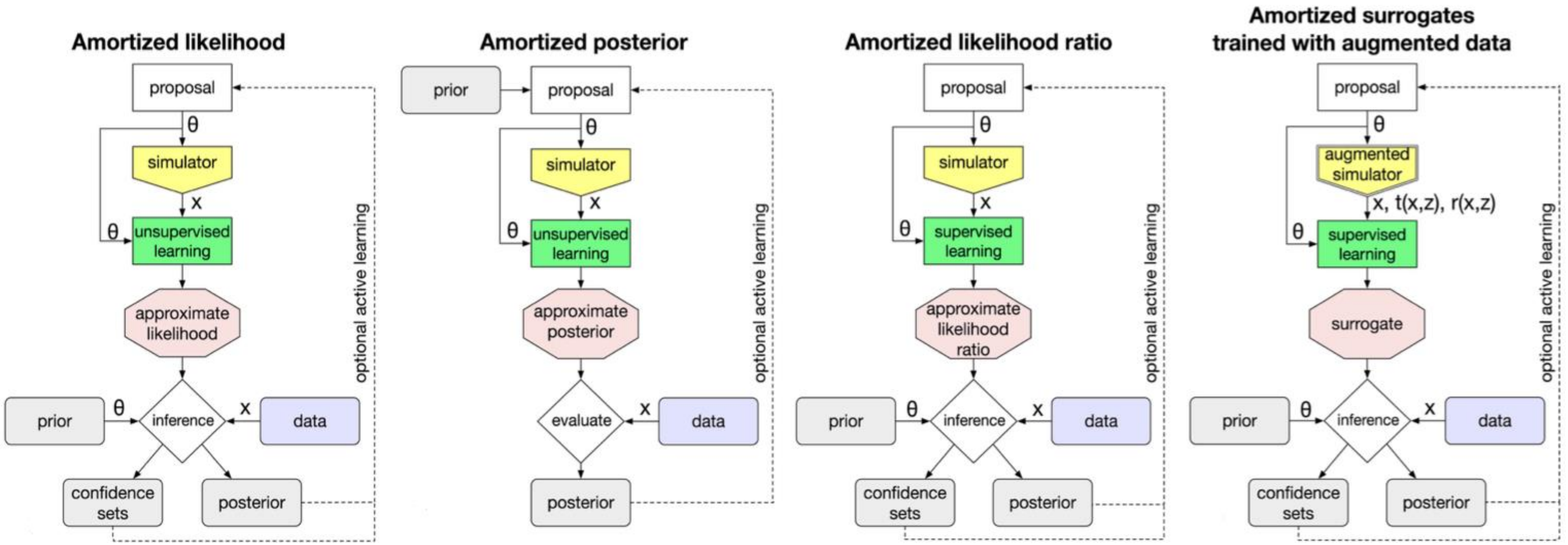

Figure from Frontier of simulation-based inference, Cranmer et al. [*10.1073/pnas.191278911*](https://doi.org/10.1073/pnas.191278911)

Jupyter notebook 1 – warm-up example in an SBI framework
Jupyter notebook 2 – data example, comparison with the classical Bayesian inference based on MCMC

Deep Learning component

1. Approximating posterior or likelihood with **normalizing flows:**

Deep Learning component

1. Approximating posterior or likelihood with **normalizing flows:**

- Transforms a simple source distribution p_Z (e.g. normal distribution) into any arbitrary target distribution p_X by a chain of bijective transformations g_i, where $\mathbf{J_g}$ is a Jacobian

$$p_X(x) = p_Z(g_1(g_0(x))) \det |\mathbf{J}_{\mathbf{g_0}}| \det |\mathbf{J}_{\mathbf{g_1}}|$$

Deep Learning component

1. Approximating <u>posterior</u> or <u>likelihood</u> with **normalizing flows:**

- Transforms a simple source distribution p_Z (e.g. normal distribution) into any arbitrary target distribution p_X by a chain of bijective transformations g_i, where $\mathbf{J_g}$ is a Jacobian

$$p_X(x) = p_Z(g_1(g_0(x)))\,\det|\mathbf{J}_{\mathbf{g_0}}|\,\det|\mathbf{J}_{\mathbf{g_1}}|$$

- We are willing to learn the posterior distribution $p(\theta|x)$. Therefore, as inputs we pass samples from our prior on the parameters θ as inputs to the normalizing flows and the simulation results x are passed as a context.

Deep Learning component

1. Approximating <u>posterior</u> or <u>likelihood</u> with **normalizing flows:**

- Transforms a simple source distribution p_Z (e.g. normal distribution) into any arbitrary target distribution p_X by a chain of bijective transformations g_i, where $\mathbf{J_g}$ is a Jacobian

$$p_X(x) = p_Z(g_1(g_0(x))) \det|\mathbf{J_{g_0}}| \det|\mathbf{J_{g_1}}|$$

- We are willing to learn the posterior distribution $p(\theta|x)$. Therefore, as inputs we pass samples from our prior on the parameters θ as inputs to the normalizing flows and the simulation results x are passed as a context.

- **Note:** If prior and posterior are very **different**, NPE may not learn the posterior well or require excessive amount of simulations, since the simulated samples are **not informative** enough with respect to the **true posterior**

Deep Learning component

2. Estimate the likelihood-ratio $\dfrac{p(x \mid y)}{p(x \mid y')}$, as a part of sampling algorithm, using a **neural network classifier:**

Deep Learning component

2. Estimate the likelihood-ratio $\frac{p(x|y)}{p(x|y')}$, as a part of sampling algorithm, using a **neural network classifier:**

- Classifier learns to discriminate samples from $p(x \mid y)$ from samples from $p(x \mid y')$

Deep Learning component

2. Estimate the likelihood-ratio $\dfrac{p(x|y)}{p(x|y')}$, as a part of sampling algorithm, using a **neural network classifier:**

- Classifier learns to discriminate samples from $p(x \mid y)$ from samples from $p(x \mid y')$

- **Note:** likelihood-ratio is a part of the probability of proposal state y' acceptance $\alpha(y, y')$ in e.g. MCMC algorithm

$$\alpha(y, y') = \frac{p(y')p(x|y')q(y, y')}{p(y)p(x|y)q(y', y)}$$

where $q(y',y)$ is the proposal distribution density at y' given the state y' and cancels out if symmetrical.

Deep Learning component

2. Estimate the likelihood-ratio $\frac{p(x|y)}{p(x|y')}$, as a part of sampling algorithm, using a **neural network classifier:**

- Classifier learns to discriminate samples from $p(x \mid y)$ from samples from $p(x \mid y')$

- **Note:** likelihood-ratio is a part of the probability of proposal state y' acceptance $\alpha(y, y')$ in e.g. MCMC algorithm

$$\alpha(y, y') = \frac{p(y')p(x|y')q(y, y')}{p(y)p(x|y)q(y', y)}$$

where $q(y',y)$ is the proposal distribution density at y' given the state y' and cancels out if symmetrical.

To summarize: NPE is a somewhat **more straightforward** method, however the **least efficient** in case of **false assumptions** on the model. NRE aims to tackle the latter problem.

On the Sequential estimation

Another way to tackle the situation when the prior is far a way from the true posterior:

Jupyter notebook 3: flexible interface of the SBI package
Jupyter notebook 4: using summary statistics

On the Sequential estimation

Another way to tackle the situation when the prior is far a way from the true posterior:

Sequential estimation

- Initially **proposal prior** is set to the **actual prior** (everything the same as before)

Jupyter notebook 3: flexible interface of the SBI package
Jupyter notebook 4: using summary statistics

On the Sequential estimation

Another way to tackle the situation when the prior is far a way from the true posterior:

Sequential estimation

- Initially **proposal prior** is set to the **actual prior** (everything the same as before)

- After the **network is trained** and an approximated posterior is obtained, **proposal prior is replaced** by the approximate posterior and **retrained.**

Jupyter notebook 3: flexible interface of the SBI package
Jupyter notebook 4: using summary statistics

On the Sequential estimation

Another way to tackle the situation when the prior is far a way from the true posterior:

Sequential estimation

- Initially **proposal prior** is set to the **actual prior** (everything the same as before)

- After the **network is trained** and an approximated posterior is obtained, **proposal prior is replaced** by the approximate posterior and **retrained.**

- Repeat until proposal prior has **converged**

Jupyter notebook 3: flexible interface of the SBI package
Jupyter notebook 4: using summary statistics

If you do a lot of simulations, the resulting posterior is more likely to be close to the true one (provided you have a good, model)!

Distribute the tasks to multiple nodes.

SBI package is already using **joblib** palckage for parallelisation over **multiple cpus**, it is easy to use **Ray** backend

in order to distribute the tasks over **multiple nodes**

Note: one simulation has to be long enough (at least ~10 secs)!

Example on the terminal