Exercise 4

Deadline: 20.12.2023 16:00.
Ask questions in discord to #ask-your-tutor

In this exercise, you will familiarize yourselves with simulation-based inference, using the SIR equations from epidemiology as an example.

Regulations

Please implement your solutions in form of *Jupyter notebooks* (`*.ipynb` files), which can mix executable code, figures and text in a single file. They can be created, edited, and executed in the web browser, in the stand-alone app JupyterLab, or in the cloud via Google Colab and similar services.

Create a Jupyter notebook `epidemiology.ipynb` for your solution and export the notebook to HTML as `epidemiology.html`. Zip all files into a single archive `ex04.zip` and upload this file to MaMPF before the given deadline.
Moreover, please set your **Anzeigename/display name** and **Name in Uebungsgruppen/name in tutorials** in MaMPF to your real name, which should be identical to your name in `muesli` and make sure you **join the submission** of your team via the invitation code before the submission deadline. Check out https://mampf.blog/handing-in-homework-assignments for instructions.

1 Noise-free data

We first start with the pure physics-based simulation of an epidemy. We define the compartments "Susceptible" (S), "Infected" (I), and "Recovered" (R), whose dynamics are described by a system of ordinary differential equations (ODEs), known as *SIR equations*:

$$\begin{aligned}\frac{dS}{dt} &= -\lambda\frac{S\cdot I}{N}\\ \frac{dI}{dt} &= \lambda\frac{S\cdot I}{N} - \mu\cdot I\\ \frac{dR}{dt} &= \mu\cdot I\end{aligned}$$

Here, $N = S+I+R$ is the total size of the population. This simplest form of the SIR model assumes that nobody dies from the infection, and people become permanently immune after recovery. The parameters λ (infection rate), μ (recovery rate), and I_0 (initial number of infected people when the disease is first noticed) determine the behavior of the disease, but they cannot be measured directly. Instead, one observes the dynamics of the cases, i.e. the daily number of newly infected and recovered people $\Delta S(t) = S(t-1) - S(t)$ and $\Delta R(t) = R(t) - R(t-1)$ (both are taken to be positive by convention, although ΔS should conceptionally be negative) for a sequence of days $t = 1...T$. Simulation-based inference attempts to estimate λ, μ, and I_0 by calculating backwards from the observations. Unfortunately, *inverse problems* like this are usually not analytically solvable and call for a solution by means of neural networks.

Using such an SIR model, solve the following tasks:

1. Implement a function `simulate_sir_simple(lam, mu, I_0, T)`, that takes the specified parameters and simulates the epidemy using the Euler forward scheme up to day T. The vector of hidden parameters is therefore $Y = [\lambda, \mu, I_0]$. The function shall return the sequence of observations

$$X = \big[(\Delta S(1), \Delta R(1)), ..., (\Delta S(T), \Delta R(T))\big]$$

For debugging and visualization, it is also useful to return the sequence of corresponding ODE variables

$$C = \big[\big(S(1), I(1), R(1)\big), ..., \big(S(T), I(T), R(T)\big)\big]$$

but these ground-truth values are not needed for the inverse problem itself, only for testing.

2. Run the simulation with $T = 100$ and visualize the results. Determine suitable ranges of the parameters such that the first 100 days exhibit interesting and diverse dynamics. This is important, because when not much happens between day 1 and day T, there will be insufficient information to solve the inverse problem. Use the results of these experiments to define the simulation prior

$$p^{\text{sim}}(Y) = p^{\text{sim}}(\lambda) \cdot p^{\text{sim}}(\mu) \cdot p^{\text{sim}}(I_0)$$

3. Define a feed-forward or convolutional network Φ that predicts parameters directly from observations, $\widehat{Y} = \Phi(X)$. Train this network with the squared loss using a training set generated by the simulation and prior distribution, and evaluate it by a simulated test set. Visualize the accuracy as a function of hyperparameter choices (especially the sizes of the network and training set) in suitable diagrams. Note that this network disregards potential uncertainty and ambiguity of the inverse problem – it just returns a *point estimate*, i.e. a single vector of plausible parameters $\widehat{Y}$ for every given X. For simplicity, the length of the sequence X (i.e. the stopping time T) shall be constant for all training and testing runs.
4. Modify the network Φ so that it can serve as a summary network (feature detector) $h(X)$ for a conditional normalizing flow. That is, remove the network's last layer and possibly fine-tune its hyperparameters. Implement a conditional RealNVP that estimates the posterior $p\big(Y \,|\, h(X)\big)$. Train the two networks jointly by minimzing the NLL loss over the training set. Visualize the resulting posteriors, along with ground-truth values Y^*, for some representative test instances.
5. Determine the quality of your model in terms of posterior calibration for both the marginal distributions of the individual parameters $\widehat{\lambda}$, $\widehat{\mu}$, and $\widehat{I}_0$, and the joint distribution of $\widehat{Y}$ via its energy $E(Y) = -\log p\big(Y \,|\, h(X)\big)$. To this end, for multiple fixed X_i, create samples $\big\{\widehat{Y}_k \sim p\big(Y \,|\, h(X_i)\big)\big\}_{k=1}^{M}$ of parameter estimates from the posterior, and use both the histogram and the empirical CDF methods to check calibration.
6. Check posterior-predictive calibration by resimulation. That is, use the $\widehat{Y}_k$ from each of the samples $\big\{\widehat{Y}_k \sim p\big(Y \,|\, h(X_i)\big)\big\}_{k=1}^{M}$ as an input to `simulate_sir_simple()`. Use the resulting outcomes C_k to define confidence intervals for the resimulated scenarios and determine if the corresponding true outcomes C_i^* are within these confidence intervals.

Comment on your findings, especially on model accuracy as a function of hyperparameter choice.

2 Noisy data

We now extend the basic SIR model with a noise model for the observations. Implement the new function `simulate_sir_noisy(lam, mu, I_0, L, rho, sigma_2, T)`, which outputs noisy sequences

$$\tilde{X} = \big[\big(\Delta\tilde{S}(1), \Delta\tilde{R}(1)\big), ..., \big(\Delta\tilde{S}(T), \Delta\tilde{R}(T)\big)\big]$$

We assume that the noise is multiplicative Gaussian, i.e. follows the rule

$$\Delta\tilde{S}(t) = \Delta S(t-L) \cdot \epsilon_S \qquad \text{with} \qquad \epsilon_S \sim \mathcal{N}(\rho, \sigma^2)$$
$$\Delta\tilde{R}(t) = \Delta R(t-L) \cdot \epsilon_R \qquad \text{with} \qquad \epsilon_R \sim \mathcal{N}(\rho, \sigma^2)$$

The new parameters $L \geq 0$, $\rho \leq 1$, and σ^2 represent the reporting delay, underreporting fraction, and reporting error variance respectively. Visualize the difference between noise-free outcomes X

and noisy outcomes $\tilde{X}$ to find good ranges for the new parameters and extend the prior $p^{\text{sim}}(Y)$ accordingly.

Extend your networks from task 1 to accomodate the additional parameters of the inverse problem. Repeat the experiments and comment on your findings.

3 Model misspecification detection

We now modify the training of the networks from task 1: In addition to the NLL loss, we use an MMD loss on the output of the summary network $h(X)$ to pull the distribution $p\big(h(X)\big)$ towards a standard normal $\mathcal{N}(0, I)$. This allows us to recognize unusual simulation outcomes X_{bad} as outliers of $p\big(h(X_{\text{bad}})\big)$, i.e. outliers to the standard normal, for example via the standard χ^2 test.

First, use test data generated by `simulate_sir_simple()` to make sure that the null hypothesis "All $h(X)$ follow the desired standard normal." cannot be rejected for noise-free data at common choices of the χ^2 test's p-value (e.g. 1%). Fine-tune your hyperparameters if this is not the case.

Now, use test data created with `simulate_sir_noisy()` and determine under which conditions the mismatch between the model assumptions (outcomes are noise-free) and the actual data (noisy outcomes) is detected by the χ^2 test. Test with increasing noise levels and increasing test set size – both should facilitate the detection. Comment on your findings.

4 Sensitivity analysis

Vary the variance of the prior distribution $p^{\text{sim}}(Y)$ (individually for each parameter) during training and make your networks *hyperparameter aware*. That is, pass the current settings of the (log-)variance as an additional input/condition to the networks, so that they learn the correct posterior $p\big(Y \,|\, h(X), \text{Var}(Y)\big)$ for each setting of the variance vector $\text{Var}(Y)$. In this way, the networks amortize the variability in $\text{Var}(Y)$, and sensitivity tests become sufficiently cheap to be practical (otherwise, you would have to train a new model for each setting of $\text{Var}(Y)$, which would be prohibitively expensive). Use a test set to determine how the posterior changes under variations of $\text{Var}(Y)$ and comment on your findings. Are there parameters which are more sensitive to the choice of prior than others? (This would mean that the data are not as informative about these parameters as we would like them to be.)