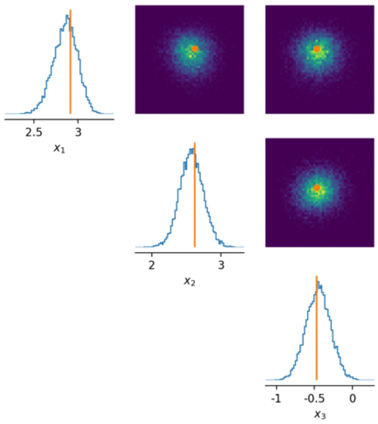
Simulation Based Inference Library

Tutorial 0 – Getting Started

- Simulator – linear Gaussian - return the parameter + 1 + 0.1*N(0,1)
- Prior – box uniform, in all 3 parameters, [-2,2]
- Run 2000 simulations; theta = prior.sample(), x = simulator(theta), inference = inference.append_simulations(theta, x), density_estimator = inference.train(), posterior = inference.build_posterior(density_estimator)
- Then take the posterior, sample from it, simulate data from the posterior samples and plot
- We can assess the posterior for the known samples in this case
- Can also take the posterior, sample from it, simulate data from the posterior samples and assess the predictive power of the posterior
- Also track the log probability – compare the log prob of posterior sample with prior sample

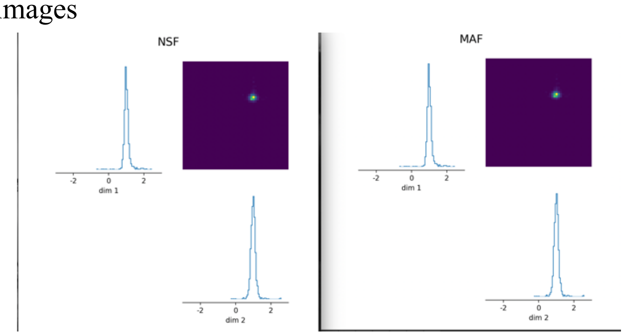


Tutorial 1 – Amortized Posterior Inference on Gaussian Example

- Same basic example as before, but this time with amortization– ability to reevaluate the posterior for different observations without having to rerun inference
- Amortized posterior is one not focused on any one observation
- Two sets of observed data – run the inference on one of them to get a posterior, and then we can draw samples from the posterior given the second observation without rerunning inference
- These also match the ground truth parameters

Tutorial 2 – More Flexibility over Training Loop and Samplers

- 2D example from above, with prior now from [-3,3]
- Now we build our own density estimator – in this case we take an NSF of theta and x
- We use the Adam optimiser and optimise the loss from the density estimator to create our own custom training loop
- After this training, we can directly sample from the posterior given some observed data (need to be careful with batch dimensions)
- Can wrap this up into a direct posterior (like inference.build_posterior) to automatically reject samples outside prior bounds and compute Maximum-a-priori estimate
- We can also use custom data loaders in this way – can be helpful for more complex or larger datasets, such as images



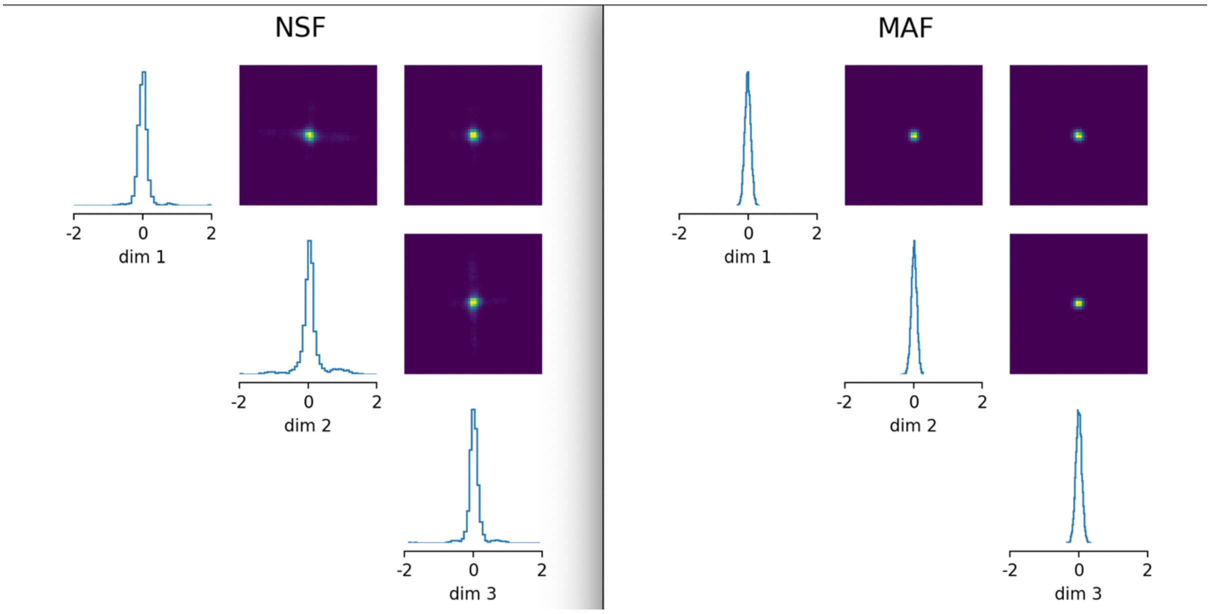
Simulation Based Inference Library

Tutorial 3 – Multi-round Inference

- So far – single round inference – draw parameters from prior, use simulator, train neural network to get parameter – can be inefficient if interested in only one observation
- Multi-round inference alleviates the problem – same as above, but after producing posterior it continues inference – this time it samples from the posterior conditioned on the observation, rather than the prior
- More efficient in the number of simulations, not amortized however, only accurate for this observation
- Beginning of the code – all the same; then create posteriors = [], start with proposal = prior and then update the proposal every time a new posterior is calculated, so each time you are training the density estimator with simulations from the most recent posterior

Tutorial 4 – Custom Density Estimators

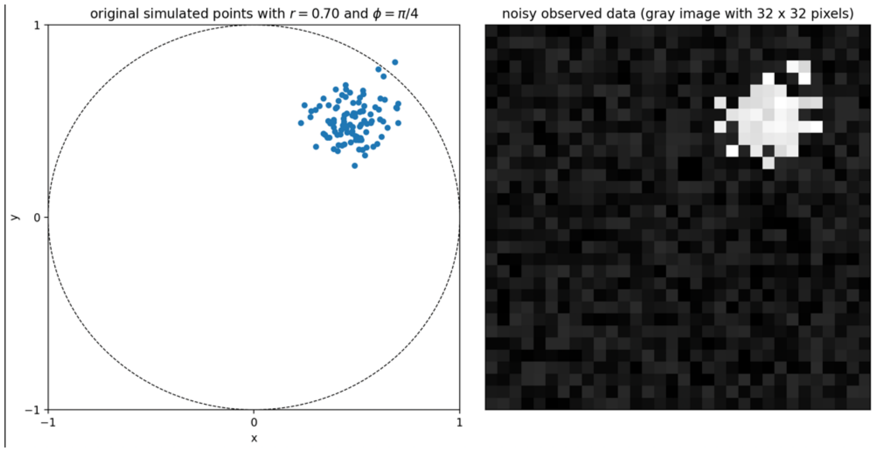
- So far using standard density estimation methods – build_maf or build_nsf
- Can define custom density estimators – make a density estimator and then put it in the inference function; inference = NPE(…*density_estimator*=density_estimator)
- In this example – have done both NSF and MAF custom density estimators, compared the two of them using the same observed data in multi-round inference
- We can also change the hyperparameters of the density estimators in this way – for example, we can change number of hidden features, or number of transforms
- Can also build new density estimators completely from scratch – need to be careful with arguments when designing it this way



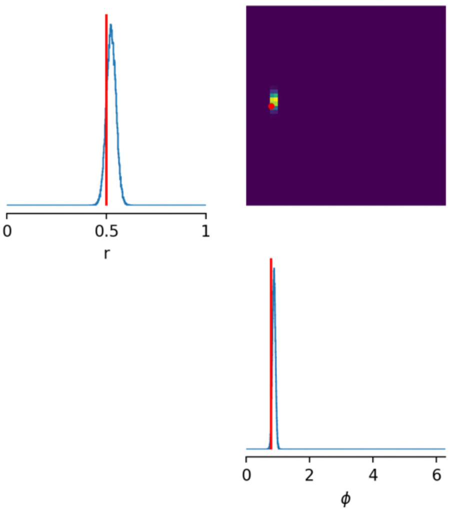
Tutorial 5 – Embedding nets for observations

- Summary statistics are very important – for high dimensional data especially. For complex data, this can be quite difficult to extract a priori, so you need to learn from the data which summary statistics to employ
- This is what embedding neural networks are for – the posterior can be written as $q_\phi(\theta|f_\lambda(x_o))$, where x_o are our observations, ϕ are the parameters of the conditional density estimator and λ the parameters of the embedding neural network
- Simulation goes to the embedding NN, then to the conditional DE; ϕ and λ are learnt jointly through the training
- In this example – our simulator takes (r, ϕ) in the 2D plane and produce Gaussian data centred at that point – then makes a greyscale image of the scattered points plus some noise as well.

- Then we take 32x32 = 1024-dimensional data through a CNN embedding net to get compressed data, which we can then do our regular inference process on (this time with a MAF)
- This is what the initial data looks like:



- And then after inference we get the following results:

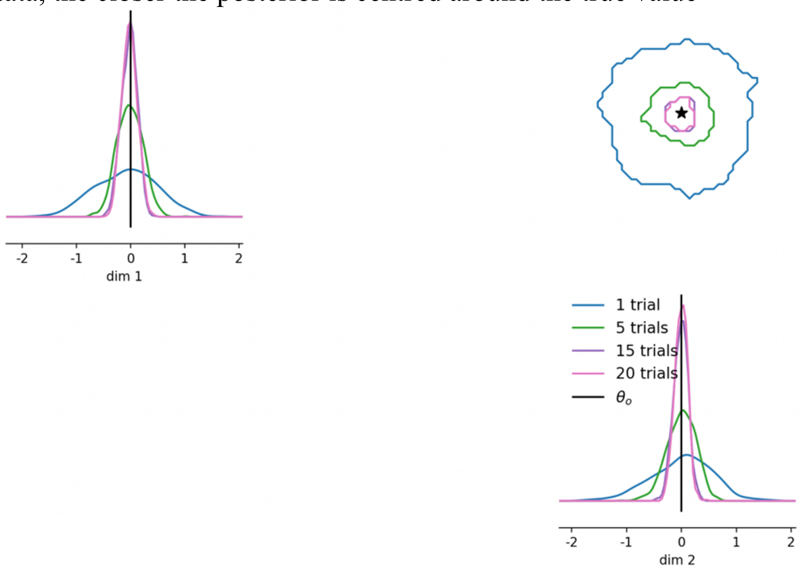


- Can also define custom embedding nets in the same way as custom density estimators – for this tutorial, defined custom CNN class called SummaryNet() and used that, can also import a more expressive CNN from the sbi library

<u>Tutorial 6 – SBI with iid data</u>

- If we have iid data, we might be interested in the posterior given all the data, p(θ| X={x_i}$_1{}^N$)
- When performing NPE, we cannot directly exploit iid assumption as we can do for NLE or NRE, so we take the full observation as an input, x. Thus, the embedding NN must be permutation invariant, and be able to handle a varying number of iid data points to be amortized, but when you have this, it is a fully amortized direct posterior
- To become invariant wrt number of trials, we need to run the simulator multiple times for individual parameter sets to get the training data – we get this done by interleaving in a for loop for this example
- To become permutation invariant, the neural net first learns embeddings for single trials and then performs a permutation invariant operation on those embeddings, e.g., by taking the sum or the mean

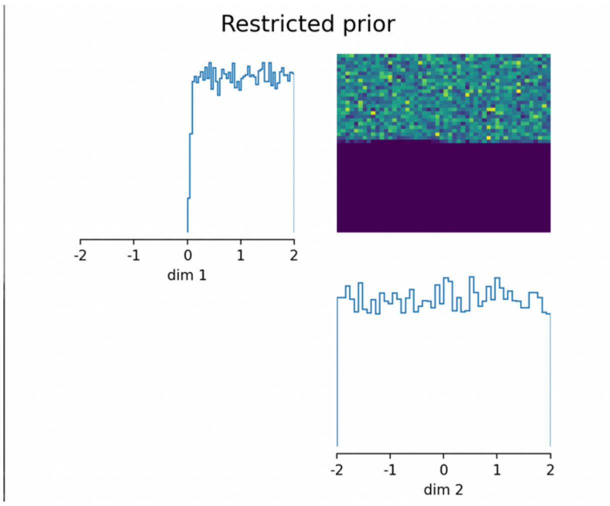
- We get lots of NaN values – this is from the missing trials – have to include this in training – and then we plot the posteriors learned from different numbers of iid data – we can see that the more data, the closer the posterior is centred around the true value

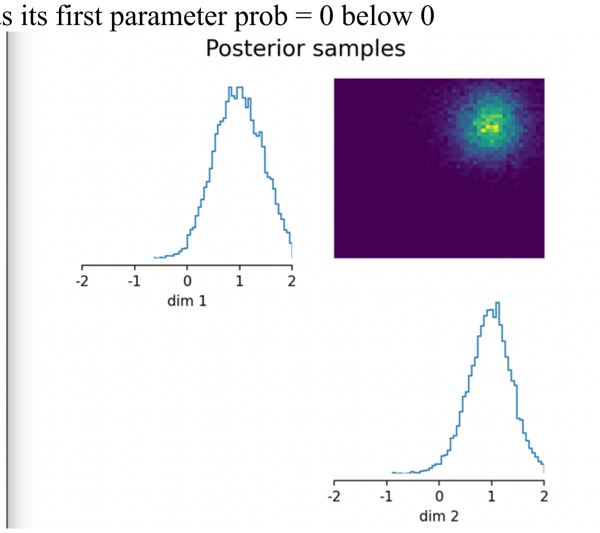


- We can easily obtain posteriors for many different observations, instantly, because NPE is fully amortize

<u>Tutorial 7 – Efficient handling of invalid simulation outputs</u>

- Our simulators can give non-sensical outputs, these data should be rejected – however, this is v inefficient – we do a lot of simulations and get little usable output
- We want the simulator to learn the parts of the parameter space which produce valid outputs – this can then discard any unsuitable parameters drawn from the start, making it more efficient
- Our simulator is defined in a way such that if the first parameter is less than 0 we return NaN, otherwise it's just Gaussian perturbation
- We set up the RestrictionEstimator – then use a classifier and train it so it can find the suitable parts of the parameter space to simulate from
- We then restrict our prior, and future posteriors, to this range, and then we simulate using this data – this time with all the data, rather than having to discard some
- Having trained the classifier to get a restricted prior, we then use it as before – all of a sudden, rather than getting many roughly 520 NaN results out of 1000 we get 0 (usually)
- Can see that the restricted estimator now has its first parameter prob = 0 below 0

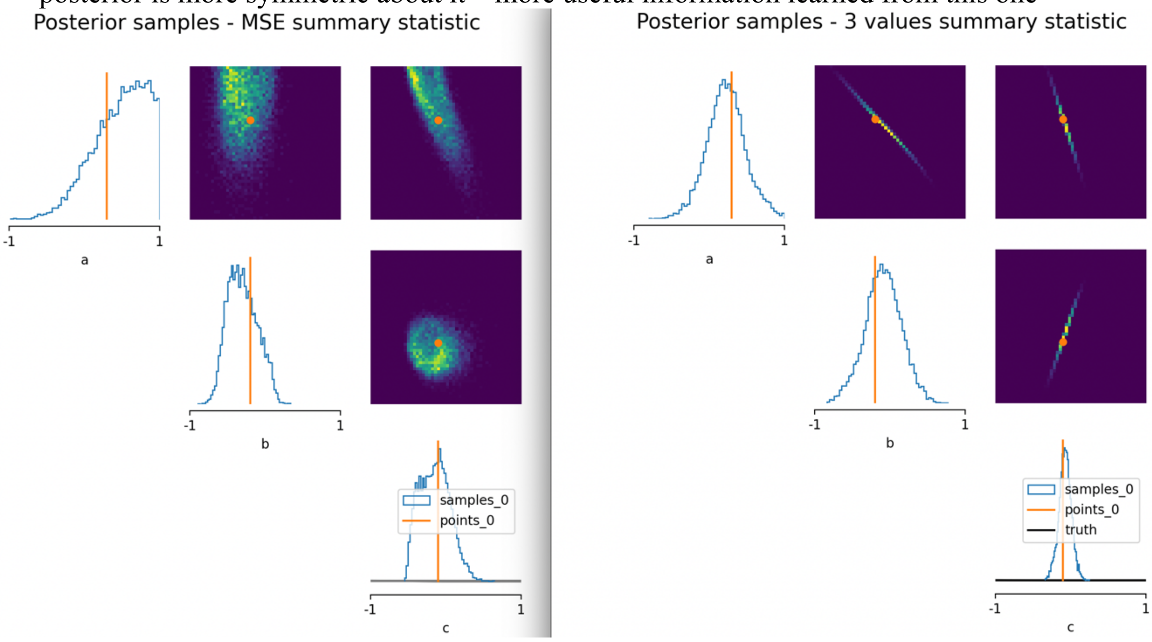




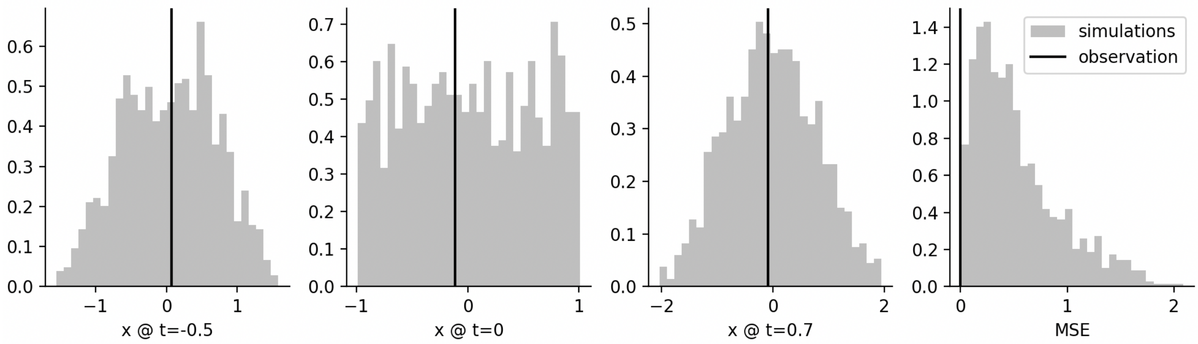
Simulation Based Inference Library

Tutorial 8 – Crafting Summary Statistics

- Hand crafting summary statistics rather than learning them from the data itself as in tutorial 5
- Choosing 2 different summary stats for the data given and comparing their inference output – shows the importance of choosing the right summary stats
- The simulator produces some quadratic output, which is given by uniform prior
- Two summary stats considered – evaluate the function at 3 points, and the MSE between the observed and simulated data
- As observed – the 3-points summary statistic is more concentrated on true parameter, and the posterior is more symmetric about it – more useful information learned from this one



- Why? The summary statistics vary in terms of which simulations they learn from, as the diagram shows – the three points learn from all simulations that cover the observations, so the trained NN can interpolate the data – the MSE learns from simulations only to the right of the MSE so the NN has to extrapolate the data – first is much preferrable

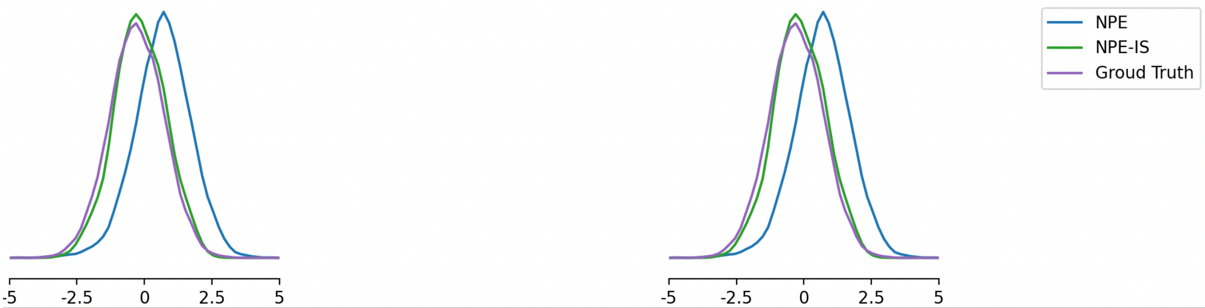


- Good strategy – create histograms like above, see how simulations cluster around observation – if they are off to one end of the simulations, not going to be as good

Tutorial 9 – Refining Posterior Estimates with Rejection Sampling

- SBI normally does sampling from prior and likelihood (via simulator) to get the posterior estimate – instead, we could use a mixture of a simulated estimate for the posterior, $q(\theta| x)$ and likelihood-based importance sampling to get an asymptotically exact estimate for the posterior, $p(\theta| x)$
- The idea is to think of $q(\theta| x)$ as a proposal, draw samples from the proposal $\theta_{Yo} \sim q(\theta| x)$ and then increase each sample with an importance weight $w_{Yo} = p(\theta| x) / q(\theta| x)$ – comes from Monte Carlo proof

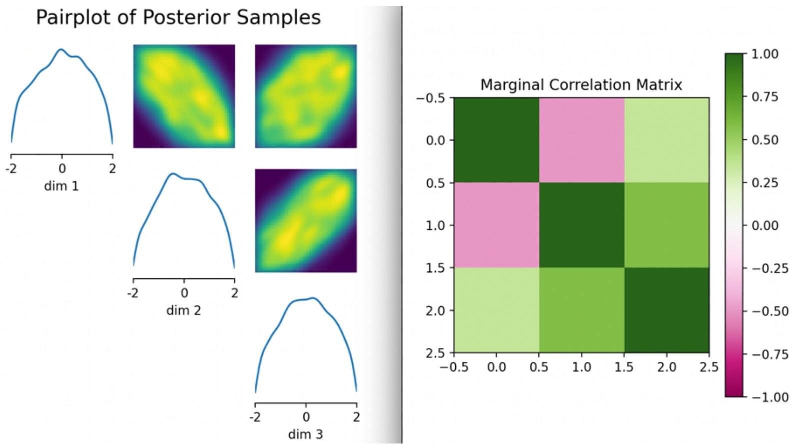
- Now we sample from $\theta_{Y_o} \sim q(\theta | x)$ and the importance weights downweight samples where $q(\theta | x)$ oversamples $p(\theta | x)$ and vice versa
- We do this using the ImportanceSamplingPosterior function – this will do the importance sampling on the trained posterior, done in the same way as before
- We can compare the ground truth distribution, the NPE distribution and the NPE distribution with importance sampling – can see that the importance sampling is far more accurate



- Need some way of sampling from the tractable likelihood – not always helpful

<u>Tutorial 10 – Variability and Compensation Mechanisms with Conditional Distributions</u>

- Advantage of SBI – posterior captures all models that can reproduce experimental data
- Can check whether the parameters can be variable or have to be finely tuned, and also to find potential compensation mechanisms between model parameters – extract conditional distributions from inferred posterior
- Here is a trained posterior's marginals and the Pearson correlation coefficient matrix:



- As seen – little correlation (weak interactions), 1 and 2D marginals fill almost the entire parameter space – in 3D can look at the 3D plot, but inefficient for many dimensions, so we look at conditional marginals
- In the plots below, the diagonal marginals are all but 1 parameter constant, off diagonal is all but 2 parameters constant – can see the conditional correlation matrix as well