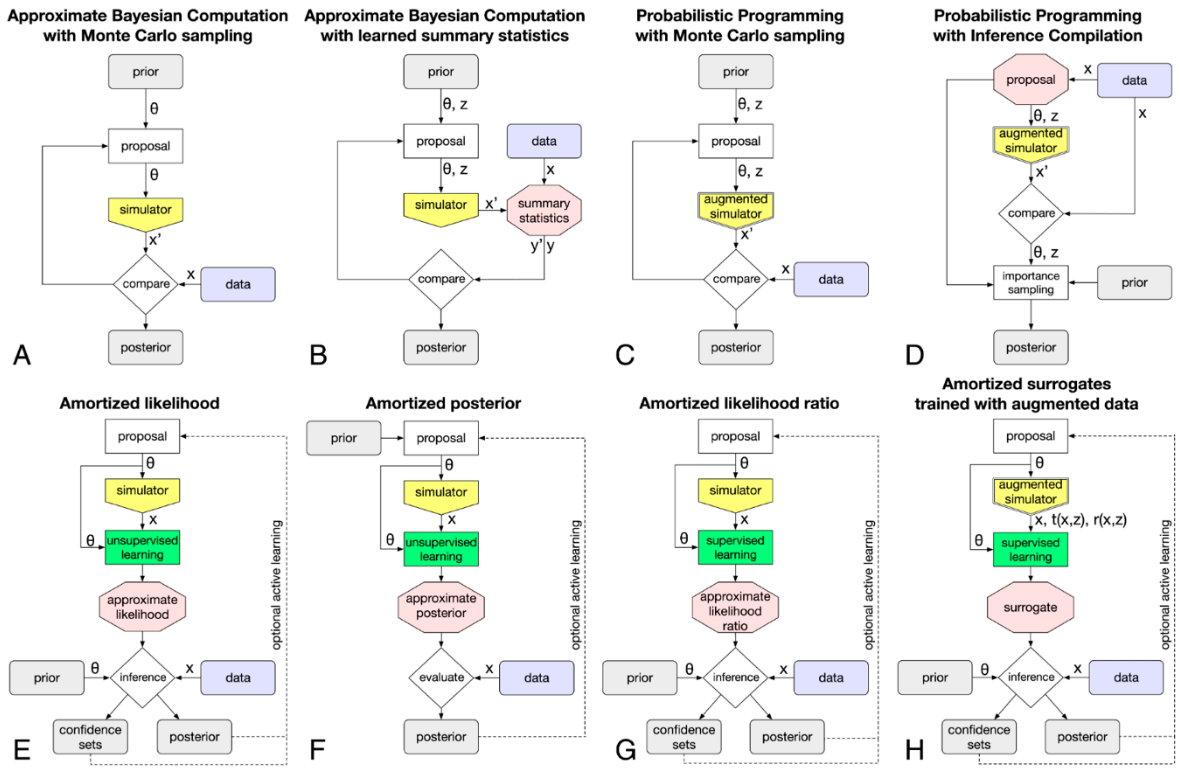
Intro to Simulation Based Inference

Introduction and Motivation

- Otherwise known as likelihood-free inference – used in situations where the likelihood is intractable – quite common for complex systems such as BBH mergers or for astronomical/particle scales
- Sometimes called implicit models, contrasted against prescribed model –likelihood can be explicitly calculated in latter, only approximated in this one
- In all situations, there is a simulator which generates some data based on a parameter input – as seen in the diagram below, this form of inference can be used in sampling, learning summary stats, likelihood and posterior estimation etc.



Approximate Bayesian Computation with Monte Carlo sampling
Approximate Bayesian Computation with learned summary statistics
Probabilistic Programming with Monte Carlo sampling
Probabilistic Programming with Inference Compilation
prior
proposal
simulator
augmented simulator
summary statistics
compare
data
posterior
importance sampling
θ
θ, z
x
x'
y'
y
x, t(x,z), r(x,z)
A
B
C
D
Amortized likelihood
Amortized posterior
Amortized likelihood ratio
Amortized surrogates trained with augmented data
unsupervised learning
supervised learning
approximate likelihood
approximate posterior
approximate likelihood ratio
surrogate
inference
evaluate
confidence sets
optional active learning
E
F
G
H

(A–H) Overview of different approaches to simulation-based inference.

- Active learning – simulator is itself trained and guided using the data generated, improves sample efficiency

Core Concept - Simulators

- The idea is to take in some parameters, and generate synthetic data which can then be compared to the actual data – this comparison helps to infer the actual data (at least in the case of posterior estimation)
- A simulator is a computer program that takes in parameters as input variables and then returns some simulated data – this is often done by sampling a series of latent variables $z_i \sim p_i(z_i|\theta, z_{<i})$ and return $x \sim p(x|\theta, z)$
- The latent variables often refer to unobservable data, and various a lot between different simulators – can vary in discrete vs continuous, dimensionality etc.
- The simulation can combine deterministic and stochastic steps, and the deterministic steps may be either differentiable or discontinuous flow etc.
- Some simulators give direct access to the latent variables at the end of the process – others act just as a black box to give some output data – this can range from a few numbers up to highly structured, high dimensional data

Intro to Simulation Based Inference

- The benefit of this is to generate data with no explicit likelihood – in the case of BBH merger, the latent space is too large to integrate over to get the likelihood
- It's worth noting – time series data, such as that you might get from a GW detector, is not made up of i.i.d points, and so must be treated as a single high-dimensional observation (check?) – compare to collision data for Higgs boson f.ex. which is lots of i.i.d measurements

<u>Traditional methods used for SBI</u>

- Approximate Bayesian Computation – traditional method used for SBI, see 1A above
    - Draw parameters from the prior
    - Simulate data using these parameters
    - Choose a distance function/metric, ρ
    - Set a tolerance ε
    - If $\rho(x_{\mathrm{sim}}, x_{\mathrm{obs}}) < \epsilon$ then accept the parameters into the posterior – the accepted samples will then approximate the posterior
    - In the limit ϵ goes to 0 then the posterior becomes exact, but for continuous data acceptance probability vanishes and requires v large number of simulations, so ϵ chosen sufficiently small to balance sample efficiency and inference quality
- Inference is only valid for a single observation – entire algorithm must be run again for each new observation – best suited for cases with one observation or maybe a few i.i.d.
- Other traditional methods – using histograms or kernel distribution estimation to estimate the distribution of simulated data, and then run analysis on this as if the likelihood were fully tractable – often called Approximate Frequentist Computation
- In the worst case, these methods' simulation requirements increase exponentially with dimensionality of the data

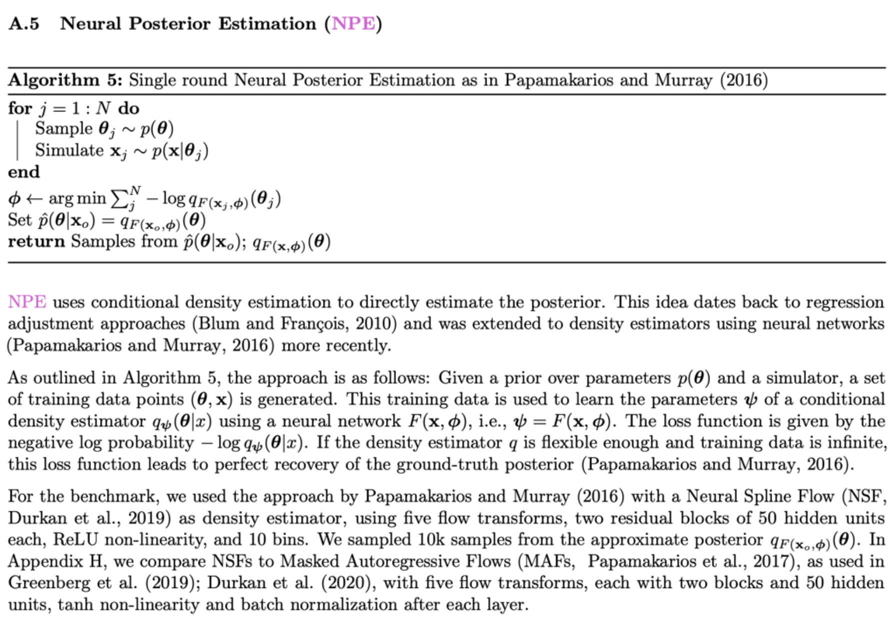
<u>ML Improvements</u>

- Many problems with traditional approach – most notably the high dimensional case
- The improvements in ML, particularly deep neural networks, means that high dimensional data can be dealt with more easily
- Already been applied with normalizing flows
- Active learning – run the simulator at points that we expect to increase our knowledge the most, so after each running of the simulation, we guide the simulator as to which parameter points to pick next
- In Bayesian setting, this is often used to steer the proposal distribution of simulator parameters

<u>Integration of ML into the simulator</u>

- The above applications of ML still treat the simulator as a black box that takes in parameters and produces data, and the processes of simulation and inference are still treated separately.
- Some research is changing this by integrating inference into the simulation stage more closes – one example is probabilistic programming
- This involves conditioning the variables in the program based on the observations, which used to be a challenge as it involved controlling the randomness, however ML advances make this a lot easier
- Also, the additional information that characterises the latent data-generating process can be extracted from simulator and augment the training data – it makes a lot more variables tractable, including but not limited to $p(x|z,\theta)$, $\nabla_z \log p(x,z|\theta)$ and $r(x,z|\theta,\theta') \equiv p(x,z|\theta)/p(x,z|\theta')$

Intro to Simulation Based Inference



A.5 Neural Posterior Estimation (NPE)

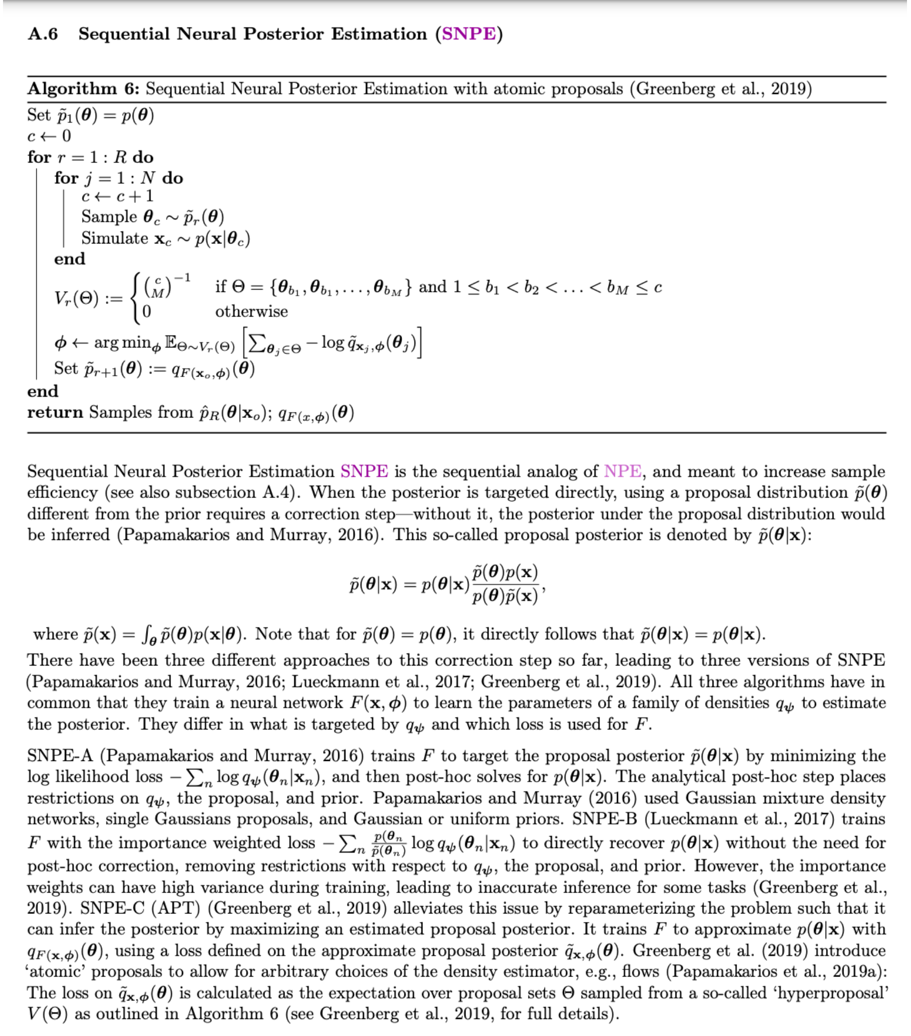
Algorithm 5: Single round Neural Posterior Estimation as in Papamakarios and Murray (2016)

```
for $j = 1 : N$ do
    Sample $\boldsymbol{\theta}_j \sim p(\boldsymbol{\theta})$
    Simulate $\mathbf{x}_j \sim p(\mathbf{x}|\boldsymbol{\theta}_j)$
end
$\boldsymbol{\phi} \leftarrow \arg\min \sum_j^N -\log q_{F(\mathbf{x}_j,\boldsymbol{\phi})}(\boldsymbol{\theta}_j)$
Set $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o) = q_{F(\mathbf{x}_o,\boldsymbol{\phi})}(\boldsymbol{\theta})$
return Samples from $\hat{p}(\boldsymbol{\theta}|\mathbf{x}_o)$; $q_{F(\mathbf{x},\boldsymbol{\phi})}(\boldsymbol{\theta})$
```

NPE uses conditional density estimation to directly estimate the posterior. This idea dates back to regression adjustment approaches (Blum and François, 2010) and was extended to density estimators using neural networks (Papamakarios and Murray, 2016) more recently.

As outlined in Algorithm 5, the approach is as follows: Given a prior over parameters $p(\boldsymbol{\theta})$ and a simulator, a set of training data points $(\boldsymbol{\theta}, \mathbf{x})$ is generated. This training data is used to learn the parameters $\boldsymbol{\psi}$ of a conditional density estimator $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|x)$ using a neural network $F(\mathbf{x}, \boldsymbol{\phi})$, i.e., $\boldsymbol{\psi} = F(\mathbf{x}, \boldsymbol{\phi})$. The loss function is given by the negative log probability $-\log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|x)$. If the density estimator q is flexible enough and training data is infinite, this loss function leads to perfect recovery of the ground-truth posterior (Papamakarios and Murray, 2016).

For the benchmark, we used the approach by Papamakarios and Murray (2016) with a Neural Spline Flow (NSF, Durkan et al., 2019) as density estimator, using five flow transforms, two residual blocks of 50 hidden units each, ReLU non-linearity, and 10 bins. We sampled 10k samples from the approximate posterior $q_{F(\mathbf{x}_o,\boldsymbol{\phi})}(\boldsymbol{\theta})$. In Appendix H, we compare NSFs to Masked Autoregressive Flows (MAFs, Papamakarios et al., 2017), as used in Greenberg et al. (2019); Durkan et al. (2020), with five flow transforms, each with two blocks and 50 hidden units, tanh non-linearity and batch normalization after each layer.



A.6 Sequential Neural Posterior Estimation (SNPE)

Algorithm 6: Sequential Neural Posterior Estimation with atomic proposals (Greenberg et al., 2019)

```
Set $\tilde{p}_1(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$
$c \leftarrow 0$
for $r = 1 : R$ do
    for $j = 1 : N$ do
        $c \leftarrow c + 1$
        Sample $\boldsymbol{\theta}_c \sim \tilde{p}_r(\boldsymbol{\theta})$
        Simulate $\mathbf{x}_c \sim p(\mathbf{x}|\boldsymbol{\theta}_c)$
    end
    $V_r(\Theta) := \begin{cases} \binom{c}{M}^{-1} & \text{if } \Theta = \{\boldsymbol{\theta}_{b_1}, \boldsymbol{\theta}_{b_1}, \ldots, \boldsymbol{\theta}_{b_M}\} \text{ and } 1 \le b_1 < b_2 < \ldots < b_M \le c \\ 0 & \text{otherwise} \end{cases}$
    $\boldsymbol{\phi} \leftarrow \arg\min_{\boldsymbol{\phi}} \mathbb{E}_{\Theta\sim V_r(\Theta)} \left[\sum_{\boldsymbol{\theta}_j\in\Theta} -\log \tilde{q}_{\mathbf{x}_j,\boldsymbol{\phi}}(\boldsymbol{\theta}_j)\right]$
    Set $\tilde{p}_{r+1}(\boldsymbol{\theta}) := q_{F(\mathbf{x}_o,\boldsymbol{\phi})}(\boldsymbol{\theta})$
end
return Samples from $\hat{p}_R(\boldsymbol{\theta}|\mathbf{x}_o)$; $q_{F(x,\boldsymbol{\phi})}(\boldsymbol{\theta})$
```

Sequential Neural Posterior Estimation SNPE is the sequential analog of NPE, and meant to increase sample efficiency (see also subsection A.4). When the posterior is targeted directly, using a proposal distribution $\tilde{p}(\boldsymbol{\theta})$ different from the prior requires a correction step—without it, the posterior under the proposal distribution would be inferred (Papamakarios and Murray, 2016). This so-called proposal posterior is denoted by $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$:

$$\tilde{p}(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x})\frac{\tilde{p}(\boldsymbol{\theta})p(\mathbf{x})}{p(\boldsymbol{\theta})\tilde{p}(\mathbf{x})},$$

where $\tilde{p}(\mathbf{x}) = \int_{\boldsymbol{\theta}} \tilde{p}(\boldsymbol{\theta})p(\mathbf{x}|\boldsymbol{\theta})$. Note that for $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$, it directly follows that $\tilde{p}(\boldsymbol{\theta}|\mathbf{x}) = p(\boldsymbol{\theta}|\mathbf{x})$.

There have been three different approaches to this correction step so far, leading to three versions of SNPE (Papamakarios and Murray, 2016; Lueckmann et al., 2017; Greenberg et al., 2019). All three algorithms have in common that they train a neural network $F(\mathbf{x}, \boldsymbol{\phi})$ to learn the parameters of a family of densities $q_{\boldsymbol{\psi}}$ to estimate the posterior. They differ in what is targeted by $q_{\boldsymbol{\psi}}$ and which loss is used for F.

SNPE-A (Papamakarios and Murray, 2016) trains F to target the proposal posterior $\tilde{p}(\boldsymbol{\theta}|\mathbf{x})$ by minimizing the log likelihood loss $-\sum_n \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}_n|\mathbf{x}_n)$, and then post-hoc solves for $p(\boldsymbol{\theta}|\mathbf{x})$. The analytical post-hoc step places restrictions on $q_{\boldsymbol{\psi}}$, the proposal, and prior. Papamakarios and Murray (2016) used Gaussian mixture density networks, single Gaussians proposals, and Gaussian or uniform priors. SNPE-B (Lueckmann et al., 2017) trains F with the importance weighted loss $-\sum_n \frac{p(\boldsymbol{\theta}_n}{\tilde{p}(\boldsymbol{\theta}_n)} \log q_{\boldsymbol{\psi}}(\boldsymbol{\theta}_n|\mathbf{x}_n)$ to directly recover $p(\boldsymbol{\theta}|\mathbf{x})$ without the need for post-hoc correction, removing restrictions with respect to $q_{\boldsymbol{\psi}}$, the proposal, and prior. However, the importance weights can have high variance during training, leading to inaccurate inference for some tasks (Greenberg et al., 2019). SNPE-C (APT) (Greenberg et al., 2019) alleviates this issue by reparameterizing the problem such that it can infer the posterior by maximizing an estimated proposal posterior. It trains F to approximate $p(\boldsymbol{\theta}|\mathbf{x})$ with $q_{F(\mathbf{x},\boldsymbol{\phi})}(\boldsymbol{\theta})$, using a loss defined on the approximate proposal posterior $\tilde{q}_{\mathbf{x},\boldsymbol{\phi}}(\boldsymbol{\theta})$. Greenberg et al. (2019) introduce 'atomic' proposals to allow for arbitrary choices of the density estimator, e.g., flows (Papamakarios et al., 2019a): The loss on $\tilde{q}_{\mathbf{x},\boldsymbol{\phi}}(\boldsymbol{\theta})$ is calculated as the expectation over proposal sets Θ sampled from a so-called 'hyperproposal' $V(\Theta)$ as outlined in Algorithm 6 (see Greenberg et al., 2019, for full details).