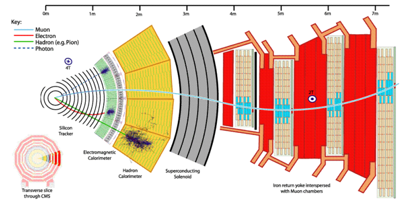
Simulation-Based Inference II

Michael Kagan
SLAC

July 20, 2023

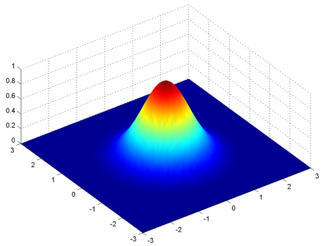
Last Time

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



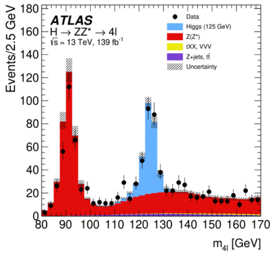
Simulator / Forward model

$\pi(\theta)$



Proposal / Prior distribution

y



Measured data

Likelihood ratio estimation

$$r_\phi(x|\theta,\theta') \approx \frac{p(x|\theta)}{p(x|\theta')}$$

Likelihood estimation

$$q_\phi(x|\theta) \approx p(x|\theta)$$

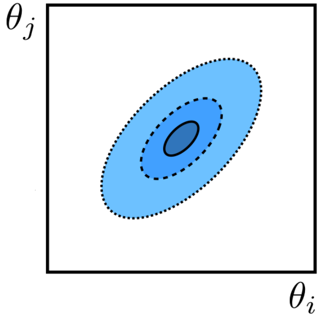
Posterior estimation

$$q_\phi(\theta|x) \approx p(\theta|x) \propto p(x|\theta)\pi(\theta)$$

Neural Net

Evaluate model on measured data

Posterior / Confidence intervals

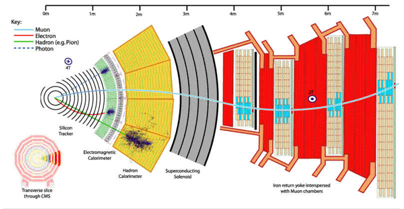


Slide adapted from J. Macke

Review: K. Cranmer, J. Brehmer, G. Louppe, PNAS (2020)

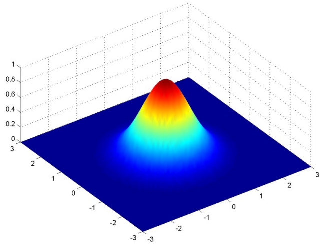
Last Time

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



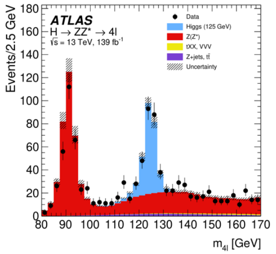
Simulator / Forward model

$\pi(\theta)$



Proposal / Prior distribution

y



Measured data

Likelihood ratio estimation

$$r_\phi(x|\theta,\theta') \approx \frac{p(x|\theta)}{p(x|\theta')}$$

Likelihood estimation

$$q_\phi(x|\theta) \approx p(x|\theta)$$

Posterior estimation

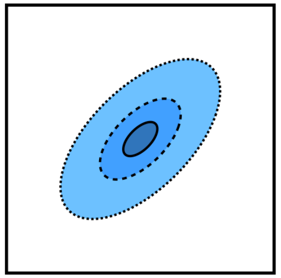
$$q_\phi(\theta|x) \approx p(\theta|x) \propto p(x|\theta)\pi(\theta)$$

Neural Density Estimation

Evaluate model on measured data

Posterior / Confidence intervals

θ_j



θ_i

Slide adapted from J. Macke

Review: K. Cranmer, J. Brehmer, G. Louppe, PNAS (2020)

Density Estimation is Hard in High Dimensions!

Estimating likelihood or posterior in high dimensions is hard!

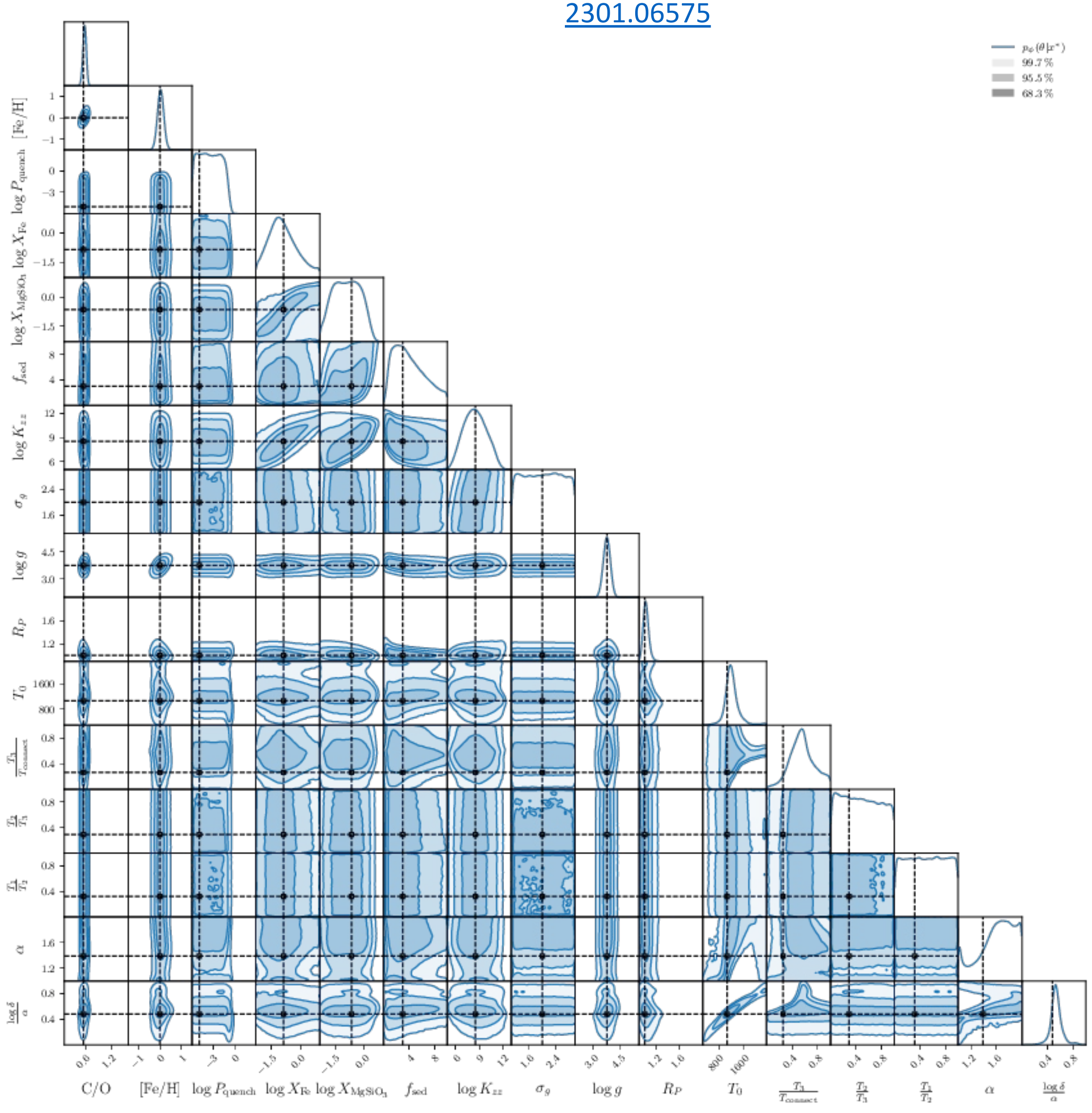
Solution 1:

- Learn summaries $s(x)$ instead of x directly

Solution 2:

- Don’t learn densities

2301.06575



Neural Ratio Estimation

Likelihood Ratio Estimation

Instead of estimating densities, a popular approach is density ratio estimation

$$\frac{p(x|\theta)}{p(x)}, \qquad \frac{p(x|\theta_0)}{p(x|\theta_1)}, \qquad \frac{p(x|\theta)}{p(x|\theta_0)}$$

Why? In many cases, don't need normalized density.

Likelihood Ratio Estimation

Instead of estimating densities, a popular approach is density ratio estimation

$$\frac{p(x|\theta)}{p(x)}, \qquad \frac{p(x|\theta_0)}{p(x|\theta_1)}, \qquad \frac{p(x|\theta)}{p(x|\theta_0)}$$

More importantly… We know the most powerful summary statistic to decide between two (simple) hypotheses due to the **Neyman-Pearson Lemma**

It's the Likelihood ratio:

$$t(x) = \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Likelihood Ratio Estimation

Instead of estimating densities, a popular approach is density ratio estimation

$$\frac{p(x|\theta)}{p(x)}, \qquad \frac{p(x|\theta_0)}{p(x|\theta_1)}, \qquad \frac{p(x|\theta)}{p(x|\theta_0)}$$

More importantly… We know the most powerful summary statistic to decide between two (simple) hypotheses due to the **Neyman-Pearson Lemma**

It's the Likelihood ratio:

$$t(x) = \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Intriguingly, we can estimate this ratio without knowing $p(x|\theta)$ explicitly!

Likelihood Ratio Trick

Given data x from two classes / hypotheses / θ's: we assign labels $y = \{0, 1\}$

Sufficiently powerful classifier, $f(x)$, trained sufficiently well will approximate

$$f(x) \approx \frac{1}{1+r^{-1}(x)}$$

- where $r(x) = \frac{p(x|y=1)}{p(x|y=0)}$ is the likelihood ratio

Equivalently: $$r(x) \approx \frac{f(x)}{1-f(x)}$$

Rough Derivation

Binary classification problem in ML: Minimize Binary Cross Entropy Loss

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} y_i \log f_w(x_i) + (1 - y_i) \log\big(1 - f_w(x_i)\big)$$

Want to minimize loss function over dataset w.r.t. model parameters

Rough Derivation

Binary classification problem in ML: Minimize Binary Cross Entropy Loss

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} y_i \log f_w(x_i) + (1 - y_i) \log\big(1 - f_w(x_i)\big)$$

What are we really doing here?

Using an empirical average:

$$\int dxdy\, p(x, y) \rightarrow \frac{1}{N} \Sigma_{i=1}^{N} \quad \text{with samples } \{x_i, y_i\} \sim p(x, y)$$

Parameterizing (and thereby restricting) a class of functions:

$$\text{All } f(\cdot) \rightarrow \ \{f_w(\cdot)\, ;\ w \in R^k\}$$

Rough Derivation

Ideally, we would minimize loss function over dataset w.r.t. model $f(\cdot)$

$$f^*(x) = \arg\min_f \mathbb{E}[L(f(x), y)]$$

$$= \arg\min_f \int p(x, y)\big[y \log f(x) + (1 - y)\log\big(1 - f(x)\big)\big]dxdy$$

Rough Derivation

Ideally, we would minimize loss function over dataset w.r.t. model $f(\cdot)$

$$f^*(x) = \arg\min_f \mathbb{E}[L(f(x), y)]$$

$$= \arg\min_f \int p(x,y)\big[y \log f(x) + (1-y)\log\big(1 - f(x)\big)\big]dxdy$$

We can try to solve these kinds of problems
using Calculus of Variations!

Rough Derivation

Ideally, we would minimize loss function over dataset w.r.t. model $f(\cdot)$

$$f^*(x) = \arg\min_f \mathbb{E}[L(f(x), y)]$$

$$= \arg\min_f \int p(x, y)\big[y \log f(x) + (1 - y) \log\big(1 - f(x)\big)\big] dx dy$$

Take functional derivative $\frac{\delta}{\delta f}$ and set to zero (minimum), we find:

$$\boldsymbol{f^*(x) = p(y = 1|x)}$$

Rough Derivation

In the infinite statistics limit, the solution to a binary classification problem:

$$f^*(x) = p(y=1|x)$$

Posterior!

$$= \frac{p(x|y=1)p(y=1)}{p(x)}$$

Bayes Rule

$$= \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1) + p(x|y=0)p(y=0)}$$

Expand $p(x)$

$$= \frac{1}{1 + \frac{p(x|y=0)}{p(x|y=1)}\frac{p(y=0)}{p(y=1)}}$$

Rough Derivation

Assuming equal marginal class probabilities $p(y = 1) = p(y = 0) = 0.5$

$$f^*(x) = \frac{1}{1 + \frac{p(x|y=0)}{p(x|y=1)}}$$

Likelihood ratio!

$$= \frac{1}{1 + e^{-\ln r(x)}}$$

With $r(x) = \frac{p(x|y=1)}{p(x|y=0)}$

$$= \sigma(\ln r(x))$$

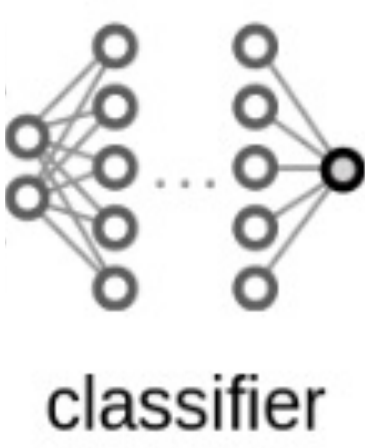
Log-Likelihoods are the logits of the classifier

Practical note

We found the optimal classifier solution: $f(x) = \sigma(\ln r(x))$

Typical neural network classifier has sigmoid as last computation to estimate class probability:

$$f(x) = \sigma\big(z = NNLayers(x)\big)$$



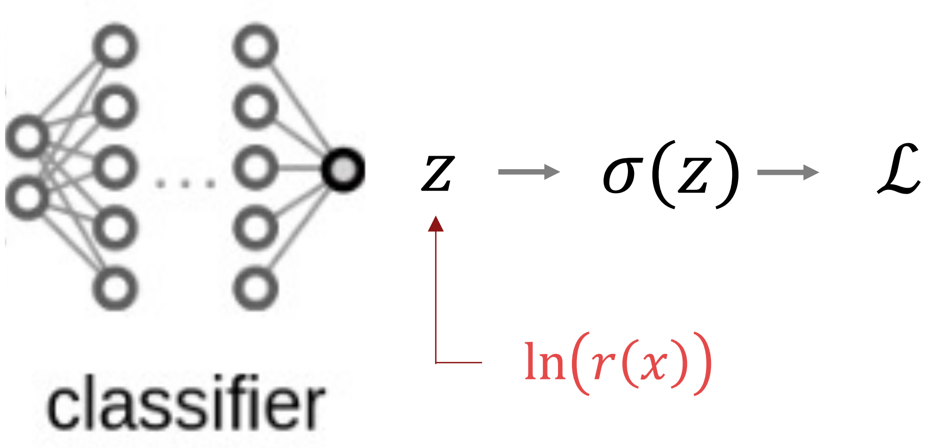
$z \rightarrow \sigma(z) \rightarrow \mathcal{L}$

Practical note

We found the optimal classifier solution: $f(x) = \sigma(\ln r(x))$

Typical neural network classifier has sigmoid as last computation to estimate class probability:

$$f(x) = \sigma\big(z = NNLayers(x)\big)$$



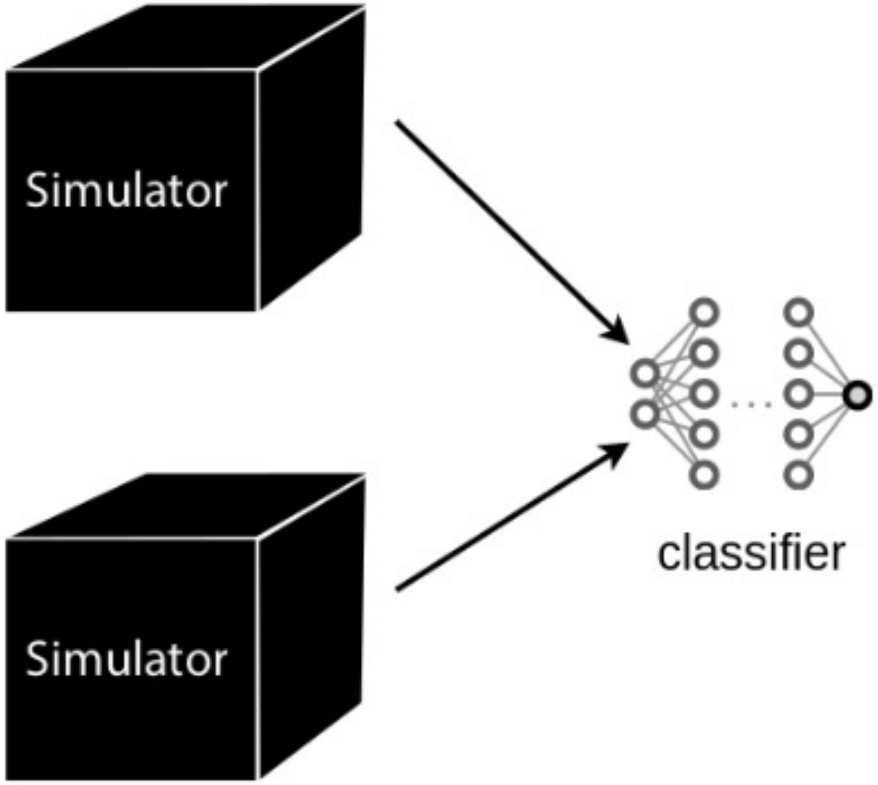
The input to the networks last sigmoid layer is the log-likelihood-ratio

More numerically stable to extract these *logit* values, than to compute ratios of classifier outputs

Two-Classes

$x \sim p(x|\theta_1)$

$x \sim p(x|\theta_0)$



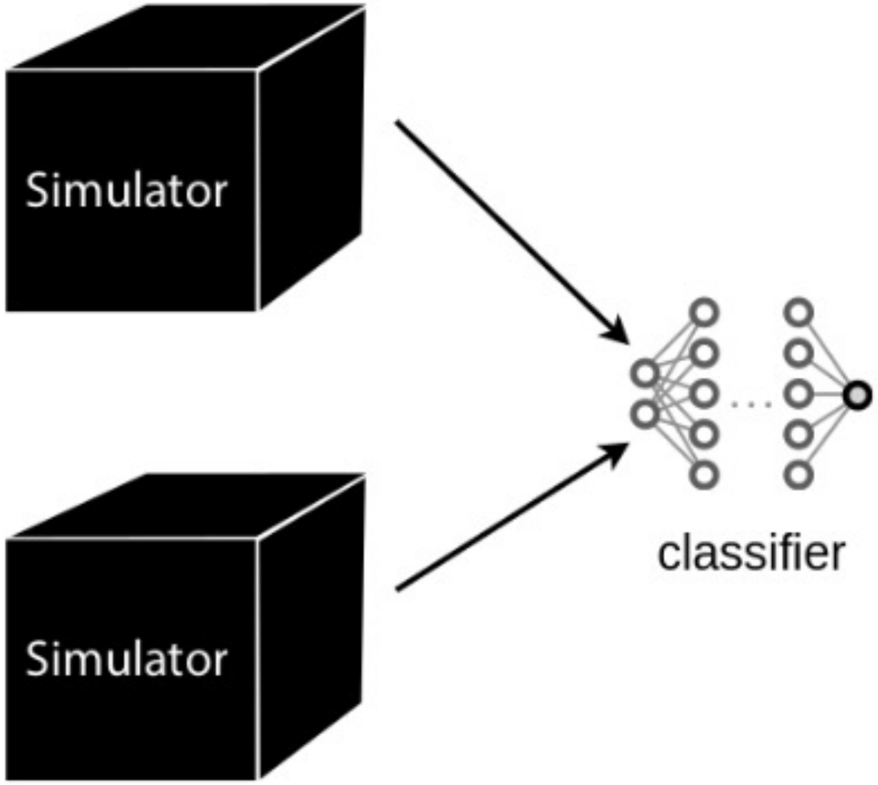
$$\hat{r}(x) \approx \frac{p(x|\theta_1)}{p(x|\theta_0)}$$

What if we want to test multiple parameters? Amortized Inference

1. Proposal distribution $\pi(\theta)$
2. Sample parameters $\theta \sim \pi(\theta)$
3. Sample batch of events $x \sim p(x|\theta)$
4. Train classifier on batch, repeat

$\theta \sim \pi(\theta)$
$x \sim p(x|\theta)$

θ_0
$x \sim p(x|\theta_0)$

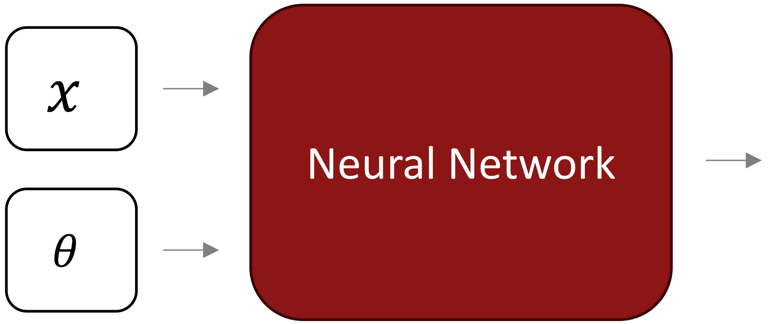


$$\hat{r}(x|\theta) \approx \frac{p(x|\theta)}{p(x|\theta_0)}$$

Amortized Inference

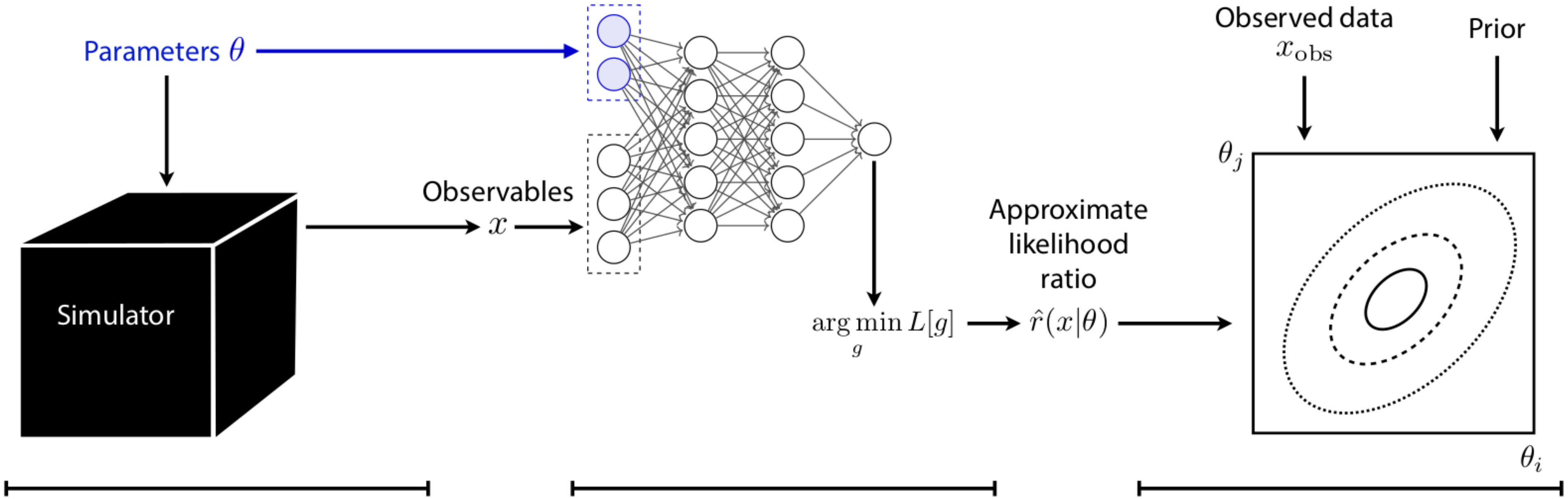
Now we have a **parameterized neural network**

Input θ tells network which classification / ratio estimation problem to solve



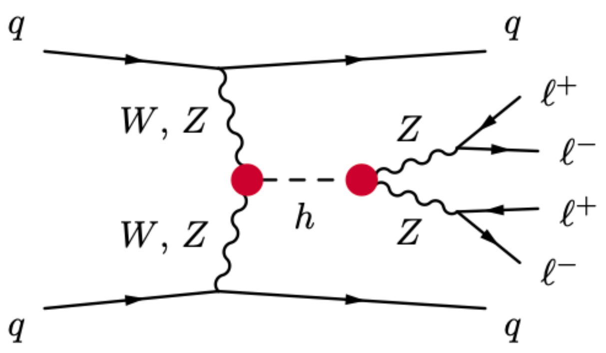
$$\hat{r}(x|\theta) \approx \frac{p(x|\theta)}{p(x|\theta_0)}$$

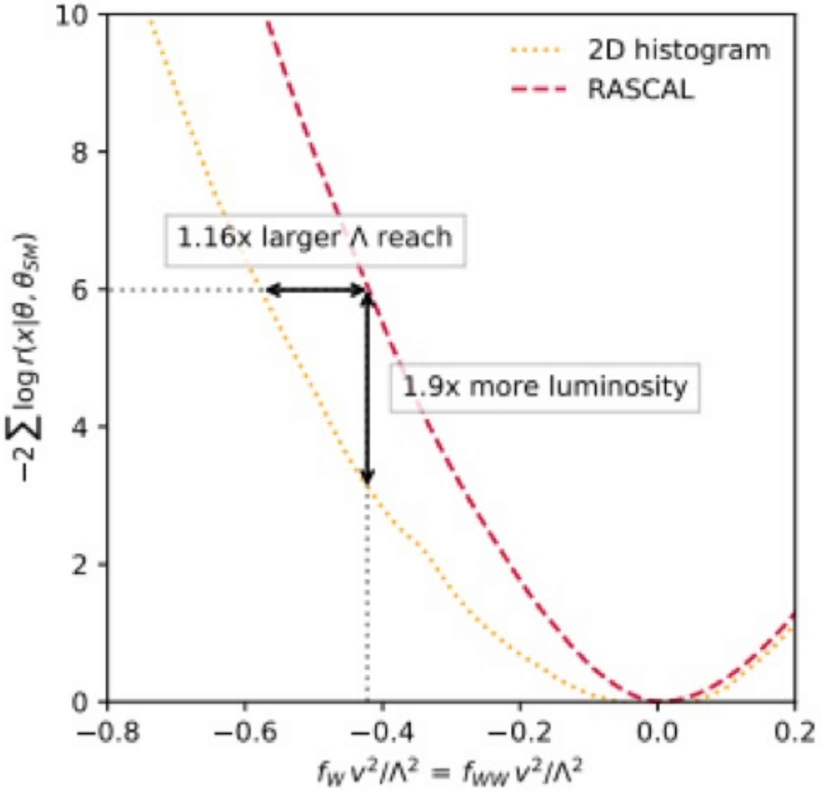
Amortized Inference

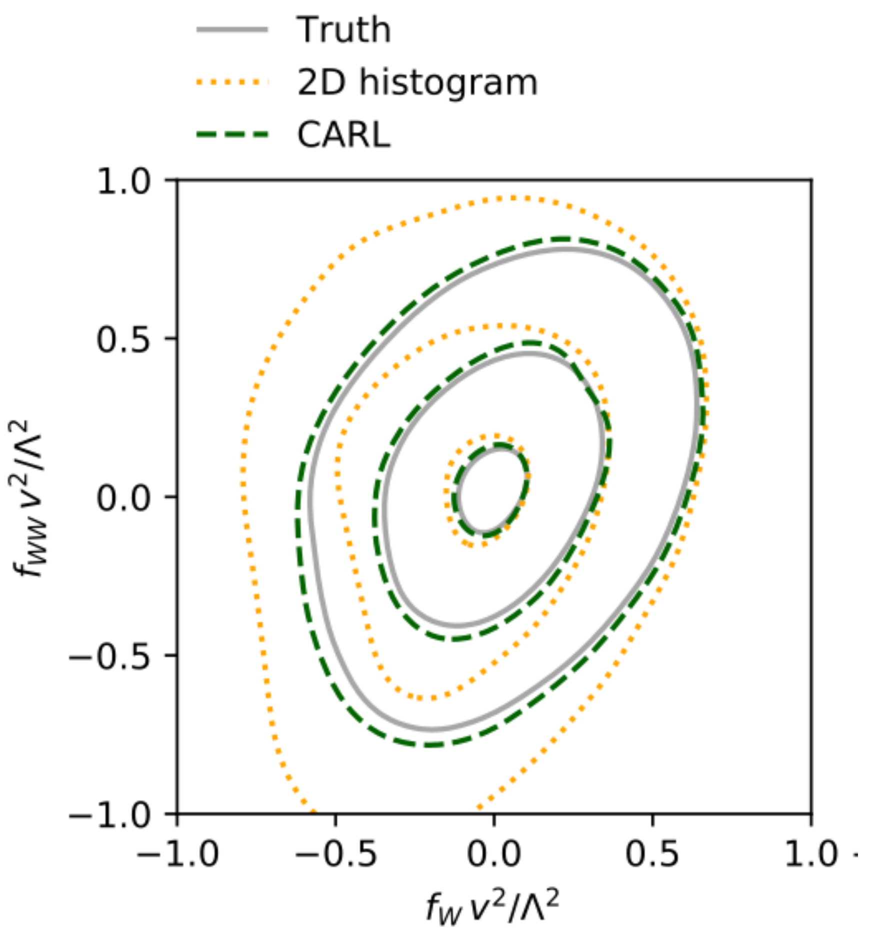


1. Simulation	2. Machine Learning	3. Inference
Run simulator and save data	Train NN classifier, interpret as likelihood ratio estimator	Amortized: cheap to repeat for new data

LHC Example







arXiv:1805.00020

Neural Ratio Estimation for Bayesian Inference

Density ratio estimation also works well for Bayesian Inference

$$p(\theta|x) = \frac{p(x|\theta)}{p(x)} p(\theta)$$

Neural Ratio Estimation for Bayesian Inference

Density ratio estimation also works well for Bayesian Inference

$$p(\theta|x) = \frac{p(x|\theta)\color{red}{p(\theta)}}{p(x)\color{red}{p(\theta)}}p(\theta)$$

Neural Ratio Estimation for Bayesian Inference

Density ratio estimation also works well for Bayesian Inference

Joint

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

Marginals

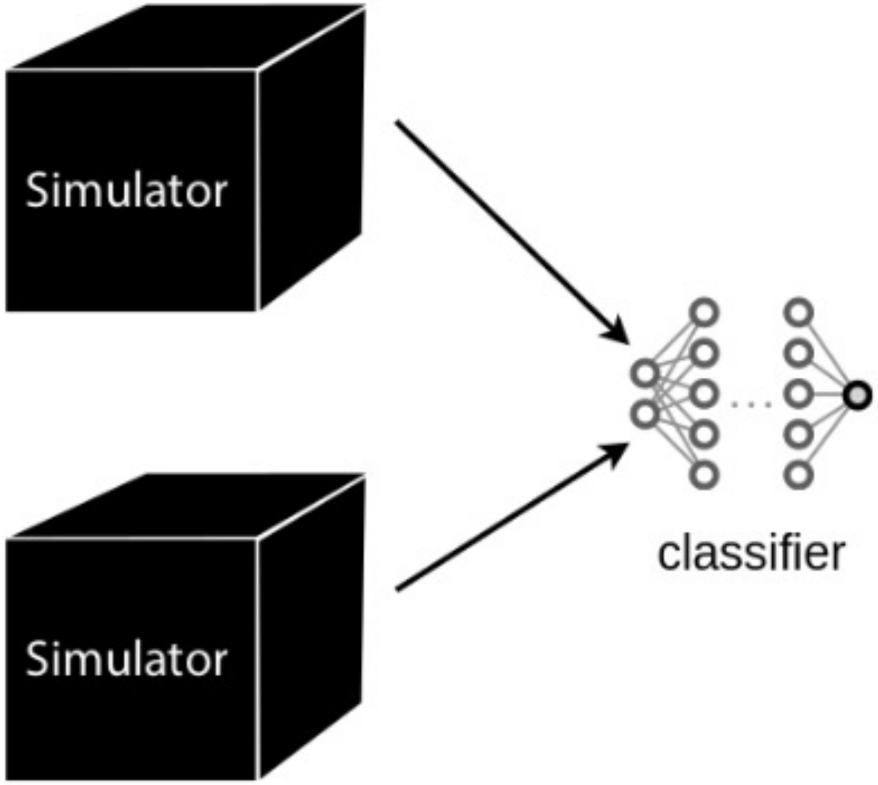
Neural Ratio Estimation for Bayesian Inference

Joint

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

$$x \sim p(x)$$
$$\theta \sim p(\theta)$$

Marginals



$$\hat{r}(x|\theta) \approx \frac{p(x|\theta)}{p(x)}$$

$$\hat{p}(\theta|x) = \hat{r}(x|\theta)p(\theta)$$
$$\approx p(\theta|x)$$

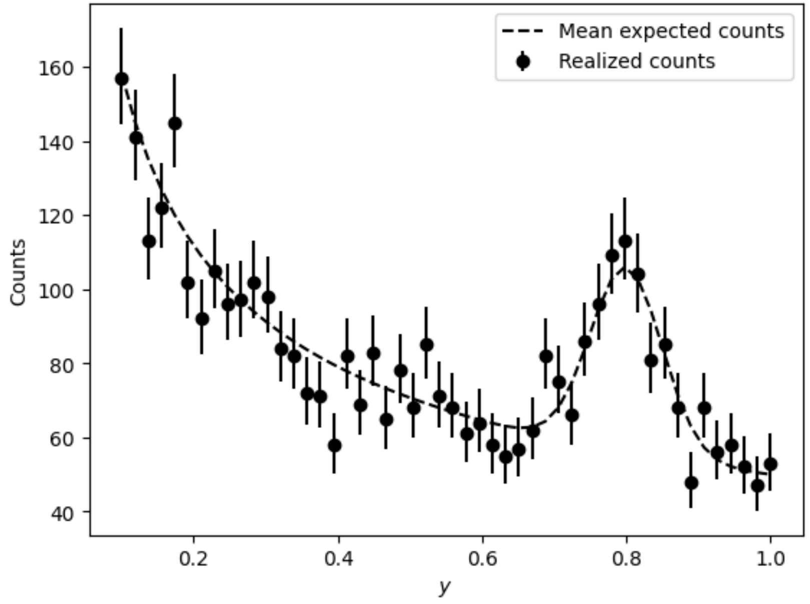
Using Neural Ratio Estimation (NRE)

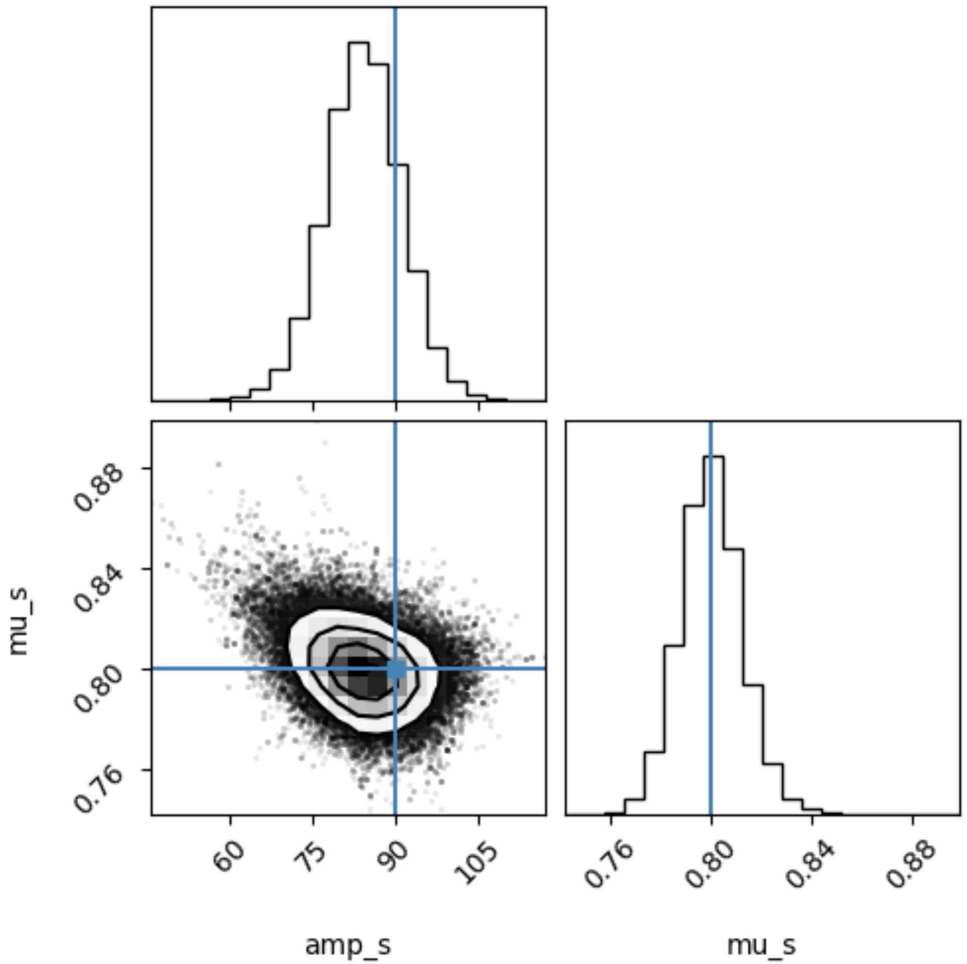
Once trained, can sample the posterior $\hat{p}(\theta|x) = \hat{r}(x|\theta)p(\theta)$ with MCMC

$$x_b = 50y^{-0.5}$$

$$x_s = A_s\, e^{-\frac{(y-\mu_s)^2}{2(0.05)^2}}$$

$$x \sim p(x|A_s, \mu_s) = Pois(x_b + x_s)$$





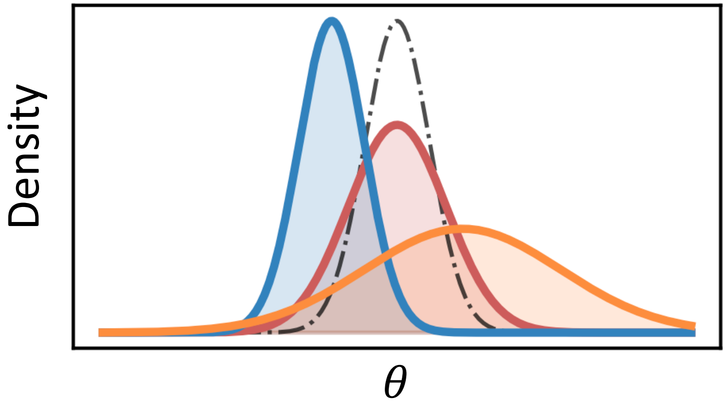
Coverage Diagnostics

Test if the posterior predicted intervals match the simulator

Simulated samples $x, \theta \sim p(x, \theta)$

Compute 1D quantiles or credible intervals of approx. posterior $\hat{p}(\theta | x)$ by sampling posterior

Empirical coverage is the fraction of samples of true θ that is contained in the interval



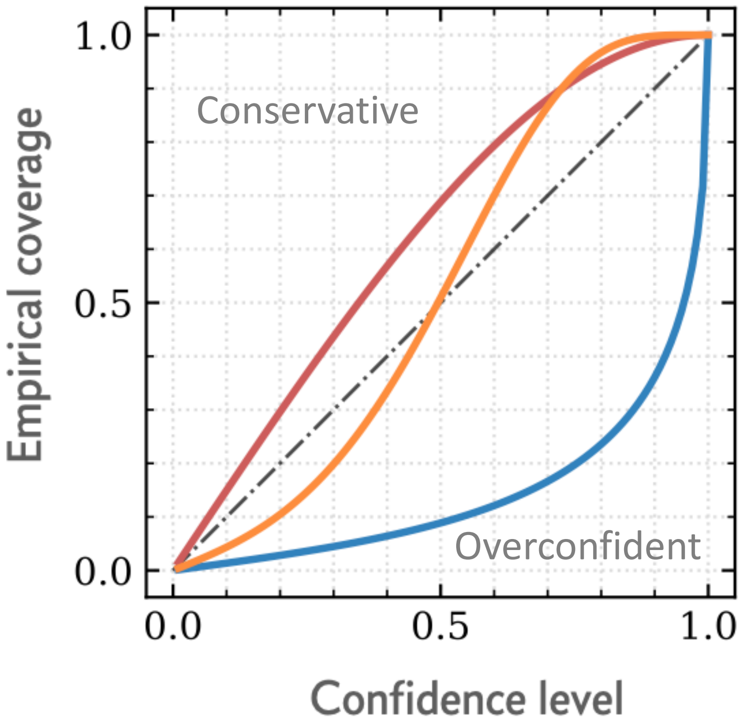
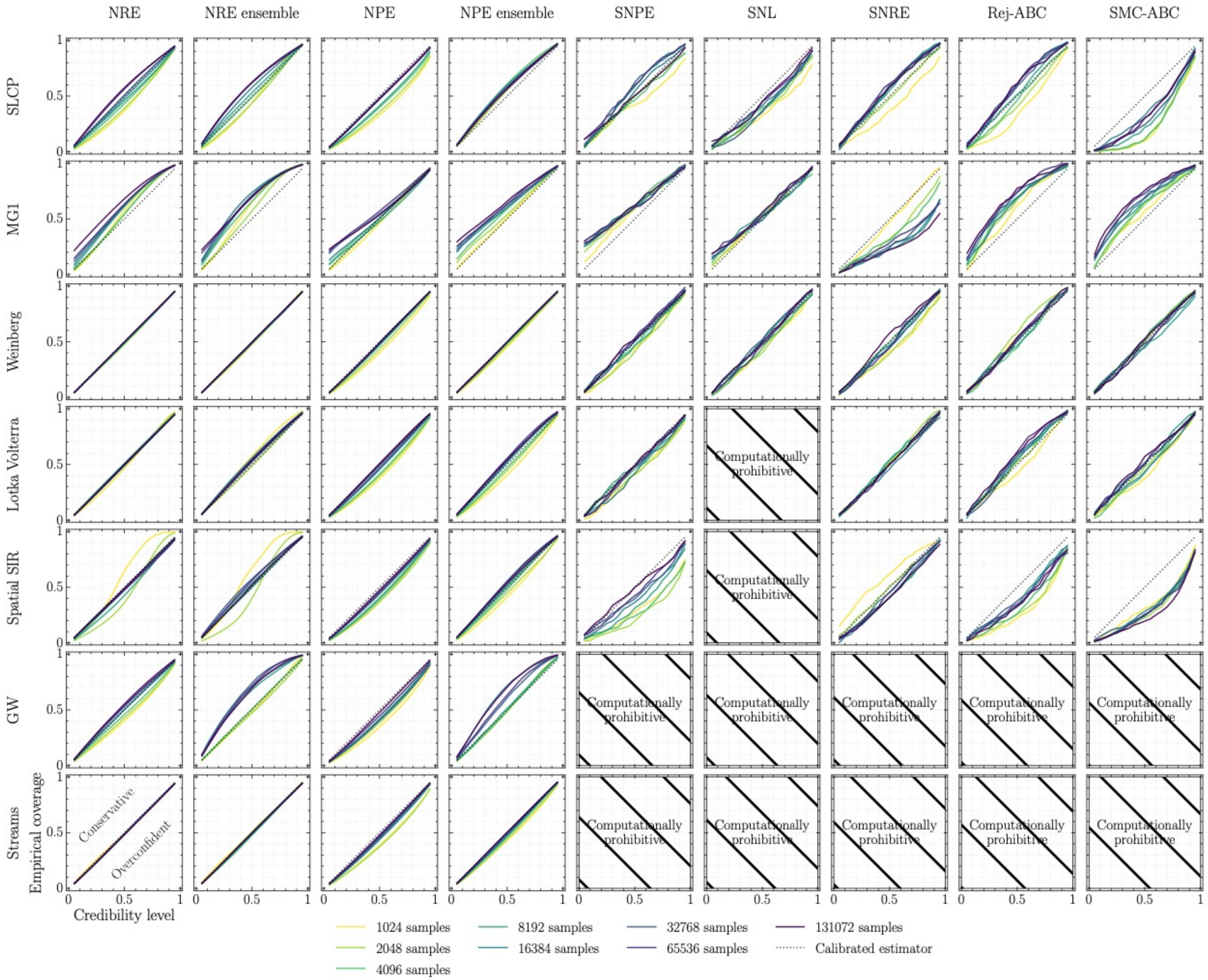


Figure credit: [arXiv:2209.01845](https://arxiv.org/abs/2209.01845)

Coverage Comparisons



arXiv:2110.06581

Sometimes, we can do more with simulators…

Getting more from Simulators

The likelihood ratio trick → Estimate density ratios from samples

Had to do this because the likelihood is intractable:

$$p(x|\theta) = \int dz\, p(x, z|\theta)$$

Why is this intractable? → *Often its because of the integral*

Example

$x \sim p(x) = \mathcal{N}(0, 1)$

$\mu_{y|x} = 4 - x^2$

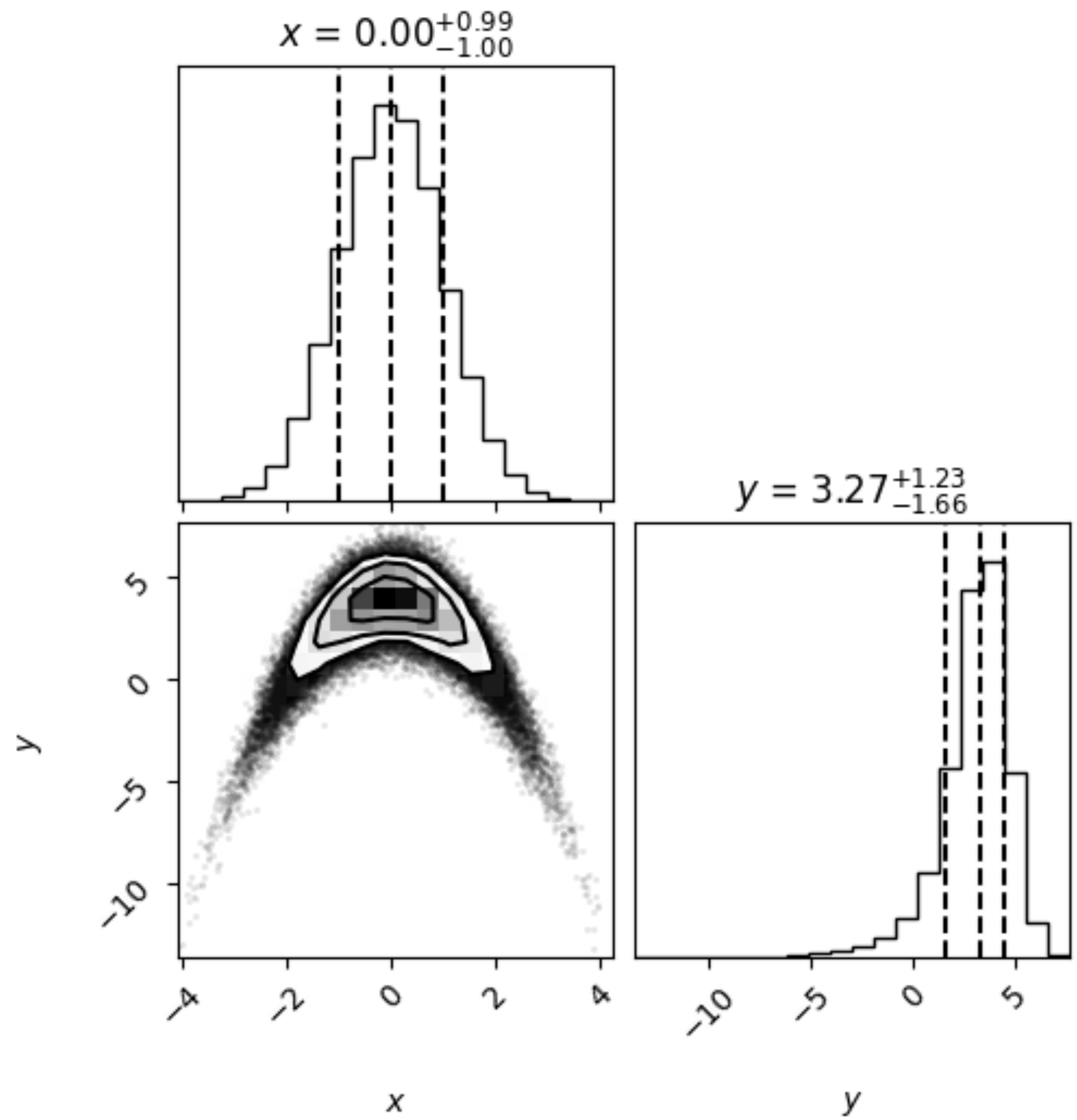
$y \sim p(y|x) = \mathcal{N}(\mu_{y|x}, 1)$

Joint:

$p(y, x) = \mathcal{N}(y \mid \mu_{y|x}, 1)\mathcal{N}(x \mid 0{,}1)$

Marginal?

$p(y) = \int p(y|x)p(x)dx$



The Joint Density

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

Often we know the joint → it's what we implemented in code!

If we keep track of all the random variables we sampled and their distribution, we can evaluate the probability of a simulation run

We can actually use these *joint densities as labels for training*

Regression Trick

What model do we learn from MSE regression?

$$f^*(x) = \arg\min_{\hat{f}} \mathbb{E}_{p(x,y)}\left[\left(y - \hat{f}(x)\right)^2\right]$$

Regression Trick

What model do we learn from MSE regression?

$$f^*(x) = \arg\min_{\hat{f}} \mathbb{E}_{p(x,y)}\left[\left(y - \hat{f}(x)\right)^2\right]$$

Same as before: Calculus of variations

$$\frac{\delta}{\delta \hat{f}} \int dxdyp(y|x)p(x)\left[\left(y - \hat{f}(x)\right)^2\right] = 0$$

Solution:

$$f^*(x) = \mathbb{E}_{p(y|x)}[y]$$

Solution to a regression is an expected value over the conditional distribution

Regression Trick

Generalizing that result

$$f^*(x) = \arg\min_{\hat{f}} \mathbb{E}_{p(x,z|\theta)}\left[\left(f(x,z) - \hat{f}(x)\right)^2\right]$$

$$= \mathbb{E}_{p(z|x,\theta)}[f(x,z)]$$

When z is a latent random variable,

Regression trick enables us to marginalize over the latent variable

What if $f(\cdot)$ is the joint likelihood ratio?

Let

$$f(x, z) = r(x, z|\theta_0, \theta_1) = \frac{p(x,z|\theta_0)}{p(x,z|\theta_1)}$$

Then

$$r^*(x) = \arg\min_{\hat{r}} \mathbb{E}_{p(x,z|\theta_1)}\left[\left(r(x, z|\theta_0, \theta_1) - \hat{r}(x)\right)^2\right]$$

$$= \mathbb{E}_{p(z|x,\theta_1)}[r(x, z|\theta_0, \theta_1)]$$

OK... but what is this???

What if $f(\cdot)$ is the joint likelihood ratio?

$$r^*(x) = \mathbb{E}_{p(z|x,\theta_1)}[r(x,z|\theta_0,\theta_1)]$$

$$= \int dz\, p(z|x,\theta_1) r(x,z|\theta_0,\theta_1)$$

Conditional definition

$$= \int dz \frac{\cancel{p(x,z|\theta_1)}}{p(x|\theta_1)} \frac{p(x,z|\theta_0)}{\cancel{p(x,z|\theta_1)}}$$

$$= \frac{1}{p(x|\theta_1)} \int dz\, p(x,z|\theta_0)$$

Marginal definition

$$= \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Marginal Likelihood Ratio
→ No latents!

Regression Trick for Score

This trick also works for the score: $t(x|\theta) = \nabla_\theta \log p(x|\theta)$

$$t^*(x|\theta) = \arg\min_{\hat{t}} \mathbb{E}_{p(x,z|\theta)}\left[\left(t(x,z|\theta) - \hat{t}(x|\theta)\right)^2\right]$$

$$= \mathbb{E}_{p(z|x,\theta)}[t(x,z|\theta)]$$

$$= t(x|\theta)$$

Regressing on the join score allows use to “marginalize out” latents

And estimate marginal likelihood score!

What’s going on here?

Instead of classifying samples to get learn likelihood ratio

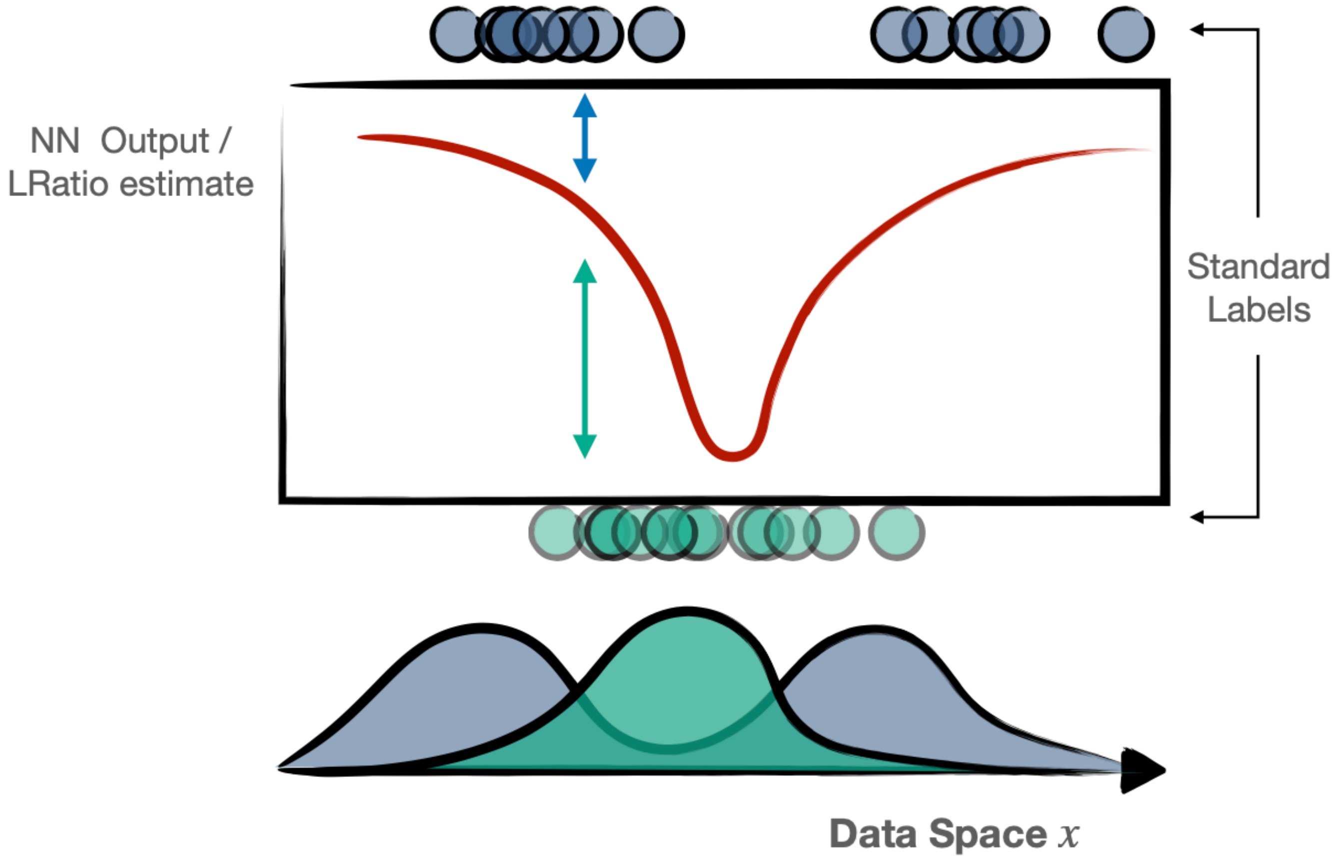


Figure credit: L. Heinrich

What's going on here?

Instead of classifying samples to get learn likelihood ratio

We can use the join density as training targets

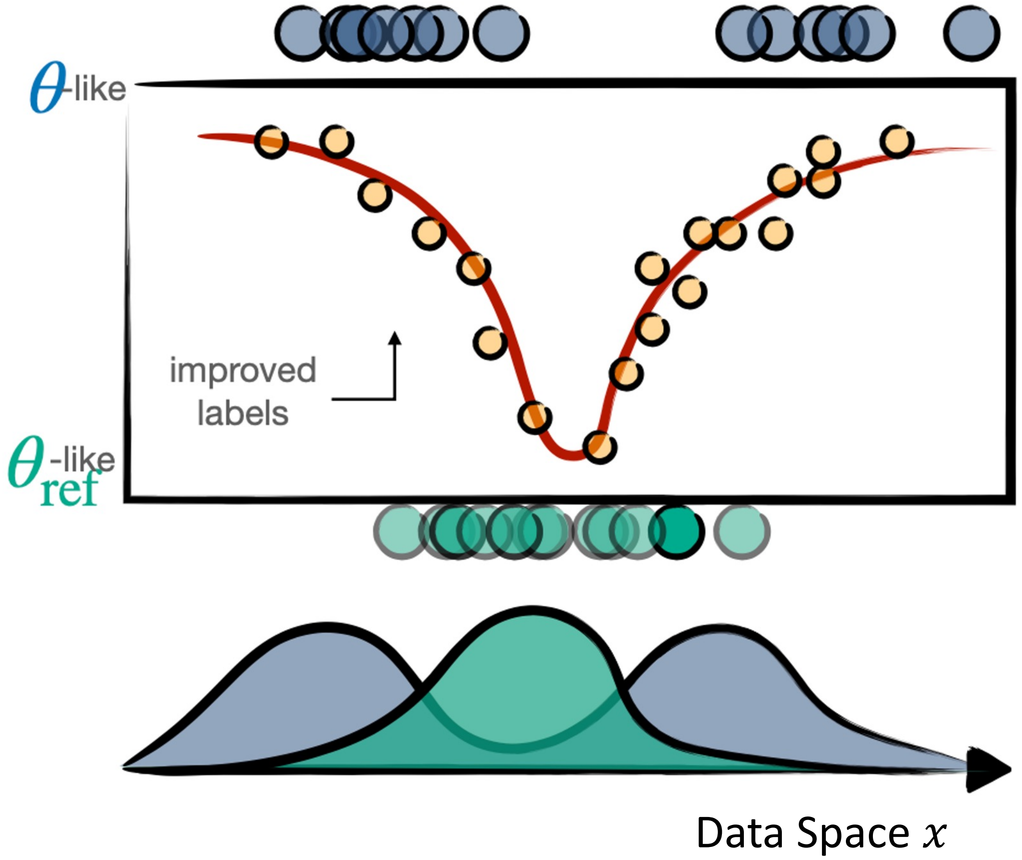


Figure credit: L. Heinrich

What’s going on here?

Instead of classifying samples to get learn likelihood ratio

We can use the join density as training targets

These are noisy targets, since they jump around due to z in $p(x,z|\theta)$

The regression will average over this “noise” to get marginal

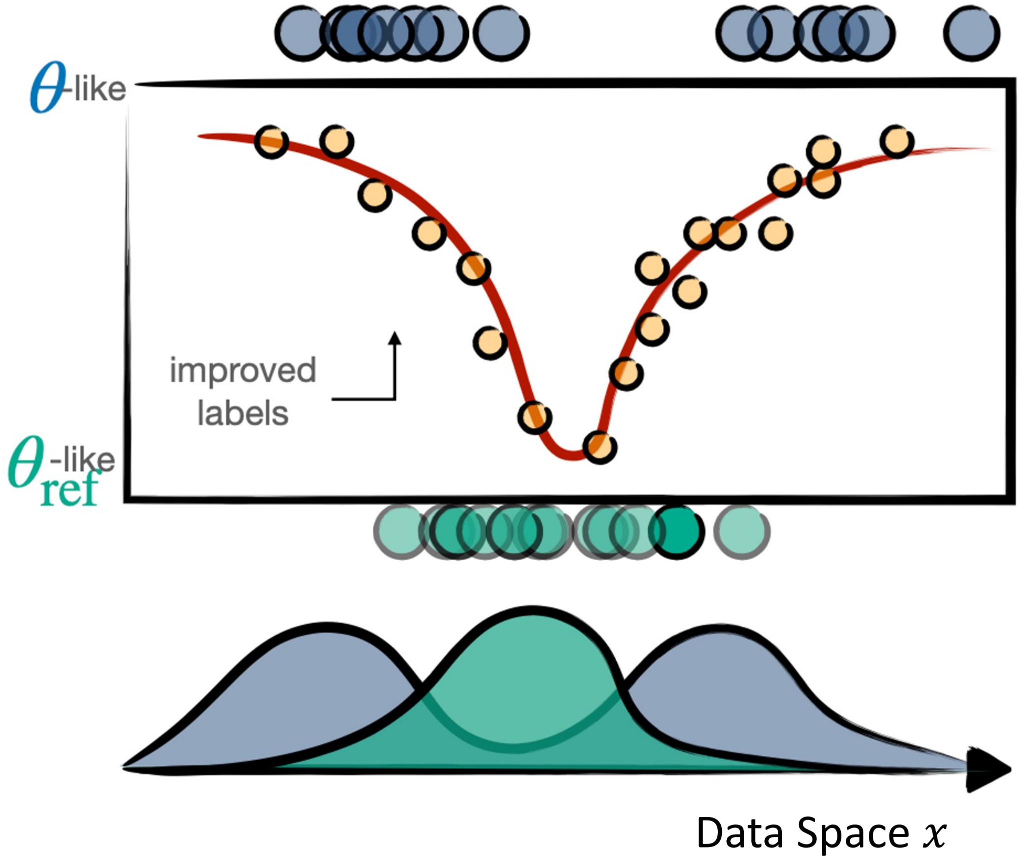


Figure credit: L. Heinrich

What about the Gradients?

The gradients give us higher order information about for training...

i.e. the slope of the density w.r.t parameters

This is more information, from each data point, to guide learning

Additional labels for training!

Compare with grad or ratio estimate

$$\hat{t}(x|\theta_0) = \nabla_\theta \log \hat{r}(x|\theta, \theta_1)\Big|_{\theta_1}$$

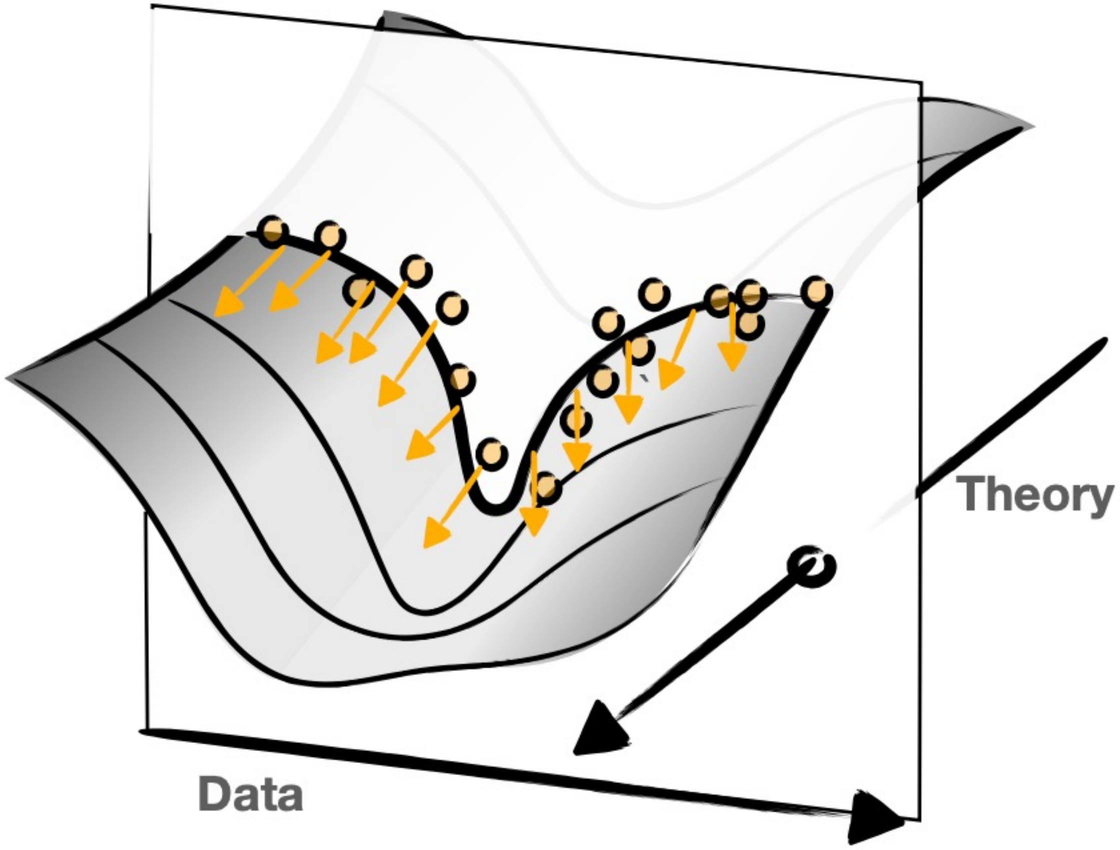


Figure credit: L. Heinrich

Ratio + Score Regression (RASCAL) Loss

$$L[\hat{r}(x|\theta_0,\theta_1)] = \frac{1}{N} \sum_{(x_e,z_e,y_e)} \Bigg[y_e \left| r(x_e,z_e|\theta_0,\theta_1) - \hat{r}(x_e|\theta_0,\theta_1) \right|^2 + (1-y_e) \left| \frac{1}{r(x_e,z_e|\theta_0,\theta_1)} - \frac{1}{\hat{r}(x_e|\theta_0,\theta_1)} \right|^2 + \alpha\,(1-y_e) \left| t(x_e,z_e|\theta_0) - \hat{t}(x_e|\theta_0) \right|^2 \Bigg]$$

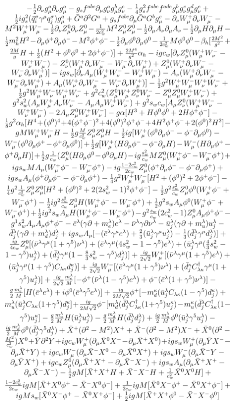
Where: $r(x,z|\theta_0,\theta_1) = \frac{p(x,z|\theta_0)}{p(x,z|\theta_1)}$

And $y_e = \begin{cases} 1 & \text{if } x,z \sim p(x,z|\theta_1) \\ 0 & \text{if } x,z \sim p(x,z|\theta_0) \end{cases}$

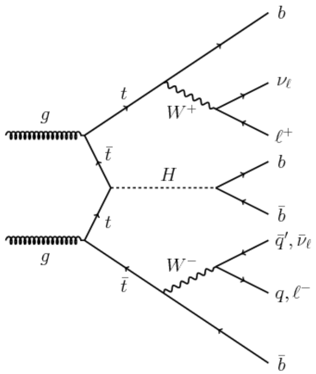
1805.12244, 1805.00020

How can we use this in HEP

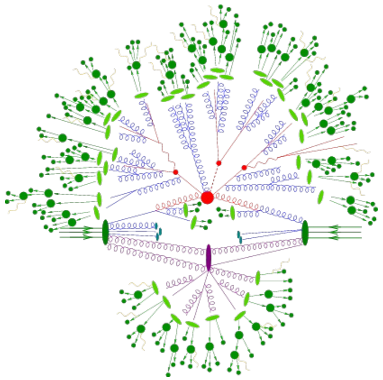
Likelihood in HEP: $p(x|\theta) = \int dz \, p(x|z_h)p(z_h|z_p)p(z_p|\theta)$



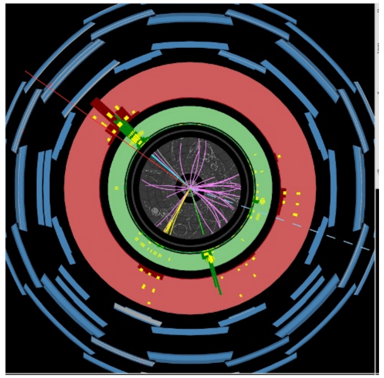
O(20) Fundamental physics parameters θ



O(10) particles



O(100) particles



O(10^8) detector elements

How can we use this in HEP

Likelihood in HEP: $p(x|\theta) = \int dz \, p(x|z_h)p(z_h|z_p)p(z_p|\theta)$

Lets look at the ratio of joint probabilities for a fixed x, z

$$\frac{p(x,z|\theta_0)}{p(x,z|\theta_1)} = \frac{p(x|z_h)p(z_h|z_p)p(z_p|\theta_0)}{p(x|z_h)p(z_h|z_p)p(z_p|\theta_1)}$$

How can we use this in HEP

Likelihood in HEP: $p(x|\theta) = \int dz\, p(x|z_h)p(z_h|z_p)p(z_p|\theta)$

Lets look at the ratio of joint probabilities for a fixed x, z

$$\frac{p(x,z|\theta_0)}{p(x,z|\theta_1)} = \frac{p(x|z_h)p\left(z_h\middle|z_p\right)p\left(z_p\middle|\theta_0\right)}{p(x|z_h)p\left(z_h\middle|z_p\right)p\left(z_p\middle|\theta_1\right)}$$

Same parton, hadronization, and observation configuration in numerator and denominator. **Only different parameter values**

How can we use this in HEP

Likelihood in HEP: $p(x|\theta) = \int dz\, p(x|z_h)p(z_h|z_p)p(z_p|\theta)$

Lets look at the ratio of joint probabilities for a fixed x, z

$$\frac{p(x,z|\theta_0)}{p(x,z|\theta_1)} = \frac{\cancel{p(x|z_h)}\cancel{p(z_h|z_p)}p(z_p|\theta_0)}{\cancel{p(x|z_h)}\cancel{p(z_h|z_p)}p(z_p|\theta_1)}$$

Same parton, hadronization, and observation configuration in numerator and denominator. **Only different parameter values**

At fixed x, z (i.e. fixed simulator evolution and observation), all the particle evolution and measurement process are the same… Cancel out in ratio!

How can we use this in HEP

$$\frac{p(x,z|\theta_0)}{p(x,z|\theta_1)} = \frac{p\left(z_p\middle|\theta_0\right)}{p\left(z_p\middle|\theta_1\right)}$$

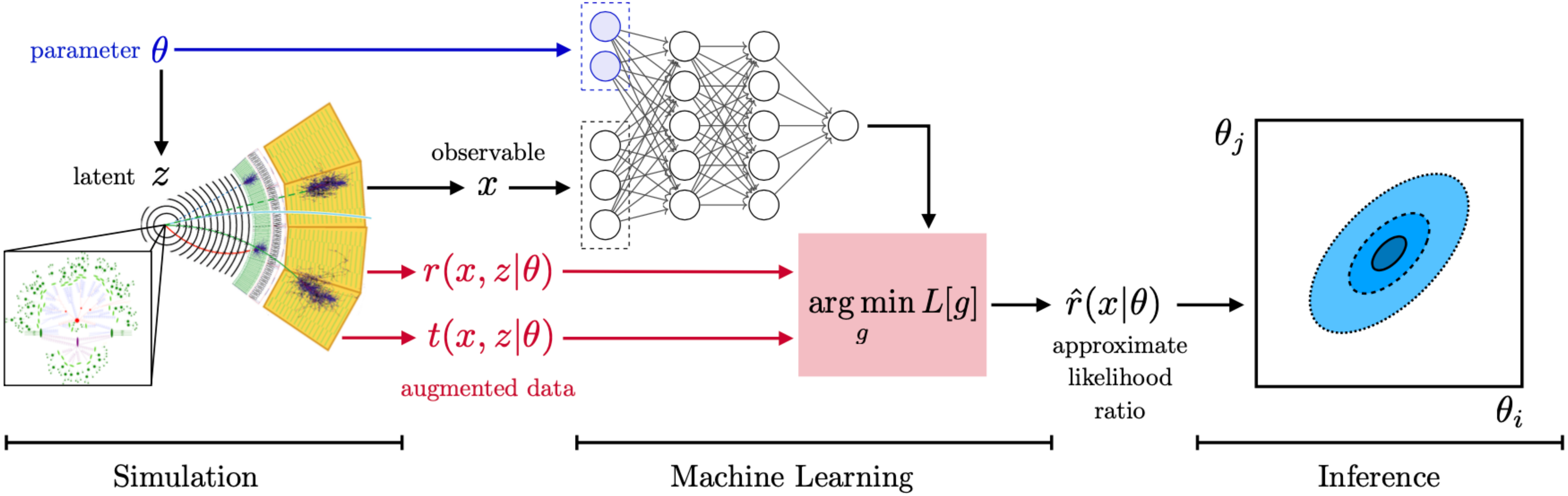
Matrix elements at different parameter values

The joint ratio is the ratio of matrix elements at a given parton configuration!

We can evaluate that and use as training target!

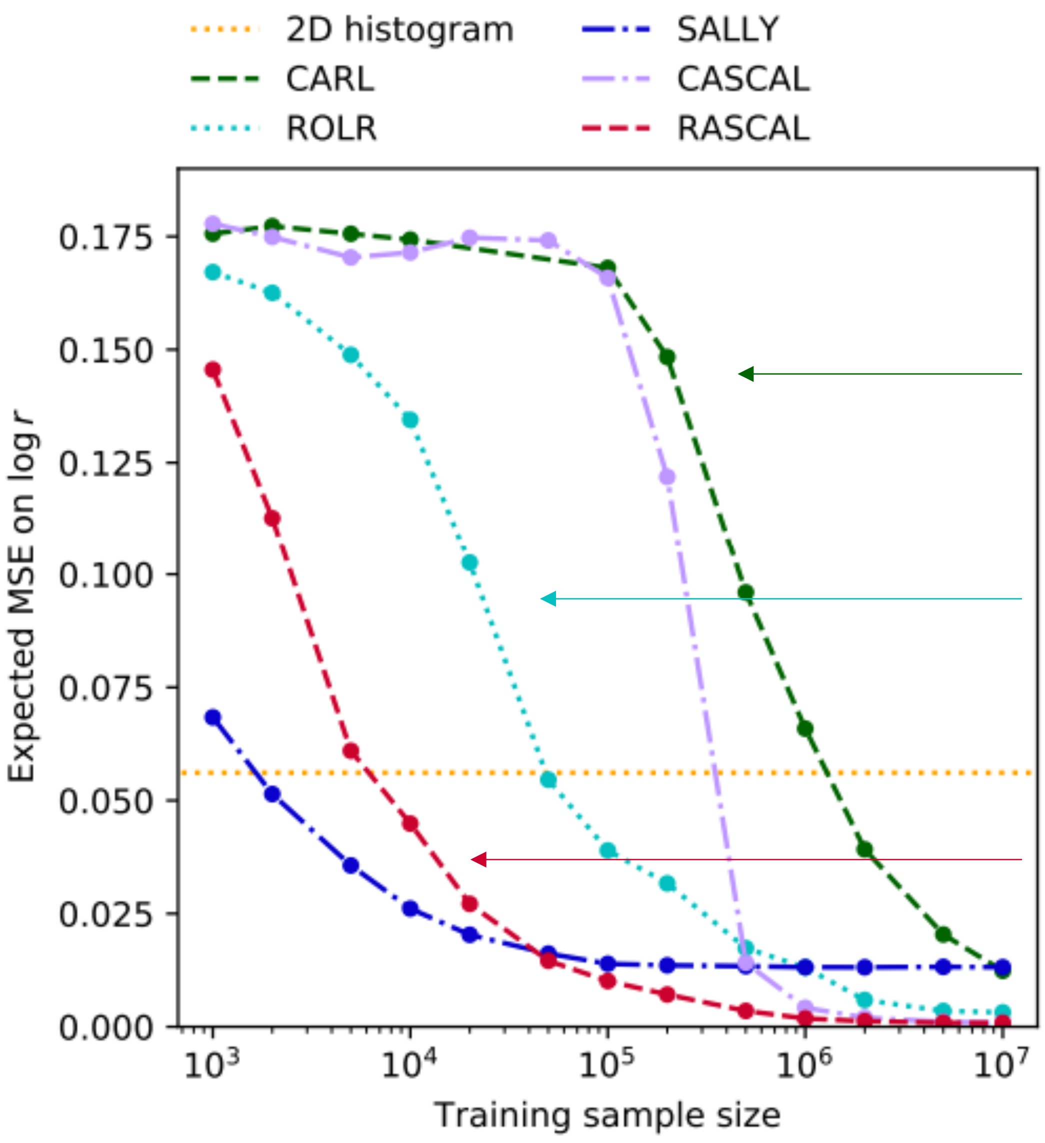
In some cases we can evaluate the gradient… more later

Mining Gold



arXiv:1805.12244

Massive Gains in Data Efficiency



Parameterize classifier

Regression trick on r

Regression trick also on gradients

arXiv:1805.12244

Getting Gradients

Getting gradients requires differentiating arbitrary Matrix Elements

Some matrix elements factorize into a sum of components, each consisting of an analytic function of parameters of interest times a phase space function

$$|\mathcal{M}|^2(z_p|\theta) = \sum_c w_c(\theta)\, f_c(z_p)$$

Getting Gradients

Getting gradients requires differentiating arbitrary Matrix Elements

Some matrix elements factorize into a sum of components, each consisting of an analytic function of parameters of interest times a phase space function

e.g. $$|\mathcal{M}|^2(z_p|\theta) = \underbrace{1}_{w_0(\theta)} \underbrace{|\mathcal{M}_{SM}|^2(z_p)}_{f_0(z_p)} + \underbrace{\theta}_{w_1(\theta)} \underbrace{2\,\mathrm{Re}\,\mathcal{M}^\dagger_{SM}(z_p)\,\mathcal{M}_{BSM}(z_p)}_{f_1(z_p)} + \underbrace{\theta^2}_{w_2(\theta)} \underbrace{|\mathcal{M}_{BSM}|^2(z_p)}_{f_2(z_p)}$$

Getting Gradients

Getting gradients requires differentiating arbitrary Matrix Elements

Some matrix elements factorize into a sum of components, each consisting of an analytic function of parameters of interest times a phase space function

e.g. $$|\mathcal{M}|^2(z_p|\theta) = \underbrace{1}_{w_0(\theta)}\underbrace{|\mathcal{M}_{SM}|^2(z_p)}_{f_0(z_p)} + \underbrace{\theta}_{w_1(\theta)}\underbrace{2\,\mathrm{Re}\,\mathcal{M}^{\dagger}_{SM}(z_p)\,\mathcal{M}_{BSM}(z_p)}_{f_1(z_p)} + \underbrace{\theta^2}_{w_2(\theta)}\underbrace{|\mathcal{M}_{BSM}|^2(z_p)}_{f_2(z_p)}$$

Often the case for effective field theories, when indirect effects of new physics are parameterized through form factors

In this case, can more easily extract the gradients ∇_θ w/o differentiating $f_i(z_p)$

This is implemented in MadMiner

MadJax

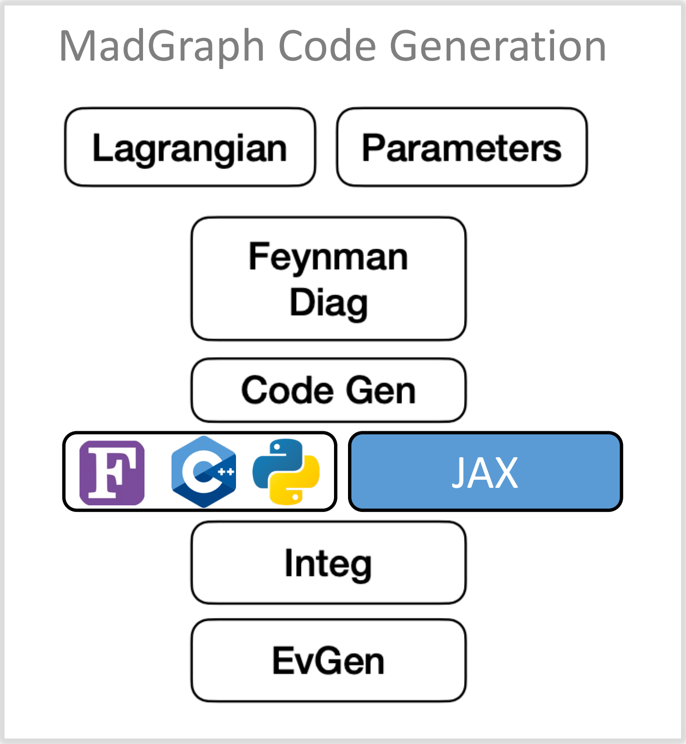
If we don't have this factorization, we need a more general tool for differentiating matrix elements with respect to arbitrary parameters

How can we do this? → Differentiable Programming

(see L. Heinrich lectures)

Create a differentiable matrix element simulator by integrating matrix element generator with an automatic differentiation framework

MadJax = MadGraph + JAX AD framework



arXiv:2203.00057

MadJax

If we don't have this factorization, we need a more general tool for differentiating matrix elements with respect to arbitrary parameters

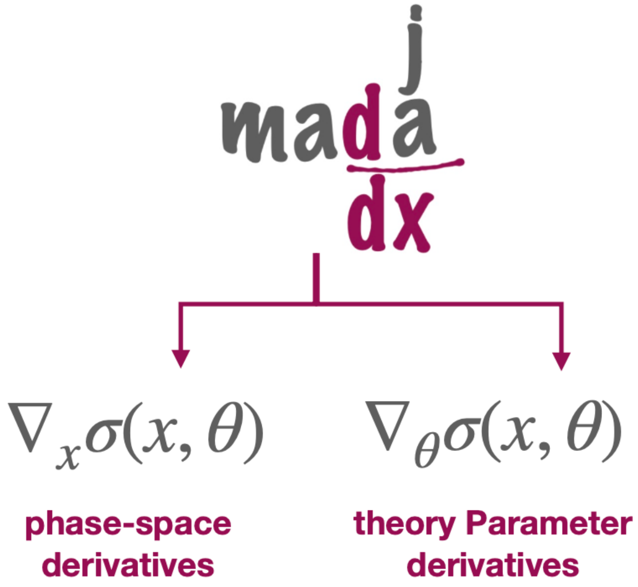
How can we do this? → Differentiable Programming

(see L. Heinrich lectures)

Create a differentiable matrix element simulator by integrating matrix element generator with an automatic differentiation framework

MadJax = MadGraph + JAX AD framework

Access to phase space and parameter gradients



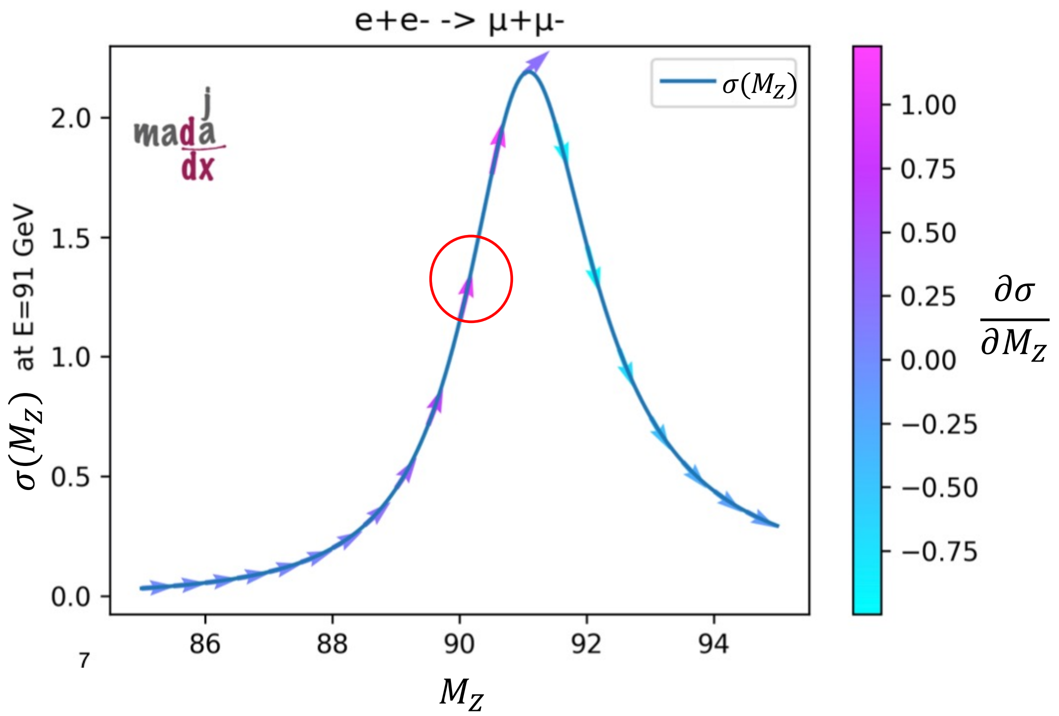
Usage

1. Generation:

```
generate p p > t t~, t > b udsc udscx , t~ > b~ udsc udscx
output madjax generated_ttbar
set auto_update 0
```

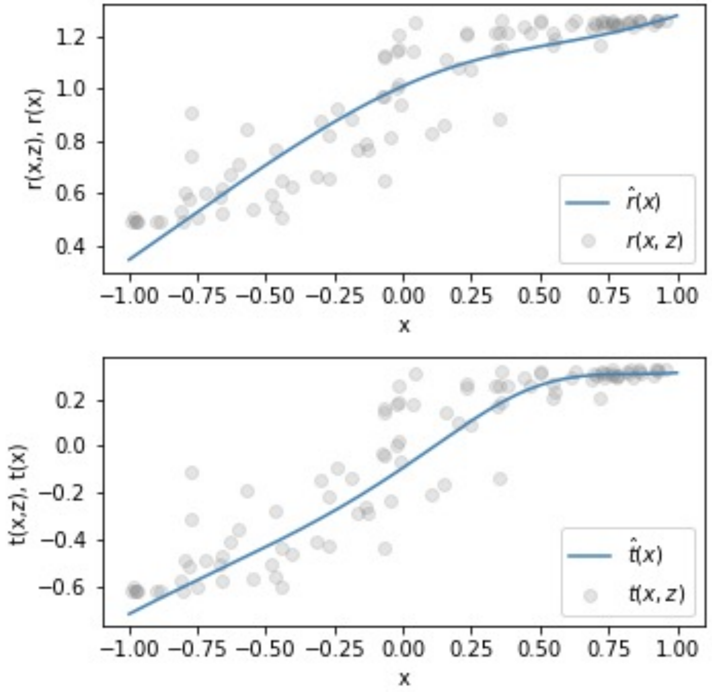
2. Evaluation:

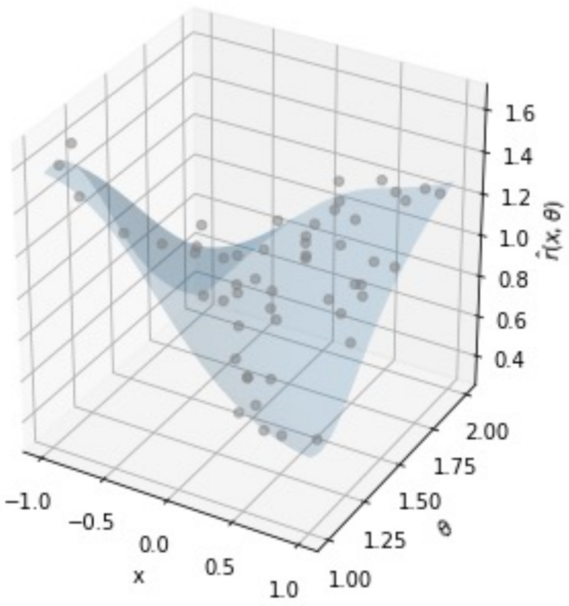
```
import madjax
mj = madjax.MadJax('generated_ttbar')
E_cm = 14000 #GeV
process = 'Matrix_1_gg_ttx_t_budx_tx_bxdux'
matrix_element = mj.matrix_element(E_cm,process)

parameters = {('mass',6): 173.0} #set top mass
phasespace_coords = [0.1]*14 #14D phasespace

val, grad = matrix_element(parameters,phasespace_coords)
grad[('mass', 6)] #gradient wrt top mass
```

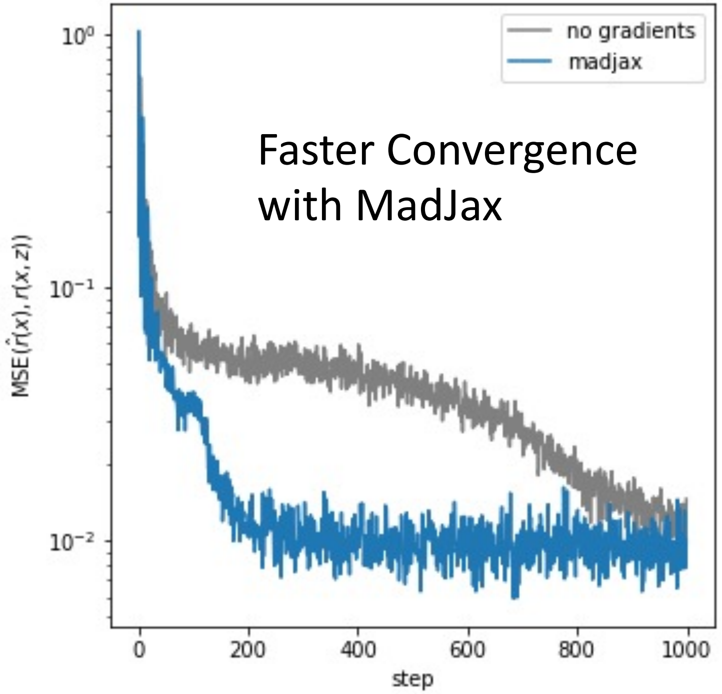


Likelihood Ratio Estimation with MadJax

MadJax enables automatic likelihood-free inference for arbitrary theory parameters (masses, mixings, couplings)





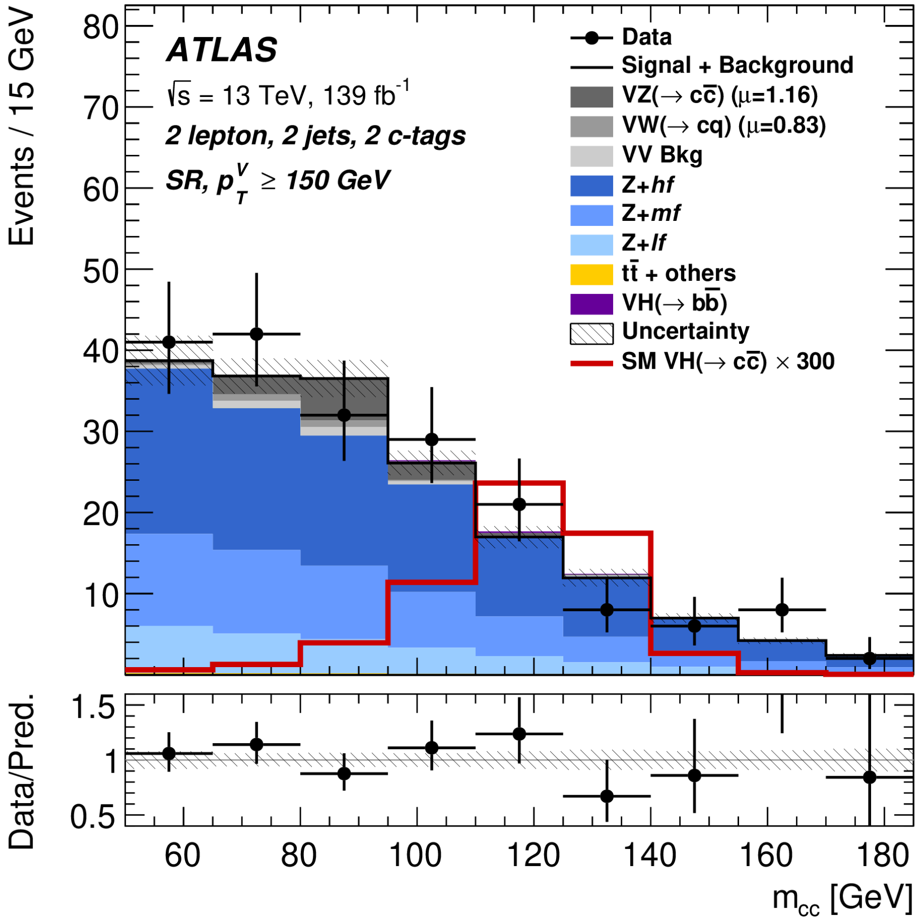


Toy Example: $r(x|G_F)$ in $e^+e^- \to Z \to \mu^+\mu^-$ events

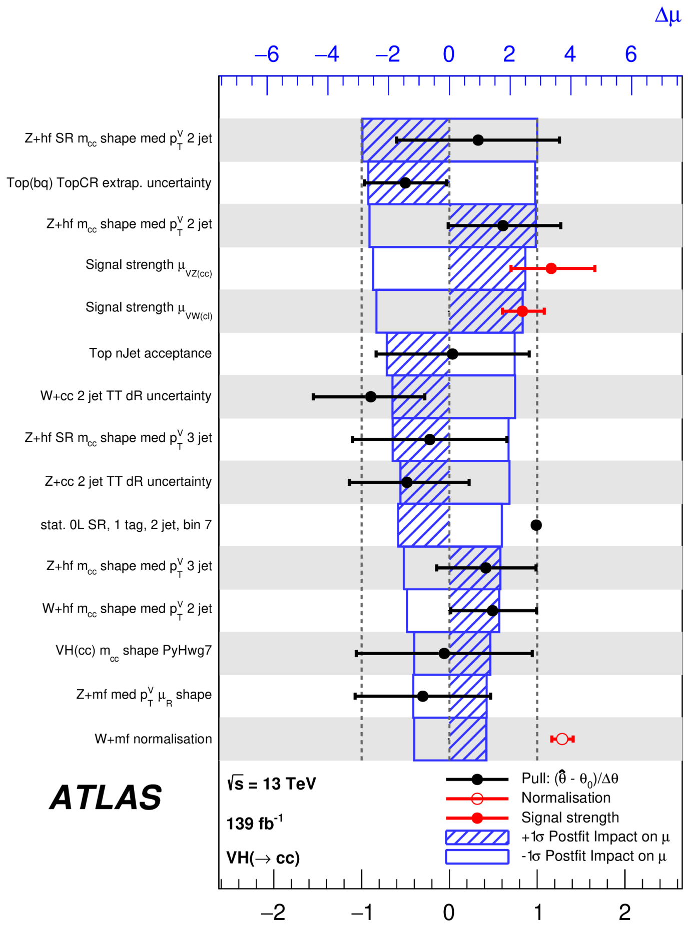
arXiv:2203.00057

What About Systematic Uncertainties

Systematic Uncertainties



Source of uncertainty		$\mu_{VH(c\bar{c})}$
Total		21.5
Statistical		16.2
Systematics		14.0
Statistical uncertainties		
Data statistics only		13.0
Floating normalisations		7.2
Theoretical and modelling uncertainties		
$VH(\to c\bar{c})$		2.1
Z+jets		7.7
Top-quark		5.6
W+jets		3.4
Diboson		0.8
$VH(\to b\bar{b})$		0.8
Multi-Jet		1.0
Simulation statistics		5.1
Experimental uncertainties		
Jets		3.7
Leptons		0.4
$E_{\mathrm{T}}^{\mathrm{miss}}$		0.5
Pile-up and luminosity		0.4
Flavour tagging	c-jets	2.3
	b-jets	1.2
	light-jets	0.7
	τ-jets	0.4
Truth-flavour tagging	ΔR correction	3.0
	Residual non-closure	1.4



arXiv:2201.11428

Nuisance Parameters

Measure / parameterize possible variations over ways data may be generated

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

Parameterized family of likelihood models

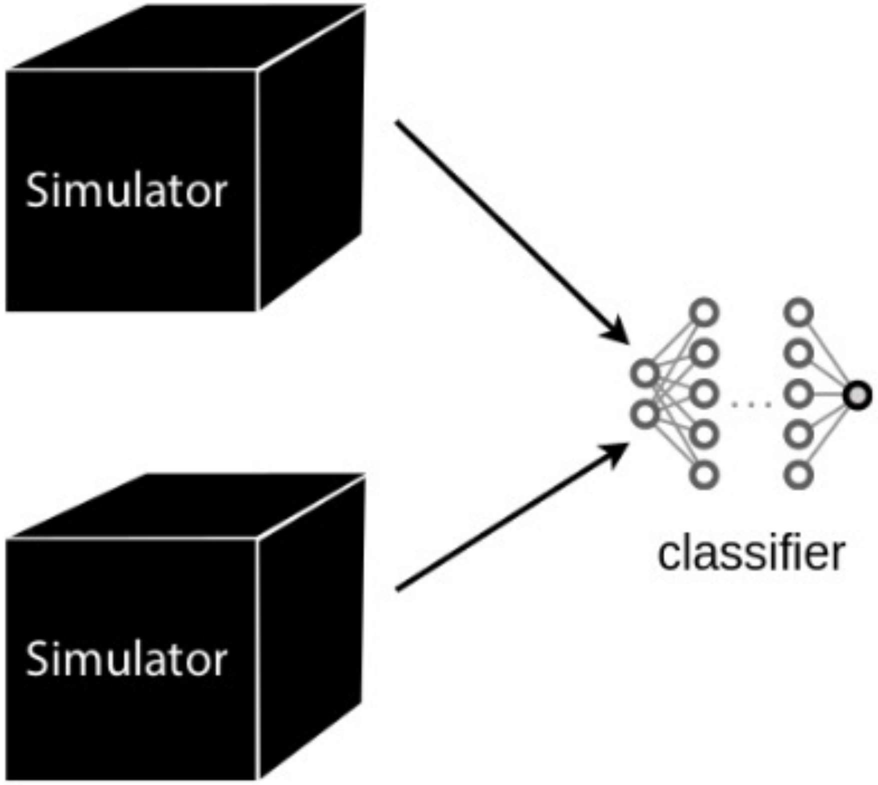
Nuisance Parameter: Parameterizing variations

Often can constrain from auxiliary measurements: $p(x_{aux}|\nu)$
(i.e. from calibrations for reconstructed objects)

Ratio Estimation with Nuisance Parameters

Proposal distribution $\pi(\theta)$, nuisance parameter proposal $\pi(\nu)$

$\theta \sim \pi(\theta)$
$\nu \sim \pi(\nu)$
$x \sim p(x|\theta, \nu)$



$$\hat{r}(x|\theta, \nu, \nu_0) \approx \frac{p(x|\theta, \nu)}{p(x|\theta_0, \nu_0)}$$

θ_0
$\nu_0 \sim \pi(\nu_0)$
$x \sim p(x|\theta_0, \nu)$

Nuisance Parameters and SBI

In principle, as far as density ratio estimation is concerned,
nuisance parameters are just like parameters of interest

→ Effectively increased the parameter dimensionality
→ Practically, need more simulated samples to estimate density ratio well

Nuisance Parameters and SBI

In principle, as far as density ratio estimation is concerned,
nuisance parameters are just like parameters of interest

→ Effectively increased the parameter dimensionality
→ Practically, need more simulated samples to estimate density ratio well

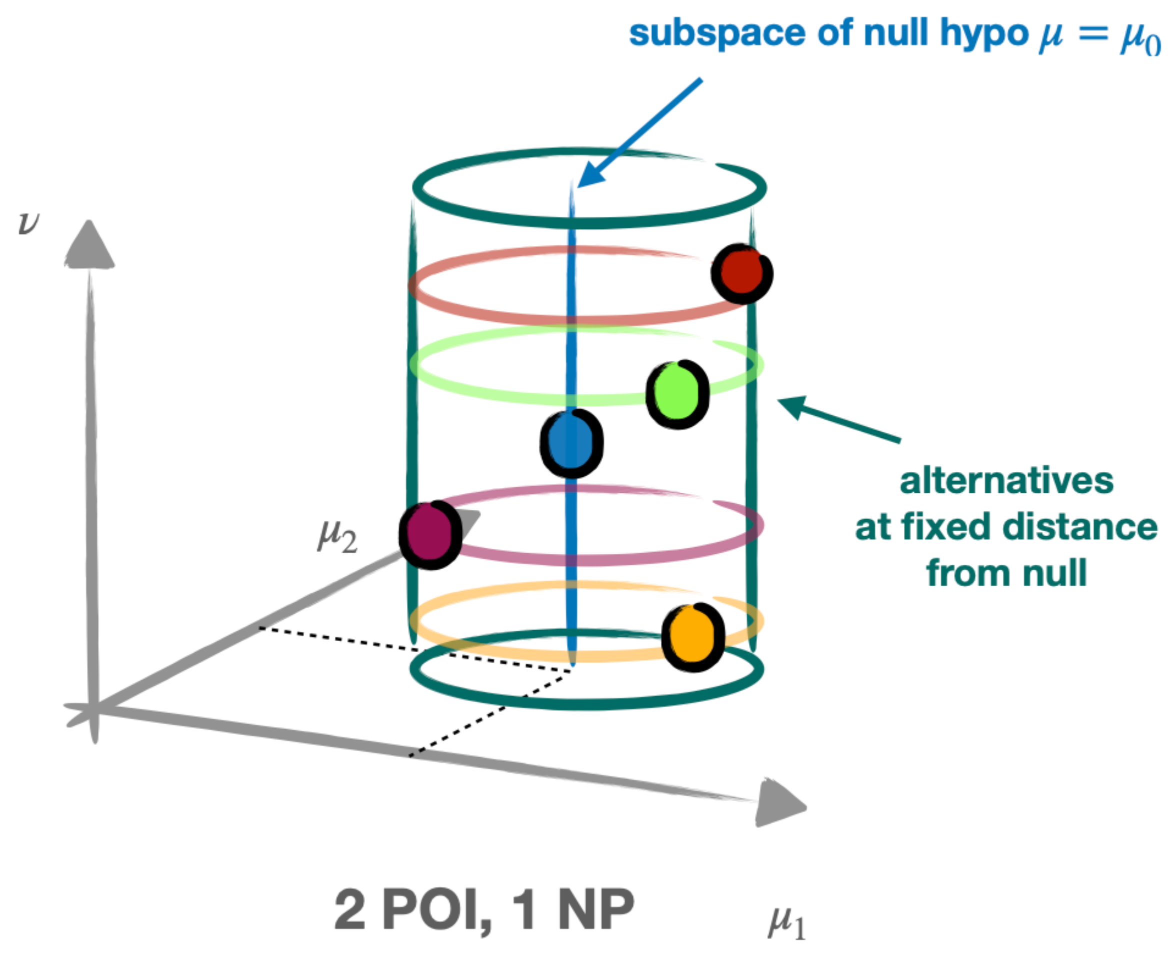
This can be prohibitive, especially for large numbers of nuisance parameters

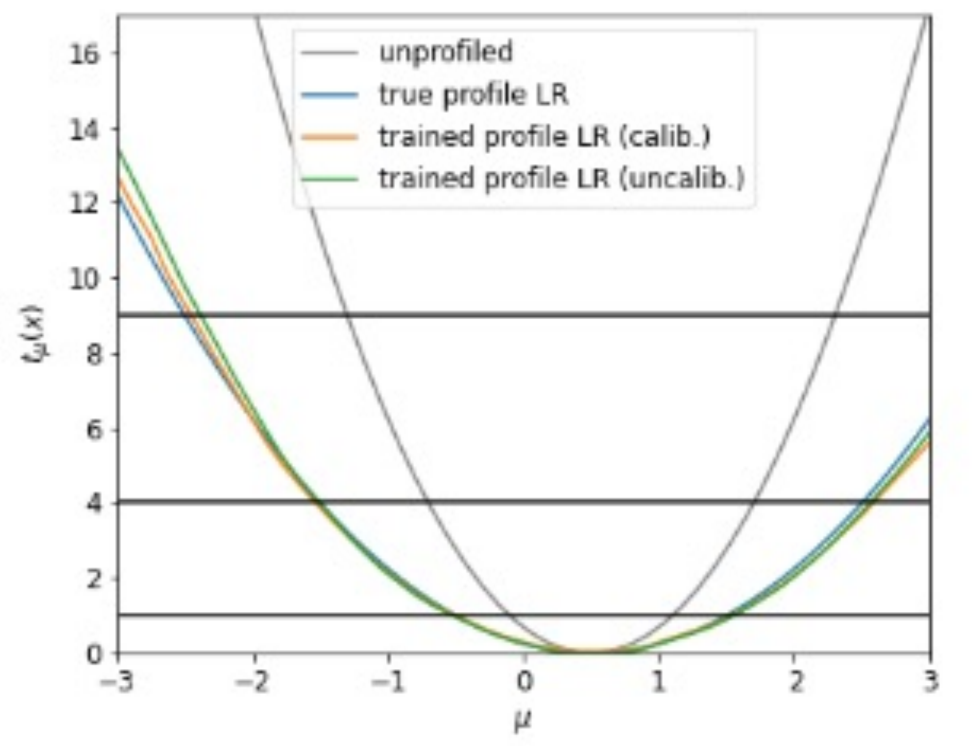
Can limit our ability to estimate profile likelihood ratio: $\dfrac{\max\limits_{\nu} p(x|\theta_0,\nu)}{\max\limits_{\theta,\nu} p(x|\theta,\nu)}$

Open problem on how best to deal w/ (large numbers of) nuisance params

Learning the Profile Likelihood

Interesting recent work aiming to use SBI to learn profile likelihood directly





2203.13079

Wrapping Up

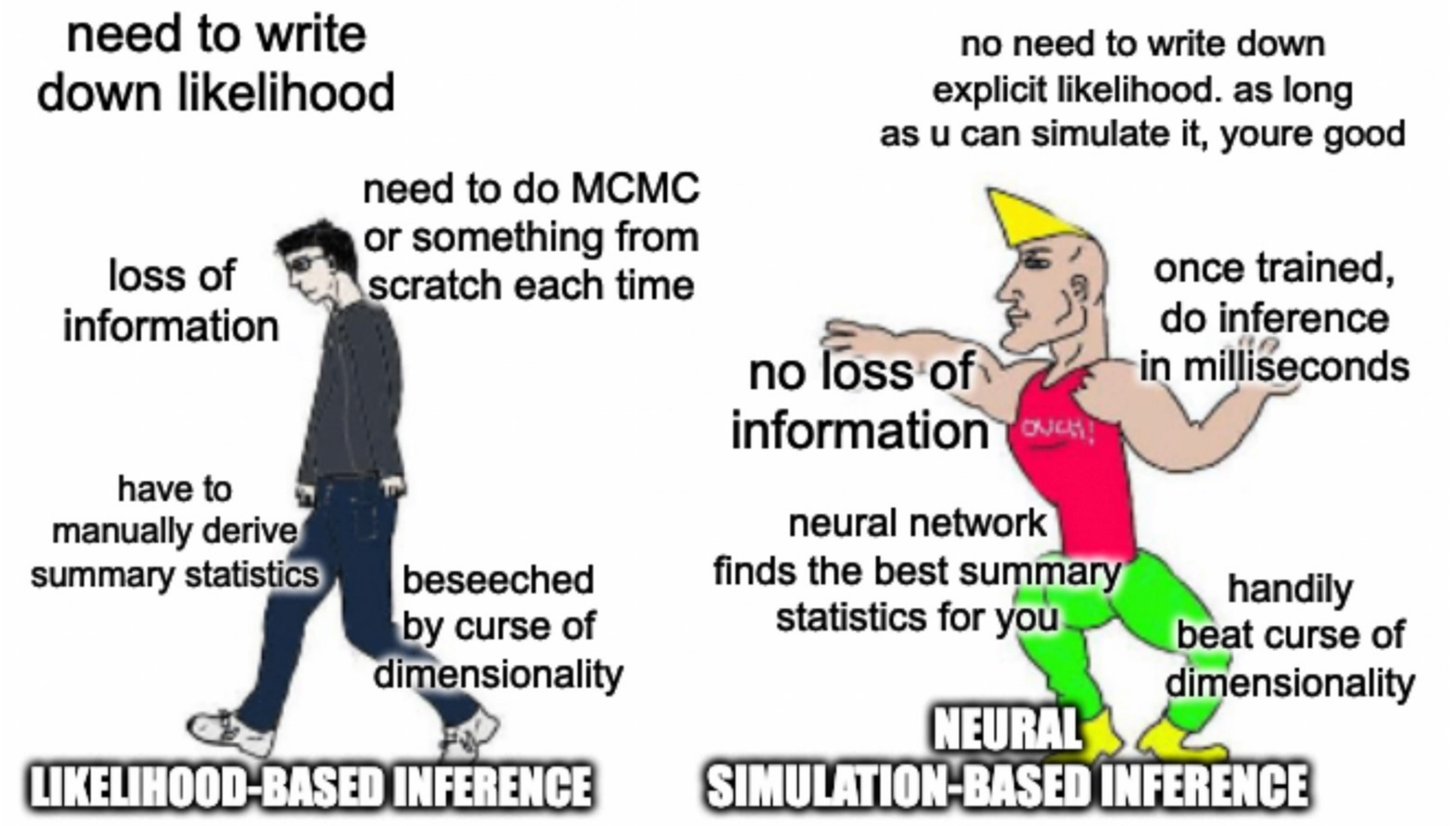


Image credit: S. Mishra-Sharma

Summary

With Simulation-Based Inference, we can use neural networks to help avoid data summarization / compression, and preform inference on high dimensional data and parameter spaces

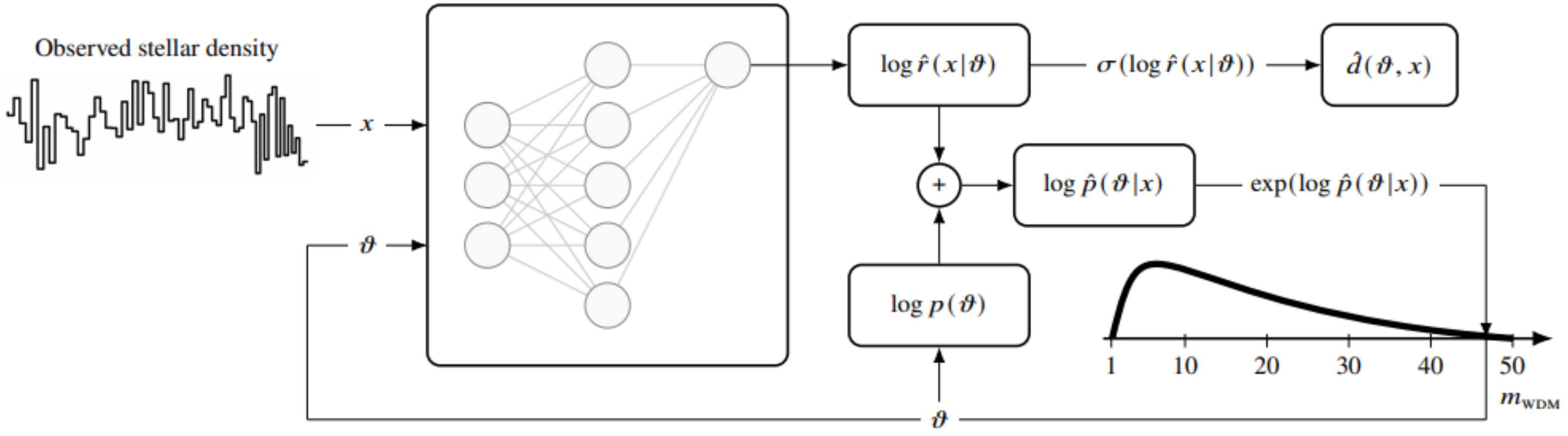
NLE / NPE require density estimation, while neural ratio estimation allows us to use the likelihood ratio trick and train classifiers. NRE can be used for both frequentist and Bayesian inference.

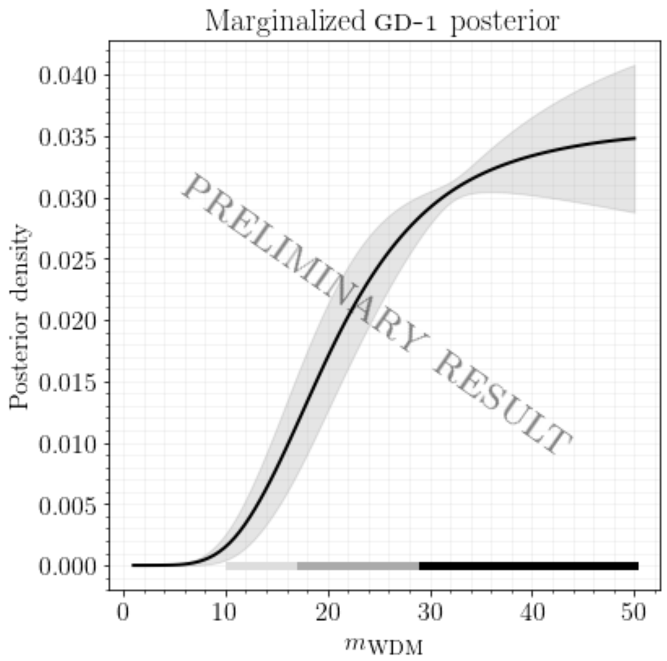
Important to keep in mind model validation and calibration

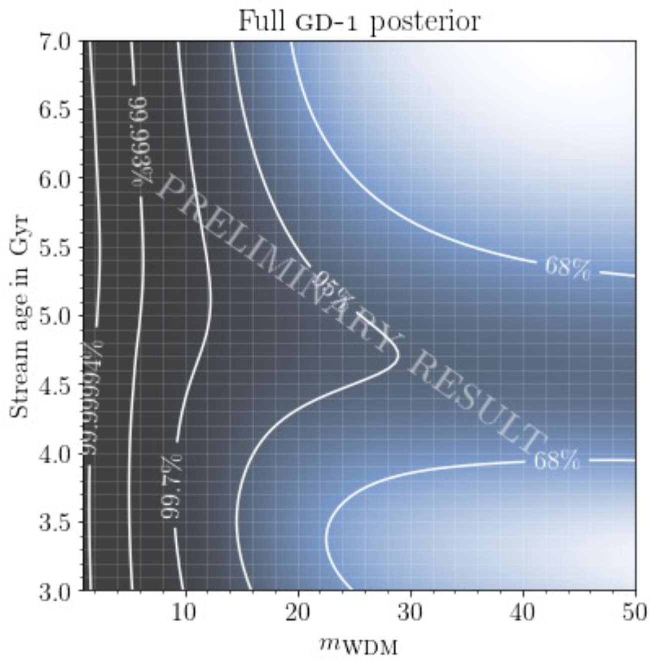
And there is still the challenge of incorporating large numbers of systematic uncertainties. More generally, it's an open question what to do in SBI when the likelihood is not perfectly specified.

Backup

Example







arXiv:2011.14923

How Do We Know If The Model Is Good?

Can we Calibrate Models?

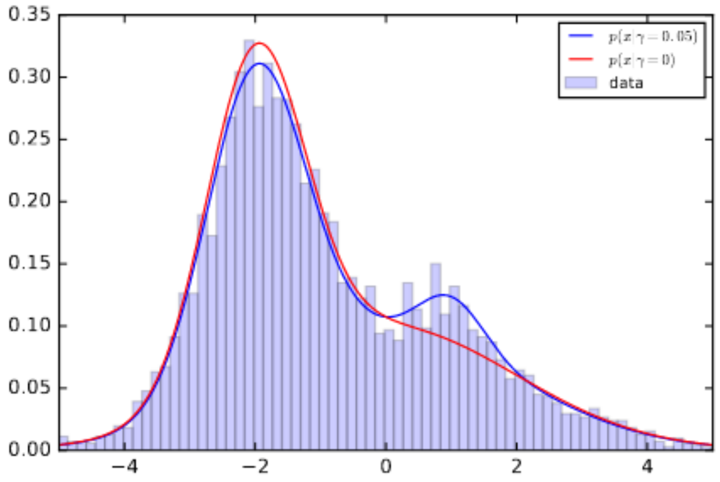
Can we correct an approximate ratio $\hat{r}(x|\theta)$ if it does not exactly predict the true likelihood ratio?

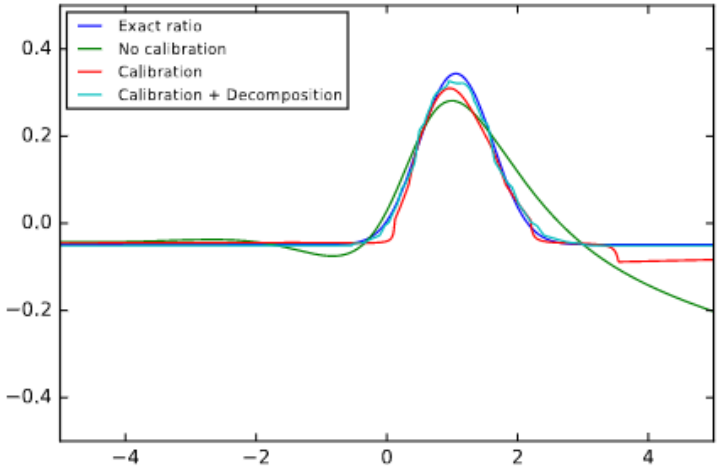
One method: Back to histograms!

- Treat $\hat{r}(x|\theta)$ as a really good summary statistic
- Bin the output values $\hat{r}_i$ evaluated into 1D histogram
  - i.e. 1D density estimation of $\hat{r}$ evaluated on a sample

$$\hat{r}_{cal} = \frac{\hat{p}(\hat{r}_{raw}|\theta_0)}{\hat{p}(\hat{r}_{raw}|\theta_1)}$$

- Perform usual HEP histogram based inference



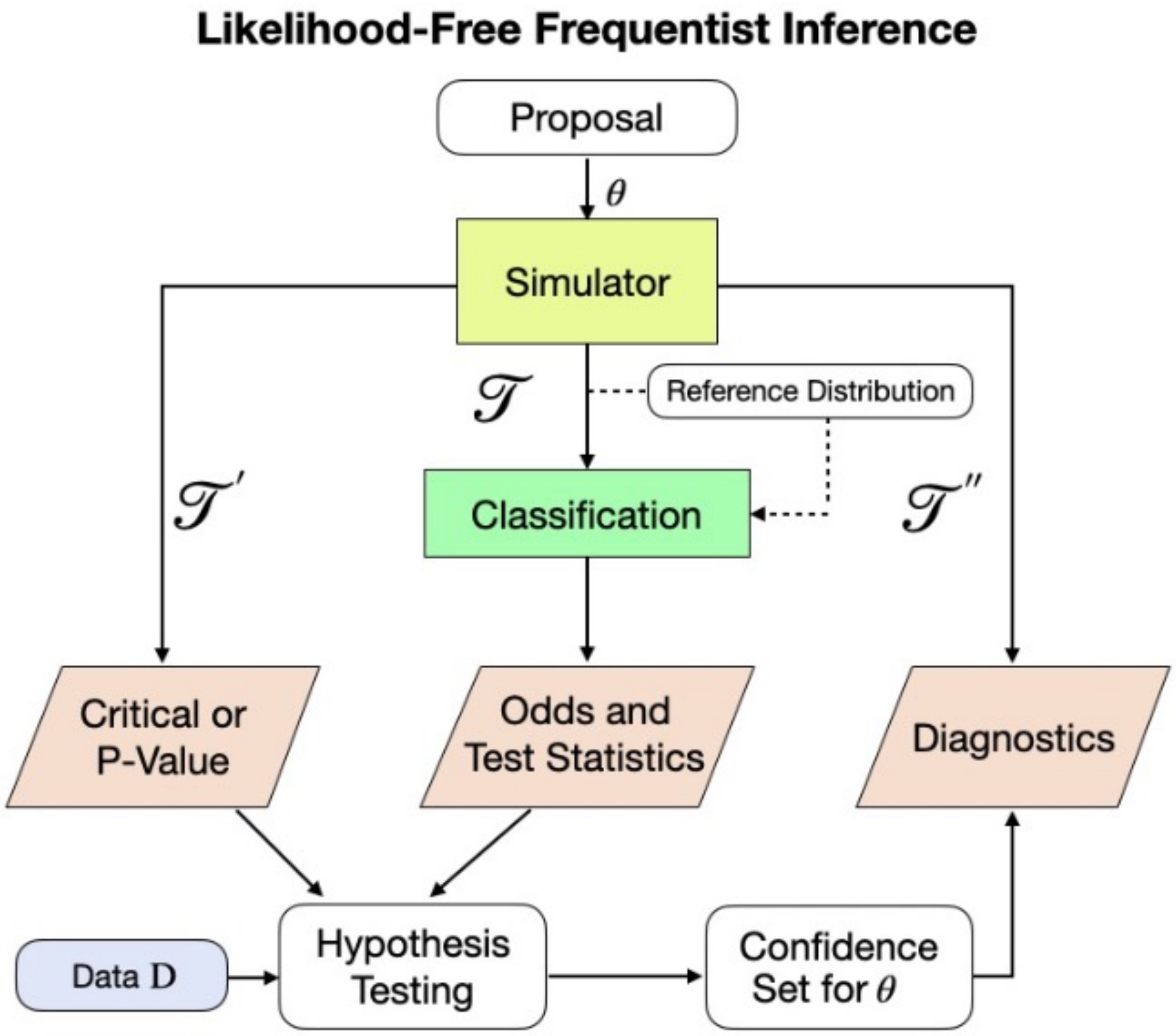


Challenge:

- Different histograms for each θ may require interpolation

1506.02169 , 1805.00020

Joint Likelihood Ratio Estimation, Calibration, and Diagnostics



arXiv:2107.03920