Overcoming limitations to parameter inference using *Neural Ratio Estimation*

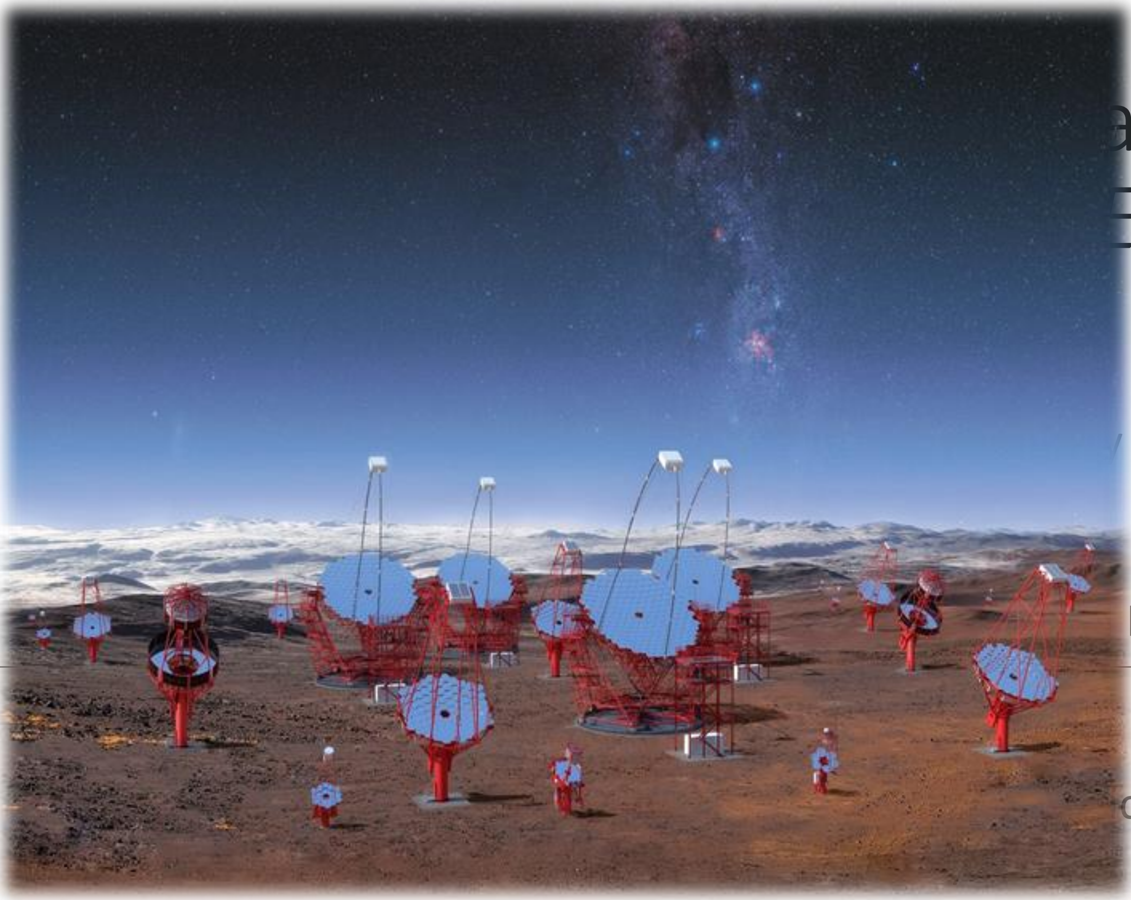
Gert Kluge

PhD candidate at the University of Oslo

In collaboration with
Giacomo D’Amico, Julia Djuvsland, and Heidi Sandaker

Nordic Conference on Particle Physics – Oppland
7th of January 2023

Cherenkov Telescope Array



Credit: ESO

MAGIC

Credit: Robert Wagner, CC BY-SA 2.0

Overcoming limitations to parameter inference using *Neural Ratio Estimation*

Gert Kluge

PhD candidate at the University of Oslo

In collaboration with
Giacomo D'Amico, Julia Djuvsland, and Heidi Sandaker

Nordic Conference on Particle Physics – Oppland
7th of January 2023

Neural Ratio Estimation: A magic trick to do parameter inference

Neural Ratio Estimation: A magic trick to do parameter inference

- Some advantages:
  - General applicability

Neural Ratio Estimation: A magic trick to do parameter inference

- Some advantages:
  - General applicability
    - Assuming that experimental outcomes can be simulated efficiently *(simulation-based inference)*

Neural Ratio Estimation: A magic trick to do parameter inference

- Some advantages:
  - ➢ General applicability
    - \- Assuming that experimental outcomes can be simulated efficiently *(simulation-based inference)*
  - ➢ Avoids over-simplifications
    (Takes into account the uncertainty on most, or all, nuisance parameters)

Neural Ratio Estimation: A magic trick to do parameter inference

- Some advantages:
  - ➢ General applicability
    - Assuming that experimental outcomes can be simulated efficiently *(simulation-based inference)*
  - ➢ Avoids over-simplifications
    (Takes into account the uncertainty on most, or all, nuisance parameters)
  - ➢ Faster than MCMC

Neural Ratio Estimation: A magic trick to do parameter inference

- Some advantages:

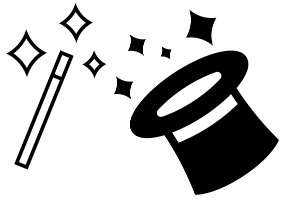
  ➢ General applicability

  - Assuming that experimental outcomes can be simulated efficiently *(simulation-based inference)*

  ➢ Avoids over-simplifications
  (Takes into account the uncertainty on most, or all, nuisance parameters)

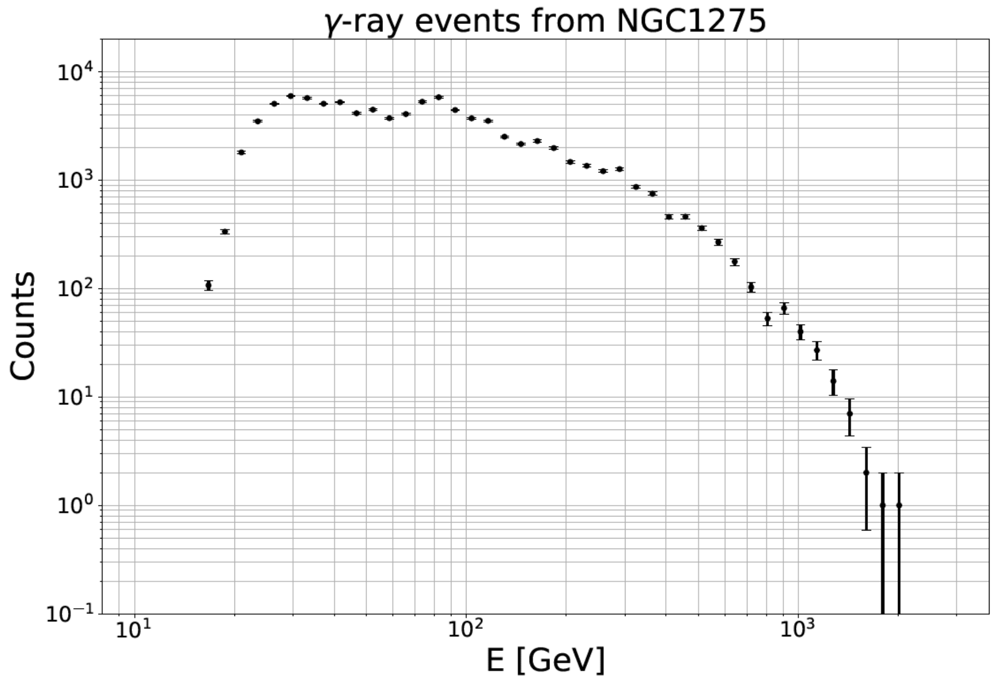
  ➢ Faster than MCMC

Magic?



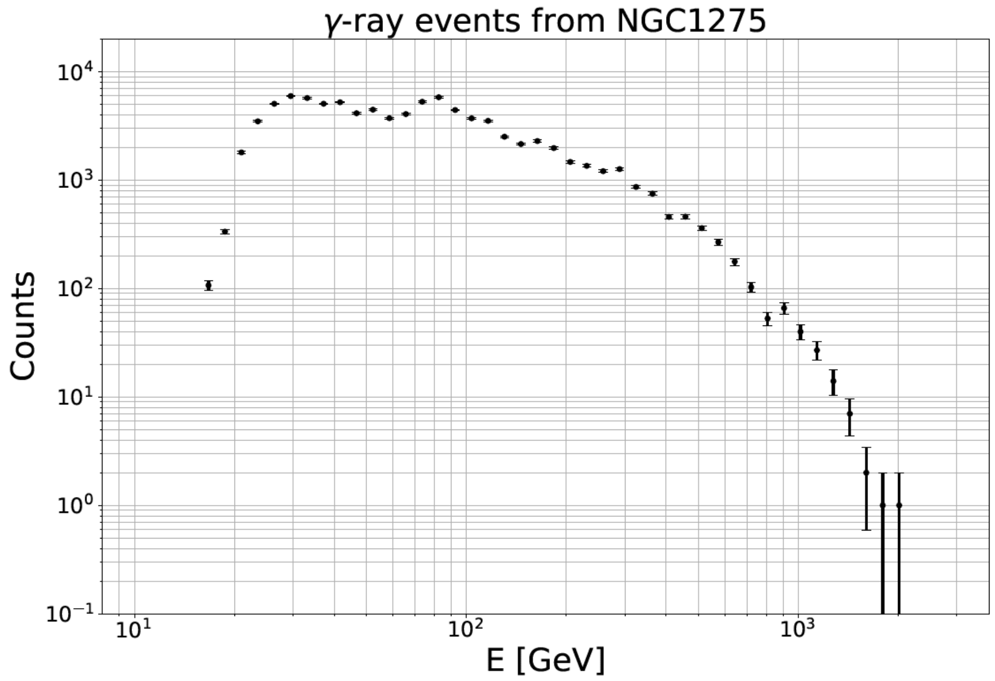
Working example for this talk: fitting the spectrum of NGC1275

Working example for this talk: fitting the spectrum of NGC1275



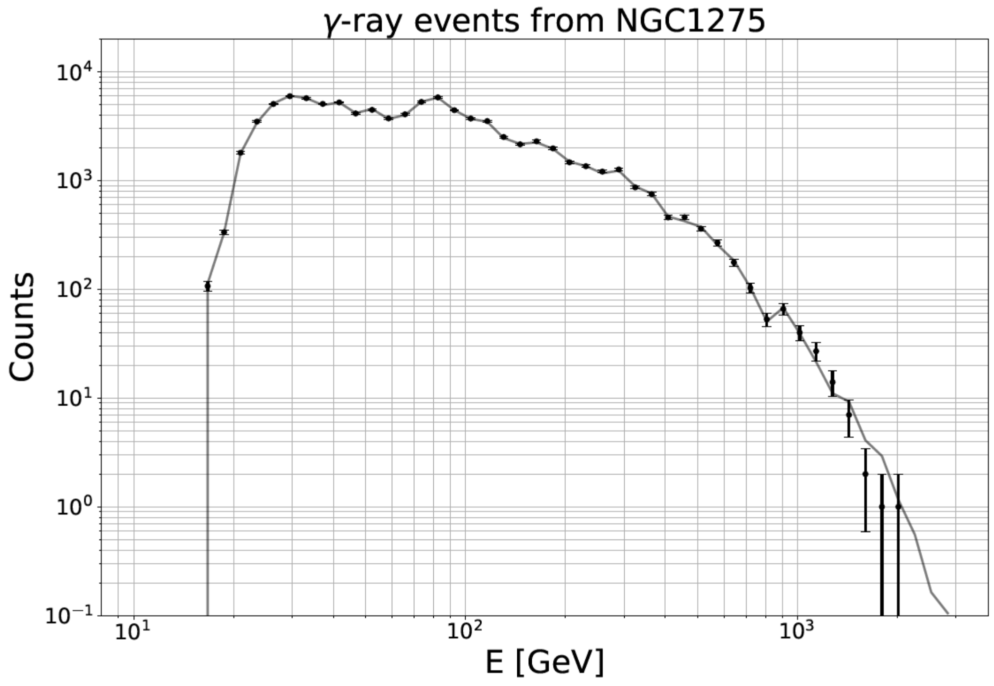
Working example for this talk: fitting the spectrum of NGC1275

$$\varphi(E) = \varphi_0 \left(\frac{E}{E_0}\right)^{\gamma} e^{-E/E_{cut}} \quad + \text{poisson noise}$$



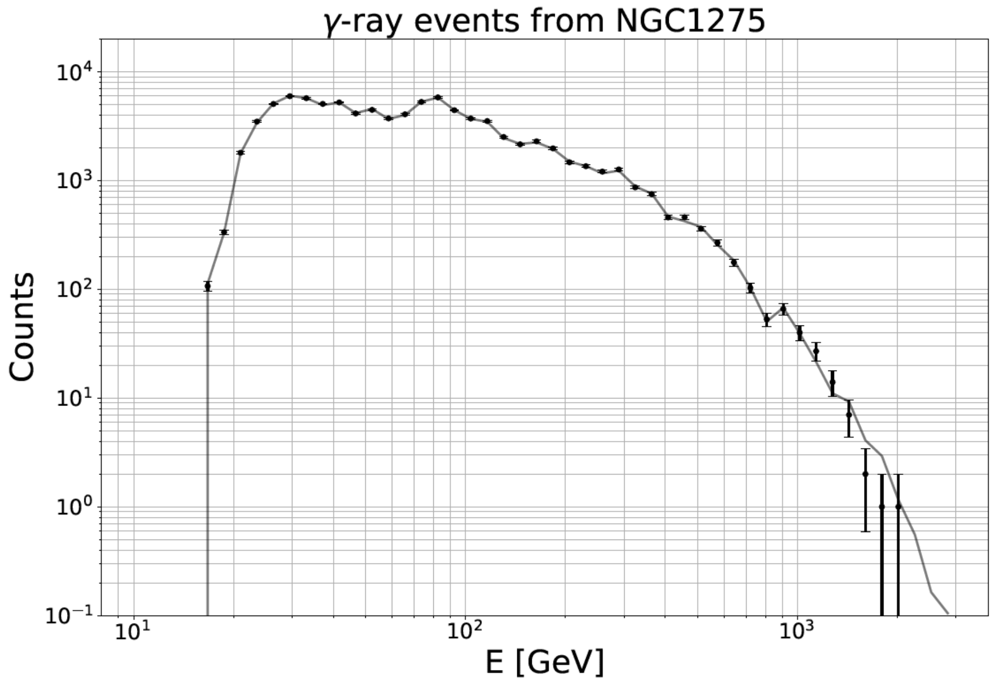
Working example for this talk: fitting the spectrum of NGC1275

$$\varphi(E) = \varphi_0 \left(\frac{E}{E_0}\right)^{\gamma} e^{-E/E_{cut}} \quad + \text{poisson noise}$$

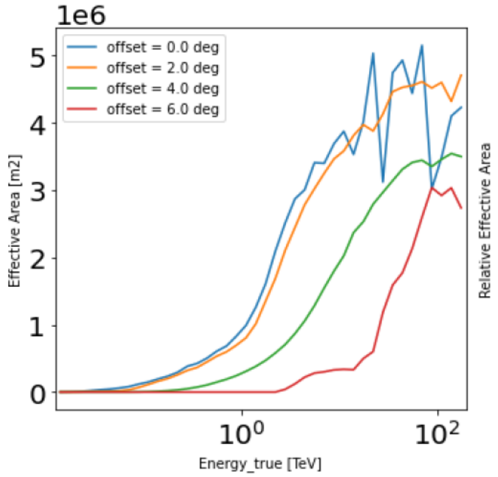


Working example for this talk: fitting the spectrum of NGC1275

$$\varphi(E) = \varphi_0 \left(\frac{E}{E_0}\right)^{\gamma} e^{-E/E_{cut}} \quad + \text{poisson noise}$$



- IRF: `prod3: South_z20_50h`

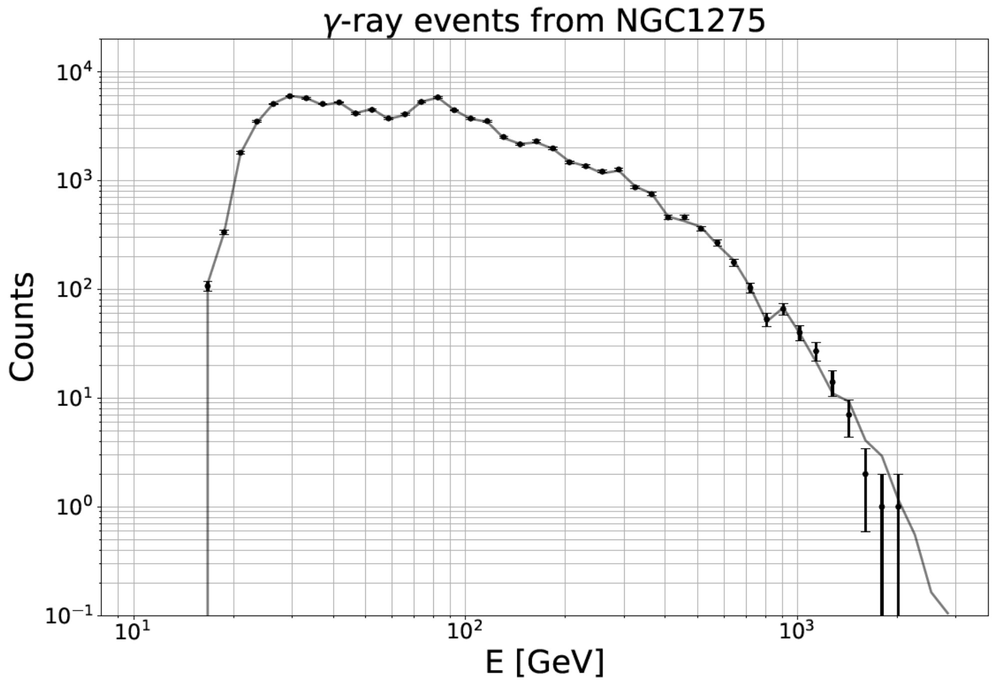


- Livetime = 2 hr
- $Total\ counts \sim 90\ 000$
- $E_0 = 153.86$ GeV

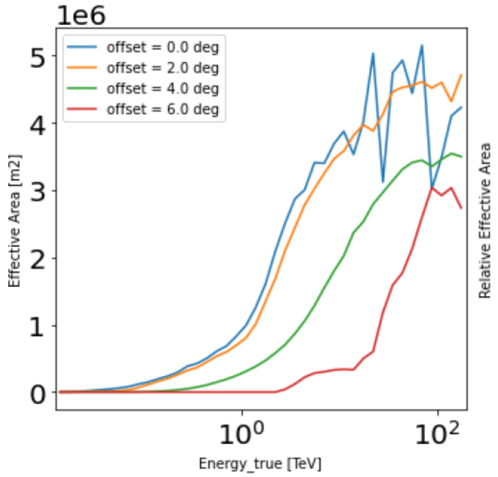
Working example for this talk: fitting the spectrum of NGC1275

$$\varphi(E) = \varphi_0 \left(\frac{E}{E_0}\right)^{\gamma} e^{-E/E_{cut}} + \text{poisson noise}$$

- Parameters of interest:
  - Amplitude φ_0
  - Spectral index γ
  - E_{cut}



- IRF: `prod3: South_z20_50h`



- Livetime = 2 hr
- $Total\ counts \sim 90\ 000$
- $E_0 = 153.86$ GeV

We want to use Bayesian inference...

Bayes theorem:

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{\int \mathrm{d}\boldsymbol{\theta}\; p(\boldsymbol{x}|\boldsymbol{\theta})} p(\boldsymbol{\vartheta})$$

$\boldsymbol{\vartheta}$ = parameters of interest (POIs)

$\boldsymbol{\theta}$ = POIs with nuisance parameters

$\boldsymbol{x}$ = observed outcome

$p(\boldsymbol{\vartheta})$ = Prior ("a priori" assumption)

$p(\boldsymbol{x}|\boldsymbol{\vartheta}) = p(\boldsymbol{x}|\boldsymbol{\theta})$ integrated over uncertainty in nuisance parameters

We want to use Bayesian inference...

Bayes theorem:

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{\int \mathrm{d}\boldsymbol{\theta}\ p(\boldsymbol{x}|\boldsymbol{\theta})} p(\boldsymbol{\vartheta})$$

$\boldsymbol{\vartheta}$ = parameters of interest (POIs)

$\boldsymbol{\theta}$ = POIs with nuisance parameters

$\boldsymbol{x}$ = observed outcome

$p(\boldsymbol{\vartheta})$ = Prior ("a priori" assumption)

$p(\boldsymbol{x}|\boldsymbol{\vartheta}) = p(\boldsymbol{x}|\boldsymbol{\theta})$ integrated over uncertainty in nuisance parameters

We want to use Bayesian inference...

Bayes theorem:

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{\int \mathrm{d}\boldsymbol{\theta}\, p(\boldsymbol{x}|\boldsymbol{\theta})} p(\boldsymbol{\vartheta})$$

$\boldsymbol{\vartheta}$ = parameters of interest (POIs)

$\boldsymbol{\theta}$ = POIs with nuisance parameters

$\boldsymbol{x}$ = observed outcome

$p(\boldsymbol{\vartheta})$ = Prior ("a priori" assumption)

$p(\boldsymbol{x}|\boldsymbol{\vartheta}) = p(\boldsymbol{x}|\boldsymbol{\theta})$ integrated over uncertainty in nuisance parameters

... but it is often too expensive

We want to use Bayesian inference...

Bayes theorem:

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{\int \mathrm{d}\boldsymbol{\theta}\, p(\boldsymbol{x}|\boldsymbol{\theta})} p(\boldsymbol{\vartheta})$$

$\boldsymbol{\vartheta}$ = parameters of interest (POIs)

$\boldsymbol{\theta}$ = POIs with nuisance parameters

$\boldsymbol{x}$ = observed outcome

$p(\boldsymbol{\vartheta})$ = Prior ("a priori" assumption)

$p(\boldsymbol{x}|\boldsymbol{\vartheta}) = p(\boldsymbol{x}|\boldsymbol{\theta})$ integrated over uncertainty in nuisance parameters

... but it is often too expensive

- Computing cost explodes with number of nuisance parameters

  → Need to make simplifying assumptions

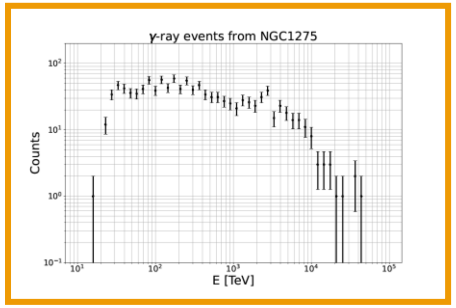
Neural networks can approximate posteriors!

Black Box

Neural Network

Neural networks can approximate posteriors!

$\boldsymbol{x} =$



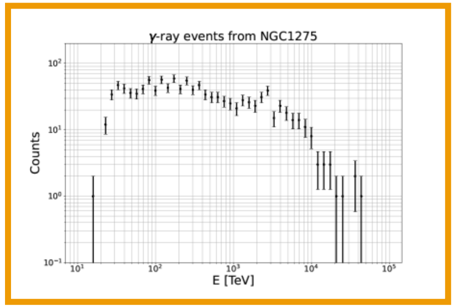
Input

Black Box

Neural Network

Neural networks can approximate posteriors!

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} = \begin{matrix} \boldsymbol{\varphi_0 = 10} \\ \boldsymbol{\gamma = 1} \\ \mathbf{E_{cut} = 3} \end{matrix}$

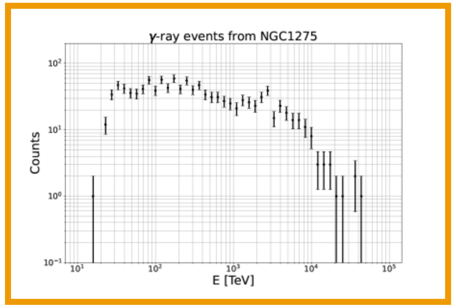
Input

Black Box

Neural Network

Neural networks can approximate posteriors!

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} =$ $\begin{matrix} \boldsymbol{\varphi_0 = 10} \\ \boldsymbol{\gamma = 1} \\ \mathbf{E_{cut} = 3} \end{matrix}$

Input

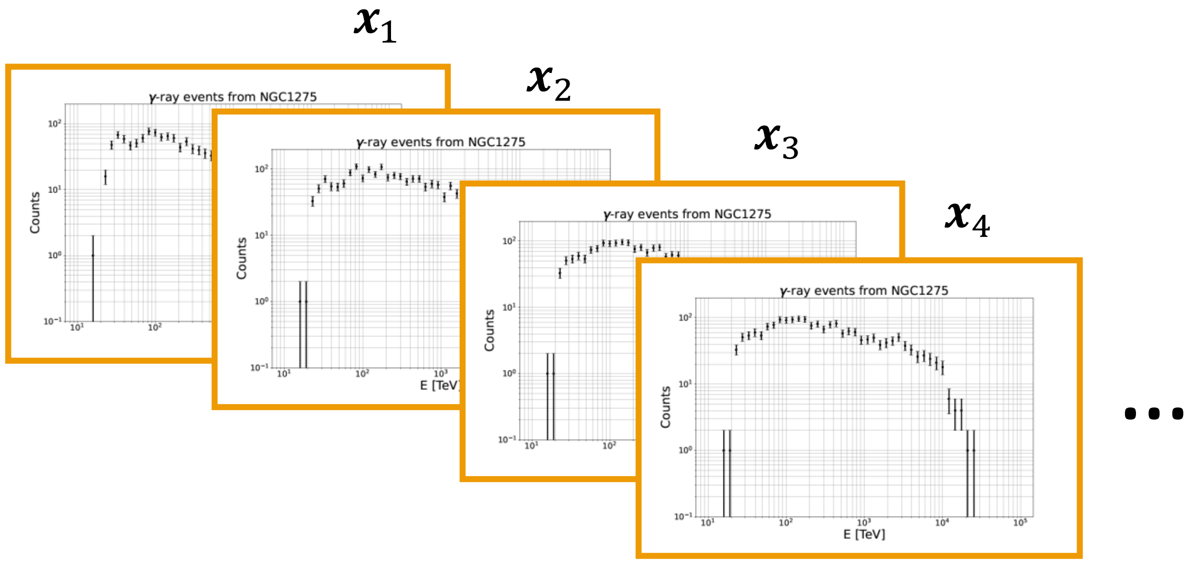
Black Box

Neural Network

Output

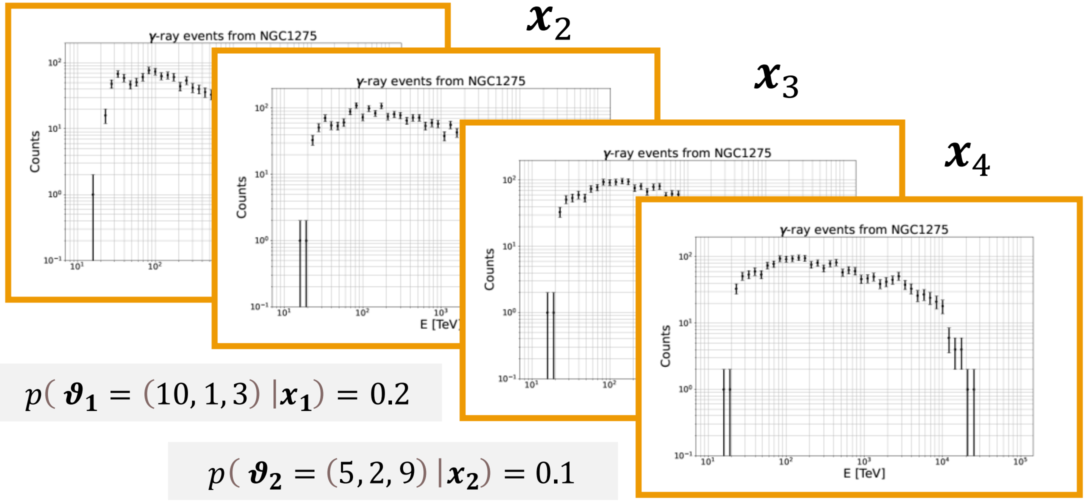
$p(\boldsymbol{\vartheta} = (10,1,3)\,|\boldsymbol{x}) = 0.2$

A network learns to do approximations by example:



A network learns to do approximations by example:

$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$



…

$p(\,\boldsymbol{\vartheta}_1 = (10, 1, 3)\,|\boldsymbol{x}_1) = 0.2$

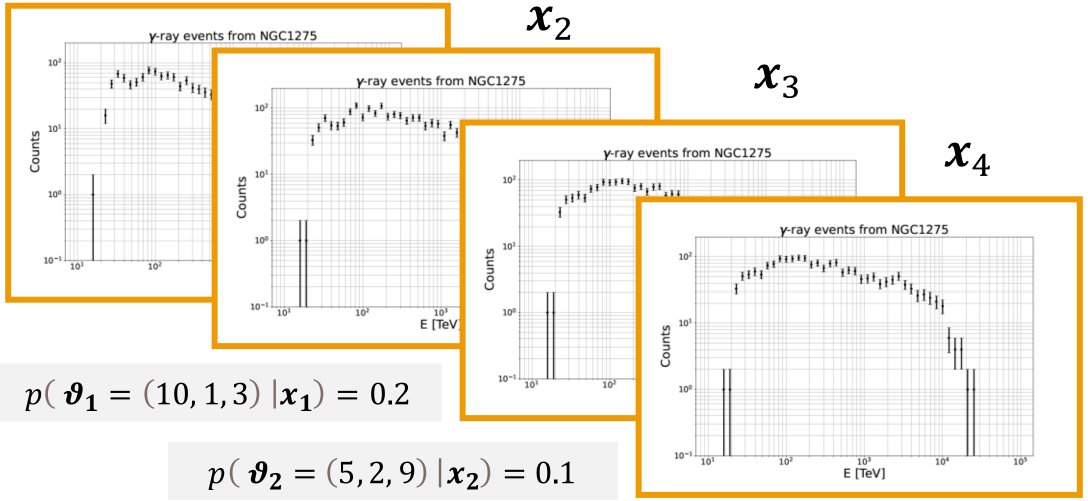
$p(\,\boldsymbol{\vartheta}_2 = (5, 2, 9)\,|\boldsymbol{x}_2) = 0.1$

$p(\,\boldsymbol{\vartheta}_3 = (1, 0.5, 4)\,|\boldsymbol{x}_3) = 0.07$

$p(\,\boldsymbol{\vartheta}_4 = (3, 8, 7)\,|\boldsymbol{x}_4) = 0.03$

A network learns to do approximations by example:

$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$



$p(\boldsymbol{\vartheta}_1 = (10, 1, 3) \,|\boldsymbol{x}_1) = 0.2$

$p(\boldsymbol{\vartheta}_2 = (5, 2, 9) \,|\boldsymbol{x}_2) = 0.1$

$p(\boldsymbol{\vartheta}_3 = (1, 0.5, 4) \,|\boldsymbol{x}_3) = 0.07$

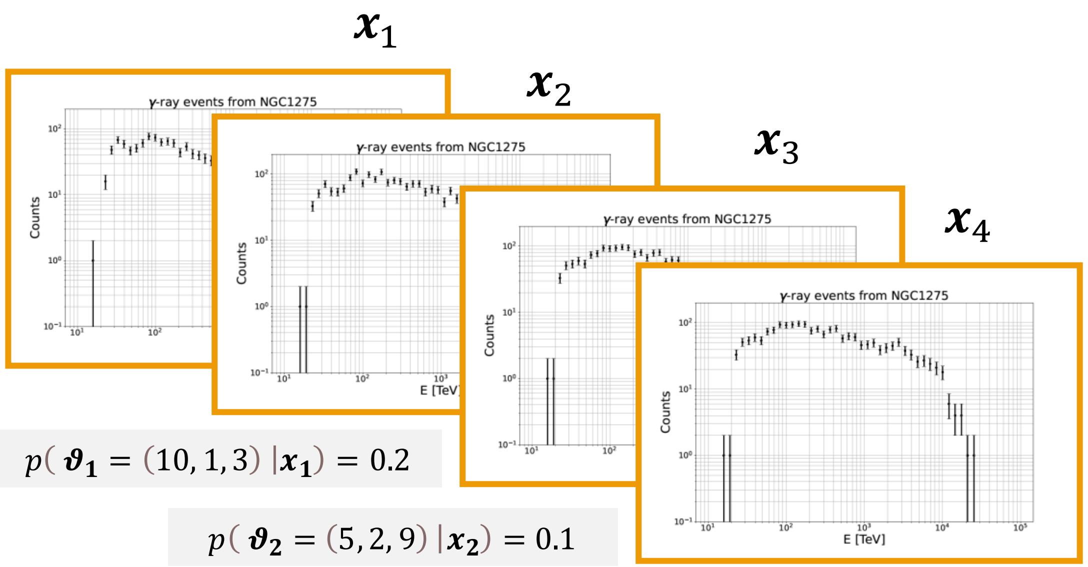
$p(\boldsymbol{\vartheta}_4 = (3, 8, 7) \,|\boldsymbol{x}_4) = 0.03$

…

Training

Neural Network

A network learns to do approximations by example:



$p(\,\boldsymbol{\vartheta}_1 = (10, 1, 3)\,|\boldsymbol{x}_1) = 0.2$

$p(\,\boldsymbol{\vartheta}_2 = (5, 2, 9)\,|\boldsymbol{x}_2) = 0.1$

$p(\,\boldsymbol{\vartheta}_3 = (1, 0.5, 4)\,|\boldsymbol{x}_3) = 0.07$

$p(\,\boldsymbol{\vartheta}_4 = (3, 8, 7)\,|\boldsymbol{x}_4) = 0.03$

...

Training

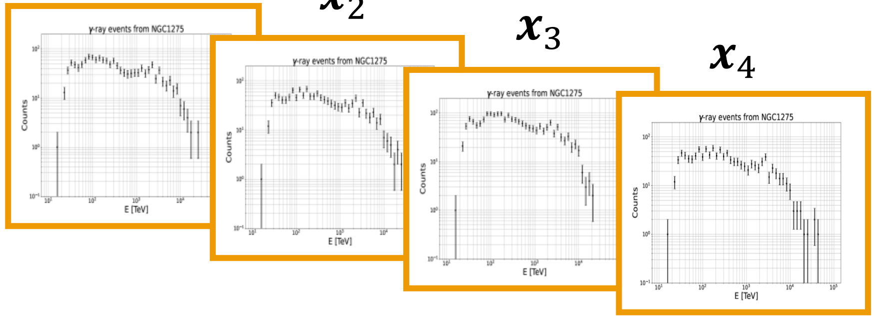
Neural Network

... but this assumes that we can already calculate the posterior!

We can “trick” a network to learn the posterior implicitly

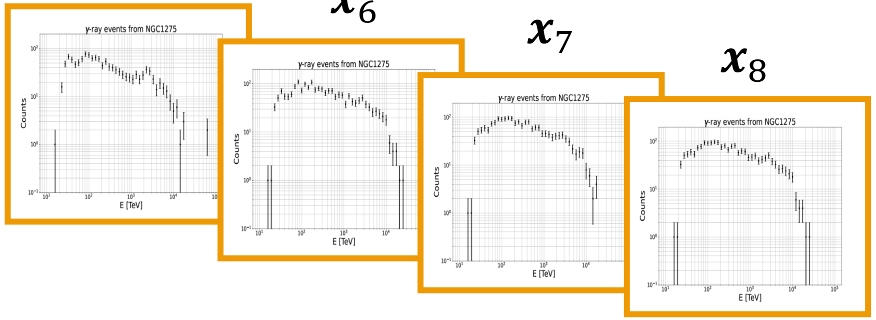
We can “trick” a network to learn the posterior implicitly

$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$



…

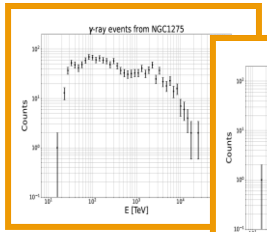
$\boldsymbol{x}_5$ $\boldsymbol{x}_6$ $\boldsymbol{x}_7$ $\boldsymbol{x}_8$



…

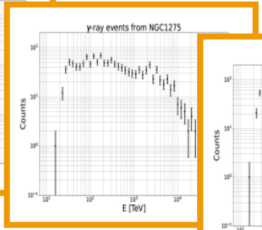
We can “trick” a network to learn the posterior implicitly

$\boldsymbol{x}_1$



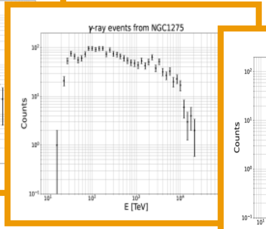
$\boldsymbol{\varphi_0 = 10}$
$\boldsymbol{\gamma = 1}$
$\mathbf{E_{cut} = 3}$

$\boldsymbol{x}_2$



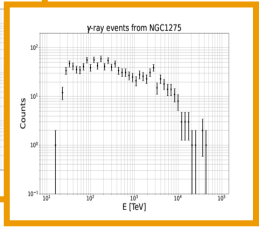
$\boldsymbol{\varphi_0 = 7}$
$\boldsymbol{\gamma = 8}$
$\mathbf{E_{cut} = 2}$

$\boldsymbol{x}_3$



$\boldsymbol{\varphi_0 = 2}$
$\boldsymbol{\gamma = 4}$
$\mathbf{E_{cut} = 3}$

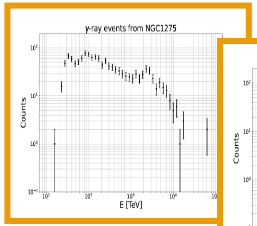
$\boldsymbol{x}_4$



$\boldsymbol{\varphi_0 = 1.6}$
$\boldsymbol{\gamma = 20}$
$\mathbf{E_{cut} = 9}$

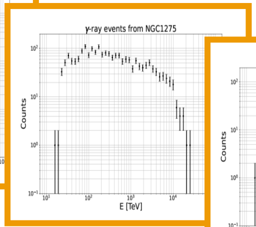
…

$\boldsymbol{x}_5$



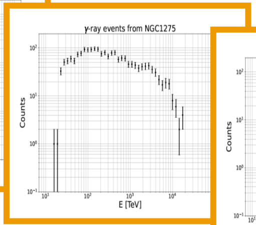
$\boldsymbol{\varphi_0 = 16}$
$\boldsymbol{\gamma = 1}$
$\mathbf{E_{cut} = 5}$

$\boldsymbol{x}_6$



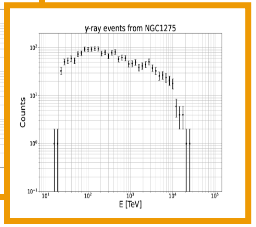
$\boldsymbol{\varphi_0 = 2}$
$\boldsymbol{\gamma = 1}$
$\mathbf{E_{cut} = 9}$

$\boldsymbol{x}_7$



$\boldsymbol{\varphi_0 = 2.5}$
$\boldsymbol{\gamma = 18}$
$\mathbf{E_{cut} = 2}$

$\boldsymbol{x}_8$

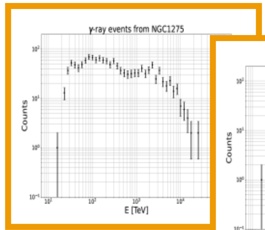


$\boldsymbol{\varphi_0 = 5}$
$\boldsymbol{\gamma = 5.6}$
$\mathbf{E_{cut} = 2}$

…

We can “trick” a network to learn the posterior implicitly

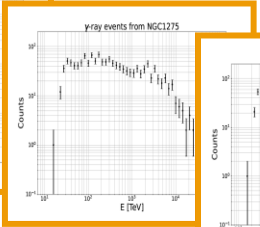
$\boldsymbol{x}_1$



$\varphi_0 = 10$
$\gamma = 1$
$E_{cut} = 3$

True

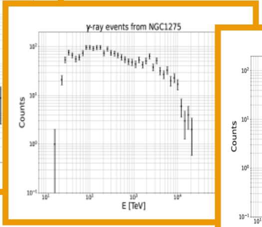
$\boldsymbol{x}_2$



$\varphi_0 = 7$
$\gamma = 8$
$E_{cut} = 2$

True

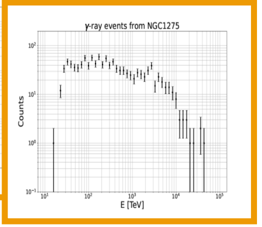
$\boldsymbol{x}_3$



$\varphi_0 = 2$
$\gamma = 4$
$E_{cut} = 3$

True

$\boldsymbol{x}_4$

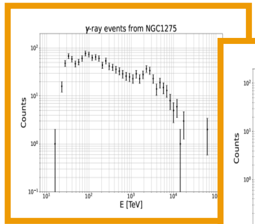


$\varphi_0 = 1.6$
$\gamma = 20$
$E_{cut} = 9$

True

…

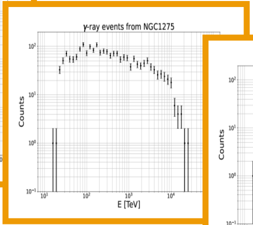
$\boldsymbol{x}_5$



$\varphi_0 = 16$
$\gamma = 1$
$E_{cut} = 5$

False

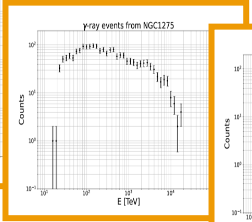
$\boldsymbol{x}_6$



$\varphi_0 = 2$
$\gamma = 1$
$E_{cut} = 9$

False

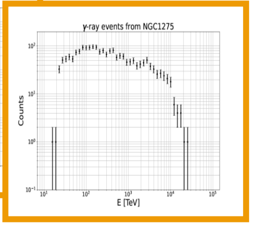
$\boldsymbol{x}_7$



$\varphi_0 = 2.5$
$\gamma = 18$
$E_{cut} = 2$

False

$\boldsymbol{x}_8$



$\varphi_0 = 5$
$\gamma = 5.6$
$E_{cut} = 2$

False

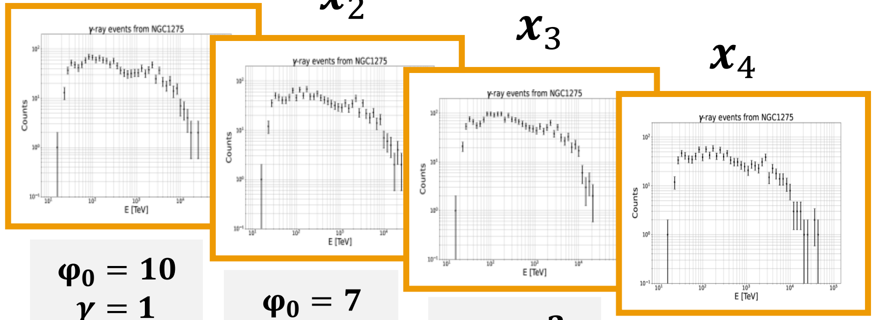
…

We can “trick” a network to learn the posterior implicitly

True

The observations are simulated according to the parameter values

$\boldsymbol{x}_1$ $\boldsymbol{x}_2$ $\boldsymbol{x}_3$ $\boldsymbol{x}_4$

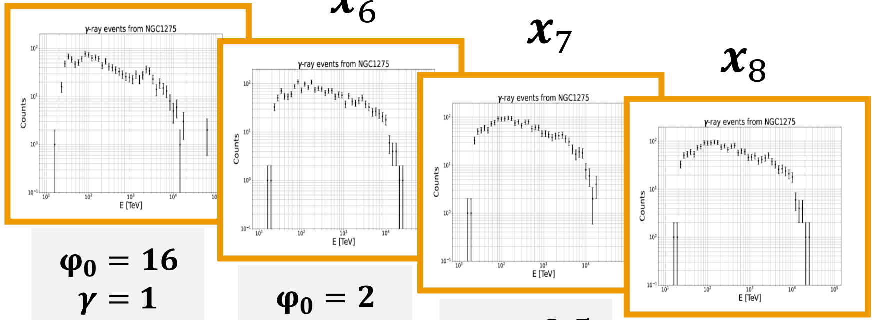


x_1	x_2	x_3	x_4	
$\varphi_0 = 10$ $\gamma = 1$ $E_{cut} = 3$	$\varphi_0 = 7$ $\gamma = 8$ $E_{cut} = 2$	$\varphi_0 = 2$ $\gamma = 4$ $E_{cut} = 3$	$\varphi_0 = 1.6$ $\gamma = 20$ $E_{cut} = 9$	…
True	**True**	**True**	**True**	

False

The parameter values are chosen independently from the simulations

$\boldsymbol{x}_5$ $\boldsymbol{x}_6$ $\boldsymbol{x}_7$ $\boldsymbol{x}_8$



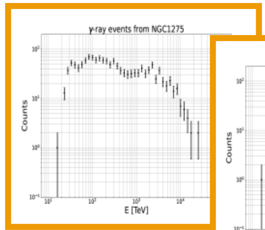
x_5	x_6	x_7	x_8	
$\varphi_0 = 16$ $\gamma = 1$ $E_{cut} = 5$	$\varphi_0 = 2$ $\gamma = 1$ $E_{cut} = 9$	$\varphi_0 = 2.5$ $\gamma = 18$ $E_{cut} = 2$	$\varphi_0 = 5$ $\gamma = 5.6$ $E_{cut} = 2$	…
False	**False**	**False**	**False**	

We can “trick” a network to learn the posterior implicitly

True

The observations are simulated according to the parameter values

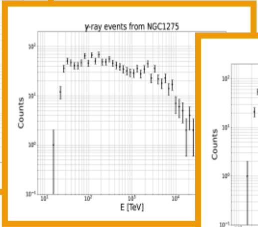
$\boldsymbol{x}_1$



$\varphi_0 = 10$
$\gamma = 1$
$E_{cut} = 3$

True

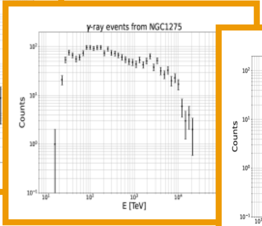
$\boldsymbol{x}_2$



$\varphi_0 = 7$
$\gamma = 8$
$E_{cut} = 2$

True

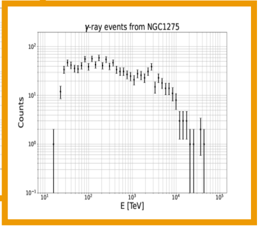
$\boldsymbol{x}_3$



$\varphi_0 = 2$
$\gamma = 4$
$E_{cut} = 3$

True

$\boldsymbol{x}_4$



$\varphi_0 = 1.6$
$\gamma = 20$
$E_{cut} = 9$

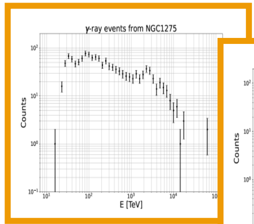
True

…

False

The parameter values are chosen independently from the simulations

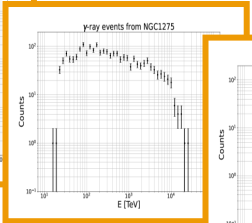
$\boldsymbol{x}_5$



$\varphi_0 = 16$
$\gamma = 1$
$E_{cut} = 5$

False

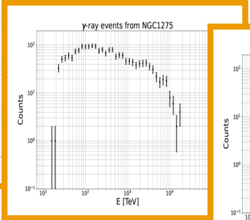
$\boldsymbol{x}_6$



$\varphi_0 = 2$
$\gamma = 1$
$E_{cut} = 9$

False

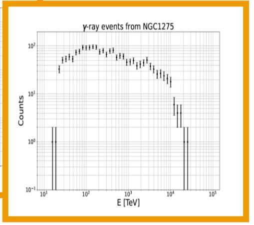
$\boldsymbol{x}_7$



$\varphi_0 = 2.5$
$\gamma = 18$
$E_{cut} = 2$

False

$\boldsymbol{x}_8$



$\varphi_0 = 5$
$\gamma = 5.6$
$E_{cut} = 2$

False

…

Training →

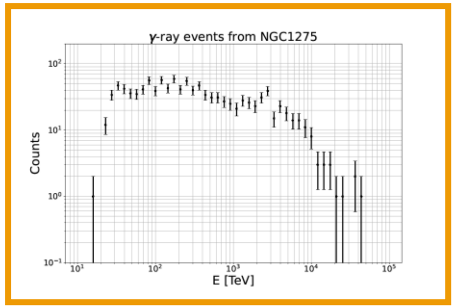
to distinguish between **True** and **False**

Neural Network

The network now has an implicit understanding of the posterior

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$



Input

Black Box

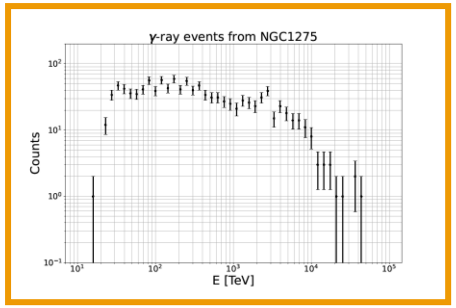
Neural Network

Input

$$\boldsymbol{\vartheta} = \begin{matrix} \varphi_0 = 10 \\ \gamma = 1 \\ E_{cut} = 3 \end{matrix}$$

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} = \begin{matrix} \boldsymbol{\varphi_0 = 10} \\ \boldsymbol{\gamma = 1} \\ \mathbf{E_{cut} = 3} \end{matrix}$

Input

Black Box

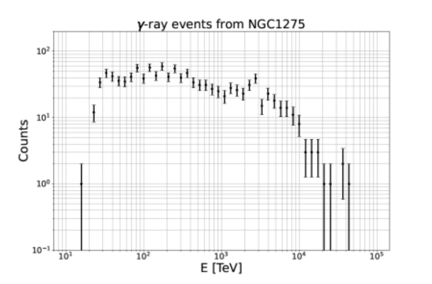
Neural Network

Output

$d \in (0\,,1)$

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} =$ $\begin{matrix} \boldsymbol{\varphi_0 = 10} \\ \boldsymbol{\gamma = 1} \\ \boldsymbol{E_{cut} = 3} \end{matrix}$

Input

Black Box

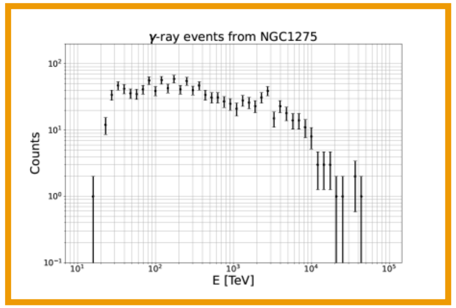
Neural Network

Output

$d \in (0\,,1)$

False **True**

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$ 

$\boldsymbol{\vartheta} =$ $\varphi_0 = 10$, $\gamma = 1$, $E_{cut} = 3$

Input

Input

Black Box

Neural Network

Output

$d \in (0\,,1)$

False

True

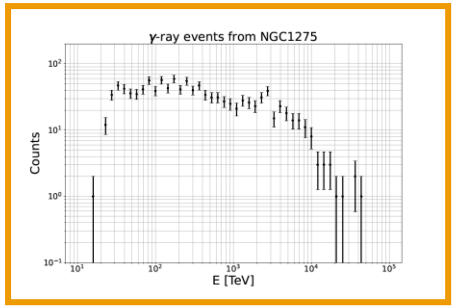
Mathematical magic

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{d}{1-d}\,p(\boldsymbol{\vartheta})$$

"Likelihood-ratio trick"

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} =$ $\boldsymbol{\varphi_0 = 10}$, $\boldsymbol{\gamma = 1}$, $\boldsymbol{E_{cut} = 3}$

Input

Black Box

Neural Network

Output

$d \in (0\,,1)$

False ⟷ True

Mathematical magic

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{d}{1-d}\,p(\boldsymbol{\vartheta})$$

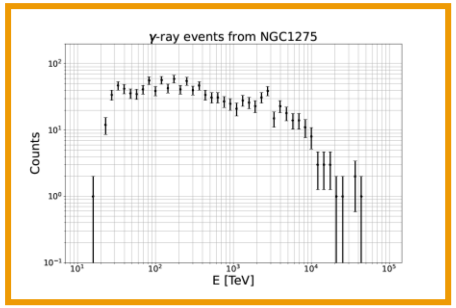
"Likelihood-ratio trick"

Conditions for the trick to work:

- When generating the training set, we have to draw all of the parameters of interest from the prior distribution.
- The network's loss function must be the binary cross entropy.

The network now has an implicit understanding of the posterior

$\boldsymbol{x} =$



Input

$\boldsymbol{\vartheta} =$ $\begin{array}{c} \boldsymbol{\varphi_0 = 10} \\ \boldsymbol{\gamma = 1} \\ \mathbf{E_{cut} = 3} \end{array}$

Input

Black Box

Neural Network

Output

$d \in (0\,,1)$

False

True

Mathematical magic

$$p(\boldsymbol{\vartheta}|\boldsymbol{x}) = \frac{d}{1-d}\,p(\boldsymbol{\vartheta})$$

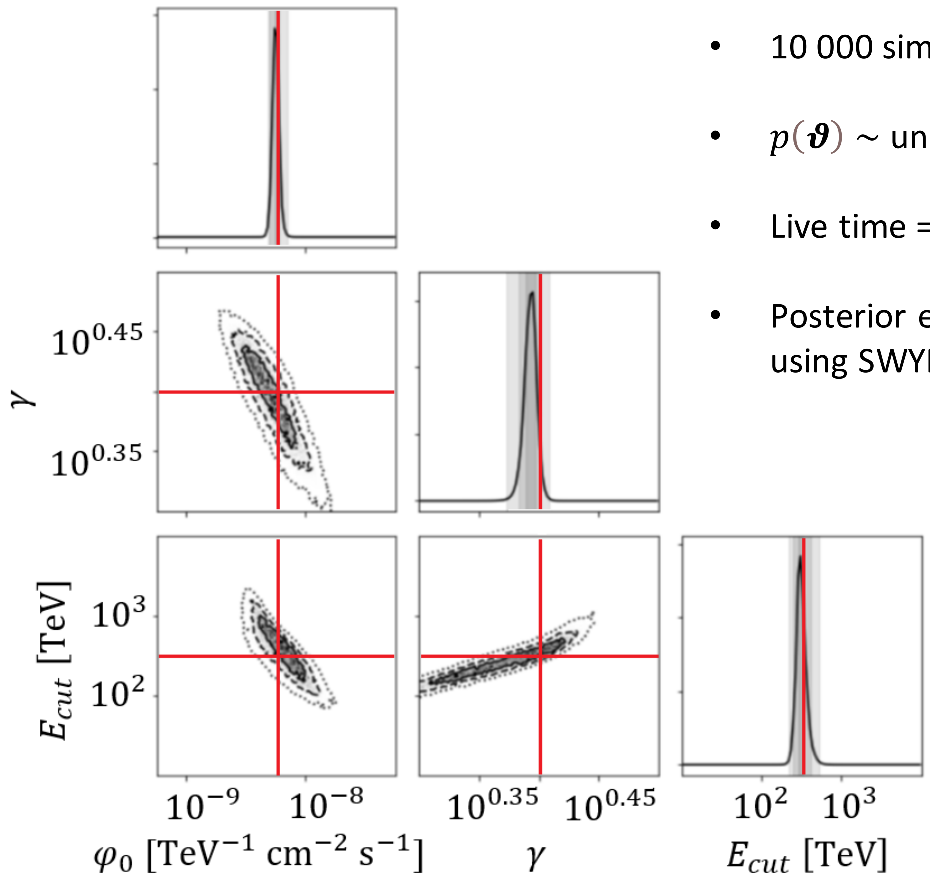
"Likelihood-ratio trick"

Vary this input
to scan the posterior
over parameter space

Conditions for the trick to work:

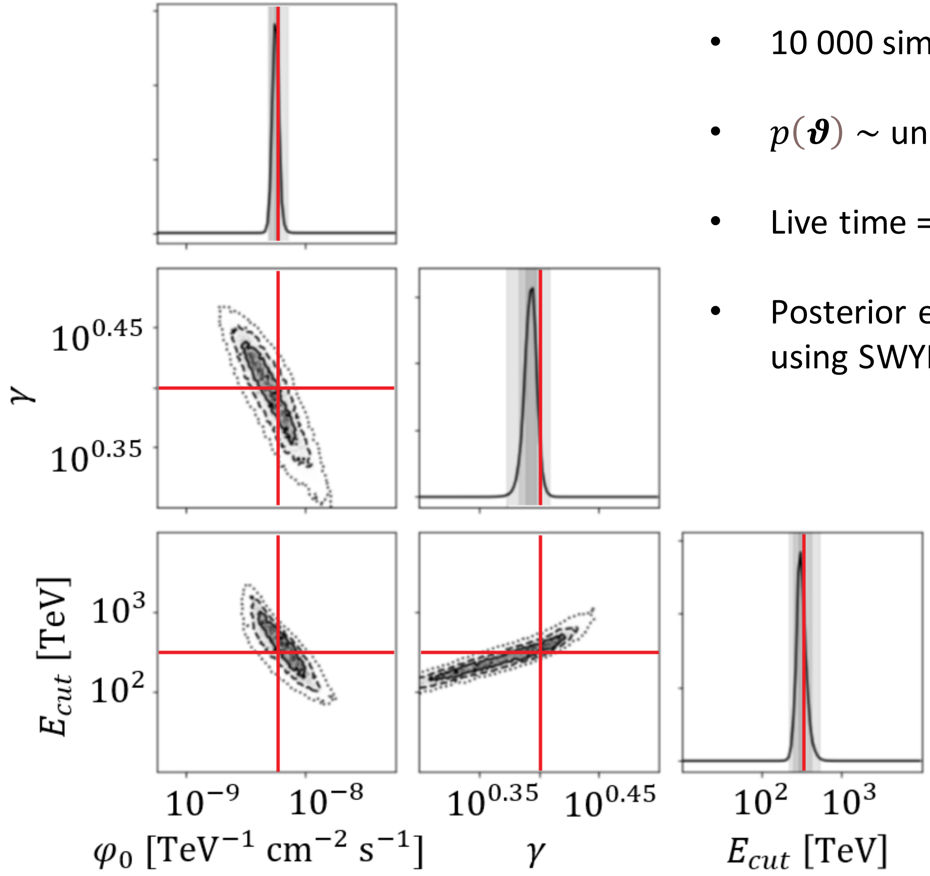
- When generating the training set, we have to draw all of the parameters of interest from the prior distribution.
- The network's loss function must be the binary cross entropy.

Inference with NRE seems to be precise for the spectral fit

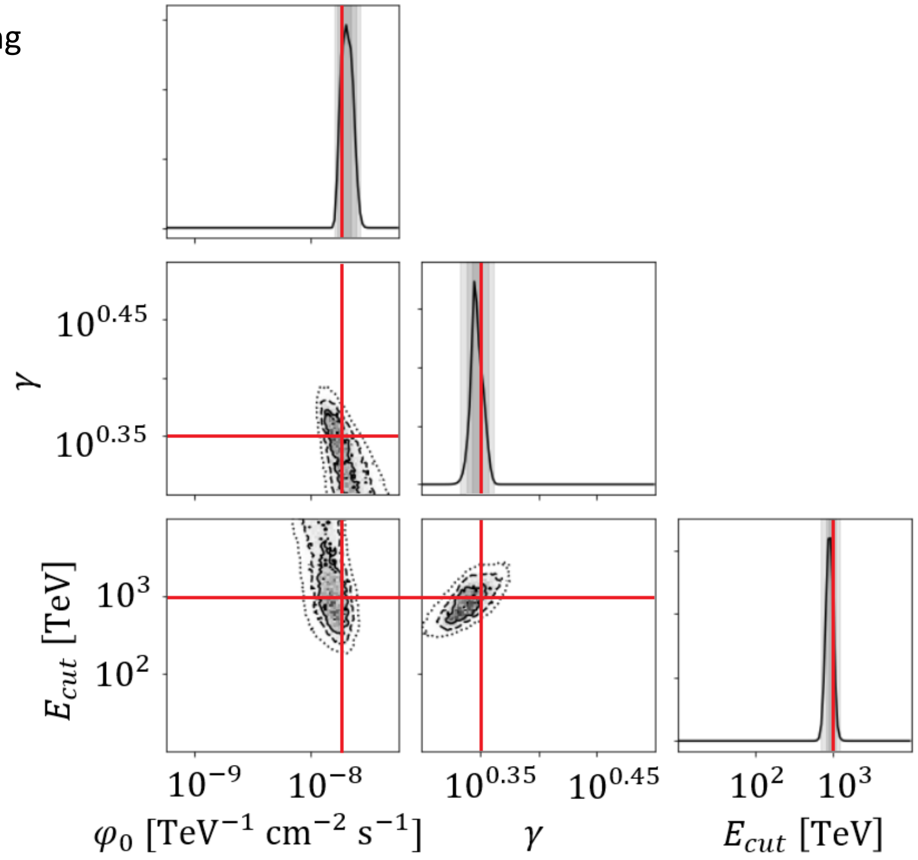


- 10 000 simulations used in training
- $p(\boldsymbol{\vartheta}) \sim$ uniform on log scale
- Live time = 50 hr'
- Posterior estimation performed using SWYFT [1]

Inference with NRE seems to be precise for the spectral fit

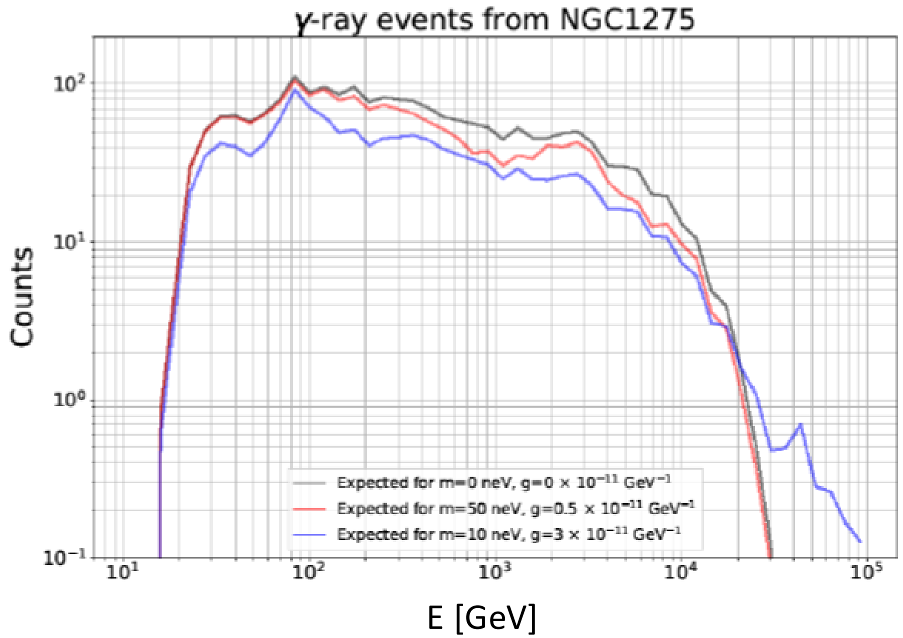


- 10 000 simulations used in training
- $p(\boldsymbol{\vartheta})$ ~ uniform on log scale
- Live time = 50 hr'
- Posterior estimation performed using SWYFT [1]



NRE seems particularly useful for ALP searches with gamma-telescopes

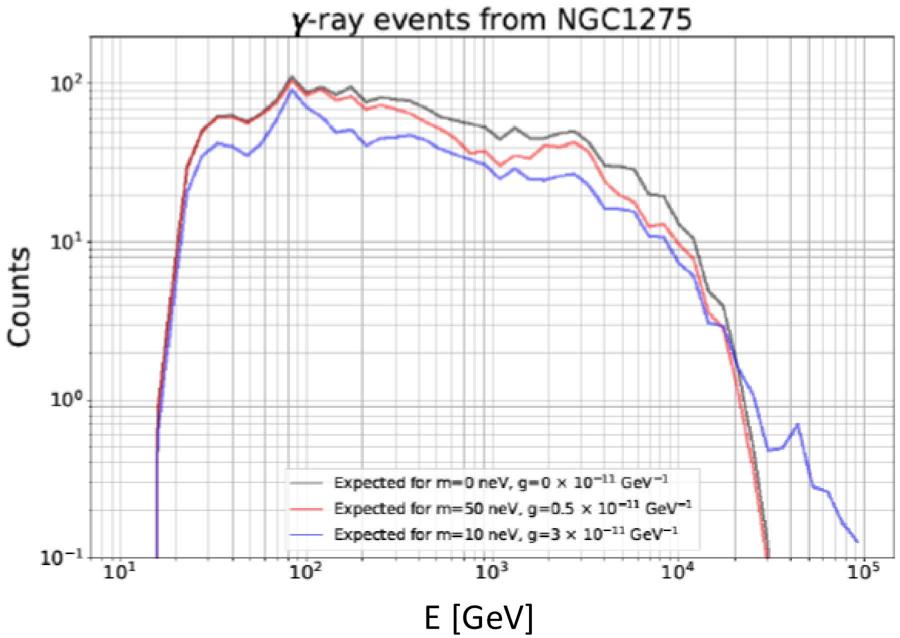
Expected result assuming ALPs with given mass *m* and coupling *g* (using `gammaALPs`):



The expected spectrum can be simulated using `gammapy[2]` **and** `gammaALPs[3]` **(by M. Meyer)**

NRE seems particularly useful for ALP searches with gamma-telescopes

Expected result assuming ALPs with given mass *m* and coupling *g* (using `gammaALPs`):

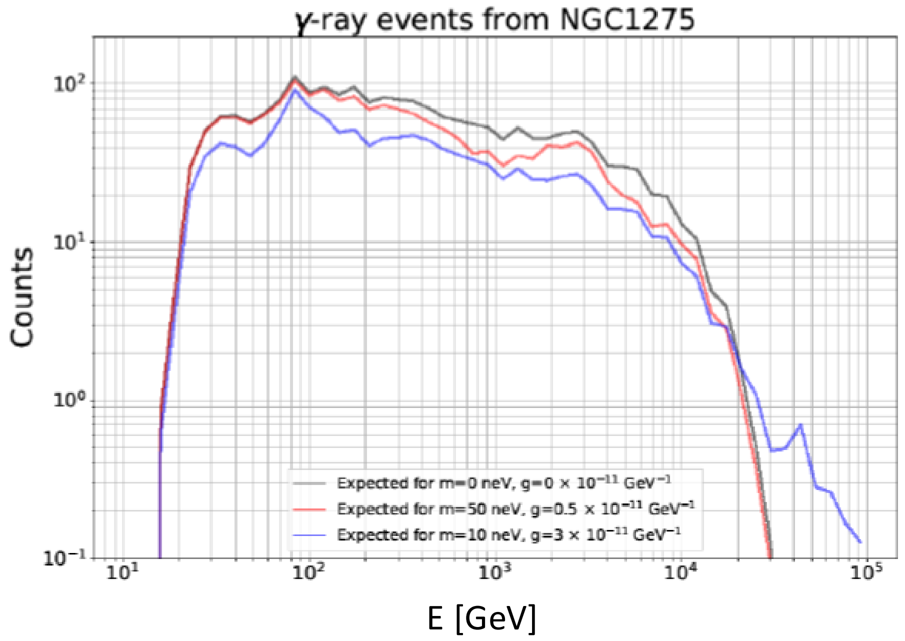


The expected spectrum can be simulated using `gammapy[2]` and `gammaALPs[3]` (by M. Meyer)

- Parameters of interest:
  - ➢ Mass of ALPs, *m*
  - ➢ ALP-photon coupling, *g*

NRE seems particularly useful for ALP searches with gamma-telescopes

Expected result assuming ALPs with given mass *m* and coupling *g* (using `gammaALPs`):

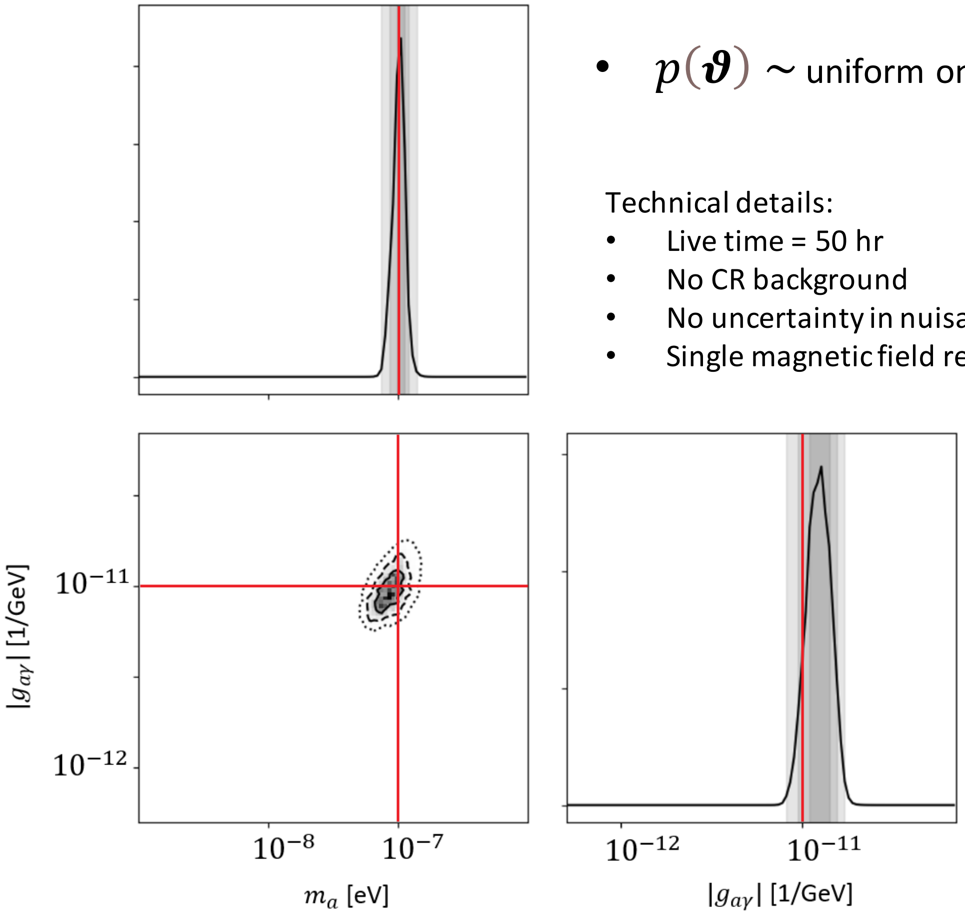


The expected spectrum can be simulated using `gammapy[2]` **and** `gammaALPs[3]` **(by M. Meyer)**

- Parameters of interest:
  - Mass of ALPs, *m*
  - ALP-photon coupling, *g*

- Nuisance parameters:
  - Amplitude
  - Spectral index
  - Cut-off energy
  - Magnetic field configuration
  - + 12 more related to configuration of NGC1275

Preliminary results indicate the method is suitable for ALP searches

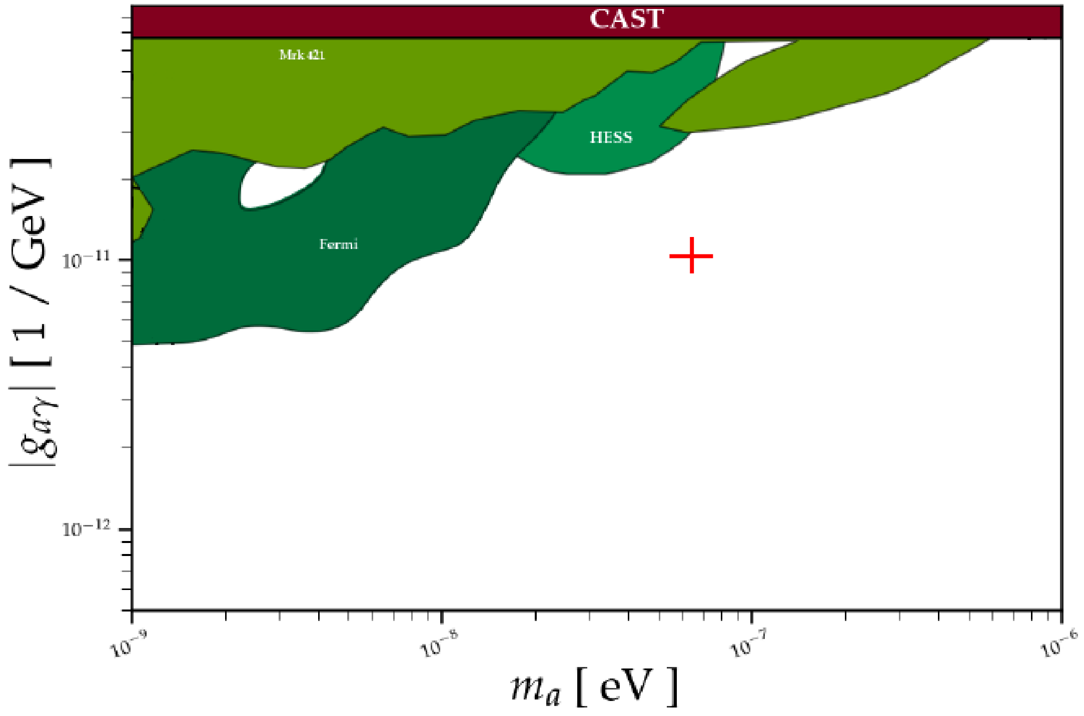


- $p(\boldsymbol{\vartheta}) \sim$ uniform on log scale

Technical details:

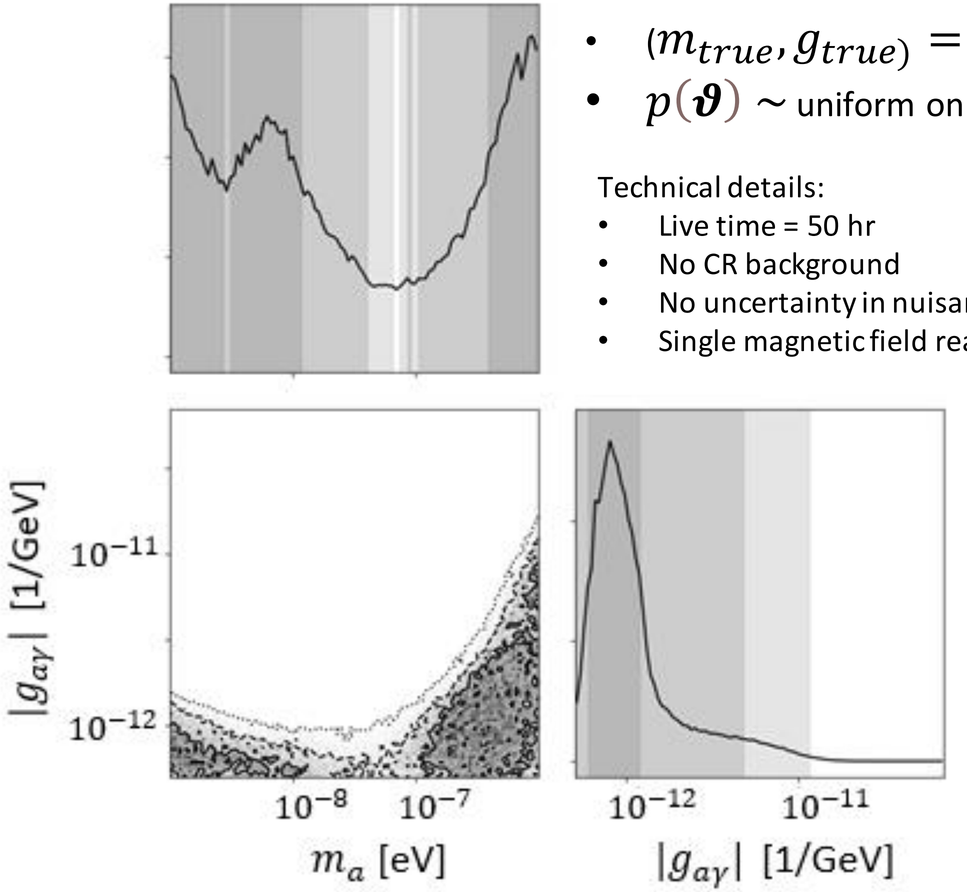
- Live time = 50 hr
- No CR background
- No uncertainty in nuisance parameters
- Single magnetic field realization

Very unrealistic!



Full limits: [4]

Preliminary results indicate the method is suitable for ALP searches

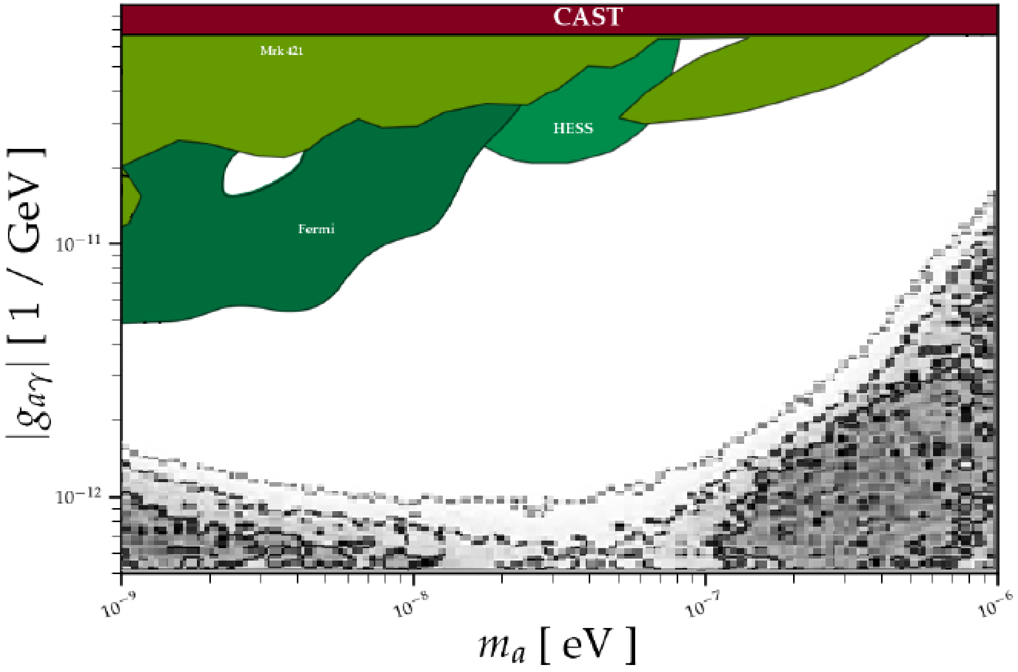


- $(m_{true}, g_{true}) = (0,0)$
- $p(\boldsymbol{\vartheta}) \sim$ uniform on log scale

Technical details:

- Live time = 50 hr
- No CR background
- No uncertainty in nuisance parameters
- Single magnetic field realization

Very unrealistic!



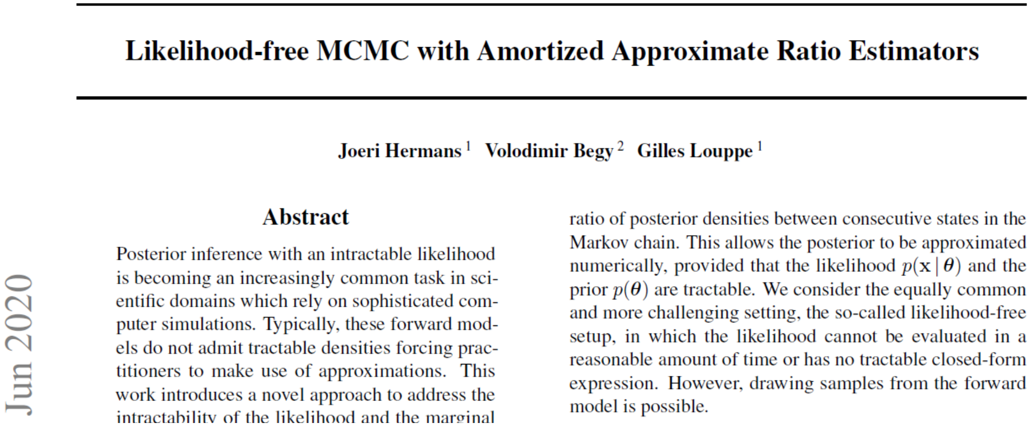
Full limits: [4]

Neural Ratio Estimation: some resources to get you started

- Original paper (to my knowledge) to introduce the concept:
  https://arxiv.org/abs/1903.04057

- B. K. Miller, A. Cole, P. Forŕ e, G. Louppe, and C. Weniger, "Truncated marginal neural ratio estimation". https://arxiv.org/abs/2107.01214

- B. K. Miller, A. Cole, G. Louppe, and C. Weniger, "Simulation-efficient marginal posterior estimation with swyft: stop wasting your precious time," https://arxiv.org/abs/2011.13951



Likelihood-free MCMC with Amortized Approximate Ratio Estimators

Joeri Hermans [1] Volodimir Begy [2] Gilles Louppe [1]

Jun 2020

Abstract

Posterior inference with an intractable likelihood is becoming an increasingly common task in scientific domains which rely on sophisticated computer simulations. Typically, these forward models do not admit tractable densities forcing practitioners to make use of approximations. This work introduces a novel approach to address the intractability of the likelihood and the marginal ratio of posterior densities between consecutive states in the Markov chain. This allows the posterior to be approximated numerically, provided that the likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$ are tractable. We consider the equally common and more challenging setting, the so-called likelihood-free setup, in which the likelihood cannot be evaluated in a reasonable amount of time or has no tractable closed-form expression. However, drawing samples from the forward model is possible.

- SWYFT (a package under development that does NRE and more): https://github.com/undark-lab/swyft

One take away

One take away

If you:

1) Want to infer parameters

One take away

If you:

1) Want to infer parameters

2) Want to avoid simplifications

One take away

If you:

1) Want to infer parameters
2) Want to avoid simplifications
3) Can make many simulations

One take away

If you:

1) Want to infer parameters
2) Want to avoid simplifications
3) Can make many simulations
4) Are not a die-hard frequentist

One take away

If you:

1) Want to infer parameters
2) Want to avoid simplifications
3) Can make many simulations
4) Are not a die-hard frequentist

Then Neural Ratio Estimation is your friend!

References

[1] SWYFT: https://github.com/undark-lab/swyft

[2] Gammapy: https://gammapy.org/

[3] gammaALPs: https://github.com/me-manu/gammaALPs

[4] ALP limits: https://github.com/cajohare/AxionLimits

Backup

Derivation of the likelihood ratio trick

When the neural network is trained, it is actually trying to minimize a loss function. Different loss functions are possible, but for the likelihood ratio trick to work, it is necessary that we use the *Binary Cross Entropy*, which looks like this:

$$\mathrm{LOSS} = -\sum\left[\quad y\ln d + (1-y)\ln(1-d)\quad\right] \quad \text{(Summed over all training samples)}$$

- $y = 1$ if training sample is from *True* category, 0 if *False*.
- $d =$ output from neural network for given sample input

Notice that the loss becomes smaller when the network manages to categorize more samples correctly. In the limit of infinite training samples, the loss becomes

$$\mathrm{LOSS} \to \iint\left[\quad \underbrace{p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})}_{\text{Probability of appearance of a sample of the } True \text{ category (because } \boldsymbol{x} \text{ is simulated taking } \boldsymbol{\vartheta} \text{ as a premise)}}\ln d \quad + \quad \underbrace{p(\boldsymbol{x})p(\boldsymbol{\vartheta})}_{\text{Probability of appearance of a sample of the } False \text{ category (because } \boldsymbol{x} \text{ and } \boldsymbol{\vartheta} \text{ are independent)}}\ln(1-d)\quad\right]\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{\vartheta}$$

We assume that the network manages to optimize the loss function perfectly. If this is the case, the derivative of the loss with respect to the neural network's hyperparameters (weights and biases, which are adjusted during training), which we denote by $\boldsymbol{\varphi}$, is 0. Notice that in the integrand, only d is dependent on $\boldsymbol{\varphi}$.

$$\frac{\partial}{\partial\boldsymbol{\varphi}}LOSS = \iint\left[\quad \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta})}{d} \quad - \quad \frac{p(\boldsymbol{x})p(\boldsymbol{\vartheta})}{1-d}\right]\frac{\partial d}{\partial\boldsymbol{\varphi}} \quad = \quad 0$$

The likelihood ratio trick follows from setting the square brackets to zero:

$$\Longrightarrow \quad \frac{d}{1-d} = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\boldsymbol{\vartheta})}{\int \mathrm{d}\boldsymbol{\theta}\; p(\boldsymbol{x}|\boldsymbol{\theta})}$$

Workflow

Workflow:

Define a function that outputs a simulated observation
(as a function of parameters of interest and nuisance parameters)

Using `gammapy`

⇩

Simulate enough observations to train a neural network
(according to a defined prior)

⇩

Train a neural network

⇩

Scan the parameter space using the neural network

Using `SWYFT` [1,2]

- The simulations implicitly contain the information on the relationship between parameters and observations