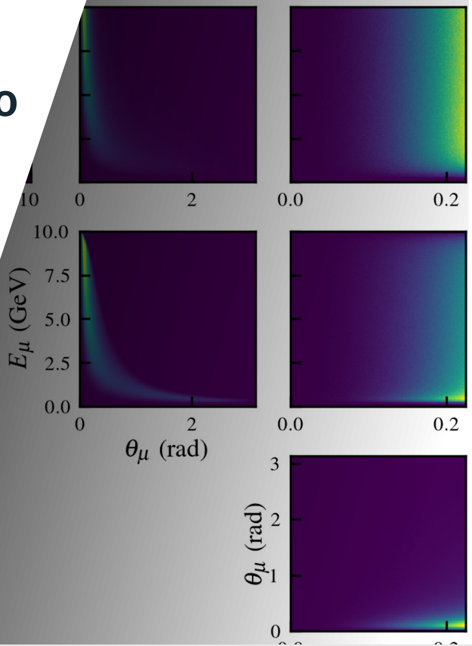
Exhaustive Neural Importance Sampling applied to Monte Carlo event generation

NuFact 2021



Sebastian Pina-Otey*, Federico Sánchez,
Thorsten Lux and Vicens Gaitan
*sebastian.pinaotey@unige.ch

UNIVERSITÉ DE GENÈVE

GRUPO AIA

IFAE
Institut de Física d'Altes Energies

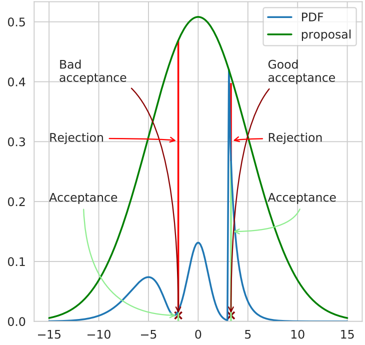
September 8, 2021

Contents

Contents

Rejection sampling

- A numerical method for sampling from an analytical PDF.
- Samples generated via a **similar proposal function** $q(\mathbf{x})$, a PDF which can be **both evaluated and sampled from.**
- Proposal function multiplied by a constant $k \geq 1$ such that $p(\mathbf{x}) \leq k \cdot q(\mathbf{x}), \forall \mathbf{x}$.
- $x \sim q(\mathbf{x})$ is accepted with probability $p(\mathbf{x}) / (k \cdot q(\mathbf{x}))$.



Goal

Find suitable proposal $q(\mathbf{x})$ for rejection sampling for better efficiency.

Main issues

1. Designing a suitable proposal function can be very costly in human time.
2. Generic proposal functions, e.g. a uniform distribution, makes the algorithm usually very inefficient.
3. The inefficiency grows rapidly with the number of dimensions.

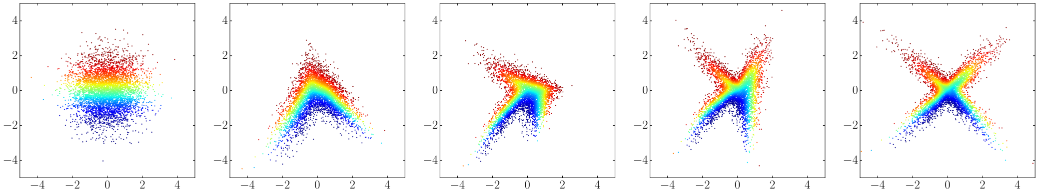
Normalizing flows proposal

1. Adapts to a given target density automatically.
   - Barely human time cost.
   - Good acceptance efficiency.
2. Grows properly with the number of dimensions.
3. Produces exact samples through rejection sampling.

Contents

Normalizing flows

- Define a **transformation** T_ϕ from a complex **target density** $q_\phi(x)$ to a simple **base density** $f(u)$: $u = T_\phi(x)$.
- T_ϕ is **invertible** and **differentiable**, and satisfies: $q_\phi(x) = f(T_\phi(x))|\det J_T(x)|$
- $f(u)$ can be evaluated and sampled from.
- T_ϕ allows to sample and evaluate from $q_\phi(x)$ using $f(u)$ via T_ϕ^{-1}.
- Example of transforming from $f(u)$ Gaussian to $q_\phi(x)$ in star shape.



Source: arXiv:1912.02762

Standard problem and objective function

- Standard problem:
  Given data $x \sim p(x)$, find $q_\phi(x) \approx p(x)$ with only samples.
- **How?** Minimizing the **Kullback-Leibler divergence**:

$$D_{\mathrm{KL}}\left(p(x) \| q_\phi(x)\right) = \int p(x) \log\left(\frac{p(x)}{q_\phi(x)}\right)\, dx.$$

$$\begin{aligned}\arg\min_\phi D_{\mathrm{KL}}\left(p(x) \| q_\phi(x)\right) &= \arg\min_\phi - \int p(x) \log q_\phi(x)\, dx \\ &\approx \arg\max_\phi \sum \log q_\phi(x) \text{ with } x \sim p(x).\end{aligned}$$

Contents

Modifying Neural Importance Sampling

- To minimize $D_{\text{KL}}\left(p\left(\mathbf{x}\right)\|q_\phi\left(\mathbf{x}\right)\right)$, Müller et al. (1808.03856) propose to use the gradient

$$\frac{1}{N}\sum_{i=1}^{N} w(\mathbf{x}_i)\nabla_\phi \log q_\phi\left(\mathbf{x}_i\right), \ \mathbf{x}_i \sim q_\phi\left(\mathbf{x}_i\right) \text{ and } w(\mathbf{x}_i) = \frac{p\left(\mathbf{x}_i\right)}{q_\phi\left(\mathbf{x}_i\right)}. \tag{1}$$

- We propose to additionally redefine the target density with a background (e.g., uniform):

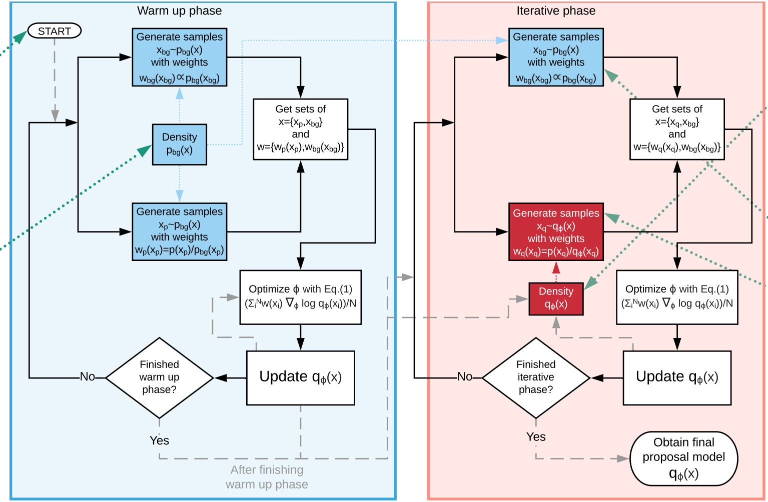
$$p_{\text{target}}\left(\mathbf{x}\right) = \left(1-\alpha\right)\cdot p\left(\mathbf{x}\right) + \alpha \cdot p_{\text{bg}}\left(\mathbf{x}\right).$$

  Aim:
  - **Improve initial training**, ensuring the full support of $p\left(\mathbf{x}\right)$ (better than randomly initialized NF).
  - **Ensure exhaustive coverage** of the phase space.

ENIS general scheme

NF initialized randomly

Uses only $p_{bg}(x)$ to sample



NF is trained

Uses both $p_{bg}(x)$ and mainly $q_\phi(x)$ to sample

Contents

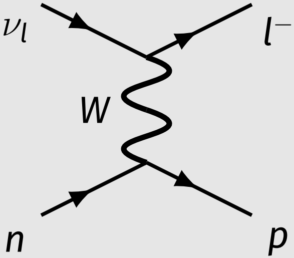
CCQE cross section

- **Charged-Current Quasi-Elastic (CCQE) interaction:**

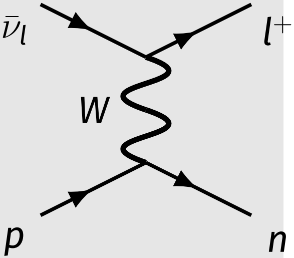
$$\nu_l + n \rightarrow l^- + p$$
$$\bar{\nu}_l + p \rightarrow l^+ + n$$

- **Cross section** is the **probability** of a specific process taking place:
    - Cross section of a CCQE interaction.

Feynman diagrams:



$\bar{\nu}_l$ CCQE scattering.

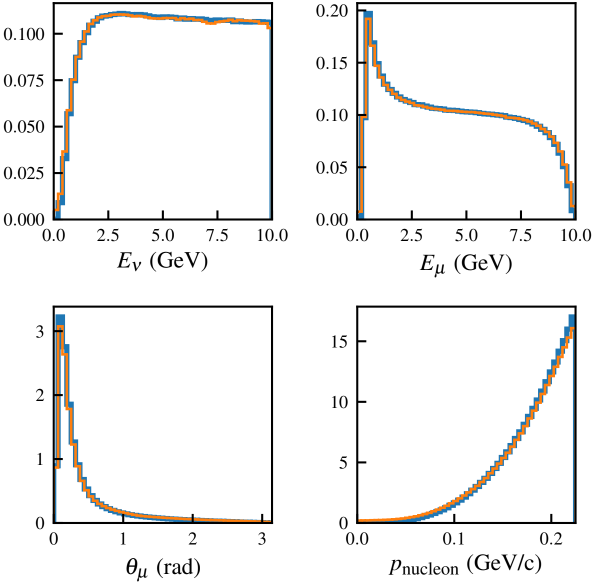


$\bar{\nu}_l$ CCQE scattering.

- Variables: $E_\nu, E_l, \theta_l, p_{\text{nucleon}}$

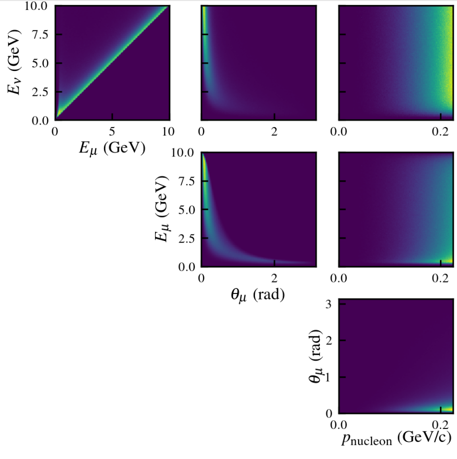
True vs proposal 1D

- Original target in blue.
- NF proposal in orange.

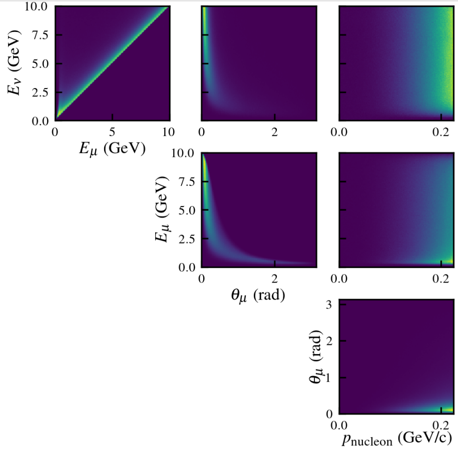


True vs proposal 2D

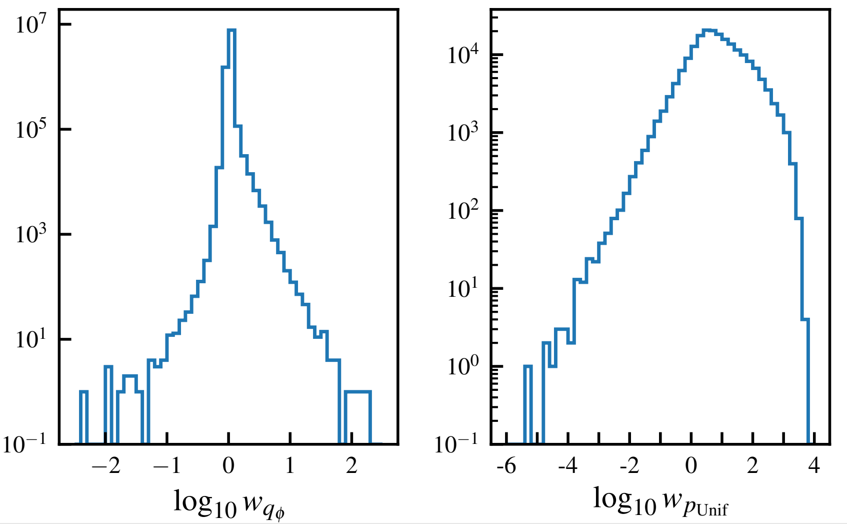
Marginalized $p(\mathbf{x})$ density



Marginalized $q_\phi(\mathbf{x})$ density



Weights $w_q(x) = p(x)/q_\phi(x)$ distributions for rejection sampling



Coverage and weight distribution

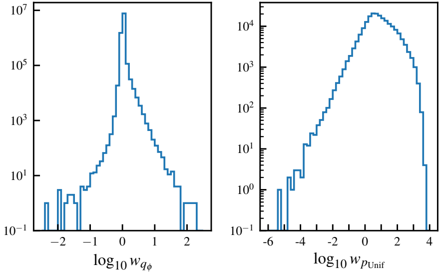
Coverage

- To perform rejection sampling, we need

$$k \cdot q\left(\mathbf{x}\right) \geq p\left(\mathbf{x}\right) \quad \forall\, \mathbf{x} : p\left(\mathbf{x}\right) > 0.$$

- Relax k with Q-quantile of weights $p\left(\mathbf{x}\right)/q_{\phi}\left(\mathbf{x}\right)\, w_{\mathrm{Q}}$, denoted by $k_{\mathrm{Q}} = (Q\text{-quantile}(w))^{-1} = w_{\mathrm{Q}}^{-1}$, to improve p_{accept}.
- Define coverage with the new k_{Q}:

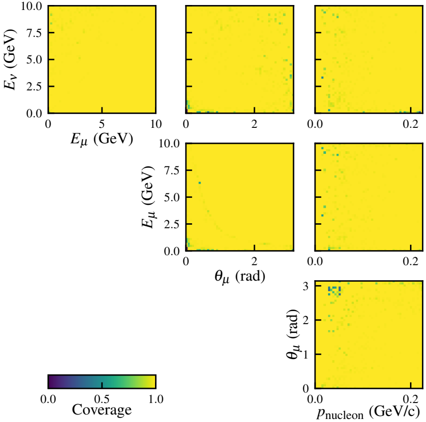
$$\text{Coverage} = \frac{\sum_{i=1}^{N} w'(\mathbf{x}_i)}{\sum_{i=1}^{N} w(\mathbf{x}_i)}.$$



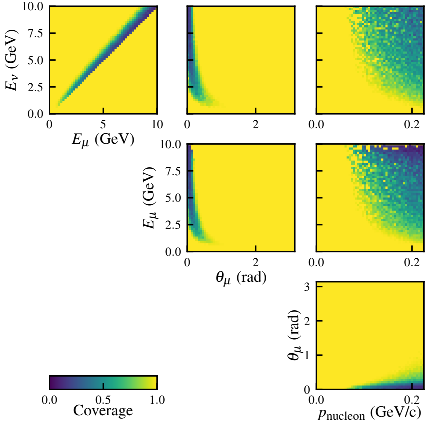
Quantile	Prop.	p_{accept}	Coverage
1.000	NSF	0.0051	1.0000
	Unif.	0.0002	1.0000
0.999	NSF	0.3590	0.9984
	Unif.	0.0027	0.6185
0.981	NSF	0.7968	0.9920
	Unif.	0.0193	0.0039

Marginalized coverage

Marginalized coverage of $q_\phi(\mathbf{x})$



Marginalized coverage of $p_{\text{Unif}}(\mathbf{x})$



Conclusions

- Utilize normalizing flows to find suitable proposal functions to perform rejection sampling.
  - Finds automatically a good proposal function.
  - Exact sampling (corrects inefficiencies of the flow).
- Propose redefining target with background:
  - Improve initial training.
  - Ensure exhaustive coverage.
- Study the possibility of relaxing constrain on rejection sampling through the concept of coverage (see backup).
- Compare it to generic proposal, the uniform distribution, on a simple 4D cross section.

Exhaustive Neural Importance Sampling applied to Monte Carlo event generation,

S. Pina-Otey, F. Sanchez, T. Lux and V. Gaitan,

Phys. Rev. D 102, 013003 (2020).

THANK YOU!

Backup slides

Normalizing Flows: Transformation

- Transformation T is partially defined through a Neural Network.
- T is usually **broken down into simpler transformations**:

$$T = T_K \circ \cdots \circ T_1.$$

- Taking $z_0 = x$ and $z_K = u$:

$$z_k = T_k(z_{k-1}),\ k = 1 : K,$$

$$|\det J_T(x)| = \left|\det \prod_{k=1}^{K} J_{T_k}(z_{k-1})\right|.$$

- We will consider a single transformation $T(x) = u$.

Normalizing Flows: Autoregressive transformation

- $|\det J_T(x)|$ has to be **easy to compute.**
  - Idea: **Autoregressive transformations:**

$$u_i = \tau(x_i; \mathbf{h}_i) \text{ with } \mathbf{h}_i = c_i(x_{<i}; \phi), \quad x_{<i} = x_{1:i-1}.$$

- **Transformer** $\tau : \mathbb{R} \to \mathbb{R}$ is bijective and differentiable, usually a **monotone function.**
- $\mathbf{h}_i$ are the parameters of these transformers for each component i.
- $c_i(x_{<i}; \phi)$ is the **conditioner** for the i-th component, usually a NN of parameters ϕ.
- All conditioners can be computed at ones efficiently using a **Masked Autoregressive Neural Network.**
- $J_T(x)$ is now a triangular matrix, hence $|\det J_T(x)|$ is the product of the diagonal.

Normalizing Flows: Masked Autoregressive Flow

- Simplest transformer, a linear one:

$$\tau(x_i; \alpha_i, \beta_i) = x_i \exp \alpha_i + \beta_i.$$

- Conditioner introduces non-linearities of the density $q(x)$:

$$f_{\alpha_i}(x_{<i}; \phi_\alpha) = \alpha_i; \quad f_{\beta_i}(x_{<i}; \phi_\beta) = \beta_i.$$

- Jabobian is trivial to compute:

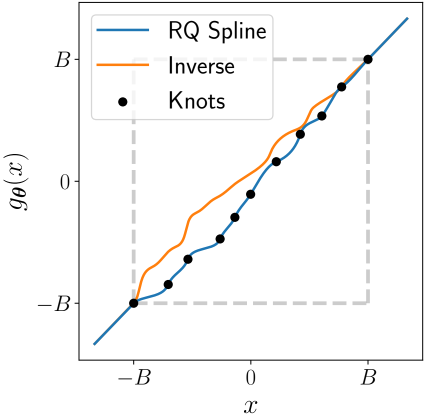
$$|\det J_T(x)| = \exp\left(\sum_i \alpha_i\right).$$

G. Papamakarios et al., NeurIPS 2017

Normalizing Flows: Neural Spline Flow

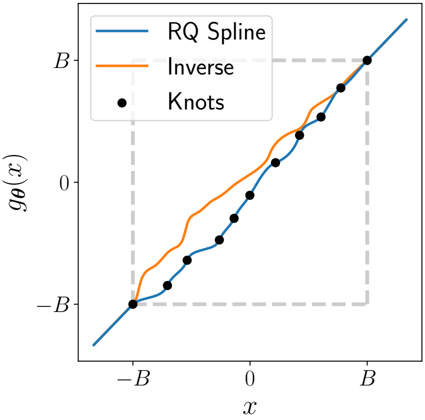
- Transformers are **rational quadratic monotonic splines**.
  - Very flexible, infinite Taylor series.
  - Easily differentiable.
  - Analytically invertible.
- Parameters of transformer:
  - Position of knots.
  - Derivative of knots.

C. Durkan et al., NeurIPS 2019

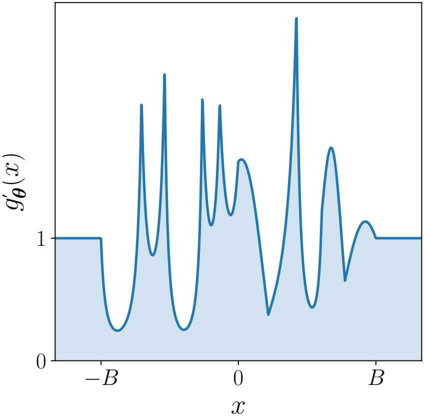


Normalizing Flows: Neural Spline Flow

Spline:



Derivative:

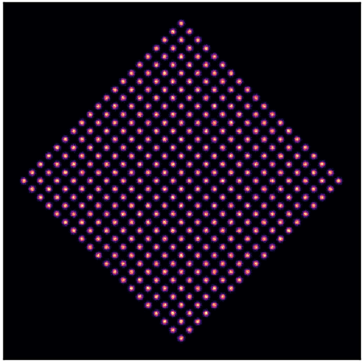


C. Durkan et al., NeurIPS 2019

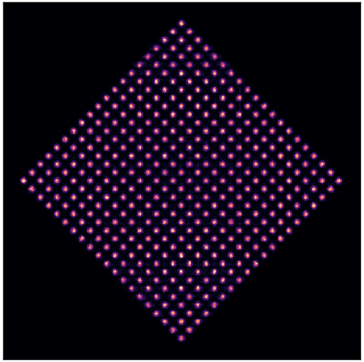
Normalizing Flows: Neural Spline Flow

Example (I) of samples of $p(x)$ vs samples of $q_\phi(x)$:

Data:



NSF:



C. Durkan et al., NeurIPS 2019

Normalizing Flows: Neural Spline Flow

Example (II) of samples of $p(x)$ vs samples of $q_\phi(x)$:

Data:

NSF:

C. Durkan et al., NeurIPS 2019

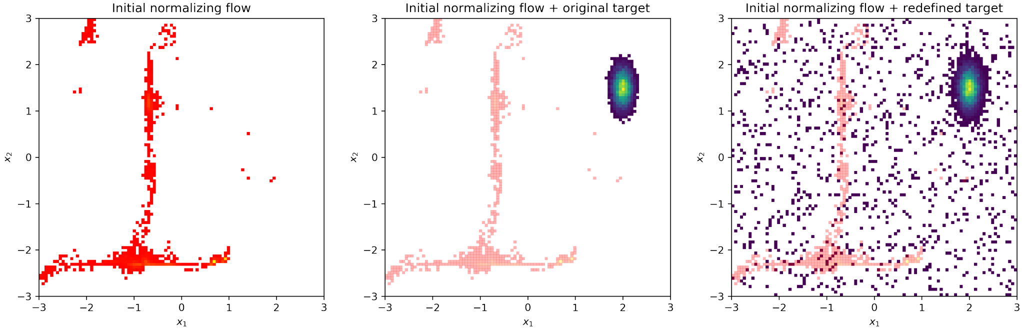
ENIS algorithm

1. Warm-up phase:
   (i) Sample $\mathbf{x}_p \sim p_{\text{bg}}(\mathbf{x})$ and compute their weights $w_p(\mathbf{x}_p) = p(\mathbf{x}_p)/p_{\text{bg}}(\mathbf{x}_p)$.
   (ii) Sample background $\mathbf{x}_{\text{bg}} \sim p_{\text{bg}}(\mathbf{x})$ with associated weights $w_{\text{bg}}(\mathbf{x}_{\text{bg}}) = C_{w_{\text{bg}}} \cdot p_{\text{bg}}(\mathbf{x}_{\text{bg}})$, where $C_{w_{\text{bg}}} = \frac{\alpha}{1-\alpha} \frac{\langle w_p(\mathbf{x}_p)\rangle}{\langle p_{\text{bg}}(\mathbf{x}_{\text{bg}})\rangle}$.
   (iii) Optimize the parameters of $q_\phi(\mathbf{x})$ via Eq. (1) using $\mathbf{x} = \{\mathbf{x}_p, \mathbf{x}_{\text{bg}}\}$ with weights $w(\mathbf{x}) = \{w_p(\mathbf{x}_p), w_{\text{bg}}(\mathbf{x}_{\text{bg}})\}$.
2. Iterative phase:
   (i) Sample $\mathbf{x}_q \sim q_\phi(\mathbf{x})$ and compute their weights $w_q(\mathbf{x}_q) = p(\mathbf{x}_q)/q_\phi(\mathbf{x}_q)$.
   (ii) Sample background $\mathbf{x}_{\text{bg}} \sim p_{\text{bg}}(\mathbf{x})$ with associated weights $w_{\text{bg}}(\mathbf{x}_{\text{bg}}) = C'_{w_{\text{bg}}} p_{\text{bg}}(\mathbf{x}_{\text{bg}})$, where $C'_{w_{\text{bg}}} = \frac{\alpha}{1-\alpha} \frac{\langle w_q(\mathbf{x}_q)\rangle}{\langle p_{\text{bg}}(\mathbf{x}_{\text{bg}})\rangle}$.
   (iii) Optimize the parameters of $q_\phi(\mathbf{x})$ via Eq. (1) using $\mathbf{x} = \{\mathbf{x}_q, \mathbf{x}_{\text{bg}}\}$ with weights $w(\mathbf{x}) = \{w_q(\mathbf{x}_q), w_{\text{bg}}(\mathbf{x}_{\text{bg}})\}$.

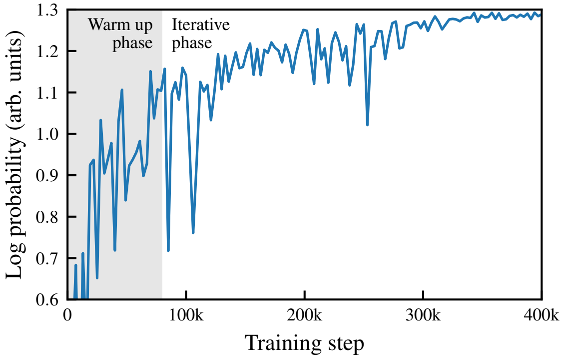
S. Pina-Otey et al., Phys. Rev. D 102, 013003 (2020)

Visualization of modification

- Visualization of full support modification of target density.
  - No overlap in the original target $\rightarrow$ no gradient.
  - Redefined target does overlap $\rightarrow$ gradients.



Proposal training validation loss



S. Pina-Otey et al., Phys. Rev. D 102, 013003 (2020)

- 400k training steps.
- 5 flow steps, depth 2 of transforming blocks.
- 32 hidden units per layer and 8 bins for the splines.
- $37\,220$ learnable parameters.
- 0.0005 learning rate and batch size of 5k.
- 200k samples for validation every 1k steps.