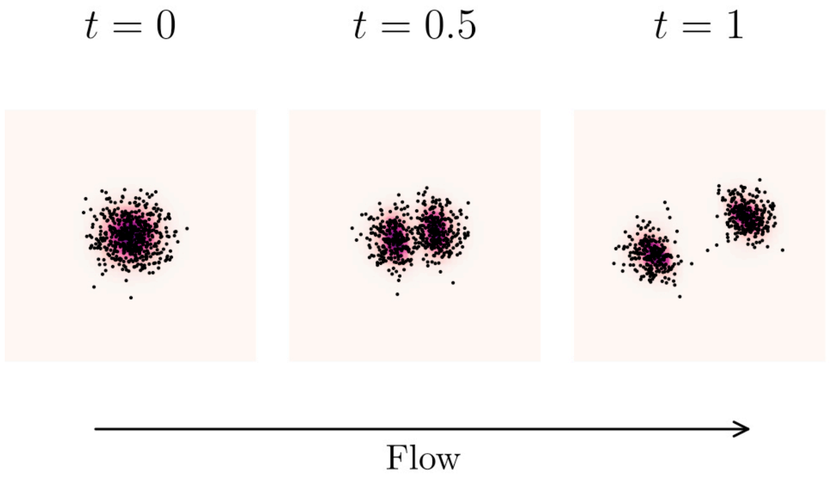
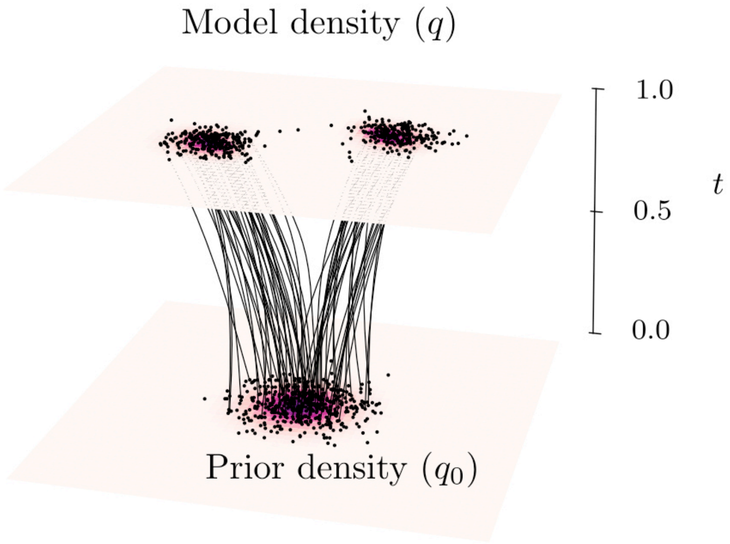
Normalizing flows for Lattice gauge theory





Gurtej Kanwar
Chancellor's Fellow in AI & Datascience
University of Edinburgh

Jul 28 - Aug 1, 2025
MITP Summer School

Lecture 2

Recap:

MCMC is almost universally affected by **critical slowing down / topo freezing**.

Generative models have the potential to circumvent the Markov chain.

Normalizing flows in particular provide:

- Direct sampling: (1) draw $V \sim r(V)$, (2) apply flow $U = f(V)$
- Evaluation of model log density

$$q(U) = r(V) \, |\det \partial U / \partial V|^{-1} = r(V) \, J_f(V)^{-1}$$

- Reweighting from samples $U \sim q(U)$ to target $p(U) = e^{-S(U)}/Z$

$$w(U) = p(U)/q(U)$$

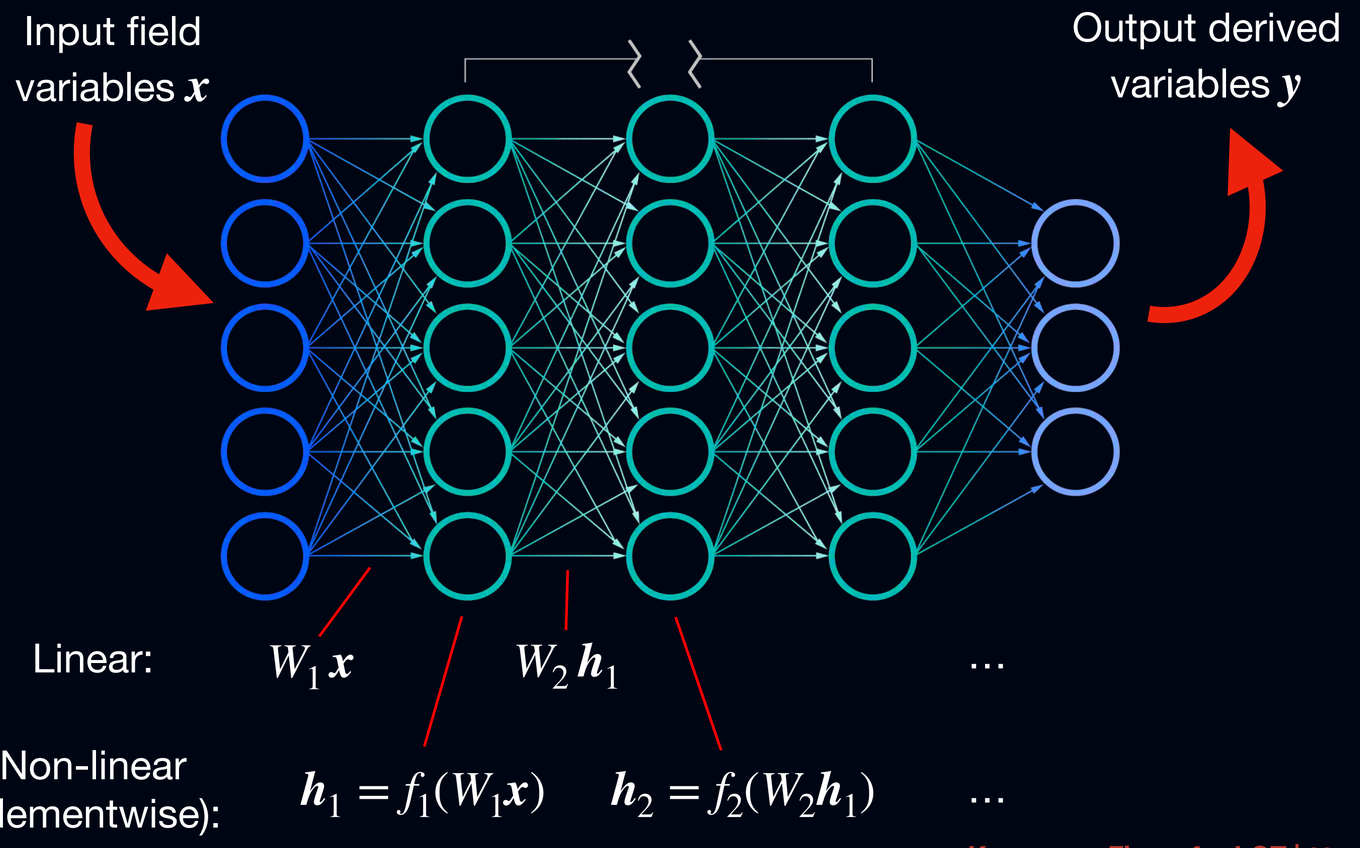
A crude picture of machine learning

Machine learning ≈ (Deep) Neural networks + Gradient-based optimization

Neural networks:
parametrized **linear transforms** +
elementwise **non-linear functions**

→ Universal function approximators

- Matrices of weights W_1, W_2 are the (optimizable) model parameters ω
- Convolutional neural networks particularly useful grid-like data



A crude picture of machine learning

Machine learning ≈ (Deep) Neural networks + Gradient-based optimization

Neural networks:
parametrized **linear transforms** +
elementwise **non-linear functions**

→ Universal function approximators

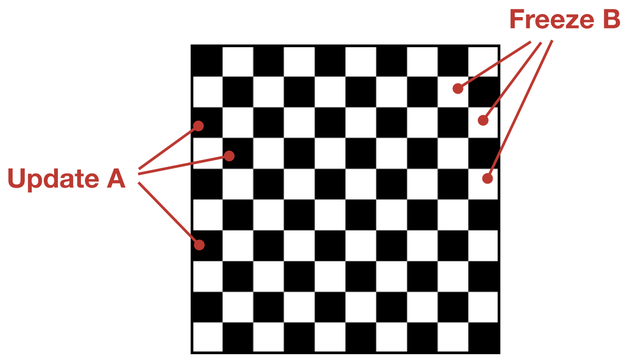
- Matrices of weights W_1, W_2 are the (optimizable) model parameters ω
- Convolutional neural networks particularly useful grid-like data

Image credit: bdtechtalks.com

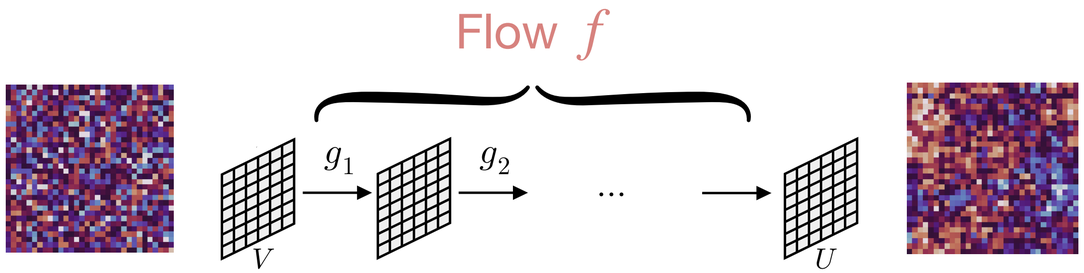
Machine learning for discrete flows

Scalar field $\phi(x) \in \mathbb{R}$, 1+1D spacetime

$$S[\phi] = \sum_x \partial_\mu \phi(x) \partial^\mu \phi(x) + \frac{M^2}{2} \phi(x)^2 + \lambda \phi(x)^4$$



Checkerboard masking pattern m

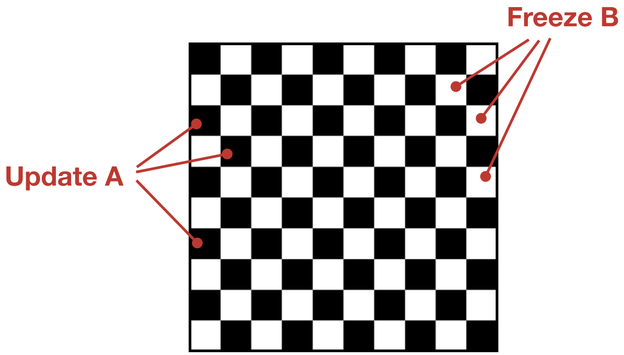


Albergo, GK, Shanahan PRD100 (2019) 034515

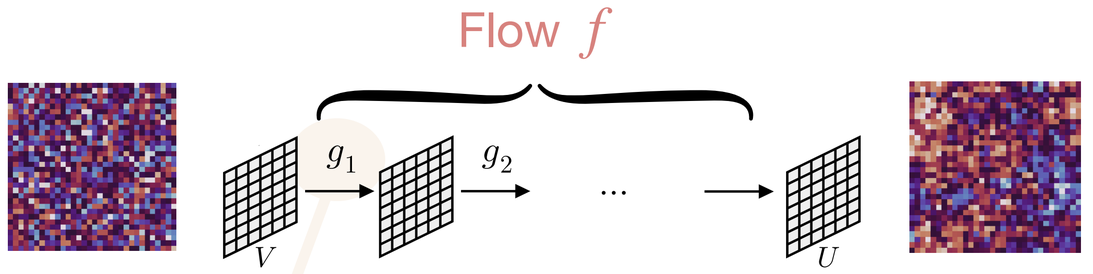
Machine learning for discrete flows

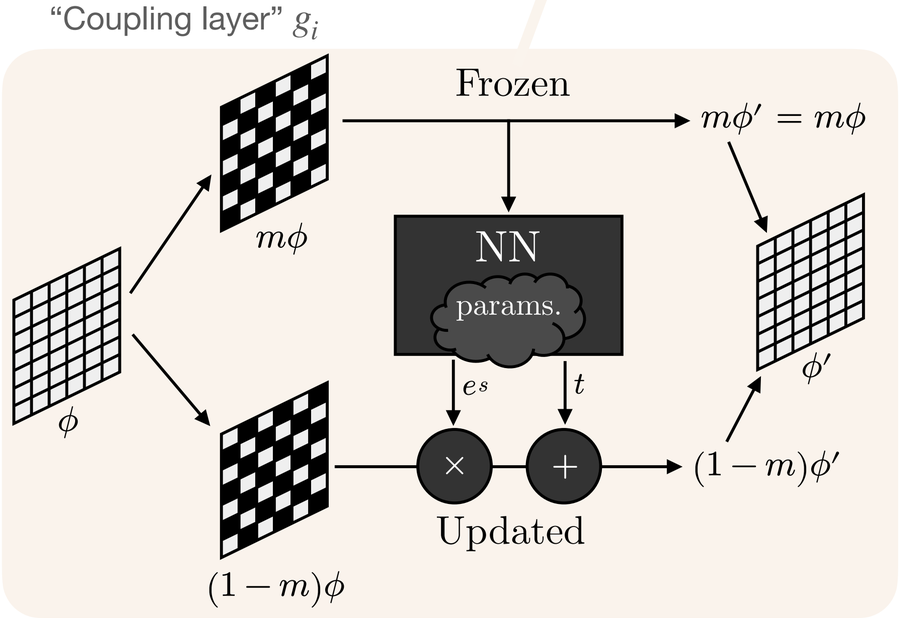
Scalar field $\phi(x) \in \mathbb{R}$, 1+1D spacetime

$$S[\phi] = \sum_x \partial_\mu \phi(x) \partial^\mu \phi(x) + \frac{M^2}{2} \phi(x)^2 + \lambda \phi(x)^4$$



Checkerboard masking pattern m





Tractable Jacobian

$$J_{ij} \equiv \partial \phi_i' / \partial \phi_j = \begin{bmatrix} I & \\ \blacksquare & \delta_{ij} e^{s_i} \end{bmatrix}$$

$$\implies \ln \det J = \sum_i s_i$$

Albergo, GK, Shanahan PRD100 (2019) 034515

Machine learning for continuous flows

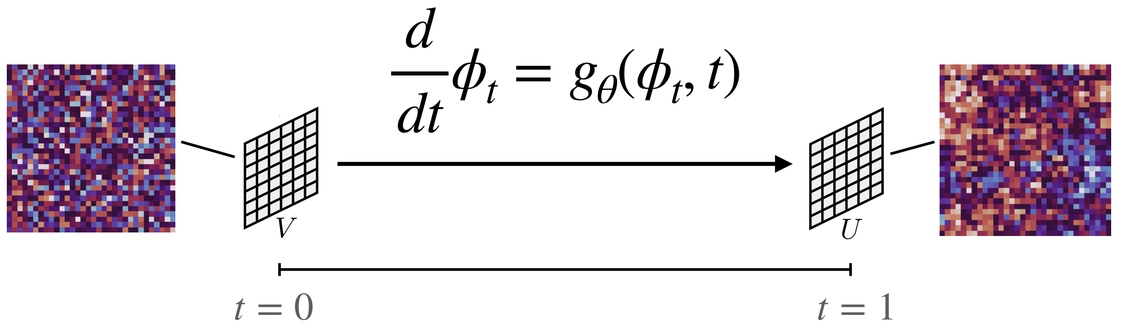
Scalar field $\phi(x) \in \mathbb{R}$, 1+1D spacetime

$$S[\phi] = \sum_x \partial_\mu \phi(x) \partial^\mu \phi(x) + \frac{M^2}{2} \phi(x)^2 + \lambda \phi(x)^4$$

- Neural net defines the velocity
- Usual methods to integrate ODE (e.g. Euler)

⚠️ Back-propagation through ODE?

✅ Adjoint-state method

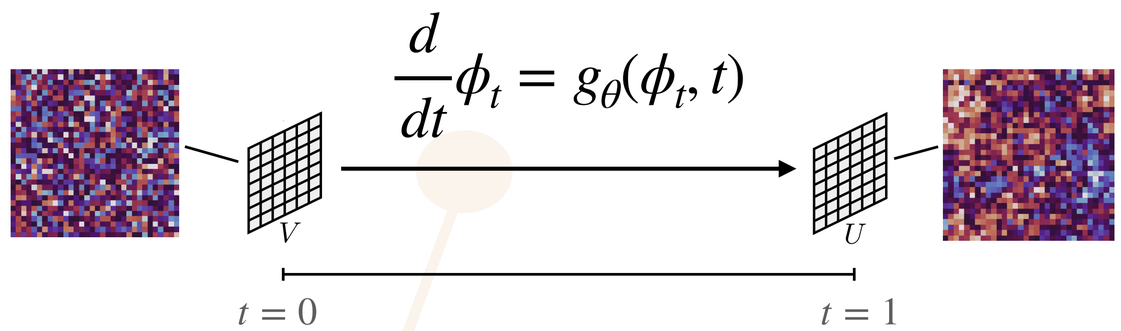


Gerdes, et al. SciPostPhys 15 (2023) 238

Machine learning for continuous flows

Scalar field $\phi(x) \in \mathbb{R}$, 1+1D spacetime

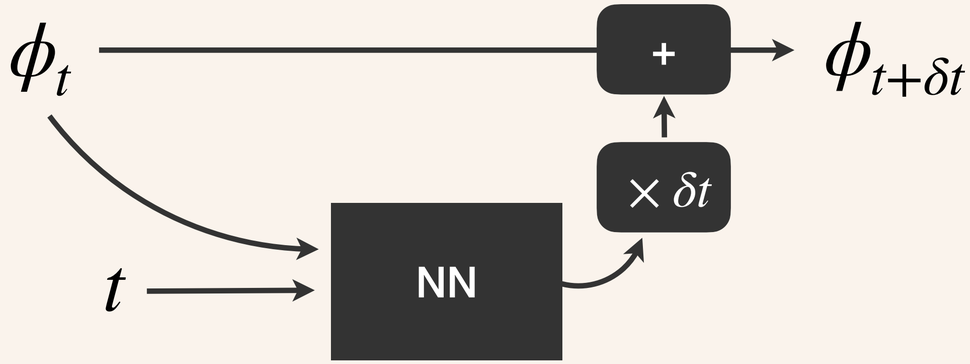
$$S[\phi] = \sum_x \partial_\mu \phi(x) \partial^\mu \phi(x) + \frac{M^2}{2} \phi(x)^2 + \lambda \phi(x)^4$$



- Neural net defines the velocity
- Usual methods to integrate ODE (e.g. Euler)

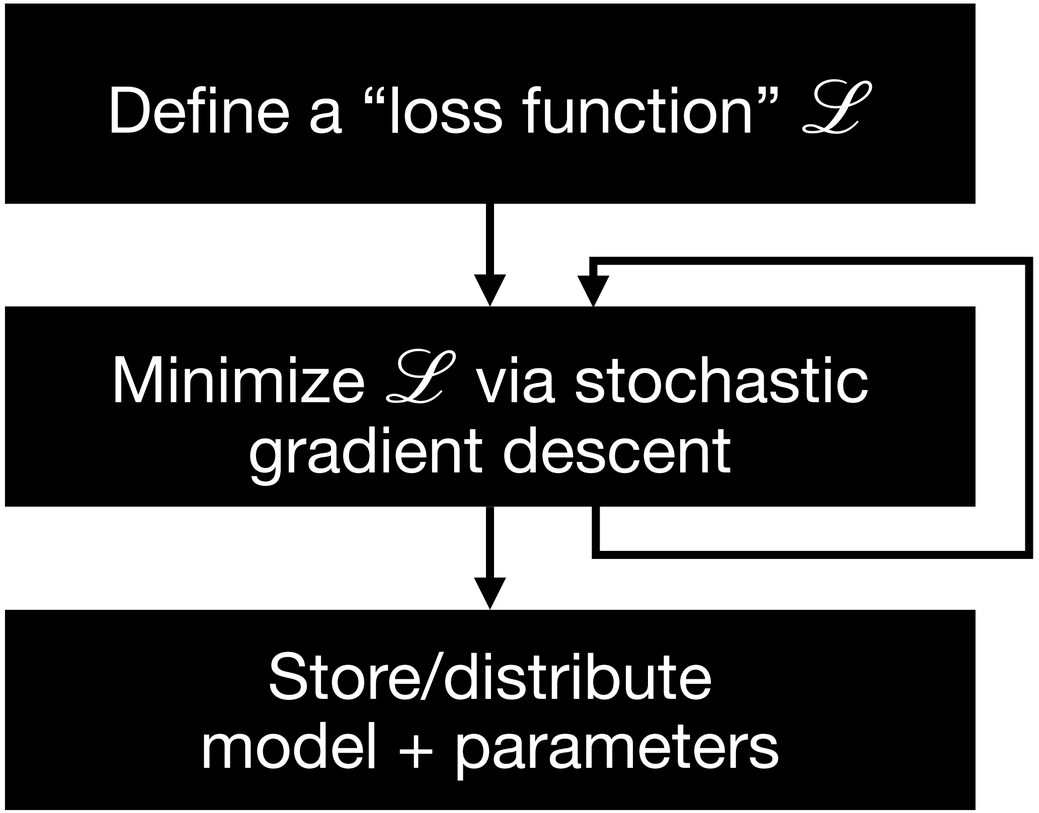
⚠️ Back-propagation through ODE?

✅ Adjoint-state method



Gerdes, et al. SciPostPhys 15 (2023) 238

Optimizing the models



Loss functions

A measure $\mathscr{L}(\theta)$ of how **badly** the network is performing, as a function of model parameters θ.

- Aim to **minimize**, giving $\mathrm{argmin}_\theta \mathscr{L}(\theta)$
- Choice of loss function depends on your objective!

Loss functions

A measure $\mathscr{L}(\theta)$ of how **badly** the network is performing, as a function of model parameters θ.

- Aim to **minimize**, giving $\text{argmin}_\theta \mathscr{L}(\theta)$
- Choice of loss function depends on your objective!

REGRESSION

$$\mathscr{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\mathscr{L}_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

...

$y_i, \hat{y}_i$ are the true, model evaluations on ith training input

Generative case: we may learn a distribution defined either empirically OR analytically

Loss functions

A measure $\mathscr{L}(\theta)$ of how **badly** the network is performing, as a function of model parameters θ.

- Aim to **minimize**, giving $\text{argmin}_\theta \mathscr{L}(\theta)$
- Choice of loss function depends on your objective!

REGRESSION

$$\mathscr{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\mathscr{L}_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

...

CLASSIFICATION

$$\mathscr{L}_{\text{Cross-entropy}} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

...

$y_i, \hat{y}_i$ are the true, model evaluations on ith training input

Generative case: we may learn a distribution defined either empirically OR analytically

Loss functions

A measure $\mathscr{L}(\theta)$ of how **badly** the network is performing, as a function of model parameters θ.

- Aim to **minimize**, giving $\text{argmin}_\theta \mathscr{L}(\theta)$
- Choice of loss function depends on your objective!

REGRESSION

$$\mathscr{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\mathscr{L}_{\text{MAE}} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

...

GENERATIVE

$$\mathscr{L}_{KL\text{fwd}} = \frac{1}{n}\sum_{i=1}^{n}\log p(x_i) - \log q(x_i), \text{ where } x_i \sim p$$

$$\mathscr{L}_{KL\text{bwd}} = \frac{1}{n}\sum_{i=1}^{n}\log q(x_i) - \log p(x_i), \text{ where } x_i \sim q$$

CLASSIFICATION

$$\mathscr{L}_{\text{Cross-entropy}} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

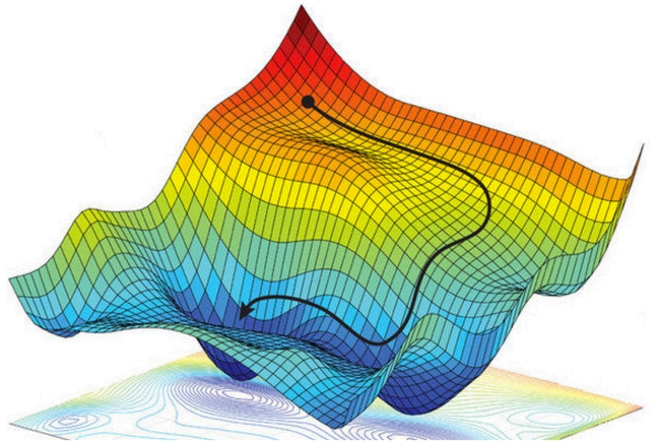
...

$y_i, \hat{y}_i$ are the true, model evaluations on ith training input

Generative case: we may learn a distribution defined either empirically OR analytically

Stochastic gradient descent

Gradient descent using **stochastic** gradient evaluations.



- Estimate true loss function by sampling “mini-batches”
- Aim to capture distribution properties, rather than population properties
- **Analytic gradient** from chain rule (“back-propagation”)

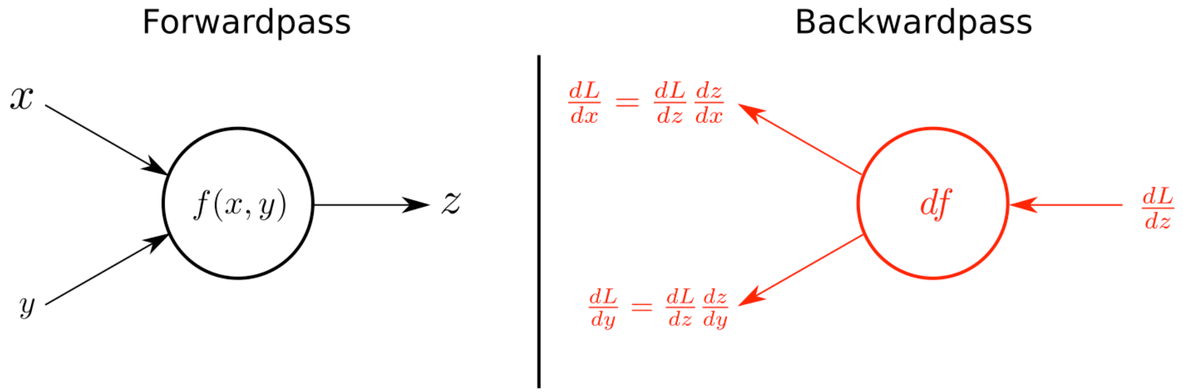


Image credits: 1805.04829, polakowo.io/datadocs

Reverse KL (self-training)

Inspired by:
- Self-Learning Monte Carlo (SLMC)
[Huang, Wang PRB95 (2017) 035105;
Liu, et al. PRB95 (2017) 041101; …]

- Self-play reinforcement learning
[Silver, et al. Science 362 (2018), 1140]

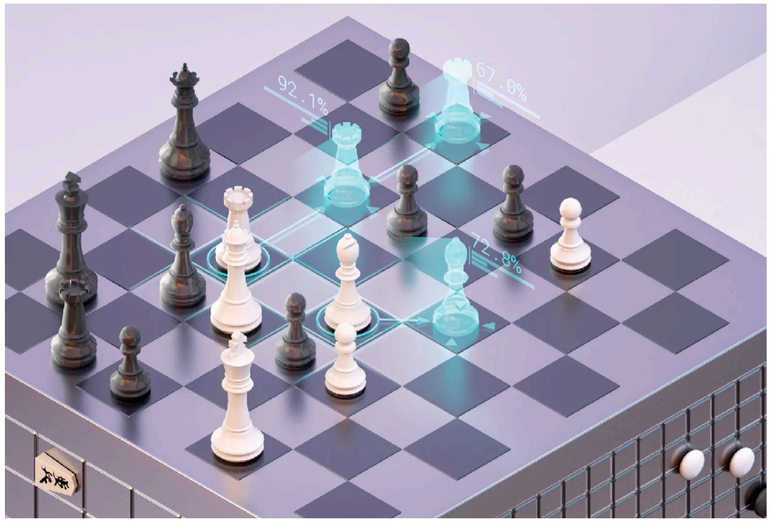
1. Define **Reverse Kullback-Leibler (KL)** divergence between $q(U)$ and $p(U) = e^{-S(U)}/Z$

$$D_{\mathrm{KL}}(q\,||\,p) := \int \mathscr{D}U\, q(U)\big[\log q(U) - \log p(U)\big] \geq 0$$

2. Measure using samples U_i **from the model**

$$\hat{D}_{\mathrm{KL}}(q\,||\,p) = \frac{1}{M}\sum_{i=1}^{M}\big[\log q(U_i) + S(U_i)\big]$$

3. Minimize by stochastic gradient descent



Minimizes average action over model samples, maximizes Shannon entropy of model q

Image credit: DeepMind

Forward KL (train from data)

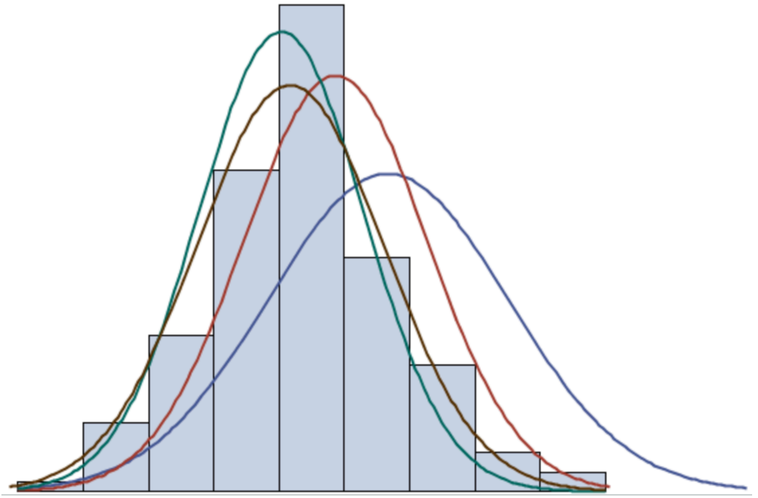
1. Define **Forward Kullback-Leibler (KL)** divergence between $q(U)$ and $p(U) = e^{-S(U)}/Z$

$$D_{\mathrm{KL}}(p\,||\,q) := \int \mathscr{D}U\, p(U)\left[\log p(U) - \log q(U)\right] \geq 0$$

INDEPENDENT OF MODEL

2. Measure using samples U_i **from training data**

$$\hat{D}_{\mathrm{KL}}(q\,||\,p) = \frac{1}{M}\sum_{i=1}^{M}\left[-\log q(U_i)\right]$$



Maximizes average model $\log q$ across samples!

3. Minimize by stochastic gradient descent

Reverse vs Forward KL

Reverse KL:

- No external data needed (“self training”)
- Mode-seeking behavior

Forward KL:

- External data needed
- Mode-covering behavior

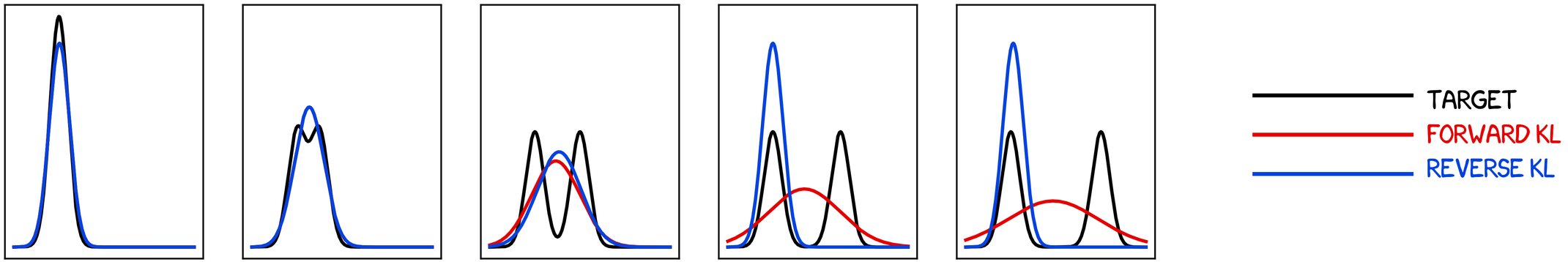


Image credit: tuananhle.co.uk

Path gradients

In either case, we only care about the **gradient** of D_{KL} with respect to parameters θ defining the flow transformation.

- Can add terms with expectation value 0, to improve variance
- Path grads is one such approach:

Gradients should stay on path: better estimators of the reverse- and forward KL divergence for normalizing flows

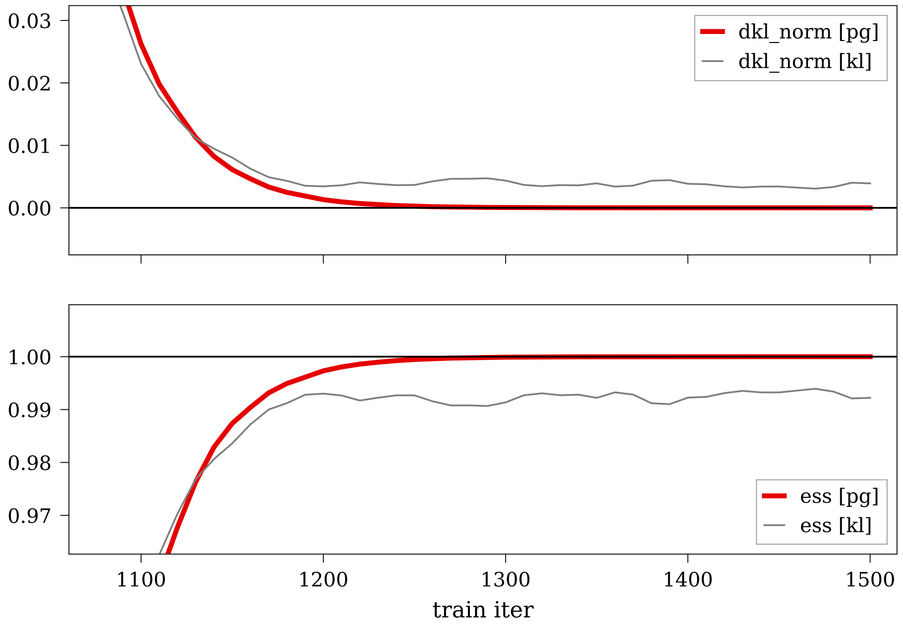
Lorenz Vaitl, Kim A Nicoli, Shinichi Nakajima and Pan Kessel

Published 19 October 2022 • © 2022 The Author(s). Published by IOP Publishing Ltd

Machine Learning: Science and Technology, Volume 3, Number 4

Citation Lorenz Vaitl *et al* 2022 *Mach. Learn.: Sci. Technol.* **3** 045006

DOI 10.1088/2632-2153/ac9455



Hands-on coding

github.com/gkanwar/flow-lectures

- Set up your env and follow along (if you want!)
- Implement basic training loop using Pytorch
- Implement and train a flow with a neural network
- Explore this code further in the exercises