University of Amsterdam

w.i.p. with Pim de Haan,
Roberto Bondesan &
Miranda Cheng

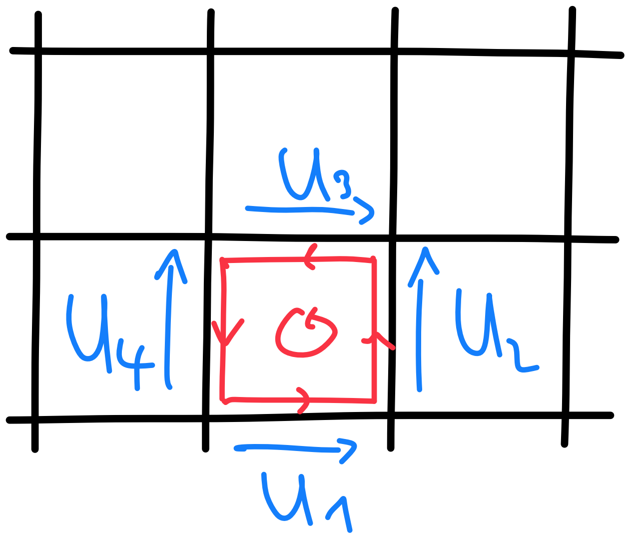
Continuous flows for SU(2)

Exploring general flow architectures for pure Yang-Mills

Mathis Gerdes — m.gerdes@uva.nl | Swansea ML meets LFT 2024

Lattice gauge theory

Wilson action



Wilson action $S = -\frac{\beta}{N} \sum_x \mathrm{Re}\left[W(x)\right]$

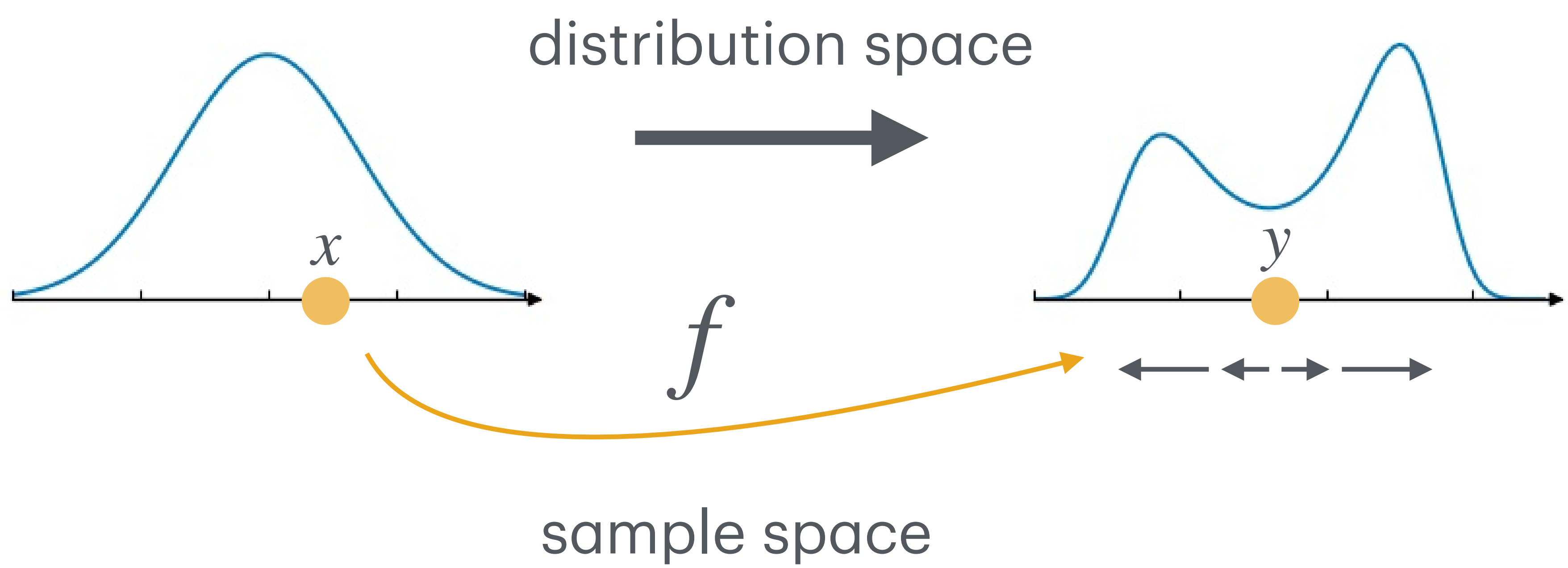
Wilson loop trace

$$W = \mathrm{tr}(U_1 U_2 U_3^\dagger U_4^\dagger)$$

→ Want to sample U-configurations $\sim e^{-S[U]}$

Change of variables

Transforming probability densities



Change of density

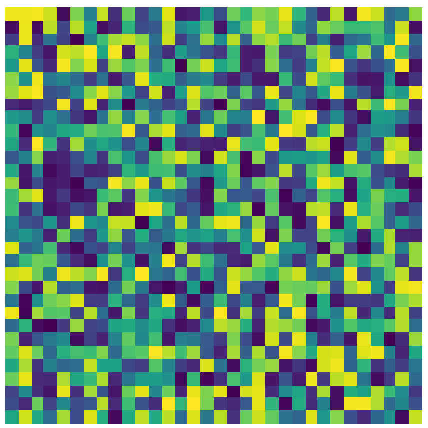
$$p(y) = p\left(f^{-1}(y)\right) \cdot \left|\det \frac{\partial f}{\partial x}\right|^{-1}$$

Source point

Normalizing flows

Learning f

trivial theory



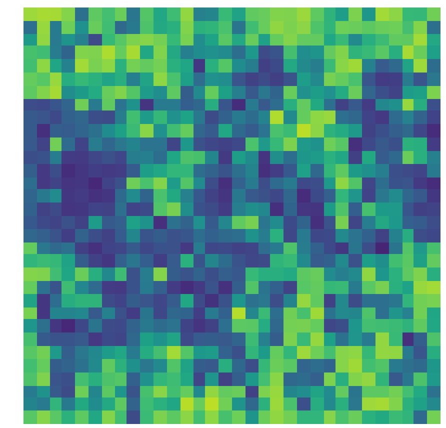
$\mathcal{N}$

bijection f

↔

"Normalizing flow"

interacting theory



$e^{-S[\phi]}$

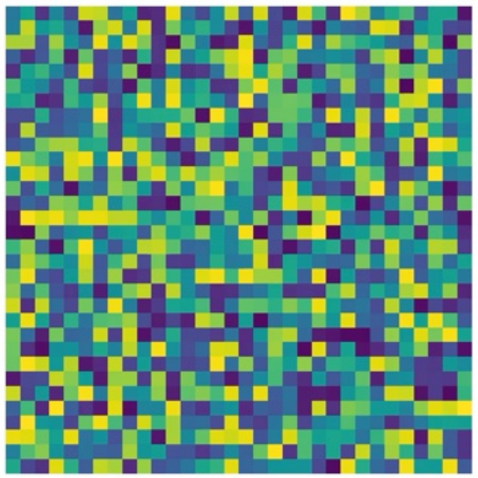
<u>We want to ***learn*** a *trivializing map* f .</u>

To compute model probability:

$$p(y) = p\left(f^{-1}(y)\right) \cdot \left|\det \frac{\partial f}{\partial x}\right|^{-1}$$

- f must be bijective.
- Computing the det-Jacobian must be tractable.

Continuous normalizing flows



Sample $\phi^0 \sim \mathcal{N}$ Final proposal $\phi^{t=1}$

Solve $\dfrac{d}{dt}\phi = g_\theta(\phi, t)$

- ODE always invertible, architecture of g_θ unconstrained!
- ODE for $p(\phi^t)$ given by divergence:

$$\frac{d}{dt}\log p(\phi) = -\nabla \cdot \dot{\phi}$$

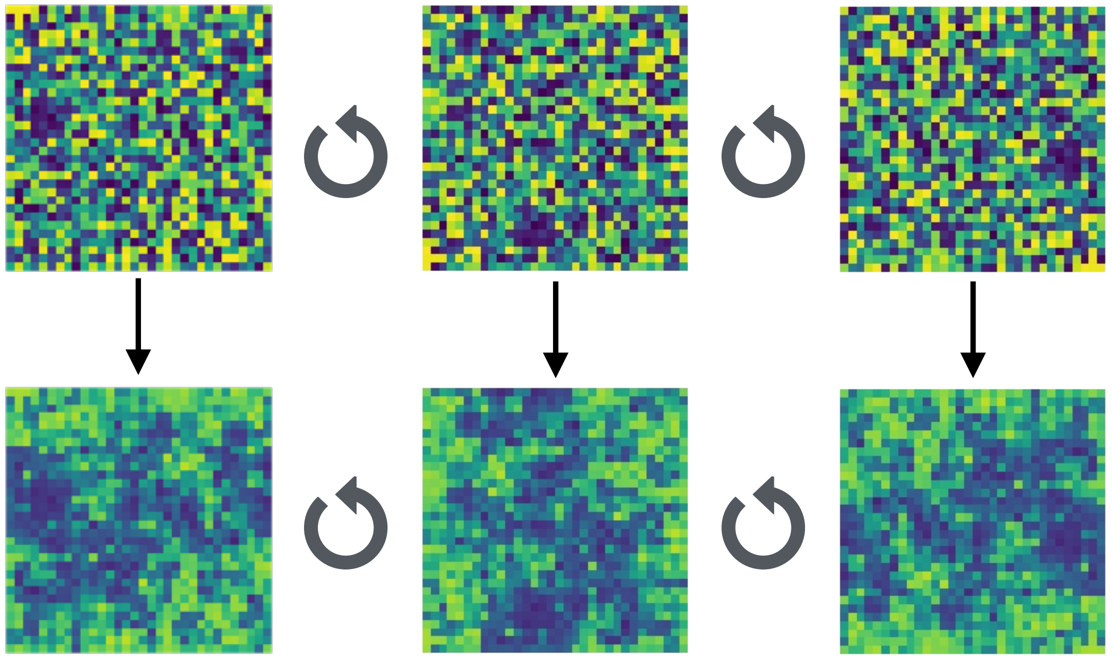
Symmetries

And the need for equivariant flows

If action is invariant under transformation $S(\phi) = S(g \cdot \phi)$

then $p(\phi) = p(g \cdot \phi)$, should be proposed equally likely.

$$f_\theta(g \cdot \phi) = g \cdot f_\theta(\phi)$$

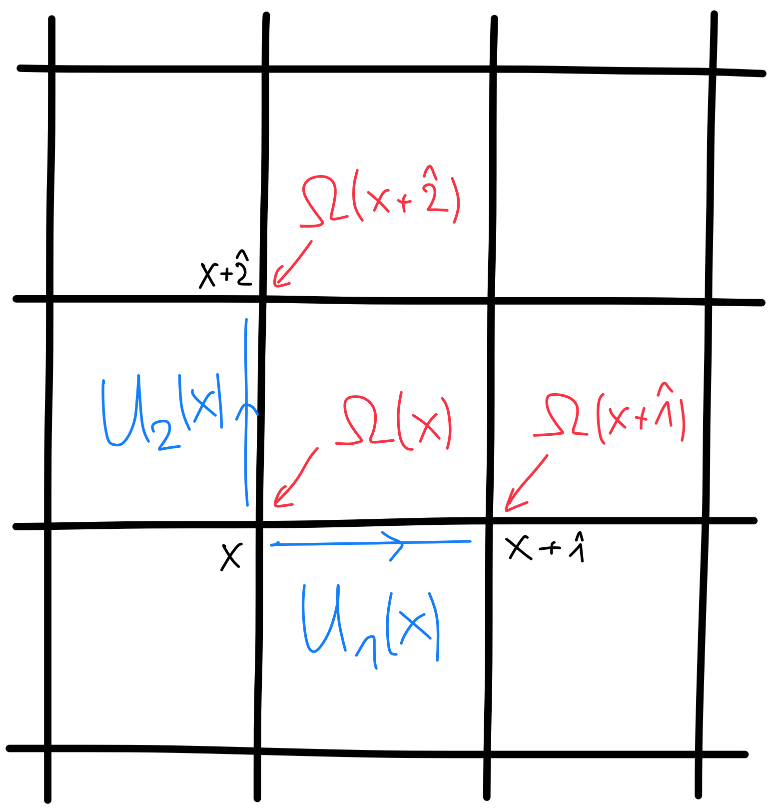


etc.

Gauge symmetry

How objects transform

$$U_\mu(x) \mapsto \Omega(x)\, U_\mu(x)\, \Omega(x+\hat{\mu})^\dagger$$



Wilson loop

$$P_{12} = U_1(x)U_2(x+\hat{1})U_1(x+\hat{2})^\dagger U_2(x)^\dagger$$

are **equivariant** $P_{12} \mapsto \Omega(x)P_{12}\Omega(x)^\dagger$.

Trace of Wilson loops

$W = \operatorname{tr} P_{12}$ are **invariant**.

Gradients of invariants

e.g. $V = \nabla_U W$ are **equivariant**

$$V \mapsto \Omega(x)V\Omega(x)^\dagger$$

Discrete normalizing flows

How to define gauge equivariant flows

Map $P_{\mu\nu} \mapsto P'_{\mu\nu} = f(P_{\mu\nu})$ to update edge in $P_{\mu\nu}$ conditioned on unmodified invariant quantities.

Get an equivariant flow, if map transform under conjugation:
$f(\Omega P \Omega^\dagger) = \Omega f(P) \Omega^\dagger$

arxiv:2008.05456

Sampling using SU(N) gauge equivariant flows

Denis Boyda,[1,*] Gurtej Kanwar,[1,†] Sébastien Racanière,[2,‡] Danilo Jimenez Rezende,[2,§]
Michael S. Albergo,[3] Kyle Cranmer,[3] Daniel C. Hackett,[1] and Phiala E. Shanahan[1]

arxiv:2305.02402

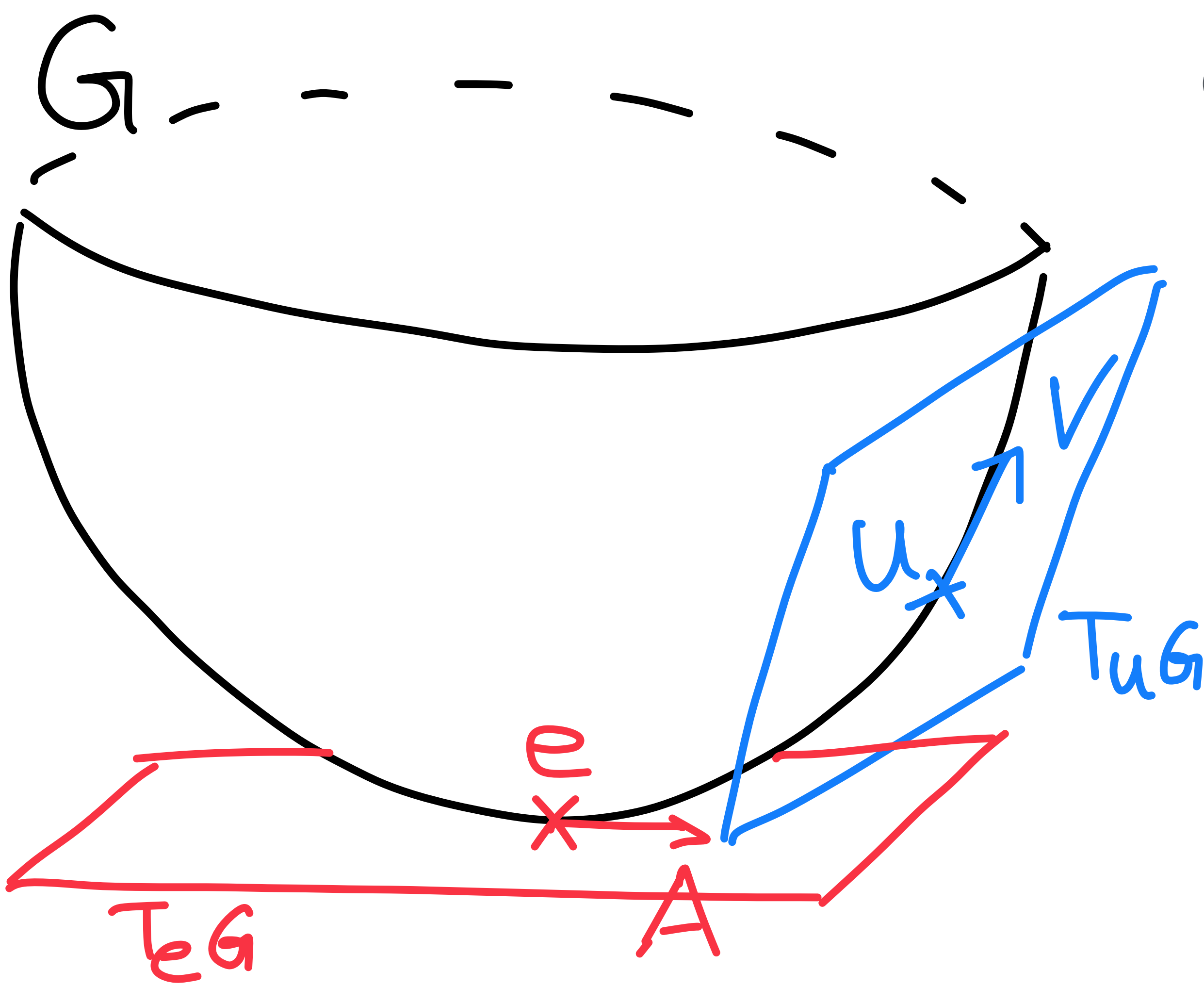
Normalizing flows for lattice gauge theory in arbitrary space-time dimension

Ryan Abbott,[1,2] Michael S. Albergo,[3] Aleksandar Botev,[4] Denis Boyda,[1,2] Kyle Cranmer,[5]
Daniel C. Hackett,[1,2] Gurtej Kanwar,[6,1,2] Alexander G.D.G. Matthews,[4] Sébastien Racanière,[4] Ali
Razavi,[4] Danilo J. Rezende,[4] Fernando Romero-López,[1,2] Phiala E. Shanahan,[1,2] and Julian M. Urban[1,

Continuous flows for gauge theories

Lie groups

A brief reminder



We can parametrize the vector space at U via the Lie algebra:

$$V \in T_U G \qquad A := VU^\dagger \in \mathfrak{g} = T_e G$$

$$V = AU$$

Transporting A to vector space at U

Lie algebra is spanned by generators T^a

In components, $V = A^a T^a U$

Continuous flows for SU(N)

Defining an ODE

In coordinates A^a, general vector at U is: $V = (T^a A^a)U$.

Path derivative $\partial^a f(U) = \frac{d}{ds}\Big|_{s=0} f(e^{sT^a}U) = Df(T^a U)$.

Then, the gradient is $\nabla f(U) = \partial^a f(U)\, T^a U$.

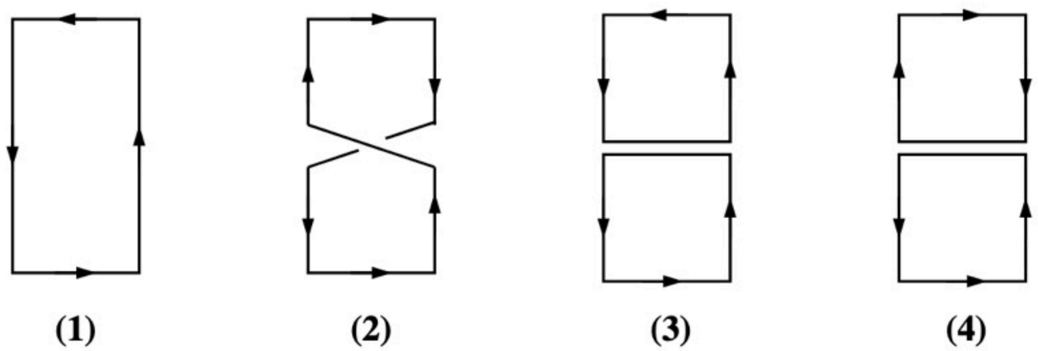
To define our flow, the network should output an algebra element:

$$\frac{d}{dt}U = A^a(U)\, T^a U$$

Continuous flows for SU(N)

Gradient flows

Define $A^a = \partial^a S$ as the gradient of some potential, given as sums and products of Wilson loops.



Can extend/do better by learning coefficients by gradient descent

Trivializing maps, the Wilson flow and the HMC algorithm

Martin Lüscher

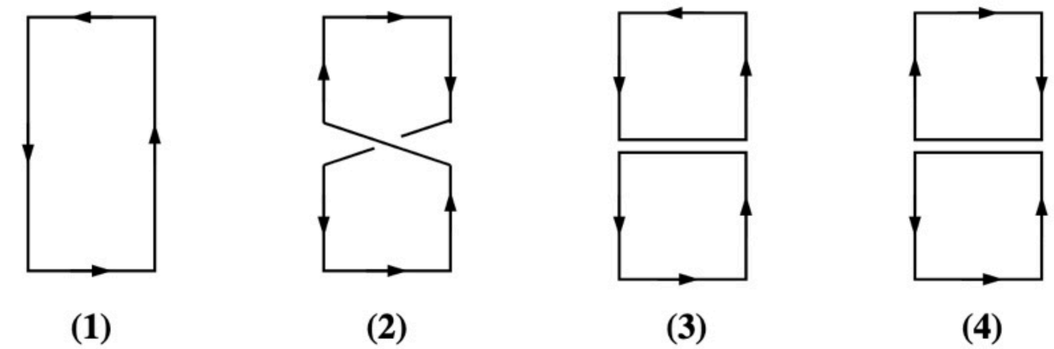
Learning Trivializing Gradient Flows for Lattice Gauge Theories

Simone Bacchio,[1] Pan Kessel,[2,3] Stefan Schaefer,[4] and Lorenz Vaitl[2]

Can we define a more general ML architecture?

Network

Idea for construction



Equivariant vector field

"Basis" vectors: Built to be gauge equivariant

Superposition function: Built out of invariant quantities

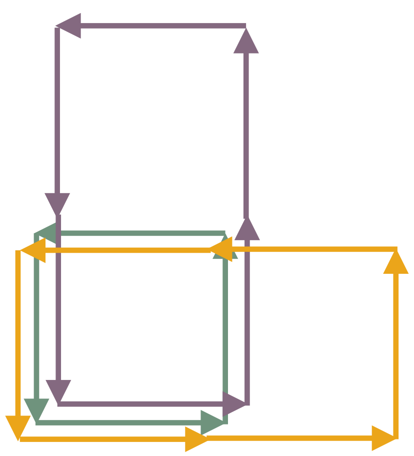
$$A_e^a(U) = \sum_k \partial^a_{U_e} W^{(k)} \cdot S_e^k(W^{(1)}, W^{(2)}, \ldots)$$

- Spatial symmetries, need to be built into S.
- Divergence must not be too expensive.

Network

Idea for construction

$$A_e^a(U) = \sum_k \partial^a_{U_e} W^{(k)} \cdot S_e^k(W^{(1)}, W^{(2)}, \ldots)$$



$$S_e^k = C_{e,x}^{k,l}\, \mathrm{NN}_x^l(\{W_x^{(m)}\})$$

$W_x^{(k)}$

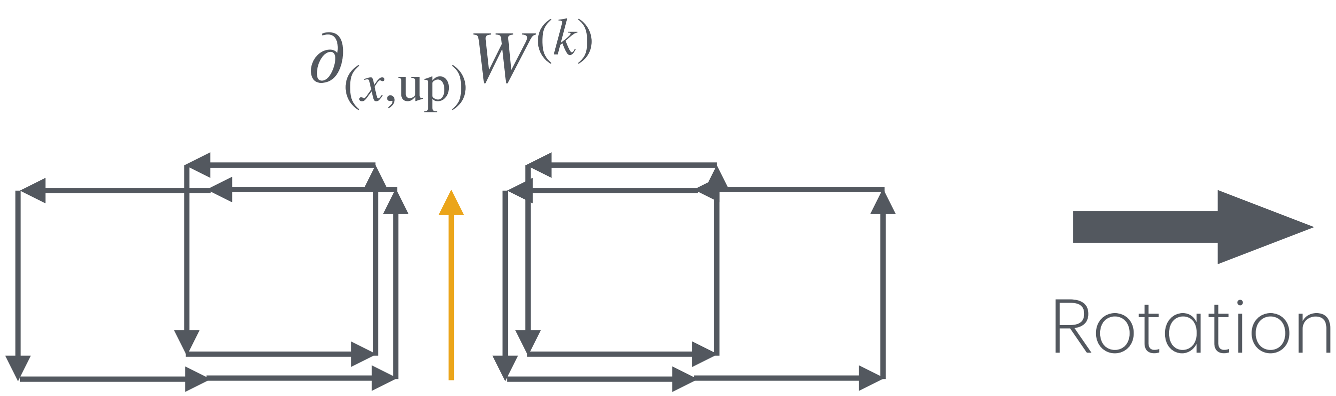
Local "stack" of Wilson loops

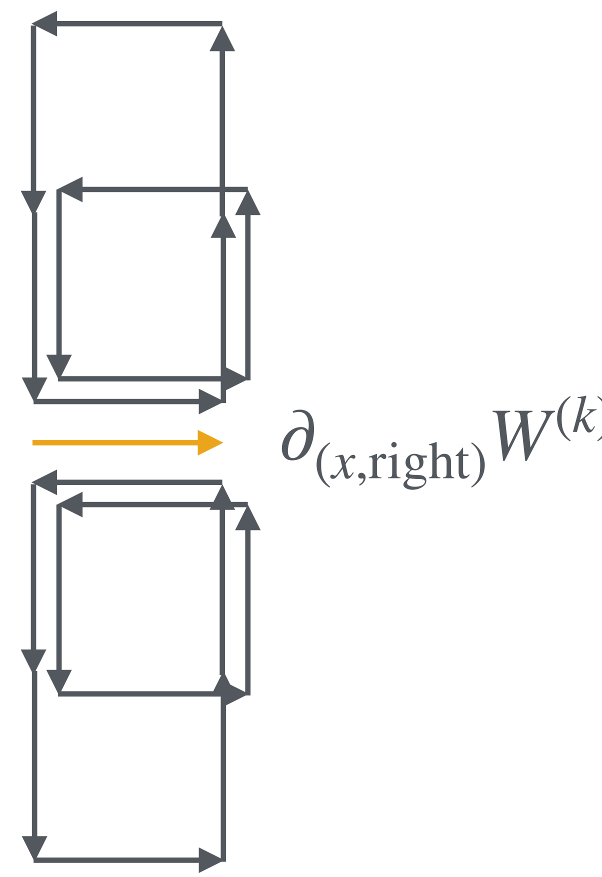
→ Arbitrary (non-linear) "local" neural network → (Equivariant) Convolution

Spatial symmetries

Work in progress

$$A_e^a(U) = \sum_k \partial^a_{U_e} W^{(k)} \cdot S_e^k(W^{(1)}, W(2), \ldots)$$





Rotation/flip symmetries impose restriction on convolutional kernel, non-linear network, and Wilson loop inputs.

Divergence computation

To track density change

$$A_e^a = \sum_k \partial_e^a W^{(k)} \cdot S_e^k(\{W\})$$

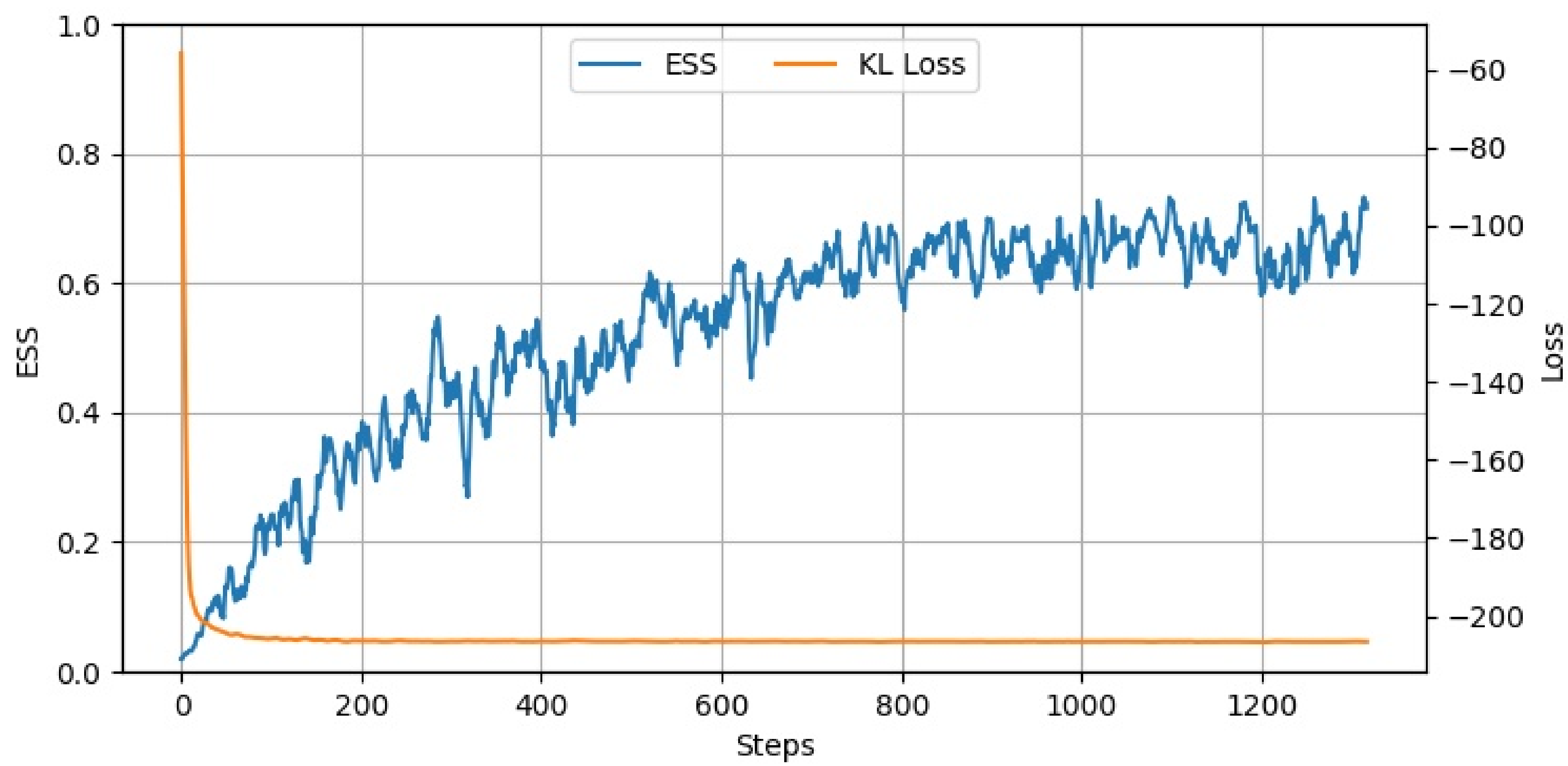
$$\partial_e^a A_e^a = \sum_k \partial_e^a \partial_e^a W^{(k)} \cdot S_e^k(\{W\}) + \partial_e^a W^{(k)} \cdot \partial_e^a S_e^k(\{W\})$$

$$\partial_e^a S_e^k = \sum_{l,x} C_{e,x}^{k,l} \, D(\mathrm{NN}_x^l)(\{\partial_e^a W_x^{(m)}\})$$

- Computational cost scales in how “local” stack is (not lattice size).
- Can be computed efficiently via JAX’s forward differentiation.

Success for SU(2)

So far, so good...



Reference results
From [2008.05456, Boyda et al]

	SU(2)		
β	1.8	2.2	2.7
ESS(%)	91	80	56

> 10% point improvement

Promising results after ~O(1h) training on a single GPU.

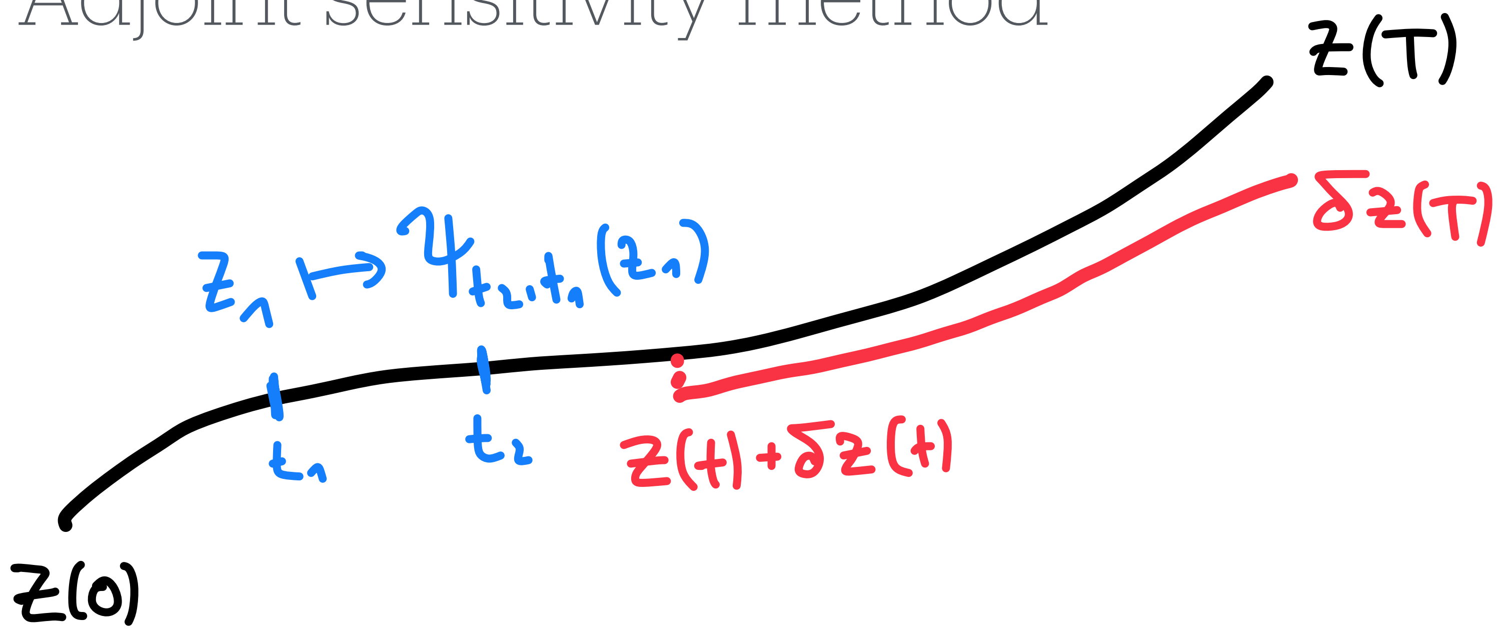
Technical challenges

- Implement integration schemes for SU(N) & real d.o.f. simultaneously.
- Computation of the laplacian (JAX's JVP/VJP magic helps).
- Book-keeping of loops and gradients.
- Implemented general adjoint sensitivity method.

For everything!

Technical challenges

Adjoint sensitivity method



Continuous flow

ODE $\dot{z} = f_\theta(z,t)$

We have a loss function $L : M \to \mathbb{R}$, so $dL_z \in T_z^*M$

Adjoint state: $a(t) = \psi^*_{T,t}\, dL_{z(T)}$.

In words: maps $\delta z(t)$ to δL.

"Compute gradients by back-integrating"

$$\frac{da(t)}{dt} = -\,a(t)\frac{\partial f_\theta(z,t)}{\partial z} \qquad \frac{dL}{d\theta} = -\int_T^0 a(t)\frac{\partial f(z,t)}{\partial \theta}\,dt$$

Takeaways

And open questions

- Success for SU(2). Still working on architecture (and optimization).
- Overcame technical hurdles, implementing everything in JAX.
    - Network can be modified without manual intervention.
    - How does it compare to dedicated libraries?
    - What are the training times of other methods?
- Many things to explore for network architectures.
  (e.g.: should be a strict superset of Lüscher's flow, could init there)