

Symmetry-preserving neural networks on the lattice

Matteo Favoni

Swansea University

18 Mar 2024

UKLFT meeting 2024

Based on:

S. Bulusu, M. F., A. Ipp, D. I. Müller, D. Schuh, Phys. Rev. D 104, **104**, 074504

M. F., A. Ipp, D. I. Müller, D. Schuh, Phys.Rev.Lett. **128**, 032003

M. F., A. Ipp, D. I. Müller, EPJ Web of Conferences **274**, 09001



FWF



- Neural networks (NNs) are a machine learning tool, capable of learning from experience
- Supervised learning: dataset $\{x^i, f(x^i)\}$; we want to find $g(x^i, \theta) \approx f(x^i)$
- NNs are universal approximators for non-linear functions
- NNs do not know a priori about symmetry but they can learn it approximately
- Strategy here: restrict parameters to enforce symmetry
- Convolutional neural networks (CNNs) incorporate translational symmetry under certain circumstances
- Lattice gauge equivariant convolutional neural networks (L-CNNs) incorporate gauge symmetry

Translational symmetry and CNNs

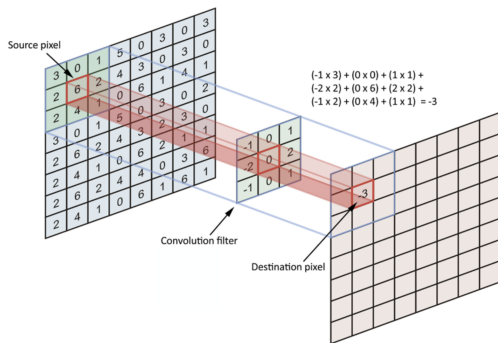


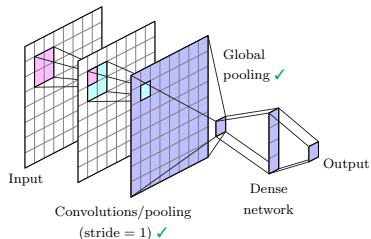
Image from [here](#)

- Locality and weight sharing
- Invariance vs equivariance

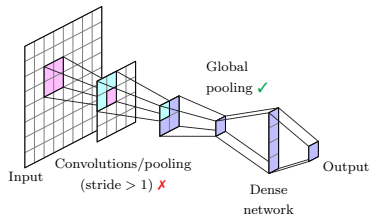
$$\Phi(L_g x) = \Phi(x) \quad \text{vs} \quad \Phi(L_g x) = L_g \Phi(x)$$

- Periodic boundary conditions through circular padding

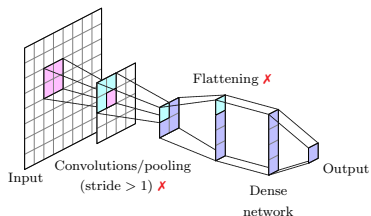
Architecture types



Equivariant architecture (EQ)



Strided architecture (ST)



Flattening architecture (FL)

- Complex scalar field in 1+1D with nonzero chemical potential

$$S = \int dx_0 dx_1 (|D_0 \phi|^2 - |\partial_1 \phi|^2 - m^2 |\phi|^2 - \lambda |\phi|^4), \quad D_0 = \partial_0 - i\mu \quad (1)$$

- Discretized action

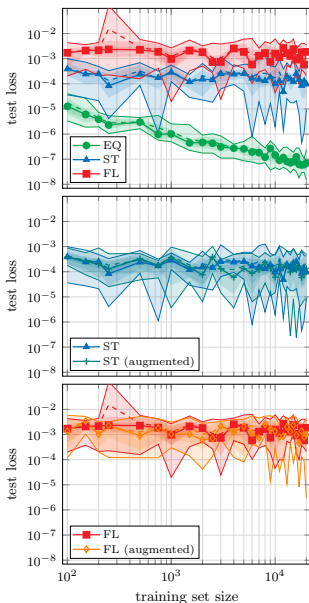
$$S_{lat} = \sum_x \left(\eta |\phi_x|^2 + \lambda |\phi_x|^4 - \sum_{\nu=1}^2 \left(e^{\mu \delta_{\nu,2}} \phi_x^* \phi_{x+\hat{\nu}} + e^{-\mu \delta_{\nu,2}} \phi_x^* \phi_{x-\hat{\nu}} \right) \right), \quad \eta = 2D + m^2 \quad (2)$$

- Sign problem solved by a dual formulation: $\phi_x \rightarrow \{k_{x,\nu}, l_{x,\nu}\}$ integer fields, [Gattringer, Kloiber, arxiv:1206.2954](#)
- Regression task: predicting observables

$$n = \frac{1}{N} \sum_x k_{x,2}, \quad |\phi|^2 = \frac{1}{N} \sum_x \frac{W(f_x + 2)}{W(f_x)} \quad (3)$$

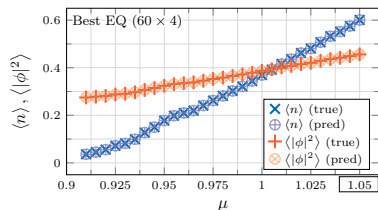
$$f_x = \sum_{\nu} [|k_{x,\nu}| + |k_{x-\hat{\nu},\nu}| + 2(l_{x,\nu} + l_{x-\hat{\nu},\nu})], \quad W(f_x) = \int_0^{\infty} dx x^{f_x+1} e^{-\eta x^2 - \lambda x^4} \quad (4)$$

Architecture comparison

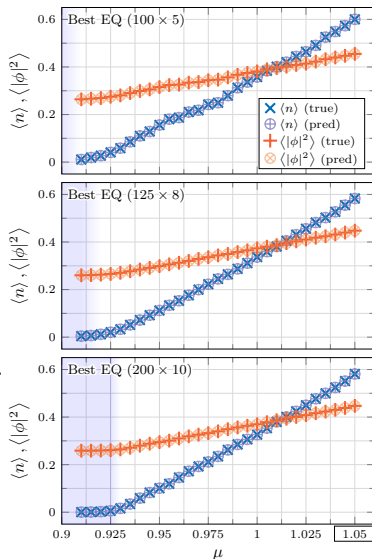


- Systematic architecture search with optuna, [Akiba et al., arxiv:1907.10902](#)
- 10 instances of the winning architectures are retrained from scratch for various training set size
- EQ beats ST and FL for any number of training samples
- EQ improves with more samples, while the other two do not
- Data augmentation does not help the two non-equivariant architectures

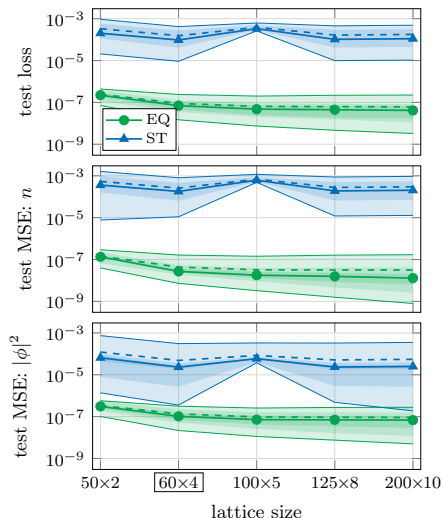
Silver blaze phase transition



Training on both phases is not necessary as long as the expression of the observables is independent of the transition



Generalization to other lattice sizes and physical parameters



- FL cannot be tested on different lattice sizes
- Training only on 60×4
- Kink in ST at 100×5 due to $s = 2$ in spatial pooling layer
- EQ clearly outperforms ST
- Problem already tackled in literature with an FL trained on both phases on 200×10 reaching test loss of 10^{-6}

Gauge symmetry and L-CNNs

- Lattice gauge transformations for \mathcal{U} and \mathcal{W}

$$T_{\Omega} U_{\mathbf{x},\mu} = \Omega_{\mathbf{x}} U_{\mathbf{x},\mu} \Omega_{\mathbf{x}+\mu}^{\dagger}, \quad \Omega_{\mathbf{x}} \in \text{SU}(N_c)$$
$$T_{\Omega} W_{\mathbf{x},\mu\nu} = \Omega_{\mathbf{x}} W_{\mathbf{x},\mu\nu} \Omega_{\mathbf{x}}^{\dagger}$$

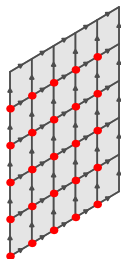
- Gauge equivariant function

$$g(T_{\Omega}\mathcal{U}, T_{\Omega}\mathcal{W}) = T_{\Omega}g(\mathcal{U}, \mathcal{W})$$

- Gauge invariant function (e.g. observables, action)

$$g(T_{\Omega}\mathcal{U}, T_{\Omega}\mathcal{W}) = g(\mathcal{U}, \mathcal{W})$$

- The individual layers of a lattice gauge equivariant convolutional neural network (L-CNN) are designed to respect gauge equivariance



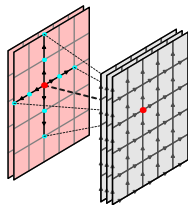
$$\mathcal{U} = \{U_{\mathbf{x},\mu}\}$$

$$\mathcal{W} = \{W_{\mathbf{x},\mu\nu}\}$$

Preprocessing layers (**Plaq**, **Poly**)

- Preprocess input \mathcal{U} to generate locally transforming objects \mathcal{W} , i.e. plaquettes and Polyakov loops $\mathcal{L}_{\mathbf{x},\mu}(\mathcal{U}) = \prod_k U_{\mathbf{x}+k\mu,\mu}$

$$\mathbf{Plaq, Poly} : \mathcal{U} \rightarrow (\mathcal{U}, \mathcal{W})$$

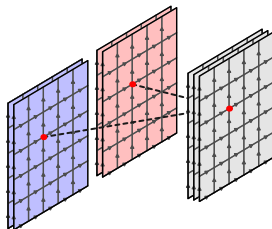


Lattice gauge equivariant convolutions (**L-Conv**)

- Properties of CNNs: **compact kernels, weight sharing**
- Parallel transport of data \mathcal{W} to common point using \mathcal{U}
- Avoid path dependence by restricting the kernel along the axis

$$\mathbf{L-Conv} : W'_{\mathbf{x},i} = \sum_{j,\mu,k} \omega_{i,j,\mu,k} U_{\mathbf{x},k\cdot\mu} W_{\mathbf{x}+k\cdot\mu,j} U_{\mathbf{x},k\cdot\mu}^\dagger$$

- Equivariant convolutions: $(\mathcal{U}, \mathcal{W}) \rightarrow (\mathcal{U}, \mathcal{W}')$

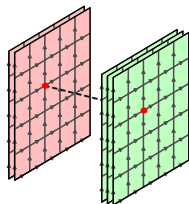


Equivariant bilinear layers (**L-BL**):

- Multiplication of \mathcal{W} 's at same lattice point is equivariant
- Allow the network to grow larger loops

$$\mathbf{L-BL} : W''_{x,i} = \sum_{j,k} \alpha_{ijk} W_{x,j} W'_{x,k}$$

- Bilinear layers: $(\mathcal{U}, \mathcal{W}) \times (\mathcal{U}, \mathcal{W}') \rightarrow (\mathcal{U}, \mathcal{W}'')$

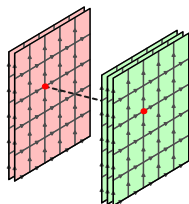


Gauge equivariant activation functions (**L-Act**):

- Multiplication of \mathcal{W} with gauge invariant scalar functions a

$$\mathbf{L-Act} : W'_x = a(\text{Tr } W_x) W_x$$

- Activation functions: $(\mathcal{U}, \mathcal{W}) \rightarrow (\mathcal{U}, \mathcal{W}')$

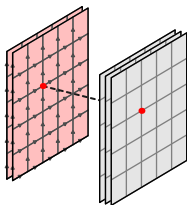


Equivariant exponential layers (**L-Exp**):

- Equivariant method to modify links $\mathcal{U} \rightarrow \mathcal{U}'$
- Multiplication of \mathcal{U} with locally transforming $SU(N_c)$

$$U'_{\mathbf{x},\mu} = \exp \left(i \sum_i \beta_{\mu,i} [W_{\mathbf{x},i}]_{\text{ah}} \right) U_{\mathbf{x},\mu}$$

- Equivariant exponential layer: $(\mathcal{U}, \mathcal{W}) \rightarrow (\mathcal{U}', \mathcal{W})$



Generate gauge invariant output (**Trace**)

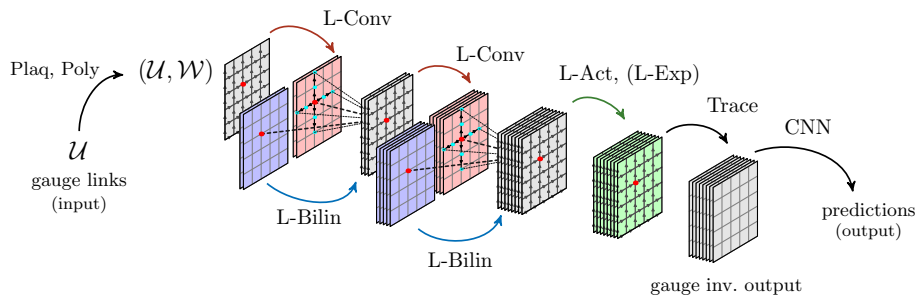
- Computes traces of \mathcal{W} 's: gauge invariant complex numbers

$$\mathbf{Trace} : w_{\mathbf{x},i} = \text{Tr } W_{\mathbf{x},i} \in \mathbb{C}$$

- No trainable parameters (“postprocessing”)
- Gauge invariant output can be passed to traditional CNN

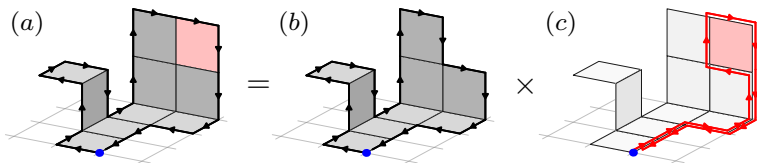
Lattice gauge equivariant convolutional neural networks

L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



Construction of arbitrary Wilson loops

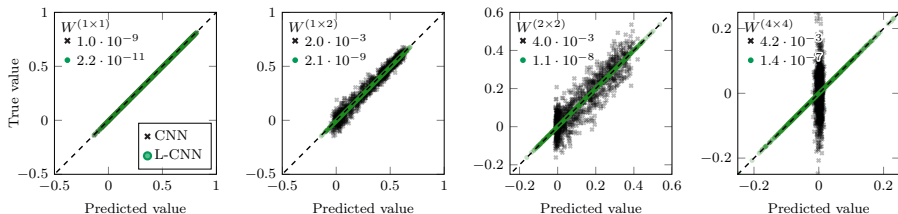
- Repeated applications of **L-Conv** and **L-Bilin** operations can be used to generate arbitrarily sized Wilson loops if input \mathcal{W} consists of plaquettes (preprocessing layer **Pla**)



- Non-contractible loops can also be generated by including Polyakov loops in the input \mathcal{W} (preprocessing layer **Poly**)
- Non-linear functions of Wilson loops are possible through **L-Act**, **Trace** and passing gauge invariant output to traditional CNNs
- L-CNNs are **universal approximators** for gauge invariant functions on the lattice

Regression on Wilson loops

Benchmark problem: regression of Wilson loops from 1×1 to 4×4 on 2D lattice



True vs. predicted values for CNNs and L-CNNs for $n \times m$ Wilson loops (best models)

- Increasing loop size from left to right \rightarrow increasingly harder problem
- Deteriorating performance of baseline CNNs with increased loop size
- Best L-CNN **always** beats best baseline CNN
- Consistent performance of L-CNNs across all loop and lattice sizes

NODE and application to gradient flow

Neural ordinary differential equations (NODEs) are ODEs parametrized by NNs (see e.g. [arXiv:1806.07366](https://arxiv.org/abs/1806.07366))

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}(t), \theta, t)$$

- $\mathbf{x}(t)$: time-dependent D-dimensional vector
- $\mathbf{f}(\mathbf{x}(t), \theta, t)$: NN parametrized by the weights θ with a D-dimensional output
- **Input**: boundary value $\mathbf{x}_0 = \mathbf{x}(t_0)$
- **Label**: desired output vector \mathbf{x}_1^*
- **Prediction**: final state vector $\mathbf{x}(t_1) = \mathbf{x}_0 + \int_{t_0}^{t_1} dt' \mathbf{f}(\mathbf{x}(t'), \theta, t')$
An ODE integrator (e.g. Euler, Runge-Kutta) is used for the state evolution
- **Training**: minimization of the loss function $\mathcal{L}(\theta) = (\mathbf{x}_1^* - \mathbf{x}(t_1))^2$

Neural gradient flow

We study the equivariant flow equation

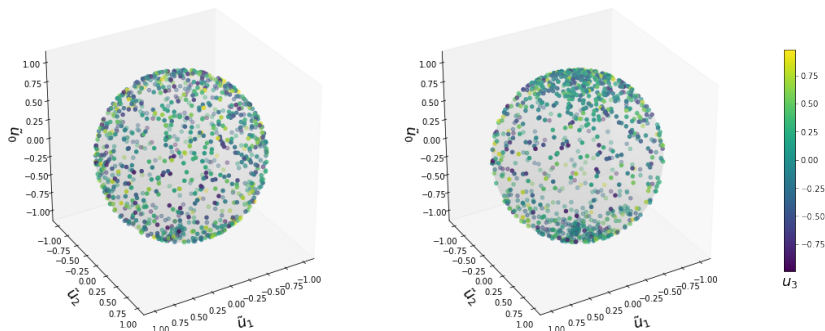
$$\frac{dU_{x,\mu}(t)}{dt} = iH_\mu[U(t), \theta, t] U_{x,\mu}(t)$$

- $U(t)$: gauge link configuration
- $H_\mu[U(t), \theta, t]$: NN parametrized by the weights θ with a traceless and hermitian output
- **Input**: boundary value $U_0 = U(t_0)$
- **Label**: desired output U_1^*
- **Prediction**: final configuration $U(t_1)$ found by the iterative application of the exponential map $U_{x,\mu}^{(i+1)} = \exp(iH_\mu[U^{(i)}]\Delta t) U_{x,\mu}^{(i)}$
- **Training**: minimization of the loss function $\mathcal{L}(\theta) = \|U_1^* - U(t_1)\|^2$ (e.g. Frobenius norm)

SU(2) toy model

We solve the Wilson flow equation on the previous slide for a single-link configuration in SU(2) with the action $S[U] = \text{Re Tr}(U^2)$. This action has two minima: $\pm \mathbf{1}$. If $\text{Tr} U > 0$ the link flows to the north pole ($+\mathbf{1}$), otherwise it flows to the south pole ($-\mathbf{1}$).

$$u_0 = \frac{1}{2} \text{Tr}(U), \quad u_i = \frac{1}{2} \text{Tr}(U \sigma_i), \quad \tilde{u}_j = u_j / (u_0^2 + u_1^2 + u_2^2)^{1/2}$$



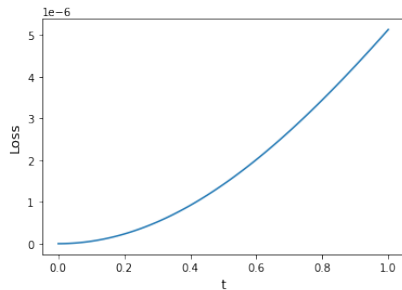
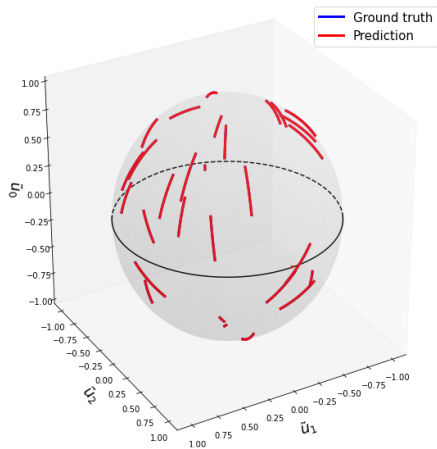
SU(2) toy model

The traceless and hermitian matrix $H[U(t), \theta, t]$ is constructed with the following steps

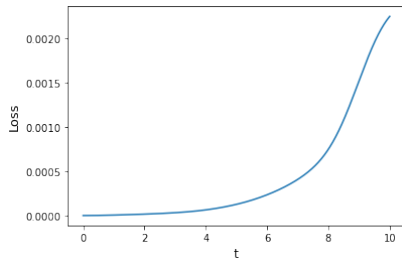
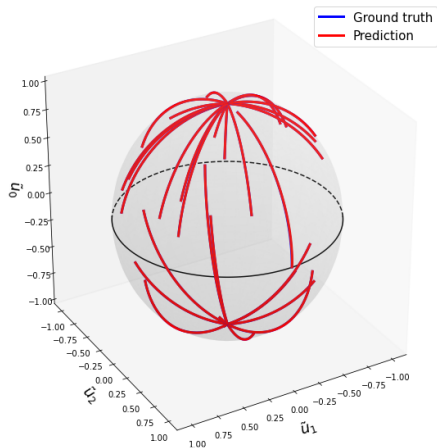
- The complex entries of U are split into real and imaginary part
- They are fed into a multi-layer perceptron with real weights
- The output is recombined into a complex matrix
- Taking the anti-hermitian, $[C]_{\text{ah}} = \frac{1}{2i} (C - C^\dagger) - \frac{1}{2iN_c} \mathbf{1} \text{Tr} (C - C^\dagger)$, projects the output onto $\mathfrak{su}(2)$
- The application of the exponential map yields a matrix in SU(2)

Training is performed by choosing the Frobenius norm as loss function

Testing



Extrapolation to larger times



Conclusions

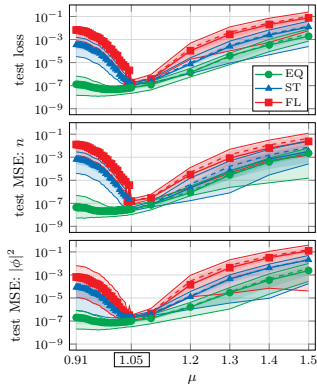
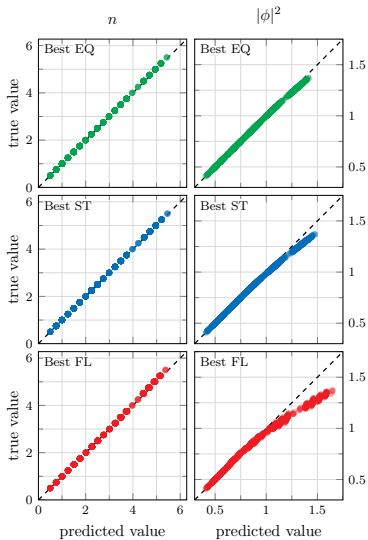
- CNNs that respected translational symmetry outperformed the ones that broke it on a toy model
- L-CNNs are gauge equivariant NNs that can be applied to lattice problems
- Better performance in regression tasks compared to traditional NNs
- They can be used for generating gauge link configurations
- A minimal example of a gradient flow with a single link has been successfully solved

Next steps

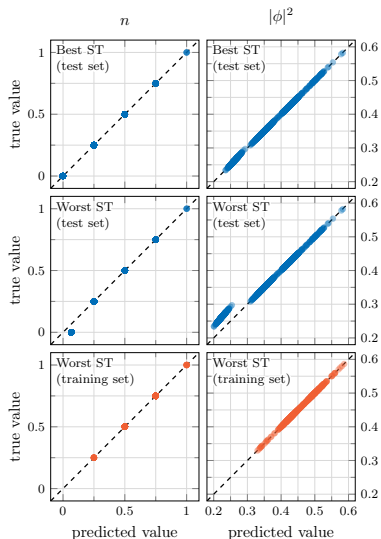
- Extend the toy model to proper lattices
- Apply the L-CNNs to gradient flows

Backup slides

Extrapolation to larger chemical potentials



Why do ST and FL fail?

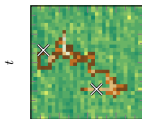


- Worst performing ST instance predicts correctly when tested on $\mu = 1.05$
- Mispredicts values not present in the training set
- Best ST generalizes well
- No EQ instance features a behavior like worst ST

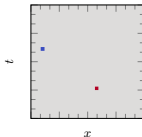
Detecting flux violations

The field k obeys the conservation law $\sum_{\nu} (k_{x,\nu} - k_{x-\hat{\nu},\nu}) = 0$. We artificially created flux violations to be detected by the models.

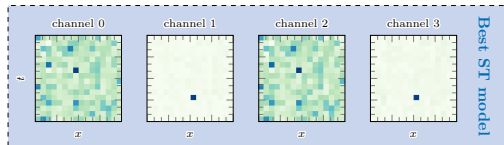
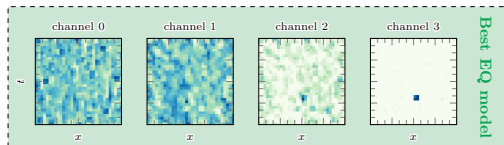
field configuration



flux violation



(a) Example field configuration

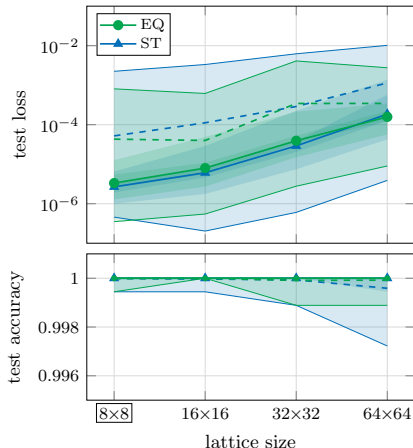
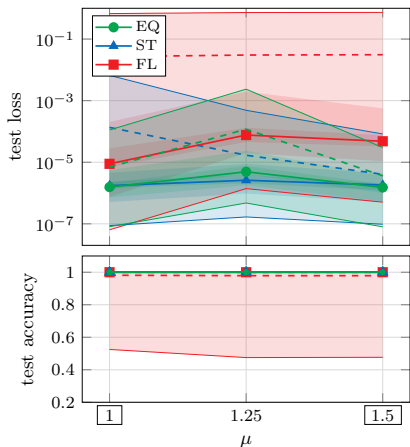


(b) Feature maps of convolutional network in best EQ and ST models

- 2x2 convolutions are necessary for this task
- Similar approach with optuna

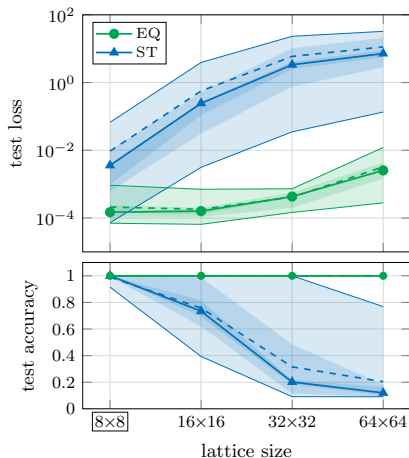
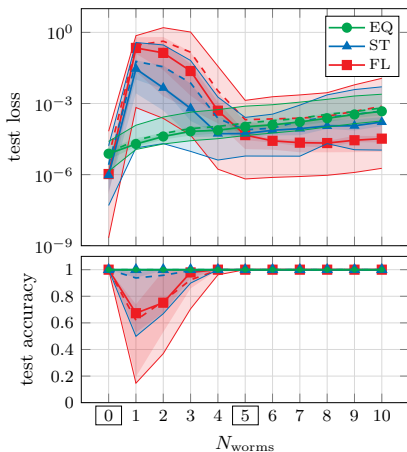
Results

Training at $(\eta, \mu) = (4.25, 1)$ and $(4.01, 1.5)$ on 8×8 lattice with $N_{train} = 4000$; testing at $\eta \in \{4.01, 4.04, 4.25\}$, $\mu \in \{1, 1.25, 1.5\}$ on 4 lattice sizes



Counting flux violations

- Simplified version of counting problems (e.g.: crowd counting)
- Training only at 0 and 5 worms (we train on 4 combinations of parameters out of 396 used for testing) with $N_{train} = 20000$



First task optuna winners

EQ	ST	FL
Conv(1×1 , 4, 64)	Conv(1×1 , 4, 80)	Conv(1×1 , 4, 64)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1 , 64, 48)	Conv(1×1 , 80, 80)	Conv(2×2 , 64, 80)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1 , 48, 80)	Conv(1×1 , 80, 48)	AvgPool(2×2 , 2)
LeakyReLU	LeakyReLU	Conv(1×1 , 80, 48)
Conv(2×2 , 80, 80)	AvgPool(2×2 , 2)	LeakyReLU
LeakyReLU	Conv(2×2 , 48, 80)	Conv(2×2 , 48, 64)
GlobalAvgPool	LeakyReLU	LeakyReLU
Linear(80, 2)	GlobalAvgPool	AvgPool(2×2 , 2)
	Linear(80, 2)	Conv(1×1 , 64, 24)
		Flatten
		Linear(360, 24)
		LeakyReLU
		Linear(24, 2)
33202	26370	47394

Second task optuna winners

EQ	ST	FL
Conv(2×2 , 4, 32)	Conv*(2×2 , 4, 16)	Conv*(3×3 , 4, 8)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1 , 32, 32)	MaxPool(2×2 , 2)	MaxPool(2×2 , 2)
LeakyReLU	Conv(1×1 , 16, 16)	Conv(2×2 , 8, 32)
GlobalMaxPool	LeakyReLU	LeakyReLU
Linear(32, 32)	Conv(1×1 , 16, 8)	AvgPool(2×2 , 2)
LeakyReLU	LeakyReLU	Conv(2×2 , 32, 32)
Linear*(32, 1)	GlobalMaxPool	LeakyReLU
Sigmoid	Linear*(8, 32)	Flatten
	Linear(32, 1)	Linear*(128, 1)
	Sigmoid	Sigmoid
2657	953	5600

The star (e.g. Conv*) indicates that the bias in that layer is set to 0

Third task EQ optuna winners

1st EQ	2nd EQ	3rd EQ
Conv(1×1 , 4, 32)	Conv(2×2 , 4, 8)	Conv(1×1 , 4, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(2×2 , 32, 8)	Conv(2×2 , 8, 8)	Conv(2×2 , 4, 8)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(2×2 , 8, 16)	Conv(1×1 , 8, 4)	Conv(2×2 , 8, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1 , 16, 8)	Conv(1×1 , 4, 8)	Conv(3×3 , 4, 1)
LeakyReLU	LeakyReLU	LeakyReLU
GlobalSumPool	GlobalSumPool	GlobalSumPool
Linear(8, 1)	Linear(8, 1)	
1800	456	308

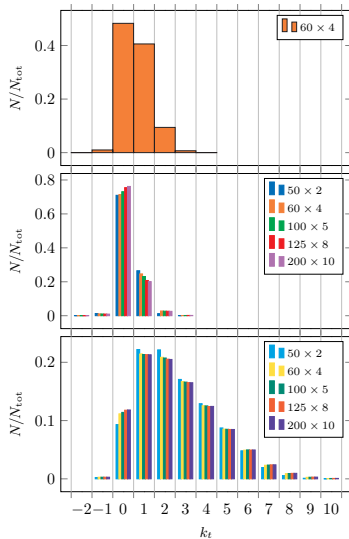
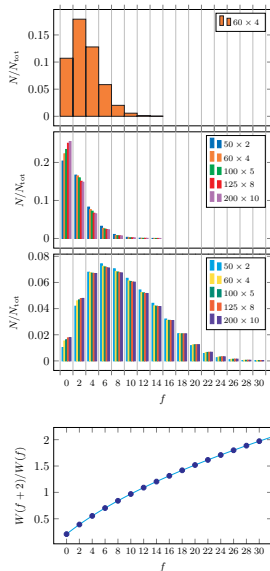
Third task ST optuna winners

1st ST	2nd ST	3rd ST
Conv(2×2 , 4, 16)	Conv(2×2 , 4, 4)	Conv(2×2 , 4, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1 , 16, 32)	MaxPool(2×2 , 2)	AvgPool(2×2 , 2)
LeakyReLU	Conv(2×2 , 4, 4)	Conv(3×3 , 4, 16)
Conv(1×1 , 32, 32)	LeakyReLU	LeakyReLU
LeakyReLU	GlobalSumPool	GlobalSumPool
AvgPool(2×2 , 2)	Linear(4, 1)	Linear(16, 32)
Conv(1×1 , 32, 8)		LeakyReLU
LeakyReLU		Linear(32, 1)
GlobalSumPool		
Linear(8, 32)		
LeakyReLU		
Linear(32, 1)		
2336	132	1184

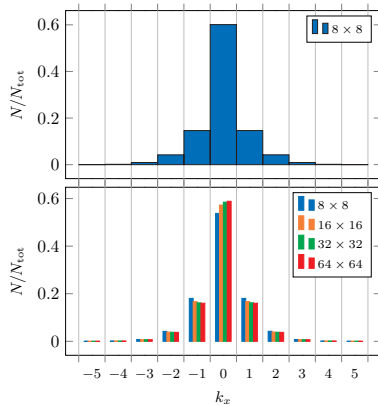
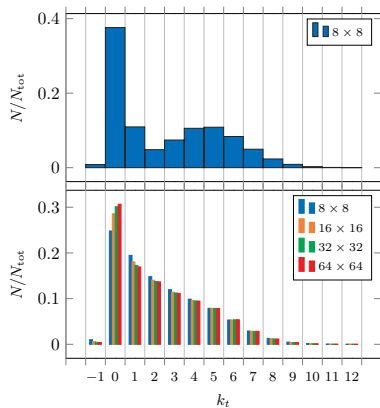
Third task FL optuna winners

1st FL	2nd FL	3rd FL
Conv(2×2 , 4, 4)	Conv(2×2 , 4, 8)	Conv(2×2 , 4, 32)
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool(2×2 , 2)	AvgPool(2×2 , 2)	AvgPool(2×2 , 2)
Conv(3×3 , 4, 8)	Conv(3×3 , 8, 4)	Conv(3×3 , 32, 4)
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool(2×2 , 2)	AvgPool(2×2 , 2)	AvgPool(2×2 , 2)
Flattening	Flattening	Flattening
Linear(8, 4)	Linear(4, 4)	Linear(4, 32)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(4, 32)	Linear(4, 32)	Linear(32, 16)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(32, 1)	Linear(32, 1)	Linear(16, 1)
640	640	2704

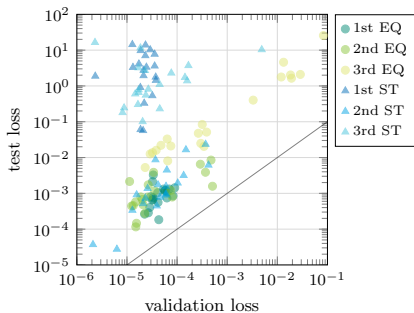
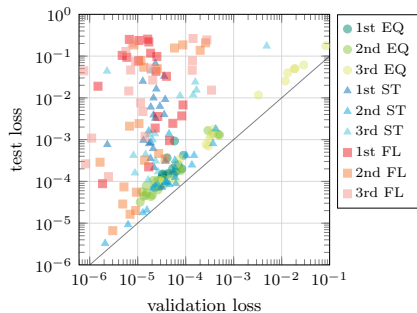
First task data distribution



Third task data distribution



Third task: test loss vs validation loss



Test loss vs validation loss table

	validation loss on 8×8		test loss on 8×8		test loss up to 64×64	
	mean	median	mean	median	mean	median
1st EQ	4.676×10^{-5}	4.137×10^{-5}	2.108×10^{-4}	1.483×10^{-4}	1.008×10^{-3}	8.308×10^{-4}
2nd EQ	1.042×10^{-4}	2.440×10^{-5}	3.525×10^{-4}	8.783×10^{-5}	1.807×10^{-3}	7.936×10^{-4}
3rd EQ	8.992×10^{-3}	3.072×10^{-4}	2.105×10^{-2}	9.163×10^{-4}	1.925	4.031×10^{-2}
1st ST	2.331×10^{-5}	2.173×10^{-5}	9.438×10^{-3}	3.576×10^{-3}	4.446	3.026
2nd ST	8.479×10^{-5}	4.372×10^{-5}	2.545×10^{-4}	9.340×10^{-5}	3.738×10^{-3}	1.171×10^{-3}
3rd ST	2.869×10^{-4}	2.171×10^{-5}	1.676×10^{-2}	1.381×10^{-3}	2.943	9.580×10^{-1}
1st FL	2.602×10^{-5}	1.787×10^{-5}	7.837×10^{-2}	3.817×10^{-2}	-	-
2nd FL	4.004×10^{-5}	1.117×10^{-5}	5.300×10^{-2}	1.285×10^{-3}	-	-
3rd FL	5.805×10^{-5}	1.031×10^{-5}	6.382×10^{-2}	3.556×10^{-2}	-	-

Regression of Wilson loops

Benchmark problem: regression of Wilson loops from 1×1 to 4×4 in 2D

- Dataset: SU(2) gauge field configurations \mathcal{U} from MC simulation, $\beta \in \{0.1, \dots, 6.0\}$
- Input (“x”): gauge field configuration $\mathcal{U} \in \mathbb{C}^{N_x \times N_y \times 2 \times N_c \times N_c}$
- Output (“y”): $\text{Re Tr} [W^{(n \times m)}] / N_c \in \mathbb{R}$
- Metric: mean squared error (MSE)
- Training on small lattice: 8×8 (10^4 samples)
- Testing on larger lattices: $8 \times 8, 16 \times 16, 32 \times 32, 64 \times 64$ (10^3 samples)

Comparison study

- **L-CNN models:** 1 – 4 **L-Conv** + **L-BL** layers
 $\mathcal{O}(10) - \mathcal{O}(10^4)$ trainable parameters, **100** individual models
- **Baseline models:** traditional CNNs, up to 6 layers, up to 512 channels, 4 activation functions
 $\mathcal{O}(100) - \mathcal{O}(10^5)$ trainable parameters, **2840** individual models

- Smoothing technique
- In 1D, one can define it as

$$\dot{x}(t) = -\lambda \frac{\partial S[x]}{\partial x} x(t),$$

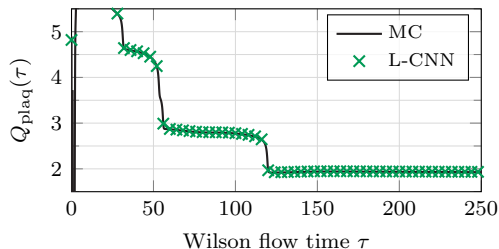
which evolves the variable x towards a local minimum of the action

- In lattice QCD the Wilson flow can be defined as the solution of

$$\dot{U}_{x,\mu}(t) = i\lambda t^a \frac{\delta S[U]}{\delta A^a} U_{x,\mu}(t),$$

where $\delta U = i\delta A U$ and t is the Wilson flow time

Regression on topological charge



- L-CNN predictions vs. true values (MC) for

$$Q_{\text{plaq}} = \sum_{x \in \Lambda} q_x^{\text{plaq}} = \sum_{x \in \Lambda} \frac{\epsilon_{\mu\nu\rho\sigma}}{32\pi^2} \text{Tr} \left[\frac{U_{x,\mu\nu} - U_{x,\mu\nu}^\dagger}{2i} \frac{U_{x,\rho\sigma} - U_{x,\rho\sigma}^\dagger}{2i} \right] \text{ on an}$$

8×24^3 configuration

- Training on 4×8^3 lattices and testing performed also on larger lattices with cooled configurations

Other applications of L-CNNs

- "Gauge-equivariant neural networks as preconditioners in lattice QCD", C. Lehner, T. Wettig, arXiv:2302.05419
- "Fixed point actions from convolutional neural networks", K. Holland, A. Ipp, D. I. Müller, U. Wenger, arXiv:2311.17816

Constraints on H

- Hermitian

$$UU^\dagger = \mathbf{1} \implies \frac{d}{dt}(UU^\dagger) = 0 \implies H = H^\dagger$$

- Traceless

$$U(t + dt) = e^{iH dt} U(t)$$

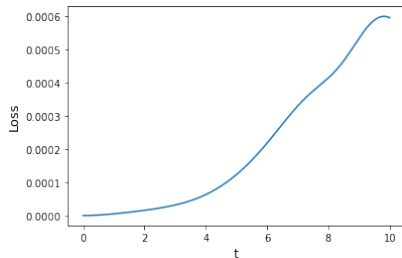
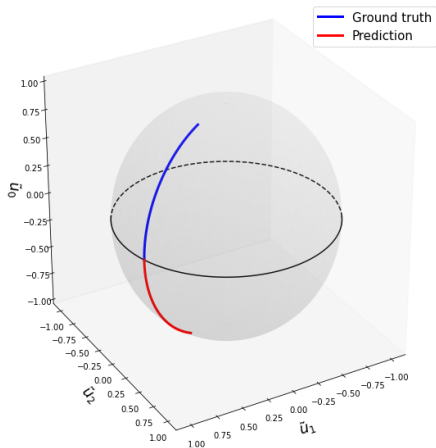
$$\det U(t + dt) = \det e^{iH dt} \det U(t) \implies e^{i \text{Tr} H dt} = 1 \implies \text{Tr} H = 0$$

- Locally transforming

$$U_{x,\mu}(t + dt) = e^{iH_\mu dt} U_{x,\mu}(t) = \sum_{n=0}^{\infty} \frac{1}{n!} (iH_\mu dt)^n U_{x,\mu}(t)$$

$$\Omega_x U_{x,\mu}(t + dt) \Omega_{x+\mu}^\dagger = \Omega_x \left(\mathbf{1} + i dt H_\mu - \frac{1}{2} dt H_\mu H_\mu \dots \right) U_{x,\mu}(t) \Omega_{x+\mu}^\dagger$$

Mispredicted directions



Best network's hyperparameters

- training set size=50000
- epochs=100
- batch size=100
- nodes={16, 64, 32, 16}
- bias=true
- activation function: tanh
- learning rate=0.001
- test set size=4000

- Computing gradients for the model parameters needs to keep track of all intermediate points of the evolution
- Memory may be saturated
- The adjoint sensitivity method circumvents this problem by calculating the gradients backward from the last point instead of storing the whole history