Preserving gauge invariance in neural networks

David I. Müller

Institute for Theoretical Physics, TU Wien, Austria

dmueller@hep.itp.tuwien.ac.at

A Virtual Tribute to Quark Confinement and the Hadron Spectrum 2021

August 6, 2021

Based on

M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Lattice gauge equivariant convolutional neural networks"
Preprint (2020) [arXiv:2012.12901]

Code: gitlab.com/openpixi/lge-cnn Group: openpixi.org

TECHNISCHE UNIVERSITÄT WIEN Vienna|Austria

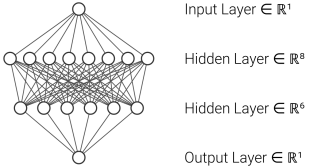
FWF Der Wissenschaftsfonds.

Introduction

Applications of machine learning (ML):
Computer vision, natural language processing, medicine and (high energy) physics

Artificial neural networks (ANNs or NNs)

- Highly expressive basis for function approximation
- Universal approximators for non-linear functions
- Typically high number of free parameters, "black boxes"



Neural networks applied to physical data (e.g. field theory)

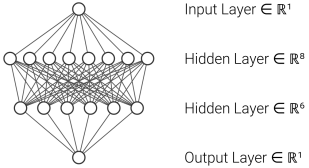
- High expressivity: NNs a priori do not know about symmetry
- Symmetries in data have to be learned (approximated)
- Alternative: restrict parameters to enforce symmetry
- This work: NNs which respect (non-Abelian) gauge symmetry

Introduction

Applications of machine learning (ML):
Computer vision, natural language processing, medicine and (high energy) physics

Artificial neural networks (ANNs or NNs)

- Highly expressive basis for function approximation
- Universal approximators for non-linear functions
- Typically high number of free parameters, "black boxes"



Neural networks applied to physical data (e.g. field theory)

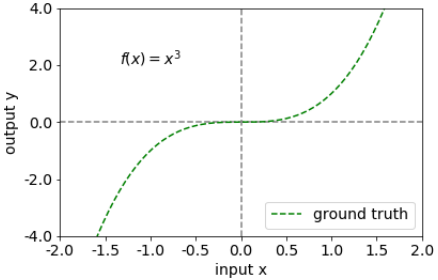
- High expressivity: NNs a priori do not know about symmetry
- Symmetries in data have to be learned (approximated)
- Alternative: restrict parameters to enforce symmetry
- This work: NNs which respect (non-Abelian) gauge symmetry

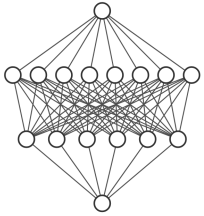
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Input Layer $\in \mathbb{R}^1$

Hidden Layer $\in \mathbb{R}^8$

Hidden Layer $\in \mathbb{R}^6$

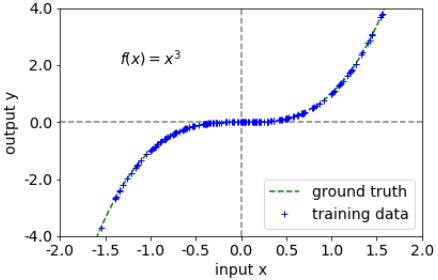
Output Layer $\in \mathbb{R}^1$

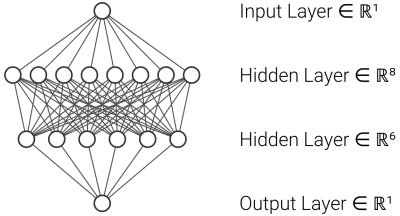
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)



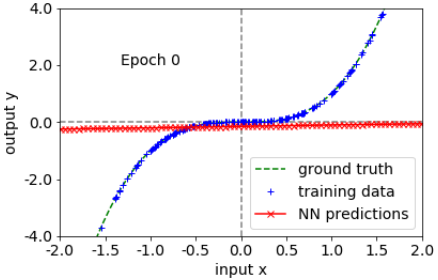


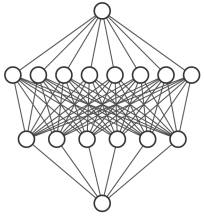
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Input Layer $\in \mathbb{R}^1$

Hidden Layer $\in \mathbb{R}^8$

Hidden Layer $\in \mathbb{R}^6$

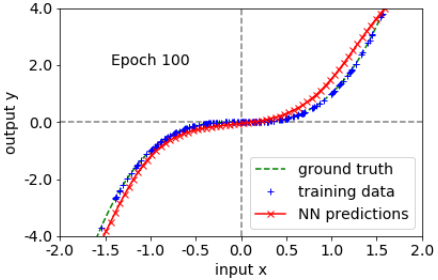
Output Layer $\in \mathbb{R}^1$

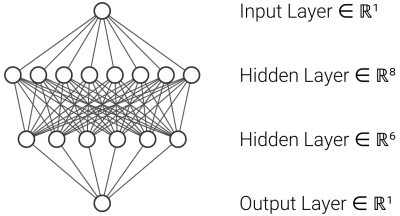
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)



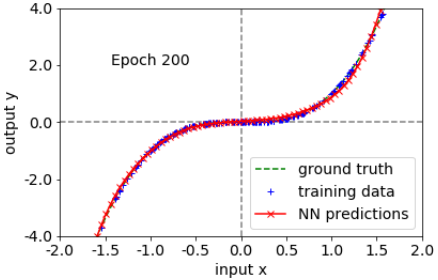


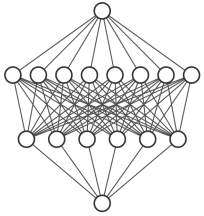
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Input Layer $\in \mathbb{R}^1$

Hidden Layer $\in \mathbb{R}^8$

Hidden Layer $\in \mathbb{R}^6$

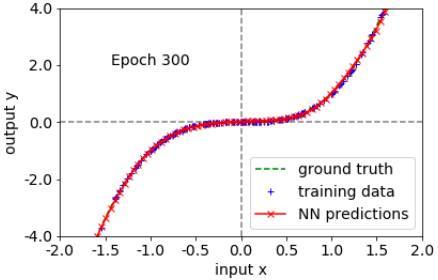
Output Layer $\in \mathbb{R}^1$

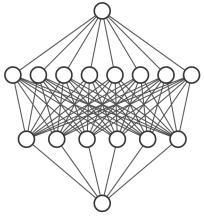
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Input Layer $\in \mathbb{R}^1$

Hidden Layer $\in \mathbb{R}^8$

Hidden Layer $\in \mathbb{R}^6$

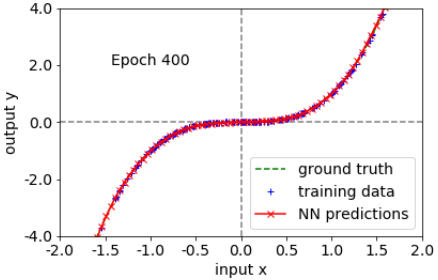
Output Layer $\in \mathbb{R}^1$

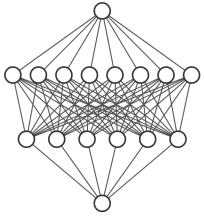
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Input Layer $\in \mathbb{R}^1$

Hidden Layer $\in \mathbb{R}^8$

Hidden Layer $\in \mathbb{R}^6$

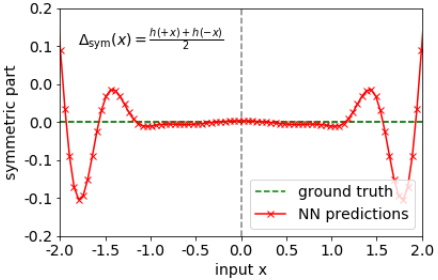
Output Layer $\in \mathbb{R}^1$

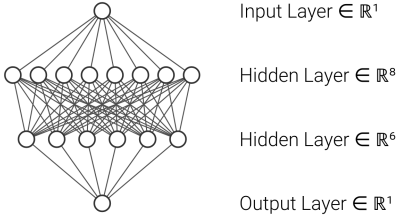
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)



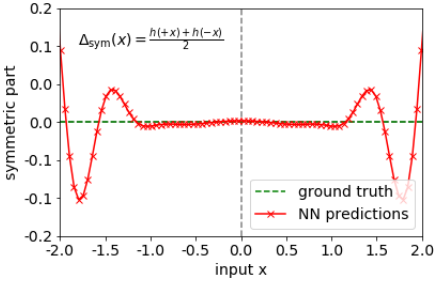


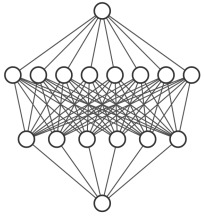
Symmetries and neural networks

Toy model: non-linear regression for $f(x) = x^3$

- Training data: input data $\mathbf{x} \in \mathbb{R}^N$ and output data (labels) $\mathbf{y} \in \mathbb{R}^N$
- Neural network $h_\theta : \mathbb{R} \to \mathbb{R}$ with unknown parameters $\theta \in \mathbb{R}^M$

  $\Rightarrow$ Composition of affine transformations and non-linear activation functions
- Minimize loss function: $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i\right)^2$ (training)





Symmetries in NNs are generally only approximate!

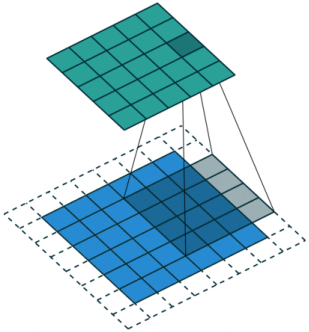
$$f(-x) = -f(x) \qquad h_\theta(-x) \neq -h_\theta(x)$$

Convolutional neural networks

Convolutional NNs (CNNs) use translational equivariance for data on lattices (e.g. images)

- Restrict parameters to enforce translational equivariance
- Compact kernels (locality)
  $\rightarrow$ only consider compact neighborhoods
- Weight sharing (homogeneity)
  $\rightarrow$ same operation at every point
- Translational equivariance
  "Translations on input induce translations on output"
- More general: G-CNNs (rotations, reflections, ...)
- Symmetry is not learned, but implemented
- Applications in lattice field theories e.g. ϕ^4 (next talk by Matteo!)

S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Generalization capabilities of translationally equivariant neural networks"
Preprint (2021) [arXiv:2103.14686]

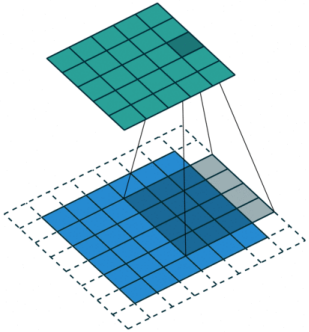


Convolutional neural networks

Convolutional NNs (CNNs) use translational equivariance for data on lattices (e.g. images)

- Restrict parameters to enforce translational equivariance
- Compact kernels (locality)
  $\rightarrow$ only consider compact neighborhoods
- Weight sharing (homogeneity)
  $\rightarrow$ same operation at every point
- Translational equivariance
  "Translations on input induce translations on output"
- More general: G-CNNs (rotations, reflections, ...)
- Symmetry is not learned, but implemented
- Applications in lattice field theories e.g. ϕ^4 (next talk by Matteo!)

S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Generalization capabilities of translationally equivariant neural networks"
Preprint (2021) [arXiv:2103.14686]

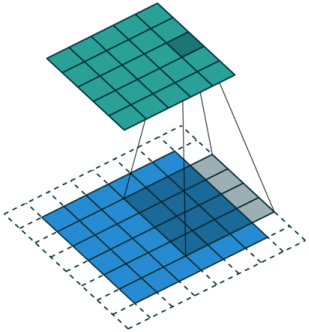


Convolutional neural networks

Convolutional NNs (CNNs) use translational equivariance for data on lattices (e.g. images)

- Restrict parameters to enforce translational equivariance
- Compact kernels (locality)
  $\rightarrow$ only consider compact neighborhoods
- Weight sharing (homogeneity)
  $\rightarrow$ same operation at every point
- Translational equivariance
  "Translations on input induce translations on output"
- More general: G-CNNs (rotations, reflections, ...)
- Symmetry is not learned, but implemented
- Applications in lattice field theories e.g. ϕ^4 (next talk by Matteo!)

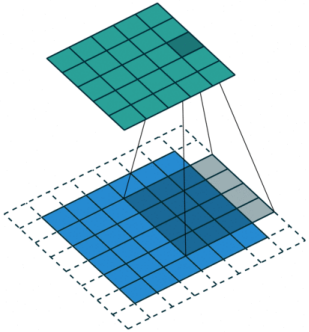
S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Generalization capabilities of translationally equivariant neural networks"
Preprint (2021) [arXiv:2103.14686]



Convolutional neural networks

Convolutional NNs (CNNs) use translational equivariance for data on lattices (e.g. images)

- Restrict parameters to enforce translational equivariance
- Compact kernels (locality)
  $\rightarrow$ only consider compact neighborhoods
- Weight sharing (homogeneity)
  $\rightarrow$ same operation at every point
- Translational equivariance
  "Translations on input induce translations on output"
- More general: G-CNNs (rotations, reflections, ...)
- Symmetry is not learned, but implemented
- Applications in lattice field theories e.g. ϕ^4 (next talk by Matteo!)

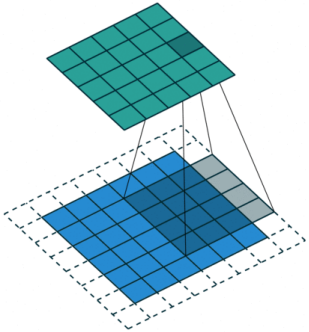


S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Generalization capabilities of translationally equivariant neural networks"
Preprint (2021) [arXiv:2103.14686]

Convolutional neural networks

Convolutional NNs (CNNs) use translational equivariance for data on lattices (e.g. images)

- Restrict parameters to enforce translational equivariance
- Compact kernels (locality)
  $\rightarrow$ only consider compact neighborhoods
- Weight sharing (homogeneity)
  $\rightarrow$ same operation at every point
- Translational equivariance
  "Translations on input induce translations on output"
- More general: G-CNNs (rotations, reflections, ...)
- Symmetry is not learned, but implemented
- Applications in lattice field theories e.g. ϕ^4 (next talk by Matteo!)



S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, D. Schuh,
"Generalization capabilities of translationally equivariant neural networks"
Preprint (2021) [arXiv:2103.14686]

Gauge equivariance and invariance

Lattice gauge theory: gauge theories on lattices with exact gauge invariance

- Gauge links $\mathcal{U}$ and 1×1 loops $\mathcal{W}$

$$U_{\mathbf{x},\mu} \simeq \exp\left(iga^\mu A_\mu(\mathbf{x}+\mathbf{a}^\mu/2)\right) \in \mathrm{SU}(N_c)$$
$$W_{\mathbf{x},\mu\nu} = U_{\mathbf{x},\mu} U_{\mathbf{x}+\mu,\nu} U_{\mathbf{x}+\mu+\nu,-\mu} U_{\mathbf{x}+\nu,-\nu}$$

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

$$T_\Omega U_{\mathbf{x},\mu} = \Omega_{\mathbf{x}} U_{\mathbf{x},\mu} \Omega^\dagger_{\mathbf{x}+\mu}, \qquad \Omega_{\mathbf{x}} \in \mathrm{SU}(N_c)$$
$$T_\Omega W_{\mathbf{x},\mu\nu} = \Omega_{\mathbf{x}} W_{\mathbf{x},\mu\nu} \Omega^\dagger_{\mathbf{x}}$$

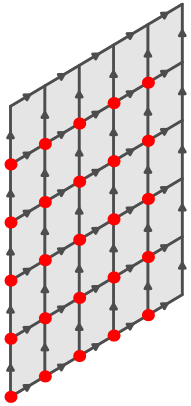
- Gauge equivariant function

$$g(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = T'_\Omega g(\mathcal{U}, \mathcal{W})$$

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

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

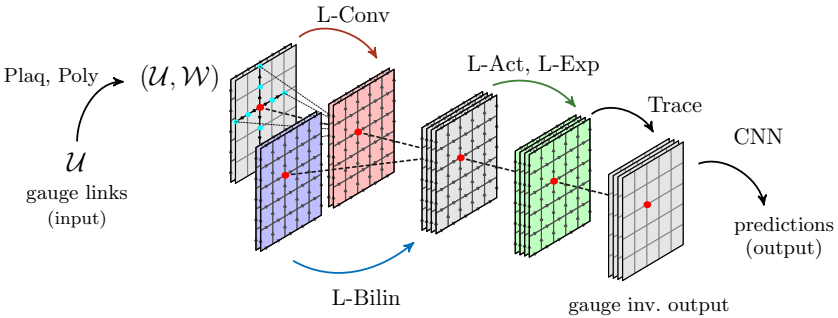
- Neural network layers should be gauge equivariant!



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

Lattice gauge equivariant convolutional neural networks

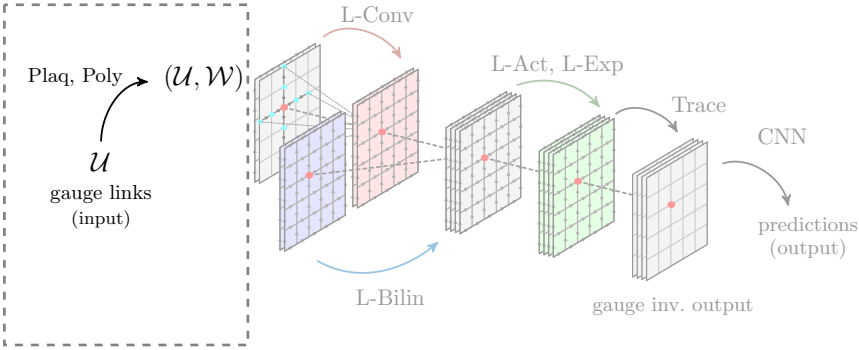
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

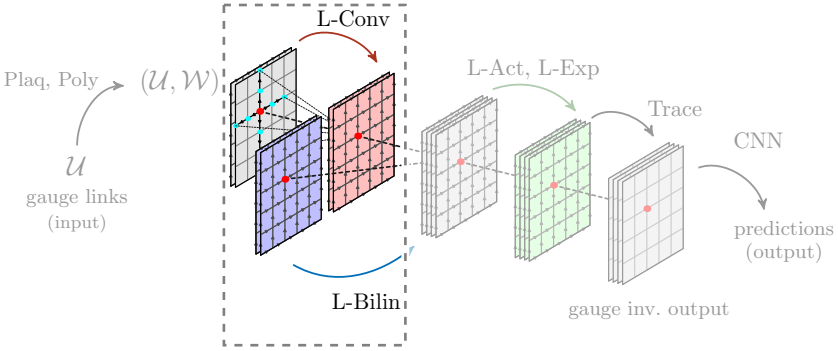
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

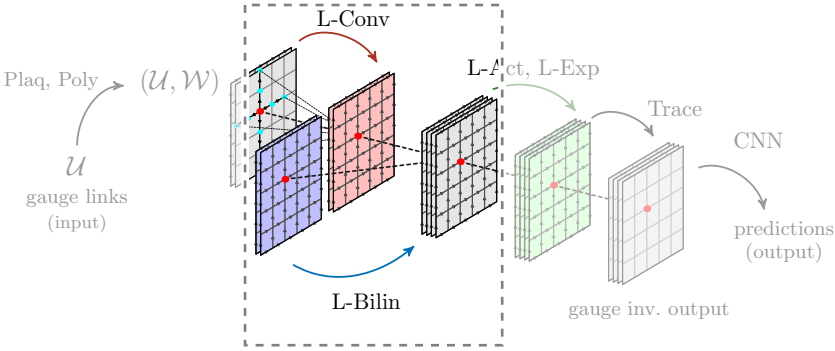
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

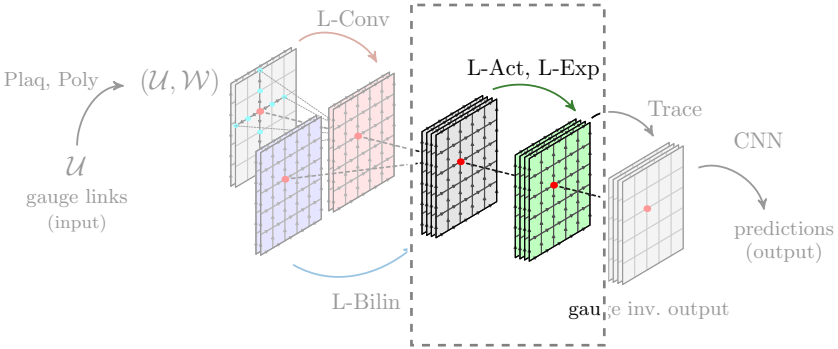
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

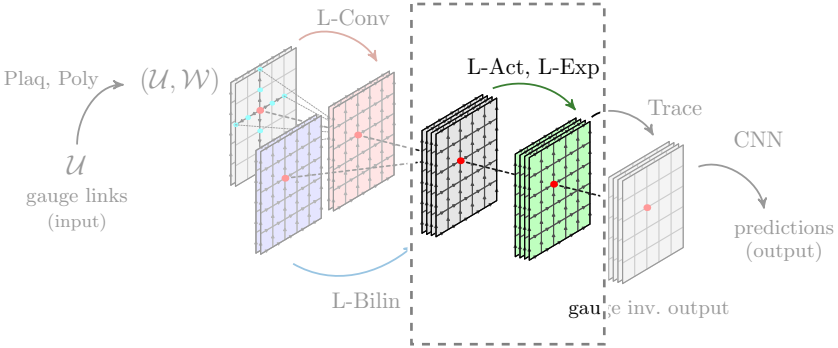
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

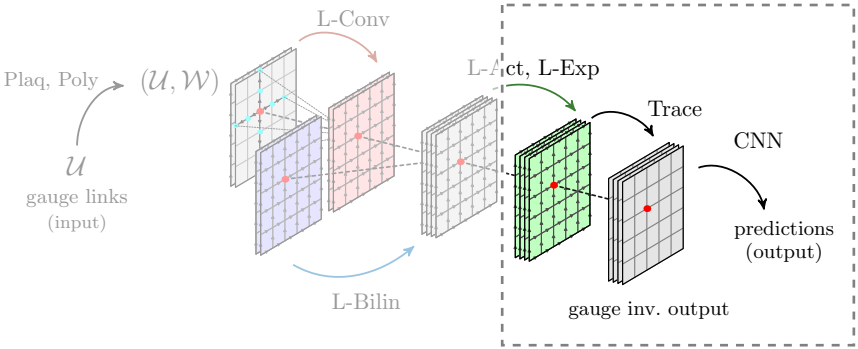
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

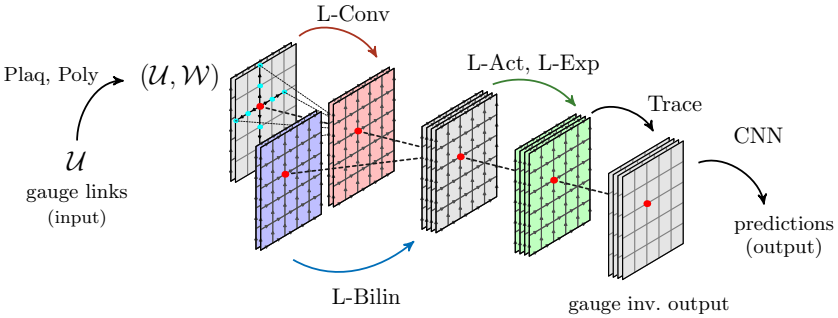
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Lattice gauge equivariant convolutional neural networks

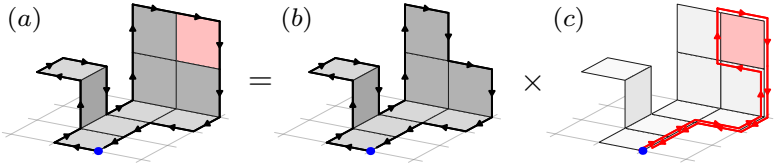
L-CNNs: A collection of gauge equivariant layers for lattice gauge configurations



- Preprocessing: **Plaq, Poly**
  $\mathcal{U} \to (\mathcal{U}, \mathcal{W})$
  [no trainable parameters]
- Convolutions: **L-Conv**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}')$
  [Conv + parallel transport]
- Bilinear layer: **L-Bilin**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W} \cdot \mathcal{W}')$
  [local matrix mult.]
- Activation layer: **L-Act**
  $(\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, a(\mathcal{W}) \cdot \mathcal{W})$
  [local scalar mult.]
- Exponential maps: **L-Exp**
  $(\mathcal{U}, \mathcal{W}) \to (e^{i\omega} \cdot \mathcal{U}, \mathcal{W})$
  [local matrix mult.]
- Postprocessing: **Trace**
  Invariant output
  [no trainable parameters]

Arbitrary Wilson loops using L-CNNs

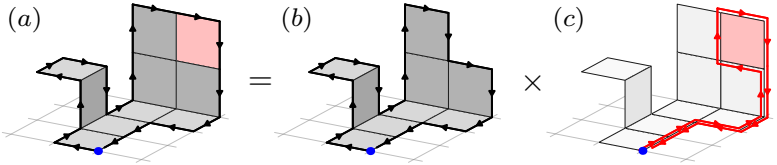
- Repeated applications of **L-Conv** and **L-Bilin** operations can be used to generate arbitrarily sized Wilson loops if input $\mathcal{W}$ consists of plaquettes (preprocessing layer **Plaq**)



- 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

Arbitrary Wilson loops using L-CNNs

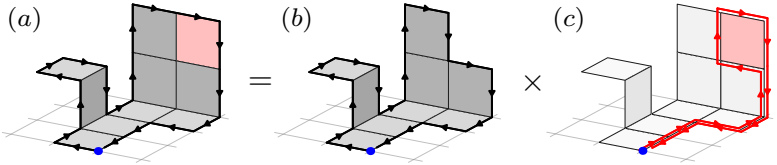
- Repeated applications of **L-Conv** and **L-Bilin** operations can be used to generate arbitrarily sized Wilson loops if input $\mathcal{W}$ consists of plaquettes (preprocessing layer **Plaq**)



- 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

Arbitrary Wilson loops using L-CNNs

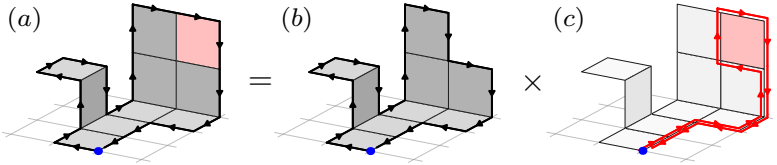
- Repeated applications of **L-Conv** and **L-Bilin** operations can be used to generate arbitrarily sized Wilson loops if input $\mathcal{W}$ consists of plaquettes (preprocessing layer **Plaq**)



- 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

Arbitrary Wilson loops using L-CNNs

- 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 **Plaq**)



- 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

Benchmarks and testing I

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

- Data set: SU(2) gauge field configurations $\mathcal{U}$ from MC simulation, $\beta \in \{0.1, \ldots, 6.0\}$
- 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"): $\mathrm{Re}\,\mathrm{Tr}\left[W^{(n \times m)}\right]/N_c \in \mathbb{R}$
- 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-Bilin** 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, **2680** individual models
- Both architectures get same information (links $\mathcal{U}$, plaquettes $\mathcal{W}$)

Benchmarks and testing I

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

- Data set: SU(2) gauge field configurations $\mathcal{U}$ from MC simulation, $\beta \in \{0.1, \ldots, 6.0\}$
- 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"): $\mathrm{Re}\,\mathrm{Tr}\left[W^{(n \times m)}\right]/N_c \in \mathbb{R}$
- 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

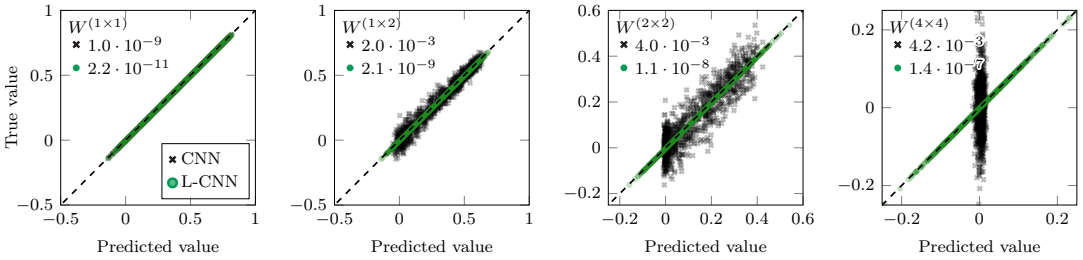
- L-CNN models: 1 – 4 **L-Conv** + **L-Bilin** layers

  $\mathcal{O}(10)$ – $\mathcal{O}(10^4)$ 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, **2680** individual models
- Both architectures get same information (links $\mathcal{U}$, plaquettes $\mathcal{W}$)

Benchmarks and testing II

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

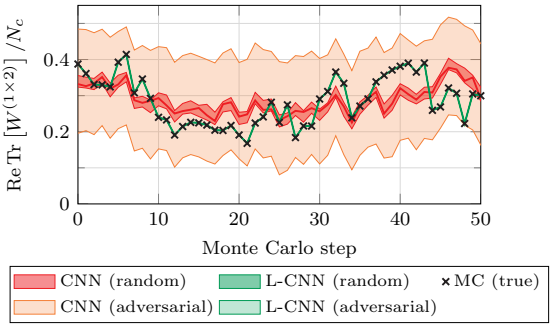


Plot: True vs. predicted values for CNNs and L-CNNs for $n \times m$ Wilson loops (best models)

- From left to right: increase in loop size $\rightarrow$ more difficult task
- 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

Benchmarks and testing III

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



Plot: Sensitivity to gauge transformations for 1×2 Wilson loop

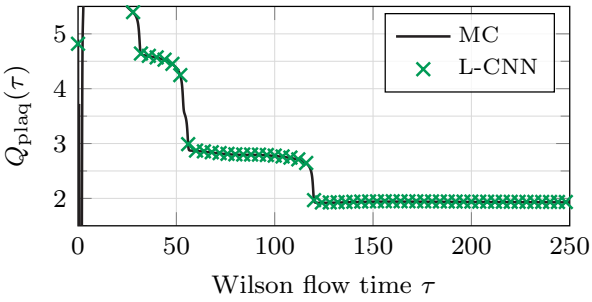
- Test for sensitivity to random and adversarial gauge transformations

$$T_\Omega U_{\mathbf{x},\mu} = \Omega_{\mathbf{x}} U_{\mathbf{x},\mu} \Omega^\dagger_{\mathbf{x}+\mu}, \qquad T_\Omega W_{\mathbf{x},\mu\nu} = \Omega_{\mathbf{x}} W_{\mathbf{x},\mu\nu} \Omega^\dagger_{\mathbf{x}}$$

- Baseline CNNs are sensitive to gauge transformations, L-CNNs are invariant

Benchmarks and testing IV

L-CNNs also work in higher dimensions!



Plot: L-CNN predictions vs. true values (MC) for Q_{plaq} on a 8×24^3 configuration

- L-CNN model for topological charge prediction
- Trained on MC configurations
- Tested on "cooled" configurations (Wilson flow)

Summary and outlook

L-CNNs: framework for lattice gauge equivariant convolutional neural networks

- Fully respects SU(N_c) gauge symmetry
- Universal approximators for gauge invariant functions
- Better performance than traditional CNNs in presented regression tasks
- Gauge equivariant replacement for traditional CNNs
- Open source (based on *PyTorch*)

  Repository: gitlab.com/openpixi/lge-cnn
  Our group: openpixi.org

What's next?

- Improvements to code (more modules, performance, memory consumption, ...)
- More complicated observables
- Application to normalizing flows?

Summary and outlook

L-CNNs: framework for lattice gauge equivariant convolutional neural networks

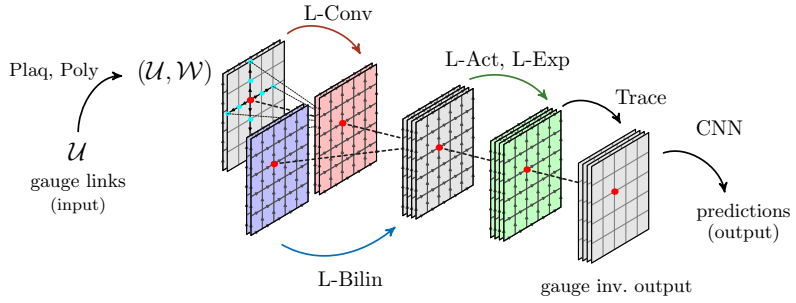
- Fully respects SU(N_c) gauge symmetry
- Universal approximators for gauge invariant functions
- Better performance than traditional CNNs in presented regression tasks
- Gauge equivariant replacement for traditional CNNs
- Open source (based on *PyTorch*)

  Repository: gitlab.com/openpixi/lge-cnn
  Our group: openpixi.org

What's next?

- Improvements to code (more modules, performance, memory consumption, ...)
- More complicated observables
- Application to normalizing flows?

Thank you for your attention!



Code: gitlab.com/openpixi/lge-cnn Group: openpixi.org

E-Mail: dmueller@hep.itp.tuwien.ac.at

Backup

Related works

Incomplete list of recent related works (from other authors, chronologically)

- P. E. Shanahan, A. Trewartha, W. Detmold
  "Machine learning action parameters in lattice quantum chromodynamics"
  PRD 97 (2018) [arXiv:1801.05784]
- T. S. Cohen, M. Weiler, B. Kicanaoglu, M. Welling
  "Gauge Equivariant Convolutional Networks and the Icosahedral CNN"
  ICML 2019 (2019) [arXiv:1902.04615]
- G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racanière, D. J. Rezende, P. E. Shanahan
  "Equivariant flow-based sampling for lattice gauge theory"
  PRL 125 (2020) [arXiv:2003.06413]
- D. Boyda, G. Kanwar, M. S. Albergo, S. Racanière, D. J. Rezende, M. S. Albergo, K. Cranmer, D. C. Hackett, P. E. Shanahan
  "Sampling using SU(N) gauge equivariant flows"
  PRD 103 (2021) [arXiv:2008.05456]
- D. L. Boyda, M. N. Chernodub, N. V. Gerasimeniuk, V. A. Goy, S. D. Liubimov, A. V. Molochkov
  "Machine-learning physics from unphysics: Finding deconfinement temperature in lattice Yang-Mills theories from outside the scaling window"
  PRD 103 (2021) [arXiv:2009.10971]
- A. Tomiya, Y. Nagai
  "Gauge covariant neural network for 4 dimensional non-abelian gauge theory"
  Preprint (2021) [arXiv:2103.11965]

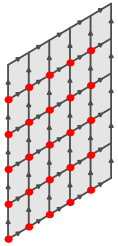
Individual layers

Preprocessing and equivariant convolutions

Preprocessing layers (**Plaq, Poly**)

- Operations defined for tuples $(\mathcal{U}, \mathcal{W})$ with gauge links $\mathcal{U}$ and locally transforming matrices $\mathcal{W}$
- Preprocess input $\mathcal{U}$ to generate $\mathcal{W}$

$$\textbf{Plaq, Poly} : \mathcal{U} \to (\mathcal{U}, \mathcal{W})$$

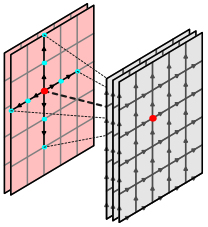


$\mathcal{U} = \{U_{\mathbf{x},\mu}\}$
$\mathcal{W} = \{W_{\mathbf{x},\mu\nu}\}$

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

- Similar to CNN layers: compact kernels, weight sharing
- Parallel transport of data $\mathcal{W}$ to common point using $\mathcal{U}$
- Path (in)dependence, implementation for D dimensions

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



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

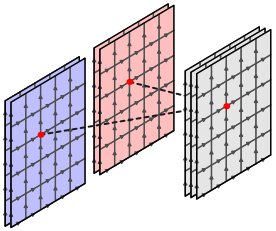
Bilinear layers and activation functions

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

- Multiplication of $\mathcal{W}$'s at same lattice point is equivariant

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

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

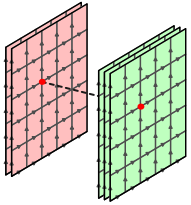


Gauge equivariant activation functions (**L-Act**):

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

$$\textbf{L-Act} : W'_{\mathbf{x}} = a(\mathrm{Tr} W_{\mathbf{x}})\, W_{\mathbf{x}}$$

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



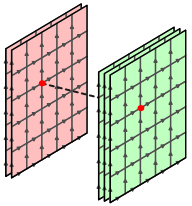
Exponential layers and the trace operation

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

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

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

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

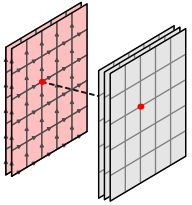


Generate gauge invariant output (**Trace**)

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

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

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



Random and adversarial gauge transformations

Random gauge transformations

Predictions of networks without gauge invariance can be sensitive to gauge transformations

- Trained neural network: $h_\theta : \mathcal{U} \to \mathbb{R}$
- Choose lattice configuration $\mathcal{U}$ and save original prediction

$$y_0 = h_\theta(\mathcal{U})$$

- Apply random gauge transformation with random Gaussian numbers $\rho^a_{\mathbf{x}}$

$$\Omega_{\mathbf{x}} = \exp\left(it^a \rho^a_{\mathbf{x}}\right), \qquad \left\langle \rho^a_{\mathbf{x}} \right\rangle = 0, \quad \left\langle \rho^a_{\mathbf{x}} \rho^b_{\mathbf{x}'} \right\rangle = \alpha \delta^{ab} \delta_{\mathbf{x}\mathbf{x}'}$$

$$\mathcal{U} \to \mathcal{U}' : \qquad U'_{\mathbf{x},\mu} = \Omega_{\mathbf{x}} U_{\mathbf{x},\mu} \Omega^\dagger_{\mathbf{x}+\mu}$$

- Save transformed prediction

$$y' = h_\theta(\mathcal{U}')$$

- Repeat N times to generate $\mathcal{Y} = \{y'_1, y'_2, \ldots, y'_N\}$
- $\max(\mathcal{Y}) - \min(\mathcal{Y})$ as a measure of sensitivity to gauge transformations

Adversarial gauge transformations

Predictions of networks without gauge invariance can be sensitive to gauge transformations

- Trained neural network: $h_\theta : \mathcal{U} \to \mathbb{R}$
- Choose lattice configuration $\mathcal{U}$ and record original prediction $y_0 = h_\theta(\mathcal{U})$
- Gauge transformation $\Omega_{\mathbf{x}}$ with parameters $\rho^a_{\mathbf{x}}$: $\Omega_{\mathbf{x}} = \exp\left(it^a \rho^a_{\mathbf{x}}\right)$
- Initialize $\rho^a_{\mathbf{x}}$ randomly and save prediction

$$y_\rho = h_\theta(\mathcal{U}_\rho)$$

- Loss function

$$\mathcal{L}[\rho] = (y_0 - y_\rho)^2$$

- *Maximize* loss function $\mathcal{L}[\rho]$ w.r.t. ρ via gradient descent (using *PyTorch*)
- Repeat process with multiple random initializations for ρ and record results $\mathcal{Y} = \{y_{\rho_1}, y_{\rho_2}, \ldots, y_{\rho_N}\}$
- $\max(\mathcal{Y}) - \min(\mathcal{Y})$ as a measure of sensitivity to gauge transformations

L-CNN and CNN architectures

Baseline CNNs for 1×1 and 1×2 loops

- Various input data: $\mathcal{U}$, $\mathcal{U}+\mathcal{W}$, $\mathcal{U}+\mathcal{W}+\mathcal{W}^\dagger$ (later just $\mathcal{U}+\mathcal{W}+\mathcal{W}^\dagger$)
- Activation functions: *ReLU*, *LeakyReLU*, *tanh*, *sigmoid*
- Architecture sizes: "small", "medium", "large", "wide"

$W^{(1\times1)}$, $W^{(1\times2)}$			
Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 4) C2D(1, 4, 8) GAP *Linear*(8, 4) *Linear*(4, 1)	C2D(2, N_{in}, 4) C2D(2, 4, 4) GAP *Linear*(4, 4) *Linear*(4, 1)	C2D(1, N_{in}, 8) C2D(2, 8, 4) GAP *Linear*(4, 1) -
$N_{\text{param}}^{(U)}$	341	353	273
$N_{\text{param}}^{(U,W)}$	469	481	337
$N_{\text{param}}^{(U,W,W^\dagger)}$	597	609	401
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 8) C2D(2, 8, 8) C2D(2, 8, 8) GAP *Linear*(8, 4) *Linear*(4, 1)	C2D(2, N_{in}, 8) C2D(2, 8, 8) - GAP *Linear*(8, 4) *Linear*(4, 1)	C2D(3, N_{in}, 4) C2D(2, 4, 8) - GAP *Linear*(8, 4) *Linear*(4, 1)
$N_{\text{param}}^{(U)}$	1089	825	757
$N_{\text{param}}^{(U,W)}$	1345	1081	1045
$N_{\text{param}}^{(U,W,W^\dagger)}$	1601	1337	1333

Large	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 16) C2D(2, 16, 16) C2D(2, 16, 16) GAP *Linear*(16, 8) *Linear*(8, 1)	C2D(3, N_{in}, 16) C2D(3, 16, 8) - GAP *Linear*(8, 8) *Linear*(8, 1)	C2D(3, N_{in}, 16) C2D(1, 16, 8) C2D(3, 8, 16) GAP *Linear*(16, 8) *Linear*(8, 1)
$N_{\text{param}}^{(U)}$	3265	3561	3769
$N_{\text{param}}^{(U,W)}$	3777	4713	4921
$N_{\text{param}}^{(U,W,W^\dagger)}$	4289	5865	6073
Wide	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 128) - GAP *Linear*(128, 1) -	C2D(2, N_{in}, 256) C2D(3, 256, 32) GAP *Linear*(32, 1) -	C2D(2, N_{in}, 512) - GAP *Linear*(512, 64) *Linear*(64, 1)
$N_{\text{param}}^{(U)}$	8449	90433	66177
$N_{\text{param}}^{(U,W)}$	12545	98625	82561
$N_{\text{param}}^{(U,W,W^\dagger)}$	16641	106817	98945

Figure: CNN architectures for 1×1 and 1×2 loops in 2D

Baseline CNNs for 2×2 and 4×4 loops

$W^{(2\times2)}$			
Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 2)	C2D(2, 32, 4)
	C2D(2, 4, 4)	C2D(1, 2, 4)	C2D(2, 4, 2)
	GAP	GAP	GAP
	Linear(4, 4)	*Linear*(4, 1)	*Linear*(2, 1)
	Linear(4, 1)	-	
N_{param}	609	275	553
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 8)	C2D(3, 32, 4)
	C2D(2, 4, 8)	C2D(2, 8, 8)	C2D(2, 4, 8)
	C2D(2, 8, 8)	C2D(2, 8, 8)	C2D(3, 8, 8)
	C2D(2, 8, 8)	C2D(2, 8, 8)	C2D(2, 8, 8)
	GAP	GAP	GAP
	Linear(8, 16)	*Linear*(8, 8)	*Linear*(8, 4)
	Linear(16, 1)	*Linear*(8, 1)	*Linear*(4, 1)
N_{param}	1341	1905	2181
Large	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 8)	C2D(2, 32, 8)	C2D(3, 32, 8)
	C2D(2, 8, 16)	C2D(2, 8, 16)	C2D(3, 8, 16)
	C2D(2, 16, 32)	C2D(2, 16, 32)	C2D(3, 16, 32)
	C2D(2, 32, 64)	C2D(2, 32, 64)	C2D(3, 32, 16)
	-	C2D(2, 64, 32)	-
	GAP	GAP	GAP
	Linear(64, 16)	*Linear*(32, 8)	*Linear*(16, 8)
	Linear(16, 1)	*Linear*(8, 1)	*Linear*(8, 1)
N_{param}	12953	20393	12889

$W^{(4\times4)}$			
Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 4)	C2D(2, 32, 4)
	C2D(2, 4, 4)	C2D(1, 4, 8)	C2D(2, 4, 2)
	GAP	GAP	GAP
	Linear(4, 4)	*Linear*(8, 4)	*Linear*(2, 1)
	Linear(4, 1)	*Linear*(4, 1)	-
N_{param}	609	597	553
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(3, 32, 16)	C2D(2, 32, 16)	C2D(3, 32, 8)
	C2D(1, 16, 8)	C2D(2, 16, 24)	C2D(2, 8, 16)
	C2D(3, 8, 16)	C2D(2, 24, 16)	C2D(1, 16, 32)
	-	-	C2D(2, 32, 16)
	-	-	C2D(2, 16, 8)
	GAP	GAP	GAP
	Linear(16, 8)	*Linear*(16, 8)	*Linear*(8, 8)
	Linear(8, 1)	*Linear*(8, 1)	*Linear*(8, 1)
N_{param}	6073	5321	6049
Large	Architecture 1	Architecture 2	Architecture 3
	C2D(3, 32, 16)	C2D(2, 32, 16)	C2D(4, 32, 16)
	C2D(3, 16, 32)	C2D(2, 16, 32)	C2D(4, 16, 32)
	C2D(3, 32, 64)	C2D(2, 32, 64)	C2D(4, 32, 32)
	C2D(3, 64, 32)	C2D(2, 64, 64)	C2D(4, 32, 16)
	-	C2D(2, 64, 32)	-
	-	C2D(2, 32, 16)	-
	GAP	GAP	GAP
	Linear(32, 16)	*Linear*(16, 16)	*Linear*(16, 8)
	Linear(16, 1)	*Linear*(16, 8)	*Linear*(8, 8)
	-	*Linear*(8, 1)	*Linear*(8, 1)
N_{param}	46769	39553	41273

Figure: CNN architectures for 2×2 and 4×4 loops in 2D

L-CNNs for 1×2, 2×2 and 4×4 in 2D

$W^{(1\times2)}$			
	Small	Medium	Large
	L-CBL(2, 1, 2) *Trace* *Linear*(4, 1)	*L-CBL*(3, 1, 4) *Trace* *Linear*(8, 1)	*L-CBL*(4, 1, 8) *Trace* *Linear*(16, 1)
N_{param}	35	117	329
$W^{(2\times2)}$			
	Small	Medium	Large
	L-CBL(2, 1, 2) *L-CBL*(2, 2, 2) *Trace* *Linear*(4, 1)	*L-CBL*(3, 1, 4) *L-CBL*(3, 4, 4) *Trace* *Linear*(8, 1)	*L-CBL*(4, 1, 8) *L-CBL*(4, 8, 8) *Trace* *Linear*(16, 1)
N_{param}	125	1305	13521
$W^{(4\times4)}$			
	Small	Medium	Large
	L-CBL(2, 1, 2) *L-CBL*(2, 2, 2) *L-CBL*(3, 2, 2) *L-CBL*(3, 2, 2) *Trace* *Linear*(4, 1)	*L-CBL*(3, 1, 4) *L-CBL*(3, 4, 4) *L-CBL*(4, 4, 4) *L-CBL*(4, 4, 4) *Trace* *Linear*(8, 1)	*L-CBL*(4, 1, 8) *L-CBL*(4, 8, 8) *L-CBL*(4, 8, 8) *L-CBL*(4, 8, 8) *Trace* *Linear*(16, 1)
N_{param}	465	4833	39905

Figure: L-CNN architectures for Wilson loops in 2D

L-CNNs for 2×2, 4×4 and Q_{plaq} in 4D

$W^{(2\times2)}$		
	Small	Medium
	L-CBL(2, 6, 2)	*L-CBL*(3, 6, 4)
	L-CBL(2, 2, 2)	*L-CBL*(3, 4, 4)
	Trace	*Trace*
	Linear(4, 1)	*Linear*(8, 1)
N_{param}	1801	8305
$W^{(4\times4)}$		
	Small	Medium
	L-CBL(2, 6, 2)	*L-CBL*(3, 6, 4)
	L-CBL(2, 2, 2)	*L-CBL*(3, 4, 4)
	L-CBL(3, 2, 2)	*L-CBL*(4, 4, 4)
	L-CBL(3, 2, 2)	*L-CBL*(4, 4, 4)
	Trace	*Trace*
	Linear(4, 1)	*Linear*(8, 1)
N_{param}	2109	14377
q^{plaq}		
	Small	
	L-CBL(2, 6, 4)	
	Trace	
	Linear(8, 1)	
N_{param}	3181	

Figure: L-CNN architectures for Wilson loops in 4D