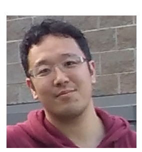
https://indico.bnl.gov/event/13576/

Gauge covariant neural network and full QCD simulation

Based on arXiv:2103.11965 + α

Towards to applicate on chiral quarks



Me: Akio Tomiya (IPUT Osaka, Assistant professor)
akio_AT_yukawa.kyoto-u.ac.jp

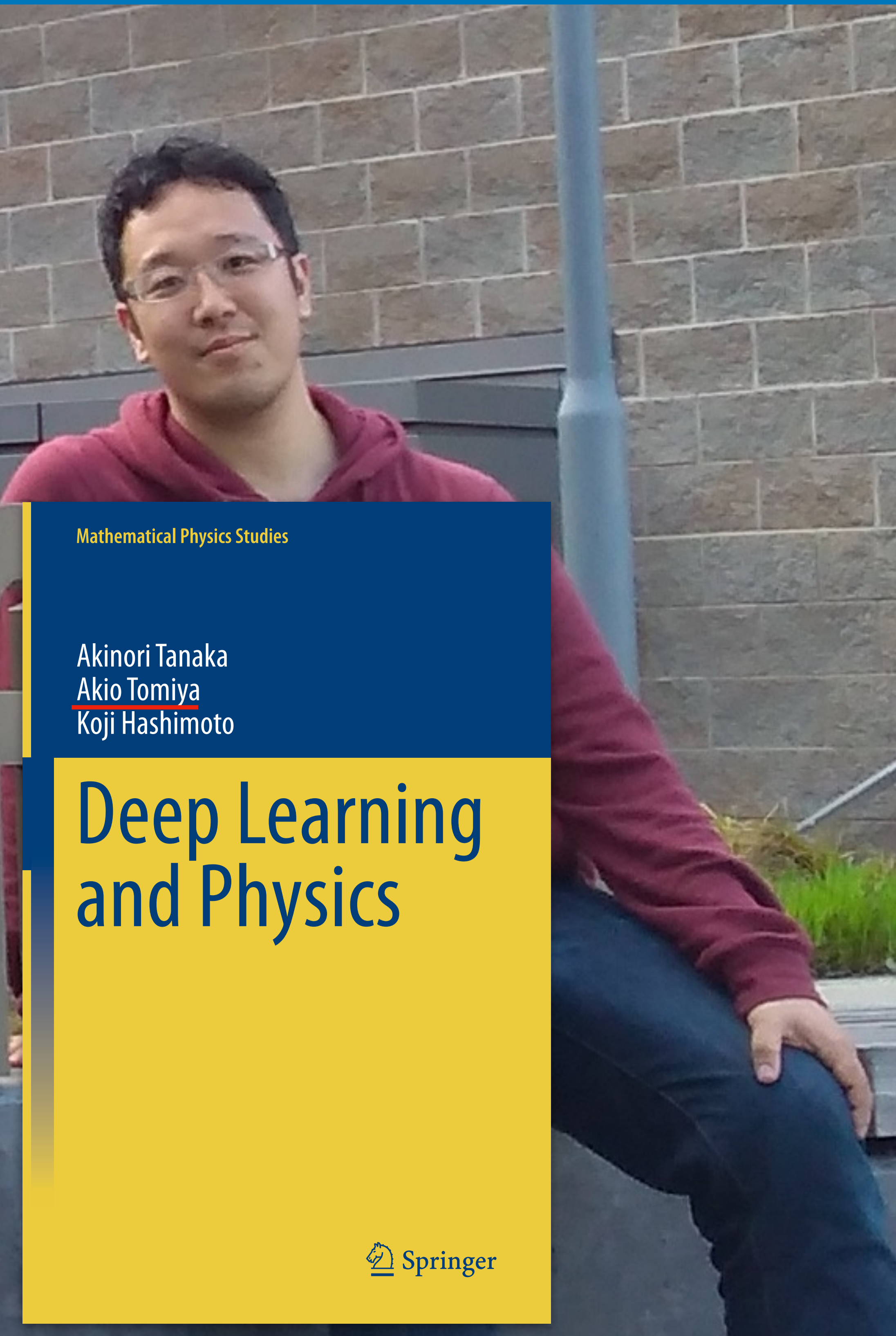
Yuki Nagai (JAEA, Senior researcher)

13/Dec/2021

At "BNL-HET & RBRC Joint Workshop "DWQ@25": The event marks the passage of twenty-five years since the first numerical simulations with Domain Wall Quarks (DWQ)"

Self-introduction

Lattice QCD & Machine learning, IPUT Osaka



What am I?

I am a particle physicist, working on lattice QCD.
I want to apply machine learning on it.

My papers

Detection of phase transition via convolutional neural networks
A Tanaka, A Tomiya
Journal of the Physical Society of Japan 86 (6), 063001

Phase transition detection with NN

Evidence of effective axial $U(1)$ symmetry restoration at high temperature QCD
A Tomiya, G Cossu, S Aoki, H Fukaya, S Hashimoto, T Kaneko, J Noaki, ...
Physical Review D 96 (3), 034509

Axial anomaly at T>0 with Domain-wall fermions

Digital quantum simulation of the schwinger model with topological term via adiabatic state preparation
B Chakraborty, M Honda, T Izubuchi, Y Kikuchi, A Tomiya
arXiv preprint arXiv:2001.00485

Schwinger model on quantum computer

Biography

2010 : University of Hyogo
2015 : PhD in Osaka university (Japan)
2015 - 2018 : Postdoc in Wuhan (China)
2018 - 2021 : SPDR in Riken/BNL (New York, US)
2021 - : Intrl. Professional Univ. of Tech. in Osaka as a faculty

Outline

1. Background motivation (Machine learning for QCD)
2. Neural networks
    1. Neural network(NN) = Signal processing/Filtering with tuning
    2. Convolutional NN = Translational equivariance
3. Smearing $\sim$ Neural network
4. Gauge Covariant NN = **trainable** smearing, training for SU(N) fields
    1. Neural ODE for Covariant NN = Gradient flow with trainable parameters
5. Application: Self-learning HMC for staggered, and domain-wall fermions

1. Background motivation (Machine learning for QCD)

Machine learning for QCD

Akio Tomiya

Lattice path integral > 1000 dim, Trapezoidal int is impossible

K. Wilson 1974

$$S = \int d^4x \left[+ \frac{1}{2} \mathrm{tr}\, F_{\mu\nu} F_{\mu\nu} + \bar{\psi}\left(\partial\!\!\!/ - \mathrm{i}g A\!\!\!/ + m \right) \psi \right]$$

Lattice regulation

$U_\mu = \mathrm{e}^{aigA_\mu}$

$$S[U, \psi, \bar{\psi}] = a^4 \sum_n \left[-\frac{1}{g^2} \mathrm{Re\ tr}\, U_{\mu\nu} + \bar{\psi}\left(D\!\!\!/ + m \right) \psi \right]$$

a is lattice spacing(cutoff = a^{-1})

Both gives same expectation value (for long range)

(They are same except for infinitely Irrelevant operators)

$$\mathrm{Re}\ U_{\mu\nu} \sim \frac{-1}{2} g^2 a^4 F_{\mu\nu}^2 + O(a^6)$$

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}U \mathcal{D}\bar{\psi} \mathcal{D}\psi e^{-S} \mathcal{O}(U) = \frac{1}{Z} \int \mathcal{D}U e^{-S_{\mathrm{gauge}}[U]} \det(D + m) \mathcal{O}(U)$$

$$= \frac{1}{Z} \int \mathcal{D}U e^{-S_{\mathrm{eff}}[U]} \mathcal{O}(U)$$

$$= \prod_{n \in \{\mathbb{Z}/L\}^4} \prod_{\mu=1}^{4} dU_\mu(n)$$

>1000 dim. We cannot use Newton–Cotes type integral like Trapezoid, Simpson etc.

We cannot control numerical error

Machine learning for QCD

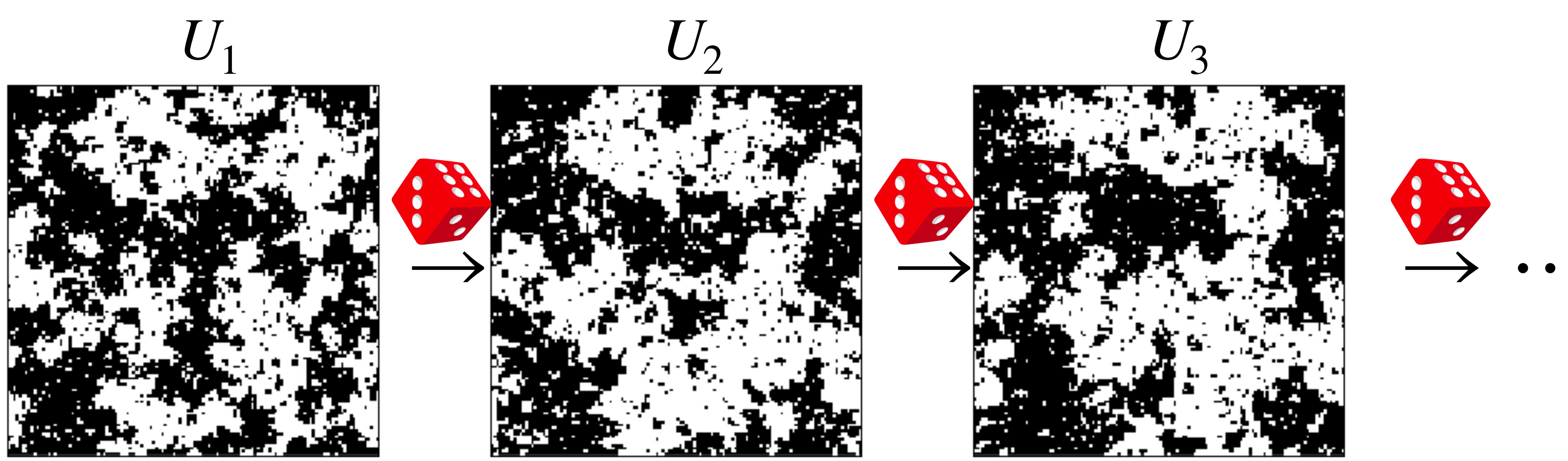
Akio Tomiya

Monte-Carlo integration is available

M. Creutz 1980

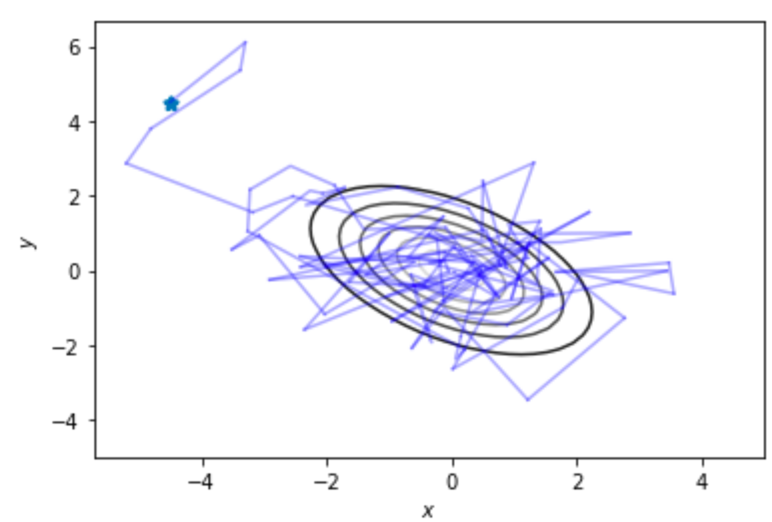
$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}U e^{-S_{\text{eff}}[U]} \mathcal{O}(U) \qquad S_{\text{eff}}[U] = S_{\text{gauge}}[U] - \log\det(\not{D}[U] + m)$$

Monte-Carlo: Generate field configurations with "$P[U] = \frac{1}{Z} e^{-S_{\text{eff}}[U]}$". It gives expectation value



HMC: Hybrid (Hamiltonian) Monte-Carlo
De-facto standard algorithm

$$S(x, y) = \frac{1}{2}(x^2 + y^2 + xy)$$



Machine learning for QCD

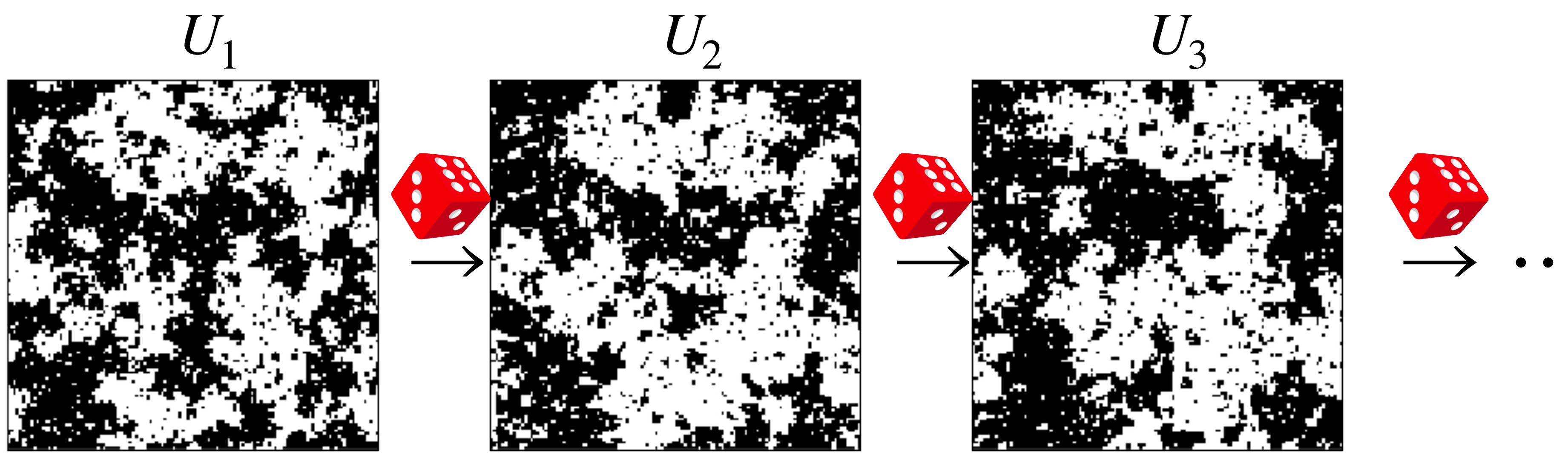
Monte-Carlo integration is available

Akio Tomiya

M. Creutz 1980

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}U e^{-S_{\rm eff}[U]} \mathcal{O}(U) \qquad S_{\rm eff}[U] = S_{\rm gauge}[U] - \log\det(\not{D}[U] + m)$$

Monte-Carlo: Generate field configurations with "$P[U] = \frac{1}{Z} e^{-S_{\rm eff}[U]}$". It gives expectation value



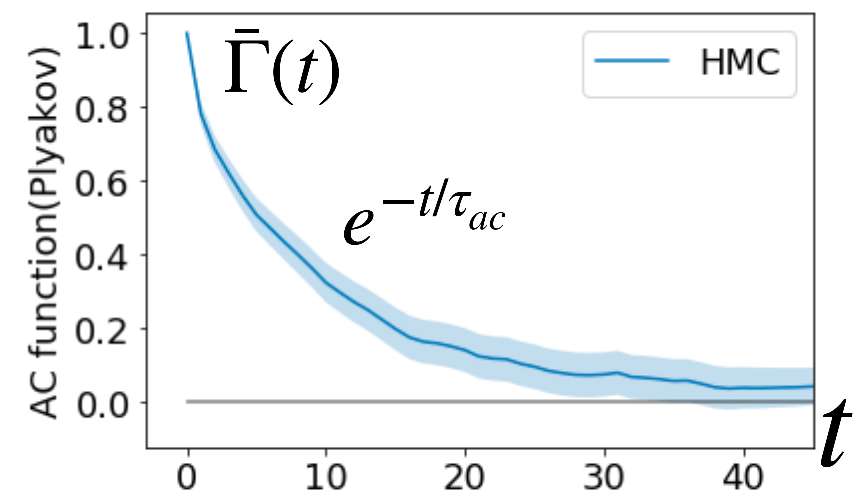
Error of integration is determined by the number of sampling

$$\langle \mathcal{O} \rangle = \frac{1}{N_{\rm sample}} \sum_{k}^{N_{\rm sample}} \mathcal{O}[U_k] \pm O\left(\frac{1}{\sqrt{N_{\rm sample}}}\right)$$

Autocorrelation& Critical slowing down

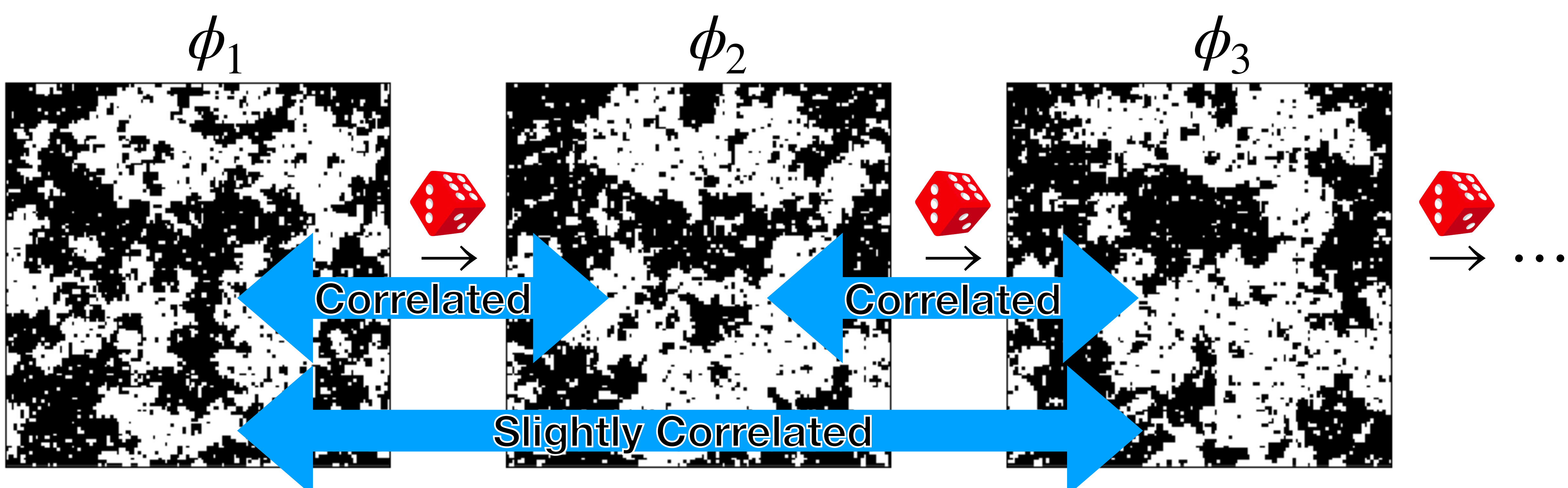
Correlation between samples = inefficiency of calculation

$$\langle O[\phi] \rangle = \frac{1}{N} \sum_{k}^{N} O[\phi_k] \ \pm \ O\left(\frac{1}{\sqrt{N_{\text{indep}}}}\right)$$



$$N_{\text{indep}} = \frac{N_{\text{sample}}}{2\tau_{ac}}$$

$$\bar{\Gamma}(t) = \frac{1}{N-t} \sum_{k} (O[\phi_{k+t}] - \bar{O})(O[\phi_k] - \bar{O}) \sim e^{-t/\tau_{ac}}$$



Large τ_{ac} means, such simulation is inefficient

Autocorrelation& Critical slowing down

Long autocorrelation around the critical temperature

Data from arXiv:2006.13422
Nf=3, **dynamical staggered** with magnetic field

$$L^3 \times N_t = 16^3 \times 4$$

$$ma = 0.03$$

β	N_{sample}	τ_{ac}	N_{indep}
5.166	15,000	47	160
5.167	20,000	224	45
5.168	20,000	656	15
5.169	**20,000**	2940	3
5.170	15,000	1306	6
5.171	14,000	58	116
5.172	10,000	48	106

$$N_{\text{indep}} = \frac{N_{\text{sample}}}{2\tau_{ac}}$$

Critical temp.

$$\langle O[\phi] \rangle = \frac{1}{N_{\text{sample}}} \sum_{k}^{N_{\text{sample}}} O[\phi_k] \pm O\left(\frac{1}{\sqrt{N_{\text{indep}}}}\right)$$

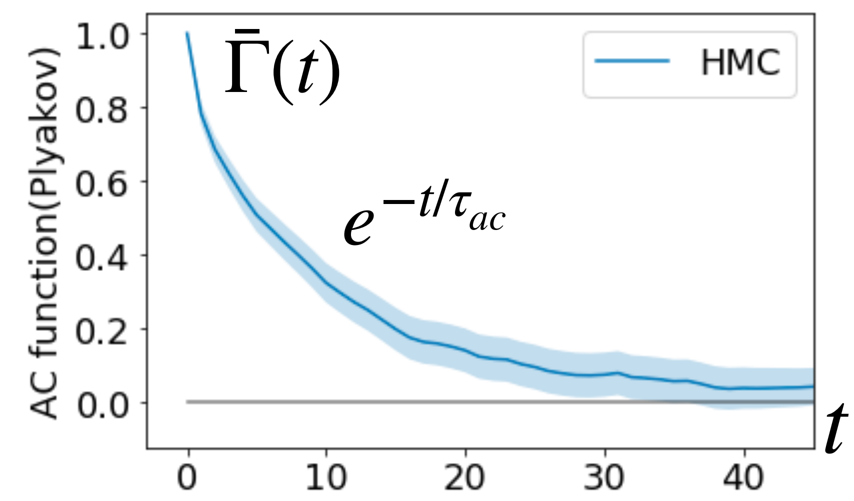
$$\tau_{ac} \sim \xi^z \sim L^z$$

z : Dynamic critical exponent (see 1703.03136)
τ_{ac}: **Algorithm dependent** (N. Madras et. al 1988)

Autocorrelation& Critical slowing down

Summary for now: long autocorrelation = inefficiency

$$\langle O[\phi] \rangle = \frac{1}{N} \sum_{k}^{N} O[\phi_k] \pm O\left(\frac{1}{\sqrt{N_{\rm indep}}}\right)$$



$$N_{\rm indep} = \frac{N_{\rm sample}}{2\tau_{ac}}$$

$$\bar{\Gamma}(t) = \frac{1}{N-t} \sum_{k} (O[\phi_{k+t}] - \bar{O})(O[\phi_k] - \bar{O}) \sim e^{-t/\tau_{ac}}$$

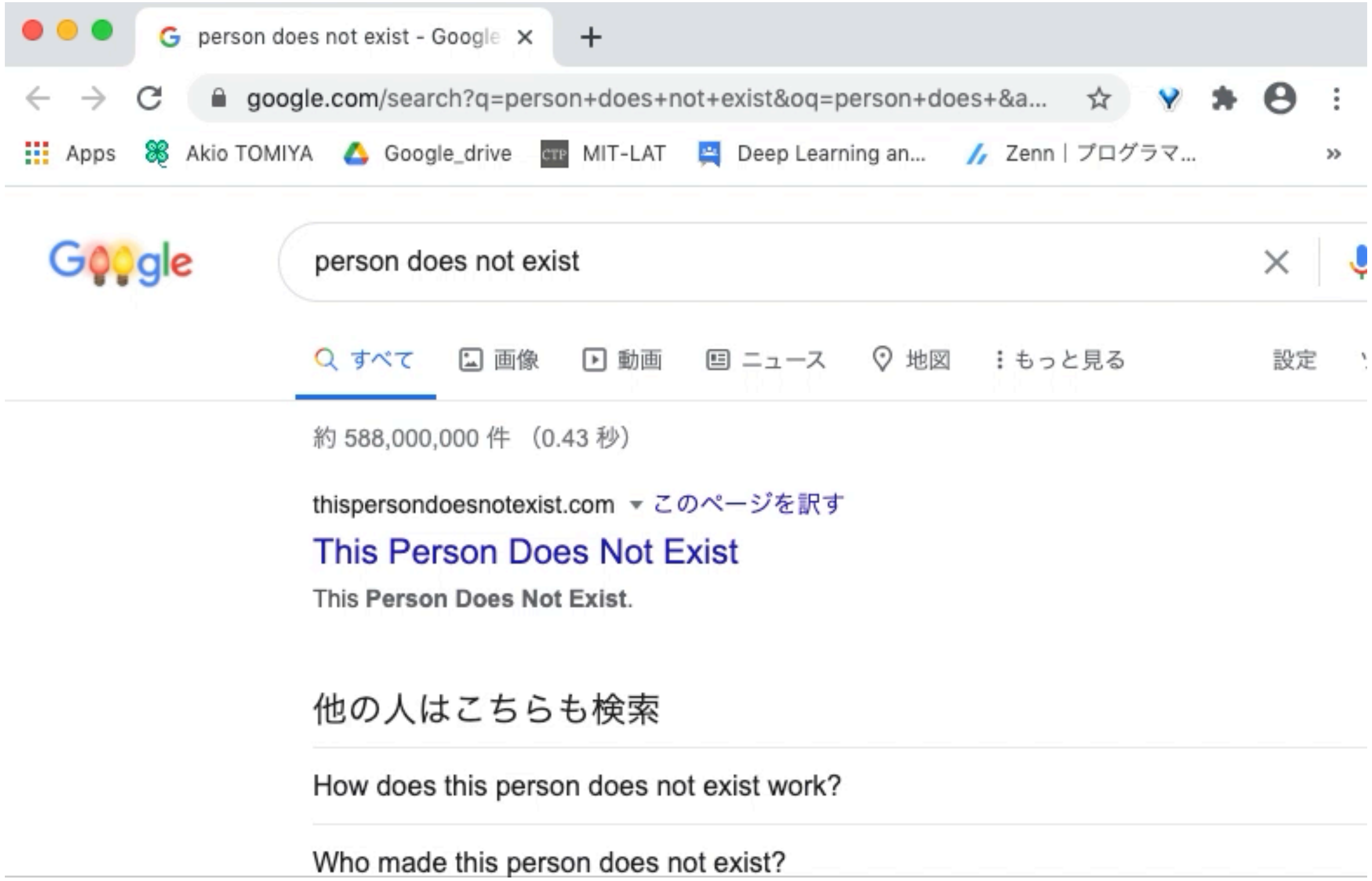
τ_{ac} is given by an update algorithm (N. Madras et. al 1988)

- Autocorrelation time τ_{ac} quantifies similarity between samples
- τ_{ac} is algorithm dependent quantity
- If τ_{ac} becomes half, we can get doubly precise results in the same time cost

Can we make this mild using machine learning?

Machine learning for QCD

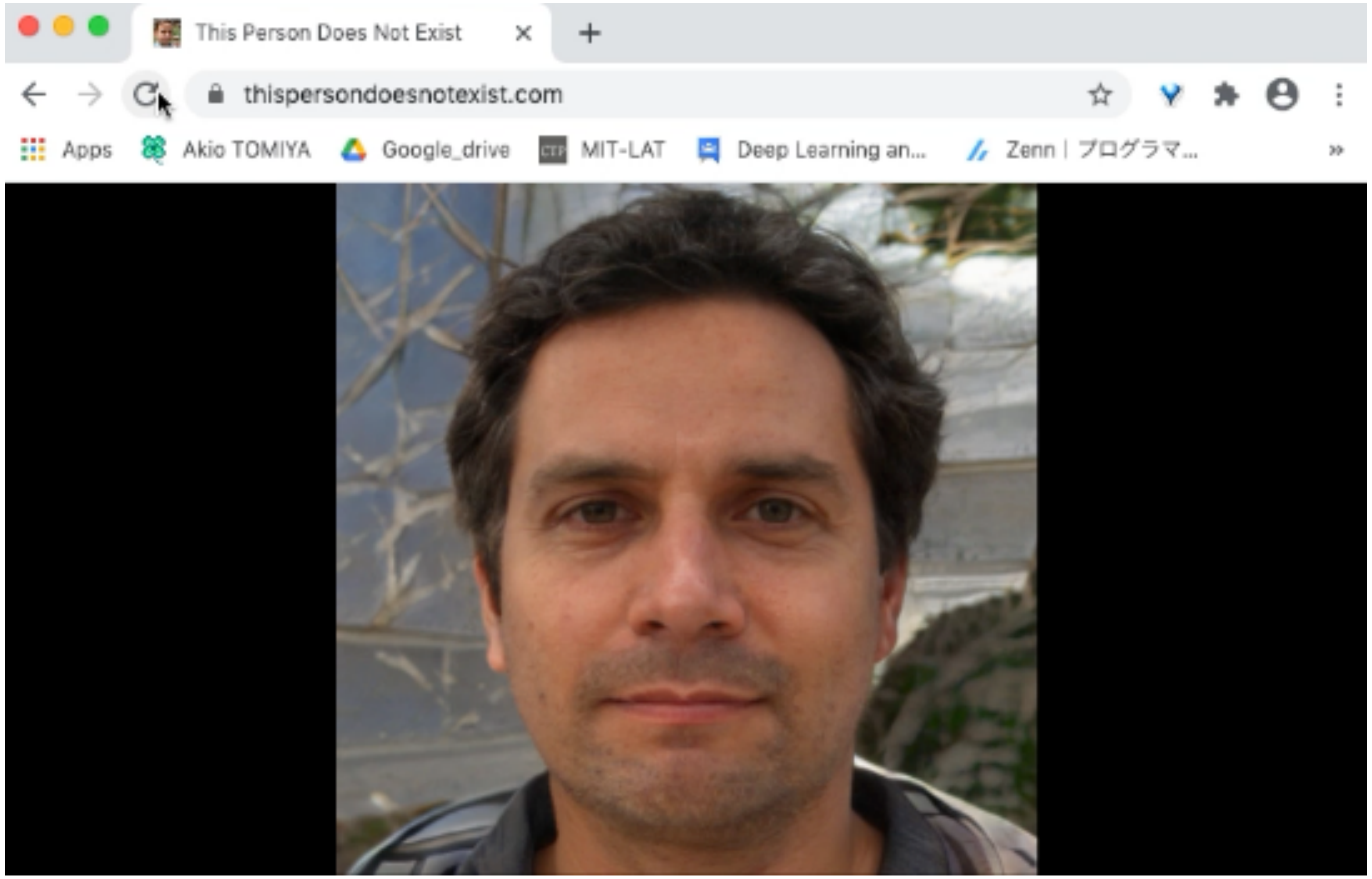
Neural net can make human face images



Machine learning for QCD

Neural net can make human face images

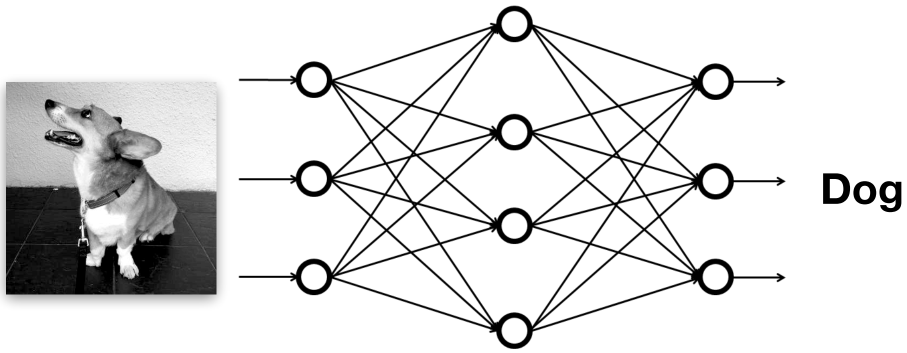
Neural nets can generate realistic human faces (Style GAN2)



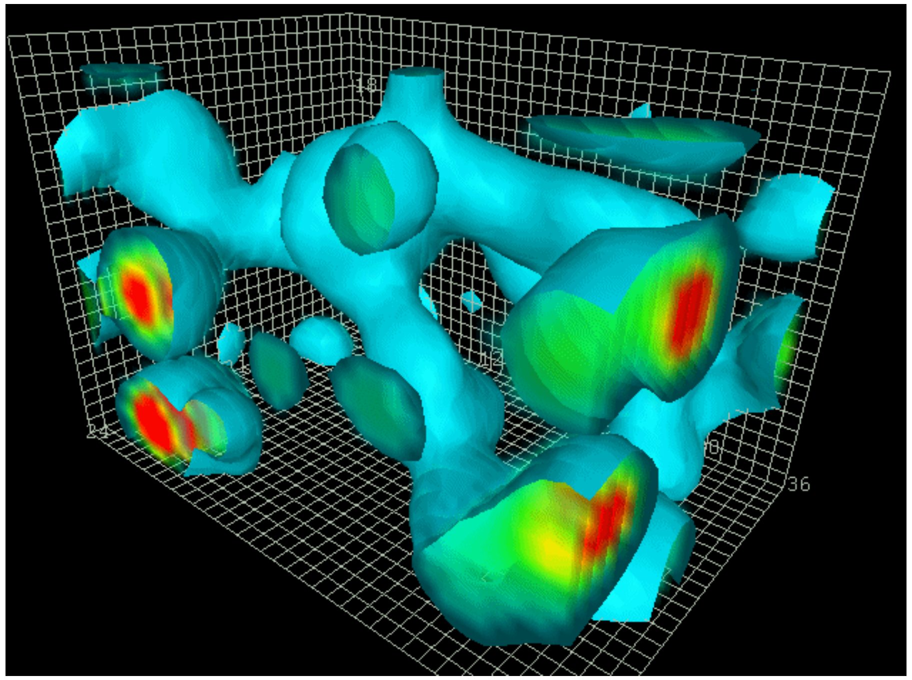
Realistic Images can be generated by machine learning!
Configurations as well? (configuration ~ images?)

ML for LQCD is needed

- Machine learning/ Neural networks
  - data processing techniques for 2d/3d data in the real world (pictures)
  - (Variational) Approximation ($\sim$ fitting)



- Lattice QCD
  - 4 dimension
  - Non-abelian gauge symmetry
  - Fermions
  - Exactness is necessary
- How can we deal with?



http://www.physics.adelaide.edu.au/theory/staff/leinweber/VisualQCD/QCDvacuum/

ML + Configuration generations

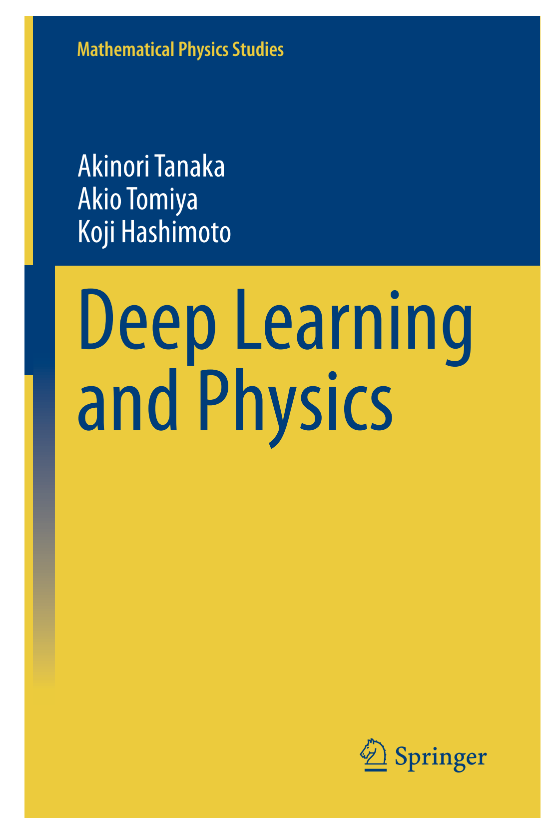
Akio Tomiya

Configuration generation with machine learning is developing

Year	Group	ML	Dim.	Theory	Gauge sym	Exact?	Fermion?	Lattice2021/ref
2017	**AT+**	RBM + HMC	2d	Scalar	-	No	No	arXiv: 1712.03893
2018	K. Zhou+	GAN	2d	Scalar	-	No	No	arXiv: 1810.12879
2018	J. Pawlowski +	GAN +HMC	2d	Scalar	-	Yes?	No	arXiv: 1811.03533
2019	MIT+	Flow	2d	Scalar	-	Yes	No	arXiv: 1904.12072
2020	MIT+	Flow	2d	U(1)	Equivariant	Yes	No	arXiv: 2003.06413
2020	MIT+	Flow	2d	SU(N)	Equivariant	Yes	No	arXiv: 2008.05456
2020	**AT+**	SLMC	**4d**	SU(N)	Invariant	Yes	No	arXiv: 2010.11900
2021	M. Medvidović+	A-NICE	2d	Scalar	-	No	No	arXiv: 2012.01442
2021	S. Foreman	L2HMC	2d	U(1)	Yes	Yes	No	
2021	**AT+**	SLHMC	**4d**	QCD	Covariant	Yes	YES!	This talk
2021	L. Del Debbio+	Flow	2d	Scalar, O(N)	-	Yes	No	
2021	MIT+	Flow	2d	Yukawa	-	Yes	Yes	
2021	**S. Foreman, AT+**	Flowed HMC	2d	U(1)	Equivariant	Yes	No but compatible	arXiv: 2112.01586
2021	XY Jing	Neural net	2d	U(1)	?	Yes?	No	

2. Neural networks

1. Background motivation (Machine learning for QCD)
2. Neural networks
    1. Neural network(NN) = Signal processing/Filtering with tuning
    2. Convolutional NN = Translational equivariance
3. Smearing ～ Neural network
4. Gauge Cov NN = **trainable** smearing, training for SU(N) fields
    1. Neural ODE for Cov NN = Gradient flow with trainable parameters
5. Application: Self-learning HMC for staggered, and domain-wall fermions



What is the neural networks?

Affine transformation + element-wise transformation

Component of neural net

$$u_i(x_j) = \begin{cases} z_i^{(l)} = \sum_j w_{ij}^{(l)} x_j + b_i^{(l)} \\ u_i = \sigma^{(l)}(z_i^{(l)}) \end{cases}$$

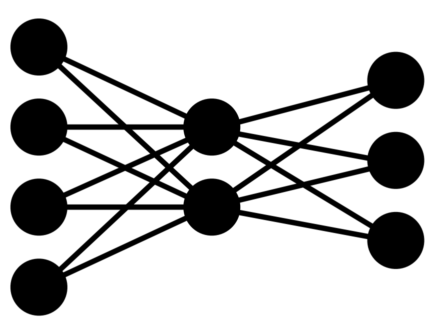
Matrix product
vector addition
(b=0 called linear transf.)

element-wise (local)
Non-linear transf.
Typically ~ tanh shape

Fully connected neural networks

$$f_\theta(\vec{x}) = \sigma^{(l=2)}(W^{(l=2)}\sigma^{(l=1)}(W^{(l=1)}\vec{x} + \vec{b}^{(l=1)}) + \vec{b}^{(l=2)})$$

θ **represents a set of parameters: eg** $w_{ij}^{(l)}, b_i^{(l)}, \cdots$ (throughout this talk!)



Neural network = (Variational) map between vector to vector

What is the neural networks?

Neural network is a universal approximator of functions

Image classification, cats and dogs

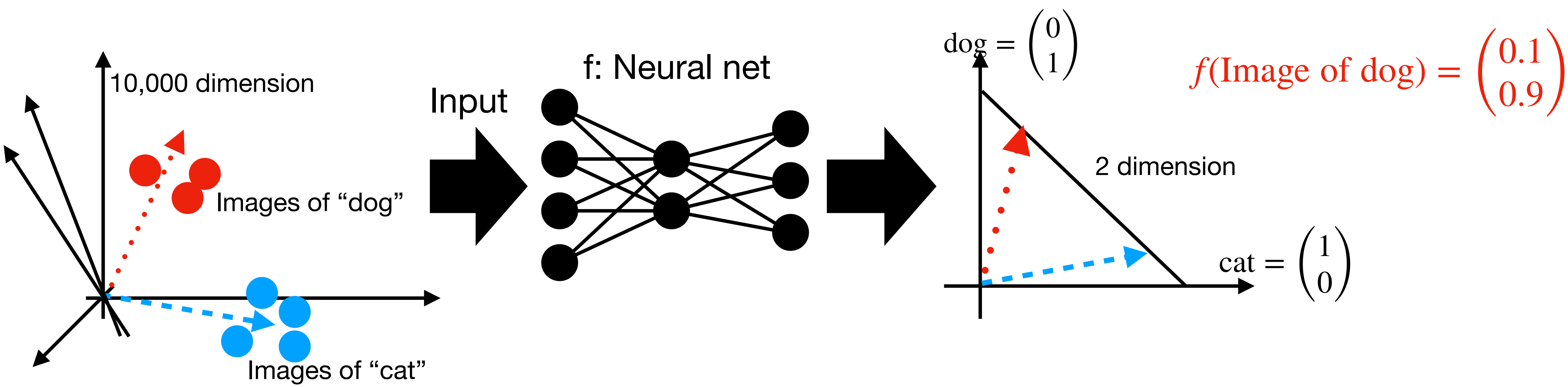
100x100

$$\xRightarrow{\text{Flatten}} \begin{pmatrix} 0.000 \\ 0.000 \\ 0.8434 \\ 0.756 \\ 0.3456 \\ \vdots \end{pmatrix}$$

Image is a vector
(this is 10,000 dimension)

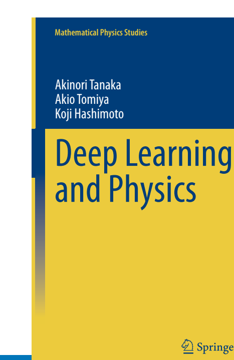
$$\text{dog} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Label is 2 dim vector
(cat = $(1, 0)^t$)



<u>**Fact: neural network can mimic any function! (universal app. thm)**</u>

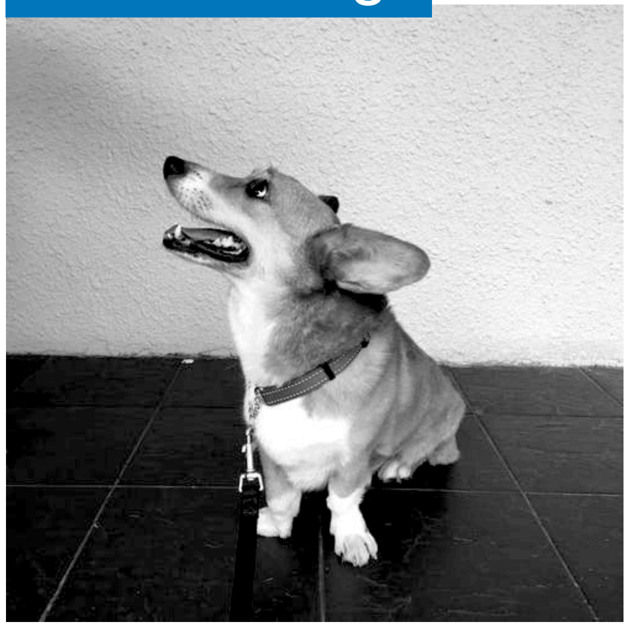
In this example, neural net mimics a map between
image (10,000-dim vector) and label (2-dim vector)



What is the neural networks?

Convolution layer = trainable filter

Filter on image



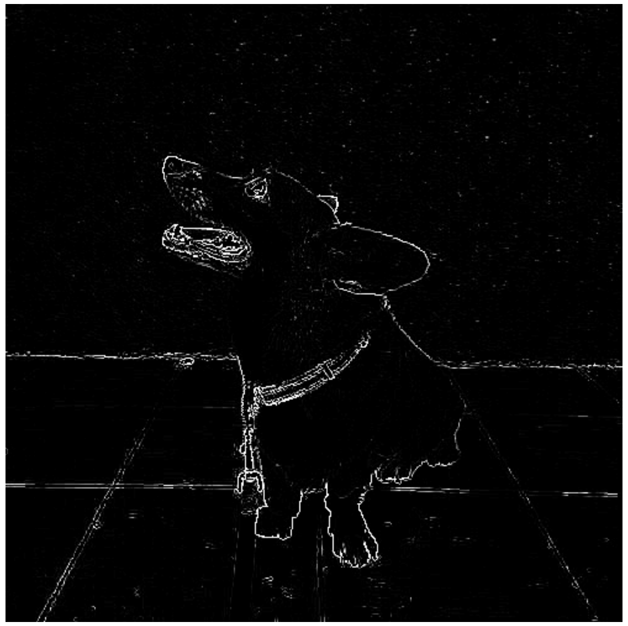
$*$

Laplacian filter

0	1	0
1	-2	1
0	1	0

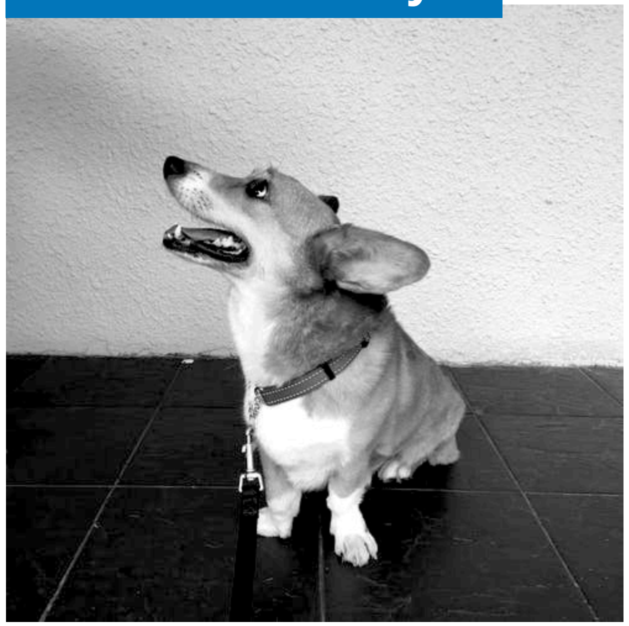
(Discretization of ∂^2)

$=$



Edge detection

Convolution layer



Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

$*$

Trainable filter

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

$=$

Edge detection

Smoothing
(Gaussian filter)

…

Gaussian filter

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

(Training and data determines what kind of filter is realized)
Extract features

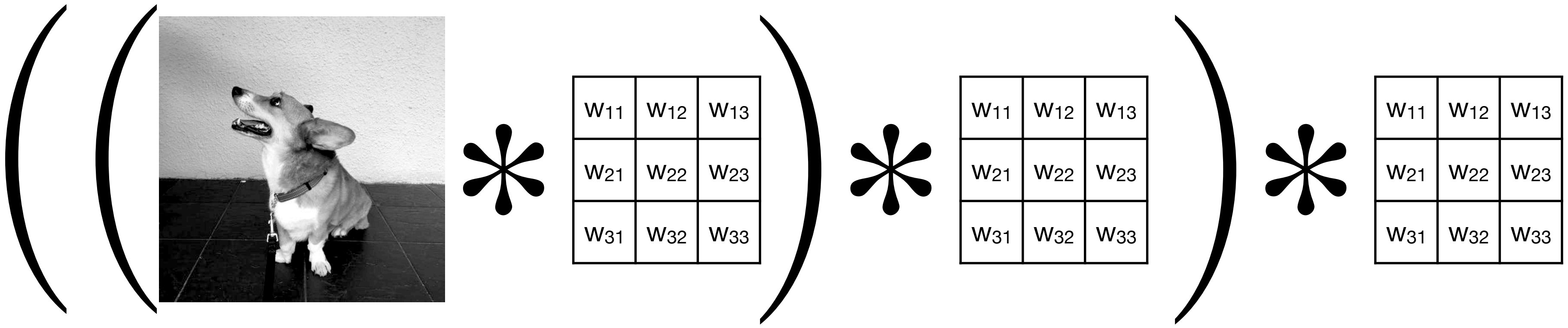
What is the neural networks?

Akio Tomiya

Convolution layers can be nested as well as fully connected

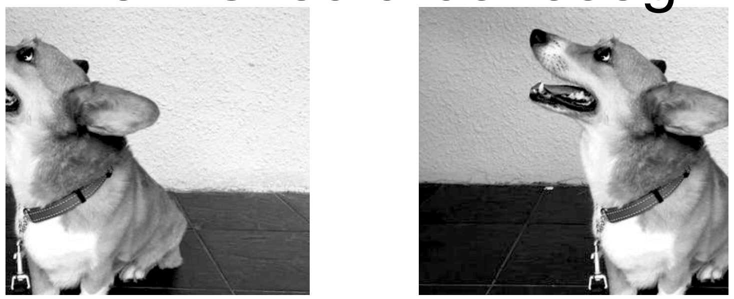
We can make a composite function with the convolutional layers

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!



1. The convolution layers are inspired from visual cortes in brains
2. Filtering operation does not care the absolute coordinate = translation symmetry

In neural net: Both should be recognized as “dog”



What is the neural networks?

Convolution layers can be nested as well as fully connected

We can make a composite function with the convolutional layers

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

$\Bigg(\Bigg($

$$* \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \Bigg) * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \Bigg) * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

1. The convolution layers are inspired from visual cortes in brains
2. Filtering operation does not care the absolute coordinate = translation symmetry

In neural net: Both should be recognized as “dog”

Modern viewpoint: (T. Cohen+, group equivariant neural network, 2016~)

1. “Convolution” is an concrete example of equivariant layer
   Translational equivariant = if input is shifted to the right, output shifted to the right.
   Translational equivariariance helps to make invariant neural net/loss function
2. e.g. For rotational symmetric data -> neural net for it should respect rotational symmetry!
   Spherical convolution (T. Cohen+) realizes an approximator which guarantees to have equivariance

3. Smearing
= (covariant) Neural network with fixed parameters

1. Background motivation (Machine learning for QCD)
2. Neural networks
    1. Neural network(NN) = Signal processing/Filtering with tuning
    2. Convolutional NN = Translational equivariance
3. Smearing ~ Neural network
4. Gauge Cov NN = **trainable** smearing, training for SU(N) fields
    1. Neural ODE for Cov NN = Gradient flow with trainable parameters
5. Application: Self-learning HMC for staggered, and domain-wall fermions

Smearing

Smoothing improves global properties

Eg.

Coarse image

Numerical derivative is unstable

Gaussian filter

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 1 \\ \hline 1 & 2 & 1 \\ \hline\end{array}$$

Smoothened image

Numerical derivative is stable

We want to smoothen gauge configuration with keeping gauge symmetry

Two types:

APE-type smearing

Stout-type smearing

M. Albanese+ 1987
R. Hoffmann+ 2007
C. Morningster+ 2003

Smearing

Smoothing with gauge symmetry, APE type

Akio Tomiya

M. Albanese+ 1987
R. Hoffmann+ 2007

APE-type smearing

$$U_\mu(n) \to U_\mu^{\text{fat}}(n) = \mathcal{N}\left[(1-\alpha)U_\mu(n) + \frac{\alpha}{6}V_\mu^\dagger[U](n)\right]$$

Normalization

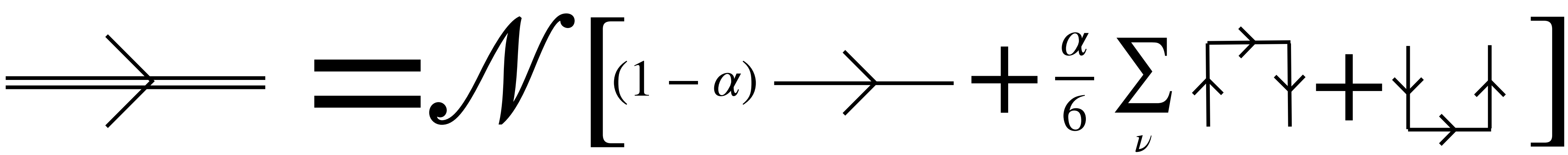
$$\mathcal{N}[M] = \frac{M}{\sqrt{M^\dagger M}}$$

Or projection

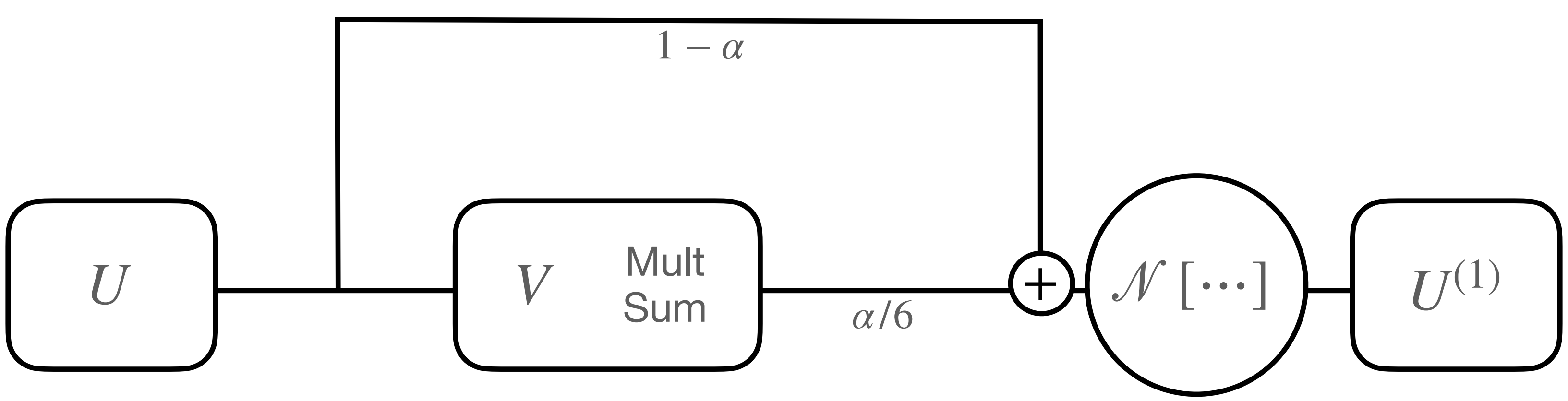
$$V_\mu^\dagger[U](n) = \sum_{\mu\neq\nu} U_\nu(n)U_\mu(n+\hat{\nu})U_\nu^\dagger(n+\hat{\mu}) + \cdots$$

$V_\mu^\dagger[U](n)$& $U_\mu(n)$ shows same transformation
→$U_\mu^{\text{fat}}[U](n)$ is as well

Schematically,



In the calculation graph,



Smearing

Smoothing with gauge symmetry, stout type

C. Morningster+ 2003

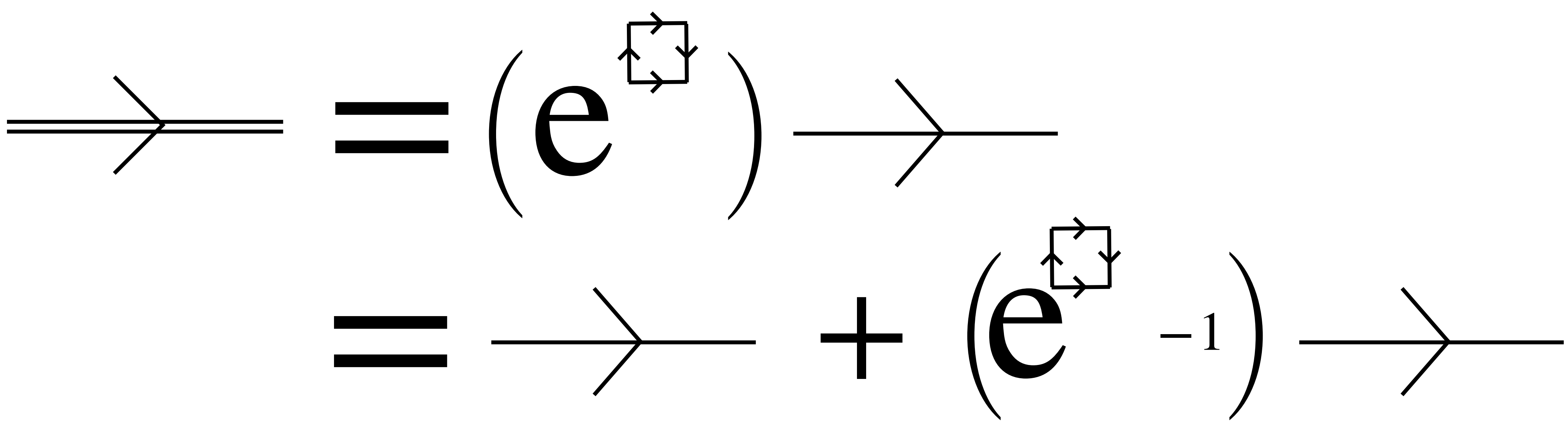
Stout-type smearing

$$U_\mu(n) \to U_\mu^{\mathrm{fat}}(n) = \mathrm{e}^{Q} U_\mu(n)$$
$$= U_\mu(n) + (\mathrm{e}^{Q} - 1) U_\mu(n)$$

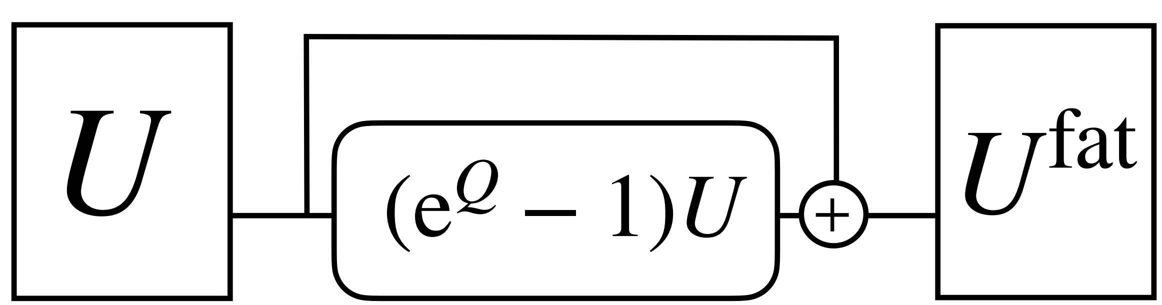
Q: anti-hermitian traceless plaquette

This is less obvious but this actually obeys same transformation

Schematically,



In the calculation graph,



Smearing

Smearing decomposes into two parts

General form of smearing

$$U_\mu^{\text{fat}}(n) = \begin{cases} z_\mu(n) = w_1 U_\mu(n) + w_2 \mathcal{G}[U] & \text{Summation with gauge sym} \\ U_\mu^{\text{fat}}(n) = \mathcal{N}(z_\mu(n)) & \text{A local function} \end{cases}$$

Smearing

Smearing ~ neural network with fixed parameter!

AT Y. Nagai arXiv: 2103.11965

General form of smearing

$$U_\mu^{\rm fat}(n) = \begin{cases} z_\mu(n) = w_1 U_\mu(n) + w_2 \mathcal{G}[U] \\ U_\mu^{\rm fat}(n) = \mathcal{N}(z_\mu(n)) \end{cases}$$

Summation with gauge sym

A local function

It has similar structure with neural networks,

$$u_i(x_j) = \begin{cases} z_i^{(l)} = \sum_j w_{ij}^{(l)} x_j + b_i^{(l)} \\ u_i = \sigma^{(l)}(z_i^{(l)}) \end{cases}$$

Affine transformation

element-wise (local)

(Index i in the neural net corresponds to n & μ in smearing. Information processing with NN is evolution of scalar field)

Multi-level smearing = Deep learning (with given parameters)

As same as the convolution, we can train weights (How?)

4. Gauge Covariant Neural networks = trainable smearing, training for SU(N) fields

Gauge covariant neural network

= trainable smearing

Akio Tomiya

AT Y. Nagai arXiv: 2103.11965

Gauge covariant neural network = general smearing with trainable parameters w

$$U_\mu^{(l+1)}(n)\left[U^{(l)}\right] = \begin{cases} z_\mu^{(l+1)}(n) = w_1^{(l)} U_\mu^{(l)}(n) + w_2^{(l)} \mathcal{G}_{\bar\theta}^{(l)}[U] \\ \mathcal{N}(z_\mu^{(l+1)}(n)) \end{cases}$$

(Weight "w" can be depend on n and μ = fully connected like. Less symmetric, more parameters)

e.g.
$$U_\mu^{\mathrm{NN}}(n)[U] = U_\mu^{(3)}(n)\left[U_\mu^{(2)}(n)\left[U_\mu^{(1)}(n)\left[U_\mu(n)\right]\right]\right]$$

Good properties: Obvious gauge symmetry. Translation, rotational symmetries.

(Analogous to convolutional layer, this fully uses information of the symmetries)

$$U_\mu(n) \mapsto U_\mu^{\mathrm{NN}}(n) = U_\mu^{\mathrm{NN}}(n)[U]$$

1. Gauge covariant composite function:
   Input = gauge field, Output = gauge field

2. Parameters in the network can be trainable using a ML technique.

Gauge covariant neural network

Akio Tomiya

Training can be done with (extended) back propagation

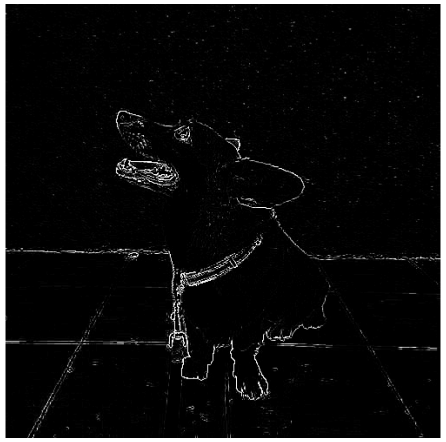
AT Y. Nagai arXiv: 2103.11965

Gauge inv. loss function can be constructed by gauge invariant actions

Usual neural network

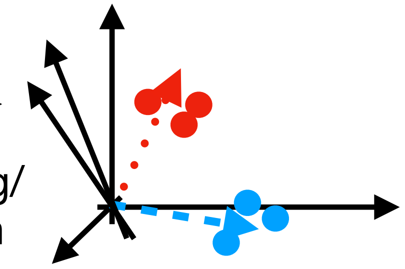
w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Translation *equivariant* map with trainable parameters

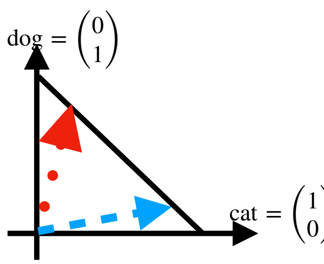


Translation *equivariant* = the image is shifted, output image is also shifted

G.A. Pooling/ flatten



Dense net

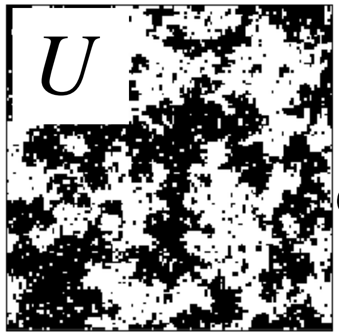


feed

Invariant loss function

L

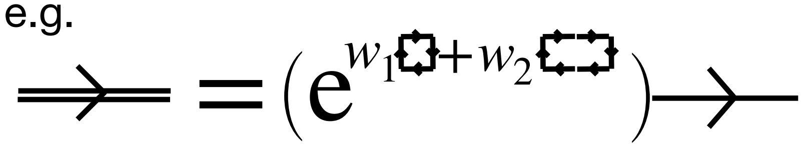
Covariant neural networks



$U_\mu^{\mathrm{NN}}(n)[U]$

Gauge covariant map with trainable parameters

e.g.



Feed to Dirac op

Covariant

$D\left[U_\mu^{\mathrm{NN}}(n)[U]\right] \longrightarrow S_{\mathrm{ferm}}\left[U^{\mathrm{NN}}\right] \xrightarrow{\text{feed}} L$

Parametrized Dirac operator

Parametrized action

Invariant loss function

Construct loops

Covariant

$W\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$

Parametrized loop operators (e.g. plaquette, Polyakov loop)

$S_{\mathrm{plaq}}\left[U^{\mathrm{NN}}\right] \xrightarrow{\text{feed}} L$

$Q\left[U^{\mathrm{NN}}\right] \longrightarrow L$

Topological charge

cf. Gauge equivariant neural net (M Favoni+)

Gauge covariant neural network

Training can be done with (extended) back propagation

AT Y. Nagai arXiv: 2103.11965

Gauge inv. loss function can be constructed by gauge invariant actions

$$S^{\mathrm{NN}}[U] = S\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

S: gauge action or fermion action

Loss function $$L_\theta[U] = f\left(S^{\mathrm{NN}}[U]\right)$$

f: mean-square for example, mini-batch

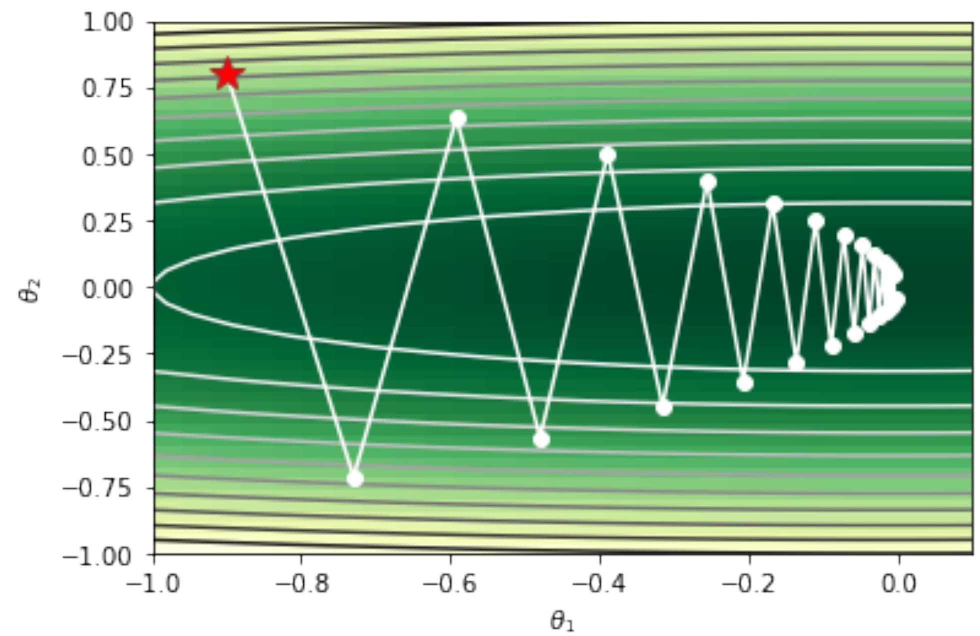
(c.f. Behler-Parrinello type neural net)

Training: We can use “gradient descent” (also “Adam” (adaptive-momentum) is applicable)

Repeat update (until converge) $$\theta^{(l)} \leftarrow \theta^{(l)} - \eta \frac{\partial L_\theta[U]}{\partial \theta^{(l)}}$$

$\theta^{(l)}$ is parameters in l-th layer

Example of Gradient descent



Gauge covariant neural network

Akio Tomiya

Training can be done with (extended) back propagation

AT Y. Nagai arXiv: 2103.11965

Gauge inv. loss function can be constructed by gauge invariant actions

$$S^{\mathrm{NN}}[U] = S\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

S: gauge action or fermion action

Loss function $$L_\theta[U] = f\left(S^{\mathrm{NN}}[U]\right)$$

f: mean-square for example, mini-batch

(c.f. Behler-Parrinello type neural net)

Training: We can use "gradient descent" (also "Adam" (adaptive-momentum) is applicable)

Repeat update (until converge) $$\theta^{(l)} \leftarrow \theta^{(l)} - \eta \frac{\partial L_\theta[U]}{\partial \theta^{(l)}}$$

$\theta^{(l)}$ is parameters in l-th layer

The second term requires the chain rule for matrix fields, we developed extended delta rule:

$$\frac{\partial L_\theta[U]}{\partial \theta^{(l)}} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial S^{\mathrm{NN}}} \frac{\partial S^{\mathrm{NN}}}{\partial U^{(l+1)}} \frac{\partial U^{(l+1)}}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial \theta^{(l)}}$$

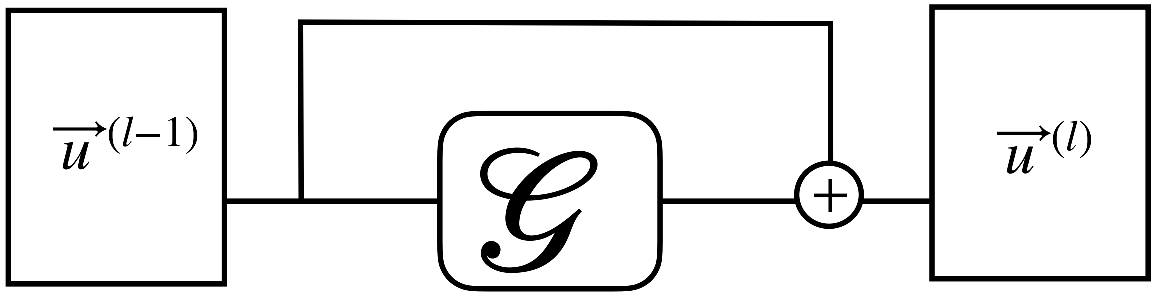
This matrix derivative is common to the stout force

Gauge covariant neural network

Akio Tomiya

Neural ODE of Cov-Net = “gradient flow”

Res-Net



arXiv: 1512.03385

Continuum Layer Limit

Neural ODE

$$\frac{d\vec{u}^{(t)}}{dt} = \mathcal{G}(\vec{u}^{(t)})$$

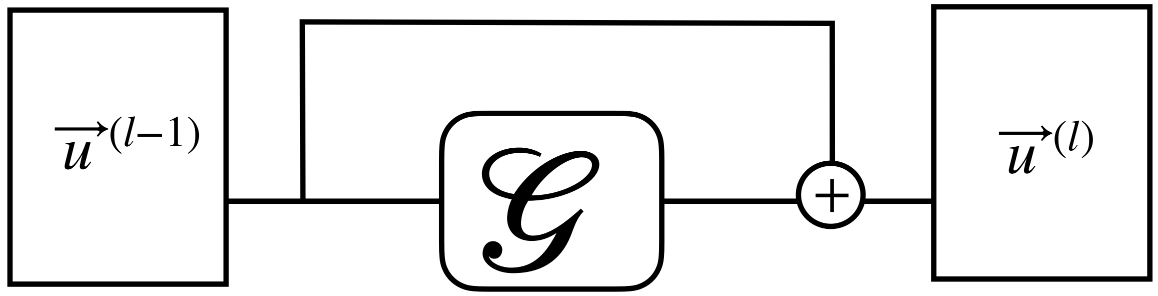
arXiv: 1806.07366
(Neural IPS 2018 best paper)

Gauge covariant neural network

Akio Tomiya

Neural ODE of Cov-Net = “gradient flow”

Res-Net



arXiv: 1512.03385

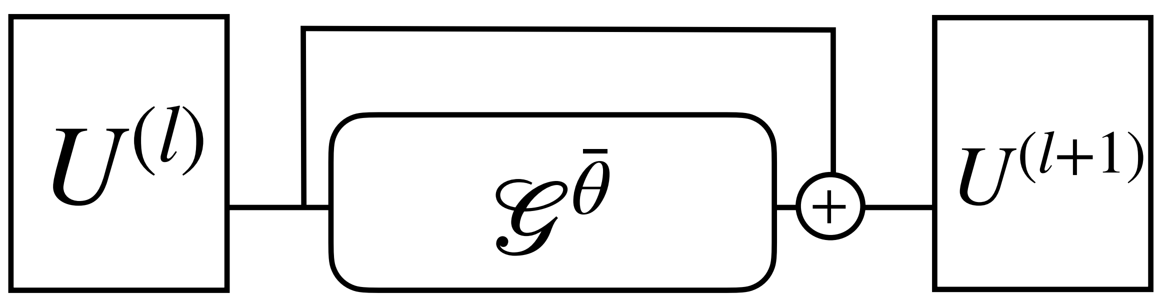
Continuum Layer Limit

Neural ODE

$$\frac{d\vec{u}^{(t)}}{dt} = \mathcal{G}(\vec{u}^{(t)})$$

arXiv: 1806.07366
(Neural IPS 2018 best paper)

Gauge-cov net



AT Y. Nagai arXiv: 2103.11965

Continuum Layer Limit

Neural ODE
for Gauge-cov NN

$$\frac{dU_\mu^{(t)}(n)}{dt} = \mathcal{G}^{\bar{\theta}}(U_\mu^{(t)}(n))$$

“Gradient” flow
(not has to be gradient of S)

“Continuous stout smearing is the Wilson flow”

2010 M. Luscher

Gauge covariant neural network

Akio Tomiya

Short summary

	Symmetry	Fixed parameter	Continuum limit of layers	How to Train
Usual neural network	Convolution: Translation	Convolution: Filtering (e.g Gaussian/ Laplasian)	Res-Net: Neural ODE	Delta rule and backprop Gradient opt.
Gauge cov. net AT Y. Nagai arXiv: 2103.11965	Gauge covariance Translation equiv, 90° rotation equiv	Smearing	Gradient flow	Extended Delta rule and backprop Gradient opt.

Re-usable stout force subroutine
(Implementation is easy & no need to use ML library)

Next, I show a demonstration
(Q. Gauge covariant net works?)

5. Application: Self-learning HMC for staggered, and domain-wall fermions

1. Background motivation (Machine learning for QCD)
2. Neural networks
   1. Neural network(NN) = Signal processing/Filtering with tuning
   2. Convolutional NN = Translational equivariance
3. Smearing ~ Neural network
4. Gauge Cov NN = **trainable** smearing, training for SU(N) fields
   1. Neural ODE for Cov NN = Gradient flow with trainable parameters
5. → Application: Self-learning HMC for staggered, and domain-wall fermions

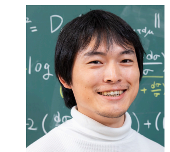
Applications

Configuration generation with machine learning is developing

Configuration generation for 2d scalar

Restricted Boltzmann machine + HMC: 2d scalar — A. Tanaka, AT 2017

The first challenge, machine learning + configuration generation. Wrong at critical pt. Not exact.




GAN (Generative adversarial network): 2d scalar — J. Pawlowski+ 2018, G. Endrodi+ 2018

Results look OK. No proof of exactness

Exact algorithm, gauge symmetry

Flow based model: 2d scalar, pure U(1), pure SU(N) — MIT, Google Brain 2019, 2020, 2021

Mimicking a trvializing map using a neural net which is reversible and has tractable Jacobian.
Exact algorithm, no dynamical fermions. SU(N) is treated with diagonalization.

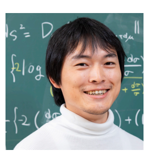
L2HMC for 2d U(1) (Sam Foreman+ 2021)

Dynamical fermions, 4 Dimension

Self-learning Monte Carlo (SLMC) for lattice QCD — arxiv 2010.11900 Y. Nagai, AT, A. Tanaka

Non-abelian gauge theory with dynamical fermion in 4d
Using gauge invariant action with linear regression
Exact. Costly (Diagonalize Dirac operator)





Self-learning Hybrid Monte Carlo for lattice QCD (SLHMC, This talk) — arxiv 2103.11965 Y. Nagai, AT

Non-abelian gauge theory with dynamical fermion in 4d
Using covariant neural network to parametrize the gauge invariant action
Exact


Application for the staggered in 4d
Problems to solve

arXiv: 2103.11965

Our neural network enables us to **parametrize** gauge symmetric action **covariant way.**

e.g.
$$S^{\mathrm{NN}}[U] = S_{\mathrm{plaq}}\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$
$$S^{\mathrm{NN}}[U] = S_{\mathrm{stag}}\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

Test of our neural network?

Can we mimic a different Dirac operator using neural net?

Artificial example for HMC:

Target action $$S[U] = S_{\mathrm{g}}\left[U\right] + S_{\mathrm{f}}\left[\phi, U; m = 0.3\right],$$

Action in MD $$S_\theta[U] = S_{\mathrm{g}}\left[U\right] + S_{\mathrm{f}}\left[\phi, \underline{U_\theta^{\mathrm{NN}}[U]}; m_{\mathrm{h}} = 0.4\right],$$

Q. Simulations with approximated action can be exact?
-> Yes! with SLHMC (Self-learning HMC)

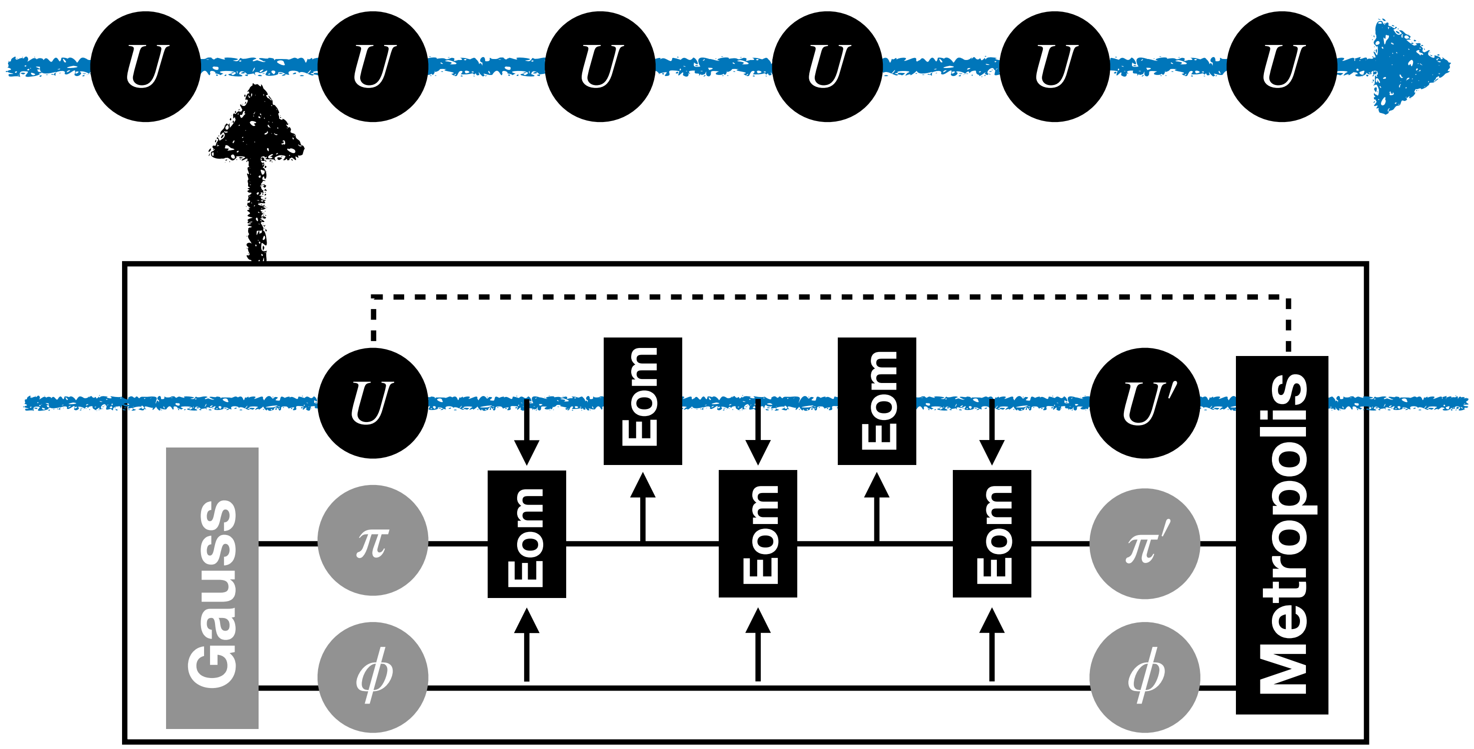
Gauge covariant net& SLHMC

SLHMC for gauge system with dynamical fermions

Akio Tomiya

arXiv: 2103.11965 and reference therein

HMC



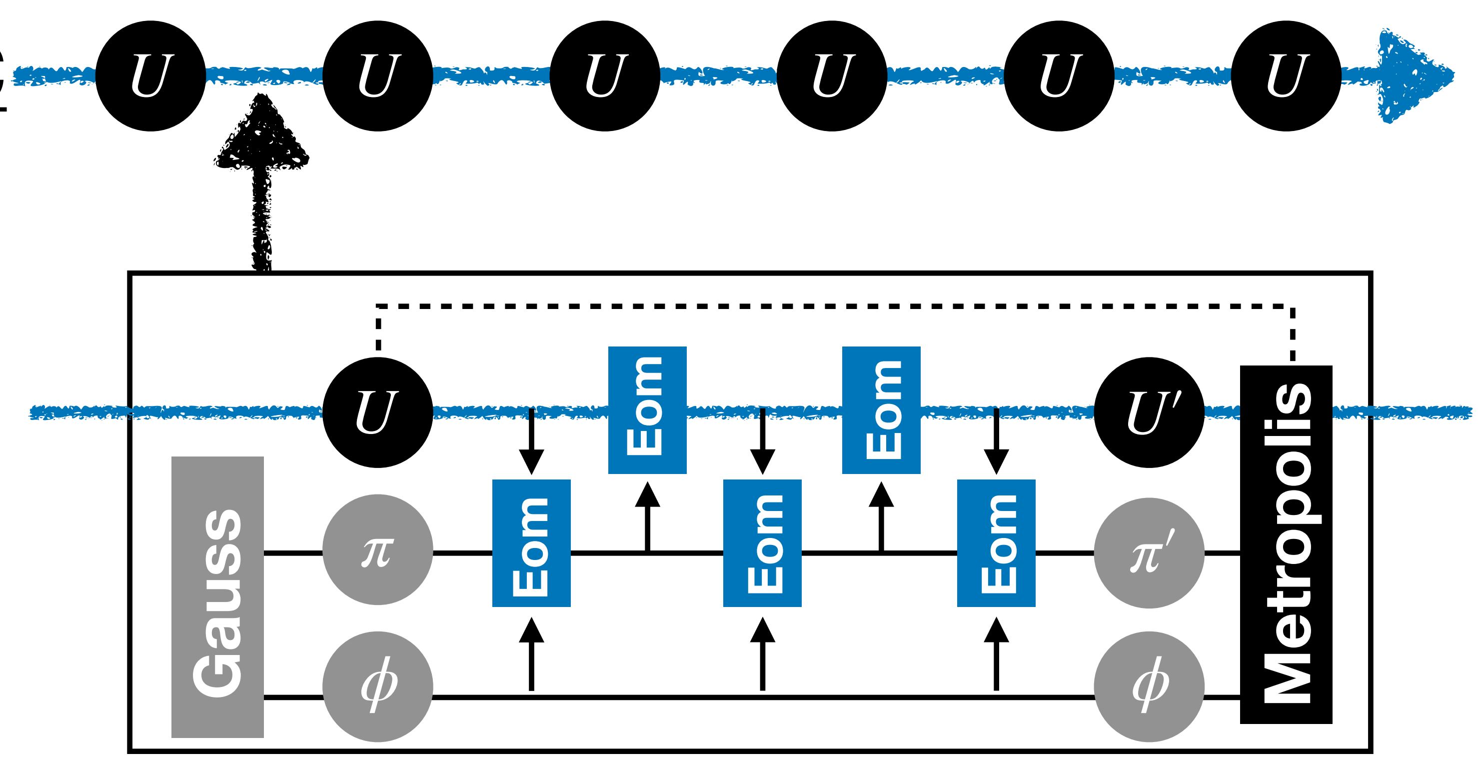
Eom **Metropolis**

Both use

$$H_{\mathrm{HMC}} = \frac{1}{2}\sum \pi^2 + S_{\mathrm{g}} + S_{\mathrm{f}}$$

Non-conservation of H cancels since the molecular dynamics is reversible

SLHMC



Metropolis

$$H = \frac{1}{2}\sum \pi^2 + S_{\mathrm{g}} + S_{\mathrm{f}}[U]$$

Eom

$$H = \frac{1}{2}\sum \pi^2 + S_{\mathrm{g}} + S_{\mathrm{f}}[U^{\mathrm{NN}}[U]]$$

Neural net approximated fermion action but exact

Application for the staggered in 4d

Lattice setup and question

arXiv: 2103.11965

Target Two color QCD (plaquette + staggered)

Algorithms SLHMC, HMC (comparison)

Parameter Four dimension, L=4, m = 0.3, beta = 2.7, Nf=4 (non-rooting)

Target action $S[U] = S_{\rm g}\left[U\right] + S_{\rm f}\left[\phi, U; m = 0.3\right],$ **For Metropolis Test**

Action in MD (for SLHMC) $S_\theta[U] = S_{\rm g}\left[U\right] + S_{\rm f}\left[\phi, U_\theta^{\rm NN}[U]; m_{\rm h} = 0.4\right],$

Observables Plaquette, Polyakov loop, Chiral condensate $\langle \bar{\psi}\psi \rangle$

Code Full scratch,
fully written in Julia lang.

LatticeQCD.jl

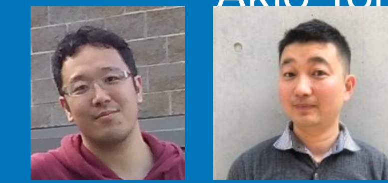
(But we added some functions on the public version)

AT+ (in prep)

Lattice QCD code

We made a public code in Julia Language

Akio Tomiya



AT & Y. Nagai in prep

What is julia?

1. Open source scientific language (Just in time compiler)
2. Fast as C/Fortran (sometime, faster)
3. Productive as Python
4. Machine learning friendly (Julia ML packages + Python libraries w/ PyCall)
5. Supercomputers support Julia

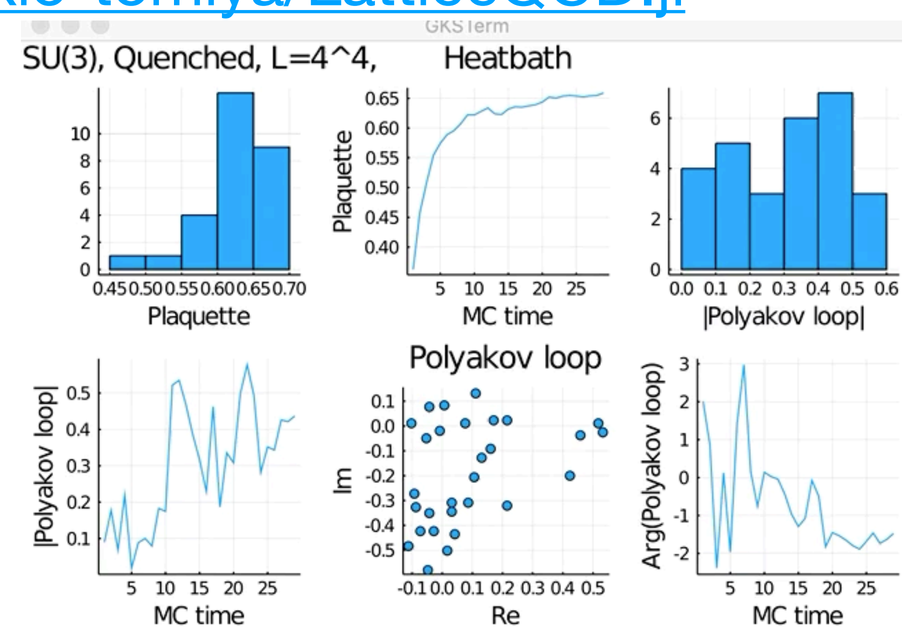
LatticeQCD.jl (Official package) : Laptop/desktop/PC-cluster/Jupyter (Google colab)

SU(Nc)-heatbath/SLHMC/SU(Nc) Stout/(R)HMC/staggered/Wilson-Clover
Domain-wall (experimental) + Measurements

3 steps in 5 min

1. Download Julia binary
2. Add the package through Julia package manager
3. Execute!

https://github.com/akio-tomiya/LatticeQCD.jl



Details (skip)

Network: trainable stout (plaq+poly)

arXiv: 2103.11965

Structure of NN

(Polyakov loop+plaq in the stout-type)

$$\Omega_\mu^{(l)}(n) = \rho_{\rm plaq}^{(l)} O_\mu^{\rm plaq}(n) + \begin{cases} \rho_{\rm poly,4}^{(l)} O_4^{\rm poly}(n) & (\mu = 4), \\ \rho_{\rm poly,s}^{(l)} O_i^{\rm poly}(n), & (\mu = i = 1,2,3) \end{cases}$$

All ρ is weight

O meas an loop operator

$$Q_\mu^{(l)}(n) = 2[\Omega_\mu^{(l)}(n)]_{\rm TA}$$

TA: Traceless, anti-hermitian operation

$$U_\mu^{(l+1)}(n) = \exp(Q_\mu^{(l)}(n)) U_\mu^{(l)}(n)$$

$$U_\mu^{\rm NN}(n)[U] = U_\mu^{(2)}(n)\left[U_\mu^{(1)}(n)\left[U_\mu(n)\right]\right]$$

2- layered stout
with 6 trainable parameters

Neural network Parametrized action:

$$S_\theta[U] = S_{\rm g}\left[U\right] + S_{\rm f}\left[\phi, U_\theta^{\rm NN}[U]; m_{\rm h} = 0.4\right],$$

Action for MD is built by gauge covariant NN

Loss function:

$$L_\theta[U] = \frac{1}{2}\left| S_\theta[U,\phi] - S[U,\phi] \right|^2,$$

Invariant under,
rot, transl, gauge trf.

Training strategy:

1.Train the network in prior HMC (online training+stochastic gr descent)
2.Perform SLHMC with fixed parameter

Details (skip)

Results: Loss decreases along with the training

arXiv: 2103.11965

Loss function:

$$L_\theta[U] = \frac{1}{2}\left|S_\theta[U,\phi] - S[U,\phi]\right|^2,$$

Intuitively, e^(-L) is understood as Boltzmann weight or reweighting factor.

Prior HMC run (training)

$$\frac{\partial S}{\partial \rho_i^{(l)}} = 2\,\mathrm{Re}\sum_{\mu',m}\mathrm{tr}\left[U_{\mu'}^{(l)\dagger}(m)\Lambda_{\mu',m}\frac{\partial C}{\partial \rho_i^{(l)}}\right]$$

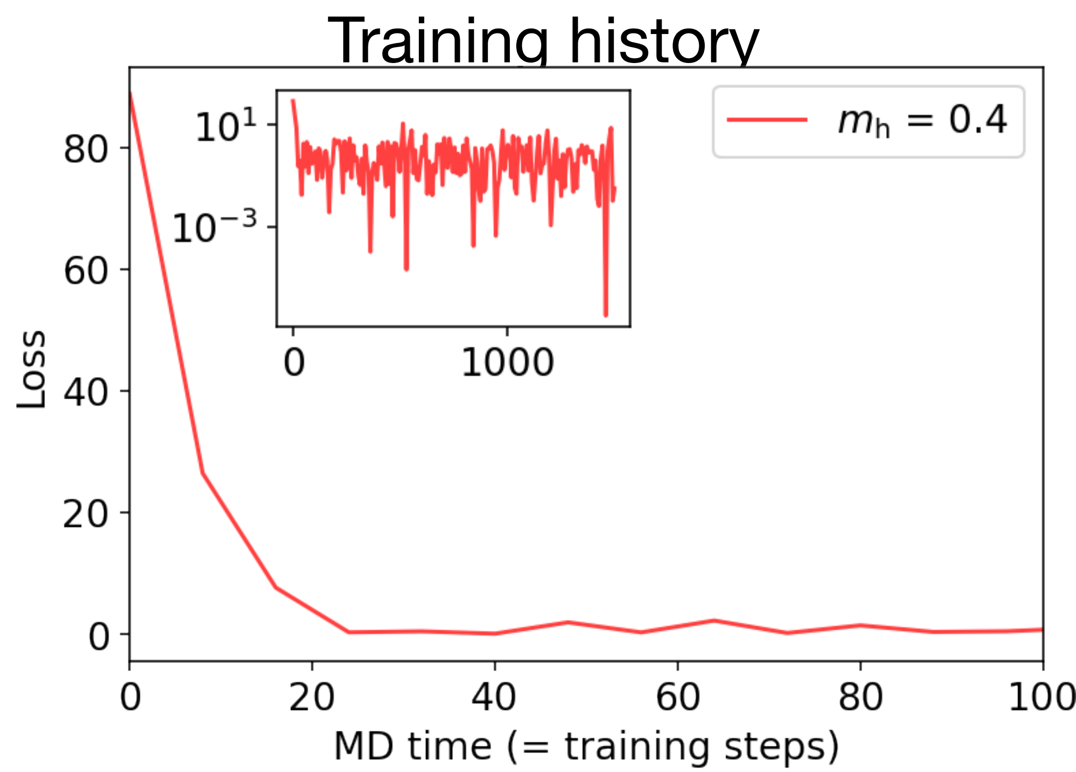
$$\theta \leftarrow \theta - \eta\frac{\partial L_\theta(\mathcal{D})}{\partial \theta},$$

$$\frac{\partial L_\theta(\mathcal{D})}{\partial w_i^{(L-1)}} = \frac{\partial L_\theta(\mathcal{D})}{\partial S_\theta}\frac{\partial S_\theta}{\partial w_i^{(L-1)}}$$

Ω: sum of un-traced loops

C: one U removed Ω

Λ: A polynomial of U. (Same object in stout)



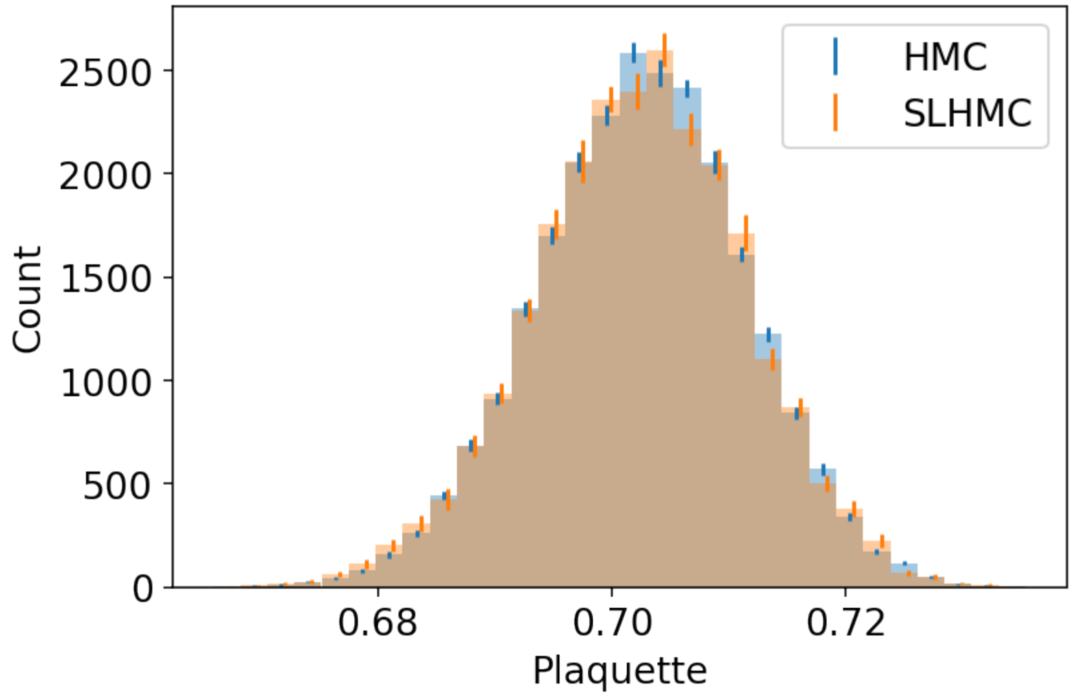
Without training, e^(-L)<< 1,
this means that candidate with approximated action never accept.
After training, e^(-L) ~1, and we get practical acceptance rate!

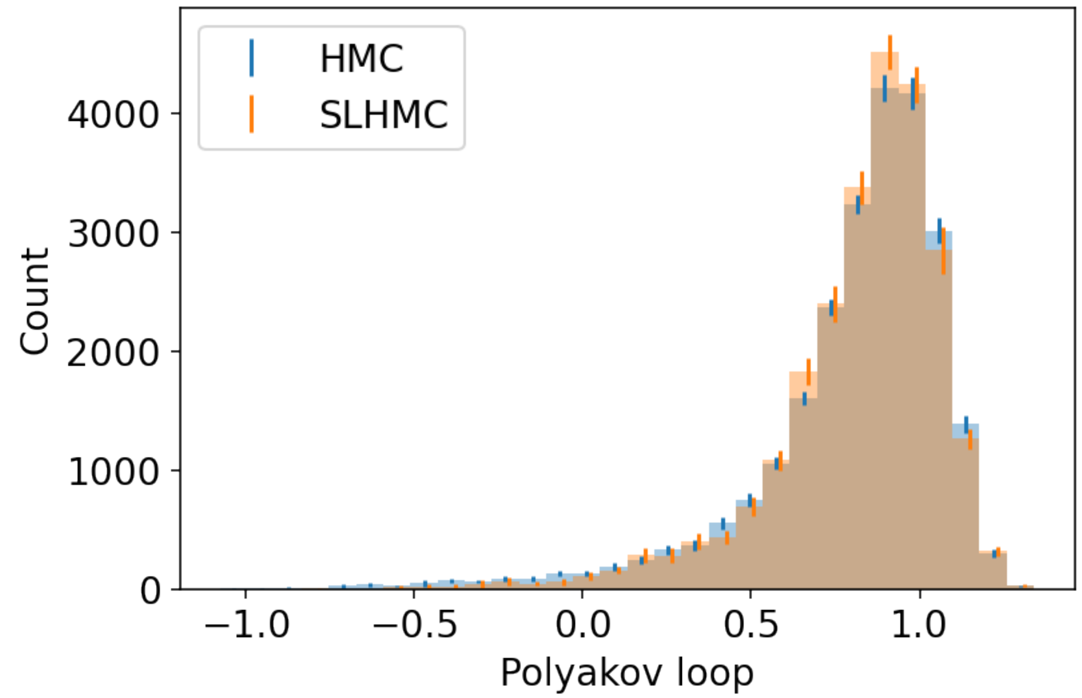
We perform SLHMC with these values!

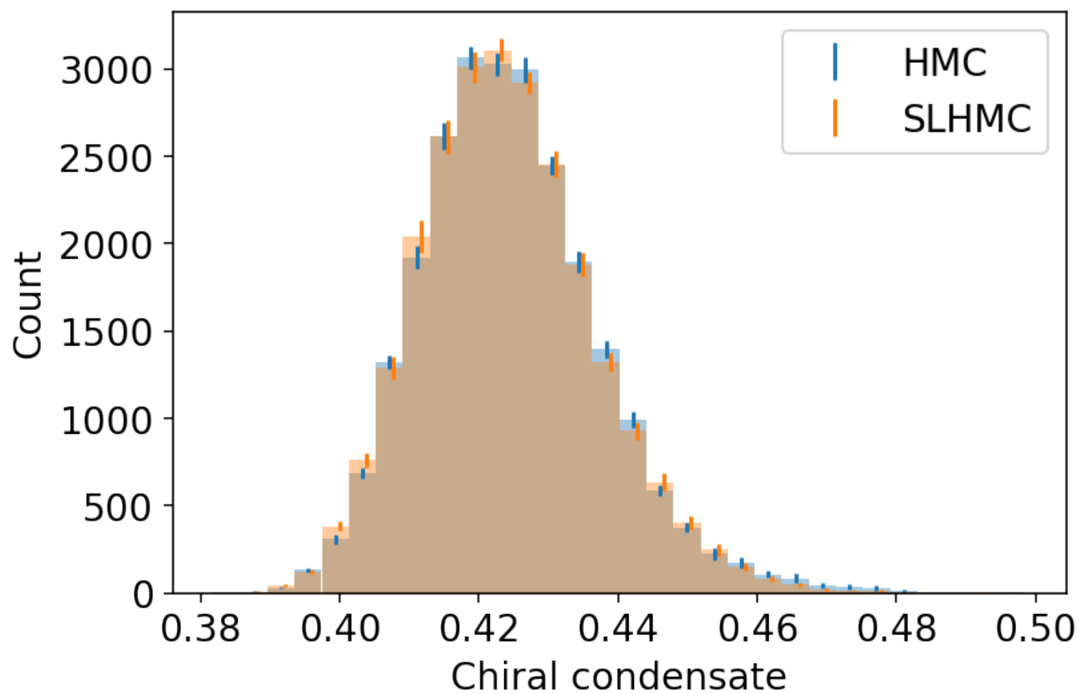
Application for the staggered in 4d

Results are consistent with each other

arXiv: 2103.11965







Expectation value

Algorithm	Observable	Value
HMC	Plaquette	0.7025(1)
SLHMC	Plaquette	0.7023(2)
HMC	|Polyakov loop|	0.82(1)
SLHMC	|Polyakov loop|	0.83(1)
HMC	Chiral condensate	0.4245(5)
SLHMC	Chiral condensate	0.4241(5)

Acceptance = 40%

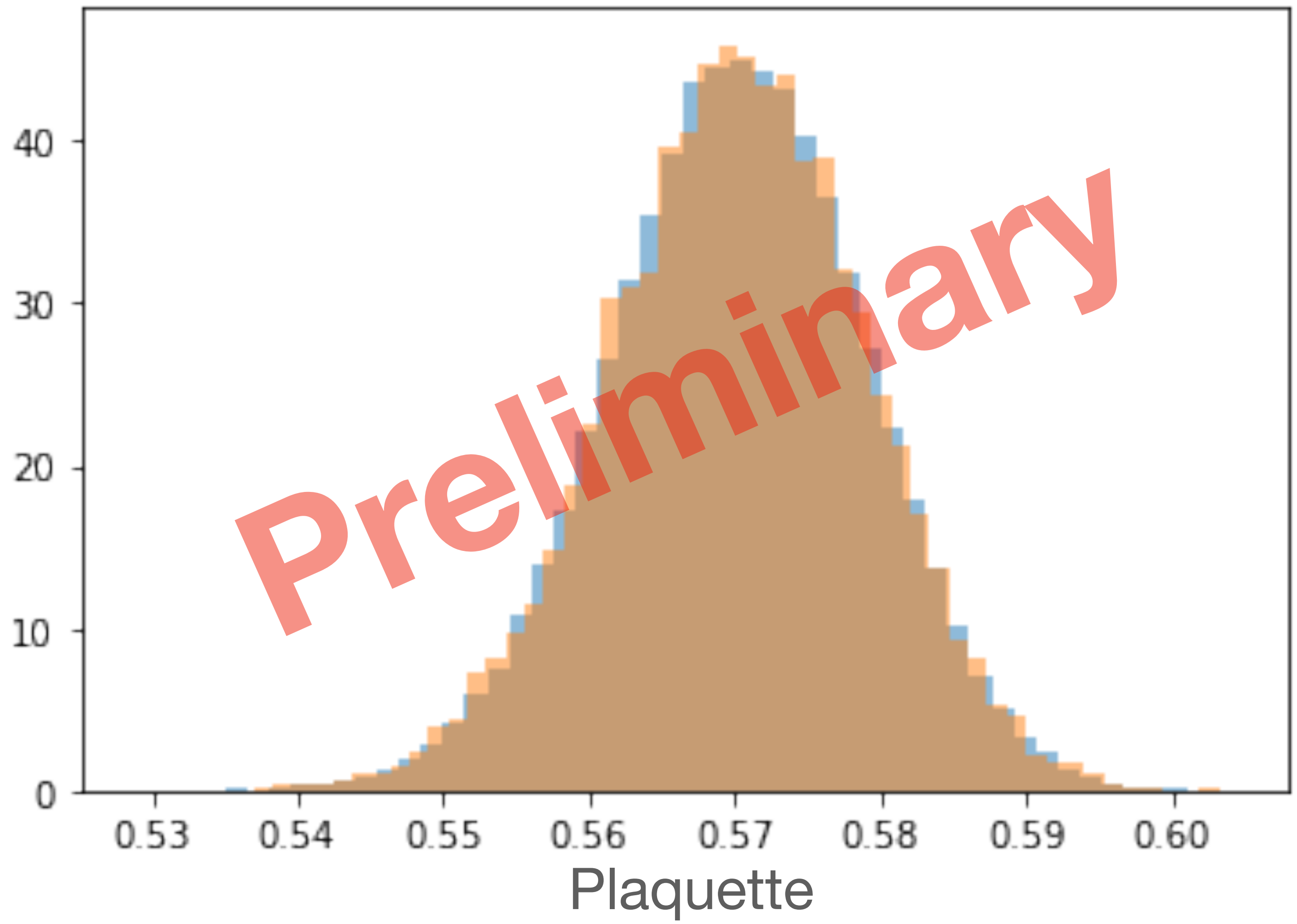
Preliminary

Results for SU(3), staggered

Akio Tomiya

SU(3), Nf=2, L=4, beta=5.7, dynamical stout staggered

Target: full QCD m = 0.3 (HMC & SLHMC)
In MD: m=0.4 + gauge cov NN (by SLHMC)



SU(3) with dynamical quarks in 4d can be deal with machine learning now!

(Very) Preliminary

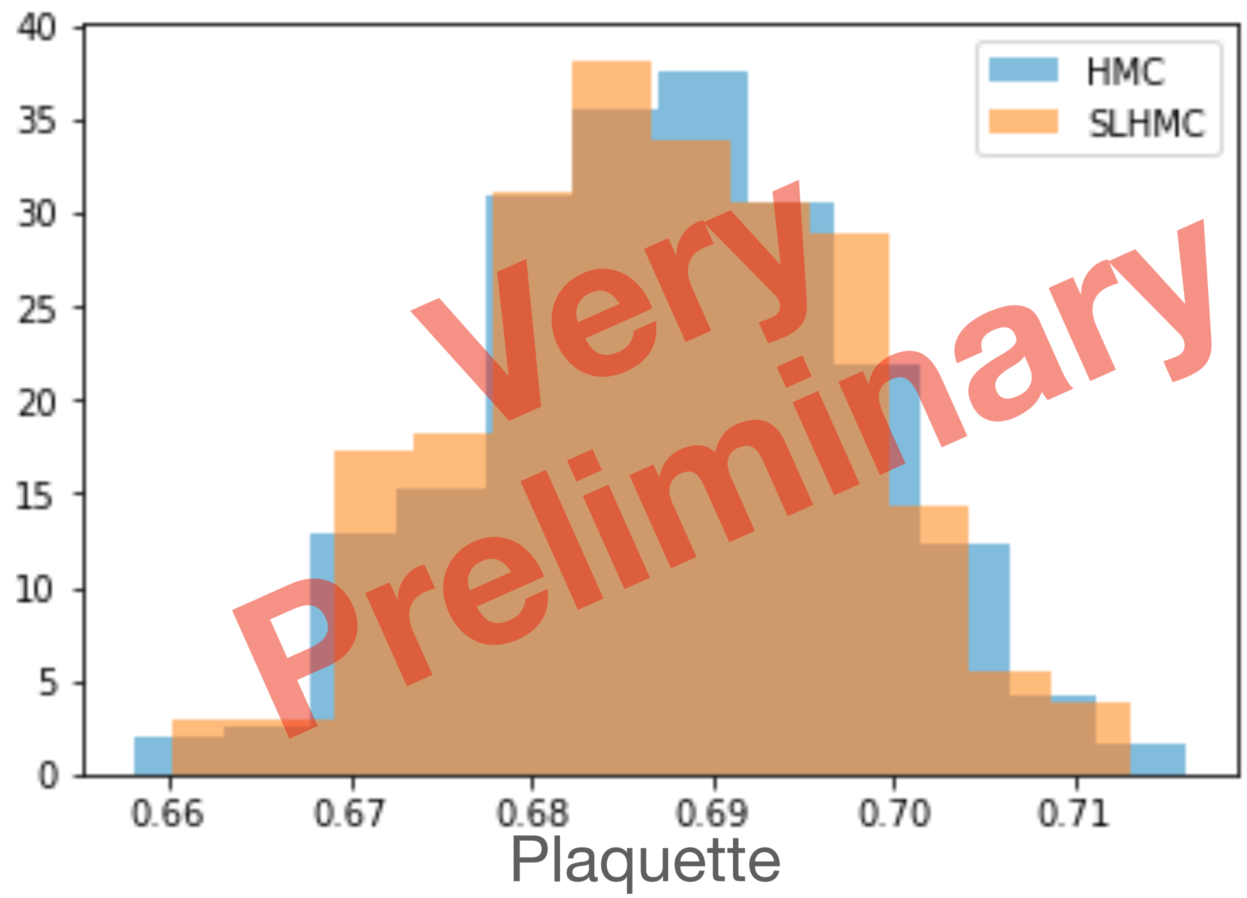
Akio Tomiya

SLHMC + Gauge Cov NN works for domain-wall fermions!

Parameter 4d, L=4, SU(2), beta = 2.7, m = 0.1, dynamical stout domain-wall (N5 = 4)

Target action (DW) $S[U] = S_{\mathrm{g}}\left[U\right] + S_{\mathrm{f}}\left[\phi, U; m = 0.1\right],$ **For Metropolis Test**

Action in MD (DW) $S_{\theta}[U] = S_{\mathrm{g}}\left[U\right] + S_{\mathrm{f}}\left[\phi, U_{\theta}^{\mathrm{NN}}[U]; m_{\mathrm{h}} = 0.12\right],$



Acceptance ~ 50%

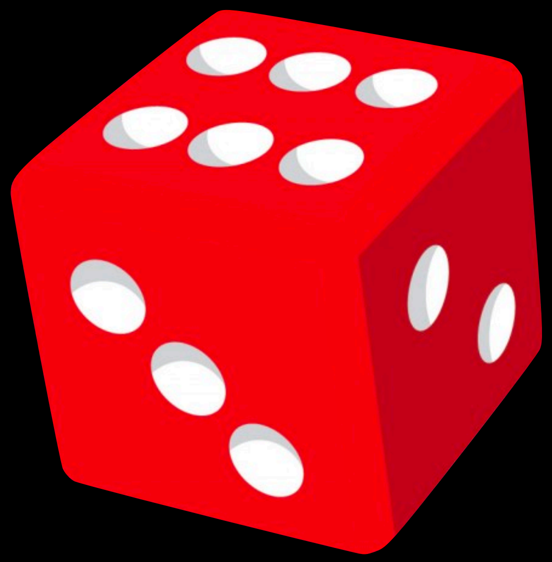
It looks working well

Summary and future work

Akio Tomiya
arXiv: 2103.11965

We propose and use gauge covariant neural net

- Covariant neural network = trainable smearing (as Convolutional layers ~ trainable filters)
    - We develop the delta rule for SU(N) valued link field variables (skipped).
      One can implement this on a code with smeared HMC. *Most of necessary subroutines are common to the stout force.*
    - We provide how to construct a gauge invariant loss function
    - We parametrize QCD action in a gauge covariant way
    - Neural ODE for the gauge covariant NN = “gradient flow”
- Self-learning HMC = HMC+ neural network parametrized molecular dynamics, exact
- We performed simulations with the covariant neural network parametrized action
    - Training: it has only 6 parameters but loss decreases to O(1).
    - Results of SLHMC consistent with HMC. We successfully generated configurations with 4 dimensional non-abelian gauge theory with dynamical fermions with parametrized action
    - With SU(3) staggered and domain-wall fermions work well
- **Future works:**
  Combine with the flow based sampling?
  Reducing N5 in domain-wall simulation (*a la* Mobius accelerated domain-wall fermions)?



What is the neural networks?
(this slide might be too much lecture-ish…)

1d example of convolution:

Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

What is the neural networks?
(this slide might be too much lecture-ish…)

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

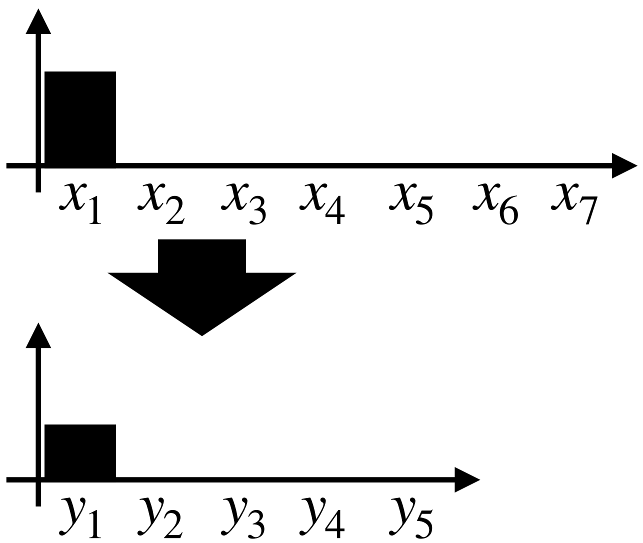
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

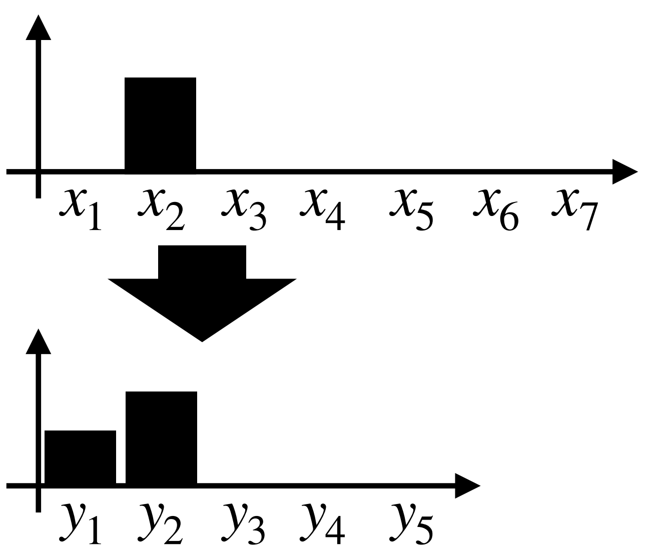
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

Akio Tomiya

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

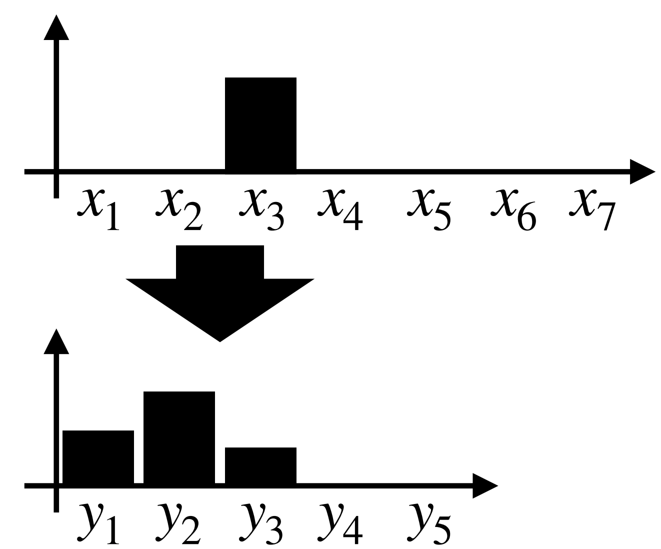
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

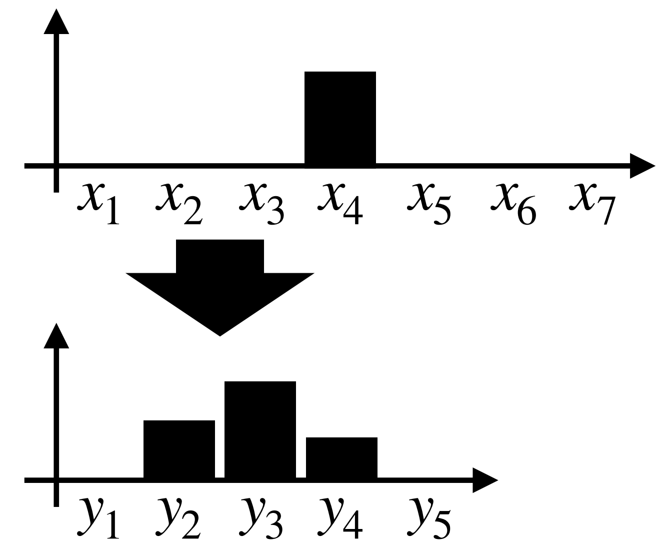
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

Akio Tomiya

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

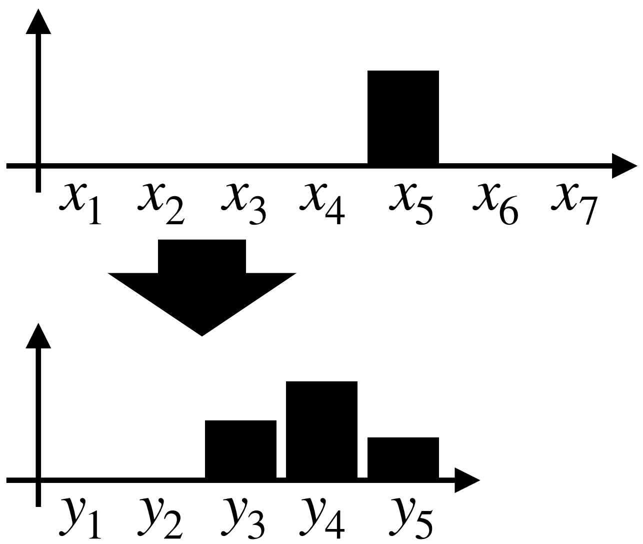
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\text{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

Akio Tomiya

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

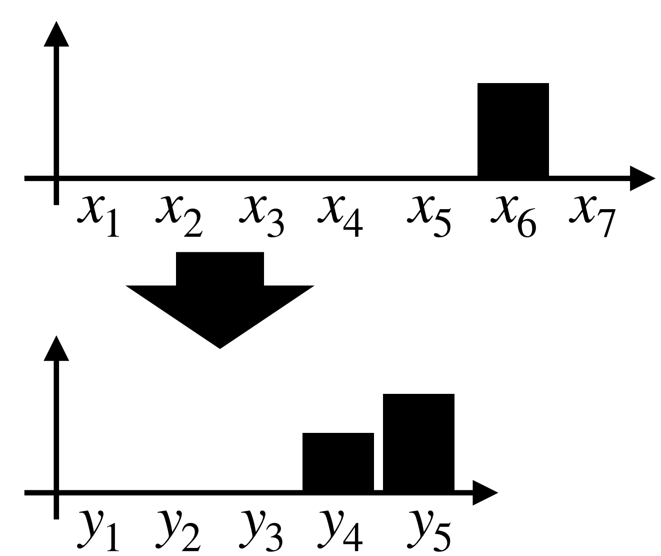
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?
(this slide might be too much lecture-ish…)

Akio Tomiya

1d example of convolution:

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

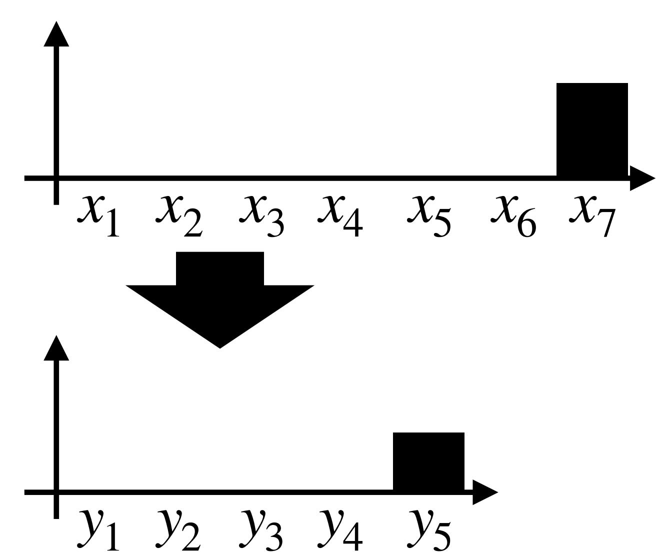
Fully connected:

$$\vec{y} = W\vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

If elements in the vector x are shifted ($x_1 \to x_2, \cdots$), output is randomly changed

Convolution (weight sharing, translation equivariant):

$$\vec{y} = W^{\mathrm{conv}}\vec{x} = \begin{pmatrix} c_1 & c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & c_1 & c_2 & c_3 & 0 & 0 & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & c_1 & c_2 & c_3 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_2 & c_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$



If elements in the vector x is shifted, output is shifted same way (equivariance)

What is the neural networks?

Akio Tomiya

Convolution + fully connected

Fukushima, Kunihiko (1980)
Zhang, Wei (1988) + a lot!

e.g.:

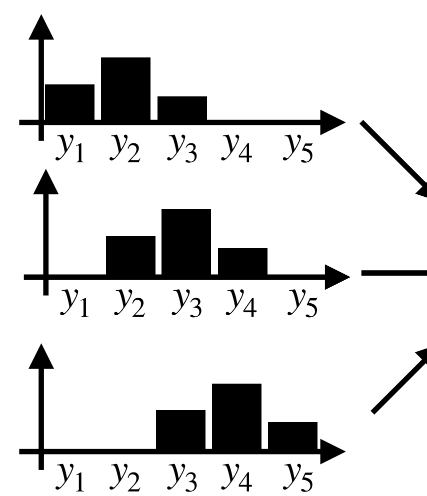
$\mathrm{Sum}\Big(\ \Big(\Big(\Big($  $* \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \Big) * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \Big) * \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \Big)$

This is invariant under the translation of input (global average pooling)
= translational *invariant* output

$* \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$ makes *equivariant* map

$\mathrm{Sum}(\,\cdot\,)$ makes output *invariant* →

(To enlarge the number of parameter, multiple filters are used)



$\sum_i y_i$ = same

QFT analogy: If field ϕ is locally transformed, $D_\mu \phi$ is *covariant*,
$\phi^\dagger D_\mu \phi$ is *invariant*

Gauge covariant neural network

Training can be done with (extended) back propagation

AT Y. Nagai arXiv: 2103.11965

Gauge inv. loss function can be constructed by gauge invariant actions

$$S^{\mathrm{NN}}[U] = S\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

S: gauge action or fermion action

For example:

$$S^{\mathrm{NN}}[U] = S_{\mathrm{plaq}}\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

$$S^{\mathrm{NN}}[U] = S_{\mathrm{plaq}}\left[U_\mu^{\mathrm{NN1}}(n)[U]\right] + S_{\mathrm{rect}}\left[U_\mu^{\mathrm{NN2}}(n)[U]\right]$$

$$S^{\mathrm{NN}}[U] = S_{\mathrm{stag}}\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

and so on

We can construct,
Gauge invariant, translational invariant, rotational invariant
outputs through actions

Gauge covariant neural network

Training can be done with (extended) back propagation

AT Y. Nagai arXiv: 2103.11965

Gauge inv. loss function can be constructed by gauge invariant actions

$$S^{\mathrm{NN}}[U] = S\left[U_\mu^{\mathrm{NN}}(n)[U]\right]$$

S: gauge action or fermion action

Loss function $$L_\theta[U] = f\left(S^{\mathrm{NN}}[U]\right)$$

f: mean-square for example, mini-batch

(c.f. Behler-Parrinello type neural net)