Geometric Deep Learning: From Pure Math to Applications

Jan E. Gerken

CHALMERS
UNIVERSITY OF TECHNOLOGY

WASP
WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

13th April 2023
Amplitudes Group Meeting
Uppsala University

What is Geometric Deep Learning?

[Bronstein et al. 2017]

Geometric Deep Learning = Geometry + Deep Learning

What is Geometric Deep Learning?

[Bronstein et al. 2017]

$$\text{Geometric Deep Learning} = \underbrace{\text{Geometry}}_{\checkmark} + \underbrace{\text{Deep Learning}}$$

What is Geometric Deep Learning?

[Bronstein et al. 2017]

$$\text{Geometric Deep Learning} = \underbrace{\text{Geometry}}_{\checkmark} + \underbrace{\text{Deep Learning}}_{\substack{\parallel \\ \text{Machine Learning} \\ \text{with neural networks}}}$$

What is Geometric Deep Learning?

[Bronstein et al. 2017]

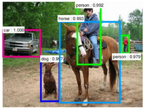
Geometric Deep Learning = Geometry + Deep Learning

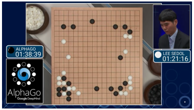
Geometry: ✓

Deep Learning = Machine Learning with neural networks

Chat GPT

OpenAI





[Bronstein et al. 2017]

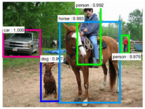
Geometric Deep Learning = Geometry ⊕ Deep Learning

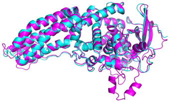
Geometry: ✓

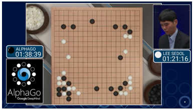
⊕: ?

Deep Learning = Machine Learning with neural networks

Chat GPT
OpenAI

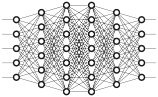






Machine learning with neural networks

- Neural networks

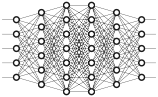


$$\Rightarrow \quad f_\theta : \mathbb{R}^m \to \mathbb{R}^n\,, \quad \theta \in \mathbb{R}^N\text{: parameters}$$

e.g. $f_\theta : \text{picture} \mapsto p(\text{cat})$

Machine learning with neural networks

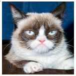
- Neural networks



$$\Rightarrow \quad f_\theta : \mathbb{R}^m \to \mathbb{R}^n \,, \qquad \theta \in \mathbb{R}^N\text{: parameters}$$

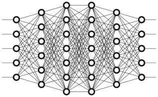
$$\text{e.g.} \quad f_\theta : \text{picture} \mapsto p(\text{cat})$$

- Training data
  examples $x \mapsto y$ of target function, e.g.

$\mathcal{D} = \{$  $\mapsto 1.0 \,,$  $\mapsto 0.0 \,, \quad \ldots \}$

Machine learning with neural networks

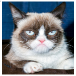
- Neural networks



$$\Rightarrow \quad f_\theta : \mathbb{R}^m \rightarrow \mathbb{R}^n \,, \qquad \theta \in \mathbb{R}^N\text{: parameters}$$

$$\text{e.g.} \quad f_\theta : \text{picture} \mapsto p(\text{cat})$$

- Training data
  examples $x \mapsto y$ of target function, e.g.

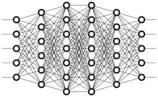
$\mathcal{D} = \Bigg\{$  $\mapsto 1.0 \,,$  $\mapsto 0.0 \,, \quad \ldots \Bigg\}$

- Error function
  quantifies deviation between current guess $f_\theta(x)$ and true answer y, e.g.

$$\mathcal{E}(f_\theta(x), y) = (f_\theta(x) - y)^2$$

Machine learning with neural networks

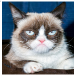
- Neural networks



$$\Rightarrow \quad f_\theta : \mathbb{R}^m \to \mathbb{R}^n \,, \qquad \theta \in \mathbb{R}^N\text{: parameters}$$

$$\text{e.g.} \quad f_\theta : \text{picture} \mapsto p(\text{cat})$$

- Training data
  examples $x \mapsto y$ of target function, e.g.

$\mathcal{D} = \Big\{$  $\mapsto 1.0\,,$  $\mapsto 0.0\,, \quad \ldots \Big\}$

- Error function
  quantifies deviation between current guess $f_\theta(x)$ and true answer y, e.g.

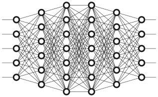
$$\mathcal{E}(f_\theta(x), y) = (f_\theta(x) - y)^2$$

- Optimization procedure
  minimize error over data iteratively

$$\theta' = \theta - \eta \nabla_\theta \sum_{(x,y)\in\mathcal{D}} \mathcal{E}(f_\theta(x), y)$$

Machine learning with neural networks

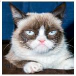
- Neural networks



$$\Rightarrow \quad f_\theta : \mathbb{R}^m \to \mathbb{R}^n\,, \qquad \theta \in \mathbb{R}^N\text{: parameters}$$

$$\text{e.g.} \quad f_\theta : \text{picture} \mapsto p(\text{cat})$$

- Training data
  examples $x \mapsto y$ of target function, e.g.

$\mathcal{D} = \Big\{$  $\mapsto 1.0\,,$  $\mapsto 0.0\,, \quad \ldots \Big\}$

- Error function
  quantifies deviation between current guess $f_\theta(x)$ and true answer y, e.g.

$$\mathcal{E}(f_\theta(x), y) = (f_\theta(x) - y)^2$$

- Optimization procedure
  minimize error over data iteratively

$$\theta' = \theta - \eta \nabla_\theta \sum_{(x,y)\in\mathcal{D}} \mathcal{E}(f_\theta(x), y)$$

$\Rightarrow$ ***Very powerful recipe***

Neural networks

- A neural network is a certain way to define a family of functions f_θ
- For $x \in \mathbb{R}^n$, a ***fully connected*** layer computes

$$z_j(x) = \sigma(\underbrace{W_j}_{\text{weights}} \cdot x + \underbrace{b_j}_{\text{biases}}) \qquad \text{where} \quad W_j \in \mathbb{R}^{m \times n}, \quad b_j \in \mathbb{R}^m, \quad \sigma : \mathbb{R}^m \longrightarrow \mathbb{R}^m \qquad (\star)$$

- σ is a (simple) non-linear function that acts component-wise, often ***ReLU***

$$\mathrm{ReLU}(x) = \max(x, 0)$$

Neural networks

- A neural network is a certain way to define a family of functions f_θ
- For $x \in \mathbb{R}^n$, a ***fully connected*** layer computes

$$z_j(x) = \sigma(W_j \cdot x + b_j) \qquad \text{where} \quad \underbrace{W_j}_{\textit{weights}} \in \mathbb{R}^{m \times n}, \quad \underbrace{b_j}_{\textit{biases}} \in \mathbb{R}^m, \quad \sigma : \mathbb{R}^m \to \mathbb{R}^m \qquad (\star)$$

- σ is a (simple) non-linear function that acts component-wise, often ***ReLU***

$$\mathrm{ReLU}(x) = \max(x, 0)$$

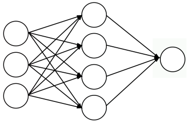
- The neural network iterates z

$$f_\theta(x) = z_L \circ z_{L-1} \circ \cdots \circ z_1(x) \qquad (\star\star)$$

  with the parameters

$$\theta = \{W_1, \ldots, W_L, b_1, \ldots, b_L\}$$

- Name stems from similarity of $(\star)$ to neurons and visualization of $(\star\star)$ as graph



Neural networks

- A neural network is a certain way to define a family of functions f_θ
- For $x \in \mathbb{R}^n$, a ***fully connected*** layer computes

$$z_j(x) = \sigma(W_j \cdot x + b_j) \qquad \text{where} \quad \underbrace{W_j}_{\textit{weights}} \in \mathbb{R}^{m\times n}, \quad \underbrace{b_j}_{\textit{biases}} \in \mathbb{R}^m, \quad \sigma : \mathbb{R}^m \to \mathbb{R}^m \qquad (\star)$$

- σ is a (simple) non-linear function that acts component-wise, often ***ReLU***

$$\mathrm{ReLU}(x) = \max(x, 0)$$

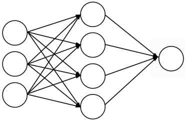
- The neural network iterates z

$$f_\theta(x) = z_L \circ z_{L-1} \circ \cdots \circ z_1(x) \qquad (\star\star)$$

with the parameters

$$\theta = \{W_1, \ldots, W_L, b_1, \ldots, b_L\}$$

- Name stems from similarity of $(\star)$ to neurons and visualization of $(\star\star)$ as graph

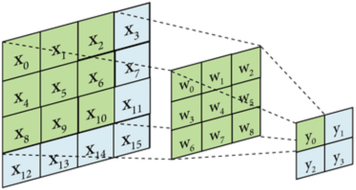


- Choose output dimension to match task, e.g. $f_\theta(x) \in \mathbb{R}$ for binary classifier
- Compute gradient of loss w.r.t. W_j, b_j using ***backpropagation***

Convolutional neural networks (CNNs)

[Fukushima 1980]

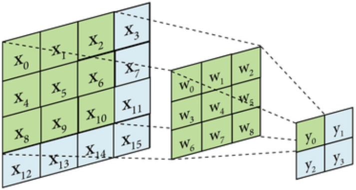
- Used mostly for input images: RGB pixel values on a grid $\mathcal{G} = ([0, w] \times [0, h]) \cap \mathbb{Z}^2$
- Interpret convolution as sliding filter over image



Convolutional neural networks (CNNs)

[Fukushima 1980]

- Used mostly for input images: RGB pixel values on a grid $\mathcal{G} = ([0, w] \times [0, h]) \cap \mathbb{Z}^2$
- Interpret convolution as sliding filter over image



- Interpret input as function $f : \mathcal{G} \to \mathbb{R}^3$
- Convolutional layer

kernel

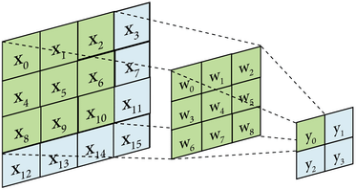
$$z_j(x) = \sigma\big((\kappa_j * f)(x) + b_j\big) = \sigma\Big(\sum_{y \in \mathcal{G}} \kappa_j(y - x) \cdot f(y) + b_j\Big), \quad \text{where} \quad \kappa_j : \mathcal{G} \to \mathbb{R}^{m \times n}$$

- Kernel has finite support, e.g. 3×3 or 5×5

Convolutional neural networks (CNNs)

[Fukushima 1980]

- Used mostly for input images: RGB pixel values on a grid $\mathcal{G} = ([0, w] \times [0, h]) \cap \mathbb{Z}^2$
- Interpret convolution as sliding filter over image



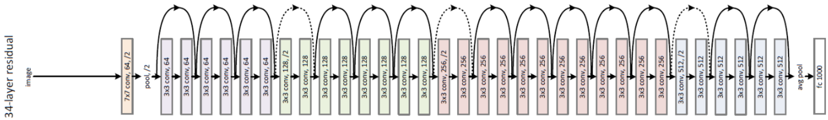
- Interpret input as function $f : \mathcal{G} \to \mathbb{R}^3$
- Convolutional layer

kernel

$$z_j(x) = \sigma\big((\kappa_j * f)(x) + b_j\big) = \sigma\Big(\sum_{y \in \mathcal{G}} \kappa_j(y - x) \cdot f(y) + b_j\Big), \quad \text{where} \quad \kappa_j : \mathcal{G} \to \mathbb{R}^{m \times n}$$

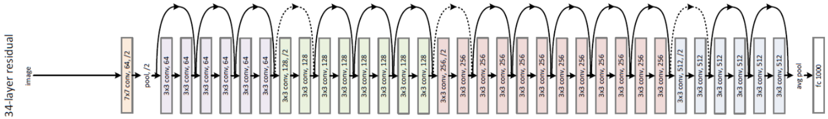
- Kernel has finite support, e.g. 3×3 or 5×5
- Effectively: Fully connected layer with constraints on W
- CNNs stack many convolutional layers
- Higher-dimensional versions and versions on graphs exist

Deep learning



- Stack many (usually $\sim 5 - 100$) NN layers to build ***deep*** neural networks
- Typically many parameters (largest model has more than 1 trillion parameters)
- Typically trained on large datasets (e.g. TBs of text data, hundreds of millions of images)

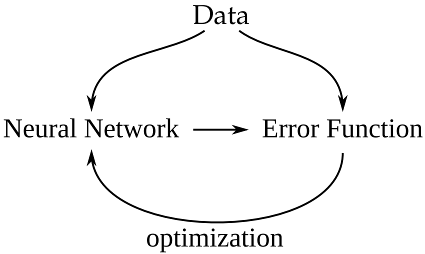
Deep learning



- Stack many (usually $\sim 5 - 100$) NN layers to build ***deep*** neural networks
- Typically many parameters (largest model has more than 1 trillion parameters)
- Typically trained on large datasets (e.g. TBs of text data, hundreds of millions of images)
- Need various tricks to make training stable and efficient
- Use graphical processing units (GPUs) and computing clusters to run training
- Hugely successful, at the heart of the AI boom:
  - Computer Vision
  - Natural Language Processing
  - Reinforcement Learning
  - Protein Folding

  ⋮

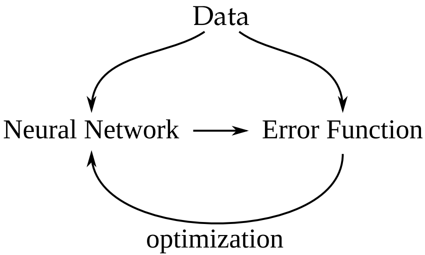
Enter: Geometry



Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

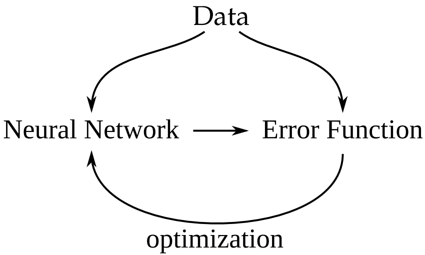


Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.

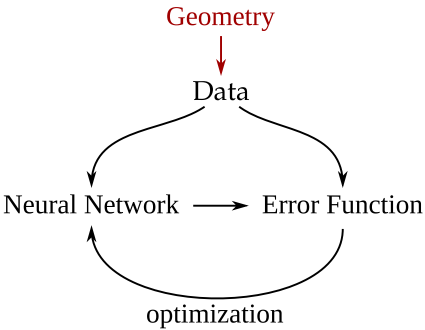


Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.

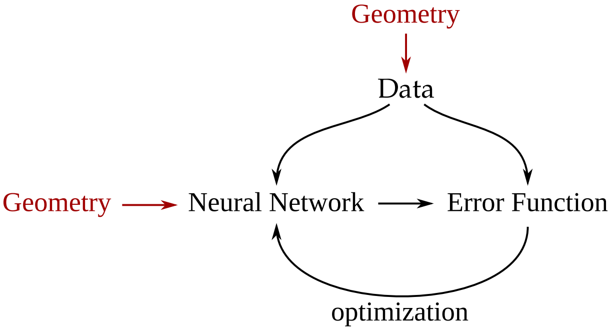


Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.

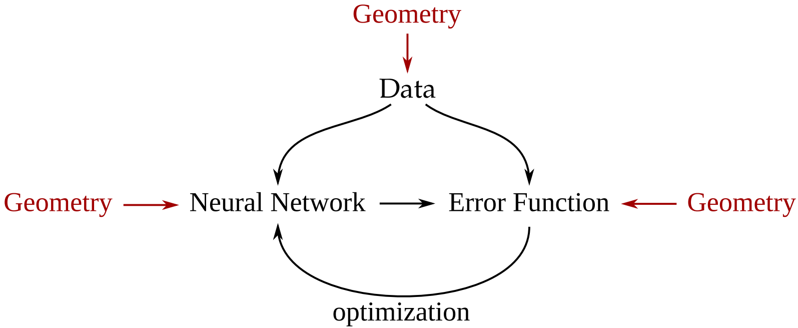


Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.

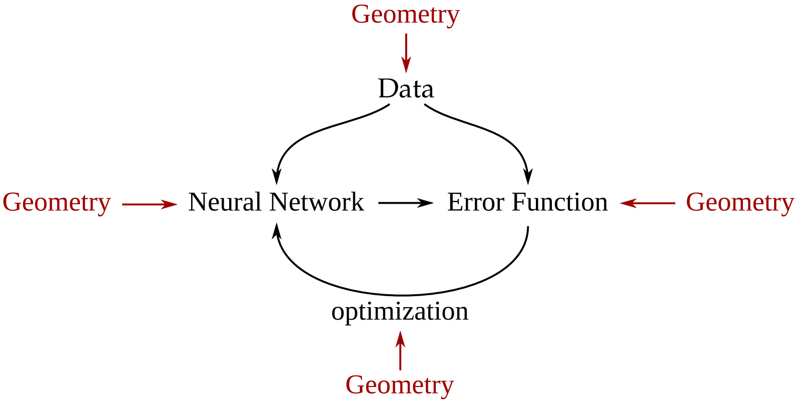


Enter: Geometry

Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.

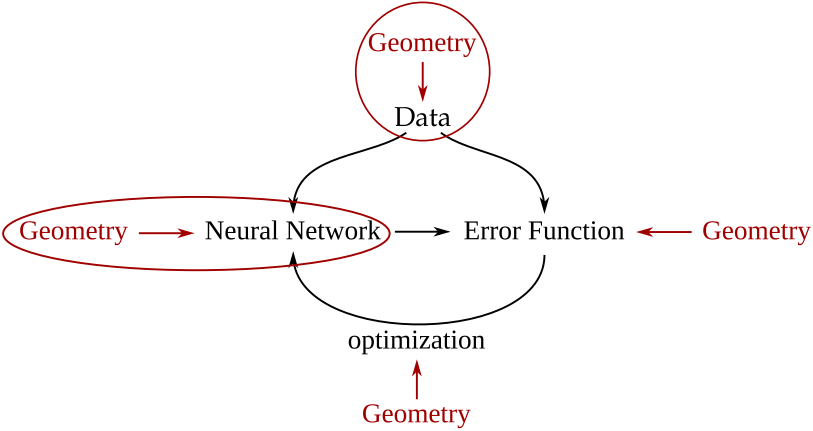


Enter: Geometry

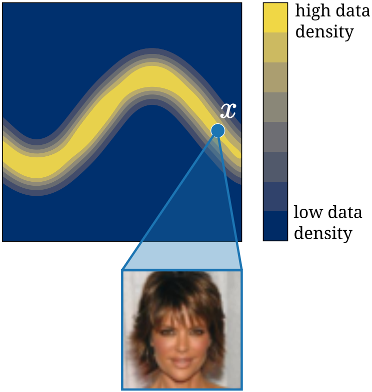
Deep learning is lacking a strong theoretical foundation:

- Neural networks are complicated functions
- The training process is stochastic

Geometry can help to alleviate these problems.



Geometry of the training data

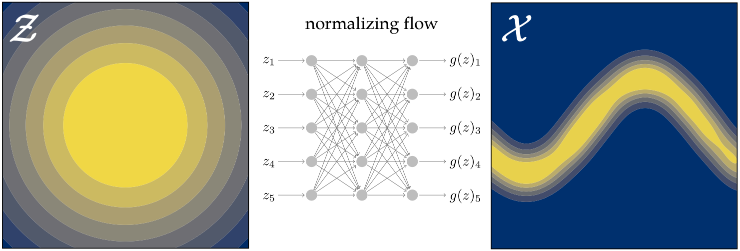


- Manifold Hypothesis: Data lies on low-dim. submanifold of high-dim. input space
- E.g. MNIST pictures lie on ~ 30-dim. submanifold of $28 \times 28 = 784$ dim. input space

Learned diffeomorphisms

[Dinh, Krueger, Bengio, 2014]

- How to characterize the data manifold?
- Can learn a diffeomorphism between a simple distribution and data distribution

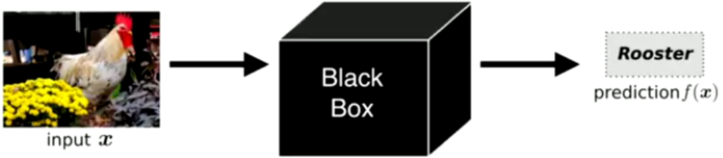


- Diffeomorphism is given by another neural network, a ***normalizing flow***
- Get access to the data manifold in a functional form
- Used e.g. to sample field configurations in lattice field theory

[Albergo et al. 2021]
[Bacchio et al. 2022]
[Nicoli et al. 2023]

Explainable AI (XAI)

[Review: Samek et al. 2021]

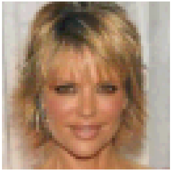


- Neural network classifiers lack inherent interpretability
- This is in contrast to more traditional methods like linear- or physical models
- For safety-critical applications this poses a serious challenge in practice
- Research progress can also be impeded
- Need explanations which provide insight into the neural network decisions

Counterfactual explanations

- ***Counterfactual of a sample***: Data point close to original but with different classification
- Difference between original and counterfactual reveals features which led to classification
- Example from CelebA dataset, classified as not-blonde:

original x



counterfactual x'

$|x - x'|$

Adversarial Examples

- Small perturbations can lead to misclassifications

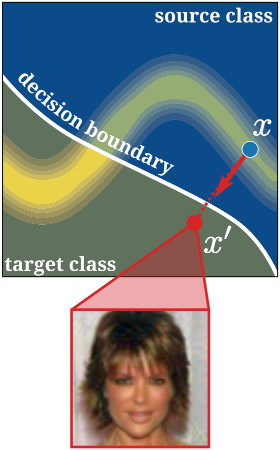
$p_{\text{blonde}}\Bigg($  $\Bigg) = 0.01$ but $p_{\text{blonde}}\Bigg($  $\Bigg) = 0.99$

Adversarial Examples

- Small perturbations can lead to misclassifications

$p_{\text{blonde}}\Big($  $\Big) = 0.01$ but $p_{\text{blonde}}\Big($  $\Big) = 0.99$

- Reason: Classifier only trained on the data manifold

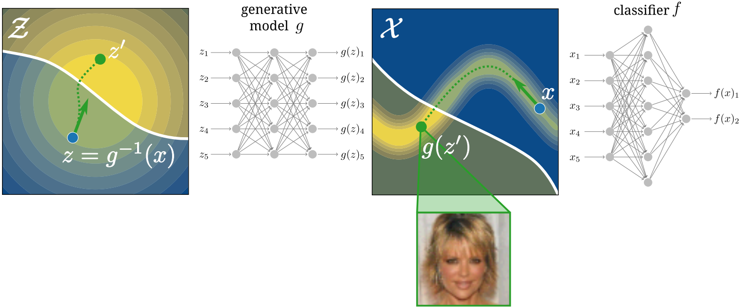


- Can use normalizing flows to optimize along the data manifold ⇒ ***counterfactuals***

original
not blond

counterfactual
blond (p ≈ 0.99)

adversarial example
blond (p ≈ 0.99)



$$x^{(i+1)} = x^{(i)} + \lambda\, \gamma^{-1} \frac{\partial f_t}{\partial x}(x^{(i)}) + \mathcal{O}(\lambda^2)$$

Gradient ascent in base space

- Gradient ascent in Z for class k of the classifier f with learning rate λ:

$$z^{(t+1)} = z^{(t)} + \lambda \frac{\partial (f \circ g)_k}{\partial z}(z^{(t)})$$

- Using change-of-variable under the flow:

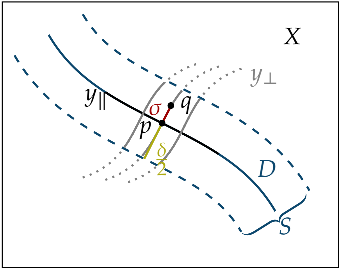
Theorem

Gradient ascent in the base space Z is given by

$$x^{(t+1)} = x^{(t)} + \lambda \, \boldsymbol{\gamma^{-1}(x^{(t)})} \, \frac{\partial f_k}{\partial x}(x^{(t)}) + \mathcal{O}(\lambda^2)$$

where $\gamma^{-1}(x) = \left(\frac{\partial g}{\partial z}\frac{\partial g}{\partial z}^T\right)(g^{-1}(x))$ is the inverse of the induced metric on X from Z under the flow g.

Data coordinates



- Assume that data lies in a region $S = \text{supp}(p)$ around data manifold D, in data coordinates x^α

$$S_x = \left\{ x_D + x_\delta \,\middle|\, x_D \in D_x,\ x_\delta^\alpha \in \left(-\frac{\delta}{2}, \frac{\delta}{2}\right) \right\}$$

with $\delta \ll 1$.

- Define normal coordinates y^μ in a neighborhood of D

Gradient ascent in y-coordinates

- By choosing $\{n_i\}$ orthogonal wrt γ, the inverse induced metric takes the form

$$\gamma^{\mu\nu}(y) = \begin{pmatrix} \gamma_D^{-1}(y) & & & \\ & \gamma_{\perp_1}^{-1} & & \\ & & \ddots & \\ & & & \gamma_{\perp_{N_X - N_D}}^{-1} \end{pmatrix}^{\mu\nu} .$$

- The gradient ascent update $g^\alpha(z^{(i+1)}) = g^\alpha(z^{(i)}) + \lambda\, \gamma^{\alpha\beta} \frac{\partial f_t}{\partial x^\beta} + \mathcal{O}(\lambda^2)$ becomes

$$\gamma^{\alpha\beta} \frac{\partial f_t}{\partial x^\beta} = \frac{\partial x^\alpha}{\partial y_\parallel^\mu} \gamma_D^{\mu\nu} \frac{\partial f_t}{\partial y_\parallel^\nu} + \frac{\partial x^\alpha}{\partial y_\perp^i} \gamma_{\perp_i}^{-1} \frac{\partial f_t}{\partial y_\perp^i}$$

- For $\gamma_{\perp_i}^{-1} \longrightarrow 0$ and $\frac{\partial x}{\partial y_\perp}$ bounded we have

$$\gamma^{\alpha\beta} \frac{\partial f_t}{\partial x^\beta} \longrightarrow \frac{\partial x^\alpha}{\partial y_\parallel^\mu} \gamma_D^{\mu\nu} \frac{\partial f_t}{\partial y_\parallel^\nu}$$

  and hence the update step points along the data manifold.

  $\Rightarrow$ ***In this case, obtain counterfactuals, not adversarial examples!***

The induced metric for well-trained generative models

Theorem (Diffeomorphic Counterfactuals)

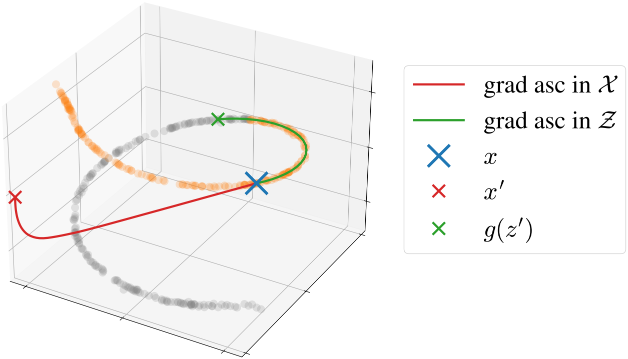
For $\epsilon \in (0, 1)$ and g a normalizing flow with Kullback–Leibler divergence $\mathrm{KL}(p, q) < \epsilon$,

$$\gamma_{\perp_i}^{-1} \longrightarrow 0 \qquad \text{as} \qquad \delta \longrightarrow 0$$

for all $i \in \{1, \ldots, N_X - N_D\}$.

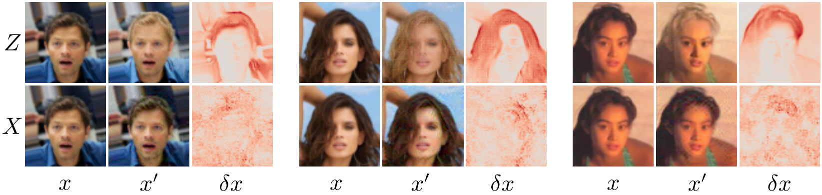
⇒ For well-trained generative models, the gradient ascent update in Z stays on the data manifold

Toy example



Diffeomorphic Counterfactuals with CelebA

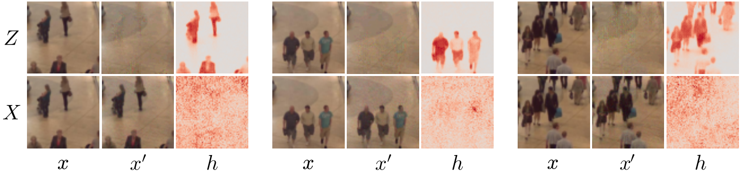
- Classifier: Binary CNN trained on ***blonde/not blonde*** attribute (test accuracy: 94%)
- Flow: Glow [Kingma et al., NeurIPS 2018]
- Task: Change classification from ***not blonde*** to ***blonde***



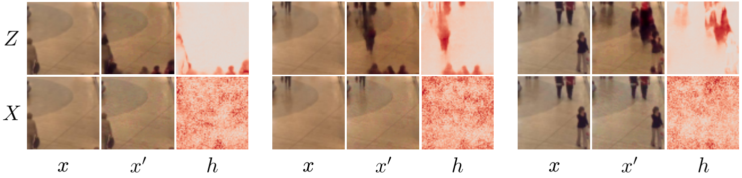
- Top row: Counterfactual computed in base space
- Bottom row: Adersarial example computed in data space

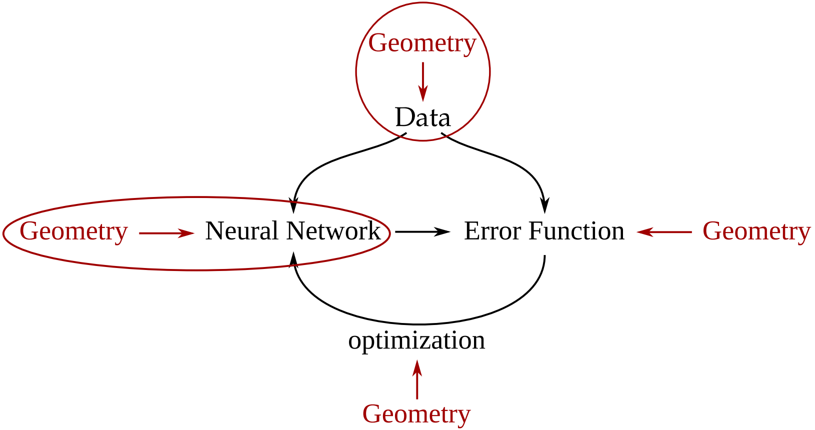
Diffeomorphic Counterfactuals for regression

- Consider crowd-counting dataset of mall images
- Count number of people in the image [Ribera et al., CVPR 2019]
- Flow: Glow [Kingma et al., NeurIPS 2018]
- Optimize for low number of people



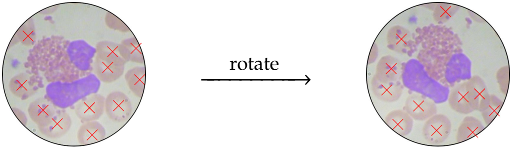
- Optimize for high number of people




Geometry
Data
Geometry
Neural Network
Error Function
Geometry
optimization
Geometry

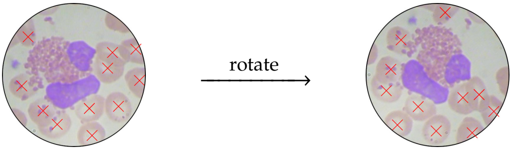
Symmetric learning problems

- Many machine learning problems have inherent symmetry, e.g.

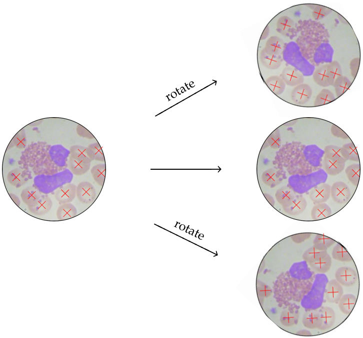


Symmetric learning problems

- Many machine learning problems have inherent symmetry, e.g.



- Naive approach: ***data augmentation***



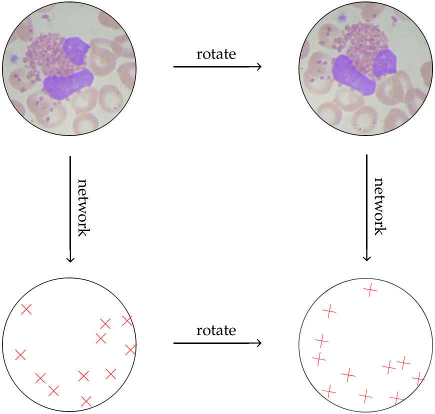
Equivariance

[Cohen, Welling, 2016]

- Equivariant neural networks build symmetry group G into network architecture:

$$f_\theta(\rho^{-1}(g)x) = \sigma(g) f_\theta(x)\,, \quad g \in G$$

e.g.



- Requires specialized architectures → ***equivariant neural networks***

Spherical Convolutions

[Cohen et al. 2018]

- For $G = \mathrm{SO}(3)$, $H = \mathrm{SO}(2)$, so $G/H = S^2$, explicit implementations are available
- Consider data $f : S^2 \to \mathbb{R}^n$ on the sphere, e.g. pictures from fisheye cameras

Spherical Convolutions

[Cohen et al. 2018]

- For $G = \mathrm{SO}(3)$, $H = \mathrm{SO}(2)$, so $G/H = S^2$, explicit implementations are available
- Consider data $f : S^2 \to \mathbb{R}^n$ on the sphere, e.g. pictures from fisheye cameras
- In the first layer, the group convolution becomes

$$(\kappa * f)(R) = \int_{S^2} \mathrm{d}x \, \kappa(R^{-1}x) f(x)$$

- In subsequent layers, the group convolution is

$$(\psi * g)(S) = \int_{\mathrm{SO}(3)} \mathrm{d}R \, \psi(R^{-1}S) g(R)$$

Spherical Convolutions

[Cohen et al. 2018]

- For $G = \mathrm{SO}(3)$, $H = \mathrm{SO}(2)$, so $G/H = S^2$, explicit implementations are available
- Consider data $f : S^2 \to \mathbb{R}^n$ on the sphere, e.g. pictures from fisheye cameras
- In the first layer, the group convolution becomes

$$(\kappa * f)(R) = \int_{S^2} \mathrm{d}x \, \kappa(R^{-1}x) f(x)$$

- In subsequent layers, the group convolution is

$$(\psi * g)(S) = \int_{\mathrm{SO}(3)} \mathrm{d}R \, \psi(R^{-1}S) g(R)$$

- Both convolutions are in practice computed in the Fourier domain: On S^2 use spherical harmonics $Y^l_m(x)$, on SO(3) use Wigner matrices $\mathcal{D}^l_{mn}(R)$, leverage FFT
- The convolutions become pointwise multiplications

$$(\kappa * f)^l_{mn} = \kappa^l_m f^l_n \qquad (\psi * g)^l_{mn} = \sum_{k=-l}^{l} f^l_{mk} \psi^l_{kn}$$

Spherical Convolutions

[Cohen et al. 2018]

- For $G = \mathrm{SO}(3)$, $H = \mathrm{SO}(2)$, so $G/H = S^2$, explicit implementations are available
- Consider data $f : S^2 \to \mathbb{R}^n$ on the sphere, e.g. pictures from fisheye cameras
- In the first layer, the group convolution becomes

$$(\kappa * f)(R) = \int_{S^2} \mathrm{d}x\, \kappa(R^{-1}x) f(x)$$

- In subsequent layers, the group convolution is

$$(\psi * g)(S) = \int_{\mathrm{SO}(3)} \mathrm{d}R\, \psi(R^{-1}S) g(R)$$

- Both convolutions are in practice computed in the Fourier domain: On S^2 use spherical harmonics $Y^l_m(x)$, on SO(3) use Wigner matrices $\mathcal{D}^l_{mn}(R)$, leverage FFT
- The convolutions become pointwise multiplications

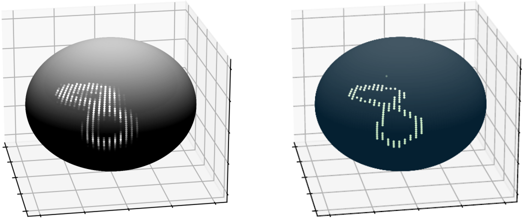
$$(\kappa * f)^l_{mn} = \kappa^l_m f^l_n \qquad (\psi * g)^l_{mn} = \sum_{k=-l}^{l} f^l_{mk} \psi^l_{kn}$$

- After final convolution integrate over SO(3) for invariance (output in $\mathbb{R}^c$) and over SO(2) for equivariance (output on S^2)

Equivariant networks vs. data augmentation

[Gerken et al. 2022]

- To compare equivariance and data augmentation, consider semantic segmentation of MNIST projected onto the sphere and randomly rotated:

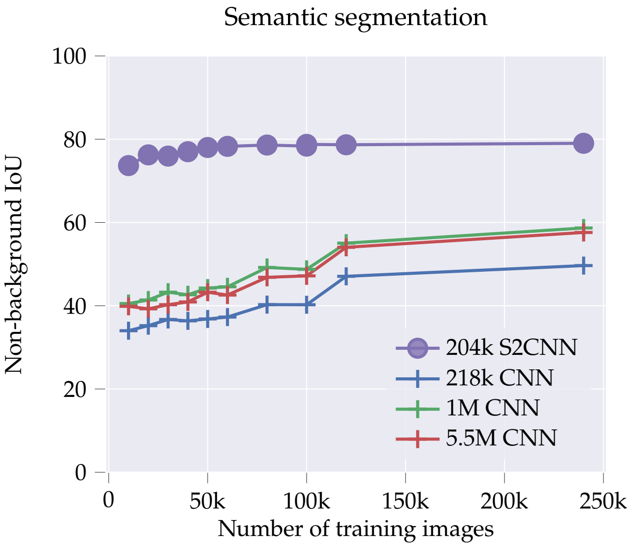


- For data augmentation, add rotated versions of projected MNIST digits

Equivariant networks vs. data augmentation

[Gerken et al. 2022]

- In this case, performance of equivariant networks cannot be reached by data augmentation



Equivariant neural networks

Rich theory:

- Very general concepts, can be specialized to sets, graphs, grids, manifolds, . . .
- Connections to many topics in pure mathematics: [Gerken et al. 2021]
  - representation theory
  - group theory
  - harmonic analysis
  - graph theory
  - fiber bundles
  - gauge theory
- Ongoing project about robustness of equivariant networks

Equivariant neural networks

Rich theory:

- Very general concepts, can be specialized to sets, graphs, grids, manifolds, ...
- Connections to many topics in pure mathematics: [Gerken et al. 2021]
  - representation theory
  - group theory
  - harmonic analysis
  - graph theory
  - fiber bundles
  - gauge theory
- Ongoing project about robustness of equivariant networks

Important applications:

- Used in all kinds of applications:
  Protein folding, Quantum Field Theory, Cosmology, Neuroscience, ...
- Ongoing project about learning topological invariants in condensed matter physics
- Ongoing project about application to fisheye-camera images for autonomous driving

Conclusions

- Deep learning is a transformative technology lacking a strong theoretical basis

Conclusions

- Deep learning is a transformative technology lacking a strong theoretical basis
- Geometry can help in several key parts of the learning process, bringing abstract mathematics to practical applications

Conclusions

- Deep learning is a transformative technology lacking a strong theoretical basis
- Geometry can help in several key parts of the learning process, bringing abstract mathematics to practical applications
  - Training data can be studied using differential geometry, yielding powerful insights

Conclusions

- Deep learning is a transformative technology lacking a strong theoretical basis
- Geometry can help in several key parts of the learning process, bringing abstract mathematics to practical applications
  - Training data can be studied using differential geometry, yielding powerful insights
  - Equivariant neural networks exploit symmetries of the learning problem

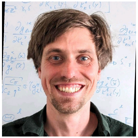
Conclusions

- Deep learning is a transformative technology lacking a strong theoretical basis
- Geometry can help in several key parts of the learning process, bringing abstract mathematics to practical applications
  - Training data can be studied using differential geometry, yielding powerful insights
  - Equivariant neural networks exploit symmetries of the learning problem
- New areas of mathematics enter the study of neural networks

Collaborators


Daniel Persson


Klaus-Robert Müller


Pan Kessel


Christoffer Petersson


Fredrik Ohlsson


Hampus Linander


Ann-Kathrin Dombrowsik


Oscar Carlsson


Jimmy Aronsson

Thank you!

Appendix

Neural tangent kernels

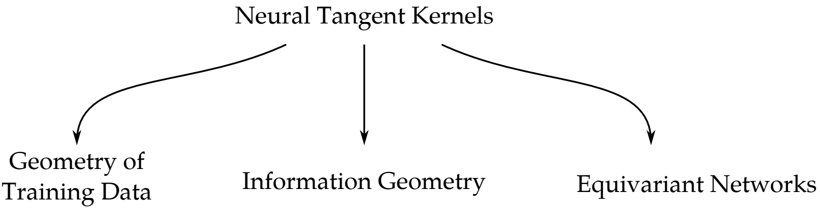
[Jacot, Gabriel, Hongla, 2018]

- Analytical description of infinite-width neural networks via Gaussian Processes
- Gives access to training dynamics
- Powerful tool to study geometric deep learning

Neural tangent kernels

[Jacot, Gabriel, Hongla, 2018]

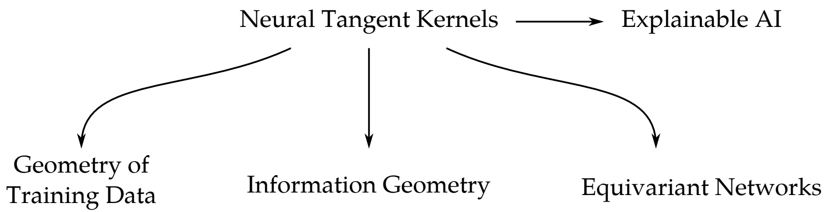
- Analytical description of infinite-width neural networks via Gaussian Processes
- Gives access to training dynamics
- Powerful tool to study geometric deep learning



Neural tangent kernels

[Jacot, Gabriel, Hongla, 2018]

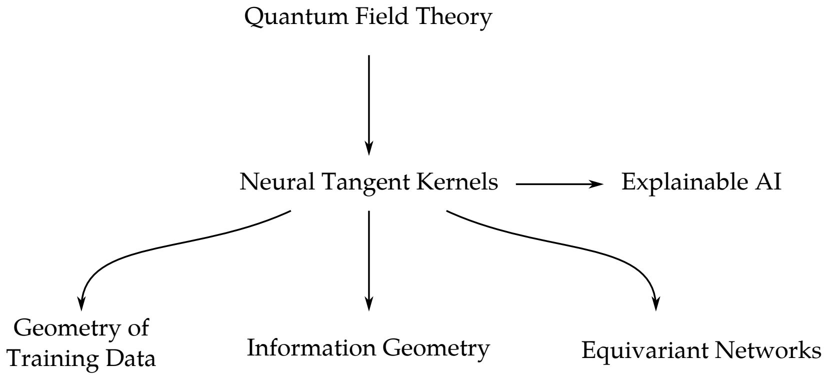
- Analytical description of infinite-width neural networks via Gaussian Processes
- Gives access to training dynamics
- Powerful tool to study geometric deep learning



Neural tangent kernels

[Jacot, Gabriel, Hongla, 2018]

- Analytical description of infinite-width neural networks via Gaussian Processes
- Gives access to training dynamics
- Powerful tool to study geometric deep learning



Novel mathematics from deep learning research

- Deep learning can be used to aide research in pure math

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Open Access | Published: 01 December 2021

Advancing mathematics by guiding human intuition with AI

Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis & Pushmeet Kohli

Nature **600**, 70–74 (2021) | Cite this article

THE SIGNATURE AND CUSP GEOMETRY OF HYPERBOLIC KNOTS

ALEX DAVIES, ANDRÁS JUHÁSZ, MARC LACKENBY, AND NENAD TOMASEV

Abstract. We introduce a new real-valued invariant called the natural slope of a hyperbolic knot in the 3-sphere, which is defined in terms of its cusp geometry. We show that twice the knot signature and the natural slope differ by at most a constant times the hyperbolic volume divided by the cube of the injectivity radius. This inequality was discovered using machine learning to detect relationships between various knot invariants. It has applications to Dehn surgery and to 4-ball genus. We also show a refined version of the inequality where the upper bound is a linear function of the volume, and the slope is corrected by terms corresponding to short geodesics that link the knot an odd number of times.

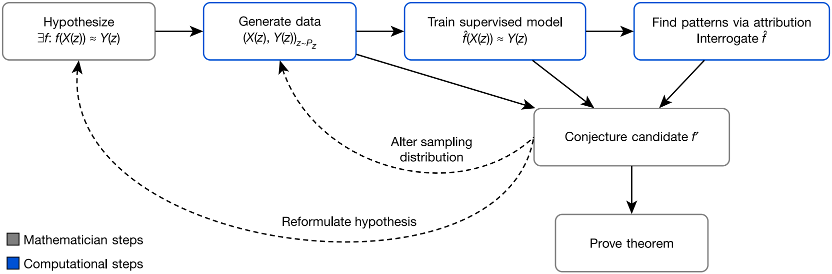
- Simplifying polylogs using reinforcement learning [Dersy, Schwartz, Zhang, 2022]

- Exciting area that is still in its infancy

New mathematics from deep learning

[Davies et al. 2021]

Process proposed in Davies et al. 2021:



Example:

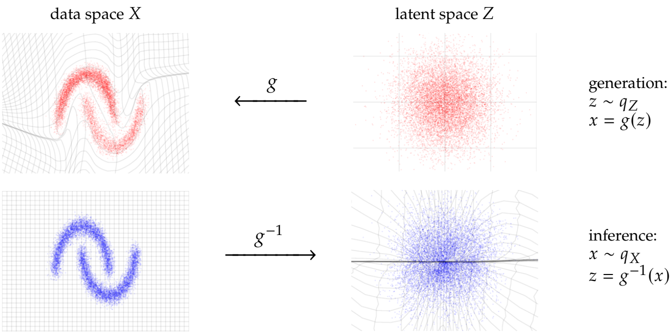
- Let z be a complex polyhedron
- Let $X(z) \in \mathbb{Z}^2 \times \mathbb{R}^2$ be the number of edges and vertices of z as well as volume and surface area
- Let $Y(z)$ be the number of faces of z
- Euler's formula $\Rightarrow X(z) \cdot (-1, 1, 0, 0) + 2 = Y(z)$
- Note: Deep learning approach also works for highly non-linear functions f

Normalizing flows

- Generative model g which maps base space Z to data space X ***bijectively***, i.e. it is a ***diffeomorphism***
- Probability distribution q_Z in Z is simple, e.g. uniform or normal
- Probability distribution q_X in X is given by change of variables

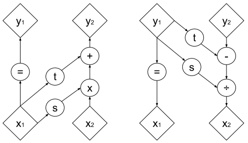
$$q_X(x) = q_Z(g^{-1}(x)) \left| \det \frac{\partial z}{\partial x} \right|$$

- Train by maximizing log-likelihood $\log q_X$ of train data



- g is realized as a neural network with bijective building blocks
- Network needs to be easily invertible and have a tractable Jacobian determinant
- RealNVP uses ***affine coupling layers***

$$
\begin{aligned}
y_{1:d} &= x_{1:d} \\
y_{d+1:D} &= x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d}) \\
s, t &: \mathbb{R}^d \rightarrow \mathbb{R}^{D-d} \qquad \text{(deep CNNs)}
\end{aligned}
$$

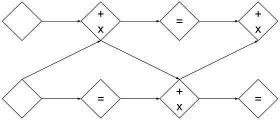


forward backward

- The Jacobian is given by

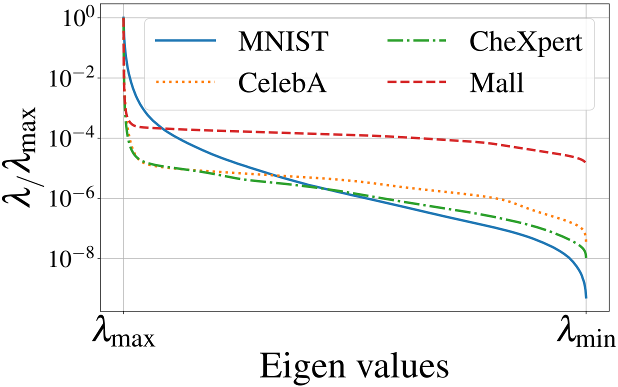
$$
\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{1}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x^T_{1:d}} & \text{diag}\left(\exp(s(x_{1:d}))\right) \end{bmatrix} \quad \Rightarrow \quad \left|\frac{\partial y}{\partial x^T}\right| = \exp\left(\sum_j s(x_{1:d})_j\right)
$$

- Alternate the parts which are modified from layer to layer



- RealNVP uses multi-scale architecture

Eigenvalue spectrum of Jacobian



Gauge networks

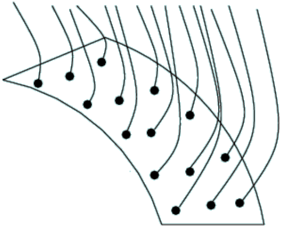
[Cheng et al. 2019]

- Gauge networks: neural networks which are invariant with respect to local changes of coordinates

Can formulate gauge networks in terms of fiber bundles

Fiber bundles (reminder)

- A bundle consists of total space E, base $\mathcal{M}$ and projection $\pi : E \to \mathcal{M}$
- Fibers are given by $E_x = \pi^{-1}(x)$
- A section of E is a map $\sigma : \mathcal{M} \to E$ such that $\pi \circ \sigma = \text{id}$



Gauge networks

[Cheng et al. 2019]
[Gerken et al. 2021]

Fiber bundles (reminder)

- A bundle consists of total space E, base $\mathcal{M}$ and projection $\pi : E \to \mathcal{M}$
- Fibers are given by $E_x = \pi^{-1}(x)$
- A section of E is a map $\sigma : \mathcal{M} \to E$ such that $\pi \circ \sigma = \mathrm{id}$

- Formulate gauge symmetry in terms of principle G-bundle P over input manifold $\mathcal{M}$: Bundle P with regular right action of G on P

$$\triangleleft : P \times G \longrightarrow P\,, \quad \text{satisfying} \quad \pi_P(p \triangleleft g) = \pi_P(p)$$

- A ***gauge*** is a section of P
- In general relativity: P is frame bundle, section of P is choice of basis in $\mathcal{T}\mathcal{M}$
- Let V be a vector space on which G acts from the left via representation ρ

$$g \triangleright v = \rho(g)v$$

- Define equivalence relation on $P \times V$ by

$$(p, v) \sim_\rho (p \triangleleft g, g^{-1} \triangleright v)\,, \quad g \in G$$

- Feature maps are sections of associated bundle $P \times_\rho V = P \times V / \sim_\rho$ with projection

$$\pi_\rho([p, v]) = \pi_P(p)$$

Gauge networks

[Cheng et al. 2019]
[Gerken et al. 2021]

Can construct explicit coordinate independent convolution by using

- Exponential map
- Parallel transport by means of connection on P

For a section s of $P \times_\rho V$, the convolution is given by

$$(\Phi s)(x) = \int_{B_R} \mathrm{d}X\, \kappa(x, X) s|_{\exp_x X}(x) \sqrt{\det(g_{\mathcal{M}})}$$

with a kernel

$$\kappa : \mathcal{M} \times \mathcal{T}\mathcal{M} \to \mathrm{Hom}(E_\rho, E_\eta)\,,$$

satisfiying

$$\kappa(x, k \triangleright X) = \eta(k^{-1})\kappa(x, X)\rho(k)$$

- Few concrete implementations of these concepts yet [Cohen et al. 2019] [de Haan et al. 2020]
- Gauge equivariant convolutions also exist for *internal* gauge symmetries, used for lattice field theory computations [Favoni et al. 2020] [Luo et al. 2020]

Gauge networks

[Cheng et al. 2019]
[Gerken et al. 2021]

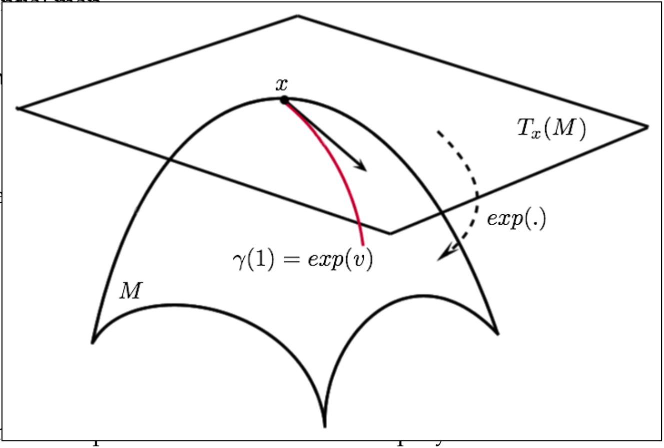
Can construct explicit coordinate independent convolution by using

- Exponential map
- Parallel

For a section

with a kerne

satisfiying



- Few co

[Cohen et al. 2019]
[de Haan et al. 2020]

- Gauge equivariant convolutions also exist for *internal* gauge symmetries, used for lattice field theory computations

[Favoni et al. 2020]
[Luo et al. 2020]

Gauge networks

[Cheng et al. 2019]
[Gerken et al. 2021]

Can construct explicit coordinate independent convolution by using

- Exponential map
- Parallel transport by means of connection on P

For a section s of $P \times_\rho V$, the convolution is given by

$$(\Phi s)(x) = \int_{B_R} \mathrm{d}X \, \kappa(x, X) s|_{\exp_x X}(x) \sqrt{\det(g_{\mathcal{M}})}$$

with a kernel

$$\kappa : \mathcal{M} \times \mathcal{T}\mathcal{M} \to \mathrm{Hom}(E_\rho, E_\eta) \,,$$

satisfiying

$$\kappa(x, k \triangleright X) = \eta(k^{-1}) \kappa(x, X) \rho(k)$$

- Few concrete implementations of these concepts yet [Cohen et al. 2019] [de Haan et al. 2020]
- Gauge equivariant convolutions also exist for *internal* gauge symmetries, used for lattice field theory computations [Favoni et al. 2020] [Luo et al. 2020]