AI Pictures at a Mathematical Exhibition: How Applied Harmonic Analysis meets Machine Learning

Radu Balan

Department of Mathematics and Norbert Wiener Center for Harmonic Analysis and Applications
University of Maryland, College Park, MD

June 30, 2023
University of Torino, Turin, Italy

Acknowledgments

This material is based upon work partially supported by the National Science Foundation under grant no. DMS-2108900 and Simons Foundation. "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation."

Papers available online at:
https://www.math.umd.edu/ rvbalan/

Table of Contents:

High-Level Overview

In this series of lectures, we discuss a few harmonic analysis techniques and problems applied to machine learning.

1. NN: Neural networks (NN) and their universal approximation property.
2. Lipschitz analysis: we provide rationals for studying Lipschitz properties of NNs, and then we perform a Lipschitz analysis of these networks. We focus on two aspects of this analysis: stochastic modelng of local vs. global analysis, and a scattering network inspired Lipschitz analysis of convolutive networks.
3. Invariance and Equivariance: We highlight the duality between invariance and covariance/equivariance, with focus on G-invariant representations.
4. Applications to data analysis and modeling: We present applications on a variety of problems: classification and regression on graphs; generative models for data sets; neural network based modeling of time-evolution of dynamical systems; discrete optimizatons.

Table of Contents

Graph Deep Learning Applications

Based on joint works with:
Naveed Haghani (UMD,APL-JHU)
Maneesh Singh (Verisk, Comcast)

N. Haghani, M. Singh, R. Balan, "Graph Regressing and Classification using Permutation Invariant Representations", AAAI-GCLR March 2022
R. Balan, N. Haghani, M. Singh, "Permutation Invariant Representations with Applications to Graph Deep Learning", arXiv preprint: 2203.07546 [math.FA], [cs.LG]

For this part of the talk, two applications performed on two graph data sets, with two different tasks: classification (on a protein data set), and regression (on the QM9 chemical compund data set).

The Protein Dataset

The Enzyme Classification Problem

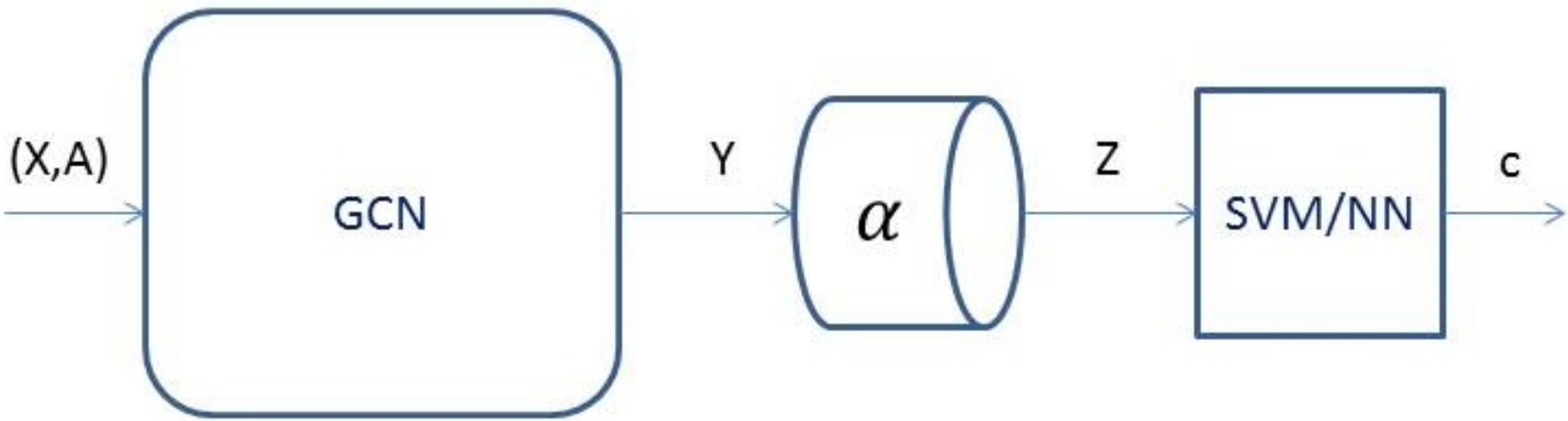
Protein Dataset: 663 non-enzymes and 450 enzymes out of 1113 proteins. Each graph associated to one protein: nodes represent amino acids and edges represent the bonds between them. Number of nodes (aminoacids): varying between 20 and 620 with average of 39. Input feature vectors os size $r = 29$.

Task: the task is classification of each protein into *enzyme* or *non-enzyme*.

The Deep Network Architecture

Architecture: ReLU activation and

- GCN with $L = 3$ layers and 29 input feature vectors, and 50 hidden nodes in each layer; no dropouts, no batch normalization. output of GCN: $d = 1, 10, 50, 100$.
- Mid-layer component: α
- Fully connected NN with dense 3-layers and 150 internal units; no dropouts, with batch normalization.



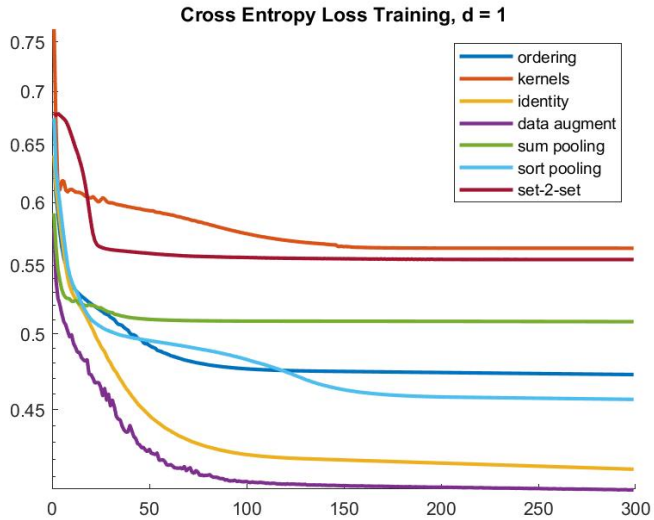
The Network

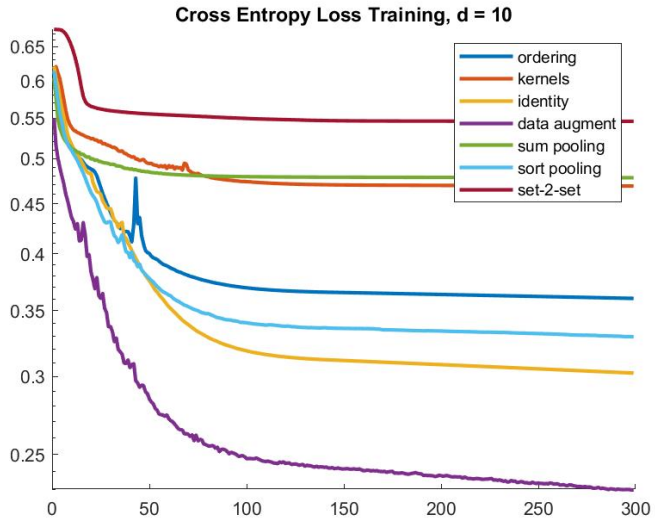
Training has been done over 300 epochs with a batch size of 128. Loss function: binary cross-entropy.
The following 7 α modules have been tested:

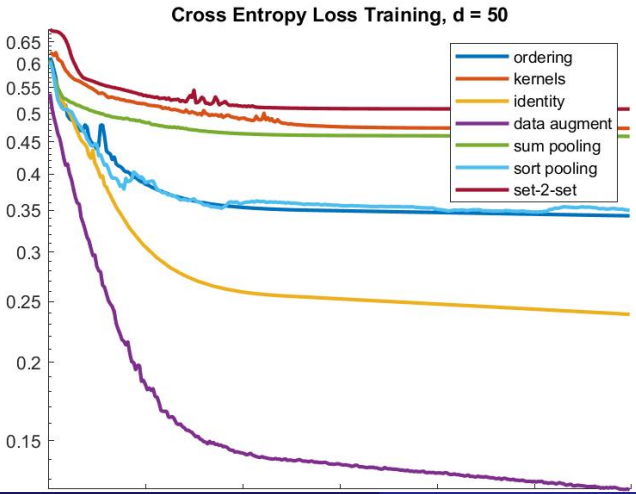
1. identity: $\alpha(X) = X$; no permutation invariance.
2. data augmentation: $\alpha(X) = X$ BUT the training data set has been augmented with 4 random permutatons of each graph.
3. ordering: $\alpha(X) =\downarrow (XA)$, $A = [I\ 1]$
4. kernels: $\alpha(X) = (\sum_{k=1}^{n} exp(-\|x_k - a_j\|^2))_{1 \leq j \leq m=5nd}$
5. sumpooling: $\alpha(X) = 1^T X$
6. sort-pooling: sorted by last column
7. set-to-set: introduced in [Vinyals&al.'16]

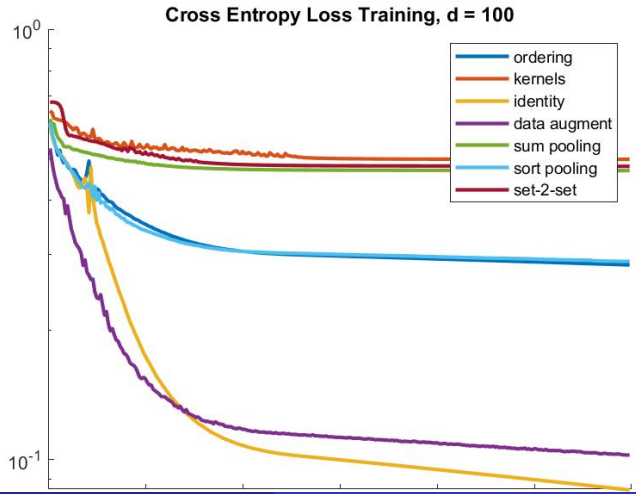
Enzyme Classification Example

Training Loss: X Entropy



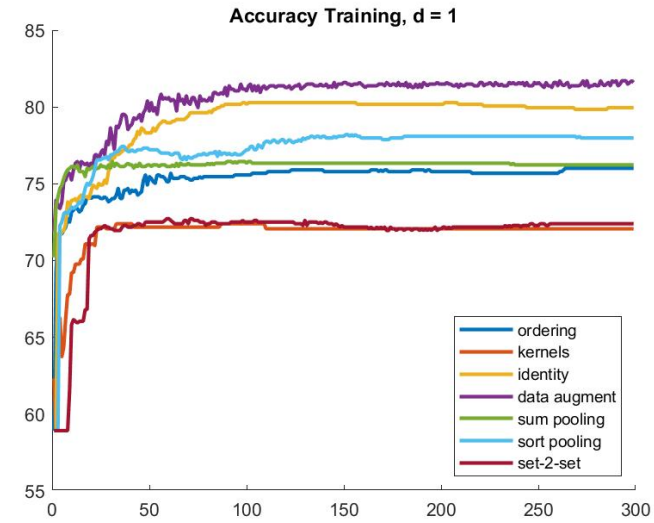


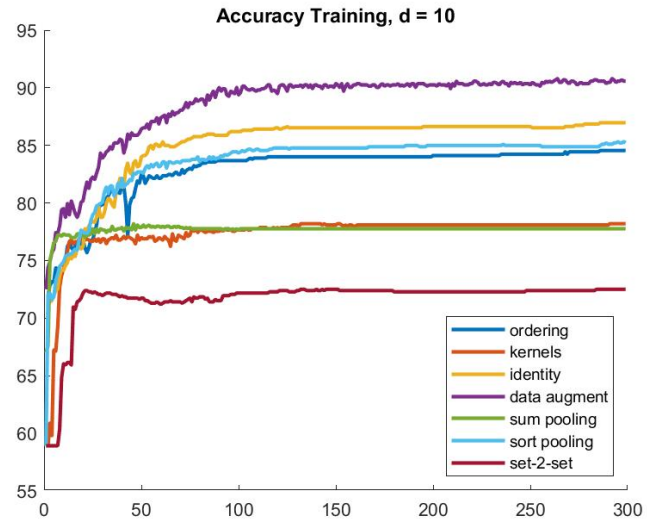


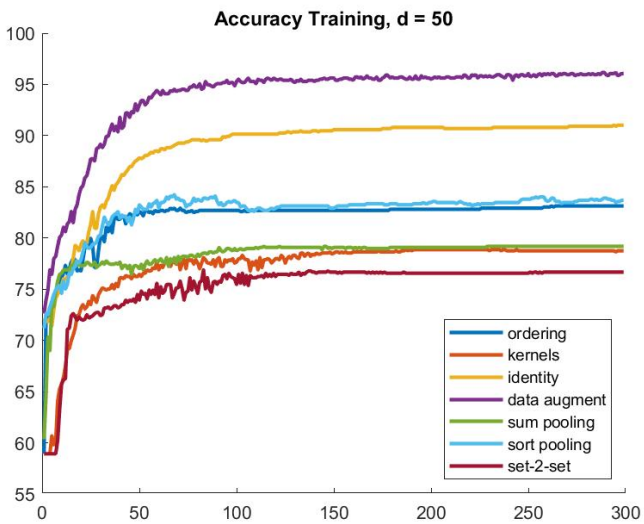


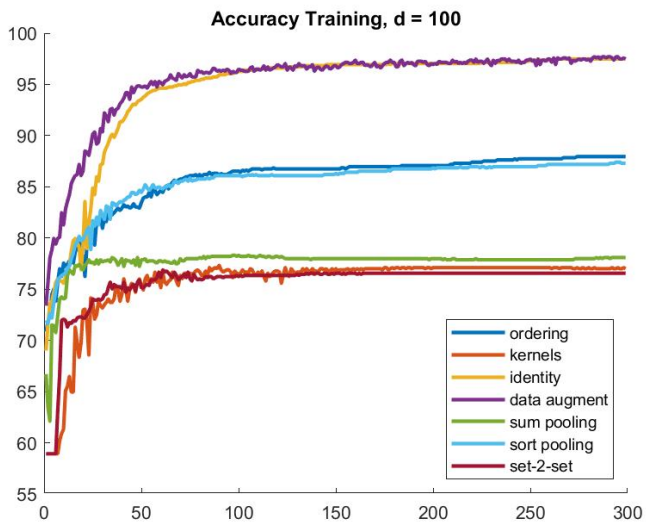
Enzyme Classification Example

Accuracy on Training set



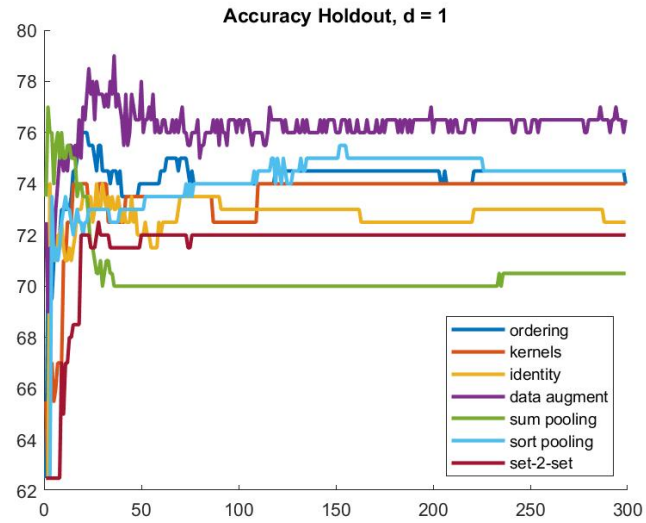


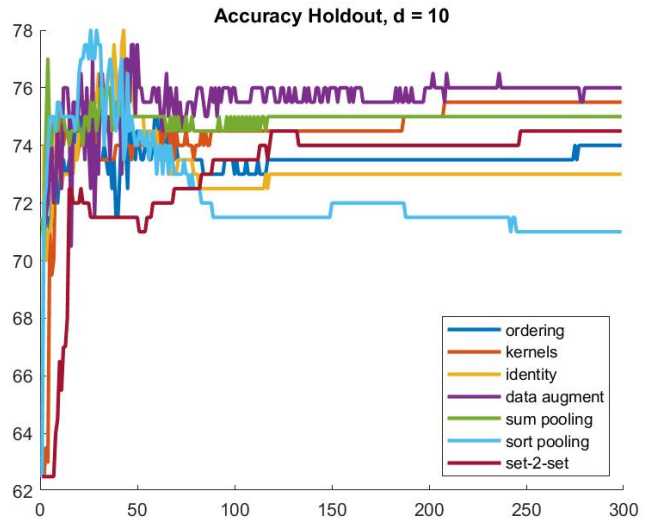


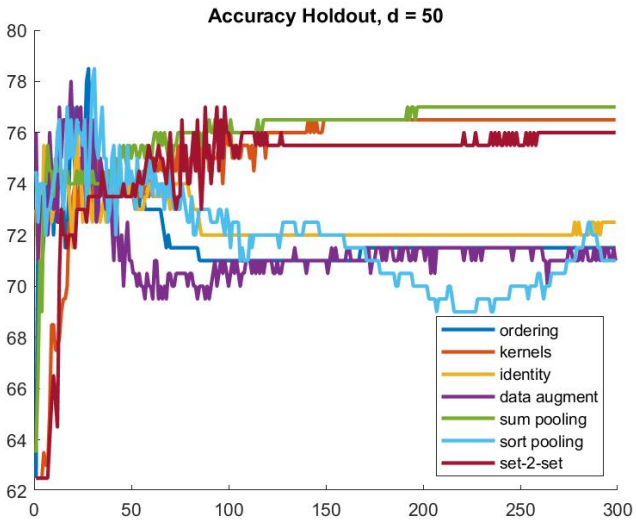


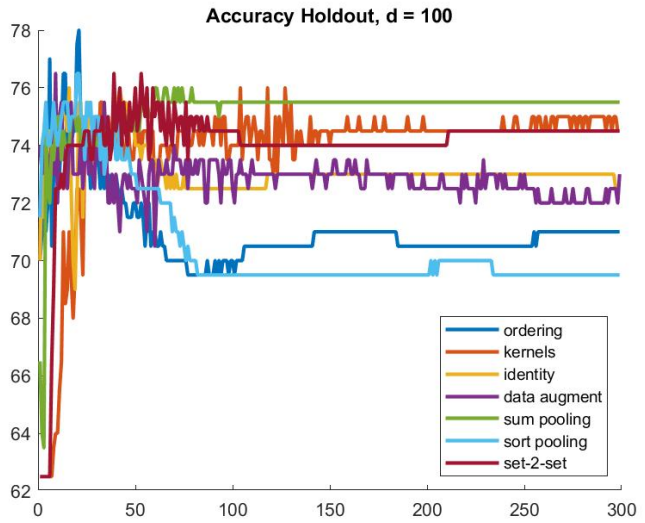
Enzyme Classification Example

Accuracy on Holdout data



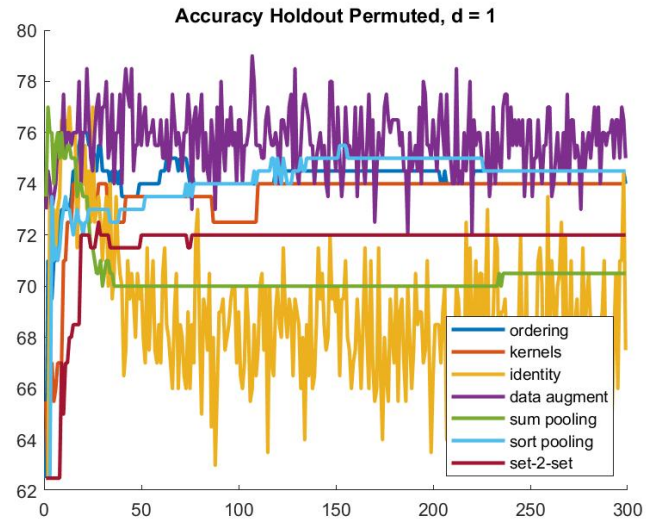


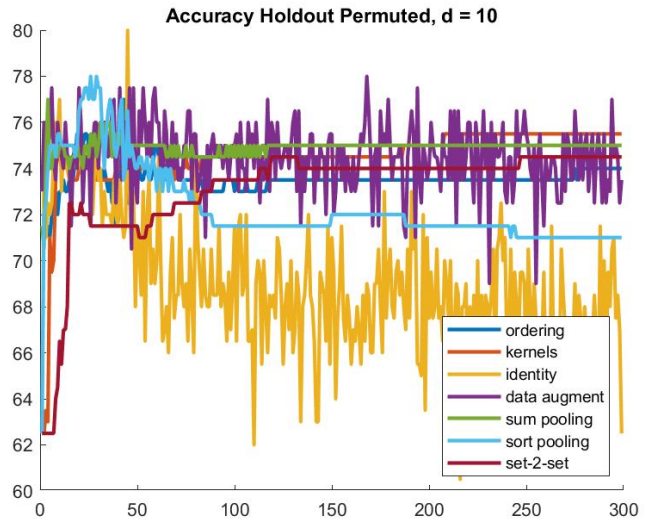


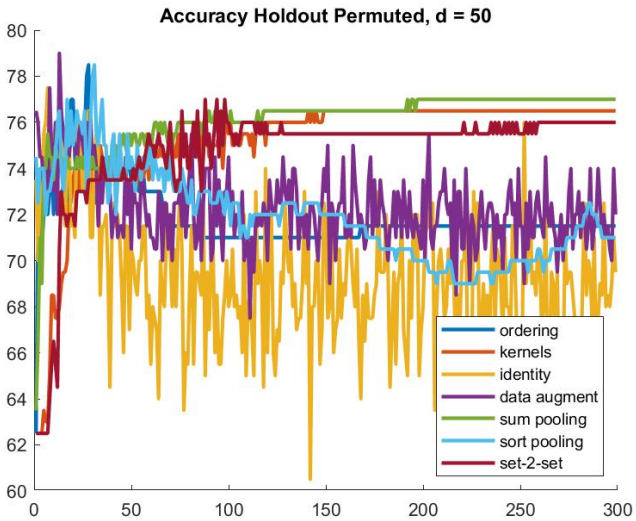


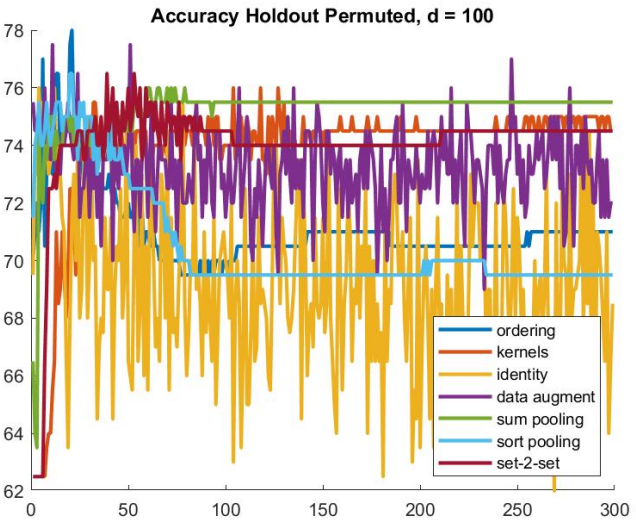
Enzyme Classification Example

Accuracy on Holdout data with nodes randomly permuted









Performance Results: Accuracy

d = 50	ordering	kernels	identity	data augment	sum-pooling	sort-pooling	set-2-set
Training	83.1	78.8	91	96	79.2	83.7	76.7
Holdout	71.5	76.5	72.5	71	77	71	76
Holdout Perm	71.5	76.5	69.5	72	77	71	76

Table: Accuracy ACC(%) for enzyme/non-enzyme classification of the seven algorithms on PROTEINS_FULL dataset after 300 epochs for embedding dimension $d = 50$

For comparison: [Dobson&al.] obtain an accuracy of 77-80% using an SVM based classifier.

The QM9 Dataset

Dataset: Consists of about 134,000 isomers of organic molecules made up of CHONF, each containing 10-29 atoms. see http://quantum-machine.org/datasets/ Nodes corresponds to atoms; each feature vector containins geometry (x,y,z coordinates), partial charge per atom (Mulliken charge), and atom type.
Task: the task is regression: predict a physical feature (electron energy gap $\Delta\varepsilon$) computed for each molecule.
Architecture: ReLU activation and

- GCN with $L = 3$ layers and 50 hidden nodes in each layer; no dropouts, no batch normalization; zero padding to $m = 29$ number of rows. output of GCN: $d = 1, 10, 50, 100$.
- Mid-layer component: α
- Fully connected NN with dense 3-layers and 150 internal units in each of the two hidden layers; no dropouts, with batch normalization.

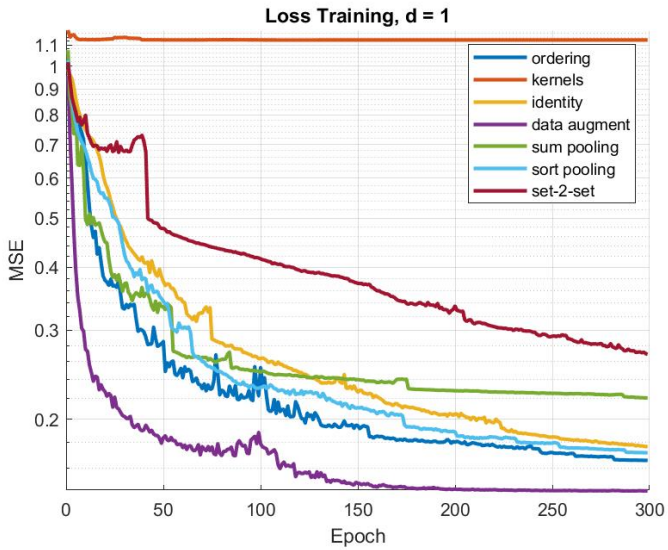
The Network

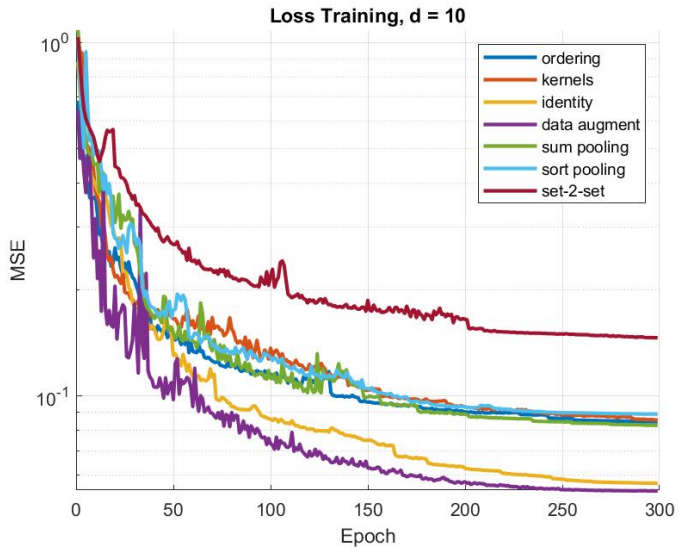
Training has been done over 300 epochs with a batch size of 128. Loss function: Mean-Square Error (MSE).
The same 7 α modules have been tested:

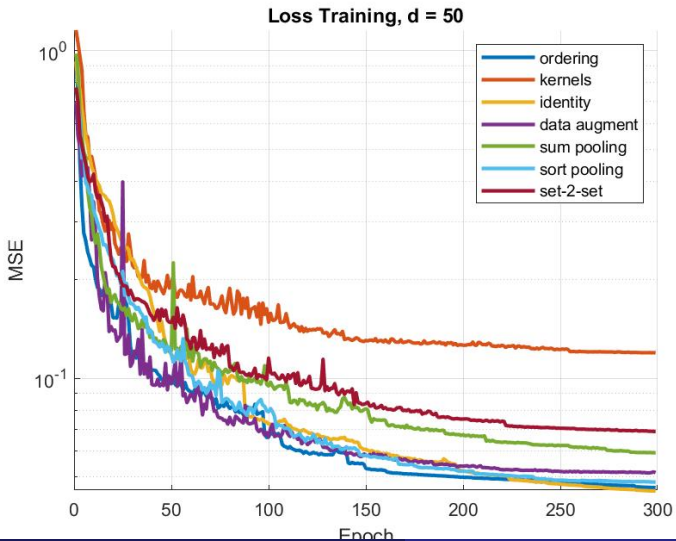
1. identity: $\alpha(X) = X$; no permutation invariance.
2. data augmentation: $\alpha(X) = X$ BUT the training data set has been augmented with 4 random permutatons of each graph.
3. ordering: $\alpha(X) =\downarrow (XA)$, $A = [I\ 1]$
4. kernels: $\alpha(X) = (\sum_{k=1}^{n} exp(-\|x_k - a_j\|^2))_{1\leq j\leq m=5nd}$
5. sumpooling: $\alpha(X) = 1^T X$
6. sort-pooling: sorted by last column
7. set-to-set: introduced in [Vinyals&al.'16]

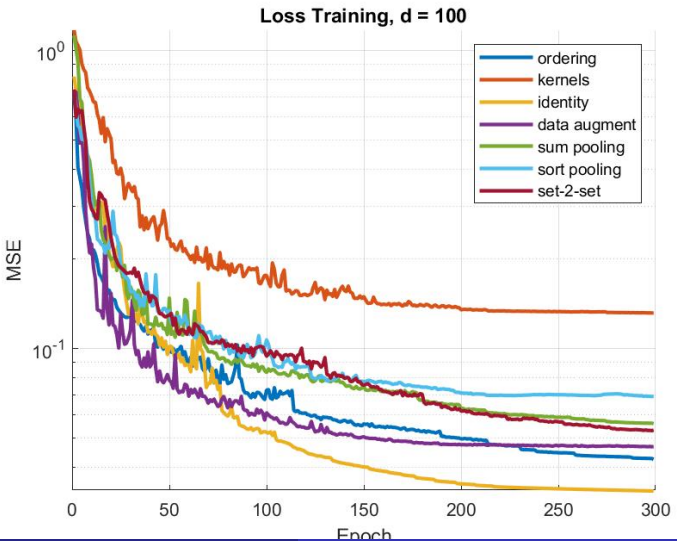
QM9 Regression Example

Training MSE



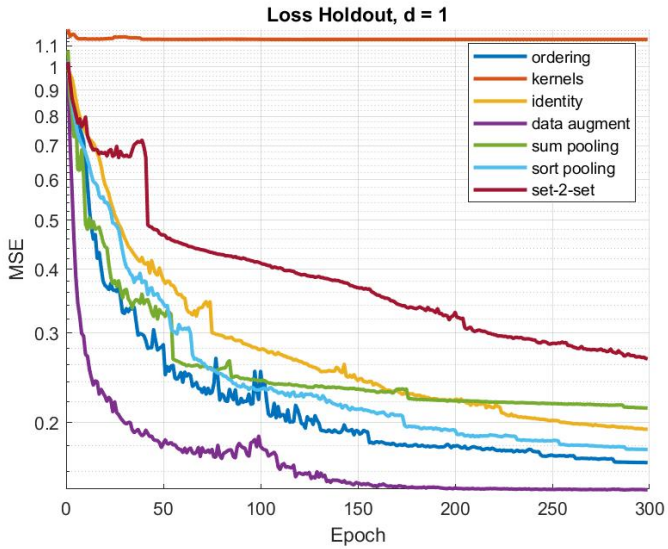


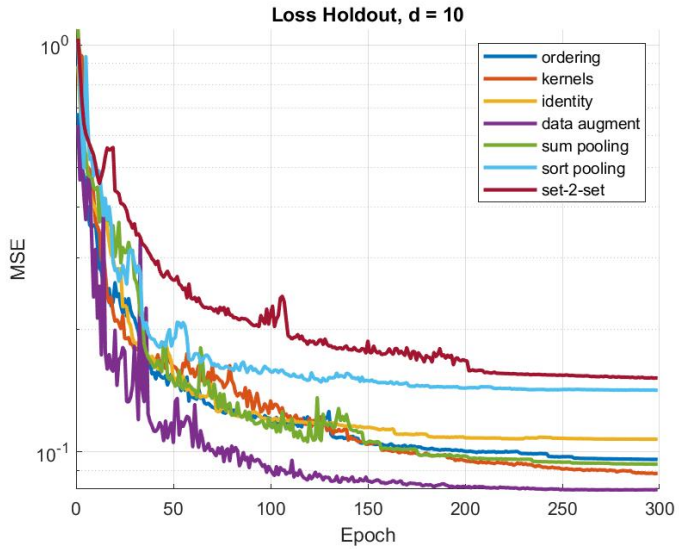


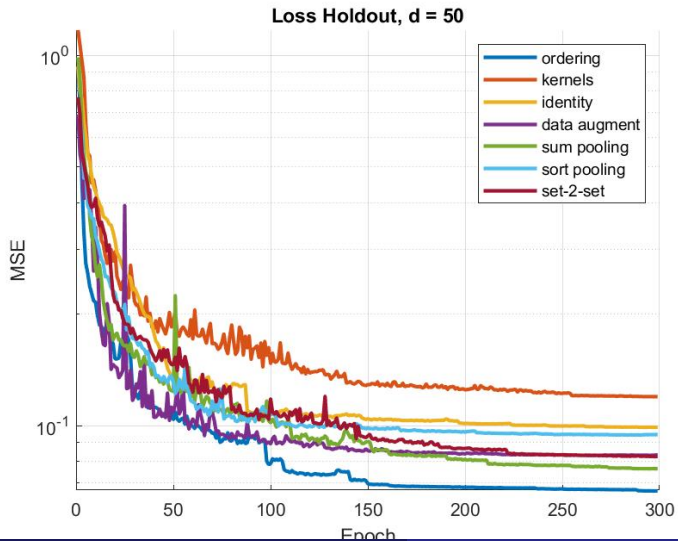


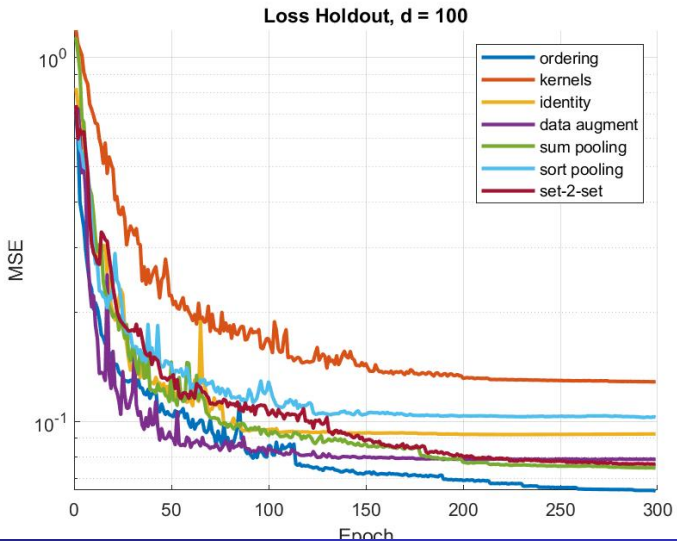
QM9 Regression Example

Validation MSE



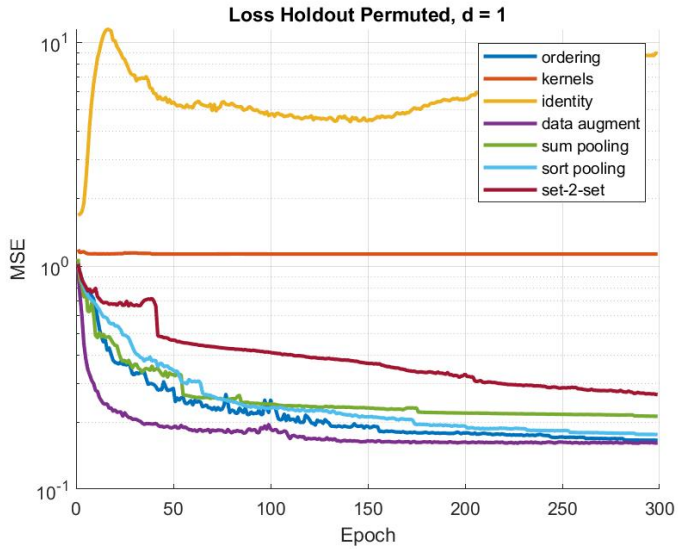


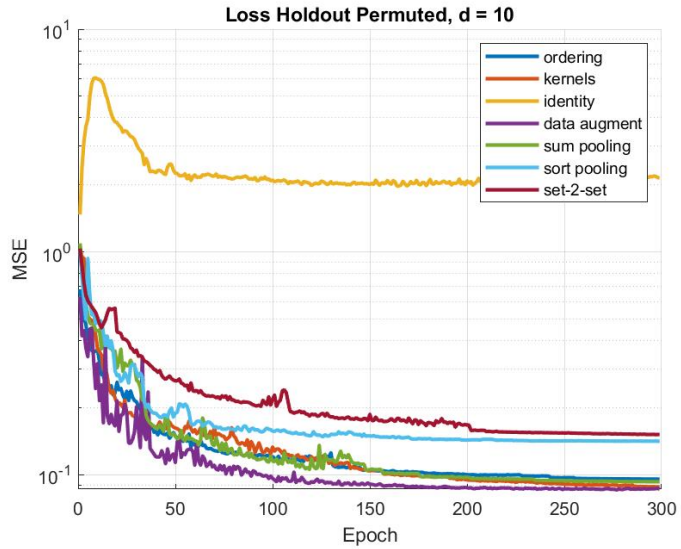


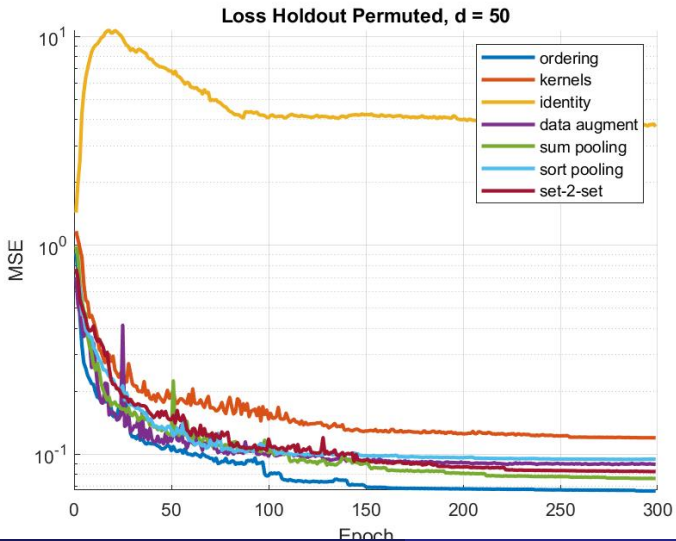


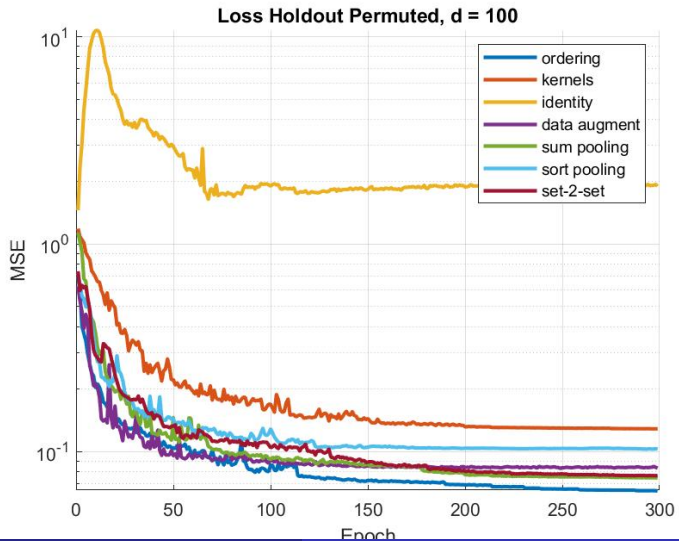
QM9 Regression Example

Validation MSE with Random Permutations









Performance Results: MAE

d = 100	ordering	kernels	identity	data augment	sum-pooling	sort-pooling	set-2-set
Training	0.155	0.269	0.139	0.164	0.178	0.199	0.173
Holdout	0.187	0.267	0.227	0.206	0.201	0.239	0.201
Holdout Perm	0.187	0.267	1.086	0.213	0.201	0.239	0.201

Table: Mean Absolute Error (MAE) for regression of the electron energy gap $\Delta\varepsilon = LUMO - HOMO$ (eV) of the seven algorithms on QM9 dataset after 300 epochs for embedding dimension $d = 100$

For comparison:

- chemical accuracy is $0.043eV$
- the best ML method [Gilmer&al.'17] achieves MAE of $0.053eV$
- Coulomb method [Rupp&al.'12] achieves MAE of $0.229eV$

Table of Contents

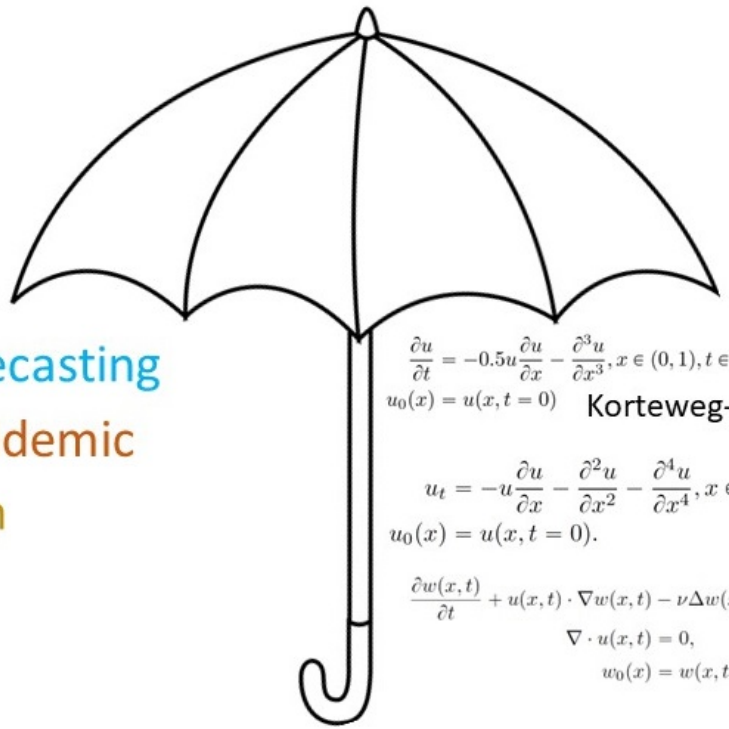
Modeling Deterministic and Stochastic Evolution Operators using Deep Networks

Collaborators:
USC: Paul Bogdan, Gaurav Gupta, Xiongye Xiao, Ruochen Yang

Joint work:
G. Gupta, X. Xiao, R. Balan, P. Bogdan, "Non-Linear Operator Approximations for Initial Value Problems", Proceedings of ICLR 2022,

Problem Formulation

A fundamental problem in machine learning: predict future states using current conditions, $x_0 \in \mathbb{R}^s \mapsto x_T = \Phi(x_0) \in \mathbb{R}^s$.
Examples: Solutions of PDEs , Epidemic Forcasting (COVID19)



Dynamical systems, Forecasting
Kermack-McKendrik- epidemic
Hodgkin-Huxley - neuron

$$\frac{\partial u}{\partial t} = -0.5u\frac{\partial u}{\partial x} - \frac{\partial^3 u}{\partial x^3}, x \in (0,1), t \in (0,1]$$
$$u_0(x) = u(x, t = 0)$$
Korteweg-de Vries

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + v\frac{\partial^2 u}{\partial x^2}, x \in (0, 2\pi), t \in (0,1]$$
$$u_0(x) = u(x, t = 0).$$
Burgers'

$$u_t = -u\frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4}, x \in (0,1), t \in (0,1]$$
$$u_0(x) = u(x, t = 0).$$
Kuramoto-Sivashinsky

$$\frac{\partial w(x,t)}{\partial t} + u(x,t) \cdot \nabla w(x,t) - \nu \Delta w(x,t) = f(x), \quad x \in (0,1)^2, t \in (0,T]$$
$$\nabla \cdot u(x,t) = 0, \quad x \in (0,1)^2, t \in [0,T]$$
$$w_0(x) = w(x, t = 0), \quad x \in (0,1)^2$$
Navier-Stokes

Our problem: How to implement Φ using a Deep Network and a Training data set?

Existing Approaches

- UAP based NN: Use of (Conv.) N.N. (Guo, 2016), (Grohs,2019);
- UAP with Reduced Basis/PCA: Galerkin-like schemes (Santo,2019), sparse networks (Boelcskei,Kutyniok, 2019);
- IVP defined NN: Physics-inspired neural networks (PINNs): (Raissi, Karniadakis, 2019), (Wang, Perdikaris, 2021);
- Reservoir Computing: (Schrauwen, 2007), (Girvan, Hunt, 2020);
- Neural Operators: Data-driven and input-resolution independent: Fourier Neural Op. (FNO) (Li,2020), Graph Nystrom sampling (Li, 2020), Multi Wavelet Transform (MWT) (Gupta, 2021);
- ...

Architecture

As special type of Neural Operator, is the exponential operator seen as the evolution operator of a linear (time-invariant) differential equation:

Equation	Solution
$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}, u(t=0) = \mathbf{u}_0$ (Linear ODE)	$\mathbf{u}(t=\tau) = e^{t\mathbf{A}}\mathbf{u}_0$
$u_t = \frac{\partial^2 u}{\partial x^2}, u(x,0) = u_0(x)$ (Heat equation)	$u(x,\tau) = e^{\tau \frac{\partial^2}{\partial x^2}} u_0(x)$
$u_t = \mathcal{L}u + \mathcal{N}f(u), u(x,0) = u_0(x)$	$u(x,\tau) = e^{\tau\mathcal{L}}u_0(x) + \int_0^\tau e^{(\tau-t)\mathcal{L}}\mathcal{N}f(u(x,t))dt$

Approach: Learn operator $\mathcal{L}$ while implementing a (nonlinear) version of $e^{\mathcal{L}}$.

Architecture

As special type of Neural Operator, is the exponential operator seen as the evolution operator of a linear (time-invariant) differential equation:

Equation	Solution
$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u}, u(t=0) = \mathbf{u}_0$ (Linear ODE)	$\mathbf{u}(t=\tau) = e^{t\mathbf{A}}\mathbf{u}_0$
$u_t = \frac{\partial^2 u}{\partial x^2}, u(x,0) = u_0(x)$ (Heat equation)	$u(x,\tau) = e^{\tau \frac{\partial^2}{\partial x^2}} u_0(x)$
$u_t = \mathcal{L}u + \mathcal{N}f(u), u(x,0) = u_0(x)$	$u(x,\tau) = e^{\tau\mathcal{L}}u_0(x) + \int_0^\tau e^{(\tau-t)\mathcal{L}}\mathcal{N}f(u(x,t))dt$

Approach: Learn operator $\mathcal{L}$ while implementing a (nonlinear) version of $e^{\mathcal{L}}$.

Performance metrics:

1. Approximation error MSE (for training), MAE (for testing);
2. Model complexity, expressed by number of parameters to be learned;

Architecture (2)

The exponential operator $\mathcal{L} \mapsto e^{\mathcal{L}}$ has been used in deep learning:

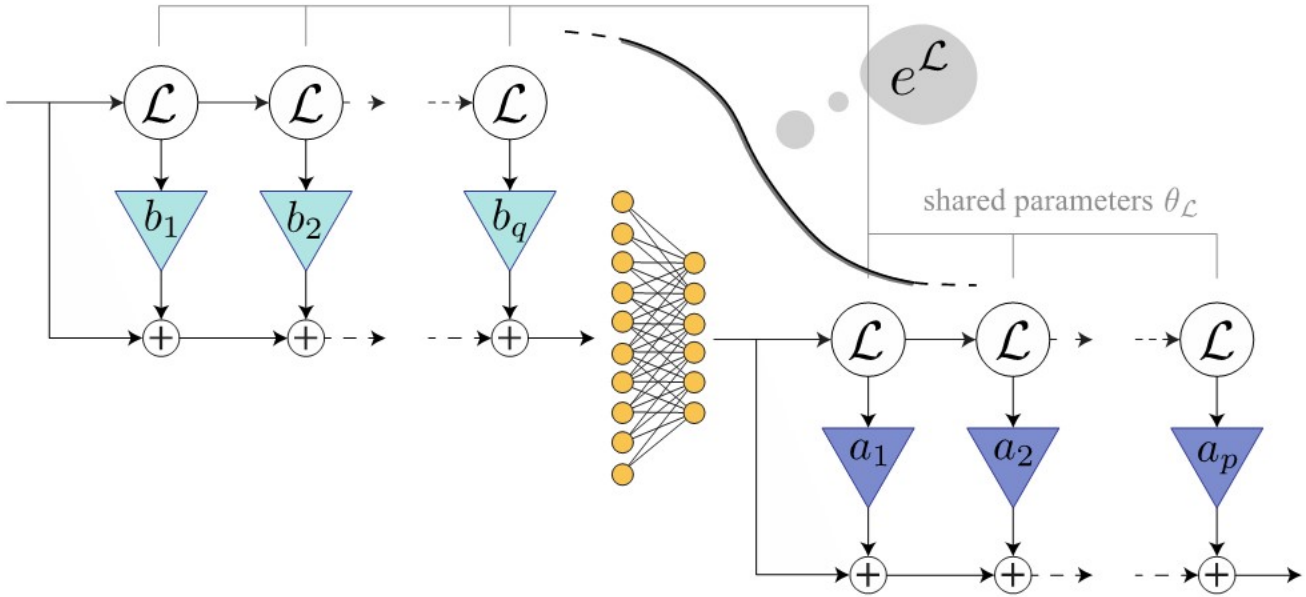
1. exponential function used to model NN non-linearity (Andoni, 2014);
2. Taylor polynomial as a truncation of the Taylor series (Hoogeboom, 2020) - particularly in the context of convolutive operators, and (Sylvester) normalizing flow;
3. Our first contribution: Use Padé approximation as a more compact polynomial form than the Taylor polynomial. **Padé Neural Operator**.

Padé Approximation of the exponential function, $x \mapsto e^x$ is denoted by $[p/q] = \frac{A_{pq}(x)}{B_{pq}(x)}$, with $p, q \geq 0$ integers and:

$$e^x \approx [p/q] := \frac{\sum_{j=0}^{p} a_j x^j}{\sum_{j=0}^{q} b_j x^j}, a_j = \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!}, b_j = (-1)^j \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!}$$

Architecture (3)

- Padé Neural Operator $[p/q]e^{\mathcal{L}}$ with a single layer nonlinear block:



- Our second contribution: Decompose $\mathcal{L}$ using a Multi-Wavelet basis (Gupta, 2021), $e^{\mathcal{L}} = \sum_{i=1}^{L} \left(Q_i e^{\mathcal{L}} Q_i + Q_i e^{\mathcal{L}} P_i + P_i e^{\mathcal{L}} Q_i\right) + P_L e^{\mathcal{L}} P_L$. Then apply the Padé Neural Operator for each term of this decomposition:

$$\Phi(x_0) = \sum_{i=1}^{L} \left(Q_i [p/q] e^{A_i} Q_i + Q_i [p/q] e^{B_i} P_i + P_i [p/q] e^{C_i} Q_i\right) + P_L [p/q] e^{\mathcal{L}_L} P_L$$

Overall we obtain: the **Multiwavelet Padé Exponential Model**.

Lipschitz Analysis of the Padé Neural Operator

Theorem

Given a linear operator $\mathcal{L} = \mathcal{L}(\theta_{\mathcal{L}})$ (or, a Lipschitz operator with Lipschitz constant $\|\mathcal{L}\|$ and $\mathcal{L}(0) = 0$), a non-linearity layer $v = \sigma(Wu + b)$, and $p, q \in \mathbb{N}$, at points of differentiability, the gradients of the operation $x \mapsto y = F(x; \theta_{\mathcal{L}}, W, b) := [p/q]e^{\mathcal{L}}(x)$ using the $[p/q]$ Padé neural operator are bounded in operator norm by

$$\left\| \frac{\partial y}{\partial \theta_{\mathcal{L}}} \right\| \leq \exp(\|\mathcal{L}\|) \left(\|b\|_2 + \|W\| \|x\|_2 \right) \left(\sum_{j=1}^{n_\theta} \left\| \frac{\partial \mathcal{L}}{\partial \theta_j} \right\|^2 \right)^{1/2}, \tag{2.1}$$

$$\left\| \frac{\partial y}{\partial W} \right\| \leq \exp(\|\mathcal{L}\|) \|x\|_2, \tag{2.2}$$

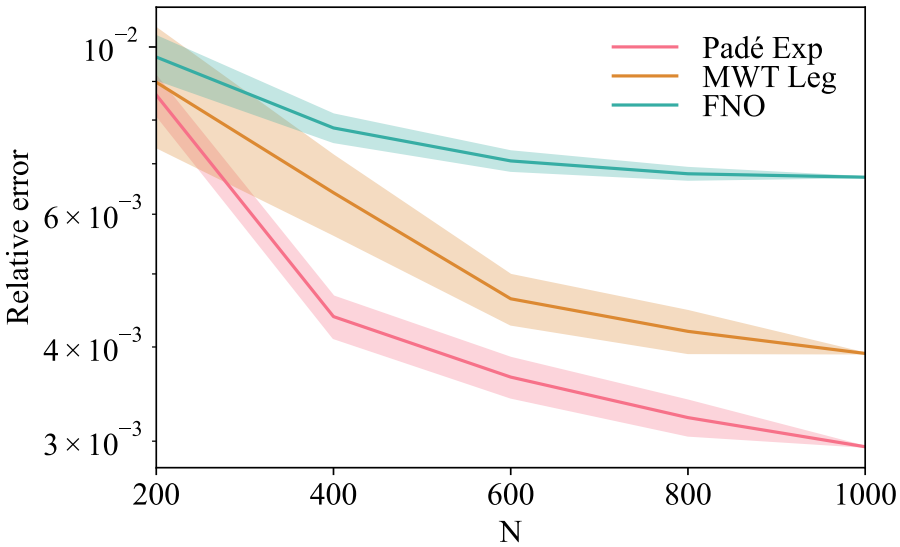
$$\left\| \frac{\partial y}{\partial b} \right\| \leq \exp\left(\frac{p}{p+q} \|\mathcal{L}\| \right). \tag{2.3}$$

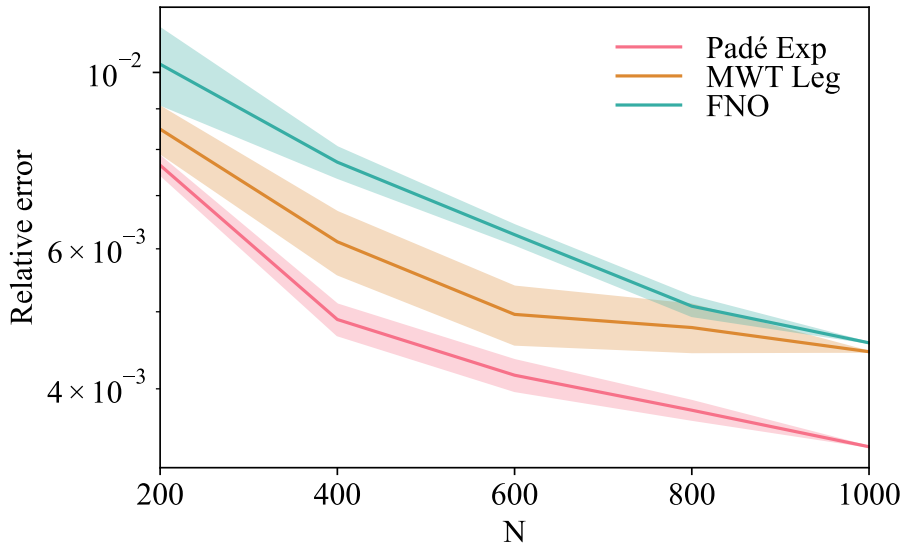
Remarks: The polynomials $A_{pq}(\mathcal{L})$ and $B_{pq}(\mathcal{L})$ are implemented as recurrent networks. This theorem guarantees the gradient does not explode with $p, q \to \infty$.

Testing on PDEs

Data Efficiency

How fast the training error decays ("data efficiency") w.r.t. number N of training samples, for Korteweg - de Vries (KdV, left), and Kuramoto-Shivashinski (SV, right):





Number of training samples N vs performance (relative L2 error) for neural operators evaluated on the KdV equation with s=1024. For $N < 1000$, each smaller dataset is sampled uniformly randomly 5 times from the complete dataset ($N = 1000$) and mean $\pm$ std.dev (shaded region) results are shown across the sampling experiments. (Right) Same analysis for KS equation with s=1024.

Testing on PDEs

Sensitivity to Input Resolution

Korteweg-de Vries (KdV) equation benchmarks for different input resolution s. The relative L2 errors are shown for each model.

Networks	s = 64	s = 128	s = 256	s = 512	s = 1024
Padé Exp	**0.00301**	**0.00308**	**0.00311**	**0.00298**	**0.00295**
MWT Leg	0.00372	0.00369	0.00391	0.00408	0.00392
FNO	0.00663	0.00676	0.00657	0.00649	0.00672
MGNO	0.12507	0.13610	0.13664	0.15042	0.13666
LNO	0.04234	0.04764	0.04303	0.04465	0.04549
GNO	0.13826	0.12768	0.13570	0.13616	0.12521

GNO: Graph Neural Operator (Li, 2020); *MGNO*: Multi-level version of GNO (Li, 2020); *LNO*: low-rank representation of the integral operator kernel, à la DeepONet (Lu,2020); *FNO*: Fourier Neural Operator (Li, 2020); *MWT Leg*: MWT with Legendre OPs;

Epidemic Forecasting (COVID19)

Problem Specifications

Epidemic Forecasting is an example where the dynamical system is unknown (or it may not be deterministic). Neural operators provide an entirely data-driven approach and are capable to learn PDE agnostic maps.
Dataset COVID-19 from April 12, 2020 to August 28,2021 provided by JHU. Data from 50 US states, and for each state, total counts of daily reported confirmed (C), recovered (R), and deaths (D). Data in each state is normalized by the respectie state total population. Total data: array of $50 \times 3 \times 484$ numbers.
Task: The forecasting problem is to learn the map between 14 consecutive counts (C,R,D) to next 7 days data for each of the 50 US state. Let d_t denote the 50×3 array on day t. Then the operator map can be written as:

$$T(\underbrace{d_{-14}, d_{-13}, \ldots, d_{-1}}_{u_0(x)}) = (\underbrace{d_0, d_1, \ldots, d_6}_{u(\tau,x)}).$$

Epidemic Forecasting (COVID19)

Prediction Benchmarks

COVID-19 prediction benchmarks for different networks using 10-fold resampling with mean $\pm$ std. dev. across folds. The Mean Average Error (MAE) is presented for Confirmed (C), Recovered (R), and Deaths (D) counts averaged across 7 days of prediction for 50 US states. The relative L2 error is the test error for each model. The last column compares each network vs FC (auto-regressive fully connected network) in terms of the total MAE improvement and total model parameters difference.

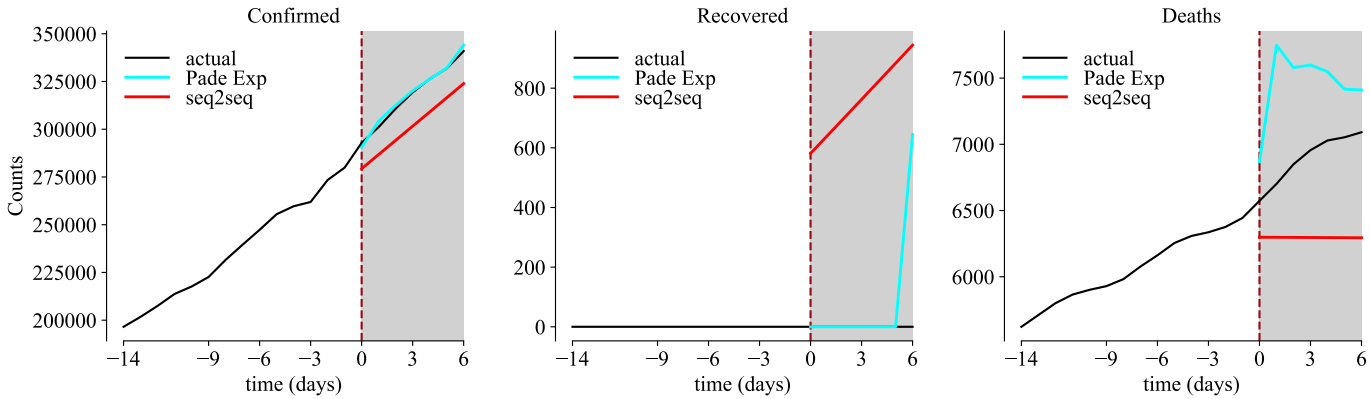
Networks	MAE			Relative L2 error	Net. vs FC
	C	R	D		
Padé Exp	**1219 $\pm$ 130**	**1752 $\pm$ 666**	**211 $\pm$ 31**	**0.0155 $\pm$ 0.0034**	82.14% (+652K)
MWT Leg	3554 $\pm$ 1157	2928 $\pm$ 1338	284 $\pm$ 209	0.0245 $\pm$ 0.0043	62.0% (+18M)
FNO 3D	4213 $\pm$ 391	3391 $\pm$ 1233	592 $\pm$ 157	0.0301 $\pm$ 0.0045	54.0% (+1.02M)
LNO 3D	28502 $\pm$ 12698	6586 $\pm$ 3442	1465 $\pm$ 965	0.1056 $\pm$ 0.0394	-105.0% (+238K)
Neural ODE	4339 $\pm$ 1174	3443 $\pm$ 1408	443 $\pm$ 192	0.0310 $\pm$ 0.0069	53.8% (+172K)
Seq2Seq	2798 $\pm$ 456	3317 $\pm$ 1690	346 $\pm$ 83	0.0273 $\pm$ 0.0058	63.7%(+1.8M)
Transformer	7087 $\pm$ 972	6613 $\pm$ 2853	1722 $\pm$ 320	0.0501 $\pm$ 0.0094	13.4% (+15.2K)
FC	10305 $\pm$ 2818	5885 $\pm$ 1609	1634 $\pm$ 686	0.0609 $\pm$ 0.0111	(37.2K)

Epidemic Forecasting (COVID19)

A Tale of Two States

COVID19 Forecasting. Confirmed, Recovered, and Deaths count forecasting results for the 07/07/20 – 07/13/20 (chosen arbitrarily) using previous 2 weeks as the input. The Padé Exp prediction and the best non-neural operator scheme from previous table (seq2seq) is shown.

California:
39.77M population.



Massachusetts:
6.89M population.

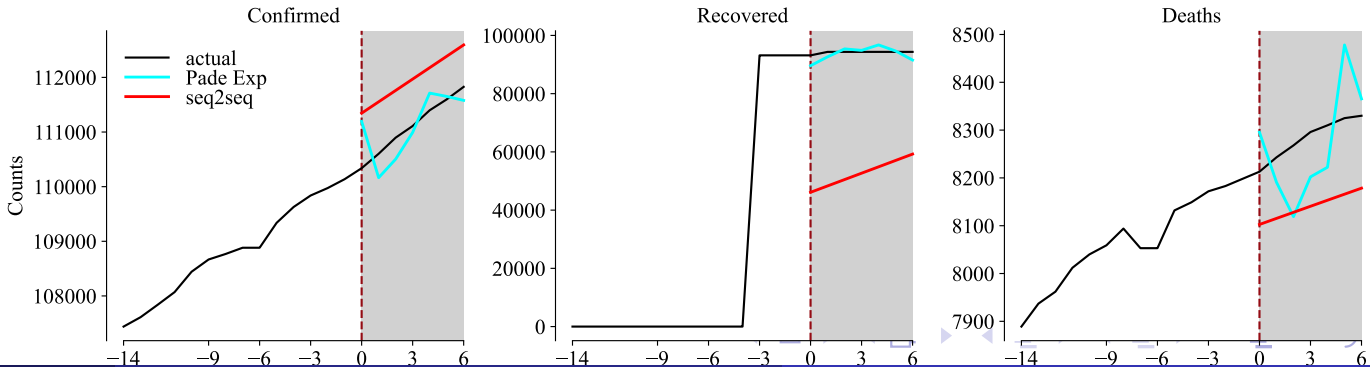


Table of Contents

Uncertainty Quantification and Propagation through DNN

Collaborators:
UMD: Danial Ludwig, Michael Rawson
UMB:Thomas Ernst, Bo Li, Xiaoke Wang, Ze Wang
Joint Works:
ISMRM 2022: Estimating Noise Propagation of Neural Network based Image Reconstruction using Automatic Differentiation

MRI Model

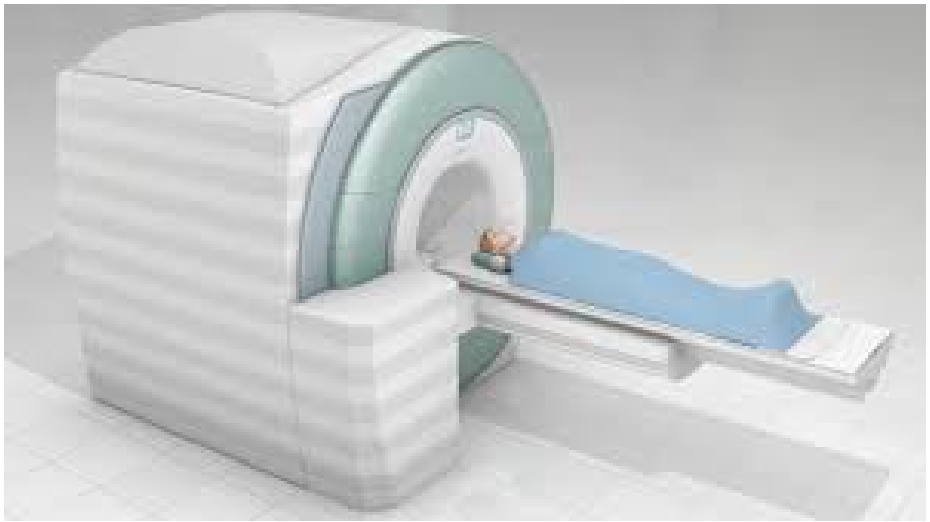


Figure: Credits: hopkinsmedicine.org

The measurement model. For coil $k \in [N_c]$,

$$x_k = \mathcal{F}(S_k z) + \nu_k$$

where $\mathcal{F}$ is the Fourier acquisition matrix, S_k is the diagonal matrix with the coil k sensitivity map, ν_k is measurement noise, and z is the brain signal.

Knowns: F, $x_1, ..., x_{N_c}$. Unknowns: $S_1, ..., S_{N_c}, \nu_1, ..., \nu_{N_c}, z$. Target: z.

Lots of research, lots of Nobel prizes, lots of companies (Siemens, GE, Philips), lots of techniques (compressive sampling, GRAPPA, SENSE, ...) to solve the inverse problem: $z = G(measurements)$.
More recent: Use of Deep Neural Networks.

The MRI Inverse Problem

At an abstract level, the forward model, $z \mapsto x$ and the reconstruction (inverse) model, $x \mapsto \hat{z}$ are:

$$x = F(z) + \nu \quad , \quad \hat{z} = G(x).$$

To fix notations: the target (brain) signal $z \in \mathbb{R}^d$, the measured (acquired) signal $x \in \mathbb{R}^n$.
The DNN approach proposes to implement G using certain Neural Network architectures. Out of many architectures out there, we focused on a specific network, namely the *end-to-end variational neural network (E2E-VarNet)* introduced by Sriram,et al, at MICCAI 2020.

Our problem: Given a trained network that implements a reconstruction algorithm G, quantify the level of uncertainty per reconstructed pixel.

Assumption: We assume the network has been trained well enough so that $G(F(z)) = z$, i.e., perfect reconstruction in the absence of noise.

Uncertainty Propagation through NN

CRLB and FIM

The standard way of quantifying uncertainty is through the Cramer-Rao Lower Bound (CRLB). The CRLB has been used many times for experimental design in Medical Imaging and elsewhere. Fisher Information Matrix $I(z)$ and $CRLB$:

$$I(z) = \mathbb{E}\left[\left(\nabla_z log(p(x;z))\right)\left(\nabla_z log(p(x;z))\right)^T\right] \quad , \quad CRLB = (I(z))^{-1}$$

Interpretation: Covariance of any *unbiased* estimator of z is lower bounded $CRLB$.

Assume further, the noise is AWGN with variance σ^2. A simple computation yields:

$$CRLB = \sigma^2 \left(J_F^T J_F\right)^{-1} \quad , \quad J_F = \left[\frac{\partial F_k}{\partial z_j}\right]_{(j,k)\in[n]\times[d]} \in \mathbb{R}^{n\times d}$$

where J_F denotes the Jacobian matrix of the forward model.

The CRLB and the Jacobian of the NN

Our main theoretical result is to connect $CRLB = (I(z))^{-1}$ and the Jacobian of G, J_G.
A simple lemma:

Lemma

Assume $A \in \mathbb{R}^{n\times d}$ is full rank with $n \geq d$.

1. *For any $B \in \mathbb{R}^{d\times n}$ such that $BA = I_d$ (i.e., a left inverse), $BB^T \geq (A^TA)^{-1}$.*
2. *If $B_0 = (A^TA)^{-1}A^T$ is the pseudo-inverse of A then, $B_0B_0^T = (A^TA)^{-1}$.*

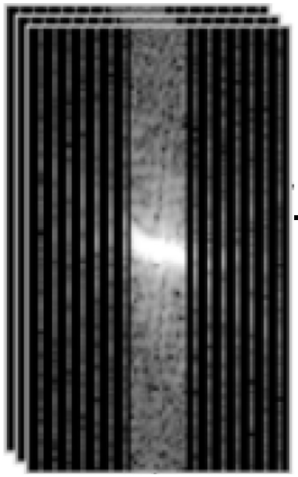
Consequence:

$$CRLB = \sigma^2 J_{G_0} J_{G_0}^T \quad , \quad G_0 = argmin_{G:G(F(z))=z} trace(J_G J_G^T)$$

Remarks:

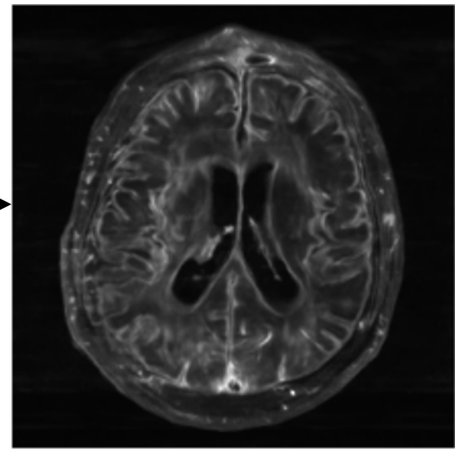
Architecture

Input k-space
fastMRI[1]



Uses U-Net for some computational steps

End-to-end
Variational Network[2]

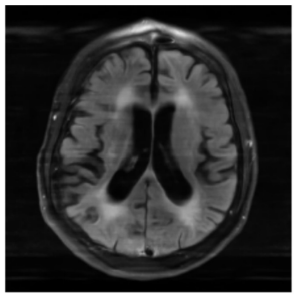


Acceleration Factor: 6~12, ACS: 24, Matrix Size: 320×320

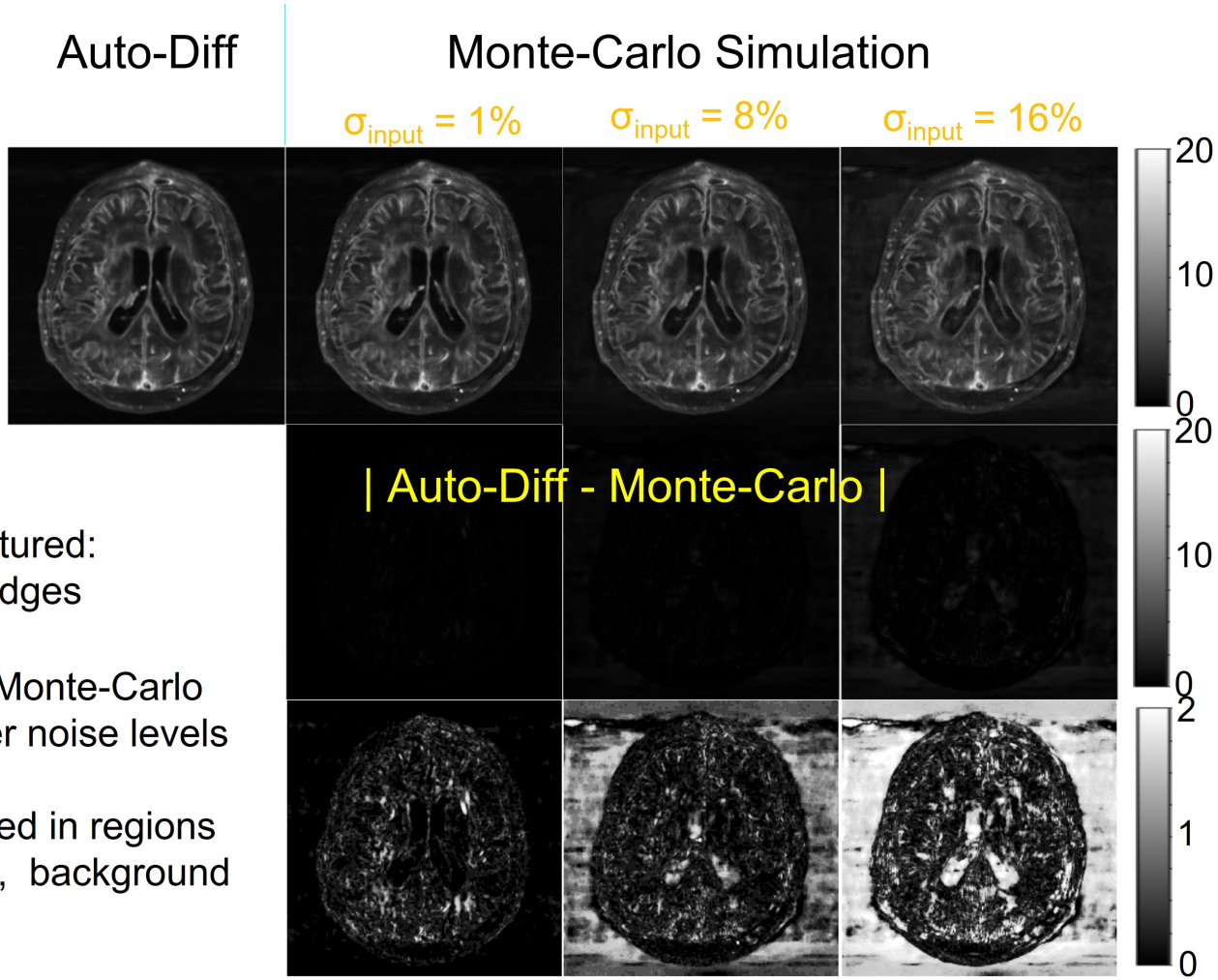
1. *Zbontar J, Knoll F, Sriram A, et al. fastMRI: An Open Dataset and Benchmarks for Accelerated MRI. Published online November 21, 2018. Accessed November 10, 2021. https://arxiv.org/abs/1811.08839v2*
2. *Sriram, Anuroop, et al. "End-to-end variational networks for accelerated MRI reconstruction." International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2020. (Facebook AI Research and NYU)*

Results (1)

Results



$$\frac{\text{RMSE}}{\sigma_{\text{input}}}$$

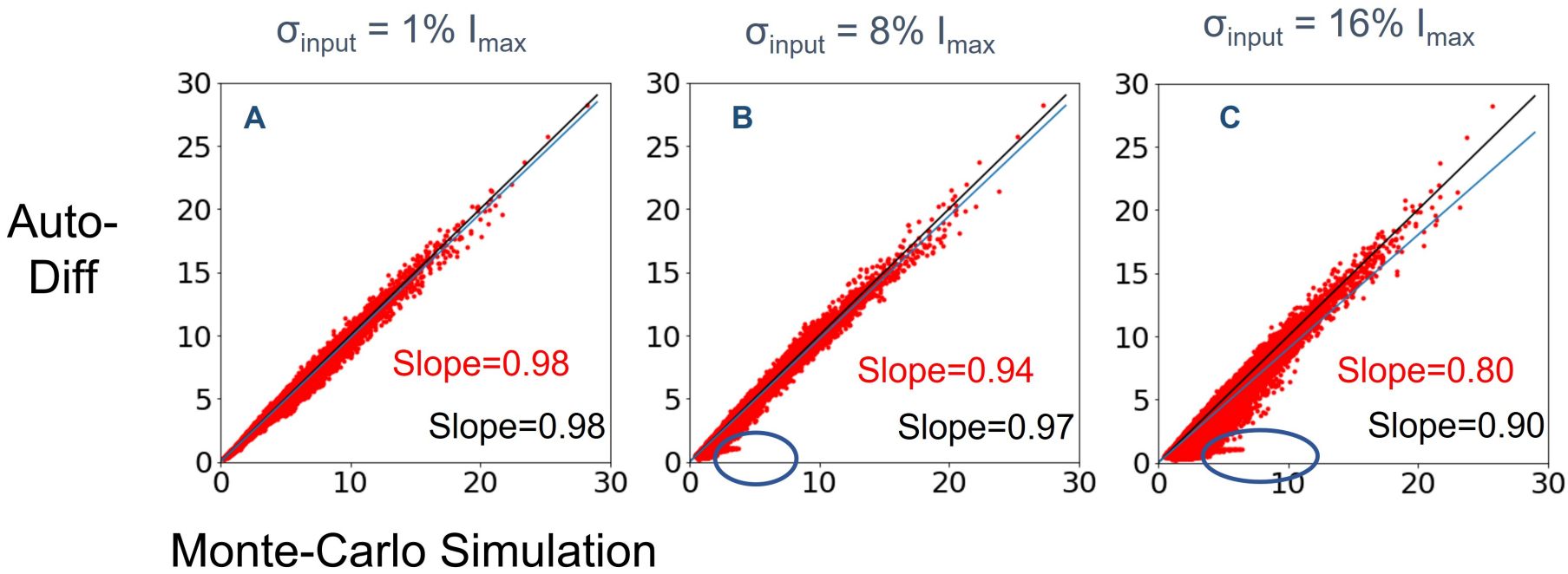


- Noise amplification is structured: generally higher at sharp edges

- Auto-Diff agrees well with Monte-Carlo
- Agreement poorer at higher noise levels → non-linearity of NN?
- Deviations more pronounced in regions of low signal intensity (e.g., background and in ventricles)

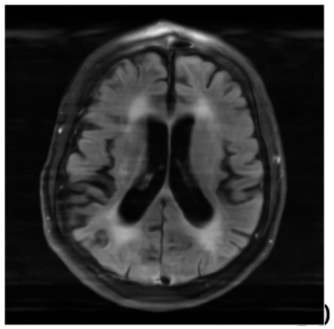
Results (2)

Pixel-by-pixel $\frac{\text{RMSE}}{\sigma_{\text{input}}}$: Auto-Diff versus MC

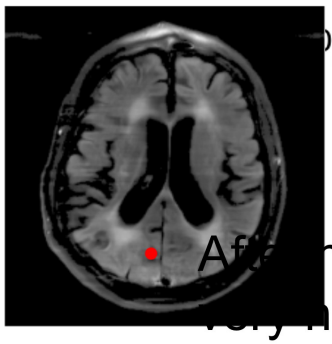


- Auto-Diff (Linear Model) agrees well with Monte-Carlo Simulation
- Agreement is less strong with higher noise level
- The outlying voxels are mostly in background and ventricles

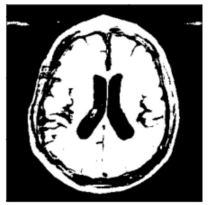
Results (3)



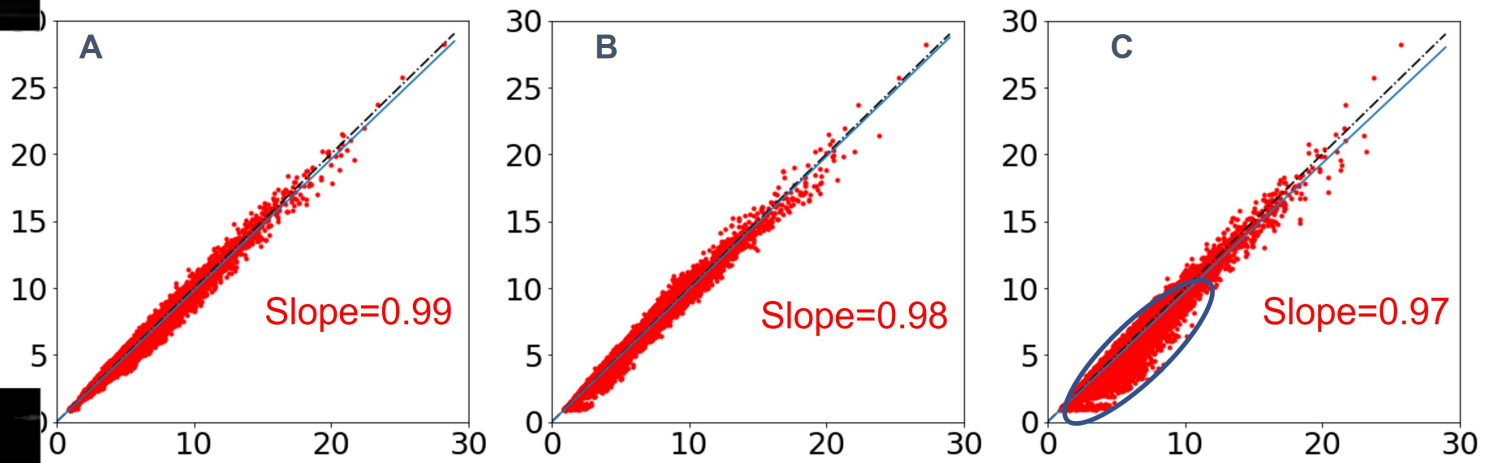
×Auto-diff



$\frac{\text{RMSE}}{\sigma_{\text{input}}}$ with masking: Auto-Diff versus MC



$\sigma_{\text{input}} = 1\%\ I_{\max}$ $\sigma_{\text{input}} = 8\%\ I_{\max}$ $\sigma_{\text{input}} = 16\%\ I_{\max}$

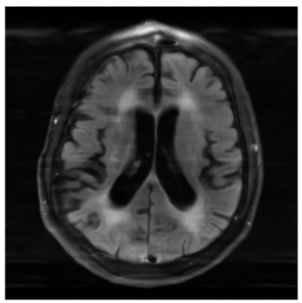


Monte-Carlo Simulation

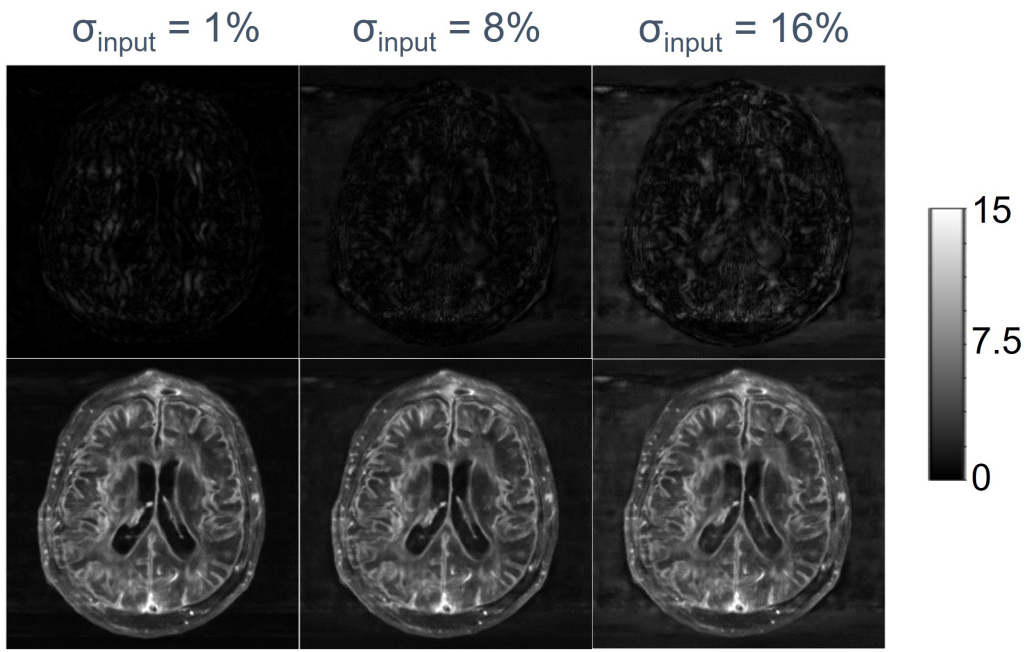
- After masking, very good agreement between MC and the Auto-Diff even at very high noise levels
- However, Auto-Diff still tends to underestimate noise

Results (4)

Reconstruction without Noise



Monte Carlo Simulation - Bias



(RMSE = $|\text{Bias}|^2$+Variation)

- As the standard deviation of noise increases, so does the bias.
- This may also have contributed to the divergence between the auto-diff and Monte-Carlo simulation
- However, even at the highest noise level, the bias was lower than the RMSE

Table of Contents

Normalizing Flows with DNN

Collaborators:
Chris Dock (UMD), currently at Tufts University
Tushar Jain, Sahil Sidheekh, Maneesh Singh (Verisk)
Joint Work:
UAI 2022: VQ-Flows: Vector Quantized Local Normalizing Flow

Deep latent variable models (DLVMs)

Given samples $X = (x_n)_{n=1}^N \in \mathcal{X}$ drawn from an unknown distribution $p(x)$, the goal of generative machine learning is to obtain new and "realistic" samples also drawn from $p(x)$. One way to do this is to assume that most of the variation in the unknown distribution arises from a latent variable z that is simply distributed according to $q(z)$ (usually Gaussian), in which case Bayes gives

$$p(x) = \int_{\mathcal{Z}} p(x|z)q(z)dz$$

Once $p(x|z)$ is known, new samples can be generated by first sampling z_0 from $z \sim q(z)$ and then sampling $x \sim p(x|z = z_0)$. LVM's are useful for

- Data Augmentation (by generating new samples that follow the same distribution as the data)
- Domain Adaptation (the latent space provides a common representation between domains)
- Outlier Detection

Deep latent variable models (DLVMs)

When $p(x|z) := p_\theta(x|z)$ are parameterized as DNNs, such models are termed DLVMs. Maximum likelihood estimation for θ gives:

$$\text{argmax}_\theta \log p_\theta(X) = \text{argmax}_\theta \sum_{n=1}^{N} \log \int_{\mathcal{Z}} p_\theta(x_n|z) q(z) dz$$

When $p_\theta(x|z)$ is given by a DNN, this objective is intractable to evaluate, let alone optimize.

- VAEs instead optimize the following variational lower bound for $\log p_\theta(X)$ that holds for any distribution $q_\phi(z|x)$, with equality when $q_\phi(z|x) = p(z|x)$:

$$\log p_\theta(X) \geq \sum_{n=1}^{N} \mathbb{E}_{z \sim q_\phi(\cdot|x_n)}[\log p_\theta(x_n|z)] - D_{KL}(q_\phi(\cdot|x_n)||q(z))$$

- GANs do not directly model $p(x|z)$, instead they sample $Z = (z_n)_{n=1}^N$ from $z \sim q(z)$ and seek to learn a generator function $G_\theta : \mathcal{Z} \to \mathcal{X}$

Global normalizing flows

Neither VAEs or GANs can provide exact densities $p(x)$, as VAEs replace $\log p(x)$ by a lower bound and GANs do not have an explicit probability model. NFs, however, make the assumption that $p_\theta(x|z)$ is of the form

$$p_\theta(x|z) = \delta(x - g_\theta(z))$$

Where $g_\theta : \mathcal{Z} \to \mathcal{X}$ is a diffeomorphism with inverse $f_\theta : \mathcal{X} \to Z$. In other words, at the level of the random variables x and z it is assumed that

$$x = g_\theta(z) \qquad z = f_\theta(x)$$

Note further that NFs are the $\sigma \to 0$ limit of the VAE given by $p_\theta(x|z) = \mathcal{N}(g_\theta(z), \sigma^2\mathbb{I})$ and $q_\theta(z|x) = \mathcal{N}(f_\theta(z), \sigma^2\mathbb{I})$.
Change of variables gives

$$\log p_\theta(X) = \sum_{n=1}^{n} \log \int_{\mathcal{Z}} p_\theta(x|z)q(z)dz$$

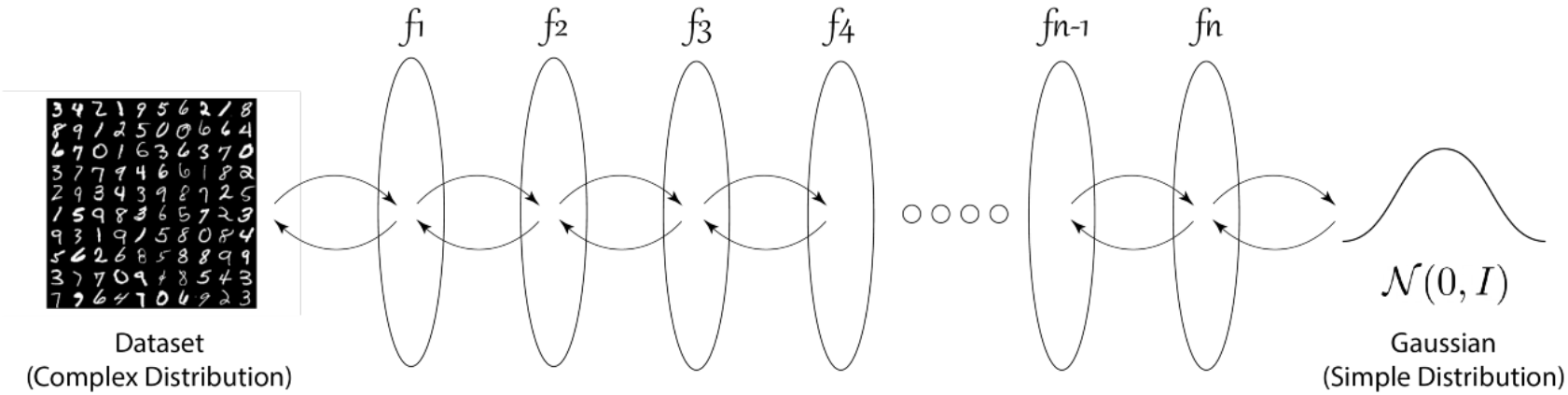
$$= \sum^{n} \log |\mathrm{Det}[\, If_\theta(x)\,]| + \log q(f_\theta(x))$$

Global normalizing flows

To compute θ for a NF one would like to maximize

$$\log p_\theta(X) = \sum_{n=1}^{n} \log |\mathrm{Det}[Jf_\theta(z)]| + \log q(f_\theta(z))$$

Because computing $\mathrm{Det}[Jf_\theta]$ is intractable for an arbitrary deep neural network, one builds f out of compositions $f_\theta = f_L^{\theta_L} \circ \cdots \circ f_1^{\theta_1}$ where $\mathrm{Det}[Jf_k^{\theta_k}(z)]$ and $g_k^{\theta_k} = (f_k^{\theta_k})^{-1}$ are simple to compute and $\theta = \mathrm{vec}(\theta_1, \cdots, \theta_L)$.



In this case the log-likelihood breaks apart to produce a tractable objective:

Global normalizing flows

The ability to exactly compute $p(x)$ makes NFs a powerful generative method, but they have an Achilles heel: they are diffeomorphisms. This means that the data manifold must be topologically equivalent to the latent space in order for NFs to get good results. In particular, the data manifold must have *the same dimension* as the latent space. The *manifold hypothesis*, however, suggests that often real data (like images) lies on a much lower dimensional submanifold $\mathcal{M} \subset \mathcal{X}$.

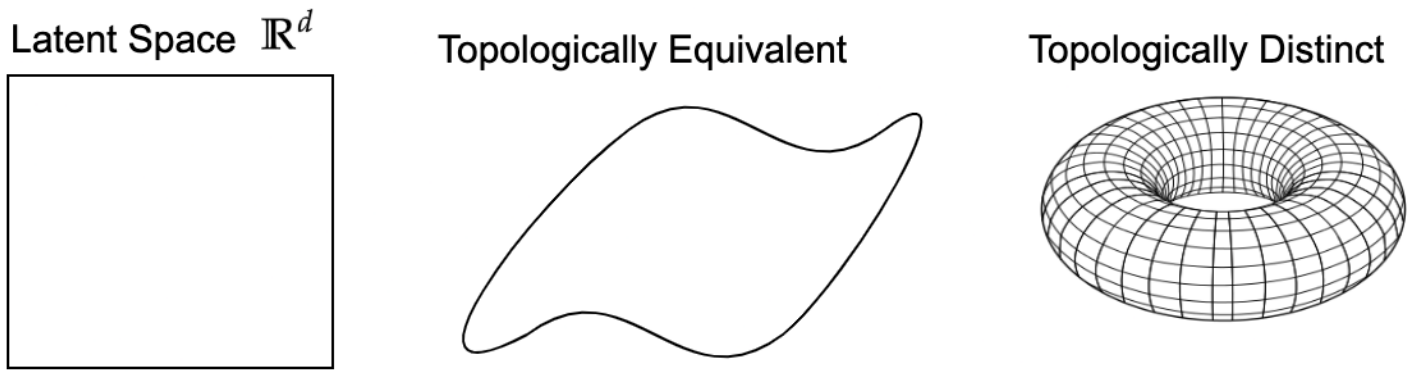


Figure: Topological constraints on an NF. Anything with "non-trivial topology"

Conformal embedding flows

One way to do dimensionality change using a NF $g_\theta : \mathcal{Z} \to \mathcal{U}$ is to post-compose it with a dimension raising embedding $h : \mathcal{U} \to \mathcal{X}$ to form $h \circ g_\theta$. Unfortunately the resulting measure change factor

$$|\text{Det}[(JhJg_\theta)^T(JhJg_\theta)]|^{-\frac{1}{2}} = |\text{Det}[Jg_\theta^T Jh^T JhJg_\theta]|^{-\frac{1}{2}}$$

does not separate into a product. A solution is to restrict h to be conformal [**?**]:

$$\mathcal{C}(\mathbb{R}^d \to \mathbb{R}^D) := \{c \in C^1(\mathbb{R}^d \to \mathbb{R}^D) \mid \exists \lambda \in C^0(\mathbb{R}) : Jc(u)^T Jc(u) = \lambda(u)^2 \mathbb{I}_d$$

If $g_\theta : \mathcal{Z} \to \mathcal{U}$ is a NF and $c \in \mathcal{C}(\mathcal{U} \to \mathcal{X})$ then the measure change factor is:

$$|\text{Det}[(JcJg_\theta)^T(JcJg_\theta)]|^{-\frac{1}{2}} = |\lambda(u)|^{-1}|\text{Det}Jf_\theta| = |\lambda(u)|^{-1}\prod_{j=1}^{L}|\text{Det}Jf_j^{\theta_j}|$$

In this case the log-likelihood separates nicely as:

$$\sum_{n=1}^{N}\left(\log p_{\mathcal{Z}}(g_\theta^{-1}(x^{(n)})) - \sum_{j=1}^{L}\log|\text{Det}Jf_j^{\theta_j}(z_j^{(n)})| - \log|\lambda(u^{(n)})|\right)$$

Conformal embedding flows

$\mathcal{C}(\mathbb{R}^d \to \mathbb{R}^D)$ is quite rich, but hard to parameterize. In [**?**] the authors restrict to embeddings of the form:

$$c = c_J \circ \cdots \circ c_1$$

where each c_j is either a trivially conformal zero padding or a dimension preserving conformal map, which for $d > 2$ are Möbius transformations (by Liouville):

$$M(A, a, b, \alpha, \epsilon)(x) = b + \alpha(Ax - a)/||Ax - a||^{\epsilon}$$

where $A \in O(d)$ is an orthogonal matrix, $\alpha \in \mathbb{R}$, $a, b \in \mathbb{R}^d$, and ϵ is either 0 or 2.

Unfortunately if $p_s : \mathbb{R}^d \to \mathbb{R}^{d+s}$ is the zero padding operation, $m_1 = M(A_1, a_1, b_1, \alpha_1, \epsilon_1)$ is a d dimensional Möbius transformation and m_2 is a $d + s$ dimensional Möbius transformation then for $x \in \mathbb{R}^d$:

$$m_2 \circ p_s \circ m_1(x) = (m_2 \cdot \tilde{m_1})(p_s(x))$$

Chart based flows - motivation

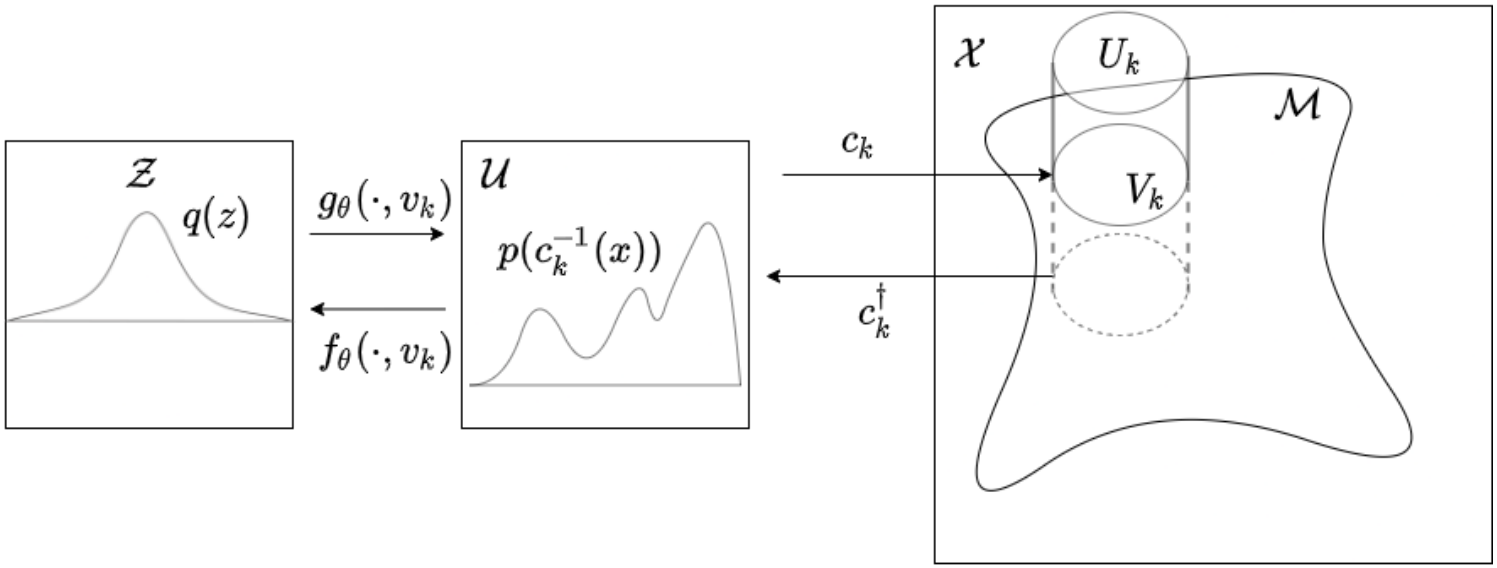
Vanilla NFs don't perform well for data on a low dimensional manifold $\mathcal{M} \subset \mathcal{X}$. Current extensions of NFs to allow for dimensionality change restrict expressivity. Idea:

- $\mathcal{M}$ diffeomorphic to $\mathcal{Z} \simeq \mathbb{R}^d$ is too restrictive. Impediment to performance is topological, suggesting a few "cuts" of the data would greatly improve NFs.
- Use a VQAE $(E, D, \{v_k\}_{k=1}^K)$ to learn $(U_k)_{k=1}^K$, a collection of open sets in $\mathcal{X}$ with $X \in \bigcup_{k=1}^K U_k$. With $V_k = U_k \cap \mathcal{M}$, $(V_k, f_{k,\theta}|_{V_k})$ provides an atlas of charts on $\mathcal{M}$. Model $p(x)$ as a mixture of normalizing flows

$$p(x|z) = \sum_{k=1}^{K} p_k \delta(x - g_{k,\theta}(z))$$

Where $g_{k,\theta} : \mathcal{Z} \to U_k$ is a conformal NF and $p_k = p(x \in U_k) / \sum_{j=1}^K p(x \in U_j)$. Note $p(x \in V_k) = p(x \in U_k)$ since

Chart based flows - motivation



Since c_k are Möbius transformations composed with zero paddings, the Riemann measure on V_k is simply a re-scaling of the pullback to the Lebesgue measure on $\mathcal{U}$. Thus in this sense the invertible normalizing flows $f_\theta(\cdot, v_k)$ are responsible for learning the probability measure $p(x)d_{\mathcal{M}}x$ and the conformal embeddings are responsible for learning the manifold $\mathcal{M}$.

Chart based flows - probability model

Assume $(U_k)_{k=1}^K$ are known. Let z be a r.v. taking values in $\mathcal{Z}$ and let k be a r.v. taking values in $\{1, \cdots, K\}$. Then assume x, z, k are jointly distributed as:

$$p(x, z, k) = \delta(x - g_{k,\theta}(z))q(z)p_k$$

Where $g_{k,\theta} : \mathcal{Z} \to U_k$ has (pseudo) inverse $f_{k,\theta} : U_k \to \mathcal{Z}$. Suppressing θ,

$$\begin{aligned} p(x, k) &= p_k \int_{\mathcal{Z}} \delta(x - g_k(z))q(z)dz \\ &= p_k \mathbb{1}_{U_k}(x) \int_{\mathcal{Z}} \delta(z - f_k(x))|\det\,[Jg_k(z)]|^{-1} q(z)dz \\ &= p_k \mathbb{1}_{U_k}(x)|\det\,[Jg_k(f_k(x))]|^{-1} q(f_k(x)) \\ &= p_k \mathbb{1}_{U_k}(x)|\det\,[Jf_k(x)]|q(f_k(x)) \end{aligned}$$

Thus we obtain the density $p(x)$ as

$$p(x) = \sum p_k|\det\,[Jf_k(x)]|q(f_k(x))$$

Chart based flows - VQAEs

Given data $\{x\}_{n=1}^{N} \in \mathcal{X}$ and a latent space $\mathcal{V}$ with $\dim \mathcal{V} << \dim \mathcal{X}$ a VQAE seeks to learn an encoder $E : \mathcal{X} \to \mathcal{V}$, a decoder $D : \mathcal{V} \to \mathcal{X}$, and a collection of encoded centers $\{v_k\}_{k=1}^{K} \subset \mathcal{V}$ so that the following loss is minimized:

$$\mathbb{E}_{x \sim p(x)}[\mathcal{L}(D(\arg\min_{v_k} ||v - E(x)||_2), x)]$$

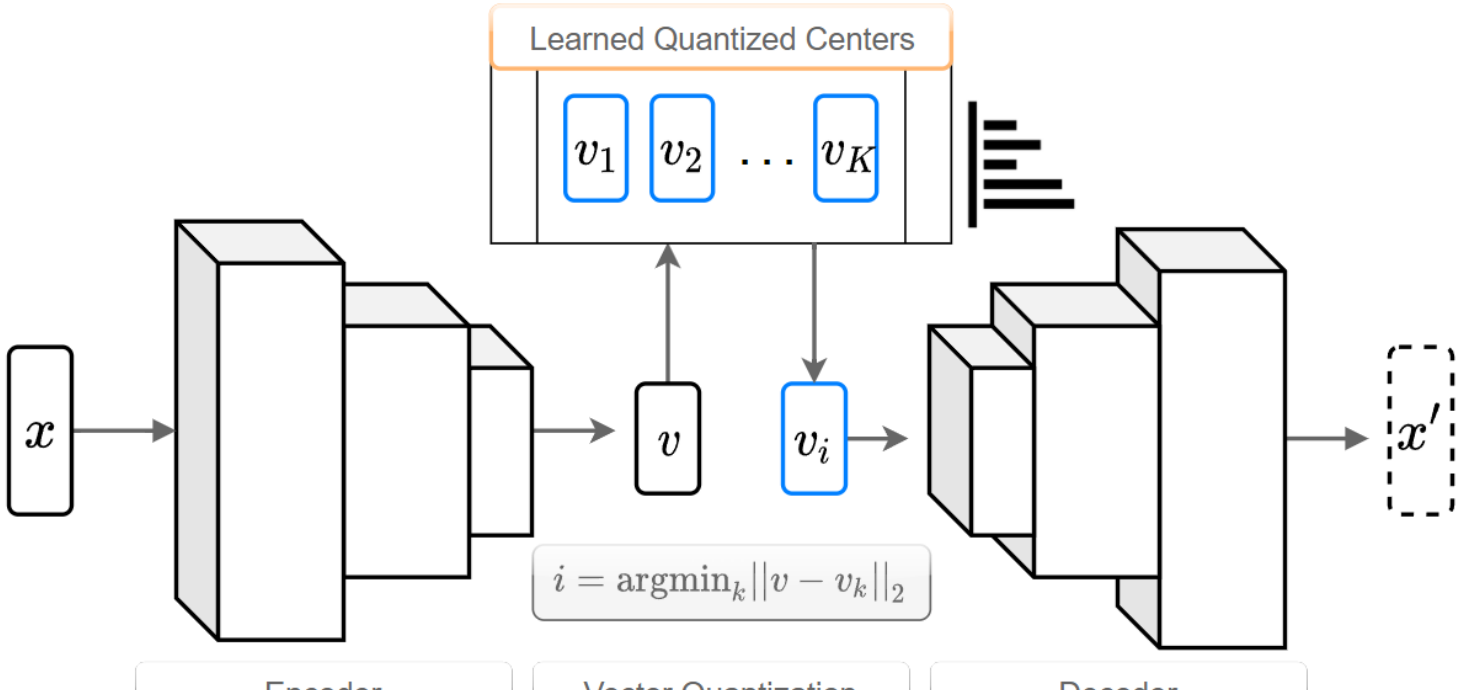


Chart based flows - chart design

We would like charts $(U_k)_{k=1}^K$ that cover X, that overlap, and that are sparse in the sense that no single $x \in X$ is contained in too many charts.

Definition

Given $v_1, \cdots, v_K \in \mathcal{V} \simeq \mathbb{R}^d$ the (m, ϵ)-Voronoi cell of v_k is

$$V_k = \{v \in \mathcal{V} \mid \exists J \subset [K] \ |J| > K - m \ \text{ and } ||v - v_k|| \leq (1+\epsilon)||v - v_j||_2 \forall j \in$$

Once a VQAE $(E, D, \{v_k\}_{k=1}^K)$ is trained, we can use the pullback through E of (m, ϵ)-Voronoi cells as charts:

$$U_k := \{x \in \mathcal{X} \mid E(x) \in V_k\}$$

Note that checking whether $x \in U_k$ amounts to computing $d_1 := ||E(x) - v_1||_2$ through $d_k := ||E(x) - v_k||_2$ and checking whether $d_k \leq (1+\epsilon)\tilde{d}_m$ where $\tilde{d}_1 \leq \cdots \leq \tilde{d}_K$. Here ϵ and m are hyper-parameters of the model. Note that if $m(x) := |\{k : x \in U_k\}|$ then $m(x) \geq m$ and

Chart based flows - implementation

Each conformal normalizing flow $g_{1,\theta}, \cdots, g_{K,\theta}$ is trained on only data lying in U_k. Even so, training K separate flows $g_{1,\theta}, \cdots, g_{K,\theta}$ becomes infeasibly time consuming as K increases (VQAE produces ~ 120 charts for the MNIST dataset), so instead let $g_\theta : \mathcal{Z} \times \mathcal{V} \to \mathcal{X}$ be such that $g(z, v_k) \in U_k$ for all z. Then assume

$$g_{k,\theta}(z) = g_\theta(z, v_k)$$

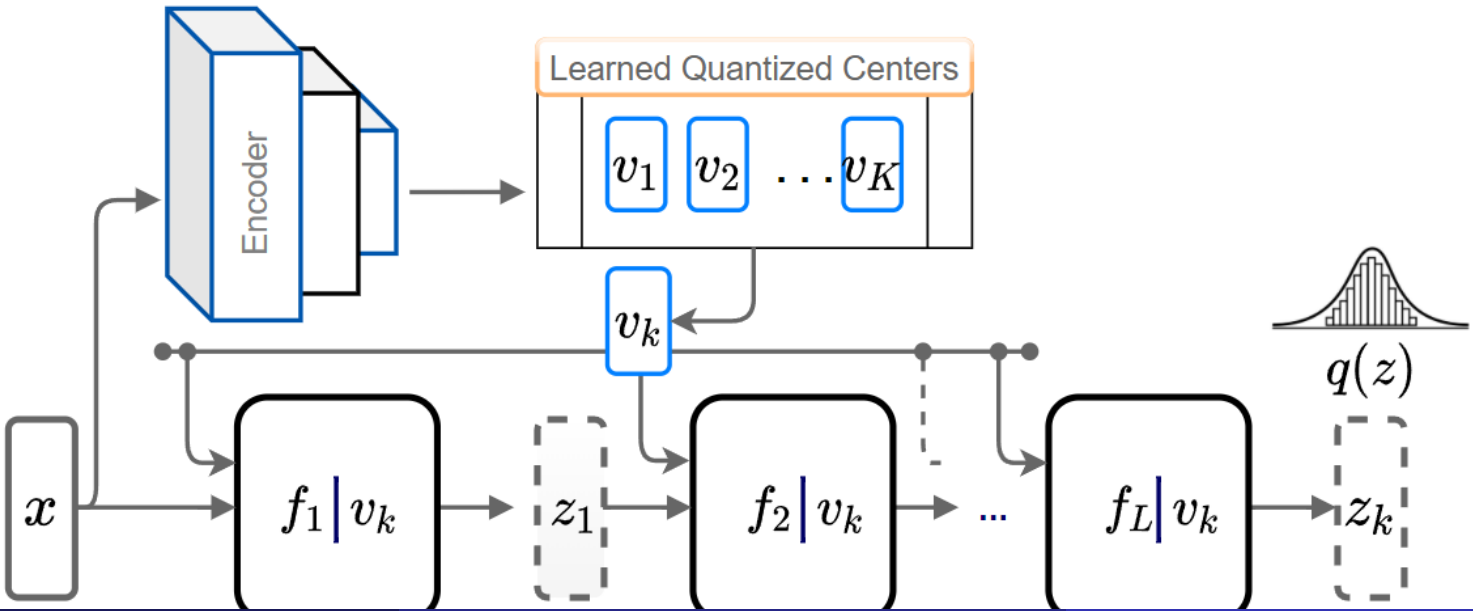


Chart based flows - training

During training the objective function is

$$\ln p_\theta(X) = \sum_{n=1}^{N} \ln p_\theta(x_n) = \sum_{n=1}^{N} \ln \sum_{k:x_n \in U_k} p_k p(x_n|k)$$

$$= \sum_{n=1}^{N} \ln \sum_{k:x_n \in U_k} p_k q(f_k(x_n)) |\lambda_k(c_k^\dagger(x_n))|^{-1} \prod_{l=1}^{L} |\det [Jf_k^l(f_k^{l+1} \circ \cdots \circ f_k^L$$

Noting that $p(x|k)$ is zero unless $x \in U_k$. The density $p(x)$ can also be written

$$p(x) = \mathbb{E}_{k \sim \tilde{p}_x(k)}[p(x|k)] \qquad \underbrace{\sum_{j:x \in U_j} p(j)}_{\text{piecewise constant}}$$

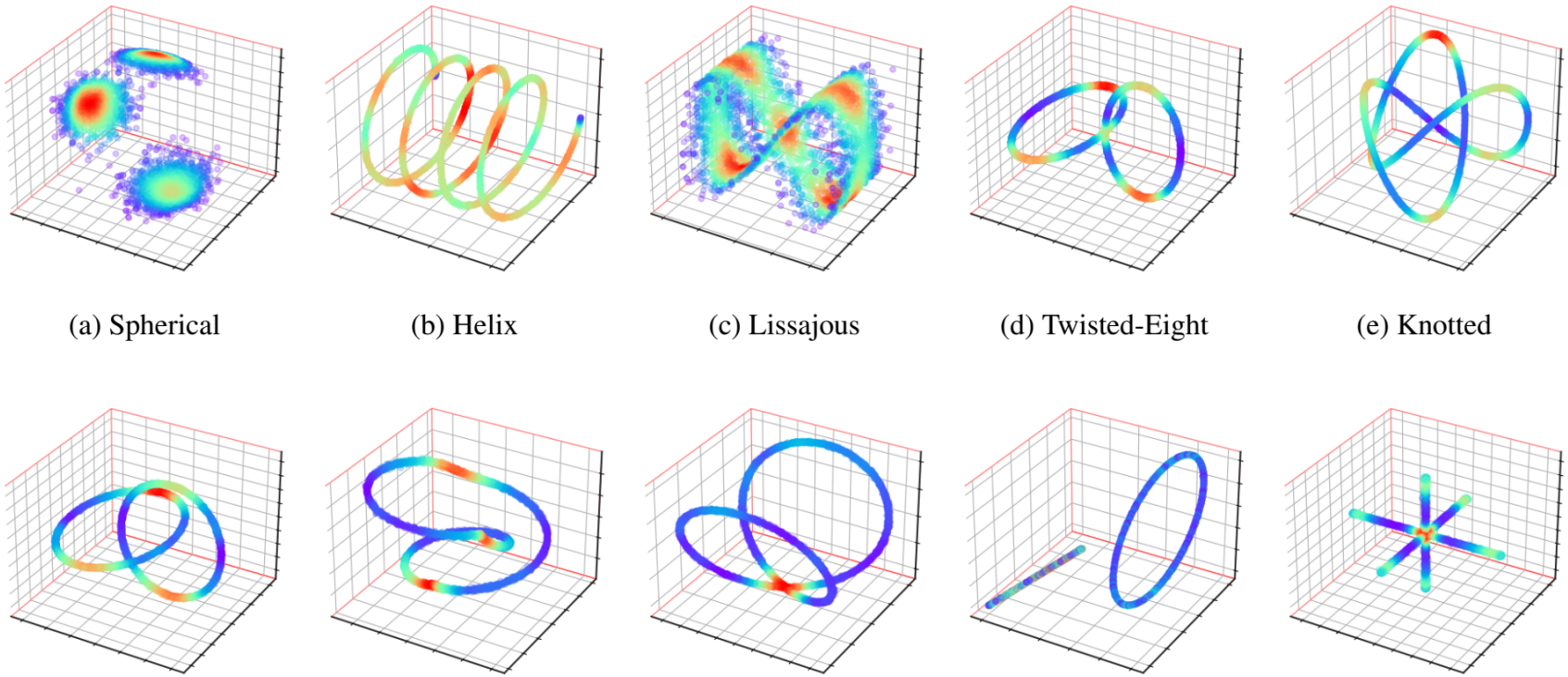
Where $\tilde{p}_x(k) = p(k|p(x|k) > 0) = p_k / \sum_{j:x \in U_j} p_j$. During training we replace the expectation $\mathbb{E}_{k \sim \tilde{p}(k)}[p(x|k)]$ with the stochastic quantity

Chart based flows - sampling, inference, and density

- Sampling: Since z and k are independent sample z from $z \sim q(z)$ and $k \sim p_k$ and then compute $x = g_\theta(z, v_k)$.
- Inference: Since z is no longer wholly determined by x, but instead takes values $(f(x, v_k))_{k:x\in U_k}$ with corresponding probabilities $(p(k|x))_{k:x\in U_k}$. One could perform a stochastic inference via sampling $k \sim p(k|x)$ and computing $z = f(x, v_k)$. If deterministic inference is preferred then one may use the expected value of z as $z = \mathbb{E}_{k\sim p(k|x)}[f_k(x)] = \sum_{k:x\in U_k} p(k|x) f_k(x)$ or the most probable value of z as $z = f_s(x)$ where $s = \text{argmax}_{k:x\in U_k} p(k|x)$.
- Density Evaluation: If the exact density $p(x)$ is needed for $x \in \bigcup_{k=1}^K U_k$ it can be computed at the cost of $m(x)$ evaluations of a normalizing flow:

$$p(x) = \sum_{k:x_n\in U_k} p_k q(f_k(x_n)) |\lambda_k(c_k^\dagger(x_n))|^{-1} \prod_{l=1}^{L} |\det [Jf_k^l(f_k^{l+1} \circ \cdots \circ f_k^L(x_n))$$

Chart based flows - performance



(a) Spherical (b) Helix (c) Lissajous (d) Twisted-Eight (e) Knotted

Figure: Toy datasets with various topological features.

Chart based flows - performance

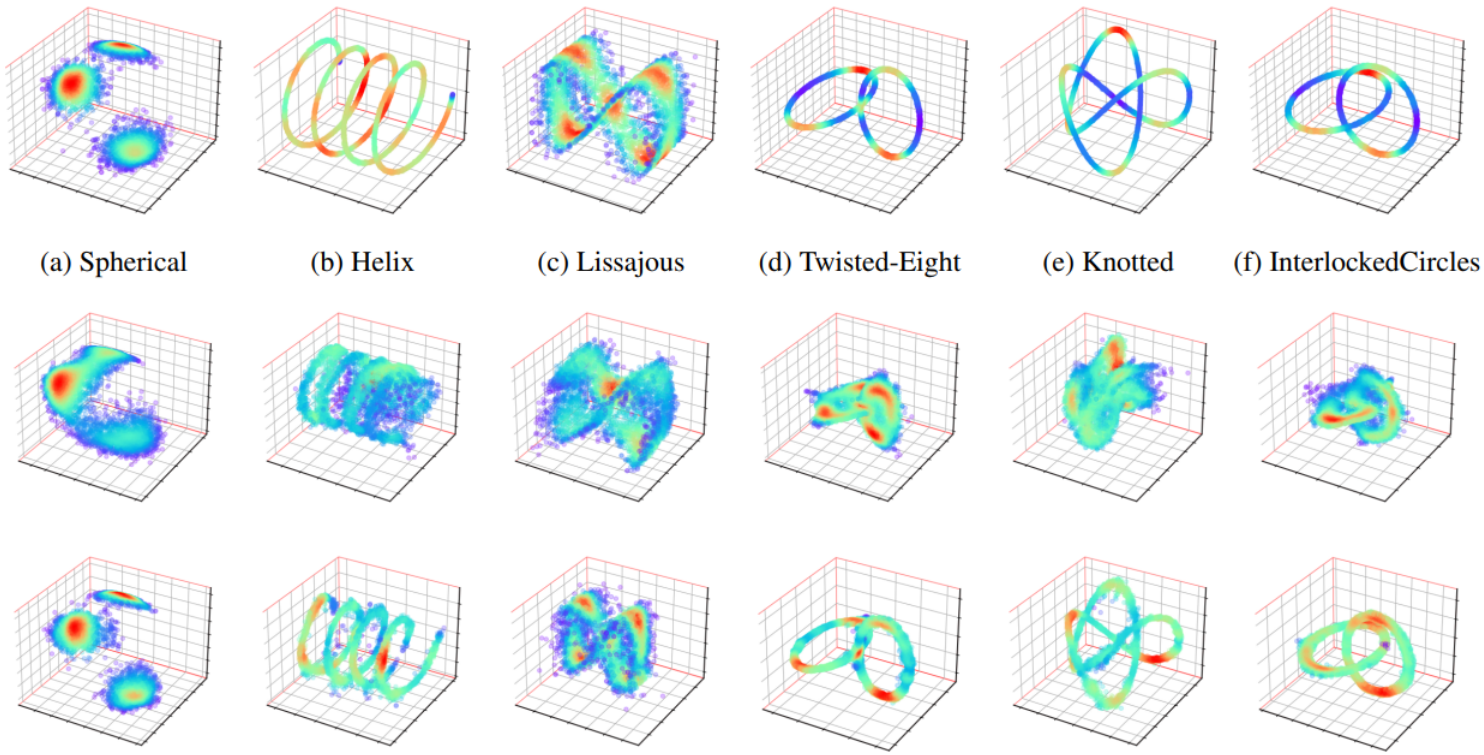


Figure: Qualitative visualization of the samples generated by a classical flow (Middle Row) and its VQ-counterpart (Bottom Row) trained on Toy 3D data distributions (Top Row).

Chart based flows - performance

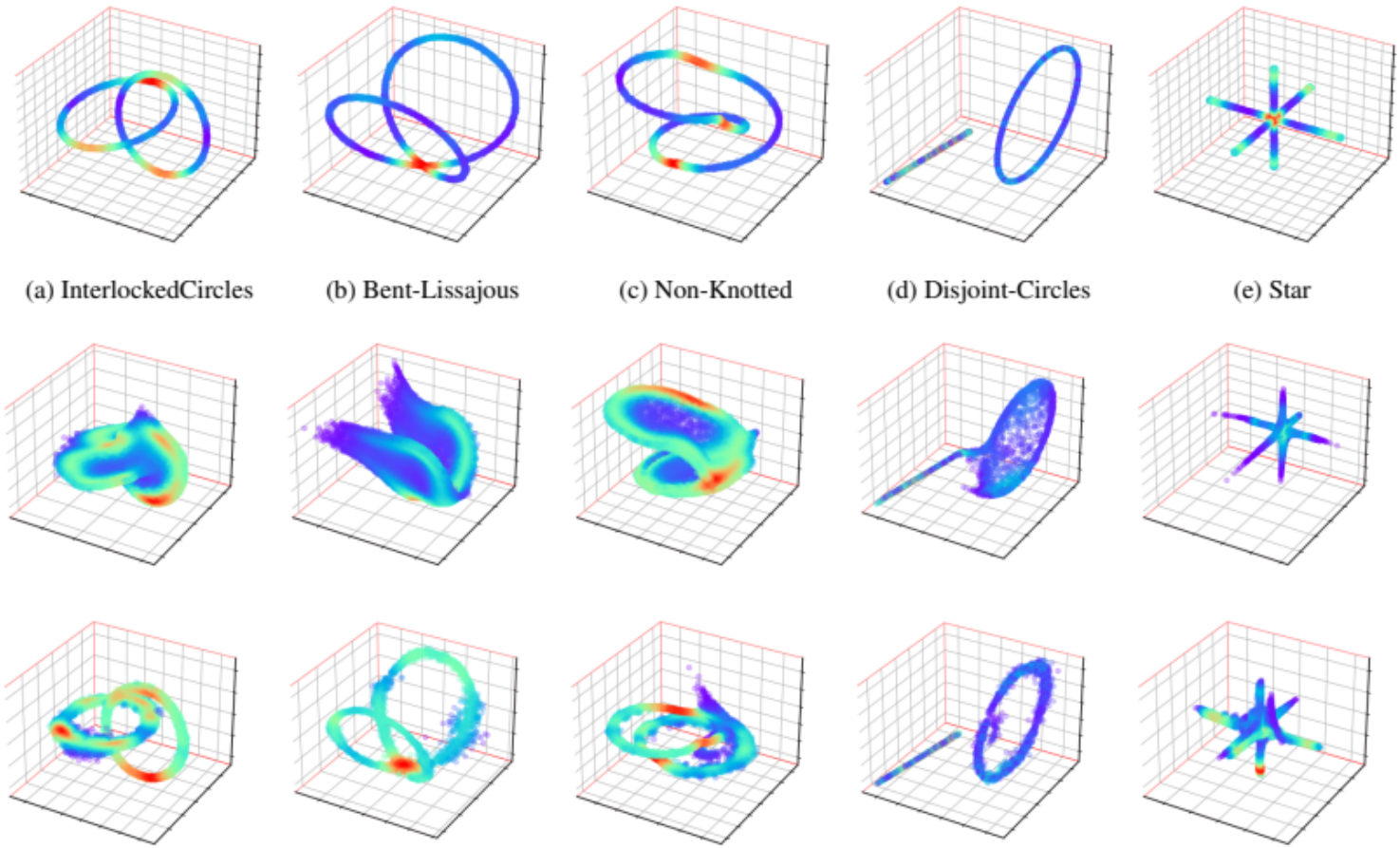


Figure: Qualitative visualization of the samples generated by a classical flow (Middle Row) and its VQ-counterpart (Bottom Row) trained on Toy 3D data

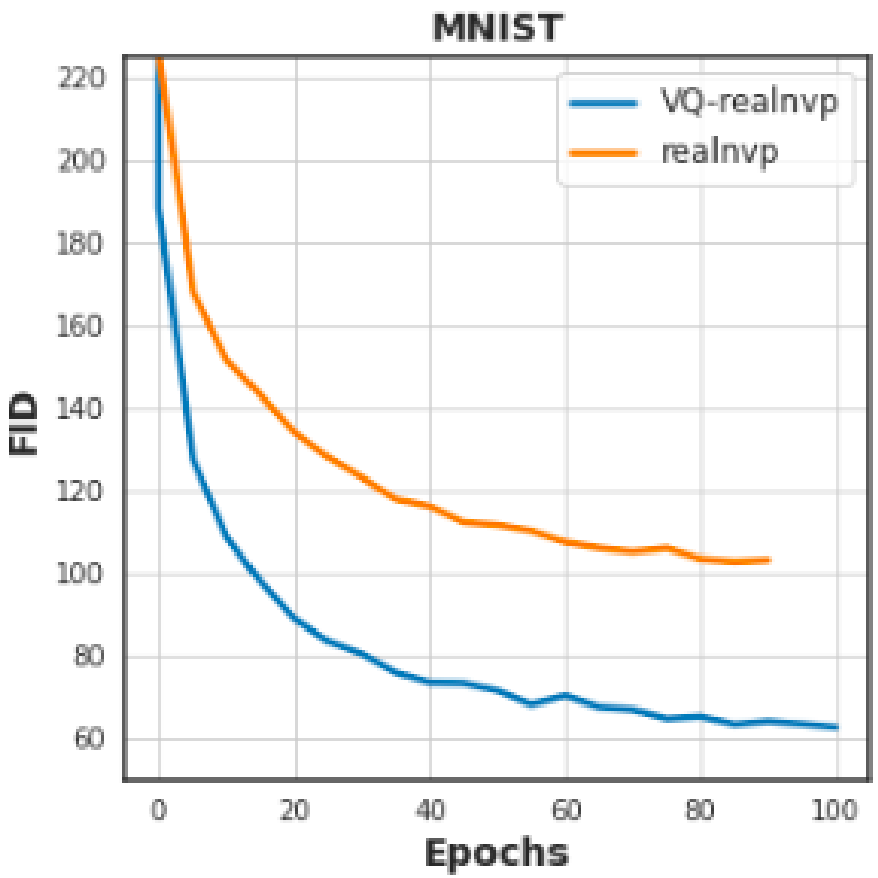
Chart based flows - performance

Model	Spherical	Helix	Lissajous	Twisted-Eight	Knotted	Interlocked-Circles
Real NVP	0.50 ± 0.07	-57.46 ± 2.11	0.18 ± 0.14	-2.72 ± 0.90	-8.65 ± 0.87	-2.18 ± 0.37
VQ-RealNVP	0.99 ± 0.14	-3.85 ± 0.98	0.59 ± 0.08	0.18 ± 0.17	-1.44 ± 0.37	-0.11 ± 0.12
MAF	0.65 ± 0.26	-92.83 ± 5.69	0.12 ± 0.16	-2.77 ± 0.81	-7.04 ± 0.49	-2.49 ± 0.14
VQ-MAF	1.01 ± 0.07	-4.62 ± 0.37	0.59 ± 0.07	-0.32± 0.13	-2.44 ± 0.11	-0.15 ± 0.08
CEF	-1.17 ± 0.06	-29.90 ± 2.12	0.38 ± 0.14	-4.03 ± 0.38	-19.40 ± 1.80	-3.42 ± 0.49
VQ-CEF	0.80 ± 3.42	-20.75 ± 2.22	0.49 ± 0.03	-3.51 ± 0.73	-14.44 ± 1.57	-3.23 ± 0.19

Model	Non-Knotted	Bent-Lissajous	Disjoint-Circles	Star
Real NVP	0.53 ± 0.18	1.04 ± 0.22	1.71 ± 0.12	3.33 ± 0.18
VQ-RealNVP	2.39 ± 0.24	2.62 ± 0.13	2.71 ± 0.19	4.23 ± 0.06
MAF	0.73 ± 0.18	1.48 ± 0.11	1.95 ± 0.12	3.53 ± 0.03
VQ-MAF	2.41 ± 0.19	2.06 ± 0.12	2.87 ± 0.07	3.59 ± 0.12
CEF	-0.46 ± 0.13	-0.51 ± 0.16	-0.71 ± 0.21	1.26 ± 0.11
VQ-CEF	-0.15 ± 0.09	-0.54 ± 0.22	0.24 ± 0.15	1.32 ± 0.02

Table: Quantitative evaluation of **Sample Generation** in terms of the log-likelihood of generated samples in nats (higher the better) on the 3D datasets. The values are averaged across 5 independent trials, ± represents the

Chart based flows - Higher Dimensional Data



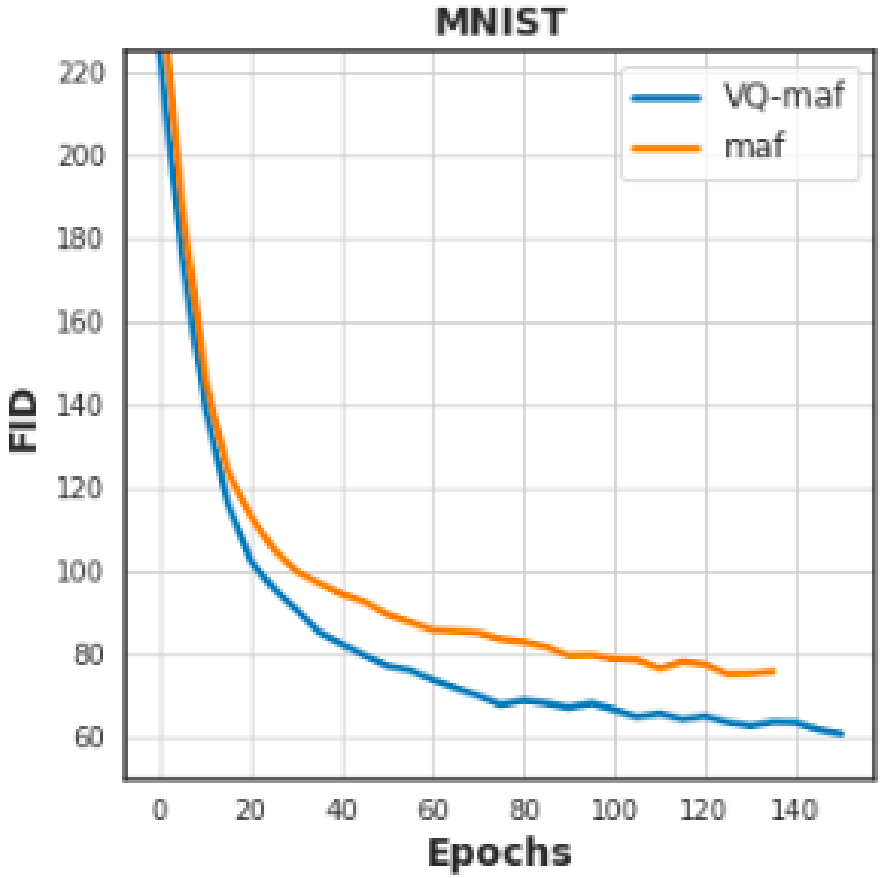


Figure: Seen here are the results of recent experiments on the MNIST dataset. FID is Fréchet Inception Distance (lower is better).

Table of Contents

Combinatorial Optimizations using DNN

Collaborators: **Maneesh Singh, Julian Yarkoni** (Verisk, NJ), Naveed Haghani (UMD)
Presented at SPIE 2019:
R. Balan, N. Haghani, "Discrete optimizations using graph convolutional networks", Proc. SPIE 11138 Wavelets and Sparsity XVIII, San Diego, August 2019

Quadratic Optimization Problems

Consider two symmetric (and positive semidefinite) matrices $A, B \in \mathbb{R}^{n \times n}$. The *quadratic assignment problem* asks for the solution of

$$\begin{aligned} &\textit{maximize} && \textit{trace}(\Pi A \Pi^T B) \\ &\text{subject to:} && \\ &\Pi \in S_n && \end{aligned}$$

where S_n denotes the symmetric group of $n \times n$ permutation matrices. Application: Develop a Graph Deep Learning architecture for solving the Quadratic Assignment Problem.

QAP

Motivation

Consider two $n \times n$ symmetric matrices A, B. In the alignment problem for quadratic forms one seeks an orthogonal matrix $U \in O(n)$ that minimizes

$$\|UAU^T - B\|_F^2 := trace((UAU^T - B)^2) = \|A\|_F^2 + \|B\|_F^2 - 2trace(UAU^T B).$$

The solution is well-known and depends on the eigendecomposition of matrices A, B: if $A = U_1 D_1 U_1^T$, $B = U_2 D_2 U_2^T$ then

$$U_{opt} = U_2 U_1^T \quad , \quad \|U_{opt} A U_{opt}^T - B\|_F^2 = \sum_{k=1}^{n} |\lambda_k - \mu_k|^2,$$

where $D_1 = diag(\lambda_k)$ and $D_2 = diag(\mu_k)$ are diagonal matrices with eigenvalues ordered monotonically.

QAP

Motivation 2

The challenging case is when U is constrained to belong to the permutation group. In this case, the previous minimization problem

$$\min_{U \in S_n} \|UAU^T - B\|_F$$

turns into the QAP:

$$\max_{U \in S_n} trace(UAU^T B).$$

In the case A, B are graph Laplacians (or adjacency matrices), an efficient solution to this optimization problem would solve the graph isomorphism problem, one of the remaining milenium problems: decide if two given graphs are the same modulo vertex labelling.

Prior work to discrete optimizations using deep learning

- Direct approach to discrete optimization: Pointer Networks (Ptr-Nets) utilize sequence-to-sequence Recurrent Neural Networks [Vinyals'15];
- Reinforcement learning and policy gradients: [Bello'16]
- Graph embedding and deep Q-learning: [Dai'17]
- QAP using graph deep learning: [Nowak'17] utilizes siamese graph neural networks that act on A and B independently to produce embeddings E_1 and E_2; then the product $E_1 E_2^T$ is transformed into a permutation matrix through soft-max and cross-entropy loss.

Shift Invariance Properties

Consider $A = A^T$ and $B = B^T$ (no positivity assumption).

Lemma

The QAP associated to (A, B) has the same optimizer as the QAP associated to $(A - \lambda I, B - \mu I)$, where $\lambda, \mu \in \mathbb{R}$.

Indeed, the proof of this lemma is based on the following direct computation:

$$trace(\Pi(A-\lambda I)\Pi^T(B-\mu I)) = trace(\Pi A \Pi^T B) - \mu trace(A) - \lambda trace(B) + n\lambda\mu$$

A consequence of this lemma is that, without loss of generality, we can assume $A, B \geq 0$. In fact, we can shift the spectrum to vanish the smallest eigenvalues of A, B.

The case of Rank One

Assume now $A = aa^T$ and $B = bb^T$ are non-negative rank one matrices. Then:

$$trace(\Pi A \Pi^T B) = |b^T \Pi a|^2 = (trace(\Pi a b^T))^2 = \frac{1}{trace(AB)}(trace(\Pi AB))^2$$

In this case we obtain the explicit solution to the QAP:

Lemma

Assume $A = aa^T$ and $B = bb^T$ are rank one. Then the QAP optimizer is the optimizer of one of the following two optimization problems:

$$\begin{array}{ll} \textit{maximize} & trace(\Pi C) \\ \textit{subject to:} & \\ \Pi \in S_n & \end{array} \quad \text{or} \quad \begin{array}{ll} \textit{minimize} & trace(\Pi C) \\ \textit{subject to:} & \\ \Pi \in S_n & \end{array}$$

where $C = AB$.

Linear Assignment Problems

Given a cost matrix $C \in \mathbb{R}^{n \times n}$, the *Linear Assignment Problem* (LAP) is defined by:

$$\begin{aligned} &\textit{maximize} && \textit{trace}(\Pi C) \\ &\text{subject to:} && \\ &\Pi \in S_n && \end{aligned}$$

Without loss of generality, max can be replace by min, for instance by solving LAP for $-C$.

Linear Assignment Problems

Given a cost matrix $C \in \mathbb{R}^{n \times n}$, the *Linear Assignment Problem* (LAP) is defined by:

$$\begin{array}{c} \textit{maximize} \quad \textit{trace}(\Pi C) \\ \text{subject to:} \\ \Pi \in S_n \end{array}$$

Without loss of generality, max can be replace by min, for instance by solving LAP for $-C$.

The key observation is that LAP can be solved efficiently by a linear program. Specifically, the convexification of LAP produces the same optimizer:

$$\begin{array}{c} \textit{maximize} \quad \textit{trace}(WC) \\ \text{subject to:} \\ W_{i,j} \geq 0 \ , \ 1 \leq i,j \leq n \\ \sum_{i=1}^{n} W_{i,j} = 1 \ , \ 1 \leq j \leq n \\ \sum_{j=1}^{n} W_{i,j} = 1 \ , \ 1 \leq i \leq n \end{array}$$

Diagonal Matrices

Another case when we know the exact solution is when A and B are diagonal matrices. Say $A = diag(a)$ and $B = diag(b)$. Then

$$trace(\Pi A \Pi^T B) = trace(diag(\Pi a) diag(b)) = trace(\Pi a b^T) = trace(\Pi C)$$

where $C = ab^T$.

Lemma

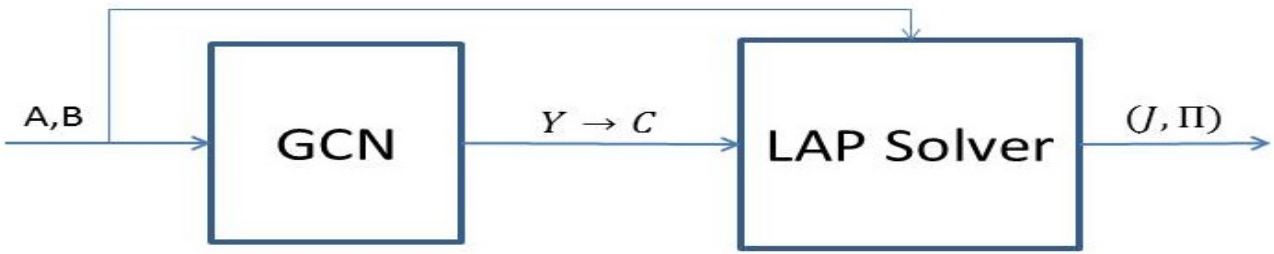
If $A = diag(a)$ and $B = diag(b)$ then the solution of the QAP is given by the solution of the LAP

$$\begin{aligned} &maximize && trace(\Pi C) \\ &subject\ to: && \\ &\Pi \in S_n && \end{aligned}$$

where $C = ab^T$.

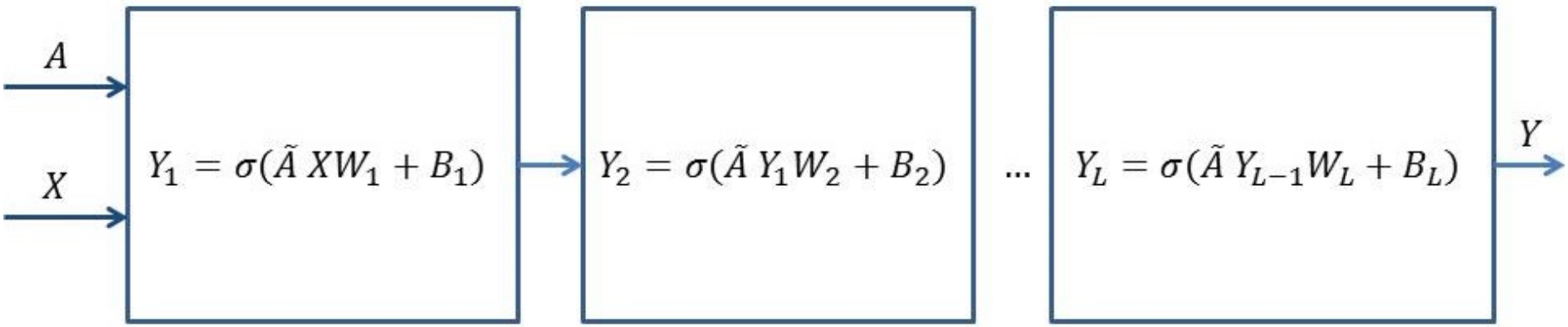
Approach

The idea is the following: First we convert the input data (A, B) into a cost matrix C, and then we solve two LAPs, one associated to C the other associated to $-C$. Finally we choose the permutation that produces the larger objective function.
The conversion step $(A, B) \mapsto C$ is performed by a Graph Convolutional Network (GCN).



Graph Convolutional Networks (GCN)

Kipf and Welling (2016) introduced a network structure that performs local processing according to a modified adjacency matrix:



Here $\tilde{T} = I + T$, where T is an input adjacency matrix, or graph weight matrix. The L-layer GCN has parameters $(W_1, B_1, W_2, B_2, \cdots, W_L, B_L)$. As activation map σ we choose the ReLU (Rectified Linear Unit).

The Specific GCN Architecture

For the QAP associated to matrices (A, B) we design a specific GCN architecture:

$$X = \begin{bmatrix} A & 0 \\ B & 0 \end{bmatrix}, \quad \tilde{T} = \begin{bmatrix} I_n & \frac{1}{\|A\|_F \|B\|_F} AB \\ \frac{1}{\|A\|_F \|B\|_F} BA & I_n \end{bmatrix} \tag{5.4}$$

where the 0 matrices in X are designed to fit the appropriate size of W_1. For σ we choose the ReLU (Rectified Linear Unit) function in each layer except for the last one; in the last layer we do not use any activation function (i.e., $\sigma = Identity$). The biases $B_1, \cdots, B_L$ are chosen of the form $B_k = 1 \cdot \beta_k^T$, i.e., each row β_k^T is repeated.

GCN Guarantee

The following result applies to this network.

Theorem

Assume $A = aa^T$ and $B = bb^T$ are rank one matrices, and consider the GCN with L layers and activation map ReLU as described above. Then for any nontrivial weights $W_1, \cdots, W_L$ and biases $B_1, \cdots, B_L$ (whose rows are repeated), the network output Y partitioned $Y = \begin{bmatrix} Y^1 \\ Y^2 \end{bmatrix}$ into two blocks of n rows each, satisfies $Y^1 Y^{2T} = \gamma AB$, for some constant $\gamma \in \mathbb{R}$. In particular, the max-LAP and min-LAP applied to the latent representation matrix $C = Y^1 Y^{2T}$ are guaranteed to produce the optimal solution of the QAP.

Reference Algorithms

We compare the GCN based optimizer with two different algorithms.
1. The *AB Method* bypasses the GCN block. Thus $Y = X$ and the cost matrix inputted into the LAP solver is simply $C = AB$ (hence the name of the method). Similar to the GCN approach, the AB Method is exact on rank 1 inputs. But there is no adaptation of the cost matrix for other input matrices.
2. The *Iterative* algorithm is based on alternating max-LAP or min-LAP as follows:

$$\Pi_{k+1} \in \left\{ \begin{array}{ll} \text{argmax} & trace(\Pi A \Pi_k^T B) \\ \Pi \in S_n & \end{array} \right. , \left. \begin{array}{ll} \text{argmin} & trace(\Pi A \Pi_k^T B) \\ \Pi \in S_n & \end{array} \right\}$$

where $\Pi_0 = I$ (identity), and the choice of permutation at each k is based on which permutation produces a larger $trace(\Pi A \Pi^T B)$.

Comparison with Ground Truth

Results for $2 \leq n \leq 10$ and raw data normal distributed

Average relative difference w.r.t. maximum objective function:

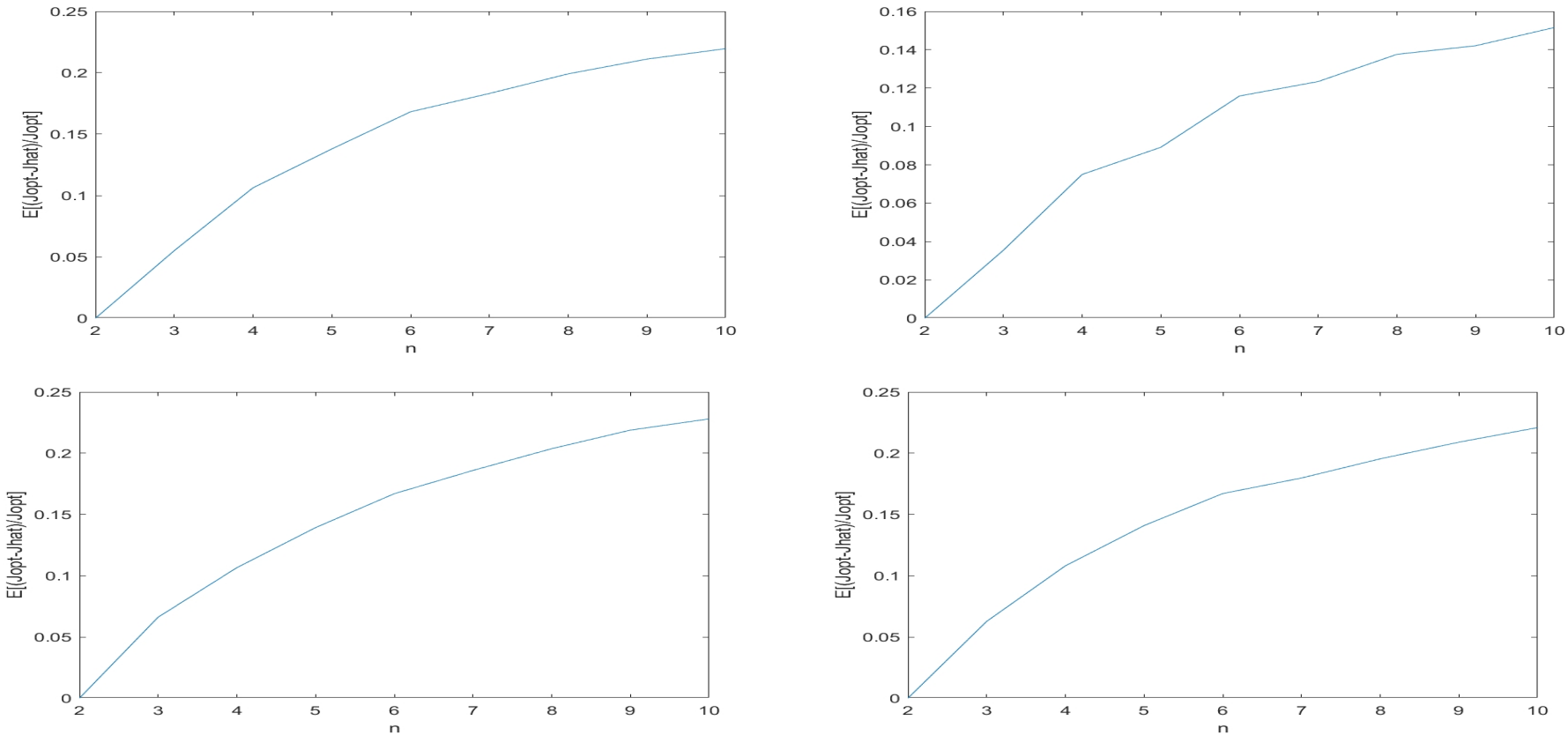


Figure: Top left: ABMethod, Top right: Iterative algorithm, Bottom left: GCN with L=2 layers and bais, Bottom right: GCN with $L = 3$ layers and bias

Comparison with Ground Truth

Results for $2 \leq n \leq 10$ and raw data uniform distributed

Average relative difference w.r.t. maximum objective function:

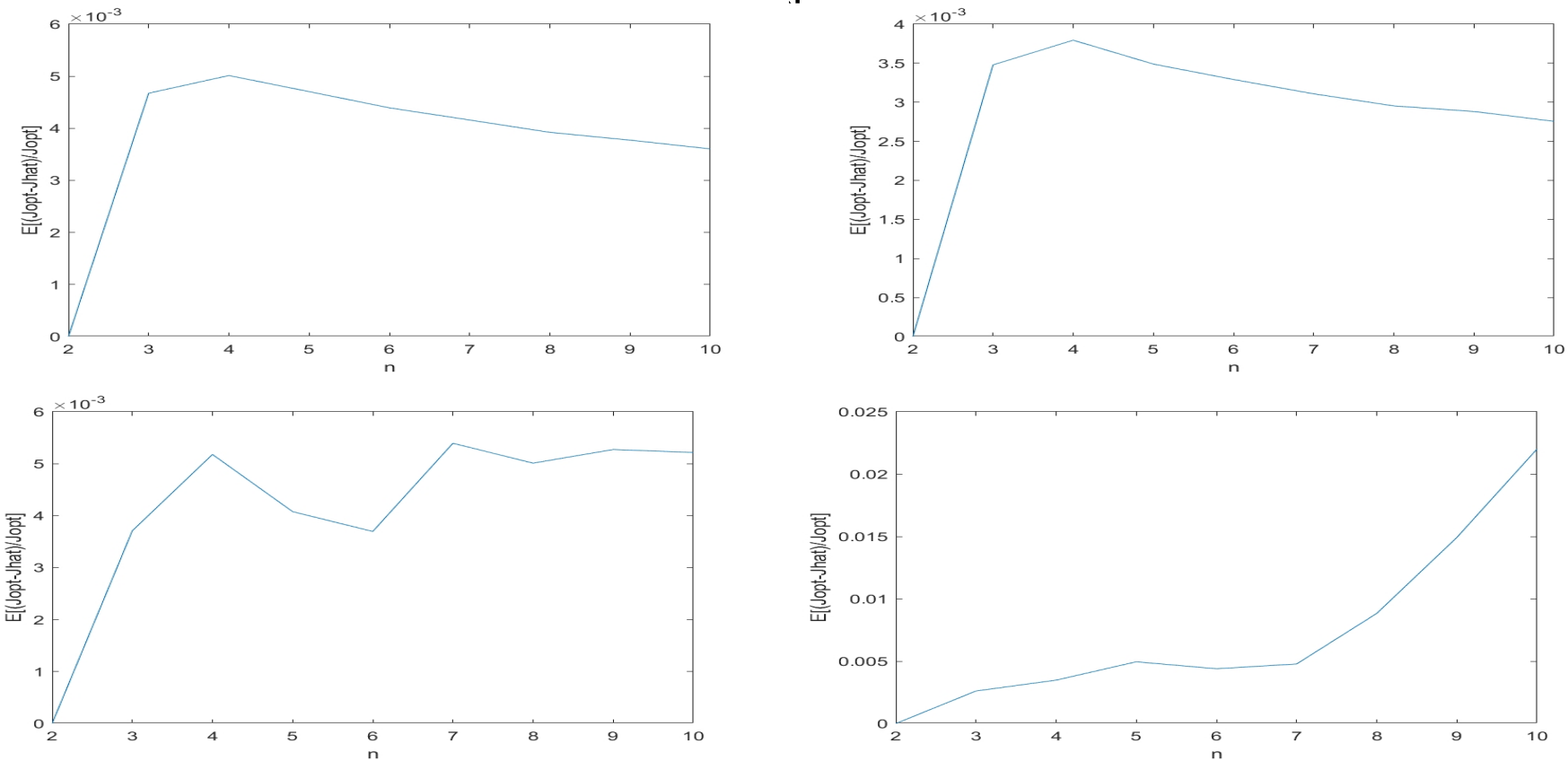


Figure: Top left: ABMethod, Top right: Iterative algorithm, Bottom left: GCN with L=2 layers and bais, Bottom right: GCN with $L = 3$ layers and bias

Relative Comparison

Results for $n = 100$ and $n = 200$ with raw data normal distributed

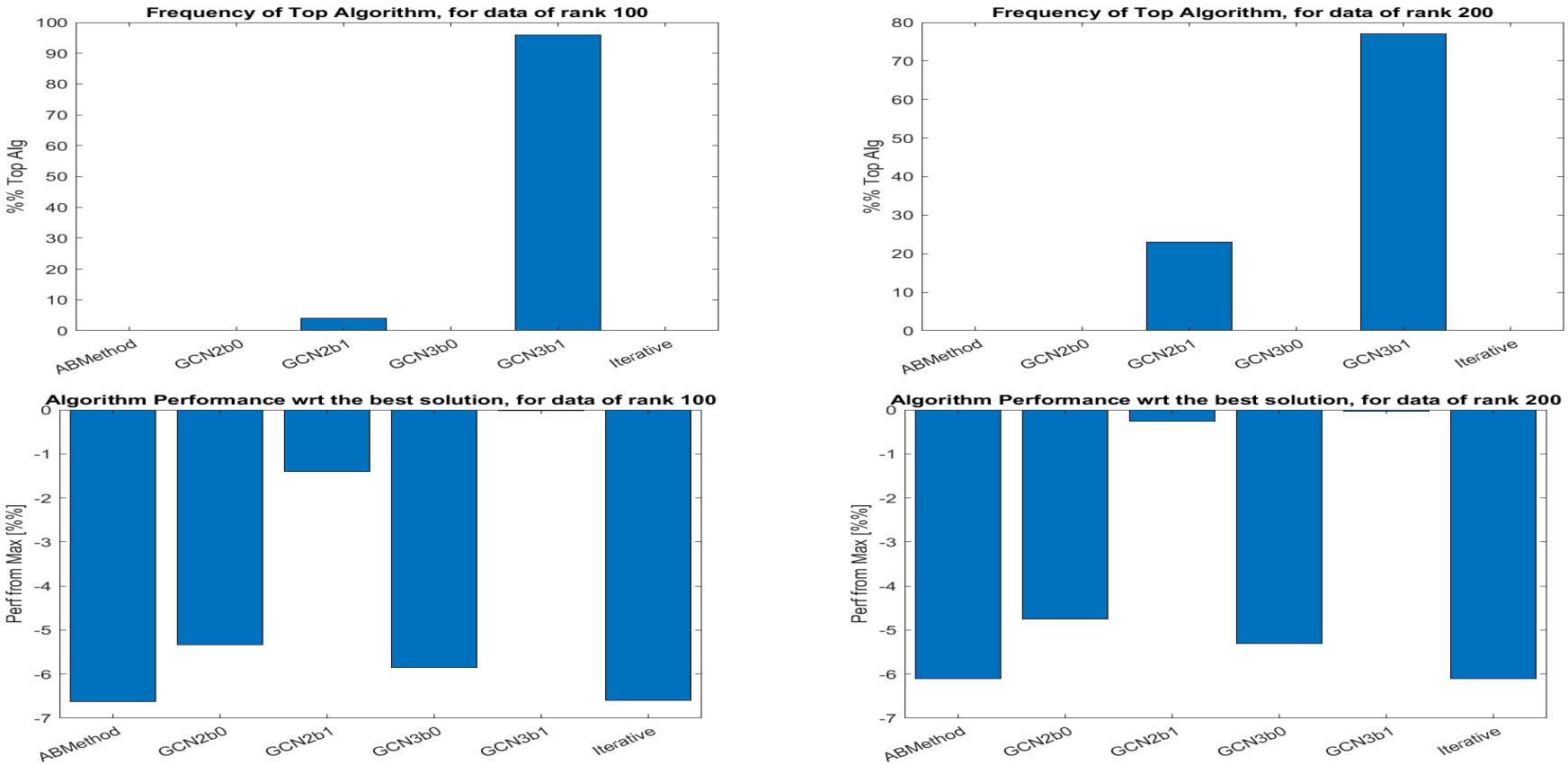


Figure: Top row: Frequency of optimal algorithm for $n = 100$ (left), and $n = 200$ (right). Borrom row: Relative performance [%] to the best algorithm for $n = 100$ (left) and $n = 200$ (right)

Relative Comparison

Results for $n = 100$ and $n = 200$ with raw data normal distributed

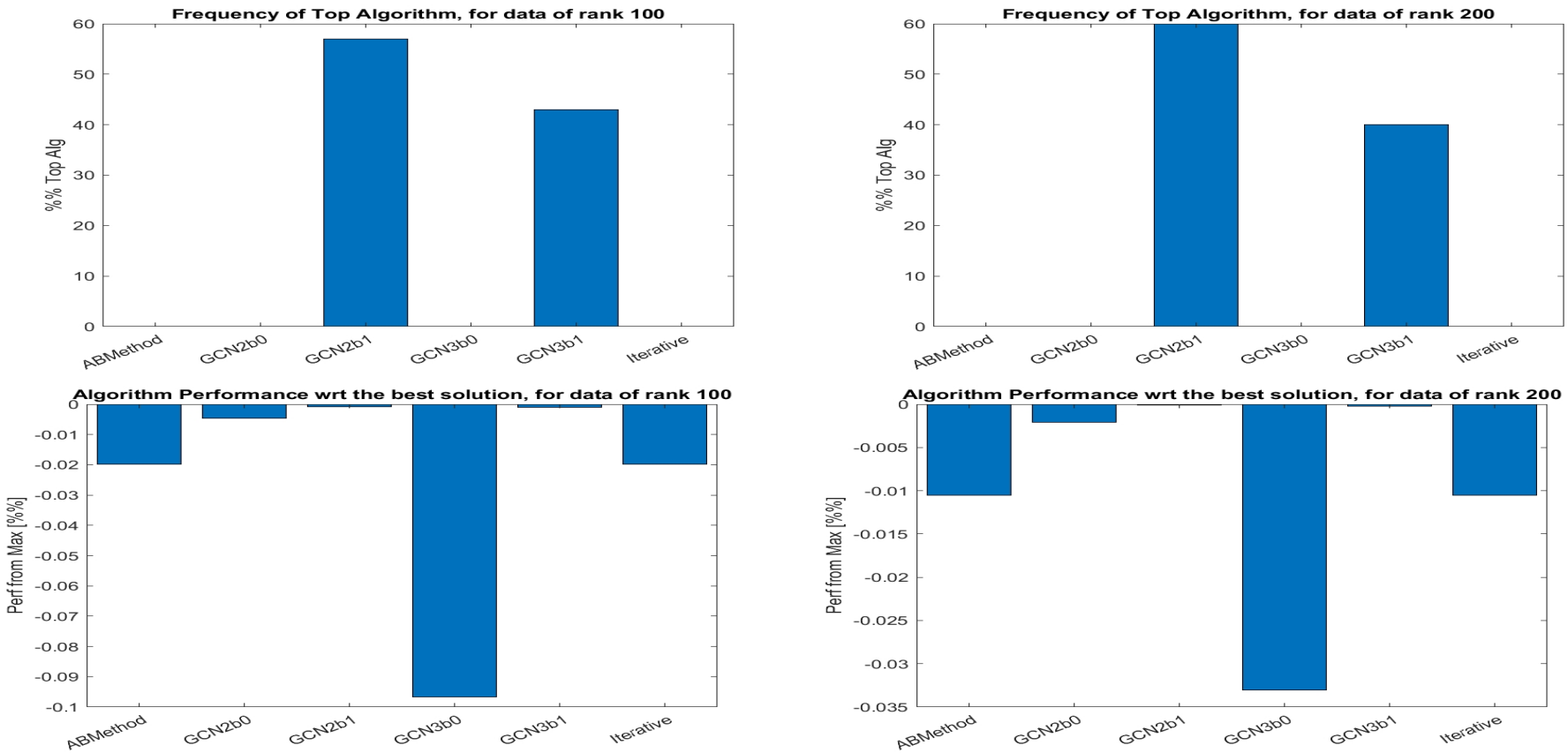


Figure: Top row: Frequency of optimal algorithm for $n = 100$ (left), and $n = 200$ (right). Borrom row: Relative performance [%] to the best algorithm for $n = 100$ (left) and $n = 200$ (right)

Conclusions

The results showed an unexpected result:

- For small n when ground truth is available, the GCN architectures performs comparable to the AB Method and in general worse than the Iterative algorithm. Among the GCN architectures, the 2 layer with bias architecture seems to have a small advantage compared to the other three GCN architectures.
- For large matrix size, the GCN algorithms consistently outperform the AB Method as well as the Iterative algorithm. However the ground truth is not available in these cases. Interestingly, for the case of uniformly $[0, 1]$ case the GCN schemes with no bias provide the best objective function, whereas for the gaussian case the GCN schemes with bias provide the best objective functions. Yet, in all cases, the GCN with bias have the smallest relative difference to the largest objective value in each instance.

Thank you!

Questions?