Artificial Neural Networks for Dimension Reduction and Reduced-Order Modeling

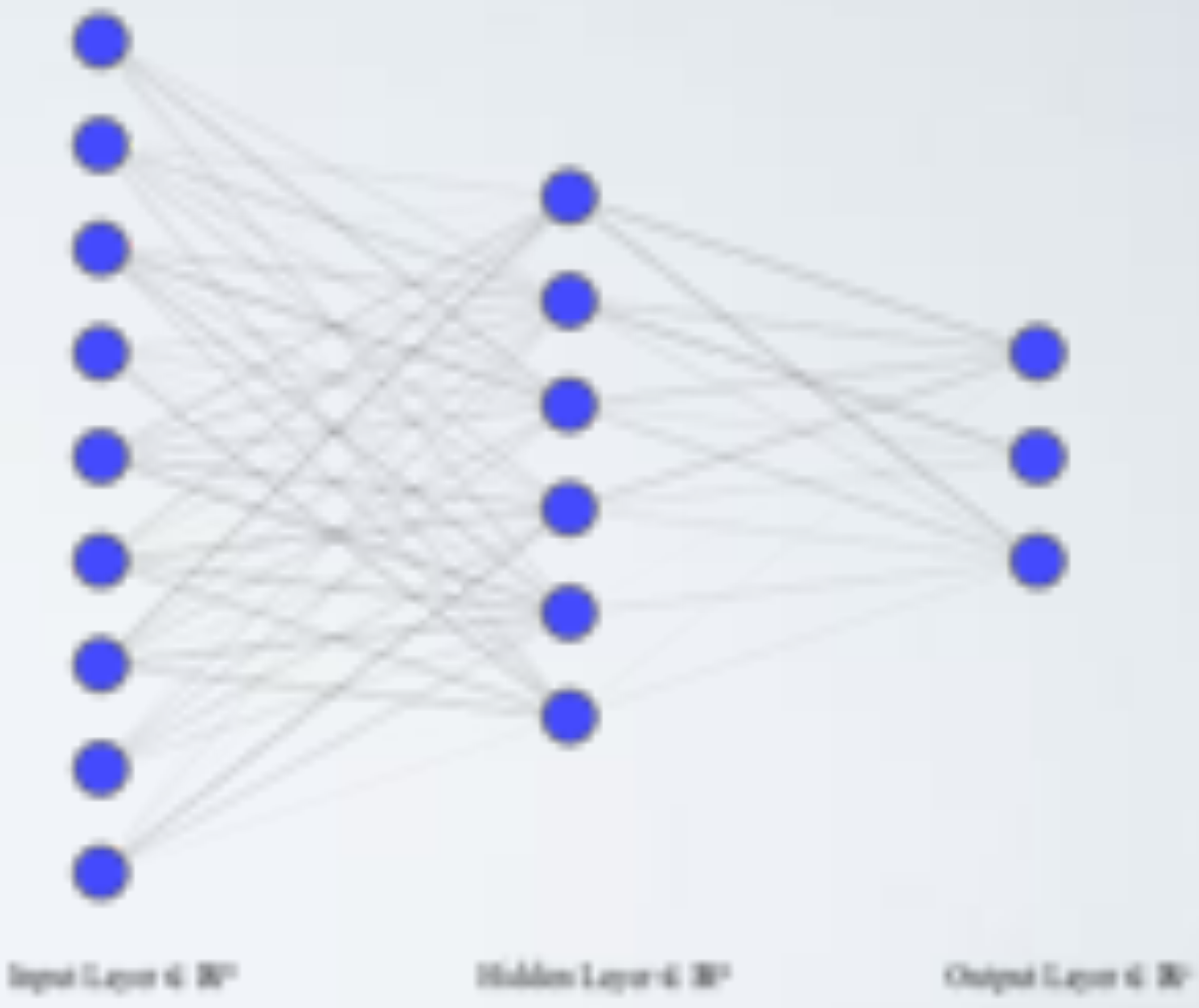
Anthony Gruber

Outline

- Introduction to neural networks
  - Basic architecture and learning procedure
  - Tech demo
- Learning high dimensional functions from sparse data
  (joint with Max Gunzburger, Lili Ju, Zhu Wang, Yuankai Teng)
- Convolutional networks for reduced-order modeling
  (joint with Max Gunzburger, Lili Ju, Zhu Wang)

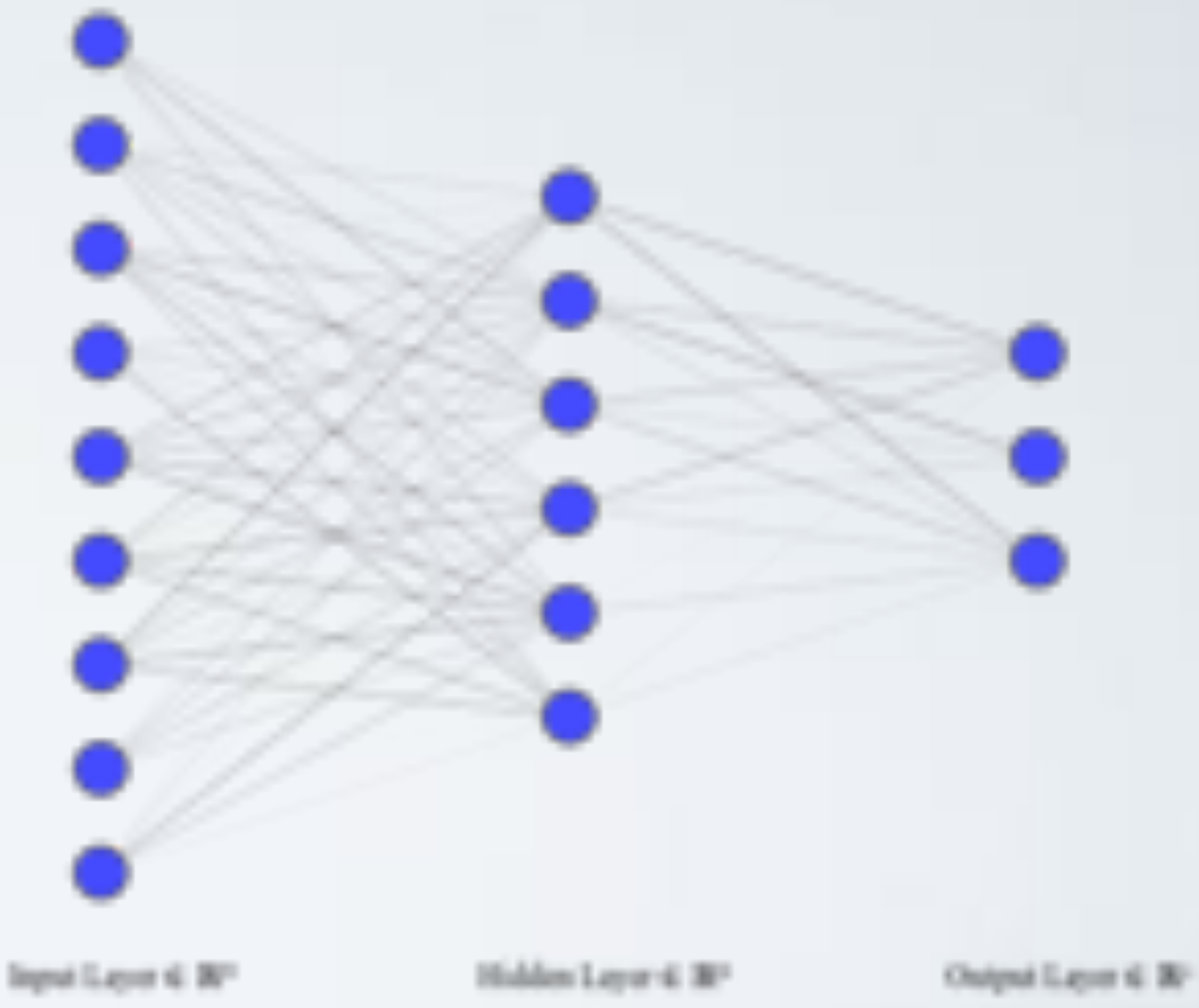
What Is a Neural Network?

- A (fully-connected, feedforward) artificial neural network (NN or ANN) is….
  - a function approximation machine loosely modeled on biological systems.
- Each layer contains neurons (nodes) which contain learnable parameters (edge weights and biases).
- # Nodes = Width,   # Layers = Depth



How Does It Learn?

- Information is passed forward from inputs to outputs.
- Outputs are used to make a prediction, which carries some associated cost.
- Information flows backward through gradient of cost (automatic differentiation).
- Derivatives update learnable parameters through gradient descent.



What Can ANNs Do?

- Extremely powerful for both predictive and generative tasks.
- Highly *overparameterized* (contrast to traditional methods)
- Fact: Two-layer ANNs contain linear FEM! (He et al. 2018)

Potential copyright issue

Wang et al., *CVPR* (2018)

What Can ANNs Do?

- New field; very limited formal understanding.
- Nonlinear and nonconvex optimization — poor robustness.
- **Huge** amount of interest in theory and algorithms.
- Very multidisciplinary; math, stats, compsci, physics, engineering, even social sciences.

+ 2% noise =

Cat

Airplane

Fully Connected Network

- Given inputs $\mathbf{x} = \mathbf{x}_0 \in \mathbb{R}^{N_0}$ and layers $1 \leq \ell \leq L$, a fully connected NN is a function:

$$\mathbf{x}_L = \mathbf{f}_L \circ \mathbf{f}_{L-1} \circ ... \circ \mathbf{f}_1(\mathbf{x})$$

$$\mathbf{x}_\ell = f_\ell(\mathbf{x}_{\ell-1}) = \boldsymbol{\sigma}_\ell \left(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell\right),$$

- Here $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$, $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$ are the weights and biases at layer ℓ, and $\boldsymbol{\sigma}$ is a nonlinear activation function.
  - Ex) Rectified Linear Unit $\boldsymbol{\sigma}(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}) = \max\{\mathbf{x}, 0\}$.

Network Training

- Pass input $\mathbf{x}_0$ forward, generating $\mathbf{y} = \mathbf{x}_L$.
- Evaluate some loss $L(\mathbf{y}) = L(\mathbf{x}, \boldsymbol{\theta})$ depending implicitly on $\mathbf{x}_0$ and some parameters $\boldsymbol{\theta}$.
- Compute derivatives backward using chain rule.
- Update parameters $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - t\, L'(\mathbf{y})\mathbf{y}_{\boldsymbol{\theta}}$ where t is the learning rate.

Backpropagation Algorithm

- In practice, we use backpropogation.
- Backpropagation is an instance of reverse mode automatic differentiation.
- Each variable is associated to a node in the graph.
- Graph is traced backward to obtain gradient information.

Potential copyright issue

Image by Christopher Olah (https://colah.github.io/posts/2015-08-Backprop/)

What Do We Get From This?

- Suppose $\boldsymbol{\sigma}$ is nonpolynomial. A network of the form $\mathbf{y} = \mathbf{W}_2 \boldsymbol{\sigma}_1 \left(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \right)$ can approximate any Borel measurable function to arbitrary accuracy! (Cybenko 1989, Hornik 1991).

- Unfortunately, there is no practical bound on width... In practice, deeper networks still perform better.

- Let's try it out!

Outline

- Introduction to neural networks
  - Basic architecture and learning procedure
  - Tech demo
- **Learning high dimensional functions from sparse data**
  (joint with Max Gunzburger, Lili Ju, Zhu Wang, Yuankai Teng)
- Neural architectures for reduced-order modeling
  (joint with Max Gunzburger, Lili Ju, Zhu Wang, Yuankai Teng)

What About High Dimensional Space?

- Can we still learn a function (e.g. $y = \sin\left(\|\mathbf{x}\|^2\right)$) when the domain is very large?
- Direct methods are not effective without large amounts of data, due to model overfitting.
- Sometimes data is expensive — what then?

Regression in High Dimensions

- One idea is to project the data into a smaller space without collapsing important features.

- Ridge regression looks for a linear $\mathbf{P} : \mathbb{R}^n \to \mathbb{R}^k$ ($k \ll n$) and a function $\hat{f} : \mathbb{R}^k \to \mathbb{R}$ such that $f(\mathbf{x}) \approx \hat{f}(\mathbf{P}\mathbf{x})$.

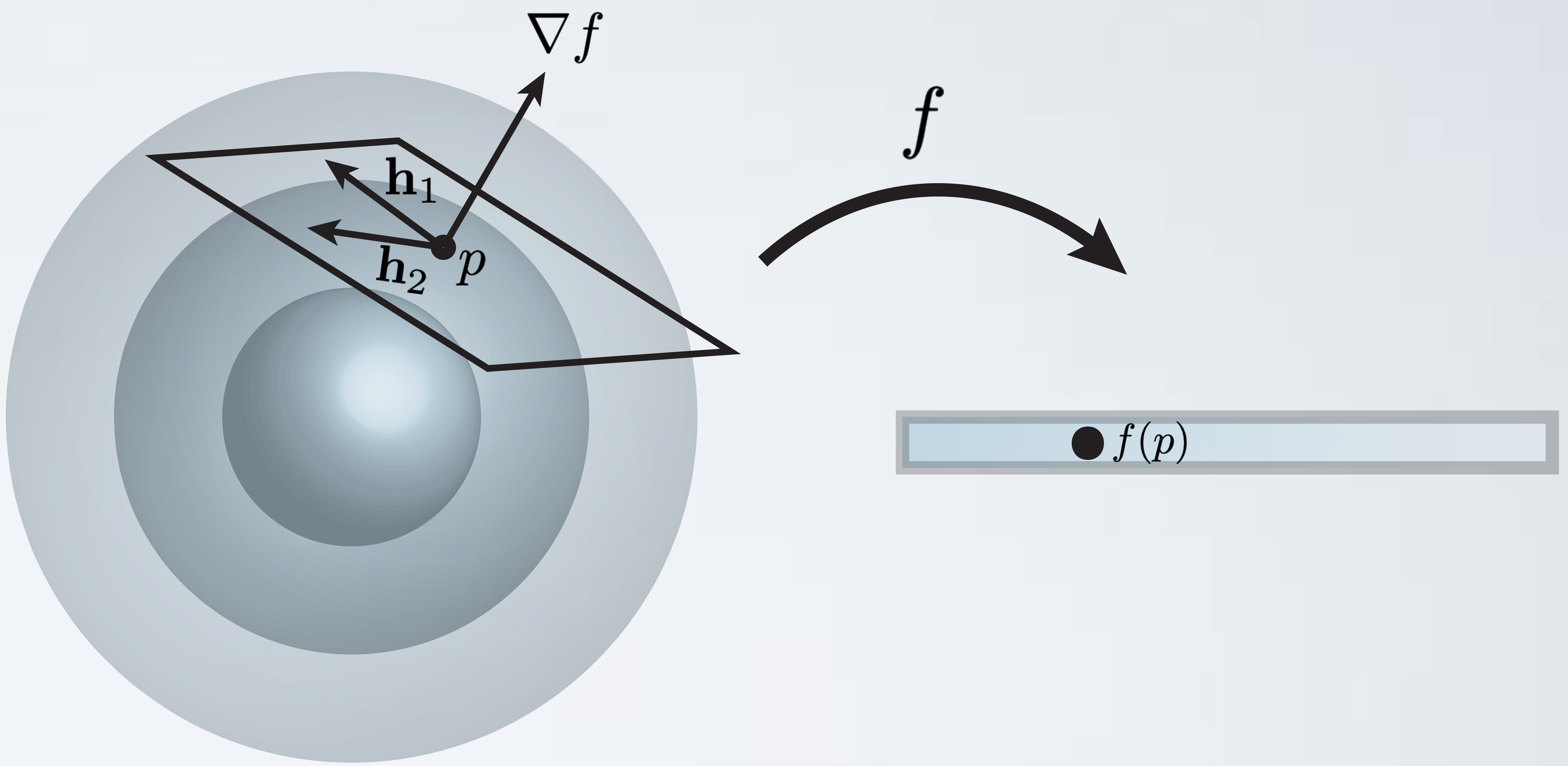
- With neural networks, it is not necessary to consider only linear $\mathbf{P}$.

Nonlinear Level Set Learning

- NLL is a network-based method (Zhang et al. 2019, Gruber et al. 2021), for solving this problem.

- Look for an invertible nonlinear transformation $\mathbf{z} = \mathbf{g}(\mathbf{x}), \mathbf{h} \circ \mathbf{g} = \mathbf{I}$ so that the domain of $f \circ \mathbf{h}$ splits into pairs $(\mathbf{z}_A, \mathbf{z}_I)$ such that $(f \circ \mathbf{h})'(\mathbf{z})\mathbf{e}_i = \mathbf{0}$ for all $i \in I$.

- By the chain rule, this implies $\langle \nabla f, \mathbf{h}_{z_i} \rangle = 0$ for all inactive coords.



Best on Average Coordinates

- Consider $\|(f \circ \mathbf{h})'(\mathbf{z})\|_{\perp}^{2} = \|\langle \nabla f(\mathbf{x}), \mathbf{h}_i(\mathbf{z}) \rangle\|^{2}$.
- Minimizing this means asking for coordinates which integrate the fields $\mathbf{h}_i(\mathbf{z})$ for $i \neq 1$, plus one which is "free".
- This implies the loss functional $L(\mathbf{h}) = \frac{1}{|S|} \sum_{s \in S} \|(f \circ \mathbf{h})'(\mathbf{z}^s)\|_{\perp}^{2}$, where S is the sample set.

Smooth Setting

- It is meaningful to note that this loss is (up to scale) a discretization of the Dirichlet-type energy functional

$$\mathcal{L}(\mathbf{h}) = \int_V \|(f \circ \mathbf{h})'(\mathbf{z})\|_\perp^2 \, d\mu^n = \int_I \int_{Z_t} \|(f \circ \mathbf{h})'(\mathbf{z})\|_\perp^2 \, d\mu^{n-1} \, dt$$

where $\mathbf{h}(V) = U$, $I = \pi_1(V)$ is a bounded interval containing the range of z_1 and $Z_t = \{\mathbf{z} \in V \,|\, z^1 = t\}$.

Smooth Setting

- Standard techniques in the calculus of variations imply

$$\delta\mathcal{L}(\mathbf{h})\boldsymbol{\varphi} = -2\int_I \int_{Z_t} (f \circ \boldsymbol{\varphi})(\mathbf{z})\Delta^{\perp}(f \circ \mathbf{h})(\mathbf{z})\, d\mu^{n-1}\, dt$$

- Critical iff $\Delta^{\perp}(f \circ \mathbf{h}) = 0$ i.e. $f \circ \mathbf{h}$ is harmonic on the slices Z_t (obviously true in ideal case).

Reversible Neural Networks

- RevNets (Gomez et al. 2017) are an invertible modification of Residual Neural Networks (ResNets), which are built from blocks of the form $\mathbf{y} = \mathbf{x} + \mathbf{F}(\mathbf{x})$.

- ResNets allow very deep networks, and are SOTA in some image classification tasks.

- RevNet blocks: $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$ such that
$$\mathbf{y}_1 = \mathbf{x}_1 + \mathbf{F}(\mathbf{x}_2)$$
$$\mathbf{y}_2 = \mathbf{x}_2 + \mathbf{G}(\mathbf{y}_1)$$

Network Architecture

- Since $\mathbf{h}$ should be a diffeomorphism, we use a RevNet due to (Chang et al. 2018) inspired by Hamiltonian systems.

- Straightforward techniques show the stability of the ODE system

$$\dot{\mathbf{u}}(t) = \mathbf{W}_1^T(t)\boldsymbol{\sigma}\left(\mathbf{W}_1(t)\mathbf{v}(t) + \mathbf{b}_1(t)\right),$$
$$\dot{\mathbf{v}}(t) = -\mathbf{W}_2^T(t)\boldsymbol{\sigma}\left(\mathbf{W}_2(t)\mathbf{u}(t) + \mathbf{b}_2(t)\right),$$

- So, this can be used for forward propagation along a network.

Network Architecture

- Let $\mathbf{x} = (\mathbf{u}_0, \mathbf{v}_0)$, then for layers $1 \leq \ell \leq L$ we have

$$\mathbf{u}_{\ell+1} = \mathbf{u}_\ell + \tau\, \mathbf{W}_{\ell,1}^T\, \boldsymbol{\sigma}\left(\mathbf{W}_{\ell,1}\mathbf{v}_\ell + \mathbf{b}_{\ell,1}\right),$$

$$\mathbf{v}_{\ell+1} = \mathbf{v}_\ell - \tau\, \mathbf{W}_{\ell,2}^T\, \boldsymbol{\sigma}\left(\mathbf{W}_{\ell,2}\mathbf{u}_{\ell+1} + \mathbf{b}_{\ell,2}\right).$$

- Defining $\mathbf{z} = (\mathbf{u}_L, \mathbf{v}_L)$ yields the mapping $\mathbf{g}$.

- Both $\mathbf{g}, \mathbf{h}$ are parameterized by $\mathbf{W}_{\ell,i}, \mathbf{b}_{\ell,i}$ (weight sharing).

Aside: Active Subspaces

- The NLL method can be considered a nonlinear substitute for the Active Subspaces (AS) method (Constantine 2013).

- AS approximates the covariance matrix $\mathbf{C} = \mathbb{E}[\nabla f (\nabla f)^T = \int_U \nabla f \, (\nabla f)^T \, d\mu^n$

- The first columns of $\mathbf{U}$ in $\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ give a basis for the active subspace.

- Function approximation takes place in this smaller space as before.

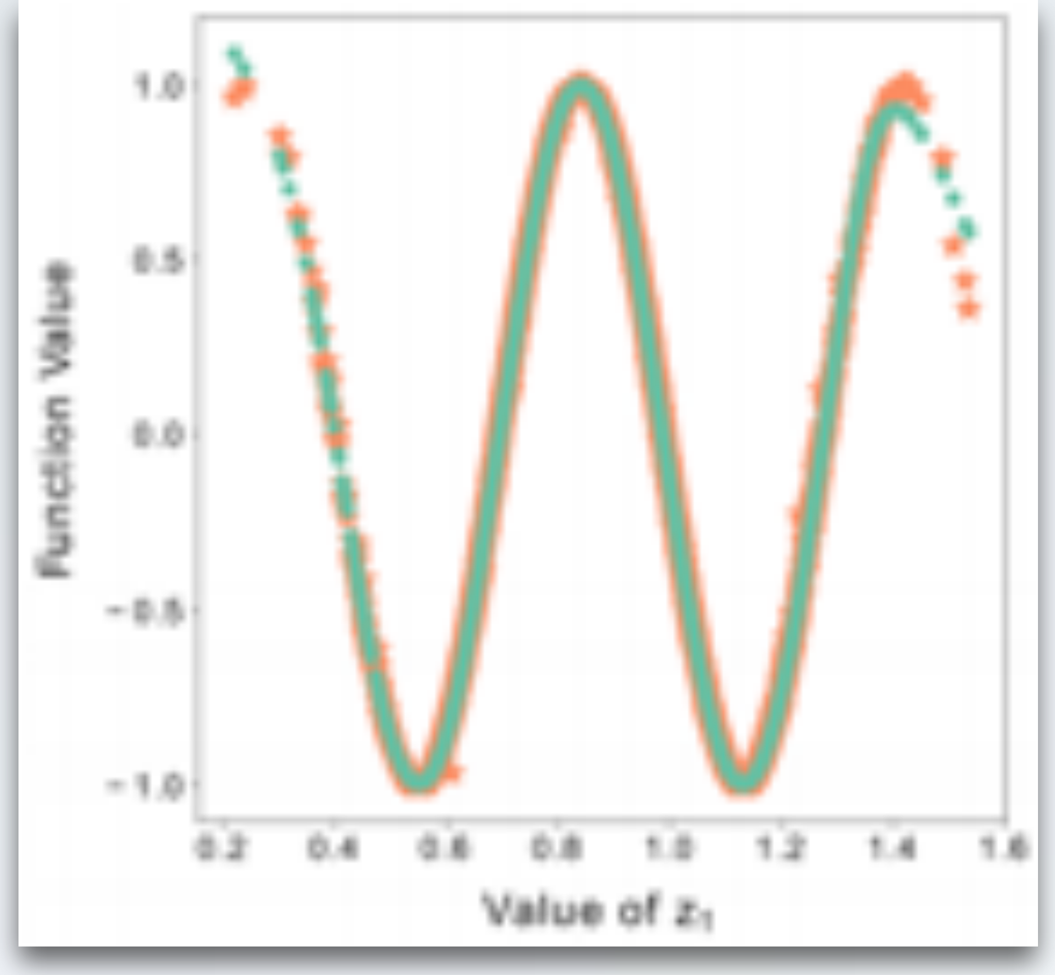
NLL on Toy Functions

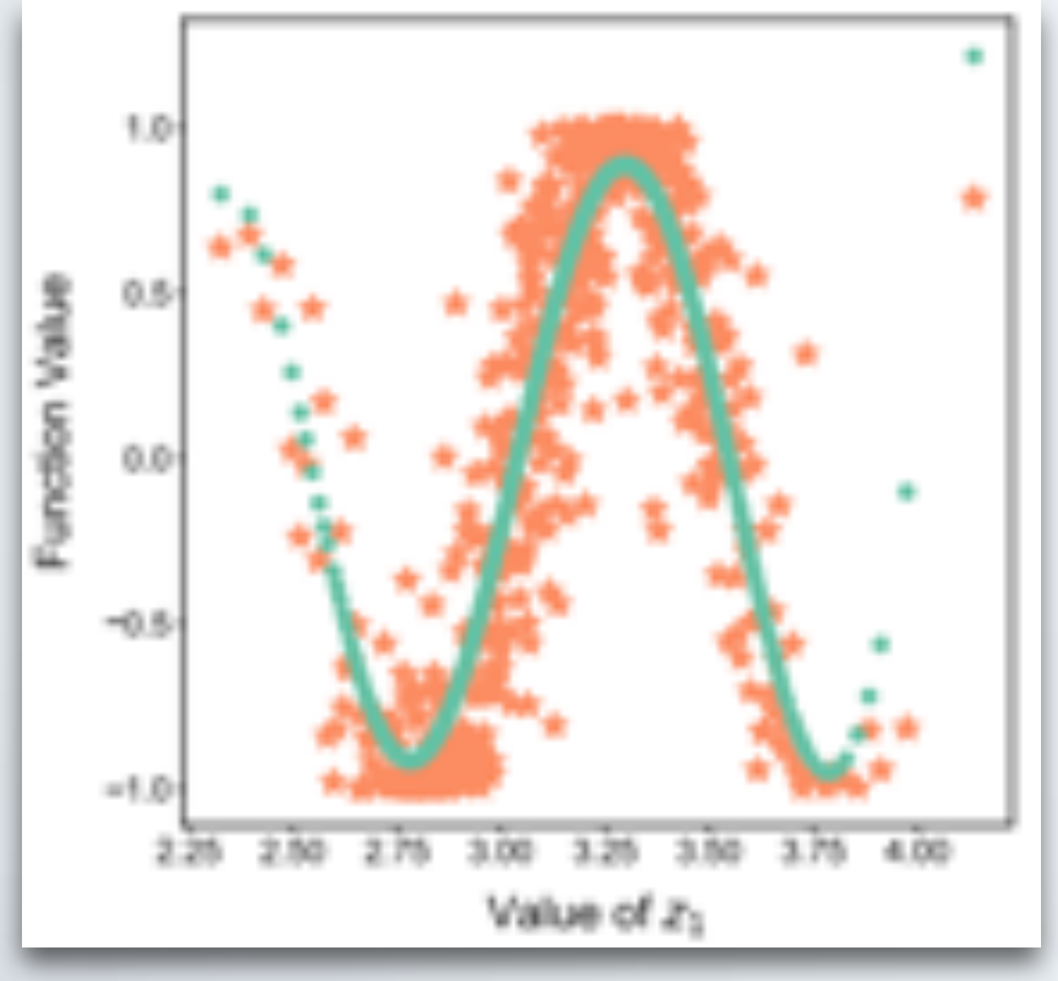
- Domain: $[0,1]^n$. Functions: $f_4(\mathbf{x}) = \sin\left(\|\mathbf{x}\|^2\right)$, $f_5(\mathbf{x}) = \prod_{i=1}^{20} \frac{1}{1+x_i^2}$.
- Sensitivity: Magnitude of $(f \circ \mathbf{h})'(\mathbf{z})\mathbf{e}_1$ as percentage of total.
- Relative L^i error: $\frac{\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\hat{\mathbf{x}})\|_i}{\|\mathbf{f}(\mathbf{x})\|_i}$ (**f** is vector of samples)

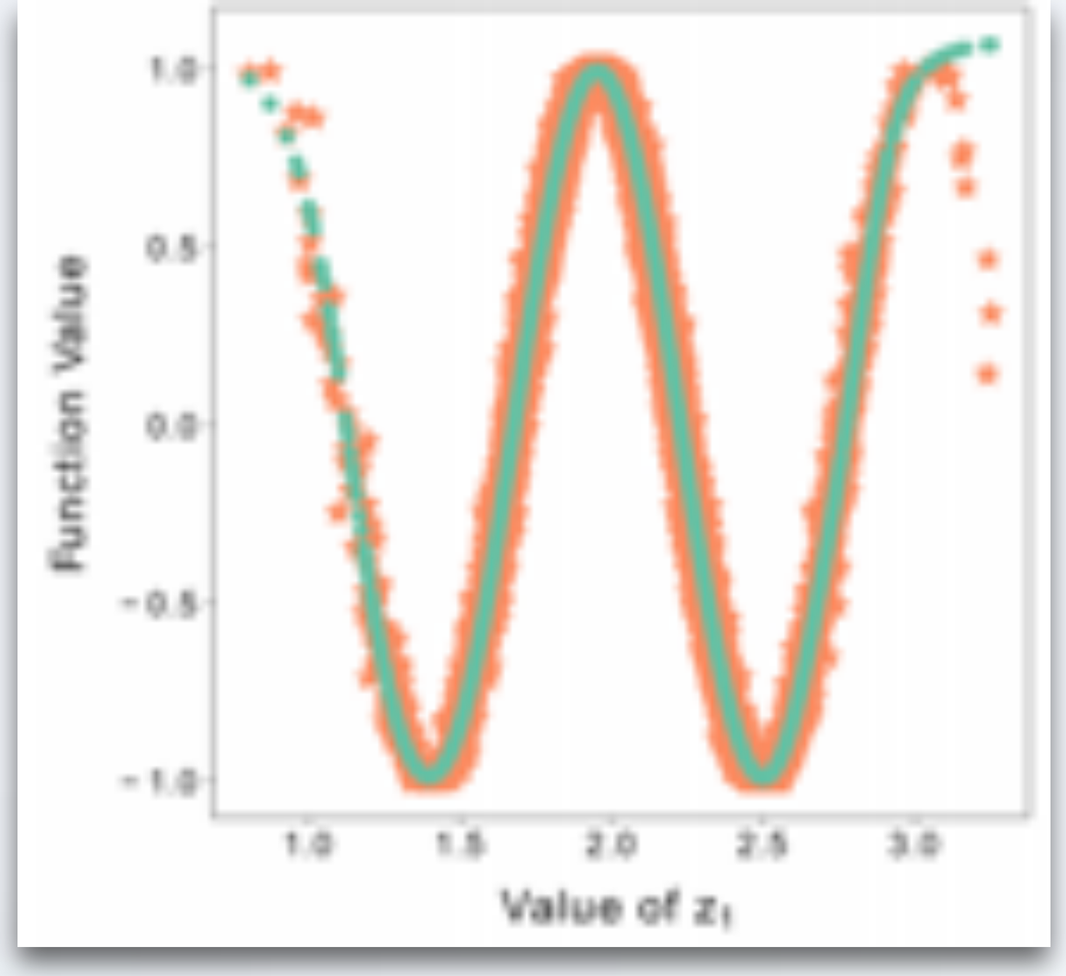
		100 Samples				500 Samples				2500 Samples			
Function	Method	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %
f_4	New NLL	78.7	3.86	8.27	10.9	89.8	1.82	3.52	5.16	94.5	0.827	1.72	2.35
	Old NLL	60.4	6.63	14.5	18.8	65.9	4.58	10.5	13.0	69.2	4.02	9.11	11.4
	AS 1-D	25.8	30.3	75.9	85.9	25.9	21.7	39.5	61.4	25.9	15.9	37.6	44.8
f_5	New NLL	75.1	0.920	5.79	7.92	88.6	0.370	2.78	3.97	93.8	0.154	1.63	1.98
	Old NLL 1	54.6	0.699	7.48	9.40	55.4	0.942	7.26	9.52	56.1	0.784	6.91	8.05
	Old NLL 2	61.8	1.80	12.9	21.1	68.7	1.03	9.22	11.1	67.5	0.894	8.16	9.69

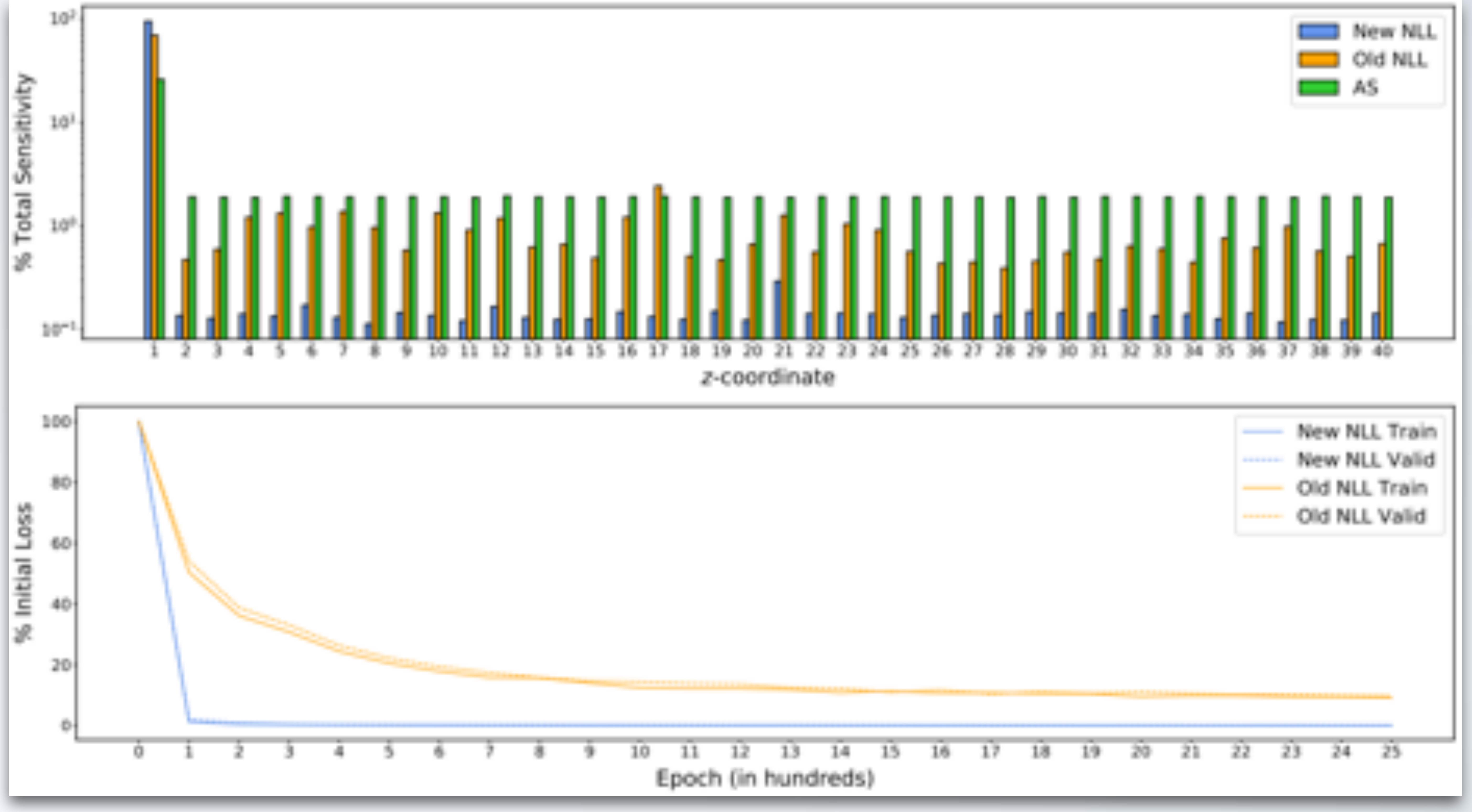
NLL on Toy Functions

- On a 40 dimensional sine wave









A. Gruber, M. Gunzburger, L. Ju, Y. Teng, Z. Wang, *Numer. Math. Theor. Meth. Appl.* (2021)

Predicting Kinetic Energy

- Consider the 1-D parametrized inviscid Burger's equation on [a,b], where $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3)$, $w = w(x, t, \boldsymbol{\mu})$.

$$w_t + \frac{1}{2}\left(w^2\right)_x = \mu_3 e^{\mu_2 x},$$
$$w(a, t, \boldsymbol{\mu}) = \mu_1,$$
$$w(x, 0, \boldsymbol{\mu}) = 1,$$

- It is useful to know the total kinetic energy at time *t*:

$$K(t, \boldsymbol{\mu}) = \frac{1}{2}\int_0^t \int_a^b w(x, \tau, \boldsymbol{\mu})^2 \, dx \, d\tau$$

$$\nabla K(t, \boldsymbol{\mu}) = (K_t \;\; K_{\boldsymbol{\mu}})^\intercal = \begin{pmatrix} \frac{1}{2}\int_a^b w(x, t, \boldsymbol{\mu})^2 \, dx & \int_0^t \int_a^b w(x, \tau, \boldsymbol{\mu}) \, w_{\boldsymbol{\mu}}(x, \tau, \boldsymbol{\mu}) \, dx \, d\tau \end{pmatrix}^\intercal$$

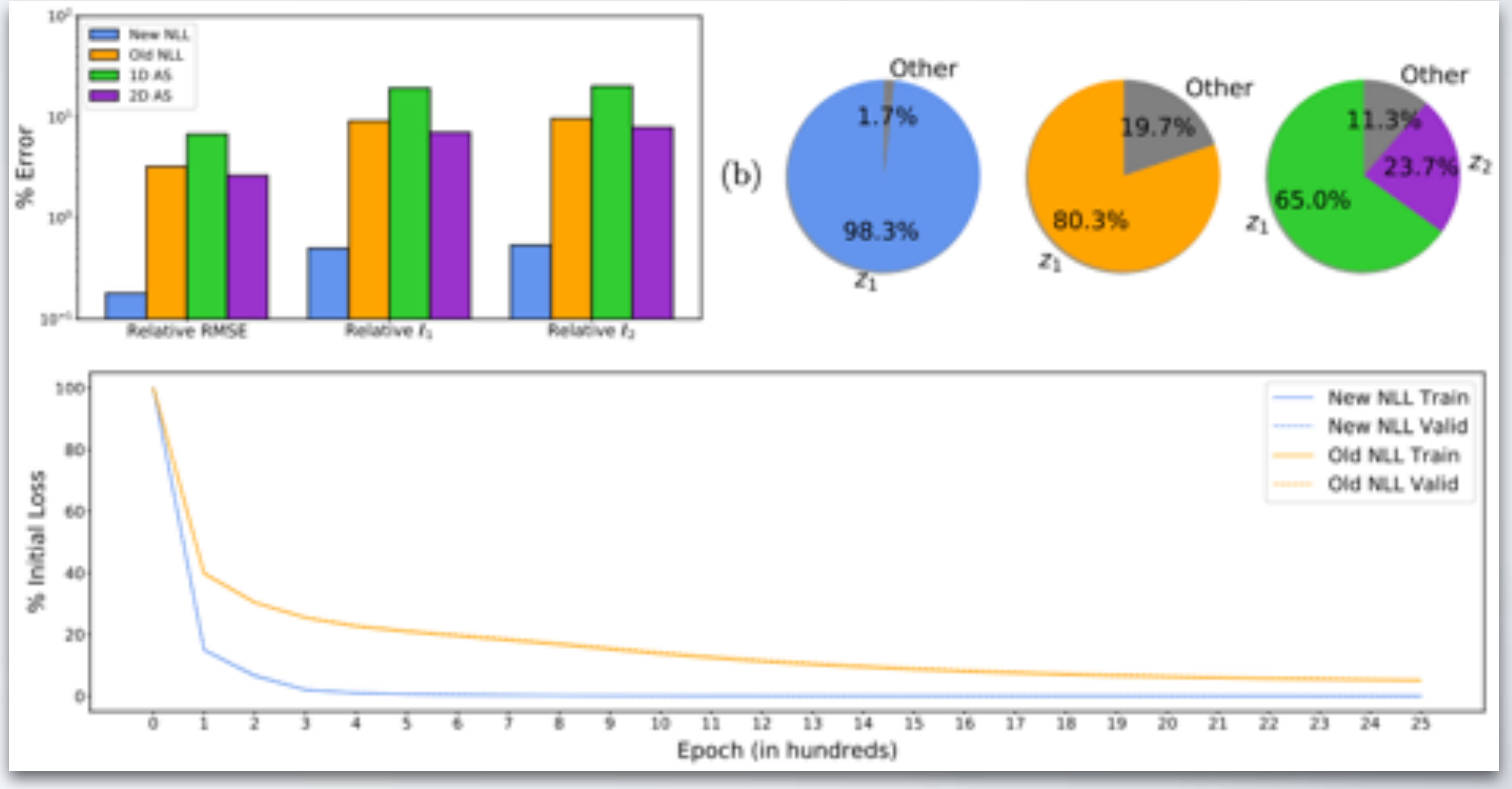
Predicting Kinetic Energy

- Can compute gradients by solving sensitivity equations:

$$w_{\boldsymbol{\mu},t} + (w w_{\boldsymbol{\mu}})_x = \begin{pmatrix} 0 & x\mu_3 e^{\mu_2 x} & e^{\mu_2 x} \end{pmatrix}^\intercal$$
$$w_{\boldsymbol{\mu}}(a,t,\boldsymbol{\mu}) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^\intercal$$
$$w_{\boldsymbol{\mu}}(x,0,\boldsymbol{\mu}) = \mathbf{0}.$$

- Can then run algorithm on samples $\{(t,\boldsymbol{\mu}), K(t,\boldsymbol{\mu}), \nabla K(t,\boldsymbol{\mu})\}$

- Forward Euler with upwinding used to solve systems.



Outline

- Introduction to neural networks
    - Basic architecture and learning procedure
    - Tech demo
- Learning high dimensional functions from sparse data
  (joint with Max Gunzburger, Lili Ju, Zhu Wang, Yuankai Teng)
- **Convolutional networks for reduced-order modeling**
  (joint with Max Gunzburger, Lili Ju, Zhu Wang, Yuankai Teng)

Reduced Order Modeling

- High-fidelity PDE simulations are expensive.
  - Semi-discretization creates a lot of dimensionality.
- Can we get good results without solving the full PDE?
- Standard is to **encode -> solve -> decode.**
  - This way, solving is low-dimensional.

Potential copyright issue

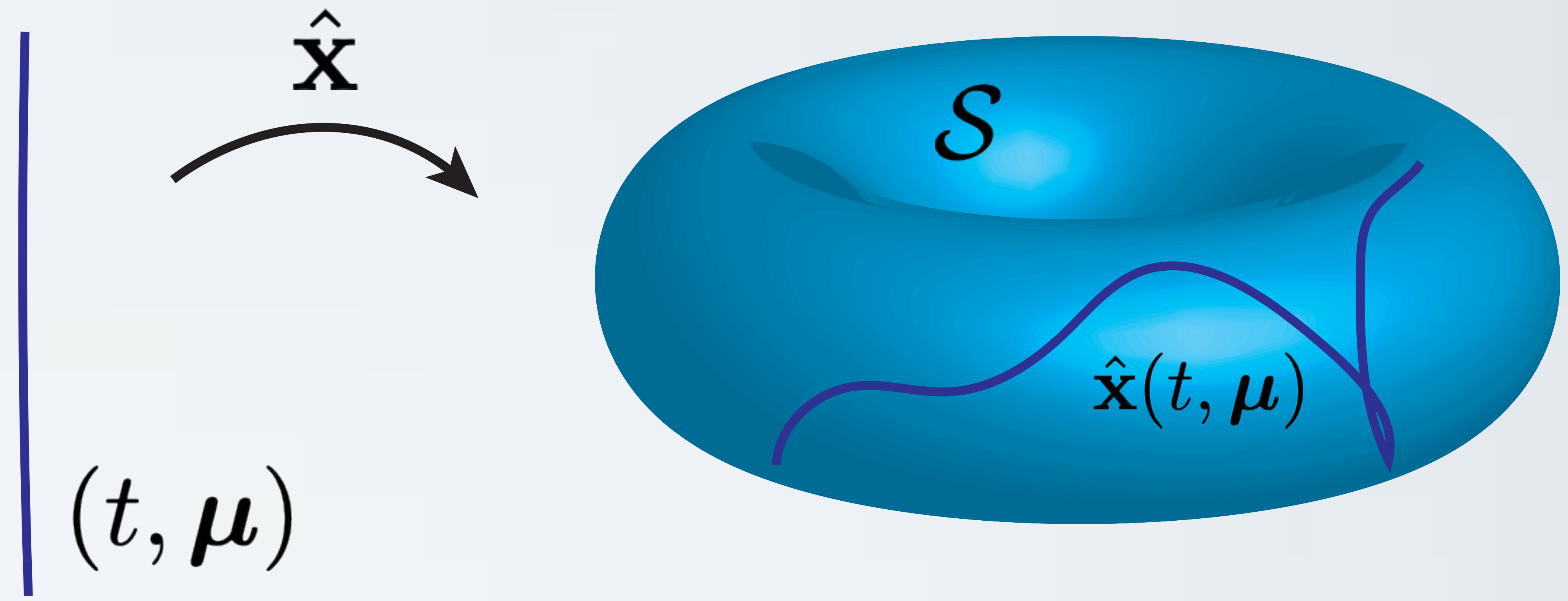
Image: https://mpas-dev.github.io/ocean/ocean.html

Full-Order Model

- FOM: $\dot{\mathbf{x}}(t,\boldsymbol{\mu}) = \mathbf{f}\left(t, \mathbf{x}(t), \boldsymbol{\mu}\right), \quad \mathbf{x}(0,\boldsymbol{\mu}) = \mathbf{x}_0(\boldsymbol{\mu})\,.$
    - $\boldsymbol{\mu}$ is vector of parameters.
    - Dimension of $\mathbf{x}$ can be huge — on the order of 10^4 to 10^6 or more.
- Typically solved with time integrator e.g. Runge-Kutta.
- Recently, neural networks used instead.
    - Difficult due to high dimensionality.

Drawback of FOM

- Do we really need all 10^6 dimensions?
- No, if $(t, \boldsymbol{\mu}) \mapsto \mathbf{x}(t, \boldsymbol{\mu})$ is unique.
  - $\mathcal{S} = \{\mathbf{x}(t, \boldsymbol{\mu}) \mid t \in [0,T], \boldsymbol{\mu} \in D\} \subset \mathbb{R}^N$, solution manifold.
  - $(n_\mu + 1)$ dimensions is enough for loss-less representation of $\mathcal{S}$.
- How can we recover $\mathcal{S}$ efficiently?



Reduced Order Model

- Consider finding $\tilde{\mathbf{x}}$ s.t. $\mathbf{x} \approx \tilde{\mathbf{x}} = \mathbf{g} \circ \hat{\mathbf{x}}$
  - $(t, \boldsymbol{\mu}) \mapsto \hat{\mathbf{x}}(t, \boldsymbol{\mu}) \in \mathbb{R}^n$ where $n \ll N$.
  - If $n \geq n_\mu + 1$, image of $\hat{\mathbf{x}}$ is potentially "big enough" to encode $\mathbf{x}$.
- $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^N$ a decoder function.
  - e.g. linear projection; NN autoencoder.

Reduced Order Model

- Suppose $\tilde{\mathbf{x}}$ obeys same dynamics as $\mathbf{x}$.
- Residual $||\dot{\tilde{\mathbf{x}}} - f(\tilde{\mathbf{x}})||^2$ is minimized when:
  - $\dot{\hat{\mathbf{x}}}(t, \boldsymbol{\mu}) = \mathbf{g}'(\hat{\mathbf{x}})^{+}\mathbf{f}\left(t, \mathbf{g}(\hat{\mathbf{x}}), \boldsymbol{\mu}\right), \quad \hat{\mathbf{x}}(0,\boldsymbol{\mu}) = \mathbf{h}(\mathbf{x}_0(\boldsymbol{\mu})),$
    - Here $\mathbf{g}'(\hat{\mathbf{x}})^{+}$ is the pseudoinverse of $\mathbf{g}'$.
    - $\mathbf{h} : \mathbb{R}^N \to \mathbb{R}^n$ left inverse of $\mathbf{g}$.
- ODE of size N converted to ODE of size n.
- "Hard part" is computing the decoder function $\mathbf{g}$.

Proper Orthogonal Decompostion (POD)

- Most popular (until recently) is *proper orthogonal decomposition.*
- Carry out PCA on solution snapshots $\{\mathbf{u}(t_j, \mathbf{x}, \boldsymbol{\mu}_j)\}_{j=1}^{N}$, generate matrix $\mathbf{S}$.
- SVD: $\mathbf{S} = \mathbf{U\Sigma V}$.
  - First n cols of $\mathbf{U}$ (say $\mathbf{A}$) — reduced basis of POD modes.
  - $\mathbf{g} = \mathbf{A}$ is linear, so $\mathbf{g}' = \mathbf{A}$.
- Instead of $\dot{\mathbf{x}} = f(\mathbf{x})$, can then solve $\dot{\hat{\mathbf{x}}} = \mathbf{A}^{+}\mathbf{f}(\mathbf{A}\hat{\mathbf{x}})$.

POD Versus ANN

- POD works well until EWs of $\boldsymbol{\Sigma}$ decay slowly.
  - Even many modes cannot reliably capture behavior.
- Conversely, FCNN/CNN captures patterns quite well.
- Are all NNs equal for this purpose?

Copyright

Lee, K. and Carlberg, K. *J. Comp. Phys.* (2019)

CNN Model Order Reduction

- Lee and Carlberg (2019) used a convolutional neural network (CNN).
  - Demonstrated greatly improved performance over POD.
- Convolution extracts high-level features which are used in encoding.

Potential copyright issue

K. Lee and K. T. Carlberg, *J. Comp. Phys.*, 2019

Disadvantages of CNN

- Recall:

$$\mathbf{y}_{\ell,i} = \boldsymbol{\sigma}_{\ell}\left(\sum_{j=1}^{C_{in}} \mathbf{y}_{(\ell-1),j} \star \mathbf{W}^{j}_{\ell,i} + \mathbf{b}_{\ell,i}\right), \quad \text{where} \;\; 1 \leq i \leq C_{out}\,.$$

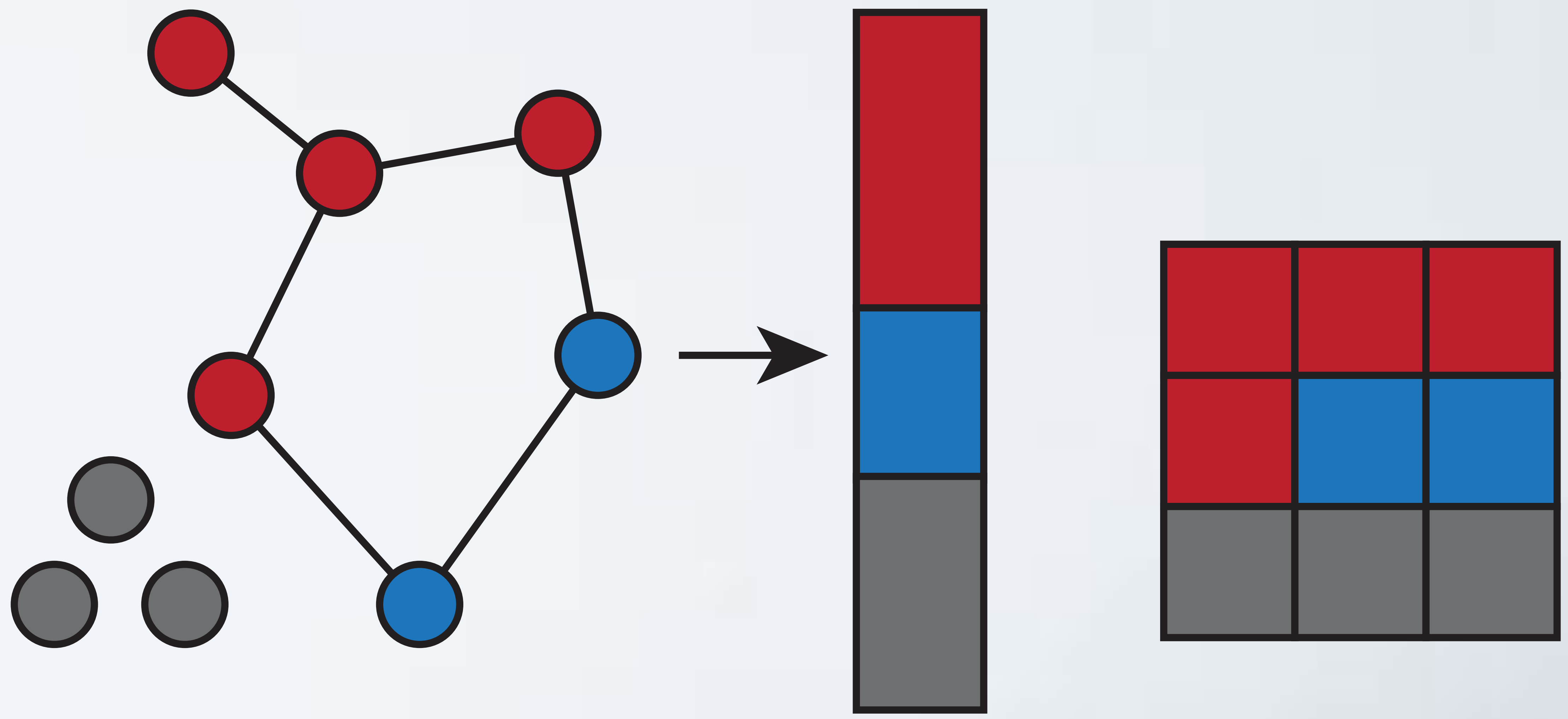
- Convolution $\star$ in 2-D:

$$(\mathbf{x} \star \mathbf{W})^{\alpha}_{\beta} = \sum_{\gamma,\delta} x^{(s\alpha+\gamma)}_{(s\beta+\delta)}\, w^{(L-1-\gamma)}_{(M-1-\delta)},$$

- Only well defined (in this form) for regular domains!

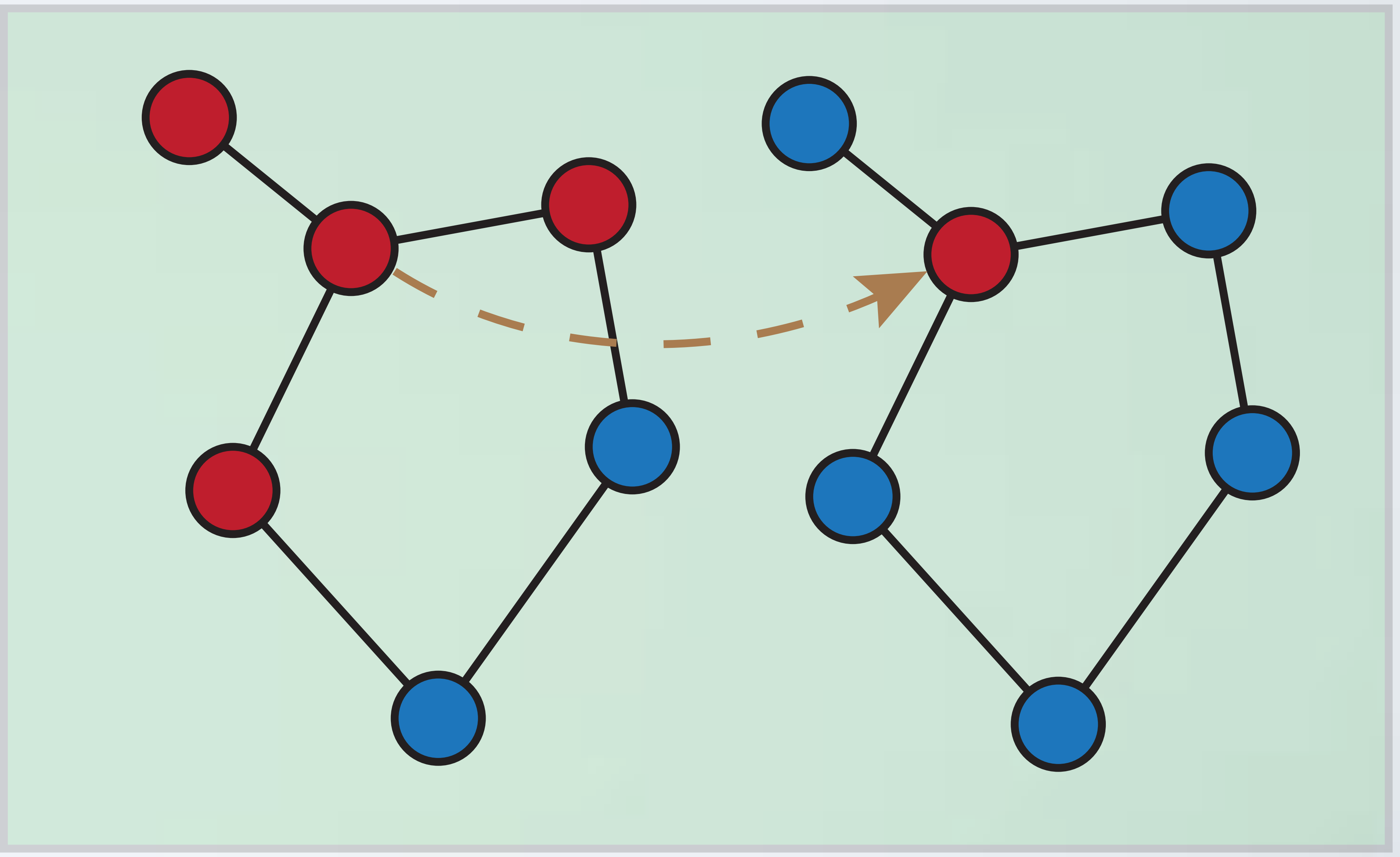
Disadvantages of CNN

- How to use CNN on irregular data?
- Current strategy is to ignore the problem:
  - inputs $\mathbf{y}$ padded with fake nodes and reshaped to a square.
  - Convolution applied to square-ified input.
  - $\tilde{\mathbf{y}}$ reassembled at end. Fake nodes ignored.
- Works surprisingly well!
  - But, not very meaningful.



Graph Convolutional Networks

- Huge amount of recent work extending convolution to graph domains.
- Suppose $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph with adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$.
- Let $\mathbf{D}$ be the degree matrix $d_{ii} = \sum_j a_{ij}$.
- The Laplacian of $\mathcal{G}$: $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^\top$.
  - Columns of $\mathbf{U}$ are Fourier modes of $\mathcal{G}$.
  - Discrete FT/IFT: simply multiply by $\mathbf{U}^\top / \mathbf{U}$.



Graph Convolutional Networks

- Let $\mathbf{y}_i : \mathbb{R}^{|\mathscr{V}|} \to \mathbb{R}$ signals defined at nodes.
- Convolution theorem: $\mathbf{y}_1 \star \mathbf{y}_2 = \mathbf{U}\left(\mathbf{U}^\top \mathbf{y}_1 \odot \mathbf{U}^\top \mathbf{y}_2\right)$.
- Well defined on any domain without reference to local neighborhoods.
- Learnable spectral filters: $g_\theta(\mathbf{L})\mathbf{y} = \mathbf{U} g_\theta(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{y}$ where $g_\theta(\mathbf{\Lambda}) = \sum \theta_k \mathbf{\Lambda}^k$.
  - Degree K filters are precisely K-localized on $\mathscr{G}$! (not obvious)

Graph Convolutional Networks

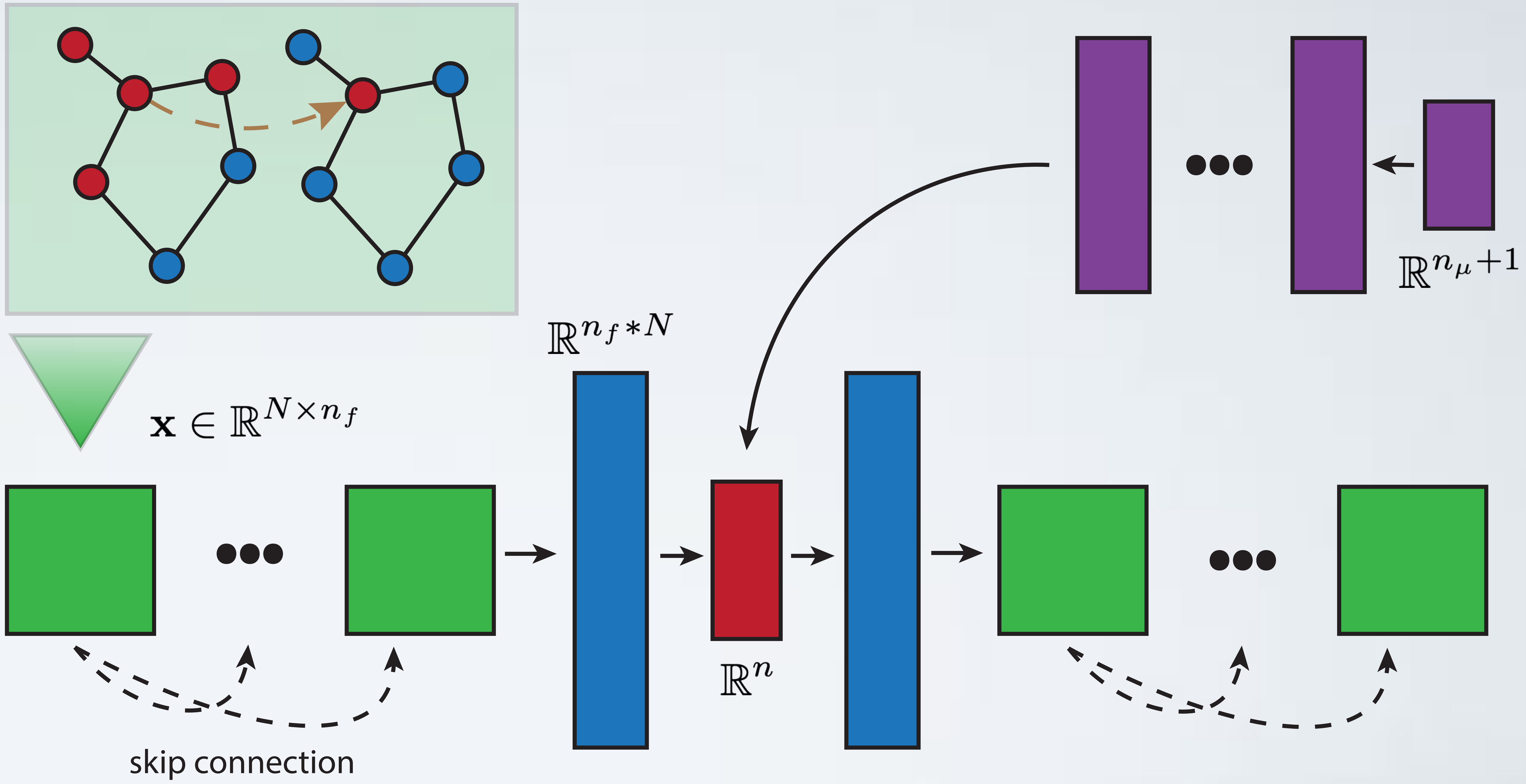
- Let $\tilde{\mathbf{P}} = (\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}$ (renormalized Laplacian).
- Simplified 1-localized GCN (Kipf and Welling 2016): $\mathbf{x}_{\ell+1} = \boldsymbol{\sigma}\left(\tilde{\mathbf{P}}\mathbf{x}_\ell \mathbf{W}\right)$.
- Good performance on small-scale classification tasks, but known for *oversmoothing*.
- (Chen et al. 2020) proposed GCN2, adding residual connection and identity map:

$$\mathbf{x}_{\ell+1} = \boldsymbol{\sigma}\left[\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{x}_\ell + \alpha_\ell \mathbf{x}_0\right)\left((1-\beta_\ell)\mathbf{I} + \beta_\ell \mathbf{W}_\ell\right)\right],$$

  - α_ℓ, β_ℓ — hyperparameters.
  - Equivalent to a degree L polynomial filter with arbitrary coefficients.

GC Autoencoder Architecture

- GCN2 layers encode-decode.
- Blue layers are fully connected.
- For ROM: purple network simulates low-dim dynamics.
- Split network idea due to (Fresca et al. 2020).



Training Details

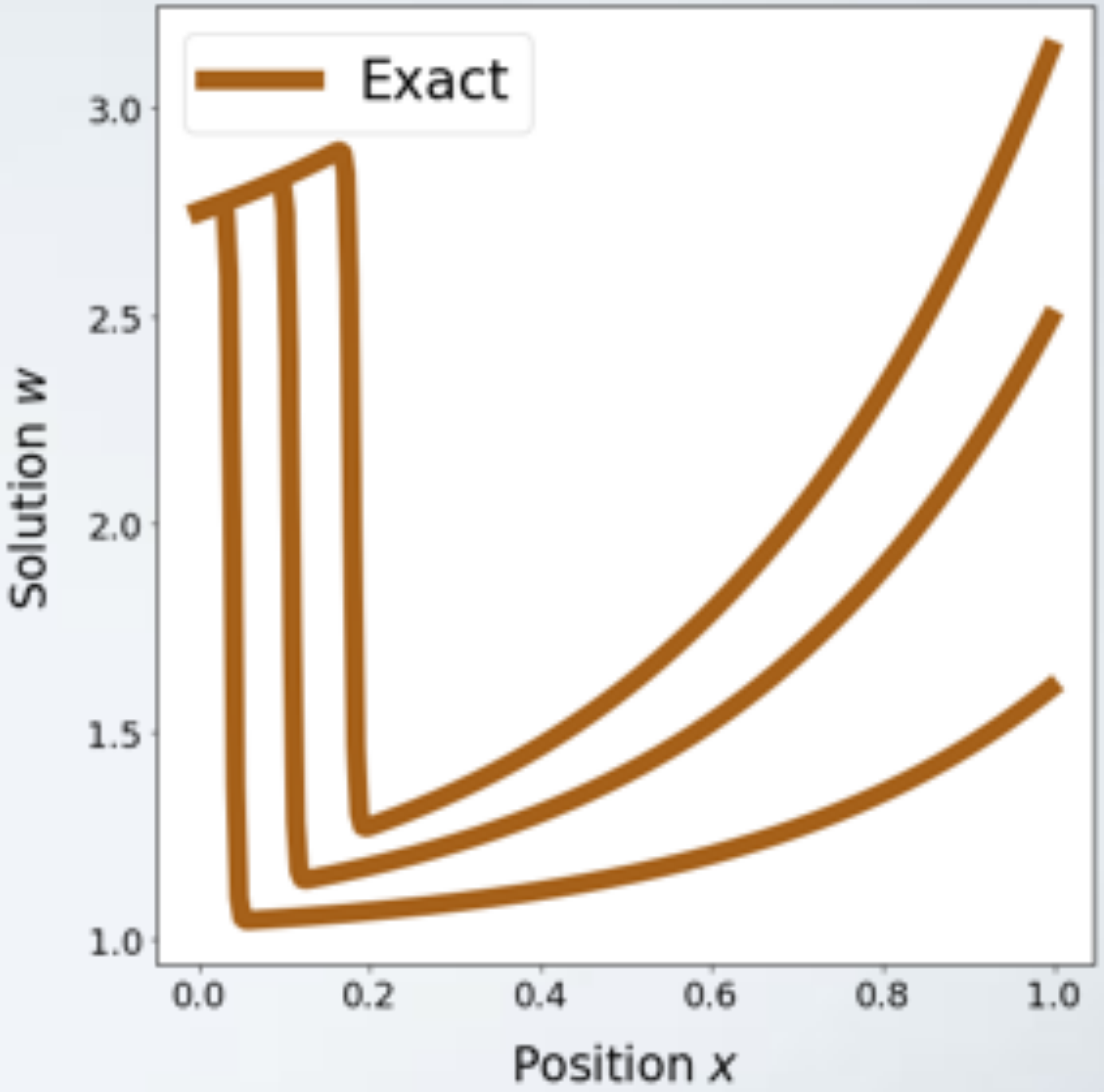
- ROM loss used: $L(\mathbf{x}, t, \boldsymbol{\mu}) = |\mathbf{x} - \mathbf{g} \circ \hat{\mathbf{x}}|^2 + |\mathbf{h} - \hat{\mathbf{x}}|^2$.
  - First term reconstructs solution from parameters.
  - Second term ties encoder and prediction network.
- Compression loss: $L(\mathbf{x}, t, \boldsymbol{\mu}) = |\mathbf{x} - \mathbf{g} \circ \mathbf{h}|^2$.
  - Used when evaluating only compression/decompression ability.
- Network trained using mini-batch descent with ADAM optimizer.
- Training done on a lattice of values $\boldsymbol{\mu}$, testing done on centers.

1-D Inviscid Burger's Equation

- Let $w = w(x, t, \boldsymbol{\mu})$ and consider:

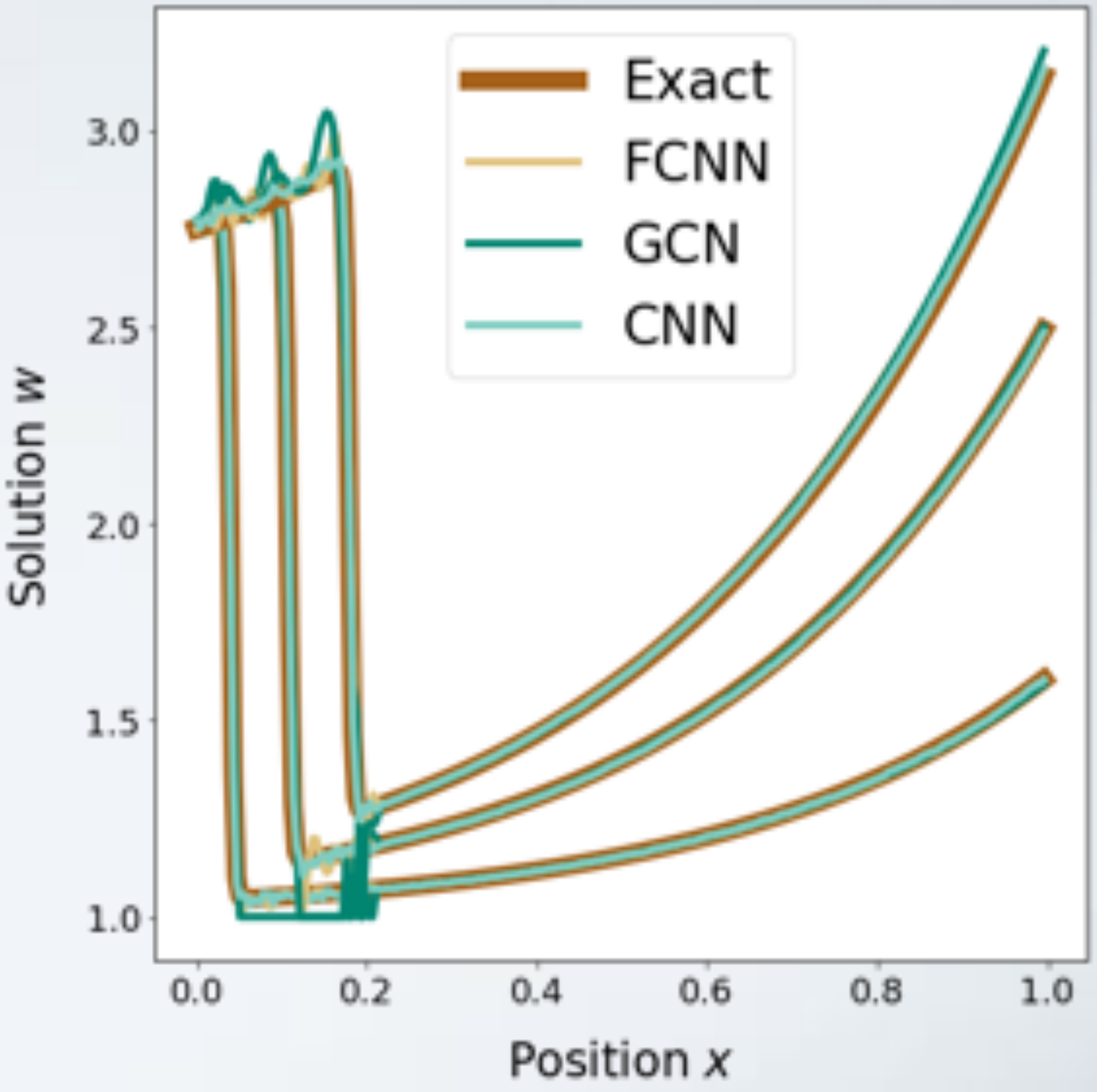
$$w_t + \frac{1}{2}\left(w^2\right)_x = 0.02e^{\mu_2 x},$$
$$w(a, t, \boldsymbol{\mu}) = \mu_1,$$
$$w(x, 0, \boldsymbol{\mu}) = 1,$$

- Want to predict semi-discrete solution $\mathbf{w} = \mathbf{w}(t, \boldsymbol{\mu})$ at any desired parameter configuration.



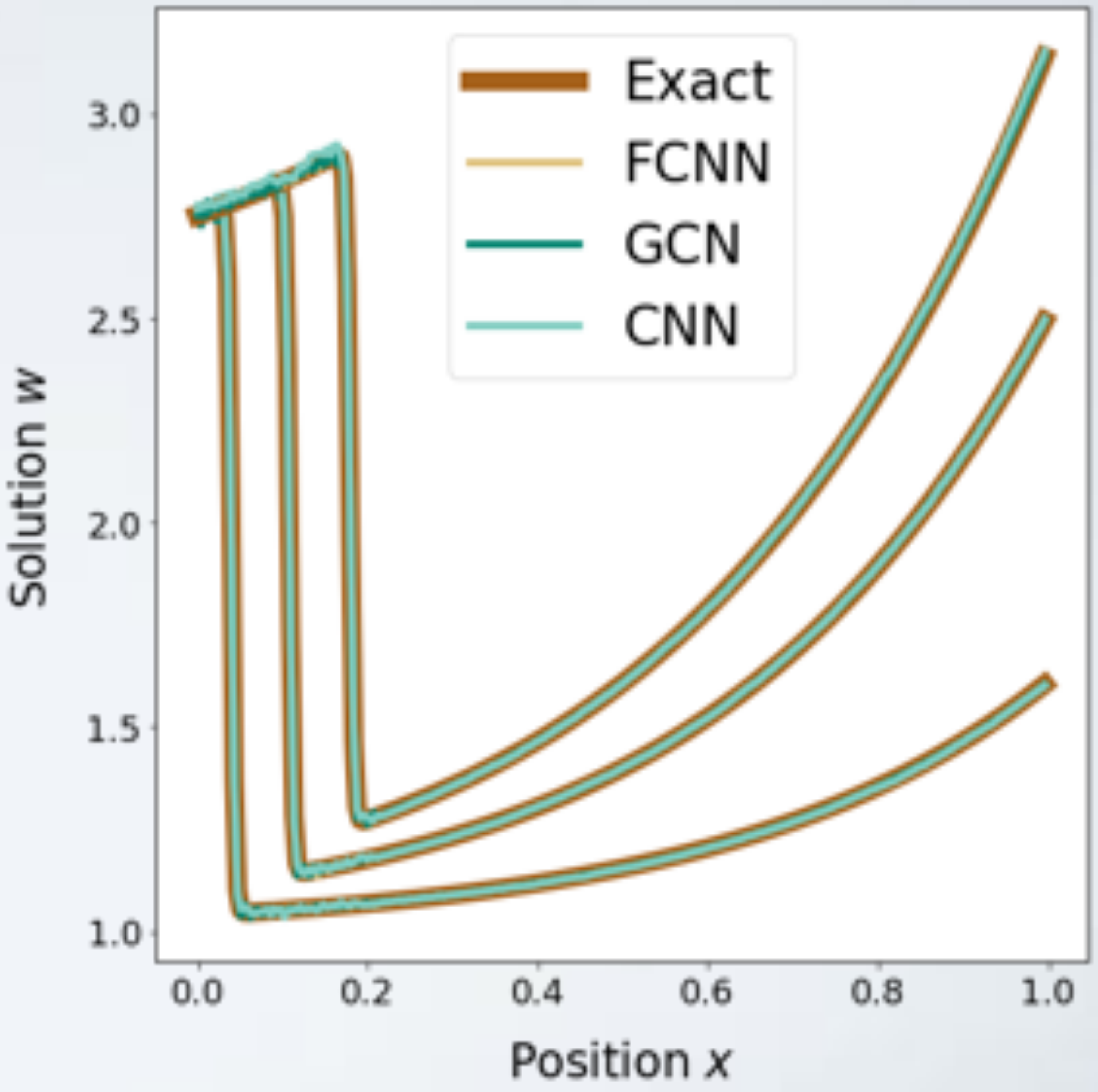
1-D Inviscid Burger's: ROM

- ROM problem is very regular: not difficult for network methods.
- Even $n = 3$ (pictured) is sufficient for <1% error with CNN.
- Conversely, GCNN and FCNN struggle when the latent space is small.

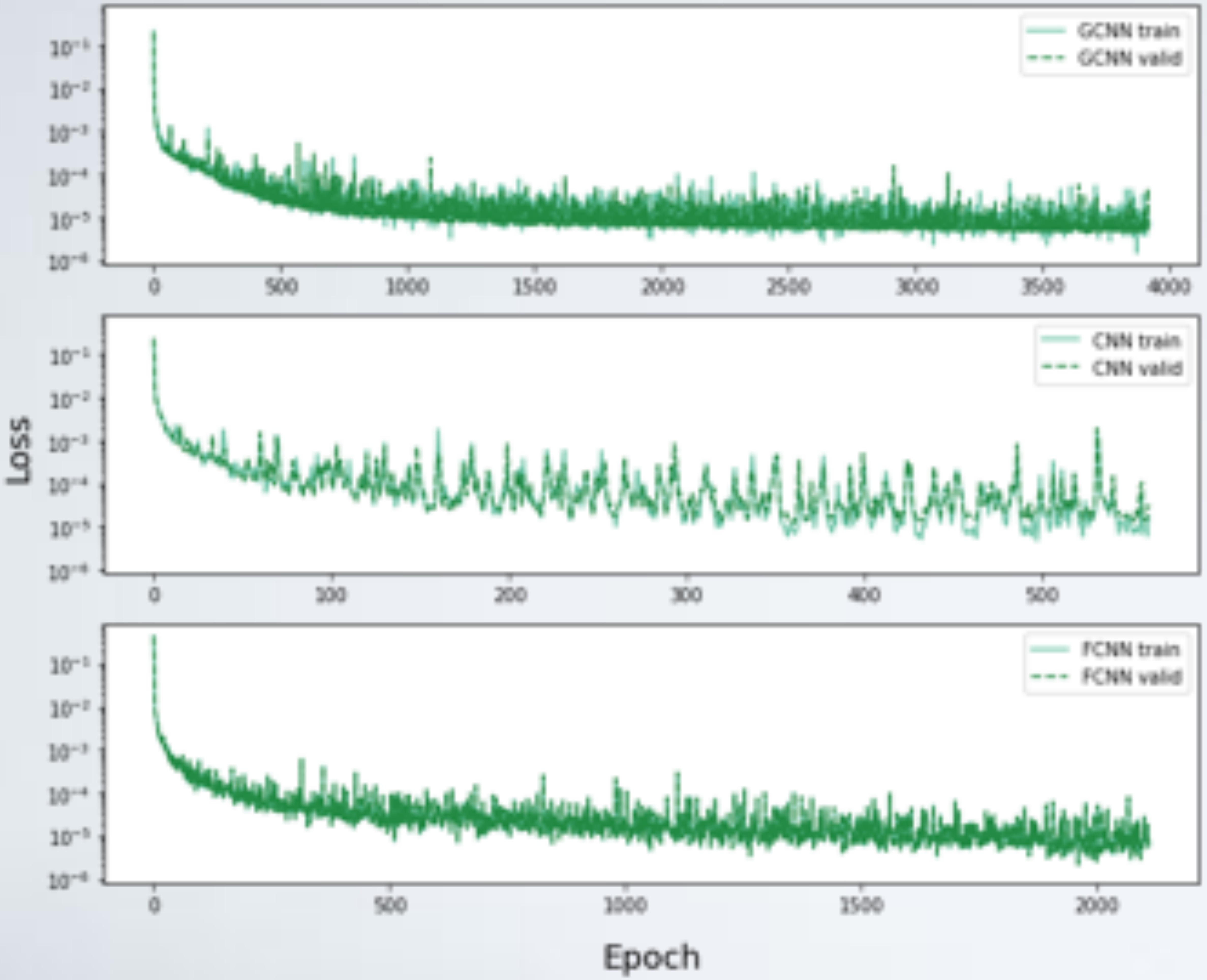


1-D Inviscid Burger's: Compression

- CNN still best for compression until latent dim 32 (pictured)
- GCNN almost matches performance of 2-layer FCNN (best) with half the memory
- Note that the CNN used requires more than 6x the memory of the FCNN.



1-D Inviscid Burger's: Results



	Encoder/Decoder + Prediction				Encoder/Decoder only			
Network	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)
GCN	3	4.41	8.49	0.164	3	2.54	5.31	0.033
CNN		0.304	0.605	1.93		0.290	0.563	1.91
FCNN		1.62	3.29	0.336		0.658	1.66	0.295
GCN	10	2.08	3.73	0.197	10	0.706	1.99	0.057
CNN		0.301	0.630	1.98		0.215	0.409	1.93
FCNN		0.449	1.15	0.343		0.171	0.361	0.303
GCN	32	2.59	4.17	0.295	32	0.087	0.278	0.147
CNN		0.350	0.675	2.08		0.216	0.384	2.03
FCNN		0.530	1.303	0.377		0.098	0.216	0.319

- Loss pictured for $n = 32$.
- ROM Errors fluctuate with n — prediction network has issues.

2-D Parameterized Heat

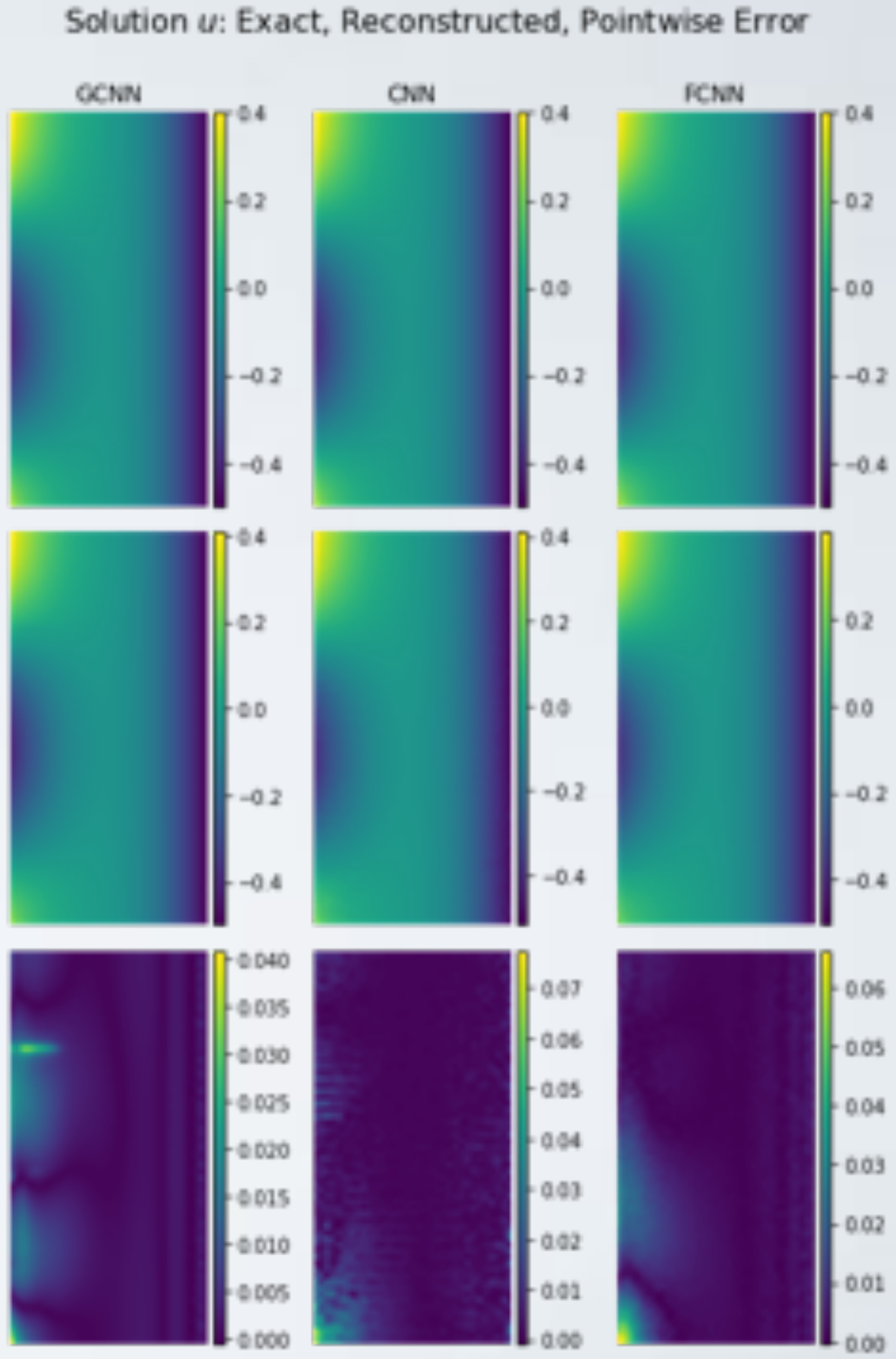
- Consider

  $u = u(x, y, t, \boldsymbol{\mu}),\ U = [0,1] \times [0,2],\ \boldsymbol{\mu} \in [0.05,0.5] \times [\pi/2, \pi]$

  and solve

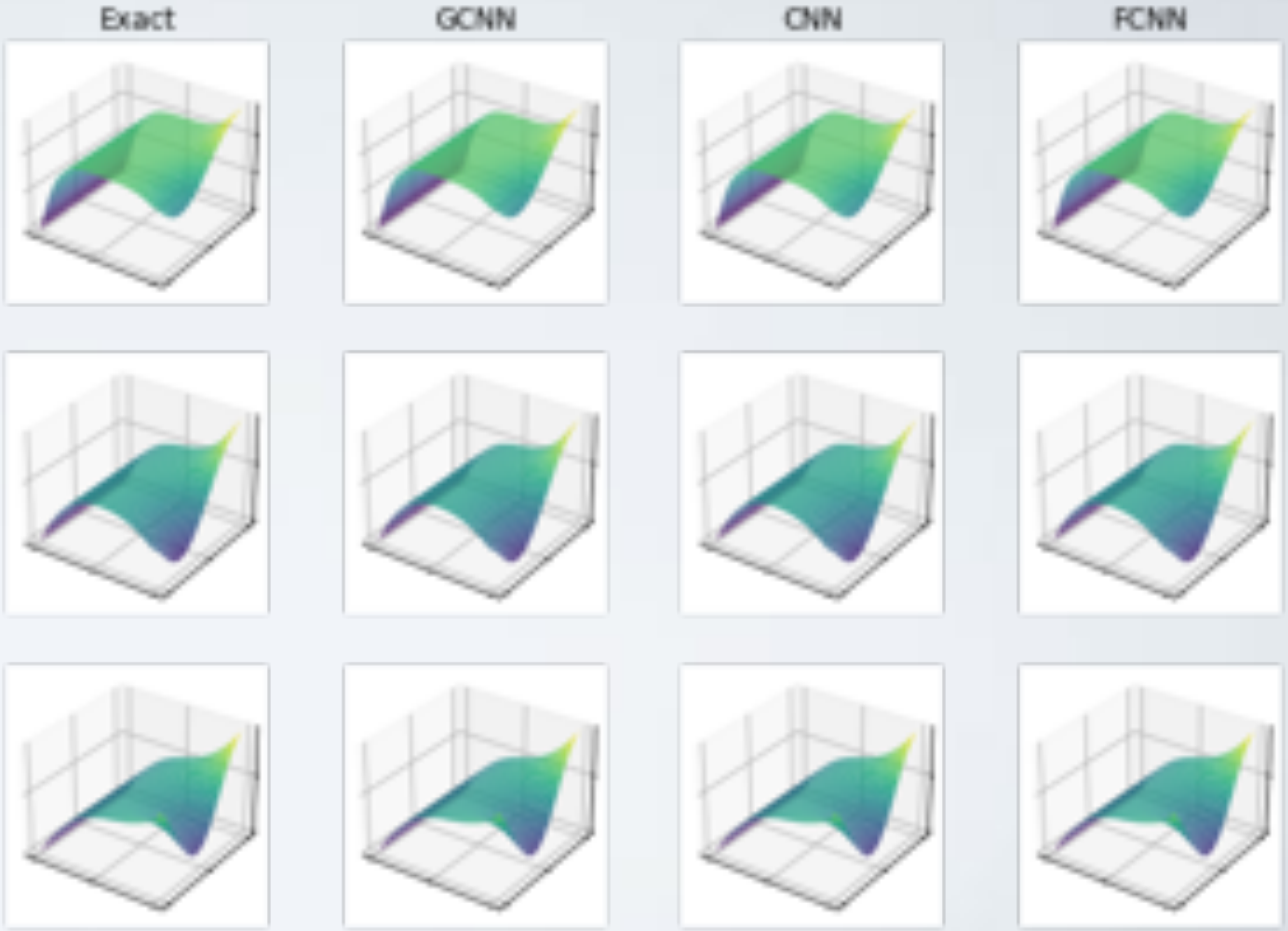
$$\begin{cases} u_t - \Delta u = 0 \quad \text{on } U \\ u(0, y, t) = -0.5 \\ u(1, y, t) = \mu_1 \cos(\mu_2 y) \\ u(x, y, 0) = 0 \end{cases}$$

- Discretizing over grid gives $u = \mathbf{u}(t, \boldsymbol{\mu})$



2-D Parameterized Heat: Results

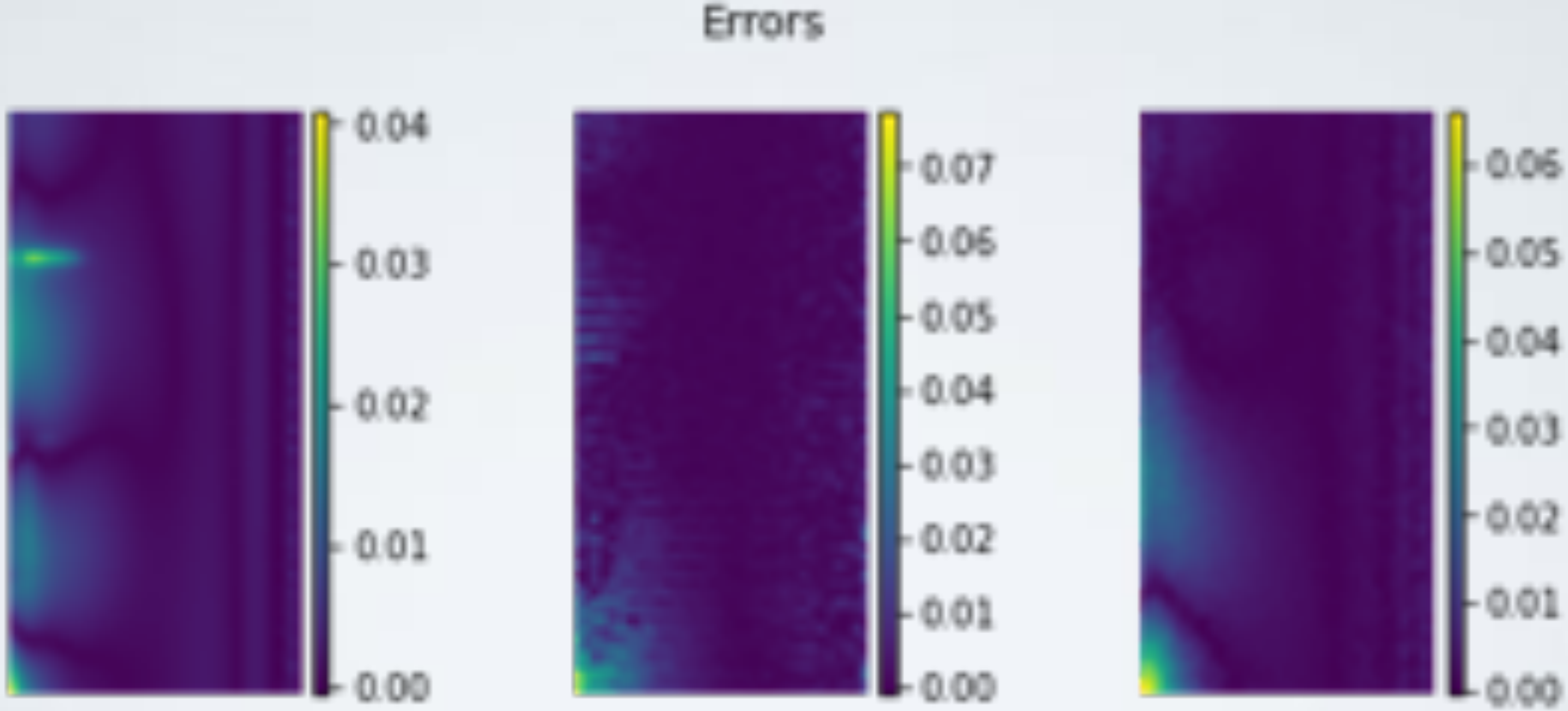
- Results shown for $n = 10$.
- GCNN has lowest error and least memory requirement (by >10x!)
- CNN is worst — cheap hacks have a cost.



2-D Parameterized Heat: Results

	Encoder/Decoder + Prediction					Encoder/Decoder only				
Network	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)
GCN		7.19	9.21	0.132	9.5		6.96	9.21	0.0659	9.2
CNN	3	3.26	4.58	3.64	18	3	3.36	3.81	3.62	18
FCNN		4.75	6.19	3.74	3.3		4.22	5.69	3.72	3.1
GCN		2.87	3.82	0.253	9.6		2.06	2.85	0.186	9.4
CNN	10	3.07	4.38	3.87	18	10	2.45	2.90	3.85	18
FCNN		2.96	3.97	3.76	3.3		2.32	2.92	3.73	3.1
GCN		2.55	3.48	0.636	9.6		1.05	1.91	0.564	9.2
CNN	32	2.30	3.73	4.60	19	32	2.34	2.91	4.57	18
FCNN		2.65	4.25	3.80	3.2		1.61	2.31	3.77	3.2

2-D Parameterized Heat: Results

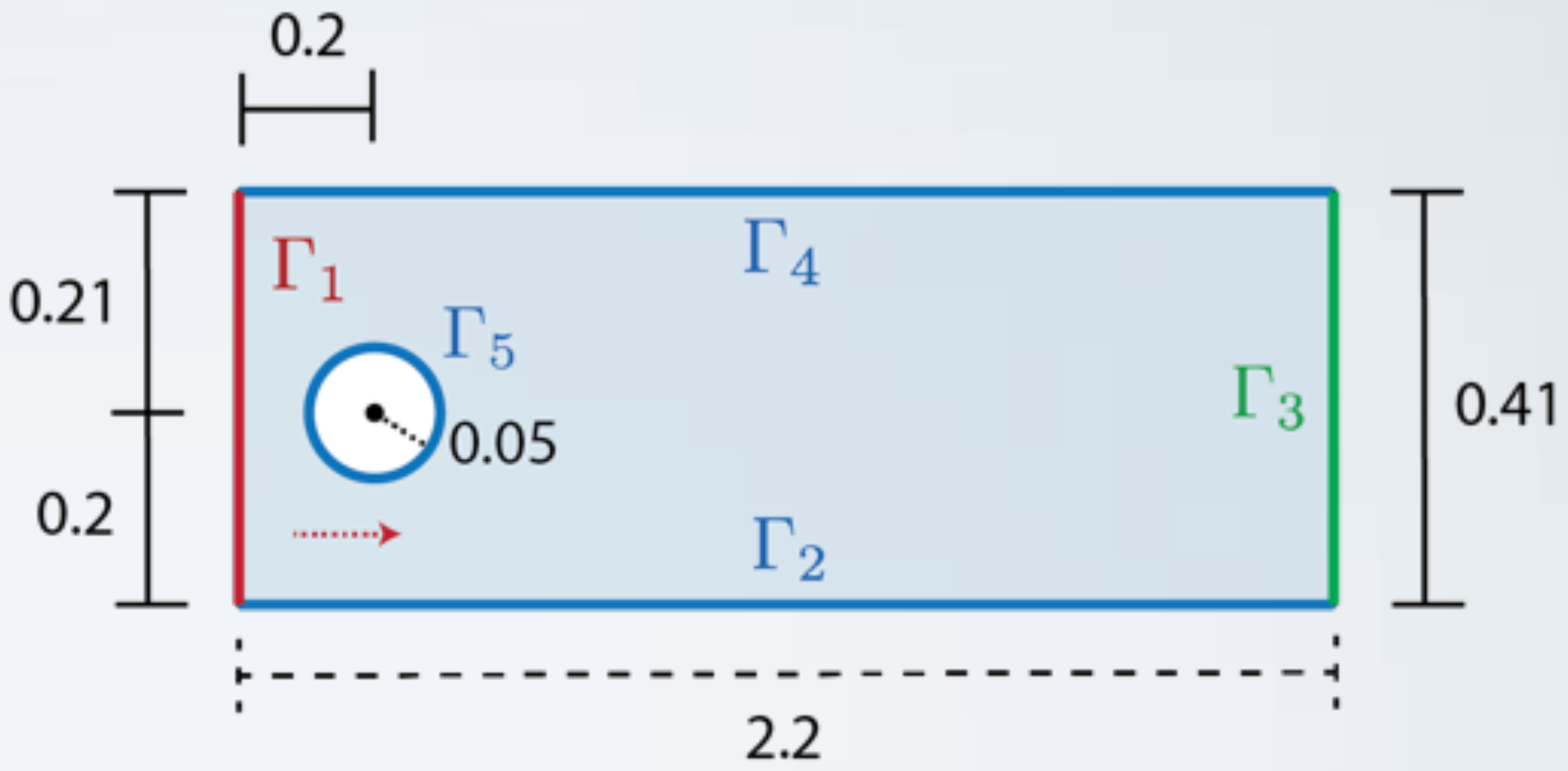


Unsteady Navier-Stokes Equations

- Consider the Schafer-Turek benchmark problem:

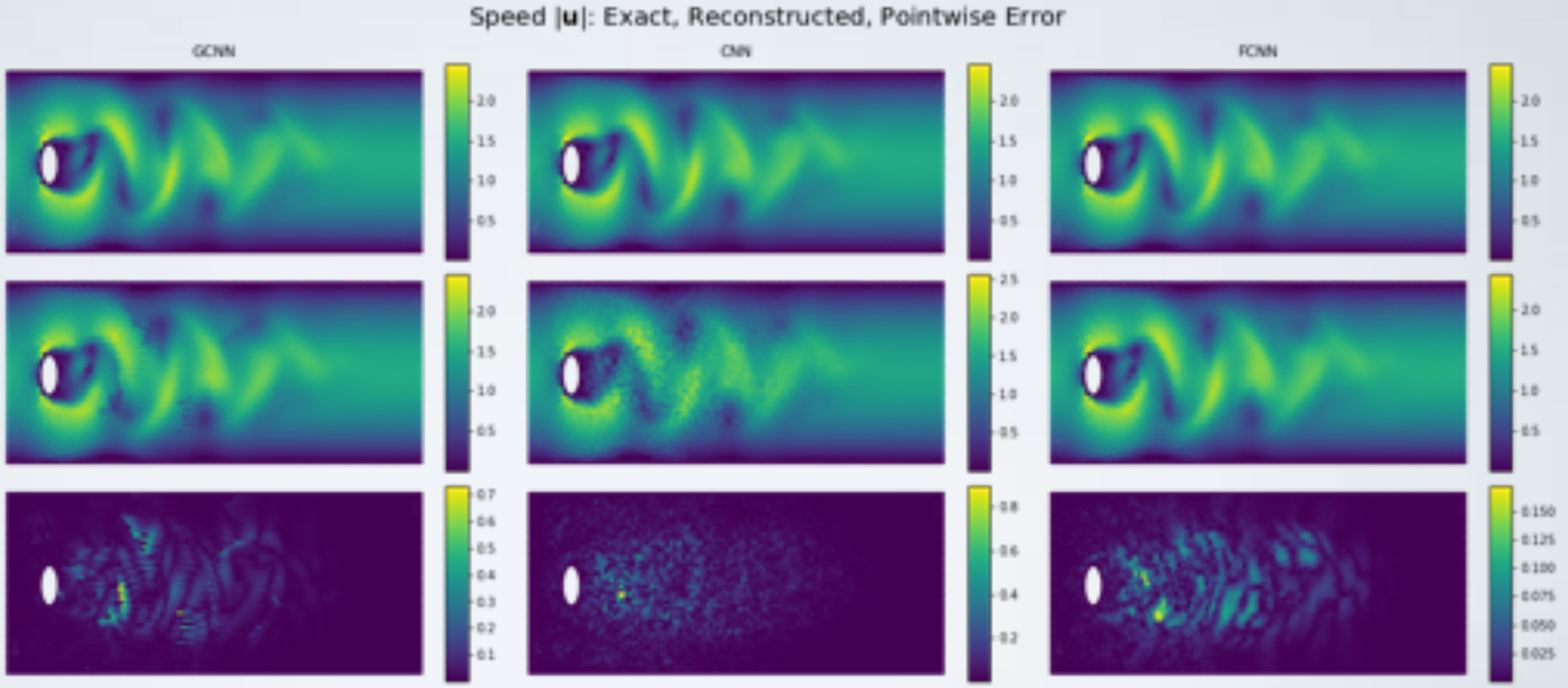
$$\dot{\mathbf{u}} - \nu\Delta\mathbf{u} + \nabla_{\mathbf{u}}\mathbf{u} + \nabla p = \mathbf{f},$$
$$\nabla\cdot\mathbf{u} = 0,$$
$$\mathbf{u}\big|_{t=0} = \mathbf{u}_0\,.$$

- Impose 0 boundary conditions on $\Gamma_2, \Gamma_4, \Gamma_5$. Do nothing on Γ_3. Parabolic inflow on Γ_1.



Navier-Stokes Equations: Results

- $N = 10104$
- $n = 32$
- Reynolds number 185.
- FCNN best on prediction problem.

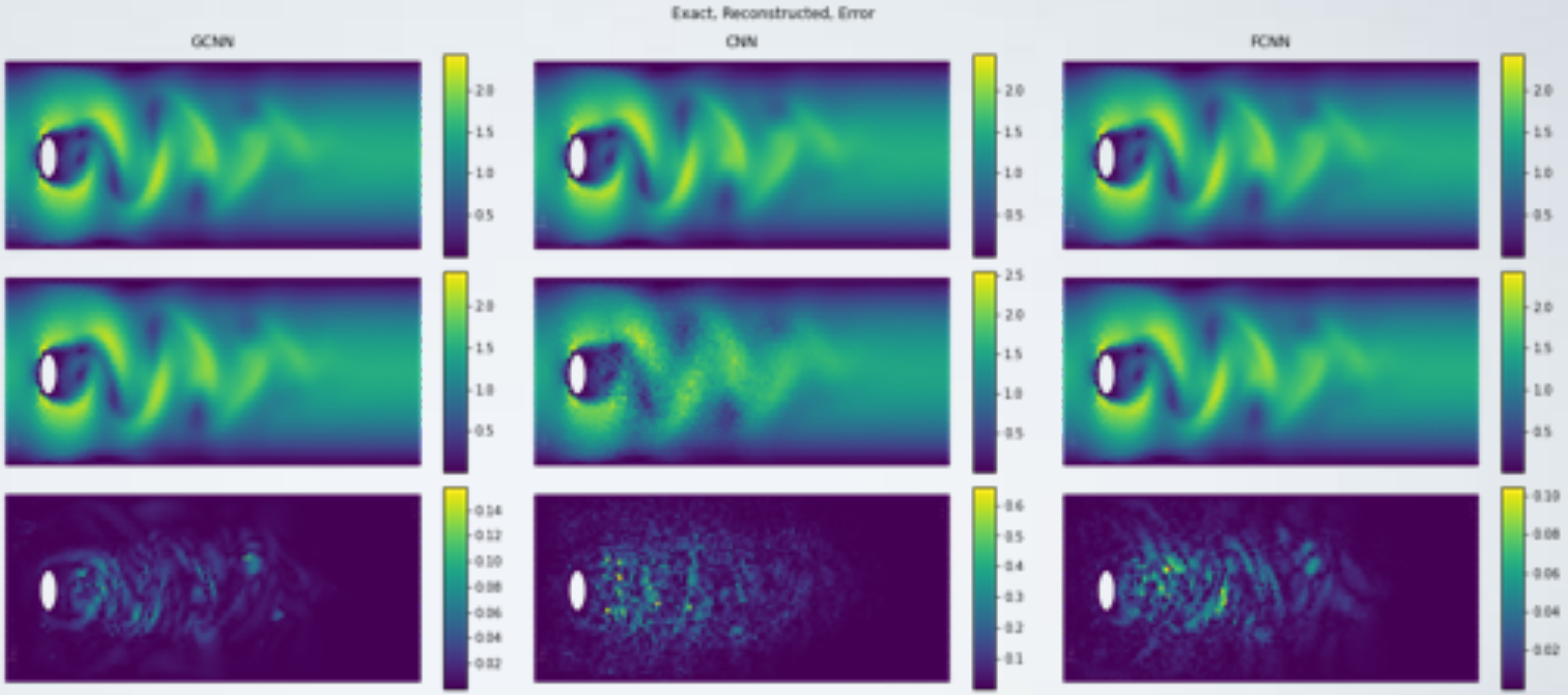


Navier-Stokes Equations: Results

	Encoder/Decoder + Prediction					Encoder/Decoder only				
Network	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)	n	$R\ell_1\%$	$R\ell_2\%$	Size (MB)	Time per Epoch (s)
GCN	2	9.07	14.6	0.476	33	2	7.67	12.2	0.410	32
CNN		7.12	11.1	224	210		11.2	17.6	224	190
FCNN		1.62	2.87	330	38		1.62	2.70	330	38
GCN	32	2.97	5.14	5.33	32	32	0.825	1.49	5.26	32
CNN		4.57	7.09	232	230		4.61	7.24	232	220
FCNN		1.39	2.64	330	38		0.680	1.12	330	38
GCN	64	2.88	4.96	10.5	33	64	0.450	0.791	10.4	33
CNN		3.42	5.33	241	270		2.42	3.57	241	260
FCNN		1.45	2.64	330	38		0.704	1.19	330	37

Navier-Stokes Equations: Results

- Compression
- GCNN matches FCNN in accuracy
- GCNN memory cost is >50x less than FCNN



Conclusion

- Neural networks are exciting new technology which brings together experts from many domains.
- There is plenty of math to be done for ML, and many scientific applications are enhanced by the use of NNs.
- You can contribute to this area!

Thank You!