Anthony Gruber

Joint work with: M. Gunzburger, L. Ju, Z. Wang (FSU & UofSC)

Neural Network Architectures for Data Compression and Reduced-Order Modeling

Full-order Model

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

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.

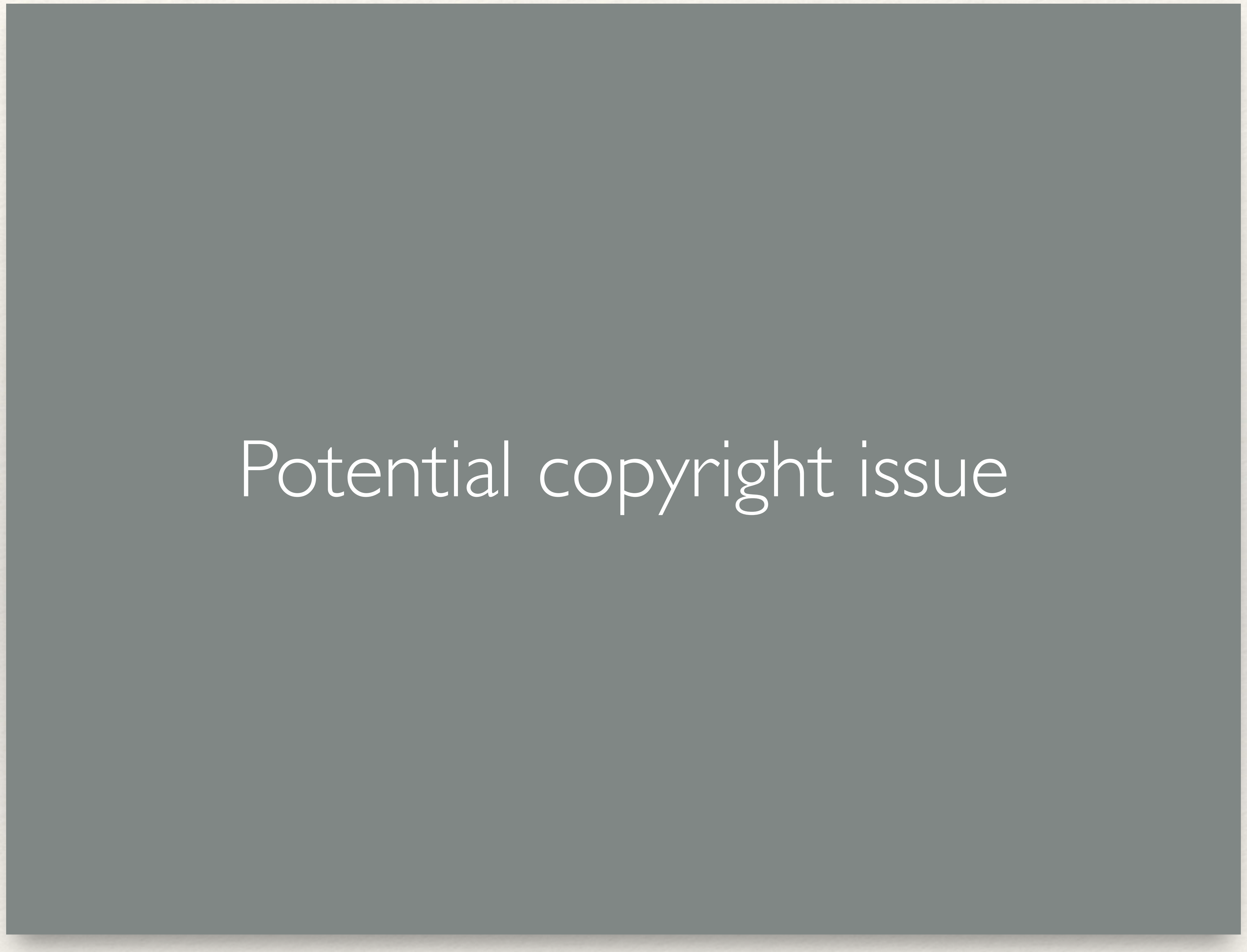
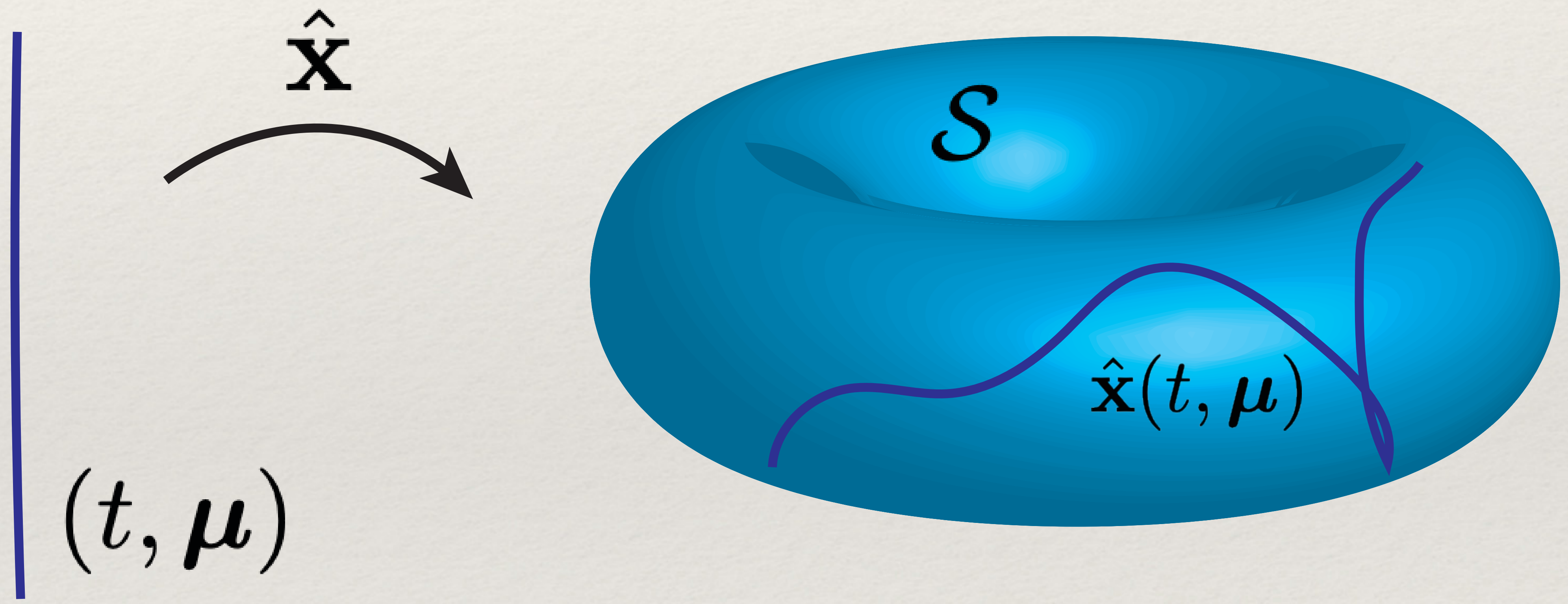


Image: https://mpas-dev.github.io/ocean/ocean.html

Idea behind ROM

- 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 Decomposition (POD)

- Most popular ROM (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: $\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}$.
  - First n cols of $\mathbf{U}$ (say $\mathbf{A}$) — reduced basis of POD modes.
  - Instead of $\dot{\mathbf{x}} = f(\mathbf{x})$, can then solve $\dot{\hat{\mathbf{x}}} = \mathbf{A}^{+}\mathbf{f}(\mathbf{A}\hat{\mathbf{x}})$.
- Totally linear procedure — good and bad.

POD vs Neural Network

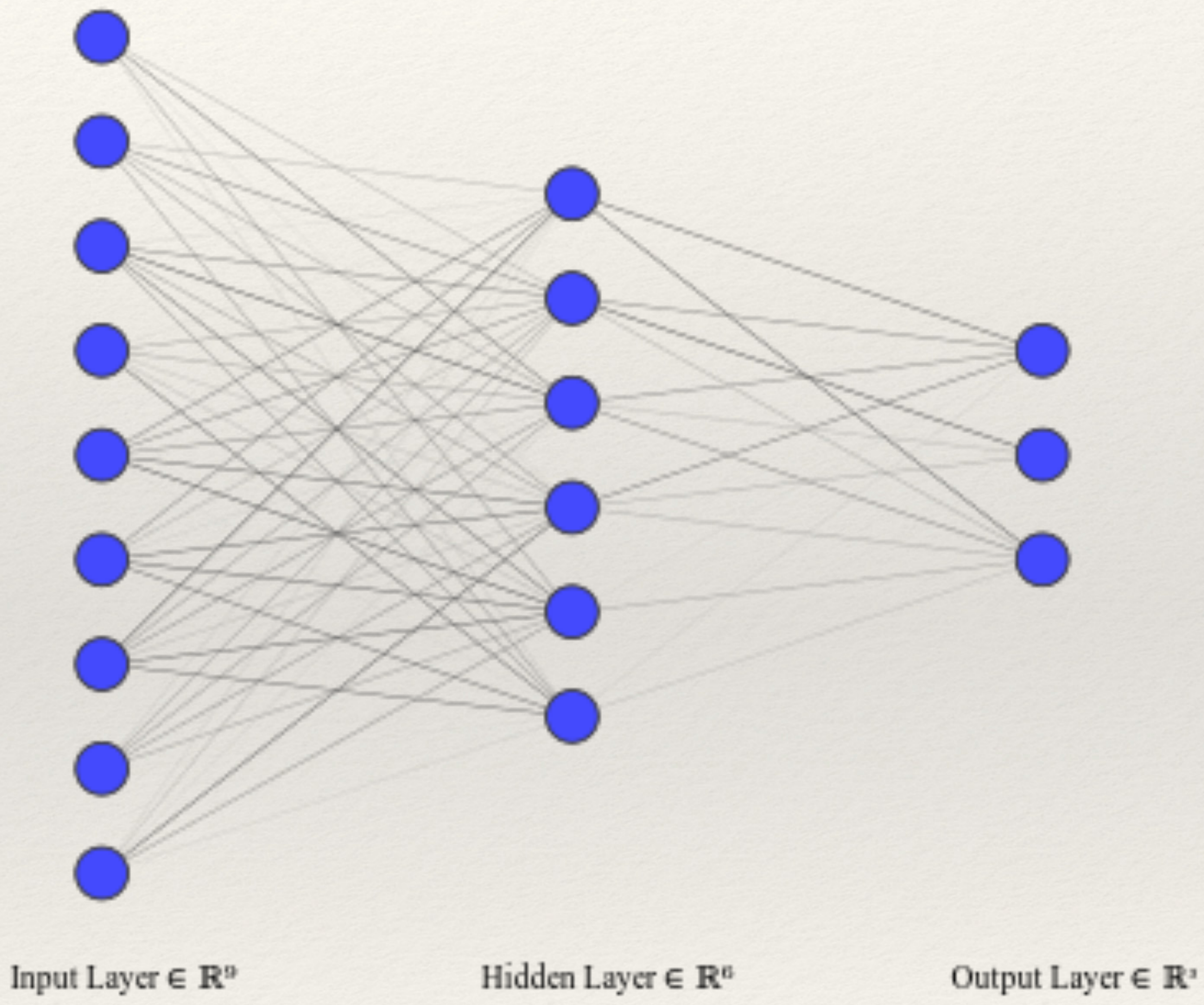
- POD works well until EWs of $\boldsymbol{\Sigma}$ decay slowly.
  - Even many modes cannot reliably capture behavior.
- Conversely, FCNN / CNN captures patterns quite well.
- *Are all NNs equal for this purpose?*

Potential copyright issue

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

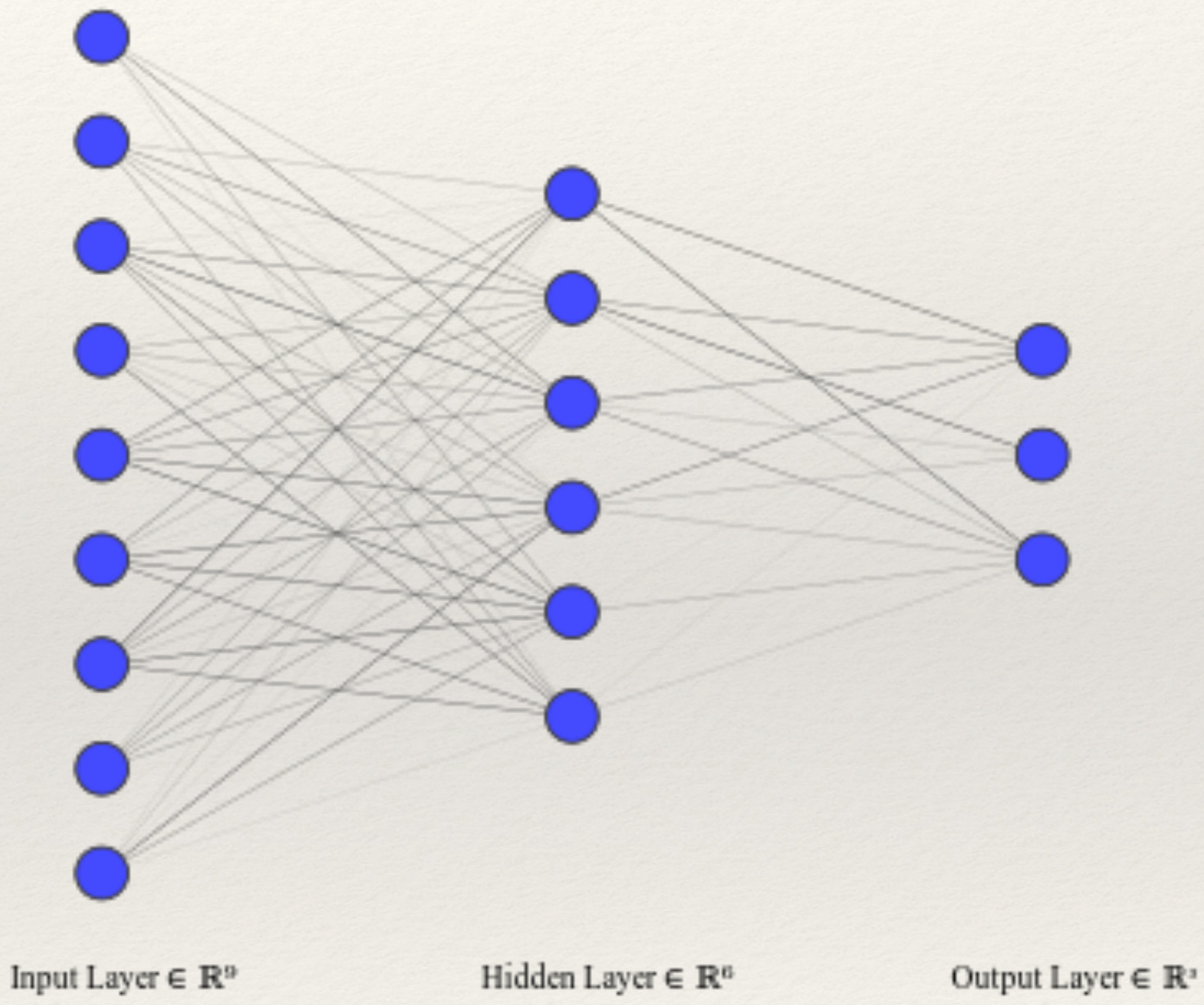
Review: Fully Connected Network

- Most obvious choice is FCNN.
- Given $\mathbf{y} = \mathbf{y}_0 \in \mathbb{R}^{N_0}$ and $1 \leq \ell \leq L$,
  - $\mathbf{y}_L = \mathbf{T}_L \circ \mathbf{T}_{L-1} \circ \ldots \circ \mathbf{T}_1(\mathbf{y}_0)$
  - $\mathbf{y}_\ell = \mathbf{T}_\ell(\mathbf{y}_{\ell-1}) = \boldsymbol{\sigma}_\ell \left(\mathbf{W}_\ell \mathbf{y}_{\ell-1} + \mathbf{b}_\ell\right),$
- $(\mathbf{W}_\ell, \mathbf{b}_\ell)$ are learnable parameters.
- $\boldsymbol{\sigma}$ is element-wise activation function.



Review: Fully Connected Network

- FCNN is most expressive, but prone to *overfitting* and *difficult to train*.
- Also requires a large amount of memory due to overparametrization.
- First used for ROM by (Milano and Koumoutsakos 2002).
  - Linear version equivalent to POD.



CNN Model Order Reduction

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

Potential copyright issue

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

Disadvantage of CNN

- Recall:

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

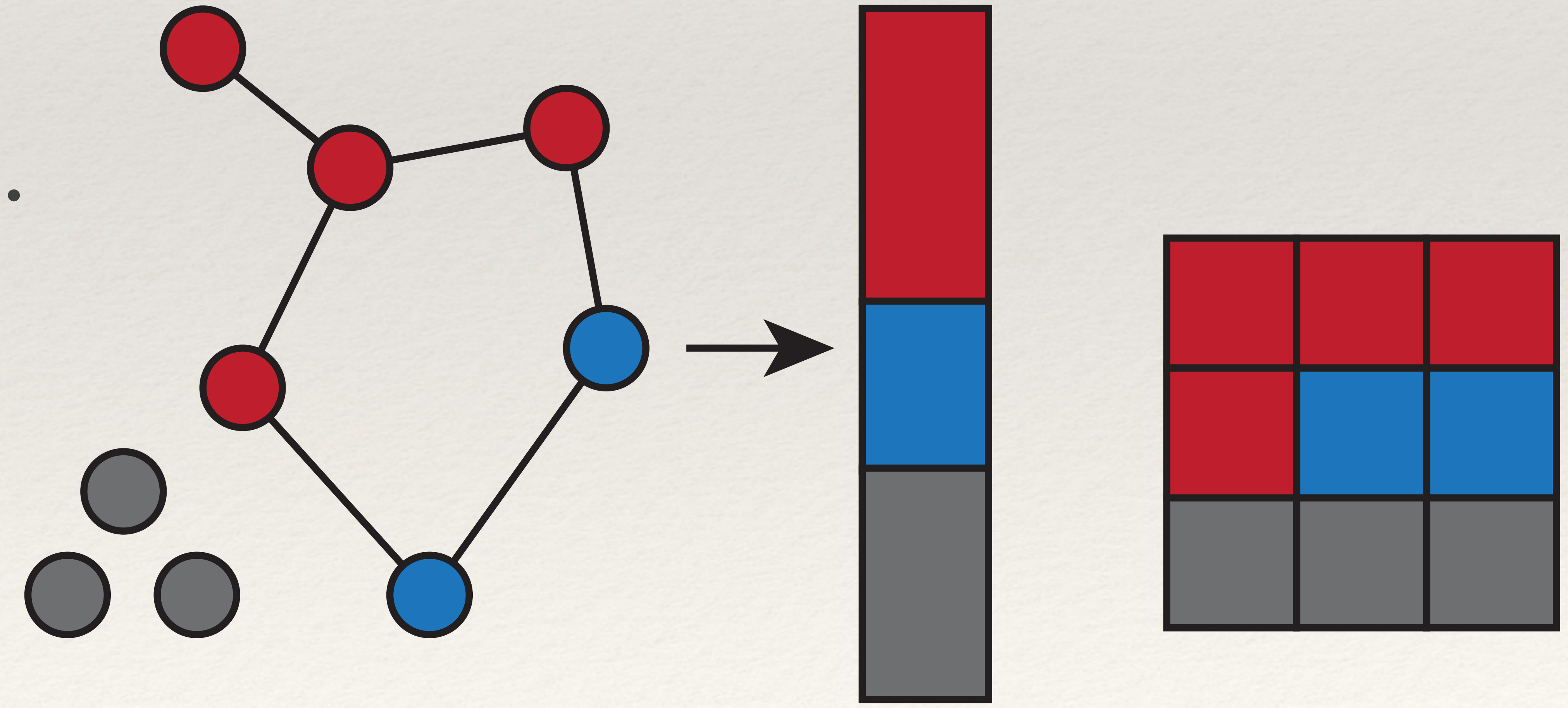
- Convolution $\star$ in 2-D:

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

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

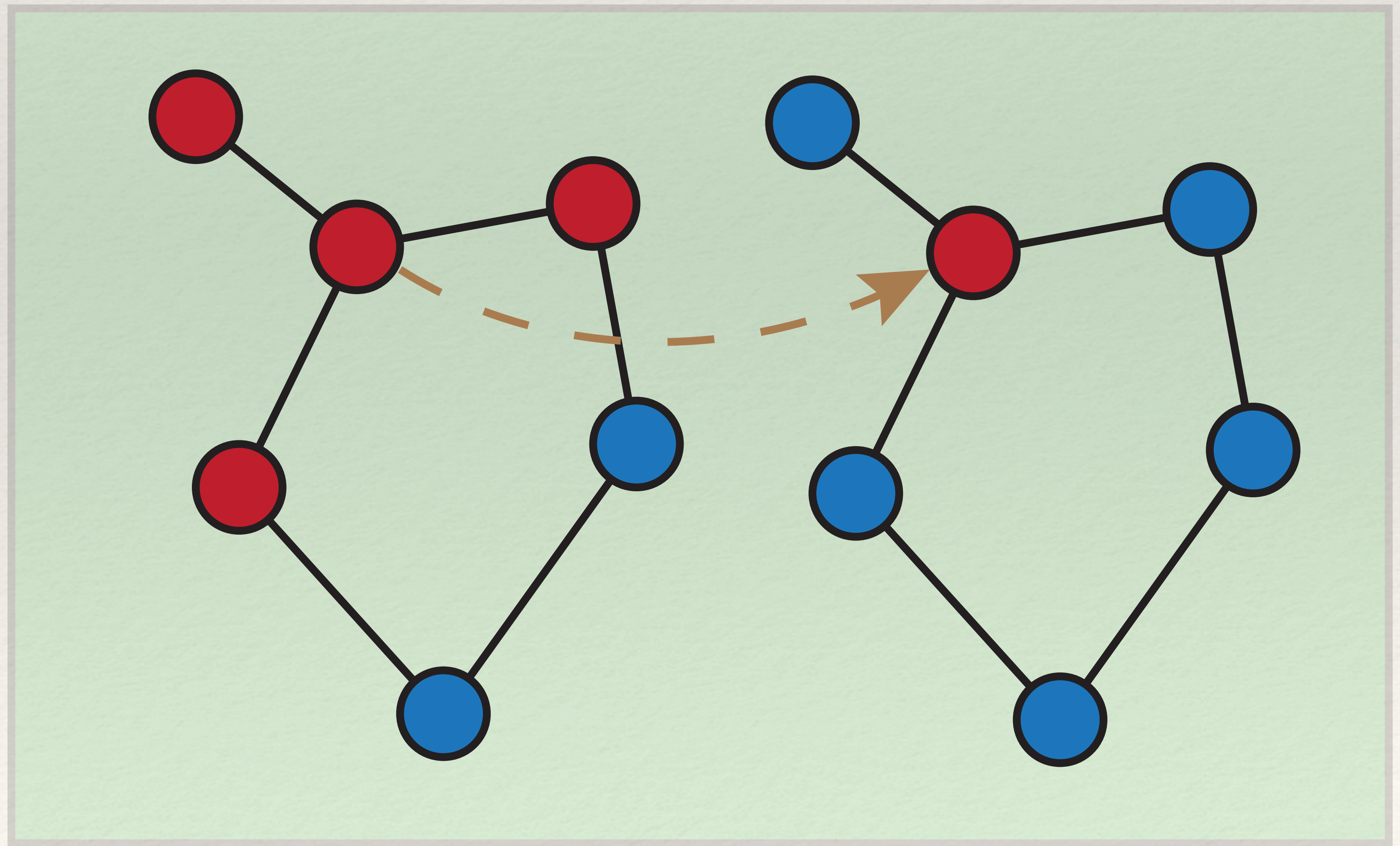
Disadvantage of CNN

- 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.
- $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ undirected graph with adj. matrix $\mathbf{A} \in \mathbb{R}^{|\mathscr{V}| \times |\mathscr{V}|}$.
- $\mathbf{D}$ the degree matrix $d_{ii} = \sum_j a_{ij}$.
- The Laplacian of $\mathscr{G}$: $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$.
  - Columns of $\mathbf{U}$ are Fourier modes of $\mathscr{G}$.
  - Discrete FT / IFT: simply multiply by $\mathbf{U}^\top / \mathbf{U}$.



Graph Convolutional Networks

- $\mathbf{y}_i : \mathbb{R}^{|\mathscr{V}|} \to \mathbb{R}$ signals 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

- Multiplication with Fourier basis is too expensive.
- (Defferard et al. 2016) Use Chebyshev polynomial filters.
- Leads to the propagation rule:

$$\mathbf{x}_{\ell,i} = \boldsymbol{\sigma}_{\ell}\left(\sum_{j=1}^{C_{in}} \mathbf{W}^{j}_{\ell,i}\mathbf{x}_{(\ell-1),j} + \mathbf{b}_{\ell,i}\right),$$

- $\mathbf{W}^{j}_{\ell,i} = \left(g_{\theta}(\mathbf{L})\right)^{j}_{\ell,i} = \sum_{k} \theta^{j}_{\ell,ki} T_k(\tilde{\mathbf{L}})$ where $\tilde{\mathbf{L}}$ is rescaled Laplacian.
    - $1 \leq k \leq K$ is a user-defined choice — leads to K-hop aggregation.

Graph Convolutional Networks

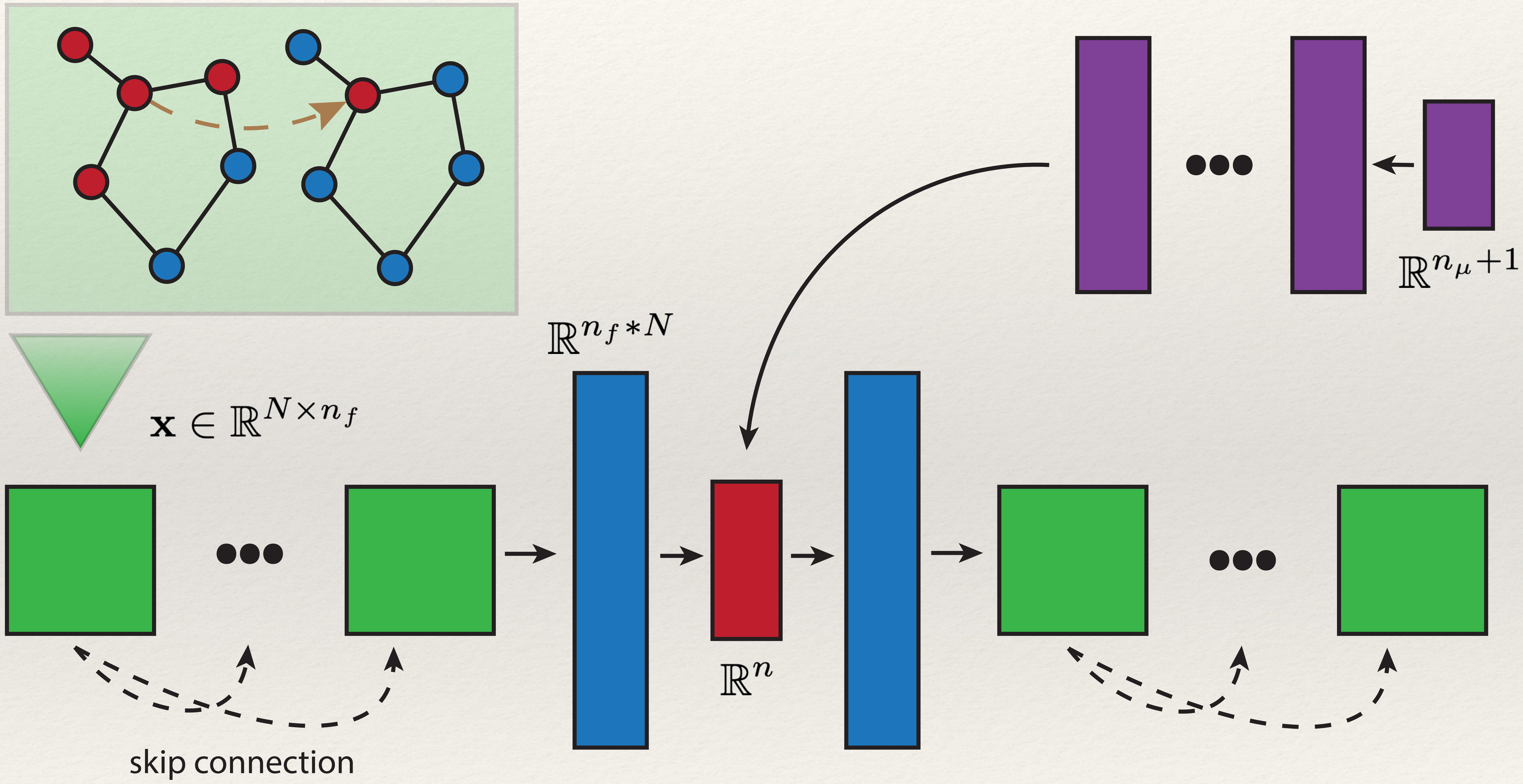
- Let $\tilde{\mathbf{P}} = (\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}$ (added self-loops).
- Simplified 1-localized GCN (Kipf and Welling 2016): $\mathbf{x}_{\ell+1} = \boldsymbol{\sigma}\left(\tilde{\mathbf{P}}\mathbf{x}_\ell \mathbf{W}_\ell\right)$.
- Good 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).



Experimental Details

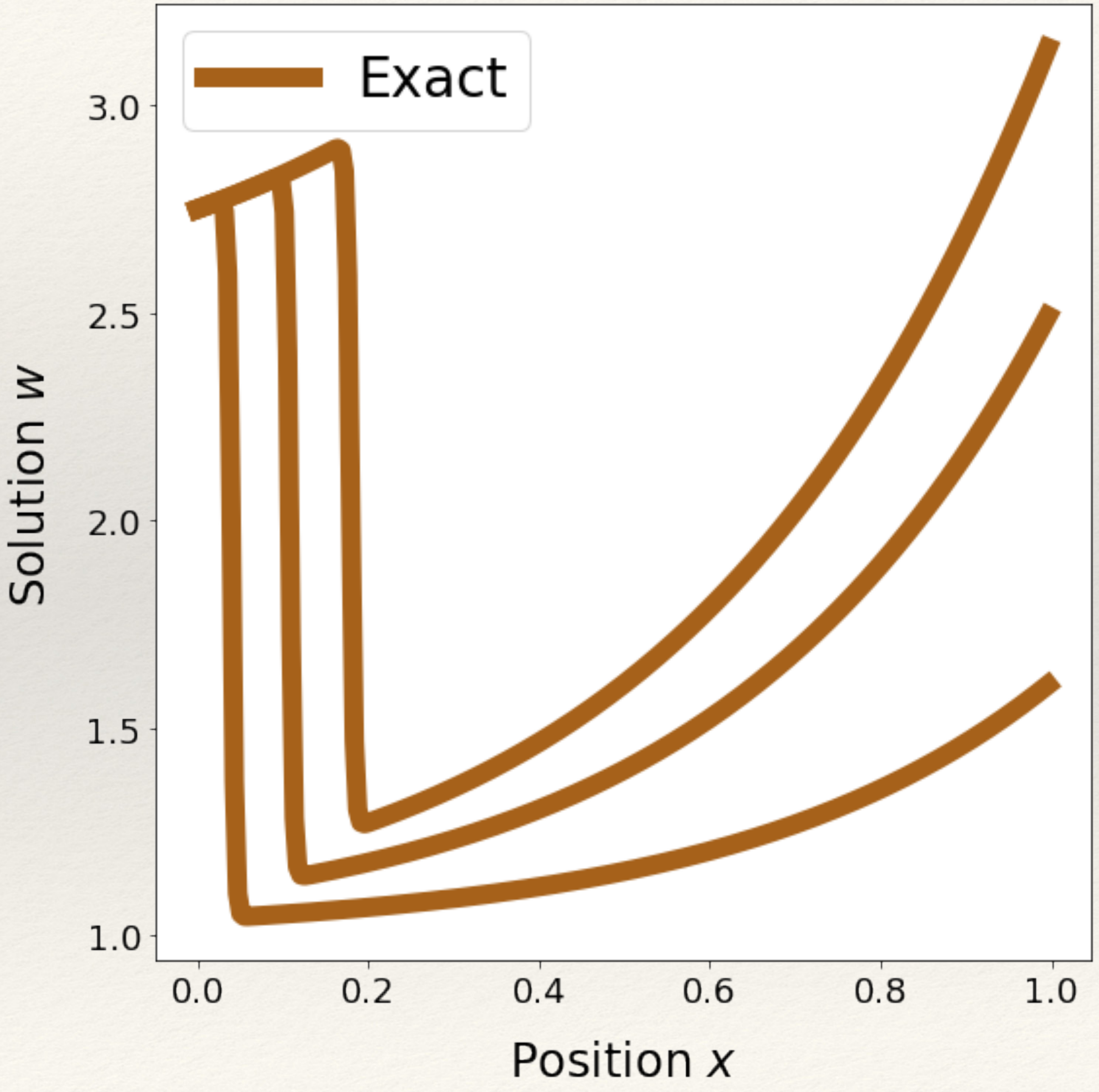
- Want to compare performance of GCAE, CAE, FCAE for ROM applications.
- Evaluation based on two criteria:
  - Pure reconstruction ability (compression problem).
  - Ability to predict new solutions given parameters (prediction problem).
- Prediction loss used: $L(\mathbf{x}, t, \boldsymbol{\mu}) = |\mathbf{x} - \mathbf{g} \circ \hat{\mathbf{x}}|^2 + |\mathbf{h} - \hat{\mathbf{x}}|^2.$
- Compression loss: $L(\mathbf{x}, t, \boldsymbol{\mu}) = |\mathbf{x} - \mathbf{g} \circ \mathbf{h}|^2.$
- Network trained using mini-batch descent with ADAM optimizer.

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 $\boldsymbol{\mu} \in [2,3] \times [0.015, 0.030]$.



Architecture Comparison

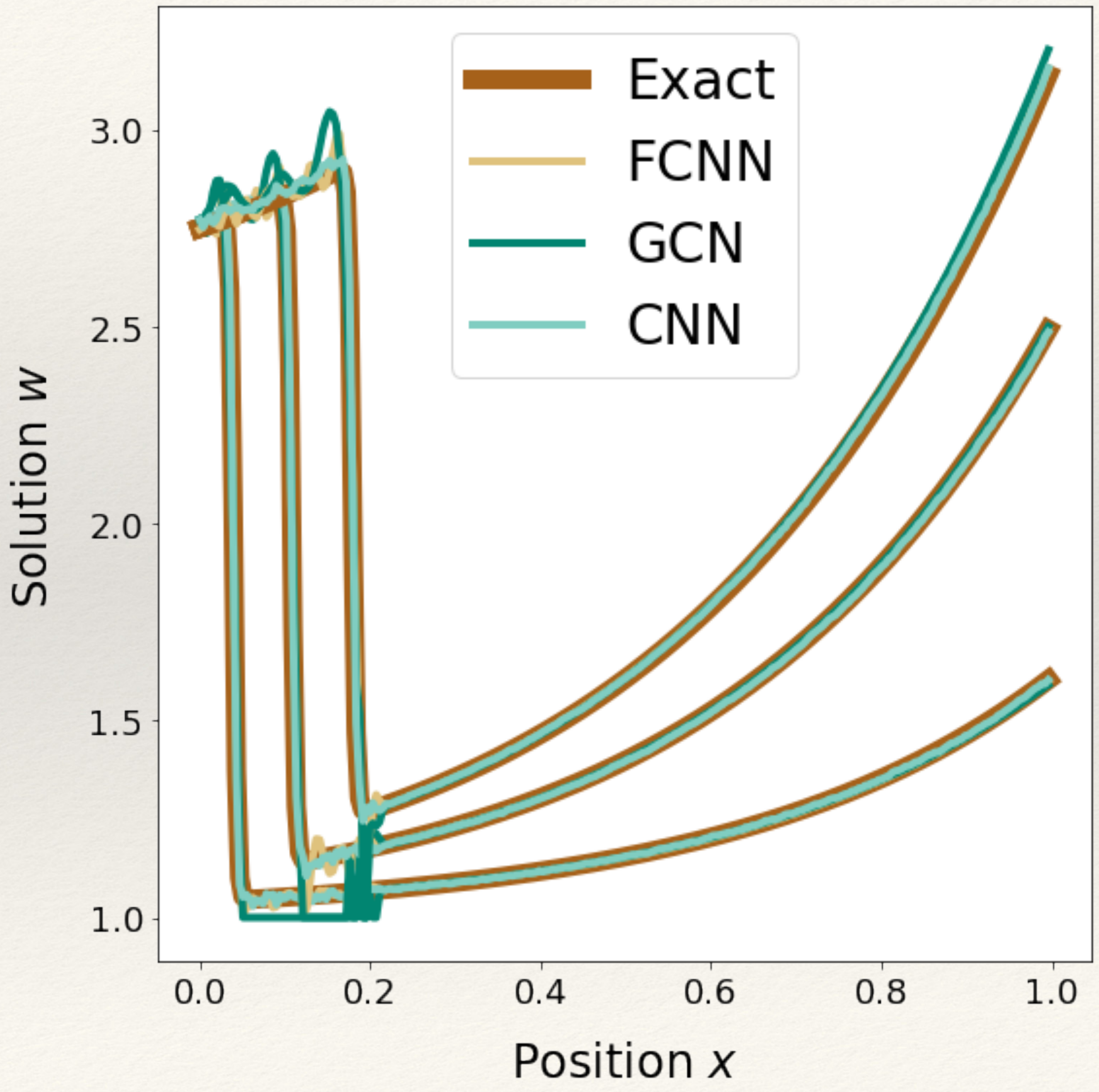
layer	input size	kernel size	stride	padding	output size	activation
Samples of size (256,1) are reshaped to size (1, 16, 16).						
1-C	(1, 16, 16)	5x5	1	SAME	(8, 16, 16)	ELU
2-C	(8, 16, 16)	5x5	2	SAME	(16, 8, 8)	ELU
3-C	(16, 8, 8)	5x5	2	SAME	(32, 4, 4)	ELU
4-C	(32, 4, 4)	5x5	2	SAME	(64, 2, 2)	ELU
Samples of size (64, 2, 2) are flattened to size (256).						
1-FC	256				reduced dim	ELU
End of encoding layers. Beginning of decoding layers.						
2-FC	reduced dim				256	ELU
Samples of size (256) are reshaped to size (64, 2, 2).						
1-TC	(64, 2, 2)	5x5	2	SAME	(64, 4, 4)	ELU
2-TC	(64, 4, 4)	5x5	2	SAME	(32, 8, 8)	ELU
3-TC	(32, 8, 8)	5x5	2	SAME	(16, 16, 16)	ELU
4-TC	(16, 16, 16)	5x5	1	SAME	(1, 16, 16)	ELU
Samples of size (1, 16, 16) are reshaped to size (256, 1).						

layer	input size	kernel size	α	θ	output size	activation
1-C	(N, n_f)	$n_f \times n_f$	0.2	1.5	(N, n_f)	ReLU
......						
N_l-C	N, n_f	$n_f \times n_f$	0.2	1.5	(N, n_f)	ReLU
Samples of size (N, n_f) are flattened to size $(n_f * N)$.						
1-FC	$n_f * N$				n	Identity
End of encoding layers. Beginning of decoding layers.						
2-FC	n				$n_f * N$	Identity
Samples of size $(n_f * N)$ are reshaped to size (N, n_f).						
1-TC	(N, n_f)	$n_f \times n_f$	0.2	1.5	(N, n_f)	ReLU
......						
N_l-TC	(N, n_f)	$n_f \times n_f$	0.2	1.5	(N, n_f)	ReLU
End of decoding layers. Prediction layers listed below.						
3-FC	$n_\mu + 1$				50	ELU
4-FC	50				50	ELU
......						
11-FC	50				n	Identity

- CNN (left), GCNN (right)
- FCNN is 2+2 layers, neurons 256, 64, *n*.

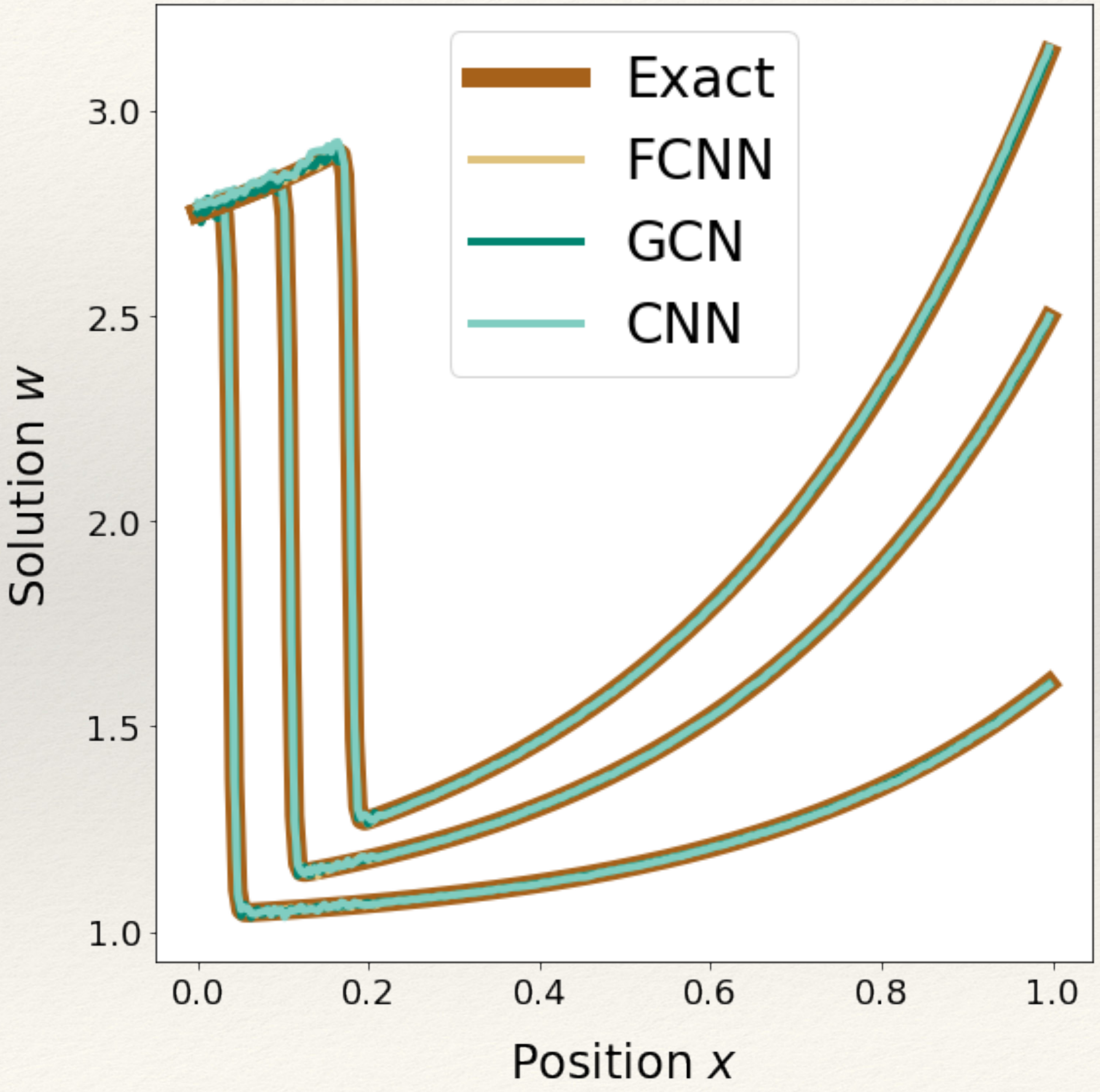
1-D Inviscid Burger's Equation

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

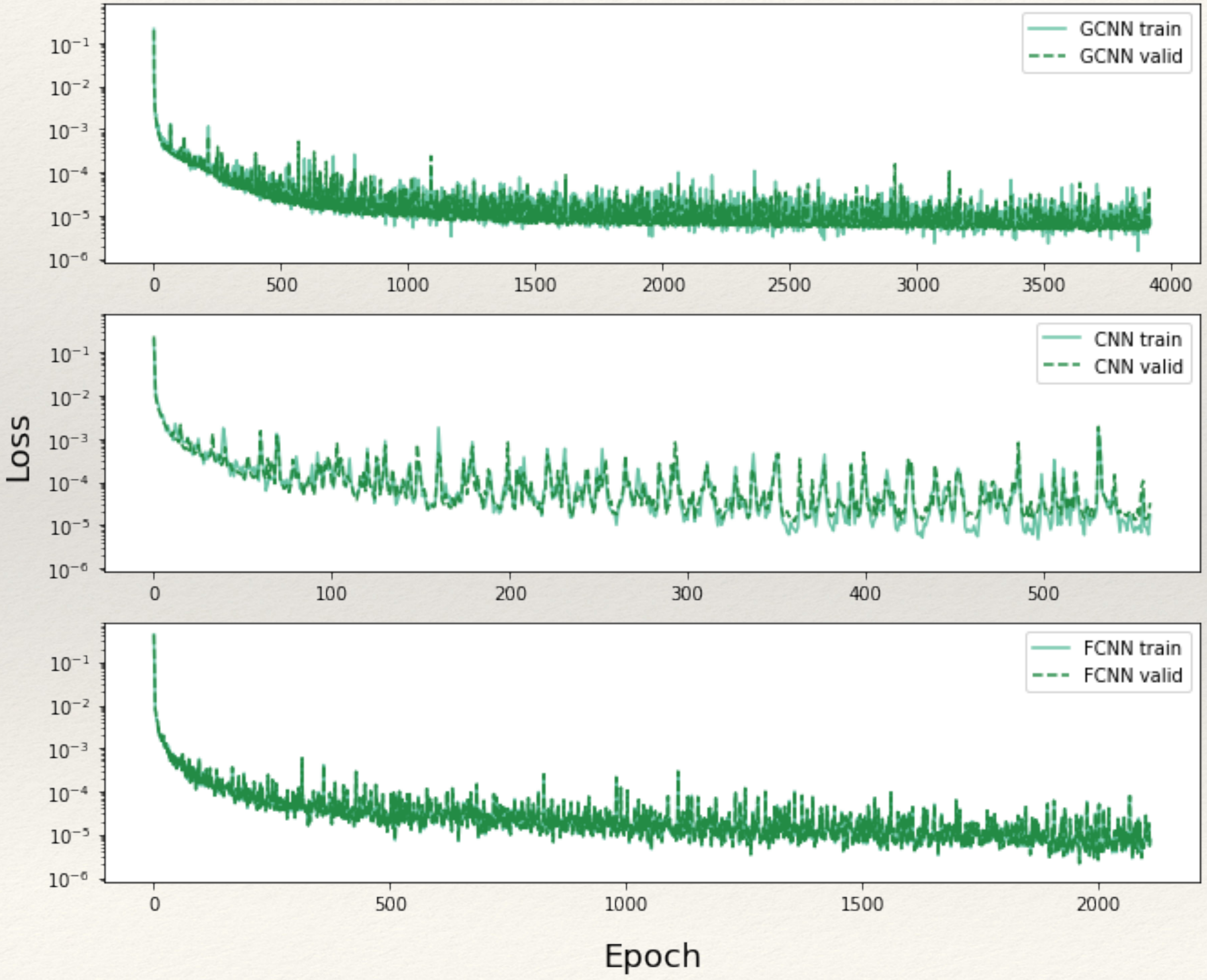


1-D Inviscid Burger's Equation: Compression

- 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 Equation: 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 Equation

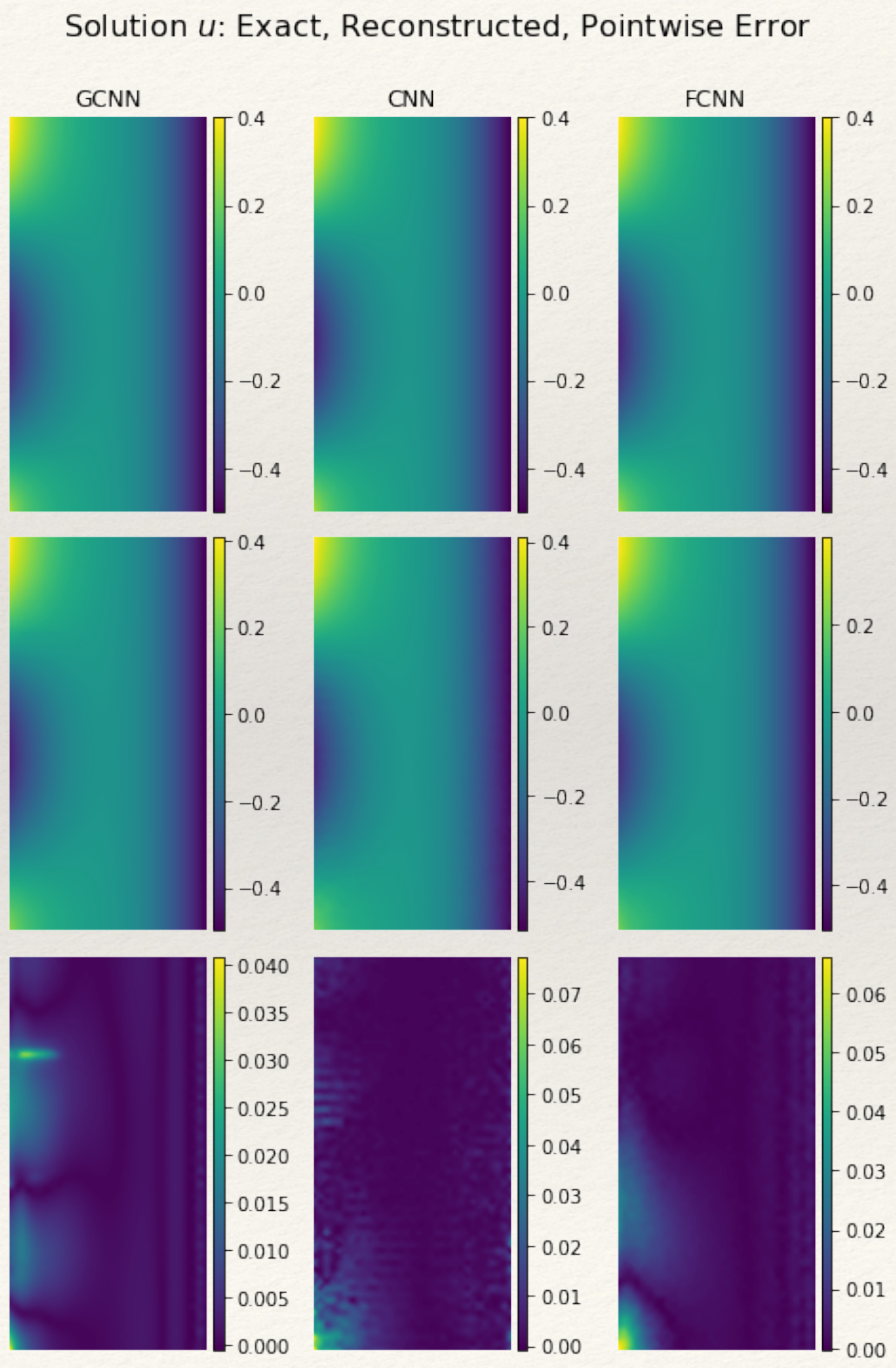
- Consider

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

  and solve

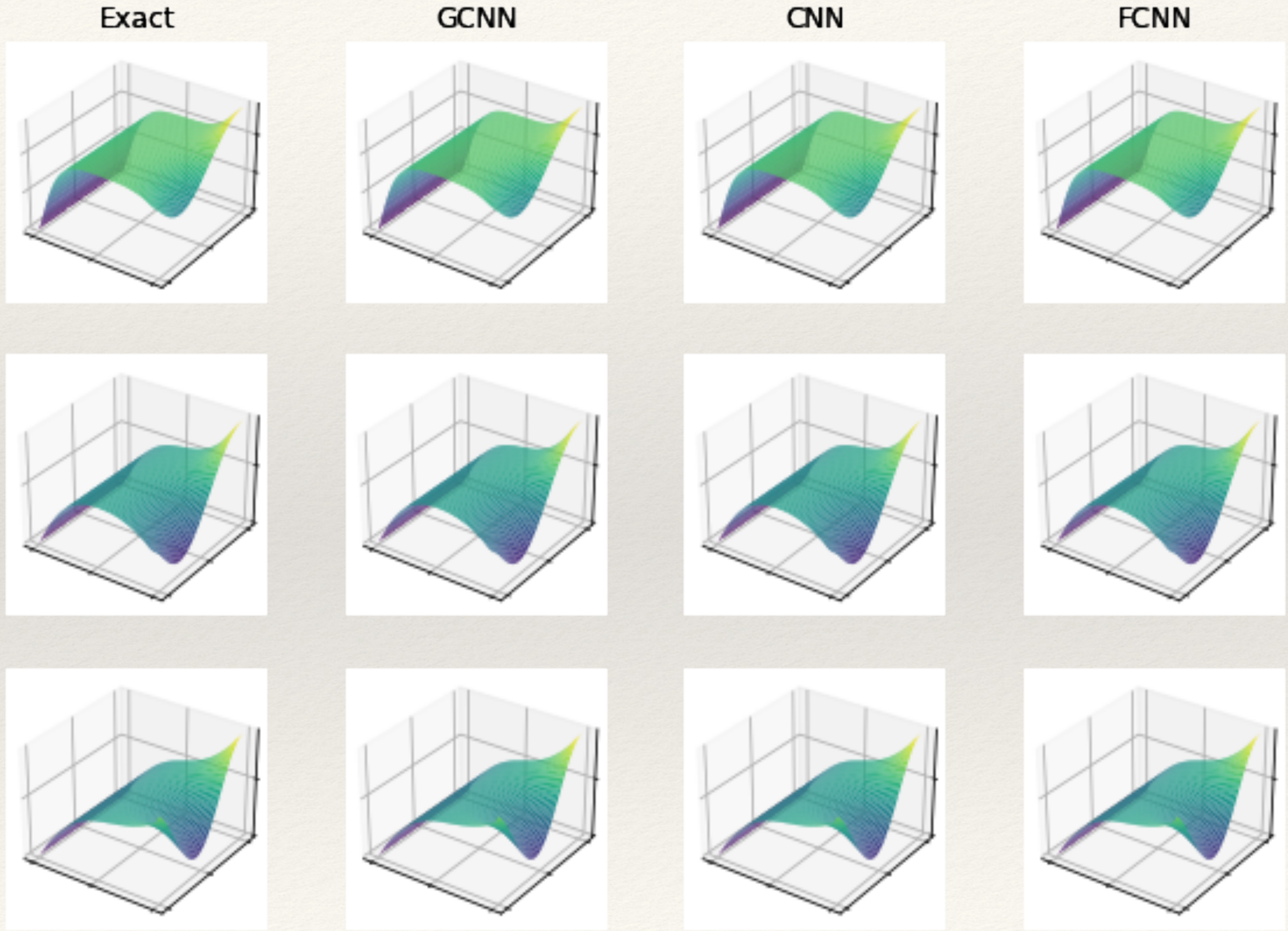
$$\begin{cases} u_t - \Delta u = 0 \quad \text{on } U \\ u(0, y, t) = -0.5 \\ u(1, y, t) = \mu_1 \cos(\mu_2 y) \\ u(x, y, 0) = 0 \end{cases}$$

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



2-D Parameterized Heat Equation: Results

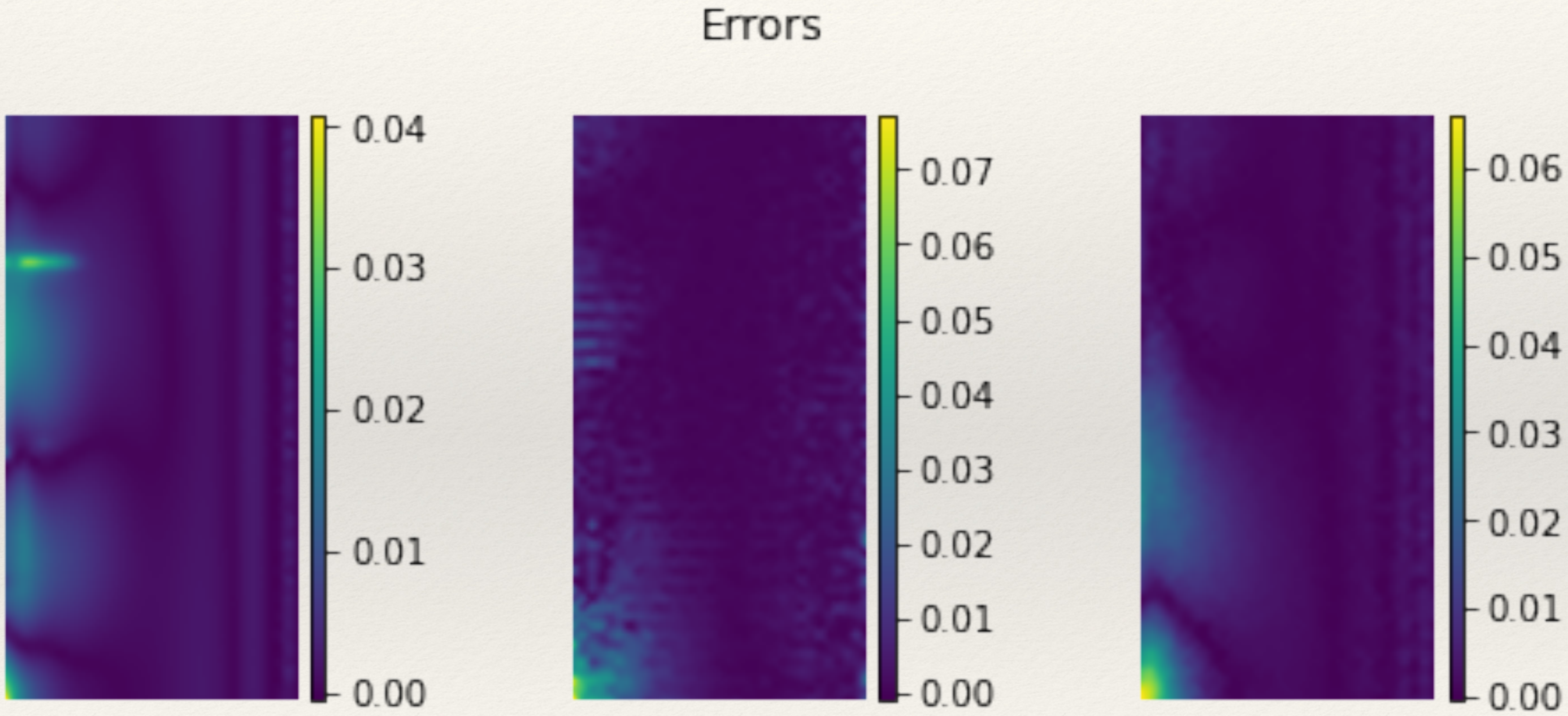
- Results shown for $n = 10$.
- GCNN has lowest error and least memory requirement (by >10x!)
- CNN is worst — cheap hacks have a cost.



2-D Parameterized Heat Equation: 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	3	7.19	9.21	0.132	9.5	3	6.96	9.21	0.0659	9.2
CNN		3.26	4.58	3.64	18		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	10	2.87	3.82	0.253	9.6	10	2.06	2.85	0.186	9.4
CNN		3.07	4.38	3.87	18		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	32	2.55	3.48	0.636	9.6	32	1.05	1.91	0.564	9.2
CNN		2.30	3.73	4.60	19		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 Equation: Results

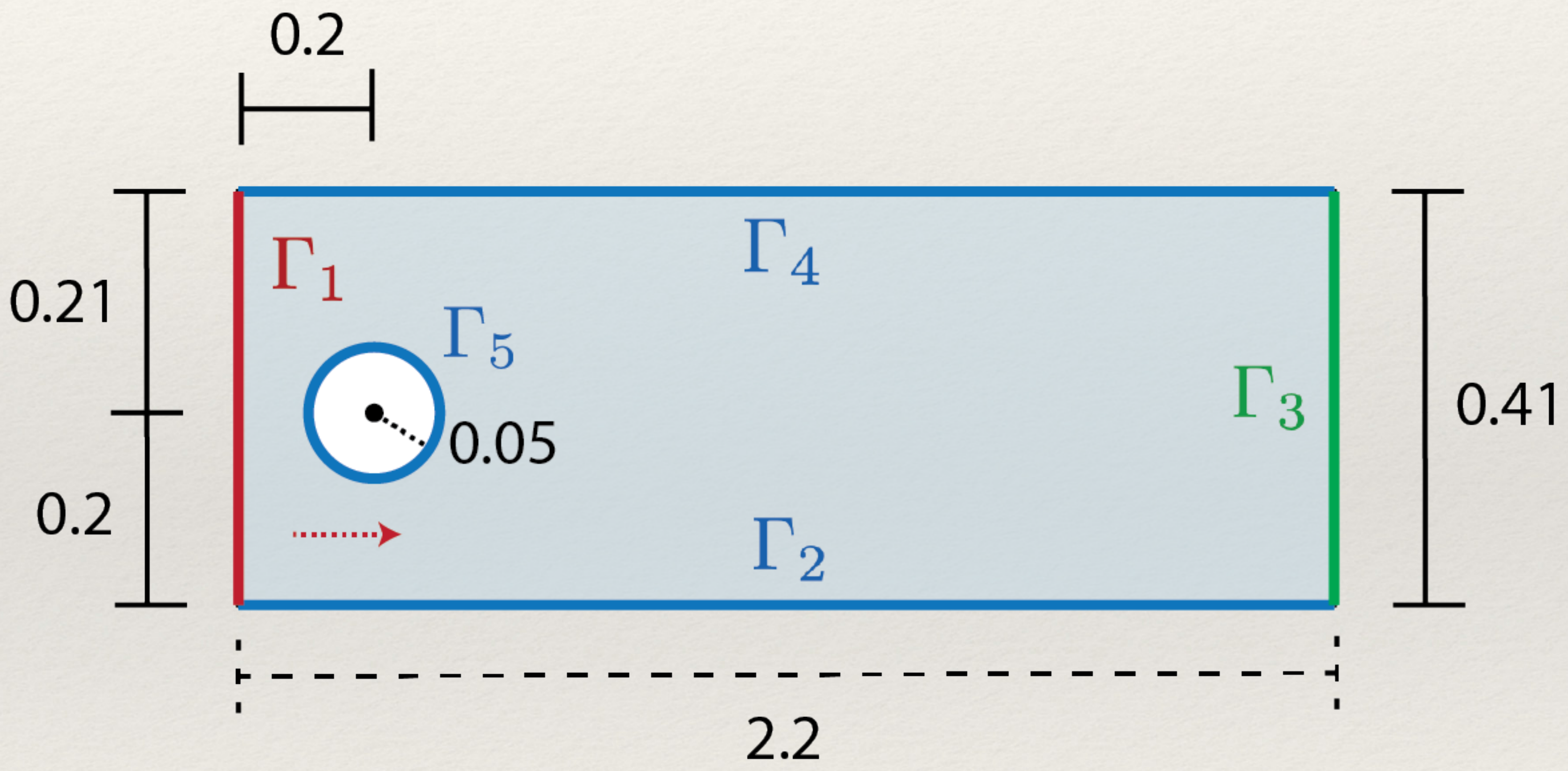


Unsteady Navier-Stokes Equations

- Consider the Schafer-Turek benchmark problem:

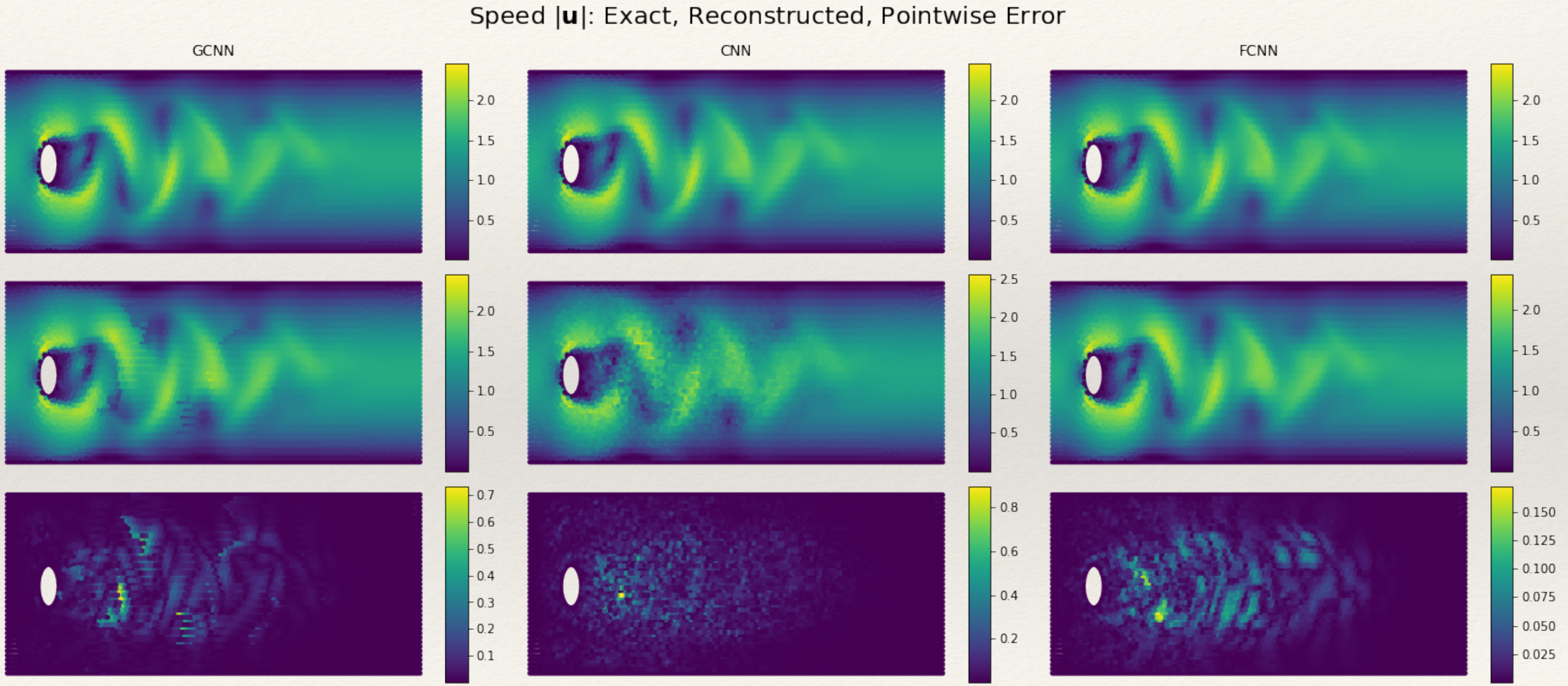
$$\dot{\mathbf{u}} - \nu\Delta\mathbf{u} + \nabla_{\mathbf{u}}\mathbf{u} + \nabla p = \mathbf{f},$$
$$\nabla \cdot \mathbf{u} = 0,$$
$$\mathbf{u}\,\big|_{t=0} = \mathbf{u}_0\,.$$

- 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.

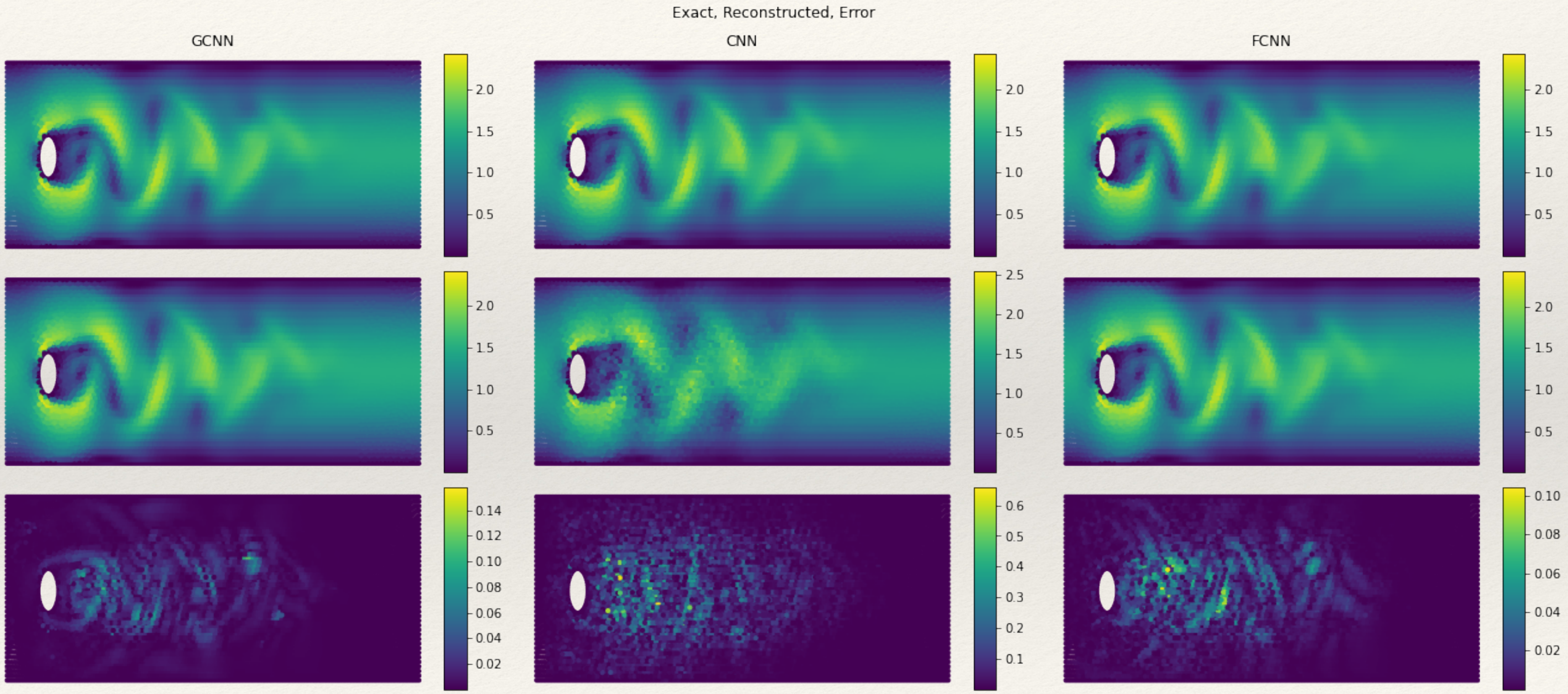


Navier-Stokes Equations: Results

	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



Conclusions

- Standard CNN is not always the best!
  - Even fully connected architectures are better in some cases.
- Graph CNN operations can be useful for ROM.
  - At least, if the latent space is not too small.
- Would be interesting to combine GCNN with Newton/quasi-Newton.

Thank You!