# Machine Learning & Trilinos

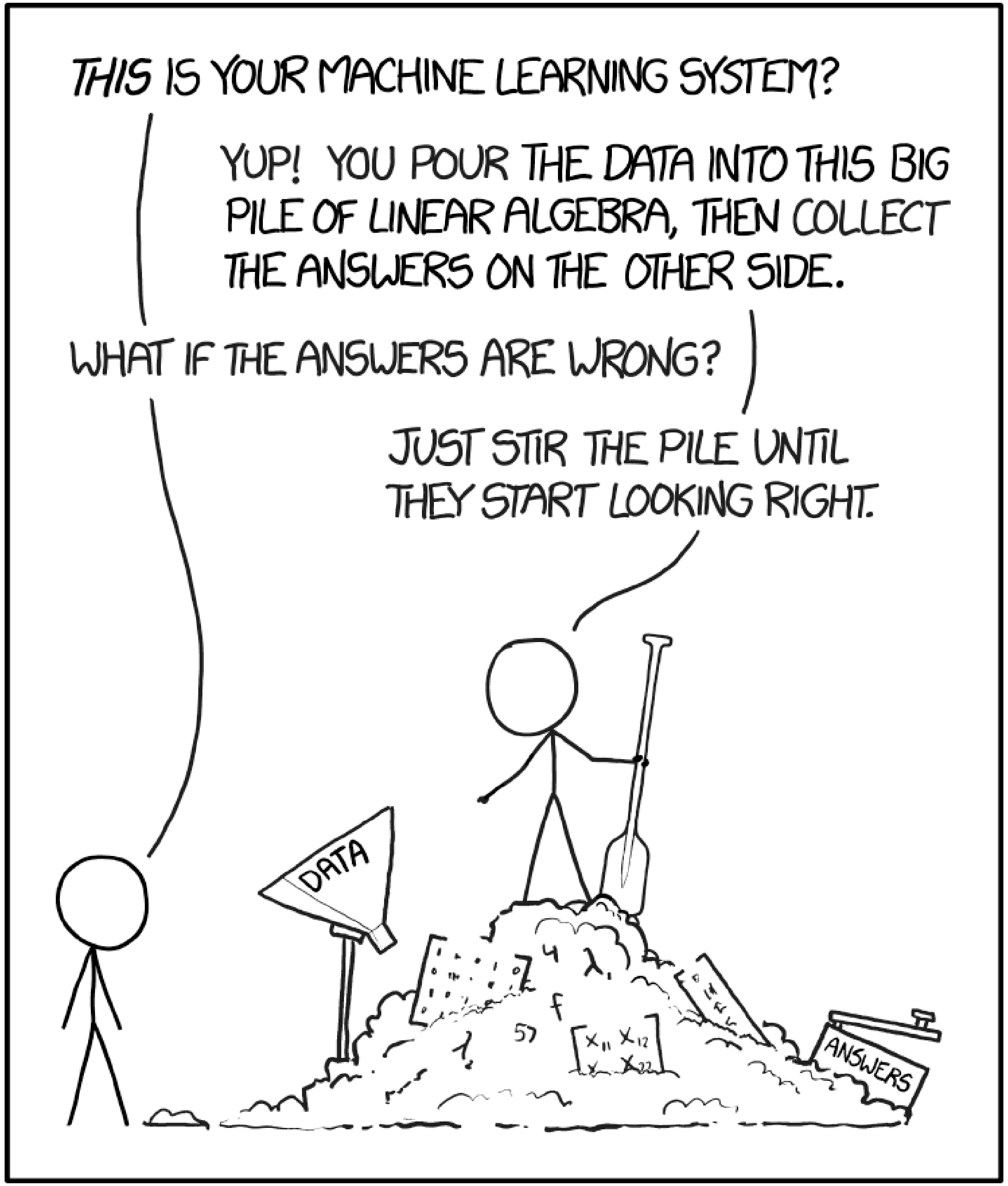
*PRESENTED BY*

Chris Siefert

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2022-14817 C

# As per Randall Munroe…




https://xkcd.com/1838/

# Machine Learning Actually In Trilinos (FY19)

- There's an interface to Avatar Tools in Trilinos/MueLu
  - URL: https://github.com/sandialabs/avatar
  - Lead developer: Philip Kegelmeyer.
  - Ensembles of decision trees with a ton of bells & whistles.
  - C/C++ code (with MPI support).
  - Can be used either as TPL or a external package (ala Drekar).

- Why ensembles of decision trees?
  - Fast & efficient.
  - Work well with small-to-moderate size datasets.
  - Ensembles give a *lot* of accuracy-boosting power.
  - Usually more explainable than neural networks.

# MueLu/Avatar Workflow

## Offline Phase

App: Run lots of problems & features

↓

MueLu: Run lots of options per problem

↓

Avatar: Trains model on offline data

## Online Phase

App: Generates features

↓

MueLu: Call Avatar

↓

Avatar: Performs classification

↓

MueLu: Choose ideal options based on classification

# Simple Example (1)

- Choose a single MueLu parameter based on 4 *mesh* features

| Feature | Training Min/Max | Test disk1 | disk2 | disk3 | expl | tubes |
|---|---|---|---|---|---|---|
| (1) | 1.17 / 100 | 14.9 | 5.19 | 4.08 | 8.25 | 5.11 |
| (2) | 1.08 / 66 | 6.88 | 2.90 | 2.69 | 3.23 | 1.65 |
| (3) | 4.6e-4 / 9.2e-1 | 1.4e-2 | 1.2e-1 | 2.9e-1 | 1.4e-2 | 2.9e-1 |
| (4) | 1.08 / 3.91 | 2.97 | 2.37 | 2.21 | 6.05 | 2.14 |

- These are five test problems unrelated to the training data.

# Simple Example (2)

| Heuristic | disk1 | disk2 | disk3 | expl | tubes |
|---|---|---|---|---|---|
| 1 | 0.005 | 0.025 | 0.025 | 0.00 | 0.025 |
| 2 | 0.025 | 0.025 | 0.025 | 0.00 | 0.025 |
| 3 | 0.005 | 0.01 | 0.025 | 0.00 | 0.01 |

- Since expl had an out-of-bounds feature *no* acceptable options were found by the heuristics.
- Defaulted to a “safe” answer (namely 0).

## Simple Example (3)

- Iteration counts (low = good)

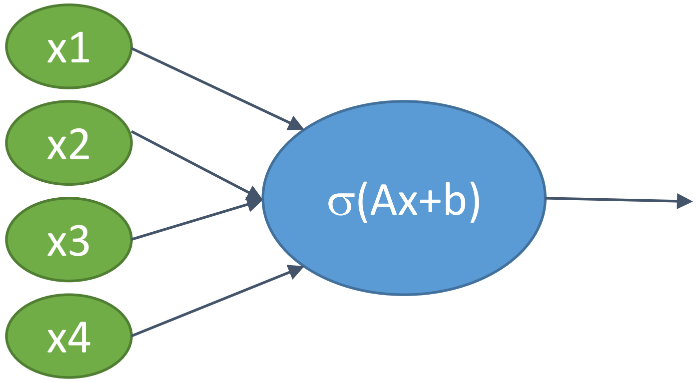
| Heuristic | disk1 | disk2 | disk3 | expl | tubes |
|---|---|---|---|---|---|
| 1 | 37 | 21 | 21 | 30 | 14 |
| 2 | 22 | 21 | 21 | 30 | 14 |
| 3 | 37 | 26 | 21 | 30 | 14 |
| fixed 0 | 57 | 28 | 24 | 30 | 13 |
| fixed 0.01 | 29 | 26 | 31 | 26 | 14 |
| fixed 0.025 | 22 | 21 | 21 | 22 | 14 |

- Heuristic 2 got optimal results *except for out-of-bounds expl.*

# But what about neural networks?

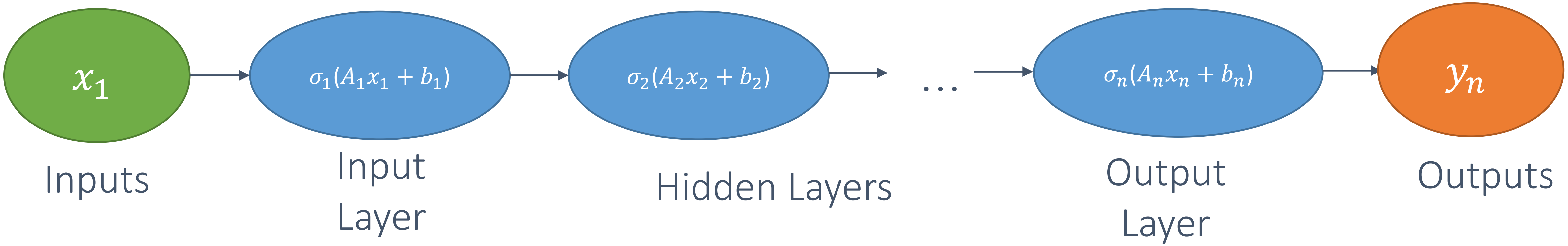
# Basic Neural Networks I

- Building block of a NN is a neuron (or perceptron), often drawn like this:

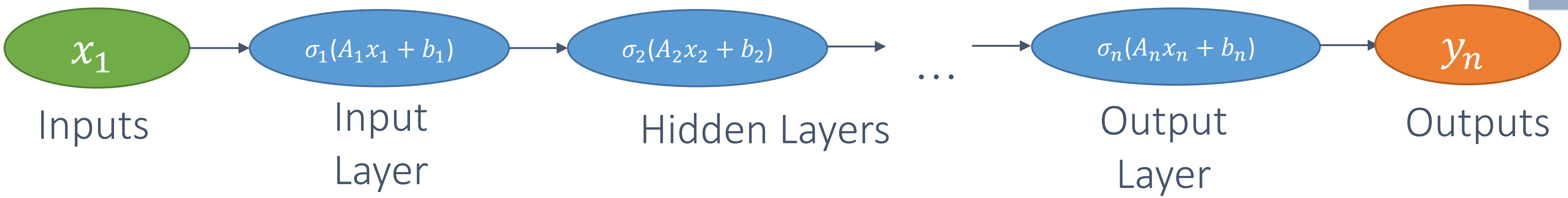



- Some nonlinear function, σ, of a matvec (with weight matrix A) and vector add.

- Feedforward Network / Multi-Layer Perceptron (MLP) is a bunch of these:

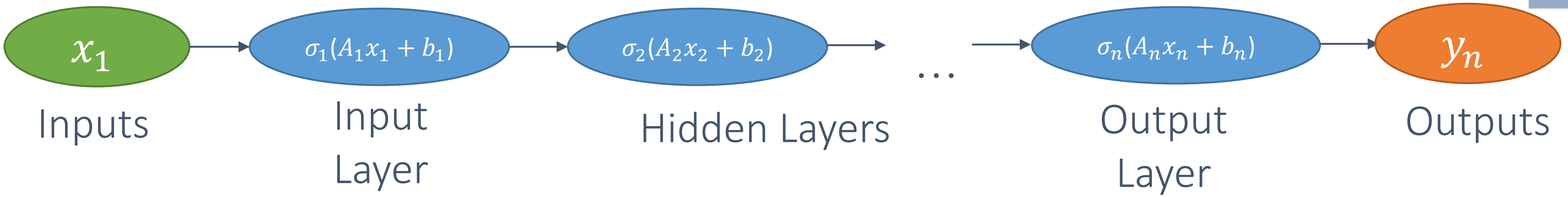


# Basic Neural Networks II




- NNs are often drawn with multiple “blocks” in each layer.
- This means each $A_i$ can be a different size.
- Lots of nonlinear $\sigma_i$ functions are available (e.g. sigmoid, ReLu, etc.).
- Done correctly, this can approximate any continuous function (similar to Stone-Weierstrass for polynomials).

# Basic Neural Networks III




- These models are "trained" by choosing parameters $A_i$ and $b_i$ to satisfy some objective, called a "loss function."

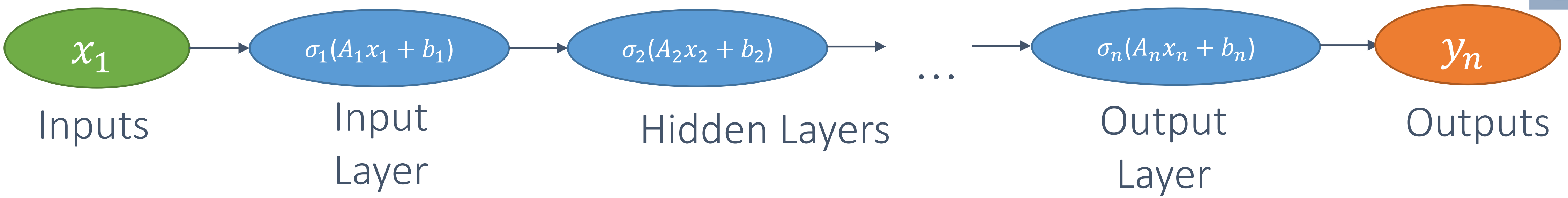
- Given training data, output pairs $(d_i, \hat{y}_i)$, for i=1… m, you might get:

$$\min_{A_1 \ldots, A_n, b_1 \ldots, b_n} \sum_{i=1}^{m} \|M(d_i) - \hat{y}_i\|$$

- Where *M* is the MLP model and $\|.\|$ is the norm of your choice.

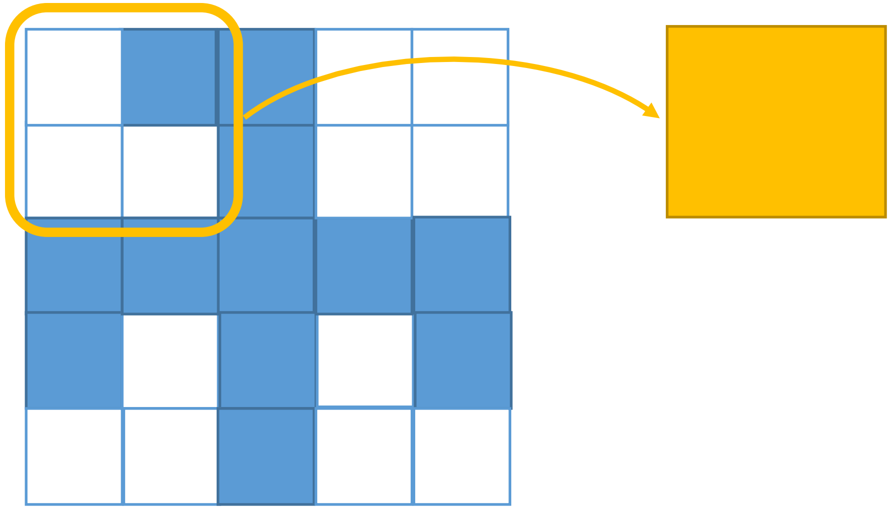
- Note that this model is *dense* or "fully connected."

# Basic Neural Networks IV




- Layers need not be dense / fully connected.

- Convolutional networks (CNNs) connect *spatially* nearby items (tiled over the inputs):



Corresponds to a banded matrix w/ the same values in each row, but shifted.

- Recurrent neural networks (RNN) connect *temporally* nearby items.

# Basic Neural Networks IV

$$\underbrace{x_1}_{\text{Inputs}} \rightarrow \underbrace{\sigma_1(A_1 x_1 + b_1)}_{\text{Input Layer}} \rightarrow \underbrace{\sigma_2(A_2 x_2 + b_2) \rightarrow \cdots}_{\text{Hidden Layers}} \rightarrow \underbrace{\sigma_n(A_n x_n + b_n)}_{\text{Output Layer}} \rightarrow \underbrace{y_n}_{\text{Outputs}}$$

- The downside here is that we've fixed *all* of the dimensions.
  - MLP: The input size is fixed.
  - CNN: The number of neighbors (and their relations is fixed).
  - RNN: The temporal recurrence length is fixed (though this is less of a problem).

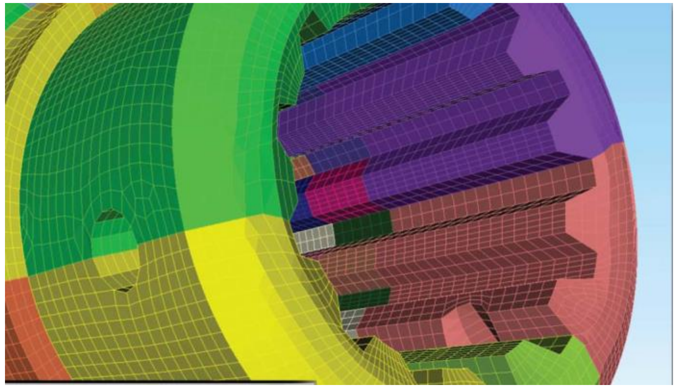


*Image from SNL Lab News, March 2014.*

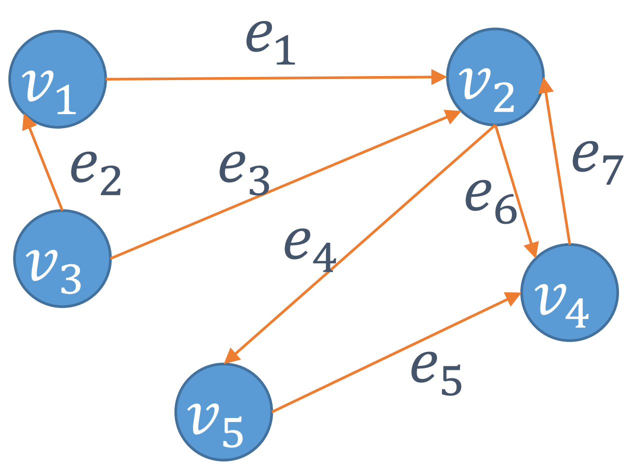
- Our real-world problems don't have fixed meshes!

- For unstructured meshes, this is not going to work unless you are very, very witty

- Alternative Solution: Graph Neural Networks (GNNs).

# Graph Neural Networks I

- Graphs have *vertices* and *edges*.
  - We will consider directed graphs.




u1 u2

- GNNs add…
  - Immutable attributes associated w/ each vertex + edge.
  - Mutable attributes associated with each vertex + edge.
  - Global variables with associated attributes.

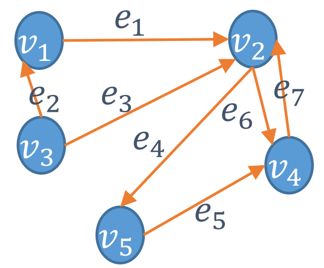
Sample GNN with:
- 5 vertices (1 attribute each)
- 7 edges (2 attributes each)
- 2 globals (5 attributes each)

| | | | | | |
|---|---|---|---|---|---|
| u1 | | | | | |
| u2 | | | | | |

| | |
|---|---|
| v1 | |
| v2 | |
| v3 | |
| v4 | |
| v5 | |

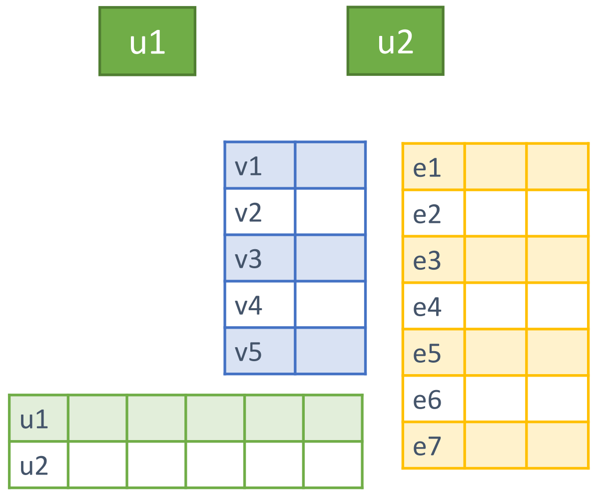
| | | |
|---|---|---|
| e1 | | |
| e2 | | |
| e3 | | |
| e4 | | |
| e5 | | |
| e6 | | |
| e7 | | |

# Graph Neural Networks II




- A GNN block has 3 update functions and 3 aggregation functions

- Updates (trained)
  - $\phi^v$: Update vertex from its attributes, aggregated edge attributes, and globals.
  - $\phi^e$: Update edge from its attributes, neighboring vertex attributes, and globals.
  - $\phi^u$: Update globals from aggregated edge and vertex attributes.
  - **Fixed size inputs and outputs.**

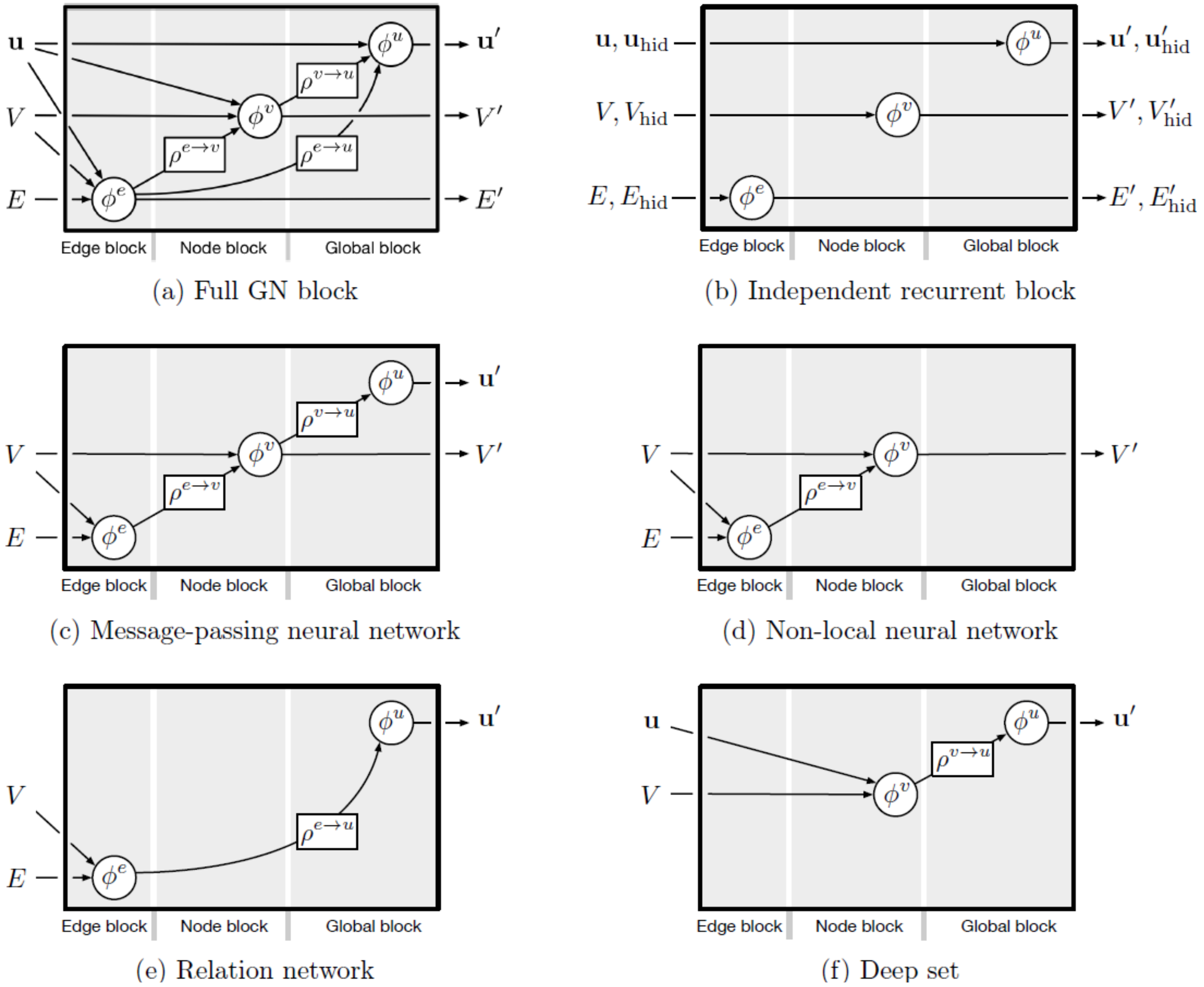



- Aggregation functions (not trained)
  - $\rho^{e \to v}$: Aggregate edge neighbors to vertex.
  - $\rho^{e \to u}$: Aggregate all edges to globals.
  - $\rho^{v \to u}$: Aggregate all nodes to globals.
  - **Variable size inputs and fixed size outputs.**
  - *Note: Since each edge always has two neighboring nodes, node attributes are not aggregated to edges.*

# Graph Neural Networks III

These can be composed in almost any order.

Most general




(a) Full GN block

(b) Independent recurrent block

(c) Message-passing neural network

(d) Non-local neural network

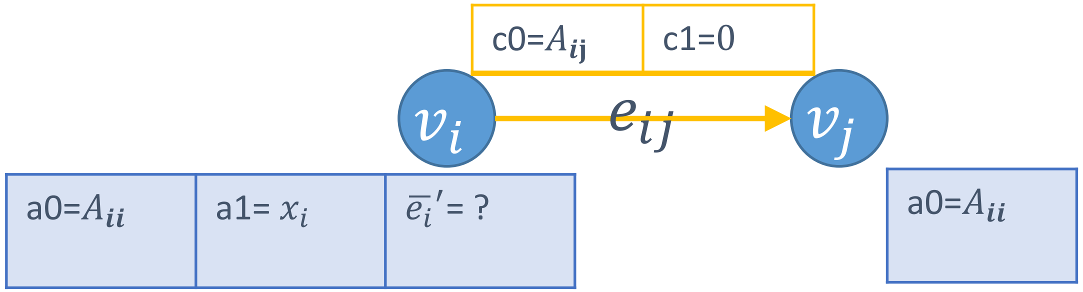
(e) Relation network

(f) Deep set

Image from: Battaglia et al. *Relational inductive biases, deep learning and graph networks.* arXiv:1806.01261v3, 2018

# Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

- Let's consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal entries are considered edges).




- Vertex *i* attributes:
  - a0: 1 immutable, $A_{ii}$.
  - a1: 1 mutable, $x_i$ on input, $b_i$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
  - c1: 1 mutable, 0 on input, $A_{ij}\ x_j$ on output.
- Global attributes: none

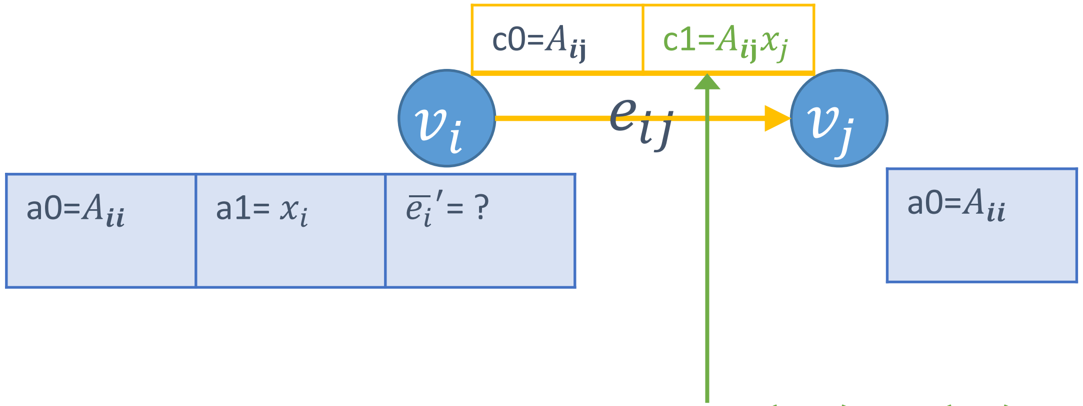
- Update Edges: $\mathrm{c1} = \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$
- Agg Edge-to-Node: $\bar{e_i}' = \rho^{e\to v}(e_{ij}\ \forall j) = \sum_{j\neq i} c1(e_{ij})$
- Nodes: $\mathrm{a1} = \phi^v(\bar{e_i}', v_i) = a0(v_i)\ a1(v_i) + \bar{e_i}'$

This is a trivial example and has no parameters, but…

# Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

- Let's consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal nodes are considered edges).

- Vertex *i* attributes:
  - a0: 1 immutable, $A_{ii}$.
  - a1: 1 mutable, $x_i$ on input, $b_i$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
  - c1: 1 mutable, 0 on input, $A_{ij}\ x_j$ on output.
- Global attributes: none



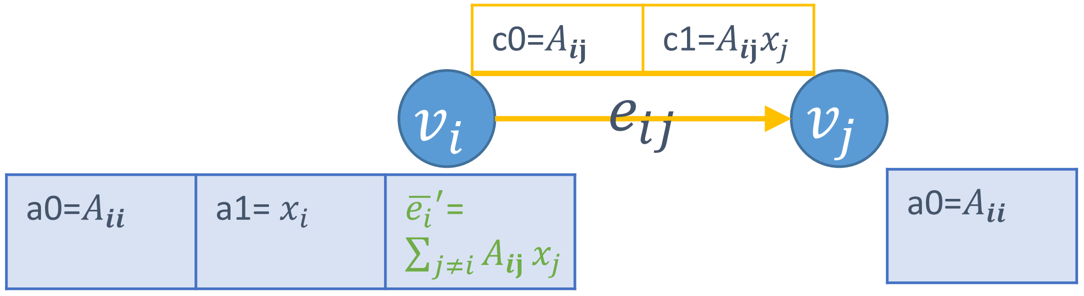

- Update Edges: $\mathbf{c1} = \boldsymbol{\phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)}$
- Agg Edge-to-Node: $\bar{e_i}' = \rho^{e\to v}(e_{ij}\ \forall j) = \sum_{j\neq i} c1(e_{ij})$
- Nodes: $\mathrm{a1} = \phi^v(\bar{e_i}', v_i) = a0(v_i)\ a1(v_i) + \bar{e_i}'$

This is a trivial example and has no parameters, but…

# Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

- Let's consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal nodes are considered edges).

- Vertex *i* attributes:
  - a0: 1 immutable, $A_{ii}$.
  - a1: 1 mutable, $x_i$ on input, $b_i$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
  - c1: 1 mutable, 0 on input, $A_{ij}\, x_j$ on output.
- Global attributes: none



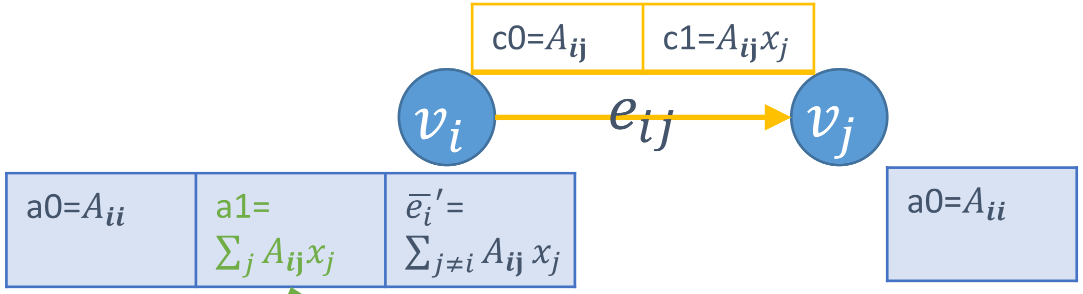

- Update Edges: $c1 = \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$
- **Agg Edge-to-Node:** $\boldsymbol{\bar{e_i}' = \rho^{e\to v}(e_{ij}\ \forall j) = \sum_{j\neq i} c1(e_{ij})}$
- Nodes: $a1 = \phi^v(\bar{e_i}', v_i) = a0(v_i)\, a1(v_i) + \bar{e_i}'$

This is a trivial example and has no parameters, but…

# Matrix-as-a-Graph: Matrix-Vector Product Example (Ax = b)

- Let's consider a very simple example: a sparse matrix vector product.
- Remember: A matrix is a graph (off-diagonal nodes are considered edges).

- Vertex *i* attributes:
  - a0: 1 immutable, $A_{ii}$.
  - a1: 1 mutable, $x_i$ on input, $b_i$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
  - c1: 1 mutable, 0 on input, $A_{ij}\, x_j$ on output.
- Global attributes: none

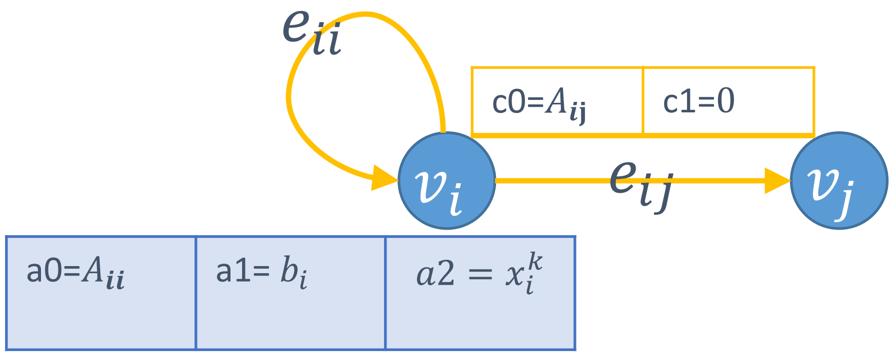



- Update Edges: c1 $= \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$
- Agg Edge-to-Node: $\bar{e_i}' = \rho^{e\to v}(e_{ij}\ \forall j) = \sum_{j\neq i} c1(e_{ij})$
- Nodes: $\mathbf{a1} = \boldsymbol{\phi}^{v}(\bar{e_i}', v_i) = \boldsymbol{a0}(v_i)\ \boldsymbol{a1}(v_i) + \bar{e_i}'$

This is a trivial example and has no parameters, but…

# Iterative Method: Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - \mathrm{A}\, x_k)$

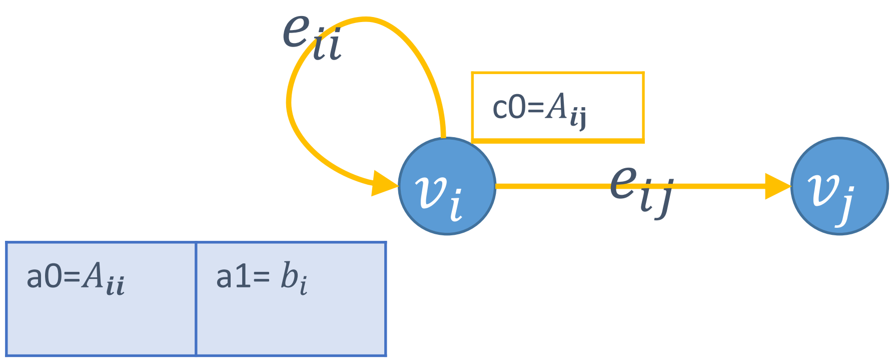
- Vertex *i* attributes:
  - a0, a1: 2 immutable, $A_{ii}$, $b_i$
  - a2: 1 mutable, $x_i^k$ on input, $x_i^{k+1}$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
  - c1: 1 mutable, 0 on input, $A_{ij}\, x_i^k$ on output.
- Global attributes:
  - d0: 1 immutable: $\omega$.




- Update Edges: $\mathrm{c1} = \phi^e(e_{ij}, v_i, v_j) = c0(e_{ij})a1(v_j)$
- Agg Edge-to-Node: $\bar{e}_i{}' = \rho^{e\to v}(e_{ij}\ \forall j) = \sum_j c1(e_{ij})$
- Nodes: $\mathrm{a1} = \phi^v(\bar{e}_i{}', v_i) = a2(v_i) + \frac{d0(a1(v_i) - \bar{e}_i{}')}{a0(v_i)}$

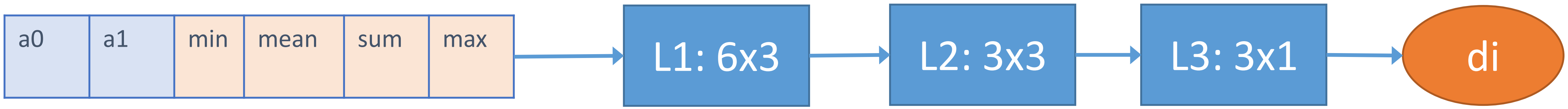
# A Trainable Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - A\, x_k)$

- Vertex *i* attributes:
  - a0: 1 immutable, $A_{ii}$,
  - a1: 1 mutable, $b_i$ on input, $d_i$ on output.
- Edge *(i,j)* attributes:
  - c0: 1 immutable, $A_{ij}$.
- Global attributes:
  - d0: 1 immutable: $\omega$.




- Special Edge aggregation:
  - Compute min, mean, sum & max of edges.

- Agg Edge-to-Node: $\bar{e}_i{}' = \rho^{e \to v}(e_{ij}\ \forall j) = [min, mean, sum, max]$
- Nodes: a1 $= \phi^v(\bar{e}_i{}', v_i) =$ *3 Level NN*




Goal: Choose local damping in lieu of doing an EV estimate for $\omega$

# A Trainable Jacobi Iteration: $x_{k+1} = x_k + \omega D^{-1}(b - A\, x_k)$

- Per-matrix loss function: Damping Factor

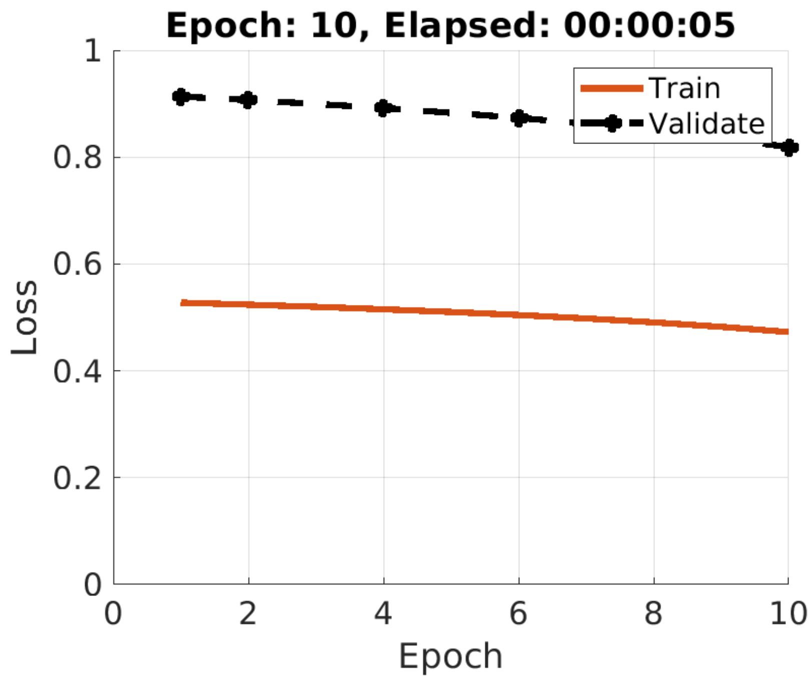
$$\left\| I - \omega D^{-1} A \right\|_2$$

- *D=GNN(A)* here is our trained "diagonal."

- Toy data: 5x5 FEM Laplacians w/ varying y stretch.

- Method: 10 epochs of Adam w/ LR=0.01.

- Test Data:
  - DF Fixed $\omega = 2/3$: 0.82.
  - Trained Diagonal DF: 0.79.
  - Optimal $\omega$ DF: 0.77.




(Done in Matlab)

# Where can we go from here?

- Many matrix algorithms can be recast as GNNs!
  - OK.  Not anything ordering dependent like Gauss-Seidel, but still.
- Once we have that, we can turn the crank on AI/ML to…
  - Choose parameters.
  - *Combine* multiple objective functions.
  - Create *new* data-driven algorithms inspired by (and informed by) old ones.
- GNN software is available in…
  - PyTorch: Geometric, Deep Graph Library.
  - TensorFlow: graph_nets and TensorFlow GNN.
  - Matlab: Deep Learning Toolbox.
- We hope to see this class of algorithms make their way into Trilinos/MueLu!

Sample GNNs Apps
- Node/Graph classification
- Computer vision (object relations)
- Natural language
- Traffic forecasting
- Recommender systems
- Molecular structure

Warning: Software support for GNNs is still a little patchy!