Sparse Deep Neural Network Graph Challenge
- draft -

Jeremy Kepner, Simon Alford, Vijay Gadepally, Michael Jones, Lauren Milechin, Ryan Robinett, Sid Samsi

June, 2019

MIT LINCOLN LABORATORY
SUPERCOMPUTING CENTER

Outline

- **Introduction**
- **Deep Neural Networks**
- **Sparse Data Sets**
- **Challenge**
- **Summary**

MIT/IEEE/Amazon GraphChallenge.org Goals

- GraphChallenge encourages community approaches to developing new solutions for analyzing graphs and sparse data

- Goal: to provide a well-defined community venue for stimulating research and highlighting innovations in graph analysis software, hardware, algorithms, and systems

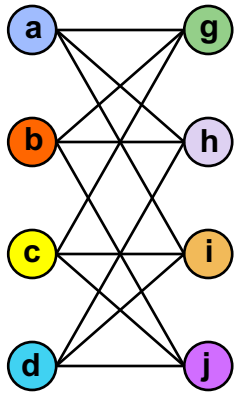
- Audience: any individual or team that seeks to highlight their contributions to graph analysis software, hardware, algorithms, and/or systems

- Graph Challenge contributions should be submitted to the IEEE HPEC conference submission website (see ieee-hpec.org)

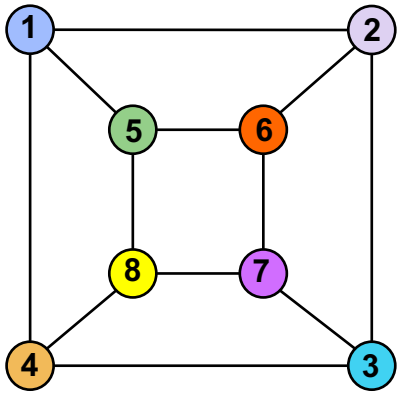
Current Graph Challenges

- **Static Graph Challenge: Sub-Graph Isomorphism**
  - This challenge seeks to identify a given sub-graph in a larger graph
  - Triangle counting and K-Truss

Graph H



Graph G

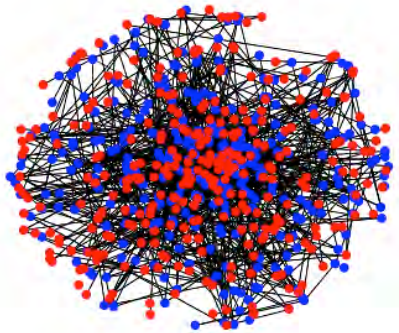


Isomorphism

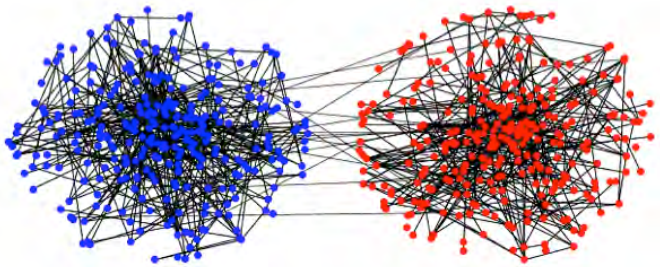
f (a) = 1
f (b) = 6
f (c) = 8
f (d) = 3
f (g) = 5
f (h) = 2
f (i) = 4
f (j) = 7

- **Streaming Graph Challenge: Stochastic Block Parition**
  - This challenge seeks to identify optimal blocks (or clusters) in a larger graph

Streaming unstructured graph



Block optimized graph



GraphChallenge.org

GraphChallenge

Motivation | Challenges | Data Sets | Scenarios | Submit | News | Champions | Contact

Motivation

Challenges such as YOHO, MNIST, HPC Challenge, ImageNet, and VAST have played important roles in driving progress in fields as diverse as machine learning, high performance computing, and visual analytics.

GraphChallenge encourages community approaches to developing new solutions for analyzing graphs and sparse data derived from social media, sensor feeds, and scientific data to enable relationships between events to be discovered as they unfold in the field.

Graph Challenge Details

IEEE HPEC 2018 Graph Challenge Submitter Guidlines

- **May 18: regular HPEC paper deadline**
- **July 15 (anywhere on Earth): Graph Challenge regular and update submission deadline**

Analysis of all 2017 Triangle Counting Submissions

Links to Graph Challenge benchmark specifications, data and sample code

Graph Datasets

Wikipedia

Twitter

Stanford Network Analysis Project

SNAP

Amazon AWS Public Datasets

amazon web services

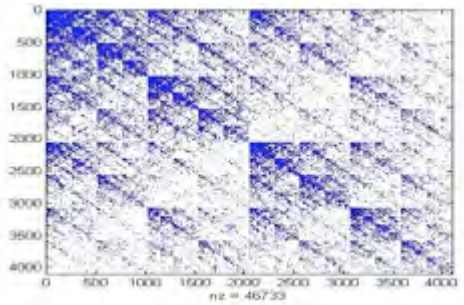
Road Networks

Autonomous Systems

Yahoo!

WEBSCOPE Datasets

Graph500.org



- Public data available from a variety of sources
- Data formats are typically not standardized (we have normalized!)

Graph Datasets Hosted on Amazon

SNAP Datasets (*click to expand*)

Name	Description
amazon0302	Amazon product co-purchasing network from March 2 2003
	Adjacency TSV Incidence TSV Adjacency MMIO Incidence MMIO
amazon0312	Amazon product co-purchasing network from March 12 2003
amazon0505	Amazon product co-purchasing network from May 5 2003
amazon0601	Amazon product co-purchasing network from June 1 2003

Partition Challenge Datasets with Known Truth Partitions (*click to expand*)

Name	Description
Static Graphs (small)	Small Static Graphs with known truth for the Stochastic Block Partitioning Challenge
50 nodes	100 nodes 500 nodes 1000 nodes 5000 nodes
Static Graphs (large)	Large Static Graphs with known truth for the Stochastic Block Partitioning Challenge
Streaming - Edge Sampling (small)	Small Streaming Graphs with known truth for the Stochastic Block Partitioning Challenge
Streaming - Edge Sampling (large)	Large Streaming Graphs with known truth for the Stochastic Block Partitioning Challenge
Streaming - Snowball Sampling (small)	Small Streaming Graphs with known truth for the Stochastic Block Partitioning Challenge

Data available for download in multiple formats

- **Adjacency and Incidence matrix for each graph**
- **File format : ASCII Tab separated file and MMIO format**

Small graphs with known partitions available for block partitioning algorithm

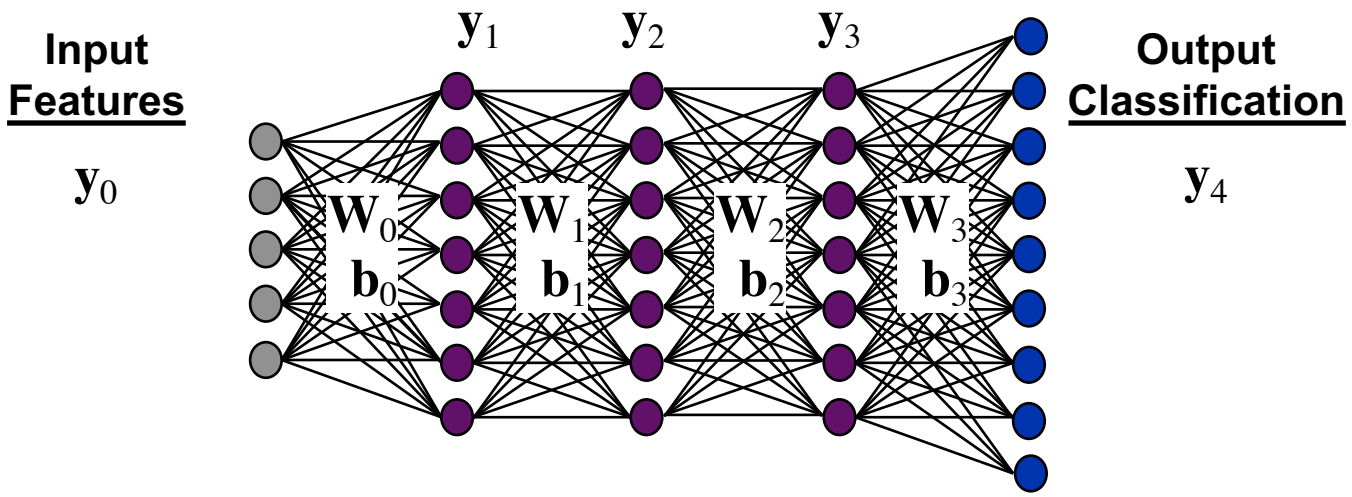
MMIO – Matrix Market I/O format. Available http://math.nist.gov/MatrixMarket/

New Challenge Goal: Enable Large Sparse Deep Neural Networks

- **Deep neural networks (DNNs) are at the heart of modern AI miracles**
- **Larger neural networks often perform better**
  - **Larger number of layers/features allow more non-linear boundaries**
  - **Problem: limited by expensive high-speed memory size**
  - **Solution: sparse (pruned) neural networks deliver comparable performance with less memory**

Dense Training	
Dense Training, then Pruning	LeCun *et al.*, Hassibi *et al.*
Dense Training while Pruning	Srinivas *et al.*, Han *et al.*
Sparse Training	Prabhu *et al.* (X-Net)

Density ↕ Sparsity



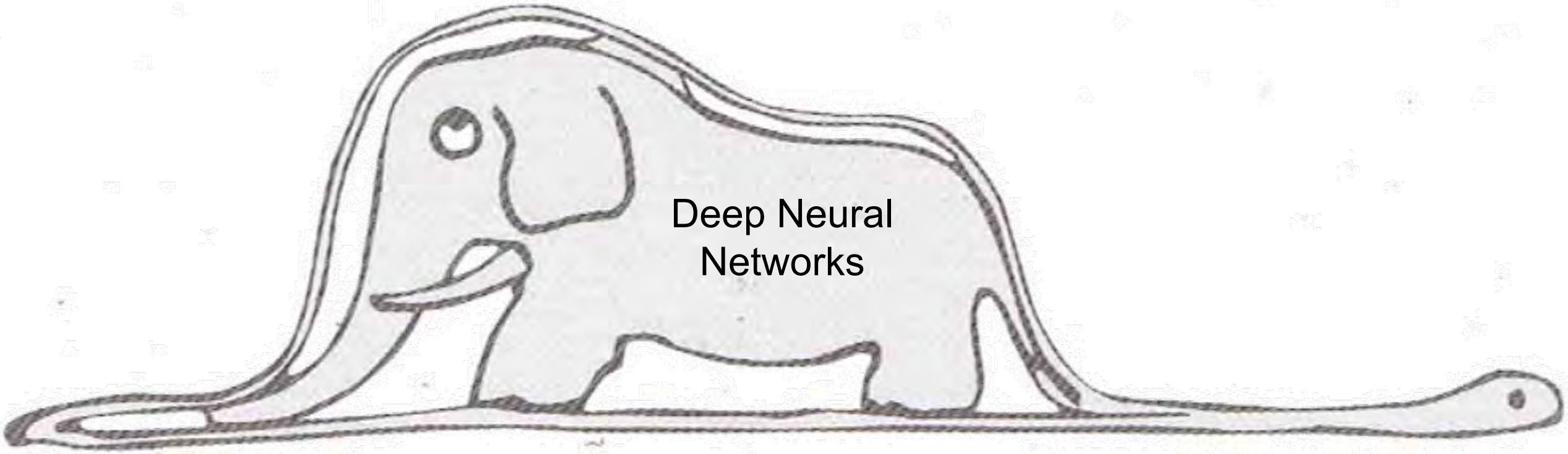
Outline

- Introduction
- **Deep Neural Networks**
- Sparse Data Sets
- Challenge
- Summary

Before 2010

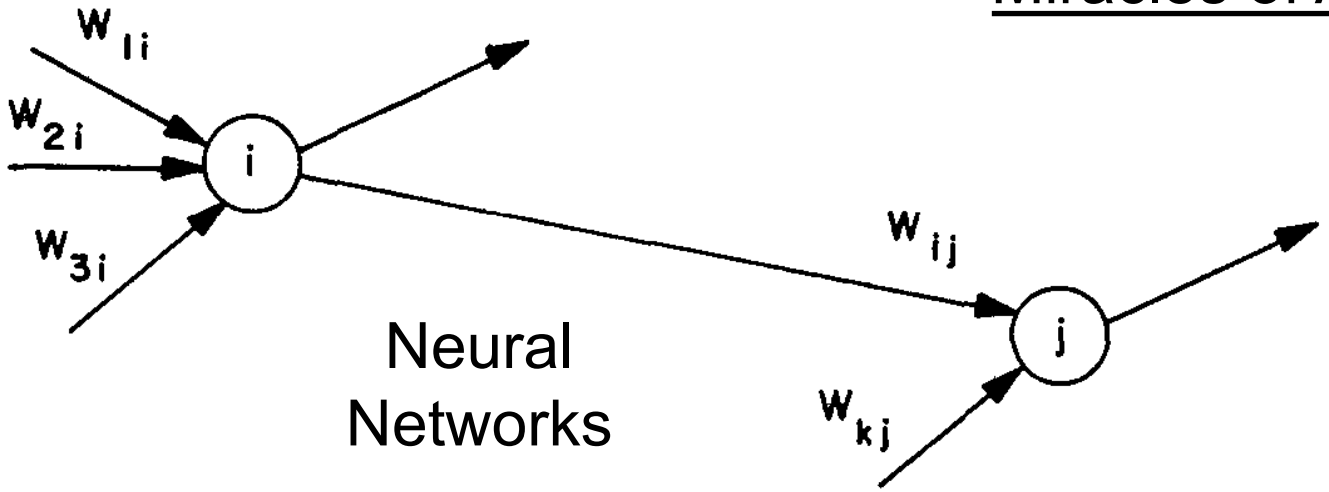
Machine Learning

After 2015

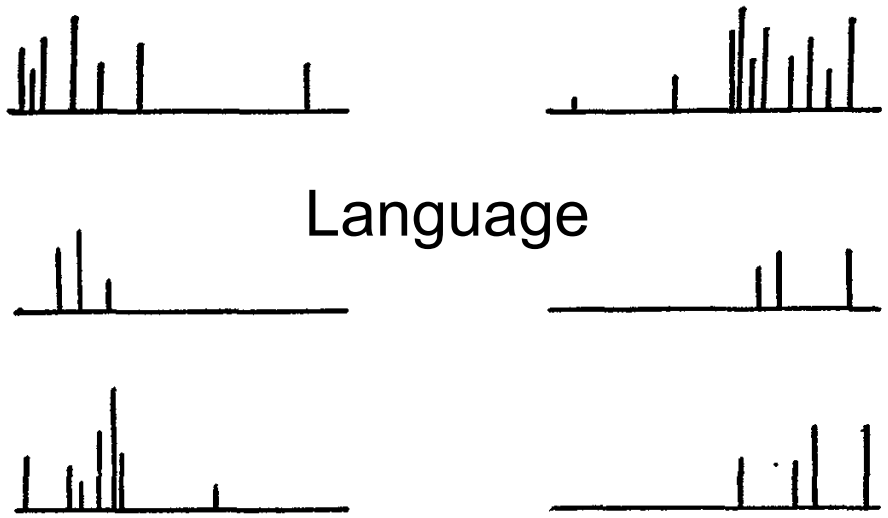


Machine Learning

Miracles of AI

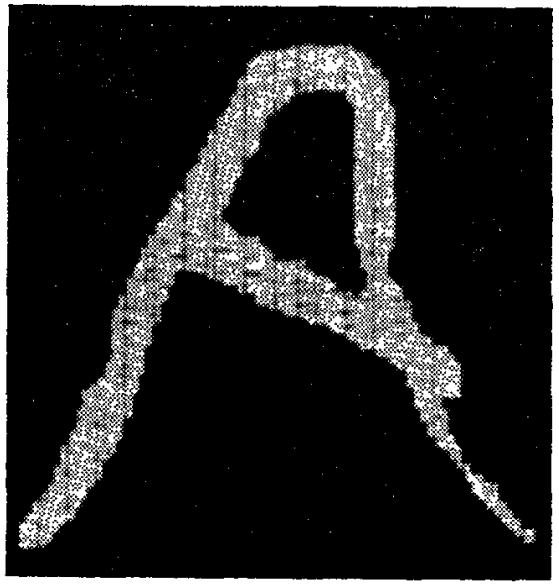


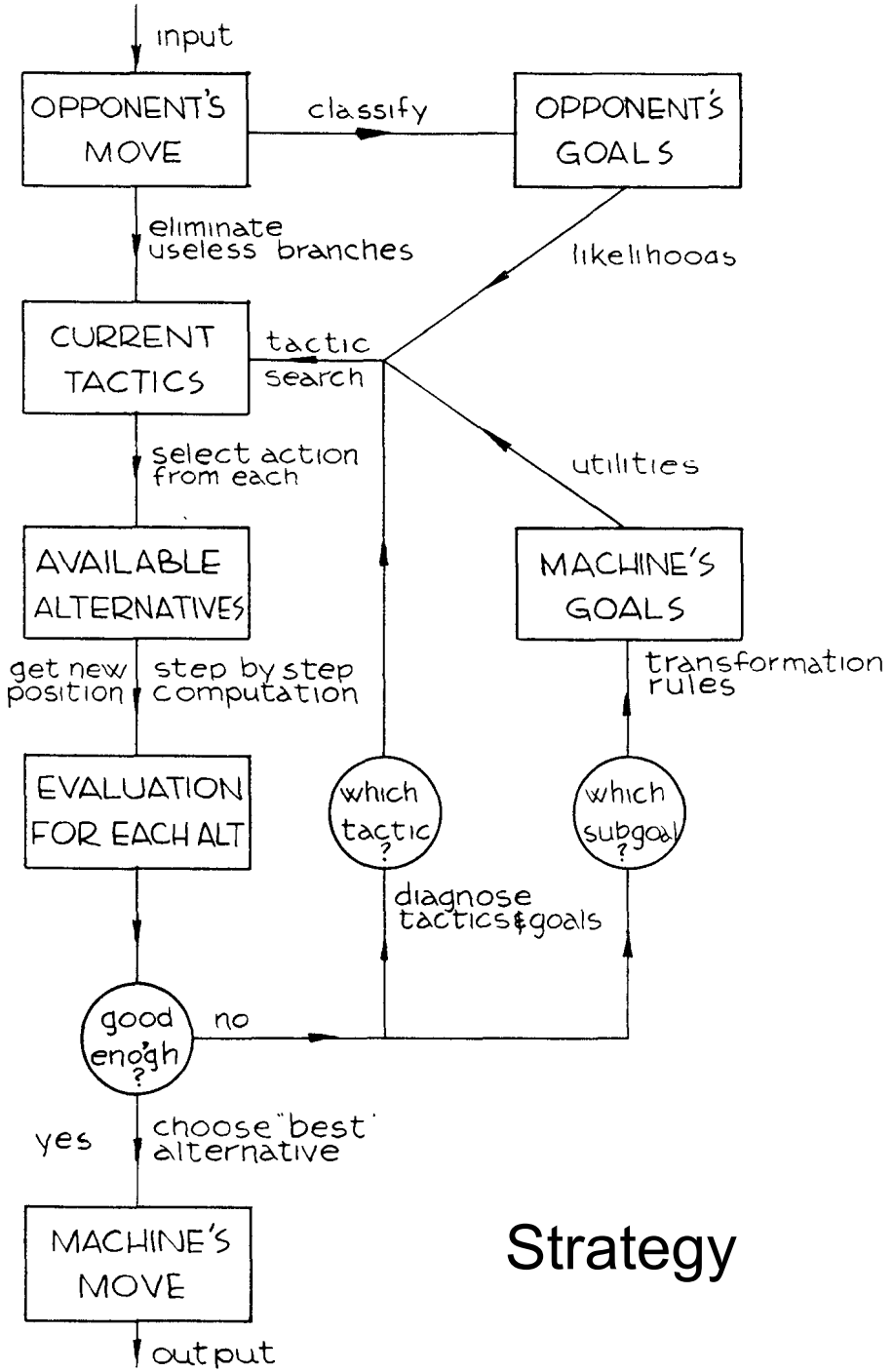
Neural Networks



Language

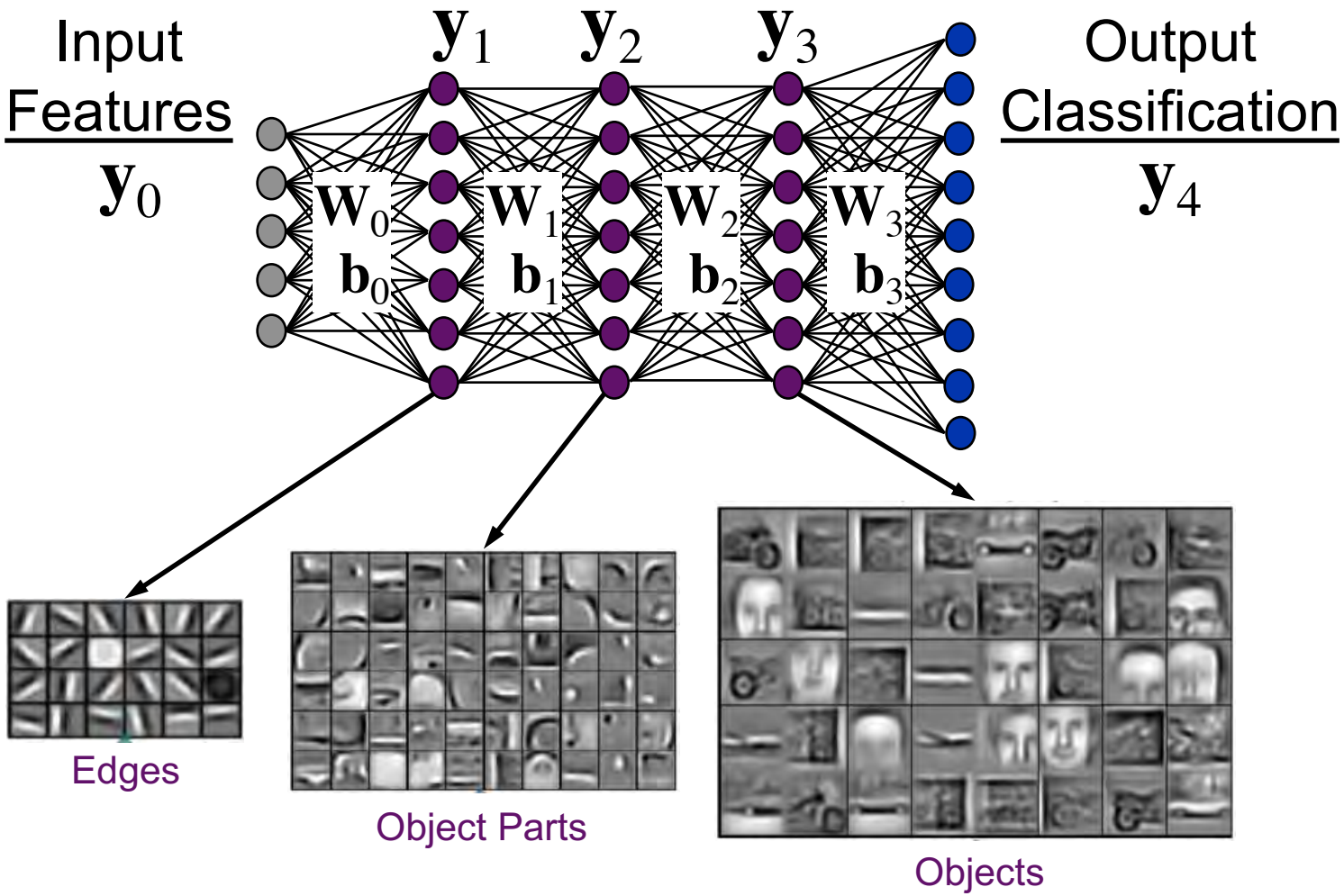
Vision





Strategy

Deep Neural Network



$$\mathbf{y}_{l+1} = \mathrm{h}(\mathbf{y}_l \mathbf{W}_l + \mathbf{b}_l)$$

Graph Challenge Math Conventions

$$\mathbf{Y}_{l+1} = h(\mathbf{Y}_l \mathbf{W}_l + \mathbf{b}_l)$$

- $\mathbf{Y}_0$ is (# inputs) x (# features); each <u>row</u> is a feature vector
  - # features = # neurons/layer (constant for all layers)
- $\mathbf{W}_l$ is (# neurons) x (# neurons); $\mathbf{W}_l(i,j) > 0$ means a connection from i to j
- $\mathbf{b}_l$ is a bias row vector applied to each input
- $h(x)$ is the ReLU function with a max cutoff

  $0 < x < 32$, x is unchanged; $x < 0$, x is changed to 0; $x > 32$, x is changed to 32

- Sparse DNN Challenge uses standard graph community terminology
  - Row vectors for features and left matrix multiply to progress through network
- Standard AI definitions can be used (transpose all matrices and multiply on right)

Next Frontier: Sparse Neural Networks

- **Larger neural networks often perform better**
  - **Larger number of layers/features allow more non-linear boundaries**
  - **Challenge: limited by GPU memory size**
  - **Solution: sparse neural networks**
- **Active research area in AI community**

Published as a conference paper at ICLR 2018

EFFICIENT SPARSE-WINOGRAD CONVOLUTIONAL NEURAL NETWORKS

Xingyu Liu[*], **Jeff Pool**[†], **Song Han**[‡§], **William J. Dally**[*†]
[*] Stanford University, [†] NVIDIA, [‡] Massachusetts Institute of Technology, [§] Google Brain
{xyl, dally}@stanford.edu

Exploring the Regularity of Sparse Structure in Convolutional Neural Networks

Huizi Mao[1], **Song Han**[1], **Jeff Pool**[2], **Wenshuo Li**[3], **Xingyu Liu**[1], **Yu Wang**[3], **William J. Dally**[1,2]
[1]Stanford University
[2]NVIDIA
[3]Tsinghua University
{huizi,songhan,dally}@stanford.edu

SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks

Angshuman Parashar[†] Minsoo Rhu[†]
Anurag Mukkara[‡] Antonio Puglielli[*] Rangharajan Venkatesan[†] Brucek Khailany[†]
Joel Emer[†‡] Stephen W. Keckler[†] William J. Dally[†◇]
NVIDIA[†] Massachusetts Institute of Technology[‡] UC-Berkeley[*] Stanford University[◇]

GPU = Graphics Processing Unit

Outline

- Introduction
- Deep Neural Networks
- **Sparse Data Sets**
- Challenge
- Summary

Sparse DNN Data

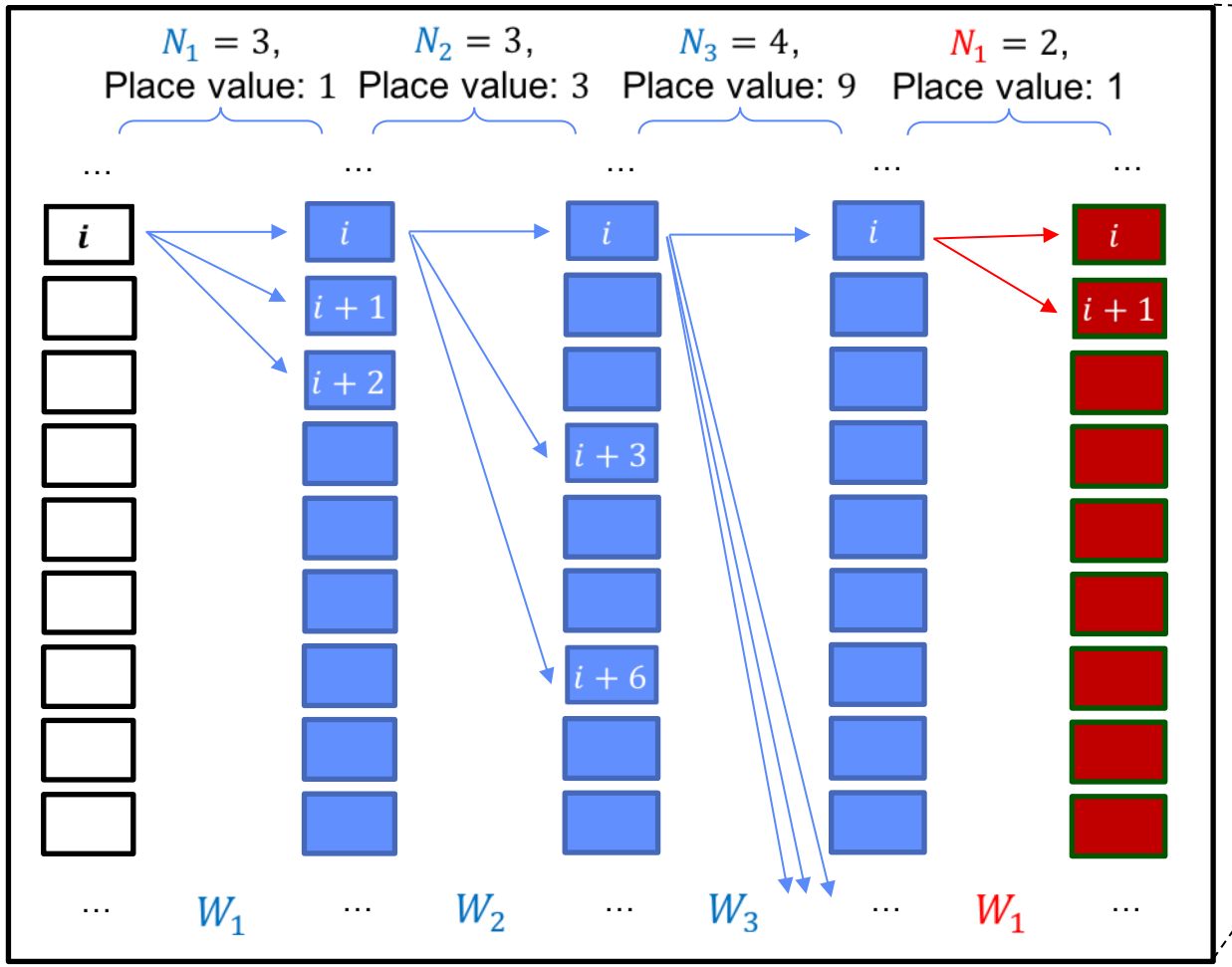
- **Large sparse DNNs are difficult to obtain from real data**
  - **Similar to early days of Graph500.org**
  - **First step is to simulate data with desired topological properties**
  - **Emphasis on the "hard" part of the problem: large sparse DNNs**

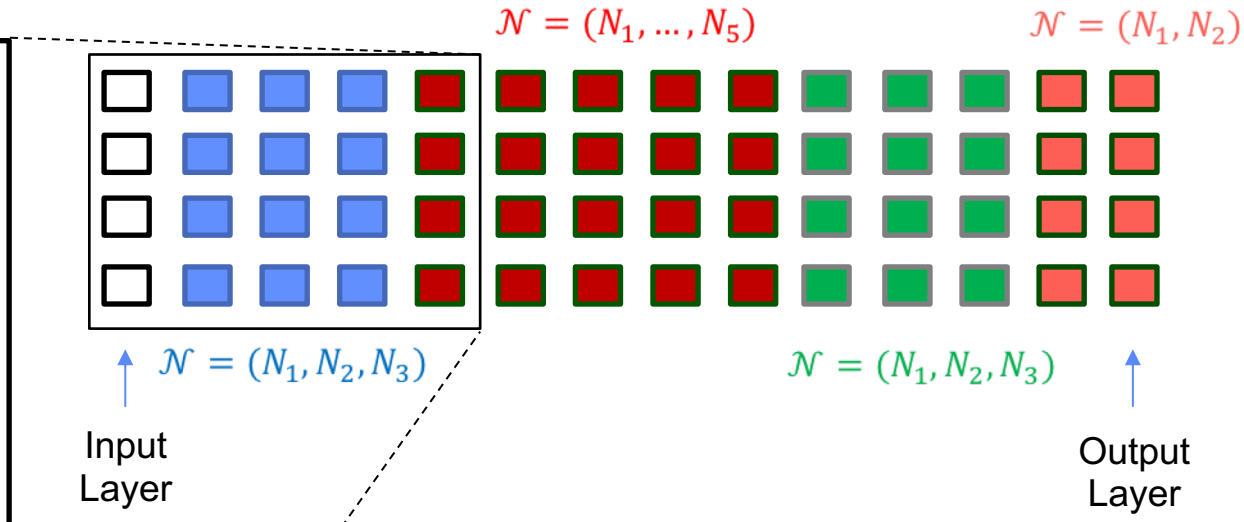
- **Approach: RadiX-Net[1] sparse DNN generator**
  - **Efficiently generates a wide range of pre-determined DNNs**
  - **Specific number of connections per neuron**
  - **Equal number of paths between all inputs/outputs and intermediate layers**

- **DNN generation algorithm**
  - **Mixed radices generate base DNNs of specified connectedness**
  - **Kronecker product base DNNs to create larger/sparser DNNs**
  - **Randomly permute and chain DNNs to create very large DNNs**

[1] *RadiX-Net: Structured Sparse Matrices for Deep Neural Networks*, Robinett & Kepner, IEEE IPDPS GrAPL Workshop 2, 2019

Sparsity, path-connectedness, and symmetry using mixed-radix numeral systems



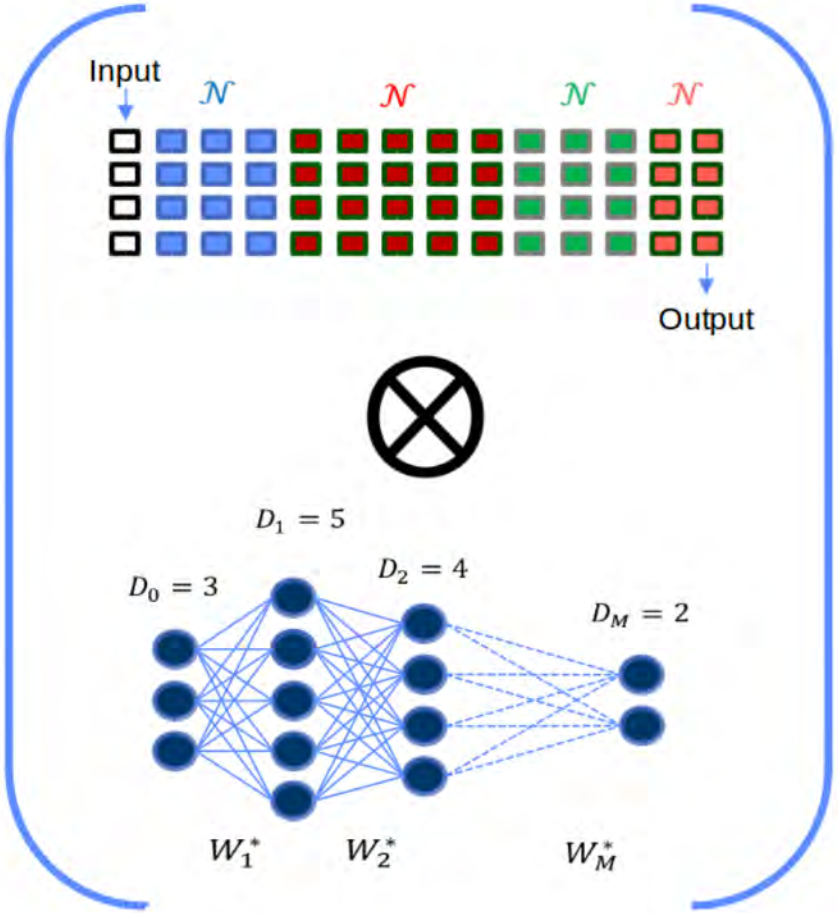


$$N' = \prod_{N \in \mathcal{N}} N = \prod_{N \in \mathcal{N}} N = \prod_{N \in \mathcal{N}} N$$

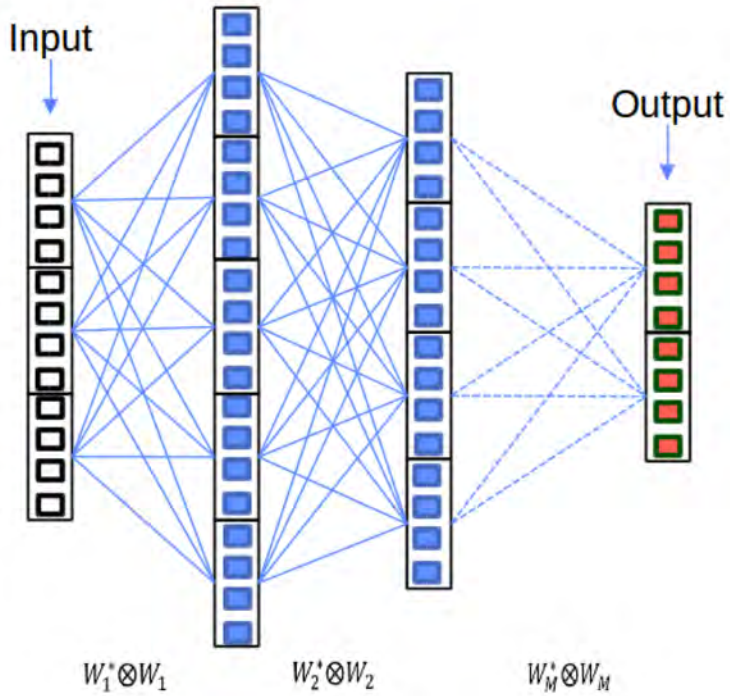
$\prod_{N \in \mathcal{N}} N$ divides N'

$$(W_1, \ldots, W_M) = (W_1, \ldots, W_1, \ldots, W_1, \ldots, W_1, \ldots)$$

RadiX-Net topologies: diversity in layer sizes allowed by Kronecker product



=



Graph Challenge Base DNNs

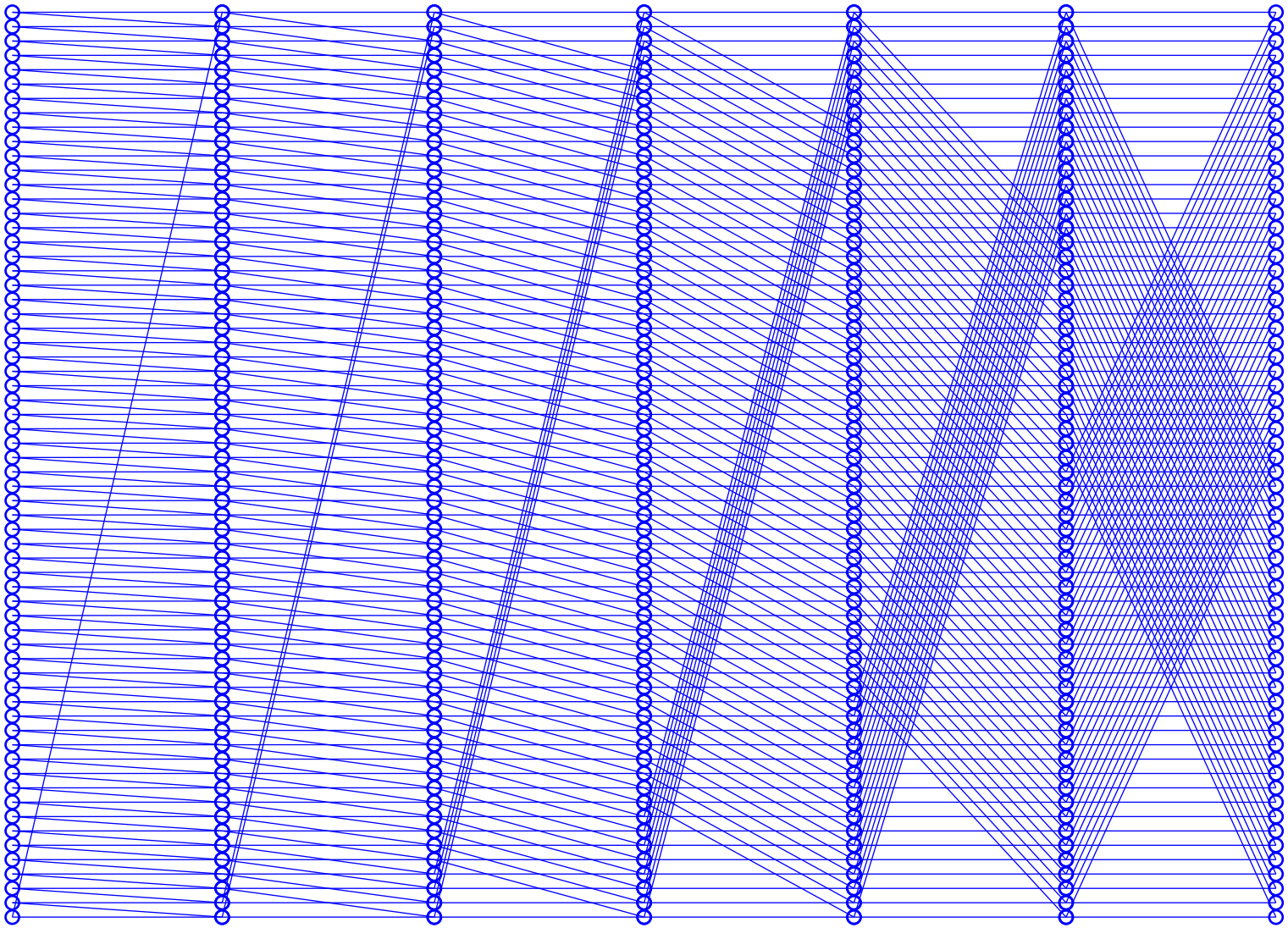
- **Base DNNs (all with** 32 **connections per neuron) generated using RadiX-Net**

- 6 **layers,** 1024 **neurons/layer**
  - **Using radix =** [[2,2,2,2,2,2]] **kron =** [16,16,16,16,16,16,16]
- 8 **layers,** 4096 **neurons/layer**
  - **Using radix =** [[2,2,2,2,2,2,2,2]] **kron =** [16,16,16,16,16,16,16,16,16]
- 10 **layers, neurons/layer =** 16384
  - **Using radix =** [[2,2,2,2,2,2,2,2,2,2]] **kron =** [16,16,16,16,16,16,16,16,16,16,16]
- 12 **layers, neurons/layer =** 65536
  - **Using radix =** [[2,2,2,2,2,2,2,2,2,2,2,2]] **kron =** [16,16,16,16,16,16,16,16,16,16,16,16,16]

- **All non-zero weights set to** 1/16 **(weights into any neuron sum to** 2**)**
  - **Observed to produce layer count independent properties (theory in progress)**

Example RadiX-Net Graph

- 6 layer, 64 neurons per layer, 2 connections per neuron RadiX-Net DNN produced from Radix set [[2,2,2,2,2,2]]
- Kronecker product of this DNN with [16,16,16,16,16,16,16] produces a 6 layer, 1024 neurons per layer DNN with 32 connections per neuron



Graph Challenge Full DNNs

- **Base DNNs are randomly permuted and replicated to generate** 12 **DNNs**
- **Each layer is written out as a** .tsv **file: row, column, weight**

Neurons/Layer

Layers	1024	4096	16384	65536
120	3,932,160	15,728,640	62,914,560	251,658,240
480	15,728,640	62,914,560	251,658,240	1,006,632,960
1920	62,914,560	251,658,240	1,006,632,960	4,026,531,840

Total Edges = 32 **x layers x neurons/layer**

- **Bias values selected to produce reasonable number of output for MNIST input**

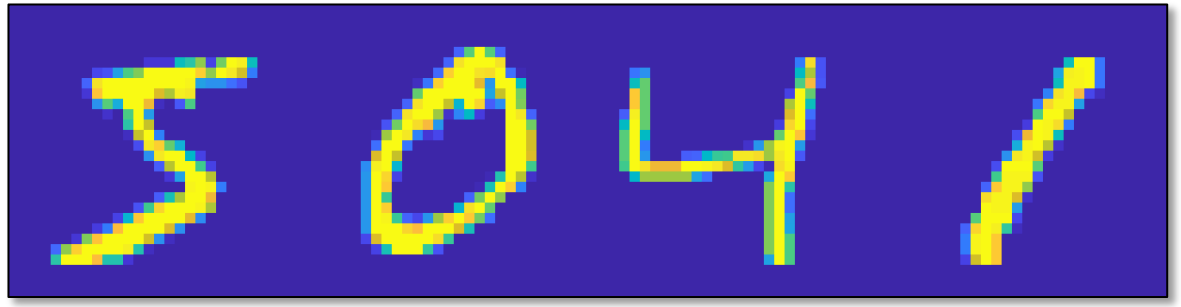
Neurons/Layer

	1024	4096	16384	65536
Bias	-0.3	-0.35	-0.4	-0.45

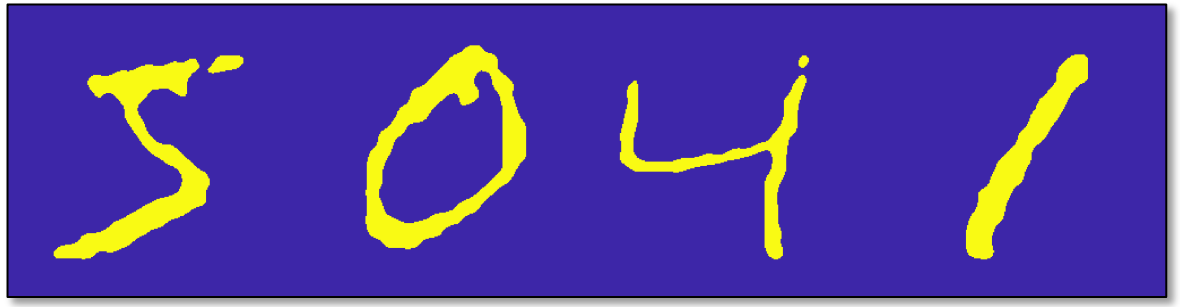
MNIST input data

- **MNIST corpus of** 60,000 28x28 **pixel images of handwritten numbers is a staple of AI**

- **Sparse DNN Graph Challenge uses interpolated sparse versions of this entire corpus as input**
  - **Each image is resized to** 32x32 **(**1024 **neurons),** 64x64 **(**4096 **neurons),** 128x128 **(**16384 **neurons), and** 256x256 **(**65536 **neurons)**
  - **Each sparse resized corpus is save as** .tsv **file: row, column,** 1



MNIST originals 28x28



MNIST sparse 256x256

The MNIST Database, Cortes, LeCun & Burges, http://yann.lecun.com/exdb/mnist/

Outline

- Introduction
- Deep Neural Networks
- Sparse Data Sets
- **Challenge**
- Summary

Challenge Instructions

- Download from GraphChallenge.org: DNNs weight matrices $\mathbf{W}_l$, sparse MNIST input data $\mathbf{Y}_0$, and truth categories
- Load a DNN and its correspond input
- Create and set the appropriate sized bias vectors $\mathbf{b}_l$ from the table
- Evaluate the ReLU DNN equation for all layers (timed)

$$\mathbf{Y}_{l+1} = \mathrm{h}(\mathbf{Y}_l \mathbf{W}_l + \mathbf{b}_l)$$

- Identify the categories (rows) in final matrix with entries > 0 (timed)
- Compare computed categories with truth categories to check correctness
- Compute the rate connection rate for the DNN: (# inputs) x (# connections) / time
- Report time and rate for each DNN measured

Example ReLU Sparse DNN Inference Matlab Code

```
function Y = inferenceReLUvec(W,bias,Y0);
% Performs ReLU inference using input feature vector(s) Y0, DNN weights W, and bias vectors.
  YMAX = 32;                    % Set max value.
  Y = Y0;                       % Initialize feature vectors.

  for i=1:length(W)             % Loop through each weight layer W{i}

     % Propagate through layer.
     % Note: using graph convention of A(i,j) means connection from i *to* j,
     % that requires *left* multiplication of feature *row* vectors.
     Z = Y*W{i};
     b = bias{i};

     Y = Z + (double(logical(Z)) .* b);     % Apply bias to non-zero entries.
     Y(Y < 0) = 0;                          % Threshold negative values.
     Y(Y > YMAX) = YMAX;                    % Threshold maximum values.
  end
return
end
```

Recommendations

Avoid

- Exploiting repetitive structure of weight matrices, weight values, bias values
- Exploiting layer independence of results
- Using optimizations that would not work on real-world data

Do

- Use an implementation that could work on real-world data
- Create compressed binary versions of inputs to accelerate reading the data
- Split inputs and run in data parallel mode to achieve higher performance
  - Requires replicating weight matrices on every processor (can require a lot of memory)
- Split up layers and run in pipeline parallel mode to achieve higher performance
  - Saves memory, but requires communicating results after each group of layers
- Use other reasonable optimizations that would work on real-world data

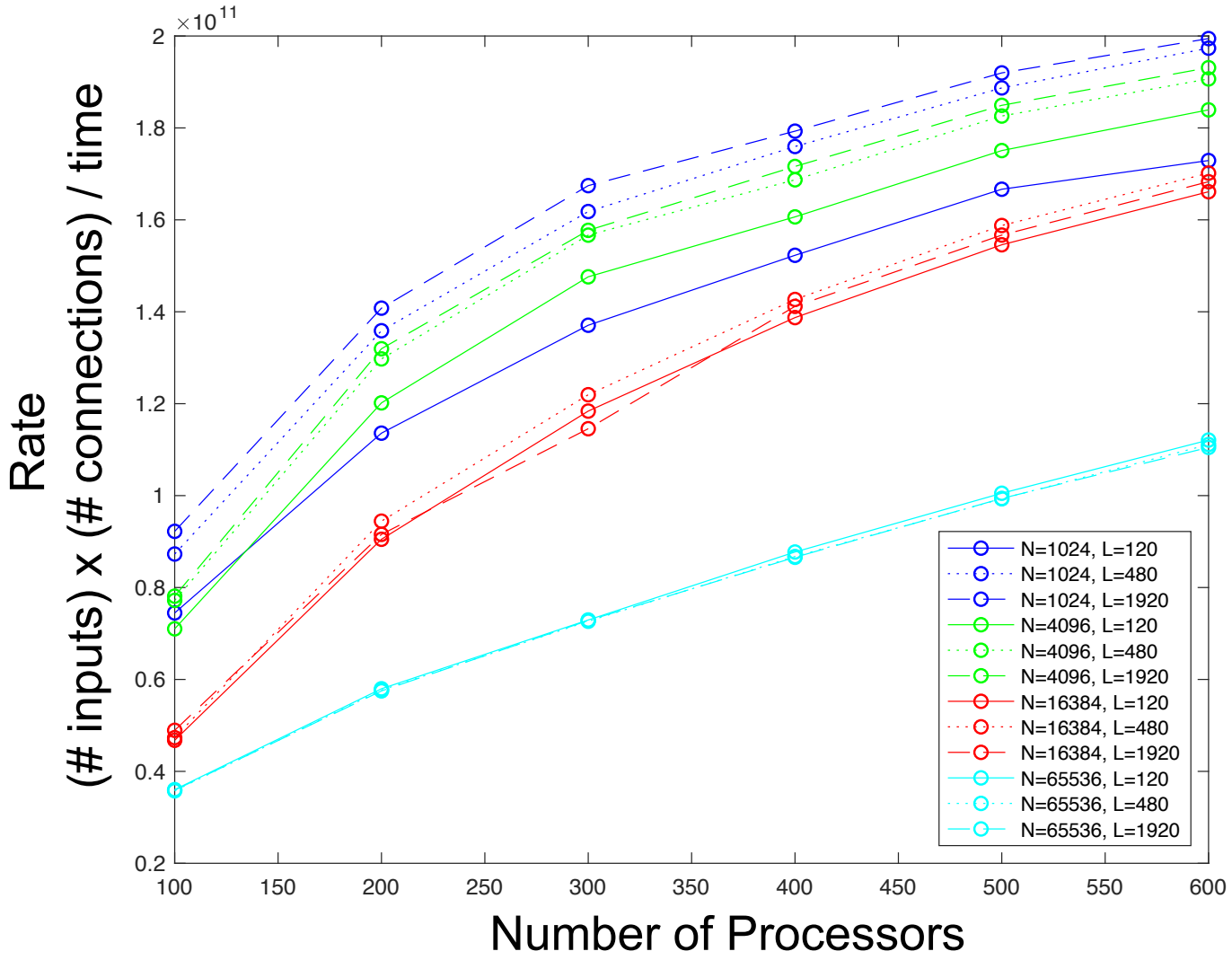
Example Serial Performance

Neurons per Layer	Layers	Connections (edges)	Inference Time (second)	Inference Rate (input[1] x edge /sec)
1024	120	3,932,160	626	376×10^6
1024	480	15,728,640	2440	386×10^6
1024	1920	62,914,560	9760	386×10^6
4096	120	15,728,640	2446	385×10^6
4096	480	62,914,560	10229	369×10^6
4096	1920	251,658,240	40245	375×10^6
16384	120	62,914,560	10956	344×10^6
16384	480	251,658,240	45268	333×10^6
16384	1920	1,006,632,960	179401	336×10^6
65536	120	251,658,240	45813	329×10^6
65536	480	1,006,632,960	202393	298×10^6
65536	1920	4,026,531,840		

[1]Number of MNIST inputs = 60,000

Example Data Parallel Performance

- Serial reference implementation run in parallel by splitting feature vectors
- DNN replicated
- Run on MIT SuperCloud
- Intel KNL processors with 192 GB RAM

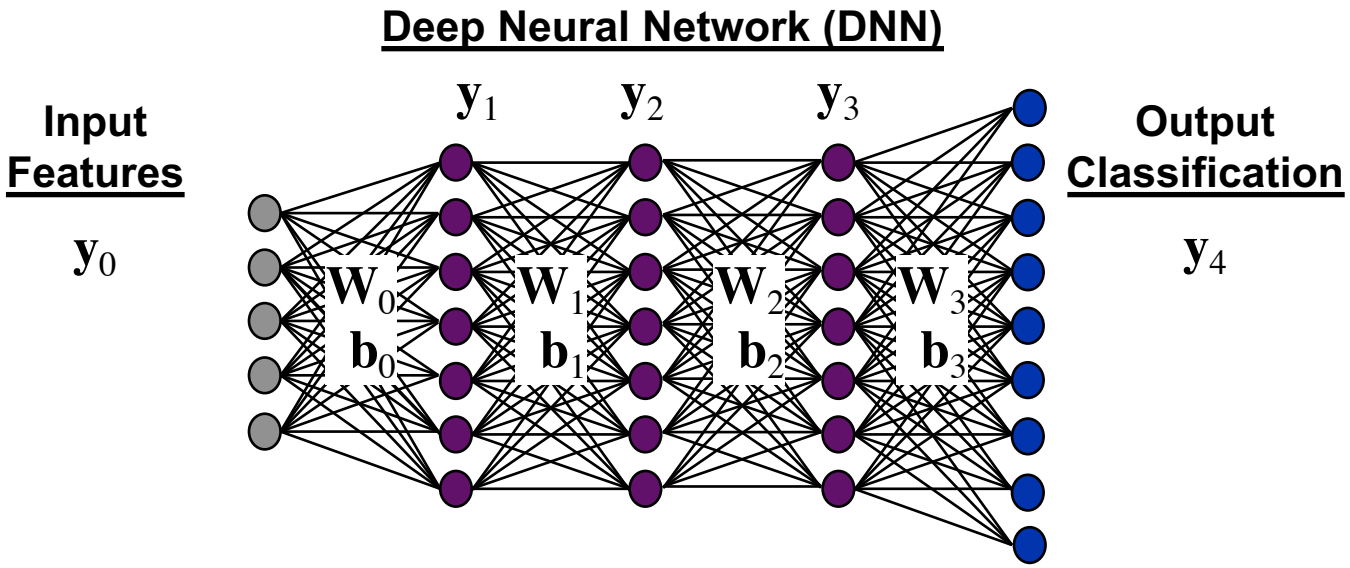


Summary

- **MIT/IEEE/Amazon Graph Challenge has provides a valuable venue for innovators in graph and sparse network analysis system**

- **Sparse DNN Challenge seeks stimulate new work in the area of sparse AI**

- **Proposed challenge provides a wide range of simulated sparse DNNs to evaluate**



Acknowledgements

- David Martinez
- Jeremy Kepner
- Albert Reuther
- Vijay Gadepally
- Michael Jones
- Lauren Milechin
- Bill Arcand
- Bill Bergeron
- David Bestor
- Chansup Byun
- Matt Hubbell
- Pete Michaleas
- Julie Mullen
- Andy Prout
- Tony Rosa
- Jason Williams
- Ryan Whelan
- Charles Yee
- Paul Monticciolo
- Sanjeev Mohindra
- Steven Smith
- Edward Kao
- Michael Hurley
- William Song