Training Behavior of Sparse Neural Network Topologies

Simon Alford, Ryan Robinett, Lauren Milechin, Jeremy Kepner

Massachusetts
Institute of
Technology

Outline

- **Introduction**
- Approach
- Results
- Interpretation and Summary

Limiting factors confronting Deep Learning

- **Quality and quantity of data**

Limiting factors confronting Deep Learning

- **Quality and quantity of data**
- **Techniques, network design, etc.**

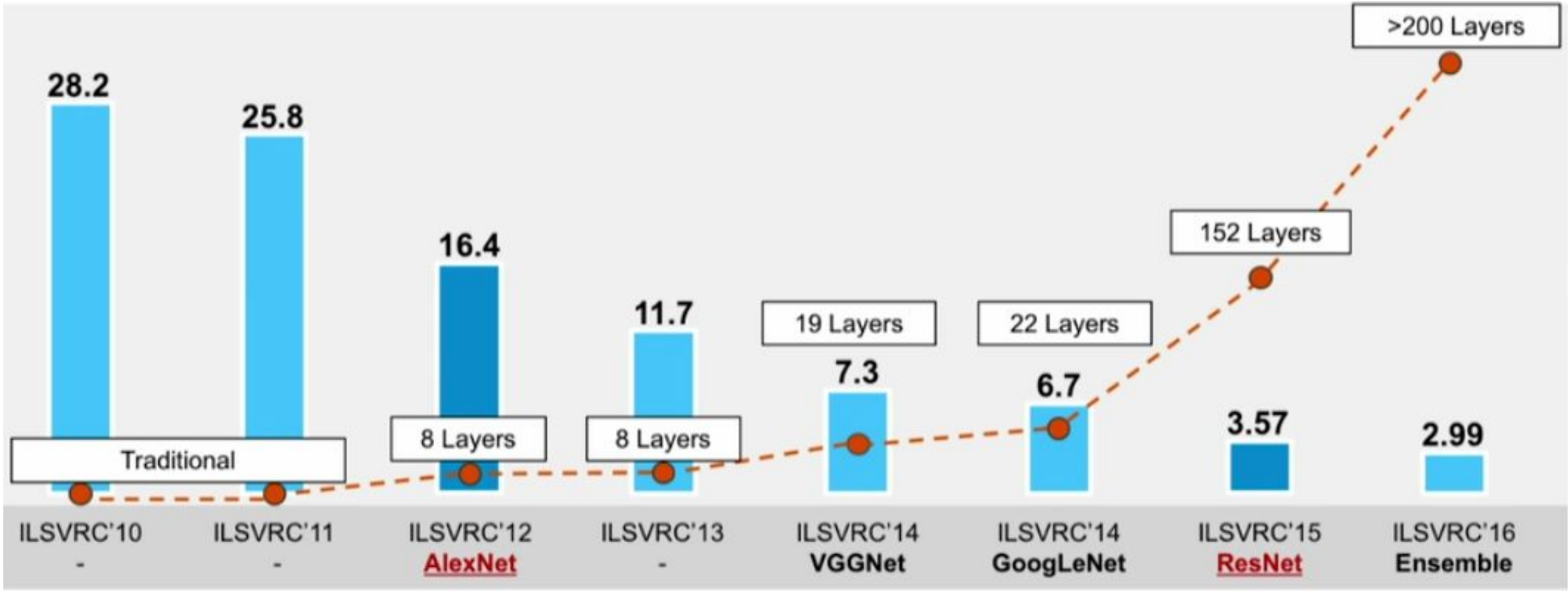
Limiting factors confronting Deep Learning

- Quality and quantity of data
- Techniques, network design, etc.
- Computational demands vs resources available

Limiting factors confronting Deep Learning

- Quality and quantity of data
- Techniques, network design, etc.
- Computational demands vs resources available

Progress in Computer Vision



http://sglml.azurewebsites.net/2017/09/12/convolutional-neural-network/

Progress in Natural Language Processing

The estimated costs of training a model

	Date of original paper	Energy consumption (kWh)	Carbon footprint (lbs of CO2e)	Cloud compute cost (USD)
Transformer (65M parameters)	Jun, 2017	27	26	$41-$140
Transformer (213M parameters)	Jun, 2017	201	192	$289-$981
ELMo	Feb, 2018	275	262	$433-$1,472
BERT (110M parameters)	Oct, 2018	1,507	1,438	$3,751-$12,571
Transformer (213M parameters) w/ neural architecture search	Jan, 2019	656,347	626,155	$942,973-$3,201,722
GPT-2	Feb, 2019	-	-	$12,902-$43,008

https://www.technologyreview.com/s/613630/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/

Progress in Reinforcement learning

AlphaGo Zero

- **29 million games over 40 days of training**
- **Estimated compute cost: $35,354,222**
- **Estimated > 6000 TPU's**

- "[This] is an unattainable level of compute for the majority of the research community. When combined with the unavailability of code and models, the result is that the approach is very difficult, if not impossible, to reproduce, study, improve upon, and extend"

Facebook, on replicating AlphaGo Zero results

https://www.yuzeh.com/data/agz-cost.html

Motivation

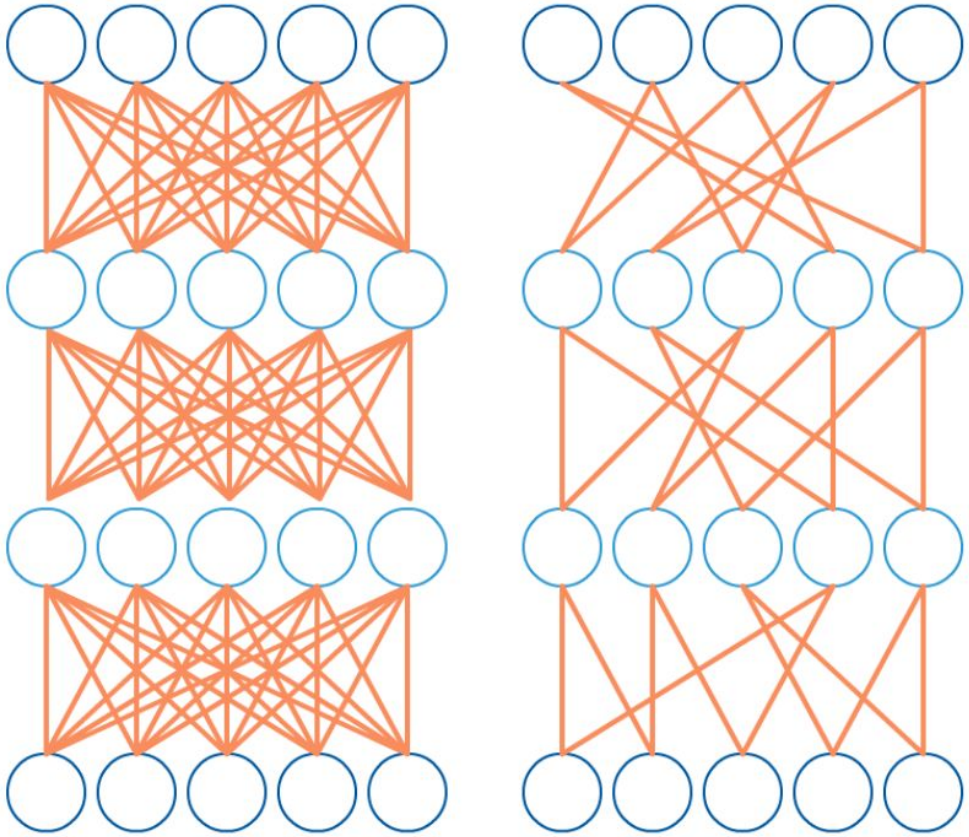
Ongoing Challenge: How can we train **larger**, **more powerful** networks with **fewer computational resources**?

Motivation

Ongoing Challenge: How can we train **larger**, **more powerful** networks with **fewer computational resources**?

Idea: “Go sparse"

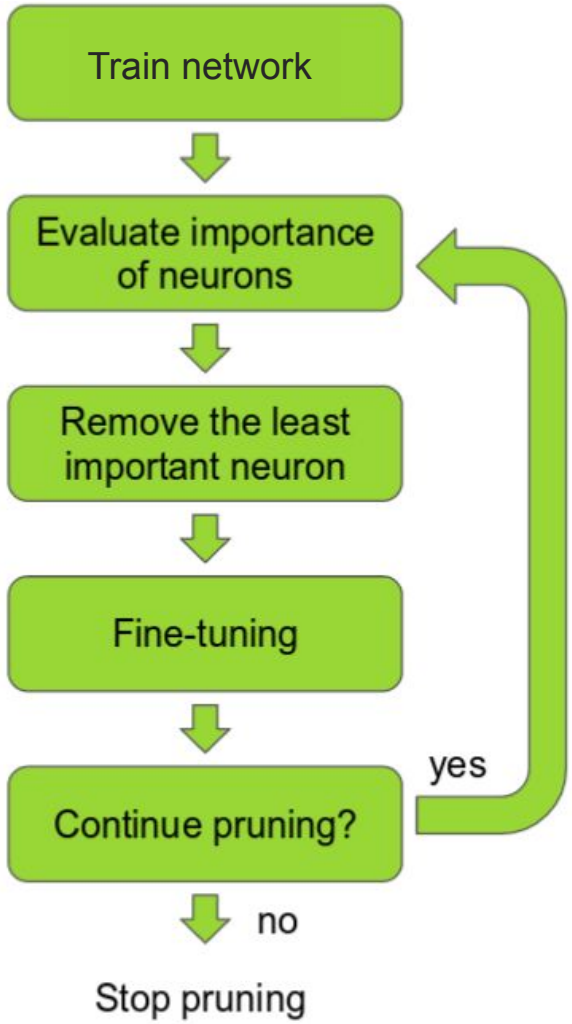
- Leverage preexisting optimizations using sparse matrices
- Scale with number of connections instead of number of neurons
- There may exist sparse network topologies which train as well or better than dense



Fully connected Sparse

Previous Work on Sparse Neural Networks

- *Optimal Brain Damage*[1]
  - Prunes weights based on second-derivative information
- *Learning both Weights and Connections for Efficient Neural Networks*[2]
  - Iteratively prunes and retrains network
- Other methods: low-rank approximation[3], variational dropout[4], . . .

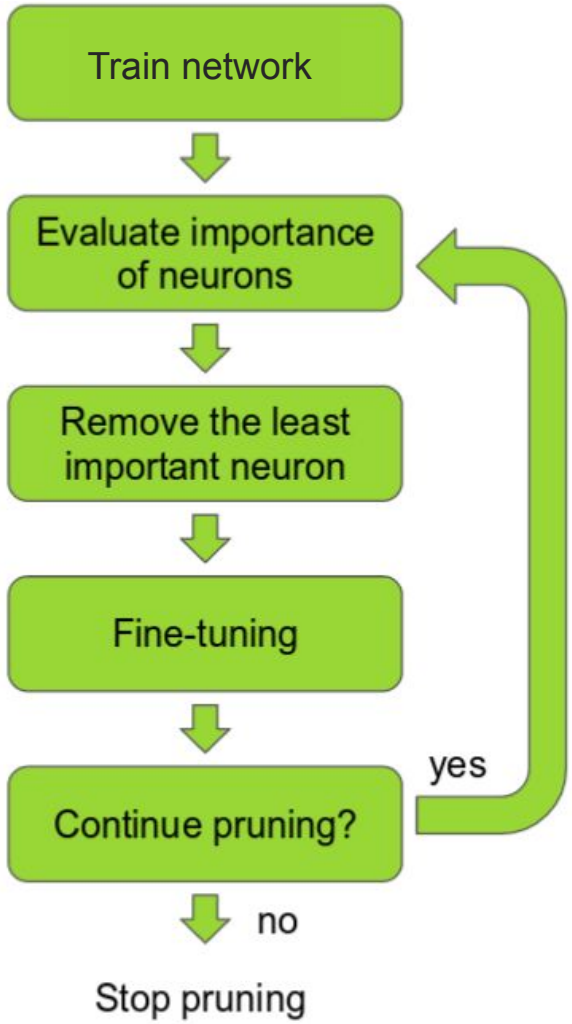


[1] LeCun et. al, *Optimal brain damage*. In NIPS, 1989.
[2] Han et. al, *Learning both weights and connections for efficient neural networks*. In NIPS, 2015
[3] Sainath et. al, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*. in ICASSP, 2013
[4] Molchanov et. al, *Variational Dropout sparsifies deep neural networks*. 2017

Previous Work

- *Optimal Brain Damage*[1]
  - Prunes weights based on second-derivative information
- *Learning both Weights and Connections for Efficient Neural Networks*[2]
  - Iteratively prunes and retrains network
- Other methods: low-rank approximation[3], variational dropout[4], . . .

… Problem?



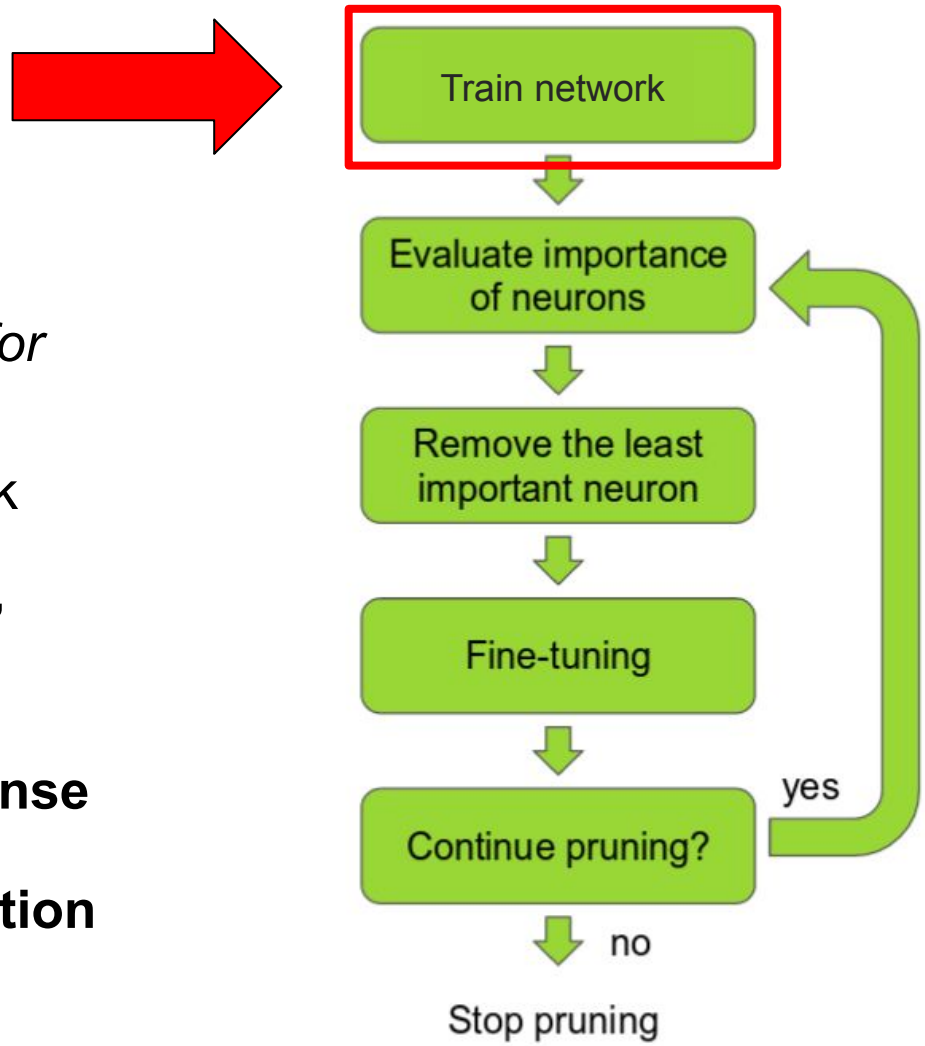
[1] LeCun et. al, *Optimal brain damage*. In NIPS, 1989.
[2] Han et. al, *Learning both weights and connections for efficient neural networks*. In NIPS, 2015
[3] Sainath et. al, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*. in ICASSP, 2013
[4] Molchanov et. al, *Variational Dropout sparsifies deep neural networks*. 2017

Previous Work

- *Optimal Brain Damage*[1]
  - Prunes weights based on second-derivative information
- *Learning both Weights and Connections for Efficient Neural Networks*[2]
  - Iteratively prunes and retrains network
- Other methods: low-rank approximation[3], variational dropout[4], . . .

… Problem? Start by training dense

- **Can't rely on sparsity to yield computation savings for training**



[1] LeCun et. al, *Optimal brain damage*. In NIPS, 1989.
[2] Han et. al, *Learning both weights and connections for efficient neural networks*. In NIPS, 2015
[3] Sainath et. al, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*. in ICASSP, 2013
[4] Molchanov et. al, *Variational Dropout sparsifies deep neural networks*. 2017

Previous Work

- **Much research has been done pruning pretrained networks to become sparse, for purposes of model compression, deployment on embedded devices, etc.**
- **Little research has been done training from scratch on sparse network structures**

- One example: *Deep Expander Networks*[1]
  - Replace connections with random and explicit expander graphs to create trainable sparse networks with strong connectivity properties

[1] Prabhu et. al, *Deep Expander Networks: Efficient Deep Networks from Graph Theory*

Previous Work

- **Much research has been done pruning pretrained networks to become sparse, for purposes of model compression, deployment on embedded devices, etc.**
- **Little research has been done training from scratch on sparse network structures**

- One example: *Deep Expander Networks*[1]
  - Replace connections with random and explicit expander graphs to create trainable sparse networks with strong connectivity properties

Our contribution: Development and evaluation of pruning-based and structurally-sparse trainable networks

[1] LeCun et. al, *Optimal brain damage*. In NIPS, 1989.
[2] Han et. al, *Learning both weights and connections for efficient neural networks*. In NIPS, 2015
[3] Prabhu et. al, *Deep Expander Networks: Efficient Deep Networks from Graph Theory*

Overview of Approach

Techniques

First approach: Pruning

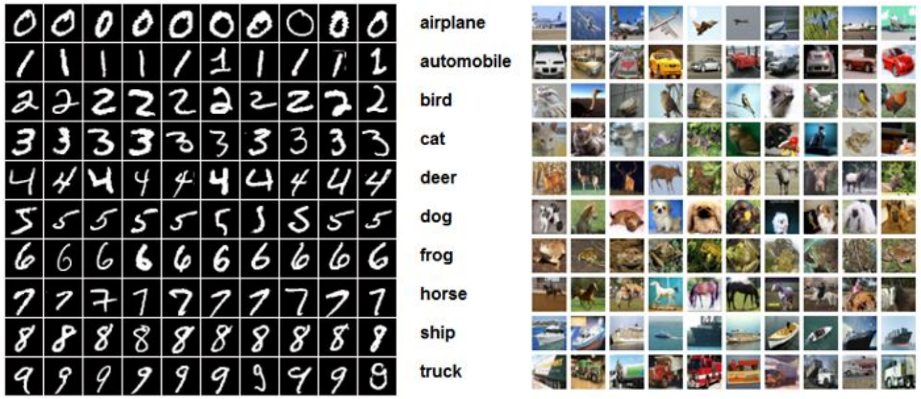
- Prune the network during/after training to learn a sparse network structure
- Initialize network with pruned network as structure and train

Second approach: RadiX-Nets

- Ryan Robinett's RadiX-Nets provide theoretical guarantees of sparsity, connectivity properties
- Train RadiX-Nets and compare to dense training

Implementation

- Experiments done using TensorFlow
- Used Lenet-5 and Lenet 300-100 networks
- Tested on MNIST, CIFAR-10 datasets



MNIST **CIFAR-10**

Outline

- Introduction
- **Approach**
- Results
- Interpretation and Summary

Designing a trainable sparse network

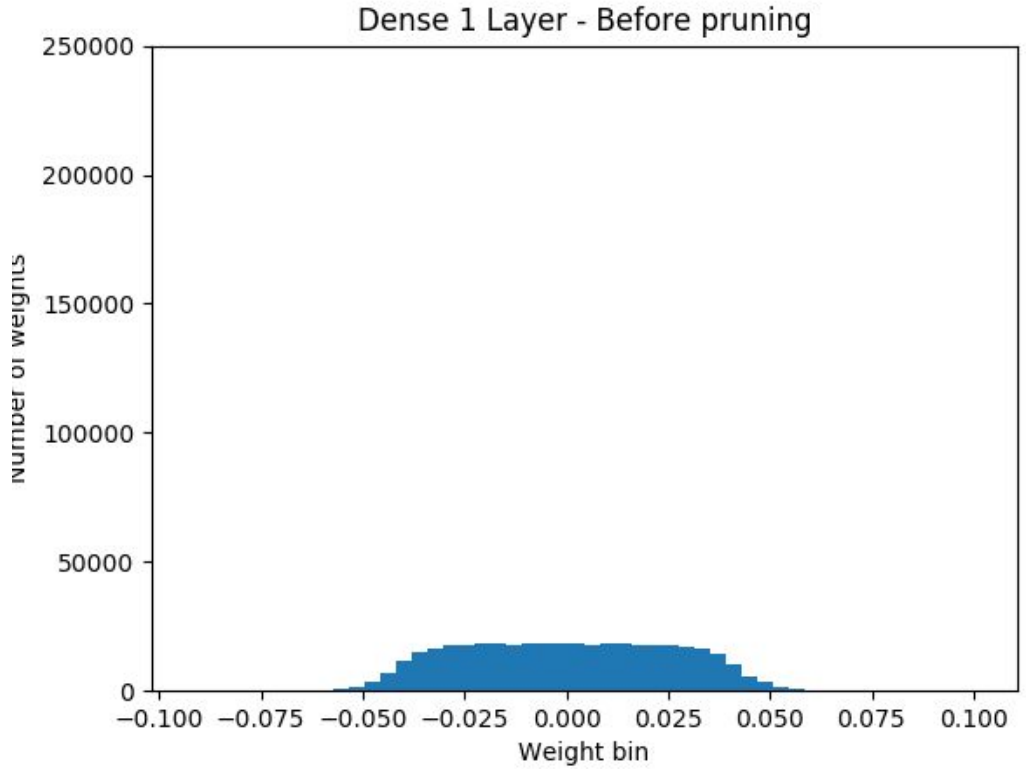
Pruning

- Train a dense network, then prune connections to obtain sparse network
- Important connections, structure is preserved
- Two pruning methods: one-time and iterative pruning

Designing a trainable sparse network

One-time Pruning

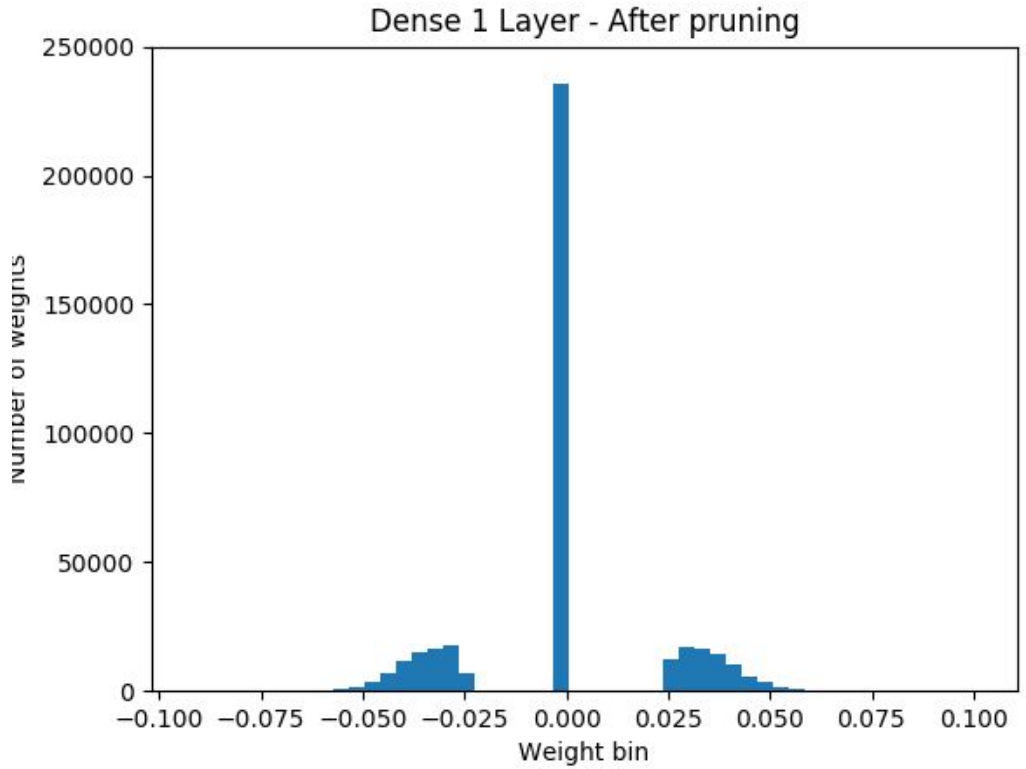
- Prune weights below threshold: `weights[np.abs(weights) < threshold] = 0`



Designing a trainable sparse network

One-time Pruning

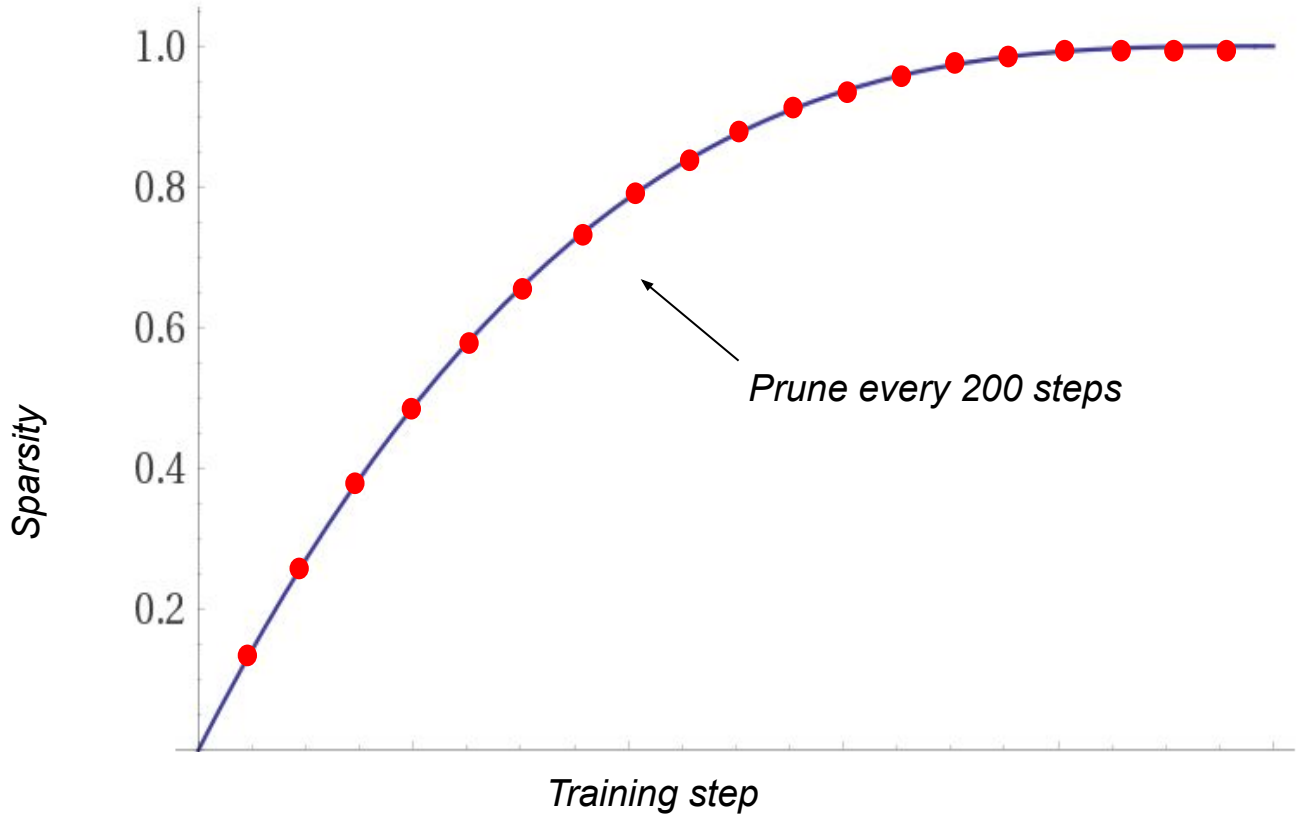
- Prune weights below threshold: `weights[np.abs(weights) < threshold] = 0`



Designing a trainable sparse network

Iterative Pruning

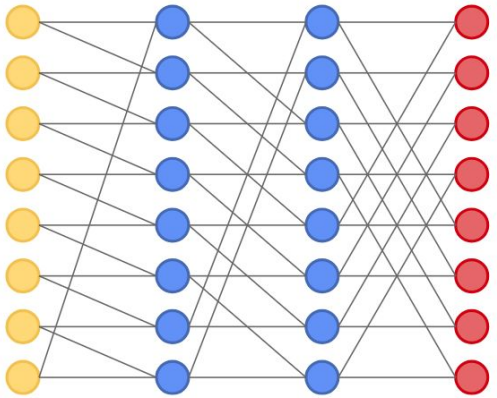
- Iteratively cycle between pruning neurons below threshold and retraining remaining neurons
- Modified technique: prune network to match monotonically increasing sparsity function $s(t)$
- Able to achieve much higher sparsity than one-time pruning without loss in accuracy (>95% vs 50%)



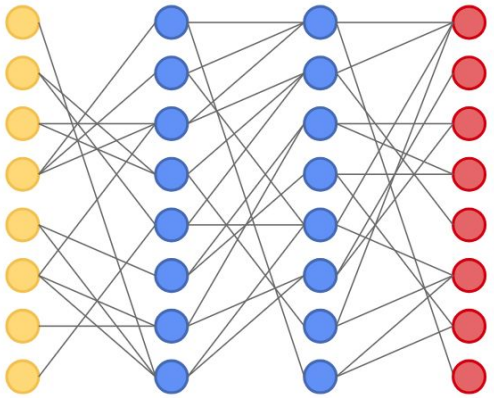
Generating a sparse network to train on

Second method: RadiX-Nets

- Building off Prabhu et. al's *Deep Expander Networks*
- Uses mixed radix systems to create sparse networks with provable connectivity, sparsity, and symmetry properties
- Ryan Robinett created RadiX-Nets as an improvement over expander networks
- Can be designed to fit different network sizes, depths, and sparsity levels while retaining properties

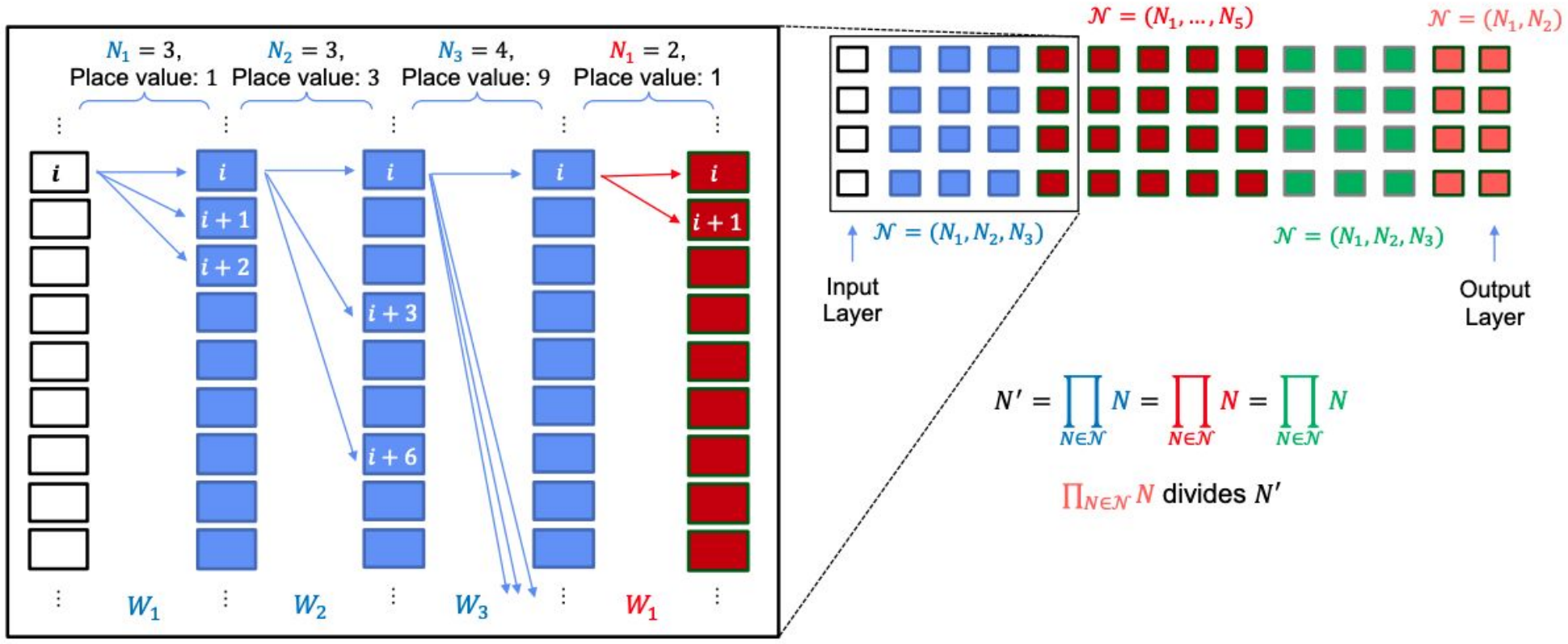


Above: A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity. Below: The random equivalent



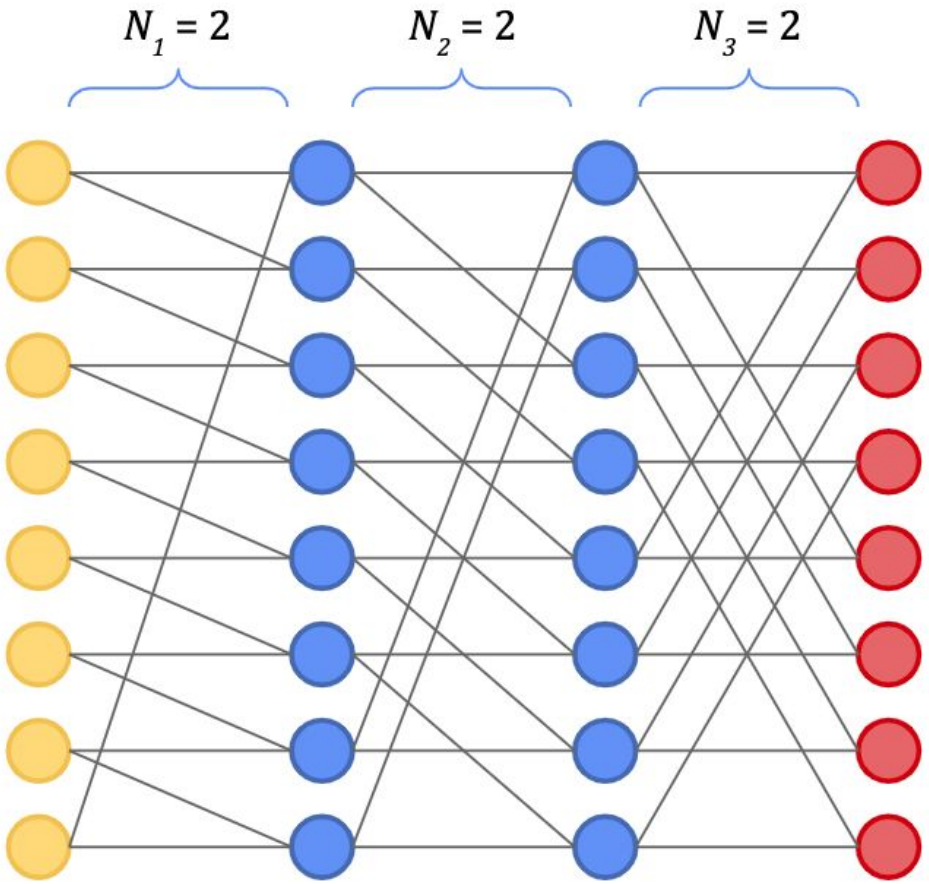
RadiX-Nets

- **Given set of radices, connect neurons in adjacent layers at regular intervals**



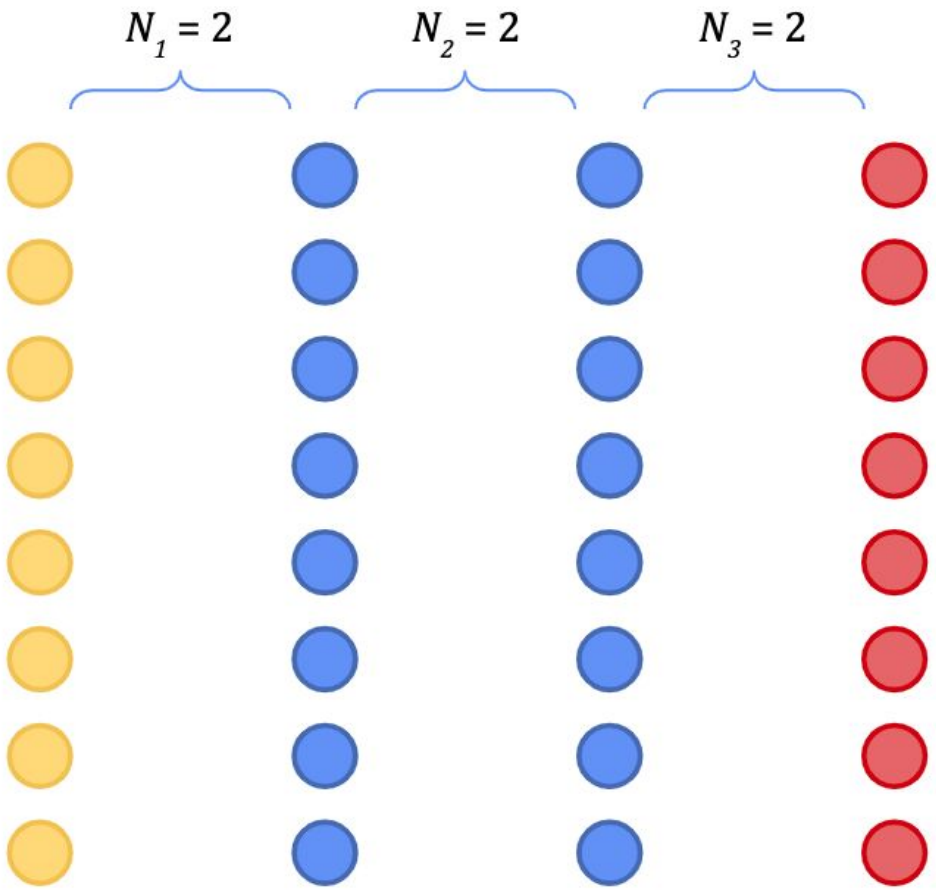
Robinett and Kepner, *Sparse, symmetric neural network topologies for sparse training.* In MIT URTC, 2018.

RadiX-Nets



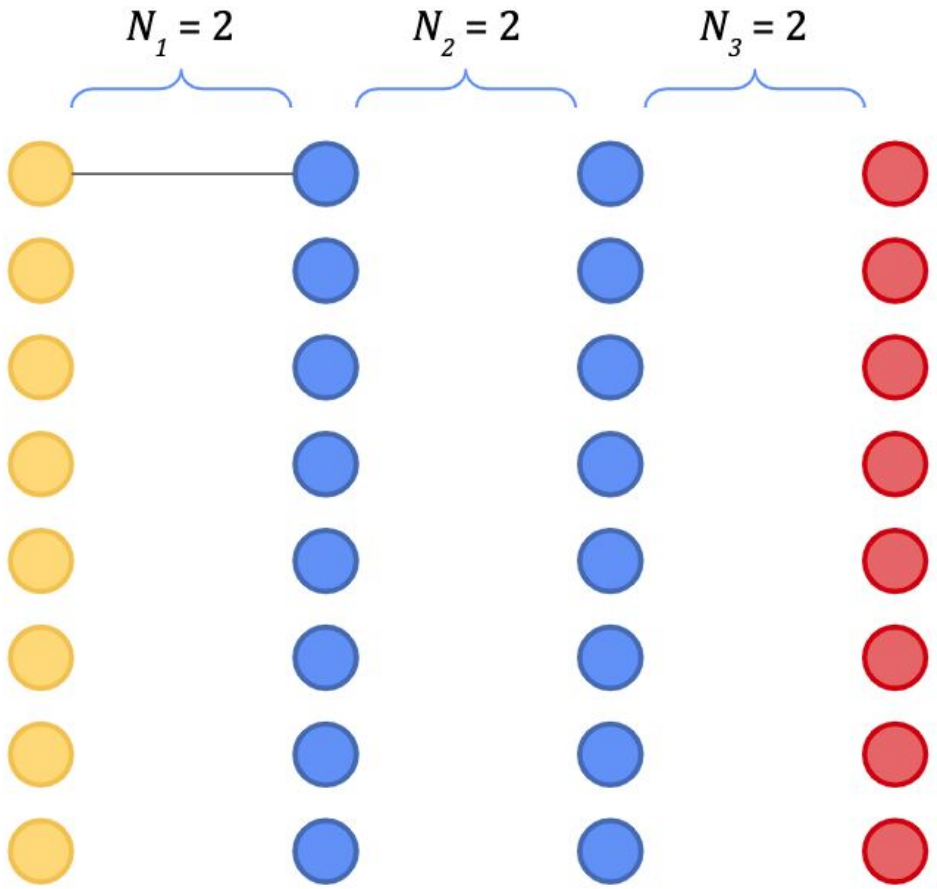
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



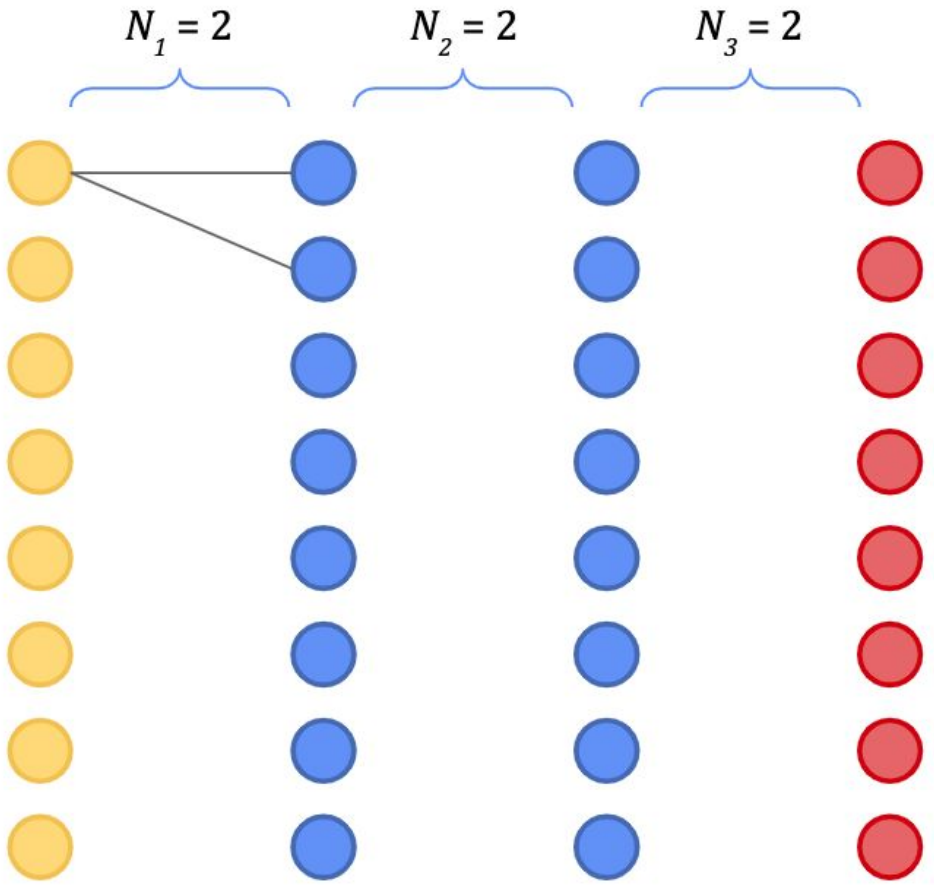
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



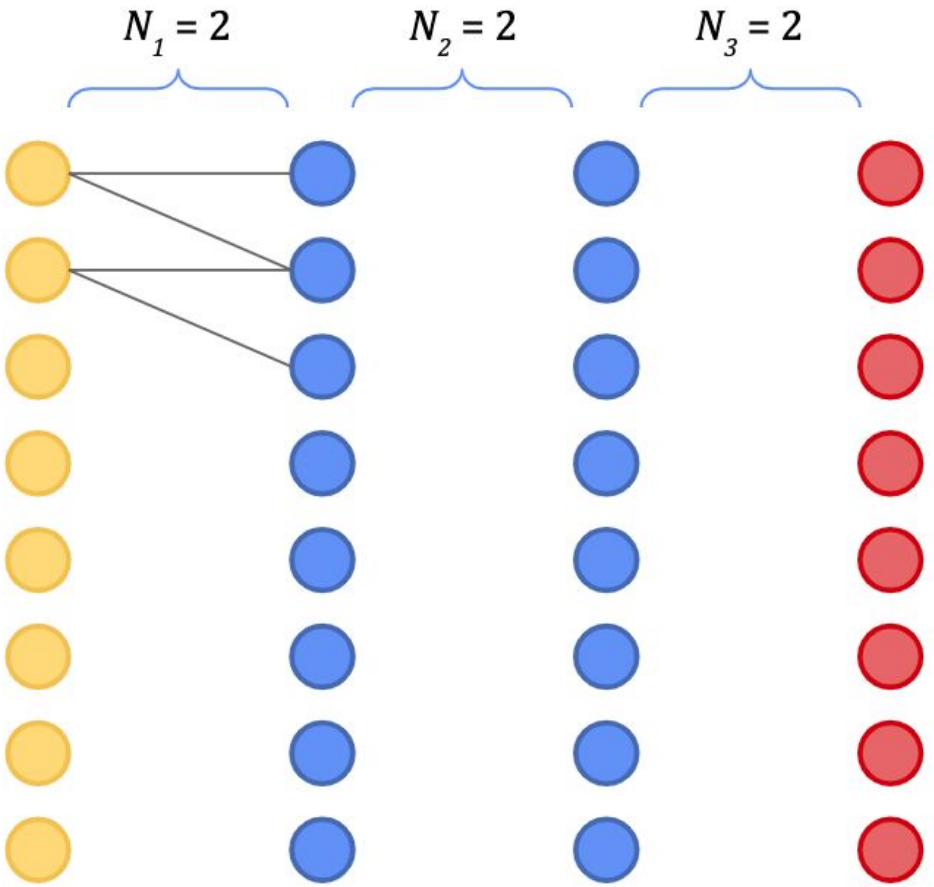
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



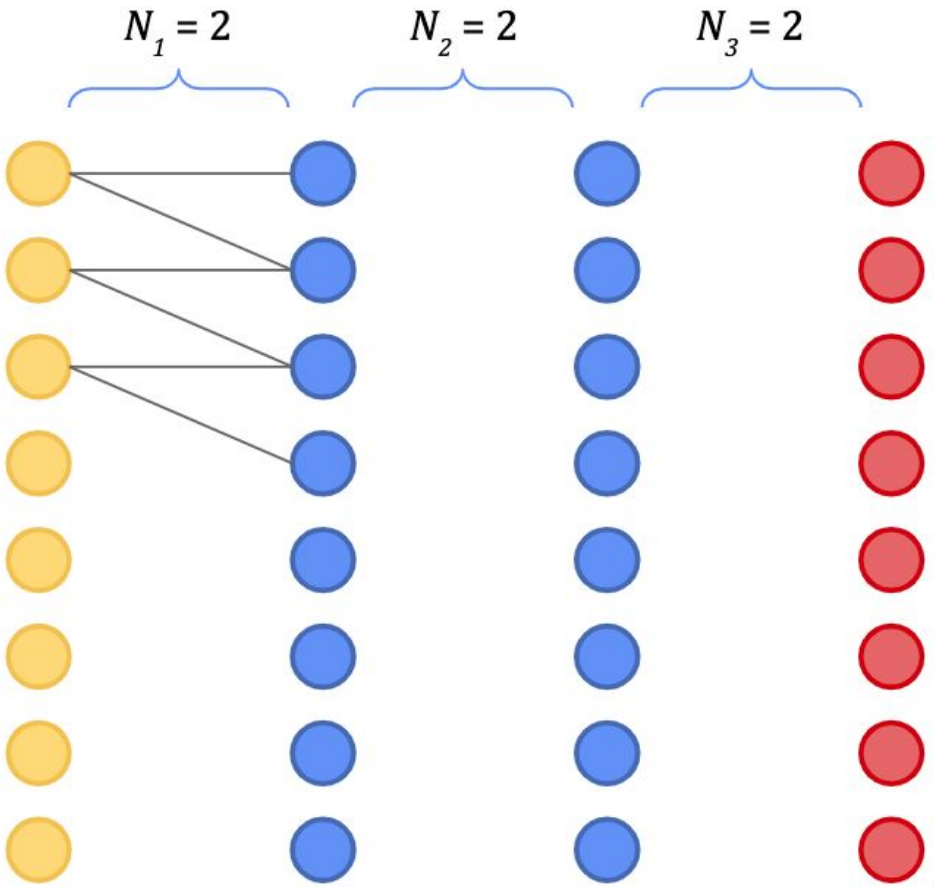
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



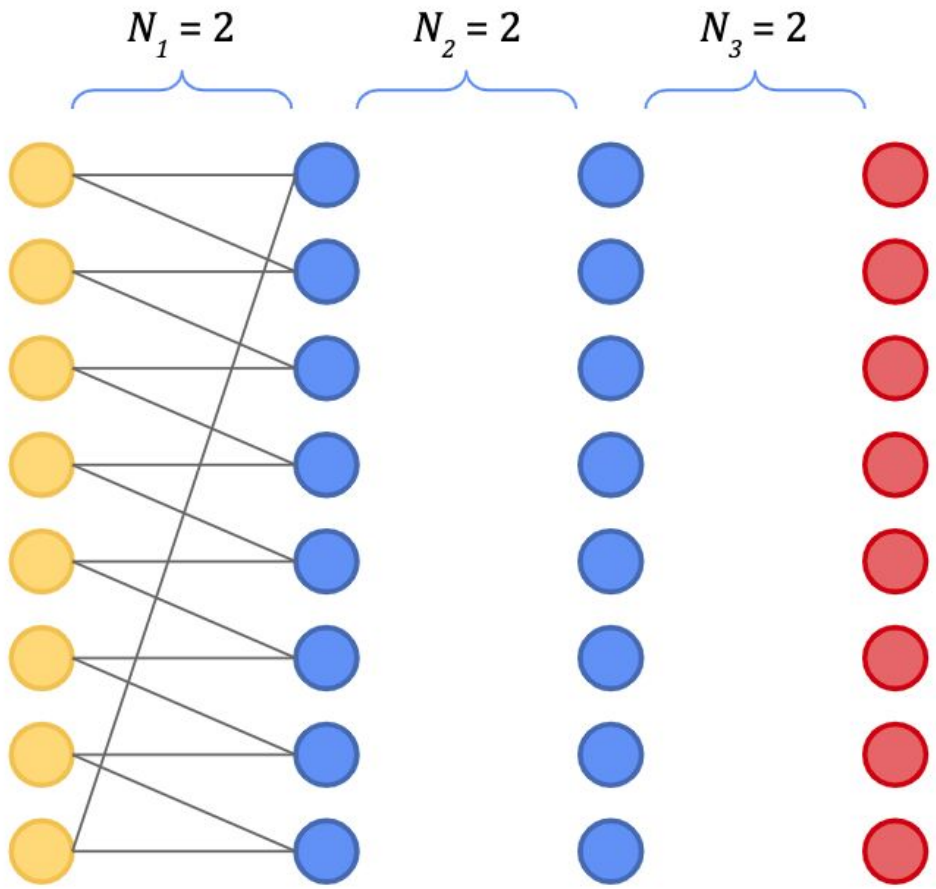
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



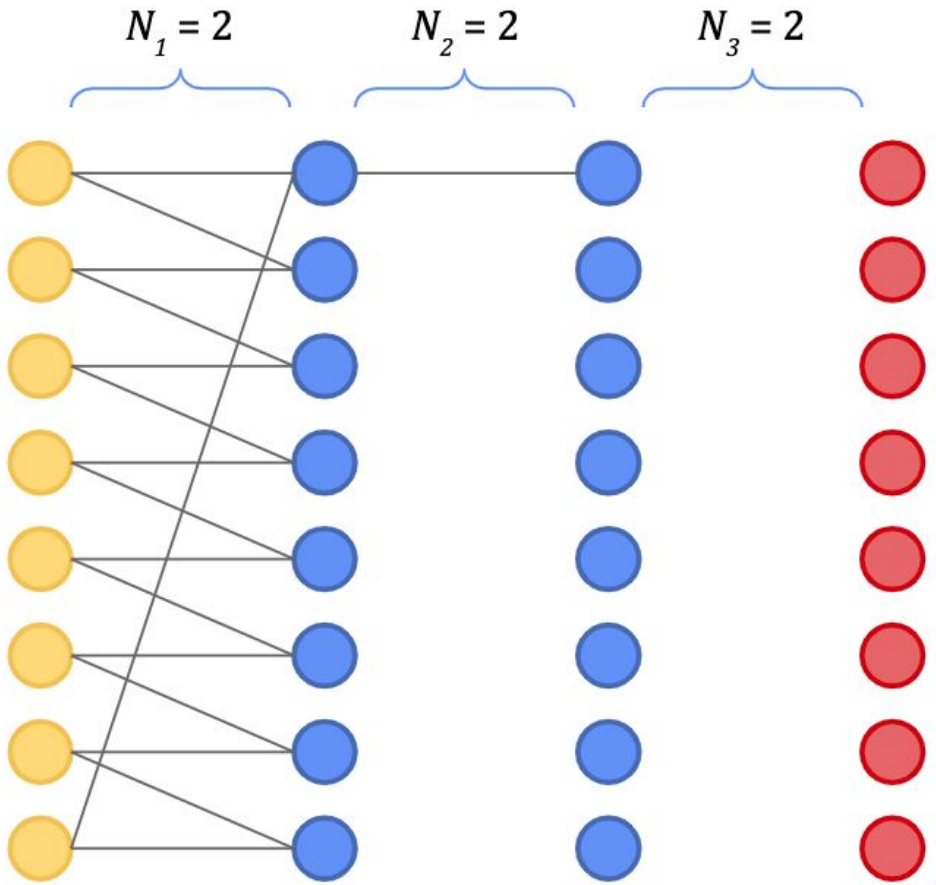
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



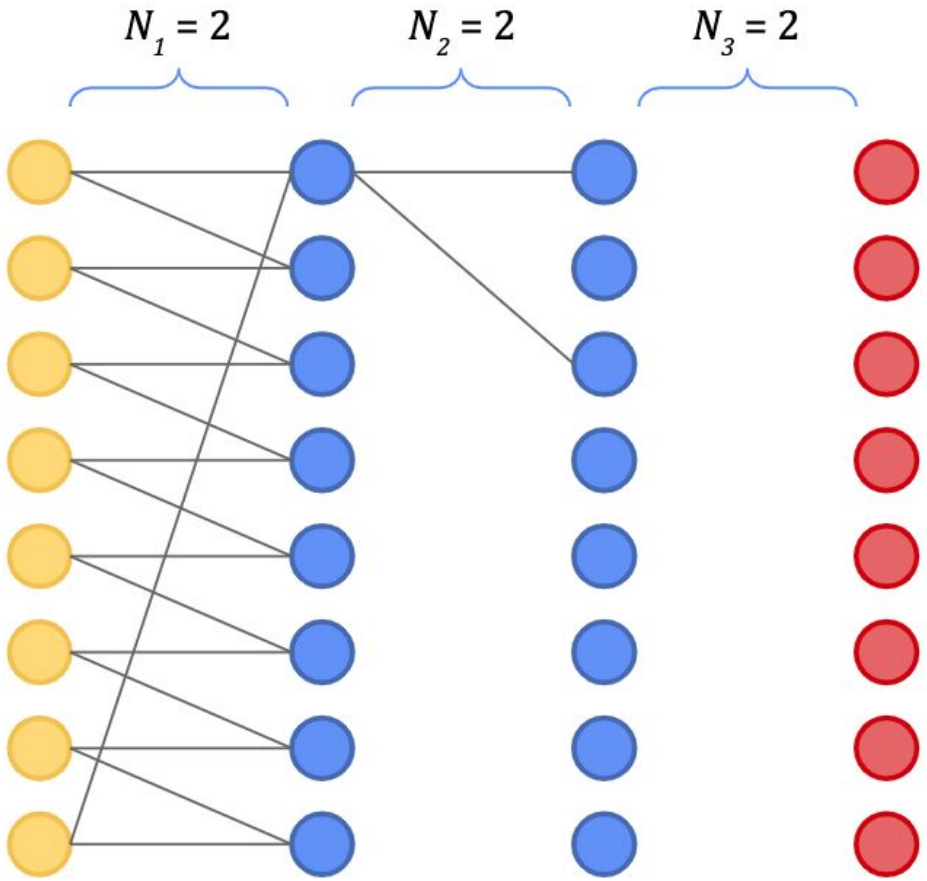
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



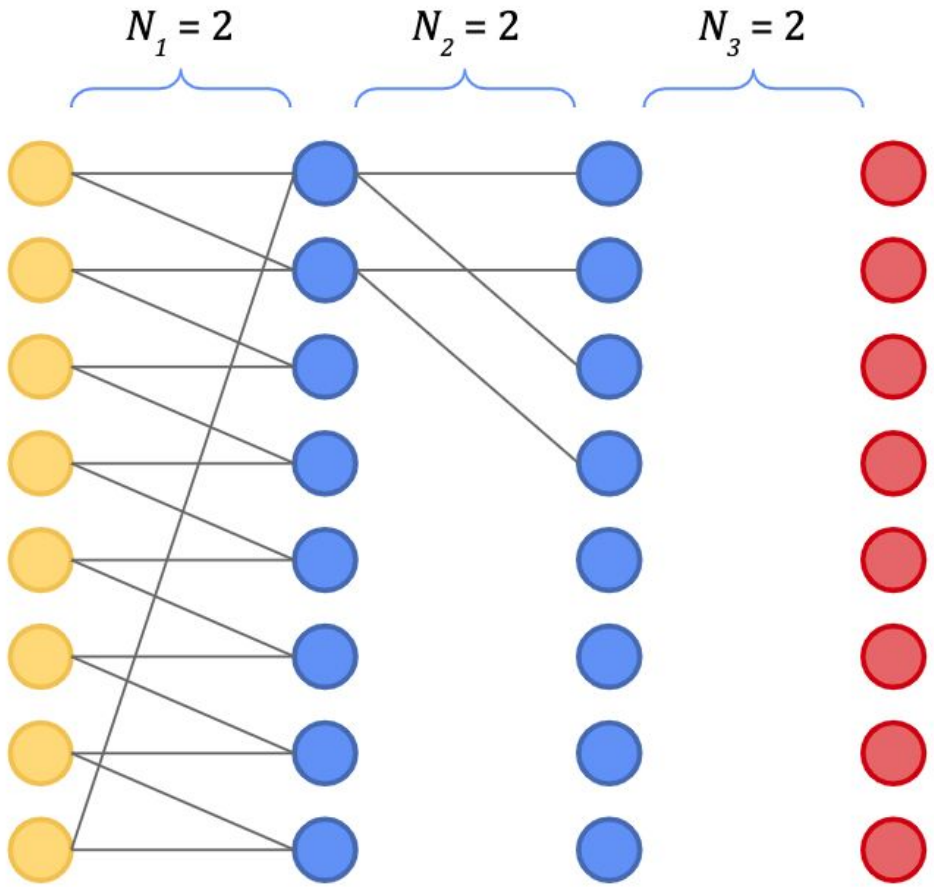
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



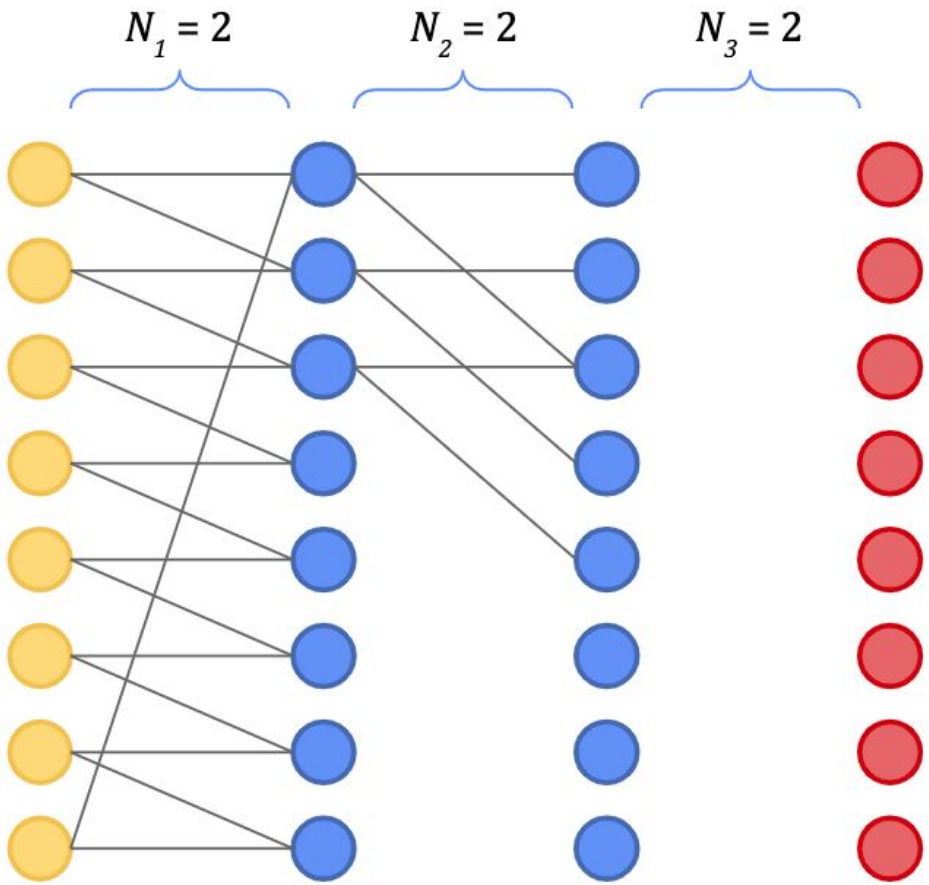
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



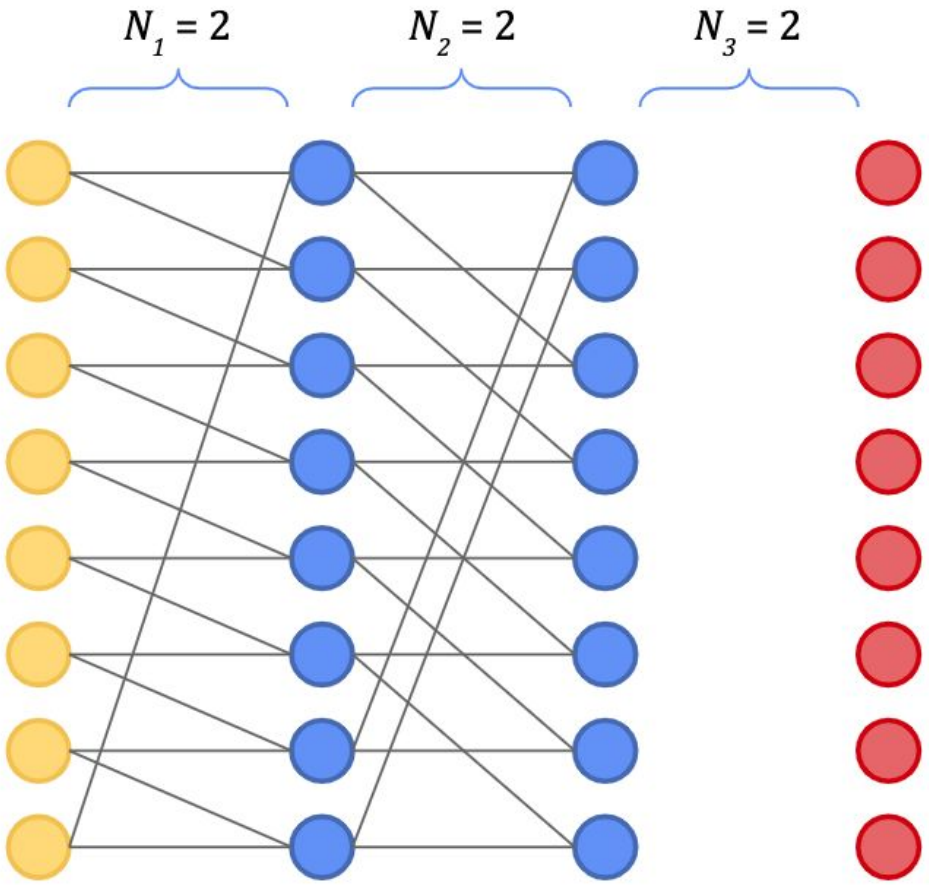
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



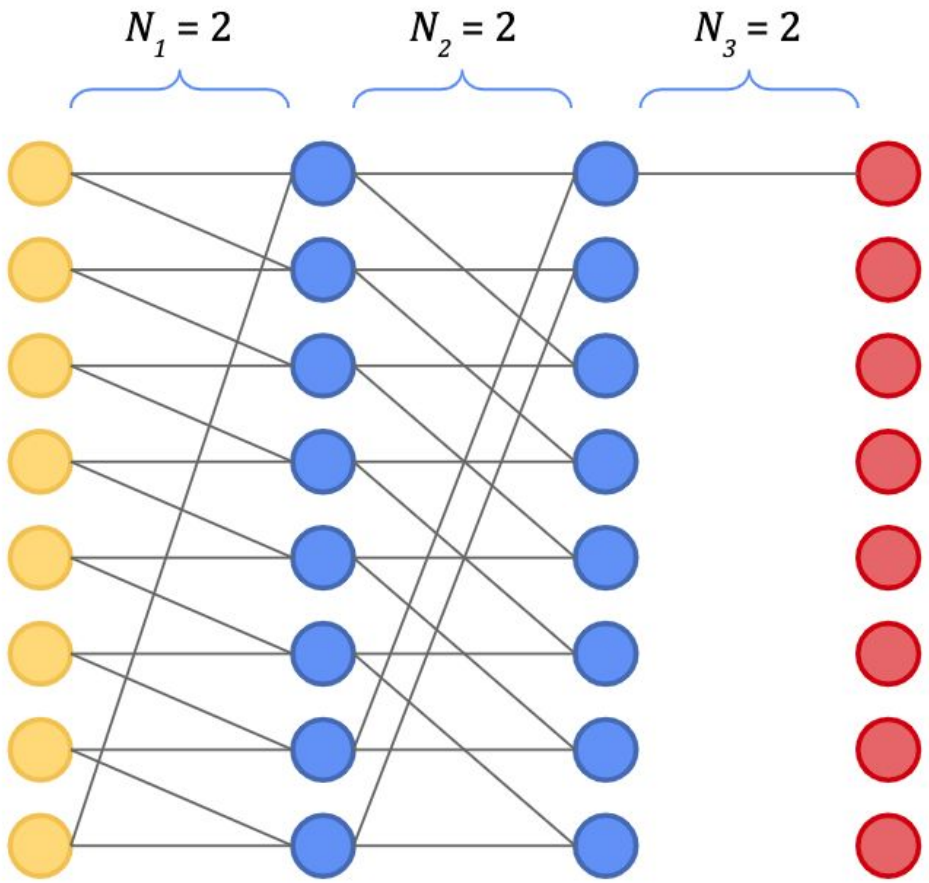
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



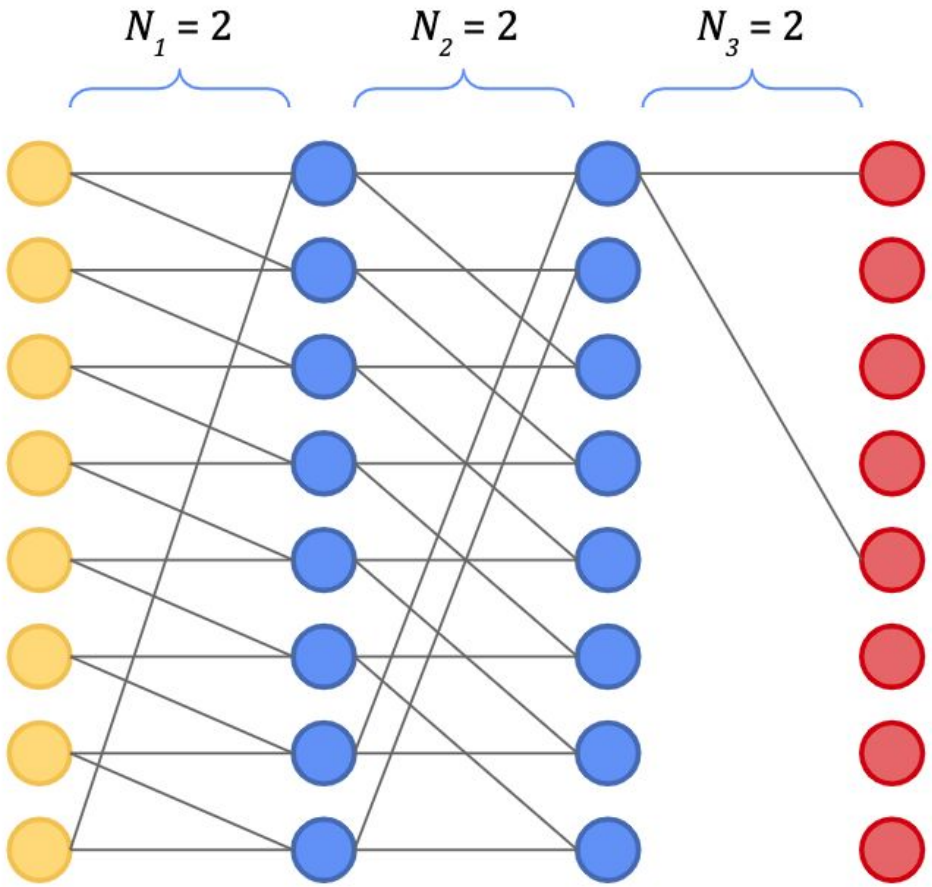
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



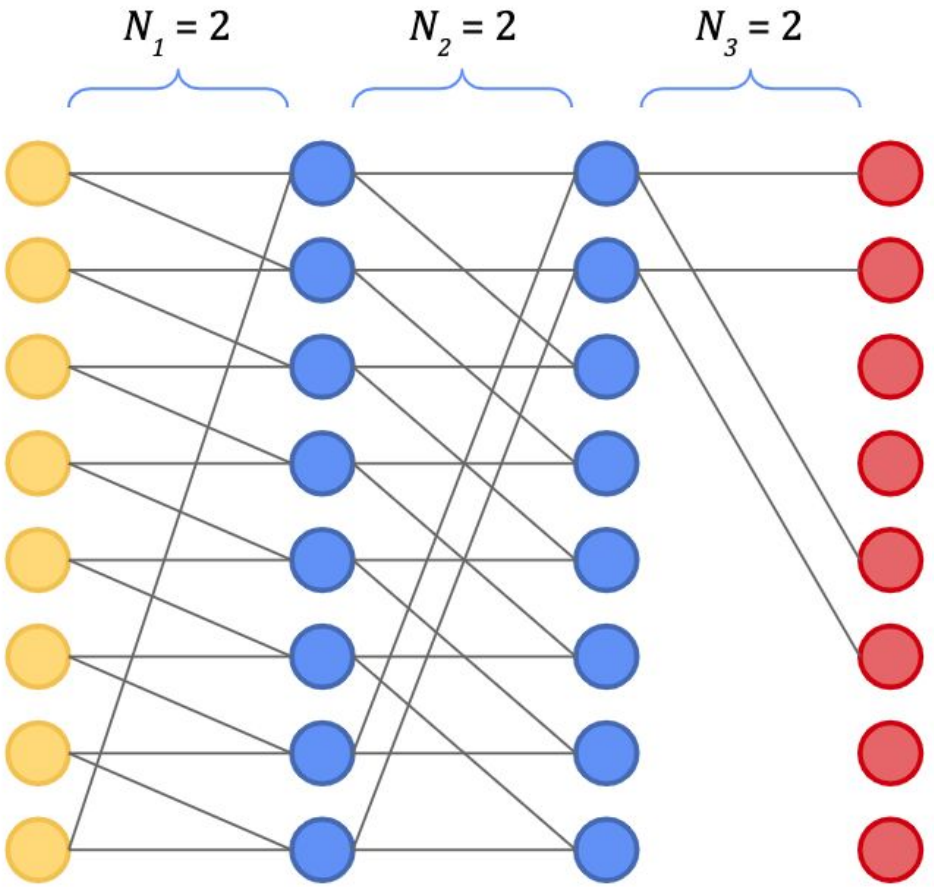
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



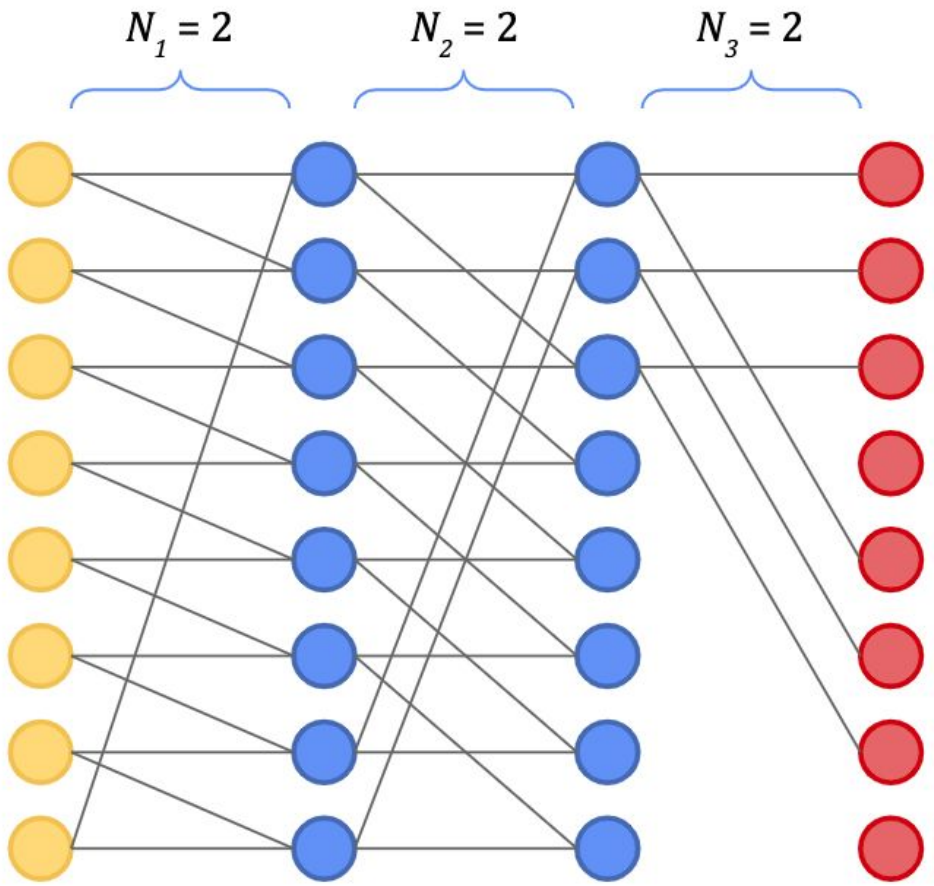
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



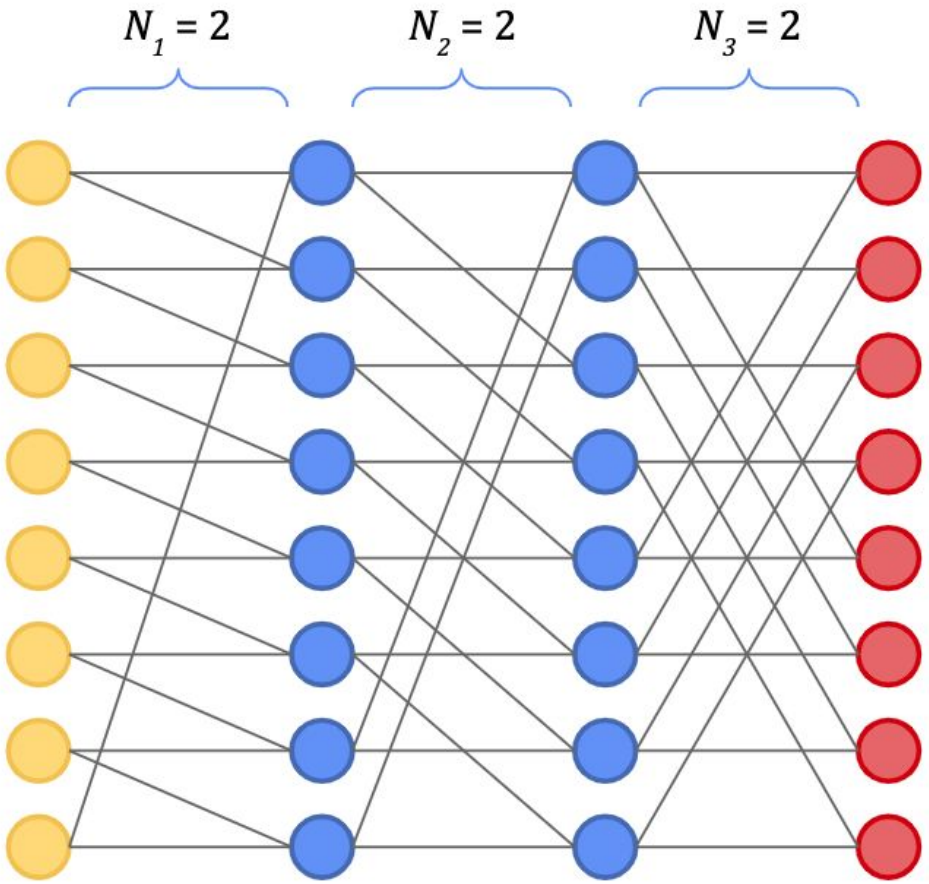
A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets



A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity.

RadiX-Nets

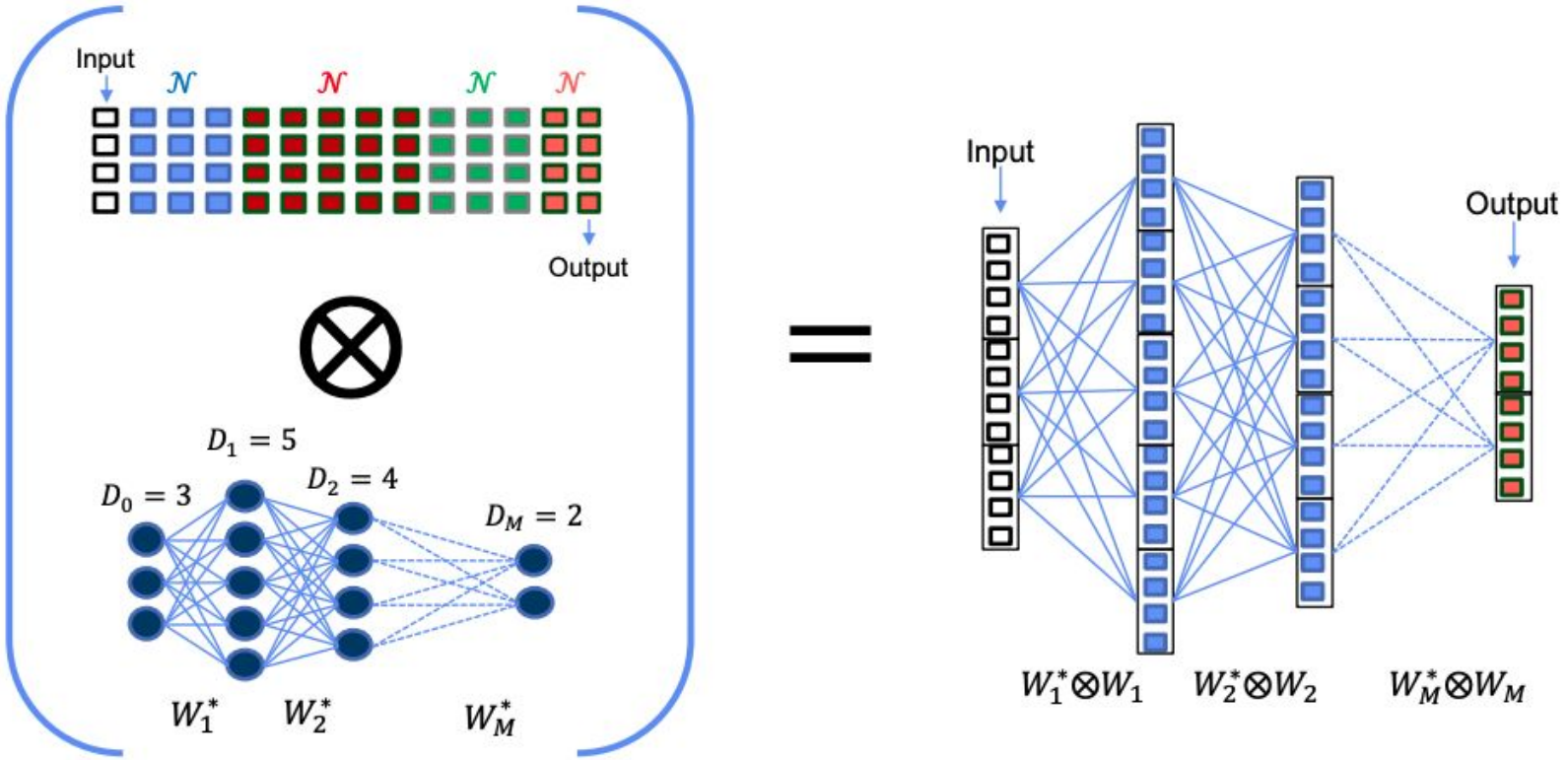
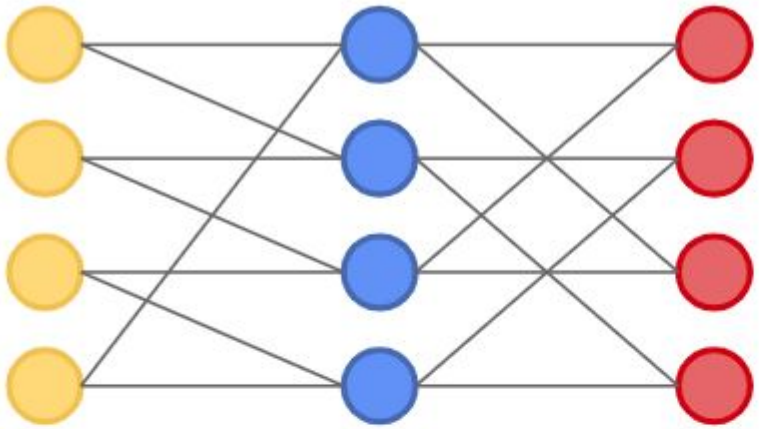
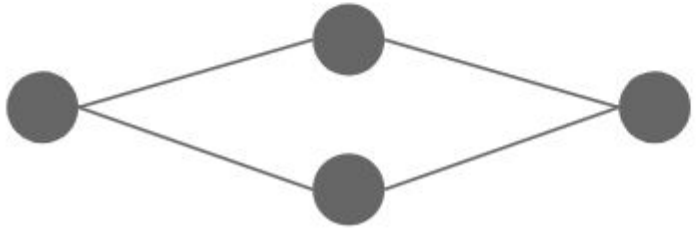


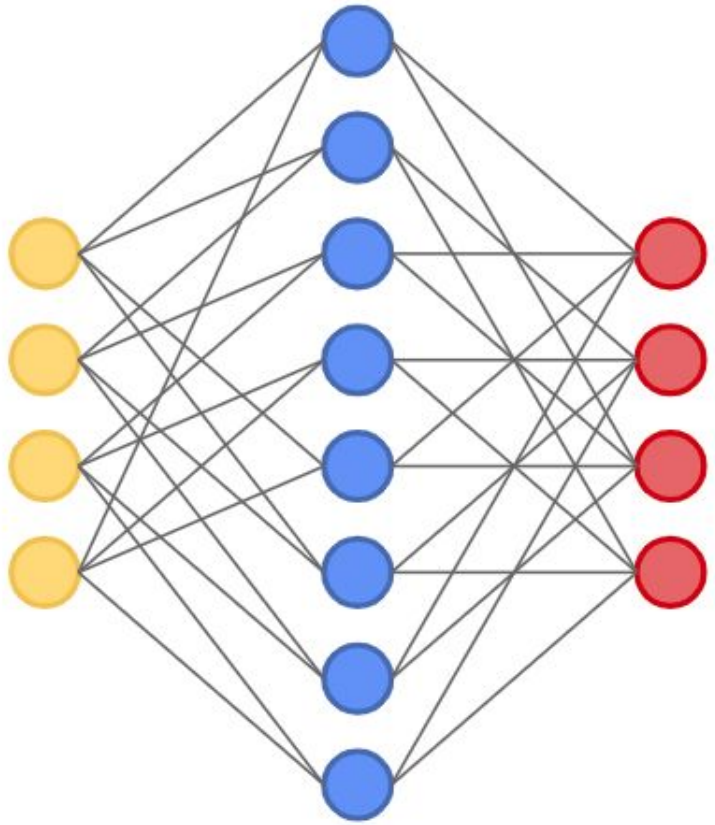
Fig. 5. The final step of RadiX-Net construction involves Kronecker products of adjacency submatrices of mixed-radix topologies and adjacency submatrices of an arbitrary dense deep neural network with the same number of layers. The number of vertices in each layer of the dense deep neural networks provides an additional set of parameters by which a wide range of RadiX-Nets can be defined.

RadiX-Nets

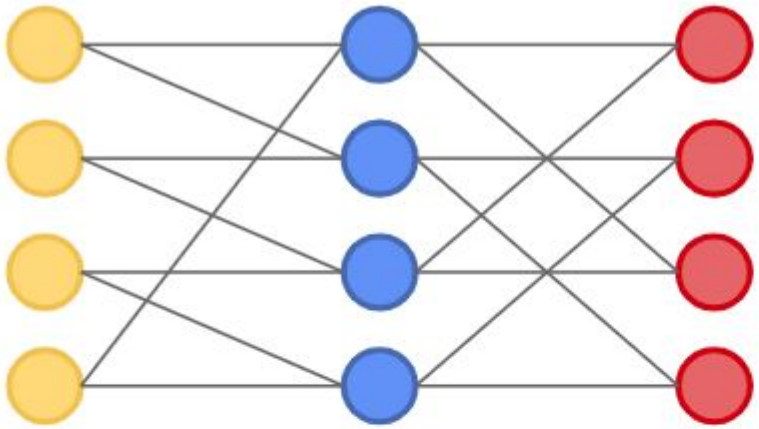


Kronecker Product

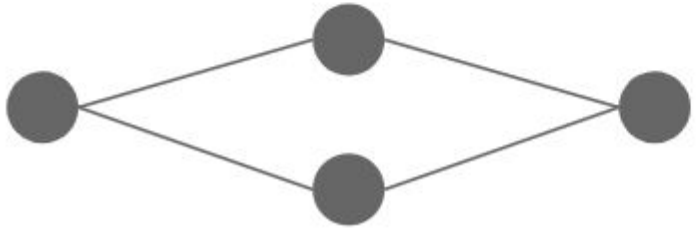


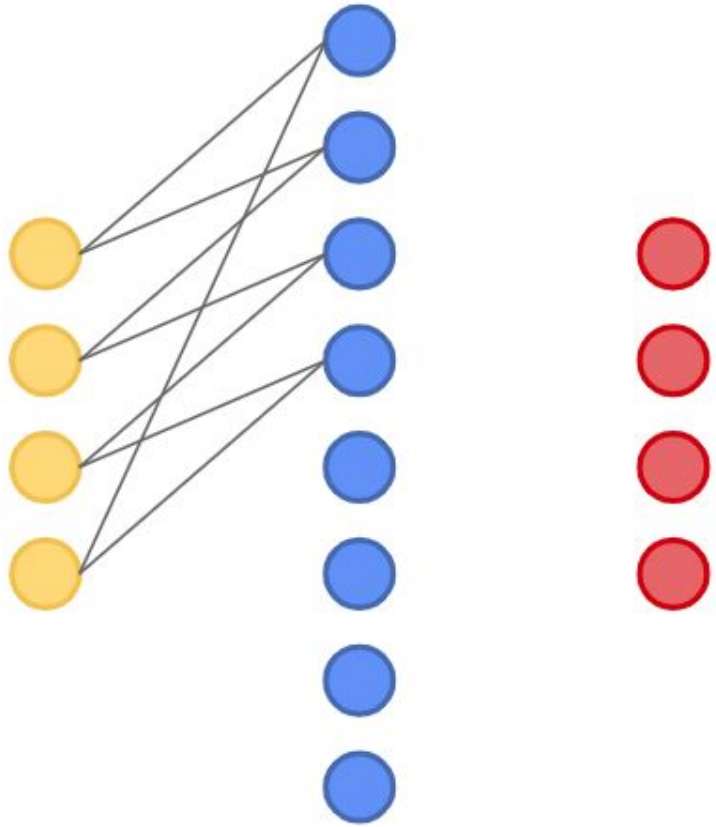


RadiX-Nets

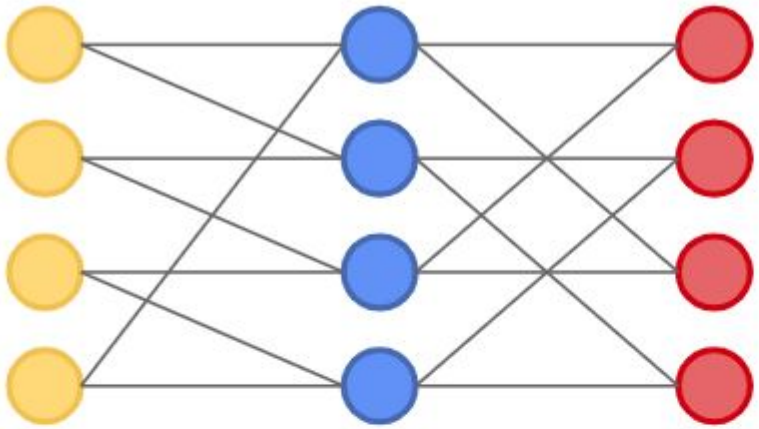


Kronecker Product

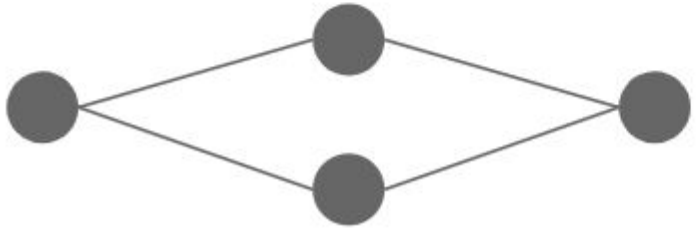


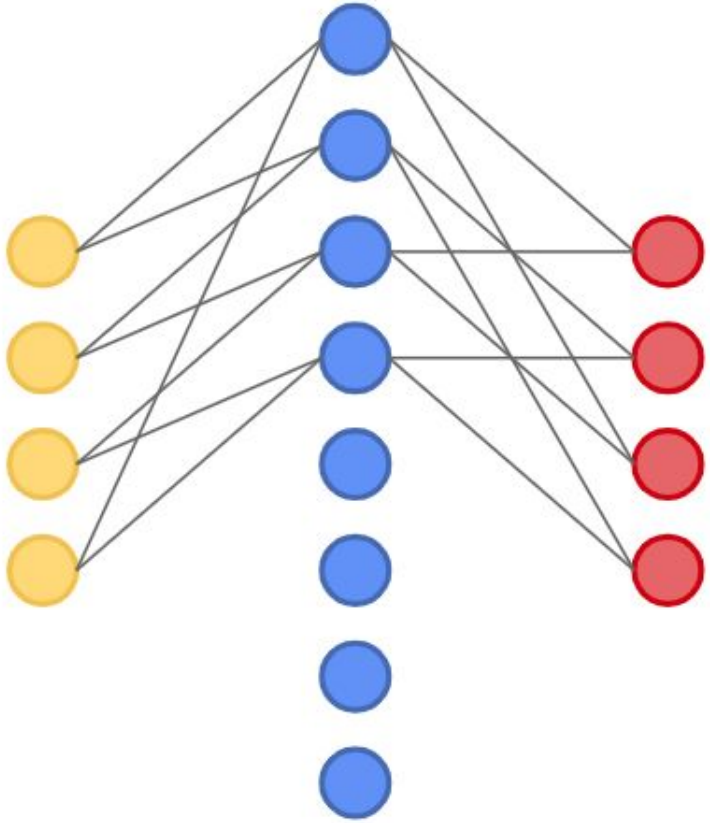


RadiX-Nets

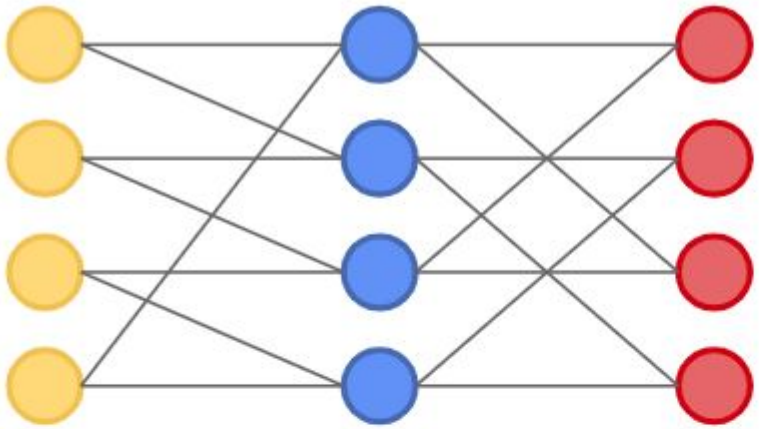


Kronecker Product

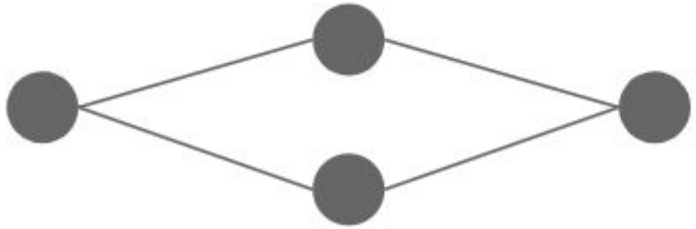


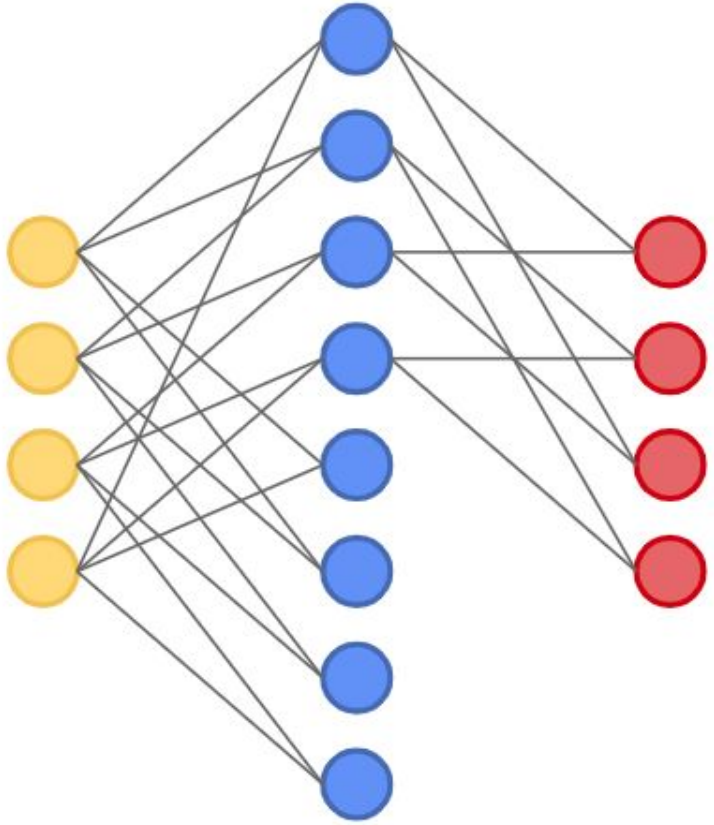


RadiX-Nets

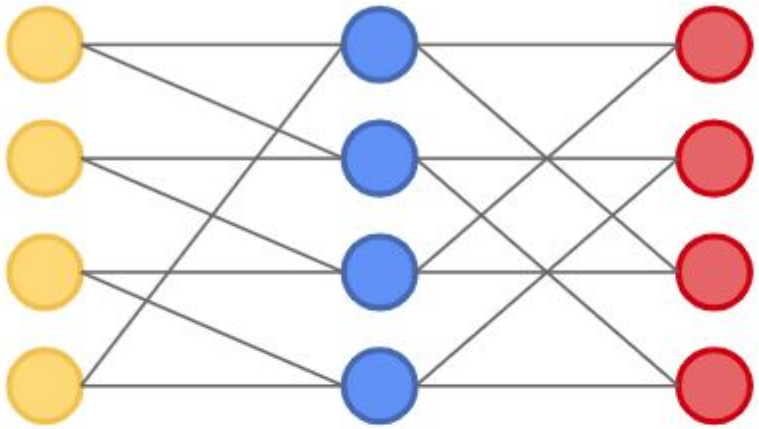


Kronecker Product

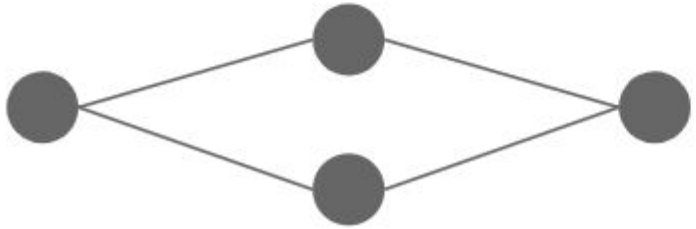




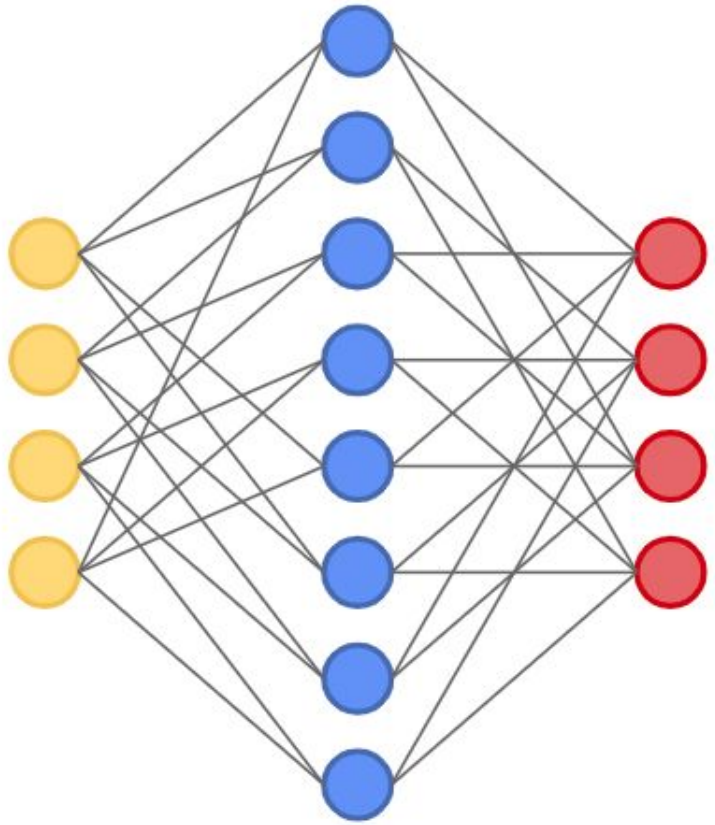
RadiX-Nets



Kronecker Product

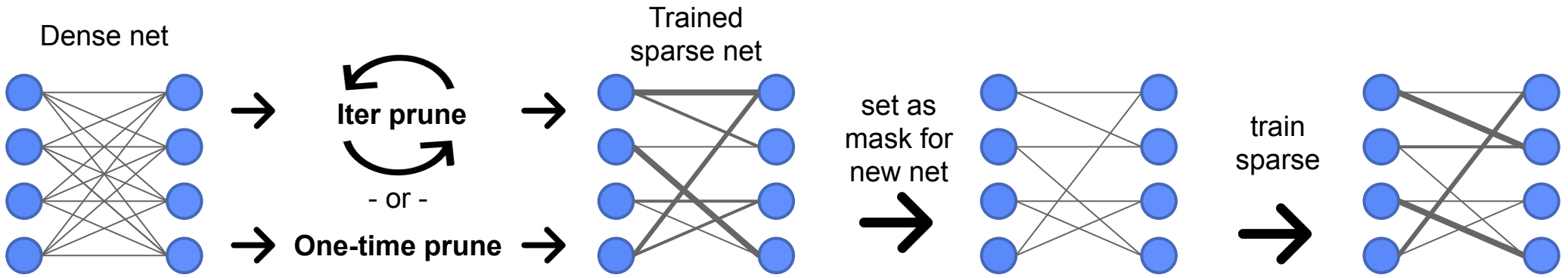


Kronecker-ed network maintains 50% sparsity



Pruning implementation details

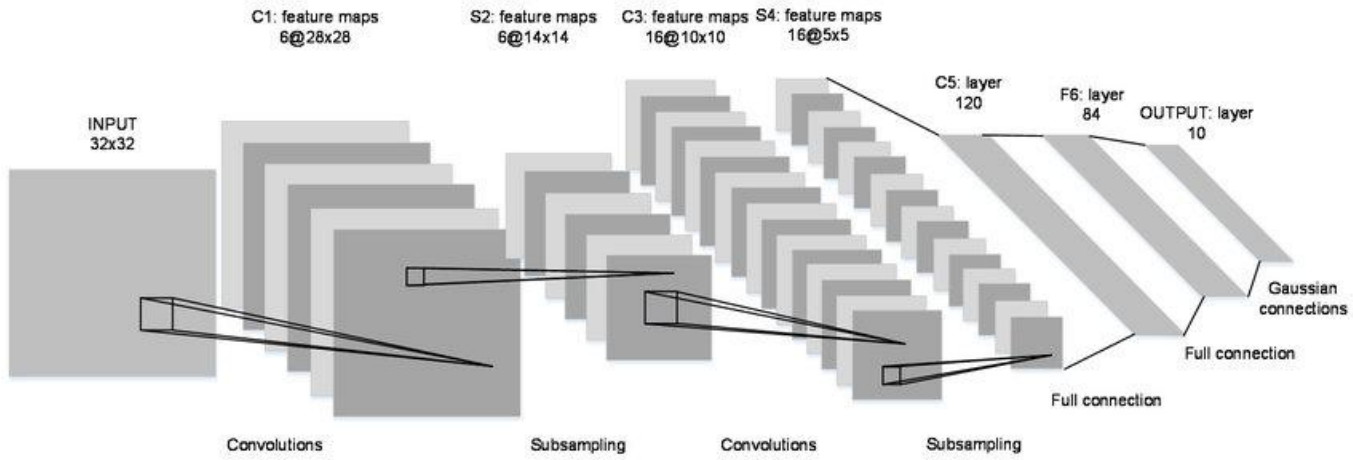
- Lenet 5 trained on MNIST and CIFAR-10
- Lenet 300-100 trained only on MNIST
- Pruned with one-time and iterative pruning to 0, 50, 75, 90, 95, and 99 percent sparsity
- Implemented in Tensorflow using mask variables to ignore pruned/nonexistent connections



Networks used

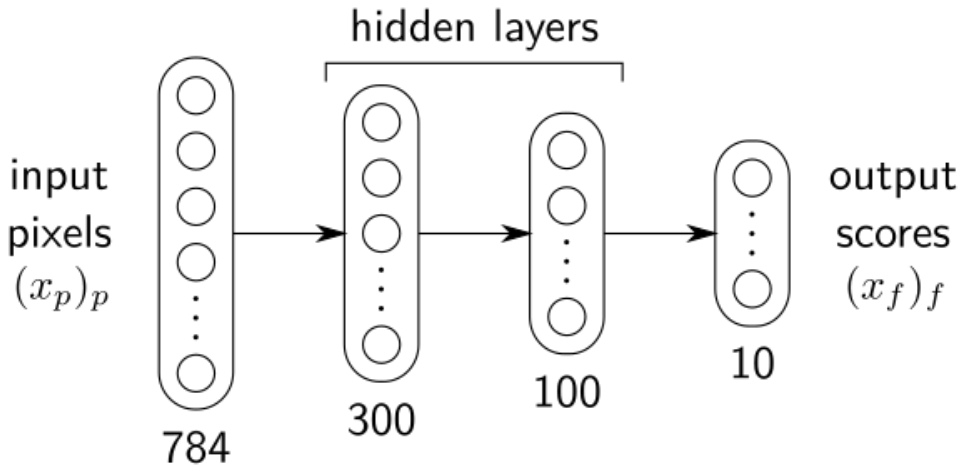
Lenet 5

- 2 convolutional layers
- 2 subsampling layers
- 1 fully-connected layer



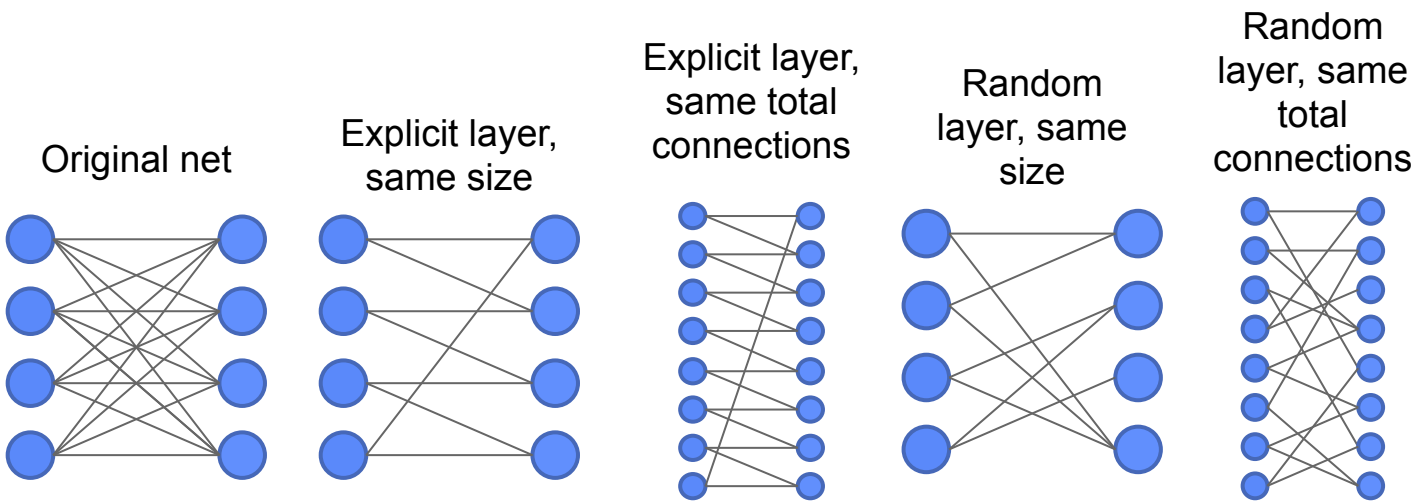
Lenet 300-100

- 2 fully-connected layers



RadiX-Net implementation details

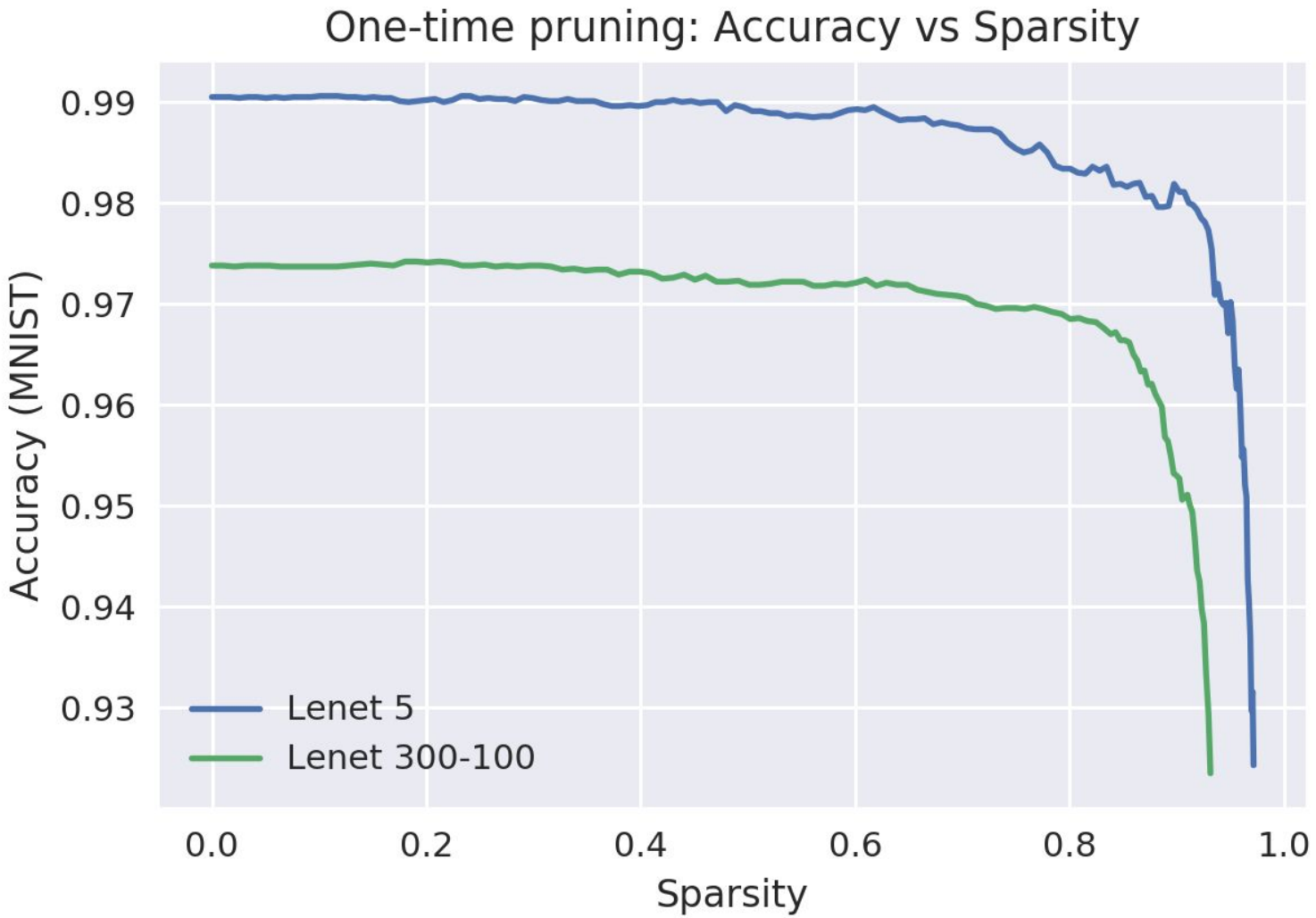
- Same networks, datasets
- Created sparse versions of each network using random and/or explicit RadiX-nets
- Compared keeping number of connections constant while varying sparsity and varying sparsity over network of same size
- Example: for Lenet 300-100, replaced fully connected layers with RadiX-Net with *N* = [10, 10], *B* = [30, 8, 1] = 90% sparse



Outline

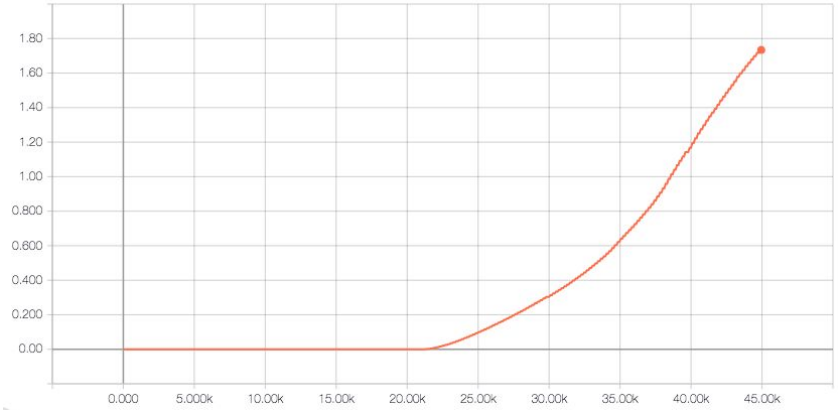
- Introduction
- Approach
- **Results**
- Interpretation and Summary

Results: One-Time Pruning

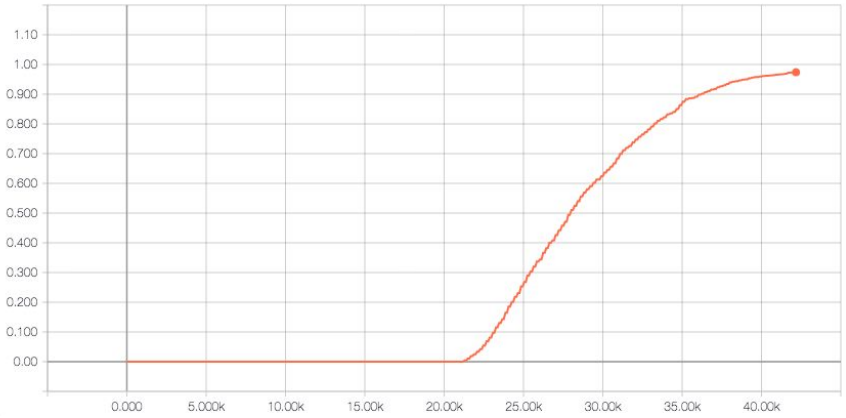


Results: Iterative Pruning

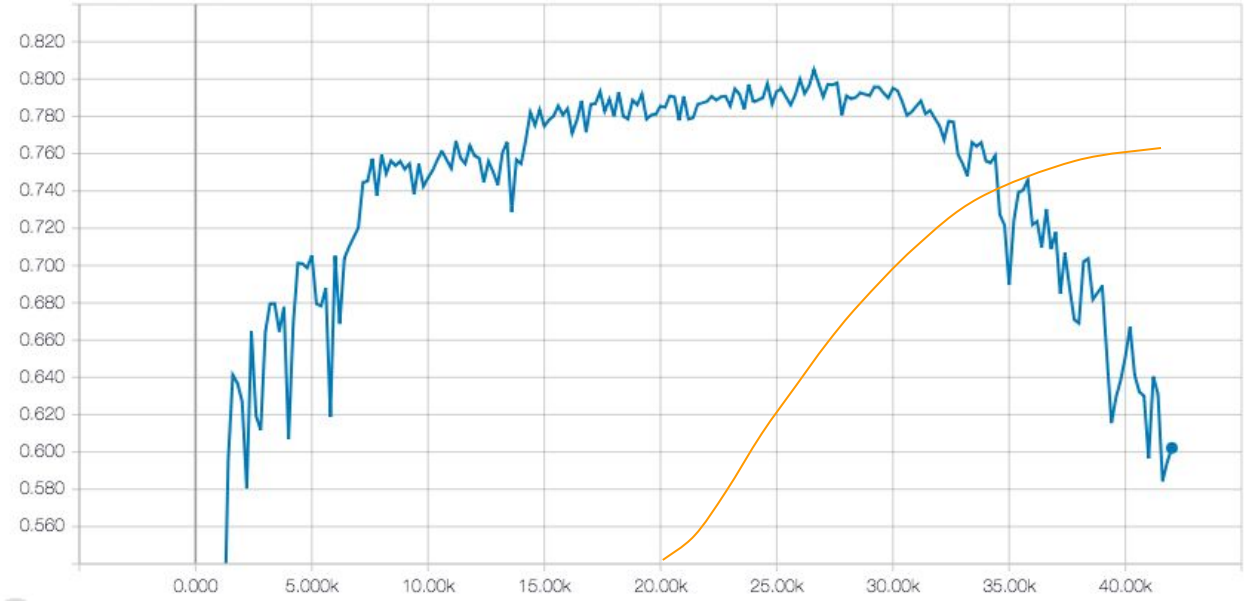
Layer pruning weight threshold over time



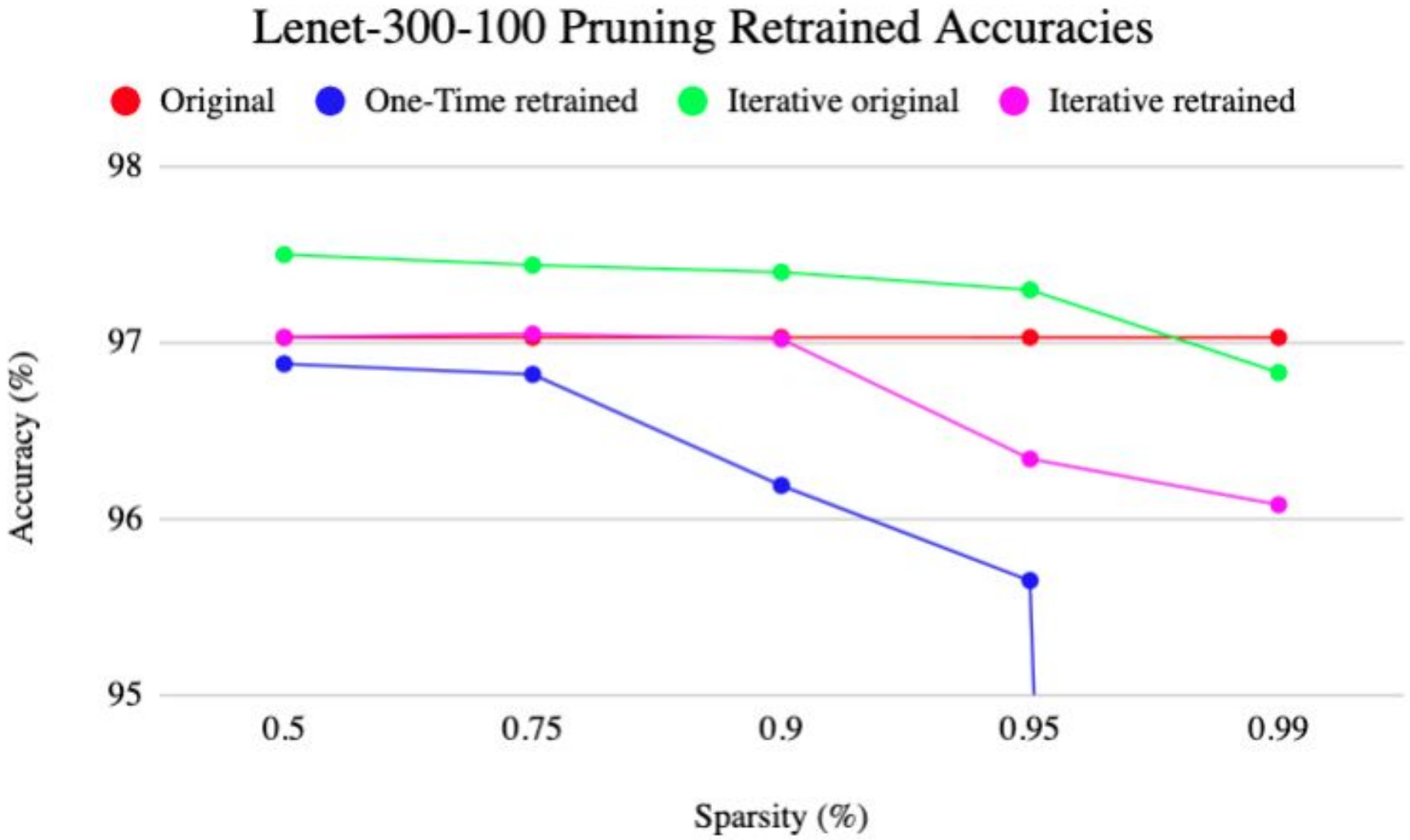
Layer sparsity over time



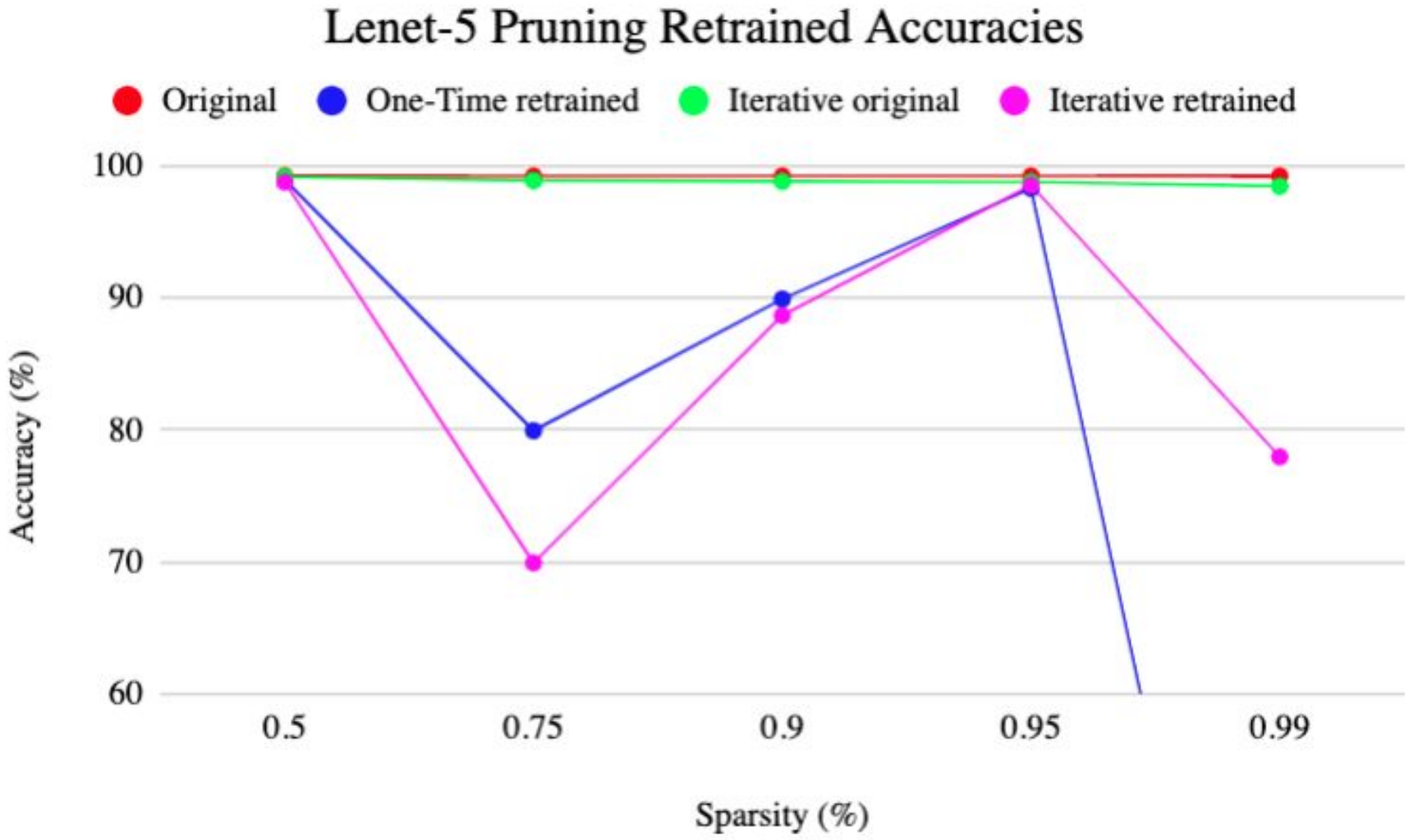
Model accuracy over time for Lenet 5 on CIFAR-10



Results: Training on pruned network structure



Results: Training on pruned network structure



Lenet-5 training on pruned network structure

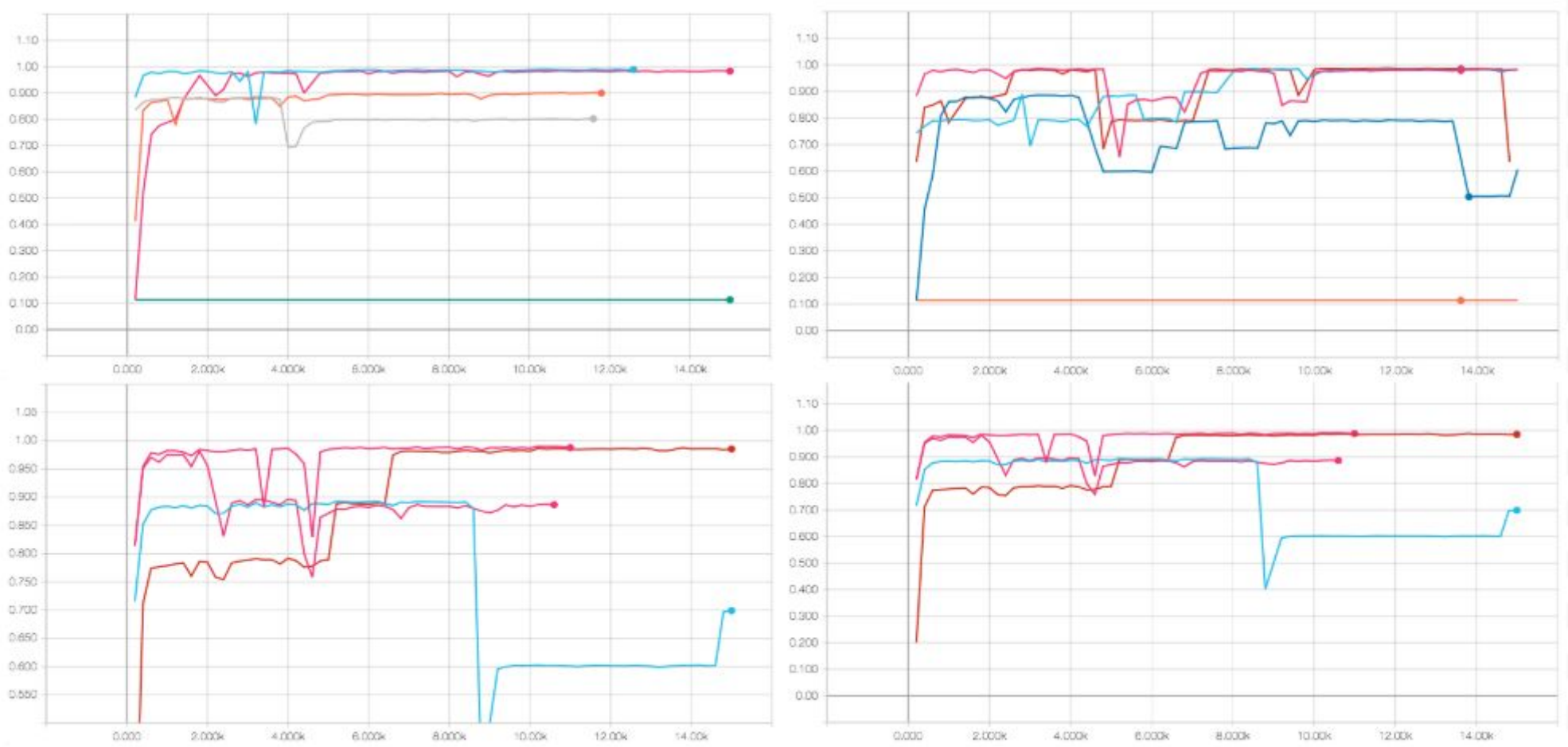
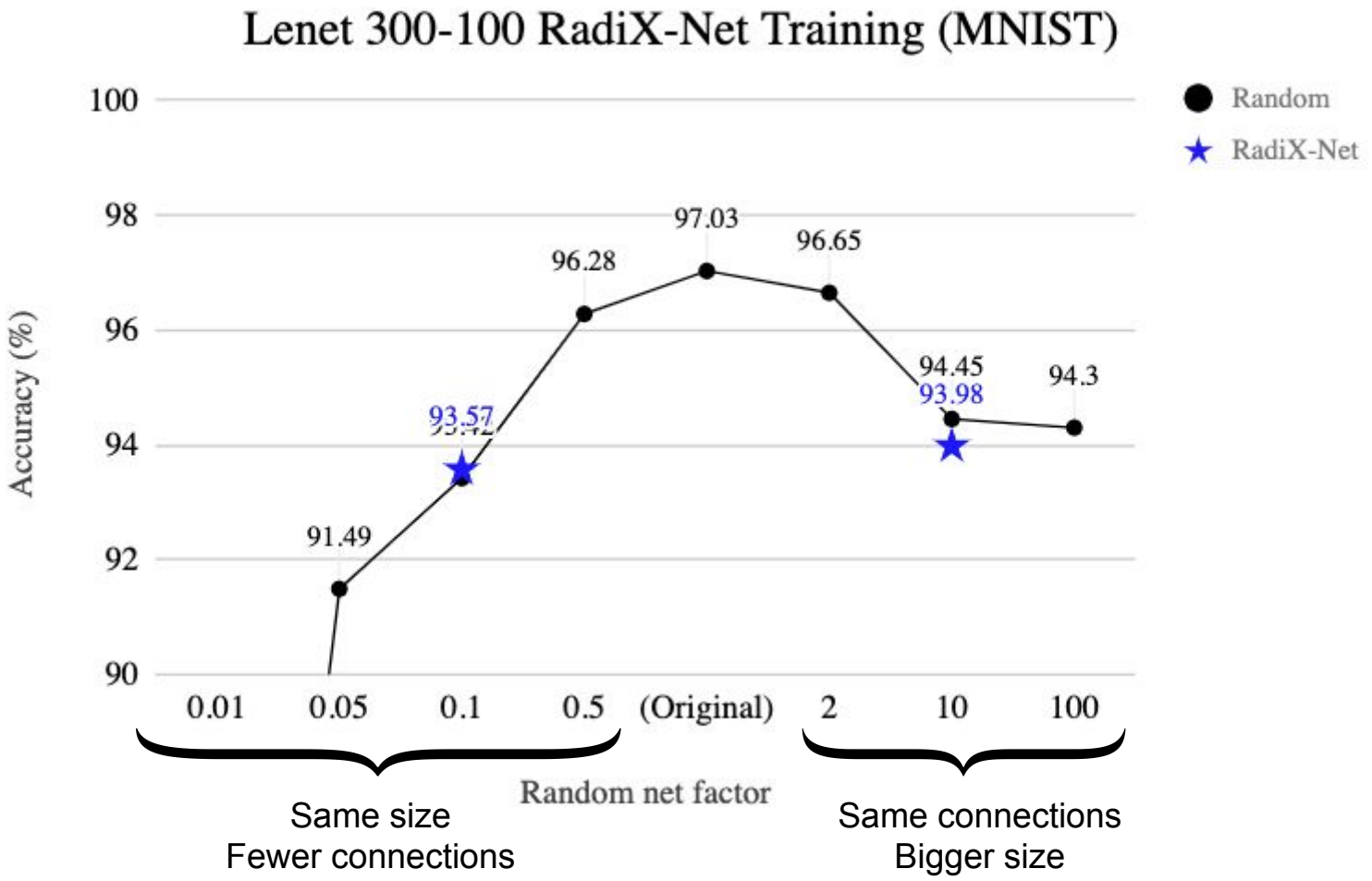
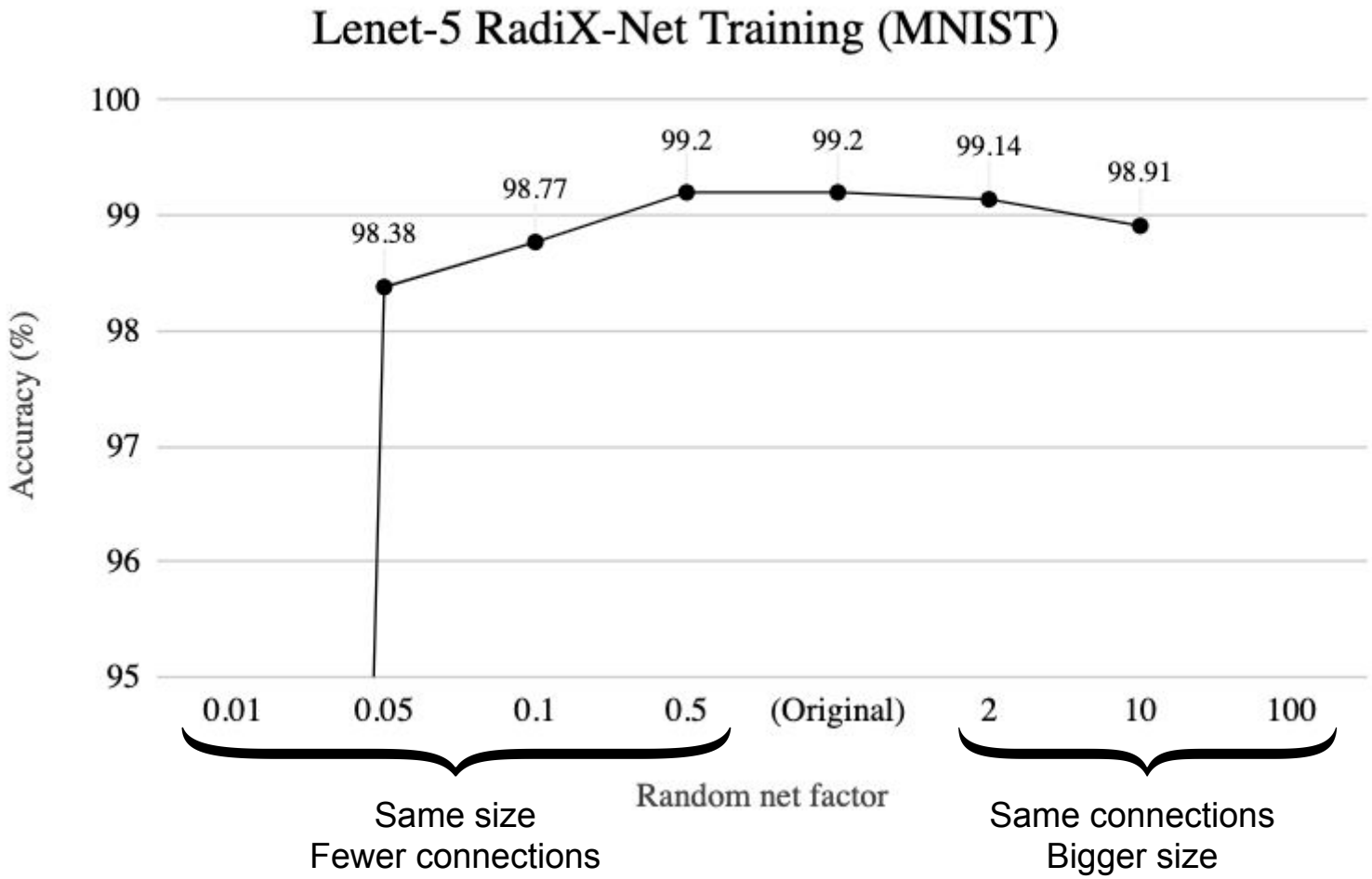


Fig. 8. From top left to bottom right, the result of training Lenet-5 on MNIST with pruned sparse structures of 0.75, 0.9, 0.95, and 0.99 percent sparsity. The figures show the instability of training on the sparse pruned network considering multiple Lenet-5 runs. Different colors represent retraining on the same sparse structure with everything identical except for different weight initializations.

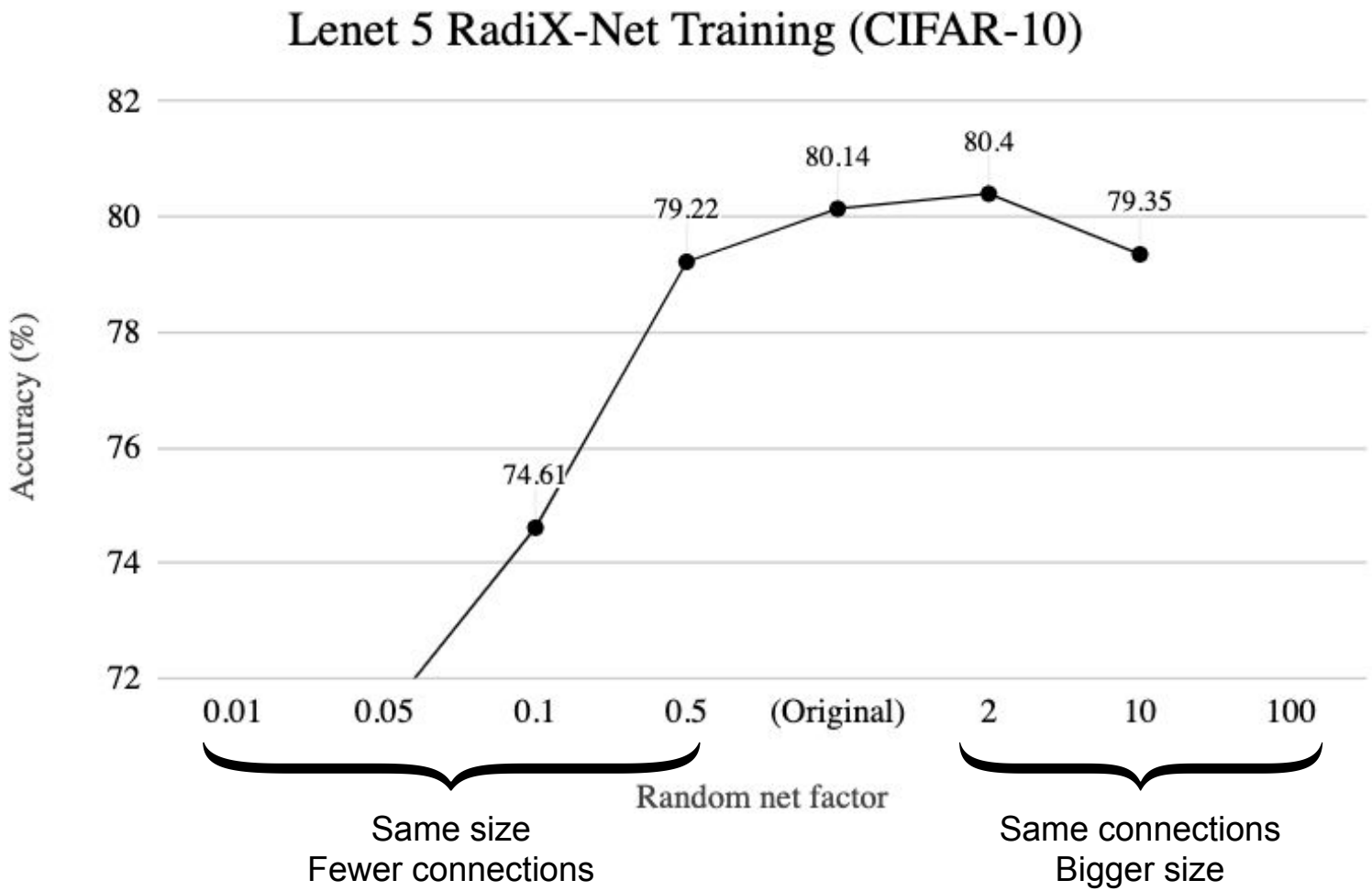
Results: RadiX-Net Training



Results: RadiX-Net Training



Results: RadiX-Net Training



Outline

- Introduction
- Approach
- Results
- **Interpretation and Summary**

Interpretation of Results

- RadiX-Net sparse networks work better with Lenet 5 than Lenet 300-100
- Better performance with lower sparsity
- Extreme levels of sparsity exhibits instability in training
- Pruning-based sparse networks work better with Lenet 300-100 than Lenet 5
- Random and explicit RadiX-Net layers behave the same
- For both RadiX-Net and pruning-based networks, performance depends on network at hand

Summary, Future Work and Next Steps

- Need to evaluate performance on larger networks to fully characterize each technique's behavior
- Investigating structure of pruned network
- Develop more fine-tuned sparse strategies for replacing more specialized layers such as convolutional layers, attention layers, etc.
- Utilize sparse matrix libraries for matrix multiplication