Training Sparse Neural Networks

Simon Alford

August 22, 2018

Massachusetts Institute of Technology

Outline

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

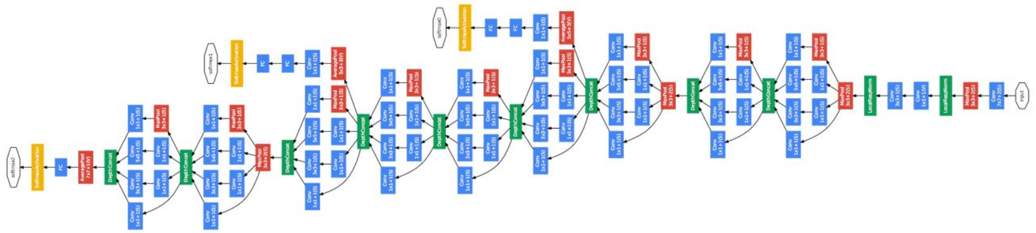
About Me

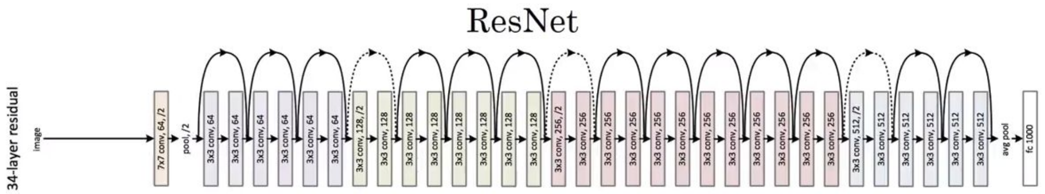
Simon Alford

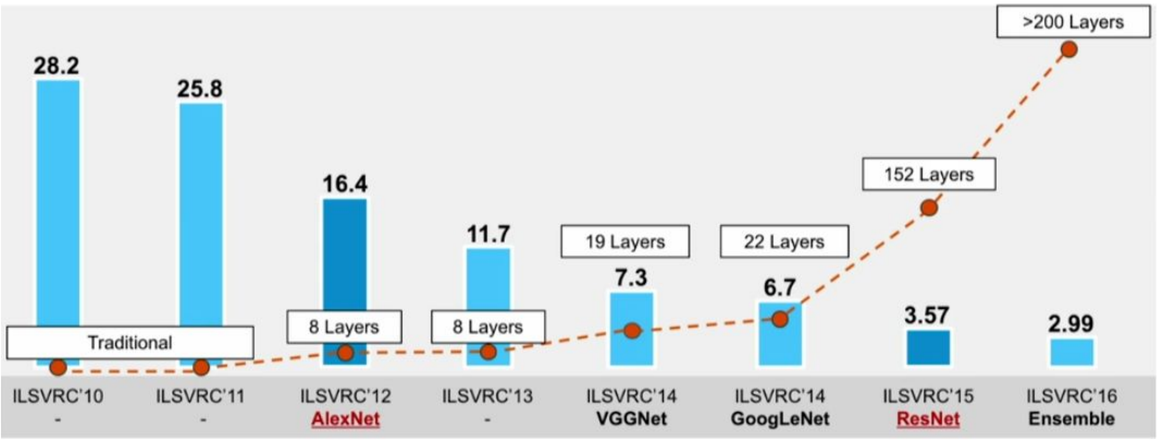
- From Dublin, Ohio
- Rising junior at MIT
- Majoring in math
- Member of MIT varsity Track & Field team (800m)
- Enjoy playing piano

Motivation

- For deep neural networks, the larger the network, the better the performance
- Current state-of-the-art is limited by computational power available
- Challenge: how can we train larger networks with fewer computational resources?





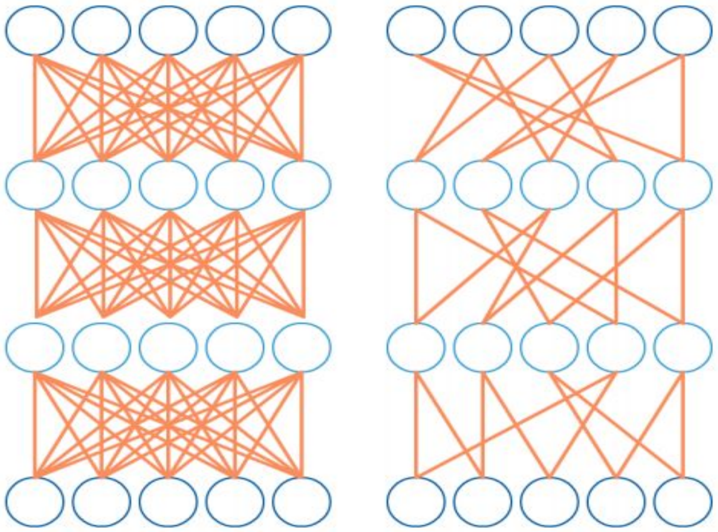


State of the art neural networks, like Microsoft's ResNet and Google's Inception network, stack many layers and take weeks to fully train

Motivation

Challenge: How can we train larger networks more efficiently*?*

$$\mathbf{y}_{k+1} = h(\mathbf{W}_k \mathbf{y}_k + \mathbf{b}_k)$$



Fully connected   Sparse

Idea: "Go sparse"

- DNN computation consists of repeated matrix operations. Each matrix represents connections between neurons of successive layers.
- To reduce computation, replace fully connected layers with sparsely connected layers
- Leverage pre-existing work done on computation and storage of sparse matrices
- Sparse algorithms can scale with the number of connections rather than the number of neurons
- Sparse network structures may train more effectively than dense

Problem Solved/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, attaining ~90% sparsity, no accuracy loss
- **While much research has been done pruning pretrained networks to become sparse, little has been done training on sparse network structures**
- *Deep Expander Networks*[3]
  - Represent connections using random and explicit expander graphs to create trainable sparse networks with strong connectivity properties

Our contribution: Evaluation of effectiveness of both 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

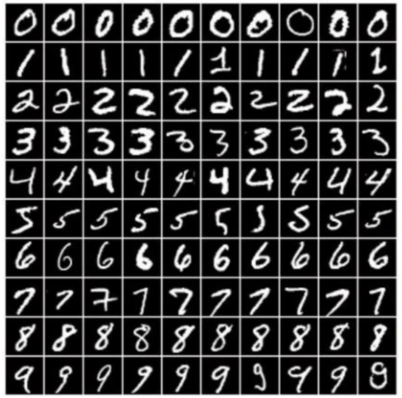
- Prune the network after/as it is being trained to learn a sparse network structure
- Initialize network with pruned network as structure and train

Second approach: RadiX-Nets

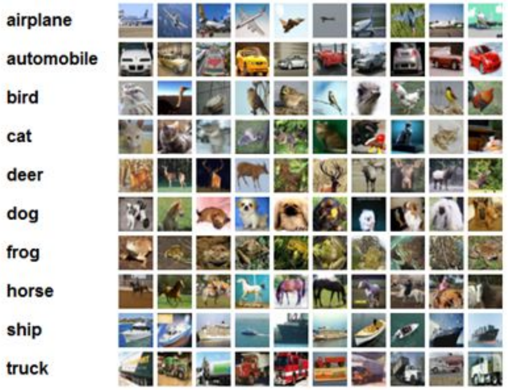
- Ryan Robinett's RadiX-Nets provide theoretical guarantees of sparsity, connectivity properties
- Train RadiX-Nets and compare to dense training

Implementation

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



MNIST



CIFAR-10

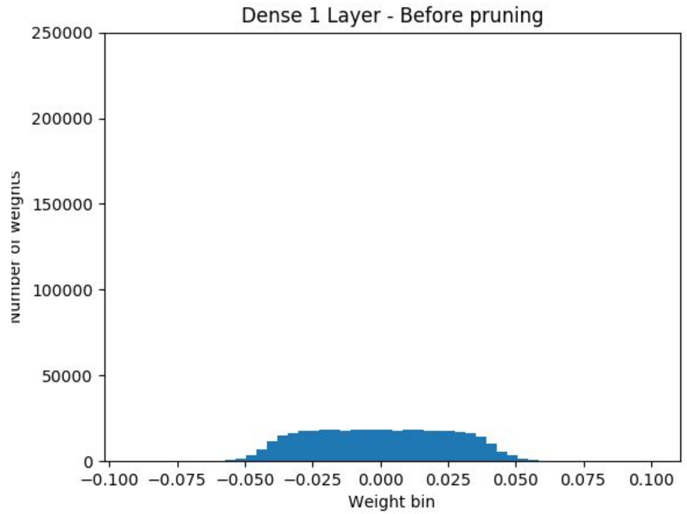
Outline

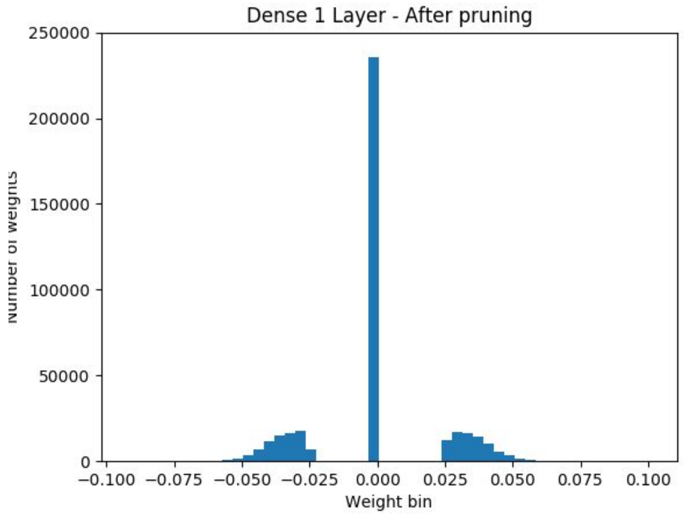
- Introduction
- **Approach**
- Results
- Interpretation and Summary

Generating a sparse network to train on

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

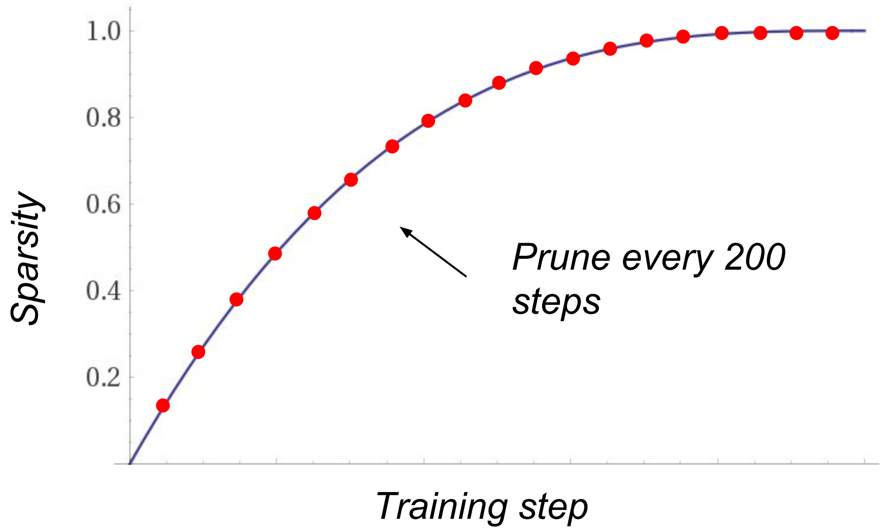




Generating a sparse network to train on

Iterative Pruning

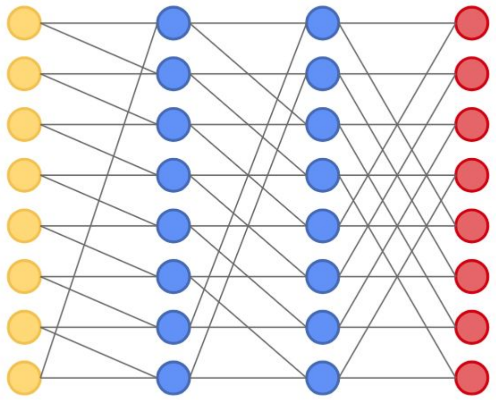
- Developed by Han et. al in *Learning both Weights and Connections for Efficient Neural Networks*
- Iteratively cycle between pruning neurons below threshold and retraining remaining neurons
- Modified iterative pruning: prune network to match monotonically increasing sparsity function $s(t)$
- Able to achieve much higher sparsity than one-time pruning without loss in accuracy (>95% vs 50%)
- Pruning more frequently led to better results — smoother transition to sparsity



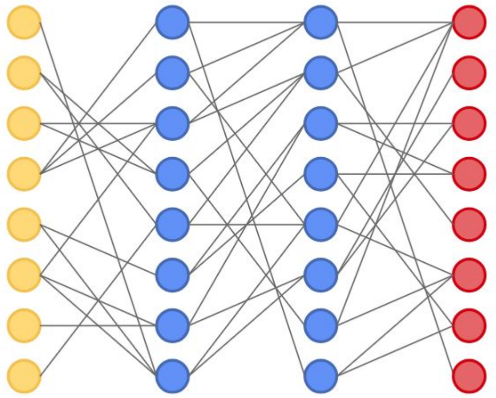
Generating a sparse network to train on

Second method: RadiX-Net

- Building off Prabhu et. al's *Deep Expander Networks*
- 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 connectedness and symmetry properties
- Designed layers to replace fully connected and convolutional layers of traditional networks with sparse equivalents
- Alternative to radix expanders: random expanders
- Both replace fully connected layer(s) with sparse layer(s)

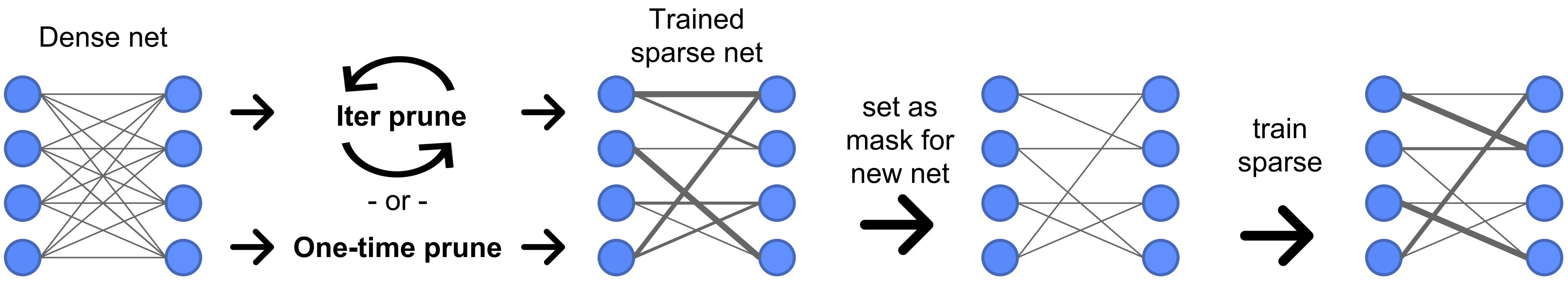


Above: A two layer RadiX-net with radix values (2, 2, 2) and 75% sparsity. Below: The random equivalent



Pruning implementation details

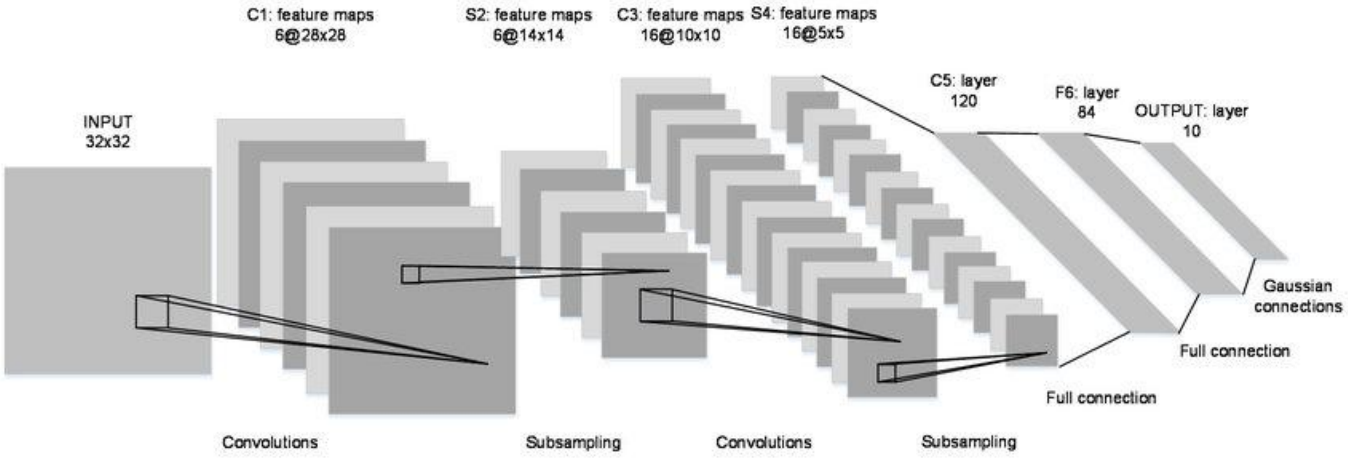
- Lenet 5 trained on MNIST and CIFAR-10
- Lenet 300-100 trained only on MNIST
- Pruned with one-time and iterative pruning to 0, 50, 75, 90, 95, and 99 percent sparsity
- Implemented in Tensorflow using mask variables to ignore pruned/nonexistent connections



Networks used

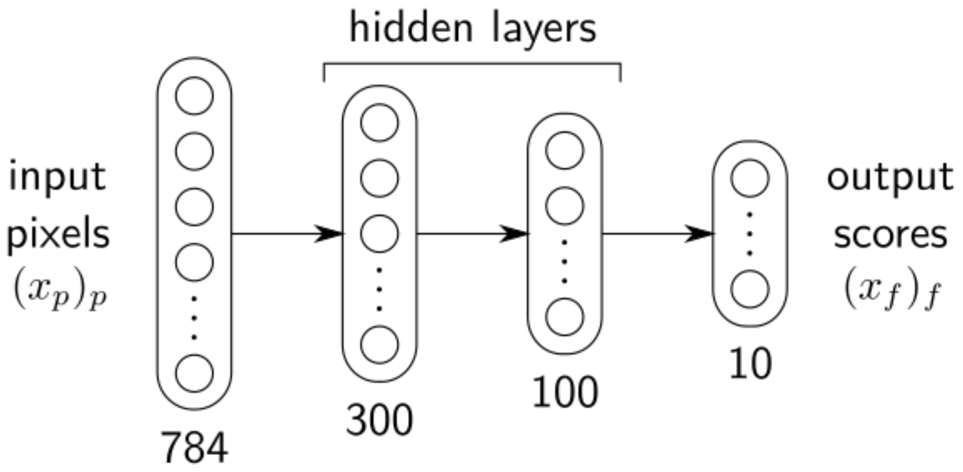
Lenet 5

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



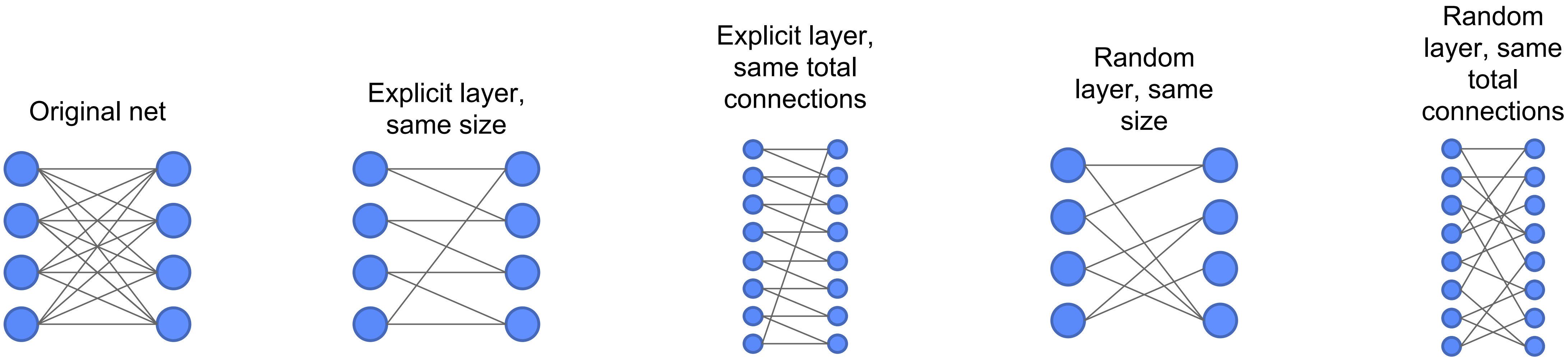
Lenet 300-100

- 2 fully connected layers



RadiX-Net implementation details

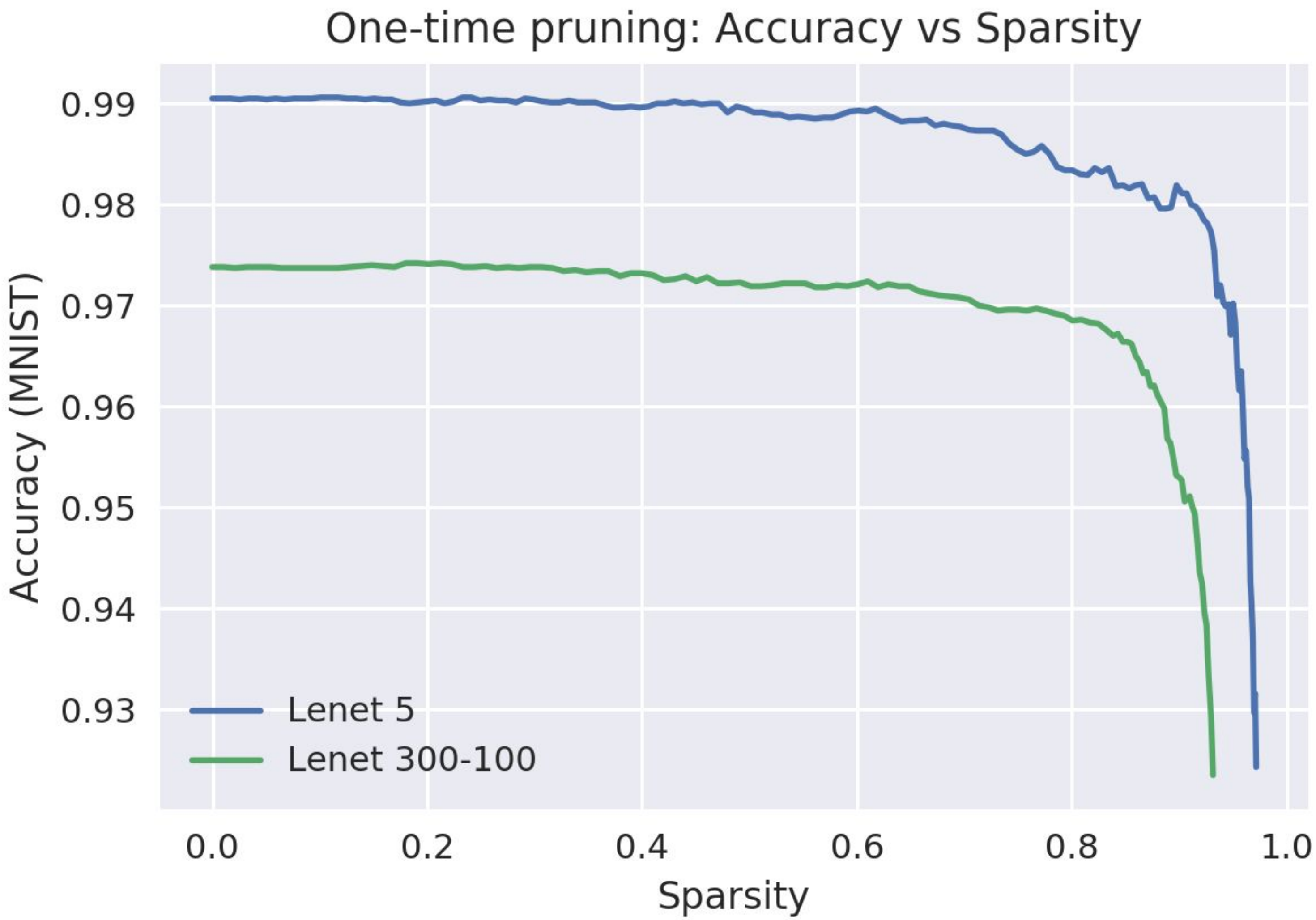
- Same networks, datasets as for pruning
- Created sparse versions of each network using random and/or explicit expanders
- Tried keeping number of connections constant while varying sparsity and varying sparsity over network of same size
- Example: for Lenet 300-100, replaced fully connected layers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



Outline

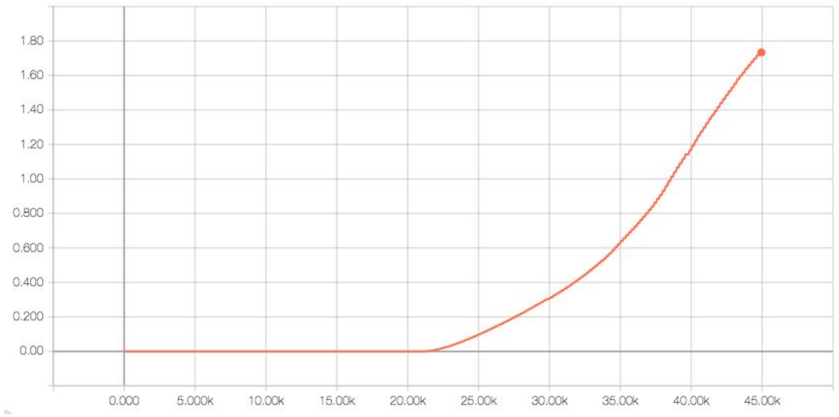
- Introduction
- Approach
- **Results**
- Interpretation and Summary

Results: One-Time Pruning

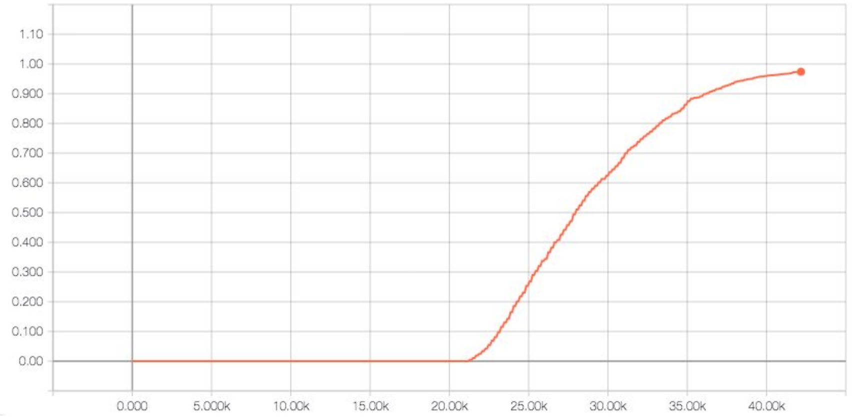


Results: Iterative Pruning

Layer pruning weight threshold over time

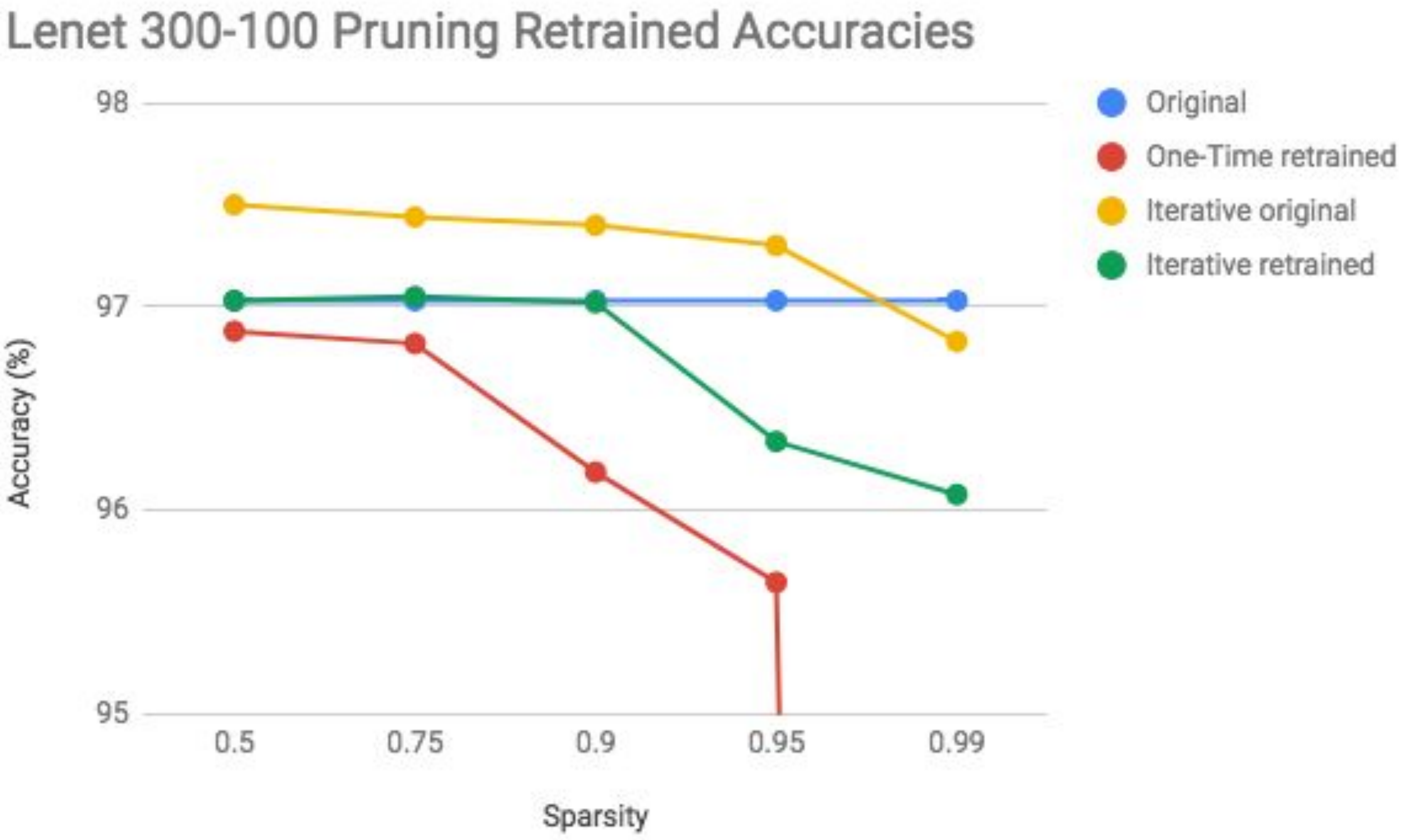


Layer sparsity over time

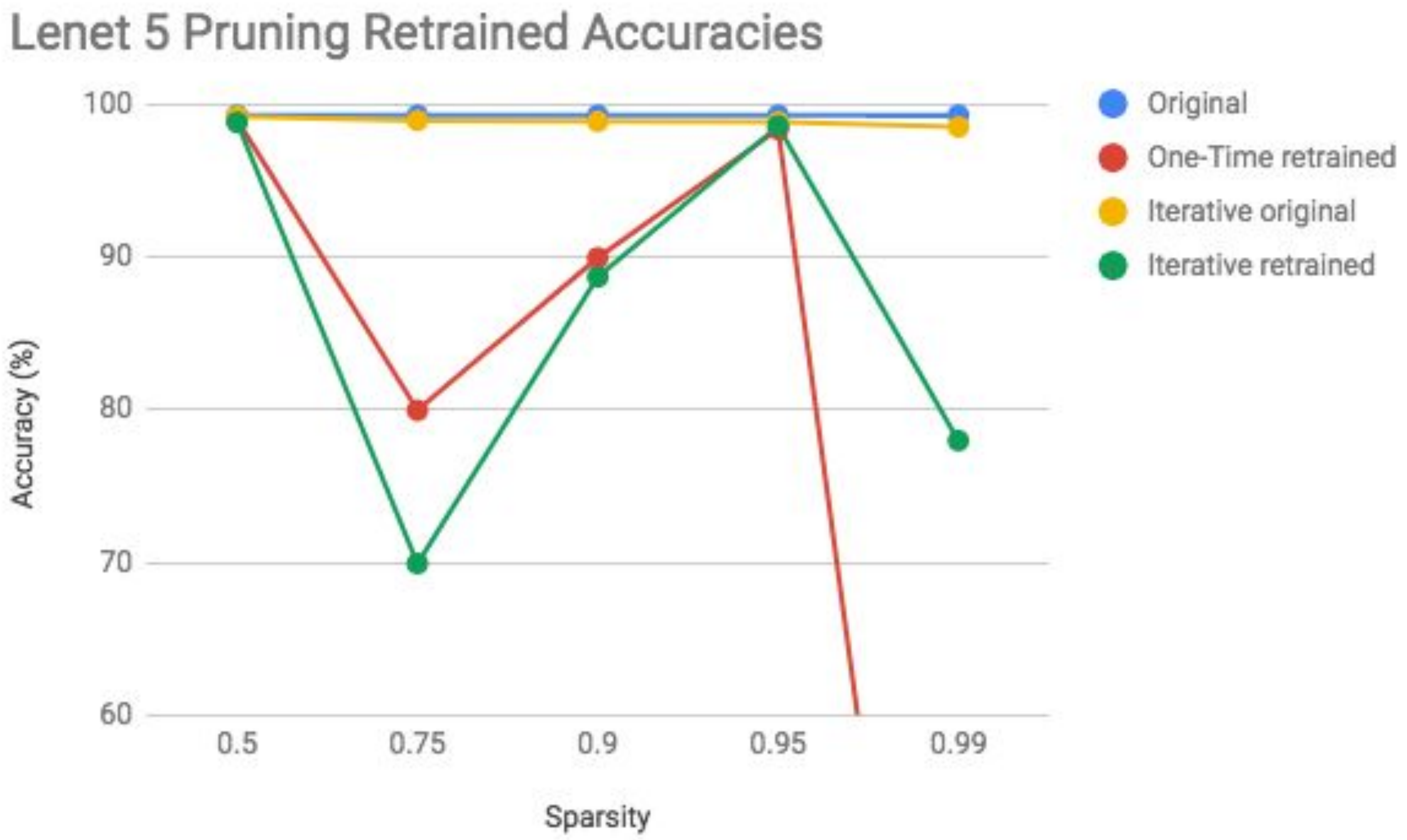


Model accuracy over time for Lenet 5 on CIFAR-10

Results: Training on pruned network structure

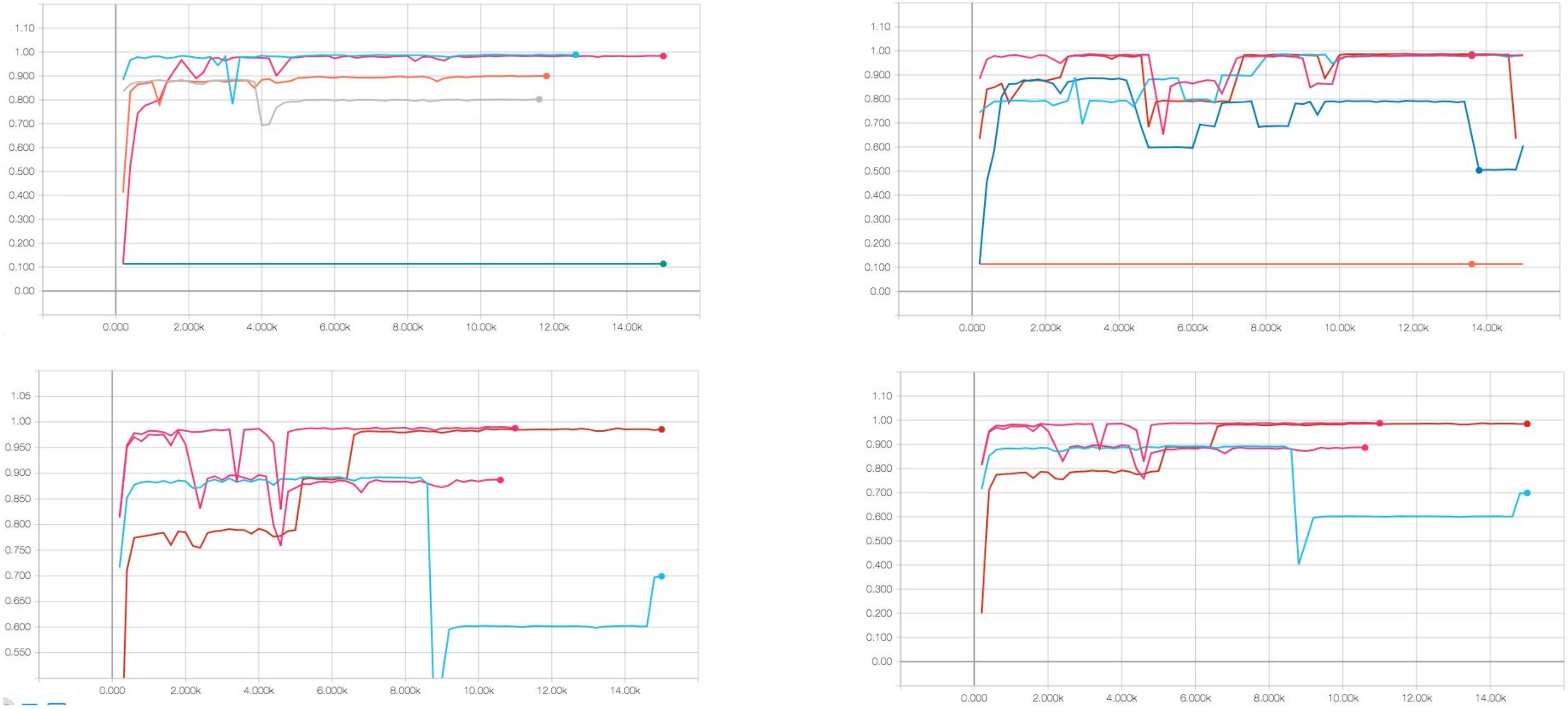


Results: Training on pruned network structure



Lenet-5 training on pruned network structure

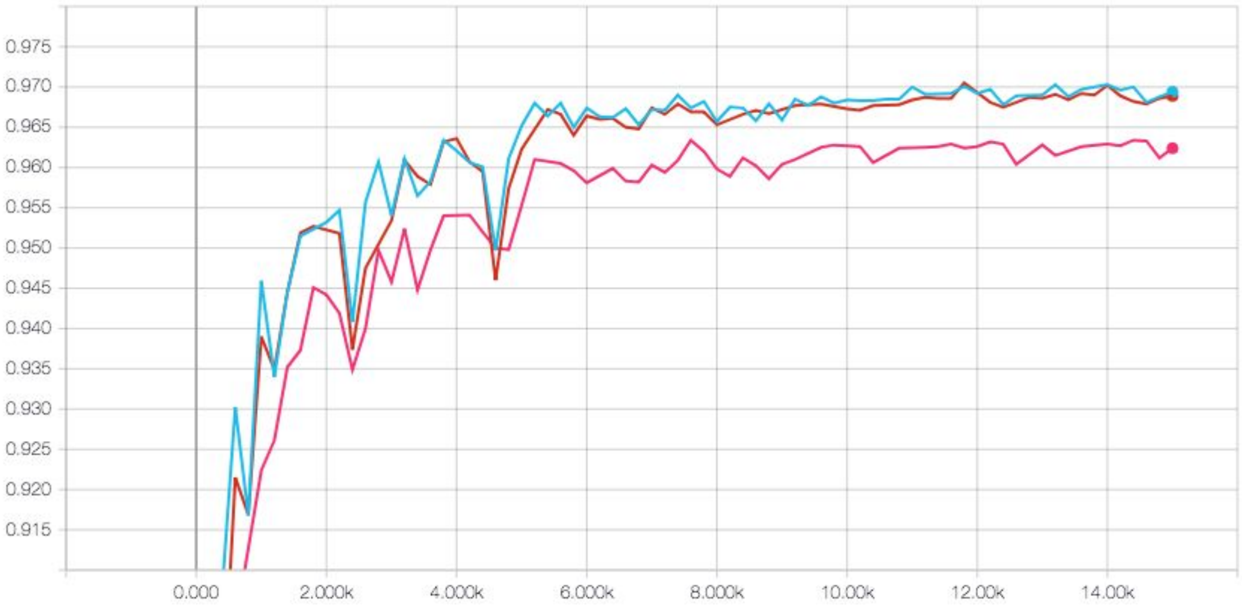
Model accuracy over time



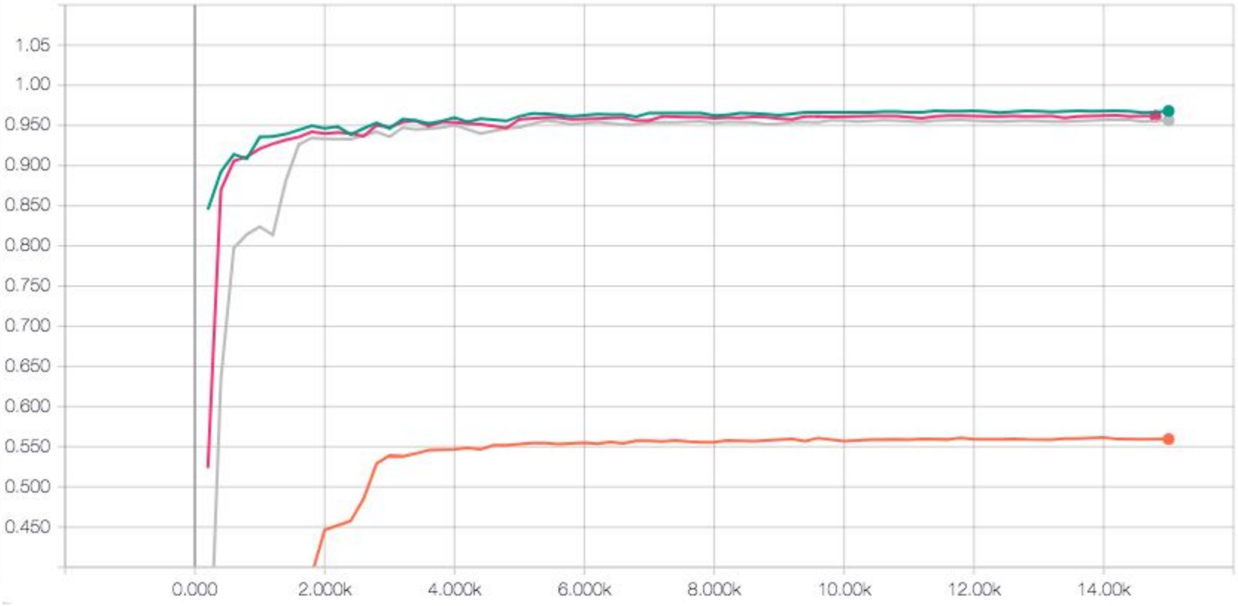
Lenet 300-100 training on pruned network structure

Model accuracy over time

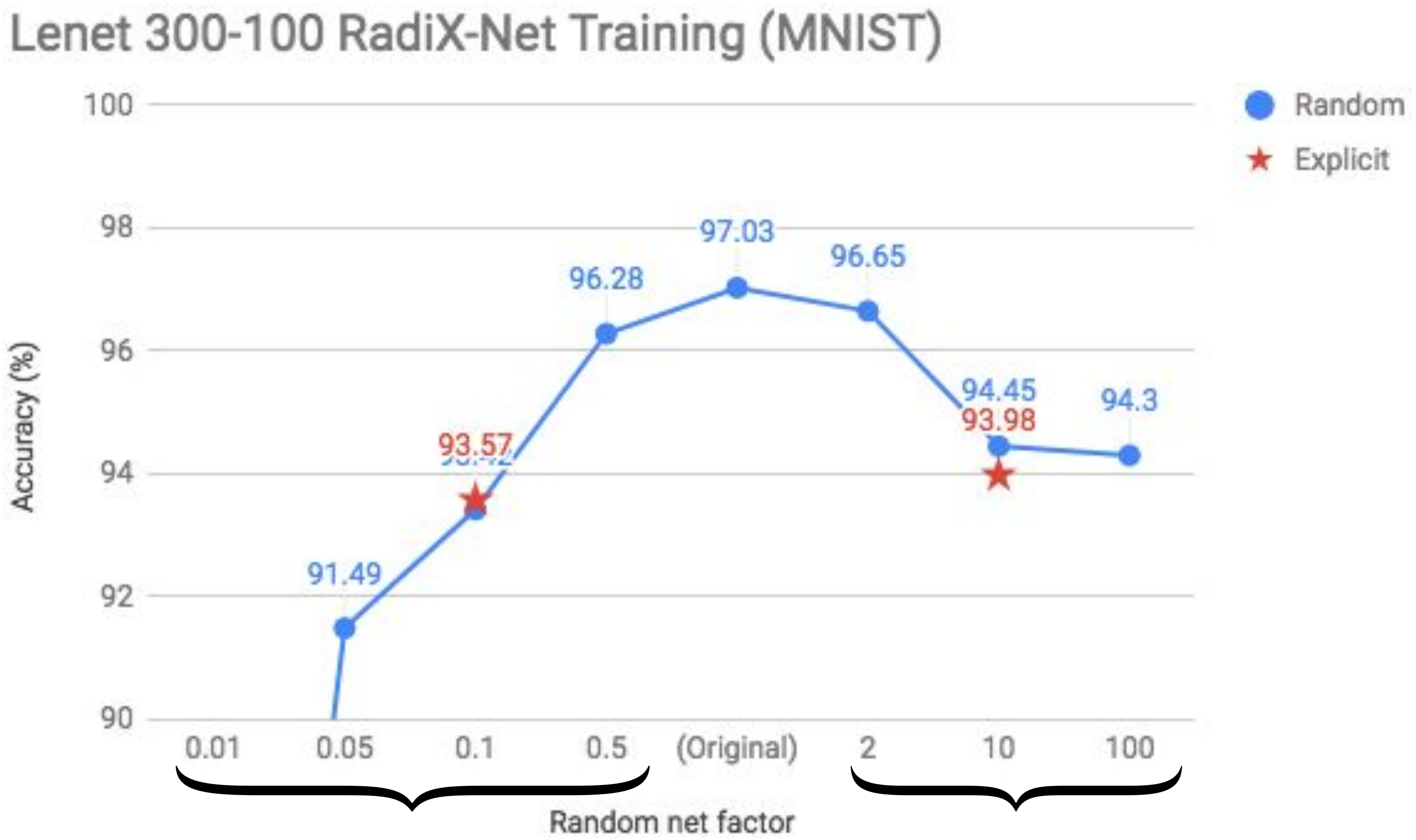
One-time pruning



Iterative pruning



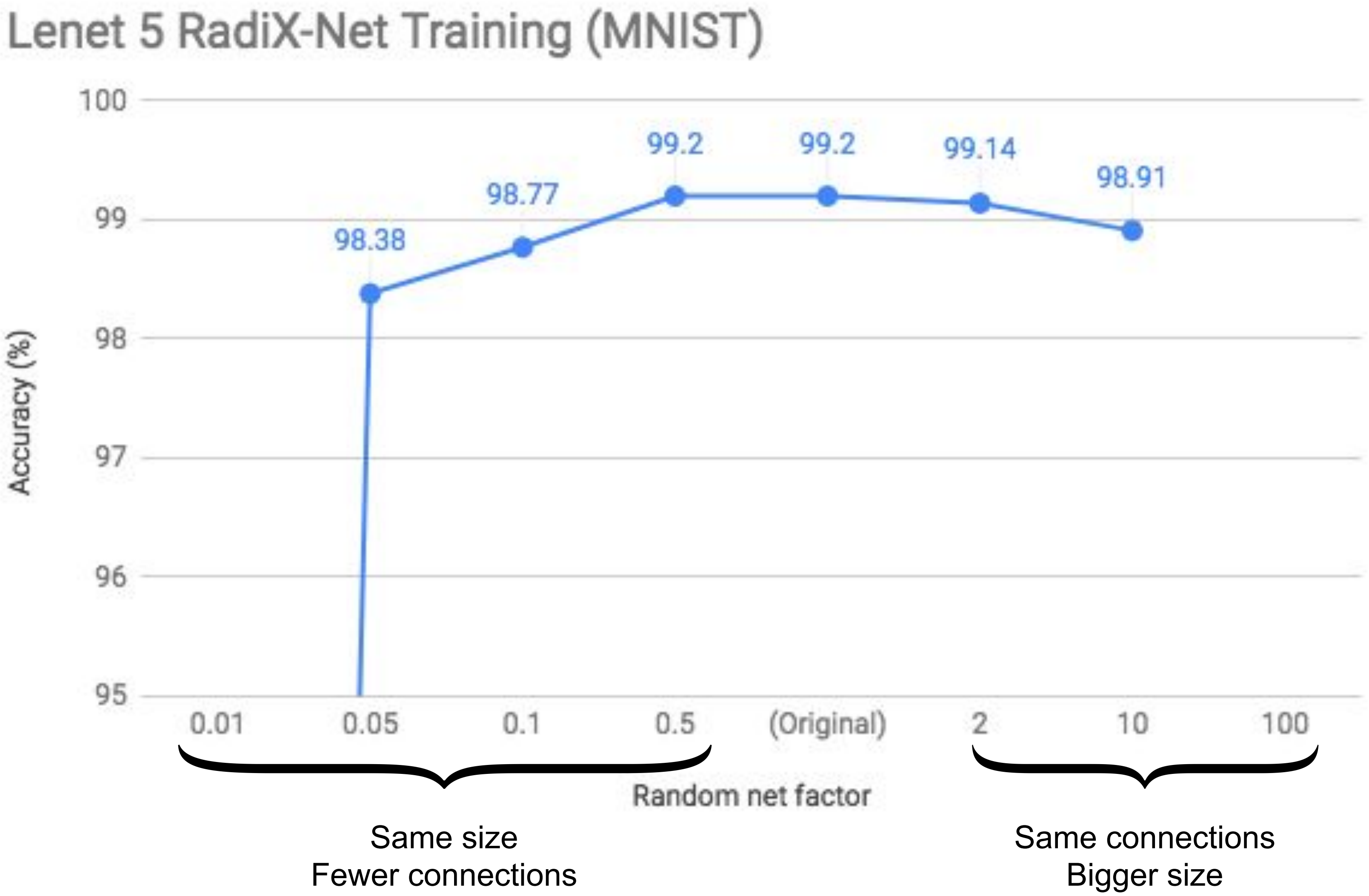
Results: RadiX-Net Training



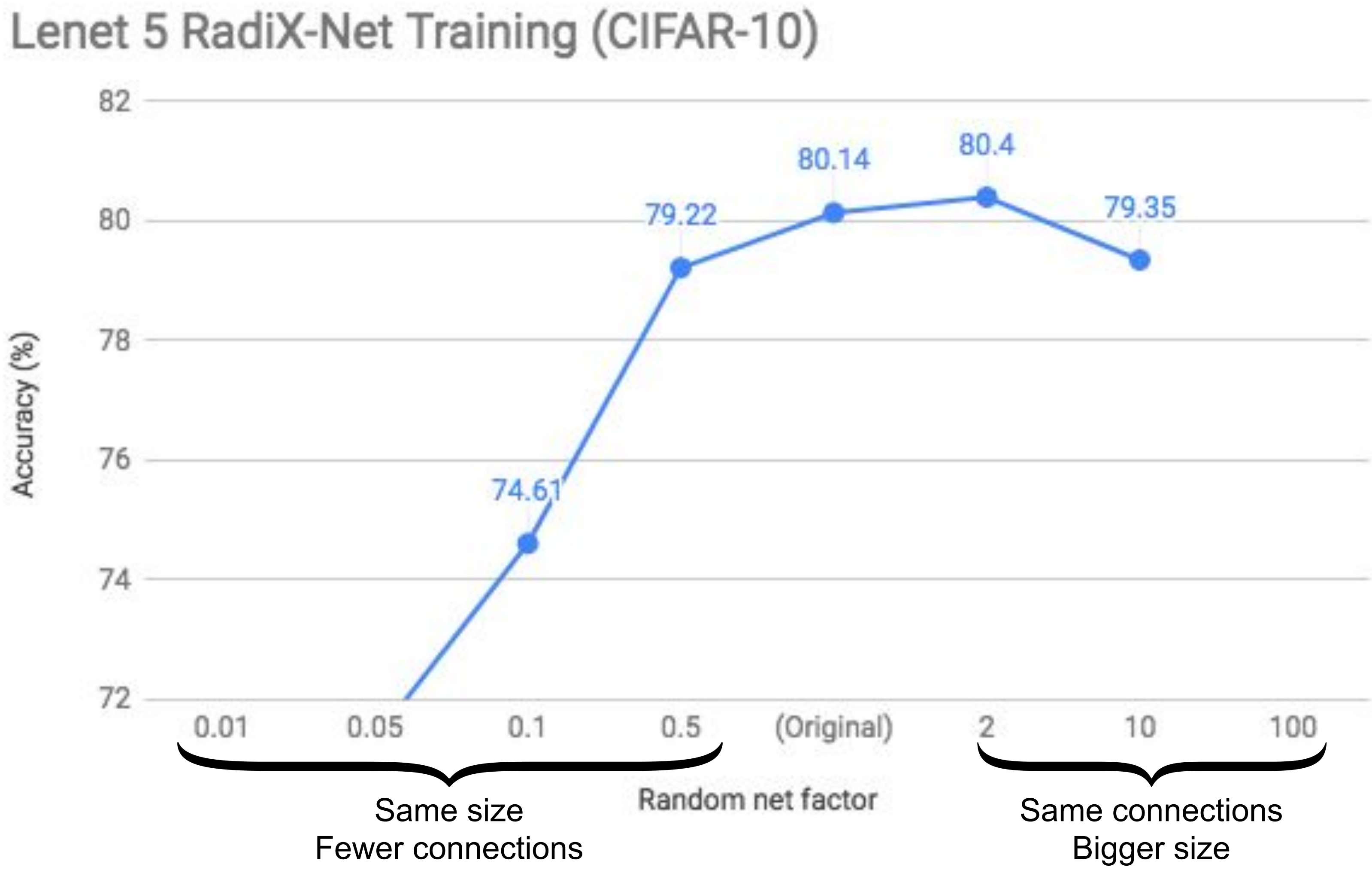
Same size
Fewer connections

Same connections
Bigger size

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
  - With Lenet 5, first and last layer left dense, while with Lenet 300-100 only last layer is dense
- Better performance with lower sparsity
- Retraining on pruning-based sparse network is unreliable
- Pruning-based sparse networks work better with Lenet 300-100 than Lenet 5
  - Pruning convolutional layers (especially first layer) can be dangerous
  - Training on pruning-based sparse networks is less stable, higher learning rate needed
- Random and explicit RadiX-Net layers behave the same
- For both RadiX-Net and pruning-based networks, performance depends on network at hand
- Both could have different performances on larger, more complex networks

Summary, Future Work and Next Steps

- Results for both pruning-based and RadiX-Net sparse networks are mixed
- Need to evaluate performance on larger networks to fully characterize each technique's behavior
- May be insightful to investigate what pruned network structure looks like
- Develop better sparse strategies focusing on convolutional layers, which make up majority of state-of-the-art network computation
- Train on sparse matrix format instead of masking dense matrices → faster models?
- Speed and accuracy can be seen as trade-off between trainability and overparameterization