# Model Compression --the Pruning Approaches

COS598D

Prof. Kai Li
Dr. Xiaoxiao Li
Xl32@Princeton.edu

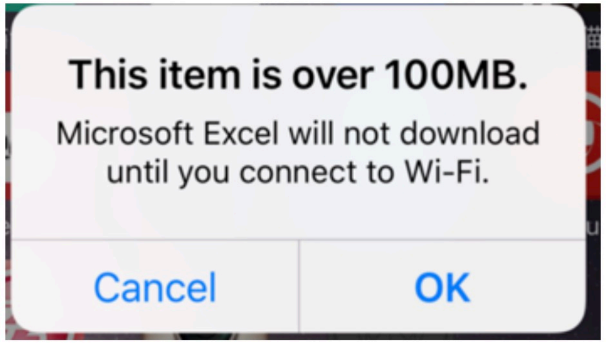
02/18/2020

# Deep Learning on Mobiles


Phones

Drones

Robots




Glasses

Self Driving Cars

**Battery Constrained!**

**App developers suffers from the model size!**



Source: http://isca2016.eecs.umich.edu/wp-content/uploads/2016/07/4A-1.pdf

# The Problem of Running on Cloud

Network Delay

Power Budget

User Privacy

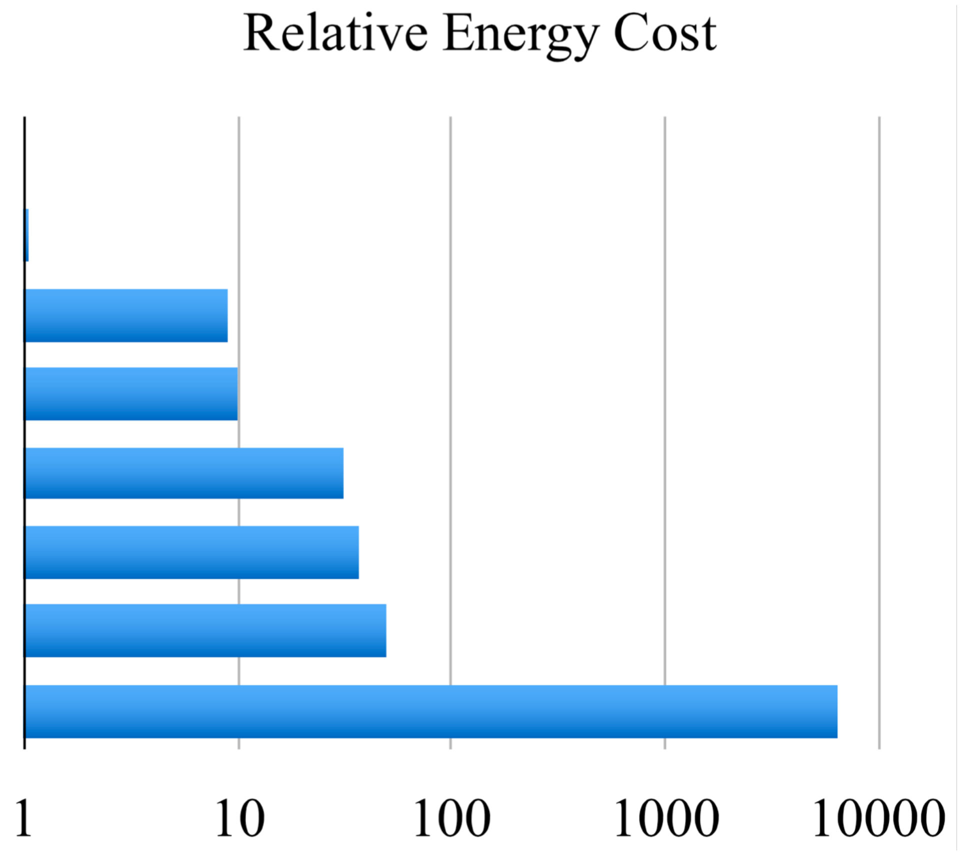
**Intelligent but Inefficient**

https://web.stanford.edu/class/ee380/Abstracts/160106-slides.pdf

# Needs for Model Compression

| Operation | Energy [pJ] | Relative Cost |
|---|---|---|
| 32 bit int ADD | 0.1 | 1 |
| 32 bit float ADD | 0.9 | 9 |
| 32 bit Register File | 1 | 10 |
| 32 bit int MULT | 3.1 | 31 |
| 32 bit float MULT | 3.7 | 37 |
| 32 bit SRAM Cache | 5 | 50 |
| 32 bit DRAM Memory | 640 | 6400 |

Energy table for 45nm CMOS process




Source: http://isca2016.eecs.umich.edu/wp-content/uploads/2016/07/4A-1.pdf

# Deep Compression Can Achieve …

**Smaller Size**
Compress Mobile App
Size by 35x-50x

**Accuracy**
no loss of accuracy
improved accuracy

**Speedup**
make inference faster

# Methods for Model Compression

- **Deep Compression:**
  - ❑Compressing Deep Neural Networks with Pruning
  - ❑Trained Quantization
  - ❑Huffman Coding
  - ❑Matrix Factorization

  * You can combine above methods

- **AutoML** for Model Compression and Acceleration on Mobile Devices

# Pruning: Motivation

| Age | Number of Connections | Stage |
|---|---|---|
| at birth | 50 Trillion | newly formed |
| 1 year old | 1000 Trillion | peak |
| 10 year old | 500 Trillion | pruned and stabilized |

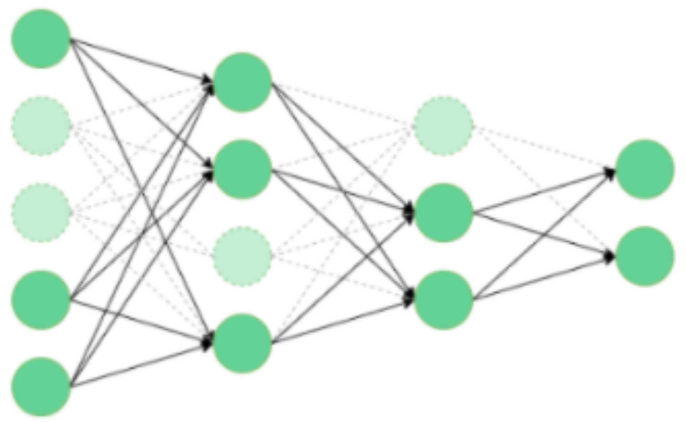
Table 1: The synapses pruning mechanism in human brain development

- Trillion of synapses are generated in the human brain during the first few months of birth.
- **1 year old**, peaked at **1000 trillion**
- Pruning begins to occur.
- **10 years old**, a child has nearly **500 trillion** synapses
- This 'pruning' mechanism removes redundant connections in the brain.

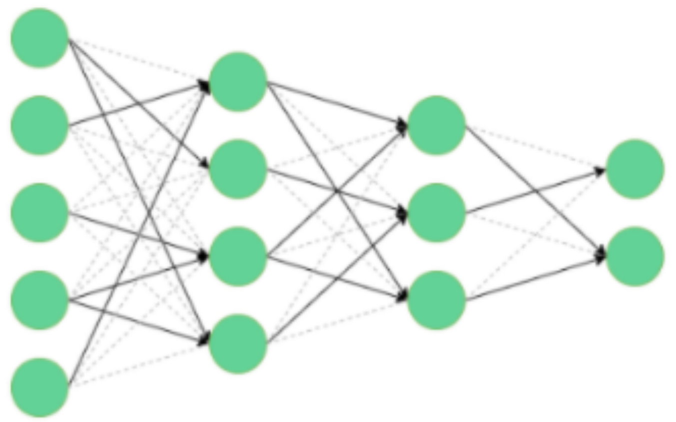
https://web.stanford.edu/class/ee380/Abstracts/160106-slides.pdf

# Structured vs. Unstructured Pruning

- **Structured pruning** involves the selective removal of a larger part of the network such as a layer or a channel.



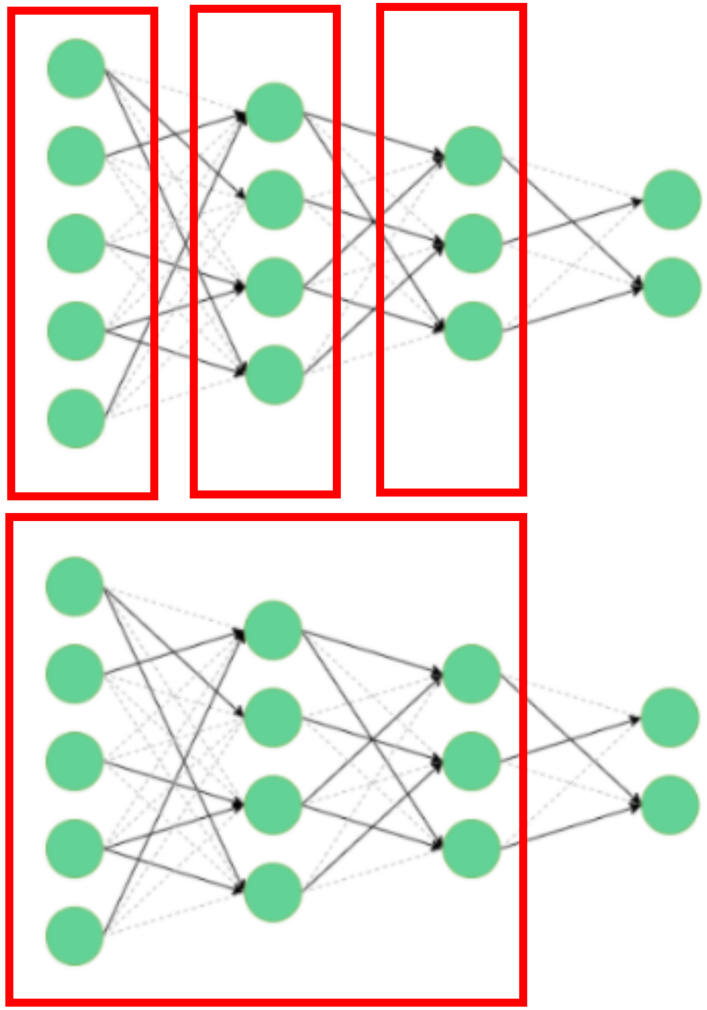
- **Unstructured pruning** find and remove the less salient connection in the model wherever they are. It does not consider any relationship between the pruned weights.



# Local vs. Global Pruning

- **Local pruning** consists of removing a fixed percentage of units/connections from each layer by comparing in the layer.

- **Global pruning** pools all parameters together across layers and selects a global fraction of them to prune.

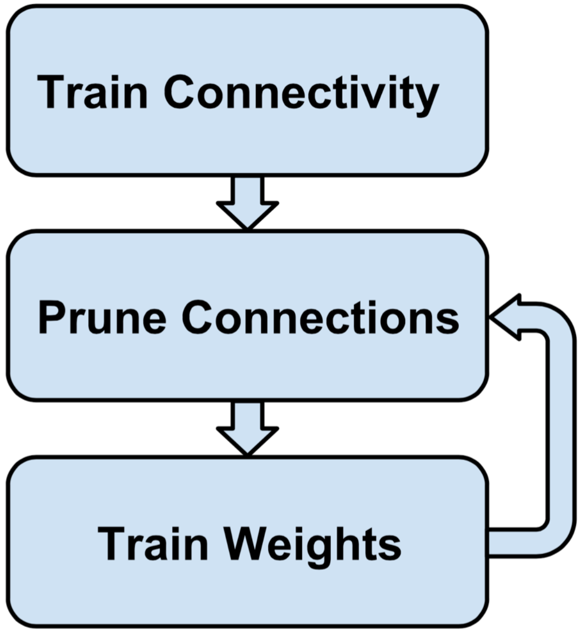


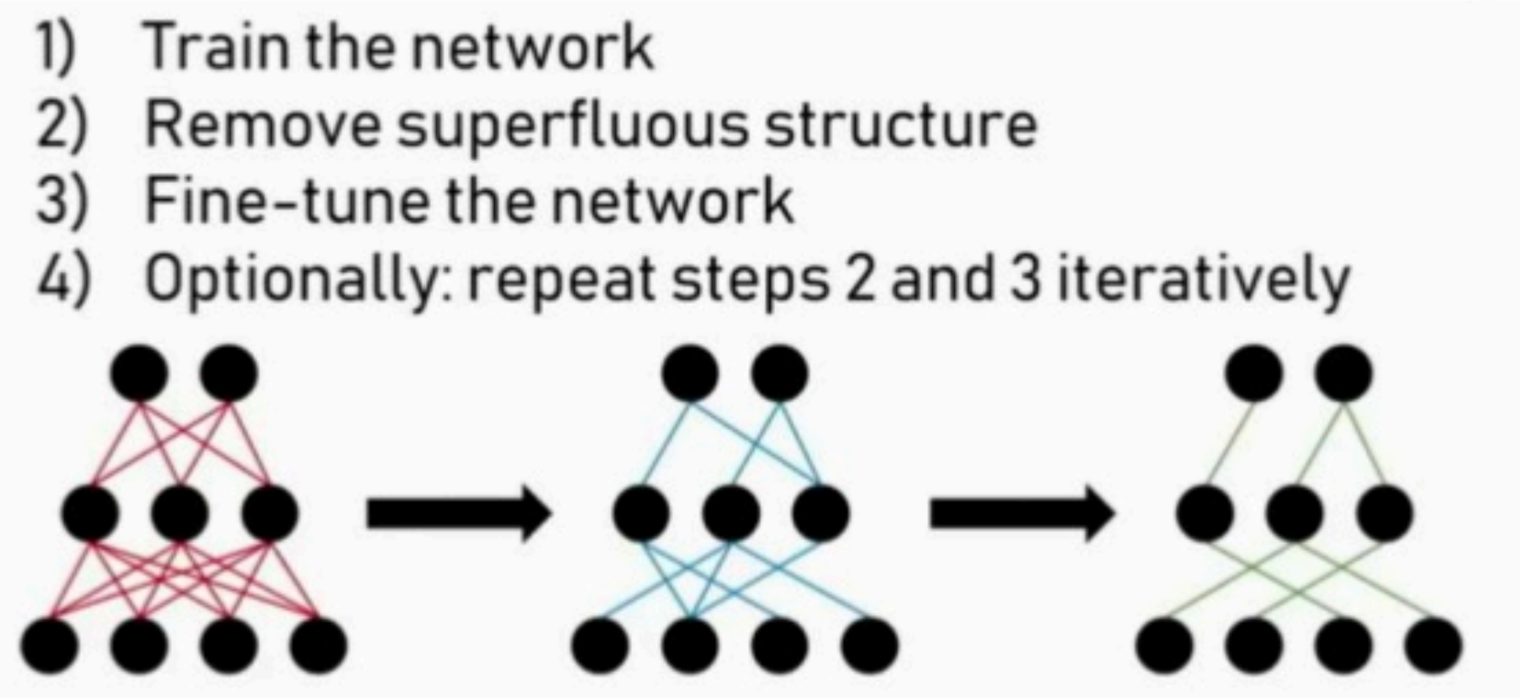
# Pruning Rate and Sparsity

- $p\%$ is the Pruning Rate
- $P_m$ is the Sparsity of the pruned network (mask)

- E.g. $P_m$ = 25% when $p\%$ = 75% of weights are pruned. In this case, compression ratio =1/ $P_m$ = 4.

# Magnitude-based method: Iterative Pruning + Retraining



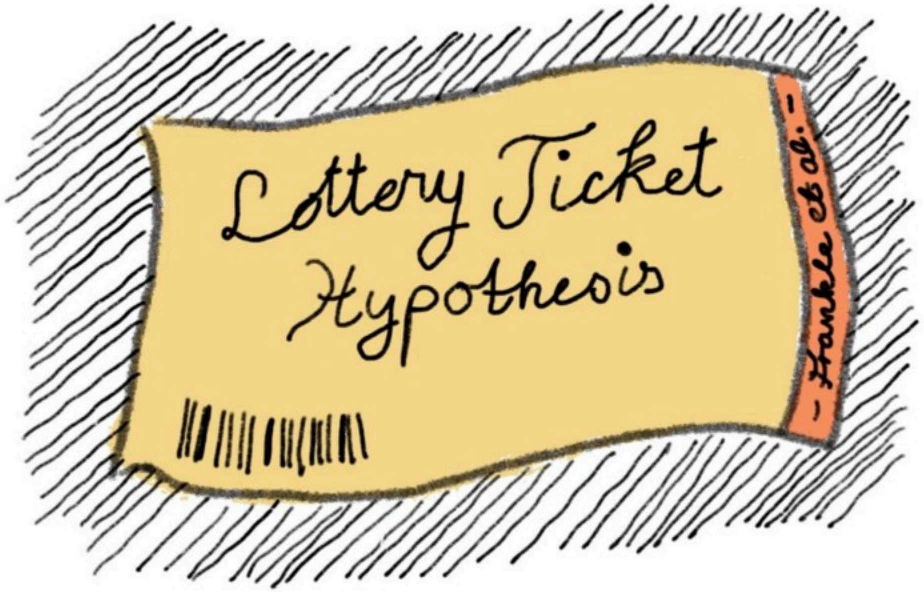

Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

# Magnitude-based method: Iterative Pruning + Retraining (Experiment: Overall)

| Network | Top-1 Error | Top-5 Error | Parameters | Compression Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 267K | 12X |
| LeNet-300-100 Pruned | 1.59% | - | 22K | |
| LeNet-5 Ref | 0.80% | - | 431K | 12X |
| LeNet-5 Pruned | 0.77% | - | 36K | |
| AlexNet Ref | 42.78% | 19.73% | 61M | 9X |
| AlexNet Pruned | 42.77% | 19.67% | 6.7M | |
| VGG-16 Ref | 31.50% | 11.32% | 138M | 13X |
| VGG-16 Pruned | 31.34% | 10.88% | 10.3M | |

Han, Song, et al. "Learning both weights and connections for efficient neural network." NIPS. 2015.

# Lottery Ticket Hypothesis

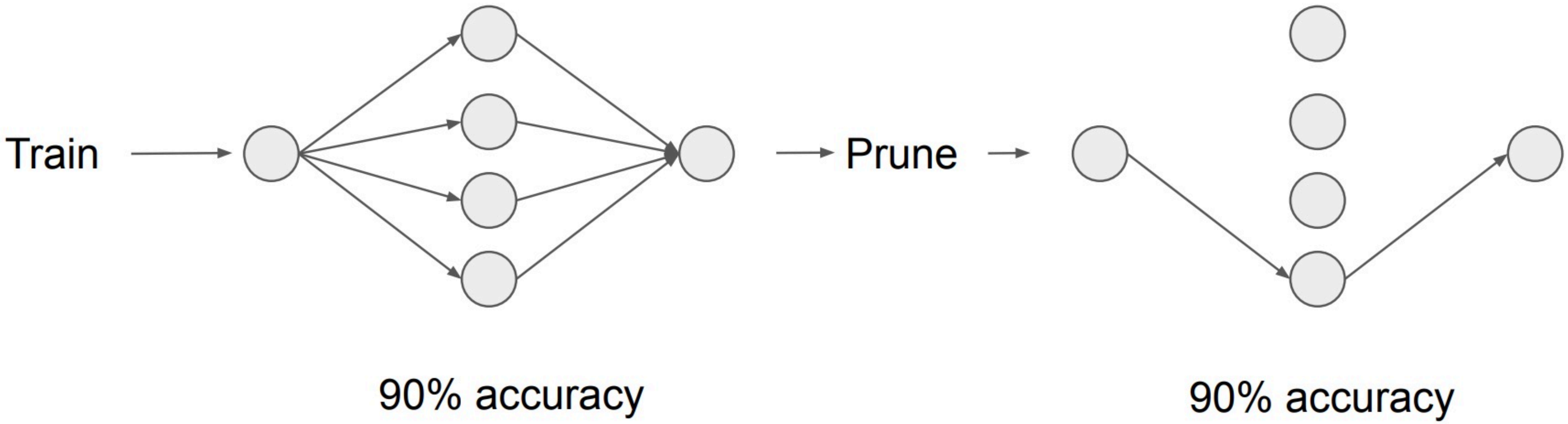



**Jonathan Frankle**
MIT CSAIL
`jfrankle@csail.mit.edu`

**Michael Carbin**
MIT CSAIL
`mcarbin@csail.mit.edu`

# Motivation

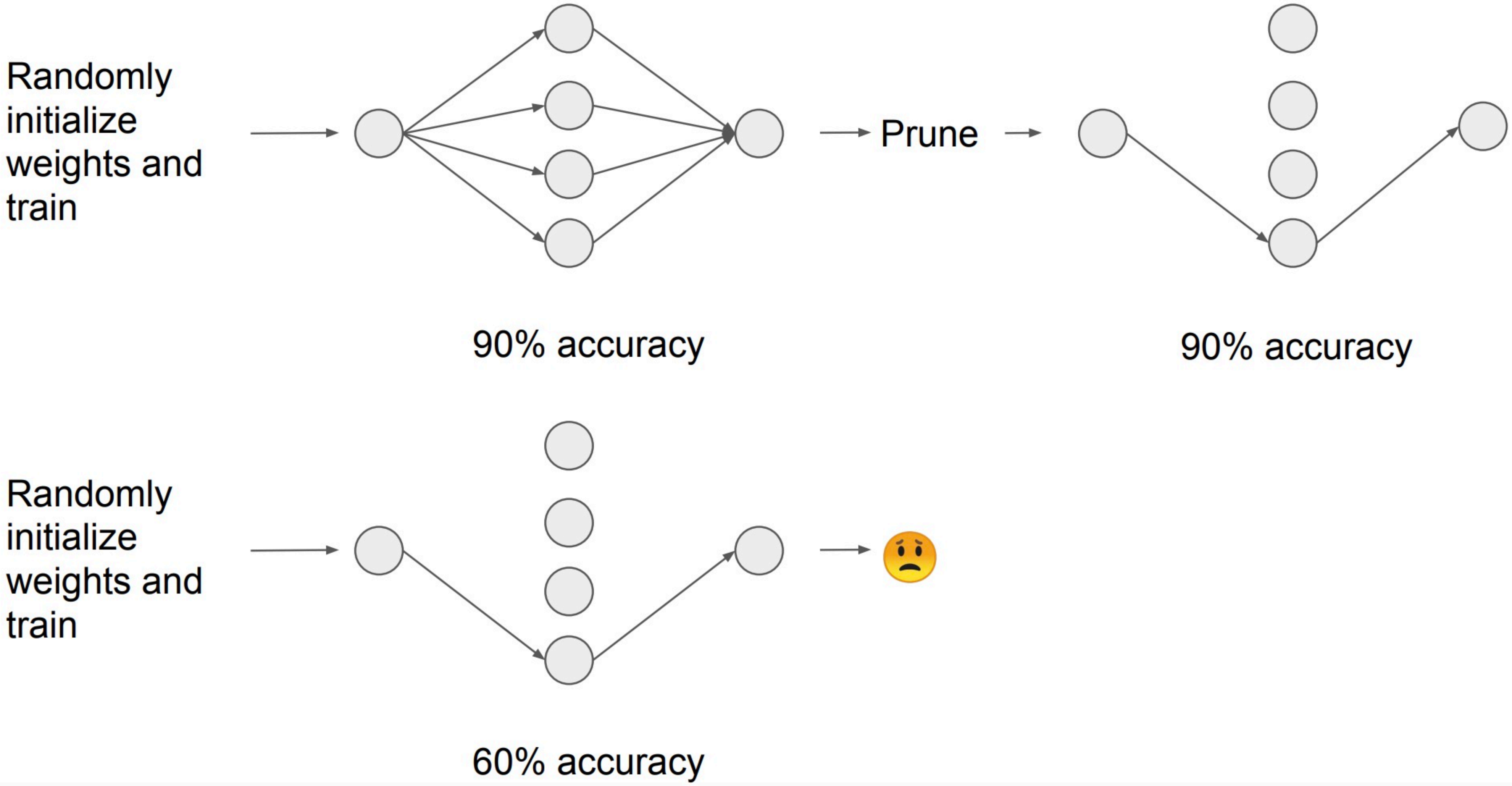
- Pruning techniques can reduce parameter counts by 90% without harming accuracy






https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

# Motivation



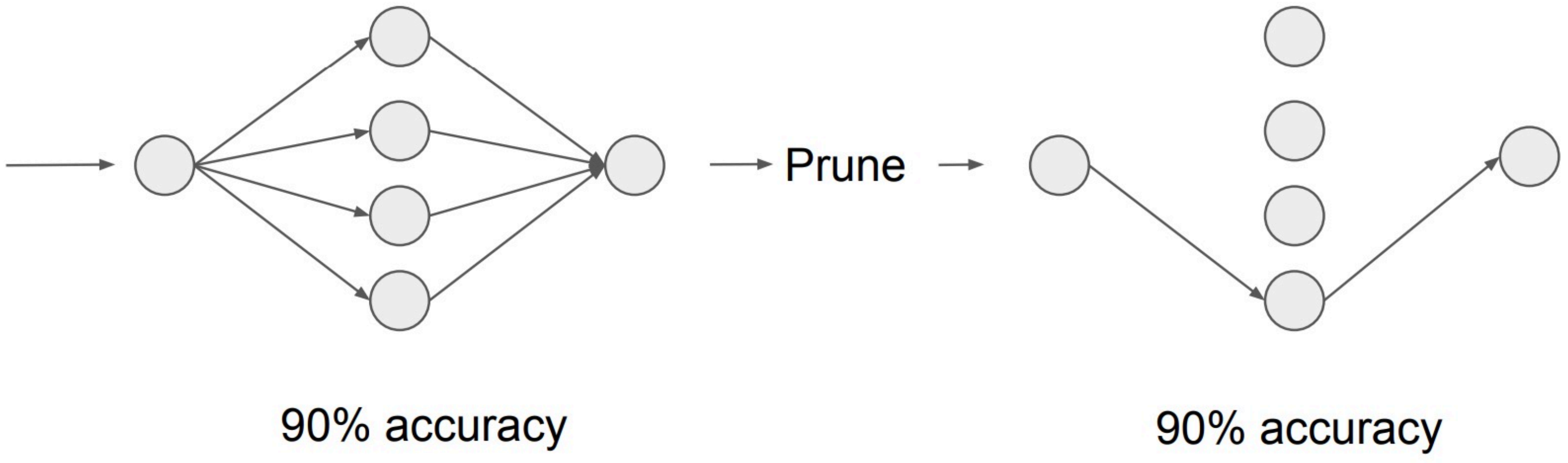

https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

# The Lottery Ticket Hypothesis

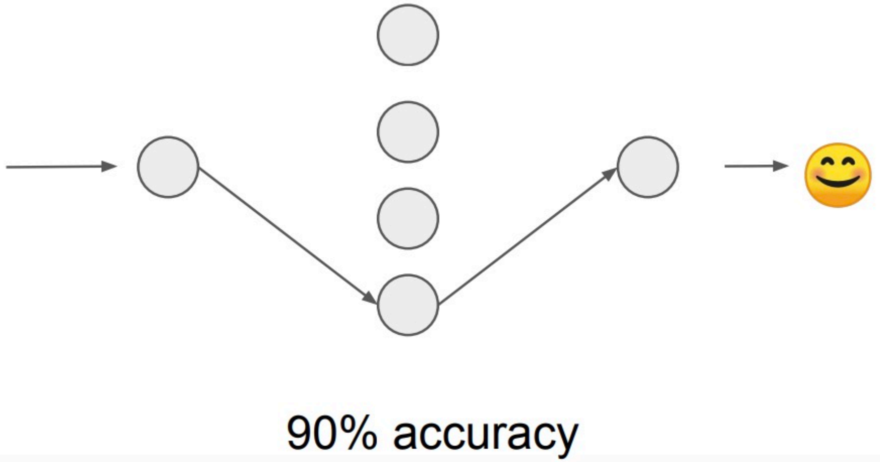
A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

# The Lottery Ticket Hypothesis

Randomly initialize weights and train




Use same weight initialization and train






https://ndey96.github.io/deep-learning-paper-club/slides/Lottery%20Ticket%20Hypothesis%20slides.pdf

# The Lottery Ticket Hypothesis

- If you want to win the lottery, just buy a lot of tickets and some will likely win
- Buying a lot of tickets = having an overparameterized neural network for your task
- Winning the lottery = training a network with high accuracy
- Winning ticket = pruned subnetwork which achieves high accuracy

# Pruning Methods

- One-shot pruning:
  1. Randomly initialize a neural network $f(x; \theta_0)$, with initial parameters $\theta_0$
  2. Train the network for $j$ iterations, arriving at parameters $\theta_j$
  3. Prune $p\%$ of the parameters in $\theta_j$, creating a mask $m$
  4. Reset the remaining parameters to their value in $\theta_0$, creating the winning ticket $f(x; m \odot \theta_0)$.
- Iterative pruning:
  1. Randomly initialize a neural network $f(x; \theta_0)$, with initial parameters $\theta_0$
  2. Train the netowork for $j$ iterations, arriving at parameters $\theta_j$
  3. Prune $p^{1/n}\%$ of the parameters in $\theta_j$, creating a mask $m$
  4. Reset the remaining parameters to their value in $\theta_0$, creating network $f(x; m \odot \theta_0)$
  5. Repeat $n$ times from 2
  6. Final network is a winning ticket $f(x; m \odot \theta_0)$.

# Experiments

- MLP for MNIST
- CNN for CIFAR10
- Ablation Study (dropout, weight decay, optimizer …)

| *Network* | Lenet | Conv-2 | Conv-4 | Conv-6 | Resnet-18 | VGG-19 |
|---|---|---|---|---|---|---|
| *Convolutions* | | 64, 64, pool | 64, 64, pool<br>128, 128, pool | 64, 64, pool<br>128, 128, pool<br>256, 256, pool | 16, 3x[16, 16]<br>3x[32, 32]<br>3x[64, 64] | 2x64 pool 2x128<br>pool, 4x256, pool<br>4x512, pool, 4x512 |
| *FC Layers* | 300, 100, 10 | 256, 256, 10 | 256, 256, 10 | 256, 256, 10 | avg-pool, 10 | avg-pool, 10 |
| *All/Conv Weights* | 266K | 4.3M / 38K | 2.4M / 260K | 1.7M / 1.1M | 274K / 270K | 20.0M |
| *Iterations/Batch* | 50K / 60 | 20K / 60 | 25K / 60 | 30K / 60 | 30K / 128 | 112K / 64 |
| *Optimizer* | Adam 1.2e-3 | Adam 2e-4 | Adam 3e-4 | Adam 3e-4 | ← SGD 0.1-0.01-0.001 Momentum 0.9 → | |
| *Pruning Rate* | fc20% | conv10% fc20% | conv10% fc20% | conv15% fc20% | conv20% fc0% | conv20% fc0% |

# Results – MLP (LeNet)

Iterative Pruning

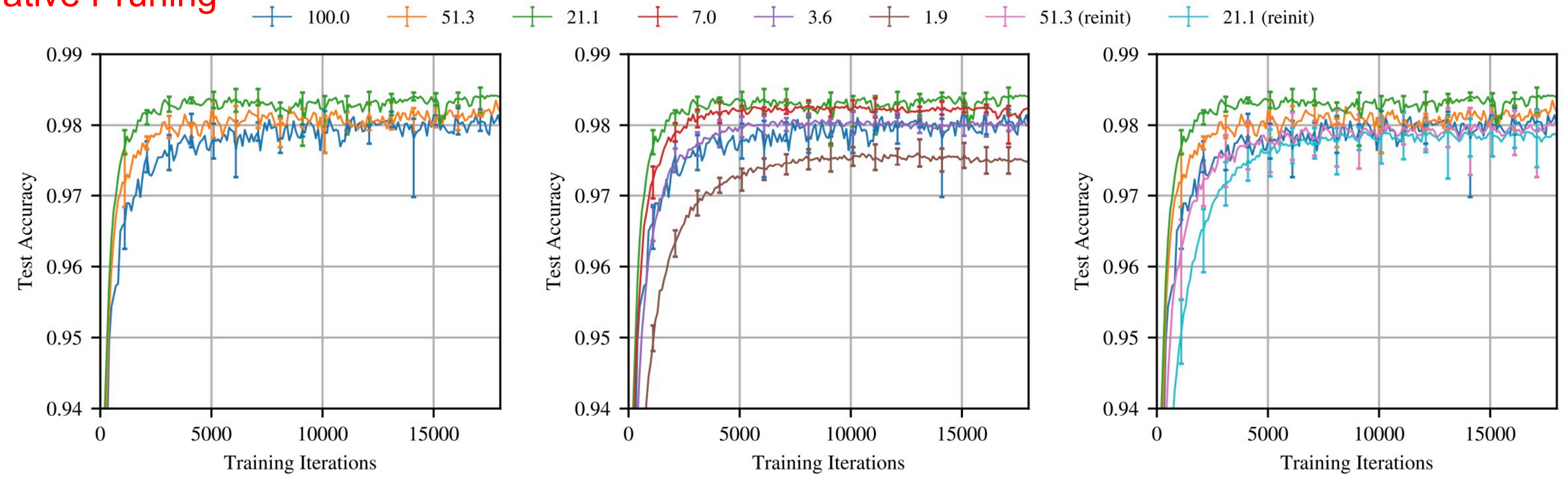



Figure 3: Test accuracy on Lenet (iterative pruning) as training proceeds. Each curve is the average of five trials. Labels are $P_m$—the fraction of weights remaining in the network after pruning. Error bars are the minimum and maximum of any trial.

- 51.3%, 21.12% is better than 100%, 3.6% is comparable with 100%
- Winning ticket is better than reinitialization

## Iterative pruning is better than One-shot

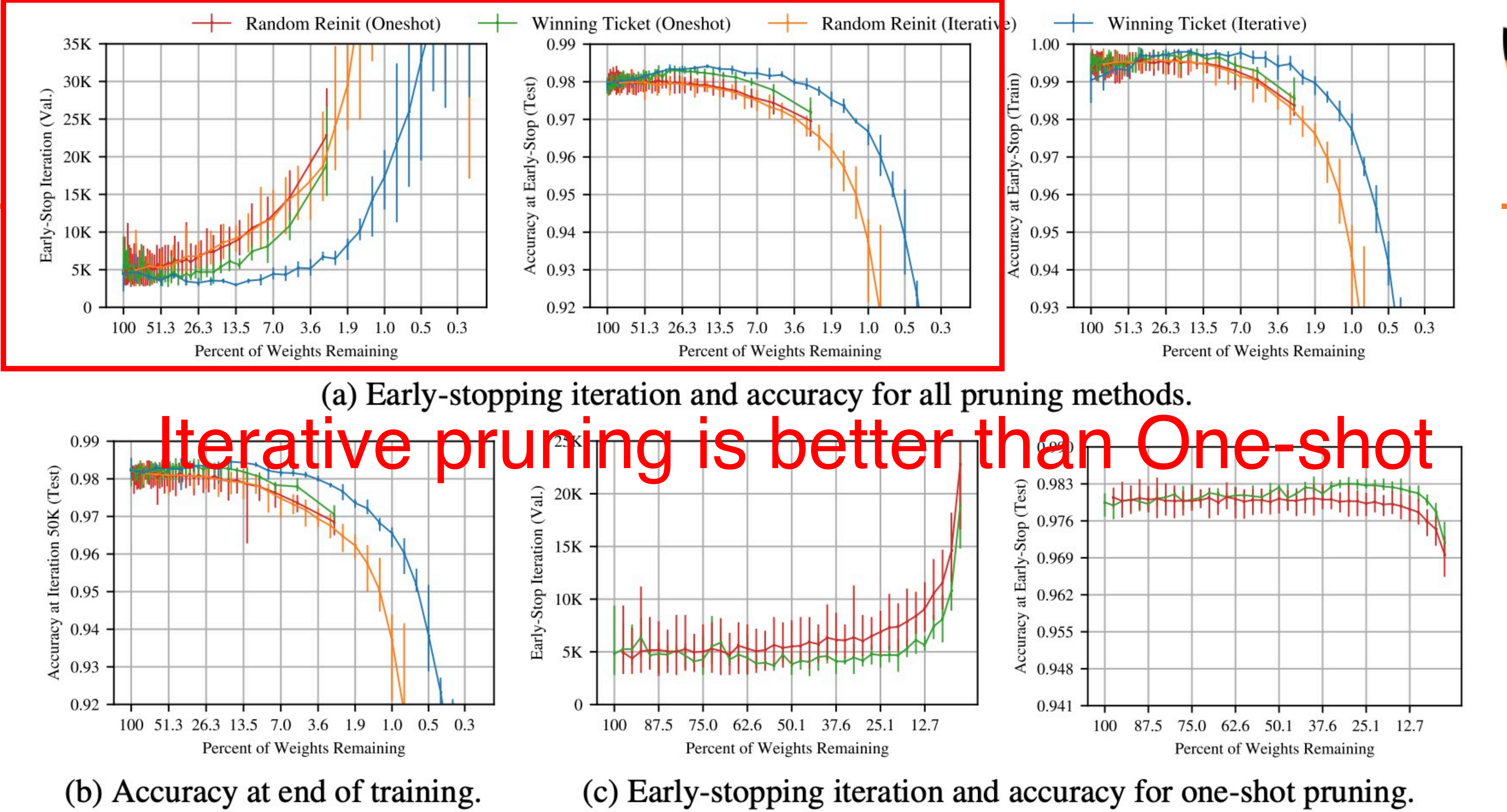



Figure 4: Early-stopping iteration and accuracy of Lenet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. At iteration 50,000, training accuracy $\approx 100\%$ for $P_m \geq 2\%$ for iterative winning tickets (see Appendix D, Figure 12).

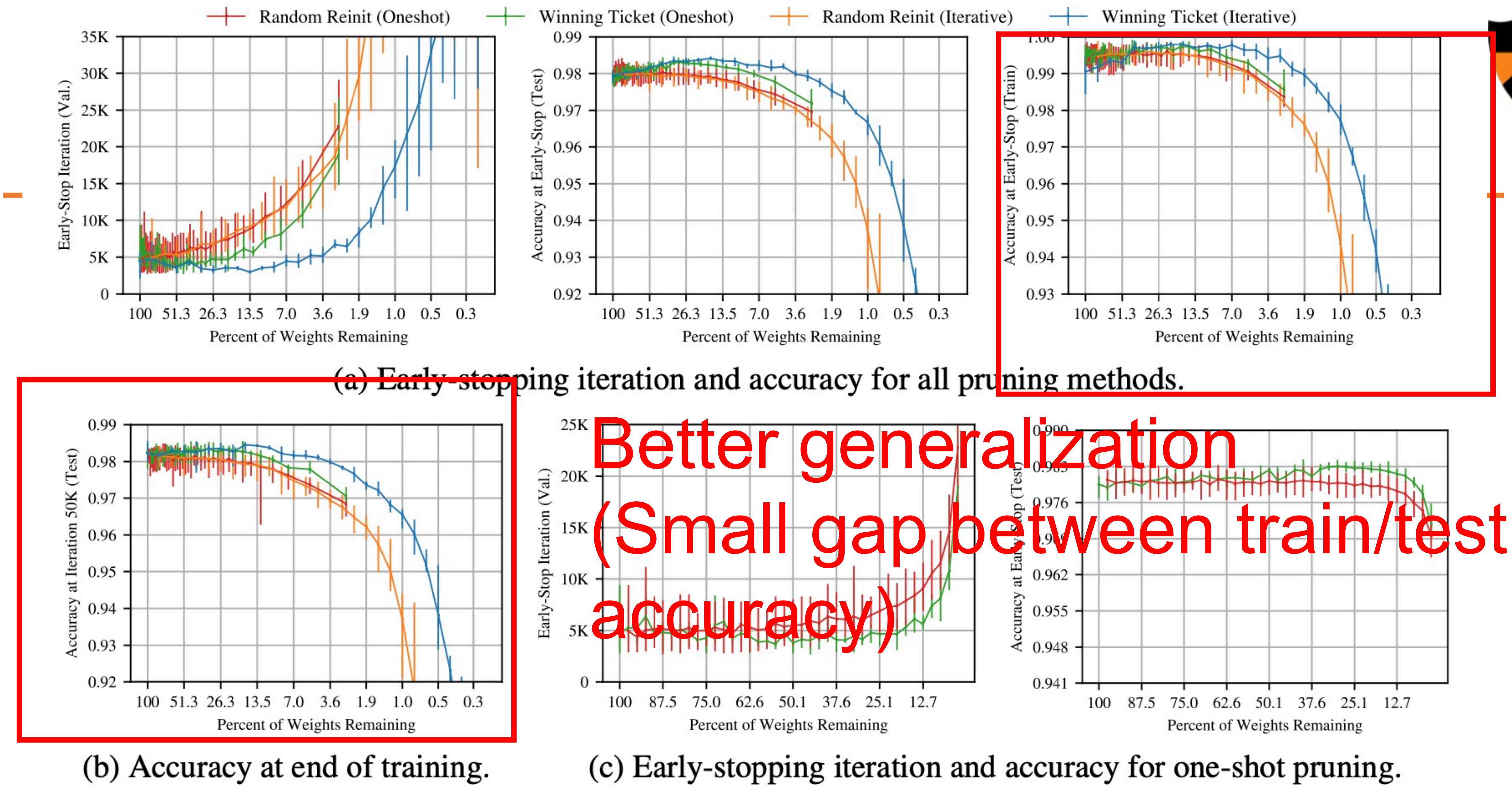



Figure 4: Early-stopping iteration and accuracy of Lenet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. At iteration 50,000, training accuracy ≈ 100% for $P_m \geq 2\%$ for iterative winning tickets (see Appendix D, Figure 12).

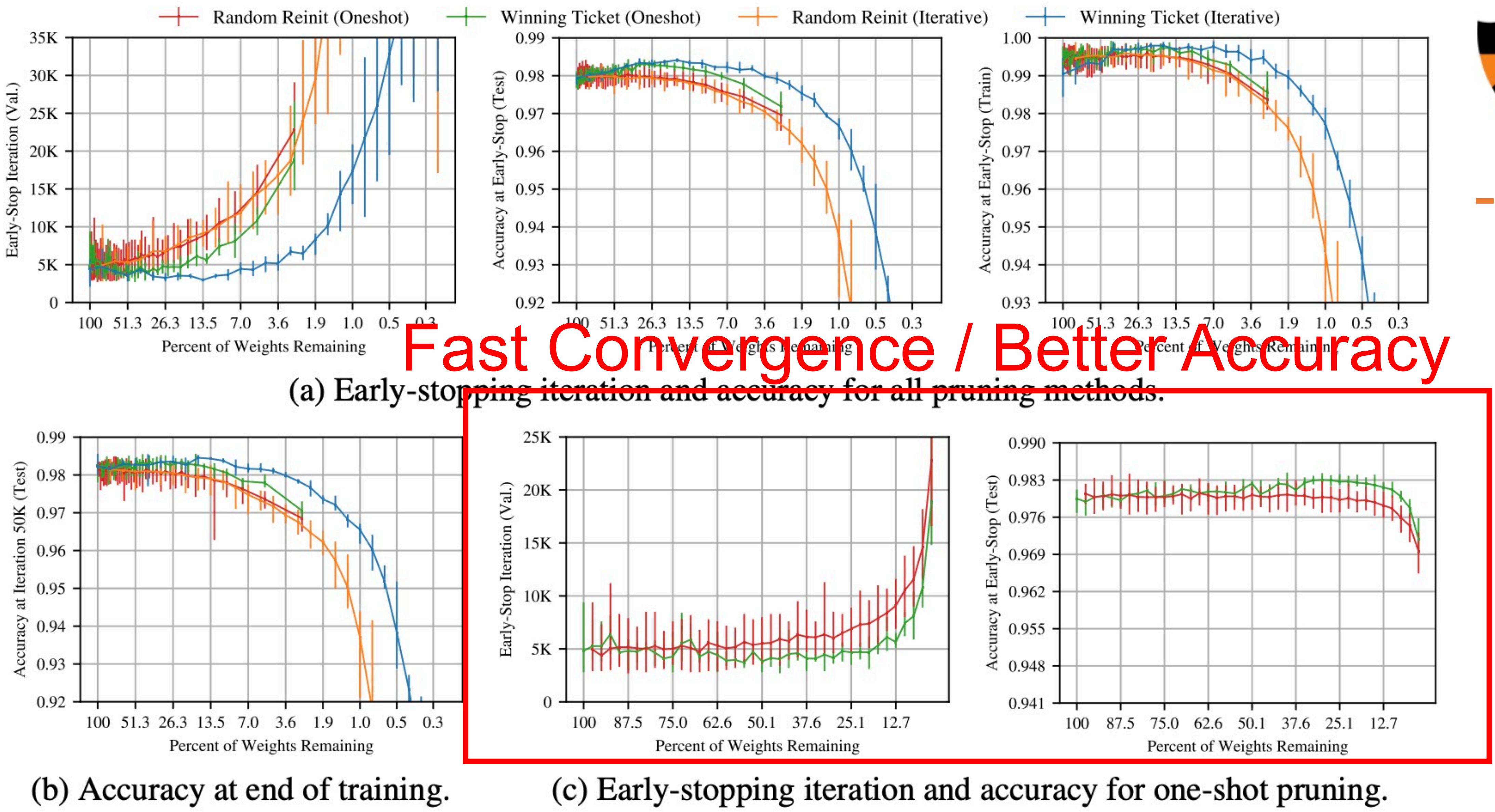



Figure 4: Early-stopping iteration and accuracy of Lenet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. At iteration 50,000, training accuracy $\approx 100\%$ for $P_m \geq 2\%$ for iterative winning tickets (see Appendix D, Figure 12).

# Results - Large CNN (ResNet-18)

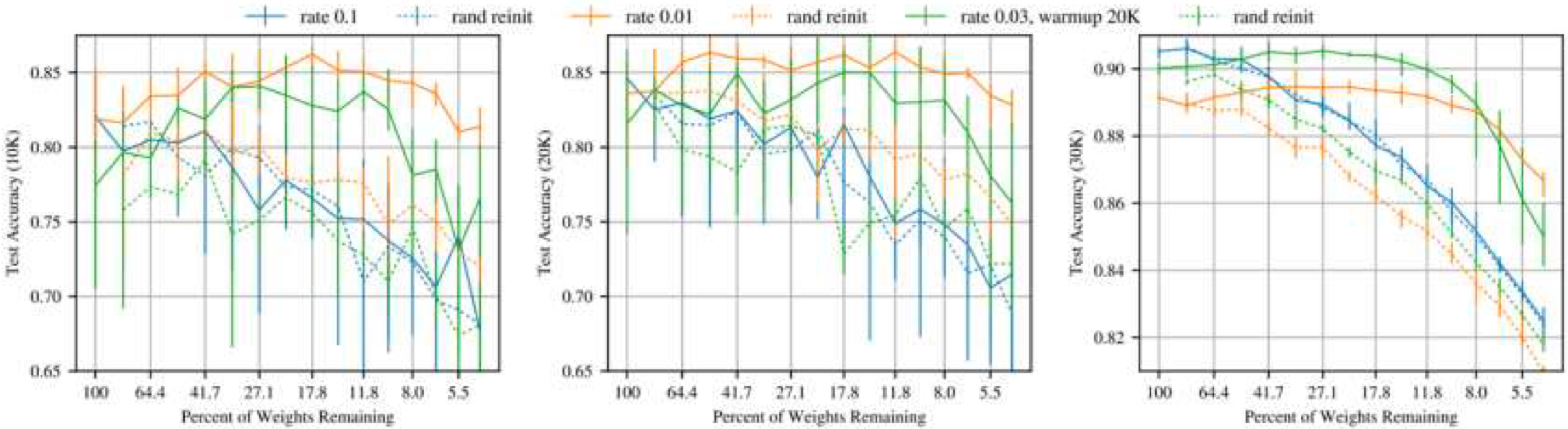


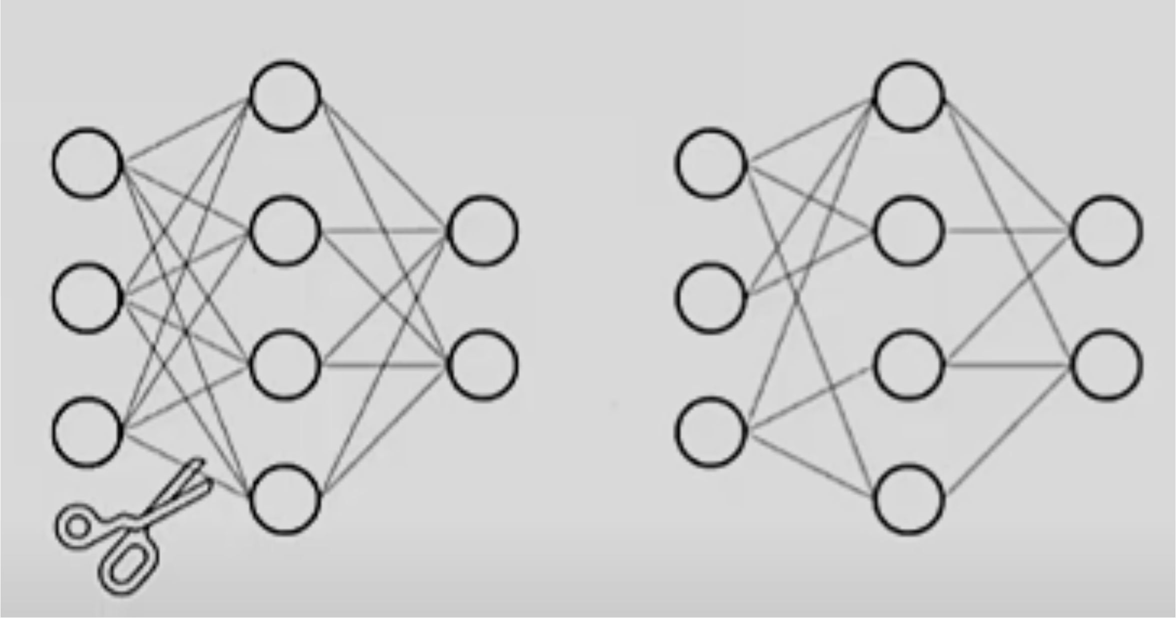

Figure 8: Test accuracy (at 10K, 20K, and 30K iterations) of Resnet-18 when iteratively pruned.

- Use Global Pruning
  - Prune small weights in all layers collectively
- Need LR Warmup
  - 10k iterations

# Limitations

- Only small datasets are tested
- Iterative pruning is computationally intensive (about 15x)
- Structured pruning and non-magnitude pruning methods
- Lack of study about the properties of initalizations
- The reason of why lr warmup is necessary for deeper networks

# SNIP: Single-Shot Network Pruning Based On Connection Sensitivity

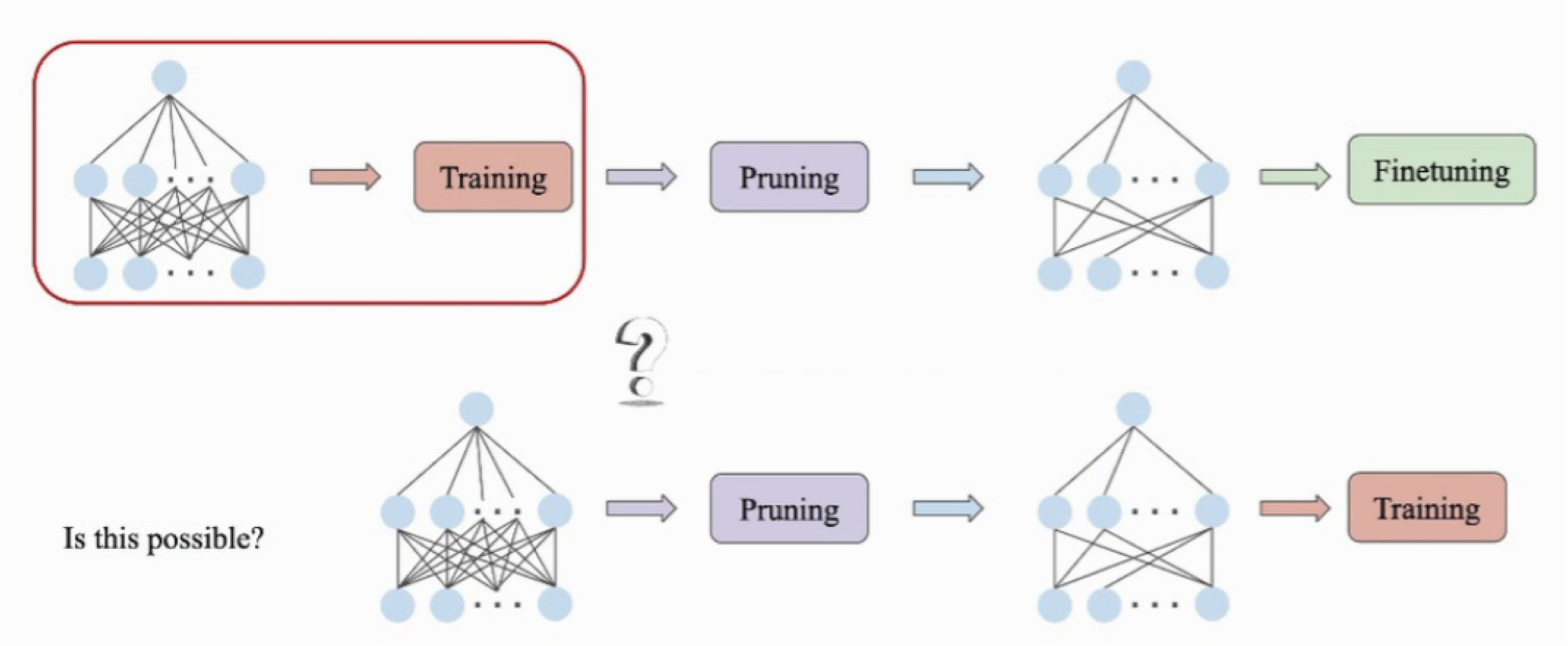


**Namhoon Lee, Thalaiyasingam Ajanthan & Philip H. S. Torr**
University of Oxford

# Motivation

Traditional:

SNIP:




- **Saliency** based methods: selectively removing redundant parameters (or connections) in the neural network

# Methods – Connection Sensitivity

- Definition: **Connection Sensitivity**

❑We hope to prune the neural networks before training

❑Denote $c_j \in \{0,1\}$ as the indicator of connection

$$\begin{aligned}\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c}\odot\mathbf{w};\mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}\ell(\mathbf{c}\odot\mathbf{w};(\mathbf{x}_i,\mathbf{y}_i))\ ,\\ \text{s.t.} \quad \mathbf{w}\in\mathbb{R}^m\ ,\\ \mathbf{c}\in\{0,1\}^m, \quad \|\mathbf{c}\|_0\le\kappa\ ,\end{aligned}$$

# Methods – Connection Sensitivity

- By focusing on the difference in the objective function when $c_j$ switches, we can determine the importance of the connection.
- Connection j is active -- $c_j$ = 1; is pruned -- $c_j$ = 0
- Precisely, the effect can be measured by

$$\Delta L_j(\mathbf{w}; \mathcal{D}) = L(\mathbf{1} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{1} - \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})$$

* $\boldsymbol{e}_j$ is the vector zeros everywhere except at the index *j* where it is one.

# Methods – Connection Sensitivity

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) ,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m ,$$
$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa ,$$

- Since $c_j$ is binary, it cannot be differentiated and is difficult to optimize, we will actually solve the problem by relaxing it.

$$\boxed{\Delta L_j(\mathbf{w}; \mathcal{D}) \approx g_j(\mathbf{w}; \mathcal{D}) = \left.\frac{\partial L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})}{\partial c_j}\right|_{\mathbf{c}=\mathbf{1}} = \lim_{\delta \to 0} \left.\frac{L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) - L((\mathbf{c} - \delta\, \mathbf{e}_j) \odot \mathbf{w}; \mathcal{D})}{\delta}\right|_{\mathbf{c}=\mathbf{1}}}$$

# Methods – Connection Sensitivity

$$\Delta L_j(\mathbf{w};\mathcal{D}) \approx g_j(\mathbf{w};\mathcal{D}) = \left.\frac{\partial L(\mathbf{c}\odot\mathbf{w};\mathcal{D})}{\partial c_j}\right|_{\mathbf{c}=\mathbf{1}}$$

- The hypothesis [**Connection Sensitivity**] is that
  - If the gradient of $g_j$ is large, it should be an important connection for the network and the task.

$$\boxed{s_j = \frac{|g_j(\mathbf{w};\mathcal{D})|}{\sum_{k=1}^{m}|g_k(\mathbf{w};\mathcal{D})|}}$$

  - Once the sensitivity is computed, only the top-κ connections are retained.

$$\boxed{c_j = \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0]\,, \quad \forall j \in \{1\dots m\}}$$

# SNIP Algorithm at Initialization

- 1. extract mini-batch data
- 2. calculate gradient for c
- 3. sort in descending order
- 4. pruning if less than $\kappa_{th}$ score

**Algorithm 1** SNIP: Single-shot Network Pruning based on Connection Sensitivity

**Require:** Loss function $L$, training dataset $\mathcal{D}$, sparsity level $\kappa$ ▷ Refer Equation 3

**Ensure:** $\|\mathbf{w}^*\|_0 \leq \kappa$

1: $\mathbf{w} \leftarrow$ VarianceScalingInitialization ▷ Refer Section 4.2

2: $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^b \sim \mathcal{D}$ ▷ Sample a mini-batch of training data

3: $s_j \leftarrow \frac{|g_j(\mathbf{w};\mathcal{D}^b)|}{\sum_{k=1}^m |g_k(\mathbf{w};\mathcal{D}^b)|}, \quad \forall j \in \{1 \dots m\}$ ▷ Connection sensitivity

4: $\tilde{\mathbf{s}} \leftarrow \text{SortDescending}(\mathbf{s})$

5: $c_j \leftarrow \mathbb{1}[s_j - \tilde{s}_\kappa \geq 0], \quad \forall j \in \{1 \dots m\}$ ▷ Pruning: choose top-$\kappa$ connections

6: $\mathbf{w}^* \leftarrow \arg\min_{\mathbf{w} \in \mathbb{R}^m} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D})$ ▷ Regular training

7: $\mathbf{w}^* \leftarrow \mathbf{c} \odot \mathbf{w}^*$

# Robustness for Network Architecture

- The initial values of the weights of a neural network are usually randomly initialized using a normal distribution

- If the initial weights have a fixed variance, the signal passing through each layer no longer guarantees to have the same variance

- To avoid this, it is recommended to use the variance scaling method for initialization (i.e. Xavier initialization).

https://www.deeplearning.ai/ai-notes/initialization/

# Robustness for Mini-batch

- The results of the proposed method depend on the data contained in the mini-batch.

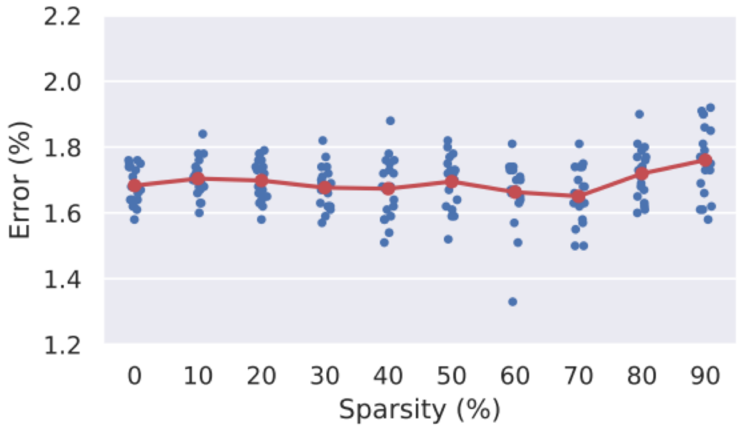
- SNIP determines the final pruning target after accumulating the importance of connections across multiple batches.

# Experiments and Results

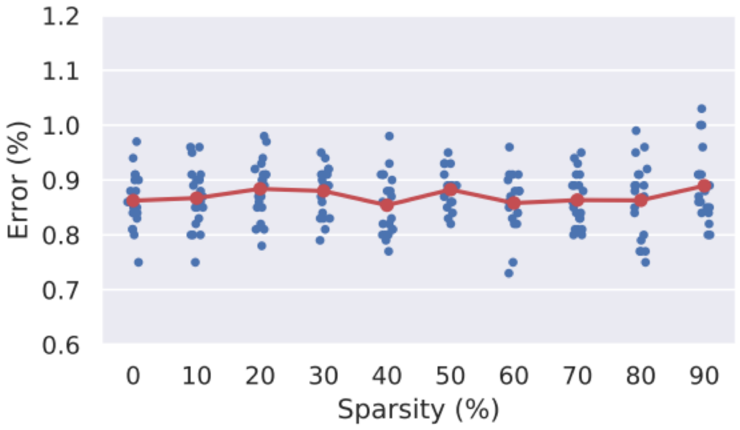
- Pruning LeNet at multiple sparsity levels

- Significantly sparse network with little loss of classification accuracy

- Generalization performance better than original network depending on sparsity level in some cases




(a) LeNet-300-100



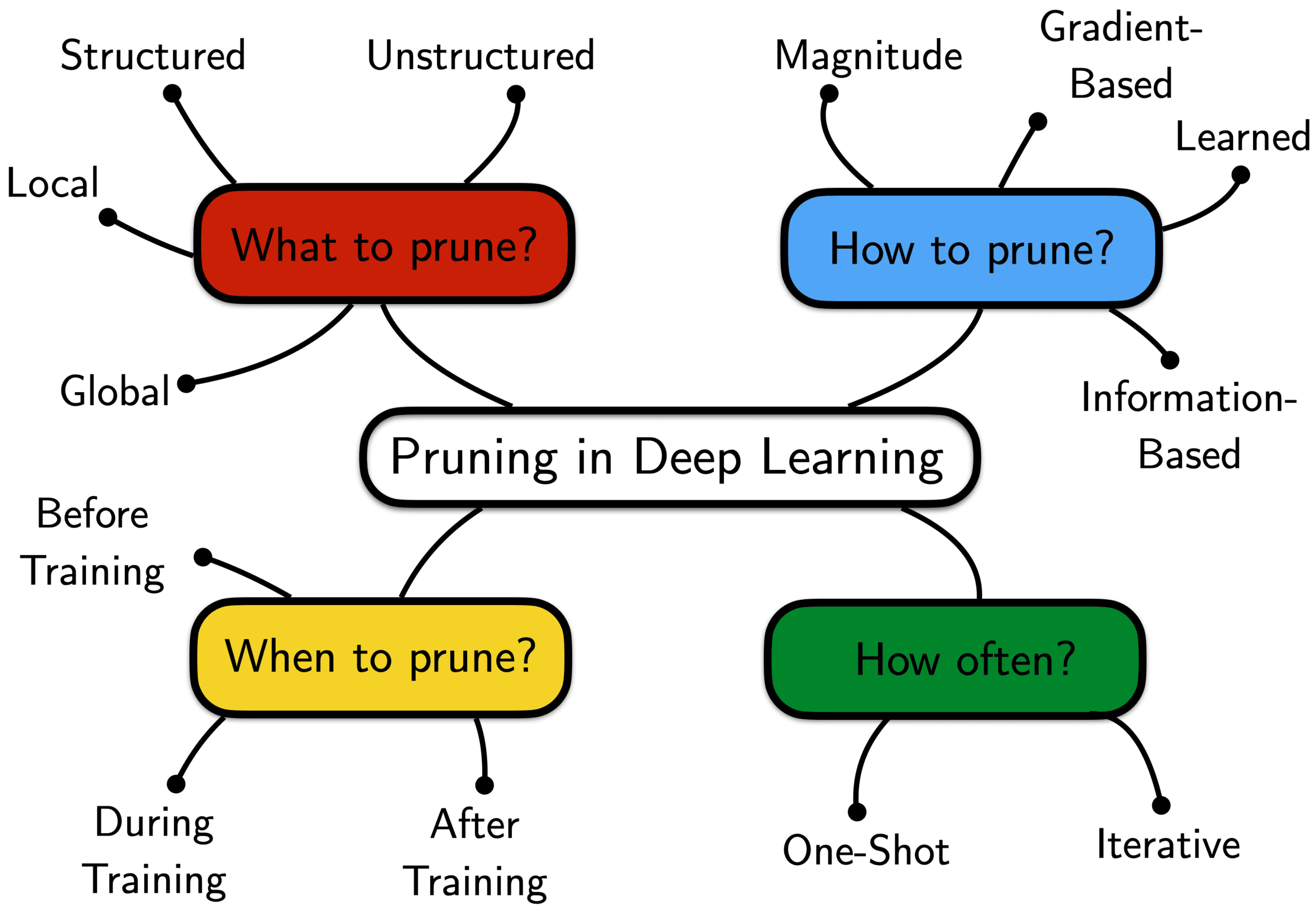

(b) LeNet-5-Caffe

# Comparison with Baselines

- Comparison experiments with existing pruning algorithms
- There are 7 comparison methods.
- The proposed method gives good results in spite of working with single-shot

| Method | Criterion | LeNet-300-100 $\bar{\kappa}$ (%) | LeNet-300-100 err. (%) | LeNet-5-Caffe $\bar{\kappa}$ (%) | LeNet-5-Caffe err. (%) | Pretrain | # Prune | Additional hyperparam. | Augment objective | Arch. constraints |
|---|---|---|---|---|---|---|---|---|---|---|
| Ref. | – | – | 1.7 | – | 0.9 | – | – | – | – | – |
| LWC | Magnitude | 91.7 | **1.6** | 91.7 | **0.8** | ✓ | many | ✓ | ✗ | ✓ |
| DNS | Magnitude | 98.2 | 2.0 | 99.1 | 0.9 | ✓ | many | ✓ | ✗ | ✓ |
| LC | Magnitude | 99.0 | 3.2 | 99.0 | 1.1 | ✓ | many | ✓ | ✓ | ✗ |
| SWS | Bayesian | 95.6 | 1.9 | 99.5 | 1.0 | ✓ | soft | ✓ | ✓ | ✗ |
| SVD | Bayesian | 98.5 | 1.9 | 99.6 | **0.8** | ✓ | soft | ✓ | ✓ | ✗ |
| OBD | Hessian | 92.0 | 2.0 | 92.0 | 2.7 | ✓ | many | ✓ | ✗ | ✗ |
| L-OBS | Hessian | 98.5 | 2.0 | 99.0 | 2.1 | ✓ | many | ✓ | ✗ | ✓ |
| SNIP (ours) | Connection sensitivity | 95.0<br>98.0 | **1.6**<br>2.4 | 98.0<br>99.0 | **0.8**<br>1.1 | ✗ | 1 | ✗ | ✗ | ✗ |

# Summary




https://towardsdatascience.com/the-lottery-ticket-hypothesis-a-survey-d1f0f62f8884

# Summary

- Neural Networks Pruning is essential for AI deployment on mobiles.
- We studied the basic magnitude-based pruning method, using pruning to find 'winning ticket' in the densely connected neural networks and SNIP – pruning at initialization.
- More recent works improve the pruning methods, thus achieve better sparsity, efficiency and accuracy trade-off.

- Submit Warmup
- Submit short paper review (ddl next Tuesday)
- Choose one model compression assignment: Assignment 1-Pruning or Assignment 2 - Binarized NN