Neural Network Pruning: from over-parametrized to under-parametrized networks

Michela Paganini
Facebook AI Research
4th IML Workshop
October 21, 2020

FACEBOOK

Who am I

B.A. in Physics, B.A. in Astrophysics,
U.C. Berkeley

Ph.D. in Physics, Yale University

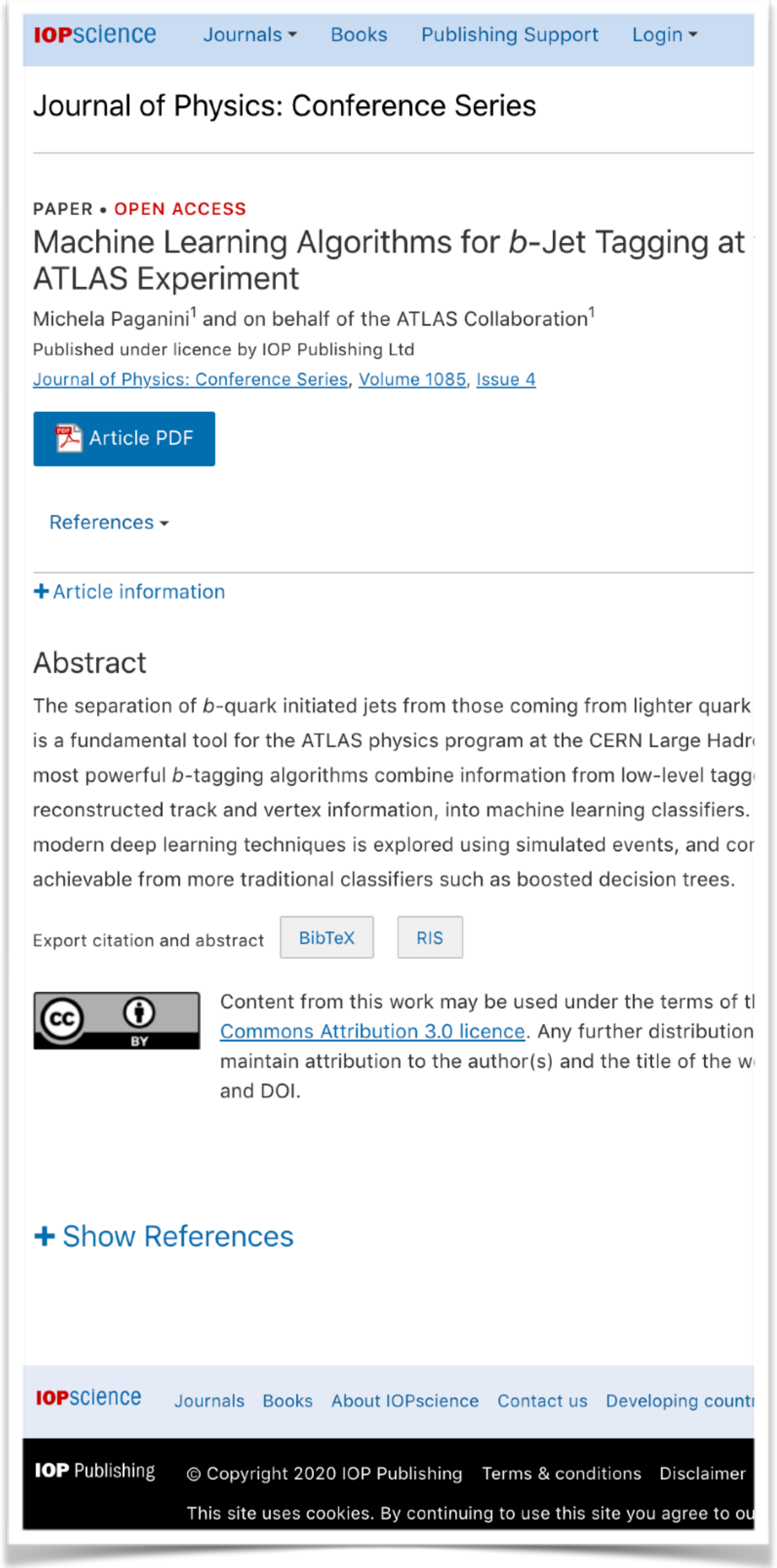
Thesis: *Machine Learning Solutions for High Energy Physics: Applications to Electromagnetic Shower Generation, Flavor Tagging, and the Search for di-Higgs Production* [arXiv:1903.05082]

Former Member,
ATLAS Collaboration, CERN

Postdoctoral Researcher,
Facebook AI Research

Visiting Affiliate,
NERSC

AI for Science



IOPscience Journals Books Publishing Support Login

Journal of Physics: Conference Series

PAPER • OPEN ACCESS

Machine Learning Algorithms for *b*-Jet Tagging at ATLAS Experiment

Michela Paganini[1] and on behalf of the ATLAS Collaboration[1]

Published under licence by IOP Publishing Ltd

Journal of Physics: Conference Series, Volume 1085, Issue 4

Article PDF

References

+ Article information

Abstract

The separation of *b*-quark initiated jets from those coming from lighter quark
is a fundamental tool for the ATLAS physics program at the CERN Large Hadr
most powerful *b*-tagging algorithms combine information from low-level tagg
reconstructed track and vertex information, into machine learning classifiers.
modern deep learning techniques is explored using simulated events, and cor
achievable from more traditional classifiers such as boosted decision trees.

Export citation and abstract BibTeX RIS

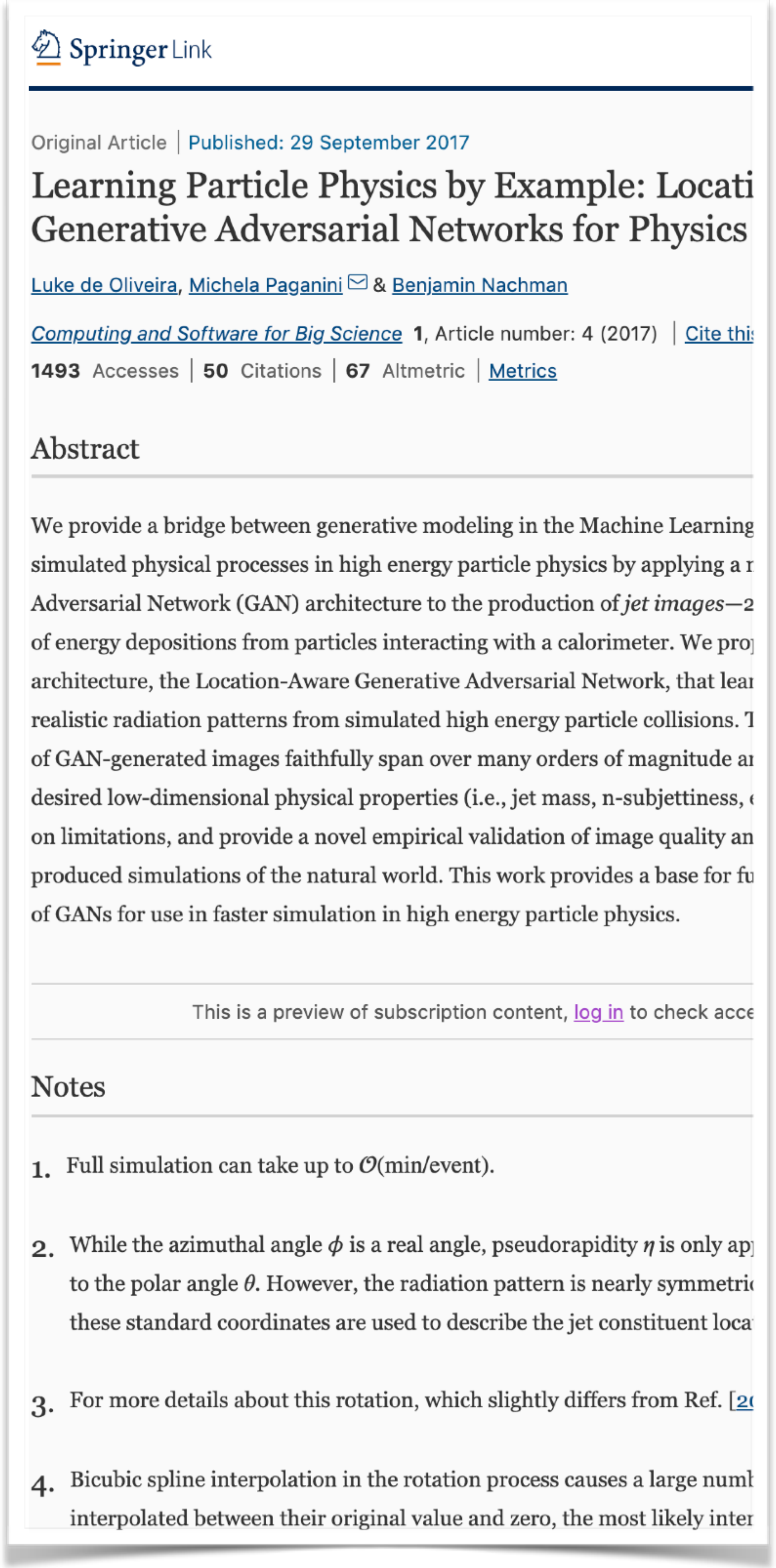
Content from this work may be used under the terms of tl
Commons Attribution 3.0 licence. Any further distribution
maintain attribution to the author(s) and the title of the w
and DOI.

+ Show References

IOPscience Journals Books About IOPscience Contact us Developing count

IOP Publishing © Copyright 2020 IOP Publishing Terms & conditions Disclaimer

This site uses cookies. By continuing to use this site you agree to ou



SpringerLink

Original Article | Published: 29 September 2017

Learning Particle Physics by Example: Locati
Generative Adversarial Networks for Physics

Luke de Oliveira, Michela Paganini & Benjamin Nachman

Computing and Software for Big Science 1, Article number: 4 (2017) | Cite thi

1493 Accesses | 50 Citations | 67 Altmetric | Metrics

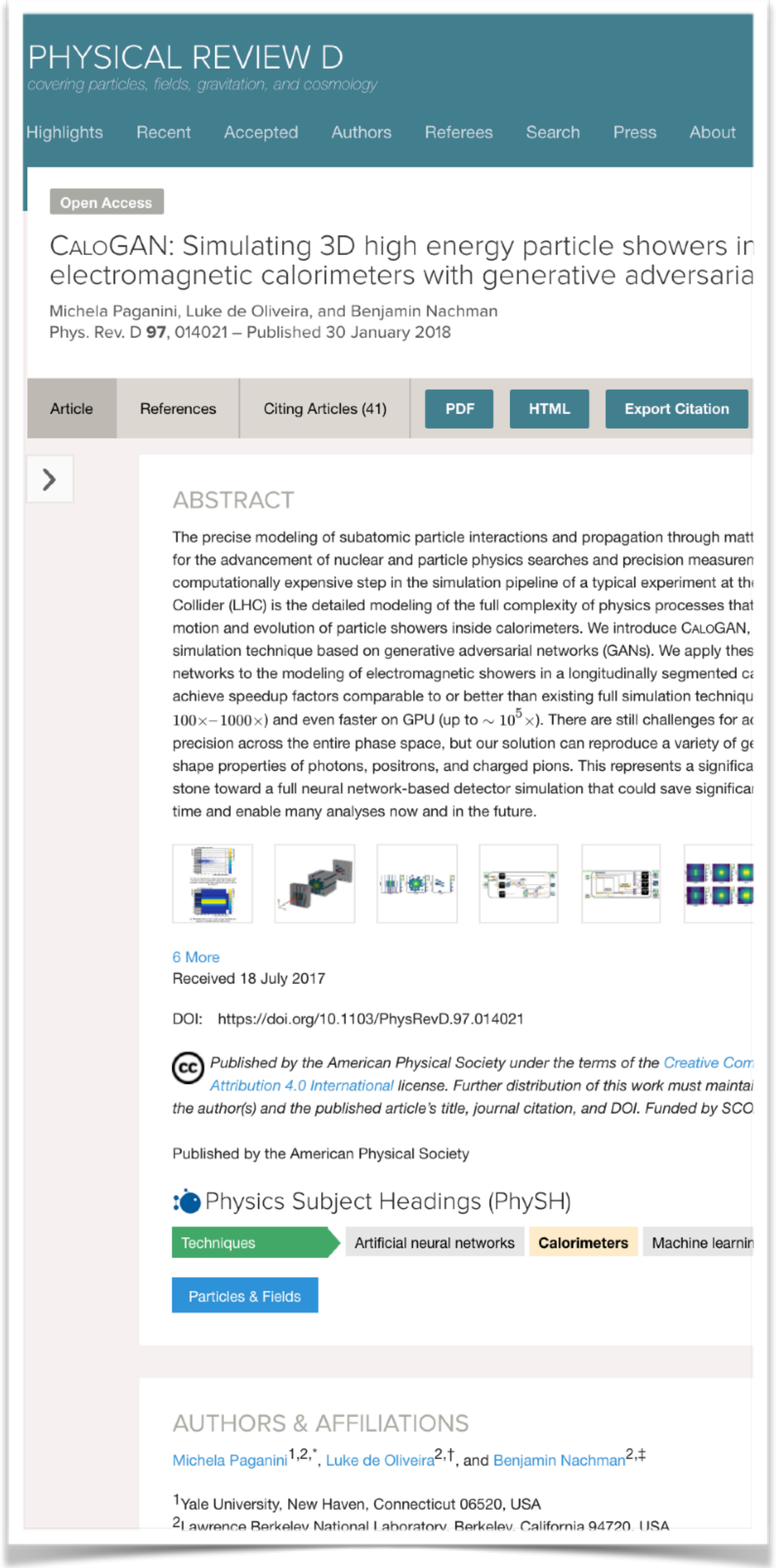
Abstract

We provide a bridge between generative modeling in the Machine Learning
simulated physical processes in high energy particle physics by applying a r
Adversarial Network (GAN) architecture to the production of *jet images*—2
of energy depositions from particles interacting with a calorimeter. We pro
architecture, the Location-Aware Generative Adversarial Network, that lea
realistic radiation patterns from simulated high energy particle collisions. T
of GAN-generated images faithfully span over many orders of magnitude ar
desired low-dimensional physical properties (i.e., jet mass, n-subjettiness, e
on limitations, and provide a novel empirical validation of image quality an
produced simulations of the natural world. This work provides a base for fu
of GANs for use in faster simulation in high energy particle physics.

This is a preview of subscription content, log in to check acce

Notes

1. Full simulation can take up to $\mathcal{O}$(min/event).
2. While the azimuthal angle ϕ is a real angle, pseudorapidity η is only ap
to the polar angle θ. However, the radiation pattern is nearly symmetri
these standard coordinates are used to describe the jet constituent loca
3. For more details about this rotation, which slightly differs from Ref. [2
4. Bicubic spline interpolation in the rotation process causes a large numb
interpolated between their original value and zero, the most likely inter



PHYSICAL REVIEW D

covering particles, fields, gravitation, and cosmology

Highlights Recent Accepted Authors Referees Search Press About

Open Access

CaloGAN: Simulating 3D high energy particle showers ir
electromagnetic calorimeters with generative adversaria

Michela Paganini, Luke de Oliveira, and Benjamin Nachman

Phys. Rev. D 97, 014021 – Published 30 January 2018

Article References Citing Articles (41) PDF HTML Export Citation

ABSTRACT

The precise modeling of subatomic particle interactions and propagation through matt
for the advancement of nuclear and particle physics searches and precision measuren
computationally expensive step in the simulation pipeline of a typical experiment at th
Collider (LHC) is the detailed modeling of the full complexity of physics processes that
motion and evolution of particle showers inside calorimeters. We introduce CaloGAN,
simulation technique based on generative adversarial networks (GANs). We apply thes
networks to the modeling of electromagnetic showers in a longitudinally segmented ca
achieve speedup factors comparable to or better than existing full simulation techniqu
$100\times$–$1000\times$) and even faster on GPU (up to $\sim 10^5\times$). There are still challenges for ac
precision across the entire phase space, but our solution can reproduce a variety of ge
shape properties of photons, positrons, and charged pions. This represents a significa
stone toward a full neural network-based detector simulation that could save significa
time and enable many analyses now and in the future.

6 More

Received 18 July 2017

DOI: https://doi.org/10.1103/PhysRevD.97.014021

Published by the American Physical Society under the terms of the Creative Com
Attribution 4.0 International license. Further distribution of this work must maintai
the author(s) and the published article's title, journal citation, and DOI. Funded by SCO

Published by the American Physical Society

Physics Subject Headings (PhySH)

Techniques Artificial neural networks Calorimeters Machine learnin

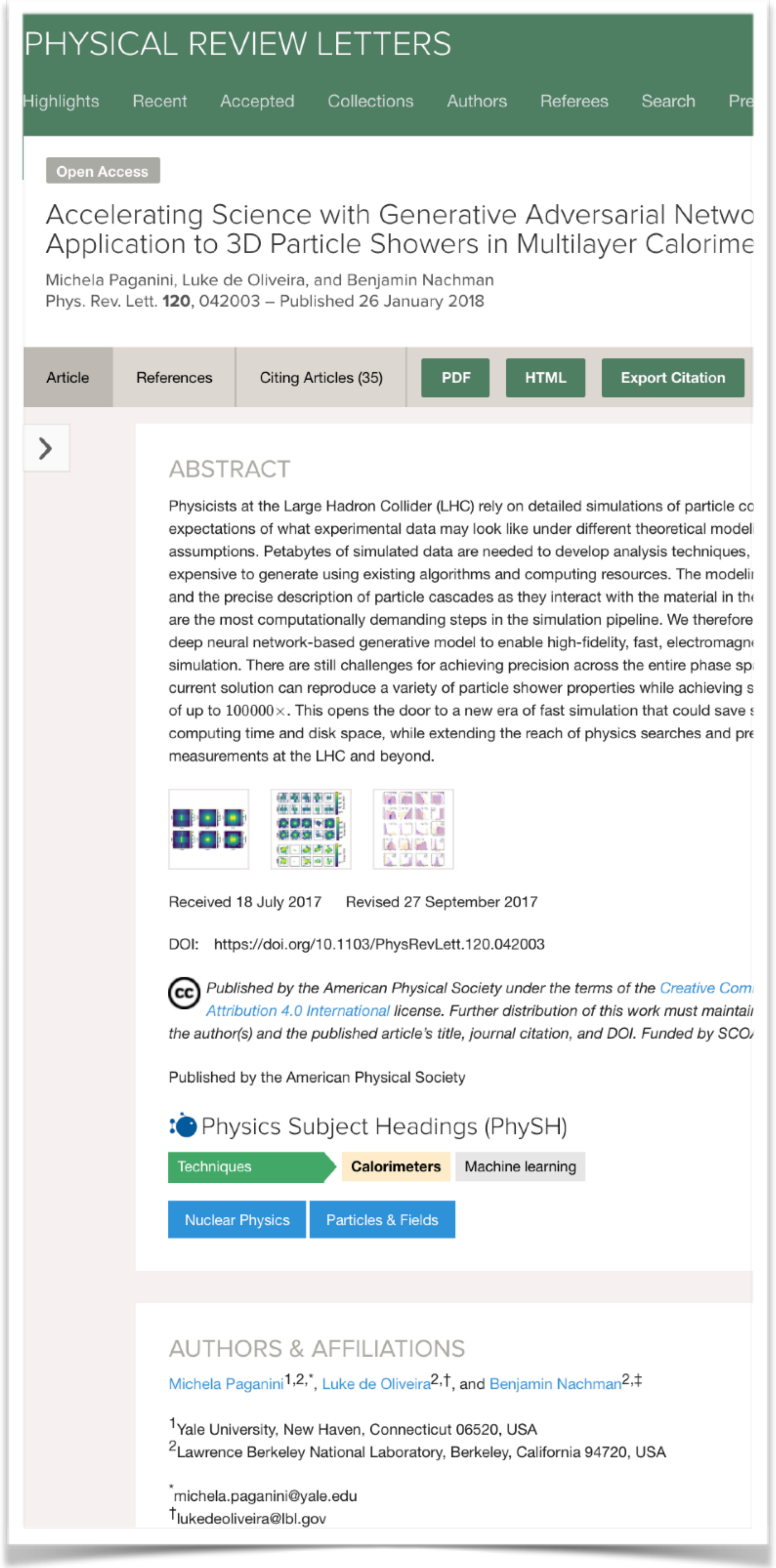
Particles & Fields

AUTHORS & AFFILIATIONS

Michela Paganini[1,2,*], Luke de Oliveira[2,†], and Benjamin Nachman[2,‡]

[1]Yale University, New Haven, Connecticut 06520, USA

[2]Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA



PHYSICAL REVIEW LETTERS

Highlights Recent Accepted Collections Authors Referees Search Pre

Open Access

Accelerating Science with Generative Adversarial Netwo
Application to 3D Particle Showers in Multilayer Calorime

Michela Paganini, Luke de Oliveira, and Benjamin Nachman

Phys. Rev. Lett. 120, 042003 – Published 26 January 2018

Article References Citing Articles (35) PDF HTML Export Citation

ABSTRACT

Physicists at the Large Hadron Collider (LHC) rely on detailed simulations of particle cc
expectations of what experimental data may look like under different theoretical model
assumptions. Petabytes of simulated data are needed to develop analysis techniques,
expensive to generate using existing algorithms and computing resources. The modeli
and the precise description of particle cascades as they interact with the material in th
are the most computationally demanding steps in the simulation pipeline. We therefore
deep neural network-based generative model to enable high-fidelity, fast, electromagn
simulation. There are still challenges for achieving precision across the entire phase sp
current solution can reproduce a variety of particle shower properties while achieving s
of up to $100000\times$. This opens the door to a new era of fast simulation that could save s
computing time and disk space, while extending the reach of physics searches and pre
measurements at the LHC and beyond.

Received 18 July 2017 Revised 27 September 2017

DOI: https://doi.org/10.1103/PhysRevLett.120.042003

Published by the American Physical Society under the terms of the Creative Com
Attribution 4.0 International license. Further distribution of this work must maintai
the author(s) and the published article's title, journal citation, and DOI. Funded by SCO

Published by the American Physical Society

Physics Subject Headings (PhySH)

Techniques Calorimeters Machine learning

Nuclear Physics Particles & Fields

AUTHORS & AFFILIATIONS

Michela Paganini[1,2,*], Luke de Oliveira[2,†], and Benjamin Nachman[2,‡]

[1]Yale University, New Haven, Connecticut 06520, USA

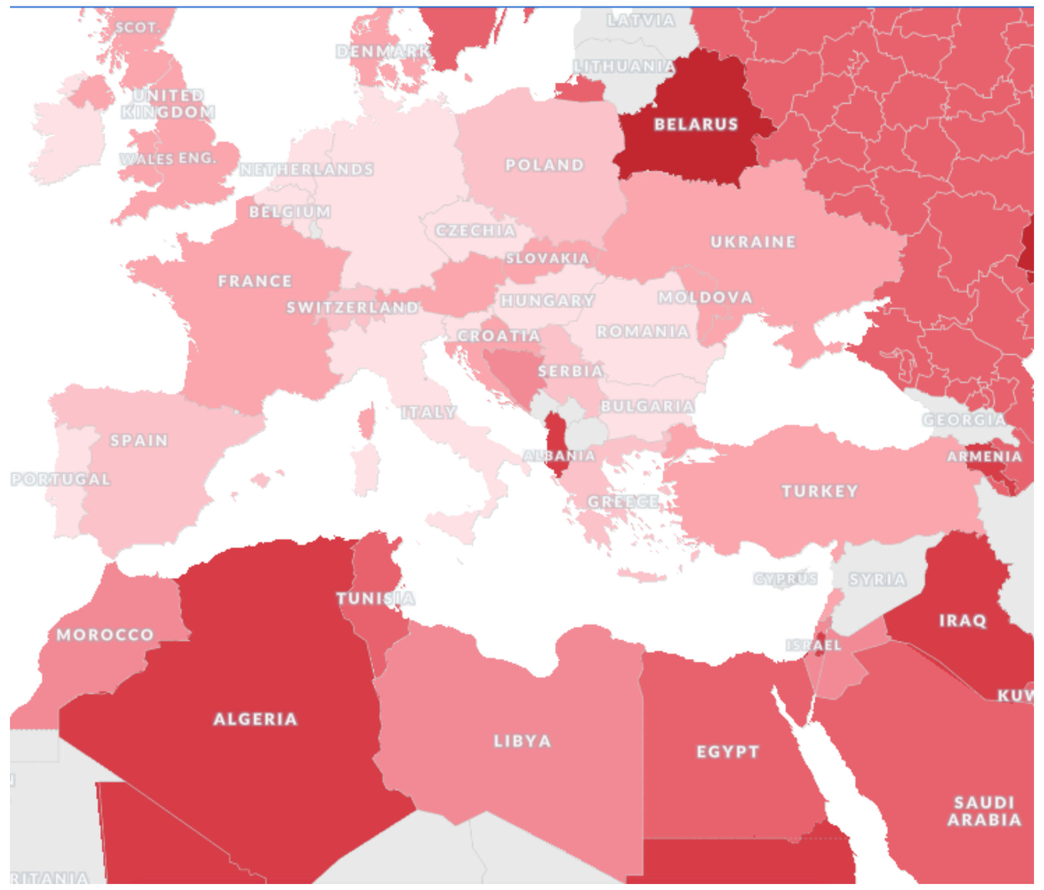
[2]Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA

*michela.paganini@yale.edu

†lukedeoliveira@lbl.gov

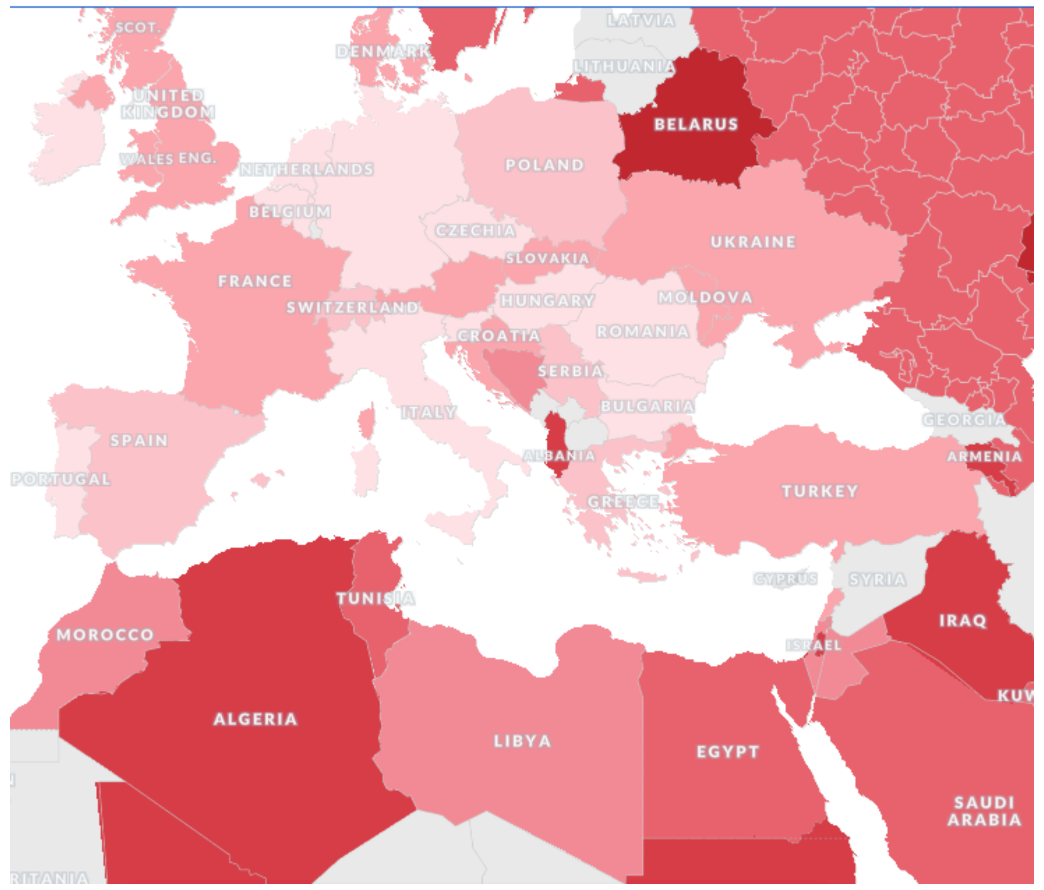
AI for Science at facebook

COVID-19 Interactive Map & Dashboard

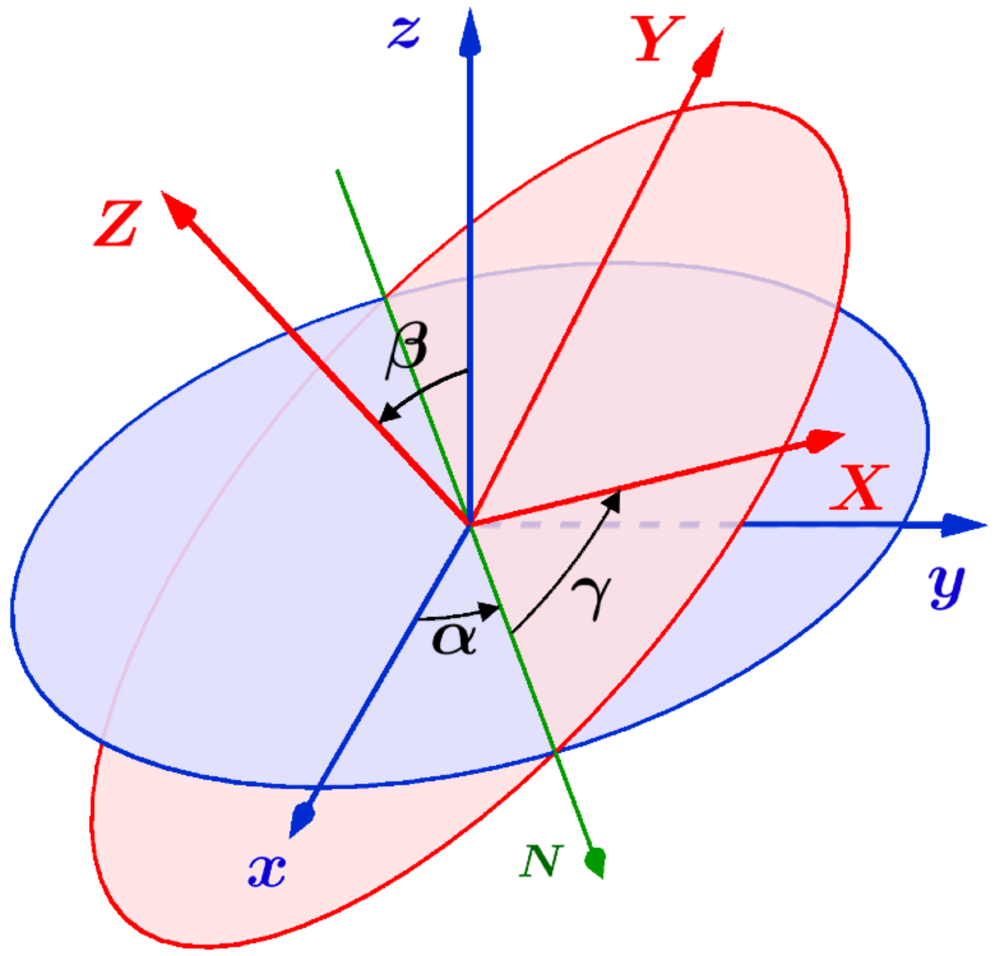


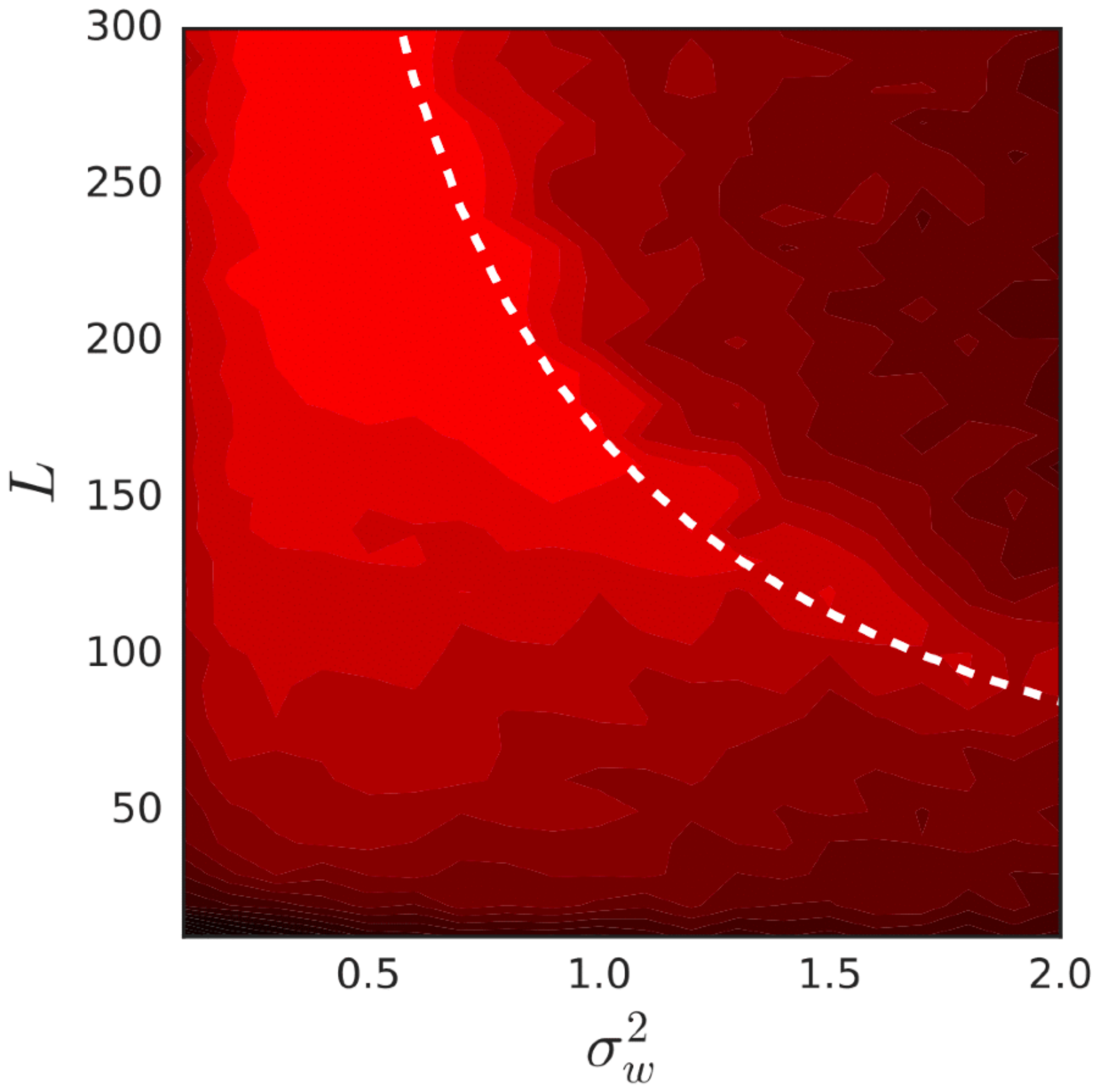
AI for Science at facebook

COVID-19 Interactive Map & Dashboard



Science for AI





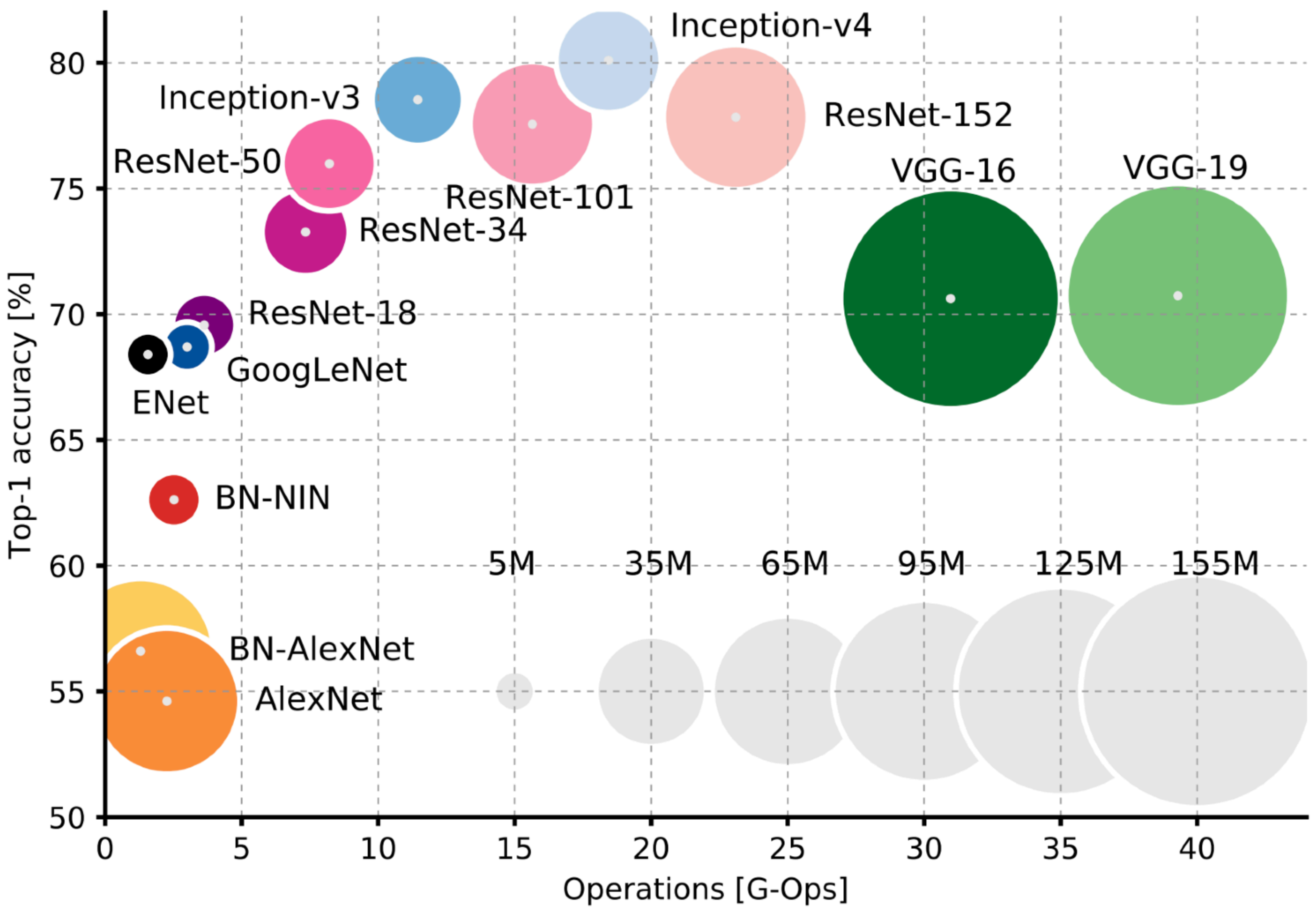
Agenda

1. Introduction to pruning
2. Pruning for applied research
3. Pruning for fundamental research

01 Introduction to Pruning

Network capacity and over-parametrization

Models continue to grow



Canziani et al., 2016

Language Models are Few-Shot Learners

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion parameters, 10x more than any previous non-sparse language model, and test its performance in the few-shot setting. For all tasks, GPT-3 is applied without any

GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, Zhifeng Chen

Neural network scaling has been critical for improving the model quality in many real-world machine learning applications with vast amounts of training data and compute. Although this trend of scaling is affirmed to be a sure-fire approach for better model quality, there are challenges on the path such as the computation cost, ease of programming, and efficient implementation on parallel devices. GShard is a module composed of a set of lightweight annotation APIs and an extension to the XLA compiler. It provides an elegant way to express a wide range of parallel computation patterns with minimal changes to the existing model code. GShard enabled us to scale up multilingual neural machine translation Transformer model with Sparsely-Gated Mixture-of-Experts beyond 600 billion parameters using automatic sharding. We demonstrate that such a giant model can efficiently be trained on 2048 TPU v3 accelerators in 4 days to achieve far superior quality for translation from 100 languages to English compared to the prior art.

Pruning Large Models

Common Practice	Train Small Model →	Stop Training When Converged →	Lightly Compress
Optimal	Train Large Model →	Stop Training Early →	Heavily Compress

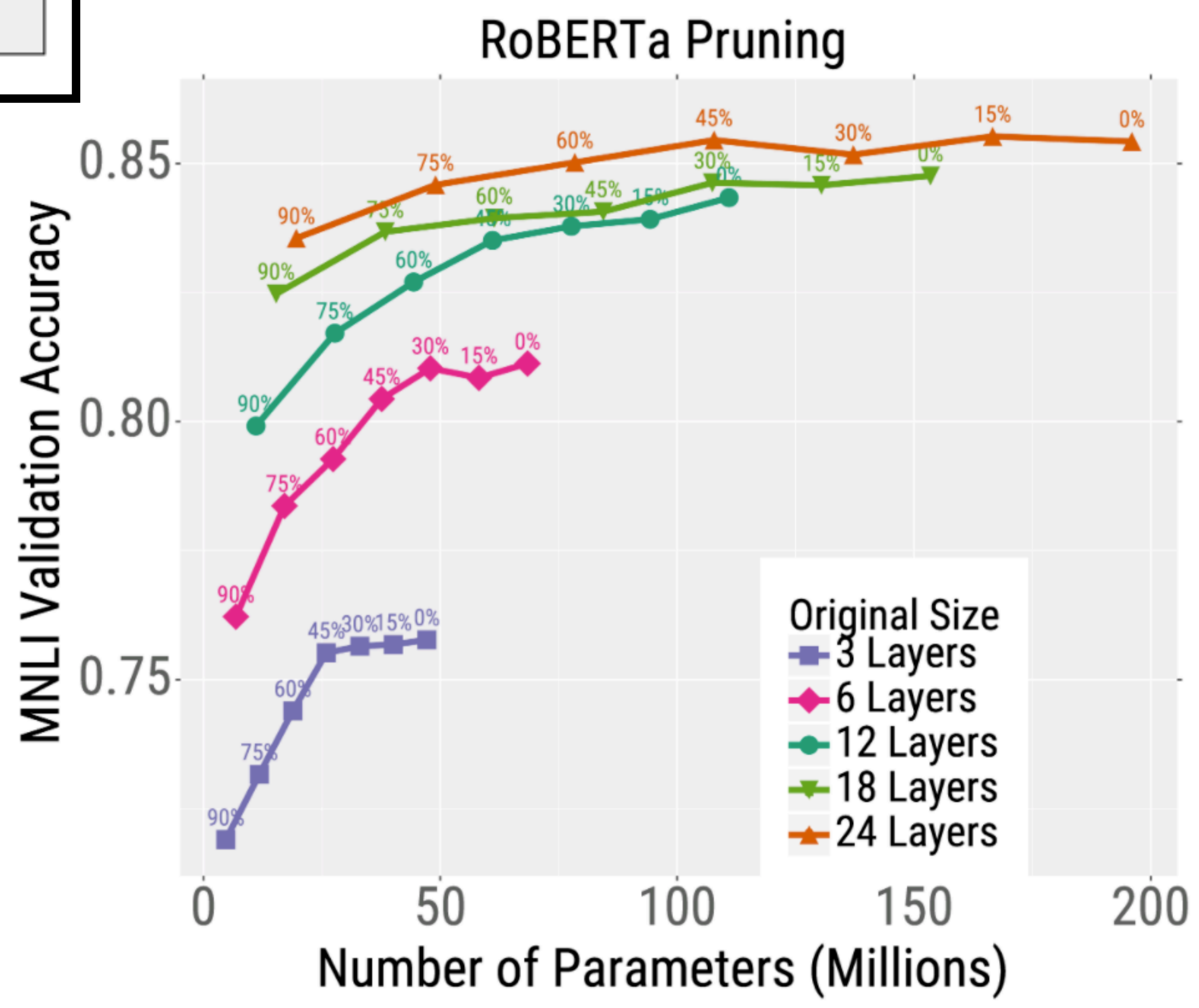
- "**Model optimization**" is extremely common in practice
- Best to start out with very large models and prune
- In transformers, can prune away many of the heads (structured pruning) or many parameters globally (unstructured pruning) or even entire layers with minimal performance penalty
- Can combine pruning and quantization for best results

References:

[Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers]

[Are Sixteen Heads Really Better than One?]

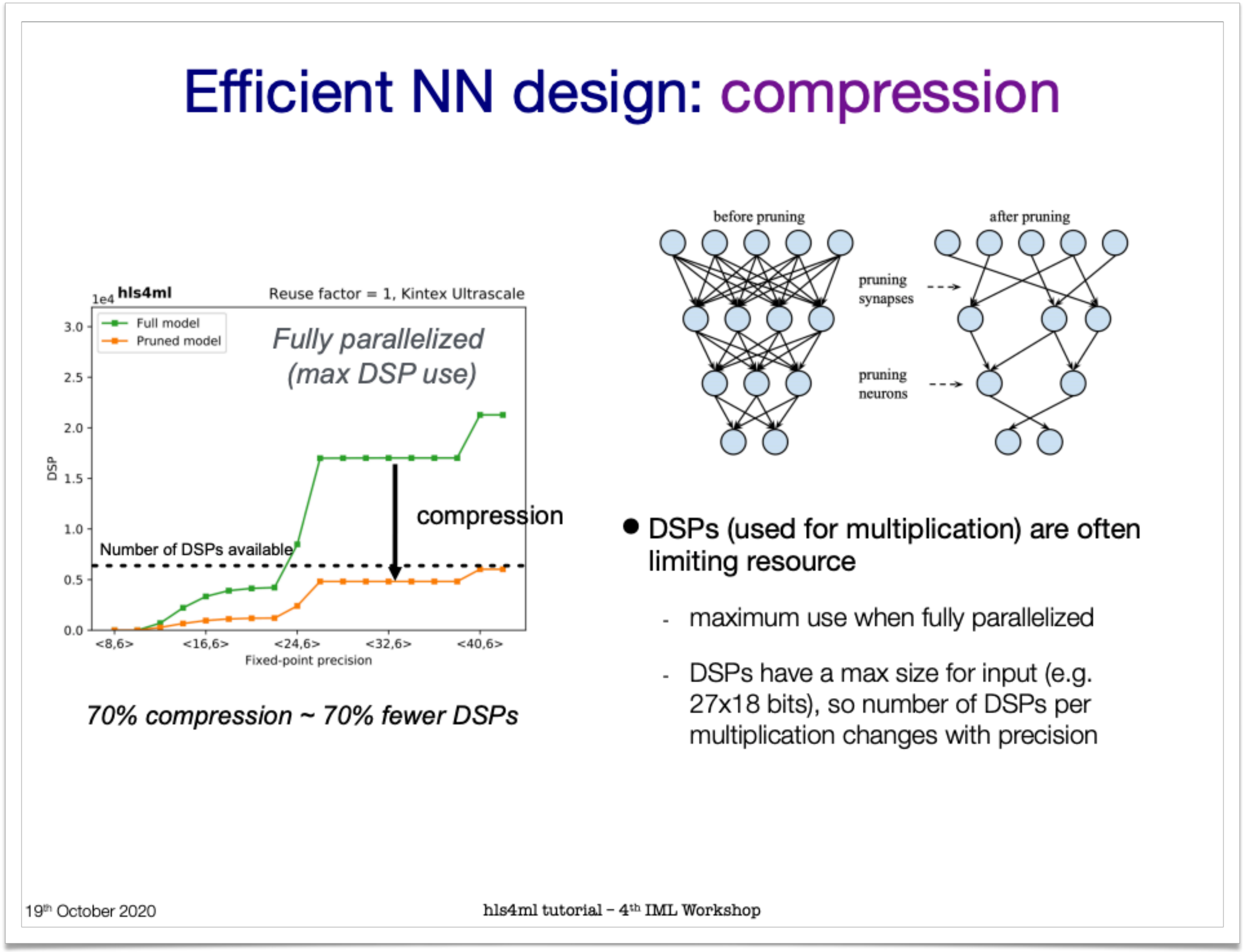
[Reducing Transformer Depth on Demand with Structured Dropout]



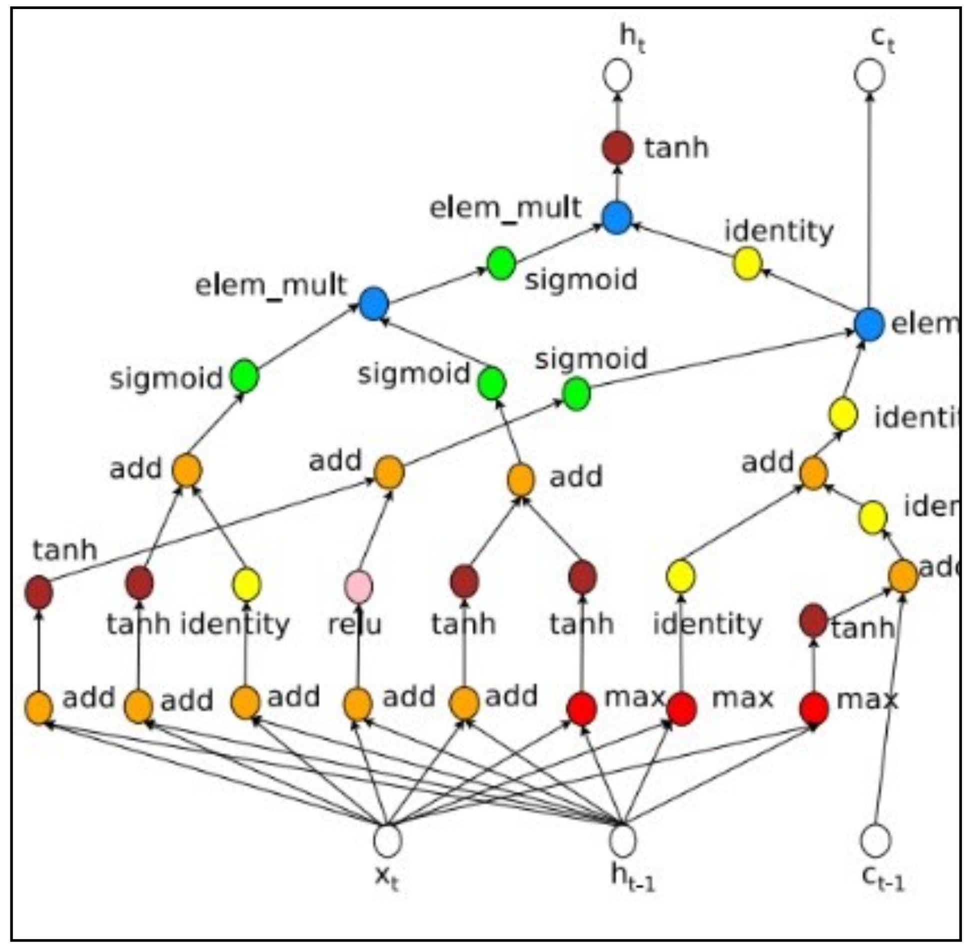
Relevance to HEP

- Trigger and real-time applications
- Low latency, high throughput, low power
- Custom hardware deployment
- Hardware-software co-design
- Memory savings

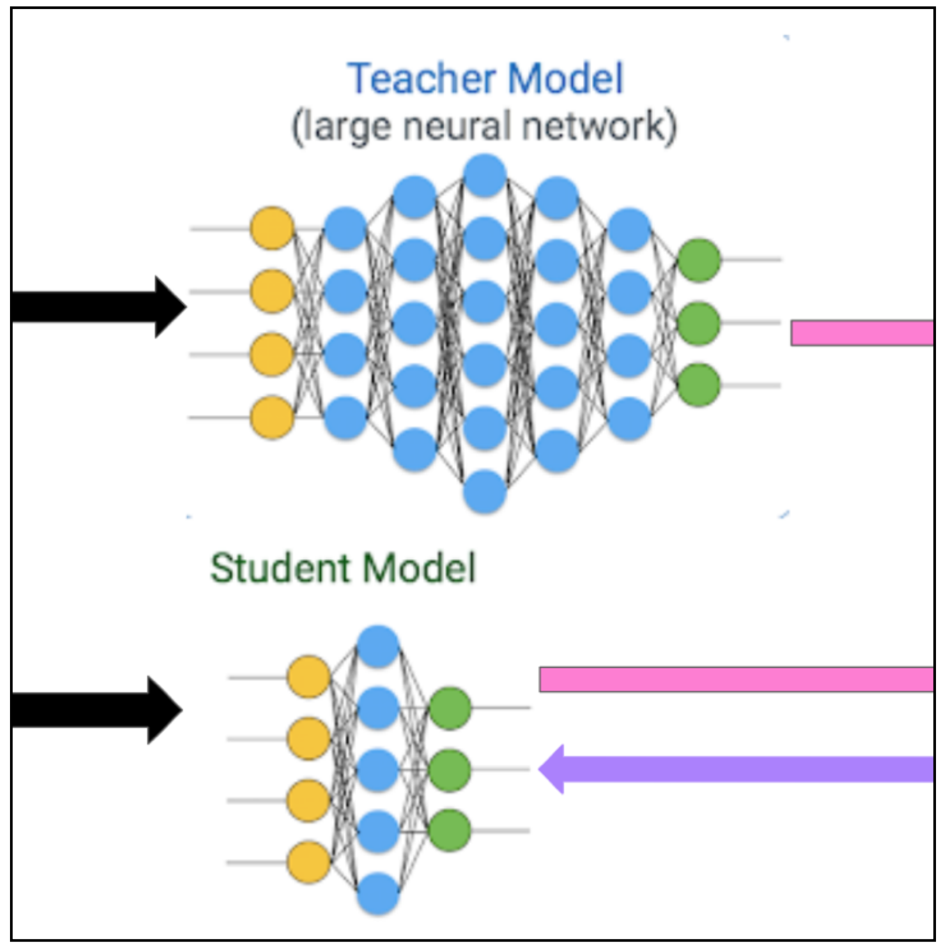
See Monday's `hls4ml` tutorial!



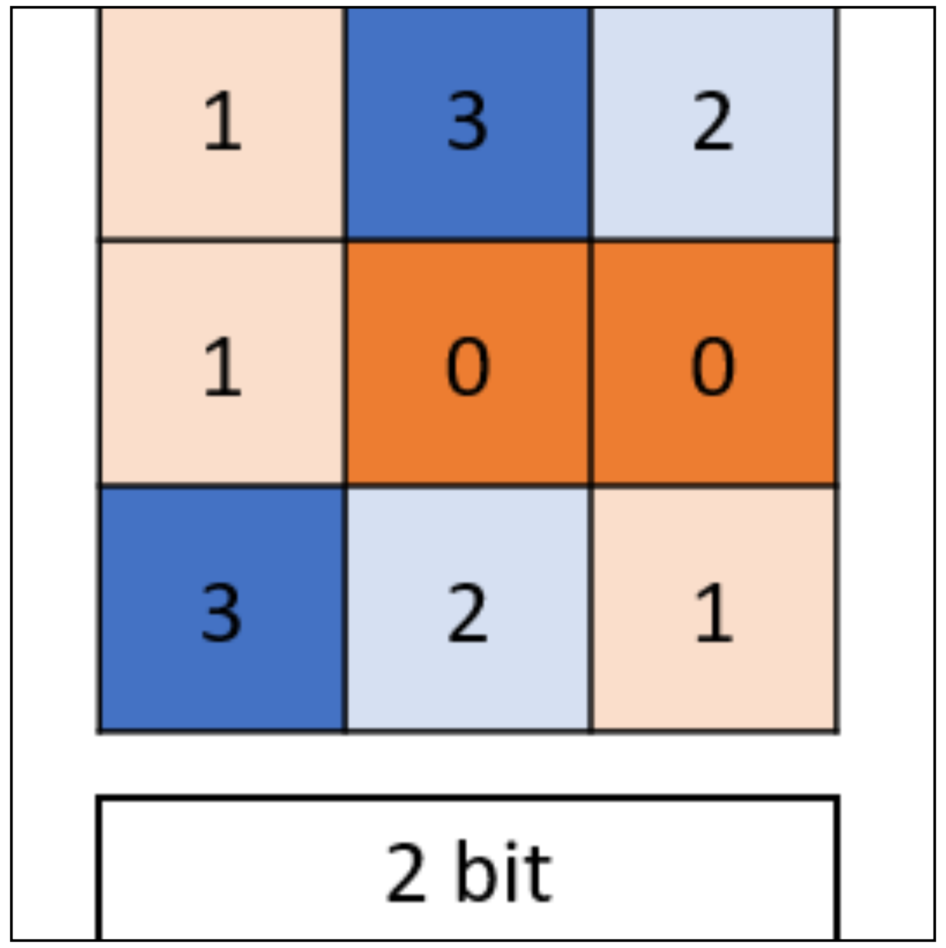
Efficient model design



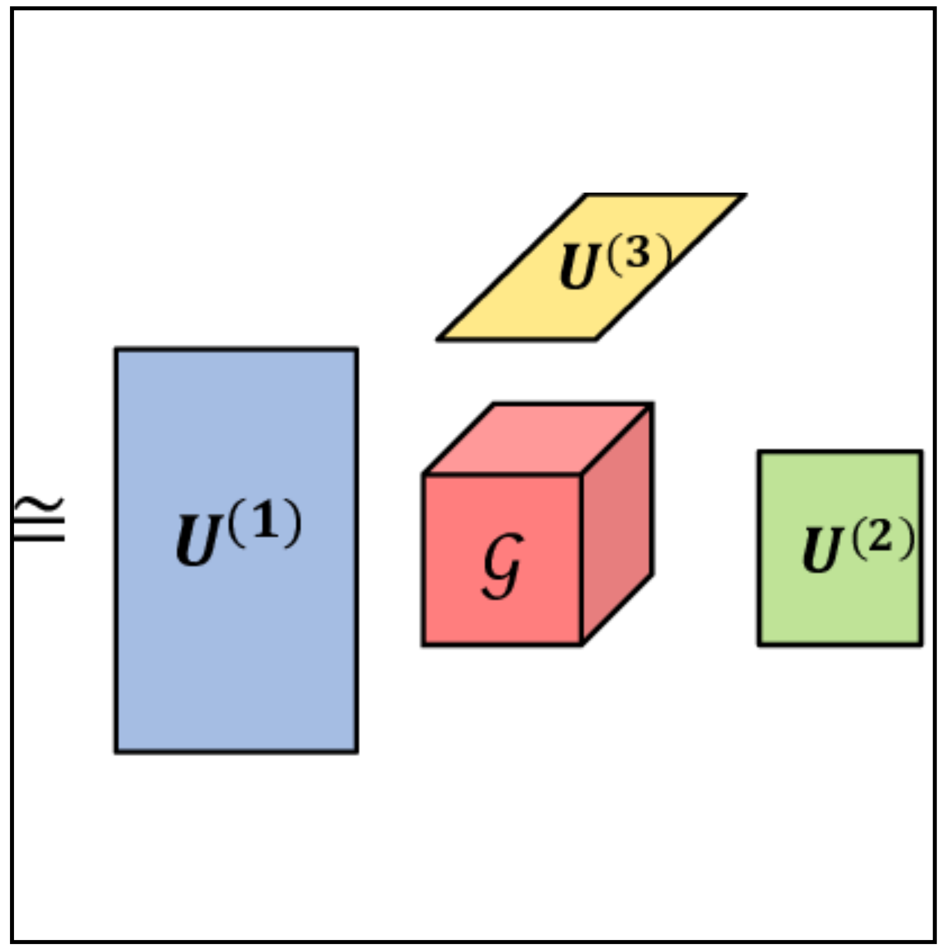
Hand-design or automated design (AutoML)



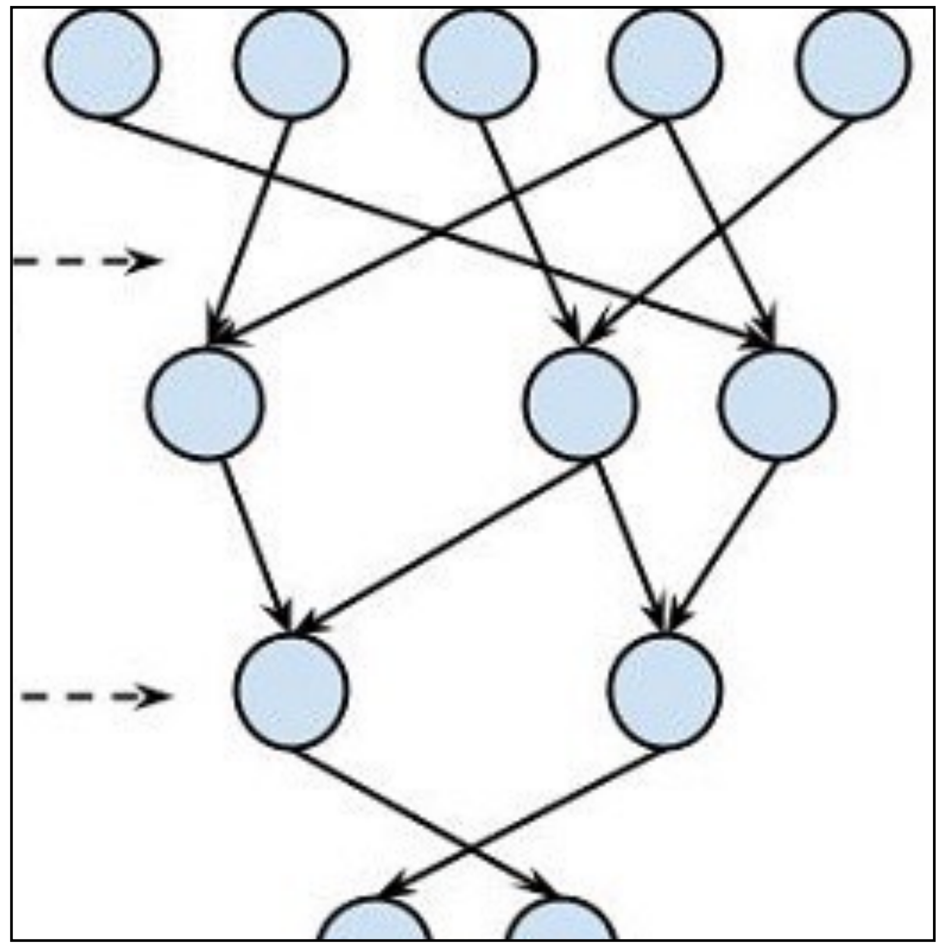
Distillation



Quantization

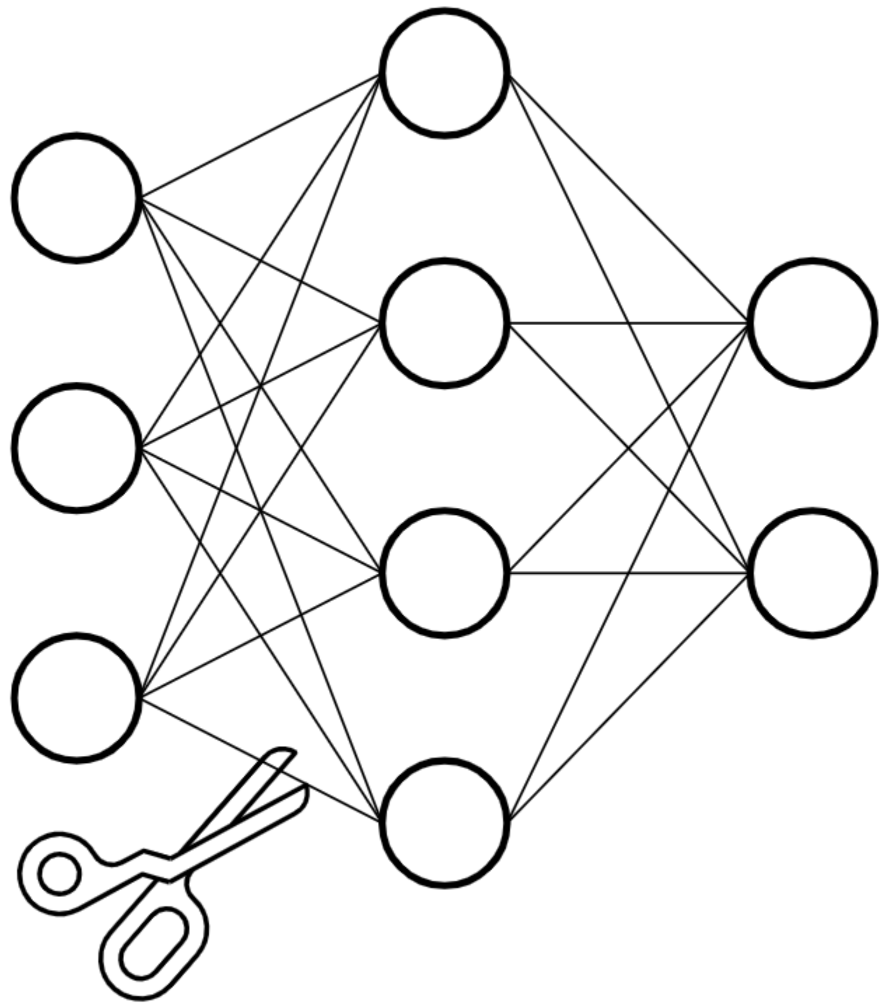


Tensor Decomposition

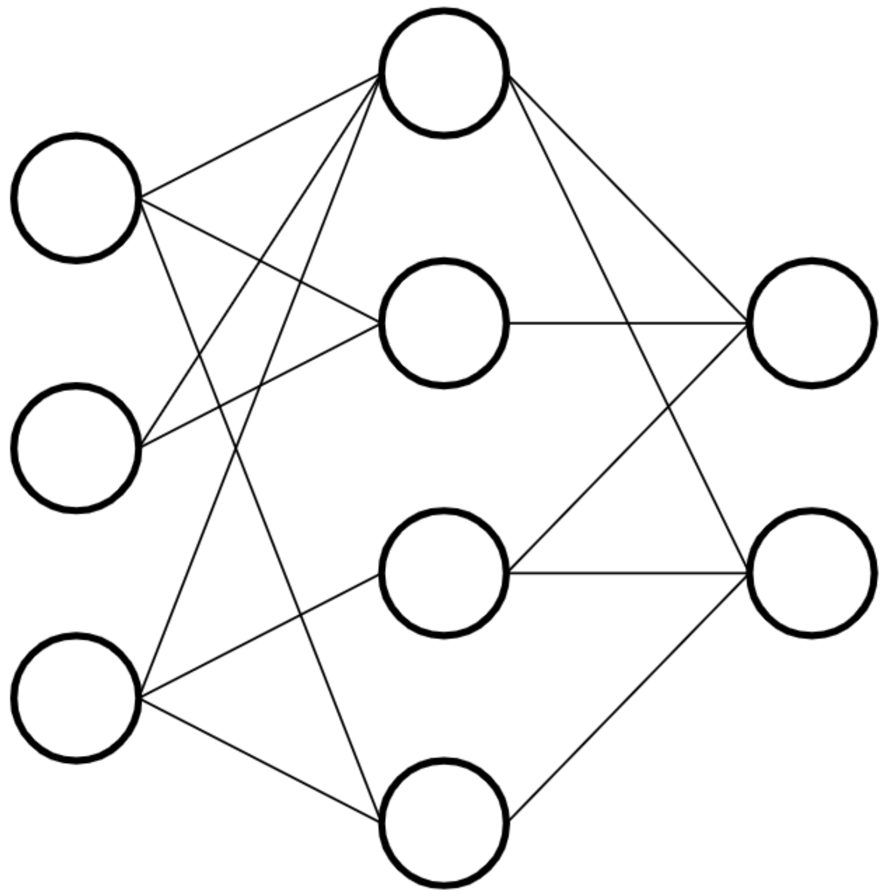


Pruning

Pruning



Before pruning



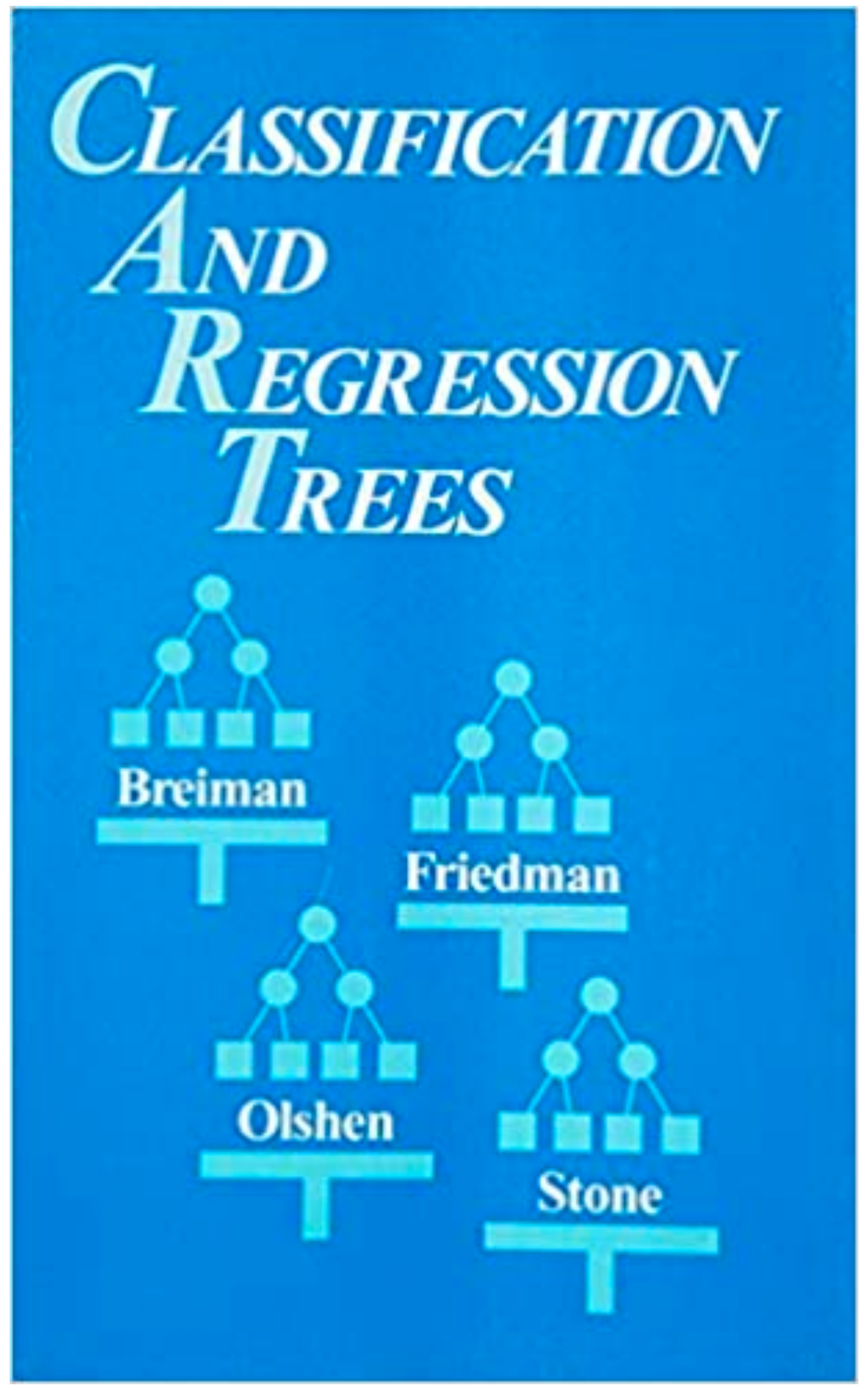
After pruning

"removing superfluous structure"

how to identify?

what kind of structure?

Tree Pruning



AI Memo No. 930 December, 1986

Simplifying Decision Trees

J. R. Quinlan[1]

Abstract: Many systems have been developed for constructing decision trees from collections of examples. Although the decision trees generated by these methods are accurate and efficient, they often suffer the disadvantage of excessive complexity that can render them incomprehensible to experts. It is questionable whether opaque structures of this kind can be described as knowledge, no matter how well they function. This paper discusses techniques for simplifying decision trees without compromising their accuracy. Four methods are described, illustrated, and compared on a test-bed of decision trees from a variety of domains.

Machine Learning, 4, 227–243 (1989)

An Empirical Comparison of Pruning Methods for Decision Tree Induction

JOHN MINGERS BSRCD@CU.WARWICK.AC.UK
School of Industrial and Business Studies, University of Warwick, Coventry CV4 7AL, England

Editor: Jaime Carbonell

Abstract. This paper compares five methods for pruning decision trees, developed from sets of examples. When used with uncertain rather than deterministic data, decision-tree induction involves three main stages—creating a complete tree able to classify all the training examples, pruning this tree to give statistical reliability, and processing the pruned tree to improve understandability. This paper concerns the second stage—pruning. It presents empirical comparisons of the five methods across several domains. The results show that three methods—critical value, error complexity and reduced error—perform well, while the other two may cause problems. They also show that there is no significant interaction between the creation and pruning methods.

Neural Net Pruning – Why and How

Authors **J. Sietsma and R.J.F. Dow**

Materials Research Laboratory
D.S.T.O., Melbourne
P.O. Box 50,
Ascot Vale 3032
Australia

Abstract

A continuing question in neural net research is the size of network needed to solve a particular problem. If training is started with too small a network for the problem no learning can occur. The researcher must then go through a slow process of deciding that no learning is taking place, increasing the size of the network and training again. If a network that is larger than required is used then processing is slowed, particularly on a conventional von Neumann computer. This paper discusses an approach to this problem based on learning with a net which is larger than the minimum size network required to solve the problem and then pruning the solution network. The result is a small, efficient network that performs as well or better than the original. This does not give a complete answer to the question since the size of the initial network is still largely based on guesswork but it gives a very useful partial answer and sheds some light on the workings of a neural network in the process.

Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 1, NO. 2, JUNE 1990

Letters

A Simple Procedure for Pruning Back-Propagation Trained Neural Networks

EHUD D. KARNIN

Abstract—One possible method of obtaining a neural network of an appropriate size for a particular problem is to start with a larger net, then prune it to the desired size. Training and retraining the net under all possible subsets of the set of synapses will result in a prohibitively long learning process; hence some methods that avoid this exhaustive search have been proposed. Here we estimate the *sensitivity* of the global error (cost) function to the inclusion/exclusion of each synapse in the artificial neural network. We do it by introducing "shadow arrays" that keep track of the incremental changes to the synaptic weights during (a single pass of) back-propagating learning. The synapses are then ordered by decreasing sensitivity numbers so that the network can be efficiently pruned by discarding the last items of the sorted list. Unlike previous approaches this simple procedure does not require a modification of the cost function, does not interfere with the learning process, and demands a negligible computational overhead.

SKELETONIZATION: A TECHNIQUE FOR TRIMMING THE FAT FROM A NETWORK VIA RELEVANCE ASSESSMENT

Michael C. Mozer
Paul Smolensky
Department of Computer Science &
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430

ABSTRACT

This paper proposes a means of using the knowledge in a network to determine the functionality or *relevance* of individual units, both for the purpose of understanding the network's behavior and improving its performance. The basic idea is to iteratively train the network to a certain performance criterion, compute a measure of relevance that identifies which input or hidden units are most critical to performance, and automatically trim the least relevant units. This *skeletonization* technique can be used to simplify networks by eliminating units that convey redundant information; to improve learning performance by first learning with spare hidden units and then trimming the unnecessary ones away, thereby constraining generalization; and to understand the behavior of networks in terms of minimal "rules."

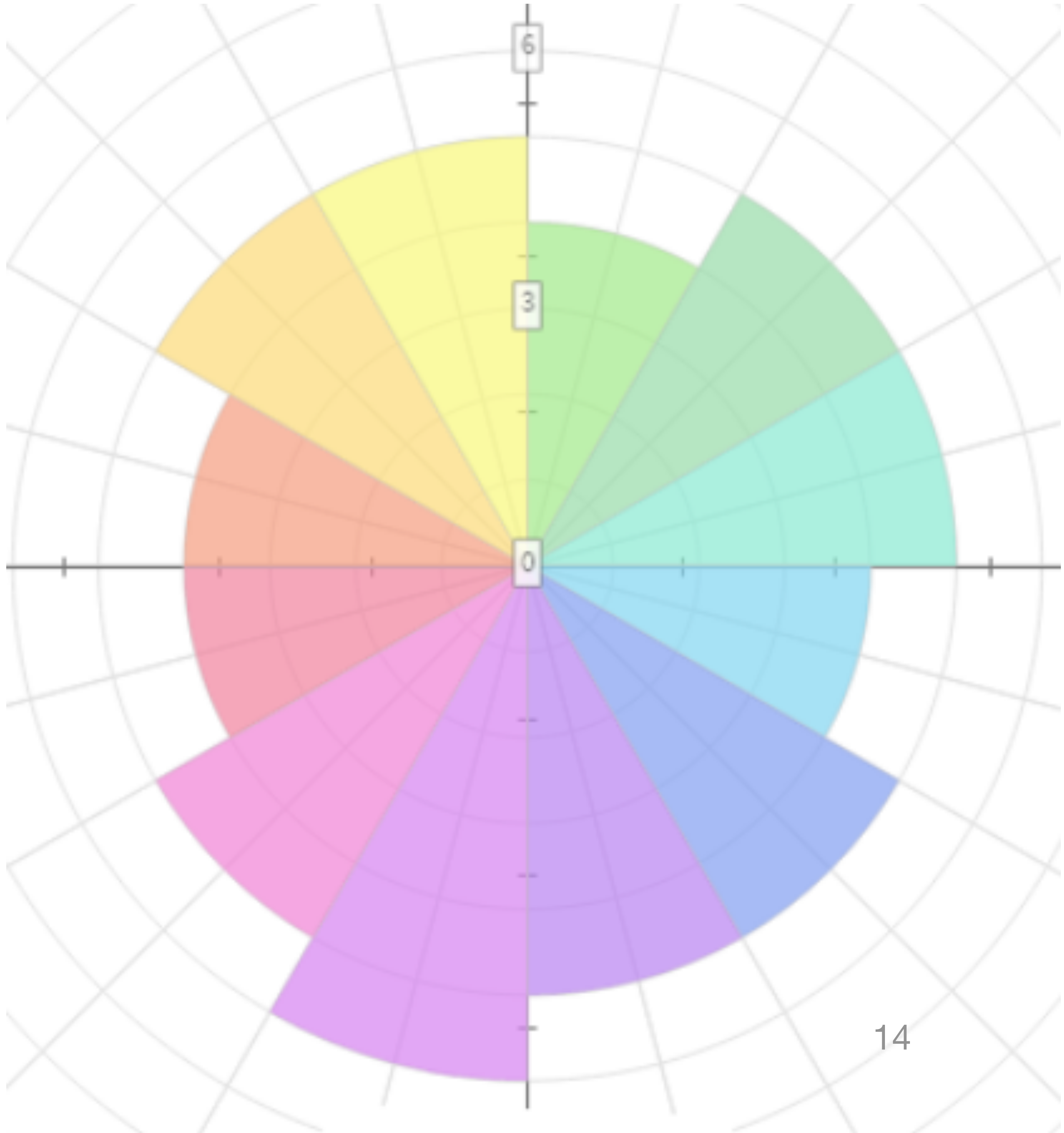
The state of pruning

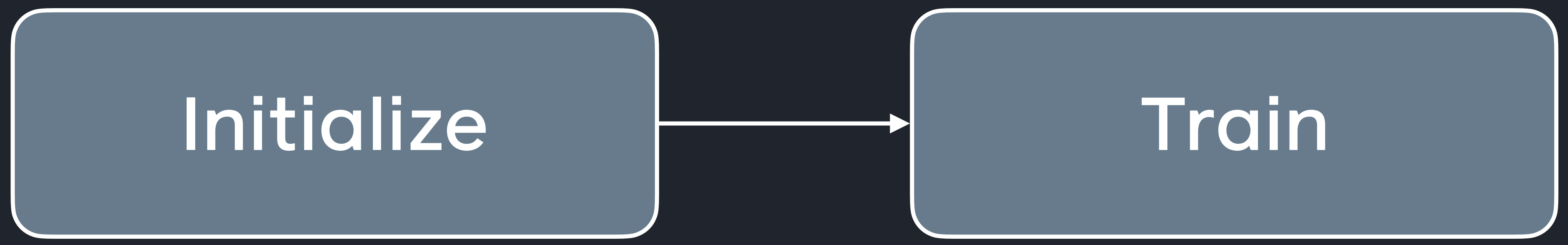
Pruning should remove unnecessary redundancy and unused capacity

Can be executed *before, during,* and *after* training

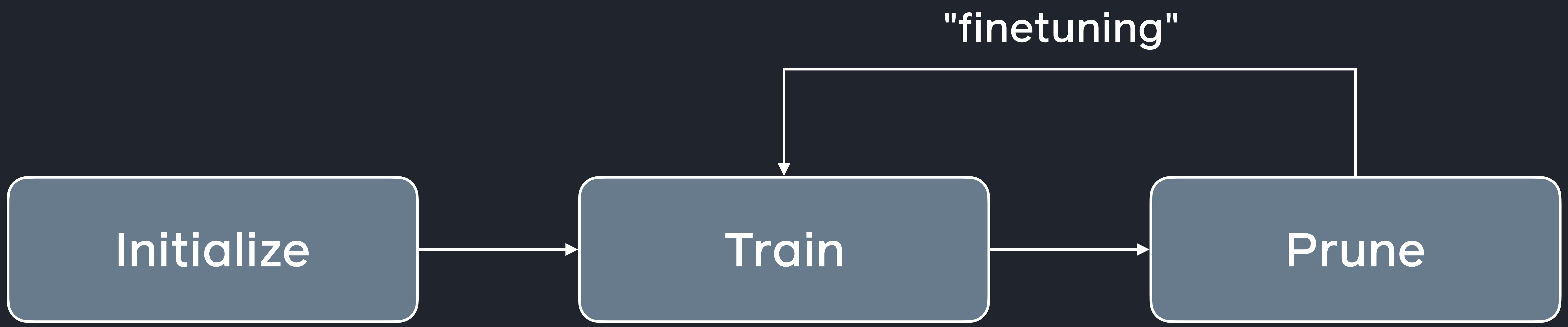
Pruning methods differ across many dimensions:

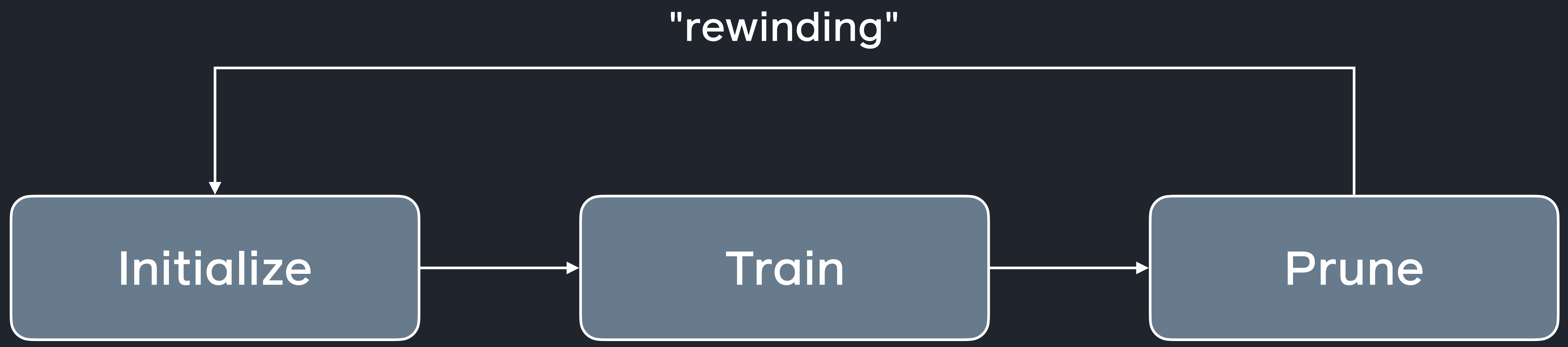
- based on weight magnitude, activations, gradients, Hessian, interpretability measures, credit assignment, random, etc.
- Layer-wise vs global, unstructured vs structured, etc.
- Rule-based, bayesian, differentiable, soft approaches, etc.
- One-shot vs iterative pruning
- Followed by: finetuning, reinitialization, rewinding

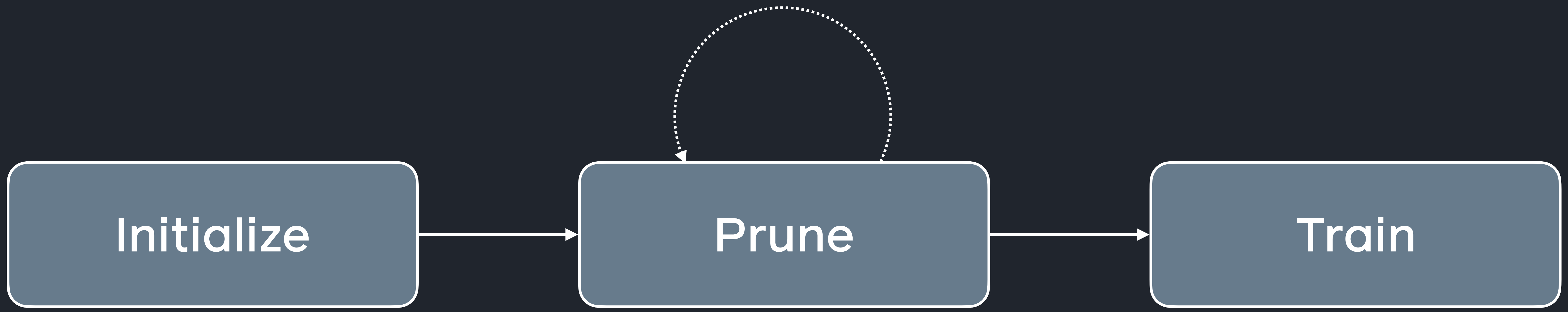



Initialize
Train


Initialize
Train
Prune


"finetuning"
Initialize
Train
Prune


"rewinding"
Initialize
Train
Prune


Initialize
Prune
Train

Structured vs unstructured pruning

Algorithm 1: Structured Pruning

Input: Tensor $w \in \mathbb{R}^{d_1 \times \cdots \times d_R}$; axis $i \in \{0, \ldots, R\}$; criterion $C : \cdot \longrightarrow \mathbb{R}$; pruning fraction $p \in [0, 1]$

Result: Masked tensor $\tilde{w} \in \mathbb{R}^{d_1 \times \cdots \times d_R}$

$\mu = J_{d_1, \ldots, d_R}$; *// Initialize mask to ones tensor*

$K = p * d_i$; *// Number of entries in axis i to prune*

`criteria = []`

for $j < d_i$ **do**

  `criteria.append`$(-C(w_{[\ldots,j,\ldots]}))$

end

for q *in arg_top_k(*`criteria`*, K)* **do**

  $\mu_{[\ldots,q,\ldots]} = 0$ *// Slice along the i^{th} axis*

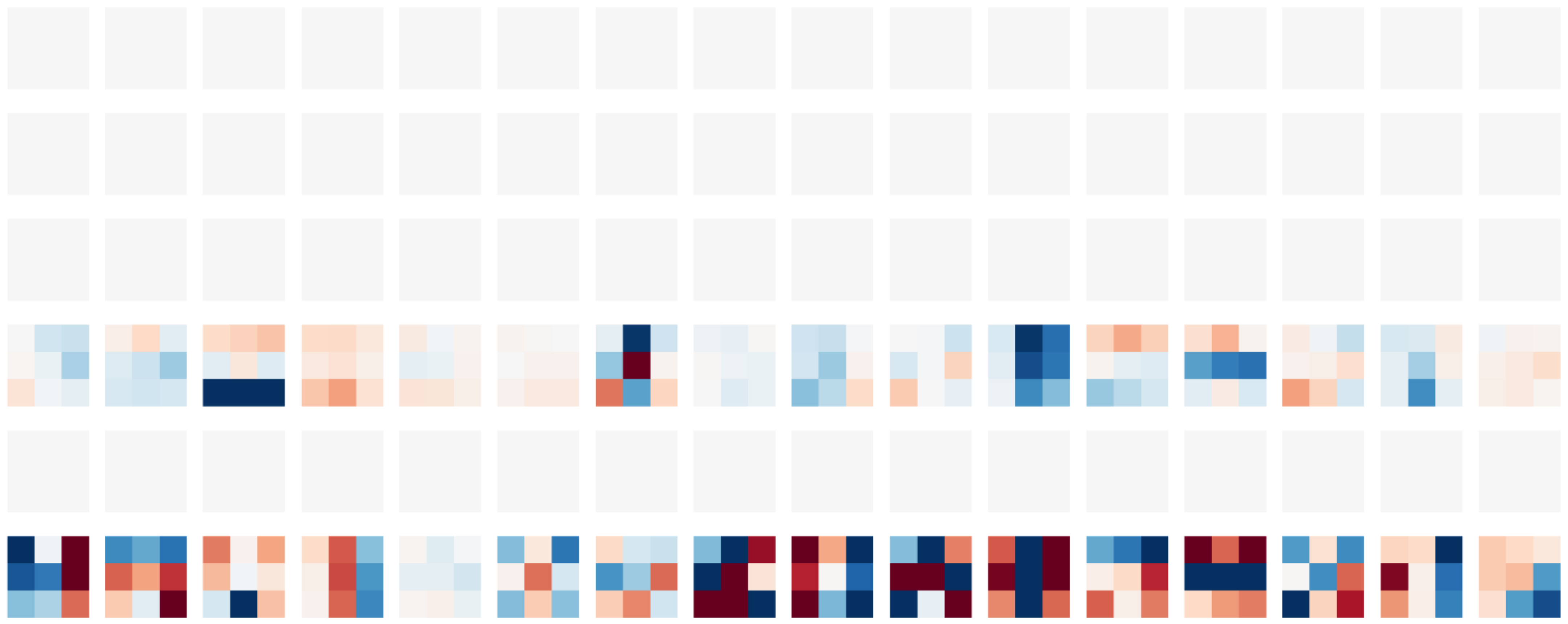
end

$\tilde{w} = w \odot \mu$

return $\tilde{w}$

parameters in a convolutional layer

$$w \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times ks_0 \times ks_1}$$



Structured: remove entire channels

Structured vs unstructured pruning

Algorithm 1: Structured Pruning

Input: Tensor $w \in \mathbb{R}^{d_1 \times \cdots \times d_R}$; axis $i \in \{0, \ldots, R\}$; criterion $C : \cdot \longrightarrow \mathbb{R}$; pruning fraction $p \in [0, 1]$
Result: Masked tensor $\tilde{w} \in \mathbb{R}^{d_1 \times \cdots \times d_R}$

$\mu = J_{d_1,\ldots,d_R}$; *// Initialize mask to ones tensor*
$K = p * d_i$; *// Number of entries in axis i to prune*
`criteria = []`
for $j < d_i$ **do**
| `criteria.append`$(-C(w_{[\ldots,j,\ldots]}))$
end
for q *in arg_top_k(criteria, K)* **do**
| $\mu_{[\ldots,q,\ldots]} = 0$ *// Slice along the* i^{th} *axis*
end
$\tilde{w} = w \odot \mu$
return $\tilde{w}$

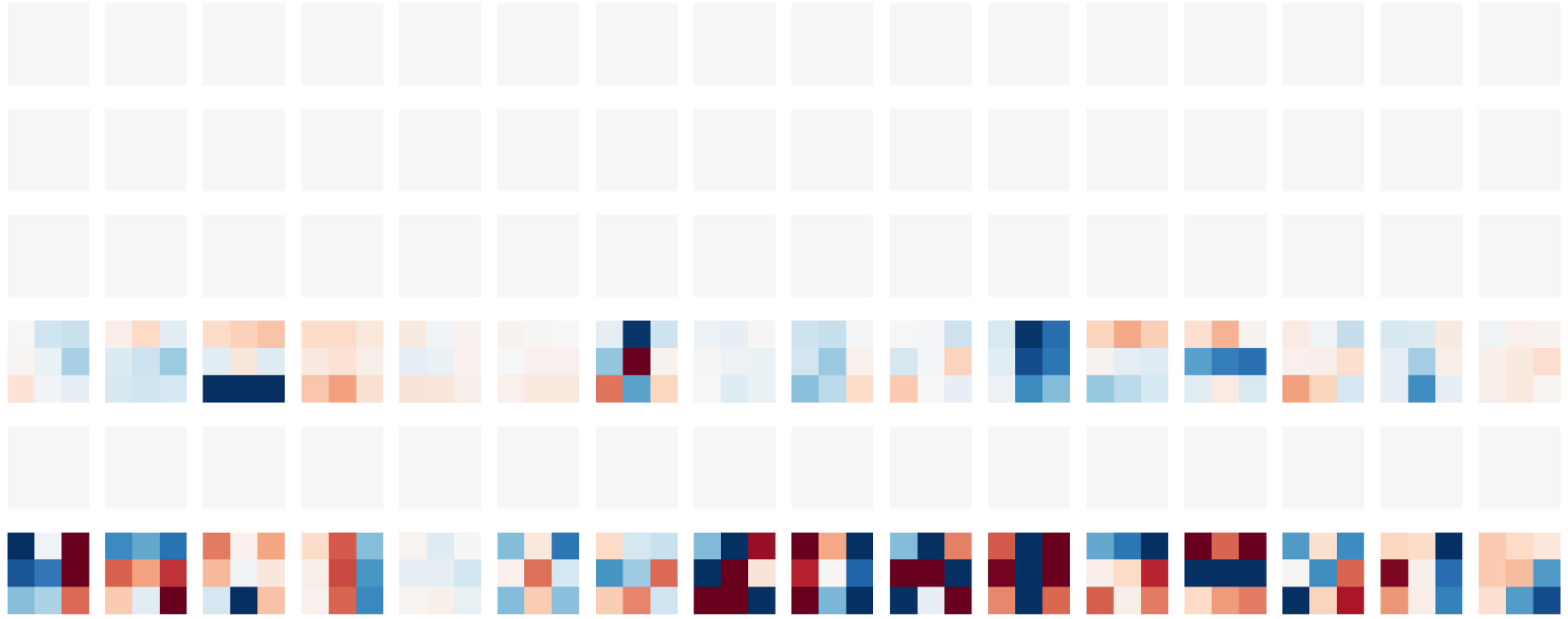
parameters in a convolutional layer

$$w \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times ks_0 \times ks_1}$$

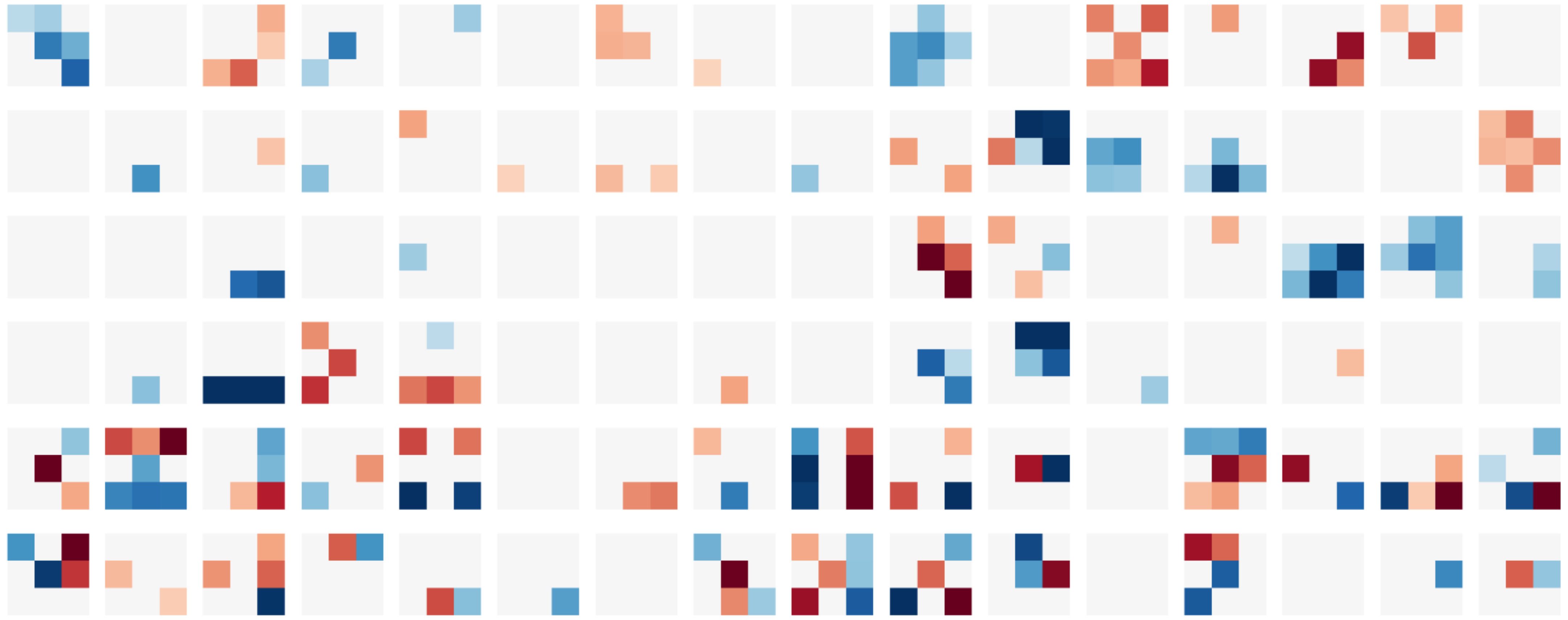
Algorithm 2: Unstructured Pruning

Input: Tensor $w \in \mathbb{R}^{d_1 \times \cdots \times d_R}$; criterion $C : \cdot \longrightarrow \mathbb{R}$; pruning fraction $p \in [0, 1]$
Result: Masked tensor $\tilde{w} \in \mathbb{R}^{d_1 \times \cdots \times d_R}$

$\mu - J_{d_1,\ldots,d_R}$; c
$K = p * \prod_{i<R} d_i$; *// Number of entries to prune*
`criteria = []`
for $j < \prod_{i<R} d_i$ **do**
| `criteria.append`$(-C(w_j))$
end
for q *in arg_top_k(criteria, K)* **do**
| $\mu_q = 0$
end
$\tilde{w} = w \odot \mu$
return $\tilde{w}$



Structured: remove entire channels



Unstructured: remove individual connections

Pruning criteria

$w_i \in \mathbb{R}^{d_1 \times \ldots \times d_N}$

$$\tilde{w}_f = w_f \odot \mathbb{1}_{f(w_i, w_f, \ldots) > \mathcal{T}}$$

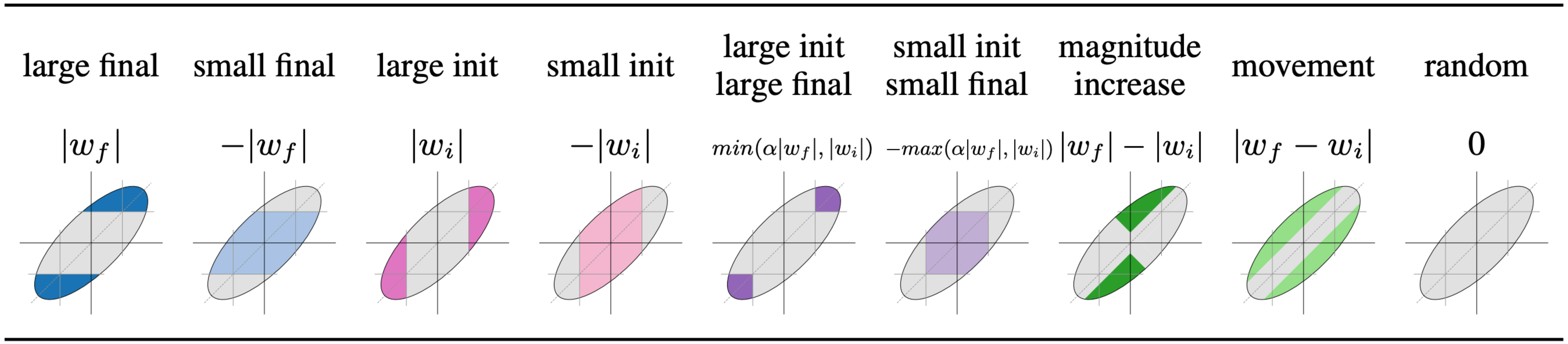


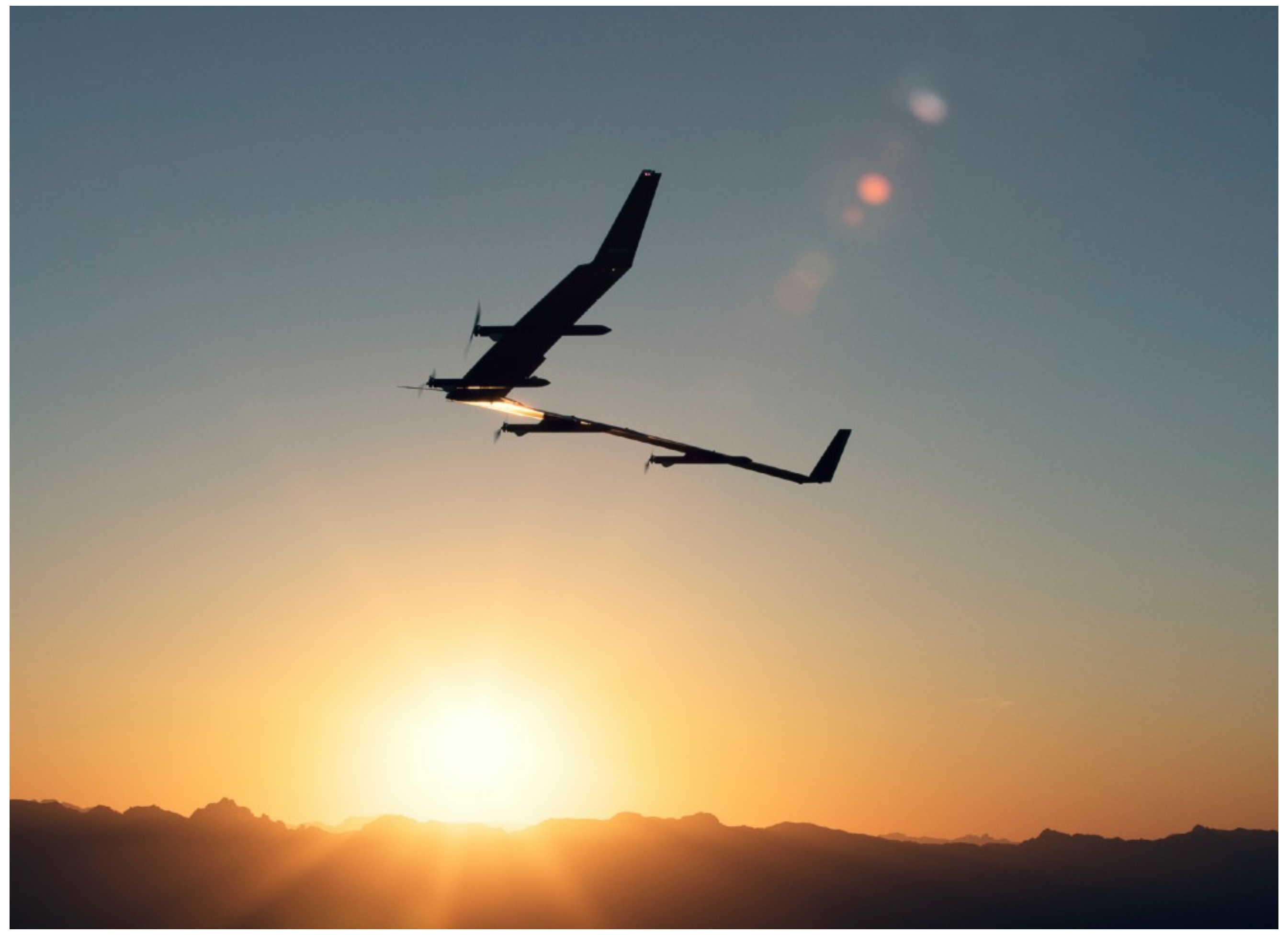
Figure 2: Mask criteria studied in this section, starting with large_final that was used in [5]. Names we use to refer to the various methods are given along with the formula that projects each (w_i, w_f) pair to a score. Weights with the largest scores (colored regions) are kept, and weights with the smallest scores (gray regions) are pruned. The x axis in each small figure is w_i and the y axis is w_f. In two methods, α is adjusted as needed to align percentiles between w_i and w_f. When masks are created, ties are broken randomly, so a score of 0 for every weight results in random masks.

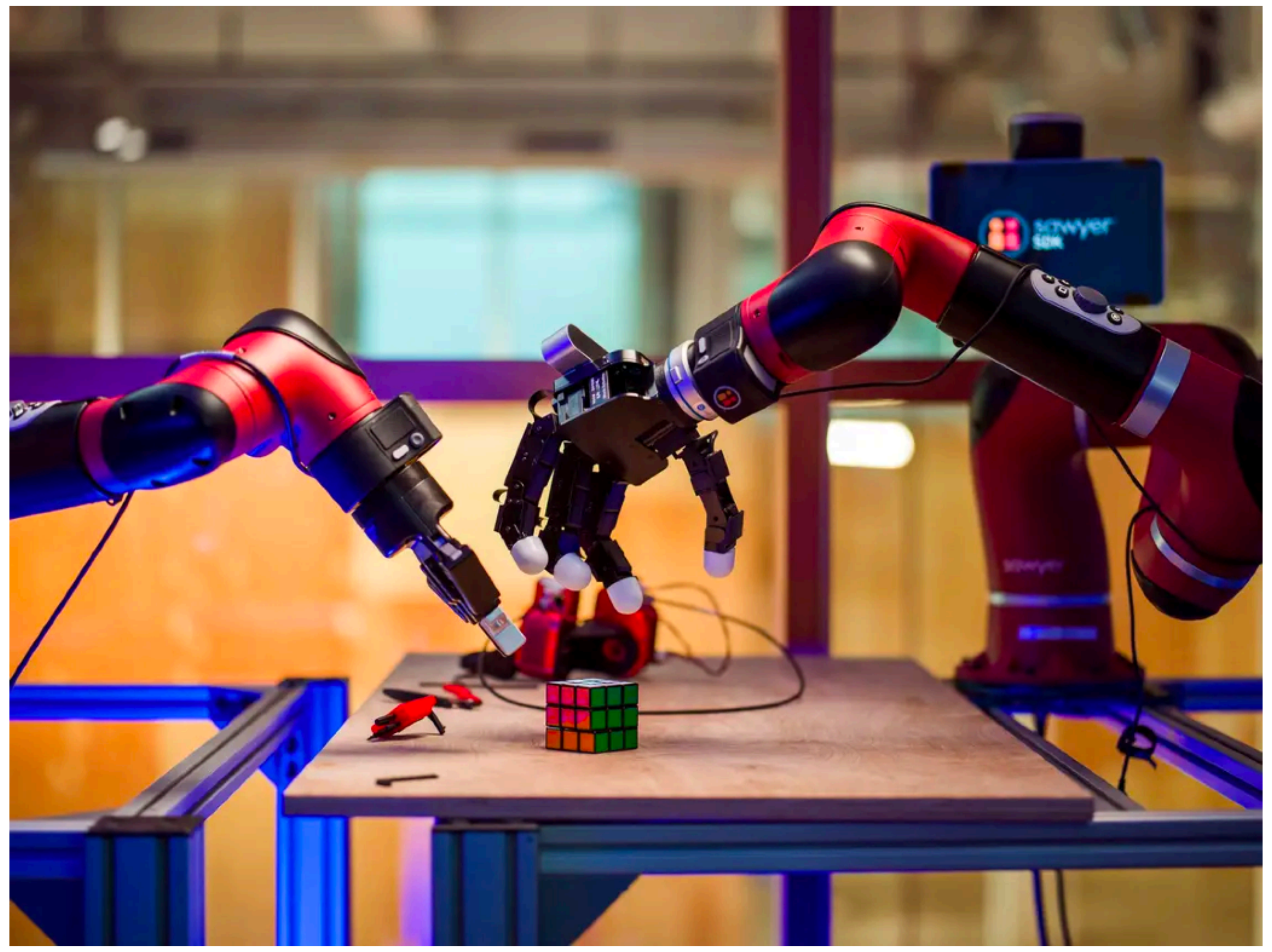
Zhou et al., 2019

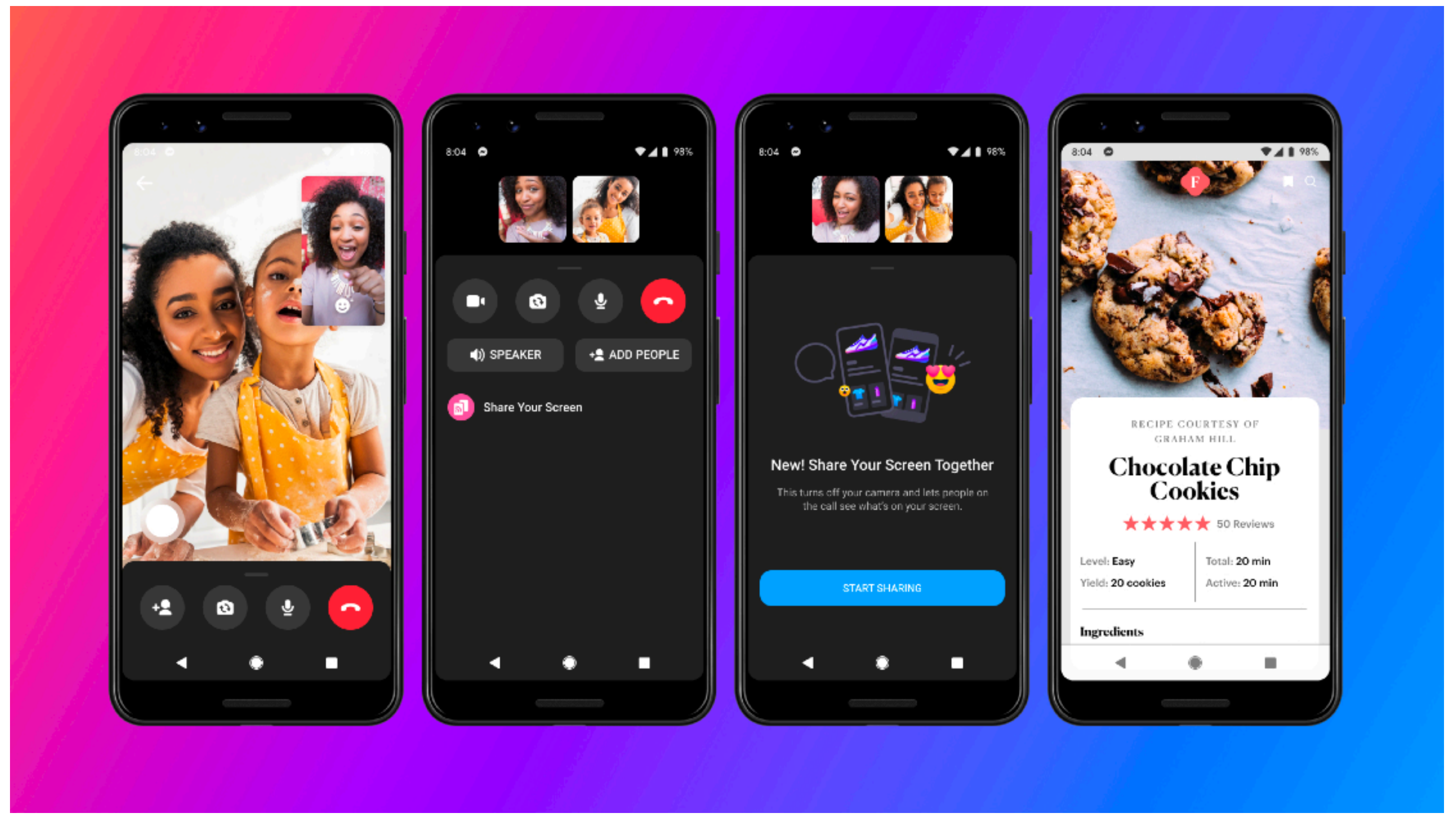
02 Pruning for applied research

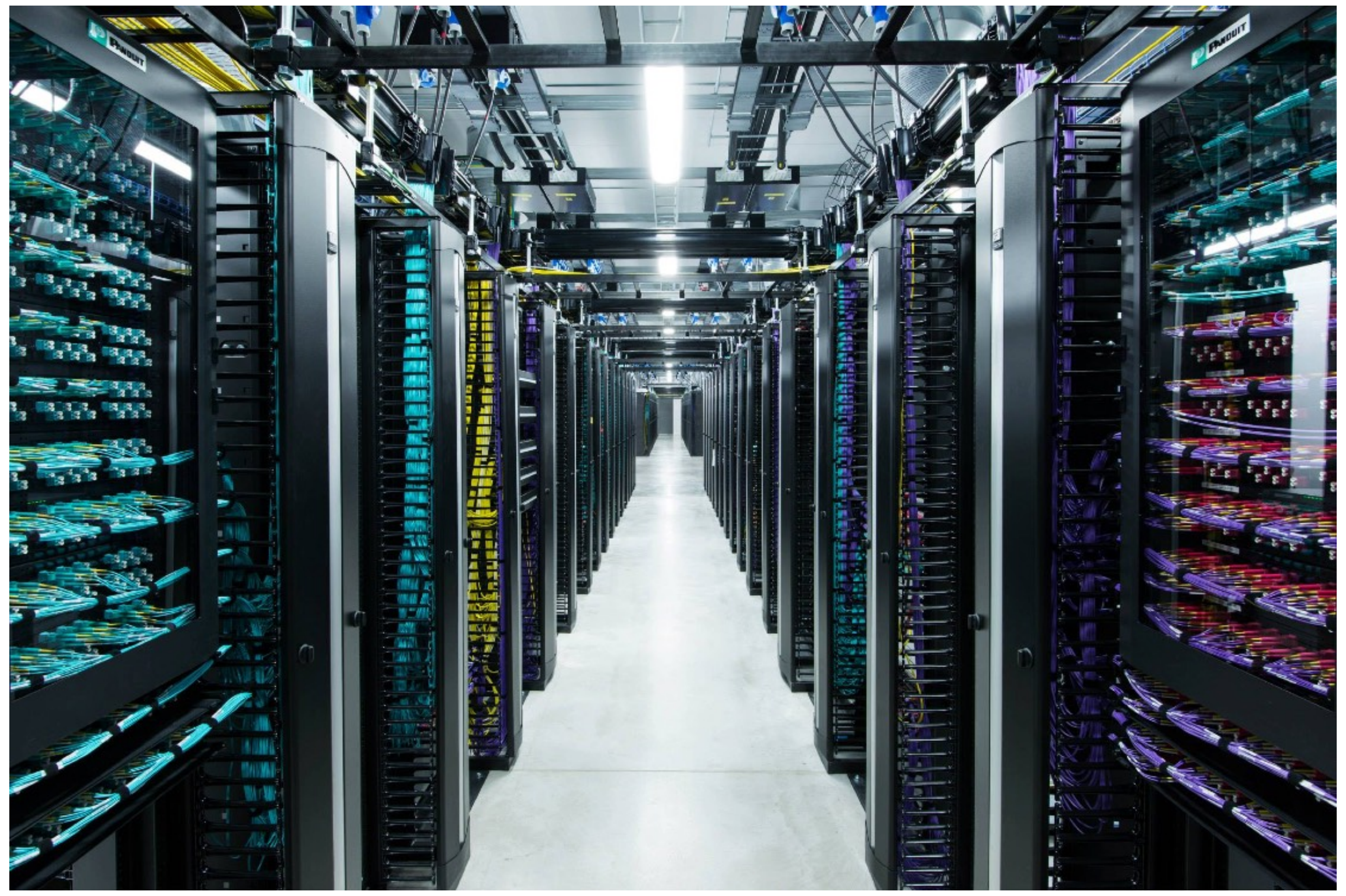
Pruning for applied research

Relevance to the outside world









Advantages

- Faster inference and/or training (depending on compression method and hardware type)
- Reduction in storage requirements
- In-memory computation
- Private on-device computation (mobile, AR/VR, IoT)
- Power savings
- Reduced heat dissipation in wearable devices
- Address some environmental concerns
- Lower barrier to entry in the field
- Way to test neuron importance assumptions
- ...

Advantages

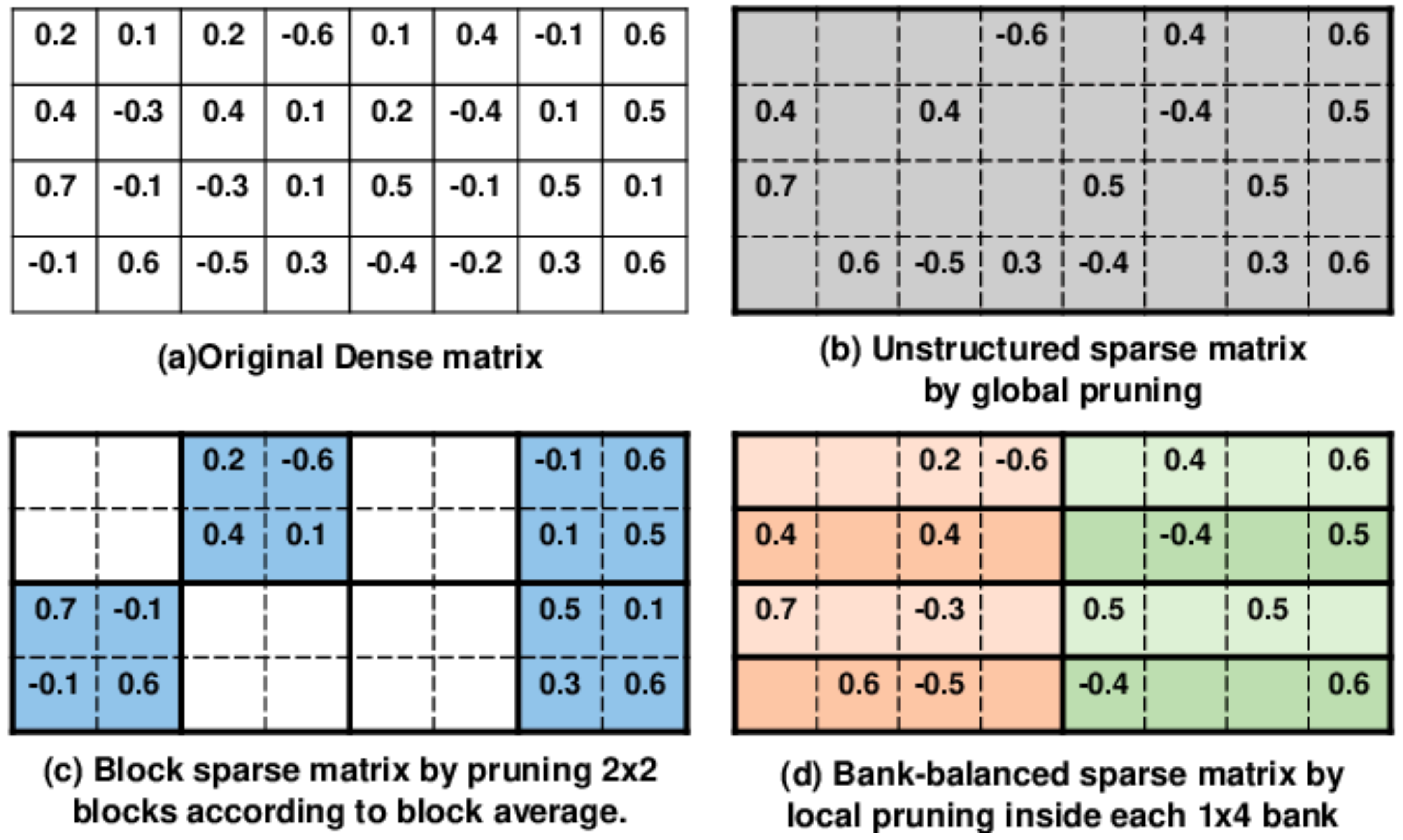
- Faster inference and/or training (depending on compression method and hardware type)
- Reduction in storage requirements
- In-memory computation
- Private on-device computation (mobile, AR/VR, IoT)
- Power savings
- Reduced heat dissipation in wearable devices
- Address some environmental concerns
- Lower barrier to entry in the field
- Way to test neuron importance assumptions
- …

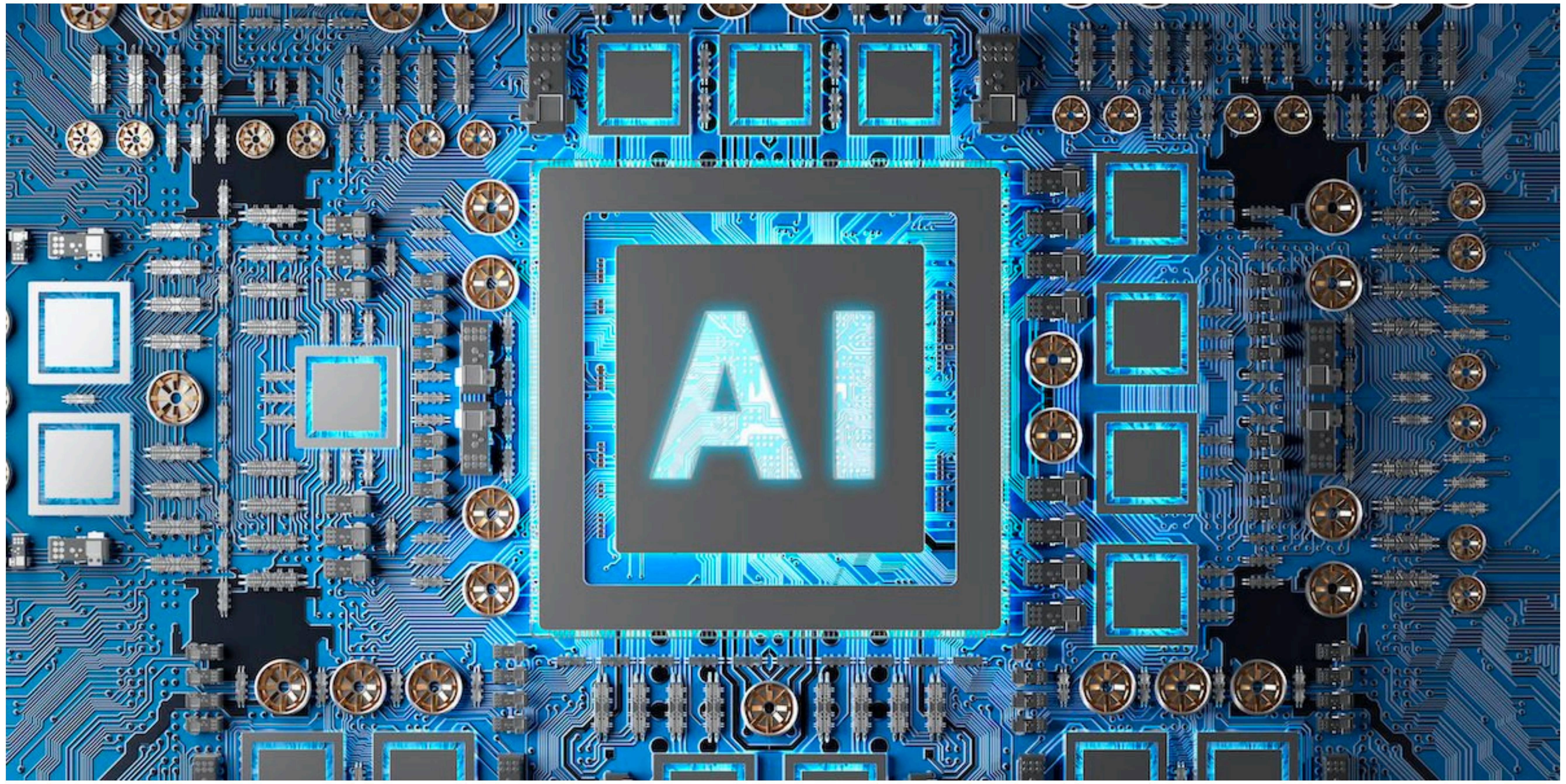
Disadvantages

- Fewer or no pre-trained versions available
- Poor quantification of impact of compression beyond overall accuracy (see fairness, bias, safety, robustness, etc.)
- Potentially no speed-up gains without custom hardware
- Hard to select compression method without exact knowledge of target hardware architecture
- Task dependence
- …

Delivering on Inference-time and Training-time Speed Ups

- Hard to exploit unstructured sparsity
- Best results for sparsity + quantization
- Active field of research!





02.5 PyTorch Pruning

`torch.nn.utils.prune`

Streamlining Tensor and Network Pruning in PyTorch, **Paganini** and Forde, arXiv:2004.13770 torch.nn.utils.prune

Different tensor pruning techniques enabled under a unified framework

BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

- CLASSMETHOD `apply(`*`module, name, *args, **kwargs`*`)` [SOURCE]
- `apply_mask(`*`module`*`)` [SOURCE]
- ABSTRACT `compute_mask(`*`t, default_mask`*`)` [SOURCE]
- `prune(`*`t, default_mask=None`*`)` [SOURCE]
- `remove(`*`module`*`)` [SOURCE]

New pruning technique?

Just subclass `BasePruningMethod` and implement `compute_mask`!

PruningContainer

CLASS `torch.nn.utils.prune.PruningContainer(`*`*args`*`)` [SOURCE]

Container holding a sequence of pruning methods for iterative pruning. Keeps track of the order in which pruning methods are applied and handles combining successive pruning calls.

Identity

CLASS `torch.nn.utils.prune.Identity` [SOURCE]

Utility pruning method that does not prune any units but generates the pruning parametrization with a mask of ones.

RandomUnstructured

CLASS `torch.nn.utils.prune.RandomUnstructured(`*`amount`*`)` [SOURCE]

Prune (currently unpruned) units in a tensor at random.

L1Unstructured

CLASS `torch.nn.utils.prune.L1Unstructured(`*`amount`*`)` [SOURCE]

Prune (currently unpruned) units in a tensor by zeroing out the ones with the lowest L1-norm.

RandomStructured

CLASS `torch.nn.utils.prune.RandomStructured(`*`amount, dim=-1`*`)` [SOURCE]

Prune entire (currently unpruned) channels in a tensor at random.

LnStructured

CLASS `torch.nn.utils.prune.LnStructured(`*`amount, n, dim=-1`*`)` [SOURCE]

Prune entire (currently unpruned) channels in a tensor based on their Ln-norm.

CustomFromMask

CLASS `torch.nn.utils.prune.CustomFromMask(`*`mask`*`)` [SOURCE]

torch.nn.utils.prune

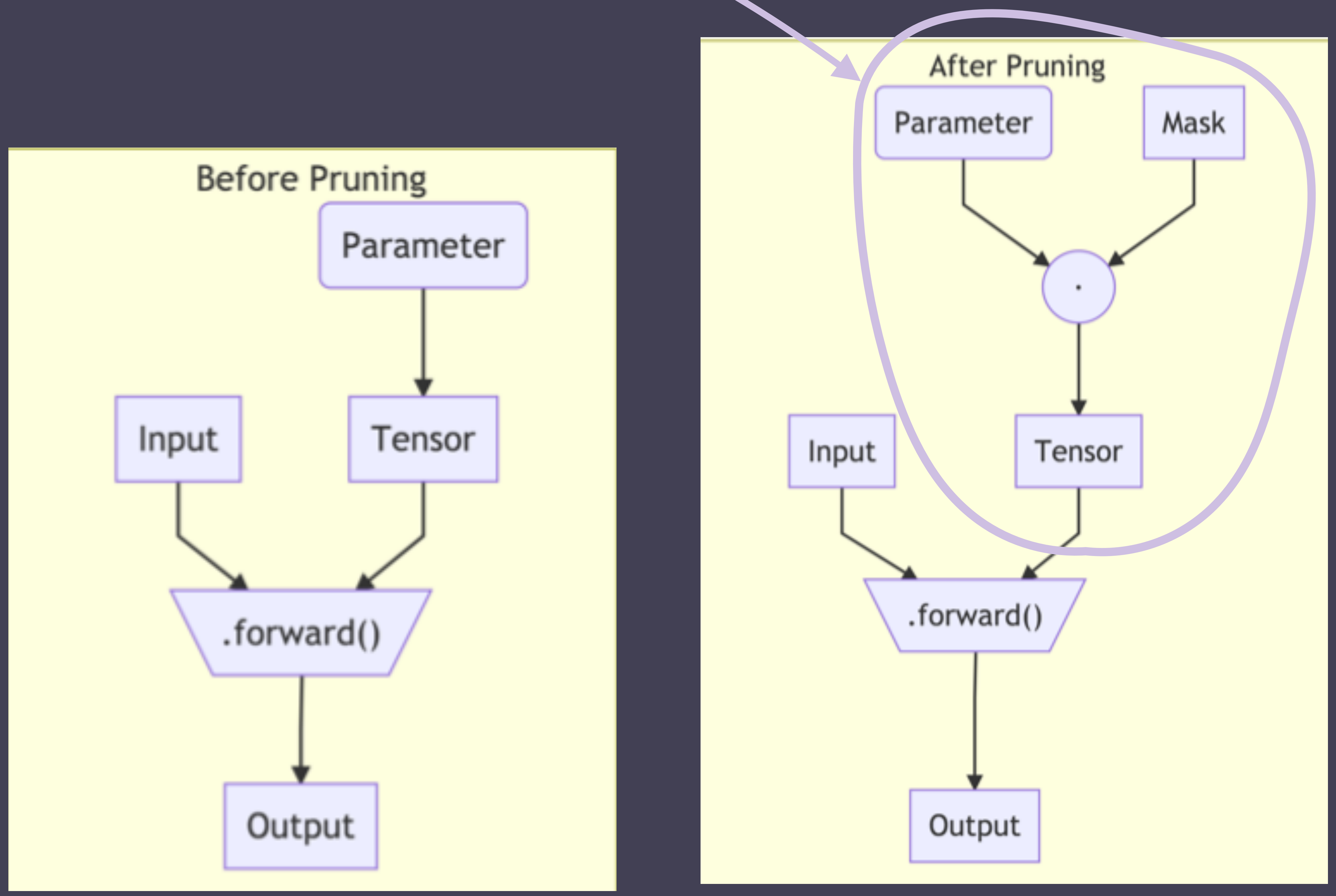
BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

- CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]
- `apply_mask(module)` [SOURCE]
- ABSTRACT `compute_mask(t, default_mask)` [SOURCE]
- `prune(t, default_mask=None)` [SOURCE]
- `remove(module)` [SOURCE]

Fetches the mask and the original, unpruned tensor to compute the pruned tensor during the forward pass → op is accounted for in the backward pass, too



PruningContainer

Michela Paganini

`torch.nn.utils.prune`

BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]

`apply_mask(module)` [SOURCE]

ABSTRACT `compute_mask(t, default_mask)` [SOURCE]

`prune(t, default_mask=None)` [SOURCE]

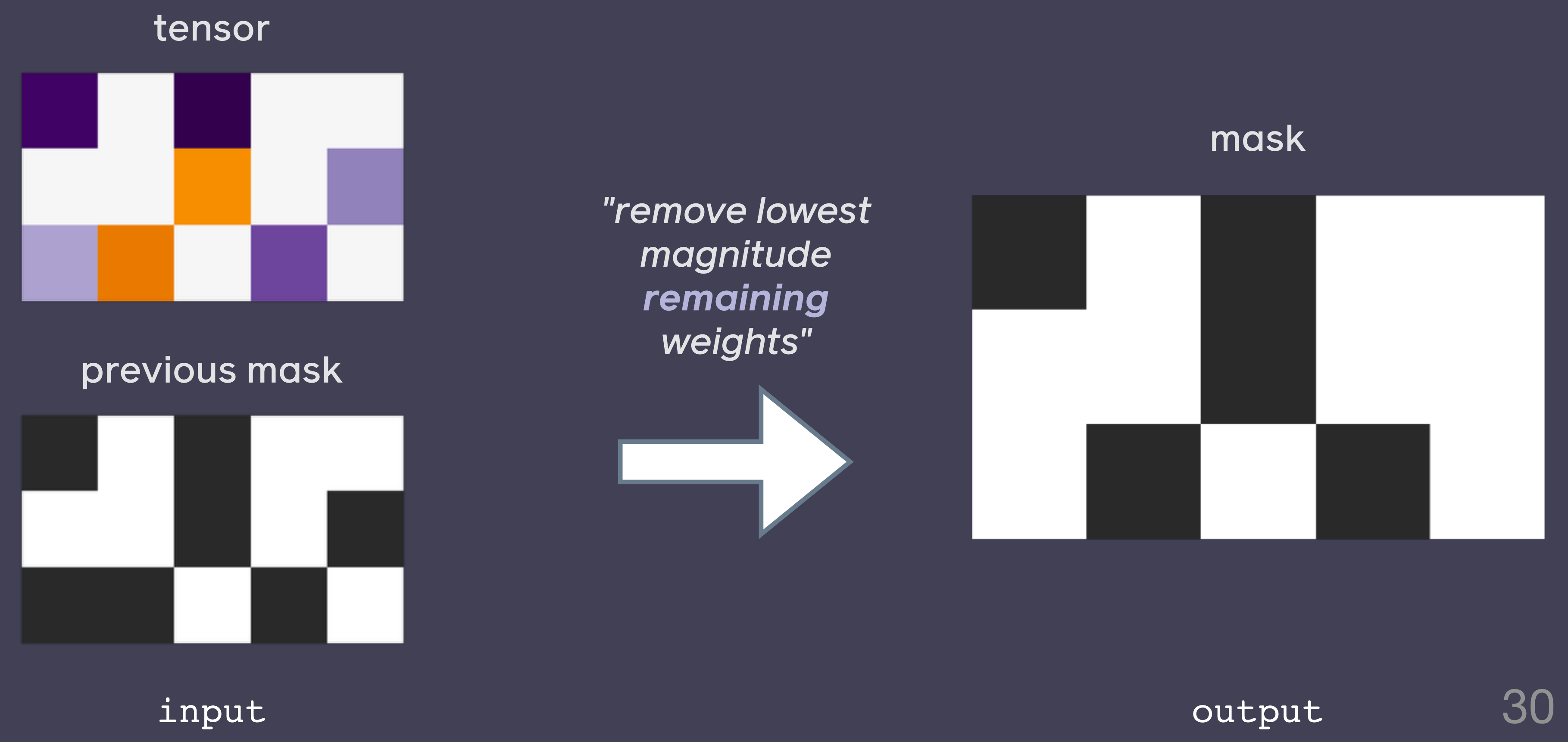
`remove(module)` [SOURCE]

defines the interface → concrete subclasses must implement the logic

For example, in `prune.L1Unstructured`:

implements the logic that defines which portions of the tensors will be zeroed out while accounting for previously pruned entries

(through a `prune.PruningContainer`) it handles the case in which the tensor had previously been pruned by computing the valid entries in the tensor that can still be pruned and then applying the new pruning technique exclusively on those entries



torch.nn.utils.prune

Easy to use

```
model = LeNet()  # unpruned model

# L_2 structured pruning will remove 50% of channels across axis 0
prune.ln_structured(
    module=model.conv1,
    name="weight",
    amount=0.5,
    n=2,
    dim=0
)
```

Iterative pruning made easy

`prune.PruningContainer` handles the combination of successive masks for you

```
for _ in range(10):
    # Remove 2 connections per iteration
    prune.l1_unstructured(module=model.fc1, name="bias", amount=2)
```

Global pruning made easy

```
parameters_to_prune = (
    (model.conv1, "weight"),
    (model.conv2, "weight"),
    (model.fc1, "weight"),
)

prune.global_unstructured(
    parameters_to_prune,
    pruning_method=prune.L1Unstructured,
    amount=0.2,
)
```

Easy to extend

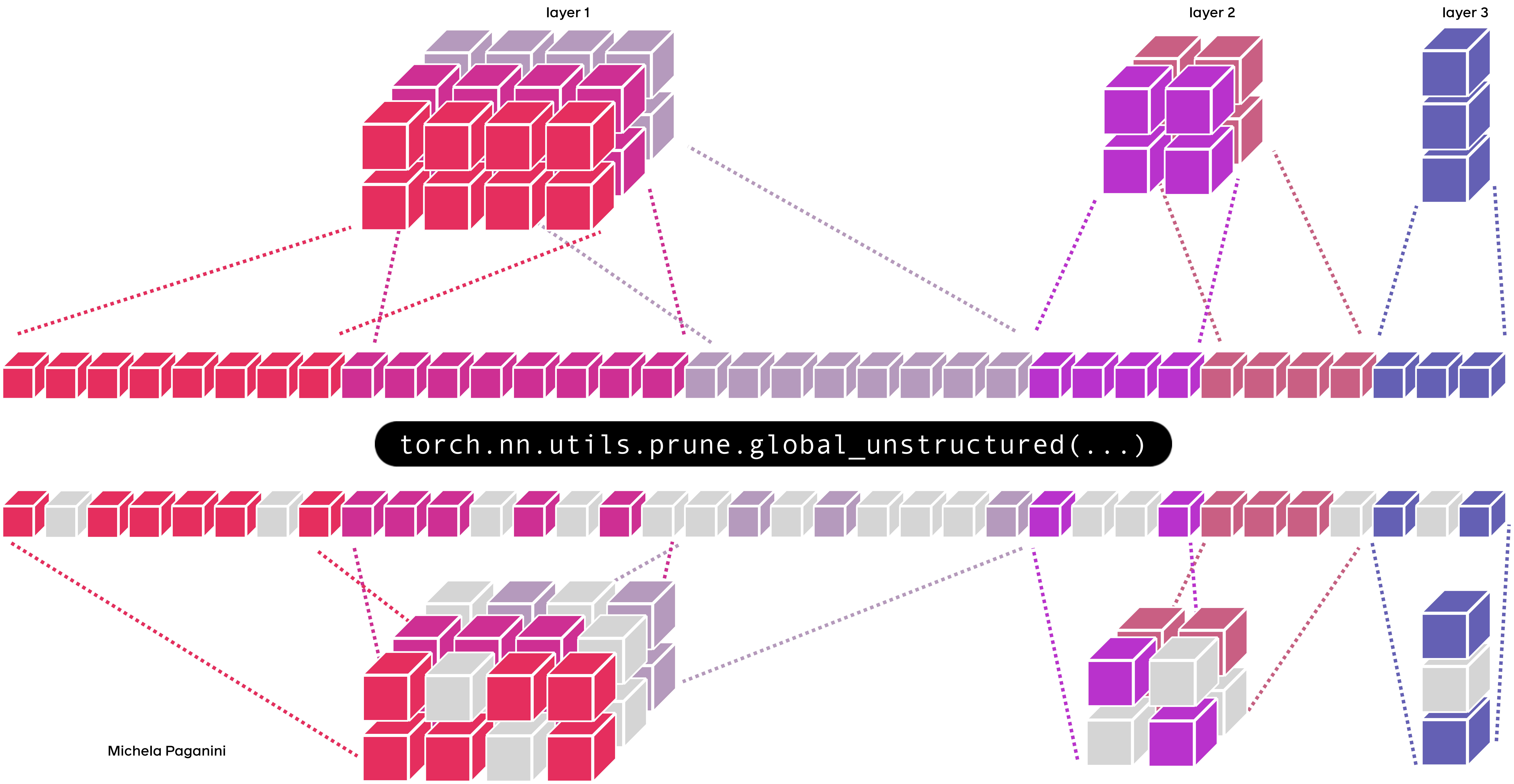
```
class FooBarPruningMethod(prune.BasePruningMethod):
    """Prune every other entry in a tensor
    """
    PRUNING_TYPE = 'unstructured'

    def compute_mask(self, t, default_mask):
        mask = default_mask.clone()
        mask.view(-1)[::2] = 0
        return mask
```

supports 3 PRUNING_TYPEs: `'global'`, `'structured'`, and `'unstructured'` (to determine how to combine masks if pruning is applied iteratively)

instructions on how to compute the mask for the given tensor according to the logic of your pruning technique

```
def foobar_unstructured(module, name):
    FooBarPruningMethod.apply(module, name)
    return module
```

GlobalPruning



Michela Paganini

torch.nn.utils.prune

BasePruningMethod

CLASS `torch.nn.utils.prune.BasePruningMethod` [SOURCE]

Abstract base class for creation of new pruning techniques.

CLASSMETHOD `apply(module, name, *args, **kwargs)` [SOURCE]

`apply_mask(module)` [SOURCE]

ABSTRACT `compute_mask(t, default_mask)` [SOURCE]

`prune(t, default_mask=None)` [SOURCE]

`remove(module)` [SOURCE]

`torch.nn.utils.prune` is designed to act on a `torch.nn.Module`

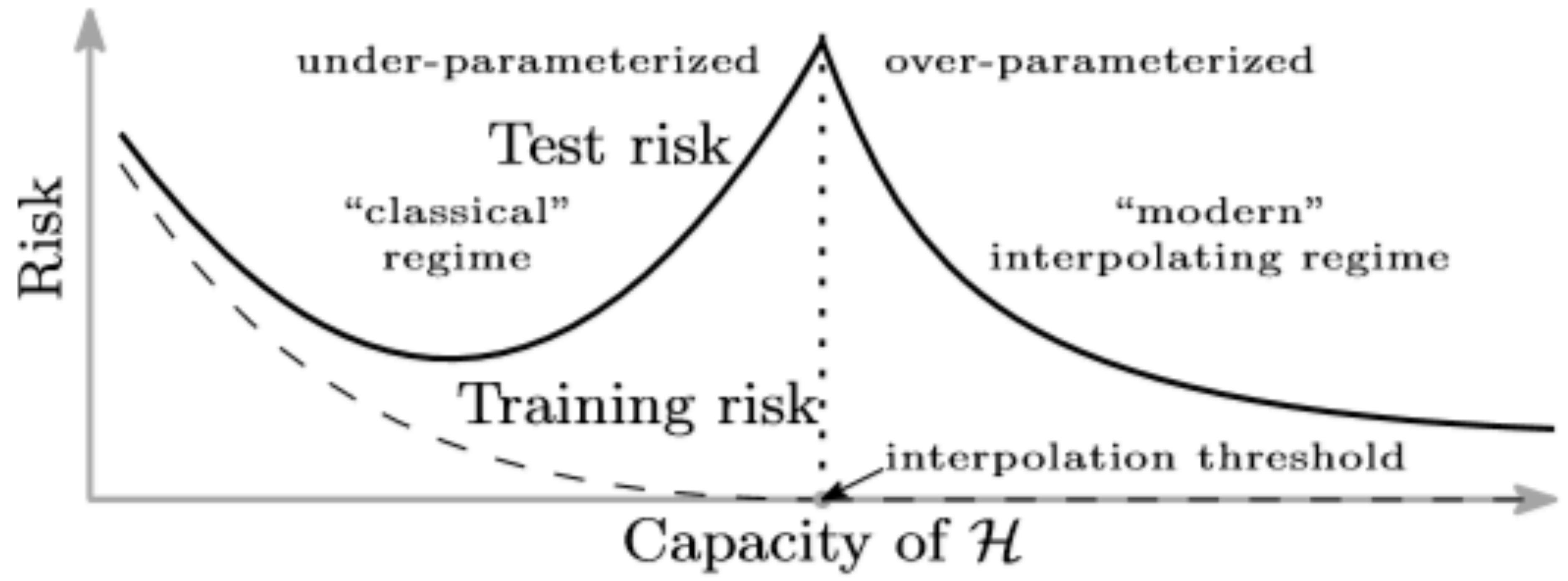
provides an interface for acting directly on a tensor

```
tensor = torch.randn([3, 5])
p = torch.nn.utils.prune.LnStructured(amount=1, dim=1, n=2)
masked_tensor = p.prune(tensor)
```

`torch.nn.utils.prune`

03 Pruning for fundamental research

Network capacity and over-parametrization



Belkin et al., 2018

Network capacity and over-parametrization

Capacity or complexity are hard to measure:

- number of parameters?
- complexity of function?
- rank?
- norm?
- a function of the architecture and the optimization algorithm?

Related Work

EMPIRICAL ANALYSIS OF THE HESSIAN OF OVER-PARAMETRIZED NEURAL NETWORKS

Levent Sagun[1], **Utku Evci**[2], **V. Uğur Güney**[3], **Yann Dauphin**[4], **Léon Bottou**[4]
[1] Institut de Physique Théorique, Université Paris Saclay, CEA
[2] Computer Science Department, NYU
[3] Data Engineer at Facebook, New York
[4] Facebook AI Research, New York

Opening the black box of Deep Neural Networks via Information

Ravid Schwartz-Ziv RAVID.ZIV@MAIL.HUJI.AC.IL
Edmond and Lilly Safra Center for Brain Sciences
The Hebrew University of Jerusalem
Jerusalem, 91904, Israel

Naftali Tishby* TISHBY@CS.HUJI.AC.IL
School of Engineering and Computer Science
and Edmond and Lilly Safra Center for Brain Sciences
The Hebrew University of Jerusalem
Jerusalem, 91904, Israel

MEASURING THE INTRINSIC DIMENSION OF OBJECTIVE LANDSCAPES

Chunyuan Li *
Duke University
cl319@duke.edu

Heerad Farkhoor, Rosanne Liu, and Jason Yosinski
Uber AI Labs
{heerad,rosanne,yosinski}@uber.com

On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization

Sanjeev Arora [1,2] **Nadav Cohen** [2] **Elad Hazan** [1,3]

GRADIENT DESCENT HAPPENS IN A TINY SUBSPACE

Guy Gur-Ari*
School of Natural Sciences
Institute for Advanced Study
Princeton, NJ 08540, USA
guyg@ias.edu

Daniel A. Roberts*
Facebook AI Research
New York, NY 10003, USA
danr@fb.com

Ethan Dyer
Johns Hopkins University
Baltimore, MD 21218, USA
edyer4@jhu.edu

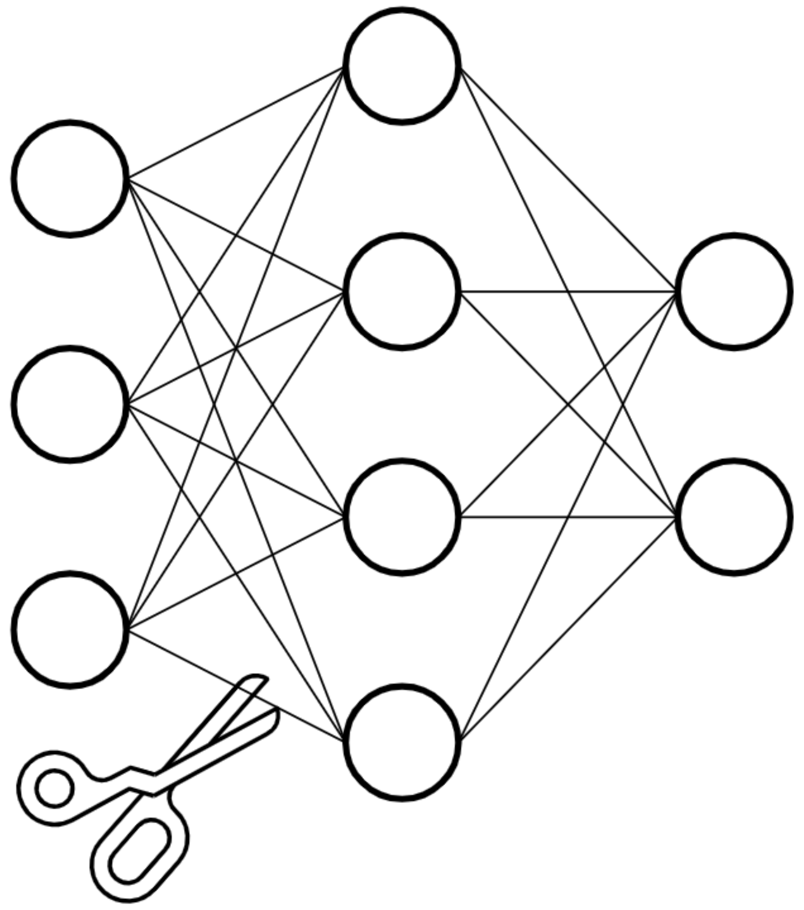
… and a lot more!

03 The lottery ticket hypothesis (LTH)

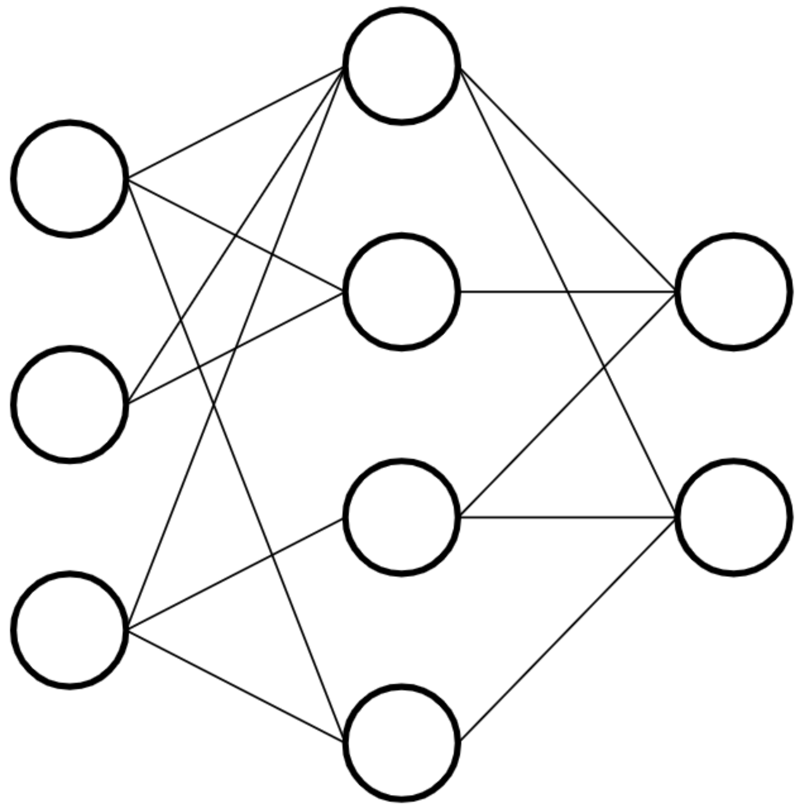
Frankle & Carbin, MIT [arXiv:1803.03635]

Frankle & Carbin, 2018 `[arXiv:1803.03635]`

- Contrary to prior belief, there exists a subset of small, sparse networks that can be successfully trained from scratch despite their low number of parameters



Before pruning



After pruning

accuracy = a_0

accuracy ~ a_0

What does it say?

Frankle & Carbin, 2018 [arXiv:1803.03635]

- Lucky sub-networks can be found with iterative magnitude-based pruning with weight rewinding to initialization

Algorithm 1 Iterative Magnitude Pruning (IMP) with rewinding to iteration k.

1: Randomly initialize a neural network $f(x; m \odot W_0)$ with initial trivial pruning mask $m = 1^{|W_0|}$.
2: Train the network for k iterations, producing network $f(x; m \odot W_k)$.
3: Train the network for $T - k$ further iterations, producing network $f(x; m \odot W_t)$.
4: Prune the remaining entries with the lowest magnitudes from W_T. That is, let $m[i] = 0$ if $W_T[i]$ is pruned.
5: If satisfied, the resulting network is $f(x; m \odot W_T)$.
6: Otherwise, reset W to W_k and repeat steps 3-5 iteratively, gradually removing more of the network.

k = 0 in original formulation

What does it say?

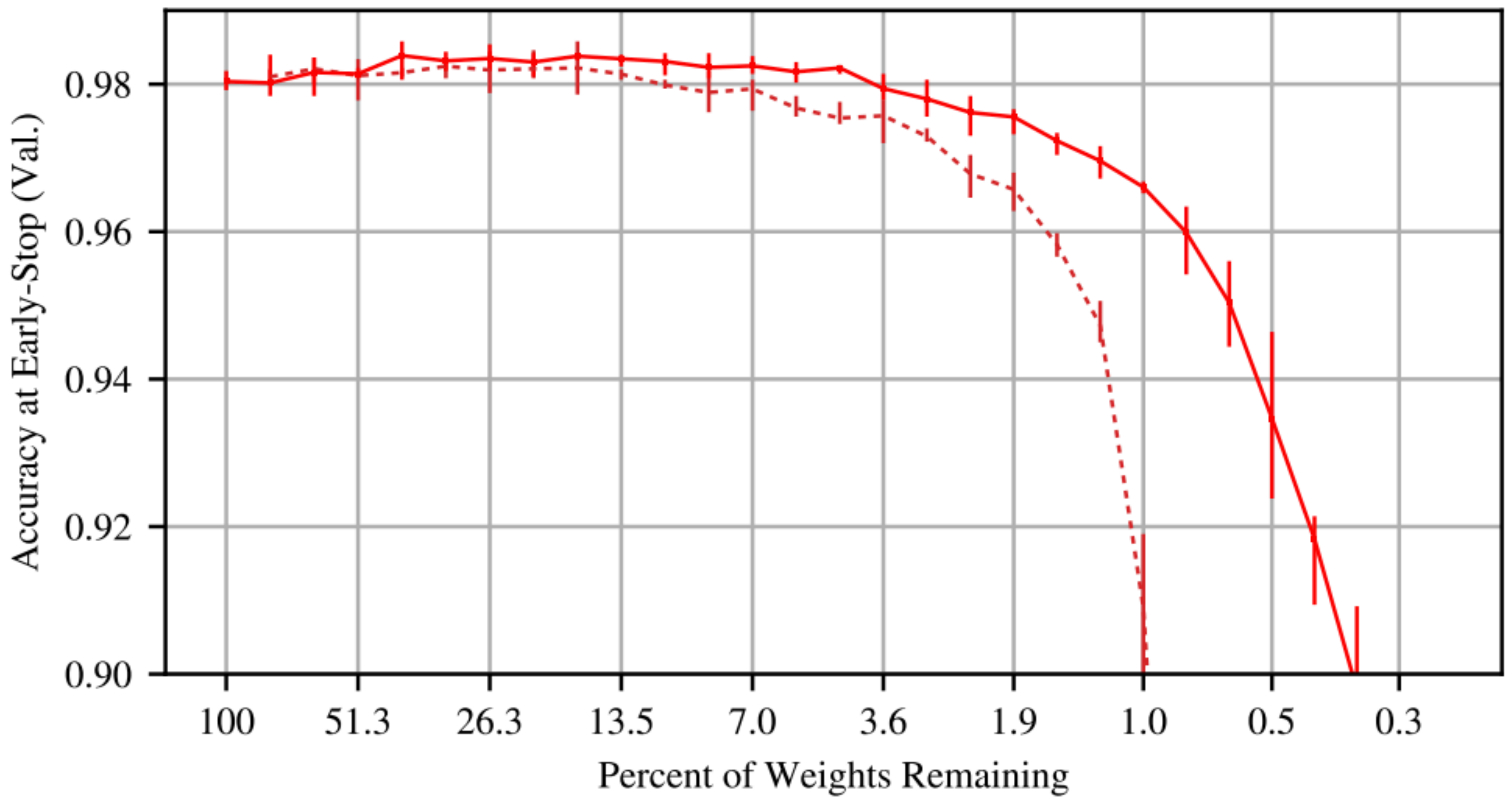
Frankle & Carbin, 2018 `[arXiv:1803.03635]`

- **Lucky sub-networks can be found with iterative magnitude-based pruning with weight rewinding to initialization**

Algorithm 1 Iterative Magnitude Pruning (IMP) with rewinding to iteration k.

1: Randomly initialize a neural network $f(x; m \odot W_0)$ with initial trivial pruning mask $m = 1^{|W_0|}$.
2: Train the network for k iterations, producing network $f(x; m \odot W_k)$.
3: Train the network for $T - k$ further iterations, producing network $f(x; m \odot W_t)$.
4: Prune the remaining entries with the lowest magnitudes from W_T. That is, let $m[i] = 0$ if $W_T[i]$ is pruned.
5: If satisfied, the resulting network is $f(x; m \odot W_T)$.
6: Otherwise, reset W to W_k and repeat steps 3-5 iteratively, gradually removing more of the network.

k = 0 in original formulation



See also:

- *Comparing Rewinding and Fine-tuning in Neural Network Pruning*, Renda et al. arXiv:2003.02389

- *On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks*, **Paganini** and Forde arXiv:2001.05050

04 Digging deeper into the LTH

Questions

Do lottery tickets require magnitude based pruning?

How similar are lottery tickets from different pruning techniques?

How similar are lottery tickets from different datasets and optimizers?

Do lottery tickets transfer?

How do unpruned weights evolve?

What patterns emerge in the structure of unpruned pathways?

Can we combine lottery tickets to speed up their discovery?

Can we find lottery tickets at initialization?

Can all pruning techniques find winning tickets?

On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks

Michela Paganini
Facebook AI Research
michela@fb.com

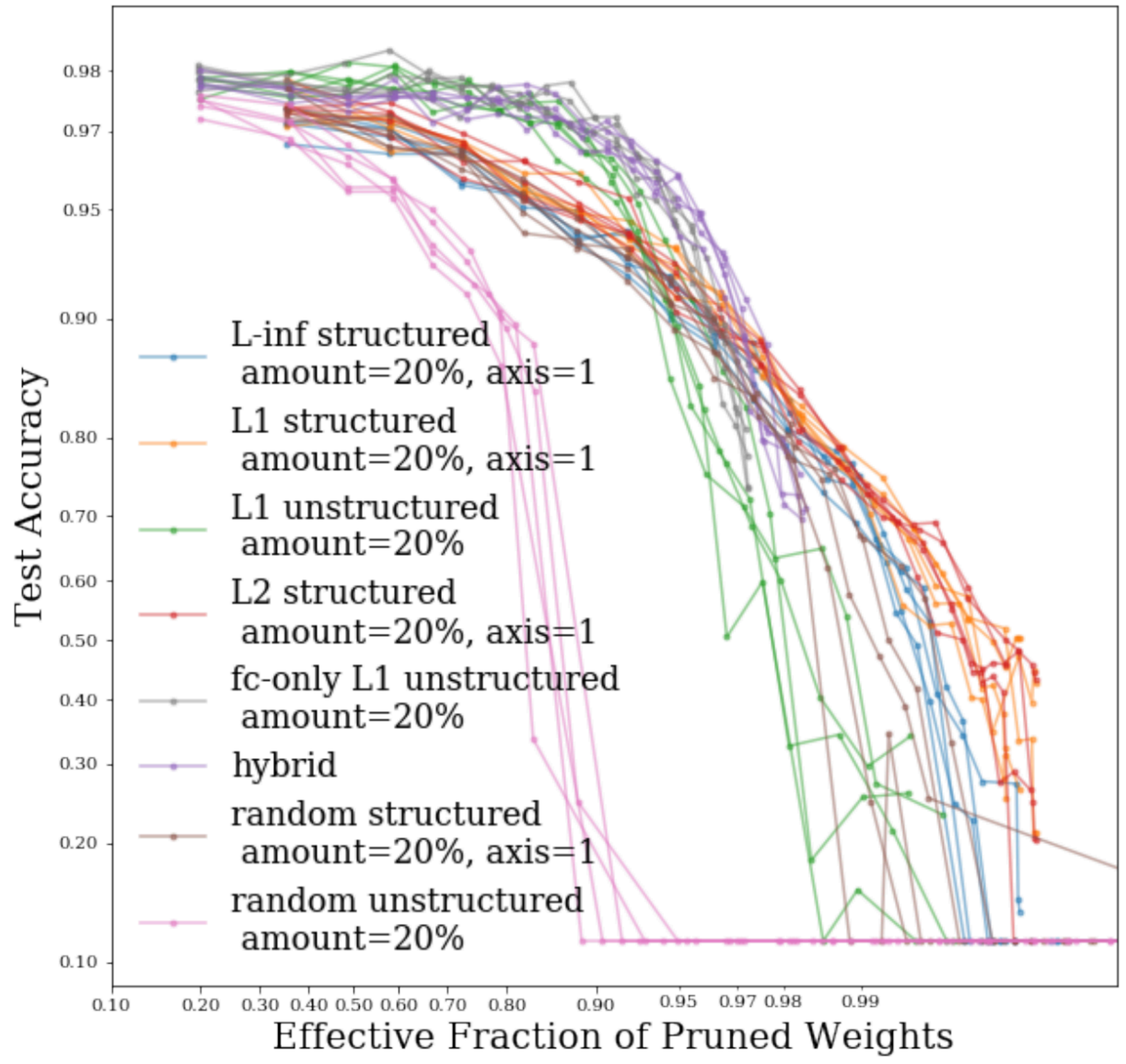
Jessica Forde *
Brown University
jessica_forde@brown.edu

Abstract

We examine how recently documented, fundamental phenomena in deep learning models subject to pruning are affected by changes in the pruning procedure. Specifically, we analyze differences in the connectivity structure and learning dynamics of pruned models found through a set of common iterative pruning techniques, to address questions of uniqueness of trainable, high-sparsity subnetworks, and their dependence on the chosen pruning method. In convolutional layers, we document the emergence of structure induced by magnitude-based unstructured pruning in conjunction with weight rewinding that resembles the effects of structured pruning. We also show empirical evidence that weight stability can be automatically achieved through apposite pruning techniques.

On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks, **Paganini** and Forde arXiv:2001.05050

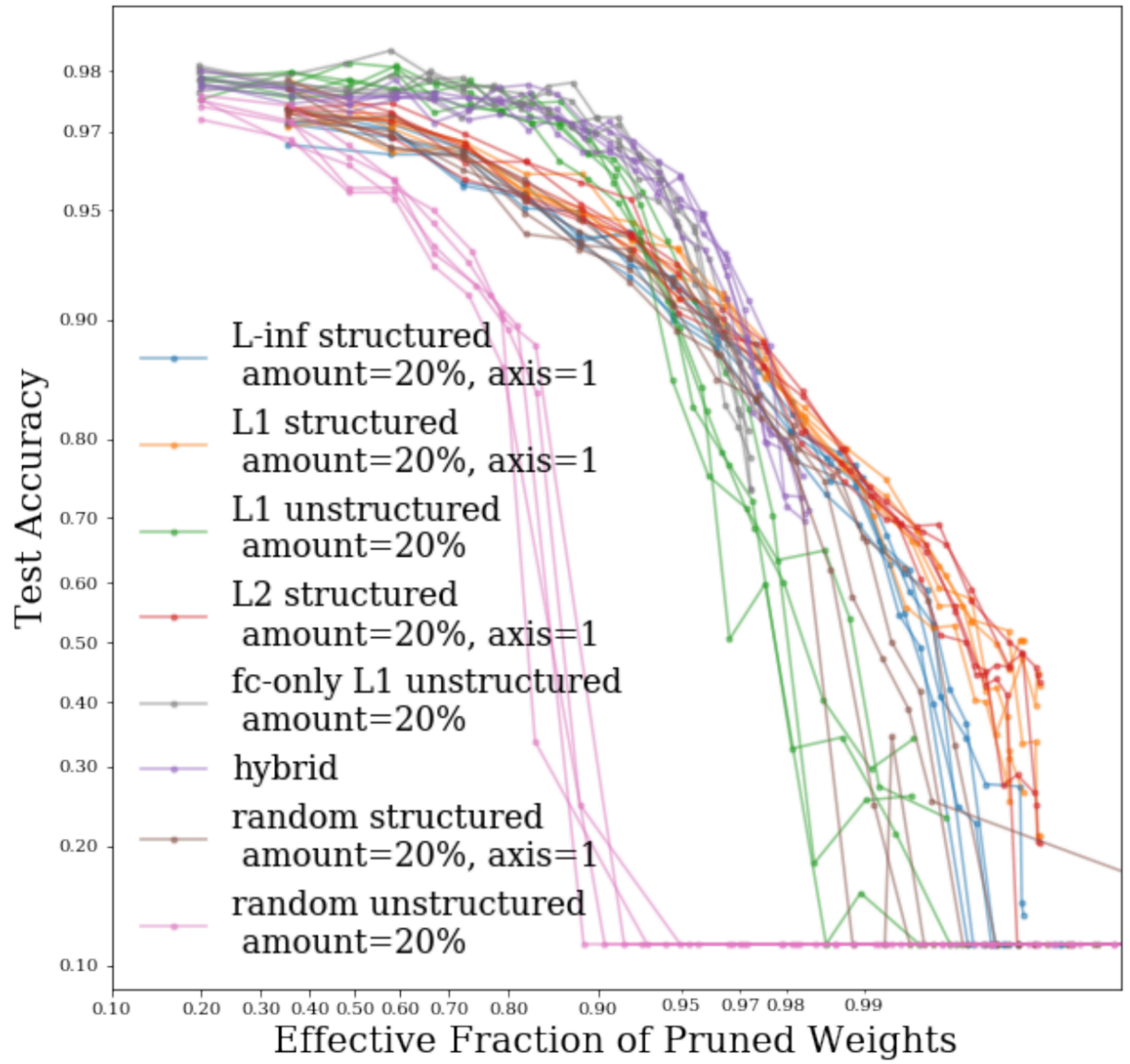
Can all pruning techniques find winning tickets?

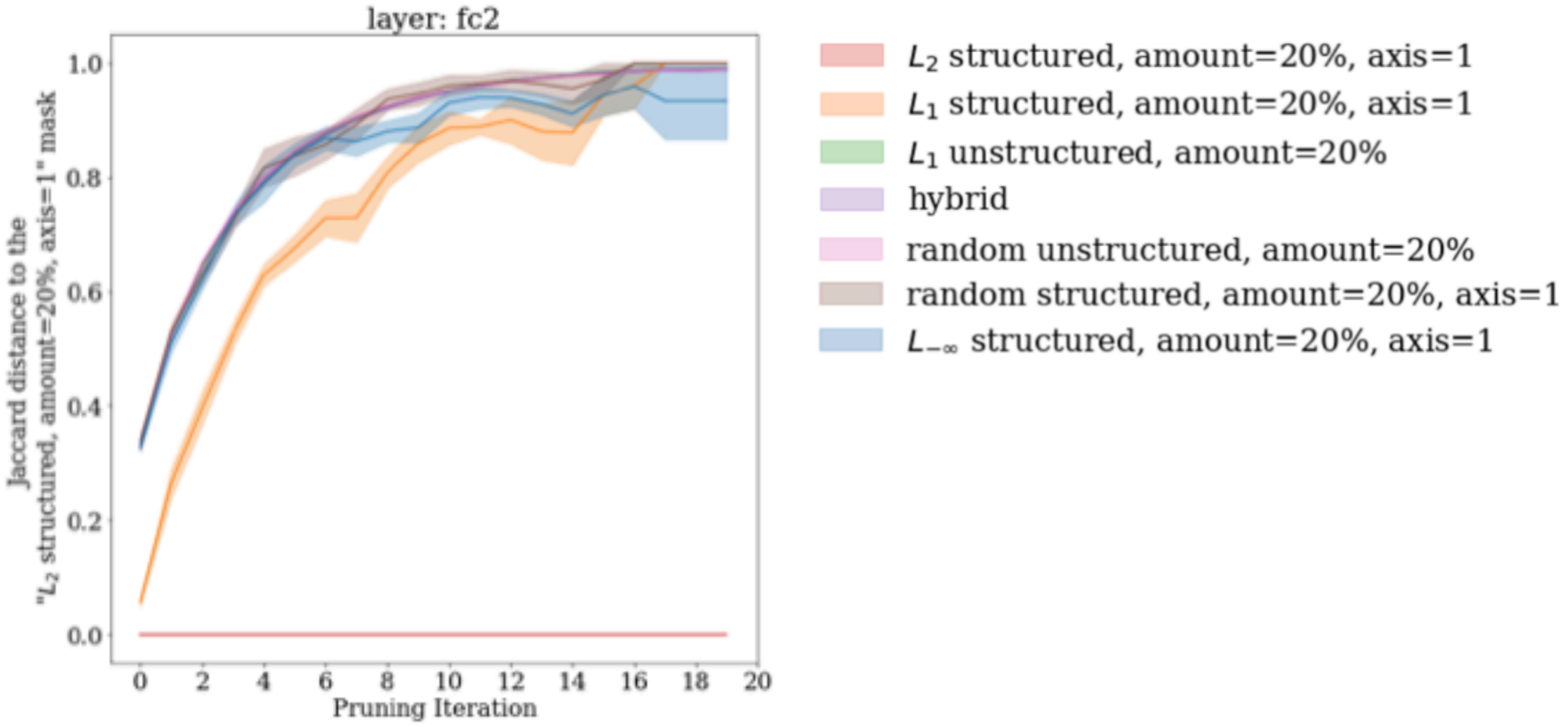


On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks, **Paganini** and Forde arXiv:2001.05050

Can all pruning techniques find winning tickets?
Do different pruning techniques agree on what subnetwork is a winning ticket?

Not exactly. Measure difference in terms of Jaccard distance: $d_J(\mathbb{M}_1, \mathbb{M}_2) = 1 - \frac{|\mathbb{M}_1 \cap \mathbb{M}_2|}{|\mathbb{M}_1 \cup \mathbb{M}_2|}$





On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks, **Paganini** and Forde arXiv:2001.05050

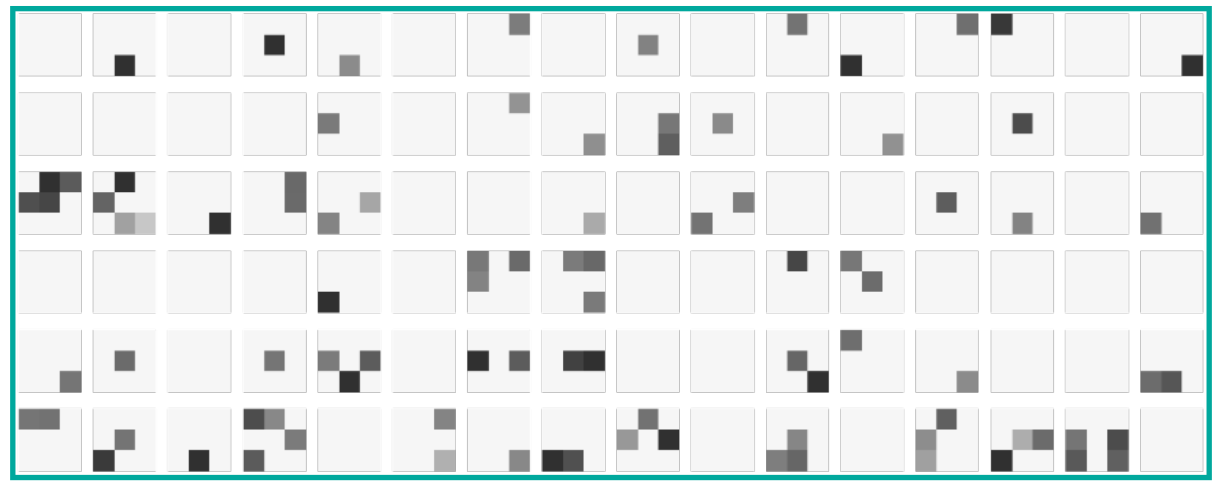
Observations

- lottery ticket-style weight rewinding, coupled with unstructured pruning, gives rise to connectivity patterns similar to structured pruning (~feature selection). Not true for finetuning.

LeNet conv1 weights

structured pruning + rewinding

unstructured pruning + rewinding



unstructured pruning + finetuning

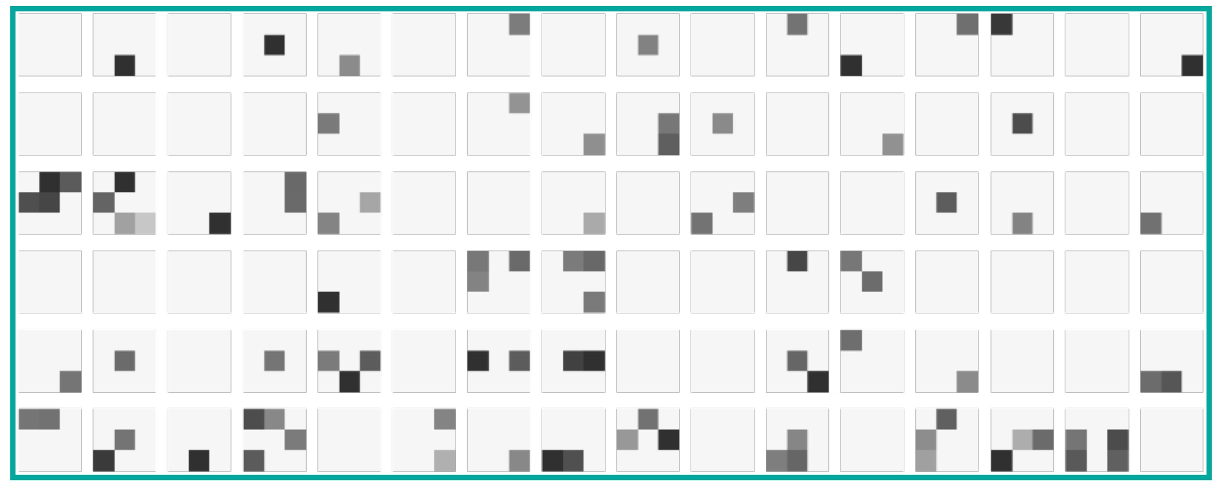
Observations

- lottery ticket-style weight rewinding, coupled with unstructured pruning, gives rise to connectivity patterns similar to structured pruning (~feature selection). Not true for finetuning.

LeNet conv1 weights

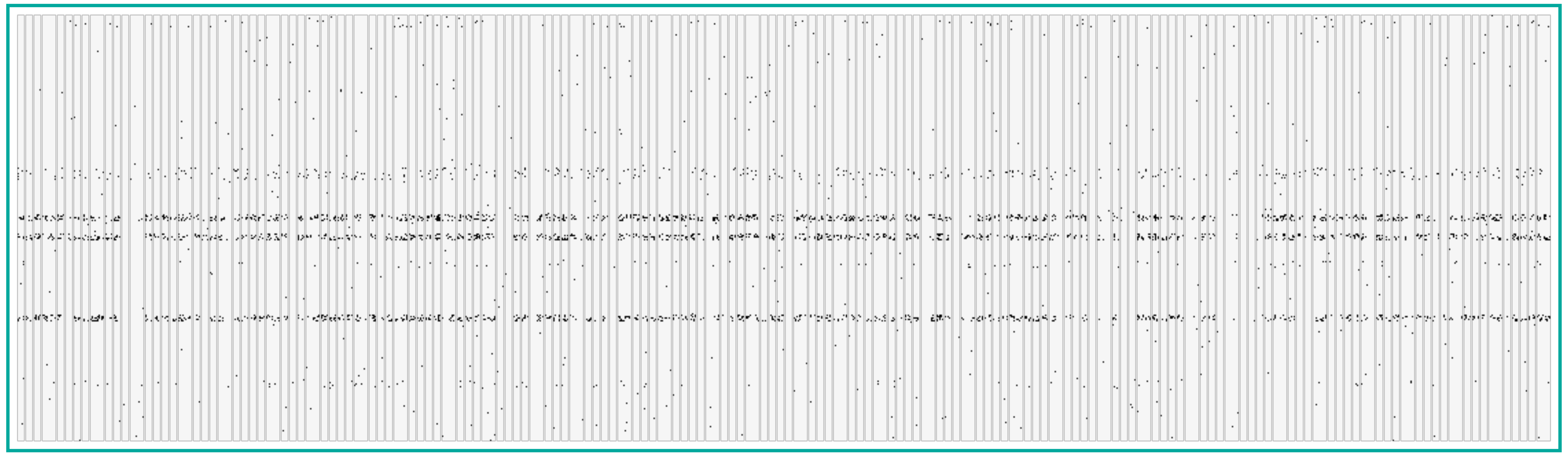
structured pruning + rewinding

unstructured pruning + rewinding



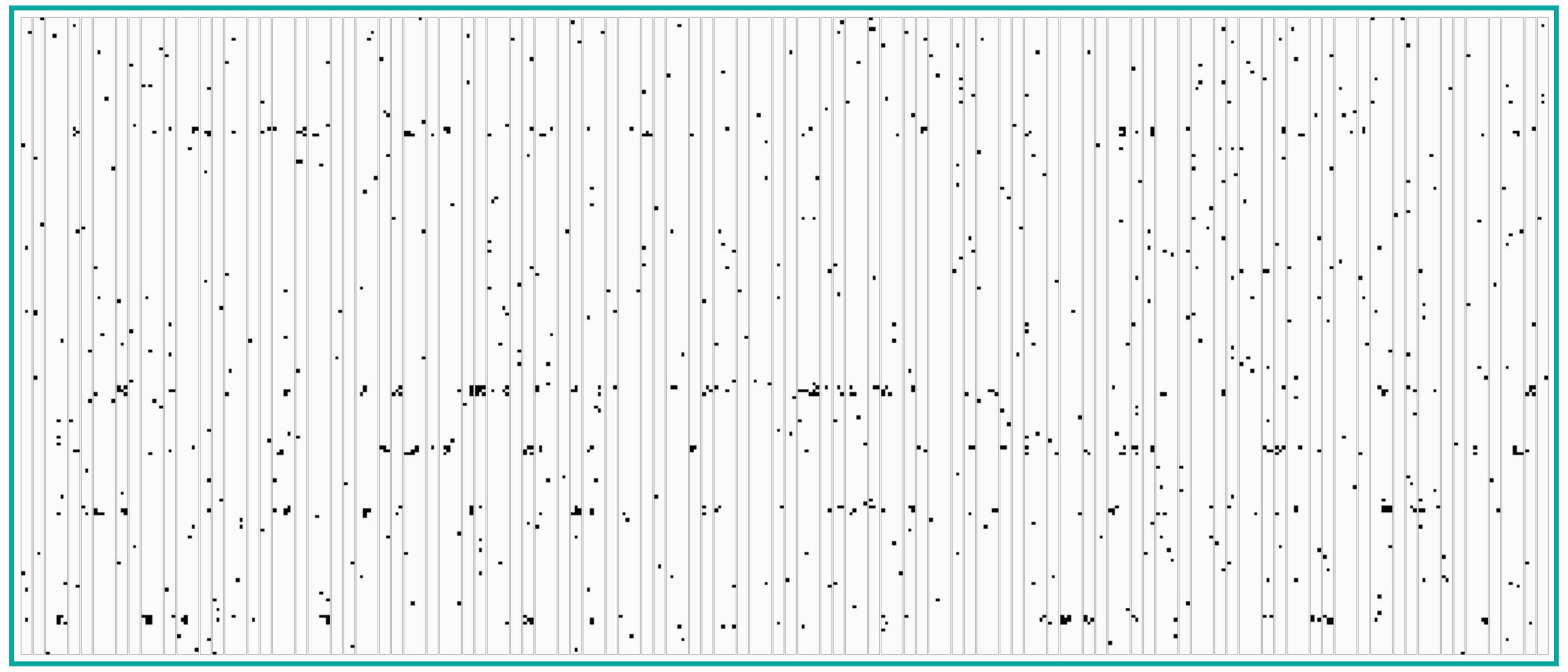
unstructured pruning + finetuning

AlexNet conv2 weights



unstructured pruning + rewinding

VGG11 conv2 weights



unstructured pruning + rewinding

Do lottery tickets found on a task transfer to another task? How similar are they?

One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers

Ari S. Morcos*
Facebook AI Research
arimorcos@fb.com

Haonan Yu
Facebook AI Research
haonanu@gmail.com

Michela Paganini
Facebook AI Research
michela@fb.com

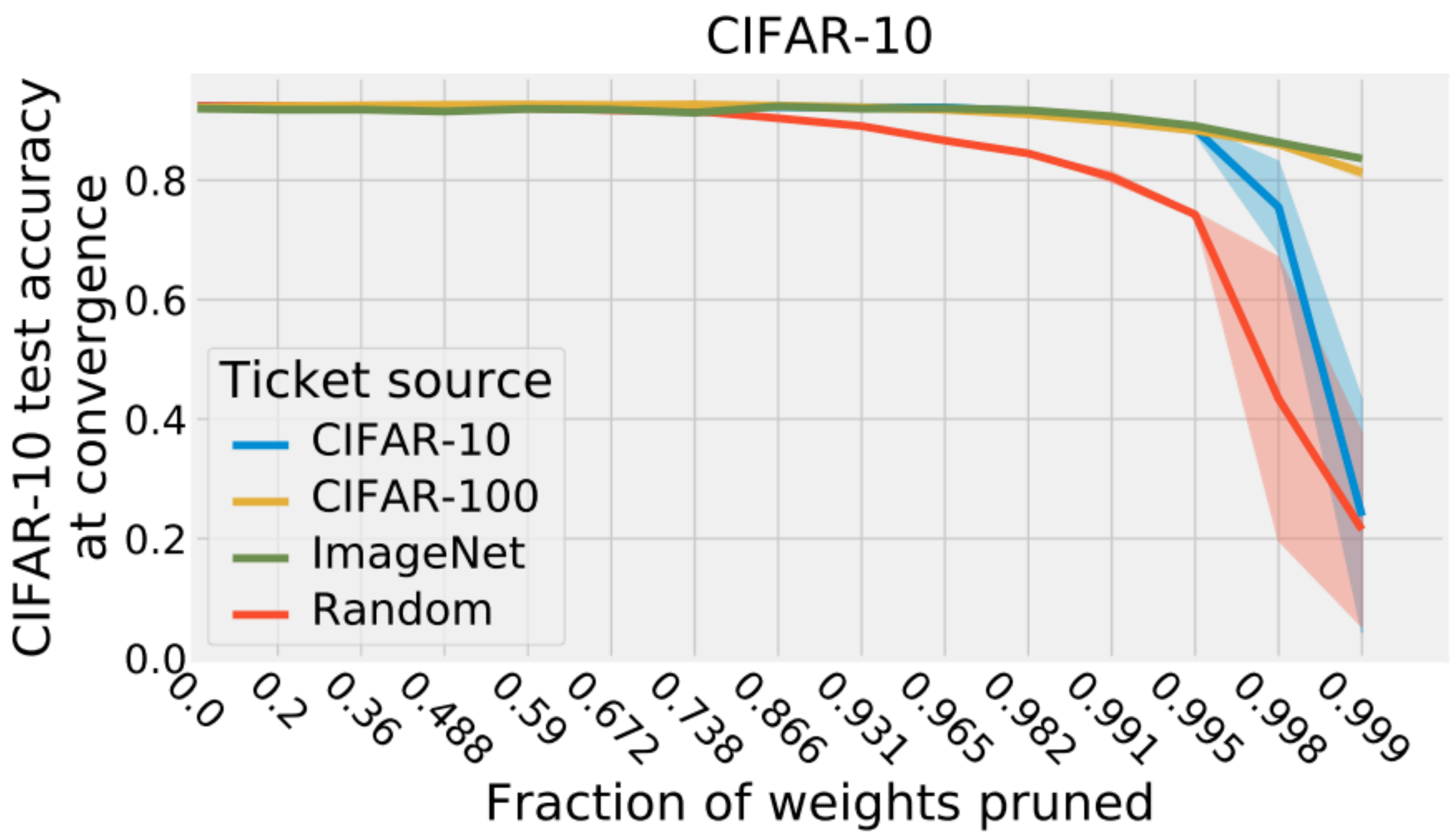
Yuandong Tian
Facebook AI Research
yuandong@fb.com

Abstract

Do lottery tickets found on a task transfer to another task?
How similar are they?

tl;dr *Winning ticket initializations contain generic inductive biases which generalize to related, but distinct datasets and across optimizers*

1. "within the natural images domain, winning ticket initializations generalized across a variety of datasets"
2. Complexity of dataset (number of examples, number of classes, ...) correlates positively with transferability
3. "winning ticket initializations generalize across optimizers"



Do lottery tickets found on a task transfer to another task?
How similar are they?

Bespoke vs. Prêt-à-Porter Lottery Tickets: Exploiting Mask Similarity for Trainable Sub-Network Finding

Michela Paganini
Facebook AI Research
michela@fb.com

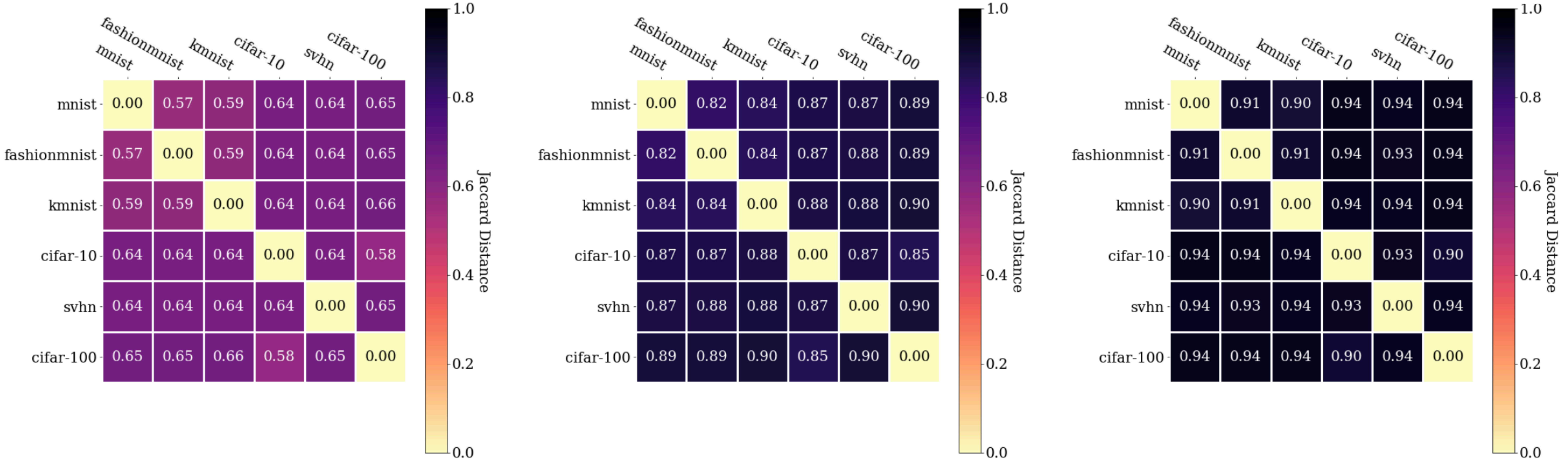
Jessica Zosa Forde
Brown University
Facebook AI Research
jessica_forde@brown.edu

Abstract

The observation of sparse trainable sub-networks within over-parametrized networks – also known as Lottery Tickets (LTs) – has prompted inquiries around their trainability, scaling, uniqueness, and generalization properties. Across 28 combinations of image classification tasks and architectures, we discover differences in the connectivity structure of LTs found through different iterative pruning techniques, thus disproving their uniqueness and connecting emergent mask structure to the choice of pruning. In addition, we propose a consensus-based method for generating refined lottery tickets. This lottery ticket denoising procedure, based on the principle that parameters that always go unpruned across different tasks more reliably identify important sub-networks, is capable of selecting a meaningful portion of the architecture in an embarrassingly parallel way, while quickly discarding extra parameters without the need for further pruning iterations. We successfully train these sub-networks to performance comparable to that of ordinary lottery tickets.

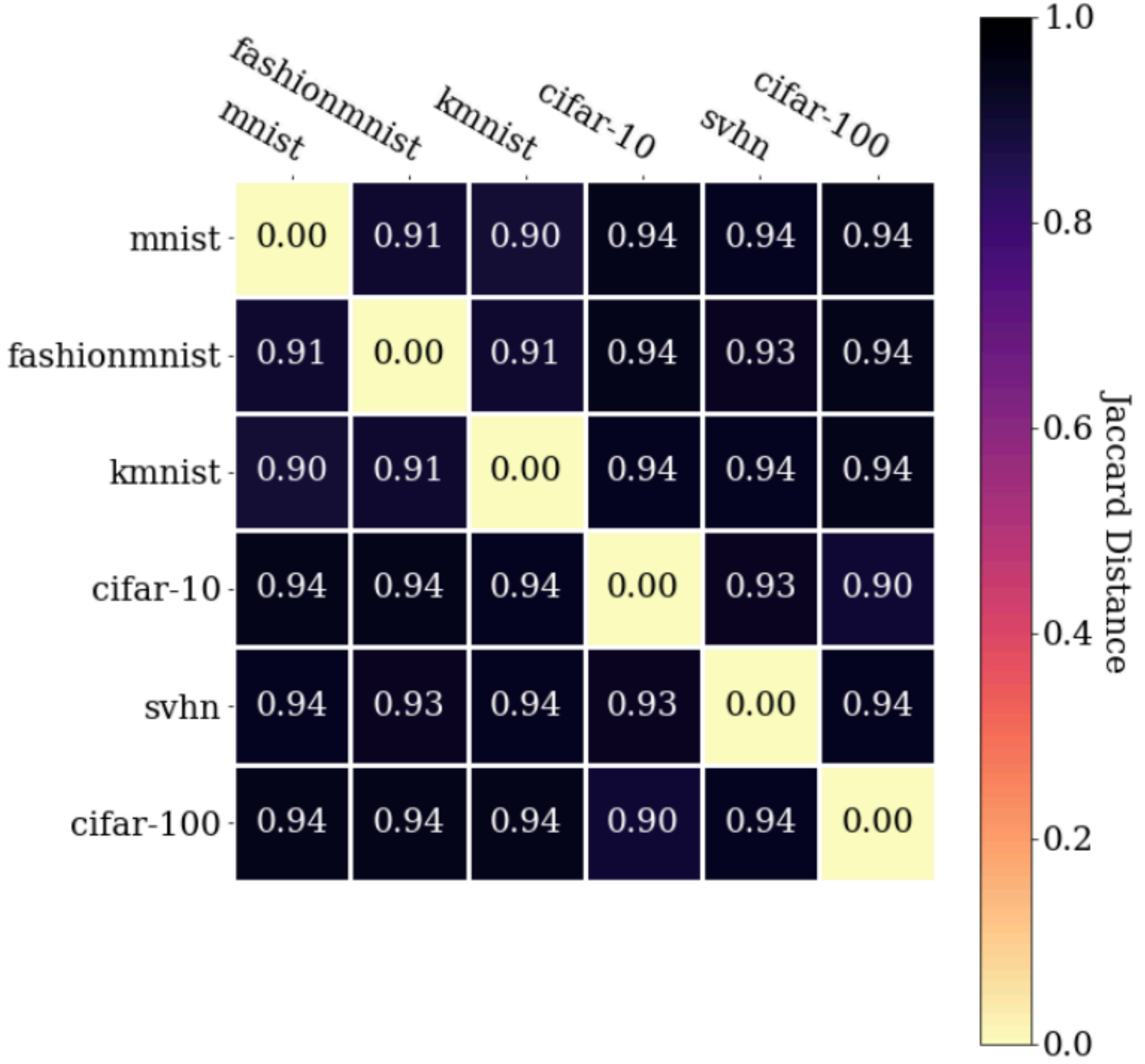
Do lottery tickets found on a task transfer to another task? How similar are they?

Not identical. Measure difference in terms of Jaccard distance: $d_J(\mathbb{M}_1, \mathbb{M}_2) = 1 - \frac{|\mathbb{M}_1 \cap \mathbb{M}_2|}{|\mathbb{M}_1 \cup \mathbb{M}_2|}$



(a) Pruning iteration 2 (b) Pruning iteration 8 (c) Pruning iteration 19

Figure 5: Pairwise total Jaccard distance between the bespoke masks obtained through global unstructured pruning on LeNet, over a set of tasks. As the networks grow progressively sparser, the distance between masks grows as their intersection shrinks.



(c) Pruning iteration 19

The intersection of lottery tickets is a lottery ticket

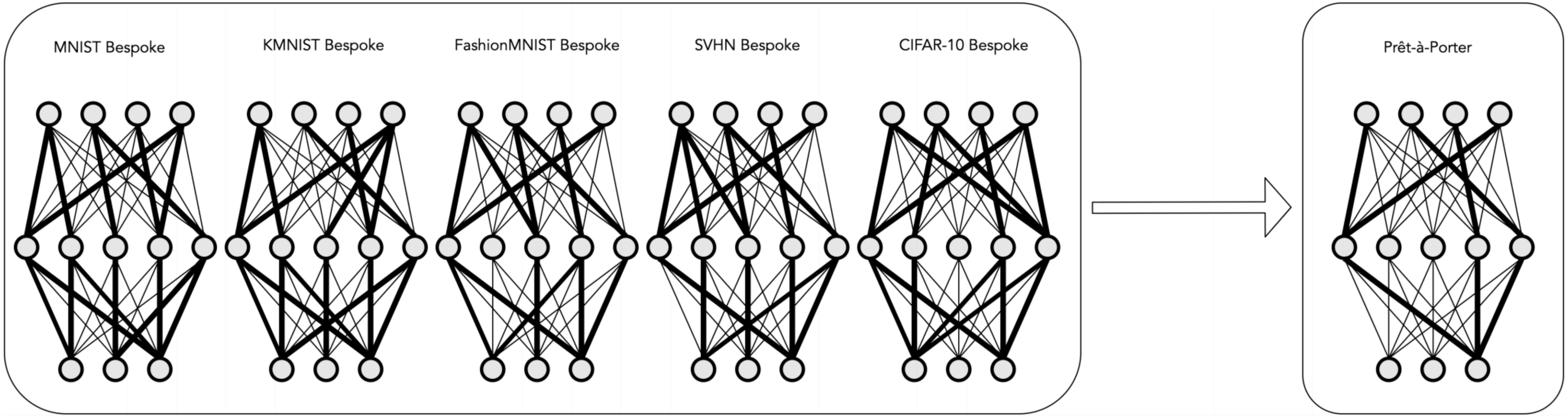


Figure 1: Consensus-based lottery ticket identification from bespoke lottery tickets sourced on different tasks. Thick lines identify unpruned connections; thin lines, pruned ones.

The intersection of lottery tickets is a lottery ticket

Algorithm 1: Prêt-à-porter (consensus) lottery ticket finding algorithm

Data: Set of datasets $\mathcal{S}$ on which to source ticket, a network with weights θ, pruning iterations T, epochs for iteration E.
Result: A consensus mask $\mathbb{M}$

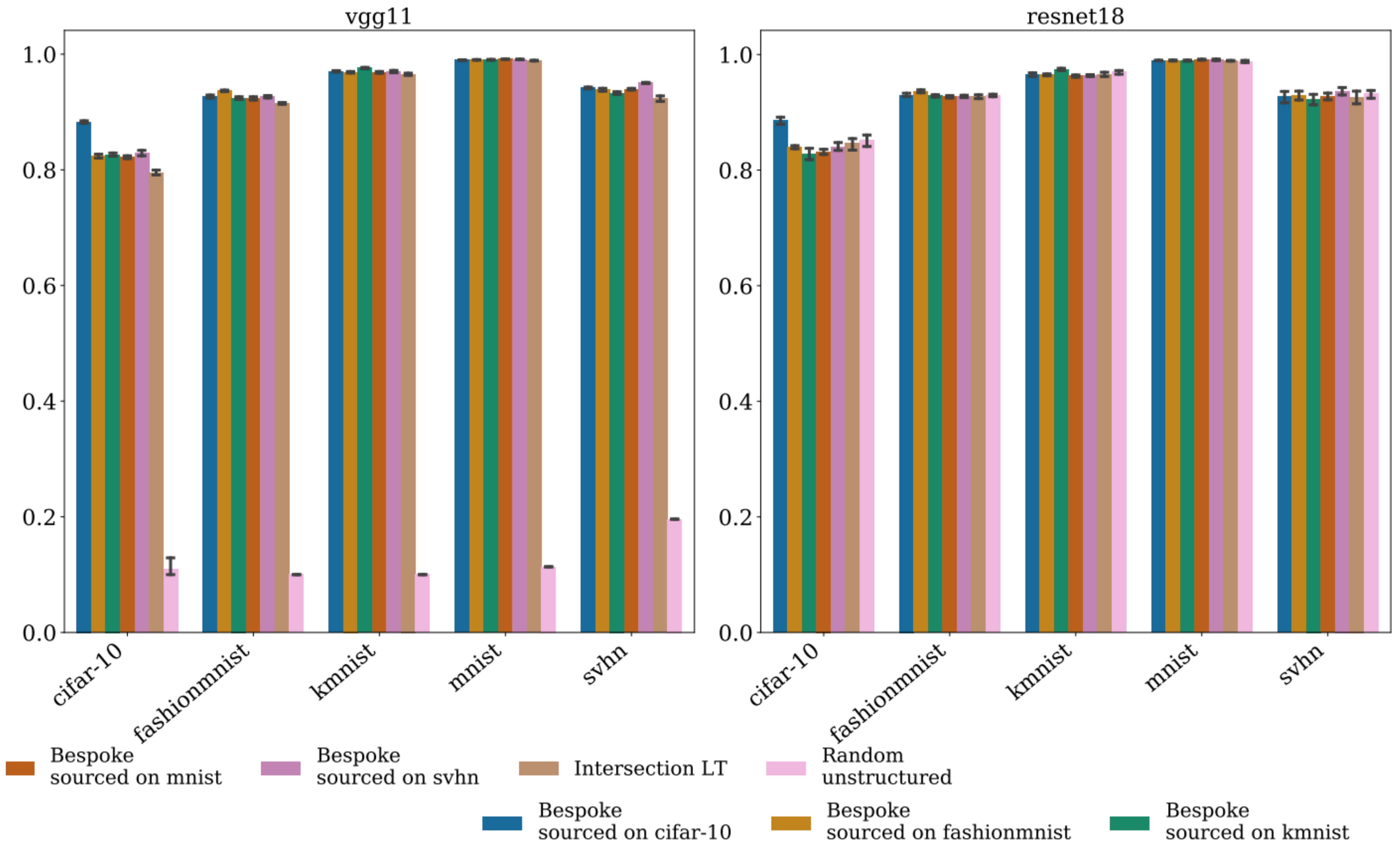
```
M = I_|θ| ;
for dataset S ∈ 𝒮 do
    θ_S = θ ;
    for t ← 1 to T do
        for e ← 1 to E do
            θ_trained = TrainEpoch(θ_S) ;
        end
        θ_pruned = Prune(θ_trained) ;
        θ_S = Reinit(θ_pruned) ;
    end
    M = M ∩ GetMask(θ_S) ;
end
return M ;
```

The intersection of lottery tickets is a lottery ticket

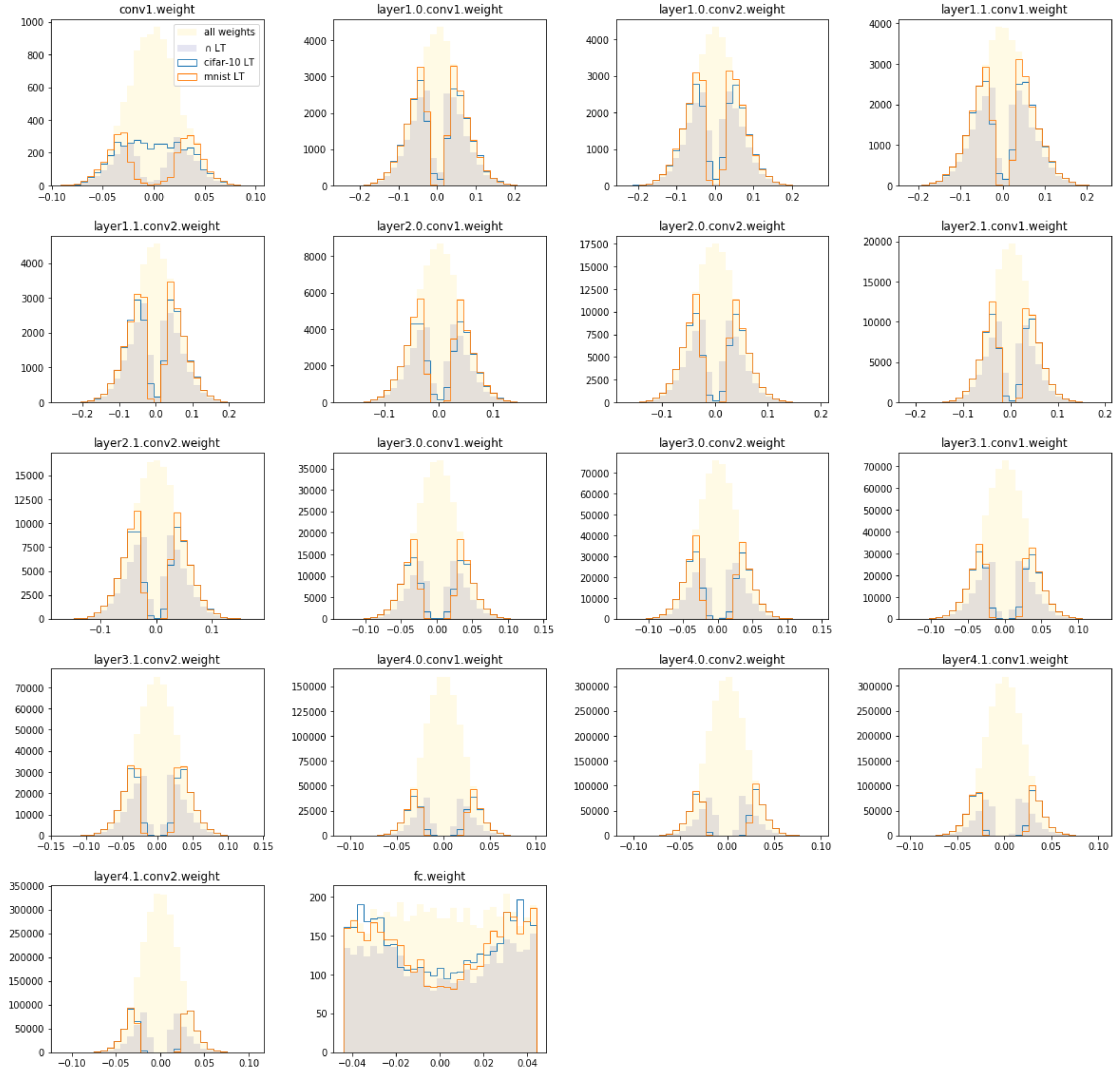
Algorithm 1: Prêt-à-porter (consensus) lottery ticket finding algorithm

Data: Set of datasets $\mathcal{S}$ on which to source ticket, a network with weights θ, pruning iterations T, epochs for iteration E.
Result: A consensus mask $\mathbb{M}$

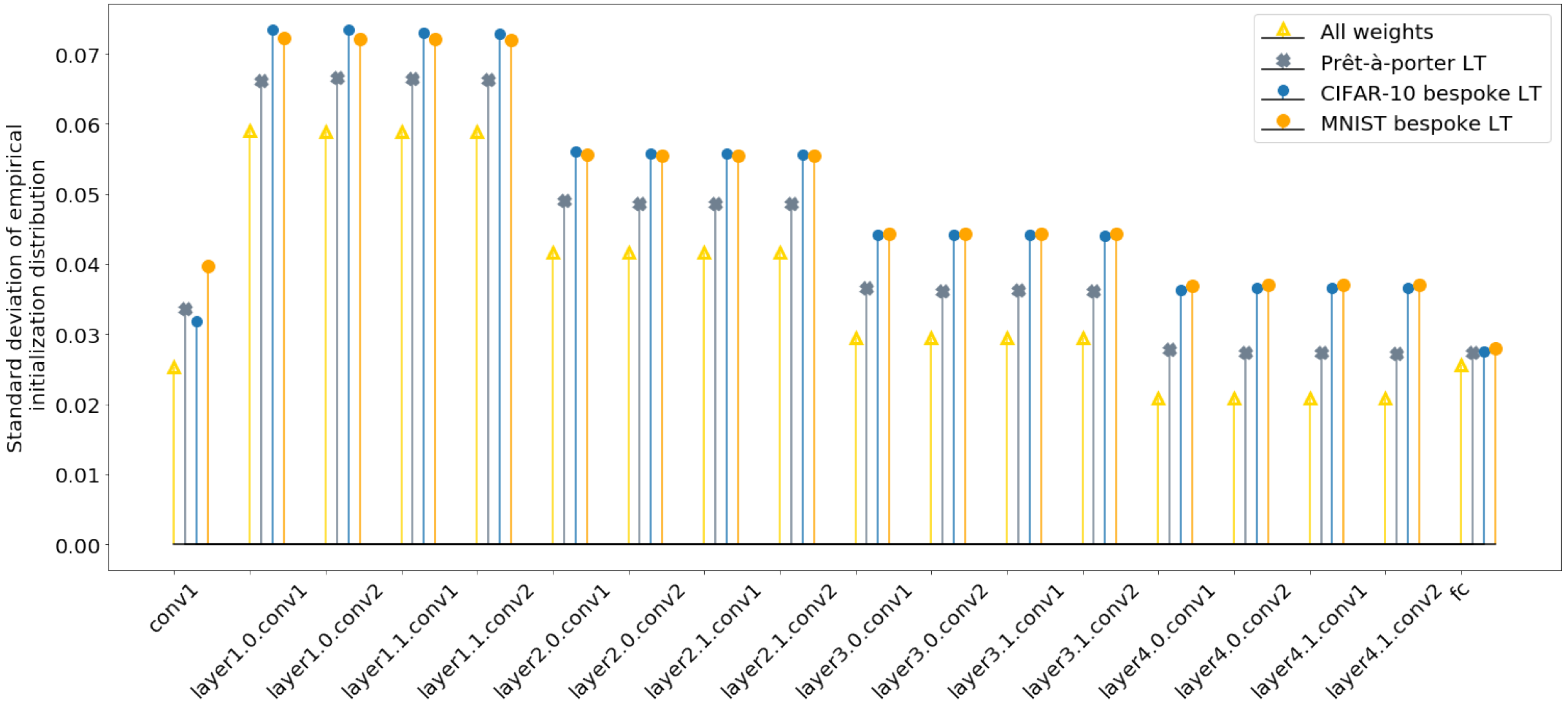
$\mathbb{M} = I_{|\theta|}$;
for *dataset* $S \in \mathcal{S}$ **do**
  $\theta_S = \theta$;
  for $t \leftarrow 1$ **to** T **do**
    for $e \leftarrow 1$ **to** E **do**
      $\theta_{\text{trained}} = \text{TrainEpoch}(\theta_S)$;
    end
    $\theta_{\text{pruned}} = \text{Prune}(\theta_{\text{trained}})$;
    $\theta_S = \text{Reinit}(\theta_{\text{pruned}})$;
  end
  $\mathbb{M} = \mathbb{M} \cap \text{GetMask}(\theta_S)$;
end
return $\mathbb{M}$;



Intersection lottery ticket at initialization



Intersection (or "pret-à-porter") lottery tickets contain weights that are not as extremely far from 0 at initialization as traditional (or "bespoke") lottery tickets

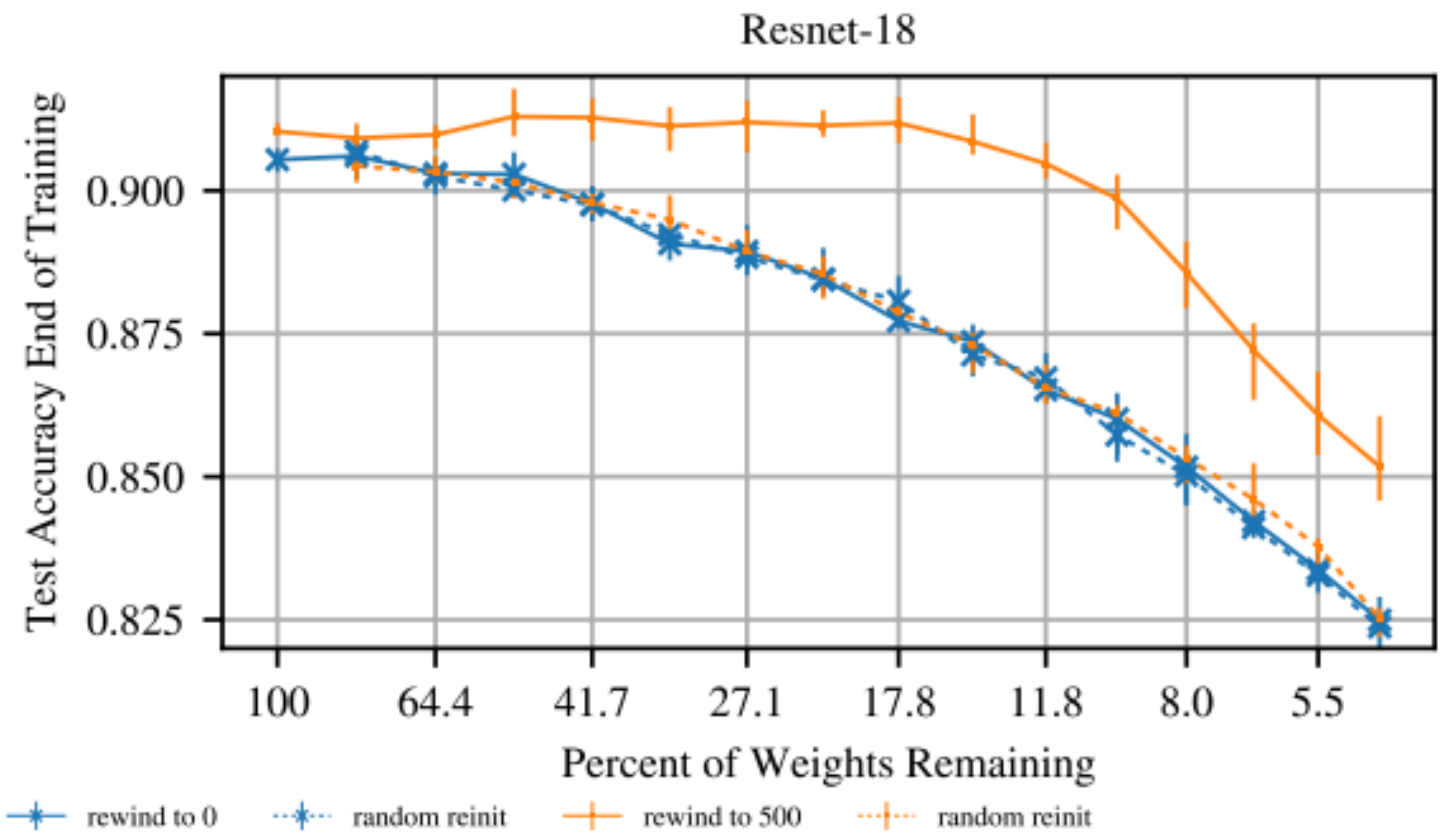


Early phases of training are messy but crucial

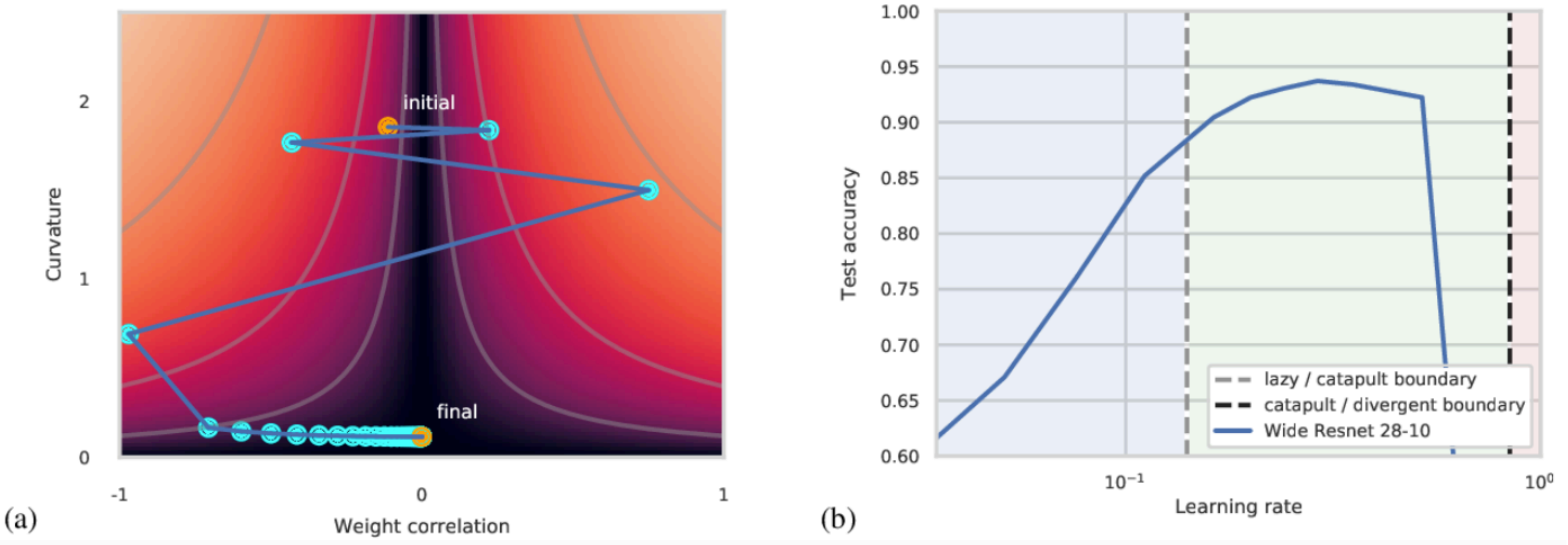
Papers from Frankle et al.

Relevant concepts:

- Learning rate warm up
- Late resetting
- Phased of neural network training
- Mode connectivity of SGD solutions



New insight into initial learning rate choice



05 What's next?

The future of AI

efficient

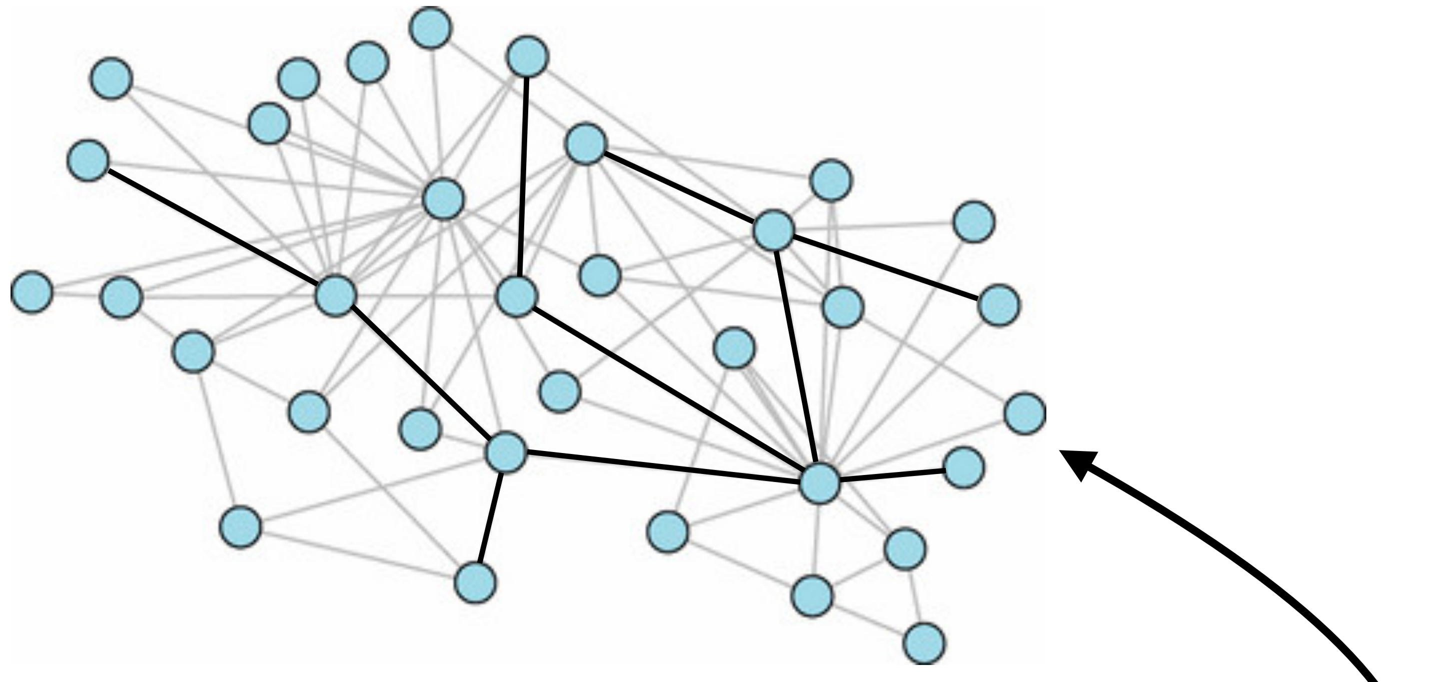
responsible

ubiquitous

private

Future directions

- Pruning @ init (SNIP, GRASP, SynFlow, and work in progress)
- Interpretability (circuits, Captum, ...)
- Connections with theory (over-parametrization, adaptive lr, hessian spectrum, ...)
- Connections with other empirical work (intrinsic dimensions, random projections, ...)
- Delivering on shared, widespread computational benefits of sparsity coupled with accessible hardware solutions
- Responsible AI, sustainability, auditing engineering decisions
- Applications to science, AR/VR, privacy, and more!



Thanks!

Questions? Contact me: mickypaganini@berkeley.edu

WonderMicky

Backup

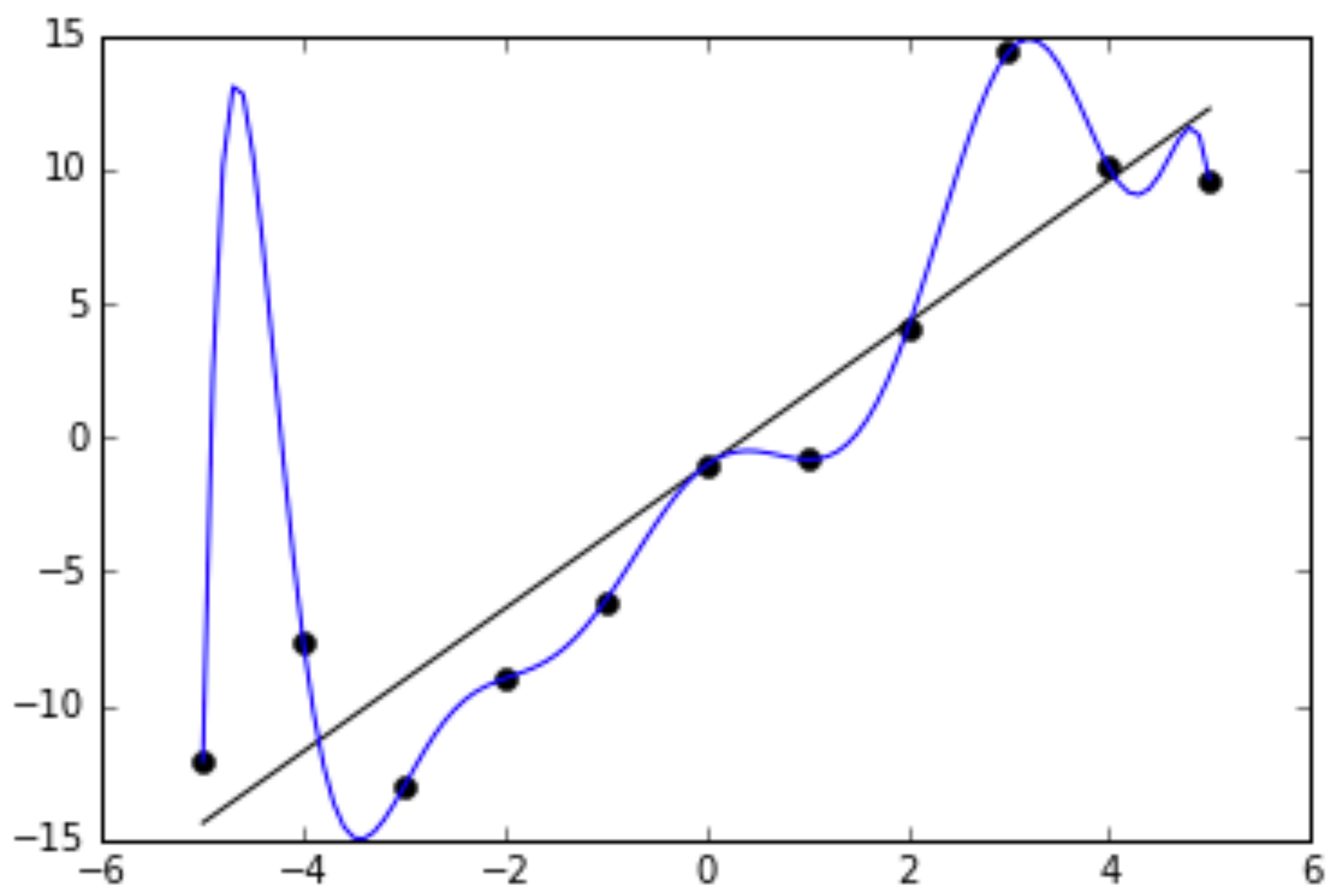
PruningContainer

Network capacity and over-parametrization

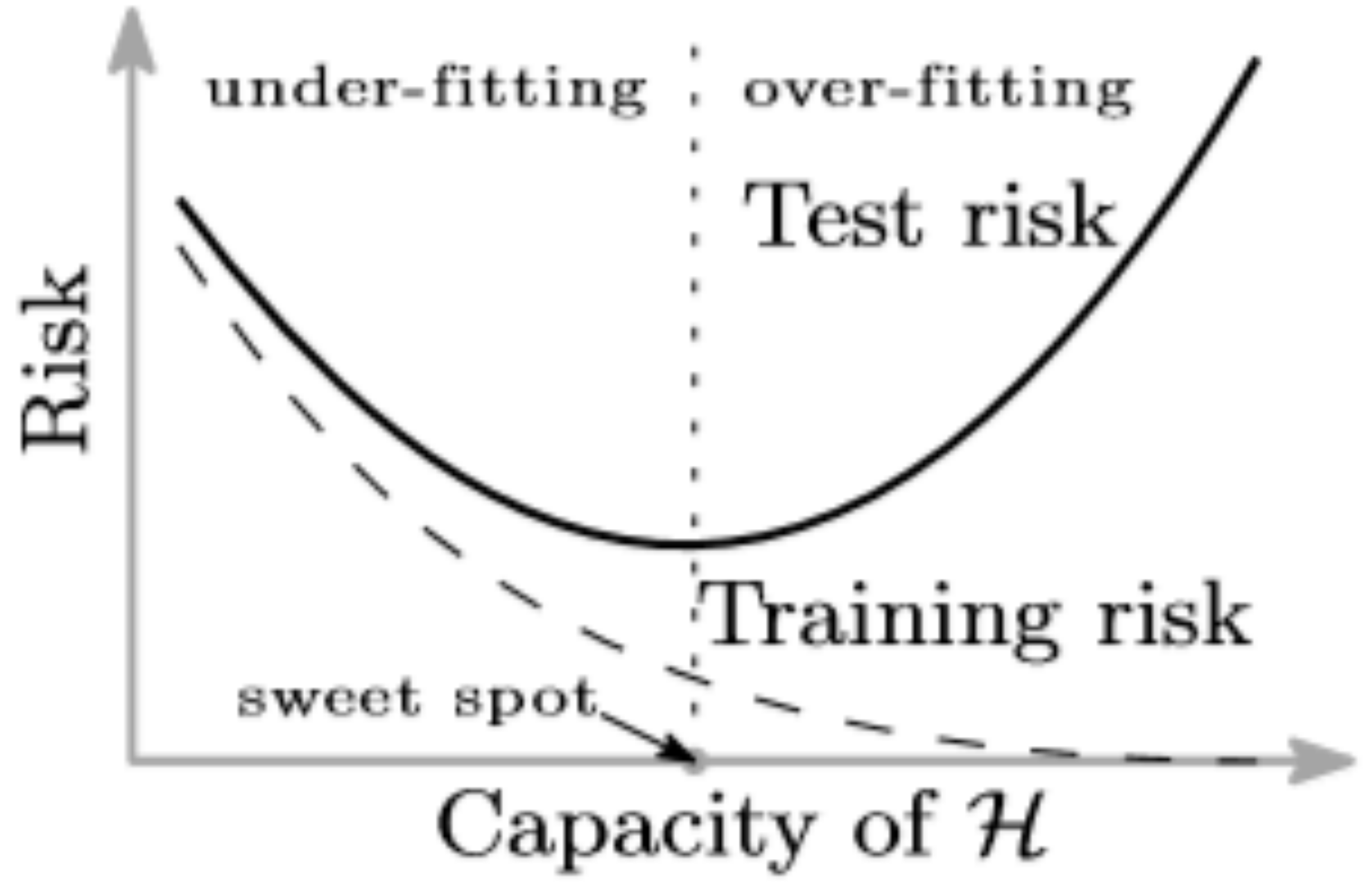
test error = train error + generalization error

can be reduced to 0 if no degeneracy

will it always grow as network size increases?



Wikipedia "Overfitting"



Belkin et al., 2018

"risk" = expected loss

not the full story...

Network capacity and over-parametrization

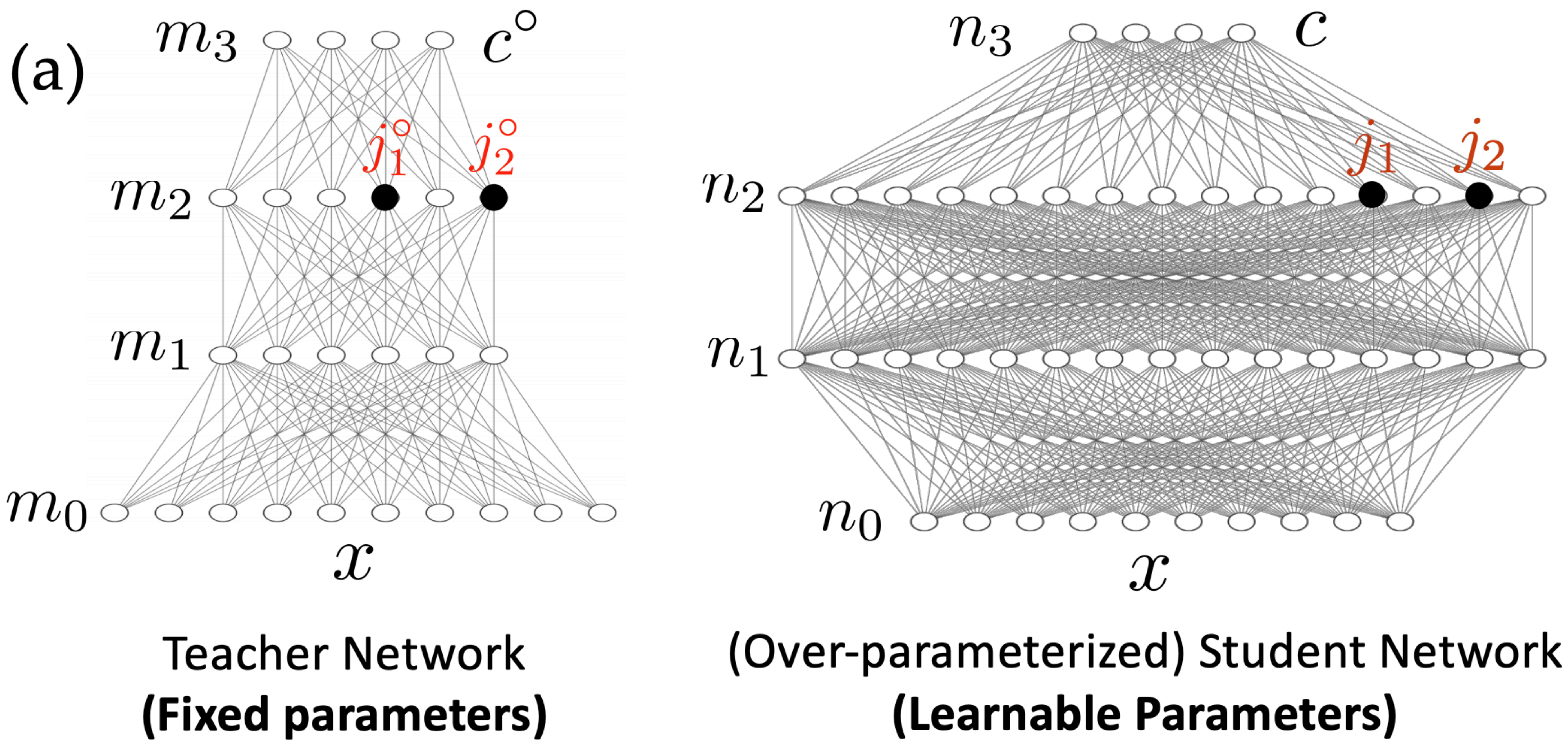
Optimizers like SGD converge to:
- global min, if the function is convex
- stationary point, if the function is non-convex non-smooth
- good solutions, in practice

Advantages of over-parametrization:
- Many high quality local minima
- Loss landscape can be designed (via {loss, architecture, regularization} pick) so that the optimizer finds good solutions
- Global minimizer close to init ↔ weights have to move less ↔ easier optimization
- Ability to learn information that generalizes

Implicit regularization helps with generalization

Teacher-student framework to understand deep ReLU nets

Nodes in over-parametrized student networks compete to explain teacher's nodes

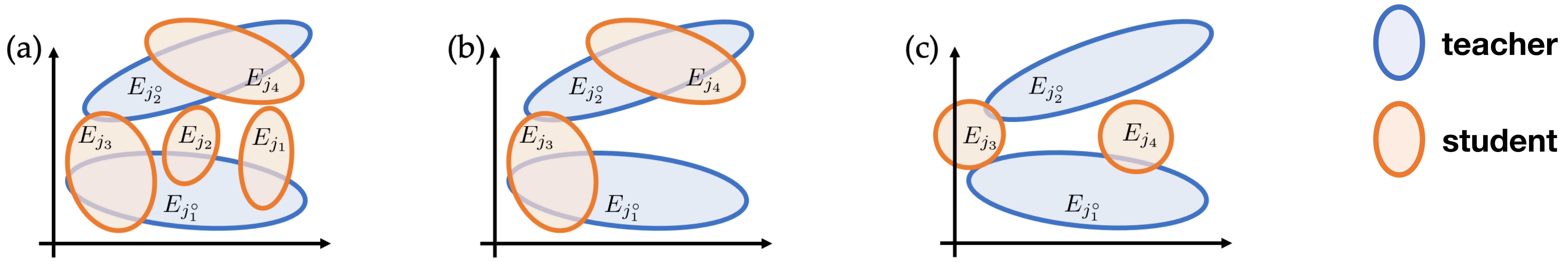


$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{2}\mathbb{E}_x\left[\|f_c(x) - f_{c^\circ}(x)\|^2\right]$$

Intuition: "why does over-parametrization matter and why is weight magnitude a good proxy for importance?"

Diagrams show activation regions of different nodes *j*, *i.e.* regions of input space for which the node is firing:

$$E_j \equiv \{x : f_j(x) > 0\}$$



Before pruning **Lottery ticket-style rewinding** **Random reinitialization**

Found that:

1. over-parametrization = more student nodes = higher probability of alignment
2. higher magnitude teacher weights attract more student weights and are aligned with faster
3. alignment between student and teacher happens in early layers first
4. student nodes that don't converge to teacher converge to 0

See full paper for formal derivation of recursive layer-by-layer optimization via top-down modulation

Linear Mode Connectivity and the Lottery Ticket Hypothesis, Frankle et al. [arXiv:1912.05671](https://arxiv.org/abs/1912.05671)

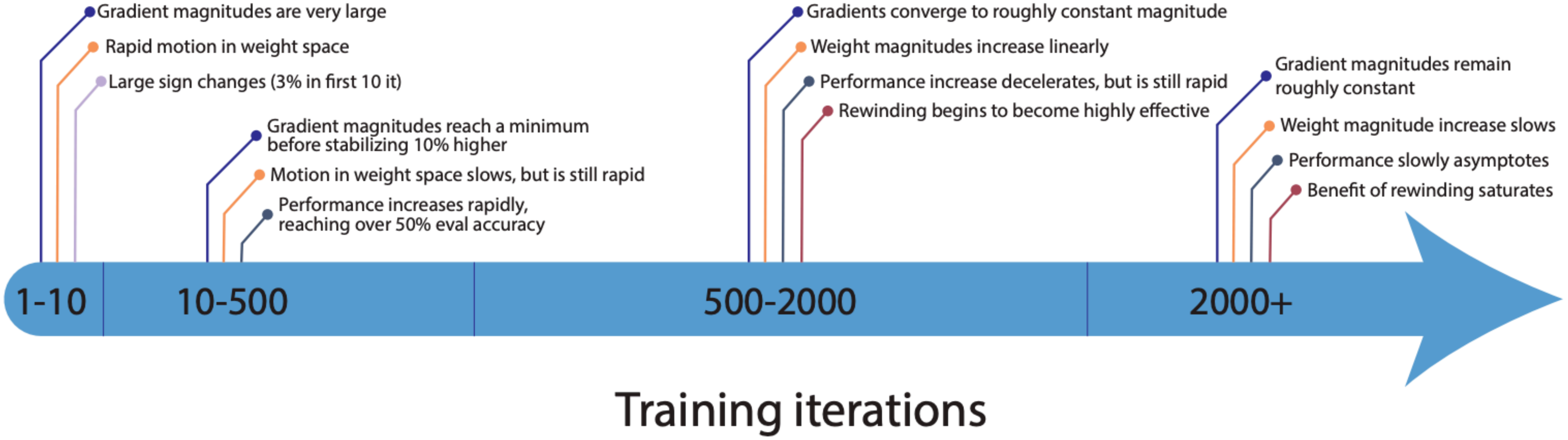
Early phases of training are messy but crucial

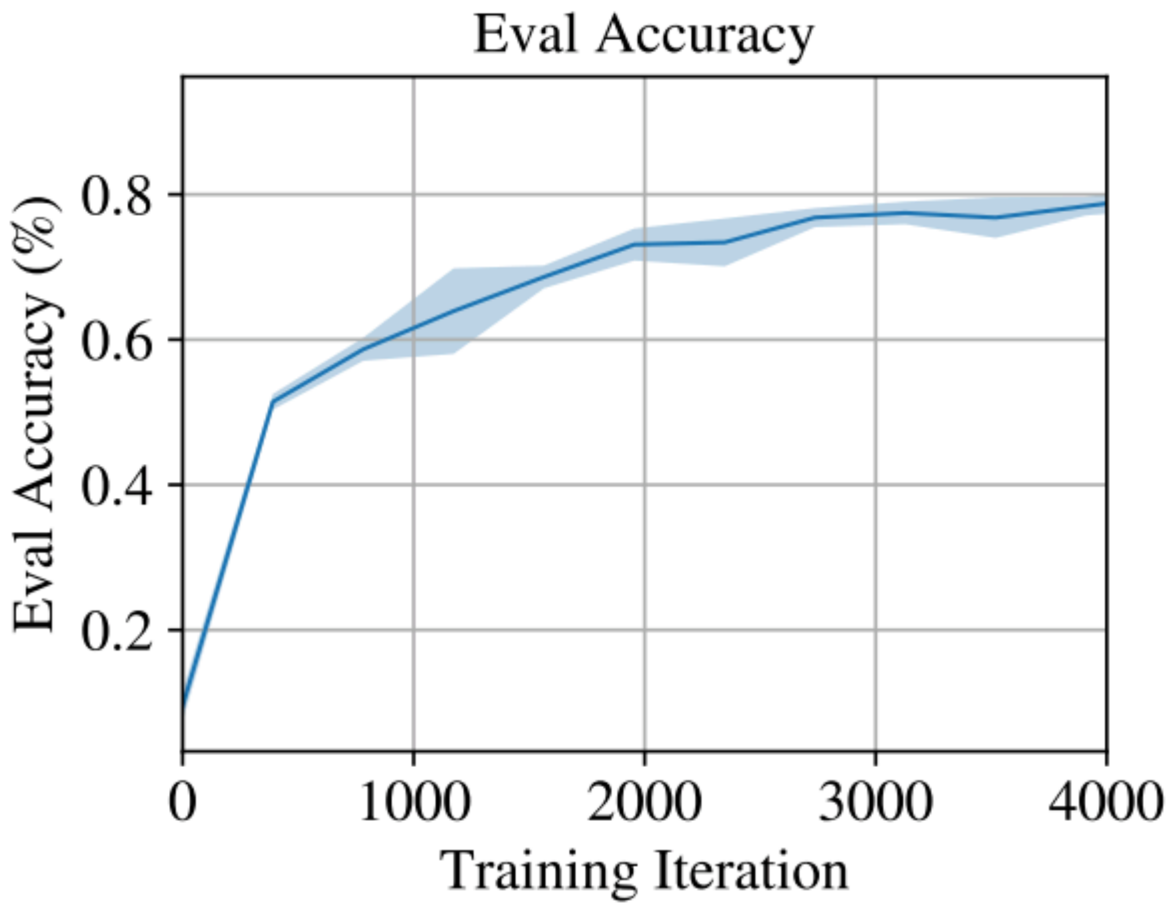
Frankle et al., 2020

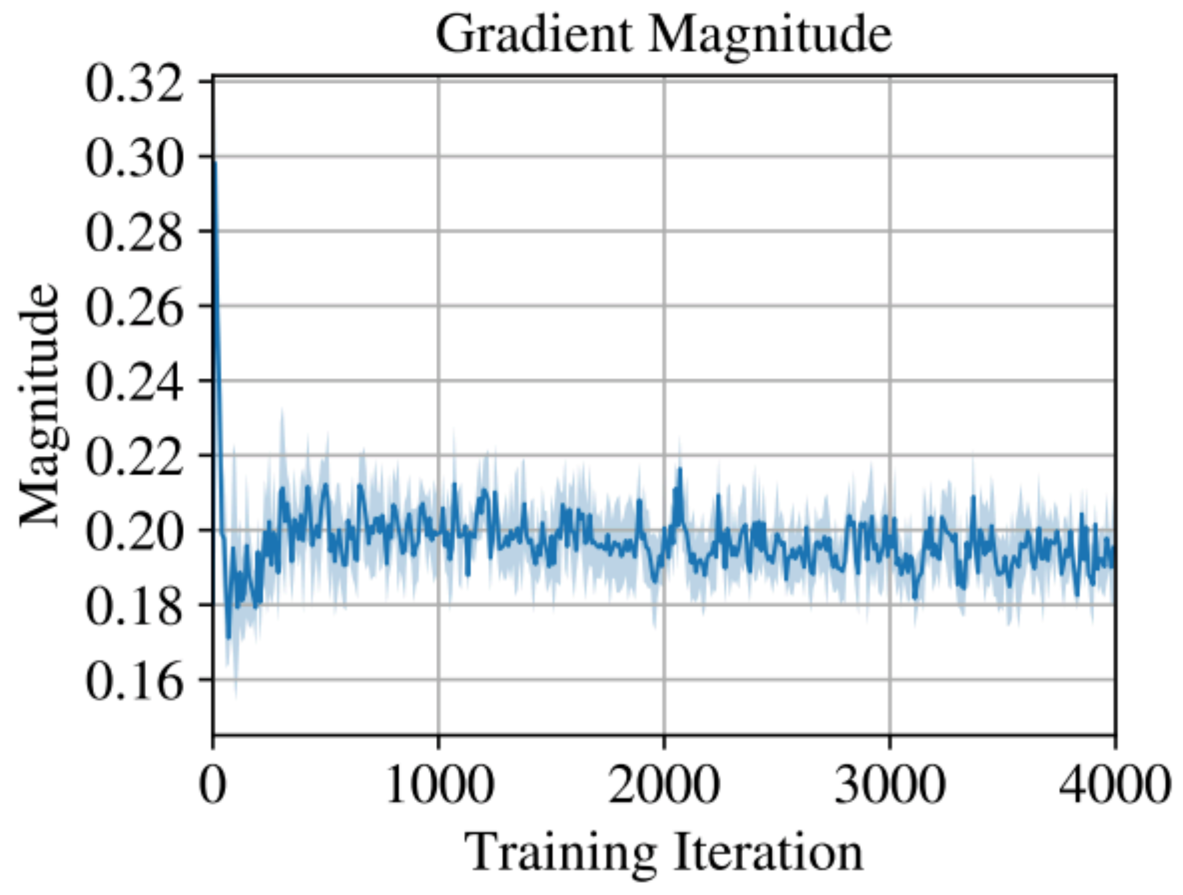
Warmup, late resetting, iterative magnitude-based pruning, and more can be justified in terms of linear mode connectivity between different SGD solutions obtained with different seeds

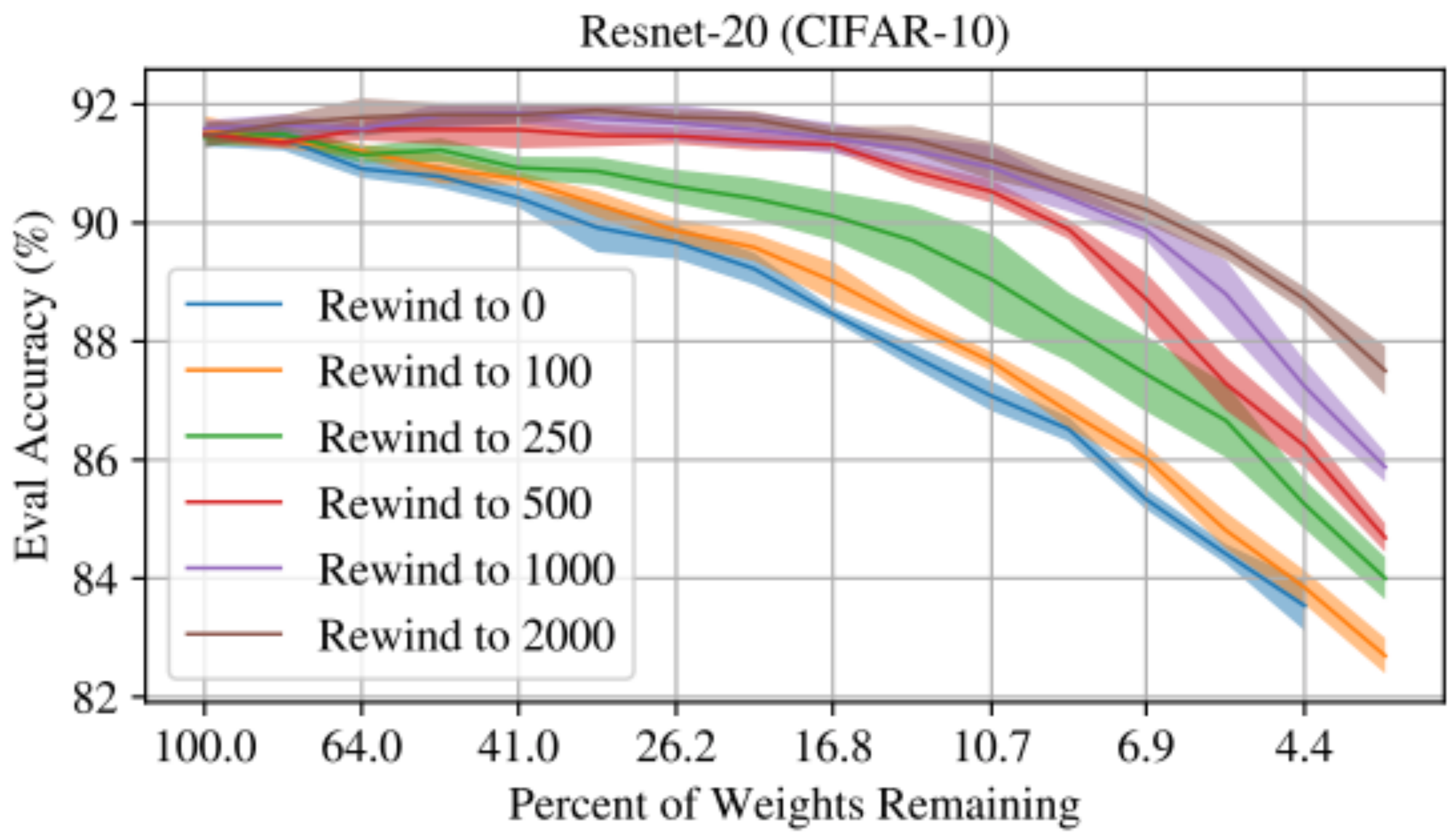
The Early Phase of Neural Network Training, Frankle et al. arXiv:2002.10365

Early phases of training are messy but crucial



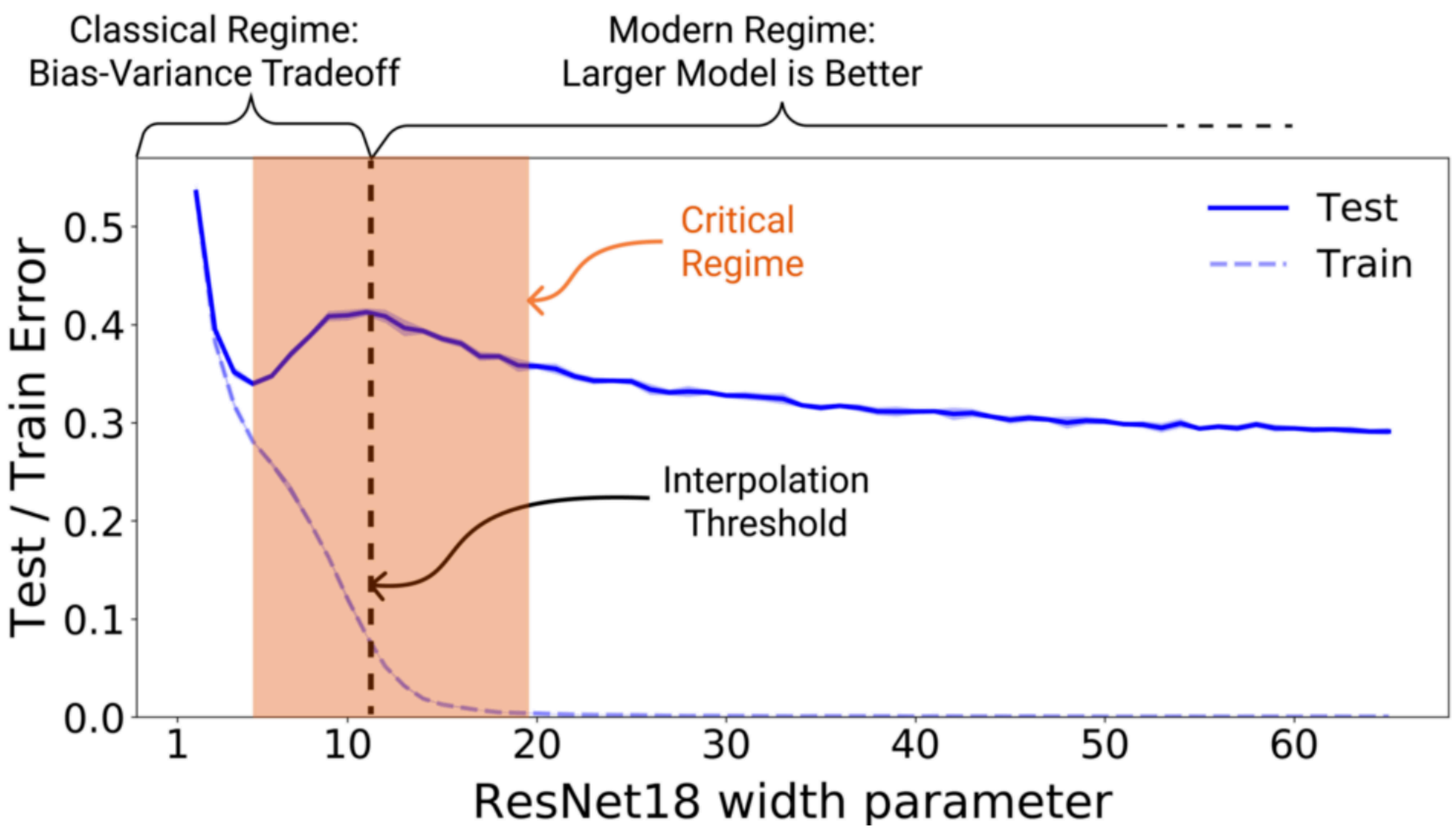






Network capacity and over-parametrization

What's the right picture?



Nakkiran et al., 2019

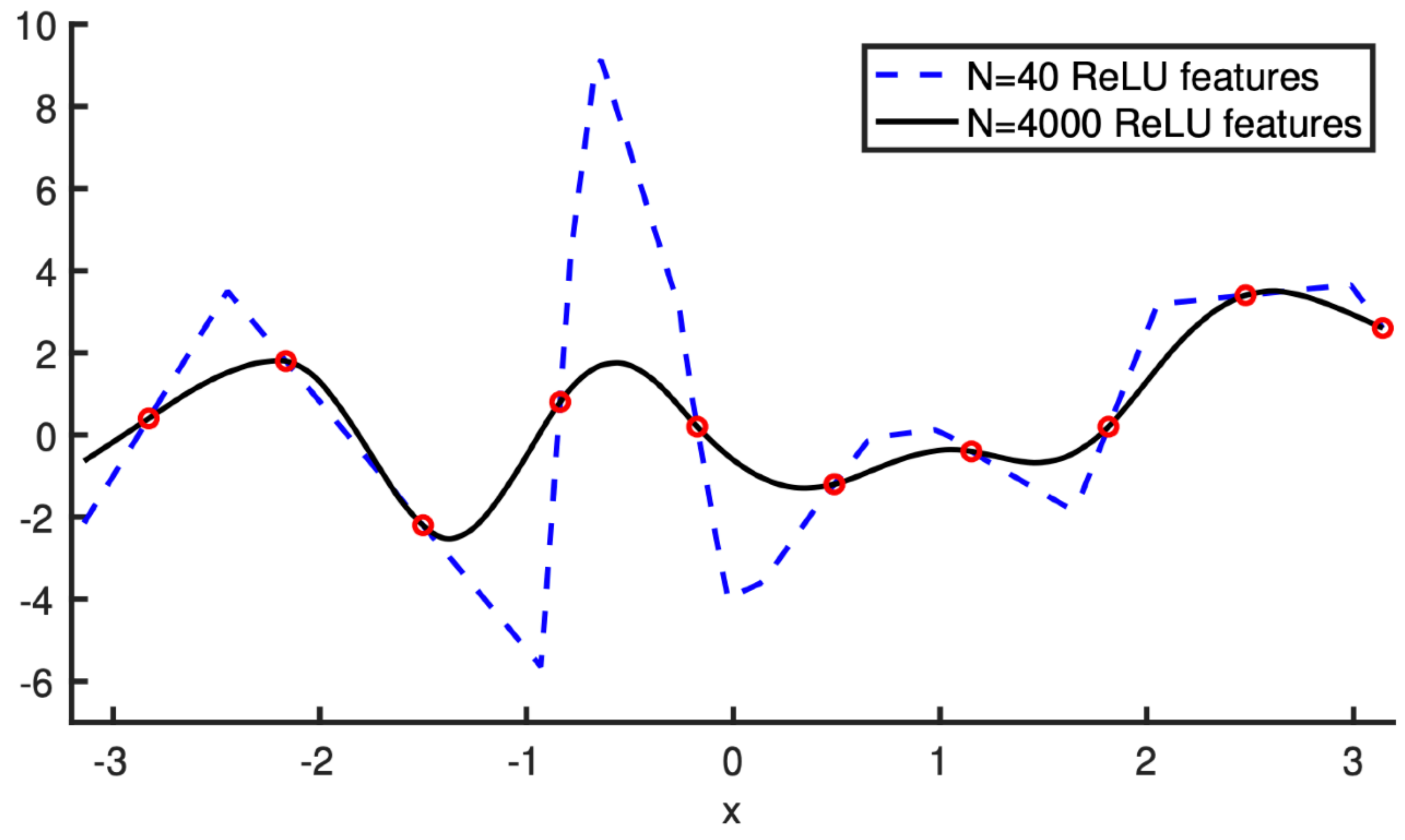
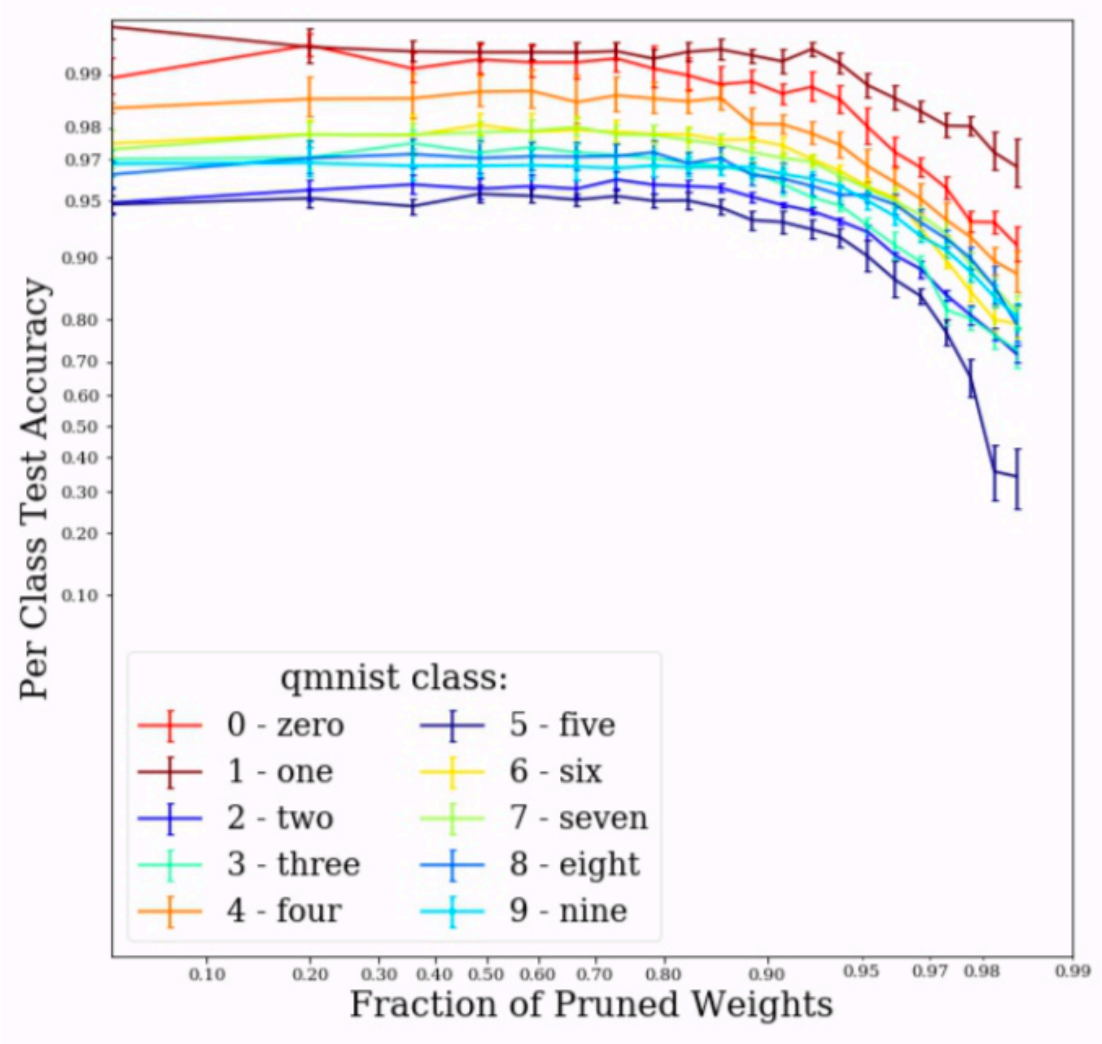


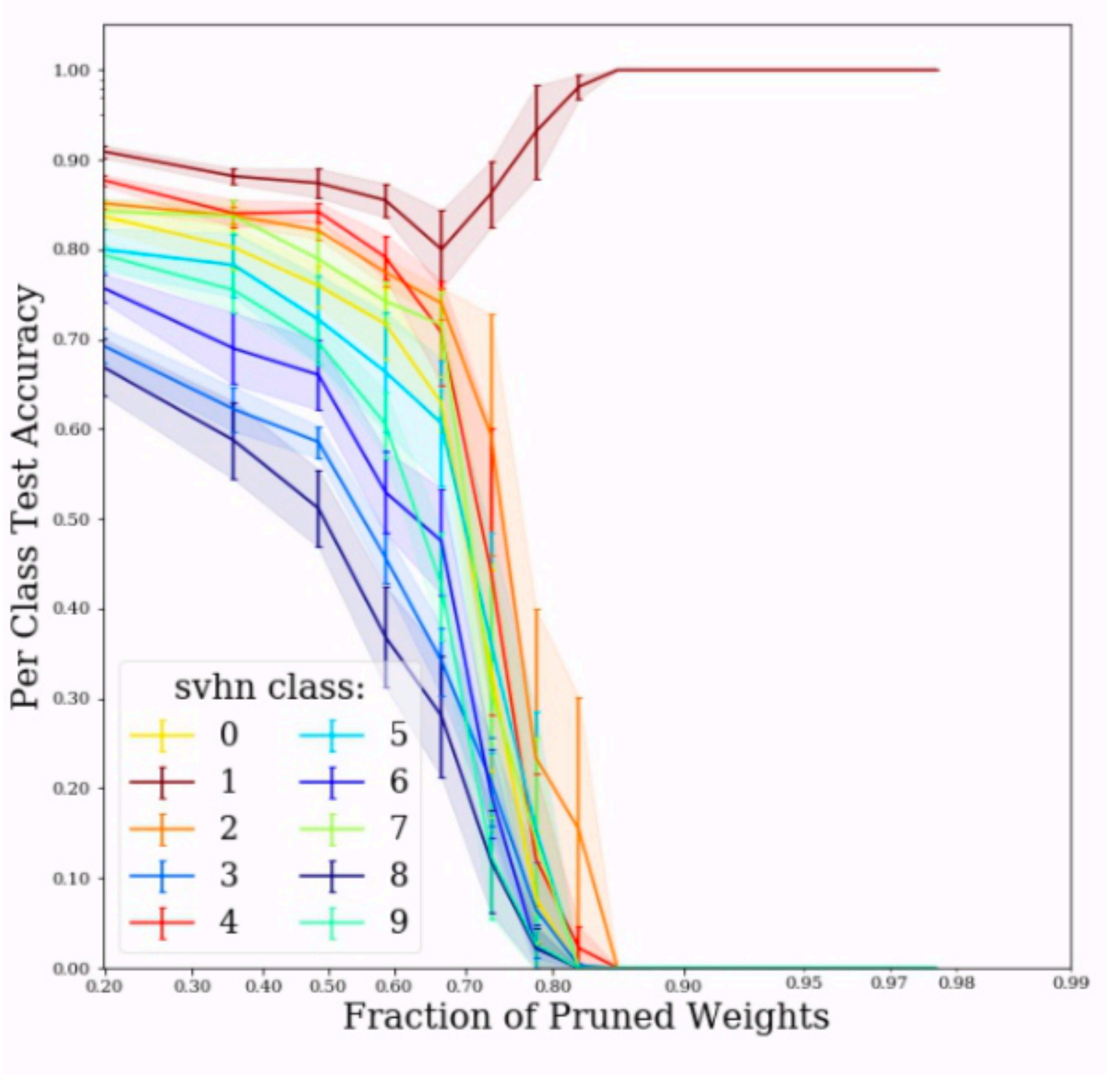
Figure 3: Plot of two univariate functions fitted to 10 data points using Random ReLU features $\phi(x;(v_1,v_2)) := \max(v_1 x + v_2, 0)$. The data points are shown in red circles. The fitted function with $N = 40$ Random ReLU features is the blue dashed line; the coefficient vector's norm (scaled by $\sqrt{N}$) is ≈ 695. The fitted function with $N = 4000$ Random ReLU features is the black solid line; the coefficient vector's norm is ≈ 159.

Belkin et al., 2018

What really happens to a network when we prune it?
How do even tiny performance losses affect individual examples?

Not all classes are identically affected by pruning





Not all groups and individuals are identically affected by pruning