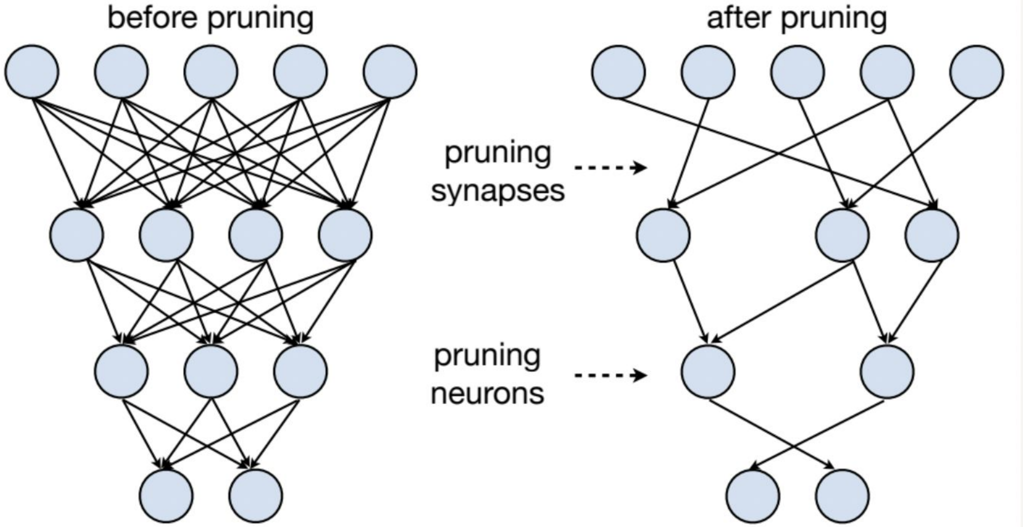
Pruning
Lottery Ticket Hypothesis

Sergey Pankevich, 2023

Pruning

The process of making neural network smaller by removing synapses and neurons



Obvious Purposes

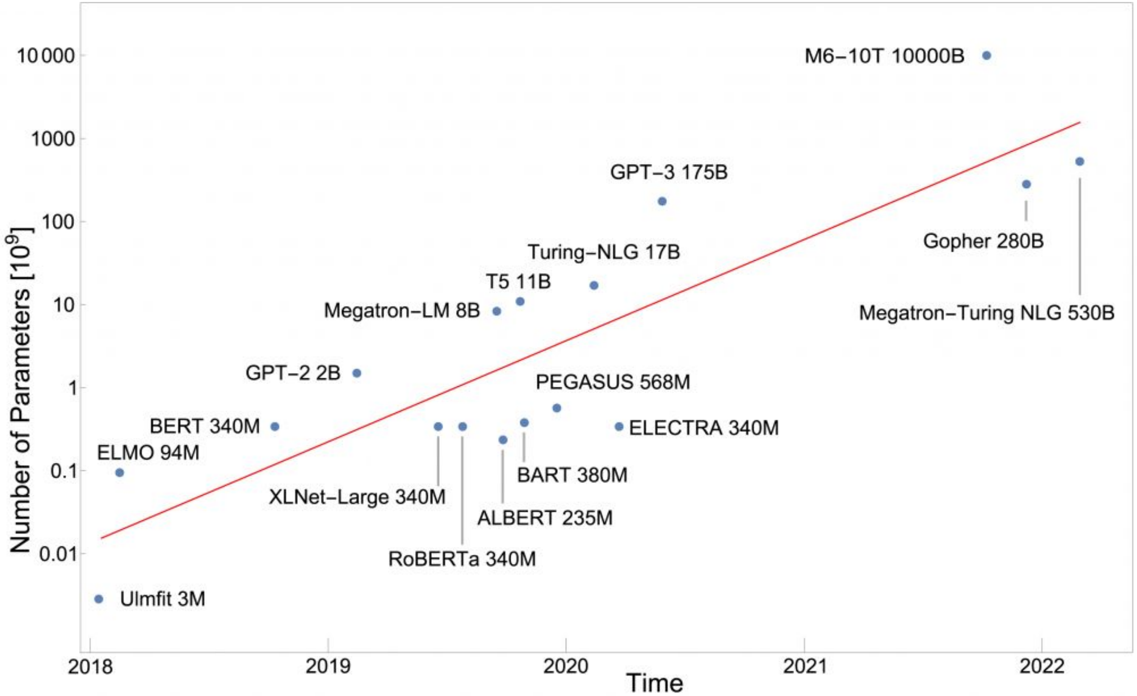
Decrease Size

Decrease FLOPs & Increase speed

Spend Less Energy

Size

Modern models are huge. Inference of large models is expensive and slow.



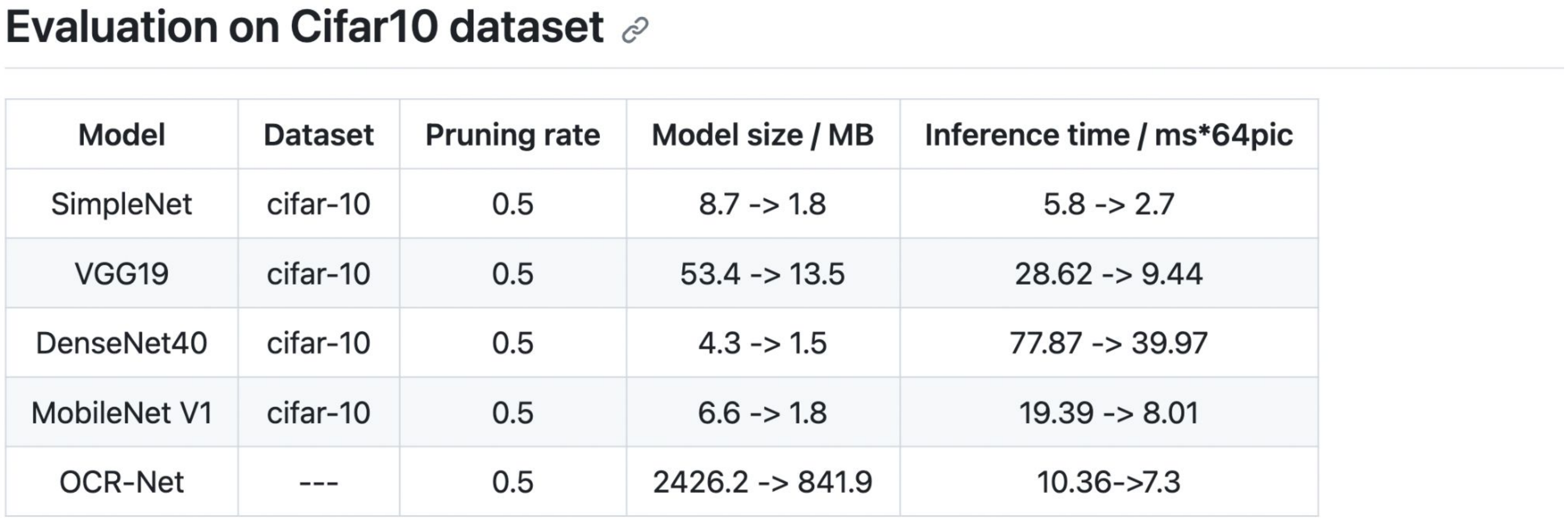
Size

Neural networks are widely used on mobile devices. For example, they help to sharpen blurry images or enhance the quality of low-light (night) photos.

However, size is a significant limitation in this case due to the small memory capacity of mobile devices.

Speed and FLOPs

Sparse models work sufficiently faster



Evaluation on Cifar10 dataset

Model	Dataset	Pruning rate	Model size / MB	Inference time / ms*64pic
SimpleNet	cifar-10	0.5	8.7 -> 1.8	5.8 -> 2.7
VGG19	cifar-10	0.5	53.4 -> 13.5	28.62 -> 9.44
DenseNet40	cifar-10	0.5	4.3 -> 1.5	77.87 -> 39.97
MobileNet V1	cifar-10	0.5	6.6 -> 1.8	19.39 -> 8.01
OCR-Net	---	0.5	2426.2 -> 841.9	10.36->7.3

Energy

Apart from doing many FLOPs large models are worse for cache

We should often go to RAM to load new data

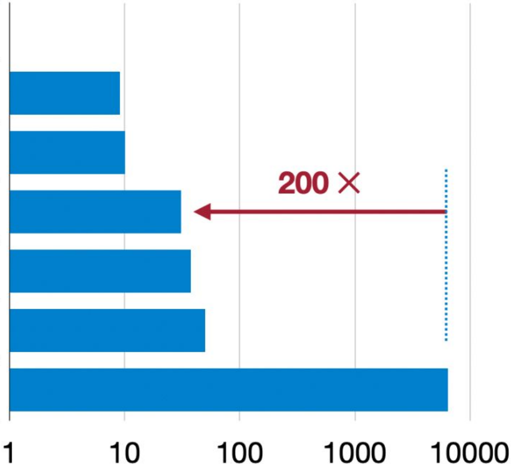
Energy

Data Movement → More Memory Reference → More Energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V

Relative Energy Cost



1 **= 200 ×+**

This image is in the public domain

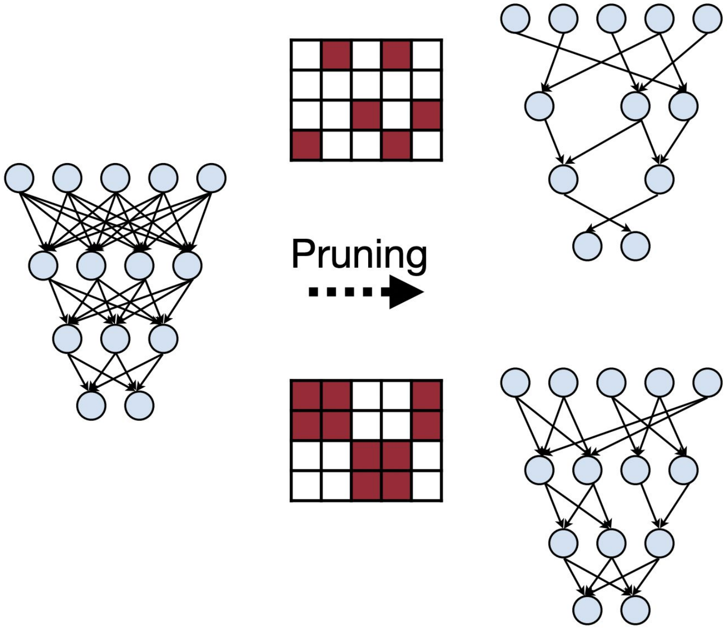
What about quality?

It is not bad.

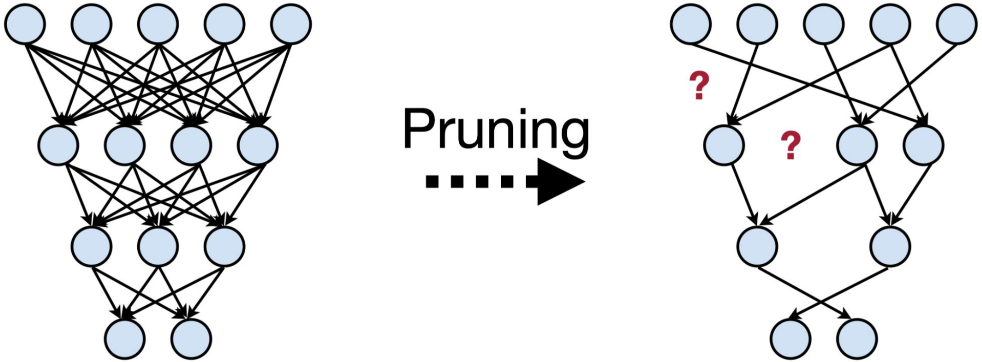
Pruning is not necessarily bad for accuracy. In fact, it can be beneficial as it reduces the risk of overfitting in less parameterized networks. By employing smart pruning techniques, it is possible to decrease the size of a network by 80-99% with minimal loss in performance.

Pruning pattern

We can prune whole neurons or single weights or some specific structures.

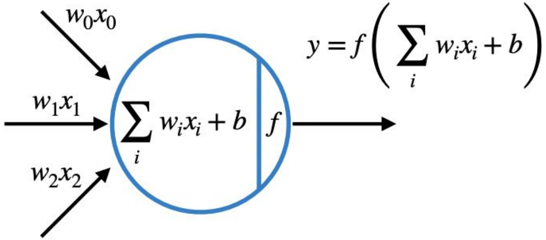


Pruning criterion



which synapses?
which neurons?

Easy Case



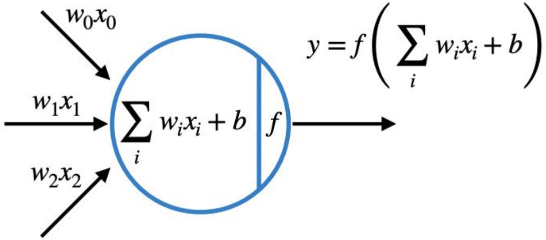
Example

$f(\cdot) = \text{ReLU}(\cdot), \ \ W = [10, -8, 0.1]$

➡ $y = \text{ReLU}(10x_0 - 8x_1 + 0.1x_2)$

- If one weight will be removed, which one?

Easy Case



Example

$$f(\cdot) = \text{ReLU}(\cdot),\ \ W = [10, -8, 0.1]$$

➡ $y = \text{ReLU}(10x_0 - 8x_1 + 0.1x_2)$

- If one weight will be removed, which one?

Obviously W_2

Magnitude-based pruning

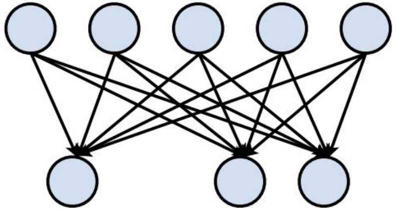
Ideally, we aim to prune the least important parameters in a network.

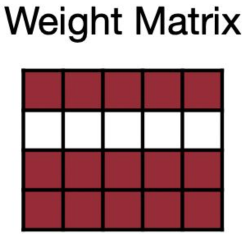
The magnitude heuristic states that parameters with larger magnitudes are considered to be more important.

Consider a matrix representing a fully connected linear layer.

Single cell corresponds to connection between two neurons.

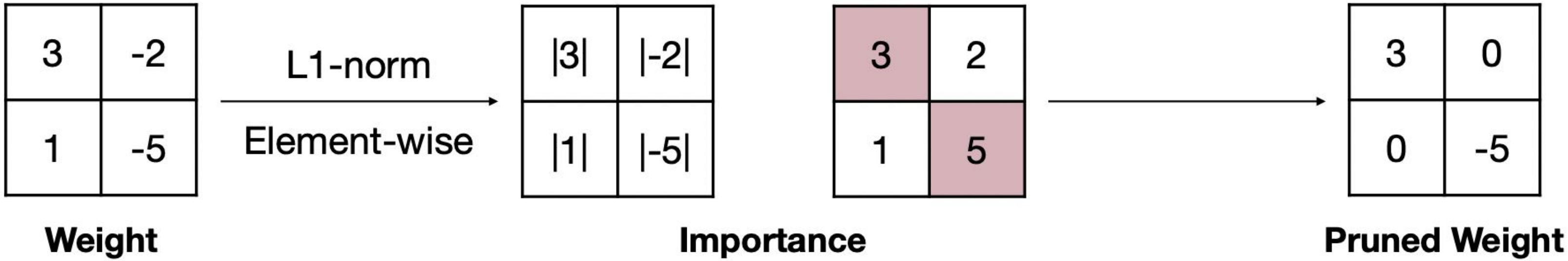
Single row corresponds to a neuron.





We can eliminate several synapses with smallest absolute value on each layer.

- **Example**



3	-2
1	-5

Weight

L1-norm

Element-wise

|3|	|-2|
|1|	|-5|

3	2
1	5

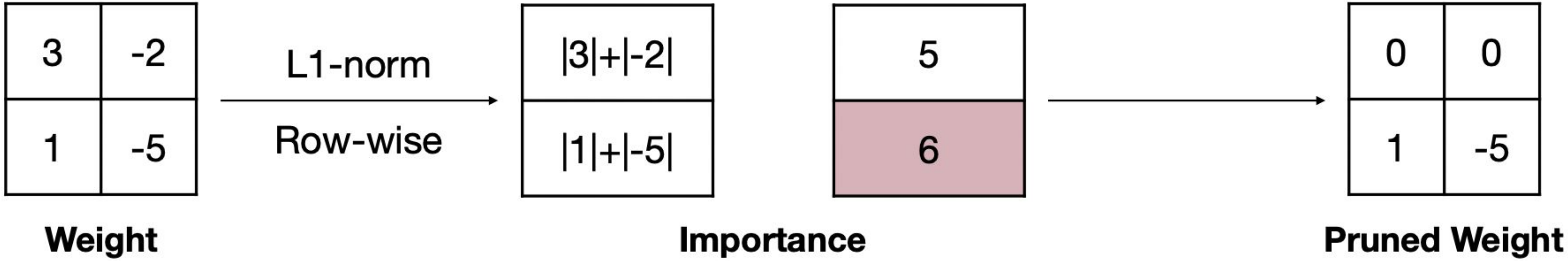
Importance

3	0
0	-5

Pruned Weight

We can eliminate neurons with smallest L1 / L2 norm of corresponding rows.

- **Example**



3	-2
1	-5

Weight

L1-norm

Row-wise

|3|+|-2|
|1|+|-5|

5
6

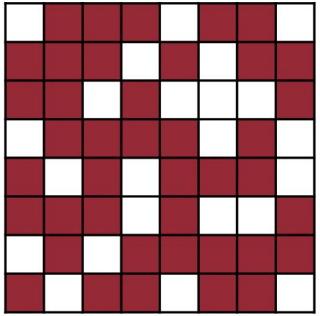
Importance

0	0
1	-5

Pruned Weight

Accuracy – Computational cost Tradeoff

From accuracy perspective it is better to prune some weights algorithm finds optimal.

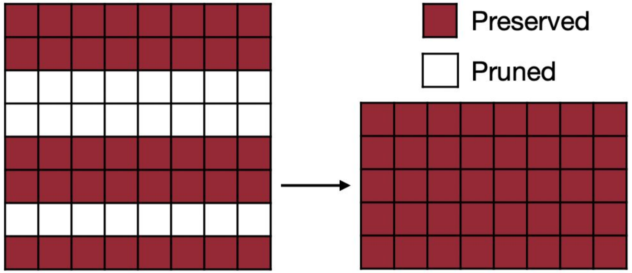


Fine-grained/Unstructured

- More flexible pruning index choice
- Hard to accelerate (irregular)

Accuracy – Computational cost Tradeoff

From computation perspective it is hard to store and operate with sparse matrices and it is better to prune some non-arbitrary patterns

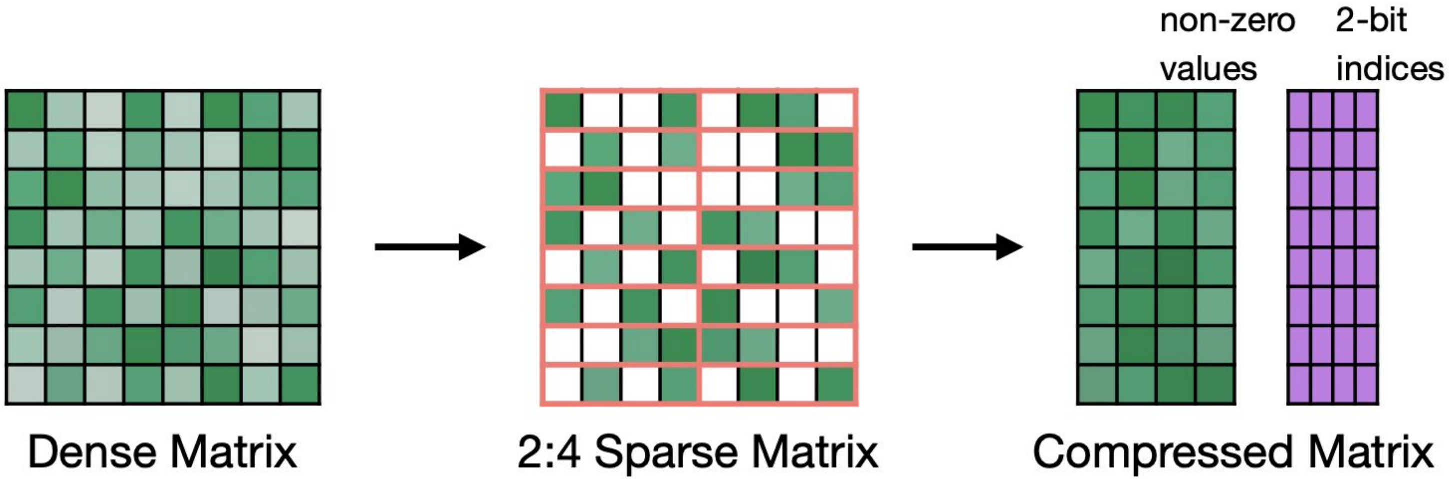


Coarse-grained/Structured

- Less flexible pruning index choice (a subset of the fine-grained case)
- Easy to accelerate (just a smaller matrix!)

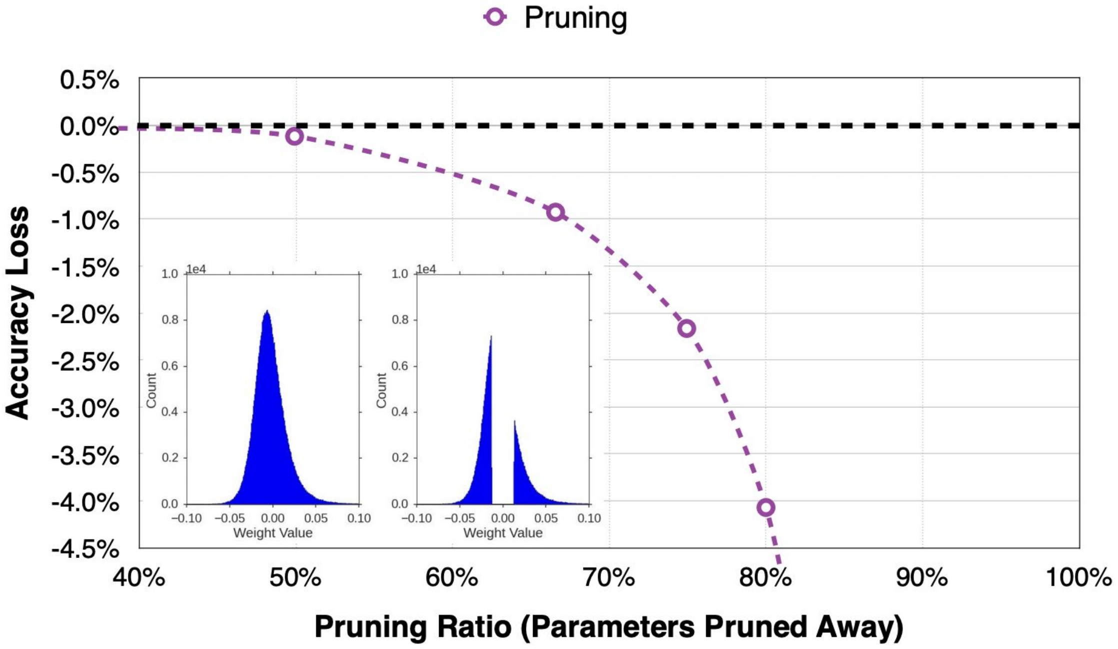
There is some support by hardware (NVIDIA)

NVIDIA supports operations with matrices such that for every m consecutive elements there are at most n nonzeros. This is called n:m sparsity



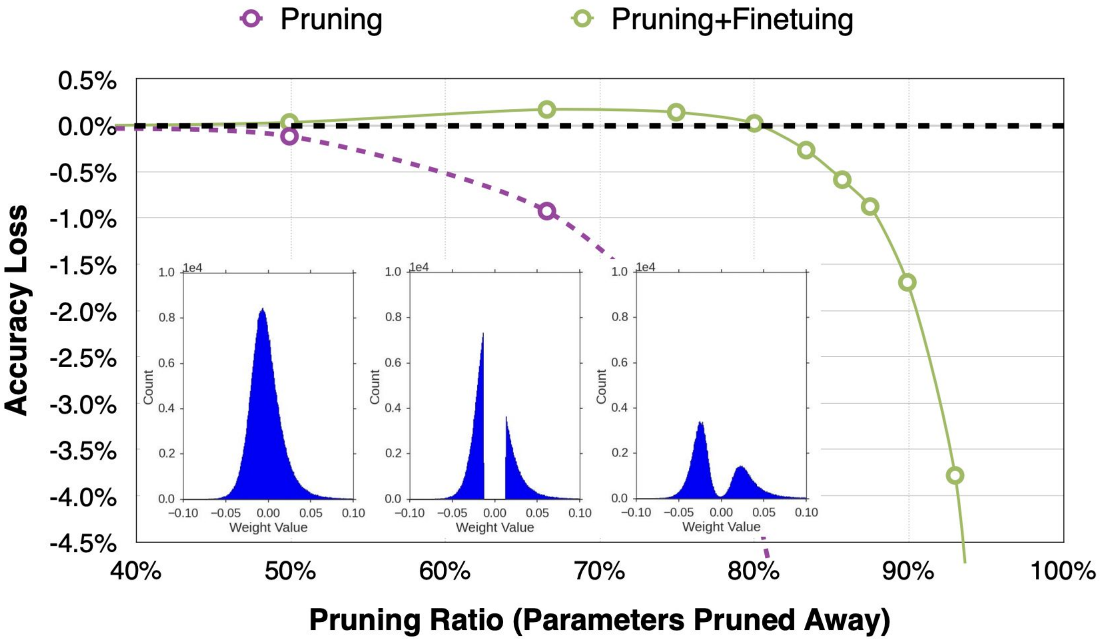
Naive algorithm

Train + prune by eliminating smallest weights.



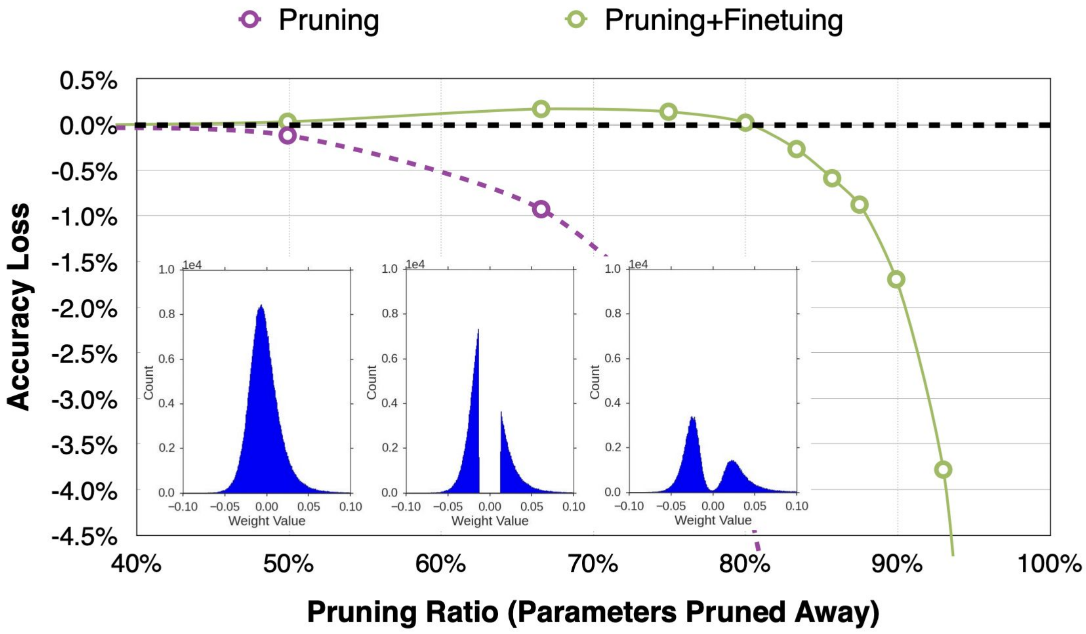
Let’s work with obtained model

Train + prune by eliminating smallest weights + **finetune**



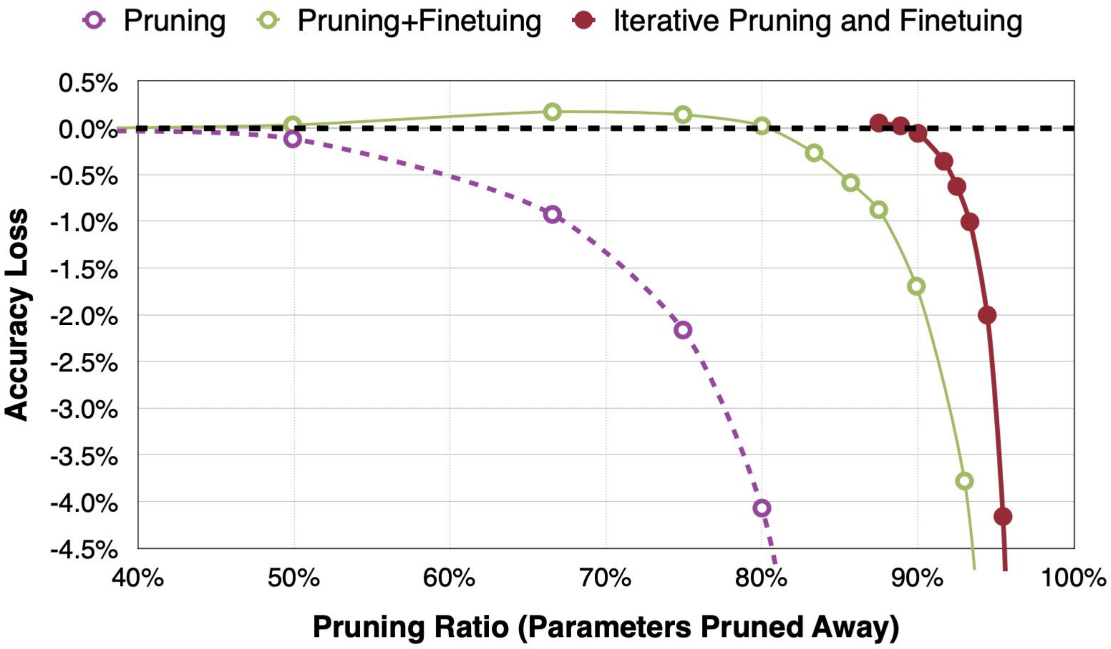
Quality increased

We’ve thrown away 80% of parameters but quality increased



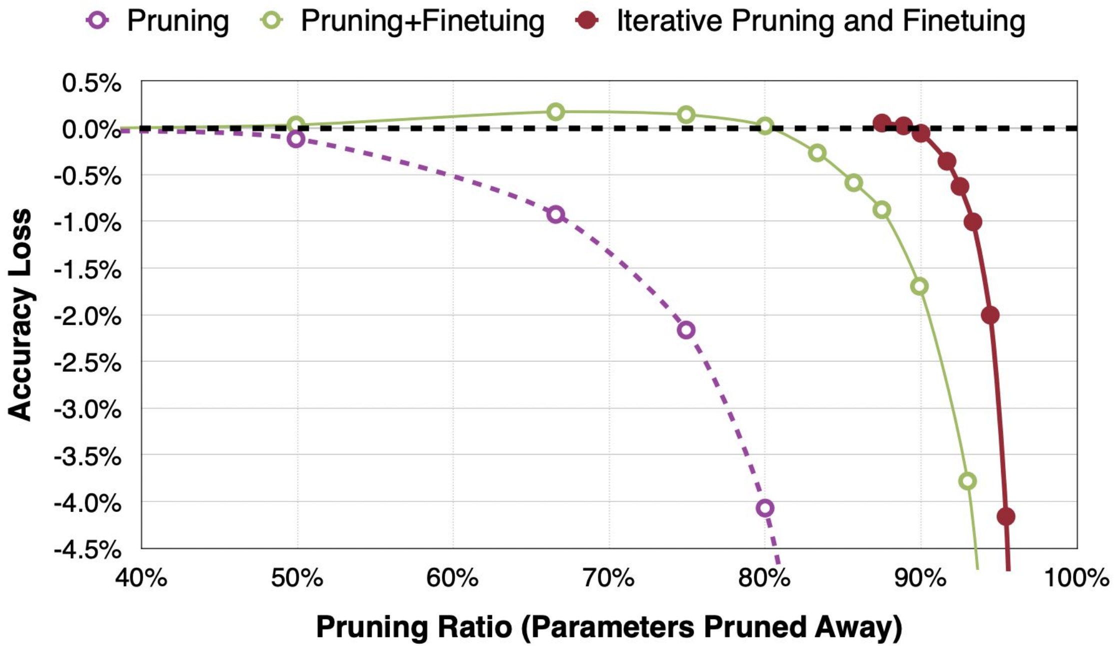
Can we repeat once again?

Train - (Prune - Finetune) - **repeat last 2 steps**



Can we repeat once again?

with less aggressive pruning ratio on each iteration after several iterations we achieve same pruning but better accuracy



General Ideas

- It is better to prune deeper layers
- It is better to prune larger layers
- Prune less each time + do more iterations

There are other approaches

- RL-based pruning
- Regression-based pruning

Training cost

Imagine you invented architecture of GPT-3 and did all the research. Now you need to find hundreds of billions of parameters. What is the cost of a single training run?

Training cost

Imagine you invented architecture of GPT-3 and did all the research. Now you need to find hundreds of billions of parameters. What is the cost of a single training run?

Each training run required at least $5’000’000

And you actually need many runs during development and tuning

Therefore, the following question appears

If we can prune models after training, can we train smallers models from the start? That would help to save resources needed for training of large models.

Lottery Ticket Hypothesis

Lottery Ticket Hypothesis

The lottery ticket hypothesis: dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation— reach test accuracy comparable to the original network in a similar number of iterations.

The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

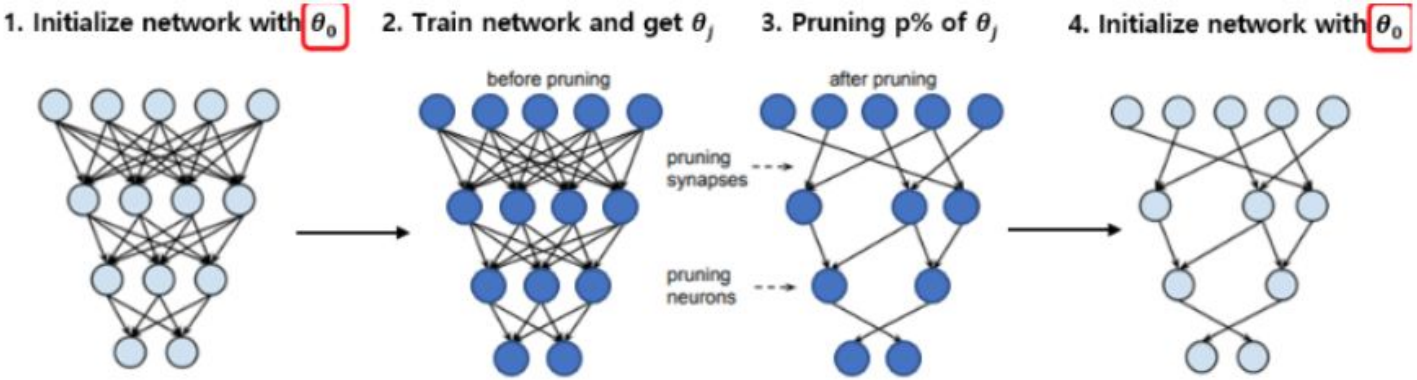
Difference

We already know how to train a big model and obtain submodel that can be finetuned.

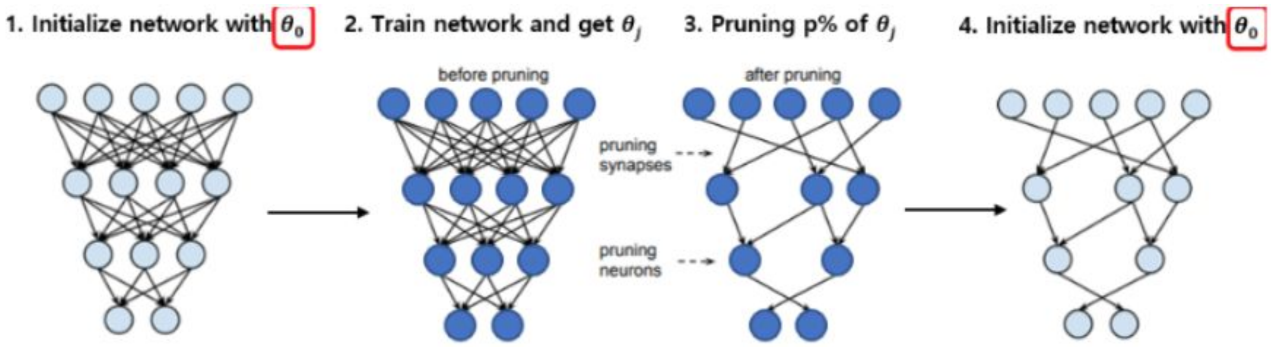
We are interested in small networks that can be trained from scratch.

Experiment

Train the model - prune it - consider subterwork - reinitialize with initial weights


1. Initialize network with $\boxed{\theta_0}$
2. Train network and get θ_j
3. Pruning p% of θ_j
4. Initialize network with $\boxed{\theta_0}$
before pruning
after pruning
pruning synapses
pruning neurons

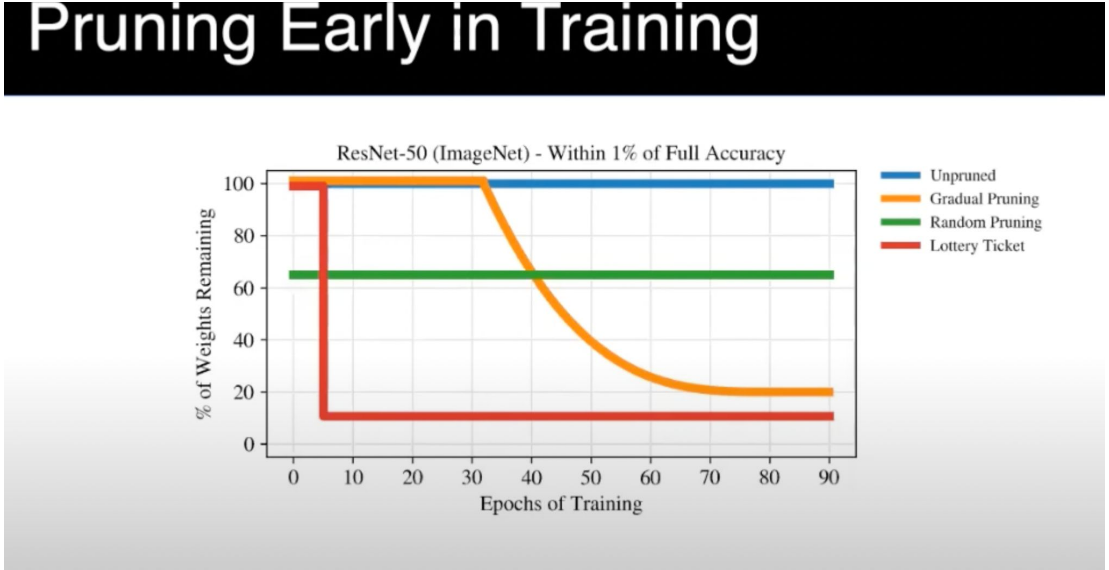
Experiment shows that hypothesis is correct


1. Initialize network with $\boxed{\theta_0}$
2. Train network and get θ_j
3. Pruning p% of θ_j
4. Initialize network with $\boxed{\theta_0}$
before pruning
after pruning
pruning synapses
pruning neurons

The last subnetwork trains from scratch pretty well.

Hypothesis is correct.

Therefore, it allows us to prune networks before training / on early steps and spend less time and resources, since there are small subnetworks that can be trained in isolation.



Pruning Early in Training

They exist, but is it possible to find them efficiently?

We already know the algorithm that requires training of a full model to find lottery tickets. How to find them faster?

We don’t know.

Applications

Experiments show that we can transfer lottery tickets between similar datasets. Therefore, you can once train model, amortize this cost once and use for another problems.

E.g. some models need to be retrained on new data regularly not to lose money.

This is found useful by Meta, Amazon and obviously can be used in trading

Applications

Future research may discover algorithms for finding lottery tickets almost without training a big model.

The Lottery Ticket Conjecture.

SGD seeks out and trains a subset of well-initialized weights. Dense, randomly-initialized networks are easier to train than the sparse networks that result from pruning because there are more possible subnetworks from which training might recover a winning ticket.

The importance of winning ticket initialization

When randomly reinitialized, a winning ticket learns more slowly and achieves lower test accuracy, suggesting that initialization is important to its success. One possible explanation for this behavior is these initial weights are close to their final values after training—that in the most extreme case, they are already trained. However, experiments show the opposite.

When randomly reinitialized, a winning ticket learns more slowly and achieves lower test accuracy, suggesting that initialization is important to its success. One possible explanation for this behavior is these initial weights are close to their final values after training—that in the most extreme case, they are already trained. However, experiments show the opposite.

References

- MIT Course TinyML and Efficient Deep Learning Computing
  - lecture 3
  - lecture 4
- PAPER: THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS
- ICLR 2019 speech
- **EI Seminar - Michael Carbin - The Lottery Ticket Hypothesis**

- **Consultant: Egor Lifar,** MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), student researcher