Neural Networks - Optimal Brain Damage

Created: Friday 28th October 2022 22:47:50
Last Edit: Friday 18th November 2022 13:14:19

s_koppelmann20@stud.hwr-berlin.de

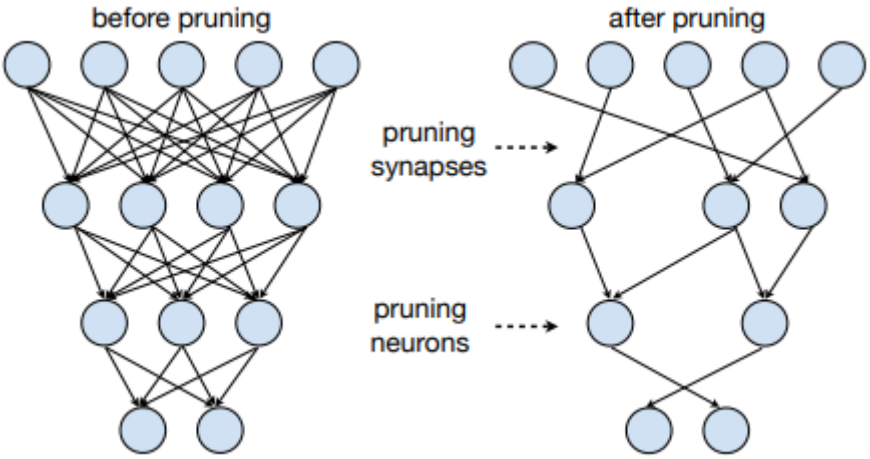
The provided outline is an extended version of the MiT 6.S965 course contents.
Papers and further resources are linked accordingly.

What is Pruning?

[Yann LeCun's paper on this (first paper on pruning)]

Neural network pruning is the process of removing internal connections (Weights and/or entire neurons) from a neural network in order to reduce its footprint. This effectively reduces the size and cost of a network **without** impacting accuracy. In some cases, it turns out to even **increase accuracy** a bit. Pruning can also be used to increase the **speed** of inference. This is important for applications, especially those running on edge devices (Smartphones, IoT), because it allows for faster inference without requiring a larger resource pool.

> Note
>
> Metaphorically, pruning means "brain damage" or eliminating unnecessary knowledge. A DNN's knowledge is understood to reside in its *weights*. Pruning consolidates the knowledge into fewer prediction operations, thereby accelerating the network's performance while maintaining accuracy.



Many if not most industrially-interesting DNN models can be decomposed into their component weight tensors. These tensors frequently contain sparsity. Therefore you should be able to

compress them and thus shrink the demands for storage, data transfer and execution time/energy of the DNN *while retaining accuracy*.

Think of it like putting each DNN tensor into a `.zip` file. It requires fewer bits than the original.

However, compressed tensors still need to be used for computation at inference-time, so we need a compression scheme that lends itself to efficient computation;

> **Note**
>
> With pruning, we do not want the compression savings to be overshadowed by the cost of decompressing the tensors for computation.

Many compression techniques exist which work by removing redundant zero-values from a tensor, such that most if not all of the **remaining values after compression are exclusively non-zero**.

It is possible to compute directly on compressed data when using such a zero-removal scheme. This is because the computations performed in DNNs are MACs, for which zero operands never cause any changes in results.

Up until now, we looked at zero-sparsity. Though in many industrially-interesting DNNs, the sparsity does not directly take the form of redundant zero-values. It rather manifests as other *hidden* patterns in the weight tensors.

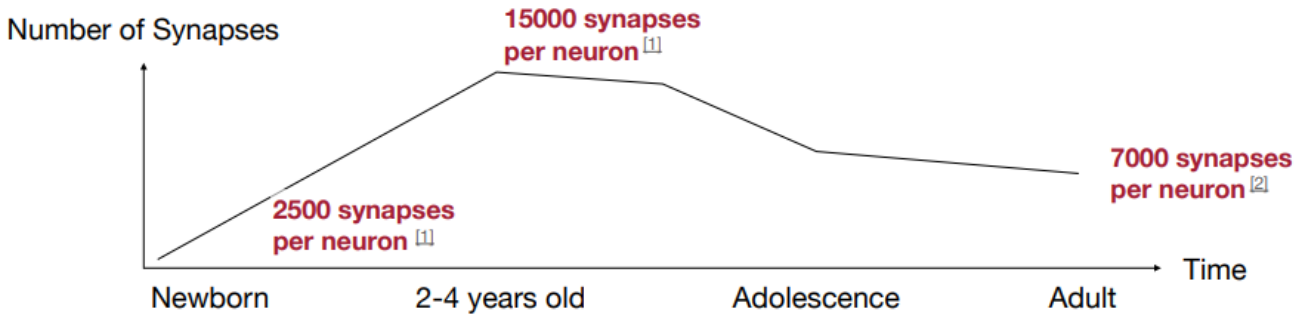
This leads to the idea of *pruning* - assuming that the tensor has *some* underlying statistical sparsity, we use *some* technique to convert this sparsity to zero-sparsity.

We flip some of the DNN weights to zero and (potentially) adjust the values of the other weights to compensate this. Since zero-valued weights are ineffectual/dead for multiply/accumulate, we treat the zero-valued weights as if they do not exist (they were "pruned"). Therefore we do not store, transfer or compute on these weights.

> **Important**
>
> In summary, we achieve savings on key metrics by converting underlying statistical sparsity to the specific form zero-sparsity. Then we rely on hardware and/or software to *exploit* this 'ignore dead neurons'-sparsity to do less.

Introducing Pruning - A Neurobiological Analogy

In neurobiology, axonal connections are understood to play a role in representing and computing on information in the brain. The number of axonal connections ramps up after birth, stays high for a period of time, until eventually a very large number of connections are destroyed toward the end of adolescence and into adulthood. Scientists assume that the brain does this in order to facilitate aggressive learning early in life, followed by honing in on only the most critical knowledge later in life. This latter stage of 'destroying' axonal connections is referred to in neurobiology as "pruning" of the neuronal axons.



The AI process of surfacing tensor sparsity through redundant zero weights and then 'ignoring' doing work on these zero weights was named "pruning" by analogy to the latter stage of axonal destruction in human development. The zero-weights have effectively been removed from consideration, and remaining DNN knowledge must be encapsulated in the smaller number of remaining non-zero weights.

Sometimes, DNNs are pre-trained before pruning, in which case the neurobiological analogy applies even more closely.

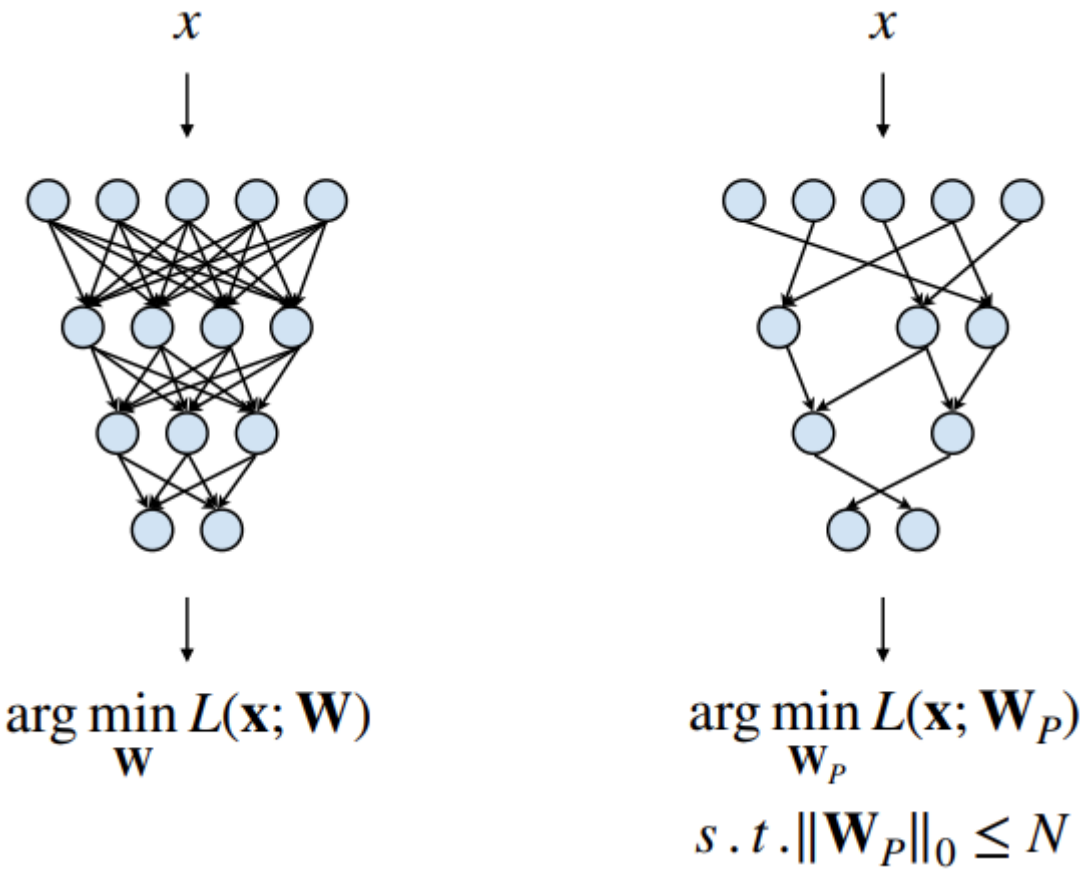
Industry Support

- Industrial pruning tools
  - AMD/Xilinx
  - Xilinx Vitis
  - Reduce model complexity by 5x-50x with minimal accuracy impact
- Key architectures for exploiting sparsity
  - Academic:
    - EIE [Han *et. al.*, ISCA 2016]
    - ESE [Han *et. al.*, FPGA 2017]
    - SpArch [Zhang *et. al.*, HPCA 2020]
    - SpAtten [Wang *et. al.*, HPCA 2021]
  - Commercial:
    - NVIDIA Ampere architecture.
      - 2:4 sparsity in A100 GPU, 2x peak performance, 1.5x measured BERT speedup

Process of Pruning

[Yann LeCun's paper on this (first paper on pruning)]

Defining the Pruning Problem (still without weight updates/retraining/fine-tuning):

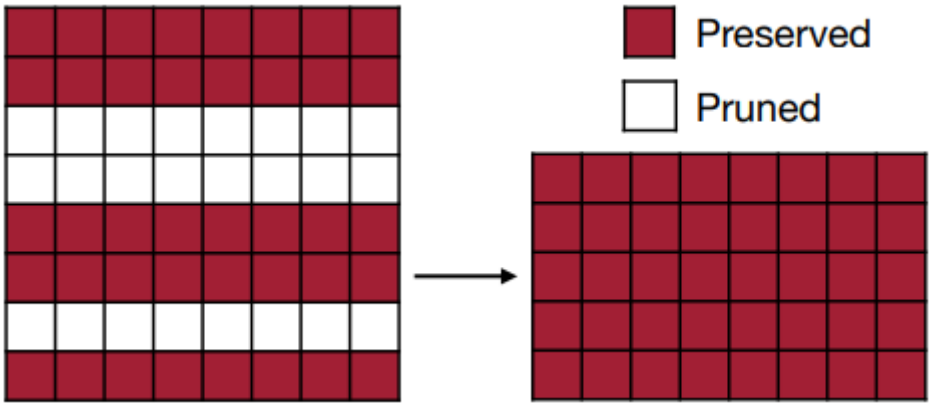


There are a number of different methods for pruning. Some of them are covered here:

Coarse-Grained Pruning

Coarse-Grained Pruning

Created: Tuesday 8th November 2022 21:39:49
Last Edit: Friday 18th November 2022 13:14:19



We apply a Coarse-grained pruning constraint: Remove entire channels at once, effectively changing layer/model geometry.

✓ Pros
Feasible to exploit on CPU and GPU. Equally possible to exploit with custom hardware.

✕ Cons
Lowest recovered accuracy/compression ratio compared to above mentioned due to barely any flexibility.

There exist a number of intermediate pruning granularities between fine-grained and channel-pruning, which we just mention quickly: - Vector-level pruning - Kernel-level pruning

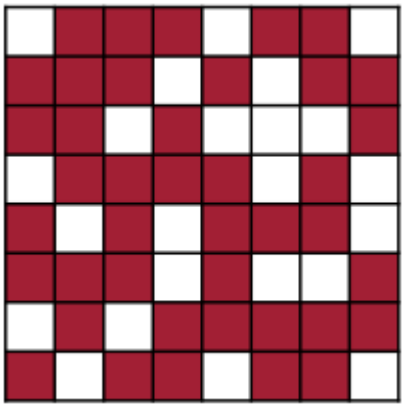
Note
If you want **extreme compression**: Use **unstructured** pruning.
If you want **extreme performance**: Use **structured** pruning.

Fine-Grained Pruning

Fine-Grained Pruning

Created: Tuesday 8th November 2022 21:36:56
Last Edit: Friday 18th November 2022 13:14:19

Fine-Grained Pruning aka Unstructured Pruning:



We apply flexible pruning indices, meaning essentially "random access" to prune any weight in any tensor at any interation individually.

✓ Pros
High recoverable accuracy/higher compression ratio applicable, retaining the same accuracy. → This is a result of this approaches' flexibility to finely match the underlying patterns of redundancy in the weight set.

✕ Cons
There is the potential to exploit/use this approach in custom hardware. But its challenging or **impossible to use effectively** on CPU/GPU.

Magnitude-Based Pruning

Created: Tuesday 8th November 2022 21:42:55
Last Edit: Friday 18th November 2022 13:14:19

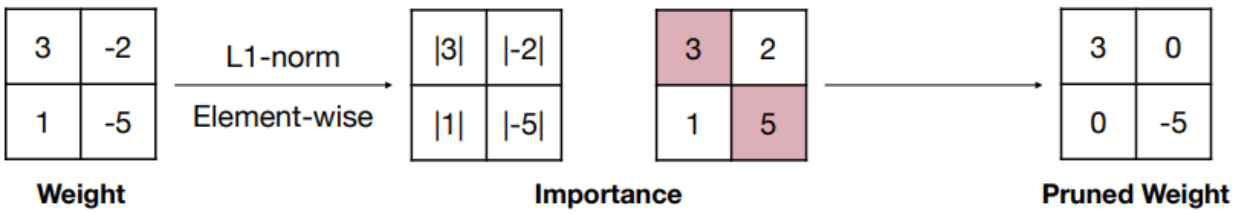
Magnitude-based pruning means that we prune the smallest weight values, as they are interpreted as the most unimportant ones.

Magnitude pruning is one of the most popular methods for pruning. Magnitude pruning is based on the assumption that the weights with the smallest magnitudes are the least important. So magnitude pruning involves removing the weights with the smallest magnitudes.

Magnitude pruning can be done iteratively, where the network is pruned multiple times. Each time the network is pruned, the network is trained for a small number of epochs, and then pruned again. This can be done several times until the desired level of pruning is achieved.

There are also a number of different ways to implement magnitude pruning. One way is to simply remove the weights with the smallest magnitudes, and then retrain the network. Another way is to gradually prune the network. In this method, a threshold is chosen, and then all weights with magnitudes below the threshold are pruned. The network is then retrained, and the threshold is gradually increased. This is done until the desired level of pruning is achieved.

$$Importance = |W|$$



> **Important**
> Magnitude-based pruning is a heuristic score. We just assume that this lower value weight is the more unimportant one. This assumption works pretty well though.

For fine-grained pruning, the ranking of individual weights based on the above importance score will be the same as the ranking based on any norm (L1, L2, etc.)

For other pruning granularities (e.g. row-wise pruning), one can distinguish formulae for aggregating the importance of groups of weights

- L1-norm criterion: $Importance = \sum_{i \in S} |w_i|$
- L2-norm criterion: $Importance = \sqrt{\sum_{i \in S} |w_i|^2}$

- Lp-norm criterion: $Importance = (\sum_{i \in S} |w_i|^p)^{\frac{1}{p}}$

First-Order-Based Pruning

First-Order-Based Pruning

Created: Tuesday 8th November 2022 21:21:40
Last Edit: Friday 18th November 2022 13:14:19

[Molchanov *et al.*, ICLR 2017]

This is an approach to prune not weights, but activations.
Another Taylor-Series based technique. Literally just mentioned for the sake of note completeness.

Structured Pruning

While magnitude pruning is very simple, it does not necessarily lead to a network that is fast enough for edge devices. This is because the resulting network is likely to have a lot of unstructured sparsity. Unstructured sparsity is sparsity where the weights that have been pruned are scattered throughout the network. While this leads to a network with a small number of weights, it does not necessarily lead to a network that is fast enough for edge devices. This is because a lot of edge devices cannot take advantage of unstructured sparsity. Instead, these devices can only take advantage of structured sparsity. Structured sparsity is sparsity where the weights that have been pruned are grouped together in some way. For example, a common form of structured sparsity is channel sparsity, where all of the weights in a given channel are pruned.

To take advantage of structured sparsity, structured pruning is often used. Structured pruning is the process of removing channels, filters, layers, or other groups of weights.

Pruning for Quantization

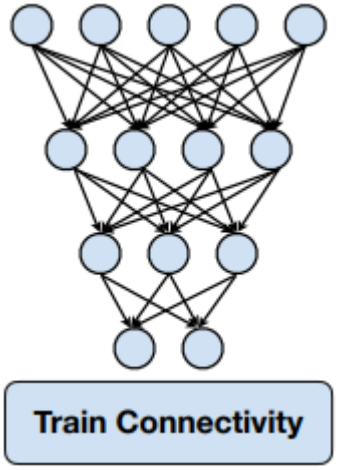
It is also important to note that pruning can be used in conjunction with quantization. Pruning can be used to reduce the size of a network, which can then be quantized. Quantization can be used to reduce the size of a network, which can then be pruned. Pruning and quantization can also be done simultaneously.

Other Methods for Pruning

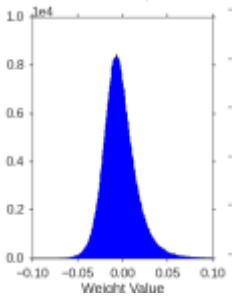
There are also a number of other methods for pruning. These include Hessian based pruning, which uses the Hessian of the loss to find the weights that can be pruned, and data-free pruning, which uses only the weights and not the training data.

Pruning Workflow

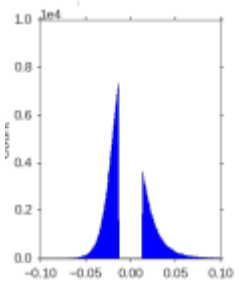
1. Train connectivities with gradient descent/stochastic gradient descent
   (we train the full, unaltered DNN with all weights to later figure out which are important)



This way, we get a weight distribution like this:

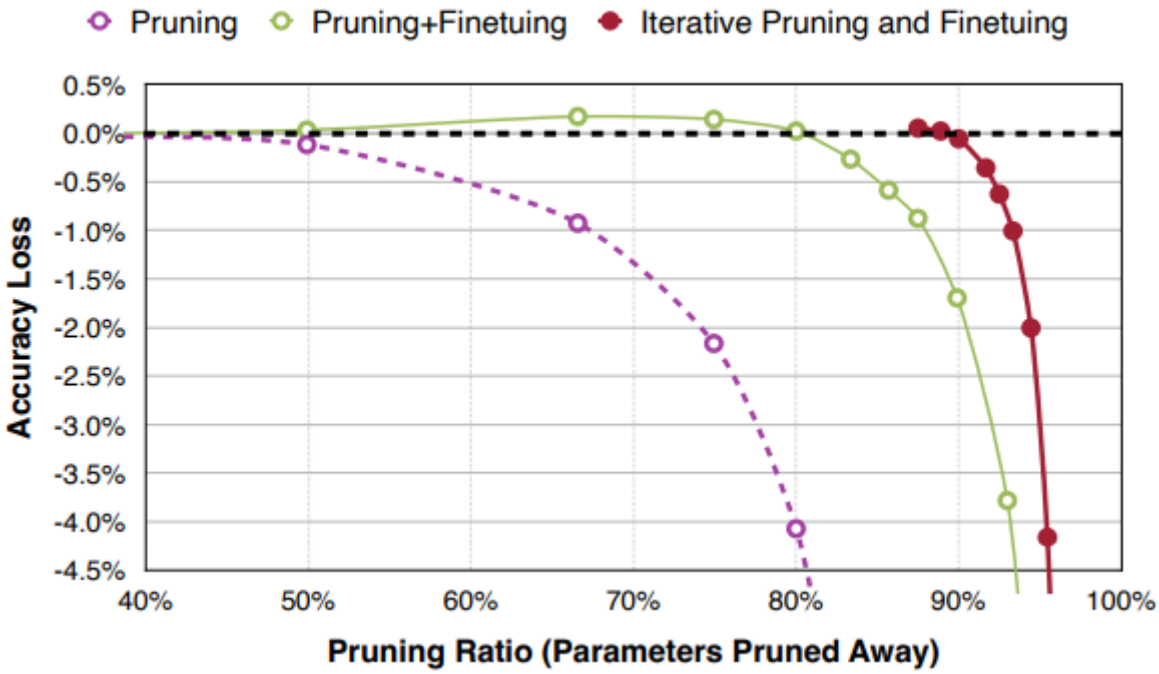


2. Prune away connections - as described above, we use a theshold: If a weight is within that (around zero), set it to zero/turn it off. The larger the threshold, the bigger the effect:



3. Train/Re-train/Finetune the remaining weights - again with gradient-descent/SGD, a smaller learning rate and a mask that holds pruned weights at zero, to adapt the *remaining* weights and recover accuracy compared to the pre-retrained pruned model. Also see Weight Updates and Iterative Pruning

> **Important**
>
> Using the above described three steps, we can prune the model so that **accuracy is retained at only 20% of the intial weights**. But: **We can repeat step 2 and 3 iteratively.** This retains accuracy even after removal of **90%** of the weights.



- Additional notes
  - Select specific weights or groups of weights to prune
  - Weights become zero
  - Zero weights may be skipped entirely from computation ⇒ time ($), energy savings
  - Zero weights do not need to be stored ⇒ storage & data transfer savings
  - Neurons for which which all ingress weights are zero can itself be pruned ⇒ prune all egress weights from that neuron ⇒ additional execution & storage savings

It is very interesting that we can train an NN, remove a sizable margin around zero-strength weights, retrain and through that recover accuracy. Let's look at that in more detail.

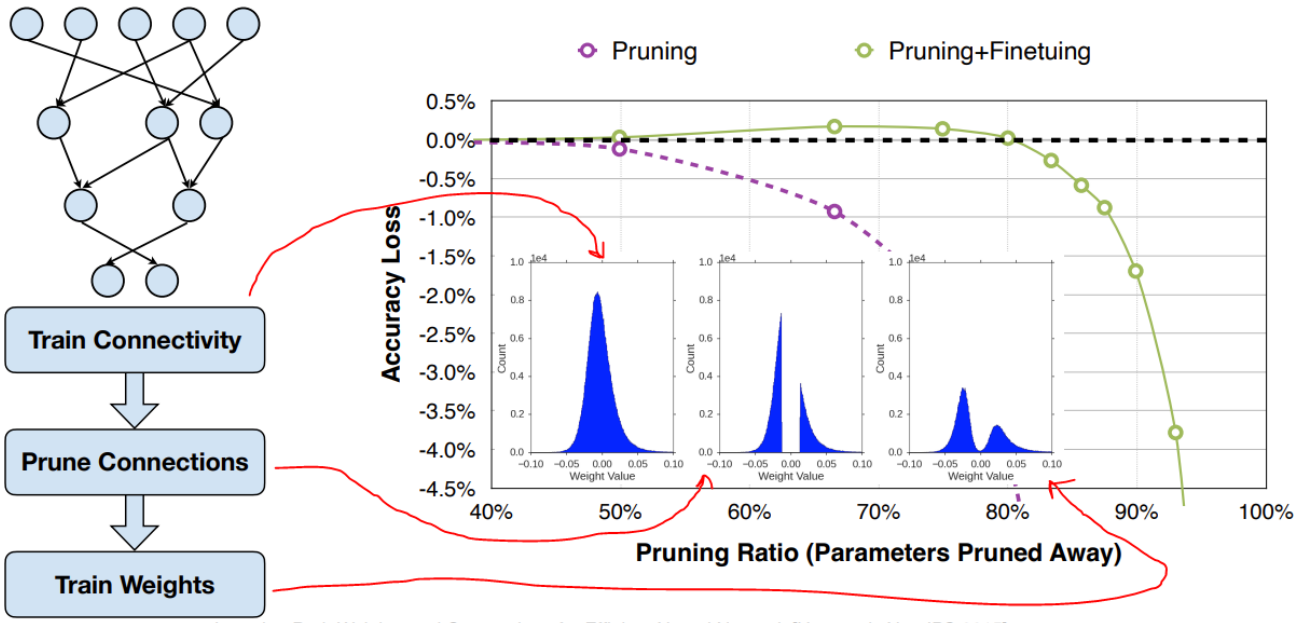
Weight Updates and Iterative Pruning

With Step 2 we force “Brain damage” upon the DNN. Left untreated, this causes lower accuracy. But we can treat this and actually recover accuracy to old levels through updating the left-over weights. Thus, we adapt to / recover from the "Brain damage". Interestingly, we can also improve accuracy even more by pruning the model incrementally, meaning in small increments of sparsity factor.

The Impact of Pruning, Weight Updates, and Iteration

Let's assume that a variant of magnitude-based pruning is used, but the ideas will be similar regardless.

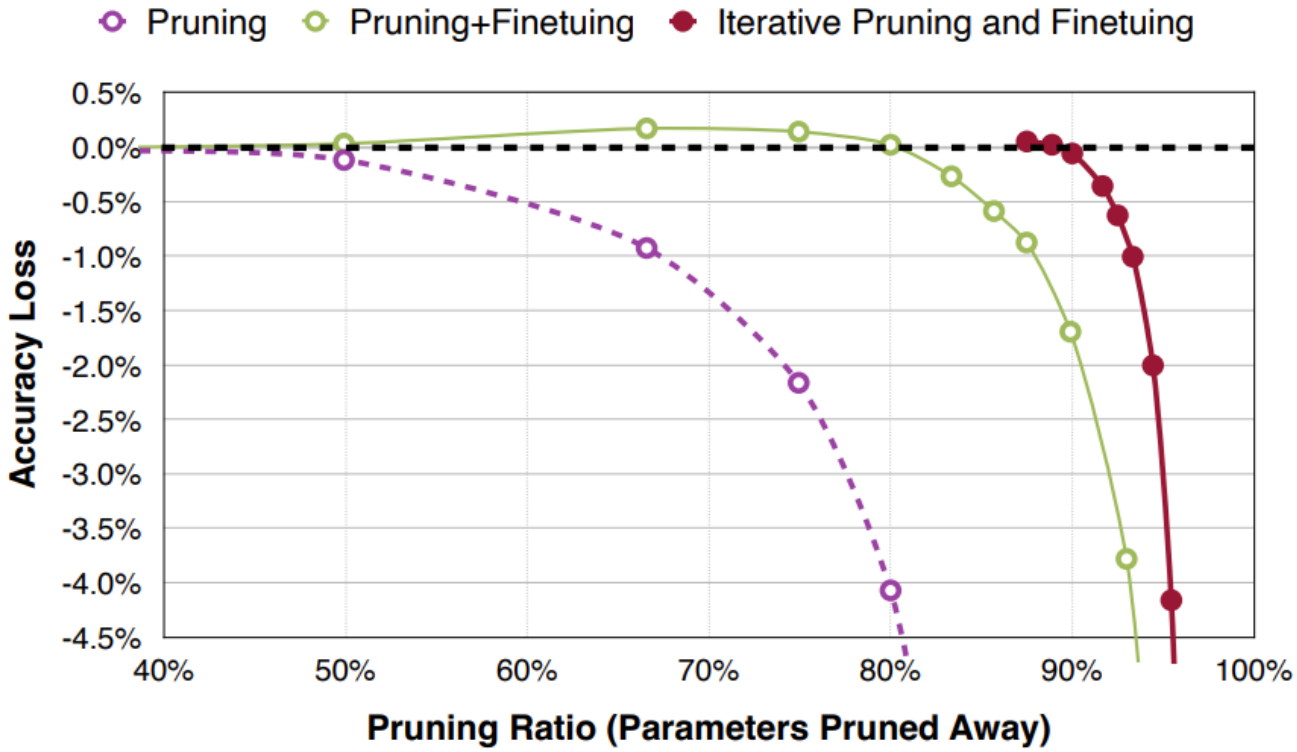
Each step of the pruning process has a distinctive effect on the weight distribution.
Without other prior knowledge, imagine that the initial weight distribution is Gaussian-like.
Pruning cuts away those weights with absolute magnitude close to zero (with a margin), leaving a bi-modal distribution of weight values above and below zero.
Retraining/fine-tuning/weight-update increases the spread of these distributions, owing to weight adjustments to recover accuracy (i.e. correct the brain damage)



Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

> **Note**
>
> Pruning sacrifices accuracy, with increasing severity with respect to the sparsity factor.

The accuracy/sparsity trend is a relatively gradual slope. Retraining/fine-tuning/weight-updating recovers accuracy. Additionally, there is now more sudden drop-off of accuracy at a certain sparsity/pruning intensity.
Adding iterative training cycles recovers accuracy and maintains the most uniform accuracy, with nearly a brick-wall accuracy drop-off at very high sparsity factors



> **Note**
>
> There is actually a slight improvement in accuracy over the course of aggressive pruning, which we might hypothesise is due to more effective avoidance of overfitting.

Common Pitfalls in Pruning

There are a number of common pitfalls when pruning. These include the following:

Overfitting

One common pitfall when pruning is overfitting. If the network is pruned too much, it can overfit to its training data. This is especially true if the network is pruned too much early in training. If the network is pruned too much early in training, it may not be able to recover its accuracy later in training.

Incorrect Pruning

Another common pitfall when pruning is incorrect pruning. Incorrect pruning is when weights are removed that are actually important. Incorrect pruning can lead to a network that is not accurate enough for its use case. It can also lead to a network that cannot be retrained.

Incorrect pruning can happen in a number of ways. One way is when weights are pruned based on their magnitude. This can lead to incorrect pruning because it assumes that the weights with the smallest magnitudes are the least important. However, this is not always the case. Another way is when weights are pruned based on their impact on the loss. This can lead to incorrect pruning because it assumes that the weights with the smallest impact on the loss are the least important. However, this is not always the case.

Incorrect Pruning and Recovery

Another common pitfall when pruning is incorrect pruning and recovery. Incorrect pruning and recovery is when weights are removed that are actually important, but the network is able to recover its accuracy later in training. This is common because it is usually easier for a network to recover its accuracy if it is pruned later in training. This is because a network will have learned more about the training data later in training, so it will have more information to recover from the pruning.

Incorrect pruning and recovery can happen in a number of ways. One way is when weights are pruned based on their magnitude. This can lead to incorrect pruning and recovery because it assumes that the weights with the smallest magnitudes are the least important. However, this is not always the case. Another way is when weights are pruned based on their impact on the loss. This can lead to incorrect pruning and recovery because it assumes that the weights with the smallest impact on the