# Pruning Deep Neural Networks

By Ashish Bhattarai

November 16, 2021

## Introduction

Deep neural networks have demonstrated great success in wide range of tasks from computer vision to natural language processing, speech recognition, game playing and a variety of other tasks from diverse areas. With the higher complexity of problem at hand, typically the corresponding number of network parameters and the dataset size grows larger. An example is the transformer-based language prediction model GPT-3 with 175 billion parameters.

Pruning is a technique of neural network compression by removal of redundant or low contribution nodes and connections from the network architecture. Similar to the pruning of the unnecessary branches from a heavy tree, pruning eliminates unnecessary parts inside a dense neural network graph (see Fig. 1). Pruning can be applied for optimizing the size of and computations inside a neural network while ensuring that the performance of the model does not deteriorate in a significant way.

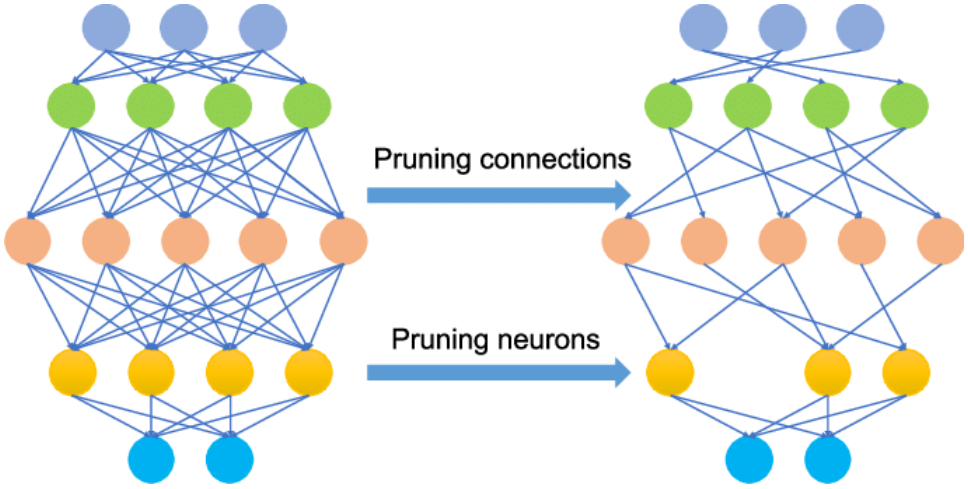


Figure 1: Pruning of a dense neural network results in a sparse model [1]

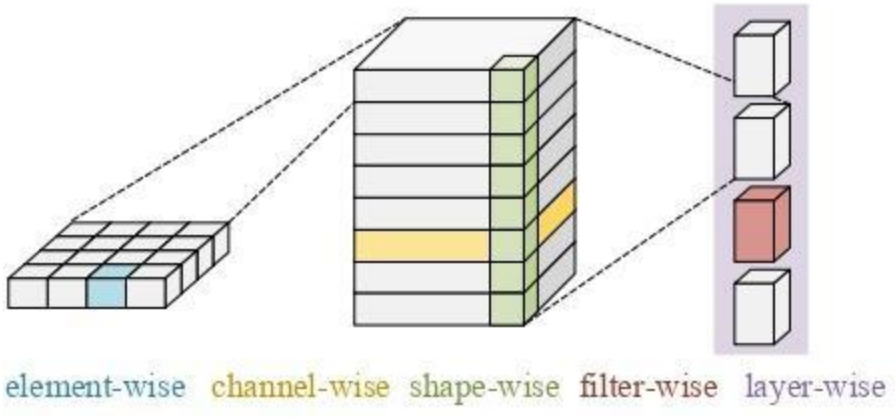


Figure 2: Granularity of pruned structures [2]

## Why Is Pruning Useful?

Pruning becomes particularly relevant when the DNN models are to be deployed in a production setting under strict latency and resource utilization constraints. Pruning reduces bandwidth as well as memory foot-print of a neural network. One pertinent question to ask is--why not always start with a small parameter neural network, rather than trying to reduce the parameters afterwards? Indeed, algorithms such as cascade correlation follow the same principle. For a single hidden layer mlp, such bound has been discussed by Huang et al in [3]. So, if we choose to start with a large parameter neural network with just one hidden layer and later prune it, upper bound on number of hidden units is already available to create an initial network.

However, it should be noted that it's in general hard to come up with a good bound on the number of parameters required beforehand as well as design the corresponding architecture that solves the problem for deep neural networks for arbitrary task. Such bound could be affected by the diversity and size of input data, the structure of neural network, choice of objective function as well as the parameter optimization technique used. This is not much a problem in practicality since we already have several successful architectures for different tasks that we can readily prune.

Aside from the theoretical considerations, it has been practically observed that having a larger parameter space generally allows networks to escape out of local minima of the loss function during training. However, once trained, significant number of parameters can be eliminated due to redundant functionalities and the network downsized with little to no loss of accuracy. In fact, in some cases the test accuracy might even increase due to better generalizability resulting from

smaller number of parameters. Thus, it's preferable to start with a larger neural network to learn a task and later reduce its size with pruning.

## Criteria for Pruning

Pruning can either be targeted towards specific neurons or towards the connections, and can result in different structural changes inside the network graph (see Fig. 2).

Mathematically, pruning can be seen as minimization of following objective:

$$\underset{\mathcal{P}}{\operatorname{argmin}}\, E = \{M(X;\theta) - \mathcal{P}(M(X;\theta))\}$$

where M is an unpruned neural network, which is a function of input dataset **X** and parameters $\boldsymbol{\theta}$, **P** is a pruning function applied to M that results in a sparser parameter set $\boldsymbol{\theta_P}$. Minimization of the error function **E** ensures that the performance of a pruned model stays close to the unpruned one.

Even though the above objective provides a good sense for what we want to achieve, brute-force searching for the optimal pruning technique is not always feasible due to a possibly large solution space. LeCun initially came up with idea of Optimal Brain Damage (OBD) based on Taylor linearization of the error function to get Hessian with respect to parameters that gives a good pruning criterion. Thisidea was later extended to the work for Optimal Brain Surgeon (OBS) that could downsize simple XOR network by 90%. However, given the huge parameter space of modern neural networks, computation of Hessian to identify prunable parameters can be very slow or computationally infeasible.

Simple practical pruning algorithms for neural networks can be based on one of the following two criteria:

- **Magnitude-based pruning**

  The idea behind magnitude-based pruning is that norm or magnitude of weight values with respect to some threshold value is to be used to identify and eliminate the unnecessary parameters. For instance, prune all the zero weights.

The choice of p $\epsilon$ [0, $\infty$] for suitable $l_p$ norm has effect on the type of sparsity of the neurons resulted by pruning. For instance, use of $l_0$ norm identifies non-zero weights, $l_1$ keeps weights with larger element-wise absolute values, $l_2$ measures sufficient deviation from mean, and so on.

- **Penalty-based pruning**

  Penalty based methods push certain weight values towards zero during training by either modification of the loss function or introducing layer-wise constraints into the network. For eg., LASSO is a popular penalty technique for enforcing sparsity during training with a constraint on $l_1$ norm of weight values. Group LASSO is an extension of LASSO that further performs grouping of sparse weights that are simultaneously removable. This creates a structured pruning that has efficiently executable weights than that are obtainable with just element-wise unstructured pruning.

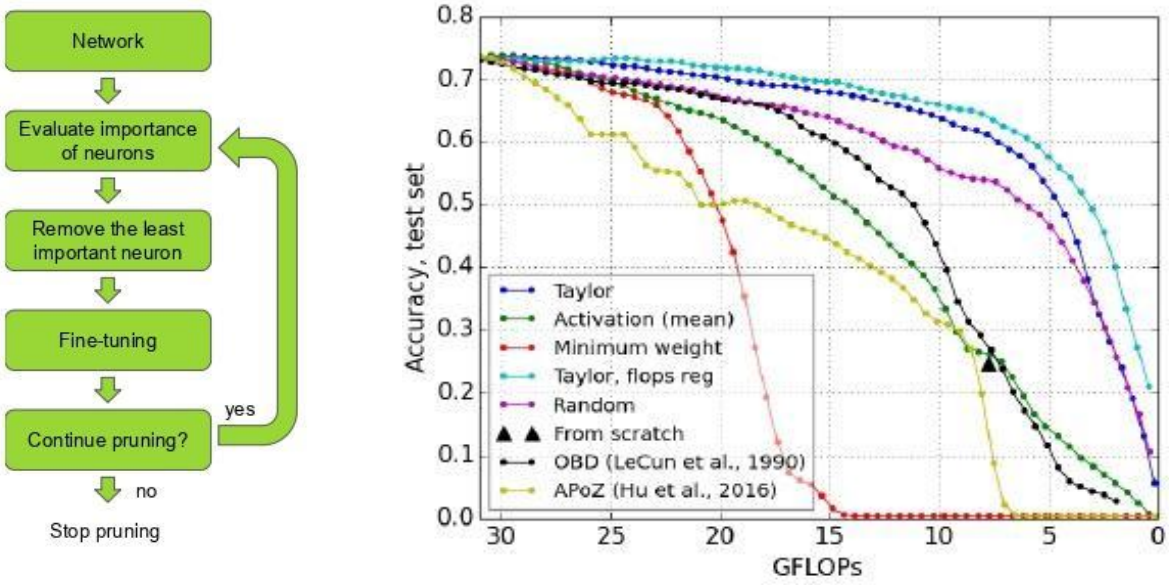


Figure 3: (**Left**) Iterative pruning strategy and (**Right**) corresponding accuracy vs computations plot for pruning of VGG-16 model under different criteria [4]

## The Benefit of Retraining

Pruning generally eliminates all weights having near zero or redundant contributions with respect to selected metric. However, such metric may not always align with the desired functionality of

network and can sometimes cause significant drop in accuracy. Retraining the sparse network post pruning can help improve and restore the network accuracy with respect to original objective. The evaluation of success of pruning procedure can be done in terms of parameter count or number of floating-point operations. Pruning with retraining and fine-tuning can be performed in a single shot or as multi-iteration procedure. In Fig. 3, we observe that a Taylor criterion-based iterative pruning performs comparatively better than simple magnitude-based pruning strategy.

## Final Comments

In addition to the static pruning methods discussed above, dynamic pruning methods have also gained a good traction for searching optimal network structures. The motivation behind these has been that static pruning techniques can cause irrecoverable change on pretrained network and can lead to sub-optimal network architecture. Instead, identification of redundant parameters during run-time itself should lead to identification of more efficient light-weight networks. Different reinforcement learning techniques, as well as differentiable methods, have been put forward with this aim. In contrast to the static pruning, since the elements to prune is to be identified at run-time itself while optimizing parameters, in the case of dynamic pruning the computational overhead can be considerably higher.

**References**

[1] Chen, J., Ran, X. “Deep Learning With Edge Computing: A Review”. Proc. IEEE, 107, 1655–1674 (2019).

[2] Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. 2021. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 461: 370–403.

[3] Huang, S. C. and Huang, Y. F. “Bounds on the number of hidden neurons in multilayer perceptrons,” IEEE Trans. Neural Networks, vol. 2, pp. 47–55, 1991.

[4] Molchanov, P., Tyree, S., T. Karras, Aila, T. and Kautz J. Pruning convolutional neural networks for resource efficient transfer learning. In ICLR, pages 1–17, 2017.

[5] LeCun, Y., Denker, J.S., Solla, S.A., 1990. Optimal Brain Damage, in: Advances in Neural Information Processing Systems (NIPS), p. 598–605.

[6] Castellano, G., Fanelli, A., Pelillo, M. An iterative pruning algorithm for feedforward neural networks. IEEE Transactions on Neural Networks 8: 519–531 (1997).