Burnt Sienna

# Sparsity in Neural Networks

*By Daniel Ernst, ECE '22*

## Introduction

Neural networks (NNs) have become increasingly important in the computing landscape. Neural networks are able to infer properties of input data with relatively high accuracy, even for computationally hard problems. Popular applications of NNs are: natural language processing (NLP), image recognition, and prediction models such as content recommendation.

This tech note is heavily informed by "Efficient processing of deep neural networks" [1]. If you would like more information on this topic, this source will do you well.

## Components of a Neural Network

NNs are able to make good inferences on difficult problems by leveraging a large number of computations and statistical correlations in the problem space. In deep neural networks (DNNs) -- which represent the lion's share of NN architectures today -- this is done through a number of independent, interconnected layers. Shallower layers extract low-level feature information from the input data such as the presence of edges or curves in an image. Deeper layers then use this feature information to make high level inferences about the input, resulting ultimately in a prediction.

The two common types of layers are convolutional and fully connected (FC).

Convolutional layers: a set of weight matrices making up a filter is passed over the input feature map (ifmap). These ifmaps consist of the input data or features extracted from previous layers, formatted as a 2 or 3 dimensional array, where the dimensions are width, height, and # of input channels. Examples of channels in an ifmap are red, green, and blue subpixel values in an image. The resulting weighted sum of the weight matrix and ifmap gives the values of the output feature map (ofmap).

Another, perhaps easier to understand usage of weight filters is in image processing, as can be read about here [2].

FC layers: each output value is a weighted sum of every input value. In our use case, this is performed as a very large matrix multiplication.

## Sources of Sparsity

### *Quantization*

All types of sparsity, but mostly that of weights, benefit greatly from or even necessitate *quantization*. When a neural network is being trained, weights and data are typically represented as 32-bit floats. This data type has the benefit of being able to represent a large range of values (from roughly $10^{-38}$ to $10^{38}$) with fairly high precision. However, floating point computation is very energy intensive and slow, and many of the benefits of floating point go unused.

Quantization is the process of taking our trained weights and removing unneeded precision from them. This can take the form of converting to a less precise floating-point type (16 or even 8-bit), or convert our weights to an integer datatype. This process requires recalibrating our weights to maintain most of our inference accuracy, but some of this accuracy is likely to be lost.

This process can make it easier to prune a trained neural network (see next section). Moving to a smaller data type and/or an integer representation also significantly lowers the energy and time required to perform mathematical operations, as mentioned earlier.

### *Pruning*

DNNs are generally overparameterized at construction, as having more parameters than strictly needed can facilitate model training. After being trained on a dataset, many parameters (weights) in the model are close to zero, or otherwise contribute little to the end result. These ineffectual weights are “removed” (i.e., set to 0) in a process called pruning. Unlike with ifmaps and ofmaps, weights are static to a trained model, and thus are known before any input data is actually processed. This can potentially be used to better schedule computations.

### *Data compression*

In addition to weight sparsity, activation (input data) sparsity can be created, primarily through the choice of particular activation functions. The purpose of these functions is to take a weighted sum from a neuron's previous layer and produce some nonlinear behavior after performing the sum. The ReLu (rectified linear) activation function is the most popular choice for creating activation sparsity, as any negative values result in a 0, while positive values are allowed through. Generally, deeper layers will have higher activation sparsity as the network approaches a final inference.

## Leveraging Sparsity

DNNs generally require a significant amount of memory, and have a significant amount of data movement. These factors lead to substantial stress on DNN memory systems, producing more heat and potentially making inferences take longer than they would otherwise. Reducing this data movement would lower power consumption, improve device life, and potentially raise throughput.

Sparsity in weights and activations allows designers to leverage compression techniques to reduce stress on DNN memory systems. By only storing nonzero values (and encoding their locations), the amount of data to be transferred or stored decreases substantially. However, care must be taken such that time spent encoding or decoding these compressed representations doesn’t interfere with the operation of the DNN accelerator. The ideal compression scheme is one which matches the dataflow of your accelerator (thus minimizing decoding overhead). Some examples of popular compression schemes are compressed sparse row (CSR), compressed sparse column (CSC), and bitmap.

Another potential way to take advantage of sparsity is in skipping computations. By far the most common operation in DNNs is multiply-accumulate $y=y+iw$. In accordance with sparsity, one or both of the input $i$ or weight $w$ will occasionally be 0. In this case, the multiplication can safely be skipped, saving energy and potentially time. However, in the context of an accelerator, this will often have to be done by modifying each processing element (PE) which could potentially outweigh the benefits of skipping some multiplications, depending on how its effect on overall resource consumption or area on-chip, and the amount of leverageable sparsity in the common case.

## Conclusion

Natural or intentionally created sparsity in neural networks gives computer scientists and engineers a variety of ways to improve throughput and reduce power consumption. This reduction in power consumption can then be used to improve device longevity through lowered thermal stress. However, leveraging sparsity can increase our design complexity or even hurt performance if poorly implemented, or if the specific application doesn’t have much sparsity to take advantage of.

## References

1. Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2020). Efficient processing of deep neural networks. Synthesis Lectures on Computer Architecture, 15(2), 1-341.
2. Wikipedia contributors. (2022, March 14). Digital image processing. In Wikipedia, The Free Encyclopedia. Retrieved 22:00, April 8,

2022, from
https://en.wikipedia.org/w/index.php?title=Digit
al_image_processing&oldid=1077171134