Inducing and leveraging both weight and activation sparsity in deep neural networks

Mahdi Nikdan, Dan Alistarh

IST Austria

mnikdan@ist.ac.at

Summary

Deep neural networks have been very popular in recent years because of their remarkable performance on a variety of tasks. State-of-the-art networks, however, usually have millions or even billions of parameters. Because of this, storing the model parameters on memory becomes challenging and the inference process becomes time-consuming, especially when running on a CPU. To tackle this problem, researchers have been trying to induce sparsity in the weights or layer activations of the model. When one of the two is highly sparsified, we can exploit sparse operations to gain significant speed-ups. On the other hand, sparsifying the weights can also reduce the size of the model in memory.

In this report, we first introduce some recent methods that induce weight or activation sparsity in neural networks (Section 1). Then, we study how the weight and activation sparsity correlate (Section 2). Next, we try to induce both weight and activation sparsity at the same time (Section 3). Finally, we try to propose a method that benefits from high sparsity in both weights and activations to gain speed-ups (Section 4).

Note: This works has been done as a rotation project at IST Austria. Because of the limited

time, some results and experiments might be incomplete.

Key words: Activation Sparsity, Deep Neural Network, Weight Sparsity

1. Related Work

1.1 *Weight Sparsity*

Structured vs unstructured weight pruning: The approaches for pruning the weights of a network can be clustered into two classes, i.e., structured and unstructured pruning. In structured pruning, the weights of the model are divided into groups (e.g., filters in a CNN), and we will either keep a group or prune it altogether. On the other hand, in unstructured pruning every weight can be pruned individually. Intuitively, unstructured pruning can achieve higher sparsity rates, while structured pruning is more hardware-friendly and can lead to higher speed-ups.

Magnitude Pruning: A simple heuristic is that the weights with smaller absolute values are probably less important than the weights with larger absolute values. Based on this heuristic, a common approach is to prune the weights with smallest absolute values.

WoodFisher [Singh and Alistarh (2020)]: This paper introduces a pruning statistic for each weight of the model, which is the minimum change in the loss when this weight is removed and other weights are updated optimally. If we define the weights of the model as a d dimensional vector W, the pruning statistic of a single weight w_q $(1 \leqslant q \leqslant d)$ is defined as $p_q = \frac{w_q^2}{2[H^{-1}]_{qq}}$, in which $[H^{-1}]_{qq}$ is the q'th diagonal element in the inverse Hessian matrix. This paper also presents an efficient way of approximating the inverse Hessian using the Woodbury Matrix Identity [Woodbury (1950)].

AC/DC [Peste *and others* (2021)]: In this paper, the authors establish an algorithm called Alternating Compressed/Decompressed (AC/DC) Training. This algorithm trains a dense model and a sparse version of it at the same time. The algorithm iterates over the following steps: 1. Train the dense model for a number of epochs, 2. Remove a certain percentage of the weights using magnitude pruning to get a sparse model, 3. Train the sparse model for a number of epochs, and 4. Add the pruned weights back to get a dense model, and go to step 1.

DeepHoyer [Yang *and others* (2019)]: This paper shows that applying Hoyer-Squared (HS) regularization, defined as $\frac{(\sum_i |x_i|)^2}{\sum_i x_i^2}$, can effectively induce state-of-the-art unstructured weight sparsity. Additionally, this paper also introduces Group Hoyer-Squared (Group-HS), which performs a similar regularization but in a group level and is useful for structured pruning.

1.2 *Activation Sparsity*

Assume that all activation functions in the model are ReLU. The fact that ReLU zeros out all the negative values suggests that around 50% of the activations may be zero. In practice, activation sparsity is usually even higher than 50%. Additionally, experiments show that the further we go in the layers, the sparser the activations of the layer will be. Particularly, the activations of last layers of a network are often 80-90% sparse

Hoyer regularization and FatReLU [Kurtz *and others* (2020)]: This paper shows that applying Hoyer regularization on activations leads to a higher activation sparsity, while preserving the accuracy. The authors also introduce an activation function called $FatReLU_T(x) = \begin{cases} 0 : x \leqslant T \\ 1 : x > T \end{cases}$, and show that replacing common ReLU activation functions with FatReLU with proper T values can further boost the activation sparsity. They also present an algorithm to exploit the induced activation sparsity to gain inference speed-ups.

2. Correlation between weight and activation sparsity

A question that might come to one's mind is that whether or not high weight sparsity will impose high activation sparsity. Our experiments in this section show that there is no significant relationship between the two.

To investigate this matter, we take WoodFisher and AC/DC ResNet50 models trained on

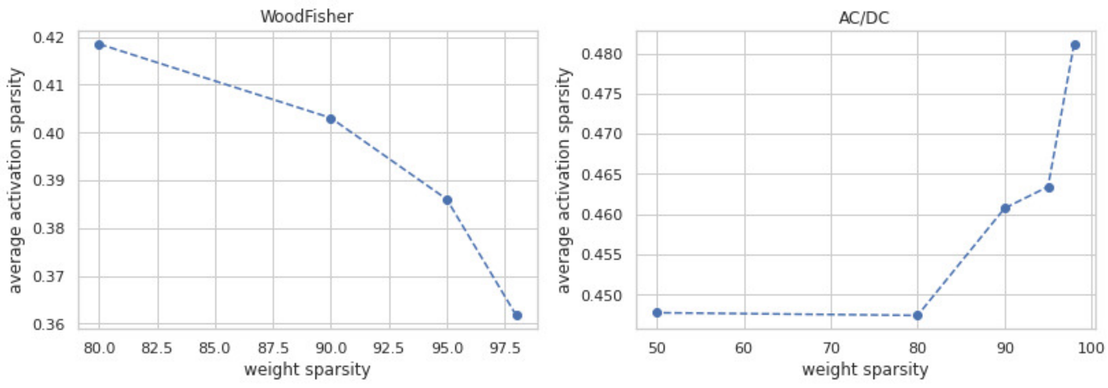

Fig. 1. Average activation sparsity for different weight sparsities in WoodFisher and AC/DC. The models have Resnet50 architectures and are trained on the ImageNet dataset.

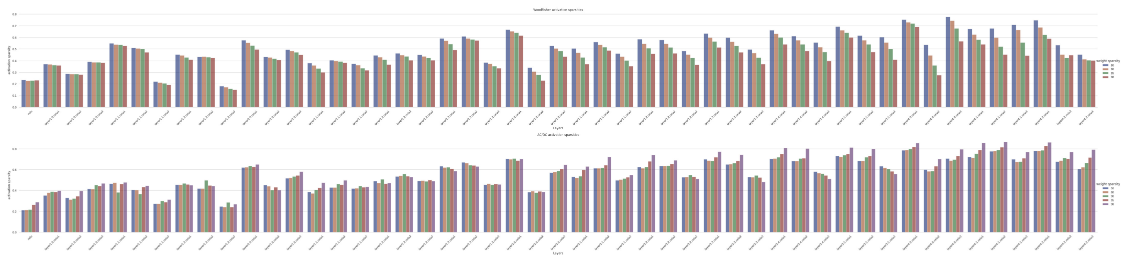

Fig. 2. Layer-wise activation sparsity for different weight sparsities in WoodFisher and AC/DC. The models have Resnet50 architectures and are trained on the ImageNet dataset.

ImageNet and see how sparse the activations are. Figure 1 shows the average activation sparsity for different weight sparsity levels of both methods. The layer-wise activation sparsities of the same models are also available in Figure 2. In the case of AC/DC, it seems that as the weights become more sparse, the activation sparsity increases as well. However, this increase is not significant. For example, the gap in activation sparsity between the AC/DC models with 50% and 98% weight sparsity is around 4%. On the other hand, WoodFisher shows the opposite behaviour, meaning that when the weight sparsity grows, the activation sparsity decreases. These results suggest that there may not be a clear correlation between weight and activation sparsity.

One additional interesting observation about AC/DC is that in Figure 1, there is almost no difference in activation sparsity between 50% and 80% weight sparsity models, but the activation

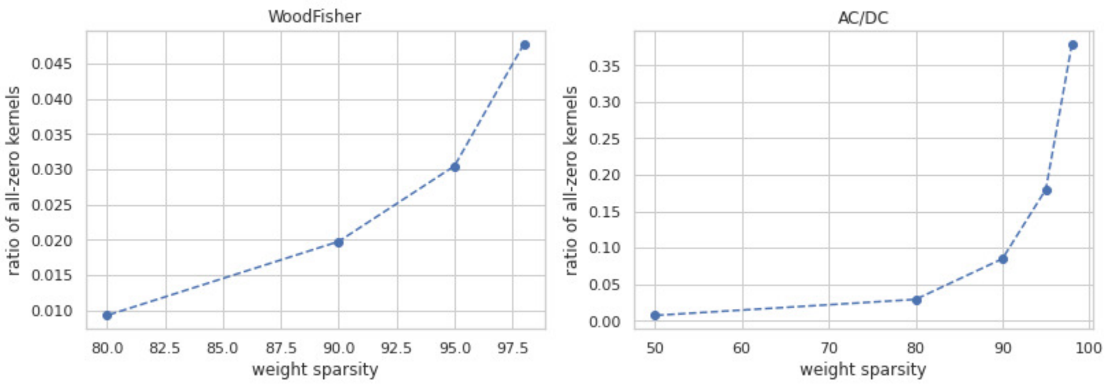

Fig. 3. Percentage of all-zero filters in WoodFisher and AC/DC with different weight sparsities. The models have Resnet50 architectures and are trained on the ImageNet dataset.

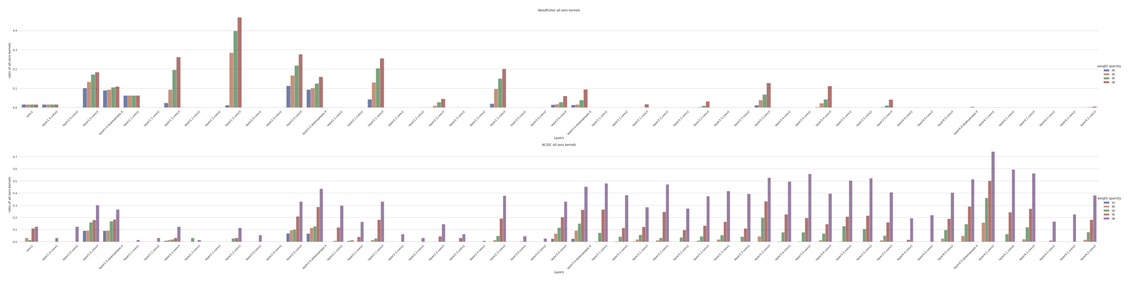

Fig. 4. Layer-wise percentage of all-zero filters in WoodFisher and AC/DC with different weight sparsities. The models have Resnet50 architectures and are trained on the ImageNet dataset.

sparsity quickly increases in weight sparsities above 80%. This suggests that there might be some filters in the convolution layers that are completely pruned, which results in full-zero channels in the next layers' activations. To investigate this matter, we compute the percentage of all-zero filters for both WoodFisher and AC/DC in different weight sparsity levels. The average and layer-wise results are available in the Figures 3 and 4, respectively. Interestingly, while WoodFisher is not completely zeroing out many filters, AC/DC zeros out around 38% of the filters in 98% weight sparsity. It is notable that if 98% of the weights were pruned uniformly random, the expected percentage of all-zero filters would be around 2%. This means, as the authors have mentioned [Peste *and others* (2021)], AC/DC induces structured weight pruning.

Method	Weight Sparsity (%)	Accuracy (%)	Activation Sparsity (%)
Baseline	0	76.13	55
WoodFisher	80	76.63	52
	95	72.03	46
AC/DC	80	76.24	55
	95	73.15	57
Hoyer on Both	80	74.62	60
	95	66.34	58
Hoyer on Activations[1]	∼0	∼76	61

Table 1. Applying Hoyer regularization on both weights and activations. [1] The results of this method are taken from the original paper. They have not reported weight sparsity and accuracy explicitly. But since no weight pruning was done, weight sparsity is close to zero. They have also mentioned that accuracy loss is negligible.

3. Inducing both activation and weight sparsity

3.1 *Hoyer regularization on both weights and activations*

In Section 1, we mentioned that Hoyer regularization is applied by [Yang *and others* (2019)] on weights and by [Kurtz *and others* (2020)] on activations. Here we try to apply it on both weights and activations to see whether we can highly sparsify both. To do so, we added hoyer regularization on weights to the existing code from [Kurtz *and others* (2020)]. The results of this approach in 80% and 95% weight sparsity and a comparison with related methods is available in Table 1. Compared to other methods, this approach generally suffers from more accuracy loss. In 80% weight sparsity we are able to achieve roughly the same activation sparsity as [Kurtz *and others* (2020)] (60%), but in 95% weight sparsity, the activation sparsity drops (58%). Overally, this approach sacrifices too much accuracy, especially when weight sparsity is high.

3.2 *Boosting activation sparsity using FatReLU*

Given a model that is sparse in weights, one might consider boosting the activation sparsity using the FatReLU activation function introduced in [Kurtz *and others* (2020)]. To investigate this, we replaced the activation functions to FatReLU in 90% sparse ResNet50 WoodFisher and AC/DC models trained on ImageNet, and then fine-tuned them. The experiments show that while in WoodFisher activation sparsity does not increase much (<60%), in AC/DC the activation is boosted up to 71%. It's notable that there is no significant loss in accuracy of the model (∼1%).

To further investigate how the activation sparsity can be boosted in AC/DC, we trained a 98% weight sparse model on MNIST, and used the same technique to boost its activation sparsity. Surprisingly, activation sparsity was increased from 43.5% to 91.5%. If we replace AC/DC with the pruning method in [Yang *and others* (2019)], the activation sparsity only goes as high as 71.3%.

4. Leveraging sparsity in both weights and activations

In the previous section, we were able to achieve 90% weight and around 71% activation sparsity. In this section, we will try to exploit this high sparsity on both weights and activations to gain inference speed-ups.

We used gpu programming (cuda) to implement a sparse-sparse version of 2D convolutional layer, and compared its inference time with pytorch's Conv2d module and SparseRT [Wang (2020)], which only exploits weight sparsity. The code for our method is accessible here. It's notable that it's not completely optimized, and further optimizations can lead to higher speed-ups.

In all the experiments, there are 90% weight and 70% activation sparsity. On 1x1 convolutions, our method gains about 3x speed-up with respect to pytorch, while SparseRT reports 3.4x speed-up.

References

Kurtz, Mark, Kopinsky, Justin, Gelashvili, Rati, Matveev, Alexander, Carr, John, Goin, Michael, Leiserson, William, Moore, Sage, Shavit, Nir and Alistarh, Dan. (2020). Inducing and exploiting activation sparsity for fast inference on deep neural networks. In: *International Conference on Machine Learning*. PMLR. pp. 5533–5543.

Peste, Alexandra, Iofinova, Eugenia, Vladu, Adrian and Alistarh, Dan. (2021). Ac/dc: Alternating compressed/decompressed training of deep neural networks. *arXiv preprint arXiv:2106.12379*.

Singh, Sidak Pal and Alistarh, Dan. (2020). Woodfisher: Efficient second-order approximation for neural network compression. *arXiv preprint arXiv:2004.14340*.

Wang, Ziheng. (2020). Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. *arXiv preprint arXiv:2008.11849*.

Woodbury, Max A. (1950). *Inverting modified matrices*. Statistical Research Group.

Yang, Huanrui, Wen, Wei and Li, Hai. (2019). Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*.