Evaluation of Gradient-Based Magnitude Pruning Technique

Ridwan Muheeb
Department of Computer Science
University of Cambridge
rm2084@cam.ac.uk

Abstract

Recently, there have been growing demands to deploy machine learning models in resource-constrained environments. This has sparked an emerging area of research that seeks to reduce training times and compress deep learning models in order to make them applicable to the aforementioned scenarios. In an attempt to understand the support available for neural network pruning in open-source deep learning frameworks, this project evaluates magnitude-based pruning in PyTorch machine learning library.

Introduction

Neural networks have brought about significant progress in areas such as image recognition, natural language processing and speech recognition. However, current neural networks contain massive number of parameters and consume a lot of resources during training and inference. The over-parameterization of their architectures have continued to prove a hinderance of their adoption on resource-constrained devices particularly those with low memory and power consumption. To address this challenge, pruning has been identified as an effective technique to improve their efficiency and increase inference time on devices with less computational budget. While many pruning approaches have been developed, this project evaluates magnitude-based pruning technique available in the open-source and popular PyTorch machine learning framework.

According to Han et al (2015), magnitude based pruning attaches *importance* to every elements in the model and considers weights with larger absolute values are more important than other weights and be either in L1 or L2 norm. The L1-norm is mathematically presented below and depicted in figure 1:

$$Importance = \sum_{i \in S} | w_i |,$$

where $\mathbf{W^{(S)}}$ is the structural set S of parameters W.

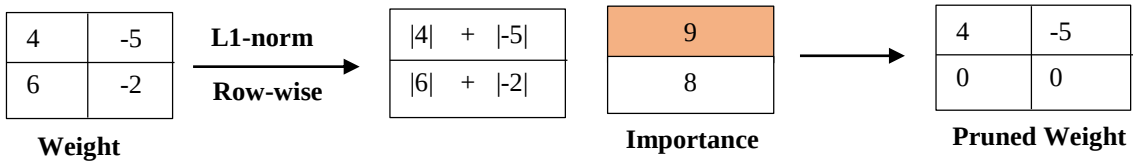

Figure 1: L1-Magnitude Based Pruning

Implementation

The pruning technique implemented in this project was carried out using PyTorch L1-unstructured magnitude based pruning. The models were first trained, then weights pruned iteratively and accuracy measured on the testing dataset. Because the models considered were fully connected convolutional neural networks, the pruning occur at every layer with the exception of the last fully connect layer which contain the output features. The pipeline and pseudo-code are presented in figures 2 and 3 respectively. While the core ideas behind this project were obtained from Han et al (2015), which used a 3-stage pipeline but re-trained the pruned weights until a pruned model with the highest accuracy is achieved. I chose to adopt a different approach, following the observations made by Liu et al (2018) that finds little benefits are accruable from re-training finetuned pruned models because "the prune architecture itself is more important to the efficiency of the final model."

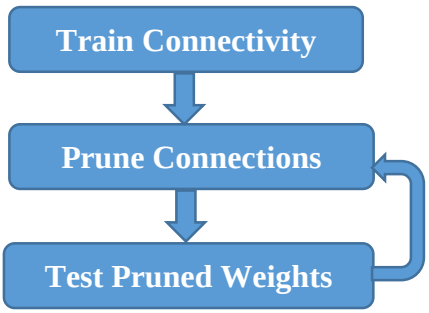

Figure 2: Iterative Pruning Pipeline

1) Perform Weight Pruning to prune away k% neurons/units for *k* in *[0, 25, 50, 60, 70, 80, 90, 95, 97, and 99]*
2) Choose all layers other than the output layer and rank the weights using the absolute values *by using the l1_unstructured function.*
3) Calculate Sparsity in layers and the Accuracy of the Pruned model on the Test set.
4) Save pruned model with the best accuracy

Figure 3: Iterative Pruning Pseudocode

Experiments

As mentioned earlier, the experiments were carried out using PyTorch machine learning library. Pruning technique in this library works by creating a copy of the parameters and also a buffer that stores the pruning mask. It then create a module-level callback that applies the pruning mask to the original weight. The module *torch.nn.utils.prune.l1_unstructured* function in the library used for the evaluation, accepts as inputs the layer name, amount and importance score.

Four convolutional networks were assessed: LeNet-5 and LeNet-300-100 on MNIST dataset and ResNet-18 and ResNet-50 on CIFAR10 dataset.

LeNet on MNIST

The first experiments were carried out on the MNIST dataset using Lecun et al's (1998) LeNet-5 and Lenet-300-100 neural networks. The original architectures of the neural networks were trained – LeNet-5 consists of two convolutional and fully connected layers and achieved an accuracy of 97.24% with 20 epochs while Lenet-300-100 consists of a fully connected network with hidden layers and achieved 97.91% with the same epochs. Then iteratively pruned according to the percentage presented in figure 3. The results of the experiments are presented in table 1.

ResNet on CIFAR10

Both ResNet-18 and ResNet-50 were also trained on CIFAR-10 using the original architectures provided in He et al (2016). ResNet-18 consists of 72-layer architecture with 18 deep layers and achieved 0.89 accuracy under 200 epochs while ResNet-50, a variant of ResNet model class had a 48 convolution layers with 1 MaxPool and 1 Average Pool Layer and achieved same 0.89 also under 200 epochs. Both trained models were also iteratively pruned and accuracy measured on the test dataset. The results is presented in table 2.

Discussion

As shown in Table 1, LeNet-5 was able to withstand pruning ratio up to $\leq$ 80% with high accuracy while LeNet-300-100 able to provide accuracy up to 81% under $\leq$ 90% pruning ratio. Both models showed significantly low accuracy for pruning ratio $\geq$ 95%.

Unlike LeNet neural networks, L1-magnitude based pruning performed poorly on both ResNet-18 and ResNet-5 models as shown in Table 2. Both models could withstood pruning ratio up to $\leq$25% and maintained similar accuracies of 10% for ratio $\geq$ 60%.

The result observed could be attributed to the fact that unstructured pruning significantly changes the weight distribution across each layer.

Although notable experiments have been done particularly the work of Han et al (2015) and Liu et al (2018), there is still a wide gap between research and real-world applicability. The results of the evaluation shows effective pruning is yet to be fully integrated in mainstream and open-source machine learning frameworks. As a matter of fact, I noticed the size of the model increased after pruning. This can be attributed to the fact that PyTorch offers sparsed pruning where invalid connections turns are zeroed and yet still participate in forward propagation and resulting tensors containing significant amount of 0s which unfortunately leads to the increase in model size despite the high compression rate observed. Also, the lack of support for converting the sparse tensors in the pruned network into dense neural network also made it difficult to truly quantify the inference time of the pruned network.

In addition to that, the creation of the parameter copies by the library in order to separate unpruned parameters from the original ones, though makes possible to access the original parameters which allows experimentation with various pruning techniques but comes at the price of performance and memory overhead because of the space consumed by the copies.

Table 1: LeNet-5 and LeNet-300-100 on MNIST L1-Magnitude Based Pruning Result

Dataset	Model	Unpruned	Prune Ratio	Accuracy
MNIST	LeNet-5	97.94	25%	97.96
			50%	97.26
			60%	95.76
			70%	93.69
			80%	90.00
			90%	51.22
			95%	22.62
			97%	9.59
			99%	10.28
MNIST	LeNet-300-100	97.91	25%	97.93
			50%	97.90
			60%	97.95
			70%	97.40
			80%	96.44
			90%	81.25
			95%	37.84
			97%	14.83
			99%	9.75

Table 2: ResNet-18 and ResNet-50 L1-Magnitude Based Pruning Result

Dataset	Model	Unpruned	Prune Ratio	Accuracy
CIFAR-10	ResNet-18	89.36	25%	88.59
			50%	17.39
			60%	9.02
			70%	10.00
			80%	10.00
			90%	10.00
			95%	10.00
			97%	10.00
			99%	10.00
CIFAR-10	ResNet-50	89.90	25%	89.31
			50%	19.42
			60%	10.00
			70%	10.00
			80%	10.00
			90%	10.00
			95%	10.00
			97%	10.00
			99%	10.00

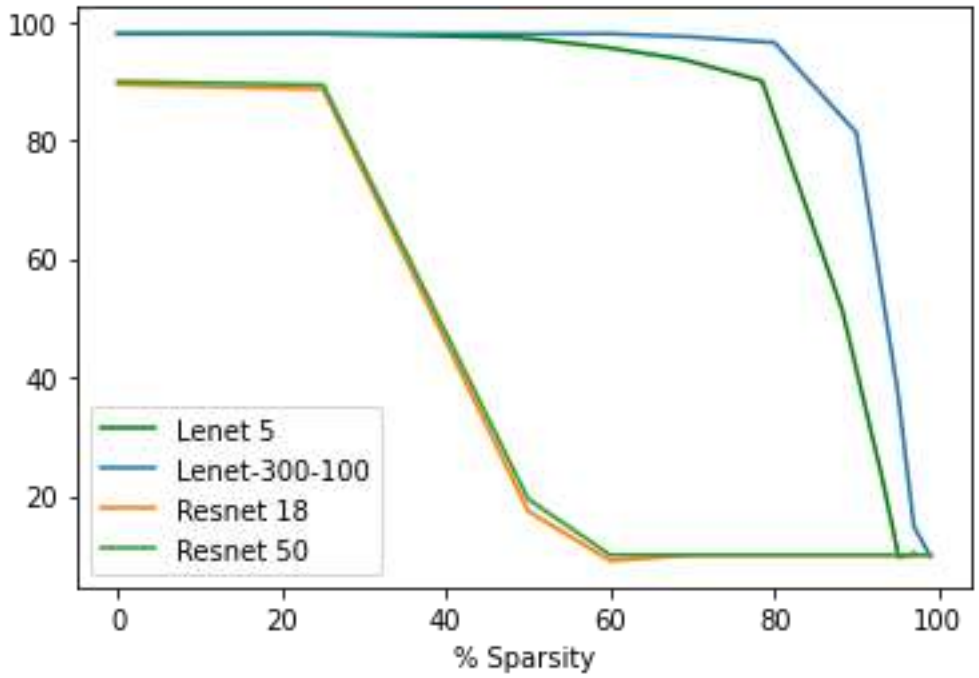

Figure 4: Accuracy Plot for L1-Magnituded Based Pruning

Conclusion

This work examined magnitude based pruning technique available in PyTorch open-source machine learning library. The results showed there is still significant pruning implementation gap in real-world setting. This is because the real-world implementation of pruning through the framework studied showed pruning comes significant memory overhead constraint.

References

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, *28*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

Lei, M. (2021). PyTorch Pruning. Retrieved from https://leimao.github.io/blog/PyTorch-Pruning/ on 15th January, 2023.

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Michela, P. Pruning Tutorial. Retrieved from https://pytorch.org/tutorials/intermediate/pruning_tutorial.html on 15th January, 2023.

Paul, G. (2022). How to Prune Neural Networks with PyTorch. https://towardsdatascience.com/how-to-prune-neural-networks-with-pytorch-ebef60316b91 *Medium.* October, 12, 2022.