Hardware Aware Sparsity Generation for Convolutional Neural Networks

Xiaofan Xu
Edge AI, Intel
xu.xiaofan@intel.com

Zsolt Biro
Edge AI, Intel
zsolt.biro@intel.com

Cormac Brick
Edge AI, Intel
cormac.brick@intel.com

Abstract

Weight sparsity has gained a lot of attention as a compression method in recent years with the increasing demand of hardware containing sparse accelerators. Among all the methods of weight sparsity, AutoML leads to the best performance in terms of compression ratio and accuracy. However, the compression ratio cannot guarantee such network to perform the best on target hardware. Therefore, in this work we introduce hardware aware method of weight sparsity targeting optimal performance on the given hardware. Using our method, ResNets and MobileNetv2 can be reduced to 45% and 70% of original cycles respectively with almost no Top 1 accuracy loss which the other weight sparsity methods cannot achieve. Our work also bring insights of sparse weight distribution on hardware which contains sparse accelerators.

1 Introduction

With the increasing size of neural networks in pursuit of state-of-the-art accuracy, it becomes increasingly harder to accelerate the processing on embedded systems with limited memory bandwidth and computational power. To address this problem, different network compression methods have been introduced; among these compression methods, exploiting sparsity is one of the most valuable techniques. Furthermore, sparsity is getting more appealing due to the release of new hardware accelerators that are finally able to take advantage of it.

Unlike the existing rule-based methods for sparsity, in this work we use an AutoML technique to identify the sparsity ratios for each layer in a network that can bring the maximum acceleration on the given hardware while maintaining state-of-the-art accuracy. This work is a key enabler for the hardware containing sparsity accelerators like Intel Movidius Keembay-vpu2, Tensilica IP [1], Samsung sparsity-aware Neural Processing Unit(10), Nvidia A100[2], etc. We propose a new state-of-the-art method for hardware aware sparsity which can be applied to any neural network and is guided by the hardware architecture. Our method uses a Reinforcement Learning (RL) agent and can quickly identify the best sparsity ratios for each layer on a given hardware. Furthermore, the network generated by the RL agent can achieve high performance on the given hardware with almost the same accuracy as the baseline model. It can also apply to any given networks without any modification of the network architecture. Moreover, this work can be run on both CPUs and GPUs with only a subset of the dataset which makes the processes fast and efficient.

2 Related Work

While a lot of existing works use human-designed rule-based compression methods (1)(3)(11), it is challenging to come up with an optimal strategy. This is because the sparsity ratio for each layer is in

[1] https://ip.cadence.com/ai

[2] https://www.nvidia.com/en-us/data-center/a100/

6th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC2 2020)

a high dimensional space. This makes it extremely unlikely for humans to identify the correct sparsity combinations that give the best accuracy. Moreover, the rule-based methods cannot generalize for all the existing networks. To be more specific, some rule-based methods(2) (4)try to sparsify the parameters in the early layers less (where useful information of the low level features are) and sparsify final fully connected layers more (which contain more parameters). However, these rules do not consider the dependency between the layers in the model and cannot easily transfer from one architecture to another. Therefore,(6) propose to use an AutoML method for model compression which is based on Reinforcement Learning (RL) agent. In order to improve the quality of the network compression this method automatically identifies the sparsity ratios for each layer.

However, none of the existing methods take into consideration the hardware performance during the sparsity generation. The key targets for sparsity methods are compression and speed-up for sparsity-aware hardware. We observed that the models generated from (6) have large overall sparsity but do not perform well (i.e. low speed-up) on given hardware. This is because a network with large overall sparsity could be sub-optimal on a particular hardware platform due to other architectural factors. Therefore, it is essential to include knowledge of the hardware architecture into the AutoML method. This allows to obtain the best sparsity ratios for each layer while maintaining state-of-the-art accuracy.

3 Methodology

This work aims to automatically sparsify the neural networks with hardware knowledge during the pruning stage. Fig.1 shows the overview of the RL agent training process which is similar to the AMC process (6) . However, in our work we added the hardware aware feature into the training pipeline of the agent. This is mainly because (6) generates large overall sparisity that does not necessarily result in a reduced number of computational cycles. The number of hardware cycles are generated by running through a hardware simulator. Using this work, any sparsity scheme can be generated starting from any hardware architecture.

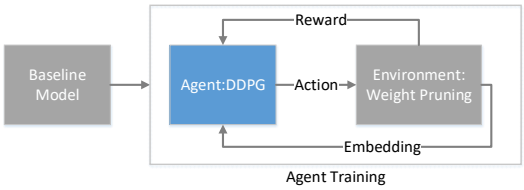


Figure 1: Overview diagram of RL agent training procedure.

3.1 DDPG Agent Training

The RL agent takes in a baseline model as input. During the training, the RL agent is learning to predict the action (i.e. the percentage of cycle reduction) that gives the sparsity to each layer and perform the pruning for the layer which helps to reduce the computation cycles on the particular hardware. In our case, a Deep Deterministic Policy Gradient (DDPG) (8) agent is used to learn the best combination of sparsity ratios for the layers of the network. Once every layer in the whole network has a sparsity level assigned, a quick evaluation is performed. The evaluation result (also known as the reward) is fed back to the DDPG agent in order to update the next prediction (i.e. next action for each layer). This process is repeated several times for the DDPG agent to learn a sparse and accurate model that has the best performance on the given hardware. Furthermore, since only forward pass is required to evaluate the sparse model to generate the reward, the DDPG agent training process is fast and efficient and can perform either on CPU or GPU. Moreover, during the DDPG agent training, we only use a subset of the training dataset to perform the evaluation to further save training time.
A detailed depiction of the RL agent training process is shown in Fig.2. DDPG (8) is an off-policy actor-critic algorithm. The agent receives the embedding state Eq. 1 for layer t from the environment and outputs the action for this layer t as the percentage of reduction in cycles for this layer. To be more specific, as the computational cycles are related to the sparsity of the layer t therefore it

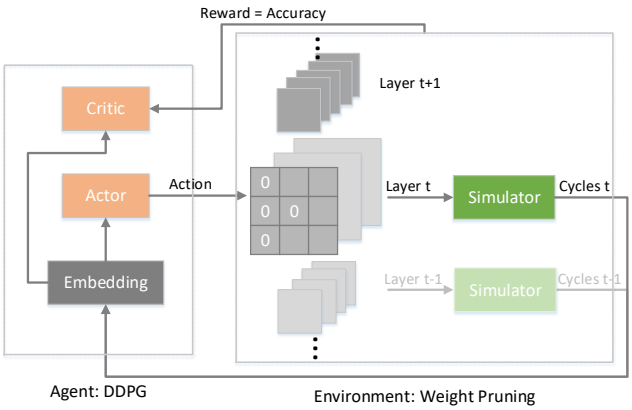


Figure 2: Reinforcement learning agent training in details.

generates sparsity for the layer t. Then it moves to the next layer $t+1$ and the agent receives the embedding for $t+1$. After all the layers in the network have passed through this process, a reward is evaluated on the subset of the validation dataset and returned to the agent. Based on the reward, the agent is learning to predict the best combination of sparsity ratios that generates the optimal cycles on the given hardware during the agent training.

Embedding state: For each layer t, it has features that characterize the layer to be the state into the DDPG agent. Eq. 1 below represents an example of the embedding state we use in the work. However this equation can be varied based on the performance. Extra terms can be easily added into the embedding state to capture the statistics of the weight tensor i.e Hessian of the weight matrix.

$$(t, n, c, h, w, stride, k, weight[t], cycles[t], reduced, rest, a[t-1], extra[t]) \tag{1}$$

where t is the layer index, the kernel size for the layer is $n * c * k * k$ and input feature is $c * h * w$, $weight[t]$ is the number of parameters in the layer t and $cycles[t]$ is the number of cycles required for this layer t on a particular hardware. $reduced$ is the total number of reduced cycles in previous layers and $rest$ is the number of remaining weights in the following layers. $a[t-1]$ is the action for the previous layer. $extra[t]$ can be None or any other characteristics for the particular layer t, it can be used to capture the statistics of the weight matrix by using Hessian of the matrix.

Action space: We use continuous action space i.e. a $\in (0, 1]$ for the reduction of cycles for each layer. Continuous action space enables a larger search space as well as less accuracy loss. We limit the action space for the network by setting a target percentage reduction for the number of cycles on the particular hardware. Therefore, the DDPG agent learns to prune the weights in each layer in order to find the best combination of cycle reduction for each layer. If during the training stage, we find the action taken cannot reach our target percentage, then the agent will aggressively prune all the remaining layers in order to reach the target cycle reduction. Moreover, we can reach the target percentage reduction while achieving the best reward by using the reward function. This reward is based on the subset of the training accuracy for the sparse model.

3.2 Fine-Tuning

Once all the actions have been performed on the baseline model, we obtain a best reward sparse model which gives the target reduction in cycles on the particular hardware. Further fine-tuning may be applied onto the sparse model to increase the accuracy a bit more for the final sparse model. To be more specific, the sparse model from the DDPG agent may have Top 1 accuracy loss from the baseline model, but a fine-tuning stage can help the sparse model to attain the baseline accuracy. Moreover, the fine-tuning requires fewer than $1/10$ the epochs for the original baseline training epochs. To be more specific, for Mobilenetv2 original training is 120 epochs, the fine-tuning epochs only need to be 10 epochs.

4 Experimental Results

We performed several experiments in order to compare our work with the existing methods. The number of hardware cycles is generated by the internal hardware simulation toolkit. The hardware architecture we targeted throughout the experiments contains sparse accelerator.

In order to have a fair comparison between our method and existing sparsity methods, we used RL to generate sparsity for both our hardware aware method and a non-hardware aware method. This RL based method gives the state-of-the-art performance in terms of number of parameters and accuracy for non hardware aware scenarios(6). The duration of the RL agent training and fine-tuning epochs are the same for both the hardware aware method and the non hardware aware method. The experiments are performed on 3 different networks ResNet18, ResNet50 and MobileNetv2(5)(9). The dataset used for all the results below is based on ImageNet(7). For each model, we have performed both a hardware aware method (this work) and a non hardware aware method(6). Baseline models are taken from the Pytorch repository.

Table 1: ResNets analysis based on hardware aware method and non-hardware aware method.

	Top 1 Accuracy (%)	Top 5 Accuracy (%)	drop @top1	No. Parameters	No. of zeros	%sparsity	Cycles % (the lower the better)
ResNet18	69.758	89.078	0	11.69M	0	0%	100.00%
HW Aware	69.782	89.172	0.024	4.0M	7.7M	65.87%	49.54%
HW Aware	69.596	89.052	-0.162	2.9M	8.8M	75.27%	**44.78%**
Non HW Aware	70.24	89.596	0.482	4.2M	7.5M	63.90%	77.14%
Non HW Aware	69.758	89.288	0	2.9M	8.8M	74.94%	67.67%
ResNet50	76.13	92.862	0	25.5M	0	0.00%	100%
HW Aware	75.564	92.614	-0.566	8.9M	16.6M	65.12%	**45.01%**
Non HW Aware	76.104	92.932	-0.026	7.6M	17.9M	70.23%	61.79%

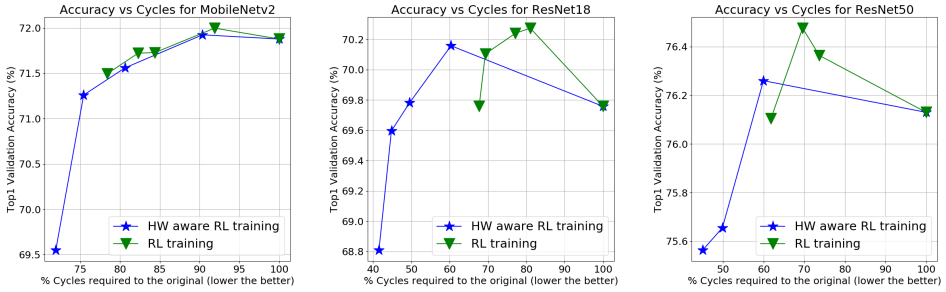
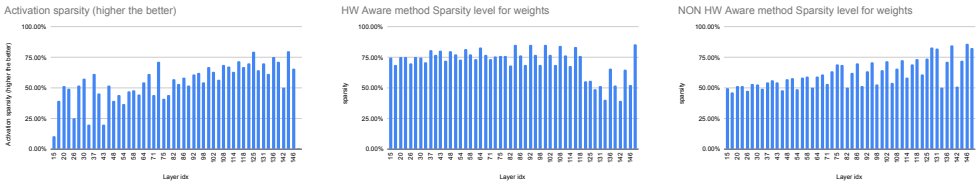


Figure 3: Accuracy vs Cycles for ResNet18, ResNet50, MobileNetv2 for both HW aware method (shown in blue line) and non HW aware method (shown in green line).

Table 1 shows the detailed results on ResNets using both our hardware aware method and the non hardware aware method. With the same sparsity level and accuracy range, the hardware aware method can reduce the cycles much more than the non hardware aware method. Therefore, it proves that high compression ratio dose not guarantee better performance on given hardware. Hardware aware method is the key enabler for reducing the computation cycles.

Another key feature that affects the performance on the target hardware is activation sparsity. But activation sparsity depends on the input data which is dynamic so it cannot be pre-generated and stored. Therefore, it doesn't mean there is linear relationship between sparsity of the weights and performance on the hardware, the correct combination between the weight sparsity for each layer and input activation sparsity for this particular layer is the key to increase the performance on the target hardware (i.e. reduce the cycle counts on the hardware). Furthermore, for certain hardware the increase in performance is different between the types of layers as well. Non hardware aware methods cannot trigger these during pruning. This is extremely important for compact models like mobileNetv2. Fig.3 clearly shows that our hardware aware method (shown in blue) reduces the cycles on the particular hardware on all the networks. To be more specific, each data point on the graph requires fine-tuning for iterative pruning, with the same amount of training time, hardware aware method outperforms in cycle counts. The non hardware aware method helps to reduce a lot of weights (shown in sparsity column in Table 1 for ResNets) but due to the configuration of the hardware it cannot perform as well as our method in terms of cycle count. The experiments prove this work works for all these networks.

Moreover, our method takes activation sparsity and layer types into consideration when generating the sparsity for weights in order to maximize the performance on the hardware. Therefore, it caused



(a) (b) (c)

Figure 4: ResNet50 activation sparsity (a); ResNet50 weight sparsity distribution for HW aware method (b) and Non HW aware method (c) .

some degradation in accuracy when reducing large amounts of computation cycles. This is due to the limitation in pruning the weights for the networks which Fig.4 explains in detail by showing the distribution of weight sparsity and activation sparsity for ResNet50. In the normal RL method the goal is to prune as much weights as possible while maintaining accuracy. Therefore, it tries to prune more weights in the later layers than in earlier layers. However, the earlier layers are generally more important to the accuracy of the model. Fig.4(c) for ResNet50 has shown the RL agent prunes more than 75% in the last few layers while only pruning less than 50% in the early layers. However, our work takes activation sparsity into account when considering the hardware efficiency, the activation sparsity for the later layers in ResNet50 are much higher than for the earlier layers (this activation sparsity is calculated based on few batches of the ImageNet training set). To be more specific for the hardware, once activation sparsity reaches a certain level, the weight sparsity cannot introduce as much speed up, therefore, the later layers do not need to be pruned as much while in contrast the earlier layers need to be pruned more to have better performance. Fig.4(b) proves that the earlier layers are almost at 75% sparsity while the later layers only contain less than 50% sparsity. Despite these factors, our work still outperforms the normal RL method at the same computation cycles.

5 Conclusion

In this work, we have shown that our hardware aware method for weight sparsity can reduce the cycles for ResNets and MobileNetv2 to 45% and 70% of original cycles respectively with almost no Top 1 accuracy loss on the target hardware which contains sparse accelerator. We have proven that large overall sparsity that does not necessarily result in a reduced number of cycles on given hardware. Using our method, we can achieve the optimal networks on the given hardware.

References

[1] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pages 1379–1387, 2016.

[2] S. Han and B. Dally. Efficient methods and hardware for deep learning. *University Lecture*, 2017.

[3] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[4] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[10] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang. 7.1 an 11.5 tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 130–132. IEEE, 2019.

[11] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.