DYNAMIC PROBABILISTIC PRUNING: TRAINING SPARSE NETWORKS BASED ON STOCHASTIC AND DYNAMIC MASKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Learning (DL) models are known to be heavily over-parametrized, resulting in a large memory footprint and power consumption. This hampers the use of such models in hardware-constrained edge technologies such as wearables and mobile devices. Model compression during training can be achieved by promoting sparse network structures both through weight regularization and by leveraging dynamic pruning methods. State-of-the-art pruning methods are however mostly magnitude-based which impedes their use in e.g. binary settings. Importantly, most of the pruning methods do not provide a structural sparsity, resulting in inefficient memory allocation and access for hardware implementations. In this paper, we propose a novel dynamic pruning solution that we term Dynamic Probabilistic Pruning (DPP). DPP leverages Gumbel top-K sampling to select subsets of weights during training, which enables exploring which weights are most relevant. Our approach allows for setting an explicit per-neuron layer-wise sparsity level and structural pruning across weights and feature maps, without relying on weight magnitude heuristics. Relevantly, our method generates a hardware-oriented structural sparsity for fully-connected and convolutional layers that facilitates memory allocation and access, in contrast with conventional unstructured pruning. We show that DPP achieves competitive sparsity levels and classification accuracy on MNIST and CIFAR-10, CIFAR-100 datasets compared to a state-of-the-art baseline for various DL architectures while respecting per-neuron sparsity constraints.

1 INTRODUCTION

The evident success of Deep Learning (DL) models is accompanied by a steadfast growth in the number of hyperparameters and computational cost. This has become a bottleneck for hardware deployment, which is constrained to certain computational and memory budget. For instance, VGG-16 architecture with 14.7M weights occupies more than 500 MB of storage and performs 1.6 x 10^{10} floating-point arithmetic operations (Cheng et al., 2019), (Sze et al., 2017), contrasting FPGA-based platforms which are constrained to a few thousand computing operations, unsuitable for large DL model implementations. These constraints have hindered the deployment of DL models in embedded and portable systems. To circumvent such issues, different solutions have been proposed such as quantization (Hubara et al., 2016) (and in the extreme case binarization (Courbariaux & Bengio, 2016)), knowledge distillation (Hinton et al., 2015) and weight sharing (Han et al., 2016).

In terms of memory reduction, one attractive solution is model compression by pruning, which has gained notable attention since their performance yields state-of-the-art results of non-pruned counterparts. Pruning methods can be subdivided into structured and unstructured pruning. Structured pruning methods remove structures at the level of e.g. layer, channels, or individual filters (Liu et al., 2019). Unstructured pruning is a process in which a fraction of weights is set to zero. Coincidentally, similar behavior is found in biological neurons, based on the fact that a fraction of biological synapses in the human brain are also removed during human growth (Chechik et al., 1999). Remarkably, pruning has been shown toprevent overfitting as well (LeCun et al., 1990) (Hanson & Pratt, 1989). Further, it was actually demonstrated that DL models possess a high level of redundancy (Denil et al., 2013). This over-parametrization is highly costly in terms of memory and power resources when

implementing such a model in hardware. Generally, structured and unstructured pruning offers a considerable reduction in the number of parameters and model complexity.

However, two disadvantages arise for conventional unstructured sparse weights. The first one is related to the irregular distribution of their resulting sparse matrices that leads to inefficiencies in terms of memory access and allocation in current hardware platforms (Zhang et al., 2015) (Zhang & Li, 2017). This problem may be solved by the use of structured pruning, however structured pruning generally aims to prune channels and filters, rather than individual weights of fully-connected layers (which for certain architectures consumes most of the memory). The second disadvantage of current pruning methodologies is the lack of integration of existing pruning algorithms with other compression techniques such as quantization and its extreme binarization.

Therefore, we propose a framework that naturally generates a structured sparsity for both, fully-connected and convolutional layers. The framework is capable of integrating both pruning and binarization, to benefit from both techniques. The proposed method exploits the notion of Deep Probabilistic Subsampling (DPS) (Huijben et al., 2020a) to dynamically generate stochastic pruning masks, which are independent of the magnitude of the weights, allowing for direct applications to quantized and even binary weight values. The main contributions of this work are the following:

- This framework generates a hardware-oriented sparse structure for fully-connected and convolutional layers (kernel weights and feature maps), which facilitates memory allocation and access.
- We adopt a layer-wise sparsity level that can be selected by the user, which is beneficial in case of hardware constraints that dictate a maximum memory usage.
- The historical importance of weights is maintained, i.e. our framework allows re-wiring during training.
- It allows for pruning of quantized or even binary sparse models, producing ultra compressed-models with low-memory and low-complexity, suitable for further dedicated hardware.
- Leveraging the probabilistic nature of DPP, we propose novel information-theoretic metrics that capture the confidence and diversity of the pruning masks among neurons.

2 Related work

2.1 Pruning techniques

Early work (LeCun et al. (1990); Hassibi et al. (1994)) has shown that pruning is an effective technique to dramatically reduce connections of a neural network, while maintaining state-of-the-art performance. The conventional pruning methodology is based on a three-stage pipeline: full model training, pruning, and fine-tuning. For instance Han et al. (2015) proposed a three-stage pipeline pruning technique, which requires a post fine-tuning of the weights, which further was extended to Deep compression (Han et al., 2016). Nevertheless, the major drawback of this method is the lack of simultaneous co-optimization for both weights and sparse topology. Other remarkable works perform pruning before training has started, identifying structurally important connections (Lee et al.)

Recent works have indeed proposed joint optimization of the network's parameters network and its sparse distribution during the training (dynamic pruning), where this joint learning is based on magnitude-pruning. And Zhu & Gupta (2017) adopts the idea of dynamic pruning based on the gradual increment of sparsity during training, which was further extended to Recurrent Networks by Narang et al. (2017). Bellec et al. (2018) is another magnitude-based pruning framework in which a minimal sparsity value is set, allowing re-wiring during training. More close to a biological behavior, Mocanu et al. (2018) proposes a random remove-and-add-weights approach (SET or Sparse Evolutionary Training) based on a fixed sparsity level. Further Mostafa & Wang (2019) proposes a Dynamic Sparse Reparametrization (DSR) based on loss gradients and sparsity distributed among layers. Dettmers & Zettlemoyer (2019) proposes a sparse momentum by identifying the layers and weights which significantly reduced the error (momentum of each layer). A most recent approach (Liu et al., 2020) provides a method to perform a fine-grain pruning schedule, with filter-wise trainable thresholds based on a piecewise polynomial function. (Lin et al., 2020) proposes a dynamic allocation of the sparsity pattern, incorporating a feedback signal to reactivate prematurely pruned weights. An

approach that differs from magnitude-based methods was proposed by (Molchanov et al., 2017) in the form of Sparse Variational Dropout, a modification of Variational Dropout with individual dropout rates per weight. This leads to highly sparse solutions both in fully connected and convolutional layers.

2.2 HARDWARE-ORIENTED PRUNING TECHNIQUES

As a complement of weight pruning, other works have explored pruning at the architectural level such as pruning channels, filters or layers. This approach offers a better solution to handle sparse arrangements for hardware platforms. In fact, Liu et al. (2019) has experimentally shown that rather than eliminating weight connections, pruning at the architectural level may offer more benefits to reduce memory, while retaining state-of-the-art accuracy. For achieving structured pruning, magnitude-based methods are commonly used, e.g. based on the absolute magnitude of the sum of the weights (Li et al., 2017). Simultaneously, Wen et al. (2016) proposed the Structured Sparsity Learning (SSL) method to regularize filter, channel, and depth structures. Different from the magnitude-based approaches, (Hu et al., 2016) relies on the output of activation layers and calculates an average percentage of zeros as a weighting for the filter relevance. On the other hand, the structured pruning approach has not been transferred to fully-connected layers, which for certain architectures consume the largest percentage of memory. In the case of individual weight pruning, the irregular weight distribution prevents translating pruned weight into memory reduction in hardware implementations. There are already several works addressing this issue at hardware level. These works have proposed compressing coding techniques for sparse weights, however they usually require extra arrays for pointing the coordinates of the non-zero weights (Lu et al., 2019), (Zhang et al., 2019), (Dey et al., 2018) (Niu et al., 2019). Since the sparse distribution is typically irregular, this leads to inefficient memory access.

3 PROPOSED METHODOLOGY

3.1 NOTATION

We introduce a neural network $\mathcal{M}$ with L layers, each indexed with i. Each layer is parametrised by a (possibly multi-dimensional) matrix $\boldsymbol{W}^{(i)}$, and a bias vector, which we ignore in our notation for sake of simplicity. We denote the functionality of each layer with $g^{(i)}_{\boldsymbol{W}}(\cdot)$, where its specific meaning depends on the layer being e.g. fully-connected or convolutional, and linearly or non-linearly activated. The output of the i^{th} layer can then be defined as $\boldsymbol{x}^{(i)} = g^{(i)}_{\boldsymbol{W}}(\boldsymbol{x}^{(i-1)})$, where $\boldsymbol{x}^{(i-1)}$ is its input with size.

3.2 DYNAMIC MASKING BASED ON PROBABILISTIC SUBSAMPLING

We aim to optimize the parameters of network $\mathcal{M}$ while simultaneously learning to prune this model. In this section we explain how this joint learning is achieved.

For all layers $i \in \{1, \ldots, L\}$, we introduce a binary mask $\boldsymbol{M}_{\boldsymbol{\Phi}^{(i)}} \in \{0, 1\}$, parametrised by $\boldsymbol{\Phi}^{(i)}$. By means of element-wise multiplication it activates a subset of the elements in $\boldsymbol{W}^{(i)}$ or $\boldsymbol{x}^{(i)}$, effectively pruning $\mathcal{M}$. Generation of these masks $\boldsymbol{M}_{\boldsymbol{\Phi}^{(i)}}$ follows the DPS-topK framework, recently proposed by Huijben et al. (2020a), on which we will elaborate here.

Huijben et al. (2020a) propose an end-to-end framework for joint learning of a discrete sampling mask with a downstream task model by introducing DPS; a parametrized generative sampling model:

$$P(\boldsymbol{M}_{\boldsymbol{\Phi}^{(i)}} | \boldsymbol{\Phi}^{(i)}). \tag{1}$$

Parameters $\boldsymbol{\Phi}^{(i)} \in \mathbb{R}^{D \times C}$ are defined as the unnormalized log-probabilities of D (≥ 1) independent multinomial distributions with each C classes. We index the rows and columns of $\boldsymbol{\Phi}$ by d, and c respectively, both starting from 1.

We sample one realization from each of the D distributions, where each realization is defined as a K-hot vector. A K-hot vector contains K ones at the selected indices, and $C - K$ zeros at the other positions. Similarly as in Huijben et al. (2020a;b), we adopt Gumbel top-K sampling (Gumbel, 1954;

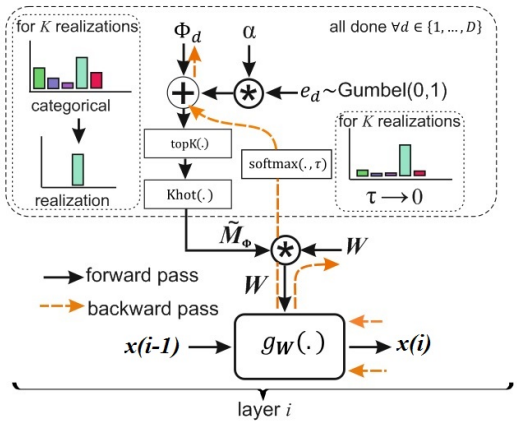

Figure 1: An illustration of the deep probabilistic pruning framework (for layer i) for dynamic masking of weights W with a stochastic mask $\tilde{\boldsymbol{M}}_{\boldsymbol{\Phi}^{(i)}}$. This mask is generated by sampling from a categorical distribution with probabilities that are jointly learnt with the weights W. The $*$ symbol indicates element-wise multiplication.

Kool et al., 2019) to create this K-hot vector realization from each d^{th} distribution, parametrised by $\boldsymbol{\phi}_d^{(i)}$. For each d in D, this can be formulated as follows:

$$\tilde{\mathbf{m}}_{\boldsymbol{\phi}_d^{(i)}} = \mathrm{Khot}\big\{ \underset{c \in \{1 \ldots C\}}{\mathrm{topK}} \; (\boldsymbol{\phi}_d^{(i)} + \alpha \cdot \boldsymbol{e}_d^{(i)}) \big\}, \tag{2}$$

where $\boldsymbol{e}_d^{(i)} \in \mathbb{R}^{1 \times C}$ are i.i.d. Gumbel noise samples from $\mathrm{Gumbel}(0, 1)$, scaled with a scalar $0 < \alpha \leq 1$, and $\mathrm{Khot}(\cdot)$ creates a K-hot encoded row from the returned indices by $\mathrm{topK}(\cdot)$. Scaling of the added Gumbel noise with α was heuristically found to improve optimization during training.

These D realizations together result in a sampling mask $\tilde{\boldsymbol{M}}_{\boldsymbol{\Phi}^{(i)}}$ that only activates $D \times K$ of its inputs. We can now interpret each element in $\boldsymbol{\Phi}^{(i)}$ as the (unnormalized) log-probability of activating a candidate element in $\boldsymbol{W}^{(i)}$ or $\boldsymbol{x}^{(i)}$ by our pruning mask.

During backpropagation the $\mathrm{Khot} \circ \mathrm{topK}$ operation must be relaxed as it is non-differentiable. To this end, we can covert it into $K \arg\max$ operations that together create a K-hot vector by sampling without replacement among each sampling action. Each $\arg\max$ can be relaxed by replacing it with a temperature (τ)-parameterised $\mathrm{softmax}_\tau(\cdot)$ function to allow flow of gradients to $\boldsymbol{\Phi}^{(i)} \forall i \in \{1, \ldots, L\}$ (Maddison et al., 2017; Jang et al., 2016; Xie & Ermon, 2019; Huijben et al., 2020a;b).

Our proposed incorporation of DPS for model pruning will be referred to as Dynamic Probabilistic Pruning (DPP) in the rest of this paper. It is important to emphasize the difference in DPP's behavior during training the model, and implementing it to generate a final pruned model after convergence. During training, the unnormalized log-probabilities $\boldsymbol{\Phi}^{(i)}$ of the multinomial distribution(s) for layer i are constantly being updated, therefore causing different masks to be generated for each element in the mini-batch, allowing the model to explore different combinations of selected candidate elements. During inference, one mask $\tilde{\boldsymbol{M}}_{\boldsymbol{\Phi}^{(i)}}$ per layer will be drawn from the trained log-probabilities $\boldsymbol{\Phi}^{(i)}$ that is then used to prune the model (Fig.1).

3.3 Case-specific pruning based on DPP

Generally speaking, DPP acts as a selector, producing a mask that connects each output node to K non-zero input nodes. In this work, experiments with DPP are performed for three scenarios that differ in the parts of the model that are learned to be pruned:

(a) **Weight pruning of fully-connected layers**: in this scenario, C is the number of neurons in layer $i-1$ and D is the number of neurons in layer i. DPP operates on $C \times D$ weights,

representing the $C \times D$ connections between the fully-connected layers, and learns to select K connections out of C values for each D (Fig.2a).

(b) **Convolutional weight kernel pruning**: for this case, D corresponds to the number of feature maps in layer $i-1$, while C corresponds to the number of elements in the kernels of layer i. In other words, DPP learns to select K weights per kernel to be connected to each feature map of layer $i-1$ (Fig.2b).

(c) **Feature map pruning of convolutional layers**: this particular scenario considers D=1, and C is the number of feature maps of layer i. Then DPP learns to select K feature maps of layer i (Fig.2c). The motivation for this setting is driven by the fact that a higher level of structured pruning (e.g. pruning channels or feature maps) is more suitable for hardware implementation than unstructured pruning as described in Section 2.2

Important to notice is that case (a) and (b) also promote a level of structured pruning by attaining each output node to the same number of input nodes. This contrasts with most of the state of the art of weight pruning, in which an irregular sparse distribution is generated, which is more inconvenient for hardware designs.

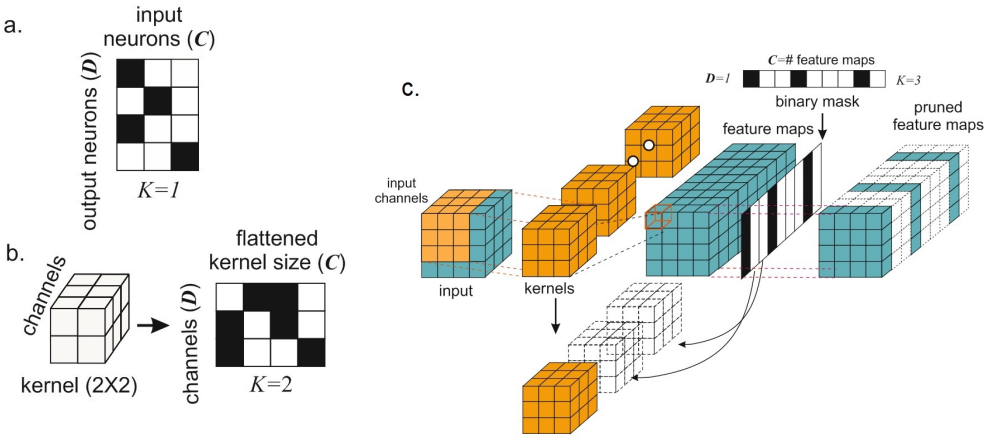

Figure 2: Visualization of the three scenarios for which we add experiments with DPP. All adopted values are illustrative. a. Fully-connected weight pruning (example 3 inputs 4 outputs, $K = 1$), b. kernel weight pruning (example 2x2 kernel and 3 channels, $K = 2$), and c. feature map pruning (example $K = 3$). The three scenarios offer a level of structured pruning, where black squares denote selected connections.

3.4 Information-theoretic metrics on sparsity confidence and diversity

We are interested in the randomness and diversity of the pruning patterns among the neurons in a layer, and how these progress during training. The probabilistic nature of DPP enables the use of the information theoretic measures of entropy and mutual information to evaluate this. We consider the case of per-neuron weight pruning as described in Section 3.3 for case (a) and (b).

We can measure the average entropy from D pruning distributions over c neurons on the i^{th}-layer mask marginal probabilities $\{\boldsymbol{\pi}^{(i)} \in \mathbb{R}^{D\times C} : 0 \leq \pi^{(i)}_{d,n} \leq 1, \sum_n \pi^{(i)}_{d,n} = K\}$. No tractable function exists to compute marginal probabilities $\boldsymbol{\pi}$ from the unnormalized log-probabilities in $\boldsymbol{\Phi}$. Instead we can easily take a Monte Carlo estimate by computing the average of T realizations of the K-hot masks $\boldsymbol{m}^{(i)}_d \in \{0,1\}^{1\times C}$:

$$\boldsymbol{\pi}^{(i)} \approx \frac{1}{T}\sum_{t=1}^{T} \tilde{\boldsymbol{M}}_{\boldsymbol{\Phi}^{(i)}}, \quad \tilde{\boldsymbol{M}}_{\boldsymbol{\Phi}^{(i)}} \sim P(\boldsymbol{M}_{\boldsymbol{\Phi}^{(i)}} | \boldsymbol{\Phi}^{(i)}), \tag{3}$$

which can be effectively estimated in parallel after every epoch for $T = 100$ at a negligible computational penalty. It's trivial to show that the entropy of any Gumbel-top-K distributed variable x can be computed using the typical Shannon entropy $H(x) = -\sum_{i=1}^{C} P(i \in x) \log P(i \in x)$ and is upper bounded by $-K \log(K/C)$. As such, we can compute

$$\text{AveragePruneEntropy}^{(i)} = \frac{1}{D} \sum_{d=1}^{D} [- \sum_{n=1}^{C} \boldsymbol{\pi}_{d,n}^{(i)} \log \boldsymbol{\pi}_{d,n}^{(i)}]. \tag{4}$$

Intuitively, this measures how confident the sparsity patterns are on average for the neurons of layer i. Furthermore, we can measure the diversity of sparsity patterns among the neurons in a layer by measuring the mutual information between the neuron-specific mask random variable $m^{(i)}$ and the output neuron index d, which we call the PruningDiversity metric:

$$\text{PruningDiversity}^{(i)} = I(\boldsymbol{m}^{(i)}, d) = H(\boldsymbol{m}^{(i)}) - H(\boldsymbol{m}^{(i)}|d) \tag{5}$$

$$= \text{EntropyAverageMask}^{(i)} - \text{AveragePruneEntropy}^{(i)} \tag{6}$$

$$\widetilde{\boldsymbol{\pi}}^{(i)} = \frac{1}{D} \sum_{d=1}^{D} \boldsymbol{\pi}_{d}^{(i)}$$

$$\text{EntropyAverageMask}^{(i)} = - \sum_{n=1}^{C} \widetilde{\pi}_{n}^{(i)} \log \widetilde{\pi}_{n}^{(i)}. \tag{7}$$

4 EXPERIMENTS

We first assess DPP for pruning connections between neurons on small convolutional and fully connected architectures for the MNIST dataset (case (a) and (b) of section 3.3). Additionally, we will demonstrate the performance in combination with binary weights (Courbariaux et al., 2015). Across these experiments, we set a K value per layer, which determines the exact number of active inputs assigned to each output neuron. The classification layer is not pruned and thus remains fully connected.

Finally, we test DPP for feature map pruning in deep convolutional networks (case (c) of section 3.3). For this case, we use K as a selector of the number of feature maps that must remain active in each layer, while for fully-connected layers, K selects the number of active inputs assigned to each output neuron.

For all experiments, we use the categorical cross-entropy as loss function, and we penalize the unnormalized log-probabilities with the entropy penalty function used by Huijben et al. (2020a). In addition, we train all our models from scratch, without using any pre-trained model. Finally, the parameter α (scaling factor for Gumbel noise) is heuristically tuned, since the latter technique is empirically observed to improve the performance of our sparse models for values $\alpha < 1$.

4.1 LENET ON MNIST

We evaluate DPP first on MNIST benchmark dataset consisting of a total of 70,000 grayscale images of handwritten digits having a size of 28×28 pixels. We use 60,000 images for training and 10,000 images for testing. We evaluate the performance on 2 architectures. First, we use LeNet 300-100 (Lecun et al., 1998), which consists of two fully-connected layers of 300 and 100 units, respectively. For this experiment, we use case (a) of section 3.3.

Second, we use LeNet-5 Caffe, which consists on two convolutional layers (20 and 50 filters respectively) followed by one fully-connected layer and a classification layer (Lecun et al., 1998). For this experiment, we use case (b) for the convolutional part, and section (a) for the fully-connected stage. We compare our experiments with the recent work of Liu et al. (2020), which proposes a similar approach based on dynamic masking, but with an unfixed sparsity. Results are shown in Table 1. Training was done with a learning rate of 0.001 using the Adam optimizer, and a batch size of 8 for LeNet 300-100 and 16 for LeNet-5. All weights are initialized based on Xavier uniform initialization. In both cases, LeNet300-100 and LeNet5 Caffe, we obtain competitive accuracies in comparison with our baselines, with higher sparsity levels.

Table 1: Experimental results of DPP for MNIST dataset using LeNet architectures

Network	Model	Remaining parameters (%)	Accuracy (%)
LeNet300-100	**DPP** (This work)	1.95	97.90
	(Liu et al., 2020)	2.24	98.03
	Non-pruned baseline	100	98.16
LeNet5-Caffe	**DPP** (This work)	1.51	99.07
	(Liu et al., 2020)	1.68	98.94
	Non-pruned baseline	100	99.18

4.2 VGG-16 AND MOBILENET V1 ON CIFAR-10 AND CIFAR-100

We evaluate DPP for structured pruning feature maps and the fully-connected layers on VGG-16 (Simonyan & Zisserman, 2015), which consists of 13 convolutional layers. Additionally, we use two fully-connected layers. For this experiment, we use case (c) for the convolutional stage, and case (a) for the fully-connected stage (Table 2).

Training details for CIFAR-10 and CIFAR-100 include a learning rate schedule, which is decreased by half every 40 epochs. SGD optimizer is used with momentum=0.9. Data augmentation is used for this experiment. All weights are initialized based on He normal initialization. We compare our experiments with the recent work of (Liu et al., 2020), and with a magnitude-based pruning method (TensorFlow Model Optimization toolkit) with fixed sparsity.

Table 2: Experimental results of DPP for CIFAR-10 and CIFAR-100 datasets using VGG-16 and MobileNet v1 architectures

Dataset	Network	Model	Remain.(%)	Accuracy(%)	Fixed Sparsity
CIFAR-10					
	VGG-16	**DPP** (This work)	12.14	93.36	**Yes**
		(Liu et al., 2020)	8.82	93.93	**No**
		Magnitude-based	10	88.59	**Yes**
		Non-pruned baseline	100	93.75	**-**
	MobileNet v1	**DPP** (This work)	36.95	93.14	**Yes**
		Magnitude-based	37	92.7	**Yes**
		Non-pruned baseline	100	93.67	**-**
CIFAR-100					
	VGG-16	**DPP** (This work)	18.8	70.32	**Yes**
		Magnitude-based	19	68.9	**Yes**
		Non-pruned baseline	100	70.40	**-**
	MobileNet v1	**DPP** (This work)	40	72.35	**Yes**
		Magnitude-based	40	70.19	**Yes**
		Non-pruned baseline	100	72.50	**-**

4.3 SPARSE BINARIZATION ON MNIST

We then turn to assess the performance of DPP for networks with binary parameters. Note that this is possible with DPP since it does not rely on the magnitude of the weights for sparsification.

We train our binary weights using the BinaryConect method (Courbariaux et al., 2015). Results using DPP are shown in Table 3. To the best of our knowledge, there is no existing method that integrates pruning and binarization for LeNet on MNIST, therefore, DPP is compared with its non-pruned baseline. For both architectures, very competitive test accuracies are obtained, and in the case of LeNet-5, the sparse model outperforms its non-pruned counterpart.

For binary LeNet300-100, we use a learning rate of 0.001 with the Adam optimizer (momentum=0.9), and batch size of 16. For binary LeNet-5, we use a learning rate of 0.01 and a batch size of 512.

While the attainable sparsity for these binary networks is reduced compared to its full-precision counterparts, the combination of both model compression techniques (pruning + binarization) could be used to deploy models at hardware level with lower memory.

Table 3: Experimental results of DPP for MNIST dataset using binary LeNet architectures

Network	Model	Remaining parameters (%)	Accuracy (%)
LeNet300-100	**DPP** (This work)	21	96.81
	Non-pruned baseline	100	97.9
LeNet5-Caffe	**DPP** (This work)	4.1	98.36
	Non-pruned baseline	100	98.1

4.4 Analysis of sparsity belief over time

We visualize the metrics in figure 3, normalized by the upper bound of the entropy to faciliate straightforward comparison between layers. Recall that high entropy of the average mask indicates diversity among the neurons in a layer, low conditional entropy indicates confidence in a specific pattern and high mutual information indicates both diversity and confidence.

We find an interesting dynamic where each of the neurons quickly learns a pattern with high confidence, but then reduces confidence again. Finally, the patterns slowly converge to a more confident distribution. This might facilitate quickly reaching an optimum followed by some additional exploration.

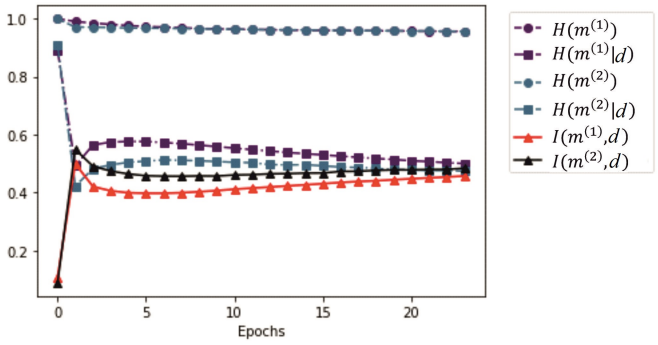

Figure 3: PruningDiversity metrics for both the first layer, $I(m^{(1)}, d)$, and the second $I(m^{(2)}, d)$, of LeNet300-100 on MNIST. The AveragePruneEntropy and EntropyAverageMas are also shown for both layers.

5 Discussion

In this paper, we propose dynamic probabilistic pruning (DPP), an algorithm that enables training sparse networks based on stochastic and dynamic masking. DPP learns to prune by selecting a set amount of connections between input and output nodes. Remarkably, DPP enables structured pruning for fully-connected and convolutional layers, suitable for hardware implementations. Leveraging its probabilistic nature, we showed how one can assess the confidence and diversity of pruning masks among neurons by monitoring proposed information-theoretic metrics. Since DPP does not rely on magnitudes for determining the relevance of weights, it can be straightforwardly integrated with weight binarization. We test its performance for three benchmark datasets and obtain competitive accuracies for different architectures. In concussion, our method generates ultra-compressed models, allowing the integration of binarization and pruning, while providing a level of structured sparsity, enabling a more efficient implementation on existing hardware platforms. Further, integration of DPP and quantization should be explored, and an evaluation of the trade-off between accuracy, sparsity and quantization level should be performed.

REFERENCES

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert A. Legenstein. Deep rewiring: Training very sparse deep networks. 2018.

G. Chechik, I. Meilijson, and E. Ruppin. Neuronal regulation: A mechanism for synaptic pruning during brain maturation. 11(8):2061–2080, 1999. ISSN 0899-7667. doi: 10.1162/089976699300016089.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A Survey of Model Compression and Acceleration for Deep Neural Networks. *arXiv:1710.09282 [cs]*, September 2019. URL http://arxiv.org/abs/1710.09282. arXiv: 1710.09282.

Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL http://arxiv.org/abs/1602.02830.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. URL http://arxiv.org/abs/1511.00363.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc{\textbackslash}textquotesingle Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2148–2156. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5025-predicting-parameters-in-deep-learning.pdf.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. 2019. URL http://arxiv.org/abs/1907.04840.

S. Dey, D. Chen, Z. Li, S. Kundu, K. Huang, K. M. Chugg, and P. A. Beerel. A highly parallel fpga implementation of sparse neural network training. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–4, 2018.

E. J. Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 33, 1954.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015. URL http://arxiv.org/abs/1506.02626.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2016. URL http://arxiv.org/abs/1510.00149.

Stephen Jose Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, pp. 177–185. Morgan-Kaufmann, 1989. URL http://papers.nips.cc/paper/156-comparing-biases-for-minimal-network-construction-with-back-propagation.pdf.

Babak Hassibi, David G. Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. In J. D. Cowan, G. Tesauro, and J. Alspector (eds.), *Advances in Neural Information Processing Systems 6*, pp. 263–270. Morgan-Kaufmann, 1994. URL http://papers.nips.cc/paper/749-optimal-brain-surgeon-extensions-and-performance-comparisons.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250, 2016. URL http://arxiv.org/abs/1607.03250.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18, 09 2016.

Iris A.M. Huijben, Bastiaan S. Veeling, and Ruud J.G. van Sloun. Deep probabilistic subsampling for task-adaptive compressed sensing. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SJeq9JBFvH.

Iris AM Huijben, Bastiaan S Veeling, and Ruud JG van Sloun. Learning sampling and model-based signal recovery for compressed sensing mri. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8906–8910. IEEE, 2020b.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 11 2016. URL http://arxiv.org/abs/1611.01144.

W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *arXiv preprint arXiv:1903.06059*, 2019.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990. URL http://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi: 10.1109/5.726791.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. URL http://arxiv.org/abs/1810.02340.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. *arXiv:1608.08710 [cs]*, March 2017. URL http://arxiv.org/abs/1608.08710. arXiv: 1608.08710.

Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJem8lSFwB.

Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden Kwok-Hay So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *ICLR*, 2020.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlnB3C5Ym.

L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang. An efficient hardware accelerator for sparse convolutional neural networks on fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–25, 2019.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Machine Learning*, 2017.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. 9(1):2383, 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3. URL https://www.nature.com/articles/s41467-018-04316-3. Number: 1 Publisher: Nature Publishing Group.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational Dropout Sparsifies Deep Neural Networks. *arXiv:1701.05369 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1701.05369. arXiv: 1701.05369.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. 2019. URL http://arxiv.org/abs/1902.05967.

Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017. URL http://arxiv.org/abs/1704.05119.

Yue Niu, Hanqing Zeng, Ajitesh Srivastava, Kartik Lakhotia, Rajgopal Kannan, Yanzhi Wang, and Viktor K. Prasanna. Spec2: Spectral sparse cnn accelerator on fpgas. *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pp. 195–204, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. Hardware for Machine Learning: Challenges and Opportunities. *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–8, April 2017. doi: 10.1109/CICC.2017.7993626. URL http://arxiv.org/abs/1612.07625. arXiv: 1612.07625.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. *arXiv:1608.03665 [cs, stat]*, October 2016. URL http://arxiv.org/abs/1608.03665. arXiv: 1608.03665.

S.M. Xie and S. Ermon. Reparameterizable subset sampling via continuous relaxations. In *International Joint Conference on Artificial Intelligence*, 2019.

Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161?170. Association for Computing Machinery, 2015.

Jialiang Zhang and Jing Li. Improving the performance of opencl-based fpga accelerator for convolutional neural network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 25?34. Association for Computing Machinery, 2017.

Min Zhang, Linpeng Li, Hai Wang, Yan Liu, Hongbo Qin, and Wei Zhao. Optimized compression for implementing convolutional neural networks on fpga. *Electronics*, 8:295, 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. 2017.