DKM: DIFFERENTIABLE k-MEANS CLUSTERING LAYER FOR NEURAL NETWORK COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural network (DNN) model compression for efficient on-device inference becomes increasingly important to reduce memory requirements and keep user data on-device. To this end, we propose a novel differentiable k-means clustering layer (DKM) and its application to train-time weight-clustering for DNN model compression. DKM casts k-means clustering as an attention problem and enables joint optimization of the DNN parameters and clustering centroids. Unlike prior works that rely on additional parameters and regularizers, DKM-based compression keeps the original loss function and model architecture fixed. We evaluated DKM-based compression on various DNN models for computer vision and natural language processing (NLP) tasks. Our results demonstrate that DKM delivers superior compression and accuracy trade-off on ImageNet1k and GLUE benchmarks. For example, DKM-based compression can offer 74.5% top-1 ImageNet1k accuracy on ResNet50 with 3.3MB model size (**29.4x** model compression factor). For MobileNet-v1, which is a challenging DNN to compress, DKM delivers 62.8% top-1 ImageNet1k accuracy with 0.74 MB model size (**22.4x** model compression factor). This result is 6.8% higher top-1 accuracy and 33% relatively smaller model size than the current state-of-the-art DNN compression algorithms. Additionally, DKM enables compression of DistilBERT model by **11.8x** with minimal (1.1%) accuracy loss on GLUE NLP benchmarks.

1 INTRODUCTION

Deep neural networks (DNN) have demonstrated super-human performance on many cognitive tasks (Silver et al., 2018). While a fully-trained uncompressed DNN is commonly used for server-side inference, on-device inference is preferred to enhance user experience by reducing latency and keeping user data on-device. Many such on-device platforms are battery-powered and resource-constrained, demanding a DNN to meet the stringent resource requirements such as power-consumption, compute budget and storage-overhead (Wang et al., 2019b; Wu et al., 2018).

One solution is to design a more efficient and compact DNN at the architecture level, such as MobileNet (Howard et al., 2017). Another solution would be to compress a model with small accuracy regression so that it takes less storage and reduces System on Chip (SoC) memory bandwidth utilization, which can minimize power-consumption and latency. To this end, various DNN compression techniques have been proposed (Wang et al., 2019b; Dong et al., 2020; Park et al., 2018; Rastegari et al., 2016; Fan et al., 2021; Stock et al., 2020). Among them, weight-clustering/sharing (Han et al., 2016; Wu et al., 2018; Ullrich et al., 2017; Stock et al., 2020) has been shown to deliver a high DNN compression ratio where weights are clustered into a few shareable weight values (or centroids) based on well-known k-means clustering. Once weights are clustered, to shrink the model size, one can store indices (2bits, 4bits, etc. depending on the number of clusters) with a lookup table rather than actual floating-point values.

Designing a compact DNN architecture and enabling weight-clustering together could provide the best solution in terms of efficient on-device inference. However, the existing model compression approaches do not usefully compress an already-compact DNN like MobileNet, presumably because the model itself does not have significant redundancy. We conjecture that such limitation comes from the fact that weight-clustering through k-means algorithm (both weight-cluster assignment and weight update) has not been fully optimized with the target task. The fundamental complexity

in applying k-means clustering for weight-sharing comes from the following: **a)** both weights and corresponding k-means centroids are free to move (a general k-means clustering with fixed observations is already NP-Hard), **b)** the weight-to-cluster assignment is a discrete process which makes k-means clustering non-differentiable, preventing effective optimization.

In this work, we propose a new activation layer for differentiable k-means clustering, DKM, based on an attention mechanism (Bahdana et al., 2015) to capture the weight and cluster interactions seamlessly, and further apply it to enable train-time weight-clustering for model compression. Our major contributions include the following:

- We propose a novel differentiable k-means clustering layer (DKM) for deep learning, which serves as a generic activation layer.
- We demonstrate that multi-dimensional k-means clustering can offer a high-quality model for a given compression ratio target.
- We apply DKM to compress a DNN model and demonstrate the state-of-the-art results on both computer vision and natural language models/tasks.

2 Related Works

Model compression using clustering: DeepCompression (Han et al., 2016) proposed to apply k-means clustering for model compression. DeepCompression initially clusters the weights using k-means algorithm. All the weights that belong to the same cluster share the same weight value which is initially the cluster centroid. In the forward-propagation, the shared weight is used for each weight. In the backward-propagation, the gradient for each shared weight is calculated and used to update the shared value. This approach might degrade model quality because it cannot formulate weight-cluster assignment during gradient back propagation (Yin et al., 2019).

HAQ (Wang et al., 2019b) uses reinforcement learning to search for the optimal quantization policy on different tasks. For model compression, HAQ uses k-means clustering similar to DeepCompression (Han et al., 2016) yet with flexible bit-width on different layers. Our work is orthogonal to this work because the k-means clustering can be replaced with our DKM with a similar flexible configuration. "And The Bit Goes Down" (Stock et al., 2020) algorithm is based on Product Quantization and Knowledge Distillation. It evenly splits the weight vector of N elements into N/d contiguous d dimensional sub-vectors, and clusters the sub-vectors using weighted k-means clustering to minimize activation change from that of a teacher network. GOBO (Zadeh & Moshovos, 2020) first separates outlier weights far from the average of the weights of each layer and stores them uncompressed while clustering the other weights by an algorithm similar to k-means.

Model compression using regularization: Directly incorporating k-means clustering in the training process is not straightforward (Wu et al., 2018). Hence, (Ullrich et al., 2017) models weight-clustering as Gaussian Mixture Model (GMM) and fits weight distribution into GMM with additional learning parameters using KL divergence (i.e., forcing weight distribution to follow k Gaussian distributions with a slight variance). (Wu et al., 2018) proposed deep k-means to enable weight-clustering during re-training. By forcing the weights that have been already clustered to stay around the assigned center, the hard weight-clustering is approximated with additional parameters. Both (Ullrich et al., 2017) and (Wu et al., 2018) leverage regularization to enforce weight-clustering with additional parameters, which will interfere with the original loss target and requires additional updates for the new variables (i.e., singular value decomposition (SVD) in (Wu et al., 2018)). Also, relying on the modified loss cannot capture the dynamic interaction between weight distributions and cluster centroids within a batch, thus requiring an additional training flow for re-training.

Enhance Model compression using dropout: Quant-Noise (Fan et al., 2021) is a structured dropout which only quantizes a random subset of weights (using any quantization technique) and thus can improve the predictive power of a compressed model. For example, when combined with (Stock et al., 2020), (Fan et al., 2021) showed good compression vs. accuracy trade-off on ResNet50 for ImageNet1k.

Model quantization: Besides clustering and regularization methods, model quantization can also reduce the model size, and training-time quantization techniques have been developed to improve the accuracy of quantized models. EWGS (J. Lee, 2021) adjusts gradients by scaling them up or

down based on the Hessian approximation for each layer. PROFIT (Park & Yoo, 2020) adopts an iterative process and freezes layers based on the activation instability.

Efficient networks: Efficient architectures such MobileNet (Howard et al., 2017; Sandler et al., 2018), Efficient Net (Tan & Le, 2019; 2021) and ESPNet (Mehta et al., 2019) are designed to be memory-efficient. MobileNet-v1 (Howard et al., 2017) on ImageNet1k dataset has top-1 accuracy of 70.3% with 16.1 MB of memory in comparison to a ResNet18 which has 69.3% accuracy with 44.6 MB of model size. Our method can be applied to these compact networks to reduce their model sizes further while delivering competitive accuracies.

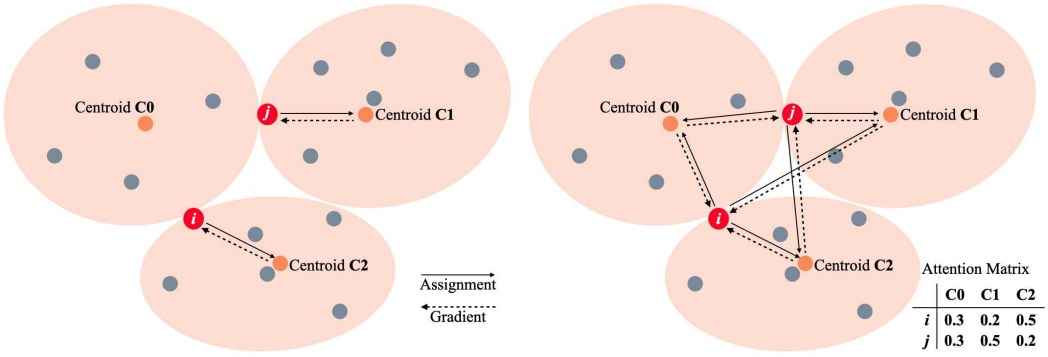
3 ALGORITHM

3.1 MOTIVATION

Popular weight-clustering techniques for DNN model compression (J. Lee, 2021; Han et al., 2016; Dong et al., 2020; Stock et al., 2020) are based on k-means clustering along with enhancements such as gradient scaling/approximation. Using k-means clustering, the weights are clustered and assigned to the nearest centroids which are used for forward/backward-propagation during training as illustrated in Fig. 1 (a). Such conventional approaches based on clustering have two critical drawbacks:

- The weight-to-cluster assignment in conventional approaches are based on a distance metric without being aligned with the training loss function.
- Gradients for the weights are computed in an ad-hoc fashion: the gradient of a centroid is re-purposed as the gradient of the weights assigned to the centroid.

These limitations are more pronounced for the weights on the boundary such as i and j in Fig. 1 (a). In the conventional approaches, i and j are assigned to the centroids C_2 and C_1 respectively, simply because of their marginal difference in a distance metric. However, assigning i to C_0 and j to C_2 could be better for the training loss as their difference in distance is so small. Such lost opportunity cost is especially higher with a smaller number of centroids (or fewer bits for quantization), as each *unfortunate* assignment can degrade the training loss significantly, yet without recoverable solutions.



(a) Conventional weight-clustering (Han et al., 2016; Wang et al., 2019b; Stock et al., 2020; J. Lee, 2021)

(b) Attention-based weight-clustering in DKM

Figure 1: In conventional weight-clustering algorithms, the boundary weights i and j are assigned to clusters C_2 and C_1 based on the distance metric respectively, which is neither necessarily suitable for the task nor differentiable against the loss function. DKM instead applies soft assignment using attention mechanism during forward-propagation and enables differentiable backward-propagation, which allows weights to consider other non-nearest clusters (especially helpful for the boundary weights) and shuttle among multiple clusters in order to directly optimize their assignments based on the task loss function.

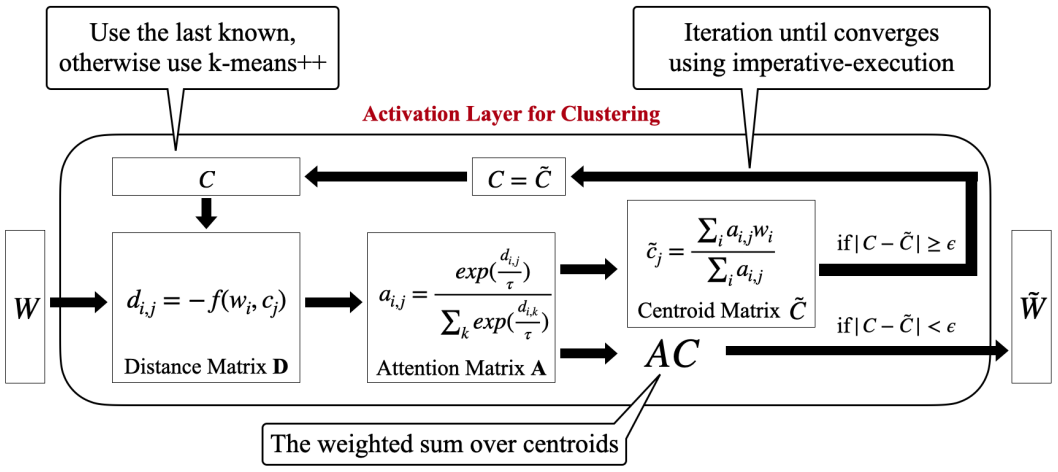


Figure 2: Weight-sharing using attention matrix $\mathbf{A}$ is iteratively performed in a DKM layer until the centroids ($\mathbf{C}$) converge. Once converged, a compressed weight, $\tilde{\mathbf{W}}$ is used for forward-propagation. Since DKM is a differentiable activation layer, backward-propagation will run through the iterative loop and the gradients for the weights will be computed against the task loss function.

We overcome such limitations by interpreting weight-centroid assignment as distance-based attention optimization (Bahdana et al., 2015) in DKM as depicted in Fig. 1 (b) and letting each weight interact with all the centroids. Note that such attention mechanism naturally allows differentiable and iterative k-means clustering as in Fig. 2 to be an activation layer, and is highly effective in weight-clustering for DNN compression.

3.2 Differentiable K-means clustering layer for Weight-clustering

DKM can perform a differentiable train-time weight-clustering iteratively for k clusters as shown in Fig. 2 for the DNN model compression purpose. Let $\mathbf{C} \in R^k$ be a vector of cluster centers and $\mathbf{W} \in R^N$ be a vector of the weights, and then DKM performs as follows:

- In the first iteration $\mathbf{C}$ can be initialized either by randomly selected k weights from $\mathbf{W}$ or k-means++. For all subsequent iterations, the last known $\mathbf{C}$ from the previous batch is used to accelerate the clustering convergence.
- A distance matrix $\mathbf{D}$ is computed for every pair between a weight w_i and a centroid c_j using a differentiable metric f (i.e., the Euclidean distance) using $d_{ij} = -f(w_i, c_j)$.
- We apply softmax with a temperature τ on each row of $\mathbf{D}$ to obtain attention matrix $\mathbf{A}$ where $a_{i,j} = \frac{exp(\frac{d_{i,j}}{\tau})}{\sum_k exp(\frac{d_{i,k}}{\tau})}$ represents the attention from w_i and c_j.
- Then, we obtain a new centroid $\tilde{\mathbf{C}}$ by gathering all the attentions from $\mathbf{W}$ for each centroid by computing $\tilde{c}_j = \frac{\sum_i a_{i,j} w_i}{\sum_i a_{i,j}}$.
- We repeat this process till $\tilde{\mathbf{C}}$ is close enough to $\mathbf{C}$ at which point k-means has converged, and we compute $\mathbf{AC}$ to get $\tilde{\mathbf{W}}$ for forward-propagation (as in Fig. 2).

The iterative process will be dynamically executed imperatively in PyTorch (Paszke et al., 2019) and Tensorflow-Eager (Agrawal et al., 2019) and is differentiable for backward-propagation, as $\tilde{\mathbf{W}}$ is based on the attention between weights and centroids. DKM uses soft weight-cluster assignment which could be hardened in order to impose weight-clustering constraints. The level of hardness can be controlled by the temperature τ in the softmax operation. During inference we use the last attention matrix (i.e., $\mathbf{A}$ in Fig. 2) from a DKM layer to snap each weight to the closest centroid of the layer and finalize weight-clustering as in prior arts (i.e., no more attention), but such assignment is expected to be tightly aligned with the loss function, as the weights have been annealed by shuttling among centroids.

Using DKM for model compression allows a weight to change its cluster assignment during train-time, but eventually encourages it to settle with the best one w.r.t the task loss. Optimizing both weights and clusters simultaneously and channeling the loss directly to the weight-cluster assignment is based on the attention mechanism. Since each step in DKM is without additional learnable parameters and transparent to a model and loss function, we can reuse the existing training flow and hyper-parameters. Therefore, the critical differences between DKM-based compression and the prior works can be summarized as follows:

- Instead of hard weight-cluster assignment and approximated gradient (Han et al., 2016; Wang et al., 2019b; J. Lee, 2021; Stock et al., 2020), DKM uses flexible and differentiable attention-based weight-clustering and computes gradients w.r.t the task loss without approximation.
- Instead of modifying the loss function with regularizers to enforce clustering (Ullrich et al., 2017; Wu et al., 2018), DKM (as a differentiable activation) can be inserted into forward functions, making the optimization fully aligned with the task objective (i.e., no interference in loss).
- DKM requires no additional learnable parameters (Ullrich et al., 2017; J. Lee, 2021), thus making the training flow simple. For example, DKM-base approach does not need to substitute a convolution layer with a specialized version with additional learning parameters.
- DKM requires no additional computation such as Hessian trace (Dong et al., 2020) or SVD (J. Lee, 2021; Wu et al., 2018) for gradient approximation, because DKM uses a differentiable process.
- DKM-based compression does not need a complex training flow such as freezing/progress (Park & Yoo, 2020) or distillation (Stock et al., 2020), keeping training flow simple and unchanged.

3.3 MULTI-DIMENSIONAL DKM

DKM can be naturally extended into multi-dimensional weight-clustering (Stock et al., 2020) due to its simplicity, and is highly effective due to its differentiability. We split N elements of weights into $\frac{N}{d}$ contiguous d dimensional sub-vectors and cluster the sub-vectors ($\mathbf{W} \in R^{\frac{N}{d} * d}$). For example, we simply flatten all the convolutional kernels into a $(\frac{N}{d}, d)$ matrix across both kernel and channel boundaries and apply multi-dimensional DKM to the matrix for clustering in our implementation. Accordingly, the cluster centroids will become d-dimensional as well ($\mathbf{C} \in R^{k*d}$) and the metric calculation is done in the d-dimensional space. With the multi-dimensional scheme, the effective *bit-per-weight* becomes $\frac{b}{d}$ for b-bit/d-dim clustering.

Such multi-dimensional clustering could be ineffective for conventional methods (i.e., DNN training not converging) (Stock et al., 2020; Wang et al., 2019b; J. Lee, 2021), as now a weight might be on the boundary to multiple centroids, and the chance of making wrong decisions grows exponentially with the number of centroids. For example, there are only two centroids for 1bit/1dim clustering, while there are 16 centroids in 4bit/4dim clustering, although both have the same effective *bit-per-weight*. Intuitively, however, DKM can work well with such multi-dimensional configurations as DKM naturally optimizes the assignment w.r.t the task objective and can even recover from a wrong assignment decision over the training-time optimization process.

For a given sufficiently large $N \gg d$ in 32 bits, the compression ratio is $\frac{32d}{b}$ and its entropy is b, assuming all the clusters are at the same size. Since higher entropy indicates more flexibility in the weight distribution and better model quality (Park et al., 2017), it can be expected that increasing b and d at the same ratio as much as possible can improve the model quality for for a given target compression ratio (see Section 4.1 for results).

The storage complexity of a DKM layer with N parameters is $O(r\frac{N}{d}2^b)$ where r is the number of iterations per Fig. 2, as all the intermediate results such as $\mathbf{D}$ and $\mathbf{A}$ at each iteration need to be kept for backward-propagation. Therefore, when there are a large number of weights and the target precision is high (i.e., more clusters), the training flow could suffer from out-of-memory error. One solution to such memory overhead is to use a sparse representation for $\mathbf{A}$ by keeping top-k centroids for each weight.

Model	Metrics	Base 32bit	DC 2bit	HAQ flex	ATB small	ATB large	DKM configuration		b/w*
ResNet18	Top-1 (%)	69.8			65.8	61.1	65.2✓	cv†:6/8§	0.717
	Size (MB)	44.6			1.58	1.07	**1.00**	fc‡:6/10	
ResNet50	Top-1 (%)	76.1	68.9	70.6	73.8 74.3△	68.2 68.8△	**74.5**	cv:6/6	1.077
	Size (MB)	97.5	6.32	6.30	5.34	3.43	**3.31**	fc:6/4	
MobileNet v1	Top-1 (%)	70.9	37.6	57.1	nc°	nc	**63.9**	cv:4/4	1.427
	Size (MB)	16.1	1.09	1.09			**0.72**	fc:4/2	
MobileNet v2	Top-1 (%)	71.9	58.1	66.8	nc	nc	**67.9**	cv:2/1	2.010
	Size (MB)	13.3	0.96	0.95			**0.84**	fc:4/4	

* effective bit-per-weight (see Section 3.3); ° not converging
† the convolution layers ; ‡ the last fully connected layer
§ clustering with 6 bits and 8 dimensions
✓ also, 66.8 Top-1 accuracy and 1.49 MB superior to ATB small with cv:4/4, fc:8/4
△ ATB with quantization-noise (Fan et al., 2021)

Table 1: Top-1 accuracy and model compression for ImageNet1k: DKM-powered compression outperforms existing compression approaches in both Top-1 accuracy and compressed model size for all the tested convolutional networks.

4 Experimental Results

We compared our DKM-based compression with multiple state-of-the-art quantization or compression schemes on various computer vision and natural language tasks and models. To study the trade-off between model compression and accuracy regression, we disabled activation quantization in every experiment for all approaches. All our experiments with DKM were done on two x86 Linux machine with eight NVIDIA V100 GPUs each in a public cloud infrastructure, and activation layers stayed unchanged, as our main goal is the model size reduction through compression. We used a SGD optimizer with momentum 0.9, and fixed the learning rate at 0.008 (without individual hyper-parameter tuning) for all the experiments for DKM. Each compression scheme starts with publicly available pre-trained models.

4.1 ImageNet1k

We compared our DKM-based compression with DeepCompression (or **DC**) (Han et al., 2016), **HAQ** (Wang et al., 2019b), and "And The Bit Goes Down" (or **ATB**) (Stock et al., 2020) combined with Quantization-noise (Fan et al., 2021) which are all designed for model compression, and summarized results in Table 1. We set the mini-batch size 128 per GPU (i.e., global mini-batch size of 2048) and ran for 200 epochs for all DKM cases. Since the public ATB implementation does not include MobileNet-v1/v2 cases (Howard et al., 2017; Sandler et al., 2018), we added the support for these two by following the paper and the existing ResNet18/50 (He et al., 2016) implementations. Instead of using a complex RL technique as in HAQ (Wang et al., 2019b), for DKM experiments, we simply used static configurations for all the convolution layers (noted as cv) and the last fully connected layer (note as fc), except that we applied 8 bit clustering to any layer with fewer than 10,000 parameters.

Even using such a simple method, DKM offers a Pareto superiority to other schemes as shown in Table 1. For ResNet50 and MobileNet-v1/v2, DKM delivered compression configurations that yielded both better accuracy and higher compression ratio than the prior arts. For ResNet18, DKM was able to make a smooth trade-off between accuracy and compression and find Pareto superior configurations to ATB: DKM can get 65.2% Top-1 accuracy with 1MB memory budget which is superior to ATB-large (at 61.1% Top-1 accuracy with 1.07MB) and also ATB-small (at 65.8% Top-1 accuracy with 1.58MB). For MobileNet-v1/v2, ATB failed to converge, but DKM outperforms DC and HAQ in terms of both accuracy and size at the same time.

Metrics	Base (32bit)	RPS	DKM 4/1	DKM 4/2	DKM 8/8[§]
Top-1	69.8	67.9	**70.9**	70.3	68.5
CR[◊]	1	4	8	16	32

[◊] compression ratio; [§] clustering with 8 bits and 8 dimensions

Table 2: GoogleNet training performance for ImageNet1k: DKM-based compression offered 2x better compression ratio with 3% higher top-1 accuracy than RPS.

We also compared DKM with a well-known regularization-based clustering method on GoolgeNet in Table 2, **RPS** (Wu et al., 2018) which has demonstrated superior performance to another regularization approach (Ullrich et al., 2017). Note that only convolution layers are compressed, following the setup in RPS (Wu et al., 2018). Table 2 clearly indicates that DKM can allow both much better compression and higher accuracy than RPS even with 1 bit-per-weight.

		ResNet18	ResNet50	MobileNet-v1	MobileNet-v2
	Base (32 bit)	69.8	76.1	70.9	71.9
3 bit	PROFIT			69.6	69.6
	EWGS	**70.5**	**76.3**	64.4	64.5
	PROFIT+EWGS			68.6	69.5
	DKM	69.9	76.2	**69.9**	**70.3**
2 bit	PROFIT			63.4	61.9
	EWGS	**69.3**	**75.8**	52.0	49.1
	DKM	68.9	75.3	**66.4**	**66.2**
1 bit	PROFIT			nc[°]	nc
	EWGS	66.6	73.8	8.5	23.0
	DKM 1/1	65.0	72.1	5.9	50.8
	DKM $4/4^{\S}$	67.0	**73.8**	60.6	55.0
	DKM $8/8$	**67.8**	oom[□]	**64.3**	**62.4**
$\frac{1}{2}$ bit	DKM $4/8$	62.1	70.6	46.5	34.0
	DKM $8/16$	**65.5**	**72.1**	**59.8**	**58.3**

[°] not converging; [□] out of memory ; [§] clustering with 4 bits and 4 dimensions

Table 3: Top-1 accuracy for ImageNet1k: When compared with the latest weight quantization algorithms, DKM-based algorithm shows superior Top-1 accuracy when the network is hard to optimize (i.e., MobileNet-v1/v2) or when a low precision is required (1 bit). Further, with multi-dimensional DKM (see Section 3.3), DKM delivers 64.3 % Top-1 accuracy for MobileNet-v1 with the 8/8 configuration which is equivalent to 1 bit-per-weight.

For a comprehensive study, we also compared our DKM-based algorithm with the latest scalar weight quantization approaches, **PROFIT** (Park & Yoo, 2020) and **EWGS** (J. Lee, 2021) (which have outperformed the prior arts in the low-precision regimes) by running their public codes on our environments with the recommended hyper-parameter sets. Table 3 summarizes our comparison results on ResNet18, ResNet50, and MobileNet-v1/v2 for the ImageNet1k classification task. Following the experimental protocol in (Zhang et al., 2018; J. Lee, 2021; Rastegari et al., 2016), we did not optimize the first and last layers for all the experiments in Table 3.

It clearly shows that our approach with DKM can provide compression comparable to or better than other approaches, especially for the low-bit/high-compression regimes. We denote clustering with b bits and d dimensions as b/d as it will assign $\frac{b}{d}$ bits in average to each weight, and the number of weight clusters is 2^b. Especially with multi-dim clustering such as 4/4 or 8/8 bits, our DKM-based compression outperforms other schemes at 1 bit, while PROFIT cannot make training converge for MobileNet-v1/v2. One notable result is 64.3% Top-1 accuracy of MobileNet-v1 with the 8/8

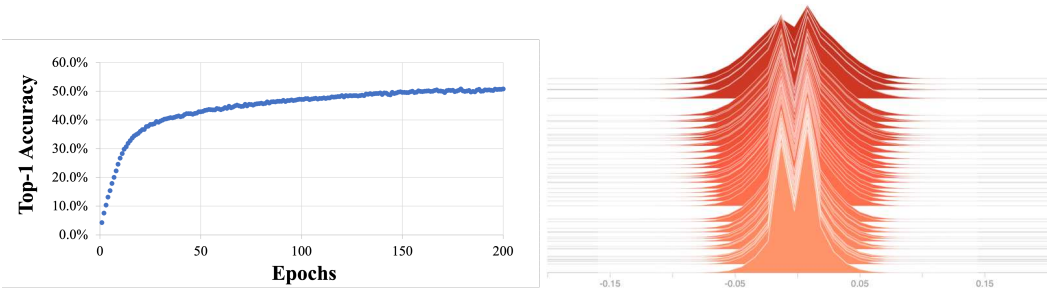
	ResNet18		ResNet50	
	GPU memory utilization (%)	Per-epoch runtime (sec)	GPU memory utilization (%)	Per-epoch runtime (sec)
3 bit	23.8	414.1	55.3	425.0
2 bit	21.7	401.2	49.2	429.3
1 bit	20.4	413.3	45.3	426.3
$4/4^{\S}$ bit	22.3	414.4	51.8	430.4
$4/8$ bit	20.7	423.6	46.7	410.4
$8/8$ bit	51.8	409.4	oom	
$8/16$ bit	35.1	421.9	78.3	454.6

§ clustering with 4 bits and 4 dimensions

Table 4: Memory and Runtime overheads from DKM on ResNet18/50.

bit configuration (which is 1 bit-equivalent). DKM with 8/16 bits (effectively 0.5 bit per weight) shows regression from the 8/8 bit configuration, but still retains a good accuracy level. We also tried PROFIT+EWGS as proposed in (J. Lee, 2021), which showed good results on MobileNet-v1/v2 for 3 bits but failed to converge for 2 and 1 bits.

With the overall compression ratio (or bit-per-weight) fixed, our experiments with DKM confirm that a higher d can yield a better quality training result. For the example of MobileNet-v2, DKM 8/16 yielded 24% better top-1 accuracy than DKM 4/8 although both have the same $\frac{1}{2}$ bit-per-weight, and the same trend is observed in other models. However, as discussed in Section 3.3, DKM 8/8 failed to train ResNet50 due to the memory limitation, while DKM 8/16 successfully trained the same model, because the larger dimension (i.e., 8 vs 16) reduces the memory requirement of the attention matrix. In detail, Table 4 shows the GPU memory utilization and per-epoch runtime for the DKM cases in Table 3. While DKM layers have negligible impacts on training speed, the GPU memory utilization increases with more bits (i.e., more clusters) and with smaller dimensions.



(a) Top-1 accuracy (b) Weight distribution of the largest conv layer.

Figure 3: MobileNet-v2 convergence with DKM 1/1: DKM delivers 50.8% top-1 accuracy with 1 bit compression by gradually clustering the weights into two centroids using the task objective only.

Fig. 3 lastly shows that DKM-based compression can offer a smooth convergence and gradual weight-clustering based on task loss back-propagated through DKM layers, without any extra regularization or custom training flows.

4.2 GLUE NLP Benchmarks

We compared our compression by DKM with **GOBO** (Zadeh & Moshovos, 2020) and **EWGS** (J. Lee, 2021) for BERT models on NLP tasks from the GLUE benchmarks (Wang et al., 2019a), QNLI (Question-answering NLI) and MNLI (Multi NLI). We fixed the learning rate as 1e-4 for all the experiments which worked best for EWGS, and all experiments used mini-batch size 64 per GPU (i.e., global mini-batch size of 1024).

		ALBERT	DistilBERT	BERT-tiny	MobileBERT
	Base (32 bit)	90.6	88.2	78.9	89.6
3 bit	EWGS	83.3	87.6	78.3	87.8
	DKM	**85.1**	**88.2**	**80.0**	**89.0**
2 bit	EWGS	79.6	85.4	77.9	81.6
	DKM	**81.7**	**87.4**	**80.0**	**83.7**
1 bit	EWGS	62.0	60.9	74.5	60.2
	DKM	79.0	82.8	**77.4**	69.8
	DKM $4/4^{\S}$	**80.0**	**84.0**	77.2	**78.3**

§ clustering with 4 bits and 4 dimensions

Table 5: Training performance for QNLI: DKM-based scheme outperforms EWGS in compressing various transformed-based architectures. Also, multi-dimensional DKM (see Section 3.3) largely improved the accuracy of MobileBERT with 1 bit-per-weight target using the 4/4 configuration.

Metrics	Base 32bit	GOBO xform$^{\dagger}3$,emb$^{\ddagger}4$	DKM xform$4/2^{\S}$,emb4	DKM xform$5/2$,emb3
Top-1	82.4	**81.3**	**81.3**	**81.3**
Size (MB)	255.4	23.9	21.8	**21.5**

† the transformer layers ; ‡ the embedding layer
§ clustering with 4 bits and 2 dimensions

Table 6: DistillBert training performance for MNLI: DKM-based compression offered 10% smaller model size with the same accuracy target than GOBO.

We compared our DKM-based compression against EWGS (J. Lee, 2021) on the QNLI dataset, and Table 5 demonstrates that DKM offers better predictability across all the tested models (Lan et al., 2019; Sanh et al., 2019; Turc et al., 2019; Sun et al., 2020) than EWGS. Note that the embedding layers were excluded from compression in QNLI experiments. As in ImageNet1k experiments, the 4/4 bit configuration delivers better qualities than the 1 bit configuration on all four BERT models, and especially performs well for the hard-to-compress MobileBERT. Table 5 also indicates that different transformer architectures will have different levels of regression for a given compression target. For the example of 1 bit, MobileBERT regressed most due to many hard-to-compress small layers, yet recovered back to a good accuracy with DKM 4/4.

When DKM compared against GOBO (Zadeh & Moshovos, 2020) (which has outperformed the prior arts on BERT compression) on DistilBERT with the MNLI dataset, our results in Table 6 clearly show that DKM could offer a better accuracy-compression trade-off than GOBO, and also enable fine-grained balance control between an embedding layer and others: using 2.5 bits for Transformer and 3 bits for embedding is better than 2 bits for Transformer and 4 bits for embedding for the case of DistilBERT.

5 Conclusion

In this work, we proposed a differentiable k-means clustering layer, DKM and its application to model compression. DNN compression powered by DKM yields the state-of-the-art compression quality on popular computer vision and natural language models, and especially highlights its strength in low-precision compression and quantization. The differentiable nature of DKM allows natural expansion to multi-dimensional k-means clustering, offering more than 22x model size reduction at 63.9% top-1 accuracy for highly challenging MobileNet-v1. We plan to extend DKM to handle weight-clustering and weight-pruning simultaneously for more aggressive DNN model optimization, and apply sparse-format for the internal tensors to reduce the memory overheads.

REFERENCES

Akshay Agrawal, Akshay Naresh Modi, Alexandre Passos, Allen Lavoie, Ashish Agarwal, Asim Shankar, Igor Ganichev, Josh Levenberg, Mingsheng Hong, Rajat Monga, and Shanqing Cai. Tensorflow eager: A multi-stage, python-embedded DSL for machine learning. *CoRR*, 2019.

Dzmitry Bahdana, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. In *NeurIPS*, 2020.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. Training with quantization noise for extreme model compression. In *ICLR*, 2021.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

B. Ham J. Lee, D. Kim. Network quantization with element-wise gradient scaling. In *CVPR*, 2021.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2019.

Sachin Mehta, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. In *CVPR*, 2019.

Eunhyeok Park and Sungjoo Yoo. Profit: A novel training method for sub-4-bit mobilenet models. In *ECCV*, 2020.

Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *CVPR*, 2017.

Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *ECCV*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, June 2018.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS*, 2019.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *ICLR*, 2020.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In *ACL*, 2020.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, 2019.

Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, 2021.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. In *CoRR*, 2019.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *ICLR*, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019a.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019b.

Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *ICML*, 2018.

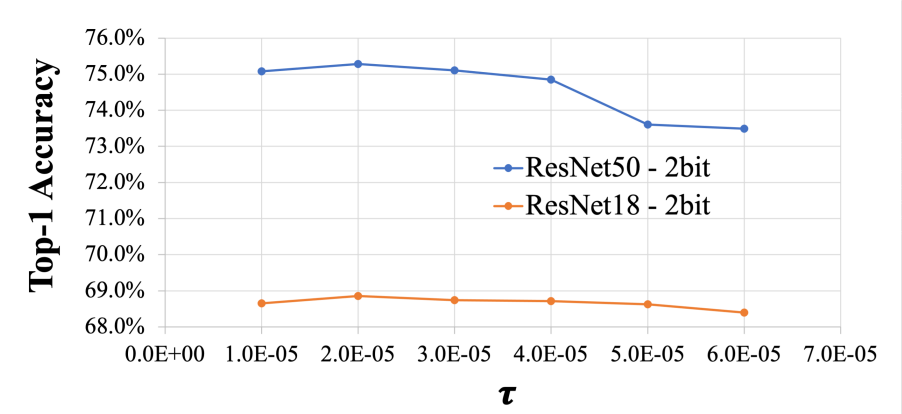
Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *ICLR*, 2019.

Ali Hadi Zadeh and Andreas Moshovos. GOBO: quantizing attention-based NLP models for low latency and energy efficient inference. *MICRO*, 2020.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, 2018.

A ABLATION STUDY: HYPER-PARAMETER τ SEARCH

In the current DKM implementation, we use a global τ to control the level of softness in the attention matrix. The selection of τ affects the model predictive power as shown in Fig. 4 where there appears to be an optimal τ for a given DNN architecture. For examples of ResNet18/80, $\tau = 2e - 5$ is the best value for the 2 bit clustering.



(a) ResNet18/50 Accuracy and τ

Figure 4: ResNet18/50 compression using 2 bits with varying τ values.

In our experiments, we used a binary search to find out the best τ values w.r.t. the top-1 accuracy, which are listed in Table 7. In general, one can observe that a complex compression task (i.e., higher compression targets, more compact networks) tends require a larger τ to provide enough flexibility or softness.

- For MobileNet-v1/v2, it requires about 10x larger τ values then for ResNet18/50, because they are based on a more compact architecture and harder to compress.
- When the number of bits decreases, the compression gets harder because there are fewer centroids to utilize, hence requiring a larger τ value.
- When the centroid dimension increases, the larger τ value is required, as the compression complexity increases (i.e., need to utilize a longer sequence).

	ResNet18	ResNet50	MobileNet-v1	MobileNet-v2
3 bit	8.0e-6	8.0e-6	5.0e-5	5.0e-5
2 bit	2.0e-5	2.0e-5	1.0e-4	1.0e-4
1 bit	5.0e-5	5.0e-5	3.0e-4	1.5e-4
$4/4^{\S}$ bit	5.0e-5	4.0e-5	1.0e-4	1.0e-4
$4/8$ bit	5.0e-5	5.0e-5	1.0e-4	1.0e-4
$8/8$ bit	8.0e-5	oom	1.0e-4	1.0e-4
$8/16$ bit	1.3e-4	6.0e-5	1.2e-4	1.4e-4

§ clustering with 4 bits and 4 dimensions

Table 7: τ for the DKM experiments in Table 3 in Section 4

It could be possible to cast τ as a learnable parameter for each layer or apply some scheduling to improve the model accuracy further (as a future work), but still both approaches need a good initial point which can be found using a binary search technique.

B RELATION TO EXPECTATION-MAXIMIZATION (EM)

Special sub-case of DKM where gradients are not propagated can be related to a standard EM formulation. Here is the formulation-level correspondence: Suppose

$$p(x) = \sum_{i=1}^{K} \frac{1}{K} \mathcal{N}(x|c_i, \sigma^2 = \tau/2)$$

is the Gaussian mixture model over cluster centers. Referring to Fig. 2, Maximizing log likelihood of weights $\ln P(W|C) = \sum_{i=1}^{N} \ln \left\{ \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(w_i|c_j, \sigma^2 = \tau/2) \right\}$ using the EM algorithm is equivalent to DKM for the case of $d_{i,j} = -(w_i - c_j)^2$. Then, the attention matrix $\mathbf{A}$ is equivalent to the responsibilities calculated in the E step, and updating $\mathbf{C}$ is equivalent to the M step. However, unlike EM where finding $\mathbf{C}$ every M step is the objective, DKM focuses on generating a representative $\tilde{\mathbf{W}}$ for the train-time compression for DNN.

Even though there is formulation-level similarity between DKM and EM, the way both are optimized is significantly different. While EM iteratively optimizes a specific likelihood function for a set of **fixed** observations, DKM needs to adjust (i.e., optimize) the observations (which are weights) without leading to a trivial solution such as all observations collapsing to a certain point. Hence, DKM can neither assume any statistical distribution nor optimize a specific likelihood function (i.e., the observations are dynamically changing). Therefore, DKM uses a simple softmax and rides on the back-propagation to fine-tune the observations w.r.t. the task loss function after unrolling multiple attention updates. When we propagate gradients, then this will turn into a stochastic non-convex joint optimization where we simultaneously optimize observations and centroids for the task loss function, which is shown to offer better accuracy vs. compression trade-offs according to our experiments.