DECOUPLING WEIGHT REGULARIZATION FROM BATCH SIZE FOR MODEL COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Conventionally, compression-aware training performs weight compression for every mini-batch to compute the impact of compression on the loss function. In this paper, in order to study when would be the right time to compress weights during optimization steps, we propose a new hyper-parameter called "Non-Regularization period" or NR period during which weights are not updated for regularization. We first investigate the influence of NR period on regularization using weight decay and weight random noise insertion. Throughout various experiments, we show that stronger weight regularization demands longer NR period (regardless of batch size) to best utilize regularization effects. From our empirical evidence, we argue that weight regularization for every mini-batch allows small weight updates only and limited regularization effects such that there is a need to search for right NR period and weight regularization strength to enhance model accuracy. Consequently, NR period becomes especially crucial for model compression where large weight updates are necessary to increase compression ratio. Using various models, we show that simple weight updates to comply with compression formats along with long NR period is enough to achieve high compression ratio and model accuracy.

1 INTRODUCTION

For Deep Neural Networks (DNNs), a common training method updates weights by gradient descent and explicit weight manipulation through regularization. To overcome some practical issues of gradient descent on non-convex optimization problems, there have been several enhancements such as learning rate scheduling and adaptive update schemes using momentum and update history (Ruder, 2016). Optimizing batch size is another way to yield efficient gradient descent. Note that large batch size has the advantage of enhancing parallelism of the training system in order to speed up training, critical for DNN research (Dean et al., 2012). Despite such advantages, small batch size is preferred because it improves generalization associated with flat minima search (Keskar et al., 2016) and other hyper-parameter explorations are more convenient (Masters & Luschi, 2018). Small batch size also affects weight regularization. For example, weight decay is conducted for every mini-batch and, thus, if batch size is modified, then the weight decay factor should also be adjusted accordingly (Loshchilov & Hutter, 2017).

Weight regularization is a process that adds information to the model as a way to avoid overfitting (Goodfellow et al., 2016; van Laarhoven, 2017). In this paper, we explore weight compression as a form of weight regularization as it severely restricts the search space of weights (i.e., regularized by compression forms). Moreover, model compression shrinks the effective model size, which is an important regularization principle (Goodfellow et al., 2016) (note that improved model accuracy by model compression is reported (Frankle et al., 2019)). Weights are regularized in numerous ways by model compression. For example, each weight can be pruned (e.g., Han et al. (2015)) or quantized (e.g., Guo et al. (2017)) to yield a sparse model representation or to reduce the number of bits to represent each weight. While weight regularization for model compression can simply be performed once after training, compression-aware training can improve model accuracy by reflecting the impact of model compression on the loss function for every mini-batch update (Courbariaux et al., 2015; Zhu & Gupta, 2017; Zhu et al., 2017; Guo et al., 2017). Now, our question in this paper is the following: *Should weights be updated at the same time for both regularization and gradient descent?* If the answer is *"No,"* then we would need to consider a new hyper-parameter

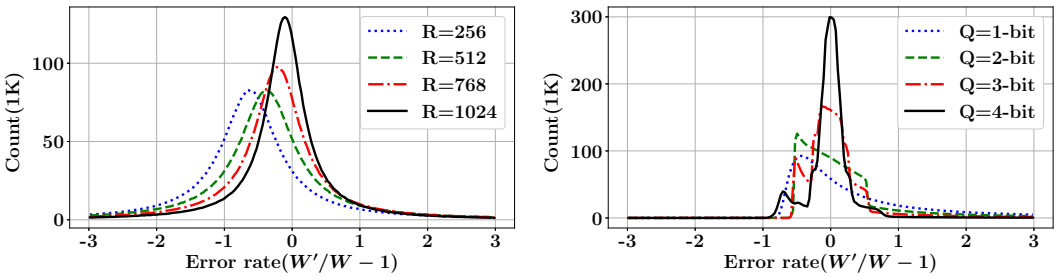


Figure 1: Distribution of weight noise ϵ after low-rank approximation via SVD (Left) or quantization (Right) when R is the rank and Q is the number of quantization bits.

search space with the right weight regularization timing for existing weight regularization methods and model compression schemes.

In this paper, we report the following observations: 1) Model compression tends to induce weight decay and random weight noise; 2) For weight decay and weight noise insertion, less frequent weight updates allow larger amounts of weight updates to best utilize regularization effects regardless of batch size; and 3) Similarly, if weight updates for compression are conducted less frequently, training for model compression permits stronger weight regularization and, thus, achieves higher compression ratios.

We confirm our above observations through various experiments and propose an asynchronous regularization method to decouple weight regularization from batch size selection that is optimized for gradient descent. We verify that our simple model compression techniques (without modifying the underlying training procedures) based on asynchronous weight updates (with longer non-regularization period) can achieve higher compression ratio and higher model accuracy compared to previous techniques that demand substantial modifications to the training process.

2 Noise Model on Weight Compression

We first study the relationship between model compression ratio and weight regularization strength using quantization and singular-value decomposition (SVD) as model compression techniques. We assume a popular quantization method based on binary codes for which a weight vector $\boldsymbol{w}$ is approximated to be $\sum_{i=1}^{q} \alpha_i \boldsymbol{b}_i$ for q-bit quantization, where α is a scaling factor and $\boldsymbol{b}(=\{-1,+1\}^n)$ is a binary vector, and n is the vector size. The quantization error $||\boldsymbol{w} - \sum_i \alpha_i \boldsymbol{b}_i||^2$ is minimized by a method proposed by Xu et al. (2018) to compute α and $\boldsymbol{b}$. For SVD, a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ is approximated to be $\boldsymbol{W'} \in \mathbb{R}^{m \times n}$ by minimizing $||\boldsymbol{W} - \boldsymbol{W'}||$ subject to $\text{rank}(\boldsymbol{W'}) \leq R$, where R is the target rank.

For our experiments, we use a synthetic (2048×2048) weight matrix where each element is randomly generated from the Gaussian distribution $\mathcal{N}(\mu = 0, \sigma^2 = 1)$. Then, we are interested in the amount of change of each weight after quantization and SVD. Assuming that weight noise through compression is expressed as ϵ in the form of $w' = w(1 + \epsilon)$, Figure 1 shows the distribution of ϵ with various quantization bits or target ranks. From ϵ distributions skewed to be negative, it is clear that weights tend to decay more with higher compression ratio, along with a wider range of random noise. Reasonable explanations of Figure 1 would include: 1) weights generated from the Gaussian distribution are uncorrelated such that an approximation step (by compression) using multiple weights would result in noise for each weight, 2) in the case of SVD, elements associated with small eigenvalues are eliminated, 3) averaging effects in quantization reduce the magnitude of large weights. For weight pruning, ϵ becomes -1 or 0 (i.e., weight decay for selected weights). Correspondingly, we study on weight decay and weight noise insertion in the next two sections as an effort to gain a part of basic knowledge on improved training for model compression, even though actual model compression would demand much more complicated weight noise models.

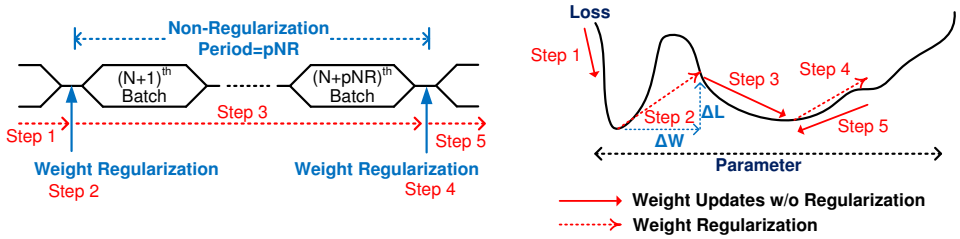


Figure 2: Gradient descent and weight regularization when NR period is given as a multiple of batches. Depending on the loss surface and/or strength of regularization, regularization would lead to step 2 (escaping from a local minimum) or step 5 (returning to a local minimum).

3 Non-Regularization Period

Since weight updates for regularization cannot precede updates gradient descent, in order to control the frequency of weight regularization, an available option is to skip a few batches without regularization. In this paper, we propose a new hyper-parameter, called "Non-Regularization period" or NR period, to enable asynchronous regularization and to define the interval of two consecutive regularization events as shown in Figure 2. NR period is expressed as a multiple of batches.

Large amount of weight updates facilitate the chance of escaping a local minimum (depicted as step 2 in Figure 2) or require longer NR period to return to a local minimum (described as step 5 in Figure 2). Let us estimate NR period (pNR) for a returning case. Given a parameter set $\boldsymbol{w}$ (that is assumed to be close enough to a local minimum) and a learning rate γ, the loss function of a model $\mathcal{L}(\boldsymbol{w})$ can be approximated as

$$\mathcal{L}(\boldsymbol{w}) \simeq \mathcal{L}(\boldsymbol{w}_0) + (\boldsymbol{w} - \boldsymbol{w}_0)^\top (H(\boldsymbol{w}_0)/2)(\boldsymbol{w} - \boldsymbol{w}_0) \tag{1}$$

using a local quadratic approximation where H is the Hessian of $\mathcal{L}$ and $\boldsymbol{w}_0$ is a set of parameters at a local minimum. After regularization is performed at step t, $\boldsymbol{w}$ can be updated by gradient descent as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}\Big|_{\boldsymbol{w}=\boldsymbol{w}_t} \simeq \boldsymbol{w}_t - \gamma H(\boldsymbol{w}_0)(\boldsymbol{w}_t - \boldsymbol{w}_0). \tag{2}$$

Thus, after pNR, we obtain $\boldsymbol{w}_{t+pNR} = \boldsymbol{w}_0 + (I - \gamma H(\boldsymbol{w}_0))^{pNR} (\boldsymbol{w}_t - \boldsymbol{w}_0)$, where I is an identity matrix. Suppose that H is positive semi-definite and all elements of $I - \gamma H(\boldsymbol{w}_0)$ are less than 1.0, $\boldsymbol{w}_{t+pNR}$ can converge to $\boldsymbol{w}_0$ with long pNR which should be longer with larger $(\boldsymbol{w}_t - \boldsymbol{w}_0)$ (i.e., larger weight update for regularization) or smaller $\gamma H(\boldsymbol{w}_0)$. In the next sections, we focus on the relationship between pNR and the strength of weight regularization.

4 NR Period Study on Weight Decay and Weight Noise Insertion

Weight decay is one of the most well-known regularization techniques (Zhang et al., 2018) and different from L_2 regularization in a sense that weight decay is separated from the loss function calculation (Loshchilov & Hutter, 2017). Weight decay is performed as

$$\boldsymbol{w}_{t+1} = (1 - \gamma\theta \boldsymbol{w}_t) - \gamma \nabla_{\boldsymbol{w}_t} \mathcal{L}(\boldsymbol{w}), \tag{3}$$

where θ is a constant weight decay factor. Weight noise insertion is another regularization technique aiminig at reaching flat minima (Goodfellow et al., 2016; Hochreiter & Schmidhuber, 1995). Suppose that random Gaussian noise is added to weights such that $\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{\epsilon}$ when $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \eta I)$. Then, $\mathcal{L}(\boldsymbol{w}') = \mathbb{E}[f_{\boldsymbol{w}+\boldsymbol{\epsilon}}(\boldsymbol{x}) - \boldsymbol{y}]^2$ where $\boldsymbol{x}$, $\boldsymbol{y}$, f are input, target, and prediction function, respectively. Using Taylor-series expansion to second-order terms, we obtain $f_{\boldsymbol{w}+\boldsymbol{\epsilon}}(\boldsymbol{x}) \approx f_{\boldsymbol{w}}(\boldsymbol{x}) + \boldsymbol{\epsilon}^\top \nabla f(\boldsymbol{x}) + \boldsymbol{\epsilon}^\top \nabla^2 f(\boldsymbol{x}) \boldsymbol{\epsilon}/2$. Correspondingly, the loss function can also be approximated as

$$\mathcal{L}(\boldsymbol{w} + \boldsymbol{\epsilon}) \approx \mathbb{E}[f_{\boldsymbol{w}}(\boldsymbol{x}) - \boldsymbol{y}]^2 + \eta \mathbb{E}[(f_{\boldsymbol{w}}(\boldsymbol{x}) - \boldsymbol{y}) \nabla^2 f_{\boldsymbol{w}}(\boldsymbol{x})] + \eta \mathbb{E}||\nabla f_{\boldsymbol{w}}(\boldsymbol{x})||^2, \tag{4}$$

where the second term disappears near a local minimum and the third term induces flat minima. Random noise insertion with other distribution models can be explained in a similar fashion (Goodfellow et al., 2016).

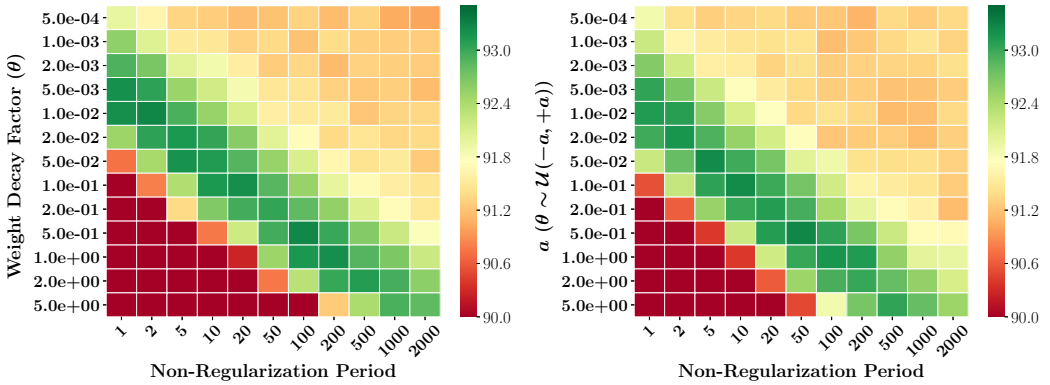


Figure 3: Model accuracy of ResNet-32 on CIFAR-10 using various NR period and amount of weight update for regularization (original model accuracy without regularization is 92.6%). (Left): Weight decay. (Right): Uniform weight noise insertion.

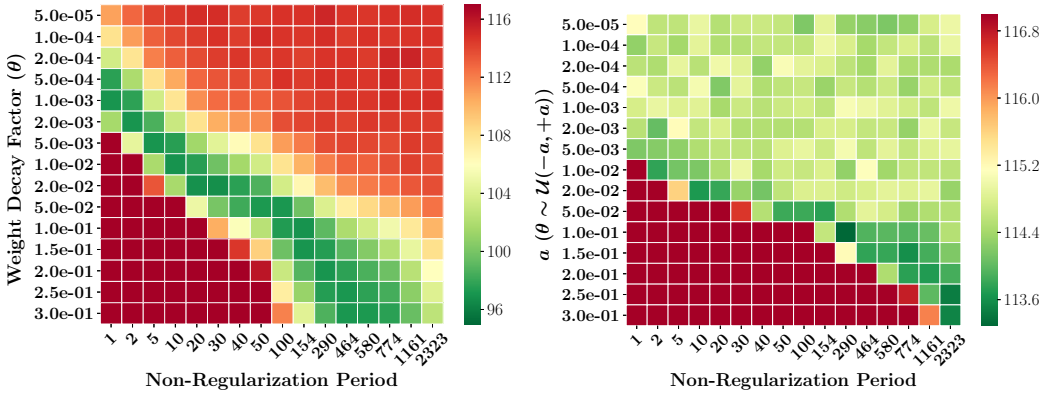
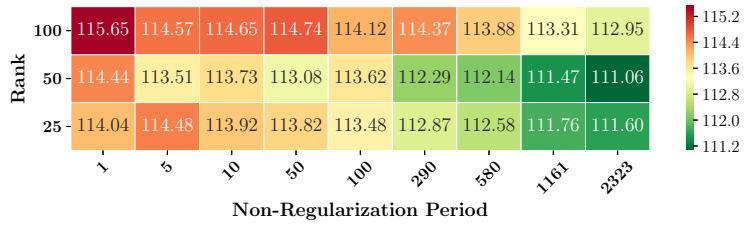


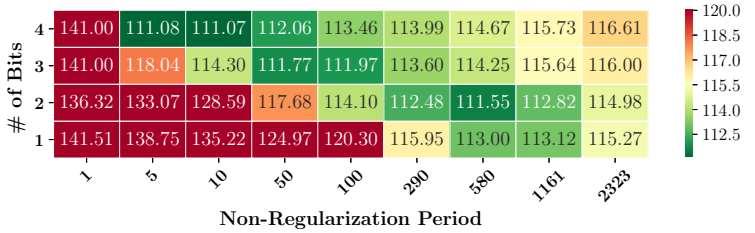
Figure 4: Perplexity of LSTM model on PTB dataset using various NR period and amounts of weight update (original perplexity without regularization is 114.60). (Left): Weight decay. (Right): Uniform weight noise insertion.

We study the impact of NR period on weight decay and weight noise insertion using ResNet-32 on CIFAR-10 model (He et al., 2016) and a long short-term memory (LSTM) model on PTB dataset (Zaremba et al., 2014). For LSTM model, we use 2 layers with 200 hidden units and the hyper-parameter set introduced by Zaremba et al. (2014). For weight noise model, we plug $\theta \sim \mathcal{U}(-a, +a)$ (uniform distribution) into Eq. (3) to simplify the experiments. Figure 3 shows model accuracy of ResNet-32 given different NR period and weight decay factors. For both weight decay and weight noise insertion, the choice of θ (representing the amount of weight updates for each regularization process) has a clear correlation with NR period (refer to Appendix for training and test accuracy graphs). If we wish to apply larger amount weight updates for regularization, then such weight updates should be conducted less frequently (i.e., larger weight decay factor requires longer NR period) to maximize the regularization effect and achieve high model accuracy. Similar observations are discovered by LSTM model on PTB as shown in Figure 4. Lower perplexity (indicating better generalization) is obtained when NR period increases as the amount of weight update becomes larger for each regularization event.

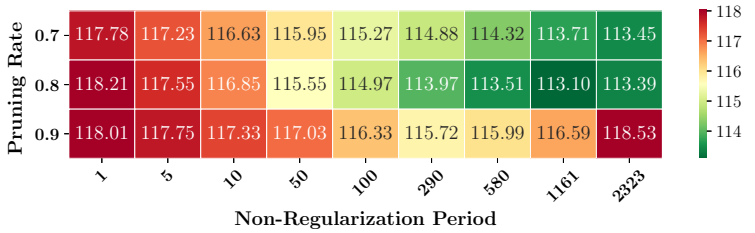
Note that compared with a conventional weight decay factor selection (i.e., θ in Eq. (3) when $pNR = 1$), weight decay factor can be approximately 1,000 times larger with $pNR \approx 1,000$ in Figure 3 and Figure 4. Consequently, we argue that a wide exploration of various NR period (for asynchronous regularization) and different amount of weight updates for regularization is necessary in order to best utilize regularization effects on a DNN model while previous attempts employ $pNR = 1$ only. In addition, *the observation that larger weight update is enabled by longer pNR is our basic training principle for model compression.*



(a) Perplexity when the weights are compressed by SVD.



(b) Perplexity when the weights are compressed by quantization based on binary codes.



(c) Perplexity when the weights are compressed by magnitude-based pruning.

Figure 5: Model accuracy of an LSTM model on PTB compressed by quantization, low-rank approximation or pruning. Original perplexity without model compression is 114.6.

5 NR Period for Model Compression

As discussed, weight compression incurs a much more complicated weight update model than weight decay or uniform weight noise insertion because 1) as shown in Figure 1, diversified noise models need to be combined to describe weight updates after model compression and 2) compression-aware training methods would reduce the amount of weight updates as training is performed with more epochs and weights converge to a compressed form. Nonetheless, we can conjecture that the best training scheme for model compression may require the condition of $pNR \neq 1$ that can be empirically justified.

We apply weight quantization, low-rank approximation (SVD), and pruning to an LSTM model on PTB that we selected for the previous section. We do not modify underlying training principles and use the following simple strategy:

1) Train the model during NR period (as if model compression is not being considered.)
2) Then, perform weight compression in the form of $\boldsymbol{w}' = h(\boldsymbol{w})$.
3) With new full-precision weight $\boldsymbol{w}'$, repeat the above two steps.

$h(\boldsymbol{w})$ can be a magnitude-based pruning (i.e., $h(w)$=w if $|w|$ is larger than a certain threshold, or $h(w)$=0, otherwise), $\alpha \boldsymbol{b}$ for quantization, SVD function, or even as-yet undiscovered functions.

Figure 5 shows model accuracy associated with a number of different sets of pNR and model compression strength (i.e., target rank for low-rank approximation and the number of quantization bits). As compression ratio increases due to lower number of quantization bits or lower rank, pNR is required to be longer to achieve the best model accuracy. Hence, Figure 5 is aligned with the experimental results of weight decay and weight noise insertion with regard to the requirement of decoupling weight regularization from batch size selected for gradient descent.

6 COMPARISON WITH PREVIOUS MODEL COMPRESSION TECHNIQUES

In this section, we compare some of previous model compression techniques with our compression scheme that introduces pNR and obviates special training algorithm modifications.

6.1 FINE-GRAINED WEIGHT PRUNING

The initial attempt of pruning weights was to locate redundant weights by computing the Hessian to calculate the sensitivity of weights to the loss function (LeCun et al., 1990). However, such a technique has not been considered to be practical due to significant computation overhead for computing the Hessian. Magnitude-based pruning (Han et al., 2015) has become popular because one can quickly find redundant weights by simply measuring the magnitude of weights. Since then, numerous researchers have realized higher compression ratio largely by introducing Bayesian inference modeling of weights accompanying supplementary hyper-parameters.

For example, dynamic network surgery (DNS) (Guo et al., 2016) permits weight splicing when a separately stored full-precision weight becomes larger than a certain threshold. Optimizing splicing threshold values, however, necessitates extensive search space exploration, and thus, longer training time. Variational dropout method (Molchanov et al., 2017) introduces an explicit Bayesian inference model for a prior distribution of weights, which also induces various hyper-parameters and increased computational complexity.

We perform magnitude-based pruning at every pNR step. As a result, even though weights are pruned and replaced with zero at pNR steps, pruned weights are still updated in full precision during NR period. If the amount of updates of a pruned weight grows large enough between two consecutive regularization steps, then the weight pruned at last pNR step may not be pruned at the next pNR step. Such a feature (i.e., pruning decisions are not fixed) is also utilized for weight splicing in DNS (Guo et al., 2016). Weight splicing in DNS relies on a hysteresis function (demanding sophisticated fine-tuning process with associated hyper-parameters) to switch pruning decisions. Pruning decisions through our scheme, on the other hand, are newly determined at every pNR step.

We present experimental results with LeNet-5 and LeNet-300-100 models on MNIST dataset which are also reported by Guo et al. (2016); Molchanov et al. (2017). LeNet-5 consists of 2 convolutional layers and 2 fully connected layers while 3 fully connected layers construct LeNet-300-100. We train both models for 20000 steps using Adam optimizer where batch size is 50. All the layers are pruned at the same time and the pruning rate increases gradually following the equation introduced in Zhu & Gupta (2017):

$$p_t = p_f + (p_i - p_f)\left(1 - \frac{t - t_i}{t_f - t_i}\right)^E, \tag{5}$$

where E is a constant, p_f is the target pruning rate, p_i is the initial pruning rate, t is the current step, and the pruning starts at training step t_i and reaches p_f at training step t_f. After t_f steps, pruning rate is maintained to be p_f. For LeNet-5 and LeNet-300-100, t_i, p_i, E are 8000 (step), 25(%), and 7, respectively. t_f is 12000 (step) for LeNet-5 and 13000 (step) for LeNet-300-100. Note that these choices are not highly sensitive to test accuracy as discussed in Zhu & Gupta (2017). We exclude dropout to improve the accuracy of LeNet-300-100 and LeNet-5 since pruning already works as a regularizer (Han et al., 2015; Wan et al., 2013). We keep the original learning schedule and the total number of training steps (no additional training time for model compression).

Table 1 presents the comparison on pruning rates (see Appendix for test accuracy). Despite the simplicity, our pruning scheme produces higher pruning rate compared with DNS and similar compared with variational dropout technique which involves much higher computational complexity. For Table 1, we use pNR=10 for LeNet-5 and pNR=5 for LeNet-300-100.

6.2 LOW-RANK APPROXIMATION

We apply our proposed asynchronous regularization algorithm integrated with Tucker decomposition (Tucker, 1966) to convolutional neural network (CNN) models and demonstrate superiority of pNR-based scheme over conventional training methods. In CNNs, the convolution operation requires a 4D kernel tensor $\mathcal{K} = \mathbb{R}^{d \times d \times S \times T}$ where each kernel has $d \times d$ dimension, S is the input

Table 1: Pruning rate comparison using LeNet-300-100 and LeNet-5 models on MNIST dataset. DC (Deep Compression) and Sparse VD represent a magnitude-based technique (Han et al., 2016) and variational dropout method (Molchanov et al., 2017), respectively.

Model	Layer	Weight Size	Pruning Rate (%)			
			DC	DNS	Sparse VD	Ours
LeNet-300-100	FC1	235.2K	92	98.2	98.9	98.9
	FC2	30K	91	98.2	97.2	96.0
	FC3	1K	74	94.5	62.0	62.0
	Total	266.2K	92	98.2	98.6	98.4
LeNet-5	Conv1	0.5K	34	85.8	67	60.0
	Conv2	25K	88	96.9	98	97.0
	FC1	400K	92	99.3	99.8	99.8
	FC2	5K	81	95.7	95	95.0
	Total	430.5K	92	99.1	99.6	99.5

feature map size, and T is the output feature map size. Then, following the Tucker decomposition algorithm, $\mathcal{K}$ is decomposed into three components as

$$\tilde{\mathcal{K}}_{i,j,s,t} = \sum_{r_s=1}^{R_s} \sum_{r_t=1}^{R_t} \mathcal{C}_{i,j,r_s,r_t} \boldsymbol{P}^S_{s,r_s} \boldsymbol{P}^T_{t,r_t}, \tag{6}$$

where $\mathcal{C}_{i,j,r_s,r_t}$ is the reduced kernel tensor, R_s is the rank for input feature map dimension, R_t is the rank for output feature map dimension, and $\boldsymbol{P}^S$ and $\boldsymbol{P}^T$ are 2D filter matrices to map $\mathcal{C}_{i,j,r_s,r_t}$ to $\mathcal{K}_{i,j,s,t}$. Each component is obtained to minimize the Frobenius norm of $(\tilde{\mathcal{K}}_{i,j,s,t} - \mathcal{K}_{i,j,s,t})$. As a result, one convolution layer is divided into three convolution layers, specifically, (1×1) convolution for $\boldsymbol{P}^S$, $(d \times d)$ convolution for $\mathcal{C}_{i,j,r_s,r_t}$, and (1×1) convolution for $\boldsymbol{P}^T$ (Kim et al., 2016).

In prior tensor decomposition schemes, model training is performed as a fine-tuning procedure after the model is restructured and fixed (Lebedev et al., 2015; Kim et al., 2016). On the other hand, our training algorithm is conducted for Tucker decomposition as follows:

Step 1: Perform normal training for pNR (batches) without considering Tucker decomposition

Step 2: Calculate $\mathcal{C}$, $\boldsymbol{P}^S$, and $\boldsymbol{P}^T$ using Tucker decomposition to obtain $\tilde{\mathcal{K}}$

Step 3: Replace $\mathcal{K}$ with $\tilde{\mathcal{K}}$

Step 4: Go to Step 1 with updated $\mathcal{K}$

After repeating a number of the above steps towards convergence, the entire training process should stop at Step 2, and then the final decomposed structure is extracted for inference. Because the model is not restructured except in the last step, Steps 2 and 3 can be regarded as special steps to encourage wide search space exploration so as to find a compression-friendly local minimum where weight noise by decomposition does not noticeably degrade the loss function.

Using the pre-trained ResNet-32 model with CIFAR-10 dataset (He et al., 2016; Kossaifi et al., 2019), we compare two training methods for Tucker decomposition: 1) typical training with a decomposed model and 2) pNR-based training, which maintains the original model structure and occasionally injects weight noise through decomposition. Using an SGD optimizer, both training methods follow the same learning schedule: learning rate is 0.1 for the first 100 epochs, 0.01 for the next 50 epochs, and 0.001 for the last 50 epochs. Except for the first layer, which is much smaller than the other layers, all convolution layers are compressed by Tucker decomposition with rank R_s and R_t selected to be S and T multiplied by a constant number R_c ($0.3 \leq R_c \leq 0.7$ in this experiment). Then, the compression ratio of a convolution layer is $d^2ST/(SR_s + d^2R_sR_t + TR_t)$ $= d^2ST/(S^2R_c + d^2R_c^2ST + T^2R_c)$, which can be approximated to be $1/R_c^2$ if $S = T$ and $d \gg R_c$. pNR is chosen to be 200.

Figure 6 shows test accuracy after Tucker decomposition[1] by two different training methods. Note that test accuracy results are evaluated only at Step 3 where the training process can stop to generate

[1] https://github.com/larry0123du/Decompose-CNN

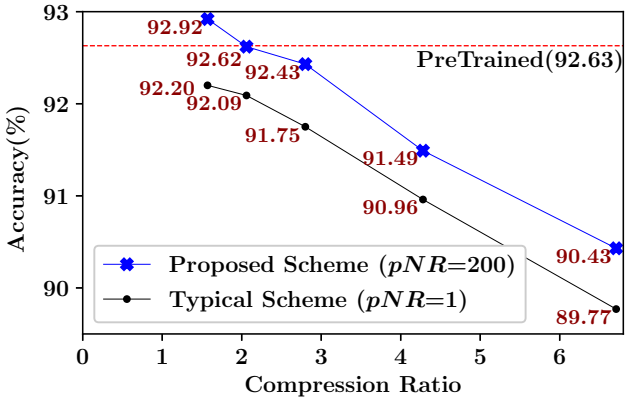


Figure 6: Test accuracy comparison on ResNet-32 using CIFAR-10 trained by typical training method and the proposed training method with various compression ratios. For the proposed scheme, test accuracy is measured only at Step 3 that allows to extract a decomposed structure, and pNR is 200.

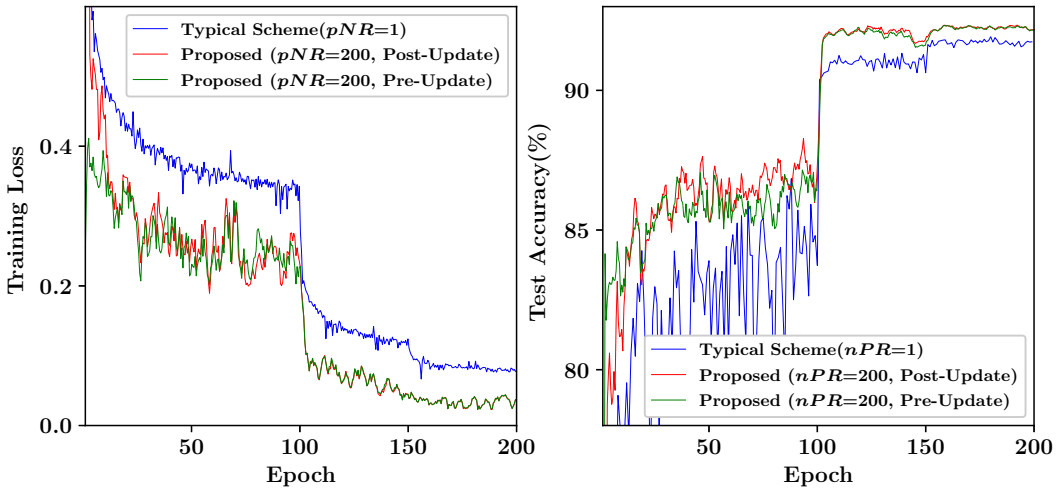


Figure 7: Training loss and test accuracy of ResNet-32 using CIFAR-10. For the proposed scheme, training loss and test accuracy are only monitored right before or after weight update for compression (pre-update or post-update). Compression ratio is 2.8 with R_c=0.5.

a decomposed structure. In Figure 6, across a wide range of compression ratios (determined by R_c), the proposed scheme yields higher model accuracy compared to typical training. Note that even higher model accuracy than that of the pre-trained model can be achieved by our method if the compression ratio is small enough. In fact, Figure 7 shows that our technique improves training loss and test accuracy throughout the entire training process. Initially, the gap of training loss and test accuracy between pre-update and post-update is large. Such a gap, however, is quickly reduced through training epochs. Overall, ResNet-32 converges successfully through the entire training process with lower training loss and higher test accuracy compared with a typical training method.

For ResNet-34 on ImageNet experiments, refer to Appendix.

7 Conclusion

In this paper, we show that weight updates for regularization should be decoupled from batch size for gradient descent. To accomplish such decoupling, we introduce a new hyper-parameter called non-regularization period or NR period during which weights are updated only for gradient compu-

tations. A larger amount of weight decay and weight noise insertion need to be supported by longer NR period to maximize the regularization effect. We find that model compression with higher compression ratio is also better trained by longer NR period. We demonstrate that various DNN models can be compressed by pruning, quantization, and low-rank approximation successfully with longer NR period while the underlying training algorithm do not need to be modified.

REFERENCES

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient primitives for deep learning. *arXiv:1410.0759*, 2014.

Minsik Cho and Daniel Brand. MEC: memory-efficient convolution for deep neural network. In *International Conference on Machine Learning (ICML)*, pp. 815–824, 2017.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

K. Fatahalian, J. Sugerman, and P. Hanrahan. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pp. 133–137, 2004.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems*, 2016.

Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. Network sketching: exploiting binary structure in deep CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4040–4048, 2017.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in Neural Information Processing Systems*, pp. 529–536, 1995.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy,

James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019. URL `http://jmlr.org/papers/v20/18-277.html`.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605, 1990.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Comput. Linguist.*, 19(2):313–330, 1993.

Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, pp. 2498–2507, 2017.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

Twan van Laarhoven. L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*, 2017.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using DropConnect. In *International Conference on Machine Learning (ICML)*, 2013.

Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization, 2018.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. In *International Conference on Learning Representations (ICLR)*, 2017.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *CoRR*, abs/1710.01878, 2017.

A APPENDIX

A.1 SUPPLEMENTARY EXPERIMENTS FOR WEIGHT DECAY AND WEIGHT NOISE

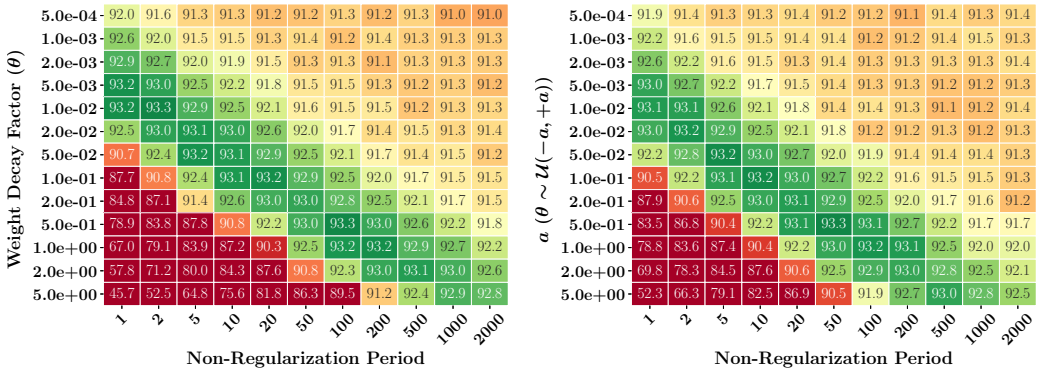


Figure 8: Model accuracy of ResNet-32 on CIFAR-10 using various NR period and amount of weight update for regularization (original model accuracy without regularization is 92.6%). (Left): Weight decay. (Right): Uniform weight noise insertion.

Table 2: Model accuracy of ResNet-32 on CIFAR-10 and LSTM model on PTB with various weight decay factor and corresponding pNR.

Model		Weight Decay Factor(θ)						
		0	1e-4	5e-4	1e-3	5e-3	1e-2	5e-2
ResNet-32	Accuracy(%)	92.6	93.3	93.2	93.2	93.3	93.2	92.9
	optimal pNR	N/A	2	5	20	100	200	1000
LSTM on PTB	Perplexity	114.6	108.1	97.7	97.1	97.1	97.0	97.2
	optimal pNR	N/A	1	1	1	5	10	100

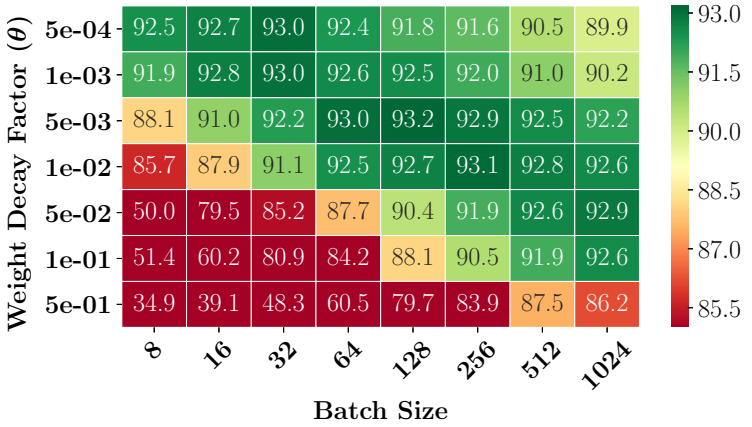
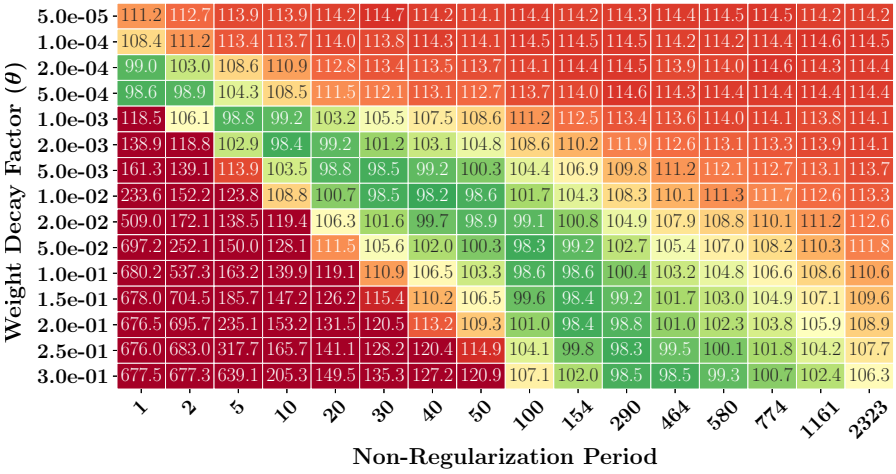
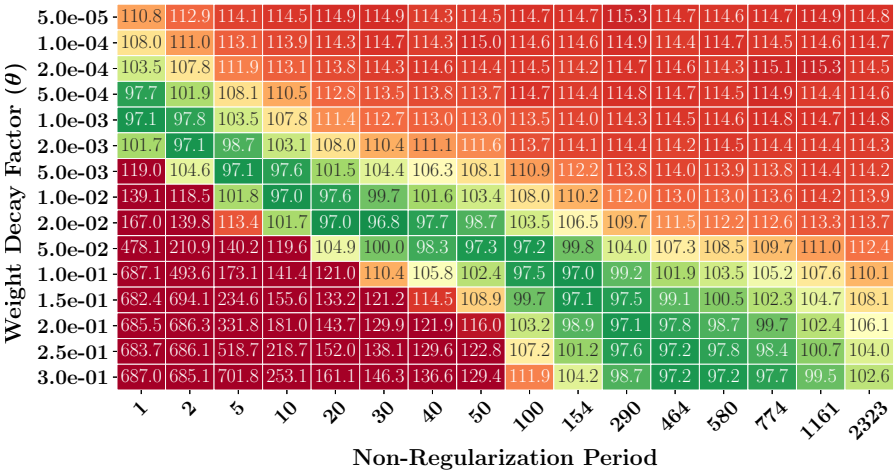


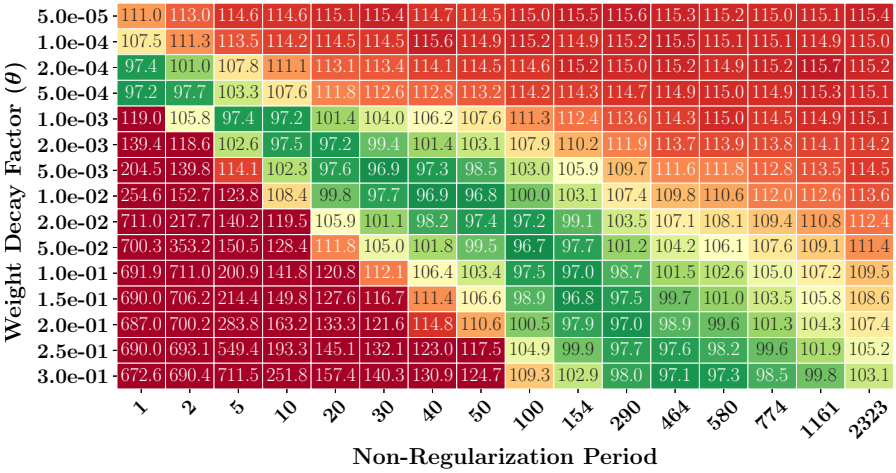
Figure 9: Model accuracy (%) of ResNet-32 for various weight decay factors and batch size when pNR=1. Large batch size demands larger weight decay factors that is also reported by Loshchilov & Hutter (2017).



(a) Initial learning rate = 1.0

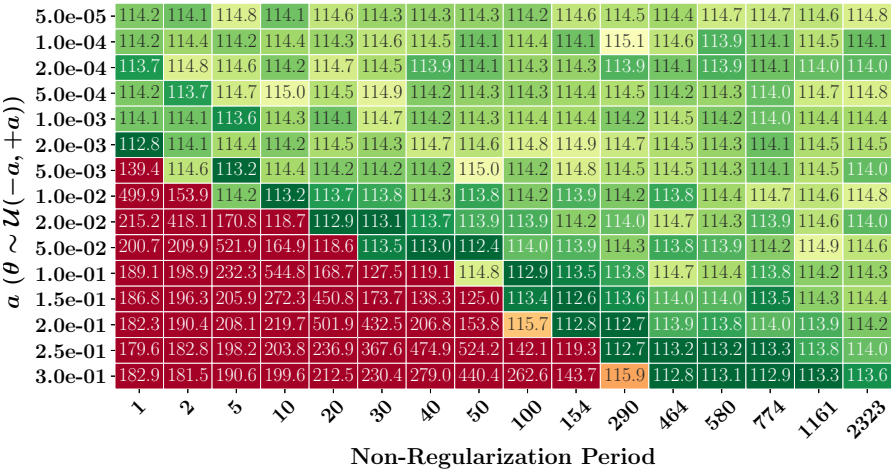


(b) Initial learning rate = 1.5

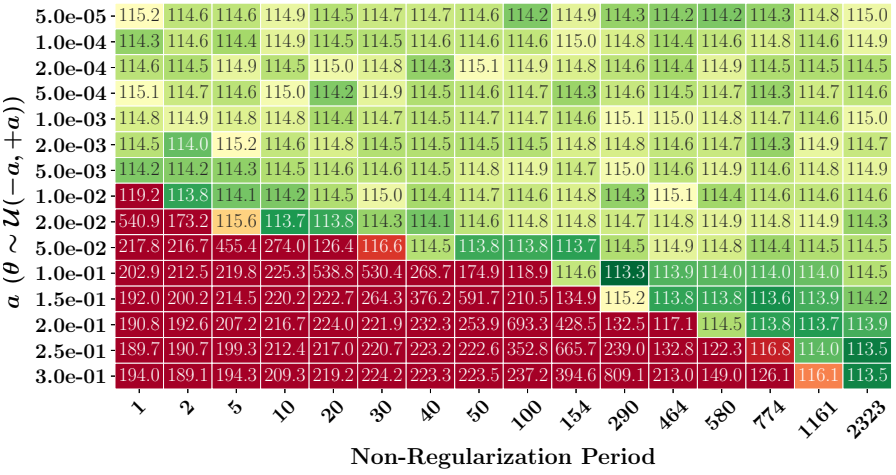


(c) Initial learning rate = 2.0

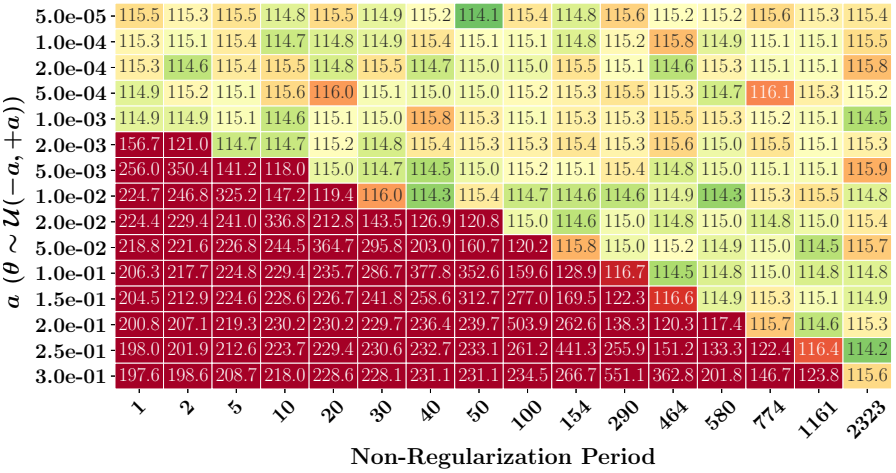
Figure 10: Perplexity of LSTM model on PTB dataset using various NR period and amounts of weight decay (original perplexity without regularization is 114.60).



(a) Initial learning rate = 1.0



(b) Initial learning rate = 1.5



(c) Initial learning rate = 2.0

Figure 11: Perplexity of LSTM model on PTB dataset using various NR period and amounts of uniform noise (original perplexity without regularization is 114.60).

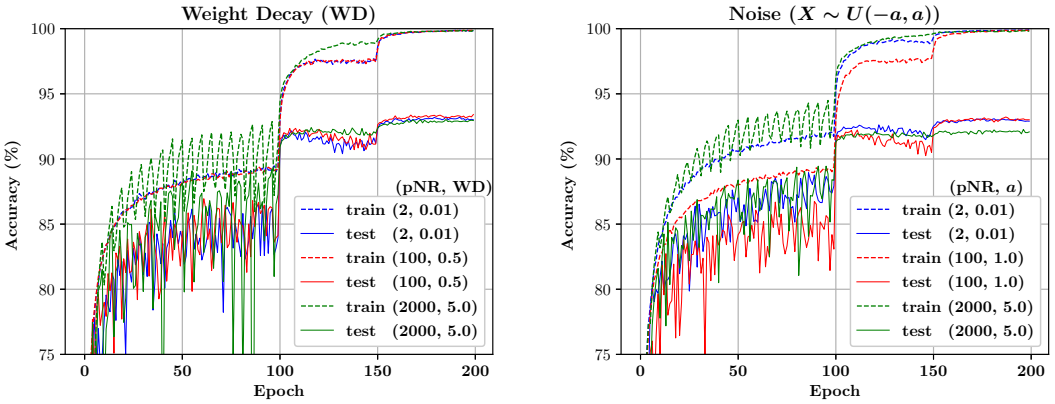


Figure 12: Training and test accuracy of ResNet-32 on CIFAR-10 for variuos NR period and the amount of weight updates for regularization. (Left): Weight decay. (Right): Uniform weight noise.

A.2 Parameter Pruning

Table 3: Model accuracy comparison using LeNet-300-100 and LeNet-5 models on MNIST dataset. DC (Deep Compression) and Sparse VD represent a magnitude-based technique (Han et al., 2016) and variational dropout method (Molchanov et al., 2017), respectively.

Model	Accuracy (%)			
	DC	DNS	Sparse VD	Ours
LeNet-300-100	98.4	98.0	98.1	98.1
LeNet-5	99.2	99.1	99.2	99.1

We apply pNR-based pruning to an RNN model to verify the effectiveness of pNR. We choose an LSTM model (Zaremba et al., 2014) on the PTB dataset (Marcus et al., 1993). Following the model structure given in Zaremba et al. (2014), our model consists of an embedding layer, 2 LSTM layers, and a softmax layer. The number of LSTM units in a layer can be 200, 650, or 1500, depending on the model configurations (referred as small, medium, and large model, respectively). The accuracy is measured by Perplexity Per Word (PPW), denoted simply by perplexity in this paper. We apply gradual pruning with $E = 3$, $t_i = 0$, $p_i = 0$, $t_f = 3^{rd}$ epoch (for medium) or 5^{th} epoch (for large) to the pre-trained PTB models. pNR-based pruning for the PTB models is performed using $pNR = 100$ and the initial learning rate is 2.0 for the medium model (1.0 for pre-training) and 1.0 for the large model (1.0 for pre-training) while the learning policy remains to be the same as in Zaremba et al. (2014).

Table 4: Comparison on perplexity using various pruning rates. p_f is the target pruning rates for the embedded layer, LSTM layer, and softmax layer.

Model Size	Pruning Method	Perplexity					
	p_f=	0%	80%	85%	90%	95%	97.5%
Medium (19.8M)	(Zhu & Gupta, 2017)	83.37	83.87	85.17	87.86	96.30	113.6
	Proposed Scheme	83.78	81.54	82.62	84.64	93.39	110.4
Large (66M)	(Zhu & Gupta, 2017)	78.45	77.52	78.31	80.24	87.83	103.20
	Proposed Scheme	78.07	77.39	77.73	78.28	84.69	99.69

For all of the pruning rates selected, Table 4 shows that our compression scheme improves perplexity better than the technique in Zhu & Gupta (2017) which is based on Han et al. (2015). The superiority of pNR-based pruning is partly supported by the observation that non-zero weights successfully avoid to be small through retraining while the conventional pruning still keeps near-zero (unmasked) weights as depicted in Figure 13.

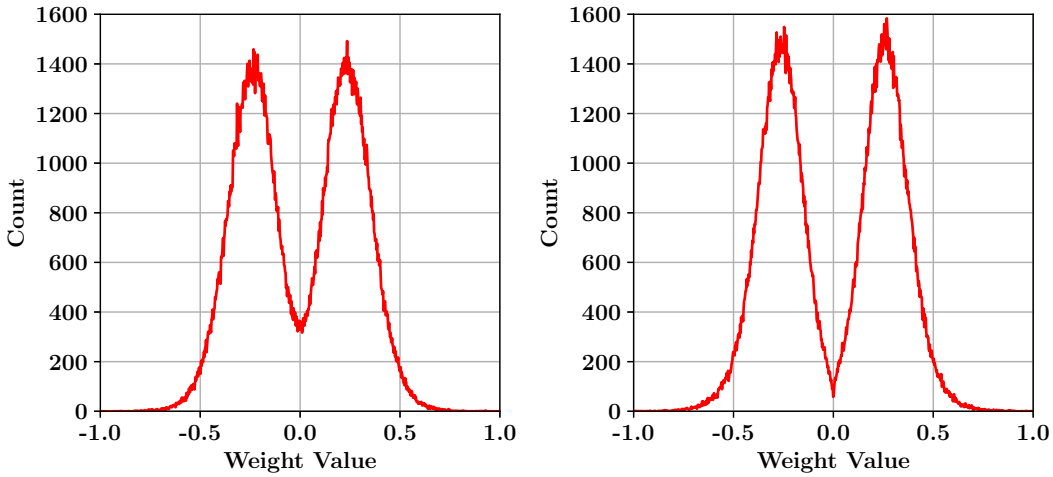


Figure 13: Weight distribution of LSTM layer 1 of the medium PTB model after retraining with (Left) a magnitude-based pruning and (Right) pNR-based pruning with 90% pruning rate. Our compression scheme incurs a sharp drop in the count of near-zero weights.

A.3 Tucker Decomposition with Asynchronous Regularization

To investigate the effect of NR period on local minima exploration with ResNet-32 on CIFAR-10, Figure 14 presents the changes of loss function and weight magnitude values incurred by asynchronous regularization. In Figure 14(left), $\Delta\mathcal{L}/\mathcal{L}$ is given as the loss function increase $\Delta\mathcal{L}$ (due to weight regularization at pNR steps) divided by $\mathcal{L}$, which is the loss function value right before weight regularization. In Figure 14(right), $\Delta\boldsymbol{w}$ is defined as $||\boldsymbol{w} - \tilde{\boldsymbol{w}}||_{\mathcal{F}}^2 / N(\boldsymbol{w})$, where $\boldsymbol{w}$ is the entire set of weights to be compressed, $\tilde{\boldsymbol{w}}$ is the set of weights regularized by Tucker decomposition, $N(\boldsymbol{w})$ is the number of elements of $\boldsymbol{w}$, and $||\boldsymbol{X}||_{\mathcal{F}}^2$ is the Frobenius norm of $\boldsymbol{X}$. Initially, $\boldsymbol{w}$ fluctuates with large corresponding $\Delta\mathcal{L}$. Then, both $\Delta\mathcal{L}$ and $\Delta\boldsymbol{w}$ decrease and Figure 14 shows that asynchronous regularization finds flatter local minima (in the view of Tucker decomposition) successfully. When the learning rate is reduced at 100th and 150th epochs, $\Delta\mathcal{L}$ and $\Delta\boldsymbol{w}$ decrease significantly because of a lot reduced local minima exploration space. In other words, asynchronous regularization helps an optimizer to detect a local minimum where Tucker decomposition does not alter the loss function value noticeably.

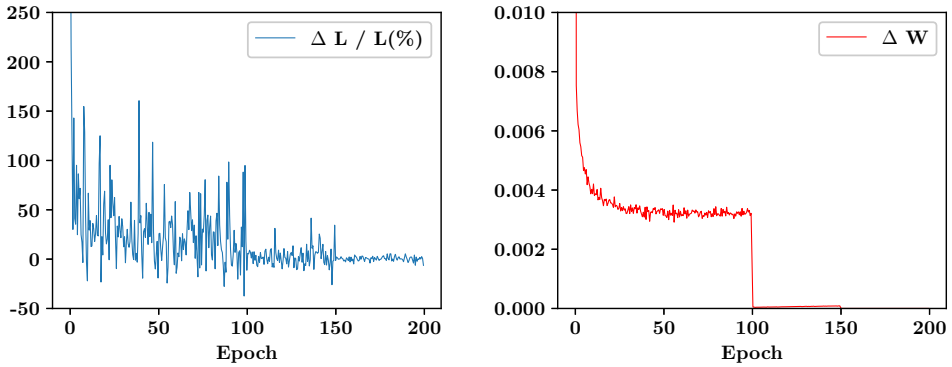


Figure 14: Difference of training loss function and average Frobenius norm of weight values by Step 2 and Step 3 of Figure 2. $R_c = 0.5$ and $pNR = 200$ are used.

A.4 2-DIMENSIONAL SVD ENABLED BY ASYNCHRONOUS REGULARIZATION

In this subsection, we discuss why 2D SVD needs to be investigated for CNNs and how asynchronous regularization enables a training process for 2D SVD.

A.4.1 ISSUES OF 2D SVD ON CONVOLUTION LAYERS

Convolution can be performed by matrix multiplication if an input matrix is transformed into a Toeplitz matrix with redundancy and a weight kernel is reshaped into a $T \times (S \times d \times d)$ matrix (i.e., a lowered matrix) Chetlur et al. (2014). Then, commodity computing systems (such as CPUs and GPUs) can use libraries such as Basic Linear Algebra Subroutines (BLAS) without dedicated hardware resources for convolution Cho & Brand (2017). Some recently developed DNN accelerators, such as Google's Tensor Processing Unit (TPU) Jouppi et al. (2017), are also focused on matrix multiplication acceleration (usually with reduced precision).

For BLAS-based CNN inference, reshaping a 4D tensor $\mathcal{K}$ and performing SVD is preferred for low-rank approximation rather than relatively inefficient Tucker decomposition followed by a lowering technique. However, a critical problem with SVD (with a lowered matrix) for convolution layers is that two decomposed matrices by SVD do not present corresponding (decomposed) convolution layers, because of intermediate lowering steps. As a result, fine-tuning methods requiring a structurally modified model for training are not available for convolution layers to be compressed by SVD. On the other hand, asynchronous regularization does not alter the model structure for training. For asynchronous regularization, SVD can be performed as a way to feed noise into a weight kernel $\mathcal{K}$ for every regularization step. Once training stops at a regularization step, the final weight values can be decomposed by SVD and used for inference with reduced memory footprint and computations. In other words, asynchronous regularization enables SVD-aware training for CNNs.

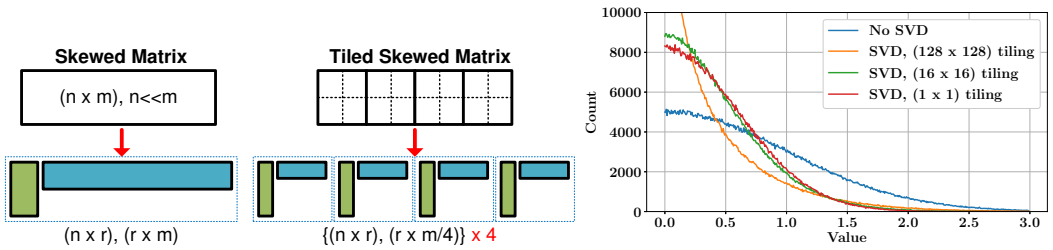


Figure 15: Skewed matrix and a tiling technique are illustrated on the left side, while the right side presents distributions of weights after SVD with different tiling schemes (only positive weights are included).

A.4.2 TILING-BASED SVD FOR SKEWED WEIGHT MATRICES

A reshaped kernel matrix $\boldsymbol{K} \in \mathbb{R}^{T \times (S \times d \times d)}$ is usually a skewed matrix where row-wise dimension (n) is smaller than column-wise dimension (m) as shown in Figure 15 (i.e., $n \ll m$). A range of available rank r for SVD, then, is constrained by small n and the compression ratio is approximated to be n/r. If such a skewed matrix is divided into four tiles as shown in Figure 15 and the four tiles do not share much common chateracteristics, then tiling-based SVD can be a better approximator and rank r can be further reduced without increasing approximation error. Moreover, fast matrix multiplication is usually implemented by a tiling technique in hardware to improve the weight reuse rate Fatahalian et al. (2004). Hence, tiling could be a natural choice not only for high-quality SVD but also for high-performance hardware operations.

To investigate the impact of tiling on weight distributions after SVD, we tested a (1024×1024) random weight matrix in which elements follow a Gaussian distribution. A weight matrix is divided by (1×1), (16×16), or (128×128) tiles (then, each tile is a submatrix of (1024×1024), (64×64), or (8×8) size). Each tile is compressed by SVD to achieve the same overall compression ratio of $4\times$ for all of the three cases. As described in Figure 15 (on the right side), increasing the number of tiles tends to increase the count of near-zero and large weights (i.e., variance of weight values increases). Figure 15 can be explained by sampling theory where decreasing the number of random samples (of

small tile size) increases the variance of sample mean. In short, tiling affects the variance of weights after SVD (while the impact of such variance on model accuracy should be empirically studied).

Table 5: Test accuracy(%) of ResNet-32 model using CIFAR-10 dataset while the 9 largest convolution layers (T=S=64, d=3) are compressed by SVD using different tiling configurations. For each tile size, rank r is selected to achieve compression ratio of $2\times$ or $4\times$. pNR=200 is used for asynchronous regularization.

Pre-Trained	Compression Ratio	Size of Each Tile			
		64×64	32×32	16×16	8×8
92.63	2×	93.34 (r=16)	93.11 (r=8)	93.01 (r=4)	93.23 (r=2)
	4×	92.94 (r=8)	92.97 (r=4)	93.00 (r=2)	92.81 (r=1)

We applied the tiling technique and SVD to the 9 largest convolution layers of ResNet-32 using the CIFAR-10 dataset. Weights of selected layers are reshaped into $64 \times (64 \times 3 \times 3)$ matrices with the tiling configurations described in Table 5. We perform training with the same learning schedule and pNR(=200) used in Section 3. Compared to the test accuracy of the pre-trained model (=92.63%), all of the compressed models in Table 5 achieves higher model accuracy due to the regularization effect of our compression scheme. Note that for each target compression ratio, the relationship between tile size and model accuracy is not clear. Hence, various configurations of tile size need to be explored to enhance model accuracy, even though variation of model accuracy for different tile size is small.

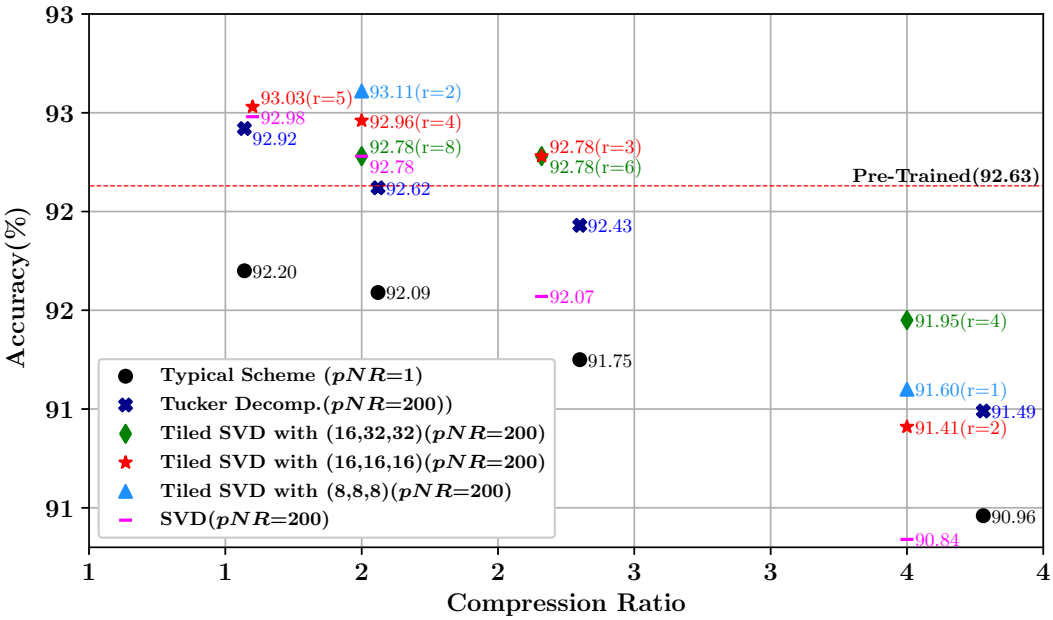


Figure 16: Test accuracy of ResNet-32 model using CIFAR-10 with various target compression ratio and decomposition methods. Except the first small convolution layer, all layers are compressed by the same compression ratio. Convolution layers can be grouped according to 3 different S values (=16, 32, or 64). For tiled SVD, three groups (of different S) are tiled in $(k_1 \times k_1)$, $(k_2 \times k_2)$, or $(k_3 \times k_3)$ tile size. (k_1, k_2, k_3) configuration is described in legends.

A.5 Experimental Results on Low-Rank Approximation for CNNs

In this subsection, we apply low-rank approximation trained by asynchronous regularization to various CNN models.

Figure 16 summarizes the test accuracy values of ResNet-32 (with CIFAR-10 dataset) compressed by various low-rank approximation techniques. Note that tiled SVD and normal SVD are enabled

only by asynchronous regularization, which obviates model structure modification during training. All configurations in Figure 16 use the same learning rate scheduling and the number of training epochs as described in Section 3. Results show that tiled SVD yields the best test accuracy and test accuracy is not highly sensitive to tile configuration. SVD presents competitive model accuracy for small compression ratios. As compression ratio increases, however, model accuracy using SVD significantly degrades. From Figure 16, tiled SVD associated with asynchronous regularization is clearly the best low-rank approximation scheme.

Table 6: Comparison on various low-rank approximation schemes of VGG19 (using CIFAR-10 dataset). To focus on convolution layers only, fully-connected layers are compressed by 8× and trained by asynchronous regularization. Then, fully-connected layers are frozen and convolution layers are compressed (except small layers of $S < 128$) by Tucker decomposition or tiled SVD.

Comp. Scheme	Parameter	Weight Size	FLOPs	Accuracy(%)
Pre-Trained	-	18.98M	647.87M	92.37
Tucker Decomposition (Typical Scheme)	R_c=0.6	9.14M (2.08×)	319.99M (2.02×)	91.97
	R_c=0.5	6.71M (2.83×)	235.74M (2.75×)	91.79
	R_c=0.45	5.49M (3.45×)	191.77M (3.38×)	91.36
	R_c=0.4	4.61M (4.11×)	161.60M (4.01×)	91.11
Tiled SVD (Asynchronous Regularization, pNR=300)	64×64 (r=16)	9.49M (2.00×)	316.28M (2.04×)	92.42
	64×64 (r=11)	6.52M (2.91×)	214.25M (3.02×)	92.33
	64×64 (r=10)	5.93M (3.20×)	193.85M (3.34×)	92.23
	64×64 (r=9)	5.55M (3.41×)	173.44M (3.73×)	92.22
	64×64 (r=8)	4.74M (4.00×)	153.04M (4.33×)	92.07

We compare Tucker decomposition trained by a typical fine-tuning process and tiled SVD trained by asynchronous regularization using the VGG19 model[2] with CIFAR-10. Since this work mainly discusses compression on convolution layers, fully-connected layers of VGG19 are compressed and fixed before compression of convolution layers (refer to Appendix for details on the structure of VGG19). Except for small layers with $S < 128$ (that presents small compression ratio as well), all convolution layers are compressed with the same compression ratio. During 300 epochs to train convolution layers, learning rate is initially 0.01 and is then halved every 50 epochs. In the case of tiled SVD, pNR is 300 and tile size is fixed to be 64×64 (recall that the choice of pNR and tile size do not affect model accuracy significantly). As described in Table 6, while Tucker decomposition with conventional fine-tuning shows degraded model accuracy through various R_c, asynchronous-regularization-assisted tiled SVD presents noticeably higher model accuracy.

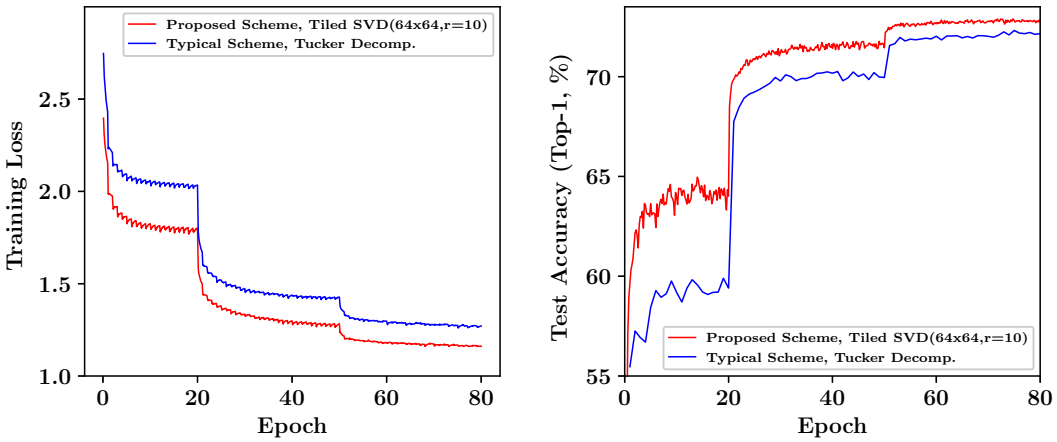


Figure 17: Comparison of two compression schemes on training loss and (top-1) test accuracy of ResNet-34 model using ImageNet. pNR=500.

[2]https://github.com/chengyangfu/pytorch-vgg-cifar10

We also test our proposed low-rank approximation training technique with the ResNet-34 model[3] He et al. (2016) using the ImageNet dataset Russakovsky et al. (2015). A pre-trained ResNet-34 is fine-tuned for Tucker decomposition (with conventional training) or tiled SVD (with asynchronous regularization) using the learning rate of 0.01 for the first 20 epochs, 0.001 for the next 30 epochs, and 0.0001 for the remaining 30 epochs. Similar to our previous experiments, the same compression ratio is applied to all layers except the layers with $S < 128$ (such exceptional layers consist of 1.4% of the entire model). In the case of Tucker decomposition, selected convolution layers are compressed with $R_c = 0.46$ to achieve an overall compression of $3.1\times$. For tiled SVD, lowered matrices are tiled and each tile of (64×64) size is decomposed with r=10 to match an overall compression of $3.1\times$. As shown in Figure 17, asynchronous-regularization-based tiled SVD yields better training loss and test accuracy compared to Tucker decomposition with typical training. At the end of the training epoch in Figure 17, tiled SVD and Tucker decomposition achieves 73.00% and 72.31% for top-1 test accuracy, and 91.12% and 90.73% for top-5 test accuracy, while the pre-trained model shows 73.26% (top-1) and 91.24% (top-5).

A.6 Lowering Technique for CNNs

Figure 18 describes a kernel matrix reshaped from a 4D kernel tensor and an input feature map matrix in the form of a Toeplitz matrix. At the cost of redundant memory usage to create a Toeplitz matrix, lowering enables matrix multiplication which can be efficiently implemented by BLAS libraries. A kernal matrix can be decomposed by 2D SVD.

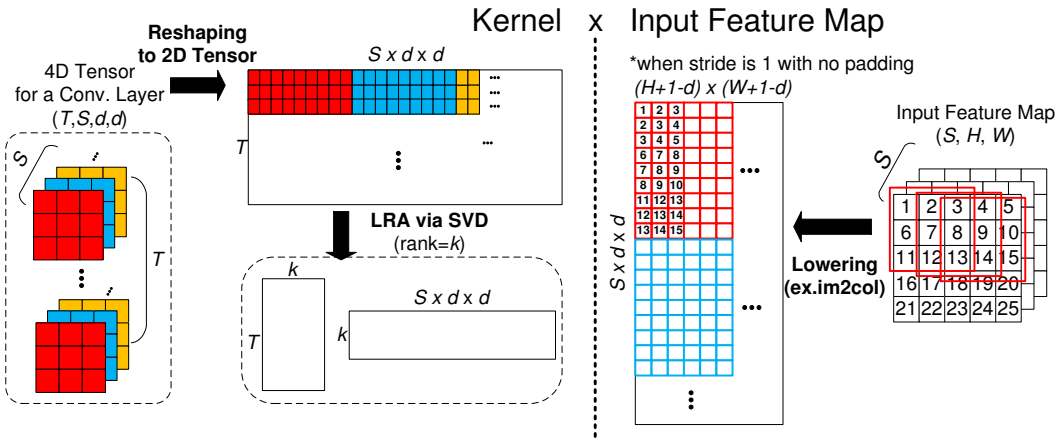


Figure 18: An example of lowering technique using im2col.

A.7 Model Descriptions for Low-Rank Approximation Experiments

In this section, we describe model structures and layers selected for low-rank approximation experiments. Small layers close to the input are not compressed because both weight size and compression rate are too small.

[3]https://pytorch.org/docs/stable/torchvision/models.html

Table 7: Convolution Layers of ResNet-32 for CIFAR-10

# of layers	T	S	d	Weight Size	Decomposed
1	16	3	3	0.4K (0.1%)	No
10	16	16	3	22.5K (5.0%)	Yes
1	32	16	3	4.5K (1.0%)	Yes
9	32	32	3	81.0K (18.0%)	Yes
1	64	32	3	18.0K (4.0%)	Yes
9	64	64	3	324.0K (71.9%)	Yes
Total				450.4K (100.0%)	

Table 8: Convolution and Fully-connected (FC) Layers of VGG-19 for CIFAR-10

Type	# of layers	T	S	d	Weight Size	Decomposed
Conv.	1	64	3	3	0.002M (0.01%)	No
	1	64	64	3	0.035M (0.18%)	No
	1	128	64	3	0.070M (0.36%)	No
	1	128	128	3	0.141M (0.72%)	Yes
	1	256	128	3	0.281M (1.44%)	Yes
	3	256	256	3	1.688M (8.61%)	Yes
	1	512	256	3	1.125M (5.74%)	Yes
	7	512	512	3	15.75M (80.37%)	Yes
FC	2	512	512	-	0.500M (2.55%)	Yes
	1	512	10	-	0.005M (0.02%)	Yes
Total					19.597M (100.0%)	

Table 9: Convolution Layers of ResNet-34 for ImageNet

# of layers	T	S	d	Weight Size	Decomposed
1	64	3	7	0.01M (0.04%)	No
6	64	64	3	0.21M (1.05%)	No
1	128	64	3	0.07M (0.35%)	No
7	128	128	3	0.98M (4.90%)	Yes
1	256	128	3	0.28M (1.40%)	Yes
11	256	256	3	6.18M (30.77%)	Yes
1	512	256	3	1.13M (5.59%)	Yes
5	512	512	3	11.25M (55.94%)	Yes
Total				20.11M (100.0%)	