Training Recipe for N:M Structured Sparsity with Decaying Pruning Mask

Anonymous authors
Paper under double-blind review

Abstract

Sparsity has become one of the promising methods to compress and accelerate Deep Neural Networks (DNNs). Among different categories of sparsity, structured sparsity has gained more attention due to its efficient execution on modern accelerators. Particularly, N:M sparsity is attractive because there are already hardware accelerator architectures that can leverage certain forms of N:M structured sparsity to yield higher compute-efficiency. While there is a large body of work proposing various recipes for N:M structured sparsity training, compute-efficient training recipes for structured sparsity is rather a less explored territory. In this work, we focus on N:M sparsity and extensively study and evaluate various training recipes for N:M sparsity in terms of the trade-off between model accuracy and compute cost (FLOPs). Building upon this study, we propose two new decay-based pruning methods, namely *"pruning mask decay"* and *"sparse structure decay"*. Our evaluations indicate that these proposed methods consistently deliver state-of-the-art (SOTA) model accuracy, comparable to unstructured sparsity, on a Transformer-based model for a translation task. The increase in the accuracy of the sparse model using the new training recipes comes at the cost of marginal increase in the total training compute (FLOPs)[1].

1 Introduction

Deep Neural Networks (DNNs) have shown success in many domains such as computer vision, language modeling, and machine translation. A trend of SOTA DNN models is that the model size increases quickly with time. For example T5 from Google (Raffel et al., 2019), OPT from Meta (Zhang et al., 2022) and GPT-3 from OpenAI (Brown et al., 2020) have over 100 billions parameters, making them hard to be deployed and inaccessible for many practitioners with limited compute resources. Another line of effort in the DNN community is to propose different methods to compress the models, such as quantization (Shen et al., 2020; Kim et al., 2021; Zafrir et al., 2019; Zhang et al., 2020), sparsification (Evci et al., 2019; Han et al., 2015a; Guo et al., 2016; He et al., 2017; Molchanov et al., 2016; Yao et al., 2019; Zhu & Gupta, 2017; Gamboa et al., 2020; Narang et al., 2017a;b; Elsen et al., 2020; Park et al., 2018; Kalchbrenner et al., 2018; Evci et al., 2020a), and distillation (Sanh et al., 2019; Jiao et al., 2019; Sun et al., 2020; Wang et al., 2020).

In this paper, we focus on sparsification (or pruning), which prunes a portion of the parameters in the model by forcefully setting their values to 0. The benefits of such sparsification is multi-fold: (1) reducing the amount of compute by skipping multiplications with 0, (2) reducing the memory usage by using compressed sparse representations (Qin et al., 2021), and (3) save energy/power by reducing memory accesses and computations. In addition, it unlocks the possibility of deploying large models in resource-limited devices. However, sparsification is often about trading-off between model quality[2] and compression ratio. For example, many studies show promising results in sparsifying image classification models to around 90%-95% sparsity (5%-10% density) without quality loss (Guo et al., 2016; Han et al., 2015b). With the success of Transformers in natural language processing, there is rising interest in investigating sparsification in Transformer models, where around 80%-90% sparsity can be achieved. Sparsification in language models has huge potential benefits

---

[1]Anonymized code at https://anonymous.4open.science/r/n_m_decay_1255

[2]In this paper, we refer to algorithmic-wise criteria such as accuracy, recall, and precision as *model quality*; we refer to model runtime/latency as *model performance*.

especially in encoder-decoder tasks such as translation. Since decoder needs to be run iteratively for all N tokens in a sequence, even minor performance improvements in the decoder can improve performance significantly. As such, in this paper, we demonstrate our method with an encoder-decoder Transformer-based translation model.

While sparsification can effectively reduce the memory requirement, generally leveraging induced (unstructured) sparsity in the model for higher performance improvements is challenging. The irregularity of the sparsity pattern makes it challenging to be effectively leveraged by the nateively-dense accelerators such as GPU and TPU. The sparsified models often ends up with similar or worse performance (because of the extra complexity to compress and decompress the parameters) than their dense counterparts (Nvidia, 2021a; Ma et al., 2019; Renda et al., 2020; He et al., 2017; Lin et al., 2021; Gamboa et al., 2020; Zhu et al., 2019; Wen et al., 2016; Evci et al., 2020b).

To this end, structured sparsity, which regularizes the sparsity pattern such as channel/filter sparsity (Li et al., 2016; Wen et al., 2016; He et al., 2017), or block sparsity (Ma et al., 2019; Tan et al., 2020), have become increasingly popular owing to their hardware-friendly nature. For example the dense accelerator can skip a full channel computation when it is sparsified without any low-level modification. The caveat is structured sparsity also introduces larger quality loss. Recent research (Yao et al., 2019; Kang, 2019) found fine-grained N:M structured sparsity, which keeps N out of consecutive M elements in the the weight tensor, can ameliorate the quality loss. Moreover, with the launch of 2:4 structured-sparse tensor core in GPU Ampere architecture (Nvidia, 2021a) developing sparse training recipes for N:M sparsity has acquired increased interest (Pool & Yu, 2021b; Mishra et al., 2021; Nvidia, 2021b; Zhou et al., 2021).

In this paper, we demonstrate a training recipe for N:M structure sparsity in Transformer-based translation task and propose two techniques. We propose *Structure Decay* an iterative pruning approach tailored for N:M sparsity. We propose *Mask Decay*, which gradually decays the mask from 1, to 0.9, 0.8, ..., to 0, instead of the conventional 1/0 mask. We found these techniques can stabilize the training and achieve better quality and compression rate. We make following contributions:

- We compare *Structure Decay* and *Mask Decay* with the state-of-the-art N:M sparsity training recipes, SR-STE (Zhou et al., 2021). They achieve (geomean) 0.004 and (geomean) 0.006 accuracy improvement over SR-STE, respectively.
- *Mask Decay* enables "structured pruning" to achieve comparable quality and compression rate to "unstructured pruning".

2 RELATED WORK

We primarily focus on weight sparsification in this work. A sparsification recipe includes: 1) pruning criteria, 2) pruning schedule, and 3) sparsity pattern.

Pruning Criteria. Pruning criteria is the criteria to decide which elements to prunes in the weight tensor. Magnitude pruning, which selects the pruning elements by their absolute values, is the most widely used method (Renda et al., 2020; Guo et al., 2016; Lee et al., 2018; Frankle & Carbin, 2018; Gale et al., 2019; Zhu & Gupta, 2017; Han et al., 2015a; Liu et al., 2018; Pool & Yu, 2021a; Mishra et al., 2021). Some other metrics such as gradient-based (Yeom et al., 2021; Evci et al., 2020a), Hessian based (LeCun et al., 1989), connection sensitivity (Lee et al., 2018), and salient-based (Molchanov et al., 2019; Lee et al., 2018) are also used. In this paper, we use magnitude pruning.

Pruning Schedule. There are coarsely four different pruning schedules: *1) Fine-tuning with one-shot pruning* (Figure 1f) (Mishra et al., 2021; Pool & Yu, 2021b; Frankle & Carbin, 2018; Lee et al., 2018), which trains a dense models, prunes the weight with one-shot, and re-trains the model in order to recover the quality loss. *2) Fine-tuning with iterative pruning* (Figure 1g) (Evci et al., 2019; Han et al., 2015a; Guo et al., 2016; He et al., 2017; Molchanov et al., 2016; Yao et al., 2019; Zhu & Gupta, 2017; Gamboa et al., 2020; Narang et al., 2017a;b; Elsen et al., 2020; Park et al., 2018; Kalchbrenner et al., 2018; Evci et al., 2020a), which trains a dense model and then iterates between pruning and re-training. This schemes are usually found to has higher ability to recover the quality loss. *3) From-scratch with learned one-shot pruning pattern* (Figure 1h) (Frankle et al., 2020; Evci et al., 2019), which determines the sparsity pattern from the trained dense version and trains a sparse
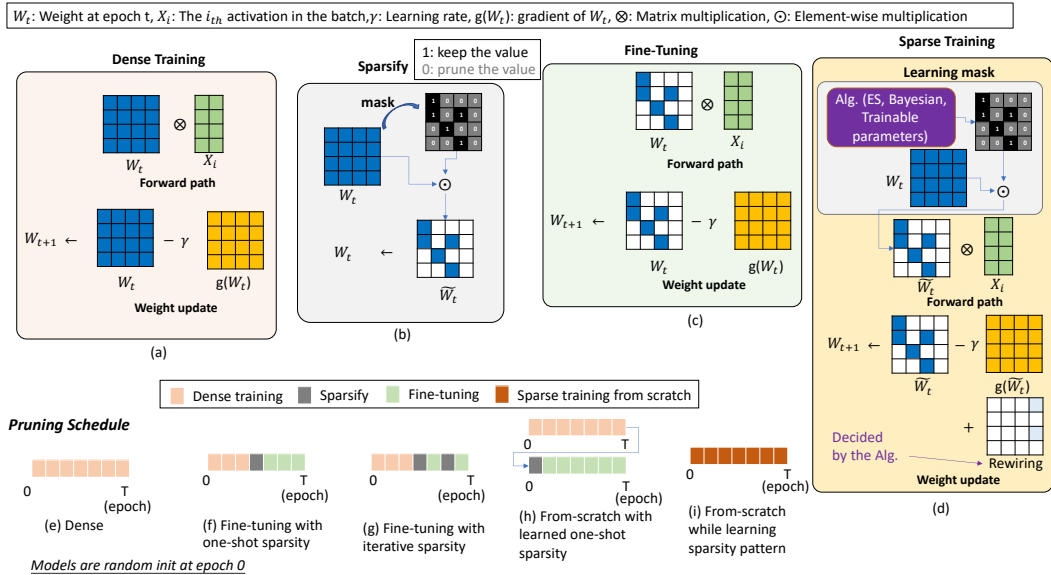
Fig. 1: The compute flows of (a) Dense training, (b) Sparsify, (c) Fine-tuning, and (d) Sparse training. The training schedule of (e) regular dense training, (f) fine-tuning with one-shot sparsifying, (g) fine-tuning with iterative sparsifying, (h) from-scratch with learned one-shot sparsity pattern, and (i) from-scratch while learning sparsity pattern. The sparsify algorithm in (d): ES such as (Mocanu et al., 2018), Bayesian Optimization such as (Bellec et al., 2017), Trainable parameters such as (Wortsman et al., 2019; Dettmers & Zettlemoyer, 2019; Kusupati et al., 2020).

model from scratch. *4) From-scratch while learning sparsity pattern* (Figure 1i) (Wortsman et al., 2019; Dettmers & Zettlemoyer, 2019; Gale et al., 2019; Kusupati et al., 2020; Evci et al., 2020a; Bellec et al., 2017; Mocanu et al., 2018; Zhou et al., 2021; Evci et al., 2020b), which trains a sparse model from scratch while learning sparsity patterns simultaneously.

Sparsity Pattern. *Unstructured Sparsity* prunes the model without any sparsity pattern constraint (Renda et al., 2020; Guo et al., 2016; Lee et al., 2018; Frankle & Carbin, 2018; Gale et al., 2019; Zhu & Gupta, 2017; Han et al., 2015a; Liu et al., 2018; Wortsman et al., 2019; Dettmers & Zettlemoyer, 2019; Gale et al., 2019; Kusupati et al., 2020; Evci et al., 2020a; Bellec et al., 2017; Mocanu et al., 2018). It is often found to be able to prune the model size to an order of magnitude smaller while keeping the model quality. However, it has the challenge of similar or worse (because of the additional complexity) runtime than the dense model owing to its irregular sparsity pattern. *Coarse-grained Structured Sparsity* constrains the pruning scheme to prune the model in a coarse-grained manner such as filter/channel pruning (Li et al., 2016; Wen et al., 2016; He et al., 2017), block-wise pruning (Wen et al., 2016; Ma et al., 2019; Narang et al., 2017b; Gray et al., 2017), and so on. By skipping the full computation at a coarse-granularity of the computation, this scheme can often achieve speedup in dense computation accelerators such as GPUs and TPUs; however this often sacrifices some model quality. These studies often trade off between performance and quality for different application needs. *Fine-grained N:M Structure Sparsity*, which prunes (M-N) out of consecutive M elements. Some early works rely on special threading and grouping techniques (Yao et al., 2019) or specialized sparse accelerators (Kang, 2019) to leverage this fine-grained pattern. With the 2:4 structured-sparse GEMM support in tensor cores in GPU Ampere architecture (Nvidia, 2021a), many recent works start to investigate in different training recipe for N:M sparsity pattern to leverage the existing accelerators (Pool & Yu, 2021b; Mishra et al., 2021; Nvidia, 2021b; Zhou et al., 2021).

Recipe in this paper and the SOTA SR-STE (Zhou et al., 2021). In this paper, our sparse training recipe is (Pruning criteria: magnitude pruning, Pruning schedule: fine-tuning with iterative pruning, Sparsity pattern: fine-grained N:M structure sparsity). The recipe for SR-STE (Zhou et al., 2021) is (Pruning criteria: magnitude pruning, Pruning schedule: from-scratch with learning iterative pruning, Sparsity pattern: fine-grained N:M structure sparsity). Our methods, "Mask Decay" and "Structure Decay", are techniques to improve the training quality of "fine-tuning with iterative
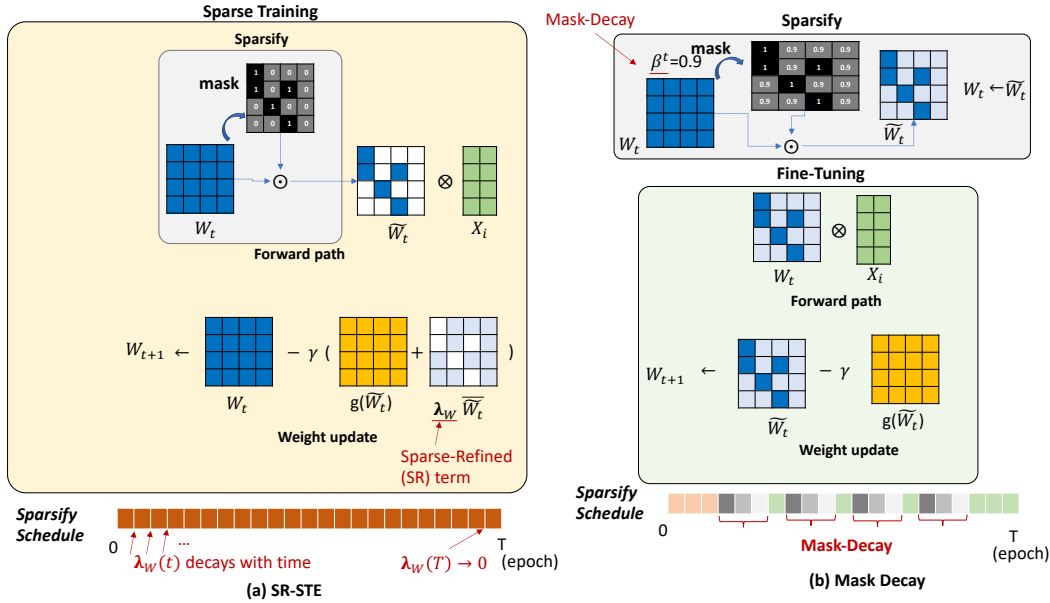
Fig. 2: The weight update scheme of (a) SR-STE (Zhou et al., 2021) and (b) Mask-Decay.

Table 1: The compute and memory contributions of the three major layers in Transformers. **Einsum:** computation of attention scores and weighted sum of values by the attention scores. **Projections:** projecting inputs to key, query, and value, and projection weighted sum of values to outputs. **Feed Forward:** The multiple feed forward layers at the end of the attention layer. The other layers/ operations such as ReLU, LayerNorm, Add, Softmax, embedding, and so on have little contributions to the FLOPS and parameters of the Transformers, hence not included in this estimation. The feed forward layers account for around 64% of overall FLOPs and 67% of parameters. These estimations are made under model configuration in **Table 2**.

	Einsum	Projections	Feed Forward	Einsum	Projections	Feed Forward
(T)FLOPS	1.6	13.2	26.4	4%	32%	64%
Params (MB)	0.0	50.3	100.7	0%	33%	67%

pruning". SR-STE (Zhou et al., 2021) is a "Sparse-Refined (SR)" technique to stabilize the training of "from-scratch with learning iterative pruning". Both SR-STE (Zhou et al., 2021) and us are proposing techniques to pursue high quality sparsification for fine-grained N:M structure sparsity pattern.

3 METHODOLOGY

Workload and model. We evaluate different sparsification methods on the WMT translate task (wmt, 2017) that uses a encoder-decoder Transformer-based model (Vaswani et al., 2017), and is a key benchmark in machine translation research. They hold several translation datasets across different languages. The encoder and decoder blocks in this model each have six attention layers with 16 heads. The embedding dimension for both input and query/key/value are 1024. The feed-forward blocks within each attention head has 4096 neurons. For all the experiments, we *only* induce sparsity in the feed-forward layers of both encoder and decoder blocks (Table 1 shows that feed-forward layers account for around 64% of FLOPS and 67% of parameters of the entire model. Therefore, we focus on feed-forward layers for sparsification). We follow the standard practice of fine-tuning using the final learning rate used during the original training phase (Liu et al., 2018).

Training details. Table 2 shows the details of training hyperparameters that we use for all the evaluations. For each experiment, we use a TPUv3 with 32 cores.

4

Table 2: Model configurations and hyper-parameters.

number of encoder layers	6
number of decoder layer	6
hidden dimension size	1024
feed forward dimension size	4096
number of head	16
max sequence length	256
training set	WMT-17
testing set	WMT-14
learning rate	0.0625
warmup steps	1000
decay factor	0.5
batch size	512
training steps	200K
Adam optimizer	beta1 = 0.9, beta2 = 0.92

Table 3: The effect of update frequency in SR-STE (Zhou et al., 2021). Raising the "Update Frequency" increases the training time significantly. Hence, we only include the results for "Update Frequency" 100 and 1000.

Accuracy		Update Frequency	
		every 1000 steps	every 100 steps
	1:16	0.709	0.710
Sparsity Target	1:32	0.707	0.707
	1:64	0.706	0.706
	1:128	0.706	0.706

3.1 SPARSIFICATION METHOD BASELINE

Fine-grained N:M sparsity. We follow the proposed method in (Zhou et al., 2021) to induce structured N:M sparsity from scratch, as shown in Figure 2(a). This method employs standard online magnitude-based pruning with an introduced sparse-refined regularization term. This regularization term applies refined gradients for pruned weights during backward pass. The authors use the refined gradient updates to increase the likelihood of pruning the same network weights at each training step, which purportedly leads to a more robust sparse training. While this work proposes to re-evaluate the pruning mask after each training iteration, we find this process time-consuming which significantly slows down the training process on TPU. Therefore, we moderately alter the frequency of updating the pruning mask to 1000 training steps. We ablate the importance of the frequency of updating pruning masks. Our results show that the model accuracy for WMT task is not sensitive to this parameter, as shown in Table 3.

3.2 PROPOSED SPARSIFICATION METHODS

In this section, we propose two sparsification methods that employ a decaying mechanism to gradually induce the target sparsity on the model. Note that, we do not alter the gradient update rule in either of these proposed methods. Instead, we simply employ various gradual update rules to the pruning mask itself.

Pruning mask decay. In the first approach, instead of using a binary pruning mask (e.g. "0" indicates pruning locations), we use a floating-point pruning mask with decaying, as shown in Figure 2(b). At the start of training, we employ an all-ones matrix as the pruning mask that simply indicates no pruning. At the beginning of sparse training phase, we use the same standard online magnitude-based pruning criteria to identify the locations of pruned weights. However, in contrast to prior work in which "0" is used to prune the weights, we use $0 < \beta < 1.0$ for the pruned weights in the pruning mask. We gradually decrease the value of β at different intervals following the formula β^d, where d indicates the decaying iteration index (e.g. β^1, β^2, β^3, ...). After sufficient decaying intervals, we set β to zero to indicate the locations of pruned weights. We postulate that using a non-binary pruning mask enables the gradients of pruned weights to flow through the network leading to a more robust sparse training and better model performance.

Sparse structure decay. In the second proposed sparsification method, we apply a decaying mechanism on the structure of pruning mask, gradually increasing sparsification degree. At the beginning of sparse training phase, we start with M-1:M structured sparsity. As training progresses, we increase the sparsification degree by applying $\frac{M}{2^d} : M$ structured sparsity at different decaying intervals. Similar to previous method, d denotes the decaying iteration index. This method at its crux follows a similar hypothesis as the pruning mask decay. That is, enabling the gradient of pruned weights to flow through the network. However, because we still use a binary pruning mask, the contribution of the gradients of the pruned weights to the network reduces after each decaying interval.

4   EVALUATION

4.1   METHODOLOGY

Task and datasets. We use translation as our target task. We use WMT dataset (En-De) (wmt, 2017). We use WMT-17 as training dataset and WMT-14 as testing dataset, and use testing accuracy for evaluation. We prune the parameters of every feed forward layers in the Transformer-based models. We follow the standard practice of fine-tuning using the final learning rate used during the original training phase (Liu et al., 2018). The model and training configuration is shown in Table 2.

Comparisons. In this study, we compare the effectiveness of different SOTA sparsification methods. All methods train for n steps.

- **Dense:** Dense training without sparsification for n steps (Figure 1(e)).
- **Dense-sparse:** Dense train for d steps, sparsify, and fine-tune for $(n-d)$ steps, as in (Mishra et al., 2021) (Figure 1(f)).
- **Sparse:** SR-STE (Zhou et al., 2021)-based sparse training for n steps (Figure 2(a)).
- **Structure Decay:** Dense train for d steps, structure decay the sparsity pattern for $(n-d-s)$ steps, and fine-tune for s steps (Figure 1(g)). The structure decay is set to decay by the power of 2 (§3.1). For example, when target sparsity pattern is 1:16, we divide $(n-d-s)$ steps to five equal time frame, and the sparsity pattern of each time frame is 15:16, 8:16, 4:16, 2:16, and 1:16, respectively.
- **Mask Decay:** Dense train for d steps, mask decay the sparsity pattern for $(n-d-s)$ steps, and fine-tune for s steps (Figure 2(b)). We use the mask decay rate (β) of 0.9 and mask update period of 1000 steps. In the above experiments, we use n = 200K, d = 20K, s = 20K.

4.2   COMPARING WITH BASELINE

Quality. We compare Structure Decay and Mask Decay, with two baseline Dense-Sparse (Mishra et al., 2021) and Sparse (Zhou et al., 2021) in Table 4. We evaluate the methods on different sparsity targets. Sparsity target is the final sparsity pattern we will achieve after model training. For example, sparsity target of 1:32 means the trained model will have only 1 non-zero parameters every 32 parameters. Table 4 shows that Dense-Sparse performs similarly to Sparse, and Mask Decay achieves the best accuracy across all sparsity targets. Structure Decay performs the second best. More interestingly, "Mask Decay" can help achieve similar or better accuracy than the "unstructured sparsity" ones. *Our results indicate that the "Mask Decay" pruning method on dense layers enables models to be pruned structurally while achieving comparable or even better accuracy to "unstructured pruning".*

Table 4: Comparisons between different sparsification strategies.

Accuracy		Dense		Structure Sparsity				Unstructure Sparsity	
Schedule		Dense	Structure Decay	Mask Decay	Dense-Sparse (Mishra et al., 2021)	Sparse (Zhou et al., 2021)		Dense-Sparse	Unstr Sparse
	1:16	0.747	**0.717**	**0.717**	0.714	0.709		0.714	0.714
Sparsity	1:32	0.747	0.713	**0.714**	0.710	0.707		0.711	0.712
Target	1:64	0.747	0.710	**0.711**	0.708	0.707		0.711	0.711
	1:128	0.747	0.708	**0.711**	0.708	0.707		0.708	0.709

Performance. Note that there are no off-the-shelf accelerators that can support 1:16 or more aggressive sparsity patterns. To demonstrate the potential performance, we build a cost model to estimate the FLOPS and memory sizes. Table 5 shows that after 1:16 sparsification, the model sizes will

reduce by 62% and inference FLOPS will reduce by 60%. More interestingly, since different methods have different sparsification schedules, the averaged training FLOPS across their training time will be different. "Dense-Sparse" is the most straight-forward and light-weighted training schedule in terms of FLOPS. "Sparse" relies on continuous mask update across the full training steps (Figure 2(a)), therefore performing worse in terms of FLOPS. Structure Decay becomes the most performant method to achieve the best quality at 1:16 sparsity. However, to sparsify more aggressively, Mask Decay is still the best method (Table 4).

Table 5: Comparisons of model performance (FLOPS) and quality (accuracy).

Sparsity Target (1:16)	Dense	Structure Decay	Mask Decay	Dense-Sparse	Sparse
Params (MB)	151.0	56.6	56.6	56.6	56.6
Inference TFLOPS	41.2	16.5	16.5	16.5	16.5
Training TFLOPS	123.7	108.0	121.2	**101.4**	123.7
Accuracy	0.75	**0.72**	**0.72**	0.714	0.709

4.3 ABLATION STUDIES

Dense training v.s. Training from scratch for SR-STE. SR-STE uses sparse training from scratch. All the other methods that we evaluated have a dense training phase at the first few steps (epochs). This recipe has been proven to be effective as shown in the previous experiments and many prior works (Mishra et al., 2021; Pool & Yu, 2021b; Frankle & Carbin, 2018; Lee et al., 2018; Evci et al., 2019; Han et al., 2015a; Guo et al., 2016; He et al., 2017; Molchanov et al., 2016; Yao et al., 2019; Zhu & Gupta, 2017; Gamboa et al., 2020; Narang et al., 2017a;b; Elsen et al., 2020; Park et al., 2018; Kalchbrenner et al., 2018; Evci et al., 2020a). Therefore, we experiment on adding a dense training phase at the beginning of SR-STE training, as shown in Table 6. We found that adding few steps of dense training (1.25% - 10% of the total training steps) can increase the accuracy by around 0.002 to 0.003. This tells that few steps of dense training does help achieve better performance even for SR-STE. Interestingly, the improved SR-STE becomes competitive to the proposed Structure Decay. However, Mask Decay is still consistently better.

Table 6: Ablation: SR-STE augmented with few epochs of dense training.

	Training Schedule		Accuracy			
	Dense steps	SR-STE -styled Sparse steps	1:16	1:32	1:64	1:128
SR-STE	0	200K	0.710	0.707	0.706	0.706
Dense + SR-STE	2.5K	197.5K	0.712	0.710	**0.708**	0.706
	5K	195K	0.712	0.709	0.707	**0.708**
	10K	190K	0.712	**0.710**	0.707	**0.708**
	20K	180K	**0.713**	0.710	**0.708**	0.707

Effect of dense training steps (d). Both our proposed methods include a dense training phase. We do an ablation study on different number of dense training steps in Table 7. We found that changing the dense step between 1.25% - 10% of the total training steps does not observably change the accuracy performance. However, empirically, we found that dense training phase is still essential. The model cannot achieve as competitive accuracy without few epochs of dense training.

Table 7: Ablation: The effect of number of dense training steps (d).

Accuracy		Mask Decay				Structure Sparsity			
Sparsity Target		1:16	1:32	1:64	1:128	1:16	1:32	1:64	1:128
Dense steps (d)	2.5 K	0.7155	0.7134	0.7106	0.7100	0.7157	0.7134	0.7108	0.7106
	5 K	**0.7160**	0.7127	**0.7110**	0.7093	0.7160	0.7136	**0.7117**	0.7100
	10 K	0.7157	**0.7137**	0.7103	0.7094	0.7164	**0.7141**	0.7107	0.7098
	20 K	0.7156	0.7126	0.7107	**0.7104**	**0.7165**	0.7128	0.7115	**0.7107**

Effects of fine-tuning steps (s). We also have a sets of study on number of fine-tuning steps in Table 8. We found that for both of our propose methods the fine-tuning steps between 10% - 20% of the total training steps does not observably change the accuracy performance. However, empirically, we also found few steps of fine-tuning at the end is essential to recover the accuracy.

Table 8: Ablation: The effect of number of fine-tuning steps (s).

Accuracy		Mask Decay				Structure Sparsity			
Sparsity Target		1:16	1:32	1:64	1:128	1:16	1:32	1:64	1:128
Fine-tuning steps (s)	20 K	0.7153	0.7130	**0.7107**	**0.7098**	**0.7160**	**0.7125**	**0.7095**	**0.7072**
	40 K	**0.7161**	**0.7132**	0.7106	0.7097	0.7121	0.7093	0.7081	0.7065

Effects of β in Mask Decay. Note that the extreme case of a mask decay rate ($\beta=0$) will turn the pruning mask back to the conventional 1/0 mask. As shown in Table 9, we found a mask decay rate of 0.9 is better than an aggressive one (0.001). It suggests the Mask Decay technique does contribute and lead to better accuracy performance.

Table 9: Ablation: The effect of mask decay rate (β).

Accuracy		Mask Decay			
Sparsity Target		1:16	1:32	1:64	1:128
Mask decay rate (β)	0.9	**0.715**	**0.713**	**0.711**	**0.710**
	0.001	0.712	0.709	0.708	0.707

5 LIMITATIONS

This paper studies only translation task with one sparsification recipe (Criteria: magnitude pruning, Schedule: Structure Decay or Mask Decay, Pattern N:M structured pruning) and only prunes feedforward layers. 1) We show we can effectively prune the most compute- and parameter-heavy layers in our model, feed forward layers. An interesting next-step is to prune other layers such as projections layers as well to further compress the model. 2) Studies on other language tasks or visions tasks and on different models such as on Resnet (He et al., 2016) or ViTs (Dosovitskiy et al., 2020)) would be an interesting follow-up. 3) In addition, we only study one combination of sparsification recipe. We might discover better recipe by exploring other combinations such as salient-based pruning + Mask Decay + N:M structured pruning, magnitude pruning + Structure Decay + unstructured sparsity, or many others. 4) Lastly, in the evaluations, most of the hyper-parameters are set manually. We did a limited scope of hyper-parameters sweep in §4.3. A full-fledged hyper-parameter search might discover more performance improvement in "Mask Decay" and "Structure Decay".

6 CONCLUSION

In this work, we study and evaluate various training recipes for N:M structured sparsity. Building on this study, we propose and compare two new training recipes for N:M structured sparsity based on decaying mechanisms. We study the trade-off between model accuracy and training compute cost (FLOPs) across these training recipes. We show that gradual decay of pruning mask values consistently yield better model accuracy, on-par with unstructured sparsity, on translate task at the cost of modest increase in the training compute cost. While structured sparsity seems to be better positioned for hardware acceleration, its associated training cost should not be overlooked. This work represents a first step in evaluating training recipes for structured sparsity from the perspective of trade-off between model accuracy and compute cost. As future work, we plan to expand the pool of models to other tasks and models and develop a platform to systematically evaluate and compare various sparse training recipes both for model accuracy and training cost.

BIBLIOGRAPHY

Emnlp 2017 second conference on machine translation (wmt17). https://www.statmt.org/wmt17/, 2017.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M.

Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14629–14638, 2020.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020a.

Utku Evci, Yani A Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. *arXiv preprint arXiv:2010.03533*, 2020b.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Noah Gamboa, Kais Kudrolli, Anand Dhoot, and Ardavan Pedram. Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators. *arXiv preprint arXiv:2001.03253*, 2020.

Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3:2, 2017.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pp. 2410–2419. PMLR, 2018.

Hyeong-Ju Kang. Accelerator-aware pruning for convolutional neural networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(7):2093–2103, 2019.

Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*, 2021.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Mingbao Lin, Liujuan Cao, Shaojie Li, Qixiang Ye, Yonghong Tian, Jianzhuang Liu, Qi Tian, and Rongrong Ji. Filter sketch for network pruning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, Xue Lin, Kaisheng Ma, and Yanzhi Wang. Non-structured dnn weight pruning – is it beneficial in any platform?, 2019. URL https://arxiv.org/abs/1907.02124.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017a.

Sharan Narang, Eric Undersander, and Gregory Diamos. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*, 2017b.

Nvidia. NVIDIA Ampere Architecture Whitepaper. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2021a.

Nvidia. Nvidia asp (automatic sparsity). https://github.com/NVIDIA/apex/tree/master/apex/contrib/sparsity, 2021b.

Mi Sun Park, Xiaofan Xu, and Cormac Brick. Squantizer: Simultaneous learning for both sparse and low-precision neural networks. *arXiv preprint arXiv:1812.08301*, 2018.

Jeff Pool and Chong Yu. Channel permutations for n: M sparsity. *Advances in Neural Information Processing Systems*, 34, 2021a.

Jeff Pool and Chong Yu. Channel permutations for n: M sparsity. *Advances in Neural Information Processing Systems*, 34, 2021b.

Eric Qin, Geonhwa Jeong, William Won, Sheng-Chun Kao, Hyoukjun Kwon, Sudarshan Srinivasan, Dipankar Das, Gordon E Moon, Sivasankaran Rajamanickam, and Tushar Krishna. Extending sparse tensor accelerators to support multiple compression formats. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1014–1024. IEEE, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

Zhanhong Tan, Jiebo Song, Xiaolong Ma, Sia-Huat Tan, Hongyang Chen, Yuanqing Miao, Yifu Wu, Shaokai Ye, Yanzhi Wang, Dehui Li, et al. Pcnn: Pattern-based fine-grained regular pruning towards optimizing cnn accelerators. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information*

processing systems, pp. 5998–6008, 2017.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv:2002.10957*, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115:107899, 2021.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: m fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

Maohua Zhu, Tao Zhang, Zhenyu Gu, and Yuan Xie. Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 359–371, 2019.

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