Dynamic Sparse Pre-Training of BERT

Anastasia S. D. Dietrich
Graphcore Research
London, UK
anastasiad@graphcore.ai

Frithjof Gressmann
Graphcore Research
Bristol, UK
frithjof@graphcore.ai

Douglas Orr
Graphcore Research
London, UK
douglaso@graphcore.ai

Ivan Chelombiev
Graphcore Research
London, UK
ivanc@graphcore.ai

Daniel Justus
Graphcore Research
London, UK
danielj@graphcore.ai

Carlo Luschi
Graphcore Research
Bristol, UK
carlo@graphcore.ai

Abstract

Identifying algorithms for compute efficient unsupervised training of large language models is an important and active area of research. In this work, we develop and study a simple, dynamic always-sparse pre-training approach for BERT language models, which leverages periodic compression steps based on magnitude pruning followed by random parameter re-allocation. As a result, we achieve Pareto improvements in terms of number of floating-point operations (FLOPs) over both static and dense baselines across model sizes. Furthermore, we demonstrate that training remains FLOP-efficient when using coarse-grained block sparsity, making it particularly promising for efficient execution on modern hardware accelerators.

1 Introduction

The increasing task performance gains of large, pre-trained language models have fueled interest in approaches for compute efficient unsupervised training [11]. In recent years, sparsity has regained popularity as a technique to make models more computationally efficient by either reducing the number of model parameters via weight sparsity [8, 9, 1, 17, 5, 3, 16, 10, 20], attention head and neuron pruning during the early pre-training phase [2] or dynamically routing activations to only interact with a subset of the network weights via conditional sparsity [19, 12, 7, 13].

In weight sparse training [8, 9], the network representation itself is compressed and reduced in parameter count by imposing sparsity patterns on the network weights. As a result, weight sparse training can lead to significant savings in FLOPs, which make it promising for scaling to larger network architectures for a given compute budget. One of the most promising candidates for weight sparse training is dynamic sparsity (DynSp), which reduces FLOPs while only requiring training of sparse subsets of the over-parameterized network [1, 17, 5, 3, 16, 10, 20]. In DynSp approaches, the sparsity pattern imposed on the weights is continuously modified during training using pruning and re-allocation strategies. This leads to a joint exploration of both network topology and parameters that has been shown to outperform static sparsity baselines [1, 17, 5, 3]. However, so far, DynSp training has not seen adoption for large-scale language modeling tasks [5], despite recent algorithmic improvements in the language domain [10]. Given the high cost and energy consumption of unsupervised training of large-scale language models [21, 18], dynamic sparsity bears the potential to make pre-training much more efficient and affordable.

In this work, we adopt and investigate DynSp training techniques [3, 5] for pre-training of BERT [4], a bidirectional language encoder that is based on the highly scalable Transformer architecture [24]. Specifically, we consider the whole family of BERT models of different sizes [23] that allows us to

Preprint. Under review.

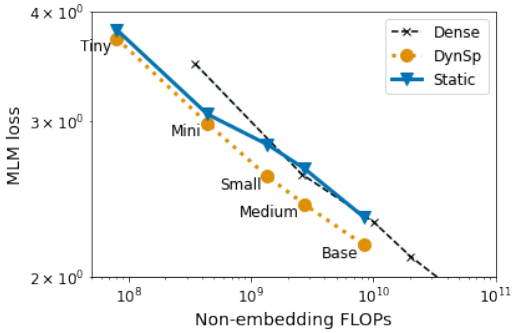

Figure (1) Pareto curve for the BERT family [23], comparing DynSp training (orange dotted line) with static sparsity (solid blue line) and the dense baseline (black dashed line,the standard deviation is not visible at this scale) as a function of FLOPs. All results are obtained for phase I of pre-training with sparsity ratio 0.9, $n = 160$ pattern updates, and optimal pruning ratio $pr = 0.5$ (see Fig. 3).

compare dense and sparse methods for a given FLOPs budget and assess the scalability of the sparse training across a wide range of model sizes. Furthermore, we analyze the importance of trainable parameters by keeping a random pattern of weights non-trainable during training. This allows us to disentangle the role of 'zero' vs. 'untrainable' (fixed) weights in the connectivity patterns, to shed light on the parameter dependence of BERT pre-training.

2 Methodology

Baseline Throughout this work, we study the self-supervised pre-training objective from the original BERT model [4], which consists of the *Masked Language Model* (MLM) loss, corresponding to the task performance in predicting a random subset of masked tokens, and the noisier *Next Sentence Prediction* (NSP) loss for binarized next sentence prediction. Using the Adam optimizer, we focus on phase I of pre-training with a sequence length of 128. All hyperparameters are given in Appendix A and are optimized for a training length of 10 epochs. We compare the sparse task performance on the full BERT-family Pareto curve [23], similar to the *Same Capacity Sparse vs. Dense Comparison* approach introduced by Tessera et al. [22]. This allows us to systematically assess algorithmic differences by comparing DynSp training with dense and static baselines on an equal FLOPs budget.

DynSp algorithm We study and adapt dynamic sparse training algorithms that have been primarily developed for vision architectures [3, 5] to pre-training of the BERT language models. Specifically, we impose sparsity on all fully-connected encoder weights (non-embedding weights). The sparsity pattern is initialized randomly and the ratio is kept fixed in each layer throughout training. Furthermore, we use random re-allocation of pruned weights instead of gradient-based techniques like RigL [5]. For one, this avoids potential issues from a collapse of the explored parameter space (compare Fig. 7). More importantly, it makes our approach always-sparse, such that the full dense model is never actually instantiated. All DynSp hyperparameters are optimized for a sparsity ratio of 0.9 (for more details, refer to Appendix A.1). The dense and sparse BERT-family learning rates are obtained from a grid search for $0.0001 \cdot 2^m$ with $m = 0, 1, ..., 5$ (see Appendix A.3).

3 Results & Discussion

Pareto efficiency We find that the DynSp training algorithm with random re-allocation leads to Pareto improvements compared to the dense BERT-family (see Fig. 3). The Pareto improvements of DynSp training over the dense baseline remain largely independent of the model size, indicating that DynSp training can achieve more efficient utilization of FLOPs or network parameters at any scale. Furthermore, we find that these performance advantages are due to the continued updates of the sparsity pattern, as we do not observe any improvements of the static baseline in FLOPs efficiency of larger models when the randomly initialized sparsity pattern is kept constant. In fact, for large model sizes static sparsity almost perfectly matches the dense baseline. This indicates that the sparse network architecture itself brings no performance advantages. Since we only optimize the DynSp hyperparameters for a sparsity ratio of 0.9, we can not expect the task performance to generalize to other sparsity ratios. However, interestingly, we observe that the task performance improvements indeed hold across a range of sparsity ratios without additional tuning (see Fig. 2). In sum, we find that DynSp training leads to a more efficient utilization of parameters for all model sizes through a more efficient utilization of FLOPs and trainable weights.

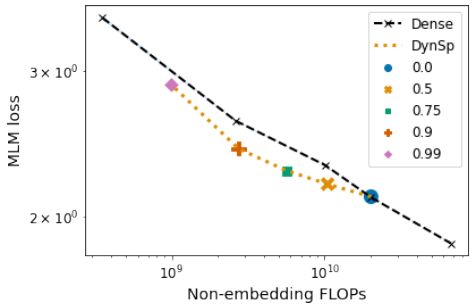

Figure (2) Pareto curve for the BERT family, comparing DynSp training of BERT-Medium with various sparsity ratios (indicated by color and marker style and joint by orange dotted line) with dense training (black dashed line) as a function of non-embedding FLOPs. For all sparsity ratios, we use the hyperparameters optimized for sparsity ratio 0.9.

Model	B	MLM	FLOPs
Small (dense)	-	2.310	$10.07 \cdot 10^9$
Matched (dense)	-	2.350	$8.33 \cdot 10^9$
Base ($s = 0.9$)	16	2.340	$8.33 \cdot 10^9$
Base ($s = 0.9$)	1	**2.176**	$8.33 \cdot 10^9$

Table (1) Task performance of DynSp training of BERT-Base with sparsity $s = 0.9$ for various block sizes B, compared to dense BERT-Small with similar number of FLOPs and linear interpolation of baseline values ("Matched") with exactly the same number of FLOPs. Hyperparameters are not specifically tuned for $B = 16$. See Appendix Table 4 for block size dependence.

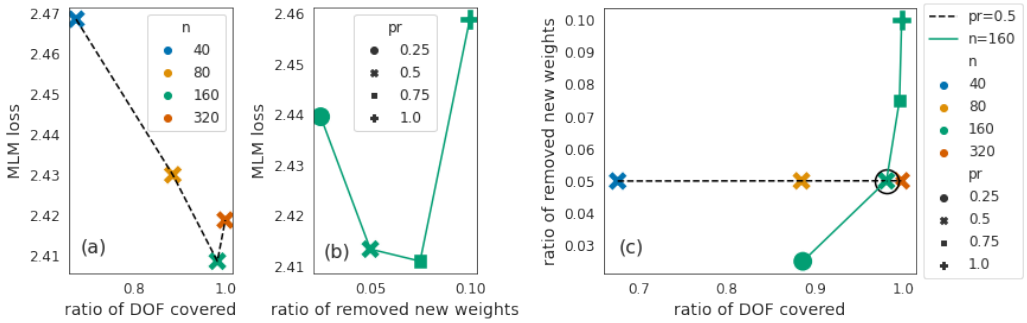

Figure (3) Characterization of the DynSp pre-training of BERT-Medium with sparsity ratio 0.9. All measures shown correspond to maximum value obtained during training and are averaged over all layers. **(a)** MLM loss as a function of the fraction of explored network parameters (DOF) with changing number of sparsity pattern updates n. **(b)** MLM loss as a function of the ratio of removed, new weights with changing pruning ratio pr. **(c)** Joint effect of pruning ratio pr (solid line) on the ratio of removed, new weights, and DOF covered during DynSp training. The best performing values ($n = 160$, $pr = 0.5$) are marked by a circle.

Understanding DynSp training dynamics To improve our understanding of the sparse training dynamics, we extract measures that can help to explain the efficiency of specific hyperparameter choices (see Appendix A.1). Given that the DynSp task performance advantage arises from the continual update of the sparsity pattern, we begin by quantifying the amount of parameter exploration. While the DynSp models have only a tiny fraction of parameters available at any given time, the pattern update means that they can explore all network parameters over the course of the training and thus increase the effective weight space. To measure the effectively covered space, we track the fraction of network weights of the corresponding dense network that have been activated at any point during the training and compare with the parameter count of the equivalent dense network to obtain the *total explored degrees of freedom* (DOF)[1].

We observe that the number of explored DOF can be controlled through the pruning ratio pr and the number of sparsity pattern updates n (Fig. 3). Increase of the update frequency leads to a simultaneous saturation in both task performance and the number of explored degrees of freedom (Fig. 3(a)). On the other hand, the pruning ratio pr reaches an optimal value and strongly influences the performance with different fraction of removed, new weights (Fig. 3(b)). Notably, we find that the optimal pruning ratio is reached once the ratio of DOF approaches 1, corresponding to full exploration of all network

[1] A similar quantity has been independently studied in Ref. [14] as "in-time over-parametrization."

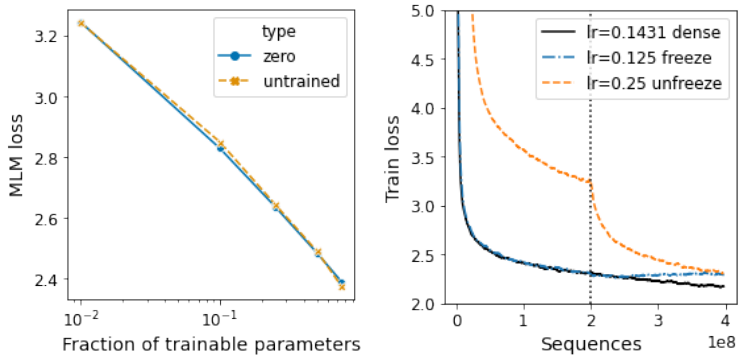

Figure (4) **Left panel** Evaluation MLM loss of BERT-Small with a random subset of the parameters set to zero (solid blue curve) or kept untrained (dashed orange). **Right panel** training loss curves of BERT-Small during phase I pre-training of 10 epochs (757k steps) for fixing a random subset of the parameter either early (orange dashed) or late (blue dash-dotted) during the training, as well as for the dense baseline (solid black). The vertical line indicates the unfreeze (freeze) event location, where untrainable parameters are made trainable (or trainable parameters are frozen). We pick the best learning rate for each experiment using a grid search over $0.000175 \cdot 2^m$ with $m = 0, 1, 2, 3, 4$.

parameters (Fig. 3(c)). Further increases in pr remove trainable weights that have just been initialized in the previous update step and lead to a deterioration in the task performance. In particular, the performance with different values of the ratio of removed parameters that had been newly allocated is almost independent of the number of updates.

Overall, we note that the best task performance is obtained by balancing the DOF while avoiding wasted compute in the form of parameters that are being allocated and immediately removed (see Fig. 3). Given these findings, we postulate that ideal training outcomes require an *exploration of all available parameters* as well as an only *moderate amount of noise injection*.

Effect of zeroed weight To better understand the impact of sparsification of the BERT architecture, we ablate the effect of zeroing of parameters by replacing zero-valued parameters with non-zero but non-trainable parameters. Like zeroed weights, the constant weights can not store information but might promote the propagation of signals or gradient-flow through the network [6, 15, 22], or lead to better utilization of the remaining trainable parameters. However, as shown in Figure 4, we find that none of these effects plays a relevant role in the training dynamics, since the task performance of the network with sparsified weights (dashed orange line) matches the one with untrained weights (solid blue line).

Sparsification and training phases To analyse the effect of sparsification at different stages of the training process, we keep a random subset of the network parameters frozen in the first half of the pre-training, before unfreezing in the second half (and vice versa). Unlike magnitude pruning, freezing and unfreezing of parameters ensures symmetry between the different phases (ignoring the linearly decaying learning rate schedule). Our findings indicate that *representation is continuously built up during training* with no particular effect of when the sparsification is applied.

Structured sparsity So far, we have used unstructured sparsity corresponding to block size 1x1, which is often less efficient when executed on hardware accelerators. However, we demonstrate that DynSp training can also preserve some task performance advantages when block sparsity of size 4x4, 8x8, and 16x16 is used (Table 1). This makes DynSp training promising for practical applications that seek to further benefit from the higher computational efficiency of block computation.

4 Conclusion & Future Work

In this work, we demonstrated that DynSp training of BERT leads to a more FLOP-efficient utilization of the trainable parameters. However, since we focused on phase I of pre-training only, it remains to be seen how the task performance of the phase I representation translates into phase II and downstream tasks. As a next step, we seek to shed further light on the conditions that enable the performance gains in unsupervised training, particularly the relationship of the number of available parameters and achievable task performance.

Future work may explore the performance of more structured sparsity. In this respect, we have found that even a naive block sparse version of the DynSp algorithm remains FLOP-efficient, which provides additional room to identify the best tradeoff for compute efficient training of large-scale language models.

References

[1] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep Rewiring: Training very sparse deep networks. nov 2017.

[2] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. EarlyBERT: Efficient BERT Training via Early-bird Lottery Tickets. dec 2020.

[3] Tim Dettmers and Luke Zettlemoyer. Sparse Networks from Scratch: Faster Training without Losing Performance. jul 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, 1:4171–4186, oct 2018.

[5] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners. nov 2019.

[6] Utku Evci, Yani A. Ioannou, Cem Keskin, and Yann Dauphin. Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win. oct 2020.

[7] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. jan 2021.

[8] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. mar 2018.

[9] Trevor Gale, Erich Elsen, and Sara Hooker. The State of Sparsity in Deep Neural Networks. feb 2019.

[10] Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-K Always Sparse Training. *NeurIPS*, 34, 2020.

[11] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. 2020.

[12] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. jun 2020.

[13] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE Layers: Simplifying Training of Large, Sparse Models. mar 2021.

[14] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do We Actually Need Dense Over-Parameterization? In-Time Over-Parameterization in Sparse Training. feb 2021.

[15] Ekdeep Singh Lubana and Robert P. Dick. A Gradient Flow Framework For Analyzing Network Pruning. sep 2020.

[16] 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. *Nat. Commun.*, 9(1):2383, dec 2018.

[17] Hesham Mostafa and Xin Wang. Parameter Efficient Training of Deep Convolutional Neural Networks by Dynamic Sparse Reparameterization. feb 2019.

[18] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon Emissions and Large Neural Network Training. apr 2021.

[19] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, jan 2017.

[20] SHiwei Liu and Decebal Constantin Mocanu and Yulong Pei and Mykola Pechenizkiy. Selfish Sparse {\{}RNN{\}} Training. Technical report, sep 2020.

[21] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Deep Learning in NLP. *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.*, pages 3645–3650, jun 2019.

[22] Kale-ab Tessera, Sara Hooker, and Benjamin Rosman. Keep the Gradients Flowing: Using Gradient Flow to Study Sparse Network Optimization. feb 2021.

[23] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. aug 2019.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Transformer: Attention is all you need. *Adv. Neural Inf. Process. Syst. 30*, pages 5998–6008, 2017.

A Technical details

- **Optimizer:** Throughout this work we use element-wise optimization based on Adam with weight decay 0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6} \times$ loss-scaling-factor and gradient clipping, which is known to work well with the sparse gradients found in NLP models.
- **Default learning rate schedule** consists of 10000 linear warmup steps up to the maximum learning rate (0.0002 for BERT-Medium and 0.0001 for BERT-Base), followed by a linear decay over the full training run.
- **Default dropout** is 0.1 for all models larger then BERT-Small.
- **Default BERT floating point precision**: We use datatype FP16.16 (16 bit compute with 16 bit partials) throughout the model. The second order moment in the Adam optimizer is computed and stored in FP32. Embedding are kept in FP16. The default loss-scaling factor for both BERT-Medium and BERT-Base is 512.
- **Initialization scheme**: The sparsity pattern is initialized randomly. The weights are initialized using a truncated normal initializer with initialization range of 0.02. This choice was motivated by having compared different initialization for the sparse model and found that the dense default truncated normal gives the best task performance as shown in Fig. 5. We found that preserving the variance of the activation statistics of the sparse model compared to the dense model [6] does not lead to any performance gains.

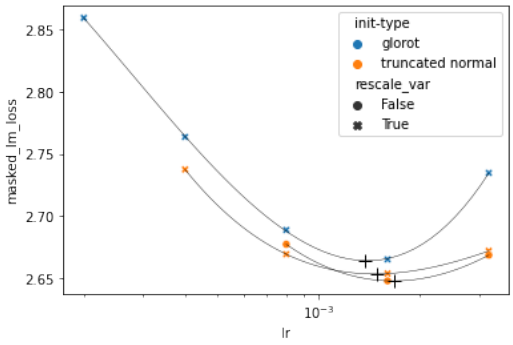

Figure (5) BERT-Medium with static unstructured sparsity of 0.9 imposed on all weights using Glorot (blue) or truncated normal (orange) as initialization scheme. The marker shape indicates whether the standard deviation of the weight initializiation was increased.

- **Pre-training dataset**: Phase I pre-training is performed on Wikipedia and BookCorpus using Whole Word Masking with a sequence length of 128.

A.1 Hyperparameters specific to dynamic sparsity (DynSp)

Two very important hyper-parameters for DynSp training are the sparsity pattern update frequency, i.e. how often the network topology is modified, and the pruning ratio, which determines the fraction of the network topology modified at each update step.

pr	n	MLM loss	NSP
0.50	40	2.468	0.645
0.50	80	2.430	0.656
0.50	160	**2.409**	0.626
0.50	320	2.419	0.649

Table (2) Number of sparsity pattern updates n dependence of DynSp BERT-Medium, $lr = 0.001397$, sparsity $s = 0.9$, 10 epoch, phase I (pruning ratio $pr = 0.5$ with cosine decay and random reallocation).

pr	n	MLM loss	NSP loss
0.25	160	2.439	0.655
0.50	160	2.413	0.684
0.75	160	2.411	0.668
1.00	160	2.459	0.698

Table (3) Pruning ratio pr dependence of DynSp BERT-Medium, $lr = 0.001397$, sparsity $s = 0.9$, 10 epoch, phase I (number of updates $n = 160$ with cosine decay and random reallocation). Same hyperparameters as in Table 2.

- **Pruning ratio and schedule dependence**: We found that the cosine decay of the pruning ratio introduced in Evci et al. (2019) [5] outperforms constant pruning schedules and leads to a reduction and convergence of the changes in network topology during training. We refer to the maximum pruning ratio pr simply as "pruning ratio" throughout the report.
- **Update frequency dependence**: Comparing the task performance of 20, 40, 80, 160 and 320 updates at sparsity ratios 0.9, we find that the task performance improves with the number of sparsity pattern updates. We chose the number of updates as $n = 160$ for sparsity ratio 0.9, as each update is associated with overhead due to the host communication, and we notice little improvements with further increases in the update frequency.

  Total number of pruned parameters The pruning ratio pr and the number of updates n jointly control the total number of pruned and re-allocated parameters. The total number of pruned and re-allocated parameters is proportional to their product. We obtain an optimal value of their product in terms of task performance as shown in Fig. 6.
- **Re-allocation criteria**: We found that random re-allocation outperforms gradient-based re-allocation. While the pruning criteria lead to a compression of the network topology, the growing criteria direct the evolution of the network topology and distinguish DynSp training as a form of neural architecture search during training from mere gradual pruning approaches. Understanding the requirements for efficient joint subspace exploration of parameter and network topology space using DynSp training will be essential to scale towards larger language models. In Fig. 7, we show that for gradient-based re-allocation, the dense gradient is dominated by outliers in the activation, e.g., along the input dimension

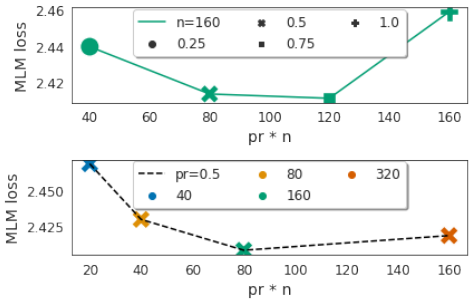

Figure (6) MLM loss vs pruning ratio pr times number of sparsity pattern updates n for DynSp training of BERT-Medium with sparsity ratio 0.9 for different values of (**Top panel**) pruning ratio pr (with $n = 160$) and (**Bottom panel**) sparsity pattern updates n (with $pr = 0.5$). Same data as in Fig. 3.

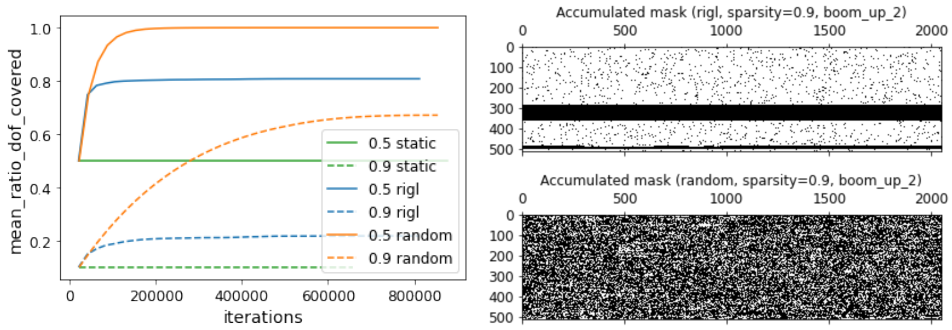

Figure (7) **Left panel** Fraction of explored degrees of freedom for static sparsity and DynSp training using gradient based (RigL [5]) vs. random re-allocation [3]. **Right panel** shows the corresponding sparsity patterns for the first up-projection in the feedfoward component ("Boom-up") of the second transformer block, accumulated throughout training, for sparsity=0.9 using RigL and random based reallocation. A black (white) dot corresponds to a parameter being non-zero (zero) at any point during training. The dark horizontal blocks in the RigL updates indicate a collapse due to outliers along the input dimension, which indicates that the effect arises from the activation part of the dense gradient update. This suggests that the collapse could be mitigated by reducing the influence of the activations during DynSp training update.

Model	B	MLM	FLOPs
Mini	-	2.614	$2.617 \cdot 10^9$
Matched (dense)	-	2.603	$2.738 \cdot 10^9$
Medium ($s = 0.9$)	16	2.621	$2.738 \cdot 10^9$
Medium ($s = 0.9$)	8	2.591	$2.738 \cdot 10^9$
Medium ($s = 0.9$)	4	2.546	$2.738 \cdot 10^9$
Medium ($s = 0.9$)	1	**2.408**	$2.738 \cdot 10^9$

Table (4) Task performance of DynSp BERT-Medium with sparsity 0.9 for various block sizes B compared to dense BERT-Mini with similar number of FLOPs and linear interpolation of the baseline values ("Matched") with exaclty the same number of FLOPs. Hyperparameters are not specifically tuned for different block sizes. See also BERT-Base results in Table 1.

of each layer, which imposes a strong bias on the available degrees of freedom during the update step. In agreement with this observation, we find that for random-based re-allocation, a significantly larger fraction of the network parameters is explored during training, while for gradient-based re-allocation the training remains constrained into a small subset of all network parameters (left panel of Fig. 7).

- **Blocksize metric** The importance of blocks of parameters is assessed by evaluating the L1-norm of the corresponding blocks.
- **Blocksize dependence** Block size dependence of BERT-Medium with sparsity 0.9 is given in Table 4.

A.2 FLOPs estimates for sparse multiplication with dense input

Throughout this report we assume the FLOPs for training a dense layer with sparse weight elements to approximately scale as $\mathcal{O}(3 \times 2 \times I \times B \times O \times f)$, where B the batch dimension, I is the input dimension, O the output dimension and f is the density of sparsity pattern imposed on the corresponding dense layer, which is to the sparsity ratio s as $f = 1 - s$. The FLOPs estimate can be divided into the following components:

1. **FLOPs estimate for sparse forward pass:** Assuming a sparse matrix $\mathbf{M}$ has a sparsity ratio s or a density $f = 1 - s$, the required matrix multiplication for a given dense input $\vec{x}$

and output $\vec{y}$ is

$$y_{bi} = \sum_{j|M_{ij}\neq 0} M_{ij} x_{bj}, \tag{1}$$

where $\mathbf{M}$ has dimension $[O, I]$ and $\dim(y) = [B, O]$, $\dim(x) = [B, I]$,

(a) **Sparse Multiplication**: performing the summation $z_{bij} = M_{ij} x_{bj}$ for i, j **iff** $M_{ij} \neq 0$ gives us a reduction of the total number of FLOPs by a fraction of non-zero elements in $\mathbf{M}$ times B leading to $B \times O \times I \times f$ FLOPs.

(b) **Sparse Addition**: performing $\sum_j z_{bij}$ requires us to calculate the exact number of non-zeros along the input dimension. $B \times O \times \text{prob}(out) \times I \times \text{prob}(in) - B \times O \times \text{prob}(out)$, where we defined some probability for non-zero values along out prob(out) and input dimension prob(in). Assuming a uniform distribution we estimate the FLOPs count to scale approximately linearly with the sparsity ratio $B \times O \times I \times f - B \times O \times f / \text{prob}(in)$ to first order.

The total FLOPs of sparse multiplication used in the forward pass scales approximately linearly in the number of non-zeros, i.e. $\mathcal{O}(2I \times B \times O \times f)$.

2. **FLOPs estimate for recursive propagation of error through the network**: Involves a multiplication of the dense error with the transposed sparse matrix leading $\mathcal{O}(2I \times B \times O \times f)$ additional FLOPs.

3. **FLOPs estimates for the outer product** The weight update itself is formed by a sparse outer product, where only the sparse components need to be updated, which leads to a further reduction in the number of FLOPs that scales linearly with the density of the matrix.

A.3 Learning rate for sparse and dense models

The results of the learning rate sweep of BERT with various sparsities are given in Table 5. The corresponding learning rate sweep for the dense BERT-family is given in Table 6. We confirmed that the optimal learning rates for static sparsity agree with the ones for DynSp in Table 7. We also evaluated the learning rate dependence of the sparse model for multiple model sizes (not shown).

lr	sparsity	MLM loss	NSP loss
0.0001	0.00	2.179	0.610
0.0002	0.00	2.115	0.598
0.0002	0.00	2.115	0.605
0.0004	0.00	2.116	0.606
0.0008	0.00	2.164	0.633
0.0001	0.25	2.278	0.627
0.0002	0.25	2.204	0.642
0.0004	0.25	2.186	0.596
0.0008	0.25	2.223	0.638
0.0001	0.50	2.412	0.679
0.0002	0.50	2.338	0.671
0.0004	0.50	2.283	0.631
0.0008	0.50	2.298	0.648
0.0002	0.75	2.551	0.741
0.0004	0.75	2.483	0.685
0.0008	0.75	2.446	0.671
0.0016	0.75	2.449	0.647
0.0032	0.75	2.547	0.707
0.0004	0.90	2.723	0.758
0.0008	0.90	2.677	0.711
0.0016	0.90	2.648	0.706
0.0032	0.90	2.669	0.697

Table (5) Learning rate (lr) sweep of static sparsity BERT-Medium, sparsity $s = 0, 0.25, 0.5, 0.75, 0.9$.

model	lr	MLM loss	NSP loss
Mini	0.000050	3.062	0.839
Mini	0.000100	2.833	0.811
Mini	0.000400	2.625	0.742
Mini	0.000800	2.606	0.775
Mini	0.001600	2.628	0.779
Mini	0.003200	2.665	0.783
Small	0.000800	2.326	0.644
Small	0.000400	2.310	0.621
Small	0.000200	2.329	0.635
Small	0.001600	2.418	0.768
Medium	0.000200	2.115	0.605
Medium	0.000400	2.116	0.606
Medium	0.000800	2.164	0.633
Medium	0.000100	2.179	0.610
Base	0.000025	2.115	0.599
Base	0.000100	1.878	0.542
Base	0.000050	1.972	0.569
Base	0.000200	1.843	0.488

Table (6) Learning rate (lr) sweep for dense BERT-family consisting of BERT-Tiny, Mini, Small, Medium and Base.

	model	lr	MLM loss	NSP loss
0	Medium	0.00064	2.467	0.647
1	Medium	0.00128	2.410	0.670
2	Medium	0.0026	2.429	0.674
3	Medium	0.0051	2.521	0.654

Table (7) Learning rate sweep of DynSp BERT-Medium, sparsity $s = 0.9$, 10 epoch, phase I used to confirm that the optimal learning rates for static sparsity from Table 5 translate into optimal learning rates for DynSp.