VARIANCE PRUNING: PRUNING LANGUAGE MODELS VIA TEMPORAL NEURON VARIANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

As language models become larger, different pruning methods have been proposed to reduce model size. However, the typical sparsity patterns that are formed by commonly pruning regimes do not fully exploit the properties of modern hardware devices on which these models are being trained and deployed. Most known unstructured, or even structured, pruning regimes usually introduce requirements for additional hardware components to make these sparsity patterns useful. Here we propose a simple pruning algorithm, based on variance analysis of output neurons that correspond to entire rows of weights. Our algorithm facilitates the construction of row-sparse matrices, allowing an extremely convenient way of exploiting this sparsity on existing hardware architectures. Empirical experiments with natural language understanding tasks show that our method leads to little to no accuracy degradation, and at times even better accuracy, using a 50% sparse $\text{BERT}_{\text{LARGE}}$ model.

1 INTRODUCTION

Over the last decade, deep learning researchers, especially in the field of Natural Language Processing (NLP), have scaled neural networks sizes from a few millions up to billions and even trillions in model parameter count. In particular, Transformer-based models (Vaswani et al., 2017) such as BERT (Devlin et al., 2018) were extended from hundreds of millions up to tens and hundreds of billions or parameters over a period of merely three years (Rosset, 2020; Brown et al., 2020). This trend of growth typically yields a substantial improvement in model quality. Transformer-based architectures have also become more popular in computer vision (Lu et al., 2020; 2019; Dosovitskiy et al., 2020; Yuan et al., 2021), matching and even outperforming convolutional neural networks in computer vision tasks.

Due to their success, industry practitioners and cloud service companies have a great interest in training and deploying large Transformers, which are a fundamental part of their business model. However, training and deploying such large models, which are constantly scaled up in memory foot-print and compute, require a significant amount of expensive resources. Therefore, by an active and immense research, hardware manufacturers are motivated to reduce these memory and computational bottlenecks. In fact, they even incorporate hardware components and software designs in their products to enable state-of-the-art algorithms and methods (Aojun Zhou, 2021).

Pruning is a common practice to reduce model size with a relatively minor cost in accuracy drop. In this setting, the model is being effectively compressed by removing weights that are considered to be less important according to some significance metric. Sparse models gain not only from low memory footprint on dedicated hardware devices, but also require much lower bandwidth and energy usage (Horowitz, 2014). Pruning methods are commonly divided to unstructured and structured pruning. In the former, the sparsity pattern is of no particular form, while in the later, elements are being eliminated in a particular arrangement. For example, a structure where any M out of a contiguous block of N elements are being zeroed out. This pattern is also often referred to as M:N structured sparsity (Aojun Zhou, 2021). Magnitude pruning (Han et al., 2015) is one of the most vastly used method to induce unstructured sparsity in domains such as vision (Guo et al., 2016) and NLP (Gale et al., 2019). The essential idea here is to remove weight elements with the smallest magnitude and let the optimization algorithm compensate for the removed parts such that other weights will sustain the lost information. Magnitude pruning, however, is less effective in the

transfer learning regime, where fine-tune to an end-task take place, since weight values are mostly predetermined by the pre-trained model (Sanh et al., 2020b; Guo et al., 2021).

In this work, we present a simple yet effective method to learn sparse Deep Neural Networks (DNNs). Inspired by Dalvi et al. (2020), who found that more than 85% of neurons are redundant in Transformer models for solving NLP tasks, we propose to prune entire rows of a weight matrix, which correspond to the matrix's output neurons. The key idea of our approach is to prune weight rows whose neurons exhibit low variance during training, since we hypothesize these neurons are less important for the model. After each matrix multiplication layer is applied, we gather statistical information for every output neuron across many timestamps and across many batches of input sentences. Rows that correspond to output neurons whose variance is the lowest are then pruned. This procedure is performed gradually during the pre-training of the language model and is halted once a desired sparsity level of the entire model is obtained. We term our method **Variance Pruning**.

Variance Pruning addresses several shortcomings in common methods for weight-based pruning. First, common pruning methods tend to yield irregular sparsity masks. Therefore, the memory access to read those sparse irregular matrices causes extremely inefficient performance and requires sparse operation libraries and even special hardware handling. In contrast, our approach induces row-wise sparse matrices, which can easily be squeezed into equivalent dense matrices by simply removing the pruned rows. Hence, memory access becomes straightforward as in the original dense model, does not require any special support of sparse multiplication libraries, and can work with existing efficient Basic Linear Algebra Subprogram (BLAS) libraries for dense matrix multiplications.

Another shortcoming of most prior pruning methods in NLP is that they are applied after the model has already been pre-trained, during the fine-tuning stage on a particular downstream task. However, Guo et al. (2021) show that during fine-tuning of a pre-trained model for a range of downstream tasks weights may have a minuscule change with respect to the pre-trained weights. Eventually, they left with only 1% of the overall weights that need to be modified on top of the pre-trained model. Therefore, pruning during fine-tuning may not be effective. In contrast, we apply our pruning scheme during the pre-training phase of the model, when the model changes the most. After we prune the model to a desired sparsity level, we can use the same pre-trained weights and fine-tune them on all downstream tasks without needing to do so for each individual task separately. All in all, our method allows us to benefit from sparse training and fine-tuning and enables significantly faster computations in training as well as in deploying NLP models.

We conduct our experiments using the $\text{BERT}_{\text{LARGE}}$ model and examine the effectiveness of our proposed Variance Pruning on commonly used General Language Understanding Evaluation (GLUE) downstream tasks (Wang et al., 2018) and on SQuADv1.1 (Rajpurkar et al., 2016) (We exclude the problematic WNLI set as in Devlin et al. (2018)). We show that our approach leads to little to no degradation and even some improvement, compared to an unpruned $\text{BERT}_{\text{LARGE}}$ baseline. Importantly, our pruned model provides a 50% reduction in memory requirements for storing weights and the optimizer states. We also show the memory limitations of fine-tuning the dense $\text{BERT}_{\text{LARGE}}$ model on a standard four GPU machine. Lastly, we show that our pruned model has a better computational efficiency on a standard GPU machine, compared to a dense full model. We release our code at http://ANONYMIZED for the benefit of the community.

2 Related Work

Unstructured and Structured pruning in modern hardware devices. As DNN models continuously grow in size (the number of parameters), model compression via pruning became a field of extensive research. Unstructured pruning removes individual elements of the weight matrix regardless to their location (Han et al., 2015; Louizos et al., 2018), while in structured pruning, elements are being eliminated in a particular pattern (Li et al., 2017; Luo et al., 2017). There is a substantial amount of work that proposes unstructured pruning, such as magnitude pruning (Han et al., 2015; Frankle & Carbin, 2019) and pruning with connection sensitivity (Lee et al., 2019). Unstructured sparsity, however, induces irregularity to the non-zero weight matrices and hinders the ability to accelerate computation on modern hardware architectures. The overhead to transport, store, and use the irregularly sparse matrices for computation can be extremely high.

Recently, the work on the Lottery Ticket Hypothesis (Frankle & Carbin, 2019) suggested that large and over-parameterized DNNs contain a sub-network (a "winning ticket") that alone can be trained and reach the full model accuracy. Moreover, such networks are easier to train because of the existence of many combinations of sub-networks to be "chosen" from during the stochastic optimization process. They use iterative magnitude pruning (IMP) as its core component to find the matching sparse sub-network. In this procedure, a network is being repeatedly trained for a pre-defined number of steps, pruned to some extent and then being reset to the initial value of weights that survived. This concept was later broadened into the scope of NLP (Chen et al., 2020), by showing that matching sub-networks exist at various levels of sparsity for a range of downstream tasks. Later, Prasanna and Rogers (Prasanna et al., 2020) showed that structured IMP pruning in BERT-like models, which is much more constrained than unstructured magnitude pruning, still achieves good accuracy and results in patterns that store a non-trivial amount of valuable information.

Nvidia recently announced the Ampere A100 GPU (Nvidia, 2020), built-in with Sparse Tensor Cores to accelerate such fine-grained sparsity patterns. This architecture includes special hardware support, which induces extra cost. On the other hand, ASIC devices with a single multiplication engine will not benefit from fine-grained sparsity, unless dedicated hardware support is introduced, which is also limited to certain unique irregular patterns. To overcome the aforementioned limitation, we introduce a method to obtain a universal sparsity pattern that is more effective and easy to accelerate on modern devices. Moreover, our pruning method creates a smaller and dense model which does not require special hardware support and the use of sparsity primitives, instead simply utilizing the original dense BLAS operations.

Structured sparsity patterns (Wen et al., 2016), in particular methods that prune blocks (Wang et al., 2019) or entire filters (Li et al., 2017) are more beneficial for some modern hardware accelerators. Still, it is not always possible to convert these patterns to a dense model that can leverage existing BLAS operations. Moreover, methods that are commonly used to obtain these patterns rely on importance criteria extracted from the weights themselves. For example, methods that consider the changes in weights during fine-tuning (Sanh et al., 2020b). This practice, rely on the presumption that weights with bigger change are more important, which may not always be the case (Guo et al., 2021). In contrast, our method is beneficial to modern hardware by design as its sparsity patterns boils down to a dense and smaller model. Furthermore, We rely on importance information retrieved from the output neurons, which we show that hold a more valuable information on the importance of their origin weights.

Analyzing output neurons. Recently, researchers attempting to interpret internal representations in deep NLP models have turned their attention to the role of individual neurons (dimensions) in these representations (Sajjad et al., 2021). For instance, Durrani et al. (2020) analyzed how individual neurons capture core linguistic properties of morphology, syntax, and semantics, at different layers. They found that neurons that capture word morphology were predominantly found in the lower and middle layers, while those capturing syntax were found at the higher layers.

More closely related to our motivation, Dalvi et al. (2020) investigated redundancy in Transformer-based models at the level of individual neurons or full layers. They analyzed representations from (fixed, not fine-tuned) language models, and found that up to 85% of the neurons across the network are redundant in general and even more w.r.t a specific task. As in most work in this area, they studied neurons at the output of each attention block (after both the self-attention and feed-forward layers). We are inspired by these findings, but search for neuron-level redundancies in each and every one of the parameterized layers inside the Transformer layers, in particular all the matrix multiplication operations. Importantly, we work in the more common fine-tuning scenario, whereas Dalvi et al. (2020) worked with frozen, non-fine-tuned models.

Pruning weights by their magnitude. Common approaches for pruning weights, such as magnitude pruning (Han et al., 2015) or IMP (Chen et al., 2020), can be quite restrictive, as they focus on the absolute values of individual or groups of learnable parameters. As such, they do not take into account the cumulative effect of the interaction between weights and the neuron activation in different linear layers. On the one hand, individual weights can be large in magnitude but negligible when multiplied by small values from the previous layer. Moreover, even if a large-magnitude weight element results in a relatively large value after the multiplication, it may remain unchanged across different timestamps and input batches. On the other hand, smaller values can have considerably

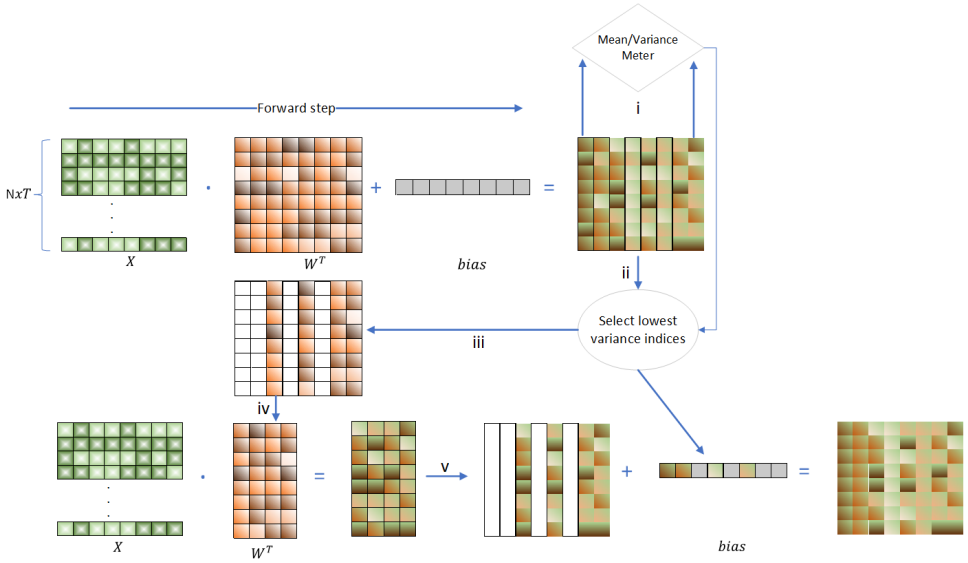

Figure 1: Illustration of our approach. At each forward step, (i) we measure the mean and the running variance of each individual output neuron over batches and time steps. After τ optimization steps, (ii) we select the neurons whose variance was the lowest, and (iii) prune the weight rows (columns in W^T) that correspond to the selected lowest variance indices. Then, (iv) we squeeze the weight matrix and perform the weight multiplication with a dense weight matrix. Finally, (v) we expand the output product and place sampled mean values in the corresponding indices of the bias term to be added to the output.

more influence when the output changes significantly across inputs. Our novel approach therefore examines the variance of neurons, rather than the magnitude of weights or other elements.

When to prune. Recent work by Sanh et al. (2020b) obtains the importance score of a weight by observing the change of explicit weight elements during task-specific fine-tuning. Their approach is to find and prune elements that are moving the most away from zero. In contrast, Guo et al. (2021) show that during fine-tuning of a pre-trained model for a range of downstream tasks, weights may have a minuscule change with respect to the pre-trained weights. They show that it is sufficient to update only 1% of the model's overall weight elements via the fine-tuning process, in order to achieve on-par accuracy results compared to fine-tuning without any limitations. According to their results, the fact that particular weights that have changed more than others during fine-tuning still does not indicate their importance, as their significant adjustment was done in the earlier pre-training phase.

The work of Wu et al. (2020) analyze the similarities between different Transformer-based architectures, when fine-tuned on different downstream tasks. They found that lower layers change less than higher layers during fine-tuning. They also found that higher layers changes are related to localized information about the specific task. Still, minor and localized changes with respect to some task is not an indication of the importance of weight elements with respect to global linguistic information which probably attained mostly during the pre-training. Inspired by the above observations, our pruning method focus on neuron activations that have minor fluctuations during the pre-training phases rather than the weights themselves.

3 METHODOLOGY

In this paper, we focus on Transfomer-based models that take an input consisting of words or tokens that are embedded into a d-dimensional feature vector. Later, this input is fed into each Transformer-layer that consists of six linear projection layers. The key point of variance pruning via output neurons is to collect the mean and variance of each linear layer's output neurons, for every input token across batches and time steps, during the pre-training phase of the model. Then, we select the neurons whose variance is the smaller and prune the entire weight matrix rows that correspond to the selected neurons. We place the collected mean of each particular neuron in its corresponding bias term. The statistics for each output neuron of each linear layer are collected at each training step. Figure 1 illustrates this procedure.

Furthermore, at some pre-determined point, after we prune part of the weights based on the smallest variance indices, we reset the mean and variance meters and start collecting them until reaching the next pruning step. We reset the meters to ensure that the new statistics are collected with respect to the new pruned model. This process proceeds until reaching the desired sparsity level. Intuitively, we prune weights whose rows correspond to neurons that stay unchanged or only slightly change compared to others and can be expressed by their mean value.

3.1 COLLECTING NEURON STATISTICS

For some linear layer l in the model, let $X^{(l)} \in \mathbb{R}^{m \times d}$ denote an input, where d is the hidden dimension size[1] and $m = N \times T$ the number of input tokens for batch size N and sequence length T,[2] and let $W^{(l)} \in \mathbb{R}^{d \times d}$ denote a square weight matrix (w.l.o.g). For every output neuron $i \in [1, \ldots, d]$ that corresponds to row i in $W^{(l)}$, the matrix multiplication operation at training step t is given by:

$$Y_t^{(l)}[i] = X_t^{(l)} W_t^{T(l)}[i] \in \mathbb{R}^{m \times 1} \tag{1}$$

Where $W_t^{T(l)}[i]$ is the transposed weight matrix of layer l at training step t for output neuron i. For a total number of τ tokens at a given time window during training, we first compute the mean of neuron i at layer l as follows:

$$\mu_\tau^{(l)}[i] = \frac{1}{m\tau} \sum_{t=1}^{\tau} \sum_{j=1}^{m} Y_{t,j}^{(l)}[i] \tag{2}$$

Next, since we train the model with SGD and attain the statistics per batch, we compute the exponentially weighted moving variance (EWMV), which is more robust to numerical instability. The variance of neuron i at layer l in step t is given by:

$$\sigma_t^{2(l)}[i] = \frac{1}{m-1} \sum_{j=1}^{m} (Y_{t,j}^{(l)}[i] - \mu_t^{(l)}[i])^2 \tag{3}$$

For multiple tokens and timestamps, and a degree of weighting decrease coefficient of α,[3] we compute the EWMV of neuron i at layer l by the following recursive formula:

$$\hat{\sigma}_t^{(l)}[i] = \begin{cases} \sigma_1^{2(l)}[i], & t = 1 \\ \alpha \sigma_t^{2(l)}[i] + (1-\alpha)\hat{\sigma}_{t-1}^{(l)}[i], & 1 < t \leq \tau \end{cases} \tag{4}$$

and denote the estimate of the variance of neuron i at layer l over a subtotal of τ tokens as $\hat{\sigma}_\tau^{(l)}[i]$. With the collected statistics, we are ready to compute the sparsity mask.

3.2 COMPUTING AND APPLYING THE MASK

We first define a new linear layer with a modified weight matrix, such that $W'^{(l)} = W^{(l)} S_t^{(l)}$, where W_l is the original weight matrix for layer l and $S_t^{(l)}$ denotes the sparsity mask at step t, which is initialized to the identity matrix $S_0^{(l)} = \mathbb{1}$. We gradually zero out the entries of $S_0^{(l)}$ based on the smallest-variance indices in $\hat{\sigma}_t^{(l)}[i]$. Instead of constructing the mask based on individual-layer local statistics, we extend our scope to be across the model. For each type of projection independently, we concatenate variance vectors of that type from all model layers. In each of L Transformer layers, there are six projection types: $(W_Q, W_K, W_V, W_O, W_{FF1}, W_{FF2})$ for query, key, value, attention output, and two feed-forward linear layer types, respectively. For example, the global variance vector at step t for output neuron i from a projection layer of type Q (query) is:

$$\hat{\sigma}_t^{(l_Q)}[i] = \{\hat{\sigma}_t^{(l_{q0})}[i], ..., \hat{\sigma}_t^{(l_{qL-1})}[i]\} \tag{5}$$

[1] d can differs between linear layers. For example, in the Transformer FF layer d is four times larger than in other linear layers.

[2] In batched training, sentences are padded up to a maximum sequence lengths. We ignore pad tokens and measure the statistics only on the valid tokens, so in practice $m \leq N \times T$.

[3] In our experiments α is set to 0.9.

where $\hat{\sigma}_t^{(l_{ql})}[i]$ is the estimated variance vector for neuron i of the query projection at layer l. For exposition purposes, we will present our implementation on layer type Q, but in practice we form the global variance vectors for the keys, values, attention output and both of feed-forward layers, as well. Next, we denote $P_r^{(l_Q)}$ as the percentile value for a pruning percentage r [4] of all neurons of layer type Q. Then, for each individual query layer l_{qn}, where $n \in [0..L-1]$, the mask is given by:

$$S_t^{(l_{qn})} = \begin{cases} 0_{\times d} & \text{if } \hat{\sigma}_t^{(l_{q0})}[i] < P_r^{(l_Q)} \\ 1_{\times d} & \text{otherwise} \end{cases} \tag{6}$$

and we update the mask to with the pruned entries from previous steps, i.e., $S_t^{(l_{qn})} = S_t^{(l_{qn})} S_{t-1}^{(l_{qn})}$.

By choosing the smallest elements in the global concatenated variance vector $\hat{\sigma}_t^{(l_Q)}[i]$, we will end up with a nonuniform sparsity level along the layers. We hypothesise that with this approach we are less restrictive with respect to which rows we should prune, in line with results on how neurons in different layers behave differently in Transformer models (Section 2). For example, if we were confining ourselves to each individual layer as most common pruning methods do (Han et al., 2015; Sanh et al., 2020b; Li et al., 2017), we might have been eliminating rows whose output neurons have a more important role than other output neurons in different layers of the same type. Nonetheless, with our approach, we still assume that linear layers of the same type have a similar role across the model layers.

3.3 The advantage of Variance Pruning in forward and backward computation

During the training process of DNNs, a general matrix multiplication layer requires three matrix multiplication operations: one in the forward pass and two in the backward pass. The forward pass is given by:

$$Y = XW^T \tag{7}$$

In our method we prune some rows of W and squeeze the matrix, in a trivial manner, to its smaller but dense counterpart. We denote as r the reduction factor from pruning and squeezing W, such that the number of valid rows in W is $\frac{d}{r}$. The forward matrix multiplication requires $\frac{md^2}{r}$ operations in the pruned model's layer as opposed to md^2 in an un-pruned model. In the backward pass, the gradients with respect to the input and the weight matrix are given by:

$$\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial Y} W \tag{8}$$

and

$$\frac{\partial \mathcal{L}}{\partial W} = X^T \frac{\partial \mathcal{L}}{\partial Y} \tag{9}$$

respectively, where $\mathcal{L}$ is the loss function we are trying to minimize in the optimization process. Since $\frac{\partial \mathcal{L}}{\partial Y} \in \mathbb{R}^{m \times \frac{d}{r}}$, both gradients with respect to W and X require $\frac{md^2}{r}$ operations each. As a result, the computational footprint is reduced by a factor of r. This reduction in the number of operations allows us to accelerate training as well as deployment. Furthermore, the memory capacity of storing the model weights and the bandwidth required to fetch them from the volatile memory is also reduced by a factor of r.

3.4 Preliminary analysis

To motivate our technique, we first perform a preliminary study where we compare pruning via neurons variance and random sampling of weight rows as a baseline. We ensure an identical sparsity level between the baseline and our approach by sampling the same amount of rows for the random setting as we decided to remove using our approach. In this experiment we first fine-tune a pre-trained $\text{BERT}_{\text{BASE}}$ to four common downstream tasks as we described later on. Then, we gradually remove weight rows in both models. We do not re-train, fine-tune or do any weight updates on both settings after removing the portion of rows in the weight matrix. Therefore, we expect that the accuracy will gradually drop until reaching an asymptotic minimum. Our hypothesis is that removing weight rows

[4] In our experiments we use a linear scheduling by globally pruning $r = 10\%$ of the model weights every 500 steps, reaching 50% sparsity in total.

whose corresponding variances are the smallest will result in a moderate decrease compared with removing random rows of the same amount at each individual layer.

We confirm our hypothesis by testing on four benchmark language tasks. Microsoft Research Paraphrase Corpus (MRPC) (Dolan & Brockett, 2005) for detecting semantic similarity. Winograd Schema Challenge (recast as QNLI) (Sakaguchi et al., 2020) for Natural Language Inference, taken from Wikipedia. Semantic Textual Similarity Benchmark (STSB) (Cer et al., 2017) and Stanford Question Answering Dataset version 1.1 (SQuAD1.1) (Rajpurkar et al., 2016), questions for machine comprehension of Text. Figure 2 demonstrates that the validation accuracy for $\text{BERT}_{\text{BASE}}$ models whose weights were pruned according to the corresponding low variance neurons dropped much more gradually than the random counterparts. This result is quite notable across all tested tasks. Following these results, we continue with larger scale model and incorporate it in the pre-training process.

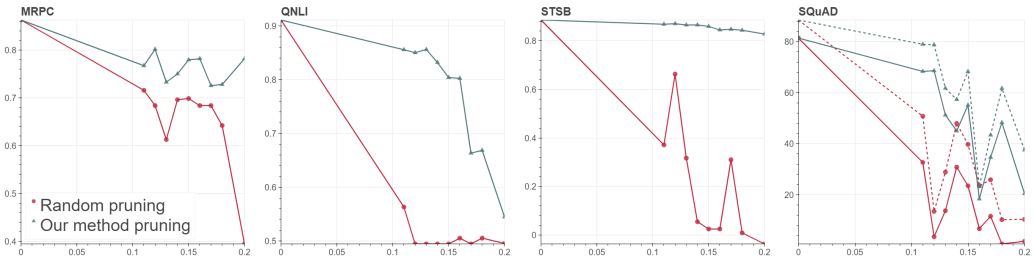

Figure 2: $\text{BERT}_{\text{BASE}}$ fine-tuned on four downstream tasks. We compare random pruning and variance pruning without any additional fine-tuning. The plots show validation accuracy on MRPC and QNLI, Spearman's correlation on STSB, and accuracy (continuous) and F1 (dotted) scores on SQuAD1.1, as a function of the sparsity level. As the sparsity level increases, variance pruning (green lines) leads to much slower performance decrease compared to pruning random rows.

4 Experimental setup

In this section we describe the experiments we performed to assess the effectiveness of Variance Pruning. We pre-train a completely random `BERT-large-uncased` model, following the original pre-training scheme from Devlin et al. (2018), with two phases: Masked language modeling (Masked-LM) and Next Sentence Prediction tasks (NSP). We experiment with a common transfer learning scenario for pre-trained NLP models Devlin et al. (2018); Ruder et al. (2019), where we fine-tune the pre-trained pruned model on the General Language Understanding Evaluation (GLUE) downstream tasks (Wang et al., 2018) and on SQuADv1.1 (Rajpurkar et al., 2016), a question answering dataset. We perform the two-phase pre-training with 8 A100 GPUs with 32G memories and report the sparsity results with respect to `BERT-large-uncased` on all of the downstream tasks. More implementation details are described in Section A

We apply our Variance Pruning method during the first pre-training phase of Masked-LM. We start with eliminating 10% of the weights when reaching step 500 and then we continue pruning an additional 10% at every additional 500 steps until we reach a final sparsity ratio of 50% 2,500 steps. Then, we maintain the pruned weights zeroed out during the rest of the Masked-LM phase as well as the entire NSP phase. Finally, we use the sparse pre-trained model to fine-tune and evaluate on the downstream tasks. Moreover, to speed-up the fine-tuning stage and decrease its memory consumption, we squeeze the non-zero rows of the weight matrices to their dense counterparts and expand them only when bias addition is required (see Section 3 and Figure 1). We note that this beneficial procedure could also be applied during the second phase of pre-training.[5]

We implement Variance Pruning based on HuggingFace's Transformers library Wolf et al. (2020). We compare the results obtained with the variance-pruned `BERT-large-uncased` both with the results reported in the original paper and with results obtained when we run a full dense model using HuggingFace's code. We report both baselines to facilitate a fair comparison with our implementation.

[5]We did not pack the non-zero rows as we launched both phases together, but this is essentially not mandatory and we could simply pack the matrices after the first phase.

Table 1: Results of full dense $\text{BERT}_{\text{LARGE}}$ baselines and our variance-pruned 50% sparse model (VP), on the GLUE and SQuAD 1.1 tasks. Baseline results are reported both from the original paper Devlin et al. (2018) and when running the HuggingFace code. Our results are based on the same pre-trained sparse model of total 50% zeroed weights on all tasks. Scores that have parenthesised counterparts inside the cell are achieved with further distillation; the ones inside the parenthesis are without distillation.

Model	MRPC		CoLA	SST2	RTE	STSB	QQP		MNLI		QNLI	SQuAD1.1		**AVG**
	Acc	F1	Corr	Acc	Acc	Corr	Acc	F1	Acc m	Acc mm	Acc	Acc	F1	
Full (original)	85.4	89.3	60.5	94.9	70.1	86.5	89.3	72.1	86.7	85.9	92.7	84.1	90.9	83.54
Full (HuggingFace)	86.2	90.3	61.8	93.1	67.1	89.5	91.6	88.8	86.7	86	92.3	84	91	85.26
50% sparse (VP)	87	90.8	62.07	92.9 (92.2)	69.7	88.6	91.2 (90.1)	88.2 (87)	84.6	85.3	91.3	84	90.54	85.09

5 RESULTS

The main results are presented in Table 1, where MRPC, QQP, and SQuAD1.1 scores are accuracy and F1, CoLA and STSB are Matthew's and Spearman's correlations respectively, SST2 and RTE are accuracy, and MNLI has accuracy on matched and mismatched sets. We keep a fixed sparsity ratio of 50% across all of the downstream tasks and compare them against the baselines. We observe that using a sparse model, obtained with Variance Pruning, has little effect, no effect at all or even slightly better effect on performance in the various tasks. In particular, our method achieved better results on MRPC (in both metrics) and on CoLA. In SST2 and QQP benchmarks, we used an additional knowledge distillation step (Hinton et al., 2015; Sanh et al., 2020a) to boost performance, where we combined the original loss with knowledge distillation loss on the output distributions. As shown in Table 1, using distillation, SST2 goes from 92.2 to 92.9 and in QQP the accuracy and F1 go from 90.1 and 87 to 91.2 and 88.2, respectively.

The average score over all datasets obtained by our sparse model is on-par with the score we get using the HuggingFace code and outperforms the average score from the original paper. Namely, our pruned model achieved an average score of 85.26 compared to 83.54 in the original paper and 85.09 when running the full dense model using the same code case we conducted our experiments with.

Memory and Speed analysis. In Table 2, we compare the memory footprint and speed of a full $\text{BERT}_{\text{LARGE}}$ dense model with the dense counterpart of our pruned model. The experiment was performed on the SQuAD1.1 dataset on a 12GB TitanX GPU. We show that in addition to the little loss to small gain in accuracy, we were able to reduce memory footprint as well as increase computational efficiency by a substantial margin. In fact, this margin can be further improved by addressing the matrix expansion after the multiplication operation in the forward as well as in the backward propagation. In our experiments, we did not leverage any type of specialized kernel to enhance this expansion operation performance; however, the scatter operation can be replaced with a direct write to an intermediate buffer in the correct index order [6] such that the bias will be added properly. The table also shows that for a batch size of four, the full model cannot run as it crashed due to an out-of-memory error. Our model, however, has no such problem.

Table 2: GPU memory usage (GB) and computational efficiency (iterations per second) when fine-tuning $\text{BERT}_{\text{LARGE}}$ on SQuAD 1.1. We compare the full dense model with the densified 50%-sparse model, with batch sizes ranging from one to four and a common fixed sequence length of 384. For a batch size of four, the full dense model goes out-of-memory.

Batch	Dense full		Dense 50%-pruned	
	GB	it/sec	GB	it/sec
1	8.95	1.92	6.41	2.45
2	9.22	1.58	7.85	2.05
3	10.63	1.3	9.26	1.6
4	-	-	10.51	1.42

[6] We can pass the valid row indices as an auxiliary input to the model.

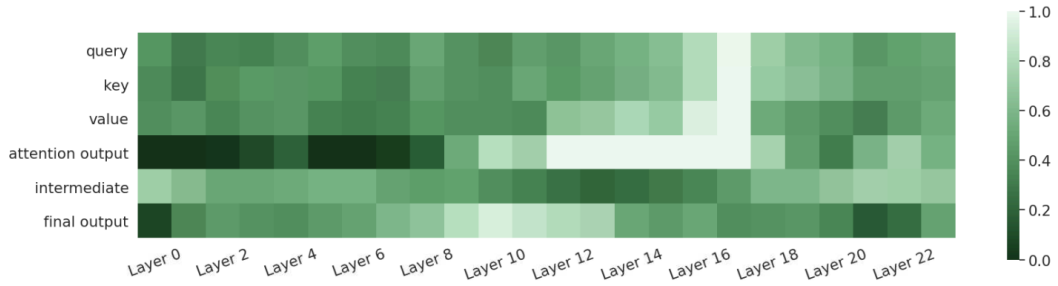

Figure 3: Sparsity level within each Attention black in a pre-trained $\text{BERT}_{\text{LARGE}}$ model after applying Variance Pruning in the pre-training phase and reaching 50% sparsity across all layers. Each block consists of six linear projection layers: query, key, value, self attention output, intermediate and final output projections.

6 Sparsity analysis

Figure 3 exhibits the sparsity levels, across all of the model's linear layers. This heat map shows the importance of weights after applying our VP method during pre-training with a (masked) language modeling objective, different from observing the importance of weights during fine-tuning (Prasanna et al., 2020). We observe that most of linear layers are less sparse (darker) in the lower layers than in the higher ones. This complies with analyses showing that (a) the lower layers carry lower-level language properties, which are the building blocks of textual language understanding and may thus be important for any end task (Durrani et al., 2020); and (b) those lower layers are less affected by fine-tuning than higher layers Kovaleva et al. (2019); Wu et al. (2020). On the other hand, during pre-training, the top layers specialize in the language modeling task, and hence it is more reasonable that they become sparser. We also observe a high sparsity of the attention output layer in the intermediate layers, different from the findings of Prasanna et al. (2020) when pruning during fine-tuning. This again highlights the importance to consider pruning during pre-training. We leave further investigation of the sparsity patterns from an interpretability perspective to future work.

6.1 Achieving additional sparsity

During the pre-training phases, we did not constrain the weight matrices within a particular linear projections type, e.g., query, key, value etc., to any sparsity ratio. However, our constraint was to reach $\frac{100}{r}\%$ sparsity across the model for the particular projection. As shown in Figure 3, for a given projection type out of the six, each individual Transformer layer reached a different sparsity ratio than its counterparts in other layers. For example, the self attention projection in layer number sixteen, was left with less than 0.1% non-zero elements. However, the linear projections that precede it in layer sixteen, e.g., query, key and value, did not zero out during the pre-training. Still, their sparsity ratios are smaller versus the same projection types in other layers. This makes sense in light of the fact that their contribution is negligible when reaching the self attention output projection. Due to this, we could further prune these three weight matrices and achieve additional sparsity with no accuracy degradation.

7 Conclusions and Implications

In this work, we consider the case of pruning NLP models by analyzing the variance of their output neurons, which correspond to entire weight matrix rows. Weight rows whose neurons exhibit low variance during pre-training are pruned. Our results sustain the hypothesis that output neurons hold informational regarding the importance of weights. Importantly, our method allows for a quite aggressive pruning strategy while still achieving on-par results with the full dense model. The idea of pruning entire rows enables a straightforward reduction in memory and computational footprints on GPU and other modern AI accelerators with small to no additional effort involved. In future work, it would be interesting to analyze the sparsity patterns achieved from Variance Pruning in terms of language model interpretability, for instance, to understand the reason that the attention output and the intermediate layers have high and low sparsity in the middle layers, respectively. Furthermore, it would be also interesting to combine our method with other pruning methods, to find better rows to prune, for instance in an ensemble of pruning methods, which may eliminate any performance degradation.

REFERENCES

Aojun Zhou, J. Z. J. L. Z. Z. K. Y. W. S. H. L., Yukun Ma. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021.

Brown, T., Mann, B., Ryder, N., et al. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics.

Chen, T., Frankle, J., Chang, S., et al. The lottery ticket hypothesis for pre-trained bert networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 15834–15846. Curran Associates, Inc., 2020.

Dalvi, F., Sajjad, H., Durrani, N., and Belinkov, Y. Analyzing redundancy in pretrained transformer models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4908–4926, Online, November 2020. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Durrani, N., Sajjad, H., Dalvi, F., and Belinkov, Y. Analyzing individual neurons in pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4865–4880, Online, November 2020. Association for Computational Linguistics.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Guo, D., Rush, A., and Kim, Y. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4884–4896, Online, August 2021. Association for Computational Linguistics.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015.

Horowitz, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

Kovaleva, O., Romanov, A., Rogers, A., and Rumshisky, A. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4365–4374, Hong Kong, China, November 2019. Association for Computational Linguistics.

Lee, N., Ajanthan, T., and Torr, P. H. S. Snip: Single-shot network pruning based on connection sensitivity, 2019.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization, 2018.

Lu, J., Batra, D., Parikh, D., and Lee, S. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, pp. 13–23, 2019.

Lu, J., Goswami, V., Rohrbach, M., Parikh, D., and Lee, S. 12-in-1: Multi-task vision and language representation learning. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Luo, J., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5068–5076, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society.

Nvidia. Nvidia A100 tensor core gpu architecture. `https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf`, 2020.

Prasanna, S., Rogers, A., and Rumshisky, A. When BERT Plays the Lottery, All Tickets Are Winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3208–3229, Online, November 2020. Association for Computational Linguistics.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

Rosset, C. Turing-nlg: A 17-billion-parameter language model by microsoft. *Microsoft Research Blog*, 2:13, 2020.

Ruder, S., Peters, M. E., Swayamdipta, S., and Wolf, T. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pp. 15–18, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Sajjad, H., Durrani, N., and Dalvi, F. Neuron-level interpretation of deep nlp models: A survey, 2021.

Sakaguchi, K., Le Bras, R., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. 34:8732–8740, Apr. 2020.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020a.

Sanh, V., Wolf, T., and Rush, A. Movement pruning: Adaptive sparsity by fine-tuning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 20378–20389. Curran Associates, Inc., 2020b.

Vaswani, A., Shazeer, N., Parmar, N., et al. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., et al. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Wang, A., Singh, A., Michael, J., et al. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.

Wang, Y., Ye, S., He, Z., et al. Non-structured dnn weight pruning considered harmful. *ArXiv*, abs/1907.02124, 2019.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Wolf, T., Debut, L., Sanh, V., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics.

Wu, J., Belinkov, Y., Sajjad, H., et al. Similarity analysis of contextual word representation models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4638–4655, Online, July 2020. Association for Computational Linguistics.

Yuan, L., Chen, Y., Wang, T., et al. Tokens-to-token vit: Training vision transformers from scratch on imagenet, 2021.

Zhu, Y., Kiros, R., Zemel, R., et al. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27, 2015.