Probing Structured Pruning on Multilingual Pre-trained Models: Settings, Algorithms, and Efficiency

Yanyang Li[1]*, **Fuli Luo**[2], **Runxin Xu**[3], **Songfang Huang**[2], **Fei Huang**[2], **Liwei Wang**[1]
[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2]Alibaba Group
[3]Key Laboratory of Computational Linguistics, Peking University, MOE, China
{yyli21,lwwang}@cse.cuhk.edu.hk, runxinxu@gmail.com
{lfl259702,songfang.hsf,f.huang}@alibaba-inc.com

Abstract

Structured pruning has been extensively studied on monolingual pre-trained language models and is yet to be fully evaluated on their multilingual counterparts. This work investigates three aspects of structured pruning on multilingual pre-trained language models: settings, algorithms, and efficiency. Experiments on nine downstream tasks show several counter-intuitive phenomena: for settings, individually pruning for each language does not induce a better result; for algorithms, the simplest method performs the best; for efficiency, a fast model does not imply that it is also small. To facilitate the comparison on all sparsity levels, we present *Dynamic Sparsification*, a simple approach that allows training the model once and adapting to different model sizes at inference. We hope this work fills the gap in the study of structured pruning on multilingual pre-trained models and sheds light on future research.

1 Introduction

Large-scale pre-trained monolingual language models like BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) have shown promising results in various NLP tasks while suffering from their large model size and high latency. Structured pruning has proven to be an effective approach to compressing and accelerating these large monolingual language models (Michel et al., 2019; Wang et al., 2020c; Prasanna et al., 2020; Liang et al., 2021), making them practical for real-world applications.

Similarly, multilingual pre-trained models (Conneau and Lample, 2019; Conneau et al., 2020; Xue et al., 2021; Luo et al., 2021) are also powerful and even have more parameters. However, little attention has been paid to evaluating the effectiveness of structured pruning on these multilingual models. Applying pruning to multilingual pre-trained models is non-trivial, as it typically involves many languages and needs to carefully design the roles of modules within the network. For example, most attention heads have little impact on the performance of monolingual pre-trained models (Michel et al., 2019; Voita et al., 2019), while it is the opposite for multilingual pre-trained models (See Section 5.3 and also Budhraja et al. (2021)).

This work intends to examine how structured pruning reacts to multilingual pre-trained models. We take the most representative multilingual pre-trained model family, XLM-R (Conneau et al., 2020; Goyal et al., 2021) for our case study and evaluate the pruning performance on nine cross-lingual understanding tasks in XTREME (Hu et al., 2020). We investigate three aspects of structured pruning: settings, algorithms, and efficiency.

Settings Traditional pruning produces a single small model, which is shared across languages (`shared` setting). Recent work on multilingual translation (Li et al., 2020; Lin et al., 2021; Xie et al., 2021; Gong et al., 2021) suggests that tailoring pruning to one language could achieve better results (`non-shared` setting). However, our comprehensive experiments show that neither of the two settings can consistently outperform the other one (See Section 5.2).

Algorithms There exists a broad spectrum of pruning algorithms (Hoefler et al., 2021), and it is impossible to test all of them considering the cost of pre-training. We focus on two pruning algorithms that have been studied the most in monolingual pre-trained models: the regularization-based pruning (Louizos et al., 2018; Wang et al., 2020c) (and our improved version) and the gradient-based pruning (Michel et al., 2019; Prasanna et al., 2020; Liang et al., 2021) (See Section 4). We experimentally find that the simplest gradient-based pruning is more effective for XLM-R (See Section 5.2).

Efficiency One meaningful way to measure pruning algorithms is to study how the performance and

* Collaborated work while doing an Alibaba DAMO Academy internship.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 1852 - 1865

speed of the pruned model vary with the sparsity (Hoefler et al., 2021). However, most pruning algorithms, including those we study in this work, require training the model for each specific sparsity. This limitation makes comparisons against a range of sparsity levels infeasible due to the prohibitive training cost. To solve this issue, we propose the *Dynamic Sparsification* (DS for short), a simple method that parameterizes subnetworks at any sparsity level and shares their weights afterward (See Section 6.1). DS only trains the model once but can obtain models at any sparsity level during inference. Experiments on XNLI (Conneau et al., 2018) show that DS does not degrade the performance much while dramatically reducing the training cost. Interestingly, we observe that the model size and inference speed are not strongly correlated in XLM-R. This observation suggests that one could not obtain a fast model by simply making the model small by using vanilla pruning algorithms (See Section 6.2).

2 Related Work

Settings Recent multilingual translation research suggests that adapting subnetworks for each language or language pair rather than for all of them gives better results. Among them, Li et al. (2020) train a shared multilingual model, then select layers for each language pair. Lin et al. (2021) also prune a shared multilingual model for each language pair, though on the level of entries in weight matrices. Instead, Gong et al. (2021) prune attention heads and feedforward networks for each language. Xie et al. (2021) first identify general and language-specific neurons in a shared multilingual network, then tune those neurons using the data of their corresponding language only. These findings inspire us to extend from multilingual translation to see how `non-shared` pruning settings work on multilingual pre-training.

Algorithms There are many structured pruning techniques proposed for monolingual pre-trained language models recently. Michel et al. (2019) propose a simple gradient-based importance score to prune attention heads. Prasanna et al. (2020); Liang et al. (2021) extend to prune other components like the feedforward network of the Transformer (Vaswani et al., 2017). Wang et al. (2020c) decompose the pre-trained model weights and apply L_0 regularization (Louizos et al., 2018) to regulate the ranks of decomposed weights. Sajjad et al. (2020) study layer pruning and show that directly dropping the top layers performs the best in fine-tuning. Peer et al. (2021) further show that by carefully choosing layers to drop, structured pruning can achieve a performance close to those trained by knowledge distillation (Hinton et al., 2015).

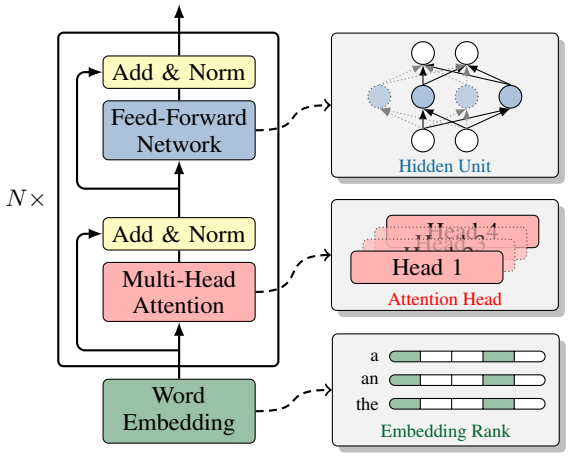

Figure 1: The left is the Transformer encoder, the right is the components that will be pruned at each layer.

Efficiency The pruning algorithms mentioned above need to train one network for each sparsity level used at inference. Hou et al. (2020) propose a dynamic structured pruning method based on Michel et al. (2019), which allows training the model once and making the inference with any size of the model. Compared with our Dynamic Sparsification, Hou et al. (2020)'s method cannot be applied to the `non-shared` setting as it needs to rearrange the network, i.e., producing a new model, for each language. Cascading methods (Schwartz et al., 2020; Xin et al., 2020) can even adapt the network size for each instance. Since cascading methods cannot perform batch inference and are only available for sentence classification tasks, we do not consider them in this work.

3 Background

In this section, we briefly review the structure of XLM-R (Conneau et al., 2020), a Transformer encoder (Vaswani et al., 2017) pre-trained by masked language modeling task (Devlin et al., 2019). We also revisit how conventional structured pruning algorithms are applied to Transformers by introducing additional gating variables and setting appropriate values to them (See Figure 1 and also Prasanna et al. (2020); Liang et al. (2021)). The XLM-R model consists of N layers. Each layer is made of the multihead attention and feedforward

networks, followed by the residual connection and layer normalization.

Attention Following Michel et al. (2019)'s formula, the multihead attention is written as:

$$\mathrm{MHA}(X) = \sum_{i=1}^{H} G_{h,i}\mathrm{head}_i \tag{1}$$

where H is the number of heads, head_i is the output of i-th head and $G_{h,i}$ is the i-th entry of the gating variables $G_h \in \mathbb{R}^H$. $G_{h,i}$ indicates whether the head i will be pruned. $G_{h,i}$ is set to 1 to retain that head and 0 if to drop it. Different pruning algorithms will have their own ways to determine the values of G_h.

Feedforward Network The feedforward network contains two linear projections with GeLU activation (Hendrycks and Gimpel, 2016) in between:

$$\mathrm{FFN}(X) = (\mathrm{GeLU}(XW_1 + b_1) \odot G_f)W_2 + b_2 \tag{2}$$

where $W_1 \in \mathbb{R}^{d\times d_f}$, $b_1 \in \mathbb{R}^{d_f}$, $W_2 \in \mathbb{R}^{d_f\times d}$ and $b_2 \in \mathbb{R}^d$ are weights of the feedforward network and d_f is the hidden size. $\odot$ denotes the Hadamard product and $G_f \in \mathbb{R}^{d_f}$ is a gating vector with a value in the range of [0, 1]. G_f functions similar to G_h in multihead attention, except that G_f controls the activation of hidden units.

Embedding To prune the large embedding matrix E (occupying 69% of all parameters), we decompose it via low-rank approximation as in Lan et al. (2020):

$$E = \hat{E}\,\mathrm{diag}(G_e)P \tag{3}$$

where $\hat{E} \in \mathbb{R}^{v\times d}$ and $P \in \mathbb{R}^{d\times d}$ are the decomposed matrices of E. v is the vocabulary size. $G_e \in \mathbb{R}^d$, governing the rank of E, is a gating vector similar to G_h and G_f. $\mathrm{diag}(G_e)$ converts G_e to a diagonal matrix. The right part of Figure 1 is an illustration of the components (such as hidden units, attention heads, and embeddings) that will be pruned.

4 Extending Pruning Algorithms to Pruning Settings

This section will first introduce pruning algorithms that we study and then describe how to adapt them to two pruning settings. The first is the `shared` setting that shares the pruned network across languages (default setting that all pruning algorithms could run on), and the second is the `non-shared` setting that prunes one subnetwork for each language (Xie et al., 2021; Gong et al., 2021).

4.1 Gradient-based Pruning

Gradient-based pruning (Michel et al., 2019) computes the importance score of each component, e.g., heads in Eq. 1. Then it sets the gating variable of a component, e.g., $G_{h,i}$ in Eq. 1, to 1 if its importance score is larger than a threshold and 0 otherwise. Taking an attention head i as an example, its importance score is defined as:

$$I_{\mathrm{head}_i} = \mathbb{E}_{X\sim\mathbf{X}}\left|\mathrm{head}_i^T \frac{\partial \mathcal{L}_{\mathrm{MLM}}(X)}{\partial \mathrm{head}_i}\right| \tag{4}$$

where $\mathbf{X}$ is the data distribution and we choose the validation set as $\mathbf{X}$ in practice, $\mathcal{L}_{\mathrm{MLM}}$ is the masked language modeling loss (Devlin et al., 2019). The values of gating variables are set and frozen after pre-training. An additional phase of pre-training is further employed to update network parameters to recover performance loss brought by pruning.

Extending gradient-based pruning to the `non-shared` setting is straightforward: to prune for one language, we use data of that language to compute a unique set of gating variables $G = \{G_h, G_f, G_e\}$ for it.

4.2 Regularization-based Pruning

The L_0 norm has been widely used in many areas, including signal processing (Zhang, 2010; Xu et al., 2011) to induce sparsity. In neural networks, regularization-based pruning, also referred to as L_0 regularization (Louizos et al., 2018), defines a differentiable L_0 norm on the gating variables $G = \{G_h, G_f, G_e\}$. It controls the network sparsity by learning the values of G during pre-training. Taking a gating variable $g \in G$ as an example, it is modeled as:

$$u \sim U(0,1) \tag{5}$$
$$s = \mathrm{sigmoid}((\log u/(1-u) + \alpha)/\beta) \tag{6}$$
$$\hat{s} = s \times (r-l) + l \tag{7}$$
$$g = \min(1, \max(0, \hat{s})) \tag{8}$$

where U is the uniform distribution, $l < 0$ and $r > 1$ are two fixed constants, β is the temperature and α is a learnable parameter of g. During training, u is sampled for each g separately. At inference,

Eq. 6 becomes $s = \text{sigmoid}(\alpha)$. Compared with gradient-based pruning, the importance score in L_0 regularization is the learnt α and the threshold is fixed to $\text{sigmoid}^{-1}\left(-\frac{l}{r-l}\right)$.

The L_0 regularization term of g is:

$$||g||_0 = \text{sigmoid}\left(\alpha - \log(-l/r)\right) \tag{9}$$

and the overall L_0 regularization term is[1]:

$$\mathcal{L}_{L_0} = ||G||_0 = \sum_{g \in G} ||g||_0 \tag{10}$$

$\mathcal{L}_{L_0}$ will be multiplied by a hyper-parameter λ_1 and added to the pre-training loss $\mathcal{L}_{\text{MLM}}$.

4.2.1 Improved L_0 Regularization

Two issues of the previous native L_0 regularization emerge in practice: 1) The hyper-parameter λ_1 does not relate to the model sparsity. It requires several expensive try-outs training runs to find an appropriate setup that can reach desired sparsity (Wang et al., 2020c). 2) If we extend L_0 regularization to `non-shared` setting as done in gradient-based pruning, it easily converges to an optimum where every language shares the network (Gong et al., 2021). This falls back to the `shared` setting. Thus, we propose two corresponding solutions as below:

1) Sparsity Constraint To address the first issue, we add a sparsity constraint to Eq. 10:

$$\mathcal{L}_{L_0} = \sum_{i=1}^{l} \left| ||G^i||_0 - t \right| \tag{11}$$

where l is the number of languages and G^i denotes the set of gating variables for language i. This loss term will keep the subnetwork size of each language close to the targeted size t.[2]

2) Diverse Subnetwork To address the second issue, we introduce a diversity loss term to encourage the model to find a distinct subnetwork for each language. It is achieved by diagonalizing the gram matrix of gating variables $\bar{G} = [G^1; \cdots; G^l]$:

$$\mathcal{L}_{\text{diag}} = ||P \odot \bar{G}\bar{G}^T \odot (\mathbf{1} - \mathbf{I})||_1 \tag{12}$$

where $\mathbf{1}$ is a matrix of ones and $\mathbf{I}$ is the identity matrix. $P \in \mathbb{R}^{l \times l}$ is used to introduce linguistic prior and is a matrix of ones by default.

Eq. 12 will penalize each language pair equally. Intuitively, the subnetworks of two languages that are close, e.g., English and Spanish, should not be penalized. Thus we add linguistic prior $P_{ij} = 0$ when the i-th and j-th languages belong to the same language family (See Appendix C) and 1 otherwise.

To the end, the loss $\mathcal{L}$ we used in pre-training is:

$$\mathcal{L} = \mathcal{L}_{\text{MLM}} + \lambda_1 \mathcal{L}_{L_0} + \lambda_2 \mathcal{L}_{\text{diag}} \tag{13}$$

Note that the parameter of the gating variable α is randomly initialized. We find that tuning only α in the first few epochs is crucial to obtain better performance. If no further notice, we will use this improved L_0 regularization for experiments with `non-shared` setting and the native L_0 regularization for `shared` setting.

5 Empirical Study of Algorithms and Settings for Multilingual Pruning

5.1 Experimental Setup

Pre-training Our pruned models are trained on the CC-100 corpus (Wenzek et al., 2020). We choose 100 languages with a total size of 2.2TB for training, which is consistent with those used in XLM-R (Conneau et al., 2020). The development set we used to induce the importance score for pruning is 3K randomly selected samples from the CC-100 corpus per language.

Our model is a 12-layer Transformer with a 768 embedding size and a 3072 hidden size. It is pruned and continually trained based on the publicly available XLM-R model for 150K steps with a batch size of 2048 and a learning rate of 0.0002. Other hyper-parameters remain the same as in the original paper (Conneau et al., 2020). We train our model on 32 Nvidia Tesla V100 32GB GPUs with mixed-precision training. It takes roughly 7-10 days to pre-train one model. For inference, we use 1 Nvidia Tesla V100 32GB GPU and Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50GHz to estimate the GPU and CPU throughput (with a batch size of 128 for GPU and 1 for CPU).

Fine-tuning We evaluate the pruned models on 9 downstream tasks from XTREME (Hu et al., 2020). These tasks can be classified into four different categories: (1) sentence-pair classification: XNLI (Conneau et al., 2018), PAWS-X (Yang

[1] In practice we weigh the L_0 regularization term of gating variables (See Appendix B).

[2] Adding a Lagrange multiplier (Wang et al., 2020c) is also doable, but we find this simple L_1-like loss is similarly effective and easy to implement.

Task Metrics #Languages	Sparsity	XNLI Acc. 15	PAWS-X Acc. 7	POS F1 33	NER F1 40	XQuAD F1/EM 11	MLQA F1/EM 7	TyDiQA F1/EM 9	BUCC F1 5	Tatoeba Acc. 33	**Avg**
Cross-lingual Transfer: *Fine-tune model on English training set and test on all languages.*											
XLM-R	0%	74.8	85.4	74.0	61.9	69.2/53.0	59.9/44.3	51.3/32.4	63.3	53.4	60.2
DistilBERT	50%	70.3	82.9	72.1	56.1	60.5/44.3	52.4/37.4	39.4/23.0	44.2	45.3	52.3
L_0 (non-shared)	50%	68.6	83.3	68.3	53.4	59.8/43.2	49.6/34.6	35.2/19.8	52.5	43.8	51.0
L_0 (shared)	20%	65.3	80.9	68.4	52.0	54.8/38.7	45.7/30.7	26.8/13.5	34.2	41.1	46.0
Grad (non-shared)	50%	68.6	83.9	68.3	53.9	60.6/44.2	52.3/36.7	40.5/22.6	**57.5**	**48.6**	53.1
Grad (shared)	50%	**70.4**	**84.7**	**72.4**	**57.4**	**64.2/48.3**	**56.1/40.5**	**45.2/28.0**	46.6	40.5	**54.5**
Translate-Train-All: *Fine-tune model on English training data and translated data of other languages.*											
XLM-R	0%	79.1	89.2	89.5	88.0	72.7/58.2	58.2/42.8	72.1/57.5	-	-	70.7
DistilBERT	50%	75.8	87.3	**88.9**	87.1	69.0/54.3	55.0/39.6	68.6/53.7	-	-	67.9
L_0 (non-shared)	50%	76.3	87.8	87.9	86.8	69.3/54.2	54.7/39.2	67.8/52.5	-	-	67.7
L_0 (shared)	20%	73.4	86.0	87.5	85.1	65.1/50.1	51.2/35.6	61.2/45.9	-	-	64.1
Grad (non-shared)	50%	76.6	88.2	87.3	86.6	68.9/53.6	55.2/39.5	68.6/53.7	-	-	67.8
Grad (shared)	50%	**76.8**	**88.4**	88.4	**88.0**	**70.1/55.0**	**56.7/40.7**	**69.5/54.6**	-	-	**68.8**

Table 1: XTREME results (Sparsity is the portion of dropped parameters in the Transformer encoder, and thus higher sparsity denotes smaller size.). We compare one representative distillation method (denoted as `DistilBERT`, Sanh et al. (2019)) and two representative structured pruning methods: gradient-based pruning (denoted as `Grad`) and regularization-based pruning (denoted as L_0), under two settings (described in Section 4: `shared` and `non-shared`). **Bold** denotes the best results among 50% sparsity. Note that since BUCC and Tatoeba do not have the translated training data, we do not report their translate-train-all results.

et al., 2019); (2) structured prediction: POS (Nivre et al., 2018), Wikiann NER (Pan et al., 2017); (3) question answering: XQuAD (Artetxe et al., 2020), MLQA (Lewis et al., 2020), TyDiQA (Clark et al., 2020); (4) sentence retrieval: BUCC2018 (Zweigenbaum et al., 2017), Tatoeba (Artetxe and Schwenk, 2019). The hyper-parameter setup of fine-tuning could be found in Appendix A.

Following previous work (Hu et al., 2020), we study the pruned models in two fine-tuning settings: *Cross-lingual Transfer* (a.k.a., zero-shot) and *Translate-Train-All* (a.k.a., multi-task). Note that for the two sequence labelling tasks POS and NER, translation cannot give us the correct training labels. We thus use human-annotated data for translate-train-all training on them.

5.2 Results

Table 1 shows the fine-tuning results of using different methods to prune XLM-R to 50% sparsity (also the value of t in Eq. 11). We follow the convention of Prasanna et al. (2020) to compute the sparsity of the encoder, which excludes the embeddings in the calculation. For `DistilBERT`, we remove half of the original layers of XLM-R as done in Sanh et al. (2019). Note that in Table 1 (the rows of "L_0 (shared)"), regularization-based pruning with `shared` setting has a lower sparsity (20%).[3]

[3]We have tried various hyper-parameters settings to pre-train models toward 50% sparsity (for a fair comparison with `DistilBERT`) using vanilla L_0, but the resulting sparsity is either too high ($\geq$70%) or too low ($\leq$20%). This is in line with the trainability issue of L_0 as indicated in Section 4.2.

Methods	Sparsity	XNLI	POS	NER	TyDiQA	**Avg**
L_0	20%	73.4	87.5	85.1	61.2/45.9	74.9
Impv. L_0	50%	76.3	**87.9**	**86.8**	67.8/52.5	77.8
Impv. L_0 + Distil	50%	**76.4**	87.5	86.7	**69.5/54.6**	**78.2**

Table 2: The results of the improved L_0 (Impv. L_0) regularization-based pruning (See Section 4.2.1).

Gradient-based pruning performs better than regularization-based pruning. Table 1 shows that vanilla L_0 in `shared` setting has more parameters (20% sparsity) but performs worse than gradient-based pruning with fewer parameters (50% sparsity). Despite that our proposed improved L_0 works better (`non-shared` setting), it still underperforms the gradient-based pruning counterpart. This is because regularization-based pruning keeps modifying the subnetwork structure when weights are updating, which might introduce too much noise during training. Gradient-based pruning, on the other hand, keeps the pruned network unchanged and adapts weights only. Despite that some works (Hoefler et al., 2021) suggest that regularization-based pruning should be preferred, it might not be the same conclusion for XLM-R.

Neither of the pruning settings performs consistently better. Previous work on multilingual

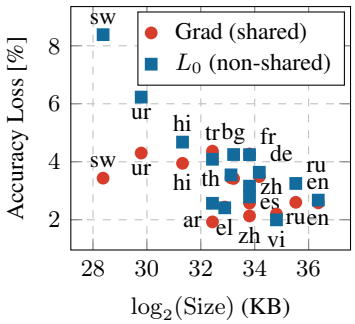

Figure 2: Accuracy loss on each language of XNLI vs. the logarithm of their pre-training corpus sizes.

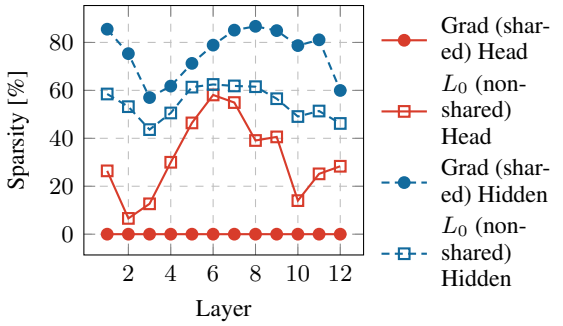

Figure 3: Sparsity of each layer pruned by two pruning algorithms.

translation has suggested that `non-shared` setting provides consistent gains, as this way allows the pruned model to adapt for each language (Li et al., 2020; Lin et al., 2021; Xie et al., 2021; Gong et al., 2021). However, this is not the case for XLM-R. As shown in Table 1, regularization-based pruning (L_0) works the best with the `non-shared` settings[4], but for gradient-based pruning it is the `shared` setting. We analyze that this is because XLM-R covers more low-resource languages (100 languages in XLM-R vs. 24 in most multilingual translation research), which makes sharing the subnetwork for a universal representation more preferable (Aharoni et al., 2019).

Simple distillation performs less effective than pruning. For most tasks, distillation is not as effective as pruning.[5] This might be that distillation prunes a whole layer, while more fine-grained components are pruned in structured pruning. But combining distillation with pruning could provide some gain, as shown in Table 2.

Our improved L_0 regularization-based pruning can further boost the performance. In Section 4.2.1, we propose an improved L_0 regularization to solve the drawbacks of standard L_0. Table 2 shows the results. Through the sparsity constraint, we can control the model sparsity to be the desired value $t = 50\%$ instead of 20% (the closest we could have using vanilla L_0). And along with diverse subnetwork, the improved L_0 can even consistently improve the fine-tuning results. Appendix E visualizes how subnetworks differ between two languages after applying the diversity loss term. Moreover, integrating with distillation (the last row of Table 2) can further improve the results.

5.3 Analysis

Why does regularization-based pruning perform poorly? Since regularization-based pruning learns the subnetwork from scratch, we believe its poor performance results from the low-resource languages. We choose XNLI with the translate-train-all setting for empirical verification. On the one hand, the translate-train-all setting ensures that each language has the same dataset for fine-tuning (except for NER and POS). This way eliminates the difference in fine-tuning. On the other hand, among all tasks except NER and POS, XNLI covers more languages.

Figure 2 supports our hypothesis. It shows the accuracy loss and corpus size of each language in regularization-based and gradient-based pruning. We observe that for regularization-based pruning accuracy loss strongly correlates with pre-training dataset size (a value of 0.83 for Pearson's τ), while it is not for gradient-based pruning.

Where does pruning methods behave differently? In Figure 3, we compare in which aspect different pruning algorithms behave differently. Figure 3 shows the sparsity of each component (attention heads and hidden units) at each layer. Interestingly, we see that gradient-based pruning preserves all attention heads and only a tiny number of hidden units, while regularization-based pruning prunes heads and hidden units more evenly. Though previous works (Michel et al., 2019; Voita et al., 2019) have suggested that most attention heads have little impact on the final performance of monolingual models, our results show that this is not the case for XLM-R. Besides, both pruning methods tend to drop more in the middle layers.

[4]Non-shared model with more parameters dropped (50% sparsity) is better than shared model with fewer parameters dropped (20% sparsity).

[5]Although adopting advanced distillation techniques might improve the result, the pruning algorithm is also simple here.

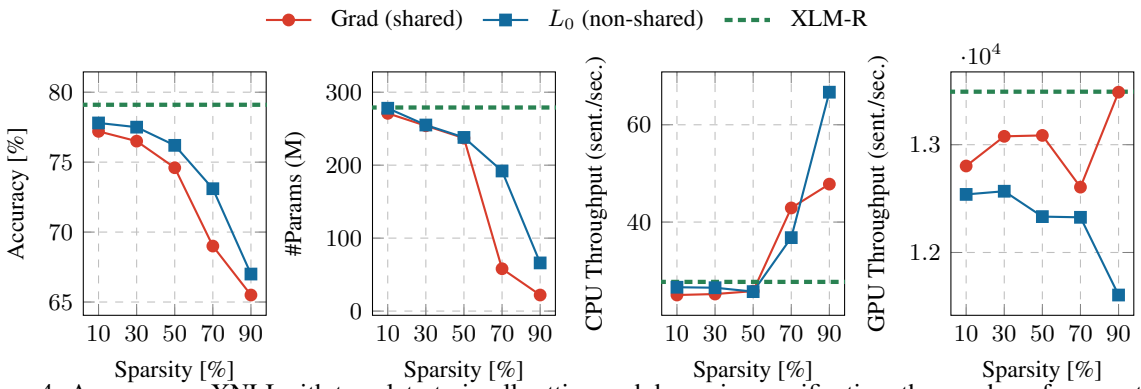

Figure 4: Accuracy on XNLI with translate-train-all setting and dynamic sparsification, the number of parameters (#Params), CPU and GPU throughput (the number of sentences per second) vs. the sparsity.

6 Toward Efficient Pruning

6.1 Dynamic Sparsification

In practice, we may need models with different sparsities to fit various resource constraints or compare a set of methods. Nevertheless, existing pruning techniques must train the model independently for each sparsity level, which is prohibitive for large models. Here we propose *Dynamic Sparsification* (**DS** for short), a method that trains the model once but allows inference with any level of sparsity.

Section 4 shows that both gradient-based and regularization-based pruning follow the same procedure: we first determine a threshold, then get the importance score for each component, and set the gating variable to 1 if its score is larger than that threshold and 0 otherwise. By adjusting the threshold, one can obtain networks with any sparsity.

Based on this, we model a gating variable g as:

$$g = f(\alpha + t\theta) \tag{14}$$

where α is a trainable importance score as in regularization-based pruning, t is the targeted network size (which is one minus the sparsity), $t\theta$ is the threshold with a learnable θ, f is a function with output ranging between 0 and 1. We choose f to be Eqs. 6 - 8 because it enables us to optimize α and θ via L_0 regularization. If α and θ are set properly, Eq. 14 will automatically determine whether its corresponding component should be activated under the targeted network size t.

Then is how to find α and θ using pruning algorithms. We know that pruning algorithms could rank different components by their importance scores. Based on this ranking, we identify the boundary network size that a specific component will be activated (denoted as $\hat{t}$) and will not. These two conditions form a system of linear equations in two unknowns α and θ:

$$\begin{cases} f\left(\alpha + \hat{t}\theta\right) = 1 \\ f\left(\alpha + \left(\hat{t} - \delta\right)\theta\right) = 0 \end{cases} \tag{15}$$

where δ is the network size that one component contributes to, $\hat{t}$ is the boundary network size where the corresponding gating variable g should be 1 if $t > \hat{t}$ and 0 if $t < \hat{t} - \delta$. $\hat{t}$ equals the ranking divided by the total number of components. Eq. 15 has a closed-form solution for α and θ:[6]

$$\begin{cases} \alpha = \left(1 - \hat{t}/\delta\right) f^{-1}(1) + (\hat{t}/\delta) f^{-1}(0) \\ \theta = \left(f^{-1}(1) - f^{-1}(0)\right)/\delta \end{cases} \tag{16}$$

Before training, we use gradient-based pruning to initialize α and θ via Eq. 16. If only gradient-based pruning is adopted, α and θ are then clamped and only the retained network parameters will be updated, otherwise they can be jointly optimized via regularization-based pruning. During training, we sample different ts to train different sized sub-networks. At inference, t is set to the targeted network size to prune the model. If one wants to extend DS to `non-shared` setting, he can prune for each language once and compute a unique set of α and θ for each language.

Methods	XNLI	POS	NER	TyDiQA	Avg
Grad (shared)	**76.8**	**88.4**	**88.0**	**69.5/54.6**	**78.8**
+ DS	74.6	87.6	87.1	64.0/48.3	76.4
L_0 (non-shared)	**76.3**	**87.9**	**86.8**	67.8/52.5	**77.8**
+ DS	76.2	**87.9**	86.7	**67.9/52.4**	77.7

Table 3: The results of gradient-based and regularization-based pruning with or without dynamic sparsification (Sparsity=50%).

[6] Eq. 16 has the numerical stability issue and weighs different components equally (See Appendix D for the solution).

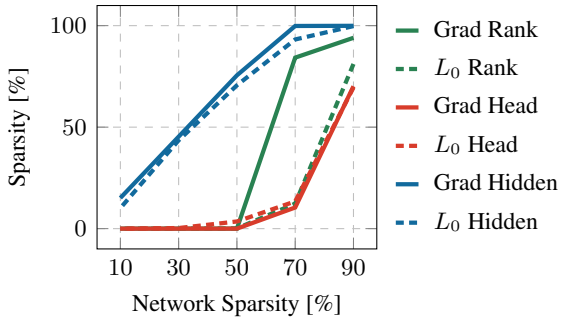

Figure 5: Sparsity of different components pruned by two pruning algorithms vs. the sparsity.

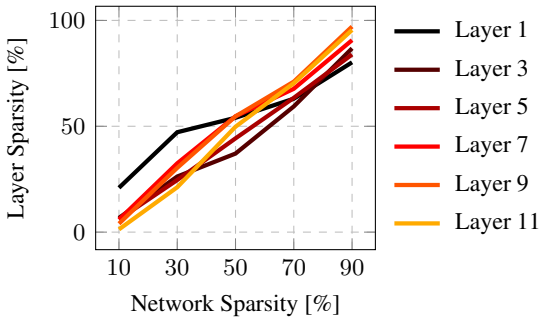

Figure 6: Sparsity of different layers pruned by regularization-based pruning vs. the sparsity.

6.2 Main Results

Table 3 (+ `DS` rows) shows the 50% sparsity results after applying DS to the two pruning algorithms under their best performing pruning settings (according to Table 1). Surprisingly, we observe that gradient-based pruning with `shared` setting suffers from a significant loss, while regularization-based pruning with `non-shared` setting has almost no loss. This is because DS shares the weights between subnetworks of different sparsities hurts the model capacity, and `non-shared` setting enlarges the subnetwork capacity by untying weights of different languages. Due to the expensive cost of training models without DS, we only test the impact of DS on 50% sparsity, but we compare it with other systems with a smaller size (See Appendix F). The leftmost part of Figure 4 shows more on how the two pruning methods trade accuracy for efficiency under various sparsities.

The second sub-figure from the left of Figure 4 shows a non-linear relationship between the number of parameters and sparsity, as embeddings are not included in sparsity calculation (Prasanna et al., 2020). Since embeddings are more important than most parts of the model and are very large (69% of the overall parameters), the number of parameters remains high even when the encoder is quite sparse (Sparsity $\leq 50\%$). Pruning algorithms only start to prune these large embeddings when the encoder is very sparse (Sparsity $> 50\%$) and results in a great drop in the number of parameters, as shown in Figure 5.

The two rightmost panels of Figure 4 describe how the CPU and GPU throughput vary as the sparsity changes. We observe a strong correlation between the CPU throughput and sparsity when the sparsity $\geq 50\%$. However, there is no such trend observed when the sparsity $< 50\%$. This might be due to the time consumption of irregular memory access out-weights the speed-up brought by the small tensor computation.

Interestingly, we see that sparse models show no acceleration on GPU even when the sparsity is high (e.g., 90%). Although pruning algorithms here optimize the model size instead of inference efficiency, it is expected that the resulting sparse models still have speedup as shown in CPU and in other work (Wang et al., 2020c). In Figure 6, we find that the highest sparsity of all layers is close to but not exactly 100%. This implies that **pruning tends to produce a deep and narrow model.** Previous studies (Sanh et al., 2019; Wang et al., 2020a; Li et al., 2021) show that GPU throughput is more sensitive to the model height instead of its width. This explains why we did not observe any acceleration even for a model with $1/10$ of the original size.

Though not shown in Table 1 and Figure 4, it is still possible to obtain actual speedup in GPU for sparse models. **Previous observations on GPU throughput only hold for inference with the same batch size**. In practice, the sparse models have a smaller memory footprint and we can use a larger batch size for higher parallelism. For pruned models in Table 1, a nearly $2\times$ speedup is observed when we double the inference batch size.

In summary, Figure 4 suggests that **the correlation between the model size and throughput is very week for XLM-R**: for model size, reducing the embedding size is important, but it has almost no impact on throughput (an $O(1)$ complexity table lookup); for throughput, compressing parts other than embeddings is more effective as shown in Figure 4, but they have much fewer parameters than the embeddings (193M parameters for embeddings vs. 86M for the others). This advocates special

care needed to be taken if one wants to compress and accelerate XLM-R simultaneously.

6.3 Analysis

Here we study what DS will prune under various sparsities. Figure 5 shows which component (embeddings, attention heads and hidden units) will be preferred during pruning. In general, gradient-based pruning behaves similar to regularization-based pruning: they first prune hidden units, and only prune attention heads and embeddings when the sparsity is high. The main difference between them is that gradient-based pruning starts to prune embeddings earlier (at 70% sparsity) than regularization-based pruning. This explains why we observe a significant drop in performance for gradient-based pruning with 70% sparsity (See the left of Figure 4): the model already lost much information at the beginning and there is no way to recover.

Figure 6 shows how regularization-based pruning prunes each layer with DS. Though we do not plot the curves of gradient-based pruning, its phenomenon is similar to regularization-basd pruning. We find that regularization-based pruning behaves differently at low and high sparsity. It first prunes bottom layers when the sparsity is low, then gradually shift to higher layers as the sparsity increases. In the end, it retains more parameters in the bottom layers instead of the top layers. This provides insight for future model design: **a pyramid structure is better when the model size is very small**.

7 Conclusion

In this work, we study three aspects of structured pruning on multilingual pre-trained models: settings, algorithms and efficiency. Experiments show interesting phenomena: The best pruning setting depends on the choice of algorithms; The simplest pruning algorithm performs the best; A fast model does not mean it should be small. We hope this work will give insight to future research.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online. Association for Computational Linguistics.

Mikel Artetxe and Holger Schwenk. 2019. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7(0):597–610.

Aakriti Budhraja, Madhura Pande, Pratyush Kumar, and Mitesh M. Khapra. 2021. On the prunability of attention heads in multilingual BERT. *CoRR*, abs/2109.12683.

Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7057–7067.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Hongyu Gong, Xian Li, and Dmitriy Genzel. 2021. Adaptive sparse transformer for multilingual translation. *CoRR*, abs/2104.07358.

Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. 2021. Larger-scale transformers for multilingual masked language modeling. *CoRR*, abs/2105.00572.

Dan Hendrycks and Kevin Gimpel. 2016. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *CoRR*, abs/2102.00554.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 9782–9793. Curran Associates, Inc.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. XTREME: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 4411–4421. PMLR.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2020. MLQA: Evaluating cross-lingual extractive question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7315–7330, Online. Association for Computational Linguistics.

Xian Li, Asa Cooper Stickland, Yuqing Tang, and Xiang Kong. 2020. Deep transformers with latent depth. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021. An efficient transformer decoder with compressed sub-layers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13315–13323. AAAI Press.

Chen Liang, Simiao Zuo, Minshuo Chen, Haoming Jiang, Xiaodong Liu, Pengcheng He, Tuo Zhao, and Weizhu Chen. 2021. Super tickets in pre-trained language models: From model compression to improving generalization. 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)*, pages 6524–6538, Online. Association for Computational Linguistics.

Zehui Lin, Liwei Wu, Mingxuan Wang, and Lei Li. 2021. Learning language specific sub-network for multilingual machine translation. 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)*, pages 293–305, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Christos Louizos, Max Welling, and Diederik P Kingma. 2018. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*.

Fuli Luo, Wei Wang, Jiahao Liu, Yijia Liu, Bin Bi, Songfang Huang, Fei Huang, and Luo Si. 2021. VECO: Variable and flexible cross-lingual pre-training for language understanding and generation. 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)*, pages 3980–3994, Online. Association for Computational Linguistics.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14014–14024.

Joakim Nivre, Rogier Blokland, Niko Partanen, and Michael Rießler. 2018. Universal dependencies 2.2.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. 2017. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada. Association for Computational Linguistics.

David Peer, Sebastian Stabinger, Stefan Engl, and Antonio Jose Rodríguez-Sánchez. 2021. Greedy layer pruning: Decreasing inference time of transformer models. *CoRR*, abs/2105.14839.

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

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. On the effect of dropping layers of pre-trained transformer models. *arXiv preprint arXiv:2004.03844*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6640–6651. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, Online. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2020c. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online. Association for Computational Linguistics.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France. European Language Resources Association.

Wanying Xie, Yang Feng, Shuhao Gu, and Dong Yu. 2021. Importance-based neuron allocation for multilingual neural machine translation. 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)*, pages 5725–5737, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics.

Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. 2011. Image smoothing via L_0 gradient minimization. *ACM Trans. Graph.*, 30(6):174.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. 2019. PAWS-X: A cross-lingual adversarial dataset for paraphrase identification. 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)*, pages 3687–3692, Hong Kong, China. Association for Computational Linguistics.

Tong Zhang. 2010. Analysis of multi-stage convex relaxation for sparse regularization. *J. Mach. Learn. Res.*, 11:1081–1107.

Pierre Zweigenbaum, Serge Sharoff, and Reinhard Rapp. 2017. Overview of the second BUCC shared task: Spotting parallel sentences in comparable corpora. In *Proceedings of the 10th Workshop on Building and Using Comparable Corpora*, pages 60–67, Vancouver, Canada. Association for Computational Linguistics.

A Hyper-parameters

Pre-training We set λ_1 to 8 and λ_2 to 1 for L_0 regularization in 50% sparsity. If Dynamic Sparsification is applied, we set λ_1 to 128 and others remain the same. The number of pre-training steps that tunes α only is 150K.

Fine-tuning We perform a grid search to find the best hyper-parameter setting for each task (except for BUCC and Tatoeba, they do not need training). We list the names of hyper-parameters as well as their search ranges below:

- Learning rate: [1e-6, 2e-6, $\cdots$, 5e-5].
- Epoch: [5, 10] for cross-lingual transfer and 3 for translate-train-all.

We use a batch size of 32 for all experiments.

B Weighting L_0 Regularization

In practice, gating variables g from different components should contribute differently to the overall L_0 regularization term $||G||_0$ in Eq. 10, as they govern different weight matrices. For example, disabling the head i will remove W_q^i, W_k^i, W_v^i and W_o^i, but disabling a hidden unit only eliminate a column of W_1 and a row of W_2. So we weigh the regularization terms from attention heads by 64×4, 2 for those from hidden units and 1 for those from the embedding matrix.

C Language Family

Table 4 is the language family information we used in Section 4. There are 15 different language families and one special `Missing` family in Table 4.

D Implementation of Dynamic Sparsification

Dynamic Sparsification described in Section 6 has two issues:

- It assumes all components in the network contribute equally to the network size. But according to the discussion in Appendix B, different components relate to different numbers of weight matrices and each weight matrix has a different size.
- The solution of α and β provided by Eq. 16 requires high precision in order to precisely activate just a single hidden unit by giving an appropriate sparsity. This fact brings difficulties in mixed-precision training as it easily causes the overflow issue.

Here we describe an improved version of Dynamic Sparsification for practical implementation. The key difference between this improved version and the original one is the way it computes δ (the network size that a component should contribute to) and $\hat{t}$ (the network size where a component should be activated).

For δ, we have:

1. We associate a weight w to each component, as done in Appendix B.
2. Then $\delta = w / \left(\sum_{w' \in \bar{W}} w'\right)$, where $\bar{W}$ is the set of all w.

For $\hat{t}$, we have:

1. We define a set of sparsities $\{s_0, s_2, \cdots, s_n\}$ (in sorted order) to be used at inference where n is the number of all possible sparsities and $s_0 = 0$ and $s_n = 1$, e.g., {0%, 10%, $\cdots$, 100%}.
2. A set of sparsity ranges can then be naturally derived from these sparsities, i.e., $\{s_0 \sim s_1, \cdots, s_{i-1} \sim s_i, \cdots, s_{n-1} \sim s_n\}$. For example, given the set of sparsities {0%, 10%, $\cdots$, 100%}, the set of ranges will be {0%$\sim$10%, 10%$\sim$20%, $\cdots$, 90%$\sim$100%}.
3. For each sparsity range $s_{i-1} \sim s_i$, we find out all components that should be activated in that range, i.e., their original $\hat{t}$ must satisfy $s_{i-1} < \hat{t} \leq s_i$ (considering their actual contributions to the total network size under the weighting scheme in Appendix B), and we denote these set of components as C_i.
4. For all components $c \in C_i$, we assign their $\hat{t} = s_i$.

The way we compute δ resolves the first issue by weighting the contribution to network size for each component. And the way how $\hat{t}$ defined resolves the second issue by constraining the precision of sparsity and thus the precision of α and β. Given $\hat{t}$ and δ, we can use Eq. 16 to induce a solution that is numerical stable.

Language	Family	Language	Family	Language	Family
af	Indo-European	am	Afro-Asiatic	ar	Afro-Asiatic
as	Indo-European	az	Turkic	be	Indo-European
bg	Indo-European	bn	Indo-European	bn-rom	Indo-European
br	Indo-European	bs	Indo-European	ca	Indo-European
cs	Indo-European	cy	Indo-European	da	Indo-European
de	Indo-European	el	Indo-European	en	Indo-European
eo	Constructed language	es	Indo-European	et	Uralic
eu	Language isolate	fa	Missing	fi	Uralic
fr	Indo-European	fy	Indo-European	ga	Indo-European
gd	Indo-European	gl	Indo-European	gu	Indo-European
ha	Afro-Asiatic	he	Afro-Asiatic	hi	Indo-European
hi-rom	Indo-European	hr	Indo-European	hu	Uralic
hy	Indo-European	id	Austronesian	is	Indo-European
it	Indo-European	ja	Japonic	jv	Austronesian
ka	Kartvelian	kk	Turkic	km	Austro-Asiatic
kn	Dravidian	ko	Koreanic	ku	Indo-European
ky	Turkic	la	Indo-European	lo	Kra-Dai
lt	Indo-European	lv	Missing	mg	Missing
mk	Indo-European	ml	Dravidian	mn	Missing
mr	Indo-European	ms	Missing	my-zaw	Sino-Tibetan
my	Sino-Tibetan	ne	Indo-European	nl	Indo-European
no	Indo-European	om	Missing	or	Indo-European
pa	Indo-European	pl	Indo-European	ps	Missing
pt	Indo-European	ro	Indo-European	ru	Indo-European
sa	Indo-European	sd	Indo-European	si	Indo-European
sk	Indo-European	sl	Indo-European	so	Afro-Asiatic
sq	Missing	sr	Indo-European	su	Austronesian
sv	Indo-European	sw	Niger-Congo	ta	Dravidian
ta-rom	Dravidian	te	Dravidian	te-rom	Dravidian
th	Kra-Dai	tl	Austronesian	tr	Turkic
ug	Turkic	uk	Indo-European	ur	Indo-European
ur-rom	Indo-European	uz	Missing	vi	Austro-Asiatic
xh	Niger-Congo	yi	Indo-European	zh-Hans	Sino-Tibetan
zh-Hant	Sino-Tibetan				

Table 4: The language family from https://www.ethnologue.com/. Missing means that there is no language family information of that language found in the website.

System	Sparsity	en	fr	es	de	el	bg	ru	tr	ar	vi	th	zh	hi	sw	ur	Avg
mMiniLMv1	70%	**81.5**	74.8	75.7	72.9	73.0	74.5	71.3	69.7	68.8	72.1	67.8	70.0	66.2	63.3	64.2	71.1
Grad (shared) + DS	70%	69.0	**76.0**	71.9	73.0	70.8	70.3	70.8	70.0	68.4	66.7	**71.0**	66.7	68.1	**65.4**	64.1	62.4
L_0 (non-shared) + DS	70%	80.0	75.3	**75.8**	**74.3**	**74.1**	**74.7**	**74.2**	**71.6**	**70.8**	**74.2**	70.0	**73.1**	**68.7**	65.0	**65.6**	**73.1**

Table 5: XNLI results of mMiniLMv1, gradient-based (Grad) and regularization-based pruning (L_0) with Dynamic Sparsification (DS).

E Language Subnetwork Diversity

Section 4 states that introducing a diversity loss term in Eq. 12 helps to diversify the subnetworks of each language. To measure the distance between these subnetworks, we first choose the gating variables G to represent a subnetwork. We then calculate the Hamming distance between Gs for each language pair. Figure 7 visualizes the results from the model pruned by our improved L_0 regularization. We can see that subnetworks of different languages are indeed different. Some languages are similar like gu and bn, but some are different like bs and om. We also see that even for the most distant language pairs, they are still significantly overlapped (a Hamming distance around 0.3). This indicates that sharing weights between languages is important.

F Comparison with Other Systems

Due to the expensive cost of pre-training models with different sparsities, we only compare the results with and without Dynamic Sparsification at 50% sparsity, as shown in Table 3. Here in Table 5, we compare our models trained by Dynamic Sparsi-

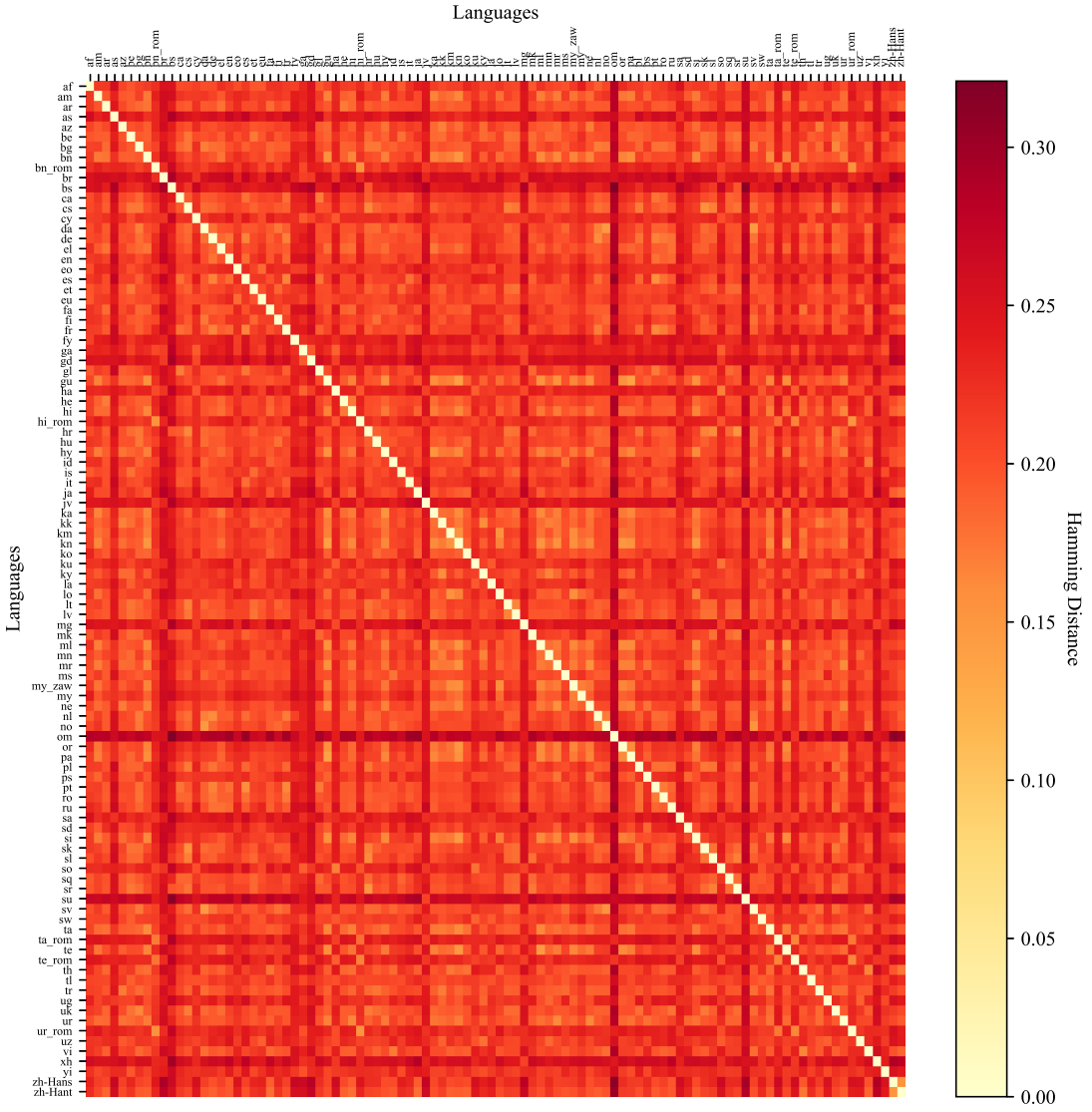

Figure 7: Hamming distance between language subnetworks from regularization-based pruning with `non-shared` setting (Sparsity=50%).

fication with mMiniLMv1[7] (Wang et al., 2020b), a system trained by advanced knowledge distillation techniques. This mMiniLMv1 system has almost the same number of parameters as our 70% sparsity models, and is also evaluated on XNLI. Thus the comparison in Tables 5 and 1 helps to justify that Dynamic Sparsification does not degrade the performance much on different sparsity levels, especially for L_0 regularization with `non-shared` pruning setting.

[7] https://github.com/microsoft/unilm/tree/master/minilm